Simplification of Graph Convolutional Networks: A Matrix Factorization-based Perspective

Qiang Liu[1,2], **Haoli Zhang**[1], **Zhaocheng Liu**[1]
[1]RealAI [2]Tsinghua University
{qiang.liu,haoli.zhang,zhaocheng.liu}@realai.ai

Abstract

In recent years, substantial progress has been made on Graph Convolutional Networks (GCNs). However, the computing of GCN usually requires a large memory space for keeping the entire graph. In consequence, GCN is not flexible enough, especially for complex real-world applications. Fortunately, for transductive graph representation learning, methods based on Matrix Factorization (MF) naturally support constructing mini-batches, and thus are more friendly to distributed computing compared with GCN. Accordingly, in this paper, we analyze the connections between GCN and MF, and simplify GCN as matrix factorization with unitization and co-training. Furthermore, under the guidance of our analysis, we propose an alternative model to GCN named Unitized and Co-training Matrix Factorization (UCMF). Extensive experiments have been conducted on several real-world datasets. On the task of semi-supervised node classification, the experimental results illustrate that UCMF achieves similar or superior performances compared with GCN. Meanwhile, distributed UCMF significantly outperforms distributed GCN methods, which shows that UCMF can greatly benefit real-world applications. Moreover, we have also conducted experiments on a typical task of graph embedding, i.e., community detection, and the proposed UCMF model outperforms several representative graph embedding models.

Introduction

Nowadays, works on graph convolutional networks (GCNs) (Kipf and Welling 2017) have achieved great success in many graph-based tasks, e.g., semi-supervised node classification (Veličković et al. 2017; Li, Han, and Wu 2018; Wu, He, and Xu 2019), unsupervised graph representation learning (Hassani and Khasahmadi 2020; Zhu et al. 2020), link prediction (Zhang and Chen 2018), clustering (Bo et al. 2020) and recommendation (Ying et al. 2018; Wang et al. 2019; Yu et al. 2020). GCN defines a graph convolution operation, which generates the embedding of each node by aggregating the representations of its neighbors. Given a graph, GCN performs the graph convolution operation layer by layer to obtain the final node representations, which are passed to neural networks to support various tasks.

However, as the computing of GCN requires to store the entire adjacency matrix of a graph (Chiang et al. 2019), it is hard to perform GCN on large scale real-world complex graphs, where we usually have a constrained memory size and need distributed computing. Accordingly, GCN is not flexible enough, and needs to be simplified while retaining the high performance. For example, various sampling methods have been proposed (Hamilton, Ying, and Leskovec 2017; Chen, Ma, and Xiao 2018; Ying et al. 2018) to simplify GCN via reducing the number of edges in the graph. These methods can be performed in mini-batches, but need to sample high-order neighbours of each node and require a high computational cost that exponentially grows with the number of graph convolution layers, as pointed in (Chiang et al. 2019). Instead of sampling, Cluster-GCN (Chiang et al. 2019) proposes an approach to convert computation on a huge adjacency matrix to computing on a set of small submatrices. However, Cluster-GCN still suffers from performance loss when conducting distributed computing, due to the ignoring of some connections in the graph. Simple Graph Convolution (SGC) (Wu et al. 2019) removes nonlinearities and collapses weight matrices between consecutive layers in GCN, which results in continuous multiplication of adjacency matrices, and we can obtain a final linear model. This simplifies GCN, and makes GCN applicable for distributed computing. However, as mentioned in previous works (Li, Han, and Wu 2018; Chen et al. 2020a), GCN greatly suffers from over-smoothing. And due to the continuous multiplication of adjacency matrices, SGC may aggravate the degree of over-smoothing. Accordingly, we need a simplified alternative to GCN, which is flexible enough for distributed computing in real-world applications, and can achieve similar or superior performances compared with the original GCN model.

Besides GCN, for transductive graph representation learning, graph embedding methods (Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016) are also widely applied. In general, these methods aim to embed very large graphs into low-dimensional vector spaces, to preserve the structure of graphs. As for GCN, previous work shows that the graph convolution operation is actually a special form of Laplacian smoothing (Li, Han, and Wu 2018). In this way, as the converging of the GCN model, the smoothing process can keep the final representation of a node more and more similar to those of its neighbors. Therefore, GCN is consistent with graph embedding methods in capturing the structural information. According to previous work (Qiu et al. 2018), graph embedding meth-

ods have been successfully unified as Matrix Factorization (MF). Meanwhile, compared with GCN, MF-based methods are extremely flexible and suitable for large scale distributed computing (Gemulla et al. 2011; Yu et al. 2014). These methods are also easy to be extended to various complex applications (Rendle et al. 2011; Liu, Wu, and Wang 2015b; Wu et al. 2016). Consequently, if we can simplify the GCN model as a special form of MF, large scale and complex real-world applications will greatly benefit from this.

In this paper, we analyze the connections between GCN and MF, and simplify GCN as matrix factorization with unitization and co-training. Here, the unitization indicates conducting vector unitization on node representations, i.e., forcing the norm of each node representation to one. And the co-training process means co-training with the classification task of labeled nodes, as in some previous works (Weston et al. 2012; Yang, Cohen, and Salakhutdinov 2016). Then, according to our analysis, we formally propose an alternative model to GCN named Unitized and Co-training Matrix Factorization (UCMF) [1].

We have conducted extensive experiments on several real-world graphs. The experimental results show that unitization and co-training are two essential components of UCMF. Under centralized computing settings, UCMF achieves similar or superior performances compared with GCN and SGC on the task of semi-supervised node classification. Meanwhile, both GCN and SGC perform poor on graphs that are relatively dense, while UCMF has great performances. This may be caused by the over-smoothing of graph convolution on dense graphs, while UCMF can balance the smoothing of neighbours and the classification of labeled nodes through the co-training process. Experiments under distributed computing settings are also conducted, where UCMF significantly outperforms distributed GCN methods. We have also conducted experiments on the task of community detection, and UCMF achieves better performances compared with several representative graph embedding models. These results clearly show that, we can use our proposed UCMF model as a simplified alternative to the original GCN model in real-world applications for transductive graph representation learning.

The main contributions of this paper are summarized as follows:

- We analyze the connections between GCN and MF, and simplify GCN as a special form of matrix factorization with unitization and co-training.
- We propose an alternative model to GCN, i.e., unitized and co-training matrix factorization.
- On the task of semi-supervised node classification, extensive experiments have been conducted on several real-world datasets under both centralized and distributed computing settings, and demonstrate the effectiveness and flexibility of UCMF. Meanwhile, on the task of community detection, UCMF outperforms several representative graph embedding models, e.g., LINE (Tang et al. 2015) and DeepWalk (Perozzi, Al-Rfou, and Skiena 2014).

[1] Code avalable at: https://github.com/johnlq/UCMF.

GCN as Unitized and Co-training MF

In this section, we are going to simplify GCN as a special form of matrix factorization. First, we start from the analysis of how node representations are learned in GCN. Then, we successfully simplify GCN as matrix factorization with unitization and co-training.

Graph Convolutional Networks

According to the definition in (Kipf and Welling 2017), we can formulate each layer of GCN as

$$\mathbf{H}^{(l+1)} = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\widetilde{\mathbf{A}}\,\widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(l)}\,\mathbf{W}^{(l+1)}\right), \qquad (1)$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix of the graph G with added self-connections, $\mathbf{I}_N$ is the identity matrix for N nodes in graph G, $\widetilde{\mathbf{D}}$ is a diagonal degree matrix with $\widetilde{\mathbf{D}}_{i,i} = \sum_j \widetilde{\mathbf{A}}_{i,j}$, $\mathbf{H}^{(l)}$ is the representation of each node at layer l, $\mathbf{W}^{(l+1)}$ is a layer-specific trainable weight matrix, and $\sigma(\cdot)$ denotes an activation function (such as $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$).

For node classification task, we can obtain a classification loss

$$l_{\mathrm{c}} = \mathrm{CrossEntropy}\left(\mathbf{Y}, \mathrm{softmax}\left(\mathbf{H}^{(-1)}\right)\right), \qquad (2)$$

where $\mathbf{Y}$ is the ground truth labels for the classification task, $\mathbf{H}^{(-1)}$ is the representation of each node at the final layer of GCN. Via optimizing Eq. (2), the cross-entropy error of the node classification task can be minimized, and the GCN model can be learned.

Simplification

In (Li, Han, and Wu 2018), GCN has been proved to be a special form of Laplacian smoothing. As the GCN model goes deeper and we have more layers of graph convolution, the representations in Eq. (1) have a termination condition as

$$\mathbf{H}^{(-1)} = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\widetilde{\mathbf{A}}\,\widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(-1)}\,\mathbf{W}^{(-1)}\right), \qquad (3)$$

where $\mathbf{H}^{(-1)}$ is the final representations on the last layer of GCN. In the simplification discussed in SGC (Wu et al. 2019), nonlinear activation function is ignored. Meanwhile, according to the implementation in some previous works (Kipf and Welling 2017; Veličković et al. 2017; Wu, He, and Xu 2019), there is no activation function on the last layer of GCN. Thus, an approximate solution of Eq. (3) can be written as

$$\mathbf{H}^{(-1)} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\widetilde{\mathbf{A}}\,\widetilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(-1)}. \qquad (4)$$

More specifically, for each node i in graph G, the approximate solution of the corresponding final representation is

$$h_i^{(-1)} = \sum_{j\in I} \frac{1}{\sqrt{(d_i+1)\,(d_j+1)}}\,\mathbf{A}_{i,j}\,h_j^{(-1)} + \frac{1}{d_i+1}\,h_i^{(-1)}, \quad (5)$$

from which we have

$$h_i^{(-1)} = \sum_{j\in I} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\,h_j^{(-1)}, \qquad (6)$$

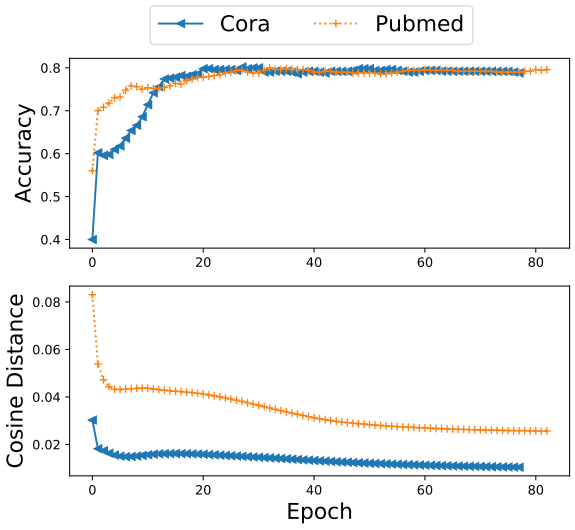

Figure 1: Consistency estimation between the changing of cosine distance and the convergence of GCN during the training procedure on the Cora dataset and the Pubmed dataset. The upper part shows the accuracy curves during the training of GCN. The lower part shows the average cosine distance between representations of nodes in the graph and those of their neighbors during the training of GCN.

where I denotes the set of all the nodes in graph G, and d_i is the degree of node i.

According to above analysis, to train an approximate GCN model, which can simultaneously model the structure of graph convolution and the node classification task, we can minimize the following loss function

$$l = \alpha\, l_c + (1-\alpha)\, l_s, \tag{7}$$

where α is a hyper-parameter to control the balance between the two losses, and the structure loss l_s refers to

$$l_s = \sum_{i\in I} \mathrm{dis}\left(h_i^{(-1)}, \sum_{j\in I} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\, h_j^{(-1)}\right), \tag{8}$$

where $\mathrm{dis}(\cdot,\cdot)$ is a distance measurement. Here, we adopt the commonly-used cosine distance, and obtain

$$l_s = \sum_{i\in I} \mathrm{Cosine}\left(h_i^{(-1)}, \sum_{j\in I} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\, h_j^{(-1)}\right), \tag{9}$$

which is equivalent to

$$l_s = -\sum_{i\in I} \frac{h_i^{(-1)}\left(\sum_{j\in I} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\, h_j^{(-1)}\right)^\top}{\left\|h_i^{(-1)}\right\|\left\|\sum_{j\in I} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\, h_j^{(-1)}\right\|}. \tag{10}$$

To verify whether the changing of cosine distance is consistent with the convergence of GCN during the training procedure, we conduct empirical experiments and train GCN models on the Cora dataset and the Pubmed dataset. Fig. 1 demonstrates the average cosine distance between representations of nodes in the graph and those of their neighbors, as well as the convergence curves estimated by accuracy during the training of GCN on the two datasets. It is obvious that, the tendencies of curves on the same dataset match with each other. That is to say, the changing of cosine distance is consistent with the convergence of the GCN model.

Then, to simplify the form of Eq. (10), we conduct vector unitization on the learned representations $\mathbf{H}^{(-1)}$, and thus each representation $h_i^{(-1)}$ has similar l2-norm. As a result, through unitization, Eq. (10) is equivalent to

$$l_s = -\sum_{i\in I}\sum_{j\in C_i} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j}\, v_i\, v_j^\top, \tag{11}$$

where C_i denotes all the nodes that node i is connected to, and $v_i = h_i^{(-1)}$ for simplicity. Moreover, for better optimization, we can incorporate negative log likelihood and minimize the following loss function equivalently to Eq. (11)

$$l_s = -\sum_{i\in I}\sum_{j\in C_i} \frac{1}{d_i}\sqrt{\frac{d_i+1}{d_j+1}}\,\mathbf{A}_{i,j} \log\left(\lambda\left(v_i\, v_j^\top\right)\right), \tag{12}$$

where $\lambda(\cdot) = \mathrm{sigmoid}(\cdot)$.

Usually, in graph embedding methods (Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016), negative sampling of edges is used, for better convergence. Thus, we can randomly sample negative edges for each edge in graph G. Following previous works in unifying word embedding (Levy and Goldberg 2014) and graph embedding (Qiu et al. 2018) as implicit matrix factorization, we can rewrite Eq. (12) as

$$\begin{aligned} l_s = &-\sum_{i\in I}\sum_{j\in C_i} \beta_{i,j}\,\mathbf{A}_{i,j} \log\left(\lambda\left(v_i\, v_j^\top\right)\right) \\ &-\sum_{i\in I} k d_i \mathbb{E}_{j'\sim P_G}\left[\beta_{i,j'} \log\left(\lambda\left(-v_i\, v_{j'}^\top\right)\right)\right] \end{aligned}, \tag{13}$$

where $\beta_{i,j} = d_i^{-1}\left(d_i+1\right)^{1/2}\left(d_j+1\right)^{-1/2}$, k is the number of negative samples for each edge, and P_G denotes the distribution that generates negative samples in graph G. For each node i, $P_G(i) = d_i / |G|$, where $|G|$ is the number of edges in graph G. We can explicitly express the expectation term as

$$\mathbb{E}_{j'\sim P_G}\left[\beta_{i,j'} \log\left(\lambda\left(-v_i\, v_{j'}^\top\right)\right)\right] = \sum_{j'\in I} \frac{\beta_{i,j'}\, d_{j'}}{|G|} \log\left(\lambda\left(-v_i\, v_{j'}^\top\right)\right), \tag{14}$$

from which we have

$$\begin{aligned} \mathbb{E}_{j'\sim P_G}\left[\beta_{i,j'} \log\left(\lambda\left(-v_i\, v_{j'}^\top\right)\right)\right] = &\frac{\beta_{i,j}\, d_j}{|G|} \log\left(\lambda\left(-v_i\, v_j^\top\right)\right) \\ &+ \sum_{j'\in I\setminus\{j\}} \frac{\beta_{i,j'}\, d_{j'}}{|G|} \log\left(\lambda\left(-v_i\, v_{j'}^\top\right)\right) \end{aligned}. \tag{15}$$

Then, we can obtain the local structure loss for a specific edge (i,j) as

$$\begin{aligned} l_s(i,j) = &-\beta_{i,j}\,\mathbf{A}_{i,j} \log\left(\lambda\left(v_i\, v_j^\top\right)\right) \\ &-\frac{k\,\beta_{i,j}\, d_i\, d_j}{|G|} \log\left(\lambda\left(-v_i\, v_j^\top\right)\right) \end{aligned}. \tag{16}$$

To optimize above objective, we need to calculate the partial derivative of $l_s\left(i,j\right)$ with respect to $v_i\,v_j^\top$

$$\frac{\partial\, l_s\left(i,j\right)}{\partial\left(v_i\,v_j^\top\right)} = -\,\beta_{i,j}\,\mathbf{A}_{i,j}\,\lambda\left(-\,v_i\,v_j^\top\right) + \frac{k\,\beta_{i,j}\,d_i\,d_j}{|G|}\lambda\left(v_i\,v_j^\top\right). \quad (17)$$

Via setting Eq. (17) to zero, we can obtain

$$e^{2\,v_i\,v_j^\top} - \left(\frac{|G|\,\mathbf{A}_{i,j}}{k\,d_i\,d_j} - 1\right)e^{v_i\,v_j^\top} - \frac{|G|\,\mathbf{A}_{i,j}}{k\,d_i\,d_j} = 0, \quad (18)$$

which has two solutions, $e^{v_i\,v_j^\top} = -1$ and

$$v_i\,v_j^\top = \log\left(\frac{|G|\,\mathbf{A}_{i,j}}{k\,d_i\,d_j}\right). \quad (19)$$

Accordingly, the GCN model can be simplified as the following matrix factorization

$$\mathbf{V}\mathbf{V}^\top = \log\left(|G|\,\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\right) - \log\left(k\right), \quad (20)$$

co-trained with the classification loss l_{c}, where node representations in $\mathbf{V}$ are unitized. $\mathbf{D}$ is a diagonal degree matrix with $\mathbf{D}_{i,i} = d_i$. According to previous analysis (Levy and Goldberg 2014; Qiu et al. 2018; Du et al. 2018), the matrix factorization in Eq. (20) is as the same as common implicit matrix factorization. In summary, we successfully simplify GCN as matrix factorization with unitization and co-training.

Discussion

In this section, we perform more discussion about our simplification of GCN.

Adopting Euler Distance

Besides cosine distance, Euler distance is another commonly-used distance measurement which can be adopted in Eq. (8). Here, we need to investigate whether the conclusion in Eq. (20) stays the same when Euler distance is adopted.

Suppose we have two node representations $p = [p_1, ..., p_d]$ and $q = [q_1, ..., q_d]$. As discussed in the simplification section, we conduct vector unitization on the node representations, which means

$$\sum_{i=1}^{d} p_i^2 = 1, \quad \sum_{i=1}^{d} q_i^2 = 1. \quad (21)$$

The cosine distance between p and q can be formulated as

$$\mathrm{Cosine}\left(p,q\right) = 1 - \frac{pq^\top}{\|p\|\,\|q\|}. \quad (22)$$

Considering Eq. (21), we have

$$\mathrm{Cosine}\left(p,q\right) = 1 - pq^\top = 1 - \sum_{i=1}^{d} p_i q_i. \quad (23)$$

Meanwhile, the Euler distance between p and q can be formulated as

$$\begin{aligned}\mathrm{Euler}\left(p,q\right) &= \sqrt{\sum_{i=1}^{d}\left(p_i - q_i\right)^2} \\ &= \sqrt{\sum_{i=1}^{d} p_i^2 + \sum_{i=1}^{d} q_i^2 - 2\sum_{i=1}^{d} p_i\,q_i}\end{aligned}. \quad (24)$$

Considering Eq. (21), we can obtain

$$\mathrm{Euler}\left(p,q\right) = \sqrt{2 - 2\sum_{i=1}^{d} p_i q_i}. \quad (25)$$

Combining Eq. (23) and Eq. (25), we can conclude the connection between cosine distance and Euler distance as

$$\mathrm{Euler}\left(p,q\right) = \sqrt{2\mathrm{Cosine}\left(p,q\right)}. \quad (26)$$

Accordingly, adopting the Euler distance, the loss function in Eq. (11) can be rewritten as

$$l_s = \sum_{i\in I}\sqrt{2 - 2\,v_i\left(\sum_{j\in C_i}\beta_{i,j}\,\mathbf{A}_{i,j}\,v_j\right)^\top}. \quad (27)$$

To optimize Eq. (27), we can optimize the following loss function equivalently

$$\begin{aligned} l_s &= \sum_{i\in I}\left(2 - 2\,v_i\left(\sum_{j\in C_i}\beta_{i,j}\,\mathbf{A}_{i,j}\,v_j\right)^\top\right) \\ &= 2\,|V| - 2\sum_{i\in I}\sum_{j\in C_i}\beta_{i,j}\,\mathbf{A}_{i,j}\,v_i\,v_i^\top \end{aligned}, \quad (28)$$

where $|V|$ is the number of nodes in graph G. It is obvious that, the loss function in Eq. (28) is equivalent to Eq. (11). Thus, when we adopt the Euler distance in Eq. (8), we can equivalently obtain the same matrix factorization as in Eq. (20).

Another Form of Graph Convolution

Besides Eq. (1), there is another form of graph convolution for GCN (Kipf and Welling 2017; Li, Han, and Wu 2018), which can be formulated as

$$\mathbf{H}^{(l+1)} = \sigma\left(\widetilde{\mathbf{D}}^{-1}\,\widetilde{\mathbf{A}}\,\mathbf{H}^{(l)}\,\mathbf{W}^{(l)}\right). \quad (29)$$

Then, Eq. (5) can be rewritten as

$$h_i^{(-1)} = \sum_{j\in I}\frac{1}{d_i+1}\,\mathbf{A}_{i,j}\,h_j^{(-1)} + \frac{1}{d_i+1}\,h_i^{(-1)}, \quad (30)$$

from which we have

$$h_i^{(-1)} = \sum_{j\in I}\frac{1}{d_i}\,\mathbf{A}_{i,j}\,h_j^{(-1)}. \quad (31)$$

Then, Eq. (11) can be rewritten as

$$l_s = -\sum_{i\in I}\sum_{j\in C_i}\frac{1}{d_i}\,\mathbf{A}_{i,j}\,v_i\,v_j^\top. \quad (32)$$

Accordingly, the only difference between Eq. (11) and Eq. (32) is that, instead of $\beta_{i,j} = d_i^{-1}\left(d_i+1\right)^{1/2}\left(d_j+1\right)^{-1/2}$ in Eq. (11), we have $\beta_{i,j} = d_i^{-1}$ in Eq. (32). Meanwhile, $\beta_{i,j}$ can be eliminated during the analysis in Eq. (12-19). It is obvious that, with the new form of graph convolution, we can still obtain the same matrix factorization as in Eq. (20).

The UCMF Architecture

In this section, we formally propose the UCMF architecture. We first need to deal with node features, which can not be directly handled in the original implicit matrix factorization. Let x_i denote the feature vector of node i, and $f_1(\cdot)$ denote the first Multi-Layer Perception (MLP) for feature modeling. According to our analysis, given x_i and $f_1(\cdot)$, we conduct vector unitization to obtain v_i, i.e., the representation of node i, as

$$v_i = \frac{f_1(x_i)}{\|f_1(x_i)\|}. \tag{33}$$

Then, following our previous analysis, UCMF consists of two losses: the structure loss l_s and the classification loss l_c. The structure loss l_s can be formulated as implicit matrix factorization with k negative samples for each edge

$$\begin{aligned} l_s = &- \sum_{i \in I} \sum_{j \in C_i} \log\left(\lambda\left(v_i\, v_j^\top\right)\right) \\ &- \sum_{i \in I} k d_i \mathbb{E}_{j' \sim P_G}\left[\log\left(\lambda\left(-\, v_i\, v_{j'}^\top\right)\right)\right] \end{aligned}. \tag{34}$$

Meanwhile, the prediction on node classification can be made as

$$\hat{y}_i = \mathrm{softmax}\left(f_2\left(v_i\right)\right), \tag{35}$$

where $f_2(\cdot)$ is the second MLP for making predictions. As in GCN, the classification loss l_c can be obtained as

$$l_{\mathrm{c}} = \sum_{i \in I_L} \mathrm{CrossEntropy}\left(y_i, \hat{y}_i\right), \tag{36}$$

where I_L is the set of labeled nodes in the graph, and y_i is the ground-truth label of node i. Co-training the two losses as in Eq. (7), we obtain the final loss function of the proposed UCMF model.

Furthermore, following some previous works on semi-supervised node classification (Weston et al. 2012; Yang, Cohen, and Salakhutdinov 2016), during co-training, the two losses l_s and l_c are alternately optimized. To be more specific, we first optimize the structure loss l_s with b batches of samples, then we optimize the classification loss l_c with one batch of samples. We repeat this process until convergence. Here, the parameter b is the balance parameter between the two losses.

Compared with previous graph modeling methods (Weston et al. 2012; Yang, Cohen, and Salakhutdinov 2016; Kipf and Welling 2017; Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016), the most unique part of UCMF is the unitization of node representations, which is derived from our above analysis. This has also been incorporated in (Hamilton, Ying, and Leskovec 2017). With unitization, it reduces the possibility of extreme values, and has chance to generate better node representations.

Related Works

In recent years, GCN (Bruna et al. 2013; Kipf and Welling 2017) has drawn tremendous attention from academia. It updates node representations with the aggregation of its neighbors. Based on GCN, Graph Attention Network (GAT) (Veličković et al. 2017) introduces the attention mechanism to model different influences of neighbors with learnable parameters.

As mentioned in previous works (Li, Han, and Wu 2018), GCN greatly suffers from over-smoothing, and thus it is hard for GCN to go deeper. In (Li, Han, and Wu 2018), the authors propose to add more supervision for training a deeper GCN. JK-Nets (Xu et al. 2018) presents general layer aggregation mechanisms to combine the output representation in every GCN layer. ResGCN and DenseGCN (Li et al. 2019) incorporate some technologies from computer vision, i.e., residual connections (He et al. 2016) and dense connections (Huang et al. 2017), to tackle with this problem. Hierarchical Graph Convolutional Network (H-GCN) (Hu et al. 2019) applies coarsening and refining operations to make GCN deeper. In (Chen et al. 2020b), the authors add initial residual and identity mapping into the GCN model. In (Chen et al. 2020a), the authors design some metrics for measuring the degree of over-smoothing, and accordingly propose an approach based on regularization to overcome the problem.

Another severe problem of GCN is that, it requires the entire adjacency matrix of the graph, which makes GCN not flexible on large scale graphs. For sake of flexibility, some works try to simplify GCN from different perspectives. GraphSAGE (Hamilton, Ying, and Leskovec 2017) samples a fixed number of neighbors for each node in the graph. FastGCN (Chen, Ma, and Xiao 2018) proposes to apply importance sampling to reduce the computation of aggregation of neighbors on graph. Instead of approximating the node representations, variance controlled GCN (Chen, Zhu, and Song 2018) uses sampled node to estimate the change of node representations in every updating step. These sampling-based methods can be implemented in mini-batches, via sampling high-order neighbours of each node in the graph. However, as pointed in (Chiang et al. 2019), this causes high computational cost growing exponentially with the number of layers. To tackle with this, Cluster-GCN (Chiang et al. 2019) uses graph partition method (Karypis and Kumar 1998) to split the whole graph into a set of small sub-graphs, where neighbour aggregation happens within each small sub-graph. With this improvement, Cluster-GCN supports constructing mini-batches and distributed computing. Simple Graph Convolution (SGC) (Wu et al. 2019) simplifies GCN to continuous multiplication of adjacency matrices with a linear classifier. Considering SGC is actually a linear model, it is capable to be distributed implemented. However, due to the continuous multiplication of adjacency matrices, SGC may aggravates the degree of over-smoothing.

Experiments

In this section, we empirically evaluate the performance of the proposed UCMF model. We first describe the datasets and settings of the experiments, then report and analyze the experimental results. Thorough evaluations are conducted to answer the following research questions:

- **RQ1** What are the roles of different components in the UCMF model, i.e., unitization and co-training?
- **RQ2** How are the performances of our UCMF model compared with those of GCN on different datasets?

- **RQ3** How do the two hyper-parameters, i.e., the negative sampling number k and the balance parameter b, affect the performances of UCMF?
- **RQ4** Compared with GCN, is our proposed UCMF model more friendly to distributed computing?
- **RQ5** How does UCMF perform on community detection, a typical task of graph embedding?

Experimental Datasets

We evaluate our proposed model on five real-world datasets, i.e., Cora, Citeseer, Pubmed, BlogCatalog and Flickr. Cora, Citeseer and Pubmed (Sen et al. 2008) are three standard citation network benchmark datasets[2], which are widely used in previous works (Kipf and Welling 2017; Veličković et al. 2017; Chiang et al. 2019). BlogCatalog and Flickr (Huang, Li, and Hu 2017) are two social network datasets[3]. The posted keywords or tags in BlogCatalog and Flickr networks are used as node features. For the splitting of Cora, Citeseer and Pubmed, we follow the classical settings in previous works (Yang, Cohen, and Salakhutdinov 2016; Kipf and Welling 2017). And the splitting of BlogCatalog and Flickr is the same as in (Wu, He, and Xu 2019). Specifically, on BlogCatalog and Flickr, we randomly select 10% and 20% of the nodes for training and validation respectively, and the rest 70% as our testing set. The sparsity of Cora, Citeseer, Pubmed, BlogCatalog and Flickr is 99.85%, 99.91%, 99.97%, 98.73% and 99.15% respectively. Accordingly, we have two different categories of datasets: sparse and dense. Sparse datasets consist of Cora, Citeseer and Pubmed. Dense datasets consist of BlogCatalog and Flickr.

Experimental Settings

In our experiments, we run each model 10 times with random weight initialization, and report the average evaluation values, as well as statistically significant improvement measured by t-test with p-value < 0.05. When we implement UCMF and its extended variations, we set batch size as 256, the dimensionality of node representation v_i as 10% of the dimensionality of original node features on each dataset, the dropout rate as 0.5, the l2 regularization as 0.002, and tune the learning rate in $[0.001, 0.005, 0.01]$. The first MLP $f_1(\cdot)$ for feature modeling is with one layer, which outputs the node representations. And the second MLP $f_2(\cdot)$ for making predictions is with two layers, where the number of hidden neurons is set as 128. Moreover, to answer the five research questions, we have conducted extensive experiments from different perspectives. We introduce the compared models and detailed settings as follow.

Firstly, we need to investigate the effects of the two components in UCMF, i.e., co-training and unitization. We conduct experiments on semi-supervised node classification, and report the results in terms of accuracy. To clarify the contribution of the two components, comparisons are conducted without utilizing node features. In this experiment, besides the **UCMF** model, we include another two variations of UCMF into the comparison: **UCMF-C** and **UCMF-U**. UCMF-C means the UCMF model without co-training with the classification loss, and UCMF-U indicates the UCMF model without vector unitization on node representations. We also include three commonly-used graph embedding methods in our experiment: **MF**, **DeepWalk** (Perozzi, Al-Rfou, and Skiena 2014) and **node2vec** (Grover and Leskovec 2016), which are incapable to directly take node features into consideration. The **GCN** model (Kipf and Welling 2017) is also taken into account, where we have two layers of graph convolution. Considering we do not utilize node features in this comparison, the input node features in GCN and UCMF are replaced with learnable embeddings of each node.

compared model	Cora	Citeseer	Pubmed	BlogCatalog	Flickr
MF	50.8	33.3	43.6	45.8	24.6
DeepWalk	67.2	43.2	65.3	61.8	41.5
node2vec	67.8	43.5	**65.8**	63.1	42.2
UCMF-U	51.2	35.4	44.4	60.9	38.6
UCMF-C	55.1	35.5	44.2	53.1	25.1
UCMF	**69.1***	**46.2***	**66.2***	**66.2***	**43.2***
GCN	65.6	44.4	58.3	58.3	30.9

Table 1: Performance comparison on semi-supervised node classification in terms of classification accuracy (%) without utilizing node features. The larger the values, the better the performances. Competitive performances on each dataset are highlighted. $*$ denotes statistically significant improvement measured by t-test with p-value < 0.05.

Secondly, to verify the effectiveness of UCMF, we conduct experiments on semi-supervised node classification utilizing node features. Besides **UCMF** and **GCN** (Kipf and Welling 2017), we involve **Planetoid** (Yang, Cohen, and Salakhutdinov 2016) as a baseline. And two simplified GCN models, i.e., **fastGCN** (Chen, Ma, and Xiao 2018) and **SGC** (Wu et al. 2019), are also compared. In our experiments, all the GCN-based models, i.e., GCN, fastGCN and SGC, are with two layers of graph convolution. And we follow the default hyper-parameter settings in the corresponding original papers.

Thirdly, we investigate the effects of the two important hyper-parameters, i.e., the negative sampling number k and the balance parameter b, on model performances. In this experiment, node features are utilized. We report the accuracy of **UCMF** on semi-supervised node classification with respect to different k and b.

Then, to test the performances under distributed computing settings, we compare **UCMF** with several distributed GCN models on semi-supervised node classification. In the comparison, we involve the state-of-the-art distributed GCN model **Cluster-GCN** (Chiang et al. 2019), as well as the baseline model **Random-GCN** in the corresponding paper. Moreover, **SGC** (Wu et al. 2019) is also involved in the comparison, for it is actually a linear model and can be distributed implemented. We also run **fastGCN** (Chen, Ma, and Xiao 2018) in mini-batches via sampling high-order neighbours of each node in the graph, though this causes exponential complexity with the number of layers as pointed in

[2]https://github.com/tkipf/gcn

[3]https://github.com/xhuang31/LANE

compared model	Cora	Citeseer	Pubmed	BlogCatalog	Flickr
Planetoid	75.7	64.3	77.2	84.7	70.9
GCN	**81.2**	70.3	79.0	65.2	62.8
fastGCN	78.8	68.8	77.4	64.2	61.6
SGC	**81.0**	**71.9**	78.9	58.8	37.2
UCMF	**81.4**	**71.5**	**80.4***	**91.6***	**77.8***

Table 2: Performance comparison on semi-supervised node classification in terms of classification accuracy (%) utilizing node features. The larger the values, the better the performances. Competitive performances on each dataset are highlighted. $*$ denotes statistically significant improvement measured by t-test with p-value < 0.05.

(Chiang et al. 2019). Under distributed computing settings, parameters of each model are learned with the Parameter Server (PS) architecture (Li et al. 2014), where we have 1 server and 2 workers. Each graph is partitioned into two subgraphs for computing on the two workers. For Cluster-GCN, following (Chiang et al. 2019), the partition is conducted with the METIS algorithm (Karypis and Kumar 1998). And for Random-GCN, fastGCN, SGC and UCMF, the partition is randomly conducted.

Finally, we want to see if UCMF can perform well on typical tasks of graph embedding. Thus, we conduct experiments on community detection, and report the results in terms of conductance. Conductance is basically the ratio between the number of edges leaving a community and that within the community. We perform k-means clustering on node representations to generate communities, where each cluster refers to one community. The number of communities is the same as the number of classes on each dataset. We involve several representative graph embedding methods: **MF**, **LINE** (Tang et al. 2015), **DeepWalk** (Perozzi, Al-Rfou, and Skiena 2014) and **node2vec** (Grover and Leskovec 2016). Considering community detection is an unsupervised task and we should not use supervision information, we perform **UCMF-C** in this experiment.

Performance Analysis

Experimental results are illustrated in Tab. 1-4 and Fig. 2.

The Roles of Unitization and Co-training (RQ1)

Tab. 1 illustrates the results of performance comparison without utilizing node features. It is clear that, both UCMF-C and UCMF-U outperforms the conventional MF. Meanwhile, UCMF can achieve better performances comparing with both UCMF-C and UCMF-U. Moreover, UCMF and GCN perform closely on some datasets, and UCMF achieves the best performances on all datasets. It is also worth to notice that, when we do not have node features, GCN has no obvious advantages comparing with DeepWalk and node2vec. In average, UCMF relatively outperforms UCMF-C, UCMF-U and GCN 36.6%, 26.2% and 13.0% respectively. These observations show that unitization and co-training are two useful and essential components of UCMF.

UCMF vs. GCN (RQ2)

Performance comparison on semi-supervised node classification are shown in Tab. 2. It is obvious that, GCN outperforms fastGCN on all five datasets. This indicates that the edge sampling method may lead to somewhat performance loss. We can also observe that, UCMF, SGC and GCN achieve competitive performances on Cora, Citeseer and Pubmed. This shows that, the two simplified GCN models inherit the effectiveness of GCN. Moreover, on BlogCatalog and Flickr, both GCN and SGC have poor performances. According to the sparsity of experimental datasets mentioned above, BlogCatalog and Flickr are relatively dense compared with the other three datasets. That is to say, models based on graph convolution perform poor on dense graphs. This may cause by the over-smoothing of graph convolution as mentioned in (Li, Han, and Wu 2018; Chen et al. 2020a), and this becomes more serious on dense graphs. Meanwhile, UCMF can balance the smoothing of neighbours and the classification of labeled nodes through the co-training process. It is also worth to notice that, on BlogCatalog and Flickr, GCN outperforms SGC. This may indicate that, SGC aggravates the degree of over-smoothing of GCN. In summary, UCMF achieves the best performances on all datasets except Citeseer. On Citeseer, UCMF performs very closely to SGC. Meanwhile, on BlogCatalog and Flickr, the advantages of UCMF are extremely large. In average, UCMF relatively outperforms GCN, fast-GCN and SGC 12.3%, 14.7% and 22.7% respectively. These results illustrate the effectiveness of UCMF.

Stability Analysis (RQ3)

As shown in Fig. 2, we investigate the effects of the two important hyper-parameters, i.e., the negative sampling number k and the balance parameter b, on the performances of UCMF. The flat lines in the left part of Fig. 2 demonstrate that k has little effects on the performances of UCMF. That is to say, UCMF is stable with different negative sampling numbers. As shown in the right part of Fig. 2, on sparse datasets, i.e. Cora, Citeseer and Pubmed, b has little impact on the performances of UCMF. Meanwhile, on dense datasets, i.e., BlogCatalog and Flickr, the performances of the UCMF model slightly decrease with the increasing of b. The hyper-parameter b balances the structure loss and the classification loss. The larger value of b indicates that UCMF pays more attention to capturing structural information. That is to say, when UCMF pays too much attention to graph structure, it faces performance loss. Combining with the viewpoint in (Li, Han, and Wu 2018; Chen et al. 2020a), this may give a reason for the poor performances of GCN on dense datasets: the graph convolution operation is easily to be extremely over-smoothing for capturing structural information on dense datasets. Moreover, though performances of the UCMF model decrease when b is large on dense datasets, the performance loss is slight. Overall, the performances of UCMF are relatively stable with hyper-parameters. According to the observations from Fig. 2, for other experiments, we set $k = 16$ on all datasets, $b = 15$ on sparse datasets, and $b = 5$ on dense datasets.

Distributed UCMF vs. Distributed GCN (RQ4)

Performance comparison under distributed settings is shown in Tab. 3. Recalling the results in Tab. 2, it is clear that both Cluster-GCN and Random-GCN greatly suffer performance loss. Via sampling high-order neighbours

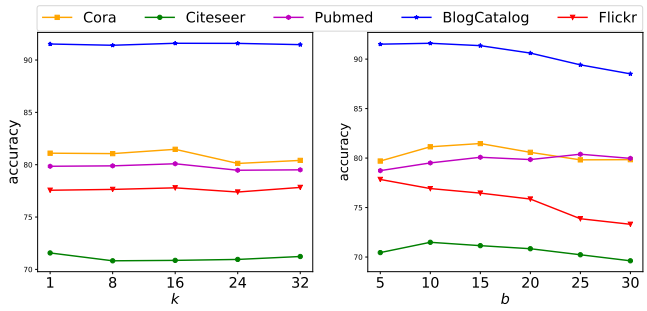

Figure 2: Performances of UCMF with varying hyper-parameters: (1) the left part shows the impact of the negative sampling number k; (2) the right part shows the impact of the balance parameter b between structure loss and classification loss.

compared model	Cora	Citeseer	Pubmed	BlogCatalog	Flickr
Random-GCN	71.7	61.9	71.8	55.2	53.9
Cluster-GCN	77.1	64.8	76.6	59.8	57.5
fastGCN	77.6	68.7	77.1	64.1	61.2
SGC	**80.8**	**71.5**	78.6	58.4	36.9
UCMF	**81.2***	**71.3**	**80.3***	**91.5***	**77.6***

Table 3: Performance comparison on semi-supervised node classification in terms of classification accuracy (%) under distributed computing settings. The larger the values, the better the performances. Competitive performances on each dataset are highlighted. $*$ denotes statistically significant improvement measured by t-test with p-value < 0.05.

of each node, fastGCN stays the performances as in Tab. 2. Meanwhile, inherit from the capacity of linear model and MF, SGC and UCMF are extremely flexible for distributed computing, and both achieve very similar performances as in Tab. 2. However, the performances of SGC are still poor on BlogCatalog and Flickr, because of the over-smoothing problem. In average, under distributed settings, UCMF relatively outperforms Cluster-GCN, fast-GCN and SGC 19.8%, 15.4% and 23.3% respectively. Overall, UCMF enlarges its advantages under distributed settings. These results strongly demonstrate the flexibility of UCMF.

Results on Community Detection (RQ5)

Tab. 4 illustrates performance comparison on the task of community detection. It is clear that, our proposed UCMF-C model can achieve best performances on all datasets. This show that, the unitization module in UCMF is very useful, for it can reduce the possibility of extreme values, and has chance to generate better node representations. These observations clearly demonstrate that, UCMF can perform well on the task of community detection, which is as typical task of graph embedding.

Conclusion

In this paper, we simplify GCN as unitized and co-training matrix factorization, and the UCMF model is therefore proposed. We conduct thorough and empirical experiments, which strongly verify our analysis. The experimental results

compared model	Cora	Citeseer	Pubmed	BlogCatalog	Flickr
MF	25.5	27.5	26.6	76.5	50.1
LINE	21.2	29.9	33.3	81.5	55.9
DeepWalk	26.3	20.2	21.6	76.2	49.6
node2vec	25.8	19.6	20.5	74.8	**48.5**
UCMF-C	**12.3***	**7.4***	**14.3***	**71.4***	**48.1***

Table 4: Performance comparison on community detection in terms of conductance. The smaller the values, the better the performances. Competitive performances on each dataset are highlighted. $*$ denotes statistically significant improvement measured by t-test with p-value < 0.05.

on semi-supervised node classification show that UCMF achieves similar or superior performances compared with GCN. We also observe that GCN performs poor on dense graphs, while UCMF has great performances. This may be caused by the over-smoothing of graph convolution on dense graphs, while UCMF can balance the smoothing of neighbours and the classification of labeled nodes via co-training. Moreover, due to the MF-based architecture, UCMF is exceedingly flexible and convenient to be applied to distributed computing for real-world applications, and significantly outperforms distributed GCN methods. Meanwhile, on the task of community detection, our proposed UCMF model can achieve better performance compared with several representative graph embedding models. Extensive experimental results clearly show that, we can use UCMF as an alternative to GCN for transductive graph representation learning in various real-world applications.

References

Bo, D.; Wang, X.; Shi, C.; Zhu, M.; Lu, E.; and Cui, P. 2020. Structural Deep Clustering Network. In *Proceedings of The Web Conference 2020*, 1400–1410.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* .

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020a. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI Conference on Artificial Intelligence*.

Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* .

Chen, J.; Zhu, J.; and Song, L. 2018. Stochastic training of graph convolutional networks with variance reduction. In *Neural Information Processing Systems*.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020b. Simple and Deep Graph Convolutional Networks. In *ICML*.

Chiang, W.-L.; Liu, X.; Si, S.; Li, Y.; Bengio, S.; and Hsieh, C.-J. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Du, L.; Wang, Y.; Song, G.; Lu, Z.; and Wang, J. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2086–2092.

Gemulla, R.; Nijkamp, E.; Haas, P. J.; and Sismanis, Y. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 69–77. ACM.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive Multi-View Representation Learning on Graphs. *arXiv preprint arXiv:2006.05582* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hu, F.; Zhu, Y.; Wu, S.; Wang, L.; and Tan, T. 2019. Hierarchical Graph Convolutional Networks for Semi-supervised Node Classification. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 10–16.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Huang, X.; Li, J.; and Hu, X. 2017. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 731–739. ACM.

Karypis, G.; and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20(1): 359–392.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Levy, O.; and Goldberg, Y. 2014. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, 2177–2185.

Li, G.; Muller, M.; Thabet, A.; and Ghanem, B. 2019. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, 9267–9276.

Li, M.; Andersen, D. G.; Park, J. W.; Smola, A. J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E. J.; and Su, B.-Y. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 583–598.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Liu, Q.; Wu, S.; and Wang, L. 2015a. Collaborative prediction for multi-entity interaction with hierarchical representation. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 613–622. ACM.

Liu, Q.; Wu, S.; and Wang, L. 2015b. Cot: Contextual operating tensor for context-aware recommender systems. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710.

Qiu, J.; Dong, Y.; Ma, H.; Li, J.; Wang, K.; and Tang, J. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 459–467.

Rendle, S.; Gantner, Z.; Freudenthaler, C.; and Schmidt-Thieme, L. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 635–644. ACM.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassirad, T. 2008. Collective Classification in Network Data. *Ai Magazine* 29(3): 93–106.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. In *International Conference on Learning Representations*.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T.-S. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 165–174.

Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2012. Deep Learning via Semi-supervised Embedding. In *International Conference on Machine Learning*.

Wu, F.; Zhang, T.; Souza Jr, A. H. d.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* .

Wu, J.; He, J.; and Xu, J. 2019. Demo-net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM*.

Wu, S.; Liu, Q.; Wang, L.; and Tan, T. 2016. Contextual operation for recommender systems. *IEEE Transactions on Knowledge and Data Engineering* 28(8): 2000–2012.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on

graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* .

Yang, Z.; Cohen, W.; and Salakhutdinov, R. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International conference on machine learning*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Yu, F.; Zhu, Y.; Liu, Q.; Wu, S.; Wang, L.; and Tan, T. 2020. TAGNN: Target Attentive Graph Neural Networks for Session-based Recommendation. In *Proceedings of the 43th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM.

Yu, H.-F.; Hsieh, C.-J.; Si, S.; and Dhillon, I. S. 2014. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems* 41(3): 793–819.

Zhang, M.; and Chen, Y. 2018. Link Prediction Based on Graph Neural Networks. In *Neural Information Processing Systems*.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020. Deep Graph Contrastive Representation Learning. *arXiv preprint arXiv:2006.04131* .