# Graph Structure Learning: Neural Network Unrollings and Bayesian Approaches

by

Max Wasserman

Submitted in Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Supervised by Professor Gonzalo Mateos

Department of Computer Science

Arts, Sciences and Engineering

Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester

Rochester, New York

2025

*Dedicated to my family*

# Table of Contents

# Biographical Sketch

Max Wasserman received his Bachelors degree in Mechanical Engineering from the University of Pennsylvania in 2015. He subsequently performed robotics research in the GRASP lab at University of Pennsylvania with Dr. Dan Lee before beginning his Ph.D. in Computer Science at the University of Rochester under the supervision of Professor Gonzalo Mateos. His research focuses on graph structure learning and probabilistic methods. In addition to his academic work, Max completed an internship at Apple from April to October 2023.

The following publications stem directly from my doctoral research and are closely related to this thesis:

- **M. Wasserman**, S. Sihag, S., Mateos, G., and A. Ribeiro. "Learning Graph Structure from Convolutional Mixtures." *Transactions on Machine Learning Research* (2023).

- **M. Wasserman** and Gonzalo Mateos. "pyGSL: A Graph Structure Learning Toolkit." *NeurIPS 2022 Workshop: New Frontiers in Graph Learning.*

- **M. Wasserman**, and Gonzalo Mateos. "Graph Structure Learning with Interpretable Bayesian Neural Networks." *Transactions on Machine Learning Research* (2024).

- **M. Wasserman** and Gonzalo Mateos. "Stabilizing the Kumaraswamy Distribution." *arXiv preprint* arXiv:2410.00660 (2024).

# Abstract

Graphs form a fundamental framework in machine learning, data science, and network science, enabling the modeling of complex systems by capturing intricate relationships across diverse domains, from social networks to biological systems; see e.g., [1, 2]. For example, in computational biology, accurately modeling graph structures can unravel gene regulatory pathways, advancing treatments for genetic disorders [3]. Similarly, in finance, precise graph representations of financial networks can facilitate risk assessment and promote market stability [4]. Despite their importance, the relevant graph structures are often unknown or only partially available, necessitating their inference from data — a process referred to as Graph Structure Learning (GSL). This thesis advances GSL by addressing key challenges in modeling and learning graph structures, introducing novel neural network unrolling techniques and Bayesian approaches to enhance interpretability, efficiency, and uncertainty quantification.

Graph Signal Processing (GSP) is essential for meeting the increasing demand for processing information over networks, particularly in applications

such as supervised classification. However, the effectiveness of GSP in these tasks depends heavily on accurately uncovering the underlying relational structures, which are frequently unavailable and need to be inferred from observed data. We postulate a graph convolutional relationship between the observed and latent graphs, and formulate the graph learning task as a network inverse (deconvolution) problem. In lieu of eigendecomposition-based spectral methods or iterative optimization solutions, we unroll and truncate proximal gradient iterations to produce a parameterized neural network architecture that we call a Graph Deconvolution Network (GDN). GDNs learn a distribution of graphs in a supervised fashion, are inherently inductive, and perform link prediction or edge-weight regression tasks by adapting the loss function.

Conventional methods for GSL typically frame the problem as a convex inverse optimization with a smoothness-promoting objective, relying on iterative algorithms to derive a solution. In supervised settings where graph labels are available, these iterative procedures can be unrolled and truncated into a deep network that is trained end-to-end, as demonstrated in the GDN. This approach results in a parameter-efficient architecture that retains inductive bias from the underlying optimization, making it particularly advantageous in data-limited domains such as medicine, finance, and the natural sciences. However, such domains often require not only point estimates of edge weights but also well-calibrated uncertainty quantification. To address this, we introduce novel iterations with independently interpretable parameters, i.e.,

parameters whose values — independent of other parameters' settings —- proportionally influence characteristics of the estimated graph, such as edge sparsity. After unrolling these iterations, we impose prior distributions over these interpretable parameters, leveraging domain knowledge to construct a Bayesian neural network (BNN) for GSL from smooth signal observations. Parameter efficiency and fast execution allow for high-fidelity posterior estimation via Markov Chain Monte Carlo (MCMC), enabling robust uncertainty quantification on edge predictions. Additionally, informative priors facilitate the application of Bayesian modeling techniques such as prior predictive checks. Empirical evaluations on both synthetic and real datasets validate the model's ability to produce well-calibrated uncertainty estimates, demonstrating its effectiveness in tasks such as identifying economic sector modular structures from S&P500 data and capturing pairwise digit similarities from MNIST images. Ultimately, this framework extends GSL to settings where understanding uncertainty in the inferred graph structure is crucial.

In order to address large-scale Bayesian graph structure learning approaches, we turn to variational inference. Large-scale latent variable models necessitate expressive continuous distributions that allow for efficient sampling and low-variance differentiation, which can be achieved using the reparameterization trick. The Kumaraswamy (KS) distribution is both expressive and amenable to reparameterization due to its simple closed-form inverse CDF. However, its broader adoption has been hindered by unresolved numerical instabilities. We identify and mitigate these instabilities in the log-pdf, CDF,

and inverse CDF, revealing issues in widely used libraries such as PyTorch and TensorFlow. Building on these insights, we introduce scalable latent variable models that enhance exploration-exploitation trade-offs in contextual multi-armed bandits and improve uncertainty quantification for link prediction in graph neural networks. Empirical results demonstrate that these models achieve their best performance when paired with the stabilized KS. Our findings establish the stabilized KS distribution as a fundamental component in scalable variational models for bounded latent variables.

# Contributors and Funding Sources

This work was guided by a dissertation committee comprising Professor Gonzalo Mateos (advisor) from the Department of Electrical and Computer Engineering, Professors Jiebo Luo and Chenliang Xu from the Department of Computer Science, Professor Alejandro Ribeiro from the Department of Electrical and Computer Engineering at the University of Pennsylvania. The graduate study was supported by funding from the University of Rochester and the National Science Foundation under awards CCF-1750428, CCF-1934962, ECCS-2231036.

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The intertwined fields of graph signal processing (GSP) [5–7], graph representation learning [2], and machine learning on graphs [8, 9] have recently emerged with the common goal of extracting actionable information from graph-structured (i.e., relational) data describing networks [1, Ch. 1]. In some real-world applications such as network neuroscience [10], said relational structures may not be explicitly available [11, 12]. Therefore, depending on the end goal the first step may be to recover the latent network topology to reveal patterns in the complex system under study; or, to rather learn graph representations that can facilitate downstream tasks such as classification; see e.g, [13]. Recognizing that many of these graph datasets grow every day in volume and complexity, there is a pressing need to develop scalable GSL algorithms.

The term *graph structure learning* encompasses a broad class of approaches to identify an underlying graph using data. Inference refers to the process of searching for a graph (represented via some graph shift operator) that is

optimal for the task at hand. Data often come in the form of nodal observations (also known as graph signals in the GSP parlance), but partial edge status information is not uncommon [1, Ch. 7]. Optimality notions and constraints are typically driven by statistical priors, physical laws, or explainability goals, all of which translate to models binding the observations to the sought graph. A common probabilistic prior is to model network observations via undirected Gaussian graphical models. In this case, the topology inference problem boils down to graphical model selection [14–16]. Other recent approaches instead assume that graph signals are e.g., stationary and possibly generated by linear network diffusion [17, 18], or, smooth with respect to the graph (i.e., they are sparse in the graph spectral domain) [19–23]. Please refer to [11, 12, 24] for recent surveys of graph structure learning advances.

## 1.1 Thesis outline and contributions

The central theme of this thesis revolves around the development of graph learning algorithms, which is structured as follows:

**Chapter 2:** This chapter provides a comprehensive overview of GSP, beginning with the fundamental concepts of graphs, graph signals, and shift operators. We then introduce the graph Fourier transform and explore the concept of signal smoothness over graphs. This discussion leads to a formal presentation of the problem of topology identification from smooth signals, which serves as a foundational concept for the subsequent chapters of this

thesis. We further include discussion on algorithm unrolling, a technique used frequently in subsequent chapters, and latent variable models, which treat a subset of variables in a probabilistic model as unobserved and allow explicit handling of uncertainty.

**Chapter 3:** Initially, we address the challenge of graph structure learning under a graph convolutional relationship between the observed and latent graphs, and formulate the graph learning task as a network inverse (deconvolution) problem. In contrast to popular existing approaches which rely on eigendecomposition-based spectral methods or iterative optimization solutions, we unroll and truncate proximal gradient iterations to arrive at a parameterized neural network architecture that we call a Graph Deconvolution Network (GDN). GDNs can learn a distribution of graphs in a supervised fashion, perform link prediction or edge-weight regression tasks by adapting the loss function, and they are inherently inductive. We corroborate GDN's superior graph recovery performance and its generalization to larger graphs using synthetic data in supervised settings. Furthermore, we demonstrate the robustness and representation power of GDNs on real world neuroimaging and social network datasets. The materials presented in this chapter have been published in a journal paper [25].

**Chapter 4:** We study the problem of graph structure learning under a signal smoothness prior on the nodal observations. Initially, we simplify and reparameterize Dual Proximal Gradient iterations, a fast dual-based proximal gradient algorithm designed to efficiently handle a strongly con-

vex, smoothness-regularized network inverse problem. Doing do eschews convergence guarantees of the original iterations in return for interpretability properties and fewer tunable parameters. We proceed to unroll these iterations to produce the first strict unrolling for graph structure learning from smooth signals, which results in a true neural network. Often settings care equally about uncertainty over edge predictions, not just point estimates. To address such settings, we use these interpretable iterations to incorporate prior knowledge over such graph characteristics by shaping prior distributions over the interpretable network parameters to yield a Bayesian neural network (BNN) capable of graph structure learning from smooth signal observations. Fast execution and parameter efficiency allow for high-fidelity posterior approximation via Markov Chain Monte Carlo (MCMC) and thus uncertainty quantification on edge predictions. Informative priors unlock modeling tools from Bayesian statistics like prior predictive checks. Synthetic and real data experiments corroborate this model's ability to provide well-calibrated estimates of uncertainty, in test cases that include unveiling economic sector modular structure from S&P500 data and recovering pairwise digit similarities from MNIST images. Overall, this framework enables GSL in modest-scale applications where uncertainty on the data structure is paramount. The materials presented in this chapter have been published in a journal paper [26].

**Chapter 5:** A promising direction to address uncertainty quantification on large-scale topology inference tasks is through variational inference. Learning

in such large-scale models require expressive continuous distributions that support efficient sampling and low-variance differentiation, achievable through the reparameterization trick. The Kumaraswamy (KS) distribution is both expressive and supports the reparameterization trick with a simple closed-form inverse CDF. Yet, its adoption remains limited. We identify and resolve numerical instabilities in the log-pdf, CDF, and inverse CDF, exposing issues in libraries like PyTorch and TensorFlow. We then introduce simple and scalable latent variable models to improve exploration-exploitation trade-offs in contextual multi-armed bandits and enhance uncertainty quantification for link prediction with graph neural networks. We find these models to be most performant when paired with the stable KS. Our results support the stabilized KS distribution as a core component in scalable variational models for bounded latent variables. The materials presented in this chapter are under currently under review for publication [27].

## 1.2 Notational conventions

The entries of a matrix $\mathbf{X}$ and a (column) vector $\mathbf{x}$ are denoted by $X_{ij}$ and $x_i$, respectively. Sets are represented by calligraphic capital letters. The notation $^\top$ and $^\dagger$ stand for transpose and pseudo-inverse, respectively; $\mathbf{0}$ and $\mathbf{1}$ refer to the all-zero and all-one vectors; while $\mathbf{I}_N$ denotes the $N \times N$ identity matrix. For a vector $\mathbf{x}$, $\mathrm{diag}(\mathbf{x})$ is a diagonal matrix whose $i$th diagonal entry is $x_i$. The operators $\circ$, $\mathrm{tr}(\cdot)$, and $\mathrm{vec}(\cdot)$ stand for Hadamard (element-wise) product,

matrix trace, and vectorization, respectively. Lastly, $\|\mathbf{X}\|_p$ denotes the $\ell_p$ norm of vec($\mathbf{X}$) and $\|\mathbf{X}\|_F$ refers to the Frobenius norm. To avoid overloading the notation, on occasion $\|\mathbf{x}\|$ is used to denote the $\ell_2$ norm of vector $\mathbf{x}$.

# Chapter 2

# Background

This chapter introduces the foundational concepts and methodologies of graph learning and Graph Signal Processing (GSP), which form the core framework for this thesis. We introduce an important signal model that used in the topology inference approaches explored in this dissertation: the signal smoothness model for undirected graphs. We further introduce the concept of algorithm unrolling, which turn iterative algorithms into neural network blueprints, and latent variable models, which which play a critical role in Bayesian approaches to GSL, allowing for structured priors, uncertainty estimation, and probabilistic inference over graph structures.

## 2.1 Graphs, signals and shift operators

We define a graph signal as $\mathbf{x} = [x_1, \ldots, x_N]^\top \in \mathbb{R}^N$ on a weighted, undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathbf{W})$, where $\mathcal{V} = \{1, \ldots, N\}$ represents a set of $N$ nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges. The signal value at node $i \in \mathcal{V}$ is given

by $x_i \in \mathbb{R}$, while the edge weights are captured by the adjacency matrix $\mathbf{W} \in \mathbb{R}_+^{N \times N}$. The symmetric, non-negative entries $Wij = W_{ji} \in \mathbb{R}_+$ reflect the connection strength or similarity between nodes $i$ and $j$. If there is no edge between nodes $i$ and $j$, i.e., $(i, j) \notin \mathcal{E}$, then $W_{ij} = 0$. Additionally, we assume that the graph has no self-loops, meaning that $W_{ii} = 0$ for all $i \in \mathcal{V}$. To ensure numerical stability, we impose a lower bound on the nodal degrees, defined as $\mathbf{d} := \mathbf{W1}$, requiring that $\mathbf{d} \succeq d_{\min}\mathbf{1}$ (in an entry-wise sense) for some positive threshold $d_{\min} > 0$. If node degrees become arbitrarily small, it is often advisable to apply a threshold and remove weakly connected nodes from $\mathcal{G}$. The framework can also be extended to support complex-valued signals [5].

The adjacency matrix $\mathbf{W}$ represents the connectivity structure of $\mathcal{G}$. More generally, one can define the *graph-shift operator* $\mathbf{S}$, which encodes the sparsity pattern of $\mathcal{G}$ without imposing constraints on the values of its non-zero elements. The shift operator $\mathbf{S} \in \mathbb{R}_+^{N \times N}$ is a symmetric matrix where $S_{ij}$ can be non-zero only if $i = j$ or if $(i, j) \in \mathcal{E}$. Beyond the adjacency matrix $\mathbf{W}$, spectral graph theory provides motivation for alternative choices of $\mathbf{S}$, such as the combinatorial graph Laplacian $\mathbf{L} := \operatorname{diag}(\mathbf{d}) - \mathbf{W}$, as well as its degree-normalized variants. Additional application-specific alternatives have also been explored; see [5] and references therein. Notably, the graph Laplacian $\mathbf{L}$ is particularly important in defining a meaningful and intuitive graph Fourier transform (GFT), which we discuss next.

## 2.2 Graph Fourier transform, graph filtering, and signal smoothness

To introduce the network's spectral basis and define the graph Fourier transform (GFT), we decompose the combinatorial graph Laplacian $\mathbf{L}$, which is symmetric and positive semi-definite, as $\mathbf{L} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$. Here, $\mathbf{\Lambda} := \text{diag}\,(\lambda_1, \ldots, \lambda_N)$ is the diagonal matrix of non-negative eigenvalues, and $\mathbf{V} := [\mathbf{v}_1, \ldots, \mathbf{v}_N]$ is the orthonormal matrix of corresponding eigenvectors. The GFT of $\mathbf{x}$ with respect to $\mathbf{L}$ is the signal

$$\tilde{\mathbf{x}} := \mathbf{V}^\top \mathbf{x}. \tag{2.1}$$

The inverse (i)GFT of $\tilde{\mathbf{x}} := [\tilde{x}_1, \ldots, \tilde{x}_N]^\top$ is given by $\mathbf{x} = \mathbf{V}\tilde{\mathbf{x}} = \sum_{k=1}^{N} \tilde{x}_k \mathbf{v}_k$, which is a proper inverse due to the orthonormality of $\mathbf{V}$. The GFT encodes a notion of signal variability over $\mathcal{G}$ (akin to frequency in Fourier analysis of temporal signals) by synthesizing $\mathbf{x}$ as a sum of orthogonal frequency components $\mathbf{v}_k$ (see the iGFT definition above). The GFT coefficient $\tilde{x}_k$ is the contribution of $\mathbf{v}_k$ to the graph signal $\mathbf{x}$.

For a graph signal $\mathbf{x}$ with GFT coefficients $\tilde{\mathbf{x}}$, filtering can be performed in the frequency domain, analogous to classical signal processing for time-varying signals. Eigenvalues of the Laplacian correspond to graph frequencies and eigenvectors serve as frequency basis. For instance, a low-pass filter can be designed by isolating the lowest $N_L$ eigenvalues and their corresponding eigenvectors [28]. Define a spectral operation $\tilde{\mathbf{x}}_L = \tilde{\mathbf{H}}_L \tilde{\mathbf{x}}$, where $\tilde{\mathbf{H}}_L =$

$\text{diag}(\tilde{\mathbf{h}}_L)$ and $\tilde{h}_{L,n} = \mathbb{I}\{n < N_L\}$ ($\mathbb{I}\{\cdot\}$ is an indicator function). This is equivalent to the following convolution operation in the vertex domain

$$\mathbf{x}_L = \mathbf{V}\tilde{\mathbf{x}}_L = \mathbf{V}\tilde{\mathbf{H}}_L\tilde{\mathbf{x}} = \mathbf{V}\tilde{\mathbf{H}}_L\mathbf{V}^\top\mathbf{x} = \mathbf{H}_L\mathbf{x}, \tag{2.2}$$

where $\mathbf{H}_L = \mathbf{V}\tilde{\mathbf{H}}_L\mathbf{V}^\top$ is the low-pass *graph filter*. In addition to $\mathbf{H}_L$, a graph band-pass filter $\mathbf{H}_M$ and high-pass filter $\mathbf{H}_H$ can also be defined analogously. Then, all graph frequencies are decomposed and assigned to each *graph filter* where $\mathbf{H}_L$ takes the lowest $N_L$ frequencies, $\mathbf{H}_M$ takes the middle $N_M$ frequencies and $\mathbf{H}_H$ takes the highest $N_H$ frequencies, with $N_L+N_M+N_H = N$. Because these filters span all graph frequencies and are mutually exclusive, we can map signals to the spectral domain via the GFT, filter them and use the iGFT to map each frequency component back into the vertex domain. This decomposes the original graph signal into $\mathbf{x} = \mathbf{x}_L + \mathbf{x}_M + \mathbf{x}_H$. This has the affect of increasing the resolution of the signal by partitioning it into components $\mathbf{x}_L, \mathbf{x}_M, \mathbf{x}_H$ that exhibit low, medium and high variability with respect to $\mathcal{G}$.

To expand on the notion of frequency for graph signals, consider the total variation (or *Dirichlet energy*) of $\mathbf{x}$ with respect to the combinatorial graph Laplacian $\mathbf{L}$ defined as

$$\text{TV}(\mathbf{x}) := \mathbf{x}^\top\mathbf{L}\mathbf{x} = \frac{1}{2}\sum_{i\neq j} W_{ij}\left(x_i - x_j\right)^2 = \sum_{k=1}^{N}\lambda_k\tilde{x}_k^2. \tag{2.3}$$

This quadratic form equation 2.3 acts as a measure of smoothness, because it

effectively quantifies how much the graph signal $\mathbf{x}$ changes with respect to $\mathcal{G}$'s topology. Evaluating the total variation of eigenvector $\mathbf{v}_k$ (itself a graph signal), one obtains $\mathrm{TV}(\mathbf{v}_k) = \lambda_k$. Accordingly, the Laplacian eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_N$ can be interpreted as graph frequencies indicating how the eigenvectors (i.e., frequency mode) vary with respect to $\mathcal{G}$ [5].

Smoothness is a foundational property of many real-world network processes; see e.g. [1]. The last equality in equation 2.3 suggests that smooth (i.e., bandlimited) signals admit a sparse representation in the graph spectral domain. Intuitively, they tend to be spanned by only a few Laplacian eiegenvectors associated with small eigenvalues. Exploiting this structure — by means of priors or TV-based regularizers — is at the heart of several graph-based statistical learning tasks including nearest-neighbor prediction (also known as graph smoothing), denoising, semi-supervised learning, and spectral clustering [1, 5]. More pertinent to our graph structure learning problem is to use smoothness as the criterion to construct graphs on which network data admit certain regularity.

### 2.2.1 Network Diffusion Model

Let $\mathbf{y}$ represent a graph signal supported on $\mathcal{G}$, assumed to be produced from an initial state $\mathbf{x}$ through linear network dynamics of the form

$$\mathbf{y} = a_0 \prod_{n=1}^{\infty} (\mathbf{I}_N - a_n \mathbf{S})\mathbf{x} = \sum_{l=0}^{\infty} b_l \mathbf{S}^l \mathbf{x}. \tag{2.4}$$

Linear transformation $\mathbf{S}$ represents one-hop network aggregation (or averaging), so repeated ($l = 1, 2, \ldots$) applications of the GSO in equation 2.4 diffuse $\mathbf{x}$ across $\mathcal{G}$. Accordingly, $\{a_l\}_{l=0}^{\infty}$ and $\{b_l\}_{l=0}^{\infty}$ can be viewed as diffusion coefficients for the multiplicative and additive signal model parametrizations in equation 2.4, respectively. This generative model for $\mathbf{y}$ is simple, yet nonetheless encompasses a broad class of linear network processes, e.g., heat diffusion, average consensus, PageRank, and the DeGroot model of opinion dynamics [29].

The Cayley-Hamilton theorem [30] provides that the infinite series in the right-hand-side of equation 2.4 can always be equivalently reparametrized using *bounded-degree polynomials* of $\mathbf{S}$. Introducing the coefficient vector $\mathbf{h} := [h_0, \ldots, h_{L-1}]^\top$ and the shift-invariant graph filter [5]

$$\mathbf{H} := h_0\mathbf{I}_N + h_1\mathbf{S} + h_2\mathbf{S}^2 + \ldots + h_{L-1}\mathbf{S}^{L-1} = \sum_{l=0}^{L-1} h_l\mathbf{S}^l, \tag{2.5}$$

the signal model equation 2.4 becomes

$$\mathbf{y} = \left(\sum_{l=0}^{L-1} h_l\mathbf{S}^l\right)\mathbf{x} = \mathbf{H}\mathbf{x}, \tag{2.6}$$

for some $\mathbf{h}$ and $L \leq N$. While graph filters are leveraged here as simple generative mechanisms to describe diffusion processes on networks, such convolutional operators play a foundational role in GSP and machine learning on graphs; see e.g., [31] for an excellent tutorial treatment.

## 2.3 Topology identification from smooth signals

Variations of the following graph learning problem are studies in Chapters 3 and 4.

**Problem 1.** *Given a set $\mathcal{X} := \{\mathbf{x}_p\}_{p=1}^P$ of graph signal observations, the goal is to learn an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathbf{W})$ such that the observations in $\mathcal{X}$ are smooth on $\mathcal{G}$.*

Here, we review the method proposed in [19,22] to tackle Problem 1, from which we build on to develop efficient, task-aware graph learning algorithms.

Consider matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_P] \in \mathbb{R}^{N \times P}$, with columns $\mathbf{x}_p$ are the observations in $\mathcal{X}$. The rows, denoted by $\bar{\mathbf{x}}_i^\top \in \mathbb{R}^{1 \times P}$, collect all $P$ measurements at vertex $i$. Define then the nodal Euclidean-distance matrix $\mathbf{E} \in \mathbb{R}_+^{N \times N}$, where $E_{ij} := \|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\|_2^2$, $i, j \in \mathcal{V}$. Using these notions, the signal smoothness measure over $\mathcal{X}$ can be equivalently written as

$$\sum_{p=1}^{P} \mathrm{TV}(\mathbf{x}_p) = \mathrm{trace}(\mathbf{X}^\top \mathbf{L} \mathbf{X}) = \frac{1}{2} \|\mathbf{W} \circ \mathbf{E}\|_1, \tag{2.7}$$

where $\circ$ denotes element-wise product [19]. Smoothness minimization as criterion in Problem 1 has the following intuitive interpretation: when pairwise nodal distances in $\mathbf{E}$ are sampled from a smooth manifold, the learnt topology $\mathbf{W}$ tends to be sparse, preferentially choosing edges $(i, j)$ whose corresponding $E_{ij}$ are smaller [cf. the weighted $\ell_1$-norm in equation 2.7].

Leveraging this link between signal smoothness and edge sparsity, a fairly

general graph-learning framework was put forth in [19]. The idea therein is to solve the following convex inverse problem

$$\min_{\mathbf{W}} \left[ \|\mathbf{W} \circ \mathbf{E}\|_1 \underbrace{-\alpha \mathbf{1}^\top \log(\mathbf{W}\mathbf{1}) + \frac{\beta}{2}\|\mathbf{W}\|_F^2}_{f(\mathbf{W})} \right]$$
$$\text{subject to } \operatorname{diag}(\mathbf{W}) = \mathbf{0},\ W_{ij} = W_{ji} \geq 0,\ i \neq j \tag{2.8}$$

where $\alpha, \beta > 0$ are tunable regularization parameters. Different from [20], the logarithmic barrier on the vertex degrees $\mathbf{d} = \mathbf{W}\mathbf{1}$ excludes the possibility of having (often undesirable) isolated vertices in the estimated graph. Through $\beta$, the Frobenius-norm penalty offers a handle on the graphs' edge sparsity level. Among the parameterized familiy of solutions to equation 2.3, the sparsest graph is obtained when $\beta = 0$. It is important to highlight that the convex objective function $f(\mathbf{W})$ augments the smoothness criterion $\|\mathbf{W} \circ \mathbf{E}\|_1$. Moreover, there exists flexibility in the choice of $f(\mathbf{W})$ beyond the proposal in [19]. Various alternatives for $f(\mathbf{W})$ have been suggested, enabling the recovery of common graph constructions based on the Gaussian kernel [32], accommodation of time-varying graphs [21], or scaling of other related graph learning algorithms [20].

Arguably, the most important takeaways of identity equation 2.7 is computational. It facilitates formulating equation 2.3 as a search over adjacency matrices, and the resulting constraints (null diagonal, symmetry and non-negativity) are separable across the variables $W_{ij}$. This does not hold for

the Laplacian $\mathbf{L}$. Exploiting this favorable structure of equation 2.3, efficient solvers were developed based on PD iterations [19], the PG method [33], or the ADMM [34].

## 2.4 Algorithm Unrolling

Algorithm unrolling, also known as deep unfolding, is a technique that leverages iterative optimization algorithms as blueprints for neural network architectures. The concept, initially introduced in the context of sparse coding by [35], involves mapping iterations of an optimization procedure onto layers of a deep network, effectively truncating an asymptotically convergent algorithm into a trainable, fixed-depth architecture. This approach enables neural networks to approximate solutions with significant computational savings while inheriting desirable properties from the original iterative method. Specifically, unrolled networks tend to be parameter-efficient, allow for fast layer-wise execution, and often generalize well in low-data environments [36]. A key design consideration in unrolling is whether to maintain strict adherence to the original iterative update—preserving its inductive bias and stability—or to introduce learnable modules that replace intermediate operators, increasing expressiveness at the potential expense of interpretability. Similarly, one can choose to either share optimization parameters across layers or decouple them, with the latter offering greater flexibility at the cost of increased model complexity. As a result, unrolled networks strike a balance between efficiency,

expressiveness, and interpretability, making them a compelling alternative to traditional iterative solvers in various applications, including signal processing and graph learning.

## 2.5 Latent Variable Models

**Latent variable modeling with stochastic variational inference (SVI).** Stochastic variational inference (SVI) [37] is the predominant method for training large-scale latent variable models. Consider a probabilistic model of the form $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$ where $\boldsymbol{x} \in \mathbb{R}^M$ is the observation, $\boldsymbol{z} \in \mathbb{R}^D$ is a vector-valued latent variable, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is the likelihood function with parameters $\boldsymbol{\theta}$, and $p(\boldsymbol{z})$ is the prior distribution. Except for special cases, direct maximum likelihood estimation in such models is intractable due to the complexity of the integral over $\mathbf{z}$. Variational inference [38] provides an alternative by introducing a variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z})$ and optimizing a lower bound on the marginal log-likelihood, known as the evidence lower bound (ELBO):

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}) \,\|\, p(\boldsymbol{z})\right) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}). \quad (2.9)$$

Modern implementations of SVI [39, 40] train models via gradient-based optimization of this bound with respect to both the model parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$. The first term in (5.1) encourages the model to

maximize likelihood over the data, but exact evaluation and differentiation are often intractable, necessitating sample-based approximations of the expectation. The KL divergence term incorporates prior information by penalizing deviations of the variational posterior from the prior $p(\boldsymbol{z})$. When a closed-form expression for the KL divergence is available, it provides an efficient means of encoding prior information; otherwise, sample-based approximations are required. For datasets where observations are assumed to be independent and each has its own associated latent variable, amortized inference is often used to improve efficiency. This involves introducing a neural network parameterized by $\boldsymbol{\phi}$ to directly map observations to variational parameters, approximating the posterior as $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. In many cases, modifying the ELBO by scaling the KL divergence term with a weighting factor $\beta_{\text{KL}} > 0$ is necessary to balance the trade-off between maximizing likelihood and imposing prior regularization [41]. The resulting sample-based approximation of this modified ELBO is denoted as $\hat{\mathcal{L}}\beta\text{KL}$.

**Gradient reparameterization: explicit and implicit.** A variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z})$ is said to be explicitly reparameterizable if it allows for the reparameterization trick, meaning its samples can be expressed as a deterministic, differentiable function $\mathbf{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi})$ of a noise variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. Typically, this base distribution is simple, such as Uniform or standard Normal, allowing for efficient sampling by first drawing $\boldsymbol{\epsilon}$ and then applying the transformation $g$. This enables efficient gradient estimation through backpropagation, as seen in the differentiation of an expectation such as [cf.

(5.1)]

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} f(g(\boldsymbol{\epsilon}, \boldsymbol{\phi}))] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{z}} f(\boldsymbol{z})|_{\boldsymbol{z}=g(\boldsymbol{\epsilon},\boldsymbol{\phi})} \nabla_{\boldsymbol{\phi}} g(\boldsymbol{\epsilon}, \boldsymbol{\phi})], \tag{2.10}$$

This reparameterization technique is particularly useful for distributions in the location-scale family (e.g., Gaussian, Laplace, Cauchy), those with closed-form inverse CDFs (e.g., Exponential, Kumaraswamy, Continuous Bernoulli), or distributions expressible as deterministic transformations of such distributions (e.g., the tanh-transformed Gaussian). For distributions that do not permit explicit reparameterization, implicit reparameterization [42] provides an alternative when the cumulative distribution function (CDF) is numerically tractable. Examples include truncated distributions, mixtures, Gamma, Beta, Dirichlet, and von Mises distributions. This approach expresses the gradient $\nabla_{\boldsymbol{\phi}} \mathbf{z}$ in terms of CDF derivatives rather than the inverse CDF, which can be computed either analytically (when feasible) or numerically via automatic differentiation. Since explicit reparameterization is often faster and more stable, implicit methods are generally employed only when no closed-form reparameterization is available. However, implicit reparameterization tends to produce higher-variance gradient estimates for (5.2), which can impact learning efficiency and stability [39, 43].

# Chapter 3

# Learning Graph Structure from Convolutional Mixtures

## 3.1 Introduction

Inferring graphs from data to uncover latent complex information structure is a timely challenge for geometric deep learning [44] and graph signal processing (GSP) [5]. But it is also an opportunity, since network topology inference advances [11,12,24] could facilitate adoption of graph neural networks (GNNs) even when no input graph is available, e.g. for disease prediction in healthcare settings [45], or, 3D point cloud segmentation in computer graphics [46]. The problem is also relevant when a given graph is too noisy or perturbed beyond what stable (possibly pre-trained) GNN architectures can effectively handle [47]. Early empirical evidence suggests that even when a graph is available, the structure could be further optimized for a downstream task e.g., when parameterizing a GNN used for node classification in citation, social, and protein-protein interaction networks [48]. Sometimes the observed graph's

large edge set can make a downstream task computationally expensive, which motivates seeking a sparser graph which retains properties of the original one – thus making the task feasible [49].

In this paper, we posit a convolutional model relating observed and latent undirected graphs and formulate the graph structure learning task as a supervised network inverse problem; see Section 3.2 for a formal problem statement. This fairly general model is motivated by various practical domains outlined in Section 3.3, such as identifying the structure of network diffusion processes [17, 50], as well as network deconvolution and denoising [51]. We propose a parameterized neural network (NN) model, termed graph deconvolution network (GDN), which we train in a supervised fashion to learn the distribution of latent graphs. The node permutation equivariant architecture is derived from the principle of algorithm unrolling used to learn fast approximate solutions to inverse problems [35, 36, 52], an idea that is yet to be fully explored for graph structure identification. Since layers directly operate on, combine, and refine graph objects (instead of nodal features), GDNs are inherently inductive and can generalize to graphs of different size. This allows the transfer of learning on small graphs to unseen larger graphs, which has significant implications in domains like social networks and molecular biology [53]. Our experiments demonstrate that GDNs are versatile to accommodate link prediction or edge-weight regression aspects of learning the graph structure, and achieve superior performance over various competing alternatives. Building on recent models of functional activity in

the brain as a diffusion over the underlying anatomical pathways [54, 55], we show the applicability of GDNs to infer brain structural connectivity from functional networks obtained from the Human Connectome Project-Young Adult (HCP-YA) dataset. We also use GDNs to predict Facebook ties from user co-location data, outperforming relevant baselines.

**Related work.** Graph structure learning (a.k.a. network topology inference) has a long history in statistics [14], with noteworthy contributions for probabilistic graphical model selection; see e.g. [1, 56, 57]. Recent advances were propelled by GSP insights through the lens of signal representation [11, 12, 24], exploiting models of network diffusion [58], or else leveraging cardinal properties of network data such as smoothness [19, 59] and graph stationarity [17, 50]. These works formulate (convex) optimization problems one has to solve for different graphs, and can lack robustness to signal model misspecifications. Network deconvolution approaches in [17, 51] operate in the graph spectral domain, and may face scalability issues because they rely on computationally-expensive eigendecompositions of the input graph for each problem instance. Moreover, none of these methods advocate a supervised learning paradigm using NNs to predict adjacency matrices as we propose here. In the broader context of NN-based models, so-termed latent graph learning has been shown effective in obtaining better task-driven representations of relational data for machine learning (ML) applications [45, 46, 60], or to learn interactions among coupled dynamical systems [61]. With regards to NN architectural design, algorithm unrolling has only recently been adopted

for graph structure learning in [62, 63]; but for different and more specific problems subsumed by the network deconvolution framework dealt with here. Indeed, [62] focuses on Gaussian graphical model selection in a supervised learning setting and [63] learns graph topologies under a smoothness signal prior.

**Summary of main contributions.** In this work, we introduce the following graph ML innovations:

•We introduce GDNs, a supervised learning NN model capable of recovering sparse latent graph structure from observations of its convolutional mixtures, i.e., related graphs containing spurious, indirect relationships. The novel supervised setting for graph structure learning is motivated via several real-world applications.

•The node permutation equivariant GDN architecture is derived from the principle of algorithm unrolling to learn fast approximate solutions to inverse problems. The unrolling offers explicit control on complexity (leading to fast inference times) and can seamlessly integrate domain-specific prior information about the unknown graph distribution. In constructing GDNs we leverage Multi-Input Multi-Output (MIMO) filters – as used in convolutional (C)NNs – for further expressiveness and training stability.

•On synthetic data, GDNs outperform comparable methods on link prediction and edge-weight regression tasks across different random-graph ensembles, while incurring a markedly lower (post-training) computational cost and inference time. GDNs are inductive and learnt models transfer across graphs

of different sizes. We empirically verify they exhibit minimal performance degradation even when tested on graphs 30× larger.

•Finally, using GDNs we propose a novel ML pipeline to learn whole brain structural connectivity (SC) from functional connectivity (FC), a challenging and timely problem in network neuroscience that motivates the supervised learning setting advocated here. Results on the HCP-YA imaging dataset show that GDNs perform well on specific brain subnetworks that are known to be relatively less correlated with the corresponding FC due to ageing-related effects – a testament to the model's robustness and expressive power. Overall, results here support the promising prospect of using graph ML to integrate brain structure and function.

## 3.2 Problem Formulation

In this work we study the following network inverse problem involving undirected and weighted graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of nodes (henceforth common to all graphs), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ collects the edges. We get to observe a graph with symmetric adjacency matrix $\boldsymbol{A}_O \in \mathbb{R}^{N \times N}$, that is related to a latent sparse, graph $\boldsymbol{A}_L \in \mathbb{R}_+^{N \times N}$ of interest via the forward data model

$$\boldsymbol{A}_O = h_0 \boldsymbol{I} + h_1 \boldsymbol{A}_L + \underbrace{h_2 \boldsymbol{A}_L^2 \ldots + h_K \boldsymbol{A}_L^K}_{\text{indirect relationships}} \tag{3.1}$$

for some $K \leq N - 1$ by the Cayley-Hamilton theorem. Matrix polynomials $\boldsymbol{H}(\boldsymbol{A}; \boldsymbol{h}) := \sum_{k=0}^{K} h_k \boldsymbol{A}^k$ with coefficients $\boldsymbol{h} := [h_0, \ldots, h_K]^\top \in \mathbb{R}^{K+1}$ as in the

right-hand-side of eq. (3.1), are known as shift-invariant graph convolutional filters; see e.g., [5, 64]. We postulate that $\boldsymbol{A}_O = \boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})$ for some filter order $K$ and its associated coefficients $\boldsymbol{h}$, such that we can think of the observed network as generated via a graph convolutional process acting on $\boldsymbol{A}_L$. That is, one can think of $\boldsymbol{A}_O$ as a graph containing spurious, indirect connections generated by the terms including higher-order powers of latent graph $\boldsymbol{A}_L$ – the graph of fundamental relationships. More pragmatically $\boldsymbol{A}_O$ may correspond to a noisy observation of $\boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})$, and this will be clear from the context when e.g., we estimate $\boldsymbol{A}_O$ from data.

Recovery of the latent graph $\boldsymbol{A}_L$ is a challenging endeavour since we do not know $\boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})$, namely the parameters $K$ or $\boldsymbol{h}$. Suppose that $\boldsymbol{A}_L$ is drawn from some distribution of sparse graphs, e.g., random geometric graphs or structural brain networks from a relatively homogeneous dataset. Then given independent training samples $\mathcal{T} := \{\boldsymbol{A}_O^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$ adhering to eq. (3.1), our goal is to learn a parametric mapping $\Phi$ that predicts the graph adjacency matrix $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$ by minimizing a loss function

$$L(\boldsymbol{\Theta}) := \frac{1}{T} \sum_{i \in \mathcal{T}} \ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})) \tag{3.2}$$

to search for the best parameters $\boldsymbol{\Theta}$. The loss $\ell$ is adapted to the task at hand – hinge loss for link prediction or mean-squared error (MSE) for the more challenging edge-weight regression problem. Notice that eq. (3.1) postulates a common filter across graph pairs in $\mathcal{T}$ – a simplifying assumption shown

to be tenable in functional magnetic resonance imaging (fMRI) studies [54] where subject-level coupling between the structural and functional modalities (SC-FC coupling) exhibit limited variability. This will be relaxed when we customize the GDN architecture to learn multiple filters; see Section 3.4.3 for details.

Potential application domains for which supervised network data $\mathcal{T} := \{\boldsymbol{A}_O^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$ is available include bioinformatics (infer protein contact structure $\boldsymbol{A}_L$ from mutual information graphs $\boldsymbol{A}_O$ of the covariation of amino acid residues [51]), gene regulatory network inference from microarray data (see e.g. the DREAM5 project to reconstruct networks for the E. coli bacterium and single-cell eukaryotes), social and information networks (e.g., learn to sparsify graphs [49] to unveil the most relevant collaborations $\boldsymbol{A}_L$ in a social network encoding co-authorship information $\boldsymbol{A}_O$ [17]), and epidemiology (such as contact tracing by deconvolving the graphs that model observed disease spread in a population). In Section 3.5.2 we experiment with social networks and the network neuroscience problem described next.

## 3.3 Motivating Application Domains and the Supervised Setting

Here we outline several graph structure learning tasks that can be cast as the network inverse problem in eq. (3.1), and give numerous examples supporting the relevance of the novel supervised setting.

**Graph structure identification from diffused signals.** Our initial focus is on identifying graphs that explain the structure of a class of network diffusion processes. Formally, let $\boldsymbol{x} \in \mathbb{R}^N$ be a graph signal (i.e., a vector of nodal features) supported on a latent graph $\mathcal{G}$ with adjacency $\boldsymbol{A}_L$. Further, let $\boldsymbol{w}$ be a zero-mean white signal with covariance matrix $\boldsymbol{\Sigma}_w = \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^\top] = \boldsymbol{I}$. We say that $\boldsymbol{A}_L$ represents the structure of the signal $\boldsymbol{x}$ if there exists a linear network diffusion process in $\mathcal{G}$ that generates the signal $\boldsymbol{x}$ from $\boldsymbol{w}$, namely $\boldsymbol{x} = \sum_{i=0}^{\infty} h_i \boldsymbol{A}_L^i \boldsymbol{w} = \boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})\boldsymbol{w}$. This is a fairly common generative model for random network processes [29, 65]. We think of the edges of $\mathcal{G}$ as direct (one-hop) relations between the elements of the signal $\boldsymbol{x}$. The diffusion described by $\boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})$ generates indirect relations. In this context, the graph structure learning problem is to recover a sparse $\boldsymbol{A}_L$ from a set $\mathcal{X} := \{\boldsymbol{x}_i\}_{i=1}^P$ of $P$ samples of $\boldsymbol{x}$ [17]. Interestingly, from the model for $\boldsymbol{x}$ it follows that the signal covariance matrix $\boldsymbol{\Sigma}_x = \mathbb{E}\left[\boldsymbol{x}\boldsymbol{x}^\top\right] = \boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})\mathbb{E}\left[\boldsymbol{w}\boldsymbol{w}^\top\right]\boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h}) = \boldsymbol{H}^2(\boldsymbol{A}_L; \boldsymbol{h})$ is also a polynomial in $\boldsymbol{A}_L$ – precisely the relationship prescribed by eq. (3.1) when $\boldsymbol{A}_O = \boldsymbol{\Sigma}_x$. In practice, given the signals in $\mathcal{X}$ one would estimate the covariance matrix, e.g. via the sample covariance $\hat{\boldsymbol{\Sigma}}_x$, and then aim to recover the graph $\boldsymbol{A}_L$ by tackling the aforementioned network inverse problem. In this paper, we propose a fresh learning-based solution using training examples $\mathcal{T} := \{\hat{\boldsymbol{\Sigma}}_x^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$.

**Remark** (Graph convolutional data model in context)**.** To further elaborate on the relevance and breadth of applicability of the graph convolutional (or network diffusion) signal model $\boldsymbol{x} = \boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})\boldsymbol{w}$, we would like to elucidate

connections with related work for graph structure learning. Note that while we used the diffusion-based generative model to formulate the problem above, we do not need it as an actual mechanistic process. Indeed, like in eq. (3.1) the only thing we ask is for the data covariance $\boldsymbol{A}_O = \boldsymbol{\Sigma}_x$ to be some analytic function of the latent graph $\boldsymbol{A}_L$. This is not extraneous to workhorse statistical methods for network topology inference, which (implicitly) make specific choices for these mappings, e.g. (i) correlation networks [1, Ch. 7] rely on the identity mapping $\boldsymbol{\Sigma}_x = \boldsymbol{A}_L$; (ii) Gaussian graphical model selection methods, such as graphical lasso in [56, 66], adopt $\boldsymbol{\Sigma}_x = \boldsymbol{A}_L^{-1}$; and (iii) undirected structural equation models $\boldsymbol{x} = \boldsymbol{A}_L\boldsymbol{x} + \boldsymbol{w}$ which implies $\boldsymbol{\Sigma}_x = (\boldsymbol{I} - \boldsymbol{A}_L)^{-2}$ [11]. Accordingly, these models are all subsumed by the general framework we put forth here. For a recent and inspiring supervised learning approach to Gaussian graphical model selection rooted on graphical lasso, the interested reader is referred to [62].

**Network deconvolution and denoising.** The network deconvolution problem is to identify a sparse adjacency matrix $\boldsymbol{A}_L$ that encodes direct dependencies, when given an adjacency matrix $\boldsymbol{A}_O$ containing extraneous indirect relationships. The problem broadens the scope of signal deconvolution to networks and can be tackled by attempting to invert the mapping $\boldsymbol{A}_O = \boldsymbol{A}_L\left(\boldsymbol{I} - \boldsymbol{A}_L\right)^{-1} = \boldsymbol{A}_L + \boldsymbol{A}_L^2 + \boldsymbol{A}_L^3 \ldots$ This solution proposed in [51] assumes a polynomial relationship as in eq. (3.1), but for the particular case of a single-pole, single-zero graph filter with very specific filter coefficients [cf. eq. (3.1) with $h_0 = 0$ and $h_k = 1$, $k \geq 1$]. This way, the indirect dependencies observed

in $\boldsymbol{A}_O$ arise due to the higher-order convolutive mixture terms $\boldsymbol{A}_L^2 + \boldsymbol{A}_L^3 + \ldots$ superimposed to the direct interactions in $\boldsymbol{A}_L$ we wish to recover. Here we adopt a general data-driven learning approach in assuming that $\boldsymbol{A}_O$ can be written as a polynomial in $\boldsymbol{A}_L$, but being agnostic to the form of the filter, thus broadening the problem setting in [51]. Unlike the problem outlined in the previous subsection, here $\boldsymbol{A}_O$ need not be a covariance matrix. Indeed, $\boldsymbol{A}_O$ could be a corrupted graph we wish to denoise, obtained via an upstream graph learning method. A related but different deconvolution problem was put forth in [67]. Rather than recovering graph structure, the goal therein is to reconstruct the input graph signals from smoothed nodal representations generated by a graph convolutional network (GCN) [68].

**Inferring structural brain networks from functional MRI (fMRI) signals.** Brain connectomes encompass networks of brain regions connected by (statistical) functional associations (FC) or by anatomical white matter fiber pathways (SC). The latter can be extracted from time-consuming tractography algorithms applied to diffusion MRI (dMRI), which are particularly fraught due to quality issues in the data [69]. FC represents pairwise correlation structure between blood-oxygen-level-dependent (BOLD) signals measured by fMRI. Deciphering the relationship between SC and FC is a very active area of research [54, 70] and also relevant in studying neurological disorders, since it is known to vary with respect to healthy subjects in pathological contexts [71]. Traditional approaches exploring the SC-FC coupling go all the way from correlation studies [70] to large-scale simulations of nonlinear

cortical activity models [70]. More aligned with the problem addressed here, recent studies have shown that linear diffusion dynamics can reasonably model the SC-FC coupling [54]. Using our notation, the findings in [54] suggest that the covariance $\boldsymbol{A}_O = \boldsymbol{\Sigma}_x$ of the functional signals (i.e., the FC graph) is related to the the sparse SC network $\boldsymbol{A}_L$ via the model in eq. (3.1). Similarly, [55] contend *FC can be modeled as a weighted sum of powers of the SC matrix*, consisting of both direct and indirect effects along varying paths. There is evidence that FC links tend to exist where there is no or little structural connection [70], a property naturally captured by eq. (3.1). These considerations motivate adopting our graph strcuture learning method to infer SC patterns from fMRI signals using the training set $\mathcal{T} := \{\mathbf{FC}^{(i)}, \mathbf{SC}^{(i)}\}_{i=1}^{T}$ (Section 3.5.2), a significant problem for several reasons. The ability to collect only FC and get informative estimates of SC open the door to large-scale studies, previously constrained by the logistical, cost, and computational resources needed to acquire both modalities.

## 3.4 Graph Deconvolution Network

Here we present the proposed GDN model, a NN architecture for graph structure learning that we train in a supervised fashion. In the sequel, we obtain 'conceptual' iterations to tackle an optimization formulation of the network inverse problem (Section 3.4.1), unroll these iterations to arrive at the parametric, differentiable GDN function $\Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$ we train using graph

data $\mathcal{T}$ (Section 3.4.2), and describe architectural customizations to improve performance (Section 3.4.3).

### 3.4.1 Iterative optimization as NN architectural blueprint

Going back to the inverse problem of recovering a sparse adjacency matrix $\boldsymbol{A}_L$ from the mixture $\boldsymbol{A}_O$ in eq. (3.1), if the graph filter $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})$ were known – but recall it is not – we could attempt to solve

$$\hat{\boldsymbol{A}}_L \in \underset{\boldsymbol{A}\in\mathcal{A}}{\operatorname{argmin}} \left\{ \|\boldsymbol{A}\|_1 + \frac{\lambda}{2}\|\boldsymbol{A}_O - \boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})\|_F^2 \right\}, \tag{3.3}$$

where $\lambda > 0$ trades off sparsity for reconstruction error. The convex set $\mathcal{A} := \{\boldsymbol{A} \in \mathbb{R}^{N\times N} \mid \operatorname{diag}(\boldsymbol{A}) = \boldsymbol{0}, A_{ij} = A_{ij} \geq 0, \forall i,j \in \{1,\ldots,N\}\}$ encodes the admissibility constraints on the adjacency matrix of an undirected graph: hollow diagonal, symmetric, with non-negative edge weights. The $\ell_1$ norm encourages sparsity in the solution, being a convex surrogate of the edge-cardinality function that counts the number of non-zero entries in $\boldsymbol{A}$. Since $\boldsymbol{A}_O$ is often a noisy observation or estimate of the polynomial $\boldsymbol{H}(\boldsymbol{A}_L;\boldsymbol{h})$, it is prudent to relax the equality in eq. (3.1) and minimize the squared residual errors instead. As described earlier, this could be the case when $\boldsymbol{A}_O = \hat{\boldsymbol{\Sigma}}_x$ (e.g., brain FC) and the empirical covariance matrices are estimated from finite data.

The composite cost in problem (3.3) is a weighted sum of a non-smooth function $\|\boldsymbol{A}\|_1$ and a continuously differentiable, non-convex function $g(\boldsymbol{A}) :=$

$\frac{1}{2}\|\boldsymbol{A}_O - \boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})\|_F^2$. But our end goal here is not to solve problem (3.3) iteratively, recall we cannot even formulate the optimization problem because $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})$ is unknown. To retain the essence of the problem structure and motivate a parametric model to learn approximate solutions from $\mathcal{T}$, it suffices to settle with 'conceptual' proximal gradient (PG) iterations ($k$ henceforth denote iterations, $\boldsymbol{A}[0] \in \mathcal{A}$)

$$\boldsymbol{A}[k+1] = \text{ReLU}\left(\boldsymbol{A}[k] - \tau\nabla g(\boldsymbol{A}[k]) - \tau\mathbf{1}\mathbf{1}^\top\right) \tag{3.4}$$

for $k = 0, 1, 2, \ldots$, where $\tau$ is a step-size parameter in which we have absorbed $\lambda$. These iterations implement a gradient descent step on $g$ followed by the $\ell_1$ norm's proximal operator. Due to the non-negativity constraints in $\mathcal{A}$, the $\ell_1$ norm's proximal operator takes the form of a $\tau$-shifted ReLU on the off-diagonal entries of its matrix argument. Also, the operator sets $\text{diag}(\boldsymbol{A}[k+1]) = \mathbf{0}$.

We cannot run the iterates in (3.4) without knowledge of $\boldsymbol{h}$, and we will make no attempt to estimate $\boldsymbol{h}$. Instead, our approach in the next section is to unroll and truncate these iterations (and in the process, convert unknowns to learnable parameters $\boldsymbol{\Theta}$) to arrive at the trainable GDN parametric model $\Phi(\boldsymbol{A}_O;\boldsymbol{\Theta})$. This way, the iterations in (3.4) will serve as NN architectural blueprint to tackle the supervised problem stated in Section 3.2.

### 3.4.2 Learning to infer graphs via algorithm unrolling

The idea of algorithm unrolling can be traced back to the seminal work of [35]. In the context of sparse coding, they advocated identifying *iterations* of PG algorithms with *layers* in a deep network of fixed depth that can be trained from examples using backpropagation. One can view this process as effectively truncating the iterations of an asymptotically convergent procedure, to yield a template architecture that learns to approximate solutions with substantial computational savings relative to the optimization algorithm. Beyond parsimonious signal modeling, there has been a surge in popularity of unrolled deep networks for a wide variety of applications; see e.g., [36] for a recent tutorial treatment focused on signal and image processing.

However, to the best of our knowledge this approach is yet to be fully explored for graph structure learning. L2G [63] and GLAD [62] represent recent inspiring attempts, but neither are NNs (meaning compositions of affine layers followed by point-wise nonlinear activations). Both process the input data and the hidden representations in a nonlinear fashion (pre activation) involving, e.g., cross-parameter products or parameter inverses within their layers, challenging optimization during training [36]. GLAD includes a matrix square root operation which poses scalability issues due to its cubic complexity.

Building on the algorithm unrolling paradigm and starting from the PG iterations (3.4), we design a non-linear, parameterized, feed-forward NN architecture that can be trained to predict the latent graph $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$.

We aim at a NN architecture for its favorable optimization properties, yet desire to retain expressiveness. To this end, we approximate the gradient $\nabla g(\boldsymbol{A})$ by retaining only linear terms in $\boldsymbol{A}$, and build a deep network by composing layer-wise linear filters and point-wise nonlinearites to capture higher-order interactions in the generative process $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h}) := \sum_{k=0}^{K} h_k \boldsymbol{A}^k$. This rationale is similar to the one followed in developing the GCN model [68]. Therein, it is argued that a rich class of convolutional filter functions (e.g., those parameterized via Chebyshev polynomials) can be recovered by stacking multiple linear GCN layers.

In more detail, we start by simplifying $\nabla g(\boldsymbol{A})$ by dropping all higher-order terms in $\boldsymbol{A}$, namely

$$\begin{aligned}\nabla g(\boldsymbol{A}) &= -\sum_{k=1}^{K} h_k \sum_{r=0}^{k-1} \boldsymbol{A}^{k-r-1} \boldsymbol{A}_O \boldsymbol{A}^r + \frac{1}{2} \nabla_{\boldsymbol{A}} \operatorname{Tr}\left[\boldsymbol{H}^2(\boldsymbol{A};\boldsymbol{h})\right] \\ &\approx -h_1 \boldsymbol{A}_O - h_2(\boldsymbol{A}_O \boldsymbol{A} + \boldsymbol{A}\boldsymbol{A}_O) + (2h_0 h_2 + h_1^2)\boldsymbol{A}. \end{aligned} \tag{3.5}$$

Notice that $\nabla_{\boldsymbol{A}} \operatorname{Tr}\left[\boldsymbol{H}^2(\boldsymbol{A};\boldsymbol{h})\right]$ is a polynomial of degree $2K-1$. Hence, we keep the linear term in $\boldsymbol{A}$ but drop the constant offset that is proportional to the identity matrix $\boldsymbol{I}$, which is inconsequential to adjacency matrix updates with null diagonal. An affine approximation leads to more benign optimization landscapes when it comes to training the resulting GDN model; see Section 3.5 for ablation studies exploring the effects of higher-order terms in the

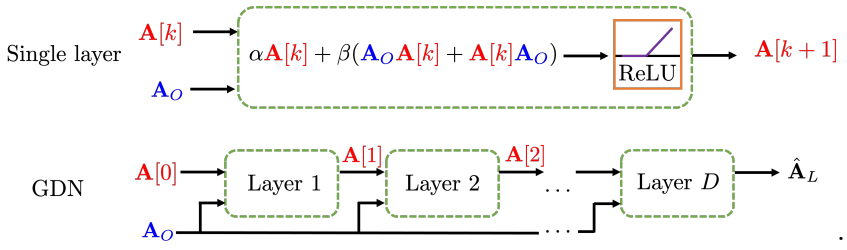


Figure 3.1: Schematic diagram of the GDN architecture obtained via algorithm unrolling.

approximation. All in all, the simplified PG iterations become

$$\boldsymbol{A}[k+1] = \text{ReLU}\left(\alpha \boldsymbol{A}[k] + \beta(\boldsymbol{A}_O \boldsymbol{A}[k] + \boldsymbol{A}[k]\boldsymbol{A}_O) + \gamma \boldsymbol{A}_O - \tau \mathbf{1}\mathbf{1}^\top\right), \quad (3.6)$$

where $\boldsymbol{A}[0] \in \mathcal{A}$ and we defined $\alpha := (1 - 2\tau h_0 h_2 - \tau h_1^2)$, $\beta := \tau h_2$, and $\gamma := \tau h_1$. The latter parameter triplet encapsulates filter (i.e., mixture) coefficients and the $\lambda$-dependent algorithm step-size $\tau$, all of which are unknown in practice.

The GDN architecture is thus obtained by unrolling the iterations (3.6) into a deep NN; see Figure 3.1. This entails mapping each individual iteration into a layer and stacking a prescribed number $D$ of layers together to form $\Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$. The unknown filter coefficients are treated as learnable parameters $\boldsymbol{\Theta} := \{\alpha, \beta, \gamma, \tau\}$, which are shared across layers as in recurrent neural networks (RNNs). The reduced number of parameters relative to most typical NNs is a characteristic of unrolled networks [36]. A first noteworthy property of the GDN layer in eq. (3.6) is that it preserves symmetry, namely if the input $\boldsymbol{A}[k]$ to layer $k+1$ is a symmetric matrix then so is the output $\boldsymbol{A}[k+1]$.

Second, GDN parametrizations are equivariant to node permutations – an essential property that guarantees the learnt graph topologies are not affected by the order in which we label the vertices.

**Proposition 1** (Permutation equivariance of GDNs)**.** Let $\boldsymbol{P} \in \{0,1\}^{N\times N}$ be a permutation matrix and consider identical row and column permutations of the observed graph $\tilde{\boldsymbol{A}}_O = \boldsymbol{P}^\top \boldsymbol{A}_O \boldsymbol{P}$ as well as the initial state $\tilde{\boldsymbol{A}}[0] = \boldsymbol{P}^\top \boldsymbol{A}[0]\boldsymbol{P}$. Let $\Phi(\boldsymbol{A}_O; \boldsymbol{\Theta}, \boldsymbol{A}[0])$ denote the GDN output corresponding to input $\boldsymbol{A}_O$, where for convenience we make explicit that the model is parameterized by the initial state $\boldsymbol{A}[0]$. Then, it holds that GDNs are permutation equivariant, namely

$$\Phi(\tilde{\boldsymbol{A}}_O; \boldsymbol{\Theta}, \tilde{\boldsymbol{A}}[0]) = \boldsymbol{P}^\top \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta}, \boldsymbol{A}[0])\boldsymbol{P}. \tag{3.7}$$

*Proof.* Let $\boldsymbol{A}[k+1] = f(\boldsymbol{A}[k], \boldsymbol{A}_O)$ represent the layer $k+1$ mapping in eq. (3.6). Then by regrouping terms, recalling that $\boldsymbol{P}^\top \boldsymbol{P} = \boldsymbol{I}$, and using the fact that $\mathrm{ReLU}(\cdot)$ and $\mathbf{1}\mathbf{1}^\top$ are both permutation equivariant, we find via simple algebraic manipulations that $f(\boldsymbol{P}^\top \boldsymbol{A}[k]\boldsymbol{P}, \tilde{\boldsymbol{A}}_O) = \boldsymbol{P}^\top f(\boldsymbol{A}[k], \boldsymbol{A}_O)\boldsymbol{P}$. Because each layer satisfies the permutation equivariance property, the GDN architecture $\Phi(\boldsymbol{A}_O; \boldsymbol{\Theta}, \boldsymbol{A}[0])$ that is a composition of $D$ such layers is also permutation equivariant; meaning that eq. (3.7) holds true. □

In the next section, we will explore a few customizations to the vanilla GDN architecture in order to broaden the model's expressive power. Given a training set $\mathcal{T} = \{\boldsymbol{A}_O^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$, learning is accomplished by using mini-batch stochastic gradient descent to minimize the task-dependent loss function

$L(\boldsymbol{\Theta})$ in eq. (3.2). We adopt a hinge loss for link prediction and mean-squared/absolute error for the edge-weight regression task. For link prediction, we also learn a threshold $t \in \mathbb{R}_+$ to binarize the estimated edge weights and declare presence or absence of edges.

The iterative refinement principle of optimization algorithms carries over to our GDN model during inference. Indeed, we start with an initial estimate $\boldsymbol{A}[0] \in \mathcal{A}$ and use a cascade of $D$ linear filters and point-wise nonlinearities to refine it to an output $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \hat{\boldsymbol{\Theta}})$; see Fig. 3.2 for an example drawn from the experiments in Section 3.5.1. Matrix $\boldsymbol{A}[0]$ is a hyperparameter we can select to incorporate prior information on the sought latent graph, or it could be learned as it is customary with the intial state in RNNs; see Section 3.4.3. The input graph $\boldsymbol{A}_O$ which we aim to deconvolve is directly fed to all layers, and contributes to defining non-uniform soft thresholds $\gamma \boldsymbol{A}_O - \tau \mathbf{1}\mathbf{1}^\top$ which sparsify the layer's output. One can also interpret $\alpha \boldsymbol{A} + \beta(\boldsymbol{A}_O \boldsymbol{A} + \boldsymbol{A}\boldsymbol{A}_O)$ as a first-order, symmetry-preserving filter defined on graph $\boldsymbol{A}_O$, which is used to process $\boldsymbol{A} \in \mathcal{A}$ – here viewed as a structured graph signal with $N$ features per node to invoke this GSP insight. In its simplest rendition, the GDN leverages elements of RNNs and GCNs [68].

### 3.4.3 GDN architecture customizations

Here we outline several customizations and enhancements to the vanilla GDN architecture of the previous section, which we have empirically found to improve graph structure learning performance.

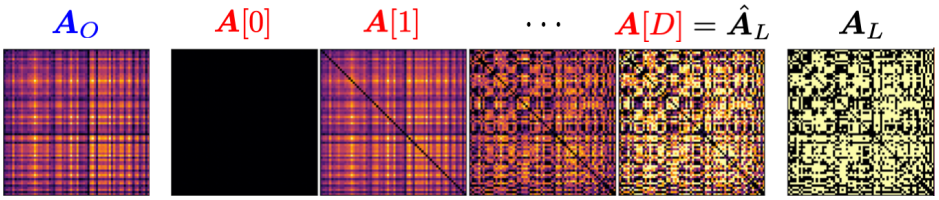


Figure 3.2: Intermediate outputs of a GDN model trained on the random geometric (RG) graphs in Table 3.1. GDNs refine the input $\boldsymbol{A}_O$ to more closely resemble $\boldsymbol{A}_L$ as we go deeper in the network. Notice how the predicted output adjacency matrix $\hat{\boldsymbol{A}}_L$ recovers the connectivity structure in $\boldsymbol{A}_L$.

**Incorporating prior information via algorithm initialization.** By viewing our method as an iterative refinement of an initial graph $\boldsymbol{A}[0]$, one can think of $\boldsymbol{A}[0]$ as a best initial guess, or *prior*, over $\boldsymbol{A}_L$. A simple strategy to incorporate prior information about some edge $(i, j)$, encoded in $A_{ij}$ that we view as a random variable, would be to set $A[0]_{ij} = \mathbb{E}[A_{ij}]$. This technique is adopted when training on the HCP-YA dataset in Section 3.5, by taking the prior $\boldsymbol{A}[0]$ to be the sample mean of all latent (i.e., SC) graphs in the training set $\mathcal{T}$. This encodes our prior knowledge that there are strong similarities in the structure of the human brain across the population of healthy young adults. When $\boldsymbol{A}_L$ is expected to be reasonably sparse, we can set $\boldsymbol{A}[0] = \mathbf{0}$, which is effective as we show in Table 3.1. Recalling the connections drawn between GDNs and RNNs, then the prior $\boldsymbol{A}[0]$ plays a similar role to the initial RNN state and thus it could be learned [72]. In any case, the ability to seamlessly incorporate prior information to the model through the initial state is an attractive feature of unrolling-based NNs [36] and of GDNs in particular.

**Multi-Input Multi-Output (MIMO) filters.** So far, in each layer we have a single learned filter, which takes an $N \times N$ matrix as input and returns another $N \times N$ matrix at the output. After going through the shifted ReLU nonlinearity, this refined output adjacency matrix is fed as input to the next layer; a process that repeats $D$ times. More generally, we can allow for multiple input channels (i.e., a tensor), as well as multiple channels at the output, by using the familiar convolutional neural network (CNN) methodology. This way, each output channel has its own filter parameters associated with every input channel. The $j$-th output channel applies its linear filters to all input channels, aggregating the results with a reduction operation (e.g., mean or sum), and applies a point-wise nonlinearity (here a shifted ReLU) to the output. This allows the GDN model to learn many different filters, thus providing richer learned representations and effectively relaxing the simplifying assumption we made regarding a common filter across graph pairs in eq. (3.1). Notice that the aforementioned enhancement to the GDN architecture does not compromise its permutation equivariance property.

**Decoupling layer parameters.** Thus far, we have respected the parameter sharing constraint imposed by the unrolled PG iterations. We now allow each layer to learn a decoupled MIMO filter, with its own set of parameters mapping from $C_{in}^k$ input channels to $C_{out}^k$ output channels. As the notation suggests, $C_{in}^k$ and $C_{out}^k$ need not be equal. By decoupling the layer structure, we allow GDNs to compose different learned filters to create more abstract

features (as with CNNs or GCNs). Accordingly, it opens up the architectural design space to broader exploration, e.g., wider layers early and skinnier layers at the end. Exploring this architectural space is beyond the scope of this paper and is left as future work. In experiments, we use GDNs for which intermediate layers $k \in \{2, \dots, D-1\}$ have $C = C_{in}^k = C_{out}^k$, i.e., a flat architecture. We denote models with layer-wise shared parameters and independent parameters as GDN-S and GDN, respectively.

## 3.5 Experiments

We present comprehensive experiments on link prediction and edge-weight regression tasks using synthetic data (Section 3.5.1) as well as real HCP-YA neuroimaging and social network data (Section 3.5.2).

**Graph generation.** In the synthetic data experiments we consider three test cases whereby the latent graphs are respectively drawn from ensembles of Erdős-Rényi (ER), random geometric (RG), and Barabási-Albert (BA) random graph models; see e.g., [1]. We study an additional scenario where we use SCs from HCP-YA (referred to as the 'pseudo-synthetic' case because the latent graphs are real structural brain networks, while the signals used to estimate covariance matrices $\boldsymbol{A}_O = \hat{\boldsymbol{\Sigma}}_x$ are synthetic; see Section 3.5.1).

**Figures of merit.** In the link prediction task, performance is evaluated using error $:= \frac{\text{incorrectly predicted edges}}{\text{total possible edges}}$. For regression, we adopt the mean-squared-error (MSE) or mean-absolute-error (MAE) as figures of merit.

**Baselines.** We compare GDNs against several relevant graph structure learning baselines: Network Deconvolution (ND) which uses a spectral approach to directly invert a very specific convolutive mixture [51]; Spectral Templates (SpecTemp) that advocates a convex optimization approach to recover sparse graphs from noisy estimates of $\boldsymbol{A}_O$'s eigenvectors [17]; Graphical LASSO (GLASSO), a regularized MLE of the precision matrix for Gaussian graphical model selection [56]; a learned unrolling of alternating minimization (AM) on the GLASSO objective (GLAD) [62]; a learned unrolling of a primal-dual splitting (PDS) algorithm to minimize a graph recovery criterion which promotes a signal smoothness prior (L2G) [63]; a GNN encoder-decoder model on graph $\boldsymbol{A}_O$ using the degree vector $\boldsymbol{A}_O\mathbf{1}$ as nodal signals, with encoders based on the GCN (GCN-D) or the graph isomorphism network [73] (GIN-D); least-squares fitting of $\boldsymbol{h}$ followed by non-convex optimization to solve problem (3.3) (LSOpt); and Hard Thresholding (Threshold) to assess how well a simple cutoff rule can perform. GLASSO, ND, and SpecTemp generally tune their regularization parameters in an unsupervised way, but here they are tuned with supervision. The network inverse problem only assumes knowledge of the observed graph $\boldsymbol{A}_O$ for inference - *there is no nodal information available and GDN layers do not operate on node features.* As such, popular GNN-based methods used in the link-prediction task [45, 46, 60, 61, 74, 75] are not directly applicable here. We carefully reviewed the network inference literature and chose the most performant baselines available for this problem setting.

**GDN-related hyperparameters and experimental setting.** Unless

otherwise stated, in all the results that follow we use GDN(-S) models with $D = 8$ layers, $C = 8$ channels per layer, take prior $\boldsymbol{A}[0] = \mathbf{0}$ on all domains except the SCs - where we use the sample mean of all SCs in the training set, and train using the ADAM optimizer [76] with learning rate of 0.01 and batch size of 200. We use one Nvidia T4 GPU; models take $< 15$ minutes to train on all datasets. A detailed account on the GDN hyperparameter search process is included in the supplementary material.

### 3.5.1 Learning graph structure from diffused signals

A set of latent graphs are either sampled from RG, ER, or the BA model, or taken as the SCs from the HCP-YA dataset. In an attempt to make the synthetic latent graphs somewhat comparable to the 68 node SCs, we sample connected $N = 68$ node graphs with edge densities in the range $[0.5, 0.6]$ when feasible – typical values observed in SC graphs. To generate each observation $\boldsymbol{A}_O^{(i)}$, we simulated $P = 50$ standard Normal white signals diffused over $\boldsymbol{A}_L^{(i)}$; from which we form the sample covariance $\hat{\boldsymbol{\Sigma}}_x$ as described in Section 3.3. The value of $P$ is chosen based on a study of robustness to noise in the observations across multiple models, which showed only marginally improved performance when averaging more signals as shown in Table 3.2. With respect to the diffusion graph filters we let $K = 2$, and sample the filter coefficients $\boldsymbol{h} \in \mathbb{R}^3$ in $\boldsymbol{H}(\boldsymbol{A}_L; \boldsymbol{h})$ uniformly from the unit sphere. To examine robustness to the realizations of $\boldsymbol{h}$, we repeat this data generation process three times (resampling the filter coefficients). We thus create three

different datasets for each graph domain (12 in total). For the sizes of the training/validation/test splits, the pseudo-synthetic domain uses 913/50/100 and the synthetic domains use 913/500/500.

Table 3.1 tabulates the results for synthetic and pseudo-synthetic experiments. For graph models that exhibit localized connectivity patterns (RG and SC), GDNs significantly outperform the baselines on both tasks. For the SC test case, GDN (GDN-S) reduces error relative to the mean prior by $27.5 \pm 1.7\%$ ($23.0 \pm 1.7\%$) and MSE by $37.3 \pm 0.8\%$ ($23.2 \pm 0.5\%$). Both GDN architectures show the ability to learn such local patterns, with the extra representational power of GDNs (over GDN-S) providing an additional boost in performance. All models struggle on BA and ER with GDNs showing better performance even for these cases.

**Robustness to noise.** To evaluate the robustness of GDN's to noise in the observations $\boldsymbol{A}_O$, we train a GDN model with $D = 30$ and $C = 1$ on multiple datasets with varying levels of noise. We impliclty control the level of noise by changing the number of signals $P$ used to estimate $\boldsymbol{A}_O$ - taken to be the sample covariance matrix $\hat{\boldsymbol{\Sigma}}_x$. The data are the same as in Table 3.1, except here we perform diffusions using one of the sampled filter coefficients $\boldsymbol{h} \in \mathbb{R}^3$. The results of this experiment on both GDN and Threshold are reported in Table 3.2. We find that GDN's are robust to noisy observed graphs $\boldsymbol{A}_O$, achieving less than 10% error with only 20 observed signals. Up to $P = 50$, GDN performs significantly better when given more signals, with only marginal increases in performance after this point. This experiment also

highlights the advantage of MIMO filters as a non-MIMO GDN of depth 30 is outperformed by the MIMO GDN of depth 8 on the $P = 50$ case.

**Scaling and size generalization: Deploying on larger graphs.** GDNs learn the parameters of graph convolutions for the processing of graphs making them inductive: we can deploy the learnt model on larger graph size domains. Such a deployment is feasible on larger graphs due to the $\mathcal{O}(N^2)$ time and memory complexity of GDNs; the same is true for L2G. This stands in contrast to the $\mathcal{O}(N^3)$ time complexity of SpecTemp, GLASSO, ND, and GLAD. SpecTemp and ND perform a full eigendecomposition of $\boldsymbol{A}_O$, the iterative algorithms for solving GLASSO incur cubic worst-case complexity [77], and GLAD performs a matrix square root in *each* layer. GLAD also requires deeper unrolled networks, discounted intermediate losses, and multiple embedded MLPs, which have significant practical implications on memory usage.

To test the extent to which GDNs generalize when $N$ grows, we trained GDN(-S) on RG graphs with size $N = 68$, and tested them on RG graphs of size $N = [68, 75, 100, 200, 500, 1000, 2000]$, with 200 graphs of each size. As graph sizes increase, we require more samples in the estimation of the sample covariance to maintain a constant signal-to-noise ratio. To simplify the experiment and its interpretation, we disregard estimation and directly use the ensemble covariance $\boldsymbol{A}_O \equiv \boldsymbol{\Sigma}_x$ as observation. As before, we take a training/validation split of 913/500. Figure 3.3 shows GDNs effectively generalize to graphs orders of magnitude larger than they were trained on,

giving up only modest levels of performance as size increases. Note that the best performing baseline in Table 3.1 - trained *and tested* on the original $N = 68$ domain - is not comparable with GDN-S in terms of performance until GDN-S is tested on graphs almost an order of magnitude larger than those it was trained on. The GDN-S model showed a more graceful performance degradation, suggesting that GDNs without parameter sharing may be using their extra representational power to pick up on finite-size effects, which may disappear as $N$ increases. The shared parameter constraint acts as regularization, we avoid over-fitting on a given size domain to better generalize to larger graphs.

**Ablation studies.** The choice of prior can influence model performance, as well as reduce training time and the number of parameters needed. When run on stochastic block model (SBM) graphs with $N = 21$ nodes and 3 equally-sized communities (within block connection probability of 0.6, and 0.1 across blocks), for the link prediction task GDNs attain an error of $16.8 \pm 2.7e$-2%, $16.0 \pm 2.1e$-2%, $14.5 \pm 1.0e$-2%, $14.3 \pm 8.8e$-2% using a zeros, ones, block diagonal (using the SBM edge connectivity probabilities), and learned prior, respectively. The performance improves when GDNs are given an informative prior (here a block diagonal matrix matching the graph communities), with further gains when GDNs are allowed to learn $\boldsymbol{A}[0]$.

We also study the effect of gradient truncation. To derive GDNs we approximate the gradient $\nabla g(\boldsymbol{A})$ by dropping all higher-order terms in $\boldsymbol{A}$ ($K = 1$). The case of $K = 0$ corresponds to further dropping the terms

linear in $\boldsymbol{A}$, leading to PG iterations $\boldsymbol{A}[k+1] = \mathrm{ReLU}(\boldsymbol{A}[k] + \gamma \boldsymbol{A}_O - \tau \mathbf{1}\mathbf{1}^\top)$ [cf. eq. (3.6)]. We run this simplified iterative soft-thresholding model with $D = 8$ layers and $C = 8$ channels per layer on the same RG graphs in Table 3.1. Lacking the linear term that facilitates information aggregation in the graph, the model is not expressive enough and yields a higher error (MSE) of $25.7 \pm 1.3e\text{-}2\%$ ($1.7e\text{-}1 \pm 4.7e\text{-}4$) for the link-prediction (edge weight regression) task. Models with $K \geq 2$ result in unstable training with highly fluctuating loss values, which we attribute to a challenging optimization landscape as polynomial minimization is known to be notoriously hard, leading to less well-behaved gradients. This motivates our choice of $K = 1$ in GDNs, which are nonetheless flexible enough to capture a rich class of (higher-order polynomial) convolutional processes by stacking multiple linear layers as in Figure 3.1.

### 3.5.2 Real Data

**HCP-YA neuroimaging dataset.** HCP represents a unifying paradigm to acquire high quality neuroimaging data across studies that enabled unprecedented quality of macro-scale human brain connectomes for analyses in different domains [78]. We use the dMRI and resting state fMRI data from the HCP-YA dataset [79], consisting of 1200 healthy, young adults (ages: 22-36 years). The SC and FC are projected on a common brain atlas, which is a grouping of cortical structures in the brain to distinct regions. We interpret these regions as nodes in a brain graph. For our experiments, we use the standard Desikan-Killiany atlas [80] with $N = 68$ cortical brain regions. The

SC-FC coupling on this dataset is known to be the strongest in the occipital lobe and vary with age, sex and cognitive health in other subnetworks [71]. Under the consideration of the variability in SC-FC coupling across the brain regions, we further group the cortical regions into 4 neurologically defined 'lobes': frontal, parietal, temporal, and occipital. We predict SC, derived from dMRI, using FC, constructed using BOLD signals acquired with fMRI. In the literature, the forward (SC to FC) problem has been mostly attempted, a non-trivial yet simpler task due to the known reliance of function on structure in many cortical regions. We tackle the markedly harder inverse (deconvolution) problem of recovering SC from FC, which has so far received less attention and can have major impact as described in Section 3.3. Notably, this network deconvolution problem represents an ideal fit to the polynomial model (3.1) relating FC with SC [54, 55], plus the supervised setting is well motivated given the availability of functional and structural connectomes from the HCP-YA repository.

From this data, we extracted a dataset of 1063 FC-SC pairs, $\mathcal{T} = \{\boldsymbol{FC}^{(i)}, \boldsymbol{SC}^{(i)}\}_{i=1}^{1063}$ and use a training/validation/test split of 913/50/100. Taking the prior $\boldsymbol{A}[0]$ as the edgewise mean over all SCs in the training split $\mathcal{T}_{train}$: $A[0]_{ij} = \text{mean } \{SC_{i,j}^{(1)}, \ldots, SC_{i,j}^{(913)}\}$ and using it directly as a predictor on the test set (tuning a threshold with validation data), we achieve strong performance on link-prediction and edge-weight regression tasks on the whole brain (error = 12.3%, MAE = 0.0615). At the lobe level, the prior achieves relatively higher accuracy in occipital (error = 1.3%, MAE = 0.064) and

parietal (error = 6.6%, MAE = 0.07) lobes as compared to temporal (error = 11.0%, MAE = 0.053) and frontal (error = 10.5%, MAE = 0.062) lobes; behavior which is unsurprising as SC in temporal and frontal lobes are affected by ageing and gender related variability in the dataset [81]. GDNs reduced MAE by 7.6%, 7.1%, 1.6%, and 1.3% in the temporal, frontal, parietal, and occipital networks respectively and 8.0% over the entire brain network, all relative to the already strong mean prior. The four lobe reductions are visualized in Figure 3.4. We observed the most significant gains over temporal and frontal lobes; clearly there was smaller room to improve performance over the occipital and frontal lobes. The resulting MAEs for the entire brain network for L2G, GLAD, GCN-D, and GIN-D are 2.23e-1, 2.07e-1, 2.23e-1, 2.10e-1 respectively; none were able to reduce MAE (or error).

In summary, our 'pseudo-synthetic' experiments in Section 3.5.1 show that SCs are amenable to learning with GDNs when the SC-FC relationship satisfies eq. (3.1), a reasonable model given the findings of [54, 55]. In general, SC-FC coupling can vary widely across both the population and within the sub-regions of an individuals brain for healthy subjects and in pathological contexts. When trained on the HCP-YA dataset, GDNs exhibit robust performance over such regions with high variability in SC-FC coupling. Therefore, our results on HCP-YA dataset justify GDNs as the leading baseline model for comparisons in this significant FC to SC task. It could also potentially serve as a baseline for characterizing healthy subjects in pathology studies in future work, where pathology could be characterized by specific

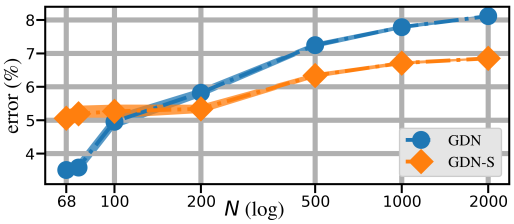


Figure 3.3: Size Generalization: GDNs maintain performance on RG graphs orders of magnitude larger than the $N = 68$ training set.

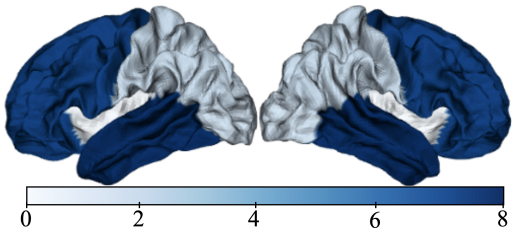


Figure 3.4: Reduction in MAE (%) for different lobes; largest improvements occur in temporal and frontal lobes.

deviations from our results.

**Friendship recommendation from physical co-location networks.** Here we use GDNs to predict Facebook ties given human co-location data. GDNs are well suited for this deconvolution problem since one can view friendships as direct ties, whereas co-location edges include indirect relationships due to casual encounters in addition to face-to-face contacts with friends. In terms of impact, a trained model could then be useful to augment friendship recommendation engines given co-location (behavioral) data. Granted, this would also pose interesting privacy questions that are beyond the scope of this paper, and certainly call for tighter control of co-location information.

The Thiers13 dataset [82] monitored high school students, recording: (i) physical co-location interactions with wearable sensors over 5 days; and (ii) social network information via survey. From this we construct a dataset $\mathcal{T}$ of graph pairs, each with $N = 120$ nodes, where $\boldsymbol{A}_O$ are co-location networks, i.e., weighted graphs where the weight of edge $(i, j)$ represents the number of times student $i$ and $j$ came into physical proximity, and

$\boldsymbol{A}_L$ is the Facebook subgraph between the same students. We trained a GDN of depth $D = 11$ without MIMO filters ($C = 1$), with learned $\boldsymbol{A}[0]$ using a training/validation/test split of 5000/1000/1000. We achieved a test error of $8.9 \pm 1.5e$-2%, a 13.0% reduction over next best performing baseline, suggesting we learn a latent mechanism that the baselines (even GLAD, a supervised graph learning method) cannot. The performance of the baselines are as follows: Threshold ($10.2 \pm 1.2e$-2%), ND ($10.2 \pm 1.2e$-2%), GLASSO (NA - could not converge for any of the $\alpha$ values tested), SpecTemp ($10.7 \pm 6.9e$-2%), GLAD ($14.7 \pm 1.3e$-2%), GCN-C ($11.7 \pm 3.7e$-4%), and GIN-D ($11.5 \pm 4.0e$-4%).

Table 3.1: Mean and standard error of the test performance across both tasks (Top: link-prediction, Bottom: edge-weight regression) on each graph domain. Bold denotes best performance.

| | Models | RG | ER | BA | SC |
|---|---|---|---|---|---|
| **Error (%)** | GDN | **4.6**$_{\pm 4e\text{-}1}$ | 41.9$_{\pm 1e\text{-}1}$ | **27.5**$_{\pm 1e\text{-}3}$ | **8.9**$_{\pm 2e\text{-}2}$ |
| | GDN-S | 5.5$_{\pm 2e\text{-}1}$ | **40.8**$_{\pm 1e\text{-}2}$ | 27.6$_{\pm 8e\text{-}4}$ | 9.4$_{\pm 2e\text{-}1}$ |
| | L2G | 44.3$_{\pm 3e\text{-}4}$ | 43.9$_{\pm 2e\text{-}5}$ | 34.9$_{\pm 1e\text{-}5}$ | 31.7$_{\pm 3e\text{-}1}$ |
| | GCN-D | 30.7$_{\pm 3e\text{-}4}$ | 41.8$_{\pm 1e\text{-}5}$ | 27.6$_{\pm 5e\text{-}5}$ | 31.6$_{\pm 9e\text{-}5}$ |
| | GIN-D | 20.0$_{\pm 2e\text{-}3}$ | 43.6$_{\pm 2e\text{-}3}$ | 28.3$_{\pm 8e\text{-}4}$ | 33.3$_{\pm 8e\text{-}3}$ |
| | GLAD | 6.3$_{\pm 2e\text{-}1}$ | 43.7$_{\pm 3e\text{-}1}$ | 35.0$_{\pm 3e\text{-}2}$ | 11.8$_{\pm 2e\text{-}3}$ |
| | GLASSO | 8.8$_{\pm 7e\text{-}2}$ | 43.2$_{\pm 1e\text{-}2}$ | 34.9$_{\pm 9e\text{-}3}$ | 20.0$_{\pm 4e\text{-}2}$ |
| | ND | 9.4$_{\pm 3e\text{-}1}$ | 43.9$_{\pm 1e\text{-}2}$ | 34.1$_{\pm 8e\text{-}3}$ | 21.3$_{\pm 9e\text{-}2}$ |
| | SpecTemp | 11.1$_{\pm 3e\text{-}1}$ | 44.4$_{\pm 7e\text{-}2}$ | 30.2$_{\pm 2e\text{-}1}$ | 30.0$_{\pm 1e\text{-}1}$ |
| | LSOpt | 24.2$_{\pm 5e\text{-}0}$ | 42.5$_{\pm 3e\text{-}1}$ | 28.0$_{\pm 2e\text{-}1}$ | 31.5$_{\pm 6e\text{-}3}$ |
| | Threshold | 26.8$_{\pm 2e\text{-}1}$ | 42.9$_{\pm 8e\text{-}1}$ | 32.3$_{\pm 1e\text{-}0}$ | 21.7$_{\pm 2e\text{-}1}$ |
| **MSE** | GDN | **4.2**e-**2**$_{\pm 4e\text{-}3}$ | 2.3e-1$_{\pm 2e\text{-}3}$ | **1.8**e-**1**$_{\pm 2e\text{-}3}$ | **5.3**e-**3**$_{\pm 7e\text{-}5}$ |
| | GDN-S | 6.0e-2$_{\pm 2e\text{-}1}$ | **2.3**e-**1**$_{\pm 2e\text{-}3}$ | 2.7e-1$_{\pm 2e\text{-}2}$ | 6.5e-3$_{\pm 4e\text{-}5}$ |
| | L2G | 2.5e-1$_{\pm 4e\text{-}4}$ | 2.5e-1$_{\pm 3e\text{-}5}$ | 2.3e-1$_{\pm 4e\text{-}5}$ | 4.9e-2$_{\pm 3e\text{-}3}$ |
| | GCN-D | 2.1e-1$_{\pm 5e\text{-}6}$ | 2.4e-1$_{\pm 3e\text{-}5}$ | 2.0e-1$_{\pm 1e\text{-}5}$ | 1.1e-1$_{\pm 6e\text{-}5}$ |
| | GIN-D | 2.0e-1$_{\pm 3e\text{-}3}$ | 2.6e-1$_{\pm 5e\text{-}3}$ | 1.9e-1$_{\pm 7e\text{-}4}$ | 2.6e-1$_{\pm 5e\text{-}3}$ |
| | GLAD | 7.8e-2$_{\pm 9e\text{-}4}$ | 2.4e-1$_{\pm 4e\text{-}3}$ | 1.9e-1$_{\pm 4e\text{-}3}$ | 1.4e-2$_{\pm 7e\text{-}6}$ |
| | GLASSO | 2.0e-1$_{\pm 3e\text{-}3}$ | 2.8e-1$_{\pm 2e\text{-}2}$ | 2.6e-1$_{\pm 2e\text{-}2}$ | 4.4e-2$_{\pm 3e\text{-}5}$ |
| | ND | 1.8e-1$_{\pm 2e\text{-}3}$ | 2.4e-1$_{\pm 5e\text{-}4}$ | 2.2e-1$_{\pm 1e\text{-}3}$ | 5.6e-2$_{\pm 7e\text{-}5}$ |
| | SpecTemp | 5.1e-2$_{\pm 3e\text{-}5}$ | 5.3e-1$_{\pm 9e\text{-}5}$ | 3.3e-1$_{\pm 2e\text{-}5}$ | 1.5e-1$_{\pm 4e\text{-}3}$ |
| | LSOpt | 9.9e-2$_{\pm 2e\text{-}1}$ | 2.5e-1$_{\pm 2e\text{-}3}$ | 2.0e-1$_{\pm 3e\text{-}3}$ | 6.1e-0$_{\pm 6e\text{-}4}$ |

Table 3.2: Mean and standard error of the test performance on link prediction task for a varied number $P$ of sampled signals on $N = 68$ RG graphs.

| Models \ P | 3 | 5 | 20 | 50 | 100 | 200 | 1000 |
|---|---|---|---|---|---|---|---|
| GDN | $26.4_{\pm 2e\text{-}3}$ | $17.9_{\pm 2e\text{-}3}$ | $9.6_{\pm 9e\text{-}4}$ | $7.3_{\pm 8e\text{-}4}$ | $6.8_{\pm 8e\text{-}4}$ | $6.4_{\pm 8e\text{-}4}$ | $6.3_{\pm 7e\text{-}4}$ |
| Threshold | $31.8_{\pm 2e\text{-}3}$ | $28.8_{\pm 2e\text{-}3}$ | $27.1_{\pm 1e\text{-}3}$ | $26.8_{\pm 2e\text{-}3}$ | $26.8_{\pm 7e\text{-}4}$ | $26.8_{\pm 7e\text{-}4}$ | $26.8_{\pm 7e\text{-}4}$ |

# Chapter 4

# Graph Structure Learning with Interpretable Bayesian Neural Networks

## 4.1 Introduction

Graphs serve as a foundational paradigm in machine learning, data science, and network science for modeling complex systems, capturing intricate relationships in data that range from social networks to gene interactions; see e.g., [1, 2]. In computational biology, accurate graph structures can offer insights into gene regulatory pathways, enhancing our ability to treat diseases at the genetic level [3]. In finance, the ability to recover precise financial networks can be useful for risk assessment and market stability [4]. In geometric deep learning [83], the lack of observed graph structure underlying data often limits the use of efficient learning models such as graph neural networks (GNNs); see e.g., [84]. Yet, despite their importance, existing methodologies for graph structure learning (GSL) from multivariate nodal data face significant limitations.

This work addresses the GSL problem where nodal observations are used to predict the completely unobserved graph structure. Traditional approaches summarize the nodal observations with a pairwise vertex (dis)similarity matrix, e.g., correlation or Euclidean distance matrices, and formulate GSL as a regularized convex inverse problem [11, 12]. The primary objective encourages data fidelity, according to a chosen model linking the nodal observations and the sought latent graph, while the regularization objectives capture prior structural knowledge, such as edge sparsity or desired connectivity patterns. These *model-based* methods solve the inverse problem using optimization algorithms, which often come with convergence rate guarantees [85, 86]. However, as noted in [87] their expressiveness is constrained to graph characteristics that can be modeled via convex criteria and they may suffer complexity and scalability issues. Specifically, these model-based approaches typically necessitate an outer loop for grid-based optimization of regularization parameters (and often other algorithm parameters, e.g., a step-size), and an inner loop that, given this fully defined optimization problem, may demand thousands of iterations to converge on a solution. When nodal observations come with corresponding graph labels, recent supervised GSL approaches partially overcome these obstacles using 'algorithm unrolling' [25, 87, 88]. Unrollings truncate these inner-loop iterations to yield a deep network architecture that is trained end-to-end to approximate the solution to the inverse problem [36]. The choice of a custom loss function, which need not be convex, plus optional architectural refinements of an already well-motivated initial network tend

to improve performance on the task of interest. Truncation depth provides explicit control on the complexity of prediction, now a forward pass in the network. And the use of backpropogated gradients makes learning the regularization (and step-size) parameters, now network parameters, more scalable. Additionally, the unrolled deep networks tend to inherit appealing aspects from the original model-based formulation, namely inductive bias and low dimensionality, as their outputs are approximations to solutions of the original inverse problem.

### 4.1.1 Towards interpretability and uncertainty quantification: Desiderata and contributions

In composite inverse problems with multiple regularizers, understanding how each regularization parameter values affect desirable solution traits (e.g., image sharpness or number of graph edges) may be complex and often beyond the modeler's knowledge. This obscurity necessitates a naive—and consequently costly—multi-dimensioanl grid search for suitable parameter values; see e.g., the numerical study in [59, Section V-B]. When the relationship between the regularization parameters and a solution characteristic is clear, we say the parameters are interpretable with respect to (w.r.t.) the output characteristic. When an output characteristic is influenced by a single parameter, independent of all others, we call this parameter *independently interpretable* w.r.t. the output characteristic. Such instances make interpretability actionable, namely they allow prior knowledge on the solution characteristic to be

incorporated onto the value of the independently interpretable parameter. A key contribution of this work is: i) in recognizing optimization problems with independently interpretable parameters as ripe for the adoption of unrollings, which inherit this interpretability because they approximate inverse problem solutions; and ii) in leveraging the Bayesian framework to seamlessly incorporate such prior knowledge into the parameters of the unrolled network.

An issue unrollings often face (within GSL and beyond) is their tendency to produce layers with pre-activation outputs that are nonlinear functions of the parameters; e.g., parameter products and parameters in denominators [36]. This prevents the deep network from being a *true neural network* (NN), i.e., a function composed of layers, where each layer consists of affine transformations of data or intermediate activations followed by non-linear functions applied pointwise [89]. To address this issue, network designers may opt for reparameterization, but at the expense of parameter interpretability and often degraded empirical performance [36,88]. Unrollings can be impeded by nuisance parameters - parameters not of direct interest, but which must be accounted for - like a step-size. Nuisance parameters are typically a vestige of the chosen optimization algorithm rather than being intrinsic to the problem formulation. Nuissance parameters can also undermine training stability and since they lack a clear connection to specific solution characteristics they hinder informative prior modeling. The preceding discussion motivates the need for innovative GSL techniques to produce true NN unrollings amenable to incorporation of prior knowledge on their parameters.

Providing estimates of uncertainty on an inferred graph structure is important for downstream applications, e.g. in biology, finance, and machine learning with GNNs. In general Bayesian statistics, low-dimensional models with interpretable parameters are constructed using background information on the specific problem. Interpretability allows informative prior modeling and thus access to enticing Bayesian modeling tools, e.g., prior predictive checks [90], while the low dimensionality allows tractable high-quality posterior inference, typically via Markov Chain Monte Carlo (MCMC) sampling. For example, traditional Bayesian approaches to GSL tend to address the transductive (tied to a single graph) setting by constructing a joint model over parameters, observed data, and latent graph structure, use MCMC to draw samples, and marginalize out all but the graph samples [91–93]. Stepping back from the specific case of GSL, the inability to specify a sufficiently expressive model often motivates the use of Bayesian neural networks (BNNs), an alternative paradigm which typically bypasses domain-specific modeling opting instead to feed the output of a performant NN directly into the likelihood function; see e.g., [94]. Both traditional Bayesian and BNN approaches derive uncertainty estimates over predictions by marginalizing out the parameter posterior. However, BNNs present notable challenges, especially in the incorporation of prior information and posterior approximation. Due to the non-interpretability of parameters, priors are often selected for computational convenience, such as a zero-mean isotropic Gaussian, which can inadvertently bias predictions *against* prior beliefs, a phenomenon known

as 'unintentionally informative priors' [95], thus negating a key benefit of Bayesian methods. Additionally, the parameters' high dimensionality and complex posterior geometry make high-quality posterior approximation a formidable task, often requiring significant approximations that undermine the interpretability of the results [36, 95, 96]. This highlights the need for an inductive GSL method that not only provides reliable uncertainty estimates over edge predictions but also combines the expressive power of BNNs with the traditional Bayesian approach's strengths in integrating prior knowledge, employing robust modeling tools like predictive checks, and ensuring high-fidelity posterior approximation.

**Summary of contributions.** In this paper, we introduce the first BNN for supervised GSL from smooth signal observations. This BNN produces a distribution over unseen test graphs allowing estimation of uncertainty over edge predictions. It leverages the independent intepretability of the paramters in the GSL formulation to allow informative prior modeling over the weights of the NN, itself a result of unrolling novel iterations for a model-based formulation with well-documented merits. We make the following technical contributions and offer experimental evidence to support our claims:

- In Section 4.3, we develop a novel optimization algorithm for GSL from smooth signals (Algorithm 1), which is step-size free and parameterized to yield independent interpretability w.r.t. edge sparsity.

- In Section 4.4, we unroll Algorithm 1 to produce the first strict unrolling

for GSL from smooth signals, which results in a true NN. This is to be contrasted with existing supervised-learning approaches to GSL, where GLAD [88] and Unrolled PDS [87] are not true NNs, and GDN (see Chapter 3) is not a strict unrolling because it resorts to gradient truncation and reparameterization. The proposed unrolled NN is used to define our BNN, dubbed 'DPG' since Algorithm 1 is a dual-based proximal gradient method [97].

- In Section 4.5, we introduce a methodology to integrate prior knowledge into BNN prior distributions, specifically for networks derived from unrolled optimization algorithms for inverse problems with independent interpretability. We show this approach unlocks classical Bayesian modeling tools like predictive checking, which we fruitfully apply to DPG. High-fidelity parameter posterior inference via Hamiltonian Monte Carlo (HMC) sampling enables the first instance of a model for GSL from smooth signals, capable of producing estimates of uncertainty over edge predictions.

- In Section 4.6, we validate DPG's ability to produce high-quality and well-calibrated uncertainty estimates from synthetic data, stock price time series (S&P500), as well as graphs learnt from images of MNIST digits. The reliability of these estimates is underscored by notable Pearson correlations between predictive uncertainty and error: 0.70 for the stock data and 0.62 for the MNIST digits. Additional experimental

evidence is provided in the appendices.

## 4.2 Related Work

A recent body of work addresses the GSL task via algorithm unrollings under varying data assumptions. Noteworthy contributions include GLAD [88], which unrolls alternating-minimization iterations for Gaussian graphical model selection, Unrolled PDS [87] which unrolls primal-dual splitting (PDS) iterations for the GSL problem from smooth signals, and GDN (see Chapter 3) which unrolls linearized proximal-gradient iterations for a network deconvolution task that posits a polynomial relationship between the observed pairwise vertex distance matrix and the latent graph. Here we deal with the GSL problem from observations of smooth signals as in [87], but the optimization algorithm we unroll is different (cf. DPG in Algorithm 1 versus PDS), more compact and devoid of step-sizes. Additionally, none of the previous GSL unrollings result in true NNs, provide estimates of uncertainty on their adjacency matrix predictions, nor leverage the interpretability of network parameters in the modeling process.

Building a probabilistic model to connect observed network data to latent graph structure has a long history in the computer and social network analysis communities; see e.g., [1, 91, 98]. Gibbs sampling was used mostly as a means to efficiently explore the resulting network posterior rather than quantify the uncertainty the model placed on particular edges. Since then, Bayesian

methods have been used with alternate network models (e.g., directed acyclic graphs specifying the structure of Bayesian networks), types of observed data (e.g., information cascades and protein-protein interactions), and posterior approximation approaches; see [93, 99–103]. Such approaches are typically transductive - tying themselves to a single training graph - and require expensive *joint* inference of the model and the latent graph structure. Our approach only requires inference of the BNN model parameters and thus is naturally inductive, i.e., able to generalize to new nodes, or entirely new graphs. Some recent works build on such graph distribution modeling approaches in an approximate Bayesian manner, to incorporate the uncertainty in an observed graph for downstream tasks with GNNs [104, 105]. Others forgo modeling the distribution of the observed graph but take approximate Bayesian approaches to modeling with GNNs. For instance, [106] uses a deep graph convolutional Gaussian process with variational posterior approximation for link prediction, and [107] pre-trains a score-matching GNN for use in annealed Langevin diffusion to draw approximate samples from the network posterior. All such Bayesian GNN approaches require (partial) observation of graph structure, and rely on approximate inference methods due to large dimensionality. Our DPG approach uses no observed graph structure (except for graph labels during training) and allows for high-fidelity posterior approximation. To the best of our knowledge, BNNs have so far not been used for GSL with uncertainty quantification.

More broadly, unrolling-inspired Bayesian deep networks have recently

found success in uncertainty quantification for computational imaging [108–110]. The inductive bias provided by the original iterations lead to gains in data efficiency, but still have limited parameter interpretability and high dimensionality leading to naive priors and coarse posterior approximation. This exciting line of work inspired some crucial ideas in this paper, cross-pollinating benefits to GSL and with the added value of overcoming the aforementioned Bayesian modeling and inference challenges.

## 4.3 Model-based Formulation and Optimization Preliminaries

Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{A})$ be an undirected graph, where $\mathcal{V} = \{1, \dots N\}$ are the vertices (or nodes), $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the edges, and $\boldsymbol{A} \in \mathbb{R}_{+}^{N \times N}$ is the symmetric adjacency matrix collecting the non-negative edge weights. For $(i, j) \notin \mathcal{E}$ we have $A_{ij} = 0$. We exclude the possibility of self loops, so that $\boldsymbol{A}$ is hollow meaning $A_{ii} = 0$, for all $i \in \mathcal{V}$. For the special case of unweighted graphs that will be prominent in our models, then $\boldsymbol{A} \in \{0, 1\}^{N \times N}$.

In this paper, we consider that $\boldsymbol{A}$ is unknown and we want to estimate the latent graph structure from nodal measurements only[1]. To this end, we acquire graph signal observations $\boldsymbol{x} = [x_1, \dots, x_N]^\top \in \mathbb{R}^N$, where $x_i$ denotes the signal value (i.e., a nodal attribute or feature) at vertex $i \in \mathcal{V}$. When

[1]This is different to the link prediction task, where one is given measurements of edge status for a training subset of node pairs (plus, optionally, node attributes), and the transductive goal is to predict test links from the same graph [1]

$P$ such signals are available we construct matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P] \in \mathbb{R}^{N \times P}$, where each row $\bar{\boldsymbol{x}}_i^\top \in \mathbb{R}^P$, $i = 1, \ldots, N$, of $\boldsymbol{X}$ represents a vector of features or nodal attributes at vertex $i$. We can summarize this dataset using a pairwise vertex dissimilarity matrix, here the Euclidean distance matrix $\boldsymbol{E} \in \mathbb{R}_+^{N \times N}$, where $E_{ij} = \|\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_j\|_2^2$. Assuming our data lie on a smooth manifold, we interpret $\mathcal{G}$ as a discrete representation of this manifold. When nodes $i \neq j \in \mathcal{V}$ have large edge weight $A_{ij}$, reflecting close points on the manifold, $E_{ij}$ will be small. Accordingly, smooth (w.r.t. $\mathcal{G}$) vectors in $\boldsymbol{X}$ have small total variation or Dirichlet energy [111], namely

$$TV_{\mathcal{G}}(\boldsymbol{X}) = \frac{1}{2} \sum_{i,j} A_{ij} \|\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_j\|_2^2 = \|\boldsymbol{A} \circ \boldsymbol{E}\|_{1,1}, \tag{4.1}$$

where $\|\boldsymbol{Z}\|_{1,1} = \sum_{i,j} |Z_{ij}|$ is the entrywise $\ell_1$-norm of matrix $\boldsymbol{Z}$ and $\circ$ is the Hadamard (entrywise) product. The prevalence of smooth network data, for instance sensor measurements [112], protein function annotations [1], and product ratings [113], justifies using a smoothness criterion for the GSL task.

### 4.3.1 Graph structure learning from smooth signals

Given $\boldsymbol{X}$ assumed to be smooth on $\mathcal{G}$, a popular model-based GSL approach is to minimize the Dirichlet energy in (4.1) w.r.t. $\boldsymbol{A}$; see e.g., [19, 22, 59, 114]. The inverse problems posed in these works can be unified under the general

composite formulation

$$\boldsymbol{A}^* = \underset{\boldsymbol{A}\in\mathcal{A}}{\operatorname{argmin}} \left\{ \|\boldsymbol{A} \circ \boldsymbol{E}\|_{1,1} + h(\boldsymbol{A}) \right\}, \tag{4.2}$$

where the feasible set is $\mathcal{A} := \{\boldsymbol{A} \in \mathbb{R}^{N\times N} : \operatorname{diag}(\boldsymbol{A}) = \boldsymbol{0}, A_{ij} = A_{ji} \geq 0, \forall i, j \in \mathcal{V}\}$, i.e., hollow, symmetric, non-negative matrices. The regularization term $h(\boldsymbol{A})$ typically promotes desired structure on the estimated edge set (e.g., sparsity, no isolated nodes) and can be used to avoid the trivial solution $\boldsymbol{A}^* = \boldsymbol{0}$. We henceforth use $h(\boldsymbol{A}) = -\alpha \boldsymbol{1}^\top \log(\boldsymbol{A}\boldsymbol{1}) + \frac{\beta}{2}\|\boldsymbol{A}\|_F^2$ ($\alpha, \beta \geq 0$ are regularization parameters), which excludes the possibility of isolated nodes and has achieved state-of-the-art results [19].

It is convenient to reformulate (4.2) in an unconstrained, yet equivalent form. We start by compactly representing variable $\boldsymbol{A}$ and data matrix $\boldsymbol{E}$ with their vectorized upper triangular parts $\boldsymbol{a}, \boldsymbol{e} \in \mathbb{R}_+^{N(N-1)/2}$, implicitly enforcing symmetry and hollowness, while also halving the problem dimension. To enforce non-negativity the indicator function $\mathbb{I}\{\boldsymbol{a} \geq 0\} = \{0 \text{ if } \boldsymbol{a} \geq 0 \text{ else } \infty\}$ is included in the objective. Finally, we substitute the nodal degrees $\boldsymbol{d} = \boldsymbol{A}\boldsymbol{1}$ with the vectorized equivalent $\boldsymbol{d} = \boldsymbol{S}\boldsymbol{a}$, where $\boldsymbol{S} \in \{0,1\}^{N\times N(N-1)/2}$ is a fixed binary matrix that maps vectorized edge weights to degrees. The resulting optimization problem is given by

$$\boldsymbol{a}^*(\boldsymbol{e}, \alpha, \beta) = \underset{\boldsymbol{a}\in\mathbb{R}^{N(N-1)/2}}{\operatorname{argmin}} \left\{ 2\boldsymbol{a}^\top \boldsymbol{e} - \alpha \boldsymbol{1}^\top \log(\boldsymbol{S}\boldsymbol{a}) + \frac{\beta}{2}\|\boldsymbol{a}\|_2^2 + \mathbb{I}\{\boldsymbol{a} \geq 0\} \right\}, \tag{4.3}$$

which is convex and admits a unique optimal solution; see e.g., [85]. Next, we comment on the role of the regularization parameters $\alpha, \beta$ and their interpretability properties. We then offer a brief discussion on optimization algorithms to tackle problem (4.3). These ingredients will be essential to build a BNN model for supervised GSL in Sections 4.4 and 4.5.

**Independent interpretability of regularization parameters.** Definition 1 formalizes the notion of independent interpretability of regularization parameters in inverse problems such as (4.3).

**Definition 1.** Let $\boldsymbol{x}^*(\mu_1, \ldots, \mu_n) \in \mathcal{X}$ be the solution to an inverse problem that depends on regularization parameters $\mu_1, \ldots, \mu_n$. Consider some scalar function of the solution $f : \mathcal{X} \mapsto \mathbb{R}$. In general, the value $f(\boldsymbol{x}^*)$ depends on all $\mu_1, \ldots, \mu_n$. When $f(\boldsymbol{x}^*)$ depends solely on a *single* regularization parameter $\mu_i$, then we say that $\mu_i$ is *independently interpretable* w.r.t. $f(\boldsymbol{x}^*)$.

The weights $\alpha$ and $\beta$ are not independently interpretable w.r.t. to relevant graph characteristics, frustrating straightforward interpretation of their effect on the solution $\boldsymbol{a}^*(\boldsymbol{e}, \alpha, \beta)$. Specifically, for fixed $\alpha$, increasing $\beta$ leads to denser edge patterns, as we have (quadratically) increased the relative cost of large edge weights. Indeed, the sparsest graph is obtained for $\beta = 0$. But in general, many interesting graph characteristics, e.g., sparsity, connectivity, diameter, and edge weight magnitude, are non-trivial functions of *both* $\alpha$ and $\beta$; see also [59] for a similar issue.

To facilitate *independent* control over the sparsity pattern and scale of the edge weights of recovered graphs, [19, Prop. 2] introduced an equivalent

$(\theta, \delta)$-parameterization of (4.3), namely

$$\boldsymbol{a}^*(\boldsymbol{e}, \alpha, \beta) = \sqrt{\frac{\alpha}{\beta}}\boldsymbol{a}^*\left(\frac{1}{\sqrt{\alpha\beta}}\boldsymbol{e}, 1, 1\right) = \delta\boldsymbol{a}^*(\theta\boldsymbol{e}, 1, 1). \tag{4.4}$$

We can map from the former parameterization to the latter by first scaling $\boldsymbol{e}$ by $\theta = 1/\sqrt{\alpha\beta}$, solving (4.3) with $\theta\boldsymbol{e}$ using $\alpha = \beta = 1$, and finally scaling the recovered edges by the constant $\delta = \sqrt{\alpha/\beta}$; we refer the reader to [19, 22] for a proof of the equivalence claim. Due to the separable structure of the right-hand-side of (4.4), any GSL algorithm would require a single input parameter $\theta$, and the obtained solution $\boldsymbol{a}^*(\theta\boldsymbol{e}, 1, 1)$ can then be scaled by $\delta > 0$. All in all, the sparsity level of $\boldsymbol{a}^*$ is determined solely by $\theta$, making $\theta$ independently interpretable w.r.t. sparsity. Indeed, this satisfies Definition 1 with the identifications $\boldsymbol{x}^*(\mu_1, \ldots, \mu_n) \leftarrow \delta\boldsymbol{a}^*(\theta\boldsymbol{e}, 1, 1)$, $\mathcal{X} \leftarrow \mathbb{R}_+^{N(N-1)/2}$, $\mu_i \leftarrow \theta$, and $f(\boldsymbol{x}^*) \leftarrow \|\delta\boldsymbol{a}^*(\theta\boldsymbol{e}, 1, 1)\|_0$, where $\|\cdot\|_0$ counts the number of non-zero elements of its vector argument. Moreover, $\delta$ is interpretable w.r.t. edge weight magnitude, but not independently so, as larger $\theta$ produces smaller weights [see Figure 4.3 (bottom-left)].

### 4.3.2 Optimization algorithms

Problem (4.3) has a favorable structure that has been well documented, and several efficient optimization algorithms were proposed to obtain a solution $\boldsymbol{a}^*(\boldsymbol{e}, \alpha, \beta)$ with $\mathcal{O}(N^2)$ complexity per iteration. Specifically, a forward-backward-forward PDS algorithm was first proposed in [19]. PDS introduces

a step-size parameter which must be tuned to yield satisfactory empirical convergence properties, thus increasing the overall computational burden. We find that effective step-size values tend to lie on a narrow interval beyond which PDS exhibits divergent behavior, further frustrating tuning. GSL algorithms based on the alternating-directions method of multipliers [115] or majorization-minimization [116] have been developed as well. Recently, [85] introduced a fast dual proximal gradient (FDPG) algorithm to solve (4.3), which is devoid of step-size parameters and – different from all three previous approaches – it comes with global convergence rate guarantees. For this problem, the strongest convergence results to date are in [86].

Our starting point in this work is the FDPG optimization framework, but different from [85] we: (i) develop a solver for the $(\theta, \delta)$-parameterization of (4.3); and (ii) turn-off the Nesterov-type acceleration from the proximal-gradient iterations used to solve the dual problem of (4.3). This yields a dual proximal gradient (DPG) method, tabulated under Algorithm 1. In a nutshell, during iterations $k = 1, 2, \ldots$ Algorithm 1 updates the vectorized adjacency matrix estimate $\boldsymbol{a}_k \in \mathbb{R}_+^{N(N-1)/2}$, an auxiliary vector of nodal degrees $\boldsymbol{d}_k \in \mathbb{R}_+^N$, as well as dual variables $\boldsymbol{\lambda}_k \in \mathbb{R}^N$ used to enforce the variable splitting constraint $\boldsymbol{d} = \boldsymbol{S}\boldsymbol{a}$. A naive DPG implementation incurs $\mathcal{O}(N^2)$ computational and memory complexities, and we note all nonlinear operations involved (i.e., $\text{ReLU}(\cdot) = \max(0, \cdot)$, $(\cdot)^2$, and $\sqrt{(\cdot)}$) are pointwise on their vector arguments. As a result of the design choices (i)-(ii), the DPG algorithm requires the fewest operations per iteration and the fewest

number of parameters among existing solvers of (4.3), and is devoid of any uninterpretable nuisance parameters, e.g., step-sizes. FDPG was only considered on the original $(\alpha, \beta)$-parameterization of (4.3); by instead opting for DPG iterations to solve the $(\theta, \delta)$-parameterization of (4.3), we reveal independent interpretability of $\theta$ w.r.t. sparsity of the optimal graphs.

Next, we will unroll Algorithm 1 to produce a GSL NN which inherits its advantages - namely simple, efficient, minimally parameterized layers, with independent interpretability - forming the backbone of our BNN.

## 4.4 Graph Structure Learning from Smooth Signals with Bayesian Neural Networks

So far we have described a model-based approach to (point) estimation of graphs from smooth signals. In this work, we assume a labeled training dataset is available. We aim to construct a BNN model to produce uncertainty estimates on graph predictions for unseen test data.

**Our BNN approach for GSL in a nutshell.** Here, we restrict ourselves to binary graphs $\boldsymbol{a} \in \{0,1\}^{N(N-1)/2}$; weighted graphs only require a change to the ensuing likelihood function. We denote all training data as $\mathcal{T} = \{\mathcal{T}_e, \mathcal{T}_a\} = \{\boldsymbol{e}^{(t)}, \boldsymbol{a}^{(t)}\}_{t=1}^{T}$, an unseen test sample as $(\tilde{\boldsymbol{e}}, \tilde{\boldsymbol{a}})$, and the collection of all parameters of our yet to be defined BNN model as $\boldsymbol{\Theta}$. Following the principles of BNNs, our goal is to construct a posterior predictive distribution

**Algorithm 1** Dual Proximal Gradient Descent

**Require:** Fixed parameters $\theta, \delta \in \mathbb{R}$ and data $\boldsymbol{e}$

**Initialize:** $\boldsymbol{a}_0$ and $\boldsymbol{\lambda}_0$ at random.

**for** $k = 1, 2, \ldots$ **do**

$$\boldsymbol{d}_k = \boldsymbol{S}\boldsymbol{a}_{k-1} - (N-1)\boldsymbol{\lambda}_{k-1}$$

$$\boldsymbol{\lambda}_k = \tfrac{-1}{2(N-1)}\left(\boldsymbol{d}_k - \sqrt{\boldsymbol{d}_k{}^2 + 4(N-1)\mathbf{1}}\,\right)$$

$$\boldsymbol{a}_k = \max\left(\mathbf{0}, \tfrac{1}{2}\boldsymbol{S}^\top\boldsymbol{\lambda}_k - \theta\boldsymbol{e}\right)$$

**end for**

**Return:** $\delta\boldsymbol{a}_k$

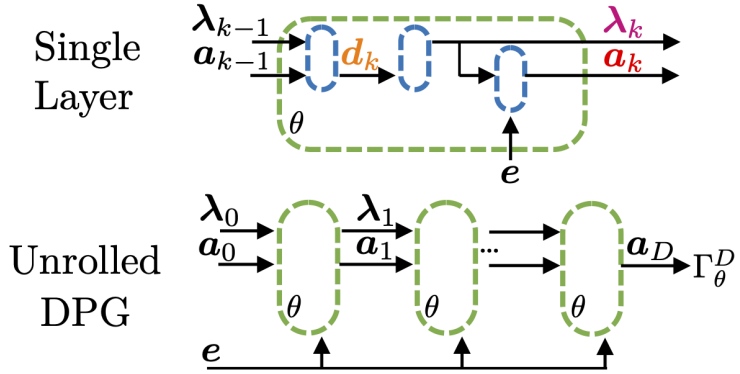


Figure 4.1: *Top:* The dual proximal gradient (DPG) algorithm to solve the GSL problem (4.3). *Bottom:* Unrolling and truncating Algorithm 1 after $D$ iterations produces Unrolled DPG.

$p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \mathcal{T})$ by marginalizing out model parameters $\boldsymbol{\Theta}$, i.e.,

$$p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \mathcal{T}) = \int p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \boldsymbol{\Theta}) \cdot p(\boldsymbol{\Theta} \mid \mathcal{T})\, \mathrm{d}\boldsymbol{\Theta} \approx \frac{1}{M}\sum_{m=1}^{M} p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \boldsymbol{\Theta}^{(m)}), \quad (4.5)$$

where we use $M$ Monte Carlo draws from the posterior $\boldsymbol{\Theta}^{(m)} \sim p(\boldsymbol{\Theta} \mid \mathcal{T})$ to approximate the intractable integral. We can then designate our edge-wise point and uncertainty estimates as the first two moments of the posterior

predictive marginals $p(\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T})$, respectively, for each candidate edge indexed by $i \in \mathcal{V} \times \mathcal{V}$.

**Roadmap.** In Section 4.4.1, we first discuss the development of a GSL NN which takes in the vectorized Euclidean distance matrix $\boldsymbol{e}$ of nodal observations $\boldsymbol{X}$ and assigns a probability to all possible edges. We do so by unrolling Algorithm 1, producing an GSL NN with an independently interpretable parameter w.r.t. sparsity of graph outputs. Treating the model parameters of this unrolled NN as stochastic, choosing a likelihood function (Section 4.4.2), and setting appropriate priors (Section 4.5), we produce a BNN. For inference we use HMC (Section 4.4.3), an unusual choice in BNNs made viable by the low dimensionality and fast execution of our NN, stemming from its origins as an unrolling. Pushing the test inputs through each such GSL NN and averaging the resulting predictive distributions provides provides the approximate (4.5) posterior predictive, from which we derive edge-wise point and uncertainty estimates as elaborated in Section 4.4.4.

### 4.4.1 Algorithm unrolling: Iterative optimization as a neural network blueprint

Unrollings (also known as deep unfoldings) use iterative algorithms as templates of neural architectures that are trained to approximate solutions of optimization problems. Iterations are truncated and mapped to NN layers, while optimization (e.g., regularization and step-size) parameters are turned into learnable weights. Originating from convergent iterative procedures,

unrolled NNs inherit many desirable properties; namely, a low number of parameters, fast layer-wise execution, and favorable generalization in low-data environments [36]. Typical architectural design choices include whether to replace intermediate operators with more expressive (possibly pre-trained) learnable modules - making them no longer 'strict' unrollings - and whether a common set of parameters should be used in all layers, or, to decouple these parameters across layers. Both decisions are context dependent and often driven by the amount of data available as well as complexity considerations. Deviating from strict unrollings with shared parameters can naturally lead to gains in expressive power, but often at the cost of inductive bias and training stability.

**Unrolling DPG iterations**. Pioneered by [35] to efficiently learn sparse codes from data, algorithm unrolling ideas are recently gaining traction for GSL as well. Starting from the formulation in Section 4.3.1, [87] unrolls a PDS solver of the $(\alpha, \beta)$-parameterization (4.3). Unrolled PDS has no independently interpretable parameters, is not a true NN (pre-activation outputs are nonlinear functions of the parameters), and includes a nuisance step-size parameter – arguably a shortcoming as gradient-based learning can easily produce large-enough weight values for divergent behavior, and thus NaN's. Attempting to unroll PDS iterates for the $(\theta, \delta)$-parameterization of (4.3) would not fix these practical problems.

We instead advocate unrolling the DPG iterations (Algorithm 1) developed to solve the $(\theta, \delta)$-parameterization of (4.3). This way, we obtain a *true NN*

*without nuisance step-size parameters* - avoiding the aforementioned issues and reducing the parameter count by a third - while inheriting independently interpretable parameter $\theta$ (w.r.t. sparsity of graph outputs). Incidentally, layers in Unrolled DPG [depicted in Figure 1 (right)] are markedly simpler and require fewer operations than Unrolled PDS. We denote the output of a $D$-layer unrolling of Algorithm 1 as $\delta\Gamma_\theta^D$. As we would like probabilities over candidate edges, we subtract a learnable mean shift $b$ and drive the output through a sigmoid $\sigma(\cdot)$, producing our desired GSL NN output

$$\hat{\boldsymbol{p}} = \sigma(\delta\Gamma_\theta^D(\boldsymbol{e}) - b\mathbf{1}) \in (0,1)^{N(N-1)/2}, \tag{4.6}$$

with parameters $\boldsymbol{\Theta} = \{\theta, \delta, b\}$. To see that Unrolled DPG is a true NN, note that $\theta$ is only involved in a linear function $\theta\boldsymbol{e}$ of the input data. Likewise, $\delta$ and $b$ are only involved in an affine mapping of the activations $\Gamma_\theta^D(\boldsymbol{e})$. All non-linear operations (specifically squaring, square root, and max) are pointwise functions of intermediate activations. Going back to the design considerations mentioned at the beginning of this section, here we keep the unrolling strict and share parameters across layers to retain independent interpretability of $\theta$, minimize parameter count, and simplify upcoming Bayesian inference. Tradeoffs arising with model expansion using multiple input and output channels per layer are discussed in Section 4.6.1.

All in all, unrolled DPG is the first true NN for GSL from smooth signals, and the first strict unrolling for GSL which produces a true NN. For the

various reasons laid out in the preceding discussion, Unrolled DPG is of independent interest as a new model for point estimation of graph structure in a supervised setting. As we show next, it will be an integral component of the stochastic model used to construct a BNN to facilitate uncertainty quantification for adjacency matrix predictions.

### 4.4.2 Stochastic model

Here we specify a stochastic model for the random variables of interest, namely the binary adjacency matrix $\boldsymbol{a}$, nodal data entering via the Euclidean distance matrix $\boldsymbol{e}$, and the BNN weights $\boldsymbol{\Theta}$. The Bayesian posterior from which we wish to sample satisfies $p(\boldsymbol{\Theta} \mid \mathcal{T}) \propto p(\mathcal{T}_a \mid \mathcal{T}_e, \boldsymbol{\Theta}) p(\boldsymbol{\Theta})$, and a first step in BNN design is to specify the likelihood $p(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{\Theta})$ and the prior $p(\boldsymbol{\Theta})$. An i.i.d. assumption on the training data $\mathcal{T}$ allows the likelihood to factorize over samples $p(\mathcal{T}_a \mid \mathcal{T}_e, \boldsymbol{\Theta}) = \prod_{t=1}^{T} p(\boldsymbol{a}^{(t)} \mid \boldsymbol{e}^{(t)}, \boldsymbol{\Theta})$. Moreover, assuming edges within a graph sample $\boldsymbol{a}^{(t)}$ are mutually conditionally independent given parameters $\boldsymbol{\Theta}$ leads to further likelihood factorization as $p(\mathcal{T}_a \mid \mathcal{T}_e, \boldsymbol{\Theta}) = \prod_{t=1}^{T} \prod_{i=1}^{N(N-1)/2} p(a_i^{(t)} \mid \boldsymbol{e}^{(t)}, \boldsymbol{\Theta})$. We model $a_i \mid \boldsymbol{e}, \boldsymbol{\Theta} \sim \text{Bernoulli}(\hat{p}_i)$, where the success (or edge $i \in \mathcal{V} \times \mathcal{V}$ presence) probability is given by the output of the Unrolled DPG network, i.e., $\hat{p}_i = \sigma\Big(\delta[\Gamma_\theta^D(\boldsymbol{e})]_i - b\Big)$ as in (4.6). Putting all the pieces together, the final expression for the likelihood is

$$p(\mathcal{T}_a \mid \mathcal{T}_e, \boldsymbol{\Theta}) = \prod_{t=1}^{T} \prod_{i=1}^{N(N-1)/2} (\hat{p}_i^{(t)})^{a_i^{(t)}} (1 - \hat{p}_i^{(t)})^{1 - a_i^{(t)}}. \tag{4.7}$$

We reiterate that, crucially, the Unrolled DPG outputs enter the stochastic model via the likelihood, as the means of the edge distributions. The specification of the prior $p(\boldsymbol{\Theta})$ will be addressed in Section 4.5.

### 4.4.3 Inference

We aim to generate $M$ Monte Carlo draws from the posterior $\boldsymbol{\Theta}^{(m)} \sim p(\boldsymbol{\Theta} \mid \mathcal{T}) \propto p(\mathcal{T}_a \mid \mathcal{T}_e, \boldsymbol{\Theta}) p(\boldsymbol{\Theta})$. In modern BNNs based on overparameterized deep networks, high dimensionality and parameter unidentifiability are prevalent. This translates to highly multi-modal posteriors which make sampling challenging [117]. Even when posterior geometries are more benign, the computational and memory requirements for evaluating the network and its gradients render the generation of Monte Carlo posterior samples impractical. As a typical workaround one can resort to posterior approximation using inexpensive mini-batch methods such as mean-field variational inference or stochastic-gradient MCMC [117]. In contrast, we now argue that generation of high-fidelity Monte Carlo posterior samples using HMC is uniquely compatible with our proposed BNN; for implementation details see Section 4.6.1.

For starters, Unrolled DPG's repetitive layers simplify model construction and allow straightforward compilation in automatic differentiation software, significantly boosting runtime efficiency. The low parameter count of Unrolled DPG facilitates the use of forward-mode auto-differentiation; this eliminates the need to store intermediate activations and perform a backward pass, thus it ensures memory usage does not increase with model depth $D$. Moreover,

as Unrolled DPG can effectively approximate inverse problem solutions with relatively shallow depths (see Section 4.6), selecting a smaller $D$ significantly reduces runtime complexity. For small- to moderately-sized graphs and datasets, our computational and memory requirements for network and gradient evaluation are low; see Section 4.6 for GSL problem instances where inference is attainable in $< 2$ minutes on a M2 MacBook laptop. The limited parameter count also aids in sampling efficiency as we avoid the curse of dimensionality that complicates sampling in higher-dimensional models. Finally, the ability to place informative priors over independently interpretable parameters (as we do in Section 4.5) is known to smoothen the posterior geometry, especially in data scarce regimes where the likelihood and prior are of comparable magnitude [118].

### 4.4.4 Prediction

By conditioning on data $\mathcal{T}$ and integrating out model parameters $\boldsymbol{\Theta}$, we obtain a predictive distribution on unseen graph adjacency matrices $\tilde{\boldsymbol{a}}$ given nodal signals $\tilde{\boldsymbol{e}}$. As the integral is intractable, we use the $M$ posterior samples obtained via HMC in the inference stage to approximate the predictive distribution $p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \mathcal{T})$. This approximation process is summarized in (4.5). .

Importantly, we can generate samples from the posterior predictive by randomly drawing from the sampling distribution with each parameter sample plugged in, i.e., $\tilde{\boldsymbol{a}}^{(m)} \sim p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \boldsymbol{\Theta}^{(m)})$. Given the form of our BNN's stochastic model as introduced in Section 4.4.2, such a draw reduces to sampling a

Bernoulli distribution for each possible edge. Randomly drawing from the sampling distribution is critical as it accounts for both forms of uncertainty in posterior predictive quantities, namely, sampling uncertainty and estimation uncertainty [118]. Using these posterior predictive samples, we can approximate the mean ('pred. mean') and standard deviation ('pred. stdv.') of the edge-wise marginals of the posterior predictive as

$$\mathbb{E}[\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T}] \approx \frac{1}{M}\sum_{m=1}^{M} \tilde{a}_i^{(m)} \quad \text{and} \quad \mathrm{Var}[\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T}]^{\frac{1}{2}} \approx \left[\frac{1}{M-1}\sum_{m=1}^{M} (\tilde{a}_i^{(m)} - \mathbb{E}[\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T}])^2\right]^{\frac{1}{2}}. \tag{4.8}$$

Naturally, $\mathbb{E}[\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T}]$ is a Bayesian point estimate for $\tilde{a}_i$, while $\mathrm{Var}[\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T}]^{\frac{1}{2}}$ offers a measure of uncertainty in such prediction of edge $i$.

## 4.5 Bayesian Modeling of Unrolling-Based BNNs with Independent Interpretability

In this section, we present a method for Bayesian modeling for BNNs which use a network produced by unrolling an optimization algorithm to solve an inverse problem with independent interpretability. We instantiate these ideas on the GSL problem (4.3) - where solutions are undirected graphs - and use prior knowledge over the sparsity of such graphs to shape prior distributions over the corresponding independently interpretable parameter $\theta$ in Unrolled DPG introduced in Section 4.4.1.

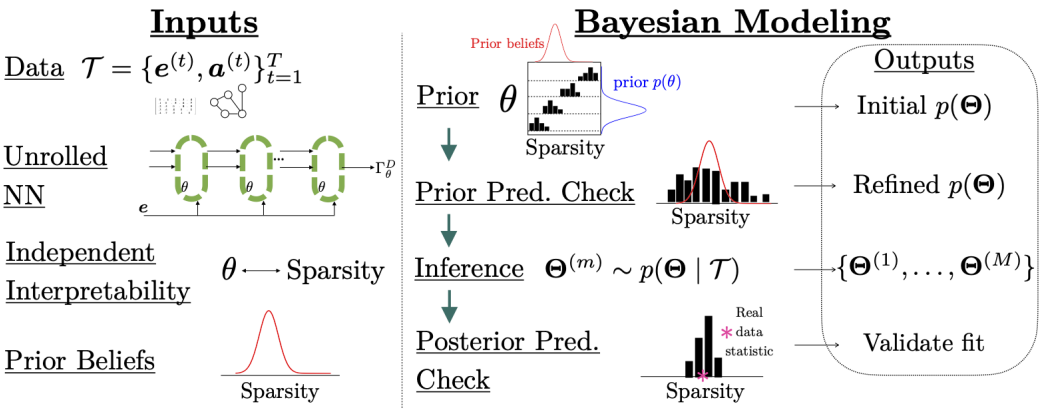


Figure 4.2: Bayesian workflow with independent interpretability. Inputs: A labeled data set $\mathcal{T}$, an inverse problem with independently interpretable parameter $\theta$ w.r.t. some characteristic of the solution, prior beliefs over this solution characteristic, and an unrolled NN which approximate solutions to the inverse problem. Bayesian Modeling: We use independent interpretability of $\theta$ to convert prior beliefs on solution characteristics to a prior distribution on the independently interpretable parameter. We use prior predictive checks to ensure priors generate data sets which encompass all plausible values of the solution characteristic, while still preferentially generating data sets which we believe are more likely apriori. If not, we can leverage independent interpretability to refine the prior. We then sample from the posterior, and use posterior predictive checks to provide a subjective validation of model fit.

### 4.5.1 Prior modeling

For Bayesian models, priors over unobserved quantities play two main roles: encoding information germane to the problem being analyzed and aiding the ensuing Bayesian inferences. Despite the name, a prior can in general only be interpreted in the context of the likelihood with which it will be paired. There are many types of priors, and we here we list the common ones in order of degree in which it is intended to affect the information in the likelihood: non-informative, reference, structural, regularizing, weakly informative, and strongly informative [119]. The more domain specific information present, the further to the end of this list we may be able to move, which often results in better behaved posterior geometries, and thus more stable Bayesian inference. Exactly how this domain specific information is encoded in the prior depends on both the problem and the specific chosen model.

**Informative priors with independent interpretability**. In the context of the GSL problem dealt with here, independent interpretability makes setting a strongly informative prior over DPG's independently interpretable parameter $\theta$ feasible. To this end, one first discretizes $\theta$ over a few orders of magnitude. For each $\theta$ value in the grid, we run Algorithm 1 on a subset of observed inputs $\mathcal{T}_{\boldsymbol{e}}$ and record the relevant data characteristic (edge sparsity) of each recovered solution. We then plot the histogram of such data characteristic and observe their agreement with prior beliefs. Finally, one chooses an appropriate parametric distribution $p(\theta)$ which approximates this relative level of agreement across the range of $\theta$ values. We can use the recovered

solutions at a priori probabale $\theta$ values to set weakly informative priors over the remaining model parameters. In our setting, that involves the scale of the edge weights of the recovered solutions. In Section 4.5.2 we discuss evaluation of this initial $p(\boldsymbol{\Theta})$ in terms of the predictions it makes. For a visualization of this prior modeling workflow, see Figure 4.2. A numerical example is now presented to ground this methodology.

**Example (Informative prior modeling of DPG parameters)**. In this example that continues in Section 4.5.2, we use a data set $\mathcal{T}$ of $T = 50$ random geometric graphs ($N = 20$ nodes, connectivity of $\frac{1}{3}$), denoted $\text{RG}_{\frac{1}{3}}$, and their corresponding analytic Euclidean distance matrices as inputs defined in Section 4.6. Prior modeling and upcoming prior predictive checks require only the distance matrices $\boldsymbol{e}$, and we use only 5 such distance matrices from $\mathcal{T}$.

Suppose we have prior beliefs on the sparsity related characteristics of recovered graphs, namely sparsity itself (= 1 − edge density) or the number of connected components. For example, suppose we believe recovered graphs should have edge densities around $[.05, .5]$ and $\leq 5$ connected components. Since Unrolled DPG approximates solutions to (4.3) - and $\theta$ determines such sparsity related characteristics independently of other optimization parameters - we can simply run Algorithm 1 to convergence using the 5 inputs on a set of discretized $\theta$ values across several orders of magnitude, as demonstrated in Figure 4.3 (top-left). We observe $\theta \in [10^{-1}, 10^{1}]$ produces solutions with sparsity characteristics consistent with our prior beliefs. Choosing $\theta \sim$

Lognormal$(0, 4)$ concentrates approximately 75% probability mass on interval $[10^{-1}, 10^{1}]$ while still covering a broader range of values to accommodate uncertainty stemming from sampling variance (small data set) and approximation error (truncated iterations). Priors are placed over the non-independently interpretable parameters $\delta$ and $b$ with the purpose of guarding against implausible values and stabilizing posterior sampling; such priors are sometimes called 'weakly informative' [118]. Because $\delta\Gamma_{\theta}^{D}(\boldsymbol{e}) - b$ will be driven through a sigmoid, we aim for both terms to be of comparable magnitude. Since our graphs are binary, $\delta\Gamma_{\theta}^{D}(\boldsymbol{e})$ can reasonably be assumed to be $\approx 1$. Utilizing Algorithm 1 solutions obtained over the $\theta$ grid, we examine the edge weight distribution (prior to scaling by $\delta$) in Figure 4.3 (bottom-left) to inform $p(\delta)$. Setting $\delta \sim$ Lognormal(2, 2) brings scaled edge weights close to 1 for high (prior) probability values of $\theta$. Similarly, setting $b \sim$ Lognormal(1, 2) ensures $b$ is also approximately 1. Both distributions assign probability mass across several orders of magnitude, both above and below 1, reflecting the breadth of our uncertainty concerning their specific value ranges. Next, we employ prior predictive checks to validate that - with the established priors - our model generates graphs that align with our prior beliefs.

### 4.5.2 Predictive checking

In Bayesian statistics, predictive checks evaluate a model's efficacy in capturing data characteristics like means, standard deviations, and quantiles [118]. Predictive checks leverage the existence of a model of the joint distribution

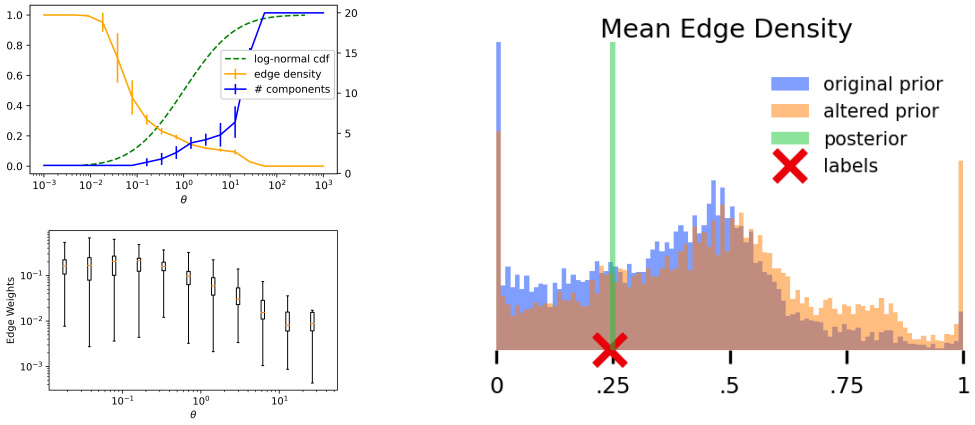


Figure 4.3: *Left:* Prior modeling with independently interpretable parameter $\theta$. We run Algorithm 1 to convergence over discretized $\theta$. *Top*: Larger $\theta$ produces sparser graphs. *Bottom*: Larger $\theta$ produces smaller edge weight magnitudes. *Right:* Predictive Checks. The original prior generates very few data sets with densities of $\approx .9$, a value we feel is plausible. We can use the independent interpretability of $\theta$ w.r.t. edge density to alter its prior accordingly. A prior predictive check with this altered prior now encompasses these plausible data sets. The posterior predictive check ensures the replicated data sets - now sampled after conditioning on the training data - have similar edge densities to the observed training labels. Indeed, these edge densities, denoted as 'posterior', are tightly distributed around the average edge density of the labels.

$p(\mathcal{T}, \boldsymbol{\Theta})$ between data and parameters. This model allows us to draw samples from the data marginal - called *replicated* data - by drawing samples from the joint and simply dropping the parameter samples. Prior predictive checks draw replicated data from the joint before conditioning on data $\mathcal{T}$, while posterior predictive checks do so after. In both cases, we apply the chosen statistic capturing our data characteristic of interest to the replicated data, producing a histogram. The goal in prior predictive checking is to shape

priors that encompass all *plausible* data sets, while still guiding the model towards data sets we deem more *likely* apriori. Thus prior predictive checking often consists of visual inspection of the histogram to ensure this holds, and adjusting the location and/or scale of prior distributions until it does. The goal in posterior predictive checking is to subjectively validate model fit; here we compare the statistic on the replicated data (a histogram) to the same statistic applied to the actual data $\mathcal{T}$ (a single scalar). A well-fit model should have a histogram tightly concentrated around the real data statistic.

In typical BNNs, the lack of parameter interpretability prevents effective use of predictive checking. [95] highlights this issue by demonstrating that an isotropic Gaussian prior on NN weights (ResNet-20) fails to generate data consistent with prior expectations in image classification tasks. But the lack of parameter interpretability prevents the use of these findings to modify the prior for improved alignment. Below, we instantiate predictive checking for DPG, with average edge density as our test statistic.

**Example (Prior predictive check over DPG parameters)**. To perform a prior predictive check we use the 5 inputs from $\mathcal{T}_{\boldsymbol{e}}$ and draw $10^4$ replicated data sets $\boldsymbol{a}^{\text{rep}}$. Figure 4.3 (right) shows the histogram of average edge densities of replicated data sets under the prior settings laid out in Section 4.5.1, which we call the 'original prior'. We observe that such a prior succeeds in producing data sets that coincide with those we deem more likely apriori, but falls short of encompassing all plausible data sets, namely those with edge densities of $\approx 0.9$. To address this, we lean on independent

interpretability of $\theta$ w.r.t. the sparsity of graph solutions: decreasing the location of $p(\theta)$ will increase edge densities (lower sparsity) of graph solutions, and so we lower $p(\theta)$'s location parameter from 0 to $-\frac{1}{2}$, with all other parameters of the prior distribution remaining the same. We call this new prior the 'altered prior'. Repeating the prior predictive check with this 'altered prior' we now observe $\approx 12\%$ of replicated data sets with edge densities in $[.75, 1]$, as opposed to 4% before, and we confirm through visual inspection that data sets with all possible edge densities are indeed produced. The 'altered prior' thus encompasses all plausible data sets while still preferentially generating data sets we feel are more likely apriori.

**Example (Posterior predictive check over DPG parameters)**. Now, we repeat this procedure, but draw replicated data sets from the joint after conditioning on $\mathcal{T}$. We perform inference drawing $M = 10^4$ posterior samples; for each posterior sample we draw a single replicated data set. In Figure 4.3 (right) we plot the histogram of the average edge densities of these replicated data sets, denoted 'posterior', and compare against the average edge density of the graph labels in $\mathcal{T}$, denoted 'labels'. Because all outcomes are tightly distributed around the mean edge density of the real data, we can have confidence the model parameters have fit appropriately.

## 4.6 Experiments

We have introduced DPG, the first BNN for GSL from smooth signals, which is capable of providing estimates of uncertainty of its edge predictions. We showed DPG can effectively incorporate prior information into the prior distribution over its parameters. In this section, we evaluate DPG across synthetic and real datasets, and introduce other baseline models for comparison, including a more expressive variant of DPG.

### 4.6.1 Models, metrics, and experimental details

**Strict unrollings: DPG and PDS.** In the following experiments, the prior used for the DPG's parameters $\{\theta, \delta, b\}$ is the 'altered prior' as developed in Section 4.5: $\theta \sim \text{Lognormal}(-1/2, 4)$, $\delta \sim \text{Lognormal}(2,\ 2)$, and $b \sim \text{Lognormal}(1,\ 2)$. We similarly refer to the BNN with Unrolled PDS as its NN model - and no change to the stochastic model - as PDS. As none of PDS's parameters $\{\alpha, \beta, \gamma, b\}$ have independent interpretability, we cannot use the prior modeling techniques outlined in Section 4.5. Attempting HMC on PDS produces significant number of divergent simulated Hamiltonian trajectories, a phenomenon known to be caused by high posterior curvature [120]. This makes HMC run slowly, produce poorly mixed chains, and ultimately non-performant PDS models. Indeed, we find that values of step-size $\gamma$ which produce divergent behavior (very low likelihood) are close those which are most performant (very high likelihood), indicating high-curvature in the likelihood

function, and thus the posterior. Because $\gamma$ is not interpretable, it is unclear how to shape $p(\gamma)$ to reduce posterior curvature without first running a discrete search. Indeed, to find a performant PDS model amenable to efficient HMC, we first run such a discrete search, fix $\gamma$ to a found performant value of 0.1, and then set priors $\alpha$, $\beta \sim \text{LogNormal}(0, 10)$ and $b \sim \mathcal{N}(0, 10^3)$. Both model-based algorithms in [86] and FDPG [85] come with faster convergence rate guarantees than DPG in Algorithm 1 - a characteristic found not to be critical in the unrolled setting [36]. They involve complex operations and have not been unrolled in prior work, unlike PDS. Specifically, [86] use 6 parameters, including 2 step-sizes, complicating stable unrolling and Bayesian inference, and so we exclude these methods from our comparisons.

**Model expansion via MIMO and partial stochasticity: DPG-MIMO and DPG-MIMO-E.** Here, we expand the Unrolled DPG NN for more expressive power by making each layer multi-input multi-output (MIMO). Each MIMO layer maps inputs $\{\boldsymbol{\lambda}_{k-1} \in \mathbb{R}^{N\times C},\ \mathbf{a}_{k-1} \in \mathbb{R}^{N(N-1)/2\times C}\}$ to outputs $\{\boldsymbol{\lambda}_k \in \mathbb{R}^{N\times C},\ \mathbf{a}_k \in \mathbb{R}^{N(N-1)/2\times C}\}$, where $C$ is the fixed number of input and output channels across layers. As before, layers share parameters, now $\boldsymbol{\theta} \in \mathbb{R}^{C\times C}$. In the $k$-th layer, parameter $\theta_{ji} \in \mathbb{R}$ is used to process $\boldsymbol{\lambda}_{k-1}[:, i]$ and $\mathbf{a}_{k-1}[:, i]$ as in a regular DPG iteration. Doing so for $i \in \{1, \ldots, C\}$ and averaging the refined $\boldsymbol{\lambda}$'s and $\mathbf{a}$'s produces $\boldsymbol{\lambda}_k[:, j]$ and $\mathbf{a}_k[:, j]$, respectively. We use parameters $\boldsymbol{\delta} \in \mathbb{R}^C$ and $b \in \mathbb{R}$ to produce edge probabilities $\sigma(\frac{1}{C}\boldsymbol{a}_D\boldsymbol{\delta} - b\mathbf{1})$. The priors are unchanged from the 'altered prior' developed in the non-MIMO setting from Section 4.5. The increased parameterization and complexity

of operations make the posterior geometry significantly more complex, such that we could not find a setup which produced well-mixed posterior sampling via HMC. Instead, we resort to Maximum a Posteriori (MAP) estimation, equivalent to maximizing the posterior with fixed observed data, using full gradient descent. We call this non-stochastic setup DPG-MIMO.

A common approach to overcome the intractable posterior inference in BNNs is partial stochasticity, where we learn point estimates of a subset of the parameters and distributions over the rest. Recent work has shown partial stochasticity of a BNN can produce similarly useful posterior predictive distributions, even outperforming fully stochastic networks in prediction in some setting [121]. Here, we 'decapitate' the MAP trained DPG-MIMO by discarding $\boldsymbol{\delta}_{\text{MAP}}$ and $b_{\text{MAP}}$; instead we feed each of the $C$ output channels $\boldsymbol{a}_D[:, j]$ of the base MAP DPG-MIMO (only parameterized by $\boldsymbol{\theta}_{\text{MAP}}$) into a new depth 20 stochastic single channel DPG $\delta_j \Gamma^{20}_{\theta_j}(\boldsymbol{a}_D[:, j])$. We average their output, shift by new stochastic $b$, and drive through a sigmoid: $\sigma\big(\frac{1}{C}\sum_{j=1}^{C} \delta_j \Gamma^{20}_{\theta_j}(\boldsymbol{a}_D[:, j])) - b\mathbf{1}\big)$. We can now run inference on these stochastic head parameters $\{\theta_1, \delta_1, \ldots, \theta_C, \delta_C, b\}$ keeping $\boldsymbol{\theta}_{\text{MAP}}$ fixed. We denote this model as DPG-MIMO-E. All MIMO models use $C = 4$: $C = 8$ offered negligible performance gains and posed (partially stochastic) inference challenges, while $C = 2$ was less performant than $C = 4$.

**Metrics.** To provide summaries of our models' predictive accuracy and quality of uncertainty we use two proper scoring rules: the Negative Log-Likelihood (NLL) $p(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \mathcal{T})$ and the Brier Score $\frac{1}{|\mathcal{E}|}\mathbb{E}_{\boldsymbol{\Theta}|\mathcal{T}}[\|\hat{\boldsymbol{p}}(\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \boldsymbol{\Theta}) - \tilde{\boldsymbol{a}}]\|^2)$.

Beyond proper scoring rules, we use Expected Calibration Error (ECE) [122], which measures the correspondence between predicted probabilities and empirical accuracy, and Error, defined as the percentage disagreement between a thresholded pred. mean $\mathbb{E}_{\boldsymbol{\Theta}|\mathcal{T}}[\tilde{\boldsymbol{a}} \mid \tilde{\boldsymbol{e}}, \mathcal{T}] > 0.5$ and the actual label $\tilde{\boldsymbol{a}}$.

**Hyperparameters and inference details.** Utilizing NumPyro's NUTS implementation of HMC [123], our experiments run 4 chains in parallel, each chain taking 500 warm-up steps before generating 1000 samples, accumulating $M = 4000$ total samples. We use depth $D = 200$ for all models unless otherwise specified, as we did not observe significant improvements in predictive performance beyond this depth. We choose $\boldsymbol{a}_0 = \frac{1}{2} \cdot \mathbf{1}$ (reflecting prior uncertainty of existent edges) and $\boldsymbol{\lambda}_0 = 17 \cdot \mathbf{1}$ (approximate average value of limiting $\boldsymbol{\lambda}$ when running Algorithm 1 to convergence on $\mathrm{RG}_{\frac{1}{3}}$ analytical distance matrices for $\theta = 1$) for all experiments. We did not find performance to be sensitive to such choices.

### 4.6.2 Synthetic data evaluation

**Generative smooth signal model.** We build on [59]'s work on probabilistic smooth graph signal generation to validate our methods using synthetic data. Let $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ be the eigendecomposition of the graph Laplacian $\boldsymbol{L} = \mathrm{diag}(\boldsymbol{A}\mathbf{1}) - \boldsymbol{A}$, with diagonal eigenvalue matrix $\boldsymbol{\Lambda}$ having associated eigenvalues $\lambda_i$ sorted in increasing order, $\boldsymbol{u}_i$ ($i$-th column of $\boldsymbol{U}$) is the eigenvector of $\boldsymbol{L}$ with eigenvalue $\lambda_i$. Further take $\boldsymbol{L}^\dagger$ to be the Moore-Penrose pseudoinverse of $\boldsymbol{L}$, with eigenvalues $\lambda_i^\dagger := \lambda_i^{-1}$ when $\lambda_i > 0$, and $\lambda_i^\dagger := 0$

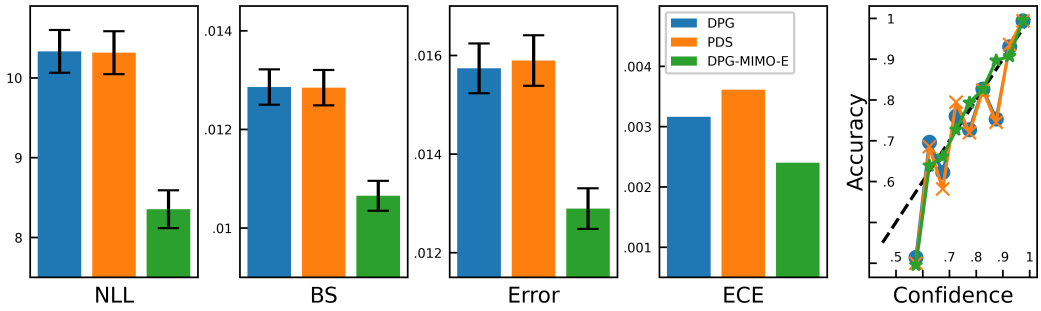


Figure 4.4: Effective i.i.d. generalization. Both DPG and PDS are performant and well calibrated BNNs, although PDS requires $\approx 3\times$ more time for inference. Further performance gains are found with the expanded, partially stochastic DPG-MIMO-E model. The rightmost plot (reliability diagram) indicates high calibration across confidence levels; the left-most bin is least calibrated but contains $< 0.4\%$ of edges across all models. This plot shows experiments on $N = 20$ $\mathrm{RG}_{\frac{1}{3}}$ graphs. Error bars are scaled by .05 for compact visual effect.

otherwise. [59] proposed that smooth signals can be generated from a colored Gaussian distribution as $\boldsymbol{x} = \boldsymbol{\mu} + \sum_i \hat{x}_i \boldsymbol{u}_i$, where $\hat{x}_i \sim \mathcal{N}(0, \lambda_i^\dagger)$ and $\boldsymbol{\mu} \in \mathbb{R}^N$ denotes an arbitrary mean vector. Therefore $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{L}^\dagger)$. To sample such a distribution, it suffices to draw an initial non-smooth signal $\boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and then compute $\boldsymbol{x} \leftarrow \boldsymbol{\mu} + \sqrt{\boldsymbol{L}^\dagger}\boldsymbol{x}_0$. When $\boldsymbol{\mu} = \boldsymbol{0}$, this procedure produces signals such that the power on the $i$-th frequency component is $\hat{x}_i^2 \sim \Gamma(k = \frac{1}{2}, \theta = 2\lambda_i^\dagger)$. Thus the expected power $\mathbb{E}[\hat{x}_i^2] = \lambda_i^\dagger$ is inversely proportional to the associated eigenvalue, concentrating energy on the low frequencies of the Laplacian spectrum. We construct the data matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P] \in \mathbb{R}^{N \times P}$, where $\boldsymbol{x}_p$ are drawn i.i.d. from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{L}^\dagger)$ as described above. Recalling the definition of the Euclidean distance matrix $\boldsymbol{E}$ in Section 4.3 and using the fact that $\bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{x}}_i \sim \Gamma(k = \frac{P}{2}, \theta = 2L_{ii}^\dagger)$ and

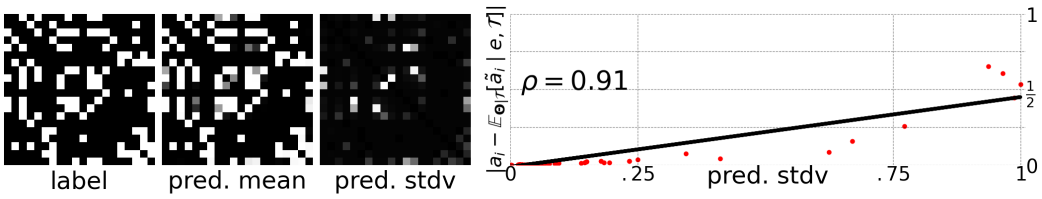


Figure 4.5: Qualitative i.i.d. generalization. *Left:* For a random test sample we show the label $\tilde{\boldsymbol{a}}$, and estimated mean (pred. mean) and standard deviation (pred. stdv) of the edge-wise marginal posterior predictive $p(\tilde{a}_i \mid \tilde{\boldsymbol{e}}, \mathcal{T})$. Comparing pred. mean to the label $\tilde{\boldsymbol{a}}$ adds qualitative evidence that the model is well fit to the data. *Right:* The edge-wise uncertainty estimate (pred. stdv.) and error $|\tilde{a}_i - \mathbb{E}_{\boldsymbol{\Theta}|\mathcal{T}}[\tilde{a}_i|\tilde{\boldsymbol{e}}, \mathcal{T}]|$ have a strong positive Pearson correlation $\rho = 0.91$.

$\mathbb{E}[\bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{x}}_j] = \sum_{p=1}^{P} \mathbb{E}[x_{ip}x_{jp}] = \sum_{p=1}^{P} \mathrm{Cov}(x_{ip}, x_{jp}) = L^\dagger_{ij}, \forall(i,j)$, we can show $\mathbb{E}[\boldsymbol{E}] = \mathbf{1}\mathrm{diag}(\boldsymbol{L}^\dagger)^\top + \mathrm{diag}(\boldsymbol{L}^\dagger)\mathbf{1}^\top - 2\boldsymbol{L}^\dagger$ which we call the 'analytic distance matrix'. We henceforth use $\mathbb{E}[\boldsymbol{E}]$ and its finite sample approximations from $P$ signals for evaluation.

**Evaluation of i.i.d. generalization.** Here, we train and test DPG, PDS, and DPG-MIMO-E on $N = 20$ $\mathrm{RG}_{\frac{1}{3}}$ graphs and their corresponding analytic distance matrices. We use $T = 50$ graphs for training and 100 for testing. We present the results in Figure 4.4. DPG and PDS perform comparably; although DPG has marginally better accuracy and calibration. Significant improvement is found in the more expressive DPG-MIMO-E across all metrics, showing our simple 3-parameter model can be effectively expanded. These models are similarly performant across graph distributions (see the experiments reported in Table 4.1; we show a single graph ensemble here for brevity.

Figure 4.5 depicts estimates of the first two moments of the posterior

predictive distributions produced by DPG on a random test sample. We observe well-calibrated confidence and uncertainty, plus a strong correlation between error magnitude and uncertainty; indeed a useful uncertainty estimate should be a proxy for error.

**Predictive uncertainty under distribution shift.** We evaluate two forms of distribution shift: corruptions to the noiseless analytic distance matrix $\mathbb{E}[\boldsymbol{E}]$ and label mismatch. To investigate corruptions, we train using $T = 50$ $\text{RG}_{\frac{1}{3}}$ ($N = 20$) labels and their analytic distance matrix, and test on 100 $\text{RG}_{\frac{1}{3}}$ samples which instead use $P$ signals to compute an Euclidean distance matrix $\boldsymbol{E}$. Fewer samples $P$ tend to produce larger corruption magnitudes, i.e., deviations from $\mathbb{E}[\boldsymbol{E}]$. Figure 4.6 shows that indeed, all models display lower predictive accuracy and higher uncertainty on the increasingly shifted input data, with the simpler models outperforming in the presence of larger corruption. To investigate label mismatch, models are fit to the same training data as above - $T = 50$ $\text{RG}_{\frac{1}{3}}$ ($N = 20$) labels with analytic distance matrix input - but now tested on 100 test samples from the following *different* $N = 20$ random graph distributions: (i) random geometric graphs with connectivity radius $\frac{1}{2}$ ($\text{RG}_{\frac{1}{2}}$), Erdős-Rényi graphs with edge probability $p = 0.5$ ($\text{ER}_{\frac{1}{2}}$), and Barabási-Albert graphs with $m = 1$ link per node ($\text{BA}_1$). Results are displayed in Table 4.1. All models show lower accuracy (↑ Brier Score) and commensurately higher uncertainty (↑ NLL) on these mismatched data. DPG methods tend to maintain better calibration than PDS.

**Scaling to larger graphs.** Direct inference in large graph settings is

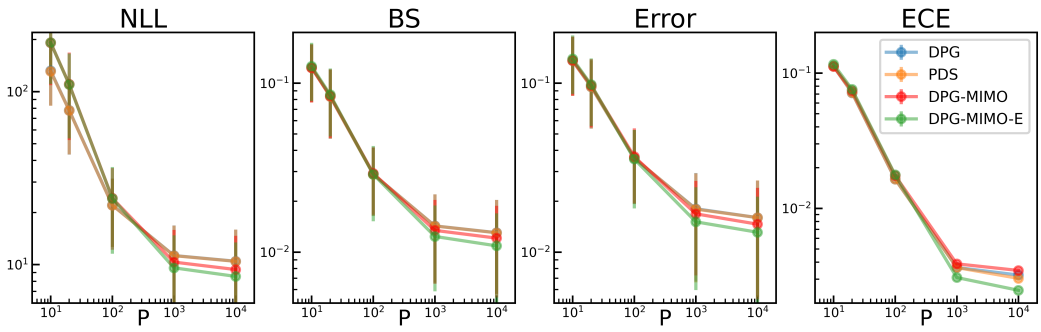


Figure 4.6: Detection of covariate shift across models. Using $\mathrm{RG}_{\frac{1}{3}}$ graphs with $N = 20$ nodes, we fit models on analytic Euclidean distance matrices and evaluate on corrupted distance matrices in a test set. Fewer signals $P$ used tends to increase corruption magnitude. Log-scaling unintentionally makes error bars appear to grow with $P$.

Table 4.1: Detection of label mismatch. Models are fit to $\mathrm{RG}_{\frac{1}{3}}$ and tested on 100 samples from the same $\mathrm{RG}_{\frac{1}{3}}$ as well as three *different* graph distributions. For each metric, the mean over the test set is reported.

| | NLL | | | | BS ($\times 10^{-2}$) | | | | ECE ($\times 10^{-3}$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\mathrm{RG}_{\frac{1}{3}}$ | $\mathrm{RG}_{\frac{1}{2}}$ | $\mathrm{ER}_{\frac{1}{2}}$ | $\mathrm{BA}_1$ | $\mathrm{RG}_{\frac{1}{3}}$ | $\mathrm{RG}_{\frac{1}{2}}$ | $\mathrm{ER}_{\frac{1}{2}}$ | $\mathrm{BA}_1$ | $\mathrm{RG}_{\frac{1}{3}}$ | $\mathrm{RG}_{\frac{1}{2}}$ | $\mathrm{ER}_{\frac{1}{2}}$ | $\mathrm{BA}_1$ |
| DPG | 10.33 | 17.53 | 33.01 | 20.91 | 1.29 | 2.29 | 4.66 | 2.10 | 3.42 | 52.06 | 171.38 | 16.60 |
| PDS | 10.32 | 17.70 | 33.43 | 20.80 | 1.29 | 2.33 | 4.80 | 2.10 | 3.75 | 52.87 | 173.53 | 16.60 |
| DPG-MIMO | 9.19 | 17.13 | 40.49 | 13.37 | 1.19 | 1.74 | 4.62 | 2.06 | 3.50 | 54.61 | 161.71 | 11.24 |
| DPG-MIMO-E | 8.35 | 12.66 | 37.77 | 8.35 | 1.07 | 1.52 | 7.82 | 1.32 | 2.40 | 29.17 | 29.07 | 2.84 |

challenged by substantial memory demands. Specifically, providing gradients for HMC via naive backpropagation necessitates storing all intermediate outputs, while full Bayesian inference mandates holding the full data set in memory, cumulatively leading to a memory footprint of $\mathcal{O}(DN^2T)$. Adopting forward-mode auto-differentiation partially mitigates this issue by removing depth dependency, effectively reducing memory complexity to $\mathcal{O}(N^2T)$. The time complexity for MCMC also presents scalability challenges: each forward pass has time complexity of $\mathcal{O}(DN^2T)$, and each posterior sample requires

multiple forward passes. Attempting to instead perform transfer learning – where inference occurs with smaller graphs to then test on larger graphs – still faces an obstacle. The objective function (4.3) contains the terms $2\boldsymbol{a}^\top \boldsymbol{e} = 2\sum_{i=1}^{N(N-1)/2} a_i e_i$, $\frac{\beta}{2}\|\boldsymbol{a}\|_2^2 = \frac{\beta}{2}\sum_{i=1}^{N(N-1)/2} a_i^2$, and $-\alpha \mathbf{1}^\top \log(\boldsymbol{S}\boldsymbol{a}) = -\alpha \sum_{i=1}^{N} \log(\boldsymbol{S}\boldsymbol{a})_i$. The former two terms involve $N(N-1)/2$ summands, while the latter is a sum over $N$ elements. This results in disparate growth rates for these terms as $N$ increases, which poses a challenge for parameter optimization across different graph sizes, i.e., we find parameters optimized for a graph with $N_i$ nodes are not effective for $N >> N_i$. Moreover, the differing growth rates cause standard cardinality-based normalization schemes (dividing each term by its number summands) to fail, and determining analytically how the parameters should change as a function of size is non-trivial.

In light of this, we perform an adjusted transfer learning experiment by fitting the empirical growth trends of MAP DPG parameter estimates on $N = \{20, 50, 100, 200\}$ $\mathrm{ER}_{\frac{1}{4}}$ graphs and their corresponding analytic distance matrices. By using a moderate depth $D = 200$ and $T = 10$ training samples, MAP inference on larger graphs with $N = 200$ nodes takes $\sim 1$ hour on a M2 MacBook laptop. Using this empirical parameter fit we can extrapolate parameter values and perform transfer learning on 100 $\mathrm{ER}_{\frac{1}{4}}$ test graphs of size up to $N = 1000$; again done locally without any GPU. The transfer learning experiment reveals an expected but graceful decay in performance in NLL as $N$ increases.

### 4.6.3 Ablation studies

**Prior modeling.** To inspect the influence of informative prior modeling on DPG, we define model 'DPG-U' which replaces DPG's informative priors with the following uninformative priors: $\log \theta \sim U\left[10^{-6}, 10^{6}\right]$, $\delta, b \sim \mathcal{N}(\mathbf{0}, 10^{3}\mathbf{I})$. What we find is that in data-rich regimes an informative prior mostly acts to improve efficiency of posterior inference, e.g., when fitting to $T = 50$ $\mathrm{RG}_{\frac{1}{3}}$ graphs with $N = 20$ nodes (using analytic distance matrices) and testing on 100 i.i.d. samples, it reduces the time needed to generate $M = 4000$ samples by a factor of 7.8× and increases the effective sample size - a measure of the number of independent samples with the same estimation power as the observed correlated samples [118] - for $\theta$ ($\mathrm{ESS}_{\theta}$) by a factor of 5×. In data poor regimes it also tends to help slow down performance degradation, e.g., when instead using $T = 2$ with the same graph data, NLL and ECE are 37% and 13.7% better with the prior than without.

**Partial stochasticity.** We conducted an ablation study investigating the benefits of adding partial stochasticity to the DPG-MIMO model ($C = 4, D = 200$) using MAP point estimates; the partially stochastic setup is described in Section 4.6.1. The training and test sets are identical to the i.i.d. generalization experiment above. Our experiments showed that introducing partial stochasticity in DPG-MIMO-E markedly improved NLL, BS, error, and ECE by 9.1%, 10.3%, 9.0%, and 31.4%, respectively.

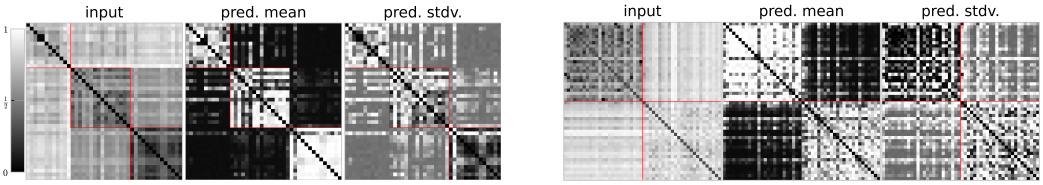


Figure 4.7: Quantifying Uncertainty with Financial Data and Images of Digits. *Left*: S&P500. We randomly split stocks from 3 sectors of the S&P500 and compute their input $\mathbf{1}\mathbf{1}^\top - |\mathbf{\Sigma}|$, where $\mathbf{\Sigma}$ is the matrix of Pearson correlation coefficients, using log daily returns over an extended period. The graph label connects stocks of the same sector indicated by red block diagonal outline. There is only a single train and test sample. Here, we display the test sample input and estimates of the mean (pred. mean) and standard deviation (pred. stdv.) of the posterior predictive. In pred. mean, white (black) inside (outside) the block diagonal indicates correct prediction for both experiments. Pred. mean which don't match their label tend to have larger uncertainty. Error and pred. stdv. have a Pearson correlation of 0.70 over the test set. *Right*: MNIST digits. We construct graphs of MNIST digits "1" and "2", connecting nodes of the same digit, similarly outlined in red ("1"s are the first block). The input is the log pairwise Euclidean distance of their vectorized image. Notice "1"s tend to be much closer to each other than "2"s. DPG again shows an ability to place higher uncertainty on edges with higher error. Error and pred. stdv. have a Pearson correlation of 0.62 over the test set. Pred. stdv. in both plots are maximum normalized for visual clarity.

### 4.6.4 Real data evaluation

**Quantifying uncertainty in networks learnt from S&P500 stock prices.** To verify that DPG provides useful measures of uncertainty on edge predictions based on real data, we first use the financial dataset presented in [124]. The time series consist of S&P500 daily stock prices from three sectors (Communication Services, Utilities, and Real Estate), comprising a total of $N = 82$ stocks. The data spans the period from Jan. 3rd 2014 to Dec. 29th 2017, yielding $P = 1006$ daily observations. We divided these stocks equally

into training and testing sets. For both sets, we created a log-returns data matrix, denoted as $\boldsymbol{X} \in \mathbb{R}^{N/2 \times P}$. Specifically, $X_{i,j} = \log SP_{i,j} - \log SP_{i,j-1}$, where $SP_{i,j}$ signifies the closing price of the $i$-th stock on the $j$-th day. From each matrix (train and test), we derived a matrix of Pearson correlation coefficients $\boldsymbol{\Sigma}$. We take the input to be $\boldsymbol{E} \leftarrow \mathbf{1}\mathbf{1}^\top - |\boldsymbol{\Sigma}|$, thus yielding measure of dissimilarity. We also constructed a binary label graph, assigning an edge weight value of 1 for stocks within the same sector and 0 for pairs in different sectors. Thus, our training and testing datasets each contain a single sample. Nodes from the same sector are numbered contiguously creating a block diagonal adjacency matrix structure, displayed with the red outline in Figure 4.7 (left).

We use DPG with depth $D = 200$ and identify a high-performing parameter value range for $\theta$ via predictive checking as in Section 4.5.2. Such values aligned closely with those found in the synthetic experiments, and so we keep the priors unchanged. We run Bayesian parameter inference on a M2 MacBook laptop which takes about 2 minutes; we visualize the input, empirical mean, and standard deviation of the posterior predictive on the test sample in Figure 4.7 (left). Notably, while our mean recovery is robust, there are areas where it deviates from the label. These areas tend to exhibit increased variation in the posterior predictive. The Pearson correlation coefficient between the error and predictive standard deviation on all edges, true positive edges, and true negative edges is 0.70, 0.70, and 0.79, respectively, suggests that our model's uncertainty can be a useful proxy for error in the absence of

labels in this financial test case.

**Learning the graph of MNIST digits 1 and 2.** To further demonstrate DPG's capability to quantify uncertainty on graphs estimated from images of digits, we emulate the experiment from [19] which learns a graph to classify handwritten digits "1" and "2" from the MNIST database [125]. Each digit is an image of $28 \times 28$ pixels, each pixel taking integer value in $0, \ldots, 255$. As [19] reports, this problem is particular because digits "1" are much close to each other than "2" are (average square distance of 45 and 102, respectively). A single sample in our dataset is constructed as follows. We randomly sample 25 images of digits "1" and "2". Each image represents a node in this graph sample, hence $N = 50$. We connect nodes of the same digit with a binary edge. The image corresponding to each node is vectorized into $\bar{\boldsymbol{x}}_i \in \mathbb{R}^{28^2}$, $i = 1, \ldots N$, becoming the nodal feature vector. For ease of label visualization, we order the nodes such that all "1" digits come before "2" digits, which creates a block diagonal adjacency matrix, indicated by the red outline in Figure 4.7 (right). The nodal dissimilarity matrix $\boldsymbol{E}$ is computed as the log pairwise Euclidean distance of the node features (vectorized digit images). We construct a training and test set of size $T = 5$ and 50, respectively.

We use the same modeling and inference setup as in the previous real data experiment; inference takes $\approx 10$ minutes. Figure 4.7 (right) displays the input $\boldsymbol{E}$, pred. mean, and pred. stdv. on a random test sample. Visual inspection reveals performant mean prediction, and that edges with high errors tend to have high variation. Indeed, across the entire test set the

two have a strong positive Pearson correlation coefficient over all edges, true positive edges, and true negative edges of 0.62, 0.72, and 0.51, respectively, suggesting DPG effectively quantifies predictive uncertainty of edges in this image analysis setting.

# Chapter 5

# Stabilizing the Kumaraswamy Distribution

## 5.1 Introduction

Probabilistic models use probability distributions as building blocks to model complex joint distributions between random variables. Such distributions can model unobserved 'latent' variables $\boldsymbol{z}$, or observed 'data' variables $\boldsymbol{x}$. Bounded interval-supported latent variables are central to many key applications, such as unobserved probabilities (e.g., user clicks in recommendation systems or links between network nodes), missing measurements in control systems (e.g., joint angles in $[0, 2\pi]$), and stochastic policies over bounded actions in reinforcement learning (e.g., motor torque in $[-10, 10]$).

To meet the demands of large-scale latent variable models, distributions supported on bounded intervals must satisfy the following criteria: (i) support the reparameterization trick through an explicit reparameterization function, such as a closed-form inverse CDF, enabling low-variance gradient estimation

and efficient sampling; (ii) provide sufficient expressiveness to capture complex latent spaces; and (iii) offer simple distribution-related functions (log-pdf, explicit reparameterization function, and gradients) that allow fast and accurate evaluation. In Section 5.2, we argue that the Kumaraswamy (KS) distribution uniquely meets these criteria, yet remains surprisingly underused. In Section 5.3, we demonstrate that the KS distribution-related functions exhibit numerical instabilities concealed by standard parameterizations and exacerbated in large-scale latent variable models.

In this paper, we make the following technical contributions:

- We introduce an unconstrained logarithmic parameterization of the KS's log-pdf, CDF, inverse CDF, and gradients, which isolate the dominant numerical instabilities, allowing application of recently developed stabilization techniques (Section 5.3).

- We propose the Variational Bandit Encoder (VBE), addressing exploration-exploitation trade-offs in contextual Bernoulli multi-armed bandits (Section 5.4.2).

- We propose the Variational Edge Encoder (VEE) for improved uncertainty quantification in link prediction with graph neural networks (Section 5.4.3).

The VBE and VEE are scalable latent variable models with bounded interval-supported latent variables. Unlike traditional methods which tend to model global latent variables (Section 5.5), such as the parameters of a shared neural

network (NN), the VBE and VEE define local latent variables per bandit arm or network link. This allows the models to incorporate prior knowledge precisely where domain expertise tends to reside—at a granular level, such as the expected reward of a specific arm or the probability of a particular link. Our numerical experiments demonstrate that both models perform best when paired with the stabilized KS distribution in their variational posterior, reinforcing its role as a core component in large-scale bounded latent variable modeling.

## 5.2 Background

The KS distribution [126,127] has pdf $f(x) = abx^{a-1}(1-x^a)^{b-1}$, CDF $F(x) = 1-(1-x^a)^b$, and inverse CDF $F^{-1}(u) = (1-u^{b^{-1}})^{a^{-1}}$, all defined for $x, u \in (0,1)$ and parameterized by $a, b > 0$. The differential entropy of a KS with parameters $a, b$ is

$$\begin{aligned}\mathcal{H}(\text{KS}) :=& -\int_0^1 f(x)\log f(x)dx \\ &= 1-b+(1-a)\left(\phi^{(0)}\left(b^{-1}+1\right)+\gamma\right)-\log a-\log b,\end{aligned}$$

where $\phi^{(0)}$ is the digamma function and $\gamma \approx 0.577$ is the Euler-Mascheroni constant. The digamma function and its gradient, the trigamma function $\phi^{(1)}(x)$, can be represented as infinite series which converge rapidly and thus can be used effectively in numerical applications. They are included as

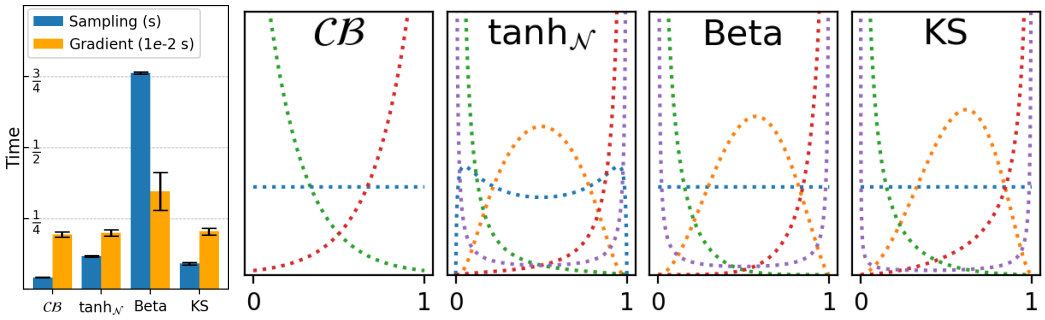


Figure 5.1: Comparison of relevant bounded interval-supported distributions. *Left:* Time for sampling and differentiating through samples. The Beta lacks explicit reparameterization, and has slower sampling and gradients. *Right:* Expressiveness in terms of attainable prototypical shapes.

standard functions in common auto-differentiation frameworks.

**Continuous distributions with bounded interval support.** Among distributions with bounded interval support, the KS uniquely satisfies desiderata (i)–(iii) in Section 5.1. It supports the reparameterization trick through its closed-form, differentiable inverse CDF, providing efficient sampling and low-variance gradients. The KS supports four distinct prototypical shapes — bell, U, increasing, and decreasing (Figure 5.1, right) — providing expressivity for diverse modeling tasks. Its log-pdf, CDF, and inverse CDF, along with their gradients, are composed only of affine transformations, exponentials, and logarithms, and can be parameterized directly in terms of unconstrained logarithmic values; see Section 5.3. This enables straightforward implementation with minimal dependencies and keeps most computation in log-space, enhancing stability and accuracy. The unconstrained logarithmic parameterization makes it well-suited for NNs, eliminating the need for positivity-enforcing link

| **Property / Distribs.** | $\mathcal{CB}$ | $\tanh_{\mathcal{N}}$ | Beta | KS |
|---|---|---|---|---|
| Expressiveness | low | high | high | high |
| Gradient Reparam. | explicit | explicit | implicit | explicit |
| Contains Uniform | ✓ | ✗ | ✓ | ✓ |
| Closed-form CDF | ✓ | ✗ | ✗ | ✓ |
| Closed-form iCDF | ✓ | ✗ | ✗ | ✓ |
| Numerical Issues | mild | high | low | low |
| Complex Functions | $\tanh^{-1}$ | $\log(1\text{-}\tanh^2(x))$ | $\beta$, $I$ | None |
| Parameterization | $\mathbb{R}$ | $\mathbb{R}^2$ | $\mathbb{R}^2_+$ | $\mathbb{R}^2$ |
| Analytical Moments | ✓ | ✗ | ✓ | ✓ |
| Closed-form KL | Exp. Fam | $\tanh_{\mathcal{N}}$ | Exp. Fam | Beta |
| Entropy $\mathcal{H}$ | ✓ | ✗ | ✓ | ✓ |

Table 5.1: Comparison of bounded interval-supported distribution families.

functions. Additionally, the KS has differentiable, closed-form expressions for moments, median, differential entropy $\mathcal{H}(\text{KS})$, and the Kullback-Leibler (KL) divergence to the Beta distribution, facilitating efficient incorporation of prior information.

We briefly introduce workhorse bounded-interval supported distribution families, namely the the Continuous Bernoulli, the Beta, and the tanh-squashed-Gaussian. The Continuous Bernoulli ($\mathcal{CB}$) [128] arises in deep learning for modeling continuous $[0, 1]$-valued pixel intensities in natural images. It provides a normalized probabilistic counterpart to the commonly used binary cross-entropy loss, with density $p(x; \lambda) = C(\lambda)\lambda^x(1-\lambda)^{1-x}, x \in [0, 1]$, $\lambda \in (0, 1)$, where $C(\lambda) = \{2 \text{ if } \lambda = \frac{1}{2}, \text{ else } \frac{2\tanh^{-1}(1-2\lambda)}{1-2\lambda}\}$ is the normalizing constant. The Beta distribution is a flexible two-parameter family, widely used for modeling probabilities and proportions. Its density, parame-

terized by $a, b > 0$, is given by: $p(x; a, b) = B(a, b)^{-1} x^{a-1}(1-x)^{b-1}, x \in (0, 1)$, where $B(a, b)$ is the Beta function. The tanh-squashed-Gaussian ($\tanh_{\mathcal{N}}$) maps Gaussian samples through the tanh function to produce outputs in $[-1, 1]$: $y = \tanh(z), z \sim \mathcal{N}(\mu, \sigma^2)$. It is widely used in reinforcement learning over continuous bounded action spaces [129] due to its support for the reparameterization trick.

Table 5.1 compares these bounded-interval supported distribution families across important properties for latent variable modeling. *Expressiveness* measures the variety of prototypical shapes a distribution can represent. All distributions except $\mathcal{CB}$ exhibit four prototypical shapes; $\mathcal{CB}$ is limited to two. *Contains uniform* refers to the ability to represent the uniform distribution, critical for modeling complete uncertainty. All distributions except $\tanh_{\mathcal{N}}$ can express the uniform. *Closed-form CDF* indicates whether a closed-form CDF is available. Only $\mathcal{CB}$ and KS provide such expressions. Similarly, *closed-form inverse CDF* indicates the availability of a closed-form inverse CDF, with only $\mathcal{CB}$ and KS satisfying this criterion. *Numerical issues* capture challenges in stable evaluation. For example, $\mathcal{CB}$ requires a Taylor expansion to handle singularities as $\lambda \to 0.5$. The $\tanh_{\mathcal{N}}$ distribution requires log-pdf clipping and parameter regularization to maintain stability, as appears in various implementations [129]. *Complex functions* highlight reliance on non-affine, non-logarithmic, or non-exponential operations. The $\tanh_{\mathcal{N}}$ involves computing $\log\left(1 - \tanh^2(x)\right)$, which is numerically unstable [130]. The Beta distribution relies on the Beta function and the regularized incomplete

Beta function in its log-pdf and CDF, respectively, both requiring numerical approximations. In contrast, KS avoids such complexity in our novel parameterization (Section 5.3) , computing $a^{-1}$ as $\exp(-\log a)$ to sidestep division. In contrast, our novel parameterization of the KS distribution avoids complex functions; note $a^{-1}$ is computed via $\exp(-\log a)$, avoiding division. *Parameterization* examines whether a distribution can be effectively expressed with unconstrained parameters. Both $\mathcal{CB}$ (via $\log \lambda \in \mathbb{R}$) and $\tanh_{\mathcal{N}}$ (via $(\mu, \log \sigma) \in \mathbb{R}^2$) support unconstrained parameterization. We introduce the first unconstrained parameterization for KS in Section 5.3, using $(\log a, \log b) \in \mathbb{R}^2$. The Beta distribution, due to its dependence on the Beta function, resists effective unconstrained parameterization. *Closed-form KL functions* refer to analytical KL divergence expressions. The $\mathcal{CB}$ and Beta distributions, as members of the exponential family, admit closed-form KL expressions with other exponential family members. The KS also has closed-form KL expressions with Beta family members, while $\tanh_{\mathcal{N}}$ is restricted to closed-form KL expressions within its own family. *Entropy* considers the availability of closed-form expressions for differential entropy. This property is present for all distributions except $\tanh_{\mathcal{N}}$.

**Latent variable modeling with stochastic variational inference (SVI).** The primary method for fitting large-scale latent variable models is SVI [37]. Consider a model $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$, where $\boldsymbol{x} \in \mathbb{R}^M$ is the observation, $\boldsymbol{z} \in \mathbb{R}^D$ is a vector-valued latent variable, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is the likelihood function with parameters $\boldsymbol{\theta}$, and $p(\boldsymbol{z})$ is the prior distribution. Except for a few special

cases, maximum likelihood learning in such models is intractable because of the difficulty of the integrals involved. Variational inference [38] provides a tractable alternative by introducing a variational posterior distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ and maximizing a lower bound on the marginal log-likelihood called the ELBO:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\mathrm{KL}} \left(q_{\boldsymbol{\phi}}(\boldsymbol{z}) \,\|\, p(\boldsymbol{z})\right) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}). \quad (5.1)$$

Training models with modern SVI [39,40] involves gradient-based optimization of this bound w.r.t. both the model parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$. The first term in (5.1) encourages the model to assign high likelihood to the data, but its exact evaluation and gradients are typically intractable and so the expectation is often approximated with samples from $q_{\boldsymbol{\phi}}(\boldsymbol{z})$. The KL divergence term incorporates prior information by penalizing deviations of the variational posterior from the prior $p(\boldsymbol{z})$. Closed-form expressions of $D_{\mathrm{KL}} \left(q_{\boldsymbol{\phi}} \left(\boldsymbol{z}\right) \,\|\, p \left(\boldsymbol{z}\right)\right)$ allow efficient encoding of prior information; otherwise, sample-based approximations are required. In the common setting of i.i.d. data with per-datapoint latent variables, amortized inference introduces a shared NN, parameterized by 'inference parameters' $\boldsymbol{\phi}$, to map observations to variational parameters, approximating their individual posteriors as $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Modifying the ELBO by scaling the KL term with a parameter $\beta_{\mathrm{KL}} > 0$ is often necessary to balance the trade-off between data likelihood and prior regularization [41]. We denote the sample-based approximation of this modified

ELBO as $\hat{\mathcal{L}}_{\beta_{\mathrm{KL}}}$.

**Gradient reparameterization: explicit and implicit.** A distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ is said to be *explicitly* reparameterizable, or amenable to the 'reparameterization trick', if it can be expressed as a deterministic, differentiable transformation $\boldsymbol{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi})$ of a base distribution $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. This base distribution is typically simple, such as Uniform or standard Normal, enabling fast sample generation by first sampling from the base and then applying $g$. This enables the use of backpropagation to estimate gradients of the form [cf. (5.1)]

$$\begin{aligned} \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z})] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} f(g(\boldsymbol{\epsilon}, \boldsymbol{\phi}))] \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{z}} f(\boldsymbol{z})\big|_{\boldsymbol{z}=g(\boldsymbol{\epsilon},\boldsymbol{\phi})} \nabla_{\boldsymbol{\phi}} g(\boldsymbol{\epsilon}, \boldsymbol{\phi})], \end{aligned} \tag{5.2}$$

an expectation with form encompassing the ELBO. Explicit reparameterization is compatible with distributions in the location-scale family (e.g., Gaussian, Laplace, Cauchy), distributions with tractable inverse CDFs (e.g., exponential, KS, $\mathcal{CB}$), or those expressible as deterministic transformations of such distributions (e.g., $\tanh_{\mathcal{N}}$). When explicit reparameterization is not available, implicit reparameterization [42] is commonly used for distributions with numerically tractable CDFs, such as truncated, mixture, Gamma, Beta, Dirichlet, or von Mises distributions. This method expresses the parameter gradient through the sample $\nabla_{\boldsymbol{\phi}} \boldsymbol{z}$ as a function only of the CDF gradients, not its inverse. Such CDF gradients are either found analytically (if feasible) or more commonly using numerical methods, e.g., forward mode autodifferentiation on CDF estimates, as in the Gamma and Beta distributions.

Without explicit reparameterization, sampling and gradient computations tend to be slower and more complex, and produce higher-variance estimates of (5.2), reducing learning efficiency and stability [39, 43].

## 5.3 Stabilizing the Kumaraswamy

The KS distribution's utility relies on stable computation of its log-pdf, CDF, inverse CDF, and their gradients. In the standard parameterization, these functions contain instabilities from hidden $\log(1 - \exp(x))$ terms. We address this by introducing an unconstrained logarithmic parameterization that isolates these unstable terms, enabling their straightforward replacement with the stable `log1mexp` function. Finally, we show why naive stabilization techniques, such as parameter clipping, fail in high-dimensional applications.

**Identifying the instability: $\log(1 - \exp(x))$.** Naive computation of $\log(1 - \exp(x))$ for $x < 0$ leads to significant numerical errors as $x$ approaches 0 (Figure 5.2, red). These errors grow so large that they can cause *numerical instability*, i.e., an irrecoverable error such as `-inf`. These errors result from *catastrophic cancellation*, which occurs when subtracting nearly equal numbers — here, $1 - \exp(x)$. As $x \to 0$, $\exp(x) \approx 1$, so `1 - exp(x)` results in the cancellation of leading significant bits, leaving only a few less significant, less accurate bits to represent the result. This causes large relative errors in `1 - exp(x)`, which are amplified when input to the logarithm as its magnitude grows sharply near zero. If the cancellation is complete, `1 -`

`exp(x)` underflows to 0 and the logarithm returns `-inf`, as seen in Figure 5.2 (red) when $\log_2 |x| < -24$.

**Numerical building blocks for accurate $\log(1 - \exp(x))$ computation.** When $|x| \ll 1$, both $\log(1 + x)$ and $\exp(x) - 1$ can suffer from severe cancellation: the former between 1 and $x$, the latter between $\exp(x)$ and $-1$. In both cases, a simple solution for accurate computation is to use a few terms of the Taylor series, as

$$\texttt{log1p}(x) := \log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots, \quad \text{for } |x| < 1,$$
$$\texttt{expm1}(x) := \exp(x) - 1 = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, \quad \text{for } |x| < 1,$$

where $n!$ denotes the factorial. These functions form the basis for two common methods to compute $\log(1 - \exp(x))$: `log(-expm1(x))` and `log1p(-exp(x))`. [131] showed neither method provides sufficient accuracy across the domain. However, each approach is accurate in complementary regions, leading to

$$\texttt{log1mexp}(x) := \begin{cases} \texttt{log(-expm1(x))} & -\log 2 \leq x < 0 \\ \texttt{log1p(-exp(x))} & x < -\log 2, \end{cases} \tag{5.3}$$

which computes $\log(1 - \exp(x))$ accurately throughout single precision, shown in Figure 5.2 (blue).

**A stable Kumaraswamy.** The direct implementation of the KS's log-pdf and inverse CDF — as found in all core auto-differentiation libraries —

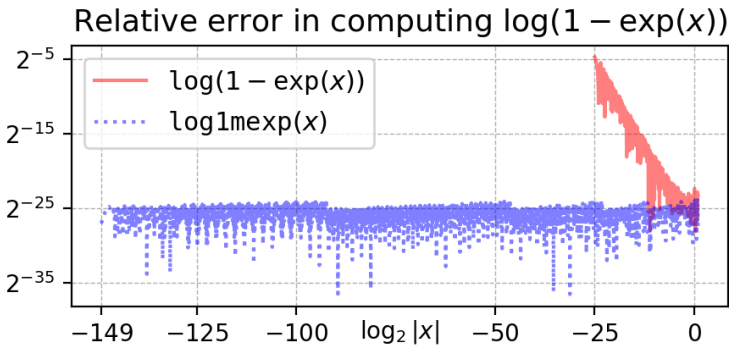


Figure 5.2: Naive computation of $\log(1 - \exp(x))$ (red) becomes unstable as $x \to 0$ due to catastrophic cancellation, while `log1mexp(x)` (blue) ensures accurate computation.

produces numerical instabilities. Here, we introduce a novel parameterization in terms of unconstrained logarithmic parameter values, which isolates and makes explicit the unstable terms

$$\begin{aligned} w_{b^{-1}}(u) &= \log(1 - u^{b^{-1}}) &&= \log(1 - \exp(b^{-1} \log u)) \\ w_a(x) &= \log(1 - x^a) &&= \log(1 - \exp(a \log x)), \end{aligned}$$

eliminates the need for positivity-enforcing link functions, and whose expressions involve only affine, exponential, and logarithmic transformations. This

allows the log-pdf and its gradients to be expressed as

$$\log f(x) = \log a + \log b + (a-1)\log x + (b-1)w_a(x) \tag{5.4}$$

$$\nabla_{\log x} \log f(x) = (a-1) - (b-1)\cdot \exp(a\log x - w_a(x) + \log a) \tag{5.5}$$

$$\nabla_{\log a} \log f(x) = 1 + a\log x \cdot \{1 - (b-1)\cdot \exp(a\log x - w_a(x))\} \tag{5.6}$$

$$\nabla_{\log b} \log f(x) = 1 + b\cdot w_a(x). \tag{5.7}$$

Likewise for the CDF

$$F(x) = 1 - (1-x^a)^b = 1 - \exp(b\cdot w_a(x)) \tag{5.8}$$

$$\nabla_x F(x) = \exp(\log a + \log b + (a-1)\cdot \log x + (b-1)\cdot w_a(x)) \tag{5.9}$$

$$\nabla_{\log a} F(x) = \exp(\log a + \log b + a\cdot \log x + (b-1)\cdot w_a(x))\cdot \log x \tag{5.10}$$

$$\nabla_{\log b} F(x) = \exp(\log b + b\cdot w_a(x))\cdot(-w_a(x)), \tag{5.11}$$

and the inverse CDF

$$F^{-1}(u) = (1-u^{b^{-1}})^{a^{-1}} = \exp(a^{-1}w_{b^{-1}}(u)) \tag{5.12}$$

$$\nabla_{\log a} F^{-1}(u) = \exp(-\log a + a^{-1}w_{b^{-1}}(u))\cdot(-w_{b^{-1}}(u)) \tag{5.13}$$

$$\nabla_{\log b} F^{-1}(u) = \exp(-\log a - \log b + b^{-1}\log u + (a^{-1}-1)w_{b^{-1}}(u))\cdot \log u. \tag{5.14}$$

This parameterization's algebraic form allows direct replacement of the dominant unstable terms, substituting $w_{b^{-1}}(u)$ with $\texttt{log1mexp}(b^{-1}\log u)$ and

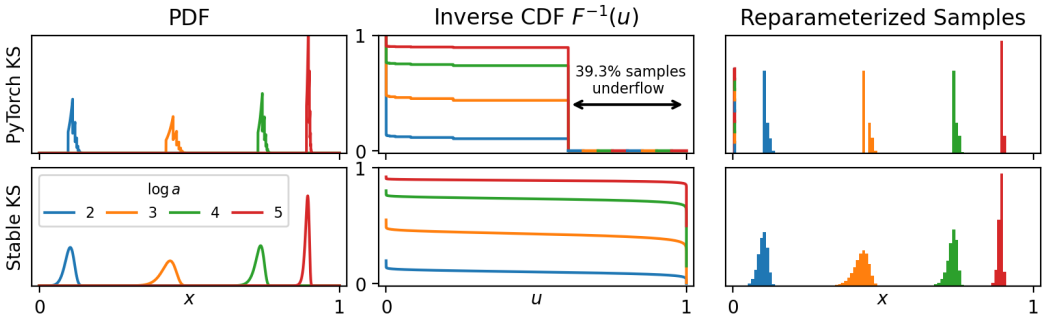


Figure 5.3: Stabilizing $\log(1-\exp(x))$ terms eliminates numerical instabilities in the KS log-pdf and inverse CDF. We compare the unstable PyTorch KS implementation (top row) and our stable KS (bottom row) for realistic KS distributions ($\log_2 b = 24$, varying $a$). Catastrophic cancellation in the $\log(1-\exp(x))$ terms in the PyTorch KS causes jagged curves and inverse CDF underflow beyond $u \approx 1 - 39.3$, resulting in a point mass of $\approx 39.3$ at $x = 0$ in the sampling distribution. Our stable KS removes the instability by using `log1mexp`.

$w_a(x)$ with $\texttt{log1mexp}(a \log x)$. Access to $\log a$ and $\log b$ avoids errors from unnecessary transitions in-and-out of log-space. We also avoid the error prone expressions produced in backpropogation's direct application of the chain rule, e.g.,

$$\nabla_{\log b} F^{-1} = \frac{1}{a} \cdot \exp\left(\frac{1}{a}\log\left(1\text{-}\exp\left(\frac{1}{b}\log u\right)\right)\right) \cdot \\ \text{-}\left(1\text{-}\exp\left(\frac{1}{b}\log u\right)\right)^{-1} \cdot \exp\left(\frac{1}{b}\log u\right) \cdot \log u \cdot \frac{-1}{b^2} \cdot b$$

and (5.14) are equivalent expressions for $\nabla_{\log b} F^{-1}$, but their computed values can differ greatly for extreme parameter values. Desirable KS distributions can obtain such problematic extreme parameter values, e.g., the KS distributions in Figure 5.3 have $b \approx 10^6$.

Figure 5.3 compares the PDF, inverse CDF, and histograms of reparameterized samples for KS distributions which are typical to real-world modeling scenarios. The PyTorch implementation (top row) shows jaggedness in both the PDF and inverse CDF, caused by catastrophic cancellation in the unstable terms $w_a(x)$ and $w_{b^{-1}}(u)$. Additionally, the PyTorch inverse CDF underflows beyond $u \approx 1 - 39.3$: here, $w_{b^{-1}}(u) = -\infty$, and $F^{-1}(u) = \exp(a^{-1} \cdot -\infty) = 0$. This underflow results in a point mass at $x = 0$ (a point outside of the KS support) with probability $\approx 39.3$ in each of the reparameterized sampling distributions, and produces infinite gradients via $\nabla_{\log a} F^{-1} = \infty$ [cf. (5.13)]. This infinite gradient triggers a cascade: infinite parameter values after the optimizer step and `NaN` activations in the next forward pass, which is what users ultimately observe when training fails.

### 5.3.1 The inadequacy of parameter clipping in large-scale settings

Numerical instability in the KS is inherently stochastic, and in high-dimensional settings, the compounded probability of failure across multiple variables makes program failure almost certain. As the program goes unstable if any single KS goes unstable, the overall instability probability can be modeled as the probability of at least one failure in $D$ independent Bernoulli trials: $1 - p(\text{KS stable})^D$, for $D$ KS latent variables. In practical large-scale settings, e.g., recommendation systems with $10^7$ users and $D = 10^7$ recommendation items, the probability of instability approaches 1 across all reasonable param-

eter clipping values, rendering clipping an ineffective stabilization strategy.

*Quantitative illustration.* Consider the stochastic instability arising from the term $\log(1 - \exp(b^{-1} \log u))$, where catastrophic cancellation occurs if $1 - \exp(b^{-1} \log u)$ becomes too small. To avoid logarithmic domain errors in single precision, we enforce $-b^{-1} \log u > 2^{-24}$ (Figure 5.2). We aim to select a $b_{\max}$ to satisfy this constraint: a larger $b_{\max}$ expands the variational family allowing improved posterior approximation, but worsens stability. Consider the moderate entropy KS distributions in Figure 5.3 which use $b = 2^{24}$. Using $b_{\max} = 2^{24}$, only $u < 0.6321$ satisfies the stability condition, i.e., $p(\text{KS stable}) \approx 0.6321$ per sample. With $D = 10^7$, the overall probability of instability becomes $1 - 0.6321^{10^7} \approx 1$. Now consider aggressively restricting $b_{\max} = 2^4$, as done in [132]. Now $u < 0.9999$ satisfies the stability condition. Even then, introducing $D = 10^7$ variables, we still have the overall probability of instability is $1 - 0.9999^{10^7} \approx 1$. Thus, even extreme clipping fails to stabilize KS distributions at scale. Further, this analysis considers only a single posterior sample. In practice, training with SVI requires $T \sim 10^3$ optimization steps, each requiring posterior samples for gradient estimation. This compounds the instability probability to $1 - p(\text{KS stable})^{TD}$, making clipping ineffective in realistic large-scale scenarios.

## 5.4 Experiments and New Variational Architectures

Using the well established Variational Auto-Encoder (VAE) framework on MNIST and CIFAR-10 datasets, we show that the stabilized KS enables reliable training as both a variational posterior [Eqns. (5.12)–(5.14)] and likelihood function [Eqns. (5.4)–(5.7)]. We then introduce two new deep variational architectures that leverage bounded interval-supported latent variables: the Variational Bandit Encoder (VBE) for improving exploration-exploitation trade-offs in contextual multi-armed bandits (Section 5.4.2), and the Variational Edge Encoder (VEE) for enhancing uncertainty quantification in link-prediction with graph neural networks (Section 5.4.3). These novel architectures tend to be most performant when using the KS as their variational posterior. Across the experimental domains, our stable KS tends to be more performant and easier to use than alternative variational distributions supported on bounded intervals. For instance, $\tanh_{\mathcal{N}}$ models require log-pdf clipping for stability, while Beta models show significant performance variability based on the chosen positivity-enforcing link function and often fail to converge, e.g., on CIFAR-10 in Section 5.4.1. Finally, our new variational models are fast: the VBEs in Section 5.4.2 are $8 - 22\times$ faster than the state-of-the-art baseline.

**Remark.** *Across all three experimental settings, models using the unstable KS produce* `NaN` *errors in training and are therefore excluded.* Prior work

using the KS in low-dimensional latent variable models [132–134] similarly find `NaN` errors, and rely on parameter clipping to avoid instability. See Section 5.3.1 for why this is approach does not work in large-scale settings. Our stabilization approach directly resolves these numerical issues, enabling stable training at scale.

### 5.4.1 Image variational auto-encoders

The VAE [39] is a generative latent variable model trained using amortized variational inference. Both the variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and the conditional likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ are parameterized using NNs, known as the encoder $e_{\boldsymbol{\phi}}(\boldsymbol{x}) : \mathbb{R}^M \mapsto \mathbb{R}^D$ and decoder $d_{\boldsymbol{\theta}}(\boldsymbol{z}) : \mathbb{R}^D \mapsto \mathbb{R}^M$, respectively. VAEs typically use the standard Normal distribution as the prior and a factorized Normal as the variational posterior. The use of alternative variational distributions allows incorporating different prior assumptions about the latent factors of the data, such as bounded support or periodicity [42].

**Experimental setup and metrics.** Inspired by [128], we train VAEs with fully factorized priors and variational posteriors on MNIST and CIFAR-10 without pixel binarization, using an unmodified ELBO ($\beta_{\text{KL}} = 1$). We adopt the most effective likelihoods from their work (Beta and $\mathcal{CB}$), identical latent dimension $D$ (MNIST: $D = 20$, CIFAR-10: $D = 50$), and the same standard NN architectures. For each variational posterior factor, we choose the canonical prior: $\mathcal{N}_{(0,1)}$ for $\mathcal{N}$, and $U_{(0,1)}$ for KS and Beta. We evaluate the models using a single sample approximation of the test ELBO. To assess

Table 5.2: VAE on MNIST and CIFAR-10.

| Prior $q_\phi(\boldsymbol{z}\|\boldsymbol{x})$ | $p_{\boldsymbol{\theta}}(\boldsymbol{x}\|\boldsymbol{z})$ | | MNIST | | CIFAR-10 | |
|---|---|---|---|---|---|---|
| | | | ELBO | $\mathcal{K}(\phi)$ | ELBO | $\mathcal{K}(\phi)$ |
| $\mathcal{N}_{(0,1)}$ | $\mathcal{N}$ | $\mathcal{CB}$ | **1825** | 97.3 | 1167 | 37.9 |
| $\mathrm{U}_{(0,1)}$ | KS | $\mathcal{CB}$ | 1818 | 97.4 | **1172** | **41.5** |
| $\mathrm{U}_{(0,1)}$ | Beta | $\mathcal{CB}$ | 1821 | **97.5** | 1167 | 40.3 |
| $\mathcal{N}_{(0,1)}$ | $\mathcal{N}$ | Beta | 4073 | **92.1** | **3566** | 48.5 |
| $\mathrm{U}_{(0,1)}$ | KS | Beta | 4061 | 91.3 | 3483 | **50.1** |
| $\mathrm{U}_{(0,1)}$ | Beta | Beta | **4082** | 90.1 | N/A | N/A |
| $\mathcal{N}_{(0,1)}$ | $\mathcal{N}$ | KS | 3328 | 96.4 | 1720 | 47.1 |
| $\mathrm{U}_{(0,1)}$ | KS | KS | **3355** | 96.8 | **1738** | **48.8** |
| $\mathrm{U}_{(0,1)}$ | Beta | KS | 3348 | **97.1** | N/A | N/A |

usefulness of the learned latent representations, we encode test data $\boldsymbol{x}_n$, compute the mean $\mathbb{E}[q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)]$, and classify the test labels using a 15-nearest neighbor classifier; the classifier accuracy (%) is denoted $\mathcal{K}(\boldsymbol{\phi})$. For subjective evaluation, we display the mean decoded likelihood of a single sample from the encoded posterior of random test digits in Figure 5.4.

**Discussion of results.** The sole purpose of this experiment is to provide evidence toward the stabilization of the KS. Notably, stable KS VAEs maintain numerical stability while all VAEs with the unstable KS produce unstable training. VAEs with Beta-distributed variational posteriors often do not converge; indeed, [42] reported strong performance on binarized MNIST using a softplus link function, but did not present results on CIFAR-10, nor could we find other works that did. We suspect this is due to similar instability issues, with the higher gradient variance of the Beta's implicit reparameterization

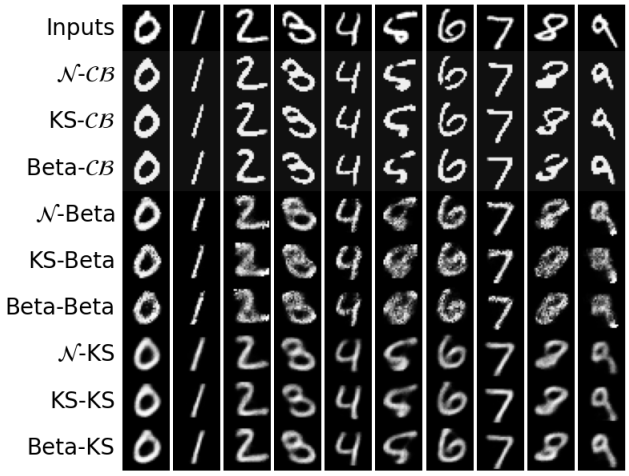


Figure 5.4: MNIST test digit VAE reconstructions.

a likely explanation. In an attempt to overcome this instability in Beta VAEs we report the best metrics across softplus or exp link functions in Table 5.2. When neither converges, we report N/A. The results in Table 5.2 show that across datasets, VAEs with KS-distributed variational posteriors consistently produce useful latent spaces, evidenced by strong $\mathcal{K}(\phi)$, and yield reconstructions with high ELBOs and visual quality.

When paired with any variational posterior, a KS likelihood yields stronger MNIST reconstructions than Beta likelihoods: compare rows $*$-Beta to $*$-KS in Table 5.4. As in [128], we find $\mathcal{CB}$ likelihoods produce the most subjectively performant VAEs on MNIST, unsurprising as $\mathcal{CB}$ was introduced specifically for the approximately binary MNIST pixel data.

### 5.4.2 Contextual Bernoulli multi-armed bandits

The contextual Bernoulli multi-armed bandit (MAB) problem involves a decision maker who, at each time step $t = 1, \ldots, T$, selects one arm from a finite set of $K$ options. Each arm has an associated context $\boldsymbol{x}_k \in \mathbb{R}^d$, and pulling an arm yields a binary reward $r_k \sim \text{Ber}(v_k)$, where $v_k \in [0, 1]$ is the unknown mean reward. MABs originate by analogy to casino slot machines, where each machine (arm) has a different payout rate, and the challenge lies in deciding which arms to pull in order to maximize total winnings while learning about their payout rates, a situation called the exploration-exploitation dilemma. MABs have found applications in modern recommendation systems [135], clinical trials design [136], and mobile health [137]. Thompson Sampling (TS) is a simple, empirically effective [138], and scalable [139] arm selection heuristic. It selects the arm corresponding to the highest value drawn from the posterior distributions over the latent $z_k$'s. This approach naturally balances exploration and exploitation: the uncertainty in the posteriors promote exploration, while concentration of probability mass on large mean rewards drive exploitation.

**Variational Bandit Encoder (VBE): VAE ∩ TS.** Our novel VBE posits a fully factorized KS variational posterior $\prod_k q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k)$, prior $p(\boldsymbol{z}) = U_{(0,1)}^K$, and a Bernoulli reward likelihood $p(r|v_k)$ for each arm. Similar to VAEs, we employ amortized inference using a shared NN encoder $e_{\boldsymbol{\phi}}(\boldsymbol{x}_k)$, which defines a reparameterizable variational distribution $q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k)$. However, unlike VAEs, VBEs omit the decoder; samples $\tilde{z}_k \sim q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k)$ directly

**Algorithm 2** Variational Bandit Encoder

**Require:** $\{\mathbf{x}_k\}_{k=1}^K$, $\{v_k\}_{k=1}^K$, $\eta$, $\beta_{\text{KL}}$

Variation posterior $q \leftarrow$ KS
Replay buffer $\mathcal{D} \leftarrow \emptyset$
**for** $t = 1 \dots T$ **do**
    Encode: $(a_k, b_k) = e_{\boldsymbol{\phi}}(\mathbf{x}_k)$
    Sample: $\tilde{z}_k \sim q(z_k; a_k, b_k)$
    TS: $a = \text{argmax}_k\{\tilde{z}_k\}$
    Reward: $r \sim \text{Ber}(v_a)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{x}_a, a, r)\}$
    Construct $\hat{\mathcal{L}}_{\beta_{\text{KL}}}$ as in (5.15)
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta \nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}_{\beta_{\text{KL}}}$
**end for**

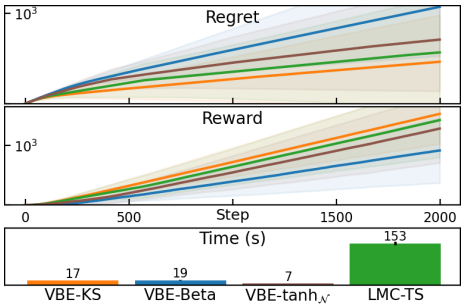


Figure 5.5: Synthetic bandit performance over 5 runs. VBE-KS best handles exploration-exploitation trade-offs.

parameterize the reward likelihood. The arm selection at step $t$ follows TS: $a = \text{argmax}_k\{\tilde{z}_k\}$. We then draw reward $r \sim \text{Ber}(v_a)$ and record it in the replay buffer $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{x}_a, a, r)\}$. We construct a sample approximation of the modified ELBO over the subset of arms $\mathcal{K}_t \subset \{1, \dots, K\}$ that have been pulled by time $t$ as

$$\hat{\mathcal{L}}_{\beta_{\text{KL}},t}(\mathcal{D}, \boldsymbol{\phi}) = \sum_{(\boldsymbol{x}_a, a, r) \in \mathcal{D}} \log p(r|\tilde{z}_a) + \beta_{\text{KL}} \sum_{k \in \mathcal{K}_t} \mathcal{H}[q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k)]. \tag{5.15}$$

The second term promotes exploration by penalizing overconfidence with the exploration effect proportional to $\beta_{\text{KL}}$. We maximize (5.15) w.r.t. $\boldsymbol{\phi}$ via gradient ascent, enabled by the reparameterizable KS. VBE execution is summarized in Algorithm 2.

**VBE advantages.** VBEs provide four primary advantages over alternative TS-based Bernoulli MAB approaches, discussed in Section 5.5

- *Scalability and Compatibility.* VBE training consists of a forward pass through a NN, sampling an explicitly reparameterized distribution, and a backward pass for gradient-based updates. This process is scalable and fully compatible with existing gradient-based infrastructure.

- *Prior Knowledge Incorporation.* When prior knowledge exists on an arm $k$ it can be efficiently encoded as $p(z_k) = \text{Beta}(a_k, b_k)$, replacing $\mathcal{H}[q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k)]$ with $-D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}_k) \,\|\, p(z_k)\right)$.

- *Interpretability and Independence.* Encoding $\boldsymbol{x}_k$ produces KS distribution parameters, fully encapsulating the model's beliefs about $v_k$. This is independent of other arms and past data.

- *Simplicity.* VBEs lack numerous hyperparameters and complex architectural components.

Alternative methods lack some or all of these properties because they do not directly model the mean rewards nor differentiate through mean reward samples; instead, they model the parameters $\boldsymbol{\phi}$.

**Experimental setup.** We construct synthetic data with $K = 10^4$ arms by first sampling a weight vector $\mathbf{w}$ and features $\{\mathbf{x}_k\}_{k=1}^K$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$. We then compute $\{\mathbf{w}^\top \mathbf{x}_k\}_{k=1}^K$ and apply min-max normalization to produce probabilities (referred to as "Original probabilities" in Figure 5.6). To introduce non-linearity, we raise these probabilities to the power 5 (shown as "Power (5) transformed probabilities" in Figure 5.6). Exponentiating the probabilities not only makes the mapping from features to mean rewards more challenging

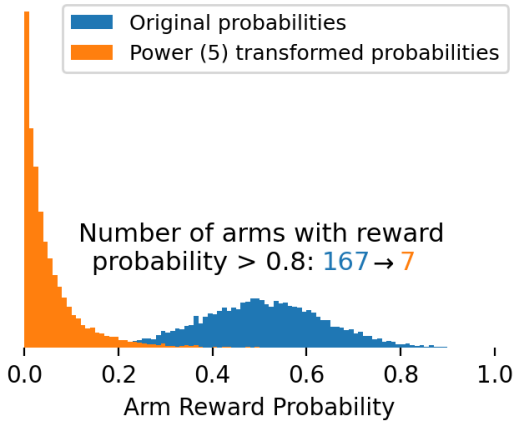


Figure 5.6: High arm reward probabilities are reduced via a power 5 exponentiation, encouraging exploration.

to learn, but it also significantly reduces the number of arms with high probabilities, forcing the agent to explore more. For instance, when raising the probabilities to the power of 5, the number of arms with large probabilities drops from 167 to just 7. We consider $T = 2 \cdot 10^3$ steps.

We evaluate VBEs with either a KS (VBE-KS), Beta (VBE-Beta), or $\tanh_{\mathcal{N}}$ (VBE-$\tanh_{\mathcal{N}}$) all using $\beta_{\text{KL}} = |\mathcal{K}_t|^{-1}$, which makes the second term in (5.15) a mean. VBE-$\tanh_{\mathcal{N}}$'s performance is sensitive to the number of samples used in its entropy estimate: we found degraded performance beyond 10 samples. The learning rate is set to $\eta = 10^{-2}$. As a baseline, we use LMC-TS, which employs Langevin Monte Carlo (LMC) to sample posterior parameters of a NN, known for state-of-the-art performance across various tasks [140]. All models use an MLP with 3 hidden layers of width 32. LMC-TS hyperparameters (inverse temperature, LMC steps, weight decay) are set or tuned based on the authors' code. We repeat experiments 5 times on an

Apple M2 CPU and report the mean and standard deviation across these runs in Figure 5.5.

**Metrics and evaluation.** The optimal policy always selects the arm with the highest mean reward $r^*$. Our objective is to minimize regret, defined as the cumulative difference between the expected reward of the chosen action and the optimal action (accessible in the synthetic setting), i.e., $\sum_{t=1}^{T}(r^* - r_{a_t})$. VBE-KS achieves lower regret and higher cumulative reward than all baselines. VBE-Beta performs significantly worse than VBE-KS and VBE-$\tanh_{\mathcal{N}}$, highlighting the importance of explicit reparameterization. LMC-TS is performant — worse than VBE-KS and better than VBE-$\tanh_{\mathcal{N}}$ — but is 8–22× slower than VBEs: VBEs avoid the computational overhead of LMC.

### 5.4.3 Variational link prediction with Graph Neural Networks

Graph Neural Networks (GNNs) have become a powerful tool for learning from graph-structured data, with applications in critical areas like drug discovery [141] and finance [142]. A key task is link prediction, where the goal is to infer unobserved edges between nodes. However, real-world deployment of graph learning models is often hindered by a lack of reliable uncertainty estimates and limited capacity to incorporate prior knowledge, see Chapter 4.To address these challenges, we propose a variational approach where the GNN encodes a KS to model the unobserved probabilities of each network link's existence, enabling uncertainty quantification and prior knowledge

integration with minimal computational overhead.

In a typical link prediction setup, the GNN has access to the features $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ of all $N$ nodes, but only a subset of positive edges in the training $\mathcal{D}_{tr}$ and validation $\mathcal{D}_{val}$ sets. Edges are labeled as 1 (present) or 0 (absent). The GNN generates edge embeddings through message passing and neighborhood aggregation, outputting probabilities $z_{u,v} \in (0, 1)$ that parameterize a Bernoulli likelihood. The seminal work of [74] proposed Variational Graph Auto-encoders (VGAEs), which posits a Gaussian variational posterior over the final *node* embeddings. When used for link prediction it samples final node embeddings from the variational posterior and decodes them to produce edge probabilities. In contrast, our approach directly models the probability of an edge using the KS. An advantage of directly modeling edge probabilities is interpretability; deep nodal embeddings are often difficult to interpret, and priors are typically selected for computational tractability rather than their ability to incorporate meaningful prior information. However, the probability of an edge $(u, v)$ existing between two nodes is an interpretable quantity that can often be informed by domain expertise. For example, in gene regulatory networks, epidemiological networks, and social networks experts often have prior knowledge about the likelihood of specific interactions, transmissions, or friendships, respectively, which can be directly incorporated into edge prior $p(z_{(u,v)})$. We believe the limited exploration of variational modeling for edge probabilities is due to the previous lack of an expressive, stable, explicitly reparameterizable bounded-interval distributions.

**Variational Edge Encoder (VEE).** We propose a fully factorized KS variational posterior $\prod_{(u,v)\in\mathcal{D}_{tr}} q_{\boldsymbol{\phi}}(z_{u,v}|\boldsymbol{X},\mathcal{D}_{tr})$ with a uniform prior $p(\boldsymbol{z}) = U_{(0,1)}^{|\mathcal{D}_{tr}|}$. The GNN encoder $e_{\boldsymbol{\phi}}$ parameterizes a KS distribution for each possible edge $(u,v)$. The remaining structure is highly similar to VBEs: a single sample $\tilde{z}_{u,v} \sim q_{\boldsymbol{\phi}}(z_{u,v}|\boldsymbol{X},\mathcal{D}_{tr})$ directly parameterizes the Bernoulli likelihood, and we maximize a sample approximation of the modified ELBO

$$\hat{\mathcal{L}}_{\beta_{\mathrm{KL}}}((\boldsymbol{X},\mathcal{D}_{tr}),\boldsymbol{\phi}) = \sum_{(u,v)\in\mathcal{D}_{tr}} \log p(z_{(u,v)}|\boldsymbol{X},\mathcal{D}_{tr}) + \beta_{\mathrm{KL}} \sum_{(u,v)\in\mathcal{D}_{tr}} \mathcal{H}[q_{\boldsymbol{\phi}}(z_{(u,v)}|\boldsymbol{X},\mathcal{D}_{tr})]. \tag{5.16}$$

From their similarity with VBEs, VEEs inherit the same advantages outlined in Section 5.4.2.

**Models, metrics, and datasets.** All models use a 2-layer GNN with Graph Convolutional Network (GCN) layers and a hidden/output nodal dimension of 32. In Base-GNN, an MLP decodes the final nodal embeddings into link probabilities. In VEE-KS/Beta/tanh$_{\mathcal{N}}$ an MLP parameterizes the KS/Beta/tanh$_{\mathcal{N}}$ variational distributions; all take $\beta_{\mathrm{KL}} = .05|\mathcal{D}_{tr}|^{-1}$. We use 10 samples in tanh$_{\mathcal{N}}$'s entropy estimate; more did not produce significant

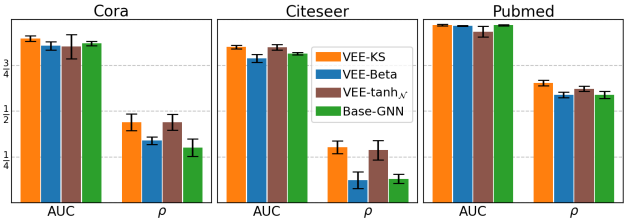


Figure 5.7: VEE-KS produces informative and calibrated edge posterior predictives across graph datasets.

performance differences. We train for 300 epochs, with a learning rate of .01, averaging results over 5 runs with different seeds. The posterior predictive distribution over binary links $p(\boldsymbol{A}|\boldsymbol{X}, D_{tr}) = \int p(\boldsymbol{A}|\boldsymbol{Z})q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X}, D_{tr})d\boldsymbol{Z}$ is estimated by using a single sample from each KS/Beta distribution, parameterizing each edge Bernoulli distribution with such samples, followed by sampling binary edges. For Base-GNN we directly sample binary edges from the likelihood. Using 30 posterior predictive samples, we compute the edge-wise posterior predictive mean (pred. mean) and standard deviation (pred. stdv.). We report the Pearson correlation $\rho$ between predictive uncertainty (pred. stdv.) and error ($\ell_1$ difference between pred. mean and the true label), as a measure of uncertainty calibration: useful uncertainty estimates should show strong associations between uncertainty and error. Additionally, we compute area under the ROC curve (AUC) using pred. mean as a predictor. Figure 5.7 shows performance across 3 standard citation networks: Cora, Citeseer, and Pubmed.

**Discussion of results.** On all datasets and all metrics, VEE-KS outperforms or matches the most performant baselines, providing higher predictive accuracy (AUC) and better uncertainty calibration (higher $\rho$). Similar to Section 5.4.2, we find Beta distributed variational posteriors perform significantly worse than those using KS or $\tanh_{\mathcal{N}}$, further underlining the importance of explicit reparameterization. Moreover, models using explicitly reparameterizable latents are faster: on the largest dataset (Pubmed), the average time (ms) per epoch for VEE-KS, VEE-$\tanh_{\mathcal{N}}$, and VEE-Beta was $381 \pm 61$,

$301 \pm 26$, and $447 \pm 86$ respectively, on an Apple M2 CPU.

## 5.5 Related Work

**VBEs in context: TS-based Bernoulli MAB approaches.** Existing TS-based approaches for Bernoulli MABs assume a prior over model parameters $p(\boldsymbol{\phi})$, which map contexts to rewards through $e_{\boldsymbol{\phi}}$. At each round, parameters are sampled from the posterior, $\tilde{\boldsymbol{\phi}}_t \sim p(\boldsymbol{\phi}|\mathcal{D})$, and used to compute mean reward posterior samples $\{e_{\tilde{\boldsymbol{\phi}}_t}(\boldsymbol{x}_k)\}_{k=1}^K$. However, the Bernoulli likelihood often leads to intractable posteriors, making parameter sampling difficult. Common methods use either variational approximations [138, 143, 144], primarily Laplace, or MCMC approaches like Gibbs sampling [145] or LMC [140]. These approaches face several limitations. First, incorporating prior knowledge is challenging since the relationship between a parameter's value and its effect on rewards is often unclear, except in the simplest models. Second, scalability is a concern: Laplace approximations become inefficient with large context dimensions or model sizes, while MCMC-based methods are compute and memory intensive, requiring long burn-in periods (typically $10^2$ iterations) and large machine memory to store the buffer $\mathcal{D}$. Third, interpreting model beliefs over mean rewards requires drawing numerous posterior samples, adding further computational cost. Finally, these methods often introduce significant complexity through intricate algorithms, architectures, optimization steps, and hyperparameters, particularly MCMC parameters (e.g., burn-in iterations,

chain length, LMC inverse temperature/weight decay and their respective schedules). By directly modeling mean rewards with a KS, instead of the parameters $\boldsymbol{\phi}$, VBEs offer a simple, scalable, and interpretable approach to Bernoulli MABs.

**Kumaraswamy as a Beta surrogate.** A simple approach to overcome the Beta distribution's lack of explicit reparameterization is to use the KS as a surrogate. This surrogate approach is feasible due to their significant similarities when defined by the same two parameters and the availability of a high-fidelity closed-form approximation of the KL divergence between Beta and KS distributions. [132, 133] use KSs as surrogates for Betas in the Dirichlet Process stick-breaking construction to allow for stochastic latent dimensionality in a VAE. However, both require parameter clipping for numerical stability. In their published code [132] constrains KS parameters $\log a, \log b \in [-2.3, 2.9]$, significantly limiting the expressiveness of latent KS distributions. Also, [133] comments under a *Computational Issues* section that 'If NaNs are encountered...clipping the parameters of the variational Kumaraswamys usually solve the problem.' [134] improved upon [132] by resolving the order-dependence issue in approximating a Beta with a KS. Similarly, [146] followed a comparable process using an Indian Buffet Process. Both works maintained numerical stability by restricting the uniform base distribution's support from the unit interval to a narrower interval, before passing the samples through the inverse CDF producing a distortion of the reparameterized sampling distribution. This work eliminates the need for such

distortions, enabling more accurate Beta approximations and simplifying the use of the KS distribution by ensuring numerical stability without additional interventions.

# Chapter 6

# Summary

Here we summarize the main results of this thesis and outline exciting directions for future work.

In Chapter 3, we introduced the Graph Deconvolution Network (GDN), an inductive model designed to infer latent graph structures from observations of their convolutional mixtures. By leveraging the principle of algorithm unrolling, we developed a novel deep learning framework to address the network inverse (deconvolution) problem in a supervised setting. This approach is particularly relevant for applications such as predicting whole-brain structural connectivity from functional signals. GDNs operate by minimizing a task-specific loss function, learning filters that iteratively refine initial graph estimates layer by layer. Additionally, the unrolled architecture naturally integrates domain-specific prior knowledge about the unknown graph distribution by encoding it as input during the initial refinement step. A key advantage of GDNs is their node permutation equivariance property, ensuring that the inferred graph topologies remain invariant to arbitrary vertex label-

ings. Furthermore, GDNs exhibit two important characteristics: (i) they are differentiable with respect to both their parameters and graph input, and (ii) they allow explicit control over model complexity, enabling efficient and fast inference. These features position GDNs as promising components in larger, end-to-end graph representation learning pipelines, including online systems. While the primary focus of this chapter was on graph structure learning, the potential applications of GDNs extend to a wide range of graph inference tasks.

In Chapter 4, we identify independent interpretability of (regularization) parameters in inverse problems as a key property, which allows one to incorporate prior information on solution characteristics into prior distributions over the NN parameters of an unrolled optimization algorithm. We explore this concept in the context of graph structure learning (GSL) from smooth signals and propose a simplified optimization algorithm that achieves competitive estimation performance while being simpler than existing approaches. By unrolling this algorithm, we introduce the first true neural network specifically designed for GSL from smooth signal observations. Leveraging the independent interpretability, we integrate prior knowledge about graph sparsity into prior distributions over the parameters of the unrolled network. This leads to the development of the first Bayesian Neural Network (BNN) for GSL from smooth signals. The advantages of unrolled architectures — such as low parameter dimensionality, efficient layers that are compiler — and autograd-friendly, and the empirical suitability of shallow networks—enable us to

perform posterior approximation using Markov Chain Monte Carlo (MCMC) sampling. This approach provides high-quality, well-calibrated uncertainty estimates for edge predictions, as demonstrated through extensive experiments on both synthetic and real datasets.

In Chapter 5, we addressed and resolved critical numerical instabilities in the Kumaraswamy (KS) distribution, which stands out as an appealing choice for variational models involving bounded latent variables. By stabilizing the KS distribution, we demonstrated its ability to power a diverse range of large-scale machine learning applications through simple yet effective deep variational models. Specifically, we introduced the Variational Bandit Encoder, which improves exploration-exploitation trade-offs in contextual Bernoulli multi-armed bandits (MABs), and the Variational Edge Encoder, which enhances uncertainty quantification in link prediction tasks using graph neural networks (GNNs). Empirical results show that these models are both efficient and highly performant, achieving their best outcomes when leveraging the stabilized KS distribution. Unlike alternatives such as the Beta or $\tanh_{\mathcal{N}}$ distributions, the KS avoids the numerical instability and added complexity often associated with these approaches. Beyond the specific applications explored, these models open up new possibilities for tackling other large-scale challenges, including recommendation systems, reinforcement learning with continuous bounded action spaces, network analysis, and uncertainty-aware deep learning.

## 6.1 Future work

The primary limitation of the approach presented in Chapter 4 is its scalability to moderate- and large-sized graphs and datasets. Achieving asymptotically exact posterior inference via MCMC sampling entails significant computational and memory demands. This is due to the need for multiple forward passes through the network per posterior sample, as well as the requirement to store the entire dataset in memory. Future work could address these challenges by exploring non-asymptotically exact inference methods, such as variational inference. These methods are computationally efficient, support mini-batch training, and enable the use of larger datasets [94]. Additionally, since the approach generates distributions over output graphs, future research could investigate incorporating Bayesian decision analysis through the introduction of a utility function. Another promising direction involves using DPG as a module within a larger system, where propagation of uncertainty over the inferred graph is important.

Generalizations of the Kumaraswamy (KS) distribution [147], addressed Chapter 5, inherit numerical instabilities related to $\log(1 - \exp(x))$, which could be addressed in future work by extending our stabilization techniques. One limitation of the current models is their inability to represent multimodal posteriors. Future research could investigate KS mixtures or hierarchical latent spaces to overcome this limitation. Additionally, optimizing the $\beta_{\mathrm{KL}}$ parameter using approaches such as warm-up schedules [41] offers a promising

avenue for improving performance. The stabilized KS distribution also has potential applications in non-parametric models, such as Dirichlet Processes, building on prior work [133, 134]. Perhaps the most promising direction includes expanding upon the Variational Bandit Encoder (VBE), including real-data applications and conducting a theoretical analysis of regret bounds and asymptotic properties. Such analyses would enhance its utility in critical domains such as clinical trials, where robust and reliable decision-making under uncertainty is essential.

# Bibliography

[1] E. D. Kolaczyk, *Statistical Analysis of Network Data: Methods and Models*. Springer-Verlag, 2009.

[2] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2020.

[3] X. Cai, J. A. Bazerque, and G. B. Giannakis, "Inference of gene regulatory networks with sparse structural equation models exploiting genetic perturbations," *PLoS Comput. Biol.*, 2013.

[4] G. Marti, F. Nielsen, M. Bińkowski, and P. Donnat, "A review of two decades of correlations, hierarchies, networks and clustering in financial markets," *Prog. Inf. Geom. Theory Appl.*, 2021.

[5] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, 2018.

[6] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, 2013.

[7] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, 2013.

[8] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *JMLR*, 2022.

[9] X. Dong, D. Thanou, L. Toni, M. Bronstein, and P. Frossard, "Graph signal processing for machine learning: A review and new perspectives," *IEEE Signal Processing Magazine*, 2020.

[10] O. Sporns, *Networks of the Brain*. MIT Press, 2011.

[11] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process. Mag.*, 2019.

[12] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 44–63, 2019.

[13] S. S. Saboksayr, G. Mateos, and M. Cetin, "Online discriminative graph learning from multi-class smooth signals," *Signal Process.*, 2021.

[14] A. P. Dempster, "Covariance selection," *Biometrics*, 1972.

[15] E. Pavez, H. E. Egilmez, and A. Ortega, "Learning graphs with monotone topology properties and multiple connected components," *IEEE Trans. Signal Process.*, 2018.

[16] S. Kumar, J. Ying, J. V. de M. Cardoso, and D. P. Palomar, "A unified framework for structured graph learning via spectral constraints," *JMLR*, 2020.

[17] S. Segarra, A. G. Marques, G. Mateos, and A. Ribeiro, "Network topology inference from spectral templates," *IEEE Trans. Signal Inf. Process. Networks*, 2017.

[18] R. Shafipour and G. Mateos, "Online topology inference from streaming stationary graph signals with partial connectivity information," *Algorithms*, vol. 13, no. 9, pp. 1–19, 2020.

[19] V. Kalofolias, "How to learn a graph from smooth signals," in *Artificial Intelligence and Statistics*, 2016.

[20] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning Laplacian matrix in smooth graph signal representations," *IEEE Trans. Signal Process.*, vol. 64, no. 23, pp. 6160–6173, 2016.

[21] V. Kalofolias, A. Loukas, D. Thanou, and P. Frossard, "Learning time varying graphs," in *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, 2017.

[22] V. Kalofolias and N. Perraudin, "Large scale graph learning from smooth signals," in *ICLR*, 2019.

[23] P. Berger, G. Hannak, and G. Matz, "Efficient graph learning from noisy and incomplete data," *IEEE Trans. Signal Inf. Process. Netw.*, 2020.

[24] G. B. Giannakis, Y. Shen, and G. V. Karanikolas, "Topology identification and learning over graphs: Accounting for nonlinearities and dynamics," *Proc. IEEE*, 2018.

[25] M. Wasserman, S. Sihag, G. Mateos, and A. Ribeiro, "Learning graph structure from convolutional mixtures," *TMLR*, 2023.

[26] M. Wasserman and G. Mateos, "Graph structure learning with interpretable Bayesian neural networks," *TMLR*, 2024.

[27] M. Wasserman and G. Mateos, "Stabilizing the kumaraswamy distribution," *TMLR*, 2025.

[28] L. Goldsberry, W. Huang, N. F. Wymbs, S. T. Grafton, D. S. Bassett, and A. Ribeiro, "Brain signal analytics from graph signal processing perspective," in *ICASSP*, 2017.

[29] M. H. DeGroot, "Reaching a consensus," *J Am Stat Assoc.*, 1974.

[30] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[31] E. Isufi, F. Gama, D. I. Shuman, and S. Segarra, "Graph filters for signal processing and machine learning on graphs," *IEEE Transactions on Signal Processing*, 2024.

[32] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009.

[33] S. S. Saboksayr, G. Mateos, and M. Cetin, "Online graph learning under smoothness priors," in *Proc. of European Signal Process. Conf.*, 2021.

[34] X. Wang, C. Yao, H. Lei, and A. M.-C. So, "An efficient alternating direction method for graph learning from smooth signals," in *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, 2021.

[35] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *ICML*, 2010, p. 399–406.

[36] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, 2021.

[37] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, 2013.

[38] T. S. Jaakkola and M. I. Jordan, "Bayesian parameter estimation via variational methods," *Stat. Comput.*, 2000.

[39] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *ICLR*, 2014.

[40] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *ICML*, 2014.

[41] A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy, "Fixing a broken ELBO," in *ICML*, 2018.

[42] M. Figurnov, S. Mohamed, and A. Mnih, "Implicit reparameterization gradients," *NeurIPS*, 2018.

[43] E. Jang, S. Gu, and B. Poole, "Categorical reparametrization with Gumble-Softmax," in *ICLR*, 2017.

[44] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017.

[45] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. M. Bronstein, "Differentiable graph module (dgm) for graph convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2023.

[46] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, 2019.

[47] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Trans. Signal Process.*, 2020.

[48] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," in *AAAI*, 2021.

[49] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM J. Comput.*, 2011.

[50] B. Pasdeloup, V. Gripon, G. Mercier, D. Pastor, and M. G. Rabbat, "Characterization and inference of graph diffusion processes from observations of stationary signals," *IEEE Trans. Signal Inf. Process. Netw.*, 2017.

[51] S. Feizi, D. Marbach, M. Medard, and M. Kellis, "Network deconvolution as a general method to distinguish direct dependencies in networks," *Nat. Biotechnol*, 2013.

[52] P. Sprechmann, A. M. Bronstein, and G. Sapiro, "Learning efficient sparse and low rank models," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2015.

[53] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron, "From local structures to size generalization in graph neural networks," in *ICML*, 2021.

[54] F. Abdelnour, H. U. Voss, and A. Raj, "Network diffusion accurately models the relationship between structural and functional brain connectivity networks," *Neuroimage*, 2014.

[55] H. Liang and H. Wang, "Structure-function network mapping and its assessment via persistent homology," *PLOS Comput. Biol*, 2017.

[56] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, 2008.

[57] M. Drton and M. H. Maathuis, "Structure learning in graphical modeling," *Annu. Rev. Stat. Appl*, 2017.

[58] H. Daneshmand, M. Gomez-Rodriguez, L. Song, and B. Schoelkopf, "Estimating diffusion network structures: Recovery conditions, sample complexity & soft-thresholding algorithm," in *ICML*, 2014.

[59] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning Laplacian matrix in smooth graph signal representations," *IEEE Trans. Signal Process.*, 2016.

[60] P. Veličković, L. Buesing, M. C. Overlan, R. Pascanu, O. Vinyals, and C. Blundell, "Pointer graph networks," in *NeurIPS*, 2020.

[61] T. N. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *ICML*, 2018.

[62] H. Shrivastava, X. Chen, B. Chen, G. Lan, S. Aluru, and L. Song, "GLAD: Learning sparse graph recovery," in *ICLR*, 2020.

[63] X. Pu, T. Cao, X. Zhang, X. Dong, and S. Chen, "Learning to learn graph topologies," in *NeurIPS*, 2021.

[64] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 128–138, 2020.

[65] A. Barrat, M. Barthelemy, and A. Vespignani, *Dynamical Processes on Complex Networks*. New York, NY: Cambridge University Press, 2008.

[66] M. Yuan and Y. Lin, "Model selection and estimation in the Gaussian graphical model," *Biometrika*, 2007.

[67] J. Li, J. Li, Y. Liu, Y. L. Jianwei Yu, and H. Cheng, "Deconvolutional networks on graph data," in *NeurIPS*, 2021.

[68] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[69] C.-H. Yeh, D. K. Jones, X. Liang, M. Descoteaux, and A. Connelly, "Mapping structural connectivity using diffusion MRI: Challenges and opportunities," *J. Magn. Reson.*, 2021.

[70] J. S. Damoiseaux and M. D. Greicius, "Greater than the sum of its parts: A review of studies combining structural connectivity and resting-state functional connectivity," *Brain Struct. Func.*, 2009.

[71] Z. Gu, K. W. Jamison, M. R. Sabuncu, and A. Kuceyeski, "Heritability and interindividual variability of regional structure-function coupling," *Nat. Commun*, 2021.

[72] G. E. Hinton, "Advanced machine learning: Recurrent neural networks," 2013.

[73] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[74] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *NeurIPS Worksh. on Bayes. Deep Learn.*, 2016.

[75] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NeurIPS*, 2018.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[77] R. Zhang, S. Fattahi, and S. Sojoudi, "Large-scale sparse inverse covariance estimation via thresholding and max-det matrix completion," in *ICML*, 2018.

[78] M. F. Glasser, S. M. Smith, D. S. Marcus, J. L. Andersson, E. J. Auerbach, T. E. Behrens, T. S. Coalson, M. P. Harms, M. Jenkinson, S. Moeller *et al.*, "The human connectome project's neuroimaging approach," *Nat. Neurosci.*, 2016.

[79] D. C. Van Essen, S. M. Smith, D. M. Barch, T. E. Behrens, E. Yacoub, K. Ugurbil, W.-M. H. Consortium *et al.*, "The WU-Minn Human Connectome Project: An overview," *Neuroimage*, 2013.

[80] R. S. Desikan, F. Ségonne, B. Fischl, B. T. Quinn, B. C. Dickerson, D. Blacker, R. L. Buckner, A. M. Dale, R. P. Maguire, B. T. Hyman *et al.*, "An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest," *Neuroimage*, 2006.

[81] J. Zimmermann, P. Ritter, K. Shen, S. Rothmeier, M. Schirner, and A. R. McIntosh, "Structural architecture supports functional organization in the human aging brain at a regionwise and network level," *Hum. Brain Mapp*, 2016.

[82] M. Génois and A. Barrat, "Can co-location be used as a proxy for face-to-face contacts?" *EPJ Data Science*, vol. 7, no. 1, p. 11, May 2018.

[83] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, 2017.

[84] L. Cosmo, A. Kazi, S.-A. Ahmadi, N. Navab, and M. Bronstein, "Latent-graph learning for disease prediction," in *Med. Image Comput. Comput. Assist. Intervent.*, 2020.

[85] S. S. Saboksayr and G. Mateos, "Accelerated graph learning from smooth signals," *IEEE Signal Process. Lett.*, 2021.

[86] X. Wang, C. Yao, and A. M.-C. So, "A linearly convergent optimization framework for learning graphs from smooth signals," *IEEE Trans. Signal Inf. Process. Netw.*, 2023.

[87] X. Pu, T. Cao, X. Zhang, X. Dong, and S. Chen, "Learning to learn graph topologies," in *NeurIPS*, 2021.

[88] H. Shrivastava, X. Chen, B. Chen, G. Lan, S. Aluru, H. Liu, and L. Song, "GLAD: Learning sparse graph recovery," in *ICLR*, 2020.

[89] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[90] J. Gabry, D. Simpson, A. Vehtari, M. Betancourt, and A. Gelman, "Visualization in bayesian workflow," *arXiv preprint arXiv:1709.01449*, 2017.

[91] C. T. Butts, "Network inference, error, and informant (in) accuracy: a Bayesian approach," *Soc. Netw.*, 2003.

[92] F. W. Crawford, "Hidden network reconstruction from information diffusion," in *Int. Conf. Inform. Fusion.*, 2015.

[93] C. Gray, L. Mitchell, and M. Roughan, "Bayesian inference of network structure from information cascades," *IEEE Trans. Signal Inf. Process. Netw.*, 2020.

[94] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on Bayesian neural networks—A tutorial for deep learning users," *IEEE Comput. Intell. Mag.*, 2022.

[95] F. Wenzel, K. Roth, B. Veeling, J. Swiatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin, "How good is the Bayes posterior in deep neural networks really?" in *ICML*, 2020.

[96] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, 2016.

[97] A. Beck and M. Teboulle, "A fast dual proximal gradient algorithm for convex minimization and applications," *Operations Research Letters*, 2014.

[98] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum likelihood network topology identification from edge-based unicast measurements," *ACM SIGMETRICS Perform. Eval. Rev.*, 2002.

[99] S. Shaghaghian and M. Coates, "Bayesian inference of diffusion networks with unknown infection times," in *Proc. IEEE Workshop on Statistical Signal Process.*, 2016.

[100] X. Jiang and E. D. Kolaczyk, "A latent eigenprobit model with link uncertainty for prediction of protein–protein interactions." *Stat. Biosci.*, 2011.

[101] S. A. Williamson, "Nonparametric network models for link prediction," *JMLR*, 2016.

[102] S. Pal and M. Coates, "Scalable MCMC in degree corrected stochastic block model," in *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, 2019.

[103] T. Deleu, A. Góis, C. C. Emezue, M. Rankawat, S. Lacoste-Julien, S. Bauer, and Y. Bengio, "Bayesian structure learning with generative flow networks," in *AAAI*, 2022.

[104] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *AAAI*, 2019.

[105] S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non-parametric graph learning for bayesian graph neural networks," in *Proc. Conf. Uncertain. Artif. Intell.*, ser. PMLR, 2020.

[106] F. Opolka and P. Lió, "Bayesian link prediction with deep graph convolutional gaussian processes," in *PMLR*, 2022.

[107] M. Sevilla and S. Segarra, "Bayesian topology inference on partially known networks from input-output pairs," *arXiv preprint arXiv:2309.06532*, 2023.

[108] R. Barbano, Z. Kereta, C. Zhang, A. Hauptmann, S. Arridge, and B. Jin, "Quantifying sources of uncertainty in deep learning-based image reconstruction," in *NeurIPS Work. Deep Learn. Inverse Probl.*, 2020.

[109] C. Zhang, R. Barbano, and B. Jin, "Conditional variational autoencoder for learned image reconstruction," *Computation*, 2021.

[110] C. Ekmekci and M. Cetin, "Uncertainty quantification for deep unrolling-based computational imaging," *IEEE Trans. Comput. Imaging*, vol. 8, 2022.

[111] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," *NeurIPS*, vol. 14, 2001.

[112] S. P. Chepuri, S. Liu, G. Leus, and A. O. Hero, "Learning sparse graphs under smoothness prior," in *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, 2017.

[113] W. Huang, A. G. Marques, and A. R. Ribeiro, "Rating prediction via graph signal processing," *IEEE Trans. Signal Process.*, 2018.

[114] C. Hu, L. Cheng, J. Sepulcre, G. El Fakhri, Y. M. Lu, and Q. Li, "A graph theoretical regression model for brain connectivity learning of alzheimer's disease," in *IEEE 10th Int. Symp. Biomed. Imaging*, 2013.

[115] X. Wang, C. Yao, H. Lei, and A. M.-C. So, "An efficient alternating direction method for graph learning from smooth signals," in *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, 2021.

[116] G. Fatima, A. Arora, P. Babu, and P. Stoica, "Learning sparse graphs via majorization-minimization for smooth node signals," *IEEE Signal Process. Lett.*, vol. 29, pp. 1022–1026, 2022.

[117] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. G. Wilson, "What are bayesian neural network posteriors really like?" in *ICML*, 2021.

[118] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, 3rd ed. Chapman and Hall/CRC, 1995.

[119] A. Gelman, D. Simpson, and M. Betancourt, "The prior can often only be understood in the context of the likelihood," *Entropy*, 2017.

[120] M. Betancourt, "Diagnosing suboptimal cotangent disintegrations in hamiltonian monte carlo," *arXiv preprint arXiv:1604.00695*, 2016.

[121] M. Sharma, S. Farquhar, E. Nalisnick, and T. Rainforth, "Do bayesian neural networks need to be fully stochastic?" *arXiv preprint arXiv:2211.06291*, 2022.

[122] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *ICML*, 2017.

[123] D. Phan, N. Pradhan, and M. Jankowiak, "Composable effects for flexible and accelerated probabilistic programming in numpyro," in *NeurIPS Workshop Program Transform. ML*, 2019.

[124] J. V. de Miranda Cardoso, J. Ying, and D. Palomar, "Graphical models in heavy-tailed markets," *NeurIPS*, 2021.

[125] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998.

[126] M. Jones, "Kumaraswamy's distribution: A beta-type distribution with some tractability advantages," *Stat. Methodol.*, 2009.

[127] P. Kumaraswamy, "A generalized probability density function for double-bounded random processes," *J. Hydrol.*, 1980.

[128] G. Loaiza-Ganem and J. P. Cunningham, "The continuous Bernoulli: Fixing a pervasive error in variational autoencoders," in *NeurIPS*, 2019.

[129] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[130] J. Björck, X. Chen, C. De Sa, C. P. Gomes, and K. Weinberger, "Low-precision reinforcement learning: running soft actor-critic in half precision," in *ICML*, 2021.

[131] M. Mächler, "Accurately computing $\log(1 - \exp(-|a|))$ assessed by the Rmpfr package," *CRAN*, 2012.

[132] E. Nalisnick, L. Hertel, and P. Smyth, "Approximate inference for deep latent Gaussian mixtures," in *NeurIPS Worksh. on Bayes. Deep Learn.*, 2016.

[133] E. Nalisnick and P. Smyth, "Stick-breaking variational autoencoders," in *ICLR*, 2017.

[134] A. Stirn, T. Jebara, and D. Knowles, "A new distribution on the simplex with auto-encoding applications," in *NeurIPS*, 2019.

[135] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *WWW*, 2010.

[136] S. S. Villar, J. Bowden, and J. Wason, "Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges," *Stat. Sci.*, 2015.

[137] A. Tewari and S. A. Murphy, "From ads to interventions: Contextual bandits in mobile health," *Mob. Health: Sens. Analyt. Methods Appl.*, 2017.

[138] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *NeurIPS*, 2011.

[139] K.-S. Jun, A. Bhargava, R. Nowak, and R. Willett, "Scalable generalized linear bandits: Online computation and hashing," in *NeurIPS*, 2017.

[140] P. Xu, H. Zheng, E. V. Mazumdar, K. Azizzadenesheli, and A. Anandkumar, "Langevin Monte Carlo for contextual bandits," in *ICML*, 2022.

[141] Z. Zhang, L. Chen, F. Zhong, D. Wang, J. Jiang, S. Zhang, H. Jiang, M. Zheng, and X. Li, "Graph neural network approaches for drug-target interactions," *Curr. Opin. Struct. Biol.*, 2022.

[142] J. Wang, S. Zhang, Y. Xiao, and R. Song, "A review on graph neural network methods in financial applications," *J. Data Sci.*, 2022.

[143] I. Urteaga and C. Wiggins, "Variational inference for the multi-armed contextual bandit," in *AISTATS*, 2018.

[144] P. Clavier, T. Huix, and A. Durmus, "VITS: Variational inference Thomson sampling for contextual bandits," in *ICML*, 2024.

[145] B. Dumitrascu, K. Feng, and B. Engelhardt, "PG-TS: Improved Thompson sampling for logistic contextual bandits," in *NeurIPS*, 2018.

[146] R. Singh, J. Ling, and F. Doshi-Velez, "Structured variational autoencoders for the Beta-Bernoulli process," in *NeurIPS Workshop Adv. Approx. Bayes. Infer.*, 2017.

[147] R. M. Usman and M. A. ul Haq, "The marshall-olkin extended inverted kumaraswamy distribution: Theory and applications," *J. King Saud Univ. - Sci.*, 2020.