Graph Compression Networks

Ting Guo*, Xingquan Zhu†, Yang Wang*, Fang Chen*
*Data Science Institute, University of Technology, Sydney, Australia
†Dept. of Electrical Engineering & Computer Science, Florida Atlantic University, Boca Raton, FL USA
{ting.guo, yang.wang, fang.chen}@uts.edu.au, xzhu3@fau.edu

2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671652

***Abstract*—Graphs/Networks are common in real-world applications where data have rich content and complex relationships. The increasing popularity also motivates many network learning algorithms, such as community detection, clustering, classification, and embedding learning, *etc.*. In reality, the large network volumes often hider a direct use of learning algorithms to the graphs. As a result, it is desirable to have the flexibility to condense a network to an arbitrary size, with well-preserved network topology and node content information. In this paper, we propose a graph compression network (GEN) to achieve network compression and embedding at the same time. Our theme is to leverage the network topology to find node mappings, such that densely connected nodes, including their node content, are compressed as a new node, with a latent vector (*i.e.* embedding) being learned to represent the compressed node. In addition to compression learning, we also develop a novel encoding-decoding framework, using feature diffusion process, to ”decompress” the condensed network. Different from traditional graph convolution which uses direct-neighbor message passing, our decompression advocates high-order message passing within compressed nodes to learning feature representation for all nodes in the network. A unique strength of GEN is that it leverages the graph neural network principle to learn mapping automatically, so one can compress a network to an arbitrary size, and also decompress it to the original node space with minimum information loss. Experiments and comparisons confirm that GEN can automatically find clusters and communities, and compress them as new nodes. Results also show that GEN achieves improved performance for numerous tasks, including graph classification and node clustering.**

***Index Terms*—Graph compression, graph neural networks, pooling, graph classification, node clustering**

I. Introduction

Graph-structured data appears in many fields, such as social networks, citation networks, knowledge graphs, telecommunication networks, and biological networks [1]. For these application domains, graphs commonly exist into two forms: (1) one single large network, consisting of many nodes and relationships, such as social networks or citation networks [2]; and (2) many small networks/graphs, each having a rather small number of nodes (*e.g.*, less than 100 nodes), such as chemical compounds [3]. For the former, the algorithms are commonly designed in a transductive learning setting [4]–[6], where nodes (including their connections) are observed and the learning objective is to cluster nodes into groups or classify nodes into predefined classes. For the latter, the algorithm designs are often based on an inductive learning setting [7]–[9], where some training graphs are given, and the objective is to classify or cluster previously unseen test graphs.

Recently, the emerging graph neural network based representation learning has motivated a graph pooling design for two purposes: (1) discovering coherent node groups in graphs, and (2) reduce the computational costs in large graphs [10], [11]. Such graph pooling mechanisms largely fall into two broad categories: *adaptive* and *topological*. The adaptive pooling, such as DIFFPOOL [12], relies on a trainable parametric pooling mechanism, where a parameterized neural layer is used to learn a clustering of the current nodes based on their embeddings at the previous layer. Such clustering is realized by means of a graph neural network (GNN) layer, followed by a softmax to obtain a soft-membership matrix associating nodes to clusters. However, similar to GNN, the clustered nodes only consider direct neighbors for embedding/assignment tasks without considering the high-order topological structure of the graph. Besides, DIFFPOOL is a task-dependent method that cannot deal with unsupervised tasks. Topological pooling, on the other hand, is non-adaptive and typically leverages the structure of graph itself as well as its communities. Note that, being non-adaptive, such mechanisms are not required to be differentiable [13], [14], and their results are not task-dependent. Hence, these methods are potentially reusable in multi-task scenarios [15]. However, if the graph has some nodes with a low degree, the spectral based clustering/partition methods may misclassify those nodes [16], [17].

The above challenges motivate our research to design graph compression networks (GEN) to assign nodes into soft clusters by using a novel two-layer neural network architecture, where the final output of this two-layer model is the trainable weights (a soft assignment matrix). The compression and embedding combined design allows GEN to be applied in a hierarchical and end-to-end fashion for graph representation learning. In addition to compression learning, we also develop a decoder network to reconstruct the original graph by using a feature diffusion process within communities/clusters learned during compression. By combining direct-neighbor and high-order message passing embedings, GEN achieves better performance for node representation learning, node clustering, and graph classification. The novelty of the proposed work, compared to existing research in the field, is threefold:

- **Learning to Compression:** We propose a unsupervised two-layer neural network to learn a soft assignment matrix, based on network topological structure, to compress a network to any sizes, as shown in Fig. 1.

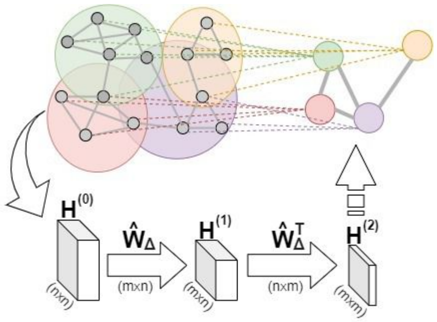

Fig. 1. The proposed GEN with soft assignment, where densely connected nodes are compressed into a new node. A two-layer neural network (lower panel) is developed to learn the assignment matrix $\hat{W}_{\Delta}$. A stacked multi-layer GEN can condense a network to arbitrary sized network hierarchies.

- **Compressed Network Embedding:** The proposed compressing module can be adapted to GNN architectures in an end-to-end fashion, allowing for developing deeper GNN models that can learn to operate on hierarchical representations of a graph.
- **Decompressed Network Embedding:** We also propose a GEN-based encoder-decoder architecture to use community-level feature-aggregation for accurate node representation learning.

II. Problem Definition & Preliminary

A. Problem Definition

An undirected connected attributed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{X}\}$ consists of a set of nodes $\mathcal{V}$ with $|\mathcal{V}| = n$ nodes, a set of edges $\mathcal{E}$ with $|\mathcal{E}| = m$, the adjacency matrix $\mathcal{A}$, and node attribute matrix $\mathcal{X}$. If there is an edge between node i and node j, the entry $\mathcal{A}_{ij}$ denotes the weight of the edge; otherwise, $\mathcal{A}_{ij} = 0$. For unweighted graphs, we simply set $\mathcal{A}_{ij} = 1$.

For each node, its content (feature) is represented as a vector $\mathbf{x} \in \mathbb{R}^n$, where $\mathbf{x}_i$ denotes feature values of node i (Node attributes, node content, and node features are equivalent terms in this paper). Therefore, $\mathcal{X} \in \mathbb{R}^{n \times d}$ denotes the node attribute matrix of the graph, and the columns of $\mathcal{X}$ are the d features of the graph.

Given a graph $\mathcal{G}$, graph compression networks **aim** to create a compressed graph $\mathcal{G}'$ with a fewer number of nodes than $\mathcal{G}$, such that $\mathcal{G}'$ not only preserves topology and content information of $\mathcal{G}$, but also preserves performance for common graph learning tasks, such as node clustering, graph classification, etc.

B. Graph Neural Networks

In this work, we build upon graph neural networks (GNNs) in order to learn both node-level and graph-level representation in an end-to-end fashion. In particular, we consider GNNs that employ the general ”message-passing” architectures:

Graph Convolutional Networks: Graph Convolutional Networks (GCNs) [4] are deep neural networks that achieve promising generalization in various tasks. At layer i, taking graph adjacency matrix $\mathcal{A}$ and hidden representation matrix $H^{(i)}$ as input, each GCN module outputs a hidden representation matrix $H^{(i+1)}$, which is described as:

$$H^{(i+1)} = ReLU(\hat{D}^{-\frac{1}{2}}\hat{\mathcal{A}}\hat{D}^{-\frac{1}{2}}H^{(i)}W^{(i)}) \tag{1}$$

where $H^{(0)} = \mathcal{X}$, $ReLU(a) = max(0, a)$, adjacency matrix with self-loop $\hat{\mathcal{A}} = \mathcal{A} + I$ (I is an identity matrix), $\hat{D}$ is the degree matrix of $\hat{\mathcal{A}}$, and $W^{(i)}$ is a trainable weight matrix. Then the output node embedding $\mathcal{Z} = H^{(K)}$ and $(K + 1)$ is the number of layers in the network architecture.

Graph Auto-Encoders: Graph auto-encoders (GAEs) [18]–[20] aim at mapping (encoding) each node to a vector from which reconstructing (decoding) the graph should be possible. More precisely, the node embedding matrix $\mathcal{Z}$ is usually the output of a graph neural network (GNN) [21], [22] processing $\mathcal{A}$. To reconstruct the graph, GAE stacks an inner product decoder to this GNN, resulting in $\bar{\mathcal{A}} = \sigma(\mathcal{Z}\mathcal{Z}^{\top})$, with $\sigma(\cdot)$ denoting the sigmoid function: $\sigma(x) = 1/(1+e^{-x})$. Therefore, the larger the inner product $\bar{\mathcal{A}}_{ij}$ in the embedding, the more likely nodes i and j are connected in $\mathcal{G}$ according to the GAE. Weights of the GNN are trained by gradient descent to minimize a reconstruction loss capturing the similarity of $\mathcal{A}$ and $\bar{\mathcal{A}}$, usually formulated as a weighted cross entropy loss.

Variational graph autoencoder [18] (VGAE) extends the variational autoencoder (VAE) [23] to graphs, by designing a probabilistic model involving latent variables $\mathbf{z}_i$ for each node $i \in \mathcal{V}$, interpreted as node representations in an embedding space. The inference model, *i.e.* the encoding part of the VAE, is defined as:

$$q(\mathcal{Z}|\mathcal{X}, \mathcal{A}) = \prod_{i=1}^{n} q(\mathbf{z}_i|\mathcal{X}, \mathcal{A}) \tag{2}$$

where $q(\mathbf{z}_i|\mathcal{X}, \mathcal{A}) = \mathcal{N}(\mathbf{z}_i|\mu_i, diag(\sigma_i^2))$. Gaussian parameters are learned from two GNNs, *i.e.* $\mu = GNN_{\mu}(\mathcal{X}, \mathcal{A})$, with μ the matrix stacking up mean vectors μ_i; likewise, $log\sigma = GNN_{\sigma}(\mathcal{X}, \mathcal{A})$. Latent vectors $\mathbf{z}_i$ are samples drawn from this distribution. From these vectors, a generative model aims at reconstructing (decoding) $\mathcal{A}$, leveraging inner products: $p(\mathcal{A}|\mathcal{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(\mathcal{A}_{ij}|\mathbf{z}_i, \mathbf{z}_j)$, where $p(\mathcal{A}_{ij} = 1|\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T\mathbf{z}_j)$. During training, GNN weights are tuned by maximizing a tractable variational lower bound (ELBO) of the model's likelihood by gradient descent, with a Gaussian prior on the distribution of latent vectors, and using the reparameterization trick from [23]. Formally, for GAE, we minimize the reconstruction error of the graph data by:

$$\mathcal{L}_R(\mathcal{Z}) = \mathbb{E}_{q(Z|\mathcal{X},\mathcal{A})}[\log p(\mathcal{A}|\mathcal{Z})] \tag{3}$$

while for VGAE, the loss function is:

$$\mathcal{L}_R(\mathcal{Z}) = \mathbb{E}_{q(Z|\mathcal{X},\mathcal{A})}[\log p(\mathcal{A}|\mathcal{Z})] - KL[q(\mathcal{Z}|\mathcal{X}, \mathcal{A})||p(\mathcal{Z})] \tag{4}$$

III. GEN: Graph Compression Networks

In the following, we first describe the criterion to evaluate the node compression quality, then we discuss how to learn a soft assignment matrix using a neural network architecture.

Graph-compression assignment matrix: Give a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{X}\}$, we denote the learned cluster/community assignment matrix as $\mathcal{M} \in \mathbb{R}^{m \times n}$, where $m \ll n$. Each column of $\mathcal{M}$ corresponds to one of the nodes and each row of $\mathcal{M}$ corresponds to one of the clusters ($\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_m\}$). When each node can only be assigned to one cluster (*i.e.* $\mathcal{M}_{ij} \in 0, 1$ and $\sum_{i=1}^{m} \mathcal{M}_{ij} = 1$), we call $\mathcal{M}$ a *hard assignment matrix*. While if each node can be partially assigned to multiple clusters (*i.e.* $0 \leq \mathcal{M}_{ij} \leq 1$ and $\sum_{i=1}^{m} \mathcal{M}_{ij} = 1$), we call it a *soft assignment matrix*.

Ideally, optimal clusters/communities should ensure that nodes in the same class are close to each other, *i.e.*, nodes in the same class should share more edges. Meanwhile, because an increasing of nodes results in the increase of edges in a cluster, we should consider the number of average edges (*cluster inter-connectivity*) per node for each cluster. Based on this property, we derive an evaluation criterion $\mathcal{J}_{\mathcal{C}}(m)$ for a given graph $\mathcal{G}$ as follows:

$$\mathcal{J}_{\mathcal{C}}(m) = \sum_{k=1}^{m} \frac{1}{|\mathcal{C}_k|} \Big(\sum_{\mathcal{V}_i, \mathcal{V}_j \in \mathcal{C}_k} \mathcal{A}_{ij} \Big) = tr(\mathcal{D}_{\mathcal{M}}^{-1} \mathcal{M} \mathcal{A} \mathcal{M}^\top) \tag{5}$$

where $\mathcal{D}_{\mathcal{M}} \in \mathbb{R}^{m \times m}$ is the diagonal row sum matrix of $\mathcal{M}$, $tr(\cdot)$ is the trace of a matrix and $|\cdot|$ is the degree of nodes in a given set[1]. We can easily observe that $[\mathcal{D}_{\mathcal{M}}^{-1} \mathcal{M} \mathcal{A} \mathcal{M}^\top]_{ii}$ shows the cluster inter-connectivity for cluster $\mathcal{C}_i$. And we can also find that $[\mathcal{M} \mathcal{A} \mathcal{M}^\top]_{ij}$ represents the intra-connectivity between two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ ($i \neq j$). So given an assignment matrix $\mathcal{M}$ for graph $\mathcal{G}$, we obtain not only the compressed graph $\mathcal{G}'$, but also the normalized compressed adjacency matrix:

$$\mathcal{A}'_{ij} = \begin{cases} 0 & if \quad i = j \\ [\mathcal{D}_{\mathcal{M}}^{-\frac{1}{2}} \mathcal{M} \mathcal{A} \mathcal{M}^\top \mathcal{D}_{\mathcal{M}}^{-\frac{1}{2}}]_{ij} & if \quad i \neq j \end{cases} \tag{6}$$

Accordingly, our goal is to find an optimal soft/hard assignment matrix with m clusters that can maximize $\mathcal{J}_{\mathcal{C}}(m)$ as:

$$\mathcal{M}^* = \arg\max_{\mathcal{M}} tr(\mathcal{D}_{\mathcal{M}}^{-1} \mathcal{M} \mathcal{A} \mathcal{M}^\top) \tag{7}$$

Learning the assignment matrix: To solve Eq. (7), we develop a two-layer neural network to learn the assignment matrix $\mathcal{M}^*$. The two layer neural network shares a weight matrix $\hat{W}_\Delta \in \mathbb{R}^{m \times n}$, whose role is to determine the optimal assignment of a node from its input space (with n nodes) to a compressed network (with m nodes). Overall, the two layer network is as follows:

$$\begin{aligned} &H^{(0)} = \mathcal{A}, \hat{W}_{\Delta \cdot i} = Softmax(W_{\Delta \cdot i}), H^{(1)} = \hat{W}_\Delta H^{(0)}, \\ &H^{(2)} = \mathcal{D}_{\hat{W}_\Delta}^{-1} (H^{(1)} \hat{W}_\Delta^\top), \mathcal{M}^* = \hat{W}_\Delta^* \end{aligned} \tag{8}$$

where $\mathcal{D}_{\hat{W}_\Delta} \in \mathbb{R}^{m \times m}$ is the diagonal row sum matrix of $\hat{W}_\Delta$. The layers $H^{(1)} \in \mathbb{R}^{m \times n}$ and $H^{(2)} \in \mathbb{R}^{m \times m}$ share the same trainable weight matrix W_Δ. Different from traditional neural network, our final output (the optimal assignment matrix $\mathcal{M}^*$) is the optimal weight matrix $\hat{W}_\Delta$. Because the i^{th} column of $\hat{W}_\Delta$ determines the assignment of the i^{th} node, which should follows a distribution with respect to the compressed node space, we force the values in W_Δ to be positive and $\sum_{i=1}^{m} W_{\Delta ij} = 1$ by applying softmax function on each column of W_Δ. As a result, we optimize the following loss function until convergence:

$$\mathcal{L}_0 = -tr(H^{(2)}) \tag{9}$$

Note that the output $\mathcal{M}^*$ is a soft assignment matrix. To obtain the corresponding hard assignment matrix $\mathcal{M}_h^*$, we just assign a node to the cluster with highest assignment possibility. Specifically, $[\mathcal{M}_h^*]_{ij} = 1$ if $[\mathcal{M}^*]_{ij} = MAX([\mathcal{M}^*]_{\cdot j})$ and $[\mathcal{M}_h^*]_{ij} = 0$, otherwise.

Graph Compression Layer: Suppose that the soft assignment matrix $\mathcal{M}$ has already been computed. The next task is to compress the nodes in the same clusters as a new node. We denote the original graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{X}\}$ and denote the output compressed graph as $\mathcal{G}' = \{\mathcal{V}', \mathcal{E}', \mathcal{A}', \mathcal{X}'\}$. Given these inputs, the GEN layer follows the layer-wise compression rule:

$$\mathcal{X}' = \sigma(D_{\mathcal{M}}^{-1} \mathcal{M} \mathcal{X} W) \tag{10}$$

In order to avoid changing the scale of the feature vectors, we normalize $\mathcal{M}$ by multiplying it with $D_{\mathcal{M}}^{-1} \mathcal{M}$, which corresponds to taking the average of neighboring node features.

By the introduction of a trainable weight matrix W in Eq. (10), GEN can achieve structural compression and attribute embedding at the same time. It is worth noting that W in Eq. (10) and W_Δ in Eq. (8) play different roles. W is the trainable parameters aiming to optimize the output $\mathcal{X}'$, whereas W_Δ is not only the trainable parameters for minimizing loss function $\mathcal{L}_0$ but also the output itself ($\hat{W}_\Delta$). The adjacency matrix $\mathcal{A}'$ for the compressed graph $\mathcal{G}'$ can be generated by applying Eq. (6).

IV. GEN FOR GRAPH REPRESENTATION LEARNING

The goal of graph representation learning is to use a compact feature vector to represent the entire graph. Our theme is to use GCN to learn node embedding, and then use GEN to compress network as a vector, as shown in Fig. 2.

The first step of our model is to apply L_1 GCN layers to learn node embedding (Eq. (1)), so node representation can indirectly capture topological information within its local neighborhood. After that, we adapt stacked L_2 GEN layers to smoothly map the original graph to a denser one and eventually as a single node (a compact feature vector). Specifically, we denote the input assignment matrix at layer l as $\mathcal{M}^{(l)}$ and denote the input node embedding matrix at this layer as $Z^{(l)}$. As the adjacency matrix $\mathcal{A}^{(l)}$ can be learned from $\mathcal{M}^{(l-1)}$ and $\mathcal{A}^{(l-1)}$, the assignment matrix for different GEN layers can be recursively generated one after another.

$$\begin{aligned} &\mathcal{M}^{(l)*} = \arg\max_{\mathcal{M}^{(l)}} tr(\mathcal{D}_{\mathcal{M}^{(l)}}^{-1} \mathcal{M}^{(l)} \mathcal{A}^{(l-1)} \mathcal{M}^{(l)\top}), \\ &\mathcal{A}_{ij}^{(l)} = \begin{cases} 0 & if \quad i = j \\ [\mathcal{D}_{\mathcal{M}^{(l)}}^{-\frac{1}{2}} \mathcal{M}^{(l)} \mathcal{A}^{(l-1)} \mathcal{M}^{(l)\top} \mathcal{D}_{\mathcal{M}^{(l)}}^{-\frac{1}{2}}]_{ij} & if \quad i \neq j \end{cases} \end{aligned} \tag{11}$$

[1] For hard assignment, the degree here means the number of nodes in a given set, while for soft assignment, the degree represent the sum of all portions of nodes that have been assigned to the given set

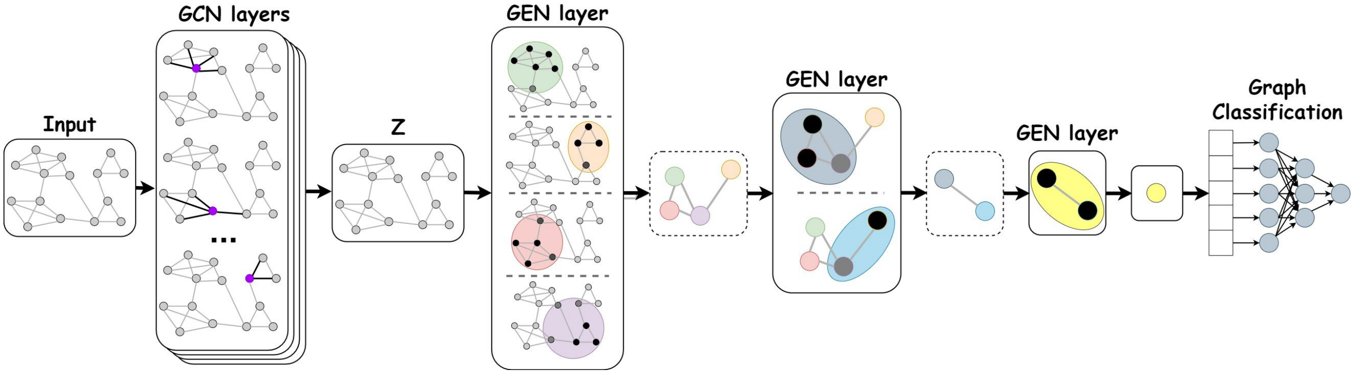

Fig. 2. The architecture of GEN-based graph representation learning for inductive graph classification. Given an input network, GEN first applies L_1 GCN layers for learning node embedding from local neighbors. After that, GEN learns assignment matrix to compress nodes, followed by one or more GEN layers on the compressed graph. This process is repeated for L_2 layers and the final output representation is used to classify the graph.

Given these inputs, the GEN layer follows the layer-wise compression rule:

$$Z^{(l+1)} = \sigma(D^{-1}_{\mathcal{M}^{(l)}} \mathcal{M}^{(l)} Z^{(l)} W^{(l)}) \tag{12}$$

For graph classification, all nodes at the final GEN layer L_2 are assigned to a single node, generating a final embedding vector corresponding to the entire graph. This final output embedding can then be used as feature input to a differentiable classifier (*e.g.*, a softmax layer), and the entire system can be trained end-to-end using stochastic gradient descent.

Permutation invariance: Note that in order to be useful for graph classification, the GEN compression layer should be invariant under node permutations. Let $P \in \{0,1\}^{n \times n}$ be any permutation matrix, then the deformation of the input graph $\mathcal{G}$ is $\mathcal{G}^P$ where $\mathcal{A}^P = P\mathcal{A}P^\top$ and $\mathcal{X}^P = P\mathcal{X}$. Based on Eqs.(7) and (8), the deformation of the optimal assignment matrix is $\mathcal{M}^P = \mathcal{M}P^\top$. And since permutation matrix is orthogonal, applying $P^\top = I$ to Eq. (10), we have $\mathcal{M}^P \mathcal{X}^P W = \mathcal{M}P^\top P\mathcal{X}W = \mathcal{M}\mathcal{X}W$. It proves that the GEN layer is invariant as the normalization factor $D^{-1}_{\mathcal{M}}$ does not affect the permutation invariance.

V. GEN for Node Representation Learning

Similar to traditional GCNs, GENs approximate smooth filters that can extract local features independently of the graph size. But different from GCNs, which is a first-order approximation of ChebNet [4], our GENs is a higher-order convolutional method that aggregates feature information from a learned cluster/community. We assume that the nodes in the same cluster should be strongly correlated. As our compression method is based upon a soft assignment, the nodes that near the cluster boundaries tend to be assigned to multiple clusters and will pass feature information crossing clusters, which better fit the real situation. Therefore, GENs can be considered as a high-order message passing process if we can restore the original nodes with the compressed nodes' representation. In this section, we develop a GEN-based decoder layer that can restore the original graph from the compressed one.

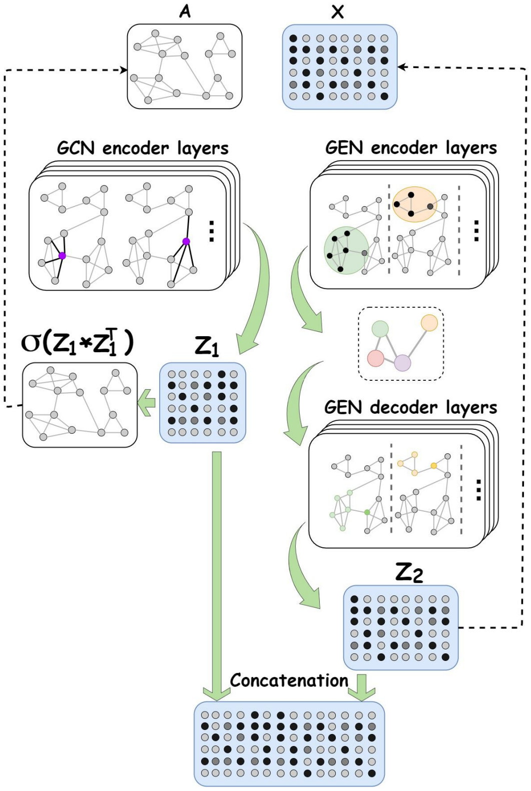

Fig. 3. GEN-based node representation learning.

Hierarchical GEN decoder layers: Given a compressed graph $\mathcal{G}$ and its corresponding assignment matrix $\mathcal{M}$, the GEN decoder is as follow:

$$\mathcal{X}' = \sigma(\mathcal{M}^\top \mathcal{X}) \tag{13}$$

And in a stacked layer-wise form:

$$Z^{(l+1)} = \sigma(\mathcal{M}^{(l)\top} Z^{(l)}) \tag{14}$$

Integrate with graph compression Layers (GEN encoder layers), the overall framework of GEN encoder-decoder networks are shown in Fig. 3 (the right panel). As mentioned before,

TABLE I
BENCHMARK NETWORK STATISTICS

Dataset	*# graphs*	*# Nodes*	*# Edges*	*# Features*	*# Classes*
Cora	1	2,708	5,429	1,433	7
Citeseer	1	3,327	4,732	3,703	6
Pubmed	1	19,717	44,338	500	3
MUTAG	188	17.9	19.8	7	2
PTC	344	25.5	26	19	2
PROTEINS	1,113	39.1	72.8	3	2

TABLE II
THE PERFORMANCE OF PARTITION RESULTS ON FOOTBALL, CORA AND CITESEER. NUMBERS IN PARENTHESIS DENOTE NUMBERS OF CLUSTERS.

	Performance of partition		
Methods	**Football** (6)	**Cora** (105)	**Citeseer** (488)
AsynFluid	0.8628	-	-
GreedyMod	0.8682	0.9271	0.9692
GEN	**0.8781**	**0.9877**	**0.9969**

the GEN encoder-decoder structure is a high-order smoothing filter acting on bigger local areas comparing with GCN-based methods. The size of the local area depends on the scale of the compression defined by the size of the assignment matrix. We will show the impact of the compression scale in the experimental section.

GAE+GEN auto-encoder model: The model contains two parts as shown in Fig. 3. The left panel is a GAE/VGAE model with GCN convolutional encoder layers to learn the first-order approximation by reducing the reconstruction loss of $\mathcal{A}$ [18]. The right panel is a GEN encoder-decoder model to learn high-order approximation by reducing the reconstruction loss of $\mathcal{X}$:

$$\mathcal{L}_{\mathcal{G}}(\mathcal{Z}) = MSE(\mathcal{Z}, \mathcal{X}) \tag{15}$$

Different from GAE/VGAE, the output of GEN encoder-decoder model is not the compressed intermediate result but the output of the final GEN decoder layer. The final node embedding Z is the concatenation of Z_1 from the GAE/VGAE and Z_2 from the GEN encoder-decoder. For supervised node representation learning tasks, the GAE/VGAE+GEN model can be considered as a block. With stacking multiple GAE+GEN blocks, the graph neural networks can go deeper and capture even higher-order topological information within and without its local neighborhood.

VI. EXPERIMENTS

A. Benchmark Networks

We use seven benchmark networks, as shown in Table I, including three networks for graph classification and three graph datasets for node clustering.

B. Network Compression Results

We investigate the extent to which GEN learns meaningful node clusters/communities ($\mathcal{M}$) by comparing it with the state-of-the-arts community detection methods and visualizing the cluster assignments as community detection. Note that, for visualization purpose, we generate hard assignment matrix from soft assignment matrix.

Baselines: We compare our algorithm against following baselines: *AsynFluid* [24]: the asynchronous fluids communities algorithm is based on the simple idea of fluids interacting in an environment, expanding and pushing each other. *GreedyMod* [25]: it finds communities in graph using Clauset-Newman-Moore greedy modularity maximization.

Metrics. The performance of a partition [16] is the ratio of the number of intra-community edges plus inter-community non-edges with the total number of potential edges. The number of clusters is set as the results of GreedyMod, which provide the partition number and partition sets at the same time. As the initialization for the community detection methods is random, we conduct each experiment 10 times and report the mean values as the final scores.

Experimental Results. We observe significant improvement in membership assignment quality with community detection objectives as shown in Table II. As the asynchronous fluids communities algorithm requires connected graphs, it cannot deal with sparse graphs like Cora and Citeseer. Comparing with GreedyMod, GEN-based node assignment method can achieve more than 6% increased performance on Cora and more than 3% on Citeseer, which means GEN performs even better on sparse graphs. What is more, different from GreedyMod, GEN can predefine the number of clusters, which makes it suitable for further graph compression with different sizes.

C. Graph Classification Results

Baselines. We use three public graph classification datasets to evaluate GEN, and compare its performance with five baselines. **DCNN** [26] adopts a diffusion-convolution operation to learn a latent representation for graphical data. **GraphSage** [8] with global mean pooling has enabled an inductive capability for inferring unseen nodes or graphs by aggregating subsampled local neighborhoods and by learning in a mini-batch gradient descent fashion. **GCN** [4] is the implementation of GCN with global mean pooling approach for graph classification tasks. **DIFFPOOL** [12] adopts the end-to-end training architectures for supervised graph classification with differentiable pooling.

Metrics. We employ 10-fold cross-validation accuracy (Acc) as the metrics to validate the graph classification results.

Experimental Results. The classification results on three benchmark datasets are reported in Table IV. The results show that GEN outperforms all other baselines, including graph pooling based approaches, for graph representation based classification.

D. Node Clustering Results

We investigate the extent to which GAE/VGAE+GEN framework learns meaningful node representation comparing to other node embedding methods. For node clustering task,

TABLE III
NODE CLUSTERING RESULT COMPARISONS ON CORA, CITESEER, AND PUBMED NETWORKS.

	CORA			CITESEER			PUBMED		
Methods	**Acc**	**NMI**	**ARI**	**Acc**	**NMI**	**ARI**	**Acc**	**NMI**	**ARI**
k-means	0.493	0.337	0.241	0.389	0.173	0.106	0.573	0.291	0.221
Spectral Clustering	0.419	0.198	0.076	0.462	0.212	0.179	0.598	0.313	0.296
DeepWalk	0.513	0.378	0.269	0.362	0.097	0.032	0.601	0.168	0.103
GAE	0.657	0.431	0.306	0.574	0.227	0.139	0.562	0.230	0.181
VGAE	0.670	0.412	0.340	0.553	0.184	0.123	0.541	0.251	0.212
ARVGA	0.672	0.427	0.305	0.544	0.260	0.211	0.597	0.238	0.197
GAE+GEN	0.690	0.459	0.372	**0.616**	**0.382**	**0.243**	**0.623**	0.304	0.289
VGAE+GEN	**0.698**	**0.473**	**0.384**	0.610	0.327	0.204	0.618	**0.335**	**0.312**

TABLE IV
GRAPH CLASSIFICATION ACCURACY ON MUTAG, PTC, AND PROTEINS.

	10-fold Cross-validation Accuracy		
Methods	**MUTAG**	**PTC**	**PROTEINS**
DCNN	0.672	0.568	0.579
GraphSage	0.656	0.372	0.703
GCN	0.723	0.538	0.721
DIFFPOOL	0.741	0.556	0.733
GEN	**0.764**	**0.576**	**0.754**

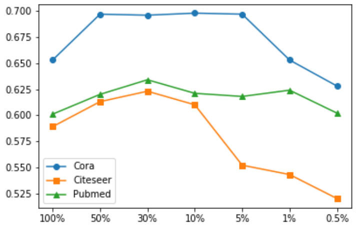

Fig. 4. Node clustering results with respect to different GEN compression ratios for testing VGAE+GAE framework. An $\alpha\%$ (x-axis) value means the number of clusters being set as $\alpha\%$ of the number of nodes of the input network.

we first learn node embedding, and then perform k-means clustering using the node embedding results.

Baselines. We compare our algorithm against following baselines: **k-means** is a classical method and also the foundation of many clustering algorithms. **Spectral Clustering**: [27] is an effective approach for learning social embedding. **DeepWalk**: [28] is a network representation approach which encodes social relations into a continuous vector space. **GAE/VGAE**: [18] are (variational) autoencoder-based unsupervised frameworks for graph data, which naturally leverages both topological and content information. And **ARVGA**: [29] is an adversarially regularized variational graph autoencoder for learning the node embedding.

Metrics. We employ three metrics to validate the clustering results: Clustering accuracy (Acc), Normalized Mutual Information (NMI), and Average Rand index (ARI).

Experimental Results. The node clustering results in Table III show that by incorporating the high-order weighted feature aggregation, GAE/VGAE+GEN achieves best performance on all three metrics. For instance, on Cora, VGAE+GEN has increased the accuracy from 3.8% compared with ARVGA to 40% compared with Spectral Clustering.

We have also checked the performance of node clustering with different GEN compression ratio to test our VGAE+GEN framework as shown in Fig. 4, where 100% means no graph compression ($Z_2 = \mathcal{X}$) before the concatenation of the two node embedding (Z_1 and Z_2). Fig. 4 shows that with 50% compression ratio, the clustering performance increase for both Cora and Citeseer. For Cora, the performance remains stable when the compression ratio changes from 50% to 5%. This is because the generated assignment matrix tends to generate similar clusters, meaning that there are some null-clusters when the compression ratio is bigger than 5%. When the compression ratio is smaller than 5%, the performance dropped significantly, implies information loss in feature aggregation. Similar phenomenon can be found for the Citeseer dataset, but the inflection point is at 10%. This illustrates that the best partition is unique for each network.

VII. CONCLUSION

In this paper, we proposed a graph compression network (GEN) to achieve network compression and embedding at the same time. Our theme is to learn a two-layer neural network to find soft node cluster assignment, and further use it to guide network compression process to learn representation for a single node or the whole graph. GEN is designed to learn network compression, and to decompress a compressed network to the original node space. It can also be stacked to form a multi-layer GEN with hierarchical compression networks. Experiments and comparisons show that GEN can find coherent structures in networks for effective compression. The compression also helps GEN deliver better graph representation and node representation for both transductive and inductive graph learning tasks.

ACKNOWLEDGMENT

This research is supported by the U.S. National Science Foundation (NSF) through Grant Nos. IIS-1763452, CNS-1828181, and IIS-2027339.

References

[1] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.

[2] S. Dawson, D. Gašević, G. Siemens, and S. Joksimovic, "Current state and future trends: A citation network analysis of the learning analytics field," in *Proceedings of the fourth international conference on learning analytics and knowledge*, 2014, pp. 231–240.

[3] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2016.

[5] S. Nandanwar and M. N. Murty, "Structural neighborhood based classification of nodes in a network," in *ACM SIGKDD*, 2016, pp. 1085–1094.

[6] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 889–898.

[7] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. of ACM SIGKDD*, 2018, pp. 1666–1674.

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[9] R. A. Rossi, R. Zhou, and N. Ahmed, "Deep inductive graph representation learning," *IEEE TKDE*, 2018.

[10] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *arXiv:1912.12693*, 2019.

[11] I. Spinelli, S. Scardapane, and A. Uncini, "Adaptive propagation graph convolutional network," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[12] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, 2018, pp. 4800–4810.

[13] J. Wu, J. He, and J. Xu, "Demo-net: Degree-specific graph neural networks for node and graph classification," in *ACM SIGKDD*, 2019, pp. 406–415.

[14] S. Jin, W. Liu, E. Xie, W. Wang, C. Qian, W. Ouyang, and P. Luo, "Differentiable hierarchical graph grouping for multi-person pose estimation," in *European Conference on Computer Vision*. Springer, 2020, pp. 718–734.

[15] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE PAMI*, vol. 29, no. 11, pp. 1944–1957, 2007.

[16] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[17] M. Li, R.-R. Liu, L. Lü, M.-B. Hu, S. Xu, and Y.-C. Zhang, "Percolation on complex networks: Theory and application," *Physics Reports*, 2021.

[18] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv:1611.07308*, 2016.

[19] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *AAAI*, 2014.

[20] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[21] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv:1312.6203*, 2013.

[22] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114*, 2013.

[24] F. Parés, D. G. Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "Fluid communities: a competitive, scalable and diverse community detection algorithm," in *Intl. Conf. on Complex Networks and their Applications*. Springer, 2017, pp. 229–240.

[25] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[26] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in neural information processing systems*, 2016, pp. 1993–2001.

[27] L. Tang and H. Liu, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery*, vol. 23, no. 3, pp. 447–478, 2011.

[28] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[29] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," *IJCAI*, 2018.