Multi-dimensional Graph Convolutional Networks

Yao Ma [*] Suhang Wang [†] Charu C. Aggarwal[‡] Dawei Yin[§] Jiliang Tang[*]

Abstract

Convolutional neural networks (CNNs) leverage the great power in representation learning on regular grid data such as image and video. Recently, increasing attention has been paid on generalizing CNNs to graph or network data which is highly irregular. Some focus on graph-level representation learning while others aim to learn node-level representations. These methods have been shown to boost the performance of many graph-level tasks such as graph classification and node-level tasks such as node classification. Most of these methods have been designed for single-dimensional graphs where a pair of nodes can only be connected by one type of relation. However, many real-world graphs have multiple types of relations and they can be naturally modeled as multi-dimensional graphs with each type of relation as a dimension. Multi-dimensional graphs bring about richer interactions between dimensions, which poses tremendous challenges to the graph convolutional neural networks designed for single-dimensional graphs. In this paper, we study the problem of graph convolutional networks for multi-dimensional graphs and propose a multi-dimensional convolutional neural network model mGCN aiming to capture rich information in learning node-level representations for multi-dimensional graphs. Comprehensive experiments on real-world multi-dimensional graphs demonstrate the effectiveness of the proposed framework.

1 introduction

Convolutional Neural Networks (CNNs) [22] have been proven to bring breakthrough improvements on various tasks on regular grid data such as image, video and speech [21, 16, 18, 17]. However, graph data such as social networks and gene data are highly irregular. The main issue is that image data have an extremely high level of spatial locality, which might not be the case in graphs data. Recent efforts [5, 15, 10, 24, 8, 20, 13, 32] have been made to generalize convolutional neural networks to irregular graph data. Some [5, 15, 10, 24, 8] focus on graph-level representation learning, while others [20, 13, 32] target on node-level representation learning. These methods have been shown to advance many graph-level tasks such as graph classification [8] and node-level tasks such as node classification [20, 13, 32] and link prediction [32]. In this work, we focus on node-level representation learning with graph covolutional neural networks. The majority of graph convolutional networks have been developed for single dimensional graphs where only one type of relation can exist between a pair of nodes as shown in Figure 1(a). However, in many real-world graphs, multiple relations can exist between pairs of nodes simultaneously as shown in Figure 1(b). For example, in video sharing site like YouTube, users can interact with each other via "subscription" relation as well as various social interaction such as "sharing" and "commenting" [34]. These complex graphs with multiple relations can be naturally treated as multi-dimensional graphs by viewing each type of relation as a dimension as shown in Figure 1(c). Each relation forms a unique graph structure, while the graphs formed by different relations are not independent to each other [3]. Since relations in different dimensions can affect each other, nodes not only can interact with other nodes in each dimension but also can interact with their own "copies" in different dimensions [3]. The multiple relations in multi-dimensional graphs bring about additional complexity. It poses tremendous challenges to existing graph convolutional networks, which have been designed for single dimensional graphs. Thus, dedicated efforts are desired to develop graph convolutional networks for multi-dimensional graphs.

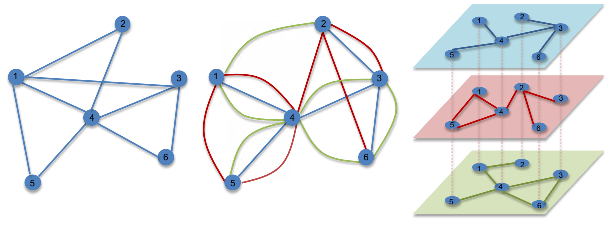

(a) Single-dimensional graph (b) Graph with multiple relations (c) Multi-dimensional graph

Figure 1: Single-dimensional graph and multi-dimensional graph.

In this work, we study the problem of graph convolutoinal networks for multi-dimensional graphs. In essence, we aim to tackle the following challenges: 1)

[*]Michigan State University, {mayao4,tangjili}@msu.edu
[†]Pennsylvania State University, szw494@psu.edu
[‡]IBM T. J. Watson Research Center, charu@us.ibm.com
[§]JD.com, yindawei@acm.org

how to model the node interaction in each dimension while considering the interactions across dimensions; and 2) how to combine the information for multi-dimensional graph convoluitonal networks. These two challenges are addressed by the proposed framework mGCN. Our major contributions can be summarized as follows: 1) We introduce a principled approach to capture the interactions within- and across-dimensions; 2) We propose a multi-dimensional graph convolutional network framework mGCN, which can model rich information in multi-dimensional graphs coherently for representation learning; and 3) We conduct comprehensive experiments on real-world multi-dimensional graphs to demonstrate the effectiveness of the proposed framework.

2 The Proposed Framework

Before we give details about the proposed framework, we first introduce some notations and definitions we will use in the remaining of this work.

A multi-dimensional graph consists of a set of N nodes $\mathcal{V} = \{v_1, \dots, v_N\}$ and D sets of edges $\{\mathcal{E}_1, \dots, \mathcal{E}_D\}$, which describe the interaction between nodes in the corresponding D dimensions. These D types of interactions can be also described by D adjacency matrices $\mathbf{A}_1, \dots, \mathbf{A}_D$. For a given dimension d, its adjacency matrix $\mathbf{A}_d \in \mathbb{R}^{N \times N}$ describes its connection in this dimension. Let $\mathbf{A}_d[i, j]$ denote the element in i-th row and j-th column of $\mathbf{A}_d$. Then $\mathbf{A}_d[i, j] = 1$ when there is a link between node v_i and v_j in dimension d, otherwise 0.

The graph convolutional network (GCN) [20] has been designed for single dimensional graphs. It aims to improve the quality of the representations by aggregating information from neighbor nodes. For a given node, its neighbors are those nodes that are directly connected to it. In other words, traditional GCN utilizes the interactions between nodes and their neighbors to learn node representations. However, in a multi-dimensional graph, all the dimensions share the same set of nodes, while nodes interact with each other differently in different dimensions. Different interactions between nodes form varied network structures in different dimensions as shown in Figure 2. Thus, for a given node, it is likely that it has different neighbors in different dimensions. These different neighbors are specific to each dimension and we call them within-dimension neighbors for each dimension. Furthermore, the same node in different dimensions is inherently related. Thus, for one node in a given dimension, we also need to consider the across-dimension interactions with its own copies in the other dimensions. In this work, we define these copies of a node in the other dimensions as across-dimension neighbors of the given node in the given dimension. More specifically, for a node in a given dimension, there are within-dimension and across-dimension interactions. These two types of interactions lead to two different types of neighbors – within-dimension and across-dimension neighbors. For a node v_i in a given dimension d, the within-dimension neighbors consist of all the nodes that are connected to it in dimension d (or for all v_j where $\mathbf{A}_d[i, j] = 1$). For a node v_i in a given dimension d, the across-dimension neighbors consist of its own copies in the other dimensions. For example, in Figure 2, the within-dimension neighbors for node 4 in the "red" dimension are nodes 1, 2 and 5 in the "red" dimension, while its across-dimension neighbors are node 4 in the "blue" dimension and node 4 in the "green" dimension.

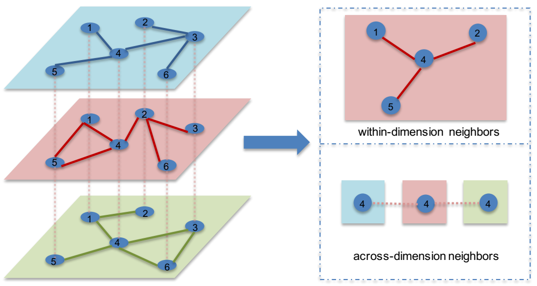


Figure 2: For node 4 in the "red" dimension, the within-dimension neighbors are nodes 1, 2 and 5 in the "red" dimension, while its across-dimension neighbors are node 4 in the "blue" dimension and node 4 in the "green" dimension.

To capture the within-dimension and across-dimension information in each dimension and the general information in the entire multi-dimensional graph, we introduce two representations for each node – the dimension-specific representations to denote the within-dimension and across-dimension information in each dimension and the general representation to denote the general information in the entire multi-dimensional graph. We first detail these two types of representations, then introduce model components to capture within-dimension and across-dimension information and finally discuss the proposed framework.

2.1 General and dimension-specific representations The dimension-specific representations are corresponding to each dimension, while the general representation is supposed to capture the information from all the dimensions. In this section, we introduce the relations between the general representation and dimension-specific representations. More specifically, we describe the procedure of transforming general representation to dimension-specific representations and the procedure of combining dimension-specific representations to form general representation. The procedure not only can help us understand better about the relations between dimension-specific and general representations but also can help us reduce the representation parameters.

Let $\mathbf{h}_i \in \mathbb{R}^{l\times 1}$ and $\mathbf{h}_{i,d} \in \mathbb{R}^{q\times 1}, d = 1,\dots,D$ denote the general representation and dimension-specific representations for node v_i, respectively. To get the dimension-specific representations $\mathbf{h}_{i,d}$ from the general representation $\mathbf{h}_i$, we project the general representation to each dimension. More specifically, we introduce a project matrix $\mathbf{W}_d \in \mathbb{R}^{q\times l}$ for each dimension d with non-linear activation functions to obtain dimension-specific representations from general representations as follows

$$\mathbf{h}_{i,d} = \text{act}(\mathbf{W}_d \cdot \mathbf{h}_i) \tag{2.1}$$

where $\text{act}(\cdot)$ is an element-wise non-linear activation function. Modeling the dimension-specific representations as Eq. (2.1) can naturally impose the dimension-specific representations of each dimensions to share some information, as they are all generated from the same general representation. Furthermore, we can use the project matrix $\mathbf{W}_d$ to understand the relations among dimensions. The transformation in Eq. (2.1) can be also written in the matrix form for all nodes as follows

$$\mathbf{H}_d = \text{act}(\mathbf{W}_d \cdot \mathbf{H}) \tag{2.2}$$

where $\mathbf{H}_d \in \mathbb{R}^{q\times N}$ is the matrix containing the dimension-specific representation for all nodes in dimension d, with column i the representation $\mathbf{h}_{i,d}$ for node v_i. Similarly, $\mathbf{H} \in \mathbb{R}^{l\times N}$ is the matrix containing the general representations for all nodes.

To get the general representation from the dimension-specific representations, we need to integrate the dimension-specific representations. We propose to use a feed forward neural network to perform the combination. We concatenate the D dimension-specific representations as the input of the feed forward network. More specifically, the combination procedure can be represented as

$$\mathbf{h}_i = \text{act}(\mathbf{W} \cdot \text{Concat}_{d=1}^{D}\mathbf{h}_{i,d}). \tag{2.3}$$

where $\mathbf{W} \in \mathbb{R}^{l\times D\cdot q}$, Concat() is the function to concatenate the dimension-specific representations and $\text{act}(\cdot)$ is an element-wise non-linear activation function. Similarly, the combination Eq. (2.3) can also be represented in the matrix form as:

$$\mathbf{H} = \text{act}(\mathbf{W} \cdot \text{Concat}_{d=1}^{D}\mathbf{H}_d). \tag{2.4}$$

Similar to traditional GCN, we can have multiple layers in the mGCN. However, to simplify the illustration of the relations of general and dimension-specific representations, we ignore the index of the layer on all parameters in this subsection. For example, the general representation $\mathbf{h}_i$ in the Eq. (2.1) and Eq. (2.3) can be from different layers. We will add the layer index when we introduce the details of the proposed framework.

2.2 Modeling the within-dimension interactions In this subsection, we introduce the modeling of the within-dimension interaction. For a node v_i, in a given dimension d, to capture the within dimension interaction, we perform the single dimensional GCN [20] to dimension d based on the within-dimension neighbors. More specifically, it can be represented as

$$\mathbf{h}_{w\,i,d} = \sum_{v_i \in N_d(v_i)} \hat{\mathbf{A}}_d[i,j] \cdot \mathbf{h}_{j,d} \tag{2.5}$$

where $N_d(v_i)$ is the set of within-dimension neighbors of node v_i in dimension d, $\hat{\mathbf{A}}_d = \mathbf{D}_d^{-1}(\mathbf{A}_d + \mathbf{I})$ is the row normalized adjacency matrix with self-loop. $\mathbf{D}_d$ is a diagonal matrix with $\mathbf{D}_d[i,i]$ the summation of the i-th row of $\hat{\mathbf{A}} + \mathbf{I}$. $\hat{\mathbf{A}}[i,j]$ is the element of $\hat{\mathbf{A}}$ in i-th row and j-th column. Note that we also include the node itself in its within dimension neighbors $N_d(v_i)$. Aggregating information from within-dimension neighbors in Eq. (2.5) can also be represented in a matrix form as:

$$\mathbf{H}_{w\,d} = \mathbf{H}_d \cdot \hat{\mathbf{A}}_d \tag{2.6}$$

2.3 Modeling the across-dimension interactions To model the across-dimension interactions, we perform a similar aggregation as we do for the within dimension interaction but on the across-dimension neighbors. To perform the across-dimension aggregation, we take a weighted average over the dimension-specific representations of node v_i:

$$\mathbf{h}_{i,d} = \sum_{g=1,\dots,D} b_{g,d} \cdot \mathbf{h}_{i,g} \tag{2.7}$$

where $\sum\limits_{g=1,\dots,D} b_{g,d} = 1$. The weight $b_{g,d}$ models the the importance of dimension g to dimension d. Dimensions do not always affect each other equally and it is likely that there are some dimensions that are more similar than others. Naturally, the dimension-specific representation from a more similar dimension should contribute more in the across-dimension aggregation step. However, this kind of correlation information between dimensions is not always explicitly available. Hence, it is difficult for us to get $b_{g,d}$ beforehand. It is desired that these importance scores between dimensions can be learned during the aggregation steps. Recall that dimension-specific representations are obtained by projecting the general representations. The projection matrices are supposed to contain some descriptive information of the dimensions. For example, if two dimensions are highly similar, the two projection matrices should also be highly related. Thus, we introduce an attention mechanism to learn these scores based on these projection matrices. The importance of a dimension g to a dimension d can be learned from the following function:

$$p_{g,d} = \mathrm{att}(\mathbf{W}_g, \mathbf{W}_d) \tag{2.8}$$

where $\mathrm{att}(\cdot,\cdot)$ is some attention function. The inputs of this attention function are the two projection matrices for the given two dimensions. In this work, we use a bilinear function to model the attention function:

$$p_{g,d} = \mathrm{tr}(\mathbf{W}_g{}^T \mathbf{M} \mathbf{W}_d) \tag{2.9}$$

where $tr()$ is the trace of a matrix and $\mathbf{M}$ is the parameters to be learned in the bilinear function. We further apply a softmax function to normalize the importance score as:

$$b_{g,d} = \frac{\exp(p_{g,d})}{\sum\limits_{g=1}^{D} \exp(p_{g,d})} \tag{2.10}$$

The across-dimension aggregation can also be represented in the matrix form as:

$$\mathbf{H}_d = \sum_{g=1,\ldots,D} b_{g,d} \cdot \mathbf{H}_g \tag{2.11}$$

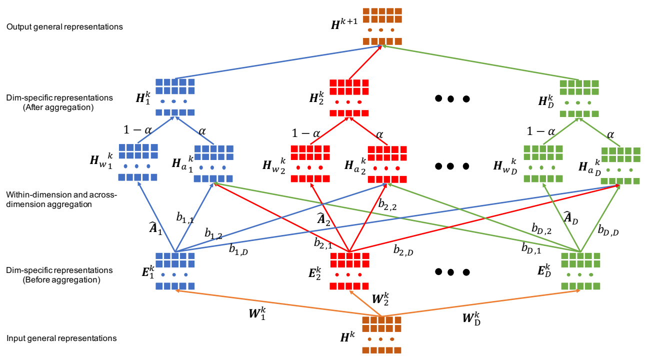


Figure 3: The k-th layer of mGCN framework

2.4 Multi-dimensional Graph Convolutional Networks In this subsection, we introduce the framework of the multi-dimensional GCN, mGCN. Figure 3 shows the k-th layer of mGCN. The input of this layer is the general representations $\mathbf{H}^k$ generated in the $(k-1)$-th layer. The general representations $\mathbf{H}^k$ are then transformed to D dimension-specific representations according to Eq. (2.2)

$$\mathbf{E}_d^k = \mathrm{act}(\mathbf{W}_d^k \cdot \mathbf{H}^k),\ d = 1,\ldots,D. \tag{2.12}$$

Here we use $\mathbf{E}_d^k$ to denote the dimension-specific representations before aggregation to differentiate from the dimension-specific representation after aggregation $\mathbf{H}_d^k$. After the procedures in Eq. (2.12), we then proceed to the within- and across-dimension aggregation procedures as shown in Figure 3 according to Eq. (2.6) and Eq. (2.11).

$$\mathbf{H}_{w}{}_d^k = \mathbf{E}_d^k \cdot \hat{\mathbf{A}}_d,\ d = 1,\ldots,D. \tag{2.13}$$

$$\mathbf{H}_{a}{}_d^k = \sum_{g=1,\ldots,D} b_{g,d} \cdot \mathbf{E}_g^k,\ d = 1,\ldots,D. \tag{2.14}$$

We then combine the results of the within- and across-dimension aggregations as follows

$$\mathbf{H}_d^k = (1-\alpha)\cdot \mathbf{H}_{w}{}_d^k + \alpha \cdot \mathbf{H}_{a}{}_d^k,\ d = 1,\ldots,D, \tag{2.15}$$

where α is the hyper-parameter to control the importance between the two components. The combination generates the dimension-specific representations after aggregation, which are denoted as $\mathbf{H}_d^k,\ d = 1,\ldots,D$. Finally, we combine these dimension-specific representations to get new general representations $\mathbf{H}^{k+1}$ according to Eq. (2.4):

$$\mathbf{H}^{k+1} = \mathrm{act}(\mathbf{W^k} \cdot \mathrm{Concat}_{d=1}^{D} \mathbf{H^k}_d). \tag{2.16}$$

We introduce the k-th layer of the multi-dimensional GCN and the output is the new general representations $\mathbf{H}^{k+1}$, which can serve as the input of the $(k+1)$-th layer. In each layer, we use the output of the previous layer as the input. To initialize the procedure, the input general representations $\mathbf{H}^0$ is needed. These initial general representations could be features associated with the nodes, representations learned by some network embedding methods, or even randomly initialized representations. Let the input be $\mathbf{X}$, then, we initialize $\mathbf{H}^0 = \mathbf{X}$. The output of the multi-dimensional GCN is the general representations $\mathbf{H^K}$ formed in the $(K-1)$-th layer. For convenience, we denote $\mathbf{Z} := \mathbf{H}^K$.

The parameters in the model include the projection matrices $\mathbf{W^k_d},\ d = 1,\ldots,D$, $\mathbf{W}^k$ of the fully connected combination layer and $\mathbf{M}^k$ of the attention function for $k = 0,\ldots,K-1$. To train the model, different loss functions can be designed. For example, we could use supervised information from the node labels. In this work, we design an unsupervised loss function using the linkage information. More specifically, we model the probability that a link existing between node v_i and node v_j in dimension d as

$$p(1|v_i, v_j, d) = \sigma((\mathbf{W}_d^{K+1}\cdot \mathbf{z}_i)^T(\mathbf{W}_d^{K+1}\cdot \mathbf{z}_j)); \tag{2.17}$$

where $\mathbf{W}_d^{K+1}$ is the projection matrix for dimension d, $\mathbf{z}_i$ is the i-th column of $\mathbf{Z}$ and $\sigma(\cdot)$ is the sigmoid function $\sigma(x) = 1/(1+\exp(-x))$. The probability that there is no link between node v_i and node v_j in dimension d can be modeled as

$$\begin{aligned} p(0|v_i, v_j, d) &= 1 - p(1|v_i, v_j, d) \\ &= 1 - \sigma((\mathbf{W}_d^{K+1}\cdot \mathbf{z}_i)^T(\mathbf{W}_d^{K+1}\cdot \mathbf{z}_j)) \\ &= \sigma(-(\mathbf{W}_d^{K+1}\cdot \mathbf{z}_i)^T(\mathbf{W}_d^{K+1}\cdot \mathbf{z}_j)) \end{aligned} \tag{2.18}$$

We can now model the negative logarithm likelihood as

$$L = -\log\left(\prod_{(v_i,v_j,d)\in\mathcal{T}_p} p(1|v_i,v_j,d)\cdot\prod_{(v_i,v_j,d)\in\mathcal{T}_n} p(0|v_i,v_j,d)\right)$$

(2.19)

$$= -\sum_{(v_i,v_j,d)\in\mathcal{T}_p}\log p(1|v_i,v_j,d) - \sum_{(v_i,v_j,d)\in\mathcal{T}_n}\log p(0|v_i,v_j,d),$$

where $\mathcal{T}_p$ is the set of positive samples, which consists of all the existing links in the multi-dimensional graphs. These links can be denoted as triplets (v_i, v_j, d), which means node v_i and node v_j are connected in dimension d. $\mathcal{T}_n$ is the set of negative samples, which consists of all the triplets (v_i, v_j, d) with no link between v_i and v_j in dimension d.

2.5 An Optimization Method Real-world graphs such as social networks or computer networks are usually very sparse [1]. The number of existing links compared to the number of missing links is small and usually it can be linearly bounded by the number of nodes [1]. This means the size of $\mathcal{T}_n$ is large. It is computationally expensive to consider all the pairs in the loss Eq. (2.19). To solve this issue, we use the negative sampling approach proposed in [28]. For each $(v_i, v_j, d) \in \mathcal{T}_p$, we fix v_i and d and randomly sample n nodes that are not connected to node v_i in dimension d. These n samples are put into the set of the negative samples. In this way, the size of $\mathcal{T}_n$ is only n times as large as the size of $\mathcal{T}_p$.

For large graphs with millions of nodes, it is prohibited to perform the within-dimension aggregation in Eq. (2.6) for all nodes at the same time. Furthermore, if we adopt a mini-batch procedure to optimize the loss function, it is not necessary to calculate the representations for all the nodes during each mini-batch. Only those nodes that are within K-hops of the nodes in the mini-batch training samples are involved. Therefore, in practice we only calculate the representations for those involved nodes. However, if we use all the neighbors for given nodes (k-hop neighbors, for $k = 1, \ldots, K$) in the within-dimension aggregation step, the number of nodes involved can still get very large, in the worst case, it can still involve all the nodes in the graph. Hence, we decide to sample s neighbors for a given node v_i from its within-dimension neighbors when performing the within-dimension aggregation as similar in [13]. We adopt a mini-batch ADAM [19] to optimize the framework.

3 Experiments

In this section, we validate the effectiveness of the proposed framework by conducting the link prediction and node classification tasks on two real-world multi-dimensional graphs. We first introduce the two multi-dimensional graphs we use in this paper. Then, we describe the link prediction and node classification tasks with discussions of the experimental results.

Table 1: Statistics of datasets

	DBLP	Epinions
number of nodes	138,072	15,108
number of edges	2,015,650	485,154
number of dimensions	20	5
number of labels	10	15

3.1 Datasets In this subsection, we introduce the two datasets we use in this paper. Some important statistics are shown in Table 1. Detailed descriptions of the two datasets are as follows:

1) **DBLP:** DBLP[1] is a computer science bibliography website. It holds bibliographic information on major computer science journals and proceedings. We collect all publication records from major computer science journals and conferences during 1998-2017. We then treat the co-authorship in each year as different relations and form a co-authorship multi-dimensional graph. Each paper belongs to a high-level category such as "Theoretical Computer Science". We assign the category which most of the author's papers belong to as his/her label. 2) **Epinions** Epinions[2] is a general review site, where users can write reviews for products and rate helpfulness for reviews written by other users. Users in this site can also form trust and distrust relations. We form a 5-dimensional graph based on 5 different relations between users: 1) co-review: two users review some common products; 2) helpfulness-rating: a user rates the reviews written by the other user; 3) co-rating: two users rate some common reviews; 4) trust relation between users; and 5) distrust relation between users. Each product belongs to a category. We assign the category which most of the products reviewed by the user belong to as his/her label.

3.2 Comparison algorithms Our method learns representations for each node in the multi-dimensional graph. So we compare our method with representative single dimensional and multi-dimensional representation learning algorithms. To apply single-dimensional algorithms, we aggregate multi-dimensional graph into a single dimensional graph. Note that we can also apply single-dimensional algorithms on each dimension and then aggregate the representations. We do not include this strategy in this work as it is likely to suffer from the data sparsity problem. Next, we describe these

[1]http://dblp.uni-trier.de/

[2]http://www.epinions.com/

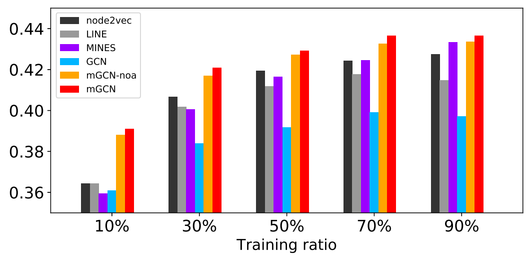
methods as below: 1) **Non-negative matrix factorization (NMF)** [25]. We apply NMF to the adjacency matrix of the aggregated single dimensional graph and use the factorized matrix as the representations; 2) **LINE** [33] is a recent proposed network embedding method designed for single dimensional graph. We aggregate the multi-dimensional graph as a single dimension graph and apply LINE on it. We use the default setting of LINE while setting the number of training samples to 800 millions; 3) **node2vec** [12] is a state-of-the-art network embedding method designed for single-dimensional graph. We apply node2vec on the aggregated single-dimensional graph. We follow the default setting of node2vec and conduct a grid search for q and p as in [12]; 4)**MINES** [27] is a recent proposed network embedding method designed for multi-dimensional network. We apply MINES to the multi-dimensional graph to learn the node representations; 5) **GCN** is a variant of the traditional GCN [20]. The difference is that we adopt the same loss as in mGCN instead of the original semi-supervised loss. Additionally, we also sample neighbors instead of using all the neighbors. We apply it on the single dimensional graph aggregated from the multi-dimensional graph; and 6) **mGCN-noa** is a variant of our method mGCN, where the weights $b_{g,d}$ in the across-dimension aggregation Eq. (2.11) are all $\frac{1}{D}$.

3.3 Node Classification In the node classification task, we try to predict labels for unlabeled nodes. In the experiments setting, we hide the labels for a fraction of nodes and use the label information of the labeled nodes and the graph structure to perform the node classification.

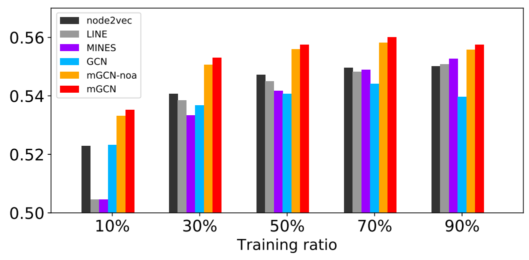
As in [29], we try different fractions of nodes to hide their labels, the remaining nodes with labels are treated as training samples to train a classifier. In this paper, we set the ratio of training samples to $10\% - 90\%$ with a step-size of 20%. We use the node representations as the input features of nodes and train a logistic regression classifier. The nodes with label hided are treated as the testing set. For each of the setting, we try 10 different splits of the training and testing and report the average performance of the 10 experiments. The metrics we use to measure the performance are F_1-macro and F_1-micro score as in [12, 29]. For all the methods, we set the length of the representation to 64 for fair comparison. For node classification task, we use the final general representations $\mathbf{Z}$ for our mGCN and mGCN-noa. We use the representation generated by LINE as input for Epinions dataset for GCN, mGCN and mGCN-noa. For DBLP, we use the representation learned by node2vec as the input for these methods. We set the value of α to 0.5, the number of negative samples n to 2, the number of sample neighbors s to 10 and the number of layers K to 1. More layers can be stacked and we leave this as a possible future direction.

3.3.1 Experiments Results The results of node classification for Epinions and DBLP dataset are shown in Figure 4 and Figure 5, respectively. Note the representations learned by NMF only achieve F_1-macro 0.2380, F_1-micro 0.4012 on the Epinions dataset and F_1-macro 0.2380, F_1-micro 0.3580 on the DBLP dataset both under the 90% training setting. We do not include the performance of NMF in the figures as it is not comparable with other methods. We can make the following observations from these results: 1) Our method mGCN outperforms all the baselines under all the settings on both datasets, which shows the effectiveness of our method; 2) The performance of mGCN is better than GCN, which suggests that utilizing the multi-dimensional relations in the multi-dimensional graph is necessary and our proposed method mGCN can facilitate them well to help learn better representations; 3) mGCN is better than MINES, which shows the effectiveness of mGCN to capture the within- and across-dimension information in a better way; 4) mGCN is better than mGCN-noa, which indicates the effectiveness of the attention mechanism in the across-dimension aggregation step.

3.4 Link prediction In the link prediction task, we try to predict whether a non-existing link will emerge in the future based on the current graph. In the traditional setting of link prediction task for single dimensional graph, a fraction of links are removed from the graph and used as the ground truth and then we try to predict their existence using the remaining graph. In the multi-dimensional graph setting, we perform link prediction in different dimensions, separately. When performing link prediction in dimension d, we first remove a fraction of links from dimension d. Then, for each removed link, if the two nodes connected by this link are connected in the other dimensions, we also remove those links in the other dimensions. After removing these links, we use the remained graph to learn the representations for all the methods. We then formulate the link prediction task as a binary classification problem as in [12] using the combination of the features of node pairs as the input features. Different combination operations can be used to get the feature for a node pair from the features of the two nodes. In this paper, we use the element-wise multiplication, as it achieves the best performance among all the operations in [12]. When a pair of nodes are connected by a link in dimension d, this pair of nodes is labeled as 1, otherwise 0. To form

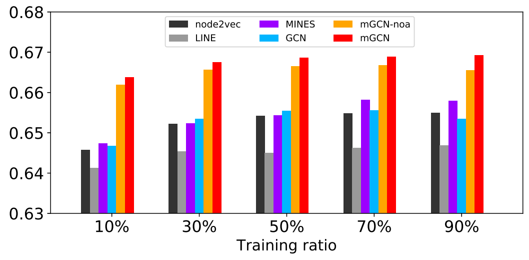


(a) Epinions: F_1-macro

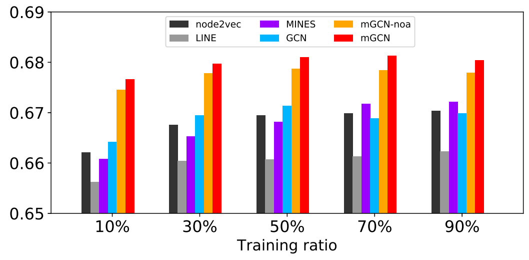


(b) Epinions: F_1-micro

Figure 4: Performance Comparison of Node classification on Epinions dataset



(a) DBLP: F_1-macro



(b) DBLP: F_1-micro

Figure 5: Performance Comparison of Node classification on DBLP dataset

the training set of the binary classifier, we add all the links in dimension d in the remaining graph as positive samples and randomly generate equal number of non-connected node pairs as negative samples. We form the testing set in a similar way, using the removed links as positive samples and randomly generated equal number of negative samples. For the DBLP data, we only perform link prediction task on the last dimension (Year 2017), as it is not reasonable to use future information to predict past links. We conduct link prediction under two different settings, one with 50% links removed and the other one with 70% links removed. For the Epinions dataset, we perform link prediction tasks in each of the 5 dimension with 20% of the links removed. We use AUC score as the metric to measure the performance of link prediction as in [12].

For the experiments on both dataset, we set the length of the representations to 64 for fair comparison. For mGCN and mGCN-noa, we use the $\mathbf{W}_d^{K+1} \cdot \mathbf{Z}$ as the representations when conducting link prediction on dimension d. As similar in the node classification task, we use the representations learned by LINE and node2vec as input for Epinions and DBLP dataset respectively. We set the value of α to 0.5, the number of negative samples n to 2, the number of sample neighbors s to 10 and the number of layers K to 1.

3.4.1 Experiments Results The results for link prediction on the Epinions dataset are shown in Table 2. We conduct link prediction experiments in each dimension of the Epinions data the set. The AUC of each dimension is reported in Table 2, where dim 0-4 denote the co-review dimension, the helpfulness-rating dimension, the co-rating dimension, the trust dimension and the distrust dimension, respectively. The average performance over all dimensions is reported in the last column. We can make the following observations from this table: 1) The link prediction performance on different dimensions varies a lot, which indicates that the network structure are indeed different in each dimension; 2) The performance of the methods designed for single dimensional graph (LINE, node2vec, etal) is worse than MINEs, mGCN-noa and mGCN. This suggests that simply ignoring the different types of the relations and combining the multi-dimensional graph as single dimensional graph may cause loss of information. mGCN performs better than GCN, which further indicates that the mGCN model effectively captures the unique information from multi-dimensional graph. 3) On average, mGCN outperforms mGCN-noa. This suggests that our attention mechanism can capture the relations between dimensions well.

The link prediction performance on DBLP dataset are shown in Figure 6. We make similar observations as those on the Epinions dataset.

4 Related work

Learning appropriate representations for nodes in graph is essential for many graph related machine learning and

Table 2: Performance comparison of link prediction in terms of AUC on Epinions dataset

method	dim 0	dim 1	dim 2	dim 3	dim 4	average
NMF	0.9463	0.8092	0.9381	0.8909	0.9066	0.8982
LINE	0.8612	0.7747	0.8100	0.8088	0.8888	0.8287
node2vec	0.8713	0.7866	0.8773	0.8121	0.8732	0.8441
MINES	0.9621	0.8268	0.9572	0.8036	0.8711	0.8842
GCN	0.9080	0.7512	0.8226	0.8194	0.9002	0.8403
mGCN-noa	0.9349	0.8031	0.9125	0.8642	0.9654	0.8960
mGCN	0.9434	0.8214	0.9273	0.8795	0.9648	0.9072

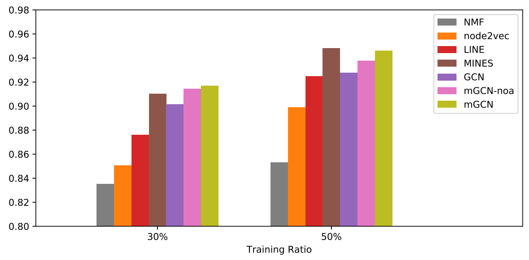


Figure 6: Performance comparison of link prediction in terms of AUC on the DBLP dataset

data mining tasks. Early methods such as Isomap [35] and Laplacian Eigenmap [2] try to perform dimension reduction on the graph. These methods usually involve computational expensive eigen-decomposition and thus are not scalable when the graph is large. Recent methods inspired by word embedding learning such as DeepWalk [29], LINE [33] and node2vec [12] can scale to large graph with millions of nodes. Recent efforts [23, 30] have been made to connect these methods with matrix factorization. There are also works [36, 7, 37, 38] trying to use deep learning techniques to learn node representations. Two recent surveys provide comprehensive overviews on graph representation learning algorithms [6, 14]. Recently, there are also some network embedding methods [27, 31] designed for multi-dimensional graphs.

Convolutional Neural Networks (CNNs) [22] breakthroughly improve the performance in images, videos related task [21, 17]. This shows its great power to learn good representations on regular grid data. However, graph or network data are highly irregular. Efforts have been made to generalize CNNs to graphs [5, 15, 10, 24, 8, 20, 13, 32]. Some of these methods [5, 15, 10, 24, 8] focus on generalizing CNNs for graph level-representation learning. The other methods [20, 13, 32, 9, 11, 26] works on learning node representations using convolutional neural networks. Most of the aforementioned GCN methods are designed for single dimensional graphs. Some recent surveys on this topic can be found in [40, 39]. However, many real-world graphs are multi-dimensional with multiple types of relations. Some examples and basic properties of multi-dimensional graph can be found in [4, 3]. In this work, we propose to study graph convolutional neural network for multi-dimensional graphs.

5 Conclusions and Future Work

In this paper, we develop a novel graph convolutional network for multi-dimensional graphs. We propose to use dimension-specific representations to capture the information for node in each dimension and general representations to capture the information for node over the entire graph. Particularly, we propose to capture the information from both within-dimension and across-dimension interactions when modeling dimension-specific representations in each dimension. We then use a fully connected layer to combine these dimension-specific representations to the general representations. We conduct comprehensive experiments on two real-world multi-dimensional networks. The experimental results demonstrate the effectiveness of the proposed framework.

In this work, we take a weighted average to combine the representations from the within-dimension aggregation and across-dimension aggregation. More advanced combination method such as feed-forward neural networks can be tried. Many real-world graphs or networks are naturally evolving. Thus it would be an interesting topic to consider the temporal information when modeling graph convolutional networks.

Acknoledgements

This work is supported by the National Science Foundation (NSF) under grant number IIS-1714741, IIS-1715940 and CNS-1815636.

References

[1] A.-L. Barabási and M. Pósfai, *Network science*, Cambridge University Press, 2016.

[2] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in NIPS, 2002, pp. 585–591.

[3] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, and D. Pedreschi, *Multidimensional networks: foundations of structural analysis*, World Wide Web, 16 (2013), pp. 567–593.

[4] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin, *The structure and dynamics of multilayer networks*, Physics Reports, 544 (2014), pp. 1–122.

[5] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral networks and locally connected networks on graphs*, arXiv preprint arXiv:1312.6203, (2013).

[6] H. Cai, V. W. Zheng, and K. C.-C. Chang, *A comprehensive survey of graph embedding: Prob-*

lems, techniques and applications, arXiv preprint arXiv:1709.07604, (2017).

[7] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, *Heterogeneous network embedding via deep architectures*, in SIGKDD, ACM, 2015, pp. 119–128.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, in NIPS, 2016, pp. 3844–3852.

[9] T. Derr, Y. Ma, and J. Tang, *Signed graph convolutional networks*, in ICDM, IEEE, 2018, pp. 929–934.

[10] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, *Convolutional networks on graphs for learning molecular fingerprints*, in NIPS, 2015, pp. 2224–2232.

[11] H. Gao, Z. Wang, and S. Ji, *Large-scale learnable graph convolutional networks*, in SIGKDD, ACM, 2018, pp. 1416–1424.

[12] A. Grover and J. Leskovec, *node2vec: Scalable feature learning for networks*, in SIGKDD, ACM, 2016, pp. 855–864.

[13] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, in NIPS, 2017, pp. 1025–1035.

[14] W. L. Hamilton, R. Ying, and J. Leskovec, *Representation learning on graphs: Methods and applications*, arXiv preprint arXiv:1709.05584, (2017).

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

[16] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, *A convolutional neural network for modelling sentences*, arXiv preprint arXiv:1404.2188, (2014).

[17] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, *Large-scale video classification with convolutional neural networks*, in CVPR, 2014, pp. 1725–1732.

[18] Y. Kim, *Convolutional neural networks for sentence classification*, arXiv preprint arXiv:1408.5882, (2014).

[19] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[20] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907, (2016).

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in NIPS, 2012, pp. 1097–1105.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[23] O. Levy and Y. Goldberg, *Neural word embedding as implicit matrix factorization*, in NIPS, 2014, pp. 2177–2185.

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, *Gated graph sequence neural networks*, arXiv preprint arXiv:1511.05493, (2015).

[25] C.-J. Lin, *Projected gradient methods for nonnegative matrix factorization*, Neural Computation, 19 (2007), pp. 2756–2779.

[26] Y. Ma, Z. Guo, Z. Ren, E. Zhao, J. Tang, and D. Yin, *Dynamic graph neural networks*, arXiv preprint arXiv:1810.10627, (2018).

[27] Y. Ma, Z. Ren, Z. Jiang, J. Tang, and D. Yin, *Multi-dimensional network embedding with hierarchical structure*, in WSDM, ACM, 2018, pp. 387–395.

[28] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, in NIPS, 2013, pp. 3111–3119.

[29] B. Perozzi, R. Al-Rfou, and S. Skiena, *Deepwalk: Online learning of social representations*, in SIGKDD, ACM, 2014, pp. 701–710.

[30] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, *Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec*, in WSDM, ACM, 2018, pp. 459–467.

[31] M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han, *An attention-based collaboration framework for multi-view network representation learning*, in CIKM, ACM, 2017, pp. 1767–1776.

[32] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, *Modeling relational data with graph convolutional networks*, arXiv preprint arXiv:1703.06103, (2017).

[33] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, *Line: Large-scale information network embedding*, in WWW, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[34] L. Tang, X. Wang, and H. Liu, *Community detection in multi-dimensional networks*, tech. report, Arizona State University, 2010.

[35] J. B. Tenenbaum, V. De Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, science, 290 (2000), pp. 2319–2323.

[36] D. Wang, P. Cui, and W. Zhu, *Structural deep network embedding*, in SIGKDD, ACM, 2016, pp. 1225–1234.

[37] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu, *Signed network embedding in social media*, in Proceedings of the 2017 SIAM international conference on data mining, SIAM, 2017, pp. 327–335.

[38] S. Wang, J. Tang, C. Aggarwal, and H. Liu, *Linked document embedding for classification*, in CIKM, 2017.

[39] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, *A comprehensive survey on graph neural networks*, arXiv preprint arXiv:1901.00596, (2019).

[40] Z. Zhang, P. Cui, and W. Zhu, *Deep learning on graphs: A survey*, arXiv preprint arXiv:1812.04202, (2018).