Disentangled Graph Convolutional Networks

Jianxin Ma [1] **Peng Cui** [1] **Kun Kuang** [1] **Xin Wang** [1] **Wenwu Zhu** [1]

Abstract

The formation of a real-world graph typically arises from the highly complex interaction of many latent factors. The existing deep learning methods for graph-structured data neglect the entanglement of the latent factors, rendering the learned representations non-robust and hardly explainable. However, learning representations that disentangle the latent factors poses great challenges and remains largely unexplored in the literature of graph neural networks. In this paper, we introduce the disentangled graph convolutional network (DisenGCN) to learn disentangled node representations. In particular, we propose a novel neighborhood routing mechanism, which is capable of dynamically identifying the latent factor that may have caused the edge between a node and one of its neighbors, and accordingly assigning the neighbor to a channel that extracts and convolutes features specific to that factor. We theoretically prove the convergence properties of the routing mechanism. Empirical results show that our proposed model can achieve significant performance gains, especially when the data demonstrate the existence of many entangled factors.

1. Introduction

Data with a graph structure, e.g., social networks, have become increasingly prevalent. Recently, graph neural networks (Gori et al., 2005; Scarselli et al., 2009), particularly graph convolutional networks (Bruna et al., 2014; Henaff et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2017), have demonstrated their remarkable ability in learning representations that are highly effective for prediction, thanks to their end-to-end nature and adoption of deep architectures.

Despite their enormous success, the existing graph neural networks generally take a holistic approach to representation learning: the representation learned for a node describes the node's neighborhood as a perceptual whole, and the nuances between the different parts of the neighborhood are ignored. Yet the formation of a real-world graph typically follows a complex and heterogeneous process, driven by the interaction of many latent factors. For example, a person in a social network usually connects with others for various reasons (e.g., work, school), and therefore possesses a neighborhood consisting of several different components. The existing holistic approaches fail to recognize and disentangle the heterogeneous factors. As a result, the representations learned by them could be non-robust (e.g., prone to overreact to an irrelevant factor) and hardly explainable.

More recently, disentangled representation learning has gained considerable attention, in particular in the field of image representation learning (Higgins et al., 2016; Chen et al., 2017; Alemi et al., 2017; Kim & Mnih, 2018). It aims to learn representations that separate the explanatory factors of variations behind the data. Such representations are demonstrated to be more resilient to the complex variants (Bengio et al., 2013), and able to bring enhanced generalization ability as well as improved robustness to adversarial attack (Alemi et al., 2017). Moreover, the disentangled representations are inherently more interpretable, and thus can potentially facilitate debugging and auditing (Doshi-Velez & Kim, 2017; Lipton, 2018). However, how to learn representations that disentangle the latent factors behind a graph remains largely unexplored in the literature of graph neural networks.

The characteristics of graphs pose great challenges to disentangled representation learning. The complex formation process of graphs requires the graph neural networks to have a sophisticated mechanism for inferring the latent factor that may have caused an edge, based on the limited information available, such as node attributes or graph structures. In addition, the mechanism needs to be differentiable so as to support end-to-end training, and be capable of conducting inductive learning in order to enable out-of-sample node processing (Ma et al., 2018; Hamilton et al., 2017) in real time for real-world deployment.

[1]Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing, 100084, China. Correspondence to: Jianxin Ma <majx13fromthu@gmail.com>, Peng Cui, Xin Wang, Wenwu Zhu <{cuip,xin_wang,wwzhu}@tsinghua.edu.cn>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

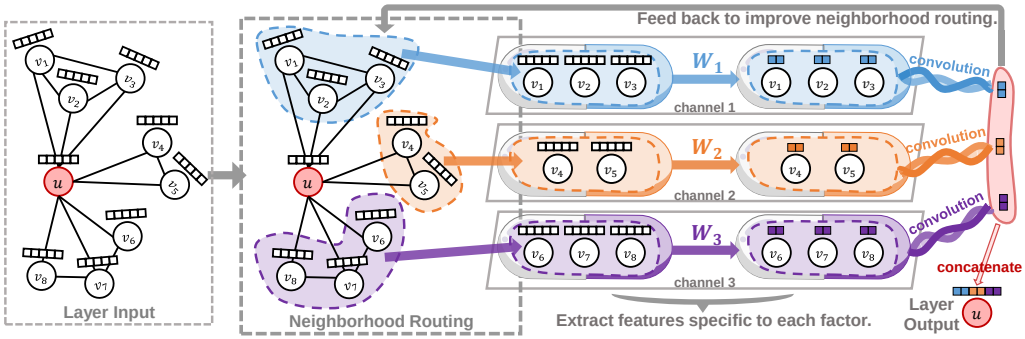


Figure 1. The disentangled convolutional (DisenConv) layer. It takes in a node and its neighbors as well as their feature vectors, which can be the output of the previous layer, and outputs a disentangled representation for the node. The neighborhood routing mechanism iteratively segments the neighborhood according to the underlying factors. The outputs of the channels are fed back to improve routing at the end of each iteration. This example assumes that there are three latent factors, hence the three channels.

In this paper, we present the disentangled graph convolutional network (DisenGCN), an end-to-end deep model that addresses the above challenges and learns disentangled node representations. The key ingredient of DisenGCN is DisenConv, a disentangled multichannel convolutional layer (see Figure 1). We propose a novel neighborhood routing mechanism, which is executed inside DisenConv, to identify the factor that may have caused the link from a given node to one of its neighbors, and accordingly send the neighbor to the channel responsible for that factor. The mechanism infers the latent factors by iteratively analyzing the potential subspace clusters formed by the node and its neighbors, after projecting them into several subspaces. Each channel [1] of DisenConv then extracts features specific to one of the disentangled factors from the neighbors it has received, and performs a convolution operation independently. The neighborhood routing mechanism is composed purely of differentiable modules. In addition, it only requires information from the local neighborhood, which allows us to express DisenConv as a mapping from a neighborhood to a node representation, and thus inductive learning can be supported. By stacking multiple DisenConv layers, DisenGCN is able to extract information beyond the local neighborhood.

We theoretically analyze the convergence properties of the neighborhood routing mechanism, by establishing its connection with probabilistic inference under a projected mixture model. Extensive experiments on various real-world graphs show that DisenGCN can achieve substantial performance gains, around 20% in many cases.

Our main contributions are summarized as follows:

- We present DisenGCN, a novel graph neural network that learns disentangled node representations.
- We propose neighborhood routing to infer the factor behind the formation of each edge, in a manner that is differentiable and supports inductive learning.
- We theoretically analyze the convergence properties of neighborhood routing, and empirically demonstrate the advantages of learning disentangled representations.

2. DisenGCN: the Proposed Model

In this section, we first present the DisenConv layer, and then describe the overall network architecture of DisenGCN.

2.1. Notations and Problem Formulation

We will focus primarily on undirected graphs, though it is straightforward to extend our approach to directed graphs. Let $G = (V, E)$ be a graph, comprised of a set of nodes V and a set of edges E. We use $(u, v) \in G$, or $(u, v) \in E$, to indicate that there is an edge between node u and node v. Each node $u \in V$ has a feature vector $\mathbf{x}_u \in \mathbb{R}^{d_{in}}$.

The key element of most graph convolutional [2] networks, including ours, is a layer $f(\cdot)$ that outputs a representation for a node when given the node's and its neighbors' features:

$$\mathbf{y}_u = f\left(\mathbf{x}_u, \{\mathbf{x}_v : (u, v) \in G\}\right).$$

The output $\mathbf{y}_u \in \mathbb{R}^{d_{out}}$ is viewed as the representation of node u, learned by the layer. The idea is that the neighborhood of a node provides rich information that can be leveraged to more comprehensively characterize the node.

We aim to derive a layer $f(\cdot)$ such that the output $\mathbf{y}_u$ is a disentangled representation. Specifically, we would like $\mathbf{y}_u$ to be composed of K independent components, i.e.,

[1] The output of each channel can be seen as a *capsule* (Hinton et al., 2011). DisenGCN is a capsule neural network in this regard.

[2] We follow the literature and use "convolutional" to refer to an operation that aggregates information from a neighborhood, which is also known as spatial filtering (Shuman et al., 2013).

Algorithm 1 The proposed DisenConv layer, with K channels. It performs T iterations of routing. Typically $T \approx 5$.

Input: $\mathbf{x}_u \in \mathbb{R}^{d_{in}}$ (the feature vector of node u), and $\{\mathbf{x}_v \in \mathbb{R}^{d_{in}} : (u,v) \in G\}$ (its neighbors' features).
Output: $\mathbf{y}_u \in \mathbb{R}^{d_{out}}$ (the representation of node u).
Param: $\mathbf{W}_k \in \mathbb{R}^{d_{in} \times \frac{d_{out}}{K}}$, $\mathbf{b}_k \in \mathbb{R}^{\frac{d_{out}}{K}}$, $k = 1, \ldots, K$.
for $i \in \{u\} \cup \{v : (u,v) \in G\}$ **do**
  for $k = 1, 2, \ldots, K$ **do**
    $\mathbf{z}_{i,k} \leftarrow \sigma(\mathbf{W}_k^\top \mathbf{x}_i + \mathbf{b}_k)$.
    $\mathbf{z}_{i,k} \leftarrow \mathbf{z}_{i,k} / \|\mathbf{z}_{i,k}\|_2$. // The k^{th} aspect of node i.
  end for
end for
$\mathbf{c}_k \leftarrow \mathbf{z}_{u,k}, \forall k = 1, 2, \ldots, K$. // Initialize K channels.
for routing iteration $t = 1, 2, \ldots, T$ **do**
  for v that satisfies $(u,v) \in G$ **do**
    $p_{v,k} \leftarrow \mathbf{z}_{v,k}^\top \mathbf{c}_k / \tau, \forall k = 1, 2, \ldots, K$.
    $[p_{v,1} \ldots p_{v,K}] \leftarrow \text{softmax}([p_{v,1} \ldots p_{v,K}])$.
  end for
  for channel $k = 1, 2, \ldots, K$ **do**
    $\mathbf{c}_k \leftarrow \mathbf{z}_{u,k} + \sum_{v:(u,v)\in G} p_{v,k}\, \mathbf{z}_{v,k}$. // Update.
    $\mathbf{c}_k \leftarrow \mathbf{c}_k / \|\mathbf{c}_k\|_2$.
  end for
end for
$\mathbf{y}_u \leftarrow$ the concatenation of $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$.

$\mathbf{y}_u = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K]$, where $\mathbf{c}_k \in \mathbb{R}^{\frac{d_{out}}{K}}$ $(1 \le k \le K)$, assuming that there are K latent factors to be disentangled. The k^{th} component $\mathbf{c}_k$ is for describing the aspect of node u that are pertinent to factor k. The key challenge is to identify the subset of neighbors that are actually connected by node u due to factor k, so as to more accurately describe the k^{th} aspect of node u. To this end, we propose the DisenConv layer, presented in the next subsection.

2.2. The DisenConv Layer

Given $\mathbf{x}_u \in \mathbb{R}^{d_{in}}$ and $\{\mathbf{x}_v \in \mathbb{R}^{d_{in}} : (u,v) \in G\}$ as input, a DisenConv layer outputs $\mathbf{y}_u = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K] \in \mathbb{R}^{d_{out}}$, where $\mathbf{c}_k \in \mathbb{R}^{\frac{d_{out}}{K}}$ describes the k^{th} aspect of node u.

DisenConv consists of K channels. And we view $\mathbf{c}_k$ as the final output of the k^{th} channel. We will first assume that the K channels can extract different features [3] when given a single node $i \in \{u\} \cup \{v : (u,v) \in G\}$, by projecting the feature vector $\mathbf{x}_i$ into different subspaces:

$$\mathbf{z}_{i,k} = \frac{\sigma(\mathbf{W}_k^\top \mathbf{x}_i + \mathbf{b}_k)}{\left\|\sigma(\mathbf{W}_k^\top \mathbf{x}_i + \mathbf{b}_k)\right\|_2}, \tag{1}$$

where $\mathbf{W}_k \in \mathbb{R}^{d_{in} \times \frac{d_{out}}{K}}$ and $\mathbf{b}_k \in \mathbb{R}^{\frac{d_{out}}{K}}$ are the parameters of channel k, and $\sigma(\cdot)$ is a nonlinear activation function. We use l_2-normalization to ensure numerical stability and prevent the neighbors with overly rich features (e.g., long text) from distorting our prediction. We then assume that $\mathbf{z}_{i,k}$ approximately describes the aspect of node i that are related with the k^{th} factor, provided that $\mathbf{x}_i$ does contain meaningful information about the related aspect.

The feature vector $\mathbf{x}_i$, however, is typically incomplete in the real world, e.g., a user may read but never post anything. We hence cannot directly use $\mathbf{z}_{u,k}$ to serve as $\mathbf{c}_k$ for the input node u. To comprehensively capture aspect k of node u, we are required to mine information from the neighborhood, i.e., to construct $\mathbf{c}_k$ from both $\mathbf{z}_{u,k}$ and $\{\mathbf{z}_{v,k} : (u,v) \in G\}$.

The key insight here is that we should not use all the neighbors when constructing $\mathbf{c}_k$ to describe aspect k of node u. Specifically, we should use only the neighbors that are actually connected with node u due to factor k. The challenge is to design a mechanism for inferring the subset of the neighbors that are connected by node u due to factor k.

We therefore propose the *neighborhood routing* mechanism, which is based on two plausible hypotheses [4]. The first hypothesis focuses on the relationships among the neighbors:

Hypothesis 1. Factor k is likely to be the reason why node u connects with a certain subset of its neighbors, if the subset is large and the neighbors in the subset are similar w.r.t. aspect k, i.e., they form a cluster in the k^{th} subspace.

This first hypothesis inspires us to search for the largest cluster in each of the K subspaces projected from the original feature space. This hypothesis is robust under the scenario where $\mathbf{x}_u$ is noisy or incomplete, since $\mathbf{x}_u$ is not involved. Moreover, when seeking the large clusters, the neighbors who lack information about factor k will be automatically pruned, since their projected features $\mathbf{z}_{v,k}$ will be noises and will not form a large enough cluster.

The second hypothesis, on the other hand, focuses on the relationship between node u and one of its neighbors:

Hypothesis 2. Factor k is likely to be the reason why node u and neighbor v are connected, if the two are similar in terms of aspect k.

This second hypothesis suggests that $\mathbf{z}_{u,k}{}^\top \mathbf{z}_{v,k}$ can provide a hint on the factor behind the edge between u and v, which is fast to compute and effective, provided that $\mathbf{x}_u$ and $\mathbf{x}_v$ do contain sufficient information about factor k.

[3] Random initialization is sufficient to ensure the difference in the beginning. During training, the channels will remain different, because they receive different subsets of the neighbors and hence different supervision signals thanks to neighborhood routing.

[4] Hypothesis 2 and Hypothesis 1 are analogous to the first-order and the second-order proximity, respectively. The two concepts of proximity are widely accepted explanations for the existence of a link, with evidence from sociology (Granovetter, 1973) and linguistics (Firth, 1930–1955), and are the essential ingredients of many algorithms, e.g., LINE (Tang et al., 2015).

The hint provided by Hypothesis 2, albeit computationally efficient, can be misleading when $\mathbf{x}_u$ or $\mathbf{x}_v$ lacks information about factor k. We therefore need to mitigate this issue by combining it with Hypothesis 1. Meanwhile, Hypothesis 1 requires a clustering procedure, which typically involves many iterations. Hypothesis 2 can then serve as a strong prior to guide clustering, in order to achieve fast convergence. Therefore, we propose our neighborhood routing mechanism based on both Hypothesis 1 and Hypothesis 2.

Let $p_{v,k}$ be the probability that factor k is the reason why node u reaches neighbor v, which should satisfies $p_{v,k} \geq 0$ and $\sum_{k'=1}^{K} p_{v,k'} = 1$. Then $p_{v,k}$ is also the probability that we should use neighbor v to construct $\mathbf{c}_k$. The neighborhood routing mechanism will iteratively infer $p_{v,k}$ and construct $\mathbf{c}_k$. It starts by initializing $p_{v,k}$ as $p_{v,k}^{(1)} \propto \exp(\mathbf{z}_{v,k}{}^\top \mathbf{z}_{u,k}/\tau)$, based on Hypothesis 2. Motivated by Hypothesis 1, it then iteratively searches for the largest cluster in each subspace, under the constraint that each neighbor should approximately belong to only one subspace cluster:

$$\mathbf{c}_k^{(t)} = \frac{\mathbf{z}_{u,k} + \sum_{v:(u,v)\in G} p_{v,k}^{(t-1)}\, \mathbf{z}_{v,k}}{\|\mathbf{z}_{u,k} + \sum_{v:(u,v)\in G} p_{v,k}^{(t-1)}\, \mathbf{z}_{v,k}\|_2}, \tag{2}$$

$$p_{v,k}^{(t)} = \frac{\exp(\mathbf{z}_{v,k}{}^\top \mathbf{c}_k^{(t)}/\tau)}{\sum_{k'=1}^{K} \exp(\mathbf{z}_{v,k'}{}^\top \mathbf{c}_{k'}^{(t)}/\tau)}, \tag{3}$$

for iteration $t = 2, \ldots, T$, where τ is a hyper-parameter that controls the hardness of the assignment. Finally, it outputs $\mathbf{c}_k = \mathbf{c}_k^{(T)}$. We can view $\mathbf{c}_k$ as the center of each subspace cluster here. Hypothesis 2 is not only used for initialization, but also used as a prior during every iteration, i.e., the term $\mathbf{z}_{u,k}$ in Equation 2, in order to ensure fast convergence. We will formally prove that the algorithm converges in section 3.

The pseudocode of a DisenConv layer is listed in Algorithm 1, which involves only differentiable operations.

2.3. Network Architecture

In this subsection, we describe the overall network architecture of DisenGCN for performing node-related tasks.

Let $G = (V, E)$ be the input graph. Node u is associated with a feature vector $\mathbf{x}_u \in \mathbb{R}^D$ and a binary vector of ground-truth labels $\mathbf{y}_u \in \{0, 1\}^C$, where C is the number of classes. Some graph datasets do not provide node features. In that case, we simply use the u^{th} row of the adjacency matrix of G to serve as the feature vector $\mathbf{x}_u$.

In practice, it may be desirable to stack multiple DisenConv layers. First, this allows us to mine information beyond the local neighborhood when producing a node's representation. For example, we can leverage the neighbors' neighbors, as well as the edges between two neighbors, by stacking two DisenConv layers. Secondly, we can potentially learn hierarchical representations, by gradually decreasing the number of channels at the later layers.

DisenGCN thus uses L DisenConv layers. Let $f^{(l)}(\cdot)$ be a DisenConv layer at the l^{th} layer, and let $\mathbf{y}_u^{(l)} \in \mathbb{R}^{K^{(l)}\Delta d}$ for $u \in V$ be the output of the layer. Here $K^{(l)}$ is the number of channels used by layer l. And we keep Δd, the output dimension of a channel, to be the same across all layers. We additionally impose the constraint $K^{(1)} \geq K^{(2)}) \geq \ldots \geq K^{(L)}$. ReLU is used as the activation function in Equation 1. The output of layer l can then be expressed as

$$\mathbf{y}_u^{(l)} = \text{dropout}\left(f^{(l)}\left(\mathbf{y}_u^{(l-1)}, \{\mathbf{y}_v^{(l-1)} : (u,v) \in G\}\right)\right),$$

where $1 \leq l \leq L$, $\mathbf{y}_u^{(0)} = \mathbf{x}_u$, and $u \in V$. The dropout operation (Srivastava et al., 2014) is appended after every layer and is enabled only during training. The final layer is a fully-connected layer, i.e., $\mathbf{y}^{(L+1)} = \mathbf{W}^{(L+1)\top}\mathbf{y}^{(L)} + \mathbf{b}^{(L+1)}$, where $\mathbf{W}^{(L+1)} \in \mathbb{R}^{K^{(L)}\Delta d \times C}, \mathbf{b}^{(L+1)} \in \mathbb{R}^C$.

We use $-\frac{1}{C}\sum_{c=1}^{C} \mathbf{y}_u(c) \ln(\hat{\mathbf{y}}_u(c))$, where $\hat{\mathbf{y}}_u = \text{softmax}(\mathbf{y}_u^{(L+1)})$, as the loss function for single-label node classification. For multi-label node classification, where $\mathbf{y}_u$ can have more than one positive bits, we use the following loss function: $-\frac{1}{C}\sum_{c=1}^{C}[\mathbf{y}_u(c) \cdot \text{sigmoid}(\mathbf{y}_u^{(L+1)}(c)) + (1 - \mathbf{y}_u(c)) \cdot \text{sigmoid}(-\mathbf{y}_u^{(L+1)}(c))]$. We compute the gradients via back-propagation, and optimize the parameters with Adam (Kingma & Ba, 2015).

3. Theoretical Analysis

In this section, we investigate two important problems about the proposed neighborhood routing mechanism: (1) whether it converges after a sufficient number of iterations, and (2) to what solution it converges if it does. We will answer these two questions simultaneously by showing the connection between neighborhood routing and a von Mises-Fisher (vMF) mixture model proposed for this purpose.

Given the observation $\{\mathbf{z}_{u,k}\}_{k=1}^K$ and $\{\mathbf{z}_{v,k} : (u,v) \in G, 1 \leq k \leq K\}$, the vMF mixture model has a set of parameters $\{\mathbf{c}_k\}_{k=1}^K$, and is defined as follows:

$$\begin{aligned}
&\mathbf{z}_{u,k} \sim \text{vMF}(\mathbf{c}_k, 1),\\
&r_v \sim \text{Categorical}\left([1/K, 1/K, \ldots, 1/K]\right),\\
&\mathbf{z}_{v,r_v} \mid r_v \sim \text{vMF}(\mathbf{c}_{r_v}, 1/\tau),\\
&\mathbf{z}_{v,k'} \mid r_v \sim \text{vMF}(\boldsymbol{\mu}, 0), \quad k' \neq r_v \ \wedge\ 1 \leq k' \leq K,
\end{aligned}$$

where $v \in \{v : (u,v) \in G\}$, $1 \leq k \leq K$, and $\boldsymbol{\mu}$ is another parameter. The value of $\boldsymbol{\mu}$ is not important, because the probability density function of $\text{vMF}(\boldsymbol{\mu}, \kappa)$ is defined as $f_{vMF}(\mathbf{x}; \boldsymbol{\mu}, \kappa) \propto \exp(\kappa \boldsymbol{\mu}^\top \mathbf{x})$, where $\|\boldsymbol{\mu}\|_2 = \|\mathbf{x}\|_2 = 1$, and thus $f_{vMF}(\mathbf{x}; \boldsymbol{\mu}, 0)$ is constant.

The mixture model views $\{\mathbf{c}_k\}_{k=1}^K$ as the true k aspects of node u to be estimated, with the following assumptions: First, the extracted features $\{\mathbf{z}_{u,k}\}_{k=1}^K$ of node u is a noisy observation of $\{\mathbf{c}_k\}_{k=1}^K$. Second, neighbor v and node u are connected due to a unknown factor r_v, and they should be similar in terms of aspect r_v. Third, if factor k' is not the factor that leads to the connection between u and v, then we do not have any information abut aspect k' of neighbor v, and the best we can do is to assume that $\mathbf{z}_{v,k'}$ is sampled uniformly, i.e., sampled from $\mathrm{vMF}(\boldsymbol{\mu}, 0)$.

With the vMF mixture model, we derive the following theorem on neighborhood routing's convergence properties:

Theorem 1. *The neighborhood routing mechanism is equivalent to an expectation-maximization (EM) algorithm for the mixture model. In particular, it converges to a point estimate of $\{\mathbf{c}\}_{k=1}^K$ that maximizes the marginal likelihood $p\left(\{\mathbf{z}_{i,k} : i = u \vee (u,i) \in G, 1 \le k \le K\}; \{\mathbf{c}\}_{k=1}^K\right)$.*

Proof. Let $\theta = \{\mathbf{c}_k\}_{k=1}^K$, $R = \{r_v : (u,v) \in G\}$, and $Z = \{\mathbf{z}_{i,k} : i = u \vee (u,i) \in G, 1 \le k \le K)\}$. To derive an EM algorithm that maximizes $p(Z;\theta) = \sum_R p(R,Z;\theta)$, we introduce here an additional auxiliary distribution $q(R)$ over R. Let $L(\theta,q) = \sum_R q(R) \ln \frac{p(R,Z;\theta)}{q(R)}$ and $D_{\mathrm{KL}}\left(q \| p_\theta\right) = \sum_R q(R) \ln \frac{q(R)}{p(R|Z;\theta)}$. We can then verify that $\ln p(Z;\theta) = L(\theta,q) + D_{\mathrm{KL}}\left(q \| p_\theta\right)$. The second term here is the Kullback-Leibler (KL) divergence from $p(R \mid Z;\theta)$ to the auxiliary distribution $q(R)$. The KL divergence is non-negative. As a result, $L(\theta,q)$ is a lower bound of $\ln p(Z;\theta)$.

The E-step of the EM algorithm is to find $q(R)$ that tightens the lower bound. This can be achieve by setting $q(R)$ to $p(R \mid Z;\theta)$, since the KL divergence will become zero. Note that $p(R \mid Z;\theta) = \prod_v p(r_v \mid Z;\theta)$, and $p(r_v = k \mid Z;\theta) \propto p(r_v = k, Z;\theta) \propto \exp(\mathbf{z}_{v,k}{}^\top \mathbf{c}_k/\tau)$. Therefore, the optimal $q(R)$ that tightens the bound is $q(r_v = k) \propto \exp(\mathbf{z}_{v,k}{}^\top \mathbf{c}_k/\tau)$. This proves that Equation 3 is performing the E-step and $p_{v,k} = q(r_v = k) = p(r_v = k \mid Z;\theta)$.

After every E-step, the EM algorithm performs an M step to maximize the lower bound $L(\theta,q)$ w.r.t. θ, with $q(R)$ fixed to the value found in the E-step. Note that we have $\frac{\partial L(\theta,q)}{\mathbf{c}_k} = \mathbf{c}_k{}^\top \left(\mathbf{z}_{u,k} + \sum_v p_{v,k}\mathbf{z}_{v,k}\right)$. We need to optimize $\mathbf{c}_k$ under the constraint that $\|\mathbf{c}_k\|_2 = 1$. We therefore find the optimal $\mathbf{c}_k$ by setting $\frac{\partial}{\partial \mathbf{c}_k}\left[L(\theta,q) + \lambda(\|\mathbf{c}_k\|_2^2 - 1)\right]$ to zero. It turns out that the optimal $\mathbf{c}_k$ is exactly Equation 2. Thus Equation 2 is in fact performing the M-step.

Let $q^{(t)}(R)$ and $\theta^{(t)}$ be the result of the t^{th} E-step and the t^{th} M-step, respectively. Then $\ln p(Z;\theta^{(t-1)}) = L(\theta^{(t-1)},q) + D_{\mathrm{KL}}\left(q \| p_{\theta^{(t-1)}}\right) = L(\theta^{(t-1)},q^{(t)}) \le L(\theta^{(t)},q^{(t)}) \le L(\theta^{(t)},q^{(t)}) + D_{\mathrm{KL}}\left(q^{(t)} \| p_{\theta^{(t)}}\right) = \ln p(Z;\theta^{(t)})$. The likelihood thus increases monotonically, while being upper-bounded by zero. The algorithm therefore converges. □

4. Empirical Results

In this section, we empirically assess the efficacy of DisenGCN on several node-related tasks, and analyze its behavior on synthetic graphs to gain further insight.

4.1. Experimental Setup

Baselines To demonstrate the advantages of our approach, we compare DisenGCN with two representative graph neural networks, including the graph convolution network (GCN) (Kipf & Welling, 2017) and the graph attention network (GAT) (Veličković et al., 2018). In particular, GAT is the state-of-the-art graph neural network on node-related tasks whose source code is available. When performing graph convolution, GCN weights a node's neighbors according to their degrees, while GAT learns an parameterized attention mechanism to prune irrelevant neighbors. Our model contains the same number of parameters as GCN, but much less than that of GAT's. The original implementations of GCN and GAT do not support multi-label tasks. We therefore modify them to use the same multi-label loss function as ours for fair comparison in multi-label tasks.

We additionally include three node embedding algorithms, including DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015), and node2vec (Grover & Leskovec, 2016), for multi-label classification, because they are demonstrated to perform strongly on the multi-label tasks.

Datasets We conduct our experiments on six real-world graphs, whose statistics are listed in Table 1. Citeseer, Cora, and Pubmed (Sen et al., 2008) are for semi-supervised node classification. The nodes, edges, and labels in these three represent articles, citations, and research areas, respectively. BlogCatalog (Tang & Liu, 2009), PPI (Breitkreutz et al., 2008; Grover & Leskovec, 2016), POS (Grover & Leskovec, 2016) are for multi-label node classification. Their labels are user interests, biological states, and part-of-speech tags, respectively. The latter three graphs do not provide node features. We therefore use the rows of their adjacency matrices in place of node features for them.

Hyper-parameters Let d be the output dimension of a graph neural network's first layer. In the semi-supervised classification tasks, we follow GAT and use $d = 64$. In the multi-label classification tasks, we follow node2vec and use $d = 128$, while setting the dimension of the node embeddings to be 128 as well for the node embedding algorithms. The output dimension of DisenGCN's first layer is $K^{(1)}\Delta d$, where $K^{(1)}$ is the number of channels used by the layer and Δd is the output dimension of each channel. We therefore use $K^{(1)}\Delta d = K^{(1)}\left\lfloor d/K^{(1)}\right\rfloor$ instead of d for our model when $d/K^{(1)}$ is not an integer. We set $T = 7$. We set $\tau = 1$, though a smaller value such as 0.1 should lead to

Table 1. Dataset statistics.

Dataset	Type	Nodes	Edges	Classes	Features	Multi-label
Citeseer	Citation network	3,327	4,732	6	3,703	No
Cora	Citation network	2,708	5,429	7	1,433	No
Pubmed	Citation network	19,717	44,338	3	500	No
Blogcatalog	Social network	10,312	333,983	39	-	Yes
PPI	Biological network	3,890	76,584	50	-	Yes
POS	Word co-occurrence	4,777	184,812	40	-	Yes

better interpretability. We tune the hyper-parameters of both our model's and our baselines' using hyperopt (Bergstra et al., 2013). Specifically, we run hyperopt for 200 trials for each setting, with the hyper-parameter search space specified as follows: the learning rate $\sim \text{loguniform}[e^{-8}, 1]$, the l_2 regularization term $\sim \text{loguniform}[e^{-10}, 1]$, dropout rate $\in \{0.05, 0.10, \ldots, 0.95\}$, the number of layers $L \in \{1, 2, \ldots, 6\}$, the number of channels used by the first layer $K^{(1)} \in \{4, 8, \ldots, 32\}$, and $K^{(l+1)} - K^{(l)} = \Delta K \in \{0, 2, \ldots, 8\}$, i.e., each layer has ΔK fewer channels than its previous layer. Then with the best hyper-parameters on the validation sets, we report the averaged performance of 100 runs on each semi-supervised dataset, and 30 runs on each multi-label dataset.

4.2. Semi-Supervised Node Classification

In this task, each dataset contains only 20 labeled instances for each class. Hence the graph structure must be leveraged when predicting the labels of the rest. We follow the experiment protocol established by the previous works (Yang et al., 2016; Kipf & Welling, 2017; Veličković et al., 2018) strictly, and use the same dataset splits as them.

The results are listed in Table 2. The three datasets used here do not contain as many factors as the multi-label ones. The majority of the nodes in them only connects with neighbors of the same class. Nevertheless, our model still outperforms the baselines. The optimal number of layers for DisenGCN found by hyperopt is 5, while GCN and GAT both use two layers. The improved performance thus might stem from DisenGCN's ability to leverage a deep architecture better. DisenGCN, by taking a disentangled approach instead of a holistic one, does not suffer from the over-smoothing problem faced by a deep GCN (Li et al., 2018). It is also less prone to over-fitting compared with a deep GAT, because our neighborhood routing mechanism does not introduce extra parameters, contrary to the attention mechanism.

4.3. Multilabel Node Classification

We follow node2vec (Grover & Leskovec, 2016) and report the performance of each method while varying the number of nodes labeled for training from $10\%|V|$ to $90\%|V|$, where $|V|$ is the total number of nodes. The rest of the nodes are split equally to form a validation set and a test set.

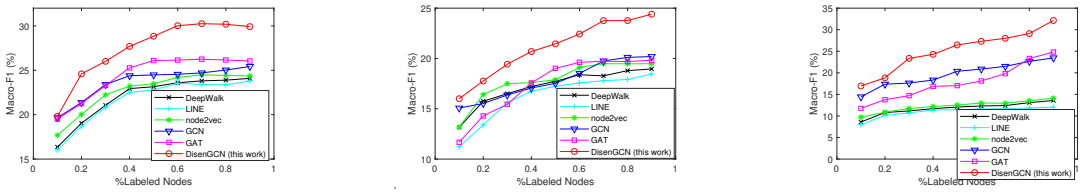
Table 2. Semi-supervised classification accuracies (%).

	Datasets		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1	46.5	71.4
ManiReg (Belkin et al., 2006)	59.5	60.1	70.7
SemiEmb (Weston et al., 2012)	59.0	59.6	71.1
LP (Zhu et al., 2003)	68.0	45.3	63.0
DeepWalk (Perozzi et al., 2014)	67.2	43.2	65.3
ICA (Lu & Getoor, 2003)	75.1	69.1	73.9
Planetoid (Yang et al., 2016)	75.7	64.7	77.2
ChebNet (Defferrard et al., 2016)	81.2	69.8	74.4
GCN (Kipf & Welling, 2017)	81.5	70.3	79.0
MoNet (Monti et al., 2017)	81.7	-	78.8
GAT (Veličković et al., 2018)	83.0	72.5	79.0
DisenGCN (this work)	**83.7**	**73.4**	**80.5**

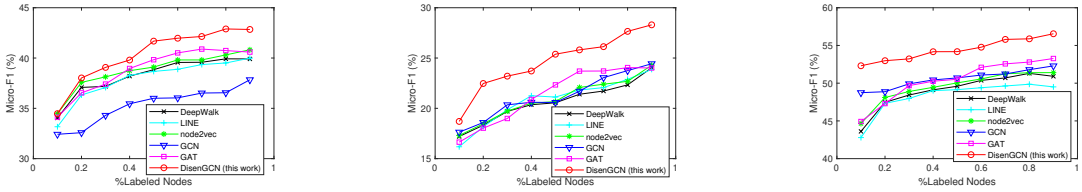
We report the results in Figure 2. GCN has relatively high Macro-F1 scores but low Micro-F1 scores, indicating that it is not robust to class imbalance and cannot handle the classes with few samples well. This is because the holistic approach taken by GCN tends to ignore the information provided by the minority of neighbors that are associated with the classes with few samples. On the other hand, GAT is likely suffering from over-fitting due to its parameterized attention mechanism, as we have observed that its performance on the validation sets is much higher than that on the test sets. Our approach, in comparison, consistently outperforms the best performing baselines by a significant margin, reaching around 10% to 20% relative improvement in most cases. This indicates that, by disentangling and preserving the factors behind edge formation, our approach can effectively address the aforementioned issues faced by GCN and GAT.

4.4. Disentangling Synthetic Graphs

To further investigate the behavior of DisenGCN, we generate synthetic graphs with various number of latent factors.



(a) Macro-F1(%), BlogCatalog. (c) Macro-F1(%), PPI. (e) Macro-F1(%), POS.



(b) Micro-F1(%), BlogCatalog. (d) Micro-F1(%), PPI. (f) Micro-F1(%), POS.

Figure 2. Macro-F1 and Micro-F1 scores on the multi-label classification tasks. Our approach consistently outperforms the best performing baselines by a large margin, reaching 10% to 20% relative improvement in most cases.

To generate a graph with K latent factors, we first generate K Erdős-Rényi random graphs, each of which has $1,000$ nodes and 16 communities. Two nodes in an Erdős-Rényi random graph are connected with probability p if they are in the same community, with probability q otherwise. We then generate the final synthetic graph with K latent factors by summing the adjacency matrices of the K Erdős-Rényi random graphs. We set q to $3e^{-5}$ to generate around 200 random edges, so as to ensure the graph is connected. For each choice of K, we tune p such that the average degree is between 39.5 and 40.5. The rows of the adjacency matrices are used as node features, and the ground-truth communities are used as labels, i.e., there are $16K$ classes and each node has K labels. We use $d = 64$ in this task. For fair comparison, we do not manually set the number of channels used by DisenGCN to K, but instead tune it as usual.

We vary the number of latent factors, and report the results in Table 3. From the results, we find that as the number of latent factors increases from 4 to 10, DisenGCN starts to achieve a greater relative improvement, which emphasizes the importance of disentangling the factors. However, when K is very large, i.e., $K > 12$, the synthetic graph becomes too challenging, and the relative improvement brought by DisenGCN starts to fall.

In Figure 3, we visualize the absolute values of the correlations between the elements of the 64-dimensional node representations learned by DisenGCN, on the synthetic graph with eight factors, using eight channels. DisenGCN's correlation plot exhibits eight clear diagonal blocks, indicating that the eight channels of DisenGCN are likely capturing mutually exclusive information.

4.5. Hyperparameter Sensitivity

In this subsection, we investigate the effect of the two hyper-parameters that are most important to DisenGCN: the number of channels, and the number of routing iterations. We use a single DisenConv layer here and run the experiments on a synthetic graph with eight latent factors (see section 4.4). The results on the other datasets follow a similar trend.

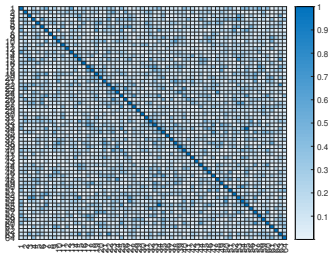
The results are reported in Figure 4. The results indicate that DisenGCN performs the best when the number of channels is around the actual number of latent factors, and routing for more iterations generally leads to better performance before saturation thanks to its convergence properties.

5. Related Work

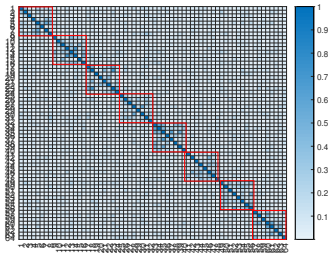
Graph neural networks (GNNs) (Gori et al., 2005; Scarselli et al., 2009), especially graph convolutional networks (Bruna et al., 2014; Henaff et al., 2015), have been attracting considerable attention lately, because of their remarkable success in various tasks, such as graph classification (Defferrard et al., 2016) and node classification (Kipf & Welling, 2017). The early attempts (Bruna et al., 2014; Henaff et al., 2015) to derive a graph convolutional layer were based on graph spectral theory, graph Fourier transformation (Shuman et al., 2013) in particular. Defferrard et al. (2016) then greatly reduced the computational cost by using polynomial spectral filters. Kipf & Welling (2017) made further simplification and suggested the usage of a linear filter. Along with spectral graph convolution, directly performing graph convolution in the spatial domain was also investigated by many researchers (Duvenaud et al., 2015; Atwood & Towsley, 2016; Hamilton et al., 2017). Later the attention mechanism (Bahdanau et al., 2015) was em-

Table 3. Micro-F1 scores on synthetic graphs generated with different numbers of latent factors.

	Number of latent factors						
Method	4	6	8	10	12	14	16
GCN	78.78 ± 1.52	65.73 ± 1.94	46.55 ± 1.55	37.37 ± 1.52	24.49 ± 1.03	18.14 ± 1.50	16.43 ± 0.92
GAT	83.77 ± 2.32	60.89 ± 3.75	45.88 ± 3.79	36.72 ± 3.58	24.77 ± 3.47	20.89 ± 3.57	19.53 ± 3.97
DisenGCN (this work)	**93.84** ± 1.12	**74.68** ± 1.92	**54.57** ± 1.79	**43.96** ± 1.45	**28.17** ± 1.22	**23.57** ± 1.28	**21.99** ± 1.34
Relative improvement	+12.02%	+13.62%	+17.23%	+17.63%	+13.73%	+12.83%	+12.6%



(a) GCN.



(b) DisenGCN (this work).

Figure 3. The absolute values of the correlations between the elements of the 64-dimensional representations learned by GCN and DisenGCN with eight channels, respectively, on a synthetic graph with eight latent factors. We can see that the eight channels of DisenGCN are likely capturing mutually exclusive information, because Figure 3b exhibits eight diagonal blocks (marked in red).

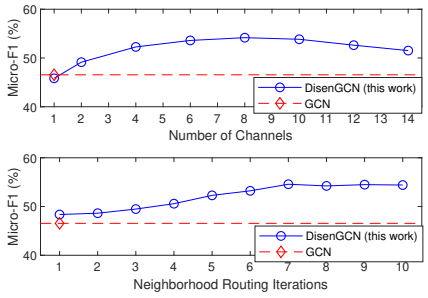


Figure 4. Hyper-parameter sensitivity of DisenGCN, using a single DisenConv layer, on synthetic graphs with eight latent factors.

ployed to adaptively specify weights to the neighbors of a node when performing spatial convolution (Veličković et al., 2018). Monti et al. (2017) proposed a unified framework that generalized the various graph convolutional networks.

The existing GNNs, however, cannot learn disentangled node representations. The existing methods typically convolute all the neighbors to obtain a node's representation, in a way oblivious to the different factors that may have caused the edges. They may, in fact, even further blur the boundary between the factors, and produce overly smoothed representations that are not desired for node classification, especially when the number of layers is increased (Li et al., 2018; Xu et al., 2018). On the other hand, Veličković et al. (2018) do notice that there can be edges caused by factors irrelevant to the task at hand, and thus use the attention mechanism to prune the irrelevant ones. Yet the attention mechanism remains a holistic approach with respective to the preserved factors, and therefore cannot disentangle the preserved ones.

Another line of works related to ours is the capsule neural network [5] (CapsNet) (Hinton et al., 2011), which replaces scalar-valued neurons with vector-valued ones, called capsules, to capture the different parts of a single instance, and uses dynamic routing (Sabour et al., 2017; Hinton et al., 2018) to group low-level capsules into high-level ones. In part inspired by dynamic routing, we propose neighborhood routing to cluster the neighbors of a node while disentangling the latent factors. Our approach generalizes dynamic routing to handle the scenario where there is an undetermined number of connected instances, each composed of several parts, instead of a single instance.

6. Conclusion

We have studied the problem of disentangling the factors behind the formation of a graph, and presented DisenGCN, a graph neural network that learns disentangled node representations. An interesting direction for future work is to investigate if the disentangled node representations can be leveraged to derive a single representation for the whole graph that can more comprehensively describe the graph.

[5] There are recent GNNs that are inspired by the concept of capsules (Verma & Zhang, 2018; Xinyi & Chen, 2019). The problems they address are different from ours. And they are proposed to learn whole graph representations for whole graph classification, while we focus on node representations and node-related tasks.

Acknowledgements

This work was supported in part by National Program on Key Basic Research Project (No. 2015CB352300), National Natural Science Foundation of China (No. 61772304, No. 61521002, No. 61531006, No. U1611461), China Postdoctoral Science Foundation (No. BX201700136), Beijing Academy of Artificial Intelligence (BAAI), the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST. Peng Cui, Xin Wang, and Wenwu Zhu supervised the project jointly, and are co-corresponding authors. All opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Alemi, A. A., Fischer, I., Dillon, J. V., and Murphy, K. Deep variational information bottleneck. In *Proceedings of ICLR 2017*, 2017.

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *Proceedings of NIPS 2016*, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR 2015*, 2015.

Belkin, M., Niyogi, P., and Sindhwani, V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 2006.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

Bergstra, J., Yamins, D., and Cox, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of ICML 2013*, 2013.

Breitkreutz, B.-J., Stark, C., Reguly, T., Boucher, L., Breitkreutz, A., Livstone, M., Oughtred, R., Lackner, D. H., Bähler, J., Wood, V., Dolinski, K., and Tyers, M. The BioGRID interaction database: 2008 update. *Nucleic Acids Research*, 2008.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. In *Proceedings of ICLR 2014*, 2014.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. In *Proceedings of ICLR 2017*, 2017.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of NIPS 2016*, 2016.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. In *eprint arXiv:1702.08608*, 2017.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of NIPS 2015*, 2015.

Firth, J. R. A synopsis of linguistic theory. *Studies in linguistic analysis*, pp. 1–32, 1930–1955.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings of IJCNN 2005*, 2005.

Granovetter, M. S. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, 1973.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of KDD 2016*, 2016.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of NIPS 2017*, 2017.

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

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *Proceedings of ICLR 2016*, 2016.

Hinton, G. E., Krizhevsky, A., and Wang, S. D. Transforming auto-encoders. In *Proceedings of ICANN 2011*, 2011.

Hinton, G. E., Sabour, S., and Frosst, N. Matrix capsules with EM routing. In *Proceedings of ICLR 2018*, 2018.

Kim, H. and Mnih, A. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of ICLR 2015*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR 2017*, 2017.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of AAAI 2018*, 2018.

Lipton, Z. C. The mythos of model interpretability. *ACM Queue*, 2018.

Lu, Q. and Getoor, L. Link-based classification. In *Proceedings of ICML 2003*, 2003.

Ma, J., Cui, P., and Zhu, W. DepthLGP: Learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of AAAI 2018*, 2018.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of CVPR 2017*, 2017.

Perozzi, B., Al-Rfou, R., and Skiena, S. DeepWalk: Online learning of social representations. In *Proceedings of KDD 2014*, 2014.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Proceedings of NIPS 2017*, 2017.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 2008.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 2013.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. LINE: Large-scale information network embedding. In *Proceedings of WWW 2015*, 2015.

Tang, L. and Liu, H. Relational learning via latent social dimensions. In *Proceedings of KDD 2009*, 2009.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *Proceedings of ICLR 2018*, 2018.

Verma, S. and Zhang, Z.-L. Graph capsule convolutional neural networks. In *Joint ICML and IJCAI WCB Workshop*, 2018.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. Deep learning via semi-supervised embedding. *Neural Networks: Tricks of the Trade*, 2012.

Xinyi, Z. and Chen, L. Capsule graph neural network. In *Proceedings of ICLR 2019*, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *Proceedings of ICML 2018*, 2018.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of ICML 2016*, 2016.

Zhu, X., Ghahramani, Z., and Lafferty, J. D. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of ICML 2003*, 2003.