Deep Network Embedding with Dimension Selection

Faming Liang

Purdue University

November 24, 2024

Network Data

Networks are a powerful tool for describing complex relationships among a large number of entities (also known as nodes in network terminology) in a real-world system.

- social networks
- biological networks
- citation networks

Information extracted from networks can help uncover hidden patterns in complex systems, predict future interactions among entities, and support data-driven decision-making.

Embedding

- The embedding method assigns a low-dimensional vector, known as an embedding vector, to each node in the network.
- This transformation converts non-Euclidean structured network data into Euclidean data, facilitating downstream statistical analyses such as node classification, community detection, and link prediction.
- In the realm of network information extraction, embedding has emerged as a popular method in recent literature.

Embedding

- Early embedding methods are often developed based on the spectral decomposition of the adjacency matrix or other graph operators.
- Generative model-based methods: They assume that the observed network is compatible with a predefined generative model in the Euclidean space and the model parameters can be learned to achieve a good fit to the observed network. These generative models can be approximated using neural networks of different structures, such as shallow neural networks, autoencoders, and convolutional neural networks (CNNs).

Embedding: Dimension Selection

- A common practice is to treat the embedding dimension as a hyperparameter of the model or simply set it as a constant.
- An excessively high embedding dimension can cause the model overfitted, leading to high variability in prediction.

To address this issue, there have been only scattered efforts in the literature, see e.g., Seshadhri et al. (2020), Chanpuriya et al. (2020), and Gu et al. (2021).

Deep Embedding

For deep embedding methods, it is important to note that they might suffer from an identifiability issue due to the universal approximation ability of deep neural networks.

- We address this issues by performing sparse deep embedding within the framework of missing data imputation.
- Specifically, we treat the embedding vectors as missing data, reconstruct the network features using a sparse decoder, and simultaneously impute the embedding vectors and train the sparse encoder using an adaptive stochastic gradient MCMC algorithm (Liang et al., 2022).
- Under mild conditions, we show that the sparse decoder provides a parsimonious mapping (defined later in an appropriate sense) from the embedding space to network features, enabling an effective selection of the embedding dimension.

Skip-gram model

Let $\mathcal{G} = (V, E)$ denote a network, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of nodes, and $E = \{e_{ij}\}_{i,j=1}^{n}$ denotes the associated adjacency matrix. We define $N_i = \{v_k : e_{ik} = 1\}$ as the neighborhood of node i. Each node v_i has an embedding vector $z_i \in \mathbb{R}^{d_z}$, and its context has an embedding vector $z_i^c \in \mathbb{R}^{d_z}$.

The skip-gram model works under the maximum likelihood principle, which aims to find the optimal (z_i, z_i^c) pairs by minimizing the objective function:

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{v_j \in N_i} \log p(z_j^c | z_i), \tag{1}$$

where $p(z_j^c|z_i) = \exp(z_i^T z_j^c)/\sum_{v_k \in V} \exp(z_i^T z_k^c)$ encodes the local neighborhood information of node v_i.

A variety of methods have been developed based on similar ideas, see e.g., DeepWalk, LINE, and Node2Vec.

Deep Neural Graph Representation (DNGR)

DNGR minimizes the objective function:

$$\mathcal{L} = \sum_{v_i \in V} \|\psi_{dec}(z_i) - U_i\|^2, \quad s.t.\ z_i = \psi_{enc}(U_i), \tag{2}$$

where $U_i \in \mathbb{R}^{|V|}$ represents the i-th column of the Positive Pointwise Mutual Information (PPMI) matrix and captures the neighborhood information of node v_i, $|V|$ denotes the cardinality of the set V, and $\psi_{enc}(\cdot)$ and $\psi_{dec}(\cdot)$ denote the encoder and decoder functions, respectively.

Structural Deep Network Embedding (SDNE)

SDNE minimizes the objective function:

$$\begin{aligned}\mathcal{L} = \sum_{v_i \in V} \|(\psi_{dec}(z_i) - \boldsymbol{e}_i) \odot \boldsymbol{w}_i\|^2 + \alpha \sum_{i,j=1}^{|V|} e_{ij}\|z_i - z_j\|^2 + \nu \mathcal{L}_{reg}, \\ s.t. \;\; z_i = \psi_{enc}(\boldsymbol{e}_i),\end{aligned} \tag{3}$$

where $\boldsymbol{e}_i$ denotes the ith column of the adjacency matrix E, $\boldsymbol{w}_i = \{w_{i,j}\}_{j=1}^{|V|} \in \mathbb{R}^{|V|}$ represents a weight vector with $w_{i,j} = 1$ if $e_{ij} = 0$ and $w_{i,j} = \beta$ otherwise, and $\odot$ denotes Hadamard product. The Laplacian eigenmap penalty term $\sum_{i,j=1}^{|V|} e_{ij}\|z_i - z_j\|^2$ encourages nearby nodes to have similar embeddings, $\mathcal{L}_{reg}$ is an L_2-norm regularizer term used to prevent overfitting, and α and ν are regularization coefficients.

Geometric Deep Learning

Geometric Deep Learning minimizes the objective function:

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{j=1}^{|V|} e_{ij} \log \hat{p}_{ij}, \quad s.t. \;\; \hat{p}_i = \psi_{dec}(z_i), \;\; z_i = \psi_{CNN}(\boldsymbol{e}_i), \tag{4}$$

where $\hat{p}_i = (\hat{p}_{i1}, \hat{p}_{i2}, \ldots, \hat{p}_{i|V|})^T$, $\psi_{dec}(\cdot)$ is a fully connected neural network with the softmax activation function that maps an embedding vector to predicted labels, and $\psi_{CNN}(\cdot)$ represents an encoder formed by a CNN.

Deep Decoder

We model the network features using a deep neural network (DNN) represented by a nonlinear mapping $h : z_i \to q_i$ with shared parameters $\boldsymbol{\theta} = \{W^{(k)}, b^{(k)} : k = 1, 2, \ldots, h+1\}$ for the nodes $v_1, v_2, \ldots, v_n$. Here, $q_1, q_2, \ldots, q_n$ encode the features of the network, the parameter h represents the number of hidden layers, and $W^{(k)}$ and $b^{(k)}$ denote the weight matrix and bias vector of the k-th layer, respectively.

We set $q_i = (q_{i,1}, q_{i,2}, \ldots, q_{i,n})^T$ with $q_{i,j} = Prob(e_{ij} = 1)$ representing the probability of $e_{ij} = 1$. The representation of each layer in the deep decoder is given by

$$\begin{aligned} y_i^{(1)} &= \psi(W^{(1)} z_i + b^{(1)}), \\ y_i^{(k)} &= \psi(W^{(k)} y_i^{(k-1)} + b^{(k)}), \quad k = 2, \ldots, h+1, \end{aligned} \tag{5}$$

where $y_i^{(k)}$ is the output of the k-th layer, $y_i^{(h+1)} = q_i$, and $\psi(x) = 1/(1 + \exp(-x))$ is the sigmoid activation function.

The likelihood function of the deep decoder is given by

$$f(\boldsymbol{S}|\boldsymbol{z}, \boldsymbol{\theta}) = \prod_{v_i \in V} \prod_{j \neq i} \left(q_{i,j}^{\delta_{ij} e_{ij}} (1 - q_{i,j})^{\delta_{ij}(1-e_{ij})} \right), \tag{6}$$

where $\boldsymbol{z} = (z_1^T, z_2^T, \ldots, z_n^T)^T \in \mathbb{R}^{nd_z}$, $q_i = h(z_i; \boldsymbol{\theta})$, and $\delta_{ij} = 1$ if $j \in S_i$ and 0 otherwise.

Prior Distribution of the Embedding Vector

Motivated by the LINE method, we impose a prior distribution on the embedding vectors with a log-density function given by

$$\begin{aligned}\log \pi(\boldsymbol{z}) = & C + \lambda_1 \sum_{v_i \in V} \Big[\sum_{j \in S_i, e_{ij}=1} \log \psi(z_j^T \cdot z_i) \\ & + \lambda_2 \sum_{j \in S_i, e_{ij}=0} \log \psi(-z_j^T \cdot z_i) \Big],\end{aligned} \tag{7}$$

where $\psi(\cdot)$ represents the sigmoid function, C is a constant, and λ_1 and λ_2 are prior hyperparameters.

Sparsity Penalty on the Deep Decoder

Mathematically, the penalty function is defined as follows:

$$\rho(\boldsymbol{\theta}) = \sum_{g=1}^{p} \tilde{\lambda}_g \| W_g^{(1)} \|_2 + \sum_{j=1}^{h+1} \tilde{\lambda}_j^{\dagger} \| \tilde{W}^{(j)} \|_1. \quad (8)$$

Here, $W_g^{(1)} \in \mathbb{R}^{d_1}$ represents the g-th column of $W^{(1)}$, which corresponds to the weights from the g-th input neuron to the neurons in the first hidden layer, d_1 is the width of the first hidden layer, and $\tilde{W}^{(j)}$ encompasses both the weight matrix $W^{(j)}$ and bias vector $b^{(j)}$.
If $\tilde{\lambda}_g$ is set to 0 for $g = 1, 2, \ldots, p$, $\rho(\boldsymbol{\theta})$ is reduced to the Lasso penalty for all decoder parameters.

Training Algorithm

To simulate $\boldsymbol{z} = (z_1^T, z_2^T, \dots, z_n^T)^T \in \mathbb{R}^{nd_z}$ and estimate $\boldsymbol{\theta}$ simultaneously, we use the following adaptive stochastic gradient MCMC algorithm:

(a) Update $\boldsymbol{z}$ by simulating a sample from $\pi(\boldsymbol{z}|\boldsymbol{S},\boldsymbol{\theta}) \propto f(\boldsymbol{S}|\boldsymbol{z},\boldsymbol{\theta})\pi(\boldsymbol{z})$ using a stochastic gradient Markovian transition kernel. For example, the stochastic gradient Hamilton Monte Carlo (SGHMC) algorithm can be employed, and the update equations are as follows:

$$\begin{aligned}\boldsymbol{v}^{(k)} =&(1-\epsilon_k\eta)\boldsymbol{v}^{(k-1)} + \epsilon_k\nabla_{\boldsymbol{z}}(\log\pi(\boldsymbol{z}^{(k-1)}) + \log f(\boldsymbol{S}|\boldsymbol{z}^{(k-1)},\boldsymbol{\theta}^{(k-1)})) + \sqrt{2\epsilon_k\eta\tau}\boldsymbol{e}^{(k)},\\ \boldsymbol{z}^{(k)} =&\boldsymbol{z}^{(k-1)} + \epsilon_k\boldsymbol{v}^{(k-1)},\end{aligned}$$

where η represents the friction coefficient, τ denotes the temperature, k indexes iterations, ϵ_k is the learning rate, and $\boldsymbol{e}^{(k)} \sim N(0, I_{nd_z})$.

(b) Update $\boldsymbol{\theta}$ by setting

$$\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)} + \gamma_k\nabla_{\boldsymbol{\theta}}(\log f(\boldsymbol{S}|\boldsymbol{z}^{(k)},\boldsymbol{\theta}^{(k-1)}) + \lambda\rho(\boldsymbol{\theta}^{(k-1)})),$$

where γ_k is the step size, and λ is the regularization parameter for the sparsity penalty.

Dimension Selection

We compute the following gradient for each dimension j at each iteration of Algorithm 1 after fixing the value of λ:

$$G_j = \|\frac{1}{n}\sum_{i=1}^{n} g_{ij}\|_2, \quad j = 1, \ldots, d_z, \tag{9}$$

where $g_{ij} = (\frac{\partial q_{i,1}}{\partial z_{i,j}}, \ldots, \frac{\partial q_{i,n}}{\partial z_{i,j}})^T$ represents the jth column of the gradient matrix

$$\frac{\partial q_i}{\partial z_i} = \begin{bmatrix} \frac{\partial q_{i,1}}{\partial z_{i,1}} & \frac{\partial q_{i,1}}{\partial z_{i,2}} & \ldots & \frac{\partial q_{i,1}}{\partial z_{i,d_z}} \\ \frac{\partial q_{i,2}}{\partial z_{i,1}} & \frac{\partial q_{i,2}}{\partial z_{i,2}} & \ldots & \frac{\partial q_{i,2}}{\partial z_{i,d_z}} \\ \ldots & \ldots & \ddots & \ldots \\ \frac{\partial q_{i,n}}{\partial z_{i,1}} & \frac{\partial q_{i,n}}{\partial z_{i,2}} & \ldots & \frac{\partial q_{i,n}}{\partial z_{i,d_z}} \end{bmatrix}_{n \times d_z}.$$

A smaller gradient G_j indicates the dimension j is less significant compared to the other dimensions.

Theoretical Analysis

Theorem 1

Suppose regularity conditions hold. For almost every observed network, as $n \to \infty$ and $t \to \infty$, the following results hold:

(i) $\|\hat{\boldsymbol{\theta}}_n^{(t)} - \boldsymbol{\theta}^*\| \xrightarrow{p} 0$, *where* $\hat{\boldsymbol{\theta}}_n^{(t)}$ *denotes the estimate of* $\boldsymbol{\theta}^*$ *obtained at iteration t of Algorithm 1, and* $\xrightarrow{p}$ *denotes convergence in probability.*

(ii) $\hat{d}_{z,n}^{(t)} - d_z^* \xrightarrow{p} 0$, *where* $\hat{d}_{z,n}^{(t)} = \#\{j : \|G_j(\hat{\boldsymbol{\theta}}_n^{(t)}, z_n)\| > c\sqrt{r_n}\}$ *is the selected embedding dimension at iteration t, and $\{r_n\}$ is a sequence converging to 0 as specified in the appendix.*

Theoretical Analysis

Theorem 2

Suppose regularity conditions hold. Then for any integrable function $\phi(\cdot)$, $\widehat{\mathbb{E}_{\mathcal{K}}\phi(\boldsymbol{z})} - \mathbb{E}\phi(\boldsymbol{z}) \xrightarrow{p} 0$ *as* $\mathcal{K} \to \infty$.

Network Examples

- Karate: The social network consists of 34 members, who are divided into two known groups, in a university karate club. There are 78 edges in the network, where an undirected edge (i, j) represents a friendship tie between member i and member j.
- Dolphin: The network consists of 62 dolphins living in New Zealand. There are two communities and a total of 159 edges in the network, where an undirected edge (i, j) represents frequent contacts between dolphin i and dolphin j.

Node Clustering: Karate

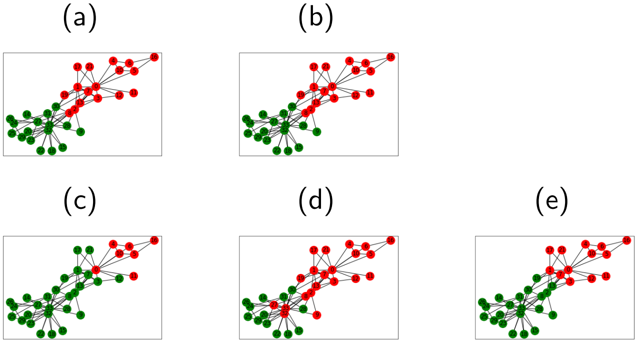

Figure 1: Node clustering results for the Karate network with 20-dimensional embedding vectors: (a) true clusters, (b) sparse decoder, (c) DNGR, (d) SNDE, and (e) Node2Vec.

Node Clustering: Dolphin

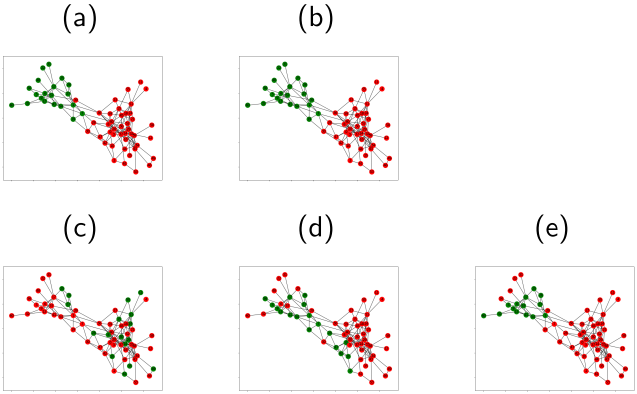

Figure 2: Node clustering results for the Dolphin network with 20-dimensional embedding vectors: (a) true clusters, (b) sparse decoder, (c) DNGR, (d) SNDE, and (e) Node2Vec.

Conclusion

- We have proposed a rigorous statistical framework for network embedding: we treat the embedding vectors as missing data, reconstruct network features with a sparse decoder, and simultaneously impute the embedding vectors and train the sparse decoder using an adaptive stochastic gradient MCMC algorithm.
- Under mild conditions, we show that the sparse decoder provides a parsimonious mapping from the embedding space to network features, which leads to an effective selection for the embedding dimension.
- This work lays down the first theoretical foundation for network embedding under the framework of missing data imputation. The proposed method serves as a prototype for deep network embedding, which highlights the importance of decoder sparsity and dimension selection, and enables uncertainty quantification in downstream statistical inference.

Reference

1. Tianning Dong, Yan Sun and Faming Liang (2024). Deep Network Embedding with Dimension Selection. *Neural Networks*, 179, 106512.
2. Liang, S., Sun, Y., and Liang, F. (2022). Nonlinear sufficient dimension reduction with a stochastic neural network. *NeurIPS 2022* (see also arXiv:2210.04349).