# Co-Regularized Deep Multi-Network Embedding

Jingchao Ni[1], Shiyu Chang[2], Xiao Liu[3], Wei Cheng[4], Haifeng Chen[4], Dongkuan Xu[1], Xiang Zhang[1]
[1]College of Information Sciences and Technology, Pennsylvania State University, [2]IBM T. J. Watson Research Center
[3]Department of Biomedical Engineering, Pennsylvania State University, [4]NEC Laboratories America
[1]{jzn47, dux19, xzhang}@ist.psu.edu, [2]shiyu.chang@ibm.com
[3]xxl213@engr.psu.edu, [4]{weicheng, haifeng}@nec-labs.com

## ABSTRACT

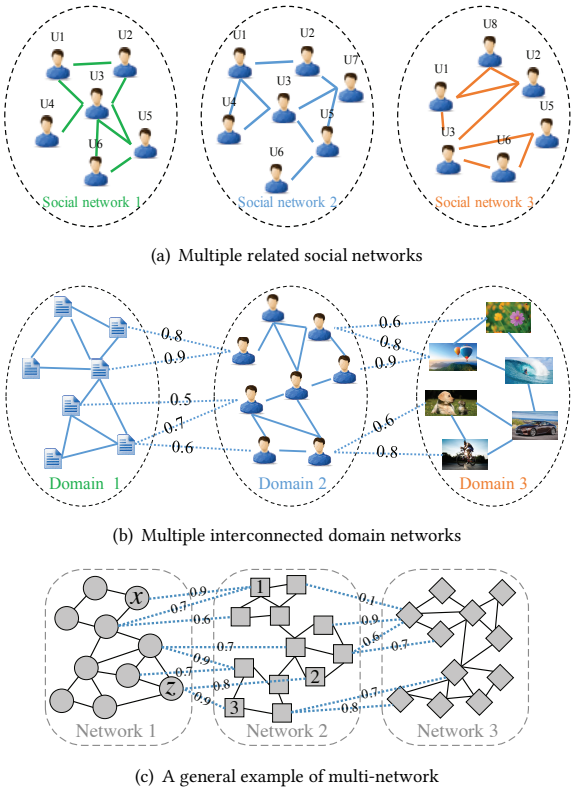
Network embedding aims to learn a low-dimensional vector representation for each node in the social and information networks, with the constraint to preserve network structures. Most existing methods focus on single network embedding, ignoring the relationship between multiple networks. In many real-world applications, however, multiple networks may contain complementary information, which can lead to further refined node embeddings. Thus, in this paper, we propose a novel multi-network embedding method, DMNE. DMNE is flexible. It allows different networks to have different sizes, to be (un)weighted and (un)directed. It leverages multiple networks via cross-network relationships between nodes in different networks, which may form many-to-many node mappings, and be associated with weights. To model the non-linearity of the network data, we develop DMNE to have a new deep learning architecture, which coordinates multiple neural networks (one for each input network data) with a co-regularized loss function. With multiple layers of non-linear mappings, DMNE progressively transforms each input network to a highly non-linear latent space, and in the meantime, adapts different spaces to each other through a co-regularized learning schema. Extensive experimental results on real-life datasets demonstrate the effectiveness of our method.

## KEYWORDS

Multi-network; Network embedding; Representation learning

## 1 INTRODUCTION

Networks (or graphs) are pervasive in real-life applications. The rapid growth of information has generated a large volume of network data, such as social networks [22], document citation networks [19], and biological networks [13]. Network data are characterized by the complex dependencies between nodes. To analyze network data, one fundamental problem is to resolve the dependencies and learn low-dimensional vector representation for each node, such that the network structure is preserved in the learned vector space [29]. By doing so, network analysis such as node classification [21], node clustering [31] and link prediction [9] can then be readily performed in vector space by using the vast off-the-shelf machine learning algorithms. Usually, learning network representation is also known as network embedding [29]. The low-dimensional vectors to be learned are called node embeddings.




(a) Multiple related social networks

(b) Multiple interconnected domain networks

(c) A general example of multi-network

**Figure 1: Examples of multi-network. In (b) and (c), the dotted lines represent cross-network relationships. The value on each dotted line indicates the weight of the relationship.**

To tackle this problem, many methods have been proposed recently. For example, Laplacian Eigenmaps [1] solves leading eigenvectors of the Laplacian matrix of a graph as node embeddings, which can preserve the direct relationship between nodes in the graph. DeepWalk [29] uses random walks to extract local communities of each node in a network, which are preserved via a word embedding technique called skip-gram [23]. LINE [34] optimizes a KL-divergence function to learn embeddings, which can preserve both 1st- and 2nd-order proximities between nodes in a network. node2vec [9] further extends DeepWalk by adopting a biased random walk, so as to preserve both breadth first search (BFS) and depth first search (DFS) based neighborhoods of each node.

Despite the encouraging progress, the focus of most existing methods is single network embedding. In many emerging applications, however, related multiple networks are common to observe.


*WWW 2018, April 23–27, 2018, Lyon, France*

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186113

For example, nowadays users are often involved in more than one online social networks. Thus multiple social networks from Facebook, Twitter, LinedIn, etc. are related one another by those common users. Fig. 1(a) illustrates this example, where the three social networks may be collected from different social platforms. Some users in them are common (e.g., U1, U2, U3) while others may be unique (e.g., U7, U8). In this scenario, there is a one-to-one correspondence between users from different networks, which represents the identity of users across different social networks, e.g., U1 represents the same user who appears in all networks.

In some applications, nodes in different networks may represent different entities. Fig. 1(b) shows a different type of multi-network. In this case, multiple networks may contain nodes of different domains, such as text documents, users and color images. Here, text-text links are formed by the hyper-links between different Web documents. Users are involved in a social network. Images co-occurring within the same Web page provide explicit linkages between them. Moreover, users may interact with texts and images by responding to them or clicking on them, which forms the cross-network relationships that interconnect multiple networks, as represented by the dotted lines in Fig. 1(b). In this scenario, the mappings between nodes in different networks may form a many-to-many correspondence, instead of one-to-one, e.g., one user may interact with multiple Web documents, and vice versa.

Similar examples can also be observed in the information networks of many other fields. In bioinformatics, one important problem is to classify genetic diseases in a disease similarity network [8]. In this network, each node is a disease, and each edge depicts the phenotype similarity between two diseases [39]. To reflect the molecular foundation, we may explore the disease similarity network together with its underlying protein-protein interaction (PPI) network, where disease nodes and protein nodes are related one another via the known disease-protein associations [13]. The disease-protein associations form a many-to-many mapping. This is because multiple proteins can function synergistically to cause a single disease and a single protein can participate in the formation of multiple diseases.

In practice, because of measurement errors and data access limitations, a single network may contain dummy nodes and false links (i.e., noise), and missing nodes and missing links (i.e., incompleteness). Such defects can largely reduce the learned embedding quality. Whereas, the false or missing information in one network may be corrected in other related networks. Therefore, a promising approach to overcome the limitation of single network embedding is to exploit the compatible and complementary information in multiple networks to refine the embedding quality. Based on this intuition and the prevalence of multi-network data on the web, in this paper, we propose to investigate network embedding in the context of multiple networks.

Fig. 1(c) shows a general example of multi-network that covers the instances in both Fig. 1(a) and 1(b). There are several characteristics that should be noticed. First, different networks may be about either the same or different sets of nodes, thus may have different sizes. Second, a node in one network may be associated with multiple nodes in another, making the cross-network relationship a many-to-many mapping, which is a generalization of one-to-one mapping. Third, each cross-network relationship may be associated with a weight, which is a generalization of a binary relationship. Fourth, some nodes in one network may not have corresponding node in another, making the cross-network relationship an incomplete, partial mapping.

In the previous social and biological applications, all these characteristics can be observed. For instance, in Fig. 1(b), a user-image relationship may be weighted by the frequency of the interaction. In the biological example, domain experts may specify weights on the disease-protein relationships using their prior knowledge, so as to mark the correlation levels of some disease-protein pairs.

To be practically useful, all the characteristics in Fig. 1(c) should be properly handled, so that refined embeddings can be learned from both instances of multi-network in Fig. 1(a) and 1(b). So far, few methods have been developed to multi-network embedding problem. Until very recently, there is one method proposed on embedding multi-view network [30]. However, this method can only be applied to a special case of Fig. 1(c) when different networks are about the same set of nodes, with a strict one-to-one cross-network relationship. Hence, a more flexible method is in demand.

Another vital challenge in our problem is how to model the non-linearity of network data. As shown by [40], the underlying structures of many real-life information networks are highly non-linear, which cannot be fully captured by linear projection approaches such as SVD. To be effective, a preferable approach thus should offer the ability to catch the non-linearity of the data.

Motivated by the powerful representation learning ability of deep learning and its intrinsic non-linearity [15], we propose a novel algorithm, Deep Multi-Network Embedding (DMNE), based on a deep learning model. DMNE coordinates multiple neural networks (one for each input network data) with a co-regularized loss function to manipulate cross-network relationships, which can be many-to-many, weighted and incomplete. With multiple layers of non-linear functions, DMNE can progressively map each input network into a highly non-linear latent space. In the meantime, different latent spaces are adaptive to each other via a joint learning procedure. Our contributions are summarized as follows.

- We propose to investigate the problem of multi-network embedding in a general context, where multiple networks can be (un)weighted and (un)directed, the cross-network relationships can be many-to-many, weighted and incomplete. This problem finds wide applications in real practice.
- We propose the first deep learning based multi-network embedding algorithm DMNE, which not only allows the general multi-network in Fig. 1(c), but also can capture the non-linear structures in multi-network data.
- We design an effective optimization algorithm, which has a solid theoretical guarantee on convergence, and can be easily parallelized to scale to large datasets.
- We perform comprehensive experiments on real-life datasets including document networks, social networks, biological networks. The results demonstrate DMNE outperforms recent network embedding methods by a large margin.

The rest of the paper is organized as follows. Sec. 2 gives the problem definition. Sec. 3 introduces DMNE method. Sec. 4 introduces the optimization solution. Sec. 5 discusses the experimental results. Sec. 6 reviews the related work. Sec. 7 concludes the paper.

**Table 1: Summary of notation**

| Symbol | Meaning |
|---|---|
| $g$ | The number of networks |
| $n_i$ | The number of nodes in the $i$-th network |
| $d_i$ | The dimensionality of the $i$-th embedding space |
| $L_i$ | The number of neural net. layers for the $i$-th net. data |
| $\mathbf{G}^{(i)}$ | The adjacency matrix of the $i$-th network |
| $\mathbf{A}^{(i)}$ | The structural context matrix of the $i$-th network |
| $\mathbf{S}^{(ij)}$ | The relationship matrix between nodes in $\mathbf{G}^{(i)}$ and $\mathbf{G}^{(j)}$ |
| $\tilde{\mathbf{S}}^{(ij)}$ | The row-normalized version of $\mathbf{S}^{(ij)}$ |
| $\mathbf{U}^{(i)}$ | The embedding matrix of the $i$-th network |
| $\{\mathbf{W}_l^{(i)}\}_{l=1}^{L_i}$ | The weight matrices for the $i$-th network data |
| $\{\mathbf{b}_l^{(i)}\}_{l=1}^{L_i}$ | The bias vectors for the $i$-th network data |
| $\boldsymbol{\theta}^{(i)}$ | The model parameters $\boldsymbol{\theta}^{(i)} = \{\mathbf{W}_l^{(i)}, \mathbf{b}_l^{(i)}\}_{l=1}^{L_i}$ |
| $\mathcal{I}$ | The set of cross-network relationships |

## 2 PROBLEM AND BACKGROUND

Suppose we have $g$ networks, each is represented by an adjacency matrix $\mathbf{G}^{(i)} \in \mathbb{R}_+^{n_i \times n_i}$ $(1 \le i \le g)$, where $n_i$ denotes the number of nodes in the $i$-th network. In this paper, our analysis applies to any (un)directed and (un)weighted network. Thus $\mathbf{G}^{(i)}$ can be either symmetric or asymmetric, and either continued or binary, with $\mathbf{G}_{xy}^{(i)}$ indicating the edge weight between nodes $x$ and $y$ in $\mathbf{G}^{(i)}$. We denote the set of pairwise cross-network relationships by $\mathcal{I} = \{(i,j)\}$. For example, $\mathcal{I} = \{(1,2),(2,3)\}$ contains two cross-network relationships: the relationships between networks $\mathbf{G}^{(1)}$ and $\mathbf{G}^{(2)}$, and the relationships between $\mathbf{G}^{(2)}$ and $\mathbf{G}^{(3)}$. Each pair $(i,j)$ is coupled with a matrix $\mathbf{S}^{(ij)} \in \mathbb{R}_+^{n_i \times n_j}$, with $\mathbf{S}_{xy}^{(ij)}$ measuring the weight between node $x$ in $\mathbf{G}^{(i)}$ and node $y$ in $\mathbf{G}^{(j)}$. For clarity, important notations are summarized in Table 1.

After embedding, each node $x$ in network $\mathbf{G}^{(i)}$ $(1 \le i \le g)$ will obtain a low dimensional vector, i.e., the embedding vector. We use $\mathbf{h}_x^{(i)} \in \mathbb{R}^{1 \times d_i}$ to represent this vector, where $d_i$ is the dimensionality of the embedding space of network $\mathbf{G}^{(i)}$, which can be different for different $i$'s. In this work, our goal is to learn embedding vectors of all nodes in all networks, based on the structures of $\{\mathbf{G}^{(i)}\}_{i=1}^{g}$ and the regularizing constraints implicitly represented by the cross-network relationships in $\mathcal{I}$.

### 2.1 Structural Context Extraction

Real-life networks are often so sparse that only using the very limited observed links is insufficient to capture reasonable relationships between nodes. In addition to direct neighbors, nodes in a network also have dependencies with indirect neighbors. Therefore, existing embedding methods usually employ certain sampling strategies to extract local community information as the *structural context* of each node [9, 29]. Among different strategies, random walk is the most widely used because of its intrinsic effectiveness in local clustering [32]. In this paper, we follow existing approaches [3] and use random walk with restart (RWR) to obtain structural context of each node.

Given a network $\mathbf{G}$ of $n$ nodes, a starting node $x$, we introduce a $k$-step RWR vector $\mathbf{p}^{(k)} \in \mathbb{R}_+^{1 \times n}$, with $\mathbf{p}_y^{(k)}$ indicates the probability of visiting node $y$ after $k$ step transitions from $x$. Let $\mathbf{p}^{(0)}$ be the initial vector with $\mathbf{p}_x^{(0)} = 1$ and all other entries being 0, then a RWR process is defined as [38]

$$\mathbf{p}^{(k)} = c\mathbf{p}^{(k-1)}[(\mathbf{D})^{-1}\mathbf{G}] + (1-c)\mathbf{p}^{(0)} \tag{1}$$

where $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{xx} = \sum_{y=1}^{n} \mathbf{G}_{xy}$, and $1-c$ $(0 < c < 1)$ represents the probability that the random walker will restart from node $x$.

To capture local information, we follow [3] and use several short-step RWR vectors to define the structural context vector $\mathbf{a}$ ($\mathbf{a} \in \mathbb{R}^{1 \times n}$) for each node $x$.

$$\mathbf{a} = \sum_{k=1}^{K} \mathbf{p}^{(k)} \tag{2}$$

where $K$ is a small integer indicating the number of considered steps. In practice, $K = 3$ is usually sufficient. Thus the computation of this step is fast.

After obtaining vectors $\mathbf{a}$ for all nodes, a *structural context matrix* $\mathbf{A}$ can be formed with each row as $\mathbf{a}$ for one node. An entry $\mathbf{A}_{xy}$ indicates to what degree a node $y$ will appear in the local community of node $x$. Using the same approach, we can obtain structural context matrices $\{\mathbf{A}^{(1)}, ..., \mathbf{A}^{(g)}\}$ for all networks $\{\mathbf{G}^{(1)}, ..., \mathbf{G}^{(g)}\}$. In the following, $\{\mathbf{A}^{(i)}\}_{i=1}^{g}$ will be used as the input of our multi-network embedding method.

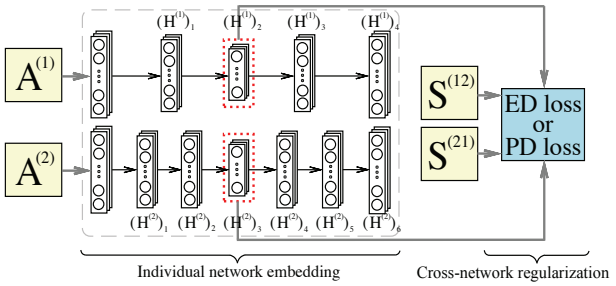



**Figure 2: The architecture of DMNE for two networks as an example. $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ are structural contexts obtained from the two networks. $\mathbf{S}^{(12)}$ and $\mathbf{S}^{(21)}$ are weighted cross-network relationships. $(\mathbf{H}^{(i)})_l$ contains vectors that represent the latent representations of all nodes at the $l$-th layer of the $i$-th network. ED and PD are two kinds of loss functions for cross-network regularization. In DMNE, different input networks can have different numbers of neural network layers. Here the two input networks have 5 and 7 layers, respectively.**

## 3 DEEP MULTI-NETWORK EMBEDDING

In this section, we introduce DMNE, a deep model that partially incorporates AutoEncoder [10]. Fig. 2 illustrates the key architecture of DMNE for two networks as an example. Here, each input network (i.e., $\mathbf{A}^{(i)}$) will be fed into a neural network to minimize the reconstruction error. Meanwhile, the bottleneck layer representations of all input networks, which are the desired node embeddings, are adapted to each other via a co-regularization function (i.e., ED loss or PD loss) that leverages the weighted cross-network relationships (i.e., $\mathbf{S}^{(ij)}$). Next, to be self-contained, we first review the

key idea of deep AutoEncoder. Then, we propose the two kinds of loss functions to leverage cross-network relationships. Finally, we discuss a speedup strategy and the joint optimization problem.

## 3.1 Single-Network Embedding

For the $i$-th input network $\mathbf{A}^{(i)}$, the neural network consists of $L_i + 1$ layers for performing $L_i$ non-linear transformations. The first $L_i/2$ hidden layers are *encoders* to learn a set of compact representations (i.e., dimension reduction) and the last $L_i/2$ layers are *decoders* to progressively reconstruct the input. For ease of presentation, we first provide the following definitions. Let $(\mathbf{h}_x^{(i)})_0 = \mathbf{A}_{x*}^{(i)} \in \mathbb{R}^{1\times n_i}$ (i.e., the $x$-th row of $\mathbf{A}^{(i)}$) be an input vector of node $x$ to the first layer and

$$(\mathbf{h}_x^{(i)})_l = \sigma((\mathbf{h}_x^{(i)})_{l-1}\mathbf{W}_l^{(i)} + \mathbf{b}_l^{(i)}) \in \mathbb{R}^{1\times k_l} \tag{3}$$

be the output of the $l$-th layer, where $l = 1, ..., L_i$, and $\sigma(\cdot)$ is a non-linear activation function[1]. $k_l$ denotes the dimensionality of the output at the $l$-th layer, $\mathbf{W}_l^{(i)} \in \mathbb{R}^{k_{l-1}\times k_l}$ and $\mathbf{b}_l \in \mathbb{R}^{1\times k_l}$ denote the weight and bias associated with the $l$-th layer, respectively. Thus, given $\mathbf{A}_{x*}^{(i)}$ as the input, $(\mathbf{h}_x^{(i)})_{L_i}$ (i.e., the last layer representation) is the reconstruction of $\mathbf{A}_{x*}^{(i)}$, while $(\mathbf{h}_x^{(i)})_{L_i/2}$ is the desired embedding of node $x$. Let $\hat{\mathbf{A}}^{(i)}$ be an $n_i$ by $n_i$ matrix with $x$-th row as $(\mathbf{h}_x^{(i)})_{L_i}$, the goal of AutoEncoder is to minimize the reconstruction error

$$\min_{\boldsymbol{\theta}^{(i)}} \mathscr{L}_{ae}^{(i)} = \|\mathbf{A}^{(i)} - \hat{\mathbf{A}}^{(i)}\|_F^2 + \lambda \sum_{l=1}^{L_i} \|\mathbf{W}_l^{(i)}\|_F^2 \tag{4}$$

where $\boldsymbol{\theta}^{(i)} = \{\mathbf{W}_l^{(i)}, \mathbf{b}_l^{(i)}\}_{l=1}^{L_i}$ are model parameters. The last $\ell_2$ norm terms are used to prevent overfitting, and $\lambda$ is the regularization parameter.

Since each row of the input $\mathbf{A}^{(i)}$ in Eq. (4) encodes the local community of a node, minimizing the reconstruction error will enforce the learned embeddings to preserve the local neighborhood of each node. This is desirable because neighboring nodes in the network should also be close to each other in the embedding space.

## 3.2 Cross-Network Regularization

To incorporate the cross-network relationship, the key idea is to add pairwise regularizers to the single-network embedding objective function. We develop two kinds of loss functions to regularize the cross-network embeddings. Both are designed to penalize the embedding inconsistency with the given cross-network relationships. The first loss function, ED loss, considers a simple case when the dimensionality of embeddings are the same in different networks. The second loss function, PD loss, is more flexible and has no such constraint.

*3.2.1 Embedding Disagreement (ED) Loss Function.* We start with a simple case when the dimensionality of embeddings are the same for different networks, i.e., $d_1 = ... = d_g = d$. For simplicity, in the following, we use $\mathbf{h}_x^{(i)}$ to represent the embedding of a node $x$ in network $\mathbf{G}^{(i)}$, i.e., $\mathbf{h}_x^{(i)} = (\mathbf{h}_x^{(i)})_{L_i/2}$. Intuitively, if a node $x$ in network $\mathbf{G}^{(i)}$ is mapped to a node $y$ in network $\mathbf{G}^{(j)}$, then the embeddings $\mathbf{h}_x^{(i)}$ and $\mathbf{h}_y^{(j)}$ should be similar. Now we generalize the relationship to many-to-many. We use $\mathcal{N}^{(i\to j)}(x)$ to denote the set of nodes in $\mathbf{G}^{(j)}$ that are mapped to $x$ in $\mathbf{G}^{(i)}$ with positive weights. To penalize the inconsistency of cross-network embeddings, we propose the following loss function.

$$\mathscr{L}_x = \|\mathbf{h}_x^{(i)} - \mathbf{h}_x^{(i\to j)}\|_F^2 \tag{5}$$

where

$$\mathbf{h}_x^{(i\to j)} = \frac{1}{\sum_{y\in\mathcal{N}^{(i\to j)}(x)} \mathbf{S}_{xy}^{(ij)}} \sum_{y\in\mathcal{N}^{(i\to j)}(x)} \mathbf{S}_{xy}^{(ij)} \mathbf{h}_y^{(j)} \tag{6}$$

is the weighted mean of the embeddings of nodes in network $\mathbf{G}^{(j)}$ that are mapped to $x$. Recall that $\mathbf{S}_{xy}^{(ij)}$ is the weight on the cross-network relationship between node $x$ in $\mathbf{G}^{(i)}$ and node $y$ in $\mathbf{G}^{(j)}$.

Let $\tilde{\mathbf{S}}^{(ij)}$ be row-normalized $\mathbf{S}^{(ij)}$. That is

$$\tilde{\mathbf{S}}_{xy}^{(ij)} = \frac{\mathbf{S}_{xy}^{(ij)}}{\sum_{z=1}^{n_j} \mathbf{S}_{xz}^{(ij)}} \tag{7}$$

and let $\mathbf{H}^{(i)} = [(\mathbf{h}_1^{(i)})^T, ..., (\mathbf{h}_{n_i}^{(i)})^T]^T \in \mathbb{R}^{n_i\times d_i}$. Then, by summing up Eq. (5) over all nodes in network $\mathbf{G}^{(i)}$, we have the following embedding disagreement (ED) loss function.

$$\mathscr{L}_{ed}^{(ij)} = \|\mathbf{O}^{(ij)}\mathbf{H}^{(i)} - \tilde{\mathbf{S}}^{(ij)}\mathbf{H}^{(j)}\|_F^2 \tag{8}$$

where we introduce a diagonal indicator matrix $\mathbf{O}^{(ij)} \in \{0,1\}^{n_i\times n_i}$, with $\mathbf{O}_{xx}^{(ij)} = 0$ if the $x$-th row of $\tilde{\mathbf{S}}^{(ij)}$ is all-zero; and $\mathbf{O}_{xx}^{(ij)} = 1$ otherwise.

*3.2.2 Proximity Disagreement (PD) Loss Function.* Next, we develop a more flexible loss function. The intuition is based on the following shortcoming of ED loss. In Eq. (6), we observe that $\mathbf{h}_x^{(i\to j)}$ is a weighted mean of the embeddings in $\mathcal{N}^{(i\to j)}(x)$. The ED loss compare $\mathbf{h}_x^{(i)}$ and $\mathbf{h}_x^{(i\to j)}$ directly to make them consistent. This is reasonable when the nodes in $\mathcal{N}^{(i\to j)}(x)$ are close to each other within network $\mathbf{G}^{(j)}$. When nodes in $\mathcal{N}^{(i\to j)}(x)$ are far from each other (e.g., in different communities), their embeddings should be dissimilar to each other within $\mathbf{G}^{(j)}$. However, directly making their mean consistent with $\mathbf{h}_x^{(i)}$ will enforce them to be similar to each other, which is counterintuitive.

To overcome this problem, the key is to avoid direct comparison between $\mathbf{h}_x^{(i)}$ and $\mathbf{h}_x^{(i\to j)}$. Thus, for each pair of nodes $x$ and $z$ in network $\mathbf{G}^{(i)}$, we first measure the proximity between $\mathbf{h}_x^{(i)}$ and $\mathbf{h}_z^{(i)}$, and the proximity between $\mathbf{h}_x^{(i\to j)}$ and $\mathbf{h}_z^{(i\to j)}$. Then, we measure the disagreement between these two proximity values. Taking Fig. 1(c) as an example. Note node $x$ in network 1 is mapped to node *1* in network 2. Node $z$ in network 1 is mapped to nodes {*2, 3*} in network 2. Intuitively, if the proximity between the embedding of node *1* and the mean embedding of nodes {*2, 3*} is small, the proximity between node $x$ and node $z$ should also be small. In this paper, we choose inner product to measure the proximity between two

[1] In this work, we use sigmoid function $\sigma(x) = \frac{1}{1+exp(-x)}$

embeddings, e.g., $\mathbf{h}_x^{(i)}(\mathbf{h}_z^{(i)})^T$. Therefore, the cross-network proximity disagreement (PD) loss function is defined as

$$\begin{aligned}\mathscr{L}_{pd}^{(ij)} &= \sum_{x=1}^{n_i}\sum_{z=1}^{n_i}[\mathbf{h}_x^{(i)}(\mathbf{h}_z^{(i)})^T - \mathbf{h}_x^{(i\to j)}(\mathbf{h}_z^{(i\to j)})^T]^2 \\ &= \|\mathbf{O}^{(ij)}\mathbf{H}^{(i)}(\mathbf{O}^{(ij)}\mathbf{H}^{(i)})^T - \tilde{\mathbf{S}}^{(ij)}\mathbf{H}^{(j)}(\tilde{\mathbf{S}}^{(ij)}\mathbf{H}^{(j)})^T\|_F^2\end{aligned} \tag{9}$$

It is worth to note that in Eq. (9), without direct comparison of embeddings, PD loss allows embeddings in different networks $\mathbf{G}^{(i)}$ to have the different dimensionality $d_i$, which is more flexible than ED loss. Moreover, both ED loss in Eq. (8) and PD loss in Eq. (9) can handle many-to-many, weighted and incomplete cross-network relationships as encoded in $\{\mathbf{S}^{(ij)}\}_{(i,j)\in\mathcal{I}}$.

### 3.3 Joint Optimization with Speedup Strategy

Next, we further develop our model to allow efficient optimization via stochastic gradient descent (SGD). In Eq. (8) and (9), all training samples, i.e., the rows of $\mathbf{H}^{(j)}$, are coupled together through the multiplication $\tilde{\mathbf{S}}^{(ij)}\mathbf{H}^{(j)}$, so SGD cannot be applied in sample-wise. As pointed out by [41], to speedup SGD via parallelization and to save memory, the model should allow samples to be divided by minibatch, which means an objective function should be decomposed into sums over training samples. Therefore, we relax Eq. (8) and (9) by introducing new variables $\{\mathbf{U}^{(i)}\}_{i=1}^g$ to replace $\{\mathbf{H}^{(i)}\}_{i=1}^g$ such that

$$\begin{aligned}\mathscr{L}_{ed}^{(ij)} &= \|\mathbf{O}^{(ij)}\mathbf{U}^{(i)} - \tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)}\|_F^2 \\ \mathscr{L}_{pd}^{(ij)} &= \|\mathbf{O}^{(ij)}\mathbf{U}^{(i)}(\mathbf{O}^{(ij)}\mathbf{U}^{(i)})^T - \tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)}(\tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)})^T\|_F^2\end{aligned} \tag{10}$$

and introduce a regularizer

$$\mathscr{L}_{hu}^{(i)} = \|\mathbf{U}^{(i)} - \mathbf{H}^{(i)}\|_F^2 \tag{11}$$

to require $\mathbf{U}^{(i)}$ to be similar to $\mathbf{H}^{(i)}$.

Now, we can integrate individual network reconstruction in Eq. (4), the loss function in Eq. (10) and the regularizer Eq. (11) into a unified objective function

$$\min_{\{\boldsymbol{\theta}^{(i)},\mathbf{U}^{(i)}\}_{i=1}^g} \mathscr{L} = \sum_{i=1}^g \mathscr{L}_{ae}^{(i)} + \alpha\sum_{(i,j)\in\mathcal{I}}\mathscr{L}_R^{(ij)} + \beta\sum_{i=1}^g\mathscr{L}_{hu}^{(i)} \tag{12}$$

where $\mathscr{L}_R^{(ij)}$ can be either $\mathscr{L}_{ed}^{(ij)}$ or $\mathscr{L}_{pd}^{(ij)}$. $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^g$ are weights and biases (see Eq. (4)). $\alpha, \beta \geq 0$ are trade-off parameters.

Note in Eq. (12), there is no multiplication $\tilde{\mathbf{S}}^{(ij)}\mathbf{H}^{(j)}$, thus Eq. (12) can be decomposed into sums over rows of $\mathbf{H}^{(i)}$, which means $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^g$ can be solved efficiently using SGD via parallelization. To solve $\{\mathbf{U}^{(i)}\}_{i=1}^g$, we also develop efficient iterative algorithm, which will be detailed in next section.

Theoretically, Eq. (11) is the negative log likelihood function of a sampling process $\mathbf{U}_{x*}^{(i)} \sim \mathcal{N}(\mathbf{H}_{x*}^{(i)}, \mathbf{I}_{d_i}/2)$ where $\mathbf{H}_{x*}^{(i)}$ represents the mean of the Gaussian distribution, $\mathbf{I}_{d_i}$ is a $d_i$-by-$d_i$ identity matrix, and $\mathbf{U}_{x*}^{(i)}$ represents the sampled embedding of node $x$ in network $\mathbf{G}^{(i)}$. Therefore, in our method, we use $\mathbf{U}^{(i)}$, instead of $\mathbf{H}^{(i)}$, as the learned embeddings of nodes in $\mathbf{G}^{(i)}$.

## 4 LEARNING ALGORITHM

In this section, we develop an alternating minimization algorithm to optimize $\mathscr{L}$ in Eq. (12). That is, the objective function is alternately minimized w.r.t. one variable while fixing others, until a stationary point is achieved. Next, we provide the solution to $\{\mathbf{U}^{(i)}\}_{i=1}^g$ and $\{\boldsymbol{\theta}\}_{i=1}^g$, respectively.

**Learning $\{\mathbf{U}^{(i)}\}_{i=1}^g$.** Given current $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^g$, we derive a multiplicative updating rule to solve $\{\mathbf{U}^{(i)}\}_{i=1}^g$. The solutions are summarized in the following theorems, which are derived using the Auxiliary Function approach [17]. The detailed proofs of the theorems are omitted for brevity, which can be found in an online Supplementary Material[2].

Theorem 1. *For ED loss, updating $\mathbf{U}^{(i)}$ by Eq. (13) monotonically decreases the objective value in Eq. (12) until convergence.*

$$\mathbf{U}^{(i)} = \mathbf{U}^{(i)} \circ \left(\frac{\sum_{(i,j)\in\mathcal{I}}\boldsymbol{\Theta}^{(ij)} + \sum_{(j,i)\in\mathcal{I}}\boldsymbol{\Lambda}^{(ji)} + \beta\mathbf{H}^{(i)}}{\sum_{(i,j)\in\mathcal{I}}\boldsymbol{\Phi}^{(ij)} + \sum_{(j,i)\in\mathcal{I}}\boldsymbol{\Pi}^{(ji)} + \beta\mathbf{U}^{(i)}}\right)^{\frac{1}{2}} \tag{13}$$

*where*

$$\begin{aligned}\boldsymbol{\Theta}^{(ij)} &= \alpha(\mathbf{O}^{(ij)})^T\tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)},\ \boldsymbol{\Phi}^{(ij)} = \alpha(\mathbf{O}^{(ij)})^T\mathbf{O}^{(ij)}\mathbf{U}^{(i)} \\ \boldsymbol{\Lambda}^{(ji)} &= \alpha(\tilde{\mathbf{S}}^{(ji)})^T\mathbf{O}^{(ji)}\mathbf{U}^{(j)},\ \boldsymbol{\Pi}^{(ji)} = \alpha(\tilde{\mathbf{S}}^{(ji)})^T\tilde{\mathbf{S}}^{(ji)}\mathbf{U}^{(i)}\end{aligned}$$

Theorem 2. *For PD loss, updating $\mathbf{U}^{(i)}$ by Eq. (14) monotonically decreases the objective value in Eq. (12) until convergence.*

$$\mathbf{U}^{(i)} = \mathbf{U}^{(i)} \circ \left(\frac{\sum_{(i,j)\in\mathcal{I}}\hat{\boldsymbol{\Theta}}^{(ij)} + \sum_{(j,i)\in\mathcal{I}}\hat{\boldsymbol{\Lambda}}^{(ji)} + \beta\mathbf{H}^{(i)}}{\sum_{(i,j)\in\mathcal{I}}\hat{\boldsymbol{\Phi}}^{(ij)} + \sum_{(j,i)\in\mathcal{I}}\hat{\boldsymbol{\Pi}}^{(ji)} + \beta\tilde{\mathbf{U}}^{(i)}}\right)^{\frac{1}{4}} \tag{14}$$

*where*

$$\begin{aligned}\hat{\boldsymbol{\Theta}}^{(ij)} &= 2\alpha(\mathbf{O}^{(ij)})^T\tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)}(\tilde{\mathbf{S}}^{(ij)}\mathbf{U}^{(j)})^T\mathbf{O}^{(ij)}\tilde{\mathbf{U}}^{(i)} \\ \hat{\boldsymbol{\Phi}}^{(ij)} &= 2\alpha(\mathbf{O}^{(ij)})^T\mathbf{O}^{(ij)}\mathbf{U}^{(i)}(\mathbf{U}^{(i)})^T(\mathbf{O}^{(ij)})^T\mathbf{O}^{(ij)}\mathbf{U}^{(i)} \\ \hat{\boldsymbol{\Lambda}}^{(ji)} &= 2\alpha(\tilde{\mathbf{S}}^{(ji)})^T\mathbf{O}^{(ji)}\mathbf{U}^{(j)}(\mathbf{O}^{(ji)}\mathbf{U}^{(j)})^T\tilde{\mathbf{S}}^{(ji)}\mathbf{U}^{(i)} \\ \hat{\boldsymbol{\Pi}}^{(ji)} &= 2\alpha(\tilde{\mathbf{S}}^{(ji)})^T\tilde{\mathbf{S}}^{(ji)}\mathbf{U}^{(i)}(\mathbf{U}^{(i)})^T(\tilde{\mathbf{S}}^{(ji)})^T\tilde{\mathbf{S}}^{(ji)}\mathbf{U}^{(i)}\end{aligned}$$

where $\circ$, $\frac{[\cdot]}{[\cdot]}$, $(\cdot)^{\frac{1}{2}}$ and $(\cdot)^{\frac{1}{4}}$ are entry-wise operators.

**Learning $\{\mathbf{W}_l^{(i)}, \mathbf{b}_l^{(i)}\}_{i=1}^g$.** Given current $\{\mathbf{U}^{(i)}\}_{i=1}^g$, we can learn the weight and bias for each layer using the back-propagation (BP) algorithm [10]. Here, the key step is to calculate the gradients of $\mathscr{L}$ in Eq. (12) w.r.t. $\mathbf{W}_l^{(i)}$ and $\mathbf{b}_l^{(i)}$, which are

$$\begin{aligned}\nabla_{\mathbf{W}_l^{(i)}}\mathscr{L} &= (\mathbf{H}_{l-1}^{(i)})^T(\boldsymbol{\delta}_l^{(i)} + \beta\boldsymbol{\Delta}_l^{(i)}) + \lambda\mathbf{W}_l^{(i)} \\ \nabla_{\mathbf{b}_l^{(i)}}\mathscr{L} &= (\mathbf{1}^{(i)})^T(\boldsymbol{\delta}_l^{(i)} + \beta\boldsymbol{\Delta}_l^{(i)})\end{aligned} \tag{15}$$

where

$$\begin{aligned}\boldsymbol{\delta}_l^{(i)} &= \begin{cases}2(\hat{\mathbf{A}}^{(i)} - \mathbf{A}^{(i)}) \circ \sigma'(\mathbf{Z}_l^{(i)}) & l = L_i \\ [\boldsymbol{\delta}_{l+1}^{(i)}(\mathbf{W}_l^{(i)})^T] \circ \sigma'(\mathbf{Z}_l^{(i)}) & \text{otherwise}\end{cases} \\ \boldsymbol{\Delta}_l^{(i)} &= \begin{cases}2(\mathbf{H}^{(i)} - \mathbf{U}^{(i)}) \circ \sigma'(\mathbf{Z}_l^{(i)}) & l = L_i/2 \\ [\boldsymbol{\Delta}_{l+1}^{(i)}(\mathbf{W}_l^{(i)})^T] \circ \sigma'(\mathbf{Z}_l^{(i)}) & l < L_i/2 \\ \mathbf{0} & \text{otherwise}\end{cases}\end{aligned} \tag{16}$$

and $\mathbf{1}^{(i)}$ is a length-$n_i$ column vector with all entries as 1, $\sigma'(\cdot)$ is the derivative of $\sigma(\cdot)$, and $\mathbf{Z}_l^{(i)} = \mathbf{H}_{l-1}^{(i)}\mathbf{W}_l^{(i)} + \mathbf{1}^{(i)}\mathbf{b}_l^{(i)}$.

[2] https://nijingchao.github.io/dmnesup/dmnesup.pdf

**Algorithm 1**: Deep multi-network embedding (DMNE)

**Input**: Networks $\{\mathbf{A}^{(i)}\}_{i=1}^{g}$, cross-network relationships $\{\mathbf{S}^{(ij)}\}_{(i,j)\in I}$, parameters $\alpha$, $\beta$ and $\lambda$
**Output**: $\{\mathbf{U}^{(i)}\}_{i=1}^{g}$ and $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{g}$

```
Pretrain neural network for each A^(i) to obtain initial {θ^(i)}_{i=1}^g and
  {H^(i)}_{i=1}^g, initialize each U^(i) = H^(i);
repeat
    for i ← 1 to g do
        Back-propagation to update θ^(i) based on Eq. (15);
    end
    for i ← 1 to g do
        Forward propagation to obtain H^(i);
        Update U^(i) by Eq. (13) or Eq. (14);
    end
until Convergence
return {U^(i)}_{i=1}^g and {θ^(i)}_{i=1}^g.
```

**Summary.** Alg. 1 summarizes our algorithm, which alternates between the updating of $\{\mathbf{U}^{(i)}\}_{i=1}^{g}$ and $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{g}$. According to Theorem 1 and 2, updating $\mathbf{U}^{(i)}$ monotonically decreases the objective $\mathscr{L}$ in Eq. (12). Using SGD, updating $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{g}$ also decreases the objective value with proper learning rate and momentum [41]. Because Eq. (12) is bounded below by 0, the alternating algorithm will eventually converge.

Since the key difference between standard BP and our algorithm is the updating of $\mathbf{U}^{(i)}$, we analyze the time complexity for Eq. (13) and (14). Let $M$ be the maximal number of cross-network links in any $\mathbf{S}^{(ij)}$, $N$ be the maximal number of nodes in any network, $D$ be the maximal dimensionality of embeddings $d_i$. We can verify that, using sparse matrix multiplication, the complexity of Eq. (13) and (14) are $O((M+N)D)$ and $O(MD+ND^2)$, respectively. In practice, $D$ is a small number, $M$ is often linear in $N$ for sparse networks, thus the actual time complexity can be considered as linear $O(N)$.

## 5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate our method. Specifically, we focus on two widely considered applications: multi-label classification and data visualization.

### 5.1 Datasets

We use four publicly available social/information networks with class labels in our experiments, which are detailed in the following. The statistics of the datasets are summarized in Table 2.

**20-Newsgroup** dataset[3] contains about 20,000 documents of 20 classes. Following [3, 37], we constructed two kinds of networks: **6-NG** and **9-NG**, which are formed by documents of 6 and 9 different classes, respectively. For brevity, we omit the names of selected classes, which are listed in [37]. For 6-NG, we generate 5 networks of different sizes. The first network contains randomly sampled 600 documents (100 from each class). For the remaining four networks, the numbers of sampled documents of each class are {125, 150, 175, 200}, forming networks of sizes {750, 900, 1050, 1200}. Using the same approach, 9-NG contains 5 networks of sizes {900, 1125, 1350, 1575, 1800}. Here, link weight is the cosine similarity between the *tf-idf* vectors of two documents. To reduce noises, we further construct a $k$-nn graph for each network with $k = 5$. The cross-network relationship is calculated by cosine similarity between documents from each pair of networks.

**Disease-protein network (DP-NET)** [13] consists of a disease network and a protein-protein interaction (PPI) network. The disease network has 5,080 nodes (diseases) and 19,729 links. Each link is weighted by the phenotype similarity between two diseases. The PPI network has 8,503 nodes (proteins) and 32,189 links. Each link has a binary weight, where 1 indicates a functional interaction between two proteins. Moreover, the disease network and the PPI network are interconnected by 2,107 disease-protein association relationships. Here, only 675 diseases are labeled, each in one of 18 classes (disease categories).

**DBIS** [35] is a social collaboration dataset, which contains a collaboration network of 12,002 nodes (authors) and 37,587 links. Each link is weighted by the number of co-authored papers. It also has a paper-paper similarity network of 12,533 nodes (papers). Each paper is first represented by a *tf-idf* vector based on its title, then pairwise cosine similarities are calculated using these vectors. To reduce noises, a $k$-nn graph is constructed for the paper network with $k = 5$, resulting in 47,597 weighted links. Additionally, collaboration network and paper network are interconnected by 38,035 author-paper publication relationships. Here, 2,890 authors are labeled, each in one of 4 classes (research areas) [14].

**CiteSeer-M10** [18] is a subset of CiteSeerX data. It has a collaboration network of 3,284 nodes (authors) and 13,781 weighted links, and two paper networks: a citation network and a similarity network. The citation network has 2,035 nodes (papers) and 3,356 binary weighted links. Each link indicate a citation relationship. The similarity network is constructed in the same way as DBIS dataset, with 10,214 nodes (papers) and 39,411 weighted links. The collaboration network and paper citation network are interconnected by 2,634 author-paper relationships, and the number is 7,173 between collaboration network and paper similarity network. The two paper networks are interconnected by 2,021 one-to-one correspondence between papers. Here, each author is labeled in one of 10 classes (research areas).

In the first three datasets, the cross-network relationships are many-to-many. CiteSeer-M10 contains two paper networks with different link types, thus has a mixture of one-to-one and many-to-many cross-network relationships. This represents a mixture of the cases in Fig. 1(a) and 1(b), which cannot be handled by existing embedding methods.

### 5.2 Comparing Methods

Since no existing method can handle the co-regularized multi-network embedding problem, we compare the proposed DMNE with the following eight state-of-the-art network embedding methods:

(1) **Laplacian Eigenmaps (LE)** [1]: it uses leading eigenvectors of the Laplacian matrix of a network as node embeddings.

(2) **Spectral clustering (Spectral)** [31]: it also uses eigenvectors, but differs from LE by using a normalized Laplacian matrix.

[3] http://qwone.com/%7Ejason/20Newsgroups/

**Table 2: Statistics of datasets**

| Dataset | # Networks | # Nodes | # Links | # CrossLinks | LabeledNet. | # LabeledNodes | # Classes |
|---|---|---|---|---|---|---|---|
| 6-NG | 5 | 4,500 | 16,447 | 66,756 | All | 4,500 | 6 |
| 9-NG | 5 | 6,750 | 24,778 | 100,585 | All | 6,750 | 9 |
| DP-NET | 2 | 13,583 | 51,918 | 2,107 | Disease | 675 | 18 |
| DBIS | 2 | 24,535 | 85,184 | 38,035 | Collaboration | 2,890 | 4 |
| CiteSeer-M10 | 3 | 15,533 | 56,548 | 11,828 | Collaboration | 3,284 | 10 |

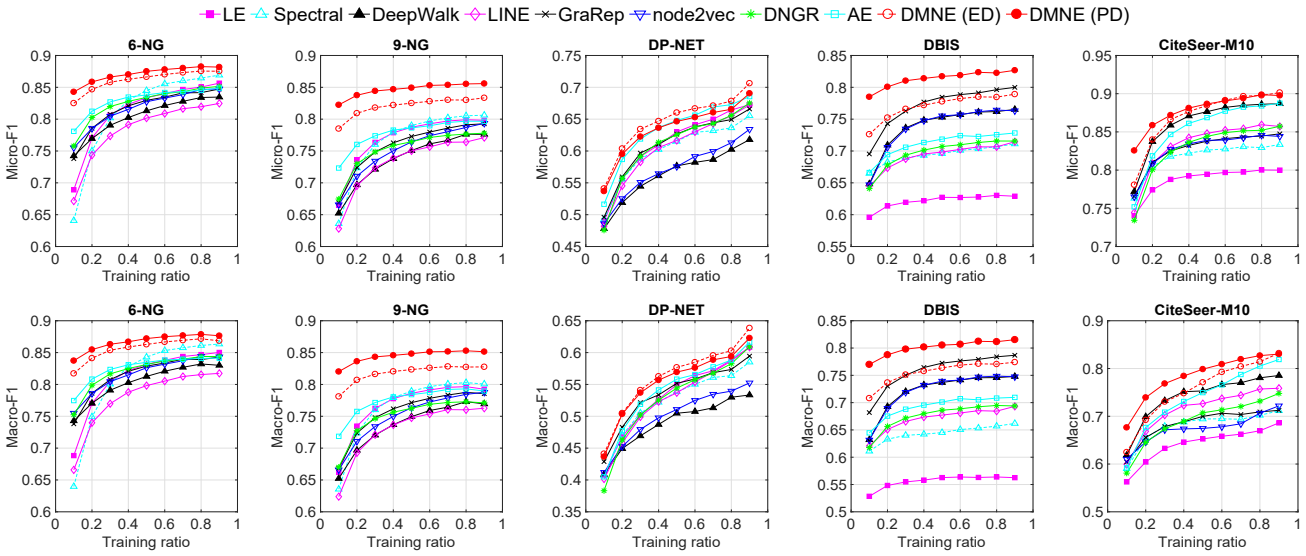



**Figure 3: Multi-label classification results of the compared methods on different datasets.**

(3) **DeepWalk** [29]: it uses truncated random walk and skip-gram to generate node embeddings.

(4) **LINE** [34]: it minimizes a loss function to learn embeddings that preserve both 1st- and 2nd-order proximity between nodes.

(5) **GraRep** [2]: a SVD based embedding method that preserves high-order proximity between nodes.

(6) **node2vec** [9]: it extends DeepWalk by using a biased random walk to generate node embeddings.

(7) **DNGR** [3]: a stacked AutoEncoder based embedding method that uses the structural contexts in Eq. (2) as input, and trains the neural network in layer-wise.

(8) **AutoEncoder (AE)** [10]: it also uses the structural contexts in Eq. (2) as input but trains the neural network as a whole.

There is another AutoEncoder based method SDNE [40] which only preserves 1st- and 2nd-order proximity between nodes. We found by using RWR based structural contexts in Eq. (2), AE outperforms SDNE. Thus we omit SDNE for brevity. Note multi-view network embedding method [30] cannot be applied on these datasets since the cross-network relationships are many-to-many.

The parameters of the compared methods are set as follows. LE and Spectral do not have model parameters. For DeepWalk, LINE, GraRep, and node2vec, we set their parameters the same as the optimal settings in their papers. Specifically, DeepWalk uses *window size* 10, *walk length* 40, *walks per vertex* 80; LINE uses *learning rate* 0.025, *# negative samples* 5, *# total samples* 10 billion. We use its advanced version with both 1st- and 2nd-order of node proximity; GraRep uses *# transition step* 3; node2vec uses *window size* 10, *walk length* 80, *walks per vertex* 10, *return* $p = 1$, *In-out* $q = 1$. For DNGR, AE and DMNE, we follow [3] to set $c = 0.98$ and $K = 3$ in Eq. (2), and set the dimensionality of each layer as below.

| | |
|---|---|
| 6NG, 9NG | $B$-200-100-200-$B$ |
| DP-NET | $B$-500-100-500-$B$ |
| DBIS | $B$-1000-500-100-500-1000-$B$ |
| CiteSeer-M10 | author net., paper citation net.: $B$-500-100-500-$B$<br>paper similarity net.: $B$-1000-500-100-500-1000-$B$ |

For 6-NG, 9-NG, DP-NET, DBIS, the dimensionality are set the same for different networks, due to the relatively small change in the sizes of networks. Here, $B$ represents the number of nodes in each network. For example, $B = 5,080$ for the disease network in DP-NET, and $B = 8,503$ for the PPI network. For all methods, the dimensionality of embeddings are 100. For DMNE, the penalty parameter $\lambda$ is set as $10^{-4}$. The model parameters $\alpha$ and $\beta$ are set as 1. A study about these model parameters will be discussed later.

## 5.3 Multi-Label Classification

First, we compare different methods through a classification task. On each dataset, the embeddings are learned from the full data.

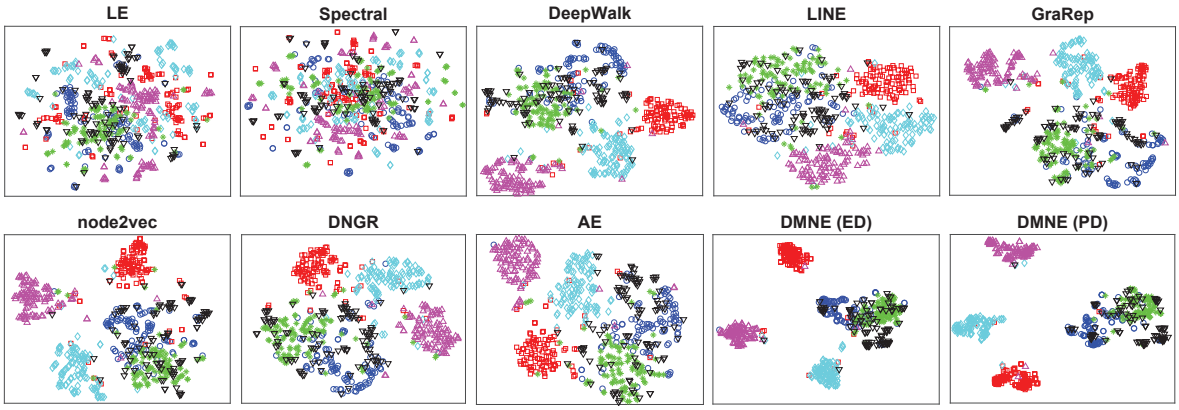



**Figure 4: Visualization results of the compared methods.**

Then, the embeddings of labeled nodes are used as input to the SVM classifier in LIBLINEAR package [7]. When training the classifier, we randomly sample a portion of the labeled nodes as training data and the rest as testing data. The ratio of training data is varied from 10% to 90% for all datasets. The classification accuracy is evaluated using the widely used Macro-F1 and Micro-F1 scores [9, 29]. For each method, the prediction experiment is repeated 100 times and the averaged performance is reported.

Fig. 3 shows the results on all datasets. Considering the available labeled data, on 6-NG and 9-NG, the accuracy is averaged over all networks; on DP-NET, the disease network is evaluated; on DBIS and CiteSeer-M10, their collaboration networks are evaluated.

From the figure, we have the following observations. First, DMNE significantly outperforms all competitors in terms of both metrics. This is because DMNE leverages the complementary information in multiple networks to refine node embeddings, while the baseline methods are subject to the noises and incompleteness in individual networks. Especially, DMNE is much better than other methods when the training ratio is small, e.g., 10%, which means DMNE is more useful in real practice when the available labels are scarce. This advantage comes from the reinforcement learning of DMNE which better uses the available information in multiple networks. On DP-NET, the performance gain of DMNE is relatively small. This is because the number of available disease-protein relationships is small, limiting the cross-network reinforcement of embeddings. We also notice, for DMNE, PD loss is usually better than ED loss. This verifies our early discussions in Sec. 3.2.2 about the superiority of PD loss when handling cross-network relationships.

## 5.4 Visualization

To better understand the difference between the compared methods, we visualize their embeddings using the visualization tool t-SNE [20], which projects the learned embeddings of each method to a 2D space. Fig. 4 shows the results on the first network of 6-NG, which has 600 nodes. The colors (or shapes) represent 6 classes.

From the figure, we can observe the eigenvector based methods LE and Spectral cannot effectively identify different classes. Other baseline methods can detect the classes to varying extents. Both DMNE (ED) and DMNE (PD) perform best as they clearly separate red, cyan and purple classes from each other, with large boundaries. For the other three classes, although all methods have difficulty to separate them, DMNE, especially DMNE (PD), still detects the boundaries among them. These results further demonstrate the better quality of the embeddings learned by DMNE.

**Table 3: Micro-F1 score results on number of layers**

| Dataset | DMNE (ED) | | | DMNE (PD) | | |
|---|---|---|---|---|---|---|
| | 3 layer | 5 layer | 7 layer | 3 layer | 5 layer | 7 layer |
| 6-NG | 0.8091 | 0.8250 | **0.8276** | 0.8328 | **0.8428** | 0.8399 |
| 9-NG | 0.7309 | 0.7850 | **0.7939** | 0.7948 | **0.8224** | 0.8188 |

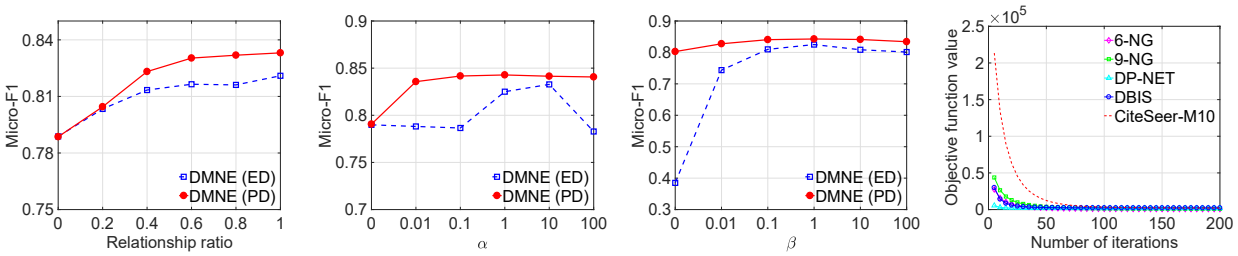
## 5.5 Insights of Effectiveness

To get more insights about DMNE, we perform experiments to evaluate DMNE in detail using 6-NG and 9-NG datasets.

**Varying cross-network relationships.** First, we divide the pairwise cross-network relationships in 6-NG into 5 equal parts. Each time, we add one part (i.e., 20% relationships) into the data and apply DMNE. Fixing the training data ratio at 10%, Fig. 5(a) shows the classification accuracy w.r.t. the ratio of available cross-network relationships. From the figure, both loss functions are effective to enhance the embedding quality as more cross-network relationships are added. Consistent with the results in Fig. 3, PD loss is superior than ED loss when handling cross-network relationships.

**Parameter study.** In our model in Eq. (12), there are two major parameters $\alpha$ and $\beta$. Fig. 5(b) and 5(c) show the classification accuracy on 6-NG by changing one parameter while fixing another as 1, with training data ratio at 10%. As can be seen, both loss functions are stable w.r.t. these parameters, and PD loss is better. For both loss functions, $\alpha$ and $\beta$ are almost best as 1. Thus it is reasonable to set them at 1 in our experiments. Moreover, the non-zero choices of $\alpha$ and $\beta$ demonstrate the importance of the regularization terms in our model $\mathscr{L}$ in Eq. (12).

**Shallow model vs deep model.** To see the effectiveness of DMNE in capturing the non-linear structures of multi-network data, we




(a) Varying cross-net. relationship ratio (b) Varying $\alpha$ (c) Varying $\beta$ (d) Convergence evaluation

**Figure 5: Performance evaluation of DMNE.**

vary the neural network structures by $B$-100-$B$ (3 layer), $B$-200-100-200-$B$ (5 layer) and $B$-200-200-100-200-200-$B$ (7 layer). Table 3 shows the Micro-F1 scores when training data ratio is 10%. For both ED and PD loss functions, deep models (i.e., 5 or 7 layer) are better than shallow models (3 layer), indicating the importance of using deep structures. For PD loss, we also notice it starts to overfit when the model exceeds 5 layers.

## 5.6 Convergence Evaluation

In this section, we study the performance of the proposed DMNE algorithm, in terms of the number of iterations before converging to a local optima. Fig. 5(d) shows the value of the objective function $\mathscr{L}$ in Eq. (12) with respect to the number of iterations on different datasets. From the figure, we observe the objective function value decreases steadily with more iterations. Usually, less than 100 iterations are sufficient for convergence.

# 6 RELATED WORK

To our best knowledge, this is the first work to study deep multi-network embedding problem. Existing network embedding methods are mostly developed on a single network, such as eigenvector based methods [1, 36], skip-gram based methods [9, 29, 34], SVD based methods [2, 27] and AutoEncoder based methods [3, 40]. As discussed before, these single network based methods are subject to noises and incompleteness in individual information network.

Recently, several methods have been proposed to embed an attributed network, in which each node is associated with an attribute vector [11, 12, 28, 42, 43]. Their key idea is to integrate a dimension reduction component of attribute vectors into a network embedding framework to leverage the complementary information in node attributes and network structure. Despite their success in using node attributes to improve performance, these methods never consider multiple networks.

There are also some methods on embedding heterogeneous information network (HIN) [4, 6]. In these methods, an HIN is a special case of our multi-network in Fig. 1(c) in two aspects. First, they ignore the scenario when any two networks are about the same nodes but have different topological structures, i.e., the case in Fig. 1(a). Second, they neglect edge weights, either within-network or cross-network (as shown by their mathematical formations). More specifically, the method in [4] was proposed on an HIN with node attributes. Its prerequisite is that each node must be associated an attribute vector, otherwise it cannot learn node embeddings from network structure only. However, in many applications, we only have link information, which necessitates methods on embedding network structures. The method in [6], on the other hand, requires users to specify meta-paths as its input, which is hard to choose in practice due to the absence of gold standard. This makes it hard to generalize to any applications. In [33], there is a text based HIN embedding method, which, however, is strictly designed for word-word, word-doc and word-label networks. It requires label information to be provided because it is a semi-supervised method. Then, how to generalize it to multiple networks of any shape for unsupervised learning is unknown. Clearly, all these methods have strong limitations, preventing them from being applied to our problem. More importantly, the goal of HIN based methods is to resolve the semantic meanings of different types of links, rather than using the complementary information in multiple networks to refine embedding quality. Therefore, there is a distinct difference between HIN based methods and multi-network based methods.

The multi-view network embedding method [30] may be the most relevant work, but, as discussed before in Sec. 1, it cannot be applied when the cross-network relationship is many-to-many, weighted and incomplete, as shown in Fig. 1(c).

Our method is also inspired by traditional multi-view and multi-graph learning methods [5, 16, 24–26, 41], which aim to integrate multiple data sources in a certain task, such as instance clustering, to obtain performance gain. However, these methods are not designed for multi-network embedding, and none of them uses deep model to exploit the non-linear structures of the network data.

# 7 CONCLUSION

Integrating the rich social and information networks on the web is important to improve the robustness of representation learning methods. In this paper, we propose a flexible co-regularized deep multi-network embedding algorithm DMNE, which is developed on a very practical scenario of multi-network, thus is widely applicable. DMNE manipulates cross-network relationships to reinforce the learning of node embeddings in different networks. Its deep architecture also provides the ability to capture the highly non-linear structures of multiple networks. Extensive experiments on real-world datasets demonstrate the effectiveness of our method.


# ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation grants IIS-1664629 and CAREER.

## REFERENCES


[1] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* 15, 6 (2003), 1373–1396.

[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*. 1145–1152.

[4] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD*. ACM, 119–128.

[5] Wei Cheng, Xiang Zhang, Zhishan Guo, Yubao Wu, Patrick F Sullivan, and Wei Wang. 2013. Flexible and robust co-regularized multi-domain graph clustering. In *KDD*. ACM, 320–328.

[6] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.

[7] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.* 9, Aug (2008), 1871–1874.

[8] Kwang-Il Goh, Michael E Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. 2007. The human disease network. *Proc. Natl. Acad. Sci. U.S.A.* 104, 21 (2007), 8685–8690.

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[10] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.

[11] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*. SIAM, 633–641.

[12] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. ACM, 731–739.

[13] TaeHyun Hwang, Gowtham Atluri, MaoQiang Xie, Sanjoy Dey, Changjin Hong, Vipin Kumar, and Rui Kuang. 2012. Co-clustering phenome–genome for phenotype classification and disease gene discovery. *Nucleic Acids Res.* 40, 19 (2012), e146–e146.

[14] Ming Ji, Jiawei Han, and Marina Danilevsky. 2011. Ranking-based classification of heterogeneous information networks. In *KDD*. ACM, 1298–1306.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[16] Abhishek Kumar, Piyush Rai, and Hal Daume. 2011. Co-regularized multi-view spectral clustering. In *NIPS*. 1413–1421.

[17] Daniel D Lee and H Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. In *NIPS*. 556–562.

[18] Kar Wai Lim and Wray Buntine. 2015. Bibliographic analysis with the citation network topic model. In *ACML*. 142–158.

[19] Kar Wai Lim and Wray Buntine. 2016. Bibliographic analysis on research publications using authors, categorical labels and the citation network. *Machine Learning* 103, 2 (2016), 185–213.

[20] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, Nov (2008), 2579–2605.

[21] Sofus A Macskassy and Foster Provost. 2007. Classification in networked data: A toolkit and a univariate case study. *J. Mach. Learn. Res.* 8, May (2007), 935–983.

[22] Julian McAuley and Jure Leskovec. 2012. Learning to discover social circles in ego networks. In *NIPS*. 539–547.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[24] Jingchao Ni, Wei Cheng, Wei Fan, and Xiang Zhang. 2018. ComClus: A self-grouping framework for multi-network clustering. *IEEE Trans. Knowl. Data Eng.* 30, 3 (2018), 435–448.

[25] Jingchao Ni, Hanghang Tong, Wei Fan, and Xiang Zhang. 2014. Inside the atoms: ranking on a network of networks. In *KDD*. ACM, 1356–1365.

[26] Jingchao Ni, Hanghang Tong, Wei Fan, and Xiang Zhang. 2015. Flexible and robust multi-network clustering. In *KDD*. ACM, 835–844.

[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*. ACM, 1105–1114.

[28] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. In *IJCAI*. 1895–1901.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.

[30] Meng Qu, Jian Tang, Jingbo Shang, Xiang Ren, Ming Zhang, and Jiawei Han. 2017. An Attention-based Collaboration Framework for Multi-View Network Representation Learning. In *CIKM*. ACM, 1767–1776.

[31] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (2000), 888–905.

[32] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*. ACM, 81–90.

[33] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*. ACM, 1165–1174.

[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.

[35] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*. ACM, 990–998.

[36] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323.

[37] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning deep representations for graph clustering. In *AAAI*. 1293–1299.

[38] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. IEEE, 613–622.

[39] Marc A Van Driel, Jorn Bruggeman, Gert Vriend, Han G Brunner, and Jack AM Leunissen. 2006. A text-mining analysis of the human phenome. *Eur. J. Hum. Genet.* 14, 5 (2006), 535.

[40] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. ACM, 1225–1234.

[41] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. 2015. On deep multi-view representation learning. In *ICML*. 1083–1092.

[42] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *IJCAI*. 2111–2117.

[43] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2016. Homophily, structure, and content augmented network representation learning. In *ICDM*. IEEE, 609–618.