Appendix A Algorithms of model-based graph learning

A.1 An ADMM approach for solving model-based graph learning

Besides PDS in Algorithm 1, we also consider an alternating direction method of multiplier (ADMM) approach in Algorithm 3 for solving the model-based graph learning problem in Eq.(4) as a baseline. The algorithm is specifically derived from Algorithm 1 in [21], where we use $\mathbf{v}$ to denote the dual variable. Compared to PDS, ADMM has another relaxation factor $\{\lambda^{(t)}\}_{t\in\mathbb{N}}$. From grid search, $\lambda^{(t)}$ is normally taken a value in $[1.5, 2]$ for all t. Also, the update steps are not entirely separated in ADMM. As shown in Figure 3a, it converges more slowly than PDS. Therefore, we choose PDS in the unrolling framework.

Algorithm 3 ADMM

Input: $\mathbf{y}$, γ, $\{\lambda^{(t)}\}_{t\in\mathbb{N}}$, α, β and $\mathcal{D}$.

Initialisation: $\mathbf{w}^{(0)} = 0$, $\mathbf{v}^{(0)} = 0$
while $|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}| > \epsilon$ **do**
  $\mathbf{r}_1^{(t)} = \mathbf{w}^{(t)} - \gamma(2\beta\mathbf{w}^{(t)} + 2\mathbf{y} + \mathcal{D}^T\mathbf{v}^{(t)})$
  $\mathbf{p}_1^{(t)} = \text{prox}_{\gamma,\Omega_1}(\mathbf{r}_1^{(t)}) = \max\{\mathbf{0}, \mathbf{r}_1^{(t)}\}$
  $\mathbf{r}_2^{(t)} = \mathbf{v}^{(t)} + \gamma\mathcal{D}(2\mathbf{r}_1^{(t)} - \mathbf{w}^{(t)})$
  $\mathbf{p}_2^{(t)} = \text{prox}_{\gamma,\Omega_2}(\mathbf{r}_2^{(t)})$, where $\left(\text{prox}_{\gamma,\Omega_2}(\mathbf{r}_2)\right)_i = \frac{r_{2,i} - \sqrt{r_{2,i}^2 + 4\alpha\gamma}}{2}$
  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda^{(t)}(\mathbf{p}_1^{(t)} - \mathbf{w}^{(t)})$
  $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \lambda^{(t)}(\mathbf{p}_2^{(t)} - \mathbf{v}^{(t)})$
end while
return $\mathbf{w}^{(t)} = \hat{\mathbf{w}}$

A.2 Derivations for PDS in Algorithm 1

The derivations for Step 5 and Step 6 in Algorithm 1 are as follows.

Lemma 1. *The proximal projection step for primal variable (i.e. Step 5) in Algorithm 1 is* $prox_{\gamma\Omega_1}(r_1^{(t)}) = \max\{0, r_1^{(t)}\}$.

Proof. From Algorithm 6 in [21], the proximal projection step for primal variable (i.e. Step 5 in Algorithm 1) is $p_1^{(t)} = \text{prox}_{\gamma\Omega_1}(r_1^{(t)})$. $\Omega_1(r_1^{(t)}) = 0$ if $r_1^{(t)} \geq 0$ else $\Omega_1(r_1^{(t)}) = \infty$. We have $\gamma\Omega_1(r_1^{(t)}) = \Omega_1(r_1^{(t)})$ and hence $\text{prox}_{\gamma\Omega_1}(r_1^{(t)}) = \text{prox}_{\Omega_1}(r_1^{(t)}) = \max\{0, r_1^{(t)}\}$. □

Lemma 2. *The proximal projection step for dual variable (i.e. Step 6) in Algorithm 1 is* $prox_{\gamma\Omega_2^*}(r_2^{(t)}) = \frac{r_2^{(t)} - \sqrt{r_2^{(t)2} + 4\alpha\gamma}}{2}$.

Proof. From Algorithm 6 in [21], the proximal projection step for dual variable for our case is $p_2^{(t)} = \text{prox}_{\gamma\Omega_2^*}(r_2^{(t)})$, where Ω_2^* is the conjugate of Ω_2. Denote $h(r_2^{(t)}) = \gamma\Omega_2(\frac{r_2^{(t)}}{\gamma})$. We have $h(r_2^{(t)}) = \gamma(-\alpha 1^T \log(\frac{r_2^{(t)}}{\gamma})) = \gamma(-\alpha 1^T \log(r_2^{(t)}) + \alpha 1^T \log\gamma) = \gamma(\Omega_2(r_2^{(t)}) + \text{constant})$. By the affine rule of proximal operators, $\text{prox}_h(r_2^{(t)}) = \text{prox}_{\gamma\Omega_2}(r_2^{(t)})$. Meanwhile, by Theorem 6.9 of [3], $\text{prox}_{\gamma\Omega_2}(r_2^{(t)}) = \frac{r_2^{(t)} + \sqrt{r_2^{(t)2} + 4\alpha\gamma}}{2}$. By Theorem 6.12 of [3], if $h(r_2^{(t)}) = \gamma\Omega_2(\frac{r_2^{(t)}}{\gamma})$, then $\text{prox}_h(r_2^{(t)}) = \gamma\text{prox}_{\frac{1}{\gamma}\Omega_2}(\frac{r_2^{(t)}}{\gamma})$. We thus have $\text{prox}_h(r_2^{(t)}) = \text{prox}_{\gamma\Omega_2}(r_2^{(t)}) = \gamma\text{prox}_{\frac{1}{\gamma}\Omega_2}(\frac{r_2^{(t)}}{\gamma})$. Plugging it back to the Moreau decomposition leads to $\text{prox}_{\gamma\Omega_2^*}(r_2^{(t)}) = r_2^{(t)} - \gamma\text{prox}_{\frac{1}{\gamma}\Omega_2}(\frac{r_2^{(t)}}{\gamma}) = r_2^{(t)} - \text{prox}_{\gamma\Omega_2}(r_2^{(t)}) = r_2^{(t)} - \frac{r_2^{(t)} + \sqrt{r_2^{(t)2} + 4\alpha\gamma}}{2} = \frac{r_2^{(t)} - \sqrt{r_2^{(t)2} + 4\alpha\gamma}}{2}$. □

Appendix B Experimental details

B.1 Model Configurations

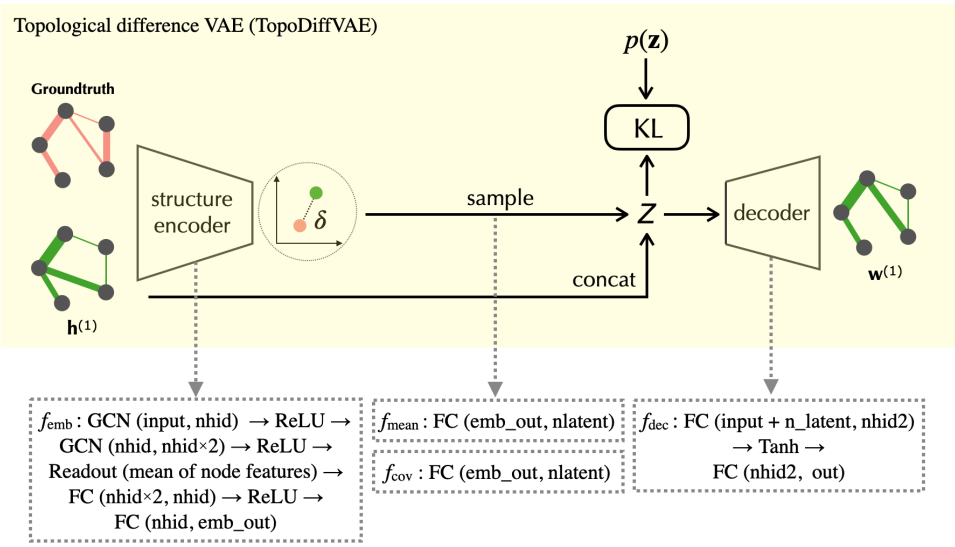


Figure 6: Model Configurations of TopoDiffVAE in L2G.

For the experiments in Section 5, the configurations for the proposed L2G are illustrated in Figure 6. The notations for neural network modules are defined as

- GCN(a, b): graph convolution with the learnable matrix $\mathbf{H}$ in a dimension of $a \times b$ in Eq 6.
- Readout: a pooling layer in GCN that takes the average of node embeddings.
- FC(c, d): a fully connected layer with the input dimension c and the output dimension d.
- ReLU, Tanh: the activation functions.

The number of hidden neurons (nhid and nhid2), the dimension of the graph embedding (emb_out) and that of the latent code $\mathbf{z}$ (nlatent) are tuned for different types and sizes of graphs. For learning graphs with a size of $m = 20$ in Section 5, we set nhid $= 64$, nhid2 $= 256$ and $|\mathbf{z}| = 16$. The input and output size are fixed to be $m \times (m-1)/2$.

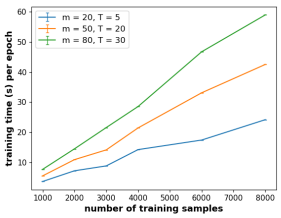
We notice that the original proximal operator in Step 5 of PDS (Algorithm 1) corresponds to the non-negative constraints of edge weights, which is not a complicated structural prior but a must-have. Meanwhile, replacing it with TopoDiffVAE at every unrolling layer might lead to a overly complex model that is hard to train and might suffer from overfitting. Therefore, for the experiments we only use TopoDiffVAE in the last layer of L2G while keeping the handcrafted prior, i.e. Step 5 of Algorithm 1, for the first $T-1$ layer. The experimental results in Section 5 show the effectiveness and efficiency of L2G.

B.2 Training settings

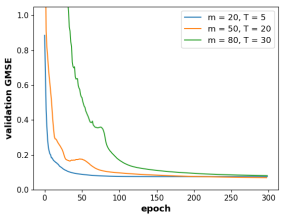
For the experiments in Section 5, we train L2G and Unrolling with the Adam optimiser [18]. The learning rate is exponentially-decayed from an initial value of 10^{-2} at a rate of 0.95. The number of training samples is changed with the graph size m. For $m = 20$, we use 4000 for training and 1000 for validation. Figure 7c shows how many training samples are needed (at different graph size). All the metrics reported in the paper are computed from 64 test samples that are additionally generated. The batch size is set to 32. We run all the experiments on a remote machine of Amazon EC2 G4dn-2xlarge instance[7]. Figure 7b shows training time per epoch with different number of samples.

[7] https://aws.amazon.com/ec2/instance-types/g4/

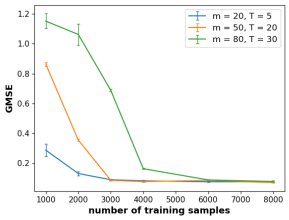
Once trained, the computational cost of applying L2G to learn a graph from new observations is negligible. The source code is included in the supplementary file.



(a) training time per epoch in seconds v.s. number of training samples



(b) validation GMSE v.s. number of epochs



(c) test GMSE v.s. number of training samples

Figure 7: Training L2G for different size of graphs m with different number of unrolls T

Appendix C Additional results

C.1 Ability to recover binary graphs

Table 3: A table of GMSE and Structural Metrics in recovering binary graphs

metric / model	groundtruth	Deep-graph	L2G
Scale-free (BA):			
GMSE	-	$0.802 \pm .003$	$0.060 \pm .002$
AUC	-	$0.565 \pm .101$	$0.999 \pm .000$
KS test score	96.15%	68.32%	96.15%
Community (SBM):			
GMSE	-	$0.901 \pm .016$	$0.080 \pm .010$
AUC	-	$0.701 \pm .088$	$0.992 \pm .001$
community score	$0.472 \pm .002$	$0.311 \pm .006$	$0.479 \pm .005$
Small-world (WS):			
GMSE	-	$0.873 \pm .010$	$0.056 \pm .004$
AUC	-	$0.519 \pm .023$	$0.997 \pm .000$
average shortest path	$2.23 \pm .028$	$1.111 \pm .008$	$2.225 \pm .041$

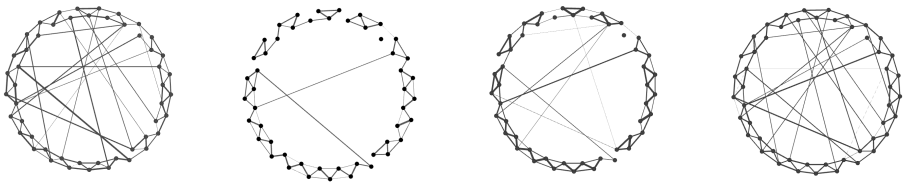
Deep-graph [4] is a state-of-the-art learning-based method, but its output is binary graph structure that is different from our setting. In order to have a fair comparison, we redo the experiments with another synthetic dataset with binary groundtruth graphs. The results are shown in Table 3 as an addition to the results of recovering weighted graph in the main body (see Table 1 and Table 2). Both results lead to the same conclusion that Deep-graph [3] has a less satisfactory performance compared to the proposed L2G.

C.2 Ability to recover graph topologies with specific structures.

Continued from Section 5, we test the accuracy of recovering groundtruth graphs of different structural properties. We deliberately increase the rewiring probability in generating WS graphs in Figure 8 and add inter-community edges on SBM graphs in Figure 10 to increase the learning difficulties in both cases. PDS may effectively detect the k-NN structure behind the construction of WS and the communities (block diagonal structure of the adjacency matrix) in SBM, but largely ignored the new edges from the procedures above. By contrast, L2G can precisely predict the additional rewired edges and inter-community edges.

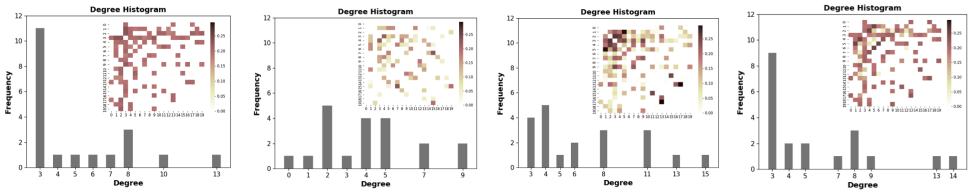
C.3 Scaling to large graphs.

In Figure 3c of Section 5, we show that L2G performs better in learning graphs with $m = 100$ nodes. The main challenge of scaling to large graphs (i.e. millions of nodes) lies in the memory required



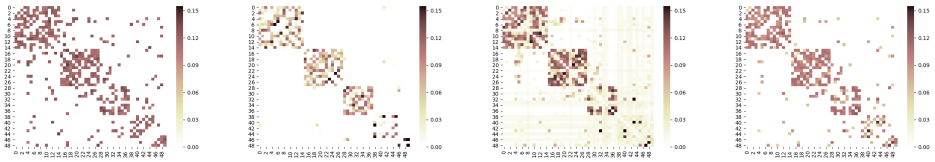
(a) groundtruth (b) PDS (c) Unrolling (d) L2G

Figure 8: The ability to recover small-world network



(a) groundtruth (b) PDS (c) Unrolling (d) L2G

Figure 9: The ability to recover scale-free networks



(a) groundtruth (b) PDS (c) Unrolling (d) L2G

Figure 10: The ability to recover community-structured networks

for large training samples including both graphs and data on nodes. Fortunately, we show that the proposed methods can be pretrained and then transferred to real-world data (see the real-world applications in Section 5). We can use a similar trick to pretrain an Unrolling model and then transferred to evaluate a graph with more nodes. This is because the learnable parameters of the Unrolling model ($\boldsymbol{\theta}_{\text{unrolling}}$) are not dependent on graph size. More extensive tests on this is one of the main directions for future work.

C.4 Stability in learning brain functional connectivity.

Following the stability experiment in [4], we also report the mean Spearman correlation in Table 4. Specifically, we apply L2G to infer the brain functional connectivity from BOLD time series collected from 35 subjects in the autistic group and control group as described in Section 5. In the whole data set, there are a total of 539 and 573 subjects in the two groups, respectively. At inference time, we apply a pretrained L2G model on a random collection of 35 subjects from each group. The inference procedure is repeated for 50 times, which yields 50 brain functional connectivity graphs for each group. For every pair of 50 graphs in each group, we apply the Spearman correlation to the rank of inferred edge weights. In Table 4, we report the mean (with 95% confidence interval) of the Spearman correlation between pairs of 50 graphs in each group, where the graphs are inferred using different models. The

Table 4: Mean Spearman correlation between estimated brain graphs

method	Control Group	Autistic Group
Graph Lasso [2]	0.21 $\pm$ .003	0.21 $\pm$ .003
DeepGraph [4]	0.23 $\pm$ .004	0.17 $\pm$ .003
DeepGraph+P[4]	0.19 $\pm$ .004	0.14 $\pm$ .004
L2G (proposed)	**0.76** $\pm$ **.056**	**0.34** $\pm$ **.061**

results show that L2G outperforms the state-of-the-art models in producing stable predictions on brain connectivity.