LEARNING GRAPH STRUCTURE FROM CONVOLUTIONAL MIXTURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning frameworks such as graph neural networks typically rely on a given, fixed graph to exploit relational inductive biases and thus effectively learn from network data. However, assuming the knowledge of said graphs may be untenable in practice, which motivates the problem of inferring graph structure from data. In this paper, 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 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, and perform link prediction or edge-weight regression tasks by adapting the loss function. Since layers directly operate on, combine, and refine graph objects (instead of node features), GDNs are inherently inductive and can generalize to larger-sized graphs after training. Algorithm unrolling offers an explicit handle on computational complexity; we trade-off training time in return for quick approximations to the inverse problem solution, obtained via a forward pass through the learnt model. We corroborate GDN's superior graph recovery performance using synthetic data in supervised settings, as well as its ability to generalize to graphs orders of magnitude larger that those seen in training. Using the Human Connectome Project-Young Adult neuroimaging dataset, we demonstrate the robustness and representation power of our model by inferring structural brain networks from functional connectivity.

1 INTRODUCTION

Inferring graphs from data to uncover latent complex information structure is a timely challenge for geometric deep learning (Bronstein et al., 2017) and graph signal processing (Ortega et al., 2018). But it is also an opportunity, since network topology inference advances (Dong et al., 2019; Mateos et al., 2019) could facilitate adoption of graph neural networks (GNNs) even when no input graph is available (Hamilton, 2020). 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 (Gama et al., 2020a). Early empirical evidence suggests that even when a graph is available, the structure could be further optimized for a downstream task (Kazi et al., 2020; Feizi et al., 2013), or else sparsified to boost computational efficiency and model interpretability (Spielman & Srivastava, 2011).

In this paper, we posit a convolutional model relating observed and latent undirected graphs and formulate the graph learning task as a supervised network inverse problem; see Section 2 for a formal problem statement. This fairly general model is motivated by various practical domains outlined in Section 3, such as identifying the structure of network diffusion processes (Segarra et al., 2017; Pasdeloup et al., 2018), as well as network deconvolution and denoising (Feizi et al., 2013). We propose a parameterized neural network model, termed graph deconvolution network (GDN), which we train in a supervised fashion to learn the distribution of latent graphs. The architecture is derived from the principle of algorithm unrolling used to learn fast approximate solutions to inverse problems (Gregor & LeCun, 2010; Sprechmann et al., 2015; Monga et al., 2021), an idea that is yet to be explored in the context of 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 un-

seen larger graphs, which has significant implications in domains like social networks and molecular biology (Yehudai et al., 2021). 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 process over the underlying anatomical pathways (Abdelnour et al., 2014; Liang & Wang, 2017), we show the applicability of GDNs to infer brain structural connectivity from functional networks obtained from the Human Connectome Project-Young Adult (HCP-YA) neuroimaging dataset.

Related work. Network topology inference has a long history in statistics (Dempster, 1972), with noteworthy contributions for probabilistic graphical model selection; see e.g. (Kolaczyk, 2009; Friedman et al., 2008; Drton & Maathuis, 2017). Recent advances were propelled by graph signal processing insights through the lens of signal representation (Dong et al., 2019; Mateos et al., 2019), exploiting models of network diffusion (Daneshmand et al., 2014), or else leveraging cardinal properties of network data such as smoothness (Dong et al., 2016; Kalofolias, 2016) and graph stationarity (Segarra et al., 2017; Pasdeloup et al., 2018). These works formulate (convex) optimization problems one has to solve for different graphs, and can lack robustness to signal model misspecifications. Scalability is an issue for the spectral-based network deconvolution approaches in (Segarra et al., 2017; Feizi et al., 2013), that require computationally-expensive eigendecompositions of the input graph for each problem instance. Moreover, none of these methods advocate a supervised learning paradigm to learn distributions over adjacency matrices. When it comes to this latter objective, deep generative models (Liao et al., 2019; Wang et al., 2018; Li et al., 2018) are typically trained in an unsupervised fashion, with the different goal of sampling from the learnt distribution. Most of these approaches learn node embeddings and are inherently transductive. Recently, so-termed latent graph inference approaches have been shown effective in obtaining better task-driven representations of relational data for machine learning (ML) applications (Wang et al., 2019; Kazi et al., 2020), or to learn interactions among coupled dynamical systems (Kipf et al., 2018).

Summary of contributions. We introduce GDNs, a supervised learning model capable of recovering latent graph structure from observations of its convolutional mixtures, i.e., related graphs containing spurious, indirect relationships. Our experiments on synthetic and real datasets demonstrate the effectiveness of GDNs for said task. They also showcase the model's versatility to incorporate domain-specific topology information about the sought graphs. 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 verify they exhibit minimal performance degradation even when tested on graphs $60\times$ 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. Results on the high-quality HCP-YA imaging dataset show that GDN performs 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 representation learning to integrate brain structure and function.

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 model

$$\boldsymbol{A}_O=\alpha_0\boldsymbol{I}+\alpha_1\boldsymbol{A}_L+\alpha_2\boldsymbol{A}_L^2+\ldots=\sum_{i=0}^{\infty}\alpha_i\boldsymbol{A}_L^i, \tag{1}$$

where $\boldsymbol{I}$ denotes the $N\times N$ identity matrix. The analytic graph mapping in (1) is a polynomial in $\boldsymbol{A}_L$ of possibly infinite degree, yet the Cayley-Hamilton theorem asserts it can always be equivalently reparameterized as a polynomial of degree smaller than N. Said matrix polynomials $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h}):=\sum_{k=0}^{K}h_k\boldsymbol{A}^k$, $K\leq N-1$, with coefficients $\boldsymbol{h}:=[h_0,\ldots,h_K]^\top\in\mathbb{R}^{K+1}$, are known

as shift-invariant graph convolutional filters; see e.g., (Ortega et al., 2018; Gama et al., 2020b). Going back to the model (1), 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 $\boldsymbol{A}_L$. 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; see also the application domains in Section 3.

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 a realization drawn from some distribution of sparse graphs, say, for e.g., random geometric graphs or structural brain networks from a homogeneous dataset. Then given independent training samples $\mathcal{T} := \{\boldsymbol{A}_O^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$ adhering to (1), our goal is to learn a judicious parametric mapping that predicts the graph adjacency matrix $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$ by minimizing a loss function

$$L(\boldsymbol{\Theta}) := \frac{1}{T} \sum_{i \in \mathcal{T}} \ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})). \tag{2}$$

The loss ℓ is chosen to accommodate the task at hand – link prediction that often suffices for the sake of graph structure identification, or, the more challenging edge-weight regression problem.

3 Motivating Application Domains

Before delving into the proposed learning-based approach to graph structure identification in Section 4, we outline graph inference tasks that can be cast as the network inverse problem (1).

Latent graph structure identification from diffused signals. Our initial focus here 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 an undirected latent graph $\mathcal{G}$ with sparse, symmetric adjacency matrix $\boldsymbol{A}_L \in \mathbb{R}_+^{N \times N}$. 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, if we can write $\boldsymbol{x} = \sum_{i=0}^{\infty} \alpha_i \boldsymbol{A}_L^i \boldsymbol{w} = \boldsymbol{H}(\boldsymbol{A}_L, \boldsymbol{h})\boldsymbol{w}$. This is a fairly common model for the generation of random network processes and subsumes heat diffusion (Barrat et al., 2008), consensus and the classic DeGroot model of opinion dynamics as special cases (DeGroot, 1974). The justification to say that $\boldsymbol{A}_L$ represents the structure of $\boldsymbol{x}$ is that we can think of the edges of $\mathcal{G}$ as direct (one-hop) relations between the elements of the signal. The diffusion described by $\boldsymbol{H}(\boldsymbol{A}_L, \boldsymbol{h})$ generates indirect relations. In this context, the latent graph 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 $\boldsymbol{x}$ (Segarra et al., 2017). Interestingly, from the model for $\boldsymbol{x}$ it follows that the signal covariance matrix $\boldsymbol{\Sigma}_x = \mathbb{E}\left[\boldsymbol{x}\boldsymbol{x}^\top\right] = \boldsymbol{H}^2$ is also a polynomial in $\boldsymbol{A}_L$ (we used $\boldsymbol{\Sigma}_w = \boldsymbol{I}$, and wrote $\boldsymbol{H} \leftarrow \boldsymbol{H}(\boldsymbol{A}_L, \boldsymbol{h})$ for notational simplicity). The connection with (1) should now be apparent with the identification $\boldsymbol{A}_O = \boldsymbol{\Sigma}_x$. 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 $\mathcal{T} := \{\hat{\boldsymbol{\Sigma}}_x^{(i)}, \boldsymbol{A}_L^{(i)}\}_{i=1}^T$, after estimating the covariance matrices from signals.

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 $\boldsymbol{A}_O$ of indirect relationships. The problem broadens the scope of e.g., signal deconvolution to networks and can be tackled by attempting to invert the mapping $\boldsymbol{A}_O = \boldsymbol{A}_L\left(\boldsymbol{I} - \boldsymbol{A}_L\right)^{-1} = \sum_{i=1}^{\infty} \boldsymbol{A}_L^i$. This solution proposed in (Feizi et al., 2013) assumes a polynomial relationship as in (1), but for the particular case of a single-pole, single-zero graph filter with very specific filter coefficients [cf. (1) with $\alpha_0 = 0$ and $\alpha_i = 1$, $i \geq 1$]. 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. Our idea in this paper is to adopt a more 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. Different from the latent graph learning problem outlined before, 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 other graph learning method upstream in the pipeline.

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 tractography algorithms applied to diffusion MRI (dMRI). 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 (Abdelnour et al., 2014; Honey et al., 2009). Traditional approaches go all the way from correlation studies (Greicius et al., 2008) to large-scale simulations of nonlinear cortical activity models (Honey et al., 2009). More aligned with the problem addressed here, recent studies have shown that linear diffusion dynamics can reasonably model the FC-SC coupling (Abdelnour et al., 2014; Surampudi et al., 2018). Using our notation, the findings in (Abdelnour et al., 2014) suggest that the covariance $\boldsymbol{A}_O = \boldsymbol{\Sigma}_x$ of the functional signals (i.e., the FC) is related to the the sparse SC graph $\boldsymbol{A}_L$ via the model in (1). Similarly, Liang & Wang (2017) contend the estimated FC can be represented as a weighted sum of the 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 (Damoiseaux & Greicius, 2009), a characteristic naturally captured by (1). These considerations motivate adopting the proposed graph learning method to infer SC patterns from fMRI signals; see the experimental results in Section 5.

4 Graph Deconvolution Network

Here we present the proposed GDN model, a parameterized neural network architecture that we train in a supervised fashion to recover latent graph structure via network deconvolution. In the sequel, we obtain conceptual iterations to tackle an optimization formulation of the network inverse problem (Section 4.1), unroll these iterations to arrive at the GDN model we train using graph data (Section 4.2), and describe architectural customizations to improve performance (Section 4.3).

4.1 Motivation via iterative optimization

Going back to the inverse problem of recovering a sparse adjacency matrix $\boldsymbol{A}_L$ from the mixture $\boldsymbol{A}_O$ in (1), if the graph convolutional filter $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})$ were known we could attempt to solve

$$\hat{\boldsymbol{A}}_L \in \arg\min_{\boldsymbol{A}\in\mathcal{A}} \left\{ \|\boldsymbol{A}\|_1 + \frac{\lambda}{2}\|\boldsymbol{A}_O - \boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})\|_F^2 \right\}, \tag{3}$$

where the regularization parameter $\lambda > 0$ trades off sparsity for reconstruction error. The convex set $\mathcal{A} := \{\boldsymbol{A} \in \mathbb{R}^{N\times N} \mid \text{diag}(\boldsymbol{A}) = \boldsymbol{0}, A_{ij} = A_{ij} \geq 0, \forall i,j \in \{1,\ldots,N\}\}$ encodes the admissibility constraints on the adjacency matrix of an undirected graph: hollow diagonal, symmetric, with non-negative edge weights. The ℓ_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 (1) and minimize the squared residual errors instead.

The composite cost in (3) is a weighted sum of a non-smooth function $\|\boldsymbol{A}\|_1$ and a continously differentiable function $g(\boldsymbol{A}) := \frac{1}{2}\|\boldsymbol{A}_O - \boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})\|_F^2$. Notice though that $g(\boldsymbol{A})$ is non-convex and its gradient is only locally Lipschitz continuous due to the graph filter $\boldsymbol{H}(\boldsymbol{A};\boldsymbol{h})$; except when $K = 1$, but the affine case is not interesting since $\boldsymbol{A}_O$ is just a scaled version of $\boldsymbol{A}_L$. By virtue of the polynomial structure of the non-convexity, provably convergent iterations can be derived using e.g., the Bregman proximal gradient method (Bolte et al., 2018) for judiciously chosen kernel generating distance; see also (Zhang & Hong, 2020). But our end goal here is not to solve (3) iteratively, recall we cannot even formulate the 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, it suffices to settle with conceptual proximal gradient (PG) iterations (k henceforth denote iterations, $\boldsymbol{A}[0] \in \mathcal{A}$)

$$\boldsymbol{A}[k+1] = \text{ReLU}(\boldsymbol{A}[k] - \tau\nabla g(\boldsymbol{A}[k]) - \tau\boldsymbol{1}\boldsymbol{1}^\top) \quad k = 0,1,2,\ldots, \tag{4}$$

where τ is a step-size parameter in which we have absorbed λ. Due to the non-negativity constraints in $\mathcal{A}$, the ℓ_1 norm's proximal operator takes the form of a shifted ReLU on the off-diagonal entries of its matrix argument. Also, the operator sets $\text{diag}(\boldsymbol{A}[k+1]) = \boldsymbol{0}$. In the next section, we unroll and truncate these iterations to arrive at the trainable GDN parametric model $\Phi(\boldsymbol{A}_O;\boldsymbol{\Theta})$.

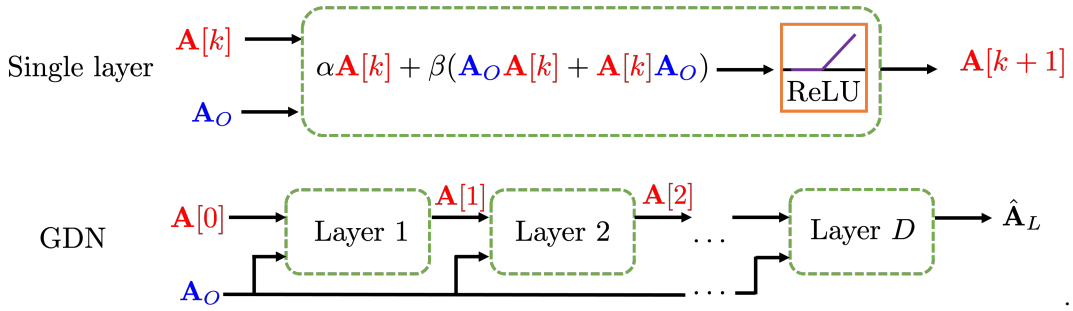

Figure 1: Schematic diagram of the GDN architecture obtained via algorithm unrolling.

4.2 Learning to infer graphs via algorithm unrolling

The idea of algorithm unrolling can be traced back to the seminal work of (Gregor & LeCun, 2010). 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., (Monga et al., 2021) for a recent tutorial treatment focused on signal and image processing. However, to the best of our knowledge this approach is yet to be explored for latent graph learning.

Building on the algorithm unrolling paradigm, our idea is to design a non-linear, parameterized, feed-forward architecture that can be trained to predict the latent graph $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$. 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 $\boldsymbol{H}(\boldsymbol{A}; \boldsymbol{h}) := \sum_{k=0}^{K} h_k \boldsymbol{A}^k$. In more detail, we start by simplifying $\nabla g(\boldsymbol{A})$ by dropping all higher-order terms in $\boldsymbol{A}$, namely

$$\begin{aligned}\nabla g(\boldsymbol{A}) &= -\sum_{k=1}^{K} h_k \sum_{r=0}^{k-1} \boldsymbol{A}^{k-r-1} \boldsymbol{A}_O \boldsymbol{A}^r + \frac{1}{2} \nabla_{\boldsymbol{A}} \operatorname{Tr}\left[\boldsymbol{H}^2(\boldsymbol{A}; \boldsymbol{h})\right] \\ &\approx -h_1 \boldsymbol{A}_O - h_2(\boldsymbol{A}_O \boldsymbol{A} + \boldsymbol{A} \boldsymbol{A}_O) + (2h_0 h_2 + h_1^2) \boldsymbol{A}. \end{aligned} \tag{5}$$

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 $\boldsymbol{I}$, which is inconsequential to adjacency matrix updates with null diagonal. An affine approximation will lead to more benign optimization landscapes when it comes to training the resulting GDN model. All in all, the PG iterations become

$$\boldsymbol{A}[k+1] = \operatorname{ReLU}(\alpha \boldsymbol{A}[k] + \beta(\boldsymbol{A}_O \boldsymbol{A}[k] + \boldsymbol{A}[k] \boldsymbol{A}_O) + \gamma \boldsymbol{A}_O - \tau \mathbf{1}\mathbf{1}^\top), \tag{6}$$

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 λ-dependent algorithm step-size, all of which are unknown in practice.

The GDN architecture is thus obtained by unrolling the algorithm (6) into a deep neural network; see Figure 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 neural networks is a characteristic of unrolled networks (Monga et al., 2021). In the next section, we will explore a few customizations to the 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 (2). Specific choices of the loss we adopt for edge-weight regression and link prediction tasks are described in Appendix A.2. 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; see Appendix A.2.

The iterative refinement principle of optimization algorithms naturally 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 non-linearities to refine it to an output $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta})$. Matrix $\boldsymbol{A}[0]$ is a hyperparameter we can select to incorporate prior information on the sought latent graph; see Section 4.3. The input graph $\boldsymbol{A}_O$ we wish to deconvolve is directly fed to all layers as in a residual neural network, and its role is also noteworthy. First, the constant matrix $\gamma \boldsymbol{A}_O - \tau \mathbf{1}\mathbf{1}^\top$ defines non-uniform soft thresholds to effectively sparsify the filter output per layer. Second, one can interpret $\alpha \boldsymbol{A} + \beta(\boldsymbol{A}_O \boldsymbol{A} + \boldsymbol{A}\boldsymbol{A}_O)$ as a first-order graph filter defined on $\boldsymbol{A}_O$, which is used to process $\boldsymbol{A}$ – here viewed as a (graph) signal with N features per node to invoke this graph signal processing insight. In its simplest rendition, the GDN architecture brings together elements of RNNs, ResNets and graph convolutional networks (GCNs) (Kipf & Welling, 2017; Gama et al., 2020b)

4.3 GDN ARCHITECTURE ADAPTATIONS

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

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 5.2, by taking the prior $\boldsymbol{A}[0]$ to be the sample mean of all latent (i.e., SC) graphs in the training set. 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 1. The ability to seamlessly incorporate prior information to the model is an attractive feature of GDNs, and differentiates it from other methods trying to solve the network inverse problem.

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 to the 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, providing richer learned representations.

Formally, the more expressive GDN architecture with MIMO filters is constructed as follows. At layer k of the neural network, we take a three-way tensor $\mathbf{A}_k \in \mathbb{R}_+^{C \times N \times N}$ and produce $\mathbf{A}_{k+1} \in \mathbb{R}_+^{C \times N \times N}$, where C is the common number of input and output channels. By defining multiplication between tensors $\mathbf{T}, \mathbf{B} \in \mathbb{R}^{C \times N \times N}$ as batched matrix multiplication: $[\mathbf{T}\mathbf{B}]_{j,:,:} := \mathbf{T}_{j,:,:}\mathbf{B}_{j,:,:}$, and tensor-vector addition and multiplication as $\mathbf{T} + \boldsymbol{v} := \mathbf{T} + [v_1 \mathbf{1}\mathbf{1}^\top; \ldots; v_C \mathbf{1}\mathbf{1}^\top]$ and $[\boldsymbol{v}\mathbf{T}]_{j,:,:} := v_j \mathbf{T}_{j,:,:}$ respectively for $\boldsymbol{v} \in \mathbb{R}^C$, all operations extend naturally.

Using these definitions, the j-th output slice of layer k is

$$[\mathbf{A}_{k+1}]_{j,:,:} = \text{ReLU}[\overline{\boldsymbol{\alpha}_{:,j}\mathbf{A}_k + \boldsymbol{\beta}_{:,j}(\mathbf{A}_O\mathbf{A}_k + \mathbf{A}_k\mathbf{A}_O) + \boldsymbol{\gamma}_{:,j}\mathbf{A}_O} - \tau_j \mathbf{1}\mathbf{1}^\top], \tag{7}$$

where $\overline{\cdots}$ represents the mean reduction over the filtered input channels and the parameters are $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^{C \times C}$, $\boldsymbol{\tau} \in \mathbb{R}_+^C$. We now take $\boldsymbol{\Theta} := \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\tau}\}$ for a total of $C \times (3C + 1)$ trainable parameters. For complete details see Appendix A.3.

We typically have a single prior matrix $\boldsymbol{A}[0]$ and are interested in predicting a single adjacency matrix $\hat{\boldsymbol{A}}_L$. Accordingly, we construct a new tensor prior $\mathbf{A}[0] := [\boldsymbol{A}[0], \ldots, \boldsymbol{A}[0]] \in \mathbb{R}^{C \times N \times N}$ and (arbitrarily) designate the first output channel as our prediction $\hat{\boldsymbol{A}}_L = \Phi(\boldsymbol{A}_O; \boldsymbol{\Theta}) = [\mathbf{A}_{k+1}]_{1,:,:}$. We denote this model as GDN-share-C, emphasizing the MIMO filter with C input and output channels and whose parameters are shared across layers.

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 (rather than them being shared across layers), 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 standard CNNs or GCNs). Accordingly, it opens up the architectural design space to broader exploration, e.g., wider layers in the beginning and skinnier layers at the top. Exploring this architectural space is beyond the scope of this paper and is left as future work. Layer k now has its own set of parameters $\mathbf{\Theta}^k = (\boldsymbol{\alpha}^k, \boldsymbol{\beta}^k, \boldsymbol{\gamma}^k, \boldsymbol{\tau}^k)$, where $\boldsymbol{\alpha}^k, \boldsymbol{\beta}^k, \boldsymbol{\gamma}^k \in \mathbb{R}^{C_{out}^k \times C_{in}^k}$ and $\boldsymbol{\tau}^k \in \mathbb{R}_+^{C_{out}^k}$, for a total of $C_{out}^k \times (3C_{in}^k + 1)$ trainable parameters. The tensor operations mapping inputs to outputs remains basically unchanged with respect to (7), except that the filter coefficients will depend on k.

In subsequent experiments, the GDN model for which intermediate layers $k \in \{2, \ldots, D-1\}$ have $C = C_{in}^k = C_{out}^k$, i.e., a flat architecture, is denoted as GDN-C.

5 EXPERIMENTS

We present experiments on link prediction and edge-weight regression tasks using synthetic data (Section 5.1), as well as the HCP-YA neuroimaging dataset (Section 5.2). In the link-prediction task, performance is evaluated using error $:= \frac{\text{incorrectly predicted edges}}{\text{total possible edges}} = 1 -$ prediction accuracy. For regression, we adopt the mean-squared-error (MSE) or mean-absolute-error (MAE) as figures of merit. 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. 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). We compare GDNs against several relevant baselines: Network Deconvolution (ND) which uses a spectral approach to directly invert a very specific convolutive mixture (Feizi et al., 2013); Spectral Templates (SpecTemp) that advocates a convex optimization approach to recover sparse graphs from noisy estimates of $\boldsymbol{A}_O$'s eigenvectors (Segarra et al., 2017); Graphical LASSO (GLASSO), a regularized MLE of the precision matrix for Gaussian graphical model selection (Friedman et al., 2008); and Hard Thresholding (Threshold) to assess how well a simple cutoff rule can perform. In all the results that follow, we use GDN(-share) models with $D = 8$ layers and train with the Adam optimizer (Kingma & Ba, 2015) with a learning rate of .01 and batch size of 200.

5.1 LATENT GRAPH STRUCTURE IDENTIFICATION 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 latent graphs somewhat comparable, across all models we let $N = 68$ (as constrained by the regions of interest in the adopted brain atlas), we force connectivity, and edge sparsity levels in the range $[0.5, 0.6]$ when feasible (these are also typical SC values). 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 in Section 3. 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. All models on synthetics are trained using $\boldsymbol{A}[0] = \mathbf{0}$, while models on the SCs take their prior as the edge-wise mean across all SCs in the training split.

Table 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-share) reduces error relative to the mean prior by $27.48 \pm 1.73\%$ ($23.02 \pm 1.73\%$) and MSE by $37.34 \pm 0.79\%$ ($23.23 \pm 0.48\%$). Both GDN architectures show the ability to learn such local patterns, with the extra representational power of GDNs (over GDN-share) providing an additional boost in performance. All models struggle on BA and ER with GDNs showing better performance even for these cases.

Table 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-8	$\mathbf{4.6}_{\pm 4.5\text{e-}1}$	$41.9_{\pm 1.1\text{e-}1}$	$\mathbf{27.5}_{\pm 1.0\text{e-}3}$	$\mathbf{8.9}_{\pm 1.7\text{e-}2}$
	GDN-share-8	$5.5_{\pm 2.4\text{e-}1}$	$\mathbf{40.8}_{\pm 1.0\text{e-}2}$	$27.6_{\pm 8.0\text{e-}4}$	$9.4_{\pm 2.1\text{e-}1}$
	GLASSO	$8.8_{\pm 6.5\text{e-}2}$	$43.2_{\pm 1.2\text{e-}2}$	$34.9_{\pm 9.8\text{e-}3}$	$20.0_{\pm 3.8\text{e-}2}$
	ND	$9.4_{\pm 3.1\text{e-}1}$	$43.9_{\pm 1.4\text{e-}2}$	$34.1_{\pm 8.2\text{e-}3}$	$21.3_{\pm 9.4\text{e-}2}$
	SpecTemp	$11.1_{\pm 3.2\text{e-}1}$	$44.4_{\pm 6.6\text{e-}2}$	$30.2_{\pm 1.8\text{e-}1}$	$30.0_{\pm 1.3\text{e-}1}$
	Threshold	$12.0_{\pm 1.8\text{e-}1}$	$42.9_{\pm 8.3\text{e-}1}$	$32.3_{\pm 1.0\text{e-}0}$	$21.7_{\pm 2.1\text{e-}1}$
MSE	GDN-8	$\mathbf{4.2\text{e-}2}_{\pm 4.3\text{e-}3}$	$2.3\text{e-}1_{\pm 2.2\text{e-}3}$	$\mathbf{1.8\text{e-}1}_{\pm 2.4\text{e-}3}$	$\mathbf{5.3\text{e-}3}_{\pm 6.7\text{e-}5}$
	GDN-share-8	$6.0\text{e-}2_{\pm 2.4\text{e-}1}$	$\mathbf{2.3\text{e-}1}_{\pm 2.1\text{e-}3}$	$2.7\text{e-}1_{\pm 1.6\text{e-}2}$	$6.5\text{e-}3_{\pm 4.0\text{e-}5}$
	GLASSO	$2.0\text{e-}1_{\pm 2.6\text{e-}3}$	$2.8\text{e-}1_{\pm 2.0\text{e-}2}$	$2.6\text{e-}1_{\pm 1.6\text{e-}2}$	$4.4\text{e-}2_{\pm 3.3\text{e-}5}$
	ND	$1.8\text{e-}1_{\pm 1.5\text{e-}3}$	$2.4\text{e-}1_{\pm 5.0\text{e-}4}$	$2.2\text{e-}1_{\pm 1.0\text{e-}3}$	$5.6\text{e-}2_{\pm 6.8\text{e-}5}$
	SpecTemp	$5.1\text{e-}2_{\pm 3.3\text{e-}5}$	$5.3\text{e-}1_{\pm 8.9\text{e-}5}$	$3.3\text{e-}1_{\pm 1.8\text{e-}5}$	$1.5\text{e-}1_{\pm 4.2\text{e-}3}$

Size generalization: Deploying on larger graphs. Unlike CNNs and GNNs, GDNs learn the parameters of graph convolutions for the processing of graphs, not the signals supported on them. GDNs are inductive and allows us to deploy the learnt model on larger graph size domains. To test the extent to which GDNs generalize when N grows, we train four GDN models (GDN-share-1, GDN-share-8, GDN-1, GDN-8) on RG graphs with size $N = 68$, and tested them on RG graphs of size $N = [68, 75, 100, 200, 500, 1000, 2000, 4000]$, with 200 graphs in each size domain. As before, all such graphs are connected and have sparsity in $[.5, .6]$. We sample a diffusion coefficient vector $\boldsymbol{h} \in \mathbb{R}^3$ on the unit sphere. As graph sizes increase, we require more samples in the estimation of the sample covariance to maintain a constant signal-to-noise ratio (SNR). To simplify the experiment and its interpretation, we instead remove the estimation step, 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 2 shows all GDN models effectively generalize to graphs orders of magnitude larger than they were trained on, giving up only modest levels of performance as size increases. For $N = 4000$, all but one of GDN models performed better than any baseline did on the original 68 node graphs. The GDN-share models maintain their top performance, with GDN-share-1 even *further reducing error* as the domain gets larger. This suggests that the 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 layer constraint in GDN-share models acts as regularization: we avoid over-fitting on a given size domain to better generalize to larger graphs.

5.2 HCP Brain Data

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 (Glasser et al., 2016). In this paper, we use the dMRI and resting state fMRI data from the HCP-YA dataset (Van Essen et al., 2013), 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 (Desikan et al., 2006) which consists of $N = 68$ cortical brain regions. The SC-FC coupling on this dataset is known to be the strongest in occipital lobe and vary with age, sex and cognitive health in other subnetworks (Gu et al., 2021). Under the consideration of the variability in SC-FC coupling across the brain regions, we further group the cortical regions into 4 larger regions called 'lobes': frontal, parietal, temporal, and occipital (the locations of these lobes in the brain are included in Fig. 4 in the supplementary material). We aim to predict SC, derived from dMRI, using FC, constructed using BOLD signals acquired with fMRI.

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

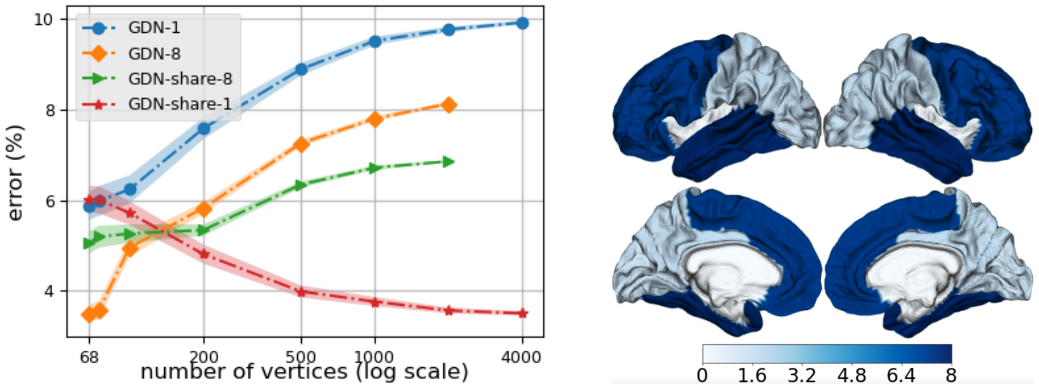

Figure 2: Left: After training on RG graphs of size $N = 68$, GDNs are capable of maintaining performance on RG graphs orders of magnitude larger. Missing data at $N = 4000$ corresponds to overwhelmed our memory resources. The simplest model GDN-share-1 improves performance with increasing graph size. Right: Reduction in MAE (%) over mean prior of SCs for different lobes. Most significant improvements are concentrated in temporal and frontal lobes.

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 $= 12.25\%$, MAE $= 0.0615$). At the lobe level, the prior achieves relatively higher accuracy in occipital (error = 98.71%, MAE $= 0.056$) and parietal (error = 93.63% MAE = 0.065) lobes as compared to temporal (error $= 88.76\%$, MAE $= 0.05$) and frontal (error $= 89.55\%$, MAE $= 0.059$) lobes; behavior which is unsurprising as SC in temporal and frontal lobes are affected by ageing and gender related variability in the dataset (Zimmermann et al., 2016). GDNs reduced MAE by 7.62%, 7.07%, 1.58%, and 1.29% in the temporal, frontal, parietal, and occipital networks respectively and 7.95% over the entire brain network, all relative to the already strong mean prior. The latter four reductions are projected onto their respective lobes in Fig. 2. Clearly, there was smaller room for improvement in performance over occipital and frontal lobes. We observed the most significant gains in performance over temporal and frontal lobes.

In summary, our pseudo-synthetic experiments in Section 5.1 show that SCs are amenable to learning with GCNs when the SC-FC relationship satisfies (1), a reasonable model given the findings of (Abdelnour et al., 2014). While the HCP-YA dataset is widely considered to be of high quality, neuroimaging data is still subject to many sources of noise and variation; dealing with such problems to extract connectomes is still an active research area (Zalesky et al., 2016). Additionally, SC-FC coupling is believed to vary widely across both the population and within the sub-regions of an individuals brain. When trained on the HCP-YA dataset, the GDN model exhibits robust performance over such regions with high variability in SC-FC coupling.

6 Conclusions

In this work we proposed the GDN, an inductive model capable of recovering latent graph structure from observations of its convolutional mixtures. By minimizing a task-dependent loss function, GDNs learn filters to refine initial estimates of the sought latent graphs layer by layer. The unrolled architecture can seamlessly integrate domain-specific prior information about the unknown graph distribution. Moreover, because GDNs: (i) are differentiable functions with respect to their parameters as well as their graph input; and (ii) they offer explicit control on complexity (leading to fast inference times); one can envision GDNs as valuable components in larger (even online) end-to-end graph representation learning systems. This way, while our focus here has been exclusively on network topology identification, the impact of GDNs can permeate to broader graph inference tasks.

Reproducibility statement. Code for running these experiments has been included in a zipped directory with the submission and contains instructions for configuring a system to run experiments presented above. When randomness is involved, as is the case when constructing the synthetic datasets, sampling white signals for diffusion, constructing a random split of the HCP-YA data, or initializing the parameters of our model before training, we use a consistent and clearly defined random seed, allowing others to reproduce the results presented. For the derivation of the model, we provide further details in A.4 to supplement those shown in the main paper body (Section 4.1). In A.5 we reference the readers to HCP website, where one can download the HCP-YA data, as well as references to the processing pipelines used to construct the FCs and SCs used in this paper. The processed brain data was too large to include with the code, but is available upon request.

REFERENCES

F. Abdelnour, H. U. Voss, and A. Raj. Network diffusion accurately models the relationship between structural and functional brain connectivity networks. *Neuroimage*, 90:335–347, Apr. 2014.

Alain Barrat, Marc Barthelemy, and Alessandr Vespignani. *Dynamical Processes on Complex Networks*. Cambridge University Press, New York, NY, 2008.

Jérôme Bolte, Shoham Sabach, Marc Teboulle, and Yakov Vaisbourd. First order methods beyond convexity and Lipschitz gradient continuity with applications to quadratic inverse problems. *SIAM J. Optim.*, 28(3):2131–2151, 2018.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

Jessica S Damoiseaux and Michael D Greicius. Greater than the sum of its parts: A review of studies combining structural connectivity and resting-state functional connectivity. *Brain Struct. Func.*, 213(6):525–533, 2009.

Hadi Daneshmand, Manuel Gomez-Rodriguez, Le Song, and Bernhard Schoelkopf. Estimating diffusion network structures: Recovery conditions, sample complexity & soft-thresholding algorithm. In *International Conference on Machine Learning*, pp. 793–801. PMLR, 2014.

Morris H. DeGroot. Reaching a consensus. *J Am Stat Assoc.*, 69:118–121, 1974.

Arthur P Dempster. Covariance selection. *Biometrics*, pp. 157–175, 1972.

Rahul S Desikan, Florent Ségonne, Bruce Fischl, Brian T Quinn, Bradford C Dickerson, Deborah Blacker, Randy L Buckner, Anders M Dale, R Paul Maguire, Bradley T Hyman, et al. An automated labeling system for subdividing the human cerebral cortex on mri scans into gyral based regions of interest. *Neuroimage*, 31(3):968–980, 2006.

Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Trans. Signal Process.*, 64(23):6160–6173, 2016.

Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Process. Mag.*, 36(3):44–63, 2019.

Mathias Drton and Marloes H Maathuis. Structure learning in graphical modeling. *Annu. Rev. Stat. Appl*, 4:365–393, 2017.

Soheil Feizi, Daniel Marbach, Muriel Medard, and Manolis Kellis. Network deconvolution as a general method to distinguish direct dependencies in networks. *Nat. Biotechnol*, 31(8):726–733, 2013.

J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Trans. Signal Process.*, 68:5680–5695, 2020a.

Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Process. Mag.*, 37(6):128–138, 2020b.

Matthew F Glasser, Stephen M Smith, Daniel S Marcus, Jesper LR Andersson, Edward J Auerbach, Timothy EJ Behrens, Timothy S Coalson, Michael P Harms, Mark Jenkinson, Steen Moeller, et al. The human connectome project's neuroimaging approach. *Nature neuroscience*, 19(9): 1175–1187, 2016.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, pp. 399–406, 2010.

Michael Greicius, Kaustubh Supekar, Vinod Menon, and Robert Dougherty. Resting-state functional connectivity reflects structural connectivity in the default mode network. *Cereb Cortex*, 19:72–8, 12 2008.

Zijin Gu, Keith Wakefield Jamison, Mert Rory Sabuncu, and Amy Kuceyeski. Heritability and interindividual variability of regional structure-function coupling. *Nat. Commun*, 12(1):1–12, 2021.

William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

C. Honey, O. Sporns, L. Cammoun, X. Gigandet, J. Thiran, R. Meuli, and P. Hagmann. Predicting human resting-state functional connectivity from structural connectivity. *Proc. Natl. Acad. Sci. U.S.A*, 106:2035–2040, 2009.

Vassilis Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pp. 920–929, 2016.

Anees Kazi, Luca Cosmo, Nassir Navab, and Michael M. Bronstein. Differentiable graph module (DGM) graph convolutional networks. *CoRR*, abs/2002.04999, 2020. URL `https://arxiv.org/abs/2002.04999`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, volume 80, pp. 2688–2697, 10–15 Jul 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

E. D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer, New York, NY, 2009.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018. URL `https://arxiv.org/abs/1803.03324`.

Hualou Liang and Hongbin Wang. Structure-function network mapping and its assessment via persistent homology. *PLOS Comput. Biol*, 13(e1005325):1–19, Jan. 2017.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, 2019.

Gonzalo Mateos, Santiago Segarra, Antonio G. Marques, and Alejandro Ribeiro. Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Process. Mag.*, 36 (3):16–43, May 2019.

Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Process. Mag.*, 38(2):18–44, 2021.

Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proc. IEEE*, 106(5):808–828, 2018.

Bastien Pasdeloup, Vincent Gripon, Grégoire Mercier, Dominique Pastor, and Michael G. Rabbat. Characterization and inference of graph diffusion processes from observations of stationary signals. *IEEE Trans. Signal Inf. Process. Netw.*, 4(3):481–496, 2018.

Santiago Segarra, Antonio G. Marques, Gonzalo Mateos, and Alejandro Ribeiro. Network topology inference from spectral templates. *IEEE Trans. Signal Inf. Process. Netw.*, 3(3):467–483, August 2017.

Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, December 2011.

P. Sprechmann, A. M. Bronstein, and G. Sapiro. Learning efficient sparse and low rank models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(9):1821–1833, 2015.

Sriniwas Govinda Surampudi, Shruti Naik, Raju Bapi Surampudi, Viktor K Jirsa, Avinash Sharma, and Dipanjan Roy. Multiple kernel learning model for relating structural and functional connectivity in the brain. *Scientific reports*, 8(1):1–14, 2018.

David C Van Essen, Stephen M Smith, Deanna M Barch, Timothy EJ Behrens, Essa Yacoub, Kamil Ugurbil, Wu-Minn HCP Consortium, et al. The WU-Minn Human Connectome Project: An overview. *Neuroimage*, 80:62–79, 2013.

Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph representation learning with generative adversarial nets. In *AAAI Conference on Artificial Intelligence*, pp. 2508–2515, 2018.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.*, 38(5), October 2019.

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pp. 11975–11986. PMLR, 2021.

Andrew Zalesky, Alex Fornito, Luca Cocchi, Leonardo L Gollo, Martijn P van den Heuvel, and Michael Breakspear. Connectome sensitivity or specificity: which is more important? *Neuroimage*, 142:407–420, 2016.

Junyu Zhang and Mingyi Hong. First-order algorithms without Lipschitz gradient: A sequential local optimization approach. *CoRR*, abs/2010.03194, 2020. URL `https://arxiv.org/abs/2010.03194`.

Joelle Zimmermann, Petra Ritter, Kelly Shen, Simon Rothmeier, Michael Schirner, and Anthony R McIntosh. Structural architecture supports functional organization in the human aging brain at a regionwise and network level. *Hum. Brain Mapp*, 37(7):2645–2661, 2016.

A APPENDIX

A.1 INCORPORATING PRIOR INFORMATION

In section 4.3 we introduce the concept of using prior information in the training of GDNs. We do so by encoding information we may have about the unknown latent graph $\boldsymbol{A}_L$ into $\boldsymbol{A}_O$, the starting matrix which GDNs iteratively refine. If the A_L's are repeated instances of a graph with fixed nodes, as is the case with the SCs with respect to the 68 fixed brain regions, a simple strategy to incorporate prior information about some edge $\mathbf{A}_{L_{i,j}}$, now viewed as a random variable, would be $\boldsymbol{A}_{0_{i,j}} \leftarrow \mathbb{E}[\mathbf{A}_{L_{i,j}}]$. But there is more that can be done. We also can estimate the variance $\mathrm{Var}(\mathbf{A}_{L_{i,j}})$, and use it during the training of a GDN, for example taking $\mathbf{A}_{0_{i,j}} \leftarrow \mathcal{N}(\mathbb{E}[\mathbf{A}_{L_{i,j}}], \mathrm{Var}(\mathbf{A}_{L_{i,j}}))$, or even simpler, using a resampling technique and taking $\mathbf{A}_{0_{i,j}}$ to be a random sample in the training set. By doing so, we force the GCN to take into account the distribution and uncertainty in the data, possibly leading to richer learned representations and better performance. It also would act as a form of regularization, not allowing the model to converge on the naive solution of outputting the simple expectation prior, a likely local minimum in training space.

A.2 TRAINING

Training the encoder will be performed using stochastic (mini-batch) gradient descent to minimize a task-dependant loss function $\mathcal{L}(\boldsymbol{\Theta})$, defined either as (i) the edgewise squared error between the predicted graph and the true graph for regression tasks, or (ii) a hinge loss with parameter $\gamma \geq 0$, both averaged over a training set $\{\boldsymbol{A}_O, ..., \boldsymbol{A}_O\}$:

$$\ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta}))_{hinge} := \sum_{i,j} \begin{cases} (\Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})_{i,j} - \gamma)^+ & \boldsymbol{A}_{L_{i,j}} == 0 \\ (-\Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})_{i,j} + 1 - \gamma)^+ & \boldsymbol{A}_{L_{i,j}} > 0 \end{cases} \tag{8}$$

$$\ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta}))_{mse} := \frac{1}{2}\|\boldsymbol{A}_L^{(i)} - \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})\|_2^2 \tag{9}$$

$$\ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta}))_{mae} := \|\boldsymbol{A}_L - \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta}) = \|_1 \tag{10}$$

$$L(\boldsymbol{\Theta}) := \frac{1}{T}\sum_{i \in \mathcal{T}} \ell(\boldsymbol{A}_L^{(i)}, \Phi(\boldsymbol{A}_O^{(i)}; \boldsymbol{\Theta})), \qquad \ell \in \{hinge, mse, mae\} \tag{11}$$

Link Prediction with GDNs and Unbiased Estimates of Generalization In the edge-weight regression task, GDNs only use their validation data to determine when training has converged. When performing link-prediction, GDNs have an additional use for this data: to choose the cutoff threshold $t \in \mathbb{R}_+$, determining which raw outputs (which are continuous) should be considered positive edge predictions, *at the end of training*.

We use the training set to learn the parameters (via gradient descent) *and* to tune t. During the validation step, when then use this train-set-tuned-t on the validation data, giving an estimate of generalization error. This is then used for early-stopping, determining the best model learned after training, etc. We do not use the validation data to tune t during training. Only after training has completed, do we tune t with validation data. We train a handful of models this way, and the model which produces the best validation score (in this case lowest error) is the tested with the validation-tuned-t, thus providing an unbiased estimate of generalization.

A.3 MIMO MODEL ARCHITECTURE

MIMO Filters

MIMO (Multi-Input Multi-Output) filters are defined as follows. At iteration k we take $\mathbf{A}_k \in \mathbb{R}_+^{C \times N \times N}$ and produce $\mathbf{A}_{k+1} \in \mathbb{R}_+^{C \times N \times N}$.

Output channel j will use $\boldsymbol{\alpha}_{:,j}, \boldsymbol{\beta}_{:,j}, \boldsymbol{\gamma}_{:,j} \in \mathbb{R}^{C_{in}}$ and $\tau_j \in \mathbb{R}_+$ to filter all input channels $i \in \{1, \cdots, C^{in}\}$. This produces tensor of stacked filtered input channels $\in \mathbb{R}^{C^{in},N,N}$. After setting the diagonal elements of all matrix slices in this tensor to 0, then perform a mean reduction edgewise

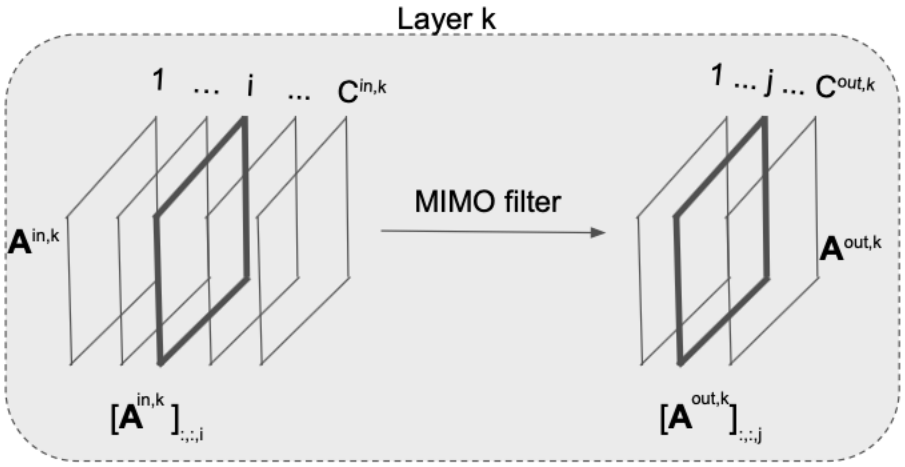

Figure 3: MIMO Filter: Layer k takes a tensor $\mathbf{A}_k \in \mathbb{R}^{C^{\text{in,k}} \times N \times N}$ and outputs a tensor $\mathbf{A}_{k+1} \in \mathbb{R}^{C^{\text{out,k}} \times N \times N}$. The i-th slice $[\mathbf{A}_k]_{i,:,:}$ is called the i-th input channel and $[\mathbf{A}_{k+1}]_{j,:,:}$ is called the j-th output channel.

(over the 0th dimension) of this tensor, producing a single matrix $\in \mathbb{R}^{N \times N}$. We then apply two pointwise/elementwise operations on this matrix: first subtract $\boldsymbol{\tau}_j^k$ (this would be the 'bias' term in CNNs) and second apply a nonlinearity ($ReLU$). This produces an $\in \mathbb{R}^{N \times N}$ activation stored in the j-th output channel Doing so for all output channels $j \in \{1, \cdots, C^{out}\}$, produces a tensor $\mathbf{A}^{out} \in \mathbb{R}^{C^{out} \times N \times N}$.

From (7), the j-th output slice of layer k is

$$[\mathbf{A}_{k+1}]_{j,:,:} = \text{ReLU}[\overline{\boldsymbol{\alpha}_{:,j}\mathbf{A}_k + \boldsymbol{\beta}_{:,j}(\mathbf{A}_O\mathbf{A}_k + \mathbf{A}_k\mathbf{A}_O) + \boldsymbol{\gamma}_{:,j}\mathbf{A}_O} - \tau_j \mathbf{1}\mathbf{1}^\top], \tag{12}$$

where $\overline{\cdots}$ represents the mean reduction over the filtered input channels, and parameters are $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^{C \times C}$, and $\boldsymbol{\tau} \in \mathbb{R}_+^C$. If we consider MIMO filters in GDNs, as opposed to GDN-share, we must add the index k to each parameter, denoting which filter it belongs to.

Just like a CNN, *each* output channel has

- $C^{\text{in,k}}$ *different* sets of parameters, $\boldsymbol{\alpha}_{:,j}^k, \boldsymbol{\beta}_{:,j}^k, \boldsymbol{\gamma}_{:,j}^k \in \mathbb{R}^{C^{\text{in,k}}}$, to process each input channel separately (and then aggregates them)
- a *single* bias term τ_j^k applied to the aggregated filtered input channels

Layer Output Normalization and Zero-ing Diagonal
The steps not shown in the main text are the normalization steps, a practical issue, and the setting of the diagonal elements to be 0, a projection onto the set of allowable adjacency matrices. Define $\overline{\mathbf{U}}_{j,:,:} = \overline{\boldsymbol{\alpha}_{:,j}^k\mathbf{A}_k + \boldsymbol{\beta}_{:,j}^k(\mathbf{A}_O\mathbf{A}_k + \mathbf{A}_k\mathbf{A}_O) + \boldsymbol{\gamma}_{:,j}^k\mathbf{A}_O} \in \mathbb{R}^{N \times N}$ as the j-th slice in the intermediate tensor $\overline{\mathbf{U}} \in \mathbb{R}^{C^{out,k} \times N \times N}$ used in the filtering of $\mathbf{A}_k$. Normalization is performed by dividing each 2-D matrix slice of $\overline{\mathbf{U}}$ by the maximum magnitude element in that respective slice: $\overline{\mathbf{U}}_{\cdot,:,:} / \max \left|\overline{\mathbf{U}}_{\cdot,:,:}\right|$.
Multiple metrics were tried on the denominator, including the value of 99th percentile of all values in $\overline{\mathbf{U}}_{\cdot,:,:}$, the frobenius norm $\|\overline{\mathbf{U}}_{\cdot,:,:}\|_F$, among others. None seemed to work as well as the maximum magnitude element, which has the additional advantage of guaranteeing entries to be in $[0, 1]$ (after ReLU), which matches nicely with (i) adjacency matrices of undirected graphs and (ii) makes it easy to normalize edge weights of a dataset of adjacencies: simply scale them to $[0, 1]$.

Other obvious options for normalization are:

1. Slice normalization before aggregation step
    - This was not chosen for the following reason. If slices are normalized before aggregation, the model loses the ability to directly scale the relative contributions of filtered

input channels into an output channel. For example, suppose output channel j wants to decrease the relative amount of processed input channel i. Doing so is not easy. It cannot simply decrease the scale of the relevant parameters $\boldsymbol{\alpha}^k_{j,i}, \boldsymbol{\beta}^k_{j,i}$ as could be done otherwise. If it attempts to do this, the normalization will bring the maximum magnitude element to $+/-1$, the same as all of the other filtered input channels.

- By agreggating and then normalizing, the model can simply scale $\boldsymbol{\alpha}^k_{j,i}, \boldsymbol{\beta}^k_{j,i}$ appropriately to achieve this.
- In practice there is not a large difference in training.
- This also is a but of a larger break from how typical CNNs operate, breaking the analogy a bit.

2. Use batch metric in denominator

- Instead of only using information within a single slice, it is concievable that a more useful strategy would be to use a metric based on batch information, as is done in 'batch norm' in typical CNNs/
- This was not explored due to time limitations.

The full procedure to produce $[\mathbf{A}_{k+1}]_{j,:,:}$ is as follows:

$$
\begin{aligned}
\overline{\mathbf{U}}_{j,:,:} &= \overline{\boldsymbol{\alpha}^k_{:,j}\mathbf{A}_k + \boldsymbol{\beta}^k_{:,j}(\mathbf{A}_O\mathbf{A}_k + \mathbf{A}_k\mathbf{A}_O) + \boldsymbol{\gamma}^k_{j,:})\mathbf{A}_O} \\
\overline{\mathbf{U}}_{j,:,:} &= \overline{\mathbf{U}}_{j,:,:} \odot (\boldsymbol{J} - \boldsymbol{I}) && \text{force diagonal elements to 0} \\
\overline{\mathbf{U}}_{j,:,:} &= \overline{\mathbf{U}}_{j,:,:} / \max(|\overline{\mathbf{U}}_{j,:,:}|) && \text{normalize entries in each slice to be in } [-1, 1] \\
[\mathbf{A}_{k+1}]_{j,:,:} &= \text{ReLU}(\overline{\mathbf{U}}_{j,:,:} - \tau^j_l)
\end{aligned}
$$

To clarify how $\overline{\mathbf{B}}$ works, let $\mathbf{B} \in \mathbb{R}^{C^{out,k} \times C^{in,k} \times N \times N}$. Then $\overline{\mathbf{B}} := \frac{1}{C^{\text{in,k}}} \sum_{i=1}^{C^{\text{in,k}}} \mathbf{B}[:, i, :, :] \in \mathbb{R}^{C^{out,k} \times N \times N}$ is a mean reduction of the 1st dimension. In our case this is a mean reduction over the filtered input channels.

By normalizing in this way, we guarantee the intermediate matrix $\overline{\mathbf{U}}_{j,:,:}$ has entries in $[-1, 1]$. This plays two important roles.

1. Training Stability

- Consider what could happen if no normalization is used. Suppose the entries of the $\overline{\mathbf{U}}_{j,:,:}$ are orders of magnitude larger than entries of $\overline{\mathbf{U}}_{l,:,:}$. This can cause the model to push $\tau^k_j >> \tau^k_l$. This can cause training instabilities and/or lack of convergence.

2. Interpretability of τ

- Our normalization allows us the interpret the learned values $\boldsymbol{\tau}^k \in \mathbb{R}_+^{C^{\text{out,k}}}$ on a similar scale. They all must be in $[0, 1]$ because (i) anything above 1 will make the output all 0, and (ii) we constrain it to be non-negative. In fact we can now plot all τ values (from all layers) against one another, and using the same scaling ($[0, 1]$) interpret if a particuler τ is promoting a lot of sparsity in the output (τ close to 1) or not (τ close to 0) by its magnitude.

A.4 GRADIENT USED IN PROXIMAL GRADIENT ITERATIONS

Calculation of $\nabla g(\boldsymbol{A})$
Let $\boldsymbol{A}, \boldsymbol{B} \in S^n$, $\boldsymbol{H}(\boldsymbol{A}) := \sum_{k=0}^{K} a_k \boldsymbol{A}^k$. Then

$$
\begin{aligned}
&\nabla_{\boldsymbol{A}} \frac{1}{2} \|\boldsymbol{B} - \boldsymbol{H}(\boldsymbol{A})\|_F^2 \\
&= \frac{1}{2} \nabla_{\boldsymbol{A}} \operatorname{Tr}\left(\boldsymbol{B}^2 - \boldsymbol{B}\boldsymbol{H}(\boldsymbol{A}) - \boldsymbol{H}(\boldsymbol{A})\boldsymbol{B} + \boldsymbol{H}^2(\boldsymbol{A})\right) \\
&= -\nabla_{\boldsymbol{A}} \operatorname{Tr}\left(\boldsymbol{B}\boldsymbol{H}(\boldsymbol{A})\right) + \frac{1}{2} \nabla_{\boldsymbol{A}} \operatorname{Tr} \boldsymbol{H}^2(\boldsymbol{A}) && \operatorname{Tr}(\boldsymbol{C}\boldsymbol{D}) = \operatorname{Tr}(\boldsymbol{D}\boldsymbol{C}) \text{ if } \boldsymbol{C}, \boldsymbol{D} \in S^n \\
&= -\sum_{k=1}^{K} a_k \sum_{r=0}^{k-1} \boldsymbol{A}^{k-r-1} \boldsymbol{B} \boldsymbol{A}^r + \frac{1}{2} \nabla_{\boldsymbol{A}} \operatorname{Tr} \boldsymbol{H}^2(\boldsymbol{A}) \\
&= -\sum_{k=1}^{K} a_k \sum_{r=0}^{k-1} \boldsymbol{A}^{k-r-1} \boldsymbol{B} \boldsymbol{A}^r + \frac{1}{2} \boldsymbol{H}_1(\boldsymbol{A}) && \boldsymbol{H}_1 \in \mathcal{P}_{2K-1}(\boldsymbol{A}) \\
&= -[a_1 \boldsymbol{B} + a_2(\boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}) + a_3(\boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{A}\boldsymbol{B}\boldsymbol{A} + \boldsymbol{B}\boldsymbol{A}^2) + \dots\,] + \frac{1}{2} \boldsymbol{H}_1(\boldsymbol{A}) \quad \square
\end{aligned}
$$

Note that in the context of the GDN Model, $\boldsymbol{A}$ is an intermediate output of the model, denoted $\boldsymbol{A}_k$ in the main text, and $\boldsymbol{B}$ corresponds to $\boldsymbol{A}_O$. Powers of such a quantity can produce numerical instabilities, and thus unstable training. We can drop the higher order terms for a first order approximation of ∇g. Doing so gives us the final form of $\nabla_{\boldsymbol{A}} g_{\boldsymbol{h}}$:

$$
\nabla g(\boldsymbol{A}) \approx -h_1 \boldsymbol{A}_O - h_2(\boldsymbol{A}_O \boldsymbol{A} + \boldsymbol{A}\boldsymbol{A}_O) + (2h_0 h_2 + h_1^2)\boldsymbol{A}
$$

A.5 NOTES ON EXPERIMENTAL SETUP

Synthetic Graphs For the experiments presented in 1, the synthetic graphs of size $N = 68$ are drawn with the following parameters

- Random Geometrics (RG): d= 2, $r = 0.56$.
- Erdős-rényi (ER): $p = .56$.
- Barabási-albert (BA): $m = 15$

When sampling graphs to construct the datasets, we reject any samples which are not connected or have sparsity outside of a given range. For RG and ER, that range is $[0.5, 0.6]$, while in BA the range is $[0.3, 0.4]$. This is an attempt to make the RG and ER graphs similar to the SC graphs, which have an average sparsity of .56, and all SCs are in sparsity range $[0.5, 0.6]$. Due to the sampling procedure of BA, it is not possible to produce graph in this sparsity range, so we lowered the range sightly. We thus take SCs to be an sbm-like domain and avoid a repetitive experiment.

Note that the sparsity ranges define the performance of the most naive of predictors: all ones/zeros. In the RG/BA/SC, an all ones predictor achieves an average error of $44\% = 1-$(average graph sparsity). In the BAs, a naive all zeros predictor achieves $35\% = 1-$(average graph sparsity). This is useful to keep in mind when interpreting the results in Table 1.

Pseudo-Synthetics The Pseudo-Synthetic datasets are those in which we diffuse synthetic signals over SCs from the HCP-YA dataset. This is an ideal setting to test the GDN models: we have weighted graphs to perform edge-regression on (the others are unweighted), while having $\boldsymbol{A}_O$'s that are true to our modeling assumptions. Note that SCs have a strong community-like structure, corresponding dominantly to the Left and Right hemispheres as well as subnetworks which have a high degree of connection, e.g. the Occipital Lobe which has 0.96 sparsity - almost fully connected - while the full brain network has sparsity of 0.56.

Error and MSE of the Baselines in Table 1 The edge weights returned by the baselines can be very small/large in magnitude and perform poorly when used directly in the regression task. We thus also provide a scaling parameter, tuned during the hyperparameter search, which provides approximately

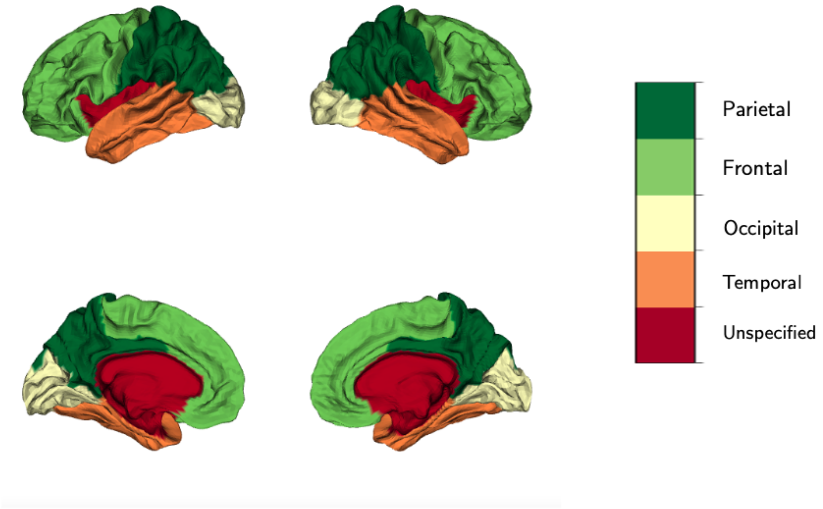

Figure 4: The four lobes in the brains cortex.

an order of magnitude improvement in MSE in GLASSO and halved the MSE in Spectral Templates and Network Deconvolution. In link-prediction, we also tune a hard thresholding parameter on top of each method to clean up noisy outputs from the baseline, only if it improved their performance (it does). For complete details on the baseline implementation, see A.6.

Size Generalization Something to note is that we do **not** tune the threshold (found during training on the small $N = 68$ graphs) on the larger graphs. We go straight from training to testing on the larger domains. Tuning the threshold using a validation set (of larger graphs) would represent an easier problem. The model at no point, or in any way, is introduced to the data in the larger size domains for any form of training/tuning.

We decide to use the covariance matrix in this experiment, as opposed to the sample covariance matrix, as our $\boldsymbol{A}_O$'s. This is for the simple reason that it would be difficult to control the snr with respect to generalization error and would be secondary to the main thrust of the experiment. When run with number of signals proportional to graph size, we see quite a similar trend, but due to time constraints, these results are not presented herein, but is apt for follow up work.

HCP Data HCP-YA provides up to 4 resting state fMRI scanning sessions for each subject, each lasting 15 minutes. We use the fMRI data which has been processed by the minimal processing pipeline (Van Essen et al., 2013). For every subject, after pre-processing the time series data to be zero mean, we concatenate all available time-series together and compute the sample covariance matrix (which is what as used are $\boldsymbol{A}_O$ in the brain data experiment 5.2).

Due to expected homogeneity in SCs across a healthy population, information about the SC in the test set could be leveraged from the SCs in the training set. In plainer terms, the average SC in the training set is an effective predictor of SCs in the test set. In our experiments, we take random split of the 1063 SCs into $913/50/100$ training/validation/test sets, and report how much we improve upon this predictor.

The raw data is available for download on the HCP website, https://www.humanconnectome.org/study/hcp-young-adult/overview.

A.6 BASELINES

We were as generous as possible in the comparison of GDNs to baseline models. All baselines were optimized to minimize generalization error, which is what is presented in 1. Many baseline methods aim to predict sparse graphs on their own, yet many fail to bring edge values fully to zero. We thus provide a threshold, tuned for generalization error using a validation set, on top of each method only if it improved the performance of the method in link-prediction. The edge weights returned by the baselines can be very small/large in magnitude and perform poorly when used directly in the the regression task. We thus also provide a scaling parameter, tuned during the hyperparameter search, which provides approximately an order of magnitude improvement in MSE in GLASSO and halved the MSE in Spectral Templates and Network Deconvolution.

A.6.1 HARD THRESHOLDING

The hard thresholding model consists of a single parameter: τ

$$\hat{A} = (x > \tau)$$

where x is either Cov or Corr. We tried both choices in our experiments, and which ever performed better is listed in the Results.

A.6.2 GLASSO

We found taking the absolute value of GLASSO improved its performance, and so we include that in the model. We first take the absolute value of the GLASSO output, and then hard thresholded it using τ.

$$\hat{\boldsymbol{A}} = (|GLASSO_{norm}(x)| > \tau)$$

There are two hyperparameters to tune for link-prediction (and a third, the scaling parameter described above, in regression). α := nonnegative regularization parameter in GLASSO
τ := nonnegative threshold used to map values into binray decisions on the existance of an edge
We used the sklearn GLASSO implementation found here: `https://scikit-learn.org/stable/modules/generated/sklearn.covariance.graphical_lasso.html`
It is important to note that we do **not** use the typical cross-validation procedure seen with GLASSO. Typically, GLASSO is used in unsupervised applications with only one graph being predicted from s observations. In our application, we are predicting many graphs, *each* with s observations.
Thus the typical procedure of choosing α using the log-likelihood (the non-regularized part of the GLASSO objective) *over splits of the observed signals, not splits of the training set*, results in worse performance (and a different α for each graph). This is not surprising: exposing the training procedure to labeled data allows it to optimize for generalization. We are judging the models on their ability to generalize to unseen graphs, and thus the typical procedure would provide an unfair advantage to our model. While we tried both sample covariance and sample correlation matrices as $\boldsymbol{A}_O$, we found that we needed the normalization that the sample correlation provides, along with an additional scaling by the maximum magnitude eigenvalue, in order to achieve numerical stability. GLASSO can take a good amount of time to run, and so we limited the validaiton and test set sizes to an even 100/100 split. With only 2 to 3 hyperparameters to tune, we found this was sufficient (no significant differences between validation and test performance in all runs, and when testing on larger graph sizes, no difference in generalization performance).

A.6.3 NETWORK DECONVOLUTION (ND)

The authors present Network Deconvolution as "a general method for inferring direct effects from an observed correlation matrix containing both direct and indirect effects." (Feizi et al., 2013). Network Deconvolution follows three steps: Linear Scaling (ensure all eigenvalues of $\boldsymbol{A}_O$ are $\in [-1, 1]$), Decomposition (eigen-decomposition of scaled $\boldsymbol{A}_O$ into $\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$), and Deconvolution (applying $f(\lambda) = \frac{\lambda}{1+\lambda}$ to all eigenvalues). We then construct our prediction: $\hat{\boldsymbol{A}}_L \leftarrow \boldsymbol{V}\text{diag}(f(\boldsymbol{\lambda}))\boldsymbol{V}^{-1}$. In the paper, they recommend a pearson correlation matrix be constructed, which we followed. We applied an extra hard thresholding on the output, tuned for best generalization error, to further increase performance. For each result shown 500 graphs were used in the hyperparameter search and 500 were used for testing.

A.6.4 SPECTRAL TEMPLATES

The Spectral Templates method consists of solving two convex optimizations problems multiple times. We first perform a binary search on $\epsilon \in \mathbb{R}_+$ over the interval $[0, 2]$ to find ϵ_{min}, which is the smallest value which allows a feasible solution to optimization problem 13.

$$\begin{aligned} \boldsymbol{S}^*(\epsilon) := \underset{\{\boldsymbol{S}, \bar{\boldsymbol{\lambda}}\}}{\text{minimize}} \quad & \|\boldsymbol{S}\|_1 \\ \text{subject to} \quad & \hat{\boldsymbol{S}} = \boldsymbol{V}\text{diag}(\bar{\boldsymbol{\lambda}})\boldsymbol{V}^T = \sum_{i=1}^{N} \bar{\lambda}_i \boldsymbol{v}_i \boldsymbol{v}_i^T \\ & \|\boldsymbol{S} - \hat{\boldsymbol{S}}\|_2^2 < \epsilon \\ & \boldsymbol{S}\mathbf{1} \geq \mathbf{1} \\ & \boldsymbol{S} \in \mathcal{S} \end{aligned} \tag{13}$$

Where $\boldsymbol{V}$ is the orthogonal eigenvector matrix $\boldsymbol{V} := [\boldsymbol{v}_1, ..., \boldsymbol{v}_N]$ and $\boldsymbol{\lambda}$ the corresponding eigenvalues of $\boldsymbol{S} : \boldsymbol{S} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^T$. $\mathcal{S}$ denotes the convex set of allowable adjacency matrices representing undirected graphs with no self-loops: hollow (zero diagonal), symmetric, with non-negative entries. Note that $\bar{\boldsymbol{\lambda}} \in \mathbb{R}^N$ is an optimization variable, while $\boldsymbol{\lambda} \in \mathbb{R}^N$ are the true eigenvalues of $\boldsymbol{S}$ (which are unknown and not used in the optimization).

With ϵ_{min} in hand, we now run an iteratively re-weighted L1 norm problem with the aim of further pushing small edge weights to 0. Defining $\boldsymbol{W}_t := \frac{\gamma \mathbf{1}}{|\boldsymbol{S}^*_{t-1}| + \delta \mathbf{1}} \in \mathbb{R}_+^{N \times N}$ where $\gamma, \delta \in \mathbb{R}_+$ are appropriately chosen positive constants and $\boldsymbol{S}_0^* := \boldsymbol{S}^*(\epsilon_{min})$, we solve a sequence $t = 1, \ldots, T$ weighted L1 norm minimization problems.

$$\begin{aligned} \boldsymbol{S}_t^* := \underset{\{\boldsymbol{S}, \bar{\boldsymbol{\lambda}}\}}{\text{minimize}} \quad & \|\boldsymbol{W}_t \odot \boldsymbol{S}\|_1 = \sum_{i=1}^{N} \bar{\lambda}_i \boldsymbol{v}_i \boldsymbol{v}_i^T \\ \text{subject to} \quad & \hat{\boldsymbol{S}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T \\ & \|\boldsymbol{S} - \hat{\boldsymbol{S}}\|_2^2 < \epsilon_{min} \\ & \boldsymbol{S}\mathbf{1} \geq \mathbf{1} \\ & \boldsymbol{S} \in \mathcal{S} \end{aligned} \tag{14}$$

If any of these iteratively reweighted problems is infeasible $\boldsymbol{S}_I^*$ is returned, where $I \in \{0, 1, \ldots, T\}$ is the last successfully solved iteratively reweighted solution.

Finally, a threshold τ is chosen with a validation set to map the output to binary decision over edges.

$$\boldsymbol{S}_{SpecTemp} = (\boldsymbol{S}_I^* > \tau)$$

We solve these optimization problems using MOSEK solvers in CVXPY. Solving such convex optimization problems can very computationally expensive/slow, and because there are only two hyperparameters, 100 graphs were used in the validation set, and 100 in the test set.

A.7 SOURCE CODE AND CONFIGURATION

Code has been made available in a zipped directory with the submission. Refer to the README for instructions on configuring a system to run the code. We rely on PyTorch heavily and use Conda to make our system as hardware-independant as possible. GPUs were used to train the larger GDNs, which are available to use for free on Google Colab. Development was done on Mac and Linux (Ubuntu) systems, and not tested on Windows machines. The processed brain data was too large to include with the code, but is available upon request.