Abstract

Graph autoencoder (GAE) serves as an effective unsupervised learning framework to represent graph data in a latent space for network embedding. Most exiting approaches typically focus on minimizing the reconstruction loss of graph structure but neglect the reconstruction of node features, which may result in overfitting due to the capacity of the autoencoders. Additionally, the adjacency matrix in these methods is always fixed such that the adjacency matrix cannot properly represent the connections among nodes in latent space. To solve this problem, in this paper, we propose a novel Graph Convolutional Auto-encoder with Bi-decoder and Adaptive-sharing Adjacency method, namely BAGA. The framework encodes the topological structure and node features into latent representations, on which a bi-decoder is trained to reconstruct the graph structure and node features simultaneously. Furthermore, the adjacency matrix can be adaptively updated by the learned latent representations for better representing the connections among nodes in latent space. Experimental results on datasets validate the superiority of our method to the state-of-the-art network embedding methods on the clustering task.

1 Introduction

Graphs are essential tools for mining connections among data. Due to the non-Euclidean topology structure, Graph Convolutional Network (GCN) can better handle the complex graph data where connections information among nodes is unordered and variable in size. For example, analyzing graph data plays an important role in various data mining tasks including node classification [Kipf and Welling, 2016a], link prediction [Pan et al., 2018] and node clustering [Zhang et al., 2019].

Graph neural networks are roughly categorized into four categories: recurrent graph neural networks (RecGNN), convolutional graph neural networks (ConvGNN), graph autoencoders (GAE), and spatial-temporal graph neural networks (STGNN) [Wu et al., 2019a].

Graph autoencoder (GAE) is an effective unsupervised learning framework that encoders the node features and graph structure into the latent representations and decoder the graph structure. GAE can be used to learn network embeddings and graph generative distributions. For network embedding, GAE mainly learns the latent representations by reconstructing graph structure such as the adjacency matrix. And there are many approaches that build GAEs for network embedding, such as DNGR [Cao et al., 2016], SDNE [Wang et al., 2016], ARVGA [Pan et al., 2018]. For graph generation, GAE encodes graphs into latent representations and decodes a graph structure from latent representations to learn the generative distribution of graphs [Li et al., 2018].

Network embedding (NE) methods aim at learning low-dimensional latent representations of nodes in a network. These representations can be used as features for a wide range of tasks on graphs such as classification, clustering and link prediction. The classical methods such as PCA [Wold et al., 1987] [Zhang and Tong, 2019] and Laplacian eigenmaps (LE) [Belkin and Niyogi, 2002] have been widely used in network embedding. GAE can also address the network embedding problems and the main distinction between GAE and network embedding is that GAE is designed for various tasks while network embedding covers various kinds of methods targeting the same task.

Clustering plays an important role in detecting communities and analyzing structures of these networks and is an important task in network embedding. Compared with classical methods such as Spectral Clustering [Ng et al., 2002], CAN [Nie et al., 2014], the network embedding algorithms such as DEC [Xie et al., 2016] perform more effectively and high non-linearity. However, some early network embedding methods such as DeepWalk [Perozzi et al., 2014], node2vec [Grover and Leskovec, 2016] only applicable to unweighted graphs, which restricts those methods’ development on the weighted graph filed.

Some GAE methods such as GAE* [Kipf and Welling, 2016b] only focus on the reconstruction of the graph adjacency matrix while ignoring the reconstruction of node features for network embedding, which may lead to overfitting due to the capacity of the autoencoders. And the properties of the fixed adjacency matrix make these methods cannot properly represent the connections among nodes in latent space. To solve this problem, in this paper, we propose a novel Graph Convolutional Auto-encoder with Bi-decoder and Adaptive-sharing Adjacency method, namely BAGA. The contributions
can be summarized below:

- Equipped with the bi-decoder, our framework can minimize the reconstruction loss of node features and graph structure simultaneously and avoid the overfitting due to the simply reconstruction of the graph adjacency matrix.
- The proposed method can update the adjacency matrix adaptively by latent representations, which result in a clearer graph structure and more accurate connections.
- The embedded adjacency matrix can not only be adaptively updated but also be shared with the Laplacian graph construction. Experimental results support the effectiveness of updating the adjacency matrix.

2 Related Work

In this section, we outline the background of graph autoencoder and network embedding by listing related literature.

2.1 Graph Autoencoder

As an important unsupervised learning framework, Graph autoencoder (GAE) arouses considerable research interests and an elaborate survey can be found in [Wu et al., 2019b]. GAE is a deep neural architecture that maps graph data into a latent space and decoder graph structure information from latent space and decoder graph structure information. Earlier approaches such as DNGR [Cao et al., 2016] and SDNE [Wang et al., 2016] mainly build the GAE frameworks for network embedding by multi-layer perceptrons. And they employ deep autoencoders to preserve the graph proximities and model positive pointwise mutual information (PPMI). Nevertheless, DNGR and SDNE only consider node structure information but ignore the feature information of nodes. Variational Graph Autoencoder (VGAE) [Kipf and Welling, 2016b] is a variational version of GAE to learn the distribution of data.

2.2 Network Embedding

The goal of network embedding is to find latent low-dimensional representations of nodes which preserves the topological information of nodes. Some other methods preserve observed pair-wise similarity and structural equivalence in recently. For example, DeepWalk [Perozzi et al., 2014] uses a random walk to generate sequences of nodes from a network and transform unweighted graph structure information into linear sequences. Inspired by DeepWalk, DRNE [Tu et al., 2018] adopts a Long Short Term Memory (LSTM) network to aggregate a node’s neighbors. Similar to DRNE, NetRA [Yu et al., 2018] 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.

3 Problem and Framework

In this section, we define problem related concepts and propose our framework.

3.1 Problem Formulation

Given a non-directed weighted graph \( G = \{ V, E, X \} \), where \( V = \{ v_1, v_2, \ldots, v_n \} \) is a set of nodes with \( |V| = n \), \( E \) is the connecting edge among each node that can be represented as an adjacency matrix \( A = \{ a_{ij} \} \in \mathbb{R}^{n \times n} \) and \( X \) is the feature matrix of all nodes, i.e., \( X = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{n \times m} \), where \( x_i \in \mathbb{R}^m \) is the feature vector of node \( v_i \), \( n \) is the number of nodes and \( m \) is the dimension of input data. Edge weights in the adjacency matrix \( A \) are real numbers denoting the degree of connection among nodes rather than simple \((0, 1)\) in the unweighted graph. Although edge weights in weighted graphs can be negative, we only consider non-negative weights in this paper. And the sum of weights between a node and the other nodes connected to it is 1.

Our goal is to learn a latent representation \( z_i \in \mathbb{R}^f \) of each node \( v_i \in V \) with the formal format as: \( f : (A, X) \rightarrow Z \), where \( z_i^T \) is the \( i \)-th row of the matrix \( Z \in \mathbb{R}^{n \times f} \) and \( f \) is the dimension of embedding. The \( Z \) is the embedding matrix in latent space which should preserve the topological structure as well as node features information. Furthermore, the latent representations \( Z \) can more accurately represent the connections among nodes for partitioning the nodes of the graph \( G \) into \( c \) clusters \( C = \{ C_1, C_2, \ldots, C_c \} \).

3.2 Overall Framework

Our objective is to learn better latent representations of nodes and more accurate connections in the graph \( G = \{ V, E, X \} \). To this end, we propose a novel graph convolution autoencoder with bi-encoder and adaptive-sharing adjacency method whose adjacency matrix can be adaptively updated. Fig. 1 shows the workflow of BAGA which consists of two three modules: the graph convolutional encoder, bi-decoder and the update of the adjacency matrix \( A \).

- **Graph Convolutional Encoder** The proposed framework takes the adjacency matrix \( A \) and feature matrix \( X \) as input to learn the latent representations \( Z \).
- **Bi-decoder** The framework reconstructs the node features \( X \) and adjacency matrix \( A \) from learned \( Z \) simultaneously which harnesses the overall loss of the network.
- **Adjacency Matrix Update** Since the feature matrix \( X \) becomes better latent representations \( Z \) during the iteration, the adjacency matrix \( A \) also needs to be adaptively updated to \( A^L \) so as to better represent the connections among nodes in latent space. The adaptive update process of the adjacency matrix is also the most important contribution of this paper.

4 Proposed Algorithm

The graph convolutional autoencoder in the framework has a graph convolutional encoder and a bi-decoder which include node features reconstruction and graph structure reconstruction. For better maintain the topology structure, the latent representations are embedded into the Laplacian graph construction.

4.1 Graph Convolutional Encoder Model \( G(X, A) \)

The neural network of BAGA consists of \( (M + 1) \) layers with \( M \) transformations layers and an initial layer, where \( M \) is an even number. The first \( \frac{M}{2} \) hidden layers are the encoders, which learn a layer-wise transformation by a spectral convolution function \( f(\{Z^{(e)}, A\}|W^{(e)})) \):

\[
Z^{(e+1)} = f(\{Z^{(e)}, A\}|W^{(e)}),
\]
where $Z^{(e)}$ is the input for convolution, $Z^{(e+1)}$ is the output after convolution, $e = (1, 2, \ldots, \frac{M}{2})$ is the layer of encoder network. $W^{(e)}$ is the weighted parameter matrix that need to be learned in the neural network. In this paper, $Z^0 = X \in \mathbb{R}^{n \times m}$ is the input node features matrix. Specifically speaking each layer of our graph convolution network can be calculated as follows:

$$f \left( Z^{(e)}, A | W^{(e)} \right) = \phi \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} Z^{(e)} W^{(e)} \right).$$  \hspace{1cm} (2)

Here, $\tilde{A} = A + I$, $\tilde{D}_{ij} = \sum_j \tilde{A}_{ij}$, $I$ is the identity matrix of $A$ and $\phi$ is the activation function such as Relu$(t) = \max(0, t)$ or sigmoid$(t) = \frac{1}{1 + e^{-t}}$. There are two transformations performed alternately in the process of encoder that one is an activation layer with a Relu$(\cdot)$ activation function and the other is a linear layer. The graph encoder is constructed as follows:

$$Z^{(e)} = f_{\text{Relu}} \left( Z^{(e-1)}, A | W^{(e)} \right);$$  

$$Z^{(e+1)} = f_{\text{linear}} \left( Z^{(e)}, A | W^{(e+1)} \right).$$  \hspace{1cm} (3)

And this process encodes both graph structure and node features into the latent representations $Z = \text{encoder}(Z|X, A) = Z^{(\frac{M}{2})}$ which is shared by the encoder and the decoder.

### 4.2 Bi-decoder Model

The last $\frac{M}{2}$ layers are the decoders, which reconstruct both graph structure $A$ and node features $X$ for the learned latent representations. The node features decoder is the mirror symmetry process of the encoder, which also has a linear layer and an activation layer with a Relu$(\cdot)$ function. The node features decoder is constructed as follows:

$$Z^{(d)} = f_{\text{Relu}} \left( Z^{(d-1)}, A | W^{(d)} \right);$$  

$$Z^{(d+1)} = f_{\text{linear}} \left( Z^{(d)}, A | W^{(d+1)} \right).$$  \hspace{1cm} (4)

where $d = (\frac{M}{2} + 1, \frac{M}{2} + 2, \ldots, M)$ is the layer of encoder network. The reconstruction loss of node features is calculated as:

$$\mathcal{L}_x = \| Z^{(0)} - Z^{(M)} \|^2_F,$$  \hspace{1cm} (5)

where $Z^{(M)} = \text{decoder}(\hat{X}|Z, A)$ is the reconstructed node features.

For the graph structure decoder, the reconstructed graph $\hat{A}$ can be presented as follows:

$$\hat{A} = \text{sigmoid}(ZZ^T),$$  \hspace{1cm} (6)

where $Z$ is the embedding latent representations that $Z = \text{encoder}(Z|X, A) = Z^{(\frac{M}{2})}$. The reconstruction loss of the graph structure is calculated as:

$$\mathcal{L}_a = \| \hat{A} - A \|^2_F.$$  \hspace{1cm} (7)

By reconstructing both graph structure $A$ and node features $X$ simultaneously, the framework can exploit both latent node features and graph structure information.

### 4.3 Laplacian Graph Structure and Loss Function

As [Roweis and Saul, 2000] proved, the aforementioned reconstruction criterion can smoothly capture the data man-

---

**Figure 1:** The architecture of the graph convolution autoencoder with bi-encoder and adaptive-sharing adjacency (BAGA). The left tier is a graph convolutional encoder that encodes the input feature matrix $X$ and adjacency matrix $A$ into the latent representations $Z$. The right tier is a bi-decoder that reconstructs the node features $\hat{X}$ and adjacency matrix $\hat{A}$. The learned latent representations $Z$ are in the middle tier where the adjacency matrix $A$ can be updated to $A^L$ for the next iteration (See Algorithm 1 for details).
ifolds and thus preserve the similarity among nodes. Nevertheless, we also want to make the nodes that have similar neighborhood structures have similar structures in latent space. Therefore, we equip our framework with the Laplacian graph structure that shares the same latent representations $Z$ as follows:

$$L_t = \sum_{i,j=1}^{n} \left( \|z_i - z_j\|_2^2 a_{ij} + \gamma a_{ij}^2 \right) = Tr(\mathbf{ZLZ}^T) + \gamma \|\mathbf{A}\|_F^2$$

s.t. $\mathbf{a}_i^T \mathbf{1} = 1, 0 \leq a_{ij} \leq 1,$

Here, $z_i$ is the latent representation of node $x_i$, $a_{ij} \in \mathbb{R}^{n \times 1}$ denotes the $i$-th column vector of adjacency matrix $\mathbf{A}$, $\gamma$ is the regularization parameters. Additionally, it is worth emphasizing that $\gamma$ is not a hyper-parameter but a parameter that can be learned adaptively, which will be proved in the next chapter. And $\mathbf{L}$ is the Laplacian matrix which can be calculated as $\mathbf{L} = \frac{1}{2}(\mathbf{D} - \mathbf{A})$, where $\mathbf{D}$ is the diagonal matrix whose the $i$-th diagonal element is $\sum_j (a_{ij} + a_{ji})/2$. Therefore, the loss of the whole framework can be expressed as:

$$L_G = L_x + L_a + \alpha L_t + \lambda L_{reg},$$

where $L_{reg}$ is a $\ell^2$-norm regularizer term with coefficient $\lambda$ to prevent overfitting, which is defined as follows:

$$L_{reg} = \frac{1}{2} \sum_{i=1}^{M} (\|W(i)\|_F^2).$$

5 Optimization

To our best knowledge, few methods can update the adjacency matrix during the optimization process in the graph convolutional network (GCN). In this section, we introduce the update of $\mathbf{A}$ and optimization of the framework.

5.1 Update The Adjacency Matrix $\mathbf{A}$

Though the element of the adjacency matrix denotes the degree of connection among nodes, the initial value may not be optimal. Since we embed updating adjacency matrix in the optimization, the learned adjacency matrix can represent the more accurate connections among nodes in latent space. In practical applications, sparse adjacency matrix $\mathbf{A}$ tends to bringing 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.

Although there are the adjacency matrix in graph reconstruction loss and node features reconstruction loss, for obtaining a sparse adjacency matrix, we only focus on the Laplacian graph structure when updating the adjacency matrix. The process of updating the adjacency matrix $\mathbf{A}$ is represented as follows:

$$\min_{\mathbf{A}} \sum_{i,j=1}^{n} \left( \|z_i - z_j\|_2^2 a_{ij} + \gamma a_{ij}^2 \right)$$

s.t. $\mathbf{a}_i^T \mathbf{1} = 1, 0 \leq a_{ij} \leq 1.$

Denote the distance $\|z_i - z_j\|_2^2$ between two nodes $z_i$, $z_j$ as $h_{ij}$ and $h_{k} \in \mathbb{R}^{n \times 1}$ is a vector with its $j$-th element $h_{ij}$, and $a_{i} \in \mathbb{R}^{n \times 1}$ is a vector with its $j$-th element $a_{ij}$. The Lagrangian equation of problem (11) is represented as:

$$L(a_i, \eta, \varphi_i) = \frac{1}{2} \left( a_{i}^2 + \frac{h_{ij}}{2\gamma a_{ij}} \right) - \eta (a_{ij} - 1) - \varphi_i a_{ij},$$

where $\eta$ and $\varphi_i \geq 0$ are the Lagrangian multipliers. Using the Karush-Kuhn-Tucker (KKT) conditions, we can derive the optimal solution of $a_{ij}$ as:

$$a_{ij} = \left( \frac{-h_{ij} + \eta}{2\gamma a_{ij}} \right),$$

where $\max(a,0)$ is a sparse similar matrices $a$, when calculating the similarity matrix $\mathbf{A}$, only the $k$ sample points closest to $z_i$ are taken into consideration. Therefore, $a_i$ satisfies $a_{ik} > 0 \geq a_{i,k+1}$, and we have

$$\begin{cases} a_{ik} > 0 \Rightarrow -\frac{h_{ik}}{2\gamma a_{ik}} + \eta > 0 \\ a_{i,k+1} \leq 0 \Rightarrow -\frac{h_{i,k+1}}{2\gamma a_{i,k+1}} + \eta \leq 0 \end{cases},$$

Based on Eq. (14), the overall $\gamma$ is set to the mean of $\gamma_i$ and it can be learned adaptively as:

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

which verified $\gamma$ is not a hyper-parameter. Without loss of generality, suppose $h_{i1}, h_{i2}, \ldots, h_{in}$ are ordered from small to large. The adjacency matrix can be solved as:

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

Based on Eq. (16), the adjacency matrix $\mathbf{A}$ can be updated into $\mathbf{A}^{L}$ by the latent representations for clearer graph structure.

In the experiments, adjacency matrix $\mathbf{A}$ is updated in each iteration where the change of the adjacency matrix $\mathbf{A}$ is showed in Fig. [2]. The adjacency matrix $\mathbf{A}$ on the iteration of 20, 60, 100, 150, 200, 250 are selected to be demonstrated. Since the datasets used in this paper are arranged in cluster order, the higher the degree of diagonalization of the adjacency matrix is, the more accurate connections among nodes. The results in Fig. [2] show the graph construction becomes more and more clear, which verifies the effectiveness of updating the adjacency matrix $\mathbf{A}$ in our method.

5.2 Optimize The Model

To optimize the aforementioned model, the goal is to minimize the overall loss $L_G$ as a function of the neural network parameter $W$. The calculation of the partial derivative of $L_G$ is estimated using the back-propagation. Furthermore, the proposed framework can be optimized by using adaptive moment estimation (Adam), where max_iter is the maximum iteration number. Besides that, the pseudocode of our method is summarized in Algorithm [1].
Algorithm 1 Algorithm to BAGE

Require: $G = \{V, E, X\}$: a Graph with node features $X$ and adjacency matrix $A$; maximum iteration number $max_{iter}$; parameters $k$, $\alpha$, and $\lambda$.

Ensure: latent representations $Z$.
1: for iterator $= 1, 2, 3, \ldots, max_{iter}$ do
2: Minimizing $L_{Q}$ using Eq. (9)
3: Compute the loss of reconstructing node features using Eq. (5)
4: Compute the loss of reconstructing graph structure using Eq. (7)
5: Compute the Laplacian graph structure using Eq. (6)
6: Compute the whole loss using Eq. (9)
7: Backpropagate loss and update $W$
8: Update the adjacency matrix $A$ to $A^{l}$ using Eq. (16)
9: $A \leftarrow A^{l}$
10: end for

6 Experiments

We show experimental results of our proposed method against baseline algorithms on several datasets. The results not only demonstrate the advantage of bi-decoder and adaptive-sharing adjacency but also support the effectiveness of updating the adjacency matrix by latent representations for our method.

6.1 Datasets

Since the proposed method deals with weighted graphs rather than unweighted graphs, we do not use graph datasets such as Cora, Citeseer and PubMed which have been widely used in other graph convolutional networks methods. The datasets used in this paper and the neurons of each layer are listed in Table 1.

6.2 Baselines

We compare our algorithm against several state-of-the-art algorithms for the clustering task on the learned latent representations.

- **GAE** [Kipf and Welling, 2016a] is an autoencoder-based unsupervised framework for graph data, which naturally leverages both topological and content information.

- **ARGA** [Pan et al., 2018] is an adversarially regularized autoencoder algorithm that uses graph autoencoder to learn the embedding.

- **SAE** [Bengio et al., 2007] is a greedy layer-wise unsupervised training strategy consisting of multiple layers of such autoencoders.

- **DEC** [Xie et al., 2016] is a well-known auto-encoder based deep clustering model which simultaneously learns feature representations and cluster assignments using deep neural network.

- **Spectral Clustering (SC)** [Ng et al., 2002] uses the similarity matrix constructed by linear kernel to perform dimensionality reduction before clustering and is widely used in graph clustering.

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

To verify the effectiveness of updating the adjacency matrix in our method, we deliberately make a baseline OURS, NA which has the same loss function as our method except for updating the adjacency matrix $A$.

It is worth explaining that the method which only suitable for unweighted graphs, such as Deep Walk [Perozzi et al., 2014], is not included in the baselines.

#### Table 1: Information of the datasets.

| Dataset | Nodes | Features | Neurons in each layer |
|---------|-------|----------|-----------------------|
| UMIST  | 575   | $32 \times 32$ | 1024-256-80-256-1024 |
| COIL20 | 1440  | $32 \times 32$ | 1024-256-80-256-1024 |
| ORL    | 400   | $32 \times 32$ | 1024-256-80-256-1024 |
| ATT40  | 400   | $32 \times 32$ | 1024-256-80-256-1024 |
| YALEB  | 16128 | $64 \times 64$ | 4096-1024-256-1024-4096 |
| FEI    | 2800  | $80 \times 80$ | 6400-1024-256-1024-6400 |

6.3 Experimental Settings

Since not all baselines rely on the graph structure, we do not take the adjacency matrix as the similarity matrix to perform the classical spectral clustering on the latent representations like [Zhang et al., 2019]. For the fairness of the experiment, we run the $k$-means algorithm on the learned latent representations 10 times with different initializations and report the average result for all baselines.

In terms of parameter settings, all the hyper-parameters settings in our method are listed in Table 3. And for other baselines, the parameters are set to the values that make the best
Table 2: Clustering performance (%).

| Methods   | UMIST ACC | UMIST NMI | COIL20 ACC | COIL20 NMI | YALEB ACC | YALEB NMI | ORL ACC | ORL NMI | ATT40 ACC | ATT40 NMI | FEI ACC | FEI NMI |
|-----------|-----------|-----------|------------|------------|-----------|-----------|---------|---------|-----------|-----------|---------|---------|
| OURS      | 55.99     | 73.66     | 59.81      | 72.32      | 50.30     | 54.23     | 59.45   | 78.22   | 64.02     | 79.93     | 42.30   | 74.11   |
| OURS_NA   | 52.69     | 72.54     | 64.19      | 78.64      | 46.33     | 49.61     | 56.64   | 74.27   | 58.65     | 73.74     | 41.09   | 73.34   |
| GAE       | 47.90     | 65.25     | 60.83      | 74.88      | 45.21     | 51.51     | 59.83   | 78.21   | 56.26     | 75.98     | 37.74   | 71.60   |
| ARGA      | 48.61     | 68.77     | 65.75      | 79.11      | 40.24     | 42.06     | 51.87   | 72.80   | 53.75     | 74.33     | 31.63   | 66.28   |
| SAE       | 41.23     | 60.19     | 55.21      | 70.34      | 40.18     | 42.77     | 48.03   | 65.67   | 52.75     | 73.74     | 31.38   | 66.16   |
| DEC       | 36.47     | 56.96     | 52.33      | 67.13      | 42.30     | 43.52     | 37.45   | 38.93   | 52.63     | 72.64     | 30.52   | 60.38   |
| SC        | 46.59     | 64.39     | 62.34      | 78.47      | 45.81     | 50.58     | 55.97   | 72.82   | 60.30     | 78.31     | 39.43   | 62.19   |
| k-means   | 42.06     | 63.95     | 58.54      | 73.16      | 39.30     | 45.14     | 51.62   | 72.48   | 52.12     | 72.60     | 35.30   | 69.48   |

Table 3: Parameter settings.

| Para. | M   | k   | α   | λ   | lr   | max_iter |
|-------|-----|-----|-----|-----|------|----------|
| Value | 4   | 15  | 0.001 | 0.001 | 1e-4 | 300      |

Table 4: Time comparison (seconds) on COIL20

| Methods   | Ours | ARGA |
|-----------|------|------|
| Running Time | 180.09 | 217.36 |

experimental results. We optimize methods with the Adam algorithm and report the excellent results during the max_iter.

6.4 Experimental Results and Analysis

In the experiment, we run all methods to generate latent representations that are used for the clustering task. The accuracy (ACC) [Papadimitriou and Steiglitz, 1982] and normalized mutual information (NMI) [Fan, 1949] are used as the metric to measure performance. The best result on each dataset has been highlighted and the second-best result is underlined in Table 2.

Fig. 3 shows the partial visualization results on COIL20 and each row corresponds to a cluster. From the result of Fig. 3, our method corresponds to clusters very well, except for confusing cats and cars. And the running time on the same experimental condition for our method and the baseline ARGA is given in Table 4. The observations are as follows:

1) BAGE outperforms the other methods that only exploit either latent node features or graph structure by a very large margin on many datasets, due to the clear reason that BAGE integrates both kinds of information by node features decoder and graph structure decoder, which can complement each other and greatly improve clustering performance.

2) BAGE outperforms the GAE and OURS_NA a very large margin, which verifies the effectiveness of updating the adjacency matrix with latent representations. As shown in Fig. 2, the graph structure of the latent representations has become clearer and connections among nodes become more accurate with the update of adjacency matrix A.

3) The embedded adjacency matrix can not only be adaptively updated but also be shared with the Laplacian graph structure, which embeds the latent representations into the Laplacian graph structure and results in more correct connections.

7 Conclusion

In this paper, we propose a novel bi-decoder and adaptive-sharing adjacency method based on graph convolutional autoencoder for network embedding. We address the case that most existing graph autoencoder algorithms neglect the construction of node features and suffer from the incorrect connections among latent representations due to the fixed adjacency matrix. We proposed an effective way to update the adjacency matrix and enforce the adjacency matrix can more accurately represent the connections among nodes in latent space. The embedded adjacency matrix can not only be adaptively updated but also be shared with the Laplacian graph structure. Furthermore, the unique bi-decoder mechanism harnesses the overall loss of the network and avoids overfitting due to the capacity of the autoencoders. Therefore, the learned latent representations can better represent the information of nodes, which enables our method to achieve competitive performance compared to classical and state-of-the-art methods. Experimental results support the effectiveness of our method by comparing it to other start-of-the-art algorithms.
References

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

[Bengio et al., 2007] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Advances in neural information processing systems, pages 153–160, 2007.

[Cao et al., 2016] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[Fan, 1949] Ky Fan. On a theorem of weyl concerning eigen-values of linear transformations i. Proceedings of the National Academy of Sciences of the United States of America, 35(11):652, 1949.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864. ACM, 2016.

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

[Kipf and Welling, 2016b] Thomas N Kipf and Max Welling. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308, 2016.

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

[Ng et al., 2002] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Advances in neural information processing systems, pages 849–856, 2002.

[Nie et al., 2014] Feiping Nie, Xiaojian 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, pages 977–986. ACM, 2014.

[Pan et al., 2018] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. arXiv preprint arXiv:1802.04407, 2018.

[Papadimitriou and Steiglitz, 1982] Christos H Papadimitriou and Kenneth Steiglitz. Combinatorial optimization, volume 24. Prentice Hall Englewood Cliffs, 1982.

[Perozzi et al., 2014] 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, pages 701–710. ACM, 2014.

[Roweis and Saul, 2000] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. science, 290(5500):2323–2326, 2000.

[Tu et al., 2018] 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, pages 2357–2366. ACM, 2018.

[Wang et al., 2016] 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, pages 1225–1234. ACM, 2016.

[Wold et al., 1987] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3):37–52, 1987.

[Wu et al., 2019a] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596, 2019.

[Wu et al., 2019b] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. arXiv: Learning, 2019.

[Xie et al., 2016] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In International conference on machine learning, pages 478–487, 2016.

[Yu et al., 2018] 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, pages 2663–2671. ACM, 2018.

[Zhang and Tong, 2019] Rui Zhang and Hanghang Tong. Robust principal component analysis with adaptive neighbors. In Advances in Neural Information Processing Systems, pages 6959–6967, 2019.

[Zhang et al., 2019] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. arXiv preprint arXiv:1906.01210, 2019.