Adaptive Graph Encoder for Attributed Graph Embedding

Ganqu Cui[1,2,3], Jie Zhou[1,2,3], Cheng Yang[4*], Zhiyuan Liu[1,2,3*]
[1]Department of Computer Science and Technology, Tsinghua University
[2]Institute for Artificial Intelligence, Tsinghua University
[3]Beijing National Research Center for Information Science and Technology
[4]School of Computer Science, Beijing University of Posts and Telecommunications
{cgq19,zhoujie18}@mails.tsinghua.edu.cn, albertyang33@gmail.com, liuzy@tsinghua.edu.cn

ABSTRACT

Attributed graph embedding, which learns vector representations from graph topology and node features, is a challenging task for graph analysis. Recently, methods based on graph convolutional networks (GCNs) have made great progress on this task. However, existing GCN-based methods have three major drawbacks. Firstly, our experiments indicate that the entanglement of graph convolutional filters and weight matrices will harm both the performance and robustness. Secondly, we show that graph convolutional filters in these methods reveal to be special cases of generalized Laplacian smoothing filters, but they do not preserve optimal low-pass characteristics. Finally, the training objectives of existing algorithms are usually recovering the adjacency matrix or feature matrix, which are not always consistent with real-world applications. To address these issues, we propose Adaptive Graph Encoder (AGE), a novel attributed graph embedding framework. AGE consists of two modules: (1) To better alleviate the high-frequency noises in the node features, AGE first applies a carefully-designed Laplacian smoothing filter. (2) AGE employs an adaptive encoder that iteratively strengthens the filtered features for better node embeddings. We conduct experiments using four public benchmark datasets to validate AGE on node clustering and link prediction tasks. Experimental results show that AGE consistently outperforms state-of-the-art graph embedding methods considerably on these tasks.

KEYWORDS

attributed graph embedding, graph convolutional networks, Laplacian smoothing, adaptive learning

ACM Reference Format:
Ganqu Cui[1,2,3], Jie Zhou[1,2,3], Cheng Yang[4*], Zhiyuan Liu[1,2,3*]. 2020. Adaptive Graph Encoder for Attributed Graph Embedding. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403140

∗ Cheng Yang and Zhiyuan Liu are corresponding authors.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403140

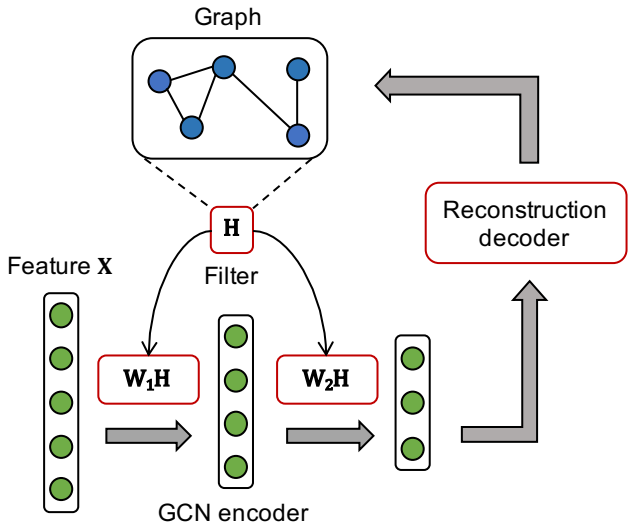


Figure 1: The architecture of graph autoencoder [15]. The components we argue about are marked in red blocks: Entanglement of the filters and weight matrices, design of the filters, and the reconstruction loss.

1 INTRODUCTION

Attributed graphs are graphs with node attributes/features and are widely applied to represent network-structured data in social networks [12], citation networks [16], recommendation systems [37], etc. For tasks analyzing attributed graphs, including node classification, link prediction and node clustering, plenty of machine learning techniques are developed. However, because of the complex high-dimensional non-Euclidean graph structure and various node features, this task imposes the challenge of jointly capturing structure and feature information on machine learning approaches.

Representation learning methods on graphs, also known as graph embedding methods, have emerged as general approaches in graph learning area. This kind of approaches aims to learn low-dimensional representations to encode graph structural information. Early graph embedding approaches are based on Laplacian eigenmaps [21], matrix factorization [3, 19, 34, 36], and random walks [10, 25]. However, these methods are also limited because of their shallow architecture.

More recently, there has been a surge of approaches that focus on deep learning on graphs. Specifically, approaches from the family of graph convolutional networks (GCNs) [16] have made great

progress in many graph learning tasks [39] and strengthen the representation power of graph embedding algorithms. In this paper, we will study the attributed graph embedding problem, which is one of the most important problems in deep graph learning and GCN-based methods have also made great progress on it. Among these methods, most of them are based on graph autoencoder (GAE) and variational graph autoencoder (VGAE) [15]. As shown in Figure 1, they comprise a GCN encoder and a reconstruction decoder. Nevertheless, these GCN-based methods have three major drawbacks:

Firstly, a GCN encoder consists of multiple graph convolutional layers, and each layer contains a graph convolutional filter (**H** in Figure 1), a weight matrix ($\mathbf{W}_1, \mathbf{W}_2$ in Figure 1) and an activation function. However, previous work [35] demonstrates that the entanglement of the filters and weight matrices provides no performance gain for semi-supervised graph representation learning, and even harms training efficiency since it deepens the paths of back-propagation. In this work, we further extend this conclusion to unsupervised scenarios by controlled experiments, showing that our disentangled architecture performs better and more robust than entangled models (Section 5.3).

Secondly, considering the graph convolutional filters, previous research [18] shows in theory that they are actually Laplacian smoothing filters [28] applied on the feature matrix for low-pass denoising. But we show that existing graph convolutional filters are not optimal low-pass filters since they can not filter out noises in some high-frequency intervals. Thus, they can not reach the best smoothing effect (Section 3.3.3).

Thirdly, we also argue that training objectives of these algorithms (either reconstructing the adjacency matrix [23, 31] or feature matrix [24, 32]) are not compatible with real-world applications. To be specific, reconstructing adjacency matrix literally sets the adjacency matrix as the ground truth pairwise similarity, while it is not proper for the lack of feature information. Recovering the feature matrix, however, will force the model to remember high-frequency noises in features, and thus be inappropriate as well.

Motivated by such observations, we propose Adaptive Graph Encoder (AGE), a unified framework for attributed graph embedding. To disentangle the filters and weight matrices, AGE consists of two modules: (1) A well-designed non-parametric Laplacian smoothing filter to perform low-pass filtering in order to get smoothed features. (2) An adaptive encoder to learn more representative node embeddings. To replace the reconstruction training objectives, we employ adaptive learning [6] in this step, which selects training samples from the pairwise similarity matrix and finetunes the embeddings iteratively. The code and data are available on https://github.com/thunlp/AGE.

Our contributions can be summarized as follows:

- *Analysis*: We make a detailed analysis of the mechanism of graph convolutional filters from the perspective of signal smoothing on graphs and Laplacian smoothing. The analysis helps us design a proper Laplacian smoothing filter to better alleviate high-frequency noises.
- *Model*: We propose AGE, a general model for attributed graph embedding. Our two-fold model disentangles the filters and weight matrices. The filters we adopt preserve the optimal low-pass properties. Furthermore, instead of the reconstruction loss, we apply a novel adaptive learning strategy to train node embeddings.
- *Experiment*: We conduct extensive experiments on node clustering and link prediction tasks with real-world benchmark datasets. The results demonstrate that AGE outperforms state-of-the-art attributed graph embedding methods.

2 RELATED WORK

2.1 Conventional Graph Embedding

Early researches on graph embedding merely focus on finding node similarity with graph structure. Methods based on dimension reduction aim to project the high-dimensional adjacency matrix to low-dimensional latent embedding space. Laplacian eigenmaps [21] and matrix factorization [3] are two widely used algorithms for these methods. Another line of researches manages to learn node embeddings with a particular objective function. [10, 25] learn node embeddings by generating random walks and input the sequences into SkipGram model [17], assuming that similar nodes tend to co-occur in same sequences. Other models [4, 27, 33] can be concluded by an encoder-decoder framework [11], while they differ from model structure and training objectives.

Taking node features into account, there are several works make adjustments to encode structural and content information simultaneously. [19, 34, 36] are matrix factorization extensions that add feature-related regularization terms. [2, 5] model features as latent variables in Bayesian networks.

2.2 GCN-based Graph Embedding

As mentioned in the introduction, due to the strong representation power of graph convolutional networks (GCNs) [16], there are several GCN-based approaches for attributed graph embedding and they have achieved state-of-the-art. For unsupervised graph embedding that lacks label information, GCN-based methods can be categorized into two groups by their optimization objectives.

Reconstruct the adjacency matrix. This kind of approaches forces the learned embeddings to recover their localized neighborhood structure. Graph autoencoder (GAE) and variational graph autoencoder (VGAE) [15] learn node embeddings by using GCN as the encoder, then decode by inner product with cross-entropy loss. As variants of GAE (VGAE), [23] exploits adversarially regularized method to learn more robust node embeddings. [31] further employs graph attention networks [30] to differentiate the importance of the neighboring nodes to a target node.

Reconstruct the feature matrix. This kind of models is autoencoders for the node feature matrix while the adjacency matrix merely serves as a filter. [32] leverages marginalized denoising autoencoder to disturb the structure information. To build a symmetric graph autoencoder, [24] proposes Laplacian sharpening as the counterpart of Laplacian smoothing in the encoder. The authors claim that Laplacian sharpening is a process that makes the reconstructed feature of each node away from the centroid of its neighbors to avoid over-smoothing. However, as we will show in the next section, there exists high-frequency noises in raw node features, which harm the quality of learned embeddings.

3 PROPOSED METHOD

In this section, we first formalize the embedding task on attributed graphs. Then we present our proposed Adaptive Graph Encoder (AGE) algorithm. Specifically, we first design an effective graph filter to perform Laplacian smoothing on node features. Given the smoothed node features, we further develop a simple node representation learning module based on adaptive learning [6]. Finally, the learned node embeddings are used for downstream tasks such as node clustering and link prediction.

3.1 Problem Formalization

Given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is the vertex set with n nodes in total, $\mathcal{E}$ is the edge set, and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2. \cdots, \mathbf{x}_n]^T$ is the feature matrix. The topology structure of graph $\mathcal{G}$ can be denoted by an adjacency matrix $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}^{n \times n}$, where $a_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, indicating there is an edge from node v_i to node v_j. $\mathbf{D} = \text{diag}(d_1, d_2, \cdots, d_n) \in \mathbb{R}^{n \times n}$ denotes the degree matrix of $\mathbf{A}$, where $d_i = \sum_{v_j \in \mathcal{V}} a_{ij}$ is the degree of node v_i. The graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

The purpose of attributed graph embedding is to map nodes to low-dimensional embeddings. We take $\mathbf{Z}$ as the embedding matrix and the embeddings should preserve both the topological structure and feature information of graph $\mathcal{G}$.

For downstream tasks, we consider node clustering and link prediction. The node clustering task aims to partition the nodes into m disjoint groups $\{G_1, G_2, \cdots, G_m\}$, where similar nodes should be in the same group. The link prediction task requires the model to predict whether there is a potential edge existing between two given nodes.

3.2 Overall Framework

The framework of our model is shown in Figure 2. It consists of two parts: a Laplacian smoothing filter and an adaptive encoder.

- **Laplacian Smoothing Filter**: The designed filter $\mathbf{H}$ serves as a low-pass filter to denoise the high-frequency components of the feature matrix $\mathbf{X}$. The smoothed feature matrix $\tilde{\mathbf{X}}$ is taken as input of the adaptive encoder.
- **Adaptive Encoder**: To get more representative node embeddings, this module builds a training set by adaptively selecting node pairs which are highly similar or dissimilar. Then the encoder is trained in a supervised manner.

After the training process, the learned node embedding matrix $\mathbf{Z}$ is used for downstream tasks.

3.3 Laplacian Smoothing Filter

The basic assumption for graph learning is that nearby nodes on the graph should be similar, thus node features are supposed to be *smooth* on the graph manifold. In this section, we first explain what *smooth* means. Then we give the definition of the generalized Laplacian smoothing filter and show that it is a smoothing operator. Finally, we answer how to design an optimal Laplacian smoothing filter.

3.3.1 Analysis of Smooth Signals. We start with interpreting *smooth* from the perspective of graph signal processing. Take $\mathbf{x} \in \mathbb{R}^n$ as a graph signal where each node is assigned with a scalar. Denote the filter matrix as $\mathbf{H}$. To measure the smoothness of graph signal $\mathbf{x}$, we can calculate the *Rayleigh quotient* [13] over the graph Laplacian $\mathbf{L}$ and $\mathbf{x}$:

$$R(\mathbf{L}, \mathbf{x}) = \frac{\mathbf{x}^\intercal \mathbf{L} \mathbf{x}}{\mathbf{x}^\intercal \mathbf{x}} = \frac{\sum_{(i,j) \in \mathcal{E}} (x_i - x_j)^2}{\sum_{i \in \mathcal{V}} x_i^2}. \quad (1)$$

This quotient is actually the normalized variance score of $\mathbf{x}$. As stated above, *smooth* signals should assign similar values on neighboring nodes. Consequently, signals with lower *Rayleigh quotient* are assumed to be smoother.

Consider the eigendecomposition of graph Laplacian $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$, where $\mathbf{U} \in \mathbb{R}^{n \times n}$ comprises eigenvectors and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$

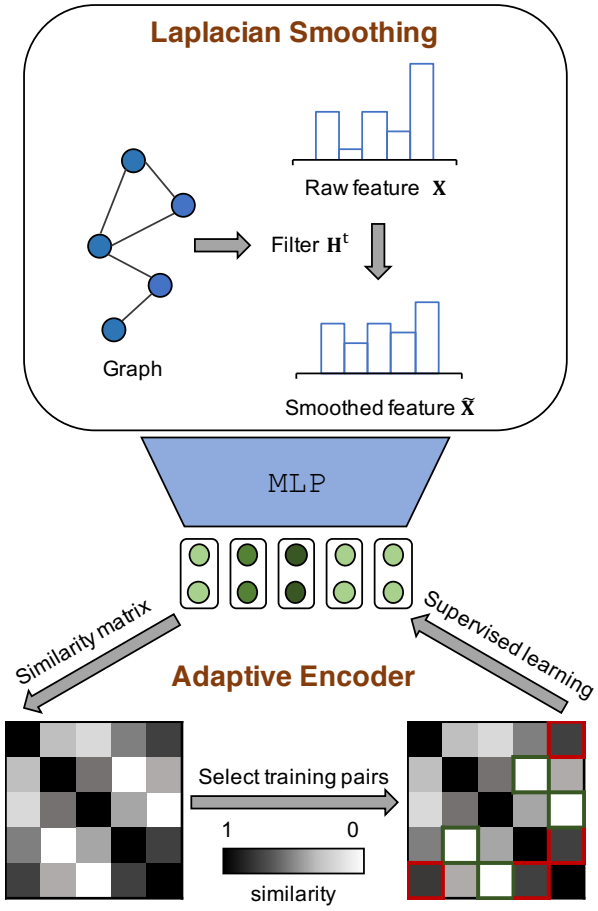


Figure 2: Our AGE framework. Given the raw feature matrix $\mathbf{X}$, we first perform t-layer Laplacian smoothing using filter $\mathbf{H}^t$ to get the smoothed feature matrix $\tilde{\mathbf{X}}$ (Top). Then the node embeddings are encoded by the adaptive encoder which utilizes the adaptive learning strategy: (1) Calculate the pairwise node similarity matrix. (2) Select positive and negative training samples of high confidence (red and green squares). (3) Train the encoder by a supervised loss (Bottom).

is a diagonal matrix of eigenvalues. Then the smoothness of eigenvector $\mathbf{u}_i$ is given by

$$R(\mathbf{L}, \mathbf{u}_i) = \frac{\mathbf{u}_i^\intercal \mathbf{L}\mathbf{u}_i}{\mathbf{u}_i^\intercal \mathbf{u}_i} = \lambda_i. \tag{2}$$

Eq. (2) indicates that smoother eigenvectors are associated with smaller eigenvalues, which means lower frequencies. Thus we decompose signal $\mathbf{x}$ on the basis of $\mathbf{L}$ based on Eq. (1) and Eq. (2):

$$\mathbf{x} = \mathbf{U}\mathbf{p} = \sum_{i=1}^{n} p_i \mathbf{u}_i. \tag{3}$$

where p_i is the coefficient of eigenvector $\mathbf{u}_i$. Then the smoothness of $\mathbf{x}$ is actually

$$R(\mathbf{L}, \mathbf{x}) = \frac{\mathbf{x}^\intercal \mathbf{L}\mathbf{x}}{\mathbf{x}^\intercal \mathbf{x}} = \frac{\sum_{i=1}^{n} p_i^2 \lambda_i}{\sum_{i=1}^{n} p_i^2}. \tag{4}$$

Therefore, to get smoother signals, the goal of our filter is filtering out high-frequency components while preserving low-frequency components. Because of its high computational efficiency and convincing performance, Laplacian smoothing filters [28] are often utilized for this purpose.

3.3.2 Generalized Laplacian Smoothing Filter. As stated by [28], the generalized Laplacian smoothing filter is defined as

$$\mathbf{H} = \mathbf{I} - k\mathbf{L}, \tag{5}$$

where k is real-valued. Employ $\mathbf{H}$ as the filter matrix, the filtered signal $\tilde{\mathbf{x}}$ is present by

$$\tilde{\mathbf{x}} = \mathbf{H}\mathbf{x} = \mathbf{U}(\mathbf{I} - k\mathbf{\Lambda})\mathbf{U}^{-1}\mathbf{U}\mathbf{p} = \sum_{i=1}^{n}(1 - k\lambda_i)p_i\mathbf{u}_i = \sum_{i=1}^{n} p\prime_i \mathbf{u}_i. \tag{6}$$

Hence, to achieve low-pass filtering, the frequency response function $1-k\lambda$ should be a decrement and non-negative function. Stacking up t Laplacian smoothing filters, we denote the filtered feature matrix $\tilde{\mathbf{X}}$ as

$$\tilde{\mathbf{X}} = \mathbf{H}^t\mathbf{X}. \tag{7}$$

Note that the filter is non-parametric at all.

3.3.3 The Choice of k. In practice, with the renormalization trick $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$, we employ the symmetric normalized graph Laplacian

$$\tilde{\mathbf{L}}_{sym} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{L}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{8}$$

where $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{L}}$ are degree matrix and Laplacian matrix corresponding to $\tilde{\mathbf{A}}$. Then the filter becomes

$$\mathbf{H} = \mathbf{I} - k\tilde{\mathbf{L}}_{sym}. \tag{9}$$

Notice that if we set $k = 1$, the filter becomes the GCN filter.

For selecting optimal k, the distribution of eigenvalues $\tilde{\mathbf{\Lambda}}$ (obtained from the decomposition of $\tilde{\mathbf{L}}_{sym} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^{-1}$) should be carefully discovered.

The smoothness of $\tilde{\mathbf{x}}$ is

$$R(\mathbf{L}, \tilde{\mathbf{x}}) = \frac{\tilde{\mathbf{x}}^\intercal \mathbf{L}\tilde{\mathbf{x}}}{\tilde{\mathbf{x}}^\intercal \tilde{\mathbf{x}}} = \frac{\sum_{i=1}^{n} p\prime_i^2 \lambda_i}{\sum_{i=1}^{n} p\prime_i^2}. \tag{10}$$

Thus $p\prime_i^2$ should decrease as λ_i increases. We denote the maximum eigenvalue as λ_{max}. Theoretically, if $k > 1/\lambda_{max}$, the filter is not low-pass in the $(1/k, \lambda_{max}]$ interval because $p\prime_i^2$ increases in this interval; Otherwise, if $k < 1/\lambda_{max}$, the filter can not denoise all the high-frequency components. Consequently, $k = 1/\lambda_{max}$ is the optimal choice.

It has been proved that the range of Laplacian eigenvalues is between 0 and 2 [7], hence GCN filter is not low-pass in the $(1, 2]$ interval. Some work [31] accordingly chooses $k = 1/2$. However, our experiments show that after renormalization, the maximum eigenvalue λ_{max} will shrink to around $3/2$, which makes $1/2$ not optimal as well. In experiments, we calculate λ_{max} for each dataset and set $k = 1/\lambda_{max}$. We further analyse the effects of different k values (Section 5.5).

3.4 Adaptive Encoder

Filtered by t-layer Laplacian smoothing, the output features are smoother and preserve abundant attribute information.

To learn better node embeddings from the smoothed features, we need to find an appropriate unsupervised optimization objective. To this end, we manage to utilize pairwise node similarity inspired by Deep Adaptive Learning [6]. For attributed graph embedding task, the relationship between two nodes is crucial, which requires the training targets to be suitable similarity measurements. GAE-based methods usually choose the adjacency matrix as true labels of node pairs. However, we argue that the adjacency matrix only records one-hop structure information, which is insufficient. Meanwhile, we address that the similarity of smoothed features or trained embeddings are more accurate since they incorporate structure and features together. To this end, we adaptively select node pairs of high similarity as positive training samples, while those of low similarity as negative samples.

Given filtered node features $\tilde{\mathbf{X}}$, the node embeddings are encoded by linear encoder f:

$$\mathbf{Z} = f(\tilde{\mathbf{X}}; \mathbf{W}) = \tilde{\mathbf{X}}\mathbf{W}, \tag{11}$$

where $\mathbf{W}$ is the weight matrix. We then scale the embeddings to the $[0, 1]$ interval by min-max scaler for variance reduction. To measure the pairwise similarity of nodes, we utilize cosine function to implement our similarity metric. The similarity matrix $\mathbf{S}$ is given by

$$\mathbf{S} = \frac{\mathbf{Z}\mathbf{Z}^\intercal}{\|\mathbf{Z}\|_2^2}. \tag{12}$$

Next, we describe our training sample selection strategy in detail.

3.4.1 Training Sample Selection. After calculating the similarity matrix, we rank the pairwise similarity sequence in the descending order. Here r_{ij} is the rank of node pair (v_i, v_j). Then we set the maximum rank of positive samples as r_{pos} and the minimum rank of negative samples as r_{neg}. Therefore, the generated label of node pair (v_i, v_j) is

$$l_{ij} = \begin{cases} 1 & r_{ij} \le r_{pos} \\ 0 & r_{ij} > r_{neg} \\ \text{None} & \text{otherwise} \end{cases}. \tag{13}$$

In this way, a training set with r_{pos} positive samples and $n^2 - r_{neg}$ negative samples is constructed. Specially, for the first time we construct the training set, since the encoder is not trained, we

directly employ the smoothed features for initializing S:

$$\mathrm{S} = \frac{\tilde{\mathbf{X}}\tilde{\mathbf{X}}^{\mathsf{T}}}{\|\tilde{\mathbf{X}}\|_2^2}. \tag{14}$$

After construction of the training set, we can train the encoder in a supervised manner. In real-world graphs, there are always far more dissimilar node pairs than positive pairs, so we select more than r_{pos} negative samples in the training set. To balance positive/negative samples, we randomly choose r_{pos} negative samples in every epoch. The balanced training set is denoted by O. Accordingly, our cross entropy loss is given by

$$\mathcal{L} = \sum_{(v_i,v_j)\in O} -l_{ij}\log(s_{ij}) - (1-l_{ij})\log(1-s_{ij}). \tag{15}$$

3.4.2 Thresholds Update. Inspired by the idea of curriculum learning [1], we design a specific update strategy for r_{pos} and r_{neg} to control the size of training set. At the beginning of training process, more samples are selected for the encoder to find rough cluster patterns. After that, samples with higher confidence are remained for training, forcing the encoder to capture refined patterns. In practice, r_{pos} decreases while r_{neg} increases linearly as the training procedure goes on. We set the initial threshold as r_{pos}^{st} and r_{neg}^{st}, together with the final threshold as r_{pos}^{ed} and r_{neg}^{ed}. We have $r_{pos}^{ed} \le r_{pos}^{st}$ and $r_{neg}^{ed} \ge r_{neg}^{st}$. Suppose the thresholds are updated T times, we present the update strategy as

$$r'_{pos} = r_{pos} + \frac{r_{pos}^{ed} - r_{pos}^{st}}{T}, \tag{16}$$

$$r'_{neg} = r_{neg} + \frac{r_{neg}^{ed} - r_{neg}^{st}}{T}. \tag{17}$$

As the training process goes on, every time the thresholds are updated, we reconstruct the training set and save the embeddings. For node clustering, we perform Spectral Clustering [22] on the similarity matrices of saved embeddings, and select the best epoch by Davies–Bouldin index [8] (DBI), which measures the clustering quality without label information. For link prediction, we select the best performed epoch on validation set. Algorithm 1 presents the overall procedure of computing the embedding matrix $\mathbf{Z}$.

Algorithm 1 Adaptive Graph Encoder

Input: Adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$, filter layer number t, iteration number max_iter and threshold update times T
Output: Node embedding matrix $\mathbf{Z}$

```
Obtain graph Laplacian L̃_sym from Eq. (8);
k ← 1/λ_max;
Get filter matrix H from Eq. (9);
Get smoothed feature matrix X̃ from Eq. (7);
Initialize similarity matrix S and training set O by Eq. (14);
for iter = 1 to max_iter do
    Compute Z with Eq. (11);
    Train the adaptive encoder with loss in Eq. (15);
    if iter mod (max_iter/T) == 0 then
        Update thresholds with Eq. (16) and (17);
        Calculate the similarity matrix S with Eq. (12);
        Select training samples from S by Eq. (13);
    end if
end for
```

4 EXPERIMENTAL SETTINGS

We evaluate the benefits of AGE against a number of state-of-the-art graph embedding approaches on node clustering and link prediction tasks. In this section, we introduce our benchmark datasets, baseline methods, evaluation metrics, and parameter settings.

4.1 Datasets

We conduct node clustering and link prediction experiments on four widely used network datasets (Cora, Citeseer, Pubmed [26] and Wiki [36]). Features in Cora and Citeseer are binary word vectors, while in Wiki and Pubmed, nodes are associated with tf-idf weighted word vectors. The statistics of the four datasets are shown in Table 1.

Table 1: Dataset statistics

Dataset	# Nodes	# Edges	# Features	# Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Wiki	2,405	17,981	4,973	17
Pubmed	19,717	44,338	500	3

4.2 Baseline Methods

For attributed graph embedding methods, we include 5 baseline algorithms in our comparisons:

GAE and VGAE [15] combine graph convolutional networks with the (variational) autoencoder for representation learning.

ARGA and ARVGA [23] add adversarial constraints to GAE and VGAE respectively, enforcing the latent representations to match a prior distribution for robust node embeddings.

GALA [24] proposes a symmetric graph convolutional autoencoder recovering the feature matrix. The encoder is based on Laplacian smoothing while the decoder is based on Laplacian sharpening.

On the node clustering task, we compare our model with 8 more algorithms. The baselines can be categorized into three groups:

(1) Methods using features only. Kmeans [20] and Spectral Clustering [22] are two traditional clustering algorithms. Spectral-F takes the cosine similarity of node features as input.

(2) Methods using graph structure only. Spectral-G is Spectral Clustering with the adjacency matrix as the input similarity matrix. DeepWalk [25] learns node embeddings by using SkipGram on generated random walk paths on graphs.

(3) Methods using both features and graph. TADW [36] interprets DeepWalk as matrix factorization and incorporates node features under the DeepWalk framework. MGAE [32] is a denoising marginalized graph autoencoder. Its training objective is reconstructing the feature matrix. AGC [38] exploits high-order graph convolution to filter node features. The number of graph convolution layers are selected for different datasets. DAEGC [31] employs

Table 2: Experimental results of node clustering.

Methods	Input	Cora			Citeseer			Wiki			Pubmed		
		ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
Kmeans	F	0.503	0.317	0.244	0.544	0.312	0.285	0.417	0.440	0.151	0.580	0.278	0.246
Spectral-F	F	0.347	0.147	0.071	0.441	0.203	0.183	0.491	0.464	0.254	0.602	0.309	0.277
Spectral-G	G	0.342	0.195	0.045	0.259	0.118	0.013	0.236	0.193	0.017	0.528	0.097	0.062
DeepWalk	G	0.484	0.327	0.243	0.337	0.089	0.092	0.385	0.324	0.173	0.543	0.102	0.088
TADW	F&G	0.560	0.441	0.332	0.455	0.291	0.228	0.310	0.271	0.045	0.511	0.244	0.217
GAE	F&G	0.611	0.482	0.302	0.456	0.221	0.191	0.379	0.345	0.189	0.632	0.249	0.246
VGAE	F&G	0.592	0.408	0.347	0.467	0.261	0.206	0.451	0.468	0.263	0.619	0.216	0.201
MGAE	F&G	0.681	0.489	0.436	0.669	0.416	0.425	0.529	0.510	0.379	0.593	0.282	0.248
ARGA	F&G	0.640	0.449	0.352	0.573	0.350	0.341	0.381	0.345	0.112	0.681	0.276	0.291
ARVGA	F&G	0.638	0.450	0.374	0.544	0.261	0.245	0.387	0.339	0.107	0.513	0.117	0.078
AGC	F&G	0.689	0.537	0.486	0.670	0.411	0.419	0.477	0.453	0.343	0.698	0.316	0.319
DAEGC	F&G	0.704	0.528	0.496	0.672	0.397	0.410	0.482	0.448	0.331	0.671	0.266	0.278
GALA	F&G	0.746	0.577	0.532	0.693	0.441	0.446	0.545	0.504	0.389	0.694	**0.327**	0.321
LS	F&G	0.638	0.493	0.373	0.677	0.419	0.433	0.515	0.534	0.317	0.656	0.300	0.315
LS+RA	F&G	0.742	0.580	0.545	0.658	0.410	0.403	0.552	0.566	0.382	0.652	0.291	0.301
LS+RX	F&G	0.647	0.479	0.423	0.674	0.416	0.424	0.553	0.543	0.365	0.645	0.285	0.251
AGE	F&G	**0.768**	**0.607**	**0.565**	**0.702**	**0.448**	**0.457**	**0.612**	**0.597**	**0.440**	**0.711**	0.316	**0.334**

graph attention network to capture the importance of the neighboring nodes, then co-optimize reconstruction loss and KL-divergence-based clustering loss.

For representation learning algorithms including DeepWalk, TADW, GAE and VGAE which do not specify on the node clustering problem, we apply Spectral Clustering on their learned representations. For other works that conduct experiments on benchmark datasets, the original results in the papers are reported.

AGE variants. We consider 4 variants of AGE to compare various optimization objectives. The Laplacian smoothing filters in these variants are the same, while the encoder of LS+RA aims at reconstructing the adjacency matrix. LS+RX, respectively, reconstructs the feature matrix. LS only preserves the Laplacian smoothing filter, the smoothed features are taken as node embeddings. AGE is our proposed model with adaptive learning.

4.3 Evaluation Metrics & Parameter Settings

To measure the performance of node clustering methods, we employ three metrics: Accuracy (ACC), Normalized Mutual Information (NMI), and Adjusted Rand Index (ARI) [9]. For link prediction, we partition the datasets following GAE, and report Area Under Curve (AUC) and Average Precision (AP) scores. For all the metrics, a higher value indicates better performance.

For the Laplacian smoothing filter, we find the maximum eigenvalues of the four datasets are all around 3/2. Thus we set $k = 2/3$ universally. For the adaptive encoder, we train the MLP encoder for 400 epochs with a 0.001 learning rate by the Adam optimizer [14]. The encoder consists of a single 500-dimensional embedding layer, and we update the thresholds every 10 epochs. We tune other hyperparameters including Laplacian smoothing filter layers t, r_{pos}^{st}, r_{pos}^{ed}, r_{neg}^{st} and r_{neg}^{ed} based on DBI. The detailed hyperparameter settings are reported in Appendix.

5 EXPERIMENTAL RESULTS

In this section, we show and analyse the results of our experiments. Besides the main experiments, we also conduct auxiliary experiments to answer the following hypotheses:

H1: Entanglement of the filters and weight matrices has no improvement for embedding quality.

H2: Our adaptive learning strategy is effective compared to reconstruction losses, and each mechanism has its own contribution.

H3: $k = 1/\lambda_{max}$ is the optimal choice for Laplacian smoothing filters.

5.1 Node Clustering Results

The node clustering results are presented in Table 2, where **bold** and underlined values indicate the highest scores in all methods and all baselines respectively. Our observations are as follows:

Algorithms using both feature and graph information usually achieve better performance than methods leveraging information from single source. This investigation demonstrates that features and graph structure contribute to clustering from different perspectives.

AGE shows superior performance to baseline methods by a considerable margin, especially on Cora and Wiki datasets. Competing with the strongest baseline GALA, our model outperforms it by 2.95%, 5.20% and 6.20% on Cora, by 12.29%, 18.45% and 13.11% on Wiki with respect to ACC, NMI and ARI. Such results show strong evidence advocating our proposed framework. For Citeseer and Pubmed, we give further analysis in section 5.5.

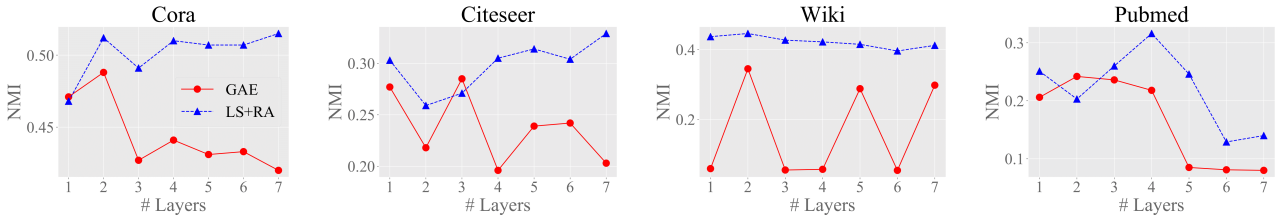


Figure 3: Controlled experiments comparing GAE and LS+RA

Table 3: Experimental results of link prediction.

Methods	Cora		Citeseer	
	AUC	AP	AUC	AP
GAE	0.910	0.920	0.895	0.899
VGAE	0.914	0.926	0.908	0.920
ARGA	0.924	0.932	0.919	0.930
ARVGA	0.924	0.926	0.924	0.930
GALA	0.921	0.922	0.944	0.948
AGE	**0.957**	**0.952**	**0.964**	**0.968**

Compared with GCN-based methods, AGE has simpler mechanisms than those in baselines, such as adversarial regularization or attention. The only trainable parameters are in the weight matrix of the 1-layer perceptron, which minimizes memory usage and improves training efficiency.

5.2 Link Prediction Results

In this section, we evaluate the quality of node embeddings on the link prediction task. Following the experimental settings of GALA, we conduct experiments on Cora and Citeseer, removing 5% edges for validation and 10% edges for test. The training procedure and hyper-parameters remain unchanged. Given the node embedding matrix $\mathbf{Z}$, we use a simple inner product decoder to get the predicted adjacency matrix

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^{\intercal}) \tag{18}$$

where σ is the sigmoid function.

The experimental results are reported in Table 3. Compared with state-of-the-art unsupervised graph representation learning models, AGE outperforms them on both AUC and AP. It is worth noting that the training objectives of GAE/VGAE and ARGA/ARVGA are the adjacency matrix reconstruction loss. GALA also adds reconstruction loss for the link prediction task, while AGE does not utilize explicit links for supervision.

5.3 GAE v.s. LS+RA

We use controlled experiments to verify hypothesis **H1**, evaluating the influence of entanglement of the filters and weight matrices. The compared methods are GAE and LS+RA, where the only difference between them is the position of the weight matrices. GAE, as we show in Figure 1, combines the filter and weight matrix in each layer. LS+RA, however, moves weight matrices after the filter. Specifically, GAE has multiple GCN layers where each one contains a 64-dimensional linear layer, a ReLU activition layer and a graph convolutional filter. LS+RA stacks multiple graph convolutional filters and after which is a 1-layer 64-dimensional perceptron. Both embedding layers of the two models are 16-dimensional. Rest of the parameters are set to the same.

We report the NMI scores for node clustering on the four datasets with different number of filter layers in Figure 3. The results show that LS+RA outperforms GAE under most circumstances with fewer parameters. Moreover, the performance of GAE decreases significantly as the filter layer increases, while LS+RA is relatively stable. A reasonable explanation to this phenomenon is stacking multiple graph convolution layers makes it harder to train all the weight matrices well. Also, the training efficiency will be affected by the deep network.

5.4 Ablation Study

To validate **H2**, we first compare the four variants of AGE on the node clustering task. Our findings are listed below:

(1) Compared with raw features (Spectral-F), smoothed features (LS) integrate graph structure, thus perform better on node clustering. The improvement is considerable.

(2) The variants of our model, LS+RA and LS+RX, also show powerful performances compared with baseline methods, which results from our Laplacian smoothing filter. At the same time, AGE still outperforms the two variants, demonstrating that the adaptive optimization target is superior.

(3) Comparing the two reconstruction losses, reconstructing the adjacency matrix (LS+RA) performs better on Cora, Wiki and Pubmed, while reconstructing the feature matrix (LS+RX) performs better on Citeseer. Such difference illustrates that structure information and feature information are of different importance across datasets, therefore either of them is not optimal universally. Furthermore, on Citeseer and Pubmed, the reconstruction losses contribute negatively to the smoothed features.

Then, we conduct ablation study on Cora to manifest the efficacy of four mechanisms in AGE. We set five variants of our model for comparison.

All five variants cluster nodes by performing Spectral Clustering on the cosine similarity matrix of node features or embeddings. “Raw features" simply performs Spectral Clustering on raw node features; “+Filter" clusters nodes using smoothed node features; “+Encoder" initializes training set from the similarity matrix of smoothed node features, and learns node embeddings via the fixed training set; “+Adaptive" selects training samples adaptively with

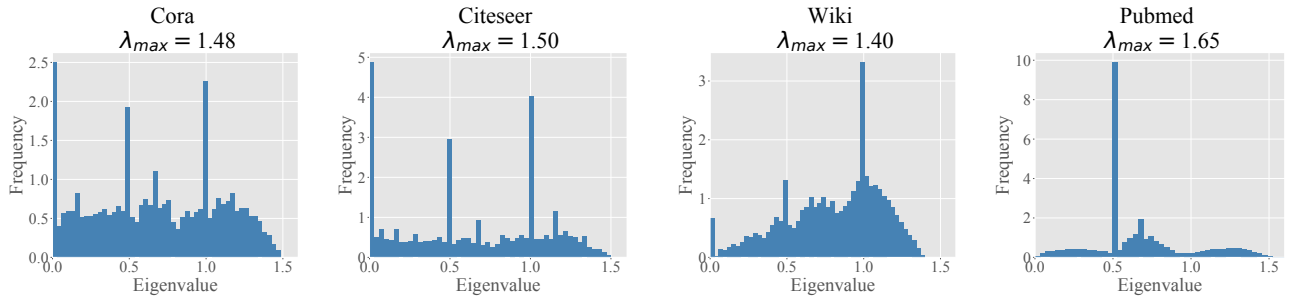


Figure 4: The eigenvalue distributions of benchmark datasets. λ_{max} is the maximum eigenvalue.

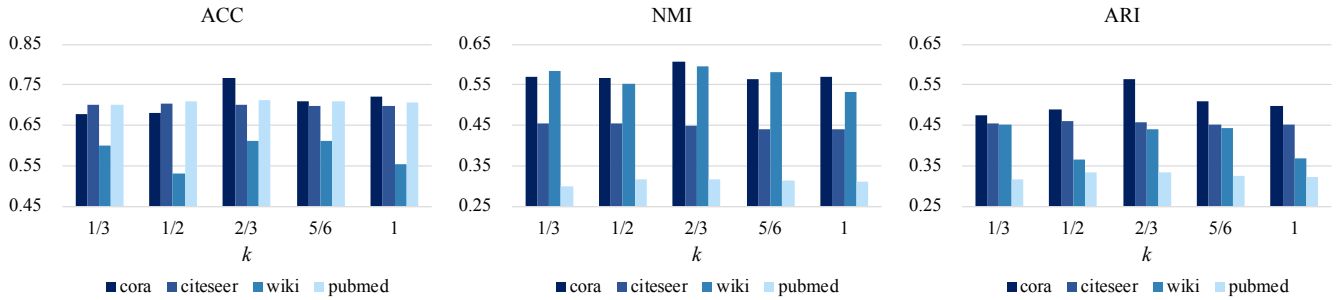


Figure 5: Influence of k on the three metrics.

fixed thresholds; "+Thresholds Update" further adds thresholds update strategy and is exactly the full model.

In Table 4, it is obviously noticed that each part of our model contributes to the final performance, which evidently states the effectiveness of them. Additionally, we can observe that model supervised by the similarity of smoothed features ("+Encoder") outperforms almost all the baselines, giving verification to the rationality of our adaptive learning training objective.

Table 4: Ablation study.

Model Variants	Cora		
	ACC	NMI	ARI
Raw features	0.347	0.147	0.071
+Filter	0.638	0.493	0.373
+Encoder	0.728	0.558	0.521
+Adaptive	0.739	0.585	0.544
+Thresholds Update	**0.768**	**0.607**	**0.565**

5.5 Selection of k

As stated in section 3.3.3, we select $k = 1/\lambda_{max}$ while λ_{max} is the maximum eigenvalue of the renormalized Laplacian matrix. To verify the correctness of our hypothesis (**H3**), we first plot the eigenvalue distributions of the Laplacian matrix for benchmark datasets in Figure 4. Then, we perform experiments with different k and the results are report in Figure 5. From the two figures,we can make the following observations:

(1) The maximum eigenvalues of the four datasets are around 3/2, which supports our selecting $k = 2/3$.

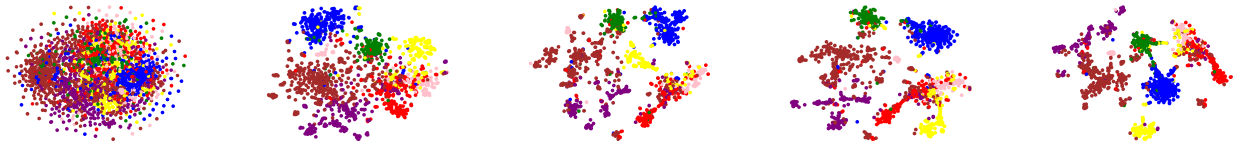
(2) In Figure 5, it is clear that filters with $k = 2/3$ work best for Cora and Wiki datasets, since all three metrics reach the highest scores at $k = 2/3$. For Citeseer and Pubmed, there is little difference for various k.

(3) To further explain why some datasets are sensitive to k while some are not, we can look back into Figure 4. Obviously, there are more high-frequency components in Cora and Wiki than Citeseer and Pubmed. Therefore, for Citeseer and Pubmed, filters with different k achieve similar effects.

Overall, for Laplacian smoothing filters, we can conclude that $k = 1/\lambda_{max}$ is the optimal choice for Laplacian smoothing filters (**H3**).

5.6 Visualization

To intuitively show the learned node embeddings, we visualize the node representations in 2D space using t-SNE algorithm [29]. The figures are shown in Figure 6 and each subfigure corresponds to a variant in the ablation study. From the visualization, we can see that AGE can well cluster the nodes according to their corresponding classes. Additionally, as the model gets complete gradually, there are fewer overlapping areas and nodes belong to the same group gather together.



(a) Raw features (b) +Filter (c) +Encoder (d) +Adaptive (e) Full model

Figure 6: 2D visualization of node representations on Cora using t-SNE. The different colors represent different classes.

6 CONCLUSION

In this paper we propose AGE, a unified unsupervised graph representation learning model. We investigate the graph convolution operation in view of graph signal smoothing, and then design a non-parametric Laplacian smoothing filter which preserves optimal denoising properties to filter out high-frequency noises. In the encoder part, we find adaptive learning is more appropriate for embedding. Experiments on standard benchmarks demonstrate our model has outperformed state-of-the-art baseline algorithms.

For future work, an intriguing direction is to improve the computational efficiency of adaptive learning by avoiding the full computation of the pairwise similarity matrix.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004503) and the National Natural Science Foundation of China (NSFC No. 61772302, 61732008).

REFERENCES

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of ICML*. 41–48.
[2] Aleksandar Bojchevski and Stephan Günnemann. 2018. Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure. In *Proceedings of AAAI*. 2738–2745.
[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of CIKM*. 891–900.
[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of AAAI*. 1145–1152.
[5] Jonathan Chang and David Blei. 2009. Relational topic models for document networks. In *Artificial Intelligence and Statistics*. 81–88.
[6] Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2017. Deep adaptive image clustering. In *Proceedings of ICCV*. 5879–5887.
[7] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. American Mathematical Soc.
[8] David L Davies and Donald W Bouldin. 1979. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), 224–227.
[9] Guojun Gan, Chaoqun Ma, and Jianhong Wu. 2007. *Data clustering: Theory, algorithms, and applications*. Vol. 20. Siam.
[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of SIGKDD*. 855–864.
[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data(base) Engineering Bulletin* 40, 3 (2017), 52–74.
[12] Matthew B Hastings. 2006. Community detection as an inference problem. *Physical Review E* 74, 3 (2006), 035–102.
[13] Roger A Horn and Charles R Johnson. 2012. *Matrix analysis*. Cambridge university press.
[14] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*. 15.
[15] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. In *NIPS Workshop on Bayesian Deep Learning*. 3.
[16] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*. 14.
[17] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of ICML*. 1188–1196.
[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of AAAI*. 3538–3545.
[19] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. 2018. Community detection in attributed graphs: an embedding approach. In *Proceedings of AAAI*. 338–345.
[20] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
[21] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical review E* 74, 3 (2006), 036–104.
[22] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Proceedings of NIPS*. 849–856.
[23] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of IJCAI*. 2609–2615.
[24] Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. 2019. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of ICCV*. 6519–6528.
[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of SIGKDD*. 701–710.
[26] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.
[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of WWW*. 1067–1077.
[28] Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 351–358.
[29] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The journal of machine learning research* 15, 1 (2014), 3221–3245.
[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of ICLR*. 8.
[31] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed graph clustering: A deep attentional embedding approach. In *Proceedings of IJCAI*. 3670–3676.
[32] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of CIKM*. 889–898.
[33] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of SIGKDD*. 1225–1234.
[34] Xiao Wang, Di Jin, Xiaochun Cao, Liang Yang, and Weixiong Zhang. 2016. Semantic community identification in large attribute networks. In *Proceedings of AAAI*. 265—-271.
[35] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. In *Proceedings of ICML*. 6861–6871.
[36] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Proceedings of IJCAI*. 2111–2117.
[37] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of SIGKDD*. 974–983.
[38] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed graph clustering via adaptive graph convolution. In *Proceedings of IJCAI*. 4327–4333.
[39] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

A MORE DETAILS ABOUT THE EXPERIMENTS

Here we describe more details about the experiments to help in reproducibility.

A.1 Hardware and Software Configurations

All experiments are conducted on a server under the same environment.

Hardware:

- Operating System: Ubuntu 18.04.3 LTS
- CPU: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
- GPU: GeForce RTX 2080 Ti

Software:

- Python 3.7.4
- PyTorch 1.3.1
- sklearn 0.21.3

A.2 Hyperparameter Settings

We report our hyperparameter settings in Table 5.

Table 5: Hyperparameter settings, where n is number of nodes in the dataset.

Dataset	t	r_{pos}^{st}/n^2	r_{pos}^{ed}/n^2	r_{neg}^{st}/n^2	r_{neg}^{ed}/n^2
Cora	8	0.0110	0.0010	0.1	0.5
Citeseer	3	0.0015	0.0010	0.1	0.5
Wiki	1	0.0011	0.0010	0.1	0.5
Pubmed	35	0.0013	0.0010	0.7	0.8