Front. Comput. Sci., 2025, 19(1): 191305
https://doi.org/10.1007/s11704-023-3339-7

RESEARCH ARTICLE

D^2-GCN: a graph convolutional network with dynamic disentanglement for node classification

Shangwei WU, Yingtong XIONG, Hui LIANG, Chuliang WENG (✉)

School of Data Science and Engineering, East China Normal University, Shanghai 200062, China

Abstract Classic Graph Convolutional Networks (GCNs) often learn node representation holistically, which ignores the distinct impacts from different neighbors when aggregating their features to update a node's representation. Disentangled GCNs have been proposed to divide each node's representation into several feature units. However, current disentangling methods do not try to figure out how many inherent factors the model should assign to help extract the best representation of each node. This paper then proposes D^2-GCN to provide dynamic disentanglement in GCNs and present the most appropriate factorization of each node's mixed features. The convergence of the proposed method is proved both theoretically and experimentally. Experiments on real-world datasets show that D^2-GCN outperforms the baseline models concerning node classification results in both single- and multi-label tasks.

Keywords graph convolutional networks, dynamic disentanglement, label entropy, node classification

1 Introduction

Convolutional Neural Networks (CNNs) have achieved great success in regular image data, but there exist many irregular data (i.e., graphs) in real-world scenarios, such as citation networks, social networks, and biochemical networks. Inspired by CNNs, Graph Convolutional Networks (GCNs) migrate the convolution operation to graphs and are widely adopted in node classification [1,2], link prediction [3,4], graph classification [5–7], drug discovery [8–10], and recommender systems [11–13].

Although GCNs have excellent feature extraction ability on graph data, standard GCNs usually study the representation of each node in a holistic way, i.e., they consider each node's neighborhood as a whole and neglect the decomposable correlations within the neighborhood. In fact, in many cases, the links in a graph are determined by various potential factors that constitute the attributes of a node. For example, in a citation network, there are only citation relationships between a paper (denoted by A) and other papers, but all the papers linked to it do not necessarily fall into the same category since the connections between A and its neighbors may happen due to the intersection of different domain knowledge, such as genetic algorithms, neural networks, or reinforcement learning. Therefore, to obtain a precise representation of A, it is essential to describe its characteristics from various perspectives. This could be achieved by factorizing A's embedded attributes (i.e., keywords reflecting multiple domain knowledge) into distinct groups.

In the image processing field, research efforts [14,15] have been devoted to disentangling the connections in image data. It was showed in [16] that such operations could improve the representation of images. Besides, the factorization of features in images would also enhance the interpretability of CNNs [17]. However, unlike grid image data, the irregular relations in non-grid graph data make it difficult to borrow the disentangling ideas in CNNs. DisenGCN [18] was a pioneering work that disentangled the separable linkages in the neighborhood of each node. It proposed a neighborhood routing algorithm to divide each node's neighborhood into K channels that corresponded to K underlying factors. In this way, the features of a node u and all its neighbors would be split into K units. Then the feature components from K aspects were concatenated to form the final representation of u. IPGDN [19] further promoted the mutual independence of the K channels in DisenGCN.

Intuitively, the value of K could reflect the quality of feature disentanglement, which directly affects the classification performance in both training and inference. However, both DisenGCN and IPGDN determined K empirically and tended to fix the value of K during training. To evaluate the impact of K, we conduct several independent training processes with different K to observe the changes in both accuracy and loss on a citation network dataset (i.e., Cora [20]). It is shown in Fig. 1 that the performance of DisenGCN varies when deploying different K. On the one hand, if K is too small (i.e., $K = 1$ and $K = 3$), the information extraction ability of the model would be inadequate, which is reflected by lower training or test accuracies (see Figs. 1(a) and 1(b)) compared with the situations when $K = 5$, 7, and 9. Besides, it is observed in Figs. 1(c) and 1(d) that the convergence speed of the model with a small K would be slower than that with a

Received April 26, 2023; accepted November 29, 2023

E-mail: clweng@dase.ecnu.edu.cn

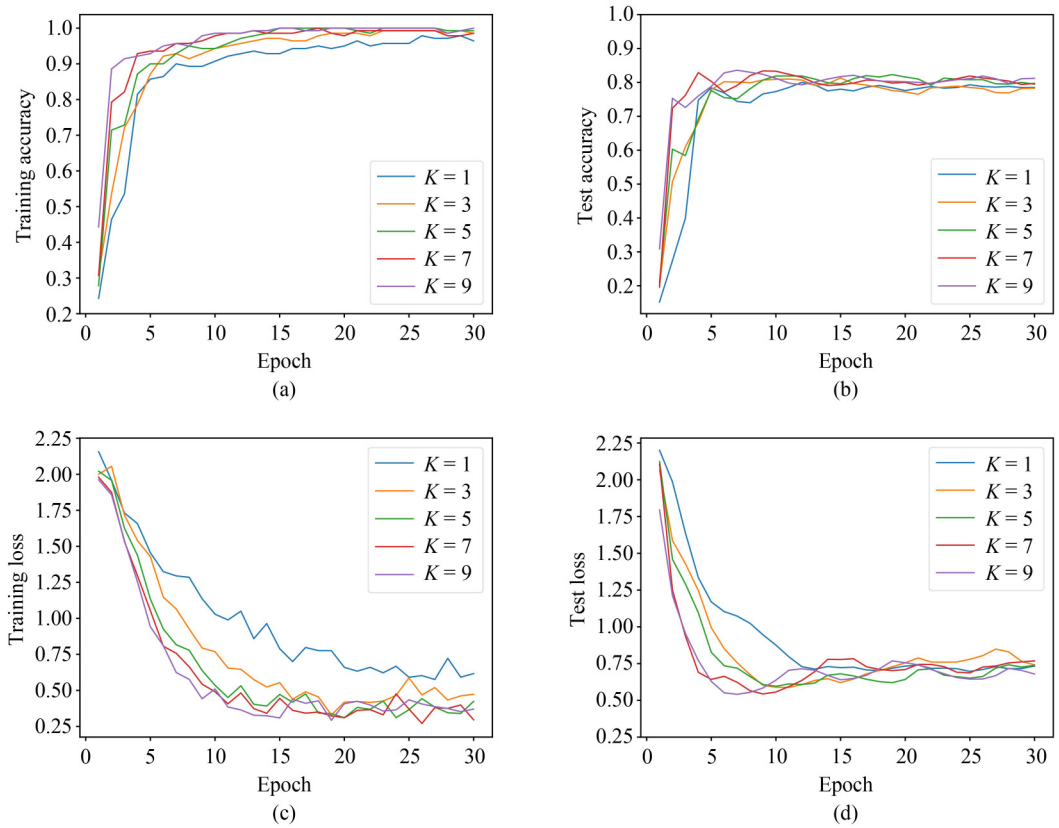


Fig. 1 Effects of different K in DisenGCN on Cora. In each subgraph, the x-axis represents the number of training epochs, the y-axis represents training/test accuracies or training/test losses. (a) Training accuracies; (b) test accuracies; (c) training losses; (d) test losses

larger K. On the other hand, if K is large (i.e., $K = 9$), the model performs well on the training set w.r.t. accuracy results but shows limited increases on the test set. Therefore, it is of great necessity to choose an appropriate K to best depict the representations of all the nodes in a graph, which also has to change with datasets.

Although we could select a proper K by grid search, it is still ignored that the classes of each node's neighbors would change over training. Next we would take the citation network again as an example. In the early training stage, the neighborhood of an article A might be divided into two sub-neighborhoods from two different domains, e.g., deep learning (DL) techniques and reinforcement learning (RL) algorithms. As the training proceeds, the partitioning of A's neighborhood needs to be reconsidered since the sub-neighborhood w.r.t. DL techniques derived in the previous training iteration might be decomposed into two smaller sub-neighborhoods, i.e., DL techniques in computer vision (CV) and DL techniques in natural language processing (NLP). To this end, the number of A's feature channels increases from two to three. Thus a fixed K would not suffice to precisely capture node representations that are dynamically changing throughout the whole training process.

In this paper, instead of trying each possible K independently, we adjust K every few epochs during training through monitoring the change of predicted types of each node's neighbors on the validation set. Specifically, we initialize K to 1 and then continuously observe the weighted number of different nodes' current neighbor classes to find the best K on each dataset. To sum up, four main contributions are made in this paper:

- We present a novel disentangled graph convolutional network named D^2-GCN to dynamically adjust the number of disentangled feature channels for each node during training, which helps to obtain adaptive node representations on datasets in various fields.
- We design a two-level disentangling mechanism in D^2-GCN that integrates epoch-level and layer-level disentanglement during training. With this mechanism, D^2-GCN could capture nuanced changes in node representations on graphs of varying sizes or complexities.
- We leverage the expectation-maximization (EM) algorithm to demonstrate the convergence of the dynamic disentanglement in D^2-GCN. We further propose an entropy-based evaluation metric to portray the convergence speed of disentanglement.
- Experiments show that our model outperforms the baselines in both single- and multi-label tasks in terms of test accuracies. Visualization results also imply that D^2-GCN displays clearer classification boundaries and higher intra-class similarity than DisenGCN.

The rest of the paper is organized as follows. We introduce the prerequisite knowledge of this paper in Section 2. Next, we describe the basic design of D^2-GCN and the implementation of the proposed dynamic disentanglement in Section 3. The convergence of the automatically optimized

disentanglement is proved in Section 4. In Section 5, we conduct comprehensive experiments to compare the performance of our work with the baseline models. We discuss related work in Section 6 and conclude this paper in Section 7.

2 Preliminaries

We briefly review the structures of classic Graph Convolutional Networks (GCNs) and DisenGCN in this section.

2.1 Graph convolutional networks

Let $G=(V,E)$ be a graph, in which $V=\{v_1,v_2,...,v_N\}$ represents the set of all nodes in the graph (N is the total number of nodes in G) and E is the set of all existing edges in G. If two nodes $u,v\in V$ are connected to each other, we say $(u,v)\in E$.

GCNs have gained promising results in the graph domain, in which the information of each node is learned by aggregating the features from its neighbors and itself. The representation of a node in GCNs can be written as

$$\mathbf{z}_u^l=\sigma(f(\mathbf{z}_u^{l-1},\{\mathbf{z}_v^l:(u,v)\in E\})), \tag{1}$$

where $\mathbf{z}_u^l\in\mathbb{R}^{d^l\times 1}$ represents u's node feature vector at convolutional layer l (d^l implies the number of feature dimensions), $f(\cdot)$ is the aggregation function in each convolutional layer, and $\sigma(\cdot)$ denotes a nonlinear activation function.

2.2 Disentangled representation on graphs

In recent years, disentanglement has become a significant attempt in interpreting DL models. For example, in the field of image processing, many efforts [14,15,17] have been paid to disentangle the semantics in an object into several disjoint factors, e.g., size, shape, color, viewing angle, etc. By separating the inherent factors in the representation space and encoding them into independent and distinct latent variables, it would bring a better understanding of the model's learning behavior and the object itself. DisenGCN [18] was a heuristic work that brought the idea of feature disentanglement to GCNs and proposed a novel neighborhood routing mechanism to identify the potential factors that might cause the connections between a node and its neighbors.

At the lth convolutional layer, DisenGCN splits the input feature vectors (derived from the $(l-1)$th layer) of node u and all its neighbors $\{v:(u,v)\in E\}$ into K channels. Then the representations of u and v (denoted by $\mathbf{z}_u^{l-1}$ and $\mathbf{z}_v^{l-1}$, respectively) would be both projected onto K feature subspaces, i.e.,

$$\mathbf{z}_{u,k}^l=\frac{\sigma((\mathbf{W}_{u,k}^{l-1})^{\mathrm{T}}\cdot\mathbf{z}_u^{l-1}+\mathbf{b}_{u,k}^{l-1})}{\left\|\sigma((\mathbf{W}_{u,k}^{l-1})^{\mathrm{T}}\cdot\mathbf{z}_u^{l-1}+\mathbf{b}_{u,k}^{l-1})\right\|_2},\ k=1,2,...,K, \tag{2}$$

and

$$\mathbf{z}_{v,k}^l=\frac{\sigma((\mathbf{W}_{v,k}^{l-1})^{\mathrm{T}}\cdot\mathbf{z}_v^{l-1}+\mathbf{b}_{v,k}^{l-1})}{\left\|\sigma((\mathbf{W}_{v,k}^{l-1})^{\mathrm{T}}\cdot\mathbf{z}_v^{l-1}+\mathbf{b}_{v,k}^{l-1})\right\|_2},\ k=1,2,...,K, \tag{3}$$

where $\mathbf{z}_{u,k}^l\in\mathbb{R}^{d_{in}\times 1}$ (or $\mathbf{z}_{v,k}^l\in\mathbb{R}^{d_{in}\times 1}$) indicates u's (or v's) initialized representation on the kth disentangled subspace before the training at the lth layer, $\mathbf{W}_{u,k}^{l-1},\mathbf{W}_{v,k}^{l-1}\in\mathbb{R}^{d_{in}\times\frac{d_{out}}{K}}$ and $\mathbf{b}_{u,k}^{l-1},\mathbf{b}_{v,k}^{l-1}\in\mathbb{R}^{\frac{d_{out}}{K}\times 1}$ are the kth partitions of the learnable 2D and 1D weight parameters obtained from layer $l-1$.

According to the neighborhood routing mechanism (NRM) in DisenGCN, $p_{v,k}$ is defined as the probability that node u is connected to v due to factor k ($p_{v,k}\geqslant 0$ and $\sum_{k=1}^K p_{v,k}=1$). At the sth routing iteration ($s=1,...,S$), DisenGCN updates the *feature center* of u's kth disentangled feature subspace cluster (denoted by $\mathbf{c}_{u,k}^{(s)}$) and $p_{v,k}$ by

$$\mathbf{c}_{u,k}^{(s)}=\frac{\mathbf{z}_{u,k}^l+\sum_{v:(u,v)\in E}p_{v,k}^{(s-1)}\mathbf{z}_{v,k}^l}{\left\|\mathbf{z}_{u,k}^l+\sum_{v:(u,v)\in E}p_{v,k}^{(s-1)}\mathbf{z}_{v,k}^l\right\|_2}, \tag{4}$$

and

$$p_{v,k}^{(s)}=\frac{\exp((\mathbf{z}_{v,k}^l)^{\mathrm{T}}\mathbf{c}_{u,k}^s)}{\sum_{k=1}^K\exp((\mathbf{z}_{v,k}^l)^{\mathrm{T}}\mathbf{c}_{u,k}^s)}, \tag{5}$$

in which $p_{v,k}^{(s)}$ represents the updated connection probability from v to u at the sth routing iteration. Note that DisenGCN starts NRM by initializing $p_{v,k}\propto\exp(\mathbf{z}_{v,k}^{\mathrm{T}}\mathbf{z}_{u,k})$ at the first routing iteration. Finally, at the Sth routing iteration, K feature centers from K subspace clusters would be concatenated to form the updated representation of node u at layer l (denoted by $\mathbf{z}_u^l$), i.e.,

$$\mathbf{z}_u^l=[\mathbf{c}_{u,1}^{(S)},\mathbf{c}_{u,2}^{(S)},...,\mathbf{c}_{u,K}^{(S)}],\ \mathbf{z}_u^l\in\mathbb{R}^{d_{out}}. \tag{6}$$

3 Methodology

In this section, we elaborate on the design and implementation of the proposed method.

3.1 Motivations for updating K

To examine whether and how the number of disentangled factors that cause the various connections in a node's neighborhood changes over training, we conduct a test to observe the changes in the kinds of predicted labels of each node's neighbors across training epochs, which is shown in Fig. 2. Here we select the situations corresponding to three typical kinds of nodes to be depicted in the figure. Figure 2(a) describes the type of node (denoted by v_α) with stable number of predicted neighbor classes from the very beginning, Fig. 2(b) describes another type of node (denoted by v_β) whose number of predicted neighbor classes would fluctuate up and down around a fixed value over training, while Fig. 2(c) describes the type of node (denoted by v_γ) whose number of neighbor classes would converge in the end but with different convergence speeds w.r.t. different K.

The phenomenon depicted in Fig. 2 indicates that the predicted label classes of one node's neighbors indeed vary in different ways during training, which motivates us to design a mechanism for dynamically adjusting the value of K to keep in pace with the convergence of the disentanglement. Next, the main focus is on how to obtain an appropriate K value at different stages of training. Suppose V_{val} denotes all the nodes on the validation set and n_{v_i} ($v_i\in V_{val}$, $i\in[1,N_{val}]$) represents

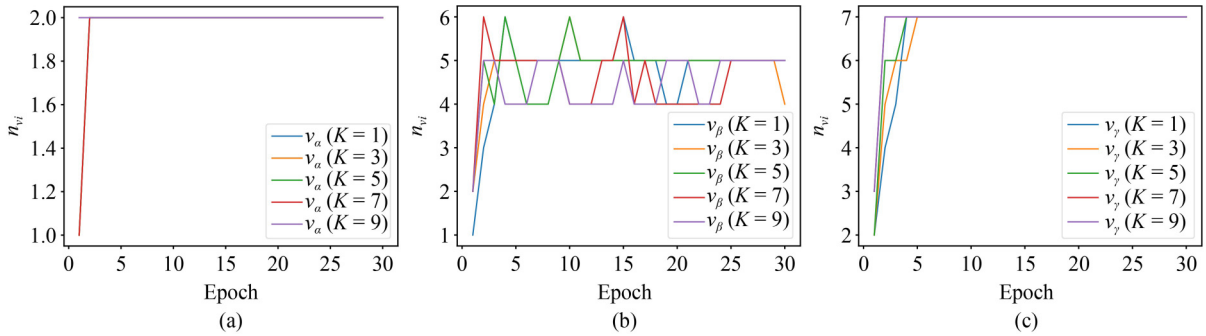


Fig. 2 Changes in the kinds of predicted labels of three typical nodes' neighbors across training epochs with different K in DisenGCN on the Cora dataset. (a) The test with v_α; (b) the test with v_β; (c) the test with v_γ

the number of predicted classes of node v_i's neighbors where N_{val} is the number of nodes in V_{val}, then we present three intuitive strategies below that would take into account all the nodes in V_{val} and help to find the most appropriate and fairest K during training:

- Calculate the mode value in $\{n_{v_i}\}$, denoted by $\hat{K}_{val}$ (i.e., the value that appears most often in $\{n_{v_i}\}$).
- Calculate the arithmetic mean of $\{n_{v_i}\}$, denoted by $\bar{K}_{val}$.
- Calculate the weighted average of $\{n_{v_i}\}$, denoted by $\tilde{K}_{val}$.

Obviously, it is easy to compute $\hat{K}_{val}$ and $\bar{K}_{val}$ in training. As for $\tilde{K}_{val}$, we need to assign a set of weighting coefficients for all the nodes in V_{val}, which requires us to consider how to evaluate the "importance" of each node. There exists several forms of *node centralities* that could be used to measure the relative importance of different nodes, such as degree centrality, eigenvector centrality, and betweenness centrality. For the first method (i.e., degree centrality), suppose the degree of node v_i is denoted by D_{v_i}, then the standardized degree centrality [21] of v_i could be computed by

$$\frac{D_{v_i}}{N-1}, \tag{7}$$

where N represents the number of nodes in G. However, once N is very large, it would make all nodes' degree centrality indistinguishable. Besides, adopting eigenvector centrality or betweenness centrality to evaluate a node's importance might introduce significant computing overhead, especially when dealing with large graphs.

To derive a set of clearly distinguishable weighting coefficients with low computational cost, we first count the number of nodes with different numbers of predicted neighbor classes (i.e., n_{v_i}) at different training stages (see Fig. 3). It is clear in the figure that there exists a highly skewed long-tailed distribution of nodes with different n_{v_i}, which indicates a majority of nodes have small number of different neighbor types (i.e., $n_{v_i}=1,2,3$) while the "tail" nodes with larger n_{v_i} ($n_{v_i}=4,5,6,7$) would account for very small portions in V_{val}. Therefore, to address the imbalance issue revealed above, we need to not only consider the majorities in $\{n_{v_i}\}$ but also the minor nodes in V_{val} with larger n_{v_i}. Suppose C_{val} denotes the number of n_{v_i}'s classes in $\{n_{v_i}\}$ and $V_{val,c}$ ($c\in[1,C_{val}]$) denotes the c-th type of nodes in V_{val}, then we have

$$\sum\nolimits_{c\in[1,C_{val}]}|V_{val,c}|=N_{val}. \tag{8}$$

Inspired by the classic approach in imbalanced label learning [22,23] which uses the reciprocal of the number of samples in each class as a balanced weight, we take the normalized reciprocal of $|V_{val,c}|$ (denoted by rw_c) to serve as the weighting coefficient for the cth classes in V_{val}, i.e.,

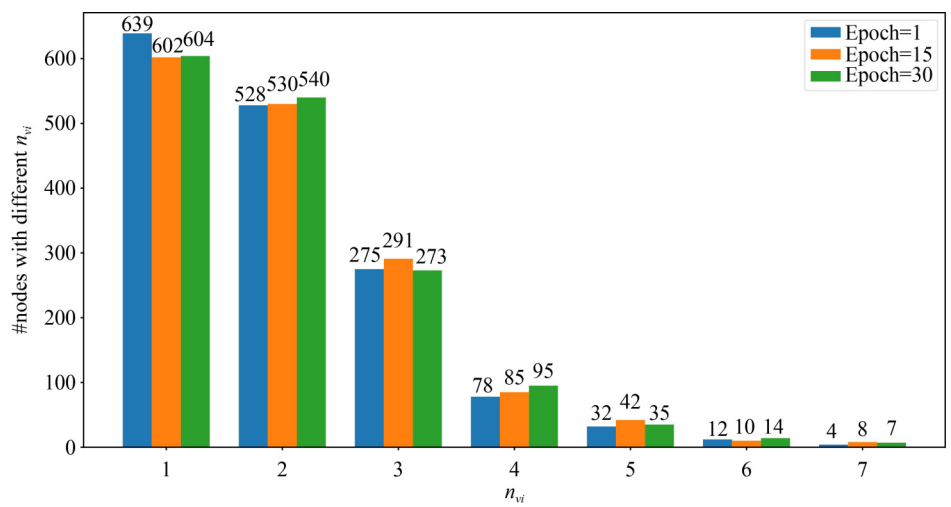


Fig. 3 Counts of nodes with different numbers of predicted neighbor classes (i.e., n_{v_i}) at different training stages on the Cora dataset

$$rw_c = \frac{|V_{val,c}|^{-1}}{\sum_{c\in[1,C_{val}]}|V_{val,c}|^{-1}}. \tag{9}$$

To this end, we could compute $\tilde{K}_{val}$ by

$$\tilde{K}_{val} = \sum\nolimits_{c\in[1,C_{val}]} c\cdot rw_c. \tag{10}$$

To determine which of the above strategies (i.e., $\hat{K}_{val}$, $\bar{K}_{val}$, and $\tilde{K}_{val}$) is optimal in adjusting K, we depict the changes of $\hat{K}_{val}$, $\bar{K}_{val}$, and $\tilde{K}_{val}$ in Fig. 4. It is implied in Figs. 4(a) and 4(b) that although $\hat{K}_{val}$ and $\bar{K}_{val}$ could cover the predicted neighbor classes which appear most often (i.e., class 1, class 2, and class 3 in Fig. 3), it is difficult for them to sufficiently discover the inherently separable connection factors in a neighborhood containing a large number of neighbor classes. On the contrary, Fig. 4(c) demonstrates that $\tilde{K}_{val}$ could cover nearly all the nodes in V_{val} that have various types of neighbors. Consequently, in our work, we consider $\tilde{K}_{val}$ as the *monitored variable* to adjust K automatically and help to find optimal disentangled node representations.

3.2 Dynamic disentanglement

According to the description of NRM (i.e., the neighborhood routing mechanism) in Section 2.2, NRM is applied in each graph convolutional layer (GCL), with K being the main parameter in NRM. Since each training epoch consists of several GCLs, the adjustment of K could be performed on two levels, i.e., layer-level and epoch-level.

3.2.1 Epoch-level disentanglement

Based on Section 3.1, we could utilize $\tilde{K}_{val}$ as a reference for updating K at different training stages. Once K is updated, we would use the new K to refine the disentanglement of node representations. Through tests we find that if K is adjusted every epoch, the potential for disentangling node features with the new K might not be fully exploited, which is also implied in the hyperparameter sensitivity experiment in Section 5.6.1. Thus we decide to change the value of K every λ epochs based on the monitoring of $\tilde{K}_{val}$. Here a period of λ epochs is called an *epoch stride* and the value of K is only updated at the first epoch in each epoch stride.

Specifically, at epoch m ($m = 1,2,...,M$), if $m-1$ is divisible by λ, meaning that epoch m is the first epoch of a certain epoch stride, we would leverage the $\tilde{K}_{val}$ obtained at epoch $m-1$ (denoted by $\tilde{K}_{val}^{(m-1)}$, i.e., the $\tilde{K}_{val}$ derived at the λ-th epoch of the previous epoch stride) to serve as the new K being deployed from epoch m to epoch $m+\lambda-1$, i.e.,

$$K^{(j)} = [\tilde{K}_{val}^{(m-1)}],\ j\in[m,m+\lambda-1], \tag{11}$$

where $[\cdot]$ represents the rounding operation to $\tilde{K}_{val}^{(m-1)}$ since $\tilde{K}_{val}^{(m-1)}$ is a weighted average value calculated by Eq. (10) and might not be an integer. On the contrary, if $m-1$ is not divisible by λ, i.e., epoch m is not the first epoch in a particular epoch stride, we would not alter K during epoch m.

Let $d_{in}^{(m)}$ be the input dimension of a node's representation in each graph convolutional layer of epoch m. Suppose each node's feature space is decomposed into $K^{(m)}$ subspaces (i.e., $K^{(m)}$ disentangled feature channels), then the input dimension of each channel (denoted by $\Delta d_{in}^{(m)}$) in each layer could be represented by

$$\Delta d_{in}^{(m)} = [d_{in}^{(m)}/K^{(m)}], \tag{12}$$

where $[\cdot]$ is a rounding operation as $d_{in}^{(m)}/K^{(m)}$ might not be an integer. Since both $d_{in}^{(m)}$ and $K^{(m)}$ remain constant across all convolutional layers at epoch m, $\Delta d_{in}^{(m)}$ would also remain fixed during epoch m.

Besides, it could be inferred from Eqs. (10) and (11) that the value of K updated in our experiments would be an integer ranging from 1 to C where C represents the ground-truth number of classes of all the nodes in a graph. Once $\tilde{K}_{val}$ have converged, K would hardly change in the rest of the training process.

3.2.2 Layer-level disentanglement

If we continue to alter K at the layer level in addition to the epoch level, we need to figure out how n_{v_i} changes across layers rather than across training epochs (see Fig. 2), in which n_{v_i} represents the number of predicted classes of node v_i's neighbors (already introduced in Section 3.1). Since n_{v_i} is obtained after all convolutional layers have been executed in each epoch, to observe the change in n_{v_i} after each layer's completion, we conduct a test on setting up only one convolutional layer within each training epoch. In this case, the change of n_{v_i} in an epoch could be equivalent to the change of n_{v_i} in a convolutional layer. We depict the results of two typical kinds of nodes in Fig. 5. Figure 5(a) describes the type of node (denoted by v_α) with nearly unchanged n_{v_i} across layers, while Fig. 5(b) describes the other type of node (denoted by v_β) whose number of predicted neighbor classes

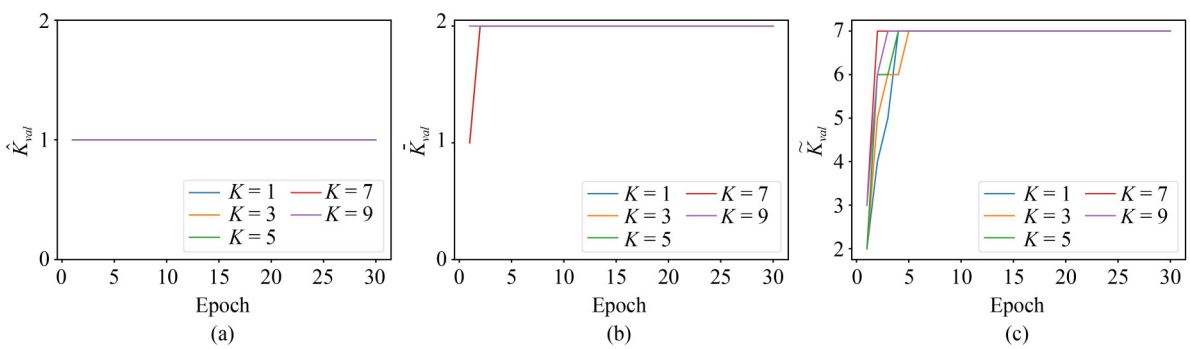


Fig. 4 Changes of $\hat{K}_{val}$, $\bar{K}_{val}$, and $\tilde{K}_{val}$ during training with different K in DisenGCN on the Cora dataset. (a) Changes of $\hat{K}_{val}$; (b) changes of $\bar{K}_{val}$; (c) changes of $\tilde{K}_{val}$

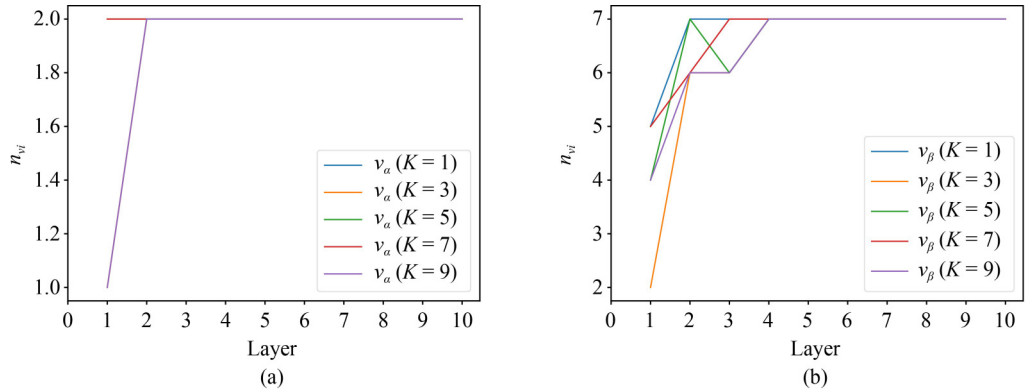


Fig. 5 Changes of n_{v_i} w.r.t. two typical nodes across layers with different K in DisenGCN on the Cora dataset. (a) The test with v_α; (b) the test with v_β

(i.e., n_{v_i}) would converge after only a few number of convolutional layers even with different K. Thus to cover the situations of all the nodes in a graph, we decide to increase K layer by layer based on the observation in Fig. 5(b).

Let $d_{in}^{(m,l)}$ be the input dimension of a node's representation in the lth ($l = 1, 2, ..., L$) layer of epoch m and each node's features are divided into $K^{(m,l)}$ disentangled feature channels. Suppose $\Delta d_{in}^{(m,l)}$ denotes the input dimension of each channel in layer l within epoch m, then $\Delta d_{in}^{(m,l)}$ could be computed by

$$\Delta d_{in}^{(m,l)} = [d_{in}^{(m,l)} / K^{(m,l)}], \tag{13}$$

where $[\cdot]$ represents a rounding operation as $d_{in}^{(m,l)} / K^{(m,l)}$ might not be an integer. Different from Eq. (12) where K and the input dimension of a node's representation are both kept unchanged across different graph convolutional layers, in layer-level disentanglement, based on Eq. (13), $K^{(m,l)}$ would be different in different layers. When $K^{(m,l)}$ varies across layers, we could manage $d_{in}^{(m,l)}$ and $\Delta d_{in}^{(m,l)}$ in two ways as explained below.

♣ **Fixed** $d_{in}^{(m,l)}$. When $d_{in}^{(m,l)}$ is kept constant throughout the whole training process, motivated by the observation from Fig. 5, we enlarge $K^{(m,l)}$ layer by layer until it reaches the ground-truth number (denoted by C) of node classes in a graph, i.e.,

$$K^{(m,l)} = K^{(m,l-1)} + \Delta K,\ K^{(m,l)} \in [1, C], \tag{14}$$

where $\Delta K \in \mathbb{N}_+$ is a hyperparameter and we would discuss its effect on the quality of disentangling in Section 5.6.2. The rationale for maintaining a fixed value of $d_{in}^{(m,l)}$ in training mainly comes from two dimensions. On the one hand, $\Delta d^{(m,l)}$ would become smaller if $K^{(m,l)}$ gets larger, which means that the number of features in each disentangled channel would decrease, thus weakening the learning ability of each channel. On the other hand, $\Delta d^{(m,l)}$ would become larger when $K^{(m,l)}$ gets smaller, indicating that the number of features in each disentangled channel is enlarged, i.e., the learning ability of each channel would be enhanced.

♠ **Fixed** $\Delta d_{in}^{(m,l)}$. Let $\Delta d_{in}^{(m,l)} = \Delta d$, then $d_{in}^{(m,l)}$ is determined by

$$d_{in}^{(m,l)} = K^{(m,l)} \times \Delta d = (K^{(m,l-1)} + \Delta K) \times \Delta d. \tag{15}$$

Additionally, when dealing with multi-label graphs in which each node would have multiple binary labels, to increase the model's capability in learning the representations of nodes with high entangled features in such graphs, we would aggregate the node representations learned in all L convolutional layers through tensor addition to do node classification.

3.2.3 Resize of model parameters

On the one hand, when we use fixed $d_{in}^{(m)}$ (see Eq. (12)) and $d_{in}^{(m,l)}$ (see Eq. (13)), the input dimension of any layer at any epoch appears to be the same during training. However, if $d_{in}^{(m)}$ (or $d_{in}^{(m,l)}$) cannot be divided by $K^{(m)}$ (or $K^{(m,l)}$), the dimension of a node's last disentangled feature channel would be set smaller than other channels. On the other hand, when we fix $\Delta d^{(m,l)}$ throughout the training process, once the value of K is altered at the beginning (e.g., at epoch m) of an epoch stride, according to Eqs. (11) and (15), the input dimension of each layer would differ from that of epoch $m-1$, which then leads to a mismatch in weight parameter sizes between two adjacent epoch strides.

Specifically, for node u, suppose its input representation in the lth layer at epoch m is represented by $\mathbf{z}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times 1}$, then before aggregating all u's neighbors' representations, $\mathbf{z}_u^{(m,l)}$ would be calculated by

$$\mathbf{z}_u^{(m,l)} = (\mathbf{W}_u^{(m,l)})^{\mathrm{T}} \cdot \mathbf{z}_u^{(m,l-1)} + \mathbf{b}_u^{(m,l)}, \tag{16}$$

where $\mathbf{W}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times d_{out}^{(m,l)}}$ and $\mathbf{b}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times 1}$ are the corresponding 2D weight matrix and 1D bias vector, respectively.

Following Eqs. (11) and (15), if epoch m is the *first epoch* of a new epoch stride, $d_{in}^{(m,l)}$ would be different from $d_{in}^{(m-1,l)}$. In this case, the 2D and 1D weight parameters obtained in the $(l-1)$th layer, i.e., $\mathbf{W}_u^{(m-1,l)} \in \mathbb{R}^{d_{in}^{(m-1,l)} \times d_{out}^{(m-1,l)}}$ and $\mathbf{b}_u^{(m-1,l)} \in \mathbb{R}^{d_{in}^{(m-1,l)} \times 1}$, could not be directly applied to Eq. (16). Since neural networks commonly keep the input and output dimensions of each layer constant in training, if we change each layer's input or output dimension at epoch m due to the update on K, the resizing of the corresponding weight tensors becomes necessary. Inspired by the practice of handling inputs of varying sizes in CNNs [24] and RNNs [25] (i.e.,

performing *padding* or *truncation* operations on the inputs), we resize $\mathbf{W}_u^{(m,l)}$ and $\mathbf{b}_u^{(m,l)}$ based on the following three principles (also illustrated in Algorithm 1):

❶ **When** $K^{(m,l)} = K^{(m-1,l)}$, we do not need to adjust the dimensions of weight tensors. We could directly use the parameters obtained from the $(m-1)$th epoch to do the training at epoch m (line 2).

❷ **When** $K^{(m,l)} > K^{(m-1,l)}$, which means that we should increase K at epoch m to enhance the disentanglement of hidden feature factors, thereby resulting in the growth of the sizes of both 2D and 1D weight tensors. Thus we have to expand $\mathbf{W}_u^{(m-1,l)}$ to fit in with the enlarged $d_{in}^{(m,l)}$. One common technique in CNNs or RNNs to handle different inputs (i.e., images or sentences) that vary in size is to apply padding to them, which aligns them to a weight tensor whose dimensions are constant. However, in GCNs, the inputs of the network are vector embeddings of different nodes which would share the same dimension in training, i.e., the dimensions of all the nodes would change synchronously if required. Once the dimension of a node u's node embedding (i.e., $\mathbf{z}_u^{(m,l)}$) changes, we must alter the sizes of the weight parameters that do tensor addition or multiplication with $\mathbf{z}_u^{(m,l)}$ (see Eq. (16)).

The traditional padding methods employed in CNNs or RNNs typically involve generating data randomly from a standard normal distribution or simply utilizing zero-value data. Since we construct $\mathbf{W}_u^{(m,l)}$ based on $\mathbf{W}_u^{(m-1,l)}$, with the iteration of the network, we find that generating random non-zero data would introduce extra noise to the newly added dimensions of $\mathbf{z}_u^{(m,l)}$ and thereby harm the model's classification performance. Thus we adopt zero padding in our work to increase the dimensions of the 2D weight matrix $\mathbf{W}_u^{(m-1,l)}$ and the 1D bias vector $\mathbf{b}_u^{(m-1,l)}$.

Note that in Eq. (16), $\mathbf{z}_u^{(m,l)}$, $\mathbf{z}_u^{(m,l-1)}$, $\mathbf{W}_u^{(m,l)}$, and $\mathbf{b}_u^{(m,l)}$ belong to $\mathbb{R}^{d_{in}^{(m,l)}\times 1}$, $\mathbb{R}^{d_{in}^{(m,l-1)}\times 1}$, $\mathbb{R}^{d_{in}^{(m,l)}\times d_{out}^{(m,l)}}$, and $\mathbb{R}^{d_{in}^{(m,l)}\times 1}$, respectively. According to the basic rules of tensor multiplication and addition, once the dimension of $\mathbf{z}_u^{(m,l)}$ (a 1D feature vector) enlarges from $d_{in}^{(m-1,l)}$ to $d_{in}^{(m,l)}$ based on Eq. (11), for $\mathbf{W}_u^{(m-1,l)}$, we need to increase both the first dimension (i.e., the number of rows) and the second dimension (i.e., the number of columns) of it with zero paddings to generate $\mathbf{W}_u^{(m,l)}$ (lines 4–7). Besides, for $\mathbf{b}_u^{(m-1,l)}$, we only need to increase its first dimension to form $\mathbf{b}_u^{(m,l)}$ (lines 8–9).

❸ **When** $K^{(m,l)} < K^{(m-1,l)}$, which indicates that the disentangling of node representations should be weakened, i.e., K becomes smaller at epoch m. In complete contrast to the padding approach used in the case where $K^{(m,l)} > K^{(m-1,l)}$, we apply the truncation operation on the 2D weight matrix $\mathbf{W}_u^{(m-1,l)}$ and the 1D bias vector $\mathbf{W}_u^{(m-1,l)}$ when $K^{(m,l)} < K^{(m-1,l)}$. Specifically, for $\mathbf{W}_u^{(m-1,l)}$, we reduce both its first and second dimensions by deleting its redundant rows and columns with the smallest row sums (line 11) and column sums (line 12), respectively. Similarly, for $\mathbf{b}_u^{(m-1,l)}$, we just eliminate the smallest redundant weights in it (line 13).

Algorithm 1 Parameter resizing algorithm

Input: $\mathbf{W}_u^{(m-1,l)} \in \mathbb{R}^{d_{in}^{(m-1,l)}\times d_{out}^{(m-1,l)}}$: 2D weight matrix at epoch $m-1$; $\mathbf{b}_u^{(m-1,l)} \in \mathbb{R}^{d_{in}^{(m-1,l)}\times 1}$: 1D bias vector at epoch $m-1$, $m = 1, 2, ..., M$.

Output: $\mathbf{W}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)}\times d_{out}^{(m,l)}}$: reconstructed 2D weight matrix at epoch m; $\mathbf{b}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)}\times 1}$: reconstructed 1D bias vector at epoch m.

1 **if** $K^{(m,l)} = K^{(m-1,l)}$ **then**
2 $\mathbf{W}_u^{(m,l)} \leftarrow \mathbf{W}_u^{(m-1,l)}$, $\mathbf{b}_u^{(m,l)} \leftarrow \mathbf{b}_u^{(m-1,l)}$;
3 **else if** $K^{(m,l)} > K^{(m-1,l)}$ **then**
4 Generate $\mathbf{W}_{u,zerorow}^{(m,l)} \in \mathbb{R}^{(d_{in}^{(m,l)}-d_{in}^{(m-1,l)})\times d_{out}^{(m-1,l)}}$ with all-zero values;
5 Generate $\mathbf{W}_{u,zerocol}^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)}\times(d_{out}^{(m,l)}-d_{out}^{(m-1,l)})}$ with all-zero values;
6 $\mathbf{W}_u^{(m,l)} \leftarrow$ concatenate $\mathbf{W}_u^{(m-1,l)}$ with $\mathbf{W}_{u,zerorow}^{(m,l)}$ vertically;
7 $\mathbf{W}_u^{(m,l)} \leftarrow$ concatenate $\mathbf{W}_u^{(m,l)}$ with $\mathbf{W}_{u,zerocol}^{(m,l)}$ horizontally;
8 Generate $\mathbf{b}_{u,zerorow}^{(m,l)} \in \mathbb{R}^{(d_{in}^{(m,l)}-d_{in}^{(m-1,l)})\times 1}$ with all-zero values;
9 $\mathbf{b}_u^{(m,l)} \leftarrow$ concatenate $\mathbf{b}_u^{(m-1,l)}$ and $\mathbf{b}_{u,zerorow}^{(m,l)}$ vertically;
10 **else**
11 $\mathbf{W}_u^{(m,l)} \leftarrow$ remove $d_{in}^{(m-1,l)} - d_{in}^{(m,l)}$ rows of $\mathbf{W}_u^{(m-1,l)}$ with the smallest row sums;
12 $\mathbf{W}_u^{(m-1)} \leftarrow$ remove $d_{out}^{(m-1,l)} - d_{out}^{(m,l)}$ columns of $\mathbf{W}_u^{(m,l)}$ with the smallest column sums;
13 $\mathbf{b}_u^{(m,l)} \leftarrow$ remove $d_{in}^{(m-1,l)} - d_{in}^{(m,l)}$ rows of $\mathbf{b}_u^{(m-1,l)}$ with the smallest values.

3.3 Architecture of D^2-GCN

Algorithm 2 describes the proposed dynamic strategy for adjusting K during training, in which DisenConv means the disentangled convolutional operations and NRM represents the neighborhood routing mechanism. We let $K = 1$ in the first layer of each epoch in the first epoch stride to initialize the model (lines 4–10). Then we begin to update K from the $(\lambda+1)$th epoch on (lines 12–28) based on the two-level disentangling approaches, i.e., the epoch-level disentanglement and the layer-level disentanglement illustrated in Section 3.2.1 and Section 3.2.2, respectively. It is worth noting that the layer-level disentanglement is an elective module in training (lines 4–7, 13–16, and 22–25), whose effect on the model performance would be evaluated through ablation study in Section 5.7.

The parameter resizing technique is only adopted at the first epoch in a particular epoch stride (line 20). We record the validation and test accuracies obtained at each epoch throughout the whole training process (line 29). When the best validation accuracy (i.e., val_{acc} in line 32) is observed at a certain epoch, the test accuracy obtained alongside val_{acc} is taken as the model's final accuracy result (i.e., $test_{acc}$ in line 33). Once $\tilde{K}_{val}$ has converged in D^2-GCN, the value of K would be rarely changed in the rest of the training process. The overall architecture of D^2-GCN is also depicted in Fig. 6.

3.4 Complexity of D^2-GCN

For a graph G with N nodes in it, we let d be the number of

Algorithm 2 Dynamic adjustment of K

Input: M: the number of training epochs; $K^{(m,l)}$: the number of disentangled feature channels in the lth convolutional layer at the mth epoch ($l = 1, 2, ..., L$ and $m = 1, 2, ..., M$); λ: the length of the epoch stride; ΔK: the growth rate of K across layers; $\tilde{K}_{val}^{(m)}$: the weighted average number of predicted classes of a node's neighbors at epoch m.

Parameter: $\mathbf{W}_{u,k}^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times \frac{d_{out}^{(m,l)}}{K^{(m,l)}}}$: 2D weight matrix w.r.t. the kth feature channel of $\mathbf{z}_u^{(m,l)}$; $\mathbf{b}_{u,k}^{(m,l)} \in \mathbb{R}^{\frac{d_{in}^{(m,l)}}{K^{(m,l)}} \times 1}$: 1D bias vector w.r.t. the kth feature channel of $\mathbf{z}_u^{(m,l)}$, $k = 1, 2, ..., K^{(m)}$.

Output: $(val_{acc}, test_{acc})$: the best classification accuracies on the validation set and the test set.

for $m = 1, 2, ..., M$ **do**
  if $1 \le m \le \lambda$ **then**
    if *doing layer-level disentanglement* **then**
      $K^{(m,1)} \leftarrow 1$;
      for $l = 2, ..., L$ **do**
        $K^{(m,l)} \leftarrow K^{(m,l-1)} + \Delta K$;
    else
      for $l = 1, ..., L$ **do**
        $K^{(m,l)} \leftarrow 1$;
  else
    if $(m-1) \mod \lambda = 0$ **then**
      if *doing layer-level disentanglement* **then**
        $K^{(m,1)} \leftarrow \tilde{K}_{val}^{(m-1)}$;
        for $l = 2, ..., L$ **do**
          $K^{(m,l)} \leftarrow K^{(m,l-1)} + \Delta K$;
      else
        for $l = 1, ..., L$ **do**
          $K^{(m,l)} \leftarrow \tilde{K}_{val}^{(m-1)}$;
      Resize $\mathbf{W}_u^{(m-1,l)}$ and $\mathbf{b}_u^{(m-1,l)}$ to initialize $\mathbf{W}_u^{(m,l)}$ and $\mathbf{b}_u^{(m,l)}$, respectively; ▷ by `Algorithm 1`
    else
      if *doing layer-level disentanglement* **then**
        $K^{(m,1)} \leftarrow K^{(m-1,1)}$;
        for $l = 2, ..., L$ **do**
          $K^{(m,l)} \leftarrow K^{(m,l-1)} + \Delta K$;
      else
        for $l = 1, ..., L$ **do**
          $K^{(m,l)} \leftarrow K^{(m-1,l)}$;
  Obtain $(val_{acc}^{(m)}, test_{acc}^{(m)})$; ▷ by `DisenConv and NRM`
  Update $\{\mathbf{W}_{u,k}^{(m,l)}\}$ and $\{\mathbf{b}_{u,k}^{(m,l)}\}$; ▷ by `backpropagation`
  $\mathbf{W}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times d_{out}^{(m,l)}} \leftarrow$ concatenate $\{\mathbf{W}_{u,k}^{(m,l)}\}$, $\mathbf{b}_u^{(m,l)} \in \mathbb{R}^{d_{in}^{(m,l)} \times 1} \leftarrow$ concatenate $\{\mathbf{b}_{u,k}^{(m,l)}\}$;
$val_{acc} \leftarrow \max\{val_{acc}^{(m)}\}, m = 1, 2, ..., M$;
$test_{acc} \leftarrow$ the test accuracy at the epoch when val_{acc} is obtained.

each node's feature dimensions and L be the number of graph convolutional layers. Regarding memory occupation, we need $O(NdL)$ space to store node embeddings. Besides, additional $O(d_{in}d_{out}L)$ memory is required to save model weights $\{\mathbf{W}^l \mid \mathbf{W}^l \in \mathbb{R}^{d_{in} \times d_{out}}, l \in [1, L]\}$ in training.

As to time complexity, in the lth layer, the calculations are mainly matrix multiplications, i.e., $\mathbf{A}'\mathbf{Z}^l\mathbf{W}^l$ where $\mathbf{A}'$ is the normalized form of the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}^l \in \mathbb{R}^{N \times d}$ implies the representations of all nodes' features. Therefore, it would take $O(\|\mathbf{A}\|_0 d + Nd_{in}d_{out})$ time to train at layer l where $\|\mathbf{A}\|_0$ is the number of non-zero elements in $\mathbf{A}$.

4 Convergence analysis

In this section, we first prove the convergence of the dynamic adjustment of K in D^2-GCN. Then we propose a criterion for evaluating the model's disentangling quality during training.

4.1 Convergence of dynamic disentanglement

Let $\theta = \{\mathbf{c}_{u,k}\}_{k=1}^K$, $R = \{r_v : (u, v) \in E\}$ (r_v implies the unknown factor why node u and its neighbor v are connected), and $Z = \{\mathbf{z}_{u,k}\} \cup \{\mathbf{z}_{v,k} \mid v : (u, v) \in E\}$ $(k = 1, 2, ..., K)$. Z could be served as the *observable variables* in the definition of the expectation-maximization (EM) algorithm while R represents the *latent variables*. The dynamic adjustment of K would directly impact the number of variables in Z and R. Since the EM algorithm would automatically search for an optimal solution, and let $\sum_R p(Z, R; \theta)$ reach the maximum despite how many observed data are given, it does not affect the local

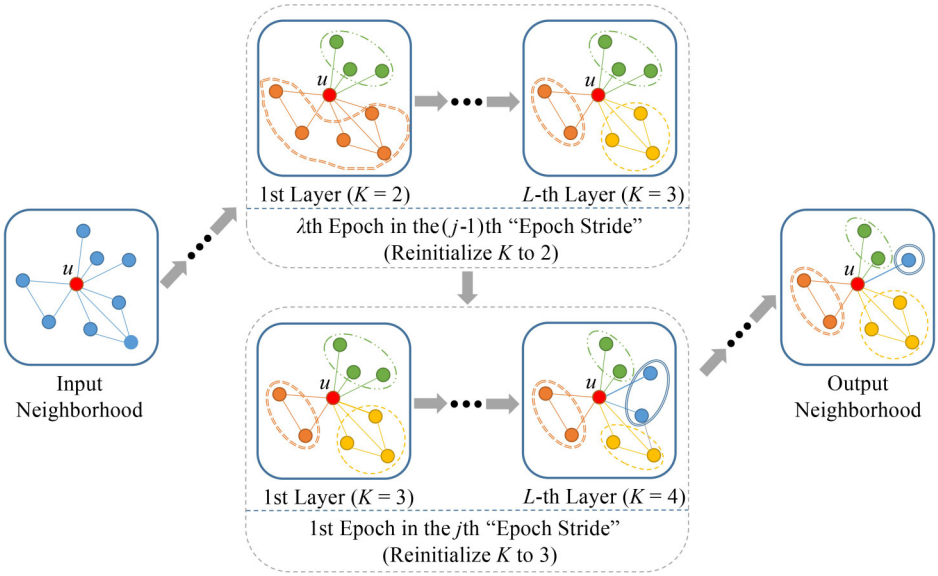


Fig. 6 Architecture of D^2-GCN with two-level disentanglement during training, i.e., the epoch level and the layer level

convergence behavior of NRM when changing K. However, the global convergence of the whole training process remains to be demonstrated. Here we propose a theorem to prove the convergence of D^2-GCN.

Theorem 1 Given dynamic K, D^2-GCN still converges to a point estimate of $\{\mathbf{c}_{u,k}\}_{k=1}^{K}$ that maximizes the marginal likelihood $p(Z;\theta)$.

Proof First we have $p(Z;\theta)=p(Z,R;\theta)/p(R|Z,\theta)$ and take the log on both sides of it:

$$\log p(Z;\theta)=\log p(Z,R;\theta)-\log p(R|Z,\theta). \tag{17}$$

According to the definition of the EM algorithm,

$$Q(\theta,\theta^{(s-1)})=\sum\nolimits_R \log p(Z,R;\theta)p(R|Z,\theta^{(s-1)}), \tag{18}$$

is used to find the estimate θ^* that makes $p(Z;\theta)$ reach the maximum where $\theta^{(s-1)}$ is the estimate acquired at iteration $s-1$ ($s=1,2,...,S$).

Let $H(\theta,\theta^{(s-1)})=\sum_R \log p(R|Z,\theta)p(R|Z,\theta^{(s-1)})$, we obtain that $\log p(Z;\theta)=Q(\theta,\theta^{(s-1)})-H(\theta,\theta^{(s-1)})$. Then we let

$$\begin{aligned}&\log p(Z;\theta^{(s)})-\log p(Z;\theta^{(s-1)})\\&=(Q(\theta^{(s)},\theta^{(s-1)})-Q(\theta^{(s-1)},\theta^{(s-1)}))\\&-(H(\theta^{(s)},\theta^{(s-1)})-H(\theta^{(s-1)},\theta^{(s-1)})).\end{aligned} \tag{19}$$

Since K would vary at different training stages, the number of underlying factors attributing to the connections between node u and its neighbors would change accordingly. When K gets bigger, it implies that the former smaller K makes the model unable to grasp sufficient information in a neighborhood. Therefore, it requires the model to become more complex, i.e., with more feature units in θ to be learned. Besides, the analysis of decreasing K is completely opposite to that of increasing K.

As the model repeats the neighborhood routing algorithm in training, it would connect the estimate of θ between adjacent epoch strides. Furthermore, the dynamic modification on K might assist the model in escaping some local optimal estimates that may arise in static disentanglement.

Although the value of the Q function at the new epoch stride would change w.r.t. K, the updating strategy of K by monitoring the change of $\tilde{K}_{val}$ ensures that Q is monotonically increasing through training, i.e.,

$$Q(\theta^{(s)},\theta^{(s-1)})-Q(\theta^{(s-1)},\theta^{(s-1)})\geqslant 0. \tag{20}$$

Moreover, according to Jensen's inequality,

$$\begin{aligned}&H(\theta^{(s)},\theta^{(s-1)})-H(\theta^{(s-1)},\theta^{(s-1)})\\&=\sum\nolimits_R\left(\log\frac{p(R|Z,\theta^{(s)})}{p(R|Z,\theta^{(s-1)})}\right)p(R|Z,\theta^{(s-1)})\\&\leqslant\log\left(\sum\nolimits_R\frac{p(R|Z,\theta^{(s)})}{p(R|Z,\theta^{(s-1)})}p(R|Z,\theta^{(s-1)})\right)\\&=\log\left(\sum\nolimits_R\log p(R|Z,\theta^{(s)})\right)=0.\end{aligned} \tag{21}$$

Finally, the log-likelihood function $L(\theta)=\log p(Z;\theta)$ turns out to be a monotonically increasing function with an upper bound 0, which proves that the dynamic adjustment of K in our model not only converges locally but also globally. □

4.2 Convergence of KL divergence

Kullback-Leibler (KL) divergence [26] is commonly used to measure the difference in information represented by two distributions, which we leverage in this paper to depict the gap between the learned node label distribution and the real one on the validation set.

Assume there are T real node labels on the validation set V_{val}, and the numbers of nodes in these types are $N_1,N_2,...,N_T$ ($\sum_{t\in[1,T]}N_t=N_{val}$), respectively. We denote $p_t=N_t/N_{val}$ as the ground-truth existence probability of nodes in label type t. Inspired by Shannon's *information entropy* theory [27], we could represent the information of the existence of type t nodes by $-\log p_t$ in which the Euler's number e is used as the log base. Then we define the *label entropy* (LE) in V_{val} by

$$LE_{V_{val}}=\sum\nolimits_{t\in[1,T]}-p_t\log p_t, \tag{22}$$

which reflects the expected (i.e., average) level of information inherent to a node's possible labels. Note that $LE_{V_{val}}$ is also called the *base entropy* in this paper in that it reveals the intrinsic label distribution on V_{val}.

Following the definition of label entropy in V_{val}, the updated LE on the validation set (denoted by LE_{val_update}) during training could be represented by

$$LE_{val\_update} = \sum\nolimits_{t\in[1,T]} -p_{val,t}\log p_{val,t}, \tag{23}$$

where $p_{val,t} = n_{val,t}/N_{val}$ is the predicted existence probability of the tth type of nodes (with number of $n_{val,t}$) in the validation set. Note that LE_{val_update} would change during training. Therefore, the KL divergence between LE_{val_update} and $LE_{V_{val}}$, denoted by KL_{val}, is given by

$$KL_{val} = \sum\nolimits_{t\in[1,T]} p_{val,t}\log\frac{p_{val,t}}{p_t}. \tag{24}$$

In Fig. 7, we display the change of KL_{val} in five independent training processes. It is observed that KL_{val} always converges during training despite the different settings of K. Besides, larger K would exhibit faster convergence speed than smaller ones. As a result, KL_{val} could be served as a metric for describing the convergence speed of the multi-channel disentanglement.

5 Evaluation

We conduct comprehensive experiments in this section on six typical benchmark datasets to evaluate the performance of D^2-GCN against DisenGCN and other baseline models. We use a machine equipped with dual 2.40 GHz Intel Xeon Gold 6240R processors (24 cores in total), 256 GB main memory, and one NVIDIA Tesla V100 GPU (32 GB memory). The installed operating system is Ubuntu 18.04, using the libraries CUDA 11.1 and cuDNN 8.0.5. Our model is implemented using PyTorch 1.8.0 and Python 3.7.9.

5.1 Experimental setup

Datasets This paper focuses on node-level classification and takes graphs in seven frequently used real-world datasets as inputs, including three single-label datasets in citation networks and four multi-label datasets.

- Cora [20] contains bibliographic records of machine learning papers that have been manually clustered into groups that refer to the same publication domain. Each publication is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.
- Citeseer [20] is a scientific literature digital library and search engine that focuses primarily on the literature in computer and information science. The attributes of different nodes in the dataset are embedded the same way as in Cora.
- PubMed [20] contains scientific publications from the PubMed database pertaining to diabetes classified into one of three classes. Each publication is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.
- PPI is a subgraph of the PPI (Protein-Protein Interactions) network [28] for Homo Sapiens. The subgraph corresponds to the graph induced by nodes containing labels from the hallmark gene sets [29] and representing biological states.
- POS [30] is a co-occurrence network of words appearing in the Wikipedia dump. The node labels in the dataset represent the Part-of-Speech tags inferred by the Stanford POS Tagger [31].
- BlogCatalog [32] is a network of social relationships of the bloggers listed on the BlogCatalog website. The node labels in the dataset represent blogger interests inferred through the metadata provided by the bloggers.
- Flickr [33] is a network for amateur and professional photographers to share high-resolution photos. The node labels in the dataset represent user interests such as *Landscapes*, *Climbing*, and *Animals*. Each node's features are embedded as the 500-dimensional bag-of-word representation of the corresponding image provided by NUS-wide.

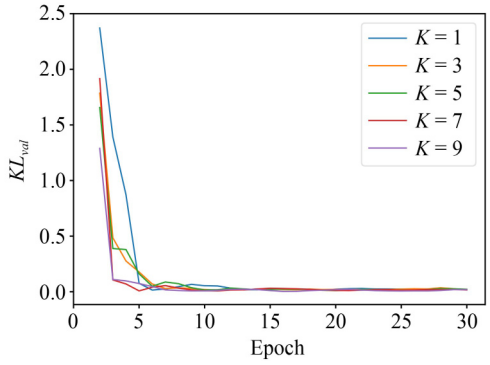


Fig. 7 Change of KL_{val} during training with different K in DisenGCN. The dataset used here is Cora

Since PPI, POS, and BlogCatalog do not contain node features, we use the rows of the adjacency matrices in them to serve as all nodes' feature vectors. The details of these datasets are described in Table 1.

Hyperparameters For the number of routing iterations in NRM (i.e., S), the l_2 regularization term (i.e., weight decay), the learning rate, the dropout rate, and the number of graph convolutional layers (i.e., L), we keep them the same as in DisenGCN. We only introduce two new hyperparameters in our model, i.e., λ (the length of the epoch stride described in Section 3.2) and ΔK (the growth scale of K across layers).

Table 1 Dataset information. The last column implies whether the graphs in the task contain single or multiple labels

Dataset	Type	#Nodes	#Edges	#Classes	#Node features	Task
Cora	Citation	2,708	5,429	7	1,433	Single
Citeseer	Citation	3,327	4,732	6	3,703	Single
PubMed	Citation	19,717	44,338	3	500	Single
PPI	Biological	3,890	76,584	50	–	Multi
POS	Word Co-occurrence	4,777	184,812	40	–	Multi
BlogCatalog	Social	10,312	333,983	39	–	Multi
Flickr	Social	89,250	899,756	7	500	Multi

The effects of λ and ΔK would be investigated via sensitivity analysis in Section 5.6. We specify the hyperparameters used in our work in Table 2.

Baselines In semi-supervised single-label tasks, we mainly compare D^2-GCN with DisenGCN, IPGDN, and two representative graph neural networks: GCNs [1] and GAT [34]. Other commonly used baseline models in node classification are shown in Table 3. We also adopt three node embedding algorithms, DeepWalk [35], LINE [36], and node2vec [37] as baselines for multi-label tasks. The experimental results of different models are recorded through five runs.

5.2 Single-label node classification

Each dataset in this task has been assigned only 20 labeled nodes in each class for training. We use the same dataset splits as in GCN and DeepWalk, i.e., all the labeled nodes for training, 1000 unlabeled nodes for testing, and the remaining nodes for validation.

The test accuracies are reported in Table 3. It implies in the table that D^2-GCN achieves the best classification results on three citation network datasets. Note that disentangled approaches (DisenGCN, IPGDN, and our model) outperform almost all the other holistic ones, demonstrating the benefit of disentangled node representation. Moreover, D^2-GCN surpasses DisenGCN by 2.64%, 2.80%, and 2.38% on Cora, Citeseer, and PubMed, respectively, which indicates that the dynamic K-channel adjustment in this paper helps enhance the learning ability of the disentangled model.

Table 2 Hyperparameter settings

Dataset	S	Weight decay	Learning rate	Dropout rate	L	λ	ΔK
Cora	6	0.001	0.061	0.41	6	5	2
Citeseer	6	0.086	0.090	0.26	5	5	2
PubMed	6	0.045	0.014	0.22	4	4	1
PPI	6	0.00059	0.0057	0.45	2	6	3
POS	6	0.00048	0.0780	0.45	2	4	3
BlogCatalog	6	0.00036	0.0079	0.40	4	4	4
Flickr	6	0.00025	0.0530	0.36	3	5	2

Table 3 Test accuracy results in single-label node classification on Cora, Citeseer, and PubMed (unit: percentage)

Model	Dataset		
	Cora	Citeseer	PubMed
MLP [38]	55.1±0.25	46.5±0.33	71.4±0.46
ManiReg [39]	59.5±0.18	60.1±0.24	70.7±0.31
SemiEmb [40]	59.0±0.23	59.6±0.21	71.1±0.28
LP [41]	68.0±0.13	45.3±0.22	63.0±0.26
DeepWalk [35]	67.2±0.11	43.2±0.32	65.3±0.36
ICA [42]	75.1±0.23	69.1±0.26	73.9±0.31
Planetoid [43]	75.7±0.22	64.7±0.19	77.2±0.30
ChebNet [44]	81.2±0.12	69.8±0.23	74.4±0.25
GCN [1]	81.5±0.17	70.3±0.23	79.0±0.42
MoNet [45]	81.7±0.24	71.4±0.27	78.8±0.35
GAT [34]	83.0±0.15	72.5±0.18	79.0±0.22
MixHop [46]	82.3±0.23	72.1±0.18	81.0±0.26
DisenGCN [18]	83.2±0.20	71.5±0.25	79.9±0.19
IPGDN [19]	83.9±0.15	72.8±0.19	80.7±0.21
D^2-GCN	**85.4±0.11**	**73.5±0.12**	**81.8±0.14**

5.3 Multi-label node classification

To evaluate the classification performance of our work on graphs in which each node would have multiple different labels, we take 50% of the total number of nodes in the dataset for training, 25% nodes for validation, and the rest 25% for inference. We adopt Macro-F1 and Micro-F1 scores to compare the node classification results between our model and five baseline models in Table 4.

It is shown in the table that both DisenGCN and our model (D^2-GCN) improve a lot in Macro-F1 and Micro-F1 scores compared with other holistic algorithms, which again demonstrates the effectiveness of the disentangling mechanism. Besides, D^2-GCN outperforms DisenGCN by up to 4.91% in Macro-F1 scores and 3.50% in Micro-F1 scores on the four multi-label datasets.

5.4 Convergence of D^2-GCN

In Figs. 8, 9, and 10, the comparisons between our model (D^2-GCN) and DisenGCN in terms of training/test loss, $\tilde{K}_{val}$, and KL_{val} are shown, respectively. It is observed that although there may exist some peak values during training due to the adjustment of K at the first epoch of every epoch stride (described at the beginning of Section 3.2), the automatic refactorization of latent feature factors in D^2-GCN does not affect the convergence of the model.

As demonstrated in the proof of Theorem 1, the reason why D^2-GCN still keeps good convergence is that we adjust K in accordance with the variable weighted average number of all nodes' neighbor classes (i.e., $\tilde{K}_{val}$) on the validation set, which could help the model dynamically grasp the instant variations of node representations and jump out of some possible local minima during training. Note that D^2-GCN would always quickly find the best K on each dataset even if we only set $K = 1$ in the first epoch stride.

5.5 Complexity of D^2-GCN

The modification to K in our paper differs from common hyperparameter optimization (HPO) methods. HPO approaches are often implemented through an *offline* grid search, which would cost much time in trials and errors,

Table 4 Test Macro-F1 and Micro-F1 scores in multi-label node classification on PPI, POS, and BlogCatalog (unit: percentage)

	Model	Dataset			
		PPI	POS	BlogCatalog	Flickr
Macro-F1	DeepWalk	17.8±0.19	11.9±0.21	23.1±0.24	15.3±0.26
	LINE	17.3±0.25	11.8±0.18	22.8±0.27	19.7±0.15
	node2vec	17.9±0.19	12.9±0.25	23.6±0.31	22.4±0.27
	GCN	17.7±0.16	20.1±0.20	24.4±0.28	27.2±0.18
	GAT	19.2±0.13	16.1±0.22	26.8±0.25	30.6±0.23
	DisenGCN	21.4±0.17	26.5±0.21	28.9±0.15	34.2±0.14
	D^2-GCN	**22.3±0.11**	**27.8±0.14**	**29.6±0.12**	**35.7±0.17**
Micro-F1	DeepWalk	20.7±0.20	49.3±0.22	38.8±0.26	30.6±0.17
	LINE	21.1±0.23	49.0±0.16	38.5±0.24	34.2±0.19
	node2vec	20.6±0.17	49.9±0.22	39.0±0.28	38.4±0.31
	GCN	20.7±0.18	50.9±0.18	35.9±0.25	44.3±0.12
	GAT	22.3±0.14	50.7±0.16	39.7±0.22	46.5±0.23
	DisenGCN	25.7±0.18	54.1±0.21	41.8±0.16	49.2±0.16
	D^2-GCN	**26.6±0.15**	**55.4±0.11**	**42.7±0.14**	**50.9±0.15**

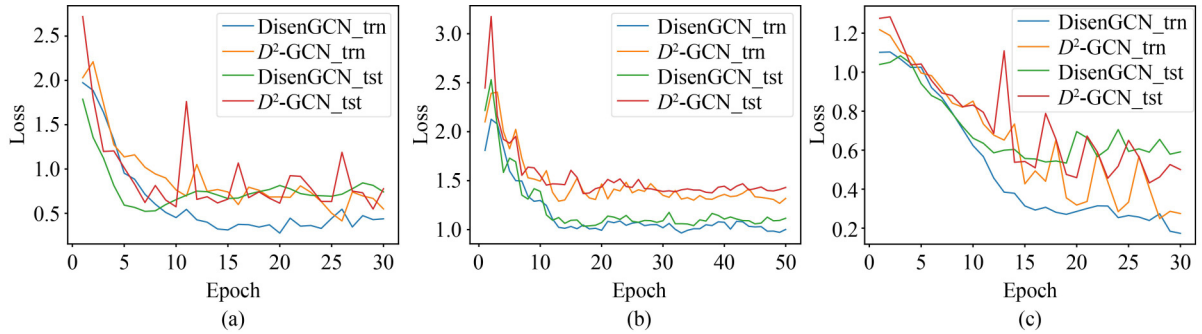


Fig. 8 Loss curves in DisenGCN and D^2-GCN on Cora, Citeseer, and PubMed. Here 'trn' or 'tst' means the loss on the training/test set. (a) On Cora; (b) on Citeseer; (c) on PubMed

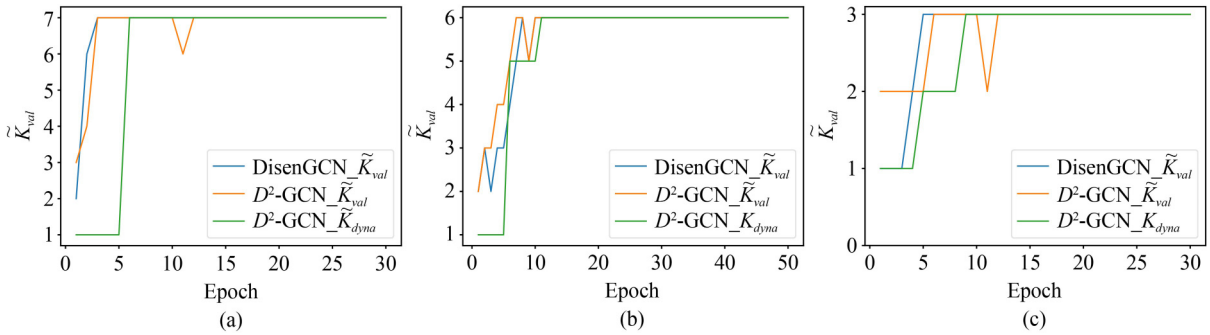


Fig. 9 $\tilde{K}_{val}$ and $\tilde{K}_{dyna}$ in DisenGCN and D^2-GCN on Cora, Citeseer, and PubMed. Here $\tilde{K}_{dyna}$ means the K altered by our model during training. (a) On Cora; (b) on Citeseer; (c) on PubMed

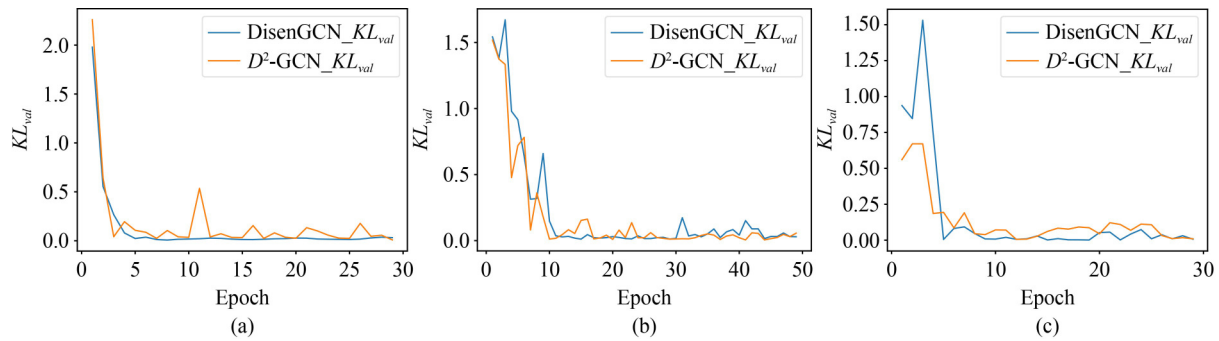


Fig. 10 KL_{val} in DisenGCN and D^2-GCN on Cora, Citeseer, and PubMed. (a) On Cora; (b) on Citeseer; (c) on PubMed

especially when dealing with large graphs. The adaptive adjustment of K in D^2-GCN is one kind of *online* method. Specifically, it only needs to train once in finding the optimal K by monitoring the weighted average number of different nodes' neighbor classes on the validation set.

We use different K to run multiple independent trainings to serve as an HPO process (denoted by DisenGCN$_{HPO}$), in which the search space of K (i.e., from 1 to the number of ground-truth node classes C) is set the same as in D^2-GCN. We compare the time cost between the HPO implementation and our model in Table 5. It is shown in Table 5 that D^2-GCN trains up to 4.34× faster than the HPO method on single-label datasets and 6.52× on multi-label datasets, which indicates the high efficiency of the dynamic adjustment of K.

Additionally, we also compare the GPU memory consumption between DisenGCN and our model in Table 6. It is observed that D^2-GCN occupies less GPU memory than DisenGCN in training on each dataset. The reason behind this is because we set K to 1 on all datasets, which directly affects the change of $d_{in}^{(m)}$ (defined in Eq. (12)) or $d_{in}^{(m,l)}$ (defined in Eq. (13)) in training.

As for the classification performance, compared with the best accuracy result obtained by DisenGCN$_{HPO}$ in all its trials, our model is still on average 2.48% higher in single-label

Table 5 Comparison of total time cost between DisenGCN$_{HPO}$ and D^2-GCN (unit: second)

Model	Dataset						
	Cora	Citeseer	PubMed	PPI	POS	BlogCatalog	Flickr
DisenGCN$_{HPO}$	81.42	79.07	445.29	2547.29	2735.17	8724.36	35672.13
D^2-GCN	23.04	18.21	237.24	470.03	626.39	1337.25	3917.56

Table 6 Comparison of memory consumption between DisenGCN and D^2-GCN (unit: MB)

Model	Dataset						
	Cora	Citeseer	PubMed	PPI	POS	BlogCatalog	Flickr
DisenGCN	2066	2186	7852	2166	2316	4488	34928
D^2-GCN	1938	2104	2884	1614	1534	1706	14772

tasks and 3.17% in multi-label ones. Note that the best node classification results of DisenGCN on different datasets are already implied in Tables 3 and 4.

5.6 Sensitivity analysis

In this subsection, we would investigate the impacts of the length of the epoch stride defined in the epoch-level disentanglement (i.e., λ) and the scale of layer-by-layer growth of K defined in the layer-level disentanglement (i.e., ΔK) on the model performance, respectively.

5.6.1 Epoch-level sensitivity

As shown in Eq. (11), λ is utilized to control the update frequency of K across training epochs. We conduct experiments on Cora and record in Fig. 11 the results of test accuracies (Fig. 11(a)), change of $\tilde{K}_{val}$ (Fig. 11(b)), and change of KL_{val} (Fig. 11(c)).

It is observed in Fig. 11(a) that the relationship between λ and test accuracy could be approximated by a globally convex mapping, which implies the good convergence of the dynamic adjustment of K at epoch level. Note that although our model (D^2-GCN) would perform worse than DisenGCN with some λ, Fig. 11(a) demonstrates that D^2-GCN has the potential to enhance the effectiveness of disentangling-based methods to a higher degree.

It also indicates in Figs. 11(b) and 11(c) that both the weighted average number of predicted types of all nodes' neighbors (i.e., $\tilde{K}_{val}$) and the updated K (i.e., KL_{val}) tend to converge throughout training with different settings of λ. Besides, bigger λ would exhibit better convergence results.

5.6.2 Layer-level sensitivity

As shown in Equation 14, ΔK is utilized to control the update scale of K across graph convolutional layers. We conduct experiments on Cora and record in Fig. 12 the results of test accuracies (Fig. 12(a)), change of $\tilde{K}_{val}$ (Fig. 12(b)), and change of KL_{val} (Fig. 12(c)).

Similar to the relationship between λ and test accuracy in Fig. 11(a), Fig. 12(a) demonstrates that the expected relationship between ΔK and test accuracy could be also approximated by a globally convex mapping, which implies the good convergence of the dynamic adjustment of K at layer level. Like in epoch-level disentanglement, Figs. 12(b) and 12(c) indicate that both the weighted average number of predicted types of all nodes' neighbors (i.e., $\tilde{K}_{val}$) and the updated K (i.e., KL_{val}) tend to converge throughout training under different ΔK settings. Besides, bigger ΔK would present better convergence results.

5.7 Ablation study

In Section 3.2, we propose disentanglement at two levels, i.e., the epoch level and the graph convolutional layer (GCL) level. Here we design a series of ablation experiments to explore the impacts of these two disentangling modules on D^2-GCN's

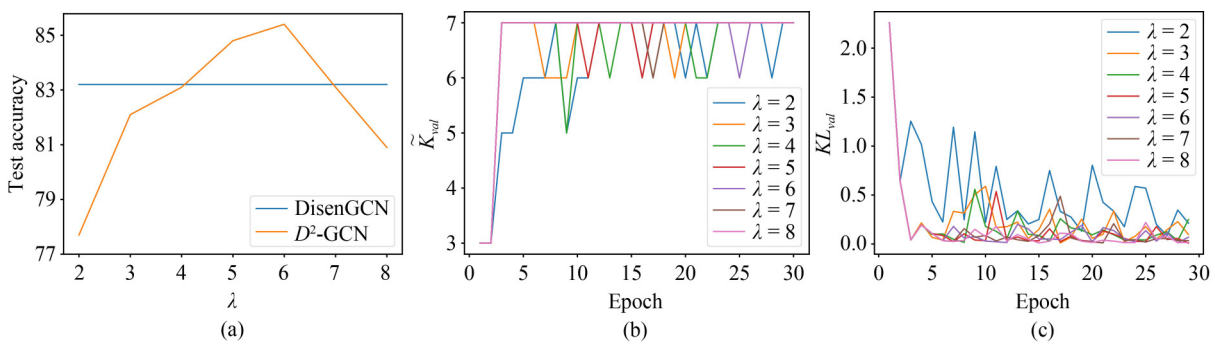


Fig. 11 Effects of λ on Cora. (a) Test accuracy; (b) $\tilde{K}_{val}$; (c) KL_{val}

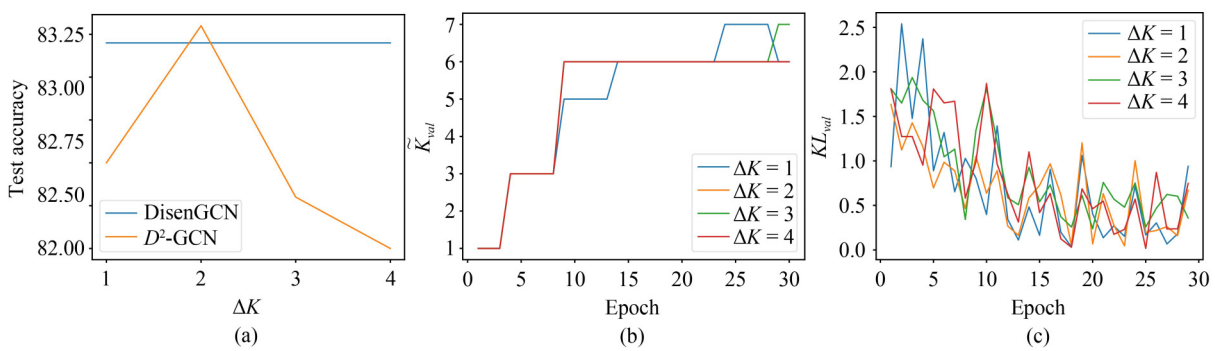


Fig. 12 Effects of ΔK on Cora. (a) Test accuracy; (b) $\tilde{K}_{val}$; (c) KL_{val}

node classification performance. We record the classification results on datasets with single-label or multi-label nodes in Tables 7 and 8, respectively.

In these two tables, "ED" (or "LD") represents disentanglement at the epoch (or GCL) level, "Fd" implies fixing the dimension of each node's embedding $d_{in}^{(m)}$ (or $d_{in}^{(m,l)}$) in all hidden layers, and "$F\Delta d$" implies fixing each disentangled feature channel's dimension Δd. It is observed from Tables 7 and 8 that graphs in single-label datasets are the most suitable for epoch-level disentanglement, while D^2-GCN performs the best with the disentanglement at both the epoch level and the GCL level on multi-label datasets. This complexity arises from the node representation of each node in multi-label datasets being more intricate than that in single-label ones. Consequently, a more sophisticated dynamic disentanglement is needed on graphs with multi-label nodes.

5.8 Visualization

To qualitatively compare the node classification performance between our model and DisenGCN, we utilize t-SNE [47] to visualize the clustering results on Cora, Citeseer, and PubMed. Specifically, we use the output embeddings derived from the last layer to plot the label distribution on the test set.

Table 7 Test accuracy results of D^2-GCN with different disentangling strategies in single-label tasks (unit: percentage)

Model	Dataset		
	Cora	Citeseer	PubMed
D^2-GCN$_{(ED,Fd)}$	**85.4±0.11**	**73.5±0.12**	**81.8±0.14**
D^2-GCN$_{(ED,F\Delta d)}$	84.8±0.23	72.7±0.18	81.3±0.16
D^2-GCN$_{(LD,Fd)}$	82.7±0.15	72.2±0.22	81.0±0.13
D^2-GCN$_{(LD,F\Delta d)}$	82.6±0.17	71.8±0.12	80.8±0.21
D^2-GCN$_{(ED+LD,Fd)}$	83.3±0.11	73.2±0.24	81.2±0.17
D^2-GCN$_{(ED+LD,F\Delta d)}$	83.1±0.14	73.0±0.16	80.9±0.22

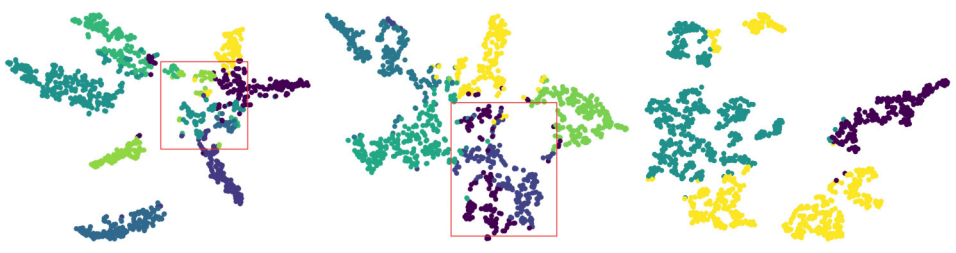
Table 8 Test Macro-F1 and Micro-F1 scores of D^2-GCN with different disentangling strategies in multi-label tasks (unit: percentage)

	Model	Dataset			
		PPI	POS	BlogCatalog	Flickr
Macro-F1	D^2-GCN$_{(ED,Fd)}$	21.3±0.16	26.5±0.12	28.4±0.15	34.0±0.17
	D^2-GCN$_{(ED,F\Delta d)}$	21.5±0.12	26.8±0.14	28.5±0.11	34.2±0.16
	D^2-GCN$_{(LD,Fd)}$	21.6±0.14	27.1±0.17	28.7±0.13	34.7±0.11
	D^2-GCN$_{(LD,F\Delta d)}$	21.8±0.11	27.2±0.15	28.8±0.12	34.9±0.13
	D^2-GCN$_{(ED+LD,Fd)}$	21.9±0.13	27.4±0.16	29.1±0.12	35.4±0.15
	D^2-GCN$_{(ED+LD,F\Delta d)}$	**22.3±0.11**	**27.8±0.14**	**29.6±0.12**	**35.7±0.17**
Micro-F1	D^2-GCN$_{(ED,Fd)}$	25.5±0.13	53.8±0.11	41.5±0.10	49.6±0.14
	D^2-GCN$_{(ED,F\Delta d)}$	25.6±0.14	54.0±0.12	41.6±0.11	49.8±0.17
	D^2-GCN$_{(LD,Fd)}$	25.8±0.12	54.3±0.13	41.8±0.15	49.9±0.10
	D^2-GCN$_{(LD,F\Delta d)}$	25.9±0.09	54.5±0.11	42.1±0.14	50.2±0.16
	D^2-GCN$_{(ED+LD,Fd)}$	26.1±0.11	54.7±0.15	42.3±0.13	50.6±0.12
	D^2-GCN$_{(ED+LD,F\Delta d)}$	**26.6±0.15**	**55.4±0.11**	**42.7±0.14**	**50.9±0.15**

It is shown in Figs. 13 and 14 that D^2-GCN exhibits clearer classification boundaries and higher intra-class similarity than DisenGCN on different datasets, which demonstrates that the dynamic adjustment of K in D^2-GCN indeed helps enhance the disentanglement of node features regarding the node classification performance and the mutual independence among different disentangled channels.

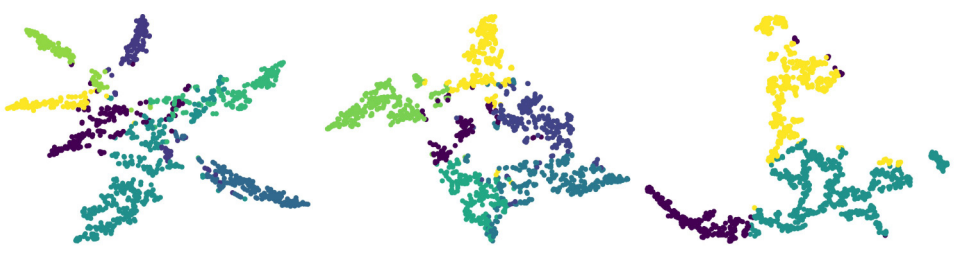
6 Related work

Inspired by the huge success of convolutional networks in the image processing domain, there has been a surge in studying to apply convolution operations on non-grid graph data. These approaches are known as Graph Convolutional Networks (GCNs) [1,44,48,49]. Existing GCNs commonly fall into two categories: spectral-based GCNs and spatial-based GCNs.



(a) (b) (c)

Fig. 13 Visualizations of DisenGCN's node classification performance on the test set in Cora, Citeseer, and PubMed. The rectangles with red borders are used to highlight the ambiguous boundaries among different node classes. (a) DisenGCN on Cora; (b) DisenGCN on Citeseer; (c) DisenGCN on PubMed



(a) (b) (c)

Fig. 14 Visualizations of D^2-GCN's node classification performance on the test set in Cora, Citeseer, and PubMed. (a) D^2-GCN on Cora; (b) D^2-GCN on Citeseer; (c) D^2-GCN on PubMed

Bruna [48] proposed the first well-known work on GCNs, which designed a variant of graph convolution based on spectral graph theory. Later, there comes out a surge of interest in the extensions and optimizations on spectral-based GCNs [1,44,49].

Since spectral GCNs use the whole graph to learn graph representation in each iteration, it is difficult for them to be scaled to large graphs. Thus spatial-based GCNs have been rapidly studied in recent years [45,50,51]. These methods stack multiple graph convolutional layers to learn each node's representation by aggregating its neighbors' features.

However, prevalent GCN models often consider the neighborhood of each node as a whole and neglect the underlying factors that may cause the various correlations among neighbors. In the last decade, disentangled representation learning has seen considerable development (particularly in the computer vision field [16]), which separates the latent explanatory factors behind the features of nodes. This kind of representation is capable of improving the generalization capability and robustness of the model against adversarial attacks.

In 2019, DisenGCN [18] was the first to propose disentangled node representation on graph data. It was inspired by the dynamic routing approach in CapsNet [52]. The novel neighborhood routing mechanism in DisenGCN aimed to cluster the neighbors of a node by disentangling the latent factors of the connections within the neighborhood. However, it does not further explore the adaptability of the disentanglement over training, which motivates us to design an automatic mechanism to help find the best partition of each node's neighborhood on a particular graph dataset. We then put forward a dynamic method for continuously adjusting the number of disentangled channels of each node's features within one full training. As a result, we achieve SOTA classification performance under the same searching space as in the statically disentangling way.

In recent years, disentangled graph representations start to gain research interests in various downstream tasks, such as skeleton-based action recognition [53], collaborative filtering based recommendation [54], point-of-interest recommendation [55], click-through rate prediction [56], citation generation [57], knowledge graph completion [58], pedestrian trajectory prediction [59], and phase grounding [60]. We would expand our proposed dynamic disentangling mechanism to various domains in the future.

7 Conclusion

We propose D^2-GCN to dynamically refine the disentanglement of each node's multi-dimension features, which helps improve the local representation of a graph. Specifically, we propose two levels of disentanglement to improve our model's adaptivity on different datasets. We also theoretically prove the good convergence of our method. Besides, D^2-GCN presents high search efficiency and outperforms the baselines in node classification tasks. In future work, we would combine the subgraph theory to develop our model's capability in grasping long-distance dependencies in a graph.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant Nos. 62141214 and 62272171).

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

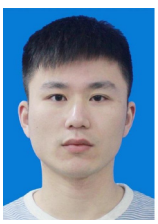
References

1. Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations. 2017
2. Rong Y, Huang W, Xu T, Huang J. DropEdge: towards deep graph convolutional networks on node classification. In: Proceedings of the 8th International Conference on Learning Representations. 2020
3. Zhang M, Chen Y. Link prediction based on graph neural networks. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, 5171−5181
4. Yun S, Kim S, Lee J, Kang J, Kim H J. Neo-GNNs: neighborhood overlap-aware graph neural networks for link prediction. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. 2021, 13683−13694
5. Zhang M, Cui Z, Neumann M, Chen Y. An end-to-end deep learning architecture for graph classification. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence. 2018, 4438−4445
6. Yang Y, Feng Z, Song M, Wang X. Factorizable graph convolutional networks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020
7. Wu S, Xiong Y, Weng C. Dynamic depth-width optimization for capsule graph convolutional network. Frontiers of Computer Science, 2023, 17(6): 176346
8. Liu K, Sun X, Jia L, Ma J, Xing H, Wu J, Gao H, Sun Y, Boulnois F, Fan J. Chemi-Net: a molecular graph convolutional network for accurate drug property prediction. International Journal of Molecular Sciences, 2019, 20(14): 3389
9. Sun M, Zhao S, Gilvary C, Elemento O, Zhou J, Wang F. Graph convolutional networks for computational drug development and discovery. Briefings in Bioinformatics, 2020, 21(3): 919–935
10. Jin W, Stokes J M, Eastman R T, Itkin Z, Zakharov A V, Collins J J, Jaakkola T S, Barzilay R. Deep learning identifies synergistic drug combinations for treating COVID-19. Proceedings of the National Academy of Sciences of the United States of America, 2021, 118(39): e2105070118
11. Ying R, He R, Chen K, Eksombatchai P, Hamilton W L, Leskovec J. Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018, 974−983
12. Fan W, Ma Y, Li Q, He Y, Zhao E, Tang J, Yin D. Graph neural networks for social recommendation. In: Proceedings of the World Wide Web Conference. 2019, 417−426
13. He X, Deng K, Wang X, Li Y, Zhang Y, Wang M. LightGCN: simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020, 639−648
14. Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A. beta-VAE: learning basic visual concepts with a constrained variational framework. In: Proceedings of the 5th International Conference on Learning Representations. 2017
15. Song J, Chen Y, Ye J, Wang X, Shen C, Mao F, Song M. DEPARA: deep attribution graph for deep knowledge transferability. In: Proceedings of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, 3922−3930
16. Alemi A A, Fischer I, Dillon J V, Murphy K. Deep variational information bottleneck. In: Proceedings of the 5th International Conference on Learning Representations. 2017

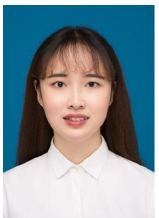
17. Lipton Z C. The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. Queue, 2018, 16(3): 31–57
18. Ma J, Cui P, Kuang K, Wang X, Zhu W. Disentangled graph convolutional networks. In: Proceedings of the 36th International Conference on Machine Learning. 2019, 4212−4221
19. Liu Y, Wang X, Wu S, Xiao Z. Independence promoted graph disentangled networks. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. 2020, 4916−4923
20. Sen P, Namata G, Bilgic M, Getoor L, Gallagher B, Eliassi-Rad T. Collective classification in network data. AI Magazine, 2008, 29(3): 93–106
21. Wasserman S, Faust K. Centrality and prestige. In: Wasserman S, Faust K. Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press, 1994, 169−219
22. Chan P K, Stolfo S J. Learning with non-uniform class and cost distributions: effects and a distributed multi-classifier approach. In: Proceedings of the Work Shop Notes KDD-98 Workshop on Distributed Data Mining. 1998, 1−9
23. Brodersen K H, Ong C S, Stephan K E, Buhmann J M. The balanced accuracy and its posterior distribution. In: Proceedings of the 20th International Conference on Pattern Recognition. 2010, 3121−3124
24. Luo W, Li Y, Urtasun R, Zemel R S. Understanding the effective receptive field in deep convolutional neural networks. In: Proceedings of the 29th Advances in Neural Information Processing Systems. 2016, 4898−4906
25. Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems. 2014, 3104−3112
26. Kullback S, Leibler R A. On information and sufficiency. The Annals of Mathematical Statistics, 1951, 22(1): 79–86
27. Shannon C E. A mathematical theory of communication. The Bell System Technical Journal, 1948, 27(3): 379–423
28. Breitkreutz B J, Stark C, Reguly T, Boucher L, Breitkreutz A, Livstone M, Oughtred R, Lackner D H, Bähler J, Wood V, Dolinski K, Tyers M. The BioGRID interaction database: 2008 update. Nucleic Acids Research, 2008, 36: D637–D640
29. Liberzon A, Subramanian A, Pinchback R, Thorvaldsdóttir H, Tamayo P, Mesirov J P. Molecular signatures database (MSigDB) 3. 0. Bioinformatics, 2011, 27(12): 1739–1740
30. Mahoney M. Large text compression benchmark. 2023
31. Toutanova K, Klein D, Manning C D, Singer Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the North American Chapter of the Association for Computational Linguistics. 2003, 252−259
32. Tang L, Liu H. Relational learning via latent social dimensions. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2009, 817−826
33. McAuley J, Leskovec J. Image labeling on a network: using social-network metadata for image classification. In: Proceedings of the 12th European Conference on Computer Vision. 2012, 828−841
34. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: Proceedings of the 6th International Conference on Learning Representations. 2018
35. Perozzi B, Al-Rfou R, Skiena S. DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014, 701−710
36. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q. LINE: large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web. 2015, 1067−1077
37. Grover A, Leskovec J. node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016, 855−864
38. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 1958, 65(6): 386–408
39. Belkin M, Niyogi P, Sindhwani V. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research, 2006, 7: 2399–2434
40. Weston J, Ratle F, Mobahi H, Collobert R. Deep learning via semi-supervised embedding. In: Montavon G, Orr G B, Müller K R. Neural Networks: Tricks of the Trade. Berlin, Heidelberg: Springer, 2012
41. Zhu X, Ghahramani Z, Lafferty J. Semi-supervised learning using Gaussian fields and harmonic functions. In: Proceedings of the 20th International Conference on Machine Learning. 2003, 912−919
42. Lu Q, Getoor L. Link-based classification. In: Proceedings of the 20th International Conference on Machine Learning. 2003, 496−503
43. Yang Z, Cohen W, Salakhutdinov R. Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on Machine Learning. 2016, 40−48
44. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. 2016, 3844−3852
45. Monti F, Boscaini D, Masci J, Rodolà E, Svoboda J, Bronstein M M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In: Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition. 2017, 5115−5124
46. Abu-El-Haija S, Perozzi B, Kapoor A, Alipourfard N, Lerman K, Harutyunyan H, Ver Steeg G, Galstyan A. MixHop: higher-order graph convolutional architectures via sparsified neighborhood mixing. In: Proceedings of the 36th International Conference on Machine Learning. 2019, 21−29
47. van der Maaten L, Hinton G. Visualizing data using t-SNE. Journal of Machine Learning Research, 2008, 9(86): 2579–2605
48. Bruna J, Zaremba W, Szlam A, LeCun Y. Spectral networks and locally connected networks on graphs. In: Proceedings of the 2nd International Conference on Learning Representations. 2014
49. Levie R, Monti F, Bresson X, Bronstein M M. CayleyNets: graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing, 2019, 67(1): 97–109
50. Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 1025−1035
51. Gao H, Wang Z, Ji S. Large-scale learnable graph convolutional networks. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018, 1416−1424
52. Hinton G E, Krizhevsky A, Wang S D. Transforming auto-encoders. In: Proceedings of the 21st International Conference on Artificial Neural Networks. 2011, 44−51
53. Liu Z, Zhang H, Chen Z, Wang Z, Ouyang W. Disentangling and unifying graph convolutions for skeleton-based action recognition. In: Proceedings of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, 143−152
54. Wang Y, Tang S, Lei Y, Song W, Wang S, Zhang M. DisenHAN: disentangled heterogeneous graph attention network for recommendation. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. 2020, 1605−1614
55. Qin Y, Wang Y, Sun F, Ju W, Hou X, Wang Z, Cheng J, Lei J, Zhang M. DisenPOI: disentangling sequential and geographical influence for point-of-interest recommendation. In: Proceedings of the 16th ACM International Conference on Web Search and Data Mining. 2023, 508−516
56. Wang Y, Qin Y, Sun F, Zhang B, Hou X, Hu K, Cheng J, Lei J, Zhang

M. DisenCTR: dynamic graph-based disentangled representation for click-through rate prediction. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2022

57. Wang Y, Song Y, Li S, Cheng C, Ju W, Zhang M, Wang S. DisenCite: graph-based disentangled representation learning for context-specific citation generation. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence. 2022, 11449−11458
58. Wu J, Shi W, Cao X, Chen J, Lei W, Zhang F, Wu W, He X. DisenKGAT: knowledge graph embedding with disentangled graph attention network. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 2021, 2140−2149
59. Bae I, Jeon H G. Disentangled multi-relational graph convolutional network for pedestrian trajectory prediction. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. 2021, 911−919
60. Mu Z, Tang S, Tan J, Yu Q, Zhuang Y. Disentangled motif-aware graph learning for phrase grounding. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. 2021, 13587−13594



Shangwei Wu is currently a PhD candidate at East China Normal University (ECNU), China. His research interests include graph neural networks and deep learning systems.



Yingtong Xiong received the ME degree at East China Normal University (ECNU), China in 2023. Her research interests include graph neural networks and deep learning systems.

Hui Liang is pursuing the ME degree at East China Normal University (ECNU), China. His research interests include graph neural networks and deep learning systems.

Chuliang Weng received the PhD degree at Shanghai Jiao Tong University (SJTU), China in 2004. He is currently a full professor at East China Normal University (ECNU), China. Before joining ECNU, he was an associate professor at SJTU and later worked at Huawei Central Research Institute, China. He was also a visiting research scientist at Columbia University, USA. His research interests include parallel and distributed systems, system virtualization and cloud computing, storage systems, operating systems, and system security.