Inductive Representation Learning on Large Graphs

William L. Hamilton* wleif@stanford.edu
Rex Ying* rexying@stanford.edu
Jure Leskovec jure@cs.stanford.edu

Department of Computer Science
Stanford University
Stanford, CA, 94305

Abstract

Low-dimensional embeddings of nodes in large graphs have proved extremely useful in a variety of prediction tasks, from content recommendation to identifying protein functions. However, most existing approaches require that all nodes in the graph are present during training of the embeddings; these previous approaches are inherently *transductive* and do not naturally generalize to unseen nodes. Here we present GraphSAGE, a general *inductive* framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Instead of training individual embeddings for each node, we learn a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. Our algorithm outperforms strong baselines on three inductive node-classification benchmarks: we classify the category of unseen nodes in evolving information graphs based on citation and Reddit post data, and we show that our algorithm generalizes to completely unseen graphs using a multi-graph dataset of protein-protein interactions.

1 Introduction

Low-dimensional vector embeddings of nodes in large graphs[1] have proved extremely useful as feature inputs for a wide variety of prediction and graph analysis tasks [5, 11, 28, 35, 36]. The basic idea behind node embedding approaches is to use dimensionality reduction techniques to distill the high-dimensional information about a node's neighborhood into a dense vector embedding. These node embeddings can then be fed to downstream machine learning systems and aid in tasks such as node classification, clustering, and link prediction [11, 28, 35].

However, previous works have focused on embedding nodes from a single fixed graph, and many real-world applications require embeddings to be quickly generated for unseen nodes, or entirely new (sub)graphs. This inductive capability is essential for high-throughput, production machine learning systems, which operate on evolving graphs and constantly encounter unseen nodes (e.g., posts on Reddit, users and videos on Youtube). An inductive approach to generating node embeddings also facilitates generalization across graphs with the same form of features: for example, one could train an embedding generator on protein-protein interaction graphs derived from a model organism, and then easily produce node embeddings for data collected on new organisms using the trained model.

The inductive node embedding problem is especially difficult, compared to the transductive setting, because generalizing to unseen nodes requires "aligning" newly observed subgraphs to the node embeddings that the algorithm has already optimized on. An inductive framework must learn to

*The two first authors made equal contributions.

[1] While it is common to refer to these data structures as social or biological *networks*, we use the term *graph* to avoid ambiguity with neural network terminology.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

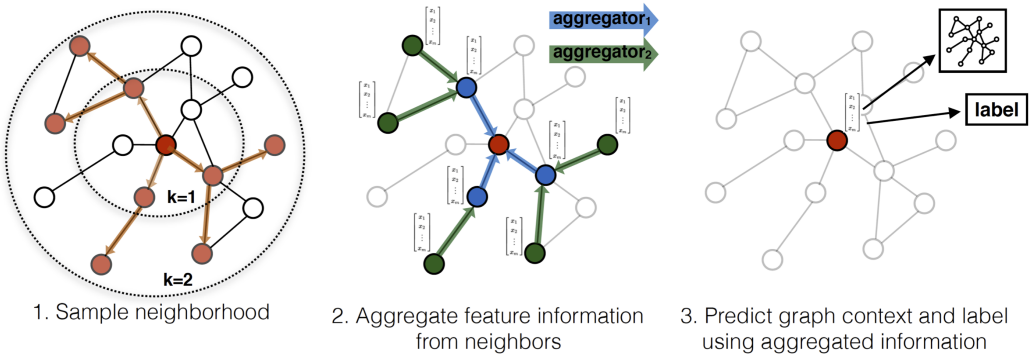


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

recognize structural properties of a node's neighborhood that reveal both the node's local role in the graph, as well as its global position.

Most existing approaches to generating node embeddings are inherently transductive. The majority of these approaches directly optimize the embeddings for each node using matrix-factorization-based objectives, and do not naturally generalize to unseen data, since they make predictions on nodes in a single, fixed graph [5, 11, 23, 28, 35, 36, 37, 39]. These approaches can be modified to operate in an inductive setting (e.g., [28]), but these modifications tend to be computationally expensive, requiring additional rounds of gradient descent before new predictions can be made. There are also recent approaches to learning over graph structures using convolution operators that offer promise as an embedding methodology [17]. So far, graph convolutional networks (GCNs) have only been applied in the transductive setting with fixed graphs [17, 18]. In this work we both extend GCNs to the task of inductive unsupervised learning and propose a framework that generalizes the GCN approach to use trainable aggregation functions (beyond simple convolutions).

Present work. We propose a general framework, called GraphSAGE (SAmple and aggreGatE), for inductive node embedding. Unlike embedding approaches that are based on matrix factorization, we leverage node features (e.g., text attributes, node profile information, node degrees) in order to learn an embedding function that generalizes to unseen nodes. By incorporating node features in the learning algorithm, we simultaneously learn the topological structure of each node's neighborhood as well as the distribution of node features in the neighborhood. While we focus on feature-rich graphs (e.g., citation data with text attributes, biological data with functional/molecular markers), our approach can also make use of structural features that are present in all graphs (e.g., node degrees). Thus, our algorithm can also be applied to graphs without node features.

Instead of training a distinct embedding vector for each node, we train a set of *aggregator functions* that learn to aggregate feature information from a node's local neighborhood (Figure 1). Each aggregator function aggregates information from a different number of hops, or search depth, away from a given node. At test, or inference time, we use our trained system to generate embeddings for entirely unseen nodes by applying the learned aggregation functions. Following previous work on generating node embeddings, we design an unsupervised loss function that allows GraphSAGE to be trained without task-specific supervision. We also show that GraphSAGE can be trained in a fully supervised manner.

We evaluate our algorithm on three node-classification benchmarks, which test GraphSAGE's ability to generate useful embeddings on unseen data. We use two evolving document graphs based on citation data and Reddit post data (predicting paper and post categories, respectively), and a multi-graph generalization experiment based on a dataset of protein-protein interactions (predicting protein functions). Using these benchmarks, we show that our approach is able to effectively generate representations for unseen nodes and outperform relevant baselines by a significant margin: across domains, our supervised approach improves classification F1-scores by an average of 51% compared to using node features alone and GraphSAGE consistently outperforms a strong, transductive baseline [28], despite this baseline taking $\sim$100$\times$ longer to run on unseen nodes. We also show that the new aggregator architectures we propose provide significant gains (7.4% on average) compared to an aggregator inspired by graph convolutional networks [17]. Lastly, we probe the expressive capability of our approach and show, through theoretical analysis, that GraphSAGE is capable of learning structural information about a node's role in a graph, despite the fact that it is inherently based on features (Section 5).

2 Related work

Our algorithm is conceptually related to previous node embedding approaches, general supervised approaches to learning over graphs, and recent advancements in applying convolutional neural networks to graph-structured data.[2]

Factorization-based embedding approaches. There are a number of recent node embedding approaches that learn low-dimensional embeddings using random walk statistics and matrix factorization-based learning objectives [5, 11, 28, 35, 36]. These methods also bear close relationships to more classic approaches to spectral clustering [23], multi-dimensional scaling [19], as well as the PageRank algorithm [25]. Since these embedding algorithms directly train node embeddings for individual nodes, they are inherently transductive and, at the very least, require expensive additional training (e.g., via stochastic gradient descent) to make predictions on new nodes. In addition, for many of these approaches (e.g., [11, 28, 35, 36]) the objective function is invariant to orthogonal transformations of the embeddings, which means that the embedding space does not naturally generalize between graphs and can drift during re-training. One notable exception to this trend is the Planetoid-I algorithm introduced by Yang et al. [40], which is an inductive, embedding-based approach to semi-supervised learning. However, Planetoid-I does not use any graph structural information during inference; instead, it uses the graph structure as a form of regularization during training. Unlike these previous approaches, we leverage feature information in order to train a model to produce embeddings for unseen nodes.

Supervised learning over graphs. Beyond node embedding approaches, there is a rich literature on supervised learning over graph-structured data. This includes a wide variety of kernel-based approaches, where feature vectors for graphs are derived from various graph kernels (see [32] and references therein). There are also a number of recent neural network approaches to supervised learning over graph structures [7, 10, 21, 31]. Our approach is conceptually inspired by a number of these algorithms. However, whereas these previous approaches attempt to classify entire graphs (or subgraphs), the focus of this work is generating useful representations for individual nodes.

Graph convolutional networks. In recent years, several convolutional neural network architectures for learning over graphs have been proposed (e.g., [4, 9, 8, 17, 24]). The majority of these methods do not scale to large graphs or are designed for whole-graph classification (or both) [4, 9, 8, 24]. However, our approach is closely related to the graph convolutional network (GCN), introduced by Kipf et al. [17, 18]. The original GCN algorithm [17] is designed for semi-supervised learning in a transductive setting, and the exact algorithm requires that the full graph Laplacian is known during training. A simple variant of our algorithm can be viewed as an extension of the GCN framework to the inductive setting, a point which we revisit in Section 3.3.

3 Proposed method: GraphSAGE

The key idea behind our approach is that we learn how to aggregate feature information from a node's local neighborhood (e.g., the degrees or text attributes of nearby nodes). We first describe the GraphSAGE embedding generation (i.e., forward propagation) algorithm, which generates embeddings for nodes assuming that the GraphSAGE model parameters are already learned (Section 3.1). We then describe how the GraphSAGE model parameters can be learned using standard stochastic gradient descent and backpropagation techniques (Section 3.2).

3.1 Embedding generation (i.e., forward propagation) algorithm

In this section, we describe the embedding generation, or forward propagation algorithm (Algorithm 1), which assumes that the model has already been trained and that the parameters are fixed. In particular, we assume that we have learned the parameters of K aggregator functions (denoted $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$), which aggregate information from node neighbors, as well as a set of weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$, which are used to propagate information between different layers of the model or "search depths". Section 3.2 describes how we train these parameters.

[2] In the time between this papers original submission to NIPS 2017 and the submission of the final, accepted (i.e., "camera-ready") version, there have been a number of closely related (e.g., follow-up) works published on pre-print servers. For temporal clarity, we do not review or compare against these papers in detail.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

The intuition behind Algorithm 1 is that at each iteration, or search depth, nodes aggregate information from their local neighbors, and as this process iterates, nodes incrementally gain more and more information from further reaches of the graph.

Algorithm 1 describes the embedding generation process in the case where the entire graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and features for all nodes $\mathbf{x}_v, \forall v \in \mathcal{V}$, are provided as input. We describe how to generalize this to the minibatch setting below. Each step in the outer loop of Algorithm 1 proceeds as follows, where k denotes the current step in the outer loop (or the depth of the search) and $\mathbf{h}^k$ denotes a node's representation at this step: First, each node $v \in \mathcal{V}$ aggregates the representations of the nodes in its immediate neighborhood, $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$, into a single vector $\mathbf{h}_{\mathcal{N}(v)}^{k-1}$. Note that this aggregation step depends on the representations generated at the previous iteration of the outer loop (i.e., $k-1$), and the $k = 0$ ("base case") representations are defined as the input node features. After aggregating the neighboring feature vectors, GraphSAGE then concatenates the node's current representation, $\mathbf{h}_v^{k-1}$, with the aggregated neighborhood vector, $\mathbf{h}_{\mathcal{N}(v)}^{k-1}$, and this concatenated vector is fed through a fully connected layer with nonlinear activation function σ, which transforms the representations to be used at the next step of the algorithm (i.e., $\mathbf{h}_v^k, \forall v \in \mathcal{V}$). For notational convenience, we denote the final representations output at depth K as $\mathbf{z}_v \equiv \mathbf{h}_v^K, \forall v \in \mathcal{V}$. The aggregation of the neighbor representations can be done by a variety of aggregator architectures (denoted by the AGGREGATE placeholder in Algorithm 1), and we discuss different architecture choices in Section 3.3 below.

To extend Algorithm 1 to the minibatch setting, given a set of input nodes, we first forward sample the required neighborhood sets (up to depth K) and then we run the inner loop (line 3 in Algorithm 1), but instead of iterating over all nodes, we compute only the representations that are necessary to satisfy the recursion at each depth (Appendix A contains complete minibatch pseudocode).

Relation to the Weisfeiler-Lehman Isomorphism Test. The GraphSAGE algorithm is conceptually inspired by a classic algorithm for testing graph isomorphism. If, in Algorithm 1, we (i) set $K = |\mathcal{V}|$, (ii) set the weight matrices as the identity, and (iii) use an appropriate hash function as an aggregator (with no non-linearity), then Algorithm 1 is an instance of the Weisfeiler-Lehman (WL) isomorphism test, also known as "naive vertex refinement" [32]. If the set of representations $\{\mathbf{z}_v, \forall v \in \mathcal{V}\}$ output by Algorithm 1 for two subgraphs are identical then the WL test declares the two subgraphs to be isomorphic. This test is known to fail in some cases, but is valid for a broad class of graphs [32]. GraphSAGE is a continuous approximation to the WL test, where we replace the hash function with trainable neural network aggregators. Of course, we use GraphSAGE to generate useful node representations–not to test graph isomorphism. Nevertheless, the connection between GraphSAGE and the classic WL test provides theoretical context for our algorithm design to learn the topological structure of node neighborhoods.

Neighborhood definition. In this work, we uniformly sample a fixed-size set of neighbors, instead of using full neighborhood sets in Algorithm 1, in order to keep the computational footprint of each batch

fixed.[3] That is, using overloaded notation, we define $\mathcal{N}(v)$ as a fixed-size, uniform draw from the set $\{u \in \mathcal{V} : (u, v) \in \mathcal{E}\}$, and we draw different uniform samples at each iteration, k, in Algorithm 1. Without this sampling the memory and expected runtime of a single batch is unpredictable and in the worst case $O(|\mathcal{V}|)$. In contrast, the per-batch space and time complexity for GraphSAGE is fixed at $O(\prod_{i=1}^{K} S_i)$, where $S_i, i \in \{1, ..., K\}$ and K are user-specified constants. Practically speaking we found that our approach could achieve high performance with $K = 2$ and $S_1 \cdot S_2 \leq 500$ (see Section 4.4 for details).

3.2 Learning the parameters of GraphSAGE

In order to learn useful, predictive representations in a fully unsupervised setting, we apply a graph-based loss function to the output representations, $\mathbf{z}_u, \forall u \in \mathcal{V}$, and tune the weight matrices, $\mathbf{W}^k, \forall k \in \{1, ..., K\}$, and parameters of the aggregator functions via stochastic gradient descent. The graph-based loss function encourages nearby nodes to have similar representations, while enforcing that the representations of disparate nodes are highly distinct:

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right), \tag{1}$$

where v is a node that co-occurs near u on fixed-length random walk, σ is the sigmoid function, P_n is a negative sampling distribution, and Q defines the number of negative samples. Importantly, unlike previous embedding approaches, the representations $\mathbf{z}_u$ that we feed into this loss function are generated from the features contained within a node's local neighborhood, rather than training a unique embedding for each node (via an embedding look-up).

This unsupervised setting emulates situations where node features are provided to downstream machine learning applications, as a service or in a static repository. In cases where representations are to be used only on a specific downstream task, the unsupervised loss (Equation 1) can simply be replaced, or augmented, by a task-specific objective (e.g., cross-entropy loss).

3.3 Aggregator Architectures

Unlike machine learning over N-D lattices (e.g., sentences, images, or 3-D volumes), a node's neighbors have no natural ordering; thus, the aggregator functions in Algorithm 1 must operate over an unordered set of vectors. Ideally, an aggregator function would be symmetric (i.e., invariant to permutations of its inputs) while still being trainable and maintaining high representational capacity. The symmetry property of the aggregation function ensures that our neural network model can be trained and applied to arbitrarily ordered node neighborhood feature sets. We examined three candidate aggregator functions:

Mean aggregator. Our first candidate aggregator function is the mean operator, where we simply take the elementwise mean of the vectors in $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$. The mean aggregator is nearly equivalent to the convolutional propagation rule used in the transductive GCN framework [17]. In particular, we can derive an inductive variant of the GCN approach by replacing lines 4 and 5 in Algorithm 1 with the following:[4]

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}). \tag{2}$$

We call this modified mean-based aggregator *convolutional* since it is a rough, linear approximation of a localized spectral convolution [17]. An important distinction between this convolutional aggregator and our other proposed aggregators is that it does not perform the concatenation operation in line 5 of Algorithm 1—i.e., the convolutional aggregator does concatenate the node's previous layer representation $\mathbf{h}_v^{k-1}$ with the aggregated neighborhood vector $\mathbf{h}_{\mathcal{N}(v)}^k$. This concatenation can be viewed as a simple form of a "skip connection" [13] between the different "search depths", or "layers" of the GraphSAGE algorithm, and it leads to significant gains in performance (Section 4).

LSTM aggregator. We also examined a more complex aggregator based on an LSTM architecture [14]. Compared to the mean aggregator, LSTMs have the advantage of larger expressive capability. However, it is important to note that LSTMs are not inherently symmetric (i.e., they are not permutation invariant), since they process their inputs in a sequential manner. We adapt LSTMs to operate on an unordered set by simply applying the LSTMs to a random permutation of the node's neighbors.

[3] Exploring non-uniform samplers is an important direction for future work.

[4] Note that this differs from Kipf et al's exact equation by a minor normalization constant [17].

Pooling aggregator. The final aggregator we examine is both symmetric and trainable. In this *pooling* approach, each neighbor's vector is independently fed through a fully-connected neural network; following this transformation, an elementwise max-pooling operation is applied to aggregate information across the neighbor set:

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\}), \tag{3}$$

where $\max$ denotes the element-wise max operator and σ is a nonlinear activation function. In principle, the function applied before the max pooling can be an arbitrarily deep multi-layer perceptron, but we focus on simple single-layer architectures in this work. This approach is inspired by recent advancements in applying neural network architectures to learn over general point sets [29]. Intuitively, the multi-layer perceptron can be thought of as a set of functions that compute features for each of the node representations in the neighbor set. By applying the max-pooling operator to each of the computed features, the model effectively captures different aspects of the neighborhood set. Note also that, in principle, any symmetric vector function could be used in place of the $\max$ operator (e.g., an element-wise mean). We found no significant difference between max- and mean-pooling in developments test and thus focused on max-pooling for the rest of our experiments.

4 Experiments

We test the performance of GraphSAGE on three benchmark tasks: (i) classifying academic papers into different subjects using the Web of Science citation dataset, (ii) classifying Reddit posts as belonging to different communities, and (iii) classifying protein functions across various biological protein-protein interaction (PPI) graphs. Sections 4.1 and 4.2 summarize the datasets, and the supplementary material contains additional information. In all these experiments, we perform predictions on nodes that are not seen during training, and, in the case of the PPI dataset, we test on entirely unseen graphs.

Experimental set-up. To contextualize the empirical results on our inductive benchmarks, we compare against four baselines: a random classifer, a logistic regression feature-based classifier (that ignores graph structure), the DeepWalk algorithm [28] as a representative factorization-based approach, and a concatenation of the raw features and DeepWalk embeddings. We also compare four variants of GraphSAGE that use the different aggregator functions (Section 3.3). Since, the "convolutional" variant of GraphSAGE is an extended, inductive version of Kipf et al's semi-supervised GCN [17], we term this variant GraphSAGE-GCN. We test unsupervised variants of GraphSAGE trained according to the loss in Equation (1), as well as supervised variants that are trained directly on classification cross-entropy loss. For all the GraphSAGE variants we used rectified linear units as the non-linearity and set $K = 2$ with neighborhood sample sizes $S_1 = 25$ and $S_2 = 10$ (see Section 4.4 for sensitivity analyses).

For the Reddit and citation datasets, we use "online" training for DeepWalk as described in Perozzi et al. [28], where we run a new round of SGD optimization to embed the new test nodes before making predictions (see the Appendix for details). In the multi-graph setting, we cannot apply DeepWalk, since the embedding spaces generated by running the DeepWalk algorithm on different disjoint graphs can be arbitrarily rotated with respect to each other (Appendix D).

All models were implemented in TensorFlow [1] with the Adam optimizer [16] (except DeepWalk, which performed better with the vanilla gradient descent optimizer). We designed our experiments with the goals of (i) verifying the improvement of GraphSAGE over the baseline approaches (i.e., raw features and DeepWalk) and (ii) providing a rigorous comparison of the different GraphSAGE aggregator architectures. In order to provide a fair comparison, all models share an identical implementation of their minibatch iterators, loss function and neighborhood sampler (when applicable). Moreover, in order to guard against unintentional "hyperparameter hacking" in the comparisons between GraphSAGE aggregators, we sweep over the same set of hyperparameters for all GraphSAGE variants (choosing the best setting for each variant according to performance on a validation set). The set of possible hyperparameter values was determined on early validation tests using subsets of the citation and Reddit data that we then discarded from our analyses. The appendix contains further implementation details.[5]

[5] Code and links to the datasets: `http://snap.stanford.edu/graphsage/`

Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

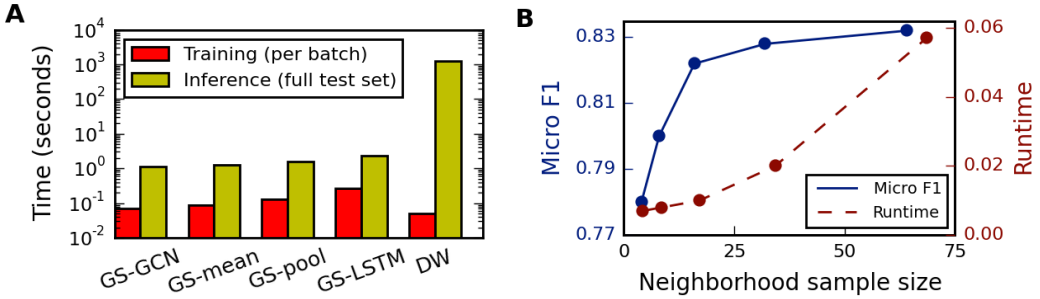


Figure 2: **A**: Timing experiments on Reddit data, with training batches of size 512 and inference on the full test set (79,534 nodes). **B**: Model performance with respect to the size of the sampled neighborhood, where the "neighborhood sample size" refers to the number of neighbors sampled at each depth for $K = 2$ with $S_1 = S_2$ (on the citation data using GraphSAGE-mean).

4.1 Inductive learning on evolving graphs: Citation and Reddit data

Our first two experiments are on classifying nodes in evolving information graphs, a task that is especially relevant to high-throughput production systems, which constantly encounter unseen data.

Citation data. Our first task is predicting paper subject categories on a large citation dataset. We use an undirected citation graph dataset derived from the Thomson Reuters Web of Science Core Collection, corresponding to all papers in six biology-related fields for the years 2000-2005. The node labels for this dataset correspond to the six different field labels. In total, this is dataset contains 302,424 nodes with an average degree of 9.15. We train all the algorithms on the 2000-2004 data and use the 2005 data for testing (with 30% used for validation). For features, we used node degrees and processed the paper abstracts according Arora et al.'s [2] sentence embedding approach, with 300-dimensional word vectors trained using the GenSim word2vec implementation [30].

Reddit data. In our second task, we predict which community different Reddit posts belong to. Reddit is a large online discussion forum where users post and comment on content in different topical communities. We constructed a graph dataset from Reddit posts made in the month of September, 2014. The node label in this case is the community, or "subreddit", that a post belongs to. We sampled 50 large communities and built a post-to-post graph, connecting posts if the same user comments on both. In total this dataset contains 232,965 posts with an average degree of 492. We use the first 20 days for training and the remaining days for testing (with 30% used for validation). For features, we use off-the-shelf 300-dimensional GloVe CommonCrawl word vectors [27]; for each post, we concatenated (i) the average embedding of the post title, (ii) the average embedding of all the post's comments (iii) the post's score, and (iv) the number of comments made on the post.

The first four columns of Table 1 summarize the performance of GraphSAGE as well as the baseline approaches on these two datasets. We find that GraphSAGE outperforms all the baselines by a significant margin, and the trainable, neural network aggregators provide significant gains compared

to the GCN approach. For example, the unsupervised variant GraphSAGE-pool outperforms the concatenation of the DeepWalk embeddings and the raw features by 13.8% on the citation data and 29.1% on the Reddit data, while the supervised version provides a gain of 19.7% and 37.2%, respectively. Interestingly, the LSTM based aggregator shows strong performance, despite the fact that it is designed for sequential data and not unordered sets. Lastly, we see that the performance of unsupervised GraphSAGE is reasonably competitive with the fully supervised version, indicating that our framework can achieve strong performance without task-specific fine-tuning.

4.2 Generalizing across graphs: Protein-protein interactions

We now consider the task of generalizing across graphs, which requires learning about node roles rather than community structure. We classify protein roles—in terms of their cellular functions from gene ontology—in various protein-protein interaction (PPI) graphs, with each graph corresponding to a different human tissue [41]. We use positional gene sets, motif gene sets and immunological signatures as features and gene ontology sets as labels (121 in total), collected from the Molecular Signatures Database [34]. The average graph contains 2373 nodes, with an average degree of 28.8. We train all algorithms on 20 graphs and then average prediction F1 scores on two test graphs (with two other graphs used for validation).

The final two columns of Table 1 summarize the accuracies of the various approaches on this data. Again we see that GraphSAGE significantly outperforms the baseline approaches, with the LSTM- and pooling-based aggregators providing substantial gains over the mean- and GCN-based aggregators[6]

4.3 Runtime and parameter sensitivity

Figure 2 A summarizes the training and test runtimes for the different approaches. The training time for the methods are comparable (with GraphSAGE-LSTM being the slowest). However, the need to sample new random walks and run new rounds of SGD to embed unseen nodes makes DeepWalk 100-500$\times$ slower at test time.

For the GraphSAGE variants, we found that setting $K = 2$ provided a consistent boost in accuracy of around 10-15%, on average, compared to $K = 1$; however, increasing K beyond 2 gave marginal returns in performance (0-5%) while increasing the runtime by a prohibitively large factor of 10-100$\times$, depending on the neighborhood sample size. We also found diminishing returns for sampling large neighborhoods (Figure 2 B). Thus, despite the higher variance induced by sub-sampling neighborhoods, GraphSAGE is still able to maintain strong predictive accuracy, while significantly improving the runtime.

4.4 Summary comparison between the different aggregator architectures

Overall, we found that the LSTM- and pool-based aggregators performed the best, in terms of both average performance and number of experimental settings where they were the top-performing method (Table 1). To give more quantitative insight into these trends, we consider each of the six different experimental settings (i.e., (3 datasets) $\times$ (unsupervised vs. supervised)) as trials and consider what performance trends are likely to generalize. In particular, we use the non-parametric Wilcoxon Signed-Rank Test [33] to quantify the differences between the different aggregators across trials, reporting the T-statistic and p-value where applicable. Note that this method is rank-based and essentially tests whether we would expect one particular approach to outperform another in a new experimental setting. Given our small sample size of only 6 different settings, this significance test is somewhat underpowered; nonetheless, the T-statistic and associated p-values are useful quantitative measures to assess the aggregators' relative performances.

We see that LSTM-, pool- and mean-based aggregators all provide statistically significant gains over the GCN-based approach ($T = 1.0$, $p = 0.02$ for all three). However, the gains of the LSTM and pool approaches over the mean-based aggregator are more marginal ($T = 1.5$, $p = 0.03$, comparing

[6]Note that in very recent follow-up work Chen and Zhu [6] achieve superior performance by optimizing the GraphSAGE hyperparameters specifically for the PPI task and implementing new training techniques (e.g., dropout, layer normalization, and a new sampling scheme). We refer the reader to their work for the current state-of-the-art numbers on the PPI dataset that are possible using a variant of the GraphSAGE approach.

LSTM to mean; $T = 4.5$, $p = 0.10$, comparing pool to mean). There is no significant difference between the LSTM and pool approaches ($T = 10.0$, $p = 0.46$). However, GraphSAGE-LSTM is significantly slower than GraphSAGE-pool (by a factor of $\approx 2\times$), perhaps giving the pooling-based aggregator a slight edge overall.

5 Theoretical analysis

In this section, we probe the expressive capabilities of GraphSAGE in order to provide insight into how GraphSAGE can learn about graph structure, even though it is inherently based on features. As a case-study, we consider whether GraphSAGE can learn to predict the clustering coefficient of a node, i.e., the proportion of triangles that are closed within the node's 1-hop neighborhood [38]. The clustering coefficient is a popular measure of how clustered a node's local neighborhood is, and it serves as a building block for many more complicated structural motifs [3]. We can show that Algorithm 1 is capable of approximating clustering coefficients to an arbitrary degree of precision:

Theorem 1. *Let $\mathbf{x}_v \in U, \forall v \in \mathcal{V}$ denote the feature inputs for Algorithm 1 on graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where U is any compact subset of $\mathbb{R}^d$. Suppose that there exists a fixed positive constant $C \in \mathbb{R}^+$ such that $\|\mathbf{x}_v - \mathbf{x}_{v'}\|_2 > C$ for all pairs of nodes. Then we have that $\forall \epsilon > 0$ there exists a parameter setting $\mathbf{\Theta}^*$ for Algorithm 1 such that after $K = 4$ iterations*

$$|z_v - c_v| < \epsilon, \forall v \in \mathcal{V},$$

where $z_v \in \mathbb{R}$ are final output values generated by Algorithm 1 and c_v are node clustering coefficients.

Theorem 1 states that for any graph there exists a parameter setting for Algorithm 1 such that it can approximate clustering coefficients in that graph to an arbitrary precision, if the features for every node are distinct (and if the model is sufficiently high-dimensional). The full proof of Theorem 1 is in the Appendix. Note that as a corollary of Theorem 1, GraphSAGE can learn about local graph structure, even when the node feature inputs are sampled from an absolutely continuous random distribution (see the Appendix for details). The basic idea behind the proof is that if each node has a unique feature representation, then we can learn to map nodes to indicator vectors and identify node neighborhoods. The proof of Theorem 1 relies on some properties of the pooling aggregator, which also provides insight into why GraphSAGE-pool outperforms the GCN and mean-based aggregators.

6 Conclusion

We introduced a novel approach that allows embeddings to be efficiently generated for unseen nodes. GraphSAGE consistently outperforms state-of-the-art baselines, effectively trades off performance and runtime by sampling node neighborhoods, and our theoretical analysis provides insight into how our approach can learn about local graph structures. A number of extensions and potential improvements are possible, such as extending GraphSAGE to incorporate directed or multi-modal graphs. A particularly interesting direction for future work is exploring non-uniform neighborhood sampling functions, and perhaps even learning these functions as part of the GraphSAGE optimization.

Acknowledgments

The authors thank Austin Benson, Aditya Grover, Bryan He, Dan Jurafsky, Alex Ratner, Marinka Zitnik, and Daniel Selsam for their helpful discussions and comments on early drafts. The authors would also like to thank Ben Johnson for his many useful questions and comments on our code. This research has been supported in part by NSF IIS-1149837, DARPA SIMPLEX, Stanford Data Science Initiative, Huawei, and Chan Zuckerberg Biohub. W.L.H. was also supported by the SAP Stanford Graduate Fellowship and an NSERC PGS-D grant. The views and conclusions expressed in this material are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the above funding agencies, corporations, or the U.S. and Canadian governments.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint* , 2016.

[2] S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.

[3] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

[5] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *KDD*, 2015.

[6] J. Chen and J. Zhu. Stochastic training of graph convolutional networks. *arXiv preprint arXiv:1710.10568*, 2017.

[7] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

[9] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015.

[10] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 729–734, 2005.

[11] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[12] W. L. Hamilton, J. Leskovec, and D. Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. In *ACL*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *EACV*, 2016.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.

[18] T. N. Kipf and M. Welling. Variational graph auto-encoders. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

[19] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[20] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2015.

[22] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[23] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. In *NIPS*, 2001.

[24] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[28] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[30] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC*, 2010.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[32] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[33] S. Siegal. *Nonparametric statistics for the behavioral sciences*. McGraw-hill, 1956.

[34] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[35] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[36] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *KDD*, 2016.

[37] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI*, 2017.

[38] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[39] L. Xu, X. Wei, J. Cao, and P. S. Yu. Embedding identity and interest for social networks. In *WWW*, 2017.

[40] Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

[41] M. Zitnik and J. Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):190–198, 2017.

Anonymous Walk Embeddings

Sergey Ivanov [1 2] **Evgeny Burnaev** [1]

Abstract

The task of representing entire graphs has seen a surge of prominent results, mainly due to learning convolutional neural networks (CNNs) on graph-structured data. While CNNs demonstrate state-of-the-art performance in graph classification task, such methods are supervised and therefore steer away from the original problem of network representation in task-agnostic manner. Here, we coherently propose an approach for embedding entire graphs and show that our feature representations with SVM classifier increase classification accuracy of CNN algorithms and traditional graph kernels. For this we describe a recently discovered graph object, *anonymous walk*, on which we design task-independent algorithms for learning graph representations in explicit and distributed way. Overall, our work represents a new scalable unsupervised learning of state-of-the-art representations of entire graphs.

1. Introduction

A wide range of real world applications deal with network analysis and classification tasks. An ease of representing data with graphs makes them very valuable asset in any data mining toolbox; however, the complexity of working with graphs led researchers to seek for new ways of representing and analyzing graphs, of which network embeddings have become broadly popular due to their success in several machine learning areas such as graph classification (Cai et al., 2017), visualization (Cao et al., 2016), and pattern recognition (Monti et al., 2017).

Essentially, network embeddings are vector representations of graphs that capture local and global traits and, as a consequence, are more suitable for standard machine learning techniques such as SVM that works on numerical vectors rather than graph structures. Ideally, a practitioner would like to have a *polynomial*-time algorithm that can convert different graphs into different feature vectors. However, such algorithm would be capable of deciding whether two graphs are isomorphic (Gärtner et al., 2003), for which currently only quasipolynomial-time algorithm exists (Babai, 2016). Hence, there are fundamental challenges in the design of polynomial-time algorithm for network-to-vector conversion. Instead, a lot of research was devoted to the question of designing network embedding models that are computationally efficient *and* preserve similarity between graphs.

Broadly speaking, network embeddings come from one of the two buckets, either based on engineered graph features or driven by training on graph data. Feature-based methods traditionally appeared in graph kernel setting (Vishwanathan et al., 2010), where each graph is decomposed into discrete components, distribution of which is used as a vector representation of a graph (Haussler, 1999). Importantly, general concept of feature-based methods implies ad-hoc knowledge about the data at hand. For example, Random Walk kernel (Vishwanathan et al., 2010) assumes that graph realization originates from the types of random walks a graph has, whereas for Weisfeiler-Lehman (WL) kernel (Shervashidze et al., 2011) the insight is in subtree patterns of a graph. For high-dimensional graph embeddings feature-based methods produce sparse solution as only few substructures are common across graphs. This is known as *diagonal dominance* (Yanardag & Vishwanathan, 2015), a situation when a graph representation is only similar to itself, but not to any other graph.

On the other hand, data-driven approach learns network embeddings by optimizing some form of objective function defined on graph data. Deep Graph Kernels (DGK) (Yanardag & Vishwanathan, 2015), for example, learns a positive semidefinite matrix that weights the relationship between graph substructures, while Patchy-San (PSCN) (Niepert et al., 2016) constructs locally connected neighborhoods for training a convolutional neural network on. Data-driven approach implies learning *distributed* graph representations that have demonstrated promising classification results (Niepert et al., 2016; Tixier et al., 2017).

Our approach. We propose to use a natural graph object

[1]Skolkovo Institute of Science and Technology, Moscow, Russia [2]Criteo Research, Paris, France. Correspondence to: Sergey Ivanov <sergei.ivanov@skolkovotech.ru>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

named *anonymous walk* as a base for learning feature-based and data-driven network embeddings. Recent discovery (Micali & Allen Zhu, 2016) has shown that anonymous walks provide characteristic graph traits and are capable to reconstruct network proximity of a node *exactly*. In particular, distribution of anonymous walks starting at node u is sufficient for reconstruction of a subgraph induced by all vertices within a fixed distance from u; and such distribution uniquely determines underlying Markov processes from u, i.e. no two different subgraphs exist having the same distribution of anonymous walks. This implies that two graphs with similar distributions of anonymous walks should be topologically similar. We therefore define feature-based network embeddings on distribution of anonymous walks and show an efficient sampling approach that approximates distributions for large networks.

To overcome sparsity of feature-based methods, we design a data-driven approach that learns distributed representations on the generated corpus of anonymous walks via backpropagation, in the same vein as neural models in NLP (Le & Mikolov, 2014; Bengio et al., 2003). Considering anonymous walks for the same source node as co-occurring words in the sentence and graph as a collection of such sentences, the hope is that by predicting a target word in a given context of words and a document, the proposed algorithm learns semantic meaning of words and a document.

To the best of our knowledge, we are the first to introduce anonymous walks in the context of learning network representations and we highlight the following contributions:

- Based on the notion of anonymous walk, we propose feature-based network embeddings, for which we describe an efficient sampling procedure to alleviate time complexity of exact computation.
- By maximizing the likelihood of preserving network proximity of anonymous walks, we propose a scalable algorithm to learn data-driven network embeddings.
- On widely-used real datasets, we demonstrate that our network embeddings achieve state-of-the-art performance in comparison with other graph kernels and neural networks in graph classification task.

2. Anonymous Walks

Random walks are the sequences of nodes, where each new node is selected independently from the set of neighbors of the last node in the sequence. Normally states in a random walk correspond to a label or a global name of a node; however, for reasons described below such states could be unavailable. Yet, recently it has been shown that anonymized version of a random walk can provide a flexible way to reconstruct a network even when global names are absent (Micali & Allen Zhu, 2016). We next define a notion of anonymous walk.

Definition 1. Let $s = (u_1, u_2, \dots, u_k)$ be an ordered list of elements $u_i \in V$. We define the positional function pos: $(s, u_i) \mapsto q$ such that for any ordered list $s = (u_1, u_2, \dots, u_k)$ and an element $u_i \in V$ it returns a list $q = (p_1, p_2, \dots, p_l)$ of all positions $p_j \in \mathbb{N}$ of u_i occurrences in a list s.

For example, if $s = (a, b, c, b, c)$, then $pos(s, a) = (1)$ as element a appears only on the first position and $pos(s, b) = (2, 4)$.

Definition 2 (Anonymous Walk)**.** If $w = (v_1, v_2, \dots, v_k)$ is a random walk, then its corresponding *anonymous walk* is the sequence of integers $a = (f(v_1), f(v_2), \dots, f(v_k))$, where integer $f(v_i) = \min_{p_j \in pos(w, v_i)} pos(w, v_i)$.

We denote mapping of a random walk w to anonymous walk a by $w \mapsto a$.

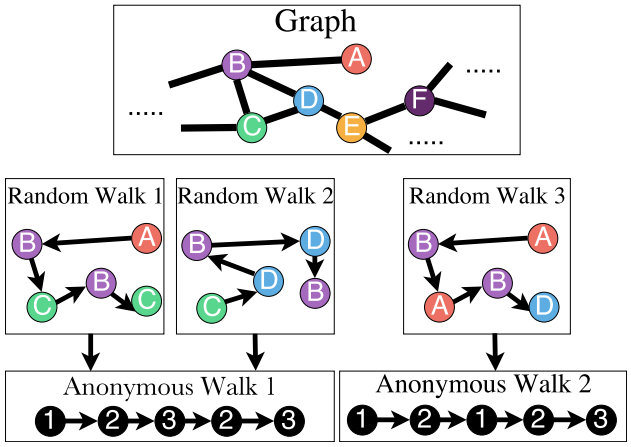


Figure 1. An example demonstrating the concept of anonymous walk. Two different random walks 1 and 2 of the graph correspond to the *same* anonymous walk 1. A random walk 3 corresponds to *another* anonymous walk 2.

For instance, in the graph of Fig. 1 a random walk $a \to b \to c \to b \to c$ matches anonymous walk $1 \to 2 \to 3 \to 2 \to 3$. Likewise, another random walk $c \to d \to b \to d \to b$ also corresponds to anonymous walk $1 \to 2 \to 3 \to 2 \to 3$. Conversely, another random walk $a \to b \to a \to b \to d$ corresponds to a different anonymous walk $1 \to 2 \to 1 \to 2 \to 3$.

Intuitively, states in anonymous walk correspond to the first position of the node in a random walk and their total number equals to the number of distinct nodes in a random walk. Particular name of the state does not matter (so, for example, anonymous walk $1 \to 2 \to 3$ would be the same as anonymous walk $3 \to 1 \to 2$); however, by agreement,

anonymous walks start from 1 and continue to name new states by incrementing the current maximum state in an anonymous walk.

Rationale. From the perspective of a single node, in the position of an observer, global topology of the network may be hidden deliberately (e.g. social networks often restrict outsiders to examine your friendships) or otherwise (e.g. newly created links in the world wide web may be yet unknown to the search engine). Nevertheless, an observer can, on his own, experiment with the network by starting a random walk from itself, passing the process to its neighbors and recording the observed states in a random walk. As global names of the nodes are not available to an observer, one way to record the states *anonymously* is by describing them by the first occurrence of a node in a random walk. Not only are such records succinct, but it is common to have privacy constraints (Abraham, 2012) that would not allow to record a full description of nodes.

Somewhat remarkably, (Micali & Allen Zhu, 2016) show that for a single node u in a graph G, a known distribution $\mathfrak{D}_l$ over anonymous walks of length l is sufficient to reconstruct topology of the ball $B(u, r)$ with the center at u and radius r, i.e. the subgraph of graph G induced by all vertices distanced at most r hops from u. For the task of learning embeddings, the topology of network is available and thus distribution of anonymous walks $\mathfrak{D}_l$ can be computed precisely. As no two different subgraphs can have the same distribution $\mathfrak{D}_l$, it is useful to generalize distribution of anonymous walks from a single node to the whole network and use it as a feature representation of a graph. This idea paves the way to our feature-based network embeddings.

3. Algorithms

We start from discussion of leveraging anonymous walks for learning network embeddings in a feature-based manner. Inspired by empirical results we train an objective function on local neighborhoods of anonymous walks, which further improves results of classification.

3.1. AWE: Feature-Based model

By definition, a weighted directed graph is a tuple $G = (V, E, \Omega)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices, $E \subseteq V \times V$ is a set of edges, and $\Omega \subset \mathbb{R}$ is a set of edge weights. Given graph G we construct a *random walk graph* $R = (V, E, P)$ such that every edge $e = (u, v)$ has a weight p_e equals to $\omega_e / \sum_{v \in N_{out}(u)} \omega_{(u,v)}$, where $N_{out}(u)$ is the set of out-neighbors of u and $\omega_e \in \Omega$. A random walk w with length l on graph R is a sequence of nodes $u_1, u_2, \ldots, u_{l+1}$, where $u_i \in V$, such that a pair (u_i, u_{i+1}) is selected with a probability $p_{(u_i,u_{i+1})}$ in a random walk graph R. A probability $p(w)$ of having a random walk w is the total probability of choosing the edges in a random walk, i.e. $p(w) = \prod_{e \in w} p_e$.

According to the Definition 1, anonymous walk is a random walk, where each state is recorded by its first occurrence index in the random walk. The number of all possible anonymous walks of length l in an arbitrary graph grows exponentially with l (Figure 2). Consider an initial node u and a set of all different random walks W_l^u that start from u and have length l. These random walks correspond to a set of η different anonymous walks $\mathcal{A}_l^u = (a_1^u, a_2^u, \ldots, a_\eta^u)$. A probability of seeing anonymous walk a_i^u of length l for a node u is $p(a_i^u) = \sum_{\substack{w \in W_l^u \\ w \mapsto a_i}} p(w)$. Aggregating probabilities across all vertices in a graph and normalizing them by the total number of nodes N, we get the probability of choosing anonymous walk a_i in graph G:

$$p(a_i) = \frac{1}{N} \sum_{u \in G} p(a_i^u) = \frac{1}{N} \sum_{u \in G} \sum_{\substack{w \in W_l^u \\ w \mapsto a_i}} p(w).$$

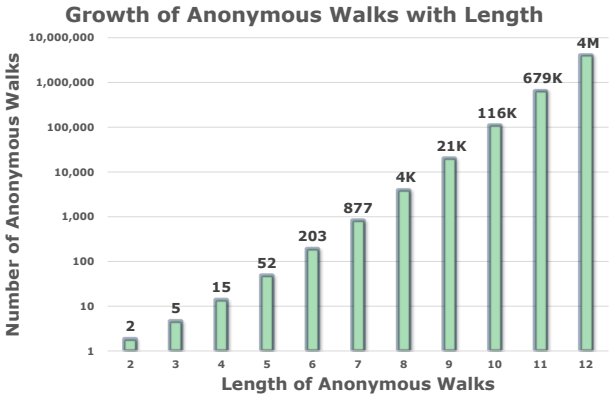


Figure 2. Y-axis is in log scale. The number of different anonymous walks increases exponentially with length of walks l.

We are now ready to define network embeddings that we name feature-based anonymous walk embeddings (AWE).

Definition 3 (feature-based AWE)**.** Let $\mathcal{A}_l = (a_1, a_2, \ldots, a_\eta)$ be the set of all possible anonymous walks of length l. *Anonymous walk embedding* of a graph G is the vector f_G of size η, whose i-th component corresponds to a probability $p(a_i)$, of having anonymous walk a_i in a graph G:

$$f_G = (p(a_1), p(a_2), \ldots, p(a_\eta)). \quad (1)$$

Direct computation of AWE relies on the enumeration of all different random walks in graph G, which is shown below to grow exponentially with the number of steps l.

Theorem 1. The running time of Anonymous Walk Embeddings (eq. 1) is $\mathcal{O}(nl(d_{in}^{max}(v) \cdot d_{out}^{max}(v))^{l/2})$, where $d_{in/out}^{max}$ is the maximum in/out degree in graph G with n vertices.

Proof. Let k_l be the number of random walks of length l in a directed graph. According to (Tubig, 2012) k_l can be bounded by the powers of in- and out-degrees of nodes in G:

$$k_l^2 \leq (\sum_{v \in G} d_{in}^l(v))(\sum_{v \in G} d_{out}^l(v)).$$

Hence, the number of random walks in a graph is at most $n(d_{in}^{max}(v) \cdot d_{out}^{max}(v))^{l/2}$, where $d_{in/out}^{max}$ is the maximum in/out degree. As it requires $\mathcal{O}(l)$ operations to map one random walk of length l to anonymous walk, the theorem follows. □

Sampling. As complete counting of all anonymous walks in a large graph may be infeasible, we describe a sampling approach to approximate the true distribution. In this fashion, we draw independently a set of m random walks and calculate its corresponding empirical distribution of anonymous walks. To guarantee that empirical and actual distributions are close with a given confidence, we set the number m of random walks sufficiently large.

More formally, let $\mathcal{A}_l = (a_1, a_2, \ldots, a_\eta)$ be the set of all possible anonymous walks of length l. For two discrete probability distributions P and Q on set $\mathcal{A}_l$, define L_1 distance as:

$$\|P - Q\|_1 = \sum_{a_i \in \mathcal{A}} |P(a_i) - Q(a_i)|$$

For a graph G let $\mathfrak{D}_l$ be the actual distribution of anonymous walks $\mathcal{A}_l$ of length l and let $X^m = (X_1, X_2, \ldots, X_m)$ be i.i.d. random variables drawn from $\mathfrak{D}_l$. The empirical distribution $\mathfrak{D}^m$ of the original distribution $\mathfrak{D}_l$ is defined as:

$$\mathfrak{D}^m(i) = \frac{1}{m} \sum_{X_j \in X^m} [\![X_j = a_i]\!],$$

where $[\![x]\!] = 1$ if x is true and 0 otherwise.

Then, for all $\varepsilon > 0$ and $\delta \in [0, 1]$ the number of samples m to satisfy $P\{\|\mathfrak{D}^m - \mathfrak{D}\|_1 \geq \varepsilon\} \leq \delta$ equals to (from (Shervashidze et al., 2009)):

$$m = \left\lceil \frac{2}{\varepsilon^2}(\log(2^\eta - 2) - \log(\delta)) \right\rceil. \tag{2}$$

For example, there are $\eta = 877$ possible anonymous walks with length $l = 7$ (Figure 2). If we set $\varepsilon = 0.5$ and $\delta = 0.05$, then $m = 4888$. If we decrease $\varepsilon = 0.1$ and $\delta = 0.01$, then the number of samples will increase to 122500.

As transition probabilities for random walks can be preprocessed, sampling of a node in a random walk of length l can be done in $\mathcal{O}(1)$ via alias method. Hence, the overall running time of sampling approach to compute feature-based anonymous walk embeddings is $\mathcal{O}(ml)$.

Our experimental study shows state-of-the-art classification accuracy of feature-based AWE on real datasets. We continue to design data-driven approach that eliminates the sparsity of feature-based embeddings.

3.2. AWE: data-driven model

Our approach for learning network embeddings is analogous to methods for learning paragraph vectors in a text corpus (Le & Mikolov, 2014). In our case, an anonymous walk is a word, a randomly sampled set of anonymous walks starting from the same node is a set of co-occurring words, and a graph is a document.

Neighborhoods of anonymous walks. To leverage the analogy from NLP, we first need to generate a corpus of co-occurring anonymous walks in a graph G. We define a neighborhood between two anonymous walks of length l if they share the same source node. This is similar to other methods such as shortest-paths co-occurrence in DGK (Yanardag & Vishwanathan, 2015) and rooted subgraphs neighborhood in graph2vec (Narayanan et al., 2017), which proved to be successful in empirical studies. Therefore, we iterate over each vertex u in a graph G, sampling T random walks $(w_1^u, w_2^u, \ldots, w_T^u)$ that start at node u and map to a sequence of co-occurred anonymous walks $s^u = (a_1^u, a_2^u, \ldots, a_T^u)$, i.e. $w_i^u \mapsto a_i^u$. A collection of all s^u for all vertices $u \in G$ is a corpus of co-occurred anonymous walks in a graph and is analogous to a collection of sentences in a document.

Training. In this framework, we learn representation vector d of a graph and anonymous walks matrix W (see Figure 3). Vector d has $1 \times d_g$ size, where d_g is embedding size of a graph. Matrix W has $\eta \times d_a$ size, where η is the number of all possible anonymous walks of length l and d_a is embedding size of anonymous walk. For convenience, we call d as a document vector and W as a word matrix. Each graph corresponds to its vector d and an anonymous walk corresponds to a row in a matrix W. The model tries to predict a target anonymous walk given co-occurring context anonymous walks and a graph.

Formally, a sequence of co-occurred anonymous walks $s = (a_1, a_2, \ldots, a_T)$ corresponds to vectors $\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_T$ of matrix W, and a graph G corresponds to vector d. We aim to maximize the average log probability:

$$\frac{1}{T} \sum_{t=\Delta}^{T-\Delta} \log p(\mathrm{w}_t | \mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}, \mathrm{d}), \tag{3}$$

where Δ is a window size, i.e. number of context words for each target word. Probability in objective (3) is defined via softmax function:

$$p(\mathrm{w}_t|\mathrm{w}_{t-\Delta},\ldots,\mathrm{w}_{t+\Delta},\mathrm{d}) = \frac{e^{y(\mathrm{w}_t)}}{\sum_{i=1}^{\eta} e^{y(\mathrm{w}_i)}} \quad (4)$$

Each $y(\mathrm{w}_t)$ is unnormalized log probability for output word i:

$$y(\mathrm{w}_t) = b + Uh(\mathrm{w}_{t-\Delta},\ldots,\mathrm{w}_{t+\Delta},\mathrm{d})$$

where $b \in \mathbb{R}$ and $U \in \mathbb{R}^{d_a+d_g}$ are softmax parameters. Vector h is constructed by first averaging walk vectors $\mathrm{w}_{t-\Delta},\ldots,\mathrm{w}_{t+\Delta}$ and then concatenating with a graph vector d. The reason is that since anonymous walks are randomly sampled, we average vectors $\mathrm{w}_{t-\Delta},\ldots,\mathrm{w}_{t+\Delta}$ to compensate for the lack of knowledge on the order of walks; and at the same time, the graph vector d is shared among multiple (context, target) pairs.

To avoid computation of the sum in softmax equation (4), which becomes impractical for large sets of anonymous walks, one can use Hierarchical softmax (Mikolov et al., 2013b) or NCE loss functions (Gutmann & Hyvärinen, 2010) to speed up training. In our work, we use sampled softmax (Jean et al., 2015) that for each training example picks only a fraction of vocabulary according to a chosen sampling function. One can measure distribution of anonymous walks in a graph via means of definition 1 and decide on a corresponding sampling function.

At every step of the model, we sample context and target anonymous walks from a graph and compute the gradient error from prediction of target walk and update vectors of context walks and a graph via gradient backpropagation. When given several networks to embed, one can reuse word matrix W across graphs, thereby sharing previously learned embeddings of walks.

Summarizing, after initialization of matrix W for all anonymous walks of length l and a graph vector d, the model repeats the following two steps for all nodes in a graph: 1) for sampled co-occurred anonymous walks the model calculates a loss (Eq. 3) of predicting a target walk (one of the sampled anonymous walks) by considering all context walks and a graph; 2) the model updates the vectors of context walks in matrix W and graph vector d via gradient backpropagation. One step of the model is depicted in Figure 3. After using up all sampled corpus, a learned graph vector d is called *anonymous walk embedding*.

Definition 4 (data-driven AWE)**.** *Anonymous walk embedding* of a graph G is a vector representation d learned on a corpus of sampled anonymous walks from a graph G.

So despite the fact that graph and walk vectors are initialized randomly, as an indirect result of predicting a walk in the context of other walks and a graph the model also learns feature representations of networks. Intuitively, a graph vector can be thought as a word with a special meaning: it serves as an overall summary for all anonymous walks in the graph.

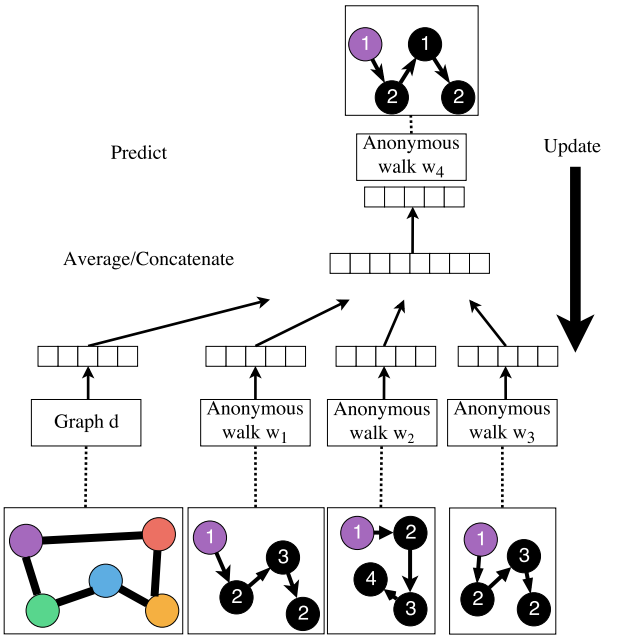


Figure 3. A framework for learning data-driven anonymous walk embeddings. Graph is represented by a vector d and anonymous walks are represented by rows of matrix W. All co-occurring anonymous walks start from the same node in a graph. The goal is to predict a target walk w_4 by its surrounding context walks $(\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3)$ and a graph vector d. We average embeddings of context walks and then concatenate with a graph vector to predict a target vector. Vectors are updated using stochastic gradient descent on a corpus of sampled anonymous walks.

In our experiments, we show how anonymous walk network embeddings can be used in graph classification problem, demonstrating state-of-the-art performance in classification accuracy.

4. Graph Classification

Graph classification is a task to predict a class label of a whole graph and it has found applications in bioinformatics (Nikolentzos et al., 2017) and malware detection (Narayanan et al., 2017). In this task, given a series of N graphs $\{G_i\}_{i=1}^N$ and their corresponding labels $\{L_i\}_{i=1}^N$, we are asked to train a model m: $G \mapsto L$ that would efficiently classify new graphs. Two typical approaches to graph classification problem are (1) supervised learning classification algorithms such as PSCN algorithm (Niepert et al., 2016) and (2) graph kernel methods such as WL kernel (Sher-

vashidze et al., 2011). As we are interested in designing task-agnostic network embeddings that do not require labeled data during training, we show how to use anonymous walk embeddings in conjunction with kernel methods to perform classification of new graphs. For this we define a kernel function on two graphs.

Definition 5 (Kernel function)**.** *Kernel function* is a symmetric, positive semidefinite function k: $X \times X \mapsto \mathbb{R}^n$ defined for a non-empty set X.

When $X \subseteq \mathbb{R}^n$, several popular choices of kernel exist (Schölkopf & Smola, 2002):

- **Inner product** $k(x, y) = \langle x, y \rangle, \forall x, y \in \mathbb{R}^n$,
- **Polynomial** $k(x, y) = (\langle x, y \rangle + c)^d, \forall x, y \in \mathbb{R}^n$,
- **RBF** $k(x, y) = \exp(-\frac{\|x - y\|_2^2}{2\sigma^2}), \forall x, y \in \mathbb{R}^n$.

With network embeddings, it is then easy to define a kernel function on two graphs:

$$K(G_1, G_2) = k(f(G_1), f(G_2)), \tag{5}$$

where $f(G_i)$ is an embedding of a graph G_i and k: $(x, y) \mapsto \mathbb{R}^n$ is a kernel function.

To train a graph classifier m one can then construct a square kernel matrix $\mathcal{K}$ for training data $G_1, G_2, \ldots, G_N$ and feed this matrix to a kernelized algorithm such as SVM. Every element of kernel matrix equals to: $\mathcal{K}_{ij} = K(G_i, G_j)$. For classifying new test instance G_τ, one would first compute graph kernels with training instances $(K(G_1, G_\tau), K(G_2, G_\tau), \ldots, K(G_N, G_\tau))$ and provide it to a trained classifier m.

In our experiments, we use anonymous walk embeddings to compute kernel matrices and show that kernelized SVM classifier achieves top performance comparing to more complex state-of-the-art models.

5. Experiments

We evaluate our embeddings on the task of graph classification for variety of widely-used datasets.

Datasets. We evaluate performance on two sets of graphs. One set contains *unlabeled* graph data and is related to social networks (Yanardag & Vishwanathan, 2015). Another set contains graphs with labels on node and/or edges and originates from bioinformatics (Shervashidze et al., 2011). Statistics of these ten graph datasets presented in Table 1.

Evaluation. We train a multiclass SVM classifier with one-vs-one scheme. We perform a 10-fold cross-validation and for each fold we estimate SVM parameter C from the range [0.001, 0.01, 0.1, 1, 10] using validation set. This process is repeated 10 times and an average accuracy is reported, i.e. the average number of correctly classified test graphs.

Table 1. Graph datasets used in classification experiments. The columns are: Name of dataset, Number of graphs, Number of classes (maximum number of graphs in a class), Average number of nodes/edges.

Dataset	Source	Graphs	Classes (Max)	Nodes Avg.	Edges Avg.
COLLAB	Social	5000	3 (2600)	74.49	4914.99
IMDB-B	Social	1000	2 (500)	19.77	193.06
IMDB-M	Social	1500	3 (500)	13	131.87
RE-B	Social	2000	2 (1000)	429.61	995.50
RE-M5K	Social	4999	5 (1000)	508.5	1189.74
RE-M12K	Social	12000	11 (2592)	391.4	913.78
Enzymes	Bio	600	6 (100)	32.6	124.3
DD	Bio	1178	2 (691)	284.31	715.65
Mutag	Bio	188	2 (125)	17.93	19.79

Competitors. PSCN is a convolutional neural network algorithm (Niepert et al., 2016) with size of receptive field equals to 10. PSCN is the state-of-the-art instance of neural network algorithms, which has achieved strong classification accuracy in many datasets, and we use the best reported accuracy for these algorithms. GK is a graphlet kernel (Shervashidze et al., 2009) and DGK is a deep graphlet kernel (Yanardag & Vishwanathan, 2015) with graphlet size equals to 7. WL is Weisfeiler-Lehman graph kernel algorithm (Shervashidze et al., 2011) with height of subtree pattern equals to 7. WL proved consistenly strong results comparing to other graph kernels and supervised algorithms. ER is exponential random walk kernel (Gärtner et al., 2003) with exponent equals to 0.5 and kR is k-step random walk kernel with $k = 3$ (Sugiyama & Borgwardt, 2015).

Setup. For feature-based anonymous walk embeddings (Def. 1), we choose length l of walks from the range $[2, 3, \ldots, 10]$ and approximate actual distribution of anonymous walks using sampling equation (2) with $\varepsilon = 0.1$ and $\delta = 0.05$.

For data-driven anonymous walk embeddings (Def. 4), we set length of walks $l = 10$ to generate a corpus of co-occurred anonymous walks. We run gradient descent with 100 iterations for 100 epochs with batch size that we vary from the range [100, 500, 1000, 5000, 10000]. Context walks are drawn from a window, which size varies in the range [2, 4, 8, 16]. The embedding size of walks and graphs d_a and d_g equals to 128. Finally, candidate sampling function for softmax equation (4) chooses uniform or loguniform distribution of sampled classes.

To perform classification, we compute a kernel matrix, where Inner product, Polynomial, and RBF kernels are tested. For RBF kernel function we choose parameter σ from the range $[10^{-5}, 10^{-4}, \ldots, 1, 10]$; for Polynomial

function we set $c = 0$ and $d = 2$. We run the experiments on a machine with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 32GB RAM[1]. We refer to our algorithms as AWE (DD) and AWE (FB) for data-driven and feature-based approaches correspondingly.

Classification results. Table 2 presents results on classification accuracy for Social unlabeled datasets. AWE approaches are consistently at the top, sharing top-2 results for all six social datasets, despite being unsupervised approach unlike PSCN. At the same time, Table 4 shows accuracy results for labeled bio datasets. Note that AWE are learned using only topology of the network and not node/edge labels. In this setting, embeddings obtained by AWE (FB) approach achieves competitive performance for the labeled datasets.

Overall observations.

- Tables 2 and 4 demonstrate that AWE is competitive to supervised state-of-the-art solutions in graph classification task. Importantly, even with simple classifiers such as SVM, AWE increases classification accuracy comparing to other more complex neural network models. Likewise, just comparing graph kernels, we can see that anonymous walks is at the top with tranditional graph objects such as graphlets (GK kernel) or subtree patterns (WL kernel).
- While feature-based and data-driven approaches are different in nature, the resulted classification accuracy is close across many datasets. As such, only on RE-B dataset data-driven approach has more than 5% increase in the accuracy. In practice, we found that using feature-based approach for small length l (e.g. ≤ 10) produces competitive results, while data-driven approach works best for large number of iterations and length l.
- Polynomial and RBF kernel functions bring nonlinearity to the classification algorithm and are able to learn more complex classification boundaries. Table 3 shows that RBF and Polynomial kernels are well suited for feature-based and data-driven models respectively.

Scalability. To test for scalability, we learn network representations using AWE (DD) algorithm for Erdos-Renyi graphs with increasing sizes from [10, 10^1, 10^2, 10^3, 10^4, $3 \cdot 10^4$]. For each size we construct 10 Erdos-Renyi graphs with $\mu = np \in [2, 3, 4, 5]$, where n is the number of nodes and p is the probability of having an edge between two arbitrary nodes. In that case, a graph has $m \propto \mu n$ edges. We average time to train AWE (DD) embeddings across 10 graphs for every n and μ. Our setup: size of embeddings equals to 128, batch size equals to 100, window size equals to 100. We run AWE (DD) model for 100 iterations in one epoch. In Figure 4, we empirically observe that the model to learn AWE (DD) network representations scales to networks with tens of thousands of nodes and edges and requires no more than a few seconds to map a graph to a vector.

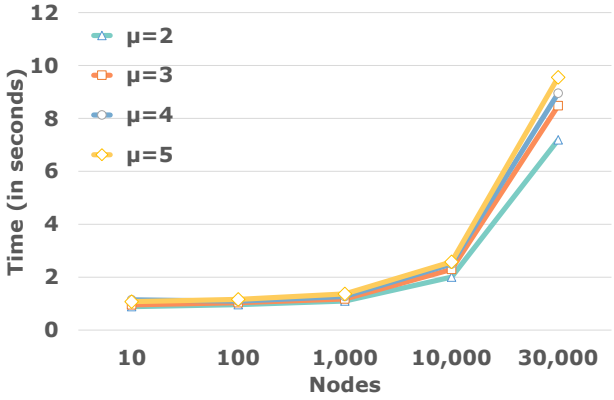


Figure 4. Average running time to generate anonymous walk embedding for Erdos-Renyi graphs, with $\mu = np \in [2, 3, 4, 5]$ where n is the number of nodes and p is probability parameter of Erdos-Renyi model. X-axis is in log scale.

Intuition behind performance. There is a couple of factors that leads anonymous walk embeddings to state-of-the-art performance in graph classification task. First, the use of anonymous walks is backed up by a recent discovery that, under certain condition, distribution of anonymous walks of a single node is sufficient to reconstruct a topology of the ball around a node. Hence, at least on a level of a single node, distribution of anonymous walk serves as a unique representation of subgraphs in a network. Second, data-driven approach reuses hitherto learned embeddings matrix W in previous iterations for learning embeddings of new graph instances. Therefore one can think of anonymous walks as words that have semantic meaning unified across all graphs. While learning graph embeddings, we simultaneously learn the meaning of different anonymous walks, which provides extra information for our model.

6. Related Work

Network representations were first studied in the context of graph kernels (Gärtner et al., 2003) and then have become a separate topic that found numerous applications beyond graph classification (Cai et al., 2017). Our feature-based embeddings originate from learning distribution on anonymous walks in a graph and is alike to the approach of graph kernels. Embeddings based on graph kernels include Ran-

[1]Code can be found at https://github.com/nd7141/AWE

Table 2. Comparison of classification accuracy (mean ± std., %) in Social datasets. Top-2 results are in **bold**. OOM is out-of-memory.

	Algorithm	IMDB-M	IMDB-B	COLLAB	RE-B	RE-M5K	RE-M12K
DD	AWE (DD)	**51.54 ± 3.61**	**74.45 ± 5.83**	**73.93 ± 1.94**	**87.89 ± 2.53**	**50.46 ± 1.91**	**39.20 ± 2.09**
	PSCN	45.23 ± 2.84	71.00 ± 2.29	72.60 ± 2.15	**86.30 ± 1.58**	49.10 ± 0.70	41.32 ± 0.32
	DGK	44.55 ± 0.52	66.96 ± 0.56	73.09 ± 0.25	78.04 ± 0.39	41.27 ± 0.18	32.22 ± 0.10
FB	AWE (FB)	**51.58 ± 4.66**	73.13 ± 3.28	70.99 ± 1.49	82.97 ± 2.86	**54.74 ± 2.93**	**41.51 ± 1.98**
	WL	49.33 ± 4.75	**73.4 ± 4.63**	**79.02 ± 1.77**	81.1 ± 1.9	49.44 ± 2.36	38.18 ± 1.3
	GK	43.89 ± 0.38	65.87 ± 0.98	72.84 ± 0.28	65.87 ± 0.98	41.01 ± 0.17	31.82 ± 0.08
	ER	OOM	64.00 ± 4.93	OOM	OOM	OOM	OOM
	kR	34.47 ± 2.42	45.8 ± 3.45	OOM	OOM	OOM	OOM

Table 3. Kernel function comparison in classification task (%).

Algorithm	IMDB-M	COLLAB	RE-B
AWE (DD) *RBF*	50.73	**73.93**	**87.89**
AWE (DD) *Inner*	**51.54**	73.77	84.82
AWE (DD) *Poly*	45.32	70.45	79.35
AWE (FB) *RBF*	**51.58**	**70.99**	**82.97**
AWE (FB) *Inner*	46.45	69.60	76.83
AWE (FB) *Poly*	46.57	64.3	67.22

Table 4. Classification accuracy (%) in labeled Bio datasets.

Algorithm	Enzymes	DD	Mutag
AWE	35.77 ± 5.93	71.51 ± 4.02	87.87 ± 9.76
PSCN	–	77.12 ± 2.41	92.63 ± 4.21
DGK	27.08 ± 0.79	–	82.66 ± 1.45
WL	53.15 ± 1.14	77.95 ± 0.70	80.72 ± 3.00
GK	32.70 ± 1.20	78.45 ± 0.26	81.58 ± 2.11
ER	14.97 ± 0.28	OOM	71.89 ± 0.66
kR	30.01 ± 1.01	OOM	80.05 ± 1.64

dom Walk (Gärtner et al., 2003), Graphlet (Shervashidze et al., 2009), Weisfeiler-Lehman (Shervashidze et al., 2011), Shortest-Path (Borgwardt & Kriegel, 2005) decompositions and all can be summarized as an instance of R-convolution framework (Haussler, 1999).

Distributed representations have become trendy after significant achievements in NLP applications (Mikolov et al., 2013a;b). Our data-driven network embeddings stem from paragraph-vector distributed-memory model (Le & Mikolov, 2014) that has become successful in learning document representations. Other related approaches include Deep Graph Kernel (Yanardag & Vishwanathan, 2015) that learns a matrix for graph kernel that encodes relationship between substructures; PSCN (Niepert et al., 2016) and 2D CNN (Tixier et al., 2017) algorithms that learn convolutional neural networks on graphs; graph2vec (Narayanan et al., 2017) learns network embeddings by extracting rooted subgraphs and training on skipgram negative sampling model (Mikolov et al., 2013b); FGSD (Verma & Zhang, 2017) that constructs feature vector from the histogram of the multiset of node pairwise distances. (Cai et al., 2017) provides a more comprehensive list of graph embeddings. Besides this, there is a list of aggregation techniques of node embeddings for the purpose of graph classification (Hamilton et al., 2017).

7. Conclusion

We described two unsupervised algorithms to compute network vector representations using anonymous walks. In the first approach, we use distribution of anonymous walks as a network embedding. As the exact calculation of network embeddings can be expensive we demonstrate how one can sample walks in a graph to approximate actual distribution with a given confidence. Next, we show how one can learn distributed graph representations in a data-driven manner, similar to learning paragraph vectors in NLP.

In our experiments, we show that our network embeddings even with simple SVM classifier achieve increase in classification accuracy comparing to state-of-the-art supervised neural network methods and graph kernels. This demonstrates that representation of your data can be more promising subject to study than the type and architecture of your predictive model.

Although the focus of this work was in representation of networks, AWE algorithm can be used to learn node, edge, or any subgraph representations by replacing graph vector with a corresponding subgraph vector. In all graph and subgraph representations, we expect data-driven approach to be a strong alternative to feature-based methods.

8. Acknowledgement

This work was supported by the Ministry of Education and Science of the Russian Federation (Grant no.14.756.31.0001).

References

Abraham, A. *Computational Social Networks: Security and Privacy*. Springer Publishing Company, Incorporated, 2012.

Babai, L. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pp. 684–697, 2016.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Borgwardt, K. M. and Kriegel, H. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pp. 74–81, 2005.

Cai, H., Zheng, V. W., and Chang, K. C. A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604, 2017. URL http://arxiv.org/abs/1709.07604.

Cao, S., Lu, W., and Xu, Q. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 1145–1152, 2016.

Gärtner, T., Flach, P. A., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pp. 129–143, 2003.

Gutmann, M. and Hyvärinen, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pp. 297–304, 2010.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.

Haussler, D. Convolution kernels on discrete structures. Technical report, 1999.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. On using very large target vocabulary for neural machine translation. In *ACL 2015*, pp. 1–10, 2015.

Le, Q. V. and Mikolov, T. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1188–1196, 2014.

Micali, S. and Allen Zhu, Z. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 200:108–122, 2016.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *CoRR*, 2013a. URL http://arxiv.org/abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems NIPS*, pp. 3111–3119, 2013b.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5425–5434, 2017.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. graph2vec: Learning distributed representations of graphs. In *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2014–2023, 2016.

Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. Matching node embeddings for graph similarity. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 2429–2435, 2017.

Schölkopf, B. and Smola, A. J. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., and Borgwardt, K. M. Efficient graphlet kernels for large graph comparison. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, pp. 488–495, 2009.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Sugiyama, M. and Borgwardt, K. M. Halting in random walk kernels. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1639–1647, 2015.

Tixier, A. J., Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. Classifying graphs as images with convolutional neural networks. *CoRR*, abs/1708.02218, 2017. URL http://arxiv.org/abs/1708.02218.

Tubig, H. The number of walks and degree powers in directed graphs. Technical report, Computer Science Department, Rutgers University, TUM-I123, TU Munich, 2012.

Verma, S. and Zhang, Z. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 87–97, 2017.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, August 2010. ISSN 1532-4435.

Yanardag, P. and Vishwanathan, S. V. N. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pp. 1365–1374, 2015.

Deep Variational Network Embedding in Wasserstein Space

Dingyuan Zhu*
Tsinghua University
zhudy11@126.com

Peng Cui
Tsinghua University
cuip@tsinghua.edu.cn

Daixin Wang
Tsinghua University
dxwang0826@gmail.com

Wenwu Zhu
Tsinghua University
wwzhu@tsinghua.edu.cn

ABSTRACT

Network embedding, aiming to embed a network into a low dimensional vector space while preserving the inherent structural properties of the network, has attracted considerable attentions recently. Most of the existing embedding methods embed nodes as point vectors in a low-dimensional continuous space. In this way, the formation of the edge is deterministic and only determined by the positions of the nodes. However, the formation and evolution of real-world networks are full of uncertainties, which makes these methods not optimal. To address the problem, we propose a novel Deep Variational Network Embedding in Wasserstein Space (**DVNE**) in this paper. The proposed method learns a Gaussian distribution in the Wasserstein space as the latent representation of each node, which can simultaneously preserve the network structure and model the uncertainty of nodes. Specifically, we use 2-Wasserstein distance as the similarity measure between the distributions, which can well preserve the transitivity in the network with a linear computational cost. Moreover, our method implies the mathematical relevance of mean and variance by the deep variational model, which can well capture the position of the node by the mean vectors and the uncertainties of nodes by the variance. Additionally, our method captures both the local and global network structure by preserving the first-order and second-order proximity in the network. Our experimental results demonstrate that our method can effectively model the uncertainty of nodes in networks, and show a substantial gain on real-world applications such as link prediction and multi-label classification compared with the state-of-the-art methods.

KEYWORDS

Network Embedding, Wasserstein space, Deep Learning

ACM Reference Format:
Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. 2018. Deep Variational Network Embedding in Wasserstein Space. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220052

*Beijing National Research Center for Information Science and Technology(BNRist)

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220052

1 INTRODUCTION

Network embedding has attracted considerable research attentions in the past few years. The basic idea is to embed a network into a low-dimensional vector space to preserve the network structure. Many network embedding methods are demonstrated to be effective in a variety of applications, such as link prediction [42, 44], classification [8, 26] and clustering [35, 46]. However, most of existing network embedding methods represent each node by a single point in a low-dimensional vector space. In this way, the formation of the whole network structure is deterministic.

Actually, real-world networks are much more complex than we assume. The formation and evolution of the networks are full of uncertainties. For example, for the nodes with low degree, they contain less information and thus their representations bear more uncertainties than others. For the nodes across multiple communities, the possible contradiction between their neighboring nodes may also be larger and thus cause the uncertainty. Furthermore, in social network, human behavior is multi-faceted which also makes the generation of edges uncertain [47]. For all of these cases, without considering the uncertainty of networks, the learned embeddings will be less effective in network analysis and inference tasks.

Gaussian distribution innately represents the uncertainty property [43]. Therefore, it is promising to represent a node by Gaussian distributions, i.e. the mean and the variance, rather than a point vector to incorporate the uncertainty. Motivated by this, to model the uncertainty of each node using Gaussian distributions, there are some basic requirements for network embedding methods to meet.

- **Transitivity**: The embedding space should be a metric space to preserve the transitivity in networks. Transitivity is a very important property in networks, especially in social networks [25]. For example, the friend of my friend is more likely to be my friend than some randomly chosen users. Moreover, the transitivity measures the density of triangles in a network, which plays an important role in calculating clustering coefficient [7]. If the metric space satisfies the triangle inequality, the transitivity in the network can be well preserved.
- **Uncertainty**: By using Gaussian distributions to represent a node, the mean and the variance should preserve different

properties to make such representations informative. Specifically, the mean vectors should reflect the position of the nodes and variance terms should contain the uncertainty of the nodes. In this way, the representations based on distributions can preserve the uncertainty while supporting network applications.
- **Strcutural Proximity**: The network structures, especially high-order proximity, should be preserved in a effective and efficient way. The high-order proximity is critical for capturing the network structure, which has been demonstrated to be useful in many real-world applications [36].

Recently, some works attempt to use Gaussian distributions to represent a node for network embedding [3, 17, 24] to integrate uncertainty. However, these methods use the Kullback-Leibler (abbreviated as KL) divergence [28] to measure the similarity between distributions. However, the KL divergence is asymmetric and does not satisfy triangle inequality. Thus, it can not well preserve the transitivity of proximity in networks, especially in undirected networks. Additionally, these methods regard the variance terms as additional dimensions of mean vectors, and use similarity measure to constrain their learning. In this way, they do not reflect the intrinsic relationship between variance terms and mean vectors in the model. Finally, very few of these works preserve the high-order proximity in network embedding, except Graph2Gauss [3]. But Graph2Gauss needs to calculate the shortest path between any two nodes, which is unaffordable in large-scale networks.

To address these problems, we propose a novel Deep Variational Network Embedding in Wasserstein Space method in this paper, named **DVNE**. The proposed method learns a Gaussian Embedding for each node in the Wasserstein Space by the deep variational model. Specifically, we employ 2-Wasserstein distance to measure the similarity between the distributions, i.e. the embeddings of the nodes. The 2-Wasserstein distance is a real metric that able to preserve the transitivity in embedding space. In this way, the proposed deep model is able to simultaneously preserve the transitivity and model the node uncertainty with linear time complexity. Meanwhile, we use a deep variational model to minimize the Wasserstein distance between the model distribution and the data distribution, which can extract the intrinsic relationship between mean vectors and variance terms. Furthermore, our method efficiently preserve the first-order and second-order proximity of the nodes in networks, empowering the learned node representations to reflect both local and global network structure [44].

The main contributions of our method are summarized as follows:

- We propose DVNE, an novel method that learns the Gaussian embedding in the Wasserstein space, which can well preserve the transitivity in networks and reflect the uncertainties of nodes.
- We imply the mathematical relevance of mean vectors and variance terms by the deep variational model, where the mean vectors denote the position of the nodes and the variance terms represent the uncertainties of the nodes.
- We efficiently preserve the first-order and second-order proximity between nodes, thus the learned representations capture the local and global network structure.
- We comprehensively evaluate the effectiveness of DVNE on several real-world networks in various applications.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we summarize the notations used in this paper and give the problem formulation. We introduce the framework of the method in Section 4 and report the experimental results in Section 5. We conclude the paper in Section 6.

2 RELATED WORK

Because of the popularity of networked data, network embedding has received more and more attentions in recent years. We briefly review some network embedding methods, and readers can referred to [13] for a comprehensive survey. Deepwalk [37] first uses the language modeling technique to learn the latent representations of a network by truncated random walks. LINE [39] embeds the network into a low-dimensional space where the first-order and second-order proximity between nodes are preserved. Node2vec [22] learns a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. HOPE [36] proposes a high-order proximity preserved embedding method. Furthermore, deep learning method for network embedding is also studied. SDNE [44] first considers the high nonlinearity in network embedding and proposes a deep autoencoder to preserve the first- and the second-order proximities. The graph variational autoencoder (GAE) [27] learns node embeddings in an unsupervised manner with variational autoencoder (VAE) [16].

All the aforementioned methods learn a point-vector for each node as its embedding. However, as we stated before, these methods have the limitation to model the uncertainty, which is a critical property needed to be considered for network embedding. Then some following works start to consider the uncertainty problem. Inspired by [43], which learns the Gaussian word embeddings to model uncertainty, KG2E [24] learns Gaussian embeddings for knowledge graphs. HCGE [17] similarly learns Gaussian embeddings for heterogeneous graphs. And Aleksandar et al. [3] proposes a deep model to learn Gaussian embeddings on the attributed network. All of these methods use the KL divergence or its variant JensenShannon divergence [19] as the similarity measure between the distributions. However, both the KL divergence and the JensenShannon divergence are not the true metrics. These metrics do not satisfy the triangle inequality. In this way, these methods cannot preserve the transitivity to get effective representations for networks. Furthermore, these methods regard the variance terms as the extra dimensions, then use the similarity measure to constrain their learning. In this way, it is difficult to capture the intrinsic relationships between the mean and the variance terms.

3 NOTATIONS AND PROBLEM DEFINITION

In this section, we summarize the notations used in this paper and give the problem formulation.

3.1 Notations

We first summarize the notations used in this paper. A network is defined as $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, where $\mathbf{V} = \{v_1, v_2, ..., v_N\}$ denotes a set of nodes and N is the number of the nodes. $\mathbf{E}$ is the set of edges

between the nodes, and $M = |\mathbf{E}|$ is the number of the edges. In this paper, we mainly consider undirected networks. Let $\mathbf{Nbrs}_i = \{v_j|(v_i, v_j) \in \mathbf{E}\}$ denote the set of neighbors of node v_i. Let $\mathbf{P} \in \mathbb{R}^{N\times N}$ be the transition matrix, where $\mathbf{P}(i,:)$ and $\mathbf{P}(:,j)$ denote its i^{th} row and j^{th} column respectively and $\mathbf{P}(i,j)$ is the element of the i^{th} row and j^{th} column. If there is an edge from v_i to v_j and the degree of node v_i is d_i, then we set $\mathbf{P}(i,j)$ to $\frac{1}{d_i}$, otherwise we mark $\mathbf{P}(i,j)$ with zero. We define $\mathbf{h}_i = \mathcal{N}(\mu_i, \Sigma_i)$ as a lower-dimensional Gaussian distribution embedding for node v_i, where $\mu_i \in \mathbb{R}^L, \Sigma_i \in \mathbb{R}^{L\times L}$. L is the embedding dimension, which satisfies $L \ll N$. In this paper, we focus on diagonal covariance matrices.

3.2 Problem Definition

In this paper, we focus on the problem of network embedding with first-order and second-order proximity preserved.

Definition 3.1. (First-Order Proximity) The first-order proximity describes the pairwise proximity between nodes. For any pair of nodes, if $\mathbf{P}(i,j) > 0$, there exists positive first-order proximity between v_i and v_j. Otherwise, the first-order proximity between v_i and v_j is 0.

The first-order proximity implies that two nodes in real-world networks are similar if they are linked by an observed edge. For example, if two users build a relationship between them on the social network, they may have a common interest. However, real-world networks are usually so sparse that we can only observe a very limited number of links. Only capturing the first-order proximity is not sufficient,thus we introduce the second-order proximity to capture the global network structure.

Definition 3.2. (Second-Order Proximity) The second-order proximity between a pair of nodes denotes the similarity between their neighborhood network structures. Then the second-order proximity between v_i and v_j is determined by the similarity between $\mathbf{Nbrs}_i$ and $\mathbf{Nbrs}_j$. If none of nodes is linked with both v_i and v_j, the second-order proximity between v_i and v_j is 0.

Intuitively, the second-order proximity assumes that if two nodes share common neighbors, they tend to be similar. The second-order proximity has been demonstrated to be a good metric to define the similarity of a pair of nodes, even if there is no edge between them [31]. Moreover, the second-order proximity has been proved to be able to alleviate the sparsity problem of the first-order proximity and better preserve the global structure of the network [39].

With the first- and second-order proximity, then we define our network embedding problem as follows:

Definition 3.3. (Gaussian-Based Network Embedding) Given a network $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, we aim to represent each node v_i as a lower-dimensional Gaussian distribution $\mathbf{h}_i = \mathcal{N}(\mu_i, \Sigma_i)$, where μ_i captures the position of the nodes in the embedding space and Σ_i investigates the uncertainty of the nodes. Meanwhile, the latent representations aim to preserve the first-order proximity and the second-order proximity between the nodes to preserve the network structure.

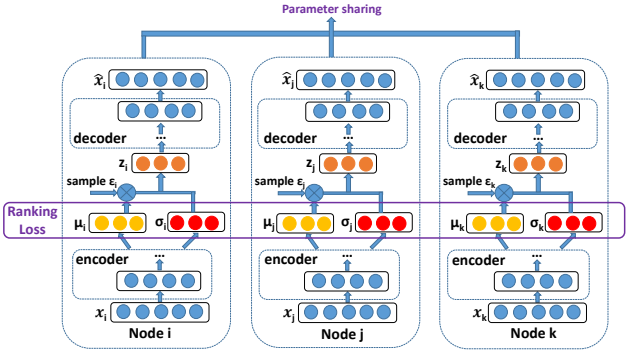


Figure 1: The framework of DVNE.

4 DEEP VARIATIONAL NETWORK EMBEDDING

4.1 Framework

In this paper, we propose a novel model to perform network embedding, namely DVNE, whose framework is shown in Figure 1. Basically, we propose a deep architecture, which is composed of multiple nonlinear mapping functions to map the input data to the Wasserstein space to preserve the uncertainties of the nodes and capture the network structure. Specifically, we first use a ranking based loss function on the Wasserstein embedding space, aiming to make nodes with edges similar and without edges dissimilar. In this way, the first-order proximity is preserved. Furthermore, we use a deep variational model to preserve the second-order proximity, by reconstructing the neighborhood structure of each node. Meanwhile, the whole deep variational model implies the the mathematical relevance of mean vectors and variance terms explicitly by the sampling process. In this way, the mean vectors find an approximate position of the node and the variance term capture the uncertainty. In the following sections, we will introduce how to realize the deep model in detail.

4.2 Similarity Measure

To support network applications, we need to define a suitable similarity measure between the latent representations of two nodes. Since we use distributions to represent our latent representations to incorporate uncertainty, the similarity measure should be able to measure the similarity between the distributions. Furthermore, as transitivity is a important property of the network, the similarity measure should simultaneously preserve the transitivity between nodes. Through extensive studies, we find that the Wasserstein distance is able to measure the similarity between two distributions while simultaneously satisfies the triangle inequality [9], which guarantees its ability to preserve the transitivity of similarity between nodes.

The p^{th} Wasserstein distance between two probability measures μ and ν is defined as:

$$W_p(\mu,\nu)^p = \inf \mathbb{E}\big[d(X,Y)^p\big], \tag{1}$$

where $\mathbb{E}[Z]$ denotes the expected value of a random variable Z and the infimum is taken over all joint distributions of the random variables X and Y with marginals μ and ν respectively. Moreover, when $p \geq 1$, the p^{th} Wasserstein distance preserves all properties of a metric [1], including both the symmetry and the triangle inequality [6]. In this way, Wasserstein distance is suitable to be a similarity measure between the latent representation of nodes, especially for an undirected network.

But the calculation of the general-formed Wasserstein distance is limited by a heavy computational cost, which poses a great challenge to network applications. To reduce the computational cost, in our case since we use Gaussian distributions for the latent representation of nodes, the 2^{th} Wasserstein distance (abbreviated as W_2) has the closed form solution to speed-up the calculation process. The W_2 distance has also been widely used in in computer vision [4, 11], computer graphics [5, 15] or machine learning [12, 14].

More specifically, we have the following formula to calculate W_2 distance between two Gaussian distributions [20]:

$$\begin{aligned} dist &= W_2(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)) \\ dist^2 &= \|\mu_1 - \mu_2\|_2^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}) \end{aligned} \tag{2}$$

In this paper we focus on diagonal covariance matrices[1], thus $\Sigma_1\Sigma_2 = \Sigma_2\Sigma_1$. Then the formula (2) can be simplified as:

$$W_2(\mathcal{N}(\mu_1, \Sigma_1); \mathcal{N}(m_2, \Sigma_2))^2 = \|\mu_1 - \mu_2\|_2^2 + \|\Sigma_1^{1/2} - \Sigma_2^{1/2}\|_F^2. \tag{3}$$

According to the above equation, the time complexity of calculating W_2 distance between the latent representation of two nodes is linear with the embedding dimension L. Therefore, we choose W_2 distance as the similarity measure, and the computational costs no longer constitute limitations.

4.3 Loss Functions

Our overall loss functions for DVNE consists of two parts, the ranking-based loss to preserve the first-order proximity and the reconstruction loss to preserve second-order proximity.

First, we consider how to preserve the first-order proximity. Intuitively, we want all nodes which are linked with v_i to be closer to v_i w.r.t. their embedding, compared to the nodes that have no edge with v_i. More specifically, we propose the following pairwise constraints to preserve the first-order proximity:

$$W_2(\mathbf{h}_i, \mathbf{h}_j) < W_2(\mathbf{h}_i, \mathbf{h}_k), \forall v_i \in \mathbf{V}, \forall v_j \in \mathbf{Nbrs}_i, \forall v_k \notin \mathbf{Nbrs}_i. \tag{4}$$

where $\mathbf{h}_i$ is the latent representation of node v_i, $\mathbf{Nbrs}_i$ is the set of neighbors of node v_i. The smaller the W_2 distance, the larger the similarities between nodes.

Then we use a energy based learning approach [29] to incorporate all of the pairwise constraints defined in the above equation. Mathematically, denoting $E_{ij} = W_2(\mathbf{h}_i, \mathbf{h}_j)$ as the energy between two nodes, we present the objective function as follows:

$$\mathcal{L}_1 = \sum_{(i,j,k)\in\mathbf{D}} ({E_{ij}}^2 + exp(-E_{ik})), \tag{5}$$

where $\mathbf{D}$ is the set of all valid triplets given in Eq. (4). The above objective function penalizes ranking errors by the energy of the pairs, which makes the energy of positive examples to be lower than that of negative examples. Equivalently, it will make the similarity between the positive examples larger than that of negative examples, thus helps preserve the first-order proximity.

For second-order proximity, we use the transition matrix $\mathbf{P}$ as our input features and propose a variant of Wasserstein Auto-Encoders (WAE) [41] as the model to preserve the neighborhood structure. WAE is a deep variational model, which can imply the mathematical relevance of mean vectors and variance terms by the sampling process. The objective of original WAE is composed of two terms, the reconstruction cost and the regularizer. The reconstruction cost aims to capture the information of the input. The regularizer encourages the encoded training distributions to match the prior distribution. As for our problem, the $\mathbf{P}(i,:)$ shows the neighborhood structure of node v_i, thus we use $\mathbf{P}(i,:)$ as the input feature to the WAE for node v_i and reconstruct it to preserve its neighborhood structure. For the regularization term, it is hard to define the prior distribution of each node in the network. Therefore, we focus only on the reconstruction cost to preserve the neighborhood structure.

Let P_X denote the data distribution, and P_G denote the encoded training distribution. The reconstruction cost can be represented as:

$$D_{WAE}(P_X, P_G) = \inf_{Q(Z|X)\in Q} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\big[c(X, G(Z))\big], \tag{6}$$

where Q is the encoders and G is the decoders, $X \sim P_X$ and $Z \sim Q(Z|X)$. It aims to minimize Wasserstein distance between the P_X and P_G.

According to [41], when using $c(x, y) = \|x - y\|_2^2$, the above loss function (6) minimizes the W_2 distance between P_X and P_G, thus P_G captures the information of the input data in the Wasserstein space.

Considering the sparsity of the transition matrix $\mathbf{P}$, we focus on non-zero elements in $\mathbf{P}$ to speed up our model. Thus, we present the loss function as follows to preserve the second-order proximity:

$$\mathcal{L}_2 = \inf_{Q(Z|X)\in Q} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\big[\|X\circ(X - G(Z))\|_2^2\big], \tag{7}$$

where $\circ$ means the element-wise multiplication.

In our model, we use the transition matrix $\mathbf{P}$ as the input feature X. The reconstruction process will make the nodes with similar neighborhoods have similar latent representations. Therefore, the second-order proximity between nodes is preserved.

To preserve first-order proximity and second-order proximity of networks simultaneously, we jointly minimize the loss function by combining Eq. (5) and Eq. (7):

$$\mathcal{L} = \mathcal{L}_1 + \alpha\mathcal{L}_2. \tag{8}$$

4.4 Optimization

For large graphs, optimizing objective function (5) is computationally expensive, which requires to calculate the all valid triplets in $\mathbf{D}$. Therefore, we sample triplets from $\mathbf{D}$ uniformly, which replace $\sum_{(i,j,k)\in\mathbf{D}}$ with $\mathbb{E}_{(i,j,k)\sim\mathbf{D}}$ in Eq. (5). In details, for each iteration, we sample M triplets from $\mathbf{D}$ to calculate the estimates of the gradient.

Considering objective function (7), we need sample Z from $Q(Z|X)$, which is a non-continuous operation and has no gradient. In this case, it is difficult for the deep models to optimize the loss

[1]When the covariance matrices is not diagonal, Wang proposed an fast iterative algorithm (called BADMM) to solve the Wasserstein distance [45]. It is not the focus of the paper and we will not discuss it.

function. To solve the problem, inspired by the Variational Auto-Encoders (VAE) [16], we can use the "reparameterization trick" to optimize the above objective equation. Mathematically, we first sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, then compute $Z = \mu(X) + \Sigma^{1/2}(X) * \epsilon$. Given a fixed X and ϵ, the objective function (7) is deterministic and continuous in the parameters of encoders Q and decoders G. In this way, the whole model can get the gradient when performing the back-propagation, and thus we can use stochastic gradient descent to optimize the model.

4.5 Implementation Details

For all the experiments in this paper we used an encoder and a decoder with a single hidden layer of size $S = 512$ respectively. More specifically, to obtain the embeddings for a node v_i, we have

$$\begin{aligned}
\mathbf{y}_i^{(1)} &= Relu(\mathbf{x}_i\mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \mathbf{W}^{(1)} \in \mathbb{R}^{N\times S}, \mathbf{b}^{(1)} \in \mathbb{R}^S \\
\mu_i &= \mathbf{y}_i^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \mathbf{W}^{(2)} \in \mathbb{R}^{S\times L}, \mathbf{b}^{(2)} \in \mathbb{R}^L \\
\sigma_i &= Elu(\mathbf{y}_i^{(1)}\mathbf{W}^{(3)} + \mathbf{b}^{(3)}) + 1, \mathbf{W}^{(3)} \in \mathbb{R}^{S\times L}, \mathbf{b}^{(3)} \in \mathbb{R}^L \\
\mathbf{z}_i &= \mu_i + \sigma_i * \epsilon, \epsilon \sim \mathcal{N}(0, \mathbf{I}) \\
\mathbf{y}_i^{(2)} &= Relu(\mathbf{z}_i\mathbf{W}^{(4)} + \mathbf{b}^{(4)}), \mathbf{W}^{(4)} \in \mathbb{R}^{L\times S}, \mathbf{b}^{(4)} \in \mathbb{R}^S \\
\widehat{\mathbf{x}}_i &= Sigmoid(\mathbf{y}^{(2)}\mathbf{W}^{(5)} + \mathbf{b}^{(5)}), \mathbf{W}^{(5)} \in \mathbb{R}^{S\times N}, \mathbf{b}^{(5)} \in \mathbb{R}^N,
\end{aligned} \tag{9}$$

where x_i is $\mathbf{P}(i, :)$, Relu [34] and Elu [10] are the rectified linear unit and exponential linear unit. We use $elu() + 1$ to guarantee that σ_i is positive. Because the range of values in x_i is between $[0, 1]$, we use the sigmoid function as the output function of the last hidden layer.

4.6 Complexity analysis

Algorithm 1 lists the procedures of our method. During the training procedure, the time complexity of calculating gradients and updating parameters is $O(T \times M \times (d_{ave}S + SL + L))$, where M is the number of the edges, d_{ave} is the average degree of all nodes, L is the dimension of embedding vectors, S is the size of hidden layer of the encoder and decoder, T is the number of iterations. Since we only reconstruct non-zero elements in x_i, the computational complexity of the first and last hidden layers is $O(d_{ave}S)$. The computational complexity of other hidden layers is $O(SL)$, and it takes $O(L)$ to calculate the W_2 distance between the distributions. In practice we found that a small number of iterations T ($T \leq 50$ for all shown experiments) is needed for convergence.

5 EXPERIMENT

In this section, we empirically evaluate the effectiveness of the our method.

5.1 Experiment Setting

We first introduce the experiment setting before presenting results of the experiments.

5.1.1 Baseline Methods. We use the following five methods as the baselines.

- DVNE_kl : In order to show the advantages of W_2 distance in undirected network. We replace the similarity measure in our method with the KL divergence.

Algorithm 1 Training algorithm of DVNE

Input: The network $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ with the transition matrix $\mathbf{P}$, the parameter α

Output: Network embeddings $\{\mathbf{h}_i\}_{i=1}^N$ and updated parameters $\theta = \{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^5$

1: Initial parameters θ by xavier initialization
2: **while** $\mathcal{L}$ do not converge **do**
3: Sample M triplets from $\mathbf{D}$ uniformly
4: Split these triplets to a number of batches
5: calculate partial derivative $\partial\mathcal{L}/\partial\theta$ with backpropagation algorithm to update θ
6: **end while**

- DeepWalk [37]: This algorithm learns embedding by simulating several uniform random walks. It assumes that a pair of nodes are similar if they are close in the random walks.
- LINE [39]: This algorithm preserves the first-order and second-order proximity between nodes respectively, and directly concatenates the representations for the first-order and second-order proximity.
- SDNE [44]: This method learns a point-vector for each node with preserving the first and the second order proximities simultaneously using deep models.
- Graph2Gauss(G2G_oh) [2]: This method aims to learn the lower-dimensional Gaussian distribution embedding by ranking similarity based on the shortest path between nodes. As the datasets have no attribute information, we compare with the one-hot encoding version of Graph2Gauss as described in the paper.

5.1.2 Dataset. In order to comprehensively evaluate the effectiveness of our proposed method, we use four different real-world datasets, including citation networks and social networks. The detailed information is shown as follows:

- Cora : This is a research paper set constructed by McCallum et al. [33], which consists of 2708 scientific publications classified into one of seven classes.
- Facebook : It is a typical social network dataset without node labels constructed by J. McAuley et al. [30].
- BlogCatalog[38]: This is a network of social relationships of the bloggers listed on the BlogCatalog website. The labels represent the topic categories provided by the authors.
- Flickr [38]: It is a social network where node represents users and edges correspond to friendships between users. The labels represent the interest groups of the users.

All the networks are undirected, and the detailed statistics of the datasets are summarized in Table 1.

5.1.3 Parameter Settings. In all experiments, we set the embedding dimension $L = 128$ unless stated. For the equality, all the methods that learn the embedding as the distribution use the length of mean vector and variance terms to match L. Specifically, our method actually uses half of the dimensionality L as the length of mean vector in all experiments.

For DVNE and DVNE_kl, the hyper-parameters of α are tuned by using grid search on the validation set. We use xavier initialization

Table 1: Statistics of datasets. $|V|$ denotes the number of nodes , $|E|$ denotes the number of edges and $|C|$ denotes the number of classes.

	Cora	Facebook	BlogCatalog	Flickr
$\|V\|$	2,708	4,039	10,312	80,513
$\|E\|$	5,429	88,234	333,983	5,899,882
$\|C\|$	7	-	39	195

[21] for all weight matrices. The parameters are optimized using RMSProp [40] with a fixed learning rate of 0.001.

The parameters for baselines are tuned to be optimal. For DeepWalk, we set window size as 10, walk length as 40, walks per node as 10. For LINE, we set the number of negative samples as 5, and line search for the optimal value of the training samples on different datasets. For SDNE, we use the default parameter settings and the multi-layer deep structure in the author's implementation. For G2G_oh, we use the default parameter settings and the fixed learning rate in the implementation details of the paper.

5.2 Network Reconstruction

The most primal objective for network embedding is to reconstruct the given network, ans a good network embedding method should ensure that the learned embeddings can preserve the original network structure. Thus, we first provide a basic evaluation on different network embedding methods with respect to their capability of network reconstruction. More specifically, we use different network embedding methods to learn the embedding vectors on the different real-world networks. Then we rank pairs of nodes according to their trained similarities between the embedding of nodes, i.e. the W_2 distance for our method, the KL divergence for G2G_oh. The larger the similarities between pairs of nodes, the more likely they have the edges. Then we can use the top ranking pairs to reconstruct the edges of the original networks. For the evaluation metric, we use Area Under Curve (AUC) [18].

Table 2: AUC scores for Network Reconstruction.

	Cora	Facebook	BlogCatalog	Flickr
DVNE	**0.996**	**0.998**	**0.962**	**0.959**
DVNE_kl	0.940	0.958	0.937	0.925
DeepWalk	0.986	0.984	0.864	0.950
Line	0.952	0.934	0.891	0.939
SDNE	0.992	0.960	0.958	0.917
G2G_oh	0.921	0.942	0.924	0.901

The results are shown in Table 2. Our proposed method outperforms the baseline methods in all datasets. The results demonstrate that our proposed method can effectively preserve the original network structure and reconstruct the network. It lays the foundation for other real-world applications of network embedding.

5.3 Link Prediction

Link prediction, aiming to predict which pairs of nodes will form edges in the future, is a typical task of network embedding. In our experiments, we randomly hide 20% of the edges as the testing network and train the embeddings on the rest of the network. After the training, we can obtain the embedding for each node and then use the embeddings to predict the unobserved edges. The pairs of nodes are ranked in a similar way as network reconstruction and the top ranking pairs are evaluated on the testing network. Unlike the reconstruction task, this task predicts the unobserved edges in testing network instead of reconstructing the existing edges in training network. We still use AUC as the evaluation metric.

Table 3: AUC scores for Link Prediction.

	Cora	Facebook	BlogCatalog	Flickr
DVNE	**0.947**	**0.982**	**0.945**	**0.942**
DVNE_kl	0.919	0.930	0.917	0.908
DeepWalk	0.880	0.923	0.827	0.931
Line	0.854	0.882	0.802	0.919
SDNE	0.917	0.931	0.920	0.927
G2G_oh	0.901	0.925	0.903	0.906

From the results in Table 3, our proposed method still outperforms the baselines in all datasets. Especially on the facebook dataset, our method significantly improve AUC scores by 0.05 than the baselines. From the results, we have the following analysis:

Deepwalk can introduce high-order proximity by changing the parameter of window size, but it can not balance the weight of the first-order proximity and the high-order proximity. This means it can not handle well both reconstruction task and prediction task at the same time, which is evident from the experimental results. We also find that LINE does not achieve as good performance as other methods do in most cases. The reason may be twofold. Firstly, LINE adopts shallow structure, which is difficult to capture the highly non-linear structure [44] in the network. Moreover, LINE directly concatenates the embeddings for the first-order and second-order proximity, which is sub-optimal than jointly optimizing them in our method.

Although DVNE and SDNE both exploit the first-order and second-order proximity to preserve the network structure, DVNE achieves better performance. The reason is that our method learns a Gaussian distribution as an embedding for each node, allowing us to capture uncertainty in the network by the latent representations. Actually, adding a new edge between two nodes is a uncertain event, it is more natural to describe this event from the perspective of the distributions.

We also find that DVNE achieves a substantial gain over DVNE_kl on all the datasets. The reason is two fold. Firstly, the KL divergence is not suitable for undirected network because of the asymmetric property of the KL divergence. Secondly, the KL divergence does not necessarily guarantee the transitivity of similarities between the nodes, which makes KL-based methods worse link prediction results.

Compared with DVNE_kl and G2G_oh, which both use the KL divergence as the similarity measures, DVNE_kl outperforms G2G_oh. It is because that G2G_oh use the variance terms as the added dimensions while DVNE_kl relates the variance terms and the mean

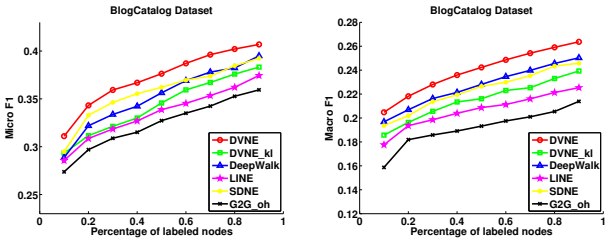


Figure 2: Micro-F1 and Macro-F1 on BlogCatalog.

vectors by the sampling process. Thus, DVNE is able to better capture the uncertainties of nodes and get a better link prediction result.

Overall, the results demonstrate that our proposed method works well for network inference tasks.

5.4 Multi-label Classification

Multi-label classification is another task commonly used to evaluate the effectiveness of the learned embeddings. We evaluate the multi-label classification performance for three datasets (Cora, Blogcatalog and Flickr) that have ground-truth labels. The representations for the nodes are generated from the network embedding methods and are used as features to classify each node into a set of labels. For all methods based on the distribution, we only use the mean vectors as the input features in this task. We adopt a linear SVC [23] as the classifiers for all methods. Then, following [37], we randomly sample a portion of the labeled nodes as the training data and the rest as the test. For BlogCatalog, we randomly sample 10% to 90% of the nodes as the training samples and use the left nodes to test the performance. For Cora and for Flickr, we randomly sample 1% to 10% of the nodes as the training samples and use the left nodes to test the performance on even more sparsely labeled networks. We use the Micro-F_1 and Macro-F_1 scores to evaluate the performance and report results averaged over 10 trials. The results are shown in Figure 2 and Figure 3 respectively.

In Figure 2 and Figure 3, the curve of our method is consistently above the curves of baseline methods. It demonstrates that our method can achieve a better classification performance than baselines even if the labelled data is limited. Such an advantage is meaningful for real-world applications, because the labelled data in real-world network is usually scarce. The variance terms of the representation can help us to deal with the noise information in the network, which makes the mean vectors to better capture the network structure. Therefore, the learned network embedding of our method can better generalize to the classification task than baselines.

In most cases, the performance of G2G_oh is the worst among all the compared network embedding methods. The reasons are twofold. First, G2G_oh uses the variance terms as the added dimensions, causing part of the information of the proximity between nodes included in variance terms. In this way, the performance of G2G_oh greatly degrades. Our method, by using the deep variational model, makes the mean vectors and the variance terms capture different properties of the network, i.e. the mean vector captures the proximity and the variance term captures the uncertainty. In this way, our method can encode more proximity-based information into the mean vectors and thus perform much better than G2G_oh. Second, similar to the previous task, the KL divergence is not a suitable similarity measure to capture the transitivity for the undirected networks.

5.5 Embedding Uncertainty

Learning an embedding as a distribution rather than a point-vector allows us to capture uncertainty of the nodes. With our intuition, the nodes that have less links with other nodes, are harder to get a exact point-vector in the latent space. In other words, the lower the degree of a node, the less discriminative information it contains, thus making its embedding more uncertain. Then we conduct the following experiment to evaluate the intuition. For each node, we select its 10 dimensions with the largest variance and averaged the variance of the 10 dimensions as the variance value for the node. Then for each network dataset, we divide the total nodes into 10 parts based on their degrees. For each part of the nodes, we report the relationship between their degree and their averaged variance values. The Figure 4a shows the result on the all datasets. The horizontal axis represents the $\log_{10}()$ values of degree. Because the max degree of the node is no more than 200 in Cora, the line of Cora is different from the other datasets.

From Figure 4a, we find that the experimental results support our intuition. The nodes with higher degree contains rich information, thus making their variance smaller. Meanwhile, we can see that when the network is denser like Facebook and Flickr, the average variance of embeddings is smaller. This means that our learned embeddings of variance can reflect the density of the network.

Moreover, to demonstrate that the uncertainty in variance terms can help to deal with the noise edges in networks, we conduct an experiment to show the benefits of the uncertainty. First, following the setting in link prediction, we randomly hide 20% of the edges as the testing network and use the rest of network as the training network. Then we randomly choose some pairs of nodes as the noise edges and add them into the training network. We use different network embedding methods to learn the representations of nodes in the modified training network. Similar to link prediction task, we use the similarity between the learned node embeddings to predict the unobserved edges in testing network. We use the results of each method reported in link prediction as the benchmark to calculate the percentage of AUC decline. We vary the percentage of noise edges from 0.05 to 0.5, then show the percentage of AUC decline with respect to it in Figure 4b.

From the results shown in Figure 4b, we can see that the performance of our method is least affected by the noise edges. It demonstrates that our method can better deal with the noise edges in networks by capturing the uncertainties of the nodes. DeepWalk adopts random walk to generate network representations. Each node walks to other communities with a lower probability in the modified training network. Thus, DeepWalk can still preserve the original network structure and the result of DeepWalk is also good. DVNE_kl uses the KL divergence as the similarity measure, which can not well preserve the transitivity in the networks. The noise edges between nodes will further damage this property, leading to worse results. For G2G_oh, there is a weak connection between

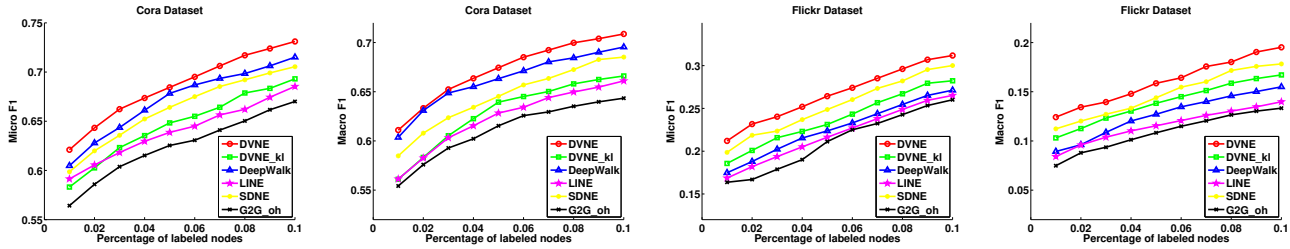
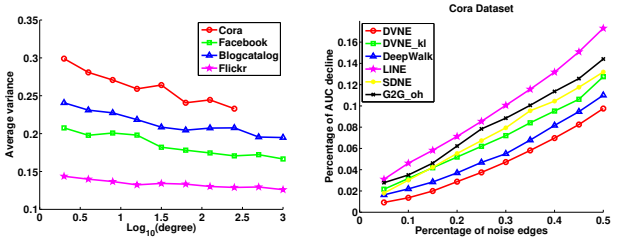


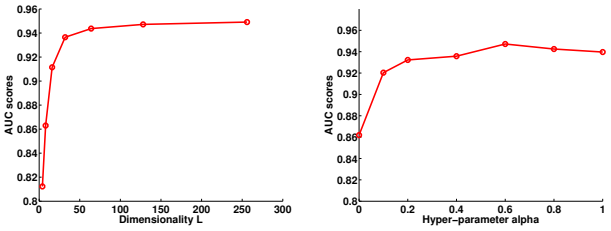
Figure 3: Micro-F1 and Macro-F1 on Cora and Flickr.



(a) The average variance wrt the degree of nodes.

(b) The performance wrt the noise edges.

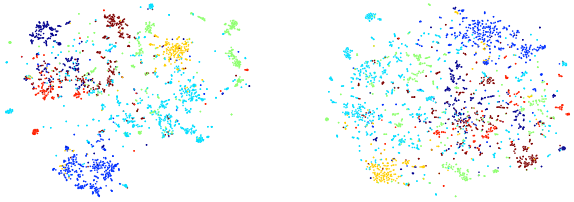
Figure 4: Results of embedding uncertainty.



(a) The AUC scores wrt the dimensionality L

(b) The AUC scores wrt the hyper-parameter α

Figure 6: Results of parameter sensitivity.



(a) DVNE **(b) DVNE_kl**

Figure 5: Visualization of network embedding.

variance terms and mean vectors in the model, which means the variance terms can not well capture the uncertainties of the nodes. Through the sampling process proposed by our method, DVNE is more natural to learn the variance terms that contains the uncertainties of the nodes. Therefore, DVNE and DVNE_kl achieve better performance than G2G_oh. SDNE and LINE treat each edge equally, thus the similarities between nodes in latent space are easily be destroyed by the noise edges.

5.6 Visualization

Visualization is another important application for network embedding. Therefore, we visualize the learned embeddings of the Cora network. Following [39], we first learn a lower-dimensional $L = 128$ embedding for each node and then map those representations in 2-dimension space by t-SNE [32]. For nodes with different labels, we use different colors. Thus, a good visualization result is that the points of the same color are near from each other.

The visualization results are shown in Figure 5, we compare the DVNE with DVNE_kl. For DVNE_kl, in the center part the nodes of different classes are mixed with each other. Obviously, the visualization of DVNE looks better because points of the same color form segmented classes, and the boundaries of each class are clearer. It demonstrate the superiority of our method that using the W_2 distance as the similarity measure in the visualization task.

5.7 Parameter Sensitivity

In this section, we investigate the parameter sensitivity. More specifically, we evaluate how different numbers of the embedding dimensions and different values of hyper-parameter α can affect the results. We report AUC scores on the dataset of Cora.

First, we show how the dimension of the embedding vectors affects the performance in Figure 6a. We can see that initially the performance raises when the number of dimension increases. However, when the number of dimensions continuously increases, the performance tends to be stable. This is because most of the useful information is already encoded into the embeddings. Additional dimensions consume more computing resources, but have less effect on performance. Overall, it is important to determine the appropriate number of dimensions for the latent space. When the number of dimensions is not too small ($L \geq 32$), DVNE is not sensitive to this parameter.

Then, we fix the number of dimensions to 128. The Figure 6b shows how the value of α affects the performance . The parameter of α balances the weight of the first-order proximity and second-order proximity between nodes. When $\alpha = 0$, our method only preserves the first-order proximity between nodes and the performance is worse than that of other parameter settings. It demonstrates that

both first-order and second-order proximity are essential for network embedding methods to capture the network structure. When $\alpha > 0$, we observe that DVNE is also not very sensitive to the choice of this hyper-parameter.

6 CONCLUSIONS

In this paper, we propose a method to learn the Gaussian embedding by the deep variational model, namely DVNE, which can model the uncertainties of nodes. It is the first unsupervised method that represents nodes in networks as Gaussian distributions in Wasserstein space. The method preserves first-order proximity and second-order proximity between nodes to capture the local and global network structure. Moreover, DVNE uses the 2-Wasserstein distance as the similarity measure to better preserve the transitivity in the network with the linear time complexity. The empirical study demonstrates the superiority of our proposed method. Our future direction is to find a good Gaussian prior for each node to better capture the network structure and model the uncertainties of nodes.

7 ACKNOWLEDGEMENTS

This work was supported in part by National Program on Key Basic Research Project (No. 2015CB352300), National Natural Science Foundation of China (No. 61772304, No. 61521002, No. 61531006, No. 61702296), National Natural Science Foundation of China Major Project (No.U1611461), the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. 2008. *Gradient flows: in metric spaces and in the space of probability measures.* Springer Science & Business Media.
[2] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).
[3] A. Bojchevski and S. Günnemann. 2017. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. *ArXiv e-prints* (July 2017). arXiv:stat.ML/1707.03815
[4] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. 2015. Sliced and radon wasserstein barycenters of measures. *Journal of Mathematical Imaging and Vision* 51, 1 (2015), 22–45.
[5] Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. 2011. Displacement interpolation using Lagrangian mass transport. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 158.
[6] Victor Bryant. 1985. *Metric spaces: iteration and application.* Cambridge University Press.
[7] Chen Chen and Hanghang Tong. 2015. Fast eigen-functions tracking on dynamic graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining.* SIAM, 559–567.
[8] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. 2017. Fast, Warped Graph Embedding: Unifying Framework and One-Click Algorithm. *arXiv preprint arXiv:1702.05764* (2017).
[9] Philippe Clement and Wolfgang Desch. 2008. An elementary proof of the triangle inequality for the Wasserstein metric. *Proc. Amer. Math. Soc.* 136, 1 (2008), 333–339.
[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).
[11] Nicolas Courty, Rémi Flamary, and Mélanie Ducoffe. 2017. Learning Wasserstein Embeddings. *arXiv preprint arXiv:1710.07457* (2017).
[12] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. 2017. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence* 39, 9 (2017), 1853–1865.
[13] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *arXiv preprint arXiv:1711.08752* (2017).
[14] Marco Cuturi and Arnaud Doucet. 2014. Fast computation of Wasserstein barycenters. In *International Conference on Machine Learning.* 685–693.
[15] Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 171.
[16] Carl Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* (2016).
[17] Ludovic Dos Santos, Benjamin Piwowarski, and Patrick Gallinari. 2016. Multilabel classification on heterogeneous graphs with gaussian embeddings. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 606–622.
[18] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.
[19] Bent Fuglede and Flemming Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on.* IEEE, 31.
[20] Clark R Givens, Rae Michael Shortt, et al. 1984. A class of Wasserstein metrics for probability distributions. *The Michigan Mathematical Journal* 31, 2 (1984), 231–240.
[21] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.* 249–256.
[22] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 855–864.
[23] Steve R Gunn et al. 1998. Support vector machines for classification and regression. *ISIS technical report* 14, 1 (1998), 5–16.
[24] Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. 2015. Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management.* ACM, 623–632.
[25] Paul W Holland and Samuel Leinhardt. 1972. Holland and Leinhardt reply: some evidence on the transitivity of positive interpersonal sentiment.
[26] Zhipeng Huang and Nikos Mamoulis. 2017. Heterogeneous Information Network Embedding for Meta Path based Proximity. *arXiv preprint arXiv:1701.05291* (2017).
[27] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[28] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.
[29] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. 2006. A tutorial on energy-based learning. *Predicting structured data* 1, 0 (2006).
[30] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems.* 539–547.
[31] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.
[32] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.
[33] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
[34] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10).* 807–814.
[35] Feiping Nie, Wei Zhu, and Xuelong Li. 2017. Unsupervised Large Graph Embedding.. In *AAAI.* 2422–2428.
[36] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proc. of ACM SIGKDD.* 1105–1114.
[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 701–710.
[38] Zafarani Reza and Liu Huan. 2009. Social Computing Data Repository. (2009).
[39] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web.* ACM, 1067–1077.
[40] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.
[41] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. 2017. Wasserstein Auto-Encoders. *arXiv preprint arXiv:1711.01558* (2017).
[42] Ke Tu, Peng Cui, Xiao Wang, Fei Wang, and Wenwu Zhu. 2017. Structural Deep Embedding for Hyper-Networks. *arXiv preprint arXiv:1711.10146* (2017).
[43] Luke Vilnis and Andrew McCallum. 2014. Word representations via gaussian embedding. *arXiv preprint arXiv:1412.6623* (2014).

[44] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.
[45] Huahua Wang and Arindam Banerjee. 2014. Bregman alternating direction method of multipliers. In *Advances in Neural Information Processing Systems*. 2816–2824.
[46] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. (2017).
[47] Chengxi Zang, Peng Cui, Christos Faloutsos, and Wenwu Zhu. 2017. Long Short Memory Process: Modeling Growth Dynamics of Microscopic Social Connectivity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 565–574.

Learning Structural Node Embeddings via Diffusion Wavelets

Claire Donnat, Marinka Zitnik, David Hallac, Jure Leskovec
Stanford University
{cdonnat,marinka,hallac,jure}@stanford.edu

ABSTRACT

Nodes residing in different parts of a graph can have similar structural roles within their local network topology. The identification of such roles provides key insight into the organization of networks and can be used for a variety of machine learning tasks. However, learning structural representations of nodes is a challenging problem, and it has typically involved manually specifying and tailoring topological features for each node. In this paper, we develop GraphWave, a method that represents each node's network neighborhood via a low-dimensional embedding by leveraging heat wavelet diffusion patterns. Instead of training on hand-selected features, GraphWave learns these embeddings in an unsupervised way. We mathematically prove that nodes with similar network neighborhoods will have similar GraphWave embeddings even though these nodes may reside in very different parts of the network. GraphWave runtime scales linearly with the number of edges and experiments in a variety of different settings demonstrate GraphWave's real-world potential for capturing structural roles in networks. All in all, GraphWave outperforms existing state-of-the-art baselines in every experiment, by as much as 137%.

CCS CONCEPTS

• **Networks** → **Topology analysis and generation**; • **Computing methodologies** → **Kernel methods**; **Learning latent representations**; **Spectral methods**; *Cluster analysis*; *Motif discovery*; • **Information systems** → *Clustering*; Nearest-neighbor search;

ACM Reference Format:
Claire Donnat, Marinka Zitnik, David Hallac, Jure Leskovec. 2018. Learning Structural Node Embeddings via Diffusion Wavelets. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3219819.3220025

1 INTRODUCTION

Structural role discovery in graphs focuses on identifying nodes which have topologically similar network neighborhoods while residing in potentially distant areas of the network (Figure 1). Intuitively, nodes with similar structural roles perform similar functions in the network, such as managers in the social network of a company or enzymes in the molecular network of a cell. This alternative definition of node similarity is very different than more traditional notions [9, 12–14, 20, 24, 26, 35], which assume some measure of "smoothness" over the graph and thus consider nodes residing in close network proximity to be similar. Such structural role information about the nodes can be used for a variety of tasks, including as input to machine learning problems, or even to identify key nodes in a system (principal "influencers" in a social network, critical hubs in contagion graphs, etc.).

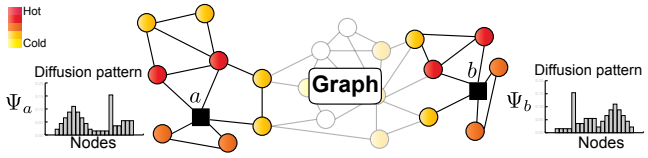


Figure 1: Nodes a and b have similar structural roles even though they are distant in the graph. While the raw spectral graph wavelets of a and b might be very different, we treat them as probability distributions and prove that the distributions of wavelet coefficients of structurally similar nodes are indeed similar.

When structural roles of nodes are defined over a discrete space, they correspond to different topologies of local network neighborhoods (*e.g.*, node on a chain, center of a star, a bridge between two clusters). However, such discrete roles must be pre-defined, requiring domain expertise and manual inspection of the graph structure. A more powerful and robust method for identifying structural similarity involves learning a continuous vector-valued *structural embedding* χ_a of each node a in an unsupervised way. This motivates a natural definition of structural similarity in terms of closeness of topological embeddings: For any $\epsilon > 0$, nodes a and b are defined to be ϵ-structurally similar with respect to a given distance if $dist(\chi_a, \chi_b) \leq \epsilon$. Thus, a robust approach must introduce both an appropriate embedding and an adequate distance metric.

While several methods have been proposed for learning structural embeddings of nodes in graphs, existing approaches are extremely sensitive to small perturbations in the topology and lack mathematical understanding of the properties of the learned embeddings. Furthermore, they often require manually hand-labeling topological features [16], rely on non-scalable heuristics [27], and/or return a single similarity score instead of a multidimensional structural embedding [18, 19].

Present work. Here we address the problem of structure learning on graphs by developing GraphWave. Building upon techniques from graph signal processing [5, 15, 30], our approach learns a multidimensional structural embedding for each node based on the diffusion of a spectral graph wavelet centered at the node. Intuitively, each node propagates a unit of energy over the graph and characterizes its neighboring topology based on the response of the

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220025

network to this probe. We formally prove that the coefficients of this wavelet directly relate to graph topological properties. Hence, these coefficients contain all the necessary information to recover structurally similar nodes, without requiring the explicit hand-labeling of features. However, the wavelets are, by design, localized on the graph. Therefore to compare wavelets for nodes that are far away from each other, typical graph signal processing methods (using metrics like correlation between wavelets or ℓ_2 distance) cannot be used without specifying an exact one-to-one mapping between nodes for every pairwise comparison, a computationally intractable task. For this reason, these wavelets have never before been used for learning structural embeddings.

To overcome this challenge, we propose a novel way of treating the wavelets as probability distributions over the graph. This way, the structural information is contained in *how* the diffusion spreads over the network rather than *where* it spreads. In order to provide vector-valued embeddings, we embed these wavelet distributions using the empirical characteristic function [23]. The advantage of empirical characteristic functions is that they capture all the moments (including higher-order moments) of a given distribution. This allows GraphWave to be robust to small perturbations in the local edge structure, as we prove mathematically. The computational complexity of GraphWave is linear in the number of edges, thus allowing it to scale to large (sparse) networks. Finally, we compare GraphWave to several state-of-the-art baselines on both real and synthetic datasets, obtaining improvements of up to 137% and demonstrating how our approach is a useful tool for structural embeddings in graphs.

Summary of contributions. The main contributions of our paper are as follows:

- We propose a novel use of spectral graph wavelets by treating them as probability distributions and characterizing the distributions using empirical characteristic functions.
- We leverage these insights to develop a scalable method (GraphWave) for learning node embeddings based on structural similarity in graphs, outperforming existing state-of-the-art baselines.
- We prove mathematically that GraphWave accurately recovers structurally similar and structurally equivalent nodes.

Further related work. Prior work on discovering nodes with similar structural roles has typically relied on explicit featurization of nodes. These methods generate an exhaustive listing of each node's local topological properties (*e.g.*, node degree, number of triangles it participates in, number of k-cliques, its PageRank score) before computing node similarities based on such heuristic representations. A notable example of such approaches is *RolX* [11, 16], a matrix-factorization based method which aims to recover a soft-clustering of nodes into a predetermined number of K distinct roles using recursive feature extraction [17]. Similarly, *struc2vec* [27] uses a heuristic to construct a multilayered graph based on topological metrics and simulates random walks on the graph to capture structural information. In contrast, our approach does not rely on heuristics (we mathematically prove its efficacy) and does not require explicit manual feature engineering or hand-tuning of parameters.

Recent neural representation learning methods (structure2vec [6], neural fingerprints [8], graph convolutional networks (GCNs) [13, 20], message passing networks [10], etc.) are a related line of research. However, these graph embedding methods do not apply in our setting, since they solve a (supervised) graph classification task and/or embed entire graphs while we embed individual nodes.

Another line of related work are graph diffusion kernels [5] which have been utilized for various graph modeling purposes [3, 22, 29, 34]. However, to the best of our knowledge, our paper is the first to apply graph diffusion kernels for determining structural roles in graphs. Kernels have been shown to efficiently capture geometrical properties and have been successfully used for shape detection in the image processing community [1, 25, 33]. However, in contrast to shape-matching problems, GraphWave considers these kernels as probability distributions over real-world graphs. This is because the graphs that we consider are highly irregular (as opposed to the Euclidean and manifold graphs). Therefore, traditional wavelet methods, which typically analyze node diffusions across specific nodes that occur in regular and predictable patterns, do not apply. Instead, GraphWave characterizes the *shape* of the diffusion, rather than the specific nodes where the diffusion occurs. This key insight allows us to uncover structural embeddings and to discover structurally similar nodes.

2 LEARNING STRUCTURAL EMBEDDINGS

Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $\mathcal{V} = \{a_1, \ldots, a_N\}$, edges $\mathcal{E}$, an adjacency matrix A (binary or weighted), and a degree matrix $D_{ii} = \sum_j A_{ij}$, we consider the problem of learning, for every node a_i, a *structural embedding* representing a_i's position in a continuous multidimensional space of structural roles.

We frame this as an unsupervised learning problem based on spectral graph wavelets [15] and develop an approach called GraphWave that provides mathematical guarantees on the optimality of learned structural embeddings.

2.1 Spectral graph wavelets

In this section, we provide background on the spectral graph wavelet-based model [15, 30] that we will use in the rest of the paper.

Let U be the eigenvector decomposition of the unnormalized graph Laplacian $L = D - A = U\Lambda U^T$ and let $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_N$ ($\Lambda = \text{Diag}(\lambda_1, \ldots, \lambda_N)$) denote the eigenvalues of L.

Let g_s be a filter kernel with scaling parameter s. In this paper, we use the heat kernel $g_s(\lambda) = e^{-\lambda s}$, but our results apply to any scaling wavelet [31]. For now, we assume that s is given; we develop a method for selecting an appropriate value of s in Section 4.

Graph signal processing [15, 30] defines the spectral graph wavelet associated with g_s as the signal resulting from the modulation in the spectral domain of a Dirac signal centered around node a. The spectral graph wavelet Ψ_a is given by an N-dimensional vector:

$$\Psi_a = U \,\text{Diag}(g_s(\lambda_1), \ldots, g_s(\lambda_N))U^T \delta_a, \tag{1}$$

where $\delta_a = \mathbb{1}(a)$ is the one-hot vector for node a. For notational simplicity, we drop the explicit dependency of spectral graph wavelet Ψ_a on s. The m-th wavelet coefficient of this column vector is thus given by $\Psi_{ma} = \sum_{l=1}^{N} g_s(\lambda_l) U_{ml} U_{al}$.

In spectral graph wavelets, the kernel g_s modulates the eigenspectrum such that the resulting signal is typically *localized on the graph and in the spectral domain* [30]. Spectral graph wavelets are

based on an analogy between temporal frequencies of a signal and the Laplacian's eigenvalues. Eigenvectors associated with smaller eigenvalues carry slow varying signal, encouraging nodes that are neighbors to share similar values. In contrast, eigenvectors associated with larger eigenvalues carry faster-varying signal across edges. The low-pass filter kernel g_s can thus be seen as a modulation operator that discounts higher eigenvalues and enforces smoothness in the signal variation on the graph.

2.2 GraphWave algorithm

We first describe GraphWave (Algorithm 1); then, we analyze it in the next section. For every node a, GraphWave returns a $2d$-dimensional vector χ_a representing its *structural embedding*, where nodes with structurally similar local network neighborhoods will have similar embeddings.

We first apply spectral graph wavelets to obtain a diffusion pattern for every node (Line 3), which we gather in a matrix Ψ. Here, Ψ is a $N \times N$ matrix, where a-th column vector is the spectral graph wavelet for a heat kernel centered at node a. In contrast to prior work that studies wavelet coefficients as a function of the scaling parameter s, we study them as a function of the network (*i.e.*, how the coefficients vary across the local network neighborhood around the node a). In particular, coefficients in each wavelet are identified with the nodes and Ψ_{ma} represents the amount of energy that node a has received from node m. As we will later show nodes a and b with similar network neighborhoods have similar spectral wavelet coefficients Ψ (assuming that we know how to solve the "isomorphism" problem and find the explicit one-to-one mapping of the nodes from a's neighborhood to the nodes of the b's neighborhood). To resolve the node mapping problem GraphWave treats the wavelet coefficients as a probability distribution and characterizes the distribution via empirical characteristic functions. This is a key insight that makes it possible for GraphWave to learn nodes' structural embeddings via spectral graph wavelets.

More precisely, we embed spectral graph wavelet coefficient distributions into $2d$-dimensional space (Line 4-7) by calculating the characteristic function for each node's coefficients Ψ_a and sample it at d evenly spaced points. The characteristic function of a probability distribution X is defined as: $\phi_X(t) = \mathbb{E}[e^{itX}]$, $t \in \mathbb{R}$. The function $\phi_X(t)$ fully characterizes the distribution of X because it captures information about all the moments of probability distribution X [23]. For a given node a and scale s, the empirical characteristic function of Ψ_a is defined as:

$$\phi_a(t) = \frac{1}{N} \sum_{m=1}^{N} e^{it\Psi_{ma}} \tag{2}$$

Finally, structural embedding χ_a of node a is obtained by sampling the 2-dimensional parametric function (Eq. (2)) at d evenly spaced points $t_1, \ldots t_d$ and concatenating the values:

$$\chi_a = \Big[\mathrm{Re}(\phi_a(t_i)), \mathrm{Im}(\phi_a(t_i))\Big]_{t_1, \cdots t_d} \tag{3}$$

Note that we sample the empirical characteristic function $\phi_a(t)$ at d points, which creates a structural embedding of size $2d$, so the dimensionality of the embedding is independent of the graph size.

Distance between structural embeddings. The output of GraphWave is a structural embedding χ_a for each node a in the graph.

Algorithm 1 Learning structural embeddings in GraphWave.

1: **Input:** Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, scale s, evenly spaced sampling points $\{t_1, t_2, \ldots, t_d\}$
2: **Output:** Structural embedding $\chi_a \in \mathbb{R}^{2d}$ for every node $a \in \mathcal{V}$
3: Compute $\Psi = U g_s(\Lambda) U^T$ (Eq. (1))
4: **for** $t \in \{t_1, t_2, \ldots, t_d\}$ **do**
5: Compute $\phi(t) =$ column-wise mean$(e^{it\Psi}) \in \mathbb{R}^N$
6: **for** $a \in \mathcal{V}$ **do**
7: Append $\mathrm{Re}(\phi_a(t))$ and $\mathrm{Im}(\phi_a(t))$ to χ_a

We can explore distances between these embeddings through the use of the ℓ_2 distance on χ_a. The structural distance between nodes a and b is then defined as: $dist(a, b) = \|\chi_a - \chi_b\|_2$. By definition of the characteristic function, this amounts to comparing moments of different orders defined on wavelet coefficient distributions.

Scaling parameter. The scaling parameter s determines the radius of network neighborhood around each node a ([15, 34]). A small value of s determines node embeddings based on similarity of nodes' immediate neighborhoods. In contrast, a larger value of s allows the diffusion process to spread farther in the network, resulting in embeddings based on neighborhoods with greater radii.

GraphWave can also integrate information across different radii of neighborhoods by jointly considering many different values of s. This is achieved by concatenating J representations $\chi_a^{(s_j)}$, each associated with a scale s_j, where $s_j \in [s_{\min}, s_{\max}]$. We provide a theoretically justified method for finding an appropriate range $s_{\min}$ and $s_{\max}$ in Section 4. In this multiscale version of GraphWave, the final aggregated structural embedding for node a is a vector $\chi_a \in \mathbb{R}^{2dJ}$ with the following form: $\chi_a = [\mathrm{Re}(\phi_a^{(s_j)}(t_i)), \mathrm{Im}(\phi_a^{(s_j)}(t_i))]_{t_i, s_j}$.

Computational complexity. We use Chebyshev polynomials [32] to compute Line 3 in Algorithm 1. As in [7], each power of the Laplacian has a computational cost of $O(|\mathcal{E}|)$, yielding an overall complexity of $O(K|\mathcal{E}|)$, where K denotes the order Chebyshev polynomial approximation. The overall complexity of GraphWave is linear in the number of edges, which allows GraphWave to scale to large sparse networks.

3 ANALYSIS OF GRAPHWAVE

In this section, we provide theoretical motivation for our spectral graph wavelet-based model. First we analytically show that spectral graph wavelet coefficients characterize the topological structure of local network neighborhoods (Section 3.1). Then we show that structurally equivalent/similar nodes have near-identical/similar embeddings (Sections 3.2 and 3.3), thereby providing a mathematical guarantee on the optimality of GraphWave.

3.1 Network structure via diffusion wavelets

We start by establishing the relationship between the spectral graph wavelet of a given node a and the topological properties of local network neighborhood centered at a. In particular, we prove that a wavelet coefficient Ψ_{ma} provides a measure of network connectivity between nodes a and m.

We use the fact that the spectrum of the graph Laplacian is discrete and contained in the compact set $[0, \lambda_N]$. It then follows

from the Stone-Weierstrass theorem that the restriction of kernel g_s to the interval $[0, \lambda_N]$ can be approximated by a polynomial. This polynomial approximation, denoted as P, is tight and its error can be uniformly bounded. Formally, this means:

$$\forall \epsilon > 0, \quad \exists P : P(\lambda) = \sum_{k=0}^{K} \alpha_k \lambda^k \quad \text{such that} \quad |g_s(\lambda) - P(\lambda)| \leq \epsilon \quad \forall \lambda \in [0, \lambda_{\max}], \tag{4}$$

where K is the order of polynomial approximation, α_k are coefficients of the polynomial, and $r(\lambda) = g_s(\lambda) - P(\lambda)$ is the residual. We can now express the spectral graph wavelet for node a in terms of the polynomial approximation as:

$$\Psi_a = (\sum_{k=0}^{K} \alpha_k L^k)\delta_a + Ur(\Lambda)U^T\delta_a. \tag{5}$$

We note that Ψ_a is a function of $L^k = (D - A)^k$ and thus can be interpreted using graph theory. In particular, it contains terms of the form D^k (capturing the degree), A^k (capturing the number of k-length paths that node a participates in), and terms containing both A and D, which denote paths of length up to k going from node a to every other node m.

Using the Cauchy-Schwartz's inequality and the facts that U is unitary and $r(\lambda)$ is uniformly bounded (Eq. (4)), we can bound the second term on the right-hand side of Eq. (5) by:

$$|\delta_m^T Ur(\Lambda)U^T\delta_a|^2 \leq \Big(\sum_{j=1}^{N} |r(\lambda_j)|^2 U_{aj}^2\Big)\Big(\sum_{j=1}^{N} U_{mj}^2\Big) \leq \epsilon^2. \tag{6}$$

As a consequence, each wavelet Ψ_a can be approximated by a K-th order polynomial that captures information about the K-hop neighborhood of node a. The analysis of Eq. (5), where we show that the second term is limited by ϵ, indicates that spectral graph wavelets are predominately governed by topological features (specifically, degrees, cycles and paths) according to the specified heat kernel. The wavelets thus contain the information necessary to generate structural embeddings of nodes.

3.2 Embeddings of structurally equivalent nodes

Let us consider nodes a and b whose K-hop neighborhoods are identical (where K is an integer less than the diameter of the graph), meaning that nodes a and b are structurally equivalent. We now show that a and b have ϵ-structurally similar embeddings in GraphWave.

First, we use the Taylor expansion to obtain an explicit K-th order polynomial approximation of g_s as $P(\lambda, s) = \sum_{k=0}^{K}(-1)^k(s\lambda)^k/k!$. Then, for each eigenvalue λ, we use the Taylor-Lagrange equality to ensure the existence of $c_\lambda \in [0, s]$ such that:

$$|r(\lambda)| = |e^{-\lambda s} - P(\lambda, s)| = \frac{(\lambda s)^{K+1}}{(K+1)!}e^{-\lambda c_\lambda} \leq \frac{(\lambda s)^{K+1}}{(K+1)!}. \tag{7}$$

If we take any s that satisfies: $s \leq ((K+1)!\epsilon))^{1/(K+1)}/\lambda_2$, then the absolute residual $|r(\lambda)|$ in Eq. (7) can be bounded by ϵ for each eigenvalue λ. Here, ϵ is a parameter that we can specify depending on how close we want the embeddings of structurally equivalent nodes to be (note that smaller values of the scale s lead to smaller values of ϵ and thus tighter bounds).

Because a and b are structurally equivalent, there exists a one-to-one mapping π from the K-hop neighborhood of a (*i.e.*, $\mathcal{N}_K(a)$) to the K-hop neighborhood of b (*i.e.*, $\mathcal{N}_K(b)$), such that: $\mathcal{N}_K(b) = \pi(\mathcal{N}_K(a))$. We extend the mapping π to the whole graph $\mathcal{G}$ by randomly mapping the remaining nodes. Using Eq. (5), we write the difference between each pair of mapped coefficients Ψ_{ma} and $\Psi_{\pi(m)b}$ in terms of the K-th order approximation of the graph Laplacian:

$$\begin{aligned}|\Psi_{ma} - \Psi_{\pi(m)b}| &= \Big|\delta_m U(P(\Lambda) + r(\Lambda)\Big)U^T\delta_a - \\ &\delta_{\pi(m)}U(P(\Lambda)+r(\Lambda)\Big)U^T\delta_b\Big| \leq \Big|(UP(\Lambda)U^T)_{ma} - (UP(\Lambda)U^T)_{\pi(m)a}\Big| \\ &+ \Big|(Ur(\Lambda)U^T)_{ma}\Big| + \Big|(Ur(\Lambda)U^T)_{\pi(m)b}\Big|. \end{aligned} \tag{8}$$

Here, we analyze the first term on the second line in Eq. (8). Since the K-hop neighborhoods around a and b are identical and by the localization properties of the k-th power of the Laplacian (k-length paths, Section 3.1), the following holds:

$$\forall m \in \mathcal{N}_K(a), (\sum_{k=0}^{K} \alpha_k L^k)_{ma} = (\sum_{k=0}^{K} \alpha_k L^k)_{\pi(m)b},$$
$$\forall m \notin \mathcal{N}_K(a), (\sum_{k=0}^{K} \alpha_k L^k)_{ma} = (\sum_{k=0}^{K} \alpha_k L^k)_{\pi(m)a} = 0,$$

meaning that this term cancels out in Eq. (8). To analyze the second and third terms on the second line of Eq. (8), we use the bound for the residual term in the spectral graph wavelet (Eq. (6)) to uniformly bound entries in matrix $Ur(\Lambda)U^T$ by ϵ.

Therefore, each wavelet coefficient in Ψ_a is within 2ϵ of its corresponding wavelet coefficient in Ψ_b, *i.e.*, $|\Psi_{ma} - \Psi_{\pi(m)b}| \leq 2\epsilon$.

As a result, because similarity in distributions translates to similarity in the resulting characteristic functions (Lévy's continuity theorem), then assuming the appropriate selection of scale, structurally equivalent nodes have ϵ-structurally similar embeddings.

3.3 Embeddings of structurally similar nodes

We now analyze structurally similar nodes, or nodes whose K-hop neighborhoods are identical up to a small perturbation of the edges. We show that such nodes have similar GraphWave embeddings.

Let $\widetilde{\mathcal{N}}_K(a)$ denote a perturbed K-hop neighborhood of node a obtained by rewiring edges in the original K-hop neighborhood $\mathcal{N}_K(a)$. We denote by $\tilde{L}$ the graph Laplacian associated with that perturbation. We next show that when perturbation of a node neighborhood is small, the changes in node's wavelet coefficients are small as well.

Formally, assuming a small perturbation of the graph structure (*i.e.*, $\sup ||L^k - \tilde{L}^k||_F \leq \epsilon$, for all $k \leq K$), we use K-th order Taylor expansion of kernel g_s to express the wavelet coefficients in the perturbed graph as:

$$\widetilde{\Psi}_a = \sum_{k=0}^{K} \alpha_k \tilde{L}^k + \tilde{U}r(\tilde{\Lambda})\tilde{U}^T. \tag{9}$$

We then use the Weyl's theorem [4] to relate perturbations in the graph structure to the change in the eigenvalues of the graph Laplacian. In particular, a small perturbation of the graph yields small perturbations of the eigenvalues. That is, for each $\tilde{\lambda}$, $r(\tilde{\lambda})$ is

close its original value $r(\lambda)$: $r(\tilde{\lambda}) = r(\lambda) + o(\epsilon) \leq C\epsilon$, where C is a constant. Taking everything together, we get:

$$|\Psi_{ma} - \widetilde{\Psi}_{ma}| \leq |\sum_{k=0}^{K} \alpha_k (L^k - \tilde{L}^k)_{ma}| + |\tilde{U} r(\tilde{\Lambda}) \tilde{U}^T|_{ma} + |U r(\Lambda) U^T|_{ma} = (\sum_{k=0}^{K} |\alpha_k| + 1 + C)\epsilon,$$

indicating that structurally similar nodes have similar embeddings in GraphWave.

4 SCALE OF HEAT DIFFUSION WAVELETS

Here, we develop a method that automatically finds an appropriate range of values for the scaling parameter s in heat kernel g_s, which we use in the multiscale version of GraphWave (Section 2.2).

We do so by specifying an interval bounded by $s_{\min}$ and $s_{\max}$ through the analysis of variance in heat diffusion wavelets. Intuitively, small values of s allow little time for the heat to propagate, yielding diffusion distributions (*i.e.*, heat diffusion wavelet distributions) that are trivial in the sense that only a few coefficients have non-zero values and are thus unfit for comparison. For larger values of s, the network converges to a state in which all nodes have an identical temperature equal to $1/N$, meaning that diffusion distributions are data-independent, hence non-informative.

Here, we prove two propositions to provide new insights into the variance and convergence rate of heat diffusion wavelets. We then use these results to select $s_{\min}$ and $s_{\max}$.

Proposition 1. *The variance of off-diagonal coefficients in heat diffusion wavelet $\Psi_a^{(s)}$ is proportional to:*

$$Var[\{\Psi_{am}^{(s)}; m \neq a\}] \propto \Delta_a^{(0)} \Delta_a^{(2s)} - (\Delta_a^{(s)})^2,$$

where $\Delta_a^{(s)} = |\Psi_{aa}^{(s)} - \frac{1}{N}|$ decreases monotonically with s.

Proof. Let us denote the mean of off-diagonal coefficients in wavelet Ψ_a by $\tilde{\mu}_a^{(s)} = \sum_{m \neq a} \Psi_{ma}^{(s)}/(N-1)$. We use the fact that $\sum_{m \neq a} \Psi_{ma}^{(s)} = 1 - \Psi_{aa}^{(s)}$, along with the definition of the variance, to obtain:

$$\begin{aligned}
\mathrm{Var}[\{\Psi_{am}^{(s)}; m \neq a\}] &= \frac{1}{N-1} \sum_{m \neq a} (\Psi_{ma}^{(s)} - \tilde{\mu}_a^{(s)})^2 \\
&= \frac{1}{N-1} \sum_{m \neq a} (\Psi_{ma}^{(s)})^2 - (\tilde{\mu}_a^{(s)})^2 \\
&= \frac{N}{(N-1)^2} (\Psi_{aa}^{(2s)} \frac{N-1}{N} - (\Psi_{aa}^{(s)})^2 + \frac{2\Psi_{aa}^{(s)}}{N} - \frac{1}{N}) \\
&= \frac{N}{(N-1)^2} (\Delta_a^{(0)} \Delta_a^{(2s)} - (\Delta_a^{(s)})^2).
\end{aligned}$$

□

Proposition 1 proves that the variance is a function of $\Delta_a^{(s)}$. Therefore, to maximize the variance, we must analyze the behavior of $\Delta_a^{(s)}$. To ensure sufficient variability in the distribution of wavelet coefficients, we need to select a range $[s_{min}, s_{\max}]$ that bounds $\Delta_a^{(s)}$. Our goal thus becomes establishing that $\Delta_a^{(s)}$ is large enough that the diffusion has had time to spread, while remaining sufficiently small to ensure that the diffusion is far from its converged state.

Proposition 2. *The convergence of heat diffusion wavelet coefficient $\Psi_{am}^{(s)}$ is bounded by:*

$$e^{-\lambda_N \lceil s \rceil} \Delta_a^{(0)} \leq \Delta_a^{(s)} \leq e^{-\lambda_2 \lfloor s \rfloor} \Delta_a^{(0)}.$$

Proof. For non-negative s, $|\Psi_{aa}^{(s+1)} - \frac{1}{N}| = |\sum_{j=2}^{N} e^{-\lambda_j (s+1)} U_{ja}^2| \leq e^{-\lambda_2} |\Psi_{aa}^{(s)} - \frac{1}{N}|$, and, symetrically, $|\Psi_{aa}^{(s+1)} - \frac{1}{N}| \geq e^{-\lambda_N} |\Psi_{aa}^{(s)} - \frac{1}{N}|$. We conclude that $e^{-\lambda_N} \leq |\Psi_{aa}^{(s+1)} - \frac{1}{N}| / |\Psi_{aa}^{(s)} - \frac{1}{N}| \leq e^{-\lambda_2}$. Given any $s \in \mathbb{N}$, we use the induction principle to get $e^{-\lambda_N s} |\Psi_{aa}^{(0)} - \frac{1}{N}| \leq |\Psi_{aa}^{(s)} - \frac{1}{N}| \leq e^{-\lambda_2 s} |\Psi_{aa}^{(0)} - \frac{1}{N}|$, which yields the desired bound, $e^{-\lambda_N s} \Delta_a^{(0)} \leq \Delta_a^{(s)} \leq e^{-\lambda_2 s} \Delta_a^{(0)}$. Since $\Delta_a^{(s)}$ is a smooth increasing function of s, we take the floor/ceiling of any non-integer $s \geq 0$ and this proposition must hold. □

Selection of $s_{\max}$. We select $s_{\max}$ such that wavelet coefficients are localized in the network. To do so, we use Proposition 2 and bound $\Delta_a^{(s)}$ by the graph Laplacian's eigenvalues. When the bulk of the eigenvalues leans towards λ_N, $\Delta_a^{(s)}$ is closer to $e^{-\lambda_N}$ (*i.e.*, lower bound in Proposition 2). When the bulk of the eigenvalues is closer to λ_2, $\Delta_a^{(s)}$ will lean towards $e^{-\lambda_2}$ (*i.e.*, upper bound in Proposition 2). In each case, the diffusion is localized if $\Delta_a^{(s)}$ is above a given threshold $\eta < 1$. Indeed, this ensures that $\Delta_a^{(s)}$ has shrunk to at most $\eta * 100$ % of its initial value at $s = 0$, and yields a bound of the form

$\Delta_a^{(s)}/\Delta_a^{(0)} \geq \eta$. The bound implies that: $e^{-\lambda s} \geq \eta$, or $s \leq -\log(\eta)/\lambda$. To find a middle ground between the two convergence scenarios, we take λ to be the geometric mean of λ_2 and λ_N. As opposed to the arithmetic mean, the geometric mean maintains an equal weighting across the range $[\lambda_2, \lambda_N]$, and a change of ϵ% in λ_2 has the same effect as a change in ϵ% of λ_N. We thus select $s_{\max}$ as $-\log(\eta)/\sqrt{\lambda_2 \lambda_N}$.

Selection of $s_{\min}$. We select $s_{\min}$ to ensure the adequate diffusion resolution. In particular, we select a minimum value $s_{\min}$ such that each wavelet has sufficient time to spread. That is, $\Delta_a^{(s)}/\Delta_a^{(0)} \leq \gamma$. As in the case of $s_{\max}$ above, we obtain a bound of $s \geq -\log(\gamma)/\lambda$. Hence, we set $s_{\min}$ to $-\log(\gamma)/\sqrt{\lambda_2 \lambda_N}$.

To cover an appropriate range of scales, we suggest setting $\eta = 0.85$ and $\gamma = 0.95$.

5 EXPERIMENTS ON SYNTHETIC GRAPHS

GraphWave's embeddings are independent of any downstream task, so we evaluate them in a variety of different synthetic settings to demonstrate their potential for capturing structural roles in networks.

Baseline methods. We evaluate the performance of GraphWave[1] against two state-of-the-art baselines for learning structural embeddings: *struc2vec* [27], a method which discovers structural embeddings at different scales through a sequence of walks on a multilayered graph, and *RolX* [16], a method based on non-negative matrix factorization of a node-feature matrix (number of neighbors, triangles, etc.) that describes each node based on this given set of latent features. While in [16], the authors develop a method for

[1] All code can be downloaded at: http://snap.stanford.edu/graphwave.

automatically selecting the number of roles in *RolX*, we use *RolX* as an oracle estimator, providing it with the correct number of classes. We note that GraphWave and *struc2vec* learn embeddings on a continuous spectrum instead of into discrete classes (and thus they do not require this parameter).

We also compare GraphWave with two recent unsupervised node representation learning methods, *node2vec* [12] and *DeepWalk* [26], to emphasize the difference between such methods and our structural similarity-based approach. For all baselines, we use the default parameter values in the available solvers, and for GraphWave, we use the multiscale version (Section 4), set $d = 50$ and use evenly spaced sampling points t_i in range $[0, 100]$. We again note that graph embedding methods (structure2vec [6], neural fingerprints [8], GCNs [13, 20], etc.) do not apply in these settings, since they embed entire graphs while we embed individual nodes.

5.1 Barbell graph

We first consider a barbell graph consisting of two dense cliques connected by a long chain (Figure 2A). We plot a 2D PCA representation the learned embeddings of the three structural-based methods, GraphWave, *RolX*, and *struc2vec*, in Figures 2B-D.

GraphWave correctly learns identical representations for structurally equivalent nodes, providing empirical evidence for our theoretical result in Section 3.2. This can be seen by structurally equivalent nodes in Figure 2A (nodes of the same color) having identical projections and overlap in the PCA plot (Figure 2D). In contrast, both *RolX* and *struc2vec* fail to recover the exact structural equivalences (as shown by the non-overlapping nodes of the same color). For *struc2vec*, the projections are not consistent with the expected ordering: the yellow nodes are farther to the green ones than the red. In contrast, *RolX* yields highly-similar node embeddings whose projections cluster into three high-level groups, meaning that *RolX* can identify three out of eight structural classes in the barbell graph.

We also note that all three methods correctly group the clique nodes (purple) together. However, only GraphWave correctly differentiates between nodes connecting the two dense cliques in the barbell graph, providing empirical evidence for our theoretical result in Section 3.3. GraphWave represents those nodes in a gradient-like pattern that captures the spectrum of structural roles of those nodes (Figure 2D).

5.2 Graphs with planted structural equivalences

We next systematically evaluate the methods on synthetic graphs. We develop a procedure that can generate a graph with planted structural equivalences and also ground-truth node labels indicating the structural role of each node. Our goal is to use these ground-truth structural roles to evaluate our method's performance.

Generating the graphs. The graphs are given by basic shapes of one of different types ("house", "fan", "star") that are regularly placed along a cycle (Table 1 and Figure 3A) of length 30. In the "varied" setup, we mix the three basic shapes by placing 8 instances of each type randomly along a cycle (of length 40), thus generating synthetic graphs with richer and more complex structural role patterns. Additional "noisy" graphs are generated by adding edges uniformly at random on these graphs. In our experiments, we set this number to be 10% of the edges in the original structure. This setup is designed to assess the robustness of the methods to data perturbations ("house perturbed", "varied perturbed").

Experimental setup. For each trial, we generate one instance of each of these four types of structure, on each of which we run the different methods to learn each embedding. Each experiment was repeated 25 times, and the performance of each algorithm was assessed by considering two settings:

- **Unsupervised setting:** We assess the ability of each method to embed close together nodes with the same ground-truth structural role. We use agglomerative clustering (with single linkage) to cluster embeddings learned by each method and evaluate the clustering quality via: (1) *homogeneity* [28], conditional entropy of ground-truth structural roles given the predicted clustering; (2) *completeness* [28], a measure of how many nodes with the same ground-truth structural role are assigned to the same cluster, and (3) *silhouette score*, a measure of intra-cluster distance vs. inter-cluster distance.
- **Supervised setting:** We assess the performance of learned embeddings for node classification. Using 10-fold cross validation, we predict the structural role (label) of each node in the test set based on its 4-nearest neighbors in the training set as determined by the embedding space (Section 2.2) The reported score is then the average accuracy and F_1-score over 25 trials.

Results. For both the unsupervised and supervised settings, GraphWave outperforms the other four methods (Table 1). *Node2vec* and *DeepWalk* perform very poorly in all these experiments, yielding significantly lower scores than the structural equivalence-based methods. This is due to the fact that the crux of these methods is to learn node embeddings that reflect their distances in the graph. Of the remaining methods, GraphWave consistently outperforms *struc2vec*, obtaining a higher score in every metric under every setting. On average, compared to *struc2vec*, GraphWave has a 63% higher homogeneity score, 61% higher completeness score, 137% higher silhouette score, 46% higher prediction accuracy, and 51% higher F_1 score. Both GraphWave and *RolX* achieved perfect performance in the noise-free "house" setting. However, while *RolX* exhibits slightly higher homogeneity and completeness in two of the experiments ("house perturbed" and "varied"), GraphWave has a higher silhouette score, prediction accuracy, and F_1 score than *RolX* in all four experiments. Furthermore, in the most challenging setting ("varied perturbed"), GraphWave outperforms *RolX* in every metric, including by 9% in homogeneity, 8% in completeness, and 23% in silhouette score. This provides empirical evidence for our analytical result that GraphWave is robust to noise in the edge structure. The silhouette scores also show that the clusters recovered by GraphWave tend to be denser and better separated than for the other methods.

Visualizing the embeddings. We also highlight the visualization power of the embeddings that we recover: their associated parametric curves provide a way of visualizing differences between nodes. As an example, using cycle graph with attached "house" (Figure 3A), we plot a 2D PCA projections of GraphWave's embeddings in Figure 3B, confirming that GraphWave accurately distinguishes between nodes with distinct structural roles. We also visualize the

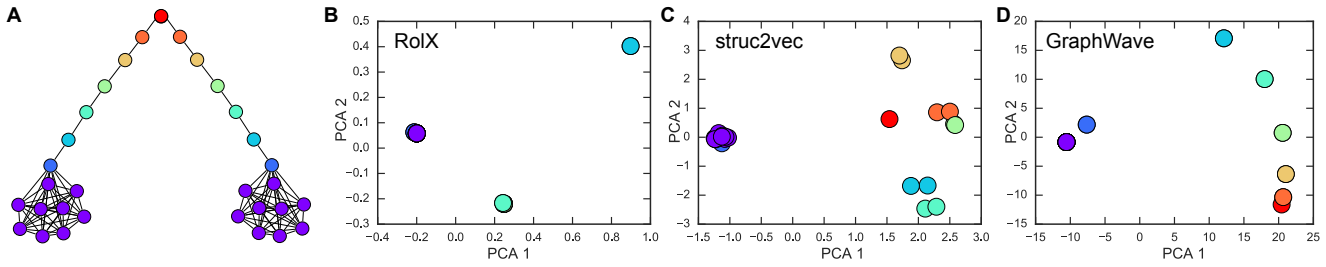


Figure 2: Barbell graph. The graph has 8 distinct classes of structurally equivalent nodes as indicated by color (A). 2D PCA projection of structural embeddings as learned by *RolX* (B), *struc2vec* (C) and GraphWave (D). Projections in (B)-(D) contain the same number of points as there are nodes in the graph (A). Identical embeddings have identical projections, resulting in overlapping points in (B)-(D).

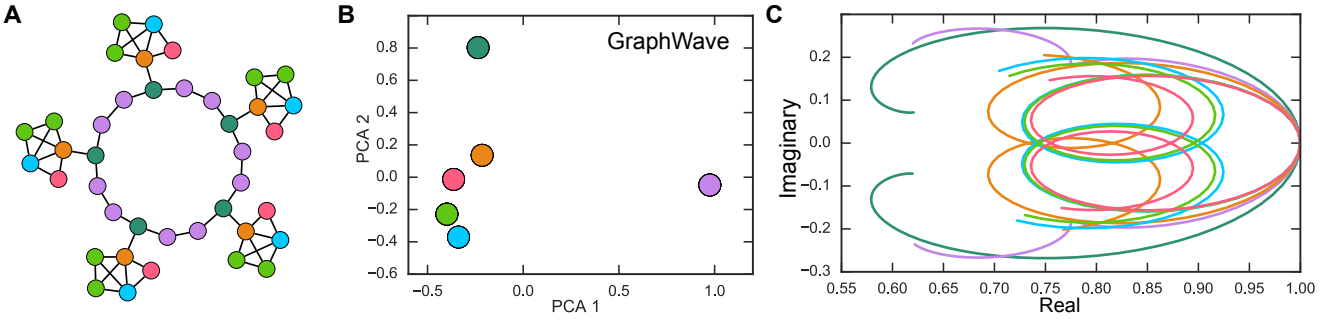


Figure 3: A cycle graph with attached "house" shapes (A). 2D PCA projection of GraphWave's embeddings. Embeddings of structurally equivalent nodes overlap, and GraphWave perfectly recovers the 6 different structural roles (B). Characteristic function for the distribution of the wavelet coefficients (C). Color of a node/curve indicates structural role. (Best seen in color.)

Table 1: Structural role discovery results for different synthetic graphs. (Best seen in color.) Results averaged over 25 synthetically generated graphs. Dashed lines denote perturbed versions of the basic shapes (obtained by randomly adding and removing edges), node colors indicate structural roles. Two best methods are shown in bold.

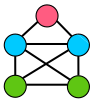
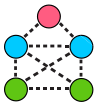
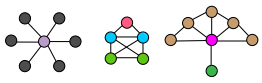
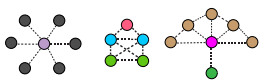
Shapes placed along a cycle graph		Method	Homogeneity	Completeness	Silhouette	Accuracy	F_1-score
House		*DeepWalk*	0.002	0.002	0.29	0.132	0.107
		node2vec	0.005	0.005	0.330	0.077	0.064
		RolX	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
		struc2vec	0.995	0.995	0.451	0.992	0.991
		GraphWave	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
House perturbed		*DeepWalk*	0.059	0.063	0.247	0.097	0.081
		node2vec	0.030	0.032	0.276	0.058	0.046
		RolX	**0.570**	**0.588**	**0.346**	**0.823**	**0.818**
		struc2vec	0.206	0.235	0.180	0.461	0.441
		GraphWave	**0.547**	**0.566**	**0.374**	**0.866**	**0.866**
Varied		*DeepWalk*	0.262	0.233	0.354	0.463	0.428
		node2vec	0.244	0.216	0.400	0.460	0.429
		RolX	**0.841**	**0.862**	**0.736**	**0.836**	**0.836**
		struc2vec	0.629	0.578	0.240	0.571	0.555
		GraphWave	**0.828**	**0.852**	**0.816**	**0.839**	**0.837**
Varied perturbed		*DeepWalk*	0.298	0.267	0.327	0.414	0.387
		node2vec	0.303	0.265	0.360	0.411	0.386
		RolX	**0.638**	**0.627**	**0.418**	**0.718**	**0.714**
		struc2vec	0.457	0.433	0.289	0.490	0.470
		GraphWave	**0.697**	**0.680**	**0.516**	**0.731**	**0.724**

Table 2: Generalization of embeddings across graphs.

Method	Accuracy	F_1-score
DeepWalk	0.531	0.506
node2vec	0.417	0.369
RolX	0.868	0.863
struc2vec	0.767	0.758
GraphWave	**0.936**	**0.936**

resulting characteristic functions (Eq. (2)) in Figure 3C. In general, using characteristic function theory [23], their interpretation is:

- Nodes located in the periphery of the graph struggle to diffuse the signal over the graph, and thus span wavelets that are characterized by a smaller number of non-zero coefficients. Characteristic functions of such nodes thus span a small loop-like 2D curve.
- Nodes located in the core (dense region) of the graph tend to diffuse the signal farther away and reach farther nodes for the same value of t. Characteristic functions of such nodes thus have a farther projection on the x and y axis.

5.3 Generalization of embeddings across graphs

Next we analyze a separate use case, showing how structural embeddings can identify structural similarities of nodes across different graphs. Our goal is to evaluate embeddings for their ability to transfer structural roles from nodes in one graph to nodes in another graph. That is, we test if nodes with the same structural role get embedded close together, even if they come from different graphs.

Generating the graphs. We generate 200 graphs with ground-truth labels for structurally equivalent nodes using the following procedure. First, we determine a skeleton of a graph (either a cycle or linear path, each with probability 0.5), then we select the size of that skeleton (*i.e.*, cycle size or path length, uniformly at random), and finally, we attach a random number of small shapes to the skeleton (5-node house or 5-node chain, each with probability 0.5). To test the methods in noisier settings, we also randomly add 10 random edges between nodes in the basis. We use the different methods to compute embeddings for every node in each graph and then use the learned embeddings to predict each node's ground-truth structural role, for example, the corner of a house. To evaluate method's ability to generalize across graphs, we use 10-fold cross validation and predict the label of each test node in a given graph based on its 4-nearest neighbors (as defined in Section 2.2) in all other graphs. We then measure the average accuracy and F_1-score.

Results. GraphWave outperforms all alternative methods on both classification metrics (Table 2). We see that recent unsupervised node representation learning methods perform poorly, and that the second best performing method is *RolX*. However, GraphWave outperforms both *RolX* (by 8% in F_1-score and 8% in accuracy), and *struc2vec* (by 23% in F_1-score and 22% in accuracy). This highlights GraphWave's ability to learn structural signatures which are meaningful across different graphs, with high predictive power.

5.4 Scalability and sensitivity to noise

We next analyze scalability of GraphWave and its sensitivity to noise in input graphs.

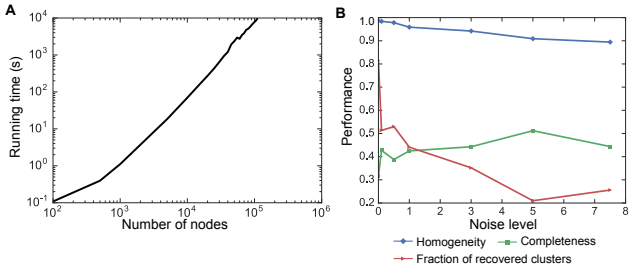


Figure 4: (A) Running time of GraphWave (in seconds) as a function of the number of nodes in an input graph. (B) Performance of GraphWave as a function of the noise level. Noise level is given by the percentage of initial number of edges that are randomly rewired. Results are averaged across ten runs.

Scalability. To evaluate how GraphWave scales to large graphs, we generate synthetic graphs of increasing size. Each graph has a skeleton given by a cycle graph to which we attach a number of network motifs, as described in Section 5.3. We take each graph and run GraphWave to learn structural node embeddings. Figure 4A shows GraphWave's running time as a function of the number of nodes in the graph. As the computation of scaling wavelet coefficients takes linear time in the number of edges, GraphWave lends itself to a fast algorithm that can be efficiently implemented using sparse matrix representation. A potential bottleneck of GraphWave is the conversion of the distribution induced by these wavelets into an Euclidean space via an empirical characteristic function (Section 2.2). To efficiently evaluate the characteristic function, GraphWave leverages the sparsity of the wavelet's coefficients. Note that wavelet coefficients are sparse because of GraphWave's approach to the selection of scales, which does not allow the signal to propagate too far away over the network (Section 4). This approach reduces the computation of embeddings to a set of sparse matrix multiplications and applications of element-wise functions on the non-zero elements of sparse matrices. In contrast, *struc2vec* does not scale to large graphs, as it takes as long as 260 seconds on a network of only 100 nodes, whereas GraphWave can scale to graphs of comparable sizes.

Sensitivity to noise. We next evaluate GraphWave's sensitivity to noise in an input graph. The noise is injected into the graph in the same manner as in the "varied perturbed" setting in Table 1. For each graph, we first learn structural node embeddings using GraphWave, then we cluster the embeddings using affinity propagation clustering algorithm, and finally, we evaluate the quality of clusters against ground-truth structural roles. Note that affinity propagation does not need the number of clusters as input but produces the clusters automatically. This is important in this experiment as the meaning of each cluster can be hindered because of high levels of noise. In addition to homogeneity and completeness metrics, we also report the number of detected clusters (as a fraction of the maximum number of possible clusters, that is, the number of nodes N) as a measure of the richness of roles recovered by GraphWave. Figure 4B shows performance of GraphWave as we vary the level of noise added to the input graph. Results show that GraphWave's performance degrades gracefully, even in the presence of strong noise.

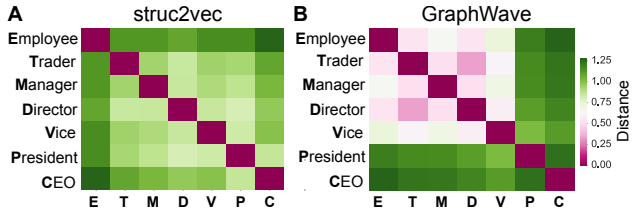


Figure 5: Heat maps indicate average distance between roles in the Enron email graph, as determined by *struc2vec* (A) and GraphWave (B).

6 EXPERIMENTS ON REAL-WORLD GRAPHS

We proceed by applying GraphWave to three real-world networks.

6.1 Mirrored Karate network

We first consider a mirrored Karate network, using the same experimental setup as in [27]. The mirrored Karate network is created by taking two copies of Zachary's karate network and adding a given number of random edges, each edge connecting mirrored nodes from different copies representing the same individual in the karate club (*i.e.*, mirrored edges). The goal of the experiment is to identify mirrored nodes by capturing structural similarity while varying the number of mirrored edges in the network.

We measure the percentage of nodes whose nearest neighbor corresponds to its true structural equivalent, comparing GraphWave with the various baselines. As the number of mirrored edges varies from 1 to 25, the results are remarkably consistent. GraphWave achieves an average accuracy of 83.2% (with a minimum score of 75.2% and a maximum score of 86.2%), *RolX* achieves an average of 82.2% (min/max of 79.4% and 84.5%, respectively), *struc2vec* scores 52.5% (43.3% and 59.4%). Alternatively, both *node2vec* and *DeepWalk* never score higher than 8.5%, since they are embedding based on proximity rather than structural similarity.

6.2 Enron email network

Next we consider an email network encoding email communication between employees in a company. We expect structural equivalences in job titles due to corporate organizational hierarchy.

Data and setup. Nodes represent Enron employees and edges correspond to email communication between the employees [21]. An employee has one of seven functions in the company (*e.g.*, CEO, president, manager). These functions provide ground-truth information about roles of the corresponding nodes in the network. We use an embedding learning algorithm to learn an embedding for every Enron employee. We then use these embeddings to compute the average ℓ_2^2 distance between every two categories of employees.

Results. GraphWave captures intricate organizational structure of Enron (Figure 5). For example, CEOs and presidents are structurally distant from all other job titles. This indicates their unique position in the email exchange graph, which can be explained by their local graph connectivity patterns standing out from the others. Traders, on the other hand, appear very far from presidents and are closer to directors. In contrast, *struc2vec* is less successful at revealing intricate relationships between the job titles, yielding an almost uniform distribution of distances between every class. We assess the separation between "top" job titles (CEO and President) and lower levels in the job title hierarchy of all three methods (GraphWave, *struc2vec*, and *RolX*) in Table 3. GraphWave achieves 28% higher homogeneity and 139% higher completeness than *RolX* (and performs even better compared to *struc2vec*).

6.3 European airline networks

We next analyze a collection of airline networks encoding direct flights between airports that are operated by different airlines. We expect structural equivalences in airports of different airlines due to the known hub-and-spoke structure, such as Frankfurt (FRA) for Lufthansa and Charles de Gaulle (CDG) for Air France.

Data and setup. We consider six airlines operating flights between European airports [2]: 4 commercial (Air France, Easyjet, Lufthansa, and RyanAir) and 2 cargo airlines (TAP Portugal, and European Airline Transport). Each airline is represented with a graph, where nodes represent airports/destinations and edges stand for direct flights between the airports. Altogether, there are 45 airports labeled as: hub airports, regional hubs, commercial hubs, and focus cities,[2] which we use as ground-truth structural roles. For each airline graph, we use a feature learning algorithm to learn an embedding of every airport in the graph. We then stack airport embeddings from different graphs and use them as input to t-SNE (for visualization) and as input to agglomerative clustering (for measuring the homogeneity, completeness, and silhouette score) that returns four airport clusters, indicative of the four ground-truth structural roles.

Results. We measure how well airports in each cluster correspond to the ground-truth structural roles. GraphWave outperforms alternative methods on all three clustering metrics (Table 3). It outperforms *RolX* and *struc2vec* by 27% and 24%, respectively (homogeneity), and by 12% and 44% (completeness), with a substantially higher silhouette score.

Figure 6 shows t-SNE visualizations of airport embeddings. We find that *struc2vec* learns embeddings that are dominated by airline networks (note how different shapes are localized in *struc2vec*'s t-SNE plot). In particular, airports from the same airline network, *e.g.*, Ryanair, are trivially embedded close together. This indicates that *struc2vec* cannot generalize embeddings across different airline networks and thus cannot successfully identify which airports are structurally equivalent across airlines (*i.e.*, hub/focus cities for each airline). Further examining the figure, we see that *RolX* learns embeddings whose projections do not exhibit any obvious pattern. In contrast, GraphWave learns embeddings that are indicative of airports' structural equivalences (note how different colors are localized in GraphWave's t-SNE plot). In particular, in GraphWave, airports with the same structural role are embedded close together even if they come from different airline networks, demonstrating GraphWave' ability to learn meaningful structural embeddings for real-world networks.

7 CONCLUSION

We have developed a new method for learning structural embeddings in networks. Our approach, GraphWave, uses spectral graph

[2]These labels were manually curated from Wikipedia based on M. Garrett: Airport Hubs. *Encyclopedia of Transportation: Social Science and Policy* (2014).

Table 3: Clustering results for Enron and airline networks.

Dataset	Method	Homogen.	Comple.	Silhouette
Enron	*RolX*	0.090	0.028	0.425
	struc2vec	0.003	0.018	0.435
	GraphWave	**0.115**	**0.067**	**0.577**
Airlines	*RolX*	0.244	0.326	-0.054
	struc2vec	0.250	0.254	-0.035
	GraphWave	**0.310**	**0.365**	**0.050**

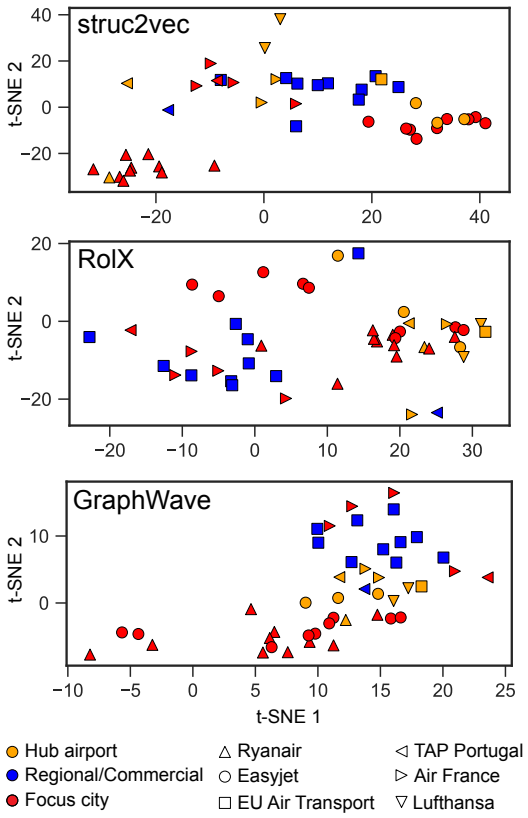


Figure 6: t-SNE projections of airport embeddings.

wavelets to generate a structural embedding for each node, which we accomplish by treating the wavelets as a distributions and evaluating the resulting characteristic functions. Considering the wavelets as distributions instead of vectors is a key insight needed to capture structural similarity in graphs.

Our method provides mathematical guarantees on the optimality of learned structural embeddings. Using spectral graph theory, we prove that structurally equivalent (or similar) nodes have near-identical (or similar) embeddings in GraphWave. Various experiments on real and synthetic networks provide empirical evidence for our analytical results and yield large gains in performance over state-of-the-art baselines. For future work, these embeddings could be used for transfer learning, leveraging data from a well-explored region of the graph to infer knowledge about less-explored regions.

REFERENCES

[1] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. 2011. The wave kernel signature: A quantum mechanical approach to shape analysis. In *ICCV Computer Vision Workshop*. 1626–1633.
[2] Alessio Cardillo et al. 2013. Emergence of network features from multiplexity. *Scientific Reports* 3 (2013), 1344.
[3] Fan Chung. 2007. The heat kernel as the PageRank of a graph. *PNAS* 104, 50 (2007), 19735–19740.
[4] Lewis Coburn et al. 1966. Weyl's theorem for nonnormal operators. *The Michigan Mathematical Journal* 13, 3 (1966), 285–288.
[5] Ronald Coifman et al. 2006. Diffusion maps. *Applied and Computational Harmonic Analysis* 21, 1 (2006), 5–30.
[6] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*. 2702–2711.
[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
[8] David K Duvenaud et al. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*. 2224–2232.
[9] Alberto Garcia-Duran and Mathias Niepert. 2017. Learning Graph Representations with Embedding Propagation. In *NIPS*.
[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *ICML* (2017).
[11] Sean Gilpin, Tina Eliassi-Rad, and Ian Davidson. 2013. Guided learning for role discovery: framework, algorithms, and applications. In *KDD*. 113–121.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[13] William Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.
[14] William Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin* (2017).
[15] David Hammond et al. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30, 2 (2011), 129–150.
[16] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. RolX: structural role extraction & mining in large graphs. In *KDD*. 1231–1239.
[17] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. 2011. It's who you know: graph mining using recursive structural features. In *KDD*. 663–671.
[18] Ruoming Jin et al. 2014. Scalable and axiomatic ranking of network role similarity. *ACM TKDD* 8, 1 (2014), 3.
[19] Ruoming Jin, Victor Lee, and Hui Hong. 2011. Axiomatic ranking of network role similarity. In *KDD*. 922–930.
[20] Thomas Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR* (2017).
[21] Bryan Klimt and Yiming Yang. 2004. Introducing the Enron Corpus.. In *CEAS*.
[22] Risi Kondor and John Lafferty. 2002. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, Vol. 2. 315–322.
[23] Eugene Lukacs. 1970. Characteristic functions. (1970).
[24] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR*, Vol. 1. 3.
[25] Maks Ovsjanikov et al. 2010. One point isometric matching with the heat kernel. In *Computer Graphics Forum*, Vol. 29. 1555–1564.
[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*. 701–710.
[27] Leonardo Ribeiro, Pedro Saverese, and Daniel Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*. 385–394.
[28] Andrew Rosenberg and Julia Hirschberg. 2007. V-Measure: a Conditional Entropy-Based External Cluster Evaluation Measure. In *EMNLP-CoNLL*, Vol. 7. 410–420.
[29] Raif Rustamov and Leonidas Guibas. 2013. Wavelets on graphs via deep learning. In *NIPS*. 998–1006.
[30] David Shuman et al. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.
[31] David Shuman et al. 2016. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis* 40, 2 (2016), 260–291.
[32] David Shuman, Pierre Vandergheynst, and Pascal Frossard. 2011. Chebyshev polynomial approximation for distributed signal processing. In *DCOSS*. 1–8.
[33] Jian Sun et al. 2009. A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion. In *Computer Graphics Forum*, Vol. 28. 1383–1392.
[34] Nicolas Tremblay et al. 2014. Graph wavelets for multiscale community mining. *IEEE TSP* 62, 20 (2014), 5227–5239.
[35] Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*, Vol. 33. 40–48.