Pattern Recognition 122 (2022) 108230

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Deep neighbor-aware embedding for node clustering in attributed graphs

Chun Wang[a], Shirui Pan[b,*], Celina P. Yu[c], Ruiqi Hu[a], Guodong Long[a], Chengqi Zhang[a]

[a] Australian Artificial Intelligence Institute, University of Technology Sydney, NSW 2007, Australia
[b] Department of Data Science and AI, Faculty of IT, Monash University, Clayton, VIC 3800, Australia
[c] The Global Business College of Australia, Melbourne, Australia

ARTICLE INFO

Article history:
Received 5 March 2019
Revised 18 July 2021
Accepted 6 August 2021
Available online 15 August 2021

Keywords:
Attributed graph
Node clustering
Graph attention network
Graph convolutional network
Network representation

ABSTRACT

Node clustering aims to partition the vertices in a graph into multiple groups or communities. Existing studies have mostly focused on developing deep learning approaches to learn a latent representation of nodes, based on which simple clustering methods like *k*-means are applied. These two-step frameworks for node clustering are difficult to manipulate and usually lead to suboptimal performance, mainly because the graph embedding is not goal-directed, i.e., designed for the specific clustering task. In this paper, we propose a clustering-directed deep learning approach, Deep Neighbor-aware Embedded Node Clustering (DNENC for short) for clustering graph data. Our method focuses on attributed graphs to sufficiently explore the two sides of information in graphs. It encodes the topological structure and node content in a graph into a compact representation via a neighbor-aware graph autoencoder, which progressively absorbs information from neighbors via a *convolutional* or *attentional* encoder. Multiple neighbor-aware encoders are stacked to build a deep architecture followed by an inner-product decoder for reconstructing the graph structure. Furthermore, soft labels are generated to supervise a self-training process, which iteratively refines the node clustering results. The self-training process is jointly learned and optimized with the graph embedding in a unified framework, to benefit both components mutually. Experimental results compared with state-of-the-art algorithms demonstrate the good performance of our framework.

1. Introduction

The development of networked applications has resulted in an overwhelming number of scenarios in which data is naturally represented in graph format rather than flat-table or vector format. Attributed graph-based representation characterizes individual properties through node attributes, and at the same time captures the pairwise relationship through the graph structure. Many real-world tasks, such as the analysis of citation networks, social networks, protein-protein interaction and knowledge graphs [1], all rely on graph-data analytics skills. However, the complexity of graph structure has imposed significant challenges on these graph-related learning tasks, including clustering, which is one of the most popular topics.

Node clustering aims to partition the nodes in the graph into disjoint groups [2–4]. It's an important and basic task in graph analysis and can be widely applied to real-world network mining. Typical applications include community detection [5–7], group segmentation [8], and functional group discovery in enterprise social networks [9]. Further for attributed graphs, a key problem is how to capture the structural relationship between nodes and exploit the node content information.

To solve this problem, more recent studies have resorted to deep learning techniques to learn compact representation or embedding to exploit the rich information of the graph data [10–12]. Based on the learned graph embedding, simple clustering algorithms such as *k*-means are applied to obtain the clustering result. Autoencoder is a mainstream solution for this kind of embedding-based approach [13,14]. An autoencoder consists of an encoder that encodes the input data into low-dimensional space, and a decoder that reconstructs the input data from the encoded embedding. Such autoencoder-based hidden representation learning approaches are specialized in dimension reduction, and can be applied to purely unsupervised environments. Many autoencoder

* Corresponding author at: Department of Data Science and AI, Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia.
E-mail addresses: chun.wang.0918@gmail.com (C. Wang), shirui.pan@monash.edu (S. Pan).

https://doi.org/10.1016/j.patcog.2021.108230
0031-3203/

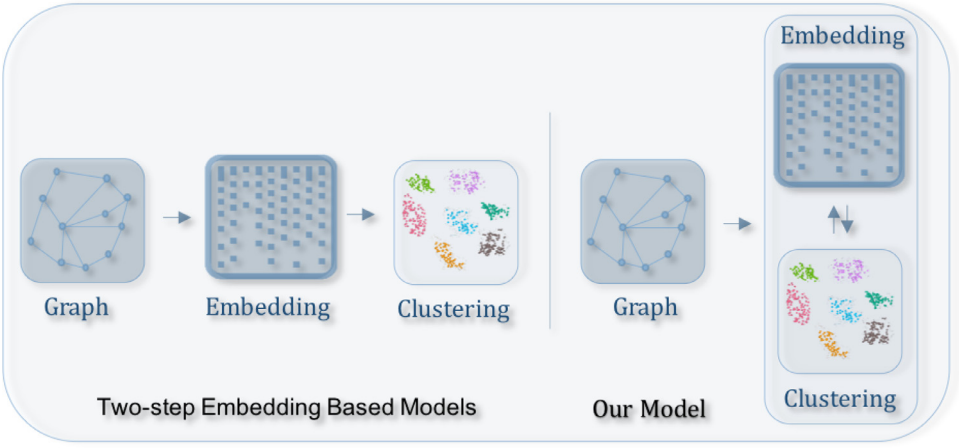


Fig. 1. The difference between two-step embedding learning models and our model.

based node clustering algorithms already exist: Tian et al. considered the similarity of autoencoder and spectral clustering and learned a latent representation for clustering through sparse autoencoder [14]. Cao, Lu, and Xu proposed a deep graph representation model for clustering by capturing structure information through random surfing [13]. The recently developed GAE and VGAE [15] based on graph convolutional network (GCN) can also be adopted for node clustering analysis.

Nevertheless, all these embedding-based methods separate the embedding learning and clustering as two steps. The drawback is that the learned embedding may not be the best fit for the subsequent node clustering task, and the node clustering task is not beneficial to the graph embedding learning. To achieve mutual benefit for these two steps, a goal-directed training framework is highly desirable. However, traditional goal-directed training models are mostly designed for the classification task. We are not aware of any studies on goal-directed node clustering, to the best of our knowledge.

Our Approach Motivated by the above observations, we propose a *Deep Neighbor-aware Embedded Node Clustering* framework (DNENC) with two variants, namely DNENC-Att (with graph attentional autoencoder) and DNENC-Con (with graph convolutional autoencoder) in this paper. To exploit the interrelationship of various-typed graph data, we develop a neighbor-aware graph autoencoder to learn latent representation, which integrates both content and structure information. The encoder progressively aggregates information from its neighbor via a *convolutional* style or an *attentional* mechanism, and multiple layers of encoders are stacked to build a deep architecture for embedding learning. The decoder on the other side, reconstructs the topological graph information and manipulates the latent graph representation. Furthermore, a carefully designed self-training module, which takes the "*confident*" clustering assignments as soft labels, is employed to guide the optimizing procedure. By forcing the current clustering distribution to approach a hypothetical better distribution, in contrast to the separated two-steps embedding learning-based methods (shown in Fig 1), this specialized clustering component makes it possible to simultaneously learn the embedding and perform clustering in a unified framework, thereby achieves better clustering performance. Our contributions can be summarized as follows:

- We introduce a neighbor-aware framework, by developing the first graph attention-based autoencoder, as well as a graph convolution-based autoencoder, to effectively integrate both the structure and content information for attributed graph representation learning.
- We propose a new end-to-end deep learning framework for node clustering. The framework jointly optimizes the embedding learning and node clustering for graph data, to the mutual benefit of both components.
- The experimental results show that our algorithm outperforms state-of-the-art node clustering methods.

The remainder of the paper is organized as follows: Section 2 reviews the related works. Section 3 defines the node clustering problem and briefly describes our framework. Section 4 presents our solution to the problem, and Section 5 details the experimental results. We conclude the paper in Section 6.

2. Related work

Our work is closely related to deep graph neural networks, the autoencoder-based deep clustering algorithms and node clustering algorithms. We briefly review some of these works in this section.

2.1. Deep neural networks for graphs

Deep learning has made remarkable achievements in many domains like voice recognition and image processing. Recently deep learning has also been generalized to graph structured data [16].

The graph convolutional network, in particular, attracts wide attention in the community. Bruna et al. made the first attempt as we are aware of in [17,18]. By using the recurrent Chebyshev polynomials, Defferrard et al. [19] further optimized the filtering scheme and avoided the expensive computation of the Laplacian eigenvectors. Graph convolutional networks (GCN) [20] further simplified the filtering for only 1-step neighborhood nodes and, convolution is thereby considered as a multiplication of the Fourier-transform of a signal in the spectral domain. Several other recent works perform convolution on graphs [21,22], vary as they use different convolutional filtering strategies.

Graph attention can be considered a special kind of graph convolution which place more value on the most relevant parts. Graph attention networks (GATs) was presented for node classification of graph-structured data [23]. It performs self-attention on the graph, computing the hidden representation of each graph node by integrating its neighbor attributes with different weights.

2.2. Autoencoder and deep clustering algorithms

Autoencoder has been a widely used tool in the deep learning area long before adopted to the graph domain. It is specialized in dimension reduction in unsupervised learning tasks such

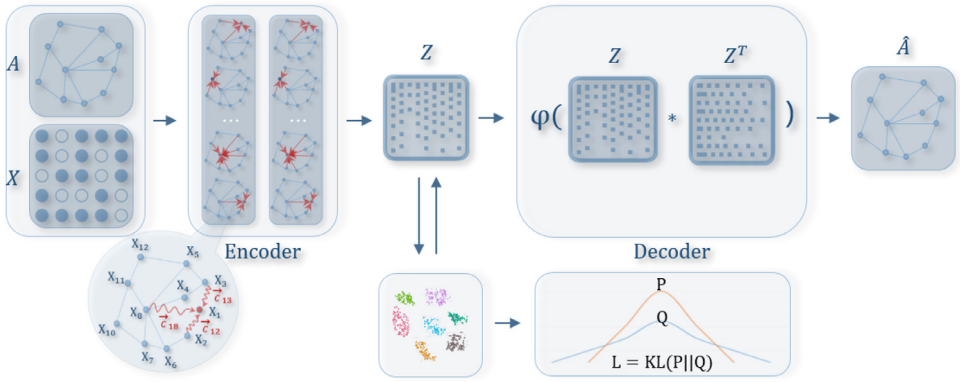


Fig. 2. The conceptual framework of Deep Neighbor-aware Embedded Node Clustering (DNENC). Given a graph $G = (V, E, X)$, DNENC learns a hidden representation Z through a graph autoencoder, which reconstructs the graph structure A with $\hat{A}$ (There's not a $\hat{X}$ reconstructing X in our model as explained in 4.1.3) to optimize Z. The representation Z is manipulated with a self-training clustering module, which is optimized together with the autoencoder and perform clustering during training. The two variants share similar framework and differ as their autoencoder encode the inputs through different strategy.

as clustering and anomaly detection [24]. The autoencoder basically consists of an encoder mapping the input feature X to some hidden low-dimensional representation $h(X)$ and a decoder mapping it back to reconstruct the input feature. The parameters of the autoencoder can be learned by minimizing the reconstruction error.

Deep Embedded Clustering (DEC) is an autoencoder-based clustering technique for plain data [25]. It employs a stacked denoising autoencoder learning approach. After obtaining the hidden representation of the autoencoder with the pre-train, rather than minimizing the reconstruction error through a decoder, the encoder pathway is fine-tuned by a defined Kullback-Leibler divergence clustering loss. Guo et al. considered that the defined clustering loss could corrupt the feature space, leading to non-representative features and a reduction in clustering performance. They improved the DEC algorithm by adding back the decoder and minimizing the reconstruction error as well as the clustering loss [26].

There have since been many algorithms based on such deep clustering framework [27,28]. However, as far as we know, they are only designed for data with flat-table representation. For graph data, complex structure and content information need to be carefully exploited, and end-to-end clustering for graph data is still an open problem in this area.

2.3. Node clustering in graphs

Node clustering has been a long-standing research topic in the graph domain. It can find communities and help with the recognition of partial structural patterns in large networks.

Early methods have taken various approaches to node clustering. Girvan and Newman used centrality indices to find community boundaries and detect social communities [29]. Hastings applied belief propagation to community detection and determined the most likely arrangement of communities [30]. Newman computed the eigenvectors of the graph Laplacian to perform clustering [31,32].

To handle attributed graphs with both content and structure information, NMF(non-negative matrix factorization)-based methods [33,34], probabilistic model [35], relational topic models [36,37], and content propagation [38] have also been widely used.

The limitations of these methods could be summarized as follow: (1) They fail to employ deep architecture to model the interplay between the graph structure and the node content, or even capture only parts of the network information. We hold common opinion that deep learning surpasses traditional methods by learning informative representation through multi-layer message passing. Therefore, these methods are relatively not effective in general; (2) They are mostly applied on original sparse graphs, in which information is not well extracted. As a result, their use of the global graph structure information is usually inefficient, and some of them are limited to the local structure only. This fact leads to the result that such methods cannot effectively exploit the topological information as deep representations do.

Benefiting from the development of deep learning, graph node clustering has progressed significantly in recent years. Many algorithms employ a deep architecture, adopting either sparse autoencoder [14,39] or denoising autoencoder [13] to exploit the deep structure information for clustering. For attributed graphs, graph convolution-based autoencoders are also developed [15], and are combined with marginalized process [40], adversarial regularization [41], etc. for node clustering, link prediction, and other unsupervised tasks. However, these methods are two-step methods, whereas the algorithm presented in this paper is a joint learning approach.

3. Problem definition and overall framework

We consider clustering task on attributed graphs in this paper. An attributed graph is represented as $G = (V, E, X)$, where $V = \{v_i\}$ consists of a set of n nodes ($i \in \{1, \ldots, n\}$), $E = \{e_{ij}\}$ is a set of edges between these nodes ($j \in \{1, \ldots, n\}$ and $i \neq j$). The topological structure of graph G can be represented by an adjacency matrix A, where $A_{ij} = 1$ if $e_{ij} \in E$; otherwise $A_{ij} = 0$. $X = \{x_i\}$ are the n attribute values where $x_i \in \mathbb{R}^m$ is a real-value attribute vector associated with vertex v_i.

Given the graph G and cluster number k, node clustering aims to partition the nodes in G into k disjoint groups $\{G_1, G_2, \ldots, G_k\}$, so that nodes within the same cluster are generally: (1) close to each other in terms of graph structure while distant otherwise; and (2) more likely to have similar attribute values. The trade-off between these two trends is depending on real-world scenarios and people's perceptions. There is no general conclusion.

3.1. Overall framework

In this paper, we construct a graph-based neighbor-aware network to solve this problem. Our framework is shown in Fig 2 and consists of two parts: a graph autoencoder and a self-training clustering module.

- **Graph Autoencoder:** Our neighbor-aware autoencoder takes the attribute values and graph structure as input, and learns the latent representation by minimizing the reconstruction loss.

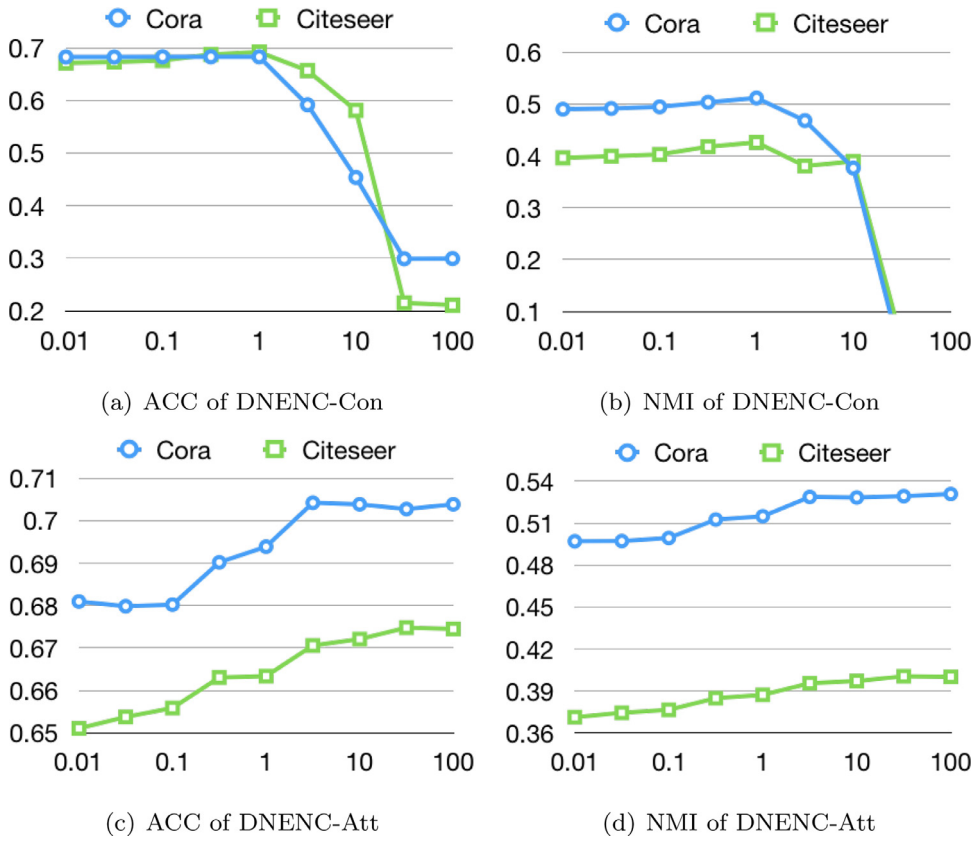


Fig. 3. Parameters study on clustering coefficient γ. The X-axis is the choice of γ and the Y-axis shows the ACC or NMI performance.

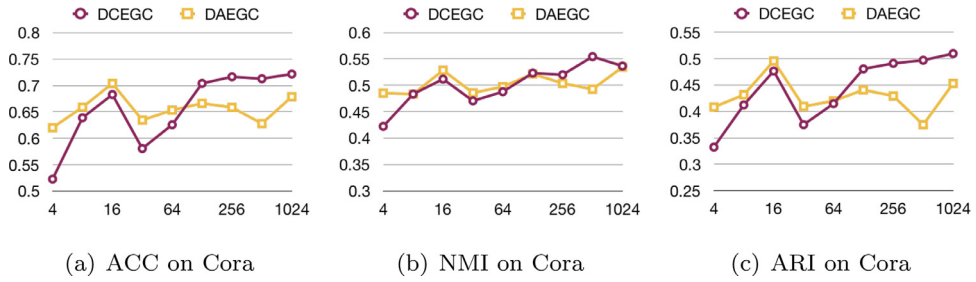


Fig. 4. Parameters study on embedding size.

- **Self-training Clustering:** The module performs clustering based on the learned representation, optimizes the clustering scheme and in return, manipulates the latent representation according to the current clustering result. Since the optimization relies on no label supervision and is totally based-on the current representation, we call it the "self-training clustering" module.

We jointly learn the graph embedding and perform clustering in an end-to-end manner, so that each component benefits the other.

4. Details of proposed method

In this section, we present our proposed Deep Neighbor-aware Embedded Node Clustering (DNENC). We will first develop a graph autoencoder which effectively integrates both structure and content information to learn a latent representation. Based on the representation, a self-training module is proposed to guide the clustering algorithm towards better performance.

4.1. Graph autoencoder

The graph autoencoder aims to learn a low-dimension embedding of the graph G based on both the node content and the graph structure. The basic idea is to progressively aggregate neighbor information (sum up their attribute values) to learn a more informative representation in a deep neural network architecture. To this end, we develop two variants, namely graph attentional encoder and graph convolutional encoder. They differ as they employ an *attentional* mechanism or a *convolutional* style respectively in their encoding strategies.

4.1.1. Graph attentional encoder

To represent both graph structure A and node content X in a unified framework, we develop a variant of the graph attention network [23] as a graph encoder for DNENC-Att. The idea is to learn hidden representations of each node by integrating its neighbor node attributes, to combine the attribute values with the graph structure in the latent representation. The most straightforward strategy to integrate the neighbors of a node is to combine its

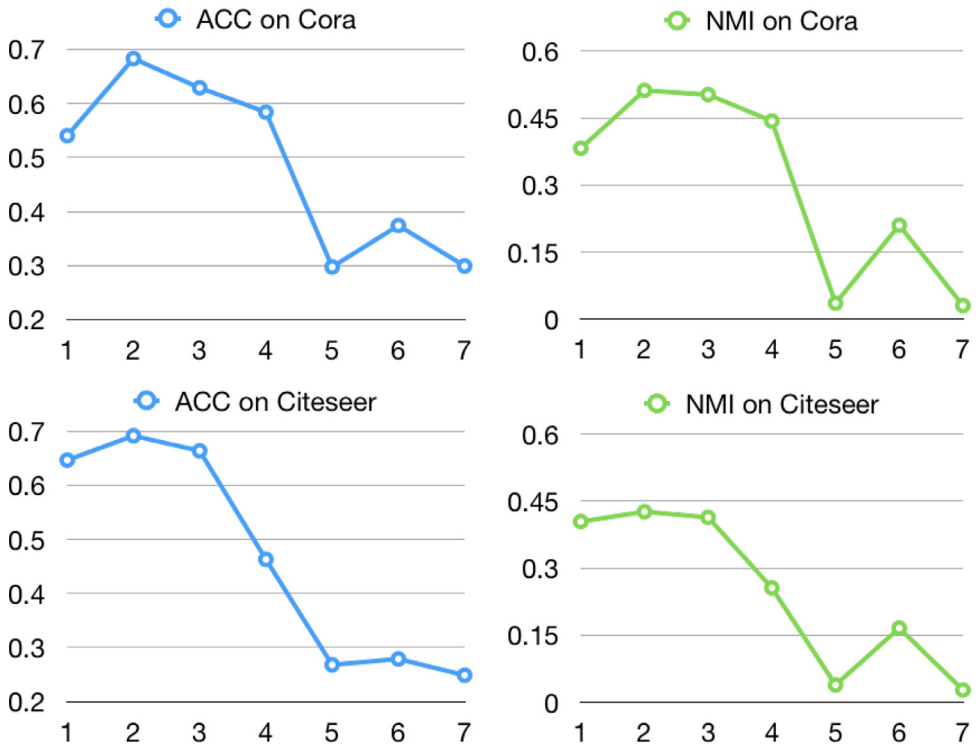


Fig. 5. Parameters study on number of layers.

representation equally with all its neighbors. However, in order to measure the importance of various neighbors, different weights are given to the neighbor representations in our layer-wise graph attention strategy:

$$z_i^{(l+1)} = \sigma\Big(\sum_{j\in N_i} \alpha_{ij} W^{(l)} z_j^{(l)}\Big). \tag{1}$$

Here for layer l, $z_i^{(l+1)}$ denotes the output representation of node i, and N_i denotes the neighbors of i. α_{ij} is the attention coefficient that indicates the importance of neighbor node j to node i, $W^l \in \mathbb{R}^{m_2 \times m_1}$ is the parameter matrix for our autoencoder to learn, with m_1 and m_2 being the input and output dimension of the layer respectively, and σ is a nonlinerity function. To calculate the attention coefficient α_{ij}, we measure the importance of neighbor node j from both the aspects of the attribute value and the topological distance.

From the perspective of attribute values, the attention coefficient α_{ij} can be represented as a single-layer feedforward neural network on the concatenation of x_i and x_j (represented by $||$) with weight vector $a \in \mathbb{R}^{2m_2}$:

$$c_{ij} = a^T [Wx_i || Wx_j]. \tag{2}$$

Topologically, neighbor nodes contribute to the representation of a target node. GAT considers only the 1-hop neighboring nodes (first-order) for graph attention [23]. As graphs have complex structure relationships, we propose to exploit high-order neighbors in our encoder. We obtain a proximity matrix by considering t-order neighbor nodes in the graph:

$$M = (B + B^2 + \cdots + B^t)/t, \tag{3}$$

here $B \in \mathbb{R}^{n\times n}$ is the transition matrix where $B_{ij} = 1/d_i$ if $e_{ij} \in E$ and $B_{ij} = 0$ otherwise. d_i is the degree of node i. B_{ij} can be regarded as the possibility a single-step random walk from node i goes to node j. $(B^2)_{ij}$ represents the possibility the random walk from node i goes to j in 2 steps. $(B^3)_{ij}$ for 3 steps and so on. Therefore $M \in \mathbb{R}^{n\times n}$ is a possibility matrix whose entry M_{ij} denotes the topological relevance of node j to node i up to t orders (performing a random walk from node i, then node j has approximate the possibility of M_{ij} to be reached within t steps).

The attention coefficients are usually normalized across all neighborhoods $j \in N_i$ with a softmax function to make them easily comparable across nodes:

$$\alpha_{ij}(original) = \mathrm{softmax}_j(c_{ij}) = \frac{\exp(c_{ij})}{\sum_{r\in N_i} \exp(c_{ir})}. \tag{4}$$

Adding the topological weights M and an activation function δ (here LeakyReLU is used), the coefficients can be expressed as:

$$\alpha_{ij} = \frac{\exp(\delta(M_{ij}(a^T[Wx_i||Wx_j])))}{\sum_{r\in N_i} \exp(\delta(M_{ir}(a^T[Wx_i||Wx_r])))}. \tag{5}$$

We have $z_i^{(0)} = x_i$ as the input for our problem, and stack two graph attention layers:

$$z_i^{(1)} = \sigma\Big(\sum_{j\in N_i} \alpha_{ij} W^{(0)} x_j\Big), \tag{6}$$

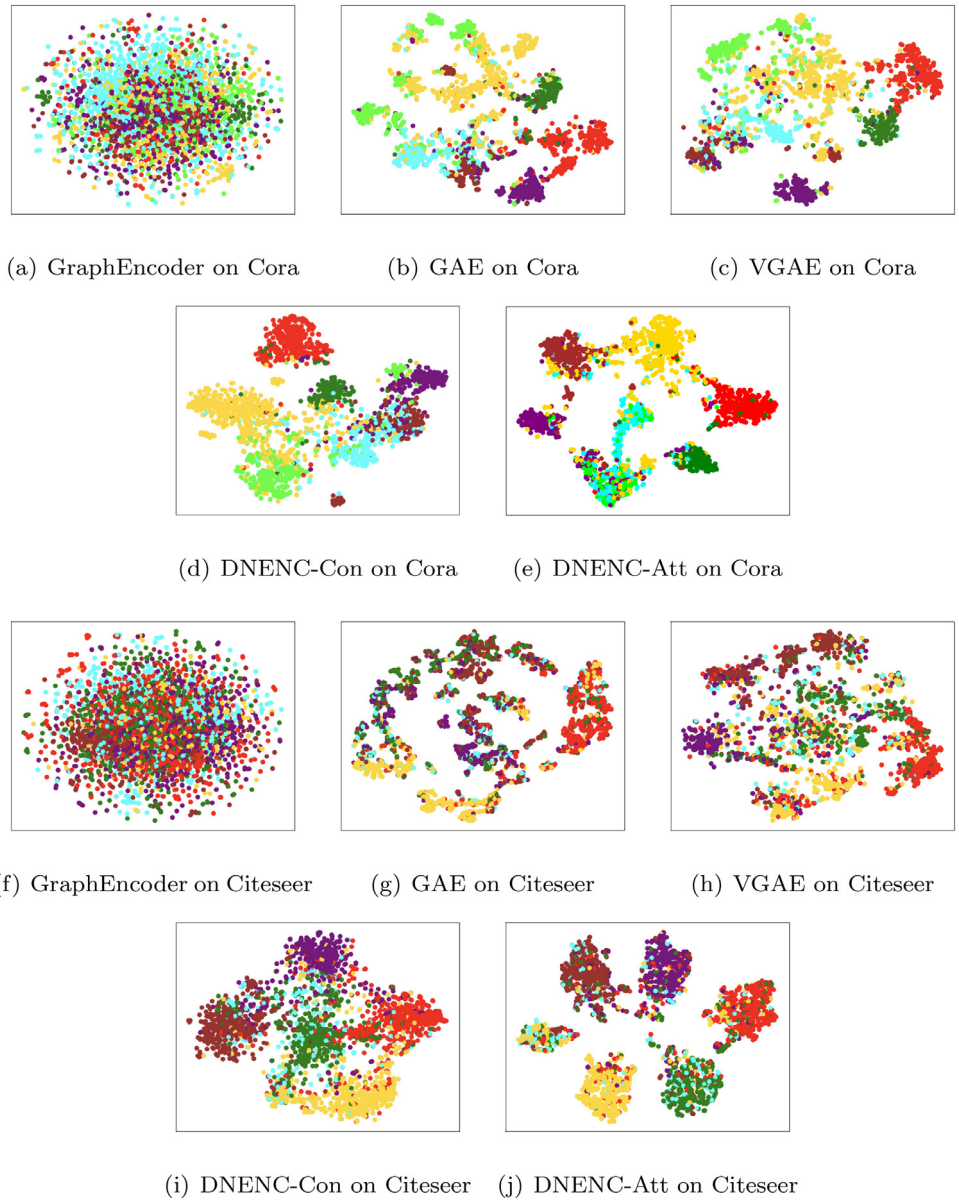
$$z_i^{(2)} = \sigma\Big(\sum_{j\in N_i} \alpha_{ij} W^{(1)} z_j^{(1)}\Big), \tag{7}$$

in this way, our encoder encodes both the graph structure and the node attributes into a hidden representation, i.e., we will have $z_i = z_i^{(2)}$.

4.1.2. Graph convolutional encoder

On the other hand for DNENC-Con, the encoder is defined as a variant of convolutional network from graph data. It extends the operation of *convolution* to graph data in a spectral domain and was formerly used in semi-supervised classification tasks [20]. Our graph convolutional encoder aims to learn a layer-wise transformation combining both the adjacency matrix A representing the graph structure and the feature matrix X by a spectral convolution function $f(z_i^{(l)}, A)$:

$$z_i^{(l+1)} = f(z_i^{(l)}, A). \tag{8}$$



(a) GraphEncoder on Cora (b) GAE on Cora (c) VGAE on Cora

(d) DNENC-Con on Cora (e) DNENC-Att on Cora

(f) GraphEncoder on Citeseer (g) GAE on Citeseer (h) VGAE on Citeseer

(i) DNENC-Con on Citeseer (j) DNENC-Att on Citeseer

Fig. 6. 2D visualization of various methods using the t-SNE algorithm on the Cora and Citeseer dataset.

Here, $z_i^l \in \mathbb{R}^m$ (m features) is the input for convolution and $z_i^{(l+1)}$ is the convolution output. We view the convolution part from the graph level, therefore define $Z^l \in \mathbb{R}^{n\times m}$ and $Z^l = \prod_{i=1}^n z_i^l$ for later use.

We first consider each feature of the graph $s \in \mathbb{R}^n$ as a signal, the convolution function can be defined as the multiplication of the signal with a filter such as:

$$g_\theta \star s = U g_\theta U^T s, \tag{9}$$

where g_θ is a filter parameterized by $\theta \in R^n$, U is the eigenvectors of the normalized graph Laplacian $L = U\Lambda U^T = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, with $D_{ii} = \sum_j A_{ij}$, and I_N the identity matrix, Λ represents a diagonal matrix where the diagonal elements are the eigenvalues of L. We can consider g_θ to be a function of the eigenvalues $g_\theta(\Lambda)$, and $U^T s$ be the graph Fourier transform of s.

Computing the eigen-decomposition of L could be expensive for large graphs. Hence, Hammond et al. suggested $g(\Lambda)$ to be approximated in terms of Chebyshev polynomials [42]:

$$g_\theta(\Lambda) \approx \sum_{y=0}^{Y} \theta_y T_y(\widetilde{\Lambda}), \tag{10}$$

where $\widetilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$. $\lambda_{\max}$ is the largest eigenvalue of L. θ is the Chebyshev coefficients, $T_0(a) = 1$ and $T_1(a) = a$. By further constraint $Y = 1$ and approximate $\lambda_{\max} \approx 2$, a linear function on the graph Laplacian spectrum is obtained:

$$g_\theta \star s \approx \theta(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})s, \tag{11}$$

where θ is the shared filter over the whole graph and $I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ could be approximated by $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ with $\widetilde{A} = A + I_N$ and $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$.

To extend this function to the graph level, or in other words for multiple features $Z^l \in \mathbb{R}^{n\times m}$, the convolution function could be adjusted as:

$$H = g_W \star Z^{(l)} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W, \tag{12}$$

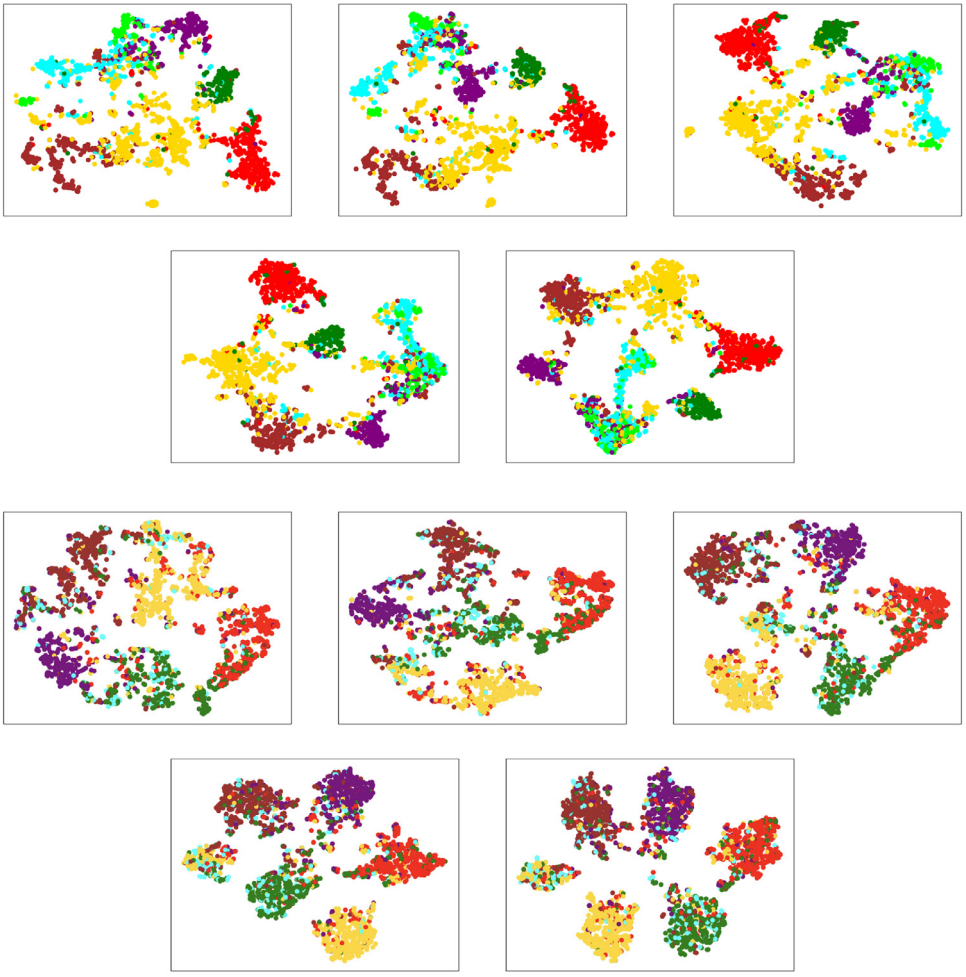


Fig. 7. 2D visualization of the DNENC-Att algorithm using the t-SNE algorithm on the Cora and Citeseer dataset during training (the top line for the Cora dataset, and the bottom line for the Citeseer dataset). The first visualization of each line illustrates the embedding training with the graph autoencoder only, followed by visualizations showing subsequent equal epochs in which the self-training component is included, till the last one being the final embedding visualization.

where $H \in \mathbb{R}^{n\times m}$ is the convolved signal matrix, and W is a matrix of filter parameters replacing θ. Then the layer-wise propagation of the GCN is:

$$f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}), \tag{13}$$

with σ being an activation function such as $\mathrm{Relu}(t) = \max(0, t)$ or $\mathrm{sigmoid}(t) = \frac{1}{1+e^{-t}}$. This convolution propagation function can be computed efficiently in $\mathcal{O}(|E|m^2)$.

We adopt this convolution propagation function and construct a two-layer encoder for our autoencoder:

$$Z^{(1)} = f_{Relu}(X, A|W^{(0)}); \tag{14}$$

$$Z^{(2)} = f_{linears}(Z^{(1)}, A|W^{(1)}). \tag{15}$$

Our encoder encodes both node content and graph structure into a unified hidden representation $Z = Z^{(2)}$.

4.1.3. Inner product decoder

There are various kinds of decoders, which reconstruct either the graph structure, the attribute value, or both. In our method, we choose to reconstruct the graph structure, as our algorithm will be more flexible and will thus fit situations in which no content information is available. The decoder reconstruction aims to optimize the encoded embedding, which already consists of information from both sides. Therefore, the choice of reconstructing the graph structure will not result in information loss of the node attributes. We use a simple inner product decoder which predicts whether there is a link between two nodes. The reconstructed link prediction layer is trained based on the hidden graph representation:

$$\hat{A}_{ij} = \mathrm{sigmoid}(z_i{}^{\top} z_j), \tag{16}$$

where $\hat{A}$ is the reconstructed structure matrix of the graph.

4.1.4. Reconstruction loss

We minimize the reconstruction error by measuring the difference between A and $\hat{A}$:

$$L_r = \sum_{i=1}^{n} loss(A_{ij}, \hat{A}_{ij}). \tag{17}$$

In our paper, the binary cross-entropy loss function is used as the reconstruction loss. By optimizing the autoencoder reconstruction, we can learn the encoder parameter $W^{(0)}$ and $W^{(1)}$, and thereupon the optimized latent embedding Z.

4.2. Self-optimizing embedding

One of the main challenges for node clustering methods is the nonexistence of label guidance. The node clustering task is naturally unsupervised and feedback during training as to whether

the learned embedding is well optimized cannot, therefore, be obtained. To confront this challenge, we develop a self-training clustering algorithm as a solution, which can gradually optimize the embedding for better clustering adaptation, totally by itself without any further information provided for supervision.

Apart from optimizing the reconstruction error, we input our hidden embedding into a self-optimizing clustering module which optimizes a KL (Kullback-Leibler) divergence [25] based clustering loss L_c to help improve the embedding:

$$L_c = KL(P||Q) = \sum_i \sum_u p_{iu} \log \frac{p_{iu}}{q_{iu}}, \tag{18}$$

Where q_{iu} measures the similarity between node embedding z_i and cluster center embedding μ_u. We measure it with a Student's t-distribution so that it could handle different scaled clusters and is computationally convenient [43]:

$$q_{iu} = \frac{(1 + ||z_i - \mu_u||^2)^{-1}}{\sum_k (1 + ||z_i - \mu_k||^2)^{-1}}, \tag{19}$$

it can be seen as a soft clustering assignment distribution of each node with the current embedding. On the other hand, p_{iu} is the target distribution defined as:

$$p_{iu} = \frac{q_{iu}^2 / \sum_i q_{iu}}{\sum_k (q_{ik}^2 / \sum_i q_{ik})}. \tag{20}$$

Soft assignments with high probability (nodes close to the cluster center) are considered to be trustworthy in Q. So the target distribution P raises Q to the second power to emphasize the role of those "confident assignments". The minimizing of the KL distance then force the current distribution Q to approach the target distribution P, so as to set these "confident assignments" as soft labels to supervise Q's embedding learning.

To this end, we first train the autoencoder without the self-optimize clustering part to obtain a meaningful embedding z as described in Eqs. (7) and (15). Self-optimizing clustering is then performed to improve this embedding. To obtain the soft clustering assignment distributions of all the nodes Q through Eq. (19), the k-means clustering is performed just once on the embedding z before training with the self-optimize clustering part, to obtain the initial cluster centers μ.

It is worth mentioning that in the following iterative training, the k-means clustering is never used again, and the cluster centers μ are updated using Stochastic Gradient Descent (SGD) based on the gradients of the clustering loss L_c with respect to μ:

$$\mu_u = \mu_u - \varphi \frac{\partial L_c}{\partial \mu_u}, \tag{21}$$

where φ is the step size. Similarly, $\partial L_c / \partial z_i$ is also computed and passed down to update the parameter matrix W in the encoder together with the gradient from the reconstruction loss of the autoencoder, so as to benefit the embedding learning.

We calculate the target distribution P according to Eq. (20), and the clustering loss L_c according to Eq. (18).

The target distribution P works as "ground-truth labels" in the training procedure, but also depends on the current soft assignment Q which updates at every iteration. It would be hazardous to update P at every iteration with Q as the constant change of target would obstruct learning and convergence. To avoid instability in the self-optimizing process and offer Q time to learn from P, we update P every T iterations. As the detailed choice of T will not observably affect clustering performance according to our experiments (unless too extreme), we simply set it to 5 in our experiments.

In summary, we minimize the clustering loss to help the autoencoder manipulate the embedding space using the embedding's own characteristics and scatter embedding points to obtain better clustering performance.

4.3. Joint embedding and clustering optimization

We jointly optimize the autoencoder embedding and clustering learning, and define our total objective function as:

$$L = L_r + \gamma L_c, \tag{22}$$

where L_r and L_c are the reconstruction loss and clustering loss respectively, $\gamma \geq 0$ is a coefficient that controls the balance in between. It can be optimized by directly back-propagate the gradient from both L_r and L_c to update W, or utilize the unrolled optimization strategy [44–46]. It is worth mentioning that we could gain our clustering result directly from the last optimized Q, and the label estimated for node v_i could be obtained as:

$$r_i = \arg\max_u q_{iu}, \tag{23}$$

which is the most likely assignment from the last soft assignment distribution Q.

Our method is summarized in Algorithm 1. Our algorithm has

Algorithm 1 Unsupervised Deep Neighbor-aware Embedded Graph Clustering.

Require: ~~
Graph G with n nodes, each node with m-dimension attribute value;Number of clusters k;Number of iterations $Iter$;Target distribution update interval T;Clustering Coefficient γ.
Ensure: ~~
Final clustering results.

Update the autoencoder by minimizing Eq. (17) to get the autoencoder hidden embedding Z;
Compute the initial cluster centers μ based on Z;
for $l = 0$ to $Iter - 1$ **do**
  Calculate soft assignment distribution Q with Z and μ according to Eq. (19);
  if $l\%T == 0$ **then**
    Calculate target distribution P with Q by Eq. (20);
  end if
  Calculate reconstruction loss L_r according to Eq. (17)
  Calculate clustering loss L_c according to Eq. (18);
  Update the embedding Z by minimizing Eq. (22);
end for
Get the clustering results with final Q by Eq. (23)

the following advantages:

- **Interplay Exploitation.** The graph neural network-based autoencoder efficiently exploits the interplay between both the graph structure and the node content information.
- **Clustering Specialized Embedding.** The proposed self-training clustering component manipulates the attributed graph embedding to improve the clustering performance.
- **End-to-end Learning.** The framework jointly optimizes the two parts of the loss functions, learns the embedding and performs clustering in an end-to-end manner.

5. Experimental data and methods

5.1. Benchmark datasets

We use three benchmark datasets in our experiments, which are widely used in assessment of attributed graph-based algorithms [20,23], summarized in Table 1. All these datasets consist of scientific publications as nodes, citation relationships as edges and unique words in the documents as features. Publications in these datasets are labeled as they could be assigned to different

Table 1
Benchmark Graph Datasets.

Dataset	Nodes	Features	Clusters	Links	Content Words
Cora	2708	1433	7	5429	3,880,564
Citeseer	3327	3703	6	4732	12,274,336
Pubmed	19,717	500	3	44,338	9,858,500

sub-fields. The numbers of clusters of these datasets k are known and required by all the baselines.

5.2. Baseline methods

We compared a total of 13 algorithms with our method in our experiments. The node clustering algorithms include approaches that use only node attributes or network structure information, and also approaches that combine both. Deep representation learning-based node clustering algorithms were also compared. These algorithms are summarized in Table 2.

5.2.1. Methods using structure or content only

- **K-means** is the base of many clustering methods. Many advanced clustering algorithms involve some kind of transformation of k-means clustering or use k-means on their embeddings. Here we run k-means on our original content data as a benchmark.
- **Spectral clustering** uses the eigenvalues of the similarity matrix to perform dimensionality reduction before clustering and is widely used in node clustering.
- **DeepWalk** [47] is a structure-only representation learning method. It obtains random walks on graphs and then trains the representation through neural networks.
- **GraphEncoder** [14] employs deep learning into node clustering by training a stacked sparse autoencoder and gets representation for later clustering.
- **DNGR** [13] is recent work which uses stacked denoising autoencoders and encodes each vertex into a low dimensional vector representation.
- **M-NMF** [48] is a Nonnegative Matrix Factorization model targeted at community-preserved embedding.

5.2.2. Methods using both structure and content

- **RTM** [37] is a relational topic model capturing both structure and content information to learn the topic distributions of documents.
- **RMSC** [49], the robust multi-view spectral clustering method via low-rank and sparsity decomposition, recovering a shared low-rank transition probability matrix for clustering with a transition probability matrix from each view. We regard structure and content data as two views of information.
- **TADW** [50], text-associated DeepWalk. It re-interprets DeepWalk as a matrix factorization method and adds the features of vertices into representation learning.
- **GAE & VGAE** [15] are representation learning algorithms. They combine the graph convolutional network with the (variational) autoencoder.
- **ARGA & ARVGA** [41] are graph convolutional autoencoder-based methods that manipulate GAE & VGAE learned embedding with an adversarial regularizer.
- **AGC** [51] is an adaptive graph convolution method that exploits high-order graph convolution and captures global cluster structure.
- **DNENC-Con** is our proposed unsupervised deep neighbor-aware embedded node clustering with graph convolutional autoencoder.
- **DNENC-Att** is our proposed unsupervised deep neighbor-aware embedded node clustering with graph attentional autoencoder.

For representation learning algorithms such as DeepWalk, TADW and DNGR which do not specify the clustering algorithm, we first learned the representation from these algorithms, and then applied the k-means algorithm on their respective representations, but for algorithms like RMSC which require spectral clustering or an alternative algorithm, we followed their preference and used the specified algorithms. The best results we got are reported in this paper.

5.3. Evaluation metrics & parameter settings

Evaluation Metrics: We use seven metrics to evaluate the clustering result namely Accuracy (ACC), Normalized Mutual Information (NMI), F-score (F), Precision (P), Recall (R), Average Entropy (AE) and Adjusted Rand Index (ARI). These values can be calculated based upon the algorithm obtained clustering scheme and the ground-truth clustering scheme (provided in the datasets). A good clustering scheme should be consistent with the ground-truth scheme, which will lead to a lower value of average entropy and higher values for all the other metrics. These evaluation matrices differ as they measure consistency differently [49]. A baseline performing novel clustering should stay ahead on most evaluation metrics.

- **ACC** is the average performance of label matching clustering results and can be represented as $\sum_i(y_i == f(l_i))/n$, where f is the mapping function which maps category labels to cluster labels.
- **NMI** measures the mutual information entropy between the resulting cluster labels and ground truth labels followed by a normalization operation.
- **F-score** is the harmonic mean value of *Precision* and *Recall*;
- **Precision** is the fraction of correctly clustered nodes among the retrieved nodes;
- **Recall** is the fraction of correctly clustered nodes that have been retrieved over the total number of relevant nodes;
- **Average Entropy** $= \sum_{i=1}^{k} \frac{m_i}{m} e_i$, where k is the cluster number and m is the number of nodes, and $e_i = -\sum_{j=1}^{k} \frac{m_{ij}}{m_i} log_2 \frac{m_{ij}}{m_i}$, with m_i representing the number of nodes in cluster i and m_{ij} representing the number of nodes in cluster i and labeled j.
- **ARI** is the adjusted rand index (RI) that guarantees a value close to 0, where RI measures the percentage of correct clustering decisions. While RI yield a value between 0 and 1, ARI could be negative.

Parameter Settings: For the baseline algorithms, we carefully select the parameters for each algorithm, following the procedures in the original papers, to achieve their best performance on the datasets. In TADW, for instance, we set the dimension of the factorized matrix to 80, the dimension of the text feature to 200 and the regularization parameter to 0.2; For the DNGR algorithm, we build a three-layers denoising autoencoder with the number of nodes set as 512 and 256 in the hidden layers; For the RMSC algorithm, we regard graph structure and node content as two different views of the data and construct a Gaussian kernel on them. We run the k-means algorithm 50 times for all embedding learning methods for fair comparison.

For our method, we set the clustering coefficient γ to 10 for DNENC-Att and 1 for DNENC-con. For the variant DNENC-Att with attentional encoder, we consider second-order neighbors and set $M = (B + B^2)/2$. The encoder is constructed with a 256-neuron hidden layer and a 16-neuron embedding layer for all datasets. For DNENC-Con with convolutional encoder, a 32-neuron hidden layer and a 16-neuron embedding layer is used instead. All these

Table 2
Algorithm Comparison.

	Content	Structure	Self-training	GCN encoder	GAT encoder	Recover A
K-means	★					
Spectral		★				
GraphEncoder		★				★
DeepWalk		★				★
DNGR		★				★
M-NMF		★				
RTM	★	★				
RMSC	★	★				
TADW	★	★				
GAE&VGAE	★	★		★		★
ARGA&ARVGA	★	★		★		★
AGC	★	★				
DNENC-Att	★	★	★		★	★
DNENC-Con	★	★	★	★		★

Table 3
Experimental Results on **Cora** Dataset.

	Variable	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	X	0.500	0.317	0.376	0.376	0.376	1.810	0.239
Spectral	A	0.398	0.297	0.332	0.312	0.355	1.871	0.174
GraphEncoder	A	0.301	0.059	0.230	0.214	0.253	2.496	0.046
DeepWalk	A	0.529	0.384	0.435	0.392	0.504	1.681	0.291
DNGR	A	0.419	0.318	0.340	0.266	0.480	1.882	0.142
M-NMF	A	0.423	0.256	0.320	0.304	0.342	1.977	0.162
RTM	X&A	0.440	0.230	0.307	0.332	0.285	2.021	0.169
RMSC	X&A	0.466	0.320	0.347	0.345	0.352	1.808	0.203
TADW	X&A	0.536	0.366	0.401	0.342	0.492	1.749	0.240
GAE	X&A	0.530	0.397	0.415	0.431	0.401	1.583	0.293
VGAE	X&A	0.592	0.408	0.456	0.489	0.429	1.545	0.347
ARGA	X&A	0.669	0.489	0.666	0.680	0.686	1.322	0.422
ARVGA	X&A	0.581	0.426	0.560	0.562	0.588	1.492	0.329
AGC	X&A	0.689	0.522	0.656	0.672	0.675	1.273	0.448
DNENC-Att	X&A	0.704	0.528	0.682	0.704	0.706	1.229	0.496
DNENC-Con	X&A	0.683	0.512	0.659	0.665	0.689	1.269	0.477

choices are carefully decided according to our sensitivity analysis (see Section 6.2) and with reference to the previous related works [15].

6. Experiment results

We compare our DNENC with baselines mentioned above on node clustering first. Then we perform detailed analysis on coefficients in the model.

6.1. Clustering performance comparison

The experiment results on the three benchmark datasets are summarized in Tables 3, 4 and 5. X, A, and X&A indicate if the algorithm is performed upon only the node content X, the graph structure A, or both content and structure information, respectively (other values like the clustering coefficient γ are hyper-parameters of the model). We can see that our methods can outperforms most of the baselines across most of the evaluation metrics when applied to these three benchmark datasets. AGC is able to outperform our method on the Pubmed dataset, may because Pubmed is a large and simple dataset which adverse to our deep architecture.

One Side v.s Both Side of Information: We can easily observe from these results that methods using both the structure and content information of the graph generally perform better than those using only one side of information. In the Cora dataset, for example, TADW, GAE, VGAE, AGC and our method outperform all the baselines using one side of information. This observation demonstrates that both the graph structure and node content contain useful information for node clustering, and illustrates the significance of capturing the interplay between two-sides information.

Deep Learning Models: The results of most of the deep learning models are satisfactory. The GraphEncoder and DNGR algorithm are not necessarily an improvement over the other algorithms, although they both employ deep autoencoder for representation learning. This observation may result from their neglect at the node content information.

Effectiveness of DNENC: It is worth mentioning that our algorithms, both DNENC-Con and DNENC-Att, can outperform GAE and VGAE on the three datasets. On the Cora dataset for example, our method DNENC-Att represents a relative increase of 18.97% and 29.49% w.r.t. accuracy and NMI against VGAE, and the increase is even greater on the Citeseer dataset. The reasons for this are that (1) we employ a graph convolutional/attentional network that effectively integrates both content and structure information of the graph; (2) we use a deep architecture to learn the representation, which captures more underlying information; (3) Our self-training clustering component is specialized and powerful in improving the clustering efficiency.

DNENC-Att v.s. DNENC-Con: The results show that DNENC-Att outperforms DNENC-Con on Cora dataset, while DNENC-Con outperforms DNENC-Att on Citeseer and Pubmed datasets. Their performance is very close. This is because both of them are clustering-directed approaches manipulated by the self-training clustering component. The embedding will be optimized by the clustering objective, and finally achieve very similar results for the clustering task.

6.2. Sensitivity analysis

We also investigate the sensitivity of the parameters for our algorithm.

Table 4
Experimental Results on **Citeseer** Dataset.

	Variable	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	X	0.544	0.312	0.413	0.411	0.416	1.738	0.285
Spectral	A	0.308	0.090	0.257	0.241	0.276	2.300	0.082
GraphEncoder	A	0.293	0.057	0.213	0.215	0.211	2.380	0.043
DeepWalk	A	0.390	0.131	0.305	0.282	0.336	2.201	0.137
DNGR	A	0.326	0.180	0.300	0.200	0.609	2.168	0.043
M-NMF	A	0.336	0.099	0.255	0.228	0.291	2.288	0.070
RTM	X&A	0.451	0.239	0.342	0.349	0.335	1.915	0.203
RMSC	X&A	0.516	0.308	0.404	0.383	0.430	1.767	0.266
TADW	X&A	0.529	0.320	0.436	0.376	0.532	1.781	0.286
GAE	X&A	0.380	0.174	0.297	0.291	0.304	2.093	0.141
VGAE	X&A	0.392	0.163	0.278	0.251	0.315	2.131	0.101
ARGA	X&A	0.559	0.289	0.544	0.578	0.539	1.795	0.257
ARVGA	X&A	0.598	0.323	0.570	0.583	0.566	1.703	0.322
AGC	X&A	0.672	0.414	0.627	0.635	0.631	1.500	0.420
DNENC-Att	X&A	0.672	0.397	0.636	0.639	0.640	1.521	0.410
DNENC-Con	X&A	0.692	0.426	0.639	0.640	0.644	1.456	0.449

Table 5
Experimental Results on **Pubmed** Dataset.

	Variable	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	X	0.580	0.278	0.544	0.488	0.621	1.133	0.246
Spectral	A	0.496	0.147	0.471	0.407	0.561	1.323	0.098
GraphEncoder	A	0.531	0.210	0.506	0.456	0.569	1.231	0.184
DeepWalk	A	0.663	0.256	0.539	0.532	0.555	1.142	0.272
DNGR	A	0.468	0.153	0.445	0.387	0.523	1.314	0.059
M-NMF	A	0.470	0.084	0.443	0.391	0.529	1.411	0.058
RTM	X&A	0.575	0.194	0.444	0.456	0.433	1.230	0.149
RMSC	X&A	0.629	0.273	0.521	0.511	0.532	1.116	0.247
TADW	X&A	0.565	0.224	0.481	0.465	0.500	1.196	0.177
GAE	X&A	0.632	0.249	0.511	0.518	0.505	1.146	0.246
VGAE	X&A	0.619	0.216	0.478	0.492	0.464	1.194	0.201
ARGA	X&A	0.632	0.235	0.636	0.636	0.669	1.167	0.221
ARVGA	X&A	0.390	0.004	0.311	0.335	0.342	1.525	0.002
AGC	X&A	0.679	0.306	0.688	0.733	0.695	1.082	0.311
DNENC-Att	X&A	0.671	0.266	0.659	0.677	0.687	1.122	0.278
DNENC-Con	X&A	0.677	0.275	0.675	0.675	0.699	1.105	0.278

Clustering Coefficient γ: We vary the clustering coefficient γ to study the effect of the self-training clustering component. The results are shown in Fig 3.

We could find that experiment on the Cora and Citeseer datasets show similar trends. For DNENC-Con, we observe the best performance with γ around 1. Before γ is increased to that peak, the clustering performance measured by ACC and NMI steadily rise; As we keep adding γ up after that, the performance plummeted as a whole. However, for DNENC-Att, the result keeps good as γ rises.

It shows that our self-training clustering component does work and improve the clustering result. However, a too large value of γ, which means excessively emphasis on the clustering loss, may distort the latent feature space since its trained on estimated targets and could lead to abnormal clustering result. DNENC-Att avoids such plummeting may because the embedding learned with weighted neighbor features are more robust and effective, leading to more accurate initial targets, and make the self-training more stable.

Embedding Size: We also vary the dimension of embedding from 8 neurons to 1024 and report the clustering results on the Cora dataset in Fig 4.

The results show that when adding the dimension of embedding from 4-neuron to 16-neuron, the performance on clustering steadily rises; if we further increase the dimension, the performance of DNENC-Con fluctuates but still have an overall tendency of rising, the performance of DNENC-Att is not necessarily a further improvement since the 8-neuron or 16-neuron embedding is already sufficient with its more efficient attention strategy as argued above. It is worth mention that we set the embedding size to 16 to obtain a stable and efficient model, but it could get markedly better performance when the embedding size is continuously enlarged, to for example, 128-neuron, 256-neuron or 1024-neuron.

Number of Layers: To show the effectiveness of deep architecture, we stack different numbers of layers to observe the alteration of the performance on DNENC-Con. For the autoencoder with only one hidden layer, we encoder the input feature directly into 16-neuron embedding; for the one with two layers, we add a 32-neuron layer in between and construct a d-32-16 encoder like the one we adopted, where d is the input layer dimension; for more layers, we construct d-64-32-16, d-128-64-32-16, etc. encoders with each newly added hidden layer doubling the dimension of its embedding. The performance of all these models on the Cora and Citeseer dataset are reported in Fig 5.

We could observe that, when we stack 2 encoder layers to the model, the performance significantly improve compared with the model with only 1 hidden layer. The performance of the model with 3 stacked hidden layers is also satisfactory. These observations demonstrate that using a stacked architecture instead of a single-layer one can improve the model performance. However, as we continuously add more layers to the model, the performance reduces sharply in terms of all the observations. This is because stacking too many layers will increase the complexity of the architecture, raise the possibility of information loss and enhance the difficulty to the training process.

6.3. Network visualization

We visualize the Cora and Citeseer datasets in two-dimensional space by applying the t-SNE algorithm [52] on the learned embedding. T-SNE is a commonly used algorithm to map high-dimensional data into 2D space for visualization. The results in Fig 6 show that we obtain outstanding embedding as well as clearer clustering results, compared with the baseline methods, benefit from our self-clustering components which contribute to both clustering and embedding learning.

We also visualize the variation of the embedding on the Cora and Citeseer datasets during training as shown in Fig 7. We can observe that, after training with our graph attentional autoencoder, the embedding is already meaningful. However by applying self-training clustering, the embedding becomes more evident as our training progresses, with less overlapping and each group of nodes gradually gathered together.

7. Conclusion

In this paper, we propose an unsupervised deep neighbor-aware embedding algorithm, DNENC, to jointly perform node clustering and learn graph embedding in an end-to-end manner. Two variants with different autoencoder used, DNENC-Con and DNENC-Att are introduced. In our method, the learned graph embedding integrates both graph structure and node content information and is specialized for clustering tasks. While the node clustering task is naturally unsupervised, we propose a self-training clustering component that generates soft labels from "confident" assignments of the current embedding, to supervise the embedding updating. The clustering loss and autoencoder reconstruction loss are jointly optimized to simultaneously obtain both graph embedding and node clustering result. A comparison of the experimental results with various state-of-the-art algorithms validates DNENC's node clustering performance on the benchmark datasets. We may explore node clustering methods that can better fit different datasets from richer real-world scenarios, especially large-scaled ones for future work.

Declaration of Competing Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the Australian Government through the Australian Research Council (ARC) under a Future Fellowship No. FT210100097.

References

[1] S. Ji, S. Pan, E. Cambria, P. Marttinen, P. S. Yu, A survey on knowledge graphs: representation, acquisition and applications, arXiv:2002.00388 (2020).
[2] A. Bojchevski, S. Günnemann, Bayesian robust attributed graph clustering: joint learning of partial anomalies and group structure, in: Proceedings of AAAI, 2018.
[3] P.-Y. Chen, L. Wu, Revisiting spectral graph clustering with generative community models, in: ICDM, IEEE, 2017, pp. 51–60.
[4] T. Guo, S. Pan, X. Zhu, C. Zhang, CFOND: consensus factorization for co-clustering networked data, TKDE (2018).
[5] A. Reihanian, M.-R. Feizi-Derakhshi, H.S. Aghdasi, Overlapping community detection in rating-based social networks through analyzing topics, ratings and links, Pattern Recognit. 81 (2018) 370–387.
[6] Y. Xie, M. Gong, S. Wang, B. Yu, Community discovery in networks with deep sparse filtering, Pattern Recognit. 81 (2018) 50–59.
[7] Y. Li, C. Sha, X. Huang, Y. Zhang, Community detection in attributed graphs: an embedding approach, in: Proceedings of AAAI, 2018.
[8] S.-Y. Kim, T.-S. Jung, E.-H. Suh, H.-S. Hwang, Customer segmentation and strategy development based on customer lifetime value: a case study, Expert Syst. Appl. 31 (1) (2006) 101–107.
[9] R. Hu, S. Pan, G. Long, X. Zhu, J. Jiang, C. Zhang, Co-clustering enterprise social networks, in: IJCNN, 2016, pp. 107–114.
[10] S. Pan, J. Wu, X. Zhu, C. Zhang, Y. Wang, Tri-party deep network representation, in: Proc. of IJCAI, 2016, pp. 1895–1901.
[11] X. Shen, S. Pan, W. Liu, Y. Ong, Q. Sun, Discrete network embedding, in: Proc. of IJCAI, 2018, pp. 3549–3555.
[12] L. Gao, H. Yang, C. Zhou, J. Wu, S. Pan, Y. Hu, Active discriminative network representation learning, in: Proc. of IJCAI, 2018, pp. 2142–2148, doi:10.24963/ijcai.2018/296.
[13] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations, in: Proc. of AAAI, AAAI Press, 2016, pp. 1145–1152.
[14] F. Tian, B. Gao, Q. Cui, E. Chen, T.-Y. Liu, Learning deep representations for graph clustering, in: AAAI, 2014, pp. 1293–1299.
[15] T.N. Kipf, M. Welling, Variational graph auto-encoders, NIPS Workshop on Bayesian Deep Learning, 2016.
[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, arXiv:1901.00596 (2019).
[17] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv:1312.6203 (2013).
[18] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv:1506.05163 (2015).
[19] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: NIPS, 2016, pp. 3844–3852.
[20] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, CoRR (2016) arXiv:1609.02907.
[21] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: NIPS, 2016, pp. 1993–2001.
[22] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: NIPS, 2015, pp. 2224–2232.
[23] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, arXiv:1710.10903 (2017).
[24] C. Zhou, R.C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proc. of KDD, ACM, 2017, pp. 665–674.
[25] J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, in: ICML, 2016, pp. 478–487.
[26] X. Guo, L. Gao, X. Liu, J. Yin, Improved deep embedded clustering with local structure preservation, in: IJCAI, 2017, pp. 1753–1759.
[27] K.G. Dizaji, A. Herandi, C. Deng, W. Cai, H. Huang, Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization, in: ICCV, IEEE, 2017, pp. 5747–5756.
[28] X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in: International Conference on Neural Information Processing, Springer, 2017, pp. 373–382.
[29] M. Girvan, M.E. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. 99 (12) (2002) 7821–7826.
[30] M.B. Hastings, Community detection as an inference problem, Phys. Rev. E 74 (3) (2006) 035102.
[31] M.E. Newman, Finding community structure in networks using the eigenvectors of matrices, Phys. Rev. E 74 (3) (2006) 036104.
[32] M.E. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. 103 (23) (2006) 8577–8582.
[33] D. Cai, X. He, X. Wu, J. Han, Non-negative matrix factorization on manifold, in: Proc. of ICDM, IEEE, 2008, pp. 63–72.
[34] Q. Gu, J. Zhou, Co-clustering on manifolds, in: Proc. of SIGKDD, ACM, 2009, pp. 359–368.
[35] D. Cohn, T. Hofmann, The missing link-a probabilistic model of document content and hypertext connectivity, NIPS (2001) 430–436.
[36] Y. Sun, J. Han, J. Gao, Y. Yu, iTopicModel: information network-integrated topic modeling, in: Proc. of ICDM, IEEE, 2009, pp. 493–502.
[37] J. Chang, D.M. Blei, Relational topic models for document networks., in: AIStats, vol. 9, 2009, pp. 81–88.
[38] L. Liu, L. Xu, Z. Wangy, E. Chen, Community detection based on structure and content: a content propagation perspective, in: Proc. of ICDM, IEEE, 2015, pp. 271–280.
[39] P. Hu, K.C. Chan, T. He, Deep graph clustering in social network, in: Proc. of WWW, 2017, pp. 1425–1426.
[40] C. Wang, S. Pan, G. Long, X. Zhu, J. Jiang, MGAE: marginalized graph autoencoder for graph clustering, in: Proc. of CIKM, ACM, 2017, pp. 889–898.
[41] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, C. Zhang, Adversarially regularized graph autoencoder for graph embedding, in: Proc. of IJCAI, 2018, pp. 2609–2615.
[42] D.K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, arXiv:0912.3848 (2009).
[43] L.v.d. Maaten, G. Hinton, Visualizing data using t-SNE, JMLR (2008) 2579–2605.
[44] U. Schmidt, S. Roth, Shrinkage fields for effective image restoration, in: Proc. of the CVPR, 2014, pp. 2774–2781.
[45] S. Diamond, V. Sitzmann, F. Heide, G. Wetzstein, Unrolled optimization with deep priors, arXiv:1705.08041 (2017).
[46] D. Liang, J. Cheng, Z. Ke, L. Ying, Deep MRI reconstruction: unrolled optimization algorithms meet neural networks, arXiv:1907.11711 (2019).
[47] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: online learning of social representations, in: Proc. of KDD, ACM, 2014, pp. 701–710.
[48] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, S. Yang, Community preserving network embedding, AAAI, 2017.
[49] R. Xia, Y. Pan, L. Du, J. Yin, Robust multi-view spectral clustering via low-rank and sparse decomposition., in: Proc. of AAAI, 2014, pp. 2149–2155.

[50] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information., in: Proc. of IJCAI, 2015, pp. 2111–2117.
[51] X. Zhang, H. Liu, Q. Li, X.-M. Wu, Attributed graph clustering via adaptive graph convolution, in: Proc. of IJCAI, 2019, pp. 4327–4333.
[52] L. Van Der Maaten, Accelerating t-SNE using tree-based algorithms, J. Mach. Learn. Res. 15 (1) (2014) 3221–3245.

Chun Wang obtained his B.S. degree in Zhejiang University. He is now pursuing the Ph.D. degree in computer science from the University of Technology Sydney (UTS), Ultimo, NSW, Australia. His research concentrates on data mining and graph embedding.

Shirui Pan received the Ph.D. degree in computer science from the University of Technology Sydney (UTS), Ultimo, NSW, Australia. He is currently a Senior Lecturer with the Faculty of Information Technology, Monash University, Australia. Prior to that, he was a Lecturer with the School of Software, University of Technology Sydney. His research interests include data mining and machine learning. To date, Dr Pan has published over 100 research papers in top-tier journals and conferences, including the IEEE Transactions on Neural Networks and Learning Systems (TNNLS), IEEE Transactions on Knowledge and Data Engineering (TKDE), IEEE Transactions on Cybernetics (TCYB), ICDE, AAAI, IJCAI, and ICDM.

Celina P. Yu received the Ph.D. degree in finance from RMIT University, Melbourne, VIC, Australia. She is currently the Managing Director with the Global Business College of Australia (GBCA), Melbourne, VIC, Australia, and plays an indispensable role in the partnership with University of Canberra, Canberra, ACT, Australia, to deliver tertiary programs in Melbourne. Dr. Yu was a recipient of RMIT University's prestigious Best Doctoral Research Excellence Award.

Ruiqi Hu received the bachelor's degree in software engineering from the Tianjin Polytechnic University (TJPU), Tianjin, China, in 2013. Since January 2016, he has been working toward the PhD degree in the Centre for Quantum Computation and Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia. His research interests include data mining and machine learning.

Guodong Long received his Ph.D. degree in computer science from University of Technology, Sydney (UTS), Australia, in 2014. He is a Associate Professor in Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS). His research focuses on machine learning, data mining and cloud computing.

Chengqi Zhang received his PhD degree from the University of Queensland, Brisbane, Australia, in 1991 and a DSc degree (higher doctorate) from Deakin University, Geelong, Australia, in 2002. Since December 2001, he has been a Professor of Information Technology with the University of Technology Sydney (UTS), Australia, where he was Director of the UTS Priority Investment Research Centre for Quantum Computation and Intelligent Systems from 2008 to 2016. He is the Executive Di- rector of UTS Data Science since 2017. He has published more than 200 research papers, including several in first-class international journals, such as the Artificial Intelligence, IEEE, and ACM Transactions. He has published six monographs and edited 16 books and has attracted 11 Australian Research Council grants. His research interests mainly focus on data mining and its applications. He has been serving as an Associate Editor for three international journals, including IEEE Transactions on Knowledge and Data Engineering (2005–2008). He is General Co-Chair of KDD 2015 in Sydney and the Local Arrangements Chair of IJCAI-2017 in Melbourne, and a Fellow of the Australian Computer Society and a Senior Member of the IEEE.