Neural Gaussian Similarity Modeling for Differential Graph Structure Learning

Xiaolong Fan[1], **Maoguo Gong**[1], **Yue Wu**[2], **Zedong Tang**[3], **and Jieyi Liu**[1],

[1] School of Electronic Engineering, Key Laboratory of Collaborative Intelligence Systems of Ministry of Education, Xidian University, Xi'an, China
[2] School of Computer Science and Technology, Key Laboratory of Collaborative Intelligence Systems of Ministry of Education, Xidian University, Xi'an, China
[3] Academy of Advanced Interdisciplinary Research, Key Laboratory of Collaborative Intelligence Systems of Ministry of Education, Xidian University, Xi'an, China
xiaolongfan@outlook.com, gong@ieee.org, {ywu, zdtang, jieyiliu}@xidian.edu.cn

Abstract

Graph Structure Learning (GSL) has demonstrated considerable potential in the analysis of graph-unknown non-Euclidean data across a wide range of domains. However, constructing an end-to-end graph structure learning model poses a challenge due to the impediment of gradient flow caused by the nearest neighbor sampling strategy. In this paper, we construct a differential graph structure learning model by replacing the non-differentiable nearest neighbor sampling with a differentiable sampling using the reparameterization trick. Under this framework, we argue that the act of sampling nearest neighbors may not invariably be essential, particularly in instances where node features exhibit a significant degree of similarity. To alleviate this issue, the bell-shaped Gaussian Similarity (GauSim) modeling is proposed to sample non-nearest neighbors. To adaptively model the similarity, we further propose Neural Gaussian Similarity (NeuralGauSim) with learnable parameters featuring flexible sampling behaviors. In addition, we develop a scalable method by transferring the large-scale graph to the transition graph to significantly reduce the complexity. Experimental results demonstrate the effectiveness of the proposed methods.

Introduction

In recent years, there has been a notable surge in academic attention towards Graph Neural Networks (GNNs) (Kipf and Welling 2017; Zeng et al. 2020; Lim et al. 2021; Fan et al. 2023a). A plethora of graph neural network models have been introduced, showcasing noteworthy advancements across diverse domains such as social network analysis (Gao et al. 2023), natural language processing (Meng et al. 2022), computer vision (Han et al. 2022), and various other fields. The efficacy of graph neural network can be attributed to their inherent capability of effectively leveraging the abundant information present within both the structure of graph topology and the input node attributes in a concurrent manner. However, the graph structure is not invariably ascertainable. For instance, within a social network, the interconnections among users encompass sensitive privacy aspects, impeding direct access to such information.

To alleviate this issue, Graph Structure Learning (GSL) (Chen, Wu, and Zaki 2020; Fatemi, El Asri, and Kazemi 2021; Wu et al. 2022) is proposed to jointly learn the latent graph structure and corresponding graph embeddings using structure learner and graph neural network encoder where the parameters are optimized together by the downstream task. Specifically, the structure learner first computes the similarity between node feature pairs as the edge weights by using similarity kernel function, such as inner-product kernel (Yu et al. 2021; Zhao et al. 2021), cosine similarity kernel (Wang et al. 2020; Chen, Wu, and Zaki 2020), and diffusion kernel (Gasteiger, Weißenberger, and Günnemann 2019). Then the graph structure is generated via a structure sampling process from edge weight distribution. After producing the graph structure, graph neural network encoder takes the node features and the generated graph structure as input to produce the final node embeddings for downstream tasks. However, two fundamental weaknesses of this framework may limit the performance and scalability of graph structure learning method. First, *the generated graph structure is not differentiable with respect to the edge weight distribution due to discrete sampling blocking gradient flow.* Second, *computing the edge similarity for all pairs of graph nodes requires huge complexity for both computational time and memory consumption, rendering significant scalability issue for large graphs.*

To solve the problem of non-differentiable sampling, we utilize the concrete relaxation of the Categorical distribution by replacing the non-differentiable sampling with a differentiable sampling mechanism using the Gumbel-Softmax distribution. The Gumbel-Softmax distribution (Jang, Gu, and Poole 2017; Zheng et al. 2020; Sun et al. 2022) is a reparameterization trick that allows for the generation of discrete samples while maintaining differentiability. Note that the sampling probability is positively correlated with edge similarity, whereby greater similarity values entail higher probabilities of sampling the edges. We term this sampling approach as the linear sampling strategy. Our analysis reveals that this linear sampling method is not universally indispensable, particularly in situations where node features demonstrate a substantial level of similarity. To alleviate this issue, we propose a bell-shaped Gaussian Similarity (GauSim) modeling strategy, enabling the edge sampling probability entails an initial increase followed by a decrease as the similarity between node pairs diminishes. Note that different Gaussian function parameters need to be set for different

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

node pairs, we further propose a Neural Gaussian Similarity (NeuralGauSim) modeling strategy to adaptively learn the bell-shaped Gaussian function parameters.

To address the scalability issue, we develop a transition graph structure learning method by involving the transformation of the initial node set into the more streamlined transition node set. Specifically, we first project the initial feature matrix into the transition feature matrix. Then for every node in the original graph, we calculate a similarity score between the corresponding node in the original graph and its counterpart in the transferred graph. By leveraging this similarity score, we differentiably sample the edges to generate the desired graph structure. Finally, the node embeddings can be generated by using the graph neural network encoder which takes the generated graph structure and the node features of the transition graph as input. Different from the previous anchor-based graph structure leaning method (Chen, Wu, and Zaki 2020), which randomly samples a set of anchors from the initial graph, the developed transition graph method adopts projection matrices to learn to generate the transition graph, thus mitigating the information loss that typically arises from random sampling.

Extensive experiments on graph and graph-enhanced application datasets demonstrate the superior effectiveness of the proposed method. To summarize, we outline the main contributions in this paper as follows:

1. We propose the neural Gaussian similarity modeling for differential graph structure learning to alleviate the issue of structure sampling.
2. We develop the transition graph structure learning by transferring the initial graph to the transition graph to reduce the complexity.
3. Extensive experiments on graph and graph-enhanced application datasets demonstrate the superior effectiveness of the proposed method.

Related Work

Graph Neural Network

Graph Neural Networks (GNNs) aim to model the non-Euclidean data structure and have been demonstrated to achieve state-of-the-art performance on graph analysis tasks. As a unified framework for graph neural networks, Message Passing Neural Network (MPNN) (Scarselli et al. 2008; Fan et al. 2022, 2023b) generalizes the several existing representative graph neural networks, such as GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAINT (Zeng et al. 2020), AM-GCN(Wang et al. 2020), and LINKX (Lim et al. 2021), which consists of two functions, i.e., message passing function and readout function. Most empirical studies of graph neural networks directly take the observed graph as input. However, the graph structure is not invariably ascertainable in practice. In this paper, we focus on the graph-unknown non-Euclidean data representation learning.

Graph Structure Learning

Graph Structure Learning (GSL) targets at jointly learning an optimized graph structure and its corresponding representations. A typical GSL model involves two trainable components, i.e., structure learner and graph neural network encoder. The structure learner is an encoding function that models the optimal graph structure represented in edge weights. In recent years, several structure learners have been proposed, such as LDS (Franceschi et al. 2019), Pro-GNN (Jin et al. 2020), IDGL (Chen, Wu, and Zaki 2020), SLAPS (Fatemi, El Asri, and Kazemi 2021), and NodeFormer (Wu et al. 2022), and achieved significant performance improvement. In this paper, we propose the Gaussian similarity modeling strategy and transition graph structure learning method to alleviate the issues of structure sampling and scalability.

Exploring Graph Structure Learning

Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with $\mathcal{V}$ and $\mathcal{E}$ denoting the node set and edge set, respectively. The node feature matrix is denoted by $X = \{x_1, .., x_i, ...x_n\}$ where $x_i \in \mathbb{R}^d$ is the attribute of node v_i. The graph structure is described by the adjacency matrix $A \in \{0, 1\}^{n \times n}$ for graphs where $A_{ij} = 1$ indicates $(v_i, v_j) \in \mathcal{E}$. In general, a GNN encoder, parameterized by Θ, receives the graph structure and node features as input, then produces node embeddings $H \in \mathbb{R}^{n \times m}$ for downstream tasks. This paper primarily centers on the exploration of graph representation learning in the context of the unknown graph structure. In this setting, graph structure learning can be formulated as producing the graph structure A^* and its corresponding node embeddings $H = \text{GNN}(A^*, X)$ with respect to the downstream tasks.

Differential Graph Structure Learning

Graph structure learning aims to jointly learn the graph structure and corresponding graph embeddings. Given node features, graph structure learning first models the similarity between node feature pairs in form of

$$z_i = \text{MLP}(x_i) \tag{1}$$

$$\pi_{i,j} = \frac{\exp(z_i z_j^\top)}{\sum_{w=1}^{n} \exp(z_i z_w^\top)} \tag{2}$$

where $\text{MLP}(\cdot)$ denotes a multi-layer perceptron and $\pi_{i,j}$ denotes the edge similarity between node v_i and v_j. Then the graph structure A^* can be generated using a structure sampling strategy that models all potential edges as a collection of independent Categorical random variables. These variables are parameterized by the learned similarity π in form of

$$A^* = \bigcup_{v_i, v_j \in \mathcal{V}} \{A_{i,j} \sim \text{Cat}(\pi_{i,j})\} \tag{3}$$

where $A_{i,j} \sim \text{Cat}(\pi_{i,j})$ denotes the edge sampling process from Categorical distribution. Here, the similarity $\pi_{i,j}$ describes the edge sampling probability and smaller $\pi_{i,j}$ indicates that the edge (v_i, v_j) tends to be removed. In general, we sample $\mathcal{K}$ times for each node v_i to form neighbors. However, this method poses a challenge that the graph structure A^* is not differentiable with respect to π due to discrete sampling blocking gradient flow. To solve this problem, we can utilize the concrete relaxation of the Categorical

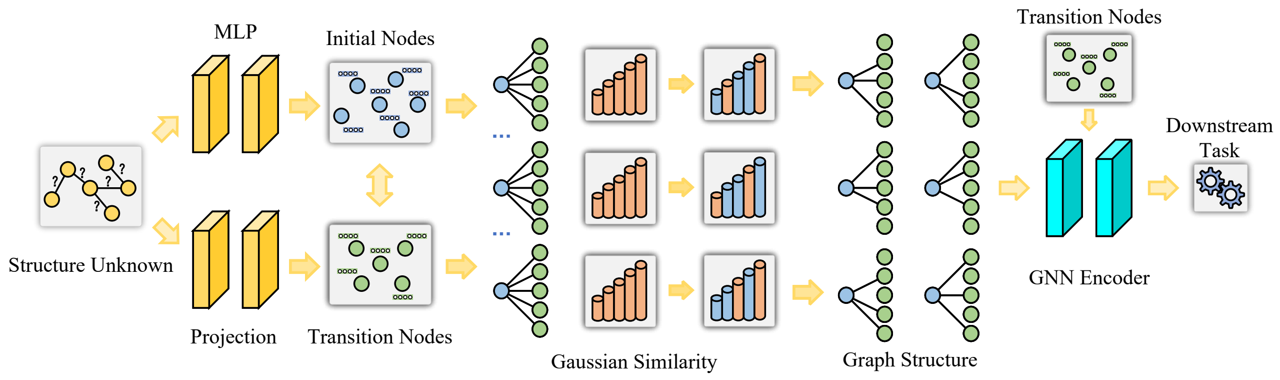


Figure 1: The proposed neural Gaussian similarity modeling and transition graph structure learning strategy.

distribution (Jang, Gu, and Poole 2017; Zheng et al. 2020; Sun et al. 2022) by replacing the non-differentiable sample from the Categorical distribution with a differentiable sample from the Gumbel-Softmax distribution, i.e.,

$$\mathrm{Cat}(\pi_{i,j}) \approx \frac{\exp((\log(\pi_{i,j}) + g_i)/\tau)}{\sum_{w=1}^{n} \exp((\log(\pi_{iw}) + g_w)/\tau)} \tag{4}$$

where $g_i = -\log(-\log(\epsilon))$ with ϵ randomly drawn from $\mathrm{Uniform}(0,1)$ and $\tau \in \mathbb{R}^+$ is the temperature which controls the interpolation between the discrete distribution and continuous categorical densities. After obtaining the graph structure A^*, we can use a GNN encoder $\mathrm{GNN}(A^*, X)$, e.g., GCN (Kipf and Welling 2017), to produce the final node embeddings for downstream tasks.

Analysis of Structure Sampling

Note that the sampling probability and similarity of the edge (v_i, v_j) exhibit a linear relationship, wherein the greater the similarity value, the higher the probability of sampling that edge. In this paper, we aim to provide a better understanding of this sampling strategy by asking the following question: *is this linear sampling strategy always necessary?* To answer this question, without loss of generality, we use GCN as GNN encoder and conduct the following theorem.

Theorem 1. *Suppose the number of structure sampling be $\mathcal{K}$ for each node, h_i be the feature embedding of node v_i, h_j be the sampled neighbors of node v_i where $j \in [1, \ldots, \mathcal{K}]$, and node features are normalized using 2-norm normalization. If $\|h_i - h_j\|_2 \leq \varepsilon$ for $\forall j \in [1, \ldots, \mathcal{K}]$ where ε is a non-negative constant, then*

$$\|\hat{h}_i - h_i\|_2 \leq \varepsilon \tag{5}$$

where $\hat{h}_i = \sum_{j=1}^{\mathcal{K}} \frac{1}{\sqrt{d_i d_j}} h_j$ is the predict probability distribution after graph convolutional operator.

Proof. Given $\|h_i - h_j\|_2 \leq \varepsilon$ for $\forall j \in [1, \ldots, \mathcal{K}]$ where ε is a non-negative constant, then we have

$$\begin{aligned} \|h_i - h_j\|_2^2 &\leq \varepsilon^2 \\ \|h_i\|_2^2 + \|h_j\|_2^2 - 2 \cdot \langle h_i, h_j \rangle &\leq \varepsilon^2 \\ \langle h_i, h_j \rangle &\geq \frac{\|h_i\|_2^2 + \|h_j\|_2^2 - \varepsilon^2}{2}. \end{aligned} \tag{6}$$

After graph convolutional operator, we can denote the 2-norm difference as

$$\|\hat{h}_i - h_i\|_2 = \sqrt{\|\hat{h}_i\|_2^2 + \|h_i\|_2^2 - 2 \cdot \langle h_i, \hat{h}_i \rangle}. \tag{7}$$

For $\langle \hat{h}_i, h_i \rangle$, we can get

$$\begin{aligned} \langle h_i, \hat{h}_i \rangle &= \langle h_i, \sum_{j=1}^{\mathcal{K}} \frac{1}{\sqrt{d_i d_j}} h_j \rangle \\ &= \frac{1}{\mathcal{K}} \sum_{j=1}^{\mathcal{K}} \langle h_i, h_j \rangle \geq \frac{1}{\mathcal{K}} \sum_{j=1}^{\mathcal{K}} \frac{2 - \varepsilon^2}{2}. \end{aligned} \tag{8}$$

Therefore, the 2-norm difference can be represented as

$$\begin{aligned} \|\hat{h}_i - h_i\|_2 &= \sqrt{\|\hat{h}_i\|_2^2 + \|h_i\|_2^2 - 2 \cdot \langle h_i, \hat{h}_i \rangle} \\ &\leq \sqrt{\|\hat{h}_i\|_2^2 + \|h_i\|_2^2 - \frac{2}{\mathcal{K}} \sum_{j=1}^{\mathcal{K}} \frac{2 - \varepsilon^2}{2}} \\ &\leq \sqrt{2 - (2 - \varepsilon^2)} = \varepsilon. \end{aligned} \tag{9}$$

□

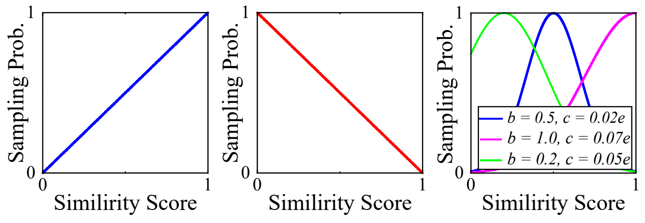
Theorem 1 tells us that the act of linear sampling may not invariably be essential, particularly in instances where node features exhibit a significant degree of similarity. In this case, the integration of the learned graph structure does not yield notable increase in informational gain.

Proposed Method

In this section, we present the proposed Neural Gaussian Similarity Modeling to alleviate the issue of structure sampling and the Transition Graph Structure Learning to reduce the complexity of computing edge similarity. The overall framework is shown in Figure 1.

Neural Gaussian Similarity Modeling

From the analysis of structure sampling, we find that it is not invariably essential to exclusively sample the edges exhibiting the highest probability distribution, i.e., the nodes



(a) LinSim (b) DiffSim (c) GauSim

Figure 2: Relationship between sampling probability and similarity score. Subfigure (a) denotes the Linear Sampling (LinSim) strategy, subfigure (b) denotes the Difference Similarity (DiffSim), and subfigure (c) denotes the Gaussian Similarity (GauSim) with different parameters.

displaying the utmost similarity. To alleviate the issue, a nature idea is obtaining the Difference Similarity (DiffSim) between pair of nodes in form of

$$\pi_{i,j} = \frac{\exp(z_i(z_i - z_j)^\top)}{\sum_{w=1}^{n} \exp(z_i(z_i - z_w)^\top)} \tag{10}$$

where $z_i(z_i - z_j)^\top = 1 - z_i z_j^\top$ when z_i, z_j are normalized using 2-norm normalization. As shown in Figure 2, we can observe that the sampling probability of edges increases as the similarity decreases, resulting in highly dissimilar node pairs being sampled as neighbors. This difference similarity modeling does not satisfy the assumption of the homophily effect in networks (Xie et al. 2020; Ma et al. 2022) and may harm the performance of GNN model.

To fulfill the requirement of the edge sampling probability, which entails an initial increase followed by a decrease as the similarity between node pairs diminishes, we propose a novel Gaussian Similarity (GauSim) modeling strategy in form of

$$\phi(z_i, z_j) = \exp\left(-\frac{(z_i z_j^\top - b)^2}{c}\right) \tag{11}$$

$$\pi_{i,j} = \frac{\exp(\phi(z_i, z_j))}{\sum_{w=1}^{n} \exp(\phi(z_i, z_w))} \tag{12}$$

where b, c are the parameters of Gaussian function. By choosing appropriate b and c, Gaussian similarity can be used as edge sampling probability to meet the nonlinear sampling requirement. Here, we set $b = 0.5$ and $c = 0.02e$, and the relationship between sampling probability and similarity is shown in the Figure 2.

Note that different node pairs may need to set different Gaussian function parameters, we further propose a Neural Gaussian Similarity (NeuralGauSim) modeling to adaptively learn the Gaussian function parameters. Specifically, we define a parameter learning network in form of

$$b_i = \sigma(z_i w_b^\top), c_i = \sigma(z_i w_c^\top) \tag{13}$$

$$\phi(z_i, z_j) = \exp\left(-\frac{(z_i z_j^\top - b_i)^2}{c_i}\right) \tag{14}$$

$$\pi_{i,j} = \frac{\exp(\phi(z_i, z_j))}{\sum_{w=1}^{n} \exp(\phi(z_i, z_w))} \tag{15}$$

where $w_b, w_c \in \mathbb{R}^m$ are the learnable parameters and σ is Sigmoid activation function. The relationship between sampling probability and similarity with different parameters b and c is shown in the Figure 2. From this figure, we can observe that the neural Gaussian similarity modeling can generalize the sampling strategy based on the previous linear and difference similarity by setting appropriate parameters.

Transition Graph Structure Learning

The edge similarity computes similarity scores for all pairs of graph nodes, which requires $\mathcal{O}(n^2)$ complexity for both computational time and memory consumption, rendering significant scalability issue for large graphs. To address the scalability issue, we develop a transition graph similarity modeling method by transferring the initial graph comprising n nodes to the more streamlined transition graph containing s nodes. Specifically, we first project the initial nodes $X \in \mathbb{R}^{n \times d}$ into the transition nodes in form of

$$R = W_t X \in \mathbb{R}^{s \times d} \tag{16}$$

where $W_t \in \mathbb{R}^{s \times n}$ is the projection matrix and $R = \{r_1, ..., r_s\}$ is transition node features. Then for each nodes of the initial graph, the similarity scores are calculated by

$$z_i = \text{MLP}(x_i) \tag{17}$$

$$\pi_{i,j} = \frac{\exp(\phi(z_i, r_j))}{\sum_{w=1}^{s} \exp(\phi(z_i, r_w))} \tag{18}$$

where $\phi(z_i, r_i)$ can use the Eqn. (11) and Eqn. (14) as the similarity measurement. Note that for node v_i, the calculation range of edge similarity score is the transition graph with s nodes instead of the original graph with n nodes, which requires $\mathcal{O}(ns)$ complexity for both computational time and memory consumption. The parameter s is often set as $s \ll n$ in practice, hence the complexity of computing similarity scores requires $\mathcal{O}(n)$ for both computational time and memory consumption. After obtaining the edge similarity scores, we can utilize the concrete relaxation of Categorical distribution to differentiably sample the edges by

$$A_t^* = \bigcup_{v_i \in \mathcal{V}, v_j \in \mathcal{V}_t} \{A_{i,j} \sim \text{Cat}(\pi_{i,j})\} \in \mathbb{R}^{n \times s} \tag{19}$$

where $\mathcal{V}_t$ denotes the node set of the transition graph. Finally, we can utilize the GCN encoder GNN$(\cdot)$ to produce the node embeddings in form of

$$X_t = W_e X \in \mathbb{R}^{s \times d} \tag{20}$$

$$H = \text{GNN}(A_t^*, X_t) \in \mathbb{R}^{n \times m} \tag{21}$$

where $W_e \in \mathbb{R}^{s \times n}$ is the learnable projection matrix and H is the final node embedding matrix. Different from the previous anchor-based graph structure leaning method (Chen, Wu, and Zaki 2020), which randomly samples a set of anchors from the initial graph, the developed transition graph method uses projection matrices to learn the node transition strategy, thus mitigating the information loss resulting from random sampling.

METHOD	CiteSeer	PubMed	Chameleon	Squirrel	CS	Physics	20News	Mini
GCN$_{knn}$	66.67±1.33	78.84±0.35	41.91±1.13	26.80±0.82	87.57±0.11	91.84±0.21	61.36±0.13	80.69±0.48
HeatGCN$_{knn}$	65.07±0.52	75.20±0.28	43.23±0.20	27.21±0.55	88.21±0.13	91.96±0.13	50.17±0.61	78.14±0.17
LINKX$_{knn}$	57.80±1.70	75.72±1.13	36.32±2.47	24.70±2.42	80.38±2.62	81.28±2.24	52.24±3.14	71.19±2.50
SLAPS	68.23±0.22	79.55±0.23	43.44±0.36	27.28±0.96	85.74±2.12	92.14±2.13	60.97±0.20	80.23±1.14
IDGL-Full	68.67±0.32	OOM	45.47±0.76	28.23±0.19	OOM	OOM	62.53±0.24	OOM
IDGL-Anchor	67.31±0.07	79.69±0.43	45.35±1.23	28.08±1.17	89.15±0.81	OOM	61.91±0.26	80.51±0.61
NodeFormer	67.19±0.28	82.57±0.45	47.02±0.61	27.80±0.75	**91.77±0.10**	**94.81±0.11**	62.70±0.84	83.10±0.49
LinSim	67.23±0.23	OOM	44.91±1.10	27.26±0.95	OOM	OOM	54.40±1.07	OOM
LinSim-T	69.51±0.60	81.41±1.31	43.63±1.38	27.75±0.54	87.67±1.55	93.88±0.19	60.84±0.26	81.94±0.14
GauSim	69.19±0.14	OOM	47.84±0.54	28.13±0.87	OOM	OOM	62.49±0.20	OOM
GauSim-T	**70.19±0.84**	82.62±0.51	47.19±1.16	27.80±0.67	88.94±0.47	94.13±0.32	61.35±0.56	83.33±0.88
NeuralGauSim	69.31±0.35	OOM	48.02±0.20	**28.88±1.08**	OOM	OOM	62.85±0.17	OOM
NeuralGauSim-T	70.15±0.37	**82.65±0.74**	**48.25±0.63**	28.57±0.36	89.22±1.55	94.64±0.10	**62.89±1.10**	**84.89±0.37**

Table 1: Experimental results comparison with baselines. Here, OOM denotes Out-of-Memory.

	METHOD	CiteSeer	Chameleon	Squirrel	20News
$\mathcal{K}$=5	LinSim	67.23±0.23	44.91±1.10	27.26±0.95	54.40±1.07
	GauSim	69.19±0.14	47.84±0.54	28.13±0.87	62.49±0.20
	NeuralGauSim	**69.31±0.35**	**48.02±0.20**	**28.88±1.08**	**62.85±0.17**
$\mathcal{K}$=10	LinSim	66.10±0.82	43.63±0.82	26.59±0.87	
	GauSim	68.31±0.52	46.73±0.54	27.29±0.67	OOM
	NeuralGauSim	**68.35±0.98**	**46.84±0.46**	**27.47±0.62**	
$\mathcal{K}$=15	LinSim	65.55±0.82	42.79±1.95	25.34±1.14	
	GauSim	69.03±0.21	47.54±0.70	28.03±1.05	OOM
	NeuralGauSim	**69.55±0.96**	**47.49±1.67**	**28.49±0.35**	
$\mathcal{K}$=20	LinSim	64.39±0.76	41.73±1.09	25.13±1.14	
	GauSim	68.47±0.97	47.66±0.44	26.77±1.19	OOM
	NeuralGauSim	**68.69±0.52**	**48.84±0.98**	**27.95±0.77**	

Table 2: Experimental results comparison with different samples $\mathcal{K}$ for full-graph models. Here, OOM denotes Out-of-Memory.

Experiment

Datasets

To evaluate the developed methods, we use eight commonly used graph datasets, including two citation network datasets, i.e., CiteSeer and PubMed (Yang, Cohen, and Salakhudinov 2016), two Wikipedia network datasets, i.e., Chameleon and squirrel (Pei et al. 2020), two coauthor network datasets, i.e., CS and Physics (Shchur et al. 2018), and two graph-enhanced application datasets, i.e., 20News and Mini-ImageNet (Mini) (Wu et al. 2022).

Baselines

We compare the proposed method with prominent existing graph structure learning baselines, including GCN$_{knn}$ (Kipf and Welling 2017), HeatGCN$_{knn}$ where the heat kernel (Wang et al. 2020) serves as similarity measurement, LINKX$_{knn}$ (Lim et al. 2021), SLAPS (Fatemi, El Asri, and Kazemi 2021), IDGL-Full, IDGL-Anchor (Chen, Wu, and Zaki 2020), and NodeFormer (Wu et al. 2022). For kNN based methods, we create a kNN graph based on the node feature similarities and feed this graph to the GNN model.

Configurations

The experiments are repeated five times for all algorithms to finally report the average accuracy and the corresponding standard deviation. For all datasets, we randomly sampled 50% of nodes for training set, 25% for validation set, and 25% for test set. The Adam optimizer are used with the learning rate 0.01 for CiteSeer dataset and 0.001 for other datasets. We set the hidden dimension to be 256 for Mini-ImageNet dataset, 32 for other datasets, and the number of transition-graph nodes to 500. The weight parameters are initialized using Glorot initialization and the bias parameters using zero initialization. For parameters c of NeuralGauSim, we multiply c by a scale factor of 0.1. We add dropout layers with probabilities of 0.5 after the first layer of the GNNs, and train two-layer GNNs.

Results Comparison with Baselines

To verify the performance of the developed models in comparison to baselines, we conduct comparative experiments and report results in Table 1. In this experiment, we set the parameter K to be 5 for kNN based baselines, and the neighbor mask parameter to be 0 for IDGL. From this table, we have the following observations. First,

	Method	CiteSeer	PubMed	Chameleon	Squirrel	CS	Physics	20News	Mini
$\mathcal{K}$=5	LinSim-T	69.51±0.60	81.41±1.31	43.63±1.38	27.75±0.54	87.67±1.55	93.88±0.19	60.84±0.26	81.94±0.14
	GauSim-T	**70.19±0.84**	82.62±0.51	47.19±1.16	27.80±0.67	88.94±0.47	94.14±0.32	61.35±0.56	83.33±0.88
	NeuralGauSim-T	70.15±0.37	**82.65±0.74**	**48.25±0.63**	**28.57±0.36**	**89.22±1.55**	**94.64±0.10**	**62.89±1.10**	**84.89±0.37**
$\mathcal{K}$=10	LinSim-T	64.23±0.70	79.64±1.76	42.40±1.06	25.18±1.50	83.49±1.18	92.30±1.36	59.60±0.77	80.65±0.98
	GauSim-T	69.03±0.68	80.42±1.19	47.49±1.07	27.08±1.12	88.63±0.69	**94.12±0.76**	**60.36±0.12**	82.17±1.24
	NeuralGauSim-T	**69.19±0.76**	**80.69±1.62**	**47.66±0.20**	**27.57±0.35**	**89.79±1.17**	94.06±0.60	60.06±0.57	**83.53±0.53**
$\mathcal{K}$=15	LinSim-T	64.75±0.92	76.46±1.12	40.94±1.16	24.42±1.98	79.48±1.54	88.02±1.86	58.87±0.61	80.55±0.74
	GauSim-T	70.03±0.57	81.61±0.43	47.60±0.44	28.16±0.38	89.51±0.34	92.17±1.87	60.19±1.07	82.64±0.71
	NeuralGauSim-T	**70.39±0.80**	**81.70±0.55**	**47.31±0.51**	**28.49±0.42**	**90.49±0.49**	**92.89±1.67**	**60.36±0.97**	**82.87±0.97**
$\mathcal{K}$=20	LinSim-T	64.31±0.53	75.03±1.57	41.35±1.62	24.62±1.13	71.68±2.92	87.91±1.39	57.54±1.11	79.70±1.14
	GauSim-T	**68.99±0.92**	80.73±1.35	47.99±0.88	28.29±0.35	87.87±1.17	92.30±0.76	59.04±0.44	80.24±0.70
	NeuralGauSim-T	68.59±0.98	**81.50±0.87**	**48.04±0.46**	**28.54±0.38**	**88.54±1.12**	**93.14±0.59**	**59.59±0.76**	**81.64±0.97**

Table 3: Experimental results comparison with different samples $\mathcal{K}$ for transition-graph models.

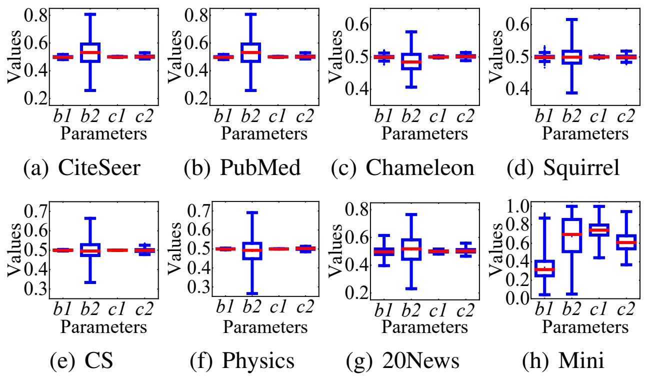


Figure 3: Analysis of Gaussian parameter distributions of Neural Gaussian Similarity modeling for all datasets.

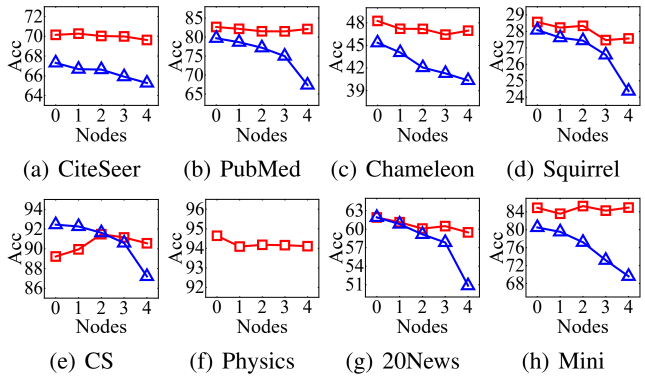


Figure 4: Analysis of removing transition-graph and anchor-graph nodes ($\times 10^2$) for NeuralGauSim-T (red line) and IDGL-Anchor (blue line), respectively.

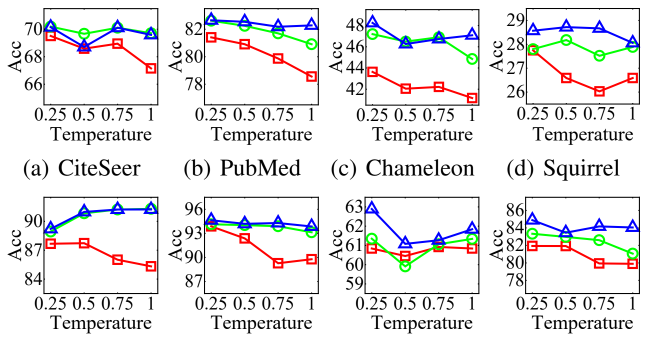
compared with the previous graph structure learning baselines, the proposed Gaussian similarity modeling methods achieve better performance on six datasets out of eight, indicating the effectiveness of the differential graph structure learning. Second, compared with the linear similarity, the proposed Gaussian similarity modeling methods consistently achieve better performance, demonstrating the effectiveness of the proposed methods. Third, comparing with transition-graph based models, i.e., LinSim-T, GauSim-T, and NeuralGauSim-T, and full-graph models, i.e. LinSim, GauSim, and NeuralGauSim, we can observe that transition-graph based models achieves performance improvements while reducing complexity. This implies the effectiveness of introducing the learnable transition-graph strategy.

Results Comparison of Different Edge Samples

To evaluate the performance of different number of sampling edges $\mathcal{K}$, we perform comparative experiments by setting $\mathcal{K} \in \{5, 10, 15, 20\}$ for full-graph and transition-graph settings. Experimental results are reported in Table 2 and Table 3, respectively. From these two tables, we have the following observations. First, compared with the Linear Similarity (LinSim), the proposed GauSim and NeuralGauSim methods consistently achieve better performance under different number of edge samples, indicating the effectiveness of the proposed Gaussian similarity modeling. Second, by comparing GauSim/GauSim-T and NeuralGauSim/NeuralGauSim-T, we can find that the NeuralGauSim method exhibits superior performance except when $\mathcal{K} = 10$ on 20News dataset, demonstrating the effectiveness of the learnable parameters of the Gaussian similarity. Third, by comparing performance of different number of edge samples, we can observe that as the parameter $\mathcal{K}$ increases, the performance of LinSim deteriorates. In contrast, the proposed GauSim and NeuralGauSim methods consistently maintain its performance level, further substantiating the effectiveness of the proposed methods.

Analysis of Gaussian Parameter Distributions

To investigate whether the parameter values of Gaussian similarity learned by the proposed similarity modeling method are meaningful, we conduct the Gaussian parameter distribution analysis experiments on all datasets. Experimental results are shown in Figure 3. Specifically, we use the boxplot to count the distribution of parameters b and c of



(a) CiteSeer (b) PubMed (c) Chameleon (d) Squirrel

(e) CS (f) Physics (g) 20News (h) Mini

Figure 5: Impact of Temperature Parameter τ. The red line denotes LinSim, the green line denotes GauSim, and the blue line denotes Gausim-T.

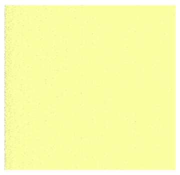
	Training Time (s)	GPU Memory (MB)
IDGL	0.7967	4,358
IDGL-Anchor	0.1249	2,986
NodeFormer	**0.0463**	**1,248**
LinSim	0.1145	4,486
LinSim-T	**0.0367**	**1,796**
GauSim	0.1193	4,618
GauSim-T	**0.0382**	**1,816**
NeuralGauSim	0.1238	4,664
NeuralGauSim-T	**0.0393**	**1,821**

Table 4: Comparison of training time (s) and GPU memory (MB) cost on CiteSeer dataset per epoch.

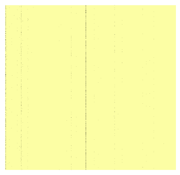
the two-layer model, where the parameters of the first layer are denoted as b_1, c_1, and the second layer are denoted as b_2, c_2. For clarity, we multiply all parameters c by a scale factor of 10. From these figures, we have the following observations. First, the adjustment range of the parameters of the second layer is larger than that of the first layer. Second, for all datasets, the parameter b_2 exhibits a wider distribution compared to other parameters.

Sensitivity of Transition Graph Nodes

To study the effect of different number of transition-graph nodes, we conduct a comparative experiment to observe the change of model performance by removing the number of transition-graph nodes for the proposed NeuralGauSim-T. Here, for the initial transition graph with 500 nodes, we remove $\{0, 100, 200, 300, 400\}$ nodes respectively. Experimental results are shown in Figure 4. Note that for Physics dataset, we only report the results of NeuralGauSim-T since the previous IDGL-Anchor method exceeds the memory limit due to the initial fully kNN graph requires $\mathcal{O}(n^2)$ memory consumption. From these figures, we can observe that for NeuralGauSim-T, removing nodes does not bring about a significant decrease in model performance, and even an improvement in performance. Conversely, for the IDGL-Anchor method, removing nodes brings significant performance degradation, especially when the number of removed



(a) Layer 1



(b) Layer 2

Figure 6: Visualization of Latent Graph Structure (given by two layers of NeuralGauSim) on CiteSeer dataset.

nodes tends to be large. The primary justification lies in the fact that the anchor graph is generated through random sampling from the original graph, whereas the construction of the transition-graph relies on model learning.

Impact of Temperature Parameter

To evaluate the impact of different temperature parameter τ of Gumbel-Softmax distribution, we conduct a comparative experiment by setting different temperature parameter $\tau \in \{0.25, 0.5, 0.75, 1\}$ on all datasets. Experimental results are shown in Figure 5. This figure illustrates the direct impact of modifying the temperature parameter on the performance of the proposed model. Different datasets may require different temperature parameter setting strategies. In this paper, following NodeFormer (Wu et al. 2022), we set the same temperature parameters $\tau = 0.25$ for all datasets for comparison.

Comparison of Complexity and Visualization of Learned Graph Structure

To compare the training time and GPU memory consumption, we statistic the training time (s) and GPU memory usage (MB) of the proposed method and previous methods, including IDGL, IDGL-Anchor, and NodeFormer, on CiteSeer dataset. Here, the number of transition nodes is set to be 500. The experimental results are shown in Table 4. we can observe that the proposed transition graph structure learning method significantly reduces the complexity of both time and memory consumption. To investigate the learned graph structure of the proposed method, we visualize the graph structures learned by NeuralGauSim (given by two layers) on CiteSeer dataset, shown in Figure 6.

Conclusion

In this paper, we focus on the differential graph structure learning framework and analyze the issue of structure sampling strategy. To fulfill the requirement of the edge sampling, we propose the Gaussian similarity modeling method and neural Gaussian similarity modeling method. To reduce the complexity, we develop a transition graph structure learning by transferring the initial nodes to transition nodes. Extensive experiments on graph and graph-enhanced application datasets demonstrate the effectiveness of the proposed methods. In future work, we aim to delve into effectiveness methods to tackle increasingly demanding scenarios, including heterogeneous graphs and multiplex graphs.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62206208, 62106185 and in part by the Fundamental Research Funds for the Central Universities under Grant XJSJ23021.

References

Chen, Y.; Wu, L.; and Zaki, M. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. *Proceedings of the Advances in Neural Information Processing Systems*, 33: 19314–19326.

Fan, X.; Gong, M.; Tang, Z.; and Wu, Y. 2022. Deep Neural Message Passing With Hierarchical Layer Aggregation and Neighbor Normalization. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12): 7172–7184.

Fan, X.; Gong, M.; Wu, Y.; and Li, H. 2023a. Maximizing Mutual Information Across Feature and Topology Views for Representing Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 1–14.

Fan, X.; Gong, M.; Wu, Y.; Qin, A. K.; and Xie, Y. 2023b. Propagation Enhanced Neural Message Passing for Graph Representation Learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(2): 1952–1964.

Fatemi, B.; El Asri, L.; and Kazemi, S. M. 2021. SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks. *Proceedings of the Advances in Neural Information Processing Systems*, 34: 22667–22681.

Franceschi, L.; Niepert, M.; Pontil, M.; and He, X. 2019. Learning Discrete Structures for Graph Neural Networks. In *Proceedings of the International Conference on Machine Learning*, 1972–1982.

Gao, C.; Zheng, Y.; Li, N.; Li, Y.; Qin, Y.; Piao, J.; Quan, Y.; Chang, J.; Jin, D.; He, X.; et al. 2023. A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions. *ACM Transactions on Recommender Systems*, 1(1): 1–51.

Gasteiger, J.; Weißenberger, S.; and Günnemann, S. 2019. Diffusion Improves Graph Learning. *Proceedings of the Advances in Neural Information Processing Systems*, 32.

Han, K.; Wang, Y.; Guo, J.; Tang, Y.; and Wu, E. 2022. Vision GNN: An Image is Worth Graph of Nodes. *Proceedings of the Advances in Neural Information Processing Systems*, 35: 8291–8303.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the International Conference on Learning Representations*.

Jin, W.; Ma, Y.; Liu, X.; Tang, X.; Wang, S.; and Tang, J. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining*, 66–74.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the International Conference on Learning Representations*.

Lim, D.; Hohne, F.; Li, X.; Huang, S. L.; Gupta, V.; Bhalerao, O.; and Lim, S. N. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. *Proceedings of the Advances in Neural Information Processing Systems*, 34: 20887–20902.

Ma, Y.; Liu, X.; Shah, N.; and Tang, J. 2022. Is Homophily a Necessity for Graph Neural Networks? In *Proceedings of the International Conference on Learning Representations*.

Meng, Y.; Zong, S.; Li, X.; Sun, X.; Zhang, T.; Wu, F.; and Li, J. 2022. GNN-LM: Language Modeling based on Global Contexts via GNN. In *Proceedings of the International Conference on Learning Representations*.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *Proceedings of the International Conference on Learning Representations*.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of Graph Neural Network Evaluation. *arXiv preprint arXiv:1811.05868*.

Sun, Q.; Li, J.; Peng, H.; Wu, J.; Fu, X.; Ji, C.; and Philip, S. Y. 2022. Graph Structure Learning with Variational Information Bottleneck. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 4165–4174.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *Proceedings of the International Conference on Learning Representations*.

Wang, X.; Zhu, M.; Bo, D.; Cui, P.; Shi, C.; and Pei, J. 2020. AM-GCN: Adaptive Multi-channel Graph Convolutional Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1243–1253.

Wu, Q.; Zhao, W.; Li, Z.; Wipf, D. P.; and Yan, J. 2022. Nodeformer: A Scalable Graph Structure Learning Transformer for Node Classification. *Proceedings of the Advances in Neural Information Processing Systems*, 35: 27387–27401.

Xie, Y.; Li, S.; Yang, C.; Wong, R. C.-W.; and Han, J. 2020. When Do GNNs Work: Understanding and Improving Neighborhood Aggregation. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Yang, Z.; Cohen, W.; and Salakhudinov, R. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of the International Conference on Machine Learning*, 40–48.

Yu, D.; Zhang, R.; Jiang, Z.; Wu, Y.; and Yang, Y. 2021. Graph-Revised Convolutional Network. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference*, 378–393.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *Proceedings of the International Conference on Learning Representations*.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O.; Jiang, M.; and Shah, N. 2021. Data Augmentation for Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11015–11023.

Zheng, C.; Zong, B.; Cheng, W.; Song, D.; Ni, J.; Yu, W.; Chen, H.; and Wang, W. 2020. Robust Graph Representation Learning via Neural Sparsification. In *Proceedings of the International Conference on Machine Learning*, 11458–11468.