Deep Streaming Graph Representations

Minglong Lei[1], Yong Shi[2,3,4,5], Peijia Li[1], and Lingfeng Niu[2,3,4,*]

[1] School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing, 100049, China
[2] School of Economics and Management, University of Chinese Academy of Sciences, Beijing, 100190, China
[3] Key Laboratory of Big Data Mining and Knowledge Management, Chinese Academy of Sciences, Beijing 100190, China
[4] Research Center on Fictitious Economy & Data Science, Chinese Academy of Sciences, Beijing 100190, China
[5] College of Information Science and Technology, University of Nebraska at Omaha, NE 68182, USA

`leiminglong16@mails.ucas.ac.cn`, `yshi@ucas.ac.cn`, `lipeijia13@mails.ucas.ac.cn`, `niulf@ucas.ac.cn`

Abstract. Learning graph representations generally indicate mapping the vertices of a graph into a low-dimension space, in which the proximity of the original data can be preserved in the latent space. However, traditional methods that based on adjacent matrix suffered from high computational cost when encountering large graphs. In this paper, we propose a deep autoencoder driven streaming methods to learn low-dimensional representations for graphs. The proposed method process the graph as a data stream fulfilled by sampling strategy to avoid straight computation over the large adjacent matrix. Moreover, a graph regularized deep autoencoder is employed in the model to keep different aspects of proximity information. The regularized framework is able to improve the representation power of learned features during the learning process. We evaluate our method in clustering task by the features learned from our model. Experiments show that the proposed method achieves competitive results comparing with methods that directly apply deep models over the complete graphs.

Keywords: Streaming Methods · Representation Learning · Deep Autoencoder · Graph Regularization

1 Introduction

Graph representation or graph embedding[17] aims at mapping the vertices into a low-dimensional space while keeping the structural information and revealing the proximity of instances[17]. The compact representations for graph vertices is then useful for further tasks such as classification[10] and clustering[15, 8].

The most intuitive and simple idea to handle graph is only using the connection information and then representing the graph as a deterministic adjacent matrix. Dimension reduction techniques[12] that directly applied in the adjacent matrix can achieve superior performance in many cases.

Directly launch dimension reduction under the complete graph are efficient in many scenarios, they also have obvious disadvantages. Generally speaking, the limitations of direct matrix models are threefold. First, the direct matrix models are easily suffered from high computation complexity in large scale graphs. Since the adjacent matrix is deterministic and fixed, such methods are not flexible enough when the dataset is large. Second, the direct matrix models have not consider enough information in the model. They only provide a global view of the graph structure. However, the local information which depicts the neighborhood information should also be considered in the learned features. Finally, the success of the direct matrix models highly depend on the representation power of dimension reduction models. Methods such as spectral learning[12] and Non-negative Matrix Factorization[9] have limited representation power.

In order to solve those challenges, we propose a new deep graph representation method that based on streaming algorithm[1, 19]. The proposed method keeps the advantages of deterministic matrix methods and also introduces several new ideas to handle the limitations.

First, we introduce a streaming motivated stochastic idea into the model. Streaming methods are methods that process data streams. Specially, the input of a streaming model is organized as a sequence of data blocks. The main target of data streams is to solve memory issues. In this paper, we sample a small portion of vertices once and formulate a graph stream. With the accumulation of vertices along with the flow of data stream, more and more information will be automatically contained in the model rather in the data. Since we choose fixed small number of vertices for each time, the dimension of the input will be reduced significantly. Consequently, the streaming strategy is helpful in handling computation complexity issues.

Second, in order to combine more information in the model, we adopt a regularization framework in the proposed method. The direct matrix models only consider visible edges between vertices. The highlight point of the regularization framework is that the graph regularization term includes the vertex similarities in the model in addition to visible connections. Vertices that are similar in the original space should have similar representations in the latent low-dimensional space.

Finally, after the graph streams are obtained, we fed the data stream into a deep autoencoder[6] to learn the representations of graph vertices. The learning power of deep autoencoder assures that the learned features keep sufficient information from the original graph.

2 Related Work

Graph representation, also known as graph embedding, is a sub topic of representation learning. What representation learning[4] attempts to do is to decode data in the original space into a vector space in an unsupervised fashion so that the leaned features can be used in further tasks.

Early methods such as Laplacian Eigenmaps(LE)[3], Local Linear Embedding(LLE)[14] and are deterministic. Among those methods, the graph is denoted as an adjacent matrix and methods under the matrix is generally related to dimension reduction techniques[18]. Specially, the intuitive idea is to solve the eigenvectors of the affinity matrix. They exploit spectral properties of affinity matrices and are known as laplace methods. More recently, deep learning models are also used as dimension reduction tools for their superior ability in representaion[8, 15].

More recent works on graph embedding are stochastic in which the graph is no longer represented as a fixed matrix[13, 5]. Methods such as Deepwalk[13] and node2vec[5] regard the graph as a vertex vocabulary where a collection node sequences are sampled from. Subsequently, language models such as skip-gram[11] can be used to obtain the ultimate representations.

The deterministic methods are not flexible[2] and the disadvantages of s-tochastic models are also obvious. Since stochastic models only consider local information that describes the nearest neighbors of vertices, they fail in providing a global picture under the whole graph view. The loss of global information influences the performance of such models when the graph structure is irregular.

3 Network Embedding Problem

3.1 Notations

In this paper, we denote vectors as lowercase letters with bold form and matrixes as uppercase letters in boldface. The elements of a matrix and a vector are denoted as $\mathbf{X}_{ij}$ and $\mathbf{x}_i$ respectively. Given a graph $G(V, E)$, V is the vertices set denoted as $\{v_1, ..., v_n\}$ and E is the edges set denoted as $\{v_{ij}\}_{i,j=1}^{n}$.

We then define the graph embedding as:

Definition 1. *(Graph Embedding) Given a N-vertex graph $G(V, E)$, the goal of graph embedding is to learn a mapping $v_i \longmapsto \mathbf{y}_i$, $\mathbf{y}_i \in \mathbb{R}^d$. The learned representations in the latent low-dimensional space should be capable to keep the structural information of the original graph.*

3.2 Streaming Strategy

In this subsection, we illustrate how to formulate the data stream from a given graph $G(V, E)$. Let K be the number of data chunks in a data stream. Denote S_k as the k^{th} data chunk in the data stream where $k \in 1, 2, \cdots, K$. The K can be extremely large since the substantial numbers of samplings is conducive to visiting the graph completely.

Let the number of vertices that selected in one time to be $D(D \ll N)$. Obviously, the D is also the input of the embedding model since D is fixed as a constant number. In the training phase, in an arbitrary step k, we select D nodes

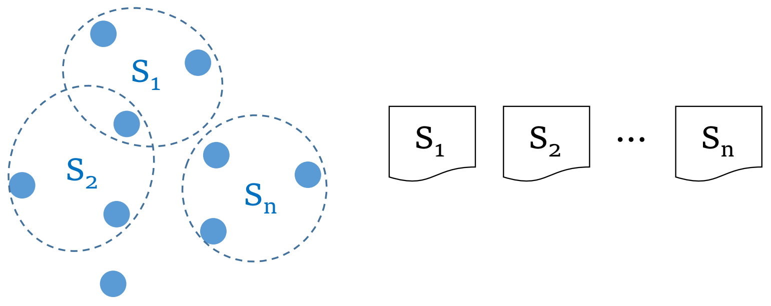

Fig. 1. The streaming strategy. Each time we choose a fixed number of vertices. The data chunks are constructed by the selected vertices.

from the vertex collection $\{v_1, ..., v_n\}$ uniformly. A subgraph is then constructed by the selected nodes. The S_k is the adjacent matrix of the subgraph. In Fig. 1 we present the sampling process to formulate a data stream. A data stream S is denoted as $S = S_k; k \in 1, \cdots, K$.

In the embedding phase, the goal is mapping each vertex to its representations by the trained model. However, the dimension of the original data N is much higher than the input dimension of the model D. Consequently, we run a simple Principal Component Analysis(PCA) in $\mathbf{X}$ to get $\mathbf{X}^D$ with a dimension D. Then the $\mathbf{X}^D$ is served as input to obtain the compact representations for each vertex.

3.3 Graph Autoendoer

Autoencoders[16] is powerful in representation task. After getting the data stream, we use a deep graph autoencoder to get the low-dimension vectors.

Deep Autoencoder: Autoencoder paradigm attempts to copy its input to its output, which results in a code layer that may capture useful properties of the input.

Let $\mathbf{X} = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^{m\times1}\}_{i=1}^n$ and $\mathbf{Z} = \{\mathbf{z}_i : \mathbf{z}_i \in \mathbb{R}^{m\times1}\}_{i=1}^n$ be the input matrix and reconstruction matrix. $\mathbf{Y} = \{\mathbf{y}_i : \mathbf{y}_i \in \mathbb{R}^{d\times1}\}_{i=1}^n$ is the code matrix where the dimension of $\mathbf{y}_i$ is usually much lower than the dimension of the original data $\mathbf{x}_i$. A layer wise interpretation of the encoder and decoder can be represented as:

$$\mathbf{Y} = f_\theta(\mathbf{X}) = \delta(W_{encoder}\mathbf{X} + b_{encoder}) \tag{1}$$

$$\mathbf{Z} = g_\theta(\mathbf{Y}) = \delta(W_{decoder}\mathbf{Y} + b_{decoder}) \tag{2}$$

For convenience, we summarize the encoder parameters as $\theta_{encoder}$, and the decoder parameters as $\theta_{decoder}$. Then the loss function can be defined as:

$$\mathcal{L} = \|\mathbf{X} - \mathbf{Z}\|_F^2 = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{z}_i\|_2^2 \tag{3}$$

Since a deep autoencoder can be thought of as a special case of feedforward networks, the parameters are optimized by backpropagate gradients through chain-rules.

Graph regularization: In order to preserve the local structure of the data, we employ a graph regularization term derived from Laplacian Eigenmaps[3]. Suppose A is the indicator matrix where $\mathbf{A}_{ij}$ indicate if node i and node j are connected, the laplacian loss is then defined as:

$$Laplacian = \sum_{i}^{n}\sum_{j}^{n} \mathbf{A}_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 \tag{4}$$

The laplacian loss can be further written as:

$$Laplacian = \sum_{i}^{n}\sum_{j}^{n} A_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = 2tr(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) \tag{5}$$

where $tr(*)$ denotes the trace and $\mathbf{L}$ is the laplace matrix calculated by matrix $\mathbf{A} : \mathbf{L} = \mathbf{D} - \mathbf{A}$. $\mathbf{D}$ is a diagonal matrix where $\mathbf{D} = \sum_i^n \mathbf{A}_{ij} = \sum_i^n \mathbf{A}_{ij}$.

Combining the graph information, the optimization problem is:

$$\mathcal{L} = \|\mathbf{X} - \mathbf{Z}\|_F^2 + \alpha' \cdot 2tr(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) + \beta' \cdot \frac{1}{2}\|W\|_F^2 \tag{6}$$

Merge the constant numbers into parameters α and β, the loss function is updated as:

$$\mathcal{L} = \|\mathbf{X} - \mathbf{Z}\|_F^2 + \alpha tr(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) + \beta\|W\|_F^2 \tag{7}$$

where α and β are the hyperparameters that control the model complexity.

Recall that each time we have a data chunk S_k, let $\mathbf{X} = \mathbf{A} = S_k$ and then run the graph regularized autoencoder under $\mathbf{X}$ and $\mathbf{A}$. Similar to most deep neural networks, we choose gradient decent to optimize the deep autoencoder. The objective function is $\mathcal{L} = \varepsilon(f,g) + \lambda\Omega(f)$. The first term $\varepsilon(f,g)$ is the reconstruction error and the second term $\Omega(f)$ is the regularization term. The partial derivatives of $\theta_{decoder}$ only depend on the first term and the partial derivatives of $\theta_{encoder}$ depend on both terms. By using chain rules, parameters at each layer can be calculated sequentially.

4 Experiments

In this section, we conduct experiments in clustering tasks to testify the effectiveness of our method.

We use two datasets, COIL20 and ORL, to testify the our efficiency. COIL20 contains 1440 instances that belongs to 20 categories and ORL contains 400 samples that belongs to 40 classes. The KNN-graph is constructed by computing the k-nearest neighbors of each sample.

We compare our approach with several deep models to evaluate the performance of our method. Specially, we employ deep autoencoder(DAE)[7] and stacked autoencoder(SAE)[16] as baseline models.

We evaluate the learned features in clustering task. Following the general settings in most clustering procedure, we employ *purity* and NMI to evaluate the results.

In our experiment, we set the D to be 500 for COIL20 and 200 for ORL. The layers of deep graph autoencoders for COIL20 and ORL are 5 and 3 respectively. For COIL20, we set the dimensions as $500-200-100-200-500$. For ORL, we set the dimensions as $200-100-200$.

The clustering results of COIL20 and ORL are presented in Table 1. The results show that the streaming method has competitive representation power comparing with baseline models that utilize the complete matrices. The results also indicate that when encountering large graphs, the streaming method is relieved from computation issues and is still able to achieve superior performance.

Table 1. Results in Clustering Task

	COIL20		ORL	
Methods	NMI	Purity	NMI	Purity
Deep Streaming + Kmeans	0.7887	0.7124	0.8425	0.7325
DAE + Kmeans	**0.8034**	**0.7241**	**0.8672**	**0.7416**
SAE + Kmeans	0.7604	0.6925	0.8390	0.7175
Kmeans	0.7301	0.6535	0.8247	0.7024

5 Conclusion

We proposed a streaming motivated embedding method to learn the low dimensional representations of the graph. The streaming strategy is used to reduce the effect of computation complexity. The deep autoencoder and graph regularization idea make sure the learned features include enough information. Experiments in clustering task verify the effectiveness of our methods. Our model achieve results as good as models that directly apply dimension reduction in the original matrix. The results can be generalized to large graphs where directly matrix models are inapplicable.

Acknowledgements

This work was supported by the National Natural Science Foundation of China(Grant No. 91546201, No. 71331005, No. 71110107026, No. 11671379, No. 11331012), UCAS Grant(No. Y55202LY00).

References

1. Aggarwal, C.C.: Data streams: models and algorithms, vol. 31. Springer Science & Business Media (2007)
2. Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., Smola, A.J.: Distributed large-scale natural graph factorization. In: Proceedings of the 22nd international conference on World Wide Web. pp. 37–48. ACM (2013)
3. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: NIPS. vol. 14, pp. 585–591 (2001)
4. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
5. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864. ACM (2016)
6. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18**(7), 1527–1554 (2006)
7. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786), 504–507 (2006)
8. Huang, P., Huang, Y., Wang, W., Wang, L.: Deep embedding network for clustering. In: Pattern Recognition (ICPR), 2014 22nd International Conference on. pp. 1532–1537. IEEE (2014)
9. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755), 788 (1999)
10. Lu, Q., Getoor, L.: Link-based classification. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 496–503 (2003)
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
12. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Advances in neural information processing systems. pp. 849–856 (2002)
13. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
14. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. science **290**(5500), 2323–2326 (2000)
15. Tian, F., Gao, B., Cui, Q., Chen, E., Liu, T.Y.: Learning deep representations for graph clustering. In: AAAI. pp. 1293–1299 (2014)
16. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research **11**(Dec), 3371–3408 (2010)
17. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234. ACM (2016)
18. Yan, S., Xu, D., Zhang, B., Zhang, H.J., Yang, Q., Lin, S.: Graph embedding and extensions: A general framework for dimensionality reduction. IEEE transactions on pattern analysis and machine intelligence **29**(1), 40–51 (2007)
19. Zhang, P., Zhu, X., Guo, L.: Mining data streams with labeled and unlabeled training examples. In: Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. pp. 627–636. IEEE (2009)