

DEGNN: Dual Experts Graph Neural Network Handling Both Edge and Node Feature Noise

Tai Hasegawa^{1,2}[0000-0003-4089-0186], Sukwon Yun³[0000-0002-5186-6563], Xin Liu²[0000-0002-2336-7409]✉, Yin Jun Phua¹[0000-0003-1178-8238], and Tsuyoshi Murata^{1,2}[0000-0002-3818-7830]

¹ Department of Computer Science, Tokyo Institute of Technology, Japan

{hasegawa.t@net, phua@, murata@}.c.titech.ac.jp

² Artificial Intelligence Research Center, AIST, Japan

✉xin.liu@aist.go.jp

³ Industrial and Systems Engineering, KAIST, Republic of Korea

swyun@kaist.ac.kr

Abstract. Graph Neural Networks (GNNs) have achieved notable success in various applications over graph data. However, recent research has revealed that real-world graphs often contain noise, and GNNs are susceptible to noise in the graph. To address this issue, several Graph Structure Learning (GSL) models have been introduced. While GSL models are tailored to enhance robustness against edge noise through edge reconstruction, a significant limitation surfaces: their high reliance on node features. This inherent dependence amplifies their susceptibility to noise within node features. Recognizing this vulnerability, we present DEGNN, a novel GNN model designed to adeptly mitigate noise in both edges and node features. The core idea of DEGNN is to design two separate experts: an edge expert and a node feature expert. These experts utilize self-supervised learning techniques to produce modified edges and node features. Leveraging these modified representations, DEGNN subsequently addresses downstream tasks, ensuring robustness against noise present in both edges and node features of real-world graphs. Notably, the modification process can be trained end-to-end, empowering DEGNN to adjust dynamically and achieves optimal edge and node representations for specific tasks. Comprehensive experiments demonstrate DEGNN’s efficacy in managing noise, both in original real-world graphs and in graphs with synthetic noise.

Keywords: Graph Neural Networks · Graph Structure Learning · Graph Self-Supervised Learning.

1 Introduction

Graphs are essential data structures for modeling a wide range of real-world phenomena, such as social networks, transportation networks, and chemical molecules. Graph Neural Networks (GNNs) have emerged as a powerful paradigm for modeling such graphs, primarily due to their message-passing mechanism that aggregates node representations via edges. These GNNs can be applied to various tasks, including node classification [11,24], link prediction [37], node ranking [7,3], community detection [29], and graph classification [4].

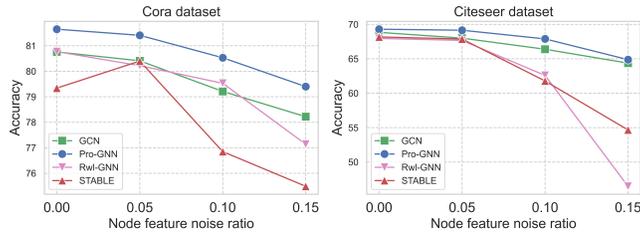


Fig. 1. A comparison of semi-supervised node classification between GCN and GSL models on Cora and Citeseer dataset when there is noise in the node features.

Despite their success, it is well known that the quality of real-world graph data is often unreliable [8]. In other words, real-world graphs are known to contain noise. For instance, in citation networks, references to unrelated papers can introduce noise in the form of inaccurate edges. Recent studies on adversarial attacks and defenses have highlighted the susceptibility of GNNs to noise within graphs [9,10]. In response, Graph Structure Learning (GSL) has been developed as a method to optimize graph structure, thus improving graph representations and ensuring more resilient predictions amidst edge noise.

Besides edge noise, node features can also contain noise. For instance, in social networks, users might provide inconsistent, overstated, or even false information about their interests or attributes, which introduces noise into the node features. GSL models often struggle with such node feature noises. The vulnerability of GSL models against node feature noise arises as these models disseminate noisy node features via message passing scheme. Additionally, their reliance on node features to rewire edges compounds the challenge. As illustrated in Figure 1 (refer to Section 4 for the experiment details), recent GSL models such as Pro-GNN [18], Rwl-GNN [35], and STABLE [20] tend to underperform in prediction accuracy when compared to traditional GNN model, GCN [11] when the node feature noise increases. Although Pro-GNN marginally surpasses the performance of GCN, there remains ample scope for improvements.

In this paper, we introduce Dual Experts Graph Neural Network (DEGNN)⁴, a novel GNN model crafted to offer robust predictions irrespective of the presence or absence of noise in edges, nodes, or both. At the heart of DEGNN is its distinctive architecture, which employs specialized branches, termed "experts", to individually learn and refine node features and edges. Using a self-supervised learning approach, these experts are seamlessly integrated and co-trained end-to-end, ensuring task-specific optimization. Our contributions are summarized as follows:

- We highlight the susceptibility of GSL models to node feature noise based on preliminary experiments.
- We introduce DEGNN, a novel GNN that offers robust predictions irrespective of the presence or absence of noise in nodes, edges, or both, by individually addressing these noises through a self-supervised learning approach.

⁴ Codes are available at: <https://github.com/TaiHasegawa/DEGNN>

- Through comprehensive experiments on real-world datasets, we establish that DEGNN consistently delivers stable predictions, outperforming state-of-the-art models in the presence of either type of noise.

2 Related Work

2.1 Graph Neural Networks

GNNs have emerged as a powerful tool for learning from graph-structured data, effectively capturing the complex relationships and interdependencies between nodes [1]. Their successful development across numerous practical fields underscores their extensive applicability and effectiveness [26,27,28,16,17]. Generally, GNNs can be classified into two categories: spectral-based methods [6,11] and spatial-based methods [12,13]. Spectral-based GNNs hinge upon spectral graph theory [5] and employ spectral convolutional neural networks. To streamline the intricacies of spectral-based GNNs, various techniques have emerged, including ChebNet [6] and GCN [11].

On the other hand, spatial-based GNNs are engineered to tackle challenges related to efficiency, generality, and flexibility, as highlighted in [15]. They achieve graph convolution operation through neighborhood aggregation. For instance, GraphSAGE [12] selectively samples a subset of neighbors to grasp local features, while GAT [13] employs an attention mechanism for adaptive neighbor aggregation. However, these models are susceptible to edge noise.

2.2 Graph Structure Learning

The purpose of GSL is to improve prediction accuracy by modifying the given graph into an optimal structure. In this paper, we primarily focus on GNN-based graph structure learning models. LDS [14] jointly optimizes the probability for each node pair and the parameters of GNNs in a bilevel way. Pro-GNN [18] aims to learn the optimal graph structure by incorporating several regularizations, such as low-rank sparsity and feature smoothness. Gaug-M [19] directly computes the edge weights by taking the inner product of node embeddings. STABLE [20] utilizes self-supervised learning to acquire node embeddings and then modifies the graph structure based on their similarities. These obtained node embeddings and the modified graph structure are employed in downstream tasks. Each of these models demonstrates the ability to robustly predict against edge noise, as shown in their respective papers. However, as elucidated in Section 1, their pronounced reliance on node features during edge rewiring inherently exposes them to vulnerabilities in situations characterized by noisy node features.

3 The Proposed Model

In this section, we introduce our proposed approach, DEGNN. We begin by formulating the problem definition, followed by an overview of the model, and then provide a detailed description of its architecture and learning procedure.

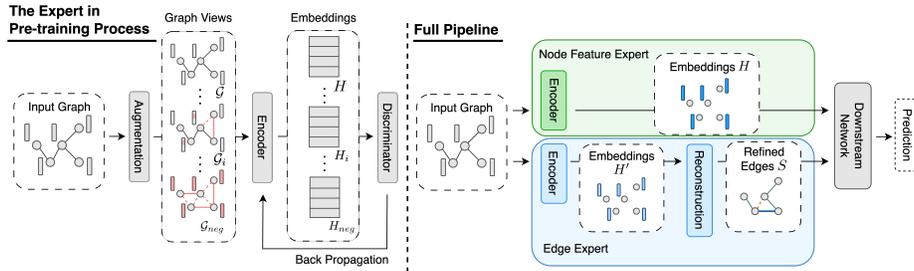


Fig. 2. The overview of DEGNN and the expert in its pre-training process.

3.1 Problem Definition

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, X\}$ represent an undirected graph, where $\mathcal{V} = \{v_i\}_{i=1}^N$ is the set of N nodes, \mathcal{E} is the set of edges, $X = [x_1, \dots, x_N] \in \mathbb{R}^{N \times D}$ indicates the node feature matrix and each $x_i \in \mathbb{R}^D$ is the feature vector of node v_i . The set of edges is represented by an adjacency matrix $A \in \{0, 1\}^{N \times N}$, where A_{ij} denotes the connection between nodes v_i and v_j . Following the common semi-supervised node classification setting, only a small portion of nodes $\mathcal{V}_L = \{v_i\}_{i=1}^l$ are associated with the corresponding labels $\mathcal{Y}_L = \{y_i\}_{i=1}^l$ while the rest of the nodes $\mathcal{V}_U = \{v_i\}_{i=l+1}^N$ are unlabeled.

Given graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, X\}$ and the available labels \mathcal{Y}_L , the goal of graph structure learning, aimed at refining node features and graph structure, is to learn optimal node embeddings $H \in \mathbb{R}^{N \times D'}$ with hidden dimension D' , a modified adjacency matrix $S \in \mathbb{R}^{N \times N}$, and the GNN parameters θ in order to improve the predictive accuracy of $\hat{\mathcal{Y}}_L$. The objective function can be formulated as

$$\min_{\theta, S, H} \mathcal{L}(A, X, \mathcal{Y}_L) = \sum_{v_i \in \mathcal{V}_L} \ell(f_\theta(H, S)_i, y_i), \quad (1)$$

where $f_\theta : \mathcal{V}_L \rightarrow \mathcal{Y}_L$ is a function learned by downstream GNNs that maps nodes to the set of labels, $f_\theta(H, S)_i$ is the prediction of node v_i , and ℓ is the loss measuring difference between prediction and true label such as cross-entropy.

3.2 Overview

The overview of DEGNN and the expert in its pre-training process are illustrated in Figure 2. The input graph is passed through both the node feature expert and the edge expert. The node feature expert outputs node embeddings H , while the edge expert produces the modified adjacency matrix S . Using the obtained H and S , the downstream network makes predictions for a specific task. Traditionally, models have primarily leaned on learned node embeddings for edge reconstruction [20]. However, when these embeddings are derived from noisy node features, the resulting in reconstructed edges often produce sub-optimal outcomes, potentially that are unintended dependencies. Our proposed

dual experts design aims to eliminate these dependencies and learn both node representations and edges to be optimal. These two experts can be trained with the downstream network end-to-end, allowing each of them to acquire representations suitable for the specific task. The details of the model are described in the following sections.

3.3 Node Feature Expert

A straightforward approach to predict robustly against node feature noise is to generate node embeddings H that capture node features more effectively than the input node features X . To obtain the node embeddings H , we utilize self-supervised learning, since it enables the model to achieve better performance, generalization, and robustness across a wide range of downstream tasks [32].

Graph Augmentation Generating views is a key component of self-supervised learning methods. For instance, in the field of computer vision [21], views are created by rotating or cropping images. This allows for mitigating the impact of differences in angles and scales on the model’s predictions. Similarly, for graph data, we presume that by generating views and exposing the model to modified edges and nodes, it can achieve enhanced generalizability, equipping it with a stronger capacity to manage noise emanating from both edges and nodes during predictions. Therefore, we generate graphs of positive view $\mathcal{G}_1 = (A, X)$ with noise added to the edges, $\mathcal{G}_2 = (A, \tilde{X})$ with noise added to the node features, and $\mathcal{G}_3 = (\tilde{A}, \tilde{X})$ with noise added to both edges and node features, where $\tilde{A} \in \{0, 1\}^{N \times N}$ and $\tilde{X} \in \mathbb{R}^{N \times D}$ denote the noisy adjacency matrix and noisy node features, respectively.

Our proposed model augments edges by randomly rewiring them. Formally, we first create a mask matrix $P \in \{0, 1\}^{N \times N}$ to rewire edges, where P is obtained from a Bernoulli distribution $P_{ij} \sim \mathcal{B}(p)$ with a hyper-parameter p that controls the rewiring probability. And then, \tilde{A} is calculated as follows:

$$\tilde{A} = (1 - A) \odot P + A \odot (1 - P), \quad (2)$$

where \odot is the Hadamard product. For node feature augmentation, unlike other studies that use node feature masking [22,23], in this paper, we shuffle elements in each row of X randomly with the probability q to replicate scenarios where there is noise in the node features.

Besides positive graphs, we generate a negative graph $\mathcal{G}_{neg} = (A_{neg}, X_{neg})$ that is entirely different from the original graph \mathcal{G} , where $A_{neg} \in \{0, 1\}^{N \times N}$ and $X_{neg} \in \mathbb{R}^{N \times D}$ denote the negative adjacency matrix and negative node features, respectively. A_{neg} is given by $A_{neg} = (1 - A) \odot P_{neg}$, where $P_{neg} \in \{0, 1\}^{N \times N}$ is a mask matrix that is obtained from a Bernoulli distribution $P_{negij} \sim \mathcal{B}(\frac{\|\mathcal{E}\|}{N^2 - \|\mathcal{E}\|})$. X_{neg} is obtained through row-wise shuffling of X .

Encoder The encoder f_ϕ parameterized by ϕ learns embeddings for each of the generated views. In this paper, we use a 1-layer GCN [11] as the encoder, which is formulated as follows:

$$f_\phi(X, A) = \sigma(\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} X W), \quad (3)$$

where $\hat{A} = A + I_N$, $\hat{D} = D + I_N$, D is the degree matrix of A , I_N is the identity matrix, σ is a non-linear activation function and W is weight matrix transforming raw features to embeddings. The embeddings for the original graph and each view $H, H_1, H_2, H_3, H_{neg} \in \mathbb{R}^{N \times D'}$ are obtained using the encoder as follows:

$$H = f_\phi(X, A) \quad (4) \quad H_2 = f_\phi(\tilde{X}, A) \quad (6)$$

$$H_3 = f_\phi(\tilde{X}, \tilde{A}) \quad (7)$$

$$H_1 = f_\phi(X, \tilde{A}) \quad (5) \quad H_{neg} = f_\phi(X_{neg}, A_{neg}) \quad (8)$$

Objective Function Following other graph contrastive learning methods [31,20], we train the encoder to maximize the mutual information between the original graph \mathcal{G} and positive graphs $\mathcal{G}_1, \mathcal{G}_2$ and \mathcal{G}_3 while minimizing agreement between original graph \mathcal{G} and negative graph \mathcal{G}_{neg} . Formally, the objective function for node feature expert can be formulated via using binary cross-entropy loss between positive samples and negative samples as follows:

$$\mathcal{L}_N = -\frac{1}{2N} \sum_{i=1}^N \left(\frac{1}{3} \sum_{j=1}^3 (\log \mathcal{D}(h_i, h_i^j) + \log(1 - \mathcal{D}(h_i, h_i^{neg}))) \right), \quad (9)$$

where h_i, h_i^j, h_i^{neg} are the embeddings of node v_i in $\mathcal{G}, \mathcal{G}_j$ and \mathcal{G}_{neg} , and \mathcal{D} is the discriminator built upon an inner product, i.e., $\mathcal{D}(a, b) = ab^T$. In conclusion, by training the node feature expert, f_ϕ , we derive a noise-robust node embedding H that subsequently serves as input for the downstream network.

3.4 Edge Expert

To differentiate between the impacts of node feature noise and edge noise, and to adeptly address each situation, we further implement an edge expert. The edge expert learns node embeddings H' using self-supervised learning as with the node feature expert. More precisely, H' is obtained through an encoder f_ψ with different parameters from the node feature expert f_ϕ , and the objective function for edge expert using the binary cross-entropy loss L_E can be formulated as follows:

$$\mathcal{L}_E = -\frac{1}{2N} \sum_{i=1}^N \left(\frac{1}{3} \sum_{j=1}^3 (\log \mathcal{D}(h'_i, h'^j_i) + \log(1 - \mathcal{D}(h'_i, h'^{neg'}_i))) \right), \quad (10)$$

where $h'_i, h'^j_i, h'^{neg'}_i$ are the embeddings of node v_i in $\mathcal{G}, \mathcal{G}_j$ and \mathcal{G}_{neg} obtained with encoder f_ψ .

Reconstruction Once the high-quality embedding H' from the edge expert is obtained, it is used to reconstruct the graph structure. This reconstruction process rewires the edges using the pairwise similarity in the embeddings H' under the homophily assumption [2], which posits that nodes with similar features are more likely to be connected.

In the beginning, we compute the cosine similarity matrix $B \in \mathbb{R}^{N \times N}$, where B_{ij} represents the cosine similarity between H'_i and H'_j . Next, we remove the $k\%$ (k is a hyper-parameter) of the edges with the smallest cosine similarity between pairs of nodes from the original edge set \mathcal{E} , resulting in a sparser adjacency matrix $\tilde{S} \in \{0, 1\}^{N \times N}$ as follows:

$$\tilde{S} = A \odot M_1, \quad (11)$$

where $M_1 \in \{0, 1\}^{N \times N}$ represents the mask matrix for edge deletion, where M_{1ij} is 1 if its cosine similarity ranks over the smallest $k\%$, and 0 otherwise.

Finally, to obtain the modified adjacency matrix S , we add the same number of edges that were removed i.e., $k * |\mathcal{E}|$ with the highest cosine similarity between pairs of nodes from the set of node pairs $\mathcal{E}' = V \times V \setminus \mathcal{E}$ that are not included in the edge set \mathcal{E} . Formally, S is obtained as follows:

$$S = \tilde{S} + B \odot M_2, \quad (12)$$

where $M_2 \in \{0, 1\}^{N \times N}$ is the mask matrix for edge addition, where M_{2ij} is 1 for the edges within top cosine similarity ($k * |\mathcal{E}'|$), and 0 otherwise. Here, it is important that the encoder in the edge expert can be trained through backpropagation from the downstream network, so the reconstruction needs to be differentiable. Consequently, the value associated with newly introduced edges in S is not clamped and retains its cosine similarity. To sum up, through training edge expert, f_ψ , we obtain a modified adjacency matrix, S , which is then utilized as input for the downstream network.

3.5 Downstream Network

Once we obtain the node embeddings H and the modified adjacency matrix S , we now use them as input for the downstream network (i.e., GNNs). It is worth noting that our proposed method allows the use of any GNNs such as GCN [11] or GAT [13], and also can be applied to various tasks beyond node classification, including link prediction [37] and graph classification [4]. In this paper, we use a 2-layer GCN f_θ as the downstream network. To tackle node classification, the model is trained to minimize the cross-entropy loss:

$$\mathcal{L}_{GNN} = \sum_{v_i \in V_L} \ell(f_\theta(H, S)_i, y_i), \quad (13)$$

where $\ell(f_\theta(H, S)_i, y_i)$ is the cross-entropy between the prediction and the ground-truth label for node v_i .

3.6 Training Methodology

In this paper, we propose two variants of DEGNN with different training methods: (i) the pre-training and fine-tuning model (referred to as DEGNN-I), and (ii) the modular learning model (referred to as DEGNN-II). DEGNN-I first pre-trains the node feature expert and edge expert separately. Then, all components are jointly fine-tuned in an end-to-end manner. This is to enable each expert to obtain the optimal node embeddings and graph structure for downstream tasks. During the fine-tuning process, it is trained to minimize the following objective function:

$$\mathcal{L} = \mathcal{L}_{GNN} + \alpha\mathcal{L}_N + \beta\mathcal{L}_E, \quad (14)$$

where α and β are hyper-parameters to balance the contributions of node embedding generation and edge reconstruction, respectively.

In contrast, DEGNN-II follows a two-step approach: initially training the node feature expert and edge expert and subsequently freezing them during the training of the downstream network. This methodology empowers each expert to attain task-agnostic representations. This allows robust predictions in contexts with limited labels or large biases, given that the approach does not depend on the provided label during training—essentially, a self-supervised paradigm.

4 Experiments

In this section, we evaluate our proposed method, DEGNN, on a variety of noisy graphs in the context of the semi-supervised node classification task.

4.1 Experimental Setup

Datasets We used four open datasets, including two citation networks (i.e., Cora [19], Citeseer [19]) and two co-purchasing networks (i.e., Photo [33], Computer [33]) Regarding the train/validation/test split, we prepared 20 training labels for each class in all datasets. For validation and test, we used 500 and 1000 nodes, respectively.

Noisy Graphs To assess the robustness of our model across various graph settings with noise, we compared models on graphs that included the following types of noise:

- **Clean Graphs:** The original graphs of the datasets which may contain inherent node feature noise and edge noise.
- **Edge Noisy Graphs:** We randomly remove a certain number of edges and insert the same number of fake edges.
- **Node Feature Noisy Graphs:** As in [36], we added independent Gaussian noise to the node features. Specifically, we obtained the reference amplitude r by calculating the mean of the maximum value across each node’s features. For each feature dimension of each node, we introduced independent Gaussian noise $\lambda \cdot r \cdot \epsilon$, where $\epsilon \sim N(0, 1)$, and λ represents the feature noise ratio.

- **Edge and Node Feature Noisy Graphs:** We introduced both the edge noise and node feature noise described above.

When adding these noises, we employed a poisoning attack, which initially prepares a graph with noise added, and used it for both training and evaluation.

Baselines We compare the proposed DEGNN-I and DEGNN-II with two categories of baselines: classical GNN models (i.e., GCN [11], GAT [13] and RGCN [34]) and graph structure learning methods (i.e., Pro-GNN [18], Rwl-GNN [35] and STABLE [20]).

Implementation Details All hyper-parameters are tuned on the clean graph. All models are trained using Adam optimizer with a default learning rate of 1e-2 and a weight decay of 5e-4 when not explicitly specified. GCN [11], GAT [13], and RGCN [34] have a fixed number of layers at 2. For GCN and RGCN, the hidden dimension is chosen from {16, 32, 64, 128}. GAT’s number of heads and head dimensions are selected from {1, 2, 4, 8, 16} and {8, 16, 32, 64, 128}, respectively, with a total hidden dimension ranging from 16 to 128. Other baselines follow hyper-parameter combinations specified in their respective papers. For DEGNN, α and β are tuned from {0, 0.1, 1.0, 10}, the hidden dimension D' is tuned from {128, 256, 512}. k is searched in {1, 5, 10, 15, 20, 25}, p and q are searched in {0.2, 0.4, 0.6}. The learning rate in the pre-training process is tuned from {1e-2, 5e-3, 1e-3}.

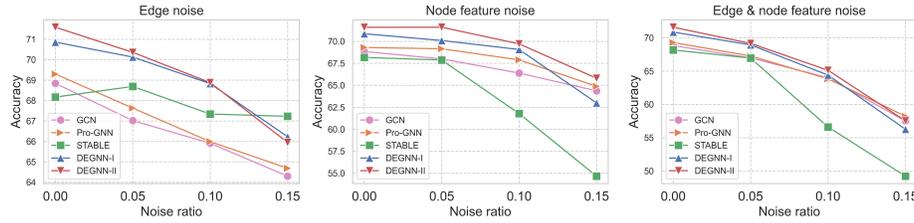
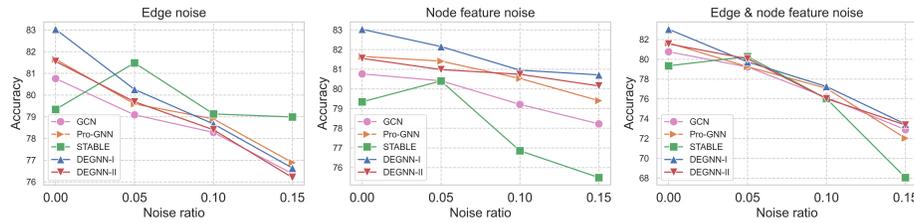
4.2 Semi-supervised Node Classification

Performance Comparison In this section, we evaluate the proposed DEGNN on semi-supervised node classification on original graphs. The average accuracy and standard deviation of the model across 10 runs are summarized in Table 1. OOM indicates out of memory. Based on the experiment, our proposed approach outperformed other models in citation networks (Cora, Citeseer), demonstrating the highest predictive accuracy. This suggests that the experts successfully obtain superior representations for both edges and node features to eliminate latent noise within these graphs. However, in co-purchasing networks (Photo, Computer), our model achieved competitive results but was marginally surpassed by the traditional GCN. Considering that other GSL models also lagged behind GCN in accuracy and the fact that GCN’s predictive accuracy in co-purchasing networks is significantly higher than in citation networks, it is conceivable that the co-purchasing networks may contain less inherent noise within the original graphs, and the GSL models might be unnecessarily altering the graph.

Robustness Evaluation In this experiment, we evaluate the robustness of the models by comparing their performance on graphs with noise added to either or both of the edge and node features. The nature of these noise is described in Section 4.1. Specifically, we added edge noise with noise ratio 0.05, 0.1 and

Table 1. The results (accuracy(%) \pm std) of semi-supervised node classification on clean graphs. The top two performance is highlighted in bold and underline.

Dataset	GCN	GAT	RGCN	Pro-GNN	Rwl-GNN	STABLE	DEGNN-I	DEGNN-II
Cora	80.8 \pm 0.9	79.5 \pm 0.8	79.6 \pm 0.7	<u>81.7 \pm 0.7</u>	80.8 \pm 0.7	79.3 \pm 1.1	83.0 \pm 0.9	81.6 \pm 1.0
Citeseer	68.8 \pm 0.7	66.7 \pm 1.7	65.7 \pm 1.5	69.3 \pm 0.6	68.0 \pm 0.6	68.2 \pm 0.5	<u>70.9 \pm 1.9</u>	71.6 \pm 1.4
Photo	91.8 \pm 0.2	85.5 \pm 17.3	91.7 \pm 0.5	<u>91.8 \pm 0.7</u>	86.4 \pm 2.9	OOM	91.3 \pm 0.4	91.6 \pm 0.6
Computers	85.0 \pm 0.9	77.2 \pm 22.7	81.6 \pm 2.6	OOM	74.6 \pm 1.4	OOM	<u>83.5 \pm 0.9</u>	82.8 \pm 0.8

**Fig. 3.** Accuracy on Citeseer with noise added to either or both of edge and node features.**Fig. 4.** Accuracy on Cora with noise added to either or both of edge and node features.

0.15, while the parameter λ for node feature perturbation is also set to 0.05, 0.1, and 0.15. All experiments were conducted 10 times, and the average accuracy on Citeseer and Cora dataset are shown in Figures 3 and Figure 4, respectively. From this experiment, we can obtain the following observation:

- When node feature noise is added, DEGNN-I or DEGNN-II demonstrated the highest accuracy among the compared models in all settings.
- When noise is introduced in only the edge and in both edge and node features, DEGNN demonstrated the best or competitive results compare to the baselines especially when the noise ratio is 0, 0.05 or 0.1.
- The smallest accuracy gap between edge noise ratio of 0 and 0.15 was observed in STABLE. However, it is highly vulnerable when node features are perturbed.
- GCN and Pro-GNN follow a similar trend, with Pro-GNN slightly outperforming GCN by a small margin. In many settings, their accuracy fell below that of DEGNN.

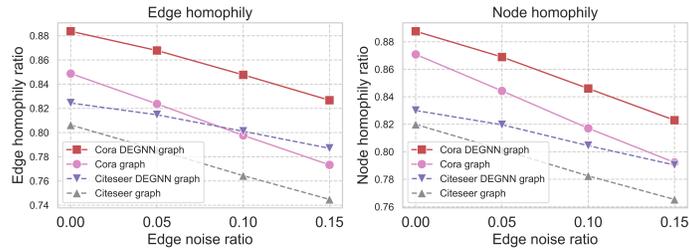


Fig. 5. Comparison of edge homophily ratio and node homophily ratio between noisy graphs and refined graphs by the edge expert on Cora and Citeseer.

4.3 Edge Expert Analysis

To evaluate the edge expert, we compared the edge homophily ratio [25] and node homophily ratio [30] between the noisy graph and the graphs refined by the edge expert (referred to as the DEGNN graph) on various edge noise added settings. In this experiment, we used DEGNN-II, and we set the hyper-parameter k , which controls the number of edge rewires, to 10%. Figure 5 shows the edge homophily ratio and node homophily ratio as edge noise is gradually added. The solid lines represent the results for the Cora dataset, while the dashed lines represent the results for the Citeseer dataset. This experiment reveals that the edge expert consistently promotes high edge homophily and node homophily in all scenarios.

4.4 Node Feature Expert Analysis

In this experiment, we compared the effectiveness of the node feature expert by comparing the models' accuracy when node features have noise added. Figure 6 shows the average of accuracy of 10 runs of GCN and DEGNN-I only having node feature expert (without edge expert) on Cora and Citeseer dataset with added node feature noise. Empirical results confirm that employing node embeddings derived from the node feature expert markedly elevates prediction accuracy across a majority of scenarios. This underscores both the indispensability and efficacy of the node feature expert.

5 Conclusion

In this paper, we identified the vulnerability of recent GSL methods to node feature noise and proposed a novel GNN model, DEGNN, to address this issue. DEGNN refines both edges and node features using two carefully designed experts via self-supervised learning, allowing it to robustly perform predictions in the presence of both edge and node feature noise. Extensive experiments verify the effectiveness of the experts in handling graphs with various noise scenarios.

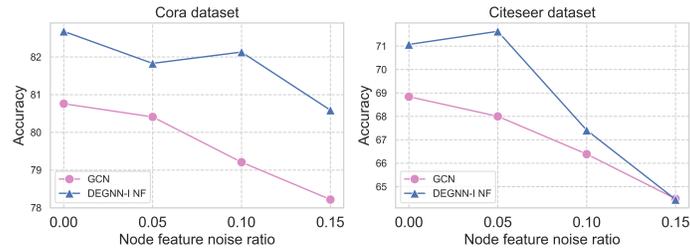


Fig. 6. Comparison of GCN and DEGNN-I without edge expert on Cora and Citeseer with added node feature noise.

In this paper, we employed GCN as both the encoder and downstream network. Additionally, simple methods were used for augmentation and edge reconstruction techniques. As future work, we plan to explore more specialized networks in the encoder and downstream network, as well as investigate different augmentation and edge reconstruction techniques.

Acknowledgements

This work is partly supported by JSPS Grant-in-Aid for Scientific Research (grant number 23H03451, 21K12042) and the New Energy and Industrial Technology Development Organization (Grant Number JPNP20017).

References

1. Zhou, Jie, et al. Graph neural networks: A review of methods and applications. *AI open* 1 (2020): 57-81.
2. McPherson, M., Smith-Lovin, L., & Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1), 415-444.
3. Maurya, S. K., Liu, X., & Murata T. Graph neural networks for fast node ranking approximation. In *TKDD* 2021.
4. Zhang, Muhan, et al. An end-to-end deep learning architecture for graph classification. In *AAAI* 2018.
5. RK Chung Fan and Graham Fan Chung. *Spectral graph theory*. number 92. American Mathematical Soc, 1997.
6. Defferrard, M., Bresson, X., & Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS* 2016.
7. Maurya, S. K., Liu, X., & Murata T. Fast approximations of betweenness centrality with graph neural networks. In *CIKM* 2019.
8. Marsden, P. V. (1990). Network data and measurement. *Annual review of sociology*, 16(1), 435-463.
9. Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., & Song, L. Adversarial attack on graph structured data. In *ICML* 2018.
10. Jin, Wei, et al. Adversarial attacks and defenses on graphs. In *SIGKDD* 2021.
11. Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
12. Hamilton, W., Ying, Z., & Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS* 2017.

13. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In ICLR 2018.
14. Franceschi, L., Niepert, M., Pontil, M., & He, X. Learning discrete structures for graph neural networks. In ICML 2019.
15. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1), 4-24.
16. Jin, R., Xia, T., Liu, X., & Murata, T. (2021). Predicting emergency medical service demand with bipartite graph convolutional networks. *IEEE Access*, 9, 9903-9915.
17. Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., & Yin, D. Graph neural networks for social recommendation. *WWW 2019*, 417-426.
18. Jin, Wei, et al. Graph structure learning for robust graph neural networks. In SIGKDD 2020.
19. Zhao, Tong, et al. Data augmentation for graph neural networks. In AAAI 2021.
20. Li, Kuan, et al. Reliable representations make a stronger defender: Unsupervised structure refinement for robust gnn. In SIGKDD 2022.
21. Berthelot, David, et al. Mixmatch: A holistic approach to semi-supervised learning. In NeurIPS 2019.
22. Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. In ICML 2020.
23. You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., & Shen, Y. Graph contrastive learning with augmentations. In NeurIPS 2020.
24. Maurya, S. K., Liu, X., & Murata T. (2022). Simplifying approach to node classification in graph neural networks. *Journal of Computational Science*, 62, 101695.
25. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., & Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. In NeurIPS 2020.
26. Marcheggiani, D., & Titov, I. Encoding sentences with graph convolutional networks for semantic role labeling. In EMNLP 2017, 1506-1515.
27. Rakhimberdina, Z., Liu, X., & Murata, T. (2020). Population graph-based multi-model ensemble method for diagnosing autism spectrum disorder. *Sensors*, 20(21), 6001.
28. Djenouri, Y., Belhadi, A., Srivastava, G., & Lin, J. C. (2023). Hybrid graph convolution neural network and branch-and-bound optimization for traffic flow forecasting. *Future Generation Computer Systems*, 139, 100-108.
29. Choong, J. J., Liu, X., & Murata, T. Learning community structure with variational autoencoder. In ICDM 2018, 69-78.
30. Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In ICLR 2020.
31. Suresh, S., Li, P., Hao, C., & Neville, J. Adversarial graph augmentation to improve graph contrastive learning. In NeurIPS 2021.
32. Liu, Y., Jin, M., Pan, S., Zhou, C., Zheng, Y., Xia, F., & Philip, S. Y. (2022). Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6), 5879-5900.
33. O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, Pitfalls of graph neural network evaluation, In NeurIPS workshop 2018.
34. Zhu, D., Zhang, Z., Cui, P., & Zhu, W. Robust graph convolutional networks against adversarial attacks. In SIGKDD 2019.
35. Runwal, B., & Kumar, S. (2022). Robust graph neural networks using weighted graph laplacian. arXiv preprint arXiv:2208.01853.
36. Wu, T., Ren, H., Li, P., & Leskovec, J. Graph information bottleneck. In NeurIPS 2020.
37. Zhang, M., & Chen, Y. Link prediction based on graph neural networks. In NeurIPS 2018.