Accepted Manuscript - Version of record at: https://doi.org/10.1109/TPAMI.2022.3225073

One-Hot Graph Encoder Embedding

Cencheng Shen, Qizhe Wang, Carey E. Priebe

Abstract—In this paper we propose a lightning fast graph embedding method called one-hot graph encoder embedding. It has a linear computational complexity and the capacity to process billions of edges within minutes on standard PC — making it an ideal candidate for huge graph processing. It is applicable to either adjacency matrix or graph Laplacian, and can be viewed as a transformation of the spectral embedding. Under random graph models, the graph encoder embedding is approximately normally distributed per vertex, and asymptotically converges to its mean. We showcase three applications: vertex classification, vertex clustering, and graph bootstrap. In every case, the graph encoder embedding exhibits unrivalled computational advantages.

Index Terms—Graph Embedding, One-Hot Encoding, Central Limit Theorem, Community Detection, Vertex Classification

1 Introduction

Graph data arises naturally in modern data collection and captures interactions among objects. Given n vertices and s edges, a graph can be represented by an $n \times n$ adjacency matrix $\mathbf{A}$ where $\mathbf{A}(i,j)$ is the edge weight between ith vertex and jth vertex. In practice, a graph is typically stored by an $s \times 3$ edgelist $\mathbf{E}$, where the first two columns store the vertex indices of each edge and the last column is the edge weight. Examples include social networks, brain regions, article hyperlinks [1], [2], [3], [4], [5], etc. A graph data has community structure if the vertices can be grouped into different classes based on the edge connectivity [1]. In case of supervised learning, some vertices come with ground-truth labels and serve as the training data; while in case of unsupervised learning, the graph data has no known label.

To better explore and analyze graph data, graph embedding is a very popular approach, which learns a low-dimensional Euclidean representation of each vertex. The spectral embedding method [6], [7], [8], [9], [10], [11], [12] is a well-studied method in the statistics literature. By using singular value decomposition (SVD) on graph adjacency or graph Laplacian, the resulting vertex embedding asymptotically converges to the latent positions under random dot product graphs [13], [14], thus consistent for subsequent inference tasks like hypothesis testing and community detection. Other popular approaches include Deepwalk [15], node2vec [16], [17], graph convolutional network (GCN) [18], which empirically work well on real graphs. However, existing methods require tuning parameters, are computationally expensive, and do not scale well to big graphs. As modern social networks easily produce billions of edges, a more scalable and elegant solution is direly needed.

Towards that target, we propose the one-hot graph encoder embedding (GEE) in this paper. The method is straightforward to implement in any programming language, has a linear computational complexity and storage requirement, is applicable to either the adjacency matrix or graph Laplacian, and is capable of processing billions of edges within minutes on a standard PC. Theoretically, the graph encoder embedding enjoys similar properties as the spectral embedding, is approximately normally distributed, and converges to a transformation of the latent positions under random graph models. We showcase three applications: vertex classification, vertex clustering, and graph bootstrap. Comprehensive experiments on synthetic and real graphs are carried out to demonstrate its excellent performance. All proofs and simulation details are in the Appendix. The MATLAB, Python, and R code are made available on Github[1].

2 Method

Graph Encoder Embedding

Algorithm 1 presents the pseudo-code for encoder embedding when all or partial vertex labels are available. The inputs consist of an edgelist $\mathbf{E}$ and a label vector $\mathbf{Y}$ of K classes. We assume the known labels lie in $\{1, \ldots, K\}$ and unknown labels are set to 0 (or any negative number suffices). The final embedding is denoted by $\mathbf{Z}$, where $\mathbf{Z}_i$ (the ith row) is the embedding of the ith vertex.

The algorithm is applicable to any graph, including directed or weighted graphs. It is also applicable to the graph Laplacian: given any edgelist, one can compute the degree coefficient for each vertex, then replace the edge weight by the degree-normalized weight. This can be achieved via iterating through the edgelist just twice (not shown in Algorithm 1 but implemented in our codebase).

- *Cencheng Shen and Qizhe Wang are with the Department of Applied Economics and Statistics, University of Delaware. E-mail: shenc@udel.edu, qizhew@udel.edu*
- *Carey E.Priebe is with the Department of Applied Mathematics and Statistics (AMS), the Center for Imaging Science (CIS), and the Mathematical Institute for Data Science (MINDS), Johns Hopkins University. E-mail: cep@jhu.edu*

This work was supported in part by the Defense Advanced Research Projects Agency under the D3M program administered through contract FA8750-17-2-0112, the National Science Foundation HDR TRIPODS 1934979, the National Science Foundation DMS-2113099, the University of Delaware Data Science Institute Seed Funding Grant, and by funding from Microsoft Research. We thank the editor and reviewers for their excellent suggestions to improve the paper. We thank Jonathan Larson and Ha Trinh from Microsoft Research for test running our code.

1. https://github.com/cshen6/GraphEmd

Accepted Manuscript - Version of record at: https://doi.org/10.1109/TPAMI.2022.3225073

Algorithm 1 Graph Encoder Embedding

Require: An edgelist $\mathbf{E} \in \mathbb{R}^{s\times 3}$, and the corresponding class label vector $\mathbf{Y} \in \{0,\dots,K\}^n$.

Ensure: The encoder embedding $\mathbf{Z} \in \mathbb{R}^{n\times K}$, and the transformation matrix $\mathbf{W} \in \mathbb{R}^{n\times K}$.

function GEE($\mathbf{E}, \mathbf{Y}$)
  $\mathbf{W} = \text{zeros}(n, K)$; ▷ initialize the matrix
  $\mathbf{Z} = \text{zeros}(n, K)$;
  for $k = 1,\dots,K$ **do**
    $ind = \text{find}(\mathbf{Y} = k)$; ▷ find indices of class k
    $n_k = \text{sum}(ind)$;
    $\mathbf{W}(ind, k) = \frac{1}{n_k}$;
  end for
  for $i = 1,\dots,s$ **do**
    $\mathbf{Z}(\mathbf{E}(i,1), \mathbf{Y}(\mathbf{E}(i,2))) = \mathbf{Z}(\mathbf{E}(i,1), \mathbf{Y}(\mathbf{E}(i,2))) + \mathbf{W}(\mathbf{E}(i,2), \mathbf{Y}(\mathbf{E}(i,2))) * \mathbf{E}(i,3)$;
    $\mathbf{Z}(\mathbf{E}(i,2), \mathbf{Y}(\mathbf{E}(i,1))) = \mathbf{Z}(\mathbf{E}(i,2), \mathbf{Y}(\mathbf{E}(i,1))) + \mathbf{W}(\mathbf{E}(i,1), \mathbf{Y}(\mathbf{E}(i,1))) * \mathbf{E}(i,3)$;
  end for
end function

Since n_k represents the number of vertices in each class, the matrix $\mathbf{W}$ equals the one-hot encoding of the label vector then column-normalized by n_k. In matrix notation, the encoder embedding can be succinctly expressed by $\mathbf{Z} = \mathbf{AW}$, or $\mathbf{Z} = \mathbf{D}^{-0.5}\mathbf{AD}^{-0.5}\mathbf{W}$ for graph Laplacian ($\mathbf{D}$ is the $n \times n$ diagonal matrix of degrees).

In the one-hot graph encoder embedding, each class label of the graph vertex is assigned its own variable in the final embedding. We shall call the adjacency version as the adjacency encoder embedding (AEE), and the Laplacian version as the Laplacian encoder embedding (LEE). They may be viewed as a transformation of the adjacency / Laplacian spectral embedding (ASE / LSE), each with its unique property as summarized in [12]. Note that Algorithm 1 assumes partial known labels and is a natural set-up for vertex classification, which is evaluated in-depth in Section 5. The unsupervised GEE (no known label) is presented in Algorithm 2 and evaluated in Section 6.

Computational Advantages

Algorithm 1 has a time complexity and storage requirement of $O(nK + s)$, thus is linear with respect to the number of vertices and number of edges. Because it iterates the input data only once with a few operations, it is extremely efficient in any programming language. The running time advantage is demonstrated in Figure 1. On a standard PC with 12-core CPU and 64GB memory and MATLAB 2022a, it takes a mere 6 seconds to process 10 million edges, one minute for 100 million edges with 1 million vertices, and 10 minutes for 1 billion edges with 10 million vertices. In comparison, other methods are order of magnitude slower and cannot handle more than 10 million edges on the same PC.

3 Theorems

To better understand graph encoder embedding, we first review three popular random graph models, then present the asymptotic properties under each model. Throughout this section, we assume n is the number of vertices with known labels; and when $n \to \infty$, so is $n_k \to \infty$ for each $k \in \{1,\dots,K\}$.

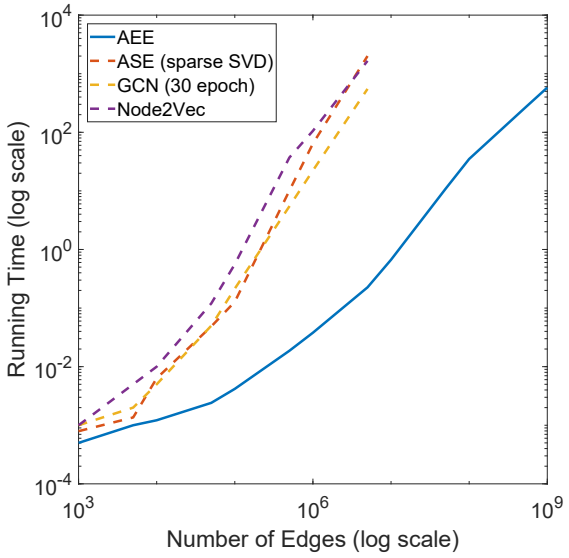


Fig. 1. We report the average running time of graph encoder embedding using 50 Monte Carlo replicates, on a random graph with $K = 10$, average degree 100, and increasing graph size. The number of edges increases from one thousand to one billion. At 1 billion edges with 10 million vertices, the encoder embedding only requires 20GB memory and finishes in 10 minutes. All other methods exceed maximum memory capacity at 10 million edges. More details on the methods compared can be found in Section 5.

Stochastic Block Model (SBM)

SBM is arguably the most fundamental community-based random graph model [13], [19], [20], [21]. Each vertex i is associated with a class label $Y_i \in \{1,\dots,K\}$. The class label may be fixed a-priori, or generated by a categorical distribution with prior probability $\{\pi_k \in (0,1) \text{ with } \sum_{k=1}^K \pi_k = 1\}$. Then a block probability matrix $\mathbf{B} = [\mathbf{B}(k,l)] \in [0,1]^{K\times K}$ specifies the edge probability between a vertex from class k and a vertex from class l: for any $i < j$,

$$\mathbf{A}(i,j) \overset{i.i.d.}{\sim} \text{Bernoulli}(\mathbf{B}(Y_i, Y_j)),$$
$$\mathbf{A}(i,i) = 0, \quad \mathbf{A}(j,i) = \mathbf{A}(i,j).$$

Degree-Corrected Stochastic Block Model (DC-SBM)

The DC-SBM graph is a generalization of SBM to better model the sparsity of real graphs [22]. Everything else being the same as SBM, each vertex i has an additional degree parameter θ_i, and the adjacency matrix is generated by

$$\mathbf{A}(i,j) \sim \text{Bernoulli}(\theta_i\theta_j\mathbf{B}(Y_i, Y_j)).$$

The degree parameters typically require certain constraint to ensure a valid probability. In this paper we simply assume they are non-trivial and bounded, i.e., $\theta_i \overset{i.i.d.}{\sim} F_\theta \in (0, M]$, which is a very general assumption.

Random Dot Product Graph (RDPG)

Another random graph model is RDPG [14]. Under RDPG, each vertex i is associated with a latent position vector $X_i \overset{i.i.d.}{\sim} F_X \in [0,1]^p$. F_X is constrained such that $X_i^T X_j \in (0,1]$, i.e., the inner product shall be a valid probability. Then the adjacency matrix is generated by

$$\mathbf{A}(i,j) \sim \text{Bernoulli}(X_i^T X_j).$$

To generate communities under RDPG, it suffices to use a K-component mixture distribution, i.e., let $(X_i, Y_i) \overset{i.i.d.}{\sim} F_{XY}$ be a distribution on $\mathbb{R}^p \times [K]$.

Asymptotic Normality

Under these random graph models, we prove the central limit theorem for the graph encoder embedding. Namely, the vertex embedding is asymptotically normally distributed per vertex. Since the mean and covariance differ under each model, we introduce some additional notations:

- Denote $\vec{n} = [n_1, n_2, \cdots, n_k] \in \mathbb{R}^K$, and $Diag(\cdot)$ as the diagonal matrix of a vector.
- Under SBM with block matrix $\mathbf{B}$, define $\Sigma_{\mathbf{B}_y}$ as the $K \times K$ diagonal matrix with

$$\Sigma_{\mathbf{B}_y}(k,k) = \mathbf{B}(y,k)(1 - \mathbf{B}(y,k)) \in [0, \frac{1}{4}].$$

- Under DC-SBM with $\{\theta_j \overset{i.i.d.}{\sim} F_\theta\}$, for any tth moment we define:

$$\bar{\theta}_k^{(t)} = E(\theta_j^t | Y_j = k),$$
$$\bar{\Theta}^{(t)} = [\bar{\theta}_{(1)}^{(t)}, \bar{\theta}_{(2)}^{(t)}, \cdots, \bar{\theta}_{(K)}^{(t)}] \in \mathbb{R}^K.$$

- Under RDPG where $(X, Y) \sim F_{XY} \in \mathbb{R}^p \times [K]$ is the latent distribution, define

$$\bar{\lambda}_k^{(t)}(x_i) = E^t(X^T x_i | Y = k),$$
$$\bar{\lambda}_{x_i}^{(t)} = [\bar{\lambda}_1^{(t)}(x_i), \bar{\lambda}_2^{(t)}(x_i), \cdots, \bar{\lambda}_K^{(t)}(x_i)] \in \mathbb{R}^K$$

for any fixed vector $x_i \in \mathbb{R}^p$.

Theorem 1. *The graph encoder embedding is asymptotically normally distributed under SBM, DC-SBM, or RDPG. Specifically, as n increases, for a given ith vertex of class y it holds that*

$$Diag(\vec{n})^{0.5} \cdot (\mathbf{Z}_i - \mu) \xrightarrow{d} \mathcal{N}(0, \Sigma).$$

The expectation and covariance are:

- *under SBM, $\mu = \mathbf{B}(y,:)$ and $\Sigma = \Sigma_{\mathbf{B}_y}$;*
- *under DC-SBM, $\mu = \theta_i \mathbf{B}(y,:) \odot \bar{\Theta}^{(1)}$ and $\Sigma = \theta_i^2 Diag(\bar{\Theta}^{(2)}) \cdot \Sigma_{\mathbf{B}_y}$;*
- *under RDPG, $\mu = \bar{\lambda}_{x_i}^{(1)}$ and $\Sigma = Diag(\bar{\lambda}_{x_i}^{(1)} - \bar{\lambda}_{x_i}^{(2)})$.*

Asymptotic Convergence

The law of large numbers immediately follows. Namely, as the number of vertices increase, the graph encoder embedding converges to the mean.

Corollary 1. *Using the same notation as in Theorem 1. It always holds that*

$$\|\mathbf{Z}_i - \mu\|_2 \xrightarrow{n \to \infty} 0.$$

As SBM, DC-SBM, and RDPG are the most common graph models, in this paper we choose to express the mean via model parameters. Alternatively, the mean can be expressed more generally by conditional expectations, i.e., for each dimension it holds that $\mathbf{Z}_i[k] \to E(\mathbf{A}_{ij} | Y_j = k)$, which estimates the probability of vertex i being adjacent to a random vertex from class k.

While the spectral embedding estimates the block probability or latent variable up-to rotation [6], [7], the encoder embedding is more informative and interpretable due to the elimination of the rotational non-identifiability. See Figure 2 - 3 for numerical examples. Finally, the asymptotic normality and asymptotic convergence also hold for weighted graphs, which is discussed in the proof section.

4 Embedding Visualization

Simulated Graphs

Figure 2 compares the graph encoder embedding to the spectral embedding under SBM, DC-SBM, and RDPG graphs at $K = 2$. While both methods exhibit clear community separation, the encoder embedding provides better estimation for the model parameters. For example, under the SBM graph, the encoder embedding clearly estimates the block probability vectors $(0.13, 0.1)$ and $(0.1, 0.13)$ and appears normally distributed within each class; and under the DC-SBM graph, the encoder embedding lies along the block probability vectors multiplied by the degree of each vertex. A normality visualization for the same simulations are provided in the Appendix.

Real Graphs

Figure 3 illustrates graph encoder embedding for the Political Blogs [2] (1490 vertices with 2 classes) and the Gene Network [23] (1103 vertices with 2 classes). Both graphs are sparse. The average degree is 22.4 for the Political Blogs and 1.5 for the Gene Network. We observe that the vertex embedding appears similar to DC-SBM, which lies along a line for each class. Within-class vertices are better connected than between-class vertices, and different communities are well-separated except a few outliers.

5 Vertex Classification

An immediate and important use case herein is vertex classification. The vertex embedding with known class labels are the training data (labels with class 1 to K), while the vertex embedding with unknown labels are the testing data (labels set to 0 in Algorithm 1). We consider five graph embedding methods: adjacency encoder embedding (AEE), Laplacian encoder embedding (LEE), adjacency spectral embedding (ASE), Laplacian spectral embedding (LSE), and node2vec. For ASE and LSE we used the sparse SVD (the fastest SVD implementation in MATLAB) with 20 eigenvalues, then report the best accuracy and the running time among $d = 1, \ldots, 20$. For node2vec, we use the fastest available PecanPy implementation [17] with all default parameters and window size 2. For every embedding, we use linear discriminant analysis (LDA) and 5-nearest-neighbor (5NN) as the follow-on classifiers. Other classifier like logistic

Accepted Manuscript - Version of record at: https://doi.org/10.1109/TPAMI.2022.3225073

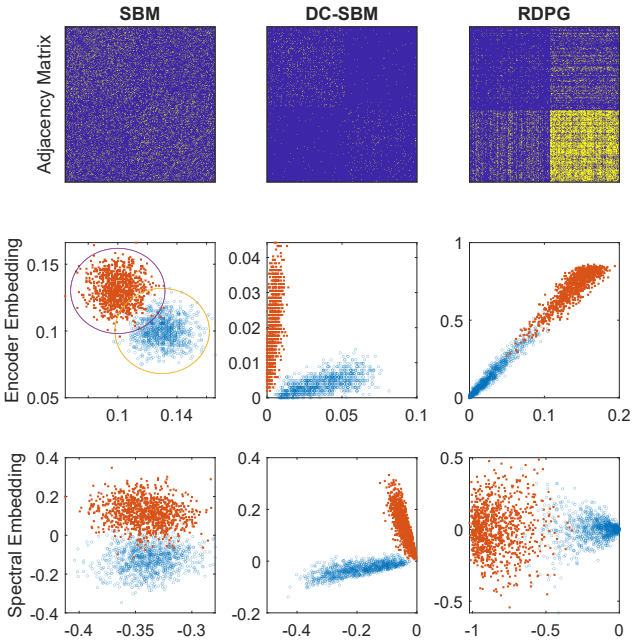


Fig. 2. Visualizing the vertex embedding: the top row is the graph adjacency heatmap (the index are ordered based on class labels), the middle row is the graph encoder embedding, and the bottom row is the adjacency spectral embedding at $d = 2$. Each graph is generated by SBM, DC-SBM, and RDPG from left column to right column at $n = 2000$, with parameter details presented in the Appendix. In each panel, the red dots denote the vertex embedding of class 1, and blue dots denote the vertex embedding of class 2.

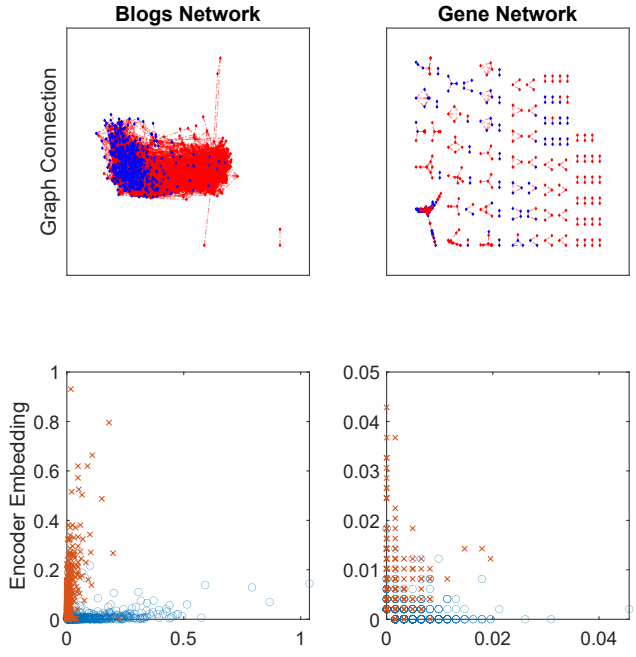


Fig. 3. Visualizing the vertex embedding for the Political Blogs and Gene Network: the top row plots the graph connectivity via MATLAB graph plot function, and the bottom row is the graph encoder embedding. Red denotes class 1 vertices and blue denotes class 2 vertices.

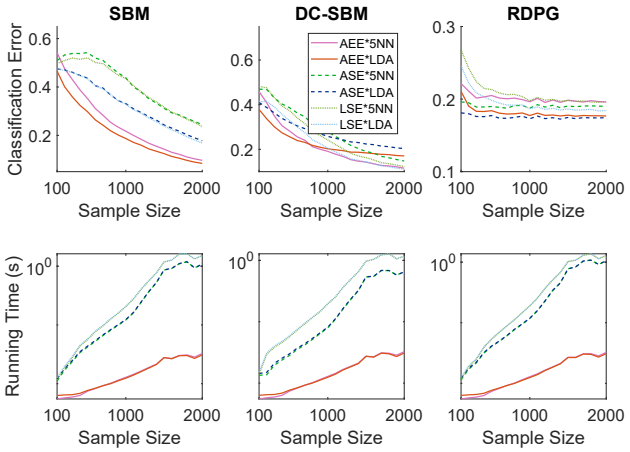


Fig. 4. Comparing the classification error (top row) and running time (bottom row in log scale) for SBM, DC-SBM, and RDPG graph with increasing n. Parameter details can be found in the Appendix.

regression, random forest, and neural network can also be used. We observe similar accuracy regardless of the classifiers, implying that the learning task largely depends on the embedding method.

Classification Evaluation on Synthetic Data

Figure 4 shows the average 10-fold classification error and average running time under simulated SBM, DC-SBM, and RDPG graphs ($K = 3$). For better clarity, only AEE, ASE, and LSE are included in the figure since they are the best performers on synthetic data. The standard deviation for the classification error is about 2% for each method, while the standard deviation for the running time is at most 10%. As the number of vertices increases, every method has better classification error at the cost of more running time. The encoder embedding has the lowest classification error under SBM, is among the lowest under DC-SBM and RDPG, and has the best running time.

Classification Evaluation on Real Graphs

We downloaded a variety of public real graphs with labels, including three graphs from network repository[2] [23]: Cora Citations (2708 vertices, 5429 edges, 7 classes), Gene Network (1103 vertices, 1672 edges, 2 classes), Industry Partnerships (219 vertices, 630 edges, 3 classes); and three more graphs from Stanford network data[3]: EU Email Network [24] (1005 vertices, 25571 edges, 42 classes), LastFM Asia Social Network [25] (7624 vertices, 27806 edges, 17 classes), and Political Blogs [2] (1490 vertices, 33433 edges, 2 classes).

For each data and each method, we carried out 10-fold validation and report the average classification error and running time in Table 1. For ease of presentation, we report the lower error between 5NN and LDA classifiers for each embedding. Comparing to the corresponding spectral embedding or node2vec, the encoder embedding achieves similar or better performance with trivial running time.

2. https://networkrepository.com/index.php
3. https://snap.stanford.edu/

Accepted Manuscript - Version of record at: https://doi.org/10.1109/TPAMI.2022.3225073

Classification Error						
	AEE	LEE	ASE	LSE	N2v	*
Cora	16.3%	**15.5%**	31.0%	33.1%	16.3%	69.8%
Email	30.6%	28.3%	30.8%	39.5%	**26.1%**	89.2%
Gene	17.1%	**16.5%**	27.2%	36.2%	21.9%	44.4%
Industry	**29.7%**	30.7%	38.8%	39.2%	32.9%	39.3%
LastFM	15.5%	15.0%	20.1%	16.5%	**14.5%**	79.4%
PolBlog	4.9%	5.0%	5.5%	**4.0%**	4.5%	48.0%
Running Time (seconds)						
	AEE	LEE	ASE	LSE	N2v	
Cora	**0.01**	**0.01**	1.55	1.60	2.1	
Email	**0.02**	0.03	0.12	0.15	1.2	
Gene	**0.01**	**0.01**	0.15	0.18	0.80	
Industry	**0.01**	**0.01**	0.02	0.02	0.25	
LastFM	**0.02**	0.03	13.0	15.3	9.2	
PolBlog	**0.01**	0.02	0.27	0.28	1.2	

TABLE 1
Comparing the embedding performance on real graphs. For each graph, the lowest classification error and running time are highlighted in bold. N2v stands for node2vec, and the last column shows the chance error. Note that the running time only includes the embedding step.

Node2vec also performs well on real data but takes significantly longer.

6 No Label and Vertex Clustering

Many graph data are collected without ground-truth vertex labels. Therefore, we also design an unsupervised graph encoder embedding in Algorithm 2. Starting with random label initialization, we utilize Algorithm 1 and k-means clustering to iteratively refine the vertex embedding and label assignments. The algorithm stops when the labels no longer change or the maximum iteration limit is reached.

The running time is $O(M(nK^2+s))$, which is still linear with respect to the number of edges and the number of vertices. In our experiments we set the maximum iteration limits to $r = 30$, which always achieve satisfactory performance.

Note that spectral embedding and node2vec are unsupervised in nature (though they do not utilize labels even when available). The clustering performance is measured by the adjusted rand index (ARI) between the clustering results and ground-truth labels. ARI lies in $(-\infty, 1]$, with larger positive number implying better matchedness and 1 for perfect match [26].

As long the graph is not too small, Algorithm 2 performs well throughout our experiments. Figure 5 provides an illustration of the clustering performance under 3-class SBM and RDPG graphs. The adjacency encoder embedding yields excellent ARI, which is similar to ASE clustering but much faster. The advantage is consistent throughout the synthetic and real graphs. Table 2 presents the clustering results for all the real data in Table 1. Comparing to Table 1, the unsupervised algorithm typically takes $2-10$ times longer than the with-label version. It is still vastly superior than other methods in the running time, while maintaining excellent ARI. The only exception is the Gene graph, which is too sparse for any clustering method.

Algorithm 2 Graph Encoder Embedding Without Label

Require: An edgelist $\mathbf{E}$, number of clusters K, and iteration limit r.

Ensure: The encoder embedding $\mathbf{Z} \in \mathbb{R}^{n\times K}$ for all vertices, and the estimated class label $\mathbf{Y} \in \{1,\ldots,K\}^n$.

```
function GEE UNSUP(E, K, M)
    Y_new = random(K, n);      ▷ randomize a label vector
    for i=1,...,M do
        Z = GEE(E, Y_new);
        Y = kmeans(Z, K);
        if ARI(Y_new, Y) == 1 then
            Stop;
        else
            Y_new = Y;
        end if
    end for
end function
```

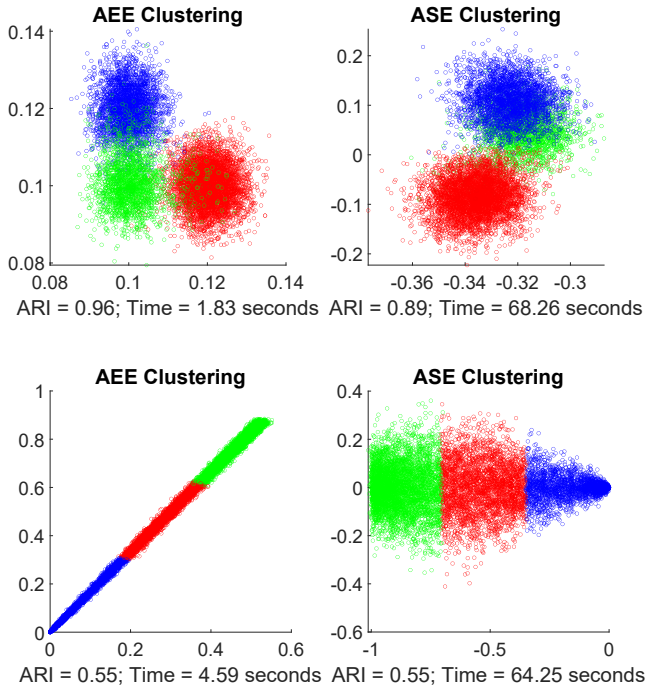


Fig. 5. The top row visualizes unsupervised AEE and ASE for an SBM graph, while the bottom row compares AEE and ASE for a RDPG graph. Those graphs are generated by the same three-class SBM and RDPG in Figure 4 at $n = 10000$. Blue, red, and green dots denote vertices of different classes. Note that the embedding dimension is 3 while we visualized the embedding of the first two dimensions.

7 Graph Bootstrap

Bootstrap is a popular statistical method for resampling Euclidean data [27], and there has been some investigations on graph bootstrap [28], [29]. A naive graph bootstrap procedure can be carried out as follows: simply resample the vertex index with replacement, then re-index both the row and column of the adjacency matrix.

Since the graph encoder embedding offers a good estimate of the block probability, it also provides an elegant graph bootstrap solution as detailed in Algorithm 3. Given a graph adjacency and a label vector, we compute the

Clustering ARI					
	AEE	LEE	ASE	LSE	N2v
Cora	0.12	0.07	0.08	0.01	**0.24**
Email	**0.40**	0.39	0.11	0.21	0.34
Gene	0.01	0.01	0.01	0.01	0.00
Industry	**0.13**	0.03	0.01	0.02	**0.13**
LastFM	0.34	0.19	0.03	**0.47**	0.43
PolBlog	**0.80**	0.58	0.07	**0.80**	**0.80**
Running Time (seconds)					
	AEE	LEE	ASE	LSE	N2v
Cora	**0.11**	0.12	1.6	1.7	2.2
Email	0.18	0.28	**0.13**	0.20	1.3
Gene	**0.03**	**0.03**	0.17	0.20	0.90
Industry	0.02	0.02	0.02	0.02	0.40
LastFM	**0.35**	0.39	13.6	15.5	9.5
PolBlog	**0.05**	0.07	0.27	0.29	1.4

TABLE 2
K-means clustering results for each embedding method. For each graph, the highest ARI and lowest running time is highlighted in bold. The running time includes both embedding and k-means clustering.

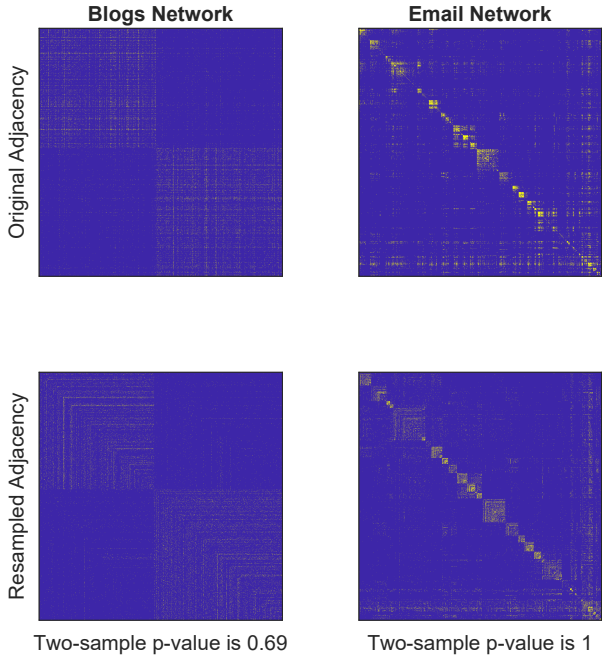


Fig. 6. The top row is the original adjacency matrix for each data, the bottom row is one bootstrap adjacency at $n = 1000$, with two-sample test p-value computed at bottom.

encoder embedding and carry out standard bootstrap on the embedding, then use Bernoulli distribution to form the resampled adjacency matrix. We validate the procedure via a two-sample distance-correlation test [30], [31] between the original and bootstrap graphs via the encoder embedding (testing using graph embedding is asymptotically valid upon mild model assumptions [11], [32]). A large p-value suggests that the resampled graph has the same distribution as the original graph, while a small p-value (say less than 0.05) implies the resampled graph is significantly different in distribution and thus breaking the intention of bootstrap.

Algorithm 3 Encoder Embedding for Graph Bootstrap

Require: $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{Y} \in \{1, \ldots, K\}^n$, and resampling size n_2.

Ensure: Resampled adjacency matrix $\mathbf{A}_2 \in \mathbb{R}^{n_2 \times n_2}$, corresponding label $\mathbf{Y}_2 \in \{1, \ldots, K\}^{n_2}$, and a two-sample test p-value $pval$.

function GEEBOOTSTRAP($\mathbf{A}, \mathbf{Y}, n_2$)
  $[\mathbf{Z}, \mathbf{W}] = \text{GEE}(\mathbf{A}, \mathbf{Y})$;
  $ind = \text{bootstrap}(n, n_2)$; ▷ sampling n_2 indices with replacement from $\{1, \ldots, n\}$
  $\mathbf{Y}_2 = \mathbf{Y}[ind]$; ▷ resampled class labels
  $\mathbf{Z}_2 = \mathbf{Z}[ind, :]$; ▷ resampled encoder embedding
  $\mathbf{A}_2 = \text{zeros}(n_2, n_2)$;
  for $i = 1, \ldots, n_2$ **do**
    for $j = i + 1, \ldots, n_2$ **do**
      $\mathbf{A}_2[i, j] = \text{Bernoulli}(\mathbf{Z}_2[i, \mathbf{Y}_2[j]])$;
      $\mathbf{A}_2[j, i] = \mathbf{A}_2[j, i]$;
    end for
  end for
  $pval = \text{twosample}(\text{GEE}(\mathbf{A}, \mathbf{Y}), \text{GEE}(\mathbf{A}_2, \mathbf{Y}_2))$;
end function

Figure 6 visualizes the graph bootstrap results for the Political Blog and the Email Network (we pick these two graphs as they have clearer community structure and thus better for visualization). The resampled graph not only appears quantitatively similar to the original graph, but also yields a very large p-value from the two-sample test. This suggests the bootstrap graph is indiscernable from the original graph in distribution.

We repeat the bootstrap sampling at $n_2 = 1000$ for 1000 times, and compute the two-sample p-value for each replicate. Algorithm 3 yields a mean p-value of 0.75 for the political blog data. Moreover, only 0.4% of the replicates yields a p-value that is less than 0.05. In comparison, we also evaluated the naive bootstrap on graph adjacency. The mean p-value is 0.25, and 26% of the replicates have p-value less than 0.05. Therefore, adjacency encoder embedding offers a better solution for graph bootstrap.

8 Conclusion

In this paper we proposed the one-hot graph encoder embedding method. The theoretical soundness is proved via asymptotic convergence and normality, and the numerical advantages are demonstrated in classification, clustering, and bootstrap. It is a flexible framework that can work with ground-truth labels, labels induced from other methods, partial or no labels at all. Most importantly, the excellent numerical performance is achieved via an elegant algorithmic design and a tiny fraction of time vs existing methods, making the graph encoder embedding very attractive and uniquely poised for huge graph data.

References

[1] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of National Academy of Science*, vol. 99, no. 12, pp. 7821–7826, 2002. 1

[2] L. Adamic and N. Glance, "The political blogosphere and the 2004 us election: Divided they blog," in *Proceedings of the 3rd International Workshop on Link Discovery*. New York: ACM Press, 2005, pp. 36–43. 1, 3, 4

[3] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. E 74, no. 036104, 2006. 1

[4] J. T. Vogelstein, Y. Park, T. Ohyama, R. Kerr, J. Truman, C. E. Priebe, and M. Zlatic, "Discovery of brainwide neural-behavioral maps via multiscale unsupervised structure learning," *Science*, vol. 344, no. 6182, pp. 386–392, 2014. 1

[5] L. Chen, C. Shen, J. T. Vogelstein, and C. E. Priebe, "Robust vertex classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 3, pp. 578–590, 2016. 1

[6] K. Rohe, S. Chatterjee, and B. Yu, "Spectral clustering and the high-dimensional stochastic blockmodel," *Annals of Statistics*, vol. 39, no. 4, pp. 1878–1915, 2011. 1, 3

[7] D. Sussman, M. Tang, D. Fishkind, and C. Priebe, "A consistent adjacency spectral embedding for stochastic blockmodel graphs," *Journal of the American Statistical Association*, vol. 107, no. 499, pp. 1119–1128, 2012. 1, 3

[8] M. Tang, D. L. Sussman, and C. E. Priebe, "Universally consistent vertex classification for latent positions graphs," *Annals of Statistics*, vol. 41, no. 3, pp. 1406–1430, 2013. 1

[9] D. Sussman, M. Tang, and C. Priebe, "Consistent latent position estimation and vertex classification for random dot product graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 1, pp. 48–57, 2014. 1

[10] M. Tang, A. Athreya, D. L. Sussman, V. Lyzinski, and C. E. Priebe, "A nonparametric two-sample hypothesis testing for random dot product graphs," *Bernoulli*, vol. 23, no. 3, pp. 1599 – 1630, 2017. 1

[11] ——, "A nonparametric two-sample hypothesis testing for random dot product graphs," *Bernoulli*, vol. 23, no. 3, pp. 1590–1630, 2017. 1, 6

[12] C. Priebe, Y. Parker, J. Vogelstein, J. Conroy, V. Lyzinskic, M. Tang, A. Athreya, J. Cape, and E. Bridgeford, "On a 'two truths' phenomenon in spectral graph clustering," *Proceedings of the National Academy of Sciences*, vol. 116, no. 13, pp. 5995–5600, 2019. 1, 2

[13] P. Holland, K. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social Networks*, vol. 5, no. 2, pp. 109–137, 1983. 1, 2

[14] S. Young and E. Scheinerman, "Random dot product graph models for social networks," in *Algorithms and Models for the Web-Graph*. Springer Berlin Heidelberg, 2007, pp. 138–149. 1, 3

[15] S. S. Bryan Perozzi, Rami Al-Rfou, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710. 1

[16] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864. 1

[17] R. Liu and A. Krishnan, "Pecanpy: a fast, efficient and parallelized python implementation of node2vec," *Bioinformatics*, vol. 37, no. 19, pp. 3377–3379, 2021. 1, 3

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016. 1

[19] T. Snijders and K. Nowicki, "Estimation and prediction for stochastic blockmodels for graphs with latent block structure," *Journal of Classification*, vol. 14, no. 1, pp. 75–100, 1997. 2

[20] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Physical Review E*, vol. 83, p. 016107, 2011. 2

[21] C. Gao, Z. Ma, A. Y. Zhang, and H. H. Zhou, "Achieving optimal misclassification proportion in stochastic block models," *Journal of Machine Learning Research*, vol. 18, pp. 1–45, 2017. 2

[22] Y. Zhao, E. Levina, and J. Zhu, "Consistency of community detection in networks under degree-corrected stochastic block models," *Annals of Statistics*, vol. 40, no. 4, pp. 2266–2292, 2012. 2

[23] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com 3, 4

[24] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017. 4

[25] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 2020, p. 1325–1334. 4

[26] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971. 5

[27] B. Efron and R. Tibshirani, *An introduction to the bootstrap*. Chapman & Hall, 1993. 5

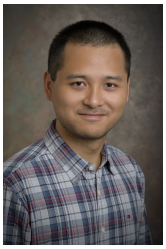
[28] S. Bhattacharyya and P. J. Bickel, "Subsampling bootstrap of count features of networks," *Annals of Statistics*, vol. 43, pp. 2384–2411, 2015. 5

[29] A. Green and C. R. Shalizi, "Bootstrapping exchangeable random graphs," *Electronic Journal of Statistics*, vol. 16, no. 1, pp. 1058–1095, 2022. 5

[30] G. Szekely, M. Rizzo, and N. Bakirov, "Measuring and testing independence by correlation of distances," *Annals of Statistics*, vol. 35, no. 6, pp. 2769–2794, 2007. 6

[31] S. Panda, C. Shen, C. E. Priebe, and J. T. Vogelstein, "Multivariate multisample multiway nonparametric manova," *https://arxiv.org/abs/1910.08883*, 2022. 6

[32] Y. Lee, C. Shen, C. E. Priebe, and J. T. Vogelstein, "Network dependence testing via diffusion maps and distance-based correlations," *Biometrika*, vol. 106, no. 4, pp. 857–873, 2019. 6



Cencheng Shen received the BS degree in Quantitative Finance from National University of Singapore in 2010, and the PhD degree in Applied Mathematics and Statistics from Johns Hopkins University in 2015. He is assistant professor in the Department of Applied Economics and Statistics at University of Delaware. His research interests include graph inference, dimension reduction, hypothesis testing, correlation and dependence.

Qizhe Wang received the BS degree in Statistics from University of Delaware in 2019, and the MS degree in Statistics from University of Delaware in 2021.

Carey E. Priebe received the BS degree in mathematics from Purdue University in 1984, the MS degree in computer science from San Diego State University in 1988, and the PhD degree in information technology (computational statistics) from George Mason University in 1993. From 1985 to 1994 he worked as a mathematician and scientist in the US Navy research and development laboratory system. Since 1994 he has been a professor in the Department of Applied Mathematics and Statistics at Johns Hopkins University. His research interests include computational statistics, kernel and mixture estimates, statistical pattern recognition, model selection, and statistical inference for high-dimensional and graph data. He is a Senior Member of the IEEE, an Elected Member of the International Statistical Institute, a Fellow of the Institute of Mathematical Statistics, and a Fellow of the American Statistical Association.