J Inf Process Syst, Vol.20, No.4, pp.501~513, August 2024
https://doi.org/10.3745/JIPS.04.0315
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

GBGNN: Gradient Boosted Graph Neural Networks

Eunjo Jang and Ki Yong Lee*

Abstract

In recent years, graph neural networks (GNNs) have been extensively used to analyze graph data across various domains because of their powerful capabilities in learning complex graph-structured data. However, recent research has focused on improving the performance of a single GNN with only two or three layers. This is because stacking layers deeply causes the over-smoothing problem of GNNs, which degrades the performance of GNNs significantly. On the other hand, ensemble methods combine individual weak models to obtain better generalization performance. Among them, gradient boosting is a powerful supervised learning algorithm that adds new weak models in the direction of reducing the errors of the previously created weak models. After repeating this process, gradient boosting combines the weak models to produce a strong model with better performance. Until now, most studies on GNNs have focused on improving the performance of a single GNN. In contrast, improving the performance of GNNs using multiple GNNs has not been studied much yet. In this paper, we propose gradient boosted graph neural networks (GBGNN) that combine multiple shallow GNNs with gradient boosting. We use shallow GNNs as weak models and create new weak models using the proposed gradient boosting-based loss function. Our empirical evaluations on three real-world datasets demonstrate that GBGNN performs much better than a single GNN. Specifically, in our experiments using graph convolutional network (GCN) and graph attention network (GAT) as weak models on the Cora dataset, GBGNN achieves performance improvements of 12.3%p and 6.1%p in node classification accuracy compared to a single GCN and a single GAT, respectively.

Keywords

Ensemble Method, Gradient Boosting, Graph Neural Network

1. Introduction

A graph is a data structure composed of nodes and edges connecting the nodes. Graphs are widely used to model various real-world structures such as social networks, molecular structures, and citation networks. In recent years, graph neural networks (GNNs) have gained significant attention for their powerful capabilities in learning complex graph-structured data [1]. In the field of computer vision, GNNs have been utilized for image segmentation, object detection, and scene understanding. In these applications, elements and their spatial relationships in an image are represented as a graph, and a GNN is used to analyze (e.g., classify) the elements based on their relationships [2]. Similarly, in recommendation systems, users and their purchased items are represented as a graph, and a GNN is employed to capture interactions between users and items and provide personalized recommendations [3]. Moreover, GNNs have demonstrated remarkable ability in graph analysis tasks like classification of

Manuscript received February 23, 2023; first revision June 23, 2023; accepted July 23, 2023.
* **Corresponding Author:** Ki Yong Lee (kiyonglee@sookmyung.ac.kr)
Dept. of Computer Science, Sookmyung Women's University, Seoul, Korea (wkddmswh99@sookmyung.ac.kr, kiyonglee@sookmyung.ac.kr)

nodes, clustering of nodes or subgraphs, and prediction of links [4,5]. However, recent studies on GNNs have focused on improving the performance of a single GNN with only two or three layers [6-8]. This is because stacking layers deeply causes the over-smoothing problem of GNNs [9]. Each layer in a GNN updates the representation of each node by aggregating the representations of the node and its neighboring nodes. Therefore, if we stack layers many times, the representations of all nodes in the graph would converge to similar values and make them indistinguishable. For this reason, the over-smoothing problem makes the performance of a GNN difficult to improve by stacking many layers.

On the other hand, ensemble methods combine individual weak models to obtain a stronger model. Combining predictions from multiple models has been shown to be an effective approach to increase the performance of the models. Ensemble methods are largely categorized into bagging, boosting, and stacking. Among them, boosting refers to the process of iteratively adding new weak models to create a strong model. Especially, gradient boosting is a powerful supervised learning algorithm that adds new weak models in the direction of reducing the errors of the previously created weak models [10]. After repeating this process, gradient boosting combines the weak models to produce a strong model with better performance. Compared to bagging and stacking, boosting is known to be more effective in reducing the bias of weak models. Because a GNN is difficult to have many layers due to the over-smoothing problem, the bias of a GNN can be high. For this reason, we choose to use boosting to reduce the bias of a GNN and thereby improve the performance of GNNs.

Until now, most studies on GNNs have focused on improving the performance of a single GNN. In contrast, improving the performance of GNNs using multiple GNNs has not been studied much yet. In this paper, we propose gradient boosted graph neural networks (GBGNN) that combine multiple shallow GNNs with gradient boosting. We use shallow GNNs as weak models and create new weak models using the proposed gradient boosting-based loss function. The proposed loss function is designed for the new model to be created in the direction of reducing the errors of the previously created weak models. We designed GBGNN as a flexible framework that can be integrated with any GNN models without additional constraints. Our empirical evaluations on three real-world datasets demonstrate that GBGNN performs much better than a single GNN. Our contributions are summarized as follows:

- We propose a novel method to enhance the performance of a single GNN by combining multiple GNNs using a gradient boosting mechanism.
- We derive a loss function used to construct subsequent GNNs based on gradient boosting and a method for combining those constructed GNNs.
- Our proposed framework can be integrated with any GNN models without additional constraints.

The structure of the paper is as follows. Section 2 presents related work that applies ensemble methods to GNNs. Section 3 provides a detailed explanation of our proposed method, GBGNN. Section 4 provides experimental results using three real-world datasets to validate the performance of GBGNN. Finally, Section 5 concludes the paper.

2. Related Work

As stated in Section 1, most studies on GNNs so far have focused on improving the performance of a single GNN. However, there are a few studies that apply ensemble methods to GNNs, which will be described below.

Before methods for applying ensemble methods to GNNs were proposed, Badirli et al. [11] proposed a method that uses shallow neural networks as weak models in the gradient boosting framework. Fig. 1 shows GrowNet proposed in [11]. At each boosting iteration, GrowNet combines the original input features with the features output from the penultimate layer of the previous neural network. GrowNet then creates a new neural network by training the neural network on the combined features. Finally, the final prediction is obtained as the weighted sum of outputs from all the neural networks. However, the authors of [11] applies gradient boosting to shallow neural networks and does not consider GNNs, which take graph data as input. Since graph data includes not only the features of each node but also the connection information between the nodes, it is difficult to apply GrowNet directly to GNNs.

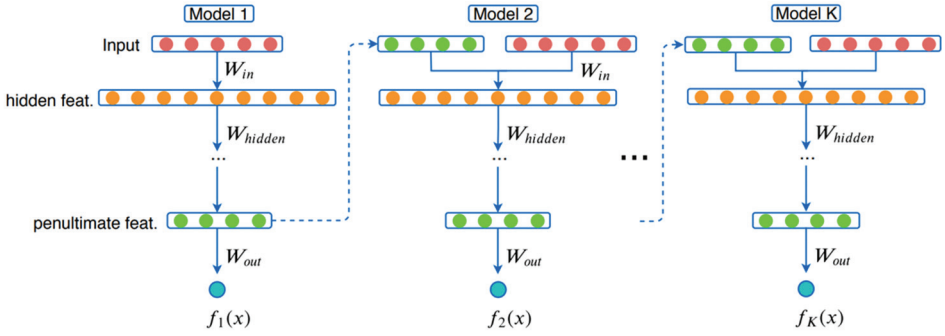

Fig. 1. The architecture of GrowNet [11].

Ivanov and Prokhorenkova [12] uses gradient boosted decision trees (GBDT) to improve the performance of a single GNN. GBDT is widely known to have excellent power for learning on tabular data like node features. Meanwhile, a GNN learns the representation of each node using the features and structural information of its neighboring nodes. For this reason, the author of [12] uses GBDT to enable a GNN to learn the representation of each node more effectively. Fig. 2 shows the architecture of boost-GNN (BGNN) proposed in [12]. In the first step, BGNN learns a GBDT model f, consisting of k decision trees, using the original node features X. Here, $f(X)$ represents the predictions made by f for X. In the second step, using $f(X)$, BGNN updates X to $\boldsymbol{X'}$, which is then passed to the GNN. In the third step, using $\boldsymbol{X'}$ and the graph G, which contains information about the connection between nodes, BGNN trains a GNN, denoted by g_θ, to minimize the loss function $L(g_\theta(G, \boldsymbol{X'}), Y)$, where both $\boldsymbol{X'}$ and the parameters θ of g_θ are optimized. Let $\boldsymbol{X'_{new}}$ represent the optimized node features. In the fourth step, BGNN uses the difference $\boldsymbol{X'_{new}} - \boldsymbol{X'}$ as the target values for the next decision trees, denoted as f^i, to be added to the GBDT. In this step, BGNN uses the equation $\boldsymbol{X'_{new}} = \boldsymbol{X'} - \eta\, \partial L(g_\theta(G, \boldsymbol{X'}), Y) / \partial \boldsymbol{X'}$ to build f^i, where η is a learning rate. In this way, BGNN adds new decision trees to GBDT that can improve the performance of the GNN. However, [12] is a study to improve the performance of a single GNN, rather than an ensemble method that combines multiple GNNs.

Sun et al. [13] propose a GNN model composed of multiple layers that are connected in a similar fashion to recurrent neural networks (RNNs). In [13], the multiple layers of a GNN are created using AdaBoost, one of the popular ensemble methods. Fig. 3 illustrates the architecture of AdaGCN proposed in [13]. In Fig. 3, each layer $f_\theta^{(i)}$ extracts information from neighbors with hop count $\leq i$ and AdaGCN combines them in an AdaBoost manner. By combining information from neighbors with different hop

counts with different weights, AdaGCN can soften the over-smoothing problem. However, [13] is a study to increase the number of layers in a single GNN and not about an ensemble method that uses multiple GNNs, which is our focus.

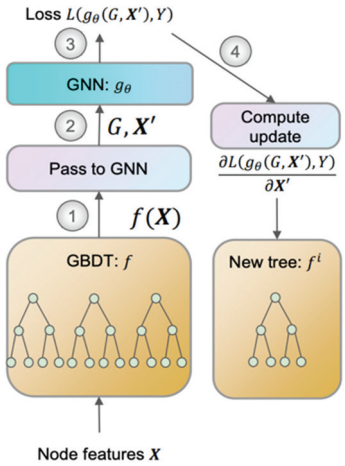

Fig. 2. The architecture of BGNN [12].

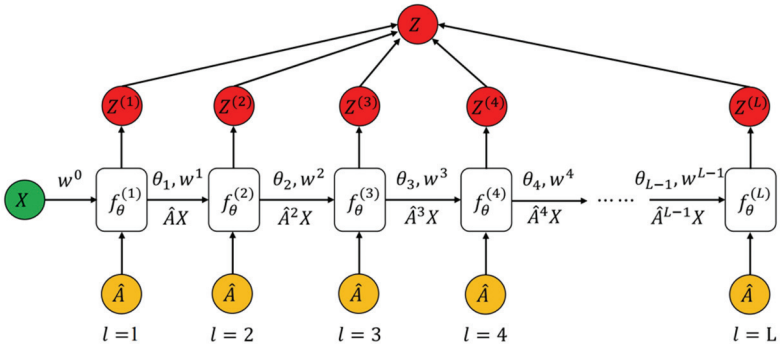

Fig. 3. The architecture of AdaGCN [13].

3. Proposed Method: GBGNN

3.1 Overview

In this paper, we propose GBGNN that uses multiple shallow GNNs as weak models and combines them into a strong model. Given a set of previous GNNs, GBGNN adds a new GNN in a gradient boosting way, which creates a new weak model that fits to the *negative* of the errors of the previous GNNs. Fig. 4 shows the overall process of constructing GBGNN. Constructing GBGNN consists of two main steps: 1) Initial model construction, where we construct the first weak model $g^{(0)}$, and 2) subsequent model construction, where we add the next weak models $g^{(1)}$, $g^{(2)}$, …, one by one to improve the performance of the combined model. In the following subsections, we elaborate each step.

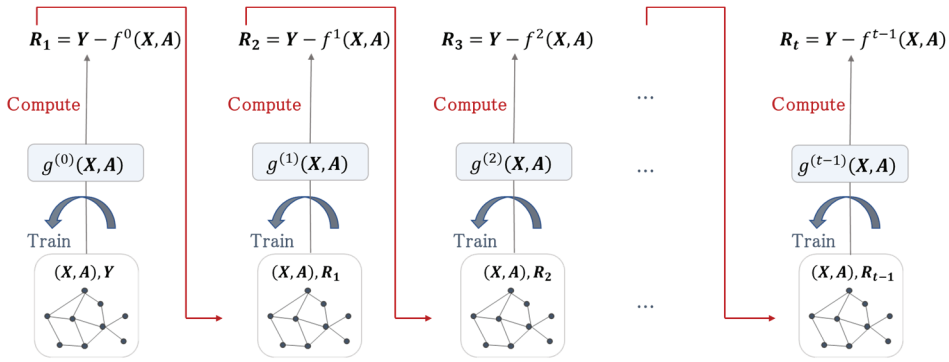

Fig. 4. The overall process of building GBGNN.

Before delving into the details of GBGNN, we first describe in more detail the working principle of gradient boosting, which is employed in GBGNN, with an illustrative example in Fig. 5. Given a dataset with features X and label Y, the initial model $f^0(X)$ is trained by minimizing the errors between Y and $f^0(X)$. Next, the negative gradient of the errors of $f^0(X)$ is computed, creating a new target Y^1 that corresponds to $Y - f^0(X)$. Subsequently, a new weak model $g^1(X)$ is trained to predict $Y^1 = Y - f^0(X)$, yielding an updated ensemble model $f^1(X) = f^0(X) + g^1(X)$. This process is repeated M times, where M denotes the number of weak models. Finally, the predictions of all weak models are combined to generate the final ensemble predictions, i.e., $f^M(X) = f^0(X) + g^1(X) + \ldots + g^M(X)$. Gradient boosting serves as a powerful ensemble method that enhances prediction performance, which progressively incorporates new weak models to compensate for errors made by previous models.

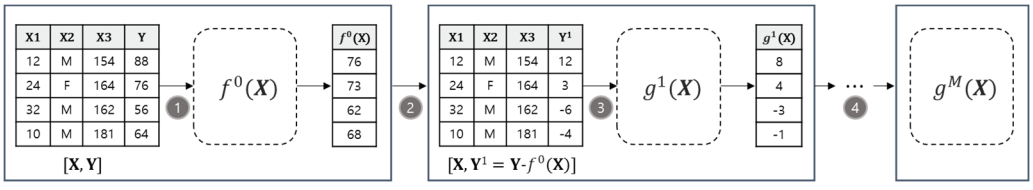

Fig. 5. Example illustrating the working principle of gradient boosting.

3.2 Step 1: Initial Model Construction

Let $G = (V, E)$ be an input graph, where $V = \{v_1, v_2, \ldots, v_N\}$ is a set of N nodes and $E = \{(v_i, v_j) | v_i, v_j \in V\}$ is a set of edges. Let $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ be the node feature matrix in which the ith row $\boldsymbol{X}_i$ represents the features of node v_i. Here, F is the number of features of each node. $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix in which each element $\boldsymbol{A}_{ij} = 1$ if $(v_i, v_j) \in E$ and $\boldsymbol{A}_{ij} = 0$ otherwise.

Let $g(\boldsymbol{X}, \boldsymbol{A})$ be a GNN whose inputs are a node feature matrix $\boldsymbol{X}$ and an adjacency matrix $\boldsymbol{A}$ representing a given graph G. Given $\boldsymbol{X}$ and $\boldsymbol{A}$, $g(\boldsymbol{X}, \boldsymbol{A})$ returns a prediction matrix $\hat{\boldsymbol{Y}} \in \mathbb{R}^{N \times C}$, where the ith row $\hat{\boldsymbol{Y}}_i$ represents the final prediction for node v_i. Here, C is the dimension of output vector $\hat{\boldsymbol{Y}}_i$. It is essential to note that there are various types of GNNs, such as graph convolutional network (GCN) [4], graph attention network (GAT) [5], and GraphSAGE [6], each of which implements a different strategy for learning the representation of each node. For example, $g(\boldsymbol{X}, \boldsymbol{A})$ for a GCN with two convolutional layers can be expressed as follows [14]:

$$\widehat{\boldsymbol{Y}} = g(\boldsymbol{X}, \boldsymbol{A}) = \sigma^{(1)}\left(\widehat{\boldsymbol{A}}\sigma^{(0)}\left(\widehat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(0)}\right)\boldsymbol{W}^{(1)}\right), \tag{1}$$

where $\widehat{\boldsymbol{A}}$ is the normalized adjacency matrix with self-loops, i.e., $\widehat{\boldsymbol{A}} = \widehat{\boldsymbol{D}}^{-1/2}(\boldsymbol{A} + \boldsymbol{I})\widehat{\boldsymbol{D}}^{-1/2}$, where $\widehat{\boldsymbol{D}}$ is the degree matrix of $\boldsymbol{A} + \boldsymbol{I}$. $\boldsymbol{W}^{(i)}$ and $\sigma^{(i)}$ represent the weight matrix and activation function (e.g., ReLU and softmax) of the $(I - 1)$th layer, respectively.

In the initial model construction step, given the target matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times C}$, we construct the first GNN, denoted as $g^{(0)}(\boldsymbol{X}, \boldsymbol{A})$, by minimizing a given loss function $L\left(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\right)$. We can define the loss function L differently for different tasks. For example, for a node classification task, we can use the cross-entropy loss function as follows:

$$L\left(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\right) = \sum_{i=1}^{N} L\left(\boldsymbol{Y}_i, \widehat{\boldsymbol{Y}}_i\right) = -\sum_{i=1}^{N}\sum_{j=1}^{C} \boldsymbol{Y}_{ij} \cdot \log\left(\widehat{\boldsymbol{Y}}_{ij}\right). \tag{2}$$

Here, C is the number of classes. $\boldsymbol{Y}_{ij} = 1$ if the ith node belongs to class j and $\boldsymbol{Y}_{ij} = 0$ otherwise. $\widehat{\boldsymbol{Y}}_{ij}$ represents the probability of the ith node belonging to class j predicted by the GNN. On the other hand, for a node regression task, we can use the mean squared error (MSE) loss function as follows:

$$L\left(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\right) = \sum_{i=1}^{N} L\left(\boldsymbol{Y}_i, \widehat{\boldsymbol{Y}}_i\right) = \frac{1}{2}\sum_{i=1}^{N} (\boldsymbol{Y}_i - \widehat{\boldsymbol{Y}}_i)^2. \tag{3}$$

Here, $\boldsymbol{Y}_i$ and $\widehat{\boldsymbol{Y}}_i$ represent the target value and the value predicted by the GNN for the ith node, respectively.

3.3 Step 2: Subsequent Model Construction

Once the initial GNN is constructed, we construct the next GNNs iteratively until we get satisfactory performance. Given a set of pre-constructed GNNs, we construct the next GNN with the goal of reducing the errors of the pre-constructed GNNs. For this goal, we construct the next GNN that fits the differences between the target values and the values predicted by the current ensemble model.

Now let us describe the subsequent model construction step more formally. Let $f^t(\boldsymbol{X}, \boldsymbol{A})$ be an ensemble model obtained at the iteration t. We define $f^0(\boldsymbol{X}, \boldsymbol{A}) = g^{(0)}(\boldsymbol{X}, \boldsymbol{A})$. At each iteration t, we produce $f^t(\boldsymbol{X}, \boldsymbol{A})$ by adding a new GNN $g^t(\boldsymbol{X}, \boldsymbol{A})$ to $f^{t-1}(\boldsymbol{X}, \boldsymbol{A})$ as follows:

$$f^t(\boldsymbol{X}, \boldsymbol{A}) = f^{t-1}(\boldsymbol{X}, \boldsymbol{A}) + \epsilon g^t(\boldsymbol{X}, \boldsymbol{A}), \tag{4}$$

where $f^{t-1}(\boldsymbol{X}, \boldsymbol{A})$ is the ensemble model constructed at the previous iteration, $g^t(\boldsymbol{X}, \boldsymbol{A})$ is an independent, shallow GNN and ϵ is a learning rate. At each iteration t, we create a new GNN $g^t(\boldsymbol{X}, \boldsymbol{A})$ that fits the differences between the target values $\boldsymbol{Y}$ and the values predicted by the current ensemble model $f^{t-1}(\boldsymbol{X}, \boldsymbol{A})$, which can be represented as follows:

$$g^t(\boldsymbol{X}, \boldsymbol{A}) = \underset{g(\boldsymbol{X}, \boldsymbol{A})}{\operatorname{argmin}} \sum_{i=1}^{N} ((\boldsymbol{Y}_i - f^{t-1}(\boldsymbol{X}, \boldsymbol{A})_i) - g(\boldsymbol{X}, \boldsymbol{A})_i)^2, \tag{5}$$

In other words, we train $g^t(\boldsymbol{X}, \boldsymbol{A})$ with the goal that $g^t(\boldsymbol{X}, \boldsymbol{A})_i = \boldsymbol{Y}_i - f^{t-1}(\boldsymbol{X}, \boldsymbol{A})_i$. Using the method described so far, we add new GNNs $g^1(\boldsymbol{X}, \boldsymbol{A})$, $g^2(\boldsymbol{X}, \boldsymbol{A})$, …, $g^t(\boldsymbol{X}, \boldsymbol{A})$ until we get satisfactory performance. Because $g^t(\boldsymbol{X}, \boldsymbol{A})$ serves to reduce the error of $f^{t-1}(\boldsymbol{X}, \boldsymbol{A})$, GBGNN can perform better than a single GNN by combining $f^{t-1}(\boldsymbol{X}, \boldsymbol{A})$ and $g^t(\boldsymbol{X}, \boldsymbol{A})$. Algorithm 1 describes the overall flow of GBGNN. In the following section, we provide experimental results that demonstrate the performance of GBGNN.

Algorithm 1. GBGNN construction

Input: Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, node feature matrix $\boldsymbol{X}$, adjacency matrix $\boldsymbol{A}$, target matrix $\boldsymbol{Y}$, loss function L, the number of iteration T, learning rate $\boldsymbol{\epsilon}$
Output: Final ensemble model $\boldsymbol{f^T(X, A)}$

// Step 1: Initial Model Construction
Train the initial GNN $\boldsymbol{g^{(0)}(X, A)}$ with the target matrix $\boldsymbol{Y}$ and loss function L
$\boldsymbol{f^0(X, A) = g^{(0)}(X, A)}$

// Step 2: Subsequent Model Construction
for t = 1 to T **do**
  Train a GNN $\boldsymbol{g^{(t)}(X, A)}$ that fits $\boldsymbol{Y - f^{t-1}(X, A)}$
  $\boldsymbol{f^t(X, A) = f^{t-1}(X, A) + \epsilon g^t(X, A)}$
end for

return $\boldsymbol{f^T(X, A)}$

4. Experiments

4.1 Experimental Setup

In this section, we investigate how much the performance of a GNN is improved by GBGNN that ensembles multiple shallow GNNs using gradient boosting. More specifically, we compare the performance of GBGNN with that of a single GNN as we progressively increase the number of iterations in GBGNN. In the experiments, we use GCN and GAT as a weak model in GBGNN to investigate how much GBGNN improves their performance. However, as mentioned in Section 1, GBGNN is designed as a flexible framework that can be integrated with any GNN models, such as GraphSAGE, without additional constraints.

To evaluate the performance of GBGNN, we use three real-world citation network datasets, i.e., Cora [15], Citeseer [16], and PubMed [17]. The following provides detailed descriptions of the three datasets:

- Cora corresponds to a graph consisting of 2,708 nodes and 5,429 edges. Each node represents a scientific publication and is classified into one of 7 classes (i.e., neural networks, reinforcement learning, genetic algorithm, theory, rule learning, probabilistic method, and case-based). Each node has 1,433 features, where each feature represents the absence (0) or presence (1) of the corresponding word in the dictionary.
- Citeseer corresponds to a graph consisting of 3,312 nodes and 4,732 edges. Each node represents a scientific publication and is classified into one of 6 classes (i.e., Agent, AI, DB, HCI, IR, and MR).

Each node has 3,703 features, where each feature represents the absence (0) or presence (1) of the corresponding word in the dictionary.

- PubMed corresponds to a graph consisting of 19,717 nodes and 44,338 edges. Each node represents a scientific publication and is classified into one of 3 classes. Each node has 500 features, where each feature presents the tf-idf value of the corresponding word in the dictionary.

Table 1 provides a detailed summary of the statistics of the datasets.

Table 1. Statistics of the datasets used in the experiments

Dataset	Statistics			
	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,312	4,732	3,703	6
PubMed	19,717	44,338	500	3

When applying GNNs to each dataset, we performed a preprocessing step to handle the input features. Since the number of words in each scientific publication varies, the dimensions of the input feature vectors are different. To address this issue, we applied row normalization to ensure consistent dimensions of the input feature vectors, which facilitates a better understanding of the relationships between nodes.

For each dataset, we split the dataset into a training set, a validation set, and a test set. We used 70%, 20%, and 10% of the dataset as a training set, a validation set, and a test set, respectively. Note that the only hyperparameter used in GBGNN is the learning rate $\boldsymbol{\epsilon}$ in Eq. (4). We used the validation set to determine the optimal value of $\boldsymbol{\epsilon}$ and set $\boldsymbol{\epsilon = 0.1}$ for all the datasets. We implemented all the models used in the experiments using PyTorch. Table 2 presents the detailed specifications of the hardware and software used in the experiments.

Table 2. Specifications of hardware and software used in the experiments

	Specification
Hardware	
Processor	Intel Core i7-11700
Memory	64 GB
Graphics Card	NVIDIA GeForce RTX 3080 Ti
Software	Microsoft Window 10
	Python 3.7
	PyTorch 1.8.0
	CUDA 11.1
	Numpy 1.19.5

When applying GBGNN to GCN, we use a GCN with two layers with 64 hidden units as a weak model. Similarly, when applying GBGNN to GAT, we use a GAT with two layers with 64 hidden units as a weak model. For GAT, we set the number of attention heads and alpha value to 8 and 0.2, respectively. For GBGNN, we set the number of iterations, which corresponds to the number of weak models, to 50. The GBGNN model is optimized using the Adam optimizer with a learning rate of 0.01. The epochs, dropout rate, and L2 regularization weight are set to 100, 0.1 and 5×10^{-4}, respectively.

In the experiments, we investigate the performance improvement of GBGNN over a single GCN and

a single GAT, respectively. As the performance measure, we evaluate the node classification accuracy of the models on the test set. In the following subsection, we provide experimental results that show the performance improvement of GBGNN over a single GCN and a single GAT, respectively.

4.2 Performance Evaluation Results

Table 3 shows the performance evaluation results when GBGNN is applied to GCN and GAT, respectively, for the three datasets.

Table 3. Performance evaluation results of GBGNN on the three real datasets

Model	Cora		Citeseer		PubMed	
	Accuracy (%)	**Training time (s)**	**Accuracy (%)**	**Training time (s)**	**Accuracy (%)**	**Training time (s)**
A single GCN	68.5	0.19	64.7	0.24	85.2	0.32
GBGNN (applied to GCN)	**80.8**	8	**78.9**	12	**86.6**	16
A single GAT	78.3	2.2	74.5	3.4	86.0	13.3
GBGNN (applied to GAT)	**84.4**	92	**78.5**	112	**87.2**	607

The bold font indicates the best performance in each test.

The evaluation results in Table 3 demonstrate the significant performance improvement of GBGNN over both a single GCN and a single GAT. Specifically, when applying GBGNN to GCN, the performance improvement of GBGNN over a single GCN was up to 14.1%p. Here, %p represents the arithmetic difference between two percentages. On the Cora dataset, GBGNN achieved an impressive accuracy improvement of approximately 12.3%p. Furthermore, on the Citeseer dataset, GBGNN showed an even greater improvement, with an increase of 14.1%p in accuracy. Although the performance improvement on the PubMed dataset was relatively smaller, i.e., 1.4%p, it still indicates a positive impact of GBGNN on the performance of GCN across different datasets. Similarly, when applying GBGNN to GAT, we found significant performance improvements as well. On the Cora, Citeseer, and PubMed datasets, GBGNN achieved performance improvements of approximately 6.1%p, 4.0%p, and 1.2%p, respectively. Note that a single GAT generally shows better performance than a single GCN on all the three datasets. This can be attributed to the attention mechanism employed by GAT, which enables it to capture more fine-grained relationships among the nodes in the graph. However, when GBGNN was applied to GCN, GBGNN exhibited even better performance than a single GAT, highlighting the performance improvement effect of GBGNN. Overall, these experimental results validate that GBGNN effectively improves the performance of a single GNN, regardless of the type of GNN. Note also that GBGNN requires a longer training time compared to a single GNN, as it involves training multiple GNNs iteratively. However, considering the significant performance improvement, we believe that GBGNN can be effectively used in various real-world environments that require improved performance of GNNs.

Fig. 6 shows how the performance of GBGNN on the Cora dataset changes as the number of iterations increases from 1 to 50. In this experiment, we varied the value of the hyperparameter ϵ used in GBGNN to 0.01, 0.05, and 0.1. It is important to highlight that the performance of GBGNN at iteration 0 corresponds to that of a single GCN in Fig. 6(a) and a single GAT in Fig. 6(b), respectively. As shown in Fig. 6, the performance of GBGNN progressively improves as the number of iterations (i.e., the number of weak models) increases for both GCN and GAT. This is because the weak models added by GBGNN

actually have the effect of reducing the errors of the previously created ensemble model. Thus, Fig. 6 confirms that the performance of GNN can be improved by GBGNN, which combines multiple shallow GNNs effectively. We can also observe that the convergence speed of GBGNN depends on the type of GNN and the value of ϵ. In Fig. 6(a), when $\epsilon = 0.1$, the performance improvement of GBGNN reaches 12%p when the number of iterations is about 20. In comparison, when $\epsilon = 0.05$, the performance improvement of GBGNN converges to 12%p when the number of iterations is about 40. However, when $\epsilon = 0.01$, GBGNN shows significantly slower convergence. This suggests that the convergence speed of GBGNN becomes slower as the value of ϵ gets smaller. Note that the same pattern is observed in Fig. 6(b). However, in Fig. 6(b), the performance of GBGNN does not converge even when the number of iterations reaches 50. This means that the number of weak models required for the best performance is different depending on the type of GNN.

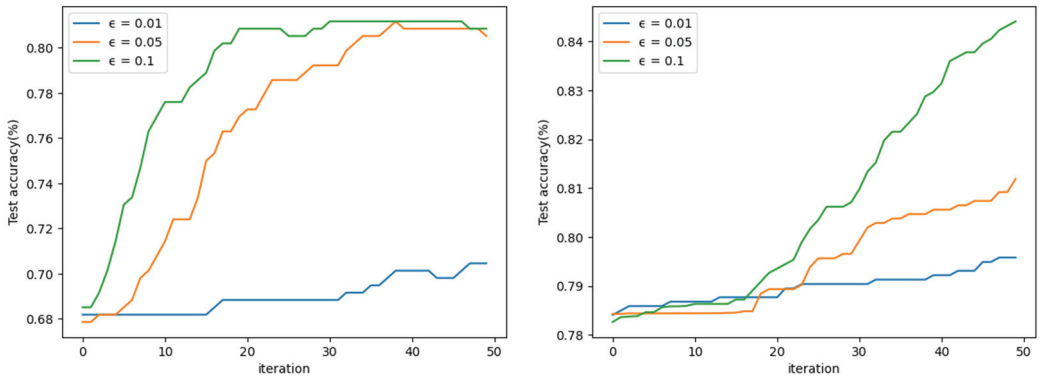

Fig. 6. The performance of GBGNN on the Cora dataset: (a) GBGNN applied to GCN and (b) GBGNN applied to GAT.

Fig. 7 depicts how GBGNN performs on the Citeseer dataset as the number of iterations increases from 1 to 50. Similarly, in this case, the performance of GBGNN steadily improves with an increasing number of iterations. Furthermore, as in the previous experiment, the convergence speed of GBGNN increases as the value of ϵ increases in both Fig. 7(a) and 7(b). Note that there is a significant performance improvement in the early iterations in Fig. 7(a), when $\epsilon = 0.1$ and 0.05. This indicates that the subsequent GCN models constructed by GBGNN effectively rectify the errors made by the previous models. However, when $\epsilon = 0.01$, the performance improvement becomes very slow because the small value of ϵ limits the extent of error reduction. Nonetheless, in both Fig. 7(a) and 7(b), the performance of GBGNN shows continuous improvement with an increasing number of iterations. In Fig. 7(a), GBGNN shows a performance improvement of about 14.1%p when $\epsilon = 0.1$ and 0.05 compared to a single GCN. Meanwhile, in Fig. 7(b), GBGNN exhibits a performance improvement of about 4%p when $\epsilon = 0.1$ compared to a single GAT. However, since the performance of GBGNN has not yet converged in Fig. 7(b), it is expected that the performance of GBGNN will further improve as the number of iterations increases.

Lastly, Fig. 8 illustrates how the performance of GBGNN on the PubMed dataset changes as the number of iterations increases from 1 to 50. Similar to the other datasets, GBGNN shows continuous improvement in node classification accuracy with increasing iterations. However, unlike the other

datasets where the performance of GBGNN applied to GCN converges quickly, the performance of GBGNN on the PubMed dataset shows a gradual increase with increasing iterations in both Fig. 8(a) and 8(b), regardless of the value of ϵ. This indicates that, depending on the characteristics of the dataset, the subsequent models may not significantly reduce the errors of the previous models in gradient boosting. Nonetheless, when compared to a single GCN and a single GAT, GBGNN achieves a performance improvement of about 1.4%p and 1.2%p, respectively. However, since the performance of GBGNN has not yet converged in both Fig. 8(a) and 8(b), it is also expected that the performance of GBGNN will further improve as the number of iterations increases.

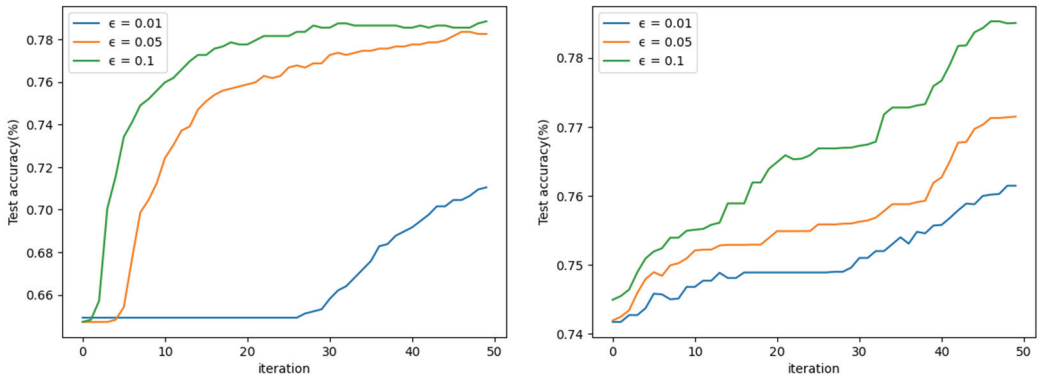

Fig. 7. The performance of GBGNN on the Citeseer dataset: (a) GBGNN applied to GCN and (b) GBGNN applied to GAT.

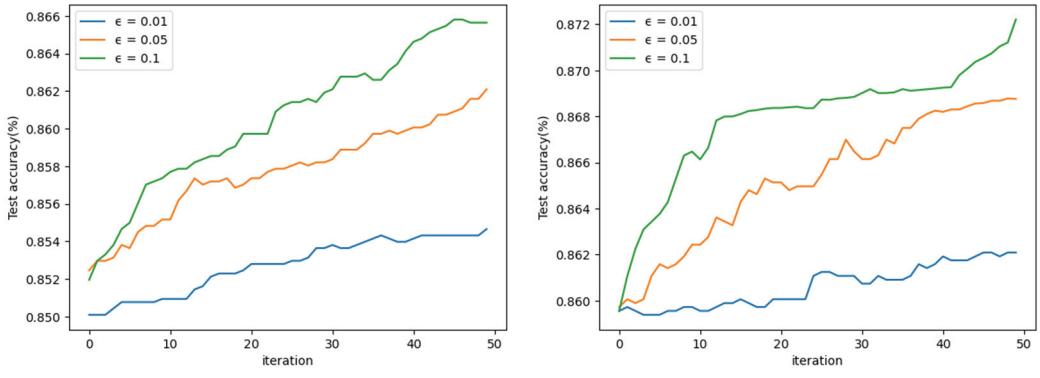

Fig. 8. The performance of GBGNN on the PubMed dataset: (a) GBGNN applied to GCN and (b) GBGNN applied to GAT.

Overall, through the various experiments on the three real datasets, we observed that GBGNN consistently enhances the performance of GNNs, irrespective of the GNN type or the hyperparameter ϵ. Therefore, we can confirm that GBGNN can improve the performance of GNNs significantly without causing the over-smoothing problem.

5. Conclusion

It is well known that it is difficult to improve the performance of a GNN by stacking layers deeply due to the over-smoothing problem of GNNs. Therefore, in this paper, we propose GBGNN, which is a gradient boosting-based ensemble method that combines multiple shallow GNNs to produce a strong model. GBGNN first constructs an initial GNN and constructs the next GNNs sequentially with the goal of eliminating the errors of the previously constructed GNNs. In order to construct the next GNN that can reduce the errors of the previous GNNs, GBGNN uses a gradient boosting-based loss function that approximates the direction in which the error can be reduced. The experimental results on three real-world datasets demonstrate that GBGNN enhances the performance of GNNs by adding shallow GNNs that are effective in reducing the errors of the current ensemble model. Thus, we can confirm that GBGNN can be used to improve the performance of GNNs without causing the over-smoothing problem.

Conflict of Interest

The authors declare that they have no competing interests.

Funding

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1A2C1012543).

Acknowledgement

This paper is the extended version of “A gradient boosting method for graph neural networks,” in the Annual Conference of KIPS (ACK 2022) held in Seoul, Republic of Korea dated November 3-5, 2022.

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4-24, 2021. https://doi.org/10.1109/TNNLS.2020.2978386

[2] Y. Ding, Z. Zhang, X. Zhao, W. Cai, F. He, Y. Cai, and W. W. Cai, “Deep hybrid: multi-graph neural network collaboration for hyperspectral image classification,” *Defence Technology*, vol. 23, pp. 164-176, 2023. https://doi.org/10.1016/j.dt.2022.02.007

[3] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, “Graph neural networks in recommender systems: a survey,” *ACM Computing Surveys*, vol. 55, no. 5, pp. 1-37, 2022. https://doi.org/10.1145/3535101

[4] M. Gori, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in *Proceedings of 2005 IEEE International Joint Conference on Neural Networks*, Montreal, Canada, 2005, pp. 729-734. https://doi.org/10.1109/IJCNN.2005.1555942

[5] W. L. Hamilton, R. Ying, and J. Leskovec, “Representation learning on graphs: methods and applications,” 2017 [Online]. Available: https://arxiv.org/abs/1709.05584.
[6] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016 [Online]. Available: https://arxiv.org/abs/1609.02907.
[7] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” 2018 [Online]. Available: https://arxiv.org/abs/1710.10903.
[8] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in Neural Information Processing Systems*, vol. 30, pp. 1024-1034, 2017.
[9] Y. Yang, T. Liu, Y. Wang, J. Zhou, Q. Gan, Z. Wei, Z. Zhang, Z. Huang, and D. Wipf, “Graph neural networks inspired by classical iterative algorithms,” *Proceedings of Machine Learning Research*, vol. 139, pp. 11773-11783, 2021.
[10] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001.
[11] S. Badirli, X. Liu, Z. Xing, A. Bhowmik, K. Doan, and S. S. Keerthi, “Gradient boosting neural networks: GrowNet,” 2020 [Online]. Available: https://arxiv.org/abs/2002.07971.
[12] S. Ivanov and L. Prokhorenkova, “Boost then convolve: gradient boosting meets graph neural networks,” 2021 [Online]. Available: https://arxiv.org/abs/2101.08543.
[13] K. Sun, Z. Zhu, and Z. Lin, “AdaGCN: Adaboosting graph convolutional networks into deep models,” 2019 [Online]. Available: https://arxiv.org/abs/1908.05081.
[14] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” *Advances in Neural Information Processing Systems*, vol. 29, pp. 3837-3845, 2016.
[15] Cora dataset [Online]. Available: https://paperswithcode.com/dataset/cora.
[16] Citeseer dataset [Online]. https://paperswithcode.com/dataset/citeseer.
[17] PubMed dataset [Online]. Avaiable: https://paperswithcode.com/dataset/pubmed.

Eunjo Jang **https://orcid.org/0000-0003-2849-0331**

She received B.S. degrees in Statistics and Computer Science from Sookmyung Women’s University, Seoul, Republic of Korea, in 2022. She is currently an M.S. student in the Department of Computer Science at Sookmyung Women’s University, Seoul, Republic of Korea. Her current research interests include data mining, graph neural network, and recommendation systems.

Ki Yong Lee **https://orcid.org/0000-0003-2318-671X**

He received his B.S., M.S., and Ph.D. degrees in Computer Science from KAIST, Daejeon, Republic of Korea, in 1998, 2000, and 2006, respectively. From 2006 to 2008, he worked for Samsung Electronics, Suwon, Korea as a senior engineer. From 2008 to 2010, he was a research assistant professor of the Department of Computer Science at KAIST, Daejeon, Korea. He joined the faculty of the Division of Computer Science at Sookmyung Women’s University, Seoul, in 2010, where currently he is a professor. His research interests include database systems, data mining, and big data.