Bi-GCN: Binary Graph Convolutional Network

Junfu Wang[1,2], Yunhong Wang[2], Zhen Yang [1,2], Liang Yang [3], Yuanfang Guo [1,2]*
[1] State Key Laboratory of Software Development Environment, Beihang University, China
[2] School of Computer Science and Engineering, Beihang University, China
[3] School of Artificial Intelligence, Hebei University of Technology, China
{wangjunfu,yhwang,yangzhen7,andyguo}@buaa.edu.cn,yangliang@vip.qq.com

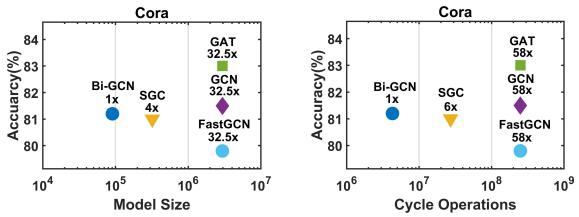
Abstract

Graph Neural Networks (GNNs) have achieved tremendous success in graph representation learning. Unfortunately, current GNNs usually rely on loading the entire attributed graph into network for processing. This implicit assumption may not be satisfied with limited memory resources, especially when the attributed graph is large. In this paper, we pioneer to propose a Binary Graph Convolutional Network (Bi-GCN), which binarizes both the network parameters and input node features. Besides, the original matrix multiplications are revised to binary operations for accelerations. According to the theoretical analysis, our Bi-GCN can reduce the memory consumption by an average of ∼30x for both the network parameters and input data, and accelerate the inference speed by an average of ∼47x, on the citation networks. Meanwhile, we also design a new gradient approximation based back-propagation method to train our Bi-GCN well. Extensive experiments have demonstrated that our Bi-GCN can give a comparable performance compared to the full-precision baselines. Besides, our binarization approach can be easily applied to other GNNs, which has been verified in the experiments.

1. Introduction

In the past few years, Graph Neural Networks (GNNs), which can learn effective representations from irregular data, have given excellent performances in various graph-based tasks [18, 28, 31, 30]. Considering the superior representation abilities of these newly developed GNNs, researchers have also applied them to many tasks, including natural language processing [33], computer vision [24], etc.

Unfortunately, the current success of GNNs is attributed to an implicit assumption that the input of GNNs contains the entire attributed graph. If the entire graph is too large to be fed into GNNs due to limited memory resources, in both the training and inference process, which is highly likely when the scale of the graph increases, the performances of GNNs may degrade drastically.



(a) (b)

Figure 1. Performances on the Cora dataset. Note that the model size is measured in bits and the number of cycle operations, which will be introduced in Sec. 5, is employed to reflect the inference speed. Bi-GCN gives the fastest inference speed and the lowest memory consumption with comparable accuracy.

To tackle this problem, an intuitive solution is sampling, e.g., sampling a subgraph with a suitable size to be entirely loaded into GNNs. The sampling based methods can be classified into two categories, neighbor sampling [12, 5] and graph sampling [4, 6, 34]. Neighbor sampling selects a fixed number of neighbors for each node in the next layer to ensure that every node can be sampled. Thus, it can be utilized in both the training and inference process. Unfortunately, when the number of layers increases, the problem of *neighbor explosion* [34] arises, such that both the training and inference time will increase exponentially. Different from neighbor sampling, graph sampling samples a set of subgraphs in the training process, which can avoid the problem of *neighbor explosion*. However, it cannot guarantee that every node can be at least sampled once in the whole training/inference process. Thus it is only feasible for the training process, because the testing process usually requires GNNs to process each node in the graph.

Another feasible solution is compressing the size of the input graph data and the GNN model to better utilize the limited memory and computational resources. Several approaches have been proposed to compress the convolutional

*Corresponding author.

neural networks (CNNs), such as designing shallow networks [2], pruning [9], designing compact layers [27] and quantizing the parameters [15]. In quantization-based methods, binarization [15, 22, 21] has achieved a great success in many CNN-based practical vision tasks when a faster speed and a lower memory consumption is desired.

However, compared to the CNN compression methods, the compression of GNNs possesses unique challenges. Firstly, since the input graph data is usually much larger than the GNN models, the compression of the loaded data demands more attention. Secondly, the GNNs are usually shallow, e.g., the standard GCN [18] only has 2 layers, which contain less redundancies, thus the compression will be more difficult to be achieved. At last, the nodes tend to be similar to its neighbors in the high-level semantic space, while they tend to be different in the low-level feature space, which is different from the grid-like data, such as images, videos, etc. This characteristic requires the compressed GNNs to possess sufficient parameters for representations. In general, the tradeoff between the compression ratio and accuracy in the compressed GNNs requires careful designs.

To tackle the memory and complexity issues, SGC [29], which is a 1-layered GNN, compresses GCN [18] by removing nonlinearities and collapsing weight matrices between consecutive layers. This shallow GNN can accelerate both the training and inference processes with comparable performance. Although SGC compresses the network parameters, it does not compress the loaded data, which is the major memory consumption when processing the graphs with GNNs.

In this paper, to alleviate the memory and complexity issue, we pioneer to propose a binarized GCN, named Binary Graph Convolutional Network (Bi-GCN), which is a simple yet efficient approximation of GCN [18], by binarizing the parameters and node attribute representations. Specifically, the binarization of the weights is performed by splitting them into multiple feature selectors and maintaining a scalar per selector to further reduce the quantization errors. Similarly, the binarization of the node features can be carried out by splitting the node features and assigning an attention weight to each node. By employing those additional scalars, more efficient information can be learned and retained efficiently. After binarizing the weights and node features, the computational complexity and the memory consumptions induced by the network parameters and input data can be largely reduced. Since the existing binary back propagation method [22] has not considered the relationships among the binary weights, we also design a new back propagation method by tackling this issue. An intuitive comparison between our Bi-GCN and the baseline methods is shown in Figure 1, which demonstrates that our Bi-GCN can achieve the fastest inference speed and lowest memory consumption with a comparable accuracy compared to the standard full-precision GNNs.

Our proposed Bi-GCN can reduce the redundancies in the node representations while maintain the principle information. When the number of layers increases, Bi-GCN also gives a more obvious reductions of the memory consumptions of the parameters and effectively alleviates the overfitting problem. Besides, our binarization approach can be easily applied to other GNNs.

The contributions are summarized as follows:

- We pioneer to propose a binarized GCN, named Binary Graph Convolutional Network (Bi-GCN), which can significantly reduce the memory consumptions by $\sim$30x for both the network parameters and input node attributes, and accelerate the inference by an average of $\sim$47x, on the citation networks, theoretically.
- We design a new back propagation method to effectively train our Bi-GCN, by considering the relationships among the binary weights in the back propagation process.
- With respect to the significant memory reductions and accelerations, our Bi-GCN can also give a comparable performance compared to the standard GCN on four benchmark datasets.

2. Related Work

2.1. Sampling Based GNNs

Sampling is an effective method that allows GNNs to process larger graphs with limited memory. Current sampling methods can be categorized into two categories, neighbor sampling [12, 5] and graph sampling [4, 6, 34]. GraphSAGE [12] gives an empirical number of the sampled neighbors and extends the GNNs to inductive learning. VRGCN [5] reduces the sampling size by maintaining the embedding of each node from the previous iteration, which requires a doubled memory consumption. Meanwhile, FastGCN [4] samples a subgraph in each layer to accelerate the training process, which sacrifices the classification accuracy. ClusterGCN [6] groups the nodes by graph clusting methods, which demands additional complexity for the clustering. GraphSAINT [34] proposes an edge sampling method with low variance and apply GCN [18] to the sampled subgraphs. Besides, DropEdge [23] generates the subgraphs randomly and DropConnection [13] adaptively samples the subgraphs, to alleviate the overfitting problem.

2.2. CNN Binarization Methods

Convolutional Neural Networks (CNNs) suffer from certain issues, such as high computational costs and etc. Binarization, as a promising type of techniques in network compression, has been widely utilized to reduce the memory and

computation costs for CNNs. BinaryConnect [7] binarizes the network parameters and replaces most of the floating-point multiplications with floating-point additions. Binarynet [8] further binarizes the activation function and uses the XNOR (not-exclusive-OR) operations to accelerate the inference process. XNOR-Net [22] proposes a scalar based binarization approach and successfully applies it to the popular CNNs, such as ResNet [14] and GoogLeNet [26].

3. Preliminaries

3.1. Notations

Here, we define the notations utilized throughout this paper. We denote an undirected attributed graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, X\}$ with the vertex set $\mathcal{V} = \{v_i\}_{i=1}^{N}$ and edge set $\mathcal{E} = \{e_i\}_{i=1}^{E}$. Each node v_i contains a feature $X_i \in \mathbb{R}^d$. $X \in \mathbb{R}^{N\times d}$ is the collection of all the features in all the nodes. $A = [a_{ij}] \in \mathbb{R}^{N\times N}$ is the adjacency matrix which reveals the relationships between each pair of vertices, i.e., the topology information of $\mathcal{G}$. $d_i = \sum_j a_{ij}$ stands for the degree of node v_i and $D = diag(d_1, d_2, \ldots, d_n)$ represents the degree matrix corresponding to the adjacency matrix A. Then, $\hat{A} = A + I$ is the adjacency matrix of the original topology with self-loops and $\hat{D}$ is its corresponding degree matrix with $\hat{D}_{ii} = \sum_j \hat{a}_{ij}$. Note that we employ the superscript "(l)" to represent the l-th layer, e.g., $H^{(l)}$ is the input node features to the l-th layer.

3.2. Graph Convolutional Network

Graph Convolutional Network (GCN) [18] has become the most popular graph neural network in the past few years. Since our binarization approach takes GCN as the basis GNN, we give a brief review of GCN here.

Given an undirected graph $\mathcal{G}$, the graph convolution operation can be described as

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W^{(l)}), \tag{1}$$

where $\tilde{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is a sparse matrix, and $W^{(l)} \in \mathbb{R}^{d_{in}^{(l)}\times d_{out}^{(l)}}$ contains the learnable parameters. Note that $H^{(l+1)}$ is the output of the l-th layer and the input of the $(l+1)$-th layer, and $H^{(0)} = X$. σ is the non-linear activation function, e.g., ReLU.

From the perspective of spatial methods, the graph convolution layer in GCN can be decomposed into two steps, where $\tilde{A}H^{(l)}$ is the aggregation step and $H^{(l)}W^{(l)}$ is the feature extraction step. The aggregation step tends to constrain the node attributes in the local neighborhood to be similar. After that, the feature extraction step can easily extract the commonalities between the neighboring nodes.

GCN typically utilizes a task-dependent loss function, e.g., the cross-entropy loss for the node classification tasks, which is defined as

$$\mathcal{L} = -\sum_{v_i\in\mathcal{V}^{label}}\sum_{c=1}^{C} Y_{i,c}log(\tilde{Y}_{i,c}), \tag{2}$$

where $\mathcal{V}^{label}$ stands for the set of the labelled nodes, C denotes the number of classes, Y represents the ground truth labels, and $\tilde{Y} = softmax(H^{(L)})$ are the predictions of the L-layered GCN.

4. Binary Graph Convolutional Network

In this section, we propose our Binary Graph Convolution Network (Bi-GCN), a binarized version of the standard GCN. As mentioned in previous section, a graph convolution layer can be decomposed into two steps, aggregation and feature extraction. In Bi-GCN, we only focus on binarizing the feature extraction step, because the aggregation step possesses no learnable parameters (which yields negligible memory consumption) and it only requires a few calculations (which can be neglected compared to the feature extraction step). Therefore, the aggregation step of the original GCN is maintained. For the feature extraction step, we binarize both the network parameters and node features to reduce the memory consumptions. To reduce the computational complexities and accelerate the inference process, the XNOR (not-exclusive-OR) and bit count operations are utilized, instead of the traditional floating-point multiplications. Finally, we design an effective back-propagation algorithm for training our binarized graph convolution layer.

4.1. Binarization of the Feature Extraction Step

Based on the vector binarization algorithm [22], we can perform the binarization to the feature extraction step $Z^{(l)} = H^{(l)}W^{(l)}$ in the graph convolution shown in Eq. 1. Note that for this feature extraction (matrix multiplication), we adopt the bucketing [1] method to generalize the binary inner product operation to the binary matrix multiplication operation. Specifically, we split the matrix into multiple buckets of consecutive values with a fixed size and perform the scaling operation separately.

4.1.1. Binarization of the Parameters

Since each column of the parameter matrix of the l-th layer $W^{(l)}$ serves as a feature selector in the computation of $Z^{(l)}$, each column of $W^{(l)}$ is splitted as a bucket, i.e., a vector. Let $\alpha^{(l)} = (\alpha_1^{(l)}, \alpha_2^{(l)}, ..., \alpha_{d_{out}^{(l)}}^{(l)})$, which are the scalars for each bucket. Let $B^{(l)} = (B_1^{(l)}, B_2^{(l)}, ..., B_{d_{out}^{(l)}}^{(l)}) \in \{-1, 1\}^{d_{in}^{(l)}\times d_{out}^{(l)}}$ be the binarized buckets of $W^{(l)}$. Then, based on the vector binarization algorithm, the optimal $B^{(l)}$ and $\alpha^{(l)}$ can be easily calculated by

$$B_j^{(l)} = sign(W_{:,j}^{(l)}), \tag{3}$$

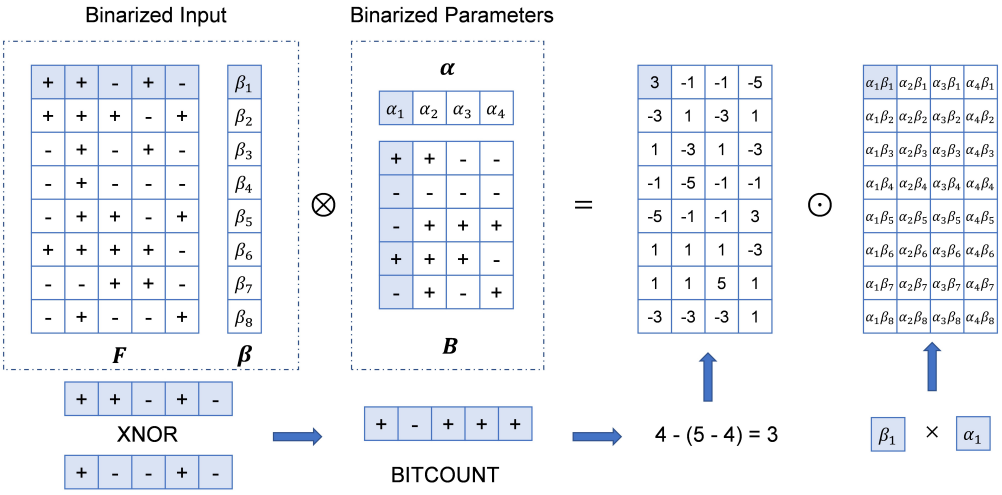


Figure 2. An example of binary feature extraction step. Both the input features and parameters will be binarized to binary matrices. $\otimes$ denotes the binary matrix multiplication defined in Sec. 4 and $\odot$ represents the element-wise multiplication.

$$\alpha_j^{(l)} = \frac{1}{d_{out}^{(l)}} ||W_{:,j}^{(l)}||_1, \tag{4}$$

where $W_{:,j}^{(l)}$ represents the j-th column of $W^{(l)}$. It can be approximated via

$$W_{:,j}^{(l)} \approx \tilde{W}_{:,j}^{(l)} = \alpha_j^{(l)} B_j^{(l)}. \tag{5}$$

Based on Eq. 5, the graph convolution operation with binarized weights can then be described as

$$H^{(l+1)} \approx H_p^{(l+1)} = \sigma(\tilde{A} H^{(l)} \tilde{W}^{(l)}), \tag{6}$$

where $H_p^{(l+1)}$ is the binary approximation of $H^{(l+1)}$ with the binarized parameters $\tilde{W}^{(l)}$. The binarization of the parameters can reduce the memory consumption by a factor of ∼30x, compared to the parameters with full precision, which will be proven in Sec. 5.

4.1.2. Binarization of the Node Features

Due to the over-smoothing issue [20] induced by the current graph convolution operation, current GNNs are usually shallow, e.g., the vanilla GCN only contains 2 graph convolution layers. Although the future GNNs may possess a larger model, the data sizes of commonly employed attributed graphs are usually much larger than the current model size. To reduce the memory consumption of the input data, which is mostly induced by the node features, we also perform binarization to the node features which will be processed by the graph convolutional layers.

To binarize the node features, we split $H^{(l)}$ into row buckets based on the constraints of the matrix multiplication to compute $Z^{(l)}$, i.e., each row of $H^{(l)}$ will conduct an inner product with each column of $W^{(l)}$. Let $\beta^{(l)} = (\beta_1^{(l)}, \beta_2^{(l)}, ..., \beta_N^{(l)})$ denote the scalars for each bucket in $H^{(l)}$. Let $F^{(l)} = (F_1^{(l)}; F_2^{(l)}; ...; F_N^{(l)}) \in \{-1, 1\}^{N \times d_{in}^{(l)}}$ be the binarized buckets. Then, with the vector binarization algorithm, the optimal β and F can be computed by

$$\beta_i^{(l)} = \frac{1}{N} ||H_{i,:}^{(l)}||_1, \tag{7}$$

$$F_i^{(l)} = sign(H_{i,:}^{(l)}), \tag{8}$$

where $H_{i,:}^{(l)}$ represents the i-th row of $H^{(l)}$. Then, the binary approximation of $H^{(l)}$ can be obtained via

$$H_{i,:}^{(l)} \approx \tilde{H}_{i,:}^{(l)} = \beta_i^{(l)} F_i^{(l)}. \tag{9}$$

Intuitively, β can be considered as the node-weights for the feature representations. At last, the graph convolution operation with binarized weights and node features can be formulated as

$$H^{(l+1)} \approx H_{ip}^{(l+1)} = \tilde{A} \tilde{H}^{(l)} \tilde{W}^{(l)}. \tag{10}$$

Note that this binarization of the node features, i.e., the input of the graph convolutional layer, also possesses the ability of activation, thus we do not employ specific activation functions (such as ReLU). Similar to the binarization of the weights, the memory consumption of the loaded attributed graph data can be reduced by a factor of ∼30x compared to the vanilla GCN.

4.1.3. Binary Operations

With the binarized graph convolutional layers, we can accelerate the calculations by employing the XNOR and bit-count operations instead of the floating-point additions and multiplications. Let $\zeta^{(l)}$ represent the approximation of $Z^{(l)}$. Then,

$$Z_{ij}^{(l)} \approx \zeta_{ij}^{(l)} = \beta_i^{(l)} \alpha_j^{(l)} F_{i,:}^{(l)} \cdot B_{:,j}^{(l)}. \tag{11}$$

Algorithm 1 Back propagation process for training a binarized graph convolutional layer

Input: Gradient of the layer above $\frac{\partial \mathcal{L}}{\partial H^{(l+1)}}$

Output: Gradient of the current layer $\frac{\partial \mathcal{L}}{\partial H^{(l)}}$

1: Calculate the gradients of $\tilde{W}^{(l)}$ and $\tilde{H}^{(l)}$
$\frac{\partial \mathcal{L}}{\partial \zeta^{(l)}} = \tilde{A}^T \cdot \frac{\partial \mathcal{L}}{\partial H^{(l+1)}}$
$\frac{\partial \mathcal{L}}{\partial \tilde{W}^{(l)}} = (\tilde{H}^{(l)})^T \cdot \frac{\partial \mathcal{L}}{\partial \zeta^{(l)}}$
$\frac{\partial \mathcal{L}}{\partial \tilde{H}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \zeta^{(l)}} \cdot \tilde{W}^{(l)}$
2: Calculate $\frac{\partial \mathcal{L}}{\partial H^{(l)}}$ via Eq. 14
3: Calculate $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ via Eq. 15
4: Update $\tilde{W}^{(l)}$ with the gradient $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$
5: **return** $\frac{\partial \mathcal{L}}{\partial H^{(l)}}$

Since each element of $F^{(l)}$ and $B^{(l)}$ is either -1 or 1, the inner product between these two binary vectors can be replaced by the binary operations, i.e., XNOR and bit count operations. Then, Eq. 11 can be re-written as

$$\zeta_{ij}^{(l)} = \beta_i^{(l)} \alpha_j^{(l)} F_{i,:}^{(l)} \circledast B_{:,j}^{(l)}, \tag{12}$$

where $\circledast$ denotes a binary multiplication operation using the XNOR and a bit count operations. The detailed process is illustrated in Figure 2. Therefore, the graph convolution operation in the vanilla GCN can be approximated by

$$H^{(l+1)} \approx H_b^{(l+1)} = \tilde{A}\zeta^{(l)}, \tag{13}$$

where $\zeta^{(l)}$ is calculated via Eq. 12 and $H_b^{(l+1)}$ is the final output of the l-th layer with the binarized parameters and inputs. By employing this binary multiplication operation, the original floating point calculations can be replaced with identical number of binary operations and a few extra floating-point calculations. It will significantly accelerate the processing speed of the graph convolutional layers.

4.2. Binary Gradient Approximation Based Back Propagation

The key parts of our training process include the choice of the loss function and the back-propagation method for training the binarized graph convolutional layer. The loss function employed in our Bi-GCN is the same as the vanilla GCN, as shown in Eq. 2. Since the existing back propagation method [22] has not considered the relationships among the binary weights, to perform the back-propagation for the binarized graph convolutional layer, the gradient calculation is desired to be newly designed.

To calculate the actual propagated gradient for the l-th layer, the binary approximated gradient $\frac{\partial \mathcal{L}}{\partial \tilde{H}^{(l)}}$ is employed to approximate the gradient of the original one as [15, 22],

$$\frac{\partial \mathcal{L}}{\partial H^{(l)}} \approx \frac{\partial \mathcal{L}}{\partial \tilde{H}^{(l)}} \mathbb{1}_{|\frac{\partial \mathcal{L}}{\partial \tilde{H}^{(l)}}|<1}. \tag{14}$$

Note that $\mathbb{1}_{|r|<1}$ is the indicator function, whose value is 1 when $|r| < 1$, and vice versa. This indicator function serves as a hard tanh function which preserves the gradient information. If the absolute value of the gradients becomes too large, the performance will be degraded. Thus, the indicator function also serves to kill certain gradients whose absolute value becomes too large.

The gradient of network parameters is computed via another gradient calculation approach. Here, a full-precision gradient is employed to preserve more gradient information. If the gradient of the binarized weights $\frac{\partial \mathcal{L}}{\partial \tilde{W}^{(l)}}$ is obtained, $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ can then be calculated as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}} &= \frac{\partial \mathcal{L}}{\partial \tilde{W}_{:,j}^{(l)}} \cdot \frac{\partial \tilde{W}_{:,j}^{(l)}}{\partial W_{ij}^{(l)}} \\ &= \frac{1}{d_{in}^{(l)}} B_{ij}^{(l)} \sum_k \frac{\partial \mathcal{L}}{\partial \tilde{W}_{kj}^{(l)}} \cdot B_{kj}^{(l)} + \alpha_j^{(l)} \cdot \frac{\partial \mathcal{L}}{\partial \tilde{W}_{ij}^{(l)}} \cdot \frac{\partial B_{ij}^{(l)}}{\partial W_{ij}^{(l)}}.\end{aligned} \tag{15}$$

To compute the gradient for the sign function $sign(\cdot)$, the straight-through estimator (STE) function [3] is employed, where $\frac{\partial sign(r)}{\partial r} = \mathbb{1}_{|r|<1}$. The back-propagation process is summarized in Algorithm 1.

5. Analysis

In this section, we theoretically analyze the performance of our Bi-GCN, i.e., the compression ratio of the model size and the loaded data size, as well as the acceleration ratio, respectively, compared to the full-precision (32-bit floating-point representation) GCN.

5.1. Model Size Compression

Let the parameters of each layer in the full-precision GCN be denoted as $W^{(l)} \in \mathbb{R}^{d_{in}^{(l)} \times d_{out}^{(l)}}$, which contains $(d_{in}^{(l)} \times d_{out}^{(l)})$ floating-point parameters. On the contrary, the l-th layer in our Bi-GCN only contains $(d_{in}^{(l)} \times d_{out}^{(l)})$ binary parameters and $d_{out}^{(l)}$ floating-point parameters. Therefore, the size of the parameters can be reduced by a factor of

$$PC^{(l)} = \frac{32 d_{in}^{(l)} d_{out}^{(l)}}{d_{in}^{(l)} d_{out}^{(l)} + 32 d_{out}^{(l)}} = \frac{32 d_{in}^{(l)}}{d_{in}^{(l)} + 32}. \tag{16}$$

According to Eq. 16, the compression ratio of the parameters for the l-th layer is depending on the dimension of input node features. For example, a 2-layered Bi-GCN, whose hidden layer contains 64 neurons, can achieve a ∼31x model size compression ratio compared to the full-precision GCN on Cora dataset. Although the memory consumption of the network parameters is smaller than the input data for the vanilla GCN, our binarization approach still contributes. Currently, many efforts have already been

made to construct deeper GNNs [19, 23, 10]. As the number of layers increases, the reductions on the memory consumptions will become much larger and this contribution will become more significant.

5.2. Data Size Compression

Currently, the loaded data tends to contribute the majority of the memory consumptions. In the commonly employed datasets, the node features tends to contribute the majority of the loaded data. Thus, a binarization of the loaded node features can largely reduce the memory consumptions when GNNs process the datasets. Note that the data size of the node features is employed as an approximation of the entire loaded data size in this paper, because the edges in commonly processed attribute graph is usually sparse and the size of the division mask is also small.

Let the loaded node features be denoted as $X \in \mathbb{R}^{N\times d}$, where N is the number of nodes and d is the number of features per node. Then, the full-precision X contains $N\times d$ floating-point values. In our Bi-GCN, the loaded data X can be binarized, and $N \times d$ binary values and N floating-point values can be obtained. Thus, the size of the loaded data X can be reduced by a factor of

$$DC = \frac{32Nd}{Nd + 32N} = \frac{32d}{d + 32}. \tag{17}$$

According to Eq. 17, the compression ratio of the loaded data size is depending on the dimension of the node features. In practical, Bi-GCN can achieve an average reduction of memory consumption with a factor of ∼30x, which indicates that a much bigger attributed graph can be entirely loaded with identical memory consumption. For some inductive datasets, we can then successfully load the entire graph or use a bigger sub-graph than that in the full-precision GCN. The results of data size compression can be found in Tables 2 and 3.

5.3. Acceleration

After the analysis of memory consumptions, the analysis of acceleration of our Bi-GCN, compared to GCN, is performed. Let the input matrix and the parameters of the l-th layer possess the dimensions $N \times d_{in}^{(l)}$ and $d_{in}^{(l)} \times d_{out}^{(l)}$, respectively. The original feature extraction step in GCN requires $Nd_{in}^{(l)}d_{out}^{(l)}$ addition and $Nd_{in}^{(l)}d_{out}^{(l)}$ multiplication operations. On the contrary, the binarized feature extraction step in our Bi-GCN only requires $Nd_{in}^{(l)}d_{out}^{(l)}$ binary operations and $2Nd_{out}$ floating-point multiplication operations. According to [22], the processing time of performing one cycle operation, which contains one multiplication and one addition, can be utilized to perform 64 binary operations. Then, the acceleration ratio for the feature extraction step of the l-th layer can be calculated as

$$S_{fe}^{(l)} = \frac{Nd_{in}^{(l)}d_{out}^{(l)}}{\frac{1}{64}Nd_{in}^{(l)}d_{out}^{(l)} + 2Nd_{out}^{(l)}} = \frac{64d_{in}^{(l)}}{d_{in}^{(l)} + 128}. \tag{18}$$

As can be observed from Eq. 18, the dimension of the node features $d_{in}^{(l)}$ determines the acceleration efficiency for the feature extraction step.

For the aggregation step, the sparse matrix multiplication contains $|\mathcal{E}|d_{out}^{(l)}$ floating-point addition and $|\mathcal{E}|d_{out}^{(l)}$ floating-point multiplication operations. If we let the average degree of the nodes be $\overline{deg}$, then $|\mathcal{E}| = N\overline{deg}/2$.

Then, the complete acceleration ratio of the l-th graph convolutional layer can be approximately computed via

$$\begin{aligned} S_{full}^{(l)} &= \frac{Nd_{in}^{(l)}d_{out}^{(l)} + |\mathcal{E}|d_{out}^{(l)}}{\frac{1}{64}Nd_{in}^{(l)}d_{out}^{(l)} + 2Nd_{out}^{(l)} + |\mathcal{E}|d_{out}^{(l)}} \\ &= \frac{64d_{in}^{(l)} + 32\overline{deg}}{d_{in}^{(l)} + 128 + 32\overline{deg}}. \end{aligned} \tag{19}$$

Note that the average degree $\overline{deg}$ is usually small in the benchmark datasets, e.g., $\overline{deg} \approx 2.0$ in the Cora dataset. When processing a graph with a low average node degree, the computational cost for the aggregation step, i.e., $32\overline{deg}$, usually possesses negligible effect on the acceleration ratio. Thus, the acceleration ratio of the l-th layer can be approximately computed via

$$S_{full}^{(l)} \approx S_{fe}^{(l)}. \tag{20}$$

Therefore, when $\overline{deg}$ is small, the acceleration ratio mainly depends on the input dimension of the binarized graph convolutional layers, according to Eqs. 18 and 20. The input dimension of the first graph convolutional layer equals to the dimension of the node features in the input graph. The input dimensions of the other graph convolutional layers equal to the dimensions of the hidden layers. Since the dimension of the input node features is usually large, the acceleration ratio tends to be high for the first layer, e.g., ∼59x on the Cora dataset. In general, the layer with a larger input dimension tends to require more calculations and can thus save more calculations with our binarization. For example, the acceleration ratio of a 2-layered Bi-GCN on the Cora dataset can achieve ∼59x acceleration ratio for the first layer and ∼21x for the second layer. In total, our 2-layered Bi-GCN can achieve ∼53x acceleration ratio on the Cora dataset.

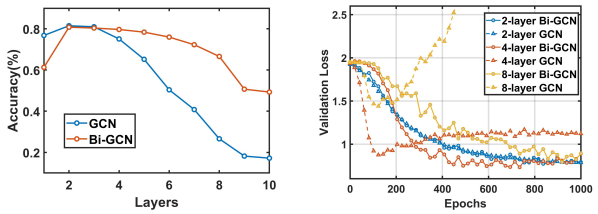
6. Evaluations

In this section, we evaluate the proposed binarization approach and our Bi-GCN on benchmark datasets for the node classification task[1]. Note that the memory consumptions

[1]More experiments can be found in the supplimentary material.

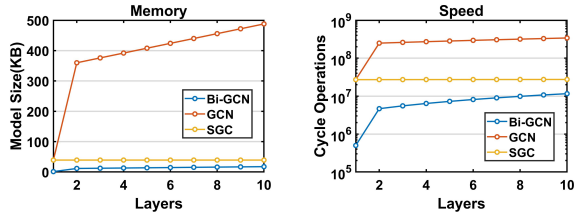
Table 1. Datasets

Dataset	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7	1,433
PubMed	19,711	44,338	3	500
Flickr	89,250	899,756	7	500
Reddit	232,965	11,606,919	41	602



(a) (b)

Figure 3. Comparisons of accuracy and validation loss on Cora.



(a) (b)

Figure 4. Comparisons of memory consumption and inference speed on Cora.

and the number of cycle operations are ideally estimated based on the specific settings of the methods and datasets.

6.1. Datasets

We conduct our experiments on four commonly employed datasets. For the transductive learning task, two commonly utilized citation networks, i.e., Cora and PubMed [25], which are also employed by GCN [18], are utilized. We adopt the same data division strategy as [32]. For the inductive learning task, Flickr and Reddit are employed. We adopt the same data division strategy as GraphSAINT [32] for Flickr and GraphSAGE [12] for Reddit. The datasets are summarized in Table 1.

6.2. Setups

For the transductive learning task, we select a 2-layered GCN [18] with 64 neurons in the hidden layer as the baseline. Our Bi-GCN is obtained by binarizing this GCN. The evaluation protocol in [18] is applied. In the training process, GCN and Bi-GCN are both trained for a maximum of 1000 epochs with an early stopping condition at 100 epochs, by using the Adam [17] optimizer with a learning rate of 0.001. The dropout layers are utilized in the training process with a dropout rate of 0.4, after binarizing the input of the intermediate layer. We initialize the full-precision weights by Xavier initialization [11]. A standard batch normalization [16] (with zero mean and variance being one) is applied to the input feature vectors in Bi-GCN. Note that we also investigate the influences of different model depths on classification performance. All the hyperparameters are set to be identical to the 2-layered case.

For the inductive learning task, we select the inductive GCN [12], GraphSAGE [12] and GraphSAINT [34] as our baselines. Note that a 2-layered GraphSAINT model is employed for fair comparisons. The settings from their own literatures are employed. We will binarize all the feature extraction steps to generalize their corresponding binarized version. The hyper-parameters in our binarized models are set to be identical to their full-precision version.

6.3. Results

6.3.1. Comparisons

The results of the transductive learning tasks are shown in Table 2. As can be observed, our Bi-GCN gives a comparable performance compared to the full-precision GCN and other baselines. Meanwhile, our Bi-GCN can achieve an average of ∼47x faster inference speed and ∼30x lower memory consumption than the vanilla GCN, FastGCN, and GAT. Besides, the proposed Bi-GCN is more effective than SGC, especially on the size of the loaded data.

For the inductive learning tasks, our binarized GNNs can significantly save the memory consumptions of both the loaded data and models, and reduce the amount of calculations with comparable performance, as shown in Table 3. The original data size of Reddit is 534.99M, while our binarized GNNs only demand 17.61M to load the data, which proves the significance of our binarization approach. Note that the acceleration ratios of binarized GNNs on Reddit dataset are only $\sim$ 10x, because the average node degree is large, as discussed in Sec 5.3. In general, these results indicate that our binarization approach is efficient and it can be successfully generalized to different GNNs.

6.3.2. Ablation Study

Here, an ablation study is performed to verify the effectiveness of the binarizations applied to the parameters and node features. As can be observed from Table 2, the prediction performances tend to vary less when the binarization is performed only to the node features. This phenomenon indicates that there exists many redundancies in the full-precision features and our binarization can maintain the majority portion of effective information for node classification. Meanwhile, the prediction results of binarizing the network parameters indicate that the binarized parameters cannot represent as much information as the full-precision parameters. However, if both the node attributes and parameters are binarized, a comparable performance can be achieved, compared to GCN. It reveals that the binarized network parameters can be effectively trained by the bina-

Table 2. Transductive learning results.(M.S., D.S., and C.O. are the abbreviations of Model Size, Data Size and Cycle Operations.)

Networks	Cora				PubMed			
	Accuracy	M.S.	D.S.	C.O.	Accuracy	M.S.	D.S.	C.O.
GCN	**81.4 ± 0.4**	360K	14.8M	2.50e8	79.0 ± 0.3	125.75K	37.6M	6.38e8
Bi-GCN(binarize features only)	81.1 ± 0.4	360K	**0.47M**	2.50e8	**79.4 ± 1.0**	125.75K	**1.25M**	6.38e8
Bi-GCN(binarize weights only)	78.3 ± 1.5	**11.53K**	14.8M	2.50e8	75.5 ± 1.4	**4.19K**	37.6M	6.38e8
Bi-GCN	81.2 ± 0.8	**11.53K**	**0.47M**	**4.67e6**	78.2 ± 1.0	**4.19K**	**1.25M**	**1.55e7**
GAT	83.0 ± 0.7	360.55K	14.8M	2.51e8	79.0 ± 0.3	126.27K	37.6M	6.44e8
FastGCN	79.8 ± 0.3	360K	14.8M	2.50e8	79.1 ± 0.2	125.75K	37.6M	6.38e8
SGC	81.0 ± 0.0	39.18K	14.8M	2.72e7	78.9 ± 0.0	5.86K	37.6M	2.98e7

Table 3. Inductive learning results. (M.S., D.S., and C.O. are the abbreviations of Model Size, Data Size and Cycle Operations.)

Networks	Reddit				Flickr			
	F1-micro	M.S.	D.S.	C.O.	F1-micro	M.S.	D.S.	C.O.
inductiveGCN	**93.8 ± 0.1**	643.00K	534.99M	4.18e10	**50.9 ± 0.3**	507.00K	170.23M	1.18e10
Bi-inductiveGCN	93.1 ± 0.2	**21.25K**	**17.61M**	**4.18e9**	50.2 ± 0.4	**16.87K**	**5.66M**	**4.65e8**
GraphSAGE	95.2 ± 0.1	1286.00K	534.99M	8.01e10	**50.9 ± 1.0**	1014.00K	170.23M	2.34e10
Bi-GraphSAGE	**95.3 ± 0.1**	**42.51K**	**17.61M**	**4.92e9**	50.2 ± 0.4	**33.74K**	**5.66M**	**6.93e8**
GraphSAINT	**95.9 ± 0.1**	1798.00K	534.99M	1.13e11	**52.1 ± 0.1**	1526.00K	170.23M	3.53e10
Bi-GraphSAINT	95.7 ± 0.1	**139.62K**	**17.61M**	**1.04e10**	50.8 ± 0.2	**65.25K**	**5.66M**	**1.28e9**

rized features, i.e., Bi-GCN can successfully reduce the redundancies in the node representations, such that the useful cues can be learned well by a light-weighted binarized network. For the memory and computational costs, binarizing the weights and features separately will reduce the memory consumption. If the binarization is performed to both of them, the inference process can also be accelerated.

6.3.3. Effects of Different Model Depths

Figure 3(a) shows the transductive results of GCN and Bi-GCN on Cora with different model depths. As can be observed, Bi-GCN is more suitable for constructing a deeper GNN than the original GCN. The accuracy of GCN has dropped sharply when it consists of three or more graph convolutional layers. On the contrary, the performance of our Bi-GCN declines slowly. According to Figure 3(b), GCN will quickly be bothered by the overfitting issue, as the number of layers increases. However, our Bi-GCN can effectively alleviate this overfitting problem. Figure 4 illustrates the comparisons of memory consumption and inference speed. Since SGC contains only one layer, its memory consumption will not change with the increase of the number of aggregations. When the number of layers increases, Bi-GCN can save more memories. For the acceleration results, the ratio between GCN and Bi-GCN tends to decrease slightly when the number of layers increases, while the actual reduced computational costs increases. Note that the required operations in SGC do not increase obviously because it only contains one feature extraction layer.

7. Conclusion

In this paper, we propose a binarized version of GCN, named Bi-GCN, by binarizing the network parameters and the node attributes (input data). The floating-point operations have been replaced by binary operations for inference acceleration. Besides, we design a new gradient approximation based back-propagation method to train the binarized graph convolutional layers. Based on our theoretical analysis, Bi-GCN can reduce the memory consumptions by an average of ∼30x for both the network parameters and node attributes, and accelerate the inference speed by an average of ∼47x, on the citation networks. Experiments on several datasets have demonstrated that our Bi-GCN can give a comparable performance to GCN in both the transductive and inductive tasks. Besides, Our binarization approach can be easily applied to other GNNs and achieve comparable results to their full-precision version.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 61802391, Grant U20B2069 and Grant 61972442, in part by the Natural Science Foundation of Tianjin of China under Grant 20JCYBJC00650, in part by the Natural Science Foundation of Hebei Province of China under Grant F2020202040, in part by State Key Laboratory of Software Development Environment (SKLSDE-2020ZX-18), and in part by the Fundamental Research Funds for Central Universities.

References

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: communication-efficient SGD via gradient quantization and encoding. In *NIPS*, pages 1709–1720, 2017.

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[5] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, pages 941–949, 2018.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *ACM SIGKDD*, pages 257–266, 2019.

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[8] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[9] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277, 2014.

[10] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks. In *AAAI*, pages 3898–3905, 2020.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[12] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[13] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling. *arXiv preprint arXiv:2006.04064*, 2020.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778, 2016.

[15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, pages 4107–4115, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[19] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *IEEE ICCV*, pages 9266–9275, 2019.

[20] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545, 2018.

[21] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, pages 747–763, 2018.

[22] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, 2016.

[23] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

[24] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[25] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE CVPR*, pages 1–9, 2015.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE CVPR*, pages 1–9, 2015.

[28] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[29] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[31] Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based graph convolutional network. In *IJCAI*, pages 4054–4061, 2019.

[32] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48, 2016.

[33] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI*, pages 7370–7377, 2019.

[34] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.