NED-GNN: Detecting and Dropping Noisy Edges in Graph Neural Networks

Ming Xu[1,2], Baoming Zhang[1,2], Jinliang Yuan[1,2], Meng Cao[1,2], and Chongjun Wang[1,2(✉)]

[1] State Key Laboratory for Novel Software Technology, Nanjing, China
{zhangbm,yuanjl19,caomeng}@smail.nju.edu.cn, chjwang@nju.edu.cn
[2] Nanjing University, Nanjing, China

Abstract. Graph neural networks have become the standard learning architectures in graph-based learning and achieve great progress in real-world tasks. Existing graph neural network methods are mostly based on message passing neural network(MPNN), which aggregates messages from neighbor nodes to update representations of target nodes. The framework follows the assumption of homophily that nodes linked by edges are similar and share the same labels. In the real world, the graphs can mostly follow the assumption. However, for nodes in the graph, the connections between nodes are not always connecting two similar nodes. We regard the edges as noisy edges. Such edges will introduce noise to message passing in the training process and hurt the performance of graph neural networks. To figure out the noisy edges and alleviate their influence, we propose the framework called *Noisy Edge Dropping Graph Neural Network*, short as NED-GNN. By evaluating the weights between sampled negative edges and existing edges for each node, NED-GNN detects and removes noisy edges. Extensive experiments are conducted on benchmark datasets and the promising performance compared with baseline methods indicates the effectiveness of our model.

Keywords: Graph neural networks · Noisy edges · Graph learning · Data mining

1 Introduction

Deep Learning has achieved great success in enhancing machine learning tasks with data in Euclidean space, like computer vision [18], natural language processing [17], etc. In recent years, research of the non-Euclidean graph-structured data is increasingly popular as graphs are widely used in presenting objects and their complex interactions. Graph learning is promoted to capture node features and topology information in graphs by representing nodes to low-dimensional embeddings. In this way, the learned embeddings can be well applied to machine learning methods, thus solving downstream tasks. Graph Neural Networks (GNNs) introduce deep learning to graph learning and show promising capacity, thus

B. Li et al. (Eds.): APWeb-WAIM 2022, LNCS 13421, pp. 91–105, 2023.
https://doi.org/10.1007/978-3-031-25158-0_8

being broadly used in tackling problems such as node classification [13,22], link prediction [8,28] and graph classification [9,24]. In the real world, GNN also becomes a promising solution for recommendation [16,21], social networks [5], text extraction [26], knowledge graphs [1], etc.

Existing GNN methods mostly follow the manner of message passing neural network (MPNN) [10]. The main idea of MPNN is to aggregate features from neighbor nodes. Then the representation of nodes is updated in each iteration and the final embeddings can be used for downstream tasks. The effectiveness of MPNN benefits from the assumption of homophily. When the two nodes linked by one edge are similar and share the same labels, nodes can aggregate similar and useful information from neighbor nodes. After that, the final representations can be robust and generalized. In real-world graphs, nodes with similar features tend to gather, which can well satisfy the assumption. However, the particular node in the graphs sometimes connects to dissimilar neighbors. For instance, nodes in the boundaries between classes connect to different-labeled nodes in the connected graph. Besides, the adversarial attacks in graphs often happens through connecting nodes of different classes. In particular, we call the edges constructed by nodes of different classes **Noisy Edges**. To validate our observations, we calculate the number of edges with nodes of different classes in widely-used citation datasets. The results are represented in Table 1.

Table 1. Noisy Edges Counting in Datasets

	Overall Edges	Noisy Edges	Propotion(%)
Cora	5278	1003	19.00
Citeseer	4614	1220	26.45
PubMed	44325	8759	19.76

From Table 1, we can find noisy edges exists commonly even in widely-used benchmark datasets. Such noisy edges may hurt the performance of GNNs as they bring the noise to target nodes by aggregating unnecessary information and mislead the training of the model. Here we show an example in the sample graph:

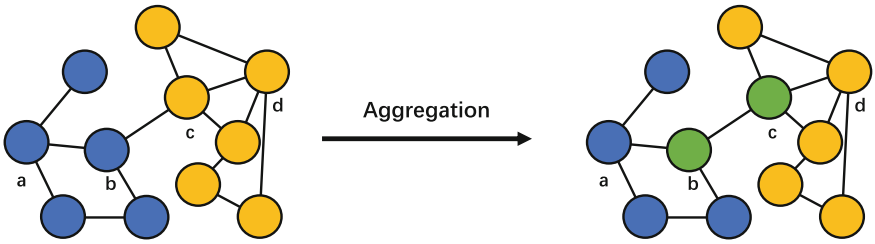


Fig. 1. An example of aggregation in graph neural networks.

In Fig. 1, we conducted 1-step aggregation in the sample graph, and different colors indicate different features. According to the figure, as the node a and node d share neighbors with same colors, respectively, their colors remain the same after aggregation. However, node b and node c change their colors as different-colored neighbors' information propagated, and their final representations are unreliable. What's worse, the noisy information will influence other nodes with the procedure of iterative aggregation.

To alleviate the undesirable effects of noisy edges, an intuitive approach is to remove such edges. Some works are proposed to update the structure with different strategies. DropEdge [20] tries to drop edges randomly to improve the generalization capacity of GNNs. However, the edge dropping without supervision sometimes breaks the topology structure and discards the essential relations among nodes in graphs. Besides, some methods update topology structure by the prediction of the model or co-training both structure learning and GNN models. The methods achieve success, but somehow they get stuck in the confidence of predictions with the setting of prediction threshold or the interpretability of the structural modifications in the graph.

In this paper, we focus on modifying the graph structure to obtain better node embeddings. In particular, we proposed an effective method, **Noisy Edges Dropping Graph Neural Networks(NED-GNN)**, to evaluate the edges and remove the noisy ones. We first sample negative edges for nodes, which connect the node with negative nodes of different classes in the graph. After that, we evaluate the weights of edges and the negative edges after sampling by calculating the similarity between linked nodes. If the weights of edges connecting neighbors are smaller than negative edges, we tend to regard the edges as noisy edges and remove them in the graph. As nodes in the graphs are mostly unlabeled, we propose that high-order neighbor nodes are more likely to be negative nodes that are different-labeled. So we sample negative edges by connecting nodes to corresponding high-order neighbor nodes. By evaluating similarity weights, we can drop the noisy edges to improve the node representations learned by graph neural networks. From the extensive experiments on benchmark datasets, promising performance indicates the effectiveness of our method.

We summarize the main contributions of this paper as follows:

- We proposed an effective framework to detect and drop noisy edges in the graph.
- We designed a method called NED-GNN by dropping detected noisy edges to update the topology structure and obtain better node embeddings.
- We conducted extensive experiments on semi-supervised node classification and prove the effectiveness of our method.

The remaining part of the paper is organized as follows. Section 2 reviews the related works. We show an observation of real-world datasets in Sect. 3. In Sect. 4 we introduce some preliminaries and our method in detail. Extensive experiments are conducted in Sect. 5 to evaluate the performance of our model. At last, Sect. 6 concludes the paper with discussions and future works.

2 Related Works

In this section, we briefly review the related works, including graph neural networks and modifications to graph structure. For more details in graph neural networks, we refer readers to some surveys [11,30].

2.1 Graph Neural Networks

Graph Neural Networks achieve great success in tackling graph learning tasks in recent years. GNN models encode nodes in the graph into low-dimensional dense embeddings and preserve both node features and structure topology at the same time. Graph convolution is first proposed in [2] from the respective of graph signal processing, and many variants are proposed to simplify the framework in both spectral and spatial domain [11,14,30]. Kipf e.t. [13] simplify the model by considering the direct neighbors of nodes in GCN. The simpleness and conciseness of GCN make the model popular and become the baseline in graph learning. Existing graph neural network models are mainly based on message passing neural network (MPNN) [10], which aggregates messages from neighbors nodes and then update representations of target nodes [22,24].

The models based on MPNN mostly follow the assumption of homogeneity, which states that nodes connected by edges are similar and beneficial information can be propagated in the graph. However, the goal is hard to achieve as there always unintentional or intentional exists noise in real world graphs. We focus on the structural noise, mainly noisy edges, and give a brief introduction to graph structure modification in the next subsection.

2.2 Graph Structure Modification

Graph neural networks utilize both node features and graph structure to encode nodes. However, the commonly existing structural noise, like noisy edges, may hurt the performance of GNNs as the noise may mislead information spread in graphs. To alleviate the influence of structural noise, training with a better topology structure is crucial for better node embeddings. For instance, DropEdge [20] randomly removes a certain number of edges at each epoch to improve the generalization capacity and alleviate over-smoothing. NeuralSparse [31] learns k-neighbor subgraphs for robust graph representation learning by selecting at most k edges for each nodes. Methods like self-enhanced GNN [25], EGAI [15], and AdaInf [6] add or remove edges based on the predicted labels by the model. Bayesian GCN [29], LDS [7] and IDGL [4] adopt different strategies to learn the graph structure and node embeddings simultaneously to make graph structure more suitable for model learning. There are contrasting models try to construct multi views by modifying the structure [3,27]. The methods achieve great progress in acquiring better node embeddings with modified structure, however they may get stuck in randomicity or lack interpretability of the modified structure.

In this paper, we try to update the structure of the graph to help the training of GNNs. In particular, we proposed a simple but effective method to detect end drop noisy edges in the graph.

3 Case Study

GNNs utilize message passing framework to propagate messages from neighbor nodes by assuming that nodes connected in graphs are similar. In this section, we did some simple analysis on the widely used benchmark citation dataset Cora from *DGL* [23], and the findings prove the necessity and rationality of our method.

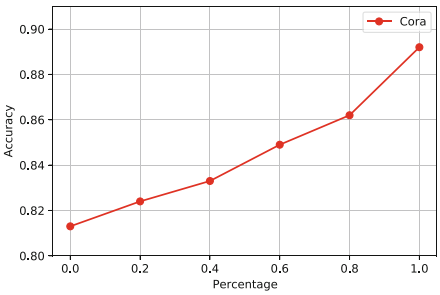


Fig. 2. Accuracy with Different Percentage of Noisy Edges Dropping

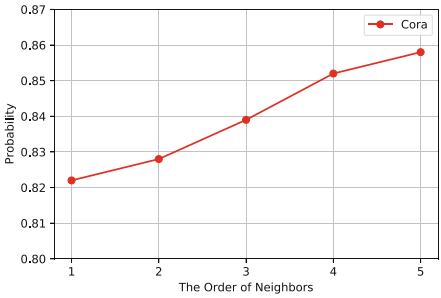


Fig. 3. Probability of Connecting Different-labeled Nodes beyond k-Order Neighbors

Model Performance Without Noisy Edges. We design an empirical experiment to figure out the influence of noisy edges. In particular, we randomly remove noisy edges in Cora with given percentages. After that, we train GCN in the updated graph and use trained embeddings for node classification. The results of classification accuracy are shown in Fig. 2. The results indicate that the representation capacity of GCN would be better with more noisy edges dropping.

The statistical results show that noisy edges are common in real-world graphs, and they will harm the performance of GNNs. Naturally, the conclusion prompts us to consider dropping noisy edges in the training process.

Nodes and the Corresponding High-Order Neighbors. According to the assumption of homophily, nodes with similar features and labels tend to gather, while different-labeled nodes keep away from each other. We calculate the probabilities of nodes sharing different labels with the corresponding high-order neighbors in Cora, and the results are presented in Fig. 3. From the figure, we can figure out that nodes are becoming more dissimilar with the distance between nodes growing. The observation can help find out the noisy edges in the graph.

4 Our Approach

The case study shows the motivation of our work. To better explain and prove our idea, we propose the method called NED-GNN. The main framework is shown in Fig. 4. Our key insight is to drop noisy edges in the graph and update the topology structure to improve the performance of GNNs.

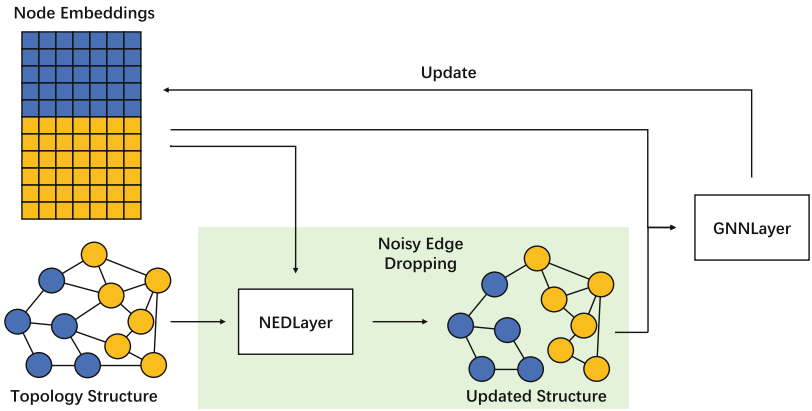


Fig. 4. The Framework of proposed NED-GNN Layer.

4.1 Notations and Preliminaries

This paper mainly focuses on undirected graphs, but the method can also be used in directed graphs. We present $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ as a graph, where $\mathcal{V}$ consists of the set of nodes in $\mathcal{G}$, with $|\mathcal{V}| = N$. $\mathcal{E}$ is the collection of edges. $\mathcal{X} \in \mathbb{R}^{N \times F}$ denotes node feature matrix, where $\mathbf{x_i} \in \mathbb{R}^F$ represents the attributes of node v_i, and F is the dimension of node features. Adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ is the topological structure of graph $\mathcal{G}$, where $\mathbf{A}_{ij} > 0$ indicates that there is an edge between nodes i and j. Otherwise, $\mathbf{A}_{ij} = 0$.

Given topological structure $\mathbf{A}$ and feature matrix $\mathcal{X}$ as input, the goal of GNN framework is to learn low-dimensional dense node embedding matrix $\mathbf{Z} \in \mathbb{R}^{N \times d}$ with $d \ll F$. The learned node embeddings can well preserve topology and feature information so as to be applied to downstream tasks.

The training procedure of GNNs is as follows. First, the framework learns node representations by aggregating the features of neighbor nodes. The output of the k-th layer of the framework can be generally expressed as,

$$\mathbf{h}_i^{(k)} = \sigma(\mathbf{h}_i^{(k-1)}, AGG(\mathbf{h}_j^{(k-1)})), \quad j \in \mathcal{N}(i) \tag{1}$$

where $h_i^{(l)}$ is the node representation of node v_i at the k-th layer with $h_i^{(0)} = \mathbf{x}_i$ and $\mathcal{N}(i)$ is the direct neighbors of node v_i. $AGG(\cdot)$ is the aggregation function and $\sigma(\cdot)$ is the non-linear function.

After that, the framework calculates the loss of prediction on labeled nodes,$\mathcal{V}_L$, to update the parameters in the framework. The loss function can be expressed as,

$$\mathcal{L} = -\frac{1}{|\mathcal{V}_L|} \sum_{i \in \mathcal{V}_L} \sum_{l=1}^{k} \mathbf{y}_{il} \log \hat{\mathbf{y}}_{il}, \tag{2}$$

where $\hat{\mathbf{y}}_i$ is the prediction of v_i, $\mathbf{y}_i$ indicates the original label for node v_i in one-hot embedding, k is the length of label embedding.

4.2 Noisy Edges Dropping

Before the introduction of our method, we regard the nodes which are labeled differently to target nodes as **negative nodes** and the connected edges as **negative edges**.

The main idea of our method is to figure out noisy edges and update the topology structure by removing the edges. Intuitively, if we find the neighbor node is less similar to the target node than a negative node, we think the edge connecting the neighbor is the noisy edge. Figure 5 is an example. We sample a negative node j for the target node t and constructed a negative edge (t, j). Then we evaluate the weight of edge (t, i) and (t, j), namely the similarities between nodes. If $sim(t, i) < sim(t, j)$, the edge between t and i is possible noisy. As a result, the edge will be removed.

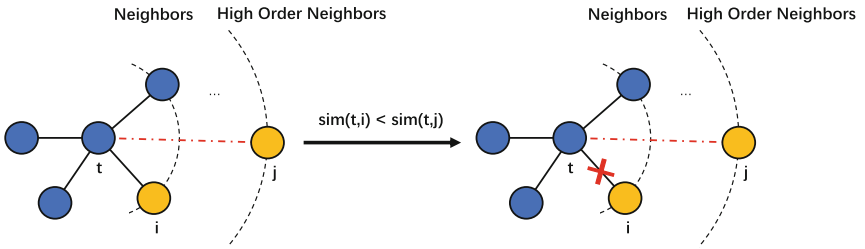


Fig. 5. An example of noisy edge dropping

To calculate the similarity between nodes, we set,

$$sim(i, j) = similarity(\mathbf{h}_i, \mathbf{h}_j) \tag{3}$$

$\mathbf{h}_i$ is the representation of node v_i, the choice of $similarity(\cdot)$ can be any function that measures the similarity of two embeddings, like Cosine similarity and Euclidean Distance. For the simplicity, we choose Cosine similarity as our similarity function.

However, the nodes in the graph are mostly unlabeled, which is unable for us to directly sample negative nodes and negative edges. To solve the problem, we proposed the idea of sampling negative edges according to the property of

graphs. The similarity of nodes becomes weaker as the distance increasing in graphs. The case study in Sect. 3 also validates our assumption. In particular, for each node, no more than r-hop neighbors can be regarded as the positive nodes which are more likely to be similar, while the other nodes are seen as the negative nodes. So we sample neighbor nodes beyond r-order as the negative nodes and then constructing negative edges.

Algorithm 1: NEDLayer Algorithm

Input: Adjacency matrix $\mathbf{A}$, Representation Matrix $\mathbf{Z}$
Output: Updated Adjcency Matrix $\mathbf{A}'$

1 $\mathbf{A}_{neg} = Sample(A^{\neg r})$ ← Sample k negative edges beyond r-order neighbors
2 $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{A}_{neg}$ ← Concatenate the negative edges with original graph
3 **for** *node i in* $\hat{\mathbf{A}}$ **do**
4 **for** *node j where* $\hat{\mathbf{A}}_{ij} > 0$ **do**
5 $sim_{ij} = similarity(\mathbf{Z}_i, \mathbf{Z}_j)$ ← Evaluate the weight of edge (i, j) in $\hat{\mathbf{A}}$
6 $indices_i = Topk(sim_{i,}, k = Degree(i))$ ← Remove the less similar edges constrained by the degree in original graph
7 $\mathbf{A}'_{i,} = \{\mathbf{A}'_{ij} = 1 | \mathbf{A}_{ij} > 0 \; and \; j \in indices_i\}$ ← Remove sampled negative edges
8 return $\mathbf{A}'$

Algorithm 1 summarizes the overall training of the NEDLayer. In each epoch, we sample k negative edges connecting nodes that are beyond r-order neighborhood in Line 1–2. For each node i, we evaluate the connected edges(i, j) by calculating the similarity of two-side nodes, thus dropping likely noisy edges by removing the edges with lower similarity in Line 3–6. The constraint of $Degree(i)$ is the degree of i in the original graph, which is used to accelerate the procedure. In Line 7, we remove the sampled negative edges. Then we get the updated topology structure $\mathbf{A}'$.

In the process, we sample negative edges randomly from the whole graph and remove noisy edges for each node, and the updated $\mathbf{A}'$ is an asymmetrical matrix. With the NEDLayer, we update the topology structure in a self-supervised manner by considering the embeddings of nodes and the property of graphs.

4.3 NED-GNN

After the procedure of NEDLayer, we detect and drop the likely noisy edges in the graph and generate the modified graph structure. With the generated structure, we can update the aggregation process by setting,

$$\mathbf{A}'^{(k)} = normalize(NEDLayer(\mathbf{A}, \mathbf{H}^{(\mathbf{k-1})})) \tag{4}$$

$$\mathbf{H}^{(\mathbf{k})} = \sigma(\mathbf{A}'^{(k)}\mathbf{H}^{(\mathbf{k-1})}\mathbf{W} + \mathbf{B}) \tag{5}$$

where $normalize(\cdot)$ performs L_1 normalization by row to the adjacency matrix. $\sigma(\cdot)$ is the non-linear method, $\mathbf{W}$ is the parameters of GNNLayer, and $\mathbf{B}$ is

the bias. As the representations of nodes are unstable at early epochs and may result in mistakenly removing edges connecting two similar nodes, we update the topology from the original graph in each epoch to avoid the situation.

4.4 The Variant of NED-GNN

Though NED-GNN can alleviate unnecessary information aggregation in graph neural networks, the modification of graph structure sometimes brings side effects. As we focus on homophily graphs in the paper, where nodes mostly follow the assumption of homophily, there is no need for the negative edges sampling for all nodes. Besides, most of the graphs are sparse in the real world, so the process of edge dropping may break the substructure and result in poor performance when excessive edges are dropped.

According to the training process of graph neural networks, we can find that the parameters of the framework mainly depend on the labeled nodes by calculating the prediction loss. Inspired by the observation, we promote the variant of NED-GNN, **Noisy Edges Dropping Graph Neural Networks on Training Nodes(NED-GNN-t)**, which focuses on the training nodes, namely sampling negative nodes and dropping noisy edges only for labeled nodes. Then, we can decrease the number of negative edges and the calculation of similarity between nodes. In this way, we can accelerate the process of NEDLayer and improve training efficiency. What's more, as NED-GNN-t only removes the edges of the training nodes, the modification of the structure is mild with less edges dropped, and the model can preserve more topology information.

5 Experiments

The goal of our method is to detect and drop noisy edges in the graph which may harm the aggregation in the training of GNNs. To prove the effectiveness of our method in improving GNN representation capacity, we designed the semi-supervised node classification experiments on benchmark datasets. The procedure is introduced as follows.

5.1 Experimental Settings

Datasets. Following previous works [13], we utilize three benchmark datasets of citation network, Cora, Citeseer and Pubmed. In these datasets, nodes and edges represent documents and citation relations between documents, respectively. Each node is represented by the bag-of-words features extracted from the content of the document. Each node corresponds a label with one-hot encoding of the document category. We employ the same data split in *DGL* [23] module and the data distribution is shown in Table 2.

Table 2. Data distribution of Three Datasets

	Cora	Citeseer	Pubmed
Nodes	2078	3327	19717
Edges	5278	4614	44325
Features	1433	3703	500
Classes	7	6	3
Training Nodes	140	120	60
Validation Nodes	500	500	500
Test Nodes	1000	1000	1000

Baseline Methods. To evaluate the effectiveness of our method, we compare with the following *state-of-the-art* methods.

- **DeepWalk** [19]. The typical shallow network embedding model.
- **GCN** [13]. The baseline of graph nerual networks, which considers aggregating messages from direct neighbors.
- **GraphSage** [12]. Extending the mean aggregator of GCN to perform multi aggregation and performing a sampling strategy before aggregation.
- **GAT** [22]. Considering introducing attention mechanism to GCN and assigns different weights to neighbor nodes according to attention scores.
- **GAT-single**. The method which only utilizes a single attention head for GAT. As our method can somehow be regarded as an further step for single-head GAT by removing low weights.
- **DropEdge** [20]. Randomly removing a certain number of edges at each epoch to improve the generalization capacity of GCN.
- **NED-GNN**. Our method, we choose the GCN and GAT as our baseline method.
- **NED-GNN-t**. The variant of NED-GNN, which samples negative edges only for labeled nodes.

Parameter Settings. In parameter settings, We designed 2-layer neural networks with the same hidden layer dimension and the same output dimension simultaneously for every method. For baseline methods like DeepWalk, GCN, GAT, and DropEdge, we follow the instruction of original codes in Github published by the authors. For GraphSage, we only consider the situation with mean aggregator, and the model is implemented the same as the authors' guidance. With our methods, we follow most settings of GCN except that we use an early stopping strategy the same as GAT with a patience of 100 epochs. For all models, we conducted 10 times and got the average results to show the performance.

5.2 Experiments Comparison

Table 3 shows the results of the compared methods on semi-supervised node classification. From the table, we can find the following observations. Our method achieves the best or competitive performance compared with baseline methods

in all datasets. The promising results show the effectiveness of our method which removes the noisy edges in the graph by comparing with negative edges. NED-GAT performs better than NED-GCN as GAT can still assign different weights to noisy edges which are not detected. DropEdge randomly removes a certain percentage of edges in the graph to achieve better performance. However, the randomicity is hard to control and sometimes discards the important interactions between nodes, resulting in unsatisfying results. GraphSage also performs a sampling strategy before aggregation and would get stuck in the randomicity with a performance drop.

Table 3. The results of node classification accuracy(%) on the three datasets

Data	Cora	Citeseer	Pubmed
DeepWalk	67.2	43.2	65.3
GraphSage	77.4	67.0	76.6
GIN	77.6	66.1	77.0
DropEdge	79.6	67.6	73.4
GCN	81.6	70.5	78.7
NED-GCN	**82.9**	70.9	**79.0**
NED-GCN-t	82.6	**73.4**	78.8
GAT	82.6	70.3	77.5
GAT-single	81.6	70.3	77.4
NED-GAT	**83.3**	71.0	**78.1**
NED-GAT-t	**83.3**	**73.1**	**78.1**

NED-GNN-t shows competitive performance in Cora compared with NED-GNN, which indicates that the representative embeddings of labeled nodes are essential for graph neural networks. In the sparse graph like Citeseer, NED-GNN-t outperforms NED-GNN as the side-effects of NED-GNN break the important topology structure, which result in sub-optimal performance. While NED-GNN-t focuses on labeled nodes and modifies a small part in the graph to preserve more topology information.

Besides, GAT outperforms other baselines in Cora as they can prevent the aggregation of nodes from different classes by assigning the noisy edges small weights. However, as GAT uses $softmax(\cdot)$ functions to calculate the weights of attention, the weights of edges will always be positive. Our method tries to remove a part of noisy edges, namely setting the weights of the edges to 0, thus our method performs competitive results. In fact, our method can somehow be regarded as a single-head attention mechanism that calculates the similarity of neighbors directly, but we apply the weights to filter out the dissimilar nodes with a further step. And the results show that our method can learn better node embeddings than GAT with a single head.

Among the baseline methods, the GCN model performs better than DeepWalk as graph convolution is powerful by combining node features and topology information. GAT outperforms GCN in some cases as GAT introduces attention

to graph convolution to decide the more correlative neighborhoods. The results are consistent with those in previous works.

5.3 Parameter Study

We conduct the ablation study to verify the effect of the number of negative edges k and the order of sampling neighbors r. The results of node classification on Cora dataset are shown in Fig. 6 and Fig. 7. We traverse k from 200 to 2000 and r from 2 to 6 to show the influence of the parameters.

The Number of Sampled Edges. From the results of Fig. 6, we can figure out that when we sample more edges, the performance of the model increase at first as more negative edges are sampled in the graph which can help detect noisy edges. However, more is not always better. When we sample more than 800 edges in the graph, the performance decreases. As the discussion in Sect. 3, noisy edges occupy a small proportion in the graph. The capacity gain brought by negative edges is then offset by the side-effects of NED-GNN. In conclusion, there is no need for a large number of sampled negative edges.

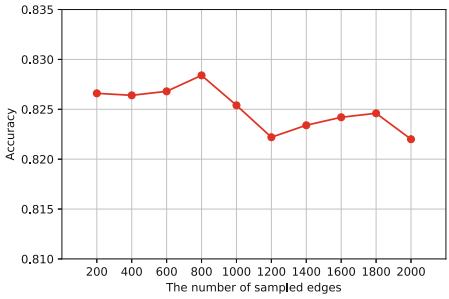


Fig. 6. Accuracy with Different Number of Sampled Edges

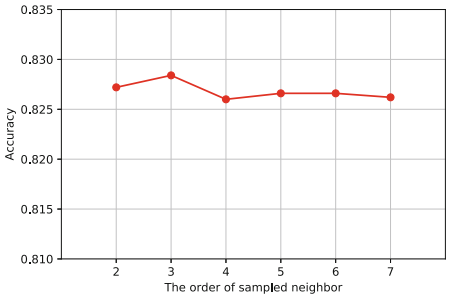
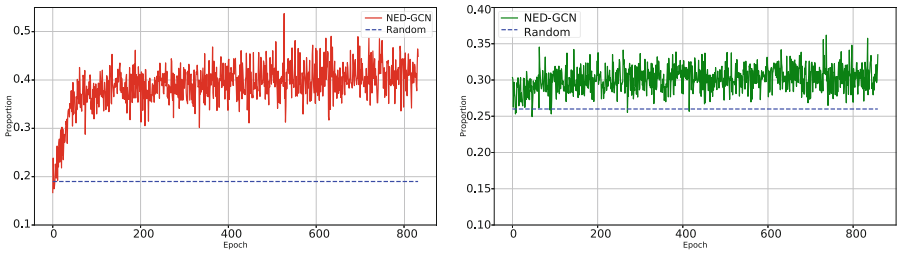


Fig. 7. Accuracy with Different Neighbor Order

The Order of Sampled Neighbors. In Fig. 7, classification accuracy is robust with the increasing order of sampled neighbors. The results are consistent with our statistical results. When we sample nodes without nodes in 3-order neighbors, the probability of noisy edges is larger than 83%. So in the practice of our method, the choice with low order neighbors can well satisfy the model training.

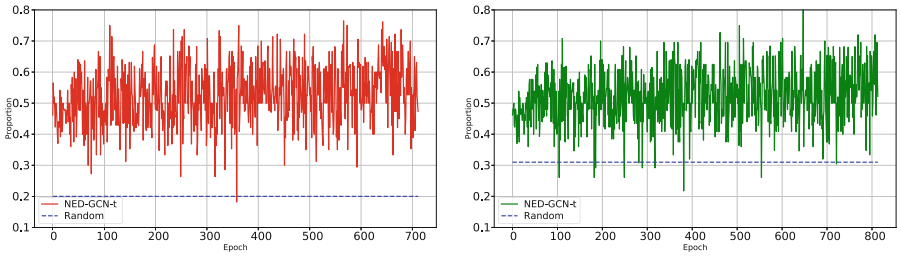
5.4 Training Visualization

In this section, we conduct experiments on Cora and Citeseer to verify the noisy edges dropping in the training of NED-GNN. The proportion of noisy edges dropped by our methods in the training process is shown in Fig. 8 and 9. The blue lines in the figures are the probability of randomly dropping noisy edges.



(a) Cora (b) Citeseer

Fig. 8. The proportion of noisy edges in dropped edges when training NED-GCN.



(a) Cora (b) Citeseer

Fig. 9. The proportion of noisy edges in dropped edges when training NED-GCN-t.

From the figures, we can conclude that our method can effectively detect and drop the noisy edges in the graph compared with randomly dropping the edges. Considering the performance in Citeseer, which is too sparse with the average degree of node is 2.8, we can also figure out that NED-GCN-t can better detect and drop noisy edges than NED-GCN. The reason is that NED-GCN-t performs a wilder edge dropping strategy and can preserve more topology structure information in the graph.

6 Conclusion

Existing graph neural network methods mostly follow the assumption of homogeneity, which states that nodes connected by edges are similar and share the same labels. However, particular nodes in the graph sometimes break the assumption. In this paper, we try to discuss noisy edges in the graph and remove the edges to obtain better embeddings. In particular, we proposed a simple and effective method called *Noisy Edge Dropping Graph Neural Network*, short as NED-GNN, to detect and remove noisy edges in the graph by sampling negative edges. Extensive experiments are conducted on benchmark datasets and promising performance shows the effectiveness of our method. In the future, we aim to improve the capacity of our method in detecting noisy edges by more strategies.

Acknowledgements. This paper is supported by the National Key Research and Development Program of China (Grant No. 2018YFB1403400), the National Natural Science Foundation of China (Grant No. 61876080), the Key Research and Development Program of Jiangsu(Grant No. BE2019105), the Collaborative Innovation Center of Novel Software Technology and Industrialization at Nanjing University.

References

1. Arora, S.: A survey on graph neural networks for knowledge graph completion. arXiv preprint arXiv:2007.12374 (2020)
2. Bruna, J., Zaremba, W., Szlam, A.D., Lecun, Y.: Spectral networks and locally connected networks on graphs. CoRR abs/1312.6203 (2014)
3. Chen, X., Zhang, Y., Tsang, I., Pan, Y.: Learning robust node representations on graphs. arXiv preprint arXiv:2008.11416 (2020)
4. Chen, Y., Wu, L., Zaki, M.: Iterative deep graph learning for graph neural networks: Better and robust node embeddings. Proc. Adv. Neural Inf. Proc. Syst. **33**, 19314–19326 (2020)
5. Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., Yin, D.: Graph neural networks for social recommendation. In: Proceedings of the World Wide Web Conference, pp. 417–426 (2019)
6. Feng, F., Huang, W., Xin, X., He, X., Chua, T.S.: Should graph convolution trust neighbors a simple causal inference method. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1208–1218 (2021)
7. Franceschi, L., Niepert, M., Pontil, M., He, X.: Learning discrete structures for graph neural networks. In: Proceedings of the International Conference on Machine Learning, pp. 1972–1982 (2019)
8. Gao, H., et al.: CSIP: enhanced link prediction with context of social influence propagation. Big Data Res. **24**, 100217 (2021)
9. Gao, H., Ji, S.: Graph u-nets. In: Proceedings of the International Conference on Machine Learning, pp. 2083–2092 (2019)
10. 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, pp. 1263–1272 (2017)
11. Hamilton, W.L.: Graph representation learning. Synth. Lect. Artif. Intell. Mach. Learn. **14**(3), 1–159 (2020)
12. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the International Conference on Neural Information Processing Systems, pp. 1025–1035 (2017)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the International Conference on Learning Representations (2017)
14. Kipf, T.N., et al.: Deep learning with graph-structured representations (2020)
15. Liu, C., Wu, J., Liu, W., Hu, W.: Enhancing graph neural networks by a high-quality aggregation of beneficial information. Neural Netw. **142**, 20–33 (2021)
16. Liu, Y., Li, B., Zang, Y., Li, A., Yin, H.: A knowledge-aware recommender with attention-enhanced dynamic convolutional network. In: Proceedings of the 30th ACM International Conference on Information Knowledge Management, pp. 1079–1088 (2021)

17. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of Advances in neural information processing systems, pp. 3111–3119 (2013)
18. O'Mahony, N., et al.: Deep learning vs. traditional computer vision. In: Proceedings of Science and Information Conference, pp. 128–144 (2019)
19. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
20. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: Proceedings of the International Conference on Learning Representations (2020)
21. Sun, J., et al.: Neighbor interaction aware graph convolution networks for recommendation. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1289–1298 (2020)
22. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. In: Proceedings of the International Conference on Learning Representations (2018)
23. Wang, M., et al.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
24. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks In: Proceedings of the International Conference on Learning Representations (2019)
25. Yang, H., Yan, X., Dai, X., Chen, Y., Cheng, J.: Self-enhanced GNN: Improving graph neural networks using model outputs. arXiv preprint arXiv:2002.07518 (2020)
26. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 7370–7377 (2019)
27. You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., Shen, Y.: Graph contrastive learning with augmentations. Adv. Neural Inf. Proc. Syst. **33**, 5812–5823 (2020)
28. Zhang, M., Li, P., Xia, Y., Wang, K., Jin, L.: Revisiting graph neural networks for link prediction. arXiv preprint arXiv:2010.16103 (2020)
29. Zhang, Y., Pal, S., Coates, M., Ustebay, D.: Bayesian graph convolutional neural networks for semi-supervised classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 5829–5836 (2019)
30. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: a survey. IEEE Trans. Knowl. Data Eng. **34**(1), 249–270 (2020)
31. Zheng, C., et al.: Robust graph representation learning via neural sparsification. In: Proceedings of the International Conference on Machine Learning, pp. 11458–11468 (2020)