World Wide Web (2021) 24:825–848
https://doi.org/10.1007/s11280-021-00878-3

Binarized graph neural network

Hanchen Wang[1] · Defu Lian[2] · Ying Zhang[1] · Lu Qin[1] · Xiangjian He[3] · Yiguang Lin[3] · Xuemin Lin[4]

Received: 2 May 2020 / Revised: 15 March 2021 / Accepted: 16 March 2021 /
Published online: 8 April 2021

Abstract
Recently, there have been some breakthroughs in graph analysis by applying the graph neural networks (GNNs) following a neighborhood aggregation scheme, which demonstrate outstanding performance in many tasks. However, we observe that the parameters of the network and the embedding of nodes are represented in real-valued matrices in existing GNN-based graph embedding approaches which may limit the efficiency and scalability of these models. It is well-known that binary vector is usually much more space and time efficient than the real-valued vector. This motivates us to develop a binarized graph neural network to learn the binary representations of the nodes with binary network parameters following the GNN-based paradigm. Our proposed method can be seamlessly integrated into the existing GNN-based embedding approaches to binarize the model parameters and learn the compact embedding. Extensive experiments indicate that the proposed binarized graph neural network, namely BGN, is orders of magnitude more efficient in terms of both time and space while matching the state-of-the-art performance.

Keywords Graph neural network · Binarized neural network · Classification

1 Introduction

Graph analysis provides powerful insights into how to unlock the value graphs hold. Due to this power, techniques for analyzing graphs are becoming an increasingly popular topic of study in both academics and industry. To effectively and efficiently support important analytic tasks on graph data, such as node/graph classification, node clustering, community detection, node recommendation, link prediction and graph visualization, a variety of graph embedding techniques (See [5, 11] for a comprehensive survey) have been developed. Graph data is mapped into low-dimension data such that the proximity relationship among graph nodes (i.e., objects) is preserved and the off-the-shelf machine learning methods, which are designed to handle vector representations, can be immediately applied.

✉ Defu Lian
liandefu@ustc.edu.cn

Extended author information available on the last page of the article.

The existing graph embedding techniques can be roughly classified into three broad categories: (1) random walk based embedding (e.g., Deepwalk [26] and Node2vec [9]) ; (2) node similarity based embedding (e.g., LINE [38] and NetMF [30]); and (3) graph neural networks (GNN) based embedding (e.g., GCN [15], GraphSage [10], GAT [40] and AS-GCN [12]). As reported by Leskovec et al. in their tutorial on graph embedding at WWW 2018[1], the first two categories of embedding techniques are only able to learn a "shallow" representation of the graph nodes due to the simplicity of the models. It is shown in [10, 15] that the neural network based embedding methods significantly outperform the state-of-the-art techniques in the first two categories for the node classification task. Therefore, exploring how to use neural network to create a "deep" representation more efficiently is a promising direction in graph representation learning. However, most of the existing graph neural network models suffer from the scalability issue due to the high time and space cost of the real-valued model.

Recently, there have been some researches on learning binary graph embedding (e.g., [20, 37, 46]), in which each node is represented by a binary vector (code), instead of a real-valued vector. It has been shown that the binarized graph embedding can achieve much better time and space efficiency.

Time efficiency. It is well-known that the distance computation of binary vectors (i.e., Hamming distance) is much more efficient than that of real-valued vectors (e.g., Euclidian distance). In addition to the specifically tailored search algorithms (e.g., [29]), the dot product between binary vectors can also enjoy the hardware support (e.g., *xnor* and build-in CPU instruction *popcount*).

As stressed in a recent work [18] from *DeepMind*, the pairwise dot product of the vectors has been intensively used by the model for some specific tasks (e.g., graph similarity computation in [1]). Thus, the binary vector has been used in their graph matching network (GMN) to speedup the computation.

Space Efficiency. The binary embedding can represent the node in a compact way while well preserving the structure information. As shown in [20], INH-MF can achieve competitive graph node classification performance with 128 bits for each node compared to the conventional embedding approaces (e.g., DeepWalk) with 128 dimensions (i.e., 128×64 bits) per node. This will be a great advantage when we face a large-scale graph because the binarized embedding of a graph is more likely to be accommodated in the main memory.

Motivation and Challenges. The existing GNN-based methods have demonstrated outstanding performance in various tasks such as classification [10, 12, 15, 40], link prediction [14, 48], graph similarity match [1, 18] and graph clustering [41, 49]. However, they may suffer from the limitation of the memory and speed due to the use of real-valued vectors for node and graph representations and model parameters.

Given the outstanding embedding quality, various applications of the GNN-based approaches and the space and time efficiency of the binarized representation, one may wonder if we can design a binarized GNN-based graph embedding approach such that we can achieve a good trade-off between embedding quality and time/space efficiency in the GNN-based methods.

We notice that the existing binarized graph embedding methods [20, 37] rely on the discretization of the matrix factorization following the node-similarity based approaches.

[1]http://snap.stanford.edu/proj/embeddings-www

They cannot be extended to binarize the GNN-based embedding due to the inherently different natures of two categories of approaches.

As to our best knowledge, the only attempt for the binarization of GNN is from *DeepMind* in their recent work [18]. Their binarization method converts each learned d-dimensional real-valued vector into a d-dimensional ”nearly” binary vector by applying well-known binarization function *tanh* to approximate hamming distance for the binarization and optimization. However, the output of *tanh* is not exact binary value and cannot be accelerated by the binary logic operations (e.g., xnor and popcount). As an alternative, one may consider the Binarized Neural Network (BNN) (e.g., [13]) for the graph embedding so that the representation is naturally binarized. However, BNN is not designed for graph data, and as to our best knowledge, there is no existing graph embedding work based on BNN.

These issues motivate us to develop a new binarized graph embedding technique which can be integrated into existing GNN-based models to binarize the parameters and produce high-quality binarized graph embeddings. The key challenge is how to generate effective compact embedding vectors with binary network parameters in an effective way. To address the challenge, we design a binarized graph neural network framework to learn the binary parameters and representations efficiently and effectively .

Contributions. Our principle contributions are summarized as follows:

- To the best of our knowledge, this is the first study on binarized graph neural network (GNN) with binary parameters to generate binary graph representations. The proposed method, namely **BGN**, can be seamlessly integrated into the existing GNNs.
- An end-to-end binarized graph neural network framework is proposed with binary weights and activations. This binarized framework can immediately reduce the memory consumption for the network; the bit-wise operations between the binary vectors can substantially speedup the inference time of the model and the gradient estimator enables our model to effectively process back-propagation through discrete parameters and activations.
- Extensive experiments on multiple benchmark networks are conducted for node classification task. The results demonstrate that our proposed method outperforms existing binarized embedding methods with a big margin. Compare to the real-valued GNNs, our BGN model can achieve nearly state-of-the-art performance while consuming much fewer computation resources (up to $1/28$ parameter and embedding memory space and $1/20$ inference time).
- Binarization approaches are employed on the GNN-based application GMN to show that, by applying our BGN techniques, GMN model can dramatically reduce the time and space complexity while keeping the performance competitiveness.
- Experiments further show that our proposed BGN technique allows users to achieve a trade-off between the space/time and embedding quality in a flexible way by tuning different level and setting of binarization on the parameters and activations.

2 Related works

Graph Embedding. A key problem in machine learning on graphs is finding a way to incorporate information about the structure of the graph into the desired machine learning model. Graph embedding is one of the most promising approaches because it maps nodes into a low-dimensional space such that the structure of the graph is well preserved. Once accomplished, an existing machine learning approach (e.g., k-means clustering) can be used to assimilate and analyse the graph in the embedded low-dimensional space. Loosely following the seminal graph embedding approach, DeepWalk, three broad categories of embedding methods have appeared in the literature: (1) node similarity based embedding methods (e.g., LINE, NetMF), which rely on the proximity of the nodes w.r.t various similarity metrics. The matrix factorization techniques have been used to learn the embedding of the nodes. (2) Random walk based embedding methods (e.g., Deepwalk and node2vec) which encode the nodes by applying the Skip-Gram technique [25] on the random walks; and (3) graph neural networks (GNN) based embedding methods (e.g., GCN, GraphSage and GIN) [10, 12, 15, 40, 42, 45] which apply the neural network techniques on graph to learn the representations of the nodes.

Most of the existing graph embedding studies use the real-valued vector to encode the graph nodes following the above three computing paradigms. Recently, three unsupervised approaches [20, 37, 46] have been proposed to learn the *binary embedding* of the graphs following the node-similarity based embedding methods. Particularly, INH-MF [20] and DNE [37] are independently developed for binarized graph embedding based on the discretization of the matrix factorization on proximity graphs. BANE proposed in [46] is a natural extension of DNE by considering both structure and attribute similarities on the attributed graphs.

Binary Hashing. The binary hashing has been widely used to learn the binary vectors (codes) of the objects in many applications. The most popular application is the approximate nearest neighbor search in high dimension space where binary hashing methods encode high-dimensional objects (e.g., documents and images) to binary codes, while preserving similarity distance in the original space. Many learning to hash approaches have been proposed including unsupervised methods (e.g., [22, 33]), supervised methods (e.g., [35]), and deep learning based methods (e.g., [21]). Please refer to [43] for a comprehensive survey. Recently, three approaches [20, 37, 46] have been proposed to learn the binary embedding of the graphs following the node-similarity based embedding methods. As to our best knowledge, there is no existing work on the binarized graph embedding based on GNNs.

Binarized Neural Networks Binarized neural networks (see [27] for comprehensive survey) was first proposed by BNN [4]. The binarization technique proposed in [4] is used by most network binarization models. Among them, XNOR-Net [31] and DoReFa-Net [50] are the most popular ones because of their great performance on the image classification task.

XNOR-Net was proposed to have high accuracy of classification task on the ImageNet dataset while XNOR-Net has 58× faster convolutional operations and 32× memory saving. DoReFa-Net replaces the binarization by quantization which allows the model to change the bit size for weights, activations and even gradient calculations during backpropagation.

Recently, more binarized neural networks [3, 17, 23, 24, 28, 36, 47] and low bitwidth neural networks [8, 51] are proposed to further reduce the time cost of performing the machine learning model and apply these networks on the devices with low computation resources.

However, these methods are all designed for computer vision tasks. Though they perform well on the image dataset, they cannot be adapted to the graph representation learning and graph analysis task directly.

Graph Neural Network Applications There are several applications that are based on the GNN. Such as Graph Matching Network [18] and SimGNN [1]. These models utilize GNN and use the similarity (distance) of graph embedding to approximate the graph edit distance and graph similarity.

The Graph Matching Network (i.e., GMN) is a novel GNN-based framework proposed by *DeepMind* to compute the similarity score between input pairs of graphs. Separate MLPs will first map the input nodes in the graphs into vector space. Then the propagation layer will aggregate the messages of the edges and cross-graph matching vector by MLP or GRU with input concatenation of node representations and edge vectors. Matching function is applied to compute the attention coefficients based on the node information between the input pair of graphs. The matching function is based on the softmax function over node vectors which requires the calculation of vector space similarity like Euclidean, cosine similarity or dot product between all pairs of node representations. This attention coefficients calculation across two graphs requires a computation cost of $\mathcal{O}(\mid V_1 \mid\mid V_2 \mid d)$, where V_1 and V_2 indicate the number of vertices of input graph 1 and 2 respectively, and d is the dimension of the node representation. The match vector $\mu_{j\to i}$ is concatenated with the message vector $\mathbf{m}_{j\to i}$ and the node representation $\mathbf{h}_i^{(t)}$, then the concatenation is fed into MLP or a recurrent neural network core to produce the new node representations. Given the learned node representations of graph, the aggregation module proposed in [19] is used to obtain the graph representations. The similarity score in vector space such as Euclidean similarity, cosine similarity and approximate hamming similarity will be computed between graph representations to approximate the similarity between the input graphs.

3 Background and preliminaries

Recent studies have revealed that graph neural network can perform excellently on label classification tasks. The existing GNN-based graph embedding approaches share the same computing paradigm. GNNs take graph nodes' feature and neighborhood information as the input. During the training, the representations of nodes (real-valued vectors) at each layer will be updated by the aggregators and non-linear activation functions. The output representations will be fed into the task-specific layer to calculate the loss of the model. Based on that, the model will be optimized by the optimizer through backpropagation. The main differences among these GNN-based graph embedding approaches are the design of the aggregator which combines the context representations and the loss function designed for different graph analytic tasks.

These models have real-valued parameters and learn a real-valued representation for each node in an end-to-end manner for graph node classification. However, the real-valued parameters and representations are space-consuming for storage and time-consuming for multiplication computation, especially for large-scale graphs. To address these issues, in this paper we devise a novel binarized graph neural network, namely BGN, with binary parameters in the neural network to learn binary embedding representations for node classification task.

The important notations used throughout the paper are summarized in Table 1.

Table 1 Summary of notations

Notation	Definition
G	The graph dataset
$\mathcal{V}, E$	The set for nodes and edges in the graph.
$\mathbf{x}_v$	The feature information for node v.
η_v	The neighborhood nodes of node v.
$(\cdot)^{\mathbf{b}}$	Denotes that the vector or matrix is binary-valued.
$\mathbf{h}_v$	The hidden representation of node v.
$\mathbf{W}$	The weight matrix in the neural network.
$\mathcal{B}(\cdot)$	The binarization function which is used to transform the real-valued vector or matrix into binary-valued vector or matrix.
α_{ij}	The attention coefficient between node i and node j.

4 Binarized graph neural network

As illustrated in Figure 1, we introduce a new graph neural network with binarized weights and activations. Our model BGN (**B**inarized **G**raph Neural **N**etwork) is based on the attention mechanism and can be easily adapted into other graph neural network frameworks. For a given graph, BGN takes the nodes and their contexts including feature and neighborhood structure information as input. Binarization function will transform the weights, activations and even coefficients into binarized vectors to reduce the time and space complexity, while the attention mechanism enables the nodes to attend over their neighborhoods' features. We also apply the balance function to ensure that $+1$ and -1 are almost equal with each other in the binarized vectors. Furthermore, the gradient estimator is used for backpropagation of gradients through discretization.

The following subsections present the listed key components of our model:

– Section 4.1 introduces the **framework** of our work.

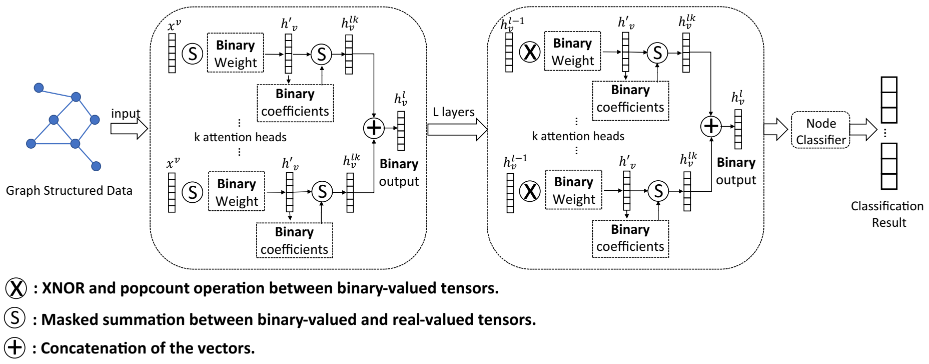

Figure 1 The overall framework of the proposed model BGN. **a** All input node features are projected into a unified representation space by binary-valued weights. **b** Masked summation between binary matrix and real-valued matrix is employed to speed up the dot product. **c** Binary attention coefficients are produced based on the hidden representations. **d** Output of the layer is calculated via multi-head attention mechanism. **e** xnor and popcount are employed to calculate the dot product between binary-valued matrix. **f** Loss calculation and end-to-end optimization for the node classification task

- Section 4.2 introduces the **binarization** of our model in detail, including the **forward propagation** and **backpropagation**.
- Section 4.3 describes the **optimization objective** of our model.
- Section 4.4 introduces the techniques we used to reduce the time and space complexity and improve the performance.
- Section 4.5 introduces the **adaptation** of our model to other GNN frameworks.

4.1 Framework

Algorithm 1 illustrates the framework of our model. We follow the attention mechanism introduced in [39, 40] to involve the *importance* of the node's neighborhoods into the graph representation learning process. Given a graph $G(\mathcal{V}, E)$, where $\mathcal{V}$ and E denote the set of graph nodes and edges respectively, we use nodes features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$, $\mathbf{x}_v \in \mathbb{R}^m$ and the neighborhood information of nodes $\{\eta_v, \forall v \in \mathcal{V}\}$ as inputs. $Balance(\cdot)$ denotes the balance function which is introduced in Section 4.4.3. Our model will first produce the binarized node representations $\mathbf{h}_v^b \in \{+1, -1\}^d$ for each node within the input graph. After that, the binarized node embeddings will be fed into the output layer to compute the loss for some specific tasks like node classification.

Algorithm 1 Binarized graph neural network.

Input: Graph $G(\mathcal{V}, E)$, node features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$, number of layers L, binarization function $\mathcal{B}(\cdot)$, number of attention heads K, neighbors of node $\{\eta_v, \forall v \in \mathcal{V}\}$
Output: Classification result $\{\mathbf{C}_v, \forall v \in \mathcal{V}\}$

```
1:  Let $\mathbf{h}_v^0 = \mathbf{x}_v, \forall v \in \mathcal{V}, \forall u \in \eta_v$;
2:  for $l = 1, 2, \ldots, L-1$ do
3:      $\alpha_{uv}^l = \mathcal{B}(Softmax_v(\mathbf{W}_0^l \mathbf{x}_u, \mathbf{W}_0^l \mathbf{x}_v))$
4:      for $v \in \mathcal{V}$ do
5:          for $k \in K$ do
6:              $\mathbf{h}_v^k = \mathcal{B}(\sum_{u \in \eta_v} Balance(\alpha_{uv}^k \mathbf{W}_k^l \mathbf{x}_u))$;
7:          $\mathbf{h}_v^l = Concat_{k=1}^K(\mathbf{h}_v^k)$
8:  $\alpha_{uv}^L = \mathcal{B}(Softmax_v(\mathbf{W}_0^L \mathbf{h}_u^{L-1}, \mathbf{W}_0^L \mathbf{h}_v^{L-1}))$
9:  for $v \in \mathcal{V}$ do
10:     $\mathbf{C}_v = Softmax(\sum_{u \in \eta_v} \alpha_{uv}^k \mathbf{W}^L \mathbf{h}_u^{L-1})$
11: return $\mathbf{C}_v$, the classification result for node $v \in \mathcal{V}$
```

Attention Mechanism Our proposed framework is based on the graph attention mechanism. The attention layer is utilized in our model to learn the importance of every node to other nodes. The key is to get the *importance* of one node's feature to other nodes that is the attention coefficients of the input graph, afterwards, the node's feature can attend on other nodes. Inspired by [40], we perform *masked attention* to the model to keep the structural information of the input graph. Only the attention coefficients of one node with its neighborhood nodes i.e., α_{ij}, $v_j \in \eta_i$ will be computed.

In order to obtain the attention coefficients, we use a shared **binarized** weight matrix $\mathbf{W} \in \{+1, -1\}^{m \times d'}$ to apply the linear transformation to each node. Softmax function is used to normalize the coefficients, but unlike the model proposed in [40], LeakyRelu activation is not employed in our model while the sign function is used to binarize the attention coefficients. With the following (1), we will get a **binarized** attention coefficient matrix $\mathcal{A} \in \{+1, 0, -1\}^{N \times N}$ where α_{ij} is the element of the matrix $\mathcal{A}$ (0 is contained in the matrix since we only compute the attention coefficients between neighbors such that the matrix is sparse).

$$\alpha_{ij} = \mathcal{B}'(Softmax_j(\mathbf{W}\mathbf{x}_i, \mathbf{W}\mathbf{x}_j)) \tag{1}$$

where $\mathcal{B}'$ is the binarization function for attention coefficients which maps 0 to 0, positive values to $+1$ and negative values to -1.

Once the attention coefficient matrix is obtained, it will be used to compute the output of the attention layer. The attention coefficients will multiply the linear transformed node feature. We employ the *multi-head attention mechanism* to stabilize the learning process. The binarization function, which is served as an activation function, is applied to every attention head to binarize the pre-activations. And concatenation of the output of K independent attention head is the output of the attention layer. Therefore, the output node representation will be like following:

$$\mathbf{h}_i = \|_{k=1}^{K} \mathcal{B}(\sum_{j \in \eta_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{x}_j) \tag{2}$$

Where $\|$ means the concatenation of the vectors and $\mathbf{h}_i$ is the output **binarized** node representation where $\mathbf{h}_i \in \{+1, -1\}^d$.

After several attention layers, the node representation will be fed into the last layer to calculate the loss for specific task which is classification in this paper. We will introduce the learning objective in the Section 4.3.

4.2 Binarization

In this section, we introduce how to obtain a graph neural network with binary parameters that can learn binary representations. Section 4.2.1 introduces the binarization function used to transform the real-valued parameters and pre-activations into binary space. Section 4.2.2 introduces the gradient estimators that enable the binarized model to be optimized by the off-the-shelf optimizers such as Adam and SGD.

4.2.1 Forward propagation

Binarization function is important in our model. Specific binarization function will be chosen in the forward propagation calculation process to binarize the weights and the activations. In that way the low-bit parameters and activations will help to reduce the time and space complexity. In our case, various binarization functions will work, and the most straightforward example is the sign function. As mentioned in [4] and [31], deterministic and stochastic binarization based sign function are widely applied to the continuous pre-activations as well as the real-valued weights to obtain binarized activations and weights.

$$\mathcal{B}_{det}(x) = \begin{cases} +1 & x \geq 0, \\ -1 & else, \end{cases} \tag{3}$$

The above equation is the deterministic binarization function, where x is the real-valued variable. The stochastic binarization is the sign function with probability:

$$\mathcal{B}_{stoch}(x) = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases} \tag{4}$$

where σ denotes the sigmoid function, that is $\sigma(x) = 1/(1 + exp(-x))$. The stochastic binarization is more appealing but needs the computer to generates random bits while the deterministic binarization is easier to calculate. Deterministic binarization function(i.e., (3)) is applied for the binarization of weights and activations because the deterministic sign function provides more stable and reproducible results. Please note that we use a variant of deterministic sign function which maps 0 to 0 to binarize the attention coefficients.

Other than directly binarize the weights in the graph neural network, we follow the quantization process in [31] to add a scaling factor $\gamma \in \mathcal{R}^+$ to estimate a real-valued weights such that $\boldsymbol{W} \approx \gamma \boldsymbol{B}$, thus achieve better performance. We can find the optimal quantizer by minimizing the quantization error:

$$\min \mathcal{J}(\gamma \boldsymbol{B}) = \|\boldsymbol{W} - \gamma \boldsymbol{B}\|^2 \tag{5}$$

According to results and analysis in [31], for each real-valued weight $\boldsymbol{W}$, the optimized binary matrix $\boldsymbol{B}^*$ and scaling factor γ^* can be obtained by the following constrained optimization:

$$\boldsymbol{B}^* = \arg\min_{\boldsymbol{B}} \boldsymbol{W}^T \boldsymbol{B} \tag{6}$$

$$\gamma^* = \frac{\boldsymbol{W}^T \boldsymbol{B}^*}{n} = \frac{1}{n} \|\boldsymbol{W}\|_{\mathcal{L}1} \tag{7}$$

where $\boldsymbol{B}$ is constrained to be a binarized matrix, n is the number of elements within the weight $\boldsymbol{W}$, if $\boldsymbol{W} \in \mathcal{R}^{m \times d}$, then $n = m \times d$.

Furthermore, we also adopt the Libra Parameter Binarization (LPB) introduced in IR-net [28] to retain the information and minimize the information loss in forward propagation by jointly considering both quantization error and information loss. LPB also quantify the real-valued weight $\boldsymbol{W}$ using a scaling factor such that $\boldsymbol{W} \approx \gamma \boldsymbol{B}$. Suppose each element in $\boldsymbol{B}$ can be viewed as a sample of random variable obeying Bernoulli distribution shown in (4). The entropy of the quantization in the following (8) is also considered as a part of loss function:

$$\mathcal{H}(\gamma \boldsymbol{B}) = \mathcal{H}(\boldsymbol{B}) = -p\ln(p) - (1-p)\ln(1-p) \tag{8}$$

Together with the quantization loss described in (5), the objective function of LPB is defined as:

$$\min \mathcal{J}(\gamma \boldsymbol{B}) - \lambda \mathcal{H}(\gamma \boldsymbol{B}) \tag{9}$$

We further apply the standardization and balance described in [28]. As a result, the optimal quantization can be obtained by solving:

$$\gamma^* \boldsymbol{B}^*_{\boldsymbol{W}} = \mathcal{B}_{det}(\widehat{\boldsymbol{W}}_{std}) \ll\gg s^* \tag{10}$$

where $\ll\gg$ is left or right bit-shift, s^* and $\widehat{\boldsymbol{W}}_{std}$ can be calculated by:

$$s^* = round(\log_2(\frac{\|\widehat{\boldsymbol{W}}_{std}\|_{\mathcal{L}1}}{n})) \tag{11}$$

$$\widehat{\boldsymbol{W}}_{std} = \frac{\widehat{\boldsymbol{W}}}{\sigma(\widehat{\boldsymbol{W}})}, \widehat{\boldsymbol{W}} = \boldsymbol{W} - \bar{\boldsymbol{W}} \tag{12}$$

where $\sigma(\cdot)$ denotes the standard deviation and $\bar{\boldsymbol{W}}$ is a matrix whose elements are all mean value of weight $\boldsymbol{W}$. LPB directly binarize the representations using the deterministic binarization function, i.e., $\boldsymbol{B}_{\boldsymbol{x}} = \mathcal{B}_d et(\boldsymbol{x})$. Hence, the operation between the real-valued weights and vectors is reformulated as follows:

$$\boldsymbol{W}\boldsymbol{x} = (\boldsymbol{B}_{\boldsymbol{W}} \odot \boldsymbol{B}_{\boldsymbol{x}}) \ll\gg s \tag{13}$$

where $\odot$ denotes the *XNOR* and *popcount* operation between binary codes.

4.2.2 Backpropagation

In this part, we describe how to backpropagate the gradients through the binarization function. We adapt the gradient estimator into our model for better optimization.

Propagation gradients through binarization function It is obvious that the binarization function has zero derivative almost everywhere, which leads to the zero gradients of the loss function w.r.t the pre-activations and weights. The trainable variables cannot be updated with zero gradient. Therefore, the model cannot be trained by simple backpropagation, and the estimation of the gradients should be obtained for optimization. Previous studies have investigated how to propagate gradients through stochastic discrete functions. Below we investigate two popular unbiased gradient estimators for binarization function: straight through estimator and REINFORCE estimator [44].

Straight through estimator The straight-through estimator is proposed a simple unbiased gradient estimator. It estimates the derivative of binarization function $\mathcal{B}(\mathbf{h})$ of pre-activation or weight $\mathbf{h}$ as $\mathbf{1}$ (a vector or matrix whose elements are all 1). Let $\mathbf{h}^b$ denote the binarized representation and $\mathbf{h}$ denote the pre-activation before binarization. The straight-through estimation of the gradient of the loss L w.r.t the pre-activation $\mathbf{h}$ is thus:

$$g_h = \frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathcal{B}(\mathbf{h})} \cdot \frac{\partial \mathcal{B}(\mathbf{h})}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathbf{h}^b}\mathbf{1} = g_{h^b}\mathbf{1} \tag{14}$$

This gradient will then be back-propagated to obtain the gradient of quantities (i.e., pre-activations or weights) that influence $\mathbf{h}$.

REINFORCE estimator The reinforce estimator is proposed in [2] to estimate the expectation of the gradient $\frac{\partial L}{\partial \mathbf{h}}$ of loss L with regard to the pre-activation vector or weight $\mathbf{h}$. When binarization function $\mathcal{B}(\cdot)$ is stochastic with the probability given by sigmoid, it has been proven that:

$$\mathbb{E}(\frac{\partial L}{\partial \mathbf{h}}) = \mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))(L - c)] \tag{15}$$

where σ is the sigmoid function and c is a constant vector. To minimize the variance of the estimation, c can be chosen as:

$$c = \frac{\mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))^2 L]}{\mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))^2]} \tag{16}$$

The reinforce estimator can work directly on the weights and pre-activations without actual computation of the gradient. The estimation is obtained by monitoring numerator and denominator during the training process.

Compared with straight through estimator, reinforce estimator is more advanced with better performance in many applications. However, we observe that its performance is not superior than the straight through estimator. On the other hand, straight through estimator helps the model to obtain the gradient faster than the reinforce estimator due to its simplicity. The comparison between these two gradient estimators with regards to the performance is included in Section 5. In practice, we choose straight through estimator for our model in the experiments.

4.3 Optimization objectives

Existing GNN-based graph embedding approaches provide an end-to-end model, which focuses on the node classification task. Therefore, our model is also learned for the node classification task. Below, we introduce the objective of BGN and the learning process that optimizes the parameters.

For the node classification learning, we feed the binarized embedding $\mathbf{h}_v^b$ into the output layer to predict the class label for the node. The predicting probability of label $\mathbf{C}_i$ is written as:

$$p(\mathbf{C}_{vk} \mid \mathbf{h}_v^b) = Softmax_\zeta^k(\sum_{u \in \eta_v} \alpha_{uv}^L \mathbf{W}^L \mathbf{h}_u^b) \tag{17}$$

where ζ denotes the number of labels for each node. After obtaining the classification result in (17), we calculate the cross-entropy as the loss for the node classification task.

$$L_{class} = -\sum_{v \in \mathcal{V}_{labeled}} \sum_{k=1}^{\zeta} \mathbf{C}_{vk}^{\mathcal{L}} \log(\mathbf{C}_{vk}) \tag{18}$$

where $\mathcal{V}_{labeled}$ is the set of nodes that have label information which are used for training process, $\mathbf{C}_{vk}^{\mathcal{L}}$ is the multi-hot encoding for ground truth classification labels.

The gradients will be back propagated via estimator and be applied on the optimization of parameters by the off-the-shelf optimizer during the training process.

4.4 Techniques to improve the model

Several techniques are used on binarized graph neural network model to reduce the time and space complexity and improve the performance. Logic operation *XNOR* between binary values, build-in CPU instruction *popcount* and the masked summation are used to replace the traditional arithmetic operation dot product to reduce time complexity. The Figure 2 is a toy example that introduces the differences between these operations. The balance function is used to make $+1$ and -1 to be balanced in the embedding vectors which can raise the performance of the GMN. Also, the binary parameters of the neural network and the binary node representations can reduce the space complexity intuitively.

4.4.1 XNOR and popcount

The logic *XNOR* and CPU build-in instruction *popcount* between binary matrices are used to replace the dot product between them.

As shown in Table 2, *XNOR* produces binary value with input of $+1$ and -1. Instruction *popcount* is then be employed to count the number of bits that is set to 1. The *XNOR* can be more than one order of magnitude faster than the dot product which can dramatically reduce the time complexity. As mentioned in [4], a 32-bit floating point multiplier costs about 200 Xilinx FPGA slices, whereas a 1-bit *XNOR* gate costs only 1 slice.

4.4.2 Masked summation

The masked summation is used to replace the dot product between binary matrix and real-valued matrix. The binary matrix will be transformed into the mask matrix with ”True” and ”False”. During the multiplication, the real-valued vector will be masked by the corresponding mask vector, then the positive and negative masked vector are produced with only the

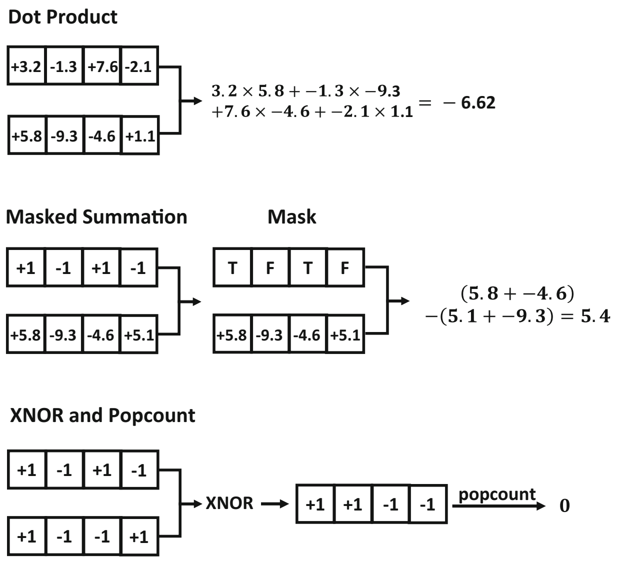

Figure 2 The toy examples of (**a**) dot product (**b**) Masked summation and (**c**) *XNOR* and *popcount* instruction

elements at the same position as "True" and "False" on the mask vector. The model calculates the summations of the positive and negative masked vector separately. The subtraction of these two summation results is the result of dot product between the given matrices.

The masked summation can reduce the time complexity of dot product of two matrix. Usually, the time complexity of naive dot product between two real-value matrices $M_1 \in \mathbb{R}^{m\times n}$ and $M_2 \in \mathbb{R}^{n\times d}$ is $\mathcal{O}(mnd)$, while the time complexity of *masked summation* between binary matrix $M_1 \in \{-1, +1\}^{m\times n}$ and real-valued matrix $M_2 \in \mathbb{R}^{n\times d}$ is $\mathcal{O}(nd)$. Theoretically and also in practice, the masked summation can significantly reduce the time complexity of our proposed binarized graph neural network.

4.4.3 Balance function

The distribution of $+1$ and -1 is sometimes unbalanced in the representation vectors. For example, if most pre-activations $\mathbf{h}$ have *positive* elements, the output graph representation vector of binarization function $\mathbf{h}^b$ will be formed mainly by $+1$. Then the dot product of two vectors will be d which is the dimension of the vectors. This unwanted situation should be avoid because it dramatically lower the effectiveness of the proposed model, especially when the BGN is applied to GMN which requires a great number of dot product between

Table 2 *XNOR* calculation

Input A	Input B	Output
+1	**+1**	**+1**
+1	**-1**	**-1**
-1	**+1**	**-1**
-1	**-1**	**+1**

representations. As a result, we apply the following balance function to the pre-activations before binarization in order to balance the distribution of *positive* and *negative* elements of pre-activations:

$$Balance(\mathbf{h}) = \mathbf{h} - \overline{\mathbf{h}} \tag{19}$$

Where the $\overline{\mathbf{h}}$ is the vector whose elements are all mean value of the pre-activation vector $\mathbf{h}$. The balance function ensures that the pre-activation vectors contain almost half *positive* and half *negative* elements, which leads to the balance distribution of $+1$ and -1 after binarization.

4.5 Adapted to Other GNN based models

The proposed binarized graph neural network is a very general framework that can be adapted to other graph neural network-based model to project the real-valued parameters and activations into the binary space to reduce the space and time cost. We introduce how we binarize the state-of-the-art GNN-based model AS-GCN [12] and the graph matching network.

4.5.1 Binarization of AS-GCN

AS-GCN is a general framework that is designed for fast representation learning based on graph neural networks such as GCN. Therefore, the binarization of AS-GCN is similar to our proposed BGN. We use deterministic binary function to binarize the parameters and pre-activations of AS-GCN. And straight through estimator is employed for back propagation. The binarized model is denoted as BGN-ASGCN in our experiment.

4.5.2 Binarization of GMN

As mentioned above, the time cost of GMN comes mainly from the pair-wise node similarity computation. We utilize the deterministic binarization function (3) on the preactivations and transform the node and graph representations into binary codes such that the XNOR can be applied to replace the dot product. Straight through estimator (14) is used for the back propagation. Furthermore, we noticed that the distribution of $+1$ and -1 is usually not symmetric which dramatically lower the performance, hence, balance function (19) is employed on the graph representations.

5 Experiment

We conduct extensive experiments to evaluate the performance of our model for the node classification task on real-world network datasets. We compare the time and space efficiency thoroughly between the proposed model and other baseline models. The case study shows the effectiveness and efficiency brought by our framework on the GNN-based application such as GMN.

5.1 Dataset

To facilitate the comparison between our model and the relevant baselines, we conduct the classification experiments on three well-known citation network datasets: **Cora**, **Citeseer** and **Pubmed** [34]. Each dataset contains bag-of-words representations of documents and citation links between the documents. Graph G is constructed based on the citation links. In the classification task, we only use 20 labeled instances per class for training. The test data contains 1000 nodes as in GCN, GAT and AS-GCN. We also include other types of dataset for extensive comparisons. The experiments are also conducted on two social network datasets: **Facebook** and **wiki-vote** [16] and two air-traffic networks [32]: **Brazil** and **USA**. For social networks, we randomly select 10% and 20% of nodes for training and validation respectively, and the rest of nodes are used as test set. For air-traffic networks, we randomly assign equal number of nodes in training, validation and test sets.

The details of the datasets are summarized in the Table 3.

5.2 Baseline methods

The following GNN-based and binary embedding methods are compared as baselines:

GCN (Graph Convolutional Network) [40] is a semi-supervised neural network method for node classification.

GAT (Graph Attention Network) [40] is a graph neural network model which first exploits the attention mechanism to solve the node classification task.

AS-GCN (Adaptive Sampling over GCN) [12] is a state-of-the-art method for node classification task. AS-GCN aims to increase the scalability of GCN using adaptive sampling. The experiments demonstrate that the application of BGN can further reduce the time and space complexity of AS-GCN.

GAT-binary and **ASGCN-binary** are the models that directly apply sign function on the node representations learned by the original version of GAT and AS-GCN. The naively binarized representations will be fed into the task-specific layer to learn the classification result.

GAT-tanh and **ASGCN-tanh** are the models that employ the binarization function *tanh* used by *DeepMind*'s work. *tanh* function is used to binarize the parameters and embedding vectors of GAT and AS-GCN. We clip the value of the parameters and activations in both models to make sure that *tanh* can produce ”exact” binary codes.

INH-MF [20] is a MF-based information network hashing algorithm that learns binary codes as node embedding which can preserve high-order proximity.

BANE(Binarized Attributed Network Embedding) [46] is an extension of DNE [37] which based on the Weisfeiler-Lehman proximity matrix factorization learning function to produce binary node representations.

Besides, we also compared the variants of our proposed BGN with the quantization methods introduced in Section 4.2.1:

Table 3 Citation datasets

Dataset	#Nodes	#Edges	#Classes	#Labled Nodes
Cora	2708	5429	7	140
Citeseer	3327	4732	6	120
Pubmed	19717	44338	3	60
Facebook	4039	88234	4	403
wiki-vote	7115	103689	4	711
Brazil	131	1038	4	43
USA	1190	13599	4	396

BGN-x-GAT and **BGN-x-ASGCN** are the binarized models of graph attention network and ASGCN using (6) and (7).

BGN-lpb-GAT and **BGN-lpb-ASGCN** are the binarized models of graph attention network and ASGCN using Libra Parameter Binarization (LPB).

5.3 Experiment setup

For the performance experiment, we evaluate the models with the same bit-width representations. For the experiment of inference efficiency, the embedding dimension of our method and other baseline methods are all set to 64. During training process, the whole graph can be seen, but only a few nodes are labeled while most nodes have no label information. We put all nodes information in one training phase due to the need of calculation for graph attention coefficients.

For this classification task, we report the average accuracy of the evaluated GNN-based embedding approaches after ten independent runs using the accuracy metric introduced in [15, 40]. Because INH-MF and BANE only produce the binary embedding vectors but have no build-in classifier, we employ the one-vs-rest logic regression implemented by Liblinear [7] to obtain the classification result of the networks, in which 90% nodes are labeled.

All the experiments were conducted on the server which is running RHEL 7.5 and has 2x 2.4GHz Intel Xeon E5-2680 v4 (14 Cores) CPU, 256GB 2400MHz ECC DDR4-RAM and 2x NIVDIA Quadro P5000 16GB Graphics Card (GPUs) (2560 Cores). The time cost of our trained binarized model is evaluated on the CPUs using the *XNOR* and *popcount* instructions. The time cost of other baseline GNN-based methods is evaluated on GPUs.

5.4 Classification results

Because our model produces the compact representations for vertices, we compare the performance between our model and other baselines with the same bit width.

5.4.1 Comparison among binary embedding methods

We compare the classification results between our model and other binary-valued embedding methods.

As shown in the Figure 3, under different embedding dimensions, BGN outperforms all the other binary-valued embedding methods significantly on all three datasets. With the help of the graph neural network, our model can make better use of the graph structured data and feature information and is trained specifically for the node classification task. Therefore, our model outperforms other MF-based binarized graph embedding models by a significantly large margin. In comparison with the naively binarized GAT-binary and ASGCN-binary, our model considers the binary property of parameters and vectors during the training process, hence our model achieves better accuracy. In terms of GAT-tanh and ASGCN-tanh, because *tanh* function has zero gradient when the output is nearly $+1$ or -1 and has real-value output when the gradient is not zero. This property determines that *tanh* function is not suitable for binarizing the neural network. When the input values are clipped to produce exact binary parameters and embeddings via *tanh* function, the gradient will be zero which results in the insufficient optimization and worse performance than BGN. Furthermore, the models that are binarized by BGN-x and BGN-lpb achieve better performance than other compared methods.

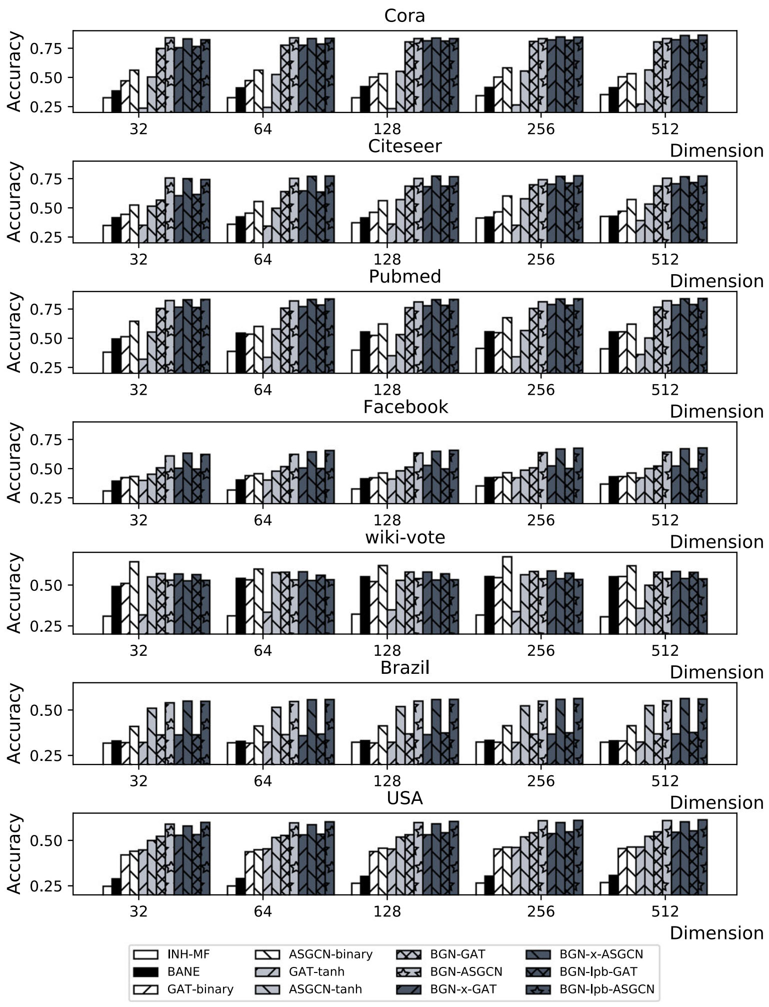

Figure 3 Classification results of three citation network dataset among the binary-valued embedding methods with different embedding dimensions

5.4.2 Comparison among the GNN-based methods

We compare our model with other GNN-based methods (GCN, GAT and AS-GCN). All baseline methods produce the real-valued embedding vectors each dimension of which is encoded by at least 32 bits. Compared with these methods, each dimension of the embedding vectors learned by our model is only encoded by 1 bit. As a result, a real-valued 16 dimension vector requires at least 256 bits while a binary vector only requires 16 bits. Figure 4 shows the performance of the models with bit width varies for a single embedding vector.

Our model significantly outperforms all the baseline methods with low bit width. When getting more space for the learned representations, our model can still achieve competitive classification results compared with the state-of-the-art graph neural network-based methods. In conclusion, the performance gap between our model and baselines with large

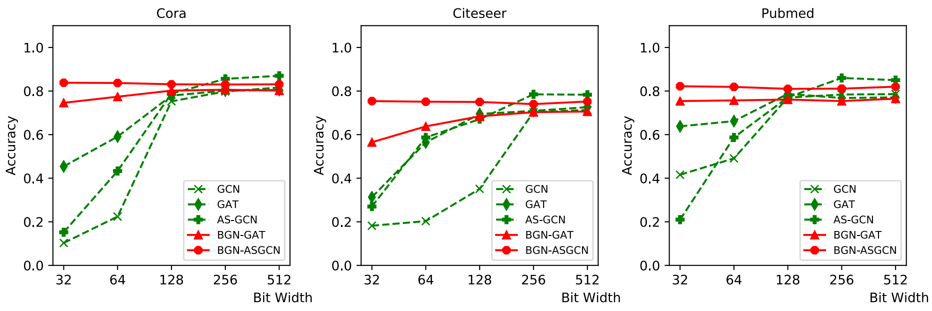

Figure 4 Classification results of three citation network dataset among the GNN-based methods with varied bit width for embedding vector

bit-width representations is acceptably small while our model's performance is notably better with the low bit-width representations.

5.5 Comparison of time and space efficiency

In this section, we report the inference time and space efficiency of our model. The inference is the process that produces the classification result when we have already trained the model. Acceleration is brought by the *XNOR* and *popcount* operation with just little sacrifice on the classification performance. In this experiment, we train the binary parameters and activations of our model, then replace dot product operation between binarized matrices by *XNOR* and *popcount* and also replace the dot product between binary matrix and real-valued matrix by masked-summation during the inference process.

Tables 4 and 5 report the experiment results. GAT-binary, ASGCN-binary, GAT-tanh and ASGCN-tanh require the same size of parameters as their real-valued version since these baslines only binarize the representations of the nodes while keep using the real-valued parameters in the model. Among these models, GAT-binary and ASGCN-binary can be accelerated by applying the mask summation during inference process, while GAT-tanh and ASGCN-tanh have to perform conventional matrix multiplication since these two methods cannot guarantee to produce exact binarized codes as node representations. Our model under the binarized framework is more than one order of magnitude faster than the baseline

Table 4 Comparison of performance, inference time and memory space required for the parameters between the real-valued and binarized models

Dataset		GAT	AS-GCN	GAT-binary	ASGCN-binary
Cora	Time(s)	1.9×10^{-1}	1.0×10^{-1}	1.5×10^{-1}	8.2×10^{-2}
	Space(bit)	2.46×10^{8}	3.04×10^{6}	2.46×10^{8}	3.04×10^{6}
	Accuracy	84.0%	87.3%	47.1%	56.1%
Citeseer	Time(s)	2.8×10^{-1}	2.8×10^{-1}	2.4×10^{-1}	2.5×10^{-1}
	Space(bit)	7.60×10^{6}	7.83×10^{6}	7.60×10^{6}	7.83×10^{6}
	Accuracy	72.1%	78.9%	45.3%	55.3%
Pubmed	Time(s)	3.8×10^{1}	4.54×10^{0}	3.5×10^{1}	3.97×10^{0}
	Space(bit)	1.03×10^{6}	1.06×10^{6}	1.03×10^{6}	1.06×10^{6}
	Accuracy	78.2%	89.0%	53.4%	60.0%

Table 5 Comparison of performance, inference time and memory space required for the parameters between the tanh-based and BGN-based models

Dataset		GAT-tanh	ASGCN-tanh	BGN-GAT	BGN-ASGCN
Cora	Time(s)	1.8×10^{-1}	8.2×10^{-2}	1.0×10^{-2}	$8.0 \times 10^{-}3$
	Space(bit)	2.46×10^{8}	3.04×10^{6}	1.32×10^{7}	1.97×10^{5}
	Accuracy	24.2%	52.3%	77.7%	84.1%
Citeseer	Time(s)	2.5×10^{-1}	2.9×10^{-1}	1.4×10^{-2}	1.8×10^{-2}
	Space(bit)	7.60×10^{6}	7.83×10^{6}	2.49×10^{5}	4.86×10^{5}
	Accuracy	34.2%	49.5%	63.7%	77.2%
Pubmed	Time(s)	4.1×10^{1}	4.47×10^{0}	1.1×10^{0}	2.1×10^{-1}
	Space(bit)	1.03×10^{6}	1.06×10^{6}	3.85×10^{4}	7.01×10^{4}
	Accuracy	33.6%	57.9%	75.7%	82.0%

methods GAT and AS-GCN with regards to the inference time. The proposed model can be up to 29× faster and save up to 28× space compared with the baseline methods.

5.6 Analysis of binarization

In this section, we introduce the effect of the estimator and binarization level with regard to the space, time and performance. We compare the space, inference time and performance between BGN-GAT and GAT on the Cora dataset. We fix the dimension of embedding vector to 64 for both methods and change the setting of BGN to show the space and time saving compared with the baseline GAT.

Result is shown in Table 6 where BGN^{w}, BGN^{e}, BGN^{we} and BGN^{wec} mean that the BGN is with weights binarized, embedding vectors binarized, weights and embedding vectors binarized, weights, embedding vectors and attention coefficients binarized based on the graph attention mechanism respectively. We can conclude from the Table 6 that (1) when the weights, activations and attention coefficients are all binarized, the BGN-GAT can save largest space for parameters and the output vectors while holding acceptable classification accuracy. (2) Straight through estimator and reinforce estimator have similar accuracy on

Table 6 Trade-off between time/space efficiency and classification accuracy of proposed BGN w.r.t the level and setting of binarization

Method	Estimator	Param space	Vec space	Speed up	Accuracy
GAT	N/A	1×	1×	1×	84.0%
BGN^{w}	STE	1/28	1×	3.7×	80.5%
BGN^{w}	Reinforce	1/28	1×	3.8×	80.3%
BGN^{e}	STE	1×	1/1.02	1.3×	81.2%
BGN^{e}	Reinforce	1×	1/1.02	1.2×	81.3%
BGN^{we}	STE	1/28	1/1.02	5.7×	77.2%
BGN^{we}	Reinforce	1/28	1/1.02	6.1×	77.5%
BGN^{wec}	STE	1/28	1/19	18.7×	77.7%
BGN^{wec}	Reinforce	1/28	1/19	19.1×	76.9%

the node classification task. Therefore, we choose the STE for our model in the above experiments because of its simplicity and certainty. (3) Compared with original GAT, BGN-GAT can save 28× space for model parameters, 19× space for activations and achieve 19× speed up.

5.7 Case study

In this section, we investigate how binarized graph neural network improve the time efficiency of the GNN-based applications such as GMN. Because GMN needs to compute the pair-wise dot product between node and graph embedding vectors, the time consumption is extremely high when the number of nodes in each graph goes up. However, with the binary representations, we can apply *XNOR* between binary vectors to replace the dot product, which will alleviate the time complexity problem significantly. The following experiment results will introduce the performance and time complexity of GMN with binary node and graph representations compared with the origin version. The graph similarity will then be used for the graph matching task.

Experiment Setup We follow the experiment setting of [18] to test the performance of Binarized GMN. The training data is generated by sampling binomial graphs G_1 with n nodes and edge probability p [6]. Then the positive example G_2 is generated by randomly substituting k_p edges from G_1 with new edges and negative example G_3 is generated by substituting k_n edges from G_1, where $k_p < k_n$. In the experiment, we set $k_p = 1, k_n = 2$ and $p = 0.2$. We also set the hamming similarity between vectors as loss function, which is more suitable for the binary-valued vectors as the loss function to train the model. The model needs to predict a higher similarity score for positive pair (G_1, G_2) than negative pair (G_1, G_3). The evaluation metric remains the same: (1) pair AUC - the area under the ROC curve for classifying pairs of graphs as similar or not on a fixed set of 1000 pairs and (2) triplet accuracy - the accuracy of correctly assigning higher similarity to the positive pair in a triplet than the negative pair on a fixed set of 1000 triplets.

Inference time and Graph Matching Performance We report the graph matching accuracy and inference time of the binarized and original GMN with regards to the number of nodes in each graph. The default setting in GMN is 20 nodes per graph, which is quite small for real-world networks. We set the number of nodes in one graph from 20 to 160 and keep other settings the same as described above to evaluate the performance and inference time. The dimensions of node and graph representations are set to 32 and 64 respectively.

As shown in Figure 5, the inference of BGN-GMN is significantly faster. This is because of the fact that the similarity computation (pair-wise dot product) between node representations of two graphs mainly accounts for the time complexity of GMN. Under the same dimension of node and graph embedding vectors, BGN-GMN is up to 21× faster than the baseline model in terms of the inference time with the help of the replacement of dot product by fast operations such as *XNOR* and *popcount* between binary vectors.

In terms of graph matching task, the original version of GMN has better performance when the number of nodes in each graph is small. However, when the number of nodes gets larger, the pair AUC and triplet accuracy will both decay. When the number of

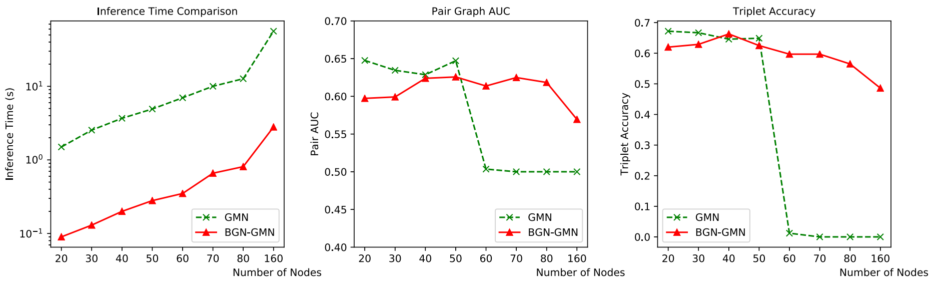

Figure 5 The performance of graph matching and inference time for GMN and BGN-GMN w.r.t the number of nodes per graph

nodes is more than 60, the real-valued representations cannot tell the similarity difference between the graphs. Hence, the model is not able to learn the different similarity scores for positive and negative pairs of graphs with the hamming similarity metric. However, with the help of binarization and balance function, the binary representations still hold an acceptable and more robust performance for the graph matching task. This is due to the fact that the binarized model produces *true* binary representations for the calculation of hamming loss and is designed for the graph matching task specifically on hamming space.

Parameter Sensitivity Analysis We compare the performance of binarized and original version GMN to show the effect of dimension for node and graph embedding vectors. We set the number of nodes in each graph $n = 30$ for this comparison. We change the dimension of graph embeddings produced by two models to ensure them to produce the same bit-width embedding vectors and keep the other settings as the same to compare the performance of two models.

The result is included in Figure 6a. We can find that the binary graph representations tend to have better performance when they are low bit-width and have similar accuracy when the bit-width for the representations getting larger. The binary representations have more robust performance compared with the baseline model when the dimension of embedding varied.

The node representations' binarization is more important than the graph representations' because the dot product operation is mainly conducted between the node representations which costs plenty of time. The performances of GMN and BGN-GMN are compared under different bit-width for the node embedding vectors by varying the dimensions.

As shown in Figure 6b, the result for the pair-wise AUC is similar between the binary and the real-valued node embedding vectors, but BGN-GMN holds a better performance with low bit-width representations. As for the triplet graph accuracy, the binary embedding vector achieves better performance with short code length and similar accuracy as real-valued node embedding with long code length. These results indicate that the binary representations are much better for the comparison between two graphs under low bit-width circumstances. In line with the result of the binary graph embedding vectors, the binary node embedding vectors also have more robust performance compared with the real-valued node representations.

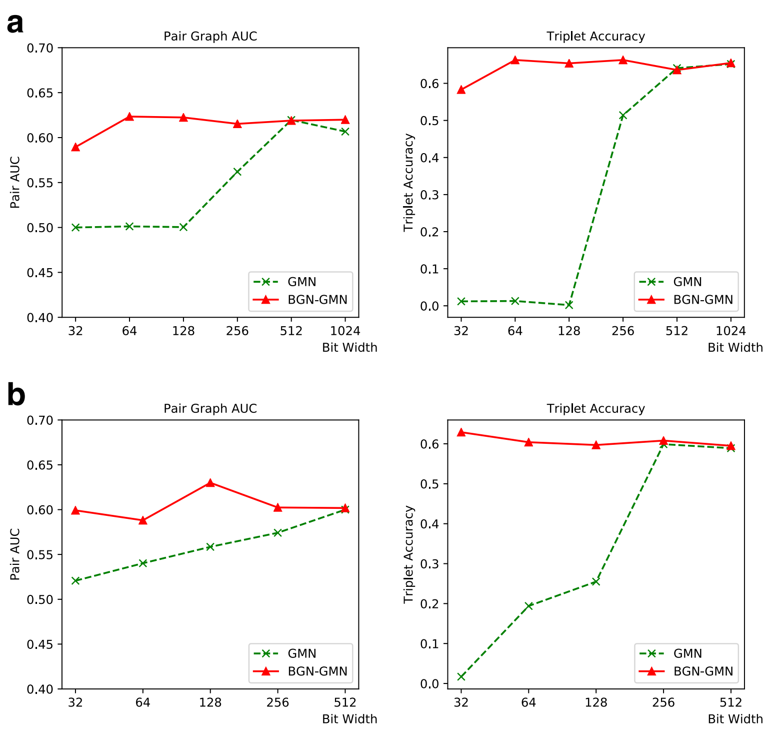

Figure 6 The performance comparison of graph matching task between original version of GMN and the BGN-GMN with **a** graph representations binarized and **b** node representations binarized

6 Conclusion

We present a model focused on the challenging problem of seeking binary representations of network embeddings using a compact neural network structure. We proposed a novel binarized graph embedding method, namely BGN, that has binarized parameters and enables GNNs to learn discrete embedding. The binarized neural network can reduce the memory and time cost of the GNN such that increases the scalability of GNNs. BGN can be naturally integrated into other GNN models to enhance the performance of the model such as graph matching network in terms of the inference time and space consumption. External experiment also illustrates that BGN can increase the time efficiency while holding competitive accuracy.

Acknowledgements Defu Lian is supported by grants from the National Natural Science Foundation of China (No. 61976198 and 62022077). Ying Zhang is supported by ARC FT170100128 and DP180103096. Lu Qin is supported by ARC FT200100787.

References

1. Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., Wang, W.: Simgnn: a neural network approach to fast graph similarity computation. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019, pp. 384–392 (2019)

2. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
3. Chen, G., He, S., Meng, H., Huang, K.: Phonebit: Efficient Gpu-Accelerated binary neural network inference engine for mobile phones. In: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 786–791. IEEE (2020)
4. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. Neural Information Processing Systems neurIPS (2016)
5. Cui, P., Wang, X., Pei, J., Zhu, W.: A survey on network embedding. IEEE Trans. Knowl. Data Eng. (TKDE) **31**(5), 833–852 (2019)
6. Erdös, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**(1), 17–60 (1960)
7. Fan, R., Chang, K., Hsieh, C., Wang, X., Lin, C.: LIBLINEAR: A library for large linear classification. JMLR **9**, 1871–1874 (2008)
8. Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., Yan, J.: Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 4852–4861 (2019)
9. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: ACM SIGKDD, pp. 855–864 (2016)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Neural Information Processing Systems NeurIPS, pp. 1024–1034 (2017)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull. **40**(3), 52–74 (2017)
12. Huang, W., Zhang, T., Rong, Y., Huang, J.: Adaptive sampling towards fast graph representation learning. In: Neural Information Processing Systems NeurIPS, pp. 4563–4572 (2018)
13. Hubara, I., Courbariaux, M., Soudry, D., El-yaniv, R., Bengio, Y.: Binarized neural networks. In: Neural Information Processing Systems NeurIPS, pp. 4107–4115 (2016)
14. Kazemi, S.M., Poole, D.: Simple embedding for link prediction in knowledge graphs. In: Neural Information Processing Systems NeurIPS, pp. 4289–4300 (2018)
15. Kipf, T.N., Welling, M.: Semi-Supervised classification with graph convolutional networks. In: ICLR (2017)
16. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection http://snap.stanford.edu/data (2014)
17. Li, Y., Gong, R., Yu, F., Dong, X., Liu, X.: Dms: Differentiable dimension search for binary neural networks
18. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, pp. 3835–3845 (2019)
19. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493 (2015)
20. Lian, D., Zheng, K., Zheng, V.W., Ge, Y., Cao, L., Tsang, I.W., Xie, X.: High-Order proximity preserving information network hashing. In: ACM SIGKDD, pp. 1744–1753 (2018)
21. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR, pp. 2064–2072 (2016)
22. Liu, W., Mu, C., Kumar, S., Chang, S.: Discrete graph hashing. In: Neural Information Processing Systems NeurIPS, pp. 3419–3427 (2014)
23. Liu, Z., Shen, Z., Savvides, M., Cheng, K.: Reactnet: Towards precise binary neural network with generalized activation functions. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J. (eds.) Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIV, Lecture Notes in Computer Science, vol. 12359, pp. 143–159. Springer (2020)
24. Martínez, B., Yang, J., Bulat, A., Tzimiropoulos, G.: Training binary neural networks with real-to-binary convolutions. In: 8Th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. Openreview.Net (2020)
25. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Neural Information Processing Systems NeurIPS, pp. 3111–3119 (2013)
26. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: ACM SIGKDD, pp. 701–710. ACM (2014)
27. Qin, H., Gong, R., Liu, X., Bai, X., Song, J., Sebe, N.: Binary neural networks: A survey. Pattern Recognition, pp. 107281 (2020)

28. Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, J.: Forward and backward information retention for accurate binary neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition CVPR, pp. 2250–2259 (2020)
29. Qin, J., Wang, Y., Xiao, C., Wang, W., Lin, X., Ishikawa, Y.: GPH: Similarity search in hamming space. In: IEEE ICDE, pp. 29–40 (2018)
30. Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., Tang, J.: Network embedding as matrix factorization: Unifying deepwalk, Line, Pte, and Node2vec. In: ACM WSDM, pp. 459–467 (2018)
31. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-Net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision, pp. 525–542. Springer (2016)
32. Ribeiro, L.F., Saverese, P.H., Figueiredo, D.R.: Struc2vec: Learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 385–394 (2017)
33. Salakhutdinov, R., Hinton, G.E.: Semantic hashing. Int. J. Approx. Reasoning **50**(7), 969–978 (2009)
34. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine **29**(3), 93–93 (2008)
35. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: CVPR, pp. 37–45 (2015)
36. Shen, M., Liu, X., Gong, R., Han, K.: Balanced binary neural networks with gated residual. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4197–4201. IEEE (2020)
37. Shen, X., Pan, S., Liu, W., Ong, Y., Sun, Q.: Discrete network embedding. In: IJCAI, pp. 3549–3555 (2018)
38. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: Large-scale information network embedding. In: WWW, pp. 1067–1077 (2015)
39. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Neural Information Processing Systems NeurIPS, pp. 5998–6008 (2017)
40. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
41. Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., Zhang, C.: Attributed graph clustering: a deep attentional embedding approach. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pp. 3670–3676 (2019)
42. Wang, H., Lian, D., Zhang, Y., Qin, L., Lin, X.: Gognn: Graph of graphs neural network for predicting structured entity interactions. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pp. 1317–1323. ijcai.org (2020)
43. Wang, J., Zhang, T., Song, J., Sebe, N., Shen, H.T.: A survey on learning to hash. IEEE Trans. Pattern Anal. Mach. Intell. TPAMI. **40**(4), 769–790 (2018)
44. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)
45. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful are Graph Neural Networks?. In: 7Th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019)
46. Yang, H., Pan, S., Zhang, P., Chen, L., Lian, D., Zhang, C.: Binarized Attributed Network Embedding. In: IEEE ICDM, pp. 1476–1481 (2018)
47. Zhang, J., Pan, Y., Yao, T., Zhao, H.: Mei, t.: dabnn: a super fast inference framework for binary neural networks on arm devices. In: Proceedings of the 27th ACM International Conference on Multimedia, pp. 2272–2275 (2019)
48. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: Neural Information Processing Systems NeurIPS, pp. 5171–5181 (2018)
49. Zhang, X., Liu, H., Li, Q., Wu, X.: Attributed graph clustering via adaptive graph convolution. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pp. 4327–4333 (2019)
50. Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR arXiv:abs/1606.06160 (2016)
51. Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., Yan, J.: Towards unified int8 training for convolutional neural network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition CVPR, pp. 1969–1979 (2020)

Affiliations

Hanchen Wang[1] · Defu Lian[2] · Ying Zhang[1] · Lu Qin[1] · Xiangjian He[3] · Yiguang Lin[3] · Xuemin Lin[4]

Hanchen Wang
hanchenw.au@gmail.com

Ying Zhang
Ying.Zhang@uts.edu.au

Lu Qin
Lu.Qin@uts.edu.au

Xiangjian He
Xiangjian.He@uts.edu.au

Yiguang Lin
Yiguang.Lin@uts.edu.au

Xuemin Lin
lxue@cse.unsw.edu.au

1 CAI, University of Technology Sydney, Sydney, Australia

2 University of Science and Technology of China, Anhui, China

3 University of Technology Sydney, Sydney, Australia

4 University of New South Wales, Sydney, Australia