Future Generation Computer Systems 117 (2021) 155–168

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Graph convolutional networks for graphs containing missing features

Hibiki Taguchi [a,b], Xin Liu [b,c,*], Tsuyoshi Murata [a,c]

[a] Department of Computer Science, Tokyo Institute of Technology, Japan
[b] Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Japan
[c] AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory, Japan

ARTICLE INFO

Article history:
Received 8 July 2020
Received in revised form 29 October 2020
Accepted 17 November 2020
Available online 1 December 2020

Keywords:
Graph convolutional neural network
GCN
Missing data
Incomplete data
Graph embedding
Network representation learning

ABSTRACT

Graph Convolutional Network (GCN) has experienced great success in graph analysis tasks. It works by smoothing the node features across the graph. The current GCN models overwhelmingly assume that the node feature information is complete. However, real-world graph data are often incomplete and containing missing features. Traditionally, people have to estimate and fill in the unknown features based on imputation techniques and then apply GCN. However, the process of feature filling and graph learning are separated, resulting in degraded and unstable performance. This problem becomes more serious when a large number of features are missing. We propose an approach that adapts GCN to graphs containing missing features. In contrast to traditional strategy, our approach integrates the processing of missing features and graph learning within the same neural network architecture. Our idea is to represent the missing data by Gaussian Mixture Model (GMM) and calculate the expected activation of neurons in the first hidden layer of GCN, while keeping the other layers of the network unchanged. This enables us to learn the GMM parameters and network weight parameters in an end-to-end manner. Notably, our approach does not increase the computational complexity of GCN and it is consistent with GCN when the features are complete. We demonstrate through extensive experiments that our approach significantly outperforms the imputation based methods in node classification and link prediction tasks. We show that the performance of our approach for the case with a low level of missing features is even superior to GCN for the case with complete features.

1. Introduction

Graphs are used in many branches of science as a way to represent the patterns of connections between the components of complex systems, including social analysis, product recommendation, web search, disease identification, brain function analysis, and many more.

In recent years there is a surge of interest in learning on graph data. Graph embedding [1–3] aims to learn low-dimensional vector representations for nodes or edges. The learned representations encode structural and semantic information transcribed from the graph and can be used directly as the features for downstream graph analysis tasks. Representative works on graph embedding include random walk and skip-gram model based methods [4], matrix factorization based approaches [5,6], edge reconstruction based methods [7], and deep learning based algorithms [8,9], etc.

Meanwhile, graph neural network (GNN) [10–12], as a type of neural network architectures that can operate on graph structure, has achieved superior performance in graph analysis and shown promise in various applications such as visual question answering [13], point clouds classification and segmentation [14], fraud detection [15], machine translation [16], molecular fingerprints prediction [17], protein interface prediction [18], topic modeling [19], and social recommendation [20].

Among various kinds of GNNs, graph convolutional network (GCN) [21], a simplified version of spectral graph convolutional networks [22], has attracted a large amount of attention. GCN and its subsequent variants can be interpreted as smoothing the node features in the neighborhoods guided by the graph structure, and have experienced great success in graph analysis tasks, such as node classification [21], graph classification [23], link prediction [24], graph similarity estimation [25], node ranking [26,27], and community detection [28,29].

The current GCN-like models assume that the node feature information is complete. However, real-world graph data are often incomplete and containing missing node features. Much of the missing features arise from the following sources. First, some features can be missing because of mechanical/electronic failures or human errors during the data collection process. Secondly, it

* Corresponding author at: Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Japan.
E-mail addresses: taguchi@net.c.titech.ac.jp (H. Taguchi), xin.liu@aist.go.jp (X. Liu), murata@c.titech.ac.jp (T. Murata).

https://doi.org/10.1016/j.future.2020.11.016
0167-739X/

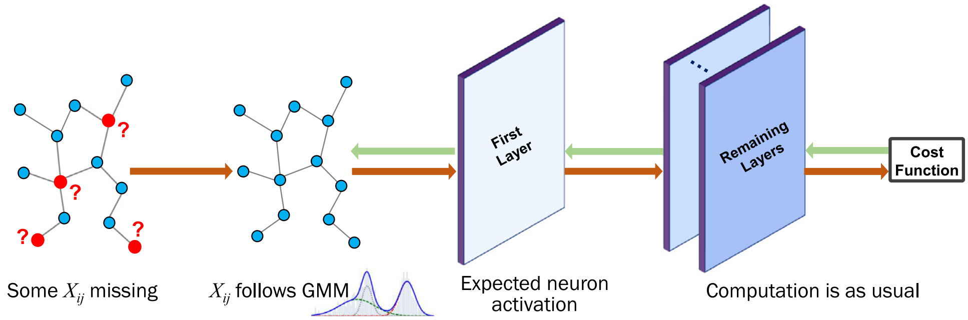


Fig. 1. The architecture of our model.

can be prohibitively expensive or even impossible to collect the complete data due to its large size. For example, social media companies such as Twitter and Facebook have restricted the crawlers to collect the whole data. Thirdly, we cannot obtain sensitive personal information. In a social network, many users are unwilling to provide information such as address, nationality, and age to protect personal privacy. Finally, graphs are dynamic in nature, and thus newly joined nodes often have very little information. All these aspects result in graphs containing missing features.

To deal with the above problem, the traditional strategy is to estimate and fill in the unknown values before applying GCN. For this purpose, people have proposed imputation techniques such as mean imputation [30,31], soft imputation based on singular value decomposition [32], and machine learning methods such as *k-NN* model [33], random forest [34], autoencoder [35,36], generative adversarial network (GAN) [37–39]. However, the process of feature filling and graph learning are separated. Our experiments reveal that this strategy results in degraded and unstable performance, especially when a large number of features are missing.

In this paper, we propose an approach that can adapt GCN to graphs containing missing features. In contrast to traditional strategy, our approach integrates the processing of missing features and graph learning within the same neural network architecture and thus can enhance the performance. Our approach is motivated by Gaussian Mixture Model Compensator (GMMC) [40] for processing missing data in neural networks. The main idea is to represent the missing data by Gaussian Mixture Model (GMM) and calculate the expected activation of neurons in the first hidden layer, while keeping the other layers of the network architecture unchanged (Fig. 1). Although this idea is implemented in simple neural networks such as autoencoder and multilayer perceptron, it has not yet been extended to complex neural networks such as RNN, CNN, GNN, and sequence-to-sequence models. The main reason is due to the difficulty in unifying the representation of missing data and calculation of the expected activation of neurons. In particular, simply using GMM to represent the missing data will even complicate the network architecture, which hinders us from calculating the expected activation in closed form. We propose a novel way to unify the representation of missing features and calculation of the expected activation of the first layer neurons in GCN. Specifically, we skillfully represent the missing features by introducing only a small number of parameters in GMM and derive the analytic solution of the expected activation of neurons. As a result, our approach can arm GCN against missing features without increasing the computational complexity and our approach is consistent with GCN when the features are complete.

Our contributions are summarized as follows:

- We propose an elegant and unified way to transform the incomplete features to variables that follow mixtures of Gaussian distributions.
- Based on the transformation, we derive the analytic solution to calculate the expected activation of neurons in the first layer of GCN.
- We propose the whole network architecture for learning on graphs containing missing features. We prove that our model is consistent with GCN when the features are complete.
- We perform extensive experiments and demonstrate that our approach significantly outperforms imputation based methods.

The rest of the paper is organized as follows. The next section summarizes the recent literature on GCN and methods for processing missing data. Section 3 reviews GCN. Section 4 introduces our approach. Section 5 reports experiment results. Finally, Section 6 presents our concluding remarks.

2. Related work

2.1. Graph convolutional networks

GNNs are deep learning models aiming at addressing graph-related tasks [10–12]. Among various kinds of GNNs, GCN [21], which simplifies the previous spectral graph convolutional networks [22] by restricting the filters to operate in one-hop neighborhood, has attracted a large amount of attention due to its simplicity and high performance. GCN can be interpreted as smoothing the node features in the neighborhoods, and this model achieves great success in the node classification task.

There are a series of works following GCN. GAT extends GCN by imposing the attention mechanism on the neighboring weight assignment [41]. AGCN learns hidden structural relations unspecified by the graph adjacency matrix and constructs a residual graph adjacency matrix [42]. TO-GCN utilizes potential information by jointly refining the network topology [43]. GCLN introduces ladder-shape architecture to increase the depth of GCN while overcoming the over-smoothing problem [44]. MixHop introduces higher-order feature aggregation, which enables us to capture mixing neighbors' information [45]. There is also work on extending GCN to handle noisy and sparse node features [46].

Training GCN usually requires to save the whole graph data into memory. To solve this problem, sampling strategy [47] and batch training [48] are proposed. Moreover, FastGCN reduces the complexity of GCN through successively removing nonlinearities and collapsing weight matrices between consecutive layers [47].

While achieving excellent performance in graph analysis tasks, GCN is known to be vulnerable to adversarial attacks [49,50]. To address this problem, researchers have proposed robust models such as RGCN that adopts Gaussian distributions as the hidden representations of nodes in each convolutional layer [51] and a new learning principle that improves the robustness of GCN [52].

We note that all of the models mentioned above assume that the node feature information is complete.

2.2. Learning with missing data

Incomplete and missing data is common in real-world applications. Methods for handling such data can be categorized into two classes. The first class completes the missing data before using conventional machine learning algorithms. Imputation techniques are widely used for data completion, such as mean imputation [30], matrix completion via matrix factorization [53] and singular value decomposition (SVD) [32], and multiple imputation [54,55]. Machine learning models are also employed to estimate missing values, such as *k-NN* model [33], random forest [34], autoencoder [35,36], generative adversarial network (GAN) [37–39]. However, imputation methods are not always competent to handle this problem, especially when the missing rate is high [56].

The second class directly trains a model based on the missing data without any imputation, and there are a range of research along this line. Che et al. improve Gated Recurrent Unit (GRU) to address the multivariate time series missing data [56]. Jiang et al. divide missing data into complete sub-data and then applied them to ensemble classifiers [57]. Pelckmans et al. modify the loss function of Support Vector Machine (SVM) to address the uncertainty issue arising from missing data [58]. Moreover, there are some research on building improved machine learning models such as logistic regression [59], kernel methods [60,61], and autoencoder and multilayer perceptron [40] on top of representing missing values with probabilistic density.

To the best of our knowledge, there is no related work on how to adapt GNNs to graphs containing missing features. Hence, we propose an approach to address this problem.

3. Preliminaries

In this section, we briefly review GCN, which paves the way for the next discussion.

3.1. Notations

Let us consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i \mid i = 1, \ldots, N\}$ is the node set, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. $\mathbf{A} \in \mathbb{R}^{N\times N}$ denotes the adjacency matrix, where $A_{ij} = A_{ji}$, $A_{ij} = 0$ if $(v_i, v_j) \notin \mathcal{E}$, and $A_{ij} > 0$ if $(v_i, v_j) \in \mathcal{E}$. $\mathbf{X} \in \mathbb{R}^{N\times D}$ is the node feature matrix and D is the number of features. $\mathcal{S} \subseteq \{(i, j) | i = 1, \ldots, N, j = 1, \ldots, D\}$ is a set for the index of missing features: $\forall(i, j) \in \mathcal{S}$, X_{ij} is not known.

3.2. Graph convolutional network

GCN-like models consist of aggregators and updaters. The aggregator gathers information guided by the graph structure, and the updater updates nodes' hidden states according to the gathered information. Specifically, the graph convolutional layer is based on the following equation:

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{L}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{1}$$

where $\mathbf{L} \in \mathbb{R}^{N\times N}$ is the aggregation matrix, $\mathbf{H}^{(l)} = (\boldsymbol{h}_1^{(l)}, \ldots, \boldsymbol{h}_N^{(l)})^\top \in \mathbb{R}^{N\times D^{(l)}}$ is the node representation matrix in lth layer, $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{W}^{(l)} \in \mathbb{R}^{D^{(l)}\times D^{(l+1)}}$ is the trainable weight matrix in lth layer, and $\sigma(\cdot)$ is the activation function such as ReLU, LeakyReLU, and ELU.

GCN [21] adopts the re-normalized graph Laplacian $\hat{\mathbf{A}}$ as the aggregator:

$$\mathbf{L} = \hat{\mathbf{A}} \triangleq \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}, \tag{2}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}} = \mathrm{diag}(\sum_i \tilde{A}_{1i}, \ldots, \sum_i \tilde{A}_{Ni})$. Empirically, 2-layer GCN with ReLU activation shows the best performance on node classification, defined as:

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \mathrm{softmax}(\mathbf{L}(\mathrm{ReLU}(\mathbf{L}\mathbf{X}\mathbf{W}^{(0)}))\mathbf{W}^{(1)}) \tag{3}$$

4. Proposed approach

In this section, we propose our approach for training GCN on graphs containing missing features. We follow GMMC [40] to represent the missing data by GMM and calculate the expected activation of neurons in the first hidden layer. Although this idea is implemented in simple neural networks such as autoencoder and multilayer perceptron, it has not yet been extended to complex neural networks such as RNN, CNN, GNN, and sequence-to-sequence models. The principal difficulty lies in the fact that simply using GMM to represent the missing data will even complicate the network architecture, which hinders us from calculating the expected activation in closed form. In the following, we propose a novel way to unify the representation of missing features and calculation of the expected activation of the first layer neurons in GCN. Specifically, we skillfully represent the missing features by introducing only a small number of parameters in GMM and derive the analytic solution of the expected activation, enabling us to integrate the processing of missing features and graph learning within the same neural network architecture.

4.1. Representing node features using GMM

Suppose $\boldsymbol{X} \in \mathbb{R}^D$ is a random variable for node features. We assume $\boldsymbol{X}$ is generated from the mixture of (degenerate) Gaussians:

$$\boldsymbol{X} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}^{[k]}, \boldsymbol{\Sigma}^{[k]}) \tag{4}$$

$$\boldsymbol{\mu}^{[k]} = (\mu_1^{[k]}, \ldots, \mu_D^{[k]})^\top \tag{5}$$

$$\boldsymbol{\Sigma}^{[k]} = \mathrm{diag}\left((\sigma_1^{[k]})^2, \ldots, (\sigma_D^{[k]})^2\right), \tag{6}$$

where K is the number of components, π_k is the mixing parameter with the constraint that $\sum_k \pi_k = 1$, $\mu_j^{[k]}$ and $(\sigma_j^{[k]})^2$ denote the jth element of mean and variance of the kth Gaussian component, respectively. Further, we introduce a mean matrix $\mathbf{M}^{[k]} \in \mathbb{R}^{N\times D}$ and a variance matrix $\mathbf{S}^{[k]} \in \mathbb{R}^{N\times D}$ for each component as:

$$M_{ij}^{[k]} = \begin{cases} \mu_j^{[k]} & \text{if } X_{ij} \text{ is missing;} \\ X_{ij} & \text{otherwise} \end{cases} \tag{7}$$

$$S_{ij}^{[k]} = \begin{cases} (\sigma_j^{[k]})^2 & \text{if } X_{ij} \text{ is missing;} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

This enables us to represent each X_{ij} with:

$$X_{ij} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(M_{ij}^{[k]}, S_{ij}^{[k]}), \tag{9}$$

no matter whether X_{ij} is missing or not. Thus, we skillfully transform the input of our model into fixed $\mathbf{A}$ and unfixed X_{ij} that follows the mixture of Gaussian distributions. The next layer is based on calculation of the expected activation of neurons, which is discussed in the next section.

4.2. The expected activation of neurons

Let us first identify some symbols that will be used. Suppose $x \sim F_x$ is a random variable and F_x is the probability density function. We define

$$\sigma[x] \triangleq \sigma[F_x] \triangleq \mathbb{E}[\sigma(x)], \tag{10}$$

which is the expected value of σ activation on x.

Theorem 1. *Let $x \sim \mathcal{N}(\mu, \sigma^2)$. Then:*

$$\mathrm{ReLU}[\mathcal{N}(\mu, \sigma^2)] = \sigma \mathrm{NR}\left(\frac{\mu}{\sigma}\right), \tag{11}$$

where

$$\mathrm{NR}(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) + \frac{z}{2}\left(1 + \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right)\right) \tag{12}$$

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt. \tag{13}$$

Proof. Please see [40] for a proof. □

Lemma 2. *Let $X_{ij} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(M_{ij}^{[k]}, S_{ij}^{[k]})$. Given the aggregation matrix $\mathbf{L}$ and the weight matrix $\mathbf{W}$, then:*

$$\mathrm{ReLU}[(\mathbf{LXW})_{ij}] = \sum_{k=1}^{K} \pi_k \sqrt{\hat{S}_{ij}^{[k]}} \mathrm{NR}\left(\frac{\hat{M}_{ij}^{[k]}}{\sqrt{\hat{S}_{ij}^{[k]}}}\right) \tag{14}$$

$$\begin{aligned}\mathrm{LeakyReLU}[(\mathbf{LXW})_{ij}] = \sum_{k=1}^{K} \pi_k \Bigg(&\sqrt{\hat{S}_{ij}^{[k]}} \mathrm{NR}\left(\frac{\hat{M}_{ij}^{[k]}}{\sqrt{\hat{S}_{ij}^{[k]}}}\right) \\ &- \alpha \sqrt{\hat{S}_{ij}^{[k]}} \mathrm{NR}\left(-\frac{\hat{M}_{ij}^{[k]}}{\sqrt{\hat{S}_{ij}^{[k]}}}\right)\Bigg),\end{aligned} \tag{15}$$

where $\odot$ is element-wise multiplication, α is the negative slope parameter of LeakyReLU *activation, and*

$$\hat{\mathbf{M}}^{[k]} = \mathbf{L}\mathbf{M}^{[k]}\mathbf{W} \tag{16}$$

$$\hat{\mathbf{S}}^{[k]} = (\mathbf{L} \odot \mathbf{L})\mathbf{S}^{[k]}(\mathbf{W} \odot \mathbf{W}). \tag{17}$$

Proof. The element of matrix **LXW** can be expressed as:

$$(\mathbf{LXW})_{ij} = \sum_{d=1}^{D} \sum_{n=1}^{N} L_{in} X_{nd} W_{dj} \tag{18}$$

Based on the property of Gaussian distribution, $(\mathbf{LXW})_{ij}$ also follows a mixture of Gaussian distributions as:

$$\sum_{d=1}^{D} \sum_{n=1}^{N} L_{in} X_{nd} W_{dj} \tag{19}$$

$$\sim \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\sum_{d=1}^{D} \sum_{n=1}^{N} L_{in} M_{nd}^{[k]} W_{dj}, \sum_{d=1}^{D} \sum_{n=1}^{N} L_{in}^2 S_{nd}^{[k]} W_{dj}^2\right) \tag{20}$$

$$= \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\{\mathbf{L}\mathbf{M}^{[k]}\mathbf{W}\}_{ij}, \{(\mathbf{L} \odot \mathbf{L})\mathbf{S}^{[k]}(\mathbf{W} \odot \mathbf{W})\}_{ij}\right) \tag{21}$$

$$= \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\hat{M}_{ij}^{[k]}, \hat{S}_{ij}^{[k]}\right). \tag{22}$$

Finally, using the result of Theorem 1, we can derive Eq. (14) as:

$$\mathrm{ReLU}[(\mathbf{LXW})_{ij}] = \sum_{k=1}^{K} \pi_k \mathrm{ReLU}\left[\mathcal{N}(\hat{M}_{ij}^{[k]}, \hat{S}_{ij}^{[k]})\right] \tag{23}$$

$$= \sum_{k=1}^{K} \pi_k \sqrt{\hat{S}_{ij}^{[k]}} \mathrm{NR}\left(\frac{\hat{M}_{ij}^{[k]}}{\sqrt{\hat{S}_{ij}^{[k]}}}\right). \tag{24}$$

Eq. (15) can be proved similarly and the proof is omitted due to lack of space. □

Thus, we can calculate the expected activation of neurons for the first layer according to Lemma 2. Calculation of the subsequent layers remains unchanged.

Algorithm 1 Algorithm of GCNmf

Input: Aggregation matrix $\mathbf{L}$, node feature matrix $\mathbf{X}$ (with some missing elements), the number of layers L, the number of Gaussian components K
Output: According to the task

```
Initialize:
    (π_k, μ^[k], Σ^[k]) are optimized by EM algorithm w.r.t X
while not converged do
    H^(1) ← ReLU[LXW^(0)]                              ▷ Lemma 2
    for l ← 2, ..., L − 1 do
        H^(l) ← ReLU(LH^(l−1) W^(l−1))
    end for
    Z ← final_layer(LH^(L−1) W^(L−1))
    ℒ ← loss(Z)
    Minimize ℒ and update GMM parameters and network parameters with a gradient descent optimization algorithm
end while
```

4.3. The network architecture

Our approach is named GCNmf. We illustrate the model architecture in Fig. 1 and present the pseudo-code in Algorithm 1, with additional explanations below.

- *Initialize the hyper-parameters*
  The additional hyper-parameters include the number of layers L, the number of Gaussian components K.
- *Initialize the model parameters*
  The model parameters include GMM parameters (π_k, $\boldsymbol{\mu}^{[k]}$, $\boldsymbol{\Sigma}^{[k]}$) and conventional network parameters. GMM parameters are initialized by EM algorithm [62] that explores the data density.[1]
- *Forward propagation*
  Calculate the first layer according to Lemma 2, and calculate the other layers as usual.
- *Backward propagation*
  Apply a gradient descent optimization algorithm to jointly learn the GMM parameters and network parameters by minimizing a cost function that is created based on a specific task.
- *Consistency*
  GCNmf is consistent with GCN when the features are complete. Suppose $\mathcal{S} = \emptyset$. It follows that $\sigma[(\mathbf{LXW})_{ij}] = \sigma\left((\mathbf{LXW})_{ij}\right)$ (see the proof below). In other words, the computation of the first layer based on expected activations is equivalent to that based on fixed features. Thus, GCNmf degenerates to GCN when the features are complete.

[1] The algorithm implementation is provided by scikit-learn: https://scikit-learn.org/.

Table 1
Statistics of datasets.

	Cora	Citeseer	AmaPhoto	AmaComp
#Nodes	2708	3327	7650	13,752
#Edges	5429	4732	143,663	287,209
#Features	1433	3703	745	767
#Classes	7	6	8	10
#Train nodes	140	120	320	400
#Validation nodes	500	500	500	500
#Test nodes	1000	1000	6830	12,852
Feature sparsity	98.73%	99.15%	65.26%	65.16%

Proof. Take ReLU activation as an example. When $\mathcal{S} = \emptyset$, we have $X_{ij} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(X_{ij}^{[k]}, 0)$, $\hat{S}_{ij}^{[k]} = 0$, and $\hat{M}_{ij}^{[k]} = (\mathbf{LXW})_{ij}$. Thus,

$$\mathrm{ReLU}[(\mathbf{LXW})_{ij}] = \sum_{k=1}^{K} \pi_k \sqrt{\hat{S}_{ij}^{[k]}} \mathrm{NR}\left(\frac{\hat{M}_{ij}^{[k]}}{\sqrt{\hat{S}_{ij}^{[k]}}}\right) \tag{25}$$

$$= \sum_{k=1}^{K} \pi_k \lim_{\epsilon \to 0+} \left(\sqrt{\epsilon}\,\mathrm{NR}\left(\frac{(\mathbf{LXW})_{ij}}{\sqrt{\epsilon}}\right)\right) \tag{26}$$

$$= \sum_{k=1}^{K} \pi_k \lim_{\epsilon \to 0+} \left(\sqrt{\frac{\epsilon}{2\pi}} \exp\left(-\frac{(\mathbf{LXW})_{ij}^2}{2\epsilon}\right) + \frac{(\mathbf{LXW})_{ij}}{2}\left(1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{(\mathbf{LXW})_{ij}}{\sqrt{2\epsilon}}} \exp(-t^2)dt\right)\right) \tag{27}$$

$$= \begin{cases} 0 & \text{if } (\mathbf{LXW})_{ij} \le 0 \\ (\mathbf{LXW})_{ij} & \text{otherwise} \end{cases} \tag{28}$$

$$= \mathrm{ReLU}\big((\mathbf{LXW})_{ij}\big), \tag{29}$$

where we have used $\int_0^{+\infty} \exp(-x^2)dx = \frac{\sqrt{\pi}}{2}$ and $\int_0^{-\infty} \exp(-x^2)dx = -\frac{\sqrt{\pi}}{2}$ in Eq. (28). □

Time complexity

In the following, we analyze the time complexity of the forward propagation. Note that GCNmf modifies the original GCN in the first layer, where the calculation of Eq. (1) is replaced by Eq. (14) or (15). We assume that $\mathbf{L}$ is a sparse matrix. The calculation of Eq. (1) takes $\mathcal{O}(|\mathcal{E}|D + NDD^{(1)})$ complexity [48].

Eq. (14) or (15) requires calculation of Eqs. (16) and (17). The complexity of Eq. (16) for all k is $\mathcal{O}(K(|\mathcal{E}|D + NDD^{(1)}))$. The complexity of Eq. (17) for all k is $\mathcal{O}(|\mathcal{E}|) + \mathcal{O}(DD^{(1)}) + \mathcal{O}(K(|\mathcal{E}|D + NDD^{(1)}))$, where the first two terms are for $(\mathbf{L} \odot \mathbf{L})$ and $(\mathbf{W} \odot \mathbf{W})$, respectively. Given $\hat{\mathbf{M}}^{[k]}$ and $\hat{\mathbf{S}}^{[k]}$, Eq. (14) or (15) takes $\mathcal{O}(KND^{(1)})$ time for all i, j.

Putting them all together, the total complexity of the first layer of GCNmf is $\mathcal{O}(K(|\mathcal{E}|D + NDD^{(1)})) + \mathcal{O}(|\mathcal{E}|) + \mathcal{O}(DD^{(1)})$

Table 2
The accuracy results for node classification task in Cora.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	MEAN	80.96	80.41	79.48	78.51	77.17	73.66	56.24	20.49	13.22
	K-NN	80.45	80.10	78.86	77.26	75.34	71.55	66.44	40.99	15.11
	MFT	80.70	80.03	78.97	78.12	76.43	71.33	45.82	27.22	23.98
	SoftImp	80.74	80.32	79.63	78.68	77.32	74.26	70.36	64.93	41.20
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	80.68	80.43	79.74	79.27	76.12	73.70	68.31	60.92	45.89
	VAE	80.91	80.47	79.18	78.38	76.84	72.41	50.79	18.12	13.27
	GAIN	80.43	79.72	78.35	77.01	75.31	72.50	70.34	64.85	58.87
	GINN	80.77	80.01	78.77	76.67	74.44	70.58	58.60	18.04	13.19
	GCNmf	**81.70**	**81.66**	**80.41**	**79.52**	**77.91**	**76.67**	**74.38**	**70.57**	**63.49**
	Performance gain (%)	0.91 | 1.58	1.48 | 2.43	0.84 | 2.63	0.32 | 3.72	0.76 | 4.66	3.25 | 8.63	5.71 | 62.33	8.69 | 291.19	7.85 | 381.35
Biased randomly missing	Mean	81.22	80.37	78.95	77.46	75.94	72.44	53.14	20.39	13.40
	K-NN	80.75	79.94	78.33	77.17	75.62	72.66	67.05	54.71	15.13
	MFT	80.75	75.01	56.28	55.76	43.81	29.31	25.88	21.79	21.07
	SoftImp	81.04	80.30	78.80	78.50	75.99	73.65	61.37	60.06	46.38
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	80.90	80.10	78.79	77.54	74.66	71.04	65.28	56.65	44.30
	VAE	80.92	80.33	78.86	77.25	75.74	69.29	53.53	18.11	13.27
	GAIN	80.68	79.62	78.54	77.41	75.84	73.82	69.18	63.99	59.41
	GINN	80.86	80.10	78.45	76.80	74.60	72.08	65.72	50.08	13.22
	GCNmf	**82.29**	**81.09**	**80.00**	**79.23**	**77.33**	**76.19**	**72.57**	**68.19**	**65.73**
	Performance gain (%)	1.32 | 2.00	0.90 | 8.11	1.33 | 42.15	0.93 | 42.09	1.76 | 76.51	3.21 | 159.95	4.90 | 180.41	6.56 | 276.53	10.64 | 397.20
Structurally missing	MEAN	80.92	80.40	79.05	77.73	75.22	70.18	56.30	25.56	13.86
	K-NN	80.76	80.26	78.63	77.51	74.51	70.86	63.29	37.97	13.95
	MFT	80.91	80.34	78.93	77.48	74.47	69.13	52.65	29.96	17.05
	SoftImp	79.71	69.47	69.31	52.53	44.71	40.07	36.68	28.51	27.90
	MICE	80.92	80.40	79.05	77.72	75.22	70.18	56.30	25.56	13.86
	MissForest	80.48	79.88	78.54	76.93	73.88	68.13	54.29	30.82	14.05
	VAE	80.63	79.98	78.57	77.42	74.69	69.95	60.71	36.59	17.27
	GAIN	80.53	79.78	78.36	77.09	74.25	69.90	61.33	41.09	18.43
	GINN	80.85	80.27	78.88	77.35	74.76	70.58	59.45	29.15	13.92
	GCNmf	**81.65**	**80.77**	**80.67**	**79.24**	**77.43**	**75.97**	**72.69**	**68.00**	**55.64**
	Performance gain (%)	0.90 | 2.43	0.46 | 16.27	2.05 | 16.39	1.94 | 50.85	2.94 | 73.18	7.21 | 89.59	14.85 | 98.17	65.49 | 166.04	99.43 | 301.44
RGCN		60.29	34.12	24.80	18.62	16.04	13.88	13.89	13.70	13.60
GCN		81.49								
GCN w/o node features		63.22								

Table 3
The accuracy results for node classification task in Citeseer.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	MEAN	69.88	69.62	68.97	65.12	54.62	37.39	18.29	12.28	11.88
	K-NN	69.84	69.38	68.69	67.18	62.64	54.75	32.20	14.84	12.73
	MFT	69.70	69.51	68.74	65.31	60.56	41.53	34.10	17.26	19.29
	SoftImp	69.63	69.34	69.23	68.47	66.35	**65.53**	**60.86**	52.23	31.08
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	69.80	69.39	68.54	64.13	50.91	29.62	18.45	12.49	11.00
	GAIN	69.64	68.88	67.56	65.97	63.86	60.74	55.77	52.05	42.73
	GINN	70.07	69.79	68.87	68.14	63.21	43.61	20.74	13.26	11.31
	GCNmf	**70.93**	**70.82**	**69.84**	**68.83**	**67.03**	64.78	60.70	**55.38**	**47.78**
	Performance gain (%)	1.23 – 1.87	1.48 – 2.82	0.88 – 3.37	0.53 – 7.33	1.02 – 31.66	−1.14 – 118.70	−0.26 – 231.88	6.03 – 350.98	11.82 – 334.36
Biased randomly missing	Mean	69.98	68.95	67.91	65.87	60.33	40.68	25.45	14.01	13.32
	K-NN	70.04	68.87	68.88	67.38	64.47	62.45	52.66	32.60	12.64
	MFT	69.88	67.68	63.17	45.49	25.99	20.22	20.82	18.53	18.30
	SoftImp	69.83	67.36	68.36	67.49	64.26	62.38	58.45	**55.63**	32.95
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	70.05	69.13	68.21	63.44	55.71	38.55	21.98	13.34	11.17
	GAIN	69.81	68.76	68.38	66.83	64.05	62.15	58.31	52.14	42.18
	GINN	69.96	69.60	69.63	68.67	64.93	62.14	55.01	31.37	12.91
	GCNmf	**71.01**	**69.99**	**69.96**	**68.89**	**66.30**	**64.67**	**61.06**	54.70	**46.14**
	Performance gain (%)	1.37 – 1.72	0.56 – 3.90	0.47 – 10.75	0.32 – 51.44	2.11 – 155.10	3.55 – 219.83	4.47 – 193.28	−1.67 – 310.04	9.39 – 313.07
Structurally missing	MEAN	69.55	68.31	**67.30**	65.18	53.64	34.07	18.56	13.19	11.30
	K-NN	69.67	67.33	66.09	63.29	56.86	31.27	19.51	13.75	11.21
	MFT	69.84	68.21	66.67	63.02	51.08	34.29	16.81	14.34	15.75
	SoftImp	44.06	27.92	25.83	25.13	25.59	23.99	25.41	22.83	20.13
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	69.63	68.07	66.34	64.33	60.46	54.37	40.71	23.14	17.20
	GAIN	69.47	67.86	65.88	63.96	59.96	54.24	41.21	25.31	17.89
	GINN	69.64	67.88	66.24	63.71	55.76	40.20	18.63	13.23	12.32
	GCNmf	**70.44**	**68.56**	66.57	**65.39**	**63.44**	**60.04**	**56.88**	**51.37**	**39.86**
	Performance gain (%)	0.86 – 59.87	0.37 – 145.56	−1.08 – 157.72	0.32 – 160.21	4.93 – 147.91	10.43 – 150.27	38.02 – 238.37	102.96 – 289.46	98.01 – 255.58
RGCN		34.37	20.69	14.16	12.15	12.01	12.34	14.36	11.97	12.57
GCN		70.65								
GCN w/o node features		40.55								

$+\ \mathcal{O}(K(|\mathcal{E}|D + NDD^{(1)})) + \mathcal{O}(KND^{(1)}) = \mathcal{O}(K(|\mathcal{E}|D + NDD^{(1)}))$. Since the number of components K is usually small, the forward propagation of GCNmf has the same complexity as GCN.

5. Experiments

We conducted experiments on the node classification task and link prediction task to answer the following questions:

- Does GCNmf agree with our intuition and perform well?
- Where do imputation based methods fail?
- Is GCNmf sensitive to the hyper-parameters?
- Is GCNmf computationally expensive?

In the following, we first explain experimental settings in detail, including baselines and datasets. After that, we discuss the results.

Datasets. We did experiments on four real-world graph datasets that are commonly used. Descriptions of these graphs are as follows and Table 1 summarizes their statistics.

- Cora and Citeseer [63]: The citation graphs, where nodes are documents and edges are citation links. Node features are bag-of-words representations of documents. Each node is associated with a label representing the topic of documents.
- AmaPhoto and AmaComp [64]: The product co-purchase graphs, where nodes are products and edges exist between products that are co-purchased by users frequently. Node features are bag-of-words representations of product reviews. Node labels represent the category of products.

To prepare graphs with missing features, we pre-processed the datasets and removed a portion of node features according to a missing rate parameter mr. We consider the following three cases.

- *Uniform randomly missing features*
  $mr = |\mathcal{S}|/(ND)$ (percentage) of the features are randomly selected and removed from the node feature matrix $\mathbf{X}$. $\mathcal{S}$ was randomly selected with uniform probability.
- *Biased randomly missing features*
  90% of certain features and 10% of the remaining features are randomly selected and removed from $\mathbf{X}$. In this scenario, the features with 90% values removed represent sensitive information, which is always missing in practice. For ease of implementation, such sensitive features are randomly selected under the condition $mr = |\mathcal{S}|/(ND)$.
- *Structurally missing features*
  The respective features of mr (percentage) random nodes are removed from $\mathbf{X}$. Specifically, $\mathcal{V}' \subseteq \mathcal{V}$ was randomly

Table 4
The accuracy results for node classification task in AmaPhoto.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	MEAN	92.15	92.05	91.81	91.62	91.40	90.76	88.98	86.41	68.88
	K-NN	92.27	92.12	91.94	91.67	91.37	90.92	90.03	87.41	81.91
	MFT	92.23	92.07	91.88	91.51	91.15	90.11	88.28	85.17	75.73
	SoftImp	92.23	92.09	91.92	91.78	91.55	91.18	90.55	88.93	85.22
	MICE	92.23	92.07	91.97	91.75	91.52	91.22	90.42	86.43	82.88
	MissForest	92.18	92.09	91.82	91.61	91.42	90.71	89.17	86.03	82.82
	VAE	92.20	92.08	91.90	91.59	91.15	90.55	89.28	86.95	81.43
	GAIN	92.23	92.11	91.90	91.73	91.49	91.24	90.72	89.49	86.96
	GINN	92.25	92.03	91.87	91.53	91.14	90.56	88.59	85.02	79.80
	GCNmf	**92.54**	**92.44**	**92.20**	**92.09**	**92.09**	**91.69**	**91.25**	**90.57**	**88.96**
	Performance gain (%)	0.29–0.42	0.35–0.45	0.25–0.42	0.34–0.63	0.59–1.04	0.49–1.75	0.58–3.36	1.21–6.53	2.30–29.15
Biased randomly missing	Mean	92.19	91.89	91.80	91.58	91.24	90.74	89.69	87.23	76.91
	K-NN	92.24	92.09	91.99	91.85	91.58	91.32	90.68	89.39	81.88
	MFT	92.17	92.03	91.98	91.71	91.40	90.99	89.89	87.46	75.14
	SoftImp	92.21	92.10	92.02	91.85	91.61	91.27	90.52	88.87	84.84
	MICE	92.16	92.06	92.00	91.76	91.58	91.24	90.54	88.64	82.45
	MissForest	92.16	92.09	92.07	91.81	91.35	90.67	89.77	86.85	82.72
	VAE	92.14	92.04	91.95	91.70	91.41	91.02	90.00	88.92	83.08
	GAIN	92.22	92.02	91.87	91.76	91.58	91.43	90.88	89.99	87.11
	GINN	92.24	92.04	91.95	91.78	91.48	91.16	90.40	88.35	79.18
	GCNmf	**92.72**	**92.69**	**92.55**	**92.61**	**92.43**	**92.33**	**91.91**	**91.58**	**89.35**
	Performance gain (%)	0.52–0.63	0.64–0.87	0.52–0.82	0.83–1.12	0.90–1.30	0.98–1.83	1.13–2.48	1.77–5.45	2.57–18.91
Structurally missing	MEAN	92.06	91.80	91.59	91.20	90.59	89.83	87.66	84.60	77.41
	K-NN	92.04	91.71	91.43	91.08	90.37	89.88	88.80	85.77	80.48
	MFT	92.08	91.83	91.59	91.18	90.56	89.80	87.58	84.36	77.69
	SoftImp	91.75	91.19	90.55	89.33	88.00	87.19	84.87	81.96	76.72
	MICE	92.05	91.87	91.59	91.24	90.60	89.86	87.82	84.57	77.32
	MissForest	92.04	91.70	91.42	91.15	90.49	90.07	88.81	85.51	75.35
	VAE	92.11	91.84	91.50	91.08	90.46	89.29	87.47	83.45	67.85
	GAIN	92.04	91.78	91.49	91.14	90.63	89.94	88.60	85.41	76.48
	GINN	92.09	91.83	91.53	91.16	90.43	89.61	87.77	84.53	77.14
	GCNmf	**92.45**	**92.32**	**92.08**	**91.88**	**91.52**	**90.89**	**90.39**	**89.64**	**86.09**
	Performance gain (%)	0.37–0.76	0.49–1.24	0.53–1.69	0.70–2.85	0.98–4.00	0.91–4.24	1.78–6.50	4.51–9.37	6.97–26.88
RGCN		91.50	90.81	88.37	85.52	75.17	84.89	87.67	89.95	90.56
GCN		92.35								
GCN w/o node features		88.77								

selected with uniform probability, such that $mr = |\mathcal{V}'|/N$. Then, $\mathcal{S} = \{(i, j)|v_i \in \mathcal{V}', j = 1, \ldots, D\}$.

Baselines. We consider the following imputation methods to fill in missing values and then apply GCN on the complete graphs.

- MEAN [30]: This method replaces missing values with the mean of observed features based on the respective row of the feature matrix $\mathbf{X}$.
- K-NN [33]: This approach samples similar features by k-nearest neighbors and then replaces missing values with the mean of these features. We set $k = 5$.
- MFT [53]: This is the imputation method based on factorizing the incomplete matrix into two low-rank matrices.
- SoftImp [32]: This method iteratively replaces the missing values with those estimated from a soft-thresholded singular value decomposition (SVD).
- MICE [55]: This is the multiple imputation method that infers missing values from the conditional distributions by Markov chain Monte Carlo (MCMC) techniques.
- MissForest [34]: This is a non-parametric imputation method that utilizes Random Forest to predict missing values.
- VAE [35]: This is a VAE based method for reconstructing missing values.
- GAIN [37]: This is a GAN-based approach for imputing missing data.
- GINN [36]: This is a imputation method based on graph denoising autoencoder.

We employed Optuna [65] to tune the hyper-parameters such as learning rate, L_2 regularization, and dropout rate. We followed the normalized initialization scheme [66] to initialize the weight matrix. We adopted Adam algorithm [67] for optimization. For GCNmf, we simply set the number of Gaussian components to 5 across all datasets. The implementation of all approaches is in Python and PyTorch and we ran the experiments on a single machine with Intel Xeon Gold 6148 Processor @2.40 GHz, NVIDIA Tesla V100 GPU, and RAM @64 GB. For reproducibility, the source code of GCNmf and the graph datasets are publicly available.[2]

5.1. Node classification

We conducted experiments for the node classification task. We followed the data splits of previous work [68] on Cora and Citeseer. As for AmaPhoto and AmaComp, we randomly chose 40 nodes per class for training, 500 nodes for validation, and the remaining for testing. We gradually increased the missing

[2] https://github.com/marblet/GCNmf.

Table 5
The accuracy results for node classification task in AmaComp.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	MEAN	82.79	82.36	81.51	80.53	79.30	77.22	74.56	61.60	5.92
	K-NN	82.89	82.73	82.18	82.00	81.54	80.58	79.34	76.81	66.04
	MFT	82.82	82.54	82.05	81.58	80.76	79.28	77.11	72.31	49.42
	SoftImp	82.99	82.75	82.37	82.06	81.48	80.48	79.27	77.29	69.04
	MICE	82.83	82.76	82.43	82.28	81.66	80.59	78.63	75.00	63.60
	MissForest	–	–	–	–	80.89	79.57	78.22	76.00	71.98
	VAE	82.65	82.47	81.72	81.15	80.47	79.99	78.55	75.80	67.26
	GAIN	82.94	82.78	82.44	81.96	81.56	80.71	79.96	78.38	76.15
	GINN	82.94	82.78	82.27	81.65	80.89	78.53	76.46	73.24	58.34
	GCNmf	**86.32**	**86.07**	**85.98**	**85.77**	**85.46**	**84.94**	**84.03**	**82.38**	**77.52**
	Performance gain (%)	4.01 – 4.44	3.97 – 4.50	4.29 – 5.48	4.24 – 6.51	4.65 – 7.77	5.24 – 10.00	5.09 – 12.70	5.10 – 33.73	1.80 – 1209.46
Biased randomly missing	Mean	83.03	83.07	82.49	81.82	81.17	79.76	78.16	73.79	8.68
	K-NN	83.01	82.79	82.43	82.14	81.57	81.40	80.24	77.86	66.45
	MFT	82.98	82.86	82.39	81.93	81.30	80.18	78.66	74.96	50.53
	SoftImp	83.07	82.88	82.13	81.87	81.23	80.53	78.98	76.74	73.91
	MICE	83.07	82.77	82.44	81.94	81.56	80.84	79.40	76.71	64.11
	MissForest	–	–	81.88	–	80.52	79.62	78.27	76.66	71.74
	VAE	82.93	82.66	82.27	81.57	81.04	80.28	78.50	76.43	72.58
	GAIN	83.04	82.90	82.70	82.15	81.69	81.35	80.45	78.88	76.47
	GINN	83.10	82.71	82.58	81.94	81.63	80.81	79.29	76.53	58.18
	GCNmf	**86.41**	**86.35**	**86.27**	**86.16**	**85.83**	**85.37**	**84.84**	**83.00**	**79.58**
	Performance gain (%)	3.98 – 4.20	3.95 – 4.46	4.32 – 5.36	4.88 – 5.63	5.07 – 6.59	4.88 – 7.22	5.46 – 8.55	5.22 – 12.48	4.07 – 816.82
Structurally missing	MEAN	82.53	82.09	81.35	80.62	79.59	77.75	75.06	69.67	23.42
	K-NN	82.59	82.15	81.57	81.07	80.25	78.86	76.91	72.89	42.23
	MFT	82.48	81.91	81.43	80.58	79.40	77.64	75.19	69.97	27.33
	SoftImp	82.64	81.97	81.32	80.83	79.68	77.66	75.92	56.62	52.75
	MICE	82.71	82.13	81.51	80.62	79.36	77.35	74.57	67.59	45.07
	MissForest	82.65	82.20	81.84	81.04	79.18	78.66	75.98	71.91	12.05
	VAE	82.76	82.40	81.72	80.88	79.23	77.62	73.76	66.33	41.37
	GAIN	82.76	82.53	82.11	81.68	80.76	78.65	74.38	67.38	54.24
	GINN	82.55	82.10	81.46	80.75	79.59	77.67	75.08	70.40	26.10
	GCNmf	**86.37**	**86.22**	**85.80**	**85.43**	**85.24**	**84.73**	**84.06**	**80.63**	**73.42**
	Performance gain (%)	4.36 – 4.72	4.47 – 5.26	4.49 – 5.51	4.59 – 6.02	5.55 – 7.65	7.44 – 9.54	9.30 – 13.96	10.62 – 42.41	35.36 – 509.29
RGCN		79.18	76.39	74.01	63.19	14.24	63.24	72.44	75.33	77.18
GCN		82.94								
GCN w/o node features		81.60								

rate *mr* from 10% to 90%. With each missing rate, we generated five instances of missing data and evaluated the performance twenty times for each instance. To ensure a fair comparison, we employed the following parameter settings of GCN model for all approaches: we set the number of layers to 2, the number of hidden units to 16 (Cora and Citeseer) and 64 (AmaPhoto and AmaComp). Moreover, we adopted an early stopping strategy with a patience of 100 epochs to avoid over-fitting [41].

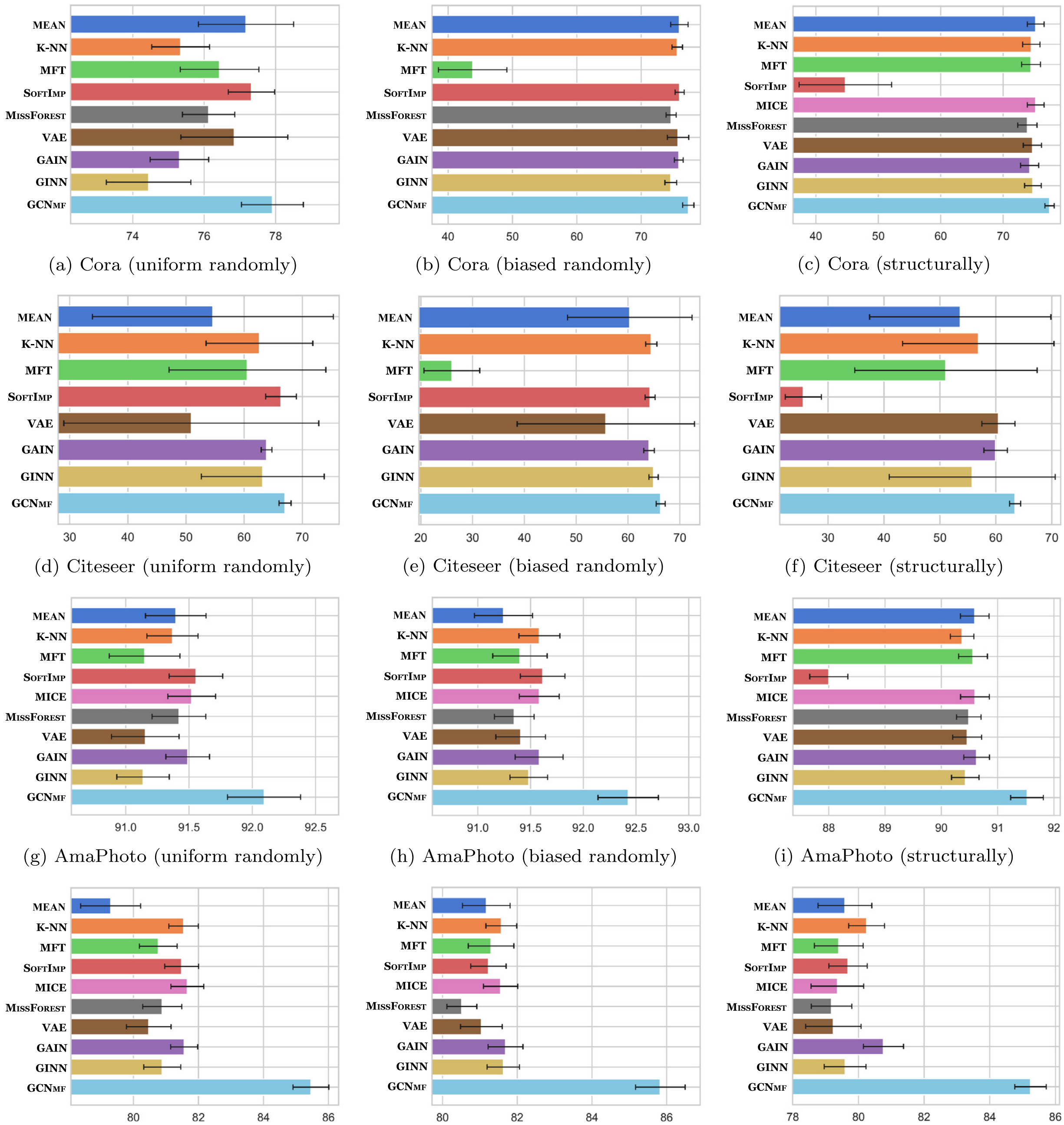
Tables 2–5 lists the accuracy obtained by different methods. Bold and underline indicate the best and the second best score for each setting. Moreover, we provide the performance results of another three methods as a reference: (1) GCN in the setting of complete features ($\mathcal{S} = \emptyset$); (2) GCN without using node features (using the identity matrix instead of the node feature matrix $\mathbf{X}$); (3) RGCN [51] in the setting that node features are under adversarial attacks (we deliberately perturbated the features that map to the same set of the uniform randomly missing features, and modified the node feature matrix $\mathbf{X}^*$; then we feed $\mathbf{X}^*$ to RGCN).

Note that some results of MICE (in Cora and Citeseer) and MissForest (in Citeseer and AmaComp) are not available because we encountered unexpected runtime errors or the program takes more than 24 h to terminate. We have the following observations.

First, GCNmf demonstrates the best performance and there is no method that clearly wins the second place. GCNmf achieves the highest accuracy for almost all of the missing rates and across all datasets, with only four exceptions. For the uniform randomly missing case, GCNmf is markedly superior to the others. It achieves improvement of up to 8.69%, 11.82%, 2.30%, and 5.24% when compared with the best accuracy scores among baselines in the four datasets, respectively. For the biased randomly missing case, the improvement is up to 10.64%, 9.39%, 2.57%, and 5.46%, respectively. For the structurally missing case, this advantage becomes even greater, with the corresponding maximum improvement raising to 99.43%, 102.96%, 6.97%, and 35.36%, respectively. Most strikingly, when the missing rate reaches 80%, i.e., the features of 80% nodes are not known, GCNmf can still achieve an accuracy of 68.00% in Cora, while all baselines fail.

Secondly, GCNmf is more appealing when a large portion of features are missing. This can be explained by the fact that the performance gain, on the whole, becomes larger and larger as the missing rate increases. In contrast, the imputation based method becomes less reliable at high missing rates. For example, the accuracy of baselines (except for SoftImp) falls to below 20.0% when the missing rate reaches 90% for the structurally missing case in Cora.

Thirdly, it is interesting to note that GCNmf even outperforms GCN when only a small number of features are missing. For example, GCNmf holds a slim advantage over GCN when the missing rate is 10% in the four datasets. This indicates that



(a) Cora (uniform randomly) (b) Cora (biased randomly) (c) Cora (structurally)

(d) Citeseer (uniform randomly) (e) Citeseer (biased randomly) (f) Citeseer (structurally)

(g) AmaPhoto (uniform randomly) (h) AmaPhoto (biased randomly) (i) AmaPhoto (structurally)

(j) AmaComp (uniform randomly) (k) AmaComp (biased randomly) (l) AmaComp (structurally)

Fig. 2. Performance variance for node classification task ($mr = 50\%$).

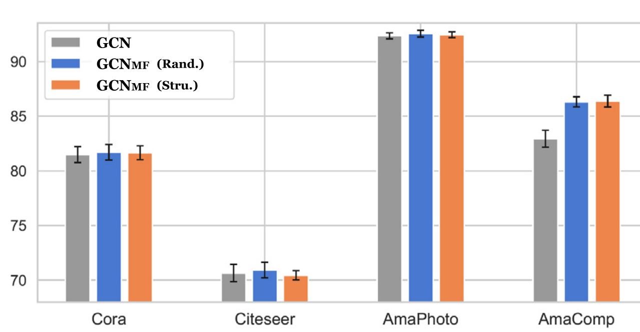


Fig. 3. Performance variance of GCNMF ($mr = 10\%$) and GCN ($mr = 0\%$) for node classification task.

Table 6
The AUC results for link prediction task in Cora.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	Mean	90.72	90.41	90.10	89.79	89.11	88.40	87.13	84.47	74.97
	K-NN	92.20	91.86	91.34	90.93	90.19	89.03	87.62	85.69	81.55
	MFT	92.16	91.86	91.37	90.91	90.14	88.37	86.11	84.10	79.94
	SoftImp	90.88	90.79	90.64	90.40	89.98	89.22	88.37	86.75	<u>84.13</u>
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	<u>92.32</u>	<u>92.04</u>	91.61	90.95	90.33	89.34	88.33	86.41	82.78
	VAE	92.23	91.91	91.33	90.54	89.28	86.98	82.52	77.74	77.27
	GAIN	92.17	91.87	91.46	<u>91.00</u>	<u>90.57</u>	<u>89.78</u>	<u>89.17</u>	<u>88.13</u>	**86.01**
	GINN	92.15	91.96	<u>91.62</u>	<u>91.00</u>	90.28	88.94	87.66	84.73	74.90
	GCNmf	**94.09**	**93.50**	**93.05**	**92.40**	**92.29**	**91.79**	**90.77**	**88.32**	81.46
	Performance gain (%)	1.92 | 3.71	1.59 | 3.42	1.56 | 3.27	1.54 | 2.91	1.90 | 3.57	2.24 | 5.53	1.79 | 10.00	0.22 | 13.61	−5.29 | 8.76
Biased randomly missing	Mean	92.18	92.08	92.14	91.89	91.43	91.01	89.55	87.19	76.96
	K-NN	92.17	92.06	92.02	91.83	91.47	90.92	89.84	87.85	81.65
	MFT	92.17	91.44	90.65	90.00	89.50	88.91	87.48	85.36	80.20
	SoftImp	<u>92.35</u>	<u>92.34</u>	<u>92.35</u>	<u>92.08</u>	<u>91.74</u>	<u>91.36</u>	**90.03**	<u>88.44</u>	<u>86.17</u>
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	92.27	92.22	92.22	91.80	91.22	90.34	88.90	86.86	83.58
	VAE	92.19	92.05	91.83	91.37	90.75	89.71	87.37	84.95	76.71
	GAIN	92.18	92.01	91.88	91.70	91.28	90.75	<u>89.87</u>	**88.75**	**86.69**
	GINN	92.15	92.11	92.04	91.88	91.52	90.89	89.45	87.36	75.38
	GCNmf	**94.35**	**94.20**	**93.90**	**93.15**	**92.43**	**91.46**	**90.03**	86.10	81.72
	Performance gain (%)	2.17 | 2.39	2.01 | 3.02	1.68 | 3.59	1.16 | 3.50	0.75 | 3.27	0.11 | 2.87	0.18 | 3.04	−2.99 | 1.35	−5.73 | 8.41
Structurally missing	Mean	90.34	89.79	89.12	88.26	87.12	85.33	83.23	79.61	71.79
	K-NN	<u>91.60</u>	91.08	<u>90.38</u>	<u>89.36</u>	<u>88.34</u>	**87.16**	**85.40**	**82.09**	76.12
	MFT	91.51	91.00	89.95	89.11	87.36	85.81	82.90	77.73	73.72
	SoftImp	90.29	89.67	88.86	87.86	86.77	85.36	83.07	<u>81.53</u>	**77.38**
	MICE	91.58	<u>91.11</u>	90.30	89.34	88.18	<u>86.70</u>	<u>84.24</u>	80.31	72.63
	MissForest	91.57	91.05	90.23	<u>89.36</u>	<u>88.34</u>	**87.16**	**85.40**	**82.09**	<u>76.22</u>
	VAE	91.49	90.76	89.49	87.27	83.81	80.07	73.46	67.55	65.80
	GAIN	<u>91.60</u>	91.08	<u>90.38</u>	<u>89.36</u>	<u>88.34</u>	**87.16**	**85.40**	**82.09**	76.12
	GINN	91.51	90.85	89.68	87.34	83.23	76.22	66.55	63.88	64.91
	GCNmf	**93.55**	**92.65**	**91.68**	**90.55**	**88.54**	86.19	81.96	76.35	67.86
	Performance gain (%)	2.13 | 3.61	1.69 | 3.32	1.44 | 3.17	1.33 | 3.76	0.23 | 6.38	−1.11 | 13.08	−4.03 | 23.16	−6.99 | 19.52	−12.30 | 4.54
GCN		92.42								
GCN w/o node features		85.90								

GCNmf is robust against low-level missing features. Moreover, GCNmf achieves much higher accuracy than RGCN. This is easy to understand because the task is more challenging for RGCN than GCNmf.

Figs. 2–3 show the variability of the performance for different methods. We can see that GCNmf is more robust than the baselines, especially in Cora and Citeseer, where there is a high level of variability. Moreover, GCNmf and GCN are on the same level of variability. This implies that representing incomplete features by GMM and calculating the expected activation of neurons do not undermine the robustness of GCN.

5.2. Link prediction

The second experiment is for the link prediction task in the Cora and Citeseer citation graphs. We took VGAE [24] as the base model, which is a variational graph autoencoder and employs GCN as an encoder. We gradually increased the missing rate *mr* from 10% to 90% and compare GCNmf against baselines within the base model framework. Following the previous work [24], we randomly chose 10% edges for testing, 5% edges for validation, and the remaining edges for training; we used a 32-dim hidden layer and 16-dim latent variables in the base model.

Tables 6 and 7 show the average AUC scores obtained by different methods. Bold and underline indicate the best and the second best score for each setting. We also provide the performance results of (1) GCN in the setting of complete features ($\mathcal{S} = \emptyset$), and (2) GCN without node features (using the identity matrix instead of the node feature matrix $\mathbf{X}$) as a reference.

We can reach a similar conclusion as the node classification task. GCNmf exhibits the best overall performance. In particular, GCNmf demonstrates excellent performance and is overwhelmingly superior to all baselines in Citeseer; GCNmf outperforms the baselines in most cases in Cora, with only several exceptions when the missing rate reaches high. Again, we can observe the robustness merit of GCNmf, as it even outperforms GCN when the missing rate is low.

We attribute the superiority of GCNmf to the joint learning of GMM and network parameters. Actually, our approach can be understood as calculating the expected activation of neurons over the imputations drawn from missing data density in the first layer. It is the end-to-end joint learning of the parameters that make our approach less likely to converge to sub-optimal solutions.

5.3. Running time comparison

We compare the running time of different approaches in Table 8. The numbers represent the sum of time for parameter initialization, missing value imputation, and model training. We

Table 7
The AUC results for link prediction task in Citeseer.

Missing type	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing	Mean	89.01	88.56	88.01	87.33	86.42	85.30	83.77	81.43	75.47
	K-NN	90.00	89.60	89.10	88.34	87.32	85.68	83.39	81.16	78.60
	MFT	89.86	89.43	88.81	87.72	85.76	83.24	81.20	79.97	77.94
	SoftImp	90.19	90.15	89.81	89.55	88.97	88.17	86.80	84.99	81.66
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	89.85	89.09	88.13	87.22	85.36	83.55	80.64	74.89	64.69
	GAIN	89.96	89.53	89.07	88.36	87.51	86.52	85.35	83.93	81.70
	GINN	90.02	89.64	89.04	87.91	86.56	84.64	83.32	81.82	77.19
	GCNmf	**93.20**	**92.96**	**92.30**	**92.19**	**90.45**	**90.08**	**88.91**	**87.28**	**83.68**
	Performance gain (%)	3.34–4.71	3.12–4.97	2.77–4.87	2.95–5.70	1.66–5.96	2.17–8.22	2.43–10.26	2.69–16.54	2.42–29.36
Biased randomly missing	Mean	89.94	89.88	89.63	89.33	89.25	88.55	87.57	85.28	78.23
	K-NN	90.00	89.98	89.81	89.54	89.31	88.52	87.47	84.97	78.85
	MFT	89.98	87.50	85.88	85.07	84.32	83.76	82.85	81.54	78.23
	SoftImp	90.31	90.25	90.23	89.99	89.90	89.03	87.12	85.96	80.63
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	89.90	89.25	88.33	87.32	86.26	83.78	83.05	80.71	62.51
	GAIN	89.97	89.87	89.60	89.32	88.89	87.85	87.00	85.05	**81.95**
	GINN	90.27	89.99	89.85	89.47	89.10	88.15	87.21	84.47	76.83
	GCNmf	**93.53**	**93.38**	**92.81**	**92.48**	**91.68**	**91.25**	**89.54**	**86.73**	81.43
	Performance gain (%)	3.57–4.04	3.47–6.72	2.86–8.07	2.77–8.71	1.98–8.73	2.49–8.94	2.25–8.07	0.90–7.46	−0.63–30.27
Structurally missing	Mean	88.16	86.95	85.76	84.20	82.43	80.83	78.92	75.79	69.76
	K-NN	89.50	88.36	87.01	85.52	83.85	82.11	79.81	76.49	70.86
	MFT	89.24	87.96	86.53	84.76	83.19	80.67	78.35	75.97	72.64
	SoftImp	89.50	88.36	87.01	85.52	83.85	82.11	79.81	76.49	70.86
	MICE	–	–	–	–	–	–	–	–	–
	MissForest	–	–	–	–	–	–	–	–	–
	VAE	88.57	86.83	84.32	80.96	77.49	74.01	67.84	63.06	60.39
	GAIN	89.50	88.36	87.01	85.52	83.85	82.11	79.81	76.49	70.86
	GINN	87.48	83.35	77.50	70.06	64.31	59.45	57.95	54.88	50.81
	GCNmf	**92.23**	**90.54**	**88.77**	**85.74**	**84.78**	**84.59**	**82.00**	**77.21**	**73.31**
	Performance gain (%)	3.05–5.43	2.47–8.63	2.02–14.54	0.26–22.38	1.11–31.83	3.02–42.29	2.74–41.50	0.94–40.69	0.92–44.28
GCN		90.25								
GCN w/o node features		79.94								

Table 8
The running time (seconds) of different approaches for the uniform randomly missing case ($mr = 50\%$). The figure in the parentheses indicates the time for initialization of GMM parameters.

	Cora	Citeseer	AmaPhoto	AmaComp
MEAN	1.10	1.24	12.09	14.14
K-NN	125.04	480.19	482.73	1505.14
MFT	141.14	567.50	428.95	906.52
SoftImp	115.15	850.55	59.26	95.14
MICE	–	–	3879.59	6705.73
MissForest	4039.10	–	32528.25	48264.58
VAE	7.91	8.64	14.23	18.78
GAIN	79.35	426.10	36.06	35.19
GINN	300.64	839.96	998.03	3199.96
GCNmf	7.43 (0.59)	13.52 (2.60)	22.64 (4.11)	42.38 (9.75)
GCN	0.86	0.91	6.82	7.79

also provide a reference time of GCN when $\mathcal{S} = \emptyset$. We can observe that GCNmf algorithm runs in reasonable time, with model training taking the majority of time (the time for initialization of GMM parameters only accounts for less than 25%). In comparison, GCNmf is slower than MEAN and VAE, but is much faster than the other seven methods. We note that some imputation techniques suffer due to the high dimension of features. For example, MissForest did not finish within 24 h in Citeseer.

5.4. Analysis of GCNmf

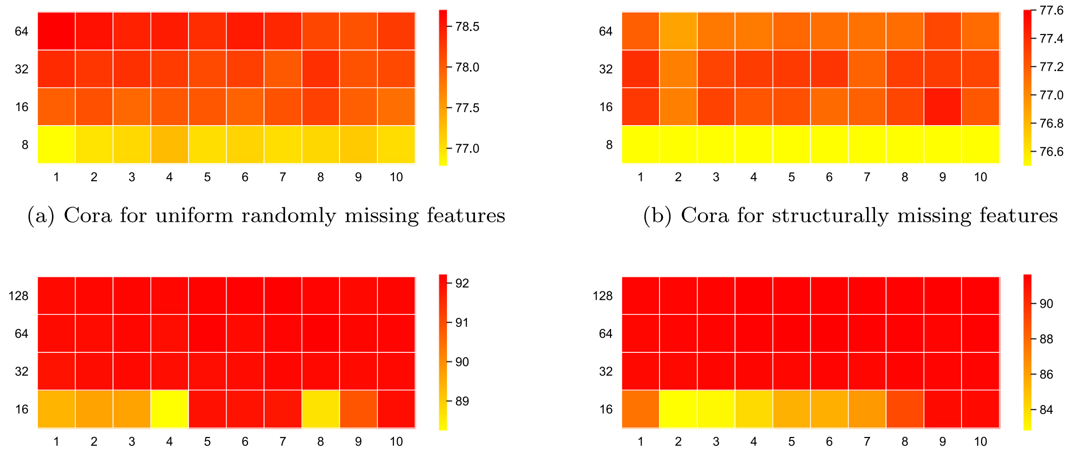
In this section, we provide study of GCNmf in terms of hyper-parameter sensitivity, optimization analysis, and quality of the reconstructed features.

5.4.1. Hyper-parameter analysis

Fig. 4 depicts the performance results with different assignments on the Gaussian components K and the number of hidden units $D^{(1)}$ in Cora and AmaPhoto datasets. We can observe that the performance reaches a plateau when we have enough number of hidden units to transcribe the information, i.e., $D^{(1)} \geq 16$ for Cora and $D^{(1)} \geq 32$ for AmaPhoto. On the other hand, the performance is not sensitive to K, with differences between the best and worst less than 0.82% when $D^{(1)} \geq 16$ in Cora and 0.30% when $D^{(1)} \geq 32$ in AmaPhoto, respectively.

5.4.2. Analysis of optimization

GCNmf employs a joint optimization of GMM and GCN within the same network architecture. Alternatively, we can consider a two-step optimization strategy: in the first step we optimize GMM parameters with input node features using EM algorithm; in the second step we optimize GCN parameters by gradient descent algorithm while fixing the GMM parameters.



(a) Cora for uniform randomly missing features

(b) Cora for structurally missing features

(c) AmaPhoto for uniform randomly missing features

(d) AmaPhoto for structurally missing features

Fig. 4. Node classification results for GCNmf with different assignments on the Gaussian components K and the number of hidden units $D^{(1)}$ ($mr = 50\%$). The x-axis represents K. The y-axis represents $D^{(1)}$.

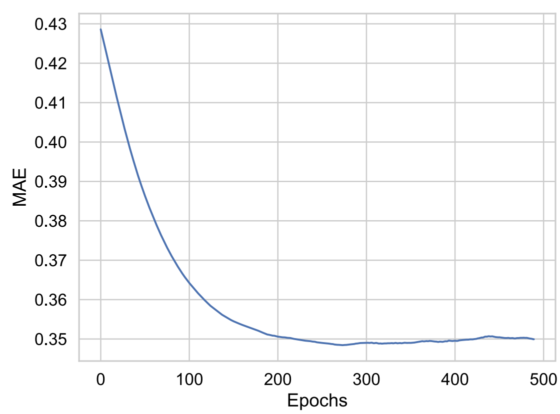


Fig. 5. MAE of the reconstructed features during the training process for node classification task (structurally missing features, mr = 20%) in AmaPhoto.

Table 9
The accuracy results for joint optimization and two-step optimization in node classification task.

Dataset	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform randomly missing features										
Cora	Joint Opt.	**81.70**	**81.66**	**80.41**	**79.52**	**77.91**	**76.67**	**74.38**	**70.57**	**63.49**
	Two-step Opt.	81.50	81.43	79.81	79.35	76.75	76.04	73.97	69.16	61.46
Citeseer	Joint Opt.	**70.93**	**70.82**	**69.84**	68.83	**67.03**	**64.78**	**60.70**	**55.38**	**47.78**
	Two-step Opt.	70.54	70.73	69.66	**69.20**	66.59	64.52	60.07	53.68	46.53
Structurally missing features										
Dataset	Missing rate	10%	20%	30%	40%	50%	60%	70%	80%	90%
AmaPhoto	Joint Opt.	92.45	**92.32**	**92.08**	**91.88**	**91.52**	**90.89**	**90.39**	**89.64**	**86.09**
	Two-step Opt.	**92.49**	92.23	91.90	91.40	90.73	87.94	84.93	62.46	32.44
AmaComp	Joint Opt.	**86.37**	**86.22**	**85.80**	**85.43**	**85.24**	**84.73**	**84.06**	**80.63**	**73.42**
	Two-step Opt.	86.30	85.99	85.49	84.69	83.79	82.76	80.23	71.80	39.98

We compare the two optimization strategies in Table 9. We can observe that the joint optimization clearly beats the two-step optimization. The advantage becomes greater and greater as the missing rate increases. In particular, when the missing rate becomes high, the two-step optimization fails to learn the "right" model parameters and the performance deteriorates sharply.

5.4.3. Analysis of reconstructed node features

Finally, we conducted a study on how well the reconstructed features by GCNmf. The reconstructed features are the mean of GMM, namely the weighted average of the mean vectors. Fig. 5 depicts the Mean Absolute Error (MAE) of the reconstructed features and true features during the training process of the node classification task in AmaPhoto. We can observe that MAE decreases as the number of training epochs increases, and it converges to around 0.35 after 200 epochs. This suggests that the trained GMM captures the density of features more accurately than the initial state optimized by EM algorithm. Although the training aims at learning node labels, it helps to reconstruct the missing features.

6. Conclusion

We proposed GCNmf to supplement a severe deficiency of current GCN models—inability to handle graphs containing missing features. In contrast to the traditional strategy of imputing missing features before applying GCN, GCNmf integrates the processing of missing features and graph learning within the same neural network architecture. Specifically, we propose a novel way to unify the representation of missing features and calculation of the expected activation of the first layer neurons in GCN. We empirically demonstrate that (1) GCNmf is robust against low level of missing features, (2) GCNmf significantly outperforms the imputation based methods in the node classification and link prediction tasks.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partly supported by JSPS Grant-in-Aid for Early-Career Scientists, Japan (Grant Number 19K20352), JSPS Grant-in-Aid for Scientific Research (B), Japan (Grant Number 17H01785), JST CREST, Japan (Grant Number JPMJCR1687), and the New Energy and Industrial Technology Development Organization (NEDO), Japan.

References

[1] P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding, IEEE Trans. Knowl. Data Eng. 31 (5) (2018) 833–852.
[2] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE Trans. Knowl. Data Eng. 29 (12) (2017) 2724–2743.
[3] S. Ji, S. Pan, E. Cambria, P. Marttinen, P.S. Yu, A survey on knowledge graphs: Representation, acquisition and applications, 2020, arXiv:2002.00388.
[4] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of KDD, 2014, pp. 701–710.
[5] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, J. Tang, Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec, in: Proceedings of WSDM, 2018, pp. 459–467.
[6] X. Liu, T. Murata, K.-S. Kim, C. Kotarasu, C. Zhuang, A general view for network embedding as matrix factorization, in: Proceedings of WSDM, 2019, pp. 375–383.
[7] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of WWW, 2015, pp. 1067–1077.
[8] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of KDD, 2016, pp. 1225–1234.
[9] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, C. Zhang, Adversarially regularized graph autoencoder for graph embedding, in: Proceedings of IJCAI, 2018, pp. 2609–2615.
[10] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.
[11] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, 2018, arXiv:1812.08434.
[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020) 1–21.
[13] M. Narasimhan, S. Lazebnik, A. Schwing, Out of the box: Reasoning with graph convolution nets for factual visual question answering, in: Proceedings of NeurIPS, 2018, pp. 2654–2665.
[14] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, in: Proceedings of CVPR, 2017, pp. 3693–3702.
[15] Z. Liu, Y. Dou, P.S. Yu, Y. Deng, H. Peng, Alleviating the inconsistency problem of applying graph neural network to fraud detection, in: Proceedings of SIGIR, 2020.
[16] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, K. Sima'an, Graph convolutional encoders for syntax-aware neural machine translation, in: Proceedings of EMNLP, 2017, pp. 1957–1967.
[17] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Proceedings of NeurIPS, 2015, pp. 2224–2232.
[18] A. Fout, J. Byrd, B. Shariat, A. Ben-Hur, Protein interface prediction using graph convolutional networks, in: Proceedings of NeurIPS, 2017, pp. 6530–6539.
[19] L. Yang, F. Wu, J. Gu, C. Wang, X. Cao, D. Jin, Y. Guo, Graph attention topic modeling network, in: Proceedings of the Web Conference, 2020, pp. 144–154.
[20] R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: Proceedings of KDD, 2018, pp. 974–983.
[21] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proceedings of ICLR, 2017.
[22] D.I. Shuman, S.K. Narang, P. Frossard, A. Ortega, P. Vandergheynst, The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains, IEEE Signal Process. Mag. 30 (3) (2013) 83–98.
[23] M. Zhang, Z. Cui, M. Neumann, Y. Chen, An end-to-end deep learning architecture for graph classification, in: Proceedings of AAAI, 2018.
[24] T.N. Kipf, M. Welling, Variational graph auto-encoders, in: NIPS Workshop on Bayesian Deep Learning, 2016.
[25] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, W. Wang, Simgnn: A neural network approach to fast graph similarity computation, in: Proceedings of WSDM, 2019, pp. 384–392.
[26] S.K. Maurya, X. Liu, T. Murata, Fast approximations of betweenness centrality with graph neural networks, in: Proceedings of CIKM, 2019, pp. 2149–2152.
[27] H. Chen, H. Yin, T. Chen, Q.V.H. Nguyen, W.-C. Peng, X. Li, Exploiting centrality information with graph convolutions for network representation learning, in: Proceedings of ICDE, 2019, pp. 590–601.
[28] D. Jin, Z. Liu, W. Li, D. He, W. Zhang, Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks, in: Proceedings of AAAI, 2019, pp. 152–159.
[29] J.J. Choong, X. Liu, T. Murata, Learning community structure with variational autoencoder, in: Proceedings of ICDM, 2018, pp. 69–78.
[30] P.J. García-Laencina, J.-L. Sancho-Gómez, A.R. Figueiras-Vidal, Pattern classification with missing data: a review, Neural Comput. Appl. 19 (2) (2010) 263–282.
[31] J. Yi, J. Lee, S. Hwang, E. Yang, Why not to use zero imputation? correcting sparsity bias in training neural networks, in: Proceedings of ICLR, 2020.
[32] R. Mazumder, T. Hastie, R. Tibshirani, Spectral regularization algorithms for learning large incomplete matrices, J. Mach. Learn. Res. 11 (2010) 2287–2322.
[33] G.E. Batista, M.C. Monard, et al., A study of k-nearest neighbour as an imputation method, HIS 87 (251–260) (2002) 48.
[34] D.J. Stekhoven, P. Bühlmann, MissForest—non-parametric missing value imputation for mixed-type data, Bioinformatics 28 (1) (2011) 112–118.
[35] D.P. Kingma, M. Welling, Auto-encoding variational bayes, in: Proceedings of ICLR, 2014, pp. 1–14.
[36] I. Spinelli, S. Scardapane, U. Aurelio, Missing data imputation with adversarially-trained graph convolutional networks, Neural Netw. 129 (2020) 249–260.
[37] J. Yoon, J. Jordon, M. van der Schaar, GAIN: Missing data imputation using generative adversarial nets, in: Proceedings of ICML, Vol. 80, 2018, pp. 5689–5698.
[38] Y. Luo, X. Cai, Y. ZHANG, J. Xu, Y. xiaojie, Multivariate time series imputation with generative adversarial networks, in: Proceedings of NeurIPS, 2018, pp. 1596–1607.
[39] S.C.-X. Li, B. Jiang, B. Marlin, Learning from incomplete data with generative adversarial networks, in: Proceedings of ICLR, 2019.
[40] M. Śmieja, Ł. Struski, J. Tabor, B. Zieliński, P. Spurek, Processing of missing data by neural networks, in: Proceedings of NeurIPS, 2018, pp. 2719–2729.
[41] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, in: Proceedings of ICLR, 2018.
[42] R. Li, S. Wang, F. Zhu, J. Huang, Adaptive graph convolutional neural networks, in: Proceedings of AAAI, 2018.
[43] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, Y. Guo, Topology optimization based graph convolutional network, in: Proceedings of IJCAI, 2019, pp. 4054–4061.
[44] R. Hu, S. Pan, G. Long, Q. Lu, L. Zhu, J. Jiang, Going deep: Graph convolutional ladder-shape networks, in: Proceedings of AAAI, 2020.
[45] S. Abu-El-Haija, B. Perozzi, A. Kapoor, H. Harutyunyan, N. Alipourfard, K. Lerman, G.V. Steeg, A. Galstyan, Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing, in: Proceedings of ICML, 2019.
[46] M. Shi, Y. Tang, X. Zhu, J. Liu, Feature-attention graph convolutional networks for noise resilient learning, 2019, arXiv:1912.11755.

[47] J. Chen, T. Ma, C. Xiao, FastGCN: Fast learning with graph convolutional networks via importance sampling, in: Proceedings of ICLR, 2018.
[48] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. Hsieh, Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks, in: Proceedings of KDD, 2019, pp. 257–266.
[49] D. Zügner, A. Akbarnejad, S. Günnemann, Adversarial attacks on neural networks for graph data, SIGKDD, 2018, pp. 2847–2856.
[50] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, L. Song, Adversarial attack on graph structured data, Proceedings of ICML, 2018, pp. 1115–1124.
[51] D. Zhu, Z. Zhang, P. Cui, W. Zhu, Robust graph convolutional networks against adversarial attacks, in: Proceedings of KDD, 2019, pp. 1399–1407.
[52] D. Zügner, S. Günnemann, Certifiable robustness and robust training for graph convolutional networks, in: Proceedings of KDD, 2019, pp. 246–256.
[53] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[54] D.B. Rubin, Multiple Imputation for Nonresponse in Surveys, Vol. 81, John Wiley & Sons, 2004.
[55] S.v. Buuren, K. Groothuis-Oudshoorn, MICE: Multivariate imputation by chained equations in r, J. Stat. Softw. (2010) 1–68.
[56] Z. Che, S. Purushotham, K. Cho, D. Sontag, Y. Liu, Recurrent neural networks for multivariate time series with missing values, Sci. Rep. 8 (1) (2018) 6085.
[57] Kai Jiang, Haixia Chen, Senmiao Yuan, Classification for incomplete data using classifier ensembles, in: Proceedings of ICNNB, 2005, pp. 559–563.
[58] K. Pelckmans, J.D. Brabanter, J. Suykens, B.D. Moor, Handling missing values in support vector machine classifiers, Neural Netw. 18 (5) (2005) 684–692.
[59] D. Williams, X. Liao, Y. Xue, L. Carin, Incomplete-data classification using logistic regression, in: Proceedings of ICML, 2005, pp. 972–979.
[60] A.J. Smola, S.V.N. Vishwanathan, T. Hofmann, Kernel methods for missing variables, in: Proceedings of AISTATS, 2005.
[61] M. Śmieja, Ł. Struski, J. Tabor, M. Marzec, Generalized rbf kernel for incomplete data, Knowl.-Based Syst. 173 (2019) 150–162.
[62] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the em algorithm, J. R. Stat. Soc. Ser. B Stat. Methodol. 39 (1) (1977) 1–38.
[63] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Mag. 29 (3) (2008) 93.
[64] J. McAuley, C. Targett, Q. Shi, A. van den Hengel, Image-based recommendations on styles and substitutes, in: Proceedings of SIGIR, 2015, pp. 43–52.
[65] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of KDD, 2019.
[66] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of AISTATS, 2010, pp. 249–256.
[67] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proceedings of ICLR, 2015.
[68] Z. Yang, W.W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, in: Proceedings of ICML, 2016, pp. 40–48.

Hibiki Taguchi received his B.E. degree from Tokyo Institute of Technology in 2019. He is now a master student of the department of computer science, school of computing, Tokyo Institute of Technology. His research interests include graph mining and machine learning.

Xin Liu is a senior researcher at Artificial Intelligence Research Center (AIRC), National Institute of Advanced Industrial Science and Technology (AIST). He received Ph.D. and M.S. degrees in Computer Science from Tokyo Institute of Technology and Wuhan University, respectively. His main research interests are graph & network analytics, data mining, machine learning, and deep learning.

Tsuyoshi Murata is a professor of the department of computer science, school of computing, Tokyo Institute of Technology. He has been doing research on artificial intelligence, especially network science, machine learning and Web mining.