UNIVERSITY OF PISA

Bayesian Deep Learning for Graphs

Federico Errica

SUPERVISORS
Prof. Davide Bacciu
Prof. Alessio Micheli

A thesis submitted in fulfillment for the degree of Doctor of Philosophy

in the
Department of Computer Science

February 2022

"O frati", dissi "che per cento milia
perigli siete giunti a l'occidente,
a questa tanto picciola vigilia

d'i nostri sensi ch'è del rimanente,
non vogliate negar l'esperienza,
di retro al sol, del mondo sanza gente.

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza".

Li miei compagni fec'io sì aguti,
con questa orazion picciola, al cammino,
che a pena poscia li avrei ritenuti;

e volta nostra poppa nel mattino,
de' remi facemmo ali al folle volo,
sempre acquistando dal lato mancino.

Dante Alighieri - Divina Commedia - Inferno, Canto XXVI

Acknowledgements

Words could never begin to describe how grateful I am to my supervisors, Alessio Micheli and Davide Bacciu. Throughout this PhD I feel I have grown in so many personal and professional ways thanks to you, and if I have ever learned something it was because of the time you took to share your knowledge with me, to provide support when I doubted myself, and to answer my countless emails at ungodly hours. You are the reason why I chose to pursue a career in research, and in these three years, despite the pandemic, you have never made me feel alone. The only pressure I had was the one I put on myself, because you have always trusted me. Today, all these memories bring me joy, purpose, and strength.

Alessio, Davide, you are the best example of men of science I could have asked for, and I will always be indebted to you for that. I hope that in these few years you perceived the profound respect I have for you, just as I hope to have been a dedicated student, a reliable colleague, and maybe even a good friend. In the years to come, and in every scientific work, I shall apply everything you taught me at the best of my abilities.

I wish to thank the international reviewers of this thesis, Mark Coates and Shirui Pan, for their valuable comments and constructive criticism, which made me question what I thought I already knew and, therefore, gave me yet another opportunity to learn from their distinguished scientific expertise. Many thanks to my internal committee members, Luca Oneto and Roberto Grossi, for the invaluable suggestions during these years, as well as to the PhD coordinators Paolo Ferragina and Antonio Brogi for their intense commitment.

During this PhD I had the chance to work side-by-side with wonderful colleagues. I would like to start by thanking Marco, with whom I shared the immense pleasure (and hardship!) of working at all hours of the day. You always reminded me that good science does not depend on the venue, and I will never forget your calm in the most desperate of deadlines. Without you, my PhD would have been half as enjoyable as it was.

A warm thank you goes to Daniele, for our endless discussions on how the hell Bayesian methods work. I am forever grateful for the time you spent teaching me what you knew and for the precious tea breaks in the kitchen that always became small machine learning workshops.

To Antonio and Andrea, thank you for your enthusiasm while working together at the intersection of our research fields, and for our beer-based evenings at the Orzo Bruno. I will not forget you, if you know what I mean.

I also had the dumb luck to get to know Fabrizio, Bora, Guillaume, Vassilis, Ola, Sebastian, Ludovic, Fabio, Lazar, Pushkar, Anton, Cameron, Daniyar and Martin, among others, during my internship at Facebook London. You made me understand how important and enriching it is to make connections and share our stories. There, I also met Pasquale, who gave me the opportunity to virtually visit UCL and whom I sincerely thank for our profound discussions around the similarities and differences of neural link predictors and deep graph networks.

The practical applications shown in this dissertation stem from two distinct collaborations. Allow me to deeply thank Marco, Roberto, and Raffaello of the VARIAMOLS group at the University of Trento, for their patience when I continuously got lost in the mysteries of molecular dynamics and for giving me the first chance to work on real world problems. Let me also thank Giacomo, Francesco, and Fabio at CNR, as we managed to improve robustness in malware classification problems against nasty intra-procedural code obfuscation techniques.

I wish to whole-heartedly thank all the people of the Computational Intelligence and Machine Learning group (we have a wonderful, brand-new website as well as 3D logos!), because each single one of you gave me something to think about at some point. You have been my source of inspiration so many times while having our frequent chats at the sofas. Keep up with the good work and let's meet at the next ESANN!
It was an honor to be part of this amazing group.

While the kitchen was still open, I had the pleasure of sharing refreshing moments with professors, students, and post-docs alike. A word of acknowledgment also goes to them.

I have to say these years have been tough under many aspects, especially because of the pandemic. But having all of you around, physically or virtually, made the journey light and worthwhile.

To me, this PhD has been a great privilege.

Ringraziamenti Personali

Tengo a ringraziare dal profondo del cuore i miei genitori, Marina e Giampaolo, per il loro amore, fiducia e supporto incondizionati, indispensabili per superare tutte le difficoltà della vita. Mamma, Babbo, spero di diventare la metà delle persone meravigliose che siete. Questa tesi è dedicata a voi.

Sono fermamente convinto che questo lavoro sia anche il frutto dell'affetto profondo di Licia e Piero, che da troppo tempo se ne sono andati ma la cui memoria resta indelebile.

Un ringraziamento particolare va a Martina, che ha sempre creduto in me durante questo percorso “infernale”. Nei momenti più difficili mi hai sempre spronato a dare il massimo, e non smetterò mai di essertene grato.

A Irene, Mattia e Giorgio, compagni di percorso e di ufficio, che più che colleghi sono diventati cari amici, dico grazie per la vostra gentilezza, generosità e bontà d’animo.

In tutti questi anni, in periodi più o meno delicati, ho sempre potuto contare sui “cavalieri del polo”, Giacomo, Iacopo con la I, Jacopo con la J e Nicola, i miei amici di sempre e più intimi confidenti. Un semplice grazie non basterà mai per delle persone importanti come voi, ma a dirla tutta non basterebbero neppure le parole, considerato anche che non sono mai stato mai bravo in queste cose. Siete e resterete sempre il mio porto sicuro.

Ho lasciato l’ultimo spazio di queste poche pagine ai ragazzi di “Info Proposte Alcoliche (I.P.A.)”, Daniele, Lorenzo, Marco, Matteo, Michele, Thomas e Tommaso, i miei compagni di università. Con voi ho condiviso i momenti più belli e fuori di testa della mia decade pisana. I ricordi si susseguono incessantemente, e ogni volta mi ci scappa una risata. Conoscervi è stata una delle più grandi fortune della mia vita.

Abstract

The adaptive processing of structured data is a long-standing research topic in machine learning that investigates how to automatically learn a mapping from a structured input to outputs of various nature. Recently, there has been an increasing interest in the adaptive processing of graphs, which led to the development of different neural network-based methodologies. In this thesis, we take a different route and develop a Bayesian Deep Learning framework for the adaptive processing of graphs. The dissertation begins with a review of the foundational principles over which most of the methods in the field are built, and the discussion is complemented with a thorough study on graph classification reproducibility issues. We then proceed to bridge the basic ideas of deep learning for graphs with the Bayesian world, by building our deep architectures in an incremental fashion. The theoretical framework allows us to consider graphs with both discrete and continuous edge features, and it produces unsupervised embeddings rich enough to reach the state of the art on a number of classification tasks. We later discover that our approach is also amenable to a Bayesian nonparametric extension, which automatizes the choice of almost all models' hyper-parameters. Real-world applications are incorporated into the discussion to demonstrate the efficacy of deep learning for graphs. The first one concerns the prediction of information-theoretic quantities useful in molecular simulations, a problem tackled with supervised neural models for graphs. After that, we exploit our Bayesian models to solve a malware-classification task in such a way that the prediction is robust to intra-procedural code obfuscation techniques. We conclude the dissertation with our attempt to blend the best of the neural and Bayesian worlds together. The resulting hybrid model is able to predict multimodal distributions conditioned on input graphs, with the consequent ability to model stochasticity and uncertainty better than most works in the literature. Overall, we aim to provide a Bayesian perspective into the articulated research field of deep learning for graphs.

Contents

Abbreviations

AI	Artificial Intelligence
a.k.a.	Also Known As
BA	Barabási-Albert
BNP	Bayesian Nonparametric
CDE	Conditional Density Estimation
c.d.f.	Cumulative Density Function
CE	Cross-Entropy
CG	Call Graph
CGMM	Contextual Graph Markov Model
CPU	Central Processing Unit
CRP	Chinese Restaurant Process
CRF	Chinese Restaurant Franchise
CV	Cross Validation
DAG	Directed Acyclic Graph
DBGN	Deep Bayesian Graph Network
DGI	Deep Graph Infomax
DGGN	Deep Generative Graph Network
DGN	Deep Graph Network
DNGN	Deep Neural Graph Network
DOAG	Directed Ordered Acyclic Graph
DPAG	Directed Positional Acyclic Graph
DP	Dirichlet Process
e.g.	Exempli Gratia
ECC	Edge Conditioned Convolution
E-CGMM	Extended Contextual Graph Markov Model

EM	Expectation Maximization
ER	Erdős-Rényi
GAE	Graph Auto-Encoder
GCN	Graph Convolutional Network
GG-NN	Gated Graph Neural Network
GIN	Graph Isomorphism Network
GMDN	Graph Mixture Density Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GraphESN	Graph Echo State Network
GRL	Graph Representation Learning
HDP	Hierarchical Dirichlet Process
HMM	Hidden Markov Model
iCGMM	Infinite Contextual Graph Markov Model
i.e.	Id Est
i.i.d.	Independent and Identically Distributed
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MCMC	Markov Chain Monte Carlo
MDN	Mixture Density Network
MLP	Multi-Layer Perceptron
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MPNN	Message Passing Neural Network
MSE	Mean Squared Error
NLP	Natural Language Processing
p.d.f.	Probability Density Function
p.m.f.	Probability Mass Function
ReLU	Rectifier Linear Unit
R-GCN	Relational Graph Convolutional Network
RNN	Recurrent Neural Network
SD	Structured Data
SDL	Structured-Data Learning

SGD	Stochastic Gradient Descent
SIR	Susceptible-Infectious-Recovered
SOM	Self-Organizing Map
SP	Switching Parent
SVM	Support Vector Machine
WL	Weisfeiler-Lehman
w.l.o.g.	Without Loss of Generality
w.r.t.	With Respect To

This thesis is dedicated to my parents
and to Licia and Piero, wherever you are.

Chapter 1

Introduction

Ahi quanto a dir qual era è cosa dura
esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!

Tant'è amara che poco è più morte;
ma per trattar del ben ch'i' vi trovai,
dirò de l'altre cose ch'i' v'ho scorte.

Inferno - Canto I

1.1 Motivations

Abstraction and compositionality are the indispensable principles that we, as humans, avail ourselves of in order to organize, compress, and comprehend the immense amount of information perceived at every instant of our lives. In its broader sense, abstraction is the process of simplifying some aspects of a complex system that are unnecessary to our original purpose. It grants us the ability to find similar patterns and connections between relatively distant ideas or, for instance, research fields. Compositionality, on the other hand, is the tendency to design and understand systems as made of smaller but entangled sub-components. That is to say, most dynamics of the real world are arguably best modeled in relational terms, by considering entities that interact with each other: bees organize themselves in a hierarchy of roles, each of which is crucial for the survival of the colony; in classical physics, the movement of planets can be explained by their mutual interactions through gravity; in chemistry, the disposition of atoms in space, together with their chemical bonds, contribute to the characterization of the properties of the molecule under consideration.

In Computer Science, a data structure is a collection of values that adheres to the principles of abstraction and compositionality and helps us to efficiently organize the information. Depending on our needs different structures are feasible, such as sequences and trees, but hereinafter we will be concerned with the notion of graphs. A graph is a data structure composed of entities freely interacting with each other, so it should come as no surprise that graphs are used in the most disparate problems, from chemistry, physics, and mathematics to linguistics and network science.

Many a time, the combinatorial nature of such structured problems makes it hard to find exact solutions with classical algorithms. In these circumstances, a viable option may reside in machine learning techniques. The adaptive processing of Structured Data (SD) is indeed a longstanding research topic of machine learning [1], whose goal is to learn a mapping from the input structure to the desired output. Over the years, researchers have developed a plethora of specialized methods to process sequences and trees [2–4] relying on their structural regularities, but it was not until recently, after the advent of deep learning, that considerable interest was devoted to the study of adaptive methodologies for graphs. Encouraged by the abrupt availability of graph data and new hardware devices, as well as stimulated by the open research challenges ahead, researchers began exploring the many facets of what is now called Deep Learning for Graphs. This auspicious research direction is characterized by a local and iterative processing of the structured datum, which favors efficiency over combinatorial complexities and is functional to the spreading of information through the graph's entities. The approach also enables automatic features' extraction to solve a task with no human intervention.

Recent research on deep learning for graphs has been very prolific and intense, especially as regards neural networks. The reasons are fairly straightforward: neural networks are incredibly flexible, they can be implemented on hardware accelerators, and we know how to propagate the error signal through very deep architectures. At the same time, such productivity has come at the price of a certain forgetfulness, if not lack of appropriate referencing, of pioneering and consolidated methodologies. Troubling trends on the reproducibility of experiments and the robustness of evaluation protocols immediately followed, generating confusion and ambiguities across the whole literature.

In addition, it could be argued that the Bayesian research direction for deep learning on graphs, i.e., statistical methods modeling the probability distribution of graphs, has been abundantly overlooked in spite of the advantages that the probabilistic approaches usually bring to the table, e.g., expressing causal relationships in the data, incorporating prior information in the process, modeling uncertainty, and building unsupervised embeddings from the posterior distribution. Perhaps, part of the general hesitation is caused by the difficulty of defining deep and end-to-end trainable architectures.

1.2 Objectives

Starting from a unified, objective, and high-level overview of deep learning for graphs, the main ambition of this thesis is to develop a fully probabilistic framework that embraces the most distinctive traits of the field in a Bayesian context. The cross-pollination of ideas between the neural and Bayesian worlds will naturally emerge throughout the manuscript, for the sake of a mathematical formalization rooted in simplicity, efficiency, and empirical efficacy. To move towards our goal, we will first have to understand and review the basic principles that guide the development of most deep learning architectures for graphs. Similarly, we shall attempt at mitigating the reproducibility issues that would make our empirical analyses inconsistent with other works in the literature. With solid ground below our feet, we will then devise deep, unsupervised, and probabilistic models for graphs, called Deep Bayesian Graph Networks, that approximate the data distribution through latent factors. As a by-product of the knowledge gained, we shall additionally investigate how to deal with graph-related uncertainty by mixing neural and probabilistic components, therefore concluding this dissertation with both worlds working in close liaison.

1.3 Contributions

In view of the objectives outlined above, our main contributions can be ascribed to an introductory review of the building blocks that are peculiar to deep learning for graphs, accompanied by a rigorous and standardized evaluation that will allow the theoretical and practical analysis of probabilistic and hybrid models. We complement the discussion with some examples of applications highlighting the advantages of the adaptive processing of structured data.

Unified Review [1] The analysis of a large body of literature, alongside the foundational works of the field, revealed that there exist elemental principles that govern how structured information is usually processed. We believed that the creation of a high-level description of such basic concepts, rather than the systematic analysis of the recent advances, would have benefited both beginners and experts. In this sense, the discourse adopts a top-down organization, in which details are presented only after the key notions have been given. Additionally, we provide a uniform mathematical notation under which different models are compared, to show how sometimes there are subtle but meaningful technical differences in the definition of the main operations. Finally, we systematically

organize a consistent number of works according to their major characteristics, e.g., how they propagate information, the choice of the number of layers, and their nature.

Fair and Robust Empirical Re-evaluation [5] As anticipated, the large stream of recent works has caused severe issues in reproducibility and standardization of the experimental settings. To alleviate this, we propose a robust re-evaluation of various models across several graph classification benchmarks. Starting from a report of the specific issues of each paper under examination, we proceeded to run more than 47000 experiments to fairly compare all models under the same controlled environment. The incorporation of a structure-agnostic baseline in the process led to the discovery that, in some cases, said baseline performs better than most of these deep learning models for graphs.

Basic Deep Probabilistic Framework for Graphs [6, 7] The first methodological contribution of the thesis is the Contextual Graph Markov Model, our attempt at building a fully probabilistic framework for deep learning on graphs. Borrowing ideas from pioneering works, the construction of the deep architecture is incremental, with each layer being trained after another, and the embeddings generation is completely unsupervised. We provide a probabilistic implementation of the neighborhood aggregation mechanisms that operate under the hood, as well as closed-form update equations that guarantee convergence to a local minima of the likelihood landscape. We empirical evaluate our approach on classification benchmarks, and we discover that the unsupervised construction of representations for the graph and its individual entities is surprisingly rich, with subsequent classification performances that are quite close to the state of the art. Furthermore, we analyze the behavior of the model across different layers, showing that depth is of paramount importance to achieve a better generalization.

Architectural Extension of the Framework [8] The Contextual Graph Markov Model can deal with discrete edge information, but as soon as we have more articulated edge features it becomes tricky how to incorporate them into the mathematical formulation. Instead of resorting to hand-crafted discretization techniques, we choose to learn a discretization mapping via an architectural extension of the model. In particular, an additional Bayesian network captures the latent discrete factors responsible for the generation of edge features, and such factors are then incorporated into the original model at the next layer of the architecture. We empirically show that this mechanism is able not only to improve performances over basic edge discretization techniques, but it also boosts classification accuracy whenever edge features are not available.

Bridging Graph Learning and Bayesian Nonparametrics The automatic selection of hyper-parameters is an intriguing research topic that finds elegant and mathematically sound solutions in the Bayesian nonparametric literature. These methods support the selection of the "right" number of latent factors, or clusters, to use in a Bayesian network. For this reason, we apply a Bayesian nonparametric treatment to each layer of the Contextual Graph Markov Model, motivated by the need to automatize as much as possible the choice of its most important hyper-parameter. We also develop a faster but approximated version of the alghoritm that scales to larger graphs, without losing predictive accuracy on the empirical classification tasks considered. Our method, born from the cross-fertilization of ideas belonging to relatively distant fields, reduces by more than 90% the size of the unsupervised graph embeddings, thus saving a great amount of computational resources for the supervised classifier built on top of said embeddings.

Hybrid Approach to Uncertainty Modeling [9] The neural and probabilistic techniques for graph learning undoubtedly have complementary advantages, namely the flexibility of neural networks and the ability of Bayesian networks to naturally handle uncertainty via probability distributions. We realized that some problems, for instance the prediction of stochastic epidemic outcomes in a social network, could not be handled by the current models in the literature. For this reason, we developed the Graph Mixture Density Network, a fairly extensible framework to output multimodal distributions conditioned on input graphs. We provide evidence that previous deep learning approaches for graphs produce unsatisfactory results in the aforementioned contexts, whereas our proposal can express its uncertainty about the plausible continuous value(s) to predict, adding a degree of trustworthiness to the process.

Applications [10, 11] Throughout the manuscript, we take advantage of two practical real-world problems to support our claims about the importance of the methodologies discussed. We will present an application of deep learning for graphs to the field of molecular biosciences, where the goal is to approximate the prediction of a molecule's information-theoretic quantity in a fraction of the time required by the original algorithm. If successful, the learned model would enable a quasi-exhaustive exploration of the output space, due to the combinatorial nature of the problem.
The other application concerns malware classification of software that is subject of obfuscation techniques, in particular those that do not change the topology of the associated "call graph". We will show that our probabilistic models are able to perform very well on a classification task where a structure-agnostic baseline dramatically fails, and such models are also robust to those software obfuscations.

1.4 Thesis' Outline

The thesis is organized into 5 more chapters.

In Chapter 2, we first review the basic definitions of probability, Bayesian learning, and the models we will take inspiration throughout the rest of this work. Then, we will talk about the formal definition of graphs, thus initiating the reader to the most used mathematical notation. Finally, we will shortly summarized related approaches for the adaptive processing of graphs that do not directly belong to deep learning.

In Chapter 3, we introduce the basic principles of machine learning for graphs, regardless of the nature of the models, let them be neural, probabilistic, or hybrid. This broad overview is integrated by an empirical fair comparison of models under the same graph classification settings, in the attempt to partially take back control of the situation, made unstable by the recent wave of (re-)discovery. We conclude the chapter with an application from the field of molecular biosciences.

In Chapter 4, we present the main methodological contributions of this thesis, which fall under the name of Deep Bayesian Graph Networks. The exposition is organized in such a way that new techniques can be seen as extensions of previous ones, and many parallelism are made with the basic notions of Chapter 3. For each of the models presented, we will show variegated empirical analyses in support of the benchmark results. At the end of the chapter, we apply the developed models to a real-world malware classification task.

In Chapter 5, we take the best of the neural and probabilistic worlds and design a hybrid model, called Graph Mixture Density Network, to output multimodal distributions conditioned on arbitrary input graphs. The empirical evaluation on synthetic random graphs and real-world chemical tasks is meant to show that, for some problems, the "standard" approach to deep learning for graphs fails at producing the correct output.

In Chapter 6, we add summarizing thoughts to our dissertation, discussing open problems and future research directions.

1.5 Origin of the Chapters

Most of the results dispensed in this thesis have already been presented at conferences and/or published at journals. The list below is the outcome of hard but much pleasant work with a number of co-authors, who gave us the opportunity to collaborate at the cross-road of different research fields.

Chapter 3

- Sections 3.1 to 3.2:

  Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 9 2020

- Section 3.3:

  Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations (ICLR)*, 2020

- Section 3.4:

  Federico Errica, Marco Giulini, Davide Bacciu, Roberto Menichetti, Alessio Micheli, and Raffaello Potestio. A deep graph network–enhanced sampling approach to efficiently explore the space of reduced representations of proteins. *Frontiers in Molecular Biosciences*, 8:136–150, 2021

Chapter 4

- Section 4.1:

  Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A deep and generative approach to graph processing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 294–303, 2018

  Davide Bacciu, Federico Errica, and Alessio Micheli. Probabilistic learning on graphs via contextual architectures. *Journal of Machine Learning Research*, 21 (134):1–39, 2020

- Section 4.2:

  Federico Errica Daniele Atzeni, Davide Bacciu and Alessio Micheli. Modeling edge features with deep bayesian graph networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021

- Section 4.3 reports unpublished work, currently under review.

- Section 4.4:

  Federico Errica, Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Alessio Micheli. Robust malware classification via deep graph networks on call graph topologies. In *Proceedings of the 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2021

Chapter 5

- All Sections:

  Federico Errica, Davide Bacciu, and Alessio Micheli. Graph mixture density networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 3025–3035, 2021

Chapter 2

Preliminaries

E poi che la sua mano a la mia puose
con lieto volto, ond'io mi confortai,
mi mise dentro a le segrete cose.

Inferno - Canto III

In this chapter, we shall delineate basic definitions and techniques that will be used throughout the rest of the manuscript. In doing so, it is assumed that the reader is familiar with linear algebra and the fundamental machine learning concepts such as supervised and unsupervised learning, multi-layer perceptrons, hidden units, and activation functions, to name a few. We shall begin with a probability refresher and an introduction to some probabilistic modeling techniques. Special attention is devoted to mixture models, as the probabilistic models developed in this thesis inherit many of their characteristics. Then, we move to more advanced topics, such as ensity networks for flat data, which borrow ideas from both neural and probabilistic worlds, and Bayesian nonparametric mixture models, where the model's complexity grows with the data.
We shall continue with a much needed discourse about the multifaceted nature of graphs, introducing standard definitions and mentioning the challenges that machine learning models have to face when handling this kind of structured data: these include the presence of cycles and the absence of a known ordering of the graph entities. Also, we describe particular instances of graphs with a more rigid structure, for which learning models are known and well-studied. Then, we discuss random graphs and the process to generate them: we will necessitate synthetic datasets to carry out some of our experiments. To conclude the chapter, we provide a brief summary of different research directions that are complementary to the topics presented in this manuscript.

2.1 Probabilistic modeling

We now review the principles of probability theory and some probabilistic modeling techniques that are especially relevant for this thesis. The reader can refer to [12, 13] for a complete treatment of these topics.

2.1.1 Probability Refresher

2.1.1.1 Basic Definitions

Probability theory provides us with the mathematical tools to rigorously formalize our intuition of uncertainty and randomness [14]. To this aim, we first introduce the set of all possible outcomes of an experiment with the symbol Ω (the *sample space*) and the set of events of interest that may occur as $\mathscr{A} \subseteq \mathscr{P}(\Omega)$, where $\mathscr{P}(\cdot)$ denotes the powerset operator; in particular, we require $\mathscr{A}$ to be a σ-algebra (or σ-field).

Definition 2.1 (**σ-algebra**)**.** Let Ω be the set of possible outcomes and consider the set of events $\mathscr{A} \subseteq \mathscr{P}(\Omega)$. Then, $\mathscr{A}$ is a σ-algebra if the following holds:

1. $\emptyset \in \mathscr{A}$ (accounting for the impossible event)
2. $\forall A \in \mathscr{A} \implies (\Omega / A) \in \mathscr{A}$ (closure under complement)
3. $\forall \{A_n\}_{n \in \mathbb{N}} \subseteq \mathscr{A} \implies \bigcup_{i \in \mathbb{N}} A_i \in \mathscr{A}$ (closure under countable union).

For instance, if we wanted to toss a coin, we would have a sample space $\Omega = \{\text{head,tail}\}$ and $\mathscr{A} = \mathscr{P}(\Omega) = \{\{\emptyset\}, \{\text{head}\}, \{\text{tail}\}, \{\text{head, tail}\}\}$. Instead, if we had considered an experiment made of two coin tosses, a single outcome could have been $\omega = \{\text{head, head}\} \in \Omega$.

In order to assign a number to a subset of events, the reason for which will become clear in a moment, we need the notion of a measure over sets.

Definition 2.2 (**Measure**)**.** Let $\mathscr{A}$ be a σ-algebra defined over Ω. A function $f : \mathscr{A} \to [0, +\infty]$ is a measure on $(\Omega, \mathscr{A})$ whenever:

1. $\forall A \in \mathscr{A} \implies f(A) \geq 0$ (non-negativity)
2. $f(\emptyset) = 0$ (null empty set)
3. For all countable collections $\{A_n\}_{n \in \mathbb{N}} \in \mathscr{A}$ of disjoint sets it holds $f(\bigcup_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} f(A_i)$ (σ-additivity).

Therefore, a measure is a function that takes a set of elementary outcomes and outputs a real number. Intuitively, this number may be regarded as the "size" of that set. Moreover, the pair $(\Omega, \mathscr{A})$ is called a measurable (or Borel) space. At this point, we explit these concepts to define what a probability is.

Definition 2.3 (**Probability**)**.** A probability is a measure P on $(\Omega, \mathscr{A})$ where $P(\Omega) = 1$. Moreover, the tuple $(\Omega, \mathscr{A}, P)$ is called a **probability space**.

Continuing with the coin flip example, we can imagine a fair coin toss experiment where the probabilities are: $P(\{\emptyset\}) = 0$, $P(\{\text{head}\}) = \frac{1}{2}$, $P(\{\text{tail}\}) = \frac{1}{2}$, and $P(\{\text{head,tail}\}) = 1$.

Throughout the following sections and chapters, we will frequently encounter the notion of random variable. Informally, a random variable is a variable whose values are associated with a probability of occurrence.

Definition 2.4 (**Random Variable**)**.** Given a probability space $(\Omega, \mathscr{A}, P)$, a random variable is a measurable function $X : \Omega \to E$ s.t. $\{\omega \in \Omega \mid X(\omega) \in E\} \in \mathscr{A}$.

A random variable X can be **discrete** or **continuous**, depending on the nature of its image E. To model the coin toss experiment, we can construct a discrete random variable X such that $E = \{0, 1\}$, $X(\text{head}) = 0$, $X(\text{tail}) = 1$, and use the notation $P(X = \text{head}) = \frac{1}{2}$ and $P(X = \text{tail}) = \frac{1}{2}$ to convey the same information as above. In general, different outcomes may be assigned the same discrete value in E. For example, if we toss a six-face dice two times and compute the sum of the numbers on the faces, we can have 36 possible outcomes (the size of the sample space) but only 11 different results (the size of the discrete set E).

Closely related to the concept of random variable is the notion of stochastic (or random) process. A stochastic process allows to mathematically model the behaviour of complex systems by considering families of random variables indexed by an appropriate set.

Definition 2.5 (**Stochastic Process**)**.** Given a probability space $(\Omega, \mathscr{A}, P)$ and a set T, a stochastic process refers to a family of random variables $\{X_t\}_{t \in T}$. The values that each random variable X_t can take are called *states*.

Any random variable is completely characterised by its Cumulative Distribution Function (or probability law), which describes the probability that the value assumed by the random variable is smaller than a given parameter.

Definition 2.6 (**Cumulative Distribution Function** (c.d.f.))**.** The cumulative distribution function F of a random variable X is defined as $F(X \leq x) = P(\{\omega \in \Omega \mid X(\omega) \leq x\})$, which is abbreviated as $P(X \leq x)$.

For discrete random variables, we can also consider the tabular **probability mass function** (p.m.f.) $p_X(x) = P(X = x)$, whereas in the case of continuous random variables we define the **probability density function** (p.d.f.) as $f_X(x) = \frac{d}{dx}F_X(x)$. When clear from the context, we shall use the notation $P(X = x)$ (or $P(x)$ for short) in place of $p_X(x)$ or $f_X(x)$. Also, the **support** of a probability distribution is, the set of values for which it returns a non-zero probability of occurrence.

Many's the time we are interested in the probability of more than one event occurring. We formalize this case with a set of random variables, each capturing the occurrence of a specific event. Accordingly, we call the **joint probability distribution** $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ the multivariate distribution that represents this particular stochastic process. In case a set of n random variables is said to be **mutually independent**, the joint probability decomposes into the product of the single terms, i.e., $\prod_{i=1}^{n} P(X_i = x_i)$. This is because knowing about a given X_i does not change our uncertainty about another random variable and viceversa. Moreover, when a set of random variables is mutually independent and each variable has the same probability distribution, the variables are said to be **independently and identically distributed** (i.i.d.). On the contrary, when the realization of an event $Y = y$ has an effect on the occurrence of other random variables, we talk about **conditional probabilities**, denoted by $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n \mid Y = y)$. Like mutual independence, random variables are **conditionally independent** if it holds $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n \mid Y = y) = \prod_{i=1}^{n} P(X_i = x_i | Y = y)$.
We are now ready to define the crucial rules that will be extensively used in the following.

Definition 2.7 (**Sum Rule**)**.** Given a random variable X, the sum of probabilities over all its values must sum to 1, i.e., $\sum_x P(X = x) = 1$.

Definition 2.8 (**Product Rule**, a.k.a. Chain Rule)**.** The joint distribution of n variables can always be rewritten as

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | \bigcap_{j=1}^{i-1} X_j = x_j).$$

Definition 2.9 (**Marginalization**)**.** Combining the sum and product rules, and given two random variables X and Y w.l.o.g., we can obtain the marginal probabilities of X and Y as follows:

$$P(X = x) = \sum_y P(X = x, Y = y)$$
$$P(Y = y) = \sum_x P(Y = y, X = x)$$

Oftentimes, we are also interested in computing the average value that X assumes, which is called the expected value of X.

Definition 2.10 (**Expected Value**)**.** Given a random variable X defined over a probability space $(\Omega, \mathscr{A}, P)$, the expected value of X is defined as

$$\mathbb{E}[X] = \int_{\omega \in \Omega} X(\omega) dP(\omega),$$

where the Lebesgue integral is taken with respect to the measure P. For a discrete random variable X with a finite number of attainable states and a known p.m.f., the above equation simplifies to:

$$\mathbb{E}[X] = \sum_{x} xP(X = x).$$

Instead, for continuous random variables whose distribution P has a p.d.f., we can write

$$\mathbb{E}[X] = \int_{\mathbb{R}} x f_X(x) dx.$$

In the remainder of this thesis, whenever we want to make explicit the distribution over which we are computing the expectation of a certain variable, we will use the notation $\mathbb{E}_{x \sim P}[X]$. In other words, we compute the expectation with respect to the values of x sampled from the distribution P.

Finally, we introduce the notion of (discrete-time) Markov Chain to later discuss about inference algorithms.

Definition 2.11 (**Markov Chain**)**.** Let $(\Omega, \mathscr{A}, P)$ be a probability space, and consider a stochastic process $\{X_t\}_{t \in T}$ where T is a totally ordered set. Then, $\{X_t\}_{t \in T}$ is a Markov Chain whenever, for all sequences $t_0 < \cdots < t_n < t_{n+1}$ and for all states $x_0, \ldots, x_n, x_{n+1}$ the 1-st order **Markov property** holds:

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, \ldots, X_0 = x_0) = P(X_{n+1} = x_{n+1} \mid X_n = x_n).$$

We can easily generalize this definition to the **k-th order** Markov property

$$P(x_{n+1} \mid X_n = x_n, \ldots, X_0 = x_0) = P(X_{n+1} = x_{n+1} \mid X_n = x_n, \ldots, X_{n-k} = x_{n-k}),$$

giving rise to a **higher-order** Markov Chain.

When a process satisfies the Markov property, we say it is *Markovian*.

2.1.1.2 Useful Distributions

As most of this work will be devoted to the development of deep and probabilistic models for graphs, it is useful to briefly introduce the distributions that model discrete and continuous data features.

Categorical Distribution. The categorical distribution is a discrete probability distribution working on a finite set of values of size C. It is often used by discrete random variables that model C different possible outcomes, and it can conveniently be represented as a real vector of size C whose entries sum to 1. The parameters of the distributions are given by the C different probabilities p_i that constitute said vector. Simply put, the p.m.f. of this distribution writes

$$p(X = i) = p_i \quad \forall i \in \{1, \dots, C\}.$$

Figure 2.1 depicts the p.m.f. and c.d.f. of a categorical distribution.

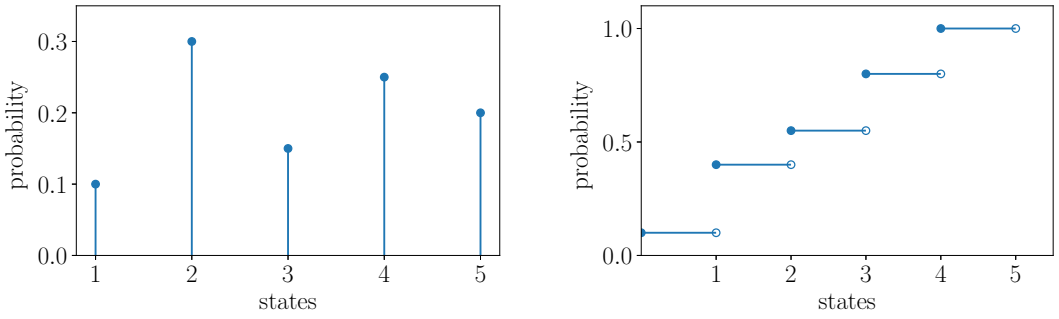


FIGURE 2.1: We present a possible realization of a categorical distribution through its probability mass function (left) and cumulative distribution function (right). Empty dots symbolize non-smooth jumps to the next probability value.

Gaussian Distribution. The Gaussian distribution is a continuous probability distribution whose support is $\mathbb{R}$. For a single random variable (**univariate** case), the parameters of the distribution are just the mean value μ and the variance σ^2, and the probability density function is defined as

$$P(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}.$$

Generalizing the distribution to n random variables (**multivariate** case), the parameters that define the distribution are a vector of n means $\boldsymbol{\mu}$ and an $n \times n$ covariance matrix $\boldsymbol{\Sigma}$.

The p.m.f. of a multivariate distribution then becomes a function that takes an input $\boldsymbol{x} \in \mathbb{R}^n$ and returns a probability score:

$$P(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})},$$

where $det(\cdot)$ is the determinant of a matrix. A multivariate Gaussian distribution is said to be **isotropic** when the covariance matrix is diagonal, meaning the random variables under consideration are independent. In this case, the total number of parameters becomes $2n$ rather than $n + n^2$. Figure 2.2 visualizes instances of the univariate and bivariate Gaussian distributions.

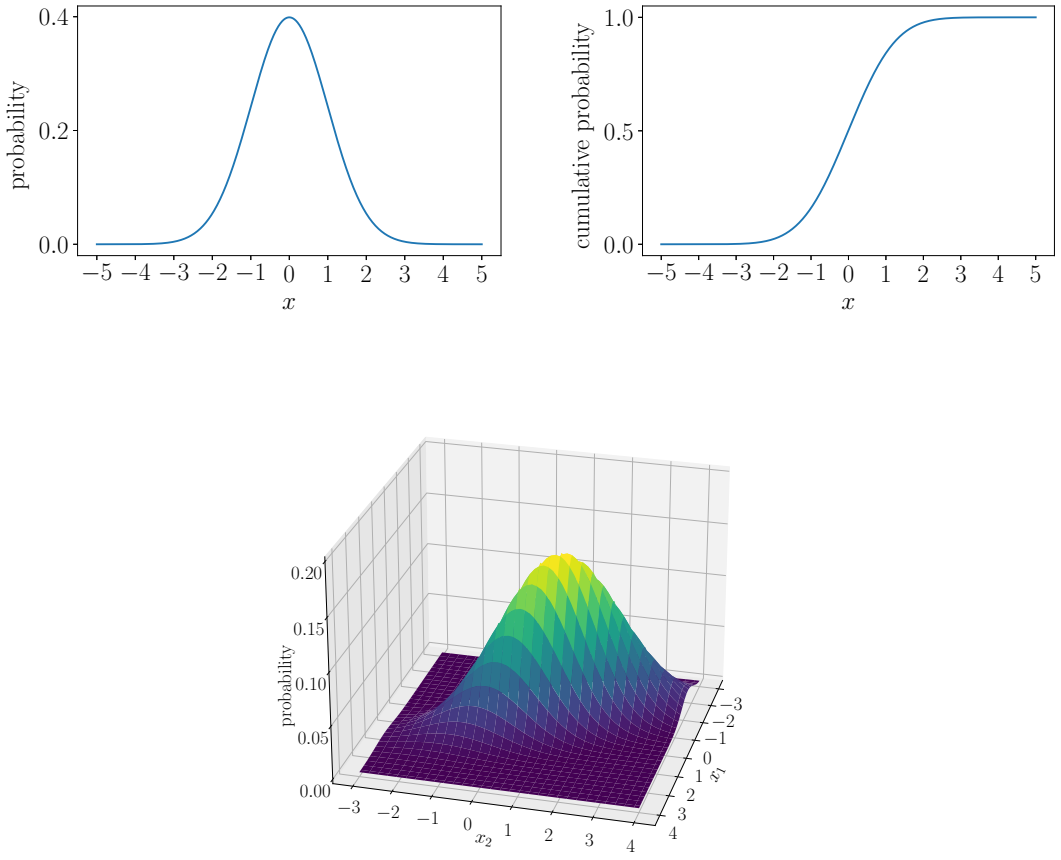


FIGURE 2.2: We depict the probability density function (left) and cumulative distribution function (right) of a univariate Gaussian distribution with mean 0 and variance 1. In addition, we plot the p.d.f. of a bivariate Gaussian with $\boldsymbol{\mu} = [0, 1]$ and $\boldsymbol{\Sigma} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$.

Binomial Distribution. The binomial distribution is another discrete probability distribution that accounts for the number of successes in a sequence of n experiments, each

of which has a probability p of success and $1-p$ of failure. Its support is the set $\{0, \dots, n\}$ and the p.m.f. is computed as

$$P(x \mid n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$
$$\text{where } \binom{n}{x} = \frac{n!}{k!(n-k)!}.$$

Usually, the single experiment is called a Bernoulli trial, whereas the entire sequence of outcomes is a Bernoulli process. The reference to Bernoulli comes from the fact that, when $n = 1$, the distribution simplifies to a Bernoulli distribution (not shown here). We conclude by showing the p.m.f. and c.d.f. of different binomial distributions in Figure 2.3.

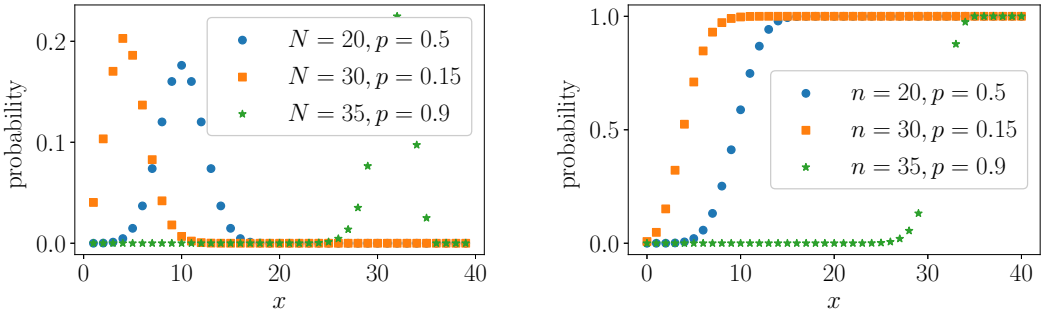


FIGURE 2.3: We plot different binomial distributions (left) and their c.d.f. (right) for different choices of n and p.

Dirichlet Distribution. The Dirichlet distribution is a multivariate continuous distribution parametrized by a fixed vector $\boldsymbol{\alpha}$ of C real values called concentration parameters. It can be seen as a distribution over distributions, because the sampling process outputs a vector $\boldsymbol{p}$, belonging to the standard $C-1$ simplex, which might be used to parametrize a categorical distribution. The p.d.f. of the Dirichlet distribution $D(\boldsymbol{\alpha})$ is defined as

$$P(\boldsymbol{x}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{C} x_i^{\alpha_i - 1}$$
$$\text{where } B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{C} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{C} \alpha_i)}, \Gamma(\alpha_i) = (\alpha_i - 1)!.$$

The Dirichlet distribution is particularly important in Bayesian statistics, being a **conjugate** distribution for discrete distributions such as the categorical. If a distribution P is the conjugate of a distribution Q, it means that multiplying the p.d.f. of P and Q will result in a distribution whose p.d.f. belongs to the same family of distributions as P. This greatly simplifies the math and provides closed-form solutions in Bayesian

learning, where the Dirichlet distribution is used to embed **prior knowledge** in the learning framework. Finally, we depict two examples of Dirichlet distributions in Figure 2.4.

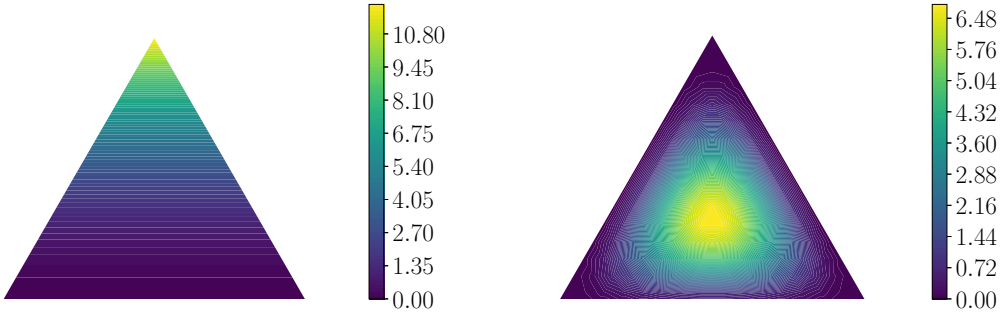


FIGURE 2.4: Two Dirichlet distributions with $\boldsymbol{\alpha} = [1, 1, 3]$ (left) and $\boldsymbol{\alpha} = [3, 3, 3]$ (right). Each side of the triangles denotes a different input component $x_i \in [0, 1]$.

Normal-Gamma Distribution. The last distribution we introduce is the Normal-Gamma distribution, a continuous probability distribution that is conjugate of the univariate Gaussian distribution. It has four parameters that act as prior knowledge: μ, the empirical mean of the data; λ, which is inversely proportional to the prior variance of the data (i.e., our belief about the robustness of the empirical mean); a and b, whose ratio $t = \frac{b}{a}$ controls the expected variance of the data captured by the univariate Gaussian. Assuming a Gaussian random variable X and a Gamma random variable T, the p.d.f. of a Normal-Gamma distribution is computed as

$$
\begin{aligned}
&X \sim \mathcal{N}(\mu, 1/(\lambda T)) \\
&T \sim \text{Gamma}(a, b) \text{ whose p.d.f. is } f(t \mid a, b) = \frac{b^a t^{a-1} e^{-bt}}{\Gamma(a)}, \Gamma(a) = (a-1)! \\
&P(x, t|\mu, \lambda, a, b) = \frac{b^a \sqrt{\lambda}}{\Gamma(a)\sqrt{2\pi}} t^{a-\frac{1}{2}} e^{-bt} \exp(-\frac{\lambda t (x-\mu)^2}{2}).
\end{aligned}
$$

2.1.1.3 Learning as an Inference Problem

The goal of a generic machine learning task is to find a suitable choice of the model's parameters in order to optimize a pre-defined objective function. Likewise, in a probabilistic setting where we need to capture the underlying distribution of the data, one chooses a family of parametrized distributions assuming it is flexible enough to mimic the true (unknown) data distribution. Therefore, the learning task can be framed as an **inference** problem in which we adjust our beliefs about the parameters of the selected family of distributions, e.g., a Gaussian or a Categorical. Before we outline the basics of

the probabilistic learning framework adopted in this thesis, we shall introduce the Bayes' Rule, which lies down the foundations of **Bayesian learning**.

Definition 2.12 (**Bayes' Rule**)**.** Given a discrete hypotheses space $\mathcal{H}$ and a set of observations $\mathcal{D}$ (both of which can be modeled as random processes), for each hypothesis $h_i \in \mathcal{H}$ it holds

$$P(h_i \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid h_i)P(h_i)}{P(\mathcal{D})} = \frac{P(\mathcal{D} \mid h_i)P(h_i)}{\sum_j P(\mathcal{D} \mid h_j)P(h_j)}$$

where $P(h)$ is called the **prior** probability of h, $P(\mathcal{D} \mid h)$ is the **likelihood** that the data has been generated by a certain hypothesis (with corresponding parameters $\boldsymbol{\theta}$), and $P(h \mid \mathcal{D})$ is the posterior probability that the hypothesis is correct given the data and out prior beliefs about that hypothesis. Therefore, under the Bayes' rule there is a trade-off between our prior beliefs and the evidence coming from the data.

Given a set of random variables $\boldsymbol{X} = \{X_1, \dots, X_n\}$ called **observations** that formalize our knowledge about "the world", the prediction for a new data point in a Bayesian learning setting can be written as

$$P(\boldsymbol{X} = \boldsymbol{x} \mid \mathcal{D}) = \sum_i P(\boldsymbol{X} = \boldsymbol{x} \mid \mathcal{D}, h_i)P(h_i \mid \mathcal{D}) = \sum_i \underbrace{P(\boldsymbol{X} = \boldsymbol{x} \mid h_i)}_{\substack{\text{hypothesis} \\ \text{prediction}}} \underbrace{P(h_i \mid \mathcal{D})}_{\substack{\text{posterior} \\ \text{weighting}}},$$

where we used the product rule, marginalization, and assumed that $\boldsymbol{X}$ does not depend on the data when hypothesis h_i holds. Unfortunately, very often the space of the hypotheses is infinite, and the exact computation of the posterior distribution of $\boldsymbol{X}$ becomes intractable. To address this practical issue, we can look for the single most likely hypothesis given our data. This is called Maximum A Posteriori (MAP) inference:

$$h_{MAP} = \arg\max_{h \in \mathcal{H}} P(h \mid \mathcal{D}) \overset{\substack{\text{Bayes'} \\ \text{Rule}}}{=} \arg\max_{h \in \mathcal{H}} \frac{P(\mathcal{D} \mid h)P(h)}{P(\mathcal{D})} = \arg\max_{h \in \mathcal{H}} P(\mathcal{D} \mid h)P(h),$$

noting that we ignored the contribution of $P(\mathcal{D})$ in the maximization because constant. When assuming a **uniform** prior distribution over the choice of $\mathcal{H}$, meaning $P(h)$ is the same everywhere, we obtain the **Maximum Likelihood Estimation** (MLE) objective:

$$h_{MLE} = \arg\max_{h \in \mathcal{H}} P(\mathcal{D} \mid h) = \arg\max_{h \in \mathcal{H}} \mathcal{L}(\boldsymbol{\theta}_h \mid \mathcal{D}).$$

Most of the techniques presented in this thesis will be based on the MLE objective, whereas a restricted number of them will adopt MAP inference.

2.1.1.4 Bayesian Networks

When modeling the world with a set of random variables, it is natural to make assumptions about the relationships between those. One way to graphically express the conditional dependencies between such variables is to use a Bayesian network. In this graphical representation, an istance of which is illustrated in Figure 2.5, nodes represent variables and edges convey the conditional independence information. We distinguish between **observed** variables that can always be inspected (i.e., the observations $\mathcal{D}$ contain information about the values of these variables) and **latent** or **hidden** variables whose values have to be inferred from the data. By definition, a Bayesian network allows us to decompose the joint probability distribution of all variables following a straightforward rule:

$$P(X_1 \ldots, X_n) = \prod_i^n P(X_i \mid pa(X_i)),$$

where the term $pa(X_i)$ refers to the set of nodes that have an edge pointing to X_i (the "parents" of X_i). Note that in a Bayesian network a variable cannot depend, directly or indirectly, on itself. In turn, this simplifies the math and enables tractable solutions in many cases.

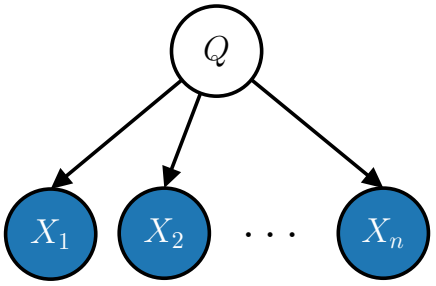


FIGURE 2.5: A Bayesian network with latent (white) and observed (blue) variables.

To graphically represent a set of random variables in a compact way, we adopt the so-called **plate notation** of Figure 2.6, where a plate (the box) symbolises replication of conditional relations and encompassed variables for a number of times indicated by the letter in the corner. Still, when clear from the context, we may drop the box and simply use a subscript to denote the identity of the variables. With slight abuse of notation, we also use white squares to represent hyper-parameters and blue squares for intermediate deterministic computations.

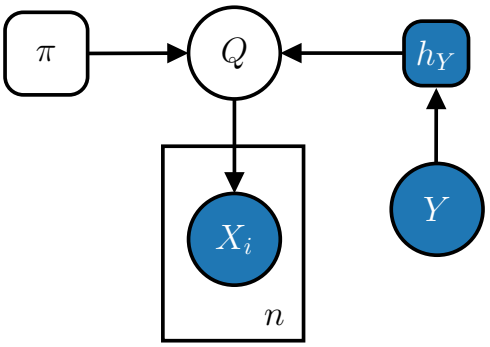


FIGURE 2.6: Plate notation for the Bayesian network of Figure 2.5, with the addition of prior hyper-parameters π and an intermediate node h_Y obtained deterministically from the observed value of Y.

2.1.1.5 The Expectation-Maximization Algorithm

A widely adopted tool to train a probabilistic model with latent variables $\boldsymbol{Z}$ is the Expectation-Maximization (EM) algorithm [15]. The key insight is that, rather than maximizing the "incomplete" likelihood of the observed data $P(\boldsymbol{X} \mid \boldsymbol{\theta})$, we focus on the **complete likelihood** of the data $P(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})$. It is a two-step iterative procedure that maximizes the likelihood of the data, in which the parameters of the model at time step t, namely $\boldsymbol{\theta}^{(t)}$, are updated (M-step) using the **current estimate** for the values of the hidden variables $\boldsymbol{Z}$ (E-step). This resulting objective is simpler to maximize, since it does not involve marginalization over all variables in $\boldsymbol{Z}$. Formally, the EM algorithm involves the following steps:

1. initialize the parameters $\boldsymbol{\theta}^{(1)}$ at random
2. (E-step) compute the expected value of the complete log-likelihood w.r.t. $\boldsymbol{\theta}^{(t)}$

$$Q_{EM}(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}[\log P(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta}^{(t)})]$$

3. (M-step) find the parameters that maximize the previous quantity

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q_{EM}(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$$

4. Repeat steps 2 and 3 until the complete log-likelihood stops increasing.

It can be shown that the quantity computed by the E-step is a lower-bound of the true likelihood [15]. Therefore, the EM algorithm can converge monotonically to a local minimum of the initial objective. In some cases, it is possible to compute the M-step solutions in closed-form, this obtaining the maximum improvement over Q_{EM}. Whenever this is not possible, common optimization algorithms may be used such as Stochastic Gradient

Descent (SGD) [16]; as long as the value of the lower-bound increases, convergence is still guaranteed. In this case, we talk about **Generalized Expectation Maximization** (GEM) [15]. Finally, note that EM can be easily extended to deal with MAP estimation, by adding the contribution of the log-prior to the quantity $E(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$ to maximize.

Knowledge about the values of hidden variables can be modeled via **indicator random variables**. An indicator variable is a binary variable that is 1 when, e.g., another random variable is in a certain state, and 0 otherwise. For instance, given a categorical variable Q with C states, an indicator variable Z_c may be described as follows:

$$Z_c = \begin{cases} 1, & \text{if } Q = c \text{ with probability } p_c \\ 0, & \text{otherwise with probability } (1 - p_c) \end{cases} \qquad \text{where } p_c = P(Q = c).$$

As we will see, indicator variables will be invaluable to define the complete log-likelihood of mixture models.

2.1.1.6 Gibbs Sampling

Whenever we shall encounter a joint probability distribution that is difficult to formalize or sample from but the conditional probability of the individual variables is easier to compute, we will depend on Markov Chain Monte Carlo (MCMC) algorithms. An MCMC algorithm that is often used in Bayesian inference is **Gibbs sampling** [17], which works by sampling the value of a single variable at a time before moving to the next one. By iterating this process over and over, eventually the sampled values should approximate the original joint distribution. Also, when some of the variables are observed, their values are never updated. Given n random variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ from a joint probability distribution $P(X_1 = x_1, \ldots, X_n = x_n)$, we can obtain k samples from $\boldsymbol{X}$ using the Gibbs sampler:

1. Initialize the sample as $\boldsymbol{X}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$
2. When creating the i+1-th sample, for each component j (in order), update its value by sampling from the known conditional probability using the following formula

$$x_j^{(t+1)} \sim P(x_j^{(t+1)} | x_1^{(t+1)}, \ldots, x_{j-1}^{(t+1)}, x_{j+1}^{(t)}, \ldots, x_n^{(t)})$$

3. Repeat the previous step k times in order to obtain k distinct samples.

In general, we assume there is a **burn-in period** of variable length in which samples do not accurately represent the joint distribution, so they are discarded. With Gibbs

sampling, we are able to infer the posterior distribution of the model's parameters conditioned on the data. Still, to do so, we need to assume some family of distributions for the different conditional probabilities; in this sense, a Bayesian network is an excellent companion for Gibbs sampling, as its joint distribution is specified by means of a set of conditional distributions.

2.1.2 Mixture Models

We shall now introduce the concept of mixture of distributions, which will be central to this thesis thanks to its simplicity and flexibility. Let us assume we have a population of samples, i.e., our data, in which there exist C sub-populations we would like to model. In other words, we would like to associate each data point with one of the C sub-populations; this process is usually referred to as **clustering**, and we can represent it with the graphical model of Figure 2.7. In particular, we assume that there is a latent

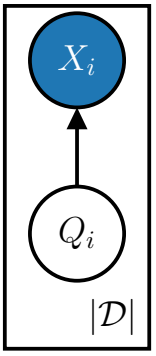


FIGURE 2.7: The probabilistic graphical representation for a mixture model. The latent variable Q assigns each data point to one of C different sub-populations (or **clusters**).

factor, represented by the categorical variable Q with C states, that is responsible for the generation of the data points. Knowledge of the probability distributions of $P(Q)$ and $P(X \mid Q)$ would allow us to generate new samples by first sampling the cluster c from $P(Q)$ and then drawing the data point x from $P(X|Q = c)$. This is a mechanism known as **ancestral sampling**.

Formally, we can model the distribution of our population by introducing the latent variables via marginalization, assuming our samples are independent and identically distributed:

$$\mathcal{L}(\boldsymbol{\theta} \mid \mathcal{D}) = P(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_{i=1}^{|\mathcal{D}|} P(X_i = x_i) = \prod_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{C} P(X_i = x_i \mid Q_i = j, \boldsymbol{\theta}) P(Q_i = j \mid \boldsymbol{\theta}),$$

where $\boldsymbol{\theta}$ are the parameters of the mixture model that we have to learn. From now on, we will refer to $P(X \mid Q)$ as the **emission distribution**.

Usually, we have no guarantees that the data will be organized in C different sub-populations. At the same time, the larger C is the more flexible our approximation will be. Hence, the parameter C, which controls the number of distributions to mix, has to be therefore treated as an hyper-parameter.

To train a mixture model that maximizes the likelihood of our data, let us adopt the EM framework and define the *complete log-likelihood* of the data. The usual way to do this is to first assume we know the assignment of each data point to its cluster. Therefore, let us introduce the indicator variables $\boldsymbol{Z} = \{Z_{11}, \dots, Z_{|\mathcal{D}|C}\}$ where a generic variable Z_{ij} has value 1 when the data point i is assigned to cluster j. We can take advantage of $\boldsymbol{Z}$ to write the complete log-likelihood

$$\begin{aligned}\mathcal{L}_c(\boldsymbol{\theta} \mid \boldsymbol{Z}, \mathcal{D}) &= \prod_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{C} z_{ij} P(X_i = x_i \mid Q_i = j, \boldsymbol{\theta}) P(Q_i = j \mid \boldsymbol{\theta}) \\ &= \prod_{i=1}^{|\mathcal{D}|} \prod_{j=1}^{C} \big(P(X_i = x_i \mid Q_i = j, \boldsymbol{\theta}) P(Q_i = j \mid \boldsymbol{\theta})\big)^{z_{ij}}.\end{aligned}$$

It is possible to obtain the last identity by noting that z_{ij} nullifies the contributions of other clusters. To compute the E-step, we now need to apply the logarithm to the above quantity, obtaining

$$\log \mathcal{L}_c(\boldsymbol{\theta} \mid \boldsymbol{Z}, \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{C} z_{ij} \log \big(P(X_i = x_i \mid Q_i = j, \boldsymbol{\theta}) P(Q_i = j \mid \boldsymbol{\theta})\big).$$

Assuming we have some parameters $\boldsymbol{\theta}^{(t)}$, the quantity to maximize at each EM iteration can be computed as

$$\begin{aligned}&Q_{EM}(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\boldsymbol{Z}|\mathcal{D},\boldsymbol{\theta}^{(t)}}[\log P(\mathcal{D}, \boldsymbol{Z} \mid \boldsymbol{\theta}^{(t)})] = \\ &= \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{C} \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}] \log \big(P(X_i = x_i \mid Q_i = j, \boldsymbol{\theta}) P(Q_i = j \mid \boldsymbol{\theta})\big).\end{aligned}$$

Depending on the nature of the data, the family of the emission distributions $P(X \mid Q)$ will be different. For instance, discrete data may require a categorical emission, whereas continuous data could be modeled by a Gaussian distribution. In the former case, it is possible to show that the optimal emission distribution is given by

$$P^{(t+1)}(X = k|Q = j) = \frac{\sum_i \delta(x_i, k) \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]}{\sum_{j'=1}^{C} \sum_i \delta(x_i, j') \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]},$$

with $\delta(\cdot, \cdot)$ being the Kronecker delta function. In practice, the above term amounts to compute the fraction of points belonging to a certain cluster j that have label k, each

weighted by the expected probability (according to our current parameters) that the point will be in cluster j. This can be seen by noticing that the expected value of an indicator variable is its associated probability.

On the other hand, for a Gaussian emission, the sufficient statistics for the j-th mixture component are

$$\mu_j^{(t+1)} = \frac{\sum_i x_i \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]}{\sum_i \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]},$$
$$\sigma_j^{(t+1)} = \sqrt{\frac{\sum_i \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}](x_i - \mu_j^{(t)})^2}{\sum_i \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]}}.$$

Finally, the prior probability $P(Q)$ is updated as follows:

$$P^{(t+1)}(Q = j) = \frac{\sum_i \mathbb{E}[z_{ij} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]}{\sum_i \sum_{j'=1}^{C} \mathbb{E}[z_{ij'} \mid \mathcal{D}, \boldsymbol{\theta}^{(t)}]}.$$

For a complete treatment of the multivariate Gaussian case and more, the reader is referred to [12, 13]. We conclude this part with an example of a three-component mixture model fitting a population with three well-separated clusters. Figure 2.8 shows the outcome of fitting a Gaussian mixture model via maximum likelihood estimation. We observe how the three components adapt their mean (the diamond symbol) and covariance matrices (the ellipses) to capture the original data distribution. These concepts will be extended later on in this thesis to accommodate the complex domain of graphs.

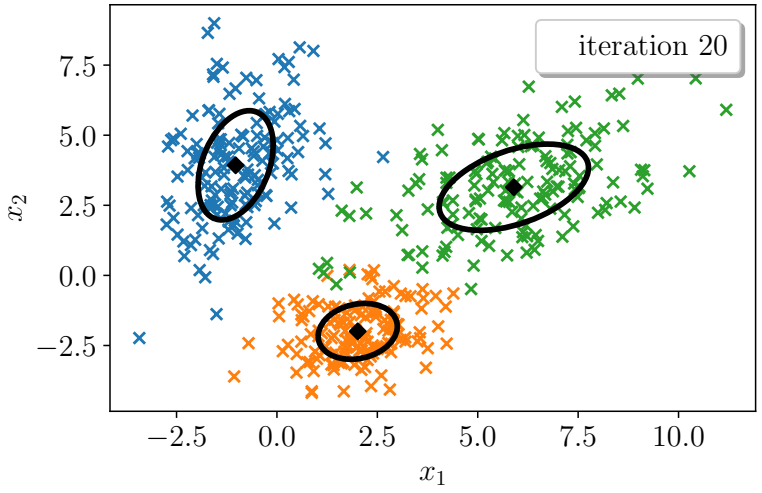


FIGURE 2.8: Fitting a three-component bivariate Gaussian mixture model on the data samples (crosses). We see that there are three different subpopulations in the data, whose distributions have been well-approximated by the components of the mixture model. The digital readers can click on this figure to start an animation.

2.1.3 Mixture Density Networks

In supervised machine learning tasks, we care about approximating the distribution of a target $\boldsymbol{y}$ conditioned on an input $\boldsymbol{x}$, i.e., $P(\boldsymbol{y} \mid \boldsymbol{x})$. As regards regression problems, the usual underlying assumption is that the output $\boldsymbol{y}$ we observe might be noisy, and we let the distribution of such noise be Gaussian. Under these conditions, it is possible to show that the **function** that computes the expected conditional value of $\boldsymbol{y}$ given $\boldsymbol{x}$, i.e., $\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}}[\boldsymbol{y} \mid \boldsymbol{x}]$, corresponds to the optimal solution of our regression problem [18]. This is intuitively represented by the left hand-side of Figure 2.9. Put differently, as long as the conditional distribution of the output is **unimodal**, we can train function approximators such as a Multi-Layer Perceptron (MLP) [19, 20] to solve the task.

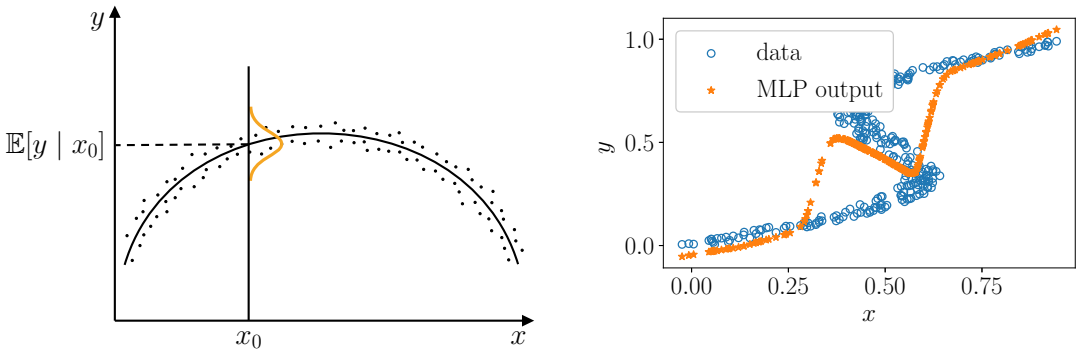


FIGURE 2.9: We sketch an example of a common regression dataset (left) in which we assume a unimodal Gaussian noise around each output value. Instead, on the right we show an example of an inverse problem, where y can be multi-valued for each x and an MLP fails at capturing the data distribution which cannot be expressed by means of a function.

Now let us imagine that the unimodality assumption does not hold anymore. In particular, given an input vector $\boldsymbol{x}$, there can be more than one plausible values for the output $\boldsymbol{y}$. This is true for the so-called *inverse* problems, such as robot inverse kinematics and stochastic simulations, where the mapping can be multi-valued. As we show in the toy example, adapted from [18], of Figure 2.9 (right), an MLP cannot express uncertainty about the possible values of $\boldsymbol{y}$ given $\boldsymbol{x}$, due to the fact that neural networks are *function* approximators. Nonetheless, we could use the mixture models from the previous section to capture the (possibly **multimodal**) distribution $P(\boldsymbol{y} \mid \boldsymbol{x})$ via maximum likelihood estimation. The crucial difference with respect to the classical formalization of mixture models is the supervised nature of the task. We call this kind of problems **Conditional Density Estimation** (CDE) tasks.

Therefore, the idea behind the Mixture Density Network (MDN) model [18] is to compute multimodal output distributions conditioned on arbitrary **flat** input data. Since a fully probabilistic formulation with closed-form solutions is difficult in the general case,

MDN minimizes the negative log-likelihood of the data using backpropagation. We take advantage of the extended plate notation introduced before and represent the Mixture Density Network as a graphical model in Figure 2.10.

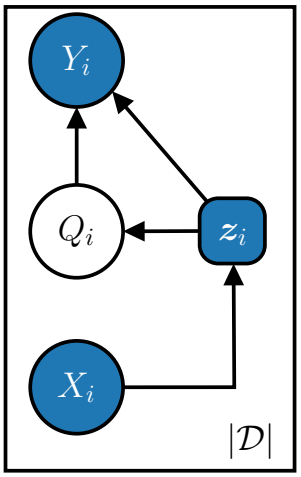


FIGURE 2.10: We sketch the graphical model for a Mixture Density Network model. Notice how the value of the variable X_i undergoes a deterministic transformation into the hidden representation $\boldsymbol{z}_i$.

The first step of the process is to encode the input $\boldsymbol{x}$ into a hidden representation $\boldsymbol{z}$. This can be achieved, for instance, with any neural network of choice. Then, the first C components of the hidden representation, namely $\boldsymbol{z}^Q$ are used to compute the *conditional* mixing weights for the C output distributions:

$$P(Q = j|\boldsymbol{x}) = \frac{e^{z_j^Q}}{\sum_{j'}^{C} e^{z_{j'}^Q}}.$$

The other components of the $\boldsymbol{z}$ vector are used to compute the sufficient statistics of the C output distributions. For a continuous output y, this amounts to having $2C$ parameters in $\boldsymbol{z}$ associated with the means ($\boldsymbol{z}^\mu$) and standard deviations ($\boldsymbol{z}^\sigma$) of univariate Gaussians. On a practical note, the variance can be kept stricly positive by applying an exponential transformation to the related components of $\boldsymbol{z}$, whereas under/overflow numerical errors can be mitigated with the "exp-normalise trick"

$$P(Q = j|\boldsymbol{x}) = \frac{e^{z_j^Q - b}}{\sum_{j'}^{C} e^{z_{j'}^Q - b}}, \quad b = \max_{j'} z_{j'}^Q.$$

Similarly to mixture models, the objective to maximize is the log-likelihood of the i.i.d. samples:

$$\log P(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{j}^{C} P(y \mid z_j^\mu, z_j^\sigma) P(Q = j \mid \boldsymbol{x}).$$

To show that MDNs can solve the conditional density estimation problem for the toy dataset of Figure 2.9, we train a three-component MDN on that dataset and visualize both the values of the mixing weights $P(Q \mid x)$ and those of the means z_i^{μ} (Figure 2.11 - top). To observe whether the learned probability distribution captures the uncertainty associated with each data point x, we also draw the probability contour plot of the model as well as the mean value of the most likely conditional mode (Figure 2.11 - bottom); the latter is selected by simple inspection of the mixing weights. We can see how a MDN is able to solve the conditional density estimation problem, by providing a multimodal distribution for each input x that can be appreciated by looking at the contour plot.

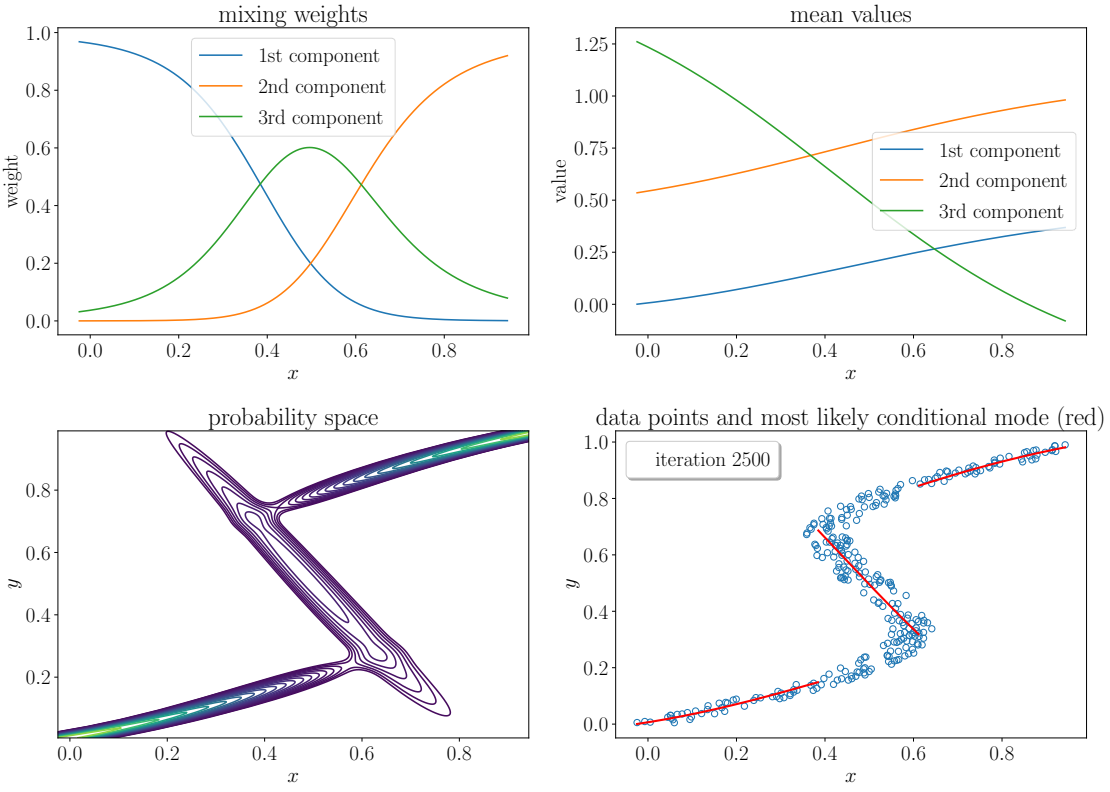


FIGURE 2.11: Fitting a three-component Mixture Density Network model on the toy dataset of Figure 2.9, as well as the value for the mixing weights and Gaussian means when the value of the scalar input x varies. The model can correctly capture the data distribution by properly mixing the different Gaussians. The digital readers can click on this figure to start an animation.

2.1.4 Bayesian Nonparametric Mixture Models

One of the most recurrent questions that machine learning practitioners ask when using mixture models is "how many components should we use?". In this section, we introduce techniques that **automatically** answer this question by exploiting the available data. The field of Bayesian Nonparametric (BNP) methods deals with statistical models that are, in fact, *not parametric*: the parameters of the model **grow/shrink with the data**, and as such the model cannot be simply formalized by means of a fixed number of parameters. Of course, this does not mean that we shall not make any assumption about the underlying data distribution, as we still have hyper-parameters to tune and distribution families to choose. The BNP field has been extensively studied [21–27] so, in the interest of conciseness, we shall focus on the most relevant and intuitive concepts that will be needed in the following. We will provide a high-level summary of a Dirichlet Processes (DPs), its basic properties, and ways to represent DP-based models, before discussing how we can perform inference in (Hierarchical) **DP mixture models**.

Let us start the discussion with the fundamental notion of **exchangeability** [25]. Informally, this notion specifies that the order in which a sequence of n identically distributed observations $\{X_1, \dots, X_n\}$ appear does not influence the joint probability distribution $P(X_1, \dots, X_n)$.

Definition 2.13 (**Exchangeability**)**.** Let $\{X_1, \dots, X_n\}$ be a set of n random variables, each defined on the same probability space, and let P be their joint distribution. These variables are said to be exchangeable if, for any permutation σ of {1,...,n} it holds

$$P(X_1 = x_1, \dots, X_n = x_n) = P(X_1 = x_{\sigma(1)}, \dots, X_n = x_{\sigma(n)}).$$

Notice that this does not imply the observations are i.i.d.; on the contrary, i.i.d. variables are implicitly exchangeable. The notion of exchangeability is very important in the Bayesian/non-Bayesian debate about modeling the parameters as random variables. Because of De Finetti's Theorem [28], under the exchangeability assumption there exists a random variable $\boldsymbol{\theta}$ such that $P(X_1, \dots, X_n) = \int P(\boldsymbol{\theta}) \prod_{i=1}^{n} P(X_i \mid \boldsymbol{\theta}) d\boldsymbol{\theta}$. Therefore, for exchangeable sequences, there exists a Bayesian model whose parameters are in fact a **random variable**.

For the rest of the section, we shall follow [26] and assume a setting where observations are exchangeable. Without going into needless technicalities, we define a **Dirichlet Process** [29] as a distribution over other probability distributions. We parametrize a DP using a **base distribution** G_0, which is the expected value of the process, and a **concentration** (or scaling) parameter α_0 that controls how close DP realizations are to

G_0. Moreover, draws from $\mathrm{DP}(G_0, \alpha_0)$ almost surely form a discrete distribution, even when G_0 is continuous.

Intuitively, to generate new data using a DP, one first draws a distribution G from the DP and then uses that distribution to independently sample values for $X_1, X_2, \ldots$. Nevertheless, the harsh truth is that, in practice, one cannot directly sample a distribution from a DP, as doing otherwise would require an infinite amount of information to represent the DP. Consequently, over the years researchers developed different ways to draw samples from a DP [25]. Some of them, like the Chinese Restaurant Process (CRP) [30] define the DP **implicitly**. Others describe a random draw, rather than the distribution, in an **explicit** matter, e.g., the Stick-Breaking construction [22]. Finally, it is possible to take the limit of a **finite and parametric** model to obtain a nonparametric one, an instance of which is the finite mixture model that can be converted into a DP mixture model [24]. In the following, we shall focus on the Stick-breaking representation for its convenient implementation.

2.1.4.1 The Stick-Breaking Construction

We now present the Stick-Breaking construction, an explicit method to represent a DP. To start, imagine having a stick of length 1 and splitting it every time we need to create a new component in an infinite but discrete distribution (as in Figure 2.12). While each component is assigned to a piece of the stick, whose length represents the prior probability of that component, all the other infinitely many components are **represented by the unassigned portion of that stick**, that is, the remaining portion of the black stick in the figure. Practically speaking, this allows us to implement a DP mixture model on

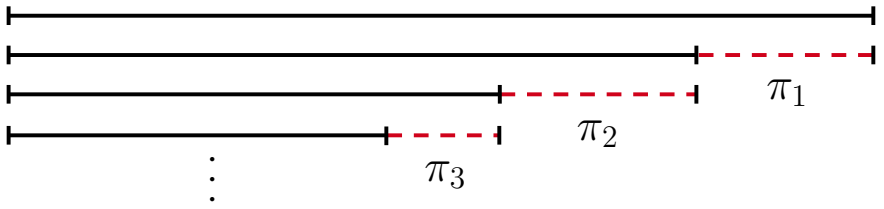


FIGURE 2.12: By recursively splitting a stick of length 1, we can obtain an infinite number of prior probabilities π_i.

a computer, since we could create a potentially infinite number of mixtures but only a finite amount of them is stored in memory.

On a more formal note, the stick-breaking construction is based on sequences of independent (but not identically distributed) random variables, namely $\{\pi_k\}_{k\in\mathbb{N}}$ and $\{\boldsymbol{\theta}_k\}_{k\in\mathbb{N}}$:

$$\pi'_k \mid \alpha_0, G_0 \sim \mathrm{Beta}(1, \alpha_0) \qquad \boldsymbol{\theta}_k \mid \alpha_0, G_0 \sim G_0$$

where $\text{Beta}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ is the Beta distribution. Defining a discrete probability distribution G as

$$\pi_k = \pi'_k \prod_{l=1}^{k-1}(1-\pi'_l) \qquad\qquad G = \sum_{k=1}^{\infty} \pi_k \delta_{\boldsymbol{\theta}_k}$$

where $\delta_{\boldsymbol{\theta}_k}$ is the **Dirac** distribution with all probability mass concentrated on $\boldsymbol{\theta}_k$, it can be shown [22] that G is distributed according to $\text{DP}(\alpha_0, G_0)$.

Because it holds $\sum_k \pi_k = 1$, we can interpret $\boldsymbol{\pi} = \{\pi_k\}_{k\in\mathbb{N}}$ as a discrete random variable whose support is $\mathbb{N}$. From now on, we shall identify the stick-breaking construction using the standard terminology $\boldsymbol{\pi} = \text{Stick}(\alpha_0)$ [24].

2.1.4.2 Dirichlet Process Mixture Models

A Dirichlet Process mixture model is a BNP model in which a DP acts as prior in an infinite mixture of distributions [24–27]. Assuming G has distribution $\text{DP}(\alpha_0, G_0)$, we can formalize this model by writing

$$\begin{aligned} \phi_i \mid G &\sim G \\ x_i \mid \phi_i &\sim F(\phi_i), \end{aligned}$$

where we use the abstraction $F(\phi_i)$ to denote which emission distribution to use for the **factor** ϕ_i. For instance, ϕ_i might take as a value the parameters of a mixture model component, whose emission distribution is represented as $F(\phi_i)$. The graphical model is visually represented on the left hand-side of Figure 2.13. We can draw an alternative formalization using the stick-breaking construction we just introduced. First of all, let us assume that the factor ϕ_i of a data point x_i can take values $\boldsymbol{\theta}_k$ with probability π_k, according to the formulation of G in the stick-breaking construction. Then, we introduce the random variable Q_i, distributed as $\boldsymbol{\pi}$, with support over the set $\mathbb{N}$; the value Q_i is interpreted as the index of an infinite mixture component. Whenever $Q_i = q_i$, the emission distribution for x_i will be a $\boldsymbol{\theta}_{q_i}$ drawn from G_0. The graphical model is sketched in Figure 2.13 (right hand-side) and formally defined below:

$$\begin{aligned} \boldsymbol{\pi} \mid \alpha_0 &\sim \text{Stick}(\alpha_0) & q_i \mid \boldsymbol{\pi} &\sim \boldsymbol{\pi} \\ \boldsymbol{\theta}_k \mid G_0 &\sim G_0 & x_i \mid q_i, \{\boldsymbol{\theta}\}_{k\in\mathbb{N}} &\sim F(\boldsymbol{\theta}_{q_i}). \end{aligned}$$

Due to the sheer complexity of the BNP treatment, we will defer details about inference (i.e., learning) to the following chapters. However, for the sake of clarity, let us conclude

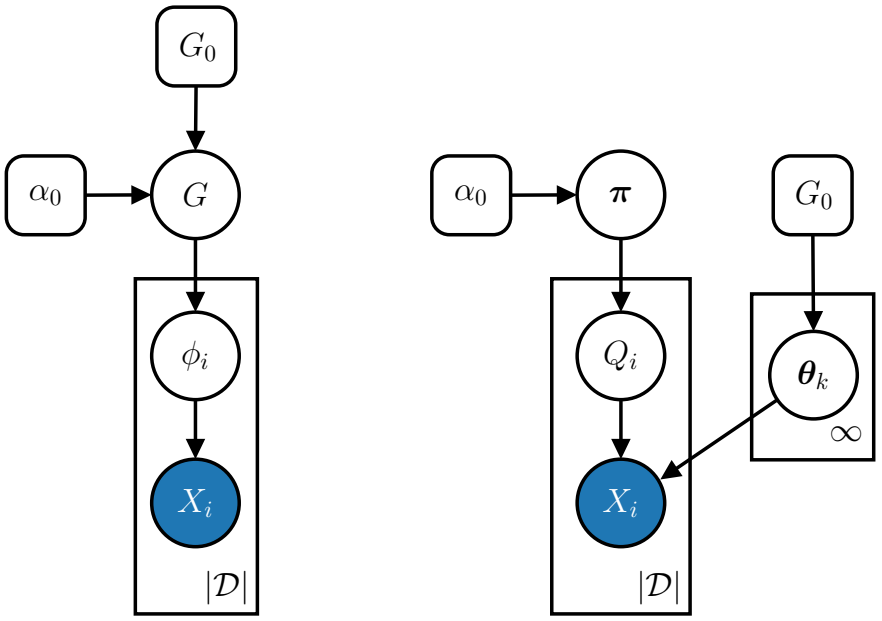


FIGURE 2.13: The Dirichlet Process Mixture Model can be graphically represented in more than one way. Here, we present its classical version (left) as well as the Stick-Breaking variant (right).

this section by showing, in Figure 2.14, how a DP Gaussian mixture model **learns** the "right" number of clusters while fitting the data of a toy problem.

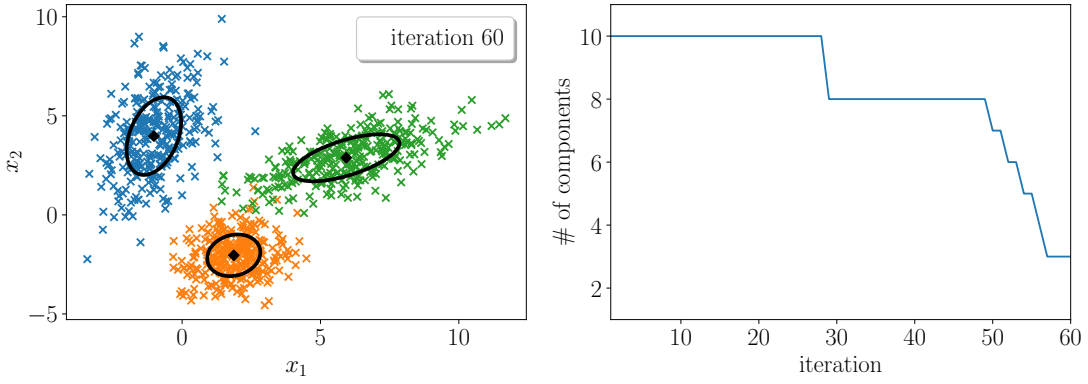


FIGURE 2.14: We show the behaviour of a DP Gaussian mixture model (variational inference implementation) that starts with 10 Gaussian components initialized using the kmeans algorithm [31]. Iteration after iteration, the number of active components decreases until it reaches 3, the same number of clusters that generated the data. In other words, the model adapted its complexity to fit the underlying data distribution. The digital readers can click on this figure to start an animation.

2.1.4.3 Hierarchical Dirichlet Process Mixture Models

In this thesis, we will work with a more complex version of a DP called Hierarchical Dirichlet Process (HDP) [24]. In essence, an HDP adds a prior on the base distribution

G_0 using another DP, which is parametrized by a concentration parameter γ and a base distribution H. The rationale behind the HDP is given by the problem setting: the data points belong to J **known** and related groups, and each group has its own DP mixture model [24]. In addition, we want to *induce dependencies* between these DP mixture models by sharing the parameters of the emission distributions, i.e., the clusters are the same. As before, there exists a stick-breaking version of the HDP that we depict in Figure 2.15.

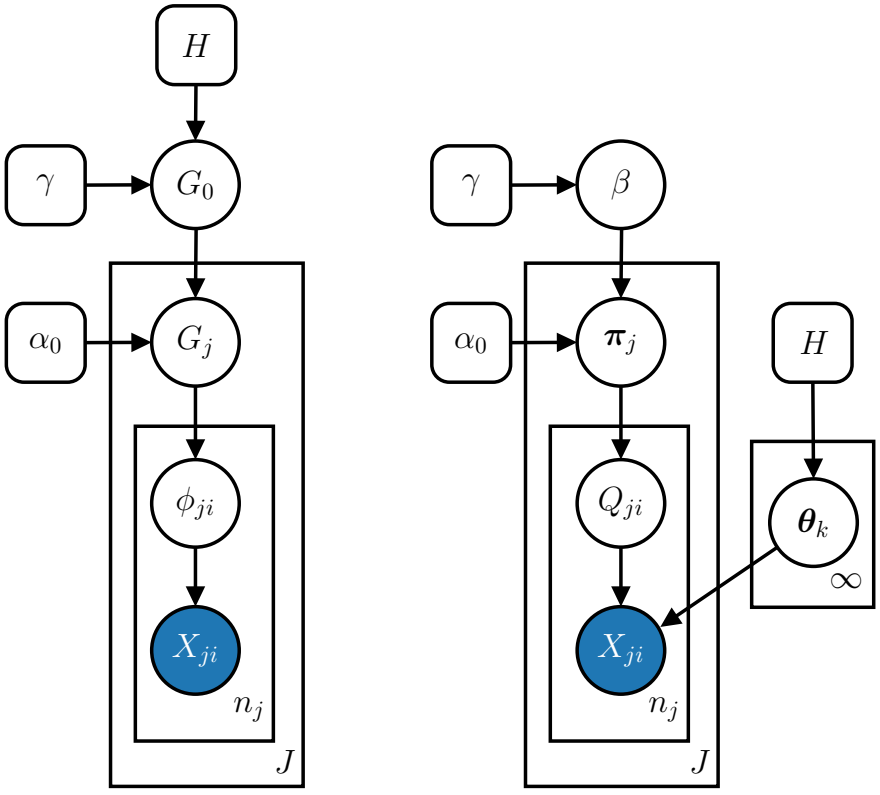


FIGURE 2.15: We visualize the Hierarchical Dirichlet Process Mixture Model in both its classical version (left) and Stick-Breaking alternative (right).

We now define the formulation of the stick-breaking construction for an HDP. Recall that, for the i-th data point, the assignment to a group j is known; we will use the subscript x_{ji} to reflect this notion.

$$\begin{aligned} \boldsymbol{\beta} \mid \gamma &\sim \text{Stick}(\gamma) \\ \boldsymbol{\pi_j} \mid \alpha_0, \boldsymbol{\beta} &\sim \text{DP}(\alpha_0, \boldsymbol{\beta}) \qquad & q_{ji} \mid \boldsymbol{\pi}_j &\sim \boldsymbol{\pi}_j \\ \boldsymbol{\theta}_k \mid H &\sim H & x_{ji} \mid q_{ji}, \{\boldsymbol{\theta}\}_{k \in \mathbb{N}} &\sim F(\boldsymbol{\theta}_{q_{ji}}). \end{aligned}$$

However, it still remains unclear how one can sample $\boldsymbol{\pi}_j$ from $\mathrm{DP}(\alpha_0, \boldsymbol{\beta})$. In [24], the authors show that there exists a relationship between $\boldsymbol{\pi}_j$ and $\boldsymbol{\beta}$. In particular, it holds

$$\pi'_{jk} \sim \mathrm{Beta}\Big(\alpha_0\beta_k, \alpha_0\big(1 - \sum_{l=1}^{k}\beta_l\big)\Big) \qquad \pi_{jk} = \pi'_{jk}\prod_{l=1}^{k-1}(1-\pi'_{jl}).$$

All in all, these are the basic notions of the model upon which we shall build an infinite mixture model for graphs.

Inference. To perform inference in BNP mixture models, one usually relies on Markov Chain Monte Carlo (MCMC) algorithms [23, 25, 27]. This is possible whenever we can efficiently integrate out the infinitely many "unused" latent components of the posterior distribution, so that only the values for the finite number of active components are to be inferred using known inference procedures. Recall that the idea of MCMC is to define a Markov chain on the hidden variables, and by drawing samples from this chain we **eventually** get a sample coming from the true posterior; the Gibbs sampling algorithm we already saw belongs to the family of MCMC methods. Generally speaking, it is not uncommon that MCMC algorithms require many draws before producing high-quality samples, though the burn-in period heavily depends on the chosen algorithm.

Notice that, due to the random variables being exchangeable, Gibbs sampling is particularly suitable for DP mixture models [27]. Indeed, each observation in the dataset can be chosen as the last to use, since the order of appearance of the values does not matter in the joint distribution. This is why we will rely on Gibbs sampling for the BNP method developed in this thesis.

2.2 Graph Basics

In this section, we shall give a few but fundamental definitions, taken from graph theory [32] and deep learning for graphs [1], which will be abundantly used throughout the rest of this work. Generally speaking, a graph is a highly flexible data structure whose entities of interest are connected to each other by some particular form of relationship. The way connections are organized is often called the **structure** (or **topology**) of the graph. It is this flexibility what makes it hard to learn from graphs, since we need to take into account the topological variability of each input sample.

2.2.1 Fundamentals

Let us commence with a more precise definition of what a graph is.

Definition 2.14 (**Graph**)**.** A graph is a tuple $g = (\mathcal{V}_g, \mathcal{E}_g, \mathcal{X}_g, \mathcal{A}_g)$ where $\mathcal{V}_g$ is the set of **vertices** (or **nodes** with slight abuse of terminology [33]) identifying the entities of interest, and $\mathcal{E}_g$ is the set of **edges** (or **arcs**) that couple pairs of **adjacent** vertices. We will follow the notational convention that a vertex in a graph is identified by a natural number with symbols u or v. Instead, $\mathcal{X}_g$ is a function that takes a vertex $u \in \mathcal{V}_g$ and maps it to a vector of **vertex features** (or attributes) $\boldsymbol{x}_u$; similarly, $\mathcal{A}_g$ maps edges to **edge features** $\boldsymbol{a}_{uv}$. Note that the **size** of a graph g corresponds to the cardinality of the vertex set, i.e., $size(g) = |\mathcal{V}_g|$.

When vertex features are available, we require that *each* vertex u has its own feature vector, and the same must hold for each edge. In jargon, when vertex and/or edge features are used, the graph must be "uniformly labelled" [4]. For many practical problems, is often the case that $\mathcal{X}_g : \mathcal{V}_g \to \mathbb{R}^d, d \in \mathbb{N}$ and $\mathcal{A}_g : \mathcal{E}_g \to \mathbb{R}^{d'}, d' \in \mathbb{N}$. Nonetheless, due to the nature of the models proposed in this thesis, most of the time we will consider discrete or continuous values . In case a graph has no vertex feature information, it suffices to consider an equivalent graph in which all vertices have the same dummy feature; clearly, the same goes whenever edge features are missing.

To express the notion of "direction" in the connections, we can distinguish between *directed* and *undirected* graphs.

Definition 2.15 (**Directed/Undirected Graph**)**.** A graph $g = (\mathcal{V}_g, \mathcal{E}_g, \mathcal{X}_g, \mathcal{A}_g)$ is said to be directed when the edges are ordered, pairs of vertices i.e., $\mathcal{E}_g \subseteq \{(u, v) \mid u, v \in \mathcal{V}_g\}$, and the edges are **oriented** with **tail** u and **head** v. On the contrary, when the vertex pairs are not ordered, i.e., $\mathcal{E}_g \subseteq \{\{u, v\} \mid u, v \in \mathcal{V}_g\}$, we talk about undirected graphs and **non-oriented** edges. In general, an edge connecting u and v is said to be **incident** to both vertices.

An undirected graph is useful when, e.g., representing molecules and mutual social interactions. Instead, a directed graph can be used to model road networks or hyperlinks, where the direction of the edge conveys additional information. Figure 2.16 depicts two directed and undirected graphs, where the direction is graphically characterized by the arrow symbol. To avoid confusion with the graphical notation of a random variable, we will place the id of the vertex *outside* the respective circle; when clear from the context, we may omit the vertex id in favor of a cleaner visualization.

Another fundamental concept is that of degree of a vertex; we first provide its definition for directed graphs.

Definition 2.16 (**Degree**)**.** Let g be a directed graph. The **in-degree** of a vertex $u \in \mathcal{V}_g$ is defined as the number of ordered edges with head u, that is, $indegree(u) =$

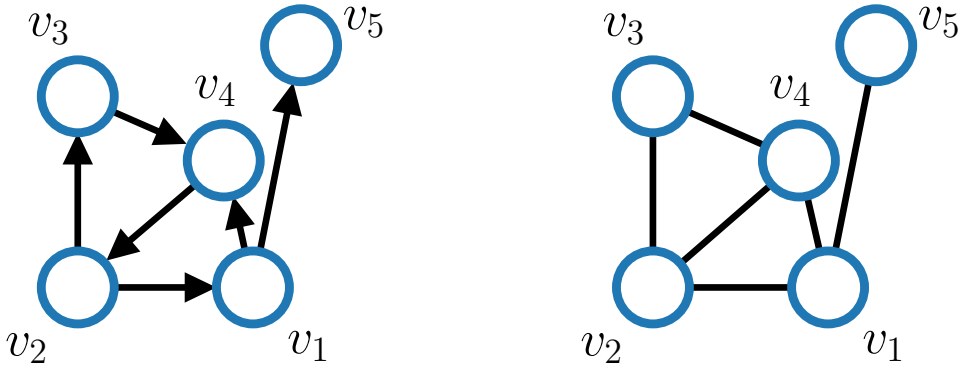


FIGURE 2.16: On the left, we present a simple example of a directed graph of size five, whereas its undirected counterpart is shown on the right.

$|\{v \mid (v,u) \in \mathcal{E}_g\}|$. In contrast, the **out-degree** of a vertex u is given by the number of ordered edges with tail u, i.e., $outdegree(u) = |\{v \mid (u,v) \in \mathcal{E}_g\}|$.

For most of the manuscript, we will implicitly work with the **in-degree** of a vertex u, denoting it as $deg(u)$ to simplify the notation. It follows that for undirected graphs there is no distinction between in-degree and out-degree.

As an alternative to $\mathcal{E}_g$, the structural information of a graph can be encoded by its square adjacency matrix.

Definition 2.17 (**Adjacency Matrix**)**.** The adjacency matrix of a graph g is a binary square matrix $\mathbf{A} \in \{0,1\}^{|\mathcal{V}_g| \times |\mathcal{V}_g|}$ where each entry A_{uv} is 1 if an edge links u and v together and 0 otherwise. In the case of undirected graphs, their adjacency matrix is symmetric. Moreover, whenever edge features are scalars, we can encode this information in a **weighted adjacency matrix** with $A_{uv} = a_{uv}, a_{uv} \in \mathbb{R}$.

The out-degree of vertex u can be extracted from the adjacency matrix by computing the sum of the values on the u-th row $A_{u,:}$, while the in-degree requires to sum over all values on the u-th column $A_{:,u}$. From the adjacency matrix, we can also construct the corresponding Laplacian matrix:

Definition 2.18 (**Symmetric Normalized Laplacian Matrix**)**.** Let g be a graph with adjacency matrix A, and let D be the diagonal degree matrix with entries $D_{uu} = deg(u)$. Then, the symmetric normalized Laplacian matrix of g is a square matrix defined as

$$L^{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

where $L = D - A$ is the unnormalized Laplacian matrix.

The symmetric normalized Laplacian has a great deal of "good" properties that, especially in the case of undirected graphs, come in handy when studying its properties and machine learning models via spectral graph theory (see section 2.3).

We now introduce the concept of a cycle. Cycles are particularly insidious for deep learning models that aim at automatically extracting features from graphs; the reason will become clear in the next chapter.

Definition 2.19 (**Cycle**)**.** Let us informally define a **path** as a sequence of distinct edges that allow us to move from vertex u to v in the corresponding graph. A graph cycle occurs when there exists a non-empty path from u to itself with no repeated edges. If a graph contains no cycles, then it is called **acyclic**.

Instances of cyclic graphs have already been provided in Figure 2.16. Moreover, a graph is called **connected** if there exists a path from each vertex to another. Strictly related to cycles, another fundamental challenge when learning from graphs is the absence of a **consistent** topological ordering of the vertices across the dataset.

Definition 2.20 (**Topological Ordering**)**.** A topological ordering of a **directed** graph g is a total order over its vertices such that, for every oriented edge $(u, v) \in \mathcal{E}_g$, u comes before v in such ordering.

Crucially, there exists a topological ordering of the directed graph if and only if it is acyclic. In this case, we talk about Directed Acyclic Graphs (DAGs). In addition, we say a graph is **ordered** if there exists a total order on the edges incident to each vertex (**unordered** otherwise) [4]. An example of an ordered graph is the Directed Ordered Acyclic Graph (DOAG) of Figure 2.17 (left). Similarly, a **positional** graph is an ordered graph with bounded in-degree and out-degree for which there exist two injective functions mapping edges that enter or leave a vertex to a distinctive positive integer (**non-positional** otherwise). The main difference between ordered and positional graphs is that, in the latter, some positions are allowed to be absent. We sketch a Directed Positional Acyclic Graph (DPAG) on the right handside of Figure 2.17.

Remark. From now on, we will assume to work with the very general class of (un)directed, (a)cyclic and (non-)positional graphs. To cope with the methodologies presented here, whenever an undirected graph is provided as input it is simply transformed into its directed counterpart. In particular, every edge $\{u, v\}$ is converted into two *distinct*, *opposite* and *oriented* arcs (u, v) and (v, u); if an edge feature $\boldsymbol{a}_{uv}$ is present in the original graph, this gets copied into both $\boldsymbol{a}_{uv}$ and $\boldsymbol{a}_{vu}$.

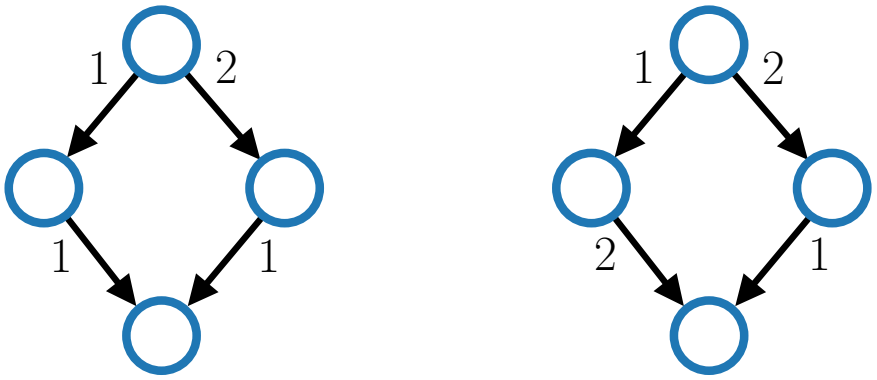


FIGURE 2.17: A Directed Ordered Acyclic Graph is sketched on the left, whereas a Directed Positional Acyclic Graph is on the right. For simplicity of exposition, we have defined total orders for outgoing edges only.

In the field of deep learning for graphs, another important notion is that of a vertex neighborhood. Intuitively, the neighborhood defines the **local view** of each vertex.

Definition 2.21 (**Neighborhood**)**.** Given a directed graph g and a vertex $u \in \mathcal{V}_g$, the neighborhood of u is the set of vertices connected to u with an ordered edge:

$$\mathcal{N}_u = \{v \in \mathcal{V}_g \mid (v, u) \in \mathcal{E}_g\}.$$

The neighborhood of u is **closed** if it always includes u and **open** otherwise. Whenever the image of $\mathcal{A}_g$ is the finite and discrete set $\{c_1, \ldots, c_n\}$, we shall extend our notation to define the subset of those neighbors that are connected to u with an arc labeled as c_k: $\mathcal{N}_u^{c_k} = \{v \in \mathcal{N}_u \mid \boldsymbol{a}_{vu} = c_k\}$.

Figure 2.18 provides a simple visual depiction of a vertex neighborhood.

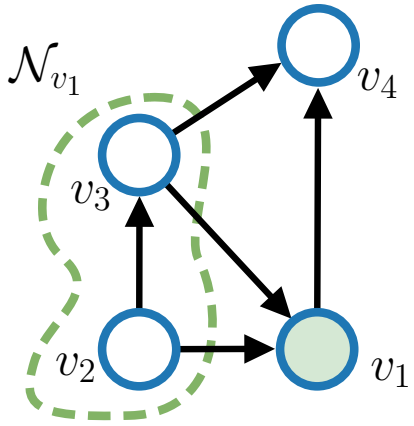


FIGURE 2.18: The neighborhood of vertex v_1 is here drawn as the set of vertices belonging to the dashed green region.

Furthermore, let us formally define when two graphs are structurally equivalent, which can be related to the expressive power of machine learning models for structured data.

Definition 2.22 (**Isomorphism**)**.** Two graphs g' and g'' are isomorphic (ignoring their vertex and edge features) if there is a bijections $f : \mathcal{V}_{g'} \rightarrow \mathcal{V}_{g''}$ such that two vertices u, v are adjacent if and only if $f(u), f(v)$ are adjacent.

We conclude this first part by informally introducing the reader to the notion of **structural transductions**, and we refer to [4] for a detailed mathematical treatment. A transduction is, in general, a binary relation between two spaces $\mathcal{U}$ and $\mathcal{Y}$, but we will restrict ourselves to functions $\mathcal{T} : \mathcal{U} \rightarrow \mathcal{Y}$. Now assume that both $\mathcal{U}$ and $\mathcal{Y}$ are *structured* spaces, e.g., they both represent the set of possible graphs, and define the skeleton of a graph g, namely $\text{skel}(g)$, as the graph obtained by discarding all vertex or edge labels. We say that a transduction $\mathcal{T}(\cdot)$ is **IO-isomorph** if it holds

$$\text{skel}(\mathcal{T}(g)) = \text{skel}(g) \quad \forall g \in \mathcal{U}.$$

As we shall see in the next chapter, IO-isomorph transductions are central to most deep learning architectures for graphs.

2.2.2 Instances of a Graph

Depending on the constraints posed on the connections of a graph, we obtain very specific families of structures.

Definition 2.23 (**Sequence**)**.** A sequence is a connected acyclic graph in which vertices are adjacent if and only if they are consecutive in the topological ordering induced on the graph.

Definition 2.24 (**Tree**)**.** A tree is a connected acyclic graph in which any two vertices are connected by exactly one path.

Because of their flexibility, trees have been successfully used in natural language processing and chemistry to model, for example, syntactic dependencies in sentences and molecules.

There is a special class of graphs that, due to specific patterns in the structure, is usually impossible to be discriminated by current deep learning models for graphs as well as isomorphism testing algorithms such as the 1-dim Weisfeiler-Lehman (WL) test [34].

Definition 2.25 (***k*-Regular Graph**)**.** A k-regular graph g is one in which $deg(u) = k \ \forall u \in \mathcal{V}_g$.

As a way of example, Figure 2.20 shows two 2-regular graphs that, however, are very different in nature. Indeed, the former is a disconnected graph whereas the second is

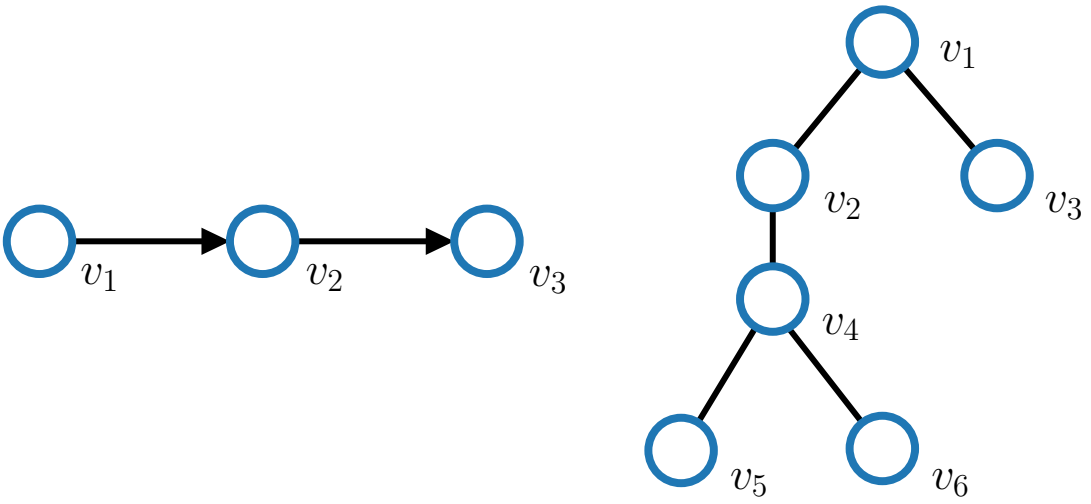


FIGURE 2.19: We sketch an directed sequence on the left and an undirected tree on the right.

connected, but from the point of view of their degree distributions they look identical. This is the reason why neither the 1-dim WL test nor most of the works on deep learning for graphs can distinguish these two graphs as non-isomorphic.

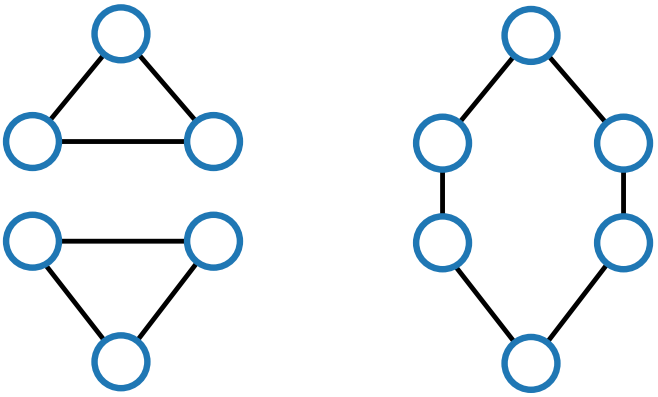


FIGURE 2.20: Two instances of 2-regular graphs are presented. When ignoring vertex or edge features, it is impossible to distinguish these two graphs by the 1-dim WL test of graph isomorphism.

2.2.3 Random Graphs

A topic at the intersection of graph and probability theory is that of random graphs [35]. This term generally refers to probability distributions defined over the discrete mathematical representation of graphs. These distributions are usually defined by means of a stochastic process over the creation of vertices and/or edges. Different random graph distributions give rise to graphs with peculiar characteristics. In what follows, we introduce two popular random graph distributions that are used to generate synthetic datasets in this thesis.

Definition 2.26 (**Random Graph**)**.** A random graph is a graph G of size N, where each pair of vertices is connected with probability p.
We use the capital letter G because we are dealing with random processes, and refer to the family of random graphs with the term $G(N, p)$.

In particular, the probability to obtain a *particular* realization g (without vertex/edge attributes) of G with $|\mathcal{E}_g| = M$ is (for undirected connections):

$$P(G(N,p)) = p^M (1-p)^{\binom{N}{2}-M}.$$

The Erdős-Rényi Model

The Erdős-Rényi (ER) model [36, 37] is one of the pioneering works on random graphs. The realization of an undirected graph g is obtained by considering all possible ordered pairs (u, v), $u, v \in \mathcal{V}_g$ and sampling an edge between them with probability p. The resulting distribution of the degree is **binomial** as follows:

$$P(deg(u) = k) = \binom{N-1}{k} p^k (1-p)^{n-1-k}$$

and a visualization of possible realizations of an ER graph are shown in Figure 2.21 for different values of p.

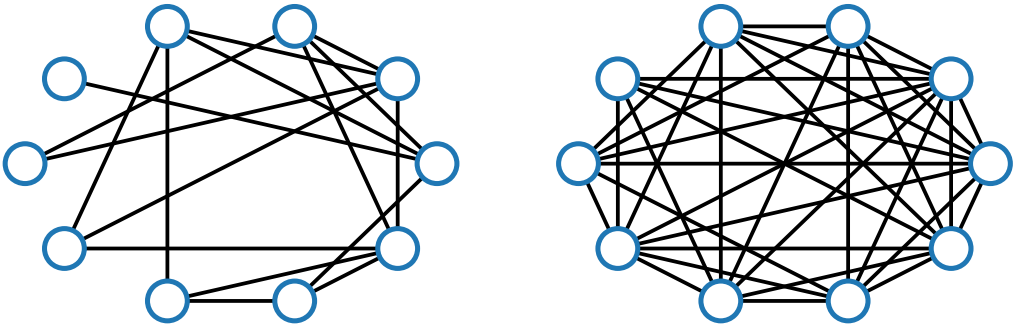


FIGURE 2.21: We generate two possible realizations of Erdős-Rényi graphs of size 10 with $p = 0.3$ (left) and $p = 0.8$ (right).

The Barabási-Albert Model

In the real world, it is not difficult to find examples of graphs that have very different properties from the ER model. For example, social networks exhibit a **scale-free**

property, i.e., the degree distribution follows the so-called **power law**

$$P(k \mid \gamma) = k^{-\gamma}$$

parametrized by γ. In simple terms, this means that there are few vertices with high degree (the "hubs") and many vertices with low degree. The construction of scale-free graphs follows the **preferential attachment process**, commonly known as "the rich get richer"[1], and the Barabási-Albert (BA) model [38] is one way to define a distribution over scale-free graphs. The construction takes the size of the graph N and a number of edges m to add at each step. It starts by instantiating n_0 vertices, then it adds a new vertex u and links it to m existing vertices following a sort of preferential attachment criterion

$$p_{uv} = \frac{deg(v)}{\sum_{v' \leq v} deg(v')}.$$

It can be shown that the degree distribution of the BA model follows the power-law distribution $P(k) = k^{-3}$. We conclude by showing, in Figure 2.22, two examples of BA graphs for varying values of the connectivity parameter m.

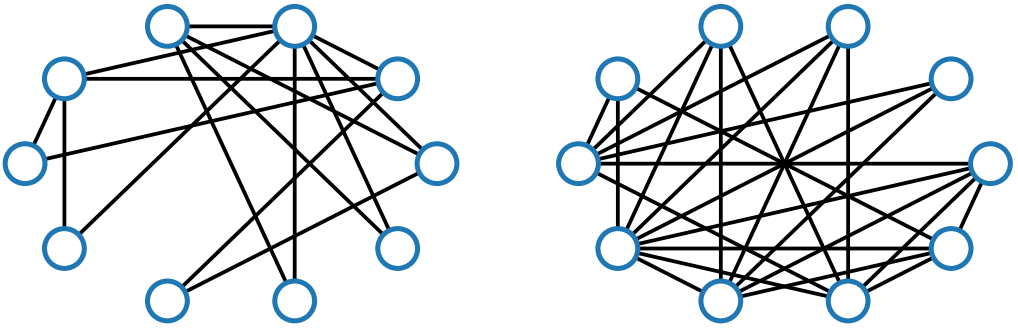


FIGURE 2.22: We generate two possible realizations of Barabási-Albert graphs of size 10 with $m = 2$ (left) and $m = 5$ (right).

2.3 What this Thesis is Not About

This thesis focuses on deep learning for graphs with an emphasis on Bayesian techniques, but it would be inappropriate not to mention the different lines of research that address graph-related problems in alternative and original ways. Below, the reader can find a non-exhaustive list of some prominent approaches in the literature. Please bear in mind that the references below are just a few representative examples of a much vaster literature [1, 39–42].

[1] The "rich get richer" view can be found in Dirichlet Processes as well.

Kernels

Kernel methods are one of the most long-standing and mature approaches that allow to compare graphs [43–48]. Briefly, a kernel is a function $k(\cdot, \cdot)$ that computes a similarity score for each pair of inputs in a given dataset $\mathcal{D}$. The resulting $|\mathcal{D}| \times |\mathcal{D}|$ matrix K is generally required to be positive-definite, even though this constraint is sometimes overlooked without significant adverse effects. The "classical" drawback of most kernels is that the function k has to be manually designed, and as such its choice may not be suitable for all tasks. Moreover, computing the kernel matrix can become unfeasible for very large datasets (with some exceptions [45]). Some of the ways in which one can compute similarities between pairs of graphs is to take inspiration from the WL test of graph isomorphism [45, 49] or to extract DAGs from each graph and then use tree-based kernels to compare them [50, 51]. The matrix K, often called **Gram matrix**, can be incorporated into Support Vector Machines (SVMs) [52] to address binary or multiclass graph classification.

Alternative methods may not be directly formulated as kernels but have an interpretation in terms of them. In particular, we mention a recent *probabilistic* approach [53] that is somewhat related to graphlet (sub-graph) kernels [54] and was shown to be competitive against both kernels and deep learning models.

Clearly, when the properties of interest are known, kernels prove to be more than adequate to solve the task at hand. Whenever this is not the case, it might be preferable to *learn* to extract features from a graph according to some unsupervised or supervised criterion, without posing handcrafted restrictions on the kind of patterns to look for.

Statistical Relational Learning

Other classical approaches to deal with graph structures belong to the field of statistical relational learning [55, 56]. This field combines probability theory, inductive logic, and learning to create data-driven models that possess a strong inductive bias rooted in logic rules. Examples to be ascribed to this field are Markov logic networks [57], which combine Markov random fields [58] and first-order logic, and conditional random fields [59] for classification of sequences. Differently from the methodologies described in this thesis, statistical relational learning does not typically rely on deep architecture to solve graph-related tasks, even though some hybrid approaches exist [60].

Spectral Graph Theory

The objective of spectral graph theory is to mathematically characterize graphs through the analysis of the adjacency and Laplacian matrices. This is another well-studied topic that finds a diverse set of applications, from Laplacian smoothing [61] to graph semi-supervised learning [62, 63] and spectral clustering [64]. We can also view the list of vertex features as a graph signal to be processed with ad-hoc signal-processing techniques: the Graph Fourier Transform [65] has allowed to extend the formal definition of convolution over graph signals, and subsequent approaches [66] used approximations of spectral graph convolutions to learn graph filters.

A limitation of learned spectral techniques is the lack of generalization to new graph instances. In fact, everything depends on the eigen-decomposition of the specific Laplacian matrix, whose eigenvector matrix Q is the orthonormal basis used to define the **Graph Fourier Transform** on the graph signal $\mathbf{f} \in \mathbb{R}^{\mathcal{V}_g}$:

$$\mathcal{F}(\mathbf{f}) = Q^T\mathbf{f}$$
$$\mathbf{f} = \mathcal{F}^{-1}(Q^T\mathbf{f}) = QQ^T\mathbf{f},$$

where we used the ortogonality of Q to obtain the inverse. Then, the convolution in the graph domain between a filter $\boldsymbol{\theta}$ and the graph signal can be written in a similar way to the usual Fourier analysis [67, 68]:

$$(\mathbf{f} \otimes \boldsymbol{\theta}) = QWQ^T\mathbf{f}$$

where $W = diag(Q^T\boldsymbol{\theta})$ is the diagonal filter matrix (which can be learned [66]). The filter matrix, however, will only work with graphs that are identical, so it will not generalize to different graph instances. Also, computing the exact eigen-decomposition becomes unfeasible for large graphs. All those issues motivated the study of approximate techniques [69] using the truncated Chebyshev expansion [65]. Later on, it was proposed to consider only the first term of said expansion [70], and the resulting approximation was used as a layer for a deep architecture for graphs.

Random Walks

A random walk in a graph is a path starting from a vertex and exploring the surrounding neighborhood in a stochastic way. Random walks are often used to characterize a wider neighborhood of vertices: it is an attempt to acquire both local and non-local information to create meaningful vertex representations [44, 71–73]. There are different ways in which random walks can be constructed and used: frameworks like Node2Vec [74] explore the

surroundings of a vertex in a way that depends on the chosen hyper-parameters. Indeed, hyper-parameters determine whether to traverse the graph in depth or breadth search style. The objective to maximize is the likelihood of a vertex given the information extracted by the random walks. Similarly, methods like DeepWalk [75] model each random walk as a sentence, still maximizing a likelihood objective inspired by skip-grams. More recently, graph generation has been tackled with random walks [76], whereas a connection between deep learning for graphs and random walks has been investigated in [77].

Graph Generation

Graphs are discrete and combinatorial mathematical objects, and as such it is not obvious how to exactly define distributions over them. We have seen that random graphs define a process by which it is possible to generate skeletons of graphs, but now the question becomes "is it possible to **train** a model to generate graphs?". Additionally, we would like to have our model creating graphs that are *original* and variable in size. Many are the practical implications of such an approach, for instance developing new drugs conditioned on certain properties that must hold true, or similarly for the science of materials discovery.

Due to the nature of the input data, it is not easy to use gradient-based methods to approximate the underlying distribution $P(g)$. Instead, one usually conditions the generative process on a set of latent representations, either for the entire graph or its individual vertices. Most approaches can be therefore divided between **graph-level** and **vertex-level** decoding. The former generates an adjacency matrix starting from a latent graph representation [78–80], whereas the second connects vertices depending on the similarity of their latent representations [81, 82]. Note that graph-level decoders are generally sensitive to the ordering of the vertices because they assume a fixed ordering of the adjacency matrix, whereas node-level decoders do not suffer from this limitation. Among the generative graph models that are fully differentiable, we mention auto-encoder based generators [79, 83–87] and generative-adversarial networks [88–91]. Finally, another family of models generates graphs as a result of a sequence of actions, which showed interesting generalization performances but is sensible to the vertex ordering [92–97].

Trustworthy AI for Complex Data

Being able to determine if a model complies with the trustworthy AI principles, such as fairness, privacy, robustness, explainability, and transparency, remains an open and valuable research question. Indeed, robustness to adversarial attacks, e.g., perturbations

of the vertices or edges of topologically different graphs, guarantees that a model will behave as expected when deployed [98–105]. Similarly, fairness principles in graph applications have been analyzed [106] to ensure that age/gender information about the entities is properly used and does not correlate too much with the target vale. While the probabilistic models presented in this thesis hold promise for what concerns interpretability and explainability (due to the explicit formulation of the causal effects), we leave such ideas for future research.

Theoretical Characterization of Models' "Expressiveness"

There is a very active line of research devoted to the theoretical analysis of the discriminative power of deep learning methods for graphs. Among them, we mention studies on the effect of depth and width [107, 108], as well as enhancements to the main graph convolution mechanism that will be introduced later [109, 110] and its theoretical characterization in terms of *communication capacity* [111]. In addition, researchers have spent much effort in deriving equivalence relations between the k-dimensional WL test of graph isomorphism and specific deep learning architectures [49, 112, 113] that discriminate k-regular graphs. Finally, others have built lower and upper bounds on the expressiveness of specific classes of learning algorithms [114].

Chapter 3

Principles of Deep Graph Networks

Io non posso ritrar di tutti a pieno,
però che sì mi caccia il lungo tema,
che molte volte al fatto il dir vien meno.

Inferno - Canto IV

The goal of the chapter is to give a high-level overview of the field of machine learning for graphs. Our main contribution is the standardization of the literature under a unified framework that let us look at similarities, differences, and novelties through the same lens. After some opening remarks on the main principles of contextual information processing, we build the discussion around the core building blocks of the field, such as neighborhood aggregation mechanisms, graph pooling, readout transductions and learning criteria. Equipped with this general understanding of deep learning for graphs, we shall then talk about scholarship issues in graph classification tasks. In fact, the rapid growth of interest in the field came at the price of poor reliability of empirical procedures. We shall discuss our effort in this direction, particularly how we carried out a fair, robust, and reproducible evaluation to help researchers avoid empirical malpractices in the future. To consolidate the theoretical notions, we will conclude the chapter with a real-world application from the field of molecular biosciences. By training a deep learning model on different protein realizations, we show that it is possible to efficiently predict the amount of "information loss" between the all-atom system we would like to study and one of its simpler but coarser representations. Overall, this "gentle" introduction creates fertile ground for what will come afterwards, that is, Bayesian deep learning models for graphs.

3.1 Contextual Processing of Information

To be able to put the recent burst of excitement about deep learning for graphs into the right perspective, we first go on a historical tour to show that the core ideas have already been there for more than twenty years. As a matter of fact, recent advancements of the field have not come without a certain forgetfulness of the pioneering methods that, however, shaped the field in its infant stages and are still relevant today.

We pick up from the transduction $\mathcal{T}$ introduced in the previous chapter. Generally speaking, when solving a supervised task on structured data, we can decompose the transduction into two phases. First, there is an IO-isomorphic transduction $\mathcal{T}_{enc}$ that computes a vectorial encoding $\boldsymbol{h}_v$ of each vertex v in the structure; ideally, such an encoding (called **state** or **representation**) should capture information about the vertex's surroundings, in order to differentiate it from other vertex states. Then, if we care about individual entities' prediction, a **readout** transduction $\mathcal{T}_{out}$ has to map each vertex state to its corresponding output value y_v. Instead, when the task requires a single prediction y_g for the whole structure, all vertex states have to be first aggregated into a single representation $\boldsymbol{h}_g$ using a readout formed by the composition of a global aggregation $\mathcal{R}$ with an output transduction $\mathcal{T}_{out}$. This rather high-level scheme is summarized in Figure 3.1, and it serves as our entry point into the world of **Structured-Data Learning** (SDL).

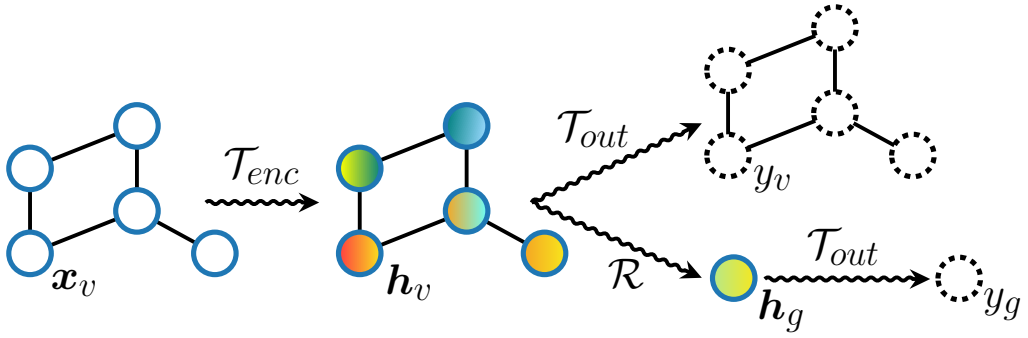


FIGURE 3.1: High-level visualization of most SDL approaches. Each vertex of the input structure g is mapped to a state $\boldsymbol{h}_v$ after which, depending on the nature of the task, an IO-isomorphic output transduction $\mathcal{T}_{out}$ is applied *or* all vertex states are first aggregated into a single state $\boldsymbol{h}_g$ by a global aggregation function $\mathcal{R}$.

Hereinafter, we will be primarily concerned with ways to define the encoding transduction $\mathcal{T}_{enc}$, as this estabilishes *how* vertex states are computed.[1] Any SDL transduction relies on three main properties listed below

- **Causality**: the encoding of an entity v depends only on itself and its *descendants* in the structure. The causality assumption intuitively acts as an inductive

[1] A reader interested in the theoretical aspects of the global aggregation and readout functions can refer to [109, 115–117] and the references therein.

bias on the transduction by stating which dependencies between entities should be considered relevant. While strictly related to the class of input structures, causality assumptions and topology are actually different concepts. For instance, in an undirected sequence there are at least three straightforward causality assumptions one can make about the direction of the dependencies, with consequently different outcomes and machine learning architectures.

- **Stationarity**: the encoding of an entity v is the same regardless of v's identity. Practically speaking, stationarity is deeply intertwined with the notion of weight sharing, as it defines how we can reuse the same encoding mechanism on every entity. We can also distinguish between *full* stationarity, where we do not make any assumption on the class of structures under consideration, and *positional* stationarity for DPAGs, in which we use different parameters for each position of the vertex's children to be considered [118].

- **Adaptivity**: the transduction is learned from the data. Whether the architecture is end-to-end differentiable or it admits closed-form update equations via the EM algorithm, the crux of the matter is that we want to avoid as much as possible any form of preprocessing of the structure. This is strongly related to the notion of **representation learning** [119, 120].

In the abundant and pioneering literature for sequence, tree and DAG learning [2–4, 115, 118, 121–125], the encoding transduction admits a recursive definition of the vertex state space, respectively. This is possible because the causality assumptions on these structures, mostly inspired by the available topological ordering, do not incur in infinite loops that would generally make the definition of the vertex state unfeasible.

From now on, we shall use the term **contextual processing** when talking about computations whose output depends on the information encoded in the structure, i.e., according to its topology and related causal assumptions. Also, the **context** of a vertex state $\boldsymbol{h}_v$ shall be the set of states that directly or indirectly contribute to determining $\boldsymbol{h}_v$.[2]

Let us make a concrete example to better understand the relation between causal assumptions and contextual processing of information. Regardless of whether the acyclic structure of Figure 3.2 is directed or undirected, we can make different causal assumptions using its topology. Here, we assume top-down (left) and bottom-up (right) causal dependencies between vertex states. The state of vertex v, indicated by the orange dotted circle, is *recursively* computed in terms of the states of the descendants. We use dashed edges to denote the state dependencies (the target depends on the source), which may differ from the original topological representation of the structure. Nested boxes

[2] Please refer to [107] for a formal characterization of the context of a vertex state.

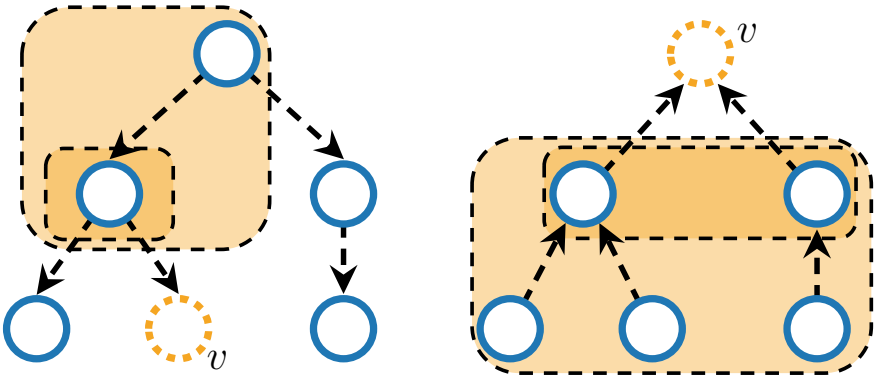


FIGURE 3.2: Example of different causal assumptions on the same tree structure. Dashed arrows denote state dependencies rather than the actual topology, whereas encapsulated boxes show how the context of vertex v increases as the recursive computation is unfolded.

aim to convey the amount of context involved at each recursive step in the definition of $\boldsymbol{h}_v$. We observe how vertices can encode different kinds of contextual information, i.e., the state of the parents up to the root or the entire subtree state rooted in v. The choice of a particular causal assumption is clearly task-dependent, since it encodes our bias about what kind of information vertex states should contain.

So far, we have considered **acyclic** structures, for which learning algorithms based on backpropagation through time [126, 127] or structure [2, 122, 128, 129] exist and are practical to implement. Notwithstanding their importance, the above algorithms are generally unsuitable when it comes to learn vertex representations of a **cyclic** graph. In addition, some of these algorithms even assume that the size or number of parents/children are bounded by some value. On the other hand, we seek a practical methodology that can seamlessly treat cyclic graphs.

3.2 Deep Learning for Graphs [1]

Having defined the challenges ahead of us, it is time to describe the solutions proposed in the literature. Simply put, most works focus on the development of practical and effective models that *automatically* extract features of interest from graphs of *varying* topology while generalizing to *unseen* instances. This is why the field is often referred to as **Graph Representation Learning** (GRL). In an attempt to disambiguate other terms [130, 131], in this thesis we will adopt the uniforming terminology of **Deep Graph Networks** (DGNs) [1]. Below, we discuss how most DGNs address the presence of cycles, lack of a topological ordering, and variable topology in the dataset.

3.2.1 Local Computation of Vertex States

Two are the common solutions adopted in the literature to abstract from graphs of different topology, even though they are rarely stated explicitly. First of all, causality assumptions are relaxed, because taking into account the descendants of a vertex v creates issues when that vertex belongs to a cycle. In other words, pairs of vertex states are assumed to be **independent** when conditioned on the states of their *neighbors*. Secondly, due to the lack of a predefined total ordering of the neighboring vertices, in the most general case one has to assume *full* stationarity of the encoding process. As a result, not only do we apply the same *learned* function to all vertices, but we also need this function to be flexible enough to accept a variable number of neighboring states in any possible ordering. This is why **permutation invariant** functions are often employed in these models: their output does not change upon reordering of the input elements. Examples of such functions are the element-wise sum, mean, and product operators. Importantly, there exist conditions [116, 117] under which it is possible to express any continuous permutation invariant function $\Psi : \mathcal{X}^M \to \mathbb{R}$, with $\mathcal{X}$ being an uncountable set: if we consider a finite number M of elements, then

$$\Psi(x_1, \dots, x_M) = \phi(\sum_{i=1}^{M} \psi(x_i))$$

with ϕ and ψ being continuous functions that can be approximated by neural networks [132]. For the rest of this work, we will use the Greek letter Ψ to denote a permutation invariant function.

3.2.2 Breaking Cycles via Iterations

By now, the attentive reader may have noticed that the issue of mutual dependencies induced by cycles has not been solved yet. To see this, just imagine that a vertex is connected to itself via a self-loop. Clearly, we cannot compute its state using the aforementioned local approach because one of the neighboring vertex states is the state of the vertex itself. Therefore, other than being local, the processing of information in DGNs has to be **iterative**: the state of a vertex is conditioned on neighboring states computed "at some previous iteration". If we already have some information on which to condition the vertex states, we can effectively **break cycles** in the structure with a simple iterative approximation.

Crucially, a local and iterative processing of information allows us to **propagate contextual information** across the graph. If we unfold the computation on the graph of Figure 3.3, we can see how at iteration $\ell = 2$ the state $\boldsymbol{h}_u$ depends on the state of $\boldsymbol{h}_v$ at

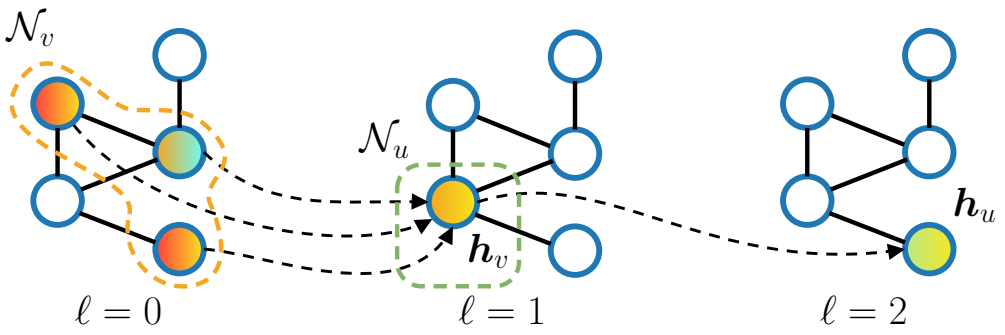


FIGURE 3.3: We show how the context of a vertex spreads in an undirected graph as we iterate the local processing of information described above. Dashed arrows represent the context flow as we unfold the computational graph.

$\ell = 1$. In turn, $\boldsymbol{h}_v$ is obtained by applying a learned permutation invariant function to the neighboring states of iteration $\ell = 0$. Therefore, by repeated iteration, each vertex increases its context according to the graph topology [107]. In what follows, we shall refer to the state of vertex u at iteration ℓ with the symbol $\boldsymbol{h}_u^\ell$. To bootstrap the overall process, it is often the case that $\boldsymbol{h}_u^0 = \boldsymbol{x}_u$.

A popular formalism to describe the above process is that of "**message passing**" [133–137]. In particular, there are two operations associated with each vertex:

- *message dispatching*: compute a message for each vertex to be propagated to the neighbors. The message may depend on the vertex state as well as edge information.
- *state update*: the state of each vertex is updated **in parallel** using the incoming messages.

Likewise, we can also imagine traversing the graph in no particular order to update the vertex states, and then iterate this traversal as many times as needed. This recalls the idea of Convolutional Neural Networks for images[3] [138], where at each layer a filter passes over all pixels to compute new values based on the pixels' surroundings and multiple layers increase the *receptive field* (i.e., the context).

3.2.3 Three Styles of Context Propagation

In the above schemes, the notion of "iterative process" is sufficiently generic to encompass different styles of context diffusion. Therefore, we partition DGNs into three main families, namely **recurrent**, **feedforward**, and **constructive** architectures. We now separately describe their characteristics.

[3]Images can be represented as graphs where vertices are organized in a grid and for which a "natural" ordering exists.

Recurrent architectures. Recurrent machine learning models for graphs treat the encoding process as a dynamical system, but at the same time they rely on contractive dynamics to ensure some convergence criterion can be met. The Graph Neural Network (GNN) [130] and the Graph Echo State Network (GraphESN) were the first models developed in this sense. On the one hand, GNN is a recurrent neural network that relies on constraints in the (supervised) objective function to ensure convergence. On the other hand, GraphESN brings the Reservoir Computing approach to the processing of graphs, inheriting convergence from the contractivity property of the *untrained* pool of neurons. That said, it is also possible to fix in advance the number of iterations, regardless of whether convergence has been reached. This was the idea behind the Gated Graph Neural Network (GG-NN) [139].

Recurrent architectures treat the single iteration ℓ of the encoding process as a "time step" of the corresponding dynamical system, and they consist of a **single** layer of **recurrent** units to be repeatedly applied. Nevertheless, there exist multi-layered versions of these models such as the Fast and Deep Graph Neural Network (FDGNN) [140], which extends GraphESN to efficiently construct multi-resolution views of the graph.

Feedforward architectures. From the perspective of feedforward models, iterations are **layers** of a possibly deep architecture, where each layer has its own parameters to be optimized. Stacking multiple layers is a way to *compose* the context learned in a very flexible way, without being forced to impose contractive dynamics or use recurrent units. The Neural Network for Graphs (NN4G) [107] was the first feedforward model for graphs to be developed, defining what was later re-discovered as the "spatial graph convolutional layer" [70, 141].

This family is the most known at both industrial and research levels, for its simplicity, ease of implementation, and competitive performances on many different tasks [41]. At the same time, it inherits the same gradient-related issues of deep neural networks, in particular when trained in an end-to-end fashion [142, 143]. Differently from deep networks for flat data, though, here depth serves two purposes, i.e., automatically extracting features *and* propagating contextual information across the graph. Nowadays, it is very straightforward to build these models, thanks to the collective effort of researchers that released easy-to-use libraries for quick development and experimentation [1, 144, 145].

Constructive architectures. A constructive model is a special case of a feedforward model where training occurs one layer at a time. As a consequence, constructive models do not necessarily suffer from oversmoothing of representations [143] or vanishing/exploding gradient effects. When used with a supervised criterion, this methodology allows to

automatically determine the number of layers needed by the task, i.e., the amount of context to propagate, for example using Cascade Correlation [146].

One of the major characteristics of constructive models is that they approach the task in a *divide-et-impera* fashion, where each layer contributes to the solution of a sub-problem and subsequent layers try to better solve the task using the information extracted from previous layers. Notably, once a layer is trained its weights are frozen, meaning they do not change while training new layers. The NN4G belongs to the family of constructive approaches, as well as the **Contextual Graph Markov Model** (CGMM) [6] of Chapter 4.

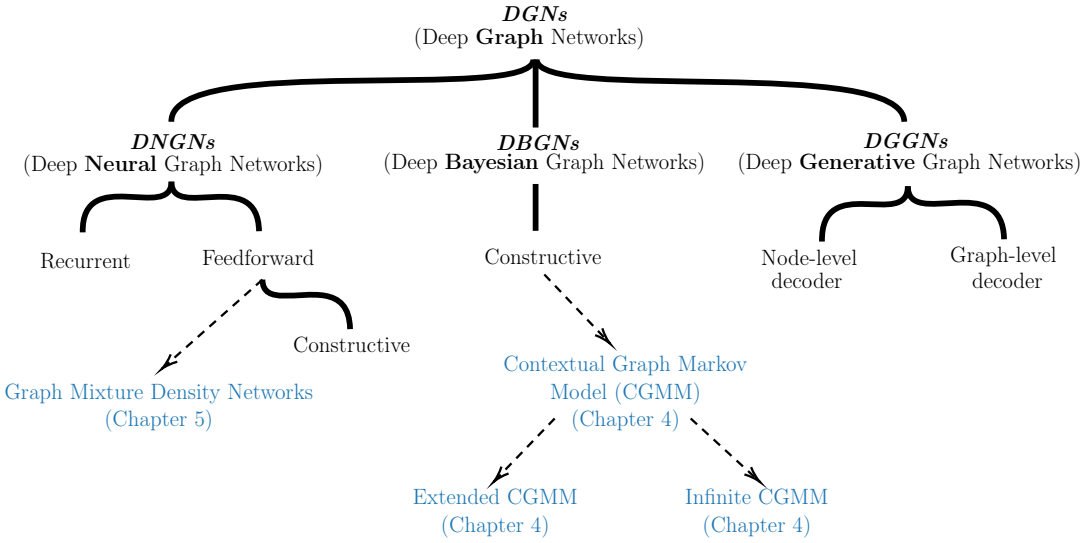


FIGURE 3.4: A taxonomy of the various context propagation mechanisms, with the addition of the specific models developed in this thesis.

Before we dive deep into the building blocks of Deep Graph Networks, we provide an illustrative taxonomy summarizing what we said so far and highlighting where the models developed in this thesis stand. The taxonomy, provided in Figure 3.4, is by no means exhaustive, but it encompasses how most works propagate contextual information. We also make the distinction between Deep **Neural** Graph Networks[4] (DNGNs), Deep **Bayesian** Graph Networks (DBGNs), i.e., probabilistic and deep models for graphs, and Deep **Generative** Graph Networks (DGGNs) [97], i.e., models that are able to *generate* new and original graphs. Notice that, in principle, one can also combine different context propagation mechanisms if the task is more complex than the usual, for instance by exploiting both feedforward and recurrent mechanisms to handle graphs that vary in time.

[4]Which are ambiguously referred to as Graph Neural Networks in the literature.

3.2.4 Core Modules

We are finally ready to describe the main mechanisms of Deep Graph Networks. Not only will these accompany us in the next chapters, making it easier to understand the rationale behind some of our technical choices, but they will also give a unifying view of the different approaches in the literature, which is one of the main contribution of this thesis.

Neighborhood Aggregation

The definition of the permutation invariant function computing the local encoding of each vertex in the graph is arguably the key building block of any DGN. Indeed, this neighborhood aggregation function imposes an architectural bias that has important consequences on the representational power of the model under consideration. A very straightforward form of neighborhood aggregation is

$$\boldsymbol{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\boldsymbol{h}_v^{\ell},\ \Psi(\{\psi^{\ell+1}(\boldsymbol{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big), \tag{3.1}$$

where ϕ and ψ are adaptive transformations of the input, e.g., Multi Layer Perceptrons. It can be shown [1, 41] how Equation 3.1 is a generalization of some of the most known aggregation schemes, such as the Graph Convolutional Network [70] and the Graph Isomorphism Network (GIN) [109]. Besides, these architectures ignore any additional information on the nature of the relation between vertices, which is usually stored by **edge attributes**. In chemistry, for instance, it is common to have discrete edge features describing the type of bond between atoms as well as continuous values associated with their inter-atomic distance. In the former case of $\mathcal{A}_g$ finite and discrete, one can extend the previous equation with additional parameters w_{c_k} to be learned for each discrete edge label c_k:

$$\boldsymbol{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\boldsymbol{h}_v^{\ell},\ \sum_{c_k \in \mathcal{A}} \big(\Psi(\{\psi^{\ell+1}(\boldsymbol{h}_u^{\ell}) \mid u \in \mathcal{N}_v^{c_k}\}) * w_{c_k}\big)\Big),$$

where the symbol $*$ stands for multiplication between a scalar and a vector and we recall, from Section 2.2.1, that $\mathcal{N}_v^{c_k}$ is the subset of v's neighbors whose connecting edges have label c_k. This is another form of stationarity over edge weights, since we are dealing with non-positional graphs. If graphs were positional, we could use a different weight for each position and each edge type. Also, please take note of how we grouped neighbors of v according to their edge type; this is a practice that will be implemented in the next chapter. In the literature, NN4G [107] and the Relational Graph Convolutional Network (R-GCN) [147] are just some of the models implementing this aggregation scheme.

On the contrary, if edge features $\boldsymbol{a}_{uv}$ are continuous, one can easily combine vertex and edge states:

$$\boldsymbol{h}_v^{\ell+1} = \phi^{\ell}\Big(\boldsymbol{h}_v^{\ell},\ \Psi(\{e^{\ell+1}(\mathbf{a}_{uv}) \cdot \psi^{\ell+1}(\boldsymbol{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big),$$

with $\cdot$ standing for the Hadamart product between vectors. The Message Passing Neural Network framework (MPNN) [137] and the Edge Conditioned Convolution (ECC) [148] implement this kind of aggregation at the expense of extra computation for the edges, which can significantly slow down the algorithm.

A natural question that comes to mind at this point is "even though the graph is non-positional, should we treat all neighbors as equal?". In fact, we may not, and that is the idea behind the **attention** mechanism [149] applied to the neighborhood aggregation of DNGNs. First of all, consider having a way to compute some sort of similarity score α_{uv} between two vertex states $\boldsymbol{h}_u$ and $\boldsymbol{h}_v$. We can extend the aggregation mechanism of Equation 3.1 to re-weight neighbors according to such a score:

$$\boldsymbol{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\boldsymbol{h}_v^{\ell},\ \Psi(\{\alpha_{uv}^{\ell+1} * \psi^{\ell+1}(\boldsymbol{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big).$$

We impose no restriction on how to compute α_{uv} but for the procedure being differentiable or non-adaptive. Nothing prevents us from combining the above equations to obtain an attention score that depends, for example, on edge features. The Graph Attention Network (GAT) [141] was the first model to apply a multi-head attention mechanism to a Deep Graph Network, with potential advantages in terms of interpretability.

From the above equations, it emerges how the time complexity of Deep Graph Networks is strictly related to the number of edges in the input graphs. Nevertheless, trying to scale DGNs to graphs with billions of edges poses two major challenges: first, the training time required is often unbearable for modest computing devices; secondly, the degree of some vertices is so high that aggregating all neighboring states can lead to numerical instability or oversmoothing, subject to the permutation invariant operator used. To mitigate these issues, **sampling** techniques have been proposed to reduce the set of neighbors to aggregate for each vertex. The idea is schematically represented in Figure 3.5, and it has been adopted by architectures like FastGCN [150] and GraphSAGE [151] to improve the generalization performances on different tasks. Moreover, sampling is not necessarily constrained to the immediate neighbors of a vertex: one can provide a more flexible notion of neighborhood, such as "all vertices at distance 2", and sample from that set [151]. This way, a wider and richer neighborhood can be explored, similarly to the random walks technique briefly mentioned at the end of last chapter.

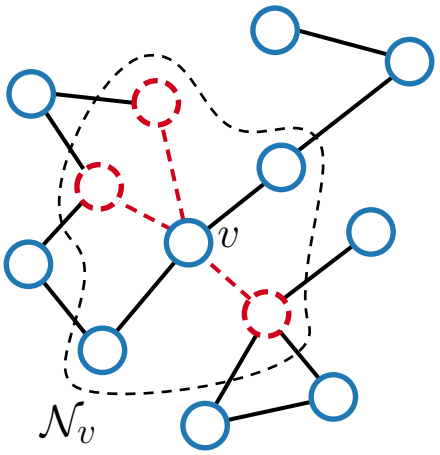


FIGURE 3.5: Sampling neighbors allows to keep the computational complexity of the aggregation function fixed and, sometimes, provides better generalization performances.

To conclude this section, we provide a table with different neighborhood aggregation schemes under the same uniform mathematical notation introduced so far. Note that one can also combine different aggregations together to increase the expressiveness of the model; this approach is adopted by models like Principal Neighborhood Aggregation [152].

Model	Neighborhood Aggregation
NN4G [107]	$\sigma\Big(\mathbf{w}^{\ell+1^T}\mathbf{x}_v + \sum_{i=0}^{\ell}\sum_{c_k\in\mathcal{C}}\sum_{u\in\mathcal{N}_v^{c_k}} w_{c_k}^i * \mathbf{h}_u^i\Big)$
GNN [130]	$\sum_{u\in\mathcal{N}_v} MLP^{\ell+1}\Big(\mathbf{x}_u, \mathbf{x}_v, \mathbf{a}_{uv}, \mathbf{h}_u^\ell\Big)$
GraphESN [153]	$\sigma\Big(\mathbf{W}^{\ell+1}\mathbf{x}_u + \hat{\mathbf{W}}^{\ell+1}[\mathbf{h}_{u_1}^\ell, \ldots, \mathbf{h}_{u_{\mathcal{N}_v}}^\ell]\Big)$
GCN [70]	$\sigma\Big(\mathbf{W}^{\ell+1}\sum_{u\in\mathcal{N}(v)} L_{vu}\mathbf{h}_u^\ell\Big)$
GAT [141]	$\sigma\Big(\sum_{u\in\mathcal{N}_v} \alpha_{uv}^{\ell+1} * \mathbf{W}^{\ell+1}\mathbf{h}_u\Big)$
ECC [148]	$\sigma\Big(\frac{1}{\lvert\mathcal{N}_v\rvert}\sum_{u\in\mathcal{N}_v} MLP^{\ell+1}(\mathbf{a}_{uv})^T\mathbf{h}_u^\ell\Big)$
R-GCN [147]	$\sigma\Big(\sum_{c_k\in\mathcal{C}}\sum_{u\in\mathcal{N}_v^{c_k}} \frac{1}{\lvert\mathcal{N}_v^{c_k}\rvert}\mathbf{W}_{c_k}^{\ell+1}\mathbf{h}_u^\ell + \mathbf{W}^{\ell+1}\mathbf{h}_v^\ell\Big)$
GraphSAGE [151]	$\sigma\Big(\mathbf{W}^{\ell+1}(\frac{1}{\lvert\mathcal{N}_v\rvert}[\mathbf{h}_v^\ell, \sum_{u\in\mathcal{N}_v}\mathbf{h}_u^\ell])\Big)$
CGMM [6–8]	$\sum_{i=0}^{\ell} w^i * \Big(\sum_{c_k\in\mathcal{C}} w_{c_k}^i * \big(\frac{1}{\lvert\mathcal{N}_v^{c_k}\rvert}\sum_{u\in\mathcal{N}_v^{c_k}}\mathbf{h}_u^i\big)\Big)$
GIN [109]	$MLP^{\ell+1}\Big((1+\epsilon^{\ell+1})\mathbf{h}_v^\ell + \sum_{u\in\mathcal{N}_v}\mathbf{h}_u^\ell\Big)$

TABLE 3.1: Here are some preeminent examples of neighborhood aggregation schemes present in the literature. We use square brackets to denote concatenation, whereas W, w and ϵ are weight to be learned. GraphESN's aggregation looks different because it assumes a maximum size of the neighborhood, but the core principles are the same. Also, we describe the *mean* version of GraphSAGE, though variations are possible.

Graph Coarsening

Separate from neighborhood aggregation, graph **pooling** is an (optional) independent module of a deep feedforward architecture that coarsens the latent graph representations in order to reduce the number of vertices. The purpose of graph pooling is three-fold, i.e., discover communities in the input graph, encode such knowledge in the vertex states, and finally to reduce the computational costs of the subsequent neighborhood aggregation modules. We can distinguish between *adaptive* pooling, whose parameters are learned using gradient descent techniques, and *topological* pooling, which leverages the topological properties of the graph with known non-adaptive algorithms. The idea of pooling is sketched in Figure 3.6.

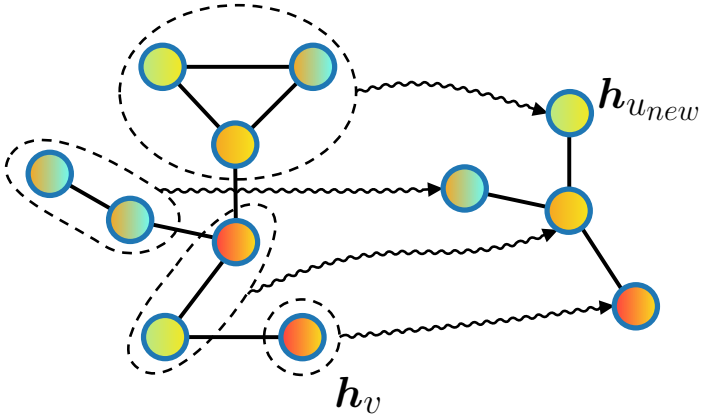


FIGURE 3.6: A pooling layer coarsens vertex representations to obtained a reduced graph representation that should encode higher-level details.

Adaptive pooling applies a differentiable transformation to vertex representations in order to produce soft cluster assignment scores, such as DiffPool [154]. The drawback of these methods is that they produce a dense adjacency matrix in output, which can become even more costly than the original graph if the number of chosen clusters is sufficiently large. Other approaches like Top-k pooling [155] try to address this problem by retaining only the top-k vertices according to some ranking. Also, adaptive pooling techniques are not restricted to vertices, rather they can be applied to edges [156] by collapsing vertices incident to the highest ranking edges.

Topological pooling, on the other hand, is non-adaptive and inspired by classical community discovery algorithms. The striking computational advantage of this family of pooling methods is that coarsened graphs can often be *precomputed*, thus significantly reducing the computational burden of the subsequent training phase. Among them, we mention spectral clustering approaches such as GRACLUS [157] and ARMA filters [158], as well as methods based on non-negative matrix factorization [159] and the k-plex cover algororithm [160].

Recently, there has been some criticism about the true benefits of pooling for graph classification on small datasets [161]. Local pooling was shown to become progressively invariant to cluster assignments of vertices, and simple baselines performed as well as methods employing pooling layers. Nevertheless, pooling can still be used to detect some form of community in the graph when we know there exists a latent hierarchy.

Global Aggregation

Whenever the task requires it, it may be necessary to aggregate all vertex representations to produce a single graph state summarizing all the information extracted by the model. Because there exists no topological ordering among vertices in general, we almost always rely on another permutation invariant function to compute the graph state at each iteration ℓ:

$$\boldsymbol{h}_g^\ell = \Psi\Big(\{f(\boldsymbol{h}_v^\ell) \mid v \in \mathcal{V}_g\}\Big). \tag{3.2}$$

Common choices for f and Ψ are the identity function and the element-wise sum, mean or max operators, even though nothing prevents us from using approximations of universal aggregators over multisets [116, 117]. In this manuscript, we will consider a graph representation that is the layer-wise concatenation of all graph states, in order to consider multiple "views" of the graph extracted by the model. Alternatives are possible though: [139] applies a Long Short-Term Memory (LSTM) [3] to the sequence of graph states $\{\boldsymbol{h}_g^0, \dots, \boldsymbol{h}_g^\ell, \dots\}$, whereas Sort Pooling picks a subset of vertex states according to a lexicographic ordering of such states [162].

To summarize the building blocks introduced so far, Figure 3.7 sketches a comparison between a feedforward architecture and a recurrent model for graphs, where branches indicate that we are either solving vertex or graph related tasks. Note how the feedforward network uses differently parametrized layers, in contrast to the single layer of a recurrent network. The application of pooling and of a global transduction can only occur in graph related tasks, since the topology of the output is irremediably changed by these operations. On top of these architectures, which implement the transductions $\mathcal{T}_{end}$ and, optionally, $\mathcal{R}$ of Figure 3.1, we can apply an output module implementing the $\mathcal{T}_{out}$ transduction.

3.2.5 Learning Criteria

Throughout the following chapters, we will deal with both unsupervised and supervised learning tasks such as maximum likelihood estimation on graphs, link prediction, vertex

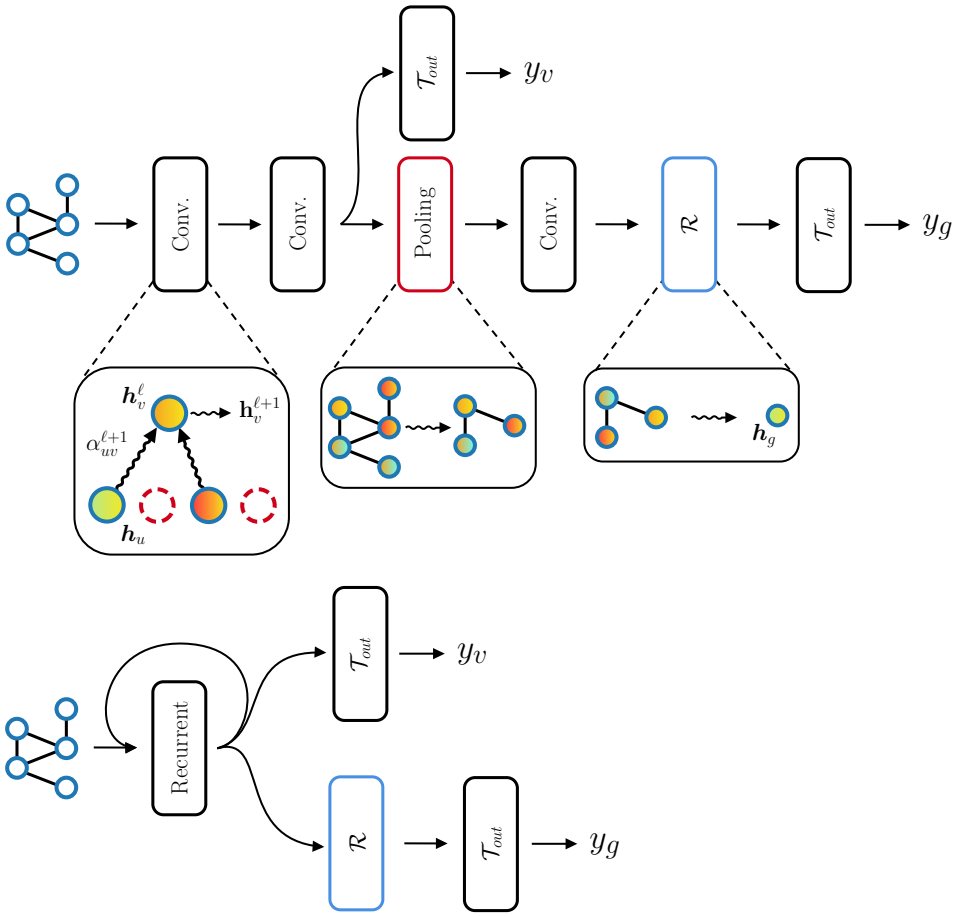


FIGURE 3.7: Putting everything together: a comparative example between feedforward (top) and recurrent (bottom) DGNs. See the text for more details.

and graph classification. Therefore, it is useful to revise how researchers have approached these tasks in the past.

Unsupervised Learning

The **maximum likelihood estimation** criterion is just one of the ways in which we can train a DGN in an unsupervised fashion. For instance, one can maximize the likelihood of all vertex features conditioned on the graph information, as described in detail in the following chapter. Whenever the graphical model does not allow a tractable computation of the likelihood, variational approximations can come in handy to train the model in a reasonable amount of time [60, 81]. As an alternative, reasearchers have also proposed ways to directly capture the distribution of a graph $P(g|\theta)$, by combining a graph encoder with a radial basis function network, as well as providing a definition of **attributed** random graphs [163, 164].

Predicting the existence of a link between entities can be very helpful in application domains such as drug repurposing and recommender systems [165–167]. **Link prediction**

tasks use the structure of the graphs in the dataset to train a model to reconstruct the adjacency matrix and predict missing links, so in some sense they correspond to a form of *self-supervision*. The most adopted loss is called **reconstruction loss**, adapted from auto-encoders:

$$\mathcal{L}_{rec}(g) = \sum_{(u,v)} ||\boldsymbol{h}_v - \boldsymbol{h}_u||^2,$$

which also has a probabilistic formulation [81]:

$$P((u,v) \in \mathcal{E}_g \mid \boldsymbol{h}_u, \boldsymbol{h}_v) = \sigma(\boldsymbol{h}_u{}^T \boldsymbol{h}_v).$$

It is important to remark that the reconstruction loss does not take into account vertex or edge features, and that it is mainly based on the **homophily assumption**: adjacent vertices are more likely to share the same characteristics and/or target label [168]. In tasks where the homophily assumption is verified, it might be advantageous to use this loss as a *regularizer*.

Another relevant unsupervised criterion is that of **graph clustering**, i.e., partitioning a graph into groups of vertices that share some similarity. Self-Organizing Maps (SOMs) [169] have been one of the most successful approaches to perform clustering on DAGs [170–172], before being extended to cyclic graphs [164, 173–175].

Borrowing ideas from information theory, **mutual information** approaches try to create representations that approximately maximize the mutual information between pairs of graphs. Deep Graph Infomax (DGI) [176] trains a discriminator to distinguish between a graph and its corrupted version, where the corruption algorithm must be defined a priori. The authors show that this kind of training indirectly maximizes the mutual information between graphs in the dataset. Similarly, the **entropy** of a categorical distribution can be used to regularize the soft-clustering matrix produced by adaptive pooling techniques such as DiffPool. Entropy can be used to encourage the pooling layer to produce one-hot assignments to the clusters for each vertex, even though it does not solve the "dense output adjacency matrix" problem.

Supervised Learning

The most common supervised tasks in which DGNs are involved are vertex classification, graph classification, and graph regression. In **vertex classification**, the goal is to assign a target label to each vertex in the graph. We also distinguish between *inductive* vertex classification, where the vertices to classify belong to unseen graphs, and *tranductive* vertex classification, which consists of vertex predictions on a single (large) graph with

some known vertex targets. Transductive vertex classification has suffered from serious reproducibility issues, due to arbitrary choices in the experimental setups regarding the same tasks [177]; in particular, it emerged that the data split was highly influential on these datasets. While there is a large stream of works around this "semi-supervised" setting, Bayesian approaches seem the more robust ones when few training labels are available [178]. In particular, one can implement a Bayesian version of the GCN model and marginalize over the neural network parameters when it comes to predict a new vertex label. This method can be extended to a nonparametric scenario, giving rise to a Bayesian version of the Variational Graph Autoencoder [81] that better models the adjacency matrix generation by taking into account noisy data [179].

As regards **graph classification and regression**, one usually applies a standard machine learning predictor on top of the graph representation computed using the techniques described earlier. The objectives to be minimized are the usual Cross-Entropy (CE) loss or the Mean Squared Error (MSE) Notwithstanding the simplicity of this approach, we noticed the same troubling trends in scholarship on a series of graph classification benchmarks [5]. Therefore, a contribution of this thesis, which we shall introduce in a moment, will be about addressing such issues.

Self-supervised Learning

Recently, many **self-supervision** objectives have been proposed to pretrain Deep Neural Graph Networks [180]. These include predicting the surrounding neighborhood information of vertices and using attribute masking strategies similar to those employed in the Natural Language Processing field [181] More generally, approaches to self-supervised learning for graphs can be divided into those that generate the feature and adjacency matrices, others than exploit contrastive learning based on information theory criteria, some that try to predict known properties of the graphs under consideration, and hybrid ones [182].

Summary

We have presented all the basic building blocks that allow us to reason about the main differences between deep architectures for graphs. To complement the discussion, we present a recap of the main properties of some DGNs in the literature. Specifically, Table 3.2 divides models by their style of context propagation, learning task, how layers are chosen, their intrinsic nature, and building blocks.

Broadly speaking, whether it is crafting a new DGN or choosing an existing one, one always has to keep in mind the characteristics of the task at hand, the available data, and the computational constraints. We can make a trivial example when considering the inductive bias of the neighborhood aggregation mechanism: if the task benefited from knowing the degree of each vertex, then a sum-based aggregation function would be the obvious choice. In contrast, when one wants to capture how neighboring representations are distributed around each vertex, a mean-based aggregation is an adequate, not to mention numerically stable, alternative. When the amount of supervised labels is limited, expressive models like GIN are prone to severely overfit the training data [109], so either one carefully applies a regularization strategy or a simpler, less parametrized aggregation like the ones of GraphSAGE or GCN is used. In cases where the amount of raw graphs is also much larger than the supervised samples or we need to quickly adapt to new tasks without retraining the entire model, unsupervised vertex/graph embedding mechanisms such as CGMM (Chapter 4) may be a viable technique to consider. In terms of computational requirements, taking into account edges and their features usually leads to an increase in the parameters of each DGN. This is because models like GAT apply multiple adaptive, non-linear transformations to each pair of adjacent vertex representations to compute an importance score for each neighbor, whereas others such as ECC transform each edge feature vector by means of an MLP. When applied to large and dense graphs, these models quickly become computationally demanding (see, for instance, Section 3.3). Here, exploiting any domain knowledge could be important to avoid such parametrizations and inject a favorable inductive bias into the model; as an example, if an atom is connected to others in space, the inverse of the inter-atomic distance could be exploited to diminish the importance of far-away atoms when aggregating neighbors in case the homophily assumption holds. More in general, whenever training is considered prohibitive because of hardware constraints and an extremely efficient solution is needed, GraphESN and FDGNN provide an advantageous trade-off between performances and a fully supervised training approach, and they can be considered good baselines against which to compare because of the randomized, untrained nature of the embedding construction. To conclude these consideration, it has to be mentioned that very few works in the field have tried to automatically determine the "right" number of graph convolutional layers to use for the underlying task during training. In this sense, NN4G also stands out as a pioneering approach that applies the principle of Cascade Correlation to tackle the task in a divide-et-impera fashion, by training one layer at a time. We believe that this could be an understudied research direction, especially as regards unsupervised and self-supervised methods that could mitigate the need of cross-validating this crucial hyper-parameter.

Model	Context	Embedding	Layers	Nature
GNN [130]	Recurrent	Supervised	Single	Neural
NN4G [107]	Constructive	Supervised	Adaptive	Neural
GraphESN [153]	Recurrent	Untrained	Single	Neural
GCN [70]	Feedforward	Supervised	Fixed	Neural
GG-NN [139]	Recurrent	Supervised	Fixed	Neural
ECC [148]	Feedforward	Supervised	Fixed	Neural
GraphSAGE[151]	Feedforward	Both	Fixed	Neural
CGMM [6, 7]	Constructive	Unsupervised	Fixed	Probabilistic
E-CGMM [8]	Constructive	Unsupervised	Fixed	Probabilistic
iCGMM (4.3)	Constructive	Unsupervised	Fixed	Probabilistic
DGCNN [162]	Feedforward	Supervised	Fixed	Neural
DiffPool [154]	Feedforward	Supervised	Fixed	Neural
GAT [141]	Feedforward	Supervised	Fixed	Neural
R-GCN [147]	Feedforward	Supervised	Fixed	Neural
DGI [176]	Feedforward	Unsupervised	Fixed	Neural
GMNN [60]	Feedforward	Both	Fixed	Hybrid
GIN [109]	Feedforward	Supervised	Fixed	Neural
NMFPool [159]	Feedforward	Supervised	Fixed	Neural
SAGPool [183]	Feedforward	Supervised	Fixed	Neural
Top-k Pool [155]	Feedforward	Supervised	Fixed	Neural
FDGNN [140]	Recurrent	Untrained	Fixed	Neural
GMDN [9]	Feedforward	Supervised	Fixed	Hybrid

Model	Edges	Pooling	Attention	Sampling
GNN [130]	Continuous	✗	✗	✗
NN4G [107]	Discrete	✗	✗	✗
GraphESN [153]	✗	✗	✗	✗
GCN [70]	✗	✗	✗	✗
GG-NN [139]	✗	✗	✗	✗
ECC [148]	Continuous	Topological	✗	✗
GraphSAGE[151]	✗	✗	✗	✓
CGMM [6, 7]	Discrete	✗	✗	✗
E-CGMM [8]	Continuous	✗	✗	✗
iCGMM (4.3)	✗	✗	✗	✗
DiffPool [154]	-	Adaptive	-	-
DGCNN [162]	✗	Topological	✗	✗
GAT [141]	✗	✗	✓	✗
R-GCN [147]	Discrete	✗	✗	✗
GMNN [60]	-	-	-	-
DGI [176]	✗	✗	✗	✓
GIN [109]	✗	✗	✗	✗
NMFPool [159]	-	Topological	-	-
SAGPool [183]	-	Adaptive	-	-
Top-k Pool [155]	-	Adaptive	-	-
FDGNN [140]	✗	✗	✗	✓
GMDN [9]	-	-	-	-

TABLE 3.2: Recap of DGNs' properties. When the symbol "-" is used, we mean "not applicable", as the row refers to a framework rather than a single model.

3.3 Scholarship Issues in Graph Classification [5]

Experimental reproducibility and replicability are core aspects of empirical machine learning. From time to time, researchers have warned their communities about flaws in scholarship regarding streams of scientific publications [184–186]. However, trying to correct bad practices does not take just one publication, rather a collective effort is required to acknowledge the current issues and take immediate action. Common examples of these troubling trends are the ambiguous or poorly detailed experimental settings, unfair comparison due to the use of different data features and/or data splits, cherry-picking of hyper-parameters on the basis of test set performances, and the impossibility of reproducing the results using the code provided by the authors. In turn, this means we are often unable to confidently assess which empirical methodology performs best on a given learning task.

The situation is not much different in the graph learning community. After the recent re-discovery of the core ideas and the subsequent exponential growth of scientific publications, it can be argued that little attention was devoted to ensure a fair and robust model assessment between models. A striking example can be found in some vertex classification benchmarks, where it was found that the use of different training/validation/test splits could completely alter the final performance ranking [177]. Indeed, in some papers, data splits were generated at random simply because there existed no common agreement on the evaluation criteria. Similarly, concerns have been raised about neural recommender systems, most of which cannot perform better than a very simple baseline [187].

This section describes our attempt to mitigate the lack of standardization of empirical comparisons in the **graph classification** scenario [5]. As a matter of fact, we observed that many practitioners did not provide thorough information about the two main steps of any machine learning evaluation, namely **model selection** and **risk assessment**. Failure to keep these phases well separated often leads to over-optimistic estimates of the generalization performances, but it also generates confusion and doubts among other researchers while building on previous results; this can easily mislead them into repeating the same methodological errors. Before continuing, let us briefly recall the basics of risk assessment and model selection.

Risk Assessment. To provide an estimate of the generalization performance of each model, a risk assessment procedure has to be followed. Risk assessment relies on a test set that must be used only after the chosen model configuration has been trained. If a test set is not given in advance, one can adopt a simple holdout split or, like we did, a *k-fold Cross*

Validation (CV) [188–190] scheme to generate k different training/test partitions (called **folds**) of the dataset. For each partition, we should perform an internal model selection procedure (based on the training set **only**) that picks the best hyper-parameters for *that specific partition*. This way, test data is **never** used to select the hyper-parameters, such as number of epochs, layers or hidden units. Note that, as model selection is performed independently for each fold, we obtain k different "best" configurations. This is why one should talk about the performance of the *class* of models rather than a single configuration. We would like to stress here that the best configuration overall does **not** exist because of the No Free Lunch Theorem [191].

Model Selection. Inside each fold, the selection of the best hyper-parameters usually happens via another holdout strategy or an inner k-fold CV, where this time the "outer" training data is further partitioned into training and validation sets (unless a validation set is already available). Because the best hyper-parameters' configuration is selected on the basis of validation performances, the key thing to remember is that these results are *biased* estimates of the true generalization capabilities of a model. Hence, it would be trivial to obtain state-of-the-art results by comparing models on validation performances: just find the configuration that maximizes the performance metric on the validation set. This is a bad practice we clearly want to avoid.

3.3.1 Chosen Criteria

Similarly to what has been done in [187], we first listed some relevant requirements for reproducibility: *i*) code for data preprocessing, model selection, and risk assessment is provided; *ii*) data splits are available; *iii*) data is split according to a stratification technique that preserves class proportions across all folds; *iv*) results are reported using standard deviations, and they refer to model evaluation (test set) rather than model selection (validation set).

We then selected the DGNs to re-evaluate according to basic principles: *i)* their graph classification performance obtained using a 10-fold cross validation; *ii)* peer-reviewed status; *iii)* architectural differences; *iv)* popularity. We ended up choosing DGCNN [162], DiffPool [154], ECC [148], GIN [109], and GraphSAGE [151], though the latter was not applied to graph classification tasks in the original paper. Table 3.3 summarizes our findings.

From the table, it seems that some of the most popular models from the literature did not meet all the listed criteria that would foster empirical reproducibility. Let us now expand the discussion about each model by highlighting the problems we found.

	DGCNN	DiffPool	ECC	GIN
Data preprocessing code	Y	Y	-	Y
Model selection code	N	N	-	N
Model evaluation code	Y	Y	-	Y
Data splits provided	Y	N	N	Y
Label Stratification	Y	N	-	Y
Report accuracy on test	Y	A	A	N
Report standard deviations	Y	N	N	Y

TABLE 3.3: Criteria for reproducibility considered in this work and their compliance among the considered models. (Y) indicates that the criterion is met, (N) indicates that the criterion is not satisfied, (A) indicates ambiguity (i.e. it is unclear whether the criteria is met or not), (-) indicates lack of information.

DGCNN In this paper, the architecture was fixed for all datasets. Although sub-optimal, learning rate and number of training epochs were tuned using only one of the 10 folds and then reused on all the other folds. We could not find the code to perform model selection despite the rest of it being publicly available. Moreover, the authors ran the 10-fold CV procedure 10 times[5] and reported the average of the 10 final scores, each of which had already been averaged over the 10 folds. As a result, the variance of the provided estimates was greatly reduced. This experimental setup, however, was different from the one used in other works, and thus we cannot reliably assess the variance of the models under the same setting.

DiffPool From both the paper and the provided code, it is unclear if reported results were obtained on the test set rather than the validation set. The authors stated that 10-fold CV was used, but standard deviations were not reported. There are some statements in the paper about applying early stopping on the validation set, but neither model selection code nor validation splits were made available. We also found that target stratification was not applied to the data splits and no random seed was set, hence we can assume the generated data splits were different each time the code was being executed.

ECC As in DiffPool, the paper lacks standard deviation values in the results. Likewise DGCNN, hyper-parameters were fixed in advance, hence it is not clear if and how model selection was performed. Importantly, there are no references in the code repository to data pre-processing, data stratification, data splitting, and model selection. This makes ECC the least reproducible model among those considered.

[5]This was computationally feasible since model selection is performed only once per CV.

GIN Here we observed another kind of troubling trend. The authors did a good job in listing the ranges of hyper-parameters tried. However, as stated explicitly in the paper and in the public review discussion, they report the mean *validation* accuracy of a 10-fold CV. In other words, the reported results refer to model selection and not to risk assessment. Furthermore, the code for model selection is not provided.

GraphSAGE This model is often used in other papers as a strong baseline [109, 154]. Nonetheless, the code to reproduce such experiments on graph classification has never been provided.

Summary It is this ample empirical inconsistency that has motivated a re-evaluation of these models within a rigorous, reproducible and fair environment. Our code has been publicly released alongside the data splits.[6]

3.3.2 Experimental Setting

The assessment of the above models is carried out on 9 graph classification datasets, four of which are chemical and five social. We considered D&D [192], PROTEINS [193], NCI1 [194] and ENZYMES [195] as binary classification chemical tasks, whereas IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-5K, and COLLAB [47] are social benchmarks. We report the statistics of these datasets in Table 3.4.

		# Graphs	# Classes	# Vertices	# Edges	# Vertex feat.
Chem.	D&D	1178	2	284.32	715.66	89
	ENZYMES	600	6	32.63	64.14	3+18
	NCI1	4110	2	29.87	32.30	37
	PROTEINS	1113	2	39.06	72.82	3
Social	COLLAB	5000	3	74.49	2457.78	-
	IMDB-BINARY	1000	2	19.77	96.53	-
	IMDB-MULTI	1500	3	13.00	65.94	-
	REDDIT-BINARY	2000	2	429.63	497.75	-
	REDDIT-5K	4999	5	508.82	594.87	-

Table 3.4: Dataset statistics. Following the literature, we use both the 18 continuous and 3 discrete vertex attributes in the case of ENZYMES. All other vertex features belong to a finite and discrete alphabet representing atom types.

As the reader can observe, the social datasets lack any kind of feature information about vertices or edges. For this reason, we will double the re-evaluations on the social tasks to consider two scenarios, that is, one in which vertex features hold a constant value 1

[6] https://github.com/diningphil/gnn-comparison.

and another in which the vertex degree is treated as the sole continuous feature. This way, we can study the effect of the inductive bias imposed by different realizations of a DGN's layer.

It is worth mentioning that in the past different choices for the vertex features have been made [109, 154], but the competing models were rarely compared under the same conditions.

Structure-agnostic Baselines The importance of proper baselines has been mostly underrated when it comes to DGNs. Yet, designing a good baseline is of paramount importance to discern between real and fallacious progress. In our specific case, testing whether structural information truly is meaningful for the task is more than just a double-check, as we will see. When DGNs performances closely match those of a structure-agnostic baseline, we can draw two conclusions: either the task does not need topological information to be solved or the models we have developed are not "powerful" enough. Whilst one may involve domain experts to check if the former conclusion is valid, the latter is more involved as multiple factors come into play, such as the amount of training data, the structural inductive bias we imposed through the architecture, and the hyper-parameters tried. On the contrary, a significant boost in performances can only indicate that the graph topology is relevant to solve the task.

Therefore, we adopted distinct baselines for the two families of datasets. On chemical datasets, with the exception of ENZYMES, we follow [43] and implement the Molecular Fingerprint technique. A Molecular Fingerprint is obtained by first applying a global *sum* aggregation $\mathcal{R}$, i.e., counting the occurrences of all atom types in the graph, followed by a single-layer MLP with ReLU activations that implements $\mathcal{T}_{out}$. Instead, on social domains and ENZYMES (due to the presence of additional features), we follow [116] and learn permutation-invariant functions over sets of vertices. This is done by first transforming the vertex features with a single-layer MLP, which are then aggregated via *sum* operator and passed to another single-layer MLP for the final classification. Hence, none of these baselines exploit the information contained in the adjacency matrix.

Setup and Hyper-parameters For the rest of the section, the evaluation setup shall consist of a 10-fold CV for risk assessment with a holdout model selection strategy inside each fold. This choice was made to keep the re-evaluation as close as possible to the procedure followed by most models. We also schematically represent it in Figure 3.8, where it becomes clearer how the computational requirements are proportional to k_{out} and the number of configurations tried for each model selection.

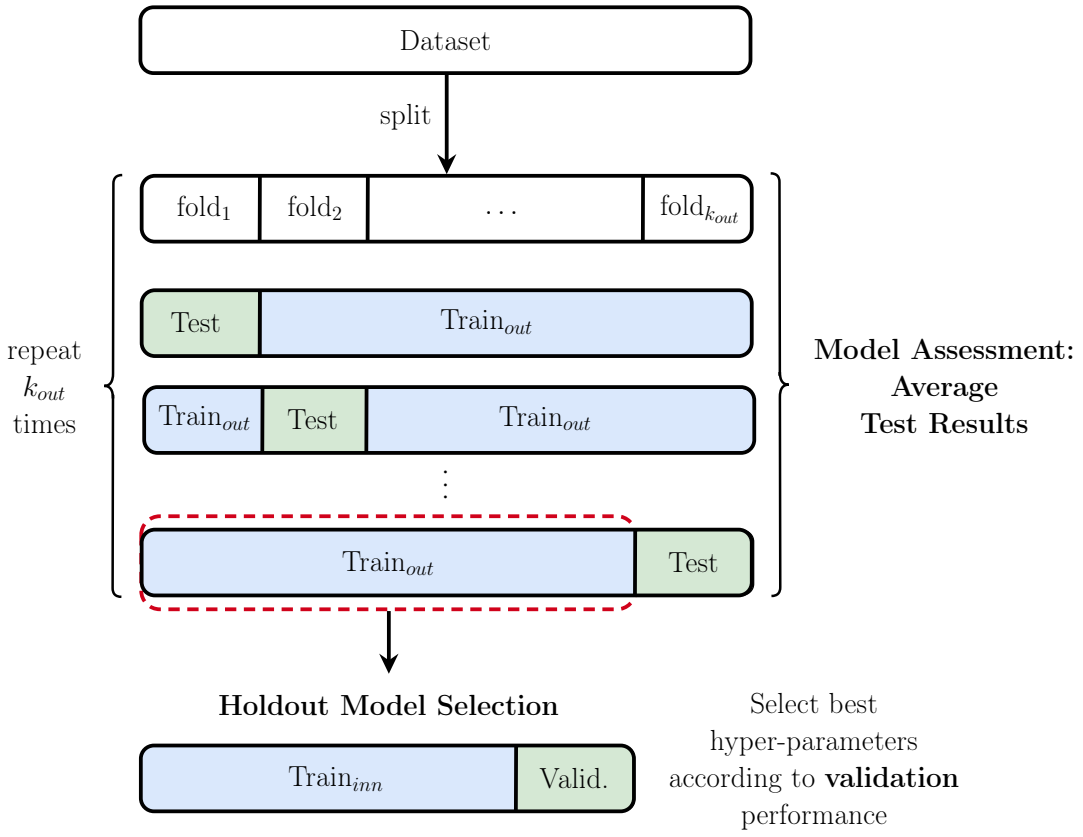


FIGURE 3.8: The evaluation framework used for the re-evaluation of some DNGNs consists of a 10-fold cross validation for risk assessment, where we carry out an *inner* holdout model selection **for each *outer* fold**.

The holdout strategy partitions the Train$_{out}$ set into Train$_{inn}$ and *Validation* sets (90% and 10% respectively). After every model selection, the best configuration for the outer fold is re-trained 3 times on Train$_{out}$, holding out a random subset of the data (10%) to perform early stopping. The separate training runs are needed to counteract the effect of an unfavorable random initialization on test performances. Finally, performances on the unseen test sets are averaged over the 3 final runs. We implemented early stopping [196] with patience n, meaning we stop training when the validation metric of interest (the accuracy), has not improved for n epochs. All data splits, with the exception of the random validation set in the final runs, have been precomputed, thus ensuring the models are cross-validated and assessed on the same data. Also, we applied a stratification strategy to ensure that class proportions are preserved. Table 3.5 summarizes what said so far.

Model selection relies on grid search, and the hyper-parameters' configurations for each model are shown in Table 3.6. We always included the hyper-parameters mentioned in the original papers, but we also di our best to ensure a fair comparison in terms of number of parameters and configurations to try.

Algorithm 1 Model Assessment (k-fold CV)

Input: Dataset $\mathcal{D}$, set of configurations Θ
Split $\mathcal{D}$ into k folds $F_1, \ldots, F_k$
for $i \leftarrow 1, \ldots, k$ **do**
    train_k, $\text{test}_k \leftarrow \left(\bigcup_{j \neq i} F_j\right)$, F_i
    $\text{best}_k \leftarrow \text{Select}(\text{train}_k, \Theta)$
    for $r \leftarrow 1, \ldots, R$ **do**
        $\text{model}_r \leftarrow \text{Train}(\text{train}_k, \text{best}_k)$
        $\text{p}_r \leftarrow \text{Eval}(\text{model}_k, \text{test}_k)$
    end for
    $\text{perf}_k \leftarrow \sum_{r=1}^{R} \text{p}_r / R$
end for
return $\sum_{i=1}^{k} \text{perf}_i / k$

Algorithm 2 Model Selection

Input: train_k, Θ
Split train_k into *train* and *valid*
$\text{p}_\theta = \emptyset$
for each $\theta \in \Theta$ **do**
    model $\leftarrow$ Train(train_k, θ)
    $\text{p}_\theta \leftarrow \text{p}_\theta \cup$ Eval(model, *valid*)
end for
$\text{best}_\theta \leftarrow \text{argmax}_\theta \text{ p}_\theta$
return best_θ

TABLE 3.5: Pseudo-code for model assessment (left) and model selection (right). "Select" refers to the model selection procedure, whereas "Train" and "Eval" represent training and prediction phases, respectively.

Computationally speaking, we had to run a very large number of experiments, which took months to complete. For each model, we tried a number of configurations ranging from 32 to 72, due to the varying number of hyper-parameters to select. The total effort amounted to more than 47000 training runs, which clearly required an extensive use of parallelism. We leveraged both multi-CPU and multi-GPU machines to complete these tasks in a reasonable amount of time. Nonetheless, training models such as ECC would have required more than 72 hours for a single training run on some social datasets. Allowing these models to complete their training would have dramatically slowed down the process; for these reason, due to the large amount of experiments to run and the limited amount of computational resources, we set a time limit of 72 hours to complete a single training run.

	Layers	Convs per layer	Batch size	Learning rate	Hidden units	Epochs	L2	Dropout	Patience	Optimizer	Scheduler	Dense dim	Embed. dim	Neighbors Aggregation
Baseline chemical	-	-	32 128	1e-1 1e-3 1e-6	32 128 256	5000	1e-2 1e-3 1e-4	-	500, loss 500, acc	Adam	-	-	-	sum
Baseline IMDB	-	-	32 128	1e-1 1e-3 1e-6	32 128 256	3000	1e-2 1e-3 1e-4	-	500, loss 500, acc	Adam	-	-	-	sum
Base. COLLAB and REDDIT	-	-	32 128	1e-1 1e-3	32 128	3000	1e-2 1e-3 1e-4	-	500, loss 500, acc	Adam	-	-	-	sum
Baseline ENZYMES	-	-	32	1e-1 1e-3 1e-6	32 64 128 256	5000	1e-2 1e-3 1e-4	-	1000, loss 1000, acc	Adam	-	-	-	sum
DGCNN	2 3 4	1	50 (cpu) 16 (gpu)	1e-4 1e-5	32 64	1000	-	0.5	500, loss 500, acc	Adam	-	128	-	mean
DiffPool	1 2	3	20 (cpu) 8 (gpu)	1e-3 1e-4 1e-5	32 64	3000	-	-	500, loss 500, acc	Adam	-	50	64 128	mean
ECC	1 2	3	32 (cpu) 8 (gpu)	1e-1 1e-2	32 64	1000	-	0.05 0.25	500, loss 500, acc	SGD	ECC-LR	-	-	sum
GIN	see hidden units	1	32 128	1e-2	32 (5 layers) 64 (5 layers) 64 (2 layers) 32 (3 layers)	1000	-	0 0.5	500, loss 500, acc	Adam	Step-LR (step: 50, gamma: 0.5)	-	-	sum
GraphSAGE	3 5	1	32 (cpu) 16 (gpu)	1e-2 1e-3 1e-4	32 64	1000	-	-	500, loss 500, acc	Adam	-	-	-	mean max sum

TABLE 3.6: Hyper-parameters tried during each grid-search model selection.

3.3.3 Results

We present the results on chemical and social benchmarks in Tables 3.7 and 3.8, respectively. Some observations can be made: first of all, none of the DGNs seem to improve over the performance of the structure-agnostic baseline on three out of four chemical datasets. On the other hand, the baseline cannot reach the same performance of DGNs on the NCI1 dataset. To confirm that this is not due to the under-parametrization of the baseline, we trained a configuration with 10000 hidden units and no regularization. The training accuracy reached a modest 67%, whereas a DGN like GIN can easily overfit the training set. This provides unambiguous evidence that structural information is actually relevant for the task. In social datasets, the addition of vertex degrees makes the baseline very competitive w.r.t. most models, but the GIN model has the best accuracy scores in almost all social tasks.

	D&D	**NCI1**	**PROTEINS**	**ENZYMES**
Baseline	**78.4** $\pm$ 4.5	69.8 $\pm$ 2.2	**75.8** $\pm$ 3.7	**65.2** $\pm$ 6.4
DGCNN	76.6 $\pm$ 4.3	76.4 $\pm$ 1.7	72.9 $\pm$ 3.5	38.9 $\pm$ 5.7
DiffPool	75.0 $\pm$ 3.5	76.9 $\pm$ 1.9	73.7 $\pm$ 3.5	59.5 $\pm$ 5.6
ECC	72.6 $\pm$ 4.1	76.2 $\pm$ 1.4	72.3 $\pm$ 3.4	29.5 $\pm$ 8.2
GIN	75.3 $\pm$ 2.9	**80.0** $\pm$ 1.4	73.3 $\pm$ 4.0	59.6 $\pm$ 4.5
GraphSAGE	72.9 $\pm$ 2.0	76.0 $\pm$ 1.8	73.0 $\pm$ 4.5	58.2 $\pm$ 6.0

Table 3.7: Results on chemical datasets with mean accuracy and standard deviation are reported. Best average performances are highlighted in bold.

		IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
No Features	Baseline	50.7 $\pm$ 2.4	36.1 $\pm$ 3.0	72.1 $\pm$ 7.8	35.1 $\pm$ 1.4	55.0 $\pm$ 1.9
	DGCNN	53.3 $\pm$ 5.0	38.6 $\pm$ 2.2	77.1 $\pm$ 2.9	35.7 $\pm$ 1.8	57.4 $\pm$ 1.9
	DiffPool	68.3 $\pm$ 6.1	45.1 $\pm$ 3.2	76.6 $\pm$ 2.4	34.6 $\pm$ 2.0	67.7 $\pm$ 1.9
	ECC	67.8 $\pm$ 4.8	44.8 $\pm$ 3.1	OOR	OOR	OOR
	GIN	66.8 $\pm$ 3.9	42.2 $\pm$ 4.6	**87.0** $\pm$ 4.4	**53.8** $\pm$ 5.9	**75.9** $\pm$ 1.9
	GraphSAGE	**69.9** $\pm$ 4.6	**47.2** $\pm$ 3.6	86.1 $\pm$ 2.0	49.9 $\pm$ 1.7	71.6 $\pm$ 1.5
With Degree	Baseline	70.8 $\pm$ 5.0	**49.1** $\pm$ 3.5	82.2 $\pm$ 3.0	52.2 $\pm$ 1.5	70.2 $\pm$ 1.5
	DGCNN	69.2 $\pm$ 3.0	45.6 $\pm$ 3.4	87.8 $\pm$ 2.5	49.2 $\pm$ 1.2	71.2 $\pm$ 1.9
	DiffPool	68.4 $\pm$ 3.3	45.6 $\pm$ 3.4	89.1 $\pm$ 1.6	53.8 $\pm$ 1.4	68.9 $\pm$ 2.0
	ECC	67.7 $\pm$ 2.8	43.5 $\pm$ 3.1	OOR	OOR	OOR
	GIN	**71.2** $\pm$ 3.9	48.5 $\pm$ 3.3	**89.9** $\pm$ 1.9	**56.1** $\pm$ 1.7	**75.6** $\pm$ 2.3
	GraphSAGE	68.8 $\pm$ 4.5	47.6 $\pm$ 3.5	84.3 $\pm$ 1.9	50.0 $\pm$ 1.3	73.9 $\pm$ 1.7

Table 3.8: Results on social datasets with mean accuracy and standard deviation are reported. Best average performances are highlighted in bold. OOR means Out of Resources, either time ($>$ 72 hours for a single training) or GPU memory.

From these results, it is clear how structure-agnostic baselines represent an essential tool to understand the impact of using DGNs. But we can extract further insights too: since

structural features are known to correlate with molecular properties [197], it is possible that the actual DGNs are still not able to extract what is needed to solve the above chemical tasks. Also, the relatively high standard deviations should suggest caution when arguing that a model performs better than another because of small (averaged) performance gains. It is highly likely, in fact, that such performance fluctuations are caused by random initializations, rather than being actual empirical progress.

It is also interesting to see how the addition of a simple feature like the vertex degree is able to provide significant performance gains on the social datasets. Indeed, the baseline provides from 10% to 20% better accuracy with this kind of information, and it even achieves state of the art results on IMDB-BINARY. As regards DGNs, instead, the effect of the degree seems to be less relevant, which is reasonable since the first layer can (in principle) compute the degree by summing neighboring features. One notable exception is DGCNN, which explicitly needs the degree as a vertex feature to improve the performances. Last but not least, the addition of this feature produces completely different rankings, much alike what happened in [177]. This demonstrates how important it is to compare different methods while using the same set of features.

Since the degree of a vertex can be computed with a simple sum-based neighborhood aggregation, we compare the median "best" number of layers chosen across the 10 different folds in the two social settings. Results are reported in Table 3.9. There exists a general trend, with the exception of GraphSAGE, in which the best number of layers is reduced by approximately 1 when we add the degree feature. Therefore, our intuitive reasoning about the inductive bias of DGNs architectures seems supported by evidence.

	IMDB-B		**IMDB-M**		**REDDIT-B**		**REDDIT-M**		**COLLAB**	
	1	**DEG**	**1**	**DEG**	**1**	**DEG**	**1**	**DEG**	**1**	**DEG**
DGCNN	3	3	3.5	3	4	3	3	2	4	2
DiffPool	1	2	2	1	2	2	2	1	2	1.5
ECC	1	2	1	1	-	-	-	-	-	-
GIN	3	2	4	2	4	4	4	3	4	4
GraphSAGE	4	3	5	4	3	4	3	5	3	5

TABLE 3.9: We report the median number of selected layers per model, depending on whether vertex degrees are used as input features or not. A "1" indicates that an uninformative feature is used as the vertex label.

Finally, to show that our estimates are actually much lower than what has been reported in the literature, we visually compare our averaged values with those of the original papers. In addition, we plot the best validation scores averaged across the 10 different model selections, so that we can clearly see how far from the (empirical) truth we can get

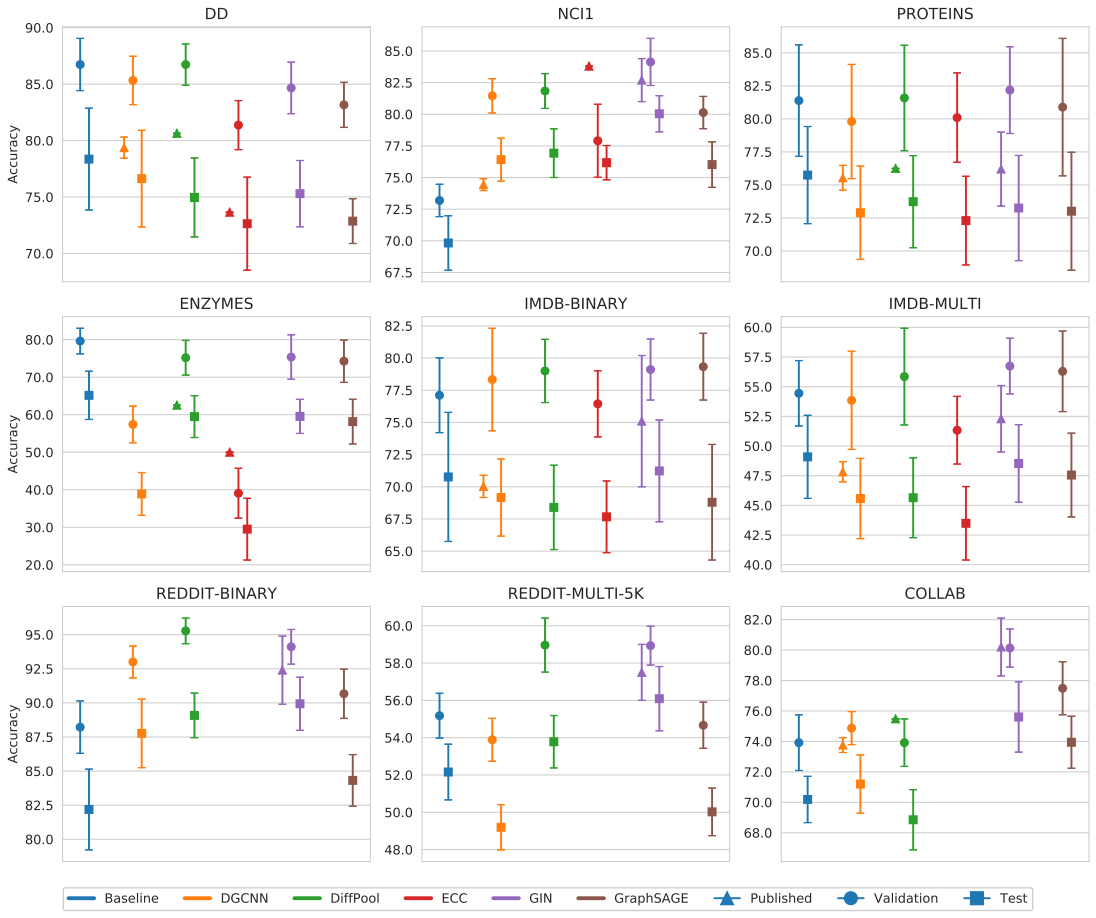


FIGURE 3.9: Chemical and social (with degree) benchmark results are shown together with published results (when available). For each of them, we report validation and test accuracy of the evaluated models, together with published results (when available).

when reporting validation scores. Figure 3.9 confirms that the gap between the validation and test estimates is usually consistent, with validation scores overestimating the true generalization performances of the model.

To conclude this section, we briefly mention that subsequent effort has been made to put together larger datasets [198, 199] and standardize the evaluation process[7]. Open Graph Benchmark [198] is a collection of graph-related tasks, each with its own evaluation process with fixed performance metrics. In another work [199], larger benchmarks related to chemistry, the travel-salesman problem, and image classification are proposed, together with an assessment of some models taken from the literature. It should be noted that in [199] the numbers are approximated estimates of the generalization performances of each model, as a proper model selection has not been carried out due to time constraints. Therefore, it could be unclear whether subsequent improvements w.r.t. those numbers will be caused by the model selection itself or by the actual improvement of a particular architecture over others.

[7] https://github.com/diningphil/PyDGN.

3.4 Application to Molecular Biosciences [10]

Now that we have discussed the building blocks of deep learning for graphs as well as our attempt to tackle some of its scholarship issues, we shall provide an example of a practical application from the field of molecular biosciences [10], more specifically related to molecular dynamics.

Molecular dynamics simulations [200, 201] are a very useful tool when it comes to investigate properties of matter. Classical all-atom simulations have allowed researchers to ultimately understand a large variety of physical systems, from metals and fluids to biological entities like proteins. As these systems grow larger, the computational costs and the intuitive understanding of the systems' behavior become increasingly challenging. In the soft and biological matter field, **coarse-graining** methods provide ways to extract relevant properties of a macro-molecular system [202–205]. To do so, the system is first "simplified" into a higher-level representation where the constituent units are called coarse-grained **sites**.

Defining a coarse-grained representation requires two things: first, a **mapping** from the units of the original system to the coarse-grained sites; second, the set of effective **interactions** between the sites, so that we can reproduce a posteriori the emergent properties of the original system through this simplified representation. Figure 3.10 depicts one such example. While there has been a substantial research effort in defining coarse-grained potentials [206–208], the study of the mapping itself has been less investigated. Sites are often selected on the basis of chemical or physical criteria that do not take into account the local and global environment of each constituent in the original system [209].

Nevertheless, this approach has an evident limitation: any coarse-grained process implies some degree of information loss, so it would be appropriate to **automatically** find the

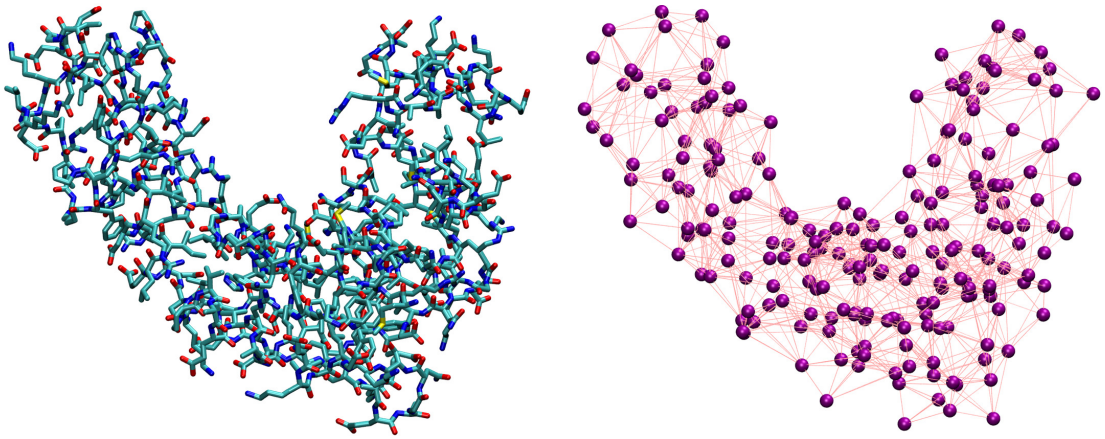


FIGURE 3.10: Comparing an all-atom system (a protein, left) with one of its possible coarse-grained representations (right). Purple vertices are coarse-grained sites.

mapping that minimizes the loss of information about the system's overall behavior. There have been attempts to solve via graph-theoretical analyses [210], geometric criteria [211] and machine learning [212–214]. The underlying idea of all these work is that the optimal coarse-grained representation can be found in a subset of the original features. In addition, there are statistical mechanics-based strategies that address the issue by means of minimization of the so-called **mapping entropy** S_{map} [215], a measure of the dissimilarity between the probability density of the original configuration and that of the lower-resolution description. [208, 216–218].

The main shortcomings of mapping entropy minimization are the costs of computing the S_{map} of a single coarse-grained representation as well the combinatorial size of the search space. Therefore, in this section, we propose to train a Deep Graph Network that predicts the mapping entropy associated with a specific coarse-grained representation of a given protein. If we managed to achieve good performances, we could incorporate the much more efficient DGN into the Wang-Landau enhanced sampling algorithm [219–222] so as to carry out a quasi-exhaustive exploration of a biomolecule's mapping space.

3.4.1 Datasets

The evaluation focuses on two proteins called *6d93* and *4ake*, extracted from [215]. The former is a mutant of *tamapin*, a toxin of the Indian red scorpion, whereas the latter is the open conformation of the *adenylate kinase*, an enzyme inside the cell. A schematic representation of both proteins is shown in Figure 3.11. The task is a *regression* problem in which, given a protein and a specific choice for the mapping, we need to predict the associated mapping entropy. To build the dataset, we first represent each protein as a graph, where vertices encode heavy atoms and edges connect pairs of atoms whose atomic distance is closer than $1nm$ in the reference structure. We incorporate a number of binary properties into each vertex's features, which are described in Table 3.10, whereas edge features consist of a single continuous value encoding the inverse atomic distance. A schematic representation of a protein as a graph, with different mappings and therefore S_{map} values, is shown in Figure 3.12

The samples in the dataset are constructed by taking the **same** graph representation of the protein and changing the binary attribute "Site" depending on the coarse-grained configuration we want to represent. If an atom is selected as a site, then the attribute is set to 1 and 0 otherwise. Note that we retain the atoms that are not selected by the coarse-grained configuration: the underlying idea is to make the DGN learn the relation between the sites' position in the protein and the mapping entropy value. To find a good estimate for the target value, we carried out expensive all-atom simulations on these

Feature name	Description
C	Carbon atom
N	Nitrogen atom
O	Oxygen atom
S	Sulphur atom
HPhob	Part of a hydrophobic residue
Amph	Part of a amphipathic residue
Pol	Part of a polar residue
Ch	Part of a charged residue
Bkb	Part of the protein backbone
Site	**Atom selected as a CG site**

TABLE 3.10: A list of the binary features used to describe the properties of each atom in the protein representation.

Protein	CPU time	Walltime	Single measure
6d93	40.7 days	2.55 days	$\simeq$ 2.1 mins
4ake	153.9 days	3.20 days	$\simeq$ 8.0 mins

TABLE 3.11: Computational costs of all-atom simulations and mapping entropy calculations for the two investigated proteins. *CPU time* (respectively *Walltime*) represents the time (user time) necessary to simulate 200ns. *Single measure* is the amount of time that is required to compute, on a single core, the S_{map} of a given mapping.

proteins. We ended up with 4968 and 1968 labeled samples, and we summarize other dataset statistics in Table 3.12. The distribution of the target values for both datasets is such that there is negligible overlap between the random and optimized[8] mappings, meaning that the best S_{map} values cannot be reached by a mere random exploration of

[8]Using a simulated annealing approach [215].

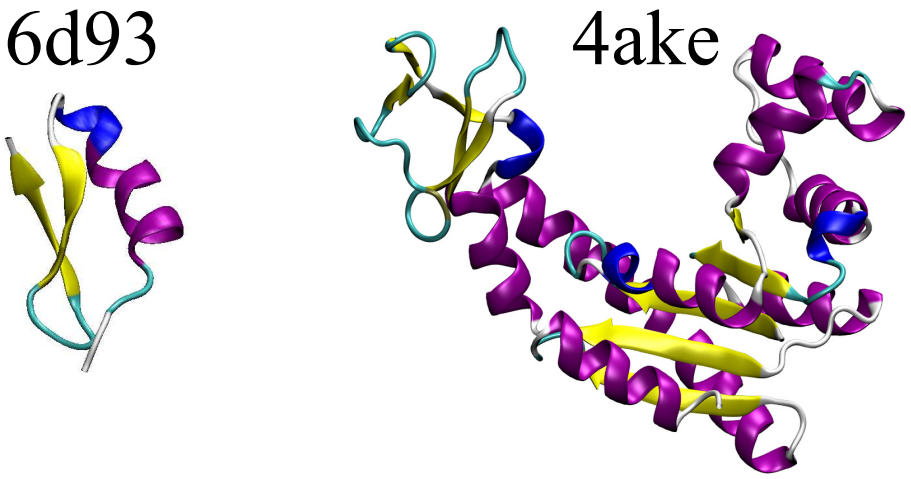


FIGURE 3.11: The *tamapin*, a.k.a. *6d93*, and the open conformation of *adenylate kinase*, a.k.a. *4ake*. Though smaller, *6d93* possesses all the elements of proteins' secondary structures. On the other hand, *4ake* is larger and has has a much wider structural variability.

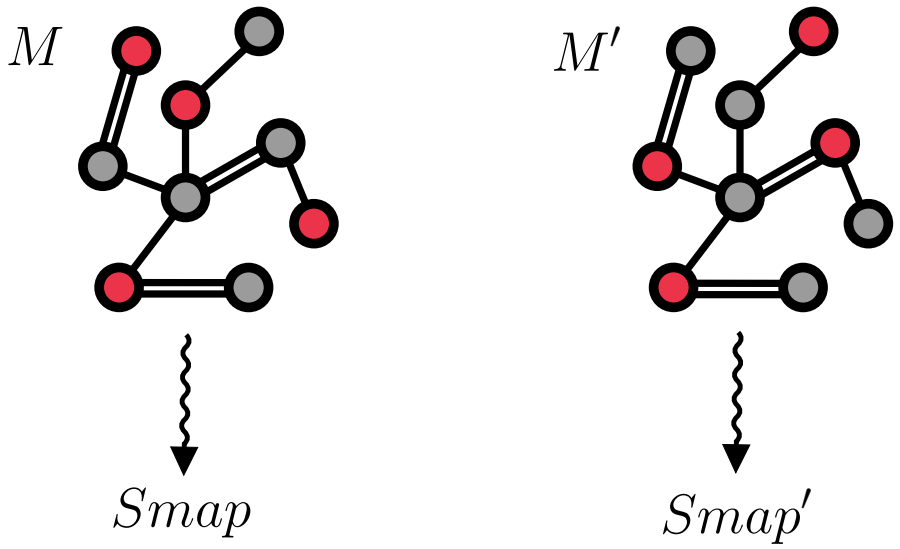


FIGURE 3.12: Each protein is converted to a graph, where vertices hold atomic features and the "site" information (in red). For the same protein and different mappings, we seek to predict the corresponding mapping entropies.

the mapping space. Moreover, the mapping entropy is proportional to the system's size, as the lower bound in mapping entropy of *4ake* (≈ 90) is almost one order of magnitude higher than that of *6d93* (≈ 10). Computationally speaking, Table 3.11 reports that the mapping entropy calculations of a single coarse-grained configuration can take up to 8 minutes for the larger protein *4ake*.

Protein	Vertices	Edges	Avg. Degree	Dataset Size
6d93	230	21474	93	4968
4ake	1656	207618	125	1968

TABLE 3.12: Dataset statistics.

3.4.2 Experimental Setting

We experiment with a structure-agnostic baseline like the one introduced in Section 3.3 for social tasks and an edge-aware DGN. The neighborhood aggregation extends that of [109] as follows:

$$\boldsymbol{h}_v^{\ell+1} = MLP^{\ell}\Big(\big(1+\epsilon^{\ell}\big) * \boldsymbol{h}_v^{\ell} + \sum_{u \in \mathcal{N}_v} \boldsymbol{h}_v^{\ell} * a_{uv}\Big),$$

where $*$ denotes element-wise scalar multiplication, $\epsilon^{\ell} \in \mathbb{R}$ is an adaptive weight of the model, and a_{uv} is the inverse atomic distance between atoms. Practically speaking, we want to penalize the contribution of neighbors that are farther away according to the protein topology. Then, we apply a **site-aware** readout function that learns to weight the

contribution of site (w_s) and non-site (w_n) atoms belonging to the disjoint sets $\mathcal{V}_g^s \subset \mathcal{V}_g$ and $\mathcal{V}_g^s \subset \mathcal{V}_g$:

$$\hat{S}_{map} = \mathbf{w}_{out}^T \Big(\sum_{u \in \mathcal{V}_g^s} \big([\boldsymbol{h}_u^1, \ldots, \boldsymbol{h}_u^L] * w_s\big) + \sum_{u \in \mathcal{V}_g^n} \big([\boldsymbol{h}_u^1, \ldots, \boldsymbol{h}_u^L] * w_n\big) \Big),$$

where L is the chosen number of layers, $\mathbf{w}_{out} \in \mathbb{R}^{K*L}$ is a vector of parameters to be learned, and square brackets denote concatenation of the different vertex states computed at different layers.

To assess the performance of each model, we first split the dataset into training, validation and test realisations, following an 80%/10%/10% hold-out strategy. During model selection. we applied early stopping to select the training epoch with the best validation score, and the chosen model was evaluated on the unseen test set. The evaluation metric for our regression problem is the coefficient of determination (or R^2-score); this score ranges from $-\infty$ (worst predictor) to 1 (best predictor).

For the purpose of this application, and due to the computational costs necessary to train a DGN on these datasets, we opted for selecting the hyper-parameters via a manual experimental screening on the validation set performances. Eventually, we chose a DGN depth of $L = 5$, and we implemented each MLP as a one-layer feed-forward network with $K = 64$ hidden units followed by an element-wise rectifier linear unit (ReLU) activation function [223]. The loss function was the Mean Absolute Error (MAE). The optimization algorithm was Adam [224] with a learning rate of 0.001 and no regularization. We trained for a maximum of 10000 epochs with early stopping patience of 1000 epochs and mini-batch size 8, accelerating the training using a GPU with 16GB of memory. Instead, we chose $K = 1024$ hidden units for the baseline while keeping the rest unchanged.

The subsequent exploration of the mapping space is carried out with the Wang-Landau sampling scheme. The parameters governing the sampler are the result of previous studies and expert knowledge [215], and they do not influence the training of the DGN. Therefore, in the interest of readability, we refer the reader to [10] for a thorough description of the whole sampling process as well as the dataset-specific parameters used to explore the mapping space.[9]

3.4.3 Results

We start by looking at the prediction performances of the aforementioned models. Table 3.13 reports the R^2 score and MAE in training, validation and test. While the baseline

[9] https://github.com/CIML-VARIAMOLS/GRAWL.

provides a surprisingly high score on *6d93*, we also observe that the DGN has much better performances on both datasets. Indeed, it achieves extremely low values of MAE for *6d93*, with an R^2 score higher than 0.95 in all cases. The model performs slightly worse in the case of *4ake*: the result of $R^2 = 0.84$ on the test set is still acceptable, although the gap with the training set ($R^2 = 0.92$) is non-negligible.

Model / Protein	TR MAE	TR R^2	VL MAE	VL R^2	TE MAE	TE R^2
Baseline / *6d93*	0.55	0.86	0.63	0.83	0.65	0.82
DGN / *6d93*	0.13	0.99	0.33	0.95	0.33	**0.96**
Baseline / *4ake*	1.78	0.70	1.75	0.65	1.86	0.69
DGN / *4ake*	0.91	0.92	1.2	0.85	1.35	**0.84**

TABLE 3.13: Mapping entropy prediction results on the training (TR), validation (VL) and test (TE) sets for the two analysed proteins. We display both the R^2 score and the mean average error (MAE, $kJ/\text{mol}/K$).

In Figure 3.13, we plot predicted values for training and test samples against their ground truth. Ideally, a perfect result would correspond to the points lying on the diagonal dotted line. As regards *6d93*, we can get pretty close to the true training and test targets. The deviation from the perfect fit becomes wider for *4ake*, but there are no relevant outliers to report, a good sign of the DGN's generalization performances. By closely inspecting the *4ake* scatter plot, we observe that the DGN slightly overestimates the mapping entropy of optimized coarse-grained samples, i.e., $S_{map} \lesssim 100\ kJ/\text{mol}/K$. Likewise, the opposite is true for $S_{map} \gtrsim 100\ kJ/\text{mol}/K$, with random coarse-grained mappings being slightly underestimated.

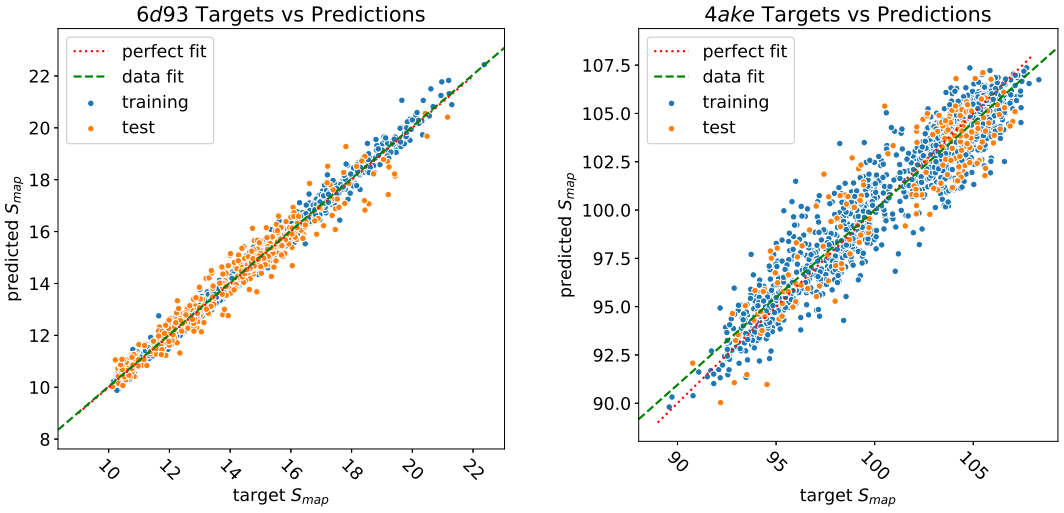


FIGURE 3.13: Scatter plot of predictions against the ground truth for both datasets.

The dissimilarity in performance between the two proteins is not surprising, given expert knowledge about their nature. In fact, we already mentioned how *adenylate kinase* is larger and more complex than the *tamapin* mutant. The datasets sizes are necessarily

very different due to the heavy computational requirements to label *4ake* samples. It is then natural to expect that training a model with excellent generalization performance on *4ake* would be harder than the other task. We would like to emphasize, however, that a completely adaptive DGN was able to approximate, in both structures, the long and computationally intensive algorithm for estimating the mapping entropy [215]. Even more significant is the fact that the model relies on a combination of static structural information and a few simple vertex features. In other words, the DGN operates in the absence of direct knowledge about the complex dynamical behavior of the two systems, in contrast to the onerous molecular dynamics simulations.

To confirm that the DGN provides a computational advantage with respect to the simulations, we report in Table 3.14 the time required to predict a single S_{map} output and compare it to the ground truth algorithm. To provide a fair comparison between the algorithm of [215], which relies on a CPU machine, we compute prediction times on both CPU and GPU. Overall, we can see that the DGN inference phase is $2-5$ orders of magnitude faster than the original algorithm, depending on the hardware used. Notably, the speedup is associated with a fairly good predictive accuracy of the machine learning model. To sum up, such drastic speedup of the trained model allows us to carry out a much wider exploration of the S_{map} landscape of both protein systems.

Protein	Single measure	Inference GPU (CPU)	Improvement GPU (CPU)
6d93	$\simeq$ 2.1 mins	$\simeq$ 0.9(98.7) ms	$\simeq 140000 \times (1276\times)$
4ake	$\simeq$ 8.0 mins	$\simeq$ 4.8(1103.2) ms	$\simeq 100000 \times (435\times)$

TABLE 3.14: Time comparison between the original mapping entropy algorithm and the inference phase of the DGN.

If we embed the trained DGN in the Wang-Landau sampler, we can better approximate the distribution of the mapping entropy values for *6d93* and *4ake*. Put differently, we can better estimate how many coarse-grained representations (sampled from the mapping space of each protein) exhibit a specific amount of information loss with respect to the all-atom system. To reach convergence of the sampling protocol, we had to probe approximately 4.8×10^6 and 3×10^7 different mappings for *6d93* and *4ake*, respectively. Clearly, such an extensive sampling was made feasible by the speedup attained by the proposed DGN.

The distributions $P(S_{map})$ of both *6d93* and *4ake* are shown in Figure 3.14. In the former case, the sampling scheme produces a probability density that is fully compatible with the (normalized) histograms of the target values. Also, notice that the statistical weight

of the optimized mappings here is negligible, but nonetheless this result is definitely non-trivial, as it proves that the trained DGN of *6d93* is not in an overfitting regime and can predict the correct population of the true mapping entropy landscape.

As regards *4ake*, the agreement between the two curves presented is still remarkable but not as precise as before. The slight mismatch is understandable if we consider the above regression scores: the DGN tends to underestimate (respecrively overestimate) the mapping entropy associated with random (optimized) coarse-grained representations.

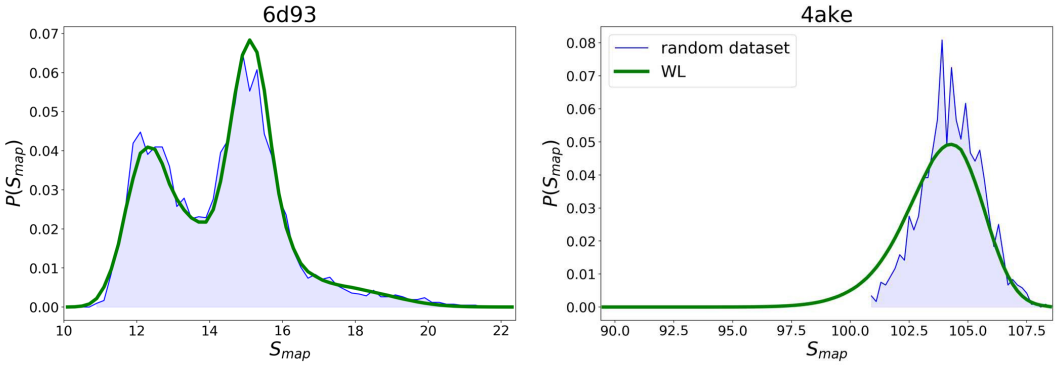


FIGURE 3.14: Comparing probability densities $P(S_{map})$ for the two proteins. The smooth distribution produced by the DGN (green lines) is similar to that generated by a random sampling of mappings (blue areas). S_{map} values are in $kJ/\text{mol}/K$. WL here stands for Wang-Landau.

3.5 Summary

In this chapter, we have described the basic building blocks of deep learning methodologies able to learn from graph-structured data. In doing so, we managed to define different methods under the same uniform mathematical notation, so that we could easily understand peculiarities among the most popular neighborhood aggregation mechanisms in the literature as well as their nature [1, 41]. In turn, this also highlighted how the main underlying mechanisms had been already there for more than 10 years, in spite of the recent wave of re-discovery caused by the growing interest in the field. However, such an incredibly rapid stream of publications lacked a standardized, fair and robust experimental procedure in both vertex [177] and graph classification tasks, so we provided an *empirical* re-evaluation of some of the most known works across a substantial number of benchmarks [5]. To convince the reader that these approaches are truly useful when applied to real-world problems, an application to the field of molecular biosciences was presented, in which we demonstrated how we can study a molecular system by quickly finding solutions with minimum mapping entropy [10].

It is now time to move to the core topics of this thesis. We shall be dealing with a deep, fully probabilistic framework to learn from graphs with varying topology. We will use the knowledge acquired in the previous chapters to simplify the exposition and focus on the technical details that position such framework into the family of Deep Bayesian Graph Networks.

Chapter 4

Deep Bayesian Graph Networks

Lo mio maestro allora in su la gota
destra si volse indietro, e riguardommi;
poi disse: «Bene ascolta chi la nota».

Inferno - Canto XV

Deep Bayesian Graph Networks are fully probabilistic models for graphs whose architecture implements the principles of local and iterative computation described in the previous chapter. We will commence the discussion with the Contextual Graph Markov Model (CGMM), a model bridging the gap between the NN4G [107] and the recursive Bottom-up Hidden Tree Markov Model [225]. We shall show how the neighborhood aggregation can be formalized and handled in a deep Bayesian framework, and we will evaluate its effectiveness on vertex and graph classification tasks. Then, we will extend CGMM to the processing of arbitrary edge features. The resulting model, called Extended CGMM (E-CGMM), uses an additional Bayesian network to model the generation of edge features, and its functioning is deeply intertwined with the original CGMM's graphical model. E-CGMM exhibits a form of dynamic neighborhood aggregation that contributes to the better performances of the model on graph classification, graph regression, and link prediction tasks. The third and last methodological contribution is the Infinite Contextual Graph Markov Model (ICGMM), which extends CGMM to the Bayesian nonparametric setting using an HDP. ICGMM is capable of automatically selecting, on the basis of the available data, almost all CGMM's hyper-parameters, including the number of latent states at each layer. Empirically, we will bring evidence that ICGMM has comparable or better performances than the original model while drastically reducing the size of the graph embeddings. We conclude the chapter with a real-world malware detection application that exploits the above models.

4.1 The Contextual Graph Markov Model [6, 7]

The core contributions of this thesis, which we are about to present, are inspired by both the probabilistic topics of Chapter 2 and the underlying principles of Deep Graph Networks developed in Chapter 3. For the rest of the chapter, we shall therefore depart from purely neural architectures and focus on a novel probabilistic framework to learn representations of graphs or vertices.

This section is devoted to the introduction of the Contextual Graph Markov Model (CGMM) [6, 7]. As in the previous chapter, we shall adopt a top-down approach and first list the four main characteristics of the $\mathcal{T}_{enc}$ mapping:

- **Unsupervised**. The model relies on the maximization of an unsupervised learning criterion, that is, the likelihood of the graph's entities, to adjust its parameters and construct vertex/graph embeddings. In principle, this means that the model can exploit large amounts of unlabelled data to produce richer vertex/graph embeddings on a given domain.
- **Fully Probabilistic**. Contrarily to other methods, which formalize the learning objective in probabilistic terms but approximate probability distributions with neural networks [60], CGMM relies on Bayesian networks to capture the latent factors of vertex features. This makes CGMM a fully probabilistic model and requires, as we will see, a completely probabilistic formulation of the neighborhood aggregation previously discussed.
- **Deep (Constructive)**. Following the principles of Deep Graph Networks, CGMM is a deep feedforward architecture, where each layer is a distinct Bayesian network. This is enough to distinguish CGMM from SRL approaches or recursive Bayesian networks for trees [118, 122, 129], where the structure is taken into account in the formalization of the probabilistic model rather than by the message passing scheme of DGNs. In addition, and similarly to NN4G [107], the model is built in a constructive fashion by training one layer at a time. Once a CGMM layer has been trained, it is *frozen* and never modified again.
- **Efficient**. Last but not least, the model has the same asymptotic complexity as most DNGNs, being linear in the number of edge. Therefore, the model is amenable to large scale graph processing.

These characteristics make CGMM a rather peculiar approach in the landscape of Deep Graph Networks. To show the richness of the extracted graph embeddings, we will use them in combination with a neural readout to tackle vertex and graph classification tasks.

4.1.1 Layer Definition

We have already seen how each layer ℓ of a DGN is responsible for the creation of intermediate vertex representations $\boldsymbol{h}_u^{\ell+1}$. Likewise, each CGMM's probabilistic layer assumes that the generation of vertex' features $\boldsymbol{x}_u$ depends on some **latent factor** that we would like to capture. Hereinafter, for the purposes of this thesis, we will assume to deal with a single discrete or continuous feature x_u.

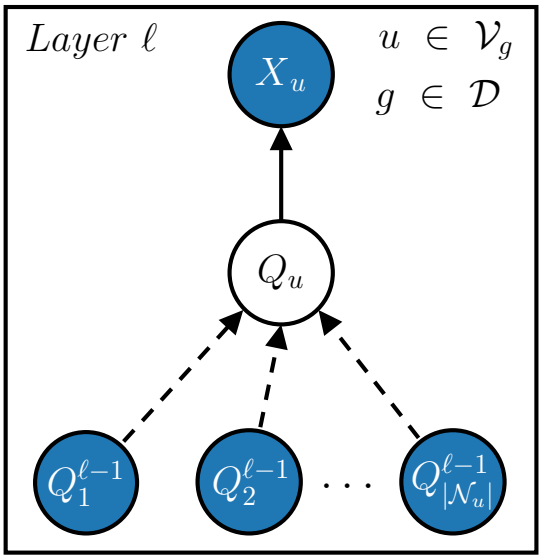


FIGURE 4.1: Graphical model of a generic layer ℓ of CGMM. Dashed arrows denote the flow of contextual information coming from previous layers.

We present the Bayesian network of a generic CGMM layer ℓ in Figure 4.1. This is a **conditional mixture model** where we formally associate each vertex feature with an observable variable X_u, whose **adaptive emission** distribution $P(X_u \mid Q_u)$ is conditioned on the latent **categorical** variable Q_u with C attainable values.

Intuitively, the latent variable Q_u plays the role of the hidden state $\boldsymbol{h}_u^{\ell+1}$ in the general formulation devised in the previous chapter; that said, we follow the notational convention of Chapter 2 when referring to random variables.

At first, it may seem that a strong assumption has been imposed here, i.e., all vertices are i.i.d.. In other words, we are completely disregarding the structural dependency between the set $\{X_u \mid u \in \mathcal{V}_g\}$ of variables. However, having already outlined the basic principles of Deep Graph Networks, it should be clearer why this assumption works well: structural information has been encoded into the neighboring observable variables $\boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1} = \{Q_1^{\ell-1}, \ldots, Q_{|\mathcal{N}_u|}^{\ell-1}\}$ computed at the previous layer. This is why, with slight abuse of notation, dashed arrows in the figure indicate that there is contextual information

flowing from the previous (frozen) layer $\ell - 1$. We will also use the symbol $\boldsymbol{q}_v^{\ell-1}$ to refer to the categorical distribution over the C states (i.e., a vector) inferred for the *latent* variable $Q_v^{\ell-1}$ when training the previous layer. In addition, the j-th component of such distribution shall be $q_v^{\ell-1}(j)$.

Formally, we can define the likelihood of a graph g at layer ℓ as

$$\mathcal{L}(\boldsymbol{\theta} \mid g) = P(g \mid \boldsymbol{\theta}) = \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} \underbrace{P_{\boldsymbol{\theta}}(X_u = x_u | Q_u = i)}_{emission} P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1}). \tag{4.1}$$

As in standard mixture models, we have introduced the latent variable Q_u in the equation via marginalization. However, not knowing the size of $\mathcal{N}_u$ for each vertex u makes the definition of the posterior distribution $P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1})$ quite hard to formalize, and conditioning on all neighboring states becomes rapidly intractable because of the exponential growth in the number of possible combinations (i.e., $\mathcal{O}(C^{|\mathcal{N}_u|})$). What is worse, the cardinality of the neighbors may vary, so either we assume a maximum size of all neighbors' sets or we rely on permutation invariant functions like DGNs do. We opt for the latter option and weigh the contributions of the neighboring states equally using the **mean** operator:

$$P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1}) \approx \frac{1}{|\mathcal{N}_u|} \sum_{j}^{C} \underbrace{P_{\boldsymbol{\theta}}(Q_u = i \mid Q_*^{\ell-1} = j)}_{transition} \sum_{v \in \mathcal{N}_u} q_v^{\ell-1}(j). \tag{4.2}$$

We can intuitively understand the last equation by imagining that all neighboring variables have been collapsed into a "macro-variable" $Q_*^{\ell-1}$, whose categorical distribution is given by the element-wise mean of the individual distributions (viewed as C-sized vectors), i.e., the probability of $Q_*^{\ell-1}$ being in state j is $\frac{1}{|\mathcal{N}_u|} \sum_{v \in \mathcal{N}_u} q_v^{\ell-1}(j)$.

While it is true that each neighbor is weighted equally (i.e., the $\frac{1}{|\mathcal{N}_u|}$ term), the **adaptive transition** distribution weights the contribution of any neighboring state differently according to the arrival state i. Crucially, since we assume **full stationarity** of **all** adaptive distributions, the identities of a neighbor or the vertex itself are irrelevant to the parametrization of such distributions.

Another peculiar characteristic of this aggregation is that we do not necessarily weight the most likely state of each neighboring variable $Q_*^{\ell-1}$, but rather we consider the entire probability mass specified in the distribution $\boldsymbol{q}_v^{\ell-1}$: Section 4.1.4 will provide more details about this point.

4.1.2 Enhancing the Neighborhood Aggregation

The neighborhood aggregation scheme presented above ensures that the rightmost term of Equation 4.1 is still a valid probability, thus allowing us to find closed-form solutions when training the layer with the exact EM algorithm (details are provided later). However, this aggregation is limited in two respects: first, it does not take into account more than one previous layer, similarly to what skip connections do in deep neural networks [70, 107, 226]; secondly, it ignores the presence of edge features. Inspired by bottom-up generative models for tree-structure data [118, 129], we address these limitations by means of the so-called Switching Parent (SP) approximation [118, 227].

The goal is to modify the above equations to consider contributions from an arbitrary subset $\mathbb{L}(\ell)$ of previous layers as well as a finite number of **discrete** edge labels; to this aim, we introduce the random categorical variables L_u and S_u, respectively. Mathematically, the role of a Switching Parent variable Ξ is to decompose a complex conditional distribution over variables (let us call them I) into a convex combination of simpler ones

$$P(I_0 = i_0 | I_1 = i_1, \dots, I_k = i_k) \approx \sum_{\xi=1}^{k} P(\Xi = \xi) P^{\xi}(I_0 = i_0 | I_\xi = i_\xi),$$

where the rightmost **transition** probability depends on the value ξ of the SP variable.

The finite cardinality of the sets $\mathbb{L}(\ell)$ and $\mathcal{A}_g$ makes it possible to apply the SP approximation to our CGMM layer. The SP technique will first assign a specific weight to frozen neighboring states computed at different layers. In addition, for each layer, neighbors of vertex u connected with diverse edge types will be weighted differently as well. If we go back to the "macro-state" idealization, this corresponds to grouping neighboring variables into many macro-states, according to their relation with the previous layers and edge types. We give a graphical overview of the extended probabilistic layer in Figure 4.2

Hence, we can see Equation 4.2 as a special case of the following neighborhood aggregation (considering the extended set of neighboring observables $\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}$):

$$\begin{aligned} P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \approx \sum_{\ell' \in \mathbb{L}(\ell)} \underbrace{P_{\boldsymbol{\theta}}(L_u = \ell')}_{\textit{SP layer}} \sum_{a=1}^{|\mathcal{A}_g|} \underbrace{P_{\boldsymbol{\theta}}^{\ell'}(S_u = a)}_{\textit{SP edge}} \times \\ \times \frac{1}{|\mathcal{N}_u^{\ell',a}|} \sum_{j}^{C} \underbrace{P_{\boldsymbol{\theta}}^{\ell',a}(Q_u = i \mid Q_*^{\ell',a} = j)}_{\textit{SP-aware transition}} \sum_{v \in \mathcal{N}_u^{\ell',a}} q_v^{\ell'}(j), \end{aligned} \tag{4.3}$$

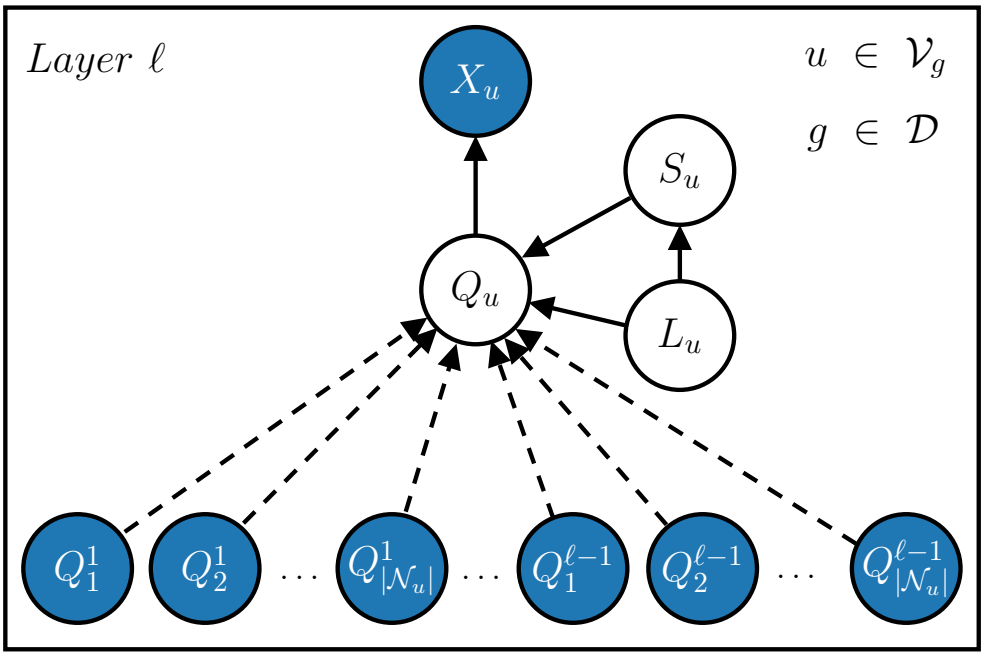


FIGURE 4.2: Graphical model of the "full" CGMM at layer ℓ. The SP variables weight the frozen neighboring states in relation to their layer and edge type. Dashed arrows denote the flow of contextual information coming from previous layers.

where $\mathcal{N}_u^{\ell',a} = \mathcal{N}_u^a$ defines the subset of neighboring observables computed at layer ℓ', whose associated vertices are connected to u with edge label a. Notice how we have adopted **positional stationarity** for the transition distribution and the switching parent S_u: the distributions are dependent on the layer and edge type we are considering. Similarly, the variable $Q_*^{\ell',a}$ identifies the macro-state obtained by averaging the neighboring contributions in $\mathcal{N}_u^a$.

Computationally speaking, each term $\frac{1}{|\mathcal{N}_u^a|} q_v^{\ell'}(j)$ is constant and can be pre-computed before training the current CGMM layer. Thanks to the incremental construction, this can substantially speed up the training process. From now on, when needed, we will talk about "pre-computed statistics" or simply **statistics**.

The Switching Parent approximation fits well inside the CGMM layer because it does not require reasoning about all layers *simultaneously*. Different approaches, such as Recurrent Neural Networks or Hidden Markov Models, assume that the "history" of states is not frozen and can change altogether through a *shared* transition function across layers. This design choice, however, would reintroduce the mutual dependencies between unobserved variables that we are trying to break with the incremental construction or, more generally, with the local and iterative processing of information described in Chapter 3. Thus, the SP variables provide a formal way to consider "skip connections" and discrete edge features in a **probabilistic** framework.

To summarize, the likelihood of a graph under the extended formulation of CGMM can be therefore written as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta} \mid g) &= \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} P_{\boldsymbol{\theta}}(X_u = x_u | Q_u = i) P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \approx \\ &\approx \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} \underbrace{P_{\boldsymbol{\theta}}(X_u = x_u | Q_u = i)}_{emission} \sum_{\ell' \in \mathbb{L}(\ell)} \underbrace{P_{\boldsymbol{\theta}}(L_u = \ell')}_{SP\ layer} \sum_{a=1}^{|\mathcal{A}_g|} \underbrace{P_{\boldsymbol{\theta}}^{\ell'}(S_u = a)}_{SP\ edge} \times \\ &\times \frac{1}{|\mathcal{N}_u^a|} \sum_{j}^{C} \underbrace{P_{\boldsymbol{\theta}}^{\ell',a}(Q_u = i \mid Q_*^{\ell',a} = j)}_{SP\text{-}aware\ transition} \sum_{v \in \mathcal{N}_u^a} q_v^{\ell'}(j). \end{aligned} \tag{4.4}$$

4.1.3 Training

When training the ℓ-th layer of CGMM, we maximize the likelihood of the data using the EM algorithm, by assuming that the graphs in the dataset $\mathcal{D}$ are i.i.d.. Similarly to the standard mixture model training, to compute the E-step we introduce the set of indicator random variables $\boldsymbol{Z}$. In particular, $Z_{ui\ell' aj} = 1$ if the latent variable Q_u of vertex u has value i while its observable neighboring variables coming from layer ℓ' are in state j and are connected to u with edge type a, and 0 otherwise. Note that there can also be other indicator variables, e.g., $Z_{ui\ell' a}$, where we express no interest in knowing the value of one or more subscripts of $Z_{ui\ell' aj}$. Using knowledge coming from $\boldsymbol{Z}$, we can write the complete log-likelihood formula to be maximized (omitting $\boldsymbol{\theta}$ to simplify the notation):

$$\begin{aligned}\log \mathcal{L}_c(\boldsymbol{\theta} \mid \boldsymbol{Z}, \mathcal{D}) &= \log \prod_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \prod_{i}^{C} \Bigg\{ P(x_u | Q_u = i) \prod_{\ell' \in \mathbb{L}(\ell)} \Bigg\{ P(L_u = \ell') \prod_{a=1}^{|\mathcal{A}_g|} \Big\{ P^{\ell'}(S_u = a) \times \\ &\times \prod_{j}^{C} \Big\{ \frac{P^{\ell',a}(Q_u = i | Q_*^{\ell',a} = j) \sum_{v \in \mathcal{N}_u^a} q_v^{\ell'}(j)}{|\mathcal{N}_u^a|} \Big\}^{Z_{ui\ell' aj}} \Big\}^{Z_{ui\ell' a}} \Bigg\}^{Z_{ui\ell'}} \Bigg\}^{Z_{ui}} \\ &= \sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \sum_{i}^{C} Z_{ui} \log P(x_u | Q_u = i) + \sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell' \in \mathbb{L}(\ell)} Z_{ui\ell'} \log P(L_u = \ell') \\ &+ \sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell' \in \mathbb{L}(\ell)} \sum_{a=1}^{|\mathcal{A}_g|} Z_{ui\ell' a} \log P^{\ell'}(S_u = a) \\ &+ \sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell' \in \mathbb{L}(\ell)} \sum_{a=1}^{|\mathcal{A}_g|} \sum_{j}^{C} Z_{ui\ell' aj} \log \frac{P^{\ell',a}(Q_u = i | Q_*^{\ell',a} = j) \sum_{v \in \mathcal{N}_u^a} q_v^{\ell'}(j)}{|\mathcal{N}_u^a|}. \end{aligned} \tag{4.5}$$

At layer $\ell = 0$, since there are no neighboring states to consider, the problem reduces to maximizing the complete log-likelihood of a standard mixture model (Section 2.1.2).

E-step

The E-step of the EM algorithm requires to compute the **expectation** of the complete log-likelihood w.r.t. $\boldsymbol{Z}$. Thanks to the properties of expectation, this is equivalent to replace each indicator variable in Equation 4.5 with its conditional expectation. Therefore, we define the following terms:

$$\begin{aligned}
&\mathbb{E}[Z_{ui}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}] = P(Q_u = i|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \\
&\mathbb{E}[Z_{ui\ell'}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}] = P(Q_u = i, L_u = \ell'|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \\
&\mathbb{E}[Z_{ui\ell' a}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}] = P(Q_u = i, L_u = \ell', S_u = a|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \\
&\mathbb{E}[Z_{ui\ell' aj}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}] = P(Q_u = i, L_u = \ell', S_u = a, K_u = j|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}),
\end{aligned}$$

noting that the first three terms can be straightforwardly obtained from the last one via marginalization. To formally model the aggregation process, we had to introduce the "macro-state" categorical variable K_u with C possible values, such that $P^{\ell',a}(K_u = j) = \sum_{v \in \mathcal{N}_u^a} q_v^{\ell'}(j)/|\mathcal{N}_u^a|$. Consequently, we can apply the Bayes Theorem on $\mathbb{E}[Z_{ui\ell' aj}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]$, yielding

$$\begin{aligned}
&\mathbb{E}[Z_{ui\ell' aj}|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}] = P(Q_u = i, L_u = \ell', S_u = a, K_u = j|\mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) \\
&= \frac{P(x_u|Q_u = i)P(Q_u = i, L_u = \ell', S_u = a, K_u = j|\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})}{P(x_u|\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})} \\
&= \frac{P(x_u|i)P(i|L_u = \ell', S_u = a, K_u = j, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})P(L_u = \ell')P^{\ell'}(S_u = a)P^{\ell',a}(K_u = j)}{Z_{norm}} \\
&= \frac{P(x_u|Q_u = i)P^{\ell',a}(Q_u = i|Q_*^{\ell',a} = j)P(L_u = \ell')P^{\ell'}(S_u = a)P^{\ell',a}(K_u = j)}{Z_{norm}} \\
&= \frac{P(x_u|Q_u = i)P(L_u = \ell')P^{\ell'}(S_u = a)P^{\ell',a}(Q = i|Q_*^{\ell',a} = j)P^{\ell',a}(K_u = j)}{Z_{norm}},
\end{aligned}$$

where Z_{norm} is the normalization term, obtained by $P(x_u|\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})$ via marginalization over all the latent variables (including K_u).

As we can see, the E-step can be computed without relying on variational approximations. Also, these operations can be easily parallelized due to the i.i.d.assumption between vertices, making the training of each layer scalable to larger graphs and amenable to GPU processing.

M-step

We use the posterior probabilities obtained in the E-step to update the parameters of the CGMM layer. After the addition of a suitable Lagrange multiplier to enforce probability requirements, we can obtain closed-form solutions for each adaptive distribution $P_{\boldsymbol{\theta}}$ in the following way [7]:

1. Compute the gradient of the **expected** complete log-likelihood w.r.t. $P_{\boldsymbol{\theta}}$. The resulting equation will depend on the Lagrange multiplier;
2. Compute the gradient of the **expected** complete log-likelihood w.r.t. the Lagrange multiplier, and plug the result into the resulting equation of the previous point.

We end up with the following update equations (omitting the subscript u to express stationarity of the learned distributions):

Transition Distribution

$$P^{\ell',a}(Q=i|Q_*^{\ell',a}=j)=\frac{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\mathbb{E}[z_{ui\ell'aj}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{i'=1}^{C}\mathbb{E}[z_{ui'\ell'aj}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}.$$

Switching Parents Distributions

$$P(L=\ell')=\frac{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{i=1}^{C}\mathbb{E}[z_{ui\ell'}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{i=1}^{C}\sum_{\ell''\in\mathbb{L}(\ell)}\mathbb{E}[z_{ui\ell''}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]},$$

$$P^{\ell'}(S=a)=\frac{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{i}^{C}\mathbb{E}[z_{ui\ell'a}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{i}^{C}\sum_{a'=1}^{|\mathcal{A}_g|}\mathbb{E}[z_{ui\ell'a'}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}.$$

Categorical Emission Distribution

$$P(X=k|Q=i)=\frac{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\delta(x_u,k)\mathbb{E}[z_{ui}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}{\sum_{\substack{g\in\mathcal{D}\\u\in\mathcal{V}_g}}\sum_{k'}^{K}\delta(x_u,k')\mathbb{E}[z_{ui}|\mathcal{D},\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}.$$

Gaussian Emission Distribution

$$\mu_i = \frac{\sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} x_u \mathbb{E}[z_{ui} | \mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}{\sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \mathbb{E}[z_{ui} | \mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]},$$

$$\sigma_i = \sqrt{\frac{\sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \mathbb{E}[z_{ui} | \mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}](x_u - \mu_i)^2}{\sum_{\substack{g \in \mathcal{D} \\ u \in \mathcal{V}_g}} \mathbb{E}[z_{ui} | \mathcal{D}, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}]}}.$$

As already mentioned, for the purposes of this thesis we will mainly focus on categorical and univariate Gaussian emission distributions. For multidimensional features, one could simply assume their conditional independence or rely on the vast amount of literature available [12, 13].

4.1.4 Inference

During inference, we compute the most likely *index* associated with the posterior of Q_u as representative for vertex u. In other words, it assigns u to one of the C potential clusters. Formally, this can be expressed as

$$\max_i P(Q_u = i | g, \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}) = \max_i \frac{P(x_u | Q_u = i) P(Q_u = i | \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})}{\cancel{P(x_u | \boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)})}}. \tag{4.6}$$

The equivalence is obtained by straightforward application of the Bayes Theorem; moreover, the denominator does not contribute to the maximization because it is independent of the state i, so it can be ignored.

Mathematically speaking, when training a CGMM layer, we formalized the neighborhood aggregation using the entire *frozen* posterior distribution of each vertex rather than its most likely state. Note, however, that our formalization is general enough to allow the use of either the former (continuous) representation or its one-hot variant, i.e., collapsing all probability mass into the most likely state.

To understand why this matters, let us assume $C = 3$ and consider the frozen posterior distribution of a vertex inferred at the previous layer being $(0.4, 0, 0.6)$ or $(0, 0.4, 0.6)$. Clearly, collapsing all the posterior mass onto the most likely state discards important information about the probability of being in the others. Therefore, there is a trade-off between a less noisy but approximate one-hot representation and a possibly noisy but exact one. We treat this choice as a hyper-parameter to be selected.

4.1.5 Building Graph Representations

After training CGMM for L layers, we can finally build a graph representation. Figure 4.3 schematizes the two-step process for $L = 2$. First of all, inferred vertex representations from all layers are concatenated into $|\mathcal{V}_g|$ vectors of size $C \times L$; concatenation is the most conservative choice when one wants to prevent loss of information. After that, these vectors are aggregated to obtain the final graph fingerprint, and we treat the choice of the permutation invariant function as a hyper-parameter. Indeed, it is reasonable to assume that the best choice is task-dependent; for instance, the *mean* aggregation abstracts from the size of a graph and focuses on variations in vertex distributions, whereas the *sum* encodes the most prevalent features in the vertex embedding space. To simplify our analysis, save computational resources, and generate task-agnostic graph representations, we do not incorporate *adaptive* aggregation functions in the process.

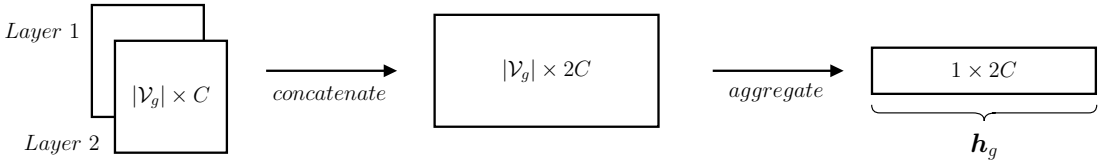


FIGURE 4.3: Example of a graph embedding construction for a 2-layer CGMM. Each layer outputs a representation of size C for each vertex $u \in \mathcal{V}_g$. After a concatenation step, vertex representations (of size $2C$) are aggregated into a graph embedding. The choice of the aggregation function, e.g., sum or mean, influences the final result.

4.1.6 Trade-offs of Vertex Representations

The $C - sized$ representation of a vertex at layer ℓ is also called a **unigram**. While this is the most straightforward way to obtain a vertex embedding, we can still build *structure-aware* representations called **bigrams**. A *bigram* is a C^2-sized vector which reflects how neighbors of a vertex are distributed. Formally, the bigram $\Phi(u)$ [228] of a vertex u is defined as

$$\Phi_{i_j}(u) = \sum_{v \in \mathcal{N}(u)} q_u(i) q_v(j), \quad i, j \in 1, \ldots, C.$$

To increase the richness of vertex and graph representations, whenever a bigram is used we concatenate it with its corresponding unigram, thus obtaining a **unibigram**. Here we have another trade-off to consider: unigrams are clearly less expensive to compute and store, but unibigrams carry more information.

4.1.7 Complexity and Scalability

Thanks to CGMM's architectural flexibility, the cost of training each layer ranges from constant (e.g., $\mathbb{L}(\ell) = \{\ell - 1\}$) to depth-specific (e.g., $\mathbb{L}(\ell) = \{1, \dots, \ell - 1\}$). Time and space complexity of a training epoch on a single graph are bounded by the cost of computing the E-step, which is $\mathcal{O}(|\mathcal{V}_g|(|\mathbb{L}(\ell)|C^2 + KC))$, where K stands for the number of vertex features. Instead, computing the statistics after training one layer has time complexity $\mathcal{O}(|\mathcal{E}_g|)$ because we just need to access the structure. The overall computation is therefore bounded by the sum of these two asymptotic terms that can be written as $\mathcal{O}(|\mathcal{V}_\mathcal{D}| + |\mathcal{E}_\mathcal{D}|)$.

Similarly to DNGNs, the *i.i.d* assumption on vertices allows to easily implement mini-batch training, with which we can arbitrarily reduce the memory fingerprint at run-time; this is especially important in hardware-constrained scenarios. Also, data parallelism can be trivially achieved by distributing the epoch's mini-batches on different CPUs or clusters of machines. For these reasons, CGMM is a suitable candidate to handle large-scale graph learning.[1]

4.1.8 Limitations

Due to the parametrized mean aggregation of neighboring observables, care must be taken when discriminating between structures with different connectivity but **same local distributions**. Indeed, when the distributions of the neighbourhood's states of two vertices are identical, CGMM cannot differentiate between them regardless of their connectivity. We can mitigate this issue by embedding the notion of vertex degree into the neighborhood aggregation mechanism. To do so, we consider $deg_{max}(g)$, i.e., the maximum degree of a graph g, and we intuitively connect each vertex u to $deg_{max}(g) - deg(u)$ dummy neighbors in a special hidden state $\perp$ called *bottom*. To maximize flexibility via the SP distributions, such dummy neighbors are connected to u with a *dedicated edge type*. Practically speaking, this means the statistics $\boldsymbol{Q}_{\mathcal{N}_u}^{\mathbb{L}(\ell)}$ will contain information about vertex u's degree, and such information is well-separated from the contextual information thanks to the use of the $\perp$ hidden state. Finally, note that we can also encode each vertex u's degree into x_u and use a Gaussian emission distribution to model its generation.

There are classes of graphs that cannot be distinguished so easily by CGMM, such as *k-regular graphs*, that is those such that $deg(u) = k \; \forall u \in g$. In particular, this is the family of structures that can be discriminated by the k-dim WL isomorphism test. Recently, Xu et al. [109] showed that almost all DNGNs are at most as powerful as the 1-dim WL test, but a similar proof for CGMM will be the subject of future works.

[1] https://github.com/diningphil/CGMM.

Summary

To summarize what we said so far, we detail the pseudo-code of the incremental training procedure in Algorithm 3, up to the construction of graph representations. In addition, we visually sketch CGMM's incremental construction in Figure 4.4.

Algorithm 3 Probabilistic Incremental Training

Input: dataset $\mathcal{D}$, maximum number of layers ℓ_{max} and epochs epoch_{max}.
Output: dataset of vertex and graph representations
for $\ell \leftarrow 1,\ldots,\ell_{max}$ **do**
    Initialize layer ℓ according to $\mathbb{L}(\ell), |\mathcal{A}|$ and C
    Load $\boldsymbol{Q}_{\mathcal{D}}^{\mathbb{L}(\ell)} = \{\boldsymbol{Q}_{\mathcal{N}_u^a}^{\ell',a} \mid \ell' \in \mathbb{L}(\ell), a \in |\mathcal{A}_g|, u \in g, g \in \mathcal{D}\}$
    for epoch $\leftarrow 1,\ldots,$ epoch_{max} **do**
        $\Delta_{likelihood}$, *posteriors* $\leftarrow$ E-Step$(\mathcal{D}, \boldsymbol{Q}_{\mathcal{D}}^{\mathbb{L}(\ell)})$
        M-Step(*posteriors*)
        if $\Delta_{likelihood} <$ threshold **then**
            break;
        end if
    end for
    $\boldsymbol{Q}_{\mathcal{D}}^{\ell} \leftarrow$ Inference$(\mathcal{D})$
    Store $\boldsymbol{Q}_{\mathcal{D}}^{\ell}$
end for
$R_{\mathcal{V}_{\mathcal{D}}} \leftarrow$ concatenate$(\{\boldsymbol{Q}_{\mathcal{D}}^{\ell'}, \ldots, \boldsymbol{Q}_{\mathcal{D}}^{\ell_{max}}\})$
$R_{\mathcal{D}} \leftarrow$ aggregate$(R_{\mathcal{V}_{\mathcal{D}}})$
return $R_{\mathcal{V}_{\mathcal{D}}}, R_{\mathcal{D}}$

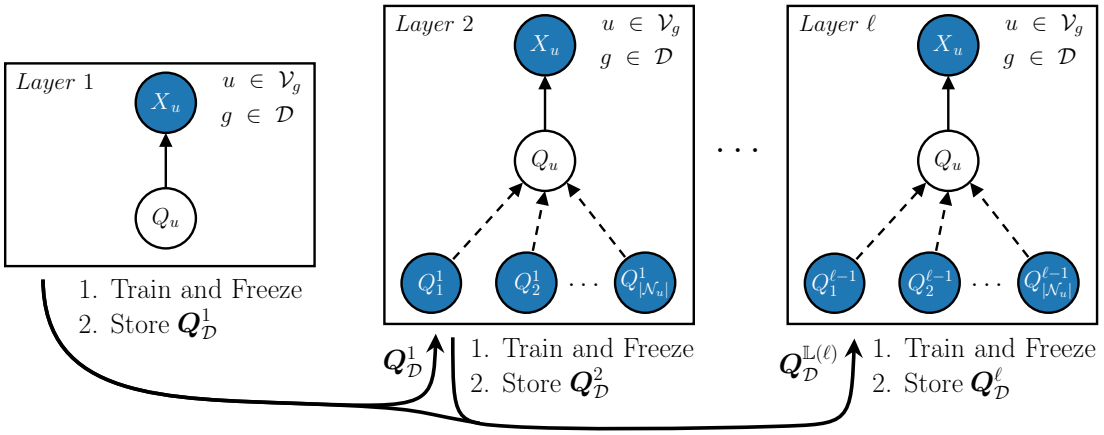


FIGURE 4.4: High-level description of CGMM's incremental construction, in light of the description given by Algorithm 3. Each layer is trained individually before being frozen; after that, statistics are computed and passed to the subsequent layers. The process can be repeated as many times as desired.

4.1.9 Experimental Setting

This section reports our experimental findings on some of the common graph classification benchmarks listed in Section 3.3.2 as well as a vertex classification task. To go more in depth, we study the effect of layering on performances, carry out ablation studies, and graphically show an example of context propagation across layers.

Datasets

As regards graph classification, we compare CGMM against PROTEINS, DD, NCI1, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-5K, and COLLAB from Table 3.4: we leave ENZYMES out of the evaluation due its very small size and nature of the features. The absence of vertex features in the social datasets constitutes a degenerate case that prevents CGMM from learning, i.e., the neighboring states will always have the same distributions. As done in [5, 229], when a vertex has no features, we add its degree as a continuous value as well as the *bottom* states discussed in Section 4.1.8.
In addition, to test CGMM performance on vertex classification tasks, we use the protein-protein interaction data set (PPI) introduced in [151]. We have already highlighted the troubling trends regarding some of the most common vertex classification benchmarks [177], so we prefer to stick to a dataset which is large and has a well-defined evaluation protocol. In this task, we are given a set of distinct large graphs, and our goal is to classify their vertices. Table 4.1 provides some details about the PPI benchmark.

	# Graphs	Classes	# Vertices	# Edges	# Vertex feat.
PPI	24	121	2372.67	34113.17	50

TABLE 4.1: Dataset statistics for the PPI vertex classification benchmark. In this task, we simultaneously have to predict 121 binary labels.

Hyper-parameters and Evaluation Protocol

This section describes the hyper-parameters tested as well as the chosen model assessment and selection procedures. However, for some of the baselines considered, there is no standardized evaluation protocol to follow such as the one of Section 3.3. For this reason, in order to compare with kernel methods and other DNGNs, we will report two experimental setups. The first uses random data splits, whereas the second follows [5] and provides a more reliable performance comparison. Both setups rely on grid searches. In Table 4.2, we list the hyper-parameters needed by CGMM and by the subsequent supervised classifier working on the unsupervised vertex/graph representations. We achieved constant per-layer complexity by setting $|\mathbb{L}(\ell)| = \{\ell - 1\}$, but we also evaluated the impact of considering all previous layers using the SP technique. The number

of EM epochs is fixed to 10, because the likelihood always stabilized around that value in our preliminary experiments. We also tried both *continuous* and *one-hot* vertex representations, and the global aggregation was either *sum* or *mean*. For some datasets, the preliminary experiments revealed that one global aggregation was dramatically better than the other, so we exploited this fact to reduce the number of combinations to test. The different configurations of a CGMM's layer mainly depend on *two* hyper-parameters only: the number of hidden states C and the number of layers. We can thus exploit the incremental nature of our model to further reduce the dimension of the grid search space: for each value C, we trained a single network for a maximum of L layers and then "cut" it to obtain the necessary configurations of depth $L' < L$. In addition, once the embeddings of all possible CGMM's configurations have been computed and stored, we can explore many combinations for the classifier without having to re-train CGMM every time. Speaking of classifiers, we considered a logistic regressor and a more flexible alternative in the form of a one-layer MLP with ReLU activations. Both of them are trained with Adam [224] and Cross-Entropy loss (Mean Squared Error for PPI). To contrast overfitting, we introduced L2 regularization and an early stopping technique.

Different setups across models. As anticipated, in the first evaluation setup we created random (stratified) data splits for the graph classification tasks. We followed a **Double Cross-Validation** strategy, with 10 external folds for risk assessment and 5 internal folds for model selection. All the other methodologies are evaluated according to a 10-fold CV strategy for risk assessment, so overall we expect the results to be roughly comparable. For an in-depth analysis of these baselines, the reader is referred to [7]. Regarding early stopping, we used the Generalization Loss [196] with $\alpha = 5$, which is considered to be a good compromise between training time and performances. In this respect, Table 4.2 reports the number of epochs after which early stopping starts; at the beginning of training, the validation loss smoothly oscillated and accuracy did not steadily increase, we believe stopping too early would have not been beneficial to get reliable performance estimates. On the contrary, the training/validation/test data splits of PPI were given, so we chose a simpler holdout approach.

Same setup across models. Here, we took advantage of the evaluation protocol of Section 3.3 to robustly re-evaluate CGMM against the most popular DNGNs. The only difference in the hyper-parameters regards the early stopping technique: for simplicity, we chose to use a simple patience-based stopping criterion, whose values correspond to those in Table 4.2. Instead, we experimented with the same hyper-parameters of COLLAB on both REDDIT datasets.

	D&D	**NCI1**	**PROTEINS**	**IMDB-***	**COLLAB**	**PPI**
C	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}
$\mathbb{L}(\ell)$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$
# layers	{5,10,15,20}	{10,20}	{5,10,15,20}	{5,10,15,20}	{5,10,15,20}	{5,10,20}
EM epochs	10	10	10	10	10	10
One-Hot/Continuous Vertex Repr.	both	both	both	both	both	both
Unigram/Unibigram	both	both	both	both	both	unigram
Sum/Mean Global Aggr.	sum	both	sum	both	mean	-
Classifier	{mlp}	{mlp}	{mlp}	{logistic, mlp}	{logistic, mlp}	{mlp}
# Hidden Units	{8,16,32,128}	{32,128}	{8,16,32,128}	{8,32,128}	{32,64,128}	{128,256,512}
Learning Rate	{1e-3,1e-4}	{1e-3}	{1e-3,1e-4}	{1e-3,1e-4}	{1e-3,1e-4}	{1e-2,1e-3}
L2 Weight Decay	{1e-2,5e-2,5e-3}	{1e-3,5e-4}	{1e-2,5e-2,5e-3}	{1e-2,1e-3}	{1e-3,5e-3,5e-4}	{0.,1e-5}
Classifier Epochs	5000	2000	5000	5000	5000	5000
Early-Stopping	1000	100	1000	1000	1000	500
Batch Size	100	200	100	100	100	20

TABLE 4.2: Hyper-parameters tried during model selection. IMDB-* refers to both binary and multi-class tasks. The number of hidden units per layer are ignored when using logistic regression.

4.1.10 Results

We now present CGMM's results on graph and vertex classification, along with empirical studies on the beneficial effects of depth. These will provide us with additional hints on CGMM's ability to extract useful information in an unsupervised fashion. We also study the impact of the layer-wise SP variable as part of our ablation studies. Finally, we visualize how the model propagates contextual information across the graph.

Graph Classification

We evaluate the performance of our method against different kernels and deep learning techniques for graph classification. Table 4.3 provides a comparison between the kernels considered and CGMM in terms of computational costs. As we can see, some kernels can be inadequate when it comes to large scale training and inference because of their (at least) quadratic time complexity in the number of graphs. Moreover, the considered kernels for graphs are not applicable to continuous vertex features, which limits their applicability to different domains.

Kernel	Cost	Reference
GK	$\mathcal{O}(\lvert\mathcal{D}\rvert^2 n d^{k-1})$	[54]
RW	$\mathcal{O}(\lvert\mathcal{D}\rvert^2 n^3)$	[44]
PK	$\mathcal{O}(m(h-1) + h\lvert\mathcal{D}\rvert^2 n)$	[230]
WL	$\mathcal{O}(\lvert\mathcal{D}\rvert hm + \lvert\mathcal{D}\rvert^2 hn)$	[45]
CGMM	$\mathcal{O}(L(\lvert\mathcal{D}\rvert n + \lvert\mathcal{D}\rvert m))$	

TABLE 4.3: Computational costs of graph kernels compared to CGMM. We assume that all graphs have size $n = |\mathcal{V}_g|$, $m = |\mathcal{E}_g|$ edges and maximum degree d. Moreover, k is the size of the graphlets (i.e., subgraphs) counted by GK, and h is the number of iterations needed by different procedures to compute the final similarity scores.

Different setups across models. Results for graph classification, under the first of the two evaluation protocols considered, are shown in Table 4.4. CGMM performs well in all data sets (scoring top-3 on five of them), even though the probabilistic architecture was not trained to solve a classification task. In particular, we achieve competing results on all three collaborative data sets, and we improve the best result on NCI1. This suggests that learning the distribution of a vertex's neighbourhood at different abstraction's levels produces a rich unsupervised graph representation. As a matter of fact, in 9 out of 10 external folds on NCI1, the model selection procedure chose a configuration with 20 layers; in contrast, the DNGNs of Table 4.4 exploit a maximum of 4 graph convolutions. The results also highlight that CGMM can perform well even when the only source of information is structural, i.e., the degree of a vertex.

	D&D	NCI1	PROTEINS	IMDB-B	IMDB-M	COLLAB
GK [54]	74.38 ± 0.7	62.49 ± 0.3	71.39 ± 0.3	-	-	-
RW [44]	> 3 days	> 3 days	59.57 ± 0.2	-	-	-
PK [230]	78.25 ± 0.5	82.54 ± 0.5	73.68 ± 0.7	-	-	-
WL [45]	78.34 ± 0.6	$\mathbf{84.46} \pm 0.5$	74.68 ± 0.5	-	-	-
ARMA [158]	74.86	-	75.12	-	-	-
PSCN [229]	76.27 ± 2.6	76.34 ± 1.7	75.00 ± 2.5	71.00 ± 2.3	45.23 ± 2.8	72.60 ± 2.15
DCNN [231]	58.09 ± 0.5	56.61 ± 1.0	61.29 ± 1.6	49.06 ± 1.4	33.49 ± 1.4	52.11 ± 0.7
ECC [148]	72.54	76.82	-	-	-	-
DGK [47]	-	62.48 ± 0.3	71.68 ± 0.5	66.96 ± 0.6	44.55 ± 0.5	73.09 ± 0.3
DGCNN [162]	$\mathbf{79.37} \pm 0.9$	74.44 ± 0.5	75.54 ± 0.94	70.03 ± 0.86	47.83 ± 0.9	73.76 ± 0.5
PGC-DGCNN [232]	78.93 ± 0.9	76.13 ± 0.7	$\mathbf{76.45} \pm 1.02$	71.62 ± 1.2	47.25 ± 1.4	75.00 ± 0.58
CGMM-nb	77.35 ± 1.6	77.02 ± 1.8	75.11 ± 2.8	71.07 ± 3.5	47.36 ± 3.4	73.3 ± 2.9
CGMM-full	77.20 ± 3.1	76.94 ± 1.6	75.45 ± 4.4	$\mathbf{72.30} \pm 3.5$	49.42 ± 3.6	$\mathbf{76.06} \pm 2.4$
CGMM	77.15 ± 3.5	$\mathbf{77.80} \pm 1.9$	75.56 ± 3.0	72.1 ± 2.3	$\mathbf{49.73} \pm 1.6$	75.50 ± 2.74

TABLE 4.4: CGMM's results of a 10-Fold Double Cross Validation for graph classification. Best results are reported in bold. We report CGMM's accuracy on NCI1 in bold because it performs better than the other neural models. CGMM-nb indicates that the model is not using bigram features, whereas CGMM-full represents the extended CGMM where each layer exploits all previous layers.

Note that kernels can process and compare graphs more explicitly than DGNs: one of the reasons why the WL kernel has higher accuracy on NCI1 may be due to the kind of structural patterns used to compute the similarity score. Still, when the number and size of the graphs to consider increases, using these kernels becomes challenging.

Same setup across models. If re-evaluated under the rigorous setup of Section 3.3, we can appreciate how CGMM is still competitive against the new pool of models, with special mention for the social tasks. The structure agnostic baseline, instead, still leads on D&D, PROTEINS, and IMDB-MULTI.

In addition, please note how large the performance gap can be on some datasets w.r.t. the former evaluation protocol, with approximately 3 points less on D&D and PROTEINS for DGCNN and even CGMM (though standard deviation are relatively high). This is further confirmation that to clearly evaluate progress one should at least keep the experimental setting identical for all methods.

	D&D	NCI1	PROTEINS
BASELINE	$\mathbf{78.4} \pm 4.5$	69.8 ± 2.2	$\mathbf{75.8} \pm 3.7$
DGCNN	76.6 ± 4.3	76.4 ± 1.7	72.9 ± 3.5
DIFFPOOL	75.0 ± 3.5	76.9 ± 1.9	73.7 ± 3.5
ECC	72.6 ± 4.1	76.2 ± 1.4	72.3 ± 3.4
GIN	75.3 ± 2.9	$\mathbf{80.0} \pm 1.4$	73.3 ± 4.0
GRAPHSAGE	72.9 ± 2.0	76.0 ± 1.8	73.0 ± 4.5
CGMM	74.9 ± 3.4	76.2 ± 2.0	74.0 ± 3.9

TABLE 4.5: Mean and standard deviation results on chemical datasets of a 10-fold Cross Validation (setup of Section 3.3). Best results are reported in bold.

	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K	COLLAB
Baseline	70.8 ± 5.0	$\mathbf{49.1} \pm 3.5$	82.2 ± 3.0	52.2 ± 1.5	70.2 ± 1.5
DGCNN	69.2 ± 3.0	45.6 ± 3.4	87.8 ± 2.5	49.2 ± 1.2	71.2 ± 1.9
DiffPool	68.4 ± 3.3	45.6 ± 3.4	89.1 ± 1.6	53.8 ± 1.4	68.9 ± 2.0
ECC	67.7 ± 2.8	43.5 ± 3.1	-	-	-
GIN	71.2 ± 3.9	48.5 ± 3.3	$\mathbf{89.9} \pm 1.9$	$\mathbf{56.1} \pm 1.7$	75.6 ± 2.3
GraphSAGE	68.8 ± 4.5	47.6 ± 3.5	84.3 ± 1.9	50.0 ± 1.3	73.9 ± 1.7
CGMM	$\mathbf{72.7} \pm 3.6$	47.5 ± 3.9	88.1 ± 1.9	52.4 ± 2.2	$\mathbf{77.32} \pm 2.2$

Table 4.6: Mean and standard deviation results on social datasets of a 10-fold Cross Validation (setup of Section 3.3). Best results are reported in bold. Note that the degree is the sole vertex feature used by all models.

Overall, CGMM has proved to be a satisfactory unsupervised model, given that the richness of its graph embeddings allowed us to get very close to the state of the art.

Vertex Classification

We now turn our attention to vertex classification, specifically on the PPI benchmark. Following the literature [176], we compare CGMM against GraphSAGE and DGI, as well as a structure-agnostic baseline that applies logistic regression to the vertex features. GraphSAGE, DGI, and CGMM share a first pre-training step, in which vertex embeddings are learned in an unsupervised fashion before feeding the learned vertex representations to a supervised classifier. Results are shown in Table 4.7: we observe that CGMM has very good performances, improving against all GraphSAGE variants but for DGI. Considering that DGI uses GraphSAGE as part of its framework, it seems that the learning procedure is what generates the gap between the two methods. In fact, while GraphSAGE relies on a link prediction loss and an entropy penalization term to learn vertex representations, DGI learns to discriminate vertices according to a contrastive noise procedure.

	Data Used	Micro F1
Baseline	$\mathcal{V}_g$	42.2
GraphSAGE-GCN	$\mathcal{V}_g, \mathcal{E}_g$	46.5
GraphSAGE-mean	$\mathcal{V}_g, \mathcal{E}_g$	48.6
GraphSAGE-LSTM	$\mathcal{V}_g, \mathcal{E}_g$	48.2
GraphSAGE-pool	$\mathcal{V}_g, \mathcal{E}_g$	50.2
DGI	$\mathcal{V}_g, \mathcal{E}_g$	**63.8**
CGMM-full	$\mathcal{V}_g, \mathcal{E}_g$	58.4
CGMM	$\mathcal{V}_g, \mathcal{E}_g$	60.2

Table 4.7: CGMM's results of inductive vertex classification on PPI. We report the Micro Average F1 score across the 121 target labels.

Hyper-parameters' Analysis

We enrich our empirical analysis with three further studies: the first concerns the impact of the unibigram technique on performances; the second inspects the potential performance advantages of using all previous layers when training a new layer; the last one investigates whether a wider model with fewer layers can perform as well as a deep model with fewer hidden states. With the exception of the second study, the other analyses will use the information coming from the previous layer, i.e., $\mathbb{L}(\ell) = \{\ell - 1\}$.

Unibigram Ablation In Table 4.4, we re-evaluated the model on all data sets by constraining CGMM to only use unigrams (CGMM-nb). Results indicate a slight performance drop on chemical data sets and a larger decrease in social data sets. This suggests that, especially when we only have access to structural information (i.e., the degree distribution), computing a graph representation that takes the structure into account can be helpful. Nevertheless, CGMM-nb performances still remain good with respect to the state of the art.

On the Impact of Previous Layers We repeated all experiments by conditioning each layer of the architecture on the entire subset of previous layers, i.e., $\mathbb{L}(\ell) = \{1, \ldots, \ell - 1\}$. This way, each layer is free to weigh the previous layers (thanks to the SP variable) to maximize the likelihood of each graph. Results (Table 4.4 and 4.7, CGMM-full) indicate that the model performs almost always on par w.r.t. the significantly more efficient version that does not use the SP variable L_u. Nonetheless, despite the negligible performance advantage obtained on these tasks, we recommend treating the use of L_u as a hyper-parameter of the model when dealing with other vertex or graph classification tasks.

Sensitivity Analysis When designing any deep network (let it be neural or probabilistic), it is useful to analyze the relation between the dimension of each layer's hidden representation and the number of layers in terms of performance variations. For DGNs the depth of the architecture is functional to context spreading, so we would expect that having a larger hidden representation for each layer is not enough to compensate for the flow of information between vertices. We provide an example in Figure 4.5 to show how the validation accuracy of a logistic regressor on NCI1 varies while changing the number of hidden states C and CGMM's depth. It can be seen that the graph representation associated with point A (of size $C = 60$) is not sufficient to achieve the same performance of the graph representation associated with point B ($C = 45$). This means that depth is crucial to encode more information.

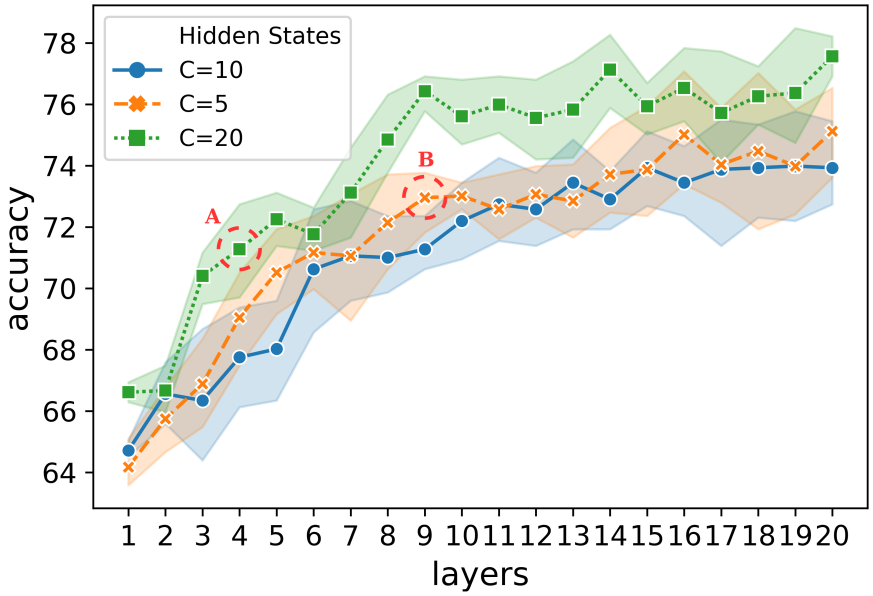


FIGURE 4.5: This picture shows that using a large C but few layers is not always enough to reach the same performance of a deeper network with a smaller C. Points A and B are associated with representations of dimensions $C = 60$ and $C = 45$, respectively. However, the latter has better validation performances if we consider a logistic regressor on NCI1, which means that the flow of context is indeed more beneficial than increasing the number of hidden states C. Results are averaged over five independent runs and standard deviations are shown as colored bands.

On the Effects of Depth This Section is meant to answer to two further research questions. First, we want to quantify the effect of depth on the architecture when coupled with a classifier. The second point is about understanding how much the model's performance is affected by a random initialization of the layers. To address both questions, we took a random train-validation-test split of NCI1 to conduct a new experiment. With its 4110 graphs, NCI1 was chosen to minimize the effect of the data split on results and consequently on the random initialization of the classifier. In contrast, the other chemical data sets seem more data split-dependent. We trained a 20-layer CGMM for some of the configurations defined in Table 4.2, and we repeated each process five times averaging the results. Figure 4.6 reports the accuracy versus the number of layers for such configurations, with logistic regression or MLP classifiers. We see that, in both cases (top part of the figure), depth has a beneficial effect on test accuracy, with slightly worse results on test accuracy when using logistic regression due to its strong bias. Notice how validation and test curves tend to an asymptote after ten layers; this information may be used as stopping criterion when constructing the architecture for supervised tasks, as proposed in [233] for convolutional networks on images.

One interesting thing to notice is that we do not necessarily incur in the curse of dimensionality as the size of the fingerprint grows larger and larger with the layers. This

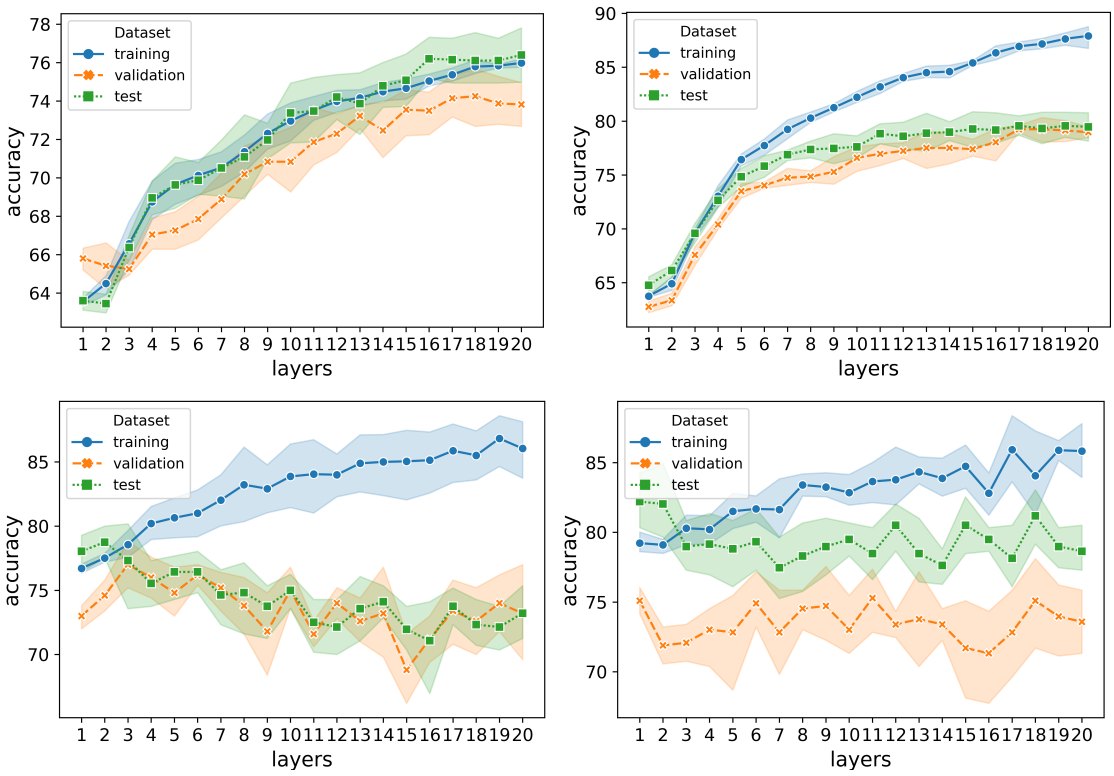


FIGURE 4.6: Stability experiments on NCI1 with logistic regressor (top left) and MLP (top right), PROTEINS with MLP (bottom left) and D&D with MLP (bottom right), which show how accuracy varies from 1 to 20 layers. Results are averaged over five independent runs. Colored bands denote standard deviation. While the effect of depth is beneficial for NCI1, a different behavior emerge on PROTEINS and D&D, which is not to be attributed to random splits.

cold be ascribed to the fact that the fingerprint construction at layer ℓ is guided by layer $\ell - 1$, and this dependency generates a completely different learning problem at each layer. This may explain why we do not quickly overfit the training data after the 20 layers.

In addition, since the training accuracy on NCI1 does not significantly vary between different runs (due to different weights' initialization), we get an indication that the pooling strategy of [146], used in our preliminary contribution [6] brings a negligible advantage to CGMM. This also holds for PROTEINS and D&D when the architecture is very shallow (up to three layers), though results need to be taken with a pinch of salt because of the discussion of Section 3.3.3. As a matter of fact, it is still an open question whether a dataset is "too simple" or the graph convolutions devised so far are unable to extract the relevant features from the graphs.

Visualization: a Case Study

This part provides a visual exploration and interpretation of CGMM's internal dynamics. This kind of analysis is meant to demonstrate how the model extracts different patterns at each layer of the architecture in a way that is consistent with what stated Chapter 3. Therefore, Figure 4.7 sketches how information spreads in a real NCI1 molecule. We represent each vertex's posterior as a pie chart with C different colours; in this experiment, we use a 4-layer CGMM with $C = 3$. Please keep in mind that colours assignment between different layers is irrelevant. The rightmost six atoms of the molecule have the same atomic symbol, so they will be assigned an identical state at layer 1. What is more, these six atoms alone form a 2-regular subgraph, which means that their state can only change if context flows from left to right, as shown by the dashed arrows on the top-left side of the figure. If it were not for the five leftmost vertices context could not flow, because all neighborhoods would look identical to the model. At each layer, we highlight the vertices of interest inside a dashed regions, and the associated heatmap of states (one vertex's posterior per row) proves that information flows as expected.

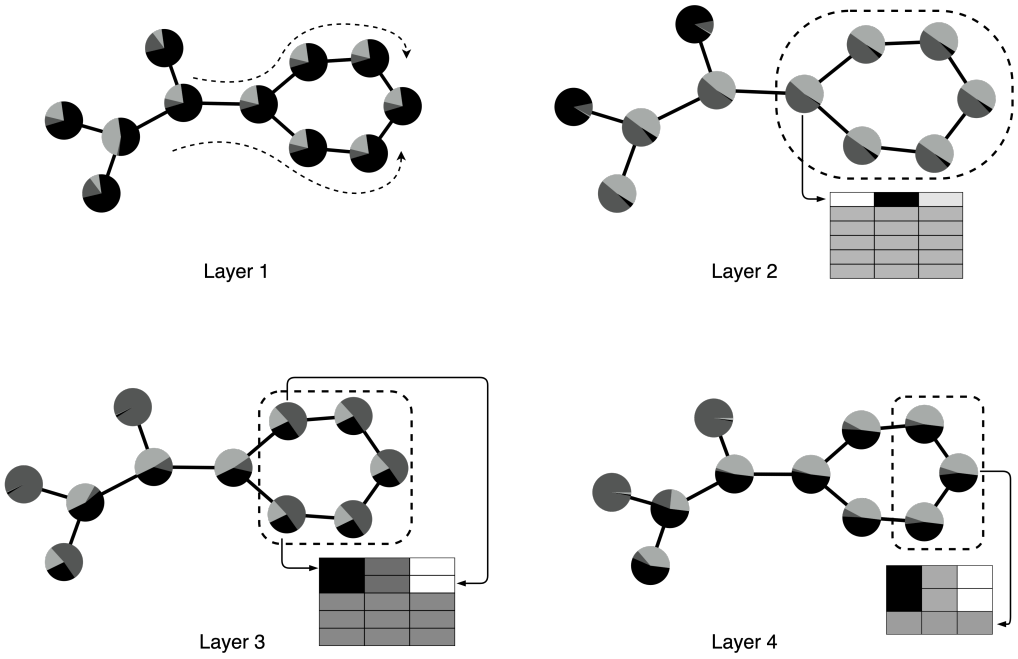


FIGURE 4.7: Context flow on a real NCI1 graph. We focus on the highly regular rightmost subgraph, which is influenced by the left part of the structure. Posterior's heatmaps show that, although relatively small, posteriors of vertices change as expected.

4.1.11 Summary

In this section we have introduced the core contribution of the thesis. The Contextual Graph Markov Model is a deep, unsupervised, fully probabilistic, and efficient way to compute vertex/graph embeddings. It brings together the principles of Deep Graph Networks and the probabilistic tools of Bayesian networks. In the experiments, we have shown that the model allows a common classifier to reach the state of the art on different classification tasks, even though the input embeddings were not specifically calibrated for supervised learning.

Despite our enthusiasm, there are still many ways in which we can improve CGMM. Throughout the thesis, we saw that regular graphs cannot be discriminated by the model, or that the mean aggregator is not as expressive as the sum. Solving this issues in CGMM while maintaining the fully probabilistic nature of the method is not an easy task. The same can be said for graphical extensions of the model to make it dependent on a target label: implementing a permutation invariant readout transduction that is also tractable remains an unsolved problem, to the best of our knowledge. Notably, the use of a SP variable to model the global aggregation would not work, because the SP does not consider mutual dependencies between the elements to aggregate. For these reasons, a potential future direction would be to devise an efficient sum-based neighbor aggregator and/or readout such that we can still compute closed form solutions when training each layer of our architecture. These two contributions would greatly enhance the expressivity of the probabilistic framework we have introduced.

Another important limitation of CGMM is that its applicability is restricted to the use of discrete edge labels. This is problematic when we want to take into account the distance between entities in the graph or a more complex edge feature that lives in a multi-dimensional space. We know, however, that this problem can be addressed, and the next section will be just devoted to that.

4.2 Beyond Discrete Edge Features [8]

We turn our attention to the problem of modeling (possibly multidimensional) continuous edge features in our deep and fully probabilistic framework. The benefits of the methodology we are about to propose are multi-faceted. First and foremost, **we extend CGMM** to enlarge the classes of graphs it can handle. Secondly, we show that the unsupervised model can build richer graph representations *even in the absence of edge features*.

It is easy to see why we cannot keep using the Switching Parent technique in the presence of continuous edge features. If the support of the p.d.f. associated with the SP variable was infinite, we would have to replace the summation over the discrete states with an integral. By doing that, we would lose the closed-form solutions to the MLE estimation problem as well as the convergence guarantees of the EM algorithm. Because we want to preserve these nice characteristics of CGMM, we have to resort to a different solution.

The key aspect of our contribution is the following: we can **learn** to "discretize" edge features so as to use them in the original CGMM model. To do that, we adopt an architectural approach in which we train a secondary Bayesian network to model the generation of edge features. The (discrete) edge state will then be used by the SP variable of the original CGMM model. Empirically, we will show that this is better than using a hand-made heuristic to obtain discrete edge features, and the advantage persists even when edge features are **absent** from the graph.

In the following sections, we will briefly introduce the mathematical variations to the model, which we will call Extended Contextual Graph Markov Model (E-CGMM), noting how the derivations do not change in any way. The asymptotic complexity will remain linear in the number of edges, so the model will still be fairly efficient. Then, we will highlight the performance improvement against CGMM on graph classification benchmarks, which can be attributed to the richness of the learned graph representations. Moreover, we will study the impact of E-CGMM on a graph regression and three link prediction tasks, to show the advantage of explicitly modeling the generation of edge features.

4.2.1 Layer Definition

The graphical model of each E-CGMM's layer is represented in Figure 4.8: an additional Bayesian network, very similar to that of CGMM, is responsible for modeling the generation each edge feature a_{uv} through a latent categorical variable Q_{uv}. This variable can take C_E different values, in contrast to the number of latent states of the original models that we will call C_V from now on. In principle, edges act as fictitious vertices whose neighbors are the source and destination vertex states inferred and frozen at the previous layers. In the interest of clarity, we will omit the layer-wise SP variable and consider contributions coming from the previous layer only. In addition, the "edge component" on the right uses two special discrete edge labels, one for the source (A_s) and one for the destination A_d states, to model the direction of the edge under consideration. If the graph is undirected, we can assume a uniform distribution for the SP variable S_{uv}.

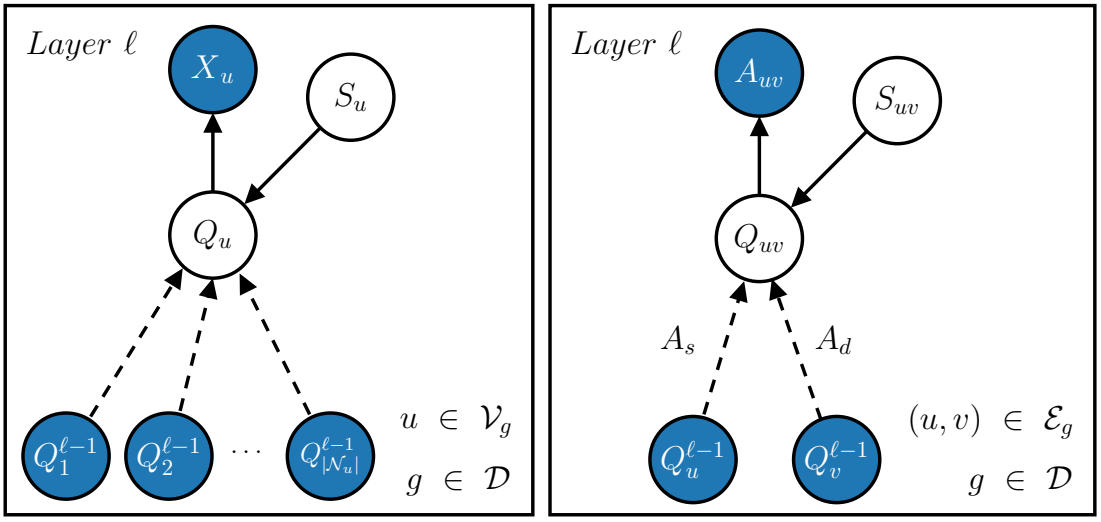


FIGURE 4.8: Graphical model of a generic layer ℓ of E-CGMM. Dashed arrows denote the flow of contextual information coming from previous layers. We have omitted the SP variables L_u and L_{uv} for simplicity of exposition.

Formally, we model the generation of vertex and edge features as follows:

$$P(x_u \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1}, \boldsymbol{Q}_{\mathcal{E}_u}^{\ell-1}) = \sum_{i=1}^{C_V} \underbrace{P(x_u \mid Q_u = i)}_{\text{vertex emission}} P(Q_u = i \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1}, \boldsymbol{Q}_{\mathcal{E}_u}^{\ell-1})$$

$$P(a_{uv} \mid Q_u^{\ell-1}, Q_v^{\ell-1}) = \sum_{i=1}^{C_E} \underbrace{P(a_{uv} \mid Q_{uv} = i)}_{\text{edge emission}} P(Q_{uv} = i \mid Q_u^{\ell-1}, Q_v^{\ell-1}),$$

where $\boldsymbol{Q}_{\mathcal{E}_u}^{\ell-1}$ denotes the set of states inferred by the edge component at the previous layer. Likewise CGMM, when $\ell = 0$, the equations simplify and the layer implements

two standard mixture models that do not consider contextual information.

Thanks to the fact that frozen edge states are inferred from a categorical variable, we approximate the conditional distribution of the vertex component as

$$P(Q_u = i \mid \boldsymbol{Q}^{\ell-1}_{\mathcal{N}_u}, \boldsymbol{Q}^{\ell-1}_{\mathcal{E}_u}) = \sum_{a=1}^{C_E} \underbrace{P(S_u = a)}_{SP} \underbrace{P^a(Q_u = i \mid \boldsymbol{Q}^{\ell-1}_{\mathcal{N}^a_u}, \boldsymbol{Q}^{\ell-1}_{\mathcal{E}_u})}_{\text{vertex transition}}.$$

This last equation, while apparently similar to the one of CGMM, shows the interplay between the vertex-centric and edge-centric components of E-CGMM, which makes it possible to incorporate arbitrary edge information in the fully probabilistic neighborhood aggregation scheme.

Similarly, the rightmost term of $P(a_{uv} \mid Q^{\ell-1}_u, Q^{\ell-1}_v)$ can be decomposed as

$$P(Q_{uv} = i \mid Q^{\ell-1}_u, Q^{\ell-1}_v) = \sum_{a}^{A_s, A_d} \underbrace{P(S_{uv} = a)}_{SP} \underbrace{P^a(Q_{uv} = i \mid Q^{\ell-1}_a)}_{\text{edge transition}},$$

where we remind that A_s and A_d are the discrete labels assigned to source and destination vertices, respectively, and $Q^{\ell-1}_a$ is $Q^{\ell-1}_u$ if $a = A_s$, and $Q^{\ell-1}_v$ otherwise.

The last brick in the formalization of the model is the definition of the transition distribution $P^a(Q_u = i \mid \boldsymbol{Q}^{\ell-1}_{\mathcal{N}^a_u}, \boldsymbol{Q}^{\ell-1}_{\mathcal{E}_u})$. Using the additional edge information we have, we can write

$$P^a(Q_u = i \mid \boldsymbol{Q}^{\ell-1}_{\mathcal{N}^a_u}, \boldsymbol{Q}^{\ell-1}_{\mathcal{E}_u}) = \sum_{j=1}^{C_V} P^a(Q_u = i \mid Q^{\ell-1}_* = j) \sum_{v \in \mathcal{N}^a_u} q^{\ell-1}_v(j) \frac{q^{\ell-1}_{uv}(a)}{\sum_{v \in \mathcal{N}^a_u} q^{\ell-1}_{uv}(a)}$$

where we recall that $q_u(j)$ and $q_{uv}(j)$ are the j-th components of the inferred states (represented as a vector) inferred at the previous layer. The transition distribution we just presented is a generalization of Equation 4.3, where we have exploited the posterior of each edge to weight the contribution of the individual neighbors. Moreover, as in CGMM, we assume full stationarity on vertices and on edges, meaning that we share the parameters of the emission, transition, and SP distributions across all vertices or edges depending on the component of E-CGMM.

To train an E-CGMM layer, and similarly for inference, it is sufficient to apply EM to the two **independent** Bayesian networks, and use their inferred states as statistics for the subsequent layer of the architecture. We remark that. mathematically speaking, we are still dealing with conditional mixture models. Therefore, at the cost of training an additional network for edges, which shares a similar time complexity as the original CGMM, we obtain a deep architecture capable of building **both vertex and edge**

embeddings from raw graphs, something that not many other DGNs can do to the best of our knowledge.

4.2.2 Dynamic Neighborhood Aggregation

There is one subtle but very important difference between CGMM and E-CGMM that could potentially contribute to the richness of vertex embeddings produced by the latter. Whenever edge features are missing but a dummy feature is used in their place, the edge latent states can still vary across the graph because they depend on the *source* and *destination* frozen states. Therefore, at different layers, the posterior distributions of the same edge may be different regardless of the absence of real edge features. From a methodological point of view, this allows for **different** groupings of the **same** vertex neighbors at different layers. On the other hand, CGMM always groups neighbors in the same way, since it relies on static and discrete edge features. We denote this peculiar characteristic of E-CGMM with the term "dynamic neighborhood aggregation", which is sketched in Figure 4.9. We believe this is the main reason why E-CGMM shows significant performance improvements with respect to CGMM on the graph classification benchmarks, as we will see in the following.

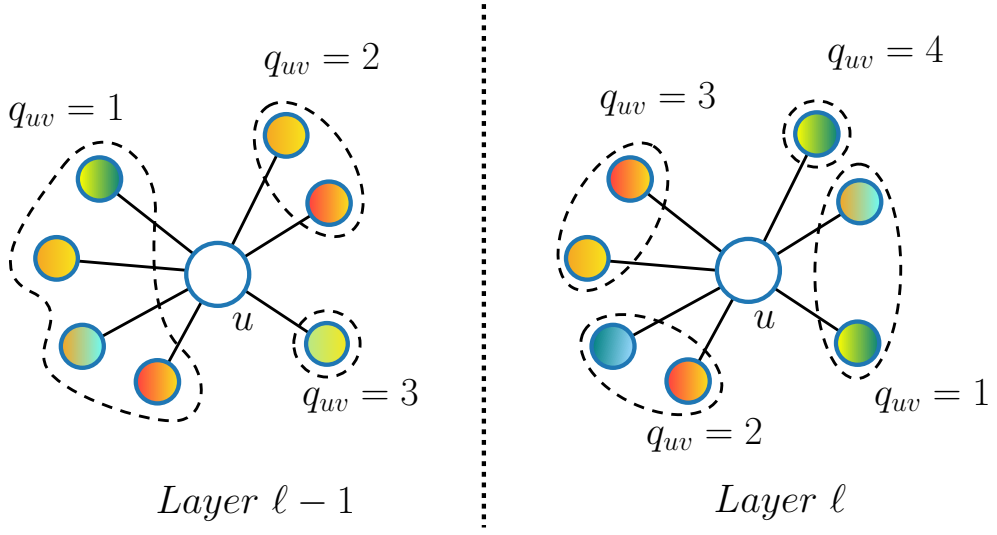


FIGURE 4.9: We show an example of dynamic neighborhood aggregation with $C_E = 4$. At layer $\ell - 1$, the neighbors of vertex u are split into 3 groups according to the **edge** states computed at layer $\ell - 2$. Because edge states vary, at layer ℓ a different grouping of the same neighbors can be induced.

4.2.3 Complexity and Scalability

E-CGMM shares an asymptotic efficiency comparable, in both time and space, to CGMM. At a given layer ℓ, the complexity of E-CGMM's vertex component is bounded by $\mathcal{O}(|\mathcal{V}_g|(C_E C_V^2 + KC_V)$, where K is the number of vertex features. To compute the posterior and bigram of each vertex, time complexity is again $\mathcal{O}(|\mathcal{E}_g|)$. Hence, the overall time complexity becomes $\mathcal{O}(|\mathcal{V}_g| + |\mathcal{E}_g|)$. Instead, the edge component of E-CGMM is bounded by $\mathcal{O}(|\mathcal{E}_g|(2C_E C_V + KC)$. Clearly, the time complexity of our extension will be always strictly greater than CGMM; however, asymptotically speaking, it is still controlled by $\mathcal{O}(|\mathcal{V}_g| + |\mathcal{E}_g|)$ whenever $C_V \ll |\mathcal{V}_g|, C_V \ll |\mathcal{E}_g|, C_E \ll |\mathcal{V}_g|$, and $C_E \ll |\mathcal{E}_g|$.

4.2.4 Embeddings Construction

Differently to CGMM, after the inference phase we will obtain both vertex and edge representations At model selection time, we can still choose to concatenate a vertex unigram and bigram (obtaining a *unibigram*) or not. However, to obtain graph representations at each layer ℓ, we shall independently aggregate all vertex and edge representations in the graph via permutation-invariant operators, such as the mean or the sum, followed by concatenation of the two resulting vectors. The final unsupervised graph embedding is then the concatenation of these embeddings across all layers of the deep architecture.

4.2.5 Experimental Setting

To assess the performances of E-CGMM, we will carry out three different evaluations. First, we will compare against the same set of graph classification benchmarks on which we evaluated CGMM. Secondly, we will show the importance of adaptively discretizing edge features on a graph regression benchmark. Finally, we will compare CGMM and E-CGMM on the three link prediction tasks following a rigorous evaluation scheme.[2].

Graph Classification

Please recall that the datasets under consideration do not provide edge attributes, so the scope of our analysis is to evaluate the richness of graph embeddings given by the **dynamic aggregation mechanism** as well as the use of posterior edge probabilities. We follow the empirical evaluation of Section 3.3.

[2] https://github.com/diningphil/E-CGMM

In terms of hyper-parameters, we set C_V to 20^3 whereas C_E was chosen from $\{5, 10\}$. The other hyper-parameters are selected according to Table 4.2, to keep the model selection as similar as possible between the two methods.

Graph Regression

To understand the importance of handling continuous edge information, we consider the QM7b graph regression task [234, 235], a chemical dataset composed of more than 7k organic molecules. The task is to predict 14 continuous properties of each molecule, where a molecule is associated with a Coulomb interaction matrix that is used to extract vertex and edge features. The entry (i, j) of the Coulomb matrix is proportional to the product of nuclear charges of atoms i and j and inversely proportional to their distance (for non-diagonal elements). In the dataset, there are 6 different continuous diagonal elements of the matrix, which are used as discrete labels for the vertices. On the other hand, we consider edges associated with matrix entries greater than 0.52 to induce sparsity on the graph. To quantitatively evaluate the goodness of E-CGMM, we rely on the Mean Absolute Error (MAE) objective function.

To show that a naive edge discretization technique may not work well, we consider an alternative representation of the molecules where we discretize the continuous edge features using 10 bins of equal widths, so that we can train CGMM using the SP technique.

We used the same model selection and assessment setup of graph classification to get reliable performance estimates. We tried different configurations with depth in $\{10, 20\}$, C_E in $\{5, 10\}$, C_V in $\{10, 20\}$ for CGMM and 20 for E-CGMM, continuous or discrete frozen states, unigrams or unibigrams, and sum or mean aggregation. The MLP configurations, after a preliminary screening on the validation set to reduce the number of configurations, were: 2000 maximum epochs, hidden layer dimension in $\{32, 128\}$, and learning rate equal to $5 \cdot 10^{-4}$. Early stopping's patience was set to 100, and the $L2$ regularization had weight decay 10^{-4}.

Link Prediction

Since E-CGMM can model the generation of edges at each layer, we also tested some link prediction benchmarks, namely Cora, Citeseer and Pubmed [177]. These are three citation networks (i.e., undirected graphs) in which vertices represent documents, and edges represent citations. Because there is no standardized evaluation on these datasets

[3] This value was shown to guarantee better or on-par performances than smaller values during our CGMM experiments, therefore we fixed it to reduce the already large number of configurations to try.

when it comes to link prediction, the goal of these experiments will be to just show how E-CGMM can tackle link prediction by design, with better performances than CGMM.

To adopt a robust experimental protocol even in this third task, we perform 10-fold cross validation for model assessment, with an hold-out technique for model selection. Yet, the difference with the previous tasks lies in how we build each outer fold. In this case, portions of positive edges in each graph are used as validation and test sets, together with a randomly selected subset of negative edges. The remaining edges are used to train both our models. Since CGMM does not produce edge embeddings, we used an MLP to predict whether a link (u, v) exists starting from the mean embedding $\boldsymbol{h}_{uv} = (\boldsymbol{h}_u + \boldsymbol{h}_v)/2$. In particular, we chose the mean operator over concatenation because the latter would introduce asymmetries, i.e., learning difficulties, when modeling the existing undirected edges. To infer the existence of an undirected link using E-CGMM, we construct the mean embedding $\boldsymbol{h}_{uv}$ at each layer using the posterior distributions of Q_{uv} and Q_{vu}.

For every dataset and for both models, we choose the following hyper-parameters: $C_V \in \{10, 20\}$ and number of layers in $\{2, 4, 6, \ldots, 20\}$, $C_E \in \{5, 10\}$ (E-CGMM only), discrete or continuous vertex representations, and unigrams or unibigrams. For the MLP, we chose the learning rate in $\{10^{-3}, 10^{-4}, 10^{-5}\}$, the hidden dimension in $\{128, 256\}$, and weight decay in $\{10^{-3}, 10^{-5}\}$.

4.2.6 Results

We now present our empirical findings starting from graph classification to graph regression and link prediction. The goal of this section is simply to show that E-CGMM almost consistently improves the metrics of interest.

Graph Classification

Chemical and social graph classification results are detailed in Tables 4.8 and 4.9. They show how E-CGMM is basically on par with CGMM on those tasks where the baseline is able to get near to or better than the state of the art, but it improves on the others. Notably, there is a substantial gap on NCI1 and both REDDIT tasks. We believe such improvements are attributable to the new capabilities introduced with the edge component of each layer, i.e., the ability to dynamically aggregate neighbors, in a way that explicitly and adaptively depends on the local connections, and the "new" graph embedding enriched with global edge information.

Table 4.8: Mean and standard deviation results on chemical datasets of a 10-fold Cross Validation (setup of Section 3.3). Best results are reported in bold.

	D&D	**NCI1**	**PROTEINS**
Baseline	**78.4** $\pm$ 4.5	69.8 $\pm$ 2.2	**75.8** $\pm$ 3.7
DGCNN	76.6 $\pm$ 4.3	76.4 $\pm$ 1.7	72.9 $\pm$ 3.5
DiffPool	75.0 $\pm$ 3.5	76.9 $\pm$ 1.9	73.7 $\pm$ 3.5
ECC	72.6 $\pm$ 4.1	76.2 $\pm$ 1.4	72.3 $\pm$ 3.4
GIN	75.3 $\pm$ 2.9	**80.0** $\pm$ 1.4	73.3 $\pm$ 4.0
GraphSAGE	72.9 $\pm$ 2.0	76.0 $\pm$ 1.8	73.0 $\pm$ 4.5
CGMM	74.9 $\pm$ 3.4	76.2 $\pm$ 2.0	74.0 $\pm$ 3.9
E-CGMM	73.9 $\pm$ 4.1	78.5 $\pm$ 1.7	73.3 $\pm$ 4.1

Table 4.9: Mean and standard deviation results on social datasets of a 10-fold Cross Validation (setup of Section 3.3). Best results are reported in bold. Note that the degree is the sole vertex feature used by all models.

	IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
Baseline	70.8 $\pm$ 5.0	**49.1** $\pm$ 3.5	82.2 $\pm$ 3.0	52.2 $\pm$ 1.5	70.2 $\pm$ 1.5
DGCNN	69.2 $\pm$ 3.0	45.6 $\pm$ 3.4	87.8 $\pm$ 2.5	49.2 $\pm$ 1.2	71.2 $\pm$ 1.9
DiffPool	68.4 $\pm$ 3.3	45.6 $\pm$ 3.4	89.1 $\pm$ 1.6	53.8 $\pm$ 1.4	68.9 $\pm$ 2.0
ECC	67.7 $\pm$ 2.8	43.5 $\pm$ 3.1	-	-	-
GIN	71.2 $\pm$ 3.9	48.5 $\pm$ 3.3	**89.9** $\pm$ 1.9	**56.1** $\pm$ 1.7	75.6 $\pm$ 2.3
GraphSAGE	68.8 $\pm$ 4.5	47.6 $\pm$ 3.5	84.3 $\pm$ 1.9	50.0 $\pm$ 1.3	73.9 $\pm$ 1.7
CGMM	**72.7** $\pm$ 3.6	47.5 $\pm$ 3.9	88.1 $\pm$ 1.9	52.4 $\pm$ 2.2	77.32 $\pm$ 2.2
E-CGMM	70.7 $\pm$ 3.8	48.3 $\pm$ 4.1	89.5 $\pm$ 1.3	53.7 $\pm$ 1.0	**77.45** $\pm$ 2.3

Graph Regression

Table 4.10 reports our graph regression analysis. We can see that E-CGMM performs better than both versions of CGMM, i.e., one that ignores edge attributes and the other working on discretized edge labels. In particular, there is a relative improvment in MAE of 17%-19% that was to be expected, given the importance of the information contained in the Coulomb interaction matrix. Such an improvement also proves the inadequacy of non-adaptive edge discretization techniques, which not only may force the user to take decisions a-priori but might also cause loss of relevant information.

	MAE	Relative Improvement
CGMM-no edge attributes	1.52 $\pm$ 0.05	19%
CGMM-discretized edges	1.49 $\pm$ 0.07	17%
E-CGMM	**1.23** $\pm$ 0.06	-

Table 4.10: Graph regression results and relative improvement of E-CGMM compared to CGMM. Best results are in bold. CGMM results are reported for both a version of the dataset with no edge attributes as well as for discretized edge labels.

Link Prediction

We conclude our analysis with link prediction experiments, summarized in Table 4.11. The numbers indicate a substantial improvements with respect to CGMM on every dataset tested, with an average accuracy increase of 3%-4%. By modeling the generation of positive and negative edges, E-CGMM captures the conditional distribution of the edges given the frozen vertex states, thus building more informative edge posteriors. On the other hand, the way in which we have built edge representations with CGMM is yet another a-priori choice that we have managed to avoid with this new method.

	Cora	Citeseer	Pubmed
CGMM	82.62 ± 1.8	74.47 ± 2.2	77.09 ± 1.9
E-CGMM	$\mathbf{86.76} \pm 2.3$	$\mathbf{77.69} \pm 1.7$	$\mathbf{81.58} \pm 1.6$

TABLE 4.11: Comparison between E-CGMM and CGMM on link prediction tasks. Best results are reported in bold.

4.2.7 Summary

We have extended our fully probabilistic framework with an "edge-aware" version of the Contextual Graph Markov Model. E-CGMM allows us the process a broader class of graphs with potentially arbitrary edge features. To achieve this goal, we took an architectural approach by introducing an additional Bayesian network responsible for the generative modeling of edge features. Thanks to the richer graph embeddings produced, we have observed empirical improvements with respect to CGMM on three different tasks, while keeping the asymptotic complexity linear in the number of edges.

It is worth noticing that many of the future directions lied down in Section 4.1.11 would implicitly apply here, as they involve modifying the Bayesian network rather than architectural aspects. Combined with the benefits of explicitly modeling edge features, we believe there is still room for improvement on both technical and empirical sides.

4.3 The Infinite Contextual Graph Markov Model

As with most Deep Graph Networks, one inherent limitation of CGMM is the absence of a mechanism to learn the size of each layer's latent representation. This is also related to one of the most challenging problems of machine learning, that is, the selection of appropriate hyper-parameters for the task at hand. In fact, due to the data-dependent nature of the learning problem, we have seen throughout this thesis that there exists no single model configuration that works well in every situation. One usually relies on model selection techniques such as grid or random searches [236], where the hyper-parameters configurations to try are chosen a-priori by the machine learning practitioner.

In Chapter 2, however, we have briefly introduced Bayesian nonparametric methods, in particular HDP mixture models, that automatically choose the "right" amount of clusters to use [27]. We recall that, in the BNP literature, the complexity of the models, e.g., the number of states, automatically grows *with the data* [24]. Since each CGMM is essentially a conditional mixture model, it would make sense to apply a BNP treatment to CGMM in order to automatize the choice of its hyper-parameters.

In this section, we present our last methodological contribution to the family of Deep Bayesian Graph Networks. The principal difficulty of extending CGMM lies in how to handle the variable-size number of neighbors of each vertex inside the BNP framework, which in CGMM is solved by (possibly weighted) convex combinations of the frozen neighbors' posteriors. We shall see how the notion of a neighboring **macro-state** will be particularly useful in this context. It is thanks to this realization that we will be able to replace the CGMM layer with an HDP mixture model, without incurring in major technical challenges.

The resulting model, called Infinite Contextual Graph Markov Model (ICGMM), can generate as many latent states as needed to solve the unsupervised density estimation task at each layer. To the extent of our knowledge, this is the first **deep**, **Bayesian nonparametric** model for **graph processing**. To increase its efficiency, and despite the existence of variational inference alternatives [237–240], we opted for a straightforward and faster heuristic that scales to the social datasets considered in this thesis, requires little code modification, and works as well as the original implementation.

We compare ICGMM against CGMM, E-CGMM and end-to-end supervised methods on the graph classification tasks of Section 4.1.9. Results show that ICGMM performs on par or better than CGMM. We complement the analysis with studies on the effects of depth and generation of our model's latent states. All in all, we believe that ICGMM is an important (if not the first) step towards a theoretically grounded and automatic construction of Deep Bayesian Graph Networks.

4.3.1 Layer Definition

There are different ways to define an HDP mixture model, but we will mainly use the stick-breaking construction of Section 2.1.4.2. Nevertheless, another representation exists, called Chinese Restaurant Franchise (CRF), which will be needed in the following. The CRF extends the CRP to hierarchical models, by assuming that there are $\bar{C}$ "restaurants", i.e., the known groups assigned to observations in an HDP. Each observation, in our case the variable X_u, is called a "customer". Following the HDP mixture model literature, each vertex u must be already assigned to one of the $\bar{C}$ different groups. Hence, we will use the term n_j to indicate the number of customers eating at restaurant j, i.e., given a graph g it must hold $\sum_{j=1}^{\bar{C}} n_j = |\mathcal{V}_g|$. In addition, a latent state c, modeled by the mixture variable Q_u, will be assigned to each vertex.

The value that Q_u takes, namely q_u, specifies one of the emission components of the **possibly infinite** mixture model, which is parametrized by $\boldsymbol{\theta}_c, c \in \mathbb{N}$. Continuing with the CRF metaphor, we can say that a customer u goes to restaurant j_u and sits at one of the **tables** t_u where **dish** $q_u = c$ is served. Importantly, we assume stationarity of the emission distributions with respect to the groups, meaning there is a form of parameter sharing of the emission distributions **across different groups**. It is appropriate to remark that the assignment of a customer u to a specific table t_u is unnecessary in the Stick-breaking formulation we will use, but the CRF notion of "table assignment" will provide an exact and efficient way to solve some technical challenges in the implementation of the model. Finally, please recall from Section 2.1.4.1 that even if the number of the possible latent states is infinite, **only a finite** number of them will be used in the model's implementation. Hence, we refer to this finite value of clusters with the usual symbol C.

The graphical model of a generic iCGMM layer ℓ is shown in Figure 4.10, where most of the notation has already been described in Section 2.15. We model the generative process of the observable vertex feature X_u conditioned on a set of observable variables of neighboring vertices $\boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1} = \{\boldsymbol{Q}_v^{\ell-1} \in [0,1]^{\bar{C}} \mid v \in \mathcal{N}_u\}$, i.e., the usual vectors of probabilities inferred and frozen at the previous layer. It follows that in iCGMM each layer has a different number of groups $\bar{C}_\ell$; when clear from the context, we will omit the symbol ℓ to ease the notation.

Overall, the generative process of a single iCGMM layer can be formalized as follows:

$$\begin{aligned}
\boldsymbol{\beta} \mid \gamma &\sim \text{Stick}(\gamma) & j_u \mid \boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1} &= \psi(\boldsymbol{Q}_{\mathcal{N}_u}^{\ell-1}) \\
\boldsymbol{\pi}_j \mid \boldsymbol{\beta}, \alpha_0 &\sim \text{DP}(\alpha_0, \boldsymbol{\beta}) & q_u \mid j_u, (\boldsymbol{\pi}_j)_{j=1}^{\bar{C}} &\sim \boldsymbol{\pi}_{j_u} \\
\boldsymbol{\theta} \mid \boldsymbol{H} &\sim \boldsymbol{H} & x_u \mid q_u, (\boldsymbol{\theta})_{c=1}^{\infty} &\sim F(\boldsymbol{\theta}_{q_u}),
\end{aligned} \tag{4.7}$$

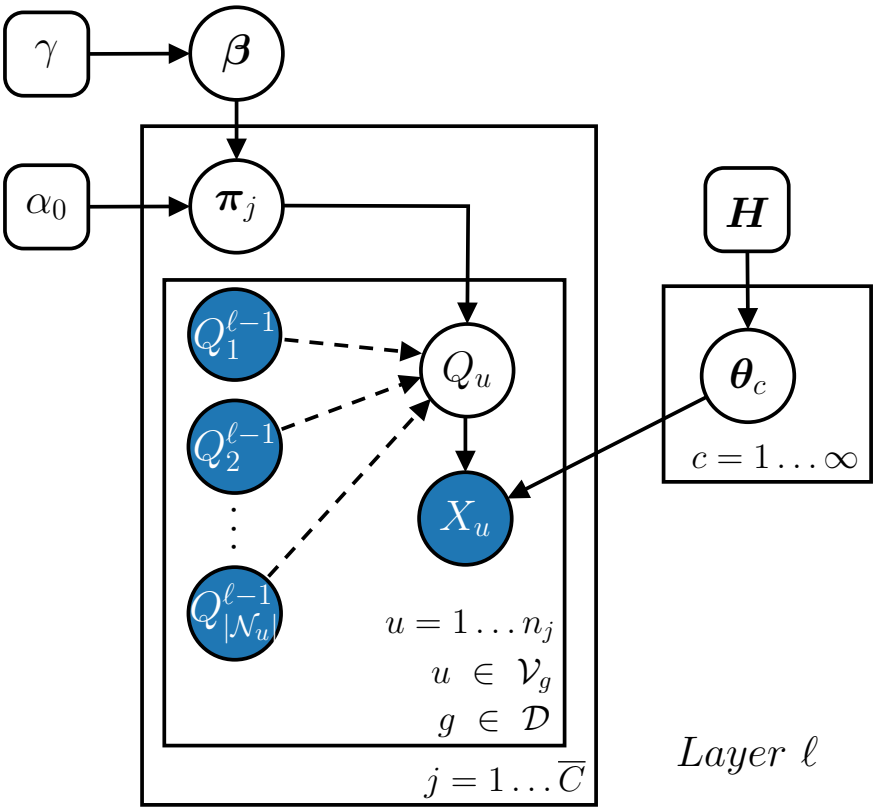


FIGURE 4.10: Graphical model of a generic ICGMM layer ℓ, where observable variables are blue circles, latent ones are white ones, and white boxes denote prior knowledge. This is an HDP mixture model where the group j_u for each observation X_u is determined by the set of neighboring states. The symbol $\bar{C}$ corresponds to the selected number of states determined at the previous layer of the deep incremental architecture. Dashed arrows denote the flow of contextual information.

where $F(\boldsymbol{\theta}_{q_u})$ denotes the emission distribution parametrized by $\boldsymbol{\theta}_{q_u}$, and j_u is the group of vertex u chosen according to a permutation invariant function $\psi(\boldsymbol{Q}_{\mathcal{N}_u})$. To generate a possibly infinite number of emission distributions, we sample from a prior distribution $\boldsymbol{H}$. Instead, we sample the distribution $\boldsymbol{\beta}$ via the Stick-breaking process Stick(γ) (Section 2.1.4.1). In turn, $\boldsymbol{\beta}$ is used by a DP to generate the distribution $\boldsymbol{\pi}_j$, responsible for sampling the dish q_u that customer u eats at restaurant j_u.

Deterministic Choice of the Group

The crux of the matter is that we need a sensible way to assign each observation to one of the $\bar{C}$ groups. Contrarily to a standard HDP, in which the groups are somehow known, here we cannot rely on a-priori information to make the assignments at each layer. The reason is two-fold: i) we would need an oracle for the layer-wise assignments, since these do not have a straightforward interpretation in our context; ii) if we fixed in advance the group for each observation for all layers, there would be no contextual information to spread across the graph, and therefore all HDPs would be truly independent between

each other. It follows that, to induce a dependency between subsequent layers, we must again rely on the frozen states of each vertex as CGMM. These states are indeed the only actionable information we have in order to propagate information across the graph.

The question now becomes how to obtain a group from the set of frozen states of each vertex. In an attempt to be as similar as possible to CGMM, we can reuse the concept of **macro-state** introduced in the previous sections. By considering the simplest version of a CGMM layer, i.e., the one without the SP variables, we can take the mean of the neighboring states and assign vertex u to the most likely position j_u in the resulting macro-state vector. Two important observations follow:

- The j_u is chosen **deterministically**, as in standard HDPs, because the states have been frozen in advance;
- The value of j_u changes at each layer according to the distributions of neighboring states. In other words, j_u is the **sole** responsible for information propagation at each layer.

Formally, to exploit the structural information of the graph and to stick as much as possible to the original CGMM formalism, we chose to select j_u according to this straightforward rule:

$$j_u = \psi(\boldsymbol{Q}^{\ell-1}_{\mathcal{N}_u}) = \underset{j \in \{1,\dots,\bar{C}_\ell\}}{\arg\max} \left(\frac{1}{|\mathcal{N}_u|} \sum_{v \in \mathcal{N}_u} \boldsymbol{Q}^{\ell-1}_v \right)_j . \tag{4.8}$$

Thanks to Equation 4.8, vertices with the same features may have a different latent state c, due to the fact that they are assigned to different groups, i.e., different $\boldsymbol{\pi}_j$, on the basis of their neighborhood. Again, this mimics the role of the CGMM transition distribution but in an HDP.

If we wanted to reason using the CRF jargon, Eq. 4.8 could be equivalent to have a customer go at the restaurant that was recommended the most by the customer's friends. As in [7], we chose to average the parameters of the richer distributions $\boldsymbol{Q}^{\ell-1}_v, \forall v \in \mathcal{N}_u$ rather than perform majority voting amongst the most likely state of every neighbor. Still, notice that the former choice reduces to the latter when the discrete distributions collapse all their probability mass into a single state (one-hot representation).

Finally, and similarly to CGMM, the very first layer of iCGMM is just an HDP mixture model with one group, as no neighboring states have been inferred yet. The reason why we did not choose a simpler DP mixture model is that an HPD tends to generate a fewer number of latent states in our experiments.

On Exchangeability

Every ICGMM relies on the *exchangeability* assumption of DPs to be trained. Recall that exchangeability (informally) states that the observations x_u of our dataset are not independent but the order in which we look at them does not matter [25]. Exchangeability is trivially satisfied in ICGMM, because the observations X_u are assumed to be i.i.d. when conditioned on the neighboring states.

Summing up, we depart from the basic CGMM layer in more than one way. First and foremost, we do not parametrize nor learn the CGMM transition distribution, which was responsible for the convex combination of neighboring states when computing the E-step of the EM algorithm. Instead, we rely on the most probable choice of the group j_u that is encoded by the neighbors' macro-state. Secondly, due to the sheer complexity of the Bayesian nonparametric treatment, we do not train the model via EM as done with CGMM and E-CGMM; instead, we will exploit Gibbs sampling to compute the quantities of interest. Nonetheless, apart from the conceptual similarities, ICGMM retains one important characteristic of CGMM, i.e., it prevents vanishing gradient effects and over-smoothing by default [7], thus allowing us to construct deeper architectures that freely propagate contextual information.

4.3.2 Inference

The inference phase determines the latent state of u and updates the ICGMM's parameters. This happens at each iteration of the HDP Gibbs sampling algorithm [23, 24, 241]. Note that it is also possible to iteratively estimate the hyper-parameters α_0 and γ: whenever that is the case, we shall append a subscript *"auto"* to our model's name. We start with the latent indicator variable q_u, which is sampled from the following conditional probability

$$\boldsymbol{q}_u^{\ell}(c) = P(q_u = c \mid j_u = j, \boldsymbol{Q}^{-u}, \boldsymbol{\beta}, \boldsymbol{\theta}, \boldsymbol{x}) \propto (\alpha_0 \beta_c + n_{jc}^{-u}) f(x_u \mid \theta_c), \quad c \in \{1, \dots, C+1\}, \tag{4.9}$$

where C denotes the number of current states in the mixture model, f is the *p.d.f.* associated with F, $\boldsymbol{Q}^{-u}$ is the set of latent states assigned so far to each vertex, and the distribution $\boldsymbol{\pi}_j$ has been integrated out [24]. Here, n_{jc}^{-u} indicates the number of observables associated with latent state c and belonging to group j. Whenever we have that $q_u = C + 1$, we create a new state and sample a new emission distribution $\boldsymbol{\theta}_{C+1}$ from $\boldsymbol{H}$. On the contrary, if at the end of an iteration there are no observables associated with state c, we can remove that state and decrease C by 1. This is how the HDP, and hence ICGMM, varies in complexity to fit the data distribution.

In the HDP stick-breaking representation, we require an auxiliary variable method to sample the base distribution $\boldsymbol{\beta}$ [24]. We therefore introduce the auxiliary variables $\boldsymbol{m} = \{m_{jc} \mid \forall j \in \{1, \ldots, \bar{C}\}, \forall c \in \{1, \ldots, C\}\}$ that need to be sampled in order to compute $\boldsymbol{\beta}$. However, since m_{jc} is dependent on n_{jc}, the sampling step of these variables is very inefficient for large values of n_{jc}, as the probability values are proportional the Stirling number of the first-kind $s(n_{jc}, \cdot)$ [242]. Luckily, we can avoid this step thanks to the CRF formulation, as anticipated. The key point is that the value m_{jc} corresponds to the number of tables where dish $q_u = c$ is served at group j in the CRF representation [24, 241]; thus, we can compute each m_{jc} by simply simulating the table assignments process in addition to the Stick-breaking machinery.

Knowing that customer u is eating dish $q_u = c$, its table assignment t_u can be sampled according to:

$$P(t_u = t \mid q_u = c, j_u = j, \boldsymbol{c}, \boldsymbol{t}^{-u}, \boldsymbol{\beta}, \alpha_0) \propto \begin{cases} \tilde{n}_{jt}^{-u}, & \forall t \text{ s.t. } c_{jt} = c; \\ \alpha_0 \beta_c, & t = t_{new}, \end{cases} \tag{4.10}$$

where $\boldsymbol{t}^{-u}$ represents the tables assigned to each vertex up to now, $c_{jt} \in \boldsymbol{c}$ specifies the dish assigned to table t at restaurant j and $\tilde{n}_{jt}^{-u}$ denotes the number of customers (except u) sitting at table t of restaurant j. Since we know the dish q_u selected by the customer u, there is zero probability that the customer sits to a table where that dish is not served. The creation and deletion of tables is very similar to that of Equation 4.9, so we skip it in the interest of the exposition and refer to the pseudocode at the end of the section for a complete treatment. Practically speaking, these auxiliary variables are counters that can be updated in parallel to the Stick-breaking implementation.

After computing m_{jc}, i.e., $m_{jc} = \sum_{t'} \mathbb{I}[c_{jt'} = c]$, the base distribution $\boldsymbol{\beta}$ is updated using:

$$\boldsymbol{\beta} \mid \boldsymbol{Q}, \boldsymbol{m} \sim \text{Dir}(\sum_{j=1}^{\bar{C}} m_{j1}, \ldots, \sum_{j=1}^{\bar{C}} m_{jC}, \gamma), \tag{4.11}$$

where Dir stands for the Dirichlet distribution and $\boldsymbol{Q}$ is the set of latent states assigned to the vertices. The last step of the Gibbs sampling aims to update the emission parameters $\boldsymbol{\theta}$ using its posterior given the observable variables:

$$P(\boldsymbol{\theta}_c \mid \boldsymbol{Q}, \boldsymbol{x}) \propto h(\boldsymbol{\theta}_c) \prod_{\forall u \mid q_u = c} f(x_u \mid \boldsymbol{\theta}_c). \tag{4.12}$$

By choosing the family of the base distribution $\boldsymbol{H}$ to be a conjugate prior for F, e.g., a Dirichlet distribution for Categorical emissions or a Normal-Gamma distribution for

Normal emissions, we can compute the posterior in closed form using the usual data statistics, summarized below.

Posterior of the Emission Distribution

We consider the two cases of a discrete and continuous vertex feature.

Categorical Emission Let the emission distribution be a categorical distribution with K possible states. When creating a new state, we can sample the emission parameter according to a Dirichlet distribution, which is a conjugate prior for the categorical distribution:

$$\theta_c \sim \mathrm{Dir}(\eta, \ldots, \eta), \tag{4.13}$$

where the subscript c indicates the new mixture component. Thanks to the conjugate prior, the emission parameters can be updated by sampling its Dirichlet posterior distribution:

$$\theta'_c \sim \mathrm{Dir}(\eta + N_c^1, \ldots, \eta + N_c^K), \tag{4.14}$$

where N_c^k indicates the number of times the observed label k has been associated with the latent state c, i.e., $N_k^c = \sum_u \mathbb{I}[q_u = c \wedge x_u = k]$.

Gaussian Emission Similarly to the categorical case, let the emission distribution be an univariate Gaussian. In this case, for each new state, we can sample the emission parameter according to a Normal-Gamma distribution:

$$\mu_c \sim \mathcal{N}(\mu_0, 1/(\lambda_0 \tau_c)) \tag{4.15}$$

$$\tau_c \sim \mathrm{Gamma}(a_0, b_0), \tag{4.16}$$

where the subscript c indicates a mixture component ant τ_c is the inverse of the variance. Then, the emission parameters of the Gaussian can be updated as follows:

$$\mu'_c \sim \mathcal{N}\left(\frac{\lambda_0 \mu_0 + N_c \bar{x}_c}{\lambda_0 + N_c}, \frac{1}{(\lambda_0 + N_c)\tau'_c}\right) \tag{4.17}$$

$$\tau'_c \sim \mathrm{Gamma}\left(a_0 + \frac{N_c}{2}, b_0 + \frac{1}{2}\left(N_c s_c + \frac{\lambda_0 N_c (\bar{x}_c - \mu_0)^2}{\lambda_0 + N_c}\right)\right), \tag{4.18}$$

where N_c indicates the number of observed labels associated with the latent state c (i.e., $N_c = \sum_u \mathbb{I}[q_u = c]$), $\bar{x}_c$ is the mean of the data associated with the class c (i.e., $\bar{x}_c = \frac{1}{N_c} \sum_{\forall u | q_u = c} x_u$), and s_c is the variance of the data associated with the class c (i.e., $s_c = \frac{1}{N_c} \sum_{\forall u | q_u = c} (x_u - \bar{x}_u)^2$).

Sampling α_0 and γ

Following [24], the concentration parameter α_0 and γ can be updated between Gibbs sampling iterations by exploiting an auxiliary variable schema. Let us start with the former, by assuming that α_0 has a Gamma prior distribution Gamma$(a, rate = b)$ (i.e., $\alpha_0 \sim$ Gamma(a, b)). Then, we define the auxiliary variables $w_1, \ldots, w_{\bar{C}}$ and $s_1, \ldots, s_{\bar{C}}$, where each w_j variable takes a value between 0 and 1, and each s_j is a binary variable. Then, the value of α_0 can be sampled according to the following schema:

$$w_j \sim \text{Beta}(\alpha_0 + 1, n_{j.}), \tag{4.19}$$

$$s_j \sim \text{Bernoulli}\left(\frac{n_{j.}}{n_{j.} + \alpha_0}\right), \tag{4.20}$$

$$\alpha_0 \sim \text{Gamma}\left(a + m_{..} - \sum_{j=1}^{\bar{C}} s_j, b - \sum_{j=1}^{\bar{C}} \log w_j\right), \tag{4.21}$$

where $n_{j.}$ is the number of costumer eating in the j-th restaurant, and $m_{..}$ is the total number of tables in all the restaurants.

Similarly, assuming that the hyper-parameter γ has a gamma prior distribution, i.e., $\gamma \sim$ Gamma(a', b'), then its value can be updated by following the auxiliary variable schema below [24, 242]:

$$r \sim \text{Beta}(\gamma + 1, m_{..}), \tag{4.22}$$

$$p \sim \text{Bernoulli}\left(\frac{m_{..}}{m_{..} + \gamma}\right), \tag{4.23}$$

$$\gamma \sim \text{Gamma}(a' + C - p, b' - \log r). \tag{4.24}$$

After training for a number of Gibbs sampling iterations, we predict the latent states for unseen data points by simply applying Equation 4.9, where all the statistics, e.g., $n_{jc}\ \forall j, c$, have been stored after training and never updated again.

To facilitate the practical understanding of our model, Algorithm 4 provides the pseudocode of the Gibbs sampling method employed by iCGMM.

4.3.3 Faster Inference with Vertex Batches

Due to the sequential nature of the above inference process, a naive implementation is slow when applied to the larger social graphs considered in this thesis. In the literature, there exist several exact distributed inference methods for the HDP [243–246]), but their effectiveness might be limited due to the unbalanced workload among workers or the

Algorithm 4 Gibbs sampling method for exact ICGMM

Require: A dataset of graphs $\mathcal{D} = \{g_1, \ldots, g_N\}$. Initialize $C = 1$, $\boldsymbol{\theta} = \{\theta_1\}$ (where $\theta_1 \sim H$), $\mathcal{T}_j = \emptyset$ (for all groups j), $\boldsymbol{q} = \boldsymbol{t} = \boldsymbol{c} = \perp$, and $\boldsymbol{n} = \tilde{\boldsymbol{n}} = \mathbf{0}$.

repeat
  for $g \in \mathcal{D}$ **do** ▷ For each graph
    for $u \in \mathcal{V}_g$ **do** ▷ For each vertex
      // assign the group
      $j_u \leftarrow \psi(\boldsymbol{q}_{\mathcal{N}_u})$ ▷ Can be done once $\forall u$
      // assign the dish
      $n_{j_u q_u} \leftarrow n_{j_u q_u} - 1$ ▷ If $q_u \neq \perp$, remove q_u from the counting
      $q_u \leftarrow \text{SAMPLING}(j_u, \boldsymbol{n}, \boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{\beta}, \alpha_0)$ ▷ Sample the dish according to Eq. 4.9
      if q_u is new **then** ▷ Create a new state
        $\theta_{\text{new}} \sim H$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \cup \{\theta_{\text{new}}\}$
        $C \leftarrow C + 1$
        $n_{jq_u} \leftarrow 0 \quad \forall j \in \{1, \ldots, \bar{C}\}$ ▷ Initialize the counters
      end if
      $n_{j_u q_u} \leftarrow n_{j_u q_u} + 1$ ▷ Update the counter
      // assign the table
      $\tilde{n}_{j_u t_u} \leftarrow \tilde{n}_{j_u t_u} - 1$ ▷ If $t_u \neq \perp$, remove t_u from the counting
      $t_u \leftarrow \text{SAMPLING}(j_u, q_u, \boldsymbol{c}, \tilde{\boldsymbol{n}}, \boldsymbol{\beta}, \alpha_0)$ ▷ Sample the table according to Eq. 4.10
      if t_u is new **then** ▷ Create a new table
        $\mathcal{T}_j \leftarrow \mathcal{T}_j \cup \{t_u\}$
        $c_{j_u t_u} \leftarrow q_u$ ▷ Save the dish-table assignment
        $m_{j_u q_u} \leftarrow m_{j_u q_u} + 1$ ▷ Update the table count
        $\tilde{n}_{j_u t_u} \leftarrow 0$ ▷ Initialize customer counter
      end if
      $\tilde{n}_{j_u t_u} \leftarrow \tilde{n}_{j_u t_u} + 1$
    end for
  end for
  // remove unused dishes
  for $c \in \{1, \ldots, C\}$ **do**
    if $\sum_{j=1}^{\bar{C}} n_{jc} = 0$ **then** ▷ No customers eat the dish c
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \setminus \{\theta_c\}$
      $C \leftarrow C - 1$
    end if
  end for
  // remove empty tables
  for $j \in \{1, \ldots, \bar{C}\}$ **do**
    for $t \in \mathcal{T}_j$ **do**
      if $\tilde{n}_{jt} = 0$ **then** ▷ No customers eat at the table t in the restaurant j
        $\mathcal{T}_j \leftarrow \mathcal{T}_j \setminus \{t\}$
        $m_{jc_{jt}} \leftarrow m_{jc_{jt}} - 1$
      end if
    end for
  end for
  // update model parameters
  $\boldsymbol{\beta} \leftarrow \text{SAMPLING}(\boldsymbol{q}, \boldsymbol{m})$ ▷ Sample according to Eq. 4.11
  $\boldsymbol{\theta} \leftarrow \text{SAMPLING}(\boldsymbol{q}, \boldsymbol{x})$ ▷ Sample according to Eq. 4.12
  if ICGMM$_{auto}$ **then**
    $\alpha_0 \leftarrow \text{SAMPLING}(a, b, \boldsymbol{n})$ ▷ Sample according to Eq. (4.19), (4.20), (4.21)
    $\gamma \leftarrow \text{SAMPLING}(a', b', \boldsymbol{m})$ ▷ Sample according to Eq. (4.22), (4.23), (4.24)
  end if
until stopping criteria

elevated rejection rate [247]. Similarly, there are variational inference approximations [237–240] that substantially differ from the approach taken here, but their investigation will be subject of future works.

We prefer to speed-up the inference procedure by introducing a straightforward heuristic rather than relying on an exact distributed computation. As suggested in [247], an approximated inference procedure may indeed suffice for many problems, so what we propose is to perform sampling for a batch of vertex observations altogether. This way, the necessary statistics are updated in batch rather than individually, and matrix operations can be used to gain efficiency.

To keep the quality of the approximation as close as possible to the original Gibbs Sampling algorithm, we choose 1 graph as the size of the batch. Such a trade-off provides a CPU speedup of up to 60× at training time, and we empirically observed that performances remain unchanged w.r.t. the original version on the smaller chemical tasks considered so far. While this faster version of iCGMM, which we call iCGMM$_f$, does not strictly adhere to the technical specifications of the previous Section, we believe that the pros largely outperform the cons. Table 4.12 reports the speedup gains on some tasks by comparing the same configurations.

		iCGMM	iCGMM$_f$
		ref.	**min/max**
Chem.	D&D	1×	17.8×/30.8×
	NCI1	1×	3.1×/5.1×
	PROTEINS	1×	4.2×/5.7×
Social	IMDB-B	1×	2.4×/5.1×
	IMDB-M	1×	1.6×/3.6×
	REDDIT-B	1×	11.1×/45.6×
	REDDIT-5K	1×	36.7×/60.6×
	COLLAB	1×	3.1×/8.6×

Table 4.12: Approximate minimum and maximum speedup across different configurations between the exact iCGMM and the faster version of iCGMM.

4.3.4 Limitations

It should be clear by now that, due to the complexity of the BNP treatment, one limitation is that naive Gibbs sampling does not scale easily to very large datasets. Yet, the vertex independence assumption made by CGMM enables a faster batch computation, which can also be run on a GPU.

The second limitation of iCGMM is that edge features are not taken into account, differently from CGMM and E-CGMM. One of our research directions for the future will be to investigate a potential extension of iCGMM to discrete edge features, perhaps by still using the SP variables in a fully Bayesian fashion.

4.3.5 Experimental Setting

Similarly to the previous models, we evaluated the performances of iCGMM using the fair, robust, and reproducible evaluation setup for graph classification defined in Section 3.3. We first tested the "exact" and faster Gibbs sampling versions of iCGMM on the three chemical datasets D&D, NCI1, and PROTEINS. When considering social datasets, instead, we only evaluated iCGMM$_f$ due to its speedup on larger graphs.[4]

We have discussed how iCGMM can automatize the choice its hyper-parameters, e.g., the size of the latent representation. In general, the choice of the Bayesian hyper-parameters is much less important than that of the number of states C, as in principle one can recursively introduce hyper-priors over these hyper-parameters [248, 249]. That said, since this is the first work to study HDP methods in the context of graph classification, we both i) explored the hyper-parameter space to best assess and characterize the behaviour of the model and ii) introduced hyper-priors to estimate α_0 and γ at each layer, thus further reducing the need for an extensive model selection. For the chemical tasks, the prior $\boldsymbol{H}$ over the emission parameters $\boldsymbol{\theta}_c$ was the uniform Dirichlet distribution. The range of iCGMM hyper-parameters tried in this case were: number of layers $\in \{5, 10, 15, 20\}$, $\alpha_0 \in \{1, 5\}$, $\gamma \in \{1, 2, 3\}$, unibigram aggregation $\in \{\text{sum}, \text{mean}\}$, and Gibbs sampling iterations $\in \{10, 20, 50\}$. Instead, for the social tasks we implemented a Normal-Gamma prior $\boldsymbol{H}$ over a Gaussian distribution. Here the prior is parametrized by the following hyper-priors: μ_0, the mean vertex degree extracted from the data; λ_0, which is inversely proportional to the prior variance of the mean; and (a_0, b_0), whose ratio $t = \frac{b_0}{a_0}$ represents the expected variance of the data. The iCGMM hyper-parameters here were: number of layers $\in \{5, 10, 15, 20\}$, $\lambda_0 \in \{1\text{e-}6\}$, $a_0 \in \{1.\}$, $b_0 \in \{0.09, 1.\}$, $\alpha_0 \in \{1, 5, 10\}$, $\gamma \in \{2, 5, 10\}$, unibigram aggregation $\{\text{sum}, \text{mean}\}$, and Gibbs Sampling iterations $\in \{100\}$. To further automate learning of iCGMM's unsupervised layers, we place uninformative $Gamma(1, rate = 0.01)$ hyper-priors on both $\alpha_0^\ell, \gamma^\ell$ hyper-parameters. To prevent the model from getting stuck in a local minimum on COLLAB (due to bimodal degree distribution and large variances), we tried $\lambda_0 \in \{1\text{e-}4, 1\text{e-}5\}$.

To conclude, we list the hyper-parameters tried for the one-layer MLP classifier trained on the unsupervised graph embeddings: optimizer $\in \{\text{Adam}\}$, batch size $\in \{32\}$, hidden

[4] https://github.com/diningphil/iCGMM.

	D&D	NCI1	PROTEINS
BASELINE	**78.4** ± 4.5	69.8 ± 2.2	**75.8** ± 3.7
DGCNN	76.6 ± 4.3	76.4 ± 1.7	72.9 ± 3.5
DIFFPOOL	75.0 ± 3.5	76.9 ± 1.9	73.7 ± 3.5
ECC	72.6 ± 4.1	76.2 ± 1.4	72.3 ± 3.4
GIN	75.3 ± 2.9	**80.0** ± 1.4	73.3 ± 4.0
GRAPHSAGE	72.9 ± 2.0	76.0 ± 1.8	73.0 ± 4.5
CGMM	74.9 ± 3.4	76.2 ± 2.0	74.0 ± 3.9
E-CGMM	73.9 ± 4.1	78.5 ± 1.7	73.3 ± 4.1
ICGMM	75.6 ± 4.3	76.5 ± 1.8	72.7 ± 3.4
ICGMM$_f$	75.0 ± 5.6	76.7 ± 1.7	73.3 ± 2.9
ICGMM$_{auto}$	76.3 ± 5.6	77.6 ± 1.5	73.1 ± 3.9
ICGMM$_{fauto}$	75.1 ± 3.8	76.4 ± 1.4	73.2 ± 3.9

TABLE 4.13: Results on chemical datasets (mean accuracy and standard deviation) are shown. The best performances are highlighted in bold.

units ∈ {32, 128}, learning rate ∈ {1e-3}, L2 regularization ∈ {0., 5e-4}, epochs ∈ {2000}, ReLU activation, and early stopping on validation accuracy with patience 300 on chemical tasks and 100 on social ones.

4.3.6 Results

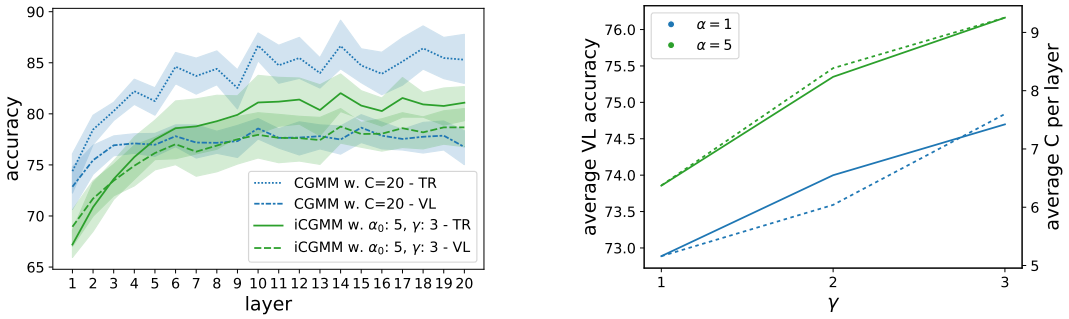
The empirical results on chemical and social benchmarks are reported in Tables 4.13 and 4.14, respectively. There are several observations to be made, starting with the chemical tasks. First of all, ICGMM performs similarly to CGMM, E-CGMM, and most of the *supervised* neural models; this suggests that the selection of j_u based on the neighboring recommendations is a subtle but effective form of information propagation between the vertices of the graph. In addition, results indicate that we have succeeded in effectively automatizing the choice of the number of latent states without compromising the accuracy, which was the main goal of this work. Finally, ICGMM$_f$ performs as well as the exact version, and for this reason we safely applied the faster variant to the larger social datasets (including IMDB-B and IMDB-M to ease the exposition).

Moving to the social datasets, we observe that ICGMM achieves better average performances than other methods on IMDB-B, REDDIT-B and COLLAB. One possible reason for such an improvement with respect to CGMM variants may be how the emission distributions are initialized. On the one hand, and differently from the chemical tasks, CGMM and E-CGMM use the k-means algorithm (with fixed k=C), to initialize the mean values of the C Gaussian distributions, which can be stuck in a local minimum around the most frequent degree values. One the other hand, ICGMM adopts a fully Bayesian treatment, which combined with the automatic selection of the latent states allows to better model outliers by adding a new state when the posterior probability of a data point is too low.

	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K	COLLAB
BASELINE	70.8 ± 5.0	$\mathbf{49.1} \pm 3.5$	82.2 ± 3.0	52.2 ± 1.5	70.2 ± 1.5
DGCNN	69.2 ± 3.0	45.6 ± 3.4	87.8 ± 2.5	49.2 ± 1.2	71.2 ± 1.9
DIFFPOOL	68.4 ± 3.3	45.6 ± 3.4	89.1 ± 1.6	53.8 ± 1.4	68.9 ± 2.0
ECC	67.7 ± 2.8	43.5 ± 3.1	-	-	-
GIN	71.2 ± 3.9	48.5 ± 3.3	89.9 ± 1.9	$\mathbf{56.1} \pm 1.7$	75.6 ± 2.3
GRAPHSAGE	68.8 ± 4.5	47.6 ± 3.5	84.3 ± 1.9	50.0 ± 1.3	73.9 ± 1.7
CGMM	72.7 ± 3.6	47.5 ± 3.9	88.1 ± 1.9	52.4 ± 2.2	77.32 ± 2.2
E-CGMM	70.7 ± 3.8	48.3 ± 4.1	89.5 ± 1.3	53.7 ± 1.0	77.45 ± 2.3
ICGMM$_f$	$\mathbf{73.0} \pm 4.3$	48.6 ± 3.4	91.3 ± 1.8	55.5 ± 1.9	78.6 ± 2.8
ICGMM$_{fauto}$	71.8 ± 4.4	49.0 ± 3.8	$\mathbf{91.6} \pm 2.1$	55.6 ± 1.7	$\mathbf{78.9} \pm 1.7$

TABLE 4.14: Results on social datasets (mean accuracy and standard deviation) are shown, where the vertex degree is used as the only vertex feature. The best performances are highlighted in bold.

Similarly to the analysis done earlier for CGMM, we will try to shed more light into the improved generalization performances of ICGMM, by analyzing the exact model from a layer-wise perspective.



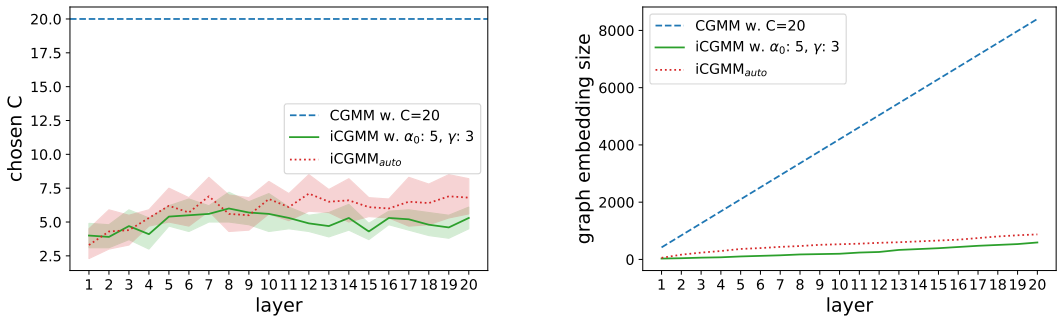
(a) Effect of depth on training/validation accuracy

(b) Average VL accuracy (solid line) and number of chosen states (dashed line) w.r.t α_0 and γ values

FIGURE 4.11: Figures 4.11(a) and 4.11(b) analyze the relation between depth, performances, and the number of chosen states on NCI1.

On the effectiveness of depth and hyper-parameters

To confirm our intuition about the benefits of the proposed information propagation mechanism, Figure 4.11(a) shows the NCI1 training and validation performances of both CGMM and ICGMM as we add more layers. For simplicity, we picked the best ICGMM configuration on the first external fold, and we compared it against the CGMM configuration with the most similar performances. Note that $C = 20$ was the most frequent choice of CGMM states by the best model configurations across the 10 outer folds: this is because having more emission distributions to choose from allows the CGMM model to find better local minima, whereas ICGMM can automatically add states whenever the data point's sampling probabilities are too low. We trained the same classifier at



(a) # states chosen at each layer

(b) Cumulative graph embedding size on NCI1

FIGURE 4.12: We show comparative results on the size and quality of graph embeddings between CGMM and iCGMM. Overall, iCGMM generates ≈ 0 unused latent states, with consequent savings in terms of memory and compute time of the classifier with respect to CGMM. See the text for more details.

different depths, and we averaged scores across the 10 outer folds. The validation performance of both models are similar, with an asymptotic behavior as we reach 20 layers; hence, depth remains fundamental to improve the generalization performances [7]. Importantly, we see that gap between iCGMM training and validation scores is thinner than its non-BNP counterpart, suggesting that there is less overfitting of the data.

We now study how iCGMM behaves as we vary the main hyper-parameters α_0 and γ. We continue our experimentation on NCI1; Figure 4.11(b) depicts the average validation performance and number of states C over all configurations and folds, subject to changes of α_0 and γ values. The trend indicates how greater values for both hyper-parameters achieve, on average, better validation performance. Also, smaller values of the two hyper-parameters tend to strongly regularize the model by creating fewer states, with consequent reduction in validation accuracy.

On the quality of graph embeddings

So far, we have argued that iCGMM selects the appropriate number of states for its unsupervised task at each layer. As a matter of fact, Figure 4.12(a) reports such a statistic on the same NCI1 configuration as before: iCGMM preferred a lower number of latent states than CGMM, i.e., around 5 per layer.In turn, the resulting graph embeddings become much smaller, with important savings in terms of memory footprint and computational costs to train the subsequent classifier. Figure 4.12(b) displays the cumulative graph embedding size across layers, using the unibigram representation without loss of generality. We see that, when compared with CGMM (C=20), the size of graph embeddings produced by iCGMM is approximately 7% of those of the original model, while still preserving the same performance as CGMM.

On the automatic estimation of α^ℓ and γ^ℓ

We conclude this work with a performance analysis of the fully automated versions of iCGMM and iCGMM$_f$, namely those with an *"auto"* subscript in Tables 4.13 and 4.14; in particular, we observe no statistically significant performance differences with respect to the original models. By estimating all hyper-parameters of our models using uninformative priors, we almost always (but for COLLAB) managed to *avoid the model selection* for the unsupervised graph embeddings creation. In turn, this amounted to a $6\times$ reduction in the overall number of configurations to be tried, but most importantly it frees the user from making hard choices about which configurations of hyper-parameters to try. Also, we observe that the number of chosen states and the consequent graph embedding size is very similar to that of iCGMM with $\alpha_0 = 5, \gamma = 3$, but this time the two hyper-parameters have been estimated by the model on the basis of the data.

4.3.7 Summary

The Infinite Contextual Graph Markov model is the last methodological contribution of the chapter. We have shown how to bridge the two distant fields of Bayesian nonparametrics and deep learning for graphs in order to build a DBGN whose complexity grows with the data. iCGMM has demonstrated very competitive performances with respect to the (supervised) state of the art, thanks to an information propagation mechanism that is inspired from the concepts of Chapter 3 but adapted to work with HDPs. Not only does this model automatically select the number of hidden states for each layer, but we can also estimate almost all hyper-parameters at each layer using uninformative hyper-priors. In turn, we can get lower memory and computational footprints without sacrificing the overall predictive performances, at least in the tasks studied so far.

It still remains to be seen whether or not more complex aggregation mechanisms could be applicable to iCGMM. Our attempts at choosing the group j for each observation in a stochastic way, i.e., by sampling from the macro-state distribution at each Gibbs sampling iteration, failed to converge or performed poorly. Moreover, there is the necessity to scale up to larger graphs, which may be achieved by distributed Gibbs sampling or variational inference procedures. We leave these interesting directions to future works, confident that the cross-fertilization of ideas between different fields will further enhance the representational power of Deep Bayesian Graph Networks.

4.4 Application to Malware Classification [11]

To conclude the chapter, we tackle a real-world malware classification problem using the DBGNs introduced so far [11]. The task of detecting malicious behavior using static analysis is indeed one fundamental process to protect devices, networks and users' personal data. By looking at how the program is written, we want to automatically find patterns that allow us to distinguish whether a program is to be *trusted* or belongs to a specific *malware* family.

As anti-malware companies become better at finding known patterns, so do malware writers that rely on *obfuscation* techniques to elude common pattern checks. There are two main categories of obfuscation: **intra-procedural**, i.e., it modifies procedural code without changing the interaction with the rest of the program, or **inter-procedural**, i.e., it alters the structure of the program by also adding *call* or *invoke* statements. While the latter is certainly more difficult to detect, it is also much more delicate to use as some mechanisms (e.g., call-return and parameter passing) may rely on information known only at run-time and can introduce concurrency problems.

Intra-procedural techniques are widely used and suffice to fool a number of static code analysis tools [250]. Recent works test their approaches on the most common obfuscation techniques [251] or group them by their magnitude of edits on the code [252]. In this context, we investigate the problem of malware classification using DBGNs, where the program is represented as a **Call Graph** (CG), i.e., a graph where vertices are procedures and edges denote calls to other procedures. Differently from the literature, we consider obfuscation techniques based on their influence on the CG topology.

Many non-adaptive malware detection solutions based on CGs exploit graph-signatures, similarity algorithms and graph-kernels [253–255]. In conjunction with formal methods, these approaches achieve excellent accuracy, but the analysis is time-consuming and requires domain-level expertise for the temporal logic formulas generation [256]. Instead, most machine learning approaches are generally more efficient and rely on static analysis features included in the graphs, such as opcodes frequencies [257] and control/data dependencies [258], some of which are easily vulnerable to intra-procedural obfuscation.

Our contribution, apart from showing a practical application of the methodologies introduced so far, is to propose a malware classification method based solely on the **CG topology**. This way, it is possible to show that the approach is intrinsically robust to intra-procedural obfuscation techniques. We exploit CGMM, E-CGMM, and iCGMM to construct CG embeddings that are then fed to a machine learning classifier. Note that, while methods exist to certify robustness of DGNs to vertex perturbations [100], our approach does not need such certificates as it only focuses on the structure.

4.4.1 Methodology

We sketch the overall methodology in Figure 4.13. Assuming we have a CG dataset, we employ an unsupervised DBGN to generate graph embeddings encoding CG structural information. We already know from the previous sections that, to enable learning with

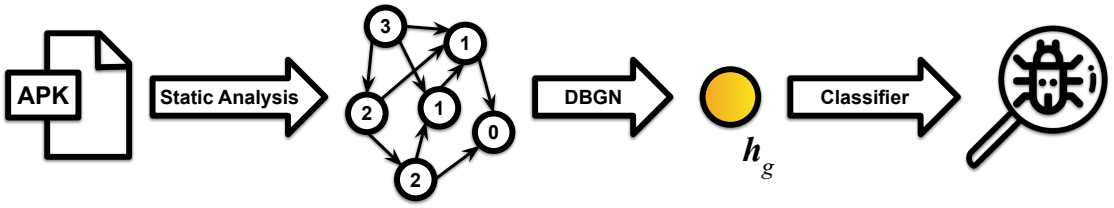


FIGURE 4.13: Given an Android Application Package (APK) , we apply static analysis to construct a CG, where vertices represent methods and arrows denote how methods are intertwined. For our purposes, the sole vertex feature we use is the out-degree of each vertex. Then, the CGMM model transforms the input graph into an embedding that is used for the final classification.

the current DBGNs, vertex features are essential. To avoid relying on features vulnerable to intra-obfuscation, we choose the out-degree of each vertex, which encodes how many calls are made by the caller. Therefore, the emission distribution $P(x_u|Q_u = i)$ will be a univariate Gaussian. Clearly, using the degree feature is just one the possible choices, but it proved to be quite effective so far.

It is worth that the unsupervised training can significantly accelerate the model selection phase, since graph embeddings need be computed only once and the downstream classifier works on simple vectors. Also, the final graph embedding is (again) the concatenation of the aggregated vertex/edge posteriors produced at each layer of the DBGNs developed in this thesis.

4.4.2 Experimental Setting

We now describe how we converted a set of Android applications, i.e., `.apk` files, into a CG dataset; nevertheless, provided a static analysis tool is available, it is straightforward to apply this methodology to other environments as well. First, of all, each `.apk` file is decompressed and the Java bytecode is decompiled into Jimple, an intermediate representation language, using the Soot Framework [259]. During decompilation, the code is analyzed to generate a CG[5], where vertices represent methods, i.e., a procedure or function construct, and **directed** edges denote calls from caller to called vertices, i.e., when an *invoke* or *call* statement is present in the method. Our analysis only considers methods in the application packages, thus discarding calls to library functions

[5]Soot transformation: https://github.com/Djack1010/graph4apk.

or external packages. Notably, the generated CGs do not contain information about the methods statements, e.g., variables, declaration, and dependencies, on the vertices; instead, as already mentioned, we add the out-degree as the sole vertex feature to be able to train the probabilistic models. Hence, our methodology is intrinsically robust to intra-procedural obfuscations techniques, such as code reordering/removal, junk code insertion, instruction substitution, control flow modifications, identifiers and variables renaming/encryption, and repacking [250]. Indeed, these obfuscation techniques modify each method's statements but they do not alter the number of *invoke* or *call* statements, i.e., the initial CG is exactly the same as any intra-procedurally obfuscated CG.

Malware samples were collected from the AMD and previous work datasets [260], and the benign samples were downloaded from Google Play. Both the malware and the trusted applications were verified with VirusTotal, to ensure either their maliciousness or trustiness. The resulting dataset consists of 5669 samples of real-world malware, split into 8 classes, where one represents the trusted software (1762 samples) and the others stand for different malware families, namely *Airpush* (736 samples), *Dowgin* (1040 samples), *FakeInst* (190 samples), *Kuguo* (879 samples), *Youmi* (959 samples), *Fusob* (73 samples), and *Mecor* (30 samples). Dataset statistics are described in Table 4.15.

# Graphs	# Classes	Avg $\lvert\mathcal{V}_g\rvert$	Avg $\lvert\mathcal{E}_g\rvert$	Min Degree	Max Degree	Avg Degree
5669	8	5069	3267	0	618	0.58

TABLE 4.15: Dataset statistics. Graphs are large but sparse, and the average out-degree is low because all calls to external libraries have been removed from the CG.

To assess the performance of the DBGNs on our CG dataset, we split the data according to a stratified hold-out strategy, with 80% of the data for training, 10% for validation and 10% for test.[6] To empirically evaluate the impact of the structure in the dataset, we follow Section 3.3 and introduce a structure-agnostic baseline. The baseline applies an MLP to the vertex features, performs global aggregation and then applies a linear output layer. We performed grid-search model selection for all models, with early stopping monitoring the classification accuracy.

The hyper-parameters tried for the baseline were: hidden units $\in$ {32,64,128}, 2000 epochs, batch size 128, global aggregation $\in$ {sum, mean}, Adam Optimizer with learning rate $\in$ {0.01, 0.001}, patience $\in$ {50}. As regards CGMM, instead, we selected the best model across the following configurations: 20 states, layers $\in \{10, 20\}$, 10 EM epochs, posterior version $\in$ {discrete, continuous}, embedding version $\in$ {unigram, unibigram}, global aggregation $\in$ {sum, mean}, batch size 64, 2000 epochs, hidden units $\in$ {32,64,256, 512}, Adam Optimizer with learning rate $\in$ {0.0001} and weight decay $\in$ {0., 0.0005},

[6] https://github.com/diningphil/robust-call-graph-malware-detection.

	TR Loss	TR Acc.	VL Loss	VL Acc.	TE Loss	TE Acc.
Baseline	$1.2_{\pm 0.05}$	$55.6_{\pm 0.5}$	$1.1_{\pm 0.01}$	$60.6_{\pm 0.9}$	$1.1_{\pm 0.03}$	$56.7_{\pm 0.5}$
CGMM	$0.01_{\pm 0.01}$	$99.8_{\pm 0.4}$	$0.16_{\pm 0.01}$	$97.9_{\pm 0.2}$	$0.13_{\pm 0.01}$	$96.4_{\pm 0.6}$
E-CGMM	$0.03_{\pm 0.02}$	$99.4_{\pm 0.6}$	$0.59_{\pm 0.003}$	$98.4_{\pm 0.3}$	$0.19_{\pm 0.02}$	$\mathbf{97.3}_{\pm 0.4}$
iCGMM$_f$	$0.05_{\pm 0.01}$	$98.7_{\pm 0.5}$	$0.27_{\pm 0.04}$	$94.8_{\pm 0.5}$	$0.35_{\pm 0.03}$	$93.6_{\pm 0.6}$
iCGMM$_{f_{auto}}$	$0.07_{\pm 0.03}$	$97.93_{\pm 0.9}$	$0.25_{\pm 0.02}$	$95,8_{\pm 0.5}$	$0.42_{\pm 0.1}$	$92.7_{\pm 0.5}$

Table 4.16: Malware classification results (mean and standard deviation) on training (TR), validation (VL) and test (TE) sets. We display both the Cross-Entropy loss as well as the multi-class accuracy. Results are averaged over 3 final runs.

and patience 100. E-CGMM shares the same hyper-parameters' set but for C_E in $\{5, 10\}$. Finally, we tried the same iCGMM$_f$ and iCGMM$_{f_{auto}}$ embedding configurations of Section 4.3.5 (but for $\lambda_0 \in \{$1e-4,1e-5$\}$), with the μ_0 being set to 0.58, i.e., the empirical mean out-degree of the dataset. However, we used the same classifier's configurations as the two models above.

4.4.3 Results

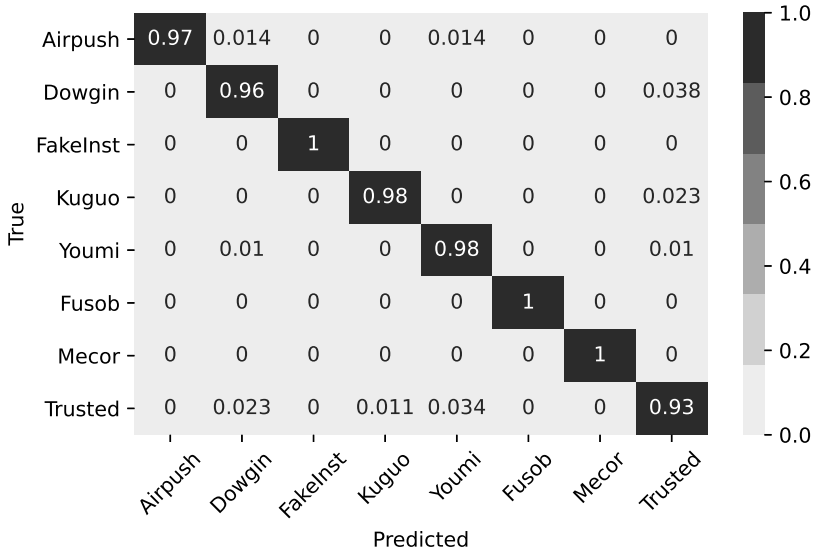


Figure 4.14: Row-normalized confusion matrix of CGMM computed on the test set.

Results are shown in Table 4.16. As we can see, the structural variability in the dataset is such that a structure-agnostic baseline cannot accurately classify instances by merely looking at the out-degree statistics of the graph. Instead, the three DBGNs are able to extract structural patterns that allow the subsequent classifier to achieve a very good accuracy on the test set (and Macro F1 score of approximately 97% for the best performing solution). Here, it seems that the configurations tried for iCGMM$_f$ and iCGMM$_{f_{auto}}$ led to a slight underfitting w.r.t. the other two models, which is probably due to the chosen ranges of hyper-parameters and hyper-priors for the peculiar out-degree distribution of the dataset.

These results support our knowledge that different malware families share detectable topological similarities and hence our hypothesis on the robustness of the approach to intra-procedural obfuscations. In fact, we were able to accurately detect such similarities without relying on non-adaptive procedures, domain expertise, and static analysis' vertex features that are susceptible to obfuscation techniques. In addition, the confusion matrix of Figure 4.14 shows how accuracy does not decrease for the most imbalanced classes, e.g., Fusob and Mecor. Rather, the classifier achieves perfect classification on those test samples. Finally, to empirically confirm that the proposed approach is robust to intra-procedural obfuscation methods, we also performed inference with CGMM on an obfuscated subset of test malwares (261 out of 391, due to intrinsic difficulties in the process, e.g., sometimes the obfuscated code did not compile) using the Code Reordering and Junk Code techniques in [250]. CGMM achieved a 99.6% accuracy.

4.5 Summary

We have introduced Deep Bayesian Graph Networks, a probabilistic alternative to Deep Neural Graph Networks to extract information from graphs of varying size and topology. Throughout the chapter, we have shown how it is possible to formalize the main building blocks of Deep Graph Networks using the well known tools of Bayesian inference. Our contributions rely on an incremental construction to break the mutual (and possibly cyclic) dependencies between latent variables associated with vertices, and the quality of the unsupervised embeddings is such that we managed to compete with state of the art supervised DNGNs on classification and regression tasks.

It may be worth mentioning that the probabilistic framework presented in this thesis is general enough to be extended in many ways, from the development of a "supervised CGMM" to the introduction of attention and more general aggregation mechanisms. In a sense, E-CGMM and ICGMM are examples of architectural and Bayesian non-parametric extensions of the basic CGMM, respectively, but it is not difficult to imagine variations of the graphical model that take advantage of variational bounds or automatic estimation of other hyper-parameters.

What is more, we still have to investigate the scenario in which DBGNs should excel at, namely pre-training of vertex/graph embeddings on a huge amount of **unlabelled** raw data. This was mainly due to the absence of large datasets which, however, are becoming more and more available these days [198].

Chapter 5

Graph Mixture Density Networks

La faccia sua era faccia d'uom giusto,
tanto benigna avea di fuor la pelle,
e d'un serpente tutto l'altro fusto;

Inferno - Canto XVII

In this final methodological chapter, we aim at building a **hybrid** model that gets the best of the two worlds presented so far, namely neural and Bayesian networks, in the context of deep learning for graphs. More specifically, our contribution is motivated by the need of modeling multimodal output distributions conditioned on topologically varying graphs: in this respect, we are extending the Mixture Density Network model to the processing of structured-data. What we call Graph Mixture Density Network (GMDN) [9] is basically the combination of a graph encoder, e.g., a DGN, and yet another conditional mixture model. This time, however, the overall architecture is a feedforward (but not constructive) DNGN. We shall present practical reasons as to why such a model is necessary, and we will formalize learning withing the framework of Generalized Expectation Maximization. A GMDN is particularly suited to express uncertainty about the possible continuous output values associated with an input graph. GMDN can tackle predictions of stochastic events, like the final outcome of an epidemic given the initial network, but it also can be applied to standard regression problems to better understand the data at hand. We complement the discussion with an alternative way to solve link prediction problems, using a measure of distance between two vertices' multimodal distributions. All in all, we will see how GMDN can be a useful tool to *i)* better analyze the data, as uncertainty usually arises from stochasticity, noise, or under-specification of the system of interest, and *ii)* train Deep Graph Networks which can provide further insights into their predictions and their trustworthiness.

5.1 Motivations

In Section 2.1.3, we discussed how the classical assumptions we make in regression problems do not hold anymore when the output distribution is multimodal. The Mixture Density Network [18] was designed to produce multimodal target distributions, but the input data has to be of vectorial nature.

In terms of applications, MDNs have been recently applied to epidemic simulation prediction [261]. The goal is to predict the multimodal distribution of the total number of infected cases under a compartmental model such as the **stochastic** Susceptible-Infectious-Recovered (SIR) model [262]. With SIR, each individual in the network can be in one of the three states (S,I or R), and there are very simple update rules to transition from one state to another, depending on the connectivity of the network and two parameters: i) infectivity β and ii) recovery γ. In the paper [261], the authors show that, given samples of SIR simulations with different infectivity and recovery parameters, the MDN could approximate the output distribution using a mixture of binomials. This result is a remarkable step in approximating way more complex compartmental models in a fraction of the time originally required, similarly to what has been done, for example, in material sciences [263] and molecular biosciences [10]. However, the work of [261] makes the strong assumption that the infected network is a complete graph. In fact, as stated in [264], arbitrary social interactions in the network play a fundamental role in the spreading of a disease, so predictive models should be able to take the topology into account [265].

Throughout this thesis, we had the chance to see that many real-world problems are best solved with relational data, where the structure substantially impacts the possible outcomes. For these reasons, we shall propose a hybrid approach to handle multimodal target distributions conditioned on graphs, namely the Graph Mixture Density Network (GMDN). This model can output distributions for either the whole structure or its individual entities. We shall use the likelihood as a metric for this kind of conditional density estimation tasks [266], since it tells us how well the model is fitting the empirical data distribution. Overall, GMDN extends the capabilities of all DNGNs whose output is restricted to unimodal distributions.

5.2 Model Definition [9]

We aim to learn the conditional distribution $P(y_g|g)$, with y_g being the continuous target label(s) associated with an input graph g in the dataset $\mathcal{D}$[1]. We assume the target distribution to be multimodal, and as such it cannot be well modeled by current DGNs.

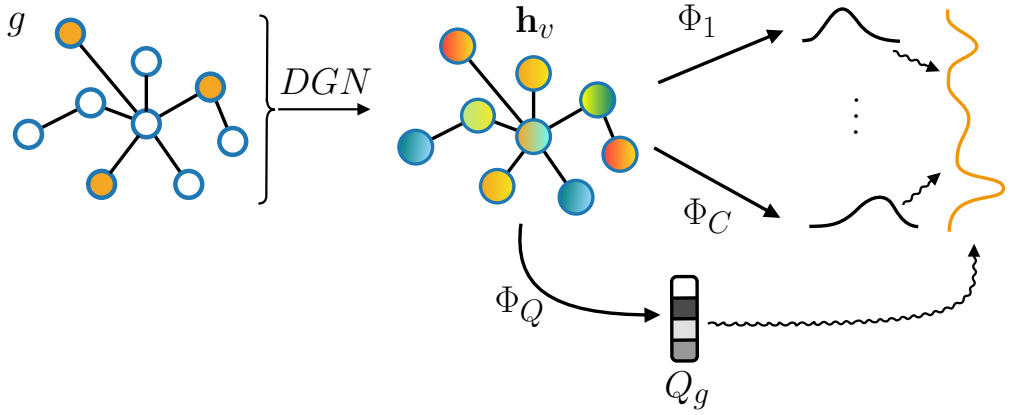


FIGURE 5.1: From a high-level perspective, we first obtain a state $\boldsymbol{h}_v$ for each vertex by applying an isomorphic transduction with a DGN encoder. Then, for each graph g, a subsequent transformation Φ_Q generates the mixing probability vector $Q_g \in [0,1]^C$ that combines the C different distributions produced by the sub-networks $\Phi_1, \ldots, \Phi_C$.

As sketched in Figure 5.1, we seek a DGN that performs an isomorphic transduction of the graph to obtain vertex representations $\boldsymbol{h}_{\mathcal{V}_g}$ as well as a set of mixing weights $Q_g \in [0,1]^C$ that sum to 1, where C is the number of unimodal output distributions we want to mix. Given $\boldsymbol{h}_{\mathcal{V}_g}$, we then apply C different sub-networks $\Phi_1, \ldots, \Phi_C$ that produce the parameters $\theta_1, \ldots, \theta_C$ of C output distributions, respectively.

In principle, we could mix distributions from different families, but this poses several issues, such as finding a rationale for their choice and choosing how many of them to use for each family. In light of these considerations, we stick to a single family for simplicity of exposition. Finally, combining the C unimodal output distributions with the mixing weights Q_g produces a multimodal output distribution.

More formally, we learn the conditional distribution $P(y_g|g)$ using the Bayesian network of Figure 5.2. We solve the CDE problem via maximum likelihood estimation, which reflects the probability that an output y is generated from a graph g. Given an hypotheses space $\mathcal{H}$, we therefore seek the following hypothesis:

$$h_{MLE} = \arg\max_{h \in \mathcal{H}} P(\mathcal{D}|h) = arg \max_{h \in \mathcal{H}} \prod_{g \in \mathcal{D}} \sum_{i=1}^{C} P(y_g|Q_g = i, g) P(Q_g = i|g),$$

[1]Note that the process to output vertex-specific distributions is almost identical, with the exception that global aggregation is not performed.

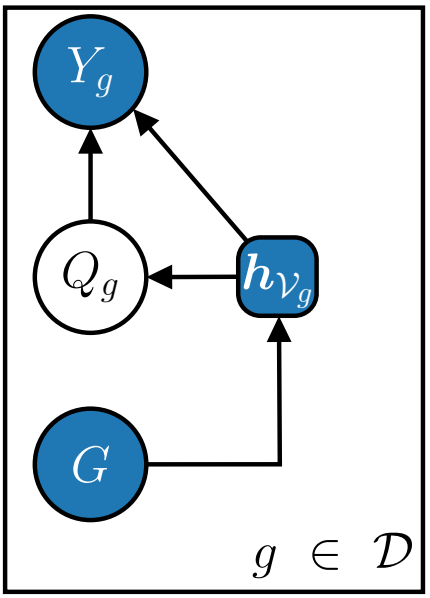


FIGURE 5.2: GMDN's Bayesian network is almost identical to that of a Mixture Density Network (see Figure 2.10), with the exception that the input observable has distribution with support over graphs rather than flat data.

where we introduced the latent variable Q_g via marginalization. We model the distributions in the equation by means of DGNs, given that a graph g may have a variable number of vertices and edges. In this respect, we choose the convolution of the Graph Isomorphism Network (GIN) [109] in our experiments. Also, the final vertex representation $\boldsymbol{h}_v$ is given by the concatenation of all L intermediate states, where L is the chosen number of layers.

Since we care about producing a single graph-related distribution, representations $\boldsymbol{h}_{\mathcal{V}_g}$ have to be globally aggregated with a function Ψ_g

$$\boldsymbol{h}_g = r_g(\boldsymbol{h}_{\mathcal{V}_g}) = \Psi_g\Big(\{f_r(\boldsymbol{h}_v) \mid v \in \mathcal{V}_g\}\Big),$$

where f_r could be a linear model or an MLP. Likewise, the mixing weights can be computed using a function r_g^Q as follows:

$$P(Q_g|g) = \sigma(r_g^Q(\boldsymbol{h}_{\mathcal{V}_g})),$$

where σ is the softmax over the components of the aggregated vector.

To learn the emission $P(y_g|Q_g^i, g),\ i = 1, \ldots, C$, we have to implement a sub-network Φ_i that outputs the parameters of the chosen distribution. For instance, if the distribution was a multivariate Gaussian, we would have

$$\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i} = \Phi_i(\boldsymbol{h}_g) = f_i(r_g^i(\boldsymbol{h}_{\mathcal{V}_g})),$$

with f_i being defined as f_r above. Note that vertex-prediction tasks do not need a global aggregation phase, so the mixing weights and emission transformations would be individually applied to $\boldsymbol{h}_v \; \forall v \in \mathcal{V}_g$.

It is important to remark that we share $\boldsymbol{h}_{\mathcal{V}_g}$ between all sub-networks; this is different from the so-called Mixture of Experts approach [267, 268], in which a different set of vertex representations would be created for each sub-network. This form of weight sharing reduces the number of parameters and pushes the model to extract all the relevant structural information into a single representation for each vertex. Last but not least, using multiple DGN encoders can easily become computationally intractable for large datasets.

5.3 Training

We train the GMDN model using the Expectation-Maximization (EM) framework [15] for MLE estimation. We continue choosing EM for the local convergence guaranteees that it offers with respect to other solutions, and since its effectiveness has already been proved on the probabilistic graph models introduced so far. By introducing the usual indicator variable $z_i^g \in \boldsymbol{Z}$, which is one when Q_g has latent state i, we can compute the lower bound of the log-likelihood as in standard mixture models [268, 269]:

$$\mathbb{E}_{\boldsymbol{Z}|\mathcal{D}}[\log \mathcal{L}_c(h|\mathcal{D})] = \sum_{g \in \mathcal{D}} \sum_{i=1}^{C} E[z_i^g|\mathcal{D}] \log \Big(P(y_g|Q_g^i, g) P(Q_g|g) \Big) \tag{5.1}$$

where $\log \mathcal{L}_c(h|\mathcal{D})$ is the complete log likelihood.

The E-step of the EM algorithm can be performed analytically by computing the posterior probability of the indicator variables:

$$E[z_i^g|\mathcal{D}] = P(z_i^g = 1|g) = \frac{1}{Z_{norm}} P(y_g|Q_g^i, g) P(Q_g|g)$$

where Z_{norm} is a normalization term obtained via marginalization. On the other hand, we do not have closed-form solutions for the M-step because of the non-linear transformations Φ used. Hence, we perform a gradient ascent step to maximize Equation 5.1. This is an instance of the GEM algorithm (Section 2.1.1.5), which still guarantees convergence to a local minimum if each optimization step improves Equation 5.1. Finally, we introduce an optional Dirichlet regularizer π with hyper-parameter $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_C)$ on the distribution $P(Q_g|g)$. The prior distribution serves to prevent the posterior probability mass of the from collapsing onto a single state; this is a well-known problem that has been addressed in the literature through specific constraints [270] or entropic

regularization terms [271]. Eventually, the objective to be maximized becomes

$$\underbrace{\mathbb{E}_{\boldsymbol{Z}|\mathcal{D}}[\log \mathcal{L}_c(h|\mathcal{D})]}_{\text{original objective}} + \underbrace{\sum_{g \in \mathcal{D}} \log \pi(Q_g|\boldsymbol{\alpha})}_{\text{Dirichlet regularizer}}, \tag{5.2}$$

where we note that $\boldsymbol{\alpha} = \mathbf{1}^C$ corresponds to a uniform prior, i.e., no regularization. Maximizing Equation 5.2 still preserves the convergence guarantees of GEM if the original objective increases at each step.

5.4 Encoding the structure via distribution distances

In the DGN literature, a common regularization technique encourages adjacent vertex representations to be similar in the Euclidean space and dissimilar otherwise [70]. This can be achieved by computing the dot product of pairs of vertex representations followed by sigmoidal activation (to obtain a "probability" of being adjacent). Ideally, this regularization term should help the model focus on structural patterns rather than overfitting vertex features.

This way, however, the **space** of vertex representations is *explicitly* constrained, and we argue that this may limit the amount of information that can be encoded into $\boldsymbol{h}_{\mathcal{V}_g}$ about the main classification/regression task. For this reason, we propose the first insights into a GMDN-based technique that *implicitly* embeds structural information into vertex representations. The idea to use GMDN to produce separate vertex distributions other than those required for the main task. Then, we can encourage the distance between pairs of such vertex distributions to be close if the vertices are indeed adjacent. For the Data Processing Inequality [272], it follows that vertex representations obtained in this way will encode structural information, but there will be **no explicit** constraint on the space they live in.

While the application of this strategy to regularization seems promising, we should first investigate whether it is actually possible to learn appropriate distribution distances that encode the adjacency information. This thesis will take a step in this direction, rather than focusing on regularization benefits, by analyzing the ability of different distance functions to solve link prediction tasks. We graphically sketch the idea behind this experiment in Figure 5.3.

In this context, mixtures of Gaussians prove useful, as there are many possible choices for the distance function. An example is the closed-form L2 distance between two Gaussian mixture distributions P and Q described in [273]. We define the L2 distance as

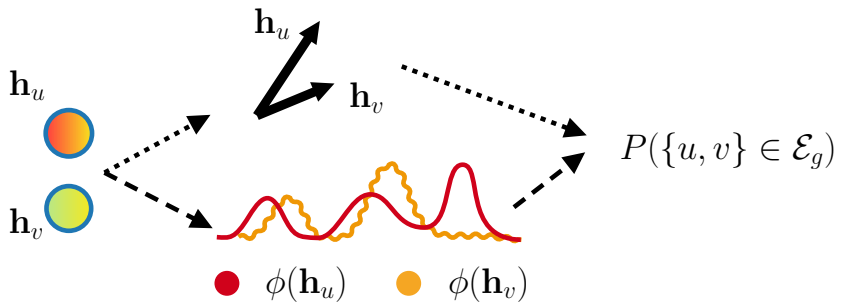


FIGURE 5.3: In explicit regularization (top), adjacent vertex representations must be aligned in the Euclidean space. Instead, we propose to implicitly encode adjacency information (bottom) into the representations $\boldsymbol{h}_{\mathcal{V}_g}$ by minimizing the distance between adjacent vertex **distributions**.

$$L_2^2(P, Q) = \int_{\mathbb{R}} (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\boldsymbol{x}. \tag{5.3}$$

This function sums the point-wise squared distances between the *pdfs* of the two distributions, and it is not difficult to implement in matrix form for univariate Gaussian mixtures.

Lastly, mapping each vertex representation into a one-dimensional distribution could also be used as a structure-aware dimensionality reduction technique, in contrast to task-agnostic alternatives commonly used in the literature [274, 275].

We now describe how we implement the different distances between pairs of Gaussian mixture distributions.

5.4.1 L2 Distance

This is the usual squared Euclidean distance

$$L_2^2(P, Q) = \int_{\mathbb{R}} (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\boldsymbol{x}$$

which, for mixture of distributions, can be written as[2]

$$
\begin{aligned}
L_2^2(P,Q) &= \int_{\mathbb{R}} \big(\sum_{i}^{C} \alpha_i p_i(\boldsymbol{x}) - \sum_{j}^{C} \beta_j q_j(\boldsymbol{x})\big)^2 d\boldsymbol{x} \\
&= \sum_{i,j} \alpha_i \alpha_j \int_{\mathbb{R}} p_i(\boldsymbol{x}) p_j(\boldsymbol{x}) d\boldsymbol{x} + \beta_i \beta_j \int_{\mathbb{R}} q_i(\boldsymbol{x}) q_j(\boldsymbol{x}) d\boldsymbol{x} \\
&- 2 \sum_{i,j} \alpha_i \beta_j \int_{\mathbb{R}} p_i(\boldsymbol{x}) q_j(\boldsymbol{x}) d\boldsymbol{x},
\end{aligned}
$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the mixture weight vectors of the two distributions. In general, we can compute the integral of the product of two Gaussians as

$$
\int_{\mathbb{R}} \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_1}, \boldsymbol{\Sigma_1}) \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_2}, \boldsymbol{\Sigma_2}) d\boldsymbol{x} = \mathcal{N}(\boldsymbol{\mu_1} \mid \boldsymbol{\mu_2}, (\boldsymbol{\Sigma_1} + \boldsymbol{\Sigma_2}))
$$

where we have used a known property of the product of Gaussians (see Section 8.1.8 of [277]) and the fact that the integral of a density function sums to 1. Therefore, if we define

$$
\begin{aligned}
A_{i,j} &= \int_{\mathbb{R}} p_i(\boldsymbol{x}) p_j(\boldsymbol{x}) d\boldsymbol{x} = \mathcal{N}(\mu_i^P \mid \mu_j^P, (\sigma_i^P + \sigma_j^P)) \\
B_{i,j} &= \int_{\mathbb{R}} q_i(\boldsymbol{x}) q_j(\boldsymbol{x}) d\boldsymbol{x} = \mathcal{N}(\mu_i^Q \mid \mu_j^Q, (\sigma_i^Q + \sigma_j^Q)) \\
C_{i,j} &= \int_{\mathbb{R}} p_i(\boldsymbol{x}) q_j(\boldsymbol{x}) d\boldsymbol{x} = \mathcal{N}(\mu_i^P \mid \mu_j^Q, (\sigma_i^P + \sigma_j^Q))
\end{aligned}
$$

then the Euclidean distance can be computed as

$$
L_2^2(P,Q) = \sum_{i,j} \alpha_i \alpha_j A_{i,j} + \beta_i \beta_j B_{i,j} - 2 \sum_{i,j} \alpha_i \beta_j C_{i,j}.
$$

5.4.2 Jeffrey's Distance

The Jeffrey's distance can be thought as the symmetric version of the KL Divergence or, equivalently, as double the α-JS divergence with $\alpha = 1$. We use the α-JS divergence implementation, though the difference w.r.t. Jeffrey's lies only in a constant value. We consider a weighted sum of C distances (univariate case) that relies on the corresponding

[2] We follow the straightforward derivation of [276].

mixing weights:

$$J_w(P,Q) = \frac{1}{2}\sum_{i=1}^{C} w_i^P KL(P_i \,||\, Q_i) + w_i^Q KL(Q_i \,||\, P_i)$$
$$\text{where } KL(P \,||\, Q) = \log\frac{\sigma_Q}{\sigma_P} + \frac{\sigma_P^Q + (\mu_P - \mu_Q)^2}{2\sigma_Q^2} - \frac{1}{2}.$$

5.4.3 Bhattacharyya's Distance

Similarly, we define the weighted sum of Bhattacharyya's distances (univariate case) as

$$B_w(P,Q) = \sum_{i=1}^{C}\int_{\mathbb{R}}\sqrt{(w_i^P p_i(x) w_i^Q q_i(x))}dx$$
$$= \sum_{i=1}^{C}\sqrt{w_i^P w_i^Q}\int_{\mathbb{R}}\sqrt{p_i(x)q_i(x)}dx$$
$$= \sum_{i=1}^{C}\sqrt{w_i^P w_i^Q}\Big(\frac{1}{8}\frac{2(\mu_P - \mu_Q)^2}{\sigma_1 + \sigma_2} + \frac{1}{2}\log(\frac{\sigma_P + \sigma_Q}{2\sqrt{\sigma_P\sigma_Q}})\Big).$$

5.5 Experiments

We now describe the datasets, experiments, evaluation process and hyper-parameters used to empirically study GMDN. Our goal is to show how GMDN can fit multimodal distributions conditioned on a graph better than using MDNs or DGNs individually. To do so, we publicly release large datasets of stochastic SIR simulations whose results depend on the underlying network, rather than assuming uniformly distributed connections as in [261]. The datasets have been generated using random graphs from the Barabasi-Albert (BA) and Erdos-Renyi (ER) families. While ER graphs do not preserve social networks' properties, here we are just interested in the emergence of multimodal outcome distributions rather than biological plausibility. That said, future investigation will cover more realistic cases, for instance using the Block Two-Level Erdos–Renyi model [278]. We also apply the model on two molecular graph regression benchmarks, to analyze the performances of GMDN on real-world data.

Two additional experiments complement the exposition: first, we analyze whether training on a particular family of graphs exhibits transfer properties; if that is the case, then the model has learned how to make informed predictions about different (let alone completely new) structures. Secondly, we study whether we can use vertex-specific multimodal distribution to perform link prediction. We recall that the main goal is to decouple

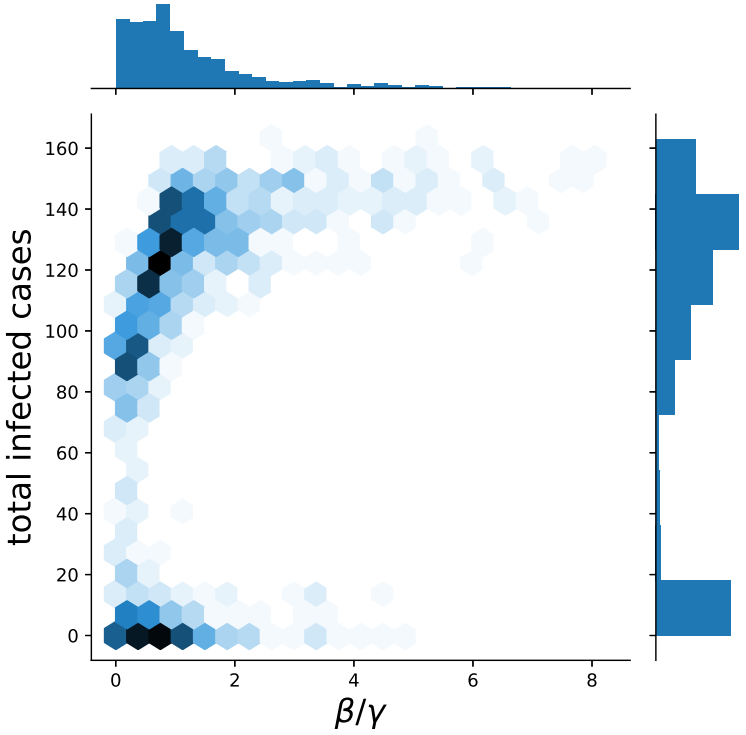


FIGURE 5.4: Given a single network and specific choices for $R0 = \beta/\gamma$, the repeated simulation of the stochastic SIR model is known to produce different outcomes. Here we plot the outcome distributions of 1000 SIR simulations on an Erdos-Renyi network of size 200. We follow [261] and sample β and γ uniformly, rather than their ratio, because higher ratios correspond to less interesting behaviors, i.e., the distribution becomes unimodal. Depending on the input structure, the distribution of the total infected cases may be multimodal or not, and GMDN should recognize this phenomenon. In our simulations, larger networks exhibited less multimodality; hence, without loss of generality, we focus on larger datasets of smaller graphs.

the role of vertex representations from the objects used to compute the pair-wise link prediction scores.[3]

5.5.1 Datasets

We simulated the well-known stochastic SIR epidemiological model on Barabasi-Albert graphs of size 100 (BA-100), generating 100 random graphs for different connectivity values (2, 5, 10 and 20). Borrowing ideas from [261], for each configuration, we run 100 simulations for each different initial infection probability (1%, 5%, 10%) sampling the infectivity parameter β from $[0, 1]$ and the recovery parameter γ from $[0.1, 1]$. We also carry out simulations for Erdos-Renyi graphs (ER-100), this time with connectivity parameters 0.01, 0.05, 0.1, and 0.2. The resulting total number of simulations (i.e., samples) in each dataset is 120.000, and the goal is to predict the distribution of the

[3] https://github.com/diningphil/graph-mixture-density-networks.

total infected cases at the end of a simulation. Vertex features consist of β, γ, their ratio $R0 = \beta/\gamma$, a constant value 1, and a binary value that indicates whether that vertex is infected or not at the beginning of the simulation. Moreover, to test the transfer learning capabilities of GMDN on graphs with different structural properties (according to the chosen random graph model), we constructed six additional simulation datasets where graphs have different sizes, i.e., 50, 200 and 500. An instance of simulation results is summarized in Figure 5.4; we observe that the outcome distribution of repeated simulations on a single graph leads to a multimodal distribution, in accord with [264]. Therefore, in principle, being able to accurately and efficiently predict the outcome distribution of a (possibly complex) epidemiological model can significantly impact the preparations for an incumbent sanitary emergency.

When dealing with real-world graph regression tasks, especially in the chemical domain, we usually do not expect such a conspicuous emergence of multimodality in the output distribution. Indeed, the properties of each molecule are assumed to be regulated by natural laws, but the information we possess about the input representation may be incomplete and/or noisy. Similarly, the way the model processes the input has an impact on the overall uncertainty; for instance, disregarding bond information makes graphs appear isomorphic to the model while they are indeed not so. As such, knowing the confidence of a trained regressor for a specific outcome becomes invaluable to better understand the data, the model behavior, and, ultimately, to determine the trust we place in each prediction. Therefore, we will evaluate our model on the large chemical benchmarks alchemy_full [279] and ZINC_full [280, 281] made of 202579 and 249456 molecules, respectively. The task of both datasets is the prediction of continuous chemical properties (12 for the former and 1 for the latter) associated with each molecule representation (9 and 28 vertex features, respectively). As in [279], the GIN convolution used only exploits the existence of a bond between atoms. In the considered datasets, this gives rise to isomorphic representations of different molecules when bond types or 3D coordinates are not considered (or simply ignored by the trained model). The same phenomena, in different contexts and forms, can occur whenever the original data or its choice of representation lack part of the information to solve a task.

Finally, to start studying the feasibility of GMDN as a link predictor, we will make use of the same Cora and Pubmed datasets introduced in the context of E-CGMM.

5.5.2 Evaluation Setup.

We assess the performance of different models using a holdout strategy for all datasets (80%/10%/10% split). Given the size of the datasets, we believe that a simple holdout

is sufficient to assess the performances of the different models considered. To make the evaluation even more robust for the epidemic datasets, different simulations about the same graph cannot appear in both training and test splits. The metric of interest is the log-likelihood of the data ($\log \mathcal{L}$), which captures how well we can fit the target distribution and the model's uncertainty with respect to a particular output value. We also report the Mean Average Error (MAE) on the real-world benchmarks for completeness. However, the MAE does not reflect the model's uncertainty about the output, as we will show.

Instead, we split the links of Cora and Pubmed according to a bootstrap sampling technique. We created ten different 85%/5%/10% link splits with an equal number of true (class 1) and false (class 0) edges. We recall that this setup is more robust than using a single split [70, 177], but it is ten times more expensive because we must perform a model selection for each split. We treat sampled false edges as directed for a better exploration of the space of unconnected pairs of vertices. We use an L1 loss with target distance 0 for the positive class and 2 for the negative class. When it comes to computing classification scores, we convert each distance d into a probability using the continuous function $1/(1+d)$ (though hard thresholds are also possible). Following the literature, we evaluate the classification performance using the area under the curve (AUC) and the average precision (AP).

We perform model selection via grid search for all the models presented. For each of them, we select the best configuration on the validation set using early stopping with patience [196]. As regards holdout, to avoid an unlucky random initialization of the chosen configuration, we average the model's performance on the unseen test set over ten final training runs. Instead, 3 final training runs are used for the link prediction experiments. Similarly to the model selection phase, in all these final training runs we use early stopping on a validation set extracted from the training set (10% of the training data).

Baselines and hyper-parameters On the synthetic and chemical tasks, we compare GMDN against four different baselines. First, RAND predicts the uniform probability over the finite set of possible outcomes, thus providing the threshold log-likelihood score above which predictions are useful. Instead, HIST computes the normalized frequency histogram of the target values given the training data, which is then converted into a discrete probability. While on epidemic simulations we can use the graph's size as the number of histogram bins to use, on the chemical benchmarks this number must be treated as a hyper-parameter and manually cross-validated against the validation set. HIST is used to test whether multimodality is useful when a model does not take

the structure into account. Finally, we have MDN and DGN, which are, in a sense, ablated versions of GMDN. Indeed, MDN ignores the input structure, whereas DGN cannot model multimodality. Neural models are trained to output unimodal (DGN) or multimodal (MDN, GMDN) binomial distributions for the epidemic simulation datasets and isotropic Gaussians for the chemical ones. The sub-networks Φ_i are linear models, and the graph convolutional layer is adapted from [109].

For link prediction, we test a Graph Auto-Encoder (GAE) [81] as a strong baseline that computes the dot product between pairs of vertex representations. Then, we test different versions of GMDN according to the distributional distance used: L2 distance (GMDN-L2), weighted Jeffrey distance (GMDN-J), and weighted Bhattacharyya distance (GMDN-B). We now list the hyper-parameters tried for each model:

- MDN: $C \in \{2,3,5\}$, hidden units per convolution $\in \{64\}$, neighborhood aggregation $\in \{\text{sum}\}$, global aggregation $\in \{\text{sum, mean}\}$, $\boldsymbol{\alpha} \in \{\mathbf{1}^C, \mathbf{1.05}^C\}$, epochs $\in \{2500\}$, $\Phi_i \in \{\text{Linear model}\}$, Adam Optimizer with learning rate $\in \{0.0001\}$, full batch, patience $\in \{30\}$.
- GMDN: $C \in \{3,5\}$, number of layers $\in \{2,5,7\}$, hidden units per convolution $\in \{64\}$, neighborhood aggregation $\in \{\text{sum}\}$, global aggregation $\in \{\text{sum, mean}\}$, $\boldsymbol{\alpha} \in \{\mathbf{1}^C, \mathbf{1.05}^C\}$, epochs $\in \{2500\}$, $\Phi_i \in \{\text{Linear model}\}$, Adam Optimizer with learning rate $\in \{0.0001\}$, full batch, patience $\in \{30\}$.
- DGN: same as GMDN but $C \in \{1\}$ (that is, it outputs a unimodal distribution).
- GAE: number of layers $\in \{1,2,3\}$, hidden units per convolution $\in \{32, 64, 128, 256, 512\}$, neighborhood aggregation $\in \{\text{sum}\}$, epochs $\in \{5000\}$, Adam Optimizer with learning rate $\in \{0.01, 0.001\}$, full batch, patience $\in \{1000\}$.
- GMDN-L2/J/B: $C \in \{2,5,10,20,30,50\}$, number of layers $\in \{1\}$[4], hidden units per convolution $\in \{128,256,512,1024\}$, neighborhood aggregation $\in \{\text{sum, mean}\}$,$\boldsymbol{\alpha} \in \{\mathbf{1}^C\}$, epochs $\in \{2500\}$, $\Phi_i \in \{\text{Linear model }\}$, Adam Optimizer with learning rate $\in \{0.01, 0.001\}$, full batch, patience $\in \{200$ (GMDN-J), $500\}$.

Note that we kept the maximum number of epochs intentionally high as we use early stopping to halt training. Also, the results of the experiments hold regardless of the DGN variant used, given the fact that DGNs output a single value rather than a complex distribution. In other words, we are comparing *families* of models rather than specific architectures.

[4] After evaluating GAE, we observed that 1 layer was sufficient to achieve the best results.

5.6 Results

We discuss our findings starting from the main empirical study on epidemic simulations, which includes CDE results and transferability of the learned knowledge. Then, we report results obtained on the real-world chemical tasks, highlighting the importance of capturing a model's uncertainty about the output predictions. Finally, we show the first insights into using distributional distances to tackle link prediction tasks.

5.6.1 Epidemic Simulation Results

We begin by analyzing the results obtained on BA-100 and ER-100 in Table 5.1. We notice that GMDN has better test log-likelihoods than the other baselines, with larger performance gains on ER-100. Being GMDN the only model that considers both structure and multimodality, such an improvement was to be expected. However, it is particularly interesting that HIST has a better log-likelihood than MDN on both tasks. By combining this fact with the results of DGN, we come to two conclusions. First, the structural information seems to be the primary factor of performance improvement; this should not come as a surprise since the way an epidemic develops depends on how the network is organized (despite we are not aiming for biological plausibility). Secondly, none of the baselines can get close enough to GMDN on ER-100, indicating that this task is harder to solve by looking individually at structure or multimodality. In this sense, BA-100 might be considered an easier task than ER-100, and this is plausible because emergence of multimodality on the former task seems slightly less pronounced in the SIR simulations. For completeness, we also tested an intermediate baseline where DGN is trained with L1 loss followed by MDN on the graph embeddings. Results displayed a $\log \mathcal{L} \approx -16$ on both datasets, probably because the DGN creates similar graph embeddings for different distributions with the same mean, with consequent severe loss of information.

	BA-100	**ER-100**	**Structure**	**Multimodal**
RAND	-4.60	-4.60	✗	✗
HIST	-1.16	-2.32	✗	✓
MDN	-1.17(.05)	-2.54(.07)	✗	✓
DGN	-0.90(.35)	-1.96(.16)	✓	✗
GMDN	**-0.67**(.02)	**-1.56**(.04)	✓	✓

TABLE 5.1: Results on BA-100 e ER-100 (12.000 test samples each). A higher log-likelihood corresponds to better performances. GMDN improves the performance on both tasks, showing the advantages of that taking into account both multimodality and structure. Neural models' results are averaged over 10 runs, and standard deviation is reported in brackets.

Similarly to what has been done in [18] and [261], we analyze how the mixing weights and the distribution parameters vary on a particular GMDN instance. We use C=5 and track the behavior of each sub-network for 100 different ER-100 graphs. Figure 5.5 shows the trend of the mixing weights (left) and of the binomial parameters p (right) for different values of the ratio $R0 = \beta/\gamma$. We immediately see that many of the sub-networks are "shut down" as the ratio grows. In particular, sub-networks 3 and 4 are the ones that control GMDN's output distribution the most, though for high values of R0 only one sub-network suffices. These observations are concordant with the behavior of Figure 5.4: when the infectivity rate is much higher than the recovery rate, the target distribution becomes unimodal. The analysis of the binomial parameter for sub-network 4 provides another interesting insight. We notice that, depending on the input graph, the sub-network leads to two possible outcomes: the outbreak of the disease or a partial infection of the network. Note that this is a behavior that GMDN can model whereas the classical MDN cannot.

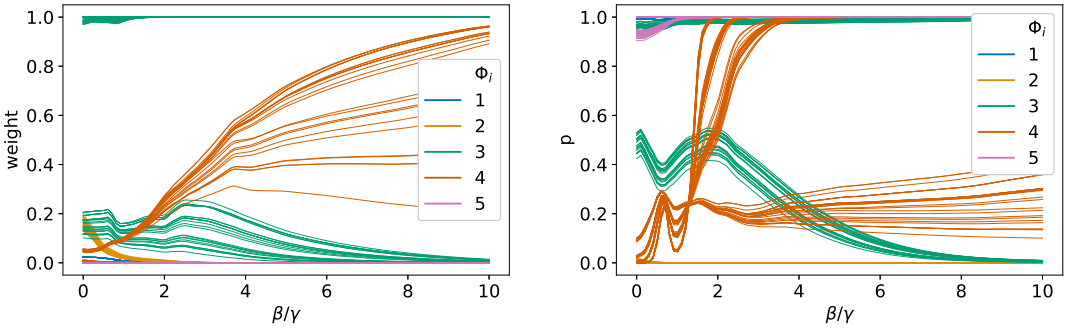


FIGURE 5.5: The trend of the mixing weights (left) and binomial coefficient (right) for each one of five sub-networks is shown on 100 ER-100 graphs. We vary the ratio between infection and recovery rate to inspect the behavior of the GMDN. Here, we see that sub-network 4 can greatly change the binomial output distribution in a way that depends on the input graph.

To provide further evidence about the benefits of the proposed model, Figure 5.6 shows the output distributions of MDN, DGN and GMDN for a given sample of the ER-100 dataset. We also plot the result of SIR simulations on that sample as a blue histogram (ground truth). Some observations can be made. First, the MDN places the output probability mass at both sides of the plot. This choice is understandable considering the lack of knowledge about the underlying structure (see also Table 5.1) and the fact that likely output values tend to be polarized at the extremes (see e.g., Figure 5.4). Secondly, the DGN can process the structure but cannot model more than one outcome. Therefore, and coherently with [18] for vectorial data, the DGN unique mode lies in between those of GMDN that account for the majority of GMDN probability mass. In contrast, GMDN produces a multimodal and structure-aware distribution that closely follows the ground truth.

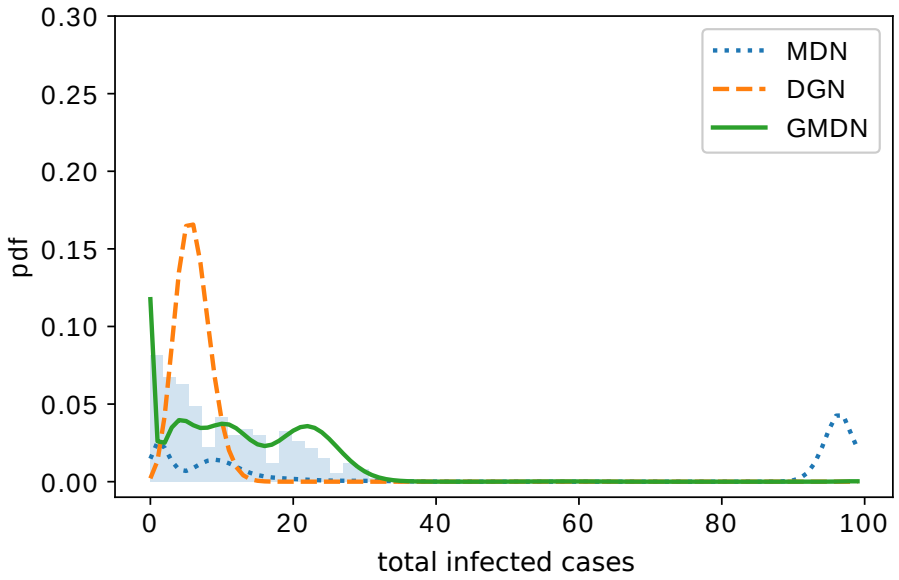


FIGURE 5.6: Output distributions of MDN, DGN, and GMDN on an ER graph of size 100. As we can see, the GMDN can provide a rich multimodal distribution conditioned on the structure close to that generated by SIR simulations (blue histogram).

5.6.2 Transfer Results

To tell whether GMDN can transfer knowledge to a random graph of different size and/or family (i.e., with different structural properties), we evaluate the trained models on the six additional datasets described in Section 5.5. Results are shown in Figure 5.7, where the RAND score acts as the reference baseline. The general trend is that the GMDN trained on ER-100 has better performances than its counterpart trained on BA-100; this is true for all ER datasets, BA-200 and BA-500. This observation suggests that training on ER-100, which we assumed to be a "harder" task than BA-100 as discussed above, allows the model to better learn the dynamics of SIR and transfer them to completely different graphs. Since the structural properties of the random graphs vary across the datasets, obtaining a transfer effect is therefore far from being a trivial task.

5.6.3 Chemical Benchmarks

We move to the results on the real-world chemical benchmarks, which are summarized in Table 5.2. We observe a log-likelihood trend similar to that in Table 5.1, with the notable difference that DGN performs much worse than MDN on alchemy_full. Following the discussion in Section 5.5, we evaluate how models deal with the uncertainty in the prediction by analyzing one of the output components of alchemy_full. Figure 5.8 shows such an example for the first component (dipole moment). The two modes of the GMDN suggest that, for some input graphs, it may not be clear which output value is more appropriate. This is confirmed by the vertical lines representing output values of

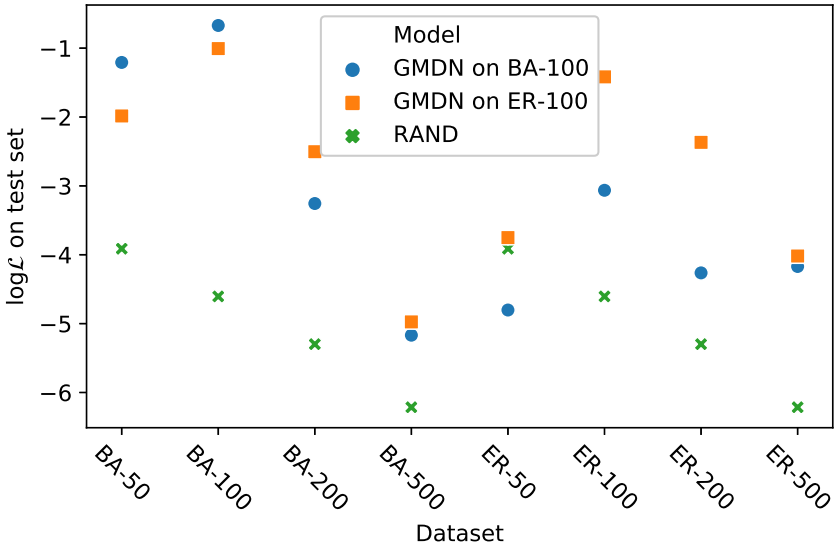


FIGURE 5.7: Transfer learning effect of the trained GMDNs are shown as blue dots and orange squares. Higher scores are better. GMDN trained on ER-100 exhibits better transfer on larger BA-datasets, which might be explained by the difficulty of the source task.

isomorphic graphs (as discussed in Section 5.5). Similarly to Figure 5.6, the DGN tries to cover all possible outcomes with a single Gaussian in between the GMDN modes. Although this choice may well minimize the MAE score over the dataset, the DGN fails to model the data we have.

	alchemy_full		**ZINC_full**	
	log $\mathcal{L}$	MAE	log $\mathcal{L}$	MAE
RAND	-27.12	-	-4.20	-
HIST	-21.91	-	-1.28	-
MDN	-1.36(.90)	0.62(.01)	-1.14(.01)	0.67(.00)
DGN	-7.19(1.3)	0.62(.01)	-0.90(.10)	0.49(.03)
GMDN	**-0.57**(1.4)	**0.61**(.02)	**-0.75**(.10)	**0.49**(.04)

TABLE 5.2: Results on the chemical tasks show how GMDN consistently reaches better log-likelihood values than the baselines. We also report the MAE as secondary metric for future reference, using the weighted mean of the sub-networks as the prediction (see [18] for alternatives). Clearly, the MAE does not reflect the amount of uncertainty in a model's prediction, whereas the log-likelihood is the natural metric for that matter. Results are averaged over 10 training runs with standard deviation in brackets.

5.6.4 Distributional Distances for Link Prediction

We conclude the chapter with an investigation into the ability of GMDN to implicitly transfer structural information into the vertex representations by computing the distance between vertex distributions. We report our structure-reconstruction results in Table 5.3.

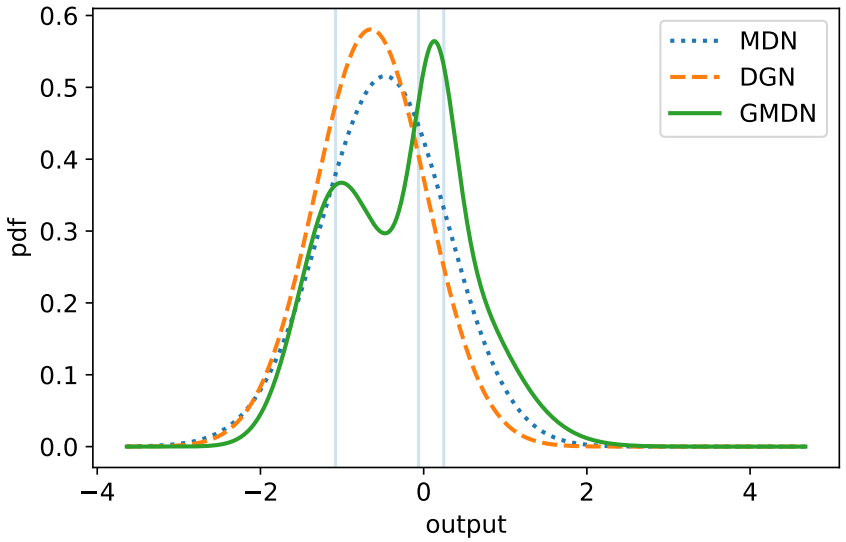


FIGURE 5.8: We illustrate the output distributions on the first component, i.e., dipole moment, of an alchemy_full graph. As noted in the text, DGN places high confidence in between the two modes of GMDN. On the contrary, GMDN is able to express uncertainty about the possible output values (vertical lines) associated with isomorphic graphs, which can be found if 3D attributes are not considered. The existence of the two modes suggests that 3D attributes are nonetheless ignored by the three models. See the discussion of Section 5.5 for a more in-depth explanation of the phenomenon.

When looking at the results, it is important to remark that the objective is *not* to perform better than GAE but rather to assess whether distribution distances can retain a good amount of structural information.

	Cora		**Pubmed**	
	AUC	AP	AUC	AP
GAE	91.9(0.6)	92.0(0.6)	96.9(0.2)	97.0(0.2)
GMDN-B	86.8(1.4)	87.0(1.4)	89.8(1.2)	86.2(1.7)
GMDN-L2	81.6(3.5)	82.9(3.4)	91.8(1.4)	90.2(1.3)
GMDN-J	87.8(1.2)	89.0(1.6)	94.9(0.4)	94.5(0.4)

TABLE 5.3: Results for the structure reconstruction tasks.

In general, all the distance functions perform properly, though the weighted Jeffrey distance is the one with the best results (even close to the ones of GAE). This distance is also more efficient than the L2, which elegantly takes into account Gaussian mixtures but has a quadratic cost in the number of sub-networks. Also, GMDN-L2 was the slowest converging model, possibly due to the complex dependencies between pairs of mixtures.

From these preliminary results, it seems clear that distributional distances can be used to approximately reconstruct most of the adjacency information without necessarily imposing explicit constraints on internal vertex representations. Therefore, future work will

further investigate the potential of this technique as a regularization strategy when solving common graph regression/classification tasks, as well as finding new distributional distances that provide even better link prediction accuracies.

5.7 Summary

With the Graph Mixture Density Networks, we have introduced a new family of models that combine the benefits of Deep Graph Networks and Mixture Density Networks. GMDN can solve challenging tasks where the input is a graph and the conditional output distribution is multimodal. In this respect, we have introduced a novel benchmark application for graph conditional density estimation founded on stochastic epidemiological simulations. The effectiveness of GMDM has also been demonstrated on real-world chemical regression tasks and as a promising tool to address link prediction.

In the future, we plan to further study the impact of GMDN on more biologically plausible synthetic datasets and find new application domains. We believe there are plenty of directions in which we can extend GMDN, for instance by using a recurrent encoder in order to model dynamically varying graphs [282, 283]. Moreover, we foresee that graph-based reinforcement learning [284, 285] may benefit from the added degree of uncertainty over continuous outputs that GMDN provides.

Overall, we hope that this general framework will play an important role in the approximation of structure-dependent phenomena that exhibit non-trivial conditional output distributions.

Chapter 6

Conclusions

E quindi uscimmo a riveder le stelle.

Inferno - Canto XXXIV

Over the last fifteen years, we have witnessed a race to digitalization throughout all public and private sectors. As a by-product of this ongoing modernization, the amount of data produced and stored has swiftly increased, so much so that it is now considered a new asset for businesses. Despite the ethical implications of this being debated by experts as well as by the general public, we could argue that the availability of data samples has fostered the research and application of machine learning techniques that provide the community with valuable services. Real-time translation, breast cancer detection, and hate-speech recognition are just a few instances of what can be seen as a direct or indirect attempt to make the world a safer and more inclusive place.

Oftentimes, however, these services make use of relatively simple realizations of structured data in the form of vectors or sequences, with the consequent inability to process more complex relationships that may exist; this is the case of molecular predictions and generation, discovery of unknown protein-protein interactions, and detection of malicious activities in a social network or software, where the data is naturally encoded as a graph. Indeed, the classical algorithmic extraction of actionable information from such structures is rarely an easy task, due to the fact that any topological variability of the input graphs must be accounted for. This dissertation discussed what is Deep Learning for Graphs and how it can help in the automatic discovery of a mapping between a graph-structured input and a flat output. In Section 3.2, we tried to give a broad but systematic perspective of the basic principles that characterize this field [1]. The top-down approach we followed was meant to be accessible even to the noninitiated, and it allowed us to see our and others' contributions through the same lens. Moreover, our

review paid attention to the foundational approaches that shaped the field both to give a historical perspective and to prevent a wave of re-discovery of ideas.

Throughout the entire manuscript, we were also conscious of the fact that experimental reproducibility on some graph benchmarks had been slightly overlooked, possibly because of the tremendous stream of works produced in the last years. For this reason, before commencing any methodological chapter, we did our best to ensure that a fair, robust, and reproducible comparison on graph classification benchmarks was available to compare our models against state of the art DGNs (Section 3.3 [5]). In the process, we discovered the importance of setting up proper structure-agnostic baselines, and we showed how an incorrect evaluation setup can result in over-optimistic estimates of the models' generalization performances.

To show that Deep Learning for Graphs is actually useful in practice, we integrated a real-world example from the field of molecular biosciences [10] in Section 3.4. Specifically, we showed that Deep Graph Networks are able to fairly well approximate a very complex process that takes a given protein and returns an information-theoretic quantity of interest. The real advantage of doing so is that said approximation can be done in a minuscule fraction of the time required by the original method, thus allowing a quasi-exhaustive study of the protein under consideration. That said, it is still unclear how to transfer the learned knowledge to a different family of proteins, which could radically change the way we approach the problem.

Moving to our main contribution, the design of Deep Bayesian Graph Networks has been guided by the principles of local and iterative processing of information as well as by the classical building blocks of Bayesian learning. The goal was to show that it is possible to implement an effective, deep, and fully probabilistic learning approach for one of the most unconstrained data structures. Inspired by pioneering methods, we have proposed a probabilistic framework for learning on graphs, founded on an incremental construction that facilitates information propagation through deeper architectures with respect to most neural counterparts. The Contextual Graph Markov Model of Section 4.1 can be seen as the simplest realization of such a framework, in which the neighborhood aggregation is defined in probabilistic terms and can deal with discrete edge labels [6, 7]. Surprisingly enough, the unsupervised nature of the model did not come at the price of significantly worse performances in the supervised tasks considered; rather, the model reached a very competitive accuracy on a number of classification benchmarks.

Later on, in Section 4.2, we continued to find ways to make our framework more general. One of the issues was that the presence of non-discrete edge features could not be modeled by CGMM. The solution lied in acting at the architectural level rather than

on the definition of the neighborhood aggregation mechanism [8]; the addition of a second Bayesian network, responsible for the generation of edge features, made possible to adaptively discretize (in an unsupervised fashion) edge information, so that we could apply again the original CGMM model. An unexpected outcome of this extension was that classification performances improved even on those benchmarks where edge features were missing. We attributed this phenomenon to the dynamic neighbor aggregation that arises from the discretization of edges at each layer of the constructive architecture.

There are still many interesting open problems that regard the generality of the framework. For instance, the neighborhood aggregation scheme is based on the mean operator, but the sum is a theoretically more expressive operator over multi-sets (under appropriate conditions). Hence, the investigation of a fully-probabilistic formulation for the sum aggregation would certainly enhance the representational capabilities of the framework, which could then capture richer patterns in the graph structure. Nonetheless, in the interest of a broad exploration of the research space, we instead focused on the automatic selection of some hyper-parameters of the framework, in particular the number of latent states of the vertices' categorical variables. Section 4.3 bridges ideas from Deep Learning for Graphs and Bayesian Nonparametric methods to create the first DBGN whose complexity grows with the data. What we named Infinite Contextual Graph Markov Model is a deep architecture where each layer is a possibly infinite conditional mixture model, implemented as a Gibbs sampling-based Hierarchical Dirichlet Process. Empirically, the model performed similarly to CGMM, but it chose a number of latent states much smaller than the best configuration of a model selection procedure, thus saving disk space where to store the embeddings. The drawbacks of the approach are the slowness of the sampler and the inability to consider edge features, both of which will be subject of future works, for instance via variational derivations of the HDP.

To apply the DBGNs developed in this thesis to a real-world application, we considered the problem of robust malware classification in Section 4.4 [11]. In particular, by considering graph representations of programs that are unaware of the intra-procedural code changes, we managed to successfully classify a substantial number of malware families by just looking at the topology of the input graph. This, in turn, allowed the proposed procedure to be robust to a particular subset of code obfuscation techniques, and the learned unsupervised embeddings were rich enough to distinguish the peculiar structural variations in the data distribution.

The common thread of the entire thesis has been the cross-fertilization of ideas belonging to different research fields. In keeping with this spirit, the last methodological contribution of Chapter 5 has been a hybrid framework that combines a generic encoding transduction, realized by a DGN, and the probabilistic capabilities of Bayesian networks

[9]. This was necessary to model multimodal output distributions conditioned on topologically varying input graphs. The resulting Graph Mixture Density Network takes the best of both neural and probabilistic worlds to solve regression problems like the prediction of an epidemic's outcome on synthetic graphs, which is inherently stochastic. This model can also be used to add a degree of trustworthiness to the predictive process, since it can express the uncertainty about the possible outcomes using a mixture of simple distributions. Being very general, GMDN is also amenable to further extensions, for example by incorporating a recurrent DGN encoder for graphs' time-series prediction. Moreover, the experiments showed that a DGN itself cannot capture any multimodality in the output distribution, due to the implicit assumptions that are usually made about the regression problem. In contrast, GMDN was able to correctly predict different outcomes alongside their likelihood.

6.1 Future Directions

There are a number of potential directions to be investigated in the future, which are mostly methodological and related to the Bayesian nature of the proposed models. One of these concerns the incorporation of supervision into the embeddings' generation process, which could make the presence of a final neural predictor unnecessary in the overall architecture. Another possibility would be to tackle unsupervised learning of temporal graphs, by making the DBGNs accept sequences of observable variables rather than static information. Moreover, the use of Bayesian networks and multimodal distributions may ease the inspection of the inner workings of our models, allowing us to detect whether or not a topological change in the input graph produces lower likelihoods or significantly different multimodal output distributions. Therefore, further studies on the interpretability properties of DBGNs and GMDN are needed to understand the extent to which these models can provide humans with actionable feedback. Finally, in terms of applications, we foresee the use of our models in contexts of scarce supervision and large amounts of raw graphs, as it may happen in chemistry, biology, or social networks' analysis, where pre-trained embeddings can play a decisive role in making the most of the available data.

These final thoughts conclude the dissertation. All in all, we hope that the few contributions made in this thesis will inspire further studies and applications of both generative and predictive approaches for the adaptive processing of structured data.

Appendix A

List of Publications with Code

- Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A deep and generative approach to graph processing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 294–303, 2018
  URL: https://github.com/diningphil/CGMM

- Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations (ICLR)*, 2020
  URL: https://github.com/diningphil/gnn-comparison

- Davide Bacciu, Federico Errica, and Alessio Micheli. Probabilistic learning on graphs via contextual architectures. *Journal of Machine Learning Research*, 21 (134):1–39, 2020
  URL: https://github.com/diningphil/CGMM

- Federico Errica, Davide Bacciu, and Alessio Micheli. Theoretically expressive and edge-aware graph learning. In *Proceedings of the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2020

- Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 9 2020
  URL: https://github.com/diningphil/PyDGN

- Federico Errica, Marco Giulini, Davide Bacciu, Roberto Menichetti, Alessio Micheli, and Raffaello Potestio. A deep graph network–enhanced sampling approach to efficiently explore the space of reduced representations of proteins. *Frontiers in*

Molecular Biosciences, 8:136–150, 2021
URL: https://github.com/CIML-VARIAMOLS/GRAWL

- Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. In *Graph Learning Benchmark Workshop, The Web Conference (WWW)*, 2021
URL: https://github.com/diningphil/continual_learning_for_graphs

- Federico Errica, Davide Bacciu, and Alessio Micheli. Graph mixture density networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 3025–3035, 2021
URL: https://github.com/diningphil/graph-mixture-density-networks

- Federico Errica Daniele Atzeni, Davide Bacciu and Alessio Micheli. Modeling edge features with deep bayesian graph networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021
URL: https://github.com/diningphil/E-CGMM

- Federico Errica, Fabrizio Silvestri, Bora Edizel, Ludovic Denoyer, Fabio Petroni, Vassilis Plachouras, and Sebastian Riedel. Concept matching for low-resource classification. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021
URL: https://github.com/facebookresearch/parcus

- Federico Errica, Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Alessio Micheli. Robust malware classification via deep graph networks on call graph topologies. In *Proceedings of the 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2021
URL: https://github.com/diningphil/robust-call-graph-malware-detection

Appendix B

List of Talks and Posters

- Oral presentation of our ICML 2018 paper [6].
- Oral presentation of our ICLR 2020 paper [5].
- Poster presentation of our ESANN 2020 paper [286].
- Spotlight presentation of our WWW 2021 workshop paper [5].
- Oral presentation of our ICML 2021 paper [9].
- Oral presentations of our IJCNN 2021 papers [8, 288].
- Poster presentation of our ESANN 2021 paper [11].
- Invited talk at IBM Research Zurich, 2021
- Invited talk at ContinualAI, 2021
- Invited talk at NEC Labs Europe, 2021

Bibliography

[1] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 9 2020.

[2] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

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

[4] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9 (5):768–786, 1998.

[5] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations (ICLR)*, 2020.

[6] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A deep and generative approach to graph processing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 294–303, 2018.

[7] Davide Bacciu, Federico Errica, and Alessio Micheli. Probabilistic learning on graphs via contextual architectures. *Journal of Machine Learning Research*, 21 (134):1–39, 2020.

[8] Federico Errica Daniele Atzeni, Davide Bacciu and Alessio Micheli. Modeling edge features with deep bayesian graph networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[9] Federico Errica, Davide Bacciu, and Alessio Micheli. Graph mixture density networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 3025–3035, 2021.

[10] Federico Errica, Marco Giulini, Davide Bacciu, Roberto Menichetti, Alessio Micheli, and Raffaello Potestio. A deep graph network–enhanced sampling approach to efficiently explore the space of reduced representations of proteins. *Frontiers in Molecular Biosciences*, 8:136–150, 2021.

[11] Federico Errica, Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Alessio Micheli. Robust malware classification via deep graph networks on call graph topologies. In *Proceedings of the 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2021.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[13] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[14] Roberto Bruni and Ugo Montanari. *Models of computation*. Springer, 2017.

[15] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[16] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[17] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-6(6):721–741, 1984.

[18] Christopher M Bishop. Mixture Density Networks. Technical report, Aston University, 1994.

[19] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[20] Do E Rumelhart, GE Hinton, and RJ Williams. Learning internal representations by error propagation, parallel distributed processing, vol. 1. *Foundations. MIT Press, Cambridge*, 1986.

[21] Fred M Hoppe. Pólya-like urns and the ewens' sampling formula. *Journal of Mathematical Biology*, 20(1):91–94, 1984.

[22] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994.

[23] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[24] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the american statistical association*, 101(476):1566–1581, 2006.

[25] Peter Orbanz and Yee Whye Teh. Bayesian nonparametric models. *Encyclopedia of machine learning*, 1, 2010.

[26] Yee Whye Teh. *Dirichlet Process*, pages 280–287. Springer US, 2010.

[27] Samuel J Gershman and David M Blei. A tutorial on bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012.

[28] Bruno de Finetti. Funzione caratteristica di un fenomeno aleatorio. *Atti della R. Accademia Nazionale dei Lincei, Ser. 6. Memorie, Classe di Scienze Fisiche, Matematiche e Naturali 4*, pages 251–299, 1931.

[29] Thomas S Ferguson. A bayesian analysis of some nonparametric problems. *The annals of statistics*, pages 209–230, 1973.

[30] David J Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. Springer, 1985.

[31] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pages 281–297. Oakland, CA, USA, 1967.

[32] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[33] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[34] Brendan L Douglas. The Weisfeiler-Lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

[35] Béla Bollobás and Bollobás Béla. *Random graphs*. Cambridge university press, 2001.

[36] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4): 1141–1144, 1959.

[37] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[38] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[39] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):25. 18–42, 2017.

[40] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, and others. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[41] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[42] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2021. doi: 10.1109/TNNLS.2021.3070843.

[43] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

[44] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[45] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[46] Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. klog: A language for logical and relational learning with kernels. *Artificial Intelligence*, 217:117–143, 2014.

[47] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (SIGKDD*, pages 1365–1374. ACM, 2015.

[48] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[49] 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 33rd AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 4602–4609, 2019.

[50] Giovanni Da San Martino, Nicolo Navarin, and Alessandro Sperduti. A tree-based kernel for graphs. In *Proceedings of the 12th International Conference on Data Mining (ICDM)*, pages 975–986. SIAM, 2012.

[51] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. Ordered decompositional DAG kernels enhancements. *Neurocomputing*, 192:92–103, 2016.

[52] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[53] Edmondo Trentin and Ernesto Di Iorio. Nonparametric small random networks for graph-structured pattern recognition. *Neurocomputing*, 313:14–24, 2018.

[54] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 488–495, 2009.

[55] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, Chris Meek, Jennifer Neville, et al. *Introduction to statistical relational learning*. MIT press, 2007.

[56] Luc De Raedt and Kristian Kersting. Statistical relational learning. In *Encyclopedia of Machine Learning and Data Mining*, pages 1177–1187. Springer, 2017.

[57] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[58] Peter Clifford. Markov random fields in statistics. *Disorder in physical systems: A volume in honour of John M. Hammersley*, pages 19–32, 1990.

[59] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, 2001.

[60] Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 5241–5250, 2019.

[61] Veeru Sadhanala, Yu-Xiang Wang, and Ryan Tibshirani. Graph sparsification approaches for laplacian smoothing. In *Artificial Intelligence and Statistics*, pages 1250–1259, 2016.

[62] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2006.

[63] Daniele Calandriello, Ioannis Koutis, Alessandro Lazaric, and Michal Valko. Improved large-scale graph learning through ridge spectral sparsification. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 687–696, 2018.

[64] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[65] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[66] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *2nd International Conference on Learning Representations (ICLR)*, 2014.

[67] Jonathan M. Blackledge. Chapter 2 - 2d fourier theory. In *Digital Image Processing*, pages 30–49. Woodhead Publishing, 2005.

[68] David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.

[69] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS)*, pages 3844–3852, 2016.

[70] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[71] László Lovász and others. Random walks on graphs: A survey. *Combinatorics, Paul Erdos is eighty*, 2(1):1–46, 1993.

[72] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd*

International Conference on Knowledge Discovery and Data Mining (SIGKDD), pages 385–394. ACM, 2017.

[73] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2191–2200, 2018.

[74] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 855–864. ACM, 2016.

[75] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 701–710. ACM, 2014.

[76] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 609–618, 2018.

[77] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5453–5462, 2018.

[78] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumbel-softmax. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[79] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *Proceedings of the 27th International Conference on Artificial Neural Networks (ICANN)*, pages 412–422, 2018.

[80] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *Workshop on Theoretical Foundations and Applications of Deep Generative Models, International Conference on Machine Learning (ICML)*, 2018.

[81] Thomas N Kipf and Max Welling. Variational graph auto-encoders. In *Workshop on Bayesian Deep Learning, Neural Information Processing System (NIPS)*, 2016.

[82] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2434–2444, 2019.

[83] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *2nd International Conference on Learning Representations (ICLR)*, 2014.

[84] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[85] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 7795–7804, 2018.

[86] Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez Rodriguez. NeVAE: A deep generative model for molecular graphs. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1110–1117, 2019.

[87] John Bradshaw, Brooks Paige, Matt J Kusner, Marwin Segler, and José Miguel Hernández-Lobato. A model to search for synthesizable molecules. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 7935–7947, 2019.

[88] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[89] S. Fan and B. Huang. Conditional labeled graph generation with GANs. In *Representation Learning on Graphs and Manifolds Workshop, International Conference on Learning Representations (ICLR)*, 2019.

[90] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph representation learning with generative adversarial nets. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 2508–2515, 2018.

[91] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2609–2615, 2018.

[92] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[93] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[94] Davide Bacciu, Alessio Micheli, and Marco Podda. Graph generation by sequential edge prediction. In *Proceedings of the 27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2019.

[95] Davide Bacciu, Alessio Micheli, and Marco Podda. Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing. Accepted*, 2019.

[96] Marco Podda, Davide Bacciu, and Alessio Micheli. A deep generative model for fragment-based molecule generation. In *Proceedings of the 23rd Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2240–2250, 2020.

[97] Marco Podda. *Deep Learning on Graphs with Applications to the Life Sciences.* PhD thesis, University of Pisa, 2021.

[98] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 2847–2856. ACM, 2018.

[99] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[100] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *Proceedings of the 25th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 246–256, 2019.

[101] Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[102] Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based graph convolutional network. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4054–4061, 2019.

[103] H. Jin and X. Zhang. Latent adversarial training of graph convolution networks. In *Workshop on Learning and Reasoning with Graph-Structured Representations, International Conference on Machine Learning (ICML)*, 2019.

[104] Daniel Zügner and Stephan Günnemann. Certifiable robustness of graph convolutional networks under structure perturbations. In *Proceedings of the 26th ACM*

International Conference on Knowledge Discovery and Data Mining (SIGKDD), pages 1656–1665, 2020.

[105] Yingxue Zhang, Florence Regol, Soumyasundar Pal, Sakif Khan, Liheng Ma, and Mark Coates. Detection and defense of topological adversarial attacks on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2989–2997. PMLR, 2021.

[106] Avishek Bose and William Hamilton. Compositional fairness constraints for graph embeddings. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 715–724, 2019.

[107] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[108] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *8th International Conference on Learning Representations (ICLR)*, 2020.

[109] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations (ICLR)*, 2019.

[110] Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[111] Andreas Loukas. How hard is to distinguish graphs with graph neural networks? In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 3465–3476, 2020.

[112] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[113] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[114] Floris Geerts, Filip Mazowiecki, and Guillermo Perez. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 3640–3649, 2021.

[115] Barbara Hammer, Alessio Micheli, and Alessandro Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17(5): 1109–1159, 2005.

[116] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, pages 3391–3401, 2017.

[117] Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 6487–6494, 2019.

[118] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. An input–output hidden Markov model for tree transductions. *Neurocomputing*, 112:34–46, 2013.

[119] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[120] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[121] Anna Maria Bianucci, Alessio Micheli, Alessandro Sperduti, and Antonina Starita. Application of cascade correlation networks for structures to chemistry. *Applied Intelligence*, 12(1-2):117–147, 2000.

[122] Michelangelo Diligenti, Paolo Frasconi, and Marco Gori. Hidden tree Markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):519–523, 2003.

[123] Alessio Micheli, Diego Sona, and Alessandro Sperduti. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, 15(6):1396–1410, 2004.

[124] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Compositional generative mapping for tree-structured data - part I: Bottom-up probabilistic modeling of trees. *IEEE Transactions on Neural Networks and Learning Systems*, 23(12):1987–2002, 2012.

[125] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Modeling bi-directional tree contexts by generative transductions. In *Proceedings of the 21st International*

Conference on Neural Information Processing (ICONIP), pages 543–550. Springer, 2014.

[126] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

[127] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[128] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks (ICNN)*, volume 1, pages 347–352. IEEE, 1996.

[129] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. A generative multiset kernel for structured data. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 57–64. Springer, 2012.

[130] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[131] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2688–2697, 2018.

[132] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[133] Dana Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th annual ACM symposium on Theory of computing*, pages 82–93, 1980.

[134] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[135] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[136] David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5:1–1, 2000.

[137] 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 (ICML)*, pages 1263–1272, 2017.

[138] Yann LeCun, Yoshua Bengio, and others. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[139] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, (ICLR)*, 2016.

[140] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[141] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[142] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[143] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[144] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *Representation Learning on Graphs and Manifolds Workshop, International Conference on Learning Representations (ICLR)*, 2019.

[145] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019.

[146] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *Proceedings of the 3rd Conference on Neural Information Processing Systems (NIPS)*, pages 524–532, 1990.

[147] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the 15th European Semantic Web Conference (ESWC)*, pages 593–607. Springer, 2018.

[148] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3693–3702, 2017.

[149] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[150] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[151] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, pages 1024–1034, 2017.

[152] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 13260–13271, 2020.

[153] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[154] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[155] Hongyang Gao and Shuiwang Ji. Graph U-nets. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2083–2092, 2019.

[156] Michael Truong Le Frederik Diehl, Thomas Brunner and Alois Knoll. Towards graph pooling by edge contraction. In *Workshop on learning and reasoning with graph-structured data, International Conference on Machine Learning (ICML)*, 2019.

[157] 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.

[158] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[159] Davide Bacciu and Luigi Di Sotto. A non-negative factorization approach to node pooling in graph convolutional neural networks. In *AI*IA 2019 – Advances in Artificial Intelligence*, pages 294–306. Springer, 2019.

[160] Davide Bacciu, Alessio Conte, Roberto Grossi, Francesco Landolfi, and Andrea Marino. K-plex cover pooling for graph neural networks. In *Workshop on Learning Meets Combinatorial Algorithms, Neural Information Processing Systems (NeurIPS)*, 2020.

[161] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 2220–2231, 2020.

[162] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[163] Edmondo Trentin and Leonardo Rigutini. A maximum-likelihood connectionist model for unsupervised learning over graphical domains. In *Proceedings of the 12th International Conference on Artificial Neural Networks (ICANN)*, pages 40–49. Springer, 2009.

[164] Marco Bongini, Leonardo Rigutini, and Edmondo Trentin. Recursive neural networks for density estimation over generalized random graphs. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5441–5458, 2018.

[165] Chen Ma, Liheng Ma, Yingxue Zhang, Jianing Sun, Xue Liu, and Mark Coates. Memory augmented graph neural networks for sequential recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5045–5052, 2020.

[166] Thin Nguyen, Hang Le, Thomas P Quinn, Tri Nguyen, Thuc Duy Le, and Svetha Venkatesh. GraphDTA: Predicting drug–target binding affinity with graph neural networks. *Bioinformatics*, 10 2020.

[167] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2861–2868, 2020.

[168] Sofus A Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8(May):935–983, 2007.

[169] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[170] Markus Hagenbuchner, Alessandro Sperduti, and Ah Chung Tsoi. A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, 14(3):491–505, 2003.

[171] Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. A general framework for unsupervised processing of structured data. *Neurocomputing*, 57:3–35, 2004.

[172] Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8-9):1061–1085, 2004.

[173] Michel Neuhaus and Horst Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):503–514, 2005.

[174] Markus Hagenbuchner, Alessandro Sperduti, and Ah Chung Tsoi. Graph self-organizing maps for cyclic and unbounded graphs. *Neurocomputing*, 72(7-9):1419–1430, 2009.

[175] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 889–898, 2017.

[176] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019*, 2019.

[177] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Workshop on Relational Representation Learning, Neural Information Processing Systems (NeurIPS)*, 2018.

[178] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5829–5836, 2019.

[179] Soumyasundar Pal, Saber Malekmohammadi, Florence Regol, Yingxue Zhang, Yishi Xu, and Mark Coates. Non parametric graph learning for bayesian graph neural networks. In *Conference on Uncertainty in Artificial Intelligence*, pages 1318–1327. PMLR, 2020.

[180] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *8th International Conference on Learning Representations (ICLR)*, 2020.

[181] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 4171—-4186, 2019.

[182] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021.

[183] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3734–3743, 2019.

[184] Drew McDermott. Artificial intelligence meets natural stupidity. *ACM SIGART Bulletin*, 57:4–9, 1976.

[185] Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship. *ACM Queue*, 17(1):80, 2019.

[186] Engineering National Academies of Sciences and Medicine. *Reproducibility and replicability in science*. National Academies Press, 2019.

[187] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys)*, pages 101–109, 2019.

[188] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147, 1974.

[189] Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7:91–98, 02 2006.

[190] Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.

[191] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[192] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[193] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[194] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[195] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(suppl_1), 2004.

[196] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[197] Jacques van Rossum. The Relation Between Chemical Structure and Biological Activity. *Journal of Pharmacy and Pharmacology*, 15(1):285–316, 1963.

[198] 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. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 22118–22133, 2020.

[199] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[200] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. i. general method. *The Journal of Chemical Physics*, 31(2):459–466, 1959.

[201] Martin Karplus. Molecular dynamics simulations of biomolecules. *Accounts of Chemical Research*, 35(6):321–323, 2002.

[202] Siewert J Marrink, H Jelger Risselada, Serge Yefimov, D Peter Tieleman, and Alex H De Vries. The martini force field: coarse grained model for biomolecular simulations. *The journal of physical chemistry B*, 111(27):7812–7824, 2007.

[203] Shoji Takada. Coarse-grained molecular simulations of large biomolecules. *Curr. Opin. Struct. Biol.*, 22(2):130–137, 2012.

[204] Raffaello Potestio, Christine Peter, and Kurt Kremer. Computer simulations of soft matter: Linking the scales. *Entropy*, 16(8):4199–4245, 2014.

[205] Marissa G Saunders and Gregory A Voth. Coarse-graining methods for computational biology. *Annu. Rev. Biophys.*, 42:73–93, 2013.

[206] William George Noid, Jhih-Wei Chu, Gary S Ayton, Vinod Krishna, Sergei Izvekov, Gregory A Voth, Avisek Das, and Hans C Andersen. The multiscale coarse-graining method. i. a rigorous bridge between atomistic and coarse-grained models. *The Journal of chemical physics*, 128(24):244114, 2008.

[207] William George Noid. Systematic methods for structurally consistent coarse-grained models. In *Biomolecular Simulations*, pages 487–531. Springer, 2013.

[208] M. Scott Shell. The relative entropy is fundamental to multiscale and inverse thermodynamic problems. *J. Chem. Phys.*, 129(14):144108, 2008.

[209] Sebastian Kmiecik, Dominik Gront, Michal Kolinski, Lukasz Wieteska, Aleksandra Elzbieta Dawid, and Andrzej Kolinski. Coarse-grained protein models and their applications. *Chemical reviews*, 116(14):7898–7936, 2016.

[210] Michael A. Webb, Jean-Yves Delannoy, and Juan J. de Pablo. Graph-based approach to systematic molecular coarse-graining. *Journal of Chemical Theory and Computation*, 15(2):1199–1208, 2019.

[211] Tristan Bereau and Kurt Kremer. Automated parametrization of the coarse-grained martini force field for small organic molecules. *Journal of chemical theory and computation*, 11(6):2783–2791, 2015.

[212] Teemu Murtola, Mikko Kupiainen, Emma Falck, and Ilpo Vattulainen. Conformational analysis of lipid molecules by self-organizing maps. *The Journal of chemical physics*, 126(5):054707, 2007.

[213] Wujie Wang and Rafael Gómez-Bombarelli. Coarse-graining auto-encoders for molecular dynamics. *npj Computational Materials*, 5(1):1–9, 2019.

[214] Zhiheng Li, Geemi P Wellawatte, Maghesree Chakraborty, Heta A Gandhi, Chenliang Xu, and Andrew D White. Graph neural network based coarse-grained mapping prediction. *Chemical Science*, 11(35):9524–9531, 2020.

[215] Marco Giulini, Roberto Menichetti, M. Scott Shell, and Raffaello Potestio. An information-theory-based approach for optimal model reduction of biomolecules. *Journal of Chemical Theory and Computation*, 16(11):6795–6813, 2020.

[216] Thomas T Foley, M Scott Shell, and William George Noid. The impact of resolution upon entropy and information in coarse-grained models. *The Journal of chemical physics*, 143(24):243104, 2015.

[217] Joseph F. Rudzinski and W. G. Noid. Coarse-graining entropy, forces, and structures. *The Journal of Chemical Physics*, 135(21):214101, 2011.

[218] M. Scott Shell. Systematic coarse-graining of potential energy landscapes and dynamics in liquids. *J. Chem. Phys.*, 137(8):084503, 2012.

[219] Fugao Wang and DP Landau. Determining the density of states for classical statistical models: A random walk algorithm to produce a flat histogram. *Physical Review E*, 64(5):056101, 2001.

[220] Fugao Wang and David P Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical review letters*, 86(10):2050, 2001.

[221] M Scott Shell, Pablo G Debenedetti, and Athanassios Z Panagiotopoulos. Generalization of the wang-landau method for off-lattice simulations. *Physical review E*, 66(5):056703, 2002.

[222] L Yu Barash, MA Fadeeva, and LN Shchur. Control of accuracy in the wang-landau algorithm. *Physical Review E*, 96(4):043307, 2017.

[223] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[224] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[225] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Bottom-up generative modeling of tree-structured data. In *Proceedings of the 17th International Conference on Neural Information Processing (ICONIP)*, 2010.

[226] 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 (CVPR)*, 2016.

[227] Lawrence K Saul and Michael I Jordan. Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, 37(1):75–87, 1999.

[228] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Generative kernels for tree-structured data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4932–4946, 2018.

[229] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 2014–2023, 2016.

[230] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 378–393. Springer, 2012.

[231] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS)*, pages 1993–2001, 2016.

[232] Dinh V Tran, Nicolò Navarin, and Alessandro Sperduti. On filter size in graph convolutional networks. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1534–1541. IEEE, 2018.

[233] Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5475–5485, 2018.

[234] Lorenz C Blum and Jean-Louis Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database gdb-13. *Journal of the American Chemical Society*, 131(25):8732–8733, 2009.

[235] Matthias Rupp, Alexandre Tkatchenko, Klaus Robert Müller, and O Anatole Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters*, 108(5):058301, 2012.

[236] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305, 2012.

[237] Chong Wang and David Blei. Truncation-free stochastic variational inference for bayesian nonparametric models. *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, pages 422–430, 2012.

[238] Michael Bryant and Erik Sudderth. Truly nonparametric online variational inference for hierarchical dirichlet processes. *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, pages 2699–2707, 2012.

[239] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

[240] Michael Hughes, Dae Il Kim, and Erik Sudderth. Reliable and scalable variational inference for the hierarchical dirichlet process. In *Proceedings of the 18th Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 370–378. PMLR, 2015.

[241] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. The sticky hdp-hmm: Bayesian nonparametric hidden markov models with persistent states. *Preprint*, 2007.

[242] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. An hdp-hmm for systems with state persistence. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 312–319, 2008.

[243] Dan Lovell, Ryan P. Adams, and Vikash K. Mansinghka. Parallel markov chain monte carlo for dirichlet process mixtures. In *Workshop on Big Learning, Advances in Neural Information Processing Systems (NIPS)*, 2012.

[244] Sinead Williamson, Avinava Dubey, and Eric Xing. Parallel Markov chain Monte Carlo for nonparametric mixture models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, pages 98–106. PMLR, 2013.

[245] Jason Chang and John W Fisher III. Parallel sampling of hdps using sub-cluster splits. In *Advances in Neural Information Processing Systems (NIPS)*, volume 27. Curran Associates, Inc., 2014.

[246] Hong Ge, Yutian Chen, Moquan Wan, and Zoubin Ghahramani. Distributed inference for dirichlet process mixture models. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2276–2284. PMLR, 2015.

[247] Yarin Gal and Zoubin Ghahramani. Pitfalls in the use of parallel inference for the dirichlet process. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, volume 32, pages 208–216. PMLR, 22–24 Jun 2014.

[248] José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[249] Prem K. Goel and Morris H. Degroot. Information about hyperparamters in hierarchical models. *Journal of the American Statistical Association*, 76(373):140–147, 1981. URL http://www.jstor.org/stable/2287059.

[250] Alessandro Bacci, Alberto Bartoli, Fabio Martinelli, Eric Medvet, and Francesco Mercaldo. Detection of obfuscation techniques in android applications. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES)*, 2018.

[251] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. Droidsieve: Fast and accurate classification of obfuscated android malware. In *Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy (CODASPY)*, pages 309–320, 2017.

[252] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security*, 51:16–31, 2015.

[253] Shanhu Shang, Ning Zheng, Jian Xu, Ming Xu, and Haiping Zhang. Detecting malware variants via function-call graph similarity. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2010.

[254] Ammar Ahmed E Elhadi, Mohd Aizaini Maarof, Bazara IA Barry, and Hentabli Hamza. Enhancing the detection of metamorphic malware using call graphs. *computers & security*, 46:62–78, 2014.

[255] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security (AISec)*, 2013.

[256] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Call graph and model checking for fine-grained android malicious behaviour detection. *Applied Sciences*, 10(22):7975–7994, 2020.

[257] Gerardo Canfora, Andrea De Lorenzo, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Effectiveness of opcode ngrams for detection of multi family android malware. In *Proceedings of the 10th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2015.

[258] Akshay Kapoor and Sunita Dhavale. Control flow graph based multiclass malware detection using bi-normal separation. *Defence Science Journal*, 66(2):138–145, 2016.

[259] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie J. Hendren, Patrick Lam, and Vijay Sundaresan. Soot: A java bytecode optimization framework. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, 1999.

[260] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Towards an interpretable deep learning model for mobile malware detection and family identification. *Computers & Security*, 105:102198–103012, 2021.

[261] Christopher N Davis, T Deirdre Hollingsworth, Quentin Caudron, and Michael A Irvine. The use of mixture density networks in the emulation of complex epidemiological individual-based models. *PLoS computational biology*, 16(3):e1006869, 2020.

[262] William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

[263] Ghanshyam Pilania, Chenchen Wang, Xun Jiang, Sanguthevar Rajasekaran, and Ramamurthy Ramprasad. Accelerating materials property predictions using machine learning. *Scientific reports*, 3(1):1–6, 2013.

[264] Marek Opuszko and Johannes Ruhland. Impact of the network structure on the SIR model spreading phenomena in online networks. In *Proceedings of the 8th International Multi-Conference on Computing in the Global Information Technology (ICCGI'13)*, 2013.

[265] Juliette Valenchon and Mark Coates. Multiple-graph recurrent graph convolutional neural network architectures for predicting disease outcomes. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3157–3161. IEEE, 2019.

[266] Krzysztof Nowicki and Tom A B Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American statistical association*, 96(455):1077–1087, 2001.

[267] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[268] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[269] Adrian Corduneanu and Christopher M Bishop. Variational bayesian model selection for mixture distributions. In *Artificial intelligence and Statistics*, volume 2001, pages 27–34. Morgan Kaufmann Waltham, MA, 2001.

[270] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *International Conference on Learning Representations (ICLR) Workshop*, 2013.

[271] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *International Conference on Learning Representations (ICLR) Workshop*, 2017.

[272] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[273] Marko Helén and Tuomas Virtanen. Query by example of audio signals using euclidean distance between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 1, pages 1–225. IEEE, 2007.

[274] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Proceedings of the 16th Conference on Neural Information Processing Systems (NIPS)*, pages 857–864, 2002.

[275] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[276] Kisung You. l_2 distance between two gaussian mixture models. 2021. URL https://kisungyou.com/notes/note004/main004.pdf.

[277] KB Petersen and MS Pedersen. The matrix cookbook, version 20121115. *Technical Univ. Denmark, Kongens Lyngby, Denmark, Tech. Rep*, 3274, 2012.

[278] Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. Community structure and scale-free collections of erdős-rényi graphs. *Physical Review E*, 85(5):056109, 2012.

[279] Guangyong Chen, Pengfei Chen, Chang-Yu Hsieh, Chee-Kong Lee, Benben Liao, Renjie Liao, Weiwen Liu, Jiezhong Qiu, Qiming Sun, Jie Tang, et al. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*, 2019.

[280] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

[281] Xavier Bresson and Thomas Laurent. A two-step graph convolutional decoder for molecule generation. In *Workshop on Machine Learning and the Physical Sciences, Neural Information Processing Systems (NeurIPS)*, 2019.

[282] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1907–1913. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/264.

[283] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural

networks. In *Proceedings of the 26th ACM International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 753–763, 2020.

[284] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[285] Sungjoon Choi, Kyungjae Lee, Sungbin Lim, and Songhwai Oh. Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6915–6922. IEEE, 2018.

[286] Federico Errica, Davide Bacciu, and Alessio Micheli. Theoretically expressive and edge-aware graph learning. In *Proceedings of the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2020.

[287] Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. In *Graph Learning Benchmark Workshop, The Web Conference (WWW)*, 2021.

[288] Federico Errica, Fabrizio Silvestri, Bora Edizel, Ludovic Denoyer, Fabio Petroni, Vassilis Plachouras, and Sebastian Riedel. Concept matching for low-resource classification. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.