# INTEGRATION OF NODE EMBEDDINGS FOR MULTIPLE VERSIONS OF A NETWORK

by

MENGZHEN LI

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Department of Computer and Data Science

CASE WESTERN RESERVE UNIVERSITY

August, 2020

# Integration of Node Embeddings for Multiple Versions of A Network

Case Western Reserve University
Case School of Graduate Studies

We hereby approve the thesis[1] of

**MENGZHEN LI**

candidate for the degree of

**Master of Science**

**Mehmet Koyutürk**

Committee Chair, Advisor
Department of Computer and Data Science

**Erman Ayday**

Committee Member
Department of Computer and Data Science

**Jing Li**

Committee Member
Department of Computer and Data Science

**Date of Defense**

07/16/2020

[1]We certify that written approval has been obtained for any proprietary material contained therein.

# Table of Contents

# List of Tables

## List of Figures

# Acknowledgements

I would like to thank my advisor Mehmet Koyuturk for his support and guidance. I also wish to thank Tyler Cowman, Serhan Yilmaz , and Kaan Yorgancioğlu for giving me valuable advice on my research.

# Integration of Node Embeddings for Multiple Versions of A Network

Abstract

by

MENGZHEN LI

Machine learning applications on large-scale network-structured data commonly encode network information in the form of node embeddings. The general principle of existing algorithms for computing network embeddings is to map the nodes into a low-dimensional space such that the nodes that are “similar" with respect to network topology are also close to each other in the embedding space. Many real-world networks that are used in machine learning have multiple versions that come from different sources, are stored in different databases, or belong to different parties. Due to efficiency or privacy concerns, it may be desirable to compute *consensus embeddings* for the superposed network directly from the node embeddings of individual versions, without explicitly constructing the superposed network. We consider multiple approaches to compute embeddings for the superposed network from the embeddings of individual versions. To systematically assess the quality of the resulting consensus embeddings, we define the notion of *fidelity*. We then test the performance of consensus embeddings on link prediction. Our results show that predictions obtained with consensus embeddings are almost as accurate as those that are obtained with embeddings computed using the superposed network.

# 1 Introduction

Large-scale information networks are becoming ubiquitous in the real world. Mining knowledge from these information networks has become very popular in a broad range of applications. Learning a representation of networks is useful for many network analysis applications[1,2], including social network analysis[3,4] and bioinformatics[5,6].

With the explosion in the scale and scope of machine learning applications, the notion of *node embedding* is introduced to facilitate effective application of machine learning algorithms to large-scale networks. Node embedding aims to map each node in the network to a low dimensional vector representation to extract features that represent the topological characteristics of the network. Many techniques are developed for this purpose[7–10], and these techniques are shown to be effective in addressing problems such as link prediction[11,12] and node classification[11,13].

With the increase in the quantity and variety of network datasets, effective extraction of knowledge from different data sources is becoming a popular and challenging task for researchers[14]. Many networks have multiple versions, as different data providers may gather their data from different sources, some of the data may not be shared due to privacy concerns[15], or the data that is available may evolve over time. In many applications, it is desirable to integrate data from multiple sources and it is repeatedly shown that data mining and machine learning algorithms are usually more effective on integrated, more comprehensive network data[16,17]. A straightforward way of integrating network data from multiple sources is to compute a superposition of all versions by taking the union of all edges (and/or assigning appropriate weights to the edges in the union).

An important task in analyzing integrated networks is the computation of node embeddings, i.e. learning low-dimensional representation of superposed networks[4,18]. Different versions of a network have the same set of nodes and different sets of edges (with identical semantics). In many settings, it may not be possible or desirable to superpose multiple versions of a network. For example, in integrated querying of networks from multiple databases, computation of embeddings for all possible combinations may not be feasible or efficient[19]. In other settings, multiple parties may not be willing the share their versions of the network due to privacy concerns, but may be willing to share the node embeddings computed from their versions. In these settings, the problem becomes one of computing an embedding that represents the node embeddings in the superposed network, using only the embeddings computed using the separate versions. For efficiency concerns, users may not want to compute node embeddings at query time. Using the node embeddings of the versions to learn the embedding of the superposed network would be more efficient than learning an embedding from the networks.

There are recently proposed methods for multiple network embedding. MNE[20] is a scalable multi-network embedding algorithm, which aims to encode multi-type relations into a unified embedding space while maintaining their distinctive properties. Mashup[21] learns feature representations of genes from multiple networks by jointly minimizing the difference between the observed diffusion states across all networks. deepNF[22] computes the probability of co-occurrence of nodes by RWR(Random Walk with Restart) for all networks and uses autoencoder to get the lower-dimensional representation. Those methods use the networks instead of embeddings as input, or computing the embedding at query time, so they are not able to avoid sharing networks and be efficient in querying systems.

We here introduce the notion of "consensus embedding" as an embedding for the superposed network that is computed from the embeddings of separate versions.

In this thesis, we propose multiple approaches to compute consensus embeddings:

(1) Linear dimensionality reduction via singular value decomposition[23].

(2) Non-linear dimensionality reduction via variational autoencoders[24,25].

(3) Construction and superposition of networks that represent node similarities computed from individual embeddings.

In order to investigate how well consensus embeddings can reconstruct the embedding of the superposed networks, we compare the consensus embeddings produced by these methods with the node embeddings computed directly from the superposed network. For this purpose we define a novel criterion, named "fidelity", which aims to measure the consistency of two different embeddings (one computed directly from superposed networks, one computed as a consensus embedding). Fidelity assesses consistency based on the pairwise similarity of nodes as captured by each embbeding. Using fidelity as a performance criterion, we systematically investigate the quality of consensus embeddings provided by different methods, as well as the reconstructability of different embedding techniques. We also investigate the relationships the number dimensions in the node embeddings and fidelity.

In our experiments, we use protein-protein interaction[26,27] and co-authorship networks[28,29] derived from multiple resources. Our results show that (i) community-based embedding techniques permit more accurate computation of consensus embeddings as compared to role-based techniques, (ii) linear dimensionality reduction produces more accurate consensus embeddings as compared to non-linear dimensionality reduction, and (iii) similarity networks improve the fidelity of consensus embeddings for role-based techniques.

We also assess the performance of consensus embeddings in the context of machine learning applications. For this purpose, we focus on link prediction, an important problem in network analysis that is often addressed using node embeddings[30,31]. The low-dimensional representations of networks preserve the structural information of graphs, e.g. proximity or similarities, and thus can be used as features in building machine learning models for link prediction[32,33]. In our work, we test the link prediction performance of consensus embeddings and compare it with that of the superposed network embeddings[11].

The remainder of the thesis is organized as follows. We first discuss some important concepts and existing approaches in Chapter 2. Then, we define consensus embeddings and present our proposed solutions to the computation of consensus embeddings in Chapter 3. We present systematic experimental results on protein interaction and social network data in Chapter 4. We conclude our discussion in Chapter 5.

# 2 Background and Related Work

Our work is based on network embedding methods and dimensionality reduction. In this section, we briefly introduce existing approaches to network embedding and some important concepts in dimensionality reduction. We also introduce the link prediction problem, which is also an application of node embedding.

## 2.1 Node Embedding

Node embedding aims to learn a low-dimensional representation of nodes in networks[1]. Given a graph $G = (V, E)$, a node embedding is a function $f : V \to R^d$ that maps each node $v \in V$ to a vector in $R^d$ where $d \ll |V|$. A node embedding method computes a vector for each node in the network such that the proximity in the embedding space reflects the proximity/similarity in the network.

In the last few years, many methods have been developed to compute node embeddings in a given network. Methods usually differ in terms of how they formulate the similarity between nodes (or the objective function that specifies the correspondence between the embedding and network topology). Communities and roles are the two important notions in studying the topology of networks[1]. Node embedding methods can also be roughly divided into community-based approaches and role-based approaches. As representatives of these different approaches, we here consider node2vec and role2vec: node2vec is a community-based approach while role2vec is a role-based approach.

### 2.1.1 Community-based node embedding

In a network, communities are defined by the connections among nodes. Nodes that are highly connected to each other are considered community. Thus, community-based embedding methods aim to map nodes that belong to the same community to closer points in the embedding space. In other words, these methods, including node2vec[7], LINE[9], and deepwalk[10], measure node similarity by their proximity in the network.

Node2vec[7] learns a representation of a network by optimizing a neighborhood preserving objective function. It measures the distances of the node representations by the random walk-based proximity of nodes. It runs short random walks starting from each node in the network. It uses the information provided by these truncated random walks with Skip-Gram to learn latent representations of vertices. The short random walks are flexible, biased random walks that can trade-off local and global structures of the network by exploring neighborhoods of nodes with both BFS(Breadth First Search) and DFS(Depth First Search). The random walk paths result in a neighborhood set $N_S(u)$ for every $u \in V$.

Node2vec optimizes an objective function that maximizes the log-likelihood of observing a network neighborhood $N_S(u)$ for a node $u$ given the embedding function $f$, i.e. it computes:

$$f^* = \text{argmax}_f \sum_{u \in V} \log Pr(N_S(u) | f(u)) \tag{2.1}$$

where $N_S(u)$ is a network neighborhood of node $u$.

### 2.1.2 Role-based node embedding

Roles are defined by structural features, e.g. the role of the hubs is to connect many nodes to each other, while the role of the bridges is to serve as one of the few nodes that connect different parts of the network. Role-based node embedding methods, e.g. role2vec[8], struc2vec[34], node2bits[35], make use of the structural features of nodes and measure node similarities by their structural graph functions. Instead of using traditional random walk, role2vec[8] uses the flexible notion of feature-based random walks.

Role2vec defines a matrix $X$ of node features, where each row $x_i$ is the observed node features of node $v_i$, including observed attributes, topological features, and/or node types for heterogeneous graphs. Role2vec maps vertices to vertex roles by learning a function $\Phi$ that maps all the vertices to a set of $m$ vertex roles where $m \ll |V|$, i.e. $\Phi(x_i)$ is the vertex role of node $v_i$. Each vertex role contains nodes that are structurally similar. Then, it uses feature-based random walks to derive role-based embedding for the nodes that capture structural properties. A feature-based random walk path is a sequence of adjacent vertex-roles in a network. Role2vec learns the embedding by the proximity relationships based on the feature-based random walks.

### 2.1.3 Applications of Node Embedding

The applications for node embeddings include network visualization, clustering, node classification, and link prediction[36].

**Visualization:** By mapping nodes into a low-dimensional space, we can visualize the graph into a 2D or 3D interface. For example, t-SNE[37] visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map.

**Clustering:** By applying clustering algorithms to node embedding, we are able to cluster similar nodes and detect communities. For example, GEMSEC[38] is a graph embedding algorithm which learns a clustering of the nodes simultaneously with computing their embedding.

**Node Classification:** : Given a partially labeled network, node classification aims to predict the label of unlabelled nodes. Neighboring nodes in node embeddings are more likely to share the same label. For example, GraphSAGE[39] uses inductive node classification that leverages node feature information to efficiently generate node embeddings for previously unseen data.

**Link Prediction:** : Link prediction is an important task in network analysis[40]. Given a network $G = (V, E)$, link prediction aims to predict the potential edges that are likely to appear in the network based on the topological relationships between pairs of nodes. Link prediction can be supervised[41] or unsupervised[42]. For supervised link prediction, the known links serve as positive samples and disconnected pairs of nodes serve as negative samples. The embedding vectors of nodes are treated as feature vectors and used to train the classifiers[11,32]. For unsupervised link prediction, the distances between pairs of vectors can be used to predict the proximity between nodes in the network and thus predicts the potential edges by ranking the distances[12,33].

## 2.2 Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of dimensions in a feature set. Suppose we are given a feature set with dimensionality $D$, dimensionality reduction methods aim to reduce it into a feature set with dimensionality $d$ where $d < D$. Dimensionality reduction methods can be linear or non-linear.

- **SVD (singular value decomposition)** [23] is a linear dimensionality reduction method. It is an algorithm that factors an $m \times n$ matrix $M$ into three component matrices,$U$, $S$, and $V$. The diagonal values of $S$ are called singular values of $M$. Dimensionality reduction can be achieved by setting the smallest singular values to 0.
- **Variational autoencoder**[24,25] is a kind of neural network that learns to recreate the input features. With the nonlinear activation functions at each layer, autoencoder is capable of learning nonlinear relationships.

# 3 Methods

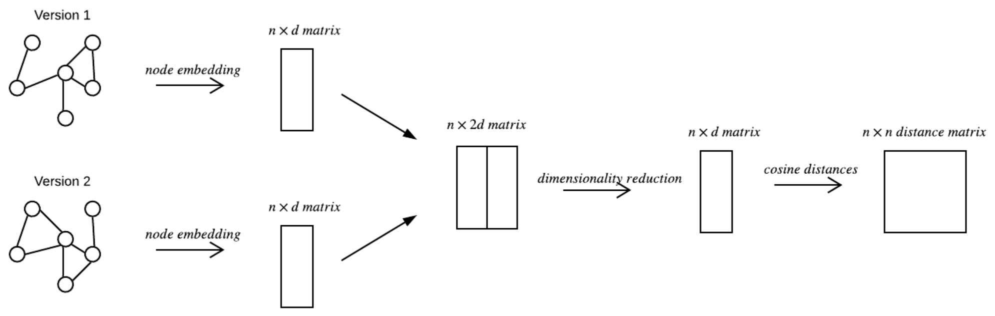

Figure 3.1. **The framework for the computation of consensus embeddings using dimensionality reduction methods.** The graphs labeled Version 1 and Version 2 represent two versions of a network with a fixed node set and different (possibly overlapping) edge sets. The objective is to compute node embeddings for the network obtained by superposing these two versions. At the absence of the superposed network, we compute the consensus embedding by computing separate embeddings for each version and then using dimensionality reduction to compute a common reduced-dimensional space for the two embedding spaces. Finally, we use the embeddings to assess the similarity between pairs of nodes in the network.

## 3.1 Problem Definition

In this section, we formalize the problem of integrating multiple networks and computing node embeddings. For simplicity, we assume that there are only two versions of the network, but the framework we develop here directly applies to multiple networks as well. Let $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ be two versions of a graph with the same set of nodes and different sets of edges. The superposition of $G_1$ and $G_2$ is $G = (V, E_1 \cup E_2)$. Assume that $d$-dimensional node embeddings $X_1$ and $X_2$ for $G_1$ and $G_2$ are given, where $X_1$ and $X_2$ are $n \times d$ matrices. Our objective is to use $X_1$ and $X_2$ to compute $d$-dimensional node embeddings $X_c$ for $G$, without using any other information on $G_1$ and $G_2$. We call $X_c$ the *consensus embedding* for $X_1$ and $X_2$. Our goal is to use the consensus embedding to represent the proximity relationships of both networks, so we want it to be similar to the embedding $X_s$ of the superposed network $G$.

This framework is illustrated in Figure 3.1, in which $node\ embedding$ can be any method for the computation of node embeddings (we here use node2vec and role2vec), and $dimensionality\ reduction$ can be any dimensionality reduction method (we here use SVD or variational autoencoder). We also use a third method for computing consensus embedding, namely vec2net, which is not a standard dimensionality reduction approach.

## 3.2 Generating Embeddings for Separate Networks

To compare the fidelities of community-based embeddings and role-based embeddings, we use node2vec and role2vec to generate node embeddings $X_1$ and $X_2$ of two networks. We will use these two node embeddings as the input of the methods in the next section to get the consensus embeddings. This step is shown as $node\ embedding$ in figure 3.1. We also compute the embedding $X_s$ by node2vec and role2vec in order for the fidelity scores in the following steps.

## 3.3 Computing Consensus Embeddings

By multiple network embedding methods explained earlier, we are able to get multiple low-dimensional representations of $G_1$ and $G_2$. In this way, we are mapping the two networks into two different $d$-dimensional spaces. Then, by merging these two representations into the same $d$-dimensional space, we are able to integrate the two embeddings. Before we implement the dimensionality reduction methods, we first create a $n \times 2d$ matrix by adding $X_2$ to the right of $X_1$. Here, we implemented two dimensionality reduction approaches to address this problem. Our goal is to get a $n \times d$ representation $X_c$ of the combined embedding $X$. Also, we developed $vec2net$ algorithm that constructs a new network maintaining the proximity relationships of both embeddings.

### 3.3.1 Singular Value Decomposition(SVD)

Singular Value Decomposition(SVD) is a matrix decomposition method for reducing a matrix to its constituent parts. The singular value decomposition of an $m \times n$ matrix $M$, whose rank is $r$, is a factorization of the form $USV$, where $U$ is an $m \times r$ unitary matrix, $S$ is an $r \times r$ diagonal matrix, and $V$ is an $r \times n$ unitary matrix. $S$ is a diagonal matrix and the diagonal values of $S$ are called the singular values of $M$.

If we need a $d$-dimensional matrix $M'$, we set $r - d$ smallest singular values to zero and remove the corresponding rows or columns in $U$ and $V$ because they will multiply with zero when calculating $M = USV$. By doing this, we can drop the lowest singular value from the decomposition of $M$ and get three new matrices $U'$, $S'$, and $V'$. The reduced matrix $M'$ is calculated by only $U'$ and $S'$, i.e. $M' = U'S'$. $M'$ will only have $d$ columns because it is the product of a $m \times d$ matrix and a $d \times d$ matrix.

### 3.3.2 Convolutional Autoencoder

An autoencoder is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. Given a set of vectors$\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, where $x^{(i)} \in \mathbb{R}^d$ where $d$ is the number of dimensions, autoencoder learns a neural network that returns a set of n vectors such that $y^{(i)} \approx x^{(i)}$. The loss function of autoencoder is:

$$L(x, y) = \|x - y\|^2 \tag{3.1}$$

By minimizing the loss function, we can reconstruct $x$ with the neural network. Figure 3.2 shows the process of standard autoencoder.

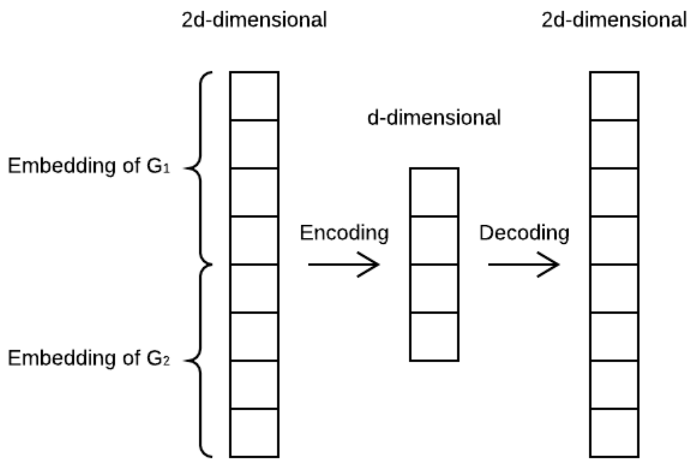

Figure 3.2. An example of autoencoder used to integrate two embeddings

In our method, we use a 2D-convolutional autoencoder[25]. Same as standard autoencoder, convolutional autoencoder also aims to output the same vectors as input. The convolutional autoencoder contains convolutional layers in the encoder part of the autoencoder. In every convolutional layter, there is a filter that slides around the input matrix to compute the next layer. Convolutional autoencoder also have pooling layers after each convolutional layer. In the decoder part, there are deconvolutional layers and unpooling layers that recovers the input matrix. After the training process, we get a $d$-dimensional vector for every node by the encoding part of the autoencoder.

### 3.3.3 vec2net

We develop an algorithm,vec2net, that computes a new network from $n$ vectors by their proximity relationships. The $n$ vectors represents $n$ nodes in a network. Figure 3.3 shows the main procedures of vec2net. Given $n$ $d$-dimensional vectors, we can get a $n \times n$ distance matrix by computing the pairwise cosine distance between every pair of vectors. Here, the cosine distance between two vectors $v_1$ and $v_2$ is defined as $1 - cos(v_1, v_2)$. The distance matrix indicates the proximity between the vectors. Then, we can add edges to the network by the rank of the distances.

In our problem, we are given two node embeddings $X_1$ and $X_2$, so the $vec2net$ algorithm should construct a new network that can maintain the proximity of both embeddings. Therefore, we have 2 sets of $d$-dimensional vectors, then 2 cosine distance matrices can be generated from them. In order to maintain the proximity of both networks, we compute the min distance of every pair of nodes. The pseudocode of multiple-network vec2net is in Algorithm 1. In the output network, two nodes are connected by an edge if they are close in at least $G_1$ or $G_2$. By computing the embedding of the new network, we are able to get a new embedding which has the same number of dimensions as $X_1$ and $X_2$. In this way, we can also get the similar $n \times d$ matrix as the dimensionality reduction approaches mentioned in the previous sections.

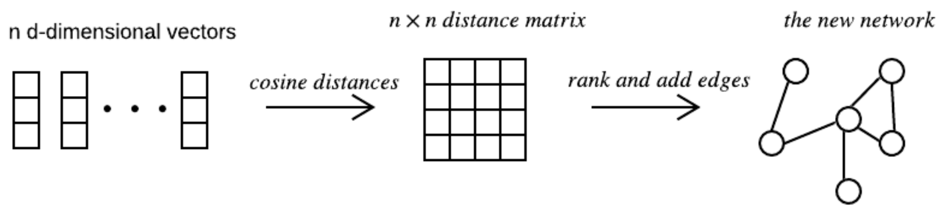

Figure 3.3. The main procedures of vec2net

**Algorithm 1:** Vec2net

```
Require: $X_1$,$X_2$: $d$-dimensional vectors for $\forall v \in V$, $k$: number of edges in the resulting
  network
Ensure: A network $G = (V, E)$ with n nodes
  $E \leftarrow \emptyset$
  $D_1 \leftarrow$ cosine_distances($X_1, X_1$)
  $D_2 \leftarrow$ cosine_distances($X_2, X_2$)
  $D \leftarrow min\{D_1, D_2\}$
  for i = 1 to n do
    $c \leftarrow$ column index of min $D[i,:]$
    $E \leftarrow E \cup (V[i], V[c])$
  end for
  sort all values in $D$
  $[idx_r, idx_c] \leftarrow$ indices of the $k$ minimum elements in $D$
  $E \leftarrow E \cup (V[idx_r], V[idx_c])$
```

In Algorithm 1, we add edges to the graph according to their proximity in the embedding spaces. To guarantee that the new graph contains all the nodes, we should at least add one edge to every node. Therefore, the first step of $vec2net$ algorithm is to find the closest neighbor of every node by cosine distances. Then, we rank all the cosine distances and find the pairs of nodes with top $k$ nearest distances. After we add top $k$ edges into the graph, we get a new graph with $n$ nodes and approximately $n+k$ edges.

## 3.4 Fidelity

To measure the fidelities of the methods, we aim to compare the resulting embedding of every method with the embedding of the superposed network. Because it is hard to compare two embedding matrices, we try to compare their cosine distance matrices.

The Mantel test[43] compares two distance matrices by computing the correlation between the distances in the lower/upper triangular portions of the symmetric distance matrices. Given two distance matrices $A$ and $B$ the Mantel test computes a test statistic

$r$ as equation 3.2:

$$r = \frac{1}{d-1} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{A_{ij} - \bar{A}}{s_A} \frac{B_{ij} - \bar{B}}{s_B} \tag{3.2}$$

where $d = \frac{n(n-1)}{2}$ and $A, B$ are distance matrices, $\bar{A}$, $\bar{B}$ are the average distance of $A$ and $B$, and $s_A$ and $s_b$ are the standard deviation of the distances in $A$ and $B$. The result $r$ falls in the range of -1 to 1. If $r$ is close to -1 or 1, then the two distance matrices have strong negative or positive correlations. If $r$ is 0, then the two distance matrices have no correlation.

In our experiments, we use the embedding of the superposed networks as the "standard" embedding, so their cosine-similarity matrices are the standard for us to measure the fidelity. Then, one of the input matrices of the Mantel test is the cosine-similarity matrix of the superposed network. The other input matrix should be the cosine-similarity matrix of the embedding resulting from SVD/autoencoder/vec2net.

# 4 Results and Discussions

In this chapter, we present comprehensive experimental results on versioned networks in the context of two different applications and discuss the implications of the results.

## 4.1 Experimental Setup

### 4.1.1 Datasets

In our experiments, we use two types of datasets: protein protein interaction(PPI) networks and co-authorship networks. Both of these two types of networks are undirected and we treat them as unweighted networks. We use a pair of datasets for both types of networks.

**PPI Networks.** Protein Protein Interaction networks express the physical interactions and functional associations between proteins. The nodes in PPI networks represent proteins and the edges connect pairs of interacting proteins. We use two human PPI network datasets in our experiments:

- BioGRID[26]: This is an extensive repository of curated genetic and protein interactions.
- STRING[27]: This is a protein interaction network with both physical and functional interactions from diverse sources.

**Co-authorship Networks.** In co-authorship networks, each vertex represents an author and each edge represents a co-author relation, i.e., there is a undirected link if between two vertices if the two authors published at least one paper together. We use two co-authorship networks, including the AMiner Computer Science (CS) dataset and the DBLP computer science bibliography.

- Aminer CS dataset[28]: The Aminer dataset is comprised of the collaboration network among the authors.
- The DBLP CS bibliography[29]: The DBLP computer science bibliography is the on-line reference for bibliographic information on major computer science publications.

The two pairs of networks are two sets of test input. To avoid issues with data inconsistency and the need for normalization, for each pair of networks, we identify the set of nodes that appear in both versions of the network and use the networks induced by the nodes at this intersection (i.e., we remove the nodes that do not appear in the other network). Because the coauthorship networks are too large and thus hard to be processed (especially for computing the pairwise distances), we only use a subnetwork of them.

### 4.1.2 Preprocessing

First, we extract data from the datasets. We get the edge lists from the datasets and then represent the datasets as networks. We assign new IDs to the nodes to map the nodes from one dataset to the other one. After that, we get a node set and an edge list of each dataset with our own IDs.

In order to obtain two networks with the same set of nodes, we remove some of the non-overlapping nodes the datasets. For both protein interaction networks and co-authorship networks, if a node only exists in one of the datasets, we remove it from the node set as well as the edges connected with it from the edge list. After the first step,

| Dataset | Biogrid | String | Aminer | DBLP |
|---|---|---|---|---|
| Number of nodes | 23268 | 15939 | 26692 | 25736 |
| Number of edges | 481338 | 1070938 | 133748 | 25355 |

Table 4.1. Number of nodes and edges before preprocessing

| Dataset | Biogrid | String | Aminer | DBLP |
|---|---|---|---|---|
| Number of nodes | 13936 | 13936 | 21757 | 21757 |
| Number of edges | 266202 | 920858 | 67954 | 20823 |

Table 4.2. Number of nodes and edges after preprocessing

some nodes may have no edge in one of the networks. Therefore, we remove them from the node set together with their edges in the other network, too. Because new nodes without an edge will appear again, so we should do this for multiple times. After removing the redundant nodes and edges for several times, we will get two smaller networks with the same set of nodes.

## 4.2 Fidelity of Consensus Embeddings Computed Using SVD and Autoencoder

To measure the fidelity of consensus embeddings computed using SVD or variational autoencoder, the input includes the consensus embedding $X_c$ resulting from both methods, and the integrated embedding $X_i$. We assess the fidelity of $X_c$ to $X_i$ as a function of the number of dimensions in the embedding.

Figure 4.1 shows the fidelity provided by each dimensionality reduction method as a function of number of dimensions. The four plots show different combinations of datasets and node embedding methods. As seen in Figure 4.1, node2vec leads to consensus embeddings with more fidelity as compared to role2vec in general. It is interesting that the fidelity of consensus embeddings for node2vec increases with increasing number of dimensions, while it decreases or stays constant with increased dimensionality for role2vec. This is not a surprising result as communities can be more robust to subsampling of networks as compared to other topological characteristics (i.e., for a densely connected group of nodes, the relative density can be preserved when edges are subsampled).

We observe that consensus embeddings computed using SVD provide slightly higher fidelity as compared to those computed using variational autoencoder for both node2vec and role2vec. We also observe that the fidelity of consensus embeddings for co-authorship networks is consistently less than that for PPI networks. We attribute this difference to the fact that co-authorship networks are sparser than PPI networks and elaborate on this finding further later in this chapter.

## 4.3 Fidelity of Consensus Embeddings Computed Using vec2net

From the two node embeddings $X_1$ and $X_2$, we compute a new network $G'$ by vec2net. After we compute the node embedding of the new network $G'$, we obtain the consensus embedding as a $n \times d$ matrix. Note that if the embedding method of vec2net is node2vec (or role2vec), we still use node2vec(or role2vec) here. Because the number of edges $k$ in $G'$ is an important parameter for vec2net, it may also have a significant effect on fidelity. For this reason, we assess fidelity as a function of the number of edges in the new network $G'$.

Figure 4.2 shows the fidelity of the consensus embeddings computed by the vec2net algorithm. The four plots show different combinations of datasets and node embedding methods. For all plots in figure 4.2,fidelity tends to increase with the number of edges $k$ in the reconstructed networks, and saturates when $k$ reaches some point. In most cases,

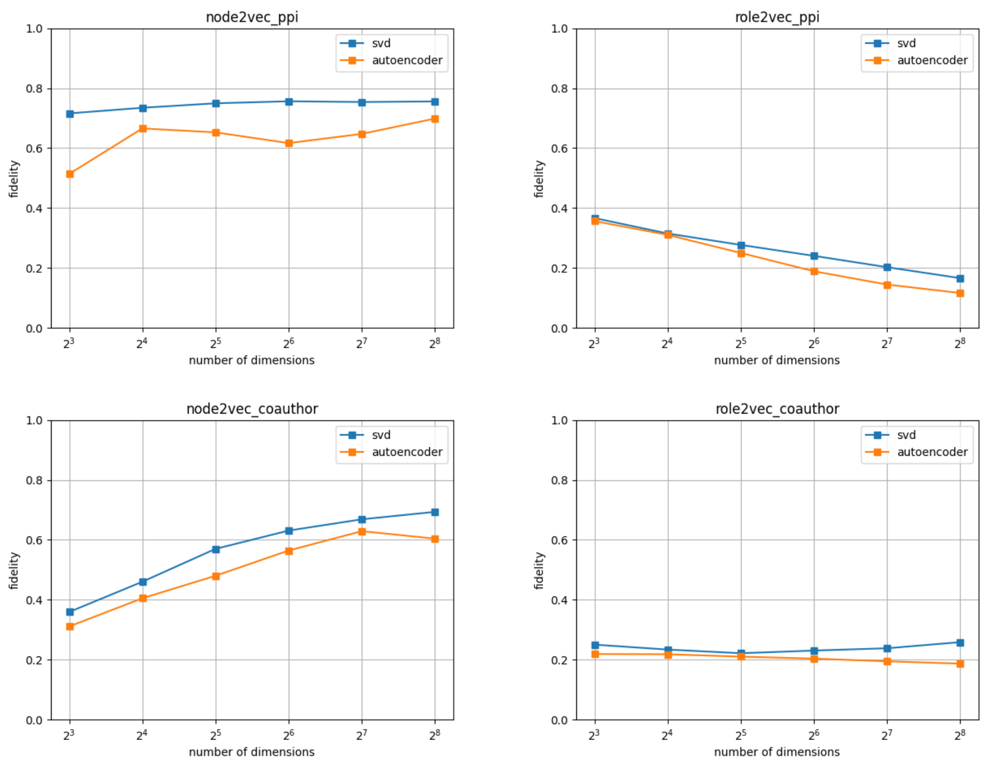

Figure 4.1. **Performance of dimensionality reduction methods in recovering the node embeddings in the superposed network.** The fidelity of the embeddings reconstructred using the embeddings from individual networks to the embeddings computed directly using the superposed network is shown as a function of number of dimensions in the embeddings. Results are shown for embedding methods (a) Node2vec, (b) Role2vec for the protein-protein interaction (PPI) (upper panel) and co-authorship networks (lower panel).

the maximum fidelity accomplished by lower-dimensional embeddings is better than that of higher-dimensional embeddings.

The red dashed lines in the plots show the best performance of SVD in the same case. We use SVD instead of autoencoder for comparison because SVD performs better than autoencoder in most cases. Compared to the red dashed line in all plots, the best fidelity of vec2net is always lower than the best fidelity of SVD, and the fidelity of vec2net is more close to the fidelity of SVD for role2vec. This result suggests that the cosine similarity of embeddings is informative of the similarity between nodes up to a certain point. After that point, addition of more edges does not provide any additional information on the relationships between the nodes.

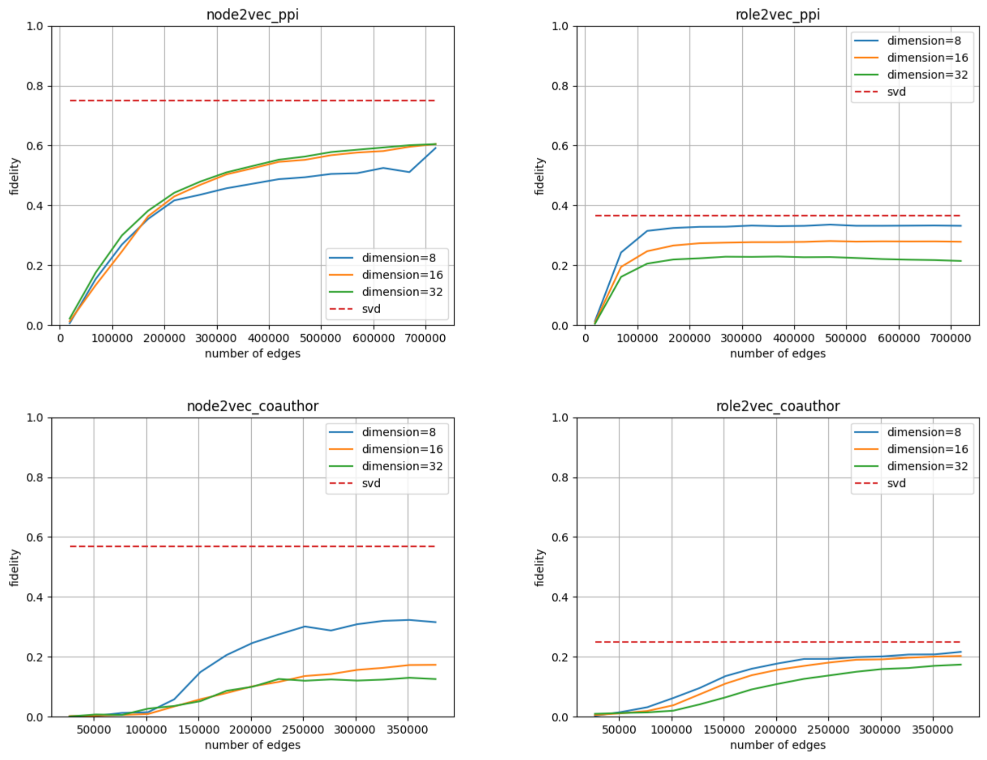

Figure 4.2. **Performance of vec2net algorithm in recovering the node embeddings in the superposed network.** The fidelity of the embeddings reconstructed using the networks constructed with cosine similarity to the embeddings computed directly using the superposed network is shown as a function of number of edges in the input of vec2net. Results are shown for embedding methods (a) Node2vec, (b) Role2vec on the protein-protein interaction (PPI) (upper panel) and co-authorship networks networks (lower panel). The red dashed line is the best fidelity of SVD for number of dimensions = 8, 16, or 32.

## 4.4 Effect of overlap and density

The input of our method includes two network versions, $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, so the similarity between the two versions might affect the fidelity.

From the experimental results, the density of $G_1$ and $G_2$ can also affect fidelity. Therefore, in our experiment, we test different densities of $G_1$ and $G_2$. For these purposes, we create our testing cases by the following steps:

- **Choose a graph $G$:** $G$ is a graph with sufficiently high density.
- **Creating $G_1$ with $k$ edges:** Randomly extract $k$ edges from $G$ and make sure all nodes are involved in $G_1$.
- **Creating $G_2$ with the same density:** We should control the percentages of overlap, i.e. $|E_1 \cap E_2| = \alpha\%|E_1|$ where $o$ is a constant for every test case. To do this, we remove $E_1$ from $G$ and randomly extract $(1-\alpha\%)k$ edges from the leftover edges. Then, randomly extract $\alpha\%k$ edges from $G_1$. At last, add all the $k$ edges to $G_2$. We should also make sure that all nodes are involved.
- **Repeat with different densities and percentages of overlap:** The numbers of edges in the dense networks are ten times of the numbers of edges in their corresponding sparse networks. The percentages of overlap loops through 0, 0.2, 0.4, 0.6, 0.8, and 1.

We use the new generated $G_1$ and $G_2$ as the input, and compare the output embeddings with the node embedding of $G' = (V, E_1 \cup E_2)$.

**Figure 4.3 and figure 4.4 shows the fidelities of dimensionality reduction methods.** The results are shown as a function of the percentages of overlapping edges of $G_1$ and $G_2$. The gray shaded part on each graph is the confidence interval of multiple experiments. The red and blue lines are the average fidelities of sparse and dense networks.

For sparse networks, the fidelity increases when the percentages of overlap increases. However, the fidelity of dense networks are higher and more stable, without an obvious increase.

**The results of Vec2net is shown in figure 4.5.** For vec2net, the number of edges in the newly constructed network is large enough to get the best fidelity of each testing case.

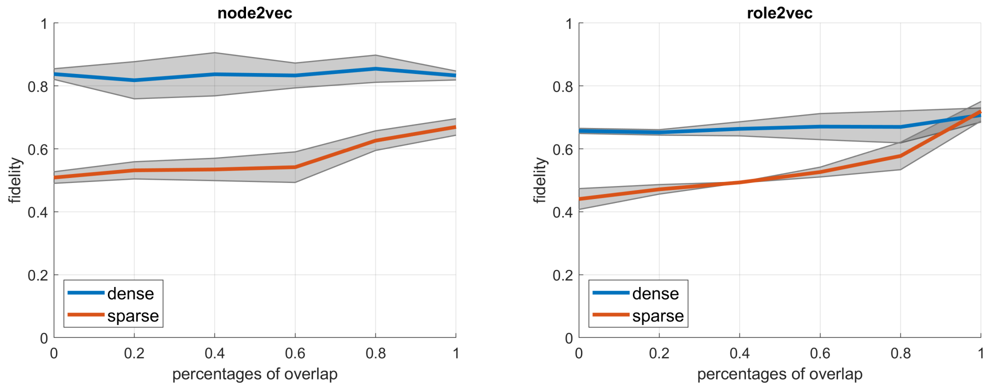

Figure 4.3. **Influence of the overlap between networks $G_1$ and $G_2$ with different densities in SVD.** Node2vec(left) and role2vec(right). Number of dimensions = 8. The sparse $G_1$ and $G_2$'s average degree is about 1.7, and the dense networks' number of edges are about 10 times of the sparse ones.

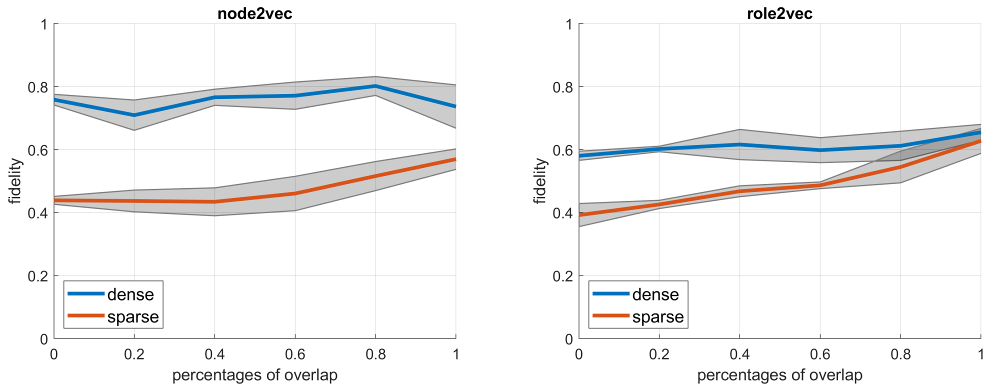

Figure 4.4. **Influence of the overlap between networks $G_1$ and $G_2$ with different densities in autoencoder.**Node2vec(left) and role2vec(right). Number of dimensions = 8. The sparse $G_1$ and $G_2$'s average degree is about 1.7, and the dense networks' number of edges are about 10 times of the sparse ones.

The overlap between $G_1$ and $G_2$ does not have a large impact on the fidelity of vec2net. Instead, different densities of the graphs result in different levels of fidelities.

## 4.5 The Performance of Link prediction

To test the influence of fidelity on the applications of node embedding, we compare the performance of link prediction of the node embeddings of superposed networks and

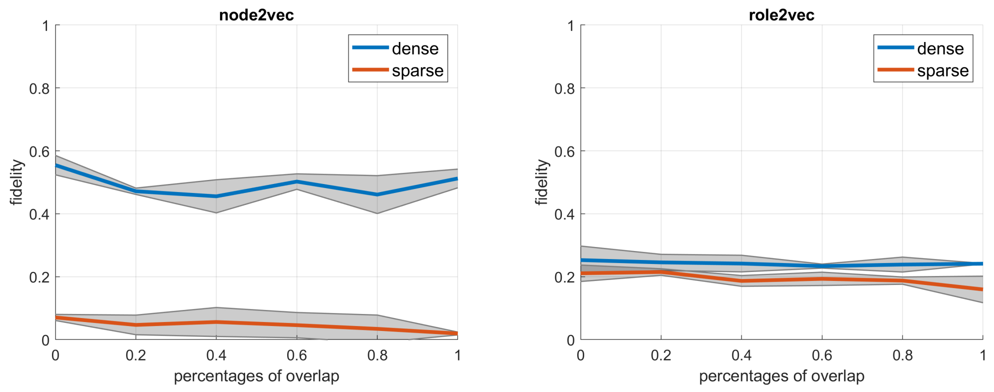

Figure 4.5. **Influence of the overlap between networks $G_1$ and $G_2$ with different densities in vec2net.** Node2vec(left) and role2vec(right). Number of dimensions = 8. The sparse $G_1$ and $G_2$'s average degree is about 1.7, and the dense networks' number of edges are about 10 times of the sparse ones.

consensus embeddings generated from different methods. BioNEV[11] is an easy-to-use Python package that can test multiple tasks of graph embedding. Given a network and its node embedding, BioNEV can generate random training and testing sets to evaluate the link prediction performance of the embedding. By using BioNEV, we can get the auc scores of the node embeddings generated from different methods as well as those of the superposed networks.

Figure 4.6 shows the performance of consensus embeddings in link prediction compared with the performance of the superposed networks' embeddings. In most cases, dimensionality reduction methods perform as well as the embedding of superposed network, and vec2net is a little worse than them.

For node2vec, the embeddings of superposed networks always have the highest auc scores, the auc scores of dimensionality reduction methods are getting higher, and most of the auc scores of vec2net are the worst. This result correlates well with the result in figures 4.1 and 4.2, where the fidelity of dimensionality reduction methods increases as the number of dimensions increases. By comparing the four lines, we can see that

SVD is a little better than autoencoder, and vec2net performs worse than dimensionality reduction methods.

For role2vec, the auc scores of superposed network embeddings are not always the best. However, when we compare the difference of blue lines and other lines in the right panel of figure 4.6, we can see that SVD(orange) is a little closer to the superposed network embedding(blue). For PPI networks, the auc scores are getting more and more apart from each other, which correlates with figure 4.1 and figure 4.2 where fidelities are decreasing with the increase of dimensions. For coauthorship networks, most of the fidelities of SVD and autoencoder are close to the blue line, while vec2net are getting more and more apart from others.

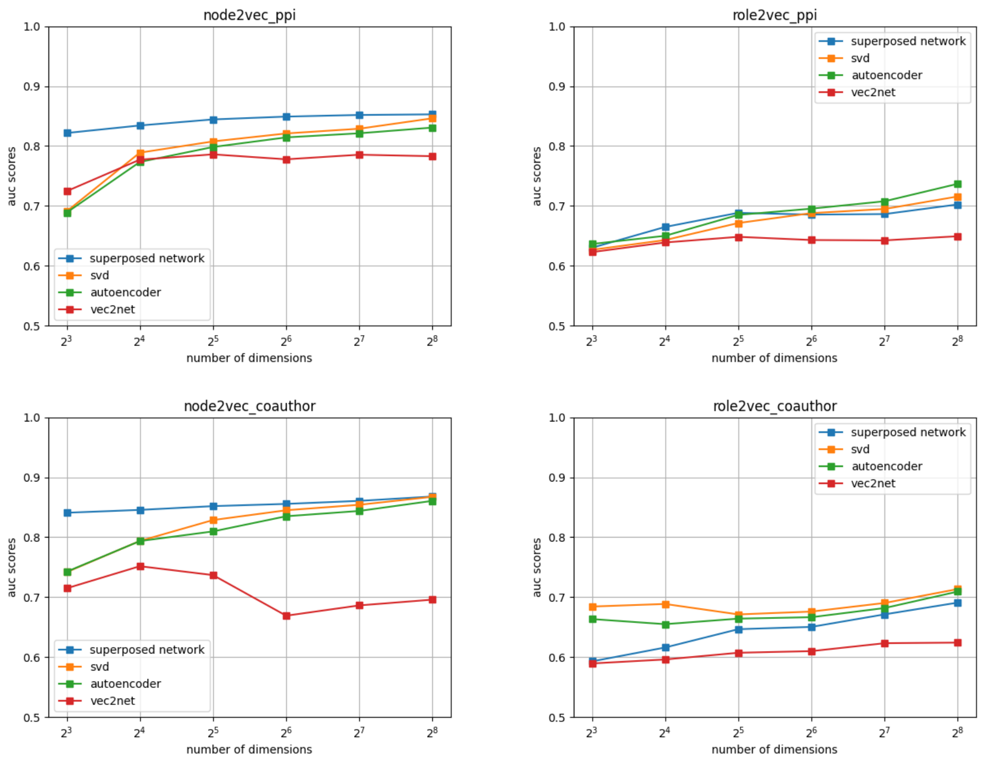

Figure 4.6. **Performance of all consensus embeddings in link prediction.** The auc scores of consensus embeddings is shown as a function of number of dimensions in the embeddings. Results are shown for embeddings methods (a) Node2vec, (b) Role2vec for the protein-protein interaction (PPI) (upper panel) and co-authorship networks networks (lower panel).

# 5 Conclusions

In this work, we consider the problem of computing node embeddings for superposed networks derived from the multiple network versions. We define consensus embeddings as the node embeddings of the superposed network derived from the embeddings of individual versions. We consider multiple approaches to computing embeddings for the superposed network from the embeddings of individual versions: (i) linear dimensionality reduction via singular value decomposition, (ii) non-linear dimensionality reduction via convolutional autoencoders, (iii) construction of new networks that represent similarity of embeddings.

We use multiple ways to generate the consensus embeddings with different combinations of node embedding methods and dimensionality reduction methods/vec2net.

We propose *fidelity* to assess the recoverability of consensus embeddings. Fidelity is defined as the Mantel test result of the distance matrices of superposed network's embeddings and the consensus embeddings. From the experimental results, node2vec performs better than role2vec, and dimensionality reduction methods perform better than vec2net.

An application of consensus embedding is link prediction. We test the performance of link prediction of the consensus embeddings and found that most of the auc scores of consensus embeddings are close to the auc scores of the superposed network embeddings.

An important future work might be exploring new applications of consensus embeddings. Besides link prediction, consensus embedding can be also useful in other fields as long as multiple versions of a network are to be processed.

Another future work is to find out the potential relationships between fidelity and the performance of consensus embedding in other applications like link prediction. This helps us to estimate the performance of consensus embeddings on any applications and thus make *fidelity* a more significant standard in assessing consensus embeddings.

## References

[1] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen Ahmed, Danai Koutra, and John Boaz Lee. From community to role-based graph embeddings. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2019.

[2] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 6(1):3–28, 2020.

[3] J. Lin, L. Zhang, M. He, H. Zhang, G. Liu, X. Chen, and Z. Chen. Multi-path relationship preserved social network embedding. *IEEE Access*, 7:26507–26518, 2019.

[4] J. Zhang, C. Xia, C. Zhang, L. Cui, Y. Fu, and P. S. Yu. Bl-mne: Emerging heterogeneous social network embedding through broad learning with aligned autoencoder. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 605–614, 2017.

[5] Sezin Kircali Ata, Yuan Fang, Min Wu, Xiao-Li Li, and Xiaokui Xiao. Disease gene classification with metagraph representations. *Methods*, 131:83 – 92, 2017. Systems Approaches for Identifying Disease Genes and Drug Targets.

[6] Chang Su, Jie Tong, Yongjun Zhu, Peng Cui, and Fei Wang. Network embedding in biomedical data science. *Briefings in Bioinformatics*, 21(1):182–197, 12 2018.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[8] Nesreen Ahmed, Ryan A. Rossi, John Boaz Lee, Xiangnan Kong, Theodore L. Willke, Rong Zhou, and Hoda Eldardiry. Learning role-based graph embeddings. *ArXiv*, abs/1802.02896, 2018.

[9] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[11] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics (Oxford, England)*, 36(4):1241—1251, February 2020.

[12] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[13] Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM ’17, page 377–386, New York, NY, USA, 2017. Association for Computing Machinery.

[14] K. Shobha and S. Nickolas. Integration and rule-based pre-processing of scientific publication records from multiple data sources. In Suresh Chandra Satapathy, Vikrant Bhateja, J. R. Mohanty, and Siba K. Udgata, editors, *Smart Intelligent Computing and Applications*, pages 647–655, Singapore, 2020. Springer Singapore.

[15] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, and Ximeng Liu. Privacy-preserving and high-accurate outsourced disease predictor on random forest. *Information Sciences*, 496:225 – 241, 2019.

[16] P. Wijegunawardana, K. Mehrotra, and C. Mohan. Finding rising stars in heterogeneous social networks. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 614–618, 2016.

[17] W. Zhang, Yanlin Chen, Shikui Tu, Feng Liu, and Qianlong Qu. Drug side effect prediction through linear neighborhoods and multiple data source integration. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 427–434, 2016.

[18] X. Shen, Q. Dai, S. Mao, F. Chung, and K. Choi. Network together: Node classification via cross-network deep network embedding. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2020.

[19] Tyler Cowman, Mustafa Coşkun, Ananth Grama, and Mehmet Koyutürk. Integrated querying and version control of context-specific biological networks. *Database*, 2020, 04 2020. baaa018.

[20] Hongming Zhang, Liwei Qiu, Lingling Yi, and Yangqiu Song. Scalable multiplex network embedding. In *IJCAI*, volume 18, pages 3082–3088, 2018.

[21] Hyunghoon Cho, Bonnie Berger, and Jian Peng. Compact integration of multi-network topology for functional analysis of genes. *Cell Systems*, 3(6):540 – 548.e5, 2016.

[22] Vladimir Gligorijević, Meet Barot, and Richard Bonneau. deepNF: deep network fusion for protein function prediction. *Bioinformatics*, 34(22):3873–3881, 06 2018.

[23] Sudeep Tanwar, Tilak Ramani, and Sudhanshu Tyagi. Dimensionality reduction using pca and svd in big data: A comparative case study. In Zuber Patel and Shilpi Gupta, editors, *Future Internet Technologies and Trends*, pages 116–125, Cham, 2018. Springer International Publishing.

[24] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232 – 242, 2016. RoLoD: Robust Local Descriptors for Computer Vision 2014.

[25] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 52–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[26] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 34(suppl_1):D535–D539, 01 2006.

[27] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, Lars J Jensen, and Christian von Mering. STRING v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47(D1):D607–D613, 11 2018.

[28] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 807–816, New York, NY, USA, 2009. Association for Computing Machinery.

[29] The dblp team. dblp computer science bibliography. Monthly snapshot release of April 2020. https://dblp.org/xml/release/dblp-2020-04-01.xml.gz.

[30] Hongxu Chen, Hongzhi Yin, Weiqing Wang, Hao Wang, Quoc Viet Hung Nguyen, and Xue Li. Pme: Projected metric embedding on heterogeneous networks for link prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1177–1186, New York, NY, USA, 2018. Association for Computing Machinery.

[31] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 592–600, New York, NY, USA, 2018. Association for Computing Machinery.

[32] K. Mallick, S. Bandyopadhyay, S. Chakraborty, R. Choudhuri, and S. Bose. Topo2vec: A novel node embedding generation based on network topology for link prediction. *IEEE Transactions on Computational Social Systems*, 6(6):1306–1317, 2019.

[33] Zhitao Wang, Chengyao Chen, and Wenjie Li. Predictive network representation learning for link prediction. In *Proceedings of the 40th International ACM SIGIR*

*Conference on Research and Development in Information Retrieval*, SIGIR '17, page 969–972, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 385–394, New York, NY, USA, 2017. Association for Computing Machinery.

[35] Di Jin, Mark Heimann, Ryan A. Rossi, and Danai Koutra. node2bits: Compact time- and attribute-aware node representations for user stitching. In Ulf Brefeld, Elisa Fromont, Andreas Hotho, Arno Knobbe, Marloes Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 483–506, Cham, 2020. Springer International Publishing.

[36] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[37] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[38] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '19, page 65–72, New York, NY, USA, 2019. Association for Computing Machinery.

[39] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[40] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4), December 2016.

[41] H. R. de Sá and R. B. C. Prudêncio. Supervised link prediction in weighted networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2281–2288, 2011.

[42] Tsung-Ting Kuo, Rui Yan, Yu-Yang Huang, Perng-Hwa Kung, and Shou-De Lin. Unsupervised link prediction using aggregative statistics on heterogeneous social networks. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 775–783, New York, NY, USA, 2013. Association for Computing Machinery.

[43] Nathan Mantel. The detection of disease clustering and a generalized regression approach. *Cancer Research*, 27(2 Part 1):209–220, 1967.