mathematics

MDPI

Article

Robust Graph Structure Learning with Virtual Nodes Construction

Wenchuan Zhang [1], Weihua Ou [1,*], Weian Li [1], Jianping Gou [2], Wenjun Xiao [1] and Bin Liu [1]

[1] School of Big Data and Computer Science, Guizhou Normal University, Guiyang 550025, China; 20010230641@gznu.edu.cn (W.Z.); liweian@gznu.edu.cn (W.L.); 460181694@gznu.edu.cn (W.X.); liubin@gznu.edu.cn (B.L.)
[2] College of Computer and Information Science, Southwest University, Chongqing 400700, China; goujianping@ujs.edu.cn
* Correspondence: ouweihua@gznu.edu.cn

Abstract: Graph neural networks (GNNs) have garnered significant attention for their ability to effectively process graph-related data. Most existing methods assume that the input graph is noise-free; however, this assumption is frequently violated in real-world scenarios, resulting in impaired graph representations. To address this issue, we start from the essence of graph structure learning, considering edge discovery and removal, reweighting of existing edges, and differentiability of the graph structure. We introduce virtual nodes and consider connections with virtual nodes to generate optimized graph structures, and subsequently utilize Gumbel-Softmax to reweight edges and achieve differentiability of the Graph Structure Learning (VN-GSL for abbreviation). We conducted a thorough evaluation of our method on a range of benchmark datasets under both clean and adversarial circumstances. The results of our experiments demonstrate that our approach exhibits superiority in terms of both performance and efficiency. Our implementation will be made available to the public.

Keywords: graph neural networks; graph representation learning; deep learning

MSC: 68R10; 57M15; 62A09; 65D18; 65S05

Citation: Zhang, W.; Ou, W.; Li, W.; Gou, J.; Xiao, W.; Liu, B. Robust Graph Structure Learning with Virtual Nodes Construction. *Mathematics* **2023**, *11*, 1397. https://doi.org/10.3390/math11061397

Academic Editor: Andrea Scozzari

Received: 10 February 2023
Revised: 5 March 2023
Accepted: 6 March 2023
Published: 13 March 2023

1. Introduction

Graph-structured data, in which nodes are inter-connected by a relational structure, are commonly encountered in a variety of domains including natural sciences (e.g., chemistry molecules) [1–3], social systems [4,5], finance [6], and so forth. Graph neural networks (GNNs) have demonstrated their effectiveness in leveraging such data dependencies in a range of applications. However, most GNNs rely on the observed graph as input, assuming it perfectly represents the accurate and complete relationships between nodes. In reality, many real-world graphs are noisy or incomplete due to data collection errors [7]. For example, transportation networks such as road networks or flight networks may not accurately represent all possible routes or connections between nodes due to missing or incomplete data. Similarly, economic networks such as supply chain networks or financial networks may not accurately depict all relationships between nodes due to market changes or other factors.

These issues pose a significant challenge to the application of GNNs to real-world scenarios, particularly in risk-critical situations, and motivate the study of graph structure learning (GSL). Graph structure learning (GSL) involves inferring optimal graph structures and representations from data that are generated by or correlated with the graph. One approach to GSL involves using similarity functions, such as Gaussian kernels [8], cosine similarity measurement [7], edge attention reweighting [9], and multilayer perceptron [10], to determine the confidence of edges based on the similarity between nodes. Another

approach to GSL involves optimizing the adjacency matrix, and has introduced techniques such as learning a discrete probability distribution [11], neural approaches [12,13], Bayesian inference [14], and direct learning [15] to address the increased difficulties in optimization. These methods generally consider the initial graph structure and sparsity, feature smoothness, and low ranking in their blending goals, aiming to learn more efficient graph structures and representations for downstream tasks. However, several challenges have been identified in this field, including:

- Discovering new connections and removing redundant connections: One of the main goals of graph structure learning is to identify important connections in the graph and eliminate unnecessary or redundant connections. This can be challenging, as it requires the ability to distinguish between relevant and irrelevant connections based on the available data and the desired properties of the graph.
- Recalculating weights: Another challenge in graph structure learning is the need to recalculate the weights of the edges in the graph. For example, to reflect the changing importance or strength of connections, or to incorporate additional information about the edges into the graph.
- Differentiability of discrete graph structures: The other challenge in graph structure learning is the differentiability of discrete graph structures, which can make it difficult to apply gradient-based optimization methods. This is especially relevant for node-level tasks, where the latent graph may potentially connect a large number of instance nodes. To enable end-to-end differentiable optimization for these discrete structures, innovative solutions are required.

To address these problems, in this work, we present a method for robust graph structure learning by augmenting the original adjacency matrix with virtual nodes construction and learning the optimized structure that model the connections between the original and virtual nodes. The refined adjacency matrix is then obtained by recalculating the weights of edges in the augmented adjacency matrix using the Gumbel-Softmax function and optimized through gradient descent. Our method is evaluated on both clean and attacked datasets and demonstrates strong performance on the benchmark dataset without significantly increasing resource consumption. In addition, our method outperforms other methods on attacked datasets, highlighting its robustness. In general, our contributions can be summarized as follows:

- Motivated by the challenges of edge discovery and removal, weight recalculation, and the differentiability of graph structures, we present a novel method for graph structure learning to discover new connections, edge weight recalculation, and redundant edge eliminating.
- We demonstrate the effectiveness of our approach on the benchmark dataset, showing performance improvement (up to 3.4% compared to the state-of-the-art on the Cora) without significantly increasing resource consumption.
- We further demonstrate the robustness of our approach through evaluations on datasets under attack conditions, where it consistently outperforms other methods.

The remainder of this paper is structured as follows. In Section 2, we review relevant works in the field. Our proposed approach is detailed in Section 3, and experimental results are presented in Section 4. Finally, we discuss future directions and conclude the work in Section 5.

2. Related Works

2.1. Graph Neural Networks

Graph neural networks (GNNs) are a type of deep learning model that operates on graphs and are used to analyze and understand graph-structured data. GNNs can be divided into two main categories: spectral-based and spatial-based methods. Spectral GNNs utilize techniques from spectral graph theory to define convolution operations in the frequency domain. Examples of spectral GNNs include the method proposed by

Bruna et al. [16], which maps the graph to the spectral domain using the discrete Fourier transform and defines graph convolution operations accordingly, as well as ChebNet [17], which employs Chebyshev polynomials to approximate spectral graph convolution operations. Spatial GNNs, on the other hand, define convolution operations directly on the graph through the aggregation of neighbor node features. Representative models of this category, including GCN [17], which can be considered a simplified version of frequency domain methods. GraphSAGE [18], which further integrates sampling strategies. Graph Attention Networks (GATs) [19,20], in particular, utilize the attention mechanism to assign different weights to neighbors based on their node features in the graph. A comprehensive survey of GNNs can be found in [21].

2.2. Graph Structure Learning

Graph structure learning is a research area that aims to learn more effective graph structures and representations for downstream tasks. A number of methods have been proposed to sparsify large input graphs, including techniques that remove connections and edges based on topological information such as node degree distribution [22], the distance between pairs of nodes [23], the size of all cuts [24], and spectral properties [25,26]. Other approaches, such as edge reweighting methods GAT [19] and GLCN [9], use the attention mechanism or similarity of node features to weight existing edges in the graph. However, these methods may not fully capture the rich information provided by both node features and topology, and they may be prone to be affected by noisy data when the edges are sparse. NeuralSparse [27] attempts to learn robust graph representations by selecting a fixed number of edges for each node, but this fixed k-neighborhood assumption may limit the learning ability of the method and result in poor generalization performance. Non-Local GNN [28] and LDS [11] are other approaches that aim to enhance model expressiveness, but they may suffer from computational inefficiency or a large number of parameters, respectively. Overall, these methods of incorporating multiple sources of information tend to rely on practical experience rather than theoretical analysis to obtain optimized graph structures. In contrast to previous methods, our approach takes a holistic approach by considering the simultaneous optimization of node features and graph structure, which is achieved through the integration of edge discovery and removal, reweighting of edge weights, and differentiability of the graph structure in an iterative process. This enables the proposed model to extract inherent structures from perturbed graphs under both clean and various attack conditions.

3. The Proposed Approach

In this study, we investigate the application of graph convolution to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. The graph is represented using its adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where the (i, j)th position of $\mathbf{A}$ is a_{ij} and its input feature matrix $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where d is the dimensionality of the features. The graph can thus be represented as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$. The degree matrix is denoted as $\mathbf{D} = \text{diag}(d_1, d_2, \ldots, d_n)$, where $d_i = \sum_j a_{ij}$ is the degree of vertex i.

In this work, we focus on the problem of semi-supervised node classification in the context of graph structure learning. Following the typical setting, only a subset of vertices $\mathcal{V}_L = \{v_1, v_2, \cdots, v_m\}$ are associated with corresponding labels $\mathcal{Y}_L = \{y_1, y_2, \cdots, y_m\}$, where y_i denotes the label of v_i. Given the partial labels $\mathcal{Y}_L$ and the graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, the goal of the graph structure and representation learning for semi-supervised node classification is to learn a function $f_\Theta : \mathcal{V}_L \to \mathcal{Y}_L$ that maps vertices to labels and the GNN parameter Θ, while also learning the optimized adjacency matrix $\widetilde{\mathbf{A}}$. The optimization objective can be expressed as:

$$\begin{aligned} &\min_{(\Theta, \widetilde{\mathbf{A}})} \mathcal{L}_{GNN}(\Theta, \widetilde{\mathbf{A}}, \mathbf{X}, \mathcal{Y}_L) \\ &= \sum_{v_i \in \mathcal{V}_L} \ell(f_\Theta(\widetilde{\mathbf{A}}, \mathbf{X})_i, y_i) \end{aligned} \tag{1}$$

where $\ell(\cdot)$ is the loss function that measures the difference between the prediction and ground-truth labels. $f_\Theta(\widetilde{\mathbf{A}}, \mathbf{X})_i$ is the prediction label of node v_i, where Θ is the GNN parameter.

We subsequently elaborate on the VN-GSL, the proposed model. As depicted in Figure 1, the VN-GSL is comprised of three components. The first component involves the utilization of virtual nodes to identify new connections within the graph structure. The second component pertains to the recalculation of weights for existing nodes in the graph. The final component involves the elimination of redundant or low-weight edges as inferred from the preceding steps. The following sections provide a detailed examination of these components.

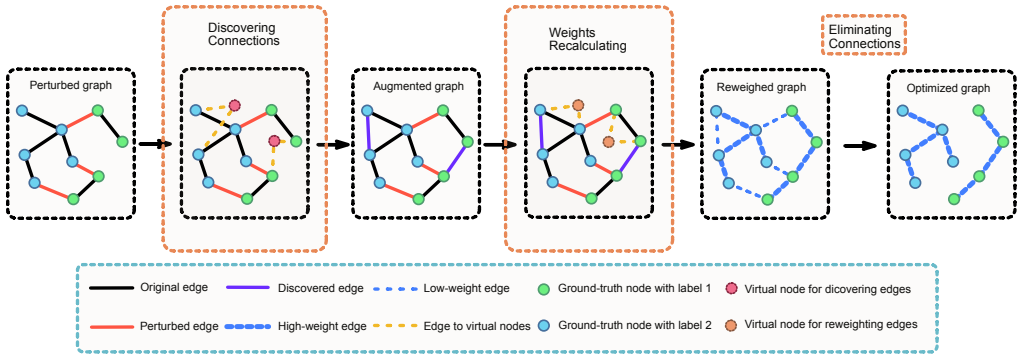


Figure 1. The conceptual illustration of the VN-GSL proposed.

3.1. Discovering Connections

To avoid the resulting exponential number of possible substructures, in this study, we augment the original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with p virtual nodes, generating two new adjacency matrices: $\mathbf{B} \in \mathbb{R}^{n\times p}$ and $\mathbf{D} \in \mathbb{R}^{p\times n}$ that represent the connectivity between the n original nodes and the p virtual nodes, and the reverse connectivity between the virtual nodes and the original nodes, respectively. In addition, we also introduce an adjacency matrix $\mathbf{C} \in \mathbb{R}^{p\times p}$ that represents the connectivity between the virtual nodes themselves. The new graph is then represented as $\mathcal{G}' = (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the original adjacency matrix and $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \in \mathbb{R}^{n\times d}$ is the input feature matrix, with d being the dimensionality of the features.

Lemma 1. *The product* **BCD** *represents the increased connectivity (weights) of the original nodes in the graph* $\mathcal{G}$ *through the* p *virtual nodes.*

Proof. Matrix $\mathbf{B} \in \mathbb{R}^{n\times p}$ represents the connectivity weights between the vertices $\{v_1, v_2, \ldots, v_n\}$ in the original graph $\mathcal{G}$ and the virtual nodes $\{u_1, u_2, \ldots, u_p\}$. The connectivity weight from v_i to u_j is given by b_{ij}, which represents the element at the (i, j)th position in $\mathbf{B}$. Matrix $\mathbf{C}$ is the adjacency matrix of the virtual nodes, with vertices ordered as $u_1, u_2, \ldots, u_p$. The (i, j)th element of the matrix product $\mathbf{BC}$ is denoted by $(\mathbf{BC})_{ij}$:

$$(\mathbf{BC})_{ij} = b_{i1}c_{1j} + b_{i2}c_{2j} + \cdots + b_{ip}c_{pj} \tag{2}$$

This element can be calculated as the sum of the products of the elements of $\mathbf{B}$ and $\mathbf{C}$, with the element at the (i, k)th position ($k \leq p$) in $\mathbf{B}$ and the element at the (k, j)th position in $\mathbf{C}$ being multiplied for all possible intermediate vertices u_k. This includes the connectivity weights from v_i to u_k as well as the edge from u_k to u_j (connections between the p virtual nodes). By summing the products of these elements according to the rules of matrix multiplication, we can obtain the desired result, which is the connectivity weights from each vertex in $\mathcal{G}$ to the p virtual nodes.

Similarly, we can find that for matrices **B**, **C** and **D** we have:

$$\begin{aligned}(\mathbf{BCD})_{ij} &= \sum_{k=1}^{p}\sum_{l=1}^{p} b_{ik}c_{kl}d_{lj} \\ &= b_{i1}c_{11}d_{1j} + b_{i2}c_{21}d_{1j} + \cdots + b_{ip}c_{pp}d_{pj}\end{aligned} \tag{3}$$

where b_{ik} ($k \leq p$) represents the connectivity between the vertex v_i and the virtual vertex u_k, including both the connectivity from v_i to an intermediate virtual vertex u_k and the connectivity from u_k to another intermediate virtual vertex u_l. Similarly, d_{lj} ($l \leq p$) represents the connectivity between the virtual vertex u_l and the vertex v_j. By the principle of matrix multiplication, when we sum these products over all possible intermediate vertices, we obtain the desired result. This completes the proof that the matrix product **BCD** represents the connectivity (weights) of the original nodes in the graph $\mathcal{G}$ through the p virtual nodes. □

We then propose our approach for discovering new connections and eliminating unnecessary connections in the graph, which is based on the aforementioned concept of matrix multiplication to extend the reachability of nodes in the graph. Specifically, we introduce virtual nodes into the original graph and learn three matrices: **B**, which maps the original nodes to the virtual nodes, **C**, which captures the interactions between the virtual nodes, and **D**, which maps the virtual nodes back to the original nodes.

Based on the above analysis, we can utilize three trainable matrices, **B**, **C**, and **D**, to discover new weights and assign negative values to existing weights to alter the weight of the original graph. The number of parameters in this training strategy is significantly smaller than the number of parameters in the original adjacency matrix ($n \times n$). The new adjacency matrix can be constructed by this method using the original adjacency matrix as a prior and adding new edges or modifying the weight of the original adjacency matrix based on this foundation.

$$\mathbf{A}_{dis} = \mathbf{BCD} \tag{4}$$

where $\mathbf{A}_{dis}$ is the newly discovered weight and connection. In this way, we can effectively discover new connections in the graph and eliminate unnecessary connections. By learning these matrices, we are able to reduce the complexity of the optimization process, while still capturing the rich structural information present in the original graph.

While this process introduces a large number of edges connecting all instance nodes, which can range from thousands to millions, presents challenges in terms of maintaining accuracy and scalability. In order to facilitate end-to-end differentiable optimization of discrete structures, we propose reducing the search space for the addition of edges. Previous studies [29,30] have addressed the problem of integrating the recalculation of discrete graph weights with the estimation of gradients through the utilization of various forms of relaxation of discrete distributions. This enables differentiability through the reparameterization trick, facilitating the backpropagation of gradients. Among these techniques, the Gumbel-Softmax trick [31,32] has been widely employed as a means of generating differentiable graph structures and improving training stability in the context of discrete optimization problems.

$$a_{ij}^{\star} = \frac{\exp\left(\left(\log a_{ij} + \epsilon_j\right)/\tau\right)}{\sum_{w\in\mathcal{N}_i}\exp\left(\left(\log a_{ij} + \epsilon_w\right)/\tau\right)} \tag{5}$$

where Gumbel distribution is introduced with noise $s \sim \text{Uniform}(0,1)$ distribution, parameterized by $\epsilon = -\log(-\log(s))$ and temperature hyperparameter τ. Here, $\mathcal{N}_i$ represents the set of non-zero weight neighboring nodes of node i. When the value of τ is small, the Gumbel distribution approximates a discrete distribution, resulting in a sparse graph structure. As τ increases, the resulting distribution becomes increasingly similar to a uniform distribution.

Following the gumbel_softmax operation, we introduce the mask operation to further reduce the search space. The mask operation with threshold λ, for matrix $\mathbf{P}$, where the (i, j)th position is p_{ij}, is defined as follows:

$$\text{mask}(\lambda; p_{ij}) = \begin{cases} p_{ij} & \text{if} \quad p_{ij} > \lambda \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We transform Equation (5) into the following form with hyperparameter λ_1:

$$\mathbf{A}_{dis} = \text{mask}(\lambda_1; \text{gumbel\_softmax}(\mathbf{BCD})) \tag{7}$$

where $\tau = 1$, λ_1 will be adjusted as a hyperparameter in the following experiments.

3.2. Weights Recalculating and Differentiable of Graph Structures

In this section, we present our approach for adjusting the weights of edges. Building upon the trick mentioned in Lemma 1, we introduce three new trainable matrices, $\mathbf{E} \in \mathbb{R}^{n \times p}$, $\mathbf{F} \in \mathbb{R}^{p \times p}$, and $\mathbf{G} \in \mathbb{R}^{p \times n}$, with the goal of recalculating weights rather than adding new edges. The new graph is then represented as $\mathcal{G}' = (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{X})$. To this end, we also introduce a new mask $\mathbf{A}$ operation on matrix $\mathbf{P}$, where the (i, j)th position of $\mathbf{A}$ is a_{ij} and the (i, j)th position of $\mathbf{P}$ is p_{ij}, for the purpose of masking out irrelevant edges. This operation is defined as follows:

$$\text{mask}_{\mathbf{A}}(p_{ij}) = \begin{cases} p_{ij} & \text{if} \quad a_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The matrix for recalculating weights can be expressed as follows:

$$\mathbf{A}_{rec} = \text{mask}_{\mathbf{A}}(\mathbf{EFG}) \tag{9}$$

In this section, $\text{mask}_{\mathbf{A}}$ allows us to adjust the weights of edges in the graph without adding new edges and ignore irrelevant edges in the graph in this section. The matrices $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$ are trainable, which means that they can be optimized during the training process. This allows us to fine-tune the weights of the edges in the graph to better fit the data and improve the performance of our model.

By combining Equations (7) and (9) and using the original adjacency matrix as a prior, along with the previously mentioned gumbel_softmax, we can obtain a new, optimized adjacency matrix $\mathbf{S}$ that incorporates newly discovered connections and reweights the edges accordingly:

$$\begin{aligned} \mathbf{S} &= \text{gumbel\_softmax}(\mathbf{A} + \mathbf{A}_{dis} + \mathbf{A}_{rec}) \\ &= \text{gumbel\_softmax}(\mathbf{A} + \\ &\text{mask}(\lambda_1; \text{gumbel\_softmax}(\mathbf{BCD})) \\ &+ \text{mask}_{\mathbf{A}}(\mathbf{EFG})) \end{aligned} \tag{10}$$

where $\mathbf{A}$ denotes the original adjacency matrix. In our approach, we utilize the Gumbel-Softmax function to achieve a smooth blend between $\mathbf{A}$ and the learned matrices $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$. The use of the gumbel_softmax function allows us to incorporate additional information about the edges into the graph structure while maintaining its differentiability. This enables the application of gradient-based optimization methods for learning the graph structure. Furthermore, the incorporation of probability sampling within this strategy introduces a degree of stochasticity, which has been shown to enhance the model's generalization abilities with respect to unseen data. This is a notable advantage of our approach, as it allows us to effectively learn the relationships between the nodes in the graph and make predictions about the graph structure.

3.3. Eliminating Connections

To delete connections, we simply apply a masking operation to the one mentioned mask in Section 3.1, and introduce a new thresholded λ_2. This operation is zero out of the low-weight edges in the matrix $\mathbf{S}$ learned in the previous section (we consider the low-weight edges in the learned $\mathbf{S}$ matrix to be insignificant or redundant). This allows us to obtain our final, optimized matrix $\tilde{\mathbf{A}}$:

$$\begin{aligned}\tilde{\mathbf{A}} &= \text{mask}(\lambda_2; \mathbf{S}) \\ &= \text{mask}(\lambda_2; \text{gumbel\_softmax}(\mathbf{A} + \\ &\text{mask}(\lambda_1; \text{gumbel\_softmax}(\mathbf{BCD})) \\ &+ \text{mask}_{\mathbf{A}}(\mathbf{EFG})))\end{aligned} \tag{11}$$

where λ_2 will be adjusted as a hyperparameter in the following experience. In this section, we present a method for eliminating connections in the graph in order to reduce complexity and improve performance. To accomplish this, we introduce the thresholded masking operation, mask with thresholded λ_2, which sets the weights of low-valued edges in the matrix $\mathbf{S}$ to zero. This allows us to selectively remove edges that are deemed to be insignificant or redundant based on their weight.

By combining the mask operation, we obtain the final, optimized matrix $\tilde{\mathbf{A}}$. This matrix is expected to be more efficient and effective than the original matrix due to the removal of unnecessary connections.

It is worth noting that the threshold λ_2 is a hyperparameter that can be adjusted to achieve different levels of connection pruning. A higher value for λ_2 will result in the elimination of more connections, but it may also lead to the loss of important information. Thus, finding an optimal value for λ_2 may require experimentation.

3.4. Proposed Method

Finally, the predicted labels of the l-layer model can be iteratively obtained through the following process:

$$\begin{aligned}&\mathbf{X}^{(l)} = \\ &\sigma(\text{mask}(\lambda_2; \text{gumbel\_softmax}(\mathbf{A} + \\ &\text{mask}(\lambda_1; \text{gumbel\_softmax}(\mathbf{B}^{(l)}\mathbf{C}^{(l)}\mathbf{D}^{(l)})) \\ &+ \text{mask}_{\mathbf{A}}(\mathbf{E}^{(l)}\mathbf{F}^{(l)}\mathbf{G}^{(l)})))\mathbf{X}^{(l-1)}\mathbf{W}^{(l)})\end{aligned} \tag{12}$$

where $\mathbf{X}^{(0)} = \mathbf{X}$ and $\mathbf{W}^{(l)}$, $\mathbf{B}^{(l)}$, $\mathbf{C}^{(l)}$, $\mathbf{D}^{(l)}$, $\mathbf{E}^{(l)}$, $\mathbf{F}^{(l)}$ and $\mathbf{G}^{(l)}$ represent the trainable parameters and corresponding trainable parameters of the l-th layer, respectively. σ is the non-linear function (e.g., ReLU). The computation procedure for the model is outlined in Algorithm 1. The predicted label distribution and cross-entropy loss can be expressed as follows:

$$\begin{aligned}\mathbf{Z} &= \text{softmax}(\mathbf{X}^{(l)}) \\ \mathcal{L} &= -\sum_{i \in \mathcal{V}_L} \sum_{p=1}^{c} \mathcal{Y}_{[i,p]} \log \mathbf{Z}[i,p]\end{aligned} \tag{13}$$

where $\mathbf{X}^{(l)}$ are then transformed into a label distribution through softmax normalization. $\mathbf{Z}[i,p]$ denotes the output of the i-th node corresponding to the p-th class, and $\mathcal{Y}_{[i,p]}$ denotes the label of the i-th node corresponding to the p-th class. It is worth noting that our model does not require additional loss functions to guide the optimization of the graph structure.

Algorithm 1: VN-GSL

Input: Number of layers l, input feature $\mathbf{X}$, adjacency matrix $\mathbf{A}$, learning rate η.
Output: Label prediction $\mathbf{Z}$, trainable parameters $\mathbf{B}^{(j)}$, $\mathbf{C}^{(j)}$, $\mathbf{D}^{(j)}$, $\mathbf{E}^{(j)}$, $\mathbf{F}^{(j)}$, $\mathbf{G}^{(j)}$, $\mathbf{W}^{(j)}$.

$\mathbf{X}^{(0)} \leftarrow \mathbf{X}$
for $j = 1$ *to* l **do**
  Randomly initialize $\mathbf{W}^{(j)}$
  Randomly initialize $\mathbf{B}^{(j)}$, $\mathbf{C}^{(j)}$, $\mathbf{D}^{(j)}$, $\mathbf{E}^{(j)}$, $\mathbf{F}^{(j)}$, $\mathbf{G}^{(j)}$
end
while *Stopping condition is not met* **do**
  for $j = 1$ *to* l **do**
    $\mathbf{X}^{(j)} =$
    $\sigma(\text{mask}(\lambda_2; \text{gumbel\_softmax}(\mathbf{A} + \text{mask}(\lambda_1; \text{gumbel\_softmax}($
    $\mathbf{B}^{(j)}\mathbf{C}^{(j)}\mathbf{D}^{(j)}))$
    $+ \text{mask}_{\mathbf{A}}(\mathbf{E}^{(j)}\mathbf{F}^{(j)}\mathbf{G}^{(j)})))\mathbf{X}^{(j-1)}\mathbf{W}^{(j)})$
  end
  $\mathbf{Z} = \text{softmax}(\mathbf{X}^{(l)})$
  for $j = 1$ *to* l **do**
    $\mathbf{W}^{(j)} \leftarrow \mathbf{W}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(j)}}$
    $\mathbf{B}^{(j)} \leftarrow \mathbf{B}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{B}^{(j)}}$
    $\mathbf{C}^{(j)} \leftarrow \mathbf{C}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{C}^{(j)}}$
    $\mathbf{D}^{(j)} \leftarrow \mathbf{D}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{D}^{(j)}}$
    $\mathbf{E}^{(j)} \leftarrow \mathbf{E}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{E}^{(j)}}$
    $\mathbf{F}^{(j)} \leftarrow \mathbf{F}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{F}^{(j)}}$
    $\mathbf{G}^{(j)} \leftarrow \mathbf{G}^{(j)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{G}^{(j)}}$
  end
end

4. Experiments

In this study, we conduct evaluations to demonstrate the efficacy of the proposed VN-GSL. Our objective is to address the following questions: Q1: How does VN-GSL perform in terms of learning graph structures under a semi-supervised setting? Q2: How robust is VN-GSL to adversarial graph structures? Q3: How do hyperparameters impact the performance of VN-GSL? Q4: What specific graph structures does VN-GSL learn?

4.1. Experimental Setup

4.1.1. Datasets

To evaluate the performance of our proposed model, we conducted experiments on five publicly available graph datasets, with the details of these datasets provided in Table 1. We adopted a semi-supervised node classification paradigm in which we partitioned the training, validation, and test sets, and only 20 nodes per label category were selected for training. This approach effectively mitigated the impact of overfitting on the evaluation outcomes.

- Wikipedia networks [33]. The Chameleon and Squirrel graphs were initially introduced as a way to examine the structure and organization of web pages within Wikipedia. These graphs consist of web pages represented as nodes and hyperlinks between pages represented as edges. Pei et al. [34] subsequently developed classification labels for the web pages in these graphs based on the average monthly traffic for each page. These labels are divided into five categories, with higher traffic corresponding

to higher categories. These graphs are still widely used in the field of network analysis and have been the focus of numerous studies in recent years.

- Citation networks [35]. In the field of graph network analysis, citation networks such as Cora, Citeseer, and Pubmed are commonly utilized datasets. These networks consist of papers represented as nodes, with citation relationships serving as connections between the nodes. Each paper in these networks either cites at least one other paper or is cited by other papers. In the context of semi-supervised classification, the titles or summaries of the papers are used as attribute information for known nodes. The goal of this task is to utilize the reference relationships between nodes, which are represented as a graph, to classify each node.

Table 1. Description of datasets.

Dataset	Nodes	Edges	Classes	Features	Train/Val/Test Nodes	Split Ratio(%)	Homophily
Cora	2708	5429	7	1433	140/500/1000	5.2/18.5/36.9	0.83
Citeseer	3327	4732	6	3703	120/500/1000	3.6/15.0/30.1	0.71
Pubmed	19,717	44,338	3	500	60/500/1000	0.3/2.5/5.1	0.79
Chameleon	2277	31,421	5	2325	100/500/1000	4.4/21.9/43.9	0.25
Squirrel	5201	198,493	5	2089	100/500/1000	1.9/9.6/19.2	0.22

4.1.2. Tasks and Baselines

In the semi-supervised node classification task, our goal is to classify papers into their respective categories or fields based on their citation relationships and content information. To evaluate the performance of our method, we compare it with state-of-the-art models and benchmark baselines specified in [19,20,36,37], including IDGL [7], GEN [38], LDS [11], Pro-GNN [15], SGC [39], APPNP [40], ChebNet [17], GCN [36], GAT [19], GATv2 [20], GraphSAGE [18], and Geom-GCN [34].

4.1.3. Implementation and Hyperparameters

In this study, we conducted experiments on a Linux server that was equipped with an NVIDIA TITAN RTX GPU and two Intel Xeon E5-2683 CPUs. The experiments were conducted using the PyTorch deep learning library version 1.7.1 and Python version 3.8.3. To improve the performance of the model, we performed a grid search over a range of hyperparameters, which included the number of layers, λ_1, λ_2, the learning rate, and various criteria for early stopping, optimization, dropout, and training epochs. These hyperparameters were carefully selected and varied in order to optimize the model's performance. Specifically, we explored the number of layers in the range of 1 to 8, searched for the p parameter in the range of 1 to 10, and tuned the dropout parameter from 0.2 to 0.8.

The model with the highest validation accuracy was selected for testing, and the detailed parameter settings can be found in Table A2. Throughout all of our experiments, we utilized the PyTorch Geometric library [41] to implement SGC, GCN, GAT, APPNP, and GraphSAGE. For the remaining baselines (ChebNet, LDS, Pro-GNN, Geom-GCN, etc.), we employed the source code provided by the authors and carefully fine-tuned it according to the settings in the original papers in order to achieve optimal performance. Our aim was to comprehensively evaluate the proposed model and compare it against various state-of-the-art approaches in the field.

4.2. Performance on Real-World Datasets (Q1)

The results presented in Table 2 evince that graph structure learning methods exhibit superior performance compared to those that rely on the original graph structure, particularly on homogeneous graphs with high homogeneity ratios such as the Cora, Citeseer, and Pubmed datasets. Notably, the VN-GSL method demonstrates the highest mean accuracy on these datasets, indicating that our approach effectively optimizes the original graph structure using ground-truth labels. This can be attributed to the model's ability to discover new edges and prune redundant ones, as well as recompute edge weights in

the original graph structure. However, on heterogeneous graphs with lower homogeneity ratios such as the Chameleon and Squirrel datasets, the VN-GSL method performs competitively, but is not the top performer. This may be due to our method's lack of explicit design to capture relationships between nodes that are further apart, and nodes with high structural and semantic similarity in heterogeneous graphs may be more distant from each other. As such, our method did not achieve optimal performance on the Chameleon dataset, and its advantage over other models on the Squirrel dataset is relatively minor. In reality, heterophilic graphs differ significantly from homogeneous graphs, and our future work needs to focus on addressing graph structure learning problems specifically in the context of heterogeneous graphs. Overall, these results demonstrate the potential of the VN-GSL method as a strong choice for graph-based machine learning tasks, particularly on homogeneous graphs with high homogeneity ratios.

Table 2. Performance comparison of graph neural network models on multiple datasets (mean accuracy (%) ± stdev).

	Cora	**Chameleon**	**Pubmed**	**Citeseer**	**Squirrel**
#Nodes $\lvert\mathcal{V}\rvert$	2708	2277	19,717	3327	5201
Hom. ratio	0.83	0.25	0.79	0.71	0.22
#Classes $\lvert\mathcal{Y}\rvert$	7	5	3	6	5
#Edges $\lvert\mathcal{E}\rvert$	5429	31,421	44,338	4732	198,493
SGC [39]	81.47 ± 0.54	49.50 ± 0.96	79.10 ± 2.21	69.57 ± 0.76	34.95 ± 1.52
GCN [17]	81.36 ± 0.43	48.52 ± 0.87	74.62 ± 0.70	68.91 ± 0.76	34.51 ± 1.43
GAT [19]	83.16 ± 0.47	46.70 ± 1.54	79.32 ± 0.64	69.72 ± 0.76	27.42 ± 2.83
APPNP [40]	83.39 ± 0.63	46.43 ± 0.66	80.32 ± 0.44	69.59 ± 0.62	33.82 ± 1.40
GraphSAGE [18]	80.35 ± 0.58	44.19 ± 1.90	73.69 ± 2.06	70.09 ± 0.90	28.69 ± 2.12
ChebNet [17]	81.95 ± 0.70	37.35 ± 0.70	78.54 ± 1.06	68.22 ± 0.91	21.42 ± 1.96
GATv2 [20]	83.34 ± 0.47	47.58 ± 1.14	79.51 ± 0.50	69.84 ± 0.68	27.98 ± 1.91
Geom-GCN [34]	64.03 ± 0.42	35.97 ± 0.52	77.26 ± 0.66	63.83 ± 0.92	25.85 ± 0.86
IDGL [7]	84.14 ± 1.00	40.03 ± 0.84	82.60 ± 0.84	70.33 ± 0.68	26.69 ± 1.46
GEN [38]	83.32 ± 0.56	**50.70 ± 0.96**	81.12 ± 1.01	71.42 ± 0.70	30.79 ± 2.74
Pro-GNN [15]	81.02 ± 1.06	50.57 ± 0.72	78.18 ± 0.94	68.00 ± 0.95	33.72 ± 2.50
LDS [11]	82.65 ± 1.35	49.78 ± 1.20	78.44 ± 1.96	70.48 ± 1.19	30.40 ± 0.54
VN-GSL	**84.68 ± 0.49**	49.99 ± 0.91	**82.79** ± 0.68	**71.55 ± 0.45**	**35.28 ± 1.19**

To further examine the efficacy of our proposed model, we applied t-SNE dimensionality reduction to the output of the final layer and obtained the result depicted in Figure 2. This visualization reveals the presence of distinct clusters in the projected 3D space, with the number of clusters corresponding to the number of labeled categories in each dataset. This correspondence indicates the discriminative ability of our model. These findings suggest that our method is effective in learning discriminant and distinct node features.

(**a**) Cora (**b**) Citeseer (**c**) Pubmed (**d**) Chameleon (**e**) Squirrel

Figure 2. We applied our model to several datasets and obtained t-SNE visualization. The class label assigned by the ground truth is represented by the outline color, while the class label predicted by the model is represented by the fill color.

In addition to assessing the effectiveness of our proposed model, we also conducted a comprehensive analysis of resource utilization, specifically focusing on memory consumption and execution time. As outlined in the appendix and summarized in Table 3, our approach did not incur a significant increase in computational overhead or memory usage by utilizing a virtual node construction technique that involves several low-rank matrix multiplications. In contrast, we avoided using resource-intensive mechanisms such as the multi-head attention employed in GAT [19] for graph reweighting. A detailed discussion of these findings can be found in Appendix B.3.

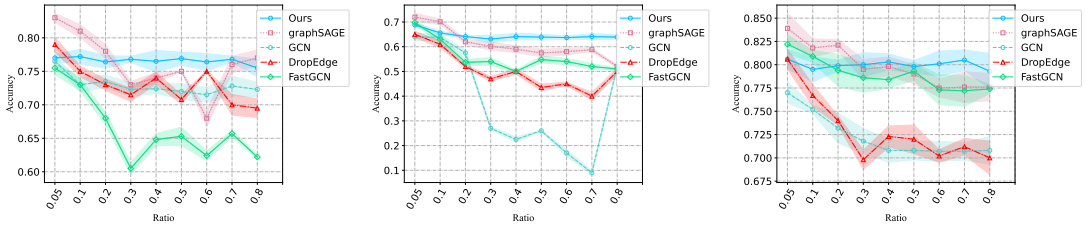
Table 3. Runtime and memory consumption on GCN, GAT, and VN-GSL.

Dataset & GNN	Runtime per Epoch (s)			Memory (MB)		
	Max	Min	Avg	Max	Min	Avg
Cora & GCN	0.00858	0.00346	0.00378	79.19	73.74	74.58
Cora & GAT	0.8921	0.7249	0.7547	7182.91	6419.65	6975.23
Cora & VN-GSL	0.2068	0.04687	0.05264	591.04	521.38	534.09
Citeseer & GCN	0.00915	0.00336	0.00382	162.07	121.36	140.88
Citeseer & GAT	1.00643	0.97289	0.99052	10,691.56	10,112.74	10,348.13
Citeseer & VN-GSL	0.15716	0.19862	0.18187	975.99	867.24	908.05

4.3. Performance on Robustness Analysis (Q2)

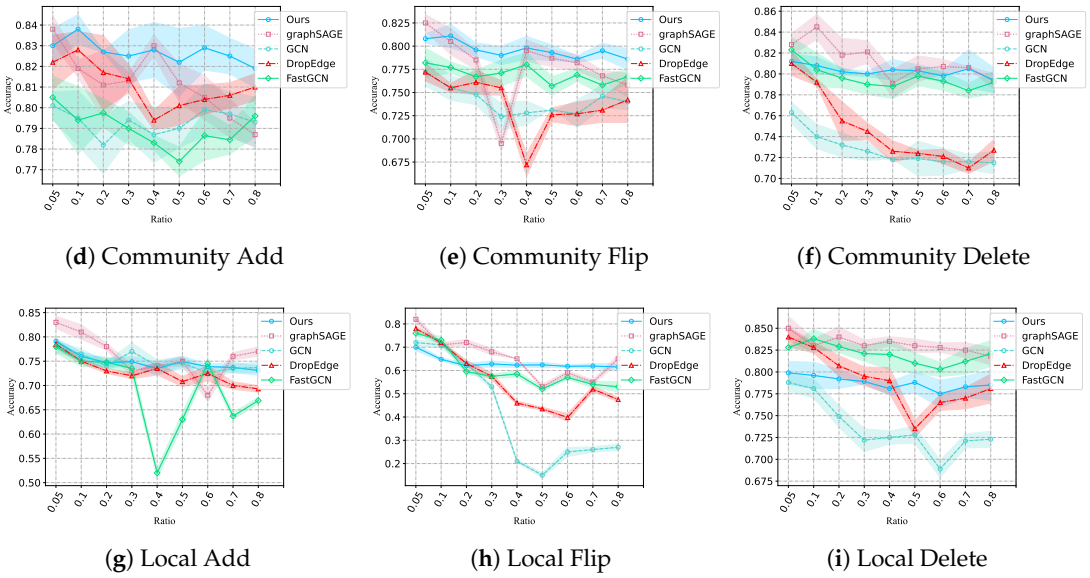
To evaluate the robustness of our model under various levels of structural noise, we conducted experiments following the methodology outlined in [42]. Specifically, we simulated different levels of attack strength by adding, flipping, and removing edges from local, community [43], and global viewpoints. The structural noise was generated using ratios ranging from 0.05 to 0.8 in increments of 0.05 for each type of noise. We compared the performance of our method to several benchmark graph neural network (GNN) models: GCN [36], FastGCN [44], GraphSAGE [18], Pro-GNN [15], and GraphSAGE [18]. Additional information on the experimental setup can be found in [42]. The hyperparameter settings used in these experiments are detailed and available in Table A2. To reduce variance, we conducted the same experiment six times with fixed random seeds.

We conducted comprehensive evaluations and comparisons, as shown in Figure 3, which demonstrated the high competitiveness of our method based on the experimental results. In the add and flip experiments, our method showed its ability to process redundant edges and recalculate weights, resulting in the most consistent performance. In the delete experiment, our method generally performed well due to its ability to predict new edges. In most cases, our method performed optimally, exhibiting stability against structural attacks. However, the local delete experiment showed only moderate performance, which may be due to the lack of implemented sampling strategies and improved aggregate functions. These issues can be addressed in future work. In summary, our method demonstrated strong stability and robustness against structural disturbance attacks.



(**a**) Global Add (**b**) Global Flip (**c**) Global Delete

Figure 3. *Cont.*



(**d**) Community Add (**e**) Community Flip (**f**) Community Delete

(**g**) Local Add (**h**) Local Flip (**i**) Local Delete

Figure 3. Overall performance of various models under various levels of structural noise. (**a**–**c**) depict global noise, (**d**–**f**) depict community noise, while (**g**–**i**) depict local noise. On the Cora dataset, we randomly delete, flip, or add edges to the original graph to create a corrupted graph and evaluate its performance. We set the ratio of modified edges to simulate attack intensities ranging from 0.05 to 0.8.

4.4. Performance on Corrupted Data and Ablation Studies (Q3)

In this study, we examine the effect of our model's weights recalculating, connections discovering, and connections eliminating on performance on the Cora dataset, using a structural noise value of 0.5 and the local removal settings from our previous experiment (Section 4.3). We also report the classification accuracy of nodes under various hyperparameter configurations, and the results are presented in Table 4. A more comprehensive analysis and experimental outcomes can be found in Appendix B.4 and Table A4.

Table 4. Node classification mean accuracy (%) ± stdev for different components (we utilize a two-layer VN-GSL model, where the graph learned by the first layer is denoted as (1) and the graph learned by the second layer is denoted as (2)).

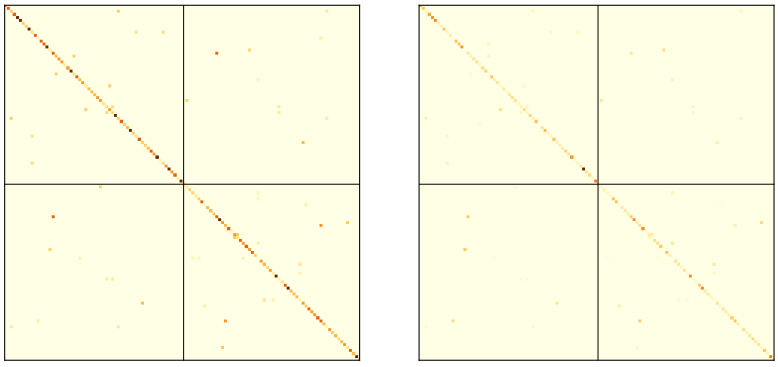
	Model Components											
Models	**Reweight**	**Add Edges**	**Mask Edges**	**Acc**	τ	λ_1	λ_2	p	**Init Edges**	**Added Edges**	**Masked Edges**	**Final Edges**
VN-GSL	×	×	×	76.47 ± 0.54	×	×	×	7	8814	0	0	8814
	✓	×	×	77.94 ± 0.36	1	×	×	7	8814	0	0	8814
	✓	✓	×	77.44 ± 0.39	1	0.5	×	7	8814	284 (1) 279 (2)	0	9098 (1) 9093 (2)
	✓	✓	×	78.48 ± 0.41	1	0.7	×	7	8814	139 (1) 138 (2)	0	8953 (1) 8952 (2)
	✓	✓	×	79.01 ± 0.35	1	0.85	×	7	8814	58 (1) 59 (2)	0	8872 (1) 8873 (2)
	✓	✓	✓	79.03 ± 0.41	1	0.85	0.01	7	8814	58 (1) 56 (2)	171 (1) 165 (2)	8701 (1) 8705 (2)
	✓	✓	✓	$\mathbf{79.19 \pm 0.34}$	1	0.85	0.02	7	8814	58 (1) 56 (2)	223 (1) 227 (2)	8649 (1) 8643 (2)
	✓	✓	✓	78.36 ± 0.36	1	0.85	0.03	7	8814	58 (1) 56 (2)	314 (1) 309 (2)	8558 (1) 8561 (2)

Based on the results presented in Table 4, it can be observed that the weights recalculating, connections discovering, and connections eliminating components of our model have a significant effect on the model's performance on the corrupted Cora dataset. The model exhibits improved mean accuracy compared to a model without any of these components, achieving 79.03% mean accuracy versus 76.47%. Furthermore, the results suggest that an increase in the value of the hyperparameter λ_1 for connections discovery leads to improved performance, with the highest mean accuracy achieved at a value of $\lambda_1 = 0.85$. On the other hand, a lower value of λ_2 for connections eliminating results in better performance, with the highest mean accuracy achieved at $\lambda_2 = 0.01$ to $\lambda_2 = 0.02$.

Our findings indicate that weight recalculation has a significant impact on model performance, suggesting that the original graph weights are suboptimal. Additionally, we observed that augmenting connections to a certain extent can improve model performance, demonstrating the efficacy of our approach in identifying relevant connections. However, as previously noted in [45], the presence of certain detrimental nodes can significantly impair model performance. In our experiments, a reduction in λ_1 values and the concomitant incorporation of a large number of edges resulted in a decline in model performance. Identifying and eliminating these harmful nodes is a focus of future work. We also observed that, to an extent, pruning low-weight connections can enhance model performance. However, excessively pruning connections via an increase in λ_2 values can be detrimental to model performance, consistent with experience and as evidenced by our results. In summary, the weight recalculation and connection discovery and pruning components of our model resulted in improved performance on the corrupted Cora dataset, with the optimal settings for the hyperparameters are task-dependent.

4.5. Visualization (Q4)

To investigate the graph structures learned by VN-GSL, a subgraph consisting of two classes of nodes from the Cora dataset is selected and the edge weights of the original graph and the graph inferred by our proposed method are visualized. As illustrated in Figure 4, the utilization of the original graph as a prior resulted in the learned graph structure exhibiting minimal deviation from the original graph. However, our method recalculated the edge weights and effectively retained within-class edges while reducing the number of inter-class edges, and decreasing the weight of numerous inter-class edges. Based on our analysis, it can be inferred that our proposed method has the capability of enhancing connections between nodes of the same class, while diminishing connections between nodes of different classes, thus improving the quality of the graph topology to a certain extent.



(**a**) Original (**b**) VN-GSL

Figure 4. The adjacency matrix heatmap (with two categories of 60 nodes each extracted from the Cora dataset) (**a**) the original graph with self-loops on the Cora dataset, and (**b**) the graph learned by our method on the Cora dataset. The darker blocks represent a greater edge weight between the two nodes.

5. Conclusions

We introduce a novel approach to graph structure learning that effectively addresses the challenge of impaired graph representations due to noise in real-world scenarios. Our method incorporates edge discovery and removal, reweighting of existing edges, and graph structure derivability, as well as virtual node introduction for optimized graph structure generation. Extensive evaluations on diverse benchmark datasets, including clean and adversarial conditions, demonstrate the superior performance and efficiency of our approach. Our experiments reveal that our approach demonstrates notable efficacy in the areas of reweighting graph structures and eliminating redundant connections. However, we observed limitations in discovering new connections, which is a significant challenge in graph structure learning. This underscores the need for further investigation and optimization in this area as a future direction for our work.

Author Contributions: Methodology, W.Z.; Software, W.L.; Validation, J.G.; Formal analysis, W.X.; Resources, B.L.; Supervision, W.O. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (No.61962010, No.62262005, No.61976107), Qiankehezhicheng[2023]Yiban197.

Data Availability Statement: Cora: This dataset used in this study is openly available in the Laboratory for Web Algorithmics (LINQS) repository at the University of California, Santa Cruz, via the following link: https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz; Citeseer: This dataset analyzed in this study is publicly available and can be accessed via the following link: https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz; Pubmed: This dataset used in this study is openly available in the LINQS repository at the University of California, Santa Cruz, via the following link: https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz; Chameleon and Squirrel: The datasets used in this study were obtained from the Stanford Large Network Dataset Collection. Due to restrictions, the availability of these data is limited. They can be accessed from the authors at http://snap.stanford.edu/data/article-datasets.html with the permission of the third party. Please refer to the following links for access to the Chameleon and Squirrel datasets, respectively: http://snap.stanford.edu/data/chameleon-2008.html and http://snap.stanford.edu/data/squirrel-financial.html.

Acknowledgments: Weihua Ou is the corresponding author. We acknowledge the support from the National Natural Science Foundation of China.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Notations

The symbols commonly used in this paper are listed in Table A1.

Table A1. Frequently used notations.

Notation	Description
$\mathcal{G} = (\mathbf{A}, \mathbf{X})$	The original graph.
$\mathcal{G}' = (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{X})$	The extended graph with virtual nodes.
$\mathbf{A}$	The original adjacency matrix.
$\mathbf{S}$	The sketched adjacency matrix.
$\tilde{\mathbf{A}}$	The learned adjacency matrix.
$\mathbf{A}_{dis}$	Adjacency matrix of newly discovered connection.
$\mathbf{A}_{rec}$	Adjacency matrix of recalculating weights.

Appendix B. More on Experiments

Appendix B.1. Hyperparameter

Table A2 presents the hyperparameters used for model selection. The table includes the number of layers, dropout rate, patience for early stopping, learning rate, optimizer, and the number of epochs for each dataset. The results indicate that the hyperparameter settings varied across the datasets. The Cora and Pubmed datasets utilized three layers and learning rates of 1×10^{-2} and 5×10^{-2}, respectively. The Citeseer and Squirrel datasets employed higher numbers of layers and lower learning rates of 2×10^{-2} and 2×10^{-2}, respectively. The dropout rate ranged from 0.5 to 0.6, with the Citeseer and Pubmed datasets using the same dropout rate of 0.5. The patience for early stopping ranged from 1000 to 2000 epochs, with the Chameleon and Squirrel datasets using the highest value. The optimizers used were Adam and SGD.

Appendix B.2. p Sizes

Tables A3 present the results of node classification on two datasets, Cora and Chameleon, for different parameter values. The mean accuracy and standard deviation are reported for each combination of parameter values. In Table A3, the parameter p is varied. The highest mean accuracy on both datasets is achieved for $p = 7$ on Cora and $p = 4$ on Chameleon. For other values of p, the mean accuracy is generally lower, with a few exceptions where the mean accuracy is similar to the highest mean accuracy. This suggests that the optimal model performance is achieved when the model is neither under-fitting nor over-fitting the data. In contrast, when the mean accuracy is significantly lower, the model may be suffering from either under-fitting or over-fitting.

Appendix B.3. Efficiency

Table 3 compares the efficiency of three graph neural network models (GCN, GAT, and VN-GSL) on two datasets (Cora and Citeseer). The table reports three measures of efficiency: runtime per epoch, maximum memory usage, and minimum memory usage. The runtime per epoch is expressed in seconds and the memory usage is expressed in megabytes. The table also reports the average values for each measure of efficiency. The results show that the VN-GSL model demonstrates a lower runtime per epoch and lower memory usage than the GAT models on both datasets. Furthermore, the table reveals that the GCN model exhibits the lowest runtime per epoch and the lowest memory usage on both datasets. These findings suggest that the VN-GSL model is more efficient than the GAT model and comparable to the GCN model in terms of computational cost and memory consumption.

Table A2. Hyper-parameters used for model selection.

	Layers	τ	p **Size**	**Dropout**	**Patience**	lr	**Optimizer**	**Epochs**	λ_1	λ_2
Cora	2	1	7	0.55	1500	1×10^{-2}	Adam	3000	0.98	0.01
Citeseer	4	1	5	0.5	1000	2×10^{-2}	Adam	2500	0.95	0.01
Pubmed	2	1	3	0.5	1000	5×10^{-2}	Adam	3000	0.98	0.005
Chameleon	2	1	4	0.6	2000	1×10^{-2}	SGD	2000	0.95	0.01
Squirrel	3	1	4	0.5	1500	2×10^{-2}	Adam	2500	0.95	0.005

Table A3. Node classification mean accuracy (%) $\pm$ stdev for different p sizes.

Dataset	p **Size**							
	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$	$p = 9$	$p = 10$
Cora	84.13 ± 0.31	84.12 ± 0.43	84.31 ± 0.62	84.49 ± 0.52	$\mathbf{84.68 \pm 0.51}$	84.62 ± 0.39	84.41 ± 0.56	84.49 ± 0.63
Chameleon	49.74 ± 0.63	$\mathbf{49.99 \pm 0.91}$	49.36 ± 1.51	49.54 ± 0.91	49.62 ± 0.79	49.51 ± 0.64	49.69 ± 0.97	49.25 ± 1.52

Table A4. Node classification mean accuracy (%) ± stdev for different components (we utilize a two-layer VN-GSL model, where the graph learned by the first layer is denoted as (1) and the graph learned by the second layer is denoted as (2)).

	Model Components											
Models	**Reweight**	**Add Edges**	**Mask Edges**	**Acc**	τ	λ_1	λ_2	p	**Init Edges**	**Added Edges**	**Masked Edges**	**Final Edges**
VN-GSL	×	×	×	76.47 ± 0.54	×	×	×	7	8814	0	0	8814
	✓	×	×	77.94 ± 0.36	1	×	×	7	8814	0	0	8814
	✓	✓	×	77.44 ± 0.39	1	0.5	×	7	8814	284 (1) 279 (2)	0	9098 (1) 9093 (2)
	✓	✓	×	78.48 ± 0.41	1	0.7	×	7	8814	139 (1) 138 (2)	0	8953 (1) 8952 (2)
	✓	✓	×	78.61 ± 0.38	1	0.8	×	7	8814	81 (1) 85 (2)	0	8895 (1) 8899 (2)
	✓	✓	×	79.01 ± 0.35	1	0.85	×	7	8814	58 (1) 59 (2)	0	8872 (1) 8873 (2)
	✓	✓	×	78.52 ± 0.39	1	0.9	×	7	8814	30 (1) 28 (2)	0	8844 (1) 8842 (2)
	✓	✓	×	77.94 ± 0.36	1	1	×	7	8814	0	0	8814
	✓	✓	✓	79.03 ± 0.41	1	0.85	0.01	7	8814	58 (1) 56 (2)	171 (1) 165 (2)	8701 (1) 8705 (2)
	✓	✓	✓	79.06 ± 0.45	1	0.85	0.015	7	8814	58 (1) 56 (2)	198 (1) 201 (2)	8674 (1) 8669 (2)
	✓	✓	✓	$\mathbf{79.19 \pm 0.34}$	1	0.85	0.02	7	8814	58 (1) 56 (2)	223 (1) 227 (2)	8649 (1) 8643 (2)
	✓	✓	✓	79.06 ± 0.45	1	0.85	0.025	7	8814	58 (1) 56 (2)	244 (1) 247 (2)	8628 (1) 8623 (2)
	✓	✓	✓	78.36 ± 0.36	1	0.85	0.03	7	8814	58 (1) 56 (2)	314 (1) 309 (2)	8558 (1) 8561 (2)
	✓	✓	✓	77.06 ± 0.55	1	0.85	0.05	7	8814	58 (1) 56 (2)	719 (1) 727 (2)	8153 (1) 8143 (2)

Appendix B.4. More on Q3

The data presented in Table A4 suggests that the implementation of weight recalculation, connection discovery, and connection elimination components in our model yields a significant improvement in performance on the corrupted Cora dataset. The model demonstrates an increase in mean accuracy, attaining 79.19% as opposed to the mean accuracy of 76.47% observed in a model without these components. Additionally, the results imply that an increase in the value of the hyperparameter λ_1 for connection discovery results in improved performance, with the highest mean accuracy achieved at $\lambda_1 = 0.85$. In contrast, a lower value of λ_2 for connection elimination leads to better performance.

Furthermore, our findings suggest that augmenting connections to a certain extent can enhance the model's performance, which supports the effectiveness of our approach in identifying relevant connections. Our experiments indicate that a reduction in λ_1 values and the concomitant incorporation of a large number of edges lead to a decline in performance. Additionally, our results suggest that pruning low-weight connections can enhance the model's performance to a certain extent. However, excessively pruning connections through an increase in λ_2 values can negatively impact performance, as is

evident in our results. In summary, the implementation of weight recalculation, connection discovery, and elimination components in our model improves the performance on the corrupted Cora dataset. The optimal settings for the hyperparameters are task-dependent and require further investigation.

References

1. Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R.P. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 2224–2232.
2. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1263–1272.
3. Kawahara, J.; Brown, C.J.; Miller, S.P.; Booth, B.G.; Chau, V.; Grunau, R.E.; Zwicker, J.G.; Hamarneh, G. BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment. *NeuroImage* **2017**, *146*, 1038–1049. [CrossRef]
4. Zhao, T.; Zhang, X.; Wang, S. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. In Proceedings of the WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Jerusalem, Israel, 8–12 March 2021; pp. 833–841. [CrossRef]
5. Dai, E.; Wang, S. Say No to the Discrimination: Learning Fair Graph Neural Networks with Limited Sensitive Attribute Information. In Proceedings of the WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Jerusalem, Israel, 8–12 March 2021; pp. 680–688. [CrossRef]
6. Wang, J.; Zhang, S.; Xiao, Y.; Song, R. A Review on Graph Neural Network Methods in Financial Applications. *arXiv* **2021**, arXiv:2111.15367.
7. Chen, Y.; Wu, L.; Zaki, M.J. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. *arXiv* **2020**, arXiv:2006.13009.
8. Wu, X.; Zhao, L.; Akoglu, L. A Quest for Structure: Jointly Learning the Graph Structure and Semi-Supervised Classification. *arXiv* **2019**, arXiv:1909.12385.
9. Jiang, B.; Zhang, Z.; Lin, D.; Tang, J.; Luo, B. Semi-Supervised Learning With Graph Learning-Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 11313–11320. [CrossRef]
10. Cosmo, L.; Kazi, A.; Ahmadi, S.; Navab, N.; Bronstein, M.M. Latent Patient Network Learning for Automatic Diagnosis. *arXiv* **2020**, arXiv:2003.13620.
11. Franceschi, L.; Niepert, M.; Pontil, M.; He, X. Learning Discrete Structures for Graph Neural Networks. *arXiv* **2019**, arXiv:1903.11960.
12. Elinas, P.; Bonilla, E.V.; Tiao, L.C. Variational Inference for Graph Convolutional Networks in the Absence of Graph Data and Adversarial Settings. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 18648–18660.
13. Sun, Q.; Li, J.; Peng, H.; Wu, J.; Fu, X.; Ji, C.; Yu, P.S. Graph Structure Learning with Variational Information Bottleneck. In Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence, Online, 22 February–1 March 2022; pp. 4165–4174.
14. Zhang, Y.; Pal, S.; Coates, M.; Üstebay, D. Bayesian Graph Convolutional Neural Networks for Semi-Supervised Classification. In Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 5829–5836. [CrossRef]
15. Jin, W.; Ma, Y.; Liu, X.; Tang, X.; Wang, S.; Tang, J. Graph Structure Learning for Robust Graph Neural Networks. *arXiv* **2020**, arXiv:2005.10203.
16. Bruna, J.; Zaremba, W.; Szlam, A.; LeCun, Y. Spectral Networks and Locally Connected Networks on Graphs. In Proceedings of the 2nd International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
17. Defferrard, M.; Bresson, X.; Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3837–3845.
18. Hamilton, W.L.; Ying, Z.; Leskovec, J. Inductive Representation Learning on Large Graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1024–1034.
19. Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
20. Brody, S.; Alon, U.; Yahav, E. How Attentive are Graph Attention Networks? In Proceedings of the the Tenth International Conference on Learning Representations, Virtually, 25–29 April 2022.
21. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, *32*, 4–24. [CrossRef]
22. Eden, T.; Jain, S.; Pinar, A.; Ron, D.; Seshadhri, C. Provable and practical approximations for the degree distribution using sublinear graph samples. *arXiv* **2017**, arXiv:1710.08607.
23. Chew, P. There are Planar Graphs Almost as Good as the Complete Graph. *J. Comput. Syst. Sci.* **1989**, *39*, 205–219. [CrossRef]
24. Benczúr, A.A.; Karger, D.R. Approximating *s-t* Minimum Cuts in $\tilde{O}(n^2)$ Time. In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 47–55. [CrossRef]

25. Hermsdorff, G.B.; Gunderson, L.M. A Unifying Framework for Spectrum-Preserving Graph Sparsification and Coarsening. *arXiv* **2019**, arXiv:1902.09702.
26. Arora, R.; Upadhyay, J. On Differentially Private Graph Sparsification and Applications. *Adv. Neural Inf. Process.* **2019**, *32*, 13378–13389.
27. Zheng, C.; Zong, B.; Cheng, W.; Song, D.; Ni, J.; Yu, W.; Chen, H.; Wang, W. Robust Graph Representation Learning via Neural Sparsification. *Proc. Mach. Learn. Res.* **2020**, *119*, 11458–11468.
28. Liu, M.; Wang, Z.; Ji, S. Non-Local Graph Neural Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 10270–10276. [CrossRef]
29. Paulus, M.; Choi, D.; Tarlow, D.; Krause, A.; Maddison, C.J. Gradient estimation with stochastic softmax tricks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 5691–5704.
30. Niepert, M.; Minervini, P.; Franceschi, L. Implicit MLE: Backpropagating Through Discrete Exponential Family Distributions. *Adv. Neural Inf. Process.* **2021**, *34*, 14567–14579.
31. Jang, E.; Gu, S.; Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv* **2016**, arXiv:1611.01144.
32. Maddison, C.J.; Mnih, A.; Teh, Y.W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv* **2016**, arXiv:1611.00712.
33. Rozemberczki, B.; Allen, C.; Sarkar, R. Multi-Scale attributed node embedding. *J. Complex Netw.* **2021**, *9*, cnab014. [CrossRef]
34. Pei, H.; Wei, B.; Chang, K.C.; Lei, Y.; Yang, B. Geom-GCN: Geometric Graph Convolutional Networks. In Proceedings of the 8th International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
35. Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Gallagher, B.; Eliassi-Rad, T. Collective Classification in Network Data. *AI Mag.* **2008**, *29*, 93–106. [CrossRef]
36. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
37. Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; Leskovec, J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Adv. Neural Inf. Process.* **2020**, *33*, 22118–22133
38. Wang, R.; Mou, S.; Wang, X.; Xiao, W.; Ju, Q.; Shi, C.; Xie, X. Graph Structure Estimation Neural Networks. In Proceedings of the WWW '21: The Web Conference 2021, Ljubljana, Slovenia, 19–23 April 2021; pp. 342–353. [CrossRef]
39. Wu, F.; Zhang,, T.; de Souza, A.H., Jr.; Fifty, C.; Yu, T.; Weinberger, K.Q. Simplifying Graph Convolutional Networks. *Proc. Mach. Learn. Res.* **2019**, *97*, 6861–6871.
40. Klicpera, J.; Bojchevski, A.; Günnemann, S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
41. Fey, M.; Lenssen, J.E. Fast Graph Representation Learning with PyTorch Geometric. *arXiv* **2019**, arXiv:1903.02428.
42. Zhang, Z.; Pei, Y. A Comparative Study on Robust Graph Neural Networks to Structural Noises. *arXiv* **2021**, arXiv:2112.06070.
43. Fortunato, S. Community detection in graphs. *arXiv* **2009**, arXiv:0906.0612.
44. Chen, J.; Ma, T.; Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *arXiv* **2018**, arXiv:1801.10247.
45. Chen, L.; Li, J.; Peng, Q.; Liu, Y.; Zheng, Z.; Yang, C. Understanding Structural Vulnerability in Graph Convolutional Networks. *arXiv* **2021**, arXiv:2108.06280.