Deep Inductive Graph Representation Learning

Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed

Abstract—This paper presents a general inductive graph representation learning framework called DeepGL for learning deep node *and* edge features that generalize across-networks. In particular, DeepGL begins by deriving a set of base features from the graph (e.g., graphlet features) and automatically learns a multi-layered hierarchical graph representation where each successive layer leverages the output from the previous layer to learn features of a higher-order. Contrary to previous work, DeepGL learns *relational functions* (each representing a feature) that naturally generalize across-networks and are therefore useful for graph-based transfer learning tasks. Moreover, DeepGL naturally supports attributed graphs, learns interpretable inductive graph representations, and is space-efficient (by learning sparse feature vectors). In addition, DeepGL is expressive, flexible with many interchangeable components, efficient with a time complexity of $\mathcal{O}(|E|)$, and scalable for large networks via an efficient parallel implementation. Compared with recent methods, DeepGL is (1) *effective* for across-network transfer learning tasks *and* large (attributed) graphs, (2) *space-efficient* requiring up to 6x less memory, (3) *fast* with up to 106x speedup in runtime performance, and (4) *accurate* with an average improvement in AUC of 20 percent or more on many learning tasks and across a wide variety of networks.

Index Terms—Graph representation learning, inductive representation learning, relational function learning, transfer learning, graph-based feature learning, higher-order structures

1 Introduction

Learning a useful graph representation lies at the heart *and* success of many machine learning tasks such as node and link classification [1], [2], anomaly detection [3], link prediction [4], dynamic network analysis [5], community detection [6], role discovery [7], visualization and sense-making [8], network alignment [9], and many others. Indeed, the success of machine learning methods largely depends on data representation [10], [11]. Methods capable of learning such representations have many advantages over feature engineering in terms of cost and effort. For a survey and taxonomy of relational representation learning, see [11].

Recent work has largely been based on the popular skip-gram model [12] originally introduced for learning vector representations of words in the natural language processing (NLP) domain. In particular, DeepWalk [13] applied the successful word embedding framework from [14] (called word2vec) to embed the nodes such that the co-occurrence frequencies of pairs in short random walks are preserved. More recently, node2vec [15] introduced hyperparameters to DeepWalk that tune the depth and breadth of the random walks. These approaches have been extremely successful and have shown to outperform a number of existing methods on tasks such as node classification.

However, the past work has focused on learning only *node features* [13], [15], [16] for a specific graph. Features from these methods do *not* generalize to other networks and thus are unable to be used for across-network transfer learning tasks.[1] In contrast, DeepGL learns *relational functions* that generalize for computation on any arbitrary graph, and therefore naturally supports across-network transfer learning tasks such as across-network link classification, network alignment, graph similarity, among others. Existing methods are also not space-efficient as the node feature vectors are completely dense. For large graphs, the space required to store these dense features can easily become too large to fit in-memory. The features are also notoriously difficult to interpret and explain which is becoming increasingly important in practice [17], [18]. Furthermore, existing embedding methods are also unable to capture higher-order subgraph structures. Finally, these methods are also inefficient with runtimes that are orders of magnitude slower than the algorithms presented in this paper (as shown later in Section 4). Other key differences and limitations are discussed below.

In this work, we present a general, expressive, and flexible *deep graph representation learning framework* called DeepGL that overcomes many of the above limitations.[2,3] Intuitively, DeepGL begins by deriving a set of base features using the graph structure and any attributes (if available).[4]

- *R. A. Rossi is with Adobe Research, San Jose, CA 95110-2704. E-mail: rrossi@adobe.com.*
- *R. Zhou is with Google, Mountain View, CA 94043. E-mail: rongzhou@google.com.*
- *N. K. Ahmed is with Intel Labs, Santa Clara, CA 95054. E-mail: nesreen.k.ahmed@intel.com.*

Manuscript received 15 Feb. 2018; revised 1 Oct. 2018; accepted 14 Oct. 2018. Date of publication 1 Nov. 2018; date of current version 4 Feb. 2020. (Corresponding author: Ryan A. Rossi.) Recommended for acceptance by M. Wang. Digital Object Identifier no. 10.1109/TKDE.2018.2878247

1. The terms transfer learning and inductive learning are used interchangeably.
2. This manuscript first appeared in [19].
3. Note a deep learning method as defined by Bengio et al. [20], [21] is one that learns multiple levels of representation with higher levels capturing more abstract concepts through a deeper composition of computations [10], [22]. This definition includes neural network approaches as well as DeepGL and many other deep learning paradigms.
4. The base graph features computed using the graph are functions since they have precise definitions and can be computed on any graph.

TABLE 1
Summary of Notation

G	(un) directed (attributed) graph
$\mathbf{A}$	sparse adjacency matrix of the graph $G = (V, E)$
N, M	number of nodes and edges in the graph
F, L	number of learned features and layers
K	number of relational operators (aggregator functions)
$\mathcal{G}$	set of graph elements $\{g_1, g_2, \ldots\}$ (nodes, edges)
d_v^+, d_v^-, d_v	outdegree, indegree, degree of vertex v
$\Gamma^+(g_i)$, $\Gamma^-(g_i)$	out/in neighbors of graph element g_i
$\Gamma(g_i)$	neighbors (adjacent graph elements) of g_i
$\Gamma_\ell(g_i)$	ℓ-neighborhood $\Gamma(g_i) = \{g_j \in \mathcal{G} \,\vert\, \text{dist}(g_i, g_j) \leq \ell\}$
$\text{dist}(g_i, g_j)$	shortest distance between g_i and g_j
S	set of graph elements related to g_i, e.g., $S = \Gamma(g_i)$
$\mathbf{X}$	a feature matrix
$\mathbf{x}$	an N or M-dimensional feature vector
x_i	the ith element of $\mathbf{x}$ for graph element g_i
$\vert\mathbf{X}\vert$	number of nonzeros in a matrix $\mathbf{X}$
$\mathcal{F}$	set of *feature definitions/functions* from DeepGL
$\mathcal{F}_k$	kth feature layer (where k is the depth)
f_i	relational function (definition) of $\mathbf{x}_i$
Φ	set of relational operators (aggregators) $\Phi = \{\Phi_1, \ldots, \Phi_K\}$
$\mathbb{K}(\cdot)$	a feature score function
λ	tolerance/feature similarity threshold
α	transformation hyperparameter (e.g., bin size in log binning $0 \leq \alpha \leq 1$)
$\mathbf{x}' = \Phi_i\langle\mathbf{x}\rangle$	relational operator applied to each graph element

Matrices are bold upright roman letters; vectors are bold lowercase letters.

The base features are iteratively composed using a set of learned *relational feature operators* that operate over the feature values of the (distance-ℓ) neighbors of a graph element (node, edge; see Table 1) to derive higher-order features from lower-order ones forming a hierarchical graph representation where each layer consists of features of increasingly higher orders. At each feature layer, DeepGL searches over a space of relational functions defined compositionally in terms of a set of *relational feature operators* applied to each feature given as output in the previous layer. Features (or relational functions) are retained if they are novel and thus add important information that is not captured by any other feature in the set. See below for a summary of the advantages and properties of DeepGL.

1.1 Summary of Contributions

The proposed framework, DeepGL, overcomes many limitations of existing work and has the following key properties:

- *Novel framework*: This paper presents a general inductive graph representation learning framework called DeepGL for learning inductive node and edge relational functions (features) that generalize for across-network transfer learning tasks in large (attributed) networks.
- *Inductive representation learning*: Contrary to existing node embedding methods [13], [15], [16], DeepGL naturally supports across-network transfer learning tasks as it learns relational functions that generalize for computation on any arbitrary graph.
- *Sparse feature learning*: It is space-efficient by learning a sparse graph representation that requires up to 6x less space than existing work.
- *Efficient, parallel, and scalable*: It is fast with a runtime that is linear in the number of edges. It scales to large graphs via a simple and efficient parallelization. Notably, strong scaling results are observed in Section 4.
- *Attributed graphs*: DeepGL is also naturally able to learn node and edge features (relational functions) from both attributes (if available) *and* the graph structure.

2 Framework

This section presents the DeepGL framework. Since the framework naturally generalizes for learning node and edge representations, it is described generally for a set of graph elements (e.g., nodes or edges).[5] A summary of notation is provided in Table 1.

2.1 Base Graph Features

The first step of DeepGL (Algorithm 1) is to derive a set of *base graph features*[6] using the graph topology and attributes (if available). Initially, the feature matrix $\mathbf{X}$ contains only the attributes given as input by the user. If no attributes are provided, then $\mathbf{X}$ will consist of only the base features derived below. Note that DeepGL can use any arbitrary set of base features, and thus it is not limited to the base features discussed below.

Given a graph $G = (V, E)$, we first derive simple base features such as in/out/total/weighted degree and k-core numbers for each graph element (node, edge) in G. For edge feature learning we derive edge degree features for each edge $(v, u) \in E$ and each $\circ \in \{+, \times\}$ as follows:

$$\left[\; d_v^+ \circ d_u^+,\; d_v^- \circ d_u^-,\; d_v^- \circ d_u^+,\; d_v^+ \circ d_u^-,\; d_v \circ d_u \;\right], \tag{1}$$

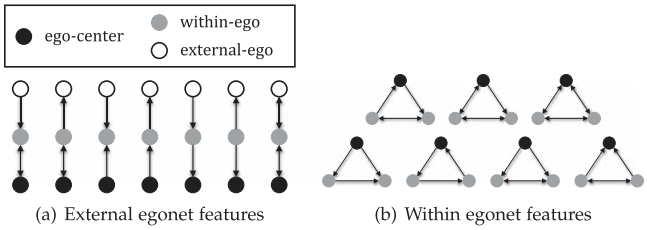
where $d_v = d_v^+ \circ d_v^-$ and recall from Table 1 that d_v^+, d_v^-, and d_v denote the out/in/total degree of v. In addition, egonet features are also used [23]. Given a node v and an integer ℓ, the ℓ-egonet of v is defined as the set of nodes ℓ-hops away from v (i.e., distance at most ℓ) and all edges and nodes between that set. More generally, we define the ℓ-egonet of a graph element g_i as follows:

Definition 1 (ℓ-egonet). *Given a graph element g_i (node, edge) and an integer ℓ, the ℓ-egonet of g_i is defined as the set of graph elements ℓ-hops away from g_i (i.e., distance at most ℓ) and all edges (or nodes) between that set.*

For massive graphs, one may set $\ell = 1$ hop to balance the tradeoffs between accuracy/representativeness and scalability. The $\ell = 1$ external and within-egonet features for nodes are provided in Fig. 1 and used as base features in DeepGL-node. For all the above base features, we also derive variations based on direction (in/out/both) and weights (weighted/unweighted). Observe that DeepGL naturally supports many other graph properties including efficient/linear-time properties such as PageRank. Moreover, fast approximation methods with provable bounds can also be used to derive features such as the local coloring number and largest clique centered at the neighborhood of each graph

5. For convenience, DeepGL-edge and DeepGL-node are sometimes used to refer to the edge and node representation learning variants of DeepGL, respectively.

6. The term *graph feature* refers to an edge or node feature.



(a) External egonet features (b) Within egonet features

Fig. 1. *Egonet Features*. The set of base (ℓ=1 hop)-egonet graph features. (a) The external egonet features; (b) the within egonet features. See the legend for the vertex types: Ego-center (•), within-egonet vertex (•), and external egonet vertices (○).

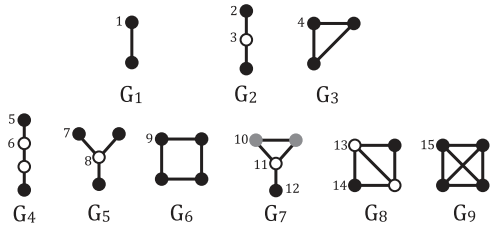


Fig. 2. Summary of the nine node graphlets and 15 orbits (graphlet automorphisms) with 2-4 nodes.

element (node, edge) in G. All of the above base features are concatenated (as column vectors) to the feature matrix $\mathbf{X}$.

In addition, we decompose the input graph G into its smaller subgraph components called graphlets (network motifs, induced subgraphs) [24] using local graphlet decomposition methods [25] and concatenate the graphlet count-based feature vectors to the feature matrix $\mathbf{X}$.

Definition 2 (Graphlet). *A graphlet $G_t = (V_k, E_k)$ is an induced subgraph consisting of a subset $V_k \subset V$ of k vertices from $G = (V, E)$ together with all edges whose endpoints are both in this subset $E_k = \{\forall e \in E \,|\, e = (u, v) \wedge u, v \in V_k\}$.*

A k-graphlet is defined as an *induced subgraph* with k-vertices. Alternatively, the nodes of every graphlet can be partitioned into a set of automorphism groups called orbits [26]. Each unique node (edge) position in a graphlet is called an *automorphism orbit*, or just orbit. More formally,

Definition 3 (Orbit). *An automorphism of a k-vertex graphlet $G_t = (V_k, E_k)$ is defined as a permutation of the nodes in G_t that preserves edges and non-edges. The automorphisms of G_t form an automorphism group denoted as $Aut(G_t)$. A set of nodes V_k of graphlet G_t define an* orbit *iff (i) for any node $u \in V_k$ and any automorphism π of G_t, $u \in V_k \Longleftrightarrow \pi(u) \in V_k$; and (ii) if $v, u \in V_k$ then there exists an automorphism π of G_t and a $\gamma > 0$ such that $\pi^\gamma(u) = v$.*

This work derives such features by counting all node or edge *orbits* with up to 4 and/or 5-vertex graphlets. Orbits (graphlet automorphisms) are counted for each node or edge in the graph based on whether a node or edge-based feature representation is warranted (as our approach naturally generalizes to both). Note there are 15 node and 12 edge orbits with 2-4 nodes; and 73 node and 68 edge orbits with 2-5 nodes. Unless otherwise mentioned, this work uses all 15 node orbits shown in Fig. 2.

A key advantage of DeepGL lies in its ability to naturally handle attributed graphs. In particular, any set of initial attributes given as input can simply be concatenated with $\mathbf{X}$ and treated the same as any other initial base feature derived from the topology.

TABLE 2
Examples of Relational Feature Operators

Operator	Definition
mean	$\Phi\langle S, \mathbf{x}\rangle = \frac{1}{\lvert S\rvert}\sum_{s_j \in S} x_j$
sum	$\Phi\langle S, \mathbf{x}\rangle = \sum_{s_j \in S} x_j$
maximum	$\Phi\langle S, \mathbf{x}\rangle = \max_{s_j \in S} x_j$
Hadamard	$\Phi\langle S, \mathbf{x}\rangle = \prod_{s_j \in S} x_j$
Weight. L^p	$\Phi\langle S, \mathbf{x}\rangle = \sum_{s_j \in S} \lvert x_i - x_j\rvert^p$
RBF	$\Phi\langle S, \mathbf{x}\rangle = \exp\Big(-\frac{1}{\sigma^2}\sum_{s_j \in S}\big[x_i - x_j\big]^2\Big)$

The term relational operator is used more generally to refer to any relational function applied over the neighborhood of a node or edge (or more generally any set S). Note that DeepGL *is flexible and generalizes to any arbitrary set of relational feature operators. The set of relational feature operators can be learned via cross-validation. Recall the notation from Table 1. For generality, S is defined in Table 1 as a set of related graph elements (nodes, edges) of g_i and thus $s_j \in S$ may be an edge $s_j = e_j$ or a node $s_j = v_j$; in this work $S \in \{\Gamma_\ell(g_i), \Gamma^+_\ell(g_i), \Gamma^-_\ell(g_i)\}$. The relational operators generalize to ℓ-distance neighborhoods (e.g., $\Gamma_\ell(g_i)$ where ℓ is the distance). Note $\mathbf{x} = \big[x_1\, x_2 \cdots x_i \cdots\big]$ is a vector and x_i is the ith element of $\mathbf{x}$ for g_i.*

2.2 Relational Function Space & Expressivity

In this section, we formulate the space of relational functions[7] that can be expressed and searched over by DeepGL.

Definition 4 (Relational function). *A relational function (feature) is defined as a composition of relational feature operators applied to an initial base feature $\mathbf{x}$.*

Definition 5. *A relational function f is said to be of order-h iff*

$$\big(\underbrace{\Phi_k \circ \cdots \circ \Phi_j \circ \Phi_i}_{h \text{ times}}\big)(\mathbf{x}), \tag{2}$$

where $\mathbf{x}$ is an arbitrary base feature applied to h relational feature operators.

Note the relational feature operators can obviously be repeated in the relational function. For instance, in the extreme case all h relational feature operators may represent the mean relational feature operator defined in Table 2. Recall that unlike recent node embedding methods [13], [15], [16], the proposed approach learns graph functions that are transferable across-networks for a variety of important graph-based transfer learning tasks such as *across-network* prediction, anomaly detection, graph similarity, matching, among others.

2.2.1 Composing Relational Functions

The space of relational functions searched via DeepGL is defined *compositionally* in terms of a set of *relational feature operators* $\Phi = \{\Phi_1, \ldots, \Phi_K\}$.[8] A few relational feature operators are defined formally in Table 2; see [11] (pp. 404) for a wide variety of other useful relational feature operators. The expressivity of DeepGL (space of relational functions expressed by DeepGL) depends on a few flexible and interchangeable components including:

7. The terms graph function and relational function are used interchangeably.

8. Note DeepGL may also leverage traditional feature operators used for *i.i.d.* data.

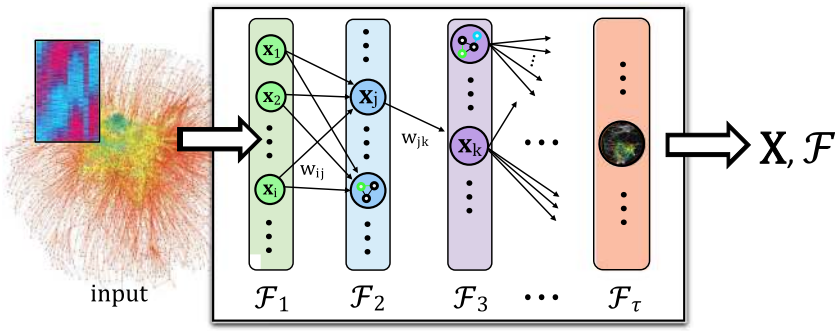


Fig. 3. Overview of the DeepGL architecture for graph representation learning. Let $\mathbf{W} = [w_{ij}]$ be a matrix of feature weights where w_{ij} (or W_{ij}) is the weight between the feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Notice that $\mathbf{W}$ has the constraint that $i < j < k$ and $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$ are increasingly deeper. Each feature layer $\mathcal{F}_h \in \mathcal{F}$ defines a set of unique *relational functions* $\mathcal{F}_h = \{\ldots, f_k, \ldots\}$ of order h (depth) and each $f_k \in \mathcal{F}_h$ denotes a *relational function*. Further, let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots \cup \mathcal{F}_\tau$ and $|\mathcal{F}| = |\mathcal{F}_1| + |\mathcal{F}_2| + \cdots + |\mathcal{F}_\tau|$. Moreover, the layers are ordered where $\mathcal{F}_1 < \mathcal{F}_2 < \cdots < \mathcal{F}_\tau$ such that if $i < j$ then $\mathcal{F}_j$ is said to be a deeper layer $w.r.t.$ $\mathcal{F}_i$. See Table 1 for a summary of notation.

i) the initial base features derived using the graph structure, initial input attributes (if available), or both,
ii) a set of *relational feature operators* $\Phi = \{\Phi_1, \ldots, \Phi_K\}$,
iii) the sets of "related graph elements" $S \in \mathcal{S}$ (e.g., the in/out/all neighbors within ℓ hops of a given node/edge) that are used with each relational feature operator $\Phi_p \in \Phi$, and finally,
iv) the number of times each relational function is composed with another (i.e., the depth).

Observe that under this formulation each feature vector $\mathbf{x}'$ from $\mathbf{X}$ (that is not a base feature) can be written as a composition of relational feature operators applied over a base feature. For instance, given an initial *base feature* $\mathbf{x}$, by abuse of notation let $\mathbf{x}' = \Phi_k(\Phi_j(\Phi_i\langle\mathbf{x}\rangle)) = (\Phi_k \circ \Phi_j \circ \Phi_i)(\mathbf{x})$ be a feature vector given as output by applying the relational function constructed by composing the *relational feature operators* $\Phi_k \circ \Phi_j \circ \Phi_i$ to every graph element $g_i \in \mathcal{G}$ and its set S of related elements.[9] Obviously, more complex relational functions are easily expressed such as those involving compositions of different relational feature operators (and possibly different sets of related graph elements). Furthermore, DeepGL is able to learn relational functions that often correspond to increasingly higher-order subgraph features based on a set of initial lower-order (base) subgraph features (Fig. 3). Intuitively, just as filters are used in Convolutional Neural Networks (CNNs) [10], one can think of DeepGL in a similar way, but instead of simple filters, we have features derived from lower-order subgraphs being combined in various ways to capture higher-order subgraph patterns of increasingly complexity at each successive layer.

2.2.2 Summation and Multiplication of Relational Functions

We can also construct a wide variety of functions compositionally by adding and multiplying relational functions (e.g., $\Phi_i + \Phi_j$, and $\Phi_i \times \Phi_j$). More specifically, any class of functions that are closed under addition and multiplication can be used as base functions in this context. A *sum of relational functions* is similar to an OR operation in that two instances are "close" if either has a large value, and similarly, a *product of relational functions* can be viewed as an AND operation as two instances are close if both relational functions have large values. This is similar to many existing architectures for learning complex functions such as the AND-like or OR-like units in convolutional networks [27], among others [10].

2.3 Searching the Relational Function Space

A general and flexible framework for DeepGL is given in Algorithm 1. Recall that DeepGL begins by deriving a set of base features which are used as a basis for learning deeper and more discriminative features of increasing complexity (Line 2). The base feature vectors are then transformed (Line 3). For instance, one may transform each feature vector $\mathbf{x}_i$ using logarithmic binning as follows: sort $\mathbf{x}_i$ in ascending order and set the αN graph elements (nodes/edges) with smallest values to 0 where $0 < \alpha < 1$, then set α fraction of remaining graph elements with smallest value to 1, and so on. Observe that we only need to store the nonzero feature values. Thus, we avoid explicitly storing αN values for each feature by storing each feature as a sparse feature vector. See Section 3.2 for further details. Many other techniques exist for transforming the feature vectors and the selected technique will largely depend on the graph structure.

The framework proceeds to learn a hierarchical graph representation (Fig. 3) where each successive layer represents increasingly deeper higher-order (edge/node) graph functions: $\mathcal{F}_1 < \mathcal{F}_2 < \cdots < \mathcal{F}_\tau$ $s.t.$ if $i < j$ then $\mathcal{F}_j$ is said to be deeper than $\mathcal{F}_i$. In particular, the feature layers $\mathcal{F}_2, \mathcal{F}_3, \ldots, \mathcal{F}_\tau$ are derived as follows (Algorithm 1 Lines 4-10): First, we derive the feature layer $\mathcal{F}_\tau$ by searching over the space of graph functions that arise from applying the relational feature operators Φ to each of the novel features $f_i \in \mathcal{F}_{\tau-1}$ learned in the previous layer (Algorithm 1 Line 5). An algorithm for deriving a feature layer is provided in Algorithm 2. Further, an intuitive example is provided in Fig. 4. Next, the feature vectors from layer $\mathcal{F}_\tau$ are transformed in Line 6 as discussed previously.

The resulting features in layer τ are then evaluated. The feature evaluation routine (in Algorithm 1 Line 7) chooses the important features (relational functions) at each layer τ from the space of novel relational functions (at depth τ) constructed by applying the relational feature operators to each

9. For simplicity, we use $\Phi\langle\mathbf{x}\rangle$ (whenever clear from context) to refer to the application of Φ to all sets S derived from each graph element $g_i \in \mathcal{G}$ and thus the output of $\Phi\langle\mathbf{x}\rangle$ in this case is a feature vector with a single feature-value for each graph element.

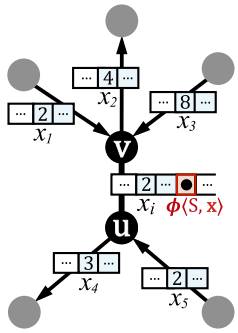


Fig. 4. An intuitive example for an edge $e = (v, u)$ and a relational operator $\Phi \in \Phi$. Suppose $\Phi =$ relational sum operator and $S = \Gamma_\ell(e_i) = \{e_1, e_2, e_3, e_4, e_5\}$ where $\ell = 1$ (distance-1 neighborhood), then $\Phi\langle S, \mathbf{x}\rangle = 19$. Now, suppose $S = \Gamma^+_\ell(e_i) = \{e_2, e_4\}$ then $\Phi\langle S, \mathbf{x}\rangle = 7$ *and* similarly, if $S = \Gamma^-_\ell(e_i) = \{e_1, e_3, e_5\}$ then $\Phi\langle S, \mathbf{x}\rangle = 12$. Note x_i is the ith element of $\mathbf{x}$ for e_i.

feature (relational function) learned (and given as output) in the previous layer $\tau - 1$. Notice that DeepGL is extremely flexible as the feature evaluation routine (Algorithm 3) called in Line 7 of Algorithm 1 is completely interchangeable and can be fine-tuned for specific applications and/or data. This approach derives a score between pairs of features. Pairs of features $\mathbf{x}_i$ and $\mathbf{x}_j$ that are *strongly dependent* as determined by the hyperparameter λ and evaluation criterion $\mathbb{K}$ are assigned $W_{ij} = \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j)$ and $W_{ij} = 0$ otherwise (Algorithm 3 Line 2-6). More formally, let E_F denote the set of connections representing dependencies between features:

$$E_F = \big\{(i,j) \mid \forall (i,j) \in |\mathcal{F}| \times |\mathcal{F}| s.t. \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) > \lambda\big\}. \quad (3)$$

The result is a *weighted feature dependence graph* $\mathcal{G}_F = (V_F, E_F)$ where a relatively large edge weight $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) = W_{ij}$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ indicates a potential dependence (or similarity/correlation) between these two features. Intuitively, $\mathbf{x}_i$ and $\mathbf{x}_j$ are strongly dependent if $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) = W_{ij}$ is larger than λ. Therefore, an edge is added between features $\mathbf{x}_i$ and $\mathbf{x}_j$ if they are strongly dependent. An edge between features represents (potential) redundancy. Now, $\mathcal{G}_F$ is used to select a subset of important features from layer τ. Features are selected as follows: First, the feature graph $\mathcal{G}_F$ is partitioned into groups of features $\{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ where each set $\mathcal{C}_k \in \mathcal{C}$ represents features that are dependent (though not necessarily pairwise dependent). To partition the feature graph $\mathcal{G}_F$, Algorithm 3 uses connected components, though other methods are also possible, e.g., a clustering or community detection method. Next, one or more representative features are selected from each group (cluster) of dependent features. Alternatively, it is also possible to derive a new feature from the group of dependent features, e.g., finding a low-dimensional embedding of these features or taking the principal eigenvector. In Algorithm 3 the earliest feature in each connected component $\mathcal{C}_k = \{\ldots, f_i, \ldots, f_j, \ldots\} \in \mathcal{C}$ is selected and all others are removed. After pruning the feature layer $\mathcal{F}_\tau$, the discarded features are removed from $\mathbf{X}$ and DeepGL updates the set of features learned thus far by setting $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_\tau$ (Algorithm 1: Line 8). Next, Line 9 increments τ and sets $\mathcal{F}_\tau \leftarrow \varnothing$. Finally, we check for convergence, and if the stopping criterion is not satisfied, then DeepGL learns an additional feature layer (Line 4-10).

An important aspect of DeepGL is the specific *convergence criterion* used to decide when to stop learning. In Algorithm 1, DeepGL terminates when either of the following conditions are satisfied: (i) no new features emerge (in the current feature layer τ and thus $|\mathcal{F}_t| = 0$), or (ii) the maximum number of layers is reached. However, DeepGL is not tied to any particular convergence criterion and others can easily be used in its place.

Algorithm 1. The **DeepGL Framework** for Learning Deep Graph Representations (node/edge features) from Large Graphs where the Features are Expressed as Relational Functions that Naturally Transfer Across-Networks

Require:
a (un)directed graph $G = (V, E)$; a set of relational feature operators $\Phi = \{\Phi_1, \ldots, \Phi_K\}$, and a feature similarity threshold λ.

1: $\mathcal{F}_1 \leftarrow \varnothing$ and initialize $\mathbf{X}$ if not given as input
2: Given G, construct base graph features (see Section 2.1 for further details) and concatenate the feature vectors to $\mathbf{X}$ and add the function definitions to $\mathcal{F}_1$; and set $\mathcal{F} \leftarrow \mathcal{F}_1$.
3: Transform *base feature vectors*; Set $\tau \leftarrow 2$
4: **repeat** ▷ feature layers $\mathcal{F}_\tau$ for $\tau = 2, \ldots, \mathrm{T}$
5: Search the space of features defined by applying relational feature operators $\Phi = \{\Phi_1, \ldots, \Phi_K\}$ to features $\big[\ \cdots\ \mathbf{x}_i\ \mathbf{x}_{i+1}\ \cdots\ \big]$ given as output in the previous layer $\mathcal{F}_{\tau-1}$ (via Algorithm 2). Add feature vectors to $\mathbf{X}$ and functions/def. to $\mathcal{F}_\tau$.
6: Transform feature vectors of layer $\mathcal{F}_\tau$
7: Evaluate the features (functions) in layer $\mathcal{F}_\tau$, e.g., using a criterion $\mathbb{K}$ to score feature pairs along with a feature selection method to select a subset (Algorithm 3) *or* by finding a low-rank embedding of the feature matrix and concatenating the embeddings to $\mathbf{X}$.
8: Discard features from $\mathbf{X}$ that were pruned (not in $\mathcal{F}_\tau$) and set $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_\tau$
9: Set $\tau \leftarrow \tau + 1$ and initialize $\mathcal{F}_\tau$ to $\varnothing$ for next feature layer
10: **until** no new features emerge **or** the max number of layers (*depth*) is reached
11: **return** $\mathbf{X}$ and the set of relational functions (definitions) $\mathcal{F}$

In contrast to node embedding methods that output only a *node* feature matrix $\mathbf{X}$, DeepGL also outputs the (hierarchical) relational functions (definitions) $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$ where each $f_i \in \mathcal{F}_h$ is a learned relational function of depth h for the ith feature vector $\mathbf{x}_i$. Maintaining the relational functions are important for transferring the features to another arbitrary graph of interest, but also for interpreting them. Moreover, DeepGL is an inductive representation learning approach as the relational functions can be used to derive embeddings for new nodes or even graphs.

There are many methods to evaluate and remove redundant/noisy features at each layer and DeepGL is not tied to any particular approach. For the experiments, we use the relational function evaluation and pruning routine in Algorithm 3 as it is computationally efficient and the features remain interpretable. There are two main classes of techniques for evaluating and removing redundant/noisy features at each layer. The first class of techniques use a criterion $\mathbb{K}$ to score the feature pairs along with any feature selection method (e.g., see Algorithm 3) to select a subset of representative and non-redundant features. The second class of techniques compute a low-rank embedding of the feature matrix at each layer and concatenates the embeddings as features for learning the next layer. Alternatively,

one can also use a hybrid approach that combines the advantages of both by simply concatenating the features from each. The above approaches are applied at each feature layer (iteration). Notice that these methods all have the same general objective of reducing noise and removing redundant features (minimality condition).

While previous work learns node embeddings (features), DeepGL instead learns complex relational functions that represent compositions of relational operators. Hence, these relational functions naturally generalize across graphs for inductive learning tasks. Both the relational operators and base graph features (which can be thought of as functions themselves) are independent of the particular graph topology G (i.e., they can be computed on any graph G) and therefore any arbitrary composition of relational operators applied to any base feature can be computed on any graph G.

2.4 Feature Diffusion

We introduce the notion of feature diffusion where the feature matrix at each layer can be smoothed using an arbitrary feature diffusion process. As an example, suppose $\mathbf{X}$ is the resulting feature matrix from layer τ, then we can set $\bar{\mathbf{X}}^{(0)} \leftarrow \mathbf{X}$ and solve

$$\bar{\mathbf{X}}^{(t)} = \mathbf{D}^{-1}\mathbf{A}\bar{\mathbf{X}}^{(t-1)}, \tag{4}$$

where $\mathbf{D}$ is the diagonal degree matrix and $\mathbf{A}$ is the adjacency matrix of G. The diffusion process above is repeated for a fixed number of iterations $t = 1, 2, \ldots, T$ or until convergence; and $\bar{\mathbf{X}}^{(t)} = \mathbf{D}^{-1}\mathbf{A}\bar{\mathbf{X}}^{(t-1)}$ corresponds to a simple feature propagation. More complex feature diffusion processes can also be used in DeepGL such as the normalized Laplacian feature diffusion defined as

$$\bar{\mathbf{X}}^{(t)} = (1-\theta)\mathbf{L}\bar{\mathbf{X}}^{(t-1)} + \theta\mathbf{X}, \quad \text{for} t = 1, 2, \ldots, \tag{5}$$

where $\mathbf{L}$ is the normalized Laplacian:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}. \tag{6}$$

The resulting diffused feature vectors $\bar{\mathbf{X}} = \begin{bmatrix} \bar{\mathbf{x}}_1 & \bar{\mathbf{x}}_2 & \cdots \end{bmatrix}$ are effectively smoothed by the features of related graph elements (nodes/edges) governed by the particular diffusion process. Notice that feature vectors given as output at each layer can be diffused (e.g., after Line 5 or 8 of Algorithm 1). Note $\bar{\mathbf{X}}$ can be leveraged in a variety of ways: $\mathbf{X} \leftarrow \bar{\mathbf{X}}$ (replacing previous) or concatenated by $\mathbf{X} \leftarrow \begin{bmatrix} \mathbf{X} & \bar{\mathbf{X}} \end{bmatrix}$. Feature diffusion can be viewed as a form of graph regularization as it can improve the generalizability of a model learned using the graph embedding.

2.5 Supervised Representation Learning

The DeepGL framework naturally generalizes for *supervised representation learning* by replacing the feature evaluation routine (called in Algorithm 1 Line 7) with an appropriate objective function, e.g., one that seeks to find a set of features that (i) maximize relevancy (predictive quality) with respect to $\mathbf{y}$ (i.e., observed class labels) while (ii) minimizing redundancy among the features in that set. The objective function capturing both (i) and (ii) is:

$$\mathbf{x} = \arg\max_{\mathbf{x}_i \notin \mathcal{X}} \left\{ \mathbb{K}\big(\mathbf{y}, \mathbf{x}_i\big) - \beta \sum_{\mathbf{x}_j \in \mathcal{X}} \mathbb{K}\big(\mathbf{x}_i, \mathbf{x}_j\big) \right\}, \tag{7}$$

where $\mathbb{K}$ is a measure such as mutual information; $\mathcal{X}$ is the current set of selected features; and β is a hyperparameter that determines the balance between maximizing relevance *and* minimizing redundancy. The first term in Eq. (7) seeks to find $\mathbf{x}_i$ that maximizes the relevancy of $\mathbf{x}_i$ to $\mathbf{y}$ whereas the second term attempts to minimize the redundancy between $\mathbf{x}_i$ and each $\mathbf{x}_j \in \mathcal{X}$ of the already selected features. Initially, we set $\mathcal{X} \leftarrow \{\mathbf{x}'\}$ where $\mathbf{x}' = \text{argmax}_{\mathbf{x}_i} \mathbb{K}\big(\mathbf{y}, \mathbf{x}_i\big)$. Afterwards, we solve Eq. (7) to find $\mathbf{x}_i$ (such that $\mathbf{x}_i \notin \mathcal{X}$) which is then added to $\mathcal{X}$ (and removed from the set of remaining features). This is repeated until the stopping criterion is reached.

Algorithm 2. Derive a Feature Layer Using the Features from the Previous Layer and the Set of Relational Feature Operators $\Phi = \{\Phi_1, \ldots, \Phi_K\}$

1: **procedure** FeatureLayer(G, $\mathbf{X}$, Φ, $\mathcal{F}$, $\mathcal{F}_\tau$, λ)
2: **parallel for each** graph element $g_i \in \mathcal{G}$ **do**
3: Set $t \leftarrow |\mathcal{F}|$
4: **for each** feature $\mathbf{x}_k$ *s.t.* $f_k \in \mathcal{F}_{\tau-1}$ in order **do**
5: **for each** $S \in \{\Gamma^+_\ell(g_i),\ \Gamma^-_\ell(g_i),\ \Gamma_\ell(g_i)\}$ **do**
6: **for each** relational operator $\Phi \in \Phi$ **do**
7: $X_{it} = \Phi(S, \mathbf{x}_k)$ and $t \leftarrow t + 1$
8: Add feature definitions to $\mathcal{F}_\tau$
9: **return** feature matrix $\mathbf{X}$ and $\mathcal{F}_\tau$

Algorithm 3. Score and Prune the Feature Layer

1: **procedure** EvaluateFeatureLayer(G, $\mathbf{X}$, $\mathcal{F}$, $\mathcal{F}_\tau$)
2: Let $\mathcal{G}_F = (V_F, E_F, \mathbf{W})$
3: **parallel for each** feature $f_i \in \mathcal{F}_\tau$ **do**
4: **for each** feature $f_j \in (\mathcal{F}_{\tau-1} \cup \cdots \cup \mathcal{F}_1)$ **do**
5: **if** $\mathbb{K}\big(\mathbf{x}_i, \mathbf{x}_j\big) > \lambda$ **then**
6: $E_F = E_F \cup \{(i, j)\}$
7: $W_{ij} = \mathbb{K}\big(\mathbf{x}_i, \mathbf{x}_j\big)$
8: Partition $\mathcal{G}_F$ using conn. components $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$
9: **parallel for each** $\mathcal{C}_k \in \mathcal{C}$ **do** ▷ Remove features
10: Find f_i s.t. $\forall f_j \in \mathcal{C}_k : i < j$.
11: Remove $\mathcal{C}_k$ from $\mathcal{F}_\tau$ and set $\mathcal{F}_\tau \leftarrow \mathcal{F}_\tau \cup \{f_i\}$

3 Analysis

Let $N = |V|$ denote the number of nodes, $M = |E|$ be the number of edges, $F =$ number of relational functions learned by DeepGL, and $K =$ number of relational feature operators.

3.1 Time Complexity

3.1.1 Learning

Lemma 3.1. *The time complexity for learning edge features (relational functions) using the* DeepGL *framework is:*

$$\mathcal{O}\big(F(M + MF)\big), \tag{8}$$

and the time complexity for learning node features using the DeepGL *framework is:*

$$\mathcal{O}\big(F(M + NF)\big), \tag{9}$$

Hence, the time complexity of both edge (Eq. 8) and *node (Eq. 9) feature learning in* DeepGL *is linear in the number of edges.*

Proof. The time complexity of each of the main steps is provided below. Recall that DeepGL is a general and flexible framework for inductive graph-based feature learning. The particular instantiation of DeepGL used in this analysis corresponds to using Algorithm 3 for feature evaluation and pruning where $\mathbb{K}$ = agreement scoring defined in Eq. (12) with logarithmic binning. We also assume the relational functions are composed of any relational feature operator (aggregator function) with a worst-case time complexity of $\mathcal{O}(|S|)$ where S is the set of related graph elements (e.g., $\ell = 1$ hop neighbors of a node or edge). Further, the related graph elements S given as input to a relational feature operator $\Phi \in \Phi = \{\Phi_1, \ldots, \Phi_K\}$ is the $\ell = 1$ hop neighborhood of a node in the worst case. Hence, if G is directed, then we consider the *worst case* where $S = \Gamma_\ell(g_i) = \Gamma^+_\ell(g_i) \cup \Gamma^-_\ell(g_i)$ since $|\Gamma_\ell(g_i)| \geq |\Gamma^+_\ell(g_i)|$ and $|\Gamma_\ell(g_i)| \geq |\Gamma^-_\ell(g_i)|$. It is straightforward and trivial to select a subset $J \subseteq S$ of related graph elements for a given node (or edge) using an arbitrary uniform or weighted distribution $\mathbb{P}$ and derive a feature value for that node (edge) using J. Moreover, the maximum size of J can be set by the user as done in [25], [28].□

Base Graph Features. Recall that all base features discussed in Section 2.1 can be computed in $\mathcal{O}(M)$ time (by design). While DeepGL is not tied to a particular set of base features and can use any arbitrary set of base features including those that are more computationally intensive, we nevertheless restrict our attention to base features that can be computed in $\mathcal{O}(M)$ to ensure that DeepGL is fast and efficient for massive networks. For deriving the graphlet (network motif) frequencies and egonet features (that are to 3-node motif variations), we use recent provably accurate estimation methods [25], [28]. As shown in [25], [28], we can achieve estimates within a guaranteed level of accuracy and time by setting a few simple parameters in the estimation algorithm. The time complexity to estimate the frequency of all 4-node graphlets is $\mathcal{O}(M\Delta_{ub})$ in the worst case where $\Delta_{ub} \ll M$ is a small constant set by the user that represents the maximum sampled degree [25], [28].

Searching the Space of Relational Functions. The time complexity to derive a novel candidate feature $\mathbf{x}$ using an arbitrary relational feature operator $\Phi \in \Phi$ takes at most $\mathcal{O}(M)$ time. For a feature $f_k \in \mathcal{F}$ with vector $\mathbf{x}_k$, DeepGL derives $K = |\Phi|$ new candidate features to search. This is repeated at most F times. Therefore, the worst-case time complexity to search the space of relational functions is $\mathcal{O}(KFM)$ where $K \ll M$. Since K is a small constant, it is disregarded giving $\mathcal{O}(FM)$.

Scoring and Pruning Relational Functions. First we score the feature pairs using a score function $\mathbb{K}$. To assign a score $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j)$ to an arbitrary pair of features $f_i, f_j \in \mathcal{F}$, it takes at most $\mathcal{O}(M)$ time (or $\mathcal{O}(N)$ time for node feature learning). Further, if $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) > \lambda$, then we set $W_{ij} = \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j)$ and add an edge $E_F = E_F \cup \{(i, j)\}$ in $o(1)$ constant time. Therefore, the worst-case time complexity to score all such feature pairs is $\mathcal{O}(F^2M)$ for edge feature learning where $F \ll M$ and $\mathcal{O}(F^2N)$ for nodes where $F \ll N$. The time complexity for pruning the feature graph is at most $F^2 + F$ and thus can be ignored since the term F^2M (or F^2N) dominate as M (or N) grows large.

3.1.2 Inductive Relational Functions

We now state the time complexity of directly computing the set of inductive relational functions $\mathcal{F}$ (e.g., that were previously learned on another arbitrary graph). The set of relational functions $\mathcal{F}$ can be used in two important ways. First, the relational functions are useful for inductive *across-network transfer learning tasks* where one uses the relational functions that were previously learned from a graph G_i and wants to extract them on another graph G_j of interest (e.g., for graph matching or similarity, across-network classification). Second, given new nodes or edges in the *same* graph that the relational functions were learned, we can use $\mathcal{F}$ to derive node or edge feature values (embeddings, encodings) for the new nodes/edges without having to relearn the relational functions each time a new node or edge appears in the graph.

Lemma 3.2. *Given a set of learned edge (node) relational functions $\mathcal{F}$ (from Section 3.1.1), the time complexity for directly deriving (extracting) the edge (or node) relational functions is:*

$$\mathcal{O}(FM). \tag{10}$$

Hence, the time complexity is linear in the number of edges.

Computing the set of inductive relational functions on another arbitrary graph obviously requires less work than learning the actual set of inductive relational functions (Section 3.1.1) since there is no learning involved as we simply derive the previously learned relational functions $\mathcal{F}$ directly from their definitions. Therefore, we avoid Algorithm 3 completely since we do not need to score or prune any candidate features from the potential space of relational functions. The time complexity is therefore $\mathcal{O}(FM)$ where $F = |\mathcal{F}|$ as shown previously in Section 3.1.1.

3.2 Space Complexity

Lemma 3.3 *The space complexity of the learned (sparse) feature matrix* $\mathbf{X}$ *given as output by* DeepGL *is*

$$\mathcal{O}(F(\lceil \alpha N \rceil)). \tag{11}$$

Proof. The space complexity stated in Eq. (11) assumes logarithmic binning is used to encode the feature values of each feature vector $\mathbf{x} \in \mathbb{R}^N$. Notice that without any compression, the space complexity is obviously $\mathcal{O}(NF)$. However, since log binning is used with a bin width of α, we can avoid storing approximately $\lceil \alpha N \rceil$ values for each N-dimensional feature by mapping these values to 0 and explicitly storing only the remaining nonzero feature values. This is accomplished using a sparse feature vector representation. Furthermore, we can store these even more efficiently by leveraging the fact that the same nodes whom appear within the αN largest (or smallest) feature values for a particular feature often appear within the $\lceil \alpha N \rceil$ largest (or smallest) feature values for other arbitrary features as well. This is likely due to the power-law observed in many real-world graphs. Similarly, the space complexity of the sparse edge feature matrix $\mathbf{X}$ is $\mathcal{O}(F(\lceil \alpha M \rceil))$. □

4 Experiments

This section demonstrates the effectiveness of the proposed framework. In particular, we investigate the predictive performance of DeepGL compared to the state-of-the-art

methods across a wide variety of graph-based learning tasks as well as its scalability, runtime, and parallel performance.

4.1 Experimental Settings

In these experiments, we use the following instantiation of DeepGL: Features are transformed using logarithmic binning and evaluated using a simple agreement score function where $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) =$ fraction of graph elements that agree. More formally, agreement scoring is defined as:

$$\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\left|\left\{(x_{ik}, x_{jk}),\ \forall k = 1, \ldots, N \mid x_{ik} = x_{jk}\right\}\right|}{N}, \quad (12)$$

where x_{ik} and x_{jk} are the kth feature value of the N -dimensional vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. As an aside, recall that a key advantage of DeepGL is that the framework has many interchangeable components including the above evaluation criterion, base features (Section 2.1), relational operators (Section 2.2), among others. The expressiveness and flexibility of DeepGL makes it well-suited for a variety of application domains, graph types, and learning scenarios [29]. In addition, DeepGL can even leverage features from an existing method (or any approach proposed in the future) to discover a more discriminative set of features. Unless otherwise mentioned, we use the base graph features mentioned in Section 2.1; set $\alpha = 0.5$ (bin size of logarithmic binning) and perform a grid search over $\lambda \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$ and $\Phi \in \big\{\Phi_{\text{mean}}, \Phi_{\text{sum}}, \Phi_{\text{prod}}, \{\Phi_{\text{mean}}, \Phi_{\text{sum}}\}, \{\Phi_{\text{prod}}, \Phi_{\text{sum}}\}, \{\Phi_{\text{prod}}, \Phi_{\text{mean}}\} \{\Phi_{\text{sum}}, \Phi_{\text{mean}}, \Phi_{\text{prod}}\}\big\}$. See Table 2. Note Φ_{prod} refers to the Hadamard relational operator defined formally in Table 2. As an aside, DeepGL has fewer hyperparameters than node2vec, DeepWalk, and LINE used in the comparison below. The specific model defined by the above instantiation of DeepGL is selected using 10-fold cross-validation on 10 percent of the labeled data. Experiments are repeated for 10 random seed initializations. All results are statistically significant with p-value < 0.01.

We evaluate the proposed framework against node2-vec [15], DeepWalk [13], and LINE [16]. For node2vec, we use the hyperparameters and grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [15]. The experimental setup mentioned in [15] is used for DeepWalk and LINE. Unless otherwise mentioned, we use logistic regression with an L2 penalty and one-vs-rest for multiclass problems. Data has been made available at NetworkRepository [30].[10]

4.2 Within-Network Link Classification

We first evaluate the effectiveness of DeepGL for link classification. To be able to compare DeepGL to node2vec and the other methods, we focus in this section on *within-network link classification*. For comparison, we use the same set of binary operators to construct features for the edges *indirectly* using the learned node representations: Given the feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ for node i and j, $(\mathbf{x}_i + \mathbf{x}_j)/2$ is the MEAN; $\mathbf{x}_i \odot \mathbf{x}_j$ is the (Hadamard) PRODUCT; $\left|\mathbf{x}_i - \mathbf{x}_j\right|$ and $(\mathbf{x}_i - \mathbf{x}_j)^{\circ 2}$ is the WEIGHTED-L_1 and WEIGHTED-L_2 binary operators, respectively.[11] Note that these binary operators (used to create edge features) are not to be confused with the relational feature operators defined in Table 2. In Table 3, we observe that DeepGL outperforms node2vec, DeepWalk, and LINE with an average gain between 18.09 and 20.80 percent across all graphs and binary operators.

TABLE 3
AUC Scores for *Within-Network Link Classification*

		escorts	yahoo-msg
MEAN $(\mathbf{x}_i + \mathbf{x}_j)/2$	**DeepGL**	**0.6891**	**0.9410**
	node2vec	0.6426	0.9397
	DeepWalk	0.6308	0.9317
	LINE	0.6550	0.7967
PRODUCT $\mathbf{x}_i \odot \mathbf{x}_j$	**DeepGL**	0.6339	0.9324
	node2vec	0.5445	0.8633
	DeepWalk	0.5366	0.8522
	LINE	0.5735	0.7384
WEIGHTED L_1 $\left|\mathbf{x}_i - \mathbf{x}_j\right|$	**DeepGL**	0.6857	0.9247
	node2vec	0.5050	0.7644
	DeepWalk	0.5040	0.7609
	LINE	0.6443	0.7492
WEIGHTED L_2 $(\mathbf{x}_i - \mathbf{x}_j)^{\circ 2}$	**DeepGL**	0.6817	0.9160
	node2vec	0.4950	0.7623
	DeepWalk	0.4936	0.7529
	LINE	0.6466	0.5346

Notice that node2vec, DeepWalk, and LINE all require that the training graph contain at least one edge among each node in G. However, DeepGL overcomes this fundamental limitation and can actually predict the class label of edges that are not in the training graph as well as the class labels of edges in an entirely different network.

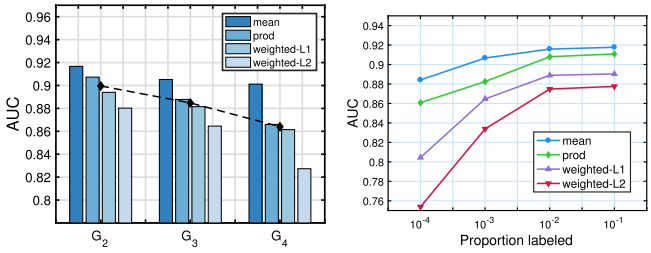
4.3 Inductive Across-Network Transfer Learning

Recall from Section 2 that a key advantage of DeepGL over existing methods [13], [15], [16] lies in its ability to learn features that naturally generalize for *inductive* across-network transfer learning tasks. Unlike existing methods [13], [15], [16], DeepGL learns relational functions that generalize for extraction on another arbitrary graph and therefore can be used for graph-based transfer (inductive) learning tasks such as across-network link classification. In contrast, node2vec [15], DeepWalk [13] and LINE [16] are unable to be used for such graph-based transfer learning tasks as the features from these methods are fundamentally tied to node identity, as opposed to the general relational functions learned by DeepGL that can be computed on any arbitrary graph. Further, these methods require the training graph to be connected (e.g., see [15]), which implies that each node in the original graph has at least one edge in the training graph. This assumption is unrealistic, though is required by these approaches, otherwise, they would be unable to construct edge features for any edges containing nodes which did not appear in the test set.

For each experiment, the training graph is fully observed with all known labels available for learning. The test graph is completely unlabeled and each classification model is evaluated on its ability to predict *all* available labels in the test graph. Given the training graph $G = (V, E)$, we use DeepGL to learn the feature matrix $\mathbf{X}$ and the relational functions $\mathcal{F}$ (definitions). The relational functions $\mathcal{F}$ are then used to extract the same identical features on an

10. See http://networkrepository.comfor data description & statistics

11. Note $\mathbf{x}^{\circ 2}$ is the element-wise Hadamard power; $\mathbf{x}_i \odot \mathbf{x}_j$ is the element-wise product.



(a) Across-network (b) Varying proportion labeled

Fig. 5. Effectiveness of the DeepGL framework for across network transfer learning. (a) AUC scores for across-network link classification using yahoo-msg. Note ◆ denotes the mean AUC of each test graph. (b) Effectiveness of DeepGL for classification with very small amounts of training labels.

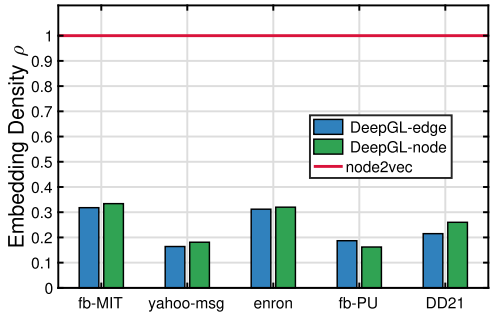


Fig. 6. DeepGL requires up to 6x less space than node2vec and other existing methods that learn dense node embeddings.

arbitrary *test graph* $G' = (V', E')$ giving as output a feature matrix $\mathbf{X}'$. Notice that each node (or edge) is embedded in the same F-dimensional space, even despite that the set of nodes/edges between the graphs could be completely disjoint, and even from different domains. Thus, an identical set of features is used for all train and test graphs.

In these experiments, the training graph G_1 represents the first week of data from yahoo-msg,[12] whereas the test graphs $\{G_2, G_3, G_4\}$ represent the next three weeks of data (e.g., G_2 contains edges that occur only within week 2, and so on). Hence, the test graphs contain many nodes and edges not present in the training graph. As such, the predictive performance is expected to decrease significantly over time as the features become increasingly stale due to the constant changes in the graph structure with the addition and deletion of nodes and edges. However, we observe the performance of DeepGL for across-network link classification to be stable with only a small decrease in AUC as a function of time as shown in Fig. 5a. This is especially true for edge features constructed using mean. As an aside, the mean operator gives best performance on average across all test graphs; with an average AUC of 0.907 over all graphs.

Now we investigate the performance as a function of the amount of labeled data used. In Fig. 5b, we observe that DeepGL performs well with very small amounts of labeled data for training. Strikingly, the difference in AUC scores from models learned using 1 percent of the labeled data is insignificant at $p < 0.01$ *w.r.t.* models learned using larger quantities.

4.4 Analysis of Space-Efficiency

Learning sparse space-efficient node and edge feature representations is of vital importance for large networks where storing even a modest number of *dense* features is impractical (especially when stored in-memory). Despite the importance of learning a sparse space-efficient representation, existing work has been limited to discovering completely dense (node) features [13], [15], [16]. To understand the effectiveness of the proposed framework for learning sparse graph representations, we measure the density of each representation learned from DeepGL and compare these against the state-of-the-art methods [13], [15]. We focus first on node representations since existing methods are limited to only node features. Results are shown in Fig. 6. In all cases, the node representations learned by DeepGL are extremely sparse and significantly more space-efficient than node2vec [15] as observed in Fig. 6. DeepWalk and LINE use nearly the same space as node2vec, and thus are omitted for brevity. Strikingly, DeepGL uses only a fraction of the space required by existing methods (Fig. 6). Moreover, the density of node and edge representations from DeepGL is between $[0.162,\ \ 0.334]$ for nodes and $[0.164,\ \ 0.318]$ for edges and up to $6\times$ more space-efficient than existing methods.

Notably, recent node embedding methods not only output dense node features, but are also real-valued and often negative (e.g., [13], [15], [16]). Thus, they require 8 bytes per feature-value, whereas DeepGL requires only 2 bytes and can sometimes be reduced to even 1 byte if needed by adjusting α (i.e., the bin size of the log binning transformation). To understand the impact of this, assume both approaches learn a node representation with 128 dimensions (features) for a graph with 10,000,000 nodes. In this case, node2vec, DeepWalk, and LINE require 10.2 GB, whereas DeepGL uses only 0.768 GB (assuming a modest 0.3 density) — a significant reduction in space by a factor of 13.

4.5 Runtime & Scalability

To evaluate the performance and scalability of the proposed framework, we learn node representations for Erdös-Rényi graphs of increasing size (from 100 to 10,000,000 nodes) such that each graph has an average degree of 10. We compare the performance of DeepGL against LINE [16] and node2vec [15] which is designed specifically to be *scalable* for large graphs. Default parameters are used for each method. In Fig. 7a, we observe that DeepGL is significantly faster and more scalable than node2vec and LINE. In particular, node2vec takes 1.8 days (45.3 hours) for 10 million nodes, whereas DeepGL finishes in only 15 minutes; see Fig. 7a. Strikingly, this is 182 times faster than node2vec and 106 times faster than LINE. In Fig. 7b, we observe that DeepGL spends the majority of time in the search and optimization phase. In Fig. 8, we also investigate effect on the number of relational functions learned, sparsity, and runtime of DeepGL as α and λ vary.

4.6 Parallel Scaling

This section investigates the parallel performance of DeepGL. To evaluate the effectiveness of the parallel algorithm we measure speedup defined as $S_p = \frac{T_1}{T_p}$ where T_1 and T_p are the execution time of the sequential and parallel algorithms (w/ p processing units), respectively. In Fig. 9, we observe strong parallel scaling for all DeepGL variants with the edge representation learning variants performing slightly better than the node representation learning methods from DeepGL. Results are reported for `soc--gowalla`

12. https://webscope.sandbox.yahoo.com/

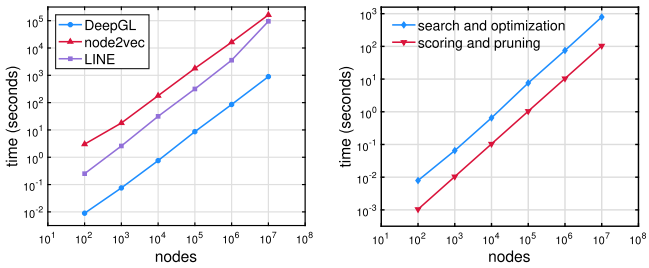


Fig. 7. Runtime comparison on Erdös-Rényi graphs with an average degree of 10. Left: The proposed approach is shown to be orders of magnitude faster than node2vec [15] and LINE [16]. Right: Runtime of the main DeepGL phases.

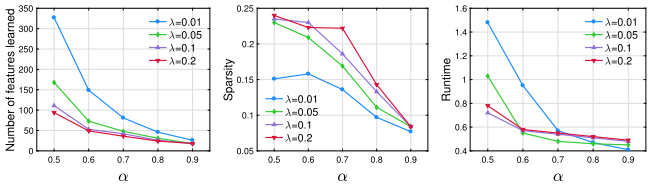


Fig. 8. Varying α and λ. Impact on the number of relational functions learned, sparsity, and runtime of DeepGL as α and λ vary.

on a machine with 4 Intel Xeon E5-4627 v2 3.3 GHz CPUs. Other graphs and machines gave similar results.

4.7 Effectiveness on Link Prediction

Given a graph G with a fraction of missing edges, the link prediction task is to predict these missing edges. We generate a labeled dataset of edges as done in [15]. Positive examples are obtained by removing 50 percent of edges randomly, whereas *negative examples* are generated by randomly sampling an equal number of node pairs that are not connected with an edge, i.e., each node pair $(i, j) \notin E$. For each method, we learn features using the remaining graph that consists of only positive examples. To construct edge features from the node embeddings, we use the mean operator defined as $(\mathbf{x}_i + \mathbf{x}_j)/2$. Using the feature representations from each method, we then learn a model to predict whether a given edge in the test set exists in E or not. Notice that node embedding methods such as node2vec require that each node in G appear in at least one edge in the training graph (i.e., the graph remains connected), otherwise these methods are unable to derive features for such nodes. This is a significant limitation especially in practice where nodes (e.g., representing users) may be added in the future. Furthermore, most graphs contain many nodes with only a few edges (due to the small world phenomenon) which are also the most difficult to predict, yet are avoided in the evaluation of node embedding methods due to the above restriction. Results are provided in Table 4. We report both AUC and F1 scores. In all cases, DeepGL achieves better predictive performance over the other methods across a wide variety of graphs from different domains with fundamentally different structural characteristics. In Fig. 10, we summarize the gain in AUC and F1 score of DeepGL over the baseline methods. Strikingly, DeepGL achieves an overall gain in AUC of 32 percent and an overall gain in F1 of 37 percent averaged over all baseline methods and across a wide variety of graphs with different structural characteristics. As an aside, common neighbors and other simple measures are known to perform significantly worse than node2vec and other embedding approaches (see [15]) and thus are not included here for brevity.

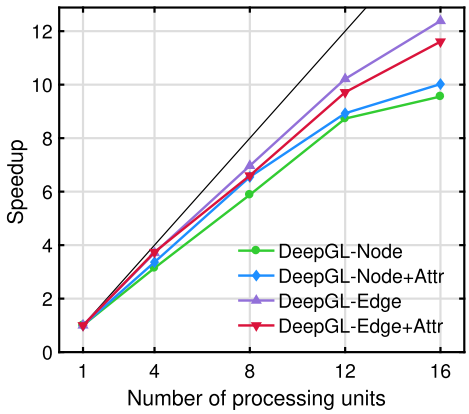


Fig. 9. Parallel speedup of different variants from the DeepGL framework.

4.8 Sensitivity & Perturbation Analysis

Many real-world networks are partially observed and noisy (e.g., due to limitations in data collection) and often have both *missing* and *noisy/spurious edges*. In this section, we analyze the sensitivity of DeepGL under such conditions using the DD242 network. Results are shown in Fig. 11. In particular, DeepGL is shown to be robust to missing and noisy edges while the number of features (dimensionality) learned by DeepGL remains relatively stable with only a slight increase as a function of the number of missing or additional edges. While one must define the number of dimensions (features) in existing node embedding methods, DeepGL automatically learns the appropriate number of dimensions and the depth (number of layers). Furthermore, we find the number of features learned by DeepGL is roughly correlated with the complexity and randomness of the graph (i.e., as the graph becomes more random, the number of features increases to account for such randomness).

4.9 Interpretability of Learned Features

The features (embeddings, representations) learned by most existing approaches are notoriously difficult to interpret and explain which is becoming increasingly important in practice [17], [18]. In contrast, the features learned by DeepGL are relatively more *interpretable* and can be explained by examining the actual relational functions learned. While existing work primarily outputs the feature/embedding matrix $\mathbf{X}$, DeepGL also outputs the relational functions $\mathcal{F}$ that correspond to each of the learned features. It is these relational functions that allow us to gain insight into the meaning of the learned features.

In Table 5, we show a few of the relational functions learned from an email network (ia-email-EU; nodes represent users and directed edges indicate an email communication from one user to another). Recall a feature in DeepGL is a relational function representing a composition of relational feature operators applied over a base feature. Therefore, the *interpretableness* of a learned feature in DeepGL depends on two aspects. First, we need to understand the base feature, e.g., in-degree.[13] Second, we need to understand the relational feature operators that are used in the relational function. As an example, $(\Phi^{-}_{mean})(\mathbf{x})$ in Table 5 where $\mathbf{x} = \mathbf{d}^{-}$ (in-degree) is interpreted as "the mean in-degree of the in-neighbors of a node." Using the semantics of the graph,

13. Notice that base features derived from the graph G are also functions (e.g., degree is a function that sums the adjacent nodes).

TABLE 4
Link Prediction Results for a Wide Range of Graphs from Different Domains

			AUC				F1			
graph	$\|V\|$	$\|E\|$	**DeepGL**	**N2V**	**DW**	**LINE**	**DeepGL**	**N2V**	**DW**	**LINE**
fb-Stanford3	11.5K	568K	**0.828**	0.540	0.531	0.603	**0.757**	0.529	0.519	0.630
fb-MIT	6.4K	251.2K	**0.817**	0.551	0.550	0.675	**0.740**	0.535	0.532	0.598
fb-Duke14	9.8K	506.4K	**0.794**	0.541	0.530	0.679	**0.725**	0.528	0.518	0.608
tech-WHOIS	7.4K	56.9K	**0.946**	0.689	0.654	0.622	**0.880**	0.638	0.610	0.626
web-google	1.2K	2.7K	**0.837**	0.606	0.595	0.647	**0.777**	0.540	0.528	0.505
web-indochina	11.3K	47.6K	**0.786**	0.571	0.550	0.676	**0.720**	0.546	0.531	0.646
bio-dmela	7.3K	25.6K	**0.871**	0.591	0.589	0.523	**0.793**	0.545	0.544	0.534
bio-celegans	453	2.0K	**0.877**	0.804	0.801	0.689	**0.834**	0.754	0.746	0.649
ia-emailEU	32.4K	54.3K	**0.992**	0.774	0.697	0.677	**0.965**	0.603	0.564	0.546
ia-email-dnc	1K	12K	**0.988**	0.662	0.654	0.800	**0.955**	0.621	0.599	0.828
ia-fbmessages	1.2K	6.4K	**0.852**	0.641	0.637	0.686	**0.775**	0.604	0.590	0.635
ia-infectdublin	410	2.7K	**0.666**	0.544	0.534	0.536	**0.633**	0.533	0.515	0.505
ca-CondMat	21.3K	91.2K	**0.771**	0.538	0.525	0.556	**0.710**	0.522	0.513	0.543
ca-cora	2.7K	5.4K	**0.697**	0.618	0.601	0.611	**0.651**	0.540	0.520	0.539
ca-citeseer	227K	814K	**0.785**	0.704	0.683	0.643	**0.724**	0.653	0.627	0.572
ca-Erdos	5.1K	7.5K	**0.961**	0.694	0.670	0.768	**0.908**	0.564	0.524	0.633
soc-wiki-elec	7.1K	107K	**0.956**	0.688	0.664	0.797	**0.884**	0.642	0.618	0.726
soc-gplus	23.6K	39.2K	**0.998**	0.760	0.753	0.780	**0.992**	0.634	0.622	0.731
soc-BlogCatalog	88.8K	2.1M	**0.950**	0.918	0.907	0.873	**0.882**	0.680	0.676	0.648
road-chicago-reg	1.5K	1.3K	**0.962**	0.830	0.766	0.727	**0.772**	0.551	0.532	0.536
econ-wm2	259	2.9K	**0.904**	0.864	0.840	0.832	**0.833**	0.771	0.727	0.719

* *N2V = node2vec, DW = DeepWalk*

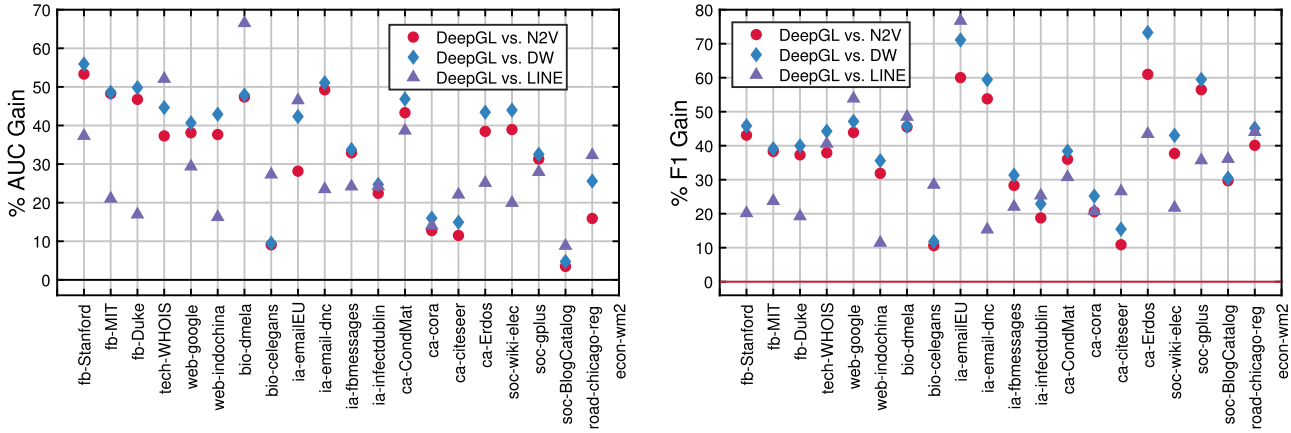


Fig. 10. DeepGL is effective for link prediction with significant gain in predictive performance. In particular, DeepGL achieves an overall gain in AUC of 32 percent and an overall gain in F1 of 37 percent averaged over all baseline methods and graphs. Note N2V = node2vec, DW = DeepWalk.

it can be interpreted further as "the mean number of emails received by individuals that have sent emails to a given individual." In other words, the mean number of emails received by users that have sent emails to a given node. This relational function captures whether individuals that send a given user emails also receive a lot of emails or not. It is straightforward to interpret the other relational functions in a similar fashion and due to space we leave this up to the reader. DeepGL also learns relational functions involving either frequent (3-stars, 4-paths) or rare (triangles) induced subgraphs as these are highly discriminative as they characterize the behavioral roles of nodes in the graph. As shown in Table 5, composing relational operators allows DeepGL to learn structured and interpretable relational functions from well understood base components. The learned relational functions yield decompositions of a signal into interpretable components that facilitate model checking by domain experts.

4.10 Visualization

We now explore the properties of the graph that are captured by the feature matrix $\mathbf{X}$ from DeepGL and node2vec. In particular, we use k-means to group nodes that are similar with respect to the node feature vectors given as output by each method. The number of clusters k is selected automatically using MDL. In Fig. 12, we visualize the graph structure and color the nodes (and edges in the case of DeepGL) by their cluster assignments. Strikingly, we find in Fig. 12a that DeepGL captures the node and edge *roles* [7]

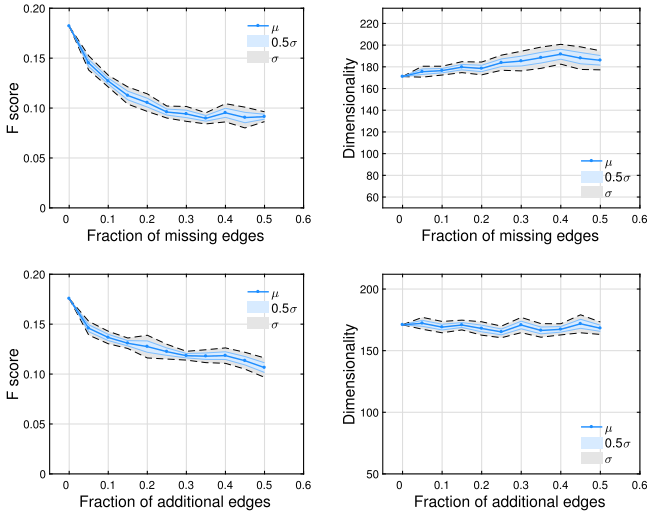


Fig. 11. DeepGL is robust to missing and noisy edges.

TABLE 5
Relational Functions Learned by DeepGL are *Interpretable*

relational function f	base feature $\mathbf{x}$
$(\Phi^{-}_{mean})(\mathbf{x})$	in-degree
$(\Phi^{-}_{sum} \circ \Phi^{-}_{mean})(\mathbf{x})$	frequency of triangles
$(\Phi^{-}_{prod} \circ \Phi^{+}_{mean})(\mathbf{x})$	frequency of induced 4-node paths
$(\Phi^{+}_{sum} \circ \Phi^{-}_{sum} \circ \Phi^{-}_{mean})(\mathbf{x})$	frequency of incoming edges to egonet
$(\Phi^{+}_{sum} \circ \Phi^{-}_{sum} \circ \Phi^{-}_{sum})(\mathbf{x})$	frequency of induced 3-node stars
$(\Phi^{-}_{sum} \circ \Phi^{-}_{sum} \circ \Phi^{-}_{prod} \circ \Phi^{+}_{mean})(\mathbf{x})$	out-degree

Examples of a few important relational functions learned from the ia-email-EU network are shown below. Recall from Section 2 that a relational function (feature) of order-h is composed of h relational feature operators and each of the h relational feature operators takes a set S of related graph elements (Table 2). In this work, $S \in \{\Gamma^{+}_{\ell}(g_i), \Gamma^{-}_{\ell}(g_i), \Gamma_{\ell}(g_i)\}$ where $\ell = 1$. For simplicity, we denote Φ^{+}, Φ^{-}, Φ as the relational feature operators that use out, in, and total (both in/out) 1-hop neighbors, respectively. See text for discussion.

that represent the important structural behaviors of the nodes and edges in the graph. In contrast, node2vec captures the community of a node as observed in Fig. 12b. Notably, the roles given by DeepGL are intuitive and make sense (Fig. 12a). For example, the red role represents authors (and co-author links) from the CS PhD co-authorship graph [30] that are gate-keepers (bridge roles) connecting different groups of authors. Furthermore, the green role represents nodes at the peripheral (or edge) of the network and are star edge nodes (as opposed to nodes at the center of a large star/hub nodes). This demonstrates the effectiveness of DeepGL for capturing and revealing the important structural properties (structural roles [7]) of the nodes and edges.

5 Related Work

Related research is categorized below.

Node Embedding Methods. There has been a lot of interest recently in learning a set of useful *node features*from large-scale networks automatically [13], [15], [16], [31]. Many of the techniques initially proposed for solving the node embedding problem were based on graph factorization [32], [33], [34]. Recently, the skip-gram model [12] was introduced in NLP to learn vector representations for words. Inspired by the success of the skip-gram model, various methods [13], [15], [16], [35], [36] have been proposed to learn node embeddings using skip-gram by sampling

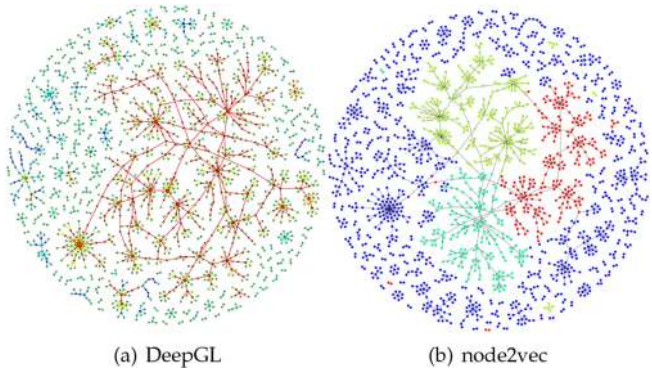


Fig. 12. Left: Application of DeepGL for edge and node role discovery (ca-PhD). Link color represents the edge role and node color indicates the corresponding node role. Right: However, since node2vec uses random walks it captures communities [39] as shown in (b) where the color depicts the community of a node. See text for discussion.

ordered sequences of node ids (random walks) from the graph and applying the skip-gram model to these sequences to learn node embeddings. However, these methods are not inductive nor are they able to learn sparse space-efficient node embeddings as achieved by DeepGL. More recently, Chen et al. [37] proposed a hierarchical approach to network representation learning called HARP whereas Ma et al.proposed MINES for multi-dimensional network embedding with hierarchical structure. Other work has focused specifically on community-based embeddings [13], [15], [36], [38] as opposed to role-based embeddings [39]. In contrast, DeepGL learns features that capture the notion of roles (i.e., structural similarty) defined in [7] as opposed to communities (Fig. 12).

Heterogeneous networks [40] have also been recently considered [41], [42], [43], [44] as well as attributed networks [45], [46], [47]. Huang et al. [45] proposed an approach for attributed networks with labels whereas Yang et al. [48] used text features to learn node representations. Liang et al. [49] proposed a semi-supervised approach for networks with outliers. Bojchevski et al. [50] proposed an unsupervised rank-based approach. Coley et al. [51] introduced a convolutional approach for attributed molecular graphs that learns graph embeddings as opposed to node embeddings. Similarly, Lee et al. [52] proposed a graph attention model that embeds graphs as opposed to nodes for graph classification. There has also been some recent work on learning node embedding in dynamic networks [33], [53], [54], [55], semi-supervised network embeddings [56], [57] and methods for improving the learned representations [36], [47], [58], [59], [60], [61]. However, these approaches are designed for entirely different problem settings than the one focused on in this work. Notably, all the above methods are not inductive (for graph-based transfer learning tasks) nor are they able to learn sparse space-efficient node embeddings as achieved by DeepGL. Other key differences were summarized previously in Section 1.

Inductive Embedding Methods. While most work has focused on transductive (within-network) learning, there has been some recent work on graph-based inductive approaches [19], [62]. Yang et al. [62] proposed an inductive approach called Planetoid. However, Planetoid is an embedding-based approach for semi-supervised learning and does not use any structural features. DeepGL first appeared in a manuscript published in April 2017 as R.

Rossi et al., "Deep Feature Learning for Graphs" [19]. A few months later, Hamilton et al. [63] proposed GraphSage that shared many of the ideas proposed in [19] for learning inductive node embeddings. Moreover, GraphSage is a special case of DeepGL when the concatenated features at each layer are simply fed into a fully connected layer with nonlinear activation function σ. However, that work focused on node classification whereas our work focuses on link prediction and link classification.

Many node embedding methods are based on traditional random walks (using node ids) [13], [15], [35], [36] and therefore are not inductive nor do they capture roles. Recently, Ahmed et al. [39] proposed an inductive network representation framework called role2vec that learns inductive role-based node embeddings by first mapping each node to a type via a function Φ and then uses the proposed notion of attributed (typed) random walks to derive inductive role-based embeddings that capture structural similarity [39]. The role2vec framework [39] was shown to generalize many existing random walk-based methods for inductive learning tasks on networks. Other work by Lee et al. [64] uses a technique to construct artificial graph data heuristically from input data that can then be used for transfer learning. However, that work is fundamentally different from our own as it constructs a graph from non-relational data whereas DeepGL is designed for actual real-world graph data such as social, biological, and information networks. Moreover, graphs derived from non-relational data are often dense and most certainly different in structure from the real-world graphs investigated in our work [65].

Higher-Order Network Analysis. Graphlets (network motifs) are small induced subgraphs and have been used for graph classification [2] and visualization/exploratory analysis [24]. However, DeepGL uses graphlet frequencies as base features for learning higher-order node and edge functions from large networks that generalize for inductive learning tasks.

Sparse Graph Feature Learning. This work proposes the first practical space-efficient approach that learns sparse node/edge feature vectors. Notably, DeepGL requires significantly less space than existing node embedding methods [13], [15], [16] (see Section 4). In contrast, previous work learns completely dense feature vectors which is impractical for any relatively large network, e.g., they require more than 3TB of memory for a 750 million node graph with 1K features.

6 Conclusion

This work introduced the notion of *relational function* representing a composition of relational feature operators applied over an initial base graph feature. Using this notion as a basis, we proposed DeepGL, a general, flexible, and highly expressive framework for learning deep node and edge relational functions (features) that generalize for (inductive) across-network transfer learning tasks. The framework is flexible with many interchangeable components, expressive, interpretable, parallel, and is both space- and time-efficient for large graphs with runtime that is linear in the number of edges. DeepGL has all the following desired properties:

- *Effective* for learning features that generalize for graph-based transfer learning *and* large (attributed) graphs
- *Space-efficient* requiring up to 6x less memory
- *Fast* with up to 106x speedup in runtime performance
- *Accurate* with a mean improvement in AUC of 20 percent or more on many applications
- *Expressive and flexible* with many interchangeable components making it useful for a range of applications, graph types, and learning scenarios.

References

[1] J. Neville and D. Jensen, "Iterative classification in relational data," in *Proc. AAAI Workshop Learn. Statistical Models Relational Data*, 2000, pp. 13–20.

[2] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.

[3] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data Mining Knowl. Discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[4] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social Network Data Analytics*, Berlin, Germany: Springer, 2011, pp. 243–275.

[5] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal Networks*, Berlin, Germany: Springer, 2013, pp. 15–40.

[6] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proc. National Academy Sci. United States America*, vol. 101, no. 9, pp. 2658–2663, 2004.

[7] R. A. Rossi and N. K. Ahmed, "Role discovery in networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 4, pp. 1112–1131, Apr. 2015.

[8] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable graph exploration and visualization: Sensemaking challenges and opportunities," in *Proc. Int. Conf. Big Data Smart Comput.*, 2015, pp. 271–278.

[9] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama, "Pairwise alignment of protein interaction networks," *J. Comput. Biol.*, vol. 13, no. 2, pp. 182–199, 2006.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[11] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, "Transforming graph data for statistical relational learning," *J. Artif. Intell. Res.*, vol. 45, no. 1, pp. 363–441, 2012.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Representations Workshop*, 2013, pp. 1–12.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2014, pp. 701–710.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[15] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2016, pp. 855–864.

[16] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1067–1077.

[17] A. Vellido, J. D. Martín-Guerrero, and P. J. Lisboa, "Making machine learning models interpretable," in *Proc. Eur. Symp. Artif. Neural Netw. Comput. Intell. Mach. Learn.*, 2012, pp. 163–172.

[18] A. Bibal and B. Frénay, "Interpretability of machine learning models and representations: An introduction," in *Proc. Eur. Symp. Artif. Neural Netw. Comput. Intell. Mach. Learn.*, 2016, pp. 77–82.

[19] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Deep feature learning for graphs," arXiv:1704.08829, 2017, Art. no. 11.

[20] Y. Bengio, "Deep learning of representations: Looking forward," in *Proc. Int. Conf. Statistical Language Speech Process.*, 2013, pp. 1–37.

[21] Y. Bengio, "Learning deep architectures for AI," *Foundations Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2010, pp. 410–421.

[24] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient graphlet counting for large networks," in *Proc. IEEE Int. Conf. Data Mining*, 2015, Art. no. 10.
[25] N. K. Ahmed, T. L. Willke, and R. A. Rossi, "Estimation of local subgraph counts," in *Proc. IEEE Int. Conf. Big Data*, 2016, pp. 586–595.
[26] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinf.*, vol. 23, no. 2, pp. e177–e183, 2007.
[27] Y. LeCun, B. Boser, *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[28] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Estimation of graphlet counts in massive networks," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: 10.1109/TNNLS.2018.2826529.
[29] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[30] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015. [Online]. Available: http://networkrepository.com
[31] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.
[32] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput.*, vol. 15, pp. 1373–1396, 2002.
[33] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, "Modeling dynamic behavior in large evolving graphs," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 667–676.
[34] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 891–900.
[35] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *Proc. 23rd ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2017.
[36] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 377–386.
[37] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," arXiv:1706.07845, 2017.
[38] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 203–209.
[39] N. K. Ahmed, R. Rossi, J. B. Lee, X. Kong, T. L. Willke, R. Zhou, and H. Eldardiry, "Learning role-based graph embeddings," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 1–8.
[40] C. Shi, X. Kong, Y. Huang, S. Y. Philip, and B. Wu, "HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2479–2492, Oct. 2014.
[41] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 119–128.
[42] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 135–144.
[43] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, 2017, pp. 295–304.
[44] L. Xu, X. Wei, J. Cao, and P. S. Yu, "Embedding of embedding (EOE): Joint embedding for coupled heterogeneous networks," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, 2017, pp. 741–749.
[45] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, 2017, pp. 731–739.
[46] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proc. SIAM Int. Conf. Data Mining*, 2017, pp. 633–641.
[47] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," arXiv:1705.04969, 2017.
[48] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 2111–2117.
[49] J. Liang, P. Jacobs, J. Sun, and S. Parthasarathy, "Semi-supervised embedding in attributed networks with outliers," in *Proc. SIAM Int. Conf. Data Mining*, 2018, pp. 153–161.
[50] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking," arXiv:1707.03815, 2017.
[51] C. W. Coley, R. Barzilay, W. H. Green, T. S. Jaakkola, and K. F. Jensen, "Convolutional embedding of attributed molecular graphs for physical property prediction," *J. Chemical Inf. Modeling*, vol. 57, pp. 1757–1772, 2017.
[52] J. B. Lee, R. A. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 1–9.
[53] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 571–578.
[54] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 387–396.
[55] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Proc. Companion Proc. Web Conf.*, 2018, pp. 969–976.
[56] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," arXiv:1603.08861, 2016.
[57] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.
[58] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1168–1175.
[59] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.
[60] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1225–1234.
[61] R. A. Rossi, N. K. Ahmed, and E. Koh, "Higher-order network representation learning," in *Proc. Companion Proc. Web Conf.*, 2018, pp. 3–4.
[62] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," arXiv:1603.08861, 2016.
[63] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Adv. Neural Inf. Process. Syst.*, pp. 1024–1034, 2017.
[64] J. Lee, H. Kim, J. Lee, and S. Yoon, "Transfer learning for deep learning on graph-structured data," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2154–2160.
[65] J. P. Canning, *et al.*, "Network classification and categorization," in *Proc. Int. Conf. Complex Netw.*, 2018, pp. 1–3.

Ryan A. Rossi received the MS and PhD degrees in computer science from Purdue University. He is a machine learning research scientist at Adobe Research. His research lies in the fields of machine learning; and spans theory, algorithms, and applications of large complex relational (network/graph) data from social and physical phenomena. Before joining Adobe Research, he had the opportunity to work at a number of industrial, government, and academic research labs including the Palo Alto Research Center (Xerox PARC), Lawrence Livermore National Laboratory (LLNL), Naval Research Laboratory (NRL), NASA Jet Propulsion Laboratory (JPL)/California Institute of Technology, and University of Massachusetts Amherst, among others. He was a recipient of the National Science Foundation Graduate Research Fellowship (NSF GRFP), National Defense Science and Engineering Graduate Fellowship (NDSEG), the Purdue Frederick N. Andrews Fellowship, and Bilsland Dissertation Fellowship awarded to Outstanding PhD candidates.

Rong Zhou is currently at Google. Prior to that, he was a senior researcher and manager of the High-Performance Analytics area of the Interaction and Analytics Laboratory at PARC. His research interests include large-scale graph algorithms, heuristic search, machine learning, automated planning, and parallel model checking. He has published extensively in top journals and conferences in the field of artificial intelligence, and his work received two Best Paper Awards from the International Conferences on Automated Planning and Scheduling (ICAPS) in 2004 and 2005, respectively. He is the co-chair of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR) 2008, the co-chair of the International Symposium on Combinatorial Search (SoCS) 2009, and the tutorial co-chair of ICAPS 2011. He holds 21 US and 14 international patents in the areas of parallel algorithms, planning and scheduling, disk-based search, and diagnosis. He is the recipient of four Golden Acorn Awards from PARC. He currently serves on the editorial board of the *Journal of Artificial Intelligence Research*.

Nesreen K. Ahmed received the MS degree in statistics and computer science from Purdue University, in 2014, and the PhD degree from the Computer Science Department, Purdue University, in 2015. She is a senior research scientist at Intel Labs. In 2018, she is the PC Chair of the IEEE Big Data Conference. She was a visiting researcher at Facebook, Adobe Research, Technicolor, and Intel analytics. Her research interests in machine learning and data mining spans the theory and algorithms of large-scale machine learning, graph theory, and their applications in social and information networks. She has authored numerous papers/tutorials in top-tier conferences/journals. Her research was selected among the best papers of ICDM in 2015, BigMine in 2012, and covered by popular press such as the *MIT Technology Review*. She was selected by UC Berkeley among the top female rising stars in computer science and engineering in 2014. She holds two US patents filed by Adobe, and coauthored the open source network data repository (http://networkrepository.com).