Unsupervised Graph Embedding via Adaptive Graph Learning

Rui Zhang, *Member, IEEE*, Yunxing Zhang, Xuelong Li*, *Fellow, IEEE*

Abstract—Graph autoencoders (GAEs) are powerful tools in representation learning for graph embedding. However, the performance of GAEs is very dependent on the quality of the graph structure, i.e., of the adjacency matrix. In other words, GAEs would perform poorly when the adjacency matrix is incomplete or be disturbed. In this paper, two novel unsupervised graph embedding methods, *unsupervised graph embedding via adaptive graph learning* (BAGE) and *unsupervised graph embedding via variational adaptive graph learning* (VBAGE) are proposed. The proposed methods expand the application range of GAEs on graph embedding, i.e, on the general datasets without graph structure. Meanwhile, the adaptive learning mechanism can initialize the adjacency matrix without be affected by the parameter. Besides that, the latent representations are embedded in the laplacian graph structure to preserve the topology structure of the graph in the vector space. Moreover, the adjacency matrix can be self-learned for better embedding performance when the original graph structure is incomplete. With adaptive learning, the proposed method is much more robust to the graph structure. Experimental studies on several datasets validate our design and demonstrate that our methods outperform baselines by a wide margin in node clustering, node classification, and graph visualization tasks.

Index Terms—Graph Embedding, Adaptive Graph Learning, Graph Autoencoder

1 Introduction

Graphs are powerful tools to seek the geometric structure of data and graph analysis has been attracting increasing attention in recent years due to the ubiquity of networks in the real world. There are various applications using graphs in machine learning and data mining fields such as node classification [1] [2], node clustering [3] [4], link prediction [5], and visualization [6]. However, it is difficult to directly apply the existing machine learning methods to graph data, due to the high computational complexity, low parallelizability, and inapplicability of most methods to graph data [7].

In response to these problems, many graph embedding methods have been proposed in recent years. As one of the representation learning, the purpose of graph embedding is to learn the low-dimensional feature vectors which should preserve the topology structure of the graph. In the early 2000s, researchers developed graph embedding algorithms as part of dimensionality reduction techniques. The early graph embedding methods such as Laplacian Eigenmaps [8] and Locally Linear Embedding (LLE) [9] map the nodes of the graph into a low-dimensional vector space. Since 2010, research on graph embedding has shifted to obtaining scalable graph embedding techniques that leverage the sparsity of real-world networks [10]. The methods at this stage such as Graph Factorization [11], LINE [12], HOPE [13], and SDNE [14] attempt to preserve both first order and second proximities.

In recent years, graph neural networks (GNNs) [15] emerge as powerful node embedding methods with successful applications in broad areas such as social networks, recommended systems, and natural language processing. Graph neural networks (GNNs) are powerful tools in representation learning for graphs and able to incorporate sparse and discrete dependency structures between data points. Graph neural networks (GNNs) could be roughly categorized into four categories: recurrent graph neural networks (RecGNNs), convolutional graph neural networks (ConvGNNs), graph autoencoders (GAEs), and spatial-temporal graph neural networks (STGNNs) [16].

Graph autoencoders (GAEs) are the effective unsupervised learning frameworks that encode the node features and graph construction into the latent representations and decode the graph construction. GAEs and most of their extensions rely on graph convolutional networks (GCN) to learn vector space representations of nodes. GAEs can be used to learn graph (network) embeddings [17] and graph generative distributions [18]. For graph embedding, GAEs mainly learn the latent representations by reconstructing the graph construction, e.g., the adjacency matrix. For graph generation, GAEs can learn the generative distribution of graphs and are mostly designed to solve the molecular graph generation problem [18]. The main distinction between GAEs and graph embedding is that GAEs are designed for various tasks while graph embedding covers various kinds of methods targeting the same task.

Although GNNs achieve great success from in representation learning of graphs, recent studies show that the performance of GNNs is very dependent on the quality of the adjacency matrix. In other words, GNNs will perform poorly while the adjacency matrix is under attack or incomplete. The incomplete means the adjacency matrix is partially missing or be disturbed. To defend against adversarial attacks, Jin et al. [19] propose a framework Pro-GNN, which can jointly learn a structural graph and a robust graph

Xuelong Li is the corresponding author.*
Rui Zhang, Yunxing Zhang, and Xuelong Li are with the School of Computer Science and Center for OPTical IMagery Analysis and Learning (OPTIMAL), Northwestern Polytechnical University, Xi'an 710072, Shaanxi, P. R. China.
E-mail: ruizhang8633@gmail.com, zhangyunxing423@outlook.com, and xuelong_li@nwpu.edu.cn

neural network model from the perturbed graph guided by these properties. For the incomplete, Chen et al. [20] propose a graph learning framework that jointly learning the graph structure and graph embeddings simultaneously.

However, these methods are all designed for the supervised graph neural networks (GNNs) not the unsupervised graph autoencoder (GAEs). Besides that, they use k-nearest neighbor (kNN) to initialize the adjacency matrix when the graph structure is unavailable. A major shortcoming of this approach is that the efficacy of the resulting models hinges on the choice of k [21]. In any case, the graph creation and parameter learning steps are independent and require heuristics and trial and error.

In this paper, two novel unsupervised graph embedding methods, *unsupervised graph embedding via adaptive graph learning* (BAGE) and *unsupervised graph embedding via variational adaptive graph learning* (VBAGE) are developed. The contributions can be summarized below:

• The proposed method expands the application range of GAEs on graph embedding, i.e, on the general datasets without graph structure. The adaptive learning mechanism is able to initialize the adjacency matrix without be affected by the parameter k.

• The latent representations are embedded in the laplacian graph structure to preserve the topology structure of the graph in the vector space.

• With adaptive learning, the adjacency matrix can be self-learned for better embedding performance which enhances the robustness of the model.

2 Related Work

In this section, we outline the background and development of graph embedding and graph autoencoders (GAEs).

The goal of graph embedding is to learn the low-dimensional latent representations of nodes that preserve the topological information of the graph. The early methods such as DeepWalk [22] uses a random walk to generate sequences of nodes from a network and transform graph construction information into linear sequences. Inspired by DeepWalk, DRNE [23] adopts a Long Short Term Memory (LSTM) network to aggregate a node's neighbors. Similar to DRNE, NetRA [24] also uses the LSTM network with random walks rooted on each node and regularizes the learned network embeddings within a prior distribution via adversarial training. The embedding method SDNE [14] exploits the first-order and second-order proximity jointly to preserve the network structure. The structure of SDNE is very similar to the graph autoencoders and can be seen as an early approach of GAEs.

The relationship between GAEs and graph embedding can be understood as: GAEs are a general term for a series of methods, and graph embedding is one of the tasks that GAE can perform. Earlier GAE approaches such as DNGR [25] and SDNE [14] mainly build the GAE frameworks for graph embedding by multi-layer perceptrons. Nevertheless, DNGR and SDNE only consider node structure information but ignore the node features information. In other words, the early GAE methods directly learn the node embeddings from a graph where each node on this graph does not contain feature information. What really kicked off graph autoencoder is [26] whose methods GAE and VGAE laid the foundation for the later GAE methods. Inspired by generative adversarial networks (GANs) and VGAE [26], Adversarially Regularized Variational Graph Autoencoder (ARGE and ARVGE) [27] is proposed that endeavors to learn an encoder that produces the empirical distribution. By replacing the GCN encoder with a simple linear model, Salha et al. [28] propose a linear graph autoencoder (LGAE and LVGAE) which is more simple.

Our methods (BAGE and VBAGE) also use the graph autoencoder to learn graph embedding. However, our method differs from these methods in the following points:
1) Our methods can be applied to more general datasets, i.e., the dataset without the graph structure.
2) The learned latent representations are embedded into the laplacian graph structure to preserve the topology structure of the graph in the vector space.
3) The adjacency matrix in our framework can be adaptively learned, which enhances the robustness of the model.

We will detailly describe these aspects in the rest of the paper.

3 Notations and Problem Statement

Before we present the problem statement, we first introduce some notations and basic concepts. The Frobenius norm of a matrix $\mathbf{A}$ is defined by $\|\mathrm{A}\|_F^2 = \Sigma_{ij} a_{ij}^2$. We use $\odot$ to denote the Hadamard product of matrices and $tr(A)$ to indicate the trace of matrix $\mathbf{A}$, i.e., $tr(\mathbf{A}) = \sum_i a_{ii}$. Epoch in the paper means the number of iteration.

Let $\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$ be a graph, where $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes with $|\mathbf{V}| = n$ and $\mathbf{E}$ is the set of connecting edges among each node. The edges describe the relations between nodes and can also be represented by an adjacency matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ where a_{ij} denotes the relation between nodes v_i and v_j. Furthermore, we use $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times m}$ to denote the node feature matrix where $\mathbf{x}_i$ is the feature vector of node v_i. The number of nodes is n and m is the dimension of the raw data.

The aim of graph embedding in our method is to learn the low-dimensional latent representation $\mathbf{Z} \in \mathbb{R}^{n \times f}$ from the node matrix $\mathbf{V}$ with the formal format as: $f : (\mathbf{A}, \mathbf{X}) \mapsto \mathbf{Z}$. The learned latent representation $\mathbf{Z}$ in latent space should preserve the topological structure of the graph as well as node feature information.

4 Framework

Fig. 1 illustrates the workflow of our methods (*BAGE* and *VBAGE*) that consists of three modules: the graph convolutional encoder, the decoder, and the laplacian graph structure.

4.1 Graph Convolutional Encoder Model

The encoder model learns a layer-wise transformation by a spectral graph convolutional function $f\left(\mathbf{Z}^{(l)}, \mathbf{A}|\mathbf{W}^{(l)}\right)$:

$$\mathbf{Z}^{(l+1)} = f(\mathbf{Z}^{(l)}, \mathbf{A}|\mathbf{W}^{(l)}), \tag{1}$$

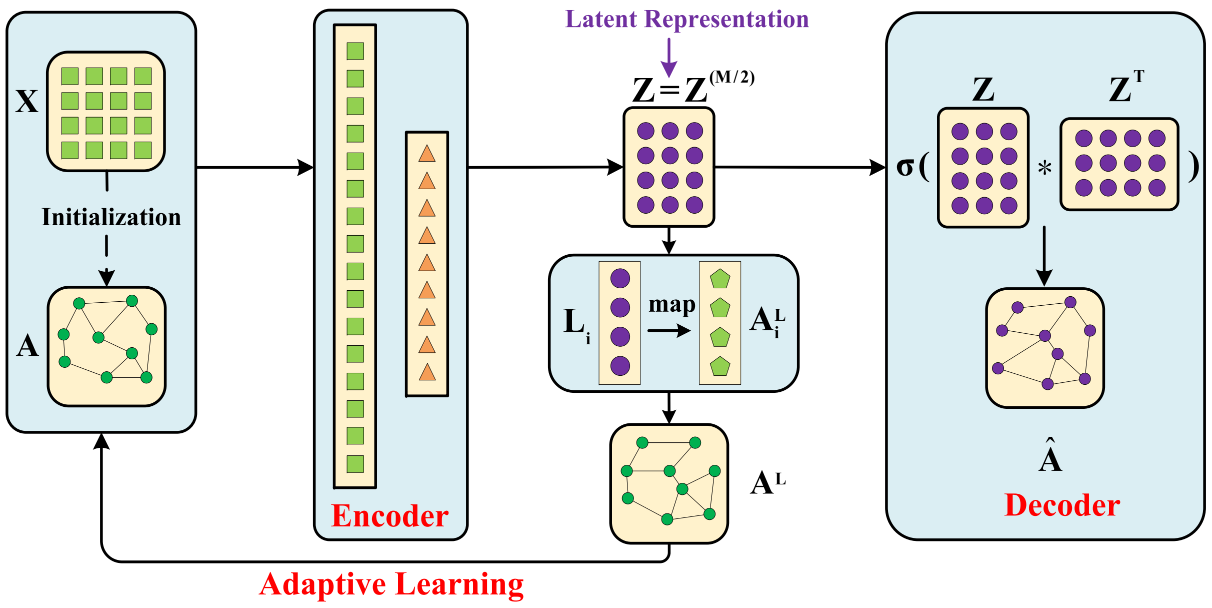


Fig. 1: The architecture of the proposed framework (See Algorithm 1 for details).

where $\mathbf{Z}^{(l)}$ is the input for convolution and $\mathbf{Z}^{(l+1)}$ is the output after convolution. $\mathbf{W}^{(l)}$ is the weight parameter matrix that needs to be learned in the neural network. In this paper, $\mathbf{Z}^{(0)} = \mathbf{X} \in \mathbb{R}^{n\times m}$ is the input node features matrix. Specifically speaking, each layer of our graph convolutional network can be calculated as follows:

$$f\left(\mathbf{Z}^{(l)}, \mathbf{A}|\mathbf{W}^{(l)}\right) = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{(l)}\mathbf{W}^{(l)}\right). \tag{2}$$

Here, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$, $\mathbf{I}$ is the identity matrix of $\mathbf{A}$, and ϕ is the activation function such as $\mathrm{Relu}(t) = \max(0,t)$.

• The *Graph Encoder* in our method (BAGE) is like [26] and [17], which is constructed as follows:

$$\mathbf{Z}^{(1)} = f_{\mathrm{Relu}}(\mathbf{X}, \mathbf{A}|\mathbf{W}^{(0)}); \tag{3}$$

$$\mathbf{Z}^{(2)} = f_{\mathrm{linear}}(\mathbf{Z}^{(1)}, \mathbf{A}|\mathbf{W}^{(1)}). \tag{4}$$

• The *Variational Graph Encoder* in our method (VBAGE) is defined by an inference model:

$$\begin{aligned} q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) &= \prod_{i=1}^{n} q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}), \\ q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) &= \mathcal{N}\left(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \mathrm{diag}\left(\boldsymbol{\sigma}^2\right)\right). \end{aligned} \tag{5}$$

Here, $\boldsymbol{\mu} = \mathbf{Z}^{(2)}$ is the matrix of mean vectors $\mathbf{z}_i$; similarly $\log\boldsymbol{\sigma} = f_{\mathrm{linear}}\left(\mathbf{Z}^{(1)}, \mathbf{A} \mid \mathbf{W}'^{(1)}\right)$ which share the weights $\mathbf{W}^{(0)}$ with $\boldsymbol{\mu}$ in the first layer in Eq. (3).

4.2 Decoder Model

The decoder model reconstructs the graph from the learned latent representations and the reconstructed adjacency matrix $\widehat{\mathbf{A}}$ can be represented as follows:

$$\widehat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{Z}\mathbf{Z}^T), \tag{6}$$

where $\mathbf{Z}$ is the latent representation and $\mathbf{Z} =$ *encoder* $(\mathbf{Z}|\mathbf{X}, \mathbf{A})$.

In terms of loss function, we did not use the cross-entropy loss function to define the reconstruction loss like [26] and [17]. We impose more penalty on the reconstruction error of the non-zero elements than that of zero elements.

• The reconstruction loss of the graph construction for BAGE is calculated as:

$$\begin{aligned} \mathcal{L}_{G1} &= \sum_{i=1}^{n} \|(\mathbf{a}_i - \hat{\mathbf{a}}_i) \odot \mathbf{b_i}\|_2^2, \\ &= \|(\mathbf{A} - \widehat{\mathbf{A}}) \odot \mathbf{B}\|_F^2, \end{aligned} \tag{7}$$

where $\odot$ means the Hadamard product, $\mathbf{b_i} = \{b_{i,j}\}_{j=1}^n$. If $a_{i,j} = 0, b_{i,j} = 1$, else $b_{i,j} = \beta > 1$.

• The reconstruction loss of the graph construction for VBAGE is calculated as:

$$\mathcal{L}_{G2} = \|(\mathbf{A} - \widehat{\mathbf{A}}) \odot \mathbf{B}\|_F^2 - \mathbf{KL}[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A})\|p(\mathbf{Z})], \tag{8}$$

where $\mathbf{KL}[q(\bullet)\|p(\bullet)]$ is the Kullback-Leibler divergence between $q(\bullet)$ and $p(\bullet)$. We also take a Gaussian prior $p(\mathbf{Z}) = \prod_i p(\mathbf{z}_i) = \prod_i \mathcal{N}(\mathbf{z}_i \mid 0, \mathbf{I})$ like [17] and [26].

4.3 Laplacian Graph Structure

The reconstruction loss in Eqs. (7) and (8) is only focused on graph reconstruction while ignoring latent representations. Aim at this, we embed the latent representations into the laplacian graph structure. The loss function for this goal is defined as follows:

$$\begin{aligned} \mathcal{L}_L &= \sum_{i,j=1}^{n} \left(\|\mathbf{z}_i - \mathbf{z}_j\|_2^2 a_{ij} + \gamma_i a_{ij}^2\right) \\ &= \mathrm{tr}(\mathbf{Z}\mathbf{L}\mathbf{Z}^T) + \gamma\|\mathbf{A}\|_F^2 \\ &\text{s.t.} \quad \mathbf{a}_i^T\mathbf{1} = 1, \mathbf{0} \le \mathbf{a}_i \le \mathbf{1}, \end{aligned} \tag{9}$$

where γ is a regularization parameter that can be adaptively solved. The laplacian graph structure is also the basis of the adaptive learning of the adjacency matrix that is also the biggest contribution of this paper.

The objective function of Eq. (9) borrows the idea of Laplacian Eigenmaps [29], which incurs a penalty when similar latent representations are far away in the embedding space. The Laplacian matrix $\mathbf{L}$ is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{10}$$

where the $\mathbf{D}$ is the degree matrix whose diagonal element $d_{ii} = \sum_j a_{ij}$.

Similar ideas appear in some works on graph learning and network embedding. The difference between our methods and them is that we directly incorporate the laplacian graph structure into the latent representations, not into the feature matrix, like CAN [30] or into the label matrix, like SDNE [14]. Thus laplacian graph structure can make the vertexes linked by an edge be mapped near in the embedding space.

• In summary, the loss function of the BAGE can be written as:

$$\mathcal{L}_{BAGE} = \mathcal{L}_{G1} + \lambda \mathcal{L}_L + \nu \mathcal{L}_{reg}. \tag{11}$$

• And the loss function of the VBAGE can be written as:

$$\mathcal{L}_{VBAGE} = \mathcal{L}_{G2} + \lambda \mathcal{L}_L + \nu \mathcal{L}_{reg}, \tag{12}$$

where $\mathcal{L}_{reg}$ is the ℓ_2-norm regularizer term with coefficient ν to prevent overfitting, which is defined as follows:

$$\mathcal{L}_{reg} = \frac{1}{2} \sum_i (\|\mathbf{W}^{(i)}\|_F^2). \tag{13}$$

5 Adaptive Learning of the Adjacency Matrix

The most important contribution of this paper is the adaptive learning of the adjacency matrix that tries to answer the following two questions:

• When the original graph structure is incomplete, can we learn an alternative graph structure to obtain better embedding effects?

• When we apply graph convolution to the data without an initial graph structure, can we get the graph structure through adaptive learning rather than kNN initialization?

5.1 The Solution of The Adjacency Matrix

In practical applications, the adjacency matrix with adjustable sparsity tends to bring better results. And that is a reason why we do not update the adjacency matrix by back-propagation algorithm directly, which will produce a meaningless dense matrix. The adaptive learning of the adjacency matrix is based on the laplacian graph structure in Eq. (9) as follows:

$$\begin{aligned} &\min_{a_{ij}} \sum_{i,j=1}^{n} \left(\|\mathbf{z}_i - \mathbf{z}_j\|_2^2 \, a_{ij} + \gamma_i a_{ij}^2 \right) \\ &\text{s.t.} \quad \mathbf{a}_i^T \mathbf{1} = 1, \mathbf{0} \le \mathbf{a}_i \le \mathbf{1}. \end{aligned} \tag{14}$$

Let us denote the distance between two nodes as h_{ij}, i.e., $h_{ij} = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$. The j-th element of vector $\mathbf{h}_i \in \mathbb{R}^{n\times 1}$ is j-th element h_{ij} and $\mathbf{a}_i \in \mathbb{R}^{n\times 1}$ is a vector with its j-th element a_{ij}. The Lagrange equation of problem (14) is represented as:

$$\mathcal{L}(\mathbf{a}_i, \eta, \boldsymbol{\zeta}_i) = \frac{1}{2} \left\| \mathbf{a}_i + \frac{\mathbf{h}_i}{2\gamma_i} \right\|_2^2 - \eta(\mathbf{a}_i^T \mathbf{1} - 1) - \boldsymbol{\zeta}_i^T \mathbf{a}_i, \tag{15}$$

where η and $\boldsymbol{\zeta}_i \ge \mathbf{0}$ are the Lagrange multipliers. Using the Karush-Kuhn-Tucker (KKT) conditions, we can derive the optimal solution of a_{ij} as:

$$a_{ij} = \left(-\frac{h_{ij}}{2\gamma_i} + \eta \right)_+, \tag{16}$$

where $(\bullet)_+ = \max(\bullet, 0)$. To equip the adjacency matrix with adjustable sparsity, we take only k nodes points closest to $\mathbf{z}_i$ into consideration and the parameter k is responsible for adjusting the sparsity of the adjacency matrix. Therefore, $\mathbf{a}_i$ satisfies $a_{ik} > 0 \ge a_{i,k+1}$ as:

$$\begin{cases} a_{ik} > 0 \Rightarrow -\frac{h_{ik}}{2\gamma_i} + \eta > 0 \\ a_{i,k+1} \le 0 \Rightarrow -\frac{h_{i,k+1}}{2\gamma_i} + \eta \le 0 \end{cases}. \tag{17}$$

According to Eq. (16) and the constraint $\mathbf{a}_i^T \mathbf{1} = 1$, we have:

$$\sum_{j=1}^{k} \left(-\frac{h_{ij}}{2\gamma_i} + \eta \right) = 1 \Longrightarrow \eta = \frac{1}{k} + \frac{1}{2k\gamma_i} \sum_{j=1}^{k} h_{ij}. \tag{18}$$

The overall γ is set to the mean of γ_i and it can be learned adaptively as:

$$\gamma = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{k}{2} h_{i,k+1} - \frac{1}{2} \sum_{j=1}^{k} h_{ij} \right). \tag{19}$$

Without loss of generality, let us suppose $h_{i1}, h_{i2}, \ldots, h_{in}$ are ordered from small to large. Then the adjacency matrix can be solved as:

$$a_{ij} = \left(\frac{h_{i,k+1} - h_{ij}}{k h_{i,k+1} - \sum_{j=1}^{k} h_{ij}} \right)_+. \tag{20}$$

Based on Eq. (20), the adjacency matrix $\mathbf{A}$ can be updated to $\mathbf{A}^{\mathbf{L}}$ by the latent representations. What is more, the adjacency matrix can also be initialized by Eq. (20) when there is no initial graph structure.

5.2 Update The Adjacency Matrix

Eq. (20) provides a solution to the graph structure when the adjacency matrix does not exist or is incomplete. During the iterative process, the learned latent representations can be used to calculate the adjacency matrix for the next iteration. However, it is harmful to discard the initial graph structure totally since the optimal graph structure is potentially a small shift from the initial graph structure [20].

With this assumption, we combine the learned graph structure with the initial graph structured as follows:

$$\mathbf{A} = \alpha \mathbf{A}_L + (1 - \alpha) \mathbf{A}_0, \tag{21}$$

where $\mathbf{A}_0$ is initial adjacency matrix and $\mathbf{A}_L$ is the learned adjacency matrix in the iteration. A hyperparameter α is used to balance the trade-off between the learned graph structure and the initial graph structure. Moreover, we set a threshold τ for stopping updates, i.e., if $(epoch > \tau)$, then the updating stop.

5.3 Distribution of Parameter k

There is only one parameter k in Eq. (20). If we set k as a fixed hyperparameter, it will cause all samples to have the same number of neighbors, i.e., the number of non-zero elements in each row of the adjacency matrix is the same. It will bring the same disadvantage as kNN since the number of neighbors for each sample is mostly different in real-world graph data.

Instead of fixing k, we make k to be a parameter that can be learned adaptively. Since the k represents the number of neighbors for each sample, i.e., the number of non-zero elements in each row of the adjacency matrix, we sample k from the normal distribution. k obeys a normal distribution with a mean μk and a variance of 1, i.e., $k \sim \mathcal{N}(uk, 1)$. The value of μk is the number of neighbors for the sample in the former iteration. Besides that, we set a maximum and minimum limit on k to keep it in a reasonable range. Specific to Eq. (20), the number of neighbors for each sample in the adjacency matrix will be dynamically obtained during the updating.

6 Optimization

To optimize the aforementioned models, the goal is to minimize the loss of $\mathcal{L}_{BAGE}$ 11 and $\mathcal{L}_{VBAGE}$ 12 which are the function of the neural network parameter $\mathbf{W}$. The calculation of the partial derivative of $\mathcal{L}_{BAGE}$ 11 and $\mathcal{L}_{VBAGE}$ 12 are estimated using the back-propagation. Furthermore, the proposed framework is optimized by adaptive moment estimation (Adam), where T is the maximum iteration number.

As for the adaptive learning of the adjacency matrix, the time complexity of the updating process is $\mathcal{O}((dn^2)\tau)$. To show clearly the information transmission in the adaptive learning process, Fig. 2 is made to show the information flow of the learned adjacency matrix $\mathbf{A}$ and the intermediate node embedding matrix $\mathbf{Z}$ during the iterative procedure. The presence of the blue arrow from $\mathbf{X}$ to $\mathbf{A}^{(0)}$ depends on whether there is a graph structure in the initial data. The pseudocode of our methods is summarized in Algorithm 1.

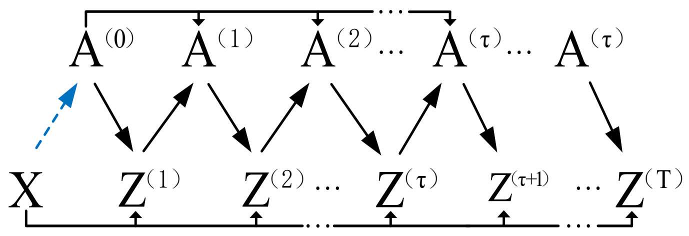


Fig. 2: Information flow of the adaptive learning process.

7 Experiments

We report our results on three tasks: node clustering, node classification, and graph visualization. The benchmark datasets used in this paper include two real-world graph datasets and six general datasets. The results not only demonstrate the advantage of our methods but also support the effectiveness of the adaptive learning of the adjacency matrix.

Algorithm 1 Algorithm to our methods (BAGE and VBAGE)

```
Input: Node features matrix $\mathbf{X}$, maximum iteration number $T$, parameters $\lambda$, $\alpha$ and $\tau$.

    Initialization:
 1: if There is the initial graph structure in the data:
 2:     Input $\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$
 3: else:
 4:     Initialize the adjacency matrix with Eq. (20)
 5: end

    Optimization:
 6: for epoch = 1,2,3, ..., $T$ do:
 7:     if epoch = 1,2,3, ..., $\tau$:
 8:         Update the adjacency matrix by Eqs. (20) and (21)
 9:     end
10:     Compute the loss of BAGE by Eq. (11)
11:     Compute the loss of VBAGE by Eq. (12)
12:     Backpropagate loss and update $\mathbf{W}$
13: end
Output: Latent representation $\mathbf{Z}$
```

7.1 Datasets

Graph Datasets. In the node clustering task, two graph datasets Cora and Citeseer are used. In order to verify the effectiveness of adaptive learning in our method, we perform incomplete processing on the adjacency matrix on these two graph datasets. Incomplete processing means that the elements in the adjacency matrix are randomly set to 0 with a certain probability (Missing Ratio). The detailed information of the two graph datasets is shown in Table 1.

General Datasets. In the node classification task, six general datasets are used. The is no graph structure in these datasets where we initialize the adjacency matrix by kNN for other methods which are based on the graph convolutional neural network. The purpose of using these datasets is to verify the superiority of the adaptive learning of the adjacency matrix in our methods. The detailed information of the six general datasets is shown in Table 2.

7.2 Competitors

We compare our algorithms against several state-of-the-art algorithms:

- **LGAE and LVGAE** [28]: is the latest improvement to GAE and VGAE, which replace the GCN encoder by a simple linear model w.r.t. the adjacency matrix of the graph.
- **ARGE and ARVGE** [17]: is the adversarial graph embedding framework that enforces latent representations to match a prior distribution, which is achieved by an adversarial training mode.
- **GAE and VGE** [26]: is the classical graph autoencoder and variational graph autoencoder. And it is the first method to to apply graph convolution in the autoencoder.
- **SDNE** [14]: exploits the first order and second order proximity jointly to preserve the network structure and we use it as a baseline.
- **Spectral Clustering** [31]: is a famous graph based clustering method and we use it as a baseline.

TABLE 1: Information of the two graph datasets.

Dataset	# Nodes	# Links	# Content Words	# Features	# Missing Ratio
Cora	2,708	5,429	3,880,564	1,433	{0%, 5%, 10%, 15%, 20%, 25%, 50%}
Citeseer	3,327	4,732	12,274,336	3,703	{0%, 5%, 10%, 15%, 20%, 25%, 50%}

TABLE 2: Information of the six general datasets.

Datasets	IMM	ATT	UMIST	COIL	USPS	PIE
# Nodes	240	400	575	1,440	2007	3332
# Features	1,024	1024	1024	1,024	256	4096

• k**-means** is the base of many clustering methods. Here we run k-means on raw node features as a baseline.

7.3 Task 1: Node Clustering

Like most papers based on the graph convolutional neural network, we apply our methods and other competitors on the node clustering task to verify the effect of graph embedding.

Parameter Settings and Study. For the Cora and Citeseer datasets, we train all autoencoder-related models for 200 iterations and optimize them with the Adam algorithm. The learning rate for the BAGE is 0.0001 and VBAGE is 0.001. When the adjacency matrix is incomplete, the scale of adaptive learning α is 10% and the number of adaptive learning iterations (τ) is limited to 10-15. The λ is set as 0.01, and the elements of weight matrix β is set as 20. What's more, the parameters for other competitors are set to the values that make the best experimental results.

As for the impact of parameters, we study the influence of parameters λ and β on the experimental results. The results and analysis of the parameter study are merged into the next subsection.

Metrics. To verify the effect of graph embedding, we run k-means on the learned latent representations to perform the node clustering task. We run k-means 10 times with different initializations and report the mean and standard deviation of all methods. To validate the clustering results, we employ two metrics: Accuracy (ACC) and Normalized Mutual Information (NMI).

Experimental Results and Analysis. The details of the experimental results on the node clustering are given in Tables 3 and 4, where the best and second results have been highlighted. Since the performance of competitors SDNE, Spectral Clustering (SC), and k-means is not very good, the results of those baselines are listed in a separate Table 5. The results observations are as follows:

1) Our methods outperform all other competitors on the Cora dataset regardless of whether the adjacency matrix is incomplete.

2) In the Cora dataset, our methods (BAGE and VBAGE) are nearly 4%-6% higher than the second place (ARGE and ARVGE) when the missing ratio is little, and 7%-9% higher than the second place (ARGE and ARVGE) when the missing ratio is large.

3) In the Citeseer dataset, our methods perform 3%-6% higher than the other competitors when the missing ratio is less than 50%.

7.4 Task 2: Node Classification

In this task, we apply all methods on six general datasets to learn the latent representations which are then applied to the node classification task to verify the quality of the embedding results.

Parameter Settings and Study. All the parameter settings for our methods in this task are the same as in the node clustering task. As for the parameters in other competitors, they are set as the values that make the best experimental results.

For the impact of the parameters, we study the influence of parameters λ and β on the experimental results. The value range of λ is $\{0.001, 0.01, 0.1, 1, 10\}$ and the value range of β in weight matrix $\mathbf{B}$ is $\{1, 10, 20, 30, 40\}$. The evaluation index is F1-score that is commonly used in the classification task. Fig. 5 shows the results of parameters study where the results show that the parameters' influence on our method on the node classification task is small.

Metrics. To verify the effect of graph embedding, we run SVM on the learned latent representations to perform the classification task. The latent representations are divided into a training set (70%) and a test set (30%). We first perform 10-fold cross-validation to select the best SVM model, and then we apply the selected SVM model on the test set. To validate the node classification results, we employ the F1-score as the metric.

Experimental Results and Analysis. The details of the experimental results on the node classification task are displayed in Table 6 where the best and second results have been highlighted. Due to space constraints, we only select a few outstanding competitors, i.e., ARGE, LGAE, GAE, and VGAE, to show in Table 6. The node classification results in Table 6 show the superiority of our method.

1) The performance of BAGE on the six general datasets has always remained in the top two and our method (BAGE and VBAGE) is higher than other methods 3%-6% in the COIL dataset.

2) In UMIST and USPS, our methods (BAGE and VBAGE) perform better than GAE, ARGE, and LGAE and the performance of our method occupies the top two.

3) In the PIE dataset, BAGE can achieve 95% when other methods can only reach 91%.

In summary, the experiment results in the node classification task fully illustrate the rationality and superiority of the adaptive learning of the adjacency matrix in our method.

TABLE 3: The node clustering results on the Cora dataset.

Methods	Metrics	0%	5%	10%	15%	20%	25%	50%
GAE	ACC(%)	59.53±2.00	58.27±3.68	55.13±2.74	57.19±2.30	54.57±1.91	55.58±3.60	48.77±2.73
	NMI(%)	40.48±1.09	38.82±1.78	32.64±1.12	36.67±1.77	35.10±1.61	36.98±1.69	25.23±1.60
VGAE	ACC(%)	58.58±4.02	57.64±2.82	56.48±3.84	56.64±2.31	56.88±2.92	56.95±4.56	54.75±4.31
	NMI(%)	38.46±2.22	37.39±1.59	38.14±2.72	35.42±1.35	36.80±2.44	35.05±3.26	33.00±2.91
ARGE	ACC(%)	68.64±0.60	67.66±0.69	65.73±1.62	63.37±2.54	60.69±1.20	59.44±1.78	55.21±0.52
	NMI(%)	49.22±0.17	46.30±0.91	45.12±1.22	44.07±1.72	42.07±0.30	39.52±0.83	34.33±0.42
ARVGE	ACC(%)	67.66±0.26	66.60±0.80	65.99±1.12	62.44±2.31	58.47±0.91	60.70±0.08	56.32±1.33
	NMI(%)	48.17±0.21	47.24±0.42	47.20±0.67	44.07±0.65	41.75±0.09	42.27±0.14	33.92±0.71
LGAE	ACC(%)	58.42±0.90	57.46±3.93	57.82±1.64	53.02±2.47	51.55±2.81	54.66±2.86	45.12±2.35
	NMI(%)	36.63±0.49	39.71±1.39	39.34±0.84	36.62±1.44	32.89±0.92	35.46±0.94	23.84±1.28
LVGAE	ACC(%)	57.10±2.29	56.25±2.75	57.82±2.93	54.08±3.87	55.00±2.84	55.92±1.86	49.68±1.63
	NMI(%)	32.77±1.63	30.71±1.10	32.08±1.23	30.26±2.25	29.41±1.42	29.93±0.77	23.50±0.68
BAGE	ACC(%)	**72.57±0.68**	**71.21±0.18**	**70.32±0.33**	**72.74±0.12**	**69.43±2.27**	**66.59±0.87**	**64.22±0.29**
	NMI(%)	**56.91±0.49**	**54.73±0.15**	**52.39±0.17**	**52.43±0.24**	**49.96±0.79**	**51.88±0.54**	**48.53±0.25**
VBAGE	ACC(%)	**73.11±0.25**	**71.97±0.05**	**71.45±0.87**	**71.23±0.19**	**70.72±0.27**	**68.25±2.42**	**64.86±1.18**
	NMI(%)	**55.66±0.14**	**54.13±0.23**	**53.15±0.28**	**51.12±0.24**	**51.02±0.32**	**49.87±1.29**	**44.85±0.21**

TABLE 4: The node clustering results on the Citeseer dataset.

Methods	Metrics	0%	5%	10%	15%	20%	25%	50%
GAE	ACC(%)	54.54±3.45	54.19±2.62	50.95±2.27	49.17±1.50	50.96±1.76	44.25±2.76	43.51±2.81
	NMI(%)	27.04±1.53	24.40±2.12	21.49±1.44	20.55±1.09	22.33±1.28	16.25±1.62	16.18±1.73
VGAE	ACC(%)	53.96±0.98	53.73±1.43	52.84±1.27	50.24±0.88	51.78±1.28	47.93±2.50	41.87±2.81
	NMI(%)	23.36±0.82	23.87±0.96	22.66±0.89	21.48±0.97	22.71±0.93	18.89±1.55	14.29±1.35
ARGE	ACC(%)	58.67±1.20	57.95±0.62	54.80±1.19	50.49±1.68	51.66±0.99	51.40±1.42	**45.42±0.14**
	NMI(%)	31.33±0.85	30.69±0.41	27.75±1.18	24.17±1.08	24.81±0.55	24.18±0.82	**20.26±0.26**
ARVGE	ACC(%)	49.85±0.60	51.43±0.50	49.93±0.10	48.14±0.11	50.04±0.20	52.14±0.88	42.96±1.88
	NMI(%)	24.67±0.42	25.10±0.48	24.15±0.27	22.71±0.28	25.42±0.20	25.58±0.43	17.28±1.41
LGAE	ACC(%)	56.66±0.96	56.93±0.88	56.58±0.94	55.08±1.14	55.63±1.01	54.74±0.42	**46.83±1.00**
	NMI(%)	27.22±0.59	28.41±0.73	27.00±0.86	26.67±0.88	26.38±0.91	25.21±0.34	**28.48±0.54**
LVGAE	ACC(%)	50.86±1.05	50.83±1.27	50.06±0.69	49.07±1.09	49.91±1.57	48.82±0.82	28.97±0.41
	NMI(%)	22.55±0.48	21.38±0.41	22.03±0.55	20.19±0.38	20.85±0.64	19.45±0.36	10.31±0.06
BAGE	ACC(%)	**64.64±0.19**	**62.19±0.34**	**58.83±2.23**	**57.96±1.14**	**59.64±0.48**	**55.18±2.12**	39.91±3.57
	NMI(%)	**39.07±0.30**	**37.99±0.21**	**36.03±1.13**	**32.13±0.61**	**35.17±0.35**	**26.17±0.98**	14.62±1.85
VBAGE	ACC(%)	**63.13±0.16**	**62.73±0.28**	**60.91±0.19**	**58.40±0.62**	**58.80±0.41**	**55.87±2.11**	35.75±1.54
	NMI(%)	**36.92±0.12**	**37.62±0.15**	**35.81±0.16**	**33.79±0.26**	**27.81±0.22**	**26.89±0.82**	13.11±0.42

TABLE 5: The node clustering results for three baselines.

Methods	Metrics	SDNE	SC	k-means
Cora	ACC(%)	41.52±3.38	38.08±0.04	37.22±3.91
	NMI(%)	20.17±2.39	15.99±0.13	18.87±3.51
Citeseer	ACC(%)	30.21±0.67	21.46±0.00	43.80±5.83
	NMI(%)	4.44±0.33	1.72±0.00	20.63±4.67

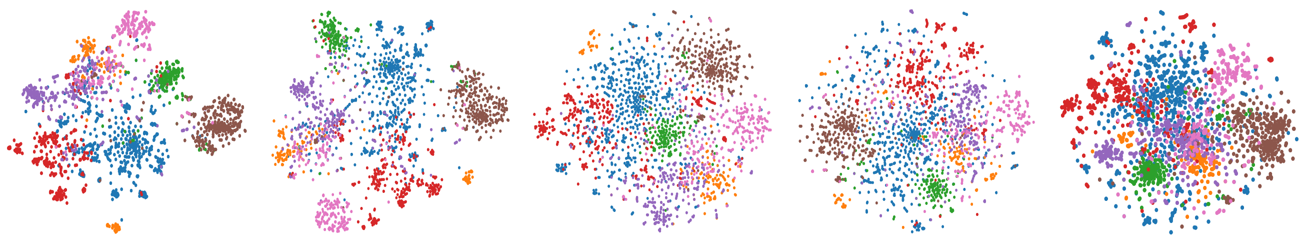
7.5 Task 3: Graph Visualization

In this task, we apply our methods and several outstanding competitors to perform visualization in two graph datasets. We visualize the Cora and Citeseer datasets in a two-dimensional space by applying the t-SNE [32] algorithm on the learned embeddings.

The results in Figs. 3 and 4 validate that by applying adaptive learning of the adjacency matrix, we can obtain a more meaningful layout of the graph data.

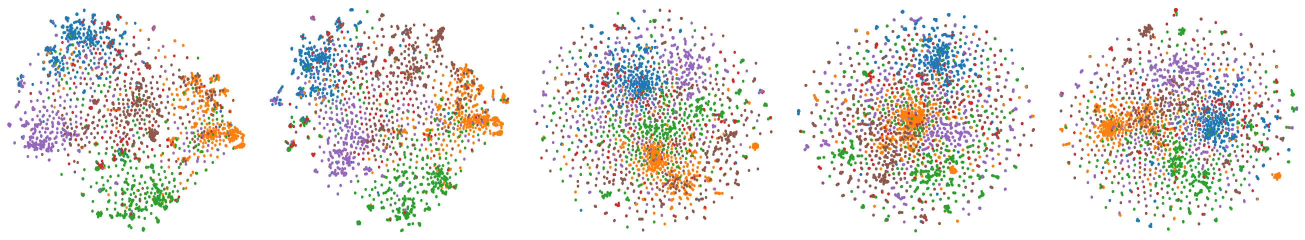
8 Conclusion

In this paper, we propose two novel unsupervised graph embedding methods, *unsupervised graph embedding via adaptive graph learning* (BAGE) and *unsupervised graph embedding via variational adaptive graph learning* (VBAGE). Aiming at the problem that the existing GAE methods are sensitive to the adjacency matrix, we embed the adaptive learning to the framework, which enhances the robustness of the model. In addition, the adaptive learning mechanism expands the application range of GAEs on graph embedding and is able to



(a) BAGE (b) VBAGE (c) ARGE (d) GAE (e) LGAE

Fig. 3: The data visualization comparison on Cora.



(a) BAGE (b) VBAGE (c) ARGE (d) GAE (e) LGAE

Fig. 4: The data visualization comparison on Citeseer.

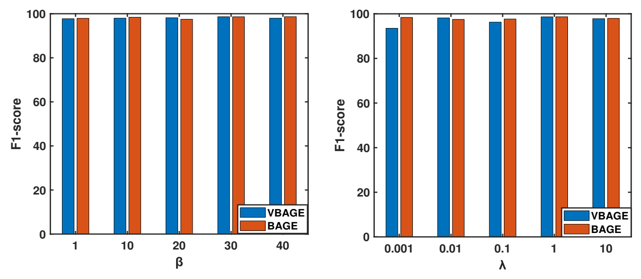


Fig. 5: The parameters study results on COIL dataset.

TABLE 6: The F1-score (%) on the node classification task.

Methods	BAGE	VBAGE	ARGE	LGAE	GAE	VGAE
COIL	**97.50**	**98.15**	94.33	93.85	93.86	92.84
ATT	**86.59**	**87.54**	83.87	83.50	78.48	76.24
IMM	**36.71**	25.81	**32.39**	30.05	20.43	19.54
UMIST	**96.46**	**95.92**	92.39	91.80	87.86	85.88
USPS	**94.19**	**93.86**	92.88	93.06	93.37	92.89
PIE	**95.29**	91.00	91.63	**91.69**	87.69	87.54

initialize the adjacency matrix without be affected by the parameter k. Furthermore, the learned latent representations are embedded in the laplacian graph structure to preserve the topology structure of the graph in the vector space. Experimental studies on several datasets demonstrated that our methods (BAGE and VBAGE) outperform baselines by a wide margin in node clustering, node classification, and graph visualization tasks.

References

[1] Zhengyang Wang and Shuiwang Ji, "Second-order pooling for graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[2] Lu Bai, Lixin Cui, Yuhang Jiao, Luca Rossi, and Edwin Hancock, "Learning backtrackless aligned-spatial graph convolutional networks for graph classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[3] Zhen Wang, Zhaoqing Li, Rong Wang, Feiping Nie, and Xuelong Li, "Large graph clustering with simultaneous spectral embedding and discretization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[4] Xuelong Li, Han Zhang, Rong Wang, and Feiping Nie, "Multiview clustering: A scalable and parameter-free bipartite graph fusion method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[5] Junming Shao, Zhong Zhang, Zhongjing Yu, Jun Wang, Yi Zhao, and Qinli Yang, "Community detection and link prediction via cluster-driven low-rank matrix completion.," in *IJCAI*, 2019, pp. 3382–3388.

[6] Junyu Gao, Tianzhu Zhang, and Changsheng Xu, "Learning to model relationships for zero-shot video classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[7] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[8] Mikhail Belkin and Partha Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[9] Sam T Roweis and Lawrence K Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[10] Palash Goyal and Emilio Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[11] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.

[12] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[13] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[14] Daixin Wang, Peng Cui, and Wenwu Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[15] Thomas N Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[16] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[17] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang, "Adversarially regularized graph autoencoder for graph embedding.," in *IJCAI*, 2018, pp. 2609–2615.

[18] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[19] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang, "Graph structure learning for robust graph neural networks," *arXiv preprint arXiv:2005.10203*, 2020.

[20] Yu Chen, Lingfei Wu, and Mohammed J Zaki, "Deep iterative and adaptive learning for graph neural networks," *arXiv preprint arXiv:1912.07832*, 2019.

[21] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He, "Learning discrete structures for graph neural networks," in *ICML*, 2019.

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

[23] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu, "Deep recursive network embedding with regular equivalence," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2357–2366.

[24] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2663–2671.

[25] Shaosheng Cao, Wei Lu, and Qiongkai Xu, "Deep neural networks for learning graph representations," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[26] Thomas N Kipf and Max Welling, "Variational graph autoencoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[27] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang, "Adversarially regularized graph autoencoder for graph embedding," .

[28] Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis, "Keep it simple: Graph autoencoders without graph convolutional networks," Workshop on Graph Representation Learning, 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019.

[29] Mikhail Belkin and Partha Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[30] Feiping Nie, Xiaoqian Wang, and Heng Huang, "Clustering and projected clustering with adaptive neighbors," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 977–986.

[31] Andrew Y Ng, Michael I Jordan, and Yair Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[32] Laurens Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.