Classic Graph Structural Features Outperform Factorization-Based Graph Embedding Methods on Community Labeling

Andrew Stolman* Caleb Levy† C. Seshadhri‡ Aneesh Sharma§

Abstract

Graph representation learning (also called *graph embeddings*) is a popular technique for incorporating network structure into machine learning models. Unsupervised graph embedding methods aim to capture graph structure by learning a low-dimensional vector representation (the *embedding*) for each node. Despite the widespread use of these embeddings for a variety of downstream transductive machine learning tasks, there is little principled analysis of the effectiveness of this approach for common tasks. In this work, we provide an empirical and theoretical analysis for the performance of a class of embeddings on the common task of pairwise community labeling. This is a binary variant of the classic community detection problem, which seeks to build a classifier to determine whether a pair of vertices participate in a community. In line with our goal of foundational understanding, we focus on a popular class of unsupervised embedding techniques that learn low rank factorizations of a vertex proximity matrix (this class includes methods like GraRep, DeepWalk, node2vec, NetMF). We perform detailed empirical analysis for community labeling over a variety of real and synthetic graphs with ground truth. In all cases we studied, the models trained from embedding features perform poorly on community labeling. In constrast, a simple logistic model with classic graph structural features handily outperforms the embedding models. For a more principled understanding, we provide a theoretical analysis for the (in)effectiveness of these embeddings in capturing the community structure. We formally prove that popular low-dimensional factorization methods either cannot produce community structure, or can only produce "unstable" communities. These communities are inherently unstable under small perturbations. This theoretical result suggests that even though "good" factorizations exist, they are unlikely to be found by computational methods.

1 Introduction

Graph structured data is ubiquitous. Capturing the graph structure is important for a wide variety of machine learning tasks, such as ranking in social networks, content recommendations, and clustering [EK10]. A long-studied challenge for building such machine learned models has been to capture the graph structure for use in a variety of modeling tasks. *Graph representation learning*, or *low-dimensional graph embeddings*, provide a convenient solution to this problem. Given a graph G on n vertices, these methods map each vertex to a vector in $\mathbb{R}^d$, where $d \ll n$, in an unsupervised or a self-supervised manner (it is also sometimes referred to as a pre-training procedure). Typically, the goal of the embedding is to represent graph proximity by (a function of) the dot product of vectors, thereby implicitly giving a geometric representation of the graph.[1] The dot product formulation provides a convenient form for building a models (e.g. using deep learning). Moreover, the geometry of the embedding allows efficient reverse-index lookups, using nearest neighbor search [CAS16, Twi18].

The study of low-dimensional graph embeddings is an incredibly popular research area, and has generated many exciting results over the past few years (see surveys [HYL18, CAEHP+20] and a Chapter 23 in [Mur21]). Nonetheless, there is limited principled understanding of the power of low-dimensional embeddings (a few recent papers address this topic [SSSG20, CMST20, Lou20, GJJ20]). Our work aims to understand the effectiveness of a class of graph embeddings in preserving graph structure as it manifests in performance on different downstream

*University of California, Santa Cruz. astolman@ucsc.edu

†University of California, Santa Cruz. cclevy@ucsc.edu

‡University of California, Santa Cruz. sesh@ucsc.edu. Supported by NSF DMS-2023495, CCF-1740850, CCF-1813165, CCF-1839317, CCF-1908384, CCF-1909790, and ARO Award W911NF1910294.

§Google. aneesh@google.com

[1] Since there is a wide range of methods for Graph representation learning, we refer the reader to the "Shallow embeddings" class in a recent survey [CAEHP+20] for a more comprehensive overview.

tasks.

Due to an explosion of interest in the area, there are by now a large class of graph embedding methods [Mur21]. For the sake of a principled study, we focus on the important class of *unsupervised low-rank factorization methods*. While there do exist many methods outside this class, such factorization methods cover a number of popular and influential embeddings methods, including GraRep [OCP+16], DeepWalk [PARS14], and Node2Vec [GL16]. In fact, a recent result shows that many existing embedding techniques can be recast as matrix factorization methods [QDM+18]. One begins with an $n \times n$ promixity matrix M, typically the adjacency matrix, the random walk matrix, or some variant thereof (e.g. Node2Vec uses the matrix for a certain second order random walk). Using optimization techniques, the matrix M is approximated as a Gramian matrix $V^T V$, where $V \in \mathbb{R}^{d \times n}$. The column vectors of V are the embeddings of the vertices. In direct factorizations, one simply tries to minimize $\|V^T V - M\|_2$. More sophisticated softmax factorizations perform non-linear entry-wise transformations on $V^T V$ to approximate M. This class of unsupervised embedding methods is among the most popular and prevalent low-dimensional graph embeddings, and hence this is a particularly useful class to quantify performance for.

Our aim is to study a natural question, albeit one that is somewhat challenging to pose formally: *to what extent do factorization-based embedding methods capture graph structure relevant to downstream ML tasks?*

To this end, we fix the following well-defined *pairwise community labeling* problem. Given two vertices i and j, the binary classification task is to determine whether they belong to the same community. We note that this community labeling problem is an instance of a broad range of community detection problems that have a long history of study in the graph mining literature [LRU20]. We then attempt a rigorous theoretical and empirical understanding of the performance of factorization-based graph embeddings on community labeling.

We note that there are graph embedding methods that are not factorization based (e.g. GraphSage [HYL17]) as well as factorization methods that do not use direct or softmax factorizations [CMST20], as well as GCNs and GNNs [TW17, CAEHP+20]. For the sake of a principled study, we chose a well-defined subclass of methods that covered many important methods such as `GraRep` [CLX15], `DeepWalk` [PARS14], `Node2Vec` [GL16], and `NetMF` [QDM+18]. Moreover, the recent `NetMF` algorithm shows how many past methods can be recast as factorization methods.

One central promise of unsupervised graph embedding methods is to preserve network structure in the geometry that can then be useful for downstream tasks. In our work, we focus on the task of community labeling both because it is directly connected to the core question: how well do embeddings preserve the graph structure?

1.1 Formal description of setting

We formally describe the graph embeddings techniques that are studied in this work. The learned factorization approach is to approximate M by the Gramian matrix $V^T V$ (the matrix of dot products). Typically, this matrix V is found by formulating a machine learning problem, which has a loss function that minimizes a distance/norm between $V^T V$ and M. Broadly speaking, we can classify these methods into two categories:

- **Direct factorizations:** Here, we set V as $\text{argmin}_V \|V^T V - M\|_2$, where M is typically (some power of) the graph adjacency matrix. Methods such as Graph Factorization, GraRep [CLX15], and HOPE [OCP+16] would fall under this category.
- **Softmax factorizations:** These methods factorize a stochastic matrix, such as (powers of) the random walk matrix. Since $V^T V$ is not necessarily stochastic, these methods apply the softmax to generate a stochastic matrix. Notable examples are such methods are DeepWalk [PARS14] and Node2vec [GL16]. Formally, consider the normalized softmax matrix $\text{nsm}(V)$ given by

$$\text{nsm}(V)_{ij} = \frac{\exp(\vec{v}_i \cdot \vec{v}_j)}{\sum_k \exp(\vec{v}_i \cdot \vec{v}_k)} \tag{1.1}$$

Note that $\text{nsm}(V)$ is stochastic by construction. The objective of the learning problem is to minimize $KL(\text{nsm}(V), M)$, which is the sum of row-wise KL-divergence between the rows (this is equivalent to cross-entropy loss).

The recent `NetMF` [QDM+18] method interpolates between these categories and shows that a number of existing methods can be expressed as factorization methods, especially of the above forms. For this study, we only focus on the above two category of unsupervised embedding methods. (We discuss this choice and other methods in §1.3.)

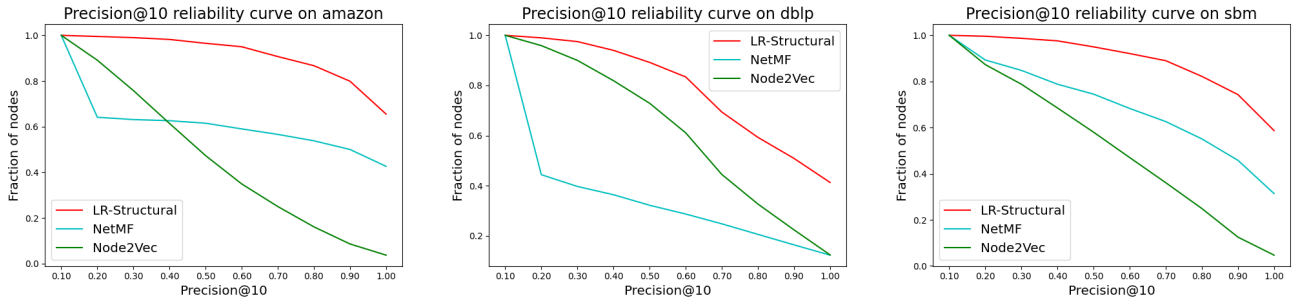

Figure 1: Each point, (x, y), on the curve represents the approximate fraction of vertices, y, for which the given method produces a precision@10 score of at least x. `LR-Structural` is plotted against the two best performing embedding methods. 1000 vertices are sampled and for each vertex v sampled, the vertices of the graph $u_1, \ldots, u_n$, are ordered by decreasing score assigned by the given classifier. The precision@10 is the fraction of $u_1, \ldots, u_{10}$ which share a community with v.

Empirical setup: We empirically investigate performance of the above methods on graphs where the ground truth communities correlate well with graph structure. In the case of real data, the ground truth is provided by the existing community labels. With synthetic data, we explicitly construct stochastic block models with well-defined communities. For every pair of vertices i, j, the prediction problem is to determine whether they belong to the same community (note that they may belong together in multiple communities, but we do not require the community label itself to be determined; just whether they belong in *any* community together).

For both real and simulated data, we note that the ground truth is sparse, i.e. the vast majority of node pairs do not belong to the same community. Hence, it is appropriate to measure the prediction performance using precision-recall curves for this highly imbalanced label distribution [DG06]. Consistent with most of the literature on graph embeddings and the applications that often require nearest neighbor lookups, the main feature we use for prediction is the value of the dot product [CAS16, CAEHP+20].

Theoretical setup: In order to analyze the performance, we provide an abstraction of community structure from a matrix standpoint: this can intuitively be thought of as having many dense blocks in an overall sparse matrix. We then attempt to quantify "how much" community structure can be present in a matrix $V^T V$ or $\text{nsm}(V)$, for *any* matrix $V \in \mathbb{R}^{d \times n}$ (for $d \ll n$). This formulation captures the fundamental notion of a low-rank factorization, without referring to any specific method to compute it. Our results hold for *any* direct/softmax factorization method, regardless of how the embedding is computed. Our formulation theoretically investigates whether it is even possible to recreate community structure using a low rank factorization of the form $V^T V$ or $\text{nsm}(V)$.

1.2 Main results All the graph embeddings methods we tested (including `GraRep`, `DeepWalk`, `Node2Vec`, and `NetMF`) perform poorly on the community labeling task, and are handily out-performed by a baseline logistic model `LR-Structural` built using just four classic graph structural features[2]. We observe the same outcome for a series of experiments on real data and synthetic data. Motivated by this empirical finding, we provide a mathematical explanation for this result by providing a theorem which shows that the community structure exhibited by softmax factorizations is unstable under small perturbations of the embedding vectors.

Evaluations on real data: In our experiments, not only do we see poor *absolute* performance on the community labeling task, but a baseline `LR-Structural` based on "classic" graph features handily outperforms the embeddings. We see this difference not only in an overall manner, but across individual nodes in the graph. In particular, Figure 1 shows a "reliability plot" for precision@10 for a set of 1000 nodes sampled randomly from each of the

[2]The features for a node pair (u, v) are: 1) Personalized-PageRank (PPR) score from u to v, 2) PPR from v to u, 3) cosine similarity among $N(u)$ and $N(v)$ (where $N(\cdot)$ denotes a node's neighborhood), and 4) the size of the cut separating $N(u)$ and $N(v)$

graphs. To produce this chart, we first randomly sampled 1000 nodes, each of which has at least 10 neighbors in the same community. Then, each of these vertices selects their top-10 predictions for a same-community neighbor using a model, producing a distribution over precision@10. Then for each model, one can produce a reliability plot (see Figure 1) that produces points (x, y): for any given value x for precision@10, define y as the fraction of nodes that have precision@10 of at least x. Thus, we'd expect that a highly accurate model would have an almost flat curve. The results in Figure 1 show that generally `LR-Structural` can produce high accurate (precision@k ≥ 0.7) community labels for 20–40% more nodes than the embedding-based models. This suggests that the embeddings can retrieve very few of the community neighbors, while classic graph features used in `LR-Structural` (PPR scores, neighborhood similarity) recover much of the community structure.

We emphasize that our empirical analysis is more nuanced than the more common aggregated measurement (such as via the AUC-ROC [CAEHP$^+$20]) as it measures the performance across individual nodes in the graph. We believe this individual measurement is more reflective of our goal, and also, as the label distribution (which pairs co-occur in communities) is highly imbalanced, a P/R-like curve provides a more useful measure than the ROC curves. The latter can be misleading as an ROC curve can still look quite good while misclassifying much of the minority class. [DG06].

For completeness, we also perform experiments with regression models using the Hadamard product of the two vectors. Again, the embeddings gives results of similar quality, showing that linear functions (of the embedding vectors) fail to predict community structure. Refer to Section 3 for more details.

Evaluations on SBMs: To further investigate this phenomenon, we also generated synthetic graph instances using Stochastic Block Models (SBMs) that have a very simple planted community structure. For example, we create a graph with $n = 10^5$ vertices and blocks of size 20. The edge density inside a block is 0.3 and we connect the blocks by a sparse Erdős-Rényi graph such that the average number of links within a block is equal to those that go between blocks. We vary the overall edge density (while keeping the ratio between inter-block and intra-block edges constant) and study the precision@10 scores. We observe that while the performance across methods tends to increase with density, `LR-Structural` still outperforms them.

Theoretical explanation: We provide a formulation for what it means for a matrix to exhibit community structure. We emphasize that this is not meant to completely capture the challenging notion of communities (which has a deep and rich history), but rather to give us some formal framework to state our impossibility results. Intuitively, an overall sparse matrix/graph has community structure if a non-trivial fraction of the rows/vertices participate in small dense blocks of entries. We then investigate when V^TV and nsm(V) exhibit community structure. First, we prove that for $d \ll n$, V^TV cannot exhibit community structure, which provides a principled justification for the empirical observation that direct factorization methods perform poorly across all real and synthetic instances.

Interestingly, we also show that despite the negative result for truncated-SVD in [SSSG20], softmax factorizations *can* exhibit community structure similar to those given by the SBMs discussed earlier. Recent work shows that threshold based sign factorizations can also embed such structure [CMST20]. But we prove that this community structure is fundamentally unstable under small perturbations of the vectors obtained from softmax factorizations. Meaning, if we take any matrix V such that nsm(V) has community structure, then with high probability, nsm$(\widetilde{V})$ does not have such a structure (where $\widetilde{V}$ is a slight random perturbation of nsm(V)).

This strongly suggests that optimization methods cannot produce low dimensional matrices V where nsm(V) has community structure. This theorem provides a mathematical understanding of the limitations of softmax factorizations. We do note that since softmax factorization are superior to direct factorizations, since they avoid the direct impossibility for the latter.

1.3 Related Work We briefly note a few important representation learning techniques that are beyond the scope of our work. The most prominent among these is the Graph Neural Networks such as [HYL17, TW17, VCC$^+$18, HLG$^+$20], which can be thought of as a class of learned message passing methods. We refer the reader to a nicely interpretable classification of these methods in a recent survey [CAEHP$^+$20]. The factorization methods we study fall under the "Shallow embeddings" classification there. There are several recent works [GJJ20,

Lou20, XHLJ19] that theoretically study the power of GNNs, but this is complementary to our work since we study a different class of methods.

A recent result shows the inability of low-dimensional SVD based embeddings of preserving the triangle structure of real-world networks [SSSG20]. A followup showed that these impossibility results can be circumvented by alternate embedding methods [CMST20]. Our result can be thought of as a deeper investigation into this issue. First, we look at a class of factorization methods subsuming those used in practice. Secondly, we also focus on a specific downstream ML task, unless previous results that focus solely on the graph structure.

2 Mathematical results and interpretation

We define a simple abstraction of community structure in a matrix M. Then, we try to quantify how much community structure Gram matrices and softmax factorizations can possess. We will state these as formal theorems, which are our main mathematical result. The full proofs are given in the full version.

Let us start with an $n \times n$ matrix M that represents the "similarity" or likelihood of connection between vertices. For convenience, let us normalize so that the $\forall i \in [n], \sum_{j \leq n} M_{i,j} \leq 1$. (So the sum of similarities of a vertex is at most 1.) A communities is essentially a dense block of entries, which motivates the following definition. We use ε to denote a parameter for the threshold of community strength. One should think of ε as a small constant, or something slowly decreasing in n (like $1/\text{poly}(\log n)$).

DEFINITION 2.1. *A pair of vertices (i, j) is a* potential community pair *if both M_{ij} and M_{ji} are at least ε.*

Note that we do not expect all such pairs (i, j) to truly be together in a community. Hence, we only consider such a pair a potential candidate. We expect community relationships to be mutual, even if the matrix M is not. A community can be thought of as a submatrix where at least a constant fraction of pairs are potential community pairs. For our purposes, we do not need to further formalize. It is natural to expect that $\Theta(n)$ pairs are community pairs; indeed, most vertices should participate in communities, and will have at least a constant number of community neighbors. Our mathematical analyses shows that direct and softmax factorizations cannot produce these many potential community pairs.

Lower bound for direct factorizations: We first show a strong lower bound for direct factorizations. We prove that the number of potential community pairs in $V^T V$ is linear in the rank, and thus, a low-dimensional factorization cannot capture community structure. The key insight is to use the rotational invariance of Frobenius norms.

THEOREM 2.1. *Consider any matrix $V \in \mathbb{R}^{d \times n}$ such that row sums in $V^T V$ have absolute value at most 1. Then V has at most $d/2\varepsilon^2$ potential community pairs.*

Proof. Since $V^T V$ has row sums of absolute value at most 1, the spectral radius (largest absolute value of eigenvalue) is also at most 1. (This can be proven directly, but it also a consequence of the Gershgorin circle theorem [Ger20].) Since the rank of $V^T V$ is at most d, $V^T V$ has at most d non-zero eigenvalues. We can express the Frobenius norm squared, $\|V^T V\|_2^2$, by the sums of squares of eigenvalues. By the arguments above, $\|V^T V\|_2^2 \leq d$.

Note that $\|V^T V\|_2^2$ is also the sums of squares of entries. Each potential community pair contributes at least $2\varepsilon^2$ to this sum. Hence, there can be at most $d/2\varepsilon^2$ potential community pairs. ∎

The instability of softmax factorizations: The properties of softmax factorizations are more nuanced. Firstly, we can prove that softmax factorizations *can* represent community structure quite effectively.

THEOREM 2.2. *For $d = O(\log n)$, there exists $V \in \mathbb{R}^{d \times n}$ such that $\text{nsm}(V)_{ij}$ exhibits community structure. Specifically, for any natural number $b \leq n$, there exists $V \in \mathbb{R}^{d \times n}$ such that $\text{nsm}(V)$ has n/b blocks of size b, such that all entries within blocks are at least $1/2b$.*

Indeed, this covers the various SBM settings we study, and demonstrates the superiority of softmax factorizations for modeling community structure. We note that a similar theorem (for a different type of factorization) was proved in [CMST20].

On the other hand, we prove that these factorizations are highly *unstable* to small perturbations. Indeed, with a tiny amount of noise, any community pair can be destroyed with high probability.

Formally, our noise model is as follows. Let $\delta > 0$ be a noise parameter. Think of the ith column of V as the d-dimensional vector $\vec{v}_i$, which is the embedding of vertex i. For every vector $\vec{v}_i$, we generate an independent random Gaussian $X_i \sim \mathcal{N}(0, \delta^2)$ and

rescale $\vec{v}_i$ as $(1+X_i)\vec{v}_i$ (formally, we rescale to $e^{X_i}\vec{v}_i$, to ensure that the scaling is positive). We denote this perturbed matrix as $\widetilde{V}^{(\delta)}$. We think of δ as a quantity going to zero, as n becomes large. (Or, one can consider δ as a tiny constant.)

THEOREM 2.3. *Let c denote some absolute positive constant. Consider any $V \in \mathbb{R}^{d\times n}$. For any $\delta > c\ln(1/\varepsilon)/\ln n$, the following holds in* $\mathrm{nsm}(\widetilde{V}^{(\delta)})$. *For at least* $0.99n$ *vertices i, for any pair (i,j), the pair is* not *a potential community pair with probability at least* 0.99.

Thus, with overwhelming probability, any community structure in $\mathrm{nsm}(V)$ is destroyed by adding $o(1)$ (asymptotic) noise. This is strong evidence that either noise in the input or numerical precision in the final optimization could lead to destruction of community structure. These theorems give an explanation of the poor performance of the embeddings methods studied.

2.1 Proof ideas In this section, we lay out the main ideas in proving Theorem 2.3. It helps to begin with the upper bound construction of Theorem 2.2. Quite simply, we take n/b random Gaussian vectors, and map vertices in a community/block to the same vector. After doing some calculations of random dot products, one can deduce that the vectors need to have length $\Omega(\sqrt{\ln n})$ for the construction to go through. By carefully look at the math, one also finds that the construction is "unstable". Even perturbing the vectors within a block slightly (so that they are no longer the same vector) affects the block structure. We essentially prove that these properties hold for *any* set of vectors.

We outline the proof of Theorem 2.3. Suppose $\mathrm{nsm}(V)_{ij} > \varepsilon$. Note that $\sum_{k\in[n]} \mathrm{nsm}(V)_{ik} = 1$, since $\mathrm{nsm}(V)$ is normalized by construction. Thus, there exists some k such that $\mathrm{nsm}(V)_{ik} \leq 1/n$. We deduce that $\mathrm{nsm}(V)_{ij}/\mathrm{nsm}(V)_{ik} > \varepsilon n$. Writing out the entries and taking logs, this implies $\vec{v}_i \cdot \vec{v}_j - \vec{v}_i \cdot \vec{v}_k > \ln(\varepsilon n)$. Therein lies the power (and eventual instability) of softmax factorizations: ratios of entries are transformed into differences of dot products. By Cauchy-Schwartz, one of $\vec{v}_i, \vec{v}_j, \vec{v}_k$ must have length $\Omega(\sqrt{\ln n})$ (ignoring ε dependencies). By an averaging argument, we can conclude that for a vast majority of community pairs (i,j), $\|\vec{v}_i\|_2 = \Omega(\sqrt{\ln n})$. Note that both $\mathrm{nsm}(V)_{ij}$ and $\mathrm{nsm}(V)_{ji}$ must be at least ε; these quantities have the same numerator, but different denominators. By analyzing these expressions and some algebra, we can prove that $\left|\|\vec{v}_i\|_2 - \|\vec{v}_j\|_2\right| = o(1/\sqrt{\ln n})$.

Thus, we discover the key property of communities expressed by softmax factorizations. Vectors with a community have length $\Omega(\sqrt{\ln n})$, but the differences in lengths must be $O(1/\sqrt{\ln n})$. Asymptotically, this is unstable to perturbations in the length. A vanishingly small change in the vector lengths can destroy the community.

3 Empirical verification

3.1 Community pair prediction We study the performance of matrix factorization embeddings at identifying community structure in a graph with many possibly overlapping communities. Formally, we state this as a binary classification task over pairs of vertices. Every dataset consists of a graph, G, and a set of (possibly overlapping) communities, $C_1, C_2, \ldots$. This gives us a ground truth labeling over the pairs from $V \times V$ where positive instances are those (u,v) such that $u, v \in C_k$, for some community C_k. We evaluate the performance of embedding techniques on this binary classification task over $V \times V$ as follows:

1. An embedding method is applied to G to obtain an embedding $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ of the nodes in V.

2. A *pair scoring function*, $f : V \times V \to \mathbb{R}$, is constructed from the embedding of node pairs.

All of the evaluation is done with respect to this pair scoring function. We use this abstraction to encapsulate all of the tasks downstream of the embedding generation itself. It consists of mapping the embedding vectors to predictions in $\mathbb{R}$. Ideally, $f(u,v) > f(x,y)$ whenever u and v share a community and x and y do not. The construction of f is based on the dot product of the embedding vectors of u and v; details are given in §3.3.2. We stress that the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ incorporate information about the neighborhoods. The underlying optimization of graph embeddings tries to produce close vectors for vertices in dense regions.

3.2 Experimental results We observe that no choice of embedding method were competitive with the baseline `LR-Structural` method on the task of community pair prediction. Across three datasets, `LR-Structural` was consistently able to identify community pairs while the embedding-based methods were not. For each method being compared, one thousand vertices were selected at random from the

method/dataset	amazon	dblp	sbm
`LR-Structural`	.92	.80	.88
DeepWalk	.44	.49	.35
NetMF	.49	.28	.66
Node2Vec	.49	.61	.55
DeepWalk-hp	.43	.44	.33
NetMF-hp	.41	.37	.76
Node2Vec-hp	.37	.55	.50

Figure 2: Average precision@10 across 100 samples for all methods across the three datasets. In all cases, `LR-Structural` is the best performing.

graph, and we examine the precision of the classifier on the neighborhoods of each selected vertex. Fig. 1 shows that `LR-Structural` makes more precise predictions on average than the best performing datasets, while Fig. 2 contains the mean precision for each classifier on each dataset.

The methods are evaluated by comparing the distribution of a precision@10 for each method on each dataset. For each of 1000 vertices sampled, v, we order the other vertices of the graph, $u_1, \ldots, u_n$ such that $f(\boldsymbol{v}, \boldsymbol{u}_i) \geq f(\boldsymbol{v}, \boldsymbol{u}_{i+1})$ for all i. We compute the precision of the classifier on $(v, u_1), \ldots, (v, u_n)$ when it predicts positive labels only for those (v, u_i) such that $i \leq 10$. In other words, we sample a vertex at random and report the fraction of its ten nearest neighbors in the embedding space with which it shares a community.

We represent the distribution of values of precision@10 scores as a reliability curve. This is the curve (x, y) such that at least a y fraction of vertices sampled had a precision@10 score score of at least x. Higher y values for a given x indicate better performance. Fig. 1 contains the curves for the best performing dot product methods against the baseline, while Fig. 2 contains the mean across samples.

Given Theorem 2.3, we expect that if (u, v) is a community pair in some ground truth matrix, M, then it is unlikely that (u, v) is a community pair in any noisy approximation of M. The results in this section bear out this conclusion. The community structure of the original graphs are not preserved by the embeddings.

3.3 Experimental setup

3.3.1 Community pair prediction methods We compare the performance of four different embedding methods (GraRep, DeepWalk, Node2Vec, NetMF) to a baseline to a simple supervised logistic regression model using common structural graph features. The implementation of the embedding methods is based on [RKS20]. Except for setting the dimension to 128 for all embedding methods, the hyperparameters are taken from [RKS20]. Our code is available at `https://github.com/astolman/snlpy`. All our experiments were run on AWS using machines with up to 96 cores and 1 TB memory. The methods were implemented in python based on the karateclub package [RKS20] . Any method which did not complete in 24 hours or which required more memory was considered not complete. In the full version, we give more details about the embedding methods.

`LR-Structural` For the baseline method, we use a logistic regression model trained to predict community pairs based on four tractable graph features found to be useful in the literature:

- Cosine similarity between u's and v's adjacency vectors [SSG17],
- Size of the cut between u's neighborhood and v's neighborhood, and
- Personalized PageRank (PPR) score from u to v, as well as that from v to u [RAL07].

We use the approximation algorithm from [RAL07] to compute sparse approximate PPR vectors. Note that this allows for all features to be computed locally, i.e. the space/time complexity for computing the features for one vertex pair are independent of graph size. The overall space/time costs in practice are on par with the embedding methods.

3.3.2 Pair features The embedding methods produce a feature vector for each vertex. Since the community pair prediction task requires a set of feature vectors over $V \times V$ and a method to produce scores, we need to turn node features into pair features. Building on strategies proposed in prior work [GL16], for the embeddings we choose two methods from the literature which are computationally efficient, and relatively accurate. Given a d-dimensional embedding, $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ we compute the feature vector for $(u, v) \in V \times V$ with the *dot product* and *Hadamard product* denoted $\boldsymbol{u} \cdot \boldsymbol{v}$ and $\boldsymbol{u} \circ \boldsymbol{v}$ respectively.

The dot product is the usual inner product over $\mathbb{R}^d$, i.e. $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i \in [d]} \boldsymbol{u}(i)\boldsymbol{v}(i)$. It takes the two d-dimensional vector embeddings and maps them to a 1-dimensional feature vector. On the other hand, the Hadamard product of $\boldsymbol{u}$ and $\boldsymbol{v}$, $\boldsymbol{u} \circ \boldsymbol{v}$, is a d-dimensional vector whose ith coordinate is the

product of the ith coordinates of $\boldsymbol{u}$ and $\boldsymbol{v}$.

We will indicate when the pair features used are the dot product or Hadamard product of the endpoints. The actual scoring function, $f : V \times V \to \mathbb{R}$, we use is implicit from the pair features. Whenever the pair features are dot products, $f(u, v) = \boldsymbol{u} \cdot \boldsymbol{v}$. Whenever the pair features are the Hadamard product of $\boldsymbol{u}$ and $\boldsymbol{v}$, we fit a logistic regression model to the features $\boldsymbol{u} \circ \boldsymbol{v}$ to produce a weight vector β, and so the pair scoring function becomes $f(u, v) = \sigma(\beta \cdot (\boldsymbol{u} \circ \boldsymbol{v}))$ where $\sigma = \exp(x)/(1 + \exp(x))$ is the sigmoid function.

In order to compute the weight vector, β, in the Hadamard product case, we select a training set of size $50n$ by choosing 50 vertices which participate in a community with at least twenty members, $v_1, \ldots, v_{50}$, and compute $\boldsymbol{u} \circ \boldsymbol{v}_i$ for each $u \in V$. This collection of $50n$ feature vectors, along with ground truth labeling of community pairs, are given to an sklearn LogisticRegression object which produces β.

3.3.3 Datasets We show the performance of the various embedding methods contrasted with `LR-Structural` on three datasets: two publicly available real world datasets with ground truth community labels, and synthetic stochastic block models (SBM). We provide details on these datasets in the full version.

- dblp: a co-authorship network of 317K computer science authors with communities defined as venues
- amazon: a network of 335K products on amazon with a link representing frequent co-purchasing. Communities are product categories.
- sbm: synthetic dataset with 100K vertices and communities of size 20. Edges are randomly generated with an inter-community edge probability of 0.3 and intra-community edge probability of $0.3/n$.

4 Stochastic block models

We complement our experimental results on real datasets with measuring performance of graph embedding methods on synthetic datasets generated according to the popular Stochastic Block Model (SBM).

An n-vertex graph, G, is generated according to an SBM by partitioning the vertex set, $[n]$ into k equal sized communities, $C_1, C_2, \ldots, C_k$. The distribution of edges is controlled by two parameters. If two vertices are in the same community, an edge is added with probability p. If not, then an edge is added with probability q.

We chose the SBM parameters in order to approximate some of the features we observed in the empirical datasets. All SBMs have one hundred thousand vertices and communities (or "blocks") of size 20. The parameter q is set so that the average number of neighbors a vertex has inside its community is equal to that outside of its community. In real data, there are often 3 or 4 times as many inter-community neighbors as intra-community neighbors. The three SBM graphs generated, sbm_sparse, sbm, sbm_dense, have average degrees 4, 12 and 20 respectively.

Motivated by Theorem 2.2, we can study the embeddings' resilience to noise in the simulated setting. Fig. 3 shows the results from this experiment. Note that the accuracy of the embeddings at pairwise community labeling decreases as the internal density of the communities decreases. In all cases, `LR-Structural` is able to outperform the factorization-based embedding methods across a regime of parameters commonly encountered in sparse datasets.

References

[CAEHP+20] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv:2005.03675*, 2020.

[CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[CLX15] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning graph representations with global structural information. In *Conference on Information and Knowledge Management (CIKM)*, pages 891–900. ACM Press, 2015.

[CMST20] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos E Tsourakakis. Node embeddings and exact low-rank representations of complex networks. *arXiv:2006.05592*, 2020.

[DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *International Conference on Machine Learning (ICML)*, pages 233–240, 2006.

[EK10] David Easley and Jon Kleinberg. *Networks, crowds, and markets*, volume 8. Cambridge university press Cambridge, 2010.

[Ger20] Gershgorin circle theorem. `https://en.wikipedia.org/wiki/Gershgorin_circle_theorem`, 2020.

[GJJ20] Vikas K Garg, Stefanie Jegelka, and Tommi

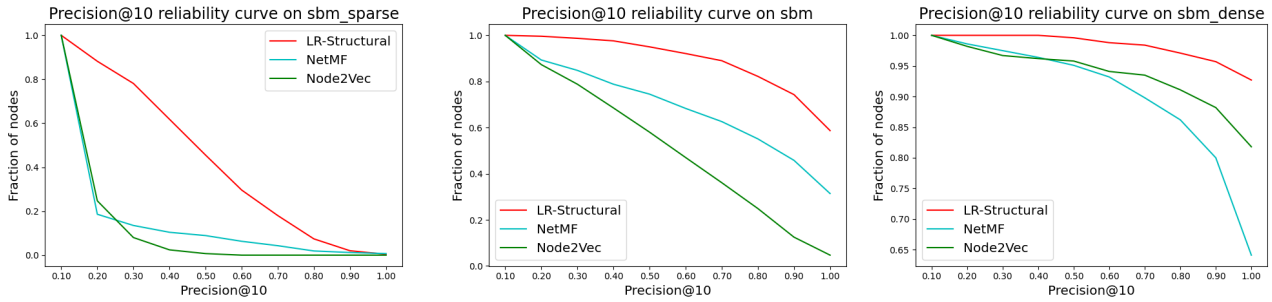

Figure 3: Precision@10 reliability curves for sbm datasets. All datasets are stochastically generated datasets with size 10,000 and disjoint communities of size 20. The average degrees are 4, 12 and 20 from left to right. Curves are generated from 100 samples.

Jaakkola. Generalization and representational limits of graph neural networks. *arXiv:2002.06157*, 2020.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864. ACM, 2016.

[HLG+20] Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Neural Information Processing Systems (NeurIPS)*, page 11, 2017.

[HYL18] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. page 24, 2018.

[Lou20] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.

[LRU20] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[Mur21] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2021.

[OCP+16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1105–1114. ACM, 2016.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 701–710. ACM Press, 2014.

[QDM+18] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Conference on Web Science and Data Mining (WSDM)*. ACM Press, 2018.

[RAL07] Fan Chung Reid Andersen and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4:35–64, 2007.

[RKS20] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Conference on Information and Knowledge Management (CIKM)*. ACM, 2020.

[SSG17] Aneesh Sharma, C. Seshadhri, and Ashish Goel. When hashes met wedges: A distributed algorithm for finding high similarity vectors. In *Conference on the World Wide Web (WWW)*, page 431440, 2017.

[SSSG20] C. Seshadhri, Aneesh Sharma, Andrew Stolman, and Ashish Goel. The impossibility of low-rank representations for triangle-rich complex networks. *Proceedings of the National Academy of Sciences*, 117(11):5631–5637, 2020.

[TW17] N Kipf Thomas and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2, 2017.

[Twi18] Embeddings@twitter. `https://blog.twitter.com/engineering/en_us/topics/insights/2018/embeddingsattwitter.html`, 2018.

[VCC+18] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.