Self-Enhanced GNN: Improving Graph Neural Networks Using Model Outputs

Han Yang
hyang@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

Xiao Yan
xyan@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

Xinyan Dai
xydai@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

James Cheng
jcheng@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

ABSTRACT

Graph neural networks (GNNs) have received much attention recently because of their excellent performance on graph-based tasks. However, existing research on GNNs focuses on designing more effective models without considering much the quality of the input data itself. In this paper, we propose *self-enhanced GNN*, which improves the quality of the input data using the outputs of existing GNN models for better performance on semi-supervised node classification. As graph data consist of both topology and node labels, we improve input data quality from both perspectives. For topology, we observe that higher classification accuracy can be achieved when the ratio of inter-class edges (connecting nodes from different classes) is low and propose *topology update* to remove inter-class edges and add intra-class edges. For node labels, we propose *training node augmentation*, which enlarges the training set using the labels predicted by existing GNN models. As self-enhanced GNN improves the quality of the input graph data, it is general and can be easily combined with existing GNN models. Experimental results on three well-known GNN models and seven popular datasets show that self-enhanced GNN consistently improves the performance of the three models. The reduction in classification error is 16.2% on average and can be as high as 35.1%.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; • **Networks** → **Topology analysis and generation**; • **Theory of computation** → **Graph algorithms analysis**; • **Mathematics of computing** → *Graph algorithms*.

KEYWORDS

Graph neural networks, graph representation learning, semi-supervised node classification

1 INTRODUCTION

Graph data are ubiquitous today, e.g., friendship graphs in social networks, phone call or message graphs in tele-communication, user-item interaction graphs in recommender systems, and protein-protein interaction graphs in biology. For graph-based tasks such as node classification, link prediction and graph classification, *graph neural networks* (*GNNs*) achieve excellent performance thanks to its ability to utilize both graph structure and feature information (on nodes or edges). Most GNN models can be formulated under the message passing framework, in which each node passes messages to its neighbors in the graph and aggregates messages from the neighbors to update its own embedding.

Different attempts have been made to design algorithms and models for graph analytics. Random walk based methods, e.g., Deep-Walk [22] uses the random walk paths as the input to a skip-gram model to learn node embeddings , while node2vec [6] learns node embeddings by combining breadth-first random walk and depth-first random walk. Motivated by graph spectral theory, graph convolutional network (GCN) [11] conducts graph convolution using the adjacency matrix to avoid the high complexity spectral decomposition. Instead of using the adjacency matrix to derive the weights for message aggregation, graph attention network (GAT) [25] uses an attention module to learn the weights from data. Simplifying graph convolution network (SGC) [27] proposes to remove the non-linearity in GCN as it observes that the good performance of GCN mainly comes from local averaging rather than non-linearity. There are also many other GNN models such as GraphSAGE [7], jumping knowledge network (JK-Net) [28], geometric graph convolutional network (Geom-GCN) [21], and gated graph neural network (GGNN) [14], and we refer readers to a comprehensive survey in [30].

In this paper, we focus on the problem of *semi-supervised node classification*, which is also most GNN models are designed for. We observed that most existing researches attempt to design more effective GNN models, while the quality of the input data has not received much attention. However, *data quality* [1] and model quality can be equally important for good performance. For example, if the input graph contains only *intra-class edges* (i.e., edges connecting nodes from the same class) and no *inter-class edges* (i.e., edges connecting nodes from different classes), node classification can achieve perfect accuracy with only one training sample from each connected component. Moreover, classification tasks are usually easier with more training samples.

At first glance, data quality (i.e., the quality of the input graph structure and training nodes) is the fixed problem input and cannot be improved. However, we observed that existing GNN models already achieve good classification accuracy, and thus their outputs

---

[1] Here, data quality is problem-specific. Given a GNN model and a specific problem, high data quality means that the GNN model produces good output for the problem on the input data. In this paper, we discuss data quality with respect to the node classification problem.

can actually be used to update the input data to improve its quality. Then, the GNN models can be trained on the improved data to achieve better performance. We call this idea **self-enhanced GNN** and propose two algorithms under this framework, namely **topology update (TU)** and **training node augmentation (TNA)**.

As GNN models essentially smooth the embeddings of neighboring nodes, inter-class edges harm the performance as they make it difficult to distinguish nodes from different classes. To this end, TU removes inter-class edges and adds intra-class edges according to node labels predicted by a GNN model. Our analysis shows that TU reduces the percentage of inter-class edges in the input graph as long as the performance of the GNN model is good enough. Since the number of labeled nodes are usually small for semi-supervised node classification, TNA enlarges the training set by treating the predicted labels of multiple GNN models as the ground truth. We show by analysis that jointly considering the predicted labels of multiple diverse GNN models reduces errors in the enlarged training set. We also develop a method to create diversity among multiple GNN models. In addition, we propose techniques such as *threshold-based selection*, *validation-based tuning* and *class balance* to stabilize the performance of TU and TNA. Both TU and TNA are general techniques that can be easily combined with existing GNN models.

We conducted extensive experiments on three well-known GNN models, GCN, GAT and SGC, and seven widely used benchmark datasets. The results show that self-enhanced GNN consistently improves the performance of different GNN models. The reduction in the classification error is 16.2% on average and can be up to 35.1%. Detailed profiling finds that TU and TNA indeed improve the input data quality for node classification. Specifically, TU effectively improves an input graph for the task by deleting inter-class edges and adding intra-class edges, while most of the nodes added by TNA are assigned a right label. Based on the results, one interesting future direction is to apply the idea of self-enhanced GNN to other problems such as link prediction and graph classification where GNNs are also used.

2 TOPOLOGY UPDATE

Denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. There are n nodes and m edges in the graph. The ground-truth label of a node v is $l(v)$. We define the *noise ratio* of the graph as

$$\alpha = \frac{|\{e_{uv} \in \mathcal{E} | l(u) \neq l(v)\}|}{|\mathcal{E}|}. \tag{1}$$

Noise ratio measures the percentage of *inter-class edges* (i.e., e_{uv} with $l(u) \neq l(v)$) in the graph.

Motivation. In Figure 1, we show the relation between classification accuracy and noise ratio for the CORA dataset, where *edge deletion* randomly removes inter-class edges in the graph and *edge addition* randomly adds *intra-class edges* (i.e., e_{uv} with $l(u) = l(v)$) that are not present in the original graph. The results show that the classification accuracy of all the three models is higher with lower noise ratio. This is understandable since GNN models are generally low-pass filters that smooth the embeddings of neighboring nodes [18]. As inter-class edges encourage nodes from different
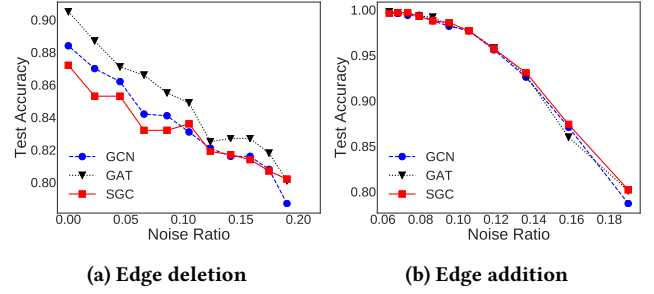

| (a) Edge deletion | (b) Edge addition |

Figure 1: The relation between noise ratio and classification accuracy for GCN, GAT and SGC on the CORA dataset (note that deleting inter-class edges or adding intra-class edges *reduces* noise ratio)

classes to have similar embeddings, they make the classification task difficult. Therefore, we make the following assumption.

ASSUMPTION 1. *Lower noise ratio leads to better classification performance for popular GNNs such as GCN, GAT and SGC.*

2.1 Topology Update Algorithms

For Figure 1, we delete/add edges using the ground-truth labels. However, we may not have access to the ground-truth labels in a practical node classification problem. As popular GNN models already provide quite accurate predictions of the true labels, we can use their output for edge edition. Denote a GNN model trained for a node classification problem with c classes as a mapping function $f : \mathbb{V} \to [c]$, where $[c]$ is the integer set $\{1, \ldots, c\}$. Edge deletion and edge addition can be conducted using Algorithm 1 and Algorithm 2, respectively.

Algorithm 1: Edge Deletion using Model Output

Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a trained GNN model $f(\cdot)$
Output: A new graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$
Initialize $\mathcal{E}' = \emptyset$;
for each edge $e_{uv} \in \mathcal{E}$ **do**
  if $f(u) = f(v)$ **then**
    Add e_{uv} to $\mathcal{E}'$;
  end if
end for

Algorithm 2: Edge Addition using Model Output

Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a trained GNN model $f(\cdot)$
Output: A new graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$
Initialize $\mathcal{E}' = \mathcal{E}$;
for each node pair $(u, v) \in \mathcal{V} \times \mathcal{V}$ **do**
  if $e_{uv} \notin \mathcal{E}$ and $f(u) = f(v)$ **then**
    Add e_{uv} to $\mathcal{E}'$;
  end if
end for

In the following, we show that Algorithm 1 and Algorithm 2 reduce the noise ratio of the input graph if the classification accuracy of the GNN model $f(\cdot)$ is high enough. We first present some assumptions and definitions that will be used in the analysis.

ASSUMPTION 2. *(Symmetric Error) The GNN model $f(\cdot)$ has a classification accuracy of p and makes symmetric errors, i.e., for every node $v \in \mathcal{G}$, we have $\mathbb{P}[f(v) = l(v)] = p$ and $\mathbb{P}[f(v) = k] = \frac{1-p}{c-1}$ for $k \in [c]$ and $k \neq l(v)$, where c is the number of classes and $l(v)$ is the ground-truth label of node v.*

Note that symmetric error is a common assumption in the literature [2] and our analysis methodology is not limited to symmetric error. As the GNN model $f(\cdot)$ makes random errors (and hence the topology update algorithms also make random errors), we use the expected noise ratio α_E as a replacement for the noise ratio α. For the graph after edition, i.e., $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$, we define the expected noise ratio as $\alpha_E = \frac{m_r}{m_r + m_a}$, in which m_r is the expected number of inter-class edges in $\mathcal{G}'$ and m_a is the expected number of intra-class edges in $\mathcal{G}'$. We can compare the expected noise ratio of $\mathcal{G}'$ with the noise ratio of the original graph $\mathcal{G}$.

THEOREM 1. *(Edge Deletion) If Assumption 2 holds and Algorithm 1 is used for edge deletion, denote the expected noise ratio of the output graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ as α_E, we have $\alpha_E < \alpha$ if $p > \frac{2}{c+1}$.*

PROOF. The probability that an intra-class edge in $\mathcal{G}$ is kept in $\mathcal{G}'$ by Algorithm 1 is $p_a = \mathbb{P}[f(v) = f(u)|l(u) = l(v)] = p^2 + \frac{(1-p)^2}{c-1}$. Therefore, $m_a = (1 - \alpha)m\left(p^2 + \frac{(1-p)^2}{c-1}\right)$, where m is the number of edges in $\mathcal{G}$. The probability that an inter-class edge is kept is $p_r = \mathbb{P}[f(v) = f(u)|l(u) \neq l(v)] = \frac{2p(1-p)}{c-1} + \frac{(c-2)(1-p)^2}{(c-1)^2}$, and thus $m_r = \alpha m\left(\frac{2p(1-p)}{c-1} + \frac{(c-2)(1-p)^2}{(c-1)^2}\right)$. We have

$$
\begin{aligned}
\alpha_E &= \frac{\alpha m\left(\frac{2p(1-p)}{c-1} + \frac{(c-2)(1-p)^2}{(c-1)^2}\right)}{\alpha m\left(\frac{2p(1-p)}{c-1} + \frac{(c-2)(1-p)^2}{(c-1)^2}\right) + (1-\alpha)m\left(p^2 + \frac{(1-p)^2}{c-1}\right)} \\
&< \frac{\alpha(1 - p^2)}{\alpha(1 - p^2) + (1 - \alpha)[(c-1)p^2 + (1-p)^2]}.
\end{aligned}
$$

Solving $\frac{\alpha(1-p^2)}{\alpha(1-p^2)+(1-\alpha)[(c-1)p^2+(1-p)^2]} < \alpha$ gives $(1-\alpha)p[(c+1)p - 2] \geq 0$, which is satisfied when $p > \frac{2}{c+1}$. □

Theorem 1 shows that edge deletion reduces noise ratio under a mild condition on the classification accuracy of the GNN model, i.e., $p > \frac{2}{c+1}$. For example, for a node classification problem with 5 classes, it only requires the classification accuracy $p > 1/3$. To analyze the expected noise ratio of the graph after edge addition, we further assume that *the classes are balanced*, i.e., each class has n/c nodes.

THEOREM 2. *(Edge Addition) If Assumption 2 holds, the classes are balanced in $\mathcal{G}$, and Algorithm 2 is used for edge addition, denote the expected noise ratio of the output graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ as α_E, we have $\alpha_E < \alpha$ if $p > \frac{\alpha+\sqrt{\alpha^2+[(c-1)(1+c\alpha\lambda)-c\alpha](c+\alpha-1)}}{c+\alpha-1}$, in which $\lambda = \frac{m}{n^2}$ is the edge density of the graph.*

PROOF. Denote the expected number of added intra-class edges as m'_a and the expected number of added inter-class edges as m'_r. To ensure $\alpha_E < \alpha$, it suffices to show that $\frac{m'_r}{m'_a + m'_r} < \alpha$. As there are $\frac{c-1}{c}n^2$ possible inter-class edges and $\frac{1}{c}n^2$ intra-class edges in $\mathcal{V} \times \mathcal{V}$, we have

$$
m'_r = \left(\frac{c-1}{c}n^2 - m\alpha\right)p_r < \frac{c-1}{c}n^2 p_r
$$

$$
m'_a = \left[\frac{1}{c}n^2 - m(1-\alpha)\right]p_a > \frac{1}{c}n^2 p_a - m,
$$

where p_r and p_a are the probability of keeping an inter-class edge and an intra-class edge in $\mathcal{G}'$, respectively. Their expressions are given in the proof of Theorem 1. The $m\alpha$ and $m(1-\alpha)$ terms are included to exclude the overlaps between the edges in the original graph and the edges that may be added by Algorithm 2. With $m = n^2\lambda$, we have

$$
\frac{m'_r}{m'_a + m'_r} < \frac{\frac{c-1}{c}n^2 p_r}{\frac{c-1}{c}n^2 p_r + \frac{1}{c}n^2 p_a - m} < \frac{1 - p^2}{1 + \frac{(1-p)^2}{c-1} - c\lambda}.
$$

Solving $\frac{1-p^2}{1+\frac{(1-p)^2}{c-1}-c\lambda} < \alpha$ gives the result. □

The bound on p in Theorem 2 is complex for interpretation but we can approximate it as $p > \frac{\alpha+\sqrt{(c-1)(c-1-c\alpha)}}{c-1}$ if we assume that the λ term is small enough to be ignored and α is very small compared to c. The bound can be further simplified as $p > \sqrt{1-\alpha}$ if we assume that $\alpha/(c-1)$ is small and approximate $c - 1 - c\alpha$ with $(c-1)(1-\alpha)$. Note that $p > \sqrt{1-\alpha}$ is a higher requirement on the classification accuracy of $f(\cdot)$ than $p > \frac{2}{c+1}$ for edge deletion. Thus, as we will show in the experiments, the performance improvement of edge addition is usually smaller than edge deletion.

Theorem 1 and Theorem 2 can be extended to more general assumptions. For example, the symmetric error assumption can be replaced with an error matrix $E \in \mathbb{R}^{c \times c}$, where $E(i, j)$ is the probability of classifying class i as class j. The number of nodes in each class can also be different. The analysis methodology in the proofs can still be applied but the bounds will be in more complex forms. In addition, we show in the experiments that edge deletion and addition can be conducted simultaneously.

2.2 Optimizations for TU

For practical topology update, we use the following techniques to improve Algorithm 1 and Algorithm 2.

Threshold-based selection. The GNN model $f(\cdot)$ usually outputs a distribution on the classes (e.g., using softmax) rather than a single decision. For a node v, we denote its class distribution provided by the model as $g_v \in \mathbb{R}^c$ with $g_v[k] \geq 0$ for $k \in [c]$ and $\sum_{k=1}^c g_v[k] = 1$. For edge deletion, we first generate a candidate edge set C based on the classification labels using Algorithm 1. For each candidate edge e_{uv} in C, we calculate the correlation between their class distributions (i.e., $g_v^\top g_u$) and select the edges with $g_v^\top g_u \leq \tau_d$ for actual deletion, where τ_d is a threshold. For edge addition, we also generate a candidate set using Algorithm 2 first and add only edges with $g_v^\top g_u \geq \tau_a$. Moreover, we constrain the number of added edges to be less than 2 times of the edges in the original graph to avoid
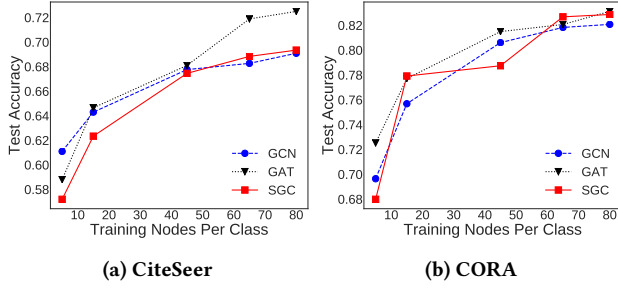
(a) CiteSeer

(b) CORA

Figure 2: The relation between the number of training samples and test accuracy for GCN, GAT and SGC

making the cost of model training too high [2]. Threshold-based selection makes Algorithm 1 and Algorithm 2 more conservative and it also helps to avoid deleting intra-class edges and adding inter-class edges.

Validation-based tuning. We use the validation set to tune the thresholds τ_d and τ_a. For each threshold, we use it to make the topology update decisions and generate a new graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$. Then we train a GNN model on the updated graph and test its accuracy on the validation set. A number of candidate thresholds are checked and the one that provides the best validation accuracy is adopted. Validation-based tuning allows us to reject topology update (by setting $\tau_d = 0$ and $\tau_a = 1$) when it cannot improve performance, e.g., the noise ratio of the graph is already very low or the accuracy of the model $f(.)$ is not good enough.

Efficiency issue. For edge addition, naively computing the label correlation $g_v^\top g_u$ for all $(n/c)^2/2$ possible node pairs incurs high complexity, especially for large graphs. Therefore, for each node v, we only find the top-k nodes (e.g., 2 or 3) that have the largest label correlation with v and use them as the candidates for edge addition. This corresponds to the well-known all-pair maximum inner product search problem, for which there are many efficient solutions such as LEMP [24] and FEXIPRO [12].

3 TRAINING NODE AUGMENTATION

Motivation. In Figure 2, we experiment the influence of the number of training nodes on classification accuracy. The results show that using more training nodes consistently leads to higher classification accuracy for GCN, GAT and SGC. Unfortunately, for the semi-supervised node classification problem, usually only a very small number of labeled nodes are available. To enlarge the training set, an intuitive idea is to train a GNN model to label some nodes and add those nodes to the training set. However, a GNN model usually makes a considerable amount of errors in its label prediction, and naively using the predicted labels as the ground-truth labels may lead to worse performance.

---

[2] The cost of GNN training is proportional to the number of edges.

3.1 Training Node Augmentation Algorithm

For a GNN model $g(\cdot)$ that outputs a distribution on c classes, we define the *confidence* (c_v) and *prediction result* (r_v) of node v as

$$c_v = \max_{1 \le k \le c} g_v[k] \quad \text{and} \quad r_v = \arg \max_{1 \le k \le c} g_v[k],$$

where r_v is the label of v predicted by $g(\cdot)$ and c_v is the likelihood of r_v. Usually r_v is more likely to be correct (i.e., $r_v = l(v)$) when c_v is large (we show this in Figure 6, Appendix B). Utilizing c_v and r_v, we present the training node augmentation (TNA) procedure in Algorithm 3, which produces an enlarged training set $\mathcal{T}'$ using the outputs of multiple GNN models. In Algorithm 3, $\mathcal{T}$ and $\mathcal{S}$ denote the original training set and validation set. Before adding a node to $\mathcal{T}'$, we check if it is already in $\mathcal{T}$ and $\mathcal{S}$ to avoid assigning a new label to nodes in the two sets.

Algorithm 3: Training Node Augmentation

Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and L trained GNN models $g^1, g^2, \cdots, g^L$
Output: An enlarged training set $\mathcal{T}'$
Initialize $\mathcal{T}' = \emptyset$;
for each model g^l **do**
　Initialize candidate set $C^l = \emptyset$;
　for each node v in $\mathcal{G}$ **do**
　　if $c_v^l \ge \tau_c$ **then**
　　　Add v to C^l;
　　end if
　end for
end for
Candidate set $C = \cap_{l=1}^L C^l$;
for each node v in C **do**
　if $v \notin \mathcal{T}$, $v \notin \mathcal{S}$ and $r_v^1 = r_v^2 = \cdots = r_v^L$　**then**
　　Add v to $\mathcal{T}'$ with label r_v^1;
　end if
end for

Algorithm 3 is based on two key ideas. The first one is only considering nodes with a high confidence (i.e., $c_v^l \ge \tau_c$) as the candidates to be added to $\mathcal{T}'$ since GNN models tend to produce more accurate label predictions at higher confidence. Similar to the case of topology update, we tune the value of τ_c based on the accuracy (of the model trained using $\mathcal{T} \cup \mathcal{T}'$) on the validation set. The second and most important idea is to utilize the *diversity* of multiple GNN models to reduce the number of errors in $\mathcal{T}'$. With multiple diverse models, even if some classifiers assign a wrong label to node v, it will not be added to $\mathcal{T}'$ as long as one classifier gives the right label. In the following, we formalize this intuition with an analysis under the case of using two GNN models g^1 and g^2, i.e., $L = 2$.

Following Assumption 2, we assume that both g^1 and g^2 have a classification accuracy of p and make symmetric error. We also simplify Algorithm 3 and assume that a node is added to $\mathcal{T}'$ if the two models give the same label (i.e., $r_v^1 = r_v^2$). Algorithm 3 can be viewed as a special case of this simplified algorithm with $p' > p$ as it adds high-confidence nodes. The accuracy of $\mathcal{T}'$ is defined as $q = \frac{|\{v \in \mathcal{T}' | l(v) = r_v^1 = r_v^2\}|}{\mathcal{T}'}$. We are interested in the relation between

p and q, which are the accuracies of $\mathcal{T}'$ when using one model and two models for TNA, respectively. As the two models are trained on the same graph structure, it is unrealistic to assume that they are independent. Therefore, we make the following assumption on how they correlate.

ASSUMPTION 3. *(Model Correlation) The correlation between the two GNN models g^1 and g^2 can be formulated as follows*

$$
\begin{cases} \mathbb{P}[r_v^2 = l(v)|r_v^1 = l(v)] = \beta \\ \mathbb{P}[r_v^2 = k|r_v^1 = l(v)] = \frac{1-\beta}{c-1} \end{cases} \text{ and } \begin{cases} \mathbb{P}[r_v^2 = l(v)|r_v^1 = k] = \gamma \\ \mathbb{P}[r_v^2 = k|r_v^1 = j] = \frac{1-\gamma}{c-1} \end{cases},
$$

where $k \in [c]$ and $k \neq l(v)$, $j \in [c]$ and $j \neq l(v)$. We also assume that $\beta \geq p$ as the two models should be positively correlated.

THEOREM 3. *(Train Set Accuracy) Under Assumption 3 and assume that $p > 1/2$, we have the following results on the accuracy q of $\mathcal{T}'$*
   (1) $q \geq p$;
   (2) q is maximized when $\beta = \gamma = p$, in which case the two models g^1 and g^2 are independent.

PROOF. The probability that g^2 gives the right label can be expressed as

$$
\mathbb{P}[r_v^2 = l(v)] = \mathbb{P}[r_v^1 = l(v)] \cdot \mathbb{P}[r_v^2 = l(v)|r_v^1 = l(v)] + \sum_{k \neq l(v)} \mathbb{P}[r_v^2 = l(v)|r_v^1 = k] \cdot \mathbb{P}[r_v^1 = k].
$$

We assume that g^2 has a classification accuracy of p and solving $\mathbb{P}[r_v^2 = l(v)] = p$ gives the relation between β and γ as $p\gamma = p\beta + \gamma - p$. We can express q as

$$
\begin{aligned}
q &= \frac{\mathbb{P}[r_v^1 = l(v), r_v^2 = l(v)]}{\mathbb{P}[r_v^1 = l(v), r_v^2 = l(v)] + \sum_{k \neq l(v)} \mathbb{P}[r_v^1 = k, r_v^2 = k]} \\
&= \frac{(c-1)p\beta}{(c-1)p\beta + (1-p)(1-\gamma)}.
\end{aligned}
$$

Substituting $p\gamma = p\beta + \gamma - p$ into the above expression gives $q = \frac{(c-1)p\beta}{cp\beta + 1 - 2p}$. Solving $q \geq p$ gives the following result

$$
\begin{cases} \beta \geq 0 & \text{for } p \leq 1 - \frac{1}{c} \\ 0 \leq \beta \leq \frac{1-2p}{c-1-cp} & \text{for } p > 1 - \frac{1}{c} \end{cases}.
$$

It can be verified that $\frac{1-2p}{c-1-cp} \geq 1$ when $1 - \frac{1}{c} < p \leq 1$. Therefore, we have $q \geq p$ regardless of the value of p and β, which proves the first part of the theorem. For the second part of theorem, we have

$$
\frac{\partial q}{\partial \beta} = \frac{p(c-1)(1-2p)}{(cp\beta + 1 - 2p)^2}.
$$

As $p > 1/2$, q is a decreasing function of β. As $\beta \geq p$, q is maximized when $\beta = p$. In this case, we can obtain that $\gamma = p$ by solving $p\gamma = p\beta + \gamma - p$. $\beta = \gamma = p$ shows that $\mathbb{P}[r_v^2 = l(v)]$ does not depend on r_v^1, which means that the two models are independent. □

Theorem 3 shows that using two models improves the accuracy of $\mathcal{T}'$ over using a single model. Theorem 3 also indicates that we should make the GNN models as independent as possible to maximize the accuracy of $\mathcal{T}'$.
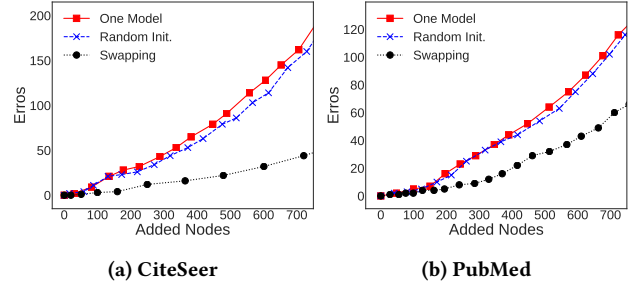

(a) CiteSeer **(b) PubMed**

Figure 3: Errors in $\mathcal{T}'$ using 1 model, 2 models with different random initializations, and 2 models with train-validation set swapping (the model is GCN)

3.2 Optimizations for TNA

Creating diversity in GNN models. A straightforward method to generate multiple different GNN models is *random initialization*, which trains the same model with different parameter initializations. We show the number of errors (i.e., nodes with wrong labels) in $\mathcal{T}'$ using random initialization and under different threshold τ_c (adjusting τ_c controls the number of added nodes) in Figure 3. The results show that using 2 models, random initialization does not significantly outperform a single model. We conducted detailed profiling and found that this is because the two models lack diversity. For example, two randomly initialized models provide the same label prediction for 2,900 nodes (out of a total number of 3,327 nodes) on the CiteSeer dataset and the prediction accuracy in these agreed nodes is 71.9%. We found that this phenomenon is consistent across different GNN models and datasets. It is observed that GNN models resemble label propagation algorithm in some sense [26] and the results of label propagation are totally determined by the graph structure and the labeled nodes. Therefore, two GNN models trained with different random initializations tend to produce the same label prediction because they use the same graph structure and training set.

Motivated by this finding, we propose to generate multiple GNN models with better diversity using *train set swapping*, which randomly re-partitions the visible set (training and validation set, i.e., $\mathcal{T} \cup \mathcal{S}$) for each model. Train set swapping first unites the original training set $\mathcal{T}$ and validation set $\mathcal{S}$. Then $|\mathcal{T}|$ nodes in the visible set are randomly selected as the training set for a model and the remaining samples go to the validation set. The motivation is to use a different training set to train each GNN model for better diversity. We plot the errors in the $\mathcal{T}'$ produced by train set swapping in Figure 3. The results show that train set swapping generates significantly fewer errors than random initialization when adding the same number of nodes. This is because the models have better diversity than random initialization and they agree on the label prediction of only 2,230 nodes on the CiteSeer dataset. The prediction accuracy in the agreed nodes is 85.4%, which is significantly higher than the 71.9% accuracy for random initialization.

Class balance. A trick that is crucial for the performance of TNA is ensuring that each class has a similar number of nodes in the enlarged training set $\mathcal{T}'$. We observed that different classes can have

Table 1: Performance results for self-enhanced GNN (abbreviated as SEG), where *Error Reduction* is the percentage of classification error reduced from the respective base model (i.e., GCN, GAT, and SGC)

	CORA	CiteSeer	PubMed	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN	78.7±1.5	66.5±2.4	75.5±1.8	90.7±0.6	93.1±0.5	71.9±12.8	85.2±10.0
GCN+SEG	82.3±1.2	71.1±0.8	80.0±1.4	92.9±0.4	93.9±0.2	80.2±6.5	90.4±0.9
Error Reduction	**16.9%**	**13.7%**	**18.4%**	**23.7%**	**11.6%**	**29.5%**	**35.1%**
GAT	79.0±1.7	65.7±1.9	75.3±2.4	89.9±0.6	92.0±0.8	82.2±2.1	89.6±1.8
GAT+SEG	81.4±1.3	70.0±1.0	78.9±1.4	91.6±0.5	93.5±0.4	83.7±0.7	90.8±1.4
Error Reduction	**11.4%**	**12.3%**	**14.6%**	**16.8%**	**18.8%**	**8.4%**	**11.5%**
SGC	77.4±2.6	65.0±2.0	73.3±2.6	91.3±0.6	93.3±0.3	81.1±2.0	89.3±1.4
SGC+SEG	82.2±1.3	70.2±0.9	78.1±2.3	93.1±0.2	94.1±0.4	82.8±1.7	89.9±0.8
Error Reduction	**21.2%**	**14.9%**	**8.5%**	**16.8%**	**18.8%**	**9.0%**	**7.7%**

a very different number of nodes. For example, for the Coauthor CS dataset, the number of nodes in the largest class is 4.78x that of the smallest class. If we assume that every node has the same probability of being added to $\mathcal{T}'$, the large classes can have significantly more training samples than the small classes. We found that TNA can even degrade the accuracy (compared to without TNA) in this case. We conjecture that this is because an unbalanced training set encourages the GNN model to label nodes as from the large classes, which does not generalize. Therefore, we constrain each class to have the same number of nodes in $\mathcal{T}'$. If the number of nodes to be added to $\mathcal{T}'$ for a class is larger than that for the smallest class, then we add only the nodes with the largest confidence for this class.

4 EXPERIMENTAL RESULTS

Settings. The experiments were conducted on seven widely used benchmark datasets for node classification. Due to the space limitation, we give the statistics of the datasets in Table 6, Appendix A.1. We evaluated the performance of *topology update* (*TU*) and *training node augmentation* (*TNA*) on three well-known GNN models, i.e., GCN [11], GAT [25] and SGC [27]. We configured all the three models to have two layers because GNN models usually perform the best with two layers due to over-smoothing [19] and increasing the layers also increases the computation cost exponentially due to neighbor propagation. All weights for the models were initialized according to Glorot and Bengio [5] and all biases were initialized as zeros. The models were trained using the Adam [10] optimizer and the learning rate was set to 0.01. For both TU and TNA, we utilized a grid search to tune their parameters (i.e., the thresholds τ_d, τ_a and τ_c) *on the validation set*. The detailed settings of other hyper-parameters can be found in Appendix A.2.

We followed the evaluation protocol proposed by Shchur et al. [23] and recorded the average classification accuracy and standard deviation of 10 different dataset splits. For each split, 20 and 30 nodes from each class were randomly sampled as the training set and validation set, respectively, and the other nodes were used as the test set. Under each split, we ran 10 random initializations of the model parameters and used the average accuracy of the 10 initializations as the performance of this split. The motivation of this

evaluation protocol was to exclude the influence of the randomness in data split on performance, which was found to be significant.

4.1 Overall Performance Results

We first present the overall performance results of self-enhanced GNN (abbreviated as **SEG**) in Table 1. The reported performance of SEG is the best performance that can be obtained using TU, TNA, or (TU + TNA). In practice, we may choose to use TU, TNA, or (TU + TNA) by their prediction accuracy on the validation set. The results in Table 1 show that SEG consistently improves the performance of the 3 GNN models on the 7 datasets, where the reduction in classification error is 16.2% on average and can be as high as 35.1%. The result is significant particularly because it shows that SEG is an effective, general framework that improves the performance of well-known models that are already recognized to be effective.

In the subsequent subsections, we analyze the performance of TU and TNA individually, as well as examine how they influence data quality.

4.2 Results for Topology Update

The performance results of TU are reported in Table 2. To control the complexity of parameter search, we constrained the number of added edges to be the same as the number of deleted edges for *Modify*. The following observations can be made from the results in Table 2.

First, TU improves the performance of GCN, GAT and SGC in most cases and the improvement is significant in some cases. For example, the error reduction is over 25% for GCN on the Amazon Photo dataset. The error reduction is zero in 4 out of the 63 cases because threshold tuning (for τ_d and τ_a) on the validation set rejects TU as it cannot improve the performance. Thus, even in the worst case, TU does not degrade the performance of the base models.

Second, edge deletion generally achieves greater performance improvements than edge addition. This is because there is a large number of possible inter-class edges (e.g., n^2/c when the classes are balanced). Even if the probability of adding an inter-class edge is small (the same as the probability of keeping an inter-class edge in $\mathcal{G}$ in edge deletion), the algorithm may still add a considerable number of inter-class edges in expectation.

Table 2: Performance results of TU, where *Delete* refers to edge deletion, *Add* refers to edge addition, and *Modify* refers to conducting both edge deletion and addition. *Error Reduction* is the percentage of classification error reduced from the respective base model (i.e., GCN, GAT, and SGC).

	CORA	CiteSeer	PubMed	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN+Delete	79.2±1.6	66.5±2.4	75.6±2.0	91.8±0.6	93.2±0.6	80.1±2.1	89.0±2.5
Error Reduction	**2.3%**	**0.0%**	**0.4%**	**11.8%**	**1.4%**	**29.2%**	**25.7%**
GAT+Delete	79.3±1.8	65.8±1.9	75.3±2.6	90.9±0.9	92.2±0.7	82.8±2.1	90.3±1.5
Error Reduction	**1.4%**	**0.3%**	**0.0%**	**9.9%**	**2.5%**	**3.4%**	**4.9%**
SGC+Delete	77.8±2.1	65.5±2.4	73.6±2.7	92.6±0.4	93.5±0.4	82.0±2.0	89.6±1.4
Error Reduction	**1.8%**	**1.4%**	**1.1%**	**14.9%**	**3.0%**	**4.8%**	**2.8%**
GCN+Add	78.8±1.7	66.8±2.4	75.6±1.7	90.7±0.7	93.2±0.4	78.9±2.2	88.2±2.3
Error Reduction	**0.5%**	**0.9%**	**0.4%**	**0.0%**	**1.4%**	**24.9%**	**20.3%**
GAT+Add	79.1±1.3	65.7±2.0	75.7±1.8	90.0±0.5	92.1±0.8	82.6±2.5	89.7±0.8
Error Reduction	**0.5%**	**0.0%**	**1.6%**	**1.0%**	**1.2%**	**2.2%**	**1.0%**
SGC+Add	77.5±2.4	65.7±1.7	73.8±2.5	91.5±0.6	93.5±0.4	81.6±1.9	89.4±1.5
Error Reduction	**0.4%**	**2.0%**	**1.9%**	**2.3%**	**3.0%**	**1.6%**	**0.9%**
GCN+Modify	79.4±1.3	67.1±2.2	75.9±2.0	91.7±0.9	93.4±0.3	79.2±2.5	88.5±4.0
Error Reduction	**3.3%**	**1.8%**	**1.6%**	**10.8%**	**4.3%**	**26.0%**	**22.3%**
GAT+Modify	79.1±1.8	65.8±2.1	76.0±2.2	90.7±0.9	92.1±0.9	82.4±2.0	90.1±1.4
Error Reduction	**0.5%**	**0.3%**	**2.8%**	**7.9%**	**1.2%**	**1.1%**	**4.8%**
SGC+Modify	78.5±2.3	66.7±1.6	74.0±2.6	92.7±0.3	93.5±0.3	81.7±2.2	89.4±1.6
Error Reduction	**4.9%**	**4.9%**	**2.6%**	**16.1%**	**3.0%**	**2.1%**	**0.9%**

Third, the performance improvement of TU is relatively smaller for CiteSeer and PubMed than that for the other datasets, which can be explained as follows. The accuracy of GCN, GAT and SGC for CiteSeer and PubMed is considerably lower than that for the other datasets. As a result, the TU algorithms are also more likely to make wrong decisions (i.e., deleting intra-class edges or adding inter-class edges) since TU decisions are guided by the model predictions. Motivated by this observation, we experimented a dual-model edge deletion/addition algorithm on CiteSeer, which uses the intersection of the edge deletion/addition decisions of two GNN models. The intuition is similar to the idea of TNA, which utilizes the diversity of different GNN models to reduce errors. The dual-model algorithm improves the error reduction of single-model edge deletion/addition from 0.0% and 0.9% to 0.6% and 2.1%, respectively.

Fourth, although GCN, GAT and SGC have high accuracy for both Coauthor CS and Coauthor Physics, TU has considerably greater performance improvements on Coauthor CS than on Coauthor Physics. This is because the noise ratio of the original Coauthor Physics graph is much lower than the Coauthor CS graph (6.85% vs. 19.20%), and thus reducing noise ratio has smaller influence on the performance for Coauthor Physics.

Finally, TU generally achieves greater performance improvements for GCN and SGC than for GAT. We plot the distributions of the attention weights of GAT on the edges that are deleted and kept by Algorithm 1 in Figure 4. The results show that the deleted edges have significantly smaller attention weights than the kept edges. As we mainly delete inter-class edges, the results suggest

Table 3: The effect of edge deletion and edge addition on noise ratio for the CORA dataset. For edge deletion, the reported tuple is the number of deleted inter-class edges and intra-class edges, respectively. For edge addition, the reported tuple is the number of added intra-class edges and inter-class edges, respectively. The noise ratio of the original graph is 19.00%.

Model	Edge Deletion	Noise Ratio	Edge Addition	Noise Ratio
GCN	(332, 218)	14.19%	(4692, 85)	10.82%
GAT	(309, 212)	14.59%	(5995, 165)	10.21%
SGC	(242, 116)	15.47%	(3807, 25)	11.28%

that GAT can prevent the inter-class edges from smoothing the embeddings of nodes from different classes by assigning them small weights. This explains why GAT is less sensitive to changes in noise ratio. However, GAT cannot really set the weights of the inter-class edges to 0 as it uses the *softmax* function to compute the attention weights. In contrast, Algorithm 1 can completely remove inter-class edges and can thus even improve the performance of GAT further in most cases.

We also examined the edge deletion and addition decisions made by TU in Table 3. For both edge deletion and addition, we report the number of correct decisions (i.e., removing inter-class edges for deletion and adding intra-class edges for addition) and wrong decisions (i.e., removing intra-class edges for deletion and adding

Table 4: Performance results of TNA, where *Error Reduction* is the percentage of classification error reduced from the respective base model (i.e., GCN, GAT, and SGC).

	CORA	CiteSeer	PubMed	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN+TNA	82.1±1.1	70.6±1.1	80.0±1.4	91.8±0.3	93.7±0.5	80.2±6.5	89.5±2.6
Error Reduction	**16.0%**	**12.2%**	**18.4%**	**11.8%**	**8.7%**	**29.5%**	**29.1%**
GAT+TNA	81.4±1.3	70.0±1.0	78.9±1.4	91.1±0.4	93.4±0.3	82.7±1.7	90.8±1.4
Error Reduction	**11.4%**	**12.3%**	**14.6%**	**11.9%**	**5.7%**	**2.8%**	**11.5%**
SGC+TNA	82.2±1.3	70.2±0.9	73.3±3.2	92.0±0.4	93.9±0.3	82.8±1.7	89.9±1.5
Error Reduction	**21.2%**	**14.9%**	**0.0%**	**8.0%**	**9.0%**	**9.0%**	**5.6%**

(a) Deleted edges (mean=0.17)   (b) Kept edges (mean=0.26)
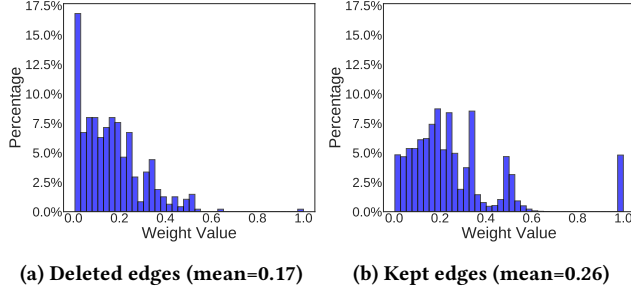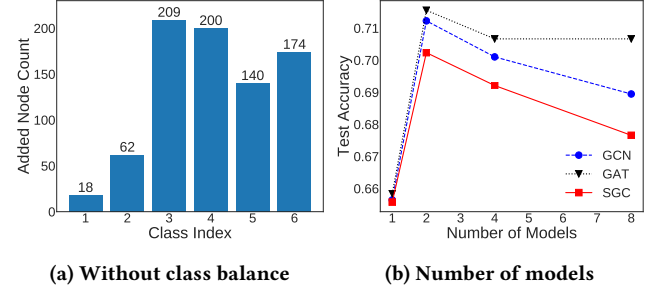
Figure 4: The distribution of GAT attention weights on the edges that are deleted and kept by Algorithm 1 on the CORA dataset

(a) Without class balance   (b) Number of models

Figure 5: Examination of the designs in TNA on CiteSeer

Table 5: The number of nodes added into $\mathcal{T}'$ by TNA and the number of errors (nodes with wrong label) in these added nodes for the CORA dataset

Model	GCN	GAT	SGC
# Added Nodes	826	714	637
# Errors	83	72	45
Error Ratio	10.05%	10.08%	7.06%

inter-class edges for addition), and the noise ratio of the CORA graph after TU. The results show that TU effectively reduces noise ratio. Most of the added edges are intra-class edges and only a few are inter-class edges. Edge deletion effectively removes inter-class edges but a considerable number of intra-class edges are also removed. This is because there are much more intra-class edges in the graph than inter-class edges, and thus the expectation of the number of removed intra-class edges may not be small even if the probability of removing an intra-class edge is small.

4.3 Results for Training Node Augmentation

We present the performance results of TNA in Table 4, which show that TNA improves the performance of GCN, GAT and SGC in 20 out of the 21 cases. The performance improvements are significant in many cases, e.g., 29.1% for GCN on the Amazon Photo dataset. The performance improvements are large on CORA and CiteSeer for all three GNN models. We conjecture that this is because the

two datasets are relatively smaller and thus adding more training samples has a large impact on the performance. To explain the good performance of TNA, we examined the number of added nodes and the errors in the added nodes in Table 5. The results show that most of the added nodes are assigned the correct label. Compared with GAT and GCN, a small number of nodes are added for SGC and the error ratio is also lower. This may be because the model of SGC is simpler than GAT and GCN (without nonlinearity) and thus SGC is more sensitive to noise in the training samples.

We examined the two important designs in TNA, i.e., class balance and multi-model diversity. We experimented with a version of TNA without class balance for GCN on the Amazon Photo dataset, which records a classification accuracy of 86.67%. In contrast, the classification accuracy with class balance is 89.54% as reported in Table 4. We plot in Figure 5a the class distribution of the nodes added by TNA without class balance, which shows that the number of nodes in the largest class is 11.6 times of the smallest class. The results show that without class balance, the enlarged training set can be highly screwed.

To demonstrate the benefits of using the diversity of multiple models in TNA, we report the relation between the test accuracy and the number of models (used for node selection) on the CiteSeer dataset in Figure 5b. The result show that using 2 models provide a significant improvement in classification accuracy over 1 model, but the improvement drops when using more models. This is because more models are difficult to agree with each other and thus a low confidence threshold (i.e., τ_c) needs to be used to add a good number of nodes. However, a low confidence threshold means that the added nodes are likely to contain errors.

5 RELATED WORK

GNN models. Many GNN models have been proposed in recent years, including GCN [11], GAT [25], SGC [27], GraphSAGE [7], Geom-GCN [21], GGNN [14], JK-Net [28], ChebNet [3], Highway GNN [31] and MoNet [17]. These works focus on improving the performance of a task, e.g., the prediction accuracy of node classification, comparing with prior methods. In contrast, our method, self-enhanced GNN, aims to improve the quality of the input data. By providing data with higher quality, self-enhanced GNN provides a general framework that can be easily applied on existing GNN models to further improve their performance.

To the best of our knowledge, our work is most related to Chen et al. [1] and Li et al. [13]. Chen et al. [1] observed that the performance of GNN models usually degrades when using more than 2 layers due to local smoothing and proposed to remove/add edges in a graph to mitigate the over-smoothing problem of GNN models. In contrast, we come from the perspective of data quality and observe that lower noise ratio leads to higher classification accuracy. In addition, we provide theoretical analysis to show that adding/removing edges can reduce noise ratio if the performance of a model is good enough. The idea of enlarging the training set with co-training and self-training was proposed in [13], which corresponds to the single-model case of our training node augmentation algorithm. However, as we have shown in our analysis and profiling results in Section 3.2, using the diversity of multiple models and explicitly balancing the classes in the training set are crucial for performance. In fact, the results reported in [13] also show that the performance of GNN (e.g., GCN) actually degrades in many cases when applying co-training and self-training with a single model.

Noisy label training. Self-enhanced GNN is partly motivated by noisy label training, which aims at learning good models from data with noisy labels, i.e., a large number of training samples come with wrong labels. Representative work along this line include Decoupling [15], MentorNet [9], Noisy Cross-Validation [2] and Co-teaching [8]. These works focus on how to select samples with possibly correct labels from a noisy dataset to conduct model training, and our multi-model sample selection method is motivated by these works. However, as GNNs work on graph data, self-enhanced GNN handles not only noise in labels but also noise in the graph structure (i.e., inter-class edges) with topology update. Given the excellent performance of GNNs on graph data, a potential direction is to apply self-enhanced GNN to noisy label training. With the assumption that samples with similar features are likely to share the same label, a similarity graph (e.g., a k-nearest-neighbor graph based on image descriptors) can be constructed on a noisy dataset and noisy label training can be modeled as a semi-supervised node classification problem on graphs.

6 CONCLUSIONS

We presented self-enhanced GNN. The main idea is to improve the quality of the input data using the outputs of existing GNN models, so that the proposed method can be used as a general framework to improve the performance of different existing GNN models. Two algorithms were developed under this idea, i.e., *topology update*, which deletes/adds edges to remove inter-class edges and add potential intra-class edges in an input graph, and *training node augmentation*, which enlarges the training set by adding nodes with high classification confidence. Theoretical analyses were provided to motivate the algorithm designs and comprehensive experimental evaluation was conducted to validate the performance of the algorithms. The results show that self-enhanced GNN is an effective general framework that consistently improves the performance of different GNN models on a broad set of datasets.

REFERENCES

[1] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2019. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv preprint arXiv:1909.03211* (2019).

[2] Pengfei Chen, Ben Ben Liao, Guangyong Chen, and Shengyu Zhang. 2019. Understanding and Utilizing Deep Neural Networks Trained with Noisy Labels. In *International Conference on Machine Learning*. 1062–1070.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[4] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Yee Whye Teh and Mike Titterington (Eds.), Vol. 9. PMLR, Chia Laguna Resort, Sardinia, Italy, 249–256. http://proceedings.mlr.press/v9/glorot10a.html

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[8] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*. 8527–8537.

[9] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2017. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *arXiv preprint arXiv:1712.05055* (2017).

[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[12] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 835–850.

[13] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[14] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[15] Eran Malach and Shai Shalev-Shwartz. 2017. Decoupling" when to update" from" how to update". In *Advances in Neural Information Processing Systems*. 960–970.

[16] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 43–52.

[17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5115–5124.

[18] Hoang NT and Takanori Maehara. 2019. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. arXiv:stat.ML/1905.09550

[19] Kenta Oono and Taiji Suzuki. 2019. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. arXiv:cs.LG/1905.10947

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach,

H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[21] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1e2agrFvS

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[23] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).

[24] Christina Teflioudi and Rainer Gemulla. 2016. Exact and approximate maximum inner product search with lemp. *ACM Transactions on Database Systems (TODS)* 42, 1 (2016), 1–49.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=rJXMpikCZ

[26] Hongwei Wang and Jure Leskovec. 2020. Unifying Graph Convolutional Neural Networks and Label Propagation. https://openreview.net/forum?id=rkgdYhVtvH

[27] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 6861–6871.

[28] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* (2018).

[29] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[30] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[31] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 4189–4198.

A DETAILS OF EXPERIMENTAL EVALUATION

All the code of this work is released via the following anonymous link [3] and will be open source later. The datasets used in the experiments have been widely used for the evaluation of GNN models and they are all publicly available.

A.1 Models and Datasets

We evaluated our methods on 3 popular GNN models, i.e., GCN [11], GAT [25] and SGC [27]. We used 7 datasets to evaluate our methods. Among them, CORA, CiteSeer and PubMed are 3 well known citation networks and we used the version provided by Yang et al. [29]. Amazon Computers and Amazon Photo are derived from the Amazon co-purchase graph in McAuley et al. [16]. Coauthor CS and Coauthor Physics are obtained from the Microsoft Academic Graph for the KDD Cup 2016 challenge [4]. For these 4 datasets, we used the version pre-processed by Shchur et al. [23]. The statistics of the datasets are summarized in Table 6, where α is the *noise ratio* defined in Section 2.

Table 6: Datasets Statistics

	Classes	Features	Nodes	Edges	α
CORA	7	1,433	2,485	5,069	0.19
CiteSeer	6	3,703	2,110	3,668	0.26
PubMed	3	500	19,717	44,324	0.19
Coauthor CS	15	6,805	18,333	81,894	0.19
Coauthor Physics	5	8,415	34,493	247,962	0.06
Amazon Computers	10	767	13,381	245,778	0.22
Amazon Photo	8	745	7,487	119,043	0.17

A.2 Implementation Details

Evaluation protocol. To eliminate the influence of random factors and ensure that the performance comparison is fair, we adopted the evaluation protocol provided by Shchur et al. [23]. A 20/30/rest split for train/val/test set was used for all the datasets. In the experiments, we evaluated each model on 10 randomly generated dataset splits, and under each split, we ran the model for 10 times using different random seeds. We reported the mean value and standard deviation of the test accuracies across the 100 runs for each model on each dataset. For the experiments comparing Self-Enhanced GNN with the base GNN models (i.e., GCN, GAT and SGC), all model implementation and evaluation settings were kept fixed and identical.

Structure of the base models. Our GCN model implementation has 2 GCN convolutional layers with a hidden size of 16. The activation function is *ReLU*. A dropout layer with a dropout rate of 0.5 is used after the first GCN layer. Our GAT model implementation has 2 GAT layers with an attention coefficient dropout probability of 0.6. The first layer is an 8-heads attention layer with a hidden size of 8. The second layer has a hidden size of 8×8. The activation function is *ELU*. Two dropout layers with a dropout rate of 0.6 are used between the input layer and the first GAT layer, and between the first GAT layer and the second GAT layer. Our SGC

---

model implementation has a SGC convolutional layer with 2 hops (equivalent to 2 SGC layers according to the SGC definition).

Model training. We used the Adam optimizer [10] with a learning rate of 0.01 and an L_2 regularization coefficient of $5e^{-4}$. We did not use learning rate decay and early stopping. As the difficulty of model training varies for different datasets, we used a different number of training epochs for each dataset, i.e., CORA 400 epochs, CiteSeer 400 epochs, PubMed 400 epochs, Amazon Computers 1000 epochs, Amazon Photo 2000 epochs, Coauthor CS 2000 epochs and Coauthor Physics 400 epochs.

Software. All models and algorithms in the experiments are implemented on PyTorch [20] and PyTorch-Geometric [4]. The software versions are python=3.6.9, torch=1.2.0, CUDA=10.2.89, pytorch_geometric=1.3.2.

Topology update. For *Delete*, before edge deletion, we remove all self-loop edges in the original graph. Then the edges are deleted according to Algorithm 1 with a threshold. After edge deletion, we add back the removed self-loop edges. For *Add*, we constrain the number of added edges to be less than 4 times of the number of edges in the original graph. This threshold is used to decide the number of candidate edges for addition, i.e., k. We get the top-k edges from the $n \times n$ potential edges according to the label correlation (i.e., $g_v^\top g_u$). After filtering the edges already in the graph, we add new edges using Algorithm 2. For *Modify*, we constrain the total number of added edges to be the same as the number of deleted edges because tuning the parameters for edge deletion and addition jointly will result in high complexity. This constraint also helps maintain the graph topology to some degree by not changing the structure too much. We conduct edge deletion first, and then add the same number of edges as that of the deleted edges. We ensure that deleted edges will not be added back.

Training node augmentation. For training node augmentation, we use two models trained with swapped training and validation set to label the nodes in the test set. Only the nodes having the same label prediction from the two models can be added to the augmented training set. A confidence threshold is used to control the number of pre-selected nodes for addition. We count the number of nodes from each class in the pre-selected nodes and obtain the class with the minimum number of pre-selected nodes. This number is used to control the number of added nodes for all classes (i.e., the class balance trick) to avoid introducing additional biases.

Joint use of TU and TNA. For experiments that jointly used topology update and training node augmentation, we applied the two techniques independently and used the thresholds selected by each algorithm individually to avoid the high complexity of joint parameter tuning. Denote the optimal parameter for topology update and training node augmentation as τ_{tu} and τ_{tna}, respectively. We considered three configurations, i.e., $(\tau_{tu}, 0)$, $(0, \tau_{tna})$ and (τ_{tu}, τ_{tna}) (setting the $\tau = 0$ means disabling the algorithm) and selected the best configuration using the validation accuracy. The reported results is the test accuracy of the selected configuration. Therefore, our framework still has the potential to perform even better if more fine-grained tuning on the thresholds parameters are conducted.

---

[3] https://gofile.io/?c=h0S6ya
[4] https://www.kdd.org/kdd-cup/view/kdd-cup-2016

All the thresholds mentioned above are determined totally by the classification accuracy on the validation set.

B ADDITIONAL EXPERIMENTAL RESULTS

Relation between confidence and classification accuracy. In Algorithm 3, we only add nodes with a high confidence c_v into the enlarged training set $\mathcal{T}'$. In Figure 6, we plot the relation between confidence and classification accuracy. The results show that the model is more likely to give the right label prediction under high confidence.
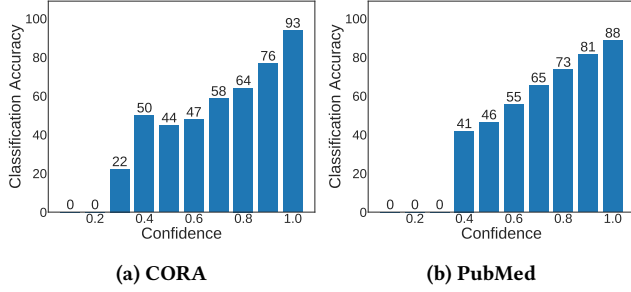

(a) CORA

(b) PubMed

Figure 6: The relation between the confidence score c_v and the probability of giving the right label prediction for GCN

Relation between label correlation and label alignment. Recall that the label correlation between a pair of nodes u and v is defined as $g_v^\top g_u$, in which g_u is the class distribution for node u predicted by a model. For topology update, we delete edges with small label correlation and add edges with large label correlation. In Figure 7, we plot the relation between label correlation and the probability that a pair of nodes have the same label (called node alignment). The results show that a pair of nodes is more likely to be in the same class under higher label correlation.
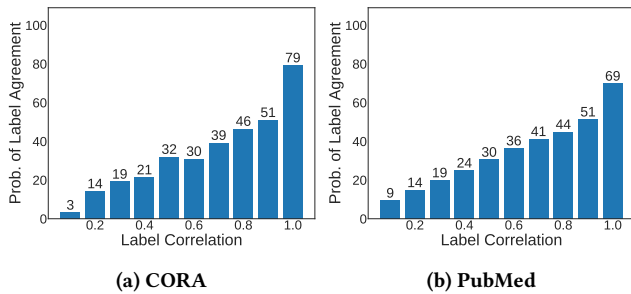

(a) CORA

(b) PubMed

Figure 7: The relation between label correlation (i.e., $g_v^\top g_u$) and the probability of having the same label for GCN