Exploring Edge Disentanglement for Node Classification

Tianxiang Zhao, Xiang Zhang, Suhang Wang
College of Information Sciences and Technology, The Pennsylvania State University
State College, PA, USA
{tkz5084,xzz89,szw494}@psu.edu

ABSTRACT

Edges in real-world graphs are typically formed by a variety of factors and carry diverse relation semantics. For example, connections in a social network could indicate friendship, being colleagues, or living in the same neighborhood. However, these latent factors are usually concealed behind mere edge existence due to the data collection and graph formation processes. Despite rapid developments in graph learning over these years, most models take a holistic approach and treat all edges as equal. One major difficulty in disentangling edges is the lack of explicit supervisions. In this work, with close examination of edge patterns, we propose three heuristics and design three corresponding pretext tasks to guide the automatic edge disentanglement. Concretely, these self-supervision tasks are enforced on a designed edge disentanglement module to be trained jointly with the downstream node classification task to encourage automatic edge disentanglement. Channels of the disentanglement module are expected to capture distinguishable relations and neighborhood interactions, and outputs from them are aggregated as node representations. The proposed DisGNN is easy to be incorporated with various neural architectures, and we conduct experiments on 6 real-world datasets. Empirical results show that it can achieve significant performance gains.

CCS CONCEPTS

• **Computing methodologies** → *Regularization*; **Unsupervised learning**; **Semi-supervised learning settings**.

KEYWORDS

graph neural networks, graph disentanglement, node classification

ACM Reference Format:
Tianxiang Zhao, Xiang Zhang, Suhang Wang. 2022. Exploring Edge Disentanglement for Node Classification . In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3511929

1 INTRODUCTION

In recent years, learning from graph-structured data is receiving a growing amount of attention due to the ubiquity of this data form in the world. For example, social networks [9, 39], molecular structures [5, 22] and knowledge graphs [27] all require utilizing the rich connection information among nodes. Graph neural networks (GNNs) [31] provide a general and effective framework for mining from graphs, and is developing rapidly among the years.

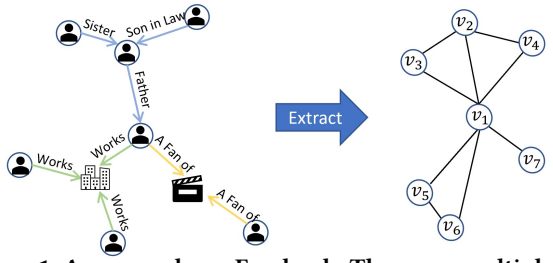


Figure 1: An example on Facebook. There are multiple possible relations behind edge existence, but such information is usually latent and unavailable during graph formation.

GNNs can be viewed as a message-passing network [10], which conduct node representation updating and neighborhood interaction modeling iteratively. Variants of GNNs [11, 17, 34] mainly differ in the proposed mechanism of aggregating messages from node neighborhoods. However, despite improved representation power from architecture designs, most of them adopt a uniform processing on edges in the input graph. Their model is designed under the assumption that all edges are generated under the same latent distribution and indicate the same relationship, hence those edges are taken as equal. Yet real-world graphs are usually formed via a complex and heterogeneous process, with multiple possible causes behind existence of edges. For example, as shown in Figure 1, connection in a social network could be resulted from similarity in interest, colleague relationship, or living in the same neighborhood, etc. Neglect of semantic differences carried by edges would harm the modeling of neighborhood interactions and the quality of obtained node embeddings. This observation motivates us to utilize each edge with considerations on its carried semantic relationship.

However, incorporating edge disentanglement into graph learning is a difficult task. The major problem is the lack of supervision signals to guide the learning process. Due to limitations in data collection and graph formation procedures, most graph datasets have plain edges without attributes, making it difficult to obtain disentangled edge set. There are few pioneering works [21, 35] in automatic disentanglement of graphs. However, Ma et al. [21] focuses on obtaining node-level representations w.r.t disentangled latent factors and pays little attention to latent multi-relations of edges, while Yang et al. [35] mainly focuses on graph-level tasks and proposes to disentangle the input graph into multiple factorized subgraphs. Furthermore, none of these methods consider the problem of providing extra supervision signals for edge disentanglement, and rely upon the holistic learning process instead.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3511929

Targeting at this problem, in this work, we focus on facilitating graph learning, particularly the node classification task, with automatic edge disentanglement. We propose an edge disentanglement module, and make the first attempt in designing a set of pretext tasks to guide the edge disentanglement. It is a difficult task, because the latent multi-relations behind edge existence are usually hidden and unavailable in real-world datasets. With a close examination of heuristics, we propose three self-supervision signals: (1) The union of disentangled edges should recover the original input edge set, (2) Edges carrying homophily information (homo-edges) models intra-class interactions while edges carrying heterophily property (hetero-edges) models inter-class interactions, hence they usually carry different semantics, (3) The edge patterns and interactions captured by different channels of DisGNN (each represents a disentangled relation group) should be discrepant, with a clear difference in distribution.

Concretely, the designed edge disentanglement module is incorporated into existing GNN layers, and three self-supervision tasks corresponding to aforementioned heuristics are implemented and applied to them. This approach, DisGNN, enables separate processing of disentangled edges during learning towards node classification via optimizing the disentanglement-eliciting pretext tasks jointly. The proposed method is evaluated on six real-world graphs, and we further conduct ablation studies to analyze its behaviors. Experimental results show that the proposed approach is effective and promising, achieving improvements on all datasets.

To summarize, in this work, we studies node classification with automatic edge disentanglement and make following contributions:

- **Node classification in the edge-disentanglement-aware manner.** Usually there are multiple latent factors behind the existence of edges between node pairs, and utilizing them could improve node embedding quality and help the node classification task. In this work, we study this problem by designing and applying an edge disentanglement module in together with GNN layers.
- **Automatic edge disentanglement with pretext tasks.** Although edges in real-world graphs usually contain a variety of different relations in nature, lack of supervision limits the discovering of them. In this work, we propose three self-supervision signals, which could encourage our designed module to automatically disentangling edges.
- **Evaluation on real-world graph sets.** We evaluate the proposed approach on six real-world datasets, and results show the improvement brought by the proposed approach.

2 RELATED WORK

2.1 Graph Neural Network

Graph neural networks (GNNs) are developing rapidly in recent years, with the increasing needs of learning on relational data structures [7, 9, 37, 38]. Generally, existing GNNs can be categorized into two categorizes, i.e., spectral-based approaches [4, 17, 28] based on graph signal processing theory, and spatial-based approaches [2, 8, 32] relying upon neighborhood aggregation. Despite their differences, most GNN variants can be summarized with the message-passing framework, which is composed of pattern extraction and interaction modeling within each layer [10].

Dedications have been made towards mining richer information from the provided relation topology. For example, Velickovic et al. [29] extends self-attention mechanism to enable learning the weights for nodes inside the neighborhood. Xu et al. [34] extends expressive power of GNNs to the same order of WL test by designing an injective neighborhood aggregator. Abu-El-Haija et al. [1] extracts multi-hop neighborhoods and learns to mix them for improving center node representations. Dai et al. [6] studies noisy information propagating along given edges and proposes to update network structure to improve robustness. However, all these methods are holistic approaches and neglect the latent factors behind edge existence. In real world cases, connections among entities are usually multi-relational in nature. Although they are given in the form of plain edges, different edges could be caused by different factors and represent different semantics. Disentangling these factors could enable us to utilize given edges with the awareness of their carried latent relations, and model richer node interactions.

Discovering latent factors behind graphs and utilizing them to improve graph learning is an important but under-exploited problem. Currently, works in this direction are rather limited. One major difficulty of this task is the lack of explicit supervisions on ground-truth disentanglement. For example, graph attention network (GAT) [29] offers a mechanism of specifying different weights to nodes in the neighborhood, and has the potential of disentangling relation types with multiple attention heads. However, analysis find that GAT tends to learn a restricted "static" form of attention and the patterns captured by different heads are limited [3, 16]. A pretext task with self-supervision on attention heads is designed in [16], but it only encourages recovering ground-truth edges as a whole and provides no signals on relation disentanglement. DisenGCN [21] makes the first attempt by learning node embeddings with respect to different factors with a neighborhood routing mechanism. IPGDN [20] further improves on that by utilizing Hilbert-Schmidt Independence Criterion to promote independence among disentangled embeddings. FactorGCN [35] studies the disentanglement of graphs into multiple sub-graphs mainly for graph-level tasks. Different from these methods, this work focuses on automatic disentanglement at the edge level, and designs three pretext tasks providing heuristics to guide the training process.

2.2 Self-supervision in Graph

Self-supervised learning targets at extracting informative knowledge through well-designed pretext tasks without relying on manual labels, and is able to utilize large amount of available unlabeled data samples. This framework has been shown to be promising in eliminating heavy label reliance and poor generalization performance of modern deep models. Besides its success in computer vision [14] and natural language processing [18], there is a trend of developing this idea in the graph learning domain [12, 19].

Rich node and edge features encoded in the graph are promising in providing self-supervision training signals to guide the graph learning process. Pretext tasks exploiting multiple different levels of graph topology have been designed to boost the performance of graph learning models. Existing approaches can be categorized into three types based on the designed pretext signals, including node-level signals [12], edge-level signals [16] and graph-level

signals [36]. Hu et al. [12] captures local node similarity via randomly masking nodes and learning to recover them from neighbors. [13] predicts pairwise node distance on graphs to encode global-structure information. Qiu et al. [25] uses contrastive learning on randomly sampled sub-graphs to highlight local-neighborhood similarities. In the classical semi-supervised setting on graphs, which contains a vast number of unlabeled nodes, these auxiliary tasks help to make learned representations more general, robust to noises, and more transferable [36]. A more complete survey of existing self-supervised learning on graphs can be found in [19, 33]

Unlike the aforementioned approaches, we target at designing pretext signals to guide the learning of attention distributions within the neighborhood. Kim and Oh [16] propose to supervise graph attention heads with real input edges, encouraging higher attention score between connected nodes while lower score between unconnected pairs. However, they aim to remove noises in edges and apply the same signal to all attention heads in the same manner. In contrast, we focus on disentangling relations and encouraging different attention heads to capture different semantics.

3 PROBLEM FORMULATION

We use $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{F}\}$ to denote an attributed network, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of n nodes. $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of $\mathcal{G}$. $\mathbf{F} \in \mathbb{R}^{n \times d}$ denotes the node attribute matrix, where $\mathbf{F}[j,:] \in \mathbb{R}^{1 \times d}$ is the d-dimensional node attributes of node v_j. In real-world, each edge is formed due to various reasons and carries rich semantic meanings. For example, in Facebook, v_i and v_j are linked because they are "collegaes"; v_i and v_k are connected because they are both interested in "Football". Hence, in learning the features of v_i, the utilization of v_j and v_k should be different in a relation-aware manner. However, for many real-world graphs such as Facebook, we only know that there exists such factors but the factors of each edge is not explicitly known. Thus, in this paper, we assume that we are only given $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{F}\}$ without edge attributes and aims to disentangle the graph to facilitate node classification.

As shown in [10], most existing GNN layers can be summarized in the following equations:

$$\begin{aligned} \mathbf{m}_v^{l+1} &= \sum_{u \in \mathcal{N}(v)} M_l(\mathbf{h}_v^l, \mathbf{h}_u^l, \mathbf{A}_{v,u}) \\ \mathbf{h}_v^{l+1} &= U_l(\mathbf{h}_v^l, \mathbf{m}_v^{l+1}) \end{aligned} \tag{1}$$

where $\mathcal{N}(v)$ is the set of neighbor of v in $\mathcal{G}$ and $\mathbf{h}_v^l$ denotes representation of v in the l-th layer. $\mathbf{A}_{v,u}$ represents the edge between v and u. M_l, U_l are the message function and update function respectively at layer l. However, this framework neglects diverse relations behind edges. With a disentangled adjacency matrix set $\mathcal{A}^{dis} = \{\mathbf{A}^0, \mathbf{A}^1, \ldots \mathbf{A}^m\}$, in which $\mathbf{A}^i \in \mathbb{R}^{n \times n}$ and $\mathbf{A}^i \subset \mathbf{A}$, we can obtain edge sets representing different relation semantics. Then, the GNN layer could be be changed into:

$$\begin{aligned} \mathbf{m}_v^{l+1,i} &= \sum_{u \in \mathcal{N}_i(v)} M_l^i(\mathbf{h}_v^l, \mathbf{h}_u^l, \mathbf{A}_{v,u}^i) \\ \mathbf{h}_v^{l+1} &= U_l(\mathbf{h}_v^{l+1}, \mathbf{m}_v^{l+1,0}, \ldots, \mathbf{m}_v^{l+1,m}) \end{aligned} \tag{2}$$

where $\mathcal{N}_i(v)$ is the set of neighbors of v in adjacency matrix $\mathbf{A}^i$. This disentangled GNN enables heterogeneous processing of given edges, and may produce more discriminant node embeddings for downstream tasks.

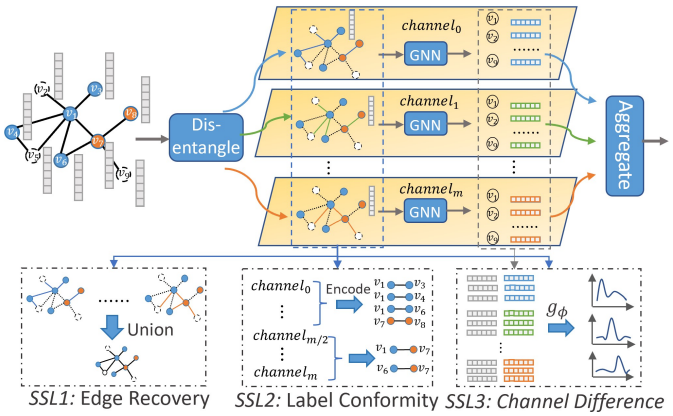


Figure 2: Overview of the DisGNN layer.

In this work, we focus on semi-supervised node classification task. $\mathbf{Y} \in \mathbb{R}^n$ is the class information for nodes in $\mathcal{G}$. During training, only a subset of $\mathbf{Y}$ is available, and the task is to train a classifier for unlabeled nodes. We use $\mathcal{V}_L$ and $\mathbf{Y}_L$ to represent the set of supervised nodes and their corresponding labels respectively.

Given $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{F}\}$, and labels $\mathbf{Y}_L$ for a subset of nodes $\mathcal{V}_L$, we aim to learn a node classifier f. f should learn to disentangle relations behind edge existence as $\mathcal{A}^{dis}$ and model node interactions accordingly during the label prediction process:

$$f(\mathcal{V}, \mathbf{A}, \mathbf{F}) \rightarrow \mathbf{Y} \tag{3}$$

4 METHODOLOGY

In this work, we implement the model as composing of two parts: a GNN-based feature extractor for obtaining node representations and a MLP-based classifier for node classification. θ, ξ are used to represent parameters of these two components respectively. The details of them will be introduced below.

4.1 Model Architecture

4.1.1 Feature Extractor. The feature extractor is adopted to encode graph structures and obtain node embeddings for classification. Following existing works [38, 40], it can be implemented by stacking two GNN layers. In this work, we design and utilize two DisGNN layers. Each DisGNN layer is implemented with the edge disentanglement module and GNN variants following Equation 2, upon which we apply our proposed self-supervision tasks. An overview of the proposed DisGNN layer is shown in Figure 2. The feature extractor is optimized with gradients from both node classification and self-supervision tasks, and its details will be presented in the next section. The parameter set of this extractor is denoted as θ.

4.1.2 Classifier. Based on the extracted node representation $\mathbf{h}_v$ for node v by feature extractor, a classifier is trained to perform node classification. Specifically, we use a 2-layer MLP:

$$\mathbf{P}_v = softmax\big(\mathbf{W}_2^{cls} \cdot \sigma(\mathbf{W}_1^{cls} \cdot \mathbf{h}_v)\big), \tag{4}$$

where $\mathbf{W}_1^{cls}, \mathbf{W}_2^{cls}$ are the parameter matrices of this classifier head. And the training objective $\mathcal{L}_{node}$ on this task would be:

$$\min_{\theta,\xi} - \sum_{v \in \mathcal{V}_{sup}} \sum_{c \in |Y|} Y_v^c \cdot log(\mathbf{P}_v^c). \tag{5}$$

ξ denotes parameters in this classification head, containing $\mathbf{W}_1^{cls}$ and $\mathbf{W}_2^{cls}$. $\mathcal{V}_{sup}$ represents supervised nodes during training, Y_v^c indicates whether node v belongs to class c, $|Y|$ represents the set of node classes, and $\mathbf{P}_v^c$ is the predicted probability of v in class c.

4.2 DisGNN Layer

4.2.1 Edge Disentanglement Module. With the input of graph $\mathcal{G}$, this module is designed to obtain a set of disentangled edge distributions, $\mathcal{A}^{dis}$. As edges are unattributed and binary in most graphs, we analyze the edge property through comparing the node pair connected by it. One module with $m+1$ channels can be constructed, with each channel capturing the dynamics reflecting one different group of relations. Each relation-modeling channel can be constructed in various ways, like dot-production, linear layer, neural network, etc.

In this work, to keep both simplicity and representation ability, we implement each relation-modeling channel as a MLP. Concretely, inside the l-th DisGNN layer, predicted edge between node pair v and u generated by the i-th channel can be written as:

$$\begin{aligned} \hat{e}_{v,u}^{l,i} &= sigmoid\big(\mathbf{W}_2^{l,i} LeakyReLU\big(\mathbf{W}_1^{l,i} \cdot [\mathbf{h}_v^l, \mathbf{h}_u^l]\big)\big) \\ \mathbf{A}_{v,u}^{l,i} &= \frac{exp(\hat{e}_{v,u}^{l,i})}{\sum_{w \in \mathcal{V}} exp(\hat{e}_{v,w}^{l,i})} \end{aligned} \tag{6}$$

In this equation, $\mathbf{h}_u^l$ corresponds to the extracted embedding of node u as input to layer l, and $\mathbf{h}_u^l, \mathbf{h}_v^l$ are concatenated as input. $\mathbf{W}_1^{l,i}, \mathbf{W}_2^{l,i}$ are the parameter matrices, and $\hat{e}_{v,u}^{l,i}$ is the edge existence probability between v and u modeled by channel i in the l-th layer. $\hat{e}$ is only calculated for connected node pairs to reduce computation complexity and avoid introducing noises. Following the pre-processing strategy in [17], edge weights in $\mathbf{A}^{l,i}$ are normalized before using as input into GNN layers.

4.2.2 Incorporating into GNN Layer. The edge disentanglement module is easy to be incorporated into various modern GNN layers through taking each disentangled edge group as a channel for modeling neighborhood interactions following Equation 2. In this subsection, we take the GNN layer proposed in [29] as an example. The process of update node representations on each channel can be written as:

$$\mathbf{h}_v^{l+1,i} = \sum_{u \in \mathcal{V}} \mathbf{A}_{u,v}^{l,i} \cdot \mathbf{W}_{feat}^{l,i} \mathbf{h}_u^l, \tag{7}$$

in which $\mathbf{W}_{feat}^{l,i}$ is the parameter matrix for learning embedding features of i-th channel in the l-th layer.

Then, we aggregate representation learned on each channel via concatenation and a 1-layer neural network:

$$\mathbf{h}_v^{l+1} = \sigma\big(\mathbf{W}_{agg}^l([\mathbf{h}_v^{l+1,0}, \mathbf{h}_v^{l+1,1}, \ldots, \mathbf{h}_v^{l+1,m}])\big). \tag{8}$$

σ represents nonlinear activation function, and we implement it as LeakyRelu. $\mathbf{W}_{agg}^l$ is a weight matrix used to fuse representations learned by each channel. Note that it is not limited to using this specific GNN variant, and we further test it on other typical GNN layers in the experiments.

4.3 Self-supervision for Disentanglement

One major difficulty in disentangling factors behind edges is the lack of supervision signals. Although it is often the case that edges contain multiple different relations and carry different semantics in a graph, it is difficult to collect ground-truth edge disentanglement as the supervision, as the example in Figure 1. Hence, in this part, we introduce how we manage to discover them automatically with carefully-designed self-supervisions, which also makes up the main contribution of this work. An overview is provided in Figure 2.

4.3.1 Signal 1: Edge Recovery. One most direct way of providing prior knowledge to guide the learning of relation distribution is utilizing the input edges. If node u and v are linked, then certain relationship exists between them, and such relation should be captured by the edge disentanglement module. Note that channels of it are expected to capture different relations, hence we require the union of them to successfully recover given edges. Specifically, based on the nonlinear property of *sigmoid* activation function, we propose to implement this pretext task on disentanglement channels via:

$$\bar{e}_{v,u}^l = sigmoid\Big(\sum_{i=0}^{m} \mathbf{W}_2^{l,i} LeakyReLU\big(\mathbf{W}_1^{l,i} \cdot [\mathbf{h}_v^l, \mathbf{h}_u^l]\big)\Big) \tag{9}$$

$\bar{e}_{v,u}^l$ represents the union of predicted edges between u and v at layer l, and supervision can be applied on top of it. Usually, the number of nodes is large with high sparsity degrees, hence it is not efficient to use positive and negative node pairs all at once. Addressing it, we sample a batch of positive node pairs $E_p \in \{(v,u) \mid \mathbf{A}_{v,u} \neq 0\}$ and negative node pairs $E_n \in \{(v,u) \mid \mathbf{A}_{v,u} = 0\}$ each time as the supervision. The size of E_p is set as $p_e \cdot |\mathcal{A}^+|$, with p_e and $\mathcal{A}^+$ being the sampling ratio and the set of connected node pairs respectively. The size of E_n is set as $\min(p_e \cdot |\mathcal{A}^-|, 3 \cdot |E_p|)$, and $\mathcal{A}^-$ represents the set of unconnected node pairs. By default, p_e is set as 1.0.

Specifically, the self-supervision loss $\mathcal{L}_{edge}$ is implemented as:

$$\min_{\theta} \mathcal{L}_{edge} = -\frac{1}{|E_p \cup E_n|} \sum_l \Big(\sum_{(v,u) \in E_p} \log \bar{e}_{v,u}^l + \sum_{(v,u) \in E_n} \log(1 - \bar{e}_{v,u}^l) \Big) \tag{10}$$

4.3.2 Signal 2: Label Conformity. Another heuristics we propose is analyzing the conformity of labels between two nodes connected by edges. Due to the lack of explicit information, it is not straightforward to design direct supervisions on edge disentanglement. However, following observations in [40], we notice that given edges can be split into two groups: homo-edges modeling intra-class interactions and hetero-edges modeling inter-class interactions. These two groups usually carry different relations and can serve as a heuristic for automatic disentanglement. Basing on this observation, we can obtain two pseudo edge sets:

$$\begin{aligned} E^{homo} &= \{\mathbf{A}_{v,u} \mid \mathbf{A}_{v,u} \neq 0, Y_u = Y_v\} \\ E^{hetero} &= \{\mathbf{A}_{v,u} \mid \mathbf{A}_{v,u} \neq 0, Y_u \neq Y_v\} \end{aligned} \tag{11}$$

As it is difficult to explicitly define the correspondence between edge sets and each disentanglement channel, we adopt a soft assignment,

assuming that the union of half channels capture an edge set:

$$\bar{e}_{v,u}^{l,homo} = sigmoid\Big(\sum_{i=0}^{m/2} \mathbf{W}_2^{l,i} LeakyReLU\big(\mathbf{W}_1^{l,i} \cdot [\mathbf{h}_v^l || \mathbf{h}_u^l]\big)\Big) \quad (12)$$

$$\bar{e}_{v,u}^{l,hetero} = sigmoid\Big(\sum_{i=m/2+1}^{m} \mathbf{W}_2^{l,i} LeakyReLU\big(\mathbf{W}_1^{l,i} \cdot [\mathbf{h}_v^l || \mathbf{h}_u^l]\big)\Big) \quad (13)$$

$\bar{e}_{v,u}^{l,homo}, \bar{e}_{v,u}^{l,hetero}$ are expected to capture homo-edges and hetero-edges respectively at layer l, and we supervise them in the same manner as Equation 14. With sampled positive set $E_p^{homo} \in E^{homo}$ and negative set $E_n^{homo} \in \mathbf{A} \setminus E^{homo}$, $\mathcal{L}_{homo}$ is implemented as:

$$\min_{\theta} -\frac{1}{|E_p^{homo} \cup E_n^{homo}|} \sum_{(v,u) \in E_p^{homo} \cup E_n^{homo}} \sum_{l} (\mathbf{1}((v,u) \in E_p^{homo}) \cdot log \bar{e}_{v,u}^{l,homo} + \mathbf{1}((v,u) \in E_n^{homo}) \cdot log(1 - \bar{e}_{v,u}^{l,homo})), \quad (14)$$

similarly is $\mathcal{L}_{hetero}$. The label conformity loss $\mathcal{L}_{conform}$ is implemented as:

$$\mathcal{L}_{conform} = \mathcal{L}_{homo} + \mathcal{L}_{hetero} \quad (15)$$

In experiments, we found that the amount of homo-edges and hetero-edges identified with given supervisions are sufficient for training. In cases when supervision is limited and cannot obtain a sufficient large set of labeled edges, a classifier can be trained to give them pseudo labels.

4.3.3 Signal 3: Channel Difference. The last signal we designed is to promote the difference in captured patterns across channels in edge disentanglement module, as relations embedded in well-disentangled edges should be distinguishable from each other. However, directly maximizing the difference of learned knowledge by each channel is a non-trivial task. Consider that as channels are expected to model different types of relations, the semantic information learned by them would also be different, which can be reflected by comparing node embedding changes produced by them. With this observation, we first obtain channel-wise node representations:

$$\tilde{\mathbf{h}}_{v,i}^l = Concat(\mathbf{h}_v^{l-1}, \mathbf{h}_v^{l,i}) \quad (16)$$

In the following part, we will omit layer number l and replace $\tilde{\mathbf{h}}_{v,i}^l$ with $\tilde{\mathbf{h}}_{v,i}$ for simplicity. $\tilde{\mathbf{h}}_{v,i}$ contains the semantics modeled by channel i. Then, we encourage distinguishability of these channel-wise representations by giving them pseudo labels w.r.t the channel.

Concretely, a label $\tilde{Y}_{v,i}$ is given to each $\tilde{\mathbf{h}}_{v,i}$, which is set as the index of channel $\tilde{Y}_{v,i} = i$. Then, a channel discriminator g is adopted to optimize this classification task. g is parameterized with ϕ:

$$\tilde{\mathbf{P}}_{v,i} = softmax\big(g_\phi(\tilde{\mathbf{h}}_{v,i})\big), \quad (17)$$

$$\min_{\theta,\phi} \mathcal{L}_{channel} = -\sum_{v \in \mathcal{V}} \sum_{i=0}^{m} \sum_{c=0}^{m} \mathbf{1}(\tilde{Y}_{v,i} = c) \log(\tilde{\mathbf{P}}_{v,i}[\tilde{Y}_{v,i}]). \quad (18)$$

$\tilde{\mathbf{P}}_{v,i} \in \mathbb{R}^{m+1}$ denotes the predicted class distribution for embedding $\tilde{\mathbf{h}}_{v,i}$, and $\mathcal{L}_{channel}$ is a cross entropy loss for embedding classification. g is implemented as a 2-layer MLP, and we train it end-to-end with feature extractor parameterized by θ. This task requires each channel to encode distinguishable semantics as well as being discriminative, hence is helpful for disentanglement.

Algorithm 1 Full Training Algorithm

Input: $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{F}\}$, node labels $\mathbf{Y}$, self-supervision weights $\lambda_1, \lambda_2, \lambda_3$, initial model parameter θ, ξ, ϕ, alternating step *n_step*

1: **while** Not Converged **do**
2: **for** *step* in *n_step* **do**
3: Update θ, ξ with gradients to minimize Loss $\mathcal{L}_{node}$;
4: **end for**
5: Input $\mathcal{G}$ to the feature extractor and classifier;
6: Use sampling to get training instances for self-supervision tasks;
7: Calculating $\mathcal{L}_{edge}, \mathcal{L}_{conform}$ on feature extractor;
8: Calculating $\mathcal{L}_{channel}$ with discriminator f_ϕ following Equation 18;
9: Calculating $\mathcal{L}_{node}$ following Equation 5;
10: Update θ, ξ, ϕ with gradients to minimize Loss $\mathcal{L}_{full}$
11: **end while**
12: **return** Learned model parameters θ, ξ

4.4 Optimization

In experiments, these self-supervisions are applied to all layers in feature extractor, and are trained in together with the node classification task. Putting everything together, the full training objective is implemented as follows:

$$\mathcal{L}_{full} = \mathcal{L}_{node} + \lambda_1 \mathcal{L}_{edge} + \lambda_2 \mathcal{L}_{conform} + \lambda_3 \mathcal{L}_{channel}, \quad (19)$$

in which $\lambda_1, \lambda_2, \lambda_3$ are weight of each pretext task respectively.

During training, we find that it would be more stable to alternatively optimize on this augmented objective and the vanilla node classification task. It can be seen as asynchronously adapt classifier to changes in edge disentanglement. We set the number of alternating steps as *n_step* and the full optimization algorithm is summarized in Algorithm 1. From line 2 to line 4, we perform conventional node classification to fine-tune the model. From line 5 to line 6 we process input for conducting pretext tasks. And the full loss is calculated and optimized from line 7 to line 10. In experiments, *n_step* is fixed as 5.

5 EXPERIMENT

In this section, we conduct a set of experiments to evaluate the benefits of proposed disentanglement-eliciting self-supervision signals. We apply them to the model introduced in Section 4, and test it on 6 real-world graph datasets. To evaluate its flexibility, we also incorporate the designed edge disentanglement module with other typical GNN layers to report the improvements. A set of sensitivity analysis is conducted on weight of self-supervision signals, and a case study on learned edge distribution is also provided. Particularly, we want to answer the following questions:

- **RQ1** Can the proposed edge disentanglement module with self-supervision signals improve the downstream task of node classification?

- **RQ2** Are the designed approach flexible to be applied in together with different GNN layer variants?
- **RQ3** Can the designed self-supervision tasks encourage edge disentanglement module to capture different relation patterns?

5.1 Experiment Settings

5.1.1 Dataset. We conduct experiments on 6 publicly-available graph datasets, and their details are given below:

- Cora: Cora is a citation network dataset for transductive learning setting. It contains one single large graph with 2, 708 papers from 7 areas. Each node has a 1433-dim attribution vector, and a total number of 13, 264 citation links exist in that graph.
- BlogCatalog: This is a social network dataset crawled from BlogCatalog[1], with 8, 652 bloggers from 38 classes and 501, 446 friendship edges. The dataset doesn't contain node attributes. Following [24], we attribute each node with a 64-dim embedding vector obtained from Deepwalk.
- Cora_full: A more complete version of Cora dataset. We filter out classes with less than 25 instances, and obtain a citation network of 70 classes with 19, 793 nodes and 146, 635 edges.
- Squirrel and Chameleon: These two datasets [23] are subgraphs containing web pages in Wikipedia discussing specific topics, and are typically used as graphs with certain degrees of heterophily [40]. Nodes represent web pages and edges are mutual links between pages. Nodes are categorized into 5 classes based on the amount of their average monthly traffic. Squirrel contains 5, 201 nodes and 401, 907 edges, while Chameleon contains 2, 277 nodes and 65, 019 edges.
- IMDB: A movie dataset collected by [30]. It contains 4, 780 nodes each representing a movie, and are divided into three classes (Action, Comedy, Drama) according to their genre. In total we have 21, 018 edges.

5.1.2 Baseline. First, we include representative models for node classification as baselines, which include:

- MLP. A MLP with one hidden layer is applied as the feature extractor. This baseline is implemented to show the node classification accuracy when relation information is not utilized;
- GCN. Two GCN layers is stacked as the feature extractor. GCN [17] is a popular spectral GNN variant based on graph Laplacian, and has been shown to perform well in graphs with high degree of homophily;
- GraphSage. Two GraphSage layers with mean aggregator are used as feature extractor. GraphSage is a spatial GNN variant introduced in [11];
- GIN [34]. It adopts an injective multiset function as the neighborhood aggregator, and is shown to have stronger representation ability than classical GNN layers. We stack two such layers as the feature extractor;
- FactorGCN [35] Similar with our motivation, FactorGCN also targets at disentangling graphs with repsect to latent factors. However, it is mainly designed for graph-level tasks, and has no self-supervisions to encourage edge disentanglement. We include it as a baseline by using it as the feature extractor.

[1]http://www.blogcatalog.com

We also select two GNN variants from heterophily graph domain, which are designed to model both inter-class and intra-class edges. These models may be better in capturing rich relation information, hence are also included as baselines:

- MixHop. MixHop [1] layer explicitly learns to mix feature representations of neighbors at various distances, and uses sparsity regularization to increase interpretability.
- H2GCN. A specially designed GNN layer proposed in [40] to work on graphs with high heterophily degrees. It enables more flexible feature aggregation from neighborhood.

To evaluate the effectiveness of proposed self-supervision signals, we also compare with the base architecture without self-supervisions, and denote it as DisGNN_Base.

5.1.3 Configurations. All experiments are conducted on a 64-bit machine with Nvidia GPU (Tesla V100, 1246MHz , 16 GB memory), and ADAM optimization algorithm is used to train the models. For all methods, the learning rate is initialized to 0.001, with weight decay being 5e-4. Besides, all models are trained until converging, with the maximum training epoch being 1000.

Alternating step *n_step* is fixed as 5, and $\{\lambda_1, \lambda_2, \lambda_3\}$ are set by grid search in $\{1e{-}4, 1e{-}2, 1, 10, 1e2\}$ for each dataset. Train:eval:test splits are set to 2:3:5. Unless specified otherwise, such configurations is adopted on all datasets throughout experiments.

5.1.4 Evaluation Metrics. Following existing works in evaluating node classification [15, 26], we adopt two criteria: classification accuracy(ACC), and macro-F score. ACC is computed on all testing examples at once, and F measure gives the harmonic mean of precision and recall for each class. We calculate F-measure separately for each class and then report their non-weighted average.

5.2 Node Classification Performance

To answer **RQ1**, in this section, we compare the performance on node classification between proposed DisGNN and all aforementioned baselines. Models are tested on 6 real-world datasets, and each experiment is conducted 3 times to alleviate the randomness. The average results with standard deviation are reported in Table 1 and Table 2. From the tables, we make the following observations:

- Our proposed DisGNN consistently outperforms baselines on all datasets with a clear margin. For example, DisGNN shows an improvement of 1.71 point in accuracy on BlogCatalog, and 1.41 point in accuracy on IMDB compared to the best-performed baseline respectively.
- Comparing with DisGNN_Base in which self-supervisions are not adopted, DisGNN also shows a consistent improvement. For example, in terms of accuracy, it improves for 3.67 point on BlogCatalog and 1.72 point on Chameleon.

To summarize, these results show the advantages of introduced edge disentanglement module and self-supervision signals in terms of node classification performance. It indicates that through automatic disentanglement and utilization of multi-relations behind edges, our approach is better at learning representations on graphs.

5.3 Combination with Different GNN Layers

To answer **RQ2**, we modify the architecture of DisGNN by replacing the message aggregation and node update mechanism in

Table 1: Comparison of different approaches on downstream task: node classification.

	Cora		BlogCatalog		Cora_full		IMDB	
Methods	ACC	F Score	ACC	F Score	ACC	F Score	ACC	F Score
MLP	$62.39_{\pm 0.54}$	$59.24_{\pm 0.40}$	$28.93_{\pm 0.14}$	$20.48_{\pm 0.19}$	$43.26_{\pm 0.12}$	$32.86_{\pm 0.05}$	$50.68_{\pm 0.40}$	$41.72_{\pm 0.31}$
GCN	$82.25_{\pm 0.19}$	$80.57_{\pm 0.26}$	$28.61_{\pm 0.28}$	$20.35_{\pm 0.49}$	$53.84_{\pm 0.18}$	$46.08_{\pm 0.31}$	$52.26_{\pm 0.08}$	$47.16_{\pm 0.36}$
GraphSage	$80.27_{\pm 0.52}$	$79.11_{\pm 0.48}$	$29.01_{\pm 0.21}$	$20.41_{\pm 0.06}$	$49.06_{\pm 0.08}$	$39.47_{\pm 0.32}$	$53.31_{\pm 0.06}$	$43.38_{\pm 0.14}$
GIN	$80.39_{\pm 0.08}$	$78.77_{\pm 0.13}$	$26.71_{\pm 1.13}$	$17.75_{\pm 0.48}$	$45.99_{\pm 0.10}$	$38.21_{\pm 0.42}$	$52.47_{\pm 0.35}$	$45.38_{\pm 0.46}$
FactorGCN	$82.14_{\pm 0.18}$	$80.39_{\pm 0.11}$	$25.35_{\pm 0.08}$	$17.40_{\pm 0.10}$	$44.08_{\pm 0.32}$	$36.60_{\pm 0.56}$	$54.52_{\pm 0.11}$	$47.16_{\pm 0.36}$
MixHop	$82.07_{\pm 0.17}$	$81.07_{\pm 0.56}$	$26.18_{\pm 0.69}$	$17.68_{\pm 0.94}$	$49.19_{\pm 0.16}$	$40.66_{\pm 0.27}$	$54.16_{\pm 0.20}$	$44.62_{\pm 0.25}$
H2GCN	$79.58_{\pm 0.23}$	$78.50_{\pm 0.14}$	$34.11_{\pm 0.25}$	$23.21_{\pm 0.80}$	$55.70_{\pm 0.03}$	$48.60_{\pm 0.07}$	$54.72_{\pm 0.23}$	$44.48_{\pm 0.57}$
DisGNN_Base	$81.81_{\pm 0.42}$	$80.47_{\pm 0.33}$	$32.15_{\pm 0.26}$	$23.22_{\pm 0.38}$	$57.20_{\pm 0.04}$	$50.23_{\pm 0.14}$	$54.98_{\pm 0.47}$	$47.02_{\pm 0.32}$
DisGNN	$\mathbf{83.16}_{\pm 0.48}$	$\mathbf{81.99}_{\pm 0.71}$	$\mathbf{35.82}_{\pm 0.19}$	$\mathbf{23.94}_{\pm 1.04}$	$\mathbf{58.83}_{\pm 0.17}$	$\mathbf{51.13}_{\pm 0.34}$	$\mathbf{56.39}_{\pm 0.14}$	$\mathbf{47.73}_{\pm 0.34}$

Table 2: Comparison of different approaches on downstream task: node classification on two heterophily graphs.

	Chameleon		Squirrel	
Methods	ACC	F Score	ACC	F Score
MLP	$35.71_{\pm 0.85}$	$34.59_{\pm 1.16}$	$25.16_{\pm 0.51}$	$25.17_{\pm 0.50}$
GCN	$63.17_{\pm 0.54}$	$63.04_{\pm 0.54}$	$51.69_{\pm 0.18}$	$51.50_{\pm 0.05}$
GraphSage	$46.45_{\pm 0.69}$	$46.33_{\pm 0.41}$	$32.34_{\pm 0.35}$	$32.53_{\pm 0.78}$
GIN	$41.50_{\pm 0.38}$	$41.21_{\pm 0.34}$	$33.20_{\pm 0.50}$	$31.67_{\pm 0.41}$
FactorGCN	$61.56_{\pm 0.21}$	$61.36_{\pm 0.20}$	$47.07_{\pm 0.50}$	$46.80_{\pm 0.56}$
MixHop	$53.63_{\pm 0.65}$	$53.40_{\pm 0.64}$	$39.72_{\pm 0.54}$	$39.63_{\pm 0.48}$
H2GCN	$63.48_{\pm 0.20}$	$63.37_{\pm 0.14}$	$49.81_{\pm 0.39}$	$49.90_{\pm 0.43}$
DisGNN_Base	$62.73_{\pm 1.15}$	$62.83_{\pm 0.60}$	$51.48_{\pm 0.40}$	$51.59_{\pm 0.43}$
DisGNN	$\mathbf{64.45}_{\pm 0.16}$	$\mathbf{64.39}_{\pm 0.16}$	$\mathbf{52.51}_{\pm 0.24}$	$\mathbf{52.35}_{\pm 0.26}$

Equation 7 with other GNN variants. Specifically, we test on GCN and GraphSage layers due to their popularity. Experiments are randomly conducted for 3 times on Cora, BlogCatalog, IMDB, and Squirrel. Results w/o SSL signals are both reported in Table 3.

From the result, a consistent improvement can be observed on these four dataset with both GCN and GraphSage layers. It is shown that the proposed approach is effective when incorporated into other GNN variants, validating the generality and advantage of proposed edge disentanglement.

Table 3: Node classification accuracy when incorporating proposed edge disentanglement with different GNNs.

	Cora	BlogCatalog	IMDB	Squirrel
GCN_Base	$82.21_{\pm 0.36}$	$33.74_{\pm 0.21}$	$55.23_{\pm 0.48}$	$51.42_{\pm 0.32}$
DisGNN-GCN	$83.89_{\pm 0.62}$	$35.09_{\pm 0.57}$	$57.06_{\pm 0.67}$	$52.62_{\pm 0.29}$
Sage_Base	$79.61_{\pm 0.06}$	$33.16_{\pm 0.35}$	$54.21_{\pm 0.19}$	$32.67_{\pm 0.36}$
DisGNN-Sage	$81.17_{\pm 0.18}$	$34.61_{\pm 0.44}$	$55.63_{\pm 0.27}$	$34.49_{\pm 0.27}$

5.4 Comparison with Different SSL Signals

To further evaluate the effectiveness of our proposed SSL signals, we compare it with three other typical self-supervision tasks: Masked attribute prediction [12], PairwiseDistance [13], and Context prediction [12]. These tasks are implemented on the DisGNN_Base, with the same optimization and configuration as DisGNN. Their weights are found via grid search in $\{1e-4, 1e-2, 1, 10, 1e2, 1e3\}$.

Comparisons are summarized in Table 4. We can observe that DisGNN is the most effective approach across these settings. This result validates our proposal of utilizing latent relations behind edges to improve the learning on graphs. Note that DisGNN can also be utilized in together with these SSL tasks, but we leave that for the future as it is not the focus of this work.

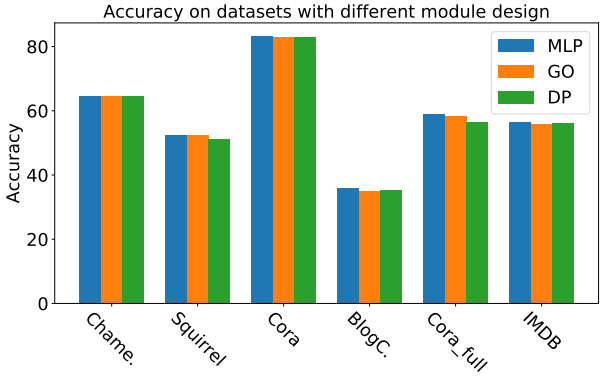


Figure 3: Influence of edge disentanglement module design.

5.5 Ablation Study

In this section, we conduct a series of ablation studies and sensitivity analysis to evaluate the influence of each component in DisGNN.

5.5.1 Edge Disentanglement Module Design. Different choices of edge-modeling mechanism could influence the representation ability of it in capturing relations [16]. To evaluate its significance, we tested two other widely-used designs in replace of the MLP-based disentanglement module used in Equation 6:

- Single-layer neural network (GO): uses a linear layer on the concatenation of node pair representations:

$$\hat{e}^{l,i}_{v,u} = sigmoid\left((\mathbf{a}^{l,i})^T\left([\mathbf{W}^{l,i}_1\mathbf{h}^l_v || \mathbf{W}^{l,i}_1\mathbf{h}^l_u]\right)\right) \tag{20}$$

- Dot-product (DP): uses inner-production on node embeddings to obtain pair-wise similarity:

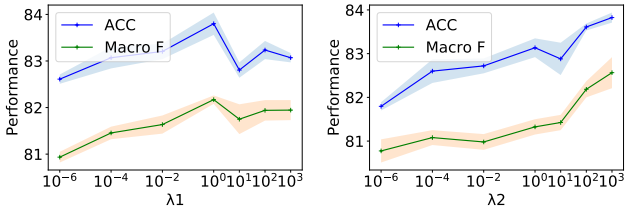
$$\hat{e}^{l,i}_{v,u} = sigmoid\left(\left(\mathbf{W}^{l,i}_1\mathbf{h}^l_v\right)^T \cdot \mathbf{W}^{l,i}_1\mathbf{h}^l_v\right) \tag{21}$$

Experiments are randomly conducted for 3 times, and all other configurations remained unchanged. The results are summarized in Figure 3. From the result, we can observe that in most cases, MLP-based disentanglement performs best while DP-based disentanglement performs worst. Generally, the performance gap is

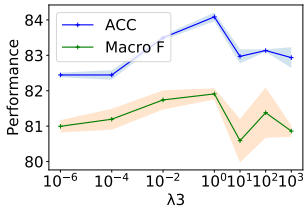
Table 4: Comparison with different SSL signals.

	Cora		BlogCatalog		IMDB		Chameleon	
Methods	ACC	F Score	ACC	F Score	ACC	F Score	ACC	F Score
DisGNN_Base	$81.81_{\pm 0.42}$	$80.47_{\pm 0.33}$	$32.15_{\pm 0.26}$	$23.22_{\pm 0.38}$	$54.98_{\pm 0.47}$	$47.02_{\pm 0.32}$	$62.73_{\pm 1.15}$	$62.83_{\pm 0.60}$
+MaskedAttr	$82.70_{\pm 0.45}$	$81.41_{\pm 0.25}$	$34.47_{\pm 0.21}$	$23.52_{\pm 0.69}$	$53.54_{\pm 0.24}$	$47.27_{\pm 0.40}$	$64.23_{\pm 0.53}$	$64.20_{\pm 0.54}$
+DistancePred	$82.74_{\pm 0.36}$	$81.37_{\pm 0.28}$	$34.72_{\pm 0.16}$	$23.53_{\pm 0.51}$	$56.07_{\pm 0.24}$	$47.03_{\pm 0.59}$	$63.87_{\pm 0.25}$	$63.63_{\pm 0.29}$
+ContextPred	$82.81_{\pm 0.57}$	$81.52_{\pm 0.57}$	$35.13_{\pm 0.19}$	$23.43_{\pm 0.31}$	$55.21_{\pm 0.51}$	$46.43_{\pm 0.50}$	$62.05_{\pm 0.36}$	$61.83_{\pm 0.53}$
DisGNN	$\mathbf{83.16}_{\pm 0.48}$	$\mathbf{81.99}_{\pm 0.71}$	$\mathbf{35.82}_{\pm 0.19}$	$\mathbf{23.94}_{\pm 1.04}$	$\mathbf{56.39}_{\pm 0.14}$	$\mathbf{47.73}_{\pm 0.34}$	$\mathbf{64.45}_{\pm 0.16}$	$\mathbf{64.39}_{\pm 0.16}$

clearer on large graphs like Cora_full, and smaller on small grpahs like Chameleon. We attribute this phenomenon to the greater expressive capacity of MLP over GO and DP.



(a) Weight of edge recovery loss (b) Weight of label conformity loss



(c) Weight of channel difference loss

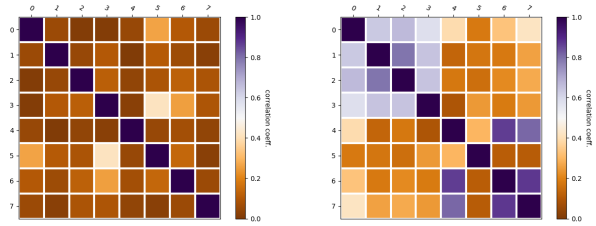
Figure 4: Parameter Sensitivity on Cora. $\lambda_1, \lambda_2, \lambda_3$ control the weight of each SSL signal respectively.

5.5.2 SSL Signal Weight. In this part, we vary the hyper-parameter $\lambda_1, \lambda_2, \lambda_3$ to test DisGNN's sensitivity towards them. These three values control the weights of each proposed SSL signal respectively. To keep simplicity, we keep all other configurations unchanged, and adopt only one SSL task at a time. Its corresponding λ varies in $\{1e-6, 1e-4, 1e-2, 1, 10, 1e2, 1e3\}$, and experiments are randomly conducted for 3 times on Cora.

From the result shown in Figure 4, we can observe that on Cora, increasing weight of Edge Recovery or Channel Difference is beneficial within the range $[0, 1]$, and further increasing that would result in performance drop. While for Label Conformity, its benefits is clear with a larger weight like with the range $[1e2, 1e3]$.

5.6 Case Study

In most cases, there is no ground-truth for disentangled factors behind edges in real-world graphs, making the direct evaluation of edge disentanglement difficult. To provide insights into the behavior of DisGNN and partly answer **RQ3**, we conduct a case study and compare the distribution of disentangled edges w/o proposed self-supervision tasks. Concretely, Pearson correlation coefficients among edges captured by each channel are shown in Figure 5.



(a) DisGNN_Base (b) DisGNN

Figure 5: Correlation coefficient among edges captured by different channels on Cora, obtained from the first layer.

Through comparing with DisGNN_base, we can see that when proposed SSL tasks are used, a clear grid-like structure can be observed from correlation among channels. In Figure 5(b), the top four channels and the bottom four channels have little correlation as they focus on homo-edges and hetero-edges respectively. Within these two groups, we can further observe that channel 1, 2 hold a higher correlation with each other, similar is the case of channels 4, 6, and 7. While channel 5 has a low correlation towards other channels within the same group. This result shows that although they all model hetero-edges, they still capture distinct patterns.

6 CONCLUSION

In this work, we studies the discovering and utilizing of latent relations behind edges to facilitate node classification, one typical task on learning from graphs. An edge disentanglement module is designed and incorporated into modern GNN layers, and three disentanglement-eliciting SSL signals are proposed to optimize jointly with node classification task. Experiments validate the advantage of the proposed DisGNN, showing the benefits of exploiting latent multi-relation nature of edges.

In the future, works can be done to discover more heuristics promoting the automatic edge disentanglement. Due to the lack of explicit supervision, it is remains an open problem. Furthermore, the benefits of disentanglement towards other graph learning tasks, like edge-level or graph-level tasks, is also a promising direction.

ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by, the National Science Foundation under grants number IIS-1707548, CBET-1638320, IIS-1909702, IIS-1955851, and the Army Research Office under grant W911NF21-1-0198. The findings and conclusions in this paper do not necessarily reflect the view of the funding agency.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29.

[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*. 1993–2001.

[3] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How Attentive are Graph Attention Networks? *arXiv preprint arXiv:2105.14491* (2021).

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[5] Hryhorii Chereda, A. Bleckmann, F. Kramer, A. Leha, and T. Beißbarth. 2019. Utilizing Molecular Network Information via Graph Convolutional Neural Networks to Predict Metastatic Event in Breast Cancer. *Studies in health technology and informatics* 267 (2019), 181–186.

[6] Enyan Dai, Charu Aggarwal, and Suhang Wang. 2021. NRGNN: Learning a Label Noise Resistant Graph Neural Network on Sparsely and Noisily Labeled Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 227–236.

[7] Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. 2022. Towards Robust Graph Neural Networks for Noisy Graphs with Sparse Labels. *arXiv preprint arXiv:2201.00232* (2022).

[8] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[9] Wenqi Fan, Y. Ma, Qing Li, Yuan He, Y. Zhao, Jiliang Tang, and D. Yin. 2019. Graph Neural Networks for Social Recommendation. *The World Wide Web Conference* (2019).

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

[11] William L. Hamilton, Zhitao Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[12] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for Pre-training Graph Neural Networks. In *International Conference on Learning Representations*.

[13] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. 2020. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141* (2020).

[14] Longlong Jing and Yingli Tian. 2020. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2020).

[15] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 27.

[16] Dongkwan Kim and Alice Oh. 2020. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*.

[17] Thomas Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2017).

[18] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[19] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. 2021. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111* (2021).

[20] Yanbei Liu, Xiao Wang, Shu Wu, and Zhitao Xiao. 2020. Independence promoted graph disentangled networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4916–4923.

[21] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. 2019. Disentangled graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 4212–4221.

[22] Elman Mansimov, O. Mahmood, Seokho Kang, and Kyunghyun Cho. 2019. Molecular Geometry Prediction using a Deep Generative Graph Neural Network. *Scientific Reports* 9 (2019).

[23] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2019. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

[24] Bryan Perozzi, Rami Al-Rfou, and S. Skiena. 2014. DeepWalk: online learning of social representations. In *KDD '14*.

[25] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1150–1160.

[26] Neelam Rout, Debahuti Mishra, and Manas Kumar Mallick. 2018. Handling imbalanced data: A survey. In *International Proceedings on Advances in Soft Computing, Intelligent Systems and Applications*. Springer, 431–443.

[27] Daniil Sorokin and Iryna Gurevych. 2018. Modeling Semantics with Gated Graph Neural Networks for Knowledge Base Question Answering. *ArXiv* abs/1808.04126 (2018).

[28] S. Tang, Bo Li, and Haijun Yu. 2019. ChebNet: Efficient and Stable Constructions of Deep Neural Networks with Rectified Power Units using Chebyshev Approximations. *ArXiv* abs/1911.05467 (2019).

[29] Petar Velickovic, Guillem Cucurull, A. Casanova, A. Romero, P. Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ArXiv* abs/1710.10903 (2018).

[30] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.

[31] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[32] Teng Xiao, Zhengyu Chen, Donglin Wang, and Suhang Wang. 2021. Learning how to propagate messages in graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1894–1903.

[33] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. 2021. Self-supervised learning of graph neural networks: A unified review. *arXiv preprint arXiv:2102.10757* (2021).

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[35] Yiding Yang, Zunlei Feng, Mingli Song, and Xinchao Wang. 2020. Factorizable Graph Convolutional Networks. *Advances in Neural Information Processing Systems* 33 (2020).

[36] Jiaqi Zeng and Pengtao Xie. 2020. Contrastive self-supervised learning for graph classification. *arXiv* (2020).

[37] Tianxiang Zhao, Xianfeng Tang, Xiang Zhang, and Suhang Wang. 2020. Semi-Supervised Graph-to-Graph Translation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1863–1872.

[38] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining*.

[39] T. Zhong, Tianliang Wang, Jiahao Wang, J. Wu, and Fan Zhou. 2020. Multiple-Aspect Attentional Graph Neural Networks for Online Social Network User Localization. *IEEE Access* 8 (2020), 95223–95234.

[40] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *Advances in Neural Information Processing Systems* 33 (2020).