mathematics

MDPI

Article

Attribute Graph Embedding Based on Multi-Order Adjacency Views and Attention Mechanisms

Jinfang Sheng, Zili Yang, Bin Wang * and Yu Chen

School of Computer Science and Engineering, Central South University, Changsha 410083, China; jfsheng@csu.edu.cn (J.S.); 214711103@csu.edu.cn (Z.Y.); 214712134@csu.edu.cn (Y.C.)
* Correspondence: wb_csut@csu.edu.cn

Abstract: Graph embedding plays an important role in the analysis and study of typical non-Euclidean data, such as graphs. Graph embedding aims to transform complex graph structures into vector representations for further machine learning or data mining tasks. It helps capture relationships and similarities between nodes, providing better representations for various tasks on graphs. Different orders of neighbors have different impacts on the generation of node embedding vectors. Therefore, this paper proposes a multi-order adjacency view encoder to fuse the feature information of neighbors at different orders. We generate different node views for different orders of neighbor information, consider different orders of neighbor information through different views, and then use attention mechanisms to integrate node embeddings from different views. Finally, we evaluate the effectiveness of our model through downstream tasks on the graph. Experimental results demonstrate that our model achieves improvements in attributed graph clustering and link prediction tasks compared to existing methods, indicating that the generated embedding representations have higher expressiveness.

Keywords: graph embedding; graph representation learning; graph neural networks; graph autoencoder

MSC: 68R10

check for updates

Citation: Sheng, J.; Yang, Z.; Wang, B.; Chen, Y. Attribute Graph Embedding Based on Multi-Order Adjacency Views and Attention Mechanisms. *Mathematics* **2024**, *12*, 697. https://doi.org/10.3390/math12050697

Academic Editors: José F. Vicent, Leandro Tortosa and Manuel Curado

Received: 10 January 2024
Revised: 15 February 2024
Accepted: 22 February 2024
Published: 27 February 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In attribute graphs, nodes possess both topological characteristics and individual node features. Attribute graphs find wide application in various fields, such as social analysis [1–3], recommendation systems [4], and biology [5]. Analyzing attribute graphs holds significant importance in comprehending the network's overall topology, dynamic characteristics, modules, functions, and evolution. Graph embedding, also known as graph representation learning, is a technique that maps nodes in a graph to low-dimensional vectors. This mapping greatly facilitates the analysis of attribute graphs. The challenge in attribute graph embedding lies in effectively integrating and representing the topology and feature information of nodes in the learned low-dimensional vectors.

Recently, graph neural networks have demonstrated exceptional performance in analyzing graph-structured data and are widely used to learn the representations of graphs, particularly GCN (graph convolutional network)-based [2] methods. The key operation in GCN updates a node's features by aggregating its neighbors' features, using the adjacency and feature matrices. This enables information spread across the graph. By layering these operations, GCNs learn the graph's complex patterns, capturing its hierarchical structure. The accuracy of graph embedding has been greatly enhanced by these methods. Presently, the majority of graph embedding techniques that utilize graph neural networks make use of graph auto-encoders [6]. This paper focuses on learning superior low-dimensional vector representations of attribute graphs using a graph auto-encoder. The graph auto-encoder structure utilizes a graph convolutional network (GCN) as the encoder, employs a vector

inner product as the decoder, and employs the cross-entropy loss function between the reconstructed graph and the original graph.

Upon analyzing existing methods, it was observed that most of the mentioned approaches overlook the varying impact of different orders of neighbors on the generation of node embedding vectors. For central nodes in the graph, low-order neighbors exert a significant influence. During the vector generation process, the low-order neighbors of the central node are also closer to the center of the graph, while the high-order neighbors of the central node are located at the periphery. Incorporating information from low-order neighbors enhances the expressiveness of the embedding vector, while fusing high-order peripheral nodes may introduce noise and impact the results.

For peripheral nodes in the graph, high-order neighbors have a greater impact. In the process of generating embedding vectors, peripheral nodes have smaller degrees and can only incorporate limited neighbor information, which makes it challenging for them to learn effective embedding vectors. Therefore, in the embedding process, peripheral nodes need to integrate high-order neighbor information to learn more accurate embedding vectors. Based on this idea, we propose a new encoder called the DGA (Different-Order-View Graph Auto-Encoder), which aims to fuse the feature information of neighbors at different orders. In this paper, we generate different node views for 0th-, 1st-, 2nd-, and 3rd-order neighbor information, considering different orders of neighbor information through different views. We then use graph neural networks to learn node embedding representations in different views. Additionally, this paper primarily utilizes attention mechanisms to integrate node embeddings from different views, as attention mechanisms can effectively differentiate the impact of different views in the generation of low-dimensional vectors. Finally, experiments undertaken demonstrate significant improvements in graph-related tasks. Our main contributions are as follows:

- The encoder model utilizes multi-order neighbor views to construct adjacency matrices. The DGA uses an attention aggregator to effectively assist nodes in the graph in aggregating information from different views;
- A multi-layer perceptron (MLP) decoder and inner product decoder are used simultaneously to decode the adjacency matrix A and feature matrix X. The model is jointly optimized using the reconstruction loss of the adjacency matrix, the self-supervised clustering loss, and the feature matrix reconstruction loss;
- We apply the learned vectors to graph clustering and link prediction tasks using different datasets. Experimental results obtained demonstrate that our model achieves excellent performance on the respective tasks.

2. Related Work

2.1. Graph Embedding

Graph embedding, also known as graph representation learning, focuses on the challenge of mapping nodes from the original graph into a lower-dimensional space. An additional complexity arises from the fact that nodes possess their own unique features, necessitating the integration of both topological and feature information. Traditional approaches have drawn inspiration from techniques such as Laplacian eigenmaps [7] and matrix factorization [8]. When dealing with graphs lacking explicit node features, existing research primarily revolves around methods that leverage the graph's topology as the main source of information. DeepWalk [9] was one of the pioneering methods that sampled node sequences through random walks and utilized the Skip-Gram model to generate the final vector representations. Similar techniques, such as node2vec [5], LINE [10], SDNE [11], and Auto-Encoders (AE) [12], were subsequently developed. However, these methods have a significant drawback: they solely capture the graph's topological structure and do not consider the node features. To incorporate node features, an additional separate module needs to be defined, which introduces extra aggregation steps [13], such as GAT (graph attention network) [14], to dynamically assign importance to neighbors' features in graph data using attention mechanisms, enhancing feature aggregation and learning.

Alternatively, a more effective approach is to seamlessly integrate the node features and topological information during the training process. This challenge is addressed by graph neural networks (GNNs). GNNs, exemplified by models like GCN [2] or GAT [14], define convolutions on non-Euclidean data, such as graphs. By doing so, GNNs can seamlessly and directly integrate the topological information and node features when generating low-dimensional vector representations.

Furthermore, in attribute graph embedding, Hao Wang et al. [15] developed the sparse attributed network embedding (SANE) framework to simultaneously learn the network structure and sparse attribute information, introducing an attention mechanism to adaptively weigh the strength of interactions based on node attribute similarity. In addressing the variation in informativeness between link structures and attributes across nodes, S. Bandyopadhyay et al. [16] proposed an unsupervised node embedding technique that intelligently prioritizes structure or attributes for each node separately, enhancing both structural and attribute embedding. The decoupled network embedding (DCNE) model by Hao Wang et al. [17] (2021) aims at learning node representations by decoupling the learning of the network structure and attributes, addressing sparse attribute issues and achieving robust embeddings. Hongyang Gao and Shuiwang Ji [18] (2019) proposed Graph U-Nets, integrating attention mechanisms with graph pooling and unpooling operations, to better capture graph topology information and improve the performance on node and graph classification tasks.

To incorporate multiple types of information and optimize downstream tasks, our proposed method builds upon these previous works. Our method utilizes multi-order neighbor views to construct adjacency matrices and attention mechanisms and employs optimization in a joint manner for the loss function

2.2. Graph Auto-Encoder

Graph auto-encoders (GAEs) apply a graph neural network into an autoencoder framework, enabling them to perform unsupervised learning tasks. GAEs aim to learn a lower-dimensional representation of the graph, allowing them to reconstruct the original graph structure or feature representations via a decoder. Variational GAE (VGAE) [6] is a variational version of GAE to learn the distribution of data. The typical model uses a stacked denoising auto-encoder [19] to reconstruct the positive pointwise mutual information matrix to capture the correlation of node pairs. These methods can be classified into three distinct categories, each focusing on enhancing a specific aspect of the graph autoencoder. The first category comprises methods such as DAEGC [20], GEC-CSD [21], and MRFasGCN [22], which aim to improve the performance of the encoder. The second category includes techniques like TGA/TVGA [23], which utilize a triadic decoder to enhance the decoding process. Lastly, there are approaches that go beyond the loss function, incorporating additional enhancements, such as a clustering loss optimizer [24] or a modularity optimizer [25], to further refine the overall performance.

In the improvement of encoders, while GCN, DAEGC, and GEC-CSD apply GAT to the encoder, GUCD [26] utilizes MRFasGCN as its encoder, which is specifically designed based on the concept of a Markov random field. MRFasGCN introduces a novel convolution layer that focuses on community detection tasks. In our research, we found that the existing encoder fails to fully consider the influence of different order views during the process of learning low-dimensional vectors. Therefore, we have taken this impact into consideration when designing the encoder for our model.

3. Method

In this section, we primarily introduce the different view of order encoder (DGA), which is a graph auto-encoder based on multi-order neighbor views. In the encoder, the model aggregates graph convolution results from different order views to generate low-dimensional embedding vectors. In the decoder, the model simultaneously decodes

the adjacency matrix A and feature matrix X while constructing a joint loss function for both to optimize the model. The overall structure of the model is shown in Figure 1.

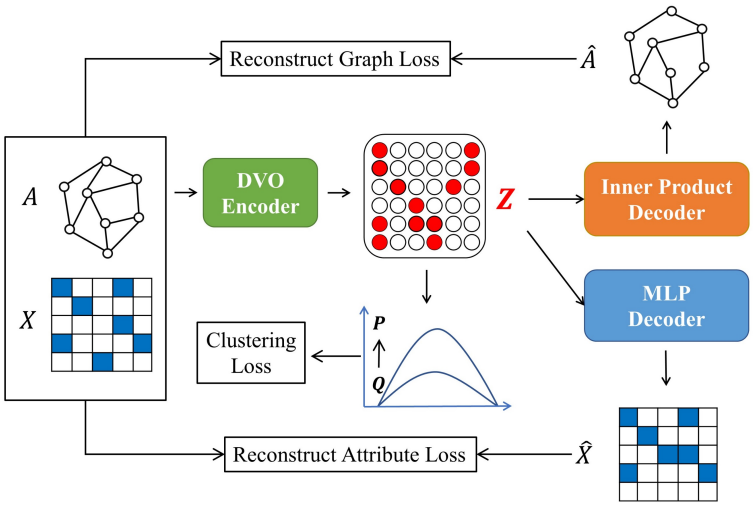


Figure 1. Overall structure of the DGA. The green module represents the encoder, which is the different view of order encoder (DVO encoder) that handles multi-order neighbor views. The orange module represents the inner product decoder, and the blue part corresponds to the MLP feature decoder. The model is optimized using a joint loss that includes the reconstruction loss for the adjacency matrix A, the reconstruction loss for the feature matrix X, and the self-supervised clustering loss.

3.1. Encoder

This section introduces the encoder called the different view of order encoder (DVO encoder), which utilizes multi-order neighbor views. The encoder learns low-dimensional embedding vectors using graph convolutional networks (GCN) under 0th-, 1st-, 2nd-, and 3rd-order neighbor views. Subsequently, the vectors from different views are aggregated using an aggregator to obtain the final embedding representation. The structure of the encoder is shown in Figure 2.

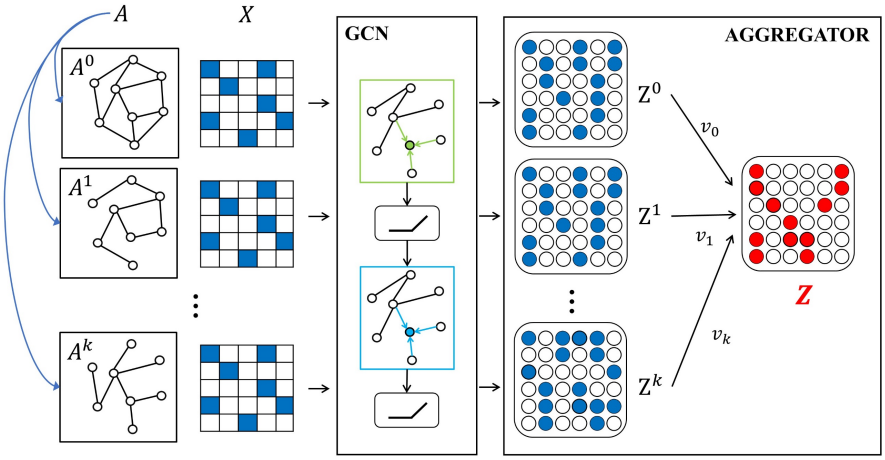


Figure 2. Multi-order neighbor views encoder structure diagram.

3.1.1. Building Multi-Neighbor Views

Before learning the embedding vectors in the encoder, we first construct the 0th-, 1st-, 2nd-, and 3rd-order neighbor views.

The 0th-order view considers only the information X of the nodes themselves, ignoring the information from the attribute adjacency matrix A. The adjacency matrix A_0 under the

0th-order view is constructed based on the correlation between the features x_i of each node and the features x_i of the other nodes, as shown in Formula (1):

$$A^0 \in \mathbb{R}^{n\times n}, \alpha_{ij} = \begin{cases} 1, \text{if } \cos(x_i, x_j) >= \theta \\ 0, \text{else} \end{cases} \tag{1}$$

where cos represents the cosine similarity and θ is the similarity threshold, which is a hyperparameter of the model. By truncating the similarity values based on the threshold, the values in the matrix are set to either 0 or 1, constructing the adjacency matrix under the 0th-order neighbor view. This facilitates subsequent symmetric normalization and enables input to the graph convolutional network (GCN) for training. as shown in Formula (2):

$$\cos(a, b) = \frac{a \bullet b}{\|a\| \times \|b\|} = \frac{\sum_{i=1}^{n}(a_i \times b_i)}{\sqrt{\sum_{i=1}^{n}(a_i)^2} \times \sqrt{\sum_{i=1}^{n}(b_i)^2}} \tag{2}$$

The 1st-, 2nd-, and 3rd-order neighbor views correspond to the 1st-order adjacency matrix, the 2nd-order adjacency matrix, and the 3rd-order adjacency matrix of the graph, respectively. In the k-th-order neighbor view, the neighbor matrix is represented as shown in Formula (3):

$$A^k \in \mathbb{R}^{n\times n}, a_{ij} = \begin{cases} 1, & \text{if } v_j \in N_i^k \\ 0, & \text{else} \end{cases} \tag{3}$$

where N_i^k represents the k-th-order neighbor set of node i.

3.1.2. Learning Embedding Vectors in a Multi-Neighbor Views

After constructing multiple neighbor views, the encoder of DGA utilizes a two-layer graph convolutional network (GCN) to learn the embedding representations for different views. In the two GCN layers, the first layer applies the ReLU activation function, while the second layer does not use any activation function. The learned vectors, representing different views, are denoted as Z^k , and their calculation is as shown in Formula (4):

$$\begin{aligned} Z_1^k &= f_1\left(A^k, X\right) = \text{ReLU}\left(\tilde{D}_k^{\frac{1}{2}} \tilde{A}^k \tilde{D}_k^{\frac{1}{2}}\right) XW^{(0)} \\ Z^k &= f_2\left(A^k, Z_1^k\right) = \tilde{D}_k^{\frac{1}{2}} \tilde{A}^k \tilde{D}_k^{\frac{1}{2}} Z_1^k W^{(1)} \end{aligned} \tag{4}$$

In this case, the two-layer GCN network shares the parameters $W^{(0)}$ and $W^{(1)}$ across different views. $\tilde{A}^k = A^k + I_n$,I_n is an identity matrix with the same dimension as A^k. $\tilde{D}^k$ is a diagonal matrix obtained by calculating the diagonal elements .$\tilde{A}^k$, as shown in Formula (5):

$$\tilde{d}_{ij}^k = \sum_j \tilde{a}_{ij}^k \tag{5}$$

3.1.3. Fusion of Embedding Vectors in a Multi-Neighbor Views

The encoder utilizes an aggregator to fuse the embedding vectors from multiple neighbor views and learns the final embedding representation Z, as shown in Formula (6):

$$Z = AGGREGATOR\left(Z^0, Z^1, \ldots, Z^k\right) \tag{6}$$

$AGGREGATOR$ is the aggregator, which primarily combines the learned embedding vectors from different views and generates the final embedding vector.

3.1.4. Attention Aggregator

DGA utilizes an attention aggregator to combine information from different views. The attention aggregator uses different weights to merge the embedding vectors from

different views. Firstly, the attention aggregator compresses the embedding vectors of each node in each view to one dimension through linear transformations. Then, it adds non-linearity through an activation function and learns the weights for different views using the softmax function, as shown in Formula (7):

$$v_i^k = \frac{\exp\left(\sigma\left(w_i^k z_i^k + b\right)\right)}{\sum_{j=0}^{k} \exp\left(\sigma\left(w_j^k z_j^k + b\right)\right)} \tag{7}$$

v_i^k represents the weight learned for node i with respect to view k. A larger v_i^k indicates a stronger influence of view k on the generation of the embedding vector for node i. σ represents the activation function tanh. For the embedding vectors from different views, the final embedding vector Z is learned by weighted summation using the learned weights, as shown in Formula (8):

$$z_i = \sum_{i=0}^{k} v_i^k z_i^k \tag{8}$$

3.2. Decoder

3.2.1. Attention Aggregator

This paper introduces a simplified approach to the decoder, which consists of two main components: reconstructing the graph and reconstructing the features. To enhance the efficiency of the model, we adopt a straightforward inner product decoder for graph reconstruction, as illustrated in Formula (9):

$$\hat{A}_{ij} = \text{sigmod}(z_i^T, z_j) \tag{9}$$

$\hat{A}_{ij}$ represents the reconstruction adjacency matrix.

3.2.2. Feature Decoder

DGA utilizes a multi-layer perceptron (MLP) to decode the feature matrix X. The model performs decoding through two fully connected layers, as shown in Formula (10):

$$\hat{X} = W_1^d \left[\sigma\left(W_2^d Z+\right) + b_1^d \right] + b_2^d \tag{10}$$

where $\hat{X}$ is the decoded feature matrix, and W_1^d, W_2^d, b_1^d and b_2^d are the learnable parameters and biases of the MLP decoder. The decoding results of different feature decoders can have varying impacts on the computation of the final loss function. This paper evaluates the effects of different decoding approaches on the embedding vectors and downstream graph tasks.

Another decoding approach involves using a GCN decoder. The GCN decoder leverages the reconstructed adjacency matrix $\hat{A}$ obtained from the inner product decoder and utilizes $\hat{A}$ and the embedding vectors Z as inputs to decode X using the GCN decoder. The GCN decoder also consists of two layers, as shown in Formula (11):

$$\hat{X} = \tilde{D}^{-\frac{1}{2}} \tilde{A}^d \tilde{D}^{-\frac{1}{2}} \left(\sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A}^d \tilde{D}^{-\frac{1}{2}}) Z W_1^d \right) W_2^d \tag{11}$$

where $\tilde{D}$ and $\tilde{A}^d$ are computed in the same way as in the encoder. W_1^d and W_2^d are the weight matrices of the two GCN layers, and σ represents the ReLU activation function.

3.3. Loss Function

The optimization objective of DGA is to reconstruct the adjacency matrix A and the feature matrix X. DGA jointly optimizes the model using these two objectives simulta-

neously. Additionally, in the node clustering task, DGA incorporates a self-supervised clustering loss to further optimize the model.

3.3.1. The Loss Function for Reconstructing the Adjacency Matrix A

The loss function for reconstructing the adjacency matrix A in DGA is the cross-entropy between the reconstructed graph structure $\hat{A}$ and the original adjacency matrix A, as shown in Formula (12):

$$L_A = -\sum_{i,j=1}^{n} \left[a_{ij} \mathrm{In}\left(\hat{a}_{ij}\right) + (1 - a_{ij}) \mathrm{In}(1 - \hat{a}_{ij}) \right] \tag{12}$$

3.3.2. The Loss Function for Reconstructing the Attribute Matrix X

DGA utilizes the mean squared error (MSE) between the reconstructed feature matrix X and the original feature matrix $\hat{X}$ as the loss function, as shown in Formula (13):

$$L_X = \frac{1}{n} \left\| X - \hat{X} \right\|^2 \tag{13}$$

3.3.3. The Self-Supervised Clustering Loss Function

DGA introduces a self-supervised clustering module in the process of the node clustering task. The self-supervised module utilizes high-confidence samples for supervision and helps improve low-confidence samples, resulting in denser sample distributions within each cluster. The self-supervised module takes the embedding vectors learned by the encoder as input to the self-supervised clustering objective, aiming to minimize the clustering objective, as shown in Formula (14):

$$L_c = KL(P \parallel Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{14}$$

where $KL(\cdot)$ represents the Kullback–Leibler divergence between the P-distribution and the Q-distribution. Q utilizes the Student's t distribution to measure the distance distribution of each node to the cluster centers, which can be considered as a "soft clustering distribution"; P represents the auxiliary distribution.

3.3.4. Joint Optimization

The optimization of the DGA model involves jointly optimizing the aforementioned loss functions. The overall loss function of the model is as shown in Formula (15):

$$L = \lambda_1 L_A + \lambda_2 L_X + \lambda_3 L_C \tag{15}$$

where λ_1 represents the weight of the reconstruction loss for the adjacency matrix, denoted as L_A. λ_2 represents the weight of the reconstruction loss for the feature matrix, denoted as L_X. λ_3 represents the weight of the self-supervised clustering loss, denoted as L_C.

4. Experiments

In this section, we evaluate the accuracy of our model in different graph tasks through experiments. We introduce the datasets used, the methods to be compared, the settings of the experimental parameters, and the evaluation metrics employed to assess the model's accuracy.

In this paper, two graph downstream tasks are addressed: graph clustering and link prediction.

Graph clustering aims to partition the nodes in a graph into non-overlapping categories, denoted as $C = \{C_1, C_2, \cdots, C_k\}$, based on their topological and feature information. Each category C_k is a subgraph of G, and they have no overlapping nodes. Moreover, it is important to note that $C_{k_a} \cap C_{k_b} = \varnothing(\forall a, b)$.

The link prediction task focuses on assessing the potential of establishing new connections between two nodes in a network that are currently not connected, based on the existing topological structure and feature information of the nodes.

These tasks involve leveraging the graph's topological and feature information to perform clustering or predict the likelihood of creating new connections, providing insights into the graph structure and potential relationships.

4.1. Datasets

We use widely used network datasets that include both the topological and feature structures of the nodes. Our study leverages three pivotal datasets: Cora, Citeseer, and Pubmed [27]. Cora features 5429 scientific publications categorized into seven classes, linked by citations, each represented as a binary word vector from a unique term dictionary. Citeseer includes 3327 articles across six categories, highlighting class imbalance and sparse linkages. Pubmed offers 19,717 biomedical papers, emphasizing dense citation networks and detailed term representations. These datasets provide a comprehensive basis for evaluating the efficacy of graph representation learning algorithms across diverse academic domains.

We conducted experiments on these three datasets for both graph clustering and link prediction tasks. The detailed information about these datasets is provided in Table 1.

Table 1. Summary of datasets.

Dataset	Nodes	Edges	Features	Classes
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19,717	44,338	500	3

Note: The Cora, Citeseer, and Pubmed datasets utilized in this table are sourced from the Laboratory for Artificial Intelligence and Network Analysis (LINQS) at the University of California, Santa Cruz. These datasets can be accessed via the following link: https://linqs-data.soe.ucsc.edu/public (accessed on 20 February 2024).

4.2. Baseline Methods

In the comparison of methods for node clustering and link prediction experiments, the methods can be categorized into the following classes:

(1) Classical Methods:
- DeepWalk [9]: This is an unsupervised graph embedding algorithm based on random walks, used to map nodes to vector representations in a low-dimensional space.
- Spectral Clustering (SC) [28]: This performs clustering by utilizing the eigenvectors of the similarity matrix to group nodes.

(2) Classical Graph Autoencoder Methods:
- GAE: This uses graph convolutional networks (GCNs) as the encoder and is a classical graph autoencoder used for unsupervised tasks.
- VGAE [6]: This is a variational version of GAE.
- ARGA [29]: This uses an adversarial learning framework to generate node embeddings by adversarial regularization. It incorporates both graph structure and node features into the embeddings. ARVGA is the variational version of ARGA.

(3) Recent Methods:
- ARGAX: [1] This extends ARGA by using GCN as the decoder to simultaneously reconstruct the graph structure and node features.
- ARVGAX [1]: This is the variational version of ARGAX.
- DBGAN [30]: This utilizes a distribution-induced bidirectional generative adversarial network, where the prior distribution is estimated through structure-aware means instead of assuming a normal distribution.

- SSGC [31]: This improves graph convolutional networks (GCNs) with an enhanced Markov diffusion kernel that balances the influences of low-pass and high-pass filters to fuse information from low-order and high-order neighboring nodes.
- R-GAE [32]: This avoids clustering noise through sampling operations and improves the GAE model by transforming the reconstruction graph step into a graph structure relevant to the reconstruction and clustering tasks.
- R-VGAE [32]: This applies the same idea to improve the VGAE model.
- DGAE [33]: This extends GAE by using GCN as the feature decoder.
- GNAE [34]: This resolves the issue of zero embeddings for isolated nodes in the graph by using $L2$ normalization.
- VGNAE: [34] This is the variational version of GNAE.

In the link prediction experiments, based on the different decoders used in DGA, three methods are considered:

- DGA: This uses an MLP feature decoder.
- DGA_EX (DGA Exclude X-decoder): This does not use a feature decoder.
- DGA_GX (DGA GCN X-decoder): This uses a GCN feature decoder.

4.3. Evaluation Metrics

Evaluation Methods: For the link prediction task, we adopt the evaluation methods described in the GAE and VGAE papers. We use two evaluation metrics, namely, the area under the curve (AUC) and the average precision (AP) to measure the performance of the link prediction task.

AUC is a metric that quantifies the overall performance of a binary classification model by considering the trade-off between the true positive rate and the false positive rate across various threshold settings, as shown in Formula (16):

$$AUC = \frac{\sum_k I(P_x, P_y)}{N \times M}, I(P_x, P_y) = \begin{cases} 1, & P_x > P_y \\ 0.5, & P_x = P_y \\ 0, & P_x < P_y \end{cases} \tag{16}$$

The average precision calculation is as shown in Formula (17):

$$AP = \frac{\sum_k \text{Precision}}{N \times M}, \text{Precision} = \frac{TP}{TP + FP} \tag{17}$$

For the clustering task, we utilize the following evaluation metrics to assess the performance of the models:

Accuracy (ACC): This measures the accuracy of the clustering results by comparing the predicted labels with the ground truth labels, as shown in Formula (18):

$$\text{ACC} = \frac{\sum_{j=1}^{n} \delta\left(g_i(j), o'(j)\right)}{n}, \delta(x, y) = \begin{cases} 1, & \text{if x=y} \\ 0, & \text{else} \end{cases} \tag{18}$$

Normalized Mutual Information (NMI): This quantifies the similarity between the predicted clustering and the ground truth clustering, taking into account the information entropy, as shown in Formula (19):

$$NM(U, V) = 2\frac{MI(U, V)}{H(U) + H(V)} \tag{19}$$

Adjusted Rand Index (ARI) [35]: This measures the similarity between two clusterings, considering all pairs of samples and taking into account the agreement between the predicted clustering and the ground truth clustering while considering the possibility of chance agreement, as shown in Formula (20):

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \tag{20}$$

4.4. Parameter Settings

For the comparative methods, we select the best-performing configurations for each respective task as the basis for comparison. For our method, in attribute graph clustering, across all datasets, the hidden layer dimension in the encoder structure is set to 256, and the embedding layer dimension is set to 32. In link prediction, to ensure comparability with other methods, we adopt the same dimensional configuration as GAE, with a hidden layer dimension of 32 and an embedding layer dimension of 16.

For the loss functions, to optimize the three losses on the same scale, we set them to 1, 100, and 10, respectively. In the clustering task, we apply [36] to the learned embedding vectors for clustering. The entire model is optimized using the Adam optimizer [37].

4.5. Experimental Results

We mainly evaluate the performance of the model in two graph tasks: node clustering and link prediction, corresponding to Experiment 1 and Experiment 2, respectively. In addition, in Experiment 3, we evaluate the performance of attribute graph clustering with different embedding dimensions, aiming to assess the effect of the embedding dimensions on attribute graph clustering. Finally, we give a visualization of the self-supervised clustering process

4.5.1. Node Clustering Results

The experimental results for node clustering are shown in Table 2. The performance of clustering is evaluated using three metrics: ACC, NMI, and ARI. The underlined values indicate the best-performing method within each category, while the bold values represent the best-performing method among all methods.

In the node clustering experiments, on the Cora dataset, compared to the best-performing comparative method, DGA achieved improvements of 2.54% in ACC, 1.74% in NMI, and 8.70% in ARI. On the Citeseer dataset, DGA achieved improvements of 1.45% in ACC, 3.04% in NMI, and 2.70% in ARI.

Table 2. Node clustering results on the Cora, Citeseer and Pubmed datasets.

Methods	Cora			Citeseer			Pubmed		
	ACC	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**	**ACC**	**NMI**	**ARI**
SC	0.367	0.127	0.031	0.239	0.056	0.010	0.403	0.042	0.002
DeepWalk	0.529	0.384	0.291	0.390	0.131	0.137	0.684	0.279	0.299
GAE	0.611	0.482	0.302	0.456	0.221	0.191	0.632	0.249	0.246
VGAE	0.592	0.408	0.347	0.467	0.261	0.206	0.619	0.216	0.201
ARGA	0.640	0.449	0.352	0.573	0.350	0.341	0.681	0.276	0.291
ARVGA	0.638	0.450	0.374	0.544	0.261	0.245	0.513	0.117	0.078
ARGA-AX	0.597	0.455	0.366	0.547	0.263	0.243	0.637	0.245	0.226
ARVGA-AX	0.711	0.526	0.495	0.581	0.338	0.301	0.640	0.239	0.226
DBGAN	0.748	0.576	0.540	0.670	0.407	0.414	0.694	0.324	0.327
SSGC	0.696	0.547	0.474	0.688	0.428	0.445	0.710	0.332	0.346
R-GAE	0.658	0.516	0.441	0.501	0.246	0.200	0.696	0.314	0.316
R-VGAE	0.713	0.498	0.480	0.449	0.199	0.125	0.692	0.303	0.309
DGA	**0.767**	**0.586**	**0.587**	**0.698**	**0.441**	**0.457**	0.672	0.263	0.267

4.5.2. Link Prediction Results

The experimental results for link prediction are shown in Table 3.

Table 3. Link prediction results on the Cora, Citeseer, and Pubmed datasets.

Methods	Cora		Citeseer		Pubmed	
	AUC	**AP**	**AUC**	**AP**	**AUC**	**AP**
SC	0.846	0.885	0.805	0.850	0.842	0.878
DeepWalk	0.831	0.850	0.805	0.836	0.844	0.841
GAE	0.910	0.920	0.895	0.899	0.964	0.965
VGAE	0.914	0.926	0.908	0.920	0.944	0.947
ARGA	0.924	0.932	0.919	0.930	0.968	0.971
ARVGA	0.924	0.926	0.924	0.930	0.965	0.968
ARGA-AX	0.913	0.913	0.919	0.934	0.966	0.967
ARVGA-AX	0.902	0.892	0.898	0.904	0.967	0.971
DBGAN	0.945	0.951	0.945	0.958	0.968	0.973
DGAE	0.949	0.955	0.949	0.958	0.974	0.972
VGNAE	0.954	0.958	0.970	0.971	**0.976**	**0.976**
GNAE	0.956	0.957	0.965	0.970	0.975	0.975
DGA	0.919	0.913	**0.980**	**0.981**	0.959	0.952
DGA_GX	0.953	0.952	0.970	0.971	0.963	0.957
DGA_EX	**0.961**	**0.962**	0.961	0.962	0.965	0.959

Note: DeepWalk was conducted using the original code from DeepWalk (https://github.com/phanein/deepwalk (accessed on 20 February 2024)). Spectral clustering results were adopted from "Graph convolutional autoencoders with co-learning of graph structure and node attributes" [38].

The underlined values indicate the best-performing method within each category, while the bold values represent the best-performing method among all methods.

In the link prediction experiments, on the Cora dataset, compared to the best-performing comparative method, DGA_EX achieved improvements of 0.523% in AUC and 0.417% in AP. On the Citeseer dataset, DGA achieved improvements of 1.03% in both AUC and AP.

4.5.3. Experiment on Influence of Embedding Size

In this experiment, we explored the impact of the embedding vector dimensions through attribute graph clustering. We conducted attribute graph clustering experiments with embedding dimensions of 4, 16, 32, 64, and 256, aiming to evaluate the influence of the embedding vector dimensions on clustering performance. We performed these experiments on three datasets and compared the changes in metrics, such as ACC, NMI, ARI, and Macro-F1 (F1-score). The experimental results are shown in Figure 3.

The trends of the node clustering metrics for the DGA model on the Cora and Citeseer datasets with varying embedding dimensions are depicted in Figure 3.

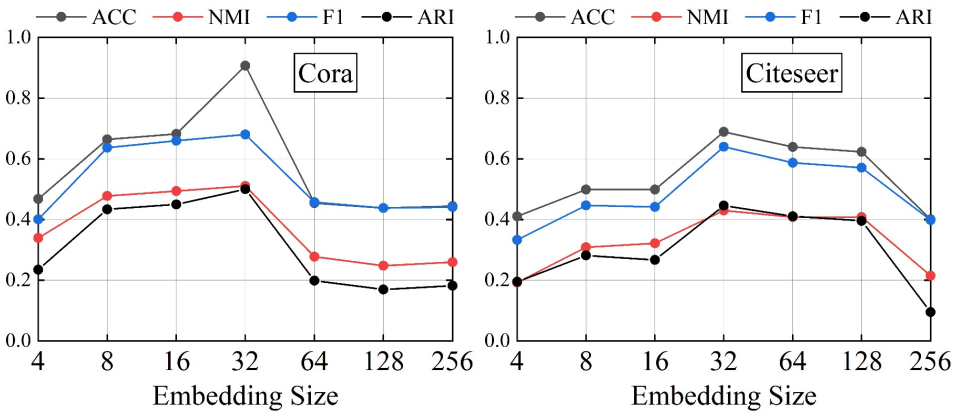


Figure 3. Influence of embedding size. (**a**) Impact of embedding dimensions on four metrics on the Cora dataset. (**b**) Impact of embedding dimensions on four metrics on the Citeseer dataset.

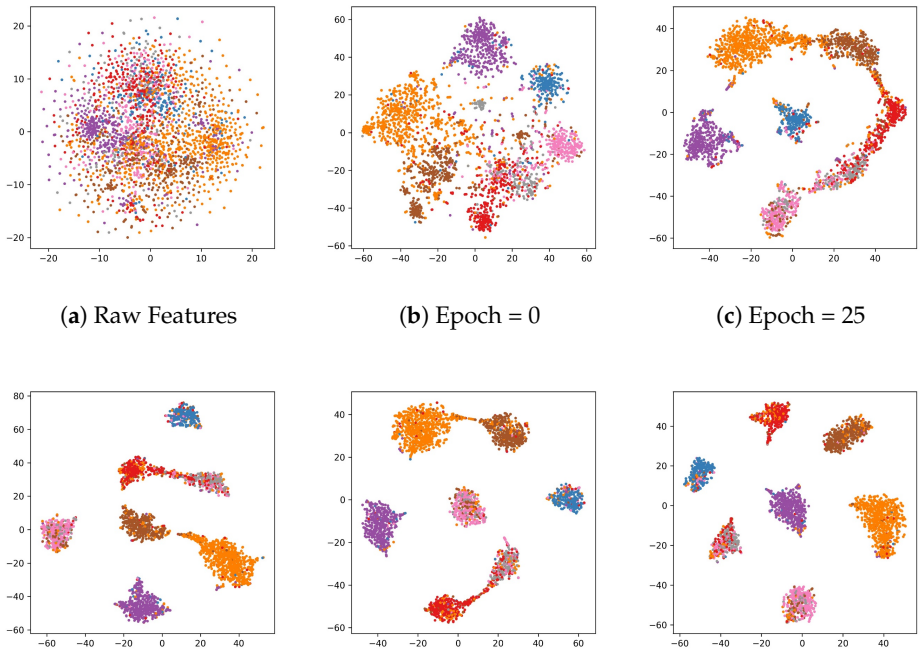
In the node clustering experiments, we can observe from the trend of the curves that the evaluation metrics initially increase and then decrease as the embedding dimensions of the nodes increase. The best performance is achieved at dimension 32, where the metrics attain their highest values. This trend is consistent across all four evaluation metrics used to assess the node clustering results on the Cora and Citeseer datasets.

4.5.4. Visualization

Through visualization, the impact of the self-supervised module on the results can be intuitively observed, thereby visually representing the effectiveness of the self-supervised module.

The experiment visualizes the raw features, and then visualizes the embedding vector results Z obtained step-by-step by the DGA model using the self-supervised clustering module on the Cora dataset. The experiment uses the t-SNE [39] algorithm to visualize the vectors in a two-dimensional space.

The visual results of the experiment comparing the original features of Cora and the self-supervised model training epochs at 0, 25, 50, 75, and 100 are shown in Figure 4. The visualization of the entire self-supervised clustering process is depicted in the figure. The seven categories in Cora are represented by seven different colors. At Epoch = 0, there is some color overlap among the seven categories in Cora. As the training epochs progress, the category boundaries of the seven classes become increasingly clear and eventually exhibit the most distinct clustering characteristics.



(**a**) Raw Features (**b**) Epoch = 0 (**c**) Epoch = 25

(**d**) Epoch = 50 (**e**) Epoch = 75 (**f**) Epoch = 100

Figure 4. 2D visualization results on Cora.

5. Discussion

The DGA model generates expressive embedding vectors and performs well in the unsupervised downstream tasks on graphs. In node clustering and link prediction, the DGA model achieves improvements in three metrics compared to traditional graph autoencoder methods and recent approaches on the Cora and Citeseer datasets, demonstrating the superiority of the DGA model in graph representation learning tasks.

In the link prediction experiment, DGA_EX utilizes a multi-neighborhood view encoder, and on the Cora dataset, DGA_EX outperforms other methods in terms of the metrics. In conclusion, the use of the multi-neighborhood view encoder enhances the effectiveness

of the model and allows for the learning of more efficient embedding vectors compared to the GCN encoder.

Noise Implications and Accuracy Concerns of Multi-Order Neighbor Views

The use of multi-order neighbor views has the potential to introduce noise into the embedding process, which may result in decreased accuracy on certain datasets. Specifically, by incorporating multi-order neighbor views, the scope of the considered neighbors is expanded to include more distant nodes as part of the current node's context. While this approach can capture broader structural information of the graph, it may also introduce information from nodes that have weak or irrelevant relationships with the current node. In such cases, the embedding vectors could be disrupted by unrelated information, thereby diminishing the model's ability to capture key structural features and affecting the performance on downstream tasks. To address this issue, we can consider the following strategies:

- Weight Adjustment: Introduce different weights for neighbors of different orders to reduce the influence of more distant neighbors. This ensures that information from immediate neighbors is prioritized while still retaining some level of distant structural information.
- Attention Mechanisms: Utilize technologies such as graph attention networks (GATs) to dynamically assign different attention weights to different neighbors of a node. Through this method, the model can adaptively focus on more relevant neighbors, thereby minimizing the impact of noisy data.
- Neighbor Selection: Employ a strategy for constructing multi-order neighbor views that only selects nodes highly relevant or contributory to the current node. This can be achieved through preliminary feature selection or task-based backpropagation.
- Regularization Techniques: Incorporate regularization terms, such as graph regularization, to penalize the contributions of neighbors excessively distant from the central node, thus avoiding over-reliance on remote information.

6. Conclusions

In this paper, a graph auto-encoder called DGA based on multi-order neighbor views is proposed and applied to graph representation learning. In the DGA model, a multi-order neighbor views encoder is used. The adjacency matrix A and feature matrix X are decoded simultaneously using a multi-layer perceptron (MLP) decoder and an inner product decoder. The model is jointly optimized using the reconstruction loss of the adjacency matrix, the self-supervised clustering loss, and the feature matrix reconstruction loss. This allows the model to learn low-dimensional embedding vectors for nodes in an unsupervised manner, which can be used for downstream graph tasks.

The experimental results obtained on the datasets demonstrate that the learned embedding vectors have stronger expressiveness, and DGA outperforms the comparison methods, highlighting the superiority of the DGA model in graph representation learning tasks.

Advantages of Multi-Order Neighbor Views in Diverse Graph Applications

Certain applications or topological conditions indeed exist where multi-order neighbor views might be more advantageous than other views.

- Heterogeneous Networks: In networks with diverse types of nodes and edges, multi-order neighbor views can better capture the complex interactions and relationships between different entity types, providing a richer representation of the network.
- Graphs with Long-range Dependencies: For applications where long-range dependencies between nodes are crucial, such as in citation networks or knowledge graphs, multi-order neighbor views can effectively capture these distant relationships, which might be missed by focusing solely on immediate neighbors.
- Community Detection and Clustering: In scenarios where identifying communities or clusters within the graph is essential, multi-order neighbor views can provide insights

into the broader community structure, helping to identify not just local but also global community memberships.
- Graphs with Sparse Connections: For graphs with sparse connections, multi-order neighbor views can help in identifying relevant connections that are not immediately apparent, aiding in tasks like link prediction by providing a wider context.
- Dynamic Networks: In dynamic networks where the topology changes over time, multi-order neighbor views can provide a more stable representation by capturing relationships that persist over multiple scales, offering resilience against temporal variations.

Author Contributions: Conceptualization, J.S. and B.W.; methodology, Z.Y.; validation, Z.Y. and Y.C.; formal analysis, Z.Y. and J.S.; writing—original draft preparation, Y.C.; writing—review and editing, J.S., Z.Y., and B.W.; visualization, Z.Y.; supervision, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Key Research and Development Program of Hunan Province (Grant No. 2023SK2038).

Data Availability Statement: These datasets can be accessed via the following link: https://linqs-data.soe.ucsc.edu/public and https://linqs-data.soe.ucsc.edu/public/lbc, accessed on 20 February 2024.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pan, S.; Hu, R.; Fung, S.f.; Long, G.; Jiang, J.; Zhang, C. Learning graph embedding with adversarial training methods. *IEEE Trans. Cybern.* **2019**, *50*, 2475–2487. [CrossRef] [PubMed]
2. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
3. Yu, F.; Liu, Q.; Wu, S.; Wang, L.; Tan, T. Attention-based convolutional approach for misinformation identification from massive and noisy microblog posts. *Comput. Secur.* **2019**, *83*, 106–121. [CrossRef]
4. Hastings, M.B. Community detection as an inference problem. *Phys. Rev. E* **2006**, *74*, 035102. [CrossRef] [PubMed]
5. Grover, A.; Leskovec, J. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
6. Kipf, T.N.; Welling, M. Variational graph auto-encoders. *arXiv* **2016**, arXiv:1611.07308.
7. Newman, M.E. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* **2006**, *74*, 036104. [CrossRef] [PubMed]
8. Cao, S.; Lu, W.; Xu, Q. Grarep: Learning graph representations with global structural information. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 18–23 October 2015; pp. 891–900.
9. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
10. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1067–1077.
11. Wang, D.; Cui, P.; Zhu, W. Structural deep network embedding. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1225–1234.
12. Chen, C.; Lu, H.; Wei, H.; Geng, X. Deep subspace image clustering network with self-expression and self-supervision. *Appl. Intell.* **2023**, *53*, 4859–4873. [CrossRef]
13. Wang, J.; Huang, P.; Zhao, H.; Zhang, Z.; Zhao, B.; Lee, D.L. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 839–848.
14. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
15. Wang, H.; Chen, E.; Liu, Q.; Xu, T.; Du, D.; Su, W.; Zhang, X. A united approach to learning sparse attributed network embedding. In Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM), Singapore, 17–20 November 2018; pp. 557–566.
16. Bandyopadhyay, S.; Biswas, A.; Kara, H.; Murty, M. A multilayered informative random walk for attributed social network embedding. In *ECAI 2020*; IOS Press: Clifton, VA, USA, 2020; pp. 1738–1745.
17. Wang, H.; Lian, D.; Tong, H.; Liu, Q.; Huang, Z.; Chen, E. Decoupled representation learning for attributed networks. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 2430–2444. [CrossRef]
18. Gao, H.; Ji, S. Graph u-nets. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 2083–2092.
19. Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.A. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, 5–9 July 2008; pp. 1096–1103.

20. Wang, C.; Pan, S.; Hu, R.; Long, G.; Jiang, J.; Zhang, C. Attributed graph clustering: A deep attentional embedding approach. *arXiv* **2019**, arXiv:1906.06532.
21. Xu, H.; Xia, W.; Gao, Q.; Han, J.; Gao, X. Graph embedding clustering: Graph attention auto-encoder with cluster-specificity distribution. *Neural Netw.* **2021**, *142*, 221–230. [CrossRef] [PubMed]
22. Jin, D.; Liu, Z.; Li, W.; He, D.; Zhang, W. Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 152–159.
23. Shi, H.; Fan, H.; Kwok, J.T. Effective decoding in graph auto-encoder using triadic closure. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 906–913.
24. Xie, J.; Girshick, R.; Farhadi, A. Unsupervised deep embedding for clustering analysis. In Proceedings of the International Conference on Machine Learning. PMLR, New York, NY, USA, 19–24 June 2016; pp. 478–487.
25. Yang, L.; Cao, X.; He, D.; Wang, C.; Wang, X.; Zhang, W. Modularity based community detection with deep learning. In Proceedings of the IJCAI, New York, NY, USA, 9–15 July 2016; Volume 16, pp. 2252–2258.
26. He, D.; Song, Y.; Jin, D.; Feng, Z.; Zhang, B.; Yu, Z.; Zhang, W. Community-centric graph convolutional network for unsupervised community detection. In Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, Yokohama, Japan, 7–15 January 2021; pp. 3515–3521.
27. Waikhom, L.; Patgiri, R. Recurrent convolution based graph neural network for node classification in graph structure data. In Proceedings of the 2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Virtual, 27–28 January 2022; pp. 201–206.
28. Ng, A.; Jordan, M.; Weiss, Y. On spectral clustering: Analysis and an algorithm. *Adv. Neural Inf. Process. Syst.* **2001**, *14*, 849–856.
29. Pan, S.; Hu, R.; Long, G.; Jiang, J.; Yao, L.; Zhang, C. Adversarially regularized graph autoencoder for graph embedding. *arXiv* **2018**, arXiv:1802.04407.
30. Zheng, S.; Zhu, Z.; Zhang, X.; Liu, Z.; Cheng, J.; Zhao, Y. Distribution-induced bidirectional generative adversarial network for graph representation learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 7224–7233.
31. Zhu, H.; Koniusz, P. Simple spectral graph convolution. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
32. Mrabah, N.; Bouguessa, M.; Touati, M.F.; Ksantini, R. Rethinking graph auto-encoder models for attributed graph clustering. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 9037–9053. [CrossRef]
33. Sun, D.; Li, D.; Ding, Z.; Zhang, X.; Tang, J. Dual-decoder graph autoencoder for unsupervised graph representation learning. *Knowl. Based Syst.* **2021**, *234*, 107564. [CrossRef]
34. Ahn, S.J.; Kim, M. Variational graph normalized autoencoders. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Virtual, 1–5 November 2021; pp. 2827–2831.
35. Gan, G.; Ma, C.; Wu, J. *Data Clustering: Theory, Algorithms, and Applications*; SIAM: Philadelphia, PA, USA, 2020.
36. Xie, Y.; Xu, Z.; Zhang, J.; Wang, Z.; Ji, S. Self-supervised learning of graph neural networks: A unified review. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 2412–2429. [CrossRef] [PubMed]
37. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
38. Wang, J.; Liang, J.; Yao, K.; Liang, J.; Wang, D. Graph convolutional autoencoders with co-learning of graph structure and node attributes. *Pattern Recogn.* **2022**, *121*, 108215. [CrossRef]
39. Van Der Maaten, L. Accelerating t-SNE using tree-based algorithms. *J. Mach. Learn. Res.* **2014**, *15*, 3221–3245.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.