Workshop at the 7th Symposium on Advances in Approximate Bayesian Inference (non-archival), 2025 1–13

Learning Graph Structure for GNNs via Marginal Likelihood

Anita Yang ANITAY@IS.S.U-TOKYO.AC.JP
Thomas Möllenhoff THOMAS.MOELLENHOFF@RIKEN.JP
Ken-ichi Kawarabayashi K_KENITI@NII.AC.JP
Mohammad Emtiyaz Khan EMTIYAZ.KHAN@RIKEN.JP

Abstract

Learning graph structures for Graph Neural Networks (GNNs) can improve their performance, but it is challenging to design a good structure-learning objective that can provide sufficient flexibility while avoiding overfitting. Here, we propose to use marginal likelihood to learn the graph structure. We adopt the standard Laplace's method to approximate the marginal likelihood and optimize it using the Straight Through Estimator. This simple scheme yields good results on standard benchmarks for graph structure learning. The method can learn generic graph structures and eliminates the effort needed to design the objective. Our work makes it easier to learn graph structures for GNNs.

1. Introduction

Graph neural network (GNN) (Gori et al., 2005; Scarselli et al., 2008; Kipf and Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018) can exploit the graph structure of the data, but the structure may be noisy and incorrect. Even when it is not, it may not be suitable for the chosen GNN architecture. For example, graphs with long-distance interactions or are heterophilic (dissimilar nodes tend to be connected) tend to perform poorly on most GNNs (Zhu et al., 2020; Alon and Yahav, 2021; Topping et al., 2022; NT and Maehara, 2019; Oono and Suzuki, 2019; Li et al., 2018). A common remedy is to change the architecture to suit the data (Li et al., 2019; Chen et al., 2020a; Xu et al., 2018; Liu et al., 2020; Pei et al., 2020; Zhu et al., 2020) but this still cannot resolve noisy and incorrect graph structures. Learning the graph structure, on the other hand, can fix noisy structures and align them with the chosen GNN architecture.

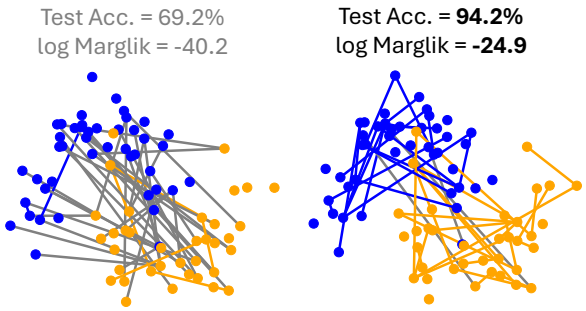


Figure 1: Using the original structure *(left)* results in worse performance than the learned graph structure *(right)*. The marginal likelihood correlates with the test accuracy.

Learning good graph structures for GNN is a promising avenue of research but not without its challenges: the exponential search space for graph means that a good regularization method is necessary to make learning tractable and ensure generalizability of the learned graph. Previous works rely on assumptions about the target graph structure (e.g. sparsity, connectivity, etc.). These methods are difficult to implement (e.g. designing custom loss functions (Yang et al., 2019; Chen et al., 2020b; Jin et al., 2020)) and prone to be overly restrictive (e.g. from defining graph structure prior (Zhang et al., 2019; Wang et al., 2021; Ma et al., 2019; Pantelis et al., 2020; Hasanzadeh et al., 2020)).

Here, we propose to use marginal likelihood as a simple solution to learn the graph structure. Motivated by Bayesian model selection (MacKay, 1995) where a higher marginal likelihood indicates a more likely model, we apply the same principle for graph structures: a better graph structure corresponds to larger marginal likelihood on the GNN parameters (as exemplified in Figure 1). As a proxy for generalization, marginal likelihood implicitly regularizes the structure learning process without requiring assumptions about the target structure, allowing generic graph structures to be learned. In practice, we use Laplace approximation of the marginal likelihood and Straight Through Estimator (STE) (Bengio et al., 2013) to learn discrete graph structures. We refer to our method as: *Laplace Approximation-based Graph (LAG)* structure learning. We showed that the GCN implementation of LAG outperforms other GCN-based graph structure learning (GSL) methods (Chen et al., 2020b; Franceschi et al., 2019) and matches the performance of Transformer-based GSL method (Wu et al., 2022) on standard benchmark datasets.

2. Graph Structure Learning for GNNs

Consider a semi-supervised node classification problem with N nodes. For each node, we observe a vector $\mathbf{x}_i \in \mathbb{R}^d$ and the corresponding one-hot label vector $\mathbf{y}_i \in \{0,1\}^C$, where C is the number of classes. The relational data is represented as a $N \times N$ binary (0 or 1) adjacency matrix $\mathbf{A}_0$, with entry 1 indicating an edge. Collectively, we can denote the data by $\mathcal{D} = (\mathbf{X}, \mathbf{Y}, \mathbf{A}_0)$, where $\mathbf{X} \in \mathbb{R}^{N\times d}$ has $\mathbf{x}_i^\top$ as rows and $\mathbf{Y} \in \{0,1\}^{N\times C}$ has $\mathbf{y}_i^\top$ as rows.

2.1. Graph Convolutional Networks (GCN)

Let us consider a simple Graph Convolutional Network (GCN) (Kipf and Welling, 2017). GCN embed the graph structure into the latent node representation, denoted by $\mathbf{Z}^{(l)} \in \mathbb{R}^{N\times d_l}$ for the l'th layer with width d_l. This is done by using the following propagation through multiple layers, starting with $\mathbf{Z}^{(0)} = \mathbf{X}$, and then following: $\mathbf{Z}^{(l)} = \sigma\big(\hat{\mathbf{A}}_0\mathbf{Z}^{(l-1)}\mathbf{\Theta}^{(l)}\big)$. Here, $\sigma(\cdot)$ is a non-linear activation function (such as ReLU). The weights of each layer are denoted by $\mathbf{\Theta}^{(l)} \in \mathbb{R}^{d_{l-1}\times d_l}$. Collectively, we denote them by $\boldsymbol{\theta}$. The matrix $\hat{\mathbf{A}}_0$ is the 'augmented' adjacency matrix: $\hat{\mathbf{A}}_0 = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}_0 + \mathbf{I}_N)\tilde{\mathbf{D}}^{-\frac{1}{2}}$, where $\tilde{\mathbf{D}}$ is the degree matrix of $\mathbf{A}_0 + \mathbf{I}_N$. The prediction $\hat{\mathbf{Y}}$ is obtained by applying the softmax function in the last layer.

The parameters $\boldsymbol{\theta}$ are learned in a standard way by minimizing the empirical loss

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \sum_{i=1}^{N_{\text{train}}} \ell\left(\mathbf{Y}, \widehat{\mathbf{Y}}(\boldsymbol{\theta})\right) + r(\theta), \tag{1}$$

where $\ell(\cdot,\cdot)$ is often chosen to be the cross-entropy loss between the true labels and its predictions. The regularizer $r(\theta)$ can either be explicit or implicit.

2.2. Learning Graph Structures

The setup above assumes that the graph structure $\mathbf{A}_0$ is given and reflects useful relationships among the nodes. However, this may not always be true because the data is often noisy and incorrect. Even when it is not, learning graph structure might be useful as the chosen architecture (e.g. GCN) may not be suitable for the graph. Learning graph structure then would only keep and add relevant relationships, and avoid introducing new noise.

Throughout, we will denote the unknown graph structure by $\mathbf{A}$, which is initialized with $\mathbf{A}_0$ (either given or constructed via k-Nearest Neighbor algorithm), but will subsequently be learned during training. This problem where $\mathbf{A}$ and $\boldsymbol{\theta}$ are to be learned together is known as the Graph Structure Learning problem (GSL) (an overview is in Appendix A).

3. Method

We introduce the *Laplace Approximation-based Graph (LAG)* structure learning algorithm, a GNN-agnostic bilevel optimization method, that maximizes the marginal likelihood. The key components of LAG are Laplace approximation that makes marginal likelihood calculation tractable and Straight Through Estimator (STE) to learn discrete graph structures.

3.1. Laplace Approximation of Marginal Likelihood

The marginal likelihood is intractable to compute for neural networks. We chose to use Laplace approximation MacKay (1995), which Immer et al. (2021) has shown as effective for online hyperparameter learning. We define the loss function for learning $\mathbf{A}$ as the negative log Laplace approximated marginal likelihood of $\boldsymbol{\theta}$, and denote it as $\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A})$. Minimizing $\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A})$ is equivalent to maximizing the marginal likelihood. Using the analytical result of the Laplace approximated marginal likelihood, the loss is given as:

$$\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A}) = \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) + \frac{1}{2}\log(|\mathbf{H}_\theta|) + C, \tag{2}$$

where $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ is the empirical risk (Eq. 1), $\mathbf{H}_\theta$ is the Hessian defined as $\nabla^2_\theta \mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$, and C is a constant. Calculation of $\mathbf{H}_\theta$ is generally not feasible, we have opted to use the diagonal Generalized Gauss-Newton (GGN) approximation (detailed in Appendix B).

3.2. Learn Discrete Graphs via STE

The graph structure $\mathbf{A}$ cannot be learned directly using gradient descent on $\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A})$ (Eq. 2) because $\mathbf{A}$ is discrete and non-differentiable. We address this by using the Straight Through Estimator (STE) (Bengio et al., 2013). STE uses the binary adjacency matrix $\mathbf{A}$ in the forward pass while accumulating gradient on the continuous adjacency matrix $\mathbf{A}'$ in the backward pass. We use a threshold of 0.5 on $\mathbf{A}'$ to obtain $\mathbf{A}$.

3.3. Laplace Approximation-based Graph (LAG)

Our proposed method LAG (outlined in Algorithm 1) is a GNN agnostic method that uses bilevel optimization approach to learn the model parameters $\boldsymbol{\theta}$ and the graph structure $\mathbf{A}$ jointly. The inner objective learns $\boldsymbol{\theta}$ to minimize the empirical loss (Eq. 1), and the outer objective learns $\mathbf{A}$ to maximize the marginal likelihood.

Algorithm 1: Laplace Approximation-based Graph (LAG) Structure Learning

Input: training data $\mathcal{D} = (\mathbf{X}, \mathbf{Y}, \mathbf{A}_0)$, burn-in epoch B, cut-off epoch C, update freq F, graph structure learning rate γ, steps K

Output: learned binary adjacency matrix $\mathbf{A}$, learned GNN parameters $\boldsymbol{\theta}$

$\mathbf{A}, \mathbf{A}' \leftarrow \mathbf{A}_0$
for *each epoch* **do**
  $\theta \leftarrow \texttt{trainGNNEpoch}(\mathbf{A}, \mathcal{D})$ {minimize Eq. 1}
  if *in between* B *and* C*, every* F **then**
    for K *steps* **do**
      compute $\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A})$ with $\mathbf{H}_\theta^{\text{GGN}}$ {from Eq. 2}
      $g_{\mathbf{A}'} \leftarrow \nabla_{\mathbf{A}'} \mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}, \mathbf{A})$;
      $\mathbf{A}' \leftarrow \mathbf{A}' + \frac{\gamma}{||g_{\mathbf{A}'}||_2} g_{\mathbf{A}'}$;
      $\mathbf{A} \leftarrow \texttt{STE}(\mathbf{A}')$ {discretize adjacency matrix}
    end
  end
end
return $\mathbf{A}, \boldsymbol{\theta}$

The GNN parameters $\boldsymbol{\theta}$ is first trained over B burn-in epochs using an initial graph $\mathbf{A}_0$ (either given or constructed e.g. kNN graph). For every F epochs after the burn-in period and before the cut-off epoch C: at epoch t, the model parameters $\boldsymbol{\theta}$ and the graph structure $\mathbf{A}$ are updated as follows:

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathbf{A}_t), \qquad \mathbf{A}_{t+1} = \arg\min_{\mathbf{A}} \mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}_t, \mathbf{A}), \tag{3}$$

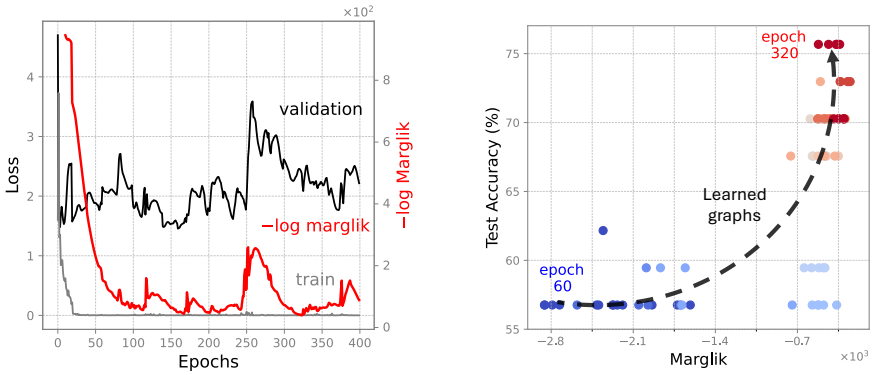
where $\mathcal{L}(\boldsymbol{\theta}; \mathbf{A}_t)$ is defined in Equation (1) and $\mathcal{L}_{\text{Marglik}}(\boldsymbol{\theta}_t, \mathbf{A})$ is defined in Equation (2). $\mathbf{A}_{t+1}$ is obtained over K steps with constant step-length (i.e. normalized learning rate γ).

4. Experiments

We demonstrate the effectiveness of LAG in solving the GNN graph structure learning problem here. We implemented LAG with GCN and refer to it as LAG-GCN. Using LAG-GCN, we show that: (1) Marginal likelihood is an effective objective for learning graph structures (Section 4.1), (2) LAG-GCN outperforms other GCN-based GSL methods while matching the performance of a transformer-based GSL method (Section 4.2).

Datasets. Our experiments use standard benchmark datasets: Planetoid (Cora and Citeseer) and WebKB (Cornell, Texas and Wisconsin) (detailed in Appendix C).

Setup. We consider both cases when the initial graph $\mathbf{A}_0$ is and is not given. In the later case, we construct the initial graph using kNN algorithm with $k = 3$. For each run, the dataset is randomly split 10 times with 60% training, 20% validation and 20% test nodes and repeated 10 times. The hyperparameters are included in Appendix D.



(a) LAG-GCN training progress. (b) GCNs trained with learned graphs from LAG-GCN.

Figure 2: LAG-GCN trained on Texas dataset (with original graph initialization) shows that **(a)** the marginal likelihood correlates with the validation loss, and **(b)** the learned graphs (earlier to later ●●●) objectively improve the performance of GCN.

4.1. Marginal Likelihood as Graph Learning Objective

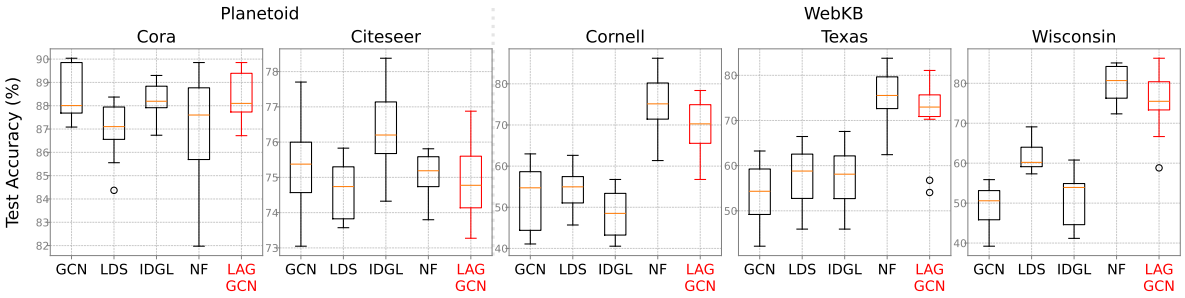
Marginal likelihood has been shown to reflect model's generalizability (MacKay, 1992; Minka, 2000; Fong and Holmes, 2020). From the training progress of LAG-GCN, shown in Figure 2(a) for Texas dataset, the marginal likelihood closely matches the shape of the validation loss of the learned model and graph structure. Indicating that marginal likelihood correlates with the generalizability of both the GNN model and the learned graph structures. This is further confirmed in Figure 2(b) where we used the checkpoint graph structures during LAG-GCN training to train GCNs: GCNs with better graph structures (higher marginal likelihood and at later epochs) attain higher test accuracies.

Comparison of the test performance of vanilla GCN and LAG-GCN in Figure 3 shows that LAG overall improves the performance of GCN when the initial graph structure is suboptimal as in the case of WebKB original graphs (in Figure 3(a)) and kNN graph initializations (Figure 3(b)) – indicating effectiveness of the marginal likelihood objective.

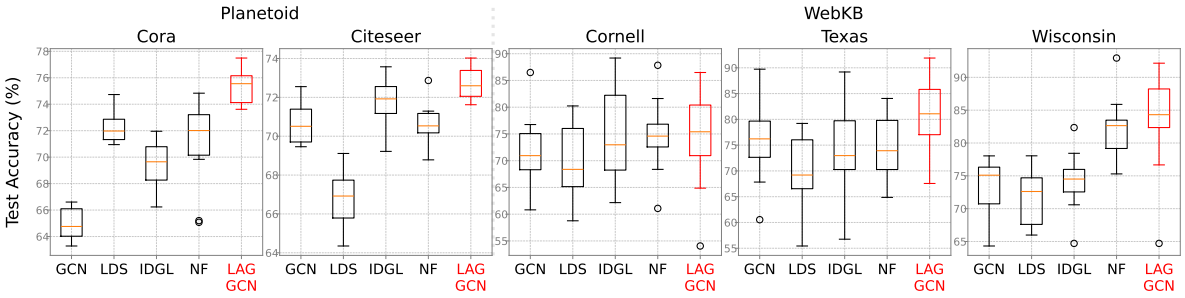
4.2. Comparison of LAG-GCN with other GSL methods

We compare LAG-GCN with other GCN-based GSL methods: Learning Discrete Structures (LDS) (Franceschi et al., 2019) and Iterative Deep Graph Learning (IDGL) (Chen et al., 2020b), and Transformer-based GSL method: NodeFormer (Wu et al., 2022). Briefly, LDS models the edges with Bernoulli and learns the graph with validation data; IDGL treats the structure learning problem as a similarity metric learning problem using learned node embeddings; NodeFormer use node pairs. The results are shown in Figure 3.

LAG-GCN in general outperforms LDS and IDGL. Under the kNN graph initialization setting, LAG-GCN outperforms LDS and IDGL across all the datasets. With original graph initialization, although IDGL perform slightly better on Planetoid, LAG-GCN boasts a significant improvement on the WebKB datasets, with average of 13.88% and 19.28% higher accuracies than LDS and IDGL, respectively.



(a) Original graph initialization



(b) kNN ($k = 3$) graph initialization

Figure 3: Comparison of LAG-GCN with GCN and other GSL methods using **(a)** original and **(b)** kNN graph initialization. Overall, LAG-GCN outperforms GCN and other GCN-based GSL methods LDS and IDGL, while matching the performance of Transformer-based GSL method NodeFormer, even with just a simple GCN model (tabulated results in App. E).

Even with a simple GCN model, LAG-GCN can match the performance of NodeFormer. Although under original graph initialization (in Figure 3(a)), LAG-GCN performed on par or lower than NodeFormer, with kNN graph initialization (in Figure 3(b)), LAG-GCN generally performs better than NodeFormer across all datasets: increasing average test accuracy by 3.27% on Planetoid and 2.09% on WebKB.

5. Conclusion

We proposed marginal likelihood as a simple objective for learning graph structures for GNN. We implemented *Laplace Approximation-based Graph (LAG)* structure learning, using Laplace approximation of the marginal likelihood and Straight Through Estimator (STE) to learn discrete graphs. Experiments on GCN-based LAG (LAG-GCN) shows the effectiveness of learning with marginal likelihood; LAG-GCN generally outperforms GCN and other GCN-based GSL methods (Franceschi et al., 2019; Chen et al., 2020b), while matching the performance of Transformer-based GSL method (Wu et al., 2022), despite the architectural disadvantage. In the future, we aim to improve the scalability of LAG.

References

Uri Alon and Eran Yahav. On the bottleneck of Graph Neural Networks and its practical implications. In *International Conference on Learning Representations*, 2021.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, 2013.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep Graph Convolutional Networks. In *Proceedings of the International Conference on Machine Learning*, 2020a.

Yao Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for Graph Neural Networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020b.

E Fong and C C Holmes. On the marginal likelihood and cross-validation. *Biometrika*, 107 (2):489–496, 2020.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for Graph Neural Networks. In *Proceedings of the International Conference on Machine Learning*, 2019.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2005.

Roger B. Grosse and Ruslan Salakhutdinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2015.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian Graph Neural Networks with adaptive connection sampling. In *Proceedings of the International Conference on Machine Learning*, 2020.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *Proceedings of the International Conference on Machine Learning*, 2021.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust Graph Neural Networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can GCNs go as deep as CNNs? In *The International Conference on Computer Vision*. IEEE, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper Graph Neural Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2020.

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference*, 2022.

Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. *A flexible generative framework for graph-based semi-supervised learning*. 2019.

David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

David J. C. MacKay. Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505, 1995.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2010.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2015.

Thomas P. Minka. Automatic choice of dimensionality for PCA. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2000.

Hoang NT and Takanori Maehara. Revisiting Graph Neural Networks: All we have is low-pass filters. *ArXiv*, 2019.

Kenta Oono and Taiji Suzuki. Graph Neural Networks exponentially lose expressive power for node classification. *ArXiv*, 2019.

Elinas Pantelis, Edwin V. Bonilla, and Louis C. Tiao. Variational inference for Graph Convolutional Networks in the absence of graph data and adversarial settings. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network model. *IEEE Transactions on Neural Networks*, 20 (1):61–80, 2008.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lió, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*, 2021.

Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the International Conference on Machine Learning*, 2018.

Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based Graph Convolutional Network. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2019.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2016.

Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Üstebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in Graph Neural Networks: current limitations and effective designs. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

Appendix A. Graph Structure Learning (GSL)

Graph structure learning (GSL) for GNN involves learning the graph structure $\mathbf{A}$ jointly with the GNN model parameters $\boldsymbol{\theta}$. The difficulty in learning $\mathbf{A}$ is due to its exponential search space. This requires regularizing the graph to ensure tractability in learning and generalizabiility in the learned graph.

Most structure regularization methods are based on assumptions about the target graph structure. These methods are difficult to implement and generally too restrictive. Many works design the loss function to encourage properties like sparsity, homophily and feature smoothness in the learned graph; for example, Yang et al. (2019); Chen et al. (2020b); Jin et al. (2020) penalize edges between nodes with large l_2-distance between their features. However, selecting which properties are desirable and designing a good loss function for them is non-trivial. Other works instead does simple postprocessing after learning the graph: Chen et al. (2020b); Liu et al. (2022) use k-Nearest Neighbor and ϵ- thresholding to sparsify the graphs, but such solution runs the risk of oversimplifying the graph.

Variational methods face the same problem. Variational GSL regularize the graph structure through prior on the structure. (Zhang et al., 2019), for example, assumes the optimal graph is a mixed-membership stochastic block model (MMSBM) graph and defined the prior on the parameters of the MMSBM model. In Pantelis et al. (2020), each edge is modeled as a Bernoulli distribution and a prior is defined over them using the input adjacency matrix. Defining a good prior for the problem is difficult, and can also be restrictive.

Appendix B. Approximation of the Hessian

B.1. Generalized Gauss-Newton (GGN)

Generalized Gauss-Newton (GGN) extends the Gauss-Newton matrix approximation to general loss function (Schraudolph, 2002; Martens, 2010) and approximates the Hessian w.r.t the parameters. The GGN matrix is given by:

$$\mathbf{H}_\theta \approx \mathbf{H}_\theta^{\text{GGN}} = \sum_{i=1}^{N} [\mathbf{J}_\theta^\top \mathbf{H}_L \mathbf{J}_\theta]_i + \mathbf{P}_\theta, \tag{4}$$

where $\mathbf{J}_\theta \in \mathbb{R}^{C\times P}$ is the Jacobian matrix w.r.t to the parameters, and $\mathbf{H}_L$ and $\mathbf{P}_\theta$ are the Hessian of the negative log -likelihood and -prior, respectively. More specifically, $\mathbf{H}_L := -\nabla^2_{\mathbf{ff}} \log p(\mathbf{y}|\mathbf{f})$ is the Hessian of the (cross-entropy) loss w.r.t to the predictions $\mathbf{f}$, therefore has dimension $C \times C$, while the Hessian of the negative log-prior $\mathbf{P}_\theta := -\nabla^2_{\theta\theta} \log p(\theta|\mathcal{M})$ has dimension $P \times P$. So the resultant GGN matrix is $P \times P$ and the complexity is $O(NPC^2 + NCP^2)$.

B.2. Empirical Fisher (EF)

The empirical Fisher (EF) (Martens and Grosse, 2015; Grosse and Salakhutdinov, 2015) estimates the Fisher information matrix using the data $\mathcal{D}$, and is defined as:

$$\mathbf{H}_\theta \approx \mathbf{H}_\theta^{\text{EF}} = \sum_{i=1}^{N} \mathbf{g}_{\theta,i}^\top \mathbf{g}_{\theta,i} + \mathbf{P}_\theta, \tag{5}$$

where $\mathbf{g}_{\theta,i} \in \mathbb{R}^{P\times 1}$ is the gradient of the log-likelihood $\mathbf{g}_{\theta,i} := \nabla_\theta \log p(\mathcal{D}_i|\theta, \mathcal{M})$, and $\mathbf{g}_{\theta,i}^\top \mathbf{g}_{\theta,i} \in \mathbb{R}^{P\times P}$ is the outer product. The resultant EF matrix is $P \times P$ and the complexity is $O(NP^2)$. For various matching pairs (such as linear activation function and square error, sigmoid and cross-entropy, and softmax and negative log-likelihood), the Fisher information matrix and GGN are equivalent (Pascanu and Bengio, 2014). Hence the argument for EF as an approximation for the Hessian (Bottou et al., 2018).

B.3. Block and Diagonal Approximation

While the GGN and EF approximations allow for more manageable computation of the Hessian, practically storing and computing the determinant of a $P \times P$ matrix is still not feasible. A solution is to make block diagonal or diagonal approximations of the Hessian. Since the Hessian is typically diagonal dominant, such an approximation is reasonable. Block diagonal approximation of the Hessian is calculated through Kronecker factorization of the Fisher matrix (Martens and Grosse, 2015) or the GGN matrix (Botev et al., 2017); that is, for each layer l of the neural network: $[\mathbf{H}_\theta]_l \approx \mathbb{E}[\mathbf{Q}_l] \otimes \mathbb{E}[\mathbf{G}_l]$, where $\mathbf{Q}_l \in \mathbb{R}^{d_{l-1}\times d_{l-1}}$ matrix is calculated from the input activations to the l-th layer, and $\mathbf{G}_l \in \mathbb{R}^{d_l \times d_l}$ is based on the gradient of the output (d_l corresponds to the output dimension of the l-th layer). Diagonal approximation further reduces computation by only using the diagonal entries of the Hessian.

Appendix C. Dataset

Experiments were conducted on 5 graph datasets (Table 1). The Planetoid datasets contain Cora and Citeseer (Yang et al., 2016) citation networks where nodes represent documents and edges represent citation links. While WebKB datasets have Cornell, Texas and Wisconsin (Pei et al., 2020) where nodes repesent web pages and edges representing hyperlinks. The node features across all the datasets are bag-of-word representation of the documents/ web pages.

Dataset	Nodes	Edges	Classes
Cora	2,708	10,556	7
Citeseer	3,327	4,732	6
Cornell	183	295	5
Texas	183	309	5
Wisconsin	251	499	5

Table 1: Datasets statistics

Appendix D. Hyperparameters

The hyperparameters used for LAG-GCN are shown here.

Table 2: Hyperparameters of LAG-GCN with original graph structure initialization.

Hyperparameter	Cora	Citeseer	Cornell	Texas	Wisconsin
n_epochs	200	100	1000	400	400
n_epoch_burnin	50	50	10	10	10
n_epoch_cutof	150	100	900	350	400
weight_decay	$5.0 \cdot 10^{-5}$	$5.0 \cdot 10^{-5}$	$5.0 \cdot 10^{-4}$	$5.0 \cdot 10^{-4}$	$5.0 \cdot 10^{-4}$
lr	0.001	0.001	0.01	0.03	0.02
lr_adj	0.8	1.5	10	10	10
hidden_channels	64	64	64	128	128
momentum_adj	0.9	0.9	0.9	0.9	0.9
residual_conn	False	False	True	True	True
norm	-	-	layer	layer	layer

Table 3: Hyperparameters of LAG-GCN with kNN graph structure initialization.

Hyperparameter	Cora	Citeseer	Cornell	Texas	Wisconsin
n_epochs	200	400	1000	400	600
n_epoch_burnin	20	50	10	10	10
n_epoch_cutof	1000	350	900	350	500
weight_decay	$5.0 \cdot 10^{-5}$	$5.0 \cdot 10^{-5}$	$5.0 \cdot 10^{-4}$	$5.0 \cdot 10^{-4}$	$5.0 \cdot 10^{-4}$
lr	0.001	0.001	0.01	0.03	0.005
lr_adj	0.8	1.0	10	10	10
hidden_channels	64	64	64	128	128
momentum_adj	0.9	0.9	0.9	0.9	0.9
residual_conn	False	False	True	True	True
norm	-	-	layer	layer	layer

Appendix E. Experimental Results (Tabulated)

We show the tabulate results of the experiments from Figure 3.

Table 4: Mean test accuracy in % (± standard error) with original graph structure initialization (Corresponding to Figure 3(*a*)).

Dataset	Cora	Citeseer	Cornell	Texas	Wisconsin
GCN	88.53 ± 0.37	75.25 ± 0.39	52.49 ± 2.47	53.92 ± 2.12	49.49 ± 1.58
LDS	86.96 ± 0.38	74.67 ± 0.26	54.28 ± 1.66	57.92 ± 2.01	61.62 ± 1.11
IDGL	88.19 ± 0.24	76.33 ± 0.36	48.62 ± 1.79	57.84 ± 2.03	51.18 ± 2.01
NodeFormer	86.94 ± 0.75	75.07 ± 0.20	75.14 ± 2.19	75.22 ± 1.90	79.80 ± 1.49
LAG-GCN	88.43 ± 0.32	74.88 ± 0.34	68.89 ± 2.26	71.05 ± 2.65	75.53 ± 2.46

Table 5: Mean test accuracy in % (± standard error) with kNN graph initialization (Corresponding to Figure 3(*b*)).

Dataset	Cora	Citeseer	Cornell	Texas	Wisconsin
GCN	64.93 ± 0.37	70.62 ± 0.32	71.76 ± 2.13	75.86 ± 2.47	73.43 ± 1.28
LDS	72.22 ± 0.36	66.82 ± 0.48	69.92 ± 2.17	70.08 ± 2.20	71.82 ± 1.33
IDGL	69.43 ± 0.60	71.68 ± 0.39	74.84 ± 2.80	74.05 ± 2.73	74.12 ± 1.41
NodeFormer	71.01 ± 1.02	70.58 ± 0.34	74.68 ± 2.15	74.57 ± 1.89	82.35 ± 1.46
LAG-GCN	75.45 ± 0.43	72.70 ± 0.24	74.00 ± 2.79	80.70 ± 2.19	83.16 ± 2.35