New Deep Neural Networks for Unsupervised Feature Learning on Graph Data

by

Hongchang Gao

B.S., Ocean University of China, 2011

M.S., Beihang University, 2014

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2020

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Hongchang Gao

It was defended on

June 19, 2020

and approved by

Heng Huang, Ph.D., John A. Jurenko Endowed Professor, Department of Electrical and Computer Engineering

Zhi-Hong Mao, Ph.D., Professor, Department of Electrical and Computer Engineering

Wei Gao, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Liang Zhan, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Wei Chen, Ph.D., Associate Professor, School of Medicine, Department of Pediatrics

Dissertation Director: Heng Huang, Ph.D., John A. Jurenko Endowed Professor, Department of Electrical and Computer Engineering

New Deep Neural Networks for Unsupervised Feature Learning on Graph Data

Hongchang Gao, PhD

University of Pittsburgh, 2020

Graph data are ubiquitous in the real world, such as social networks, biological networks. To analyze graph data, a fundamental task is to learn node features to benefit downstream tasks, such as node classification, community detection. Inspired by the powerful feature learning capability of deep neural networks on various tasks, it is important and necessary to explore deep neural networks for feature learning on graphs. Different from the regular image and sequence data, graph data encode the complicated relational information between different nodes, which challenges the classical deep neural networks. Moreover, in real-world applications, the label of nodes in graph data is usually not available, which makes the feature learning on graphs more difficult. To address these challenging issues, this thesis is focusing on designing new deep neural networks to effectively explore the relational information for unsupervised feature learning on graph data.

First, to address the sparseness issue of the relational information, I propose a new proximity generative adversarial network which can discover the underlying relational information for learning better node representations. Meanwhile, a new self-paced network embedding method is designed to address the unbalance issue of the relational information when learning node representations. Additionally, to deal with rich attributes associated to nodes, I develop a new deep neural network to capture various relational information in both topological structure and node attributes for enhancing network embedding. Furthermore, to preserve the relational information in the hidden layers of deep neural networks, I develop a novel graph convolutional neural network (GCN) based on conditional random fields, which is the first algorithm applying this kind of graphical models to graph neural networks in an unsupervised manner.

Table of Contents

List of Tables

List of Figures

Preface

First and foremost, I would like to thank my advisor, Dr. Heng Huang, for all his advice, encouragement, and support throughout my PhD study. Dr. Huang is an ideal advisor in every respect. He is knowledgeable and deep in machine learning. His vision and insight in research always inspire me when exploring new ideas. His extensive experience in research and education helps me grow as a better machine learning researcher. The knowledge I learned from him will be invaluable to me in conducting independent research. I will carry these knowledge into my future career. I am also very grateful for his comprehensive advice and strong support on the development of my future career.

I would also like to thank my dissertation committee members, Dr. Zhi-Hong Mao, Dr. Wei Gao, Dr. Liang Zhan, and Dr. Wei Chen, for their patience, insightful questions and constructive feedback on the dissertation proposal and defense.

Thanks to all members in CSL lab. I feel so lucky to work with so many excellent machine learning researchers, Dr. Feiping Nie, Dr. Wenhao Jiang, Dr. Peng Li, Dr. Hong Chen, Dr. Bin Gu, Dr. Feng Zheng, Dr. Lei Luo, Dr. Feihu Huang. Thanks for their insightful suggestions and comments. I am also grateful to my fellow doctoral students, De, Xiaoqian, Kamran, Zhouyuan, Guodong, Yanfu, Shangqian, An, Runxue, Wenhan, Alireza, Junyi, Zhengmian, for all the great time we had together.

Last, but certainly not least, I want to thank my family for their eternal love. Without their endless encouragement and support, none of this would have been possible. I would also like to thank my fiancèe, Han Ma. Thanks for her love and support during my PhD study. Thanks for everything that she did for me.

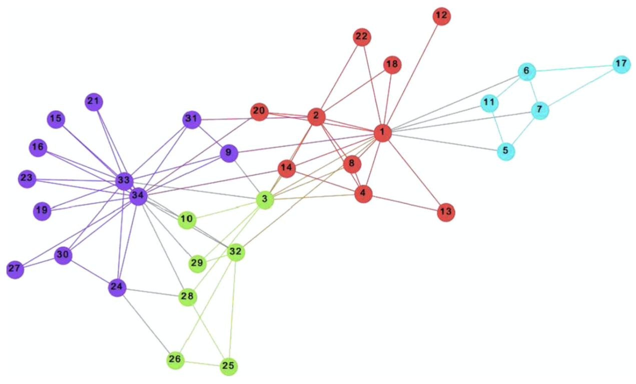
1.0 Introduction

1.1 Feature Learning on Graphs

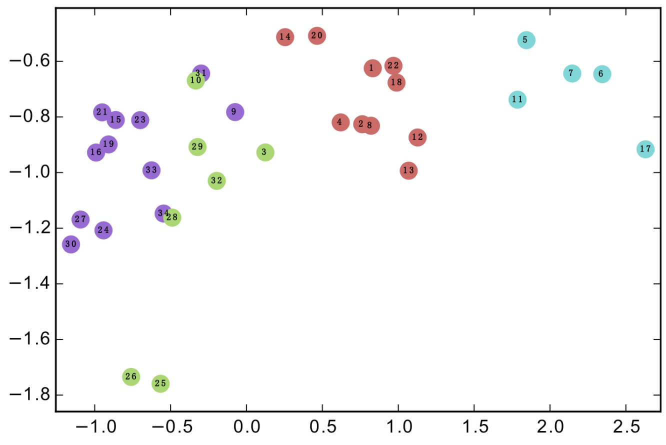
Graph (or Network), a data structure that describes the pairwise relationship between nodes, is very common in a wide variety of areas. For instance, in social science, people connect with each other, forming the social network. In e-commence, customers and items constitute a bipartite graph. In IoT, connected devices compose the IoT network. In chemistry, compound can be represented as a molecular graph. In the biology domain, there are brain networks and protein-protein interaction networks. Due to the abundance of graphs, analyzing graphs for desired tasks is important and attracts increasing attention in recent years.

For graph data, there are a wide range of learning tasks, such as node classification, link prediction, community detection, etc. Specifically, node classification is to predict the type of a given node. For example, in a collaboration network, we can predict the research area of a given researcher by doing node classification. Link prediction is to predict whether two nodes are linked. It can be applied to the recommender system, such as friend recommendation in a social network. Community detection is to identify densely linked clusters of nodes. For example, in a brain network, we can identify the connectome module which potentially associate to specific brain functions. To facilitate these learning tasks on graphs, a critical step is to learn effective node features for the complicated graph. In many real-world applications, the raw feature of a node is noisy or not available. Without effective node features, it is difficult to conduct the aforementioned learning tasks on graphs. Thus, it is necessary to learn effective node features to benefit the learning tasks on graphs.

Learning node features is to learn the low-dimensional representation for each node in a graph. However, it is a challenging task. On one hand, unlike the regular data that are assumed to be independent to each other, a graph encodes complex relationship between different nodes. For instance, in a social network, two users might share similar interests or friend lists. To learn an effective representation, it is necessary to capture and preserve these kinds of relationships in the low-dimensional space. Just as shown in Figure 1, the learned low-dimensional node representations in Figure 1(b) should approximately preserve the relationship between different nodes of the original graph in Figure 1(a). On the other hand, in real-world applications, the node's label is usually not available and difficult to obtain. For example, it is prohibitive to label all users in a social network like Facebook. Without node labels, learning low-dimensional node representations become more difficult. To address these challenging issues, in this thesis, I will put forward unsupervised feature learning algorithms for graph data.



(a) Graph



(b) Learned node features

Figure 1: An illustration of feature learning on the graph. Figure is from [18].

1.2 Feature Learning with Deep Neural Networks

Recent decades have witnessed the rapid development of deep learning. In particular, the seminal work [51] employs deep convolutional neural network to learn image representation for classification and achieves significant improvement over traditional methods. From then on, deep neural networks have been widely used in a wide variety of fields. For instance, convolutional neural networks (CNN), such as AlexNet [51], ResNet [37], MaskRCNN [36], have shown impressive performance on the image data. They have been widely applied to image classification, autonomous driving, and disease detection. In particular, ResNet and its variants even outperform human beings in the ImageNet [21] classification task. Moreover, deep learning also has shown impressive performance on the language data. For example, the transformer network [85], BERT [22] have been widely used in many tasks, such as machine translation, chat robot. Besides that, deep learning has been applied to the sequential decision, such as the robotic manipulation, game playing, automated trading. Especially, AlphaGo [80] outperforms the best human Go player.

Considering the great success of deep neural networks on image data, language data, and sequential data, a natural question is how to have a well-performing deep neural network for graphs, such that it can learn good node representations for the graph learning tasks, such as node classification, link prediction, community detection, etc. However, in spite of achieving great success of deep neural networks on the aforementioned data, how to apply deep neural networks to graph data is still challenging. Unlike image or language data which have a regular grid or line structure, graph data have no such structures so that these networks cannot be applied to graph data directly.

In fact, the success of deep neural networks can be attributed to the powerful feature learning capability to deal with the intrinsic structure of data. In particular, as for image data, the widely used model is the convolutional neural network. It uses the small filter to capture the local correlation of image pixels. Then, CNN can capture edges and shapes of objects for prediction. As for the language data, the commonly used model is recurrent neural networks (RNN) or attention neural networks, which can capture the dependence between different words in a sentence, which is also helpful for prediction. Here, it can bee seen that both of them are designed for dealing with the intrinsic structure of data. Inspired by the aforementioned observation, to have an effective deep neural network for graphs, we also need to deal with the intrinsic structure of the graph data. Compared with image data and sequence data whose samples have a simple relationship, nodes in a graph have complicated relationships between each other, which is encoded by edges. Therefore, to design a deep neural network for learning node representations, it is necessary to deal with this kind of relational information between different nodes. Actually, it is consistent with the requirement of capturing and preserving node relationships in the previous subsection.

Other than dealing with the relational information when designing deep neural networks for graph data, another challenge is the absence of the supervised information. It is well known that a critical reason for the big success of deep neural networks is the availability of large-scale labeled data. For instance, the availability of ImageNet facilitates the development of convolutional neural networks significantly. Considering that, without supervised information, it is difficult to learn node representations with deep neural networks. On the other hand, as we discussed earlier, it is difficult to obtain the label of nodes in real-world applications. Thus, it is necessary to explore designing deep neural networks to learn node representations without node labels.

1.3 Challenges and Contributions

Based on the aforementioned discussion, in this thesis, I will put forward new deep neural networks for learning node representations in an unsupervised manner. In particular, this thesis will focus on how to capture and preserve the relational information to learn good node representations. To this end, I formulate four research questions, which challenge classical deep neural networks and constitute the main contributions of this thesis.

Challenge 1: *How to learn good node representations when the relational information is sparse?*

The task of learning node representations can be decoupled into discovering the proximity in the original space and preserving it in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, it is critical to discover the proximity between different nodes to learn good node representations. Due to the number of discovered links in a graph is limited, the relational information which is represented by the adjacency matrix of the graph is very sparse. Thus, these links are not sufficient to learn good node representations.

To address the sparseness issue, in Chapter 3, I proposed a novel proximity generative adversarial network (ProGAN) to generate links whose distribution is an approximation to that of true links. With this generative model, the underlying relationship can be approximately discovered by the generated links. Then, with both discovered and generated links, the sparseness issue will be alleviated so that the low-dimensional representation of nodes can preserve the relationship between different nodes well. Extensive experimental results have verified the effectiveness of ProGAN. To the best of our knowledge, this is the first work using the generative model to address the sparseness issue of the relation.

Challenge 2: *How to learn good node representations when the relational information is unbalanced?*

When learning the low-dimensional representation of nodes of a graph, it is usually formulated as a contrastive learning problem where the reference node and its neighbouring node compose the positive pair while the reference node and its non-neighbouring node compose the negative pair. However, the positive and negative pairs are severely unbalanced where negative pairs are much more than positive ones. When sampling the negative context nodes to compose the negative pair, existing methods usually employ a predefined sampling distribution based on the node popularity. This sampling distribution often fails to capture the real informativeness of each node and cannot reflect the training state.

To address this unbalance issue, in Chapter 4, I proposed a novel sampling method which can dynamically select negative context nodes. With this dynamic sampling method, the negative context nodes can be selected gradually in terms of their difficulty in the course of training. In addition, to better model the sampling distribution, I also proposed a novel generative adversarial network based sampling method which has a large model capacity for dynamic sampling. Consequently, the unbalance issue is alleviated significantly.

Challenge 3: *How to learn good node representations when nodes have rich attributed information?*

Unlike the plain network where only the topological structure is available, nodes of attributed networks possess rich attributed information. These informative attributes can benefit network analysis. Thus, it is important and necessary to learn node representations based on both the topological structure and node attributes.

In Chapter 5, I proposed a novel deep attributed network embedding approach, which can capture the high non-linearity and preserve various proximities in both topological structure and node attributes. At the same time, a novel strategy is proposed to guarantee the learned node representation can encode the consistent and complementary information from the topological structure and node attributes. Extensive experiments on benchmark datasets have verified the effectiveness of our proposed approach.

Challenge 4: *How to preserve the relational information in the hidden layers of graph convolutional neural networks when learning node representations?*

Compared with the generic data, the graph data possess the similarity information between different nodes. Thus, when applying graph neural networks to the graph data, it is necessary to preserve the relational information at each layer of graph neural networks. Otherwise, the relationship between different nodes in the output layer will be violated. But it is challenging to enforce the hidden layers to preserve the similarity relationship.

In Chapter 6, to address this issue, I proposed a novel conditional random field enhanced graph convolutional neural network. The conditional random field is designed as a layer which is easy to be inserted after each hidden layer of graph neural networks to preserve the relational information. We applied this novel CRF layer to different graph neural networks and obtained significant improvement over them. To the best of our knowledge, this is the first work that applies conditional random field to graph neural networks in an unsupervised way.

2.0 Background

2.1 Traditional Graph Embedding Methods

Unsupervised feature learning on graphs can be traced back to the graph-based dimensionality reduction method. In particular, graph-based dimensionality reduction methods belong to the manifold learning, which aims to preserve the topological structure when learning low-dimensional node representations. Representative methods include locally linear embedding (LLE) method [75], Laplacian eigenmaps (LE) method [5], locality preserving projection (LPP) method [39], etc. In the following, we will give a brief overview of these classical methods.

2.1.1 Locally Linear Embedding Method

Locally linear embedding method is first proposed in [75]. It assumes data points are drawn from some underlying manifold where each data point and its neighbours lie on a locally linear patch of a manifold [75]. Based on this local linearity assumption, LLE constructs the graph by optimizing the following problem:

$$\min_{W} \sum_{i=1}^{n} \|x_i - \sum_{j=1}^{n} w_{ij} x_j\|^2 \ , \tag{2.1}$$

where $x_i \in \mathbb{R}^d$ denotes the data point, $W = [w_{ij}] \in \mathbb{R}^{n\times n}$ represents the edge weight between x_i and x_j. After obtaining the adjacency matrix W, based on the local linearity assumption, LLE learns the low-dimensional representation by optimizing the following problem:

$$\min_{U} \sum_{i=1}^{n} \|u_i - \sum_{j=1}^{n} W_{ij} u_j\|^2 \ , \tag{2.2}$$

where $U = [u_1, u_2, \cdots, u_n] \in \mathbb{R}^{d' \times n}$ $(d' < d)$ denotes the low-dimensional representation of data points. Its optimal solution can be easily obtained by conducting eigendecomposition.

2.1.2 Laplacian Eigenmaps Method

Similar with LLE, Laplacian eigenmaps method [75] also aims to preserve the local structure when learning low-dimensional representations. Different from LLE, it doesn't construct the graph by minimizing the reconstruction error. Instead, LE employs some heuristic methods to construct the graph, such as Gaussian kernel. After that, it optimizes the following problem:

$$\min_{U} \sum_{i=1}^{n} \sum_{j=1}^{n} \|u_i - u_j\|^2 w_{ij} \ . \tag{2.3}$$

Intuitively, when the edge weight $w_{ij} > 0$ is large, x_i and x_j are similar. Then, optimizing this objective function will push u_i and u_j to be close to each other. As a result, the proximity can be preserved in the low-dimensional space. This objective function can also be solved by conducting eigendecomposition.

2.1.3 Locality Preserving Projection Method

Locality preserving projection method [39] is a linear method to learn low-dimensional representation based on the graph. In particular, it assumes that there exists a linear projection matrix $P \in \mathbb{R}^{d' \times d}$ such that $U = PX$. Besides that, it employs the same procedure as LE method to construct the graph and optimizes the follows problem:

$$\min_{P} \sum_{i=1}^{n} \sum_{j=1}^{n} \|Px_i - Px_j\|^2 w_{ij} \ . \tag{2.4}$$

It can be seen that this objective function also pushes the low-dimensional representation Px_i and Px_j to be close to each other if x_i and x_j are similar. By optimizing this objection function with eigendecomposition, we can get the projection matrix P so that the low-dimensional representation can be easily obtained.

In summary, these traditional unsupervised methods learn low-dimensional representations by preserving the local structure of the data. However, they are not sufficient. On one hand, they cannot capture and preserve the complicated relational information. In particular, they only use the edge weight to measure the proximity between two data points. However, there exist some other proximities between two data points. For instance, if two data points are not connected but they share similar neighboring data points, we can still view they are similar to each other. In fact, this phenomenon is common in social network. Two users have similar friends are also tending to be similar to each other. Thus, it is necessary to capture and preserve more complicated relational information when learning low-dimensional representations. On the other hand, traditional methods have large computational overhead since they involve the time-consuming eigendecomposition operation whose time complexity is $O(n^3)$ where n is the number of data points. Thus, they are not applicable for large-scale applications.

2.2 Network Embedding

With the emergence of large-scale graph data, a lot of effort has been made to propose new feature learning methods for graphs. In particular, network embedding has attracted a lot of attention in the past few years. In this subsection, we will give a brief overview about a commonly used contrastive-learning-based network embedding schema, which makes unsupervised feature learning for large-scale and featureless networks feasible.

The contrastive-learning-based network embedding schema was first proposed in DeepWalk [72]. Its goal is to capture and preserve the rich relational information when learning low-dimensional node representations. To this end, DeepWalk employs random walk over the network to capture the rich relational information between different nodes. In particular, for each node, DeepWalk uses random walk to get a node sequence $s = \{v_1, v_2, \cdots, v_s\}$ where v_i denotes the node. Then, it employs a sliding window with a specific window size to slide on the node sequence. If two nodes appear in the window simultaneously, they are viewed to be similar two each other. Intuitively, this operation can be viewed to augment the adjacency matrix by adding more edges to it. Thus, compared with traditional methods in the last subsection, this augmented adjacency matrix can capture high-order relational information for learning better node representations.

To augment the adjacency matrix, there are different approaches. For instance, DeepWalk utitlizes the regular random walk. Node2Vec [33] argues that DeepWalk is not capable of capturing the diversity of relational information. To address this issue, Node2Vec proposed a biased random walk to combine the breadth-first sampling and depth-first sampling. In this way, it can capture both the global and local relational information. LINE [83] proposed the second-order proximity to augment the adjacency matrix. In particular, LINE argues that the original edges in the adjacency matrix represent the first-order proximity, while the second-order proximity is the proximity between the context of nodes. In other words, if two nodes share similar neighbours, they are similar. Based on this argument, LINE employs the second-order proximity to augment the adjacency matrix.

After obtaining the augmented adjacency matrix, the contrastive-learning-based network embedding schema optimizes the following objective function to learn low-dimensional node representations:

$$\max \log p(v_p|v_i) + \sum_{j \in \mathcal{N}_{v_i}} \log(1 - p(v_j|v_i)) \,, \tag{2.5}$$

where v_i denotes the anchor node, v_p is the positive context node of v_i which is indicated by the edges in the adjacency matrix, v_j denotes the negative context node of v_i which means that there are no edges between v_i and v_j, and $\mathcal{N}_{v_i}$ stands for all the negative context nodes of v_i. This problem can be viewed as a binary classification problem where v_i and its neighbour v_p compose the positive pair while v_i and its non-neighbouring node v_j compose the negative pair. By optimizing this objective function, it can push similar nodes together and push dissimilar nodes away to each other. In this way, the relational information can be preserved in the low-dimensional space.

Although the contrastive-learning-based network embedding schema has achieved success in some scenarios, yet it still has some limitations. Firstly, as we discussed earlier, this schema employs the heuristic methods to capture the underlying relational information. Their capacity is limited so that it fails to capture the rich relational information. Secondly, as indicated in Eq. (2.5), we should sample a subset from the negative context of the anchor node to optimize the loss function. However, how to sample the negative context nodes to ease the learning of low-dimensional node representations is challenging. Current methods, such as DeepWalk, only use some predefined sampling distributions, which are not satisfactory. Thirdly, the contrastive-learning-based network embedding schema only focus on the topological structure of the network, failing to explore the rich node attributes. Thus, how to deal with node attributes to capture and preserve the proximity in node attributes is still a challenge.

2.3 Deep Neural Networks

In recent years, deep neural networks have been applied to different areas and achieve good performance. In this subsection, we will give a brief overview about three fundamental deep neural networks which are used in this thesis.

2.3.1 Multi-Layer Perceptron Neural Networks

Multi-Layer Perceptron Neural Networks (MLP) is a basic deep neural network. Its each layer is a fully-connected layer: including linear transformation and non-linear activation operation, which is defined as follows:

$$z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)} \ , \ h^{(l)} = \sigma(z^{(l)}) \ , \tag{2.6}$$

where $h^{(l)} \in \mathbb{R}^{d_l}$ denotes the hidden feature in the l-th layer, $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{d_l}$ are the model parameter in the l-th layer to be learned, $z^{(l)} \in \mathbb{R}^{d_l}$ is the hidden feature before conducting non-linear activation operation, $\sigma(\cdot)$ is the non-linear activation function.

By stacking multiple layers together, MLP is actually a highly non-linear mapping function which maps the input to a new space to be easily predicted. In addition, we can use backpropagation to compute gradients and then use stochastic gradient descent method to iteratively learn the model parameter.

2.3.2 Graph Convolutional Neural Networks

Graph convlutional neural network (GCN) is the architecture designed for the graph data. The classical neural networks like MLP cannot deal with the graph structure, while

GCN can leverage the relational information when learning node representations. In detail, a typical GCN layer is defined as follows:

$$\begin{aligned} Z^{(l)} &= W^{(l)} H^{(l-1)} A \ , \\ H^{(l)} &= \sigma(Z^{(l)}) \ , \end{aligned} \tag{2.7}$$

where $H^{(l)} = [h_1^{(l)}, h_2^{(l)}, \cdots, h_n^{(l)}] \in \mathbb{R}^{d_l \times n}$ is the hidden feature of all nodes in the l-th layer, $Z^{(l)} = [z_1^{(l)}, z_2^{(l)}, \cdots, z_n^{(l)}] \in \mathbb{R}^{d_l \times n}$ is the hidden feature before conducting non-linear activation operation, $\sigma(\cdot)$ is the non-linear activation function, $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the model parameter in the l-th layer. Here, we omit the bias for simplification. Compared with MLP, GCN has an aggregation operation to leverage the relational information, which is denoted by $H^{(l-1)}A$ where $A \in \mathbb{R}^{n \times n}$ determines how to aggregate neighbours. For instance, in the standard GCN [48], it is defined as follows:

$$z_i^{(l)} = \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} W^{(l)} h_j^{(l-1)} \ , \tag{2.8}$$

where $\mathcal{N}_i$ denotes the neighbours of the i-th node and $|\mathcal{N}_i|$ represents the degree of the i-th node. In the graph attention network (GAT) [86], the aggregation matrix is learned from the data for each layer by the following operation:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^T \left[W^{(l)} h_i^{(l-1)} \| W^{(l)} h_j^{(l-1)}\right]\right)\right)}{\sum_{j \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(a^T \left[W^{(l)} h_i^{(l-1)} \| W^{(l)} h_j^{(l-1)}\right]\right)\right)} \ , \tag{2.9}$$

where $a \in \mathbb{R}^{d_l}$ is a learnable vector of parameters, and $\cdot \| \cdot$ denotes the concatenation operation. Then, GAT aggregates the neighbours as follows:

$$z_i^{(l)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l-1)} \ . \tag{2.10}$$

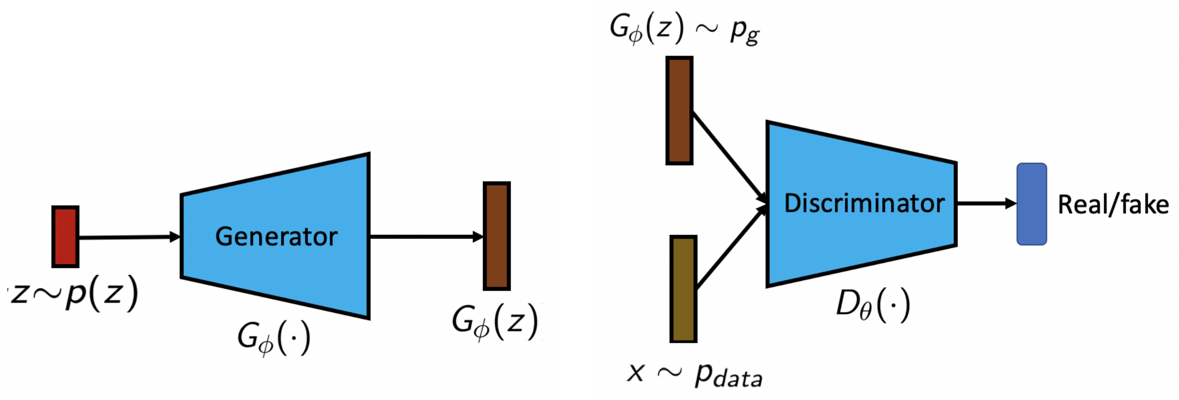
It is obvious that the graph convolutional operation in Eq. (2.8) and Eq. (2.10) can deal with the relational information, compared with the classical MLP in Eq. (2.6).

2.3.3 Generative Adversarial Networks

In recent years, Generative Adversarial Networks (GAN) [32], as an unsupervised learning approach, have attracted a surge of attention. Specifically, GAN is a generative model and its goal is to generate samples as real as true samples. To this end, it has two components: generator and discriminator, just as shown in Figure 2. The generator tries to use a deep neural network to map a simple prior distribution $p(z)$, such as Gaussian distribution, to the complicated data distribution p_g. The discriminator serves as a metric to measure the distance between the true data distribution p_{data} and the generated data distribution p_g. By optimizing the following objective function:

$$\min_{\phi} \max_{\theta} E_{x \sim p_{data}} [\log D_\theta(x)] + E_{z \sim p(z)} [\log (1 - D_\theta (G_\phi(z)))] \ . \tag{2.11}$$

The approximated distribution p_g will approach to the true distribution p_{data}. Then, the generated samples can be viewed drawn from the true data distribution.



(a) Generator

(b) Discriminator

Figure 2: An illustration of GAN.

3.0 Network Embedding via Proximity Generative Adversarial Network

3.1 Introduction

As a fundamental tool to analyze networks, network embedding is to learn a low-dimensional representation for each node in a network. This low-dimensional node representation can preserve the proximity between different nodes of the network. Essentially, a network embedding method includes two phases. The first phase is to *discover* the proximity between different nodes in a network. The second phase is to *preserve* the proximity in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, it is considerably important to discover the underlying proximity in a network well.

To discover the underlying proximity in a network, various proximities have been explored in recent years. Specifically, these proximities include the first-order proximity, second-order proximity, high-order proximity, and so on. In detail, the first-order proximity [83] considers that two nodes are similar if there is an edge between them. However, the discovered edges in a network are usually very sparse so that they are not enough to disclose the proximity between different nodes. The second-order proximity [83] views two nodes similar if they share similar neighbors. Thus, the second-order proximity is supposed to discover more underlying relationships than the first-order proximity. Furthermore, GraRep [12] proposes to discover the high-order proximity by constructing the k-step probability transition matrix explicitly. In this way, the long-distance relationship between two different nodes can be captured. Additionally, DeepWalk [72] employs random walk to discover the high-order proximity. Specifically, it utilizes random walk to get node sequences. For each node sequence, it uses a sliding window to get the neighborhood of its nodes. As a result, the high-order proximity information can be captured [92]. In this section, we will propose a novel method to discover the underlying proximity among different nodes to benefit network embedding.

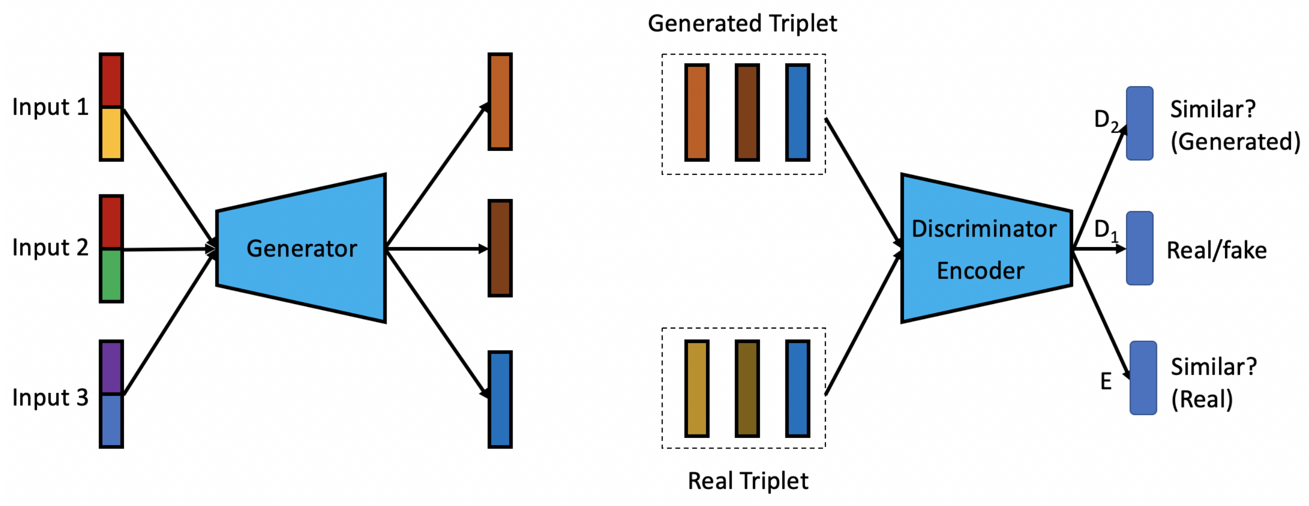


Figure 3: The architecture of our proposed ProGAN. For the generator, Input 1 and Input 2 share a common part so that they will generate similar output, while Input 3 does not share any parts with the other two so that it will generate different output. Best viewed in color. For the discriminator and encoder, the input is real triplet and generated triplet. The discriminator distinguishes real and fake nodes, and discriminates whether two generated nodes are similar. The encoder discriminates whether two real nodes are similar.

In the past few years, generative adversarial networks (GAN) [32] have shown promising results in a wide variety of tasks, such as image generation [100], text generation [96]. The basic idea of GAN is to learn a map which can transform a noise from a simple distribution to a sample from a complicated distribution. This map is usually constructed by a deep neural network which has large expressivity. Inspired by the development of GAN, we propose to use GAN to generate proximities between different nodes to approximate the underlying proximity. In this way, the generated proximity can capture the underlying relationship between different nodes, which will benefit the network embedding. However, the standard GAN is used to generate samples rather than proximities. Thus, how to generate the proximity is challenging.

To address the aforementioned problems, in this chapter, we propose a novel proximity generative adversarial network (ProGAN) for network embedding. Specifically, to capture the underlying proximity, ProGAN tries to generate proximities to approximate the distribution of the real proximity. In this way, undiscovered proximities can be captured by the generated ones. Furthermore, to generate the proximity, we instantiate it to the relationship of a triplet of nodes, in which the positive node is similar to the reference node while the negative node is dissimilar with that. By generating this kind of triplets, ProGAN can capture the underlying proximity. Moreover, to generate the desired triplet, we propose the novel neural network architecture as shown in Figure 3. With this novel architecture, ProGAN can generate similar nodes from similar input noise and dissimilar nodes from dissimilar input noise. Then, with these generated triplets and real triplets, ProGAN trains an encoder to learn node representations, which can preserve both the real proximity and the generated proximity. At last, extensive experimental results have verified the performance of our proposed ProGAN.

3.2 Related Works

As a fundamental tool to analyze networks, various network embedding methods have been proposed in recent years. In this part, we will give a brief review on these related works.

In terms of the availability of node attributes, existing methods can be categorized into two classes: *plain network embedding* and *attributed network embedding*. Plain network embedding methods [72, 33, 83, 12, 74] only employ the topological structure to learn node representations, while attributed network embedding methods [92, 42, 43, 71, 48, 35, 25] utilize both the topological structure and node attributes.

The seminal DeepWalk [72] formulates network embedding as word embedding. Specifically, it employs random walk on the network to get node sequences which can be viewed as sentences. With these node sequences, DeepWalk then utilizes the Skip-Gram model to learn node representations. Later, Node2Vec [33] proposes a biased random walk method, which combines the breadth-first and depth-first sampling to preserve the local and global proximity, to construct the context of nodes. Afterwards, LINE [83] is proposed to preserve the first-order and second-order proximity when learning the node representation. GraRep [12] aims to preserve the high-order proximity. Specifically, two nodes having similar k-step neighbors should have similar latent representations. However, all these methods ignore node attributes when learning the low-dimensional node representation.

In an attributed network, nodes usually possess rich information. By exploring such kind of rich attributed information, the proximity among different nodes can be discovered and preserved better when learning node representations. For instance, [92] proposes to incorporate both topological structure and node attributes by using inductive matrix factorization method. [42] proposes to factorize the attribute matrix and regularize the factorization by the toplogical structure. Recently, [25] utilizes a multi-modal auto-encoder to combine topological structure and node attributes to learn node representations. By incorporating node attributes, these methods have shown some improvement over the counterpart which only employs the topological structure.

However, most of aforementioned methods are shallow methods, failing to capture the highly non-linear property in a network. Networks in real-world applications are usually complicated and highly non-linear. Thus, it is important to capture the high non-linearity in the network.

Recent developments of deep neural networks have accelerated much progress in data mining and machine learning. Deep neural networks have much larger expressivity to capture the highly non-linear property of the data than shallow methods. An example is the seminal work [51] which adopts deep convolutional neural network to learn image representation for classification and achieves significant improvement over traditional methods. From then on, deep neural network has been widely used in a wide variety of applications.

In recent years, some works [14, 87, 48, 25, 8, 35, 29, 27] have been proposed to apply deep neural networks for network embedding. For example, [14] proposes a heterogeneous network embedding method. This method focuses on heterogeneous data embedding, such as image-text embedding or text-text embedding, ignoring to explore the network structure. The work [87] employs deep neural network for network embedding, in which an autoencoder is utilized to learn the low-dimensional representation for each node. This model can discover the highly non-linear structure in the network while preserving the similarity. In addition, [48] proposes a semi-supervised network embedding method based on graph convolutional neural network by combining the topological structure and the node attribute. Recently, [8] utilizes a multi-layer perceptron neural network to map each node in the network to a Gaussian distribution, which can capture the uncertainty of the learned representation.

Recently, a variant of deep neural networks, generative adversarial network (GAN) [32], has attracted much attention due to its impressing performance on the unsupervised task. Its basic idea is to learn a map which can transform a simple distribution to a complicated distribution. Specifically, GAN includes a generator and a discriminator. The generator tries to generate samples as real as possible while the discriminator is to distinguish the generated samples and the real ones. Formally, the objective function of GAN is defined as follows:

$$\min_{\phi} \max_{\theta} E_{x \sim p(x)}[\log D_{\theta}(x)] + E_{z \sim p(z)}[\log(1 - D_{\theta}(G_{\phi}(z)))] \ , \tag{3.1}$$

where $p(x)$ denotes the real data distribution and $p(z)$ represents the simple prior distribution. $D_{\theta}(\cdot)$ corresponds to the discriminator while $G_{\phi}(\cdot)$ corresponds to the generator. By optimizing this objective function, the generator $G_{\phi}(\cdot)$ can learn a map to transform the prior distribution $p(z)$ to the complicated data distribution $q(x)$ where $q(x)$ is the approximation to the real data distribution $p(x)$.

Due to the impressive performance of GAN, some researchers have tried to apply GAN for network embedding. For instance, [88] proposes GraphGAN to learn low-dimensional node representations. Specifically, network embedding is to push similar nodes together and push away dissimilar nodes. When pushing away dissimilar nodes, it needs to sample negative nodes for the reference node. GraphGAN employs a generator to generate the sampling distribution to sample the negative nodes, benefiting the embedding result. [19] proposes the adversarial network embedding, which utilizes the adversarial technique to regularize the learned representation. In this chapter, we will exploit GAN to discover the underlying proximity to benefit node embedding.

3.3 Network Embedding via Proximal Generative Adversarial Network

3.3.1 Problem Definition

Let $G = \{V, W, X\}$ denote an attributed network. $V = \{v_i\}_{i=1}^{n}$ is a set of n nodes. $W = [w_{ij}] \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix. $w_{ij} = 1$ denotes there exists an edge between node v_i and node v_j. Otherwise, $w_{ij} = 0$. $X = [x_{ij}] \in \mathbb{R}^{n \times d}$ represents the attribute matrix. The i-th row $X_{i\cdot} \in \mathbb{R}^{d}$ denotes the attribute of node v_i. In this chapter, we will focus on the attributed network embedding.

Network embedding is to learn a low-dimensional representation for each node v_i from the topological structure W and attributes X. Additionally, the low-dimensional representation should preserve the proximity between different nodes. In other words, in the low-dimensional space, a network embedding method should push similar nodes together and push dissimilar nodes away. Therefore, the proximity between different nodes can be instantiated to the triplet $\langle v_i, v_j, v_k \rangle$ where v_i denotes the reference node, v_j represents the positive node which is similar with v_i, v_k stands for the negative node which is dissimilar with v_i. Therefore, we have the following formal definition.

Definition 1. *Network embedding is to learn a map $f : \{W, X\} \mapsto E$ where $E \in \mathbb{R}^{n \times d'}$ denotes the low-dimensional representation. Meanwhile, given a triplet $\langle v_i, v_j, v_k \rangle$ in the original space such that*

$$sim(v_i, v_j) > sim(v_i, v_k) \ , \tag{3.2}$$

the learned low-dimensional representation should guarantee

$$sim(E_{i\cdot}, E_{j\cdot}) > sim(E_{i\cdot}, E_{k\cdot}) \ , \tag{3.3}$$

where $sim(\cdot,\cdot)$ *denotes the proximity between two data points,* $E_{i\cdot}$ *represents the low-dimensional embedding of the i-th node* v_i*.*

This definition indicates that the task of network embedding can be decoupled into *discovering* the proximity in the original space and *preserving* it in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, discovering the proximity between different nodes is critical to learn a good node representation. However, the edges in a network are very sparse so that they are not enough to disclose the proximity between different nodes. Existing network embedding algorithms proposed various methods to discover underlying proximities, such as first-order proximity, second-order proximity, and so on. However, most of them fail to fully utilize both the topological structure and node attributes. To address these problems, we will propose a novel proximity generative adversarial network to discover the underlying proximity among different nodes, benefiting network embedding.

3.3.2 Proximity Generative Adversarial Network

Unlike existing methods which discover proximities among different nodes, we propose to generate the proximity. But how to generate it is challenging. Inspired by the development of generative adversarial networks, we propose to employ GAN to generate the underlying proximity. However, the standard GAN is designed to generate samples rather than proximities. Thus, how to generate the underlying proximity is also challenging. To address these challenges, we propose a novel proximity generative adversarial network (ProGAN).

As discussed in the previous subsection, the proximity can be instantiated to the triplet $\langle v_i, v_j, v_k\rangle$ such that $sim(v_i, v_j) > sim(v_i, v_k)$. Thus, the key idea of generating the proximity is to generate triplets. Formally, assume the true distribution of the triplet is denoted by $P(v_i, v_j, v_k)$ which is unknown. Our task is to learn a distribution $Q(v_i, v_j, v_k)$ to approximate $P(v_i, v_j, v_k)$. Since $Q(v_i, v_j, v_k)$ is similar with $P(v_i, v_j, v_k)$, the generated triplets from $Q(v_i, v_j, v_k)$ can disclose the real proximity. Thus, by generating triplets, ProGAN can discover the underlying complicated proximity between different nodes. In addition, when generating the triplet to get the proximity, it is supposed that the generated individual nodes in the triplet are as real as the true nodes. In other words, our another task is to learn the distribution $Q(v_i)$ to approximate the true node distribution $P(v_i)$. Moreover, after discovering the underlying proximity, we should preserve it when learning the low-dimensional representation. To this end, ProGAN includes three components: generator, discriminator, and encoder. Generator is expected to generate the desired triplet and node. Discriminator needs to discriminate them from the real ones. Encoder is to learn the low-dimensional representation for each node by preserving the real and generated proximity. In the following part, we will describe the detail of these three components.

Generator To approximate the distribution $P(v_i, v_j, v_k)$, the generator needs to generate the triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ such that $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$ where $\hat{v}_i$ denotes the generated nodes. But how to generate this kind of triplets? Here, we consider a generator $\hat{v} \sim G(z_1, z_2)$ where z_1 corresponds to the first input noise while z_2 corresponds to the second input noise. Comparing with the standard GAN which has only one input noise z, our proposed ProGAN decouples the input noise into two parts: z_1 and z_2. In this way, our objective is to learn the generator such that varying z_1 or z_2 can control the similarity of two generated nodes. Doing so allows us to generate the desired triplets.

In detail, to generate two similar nodes $(\hat{v}_i, \hat{v}_j)$, ProGAN enforces their input noise to share a common z_1. Otherwise, there is no such a constraint. Thus, to generate the desired triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ such that $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$, the input to the generator is (z_{i_1}, z_{i_2}), (z_{i_1}, z_{j_2}), and (z_{k_1}, z_{k_2}) respectively. Apparently, $\hat{v}_i$ and $\hat{v}_j$ share the same input noise z_{i_1}, then they are expected to be similar with each other. In addition, $\hat{v}_i$ and $\hat{v}_j$ does not share the second input noise, then they will not be exactly same with each other. On the other hand, $\hat{v}_i$ and $\hat{v}_k$ do not share any common input, thus they are not expected to be similar.

On the other hand, to approximate the distribution $P(v_i)$, we employ the same generator $G(z_1, z_2)$. When concatenating the two input z_1 and z_2 together, this generator acts as the standard one in GAN to generate nodes. Thus, with this proposed generator $G(z_1, z_2)$, we can generate not only the desired node $\hat{v}_i$ but also the triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$. Since the generated nodes are supposed to be similar with real nodes and the generated triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ is also similar to real triplets, the proximity between generated nodes should also be similar with real nodes. Thus, by generating triplets $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$, we can discover the underlying proximity between different nodes.

Discriminator To learn the aforementioned generator, the standard discriminator of GAN is not enough, since it cannot disentangle the latent space. As a result, there is no possibility to control the similarity of generated nodes. The discriminator of ProGAN is supposed to complete two goals. The first goal is to distinguish the generated nodes and the real ones. This is same as the standard GAN, which will guide the generator to generate nodes as real as possible. The second goal is to determine whether the generated triplet satisfies $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$. In other words, the discriminator should determine whether two generated nodes are similar or not. In this way, the generated triplet can encode the proximity information. As a result, the discriminator will guide the generator to approximate two distributions $P(v_i)$ and $P(v_i, v_j, v_k)$.

For the discriminator, we need real nodes so that we can distinguish the real and generated ones. But how to represent them? Here, in this chapter, we employ the node attribute $X_{i\cdot}$ to represent the real node v_i. Then, the triplet can be represented in the same way. Correspondingly, the generator should generate node attributes $\hat{X}_{i\cdot}$. In other words, to fool the discriminator, the generator will learn a distribution $Q(\hat{X}_{i\cdot})$ to approximate the real node distribution $P(X_{i\cdot})$. Furthermore, the discriminator will also guide the generator to learn a distribution $Q(\hat{X}_{i\cdot}, \hat{X}_{j\cdot}, \hat{X}_{k\cdot})$ to approximate $P(X_{i\cdot}, X_{j\cdot}, X_{k\cdot})$.

Encoder After obtaining the proximity between different nodes, we can use it to learn the low-dimensional representation for each node. To do that, the proximity should be preserved in the low-dimensional space. As we discussed early, the discovered proximity is not enough to learn a good representation. Thus, we should utilize both the discovered proximity and the underlying one.

For the discovered proximity, we also resort to the triplet. Specifically, to construct the real triplet $\langle X_{i\cdot}, X_{j\cdot}, X_{k\cdot} \rangle$ such that $sim(X_{i\cdot}, X_{j\cdot}) > sim(X_{i\cdot}, X_{k\cdot})$, we use the following steps. At first, we randomly select the reference node $X_{i\cdot}$. Then, $X_{j\cdot}$ is selected from $\{j | w_{ij} = 1\}$ while $X_{k\cdot}$ is selected from $\{k | w_{ik} = 0\}$. In other words, if there exists an edge between two nodes, they are similar. Otherwise, they are dissimilar. For the underlying proximity, we directly utilize the generated triplet $\langle \hat{X}_{i\cdot}, \hat{X}_{j\cdot}, \hat{X}_{k\cdot} \rangle$ such that $sim(\hat{X}_{i\cdot}, \hat{X}_{j\cdot}) > sim(\hat{X}_{i\cdot}, \hat{X}_{k\cdot})$. With these proximities, we expect $sim(E_{i\cdot}, E_{j\cdot}) > sim(E_{i\cdot}, E_{k\cdot})$ and $sim(\hat{E}_{i\cdot}, \hat{E}_{j\cdot}) > sim(\hat{E}_{i\cdot}, \hat{E}_{k\cdot})$ in the low-dimensional space. In this way, we can push similar nodes together and push away dissimilar nodes in the low-dimensional space.

Since the second goal of the discriminator is also push similar generated nodes together and push away dissimilar generated nodes, the encoder can share the same architecture with the discriminator. The only difference is that the discriminator has one more function to distinguish the real nodes and the generated ones. Therefore, in this chapter, we use a single neural network for both the discriminator and encoder. Based on this common neural network, we add one more branch at the last layer for the discriminator to differentiate the generated nodes from the real ones.

At last, all of these three components are implemented by deep neural networks to capture the highly non-linear property in a network. In summary, the generator can generate the desired triplet to discover the underlying proximity. The discriminator tries to guide the generator to generate nodes and triplets to be similar as real ones. With the discovered proximity and the synthesized underlying proximity, the encoder employs a deep neural network to learn low-dimensional node representations. Comparing with existing methods, the difference lies in that we employ our proposed ProGAN to synthesize the underlying proximity rather than construct it by heuristic strategies.

Loss Function To learn the three components in ProGAN, we are going to define the objective function. Overall, it includes the *adversarial loss* and *encoder loss.* The adversarial loss will enforce ProGAN to generate nodes and triplets as real as possible and the encoder loss will push ProGAN to learn a good node representation.

Regarding the adversarial loss, it is designed as a minimax game between the discriminator and the generator, in which the discriminator is trained to distinguish the real and generated nodes while the generator is trained to generate nodes to fool the discriminator. Specifically, the discriminator's loss function is defined as follows:

$$\begin{aligned} \mathcal{L}_{D_1} &= E_{x\sim p(x)}[\log D_1(x)] + E_{z_1\sim p(z), z_2\sim p(z)}[\log(1 - D_1(G(z_1, z_2)))] \ , \\ \mathcal{L}_{D_2} &= E_{z_1\sim p(z), z_2\sim p(z), z_2'\sim p(z)}[\log \sigma(s_1)] + E_{z_1\sim p(z), z_2\sim p(z), z_1''\sim p(z), z_2''\sim p(z)}[\log(1 - \sigma(s_2))] \ , \end{aligned} \tag{3.4}$$

where $s_1 = D_2(G(z_1, z_2))^T D_2(G(z_1, z_2'))$ measures the similarity of two similar nodes while $s_2 = D_2(G(z_1, z_2))^T D_2(G(z_1'', z_2''))$ measures that of two dissimilar nodes. In addition, $\sigma(\cdot)$ denotes the Sigmoid function. Note that the discriminator has two branches as shown in the Figure 3. Here, we use D_1 and D_2 to denote the output of these two branches respectively. Correspondingly, $\mathcal{L}_{D_1}$ implements the first goal of the discriminator to distinguish the real and generated nodes. $\mathcal{L}_{D_2}$ fulfills the second goal to discriminate whether two generated nodes are similar or not. As discussed earlier, $G(z_1, z_2)$ and $G(z_1, z_2')$ share the same input z_1 so that they are considered to be similar. On the contrary, $G(z_1, z_2)$ and $G(z_1'', z_2'')$ do not share any input so that they are supposed to be dissimilar.

Regarding the generator loss, it should guide the generator to fool the discriminator and synthesize similar nodes when given similar input. To this end, the loss function of the generator is defined as follows:

$$\begin{aligned}\mathcal{L}_G = \; & E_{z_1 \sim p(z), z_2 \sim p(z)}[\log(D_1(G(z_1, z_2)))] \\ & + E_{z_1 \sim p(z), z_2 \sim p(z), z_2' \sim p(z)}[\log \sigma(s_1)] \\ & + E_{z_1 \sim p(z), z_2 \sim p(z), z_1'' \sim p(z), z_2'' \sim p(z)}[\log(1 - \sigma(s_2))] \; ,\end{aligned} \tag{3.5}$$

where s_1 and s_2 are defined as the discriminator. Specifically, the first term guides the generator to generate nodes as real as possible. The second and third terms guide the generator to generate similar nodes when given similar input, while generate dissimilar nodes when given different input.

Regarding the encoder loss, it should push similar nodes together and push dissimilar nodes away. In addition, to train the encoder, we utilize both the discovered proximity and the underlying proximity generated by the generator. Therefore, the loss function of the encoder is defined as follows:

$$\begin{aligned}\mathcal{L}_E = \; & E_{(x_1, x_2, x_3) \sim p(x, y, z)}[\log \sigma(E(x_1)^T E(x_2)) \\ & + \log(1 - \sigma(E(x_1)^T E(x_3)))] \\ & + E_{z_1 \sim p(z), z_2 \sim p(z), z_2' \sim p(z)}[\log \sigma(t_1)] \\ & + E_{z_1 \sim p(z), z_2 \sim p(z), z_1'' \sim p(z), z_2'' \sim p(z)}[\log(1 - \sigma(t_2))] \; ,\end{aligned} \tag{3.6}$$

Table 1: Descriptions of benchmark datasets

Dataset	# Nodes	#Edges	#Attributes	#Labels
Citeseer	3,312	4,660	3,703	6
Cora	2,708	5,278	1,433	7
Flickr	7,564	239,365	12,047	9
Blogcatalog	5,196	171,743	8,189	6

where $t_1 = E(G(z_1, z_2))^T E(G(z_1, z_2'))$ measures the similarity of node representations of two similar nodes while that of two dissimilar nodes is measured by $t_2 = E(G(z_1, z_2))^T E(G(z_1'', z_2''))$. Here, the first two terms act on the discovered proximity while the last two terms act on the underlying proximity generated by the generator. Note that the parameter of encoder E is same as the discriminator D_2. Thus, we can train the encoder and the discriminator simultaneously.

By optimizing these objective functions, we can learn the parameter of three components of our ProGAN. Particularly, we can use the stochastic gradient descent method to optimize it easily. At last, we summarize the optimization step in Algorithm 1. Note that step 3 and 4 can be updated simultaneously.

Algorithm 1 Algorithm to optimize ProGAN.

```
1: repeat
2:     Sample the input noise as Eq.(3.5) to optimize the generator loss $\mathcal{L}_G$.
3:     Sample the input noise and real nodes as Eq.(3.4) to optimize the descriminator loss $\mathcal{L}_{D_1} + \mathcal{L}_{D_2}$.
4:     Sample the real triplet and the input noise as Eq.(3.6) to optimize the descriminator loss $\mathcal{L}_E$.
5: until Converges
```

3.4 Experiments

In this chapter, we will conduct extensive experiments to show the performance of our proposed methods.

3.4.1 Dataset Descriptions

Throughout this chapter, four attributed networks are employed, including citation networks, social networks. The details about these datasets are described as follows.

- Citeseer [65] is a paper citation network. Papers from different topics constitute nodes and the citation among them composes edges. We use the content of papers as node attributes and topics as class labels.
- Cora [65] is also a paper citation network. Similarly, nodes are papers from different topics, edges are the citation among different papers, and the node attribute is the bag-of-words representation of the corresponding paper. In addition, the topics that papers belong to are considered as class labels.
- Flickr [42] is a social network where users share their photos. Users follow each other to form a network. Here, the tags on their images are used as the attribute. Additionally, the groups that users joined are considered as class labels.
- Blogcatalog [42] is also a social network where nodes are users and edges are the friendship between different users. The attribute of nodes is the extracted keywords from their blogs. Regarding the class label, we use the predefined categories that blogs belong to.

For all datasets, we normalize its attribute to have a unit length. At last, we summarize the statistics of these networks in Table 1.

3.4.2 Experiment Settings

To verify the performance of ProGAN, we compare it with 9 state-of-the-art methods, which include 5 pure network embedding methods: DeepWalk [72], Node2Vec [33], LINE [83], GraRep [12], GraphGAN [88], and 4 attributed network embedding methods: TADW

Table 2: Node classification result of Citeseer dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5146	0.4604	0.5623	0.5149	0.5830	0.5397
Node2Vec	0.5059	0.4541	0.5744	0.5249	0.5812	0.5339
LINE	0.4951	0.4472	0.5317	0.4778	0.5395	0.4942
GraRep	0.4908	0.4355	0.5326	0.4622	0.5335	0.4662
GraphGAN	0.4260	0.3837	0.5347	0.4888	0.5570	0.5146
TADW	0.6451	0.5990	0.7055	0.6487	0.7174	0.6639
GAE	0.6273	0.5806	0.6727	0.6055	0.6868	0.6059
SAGE	0.5039	0.4707	0.5692	0.5305	0.5999	0.5563
DANE	0.6585	0.6121	0.7085	0.6528	0.7115	0.6553
ProGAN	**0.7186**	**0.6488**	**0.7417**	**0.6748**	**0.7440**	**0.6931**

[92], GAE [49], SAGE [35], and DANE [25]. For DeepWalk and Node2Vec, when conducting random walk, we set the number of walks to 10 and the walk length to 80. When constructing the context from node sequences, the window size is set to 10. For LINE, we utilize both the first-order and second-order proximity. In addition, the number of negative samples is 5. For GraRep, the transition step length is set to 5. Throughout our experiments, the dimension of node representations is set to 100.

Regarding our proposed method, when constructing the real triplet, we need to sample negative neighbors. In our experiment, following [83], the sampling probability is $P_v = d_v^{3/4}$ where d_v denotes the degree of node v. When sampling positive neighbors, we randomly select them from the connected nodes of a reference node. In addition, all three components of our proposed ProGAN employ the multi-layer perceptron (MLP) model. Specifically, the architecture of the generator is $128 \rightarrow 512 \rightarrow d$ where 128 is the dimension of input noise while d is the dimension of the node attribute. Furthermore, we enforce z_1 and z_2

Table 3: Node classification result of Cora dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.7424	0.7368	0.7975	0.7866	0.8148	0.8032
Node2Vec	0.7777	0.7649	0.8107	0.8001	0.8118	0.8007
LINE	0.7473	0.7399	0.7943	0.7883	0.8081	0.8011
GraRep	0.7609	0.7510	0.7700	0.7558	0.7764	0.7617
GraphGAN	0.6957	0.6804	0.7405	0.7241	0.7668	0.7557
TADW	0.7683	0.7462	0.8201	0.7989	0.8435	0.8293
GAE	0.7662	0.7587	0.7980	0.7852	0.8015	0.7896
SAGE	0.6608	0.6403	0.7664	0.7520	0.8044	0.7921
DANE	0.7769	0.7558	0.8212	0.8062	0.8258	0.8094
ProGAN	**0.8080**	**0.7866**	**0.8365**	**0.8172**	**0.8486**	**0.8357**

have the same dimension. Thus, their dimension is 64. The architecture of the encoder is $d \to 512 \to 100$ where 100 is the dimension of the node representation. The discriminator has an almost same architecture as the encoder. The only difference is that the discriminator has one more branch to distinguish real nodes and generated nodes. Moreover, other than the last layer, the generator uses ReLU as the activation function. Sigmoid function is used in the last layer. For the discriminator and the encoder, LeakyReLU is used as the activation function. In our experiments, to evaluate the learned node representation, we conduct node classification, node clustering, and node visualization on the learned low-dimensional representation. To measure the performance of these tasks, we employ Micro-F1 and Macro-F1 to measure the classification performance while utilizing clustering accuracy (ACC) and normalized mutual information (NMI) to measure the clustering performance. The larger these metrics are, the better the performance is.

Table 4: Node classification result of Flickr dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.4421	0.4370	0.5022	0.4960	0.5187	0.5108
Node2Vec	0.4682	0.4623	0.5069	0.5029	0.5261	0.5204
LINE	0.5122	0.5013	0.5265	0.5185	0.5399	0.5320
GraRep	0.4759	0.4683	0.5182	0.5106	0.5132	0.5030
GraphGAN	0.4503	0.4466	0.4982	0.4928	0.5206	0.5133
TADW	0.7524	0.7471	0.7707	0.7662	0.7777	0.7738
GAE	0.6273	0.5806	0.6727	0.6055	0.6868	0.6059
SAGE	0.2796	0.2741	0.3194	0.3132	0.3303	0.3253
DANE	0.6078	0.6085	0.6753	0.6747	0.7030	0.7020
ProGAN	**0.7958**	**0.7899**	**0.8192**	**0.8143**	**0.8271**	**0.8239**

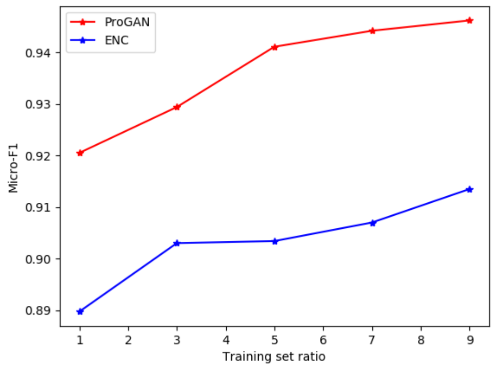
3.4.3 Results and Analysis

3.4.3.1 Node Classification To verify whether a network embedding method can discover and preserve the proximity, we conduct node classification on the learned node representation. In this experiment, the classifier is the ℓ_2-norm regularized logistic regression. In detail, all nodes are used to train network embedding methods. Then, we can get the low-dimensional representation for each node. After that, to conduct node classification, we randomly select $10\%, 30\%, 50\%$ nodes as the training set and the remained nodes as the testing set. For each case of the training set, we employ five-fold cross-validation to select the best classifier parameters. Then, we report the classification accuracy of the testing set in Table 2, 3, 4, 5. From these tables, we can find that our proposed ProGAN has a better performance than all the other state-of-the-art methods. In particular, we have several observations as follows.

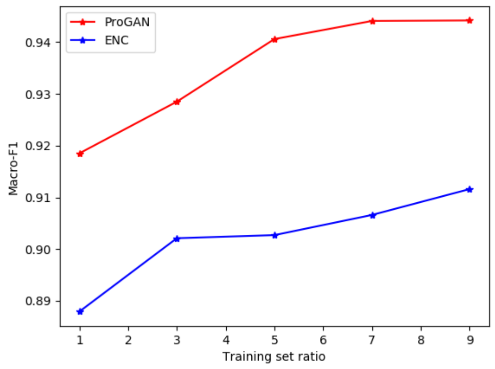
First, comparing with GraphGAN which learns to choose negative neighbors rather than discover the proximity, our method has significant improvement. Especially, ProGAN only

Table 5: Node classification result of Blogcatalog dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.6241	0.6156	0.6778	0.6730	0.6999	0.6954
Node2Vec	0.6104	0.6040	0.6501	0.6431	0.6506	0.6455
LINE	0.6630	0.6539	0.6844	0.6737	0.6933	0.6863
GraRep	0.7000	0.6956	0.7163	0.7118	0.7426	0.7398
GraphGAN	0.5196	0.5158	0.5877	0.5811	0.6056	0.6000
TADW	0.9128	0.9115	0.9241	0.9233	0.9296	0.9290
GAE	0.3917	0.3318	0.4219	0.3717	0.4179	0.3681
SAGE	0.5390	0.5321	0.5547	0.5471	0.5387	0.5340
DANE	0.8642	0.8619	0.9065	0.9054	0.9104	0.9091
ProGAN	**0.9294**	**0.9280**	**0.9426**	**0.9418**	**0.9450**	**0.9443**



(a) Blogcatalog: Micro-F1



(b) Blogcatalog: Macro-F1

Figure 4: The node classification result of ProGAN and ENC. ENC denotes the model without generator and discriminator.

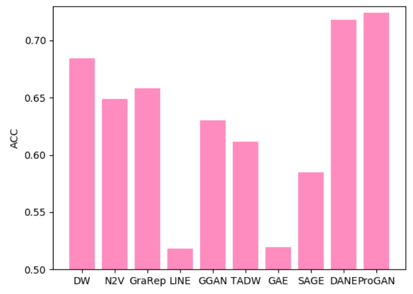
uses the popularity-based negative sampling method, but it can improve the embedding result significantly. Thus, we can conclude that the learned proximity is helpful to learn node representations.

Second, comparing with attributed network embedding methods which use the high-order proximity, such as GAE and SAGE, our proposed ProGAN shows a better performance. Specifically, when constructing the real triplet, our method only uses the first-order proximity. However, ProGAN can beat those methods which use the high-order proximity. Hence, we can conclude that the generated proximity can serve as the high-order proximity to learn a good node representation.

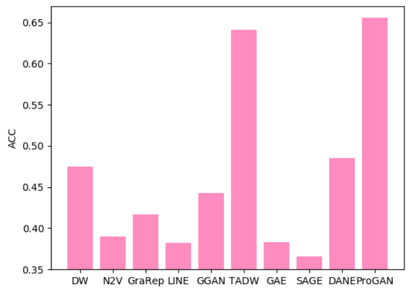
To further verify this point, we conduct another experiment. In detail, we remove the discriminator and generator, only training the encoder with real triplets. Then, we conduct node classification on the learned node representation. The result is shown in Figure 4. Here, this baseline method is denoted by ENC. Due to the space limitation, we only report the results of Blogcatalog. From Figure 4, we can find the improvement of our proposed ProGAN is significant, which verifies that the generated proximity by ProGAN acts as the high-order proximity to benefit node representation learning.

3.4.3.2 Node Clustering To further verify the performance of our proposed ProGAN, we conduct node clustering on the learned node representation. Similar with node classification, all nodes are used to train network embedding models. After that, we run K-means on the learned low-dimensional representations. To measure the clustering performance, clustering accuracy (ACC) and normalized mutual information (NMI) are used. The result is shown in Figure 5. From it, we can find that our proposed ProGAN can outperform the other baseline methods significantly in terms of both ACC and NMI, which further verifies the effectiveness of our proposed method.

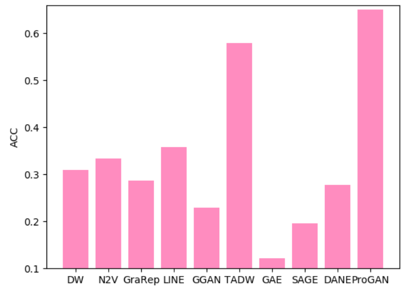
3.4.3.3 Network Visualization Network visualization is another widely used task to verify the embedding result. Hence, we visualize the learned node representations of these methods by using T-SNE [62]. Specifically, it projects the learned low-dimensional nodes into the two-dimensional space and then visualizes them. Due to the space limitation, we only



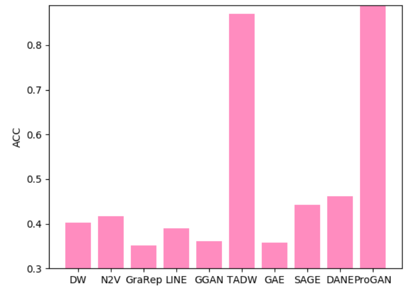
(a) Cora: ACC



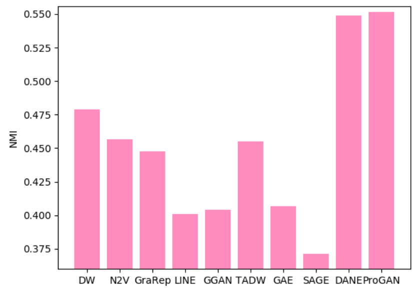
(b) Citeseer: ACC



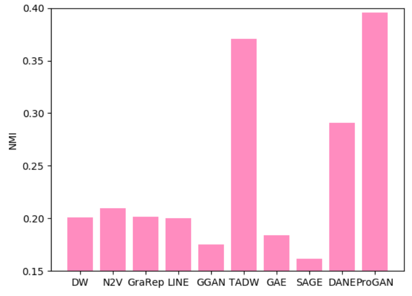
(c) Flickr: ACC



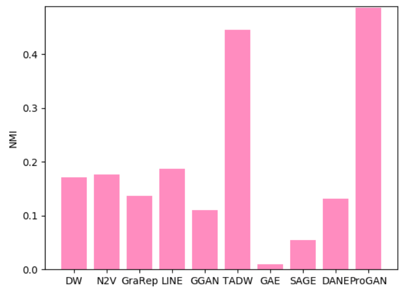
(d) Blogcatalog: ACC



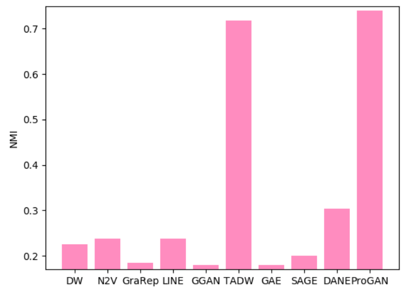
(e) Cora: NMI



(f) Citeseer: NMI



(g) Flickr: NMI



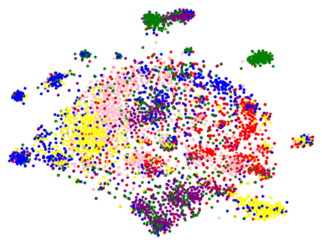
(h) Blogcatalog: NMI

Figure 5: Node clustering results.

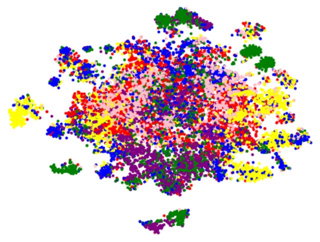
report the result of Blogcatalog dataset. The result is shown in Figure 6. Here, we only report the result of the best 7 baseline methods. Apparently, comparing with the other state-of-the-art methods, the learned node representation of ProGAN has a more compact structure within a group and a larger margin between different groups. In particular, although TADW and DANE have good node classification result, yet its node representations do not have a compact group structure and large margin between groups as our method, which is also verified by the clustering result. All in all, ProGAN can discover and preserve the proximity better than the baseline methods.

3.5 Conclusion

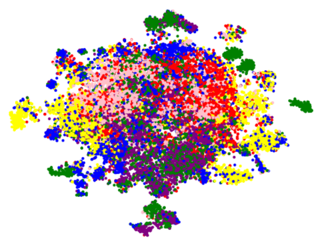
In this chapter, we propose a novel proximity generative adversarial network for network embedding. Specifically, the proposed ProGAN can generate proximities, with which we can discover the underlying complicated relationship between different nodes. Then, by preserving these underlying proximities in the low-dimensional space, the learned node representation has shown better performance on several tasks, which verifies the effectiveness of our proposed method. For the future work, we will focus on generating the structural proximities rather than the ordinary proximity to further improve network embedding.



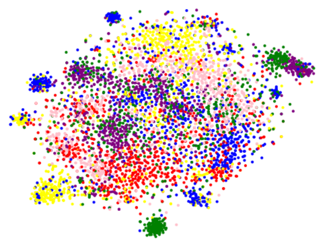
(a) DeepWalk



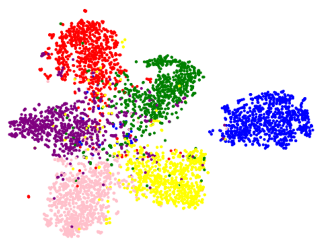
(b) GraRep



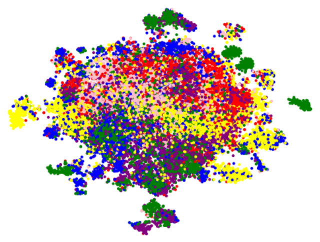
(c) LINE



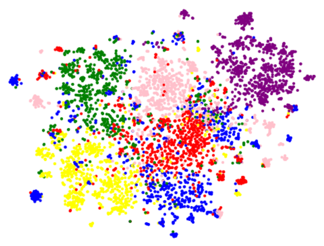
(d) GraphGAN



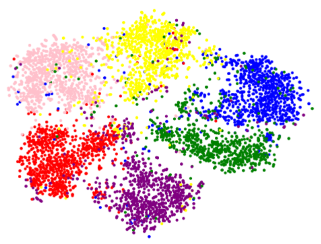
(e) TADW



(f) SAGE



(g) DANE



(h) ProGAN

Figure 6: Network visualization of Blogcatalog dataset.

4.0 Self-Paced Network Embedding

4.1 Introduction

The essential idea to learn the low-dimensional representations is to preserve the proximity in the topological structure of a network. By preserving such proximities, nodes with large proximity will have similar low-dimensional representations, benefiting downstream tasks. To obtain the effective low-dimensional representation, a wide variety of approaches have been proposed. They aim at capturing various proximities in a network. For example, the seminal DeepWalk model [72] employs random walk to obtain the node sequences and then reformulates network embedding as word embedding by regarding node sequences as word sequences. In this way, the proximity between nodes can be captured by Skip-gram [67] model. Afterwards, Node2Vec [33] was proposed based on the similar idea but with a novel random walk algorithm. Additionally, LINE [83] was introduced to preserve the first and second order proximity when learning the low-dimensional representation. GraRep [12] aims at preserving high order proximity during representation learning. Struc2Vec [74] targets to preserve to structural proximity by identifying structure identities.

Among these existing methods, the methods based on Skip-gram [67] model, such as DeepWalk [72], Node2Vec [33], and LINE [83] have attracted much attention. Their basic idea is to push together similar nodes while pushing away dissimilar nodes. Mathematically, it is formulated as a binary classification problem which classifies two similar nodes as a positive pair while classifying two dissimilar nodes as a negative pair. Although these approaches have shown good performance in many tasks, yet it has a drawback. Specifically, to train this binary classification problem, we should feed both positive and negative pairs to the model. For each node, its paired positive node can be easily obtained according to edges of the network [83] or the result of random walk [72, 33]. But how to obtain the negative nodes is not obvious. Some negative nodes are more informative while some are not. If employing random sampling method, we cannot discriminate the informativeness of different nodes. Therefore, existing methods, such as DeepWalk [72], Node2Vec [33], and

LINE [83], propose to sample negative nodes based on a pre-designed distribution. This distribution depends on the degree of nodes. The intuition behind this sampling method is that oversampling more connected nodes will lead to better performance because these nodes carry more information than less connected ones. Although this strategy has correct intuition, the idea is incomplete because this method ignores that the less connected nodes may also be informative in practice. Thus, we cannot just use the connection information to determine the informativeness of a node. How to discover the really informative nodes for training models is important and also challenging.

Moreover, the sampling distribution employed by the existing network embedding methods [72, 33, 83] is static, which means that the sampling distribution does not change during training. As a result, the informativeness of a node is constant when training models. However, with the training process going on, some negative nodes are well pushed away from the anchor node while some are not. In such a case, we should put more focus on these difficult nodes. Therefore, it is better to sample nodes with different probabilities at different training phases. On the other hand, we cannot always focus on the difficult nodes, ignoring the easy ones. According to the learning process of human beings, we should start with easy samples when learning a new model and then learn difficult samples gradually [7, 52]. The reason is that we can learn a draft model by using easy samples and then refine it by taking difficult samples. In our case, if we always select difficult negative nodes to train our model, the easy ones cannot be fully utilized, and it is easy to disturb the low-dimensional representations by too much focus on the difficult nodes.

To address the above challenging problems, in this chapter, we propose a novel self-paced network embedding method. With this method, we can dynamically sample the informative negative nodes for training models. Specifically, we propose a self-paced node sampling strategy. This strategy can discover the informativeness of each node based on current model parameters and then sample negative nodes according to their informativeness. In addition, this self-paced strategy can sample difficult negative nodes gradually with the training process going on. Moreover, we extend this self-paced sampling strategy to the generative adversarial network framework. The extensive experiments are conducted on seven benchmark network datasets to validate the effectiveness of our proposed methods.

4.2 Related Work

Network Embedding Network embedding has become a popular data mining research topic in recent years due to its important role in network data analysis. It is to learn a low-dimensional representation for each node in a network while preserving proximity. Recently, there has been much progress towards effective network embedding algorithms. For example, DeepWalk [72] was proposed to utilize random walk to get node sequences as word sentences, then employ Skip-gram [67] model to learn node embedding, which is essential to predict the context of each anchor node by maximizing the likelihood function as follows:

$$\max \prod_{i=1}^{n} \prod_{j \in c_i} p(v_j | v_i) \,, \tag{4.1}$$

where v_i is the anchor node, v_j denotes its context node, and c_i denotes the context set of node v_i. $p(v_j|v_i)$ is the conditional probability of node v_j given node v_i. Based on this schema, LINE [83] and Node2Vec [33] were proposed respectively with different context constructing approaches. In addition, some other kinds of methods also make much progress. For example, GraRep [12] aims at preserving high order proximity during representation learning. Struc2Vec [74] aims to preserve structural proximity by identifying structure identities. M-NMF [90] captures community structure when learning the low-dimensional representation. TADW [92] and ANE [42] target to integrate the topological structure and node attributes to learn node representations. A dynamic network embedding method [55] was proposed to handle the dynamic networks. SDNE [87] employs deep autoencoder to capture the high non-linearity in the network. In [48], a semi-supervised network embedding method was proposed based on the graph convolutional network. Although these methods are promising in many tasks, in this chapter, we will limit our interest to solve the model related to Eq. (4.1).

Negative Sampling Negative sampling [67, 34] is an efficient approach for solving multi-class classification problems. In detail, the multi-class classification problem, such as problem (4.1), is considerably inefficient when the number of classes is very large. Negative sampling (or termed as contrastive learning) reformulates it as a binary classification problem by constructing positive pairs and negative pairs. The positive pair is usually easy to construct based on some prior knowledge. For example, positive pairs in a network can be easily constructed based on the connection between nodes [83]. The challenge is how to construct negative pairs. In [67], the authors propose to sample negative pairs based on the word popularity, while [83] considers constructing the sampling distribution in terms of the degree of nodes. Actually, both of them share a common intuition that more popular or connected samples should be selected more frequently since they are more informative. Under the context of recommender systems, [98] proposes a dynamic negative sampling algorithm based on the temporary recommend result. Recently, IRGAN [89] was proposed to have the same spirit as negative sampling for information retrieval. Unlike conventional negative sampling, it employs a generator to generate the negative samples dynamically.

Self-Paced Learning In machine learning community, how to select training samples to learn a good model is an important topic. As we know, human beings usually learn from easy concepts to complex ones. To mimic this cognitive activity of humans, curriculum learning [7] and self-paced learning [52] have attracted much attention. Particularly, based on some fixed prior knowledge, curriculum learning constructs a ranking function to assign different learning priorities to different samples. Self-paced learning selects training samples voluntarily by measuring the performance of samples dynamically as follows:

$$\min_{v_i,w} \sum_{i=1}^{n} v_i l(y_i, f(x_i; w)) - \lambda \sum_{i=1}^{n} v_i \ , \tag{4.2}$$

where w is the model parameter, $v_i \in \{0, 1\}$ indicates whether the training sample (x_i, y_i) is selected, and $l(\cdot, \cdot)$ denotes the loss function.

Along this line, many works have been proposed for different situations. For example, [44] proposes a self-paced learning algorithm to select easy and diverse samples. [59] proposes the self-paced co-training under the semi-supervised settings. [82] is proposed for object tracking. [45] is proposed for combining curriculum learning and self-paced learning. However, as far as we know, there are no existing works for our settings. Thus, inspired by self-paced learning, we will focus on how to perform negative sampling from easy samples to difficult ones dynamically to solve Eq. (4.1).

4.3 New Self-Paced Network Embedding

4.3.1 Problem Definition

Given a network $\mathcal{G} = \{\mathcal{V}, E\}$ where $\mathcal{V} = \{v_i\}_{i=1}^n$ denotes a set of n nodes and $E = [e_{ij}] \in \mathbb{R}^{n\times n}$ denotes the adjacency matrix, if there exists an edge between node v_i and v_j, $e_{ij} = 1$. Otherwise, $e_{ij} = 0$. The embedding of each node v_i is a low-dimension vector $u_i \in \mathbb{R}^d$.

Network embedding is to learn low-dimensional representations $\{u_i\}_{i=1}^n$ of nodes. The essential idea is to push similar nodes together while pushing dissimilar nodes away to each other in the low-dimensional vector space. Thus, DeepWalk [72], LINE [83], and Node2Vec [33] propose to optimize Eq. (4.1). By optimizing this model, the context node v_j of the anchor node v_i will have a high probability $p(v_j|v_i)$, while other nodes have a small $p(v_j|v_i)$. Thus, Eq. (4.1) can push positive context nodes v_j to the anchor node v_i while pushing negative context nodes away from the anchor node.

However, Eq. (4.1) actually is a multi-class classification problem in which the number of classes equals the number of nodes. When n is very large, it is inefficient. Therefore, DeepWalk [72], LINE [83], and Node2Vec [33] reformulate Eq. (4.1) as an equivalent problem and employ negative sampling method to accelerate it as follows:

$$\max \log p(v_p|v_i) + \sum_{j\in\mathcal{N}_{v_i}} \log(1 - p(v_j|v_i)) \,, \tag{4.3}$$

where v_p denotes the positive context of node v_i while v_j denotes the negative context, $\mathcal{N}_{v_i}$ denotes the sampled negative context node set and $|\mathcal{N}_{v_i}| = k$. By using this effective negative sampling method, the complexity is reduced to $O(km)$ while the original complexity is $O(nm)$ where $O(m)$ is the complexity of computing the loss of each sample and $k \ll n$. Now, a natural question is how to select the negative context samples $\mathcal{N}_{v_i}$ effectively? In existing works [72, 83, 33], they use a predefined distribution, which reflects the popularity of nodes, to sample them. Specifically, LINE [83] constructs the sampling distribution based on the number of degrees of nodes such that a more connected node will be selected in a large probability. However, it has some drawbacks in practice as follows:

- The popularity-based sampling method cannot really reflect the informativeness of a node. Some less connected nodes can also be much informative in practice.
- The informativeness of a node is usually changing with the training process going on. But the predefined sampling method fails to reflect this change.

Therefore, in this chapter, we will focus on developing effective negative sampling algorithms to address these problems.

4.3.2 Self-Paced Network Embedding

In this section, we will introduce a novel dynamic sampling method for network embedding: self-paced network embedding. At first, to fully capture the informativeness of nodes and reflect the training state, we propose a dynamic negative sampling method. Based on this method, we further propose a self-paced sampling method to feed difficult samples gradually with the training process going on to learn a better embedding result. At last, we extend the proposed self-paced sampling method to the generative adversarial network framework.

4.3.2.1 Informativeness of Nodes Formally, the conditional probability in Eq. (4.1) and Eq. (4.3) is defined as follows:

$$p(v_j|v_i) = \sigma(u_j^T u_i) = \frac{1}{1+\exp(-u_j^T u_i)} \, , \tag{4.4}$$

where u_i and u_j are the low-dimensional representation of node v_i and v_j respectively. Here, we call v_i *anchor node*. As we can see from this formulation, a large inner product between u_i and u_j denotes a high probability of v_j given v_i. For all the negative context nodes $\mathcal{N}_{v_i}$ of the anchor node v_i, they will have different conditional probabilities based on the currently learned low-dimensional representation. Moreover, as for a negative context node, if its conditional probability is high, it should be a *difficult negative context node* since it is close to the anchor point based on the inner product but we should push it away from the anchor node. Otherwise, it is an *easy negative context node* because it is far away from the anchor point. Thus, it is natural to use Eq. (4.4) to denote the *informativeness* of a negative context node given the anchor node. If $p(v_j|v_i)$ is large, v_j is more informative to the anchor node v_i. Otherwise, it is less informative since it has already been far away from the anchor point.

On the other hand, the essentiality of network embedding is to push away negative context nodes from the anchor point. Thus, to learn a good low-dimensional representation such that similar nodes are clustered while dissimilar nodes are separated, it is better to put more focus on difficult negative context nodes to push them away from the anchor node. Based on this intuition, it is natural to construct the negative sampling distribution based on the informativeness of nodes between the currently learned low-dimensional representation at each iteration. Mathematically, we define the informativeness-aware negative sampling distribution as follows:

$$p_{ij} = \frac{\exp{(u_j^T u_i)}}{\sum_{j \in \mathcal{N}_{v_i}} \exp{(u_j^T u_i)}} \ , \tag{4.5}$$

where $\mathcal{N}v_i$ is the negative context node set of node v_i. As shown in this formulation, if a negative context node is close to the anchor node, the inner product between them will be large so that the probability p_{ij} is large. As a result, the difficult negative context node has a large chance to be selected. Then, it will be updated more frequently than the easy ones so that it is pushed away from the anchor point. Thus, our method can always select the more informative nodes to update model parameters. On the contrary, the conventional sampling distribution used in [72, 83, 33] only focuses on the more connected nodes. With such a connection-based sampling method, even when a more connected node has already been well separated from the anchor node, it still has a large chance to be updated. While a less connected node which is close to the anchor node has small chance to be pushed away from the anchor node. Thus, their result is not satisfactory. As for our method, we put more focus on the difficult negative context node. As a result, even though a negative context node is less connected, it still has a large chance to be selected if it has a large inner product with the anchor node. All in all, our proposed method can effectively select the more informative negative context node.

Furthermore, unlike the predefined sampling distribution employed in [72, 83, 33], our proposed sampling distribution can reflect the training state since the low-dimensional representation is updated at each iteration. In particular, the sampling distribution changes dynamically according to the current state. An easy negative context node may become a difficult one after updating model parameters, then it will have a large chance to be selected. As a conclusion, our proposed method can automatically and dynamically select the informative negative context node based on the training state.

4.3.2.2 Self-Paced Negative Sampling Although the proposed sampling distribution in Eq. (4.5) can always select the more informative nodes at each iteration, yet it fails to utilize all nodes. Specifically, at each iteration, it always selects the difficult negative context node, the easy one almost has no chance to be pushed away from the anchor node. Although the easy negative context node may already be separated with the anchor node, yet it still needs to refine so that we can get a better result.

On the other hand, according to self-paced learning [7, 52], like human beings learning from easy samples to complex ones, we should feed easy samples at early training stage and gradually provide difficult samples with the training process going on. However, the sampling distribution in Eq. (4.5) always feed difficult samples at any training stage. Therefore, to fully utilize all nodes and gradually select difficult nodes, we propose the self-paced network embedding (SeedNE) model as follows:

$$\begin{aligned} &\max \log p(v_p|v_i) + \sum_{j \overset{p'_{ij}}{\sim} \mathcal{N}_{v_i}} \log(1 - p(v_j|v_i)) + l(\mu) \\ &s.t. \quad p'_{ij} = \begin{cases} p_{ij}, & p_{ij} < l(\mu) \\ 0, & otherwise \end{cases}, \end{aligned} \tag{4.6}$$

where $\mathcal{N}v_i$ is the negative context node set of node v_i, $l(\mu)$ is a threshold function parameterized by μ, which acts as a threshold to the sampling probability. In detail, for the probability p_{ij} obtained from Eq. (4.5), if it is larger than the threshold $l(\mu)$, we set it as zero as shown in Eq. (4.6). Actually, it is to set the probability of difficult negative context nodes to zero such that the easy ones have larger chance to be selected at the early training stage. With the training process going on, $l(\mu)$ is increasing so that difficult negative context nodes will be included gradually. This is consistent with our motivation. The concrete threshold function $l(\mu)$ is deferred to Section 4.4.

Algorithm 2 Self-Paced Network Embedding (SeedNE)

```
1: for each node $v_i$ do
2:     Sample positive context nodes $v_p$.
3:     Sample negative context nodes $v_j$ according to $p'_{ij}$.
4:     Update node representations $u_i$, $u_p$, $u_j$ and the parameter $\mu$ by SGD.
5:     Update the sampling probability $p'_{ij}$ as Eq. (4.5) and Eq. (4.6).
6: end for
```

By optimizing problem (4.6) for each anchor node v_i, we can gradually sample difficult negative context nodes based on the training performance to learn low-dimensional representations of nodes. At last, we summarize the self-paced network embedding (SeedNE) method in Algorithm 2.

4.3.2.3 Extension: Adversarial Self-Paced Network Embedding In recent years, Generative Adversarial Network (GAN) [32] has attracted a surge of attention due to its flexibility to simulate distributions. The GAN framework includes a generator and a discriminator. The generator is to approximate the data distribution while the discriminator is to discriminate the approximated and true data distribution. Mathematically, the GAN framework is to optimize the following problem:

$$\min_{\theta} \max_{\phi} E_{x\sim p(x)} \log[D(x)] + E_{x\sim G(z),z\sim p(z)}[\log(1 - D(x))] \ , \tag{4.7}$$

where $D(\cdot)$ denotes the discriminator parameterized by ϕ, $G(\cdot)$ denotes the generator parameterized by θ, $p(x)$ denotes the data distribution, and $p(z)$ denotes a prior distribution. Here, the generator $G(\cdot)$ tries to use a deep neural network to map a simple prior distribution $p(z)$ to the complicated data distribution. By optimizing this objective function, the generator is expected to generate similar samples with true ones as well as possible.

Inspired by the GAN framework, we can use a generator to generate negative samples for Eq. (4.3). Specifically, for each anchor node v_i, it is defined as follows:

$$\min_{\theta} \max_{\phi} E_{v_j\sim p_d(v_j|v_i)} \log p_{\phi}(v_j|v_i) + E_{v_j\sim G_{\theta}(v_j|v_i)}[\log(1 - p_{\phi}(v_j|v_i))] \ , \tag{4.8}$$

where $p_\phi(v_j|v_i)$ is identical with Eq. (4.4) in which $\phi = \{u_i, u_j\}$, $p_d(v_j|v_i)$ denotes the sampling distribution for the positive context node of node v_i, $G_\theta(v_j|v_i)$ denotes the sampling distribution for the negative context node, which is constructed from the generator.

This model includes two components: discriminator and generator. The discriminator has the same functionality as Eq. (4.3), which pushes similar nodes together while pushing away different nodes in the low-dimensional vector space. The generator is to generate negative context nodes by constructing the sampling distribution $G_\theta(v_j|v_i)$. More details are explained in the following.

Discriminator Specifically, the discriminator is to solve the following problem:

$$\max_\phi E_{v_j \sim p_d(v_j|v_i)} \log p_\phi(v_j|v_i) + E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \ , \tag{4.9}$$

which is actually identical with Eq. (4.3). The only difference is how to obtain negative context nodes. The negative context node of this model is generated from the generator $G_\theta(v_j|v_i)$ while Eq. (4.3) uses a predefined distribution for sampling them. Similarly, the difference between this discriminator and Eq. (4.6) is also the sampling method. Eq. (4.6) constructs the basic sampling distribution according to the current training state as shown in Eq. (4.5). This discriminator also sample negative nodes based on the current training state, but it uses a generator to construct such a distribution in a more flexible way, which will be shown in the following. By optimizing this model, we can obtain the embedding result $\phi = \{u_i\}_{i=1}^n$.

Generator On the other hand, the generator is to solve the following problem:

$$\min_\theta E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \ , \tag{4.10}$$

where the generator model $G_\theta(v_j|v_i)$ constructs a sampling distribution as follows:

$$G_\theta(v_j|v_i) = \frac{\exp(u_j'^T u_i')}{\sum_j \exp(u_j'^T u_i')} \ , \tag{4.11}$$

where $\theta = \{u_i'\}_{i=1}^n$ is the model parameter of the generator. Specifically, $u_i' \in \mathbb{R}^d$ denotes the low-dimensional representation of node v_i in the generator. At first glance, Eq. (4.11) has no difference with Eq. (4.5). However, Eq. (4.5) uses the current embedding result $\{u_i\}_{i=1}^n$ to construct the sampling distribution while Eq. (4.11) employs a more flexible way to construct the sampling distribution to find the underlying informative negative context nodes. In particular, it constructs the sampling distribution by training another model parameterized by new parameters $\theta = \{u'_i\}_{i=1}^n$. With these new parameters, the sampling distribution will have more capacity to discover the informative nodes. Furthermore, similar with Eq. (4.5), these new parameters can also reflect the current embedding result. Specifically, the loss function in Eq. (4.10) is computed based on the current embedding result. Thus, when updating new model parameters $\theta = \{u'_i\}_{i=1}^n$ by gradient descent method, the current embedding result can directly affect θ through the gradient information.

With this flexible distribution, we can sample v_j as the negative context node of node v_i to train this model. However, there is a challenge to optimize Eq. (4.10). In particular, to use gradient descent method to update model parameter θ, the model should be continuous. However, the sample v_j that the loss function depends on is not the direct output of the generator $G_\theta(v_j|v_i)$. Instead, v_j is sampled from a node set according to $G_\theta(v_j|v_i)$, which is discrete. Thus, we cannot use gradient descent method directly to update model parameter θ. A practical way is to employ the policy gradient [78] to update model parameter θ, which is defined as follows:

$$\begin{aligned}
&\nabla_\theta E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \\
&= \sum_{j=1}^{K} \nabla_\theta G_\theta(v_j|v_i) \log(1 - p_\phi(v_j|v_i)) \\
&= \sum_{j=1}^{K} G_\theta(v_j|v_i) \nabla_\theta \log[G_\theta(v_j|v_i)] \log(1 - p_\phi(v_j|v_i)) \\
&= E_{v_j \sim G_\theta(v_j|v_i)} \nabla_\theta \log[G_\theta(v_j|v_i)] \log(1 - p_\phi(v_j|v_i)) \ .
\end{aligned} \tag{4.12}$$

However, taking a look at the sampling distribution Eq. (4.11) defined in the generator, we can find that this distribution has the same spirit as Eq. (4.5), which always selects the difficult negative context node. Thus, it potentially ignores the easy negative context node and cannot gradually select difficult ones with the training process going on. To alleviate these problems, we further propose the adversarial self-paced network embedding (ASeedNE) method as follows:

$$\min_{\theta}\max_{\phi} E_{v_j\sim p_d(v_j|v_i)}\log p_\phi(v_j|v_i) + E_{v_j\sim G'_\theta(v_j|v_i)}[\log(1-p_\phi(v_j|v_i))] + l(\mu)$$
$$s.t.\quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise\ . \end{cases} \tag{4.13}$$

There are still two components in this novel model. Specifically, the discriminator is to optimize the following problem:

$$\max_{\phi} E_{v_j\sim p_d(v_j|v_i)}\log p_\phi(v_j|v_i) + E_{v_j\sim G'_\theta(v_j|v_i)}[\log(1-p_\phi(v_j|v_i))] + l(\mu)$$
$$s.t.\quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise\ . \end{cases} \tag{4.14}$$

The difference between this discriminator and that in Eq. (4.9) is that here we employ a self-paced sampling strategy such that the difficult negative context node is gradually utilized to train this model. While the model in Eq. (4.9) always choose the most difficult negative context nodes to train modal parameters.

Additionally, the generator is to optimize the following problem:

$$\min_{\theta} E_{v_j\sim G'_\theta(v_j|v_i)}[\log(1-p_\phi(v_j|v_i))]$$
$$s.t.\quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise\ . \end{cases} \tag{4.15}$$

For this generator, after sampling v_j according to the self-pace sampling strategy, we can still use the policy gradient defined in Eq. (4.12) to update model parameters. At last, our adversarial self-paced network embedding method is summarized in Algorithm 3.

Note that although IRGAN [89] also utilizes the GAN framework to mimic the negative sampling procedure, yet our method is different with IRGAN. In particular, IRGAN always chooses the difficult negative samples while our method can gradually select difficult negative context nodes according to the training state. In this way, our method learns node representations from easy negative context nodes to get a draft of the embedding and then refine the embedding result by selecting difficult negative context nodes gradually.

Algorithm 3 Adversarial Self-Paced Network Embedding (ASeedNE)

```
1: repeat
2:     Sampling positive and negative context nodes according to $p_d$ and $G'_\theta$ respectively to
       update the discriminator in Eq. (4.14).
3:     Sampling negative context nodes according to $G'_\theta$ to update the generator in Eq. (4.15).
4:     Update the sampling probability $G'_\theta$ according to Eq. (4.11) and Eq. (4.15).
5: until Converges
```

Table 6: Description of benchmark datasets

Dataset	#Nodes	#Edges	#Labels
Wiki	4,777	184,812	40
PPI	3,890	76,584	50
Cora	2,708	5,278	7
Citeseer	3,312	4,660	6
BlogCatalog	10,312	333,983	39
Facebook	4,039	88,234	-
GR-QC	5,242	14,496	-

4.3.2.4 Summarization Compared with the predefined negative sampling distribution used in [72, 33, 83], our proposed SeedNE can effectively capture the informativeness of each node based on the current embedding result. Based on the informativeness of nodes, SeedNE can gradually select difficult negative context nodes to train the model by employing the self-paced sampling strategy. For the ASeedNE method, it can also discover the informativeness of each node in a more flexible way by training a generator. Similarly, the sampling distribution constructed by the generator can also reflect the current embedding result. All in all, both SeedNE and ASeedNE gradually select difficult negative context nodes to train model parameters.

4.4 Experiments

In this section, we will conduct experiments to show the performance of the proposed methods.

4.4.1 Dataset Descriptions

We use 7 benchmark datasets including social networks, citation networks, biomedical networks, and so on. The details about these datasets are shown as follows.

- Wikipedia (Wiki) [63]: This is a co-occurrence network of words in the Wikipedia database. Here, we employ the preprocessed version of [33]. It contains 4,777 nodes, 184,812 edges. There are 40 classes denoting the Part-of-Speech tag.
- Protein-Protein Interactions (PPI) [9]: This is a biomedical network. We utilize the preprocessed network in [33]. The details about the preprocessing operation can be found in [33]. In particular, there are 3,890 nodes and 76,584 edges in this network. The number of node classes is 50. Different classes denote different biological states.
- Cora [65]: This is an academic citation network, which contains 2,708 machine learning papers. There are 5,429 edges in this network. These papers are from 7 classes, which represent the topic of these papers.
- Citeseer [65]: This is another citation network. It has 3,312 publications from 6 classes. The edge between two publications denotes the citation relationship between them. There are 4,660 edges totally in this network.
- BlogCatalog [84]: This is a social network from the BlogCatalog website. The nodes are bloggers and the edges are friendship relationships among bloggers. The class is inferred from bloggers' interests. Specifically, there are 10,312 nodes, 333,983 edges. The number of classes is 39.
- Facebook [54]: This is a social network where nodes represent users and edges denote the friendship between two users. This dataset is used for the link prediction task
- GR-QC [54]: This is a collaboration network where nodes denote authors and edges represent the collaborative relationship between two authors. This dataset is also used for the link prediction task.

We summarize the statistics of these benchmark datasets in Table 6.

Table 7: Node classification result of Wiki

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.4720	0.0993	0.4850	0.1047	0.4795	0.0946
Node2Vec	0.4753	0.1058	0.4860	0.1154	0.5015	0.1056
GraRep	0.4729	0.1071	0.4929	0.1213	0.4839	0.1000
LINE	0.4830	0.1082	0.4919	0.1014	0.4868	0.1067
SeedNE	**0.5283**	**0.1207**	**0.5429**	**0.1428**	**0.5367**	**0.1355**
ASeedNE	**0.5343**	**0.1222**	**0.5449**	**0.1619**	**0.5513**	**0.1445**

4.4.2 Baseline Methods

To show the performance of our proposed methods, we compare them with four state-of-the-art methods, including DeepWalk, Node2Vec, GraRep, LINE. The details about these methods are described as follows.

- DeepWalk [72]: This method utilizes random walk to get node sequences. By viewing node sequences as word sentences, DeepWalk formulates network embedding as word embedding so that it uses Skip-gram [67] model to learn node representations.

- Node2Vec [33]: Similar with DeepWalk, this method also uses Skip-gram [67] model to learn node representations. But it uses a biased random walk algorithm to get node sequences for constructing node contexts.

- GraRep [12]: This method proposes to preserve high order proximity by constructing k-step probability transition matrix when learning node representations.

- LINE [83]: This method proposes first and second order proximity among nodes. It learns to represent the node by preserving the first and second order proximity.

Table 8: Node classification result of PPI

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.1915	0.1629	0.2156	0.1853	0.2009	0.1659
Node2Vec	0.1858	0.1590	0.2004	0.1663	0.2040	0.1630
GraRep	0.2035	0.1730	0.1999	0.1712	0.2100	0.1654
LINE	0.2092	0.1812	0.2065	0.1768	0.2040	0.1675
SeedNE	**0.2191**	**0.1825**	**0.2268**	**0.1940**	**0.2131**	**0.1835**
ASeedNE	**0.2209**	**0.1854**	**0.2293**	**0.1925**	**0.2268**	**0.2010**

Table 9: Node classification result of Cora

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.8111	0.8023	0.8069	0.8018	0.8044	0.7956
Node2Vec	0.8096	0.8034	0.8007	0.7989	0.8081	0.8080
GraRep	0.7749	0.7577	0.7872	0.7721	0.7897	0.7839
LINE	0.8118	0.8016	0.8069	0.7994	0.7970	0.7842
SeedNE	**0.8155**	**0.8066**	**0.8229**	**0.8200**	**0.8413**	**0.8339**
ASeedNE	**0.8133**	**0.8034**	**0.8167**	**0.8107**	**0.8339**	**0.8234**

4.4.3 Experiment Settings

In our experiments, we set the window size as 10, the walk length as 80, and the number of walks as 10 for DeepWalk. For Node2Vec, it has the same settings as DeepWalk. Additionally, we set the parameter $p = 1$ and $q = 1$ for Node2Vec. For GraRep, the maximum matrix transition step is set as 5. For LINE, we set the number of negative samples as 5. To have a fair comparison, the dimension of node representations is set as 100 for all methods.

For our proposed methods, there is no any preprocessing operation on the network. For each node, we use its connected neighbor nodes as the positive context nodes while the disconnected ones as the negative context nodes. When sampling negative context nodes, we set the number of negative context nodes as 1, that is $\mathcal{N}_{v_i} = 1$. The implementation is based on Tensorflow [1]. Adam [47] algorithm is employed to update model parameters. The batch size is set as 100. Furthermore, the threshold function employed in Eq. (4.6) and Eq. (4.13) is defined as follows:

$$l(u) = \min(au^2 + b, 1) \ , \tag{4.16}$$

where $a > 0$ and $b > 0$ are set as different values for different datasets. Additionally, u is initialized as 1 for all datasets. We can see that this function is always positive and no larger than 1. When it is less than 1, we have its gradient as follows:

$$\nabla l(u) = 2au \ . \tag{4.17}$$

Then, by maximizing Eq. (4.6) or Eq. (4.13), we can update u as follows:

$$u_{t+1} = u_t + 2\eta a u_t \, , \tag{4.18}$$

where $\eta > 0$ is the step size. Because $a > 0$ and u is initialized as 1, then the second term on the right-hand side is positive. As a result, u is increasing such that $l(u)$ is increasing. Then, we can gradually include difficult negative context nodes.

Table 10: Node classification result of Citeseer

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5782	0.5306	**0.6036**	**0.5426**	0.6235	0.5426
Node2Vec	0.5763	0.5094	0.5895	0.5237	0.6205	0.5346
GraRep	0.5522	0.4842	0.5503	0.4800	0.5813	0.4994
LINE	0.5534	0.5026	0.5674	0.5137	0.5904	0.5424
SeedNE	**0.5890**	**0.5419**	**0.5946**	**0.5463**	**0.6416**	**0.5795**
ASeedNE	**0.5914**	**0.5440**	0.5865	0.5354	**0.6295**	**0.5701**

Table 11: Node classification result of Blogcatalog

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.3927	0.2556	0.4047	0.2689	0.4169	0.2836
Node2Vec	0.3965	0.2582	0.4082	0.2698	0.4149	0.2828
GraRep	0.3674	0.2039	0.3830	0.2278	0.3869	0.2322
LINE	0.3518	0.1786	0.3597	0.1845	0.3597	0.1869
SeedNE	**0.4095**	**0.2646**	**0.4216**	**0.2843**	**0.4312**	**0.2884**
ASeedNE	**0.4106**	**0.2680**	**0.4243**	**0.2873**	**0.4326**	**0.3037**

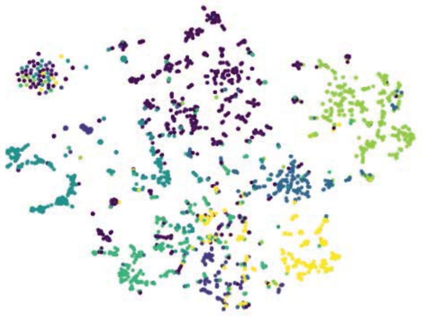
4.4.4 Results and Analysis

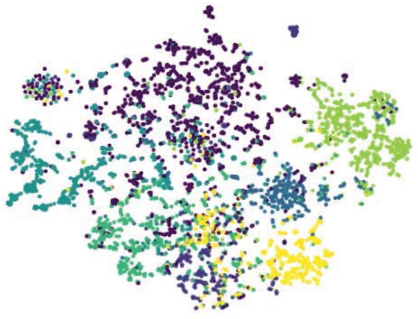
4.4.4.1 Node Classification To evaluate the performance of network embedding, node classification is the most widely used method [72, 33, 12, 83]. In this chapter, we also conduct node classification to show the performance of our proposed methods. The classifier used in this chapter is Logistic Regression. Specifically, we use all nodes to train the network embedding method to get node representations. After that, we conduct node classification on these representations. To evaluate these network embedding methods comprehensively, we randomly select $\{50\%, 70\%, 90\%\}$ nodes to train the classifier respectively. Then, the classifier's performance is evaluated on the rest nodes. To measure the classification performance, we employ Micro-F1 and Macro-F1 as metrics. The larger the two metrics are, the better the classification performance is.

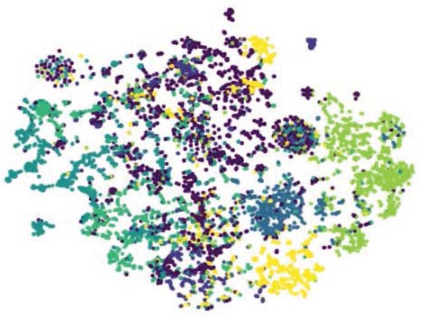
The classification results are shown in Tables 7, 8, 9, 10, 11. In these tables, the best two results in each case are marked in bold. From these tables, we can find that both SeedNE and ASeedNE outperform the other state-of-the-art methods significantly. Specifically,

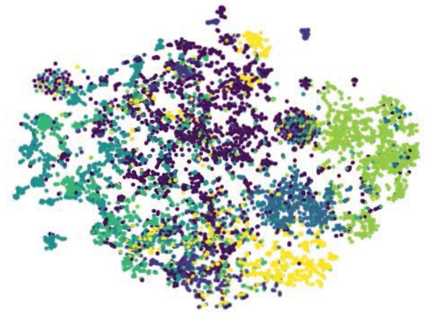
- In Table 7, the best two results are our proposed SeedNE and ASeedNE. Both of them have achieved significant improvement over the other baseline methods. The reason behind this improvement is that our methods can effectively discover informative negative context nodes according to the current embedding result, and then feed difficult negative context nodes gradually to train model parameters such that similar nodes are pushed together while dissimilar nodes are pushed away.
- In Table 7, comparing SeedNE with ASeedNE, we can find that ASeedNE can outperform SeedNE in most cases. The reason is that ASeedNE employs a generator to construct the negative sampling distribution, which has the larger capacity to capture the informativeness of each node than SeedNE. Thus, it can select more helpful negative context nodes to improve model's performance. Similar results can also be found in the other tables, which further verify the effectiveness of our proposed methods.

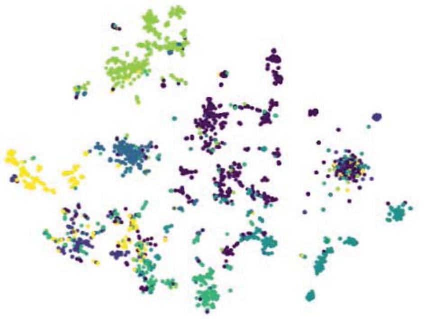
4.4.4.2 Network Visualization Network visualization is another widely used method to evaluate the performance of network embedding algorithms. It is to project the learned node representations into a two-dimensional space such that we can visualize them. In this chapter, we employ the well-developed tool t-SNE [62] to visualize node representations. Here, we only visualize the Cora dataset due to the space limitation. The visualization result is shown in Figure 7. From this figure, we have the following observations:


(a) DeepWalk


(b) Node2Vec


(c) GraRep


(d) LINE


(e) SeedNE

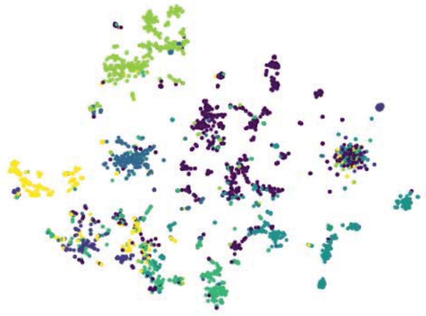

(f) ASeedNE

Figure 7: The visualization of Cora dataset.

Table 12: Link prediction accuracy

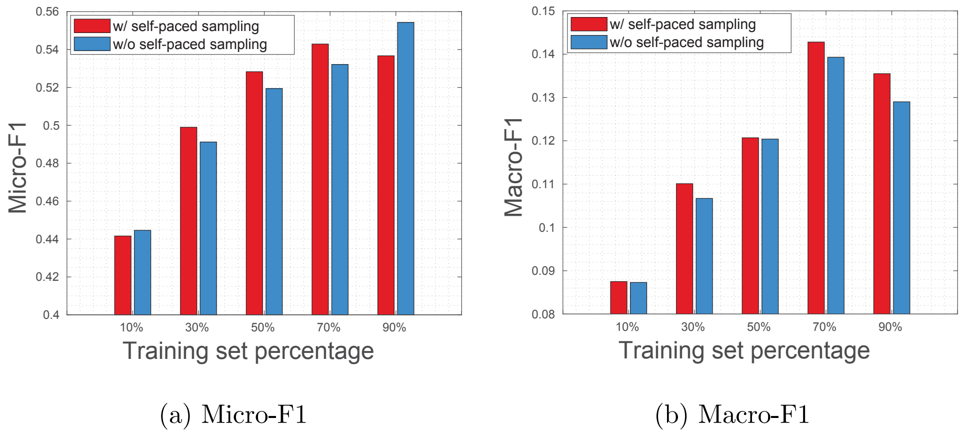
Method	Facebook	GR-QC
DeepWalk	0.9050	0.8354
Node2Vec	0.8900	0.7949
GraRep	0.9445	0.8899
LINE	0.9329	0.8847
SeedNE	**0.9532**	**0.9208**
ASeedNE	**0.9545**	**0.9230**

- In Figure 7(a), nodes scatter over the entire space such that these nodes have no a compact group structure, which means this method fails to push together similar nodes. Similar results can be found in Figure 7(b).
- In Figure 7(c), nodes from different classes are mixed with each other, especially for nodes in the center of this figure. This means that this method fails to push away dissimilar nodes. Figure 7(d) shows the similar result.
- In Figure 7(e), we can see that these nodes have compact group structures and the margin between different groups is large compared with baseline methods, which means our proposed SeedNE can push similar nodes together and push dissimilar nodes away. Thus, our method can achieve a better result on the node classification task. Our proposed ASeedNE has similar results, just as shown in Figure 7(f).

In conclusion, our proposed methods can successfully push similar nodes together and push dissimilar nodes away by sampling informative nodes in each iteration so that the embedding result is better than baseline methods.

4.4.4.3 Link Prediction In this section, we will evaluate the embedding result by using the link prediction task. The goal of this task is to predict the existence of an edge between two nodes. Specifically, Facebook and GR-QC are used in this experiment. We randomly select 10% edges from the original network as positive samples of the testing set. The remained network is used for learning node representation. Additionally, we randomly select the same number of edges from the undiscovered edges as negative samples of the testing set.

The link prediction accuracy of these two datasets is shown in Table 12. From this table, we can observe that our proposed methods outperform all the other baseline methods. Especially, the improvement of the GR-QC dataset is very significant. Specifically, the improvement over the best baseline method is around 4%, which further verifies the effectiveness of our proposed methods.

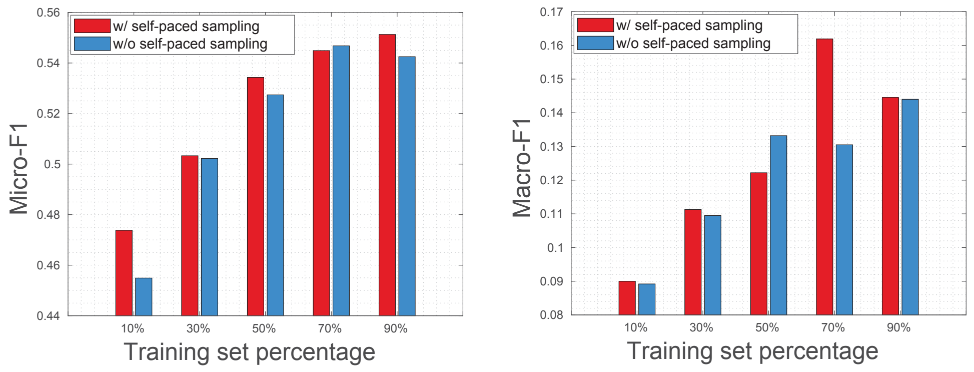


(a) Micro-F1 (b) Macro-F1

Figure 8: Node classification result of Wiki dataset from the SeedNE method.

4.4.4.4 More Results To further show the effect of self-paced sampling strategy, we compare our proposed methods with that without self-paced sampling strategy. In particular, this baseline method employs Eq. (4.5) to sampling negative context nodes at each iteration. Thus, it always feeds difficult negative nodes to train models. Here, we only show the result about the Wiki dataset due to the space limitation. Figure 8 shows node classification results about SeedNE. From this figure, we can find that our method can achieve better classification result for most cases based on both Micro-F1 and Macro-F1. Figure 9 shows node classification results about ASeedNE. Similarly, we can find that our method can beat that without self-paced sampling strategy for most cases, which further verifies the effectiveness of our proposed sampling strategy.

Moreover, in Figure 10, we show the threshold function value during training models. Here, we only show the result of Cora dataset due to the space limitation. Additionally, we only show the first 50 iterations because it is a constant value in the following iterations. Note that we let the parameter a change in the training process. Specifically, a is doubled in every 10 iterations. Intuitively, this operation will accelerate to include difficult negative context nodes. In practice, if a stays as a small value for a long time such that the threshold function value is small, only easy negative context nodes can be included so that the embedding result cannot be further refined. This is consistent with the cognitive activity of human beings. In Figure 10, the threshold function value increases slowly so that only easy negative context nodes are selected to train models at the beginning phase. With the training process going on, the threshold value becomes larger and larger. Therefore, more and more difficult negative context nodes are fed into the model, which is consistent with the intuition of our proposed methods.



(a) Micro-F1 (b) Macro-F1

Figure 9: Node classification result of Wiki dataset from the ASeedNE method.

4.5 Conclusion

In this chapter, we proposed the novel self-paced network embedding methods. Our proposed methods can effectively capture the informativeness of each node. Based on the

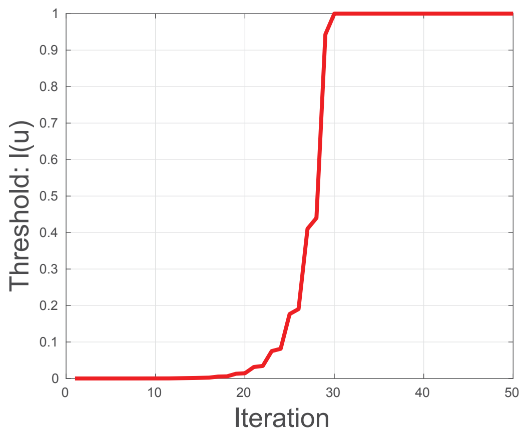


Figure 10: The threshold function value when training Cora.

informativeness of nodes, we introduced a self-paced informativeness-aware sampling strategy. With this sampling strategy, our proposed methods can gradually select informative nodes to train model parameters. Extensive experiments on different data mining tasks have demonstrated the superior performance of our proposed methods.

5.0 Deep Attributed Network Embedding

5.1 Introduction

Attributed networks are ubiquitous in the real world, such as academic citation networks. Unlike the plain network where only the topological structure is available, nodes of attributed networks possess rich attributed information. These informative attributes can benefit network analysis. For example, in an academic citation network, the citation among different articles compose a network where each node is an article and each node has substantial text information about the article's topic. Another example is the social network where users connect with others and post their profiles as the attribute. Furthermore, the social science [64, 66] has shown that attributes of nodes can reflect and affect their community structures [43, 94]. Thus, it is necessary and important to study the attributed network.

Network embedding as a fundamental tool to analyze networks has attracted a surge of attention in data mining and machine learning community recently. It is to learn a low-dimensional representation for each node of a network while preserving the proximity. Then, the downstream tasks, such as node classification, link prediction, and network visualization, can benefit from the learned low-dimensional representation. In recent years, various network embedding methods have been proposed, such as DeepWalk [72], Node2Vec [33], and LINE [83]. However, most of the existing approaches mainly focus on the plain network, ignoring useful attributes of nodes. For example, in the social network such as Facebook or Twitter, each user connects with others, constituting a network. Most of existing methods only focus on the connection when learning node representations. But the attribute of each node can also provide useful information. A good example is the user profile. A young user may have much more similarity with another young guy, rather than an old user. Thus, it is important to incorporate node attributes when learning node representations.

Moreover, the topological structure and attributes of networks are highly non-linear [87]. Thus, it is important to capture the highly non-linear property to discover the underlying pattern. Then, the proximity can be preserved better in the learned node representations.

However, most of existing methods, such as [42, 92], only employ the shallow model, failing to capture the highly non-linear property. Furthermore, how to capture this highly non-linear property is difficult due to the complicated topological structure and attributes. Therefore, it is challenging to capture the highly non-linear property for attributed network embedding.

To address the aforementioned problems, we propose a novel deep attributed network embedding (DANE) approach for attributed networks. In detail, a deep model is proposed to capture the underlying high non-linearity in both topological structure and attributes. Meanwhile, the proposed model can enforce the learned node representations to preserve the first-order and high-order proximity in original networks. Moreover, to learn the consistent and complementary representation from the topological structure and attributes of a network, we propose a novel strategy to combine these two kinds of information. Furthermore, to obtain a robust node representation, an efficient most negative sampling strategy is proposed to make the loss function robust. At last, extensive experiments have been conducted to verify the effectiveness of our proposed approach.

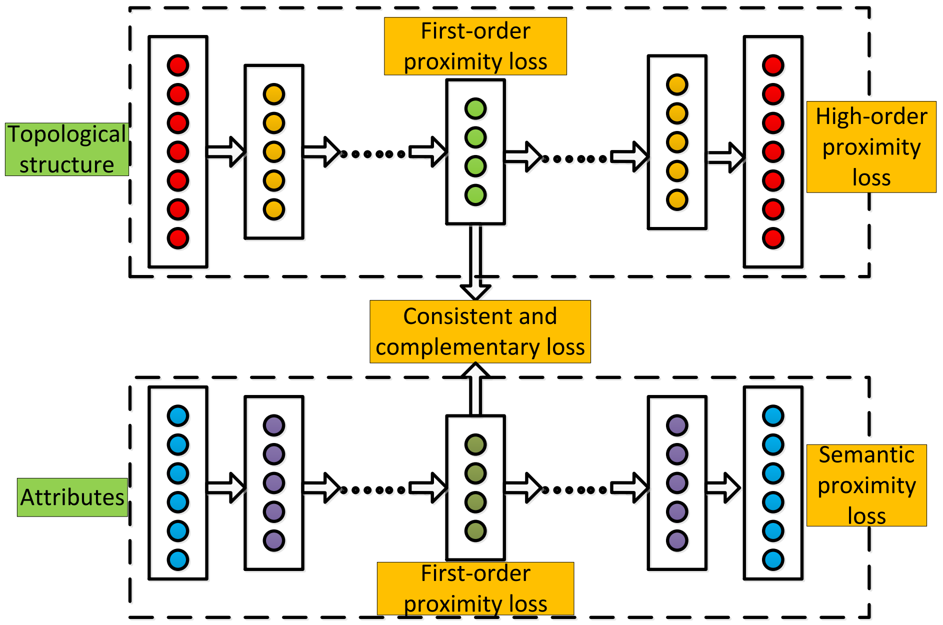

First-order proximity loss
Topological structure
High-order proximity loss
Consistent and complementary loss
Attributes
Semantic proximity loss
First-order proximity loss

Figure 11: The architecture of DANE.

5.2 Related Work

Plain Network Embedding Network embedding can be traced back to the graph embedding problem, such as Laplacian Eigenmaps [4], LPP [38]. These methods are to learn data embedding while preserving the local manifold structure. However, these methods are not applicable for large-scale network embedding, since they involve the time-consuming eigendecomposition operation whose time complexity is $O(n^3)$ where n is the number of nodes. Recently, with the development of large-scale networks, a variety of network embedding methods [72, 33, 12, 83, 27] have been proposed. For example, DeepWalk [72] adopts random walk and Skip-Gram to learn node representations, based on the observation that the distribution of nodes in the random walk is similar to that of words in natural language. LINE [83] proposes to preserve the first- and second-order proximity when learning node representations. GraRep [12] is proposed to preserve high order proximity. Furthermore, Node2Vec [33] is proposed by designing a biased random walk on the flexible node's neighborhood. However, all of these methods only utilize the topological structure, ignoring the useful attribute of nodes.

Attributed Network Embedding Attributed network embedding has attracted much attention in recent years. A wide variety of models have been proposed for attributed networks. For example, [92] proposes an inductive matrix factorization method to combine the topological structure and attributes of networks. However, it is a linear model essentially, which is not enough for the complicated attributed network. [42, 43] employ the graph Laplacian technique to learn the joint embedding from the topological structure and attributes. [48] proposes a graph convolutional neural network model for attributed networks. But this model is a semi-supervised approach, failing to deal with the unsupervised case. [71] proposes to combine DeepWalk with a neural network for network representation. Nonetheless, the DeepWalk part is still a shallow model. Recently, two unsupervised deep attributed network embedding approaches [49, 35] have been proposed. However, they can only explore the topological structure implicitly. Thus, it is necessary to explore the deep attributed network embedding method in a more effective way.

5.3 Deep Attributed Network Embedding

In this section, we first give the formal definition of attributed network embedding and then develop our novel deep attributed network embedding approach.

5.3.1 Problem Definition

Let $G = \{E, Z\}$ denote an attributed network with n nodes, where $E = [E_{ij}] \in \mathbb{R}^{n \times n}$ is the adjacency matrix and $Z = [Z_{ij}] \in \mathbb{R}^{n \times m}$ is the attribute matrix. In detail, if there exists an edge between the i-th node and the j-th node, $E_{ij} > 0$. Otherwise, $E_{ij} = 0$. $Z_{i\cdot} \in \mathbb{R}^m$ is the i-th row of Z, which denotes the attribute of the i-th node.

Before giving the problem definition, we first introduce the definition of different proximities, which is important for our approach.

Definition 2. *(First-Order Proximity [83]) Given a network $G = \{E, Z\}$. The first-order proximity of two nodes i and j is determined by E_{ij}. Specifically, a larger E_{ij} denotes a larger proximity between the i-th node and the j-th one.*

The first-order proximity indicates that if there exists a link between two nodes, they are similar. Otherwise, they are dissimilar. Thus, it can be viewed as a local proximity.

Definition 3. *(High-Order Proximity [12]) Given a network $G = \{E, Z\}$, the high-order proximity of two nodes i and j is determined by the similarity of $M_{i\cdot}$ and $M_{j\cdot}$, where $M = \hat{E} + \hat{E}^2 + \cdots + \hat{E}^t$ is the high-order proximity matrix, $\hat{E}$ is the 1-step probability transition matrix which is obtained from the row-wise normalization of the adjacency matrix E.*

The high-order proximity actually indicates the neighborhood similarity. Specifically, if two nodes share similar neighbors, they are similar. Otherwise, they are not similar. Here, the high-order proximity can be viewed as a global proximity.

Definition 4. *(Semantic Proximity) Given a network $G = \{E, Z\}$, the semantic proximity of two nodes i and j is determined by the similarity of $Z_{i\cdot}$ and $Z_{j\cdot}$.*

The semantic proximity denotes that if two nodes have similar attributes, they are similar. Otherwise, they are dissimilar.

Attributed network embedding is to learn the low-dimensional representation of each node based on the adjacency matrix E and the attribute matrix Z, such that the learned representation can preserve the proximity existing in the topological structure and node attributes. Formally, we aim at learning a map $f : \{E, Z\} \to H$ where $H \in \mathbb{R}^{n \times d}$ is the node representation, such that H can preserve *first-order proximity*, *high-order proximity* and *semantic proximity*. Then, downstream tasks, such as node classification and link prediction, can be performed on the learned H.

5.3.2 Deep Attributed Network Embedding

Essentially, the attributed network embedding faces three great challenges to obtain a good embedding result. They are:

• Highly non-linear structure: The underlying structure of the topological structure and attributes are highly non-linear, thus it is difficult to capture this non-linearity.

• Proximity preservation: The proximity in an attributed network depends both on the topological structure and the attribute, thus how to discover and preserve the proximity is a tough problem.

• Consistent and complementary information in the topological structure and attributes: These two kinds of information provide different views for each node, thus it is important to make the learned node representations preserve the consistent and complementary information in these two modalities.

To address these three challenges, we develop a novel deep attributed network embedding (DANE) approach. The architecture is shown in Figure 11. Overall, there are two branches where the first branch is composed of a multi-layer non-linear function, which can capture the highly non-linear network structure, to map the input M to the low-dimensional space. Similarly, the second branch is to map the input Z to the low-dimensional space to capture the high non-linearity in attributes.

5.3.2.1 Highly Non-linear Structure To capture the highly non-linear structure, each branch in Figure 11 is an autoencoder. Autoencoder is a powerful unsupervised deep model for feature learning. It has been widely used for various machine learning applications [46]. The basic autoencoder contains three layers, they are the input layer, the hidden layer, and the output layer, which are defined as follows:

$$\begin{aligned} h_i &= \sigma(W^{(1)} x_i + b^{(1)}), \\ \hat{x}_i &= \sigma(W^{(2)} h_i + b^{(2)}) \ . \end{aligned} \tag{5.1}$$

Here, $x_i \in \mathbb{R}^d$ is the i-th input data, $h_i \in \mathbb{R}^{d'}$ is the hidden representation from the encoder, and $\hat{x}_i \in \mathbb{R}^d$ is the reconstructed data point from the decoder. $\theta = \{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}\}$ are model parameters. $\sigma(.)$ denotes the non-linear activation function.

One can learn model parameters by minimizing the reconstruction error:

$$\min_{\theta} \sum_{i=1}^{n} ||\hat{x}_i - x_i||_2^2 \ . \tag{5.2}$$

To capture the high non-linearity in the topological structure and attributes, the two branches in Figure 11 employ K layers in the encoder as follows:

$$\begin{aligned} & h_i^{(1)} = \sigma(W^{(1)} x_i + b^{(1)}), \\ & \cdots \\ & h_i^{(K)} = \sigma(W^{(K)} h_i^{(K-1)} + b^{(K)}) \ . \end{aligned} \tag{5.3}$$

Correspondingly, there will be K layers in the decoder. Here, $h_i^{(K)}$ is the desired low-dimensional representation of the i-th node.

For our approach, the input of the first branch in Figure 11 is the high-order proximity matrix to capture the non-linearity in the topological structure. The explanation is delayed to the next section. The input of the second branch is the attribute matrix Z to capture the non-linearity in the attribute. Here, we denote the learned representation from the topological structure and attributes as H^M and H^Z respectively.

5.3.2.2 Proximity Preservation To preserve the **semantic proximity**, we minimize the reconstruction loss between the input Z of the encoder and the output $\hat{Z}$ of the decoder:

$$L_s = \sum_{i=1}^{n} \|\hat{Z}_{i\cdot} - Z_{i\cdot}\|_2^2 \ . \tag{5.4}$$

The reason is disclosed in [76]. Specifically, the reconstruction loss can enforce the neural network to capture the data manifold smoothly and thus can preserve the proximity among samples [87]. Therefore, by minimizing the reconstruction loss, our approach can preserve the semantic proximity in attributes.

Similarly, to preserve the **high-order proximity**, we also minimize the reconstruction loss as follows:

$$L_h = \sum_{i=1}^{n} \|\hat{M}_{i\cdot} - M_{i\cdot}\|_2^2 \ . \tag{5.5}$$

Specifically, the high-order proximity M denotes the neighborhood structure. If two nodes have similar neighborhood structure, which means $M_{i\cdot}$ and $M_{j\cdot}$ are similar, the learned representation $H_{i\cdot}^M$ and $H_{j\cdot}^M$ by minimizing the reconstruction loss will also be similar with each other.

As discussed before, we need to preserve the **first-order proximity** which captures the local structure. Recall Definition 2, two nodes are similar if there exists an edge between them. Thus, to preserve this proximity, we maximize the following likelihood estimation:

$$L_f = \prod_{E_{ij}>0} p_{ij} \ , \tag{5.6}$$

where p_{ij} is the joint probability between the i-th node and the j-th node. Note that we should preserve the first-order proximity in the topological structure and attributes simultaneously so that we can obtain the consistent result between these two kinds of information. For the topological structure, the joint probability is defined as follows:

$$p_{ij}^M = \frac{1}{1 + \exp(-H_{i\cdot}^M (H_{j\cdot}^M)^T)} \ . \tag{5.7}$$

Similarly, the joint probability based on the attribute is defined as follows:

$$p_{ij}^Z = \frac{1}{1 + \exp(-H_{i\cdot}^Z (H_{j\cdot}^Z)^T)} \ . \tag{5.8}$$

Thus, we can preserve the first-order proximity in the topological structure and attributes simultaneously by minimizing the negative log-likelihood as follows:

$$L_f = -\sum_{E_{ij}>0} \log p_{ij}^M - \sum_{E_{ij}>0} \log p_{ij}^Z \ . \tag{5.9}$$

5.3.2.3 Consistent and Complementary Representation

Since the topological structure and attributes are the two-modal information of the same network, we should guarantee the learned representation from them is consistent [24, 28]. On the other hand, these two kinds of information describe different aspects of the same node, providing complementary information. Thus, the learned representation should also be complementary. All in all, how to learn the *consistent* and *complementary* low-dimensional representation is very important.

A direct and simple method is to concatenate these two representations H^M and H^Z directly as the embedding result. Although this method can maintain the complementary information between two modalities, yet it cannot guarantee the consistency between these two modalities. Another widely used method is to enforce the two branches in Figure 11 to share the same highest encoding layer, that is $H^M = H^Z$. Although this method can guarantee the consistency between two modalities, it will lose too much complementary information from two modalities due to the exactly same highest encoding layer. Therefore, how to combine the topological structure and attributes together for attributed network embedding is a challenging problem.

To address this challenging problem, we propose to maximize the following likelihood estimation:

$$L_c = \prod_{i,j}^{n} p_{ij}^{s_{ij}} (1 - p_{ij})^{1-s_{ij}} \ , \tag{5.10}$$

where p_{ij} is the joint distribution between two modalities, which is defined as follows:

$$p_{ij} = \frac{1}{1 + \exp(-H_{i\cdot}^M (H_{j\cdot}^Z)^T)} \ . \tag{5.11}$$

Additionally, $s_{ij} \in \{0, 1\}$ denotes whether $H_{i\cdot}^M$ and $H_{j\cdot}^Z$ are from the same node. In detail, $s_{ij} = 1$ if $i = j$. Otherwise, $s_{ij} = 0$. Furthermore, Eq. (5.10) is equivalent to minimize the negative log-likelihood as follows:

$$L_c = -\sum_{i} \{\log p_{ii} - \sum_{j \neq i} \log(1 - p_{ij})\} \ . \tag{5.12}$$

By minimizing Eq. (5.12), we can enforce $H_{i\cdot}^{M}$ and $H_{j\cdot}^{Z}$ as consistent as possible when they are from the same node while pushing away them when they are from different nodes. On the other hand, they are not exactly same, thus they can preserve the complementary information in each modality.

However, the second term in the right hand side of Eq. (5.12) is over strict. For instance, if two nodes i and j are similar according to the first-order proximity, the representation $H_{i\cdot}^{M}$ and $H_{j\cdot}^{Z}$ should also be similar although they are from different nodes. That is to say we should not push them away. Thus, we relax Eq. (5.12) as follows:

$$L_c = -\sum_{i}\{\log p_{ii} - \sum_{E_{ij}=0}\log(1-p_{ij})\} \ . \tag{5.13}$$

Here, we push $H_{i\cdot}^{M}$ and $H_{j\cdot}^{Z}$ together when they are from the same node while pushing them away when two nodes are not connected. As a result, to preserve proximities and learn the consistent and complementary representation, we optimize the following objective function jointly:

$$\begin{aligned} L = &-\sum_{E_{ij}>0}\log p_{ij}^{M} - \sum_{E_{ij}>0}\log p_{ij}^{Z} + \sum_{i=1}^{n}\|\hat{M}_{i\cdot} - M_{i\cdot}\|_2^2 \\ &+\sum_{i=1}^{n}\|\hat{Z}_{i\cdot} - Z_{i\cdot}\|_2^2 - \sum_{i}\{\log p_{ii} - \sum_{E_{ij}=0}\log(1-p_{ij})\} \ . \end{aligned} \tag{5.14}$$

By minimizing this problem, we can obtain $H_{i\cdot}^{M}$ and $H_{i\cdot}^{Z}$, then we concatenate them as the final low-dimensional representation of the node such that we can preserve the consistent and complementary information from the topological structure and attributes.

5.3.3 Most Negative Sampling Strategy

Take a more detailed look at Eq. (5.13), it is to solve the following problem with respect to each node:

$$L_{c_i} = -\log p_{ii} - \sum_{j:s.t.E_{ij}=0}\log(1-p_{ij}) \ . \tag{5.15}$$

In practice, the adjacency matrix E is very sparse since many edges are not discovered. However, the undiscovered edge does not really mean two nodes are not similar. If we push away two potential similar nodes, the learned representation will become worse.

In detail, the gradient of L_{c_i} with respect to $H^M_{j\cdot}$ where $E_{ij} = 0$ is

$$\frac{\partial L_{c_i}}{\partial H^M_{j\cdot}} = p_{ij} H^Z_{i\cdot} \ . \tag{5.16}$$

According to the gradient descent method, the updating rule for $H^M_{j\cdot}$ is $H^M_{j\cdot} \leftarrow H^M_{j\cdot} - \alpha p_{ij} H^Z_{i\cdot}$ where α is the step size. Since $H^Z_{i\cdot}$ is constant when updating $H^M_{j\cdot}$, p_{ij} determines the updating rule. Furthermore, p_{ij} depends on $H^M_{i\cdot}(H^Z_{j\cdot})^T$. If two nodes i and j are similar potentially but there is no direct link, p_{ij} will be large so that $H^M_{j\cdot}$ will be pushed much away from $H^Z_{i\cdot}$. As a result, the embedding result will become worse and worse.

Table 13: Description of benchmark datasets

Dataset	#Nodes	#Edges	#Attributes	#Labels
Cora	2,708	5,278	1,433	7
Citeseer	3,312	4,660	3,703	6
Wiki	2,405	12,761	4,973	17
PubMed	19,717	44,338	500	3

To alleviate this problem, we propose a most negative sampling strategy to get a robust embedding result. Specifically, at each iteration, we calculate the similarity between H^M and H^Z by $P = H^M(H^Z)^T$. Then, for each node i, we select the most negative sample as follows:

$$j = \arg \min_{j, E_{ij}=0} P_{ij} \ . \tag{5.17}$$

Based on this negative sample, the objective function Eq. (5.15) becomes

$$L_{c_i} = -\log p_{ii} - \log(1 - p_{ij}) \ , \tag{5.18}$$

where j is sampled based on Eq. (5.17). With this most negative sampling strategy, the potential similar nodes will not be violated as far as possible. Thus, the embedding result will be more robust.

Sampling Complexity The sampling complexity is dominated by the computation of the similarity P whose complexity is $O(n^2)$ while the complexity of Eq. (5.13) is also $O(n^2)$.

Note that we omit the complexity of computing each p_{ij}. Thus, there is no much increased overhead to employ our sampling strategy. As a result, our proposed sampling strategy is efficient and effective.

5.4 Experiments

5.4.1 Experiment Settings

Datasets In our experiments, we employ four benchmark datasets [1]: Cora, Citeseer, PubMed, and Wiki. The first three datasets are paper citation networks. The edge of each network is the citation link. The attribute of each node is the bag-of-words representation of the corresponding paper. Wiki is a network with nodes as web pages. The link among different nodes is the hyperlink in the web page. We summarize the statistics of these benchmark datasets in Table 13.

Table 14: The architecture of our approach for different datasets.

Dataset	#neurons in each layer	Dataset	#neurons in each layer
Cora	2708-200-100	Citeseer	3312-200-100
	1433-200-100		3703-500-100
Wiki	2405-200-100	PubMed	19717-500-100
	4973-500-100		500-200-100

Baselines To evaluate the performance of our proposed DANE, we compare it with 9 baseline methods, including 4 plain network embedding methods and 5 attributed network embedding methods. The former category contains DeepWalk [72], Node2Vec [33], GraRep [12], and LINE [83]. The latter one includes TADW [92], ANE [43], Graph Auto-Encoder (GAE) [49], Variational Graph Auto-Encoder (VGAE) [49], and SAGE [35]. For DeepWalk and Node2Vec, we set the window size as 10, the walk length as 80, the number of walks as

[1]`https://linqs.soe.ucsc.edu/data`

10. For GraRep, the maximum transition step is set to 5. For LINE, we concatenate the first-order and second-order result together as the final embedding result. For the rest baseline methods, their parameters are set following the original papers. At last, the dimension of the node representation is set as 200.

To obtain the high-order matrix M in Definition 3, we utilize the random walk as DeepWalk [72] to obtain the high-order context of each node and then construct the adjacency matrix M with a sliding window size as 10. Moreover, the architecture of our approach for four datasets is summarized in Table 14. We use LeakyReLu [61] as the activation function.

Table 15: Node classification result of Cora.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.7568	0.7498	0.8064	0.7943	0.8287	0.8177
Node2Vec	0.7477	0.7256	0.8201	0.8121	0.8235	0.8162
GraRep	0.7568	0.7441	0.7927	0.7893	0.7999	0.7921
LINE	0.7338	0.7191	0.8122	0.8105	0.8353	0.8254
TADW	0.7510	0.7234	0.8006	0.7801	0.8354	0.8187
ANE	0.7203	0.7150	0.8027	0.7906	0.8117	0.7987
GAE	0.7691	0.7573	0.8059	0.7921	0.8095	0.7989
VGAE	**0.7888**	0.7736	0.8054	0.7909	0.8117	0.7994
SAGE	0.7633	0.7475	0.8096	0.7999	0.8154	0.8080
DANE	0.7867	**0.7748**	**0.8281**	**0.8127**	**0.8502**	**0.8377**

5.4.2 Results and Analysis

Node Classification To show the performance of our proposed DANE, we conduct node classification on the learned node representations. Specifically, we employ ℓ_2-regularized Logistic Regression as the classifier. To make a comprehensive evaluation, we randomly select $\{10\%, 30\%, 50\%\}$ nodes as the training set and the rest as the testing set respectively. With

Table 16: Node classification result of Citeseer.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.5052	0.4645	0.5783	0.5329	0.5900	0.5486
Node2Vec	0.5233	0.4832	0.6110	0.5651	0.6335	0.5972
GraRep	0.4817	0.4589	0.5511	0.5118	0.5707	0.5048
LINE	0.5139	0.4726	0.5761	0.5384	0.6075	0.5700
TADW	0.6048	0.5344	0.6481	0.5769	0.6578	0.5897
ANE	0.5877	0.5451	0.6718	0.6174	0.7071	0.6596
GAE	0.6058	0.5532	0.6550	0.5814	0.6540	0.5808
VGAE	0.6115	0.5662	0.6386	0.5824	0.6443	0.5837
SAGE	0.5351	0.4988	0.6304	0.5948	0.6528	0.6137
DANE	**0.6444**	**0.6043**	**0.7137**	**0.6718**	**0.7393**	**0.6965**

these randomly chosen training sets, we use five-fold cross-validation to train the classifier and then evaluate the classifier on the testing sets. To measure the classification result, we employ Micro-F1 (Mi-F1) and Macro-F1 (Ma-F1) as metrics. The classification results are shown in Table 15,16,17,18 respectively. From these four tables, we can find that our proposed DANE achieves significant improvement compared with plain network embedding approaches, and beats other attributed network embedding approaches in most situations.

Node Clustering To show the performance of our proposed approach on the unsupervised task, we conduct node clustering on the learned node representations. Here, we employ K-means as the clustering method and use clustering accuracy as the metric. The clustering result is summarized in Table 19. From this table, we can find that our approach achieves much better clustering performance than the others for most cases, which further verifies the effectiveness of our proposed DANE.

Table 17: Node classification result of Wiki.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.5621	0.4536	0.6479	0.5267	0.6675	0.5942
Node2Vec	0.5603	0.4131	0.6099	0.4760	0.6376	0.5203
GraRep	0.5801	0.4393	0.6223	0.5143	0.6642	0.5341
LINE	0.5806	0.4634	0.6538	0.5425	0.6766	0.5656
TADW	0.7266	**0.6300**	0.7565	0.6434	0.7764	0.6519
ANE	0.6263	0.5298	0.7144	0.5907	0.7298	0.6817
GAE	0.6245	0.4842	0.6526	0.5038	0.6567	0.5076
VGAE	0.6591	0.5215	0.6817	0.5621	0.7041	0.5790
SAGE	0.6259	0.5179	0.6764	0.5904	0.6866	0.6039
DANE	**0.7293**	0.6180	**0.7702**	**0.6597**	**0.7839**	**0.6838**

Network Visualization To further show the embedding result of our approach, we visualize the node representation by using t-SNE [62]. Due to the space limitation, we only post the result of three representative baseline methods for the Cora dataset. The visualization result is shown in Figure 12. From Figure 12, we can find that our approach can achieve more compact and separated clusters compared with the rest baseline methods. Thus, our approach can achieve better performance on both supervised and unsupervised tasks.

Effect of the Sampling Strategy To evaluate the effect of our sampling strategy, we compare it with two alternative methods. The first one is the random sampling method (DANE-RS). The second one is the importance sampling method (DANE-IS). For DANE-IS, we use $p^i = \frac{\exp(-P_{ij})}{\sum_j \exp(-P_{ij})}, \forall i$ as the sampling probability. Intuitively, this probability will let dissimilar points have large probability. With these two sampling methods, we learn node representations and then conduct node classification on the learned node representations for

Table 18: Node classification result of PubMed.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.8047	0.7873	0.8168	0.8034	0.8156	0.8034
Node2Vec	0.8027	0.7849	0.8110	0.7965	0.8103	0.7981
GraRep	0.7951	0.7785	0.8031	0.7901	0.8051	0.7937
LINE	0.8037	0.7892	0.8129	0.8007	0.8110	0.7994
TADW	0.8358	0.8343	0.8586	0.8584	0.8643	0.8633
ANE	0.7977	0.7875	0.8263	0.8191	0.8284	0.8203
GAE	0.8285	0.8238	0.8310	0.8263	0.8306	0.8257
VGAE	0.8299	0.8240	0.8350	0.8291	0.8361	0.8299
SAGE	0.8170	0.8090	0.8250	0.8160	0.8267	0.8176
DANE	**0.8608**	**0.8579**	**0.8731**	**0.8706**	**0.8775**	**0.8749**

Table 19: Clustering accuracy

Method	Cora	Citeseer	Wiki	PubMed
DeepWalk	0.6813	0.4145	0.4286	0.6660
Node2Vec	0.6473	0.4504	0.3792	0.6754
GraRep	0.5579	0.3777	0.3800	0.5501
LINE	0.4789	0.3913	0.4087	0.6614
TADW	0.5993	**0.6642**	0.4257	0.6257
ANE	0.3829	0.2557	0.3006	0.4486
GAE	0.6410	0.3677	0.4191	0.5915
VGAE	0.5546	0.3744	0.4445	0.6721
SAGE	0.6022	0.4589	0.4074	0.6020
DANE	**0.7027**	0.4797	**0.4731**	**0.6942**

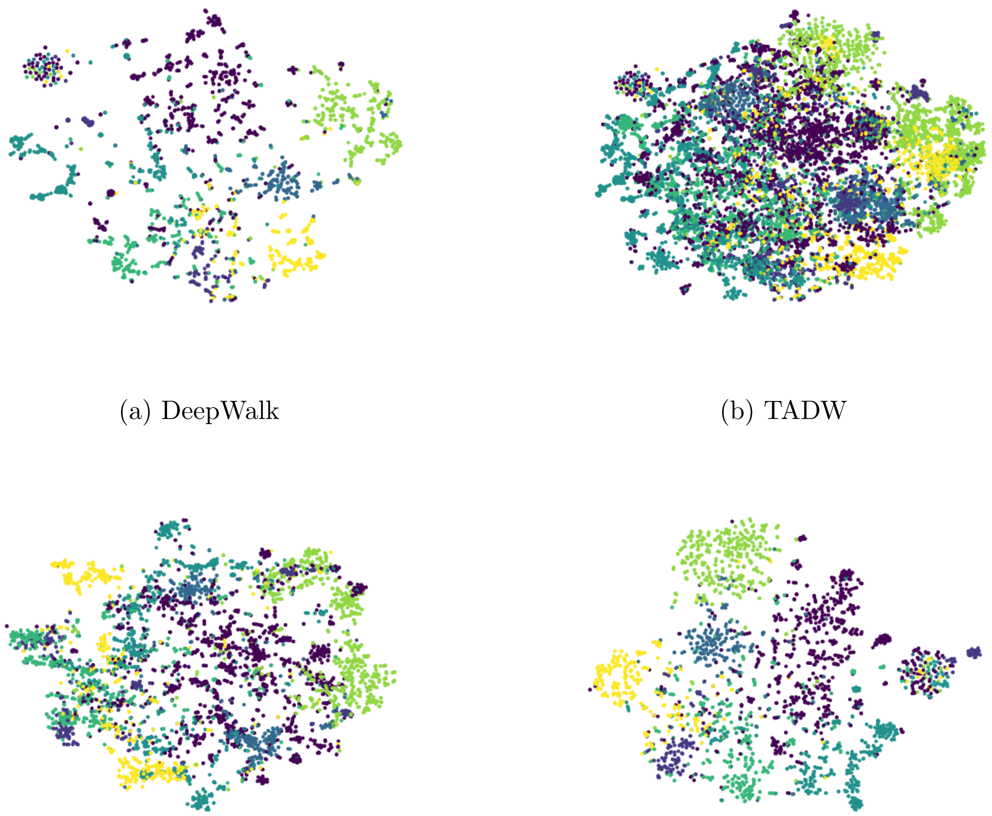
the Cora dataset. The result is shown in Table 20. We can find that DANE-RS has the worst performance since it fails to discriminate different negative samples. DANE-IS has a better result, but it is still worse than our method. Because our proposed sampling strategy can always find the most negative samples to refine node representations as far as possible.

Table 20: Cora: Node classification of different sampling methods.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DANE-RS	0.7301	0.7151	0.8038	0.7887	0.8125	0.8015
DANE-IS	0.7748	0.7635	0.8228	0.8091	0.8443	0.8308
DANE	**0.7867**	**0.7748**	**0.8281**	**0.8127**	**0.8502**	**0.8377**

5.5 Conclusion

In this chapter, we propose a novel deep attributed network embedding approach, which can capture the high non-linearity and preserve the proximity both in the topological structure and node attributes. Meanwhile, the proposed approach can learn a consistent and complementary representation from the topological structure and node attributes. The effectiveness has been verified by extensive experiments.



(a) DeepWalk

(b) TADW

(c) VGAE

(d) DANE

Figure 12: Visualization of different approaches for the Cora dataset.

6.0 Conditional Random Field Enhanced Graph Convolutional Neural Network

6.1 Introduction

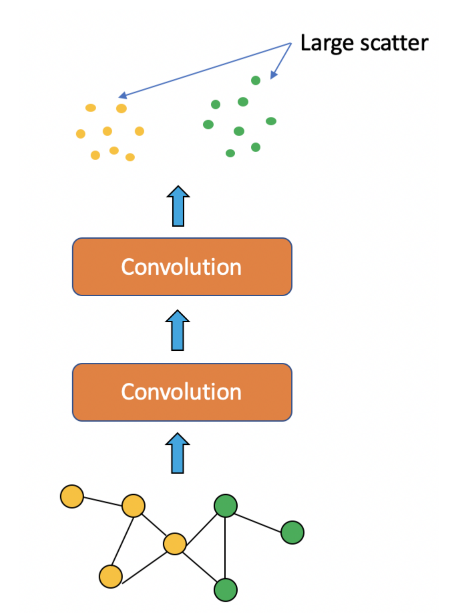
In recent years, deep convolutional neural networks have achieved great success in a wide variety of tasks, such as image classification [51, 37], image generation [51, 10, 26], and machine translation [73]. The basic idea of convolutional neural networks is to conduct the convolutional operation in a local neighborhood to explore the local correlation. For image data, there is a grid-like structure so that the implicit spatial order makes it easy to perform the convolutional operation in the local neighborhood. In practical applications, graph is a natural representation of numerous real-world data, such as social networks, knowledge graphs, citation networks. However, graphs have no the regular grid-like structure so that it is difficult to determine the local neighborhood to conduct the convolutional operation. To address this issue, the graph convolutional neural network has been proposed, which is designed to perform convolution on the complicated graph data. It has shown superior performance in various tasks, such as node classification [48], recommender system [95]. As a result, graph convolutional neural networks have attracted much attention in recent years.

Generally speaking, graph convolutional neural networks can be categorized into two classes: spatial approaches and spectral approaches. Specifically, spatial approaches [23, 3, 70, 35] conduct convolutions directly on the graph. More specifically, this kind of methods first construct a fixed-size neighborhood for each node in the graph and then perform regular convolution on this neighborhood. For instance, [70] proposes to construct the neighborhood from a fixed-size node sequence and then conduct the convolutional operation. Spectral approaches [11, 20] perform the convolutional operation in the spectral domain. In this way, spectral approaches do not need to construct the neighborhood explicitly for the complicated graph data. For instance, [11] defines the convolution in the Fourier domain which can be done in the eigenspace of the graph Laplacian. [20] proposes the fast localized spectral filtering for graph convolutional neural networks to avoid the expensive eigendecomposition. Recently, [48] further simplifies the spectral approach by directly aggregating the 1-hop neighboring nodes. With these development, graph convolutional neural networks have been successfully applied to a couple of tasks [77, 41, 48, 85, 99, 69, 57, 56], such as node classification [48, 85], relation extraction [99].

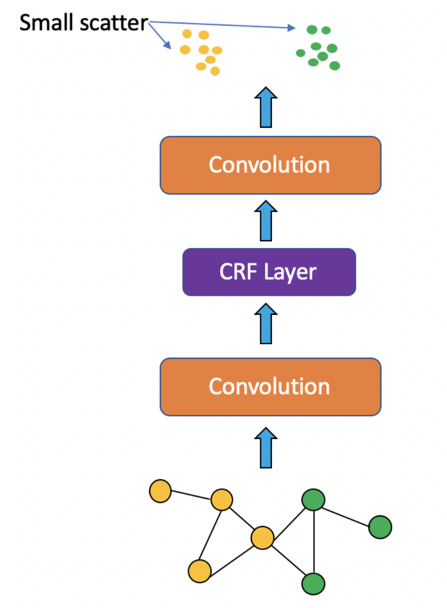
Although the aforementioned graph convolutional neural networks can deal with the graph data, yet they fail to fully utilize the properties of graphs. For the graph data, there exist edges which indicate the similarity relationship between different nodes. Two connected nodes imply that they are similar to each other while disconnected nodes indicate the dissimilarity between them. Existing graph convolutional neural networks can utilize this kind of connectivity information when performing the convolutional operation. For instance, the GCN proposed in [48] aggregates the 1-hop neighboring nodes such that the connectivity information is encoded into the new representation. On the other hand, the connectivity (similarity) information should also be preserved in the new representation (hidden features) obtained from the convolutional operation. However, although the convolutional operation can incorporate the connectivity information, yet it cannot guarantee the obtained hidden features preserve the similarity relationship explicitly. If this kind of relationship is violated in the hidden features, the downstream tasks will be degenerated severely. Thus, it is important and necessary to preserve the similarity information explicitly in the hidden layers of graph convolutional neural networks.

To preserve the similarity relationship in the new representation, numerous methods [2, 39, 75] have been proposed. The most widely used method is the Laplacian regularization, which has been used for various tasks, such as manifold learning [2, 39]. However, this method usually requires the expensive eigendecomposition. Thus, it is not suitable for large-scale neural networks. On the other hand, since we need to enforce the hidden layers of the graph convolutional neural network to satisfy the similarity constraint, it is necessary and important to use a lightweight operation which has small computational overhead and is easy to optimize by the backpropagation. Hence, it is challenging to restrict the behavior of the hidden layers of the graph convolutional neural network.

To address the aforementioned issues, in this chapter, we propose a novel CRF layer to regularize the standard graph convolutional neural network to preserve the similarity



(a) Standard Graph Convolutional Neural Network



(b) Graph Convolutional Neural Network with CRF Layer

Figure 13: The comparison between the standard graph convolutional neural network and that with the CRF layer. With the proposed CRF layer, the output features are expected to preserve the similarity better than those of the standard GCN.

relationship. Specifically, we resort to the CRF model to restrict the hidden feature of the graph convolutional layer. Then, we find that the solution of the CRF model can be viewed as an individual layer to encourage similar nodes to have similar hidden representations. As shown in Figure 13, this novel CRF layer can be inserted into standard graph convolutional neural networks to regularize the output of the convolutional operation. Furthermore, the proposed CRF layer is easy to compute and optimize. Thus, it is an efficient regularization layer to regularize the behavior of the hidden layers of graph convolutional neural networks. At last, we summarize the contribution of this chapter as follows:

- We propose a novel CRF layer for graph convolutional neural networks to encourage similar nodes to have similar hidden features.
- The proposed CRF layer is easy to compute and optimize. It can be inserted into existing graph convolutional neural networks easily.
- Extensive experimental results have verified the effectiveness of our proposed CRF layer.

6.2 Related Work

In this section, we will review related works about existing graph convolutional neural networks.

Recently, deep learning for the graph data has attracted increasing attention. A wide variety of methods [23, 3, 70, 35, 11, 20, 48, 25, 27, 30] have been proposed for this task. Among them, the graph convolutional neural network becomes more and more popular. Essentially, it is to conduct the convolutional operation on the non-Euclidean graph data. Compared with the regular convolutional neural network, there exists a challenge since different nodes in the graph have different size of the neighborhood while the regular convolutional operation requires a fixed-size neighborhood for each node. To address this issue, different approaches [23, 3, 70, 35, 11, 20, 48, 40, 2, 102, 50, 16] have been proposed. In terms of how to perform convolution, existing works can be categorized into two classes: spatial approaches and spectral approaches.

Spatial approaches [23, 3, 70, 35] conduct convolution directly on the graph. Recall that the regular convolution requires that the size of the neighborhood is fixed and nodes in the neighborhood are ordered. However, different nodes in a graph usually have different size of neighborhoods and have no ordering information. Thus, the goal of spatial approaches is to construct a fixed-size and ordered neighborhood to conduct the generic convolutional operation. For instance, [70] first selects a fixed-size sequence of nodes and then constructs the desired neighborhood from this sequence. As a result, the standard convolutional operation can be performed on this fixed-size neighborhood. [3] proposes a parametric construction method to obtain the neighborhood. [35] introduces an inductive method which randomly selects a fixed-size neighborhood and then aggregates the feature of these nodes in a specific manner.

Unlike spatial approaches, spectral approaches [11, 20, 48] try to perform the convolutional operation in the spectral domain rather than the spatial domain. In this way, spectral approaches do not require to explicitly construct the fixed-size neighborhood. For instance, [11] defines the convolution in the Fourier domain which can be done in the eigenspace of the graph Laplacian. However, the computational overhead is large due to the eigendecomposition. To address this issue, [20] proposes the fast localized spectral filtering for graph convolutional neural networks. Specifically, this method resorts to the Chebyshev expansion of the graph Laplacian to avoid the eigendecomposition. Thus, it has the same linear computational complexity as the standard CNN. Later, [48] further simplifies the spectral approach. In detail, [48] proposes to restrict the filters to perform only on the 1-hop neighboring nodes. In this way, it only needs to aggregates the features of these neighboring nodes, which is efficient compared with previous works. Then, graph convolutional neural networks become more and more popular and widely used for a wide variety of tasks. Recently, [85] proposes the graph attention network (GAT) which applies the attention mechanism to graph convolutional neural networks. Specifically, compared with standard graph convolutional neural network which aggregates features of neighboring nodes uniformly, GAT assigns different weights according to the importance of the neighboring node to the reference node.

Although the aforementioned methods can apply the convolutional operation to the graph data, yet they share a common drawback. Unlike the standard data, the graph data possess the connectivity information between different nodes. This connectivity information encodes the similarity relationship. In particular, two connected nodes indicate that they are similar, while disconnected nodes imply the dissimilarity between them. Although existing graph convolutional neural networks can utilize this kind of information when conducting the convolutional operation, yet they cannot guarantee the obtained new features from the convolutional operation satisfy the similarity constraint. Thus, if the new features violate the implicit constraint, the learned features will degenerate downstream tasks. All in all, it is important and necessary to enforce the learned new features to preserve the similarity relationship.

In fact, to deal with the similarity relationship, various works [2, 39, 75] have been proposed. For instance, the graph regularization method [2, 39] is a widely used approach to restrict the new features to preserve the similarity. This method has been successfully used for conventional machine learning models, such as manifold learning [39]. However, this method requires the expensive eigendecomposition of the graph Laplacian. Thus, it is not efficient for large-scale graphs. In addition, to handle the similarity relationship, the conditional random field (CRF) method is another potential choice. CRF is a probabilistic graphical model which can model the pairwise relationship. It is first proposed in [53] for predicting labels of the sequential data. After that, it has been applied to different tasks to encourage similar data points to have similar predictions. For instance, in the image segmentation task [17], CRF is used to refine the coarse pixel-level prediction by exploring the relationship between each pixel and their contexts. Recently, CRF is also applied to the information retrieval task [15] by modeling the pairwise similarity between the query data point and the gallery data point. Besides, a contemporary work, Conditional Graph Neural Field (CGNF) [60], also applies CRF to the graph convolutional neural network. In particular, CGNF utilizes CRF to exploit the correlation of node labels like the conventional CRF model [53, 17]. Conversely, our method does not use node labels in the CRF part. Thus, our method is totally different from CGNF.

6.3 Preliminary Knowledge

In this section, we will provide some preliminary knowledge about the conditional random field which is important for our proposed method.

Conditional random field (CRF) is a probabilistic graphical model, which is first proposed in [53] for predicting labels of the sequential data. Later, CRF is introduced to different tasks for structured prediction, such as image segmentation [17], information retrieval [15]. Essentially, CRF is capable of modeling the pairwise relationship between the reference data point and its context to refine the final prediction. Formally, given the input data x_i, CRF aims at predicting y_i by maximizing the conditional probability as follows:

$$P(y_i|x_i) = \frac{1}{Z(x_i)} \exp(-E(y_i|x_i)) \ , \tag{6.1}$$

where $Z(x_i)$ denotes the partition function which serves as the normalization factor, $E(y_i|x_i)$ represents the energy function. Here, the definition of y_i depends on the specific task. For instance, in the image segmentation task, y_i represents the label for each pixel. In our task, y_i denotes the new representation.

As for the energy function, it includes two components: the unary energy component and the pairwise energy component. The unary energy function gives the prediction for each individual data point. The pairwise energy function aims at capturing the correlation between the individual data point and its context to regularize the unary energy function. As a result, the prediction for each individual data point can benefit from its own information and its neighboring data points' information. Formally, the energy function is defined as follows:

$$E(y_i|x_i) = \psi_u(y_i, x_i) + \sum_j \psi_p(y_i, y_j, x_i, x_j) \ , \tag{6.2}$$

where $\psi_u(y_i, x_i)$ denotes the unary energy function while $\psi_p(y_i, y_j, x_i, x_j)$ represents the pairwise energy function. For instance, in the image segmentation task, the unary energy function predicts the label for individual pixels in terms of the property of each pixel. The pairwise energy function provides the context information to encourage similar pixels to have similar label assignment. Finally, the mean-field approximation method is usually used to optimize the CRF model.

Inspired by its capability of capturing the pairwise relationship between the reference data point and its context, we will propose to apply CRF to the graph convolutional neural network to preserve the similarity in hidden layers.

6.4 Methodology

Assume the input graph $G = \{A, X\}$ includes the adjacency matrix $A \in \mathbb{R}^{n \times n}$ and the node feature matrix $X \in \mathbb{R}^{n \times d}$ where n denotes the number of nodes and d represents the dimension of node features. Specifically, $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ gives the connectivity information between different nodes where $a_{ij} > 0$ represents there exists an edge between node i and node j, otherwise $a_{ij} = 0$. Based on these terminologies, the standard graph convolutional neural network (GCN) proposed in [48] can be represented as follows:

$$H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l+1)}) \ , \tag{6.3}$$

where $H^{(l)}$ denotes the representation of the graph nodes in the l-th layer, $W^{(l)}$ stands for the model parameter in the l-th layer, $\sigma(\cdot)$ represents the non-linear activation function. Additionally, $\hat{A}$ represents the normalized graph adjacency matrix $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. Here, $\tilde{A} = A + I$ where I is an identity matrix, and $\tilde{D}$ is a diagonal matrix with diagonal elements being $\tilde{D}_{ii} = \sum_{j=1}^{n} \tilde{A}_{ij}$.

Intuitively, GCN in Eq. (6.3) first aggregates features of neighboring nodes in terms of the adjacency matrix by using $\hat{A} H^{(l)}$ and then conducts the rest transformation. For the graph data, nodes are connected by edges. These edges carry the similarity relationship between different nodes. Thus, to learn effective representations $H^{(l+1)}$ from GCN, it is necessary to enforce $H^{(l+1)}$ to preserve the similarity relationship. Otherwise, $H^{(l+1)}$ cannot fully represent the input graph data, degenerating the downstream task. To address this issue, we will propose a novel method to restrict the behavior of $H^{(l+1)}$.

6.4.1 CRF Layer

In this section, following [48, 85], we consider the following problem whose objective function is defined as follows:

$$J(W; X, \hat{A}, Y) = \mathcal{L}(Y; F(X, \hat{A}, W)) \ , \tag{6.4}$$

where Y denotes the label of nodes for supervised tasks and the reconstruction of the graph for unsupervised task, W represents the all model parameters of graph convolutional neural networks, $F(\cdot, \cdot, \cdot)$ indicates the graph convolutional neural network mapping, and $\mathcal{L}(\cdot, \cdot)$ stands for the loss function. Following [13], the objective function of graph convolutional neural networks can be reformulated as the following one with the quadratic penalty:

$$J(W; X, \hat{A}, Y) = \mathcal{L}(Y; F_L(H^{(L-1)}, \hat{A}, W^{(L)})) + \sum_{l=1}^{L-1} \frac{\gamma}{2} \|H^{(l)} - F_l(H^{(l-1)}, \hat{A}, W^{(l)})\|_F^2 \ , \tag{6.5}$$

where $\gamma > 0$, L represents the total number of layers, and $F_l(\cdot, \cdot, \cdot)$ denotes the mapping in the l-the layer. For the GCN defined in Eq. (6.3), $F_l(\cdot, \cdot, \cdot)$ is $\sigma(\hat{A}H^{(l-1)}W^{(l)})$. In terms of [97], under mild conditions, the solution of Eq. (6.5) converges to that of Eq. (6.4) when $\gamma \to \infty$. From this new formulation, it is easy to find that there is no any constraint for $H^{(l)}$ to enforce it to satisfy the similarity constraint that a graph possesses. To address this problem, the high-level idea of our method is to enforce a regularization for $H^{(l)}$ to make it satisfy the similarity relationship. Specifically, our general objective function is defined as follows:

$$J(W; X, \hat{A}, Y) = \mathcal{L}(Y; F_L(H^{(L-1)}, \hat{A}, W^{(L)})) + \sum_{l=1}^{L-1} \frac{\gamma}{2} \|H^{(l)} - F_l(H^{(l-1)}, \hat{A}, W^{(l)})\|_F^2 + \mathcal{R}(H^{(l)}) \ , \tag{6.6}$$

where $\mathcal{R}(\cdot)$ denotes a regularization function for implementing the similarity constraint. In fact, there are numerous works for such kind of regularization in the traditional machine learning and data mining models. However, since our purpose is to restrict the behavior of the layers of graph convolutional neural networks, there exist several challenges:

- The regularization term should be easy to compute. It cannot have expensive computation.

- The regularization term should be friendly to the backpropagation strategy. Then, the neural network can still be optimized in the backpropagation way.
- The regularization term should be easy to plug into existing graph convolutional neural networks.

All in all, it is critical and challenging to get an efficient and effective regularization term.

To address the aforementioned challenges, we resort to the conditional random field (CRF) which can capture the pairwise relationship between different nodes. Specifically, the node representation $H^{(l)}$ is viewed as random variables $\{H_{i\cdot}^{(l)}\}$ where $H_{i\cdot}^{(l)}$ represents the i-th row of $H^{(l)}$, corresponding to the representation of node i. These random variables are conditioned on $\{B_{i\cdot}^{(l)}\}$ where $B_{i\cdot}^{(l)} = F_l(H_{i\cdot}^{(l-1)}, \hat{A}, W^{(l)})$ stands for the preliminary representation of node i obtained from the convolutional operation. Based on these terminologies, we have the following CRF model:

$$P(H^{(l)}|B^{(l)}) = \frac{1}{Z(B^{(l)})} \exp(-E(H^{(l)}|B^{(l)})) \ , \tag{6.7}$$

where $Z(\cdot)$ serves as the normalization factor and $E(\cdot)$ is the energy function. As we discussed earlier, the energy function includes two components: the unary energy component and the pairwise energy component. In this chapter, the unary function is defined as follows:

$$\psi_u(H_{i\cdot}^{(l)}, B_{i\cdot}^{(l)}) = \|H_{i\cdot}^{(l)} - B_{i\cdot}^{(l)}\|_2^2 \ . \tag{6.8}$$

In fact, minimizing this unary function will enforce the node representation $H_{i\cdot}^{(l)}$ to be close to that obtained from the convolutional operation. To capture the similarity relationship between different nodes, the pairwise function is defined as follows:

$$\psi_p(H_{i\cdot}^{(l)}, H_{j\cdot}^{(l)}, B_{i\cdot}^{(l)}, B_{j\cdot}^{(l)}) = g_{ij}\|H_{i\cdot}^{(l)} - H_{j\cdot}^{(l)}\|_2^2 \ , \tag{6.9}$$

where g_{ij} denotes the similarity between node i and node j. Intuitively, when g_{ij} is large, minimizing $\psi_p(H_{i\cdot}^{(l)}, H_{j\cdot}^{(l)}, B_{i\cdot}^{(l)}, B_{j\cdot}^{(l)})$ will enforce $H_{i\cdot}^{(l)}$ to be close to $H_{j\cdot}^{(l)}$. Otherwise, it will push $H_{i\cdot}^{(l)}$ away from $H_{j\cdot}^{(l)}$. As a result, similar nodes will be encouraged to have similar representations.

Finally, the energy function for node i is defined as follows:

$$E(H_{i\cdot}^{(l)}|B_{i\cdot}^{(l)}) = \alpha\|H_{i\cdot}^{(l)} - B_{i\cdot}^{(l)}\|_2^2 + \beta \sum_{j\in\mathcal{N}_i} g_{ij}\|H_{i\cdot}^{(l)} - H_{j\cdot}^{(l)}\|_2^2 \ , \tag{6.10}$$

where $\mathcal{N}_i$ denotes the neighborhood of node i. Here, we introduce two parameters $\alpha > 0$ and $\beta > 0$ to adjust the importance of the two energy functions. Obviously, the energy function defined in Eq. (6.10) will enforce the new representation in the l-th layer to be close to that obtained from the convolutional operation and encourage similar nodes to have similar new representations. In addition, comparing with Eq. (6.6), the pairwise energy function in Eq. (6.10) actually acts as the regularization to encourage the new representations $H^{(l)}$ to preserve the similarity relationship.

After obtaining the energy function, we need to derive a tractable and efficient updating rule for the new representations $H^{(l)}$. Here, we resort to the mean-field approximation method. The basic idea is to find a simple distribution to approximate $P(H^{(l)}|B^{(l)})$ rather than computing $P(H^{(l)}|B^{(l)})$ exactly. Specifically, we employ the simple distribution $Q(H^{(l)})$ to approximate $P(H^{(l)}|B^{(l)})$. This simple distribution can be represented by the product of independent marginal distributions as $Q(H^{(l)}) = \prod_{i=1}^{n} Q_i(H_{i\cdot}^{(l)})$. Then, we minimize the KL divergence between the original distribution $P(H^{(l)}|B^{(l)})$ and the simple distribution $Q(H^{(l)})$ as follows:

$$\min \mathrm{KL}(Q(H^{(l)})||P(H^{(l)}|B^{(l)})) \ . \tag{6.11}$$

As a result, we can get the optimal distribution $Q_i^*(H_{i\cdot}^{(l)})$ as follows:

$$\ln Q_i^*(H_{i\cdot}^{(l)}) = \mathbb{E}_{j\neq i}[\ln P(H_{j\cdot}^{(l)}|B_{j\cdot}^{(l)})] + const \ . \tag{6.12}$$

According to Eq. (6.7) and Eq. (6.10), we can get

$$Q_i^*(H_{i\cdot}^{(l)}) \sim \exp\Big(\alpha\|H_{i\cdot}^{(l)} - B_{i\cdot}^{(l)}\|_2^2 + \beta \sum_{j\in\mathcal{N}_i} g_{ij}\|H_{i\cdot}^{(l)} - H_{j\cdot}^{(l)}\|_2^2\Big) \ , \tag{6.13}$$

which indicates that $Q_i^*(H_{i\cdot}^{(l)})$ is a Gaussian function. As a result, the maximum probability is achieved at the expectation of $Q_i^*(H_{i\cdot}^{(l)})$. Then, by computing its expectation, we have the updating rule for the new representations $H^{(l)}$ as follows:

$$(H_{i\cdot}^{(l)})^{k+1} = \frac{\alpha B_{i\cdot}^{(l)} + \beta \sum_{j\in\mathcal{N}_i} g_{ij}(H_{j\cdot}^{(l)})^k}{\alpha + \beta\sum_{j\in\mathcal{N}_i} g_{ij}} \ . \tag{6.14}$$

Note that it is an iterative updating rule and $(H_{i\cdot}^{(l)})^k$ denotes $H_{i\cdot}^{(l)}$ in the k-th iteration. After K iterations, we set $H_{i\cdot}^{(l)} = (H_{i\cdot}^{(l)})^K$, which is the final node representations in the l-th layer. From Eq. (6.14), it can be seen that $H_{i\cdot}^{(l)}$ depends on not only the representation $B_{i\cdot}^{(l)}$ which is obtained from the convolutional operation but also the representation of its neighboring nodes. Especially, when the coefficient g_{ij} is large, which means that node j is more similar with node i, it will contribute more to $H_{i\cdot}^{(l)}$. In this way, similar nodes will have similar representations.

In Eq. (6.14), we need to compute the coefficient g_{ij} between two nodes. In fact, there are a couple of choices to implement it. In this chapter, we employ two approaches, which is shown as follows.

Gaussian The most straightforward approach is to use the Gaussian function to compute g_{ij} as follows:

$$g_{ij} = \exp\Big(\frac{B_{j\cdot}^{(l)} B_{i\cdot}^{(l)T}}{\|B_{j\cdot}^{(l)}\|_2 \|B_{i\cdot}^{(l)}\|_2} / \sigma^2\Big) \ , \tag{6.15}$$

where σ is set as a learnable parameter in this chapter. Note that $B_{i\cdot}^{(l)}$ is a *row vector*. Intuitively, a large inner product implies a large similarity. Then, node j will contribute more to the representation of node i. After obtaining g_{ij}, it is easy to update $H_{i\cdot}^{(l)}$ without expensive operations.

Neural Network Another choice to compute g_{ij} is the flexible neural network, which has a larger capability than Gaussian function to deal with the similarity between node i and node j. Specifically, we employ a one-layer MLP to compute it as follows:

$$g_{ij} = \frac{\exp(s_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(s_{ik})} \ , \tag{6.16}$$

where

$$s_{ij} = (W_\theta B_{i\cdot}^{(l)T})^T (W_\beta B_{j\cdot}^{(l)T}) \ . \tag{6.17}$$

Here, W_θ and W_β are the weights of neural networks. All these weights are learnable parameters which can be optimized with those of standard graph convolutional neural networks.

Finally, Eq. (6.14) can serve as an individual layer to plug into the existing graph convolutional neural networks. Specifically, as shown in Figure 13, for the GCN proposed in [48], we may have the following layer configuration:

$$\begin{aligned} &Conv\ Layer : B^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)}) \ , \\ &CRF\ Layer : (H_{i\cdot}^{(l)})^{k+1} = \frac{\alpha B_{i\cdot}^{(l)} + \beta \sum_{j\in\mathcal{N}_i} g_{ij}(H_{j\cdot}^{(l)})^k}{\alpha + \beta \sum_{j\in\mathcal{N}_i} g_{ij}} \ . \end{aligned} \tag{6.18}$$

Here, we call Eq. (6.14) as the CRF layer. In addition, for the CRF layer, we initialize $(H_{i\cdot}^{(l)})^0 = B_{i\cdot}^{(l)}$ and set the number of iterations to 2 in our experiment.

Obviously, with our proposed CRF layer, similar nodes will be encouraged to have similar representations. Consequently, the similarity relationship can be preserved in the layers of graph convolutional neural networks. In addition, it can also be find that it is friendly to back propagation. Thus, we can still use the backpropagation strategy to optimize this kind of neural networks. At last, we summarize our method in Algorithm 4.

Algorithm 4 Graph convolutional neural network with CRF layer.

```
Require: the number of iterations $K$
Ensure: $(H^{(l)})^K$
1: Graph Convolutional Layer
       $B^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)})$
2: CRF Layer
3:     for $k = 0, 1, \cdots, K-1$ do
4:         $(H_{i\cdot}^{(l)})^{k+1} = \frac{\alpha B_{i\cdot}^{(l)} + \beta \sum_{j\in\mathcal{N}_i} g_{ij}(H_{j\cdot}^{(l)})^k}{\alpha + \beta \sum_{j\in\mathcal{N}_i} g_{ij}}$
5:     end for
```

6.4.2 Discussion

Recently, [85] applies the attention mechanism to the graph convolutional neural network. That is the graph attention network (GAT) whose convolutional layer is defined as follows:

$$H_{i\cdot}^{(l+1)} = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ij} H_{j\cdot}^{(l)} W^{(l+1)}) \ , \tag{6.19}$$

Table 21: Descriptions of benchmark datasets.

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

where $H_{i\cdot}^{(l+1)}$ denotes the feature of the i-th node, α_{ij} represents the attention coefficient between the i-th node and the j-th node. In this way, if α_{ij} is large, which indicates that the j-th node is an important neighbor to the i-th node, it will contribute more to the new feature of the i-th node.

In our proposed CRF layer, there is also a coefficient g_{ij} which represents the coefficient between two nodes. However, the mechanism of our method is different from that of GAT. Specifically, although both GAT and our method aggregate neighboring nodes in terms of the similarity between different nodes, yet GAT acts before the non-linear activation function while our method happens after the non-linear activation function. Since the non-linear function cannot guarantee to preserve the structure of node distributions, the similarity may not be preserved after the convolutional operation. On the contrary, our method acts on the output of the non-linear activation function directly. Thus, it can guarantee the output of the convolutional layer to preserve the similarity relationship. In addition, our method is an iterative approach which can be dynamically affected by the updating of the neighboring nodes. Thus, our method is more flexible.

6.5 Experiments

In this section, we will conduct extensive experiments to verify the performance of proposed CRF layer.

6.5.1 Datasets

The datasets used in our experiments are same as [48, 85]. Specifically, there are three datasets: Cora, Citeseer, and Pubmed [79]. The nodes of these networks are documents from different topics. The citation between documents serves as the edges. The content of documents corresponds to node features. Here, node features are represented by the bag-of-words of the corresponding document. The details about these documents are described as follows:

- Citeseer is a research paper citation network. There are 3,327 papers from 6 topics. The total number of citations is 4,732. The dimension of node features is 3,703.
- Cora is also a citation network. It has 2,708 documents from 7 topics. Each document has 1,433 features. The number of citations between different documents is 5,429.
- Pubmed is another citation network. It consists of 19,717 documents from 3 classes. Each document has 500 features.

The statistics of these datasets are summarized in Table 21.

In our experiments, we conduct three tasks to evaluate the performance of our proposed method. They are unsupervised graph auto-encoder, semi-supervised node classification and supervised node classification. As for the unsupervised graph auto-encoder, we use the link prediction task to evaluate the performance of our method. Following [49], 5% and 10% edges are randomly selected as the validation and test sets. The rest edges are used for the training set. As for the semi-supervised task, following [48, 85], 20 nodes per class are labeled as the training set. 500 nodes are selected as the validation set to conduct model selection. After that, the trained model is evaluated on the testing set which has 1,000 nodes. As for the supervised task, following [16], we keep the same validation and testing sets as the semi-supervised task. The rest nodes are all labeled as the training set.

6.5.2 Experiment Settings

For the unsupervised task, we apply our proposed CRF layer to graph auto-encoder (GAE) [49], and we call it CRF-GAE. Following [49], there are two layers in GAE where the

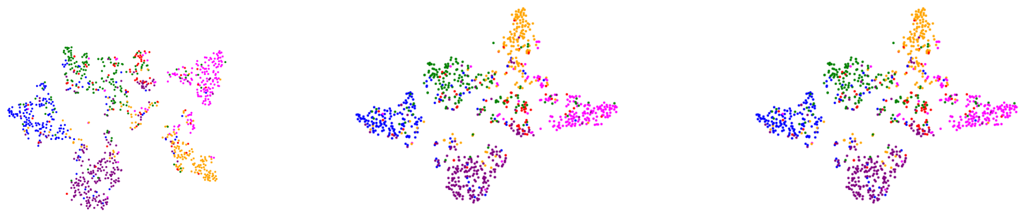

(a) GCN (b) CRF-GCN-Gaussian (c) CRF-GCN-NN

Figure 14: The visualization of features from the last layer for Citeseer.

dimension of first graph convolutional layer is 32 and that of the second graph convolutional layer is 16. Here, we insert our CRF layer after the first layer. Then, we keep the same training protocol as GAE to learn model parameters. In particular, we use Adam [47] with the learning rate being 0.1 to train CRF-GAE for 200 epochs. To evaluate the performance of CRF-GAE, following [49], we compare it with spectral clustering (SC), DeepWalk [72], and the regular GAE.

For the semi-supervised task, we compare it with various state-of-the-art methods. Following [85], the following semi-superivsed methods are employed as the baseline methods. They are Label Propagation (LP) [101], Semi-supervised Embedding (SemiEmb) [91], Manifold Regularization (ManiReg) [6], DeepWalk [72], Iterative Classification Algorithm (ICA) [58], Planetoid [93], the graph convolutional neural network with Chebyshev filters [20], and the MoNet method [68].

To show the performance of the CRF layer on the semi-supervised task, we apply it to two existing graph convolutional neural networks, including GCN [48] and GAT [85]. Here, we call them CRF-GCN and CRF-GAT. As for CRF-GCN, we employ the same network configuration as GCN. Specifically, there are two convolutional layers. The dimension of the hidden layer is 16. In our experiments, we insert the CRF layer after the first convolutional layer. All the weights are initialized with the Glorot method [31]. Additionally, we apply dropout [81] to all convolutional layers and the dropout ratio is set to 0.5. Furthermore, the weight decay whose parameter is 0.0005 is employed to regularize model parameters. The Adam [47] optimizer with a learning rate of 0.01 is utilized to optimize CRF-GCN.

As for CRF-GAT, there are also two convolutional layers. Similarly, we insert the CRF layer after the first convolutional layer. The other configuration is almost same with that of the original GAT. Specifically, for Cora and Citeseer, the first convolutional layer has 8 attention heads, each of whom has 8 features. The second convolutional layer has only one attention head and the number of its features equals the number of classes. Similar with CRF-GCN, the weight decay with $\lambda = 0.0005$ is employed to regularize the weights of the neural network. Additionally, the dropout with drop rate being 0.6 is applied to all convolutional layers and the normalized attention coefficients. For Pubmed, there are 8 attention heads at the second convolutional layer. In addition, the weight decay is increased to $\lambda = 0.001$. At last, all models are optimized by Adam [47] optimizer.

Table 22: The performance of link prediction.

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC	84.6	88.5	80.5	85.0	84.2	87.8
DeepWalk	83.1	85.0	80.5	83.6	84.4	84.1
GAE	91.0	92.0	89.5	89.9	96.4	96.5
CRF-GAE-Gaussian (ours)	**92.08**	**93.14**	**90.53**	**91.35**	**96.53**	**96.61**
CRF-GAE-NN (ours)	**92.25**	**93.04**	**90.90**	**91.80**	**96.51**	**96.63**

6.5.3 Results and Analysis

6.5.3.1 Unsupervised Task In Table 22, we report the performance of link prediction. Here, we use AUC (area under the ROC curve) and AP (average precision) to measure the performance. The larger value means better performance. Following [49], we run the experiments for 10 times and report the mean value. From Table 22, it can be seen that,

Table 23: The accuracy of semi-supervised node classification. Our results are marked in bold.

Methods	Cora	Citeseer	Pubmed
ManiReg [6]	0.595	0.601	0.707
SemiEmb [91]	0.590	0.596	0.717
LP [101]	0.680	0.453	0.630
DeepWalk [72]	0.672	0.432	0.653
ICA [58]	0.751	0.691	0.739
Planetoid [93]	0.757	0.691	0.739
Chebyshev [20]	0.812	0.698	0.744
MoNet [68]	0.817	-	0.788
GCN [48]	0.815	0.703	0.790
CRF-GCN-Gaussian (ours)	**0.828**	**0.718**	**0.790**
CRF-GCN-NN (ours)	**0.825**	**0.721**	**0.792**
GCN-64	0.814	0.709	0.790
GAT [85]	0.830	0.725	0.790
CRF-GAT-Gaussian (ours)	**0.846**	**0.731**	**0.791**
CRF-GAT-NN (ours)	**0.841**	**0.726**	**0.790**

equipped with our proposed CRF layer, both CRF-GAE-Gaussian and CRF-GAE-NN outperform the regular GAE model, which confirms the effectiveness of our proposed method for the unsupervised task.

6.5.3.2 Semi-supervised Task In Table 23, we report the classification accuracy of the semi-supervised task. Here, the results of state-of-the-art methods are extracted from the original GAT [85]. In addition, CRF-GCN-Gaussian denotes that g_{ij} is computed by using the Gaussian function while CRF-GCN-NN denotes that it is computed by using the neural

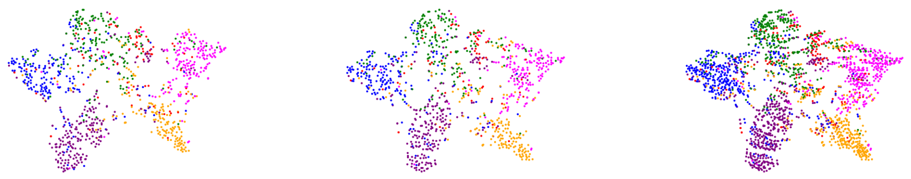

(a) GCN (b) Input of the CRF layer (c) Output of the CRF layer

Figure 15: The visualization of features from the hidden layer of GCN and CRF-GCN-Gaussian for Citeseer.

network. So does CRF-GAT. Note that since the dimension of the hidden layer of GAT is 64, we also report the result of GCN-64 which has 64 hidden features either.

From Table 23, it can be seen that the proposed CRF regularization layer does be helpful to improve the performance of existing graph convolutional neural networks. Specifically, both CRF-GCN-Gaussian and CRF-GCN-NN can beat the counterpart GCN for almost all datasets. For instance, CRF-GCN-NN can improve upon the standard GCN by 1.0% and 1.8% for Cora and Citeseer, respectively, which indicates that preserving the similarity relationship is beneficial. As for CRF-GAT-Gaussian, it improves upon the standard GAT by 1.6% and 0.6% for Cora and Citeseer, respectively. This further verifies the effectiveness of our proposed method.

To further show the effectiveness of our proposed method, we visualize the learned features in the last layer of GCN for Citeseer dataset. Specifically, in Figure 14. we plot the learned features of the testing set by using T-SNE [62]. It can be seen that the learned features from our proposed methods have a more compact structure than those learned from the standard GCN. In other words, the similarity relationship does be preserved better than the standard GCN. Thus, our method has a better classification performance.

Furthermore, to show the effect of the CRF layer, we visualize the hidden features of the CRF layer. Specifically, in Figure 15(a), we show the output feature of the first convolutional layer of the standard GCN for Citeseer dataset. In Figure 15(b) and Figure 15(c), we visualize the input and output feature of the CRF layer which follows the first convolutional layer of our proposed CRF-GCN-Gaussian, respectively. Compared Figure 15(b) with Figure 15(c), it can be seen that the output feature of the CRF layer have a more compact structure than the input. In other words, the CRF layer makes the features more compact to preserve the similarity. All these results have verified the effectiveness of our proposed method.

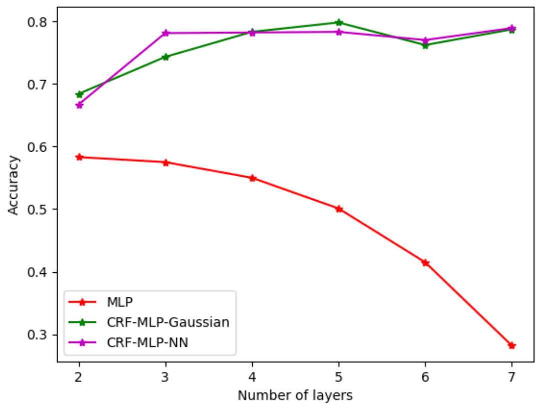


Figure 16: The semi-supervised classification accuracy of different MLP methods with different layers for Cora.

At last, to comprehensively verify the effectiveness of the CRF layer, we apply it to the standard fully connected neural networks. In detail, we insert the CRF layer after each fully connected layer other than the last layer. Here, we call them CRF-MLP-Gaussian and CRF-MLP-NN. In detail, the dimension of the hidden layer is set to 64. The training configuration, such as learning rate, weight decay parameter, dropout ratio, is same with CRF-GCN. In Figure 16, we demonstrate the testing accuracy of the semi-supervised classification for Cora. Here, we show the result of MLP with different layers. It can be seen that CRF-MLP-Gaussian and CRF-MLP-NN can outperform the standard MLP consistently and significantly. IN particular, the standard MLP performs worse when increasing the number of layers while our methods do not. The possible reason is that the hidden layer of the standard MLP cannot preserve the similarity. As a result, the learned representations violate the similarity relationship more severely when stacking more layers. On the contrary, due to the regularization of the CRF layer, our methods perform well when stacking multiple layers. Thus, based on the aforementioned observations, we can conclude that the proposed CRF layer can preserve the similarity relationship between different nodes effectively.

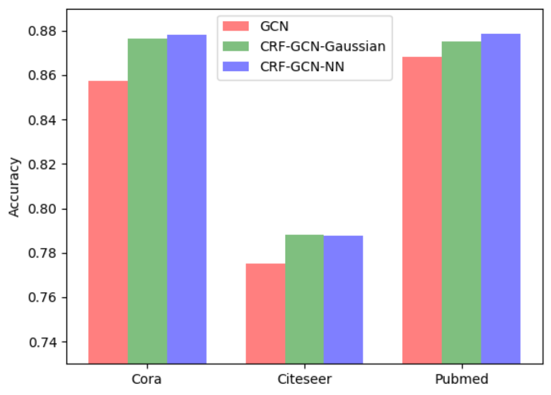


Figure 17: The supervised classification accuracy of different supervised GCN methods.

6.5.3.3 Supervised Task To further demonstrate the performance of our proposed method, we conduct the supervised node classification task as [16]. Specifically, we also insert the CRF layer after the first convolutional layer of GCN. Additionally, in this task, we employ the residual connection in the CRF layer as follows:

$$\hat{H}^{(l)} = (H^{(l)})^K + B^{(l)} \ , \tag{6.20}$$

where $(H^{(l)})^K$ is the last iteration of the CRF layer.

Similar with the semi-supervised task, we use the early stop strategy to terminate the training procedure when the loss function is not further decreased. The classification result is reported in Figure 17. It can be seen that our proposed methods outperform the standard GCN for almost all cases, which demonstrates the effectiveness of our proposed method.

Furthermore, we also apply the proposed CRF layer to fully connected neural networks for the supervised task. Following the configuration of the semi-supervised task, the dimension of hidden layers is also set to 64. In this experiment, there is only one hidden layer and the proposed CRF layer is inserted after the hidden layer. The classification result is shown in

Table 24: The supervised classification accuracy of MLP.

Dataset	Cora	Citeseer	Pubmed
MLP	0.733	0.757	0.863
CRF-MLP-Gaussian	0.873	0.799	0.886
CRF-MLP-NN	0.852	0.783	0.884

Table 25: The semi-supervised classification accuracy of Cora dataset. The "const" variant represents $\alpha = 1$ and $\beta = 1$.

Dataset	Cora	Citeseer	Pubmed
CRF-GCN-Gaussian	0.828	0.718	0.790
CRF-GCN-Gaussian-const	0.813	0.698	0.791
CRF-GCN-NN	0.825	0.721	0.792
CRF-GCN-NN-const	0.811	0.700	0.788

Table 24. It can be seen that MLP with the proposed CRF layer significantly outperforms the standard MLP. The underlying reason is that the proposed CRF layer can preserve the similarity relationship in the hidden layer. As a result, the learned representation for nodes is discriminative, benefiting the classification task.

6.5.3.4 Ablation Study In the proposed CRF layer, there are two important parameters: α and β, which are used to adjust the importance of $B^{(l)}$ and $H^{(l)}$. In this chapter, we set them as the learnable parameters. To show the importance of these parameters, we compare it with the variants: CRF-GCN-Gaussian-const and CRF-GCN-NN-const, which set $\alpha = 1$ and $\beta = 1$. In Table 25, we report the semi-supervised classification accuracy. It can be seen that these two variants degrade the performance significantly for almost all cases. Thus, it is necessary to learn these two parameters automatically.

6.6 Conclusion

In this chapter, we propose a novel CRF layer for the graph convolutional neural network. Specifically, by resorting to the CRF model for the hidden layers of graph convolutional neural networks to explore the similarity relationship, we obtain an efficient CRF layer, which can encourage the hidden features to preserve the similarity between different nodes. In addition, the proposed CRF layer is easy to compute and optimize so that it can be inserted into existing graph convolutional neural networks to improve their performance. The extensive experimental results have verified the effectiveness of our proposed method.

7.0 Conclusion

Unsupervised feature learning on graphs is an important and challenging task. It has attracted a surge of attention in recent years with the emergence of large-scale graph data. In this thesis, we study how to learn effective node features with deep neural networks. We have deeply incorporated the domain knowledge into the design of new deep neural networks to address the challenging issues in this task.

Essentially, the task of feature learning on graphs can be decoupled into *discovering* the proximity in the original space and *preserving* it in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. To discover the underlying proximity, we have proposed a new proximity generative adversarial network. It can generate the triplet of nodes to approximate the underlying true proximity so that the underlying proximity can be discovered and then preserved in the low-dimensional node representations. The extensive experiments on node classification, community detection, and network visualization validate the effectiveness of our new deep neural networks in discovering proximities.

For discovering the proximity, we also developed a new self-paced network embedding method to dynamically select the negative node for a given anchor node to push together similar nodes and push away dissimilar nodes in the low-dimensional space. Especially, our ASeedNE method, which is based on the generative adversarial network, has large model capacity to select more informative nodes to learn better node representations. This new neural network also shows superior performance on various tasks, such as link prediction and network visualization.

Attributed networks bring more challenges to feature learning on graphs due to the coupling of topological structure and node attributes. We have proposed a new deep auto-encoder neural network to capture and preserve various proximities in both topological structure and node attributes. Additionally, the learned node representation can encode the consistent and complementary information from both the topological structure and node attributes. Extensive experiments have verified the effectiveness of our proposed approach.

At last, we have studied how to couple graph neural networks with graphical models to benefit learning node representations. In particular, we have proposed a conditional random field layer to regularize the hidden layers of graph neural network to preserve the proximity. This layer is easy to be applied to existing graph neural networks, such as GAE, GCN, GAT. It is worth noting that our proposed CRF layer does not depend on the label. Thus, it is friendly for both unsupervised and supervised tasks. To the best of our knowledge, this is the first work that can couple graphical model and graph neural network in an unsupervised manner. We believe our work opens a new way to explore these two important techniques. Extensive experiments on various architectures have confirmed the superior performance of our proposed new neural networks.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning.

[2] R. K. Ando and T. Zhang. Learning on graph with laplacian regularization. In *Advances in neural information processing systems*, pages 25–32, 2007.

[3] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

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

[6] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

[7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[8] A. Bojchevski and S. Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[9] B.-J. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. Livstone, R. Oughtred, D. H. Lackner, J. Bähler, V. Wood, et al. The biogrid interaction database: 2008 update. *Nucleic acids research*, 36(suppl 1):D637–D640, 2008.

[10] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[12] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[13] M. Carreira-Perpinan and W. Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.

[14] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128. ACM, 2015.

[15] D. Chen, D. Xu, H. Li, N. Sebe, and X. Wang. Group consistent similarity learning via deep crf for person re-identification. In *CVPR*, 2018.

[16] J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[18] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2018.

[19] Q. Dai, Q. Li, J. Tang, and D. Wang. Adversarial network embedding. *arXiv preprint arXiv:1711.07838*, 2017.

[20] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[23] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[24] H. Gao, C. Cai, J. Yan, L. Yan, J. G. Cortes, Y. Wang, F. Nie, J. West, A. Saykin, L. Shen, et al. Identifying connectome module patterns via new balanced multi-graph normalized cut. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 169–176. Springer, 2015.

[25] H. Gao and H. Huang. Deep attributed network embedding. In *IJCAI*, 2018.

[26] H. Gao and H. Huang. Joint generative moment-matching network for learning structural latent code. In *IJCAI*, 2018.

[27] H. Gao and H. Huang. Self-paced network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1406–1415. ACM, 2018.

[28] H. Gao, F. Nie, X. Li, and H. Huang. Multi-view subspace clustering. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4238–4246. IEEE Computer Society, 2015.

[29] H. Gao, J. Pei, and H. Huang. Conditional random field enhanced graph convolutional neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[30] H. Gao, J. Pei, and H. Huang. Progan: Network embedding via proximity generative adversarial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[31] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[32] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[34] M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

[35] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[36] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[38] X. He and P. Niyogi. Locality preserving projections. In *NIPS*, volume 16, 2003.

[39] X. He and P. Niyogi. Locality preserving projections. In *Advances in neural information processing systems*, pages 153–160, 2004.

[40] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[41] Y. Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[42] X. Huang, J. Li, and X. Hu. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 633–641. SIAM, 2017.

[43] X. Huang, J. Li, and X. Hu. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 731–739. ACM, 2017.

[44] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*, pages 2078–2086, 2014.

[45] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann. Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2694–2700. AAAI Press, 2015.

[46] W. Jiang, H. Gao, F.-l. Chung, and H. Huang. The l2, 1-norm stacked robust autoencoders for domain adaptation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1723–1729. AAAI Press, 2016.

[47] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[49] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[50] J. Klicpera, A. Bojchevski, and S. Günnemann. Combining neural networks with personalized pagerank for classification on graphs. *arXiv*, 2018.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[52] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.

[53] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[54] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[55] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396. ACM, 2017.

[56] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 537–546, 2018.

[57] X. Liang, Z. Hu, H. Zhang, L. Lin, and E. P. Xing. Symbolic graph reasoning meets convolutions. In *Advances in Neural Information Processing Systems*, pages 1858–1868, 2018.

[58] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.

[59] F. Ma, D. Meng, Q. Xie, Z. Li, and X. Dong. Self-paced co-training. In *International Conference on Machine Learning*, pages 2275–2284, 2017.

[60] T. Ma, C. Xiao, J. Shang, and J. Sun. Cgnf: Conditional graph neural fields. 2018.

[61] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models.

[62] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[63] M. Mahoney. Large text compression benchmark. *URL: http://www. mattmahoney. net/text/text. html*, 2009.

[64] P. V. Marsden and N. E. Friedkin. Network studies of social influence. *Sociological Methods & Research*, 22(1):127–151, 1993.

[65] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[66] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[67] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[68] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, page 3, 2017.

[69] M. Narasimhan, S. Lazebnik, and A. Schwing. Out of the box: Reasoning with graph convolution nets for factual visual question answering. In *Advances in Neural Information Processing Systems*, pages 2659–2670, 2018.

[70] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[71] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang. Tri-party deep network representation. In *International Joint Conference on Artificial Intelligence*. AAAI Press/International Joint Conferences on Artificial Intelligence, 2016.

[72] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[73] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[74] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

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

[76] R. Salakhutdinov and G. Hinton. Semantic hashing. *RBM*, 500(3):500, 2007.

[77] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

[78] J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

[79] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[80] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[81] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[82] J. S. Supancic III and D. Ramanan. Self-paced learning for long-term tracking. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2379–2386. IEEE, 2013.

[83] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[84] L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826. ACM, 2009.

[85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[86] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[87] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[88] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267*, 2017.

[89] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM, 2017.

[90] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. 2017.

[91] J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[92] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. 2015.

[93] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[94] Z. Yang, J. Guo, K. Cai, J. Tang, J. Li, L. Zhang, and Z. Su. Understanding retweeting behaviors in social networks. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1633–1636. ACM, 2010.

[95] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. *arXiv preprint arXiv:1806.01973*, 2018.

[96] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.

[97] H. Zhang, W. Chen, and T.-Y. Liu. On the local hessian in back-propagation. Microsoft Research Asia, December 2018.

[98] W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 785–788. ACM, 2013.

[99] Y. Zhang, P. Qi, and C. D. Manning. Graph convolution over pruned dependency trees improves relation extraction. *arXiv preprint arXiv:1809.10185*, 2018.

[100] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision*, 2017.

[101] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[102] C. Zhuang and Q. Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 499–508. International World Wide Web Conferences Steering Committee, 2018.