A GENERATIVE MODEL FOR LATENT POSITION GRAPHS

by

Vittorio Loprinzo

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
January 2023

Abstract

Recently, there has been an explosion of research into machine learning methods applied to graph data. Most work is focused on performing either node classification or graph classification; however, there is much to be gained by learning instead a generative model for the underlying random graph distribution. We present a novel neural network-based approach to learning generative models for random graphs. The features used for training are graphlets, subgraph counts of small order, and the loss function is based on a moment estimator for these features. Random graphs are realized by feeding random noise into the network and applying a kernel to the output; in this way, our model is a generalization of the ubiquitous Random Dot Product Graph. Networks produced this way are demonstrated to be able to imitate data from chemistry, medicine, and social networks. The created graphs are similar enough to the target data to be able to fool discriminator neural networks otherwise capable of separating classes of random graphs. This method is inexpensive, accurate, and is readily applied to data-poor problems.

Thesis Readers

Dr. Laurent Younes (Primary Advisor)
Professor
Department of Applied Mathematics & Statistics
Johns Hopkins University

Dr. Soledad Villar
Assistant Professor
Department of Applied Mathematics & Statistics
Johns Hopkins University

Committee Members

Dr. Daniel Q. Naiman
Professor
Department of Applied Mathematics & Statistics
Johns Hopkins University

Dr. Donald Geman
Professor
Department of Applied Mathematics & Statistics
Johns Hopkins University

Dr. Zachary M. Pisano
Assistant Research Professor
Department of Applied Mathematics & Statistics
Johns Hopkins University

This page intentionally left blank.

This dissertation is dedicated to my late grandfather, Americo Marinelli, as well as his parents and all the rest of my ancestors who immigrated from Italy. It was his relentless encouragement of my schooling that pushed me to finish a PhD, and their sacrifices which afforded me the opportunity in the first place. Grandpa, thanks for always telling me I was the "smartest kid on the block."

This page intentionally left blank.

Acknowledgements

I have many people to acknowledge:

First and foremost, to my parents, Rob and Gina Loprinzo, who have given me everything I've ever needed for the life I live and have supported me from Day 0 to Day -1. And to my sister, Rose Loprinzo, who inspires me despite walking a path very opposite to my own.

My graduate school friends and colleagues: Zach Pisano, Joshua Cape, Joshua Agterberg, Jason Miller, Phil Kerger, Sieu Tran, and Melissa Diamond, as well as Ryan Walter and Nicole Steinberg.

My girlfriend and partner, Julia Kuhlman, who has been steadfast in her support and gives me great joy as I look forward to the next chapter.

All the members of my dance team, The Eclectics, whose relentless positivity kept my spirits high all throughout grad school.

Justin Massey and Susan Bailey, without whom I certainly would have never made it past the beginning of my degree, let alone to the end.

Dan Naiman, who provided endless support on every issue, academic and not, that arose in grad school, and treated me as an equal and a friend.

And finally, my advisor, Laurent Younes, who guided me to success, as well as Don Geman, who assisted in that process. Without you, this thesis would not be possible.

There are numerous others who deserve mentioning; any list I could give would almost surely be incomplete. To all the rest: Thank you.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Tables

List of Figures

Chapter 1

Notation and Overview

1.1 Introduction

It has recently grown apparent that network data is all around us. From social networks to networks of protein interactions, more and more fields are discovering the utility of data structured as graphs, consisting of vertices representing objects and edges codifying their relationships. As the amount of available data has grown, so too has the corresponding branch of data science. Statistical learning on graphs is a burgeoning field, with new ideas introduced constantly, including both original designs and generalizations of machine learning methods on traditional objects.

The contribution of this dissertation is one such idea: We will introduce a generative model for random graph distributions based on sufficient statistics called graphlets. The model is built on a neural network foundation, and has the capability of generating graphs end-to-end, from its edge structure to the attributes assigned to its vertices and edges. We will carefully lay out the model, apply it to a number of real datasets for validation, and analyze results.

1.2 Notation

In general, introduced notation that uses the Roman alphabet will be chapter-specific, while notation that uses the Greek alphabet will be considered defined for the span of the dissertation. However, we will depart from this occasionally to adhere to some conventions of the field.

$G = (V, E)$ will always represent a graph, where V and E are the graph's vertex and edge set, respectively. $\mathfrak{G}$ will represent a set of random graphs drawn from distribution $\mathbb{G}$. n will always represent the number of vertices in the graph being discussed.

In general, capital letters will be used to represent random variables, while lower case letters will represent real variables or realizations of the corresponding random variable. To avoid any ambiguity, other mathematical objects, such as sets or nonrandom matrices, will be represented by symbols in other scripts, e.g. $\mathcal{A}$.

We will denote the real numbers by $\mathbb{R}$ and the integers by $\mathbb{Z}$. $\mathcal{I}_n$ will represent the $n \times n$ identity matrix, and we will drop the suffix when it is understood from context. We will always reserve i and j as index variables. If X is an array, then X^T will represent the transpose of that array.

A list of all notations defined for the duration of the dissertation can be found in Table 1-I.

1.3 Organization

This dissertation will begin in Chapter 2 with a historical overview of the field of random graphs, as well as some background on relevant methods from machine learning. It will also provide an overview of the state of the art of the thesis' subject matter by discussing inference on graphs, generative models, and competing methods.

Chapter 3 will introduce the Visual Turing Test, a motivating problem that underlies most of the work contained herein.

Chapter 4 will contain the main body of the work, introducing a novel way to learn generative models for random graph distributions. The model is based on graphlets, subgraph counts of small size.

Chapter 5 will discuss extensions to the main model to graphs with vertex and edge attributes.

Chapter 6 will give the implementation and results from both the main model described in Chapter 4 and the extensions to attributed graphs in Chapter 5.

Chapter 7 will conclude the thesis with discussion and suggestions for future directions of research.

References can be found at the end of each chapter, and all references are collected at the end of the dissertation.

Symbol	Meaning	Chapter defined
G	A graph	1
$\mathfrak{G}$	Sample of random graphs	1
$\mathbb{G}$	Random graph distribution	1
V	Vertex set	1
E	Edge set	1
n	Number of vertices	1
$\mathbb{R}$	Real numbers	1
$\mathbb{Z}$	Integers	1
$\mathbf{\Gamma}$	SBM edge probability matrix	2
Π	SBM block probability	2
ϕ	kernel function	2
λ	Loss function	2
θ	Machine learning parameters	2
$\mu(\cdot)$	Loss function	2
$\tau(\cdot)$	ReLU function	2
$\sim$	Graph isomorphism	4
$[n]$	Set of first n integers	4
$[n]_p$	Subsets of $[n]$ of size p	4
H	Graphlets	4
H_F	Graphlet isomporphic to graph F	4
Ω	Random noise	4
Ξ	Diagonal matrix of weights	4
Ψ_1	First neural network	4
Ψ_2	Second neural network	5
Δ	Vertex attribute vocabulary	5
Λ	Edge attribute vocabulary	5

Table 1-I. A list of notations defined for the duration of the dissertation.

Chapter 2

Background

2.1 Random Graphs

2.1.1 Erdos-Renyi Graphs

The study of edge random graphs, i.e. graphs whose vertices are fixed but whose edges appear according to some probability distribution, dates to 1959, when they were introduced independently by Gilbert [1] and by Erdos and Renyi [2]. In the simplest of terms, a random graph is one for which we associate a binary random variable X_{ij} with each pair of vertices i and j in the graph, and include an edge between two vertices if their corresponding variable takes value 1. In the case that all the X_{ij} are independent and identically distributed, the model is called an Erdos-Renyi random graph. This model effectively utilizes only two parameters - namely, the number of vertices and the common edge probability - to create graphs.

Despite its simplicity, the Erdos-Renyi model exhibits a great deal of interesting emergent behavior. We can easily calculate the probability that the ER model generates a particular graph G; if we call n the number of vertices in G, p the common edge probability, and let b be the number of edges in the G, then the probability of creating that graph is simply

$$p^b(1-p)^{\binom{n}{2}-b}.$$

The degree of any given vertex in an ER graph follows the binomial distribution with parameters $(n-1)$ and p.

ER graphs were studied further by Erdos and Renyi in [3], where the primary focus was on connectedness of the created graphs for various values of the parameter p. If we set

$$p = c\frac{\log n}{n},$$

then the value of c determines the graphs's behavior as $n \to \infty$: If $c > 1$, the graph is almost surely connected, and if $c < 1$, the graph is almost surely disconnected. $\frac{\log n}{n}$ is often called the *percolation threshold* for the ER model, and the concept of percolation has been heavily studied for a variety of graph models built on top of the Erdos-Renyi [4] [5].

Beyond the simplicity of the Erdos-Renyi, a great many models for random graphs have been proposed and studied over the past six decades. The two broadest categories for models are those that are based on creating community structure, and those that are designed to enforce certain values for sets of sufficient statistics.

2.1.2 Stochastic Block Models

Among the most popular today on the community structure side is the Stochastic Block Model (SBM), first introduced in 1983 by Holland et al. [6]. Here, to each of the n vertices i we associate a latent variable Q_i, taking values in $[1, 2, \ldots, m]$ for some choice of a parameter $m \leq n$. These variables are usually independent and identically distributed, and are understood to represent the *community* to which each vertex belongs. The model is designed to create differing edge structure for nodes within communities and between them; indeed, $P(X_{ij} = 1)$ depends on, and only on, the values of Q_i and Q_j; the X_{ij} are conditionally independent given the Π_i. In fact, the parameters of an SBM are commonly represented in a matrix $\mathbf{\Gamma}$ of size $m \times m$, with

$$P(X_{ij} = 1) = \mathbf{\Gamma}_{Q_i Q_j}.$$

This matrix of m^2 parameters, plus the $m-1$ parameters that describe the probability distribution for the Π random variables, fully define the model.

Typical uses of the Stochastic Block Model include the study of social networks, though they have been employed in many places, including the physical sciences and medicine [7] [8]. Most commonly, we see the diagonal entries of the matrix $\mathbf{\Gamma}$ larger than the off-diagonal entries; this represents the common situation in nature where individuals belonging to the same community are more likely to hold a relationship than those in separate communities. It is also common to see all diagonal and all off-diagonal entries of the matrix to be respectively identical, in which case the model is sometimes called the *planted partition model*, or more simply, *symmetric*.

The SBM exhibits many of the same characteristics as the ER model. While it is no longer tractable to compute the probability of any given graph being produced, some other calculations remain feasible. For instance, the expected vertex degree for any node in an SBM is given as

$$(n-1)\sum_{j=1}^{m} P(Q=j)\sum_{k=1}^{m} \mathbf{\Gamma}_{jk} P(Q=k).$$

We note that the ER model is a special case of the SBM, where all entries of $\mathbf{\Gamma}$ are equal.

2.1.3 Random Dot Product Graphs

It is the proverbial older brother of the Stochastic Block Model that we make use of most in this work. The Random Dot Product Graph (RDPG) [9] shares many properties with the SBM, and is in some cases a direct supermodel of it. The RDPG was introduced in 1998 by Fiduccia, Scheinerman et al. [10]. To each node i in the graph, we associate a latent vector $z_i \in \mathbb{R}^d$ for some choice of dimension d. We then set $P(X_{ij}=1)$ to be equal to the normalized dot product of z_i with z_j, i.e.

$$P(X_{ij}=1) = \frac{z_i^T z_j}{|z_i|\,|z_j|}.$$

In this way, nodes whose latent vectors (henceforth "embeddings") are near each other in real space are more likely to share an edge.

It is quite common to consider the embeddings z to be random variables rather than fixed quantities. Early work on RDPGs considered embeddings distributed over the unit sphere [11] [12], and some structural results, such as the diameter of the largest connected component or clustering of nodes, were quickly proved. Later work has focused more on statistical inference on RDPGs [9]; numerous theorems exist about recovery of the embeddings and comparison of distributions.

2.1.4 Latent Position Graphs

While the RDPG is already quite a flexible model, it extends rather easily to a more general class of models that have the ability to more easily capture a wider variety of graph behaviors. We point out that, once the latent vectors z_i are set, there is no reason that the way we use them be limited to inner products; instead, we could choose to apply any of a variety functions to pairs of latent vectors.

We define a *kernel* to be a symmetric function $\phi : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$, that is, any symmetric function which accepts two inputs from the latent vector space and returns a probability. We call a kernel positive semidefinite if

$$\sum_{i=1}^{k} \sum_{j=1}^{k} a_i a_j \phi(z_i, z_j) \geq 0$$

holds for every $z_1 \dots z_k \in \mathbb{R}^d$ and $a_1 \dots a_k \in \mathbb{R}$. We further call a kernel positive definite if equality in the above implies that $a_i = 0 \;\; \forall i$. We note that we will not in general require that any kernel we use in this work be either positive semidefinite or positive definite unless otherwise specified.

If we replace the Euclidean inner product in the RDPG with a more general kernel, the result remains a generative model for random graphs. This class of model has a variety of names in the literature, but we will choose to call it the *latent position* *model.*

In [13], the authors demonstrate that the latent position model is inherently nonidentifiable. A kernel function ϕ is called invariant to some transformation q if $\phi(q(z_i), q(z_j)) = \phi(z_i, z_j)$, which also would immediately imply the model is not identifiable. Many kernels have this property for one or more transformations; for example, the usual inner product kernel is invariant to angle-preserving rotations. This may present difficulties when learning a set of embeddings z; our future loss function will not have a unique minimum.

It was shown in [11] that latent position graphs can be approximated arbitrarily well by what the authors call 'vertex random graphs'. In a vertex random graph, latent positions are randomly selected as above, but the existence of edges is nonrandom; it is instead determined by a deterministic function of the latent positions. Despite this, we will choose to retain the randomness in both the edges in vertices and edges inherent in the latent position model to encourage variation in our results.

Both the ER model and some varieties of SBM are submodels of the latent position model. The ER model corresponds to the situation where the kernel function is constant (so the embedding becomes irrelevant). The SBM can be recovered by allowing all vertices assigned to the same community to share an embedding, an idea that will be explored in detail in Chapter 4.

2.1.5 Exponential Random Graphs

For sufficient statistic-focused methods, the most ubiquitous choice is the Exponential Random Graph Model (ERGM) [14, 15]. Introduced in the late 1980s, the ERGM is designed to maximize Gibbs Entropy for the probability distribution over all possible graphs of a certain size. The resulting set of probability distributions is an exponential family.

The ERGM focuses on a vector of sufficient statistics s for the random graph distribution, with the assumption that all information about the dependence between edge probabilities is contained in the vector. The sufficient statistics used could be characteristics of the graph as a whole, such as connectivity or number of subgraphs of a certain type, or can be more specific about each node, such as individual node degrees. In any case, the probability that an ERGM produces a graph G on n nodes is given by

$$P(G|W) = \frac{\exp(W^T s(G))}{\sum_{G'} \exp(W^T s(G'))},$$

where W is a parameter vector and the sum is over all possible graphs on n nodes.

ERGMs should be particularly useful when there is not much information available about the nature of the random graphs in question. Instead of a broad understanding of the underlying structure of the graphs, the model uses only the values of a curated set of sufficient statistics to represent their distribution. The model further has the advantage of being highly interpretable. However, in the next chapter we will discuss the difficulties associated with learning these models in practice.

2.2 Machine Learning and Neural Networks

The field of statistical and machine learning has grown rapidly in the past few decades due to the near-universal applicability of its methods and their feasibility thanks to modern computing. In this work, we make heavy use of certain methods from machine learning, especially neural networks.

Machine learning generally falls into two broad categories: Supervised and unsupervised learning. In supervised ML, we generally have some kind of function that we want to approximate; that function may be a data classifier, a value predictor, a probabilistic density, or one of any number of other things. To estimate, or ‘learn’ that function, we are provided with a sample of evaluations of it; we may know its value at a number of inputs. We leverage this knowledge to build a working imitation of that function. In the unsupervised case, there is usually no specific target function; instead, given a set of data, we attempt to extract patterns from it. This may mean classifying data into groups that were not prespecified, or reducing the dimensionality of large data. While both settings are meaningful, we will focus entirely on the former in this work.

Our general setting for supervised machine learning will be this: Given a model parameterized by θ, a set of data $\mathcal{G}$, and a loss function $\lambda(\cdot)$, we seek to optimize θ to minimize λ.

The neural network is now so widely used that it warrants little discussion on its own; we provide only a brief overview of its function. Broadly speaking, neural networks are (or at least appear to be) universal function approximators [16, 17]. Through optimization of a set of parameters θ by means of the gradient descent algorithm or any of numerous other related methods, neural networks 'learn' to imitate a function from which we possess a number of sample points (which are usually tarnished by some added noise). (In this work, the sample points, or 'data', will be a set of random graphs, but in general could be anything from grayscale images to word counts in a novel.) Creation of a neural net requires two things to be carefully chosen: First, a good choice of the loss function λ whose minimization corresponds to desirable results; second, an *architecture* for the network that is conducive to learning the needed function.

The architecture of a neural network refers to the set of iterated functions that compose the overall action of the network on its data. Usually, these so-called layers are a series of relatively simple functions. Examples include the fully-connected layer, which is little more than the multiplication of the data by a matrix; convolutional layers, which involve passing a filter over each section of spatially-structured data; pooling layers, which involve summing pieces of the data together; and activation layers, which involve applying a nonlinear function to the data. Activation layers are what give neural nets all their power, and the field has developed a few favorites, including the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

and the Rectified Linear Unit, or ReLU:

$$\tau(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases},$$

both of which we make use of in this work.

There are dozens of other layer types and thousands of published architectures for neural networks in existence; there is nothing to say whether one is superior to another other than demonstrated outperformances on given tasks. Effective design of neural networks is currently more of an art than a science.

2.3 Statistical Learning on Random Graphs

In the main setting of this work, our chief concern will be *learning* models for random graphs; we will be presented with a sample of random graphs $\mathcal{G}$ drawn from some distribution $\mathbb{G}$, and will be interested in building an approximator for $\mathbb{G}$ by fitting one of the models described above to the data as well as possible. We will approach this task from a machine learning perspective, building our model on a neural network foundation.

The subject of machine learning on graph data has garnered huge amounts of attention in recent years. Graphical data is simply a different sort of data structure than previously studied objects in machine learning, as its matrix representation takes values in $\{0, 1\}$ and the order of the rows and columns are often not fixed. As such, new methods needed to be created to deal with them in order to perform the usual tasks (e.g., classification) that we should expect to be able to do.

There exist a number of other statistical learning tasks that one may be interested in performing on graph data. Of most interest to us will be *generation*, that is, creating graphs belonging to a certain distribution. Leaving aside the process of learning that distribution for the moment (see Section 2.5), the objective in this case is to use a machine learning method, such as a neural network, to transform simple random number generation into graph adjacency matrices. Graphs are fundamentally discrete objects, so we will require some cleverness to create them in this way. Several of the models described above can be deployed for this purpose.

2.3.1 Neural Networks on Graph Data

We now provide a brief overview of machine learning methods on graph data in general, as this work operates on the same principles of such methods, and we will directly make use of some of them during our validation in Chapter 6.

Interest in graph-structured data and activity in the field of machine learning have grown in parallel over the last three decades or so, and these two fields naturally began to intersect. The first published record of neural networks applied to graph data appears to have come in 1997 from Sperduti and Starita [18], who applied Recursive Neural Networks to directed acyclic graphs. These ideas where later extended to graphs containing cycles by Scarselli [19] and Micheli [20].

Machine learning tasks on graph data fall broadly into two categories: Node-level tasks and graph-level tasks. In a node-level task, the data we are presented with generally consists of one massive graph (e.g. the Facebook social network), and we are interested in the value of some function whose input is an individual node. We may wish to classify nodes based on the neighborhood structure or cluster them into communities. If the machine learning is supervised, we usually begin with knowledge of the function value for some subset of the graph's nodes, and wish to extend to all the others. In contrast, graph-level tasks typically involve a dataset consisting of many small or medium graphs (e.g. brain region scans from a sample of individuals). In this case, we attempt to learn something about entire graphs at once; we may wish to classify graphs as belonging to one distribution or another, or even to sample entire graphs from their underlying distribution. There are, of course, other machine learning problems defined on graphs: For example, we may have one fixed graph and data consisting of functions defined on that graph. However, we focus exclusively on graph-level tasks as described above in this work.

A significant development in machine learning on graphs has been the introduction of the Graph Neural Network (GNN) [19]. GNNs are convolutional neural networks, but the convolution operations are defined over sets of node neighbors rather than, for example, adjacent pixels in an image. GNNs have proven quite effective at classification tasks [21], and we make use of them for validation of our model.

Wu et. al. [22] explain that graph convolution operations fall into two categories: spectral methods and spatial methods. In either case, we assume there is a vector z of signal features associated with the nodes in the graph.

In the spectral case, one first transforms the graph's signal to the Fourier domain by first computing the *normalized* graph Laplacian:

$$\mathcal{L} = \mathcal{I} - D^{-1/2} A D^{-1/2};$$

where A is the adjacency matrix and D is the diagonal matrix of vertex degrees, then factoring this Laplacian as

$$\mathcal{L} = USU^T,$$

for S diagonal; and then finally applying the Fourier transform to the graph signal y (which typically consists of features located at each node) as

$$\mathcal{F}(y) = U^T y.$$

Once one arrives in the Fourier domain, one can define convolutional operations on the graph signal by using a filter vector r. Carrying out the transformation of the filter to the Fourier domain, the convolution, and the inverse transform all in one step, we arrive at:

$$\begin{aligned} y * r &= \mathcal{F}^{-1}(\mathcal{F}(y) \odot \mathcal{F}(r)) \\ &= U(U^T y \odot U^T r) \end{aligned}$$

where $\odot$ denotes the entrywise product.

These spectral methods take their inspiration from the field of signal processing. They encode local patterns in the graph using a small number of parameters, but are highly sensitive to changes in the graph data. Examples using this approach are ChebNet [23] and GCN [24]. They are often used in situations where the edge structure of the graph remains the same throughout the problem, but the signal z associated to nodes in the graph changes.

Spatial methods for graph convolutions, in contrast, more closely mirror the types of convolutions often used in image processing tasks [25]. In each layer, they seek to aggregate information around the neighborhood of each node, coarsening the neural network's view of the graph. Given a set of node states $h_v^{(k)}$ to be used as inputs to the k^{th} layer, the output of that layer is

$$h_v^{(k+1)} = w_1^{(k)\,T} x + \sum_{u \in N(v)} w_2^{(k)\,T} h_v^{(k)},$$

where $N(v)$ denotes the set of neighbors of v and w_1, w_2 are learnable parameters. The second term on the right hand side parallels the local filters used in conventional convolutional neural networks.

Spatial methods have the advantage that they can be easily applied to multiple graphs. If two graphs have different Laplacians, the eigenfunctions of those Laplacians are different, and a spectral GNN built on one is incompatible with the other [26]. For this reason, they are more appropriate for tasks where the edge structure of the graph itself is the main interest, rather than some dataset placed on top of a particular structure. It is these that we make use of for validation in later chapters.

2.4 Statistical Inference on Graphs

A great deal of attention has been paid in the last several years to the types of statistical operations that can be carried out on graphical data. Graphs are a unique class of objects, and due to their discrete nature, many of the typical solutions to well-studied statistical problems cannot be readily applied in the graph setting. We therefore require new methods to carry out tasks like hypothesis testing or parameter estimation.

At the heart of any statistical task to be performed on graph data is the representation of the graph(s) of interest. In the realms of graph theory and optimization, graphs are most often represented by their adjacency matrices. However, this representation is unattractive in a statistical setting due to the inherent data symmetry stemming from the existence of graph isomorphisms. A graph who vertices are permuted remains the same graph, but its adjacency matrix changes completely, which would heavily affect any statistical results. Because of this, the first task in any statistical study on graph data is to choose an appropriate representation that is indifferent to these isomorphisms.

2.4.1 Graph Invariants

A *graph invariant* is any characteristic quantity belonging to a graph whose value is unaffected when a graph isomorphism is applied. There are numerous examples, with popularity varying greatly depending on application. The maximum degree of any node in the graph is one such example, as is the width of the minimum spanning tree [27], the chromatic number [28], or the matching number [29].

One widely used graph invariant designed for statistical study is the Adjacency Spectral Embedding (ASE) [30]. If A is the adjacency matrix of a random graph, then its ASE is its normalized eigendecomposition; A is diagonalizable because it is symmetric, and we can write

$$M = USU^T$$

similarly to how the Laplacian was defined above. The set of eigenvalues on the diagonal of S are invariant to permutations of A.

Another graph invariant popularly used is the distribution of node degrees. This metric has been studied widely [31] [32], most commonly in relation to social networks. Many human built social networks, such as the world wide web, have been shown to be scale-free, i.e. their degree distributions follow a power law. We make use of differences in degree distributions to evaluate our model in Chapter 6.

A third graph invariant often used for analysis is the *homomorphism number* with respect to another graph F [33], which is typically well understood or otherwise some type of reference point. A homomorphism from G into F is a mapping from $V(G)$ to $V(F)$ which preserves adjacency. We then define $\hom(G, F)$ to be the number of homomorphisms from G into F.

In Chapter 4, we will introduce *graphlets*, which are a set of graph invariants suited well to the graph generation task. Graphlets are closely related to homomorphism numbers; we will allow the reference graph F to have fewer nodes than G and sample subgraphs of G to compare to the reference.

2.4.2 Statistical Tests

Theoretical results about the distributions of these graph statistics exist when the graphs are assumed to be drawn from a certain model. In some cases, one can express their distributions specifically; for instance, in the case of the maximum vertex degree K in a graph on n vertices drawn from a Stochastic Block Model with block membership probability π_i and edge probability matrix $\mathbf{\Gamma}_{ij}$, we can easily find that

$$\begin{aligned}
K &= \max_i K_i, \;\; i = 1, 2, \ldots n \\
K_i &\sim \text{Bin}(n-1, p) \\
p &= \sum_i \pi_i \sum_j \mathbf{\Gamma}_{ij} \pi_j,
\end{aligned}$$

i.e. K is the maximum of n i.i.d. Binomial random variables.

The ASE is known to be asymptotically normal when the graphs in question are drawn from an RDPG distribution. Without delving into too much detail, it is shown in [9] that, as the number of vertices in an RDPG goes to infinity, the difference between the ASE estimates for the latent positions and the true latent positions follows a multivariate normal distribution.

Armed with the normality of the ASE, it is possible to carry out hypothesis testing with it. We can, for instance, test whether two vertex-matched graphs were drawn from RDPGs built on the same set of latent positions [9]. The test considers a *Procrustes fit* between the ASEs M_1 and M_2 of the two graphs:

$$\min_{Y \in \mathcal{O}} ||M_1 - YM_2||_{Fr}$$

where $\mathcal{O}$ is the set of orthogonal matrices of the appropriate size and $|| \cdot ||_{Fr}$ is the Frobenius norm. This quantity is the test statistic needed for the hypothesis test.

A number of other statistical operations can be performed on graphs by using what is known as the graph Laplacian:

$$\mathcal{L} = D - A$$

where A is the graph's adjacency matrix and D is a diagonal matrix of node degrees, i.e. $D_{ii} = \sum_i A_{ij}$. One often-used algorithm is *spectral clustering*, where the nodes of a large graph are classified into several communities based on mutual edge occurrence [34]. In spectral clustering, we compute the matrix U whose columns are the eigenvalues of the Laplacian, and then perform the usual k-means clustering method on the *rows* of U. The result is a set of clusters corresponding to communities of nodes in the graph.

2.5 Generative Models for Random Graphs

As stated in Section 2.3, our goal in this work is to build a generative model for random graphs using methods from statistical learning. Finding a generative model is, in essence, estimating the distribution of a set of random graphs; we will be given a sample from said distribution and will attempt to learn the distribution from it.

2.5.1 Statistical Models

In the statistics literature, there is a large body of work related to generative models for random graphs. We have already introduced the Stochastic Block Model and the Latent Position Graph; it remains to be discussed how these models are usually learned.

To learn a Stochastic Block Model, the common approach, as explained by Bickel et. al. [35], is a form of maximum likelihood estimation. The maximum likelihood estimators for $\mathbf{\Gamma}$ and for the distribution of $Q\Pi$ are quite straightforward; they are given by

$$\hat{P}(Q = k) = \frac{1}{n}\sum_{i=1}^{n} 1(Q_i = k)$$

and

$$\hat{\mathbf{\Gamma}}_{kl} = \frac{1}{|\mathcal{T}_{kl}|}\sum_{(i,j)\in\mathcal{T}_{kl}} 1((i,j) \in E),$$

where

$$\mathcal{T}_{kl} = \{(i,j) \mid Q_i = k, Q_j = l\}.$$

However, the community variables Q are most commonly latent - they are not observed or even observable. In that case, the optimization of these parameters is not simple, but success has been found using the EM algorithm [36]. Bickel et. al. show that these estimators are asymptotically normal and prove concentration inequalities. The result of this method is a recovery of the edge probabilities and block membership probabilities that parameterize the distribution.

Of course, there exists a certain amount of nonidentifiability inherent in the Stochastic Block Model. The community blocks can be relabeled, which is one source of nonidentifiability, and the nodes within each community can be relabeled, which is another. If any of the rows of $\mathbf{\Gamma}$ are identical, this is a third source.

In the case of Latent Position Graphs, Tang, Sussman, and Priebe demonstrate that the unknown latent positions can be accurately estimated by way of the Adjacency Spectral Embedding [30]. Once the ASE $A = USU^T$ is calculated, we can relate the matrix of latent positions of the nodes in the graph to the quantity $US^{\frac{1}{2}}$.

As long as the kernel function is assumed known, the latent positions fully parameterize a Latent Position Graph. Estimating the latent positions is therefore sufficient for learning the model. However, the above strategy generally applies to the setting where we observe one massive graph, rather than a sample of small graphs from one distribution. It also assumes the number of nodes in the graph is fixed, which is an assumption we will not generally make. We therefore will look for another strategy for learning latent position models.

2.5.2 Machine Learning Methods

Seeing the difficulties associated with learning generative models along the avenues described above, we seek instead to leverage the power of modern machine learning. Our method in particular will try to model the observed graphs as Latent Position Graphs and attempt to estimate the embeddings using Stochastic Gradient Descent; this is not, however, the only viable approach.

We are only aware of a few methods that directly compete with our own. We will give the most attention to GraphRNN [37]. GraphRNN models each graph it creates as a sequence of binary vectors $Y_1 \ldots Y_n$, with the length of vector Y_i equal to $i - 1$. The i^{th} binary vector shows the existence of edges between node i and all preceding nodes. The vectors are generated in sequence, using a recurrent neural network to model a conditional probability distribution. In this way, graphs are built up iteratively; at each step, one or more nodes are added, and connections are added from the new node to any number of existing nodes. The sequential connection probabilities are modeled with a Bernouilli distribution conditional on previous connections, and these probabilities are learned through direct sampling of the dataset. The result is graphs that are built up node-by-node to match a distribution on any number of nodes.

As we will see in the next chapter, our method accomplishes something similar to GraphRNN, but there are key differences in approach. We learn the graph all at once, rather than building it up one node at a time. We also choose a set of entirely different statistics to train. We are not the first to introduce these statistics, but we appear to be the first to utilize them in machine learning training in this way. Our model defeats GraphRNN on a number of metrics, and also is more robust thanks to key extensions addressing potential weaknesses.

There are a few other methods that accomplish similar things to our model and GraphRNN. BiGG [38] is one such method which is specifically designed to address the sparse case; the authors assume the number of edges in the graph is much smaller than the number of nodes. Rather than simulate the full adjacency matrix, the method uses a tree-structured autoregressive model for generating the set of edges associated with each node. They are able to reduce a process which is usually $\mathcal{O}(n^2)$ to one which is $\mathcal{O}(n \log n)$.

GraphVAE [39] and Graphite [40] both make use of Variational Auto-Encoders (VAEs) to generate random graphs. VAEs work by first taking training datapoints and finding low-dimensional representations for them, and then finding a 'decoder' capable of restoring the low-dimensional representations back to something close to the original data. The decoder is then employed on its own to generate new data. GraphVAE's decoder models both nodes and edges as Bernouilli random variables, and includes attributes in the baseline model (see Chapter 5). Graphite, in contrast, uses an iterative approach, generating a sequence of adjacency matrices and performing message passing each time using a Graph Neural Network.

A few other approaches have been published, and many of those are detailed by Hamilton [41]. These include autoregressive methods similar to GraphRNN, such as Graph Recurrent Attention Networks [42], as well as methods that use Generative Adversarial Networks [43, 44].

References

1. Gilbert, E. N. Random plane networks. *Journal of the society for industrial and applied mathematics* **9,** 533–543 (1961).
2. Erdös, P. & Rényi, A. On Random Graphs I. *Publicationes Mathematicae Debrecen* **6,** 290 (1959).
3. Erdos, P. & Renyi, A. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.* **5,** 17–61 (1960).
4. May, W. D. & Wierman, J. C. Using symmetry to improve percolation threshold bounds. *Combinatorics, Probability and Computing* **14,** 549–566 (2005).
5. Parviainen, R. Estimation of bond percolation thresholds on the Archimedean lattices. *arXiv preprint arXiv:0704.2098* (2007).
6. Holland, P. W., Laskey, K. B. & Leinhardt, S. Stochastic blockmodels: First steps. *Social Networks* **5,** 109–137 (1983).
7. Peixoto, T. P. Entropy of stochastic blockmodel ensembles. *Physical Review E* **85,** 056122 (2012).
8. Paul, S. & Chen, Y. A random effects stochastic block model for joint community detection in multiple networks with applications to neuroimaging. *The Annals of Applied Statistics* **14,** 993–1029 (2020).
9. Athreya, A. *et al.* Statistical Inference on Random Dot Product Graphs: a Survey. *Journal of Machine Learning Research* **18,** 1–92 (2018).
10. Fiduccia, C. M., Scheinerman, E. R., Trenk, A. & Zito, J. S. Dot product representations of graphs. *Discrete Mathematics* **181,** 113–138 (1998).
11. Beer, E., Fill, J. A., Janson, S. & Scheinerman, E. R. *On vertex, edge, and vertex-edge random graphs* 2008.
12. Scheinerman, E. R. & Tucker, K. Modeling graphs using dot product representations. *Computational Statistics* **25,** 1–16 (Mar. 2010).
13. Agterberg, J., Tang, M. & Priebe, C. E. *On Two Distinct Sources of Nonidentifiability in Latent Position Random Graph Models* 2020.
14. Frank, O. & Strauss, D. Markov Graphs. *Journal of the American Statistical Association* **81,** 832–842 (1986).

15. Robins, G., Pattison, P., Kalish, Y. & Lusher, D. An introduction to exponential random graph (p*) models for social networks. *Social Networks* **29.** Special Section: Advances in Exponential Random Graph (p*) Models, 173–191 (2007).

16. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2,** 359–366 (1989).

17. Scarselli, F. & Tsoi, A. C. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks* **11,** 15–37 (1998).

18. Sperduti, A. & Starita, A. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* **8,** 714–735 (1997).

19. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* **20,** 61–80 (2009).

20. Micheli, A. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* **20.** Cited by: 222, 498–511 (2009).

21. Zhou, J. *et al. Graph Neural Networks: A Review of Methods and Applications* 2019. arXiv: 1812.08434 [cs.LG].

22. Wu, Z. *et al.* A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

23. Defferrard, M., Bresson, X. & Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *CoRR.* arXiv: 1606.09375 (2016).

24. Kipf, T. N. & Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks* 2016.

25. Zhang, M., Cui, Z., Neumann, M. & Chen, Y. *An end-to-end deep learning architecture for graph classification* in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

26. Zhang, S., Tong, H., Xu, J. & Maciejewski, R. Graph convolutional networks: a comprehensive review. *Computational Social Networks* **6** (Nov. 2019).

27. Graham, R. & Hell, P. On the History of the Minimum Spanning Tree Problem. *Annals of the History of Computing* **7,** 43–57 (1985).

28. Bollobás, B. The chromatic number of random graphs. *Combinatorica* **8,** 49–55 (Mar. 1988).

29. Livi, L. & Rizzi, A. The graph matching problem. *Pattern Analysis and Applications* **16,** 253–283 (2013).

30. Sussman, D. L., Tang, M., Fishkind, D. E. & Priebe, C. E. A Consistent Adjacency Spectral Embedding for Stochastic Blockmodel Graphs. *Journal of the American Statistical Association* **107,** 1119–1128 (2012).

31. Dorogovtsev, S., Mendes, J. F. & Samukhin, A. Size-dependent degree distribution of a scale-free growing network. *Physical review. E, Statistical, nonlinear, and soft matter physics* **63,** 062101 (July 2001).

32. Holme, P. Rare and everywhere: Perspectives on scale-free networks. *Nature Communications* **10,** 1016 (Mar. 2019).

33. Lovász, L. *Large networks and graph limits* (American Mathematical Soc., 2012).

34. Micali, S. & Vazirani, V. V. *An O (v| v| c| E|) algoithm for finding maximum matching in general graphs* in *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)* (1980), 17–27.

35. Bickel, P., Choi, D., Chang, X. & Zhang, H. Asymptotic normality of maximum likelihood and its variational approximation for stochastic blockmodels. *The Annals of Statistics* **41,** 1922–1943 (2013).

36. Daudin, J.-J., Picard, F. & Robin, S. A mixture model for random graphs. *Statistics and Computing* **18,** 173–183 (June 2008).

37. You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models* en. in *International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, July 2018), 5708–5717.

38. Dai, H., Nazi, A., Li, Y., Dai, B. & Schuurmans, D. *Scalable Deep Generative Modeling for Sparse Graphs* in *Proceedings of the 37th International Conference on Machine Learning* (eds III, H. D. & Singh, A.) **119** (PMLR, 2020), 2302–2312.

39. Simonovsky, M. & Komodakis, N. *GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders* 2018.

40. Grover, A., Zweig, A. & Ermon, S. *Graphite: Iterative Generative Modeling of Graphs* in *Proceedings of the 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) **97** (PMLR, June 2019), 2434–2444.

41. Hamilton, W. L. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **14,** 1–159.

42. Liao, R. *et al. Efficient Graph Generation with Graph Recurrent Attention Networks* 2019.

43. De Cao, N. & Kipf, T. MolGAN: An implicit generative model for small molecular graphs (2018).

44. Bojchevski, A., Shchur, O., Zügner, D. & Günnemann, S. NetGAN: Generating Graphs via Random Walks (2018).

This page intentionally left blank.

Chapter 3

The Visual Turing Test: A Motivating Problem

In this chapter, we introduce a motivating problem for the main body of work. Here, we present the Visual Turing Test [1], a statistical test for the performance of a computer vision system.

3.1 The Visual Turing Test

Suppose we have some black-box computer vision system that claims to be able to understand a certain class of images; that is, it accepts image-structured data as input and can produce meaningful information about those images as output, in some form or another. The discerning mathematician may be interested in testing the validity of this claim: Does the computer vision system actually do what its software engineer says it does?

A sensible way to test this might be to ask the computer vision system yes-or-no questions about the image, e.g. "Is there a person in this quadrant of the picture?". We will choose not to worry about whether or not the system can actually *understand* the question in the context of this problem; we will suppose that either it is equipped with some sort of previously-verified natural language processing engine, or perhaps that all of the questions in our question space come from a certain hard-coded vocabulary.

Regardless, if the system consistently gets our questions correct, we can say that it understands the image...

...Assuming, of course, that the questions we ask it are not easily guessable. For instance, if we were to ask it something akin to “Is there a cat driving this car?” or “Is there a cat halfway up this skyscraper?”, it might be able to ‘cheat’, that is, answer the question without looking at the image at all. (In both of these cases, the answer is *probably* no!)

So what constitutes a good question? Improbable locations of cats aside, we might demand that any question we ask has a probability near 0.5 of having correct answer ‘Yes’ across the entire distribution of images in the class we are considering. If we were to look at images of cars taken from Google Images, perhaps a fair question might be something like “Is this car black / gray?”. We will need some knowledge of the distribution of images in order to come up with fair questions.

However, it is not enough for each question we pose to individually be fair or balanced. Since we are asking questions in sequence, it could be possible for the system to use the answers to previous queries to predict the answer to the next. In fact, we must require that *each question we ask is balanced, conditional on the answers to all previous questions.* This, of course, will make our task of locating fair questions far more difficult.

We arrive, then, at the Visual Turing Test: Ask a computer vision system a sequence of binary questions about an image, such that each question is balanced given the entire history of questions. Our main task in this section, then, will be to find a way to discover 50/50 questions.

3.2 Scene Graphs

It is naturally very difficult to talk about the distribution of images belonging to some class. Even the weaker cellphone cameras available nowadays are 10 megapixels or more; the images they capture are therefore 10-million-dimensional vectors. However, it is quite clear that the meaningful parts of images are in fact low-dimensional objects in a high-dimensional space; while the image may consist of 10^7 pixels, they are not completely random or independent. Most images consist of only a few objects of interest.

We introduce the *scene graph* as a low-dimensional representation of an image. In a scene graph representation of an image, each important object is represented by a node, and edges are drawn between the objects if they are interacting in some way. Scene graphs are typically both vertex- and edge-attributed, with the node attributes representing the type of object in question and the edge attributes codifying the nature of their relationships.

For the purposes of this work, the class of images we will work with will be street views, i.e. pictures of city streets and sidewalks. In our case, we will use a node to represent each person or vehicle in the scene (with a corresponding attribute for each), and some possible edges may include talking to each other (in the case of two persons), operating (in the case of a person and a vehicle), or following behind (in the case of two vehicles). We will also include additional vertex attributes to describe what the objects are doing, like sitting down or being in a parking space.

This representation makes it much easier to talk about the distribution of images belonging to a certain class (in this case, street views). We now only need to think about a probability distribution over attributed graphs, for which there are some well-known models we can attempt to use.

3.3 Using ERGMs

We will discuss the use of the Exponential Random Graph Model (ERGM) to understand the distribution of street view scene graphs. As discussed in the previous chapter, ERGMs require a set of carefully chosen feature statistics in order to assign probabilities to possible graphs. In some settings, the form of these features may be known in advance, if the statistician has some knowledge of how the graphs in the dataset were generated. However, in the case of the street view scene graphs, the only thing we know about the graphs is that they were generated by taking a photo of a sidewalk, which does not provide any obvious information about how to structure the ERGM.

The original paper on scene graph generation for the VTT [2] makes use of a number of statistics related to the number and type of objects found in various parts of an image; for example, one statistic used is the number of vertices with the label 'person' found in the entire image. The authors use 21 statistics in all, chosen based on qualitatively common characteristics present in a training set of images. The values of these statistics were then calculated from said training set.

However, we must keep aware of the main criticism of this approach: We have no idea whether these features are enough to capture the main characteristics of the underlying distribution of images. Learning the values of these features does not equate to learning the distribution, unless it can be verified that the distribution can actually be parameterized by the features. In the very likely scenario that this is not the case, learning an ERGM is an art, and a difficult one at that.

3.4 Test Procedure

Equipped with our ERGM learned from training data, we can implement the test procedure. First, we prepare the computer vision system and the image about which it will be tasked with answering questions. We will assume that we possess an oracle that is capable of accurately answering any question about this image; in most cases, any human would suffice.

We then generate a very large sample of scene graphs from the ERGM - if we plan to ask m binary questions, we will need at least 2^{m+1} graphs. From here, we iterate through a prespecified vocabulary of possible questions. For each candidate question, we (quickly) answer it for each graph in our sample, which effectively splits the sample into two. We look for a question which approximately divides the sample into two equal pieces.

When we find one whose division is acceptable within a certain tolerance, we can stop looking, and immediately pose that question to the computer vision system about the image of interest. We compare its answer to that of the oracle. (Hopefully, they match!) In either case, we now drop the half of the sample that does not match the answer given by the oracle. In this way, we are left with a subsample consistent with the question history. We then repeat the procedure using only this subsample. If we start with 2^{m+1} graphs and each question is balanced, we should have enough graphs to continue through m questions. (However, the sample size shrinks with each question, so the later questions are less certain to be fair.)

When all is done, we will have asked m conditionally fair questions about the image to the computer vision system. By repeating this procedure for many images and recording the system's accuracy, we can get a measurement of its performance.

3.5 Conclusions

The Visual Turing Test was implemented in [2] by Hallonquist et. al., applied to the aforementioned street view images. The querying system was designed to select questions in stages, beginning with questions that would instantiate objects in the image as part of the scene graph (i.e. vertex discovery), and then moving on to questions about the attributes of those objects (attribute discovery), and then finally about the relationships between the objects (edge discovery).

To build its ERGMs, the study made use of a number of graph invariants as statistical features. Some examples of the features of which it made use include the number of objects in the image, the overlap between objects, the height of objects within the image, the distance between objects, and the number of relationships (edges) present.

Hallonquist et. al. reported some difficulty in the task of generating unbiased questions in each of the three querying stages. In an attempt to avoid the brute-force approach of generating 2^{m+1} graphs, they endeavored to sample from the conditional distribution of graphs given the question history by way of the Metropolis-Hastings algorithm. Questions asked to instantiate took the form of "Is there an object (from the vocabulary) in this section of the image?". 37.5% of the instantiation questions asked in that study were answered in the affirmative, indicating a difficulty in locating questions that would be answered positively and negatively with equal probability.

Much of the difficulty can be attributed to the ERGM and the somewhat artistic way that it must be constructed. The graph invariants used as features when building an ERGM are not selected according to any strategy with foundation in theory; rather, the statistician must manually try to identify key characteristics of the graph distribution and capture them accordingly. This is naturally difficult, which may preclude the use of the model entirely.

Seeing the difficulty in building ERGMs, we are led naturally to the central question of this work: How can one learn a general distribution for random graphs from a sample drawn from that distribution?

References

1. Geman, D., Geman, S., Hallonquist, N. & Younes, L. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences* **112,** 3618–3623 (2015).
2. Hallonquist, N., German, D. & Younes, L. *Graph Discovery for Visual Test Generation* in *2020 25th International Conference on Pattern Recognition (ICPR)* ISSN: 1051-4651 (Jan. 2021), 7500–7507.
3. Malinowski, M. & Fritz, M. *Towards a Visual Turing Challenge* 2014.
4. Qi, H., Wu, T., Lee, M.-W. & Zhu, S.-C. *A Restricted Visual Turing Test for Deep Scene and Event Understanding* 2015.

This page intentionally left blank.

Chapter 4

A Generative Model for Latent Position Graphs

In this chapter, we introduce the model that constitutes the main contribution of this work. We present a neural network-based approach to learning general random graph distributions, trained using a moment estimator for subgraph count sufficient statistics called graphlets.

4.1 Introduction and Motivation

In Chapter 3, we introduced the Visual Turing Test, whose use required the ability to learn a general model for random graph distributions. Beyond that single need, there are numerous problems that require the use of a generative model for random graphs. For instance, in many applications of data science methods in medicine, computational vision, chemistry, and related fields, availability of data is often a limiting factor. In the case of graphical data, it is not always possible to obtain datasets containing large numbers of instances (graphs) from a specific class of interest. Brain connectivity data (produced, e.g., by functional MRI), for example, is limited by cost and subject availability when associated with diseases, and generally restricted to a few dozen subjects. Chemical connectivity data [1–3] is limited by the number of compounds and therefore naturally scarce. Many modern machine learning methods require more data than may be readily available, so it is desirable to create generative models for the distributions underlying such data. Such models can indeed be used for statistical inference, method evaluation, and validation.

A large variety of random graph models are used in the literature, but most of those that are invariant by isomorphism belong to the family of latent position graphs introduced in Chapter 2. Our model, presented in Section 4.2 also belongs to this category, which presents the advantage that, once the kernel function ϕ is fixed, the modeling effort is reduced to the distribution of the node variables.

Neural approaches for generative models have grown in popularity recently in part due to the successes of generative adversarial networks (GANs) [4, 5] and variational auto-encoders (VAEs) [6]. Such generative models are capable of producing new data (such as fake images of faces of nonexistent humans [7]), cleaning up noisy data, and simulating unobserved phenomena in the sciences. However, only a few attempts have been made at using the GAN paradigm for graph data [8, 9]. Other approaches include the previously discussed use of ERGMs to implement maximum likelihood estimation of scene graphs [10], as well as [11], which is based on an autoregressive modeling of vectorized connectivity data.

As generative neural models have demonstrated their ability at modeling complex large dimensional random processes, they provide a natural resource for the modeling of the node process ζ in latent position graphs and will provide the basis of our model. We will not use, however, the GAN or VAE learning paradigms, but rather implement a more classical parametric estimation approach, using moment estimators. These moments will be based on "graphlets", or subgraph counts [12, 13], which are statistics that evaluate the number of induced subgraphs of small size in a given isomorphism class that are present in the random graph. These graphlets can be seen as an analog of polynomial moments for collections of binary random variables, and act as sufficient statistics in the characterization of the distribution of relabeling invariant distributions on graphs of infinite size [14, 15]. Because these counts can be represented as sums of binary variables, they are also compatible with stochastic gradient descent strategies in the learning algorithm, as will be seen in Section 4.3.

We will introduce, in the rest of the chapter, a novel neural-network-based approach to creating generative models for random graphs (Section 4.2) and associated training algorithm (Section 4.3). We use small amounts of data to learn kernel-based models for random graph distributions, which are then able to generate unlimited amounts of artificial data. These generative models will then be evaluated in Chapter 6 by comparing, in particular, real to artificial data.

4.2 Notation and Model

4.2.1 Graphs and Basic Terminology

As before, a graph is a pair $G = (V, E)$, where V is a set of elements called *vertices* and E is a set of pairs of vertices. If V is ordered, say $V = (v_1, \ldots, v_n)$, the adjacency matrix of G, denoted A_G is the binary matrix with entry (i, j) equal to 1 if $(v_i, v_j) \in E$ and to 0 otherwise. If V is a set of integers, we will always assume that it is listed in increasing order.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are said to be isomorphic (we will write $G \sim G'$) if there exists a bijection $f : V \to V'$ such that $\tilde{f} : (v_1, v_2) \mapsto (f(v_1), f(v_2))$ is also a bijection from E to E'.

A graph (V', E') is a sub-graph of (V, E) if $V' \subset V$ and $E' \subset E$. It is an induced subgraph if $E' = \{(i, j) \in E \mid i, j \in V'\}$. We will denote by $G_{V'}$ the subgraph of G induced by V'.

Finally, if V' is a set, we will make the abuse of notation $V' \subset G$ if $G = (V, E)$ and $V' \subset V$.

In the following, we will use $[n]$ to denote the set $\{1, \ldots, n\}$ and $[n]_p$ the set of subsets of $[n]$ with p elements. Letting $\mathbb{K}_n$ denote the complete graph with vertex set $[n]$, we will let $\mathcal{K}_n$ denote the family of all subgraphs of $\mathbb{K}_n$. Denoting by $\mathcal{S}_n$ the set of all $n!$ permutations of $[n]$, we associate to $s \in \mathcal{S}_n$ and $G = (V, E) \in \mathcal{K}_n$ the relabelled graph $s \cdot G$ with vertices $\{s(i) : i \in V\}$ and edges $\{(s(i), s(j)) : (i, j) \in E\}$, which provides a group action of $\mathcal{S}_n$ on $\mathcal{K}_n$.

4.2.2 Graph Model

4.2.2.1 Kernels

In this chapter, we will develop models and training algorithms for latent position graphs. To describe our model in full generality, we let $\mathcal{Z} = [0, +\infty)^d$ denote the d-dimensional positive Euclidean hyperquadrant and we consider a symmetric function $\phi : \mathcal{Z} \times \mathcal{Z} \to [0, 1]$ that we will call a kernel. Examples of such kernels include

$$\phi(z_1, z_2) = \frac{z_1^T z_2}{|z_1|\,|z_2|}$$

where $|z|$ denotes the Euclidean norm of z,

$$\phi(z_1, z_2) = \exp(-|z_1 - z_2|^2),$$

the Gaussian kernel, or

$$\phi(z_1, z_2) = \frac{(1 + z_1^T z_2)^c}{\sqrt{(1 + z_1^T z_1)^c (1 + z_2^T z_2)^c}}$$

where c is an integer (the normalized polynomial kernel). Note that these examples correspond to reproducing kernels [16], and can therefore be interpreted as inner products in possibly infinite-dimensional reproducing kernel Hilbert spaces. However, the reproducing property is not a required condition. For example, we obtain good numerical results when modeling sparse graphs and using

$$\phi(z_1, z_2) = 1 - \frac{z_1^T z_2}{|z_1|\,|z_2|}\,.$$

4.2.2.2 A family of kernel graphs

Fixing a kernel ϕ, we will model graphs with random, but bounded, size, where the upper bound on the size is given by a fixed integer n. More precisely, our model will be supported by the set (denoted $\mathcal{K}_n$ above) of subgraphs of the complete graph $\mathbb{K}_n$. We will model below a collection of random variables $\boldsymbol{Y} = (\boldsymbol{Q}, \boldsymbol{Z}) = ((Q_1, \ldots, Q_n), (Z_1, \ldots, , Z_n))$ taking values in $[0,1]^n \times \mathcal{Z}^n$. Conditionally to a realization $\boldsymbol{y} = (\boldsymbol{q}, \boldsymbol{z})$, we define random variables $\boldsymbol{B}, \boldsymbol{R}$, where

1. $\boldsymbol{B} = (B_1, \ldots, B_n)$ is a collection of independent Bernoulli variables with respective parameters $q_1, \ldots, q_n$.
2. $\boldsymbol{R} = (R_{ij}, 1 \leq i < j \leq n)$ is a collection of independent Bernoulli random variables with $P(R_{ij} = 1) = \phi(z_i, z_j)$.
3. $\boldsymbol{B}, \boldsymbol{R}$ are mutually independent.

We then define, for given realizations of $\boldsymbol{B}$ and $\boldsymbol{R}$, the graph $G = \mathcal{G}(\boldsymbol{b}, \boldsymbol{r}) \in \mathcal{K}_n$ with vertex set $\{i : b_i = 1\}$ and edge set $\{(i,j) : r_{ij} = 1, b_i = b_j = 1\}$.

Our random graph model is then

$$\mathbb{G} = S \cdot \mathcal{G}(\boldsymbol{B}, \boldsymbol{R})$$

where S follows a uniform distribution on $\mathcal{S}_n$. Equivalently, $\mathbb{G}$ is built by first forming a graph with vertex set $[n]$ and inserting an edge (i,j) with probability $\phi(z_i, z_j)$, then selecting an induced subgraph by retaining each vertex with probability q_i before randomly labelling the nodes. As required, this distribution is invariant by isomorphism, i.e., it gives the same probability to two isomorphic subgraphs of $\mathbb{K}_n$.

Note that this graph construction differs from the one that is commonly used for RDPGs [17, 18], which does not include a node selection mechanism, for which the graph size is generally large compared to n, and $Z_1, \ldots, Z_n$ are associated with "communities" that each node selects at random before forming edges. Our model and algorithms can however be easily modified to represent community graphs, and we will return to this in Section 4.3.5.2.

4.2.2.3 Latent variable model

The latent variable $\boldsymbol{Y}$ is modeled as a function $\boldsymbol{Y} = \Psi(\Omega, \theta)$ where Ω is a random variable with known distribution (e.g., multivariate standard Gaussian) and θ is a vector of parameters. In our implementation Ψ is specified by a neural net architecture of which Ω is the input layer and θ the vector of weights. However, a neural net implementation is by no means a necessity, and the further developments only require that Ψ and its derivative in θ are reasonably easy to compute, for the learning algorithm to be feasible.

Our model is therefore fully specified by the kernel function, ϕ, the maximal number of nodes, n, the "architecture" Ψ, and the parameter θ. In the following, we assume that the first three are fixed and selected once for all, and focus on the estimation of θ.

Returning to the notation above, we will denote by P_θ (with expectation E_θ) the distribution of the random graph before relabelling, i.e., that of $\mathcal{G}(\boldsymbol{B}, \boldsymbol{R})$. The distribution of $S \cdot \mathcal{G}(\boldsymbol{B}, \boldsymbol{R})$ is denoted P^*_θ, with expectation E^*_θ, so that, for $G \in \mathcal{K}_n$,

$$P^*_\theta(G) = \frac{1}{n!} \sum_{s \in \mathcal{S}_n} P_\theta(s \cdot G).$$

4.3 Learning Algorithm

4.3.1 General Setting

We use a moment estimation approach, solving the equation

$$E^*_\theta(H) = \bar{H},$$

where $H : \mathcal{K}_n \to \mathbb{R}^m$ is a vector of graph statistics, and $\bar{H}$ represents the empirical value of these statistics in the training data.

Since we want to learn a distribution that is invariant by graph isomorphism, H should also have this property, i.e., $H(G) = H(G')$ if G and G' are isomorphic. Note that, if H is relabelling invariant, one has

$$\begin{aligned}
E^*_\theta(H) &= \sum_{G \in \mathcal{K}_n} H(G) P^*_\theta(G) \\
&= \frac{1}{n!} \sum_{s \in \mathcal{S}_n} \sum_{G \in \mathcal{K}_n} H(G) P_\theta(s \cdot G) \\
&= \frac{1}{n!} \sum_{s \in \mathcal{S}_n} \sum_{G \in \mathcal{K}_n} H(s^{-1} \cdot G) P_\theta(G) \\
&= \sum_{G \in \mathcal{K}_n} H(G) P_\theta(G) \\
&= E_\theta(H)\,.
\end{aligned}$$

so that we can solve the simpler equation

$$E_\theta(H) = \bar{H},$$

which will remove the relabelling step from our consideration.

4.3.2 Graphlets

Our function H will be based on counts of certain subgraphs in the graphs, which are called *graphlets*. More precisely, let F be a graph with p vertices, and G a subgraph of $\mathbb{K}_n$. Denote by $H_F(G)$ the probability that a set $A \in [n]_p$, chosen uniformly at random, is included in the vertex set of G and induces a subgraph that is isomorphic to F. For a graph G with vertex set $V \subset [n]$, we have

$$\begin{aligned}
H_F(G) &= \binom{n}{p}^{-1} \sum_{W \in [n]_p} \mathbf{1}_{W \subset V} \mathbf{1}_{G_W \sim F} \\
&= \binom{n}{p}^{-1} \sum_{W \subset V, |W| = p} \mathbf{1}_{G_W \sim F},
\end{aligned}$$

so that $H_F(G)$ is the number of induced subgraphs of G that are isomorphic to F normalized by $\binom{n}{p}$. This awkward probabilistic definition will make sense when we will discuss stochastic gradient descent.

We select a finite set $\mathfrak{F}$ of graphs and define $H(G) = (H_F(G), F \in \mathfrak{F})$. Letting $\mathcal{G}_p$ denote the set of isomorphism classes of graphs with p nodes, we use, in our experiments, $\mathfrak{F} = \mathcal{G}_1 \cup \mathcal{G}_p$ for some integer p. ($\mathcal{G}_1$ is the trivial class of graphs with one node and the corresponding H_F associated to G its number of nodes divided by n.)

Equivalent classes of graphs are shown in Figure 4-1 for $p = 3$ and 4. There are 34 isomorphism classes with 5 nodes, 156 with 6, more than 1,000 with 7 and more than 12,000 with 8 (note that, in this work, we do not require F to be connected in order to constitute a graphlet). These isomorphism classes were identified using a brute-force algorithm which matched each possible graph on p vertices to a unique class.

Subgraph counts are polynomials in the binary edge variables of the random graph corrected for isomorphism invariance and therefore are generalizations of polynomial moments to random graphs. It is therefore not surprising that they constitute key elements in the asymptotic study of isomorphism-invariant graph distributions, as illustrated in [14, 15], for which they essentially provide sufficient statistics. Even though we are interested in finite, not necessarily large, graph models in this work, using subgraph counts as the basis of our moment estimator remains natural, and they have proven to be powerful also for the discrimination between different classes of graphs, in a wide range of applicative contexts [19–21].

4.3.2.1 Objective function

In the next section, we design a stochastic gradient descent (SGD) algorithm for the minimization of

$$\lambda(\theta) = (E_\theta(H) - \bar{H})^T \Xi (E_\theta(H) - \bar{H}) \tag{4.1}$$

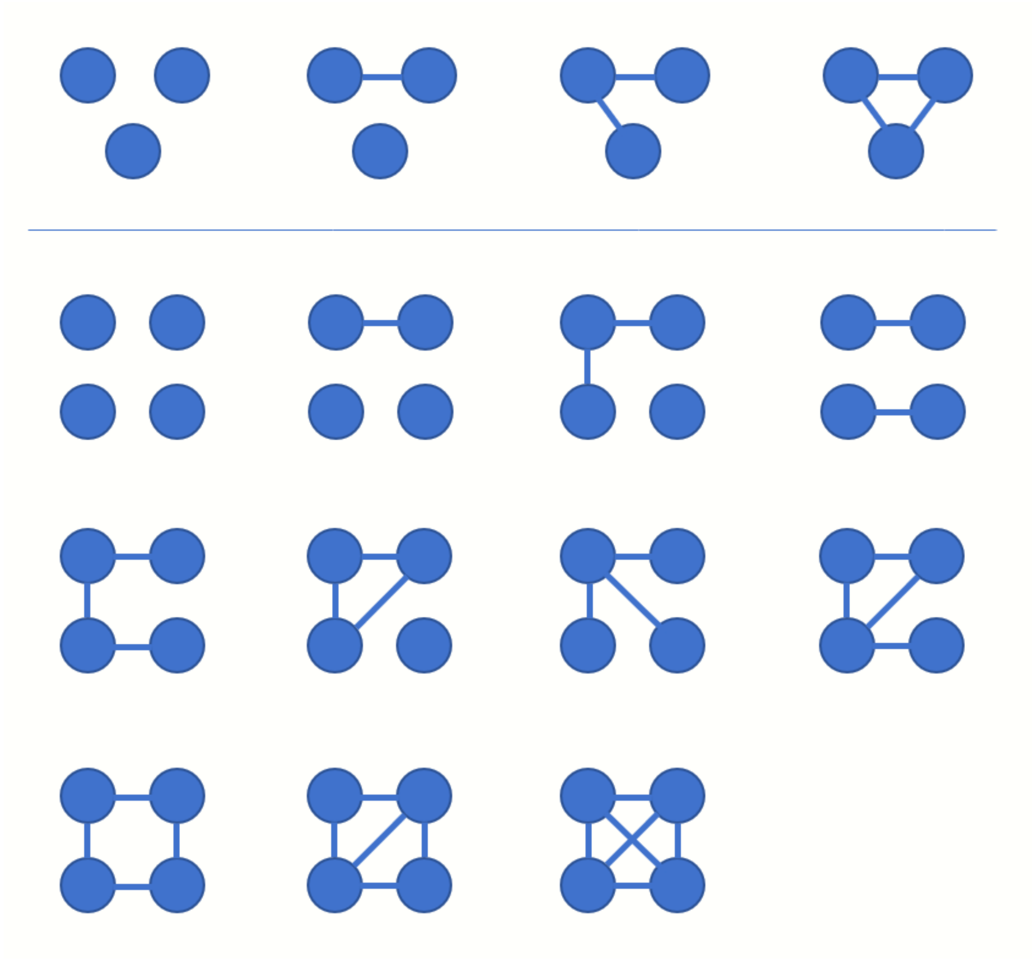


Figure 4-1. On three nodes, there are four graph isomorphism classes, and on four nodes, there are eleven.

where Ξ is a fixed diagonal matrix of weights, with positive coefficients, the simplest choice being the identity matrix. We will write $\mathrm{diag}(\Xi) = (\Xi_F, F \in \mathcal{F})$.

The first part of the training procedure therefore consists in estimating the target expectation $\bar{H}$ from training data (constituted, say, by graphs $G_1, \ldots, G_N$). It is in principle simply given by the empirical averages

$$\bar{H}_F = \frac{1}{N} \sum_{k=1}^{N} H_F(G_k).$$

However, the computation of $H_F(G)$ has polynomial complexity in the size of G, with a power equal to the size of F. The resulting cost can therefore be prohibitive for large graphs/graphlets, and one must replace exact counts with approximations in the determination of $\bar{H}_F$.

A notable body of work has been dedicated to the derivation of efficient algorithms for the approximation of subgraph counts (see, e.g., [22] for a recent review), many of them optimized for the search of one specific graphlet rather than all graphlets of a given size. Since we here need all graphlets of size p, we have used the simple strategy of randomly sampling a sufficient number of subsets of nodes of cardinality p, identifying for each of them its isomorphism class to estimate frequencies. More explicitly, if J subsets are sampled from G_k, and J_F among them are found to be isomorphic with F, we estimate

$$\hat{H}_F(G_k) = \frac{\binom{n_k}{p}}{\binom{n}{p}} \frac{J_F}{J}$$

to approximate $H_F(G_k)$, where $n_k \leq n$ is the number of nodes in G_k.

4.3.3 SGD Formulation

We recall that our model defines a vector $\boldsymbol{Y}$ of latent variables generated as a function $\boldsymbol{Y} = \Psi(\theta, \Omega)$ where Ω is a random variable with known distribution, θ is a parameter, Ψ is assumed to be differentiable in θ and $\boldsymbol{Y}$ takes the form

$$\boldsymbol{Y} = ((Q_1, \ldots, Q_n), (Z_1, \ldots, Z_n)).$$

Conditionally to $\boldsymbol{Y}$, we defined a node selection process $\boldsymbol{B}$ as a vector of independent Bernoulli variables with respective parameters $Q_1, \ldots, Q_n$ and an edge selection process $\boldsymbol{R}$ as a triangular array of Bernoulli variables with parameters $\phi(Z_i, Z_j)$ yielding a graph $G = \mathcal{G}(\boldsymbol{B}, \boldsymbol{R})$. In the following sequence of steps, we progressively express the objective function U as the expectation, over an increasing number of random variables, of an increasingly simple function.

(i) If we introduce two independent copies of Ω, say, $\Omega, \tilde{\Omega}$, we can write

$$
\begin{aligned}
\lambda(\theta) &= (E_\theta(H) - \bar{H})^T \Xi (E_\theta(H) - \bar{H}) \\
&= E(E_\theta(H - \bar{H} \mid \Omega))^T \Xi E(E_\theta(H - \bar{H} \mid \Omega)) \\
&= E(E_\theta(H - \bar{H} \mid \tilde{\Omega})^T \Xi (E_\theta(H - \bar{H} \mid \Omega))
\end{aligned}
$$

where the outer expectation is with respect to the distribution of $\Omega, \tilde{\Omega}$.

(ii) Moreover, we have

$$
H_F(G) = P_{n,p}(\mathbb{W} \subset G \text{ and } G_{\mathbb{W}} \sim F)
$$

where $P_{n,p}$ denotes the uniform distribution over random subsets, $\mathbb{W}$, of $[n]$ with cardinality p, so that, introducing the set $\tilde{\mathcal{J}}_{W,F}$ of graphs $G \in \mathcal{K}_n$ such that $W \subset G$ and $G \sim F$, we have

$$
E_\theta(H \mid \Omega = \omega) = E_{n,p}(P_\theta(\tilde{\mathcal{J}}_{\mathbb{W},F} \mid \Omega = \omega)).
$$

As a consequence, we can write

$$
\lambda(\theta) = E((\zeta(\theta, \tilde{\Omega}, \tilde{\mathbb{W}}) - \bar{H})^T \Xi (\zeta(\theta, \Omega, \mathbb{W}) - \bar{H}))
$$

where E now represents an expectation with respect to $\Omega, \tilde{\Omega}, \mathbb{W}, \tilde{\mathbb{W}}$ and $\zeta(\theta, \omega, W)$ is the vector formed by

$$
\zeta_F(\theta, \omega, W) = P_\theta(\tilde{\mathcal{J}}_{W,F} \mid \Omega = \omega), F \in \mathcal{F}.
$$

(iii) Let F be a graph with vertex set $[p]$. Let $\mathcal{A}_F$ denote the set of possible adjacency matrices of graphs isomorphic to F, i.e., the set of all distinct matrices $(A_F(s(i), s(j)))$ where A_F is the adjacency matrix of F and $s \in \mathcal{S}_n$.

Then, letting, for an adjacency matrix A, $\mathcal{J}_{W,A}$ denote the set of graphs $G \in \mathcal{K}_n$ such that $W \subset G$ and $A_{G_F} = A$, we have

$$
P_\theta(\tilde{\mathcal{J}}_{W,F} \mid \Omega = \omega) = \frac{1}{|\mathcal{A}_F|} \sum_{A \in \mathcal{A}_F} \eta(\theta, \omega, W, \tilde{A})
$$

where

$$\eta(\theta, \omega, W, A) = |[A]| P_\theta(\mathcal{J}_{W,A} \mid \Omega = \omega),$$

where $[A]$ denotes the isomorphism class of A, and we have $|[A]| = |\mathcal{A}_F|$ if $A \in \mathcal{A}_F$.

(iv) We have

$$\begin{aligned}
&(\zeta(\theta, \tilde{\omega}, \tilde{W}) - \bar{H})^T \Xi (\zeta(\theta, \omega, W) - \bar{H}) = \\
&\quad = \sum_{F \in \mathcal{F}} \Xi_F (\zeta_F(\theta, \tilde{\omega}, \tilde{W}) - \bar{H}_F)(\zeta_F(\theta, \omega, W) - \bar{H}_F) \\
&\quad = \sum_{F \in \mathcal{F}} \Xi_F |\mathcal{A}_F|^{-2} \\
&\qquad \sum_{A, \tilde{A} \in \mathcal{A}_F} (\eta(\theta, \tilde{\omega}, \tilde{W}, \tilde{A}) - \bar{H}_F)(\eta(\theta, \omega, W, A) - \bar{H}_F) \\
&\quad = \operatorname{tr}(\Xi) E\left((\eta(\theta, \tilde{\omega}, \tilde{W}, \tilde{\mathbb{A}}) - \bar{H}_{\tilde{\mathbb{A}}})(\eta(\theta, \omega, W, \mathbb{A}) - \bar{H}_{\mathbb{A}})\right)
\end{aligned}$$

where we have introduced the random variables $(\mathbb{A}, \tilde{\mathbb{A}})$ whose joint distribution is as follows: first choose a graphlet class $\mathbb{F}$ over $\mathcal{F}$ with probability proportional to Ξ_F and, conditionally to $\mathbb{F} = F$, take $\mathbb{A}$ and $\tilde{\mathbb{A}}$ independent and both uniformly distributed over $\mathcal{A}_F$. Here, we made the abuse of notation $\bar{H}_A := \bar{H}_F$ for any F with adjacency matrix A.

By reformulating our loss function in this way, we ensure that it is differentiable. η is a probability, and does not suffer from the same discontinuities that H does a priori, being based on counting.

(v) This leads to our SGD implementation, that computes a sequence of parameters $(\theta_t, t \geq 1)$ using

$$\theta_{t+1} = \theta_t - \gamma_t \sum_{i=1}^{L} \sum_{j=1}^{M} \partial_\theta \Big((\eta(\theta, \tilde{\omega}^{(j)}, \tilde{W}^{(i)}, \tilde{A}) - \bar{H})^T (\eta(\theta, \omega^{(j)}, W^{(i)}, A) - \bar{H}) \Big) \tag{4.2}$$

where γ_t is the learning rate, $\omega^{(1)}, \ldots, \omega^{(M)}$, $\tilde{\omega}^{(1)}, \ldots, \tilde{\omega}^{(M)}$, $W^{(1)}, \ldots, W^{(L)}$, $\tilde{W}^{(1)}, \ldots, \tilde{W}^{(L)}$ are independent samples of Ω and $\mathbb{W}$ and $A, \tilde{A}$ independent samples of $\mathbb{A}$.

To complete the presentation of the SGD algorithm, we need make explicit the computation of $\eta(\theta, \omega, W, A)$ and its derivative in θ, as a function of the derivatives of

Ψ in θ that are supposed to be computable either explicitly or using known algorithms (such as back-propagation).

4.3.4 Conditional Expectations of Graphlets

Let $W \in [n]_p$. We let $k_i \in [p]$ denote the rank in W of one of its elements, i. Then, a graph $G = \mathcal{G}(\boldsymbol{b}, \boldsymbol{r})$ is such that $W \subset G$ and $G_W \sim F$ if and only if $b_j = 1$ for all $j \in W$ and there exists $A \in \mathcal{A}_F$ such that $r(i,j) = A(k_i, k_j)$ for $i, j \in W$, $i < j$. This shows that

$$\begin{aligned}\eta(\theta, \omega, W, A) &= |[A]| P_\theta(\mathcal{J}_{W,A} | \Omega = \omega) \\ &= |[A]| \prod_{i \in W} q_i \prod_{\substack{i,j \in W \\ i<j}} \phi(z_i, z_j)^{A(k_i,k_j)} (1 - \phi(z_i, z_l))^{1-A(k_i,k_j)}.\end{aligned}$$

Denote the right-hand side by $u_A(\boldsymbol{q}, \boldsymbol{z})$. Then

$$\partial_\theta \eta(\theta, \omega, W) = \sum_{i \in W} (\partial_{q_i} u_A \partial_\theta q_i + \partial_{z_i} u_A \partial_\theta z_i)$$

where, we recall, $(\boldsymbol{q}, \boldsymbol{z}) = \Psi(\theta, \omega)$ whose derivative in θ are assumed to be known. So, only the derivatives of u_A in $\boldsymbol{q}$ and $\boldsymbol{z}$ need to be made explicit, and since u_A is polynomial in these variables, the computation is elementary, with

$$\partial_{q_i} u_A(\boldsymbol{q}, \boldsymbol{z}) = \frac{1}{q_i} u_A(\boldsymbol{q}, \boldsymbol{z})$$

and, letting

$$L_{A,W}(\boldsymbol{q}, \boldsymbol{z}) = |[A]| \prod_{i \in W} q_i \prod_{\substack{i,j \in W \\ i<j}} \phi(z_i, z_j)^{A(k_i,k_j)} (1 - \phi(z_i, z_j))^{1-A(k_i,k_j)},$$

we have

$$\begin{aligned}&\partial_{z_i} u_A(\boldsymbol{q}, \boldsymbol{z}) = \\ &\qquad \sum_{\substack{j \in W \\ j \neq i}} \left(\frac{\partial_{z_i} \phi(z_i, z_j)}{\phi(z_i, z_j)} A(k_i, k_j) - \frac{\partial_{z_i} \phi(z_i, z_j)}{1 - \phi(z_i, z_j)} (1 - A(k_i, k_j)) \right) \\ &\qquad\qquad \times L_{A,W}(\boldsymbol{q}, \boldsymbol{z}).\end{aligned}$$

4.3.5 Extensions

The formulation of the SGD algorithm, leading to equation (4.2), relies on the fact that subgraph counts were used for the moment estimators, but not on the specific stochastic model used to generate the graphs. The approach that it suggests can be applied to graph models that differ from the one we have introduced and therefore defines a general learning strategy for random graphs. The feasibility of the approach however depends on whether the functions that we have denoted by $\eta(\theta, \omega, W, A)$ in Section 4.3.3 and their derivatives are easy to compute, as described in Section 4.3.4 for our model. This would be the case for any model that would define a latent variable $\mathbf{Y} = \Psi(\omega, \theta)$, like ours, conditionally to which vertex selection and edge insertion are modeled as independent variables. We now discuss a few examples.

4.3.5.1 Adjacency matrix model

Instead of using a kernel graph, one may choose to directly model the adjacency matrix of the random graph. To simplify the discussion, we restrict to the situation where the generated graphs have a fixed size (and therefore any vertex selection step is not included). In this model, the latent variable $\boldsymbol{Y}$ is a symmetric matrix with entries in the unit interval, and, conditionally to $\boldsymbol{Y} = \boldsymbol{y}$, an edge (i, j) is included in the graph with probability y_{ij} (before random relabelling). In that case,

$$\eta(\theta, \omega, W, A) = |[A]| \prod_{i \in W} q_i \prod_{\substack{i,j \in W \\ i<j}} \phi(z_i, z_j)^{A(k_i,k_j)} (1 - \phi(z_i, z_l))^{1-A(k_i,k_j)},$$

which is still a polynomial in ϕ. We have opted for using kernel graphs models rather than full adjacency matrices in our main model because the former are more parsimonious, in terms, in particular, of the dimension of the generative network, while still offering a wide modeling range. Moreover, they implicitly provide a linear embedding of the generated graph, which has clear advantages for data analysis.

4.3.5.2 Community graphs

Community models of random graphs may also be generated using our approach. In this case, rather than pairing representations to nodes, they are paired to communities, of which only a small fixed number, say, t are learned. Assuming that t is known, the corresponding latent variable is a collection of t vectors, $\boldsymbol{Y} = \boldsymbol{Z} = (Z_1, \ldots, Z_t)$. We also model an intermediary random variable $\boldsymbol{C}$ that assigns nodes to communities, such that (assuming n nodes) $\boldsymbol{C} = (C_1, \ldots, C_n) \in [t]^n$ with $C_1, \ldots, C_n$ independent and identically distributed. Letting $s_c = P(C_i = c)$, we now have

$$\eta(\theta, \omega, W, A) = |[A]| \times \sum_{c \in [t]^{|W|}} \prod_i s_{c_i} \prod_{\substack{i,j \in W \\ i<j}} \phi(z_{c_i}, z_{c_j})^{A(k_i,k_j)} (1 - \phi(z_{c_i}, z_{c_j}))^{1-A(k_i,k_j)}$$

Note that the s_c's are model parameters and therefore also need to be learned. Importantly, η is polynomial in s and ϕ, so that the previous SGD approach can be used, with the following simplification.

We recall that, in the previous SGD formulation, a set of nodes $\mathcal{W}$ was selected uniformly at random over all nodes in G, where the size of W is equal to the size of the graphlet being considered. In the community model described here, selection of $\mathcal{W}$ is equivalent to selecting a set of the representations produced by the model with replacement, since nodes now share representations. Adjacency matrices $\mathbb{A}$ and $\tilde{\mathbb{A}}$ were also chosen uniformly at random over $\mathcal{A}_F$. The set $\mathcal{A}_F$ is a permutation class of matrices, i.e., for each pair of matrices A, $\tilde{A} \in \mathcal{A}_F$, there exists a permutation matrix V such that $\tilde{A} = VAV^T$, and $\mathcal{A}_F$ is also closed under such permutations.

There is now redundancy between the above sum and the choices of A and $\tilde{A}$. Since the sum is over all possible choices of representations for the nodes in W, all permutations of a choice of representations are included in the sum as separate terms. Since, by sampling $\mathcal{A}$ in the previous SGD algorithm, we were only selecting a permutation, this step is no longer necessary with the community model. Computing the sum comes with computational cost, but removes one of the stochastic components from the algorithm.

The computation of the derivatives required for stochastic gradient descent is largely unchanged, and the derivatives for the new s parameters are trivial.

4.3.5.3 Partial Graphlets

When dealing with graphs that include many vertices with high degree, large edge combinations (i.e., large graphlets) may be needed to accurately model the data. However, due to the large number of graphlets of order larger than six, and of their probable scarcity in a finite dataset, it is necessary to select them and group them before computing a moment estimator.

We here introduce *partial graphlets*, that we define to be a graphlet where a subset of the edges are ignored. More formally, we define a partial graphlet of order p by a $p \times p$ matrix M whose entries are 1, 0, or -1. We then say that a subgraph G of size p with adjacency matrix A is consistent with a partial graphlet if, for some permutation matrix D,

$$(DAD)_{ij} = M_{ij} \quad \text{or} \quad M_{ij} = -1, \ \forall i, j \in [p].$$

This operation groups together graphlets of some maximum size that contain a specific pattern, avoiding the issue of seeing their probabilities shrinking too small.

In particular, we make use of the "star" partial graphlet:

$$M_{ij} = \begin{cases} 0 & i = j \\ 1 & i = 1, \ j \neq i \\ -1 & \text{otherwise} \end{cases}$$

At values of p larger than previously employed in this work, this partial graphlet is closely related to the probability of a node having high degree. Whether a chosen subgraph is consistent with this partial graphlet is also simple to check through a row sum of the subgraph's adjacency matrix.

4.4 Examples of Kernels

The graph kernel, ϕ, is an essential component of our model. Its selection is an important step when designing a model, and requires some basic understanding of the dataset being modeled. Useful information includes the average number of vertices, the distribution of vertex degrees, and any community structure information that may be available. Here, we list kernels that were used in experiments, including a discussion of strengths and weaknesses of each. Each kernel $\phi(z_1, z_2)$ acts as $\phi : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$.

4.4.1 Dot product kernel

The dot product kernel is

$$\phi(z_1, z_2) = \frac{|z_1^T z_2|}{|z_1||z_2|}.$$

This is among the most basic kernels, and is also that most commonly used when RDPGs are discussed. For small values of d, it has the advantage of training very quickly because of its simple functional form. However, if the graphs are sparse, the node representations $z_i \in \mathbb{R}^d$ must be nearly mutually perpendicular, requiring d to grow with the number of nodes in the produced graphs. For this reason, the dot product kernel is effective for small graphs (or small numbers of communities in the case of community graphs), but becomes inefficient as the graph size grows.

4.4.2 Complement dot product kernel

This kernel is simply the complementary probability of that produced by the dot product kernel, ie.

$$\phi(z_1, z_2) = 1 - \frac{|z_1^T z_2|}{|z_1||z_2|}.$$

If the graph contains large groups of nodes with very small probabilities of edges being formed between them, this kernel is highly effective. It was designed specifically as a solution for generating bipartite graphs.

4.4.3 Radial basis function kernel

The RBF or Gaussian kernel is

$$\phi(z_1, z_2) = \exp(-|z_1 - z_2|^2).$$

When this kernel is used, the probability of an edge between two nodes is dependent only on the proximity of their representations. Because of this, many more probabilities can be accurately modeled without the need for large d; for this reason, it vastly outperformed the dot product kernel in both accuracy and speed of training on all datasets with large numbers of nodes.

4.4.4 RBF kernel with scale factor

This is a variation on the previous kernel:

$$\phi(z_1, z_2) = \sqrt{(1 + |z_1|^2)(1 + |z_2|^2)} \exp(-|z_1 - z_2|^2).$$

With the unscaled RBF kernel, if a set of nodes all mutually have edge probability $p < 1$, all of their representations must exist at the same mutual distance from each other, which is difficult to achieve in low dimension d. By adding the scale factor, the set of nodes can all share a representation z^*, as long as that representation's distance to the origin creates edge probability $\phi(z^*, z^*) = p$. As such, this kernel is especially effective for graph distributions with community structure (for which, however, the variant described in Section 4.3.5.2 may be preferred).

Because this kernel is not invariant to the scale of the inputs, we found the most success when adding a penalty to the objective function controlling the size of the network's weights, i.e.,

$$\lambda(\theta) = |E_\theta(H) - \bar{H}|^2 + \delta g(\theta),$$

with

$$g(\theta) = \begin{cases} 0 & |\theta| < \kappa \\ |\theta - \kappa|^2 & |\theta| \geq \kappa \end{cases}$$

for some constants δ and κ. This ensures that the network outputs do not grow too large, avoiding, in particular, the region where the derivative of the kernel is close to 0.

4.4.5 Polynomial kernel

We include this example for completeness, as it was not used in our experiments. The normalized version of the polynomial kernel is

$$\phi(z_1, z_2) = \frac{(1 + z_1^T z_2)^c}{\sqrt{(1 + z_1^T z_1)^c (1 + z_2^T z_2)^c}}$$

for some integer c. This kernel again depends on the inner product of the inputs, similarly to the dot product kernels. Its nonlinearity in the inner product avoids the issue of dimensionality described above to some extent, but the kernel still suffers when large numbers of nodes all share a mutual edge probability. While it offers great flexibility, we found less practical use for this kernel than the dot product and RBF variations.

This page intentionally left blank.

References

1. Riesen, K. & Bunke, H. *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning* in *Structural, Syntactic, and Statistical Pattern Recognition* (eds da Vitoria Lobo, N. *et al.*) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), 287–297.
2. Kriege, N. & Mutzel, P. *Subgraph Matching Kernels for Attributed Graphs* 2012. arXiv: 1206.6483 [cs.LG].
3. Pan, S., Wu, J., Zhu, X., Long, G. & Zhang, C. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Transactions on Cybernetics* **47,** 744–758 (2017).
4. Goodfellow, I. *et al. Generative adversarial nets* in *Advances in neural information processing systems* (2014), 2672–2680.
5. Arjovsky, M., Chintala, S. & Bottou, L. *Wasserstein Generative Adversarial Networks* in *Proceedings of the 34th International Conference on Machine Learning - Volume 70* event-place: Sydney, NSW, Australia (JMLR.org, 2017), 214–223.
6. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. en (Dec. 2013).
7. Bao, J., Chen, D., Wen, F., Li, H. & Hua, G. *CVAE-GAN: Fine-Grained Image Generation Through Asymmetric Training* in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2017).
8. Zhang, M., Cui, Z., Neumann, M. & Chen, Y. *An end-to-end deep learning architecture for graph classification* in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).
9. De Cao, N. & Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *arXiv:1805.11973 [cs, stat].* arXiv: 1805.11973 (May 2018).
10. Hallonquist, N., German, D. & Younes, L. *Graph Discovery for Visual Test Generation* in *2020 25th International Conference on Pattern Recognition (ICPR)* ISSN: 1051-4651 (Jan. 2021), 7500–7507.
11. You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models* en. in *International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, July 2018), 5708–5717.
12. Bickel, P. J., Chen, A. & Levina, E. The method of moments and degree distributions for network models. *The Annals of Statistics* **39.** arXiv: 1202.5101 (Oct. 2011).

13. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K. & Borgwardt, K. *Efficient graphlet kernels for large graph comparison* in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (eds van Dyk, D. & Welling, M.) **5** (PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009), 488–495.

14. Lovász, L. *Large networks and graph limits* (American Mathematical Soc., 2012).

15. Diaconis, P. & Janson, S. Graph limits and exchangeable random graphs. *arXiv:0712.2749 [math].* arXiv: 0712.2749 (Dec. 2007).

16. Aronszajn, N. Theory of Reproducing Kernels. *Trans. Am. Math. Soc.* **68,** 337–404 (1950).

17. Young, S. & Scheinerman, E. *Random Dot Product Graph Models for Social Networks* in (Nov. 2007), 138–149.

18. Athreya, A. *et al.* Statistical Inference on Random Dot Product Graphs: a Survey. *Journal of Machine Learning Research* **18,** 1–92 (2018).

19. Zhang, L., Bian, W., Song, M., Tao, D. & Liu, X. *Integrating local features into discriminative graphlets for scene classification* in *International Conference on Neural Information Processing* (Springer, 2011), 657–666.

20. Jin, J., Ke, Z. & Luo, S. *Network global testing by counting graphlets* in *International Conference on Machine Learning* (PMLR, 2018), 2333–2341.

21. Janssen, J., Hurshman, M. & Kalyaniwalla, N. Model selection for social networks using graphlets. *Internet Mathematics* **8.** Publisher: Taylor & Francis, 338–363 (2012).

22. Ribeiro, P., Paredes, P., Silva, M. E., Aparicio, D. & Silva, F. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Computing Surveys (CSUR)* **54.** Publisher: ACM New York, NY, USA, 1–36 (2021).

Chapter 5

Extensions to Attributed Graphs

Having successfully created a system that can learn distributions of random graphs, we now seek to extend the system to richer classes of graphs. Here, we will attempt to add to the model in order to learn vertex- and edge-attributed graphs.

5.1 Attributed Graphs

Attributed graphs fall into two classes: Those with vertex attributes and those with edge attributes, with the possibility of both included in the intersection. Vertex attributes are common in a wide range of applications, especially chemistry [1], where nodes are often atoms and their attributes their atomic numbers, and social networking [2], where nodes tend to represent individuals and attributes their communities. Edge attributes, on the other hand, are very common in optimization and operations research, where edges could be shipping routes and the attributes their capacities [3], or where edges could even be baseball matches and the attributes the results or scores [4]

5.1.1 Vertex-Attributed Graphs

In a vertex-attributed graph, to each vertex we associate a list of attributes, which are generally categorical variables. There are two possible ways we can assign attributes to vertices. The first way is to start from one set of possible attributes $\mathcal{D}$, and to each vertex assign a fixed or random number of unique random attributes from $\mathcal{D}$. In this case, every combination of attributes is possible at each vertex.

The second way is to start from a *list* of sets of attributes $\mathbf{\Delta}_k$, $k = 1, 2, \ldots, \mu_v$, and then assign each vertex exactly one attribute from each set $\mathbf{\Delta}_k$. Every vertex will be assigned exactly μ_v attributes, and the vocabulary is more strict. We note that the first scenario is a special case of the second if each set $\mathbf{\Delta}_k$ has two elements; we can think of the binary choice of attributes in the second case as equivalent to inclusion/exclusion of an attribute in the first.

While the extension we build herein will easily adapt to either of the above scenarios, it is the latter on which we will focus. Formally, for a graph G with vertices $i = 1, 2, \ldots, n$, we associate a matrix of random variables X such that the $(i,k)^{\text{th}}$ entry of X is the random attribute for vertex i drawn from list $\mathbf{\Delta}_k$. In effect, we are defining a map

$$r_v : V \to \prod_k \mathbf{\Delta}_k.$$

5.1.2 Edge-Attributed Graphs

The edge-attributed case is very similar to the vertex case. Again, there are two possible methods of attribute assignment, but we will pay active attention only to the one-from-each scenario described above. We shall this time assume there are μ_e attribute sets, and index them as $\mathbf{\Lambda}_k$, $j = 1, 2, \ldots, \mu_e$.

For a graph G, we associate a three-mode tensor Y of random variables such that the $(i,j,k)^{\text{th}}$ entry of Y is the random attribute for edge (i,j) drawn from list $\mathbf{\Lambda}_k$. (Note: If edge (i,j) does not exist in G, we will instead enter a null symbol into the tensor.) In this way, each edge is associated exactly μ_e attributes. Similarly to above, we are defining a map

$$r_e : E \to \prod_k \mathbf{\Lambda}_k.$$

5.2 Adding Vertex Attributes

We now return to our model, and seek an extension that will allow us to not only learn the structure of random graphs, but also a distribution over vertex attributes. To accomplish this, we will make use of a secondary machine learning method built on top of the first, and train it using a new kind of graphlet.

Our task here parallels the one addressed in the previous chapter: We find ourselves with a sample of random graphs from some distribution, and we seek to train a machine learning method to learn that distribution from the sample. However, since we now have a fully-functional system capable of learning the edge structure of graphs in the distribution, we will henceforth assume that we can sample from the distribution, sans attributes, at will. This means that the remaining task is to find a way to sample attributes, conditionally given the edge structure of a graph.

5.2.1 Vertex-Attributed Graphlets

As before, we require a representation of graphs from the distribution that is vertex permutation-invariant, since our training data remains unlabeled. We now introduce a new kind of graphlet that includes attribute information for this purpose.

The k^{th} *vertex-attributed graphlet* is a subgraph of a particular size that additionally pays attention to one attribute k's value for each vertex in the subgraph, $k = 1, 2, \ldots, \mu_v$. Formally, let F be an attributed graph on p vertices and G a graph on n vertices. Denote by $H_F(G)$ the probability that a set from $[n]_p$ chosen uniformly at random induces a subgraph in G that is isomorphic to F both by edge structure and vertex attributes.

This definition means that two subgraphs of G with the same edge structure are now considered to be isomorphic to different k^{th} graphlets if the k^{th} attribute differs on their respective nodes. This notion of isomorphism is sometimes called *strong*, with *weak* isomorphism used for the case where the edge structures of two graphs are the same but their attributes differ.

We have now introduced a great many more graphlets. If we suppose $|\mathbf{\Delta}_k| = c$ $\forall k$, then there are now $\kappa_p \mu_v c^p$ vertex-attributed graphlets, where p is the order of graphlets considered and κ_p is the number of non-attributed graphlets of order p. This number rapidly becomes intractable for training purposes as p increases, so we will limit our attention to $p = 2$. This effectively means we will be learning from pairwise interactions of attributes only, which seems like an acceptable concession. (However, most of what we do in this chapter will not specifically require that $p = 2$, and we will make note of generalizations to higher orders where possible.) We will, for convenience in the next section, denote by $h_1^{(F)}$ and $h_2^{(F)}$ the two attributes on the two nodes in the graph F for which H_F is the corresponding graphlet, and let $h_3^{(F)} = 1$ if F contains and edge and 0 if it does not.

Since our implementation in the next section will assume the edge structure of the graph is known prior to creating the vertex attributes, we will find it convenient to use a set of *conditional* graphlets: We will replace each H_F by

$$H_F \to \frac{H_F}{\sum\limits_{\tilde{F} \in \mathbb{F}} H_{\tilde{F}}},$$

where

$$\mathbb{F} = \{\tilde{F} \mid \tilde{F} \sim F \text{ weakly}\}.$$

In the $p = 2$ case, the above reduces to

$$\mathbb{F} = \{\tilde{F} \mid h_3^{(\tilde{F})} = h_3^{(F)}\}.$$

5.2.2 Another Neural Network

We again are faced with the task of generating graphs from a set of graphlets, and likewise to the previous chapter, we will introduce a neural network to accomplish the task. This time, the input for the network will be adjacency matrices (either produced by the first neural network we built or taken directly from the distribution of graphs), and the output will be n lists of m_v attributes, each associated to a node in the graph.

5.2.2.1 SGD Formulation

We again train the neural net using a moment estimator, this time for the new attributed graphlets.

Formally, we say that our model defines a matrix X of random variables generated as a function $\Psi_2(\theta, A_G)$ where A_G is the adjacency matrix associated with a random graph G drawn from distribution $\mathbb{G}$, θ is a parameter, and Ψ_2 is assumed to be differentiable in θ. The $(i,j)^{\text{th}}$ entry of X is the j^{th} attribute associated with vertex i in G. As before, we progressively express the objective function U as the expectation, over an increasing number of random variables, of an increasingly simple function.

(i) If we introduce two independent copies of A_G (and drop the subscript), say, $A, \tilde{A}$, we can write

$$\begin{aligned} U(\theta) &= (E_\theta(H) - \bar{H})^T \Xi (E_\theta(H) - \bar{H}) \\ &= E(E_\theta(H - \bar{H} \mid A))^T \Xi E(E_\theta(H - \bar{H} \mid A)) \\ &= E(E_\theta(H - \bar{H} \mid \tilde{A})^T \Xi (E_\theta(H - \bar{H} \mid A)) \end{aligned}$$

where the outer expectation is with respect to the distribution of $A, \tilde{A}$.

(ii) We introduce $\mathbb{M} = \{m \subset [n] \mid |m| = 2, m \in E(G)\}$ and $\tilde{\mathbb{M}}$ likewise for $\tilde{G}$, and let M and $\tilde{M}$ be uniformly drawn from $\mathbb{M}$ and $\tilde{\mathbb{M}}$ respectively. We also write $\boldsymbol{\sigma}$ to represent the set of possible permutations of a p-element set (such as M).

(iii) As before, we can write the loss function as

$$U(\theta) = E(E_\theta(H - \bar{H} \mid A)^T E_\theta(H - \bar{H} \mid \tilde{A})),$$

and further express

$$E_\theta(H_F \mid G) = \frac{1}{|\boldsymbol{\sigma}|} \sum_{\sigma \in \boldsymbol{\sigma}} \eta(\theta, a, m, \sigma, F)$$

with $\eta(\theta, a, m, \sigma, F)$ representing the probability that the pair of nodes M selected from A and assigned ordering σ form the graphlet designated by F.

(iv) We have now introduced a procedure for estimating the loss function: First, select two graphs without attributes A and $\tilde{A}$ to consider; choose two nodes from each graph M and $\tilde{M}$; and compute the probabilities η that these node pairs would form the given graphlets when assigned attributes.

(v) This leads to our SGD implementation, which computes a sequence of parameters $(\theta_t, t \geq 1)$ using, as before,

$$\theta_{t+1} = \theta_t - \gamma_t \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \partial_\theta \Big((\eta(\theta, \tilde{a}^{(i)}, \tilde{m}^{(j)}, \tilde{\sigma}^{(j)}, H) - \bar{H})^T \\ (\eta(\theta, \tilde{a}^{(i)}, m^{(j)}, \sigma^{(j)}, H) - \bar{H}) \Big) \quad (5.1)$$

where γ_t is the learning rate, $a^{(i)}$, $\tilde{a}^{(i)}$, $m^{(j)}$, $\tilde{m}^{(j)}$, are independent samples of A and M and $\sigma^{(j)}, \tilde{\sigma}^{(j)}$ are independent samples chosen uniformly between σ_1 and σ_2.

Similarly to last time, to complete the presentation of the SGD algorithm, we need to make explicit the computation of $\eta(\theta, a, m, \sigma_i, H)$ and its derivative in θ. These computations will necessarily need to be explicit or able to be computed using algorithms such as backpropagation.

5.2.3 Conditional expectations of graphlets

In contrast to the math laid out in Section 4.3.4, the calculation of $\eta(\theta, a, m, \sigma, H)$ here is quite simple. Let h_1, h_2 be two attributes specified in graphlet F. Then order the elements of M (two nodes) according to σ as m_1, m_2. We will denote by q the output of the neural network; q can be viewed as a 3-dimensional array, with number of rows equal to the number of nodes in the graph, number of columns equal to the number μ_v of attribute lists, and height equal to the length of the attribute lists. (We note that the attribute lists may have unequal lengths, so it is somewhat improper to refer to q as an array, but we will index it like one all the same.)

In this case, we find that

$$\eta(\theta, a, m, \sigma_i, F) = |[F]| \prod_{j=1}^{m_v} \frac{q_{m_1,j,h_1}}{\sum_k q_{m_1,j,k}} \prod_{j=1}^{\mu_v} \frac{q_{m_2,j,h_2}}{\sum_k q_{m_2,j,k}}$$

where

$$|[F]| = \begin{cases} 2 & H \text{ is symmetric} \\ 1 & H \text{ is not symmetric} \end{cases}.$$

Denote the right hand side by ψ. We can then write

$$\partial_\theta \eta(\theta, a, m, \sigma_i, F) = \sum_{i,j,k} \frac{\partial \psi}{\partial q_{ijk}} \frac{\partial q_{ijk}}{\partial \theta}$$

and we can express the derivatives of ψ in cases:

$$\frac{\partial \psi}{\partial q_{ijk}} = \begin{cases} \dfrac{\frac{2}{q_{ijk}} \sum_l q_{ijl} - 2}{\sum_l q_{ijl}} \psi & h_1 = h_2 = k \\ \dfrac{\frac{1}{q_{ijk}} \sum_l q_{ijl} - 2}{\sum_l q_{ijl}} \psi & h_1 = k, \ h_2 \neq k \\ \dfrac{-2}{\sum_l q_{ijl}} \psi & h_1 \neq k, \ h_2 \neq k \end{cases}.$$

The derivatives of q_{ijk} can be easily computed using backpropagation. Armed with these derivatives, we can compute the SGD steps for θ and complete the learning process.

5.3 Adding Edge Attributes

In lieu of employing yet another neural network and introducing a few dozen more graphlets, we choose to make the simplifying assumption that the probability that an edge has a certain attribute depends only on the vertex attributes located at each of its endpoints. This greatly reduces the number of codependencies in the model, setting all edge attributes to be conditionally independent given the vertex attributes.

We simply use the empirical edge attribute probabilities from our sample of graphs in order to calculate these dependencies. We simply set

$$P(Y_{ij\cdot} = y_{ij\cdot} | X_{i\cdot} = x_{i\cdot}, X_{j\cdot} = x_{j\cdot} = \frac{1}{|B|} \sum_{(u,v) \in B} 1(Y_{uv\cdot} = y_{uv\cdot})$$

where B is the set of all pairs of vertices in the sample $\mathfrak{G}$ who share an edge and have respective vertex attributes $x_{i\cdot}$ and $x_{j\cdot}$.

As long as our dataset is large enough relative to the number of vertex and edge attributes, these probabilities should be usable. To put the probabilities in context, a dataset with 200 graphs on 200 nodes each yields nearly 4,000,000 possible node pairs. A typical problem that we will address in the next chapter may have around 32,000 possible combinations of attributes, so even the rarer combinations of attributes should still receive a reasonable sample size.

We note that, in the absence of vertex attributes, this approach will still function, though our simplifying assumption may then appear a bit too simplifying. In that case, the edge attribute probabilities would be homogeneous for all edges in every graph, which is probably not realistic. We may choose to introduce some artificial vertex attributes in lieu of real ones, utilizing graph characteristics that always can be computed. For example, we might assign each vertex an attribute equal to its degree, or choose to color all the vertices using standard minimal graph coloring techniques.

5.4 Complete Model

With these additions, our end-to-end model is now complete. The procedure for generating new graphs from a sample $\mathfrak{G}$ is as follows:

1. Convert $\mathfrak{G}$ to a set of graphlets H, including vanilla graphlets, vertex attributed graphlets, and edge attribute probabilities.
2. Train the base neural network $\Psi^{(1)}$, which generates latent positions for kernels, using H.
3. Generate new edge structures by inputting random noise Ω into $\Psi^{(1)}$.
4. Train the vertex attribute neural network $\Psi^{(2)}$ using the new edge structures and H.
5. Generate new edge structures using $\Psi^{(1)}$, then use those edge structures as inputs to $\Psi^{(2)}$ to generate new vertex attributes.
6. Generate edge attributes using the edge attribute probabilities calculated in Step 1.

This algorithm is capable of generating all parts of a random graph, including edge structure, vertex attributes, and edge attributes.

This page intentionally left blank.

References

1. Trinajstic, N. *Chemical graph theory* (CRC press, 2018).
2. Scott, J. *What is social network analysis?* (Bloomsbury Academic, 2012).
3. Leighton, T. & Rao, S. *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms* tech. rep. (Massachusetts Inst. of Tech. Cambridge Microsystems Resesarch Center, 1989).
4. Schwartz, B. L. Possible Winners in Partially Completed Tournaments. *SIAM Review* **8,** 302–308 (1966).

This page intentionally left blank.

Chapter 6

Implementation and Results

In this chapter, we use and give results from the main model and extensions presented in Chapters 4 and 5.

6.1 Main Implementation

In addition to the kernel discussed in Chapter 4, the training algorithm needs to be provided with the maximum number of nodes, n, maximum size of graphlets used for learning, the geometry of the neural network used to train the function Ψ_1, the distribution of its input Ω and the weight matrix Ξ used in the definition of the objective function. Additional inputs can also specify some parameters of the SGD procedure, such as the number of examples used in minibatches.

In our implementation, a library of graphlets of size up to $p = 8$ was pre-computed and an array of size $p(p-1)/2$ was built and stored for fast retrieval of the graphlet class associated with a $p \times p$ binary connectivity matrix. We used a feed-forward fully connected neural network with two hidden layers for the function Ψ. Its input layer, forming the variable Ω, was generated as independent standard Gaussian variables. The activation function was Leaky ReLU, i.e.,

$$\tau(x) = \begin{cases} x & x \geq 0 \\ \epsilon x & x < 0 \end{cases}$$

with $\epsilon = 0.01$. The parameter θ is associated with the network's weights. The implementation was programmed using PyTorch, allowing for a straightforward computation of the derivatives of Ψ with respect to θ.

The maximum number of nodes, n, was chosen to be the maximum number of nodes in any graph in the dataset in use.

While training, a step-down step size strategy was employed. Fix a number of phases, κ, target values of cost $u_1, \ldots, u_\kappa$ and step sizes $\gamma_0 > \cdots > \gamma_{\kappa-1}$. We run the algorithm with step size γ_0 until the first time t_1 such that $U_{t_1}(\theta) < u_1$. We then use γ_1 until the first time $t_2 > t_1$ such that $U_{t_2}(\theta) < u_2$ and so on until time t_κ is reached and the procedure is stopped. (We used $\kappa = 3$ in our experiments.) For this purpose, $U(\theta)$ was computed at regular intervals of training time.

6.2 Datasets

The algorithm was tested on a variety of datasets. It was first applied to a number of synthetic datasets, generated from popular generative models for random graphs. It was then tested on a series of real datasets with different characteristics, several of which were accessed from a repository at [1]. Details of each dataset are given below and in Table 6-I.

6.2.1 Empty graph

As a first step and sanity check, we attempt to imitate the empty graph distribution, i.e., graphs that always have zero edges. With the dot product kernel, this requires all representations to be mutually perpendicular. With the RBF kernel, the representations just must be as isolated as possible in Euclidean space.

6.2.2 Stochastic Block Models

We attempt to learn the distribution of a class of graphs generated from a particular stochastic block model (SBM). In an SBM, each node is randomly assigned to one of several "blocks," and it is the block memberships of a pair of nodes that determine the probability of an edge existing between them. Two SBMs were chosen. The first is the two-block model described in [2] (there called "Community") for comparison purposes. The second is a four-block model with higher within-block edge probabilities so that clustering of the representations can be cleanly visualized. For the two-block model, the membership probabilities for each block are

$$\pi = [0.5, 0.5].$$

The probability of an edge existing between a node from block i and a node from block j is given by the i^{th}, j^{th} entry of the matrix

$$\mathbf{\Gamma} = \begin{pmatrix} 0.3 & 0.05 \\ 0.05 & 0.3 \end{pmatrix}.$$

For the four-block model, the membership probabilities for each block are

$$\pi = [0.25, 0.25, 0.25, 0.25]$$

and the community probability matrix is

$$\mathbf{\Gamma} = \begin{pmatrix} 0.75 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.75 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.75 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.75 \end{pmatrix}.$$

It is simple to show that if B is positive semidefinite, the SBM is a submodel of the RDPG [3], so the method described above should be able to imitate the distribution produced by these SBMs.

6.2.3 AIDS

This is a chemical dataset, sourced from [4]. It consists of graphs from two classes: Compounds that are active against the HIV virus, and compounds that are not. Each graph represents a single chemical compound, where nodes represent atoms and edges represent covalent bounds between the atoms. We attempt to learn the "Active" class of graphs, and use the other class for comparison.

6.2.4 COX2_MD

This is a publicly available chemical dataset, converted to a graph format by [5]. Each graph represents a cyclooxygenase-2 inhibitor. Nodes represent atoms and edges exist between nodes if they are sufficiently close together in physical space.

6.2.5 OHSU

This dataset was constructed by [6] out of BrainNet Functional Brain Network Analysis Data. Here, nodes parcellate regions of the brain, and edges represent correlations between the regions. The two classes in this set represent typical brains and brains with Hyperactive-Impulsive classification.

6.2.6 Brain

Provided by [7] and first used in [8, 9], this dataset consists of diffusion MRI data. Each graph is a component of a connectome, where nodes represent regions of the brain and edges exist between nodes when there are fiber streamlines connecting the regions.

6.2.7 Protein

Used in [2]. Here, each graph represents a protein, comprised of amino acids. Each node represents an amino acid, and two nodes are connected if their amino acids are physically less than six angstroms apart in the protein structure.

6.2.8 IMDb

Used in [10]. The dataset consists of ego graphs created using the IMDb network. In the IMDb network, nodes represent actors and an edge exists between two actors if they appeared in the same film. In an ego network, there is one central node (the "ego") around which the rest of the network is constructed. In this case, a particular actor was chosen as the ego for each graph in the dataset, and any node connected to the ego was included in the graph. All edges between selected nodes were included as well.

6.3 Main Results

6.3.1 Basic Model

We start with some results and validation of our basic model presented in section 4.2.2.2. For this first set of results, the matrix Ξ in (4.1) is the identity matrix.

6.3.1.1 Training performance

Table 6-III provides some details on the outcome of the training algorithm in terms of correctly learning the graphlet expectations.

The results of the base algorithm's attempts to learn various random graph distributions were favorable. On the simplest examples, the algorithm is able to complete its task quickly and perfectly. When asked to learn the empty graph distribution, the algorithm is able to produce the desired results exactly within a few hundred iterations.

When attempting a more complicated example, such as the four-block stochastic block model described in section 6.2.2, success can be found with the right combination of hyperparameters, such as the number of nodes in the produced graphs, the dimension of their representations, the minibatch size, and the number of neurons in the neural network. In the case of the four-block SBM described above, ten-dimensional representations were used to create graphs on 80 nodes. Each hidden layer of the NN contained ten neurons. Within a few thousand iterations, the algorithm was able to match the graphlet distribution from the SBM to within 0.01%.

The embeddings that resulted from the application of the algorithm in the case of

Dataset	Source	Num. Graphs	Avg. Nodes	Avg. Edges	Tot. Diff.	Max Diff.	Kernel
Empty graph	N/A	N/A	10	0	0.000	0.000	RBF3
4-block SBM	N/A	N/A	16	45	0.024	4.3e-3	DP4
2-block SBM	N/A	N/A	80	548	0.067	8.8e-3	RBF4
AIDS	[4] [11]	2000	15.69	16.20	0.015	1.8e-3	DP5
COX2_MD	[5]	303	26.28	335.12	0.045	5.2e-3	DP5
OHSU	[6]	79	82.01	199.66	0.052	8.8e-3	RBF5
Brain	[7]	114	70	1037	0.101	7.5e-3	RBF5
Protein	[2]	1113	39.05	72.82	0.071	8.2e-3	RBF5
IMDb	[10]	1000	19.77	96.53	0.122	9.3e-3	RBF5

Table 6-I. Description of the datasets used to test the generator. The first three are artificial, created from a known distribution. "Total Difference" is the total absolute difference between the target graphlets and the generator graphlets after training, with an optimal value of 0. "Max Difference" is the largest difference for a single graphlet between the target and generator. The best-performing kernel for each is listed: DP for dot product, RBF for radial basis function. The number following is the order of graphlets used to produce results.

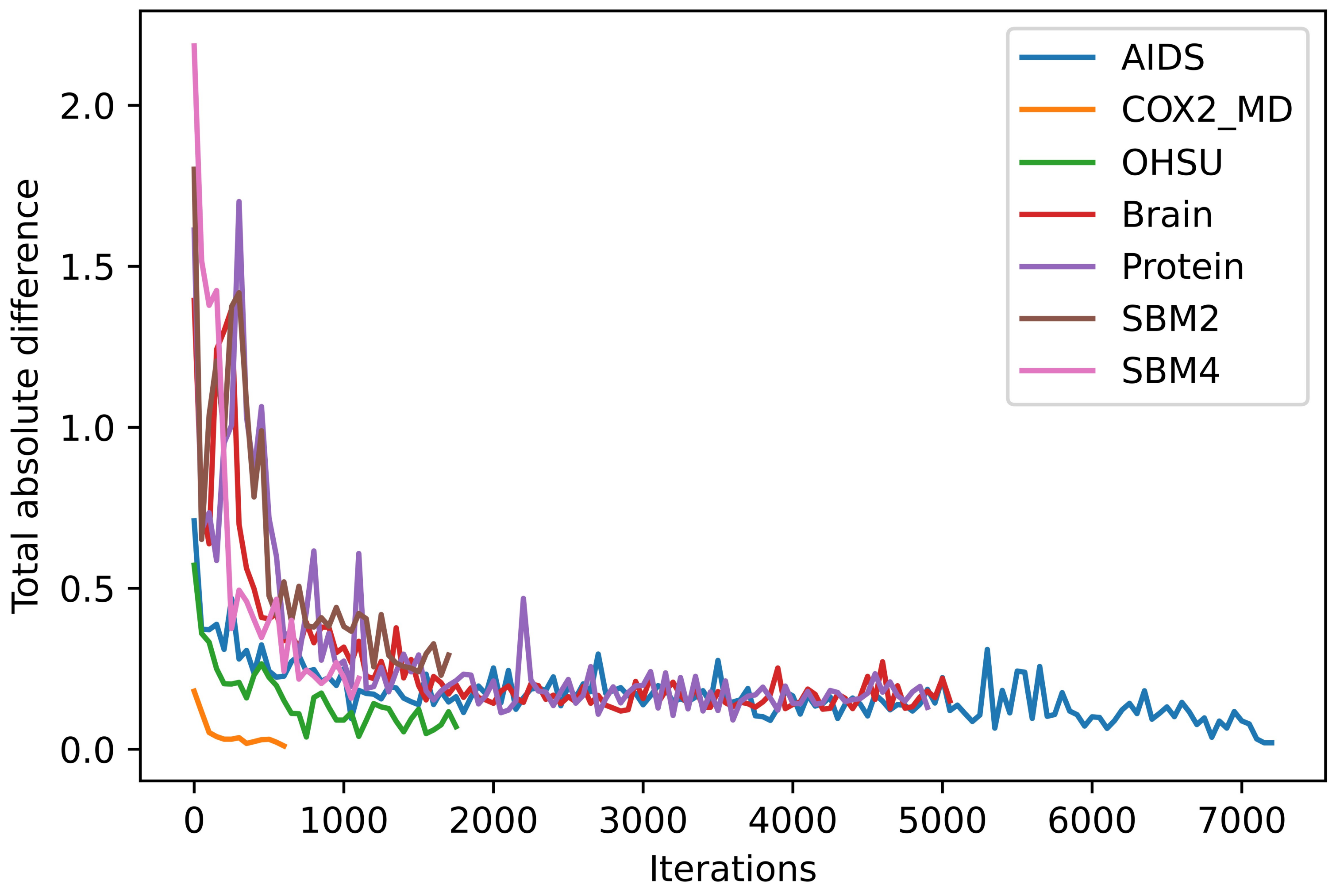


Figure 6-1. Training error for the latent variable generator for each dataset. The y-axis shows the summed difference between the target vector of graphlets and the vector of graphlets produced by the generator.

the four-block SBM were separated into four clusters, with each cluster corresponding to a block in the SBM. Nodes whose embeddings are close together are more likely to have en edge between them, and thus should be members of the same block in the SBM. A two-dimensional projection of the embeddings produced using principal component analysis can be seen in Figure 6-1.

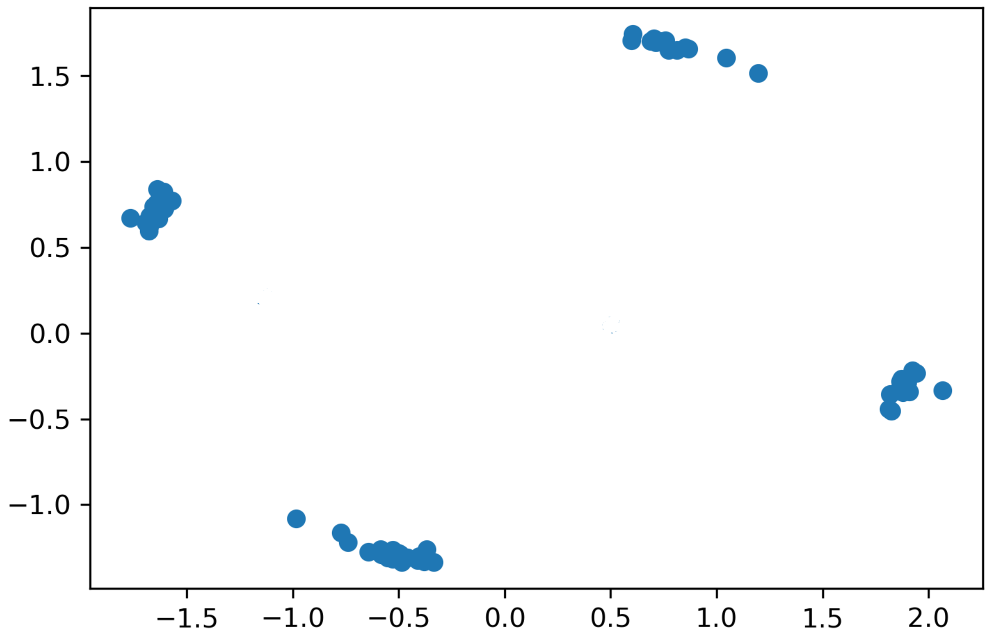


Figure 6-2. PCA visualization of node representations for four-block SBM trained using an RBF kernel. Each point represents one node in the graphs that will be produced by the generator; PCA was performed on the latent vectors it produces. Clear clustering of these latent vectors has occurred, indicating successful learning of the SBM community structure.

After running these test cases, the algorithm was used to learn the distributions of the several real datasets listed above. In these cases, the algorithm quickly converges to a solution, closely matching the desired graphlet proportions from the target distribution. The metric for error in these cases is the total absolute difference between the target graphlet proportions and the graphlet proportions of graphs produced by the network midway through training. The algorithm could usually

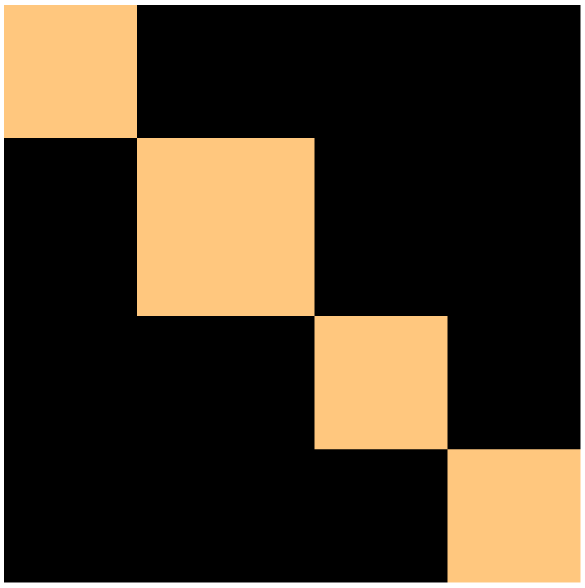


Figure 6-3. The model was trained on the 4-block SBM. This matrix shows the frequency with which each node fell into the same community as each other node. Four clear communities are established, with zero mixing between them.

reduce the absolute difference to less than 5% of the overall graphlet proportions.

Remark. An examination of the typical output of the trained neural network reveals a shortcoming of this model. When training for a dataset with community structure, such as the SBM dataset, the embeddings cluster together, which is desirable. However, embeddings tend to become "stuck" in their clusters, and do not typically move between clusters when different inputs are passed through the net. As a result, in the case where the number of nodes in the produced graphs are fixed (ie. $q_i = 1 \ \forall i$), each community will always contain the same number of nodes. This does not properly capture the behavior of the SBM or other community models, where nodes are randomly assigned to communities each time a graph is realized. Figure 6-3 was produced by a net trained on the SBM described above; it is a heatmap visualization of a matrix whose i, j^{th} entry is the frequency that the i^{th} node fell in the same community as the j^{th} node. This shortcoming limits the usefulness of the model when applied to certain types of datasets; however, using our community model from 4.3.5.2 properly addresses the issue (see section 6.3.2.1).

6.3.1.2 Validation

In order to further test the validity of the algorithm, we evaluated the performance of neural net classifiers in separating true from generated data in our datasets, where poor performance in classification is an indicator of the quality of the generative model.

Two NN classifiers were constructed. The first is a standard feed-forward neural network with two fully-connected hidden layers. The features used by this network were the graphlets of order one larger than those used to train the generators. These features were obtained for graphs in the datasets using the sampling technique described in section 4.3.2.1. The resulting classifier is powerful enough to distinguish with near perfect accuracy between the two real classes from the AIDS, COX2_MD, or OHSU datasets.

We used Graph Neural Nets (GNNs) as second classifier, because they share less with the generator in terms of features and architecture. (GNNs were introduced in Section 2.3.1.) GNNs, which are now fairly ubiquitous for graph classification tasks [12, 13], make use of convolutional operations specially suited for graph data. A simple GNN was employed here, with two hidden graph-convolutional layers, and node degrees used as the input features. The GNN was less effective than the graphlet-based classifiers at distinguishing between classes of the real datasets, but still was able to discriminate effectively enough for testing purposes.

After training, all classifiers performed poorly on the six real datasets. Results are summarized in Table 6-II. Experiments generally used graphlets up to order 5, but results for smaller graphlets are depicted in Figure 6-4.

To further evaluate the model, we make use of the same Maximum-Mean Discrepancy (MMD) metrics introduced by [2]. The graph statistics used there are degree distributions, clustering coefficient, and orbit counts. A graph's clustering

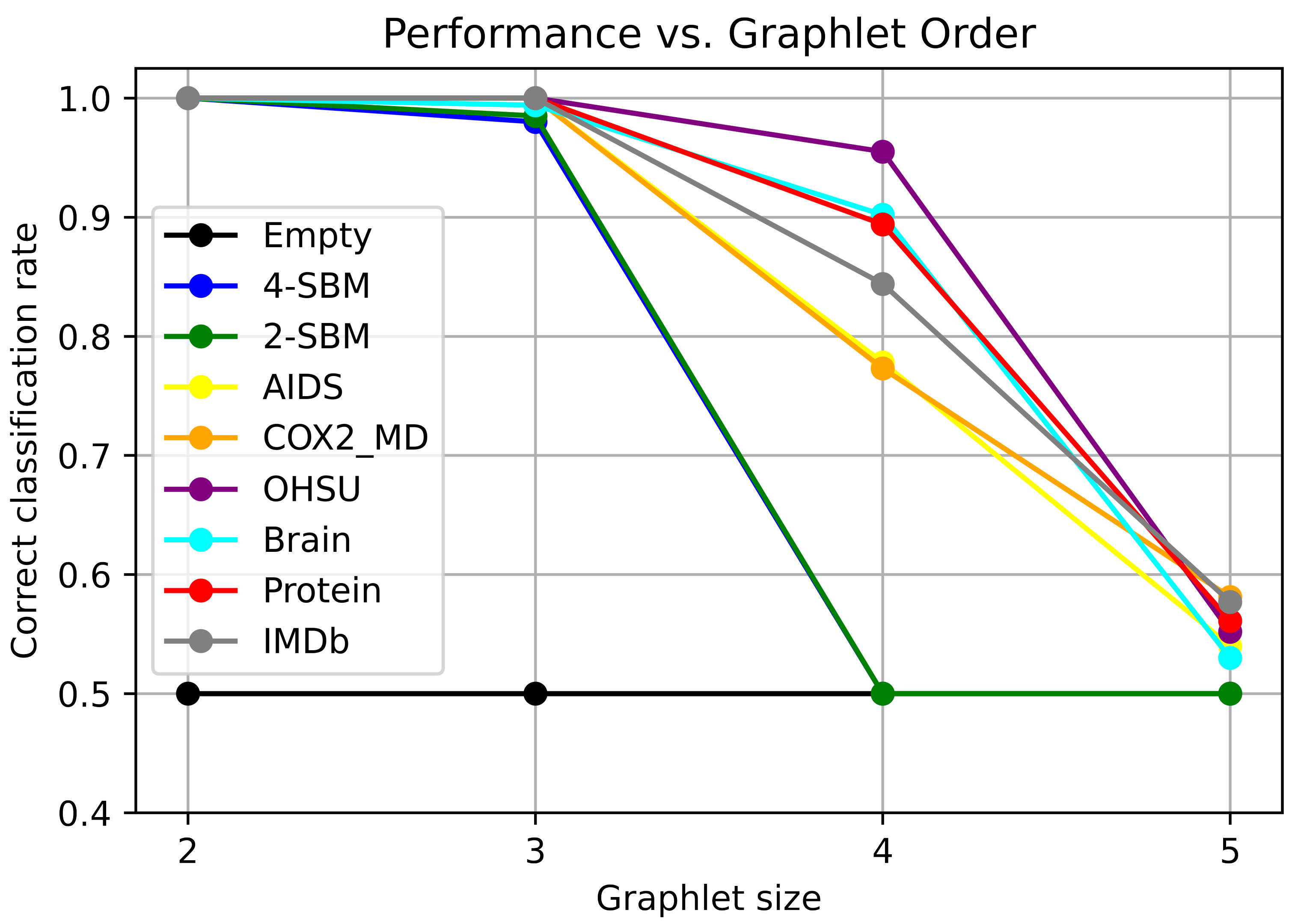


Figure 6-4. Ability of the graphlet classifier to correctly separate real graphs from simulated graphs for each dataset. The given size of graphlets is the size used for training; the classifier used graphlets of one size larger.

Dataset	Graphlet	GNN
Empty graph	0.500	0.500
4-block SBM	0.500	0.500
2-block SBM	0.500	0.500
AIDS	0.540	0.581
COX2_MD	0.581	0.696
OHSU	0.552	0.579
Brain	0.530	0.539
Protein	0.561	0.573
IMDb	0.577	0.621

Table 6-II. This table lists the results of the two tests described in Section 4 for each dataset used. In each test, a classifier was trained to distinguish between real data and generated data. The given rate is how often the classifier correctly classified a graph, with an optimal value of 0.5.

coefficient and orbit counts are equivalent to the "triangle" three-graphlet and the list of four-graphlets, respectively. On the two-block SBM and Protein datasets, we compare directly to GraphRNN [2]. Results can be found in Table 3 below.

6.3.2 Results for extensions.

In this section, we give additional results from applying various extensions and changes to the model.

6.3.2.1 Community model.

We described an alternate version of the model tailored to graph data with community structure in Section 4.3.5.2. This version of the method learns only a small number of node representations, allowing it to be applied to datasets with significantly larger numbers of nodes, provided they exhibit the appropriate structure. Here, we make use of the two-block SBM described in Section 6.2.2. We train the model using the RBF kernel with scale factor, up to graphlets of size five. The results are in Table 6-IV.

The power of this alternative version of the model, however, is its ability to match graphs of much larger size. We increase the number of nodes to 10,000 and perform

Dataset	4-block SBM			AIDS			COX2_MD			OHSU		
	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.
Self	0.101	3.75e-3	6.84e-3	2.011	1.27e-2		0.401	1.17e-3	1.54e-3	0.352	2.67e-2	1.04e-2
GraphMoE	0.225	8.12e-3	2.06e-2	1.404	2.10e-4	1000	1.676	2.77e-2	5.01e-2	1.455	2.99e-2	8.83e-3
Dataset	Brain			IMDb			2-block SBM			Protein		
	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.	Deg.	Clust.	Orb.
Self	0.245	1.30e-2	5.39e-3	1.660	3.71e-2	1.45e-2	5.44e-2	4.10e-4	1.10e-3	4.20e-2	9.23e-3	1.45e-2
GraphMoE	0.251	5.44e-2	7.23e-3	5.681	8.69e-2	4.33e-2	0.475	**9.37e-4**	**8.62e-3**	0.767	**3.88e-2**	**5.71e-2**
GraphRNN	-	-	-	-	-	-	**1.40e-2**	2.00e-3	3.90e-2	**3.40e-2**	0.935	0.217

Table 6-III. This table lists the values of three differentiation statistics for each real dataset. First, the dataset is split in half and the halves are compared. Then, the real data is compared to data generated by GraphMoE. For the 2-block SBM and Protein datasets, the statistic values for real data vs. data generated by GraphRNN are also given.

the same experiment. Since we have been using sampling to compute graphlets, it is no more difficult to calculate graphlets for small graphs than for large ones, and so the number of nodes included can be as large as desired. The results are again in Table 6-IV.

We also include visualizations of a larger graph generated from the SBM and from the model trained to imitate it; see Figure 6-5. The drawings include 1000 nodes, as drawing more becomes computationally intensive. (*Using* more is not any more difficult.) The visualizations were created using the *igraph* package in Python.

Dataset	Total Diff.	Max Diff.	Classification Rate
16 Nodes	0.0451	6.6e-3	0.557
10,000 Nodes	0.0194	9.4e-4	0.509

Table 6-IV. Performance of community model trained on the four-block SBM. The classification rate is the rate at which the graphlet classifier could correctly separate SBM graphs from learned graphs, with optimal value 0.5.

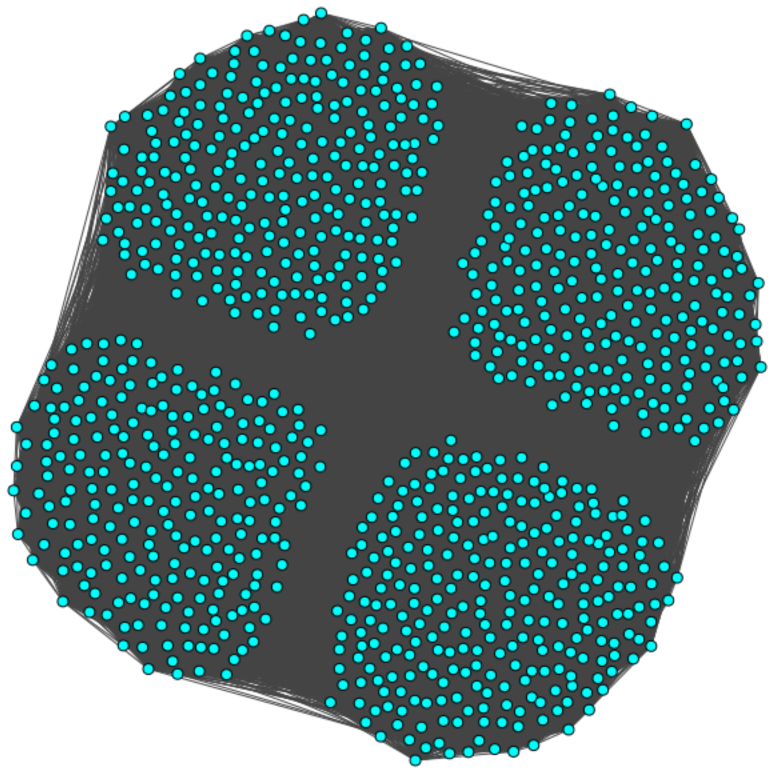
6.3.2.2 Inversely-weighted loss function.

For most datasets, there can be large variations among the frequencies of some graphlets of given order. For instance, the AIDS dataset is quite sparse, and of all its subgraphs on 3 nodes, the subgraph containing one edge is ten times more present than the subgraph containing two edges. However, we find that these less common graphlets often contain much of the information about the overall random graph distribution, and matching them less than perfectly can have powerful effects on the results.

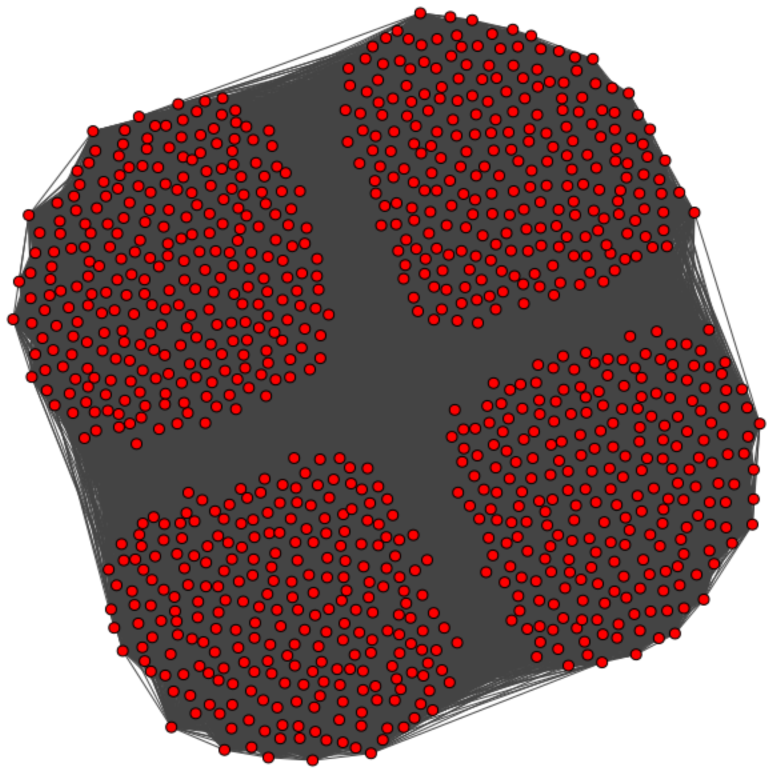
In such cases, it may be preferable to reweight the loss function to increase the importance of rare graphlets. More precisely, we can replace the function $\lambda(\theta) = |E_\theta(H) - \bar{H}|^2$ by

$$\lambda^*(\theta) = \sum_{F \in \mathcal{F}} \frac{(E_\theta(H_F) - \bar{H}_F)^2}{|\bar{H}_F|}.$$

This corresponds to using a diagonal matrix D with diagonal elements given by $|\bar{H}_F|^{-1}$



(a) A graph produced by the SBM.



(b) A graph produced by the model.

Figure 6-5. Comparison of graphs from the community model. The topological arrangement of nodes optimized by the *igraph* package are quite similar. (Due to the very large number of edges, the connectivity structure cannot be observed at this image resolution.)

in equation (4.1).

The overall result is an increase in the "Total Difference" between the target graphlets and the produced graphlets, but slight improvements in the ability of the model to fool the discriminators. Numerical results for three of the datasets are included in Table 6-V below.

Dataset	Total Difference	Classification Rate
AIDS	0.102	0.556
COX2_MD	0.113	0.687
Brain	0.157	0.515

Table 6-V. Results for the inversely-weighted loss function on three of the datasets, providing a slight improvement on the results in table 6-III

6.3.2.3 Use of larger graphlets.

We now show how using partial graphlets (Section 4.3.5.3) can help better fitting the degree distribution in our simulated graphs. (We note that GraphMoE is outperformed by GraphRNN in this metric on most datasets.)

In Figure 6-6 we display a comparison of the produced degree distributions to the ground truth for the two-block SBM and Protein datasets with and without use of the star partial graphlets for training. The addition of these higher-order graphlets does produce some visible improvement; however, it does not entirely fix the discrepancies. Going even further in the graphlet size would probably help, but we are limited by computing power and training time.

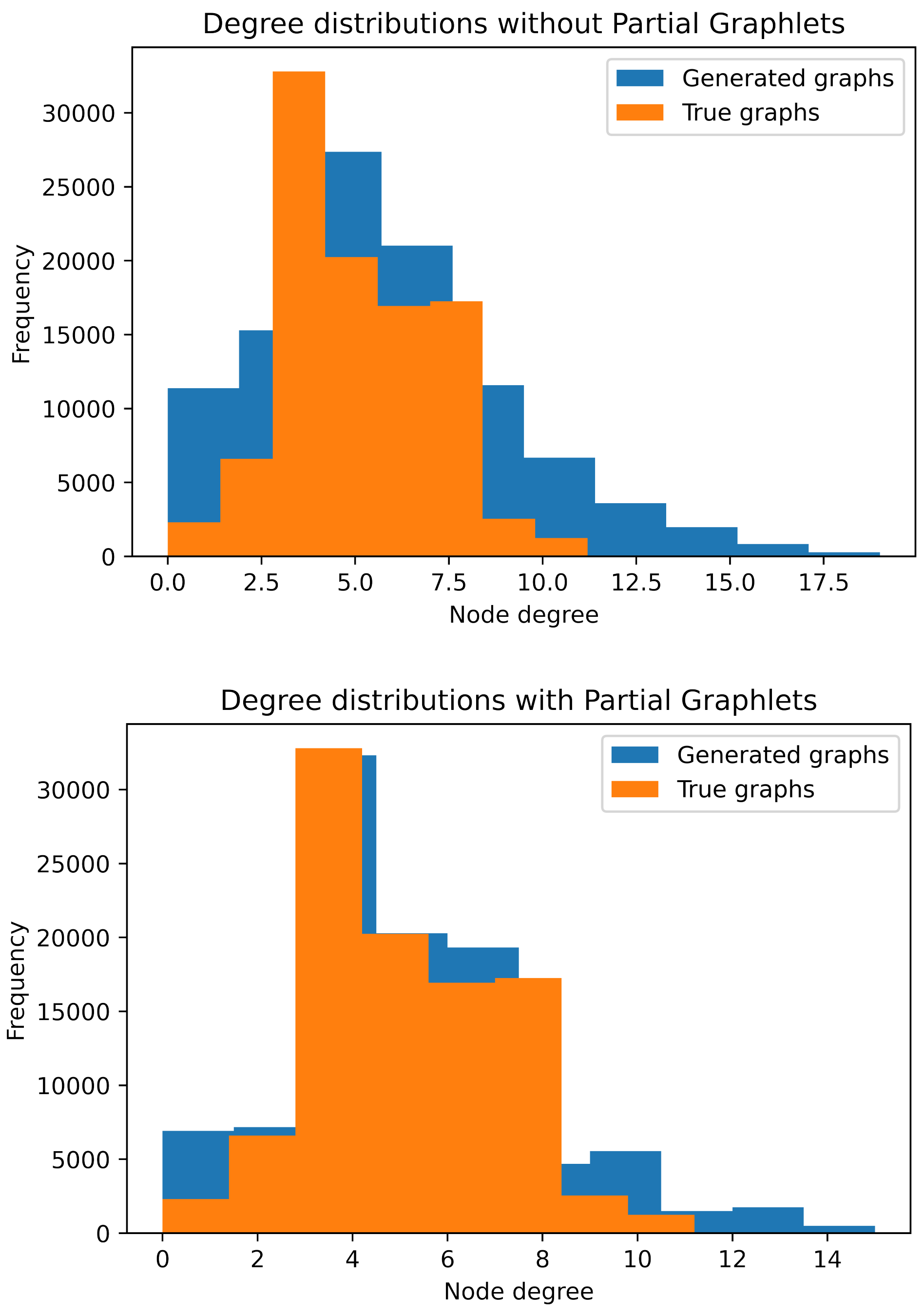


Figure 6-6. Histogram of degrees for nodes randomly sampled from a 4-block SBM (truth) and from GraphMoE trained with and without partial graphlets.

6.4 Implementation for Attributed Graphs

We now discuss the implementation of the second neural net described in Chapter 5 that was designed to create vertex attributes, as well as the algorithm described in the same chapter to create edge attributes.

The model is designed in an end-to-end fashion: The network in the previous section is to be used first to create a graph's edge structure; then the attribute network in this section is to be used, taking the edge structure as input; finally, the edge algorithm is employed, which uses the vertex attributes as input. However, to isolate each component of our analysis, we will instead use real data as the input at each step, i.e. when evaluating the vertex network, for example, we will use the adjacency matrices of real graphs, not those created by the preceding network.

6.4.1 Vertex Attributes Neural Network

As the set of attributed graphlets is unique to each problem and its vertex attributes, we begin by precomputing all possible graphlet combinations for fast retrieval. The neural network has a feed-forward architecture with two hidden layers. The activation function is again Leaky ReLU. A sigmoid function is applied before the final output. The implementation was again programmed using PyTorch, and a similar step-down step size strategy to that of the last section was used. To evaluate the network, the loss function $\lambda(\theta)$ was computed at regular intervals.

6.4.2 Edge Attributes Algorithm

The edge attributes algorithm was implemented using brute force for most datasets. Every edge in the dataset was considered so that conditional probabilities of each edge attribute given the endpoint vertex attributes could be computed. A simple lookup table of probabilities was created for later reference.

6.5 Attributed Datasets

As before, the methods were tested on a number of real and synthetic datasets, derived from other popular generative models and retrieved from the repository at [1]. Details of each dataset are given below and in Table 6-VI.

6.5.1 Complete Graph

We again employ a sanity check. This time, we create one vertex attribute list containing two attributes, and assign every vertex the same attribute. We do the same for edge attributes.

6.5.2 Stochastic Block Model

The 4-block stochastic block model described previously lends itself well to the task of testing an attributed dataset, as the nodes in the graph have already been assigned latent attributes in their communities. We also designate two types of edges: those within communities, and those between communities.

6.5.3 AIDS

We return to the AIDS chemical dataset [4]. Node attributes in this case represent different atomic elements. Edge attributes represent the covalence of bonds between atoms, i.e. the attribute is equal to 1 for a single bond, 2 for a double bond, and 3 for a triple bond.

6.5.4 COX2 and COX2_MD

This dataset which we used previously represents a set of cyclooxygenase-2 inhibitors [5]. Node attributes denote different types of atoms, and edge attributes denote different types of bonds.

6.5.5 BZR and BZR_MD

Another chemical dataset, described in [14]. In this case, however, the edge attributes are numerical and represent distances between atoms; we discretize these distances into bins before attempting to learn the model.

6.5.6 FirstMM_DB

Created in [15], the graphs in this dataset represent common household objects. Nodes represent pieces of the objects (as identified by a human observer) and and edge is present if two pieces are connected. The original authors do not provide much information about the vertex and edge attributes, but it can be inferred that they represent types of object parts and connections. While more information would be desirable, we make use of this dataset to avoid only performing analyses on chemical compounds.

6.5.7 Cuneiform

We additionally include this dataset, which was created by [16]. Each graph represents a single cuneiform sign, with individual characters in that sign each represented by four vertices with four different labels. The four vertices always form a clique. Edges are also allowed between pairs of vertices both belonging to one particular vertex label; other than that, no edges are allowed. Because of the strict rules in place for graph creation in this case, we anticipate significant challenges in learning vertex attributes.

6.6 Results for Attributed Graphs

6.6.1 Vertex Attribute Results

We now present the results of the above implementation for vertex attributes.

The generator again performs well within a relatively small number of training

Name	Source	Num. Graphs	Avg. Nodes	Avg. Edges	m_v	$\max \lvert\Delta_k\rvert$
Complete	N/A	N/A	20	190	1	2
4-block SBM	N/A	N/A	32	244.8	1	4
AIDS	[4]	2000	15.7	16.2	2	37
BZR	[14]	405	35.8	38.4	1	9
COX2	[5]	467	41.2	43.5	1	7
Cuneiform	[16]	267	21.3	44.8	2	4
FirstMM	[15]	41	1377.3	3074.1	1	5

Table 6-VI. Summary information for each dataset used in the vertex attribute learning. The first two are artificial. AIDS, BZR, and COX2 are chemical in nature, while Cuneiform and FirstMM are object-oriented. The final column gives the maximum number of possible attributes on any attribute list for that dataset.

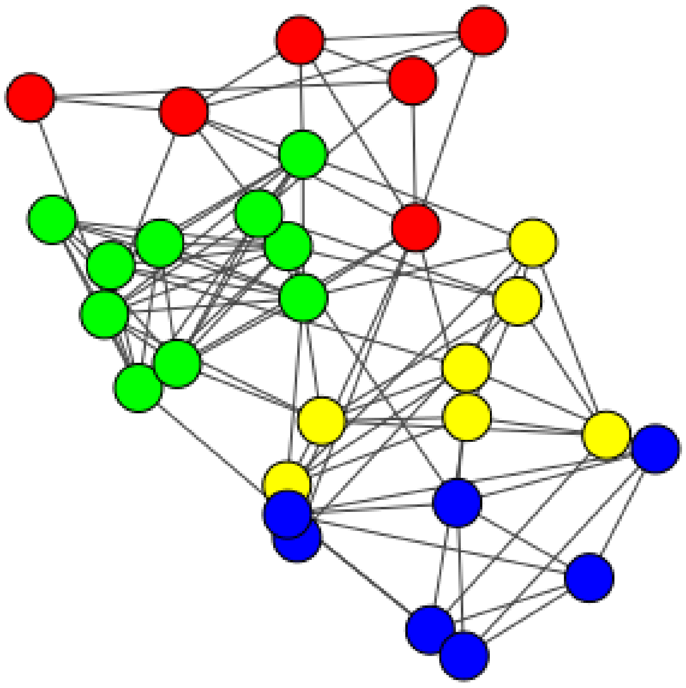
iterations. The network is able to accurately reproduce the distribution of graphlets; training error is given in Table 6-VII. (Since there are a different number of attributed graphlets inherent to each dataset, all training errors were normalized as a percentage of the maximum possible error.)

For the purposes of validation, we took the subgraph of each graph induced by the subset of nodes assigned to the mode of the attributes. These subgraphs, from both the dataset and from the generated graphs, were then fed into a Graph Neural Network (similar to that described in Section 6.3.1.2 above) tasked with distinguishing between them. The results of this analysis were not as favorable as those for the base model, but the generator still performed well given the limited nature of the pairwise graphlets it was provided. Results for this validation test can also be found in Table 6-VII.

Results were poorest on the Cuneiform dataset, where the underlying graph distribution had strict rules about edges in relation to attributes. The neural network appeared to merge several of the attribute classes into one in this case, most likely due to the lack of consistent connectivity characteristics of each class. This is an unfortunate consequence of this method because all attributes are generated at once.

Remark. The neural network that assigns the attribute probabilities to each node appears to display an unsupervised emergent behavior. For some distributions, such as the 4-block SBM, all of the attribute classes are symmetric, and we anticipated that the neural net would have difficulty in distinguishing them. However, when looking at real data, there is symmetry breaking: Because nodes are randomly assigned to each of the four communities / attributes according to a multinomial distribution, each graph will naturally contain some communities that are larger than others.

The expected size of each community in our SBM is 5 nodes (one-fourth of the total number), but the expected size of the maximum community is around 7.4. The network consistently assigned the same attribute to whatever nodes fell into



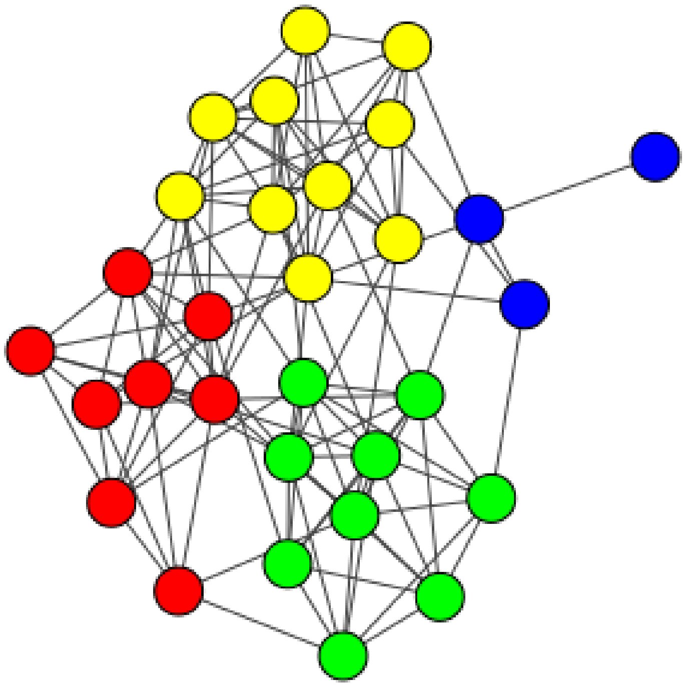


Figure 6-7. (Top) A graph with vertex attributes generated directly from the Stochastic Block Model and (Bottom) a graph whose attributes were created by the neural network model. The specific community labels are not consistent, but the separation of communities is clear. The software which drew the graphs used a clustering algorithm to determine node placement.

the largest community (with similar behavior for the smallest community, etc.) in any particular graph. Naturally, the true attribute value for this community varied between all options from graph to graph, but once the attributes were matched, the neural network was quite consistent.

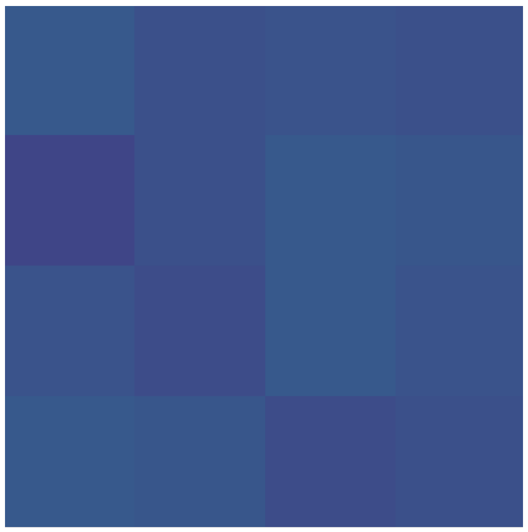
This behavior can be understood as the network 'naming' the community according to one of its basic characteristics- in this case, the community size. The network was able to gain an understanding of the attribute structure in an unsupervised setting.

A pair of confusion matrices is given in Figure 6-8. The first matrix displays the rate at which the network mislabeled nodes belonging to each community without matching the generated attributes to the true attributes. The second confusion matrix, in contrast, shows the misclassification rates once the attributes were registered with each other, allowing the network to use its newfound knowledge effectively.

We note that this behavior did not emerge, and was not necessary, when the attributes were not symmetric. When the attribute classes had distinguishing characteristics to begin with, the network did not need to invent differences between them in order to be consistent.

6.6.2 Edge Attribute Results

In most cases, this simple algorithm to assign attributes to edges works well. The example we had in mind when designing it was the Visual Turing Test of Chapter 3; there, edges represented visual interactions between objects of interest, and the range of possible attributes in the vocabulary were entirely dependent on the attributes of the objects themselves. For instance, it is possible for a 'person' to be 'driving' a 'car', but (probably) not possible for a person to be driving a person. In cases like this, the distribution of edge attributes is well described by the endpoint vertex attributes, and the algorithm performs well. Other cases where edge and vertex attributes are more decoupled are more challenging.



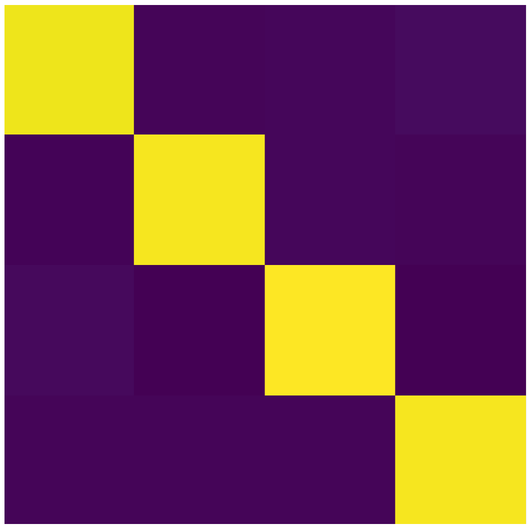


Figure 6-8. Confusion matrices for the attributes assigned by the neural net to vertices from four communities in the Stochastic Block Model. The top matrix shows the result without matching the labels the NN assigned to those in the model. However, once we identify which label is which, the result is the bottom matrix. The NN generally assigned labels based on community size, while the actual model simply had a set ordering for the labels.

Name	$\max \lvert V_a \rvert$	$\max \lvert \hat{V}_a \rvert$	Training	Classifier
Complete	20	20	0.0	0.500
4-block SBM	10.40	10.29	0.031	0.547
AIDS	9.78	10.12	0.064	0.588
BZR	16.72	18.45	0.070	0.728
COX2	18.31	18.24	0.079	0.590
Cuneiform	2.26	6.51	0.244	0.994
FirstMM	9.52	10.08	0.121	0.701

Table 6-VII. Results for the vertex attribute generator. The second and third column give information about the size of the subgraph induced by looking only at the vertices labeled with the most common label, respectively in the data and in the generated graphs. (We use V_a to denote the vertex set induced by attribute set a.) The fourth column shows the normalized total difference between the produced graphlets and the target. The final column lists the performance of a GNN classifier trained to distinguish the induced subgraphs, with optimal value 0.5.

For the purpose of evaluating the method, a test set was separated from a training set for each of the above described samples. The attribute probabilities were calculated from each of the training and test sets. The attribute probabilities from the training sets were applied to the graphs in the test sets, and the distributions of attributes that produced were compared to the test set probabilities. Each discrepancy was weighted by the probability of that combination of attributes in the dataset, and then summed up. The total difference can be found in Table 6-VIII, along with some characteristic information.

The algorithm performed perfectly on the SBM synthetic dataset, where edge attributes encoded same or different communities for the endpoints. Perfect performance was also achieved on the Cuneiform dataset, where edge attributes show whether two symbols are part of the same character or not. On the chemical datasets, where the bond covalence is entirely dependent on the atomic numbers of the pair of vertices in question, good performance was observed. The worst performance was on the AIDS dataset, which had the largest number of vertex attributes possible (and thus had the largest number of rare combinations of attributes).

Name	Source	Num. Graphs	Avg. Nodes	Avg. Edges	m_e	$\max\lvert\Gamma_k\rvert$	Test
4-block SBM	N/A	N/A	32	244.8	1	2	0.0
AIDS	[4]	2000	15.69	16.2	1	4	1.2e-2
BZR_MD	[14]	405	35.75	38.4	1	4	5.2e-3
COX2_MD	[5]	303	26.28	43.5	1	4	4.4e-3
Cuneiform	[16]	267	21.27	44.8	2	2	0.0

Table 6-VIII. Characteristic information for each dataset used for testing the edge attribute algorithm, as well as the result of the test in the final column. The value given is the percentage observed of the largest possible error that could be made for that dataset, weighted by probability. For example, the BZR dataset has 9 vertex attributes and 4 edge attributes, so there are 144 possible combinations of vertex and edge attributes.

References

1. Kersting, K., Kriege, N. M., Morris, C., Mutzel, P. & Neumann, M. *Benchmark Data Sets for Graph Kernels* 2016.
2. You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models* en. in *International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, July 2018), 5708–5717.
3. Athreya, A. *et al.* Statistical Inference on Random Dot Product Graphs: a Survey. *Journal of Machine Learning Research* **18,** 1–92 (2018).
4. Riesen, K. & Bunke, H. *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning* in *Structural, Syntactic, and Statistical Pattern Recognition* (eds da Vitoria Lobo, N. *et al.*) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), 287–297.
5. Kriege, N. & Mutzel, P. *Subgraph Matching Kernels for Attributed Graphs* 2012. arXiv: `1206.6483 [cs.LG]`.
6. Pan, S., Wu, J., Zhu, X., Long, G. & Zhang, C. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Transactions on Cybernetics* **47,** 744–758 (2017).
7. Priebe, C. E. *et al.* On a two-truths phenomenon in spectral graph clustering. *Proceedings of the National Academy of Sciences* **116,** 5995–6000 (2019).
8. Lawrence, R. *et al.* A low-resource reliable pipeline to democratize multi-modal connectome estimation and analysis. *bioRxiv* (2021).
9. Desikan, R. S. *et al.* An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. *Neuroimage* **31,** 968–980 (2006).
10. Yanardag, P. & Vishwanathan, S. in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 1365–1374 (Association for Computing Machinery, New York, NY, USA, 2015).
11. *AIDs antiviral screen (2004)*
12. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* **20,** 61–80 (2009).
13. Zhou, J. *et al.* *Graph Neural Networks: A Review of Methods and Applications* 2019. arXiv: `1812.08434 [cs.LG]`.

14. Sutherland, J. J., O'Brien, L. A. & Weaver, D. F. Spline-fitting with a genetic algorithm: a method for developing classification structure-activity relationships. en. *J Chem Inf Comput Sci* **43,** 1906–1915 (Nov. 2003).

15. Neumann, M., Moreno, P., Antanas, L., Garnett, R. & Kersting, K. *Graph kernels for object category prediction in task-dependent robot grasping* eng. 2013.

16. Kriege, N. M., Fey, M., Fisseler, D., Mutzel, P. & Weichert, F. *Recognizing Cuneiform Signs Using Graph Based Methods* in *Proceedings of The International Workshop on Cost-Sensitive Learning* (eds Torgo, L., Matwin, S., Weiss, G., Moniz, N. & Branco, P.) **88** (PMLR, May 2018), 31–44.

Chapter 7

Discussion and Conclusion

Thus far through this dissertation, we have introduced and evaluated a novel method and some variants for learning generative models of random graph distributions. We will now attempt to address the utility of the model and to explore future areas of research that may be performed.

7.1 Discussion of Main Model

Our experimental results report good quality performance on several simulated and real datasets. Our model is able to imitate complicated classes of random graphs with a relatively simple architecture and small amount of computational time. The use of graphlets as main training statistic ensures that the algorithm is able to properly capture distributions of graphs with unlabeled vertices.

There are two potential sources of variation in the graphs produced by the base generator. The first is simply due to the nature of the RDPG; since each edge is generated randomly and independently, a variety of graphs can be produced using a single set of representations put out by the neural net. However, the use of ω, a vector of Gaussian noise, as the input of the neural network creates further variation in the output of the network. Tests of variance were performed on the output of the neural network after being fed multiple inputs to determine to what extent this randomness is actually felt by the produced representations. We find that node embeddings tend to move around the embedding space greatly from run to run, with the variance of the embeddings' positions on average being around 30% of their means. However, clusters of embeddings tend to move together, so that, depending on the kernel used, edge probabilities may remain more consistent despite the motion in response to ω.

Computational costs for this method are quite low. While best performances are achieved after training times on the order of hours, fairly good performance begins to occur after just a few minutes of training. This allows for deeper or more complex neural network architectures to be employed if desired.

Scalability is a desirable quality of generative models for random graphs; many datasets consist of large graphs, and the ability to add a great number of nodes is thus important. The model remains optimized only for small-to-medium graph sizes. As discussed in Chapter 4, increasing the number of nodes directly increases the dimension of the latent variable space. Because of this, the complexity of the model grows with the number of nodes, and computation power quickly becomes an issue for large graphs.

In certain special cases, we have found ways to circumnavigate these limitations: If the graphs are known to exhibit community structure, or indeed if the nodes fall into a limited number of classes and are homogeneous within each class, then the extension described in Section 4.3.5.2 can be employed to increase the number of nodes arbitrarily.

The method's performance on disparate datasets is quite consistent. The graphs of the COX2_MD dataset are nearly complete and node-homogeneous; in contrast, the IMDb Ego dataset is much sparser and has a central node. Despite these fundamental structural differences, the method's performance on each is similar (though the training speed for the COX dataset was much faster). The method appears robust enough to handle a variety of random graph distributions.

Our validation of our model was limited to results from two classifers, both of which were based on neural networks. While the ability to fool a neural network classifier is a good marker for many applications, we may be able to gain new understanding of our model if a statistical test could be designed to evaluate it. Tang et. al. have invented a hypothesis testing scheme to determine if two RDPGs have the same latent positions [1]; however, the test does not immediately generalize to kernels other than the Euclidean inner product, so further work would be required to apply such a test to our problem.

It remains unclear to what extent information contained in sets of graphlets is redundant. We see improved performance as the size of graphlets used to train is increased, and it seems obvious that the smallest graphlets alone do not contain enough information to completely specify a random graph distribution. The problem of reconstructing a single graph knowing only its graphlets of size $n-1$ is an open problem in graph theory, often called the Graph Reconstruction Conjecture [2]. For our purposes, using graphlets of only size 5 and smaller is enough to fool discriminators (in particular, one that uses graphlets of size 6), so it is unclear how much information is added each time the size of graphlets is increased.

This method has potential to be applied in various fields of research where limited graph data is available. When a large amount of data is needed but acquiring it is cost-prohibitive, the method can be used to simulate additional data. Conditional distributions based on partially observed graphs can also be simulated empirically - we will discuss this in the context of the Visual Turing Test shortly. Because of its low computational cost and high training speed, the method can easily be applied in computational vision, computational medicine, the study of social networks, or a number of other fields.

7.2 Discussion of Attributed Models

The attributed pieces of the method have not been as thoroughly tested as the main generator, but still have shown good performance as measured by a few metrics. For the vertex attribute neural network, we were usually able to get the training error to within a range comparable to that of the main generator. Our test involving the subgraph induced by the largest class of vertex attributes does not by any means fully characterize the graph distribution, but it provides a convenient way to reuse the graph neural networks we utilized to test the main generator.

The main criticism of our models for vertex and edge attributes is that in both cases, we have restricted our learning to statistics involving only pairwise interactions between nodes. In the vertex case, we only measure graphlets of order 2, and in the edge case, we consider only the endpoints of each edge to determine its attribute distribution. It is clear that this is enough to capture some types of random graphs, such as the stochastic block models we used for testing. However, more complicated classes of graphs may not be described well by these limited statistics. Further testing would be advantageous to determine the size of the gap between models of first order and higher order models.

Our approach to edge attributes is somewhat ad hoc. We add edge attributes to our model by way only of a set of probabilities captured from the dataset based on the vertex attributes at the endpoints. This model does not have much sophistication, and notably does not directly allow the edge attributes to be dependent. A more advanced approach might be to apply our vertex-attribute algorithm to the dual graph, but we have not studied how the dual might behave under our model. However, for the purposes of our applications, our method was sufficient.

Our model is specifically equipped to handle categorical vertex and edge attributes. However, many scenarios, including some aspects of the Visual Turing Test, involve numerical attributes for the vertices and edges. One dataset we used for testing included these attributes, but we circumvented the issue by discretizing the range of possible attribute values. Because graphlets are fundamentally discrete objects, it is not immediately clear how to extend our model to numerical attributes. Doing so, however, would broaden its applicability to practically every class of simple random graphs.

7.3 Using the Model for Statistical Inference

The utility of a generative model can be measured by its applicability to further problems of statistical inference. Oftentimes, a generative model is employed because a dataset is incomplete, contains only partial information about a distribution, or makes it difficult to see that information, and it is desirable to discover the remaining information to draw conclusions.

We have presented our model in the context of the Visual Turing Test, for which the task to be tackled by the generative model is clear: The VTT requires sampling from a conditional distribution, where the space on which we are conditioning grows successively more complex. In this case, the approach to the task is also clear: Once the model has been trained, it is extremely cheap to generate a massive number of graphs, requiring a number of operations that is only polynomial in the number of graphs and their size. From there, one can look at the conditional distribution by simply removing the graphs from the sample that do not match the condition; as long as the size of the conditional space is not too small relative to the overall sample space, this can be done quite easily.

With this in mind, the problem of generating unbiased questions for the Visual Turing Test is essentially solved. If a '20 questions' approach is used, we might need to generate only $2^{20+8} = 3e8$ graphs, which is downright easy.

More generally, this generative model is well-equipped to handle any conditioning task for which there exists an oracle to quickly determine whether or not a given graph satisfies the conditions. This brute-force approach of generating numerous graphs and then cutting the sample down to only those that satisfy the condition only causes problems if the conditions occur too rarely. The capability of the model to answer difficult questions is limited only by computational power.

The model does not, however, provide an automatic path to statistical inference. It only implicitly models the distribution of random graphs; that is, it allows one to sample from the distribution, but does not express the distribution in any closed form. It remains up to the statistician to glean the needed information from the graphs produced, which may be equally difficult to doing so from the original sample used to train the model in the first place.

7.4 Directions for Further Work

The model is functionally complete; however, there are areas of research that could be pursued to improve both training and performance.

Our model is trained using the simplest form of stochastic gradient descent; however, there is a plethora of research into optimizing the training of neural networks (including many papers specific to graph data). Two illustrating examples are [3], which introduces a label propagation technique for training neural networks on graph data, and [4], which shows an adversarial method for a similar task.

Beyond these sophisticated modern techniques, there remain ideas in optimization that are now considered somewhat classical in machine learning that we have not yet applied. There are well known algorithms for training neural networks, such as Adam [5]. Our architecture is also as simple as it can be, being nothing more than a very basic feed-forward network. We have not experimented with other architectures, but it would not be surprising to see improvements in performance or training if a more complex architecture was used.

It is clear that there remain limitations in how well the model approximates random graph distributions. Some of those limitations cannot be overcome; low-dimensional latent position graphs are a restrictive parameter space and cannot capture all random graph distributions. Any discrepancies between the model class and the true random graph distribution is, quite simply, modeling bias, and this bias can only be mitigated by increasing the dimension of the latent positions. It *is* true that any edge-independent random graph distribution can be expressed as a latent position graph of sufficiently high dimension: One only need to include a number of dimensions larger than the number of possible edges, and any distribution can be captured. However, there are plenty of random graph distributions that cannot be expressed as a latent position graph at all; for instance, $\mathbb{G}$ could be a distribution on 3 nodes such that the only adjacency matrices with nonzero probability measure are

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ or } \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

This type of strict codependence between edges cannot be expressed by a valid kernel on latent positions.

There is plenty of ground to gain outside of modeling bias, however. As seen in Section 6.3, the model is inferior to some competing methods in certain metrics such as clustering and orbital coefficients. There is also a degree of inability to always fool discriminator networks, as seen in Section 6.3.1.2. Certain graph distributions appear harder to learn than others, and it is not apparent for what reason the model may perform well on some datasets and poorly on others.

Furthermore, there is a fair level of artistry involved in training the networks. Hyperparameters such as layer width, as well the kernel involved in the final step of the main model, have to be chosen carefully, as many failed attempts at training made apparent. A better understanding of the behavior of the model relative to these selections would remove much of the guesswork and improve the consistency of the model.

7.5 Conclusion

Our work accomplishes the task of learning a generative model for an unknown random graph distribution from a small sample. It is readily employable for a variety of applications, including the Visual Turing Test by which it was inspired.

The model is robust enough to handle a variety of applications, with demonstrated results on datasets from chemistry, neuroscience, medicine, and social networking. It is equipped with numerous extensions designed to address a plethora of different problem settings. It has proven itself against modern methods of validation.

Graphlets have been shown to be powerful statistics, capable of capturing a broad range of flavors of graph distributions. We show their effectiveness for training machine learning techniques and as characterizing features for distributions as a whole. We believe they have great potential as tools in both graph inference and graph learning.

Our model and our use of graphlets has opened a window into still-new world of random graphs. A thorough understanding of what we have and why it works may lead to greater insight into the burgeoning field of random graph study. As our model is employed in the future, we will perhaps move towards new results and discoveries that have not yet become visible.

References

1. Tang, M., Athreya, A., Sussman, D. L., Lyzinski, V. & Priebe, C. E. *A non-parametric two-sample hypothesis testing problem for random dot product graphs* 2014.
2. Bondy, J. A. & Hemminger, R. L. Graph reconstruction—a survey. *Journal of Graph Theory* **1,** 227–268 (1977).
3. Bui, T. D., Ravi, S. & Ramavajjala, V. *Neural Graph Learning: Training Neural Networks Using Graphs* in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (Association for Computing Machinery, Marina Del Rey, CA, USA, 2018), 64–71.
4. Zügner, D., Akbarnejad, A. & Günnemann, S. *Adversarial Attacks on Neural Networks for Graph Data* in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Association for Computing Machinery, London, United Kingdom, 2018), 2847–2856.
5. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014.

This page intentionally left blank.

All References

1. Loprinzo, V. & Younes, L. *A generative neural network model for random dot product graphs* 2022.
2. Holland, P. W., Laskey, K. B. & Leinhardt, S. Stochastic blockmodels: First steps. *Social Networks* **5,** 109–137 (1983).
3. Fiduccia, C. M., Scheinerman, E. R., Trenk, A. & Zito, J. S. Dot product representations of graphs. *Discrete Mathematics* **181,** 113–138 (1998).
4. Athreya, A. *et al.* Statistical Inference on Random Dot Product Graphs: a Survey. *Journal of Machine Learning Research* **18,** 1–92 (2018).
5. Gilbert, E. N. Random plane networks. *Journal of the society for industrial and applied mathematics* **9,** 533–543 (1961).
6. Erdös, P. & Rényi, A. On Random Graphs I. *Publicationes Mathematicae Debrecen* **6,** 290 (1959).
7. Erdos, P. & Renyi, A. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.* **5,** 17–61 (1960).
8. May, W. D. & Wierman, J. C. Using symmetry to improve percolation threshold bounds. *Combinatorics, Probability and Computing* **14,** 549–566 (2005).
9. Parviainen, R. Estimation of bond percolation thresholds on the Archimedean lattices. *arXiv preprint arXiv:0704.2098* (2007).
10. Frank, O. & Strauss, D. Markov Graphs. *Journal of the American Statistical Association* **81,** 832–842 (1986).
11. Robins, G., Pattison, P., Kalish, Y. & Lusher, D. An introduction to exponential random graph (p*) models for social networks. *Social Networks* **29.** Special Section: Advances in Exponential Random Graph (p*) Models, 173–191 (2007).
12. Dorogovtsev, S., Mendes, J. F. & Samukhin, A. Size-dependent degree distribution of a scale-free growing network. *Physical review. E, Statistical, nonlinear, and soft matter physics* **63,** 062101 (July 2001).
13. Holme, P. Rare and everywhere: Perspectives on scale-free networks. *Nature Communications* **10,** 1016 (Mar. 2019).
14. Bickel, P., Choi, D., Chang, X. & Zhang, H. Asymptotic normality of maximum likelihood and its variational approximation for stochastic blockmodels. *The Annals of Statistics* **41,** 1922–1943 (2013).

15. Livi, L. & Rizzi, A. The graph matching problem. *Pattern Analysis and Applications* **16,** 253–283 (2013).

16. Peixoto, T. P. Entropy of stochastic blockmodel ensembles. *Physical Review E* **85,** 056122 (2012).

17. Paul, S. & Chen, Y. A random effects stochastic block model for joint community detection in multiple networks with applications to neuroimaging. *The Annals of Applied Statistics* **14,** 993–1029 (2020).

18. Scheinerman, E. R. & Tucker, K. Modeling graphs using dot product representations. *Computational Statistics* **25,** 1–16 (Mar. 2010).

19. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2,** 359–366 (1989).

20. Scarselli, F. & Tsoi, A. C. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks* **11,** 15–37 (1998).

21. Hamilton, W. L. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **14,** 1–159.

22. Liao, R. *et al. Efficient Graph Generation with Graph Recurrent Attention Networks* 2019.

23. De Cao, N. & Kipf, T. MolGAN: An implicit generative model for small molecular graphs (2018).

24. Bojchevski, A., Shchur, O., Zügner, D. & Günnemann, S. NetGAN: Generating Graphs via Random Walks (2018).

25. Malinowski, M. & Fritz, M. *Towards a Visual Turing Challenge* 2014.

26. Qi, H., Wu, T., Lee, M.-W. & Zhu, S.-C. *A Restricted Visual Turing Test for Deep Scene and Event Understanding* 2015.

27. Defferrard, M., Bresson, X. & Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *CoRR.* arXiv: 1606.09375 (2016).

28. Kipf, T. N. & Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks* 2016.

29. Zhang, S., Tong, H., Xu, J. & Maciejewski, R. Graph convolutional networks: a comprehensive review. *Computational Social Networks* **6** (Nov. 2019).

30. Graham, R. & Hell, P. On the History of the Minimum Spanning Tree Problem. *Annals of the History of Computing* **7,** 43–57 (1985).

31. Bollobás, B. The chromatic number of random graphs. *Combinatorica* **8,** 49–55 (Mar. 1988).

32. Micali, S. & Vazirani, V. V. *An O (v| v| c| E|) algoithm for finding maximum matching in general graphs* in *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)* (1980), 17–27.

33. Von Luxburg, U. A Tutorial on Spectral Clustering (2007).

34. Micheli, A. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* **20.** Cited by: 222, 498–511 (2009).

35. Daudin, J.-J., Picard, F. & Robin, S. A mixture model for random graphs. *Statistics and Computing* **18,** 173–183 (June 2008).

36. Dai, H., Nazi, A., Li, Y., Dai, B. & Schuurmans, D. *Scalable Deep Generative Modeling for Sparse Graphs* in *Proceedings of the 37th International Conference on Machine Learning* (eds III, H. D. & Singh, A.) **119** (PMLR, 2020), 2302–2312.

37. Simonovsky, M. & Komodakis, N. *GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders* 2018.

38. Grover, A., Zweig, A. & Ermon, S. *Graphite: Iterative Generative Modeling of Graphs* in *Proceedings of the 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) **97** (PMLR, June 2019), 2434–2444.

39. Beer, E., Fill, J. A., Janson, S. & Scheinerman, E. R. *On vertex, edge, and vertex-edge random graphs* 2008.

40. Agterberg, J., Tang, M. & Priebe, C. E. *On Two Distinct Sources of Nonidentifiability in Latent Position Random Graph Models* 2020.

41. Young, S. & Scheinerman, E. *Random Dot Product Graph Models for Social Networks* in (Nov. 2007), 138–149.

42. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K. & Borgwardt, K. *Efficient graphlet kernels for large graph comparison* in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (eds van Dyk, D. & Welling, M.) **5** (PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009), 488–495.

43. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* **20,** 61–80 (2009).

44. Zhou, J. *et al. Graph Neural Networks: A Review of Methods and Applications* 2019. arXiv: 1812.08434 [cs.LG].

45. Bickel, P. J., Chen, A. & Levina, E. The method of moments and degree distributions for network models. *The Annals of Statistics* **39.** arXiv: 1202.5101 (Oct. 2011).

46. Maugis, P.-A. G., Priebe, C. E., Olhede, S. C. & Wolfe, P. J. Statistical inference for network samples using subgraph counts. *Journal of Computational and Graphical Statistics* **29.** arXiv: 1701.00505, 455–465 (July 2020).

47. Diaconis, P. & Janson, S. Graph limits and exchangeable random graphs. *arXiv:0712.2749 [math].* arXiv: 0712.2749 (Dec. 2007).

48. Aldous, D. J. Representations for partially exchangeable arrays of random variables. en. *Journal of Multivariate Analysis* **11,** 581–598 (Dec. 1981).

49. Ruiz, L., Chamon, L. & Ribeiro, A. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems* **33** (2020).

50. You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models* en. in *International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, July 2018), 5708–5717.

51. Goodfellow, I. *et al. Generative adversarial nets* in *Advances in neural information processing systems* (2014), 2672–2680.

52. Arjovsky, M., Chintala, S. & Bottou, L. *Wasserstein Generative Adversarial Networks* in *Proceedings of the 34th International Conference on Machine Learning - Volume 70* event-place: Sydney, NSW, Australia (JMLR.org, 2017), 214–223.

53. De Cao, N. & Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *arXiv:1805.11973 [cs, stat].* arXiv: 1805.11973 (May 2018).

54. Hallonquist, N., German, D. & Younes, L. *Graph Discovery for Visual Test Generation* in *2020 25th International Conference on Pattern Recognition (ICPR)* ISSN: 1051-4651 (Jan. 2021), 7500–7507.

55. Erdős, P. & Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* **5,** 17–60 (1960).

56. Lovász, L. *Large networks and graph limits* (American Mathematical Soc., 2012).

57. Holland, P. W., Laskey, K. B. & Leinhardt, S. Stochastic blockmodels: First steps. en. *Social Networks* **5,** 109–137 (June 1983).

58. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. en (Dec. 2013).

59. Geman, D., Geman, S., Hallonquist, N. & Younes, L. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences* **112,** 3618–3623 (2015).

60. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K. & Borgwardt, K. *Efficient graphlet kernels for large graph comparison* in *Artificial Intelligence and Statistics* (2009), 488–495.

61. Kang, R. J. & Müller, T. Sphere and dot product representations of graphs. *Discrete & Computational Geometry* **47,** 548–568 (2012).

62. Wu, Z. *et al.* A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

63. Sperduti, A. & Starita, A. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* **8,** 714–735 (1997).

64. Sussman, D. L., Tang, M., Fishkind, D. E. & Priebe, C. E. A Consistent Adjacency Spectral Embedding for Stochastic Blockmodel Graphs. *Journal of the American Statistical Association* **107,** 1119–1128 (2012).

65. Zhang, M., Cui, Z., Neumann, M. & Chen, Y. *An end-to-end deep learning architecture for graph classification* in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

66. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks* **20,** 61–80 (2008).

67. Aronszajn, N. Theory of Reproducing Kernels. *Trans. Am. Math. Soc.* **68,** 337–404 (1950).

68. Bressan, M., Chierichetti, F., Kumar, R., Leucci, S. & Panconesi, A. *Counting graphlets: Space vs time* in *Proceedings of the tenth ACM international conference on web search and data mining* (2017), 557–566.

69. Zhang, L. *et al.* Discovering discriminative graphlets for aerial image categories recognition. *IEEE Transactions on Image Processing* **22.** Publisher: IEEE, 5071–5084 (2013).

70. Janssen, J., Hurshman, M. & Kalyaniwalla, N. Model selection for social networks using graphlets. *Internet Mathematics* **8.** Publisher: Taylor & Francis, 338–363 (2012).

71. Bhuiyan, M. A., Rahman, M., Rahman, M. & Al Hasan, M. *Guise: Uniform sampling of graphlets for large graph analysis* in *2012 IEEE 12th International Conference on Data Mining* (IEEE, 2012), 91–100.

72. Jin, J., Ke, Z. & Luo, S. *Network global testing by counting graphlets* in *International Conference on Machine Learning* (PMLR, 2018), 2333–2341.

73. Doria-Belenguer, S., Youssef, M. K., Böttcher, R., Malod-Dognin, N. & Pržulj, N. Probabilistic graphlets capture biological function in probabilistic molecular networks. *Bioinformatics* **36.** Publisher: Oxford University Press, i804–i812 (2020).

74. Ribeiro, P., Paredes, P., Silva, M. E., Aparicio, D. & Silva, F. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Computing Surveys (CSUR)* **54.** Publisher: ACM New York, NY, USA, 1–36 (2021).

75. Wang, P. *et al.* MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* **30.** Publisher: IEEE, 73–86 (2017).

76. Zhang, L., Bian, W., Song, M., Tao, D. & Liu, X. *Integrating local features into discriminative graphlets for scene classification* in *International Conference on Neural Information Processing* (Springer, 2011), 657–666.

77. Rossi, R. A., Zhou, R. & Ahmed, N. K. Estimation of graphlet counts in massive networks. *IEEE transactions on neural networks and learning systems* **30.** Publisher: IEEE, 44–57 (2018).

78. Riesen, K. & Bunke, H. *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning* in *Structural, Syntactic, and Statistical Pattern Recognition* (eds da Vitoria Lobo, N. *et al.*) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), 287–297.

79. *AIDs antiviral screen (2004)*

80. Kriege, N. & Mutzel, P. *Subgraph Matching Kernels for Attributed Graphs* 2012. arXiv: 1206.6483 [cs.LG].

81. Sutherland, J. J., O'Brien, L. A. & Weaver, D. F. Spline-fitting with a genetic algorithm: a method for developing classification structure-activity relationships. en. *J Chem Inf Comput Sci* **43,** 1906–1915 (Nov. 2003).

82. Yanardag, P. & Vishwanathan, S. in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 1365–1374 (Association for Computing Machinery, New York, NY, USA, 2015).

83. Bao, J., Chen, D., Wen, F., Li, H. & Hua, G. *CVAE-GAN: Fine-Grained Image Generation Through Asymmetric Training* in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2017).

84. Pan, S., Wu, J., Zhu, X., Long, G. & Zhang, C. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Transactions on Cybernetics* **47,** 744–758 (2017).

85. Kersting, K., Kriege, N. M., Morris, C., Mutzel, P. & Neumann, M. *Benchmark Data Sets for Graph Kernels* 2016.

86. Priebe, C. E. *et al.* On a two-truths phenomenon in spectral graph clustering. *Proceedings of the National Academy of Sciences* **116,** 5995–6000 (2019).

87. Lawrence, R. *et al.* A low-resource reliable pipeline to democratize multi-modal connectome estimation and analysis. *bioRxiv* (2021).

88. Desikan, R. S. *et al.* An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. *Neuroimage* **31,** 968–980 (2006).

89. Bondy, J. A. & Hemminger, R. L. Graph reconstruction—a survey. *Journal of Graph Theory* **1,** 227–268 (1977).

90. Zügner, D., Akbarnejad, A. & Günnemann, S. *Adversarial Attacks on Neural Networks for Graph Data* in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Association for Computing Machinery, London, United Kingdom, 2018), 2847–2856.

91. Bui, T. D., Ravi, S. & Ramavajjala, V. *Neural Graph Learning: Training Neural Networks Using Graphs* in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (Association for Computing Machinery, Marina Del Rey, CA, USA, 2018), 64–71.

92. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014.

93. Trinajstic, N. *Chemical graph theory* (CRC press, 2018).

94. Scott, J. *What is social network analysis?* (Bloomsbury Academic, 2012).

95. Leighton, T. & Rao, S. *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms* tech. rep. (Massachusetts Inst. of Tech. Cambridge Microsystems Resesarch Center, 1989).

96. Schwartz, B. L. Possible Winners in Partially Completed Tournaments. *SIAM Review* **8,** 302–308 (1966).

97. Kriege, N. M., Fey, M., Fisseler, D., Mutzel, P. & Weichert, F. *Recognizing Cuneiform Signs Using Graph Based Methods* in *Proceedings of The International Workshop on Cost-Sensitive Learning* (eds Torgo, L., Matwin, S., Weiss, G., Moniz, N. & Branco, P.) **88** (PMLR, May 2018), 31–44.

98. Neumann, M., Moreno, P., Antanas, L., Garnett, R. & Kersting, K. *Graph kernels for object category prediction in task-dependent robot grasping* eng. 2013.

99. Tang, M., Athreya, A., Sussman, D. L., Lyzinski, V. & Priebe, C. E. *A non-parametric two-sample hypothesis testing problem for random dot product graphs* 2014.

This page intentionally left blank.

Biographical Sketch

Vittorio Loprinzo grew up in Carmel, NY, and was interested in science and mathematics from a very young age. He completed his undergraduate education at The Johns Hopkins University in Baltimore, MD in 2016, majoring in Physics and Applied Mathematics & Statistics and also completing an MA in Physics during that time. He then enrolled in a PhD program in Applied Mathematics & Statistics at the same institution.

Loprinzo's research interests lie in statistics and statistical learning, but he is happy to explore applications in any field, be that graphical models, the physical sciences, or computational medicine. In the past, he has worked in discrete math, obtaining results in percolation theory, as well as laboratory physics, having contributed to work on the subject of condensed matter. Following the completion of this dissertation, Loprinzo will enter the field of pharmaceutical research as a biostatistician.

Loprinzo has a passion for teaching, having received nominations and awards from JHU for his teaching abilities and having taught several courses as instructor of record, on topics ranging broadly from statistics to musicology. He is also a Fellow of JHU's MINDS Institute and of ΣΠΣ.

Outside of work, Loprinzo enjoys hip-hop dance, tabletop gaming, trivia, and racquetball, all hobbies he discovered while at JHU. He will miss his time and friends at the university dearly.