Deep Inductive Network Representation Learning

Ryan A. Rossi
Adobe Research
rrossi@adobe.com

Rong Zhou
Google
rongzhou@google.com

Nesreen K. Ahmed
Intel Labs
nesreen.k.ahmed@intel.com

ABSTRACT

This paper presents a general inductive graph representation learning framework called DeepGL for learning deep node *and* edge features that generalize across-networks.[1] 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 $O(|E|)$, 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 6× less memory, (3) **fast** with up to 182× speedup in runtime performance, and (4) **accurate** with an average improvement in AUC of 20% or more on many learning tasks and across a wide variety of networks.

KEYWORDS

Inductive network representation learning, inductive learning, transfer learning, network embeddings, representation learning, attributed networks, function learning, network motifs, deep learning

ACM Reference Format:
Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2018. Deep Inductive Network Representation Learning. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184558.3191524

1 INTRODUCTION

Learning a useful graph representation lies at the heart *and* success of many machine learning tasks such as node and link classification [20, 34], anomaly detection [5], link prediction [6], dynamic network analysis [21], community detection [25], role discovery [27], visualization and sensemaking [24], network alignment [16], and many others. Indeed, the success of machine learning methods largely depends on data representation [12, 28]. 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 [28].

Recent work has largely been based on the popular skip-gram model [18] originally introduced for learning vector representations of words in the natural language processing (NLP) domain. In particular, DeepWalk [23] applied the successful word embedding framework from [19] (called word2vec) to embed the nodes such that the co-occurrence frequencies of pairs in short random walks are preserved. More recently, node2vec [13] 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, 23, 32] 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.[2] 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 [9, 33]. Furthermore, existing embedding methods are also unable to capture higher-order subgraph structures as well as learn a hierarchical graph representation from such higher-order 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 3). 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.[3] Intuitively, DeepGL begins by deriving a set of base features using the graph structure and any attributes (if available). 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

[1] This manuscript first appeared in April 2017 as R. Rossi *et al.*, "Deep Feature Learning for Graphs" [30].

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191524

[2] The terms transfer learning and inductive learning are used interchangeably.
[3] Note a deep learning method as defined by Bengio *et al.* [7, 8] is one that learns multiple levels of representation with higher levels capturing more abstract concepts through a deeper composition of computations [12, 17]. This definition includes neural network based approaches as well as DeepGL and many other deep learning paradigms.

(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 approach, DeepGL, provides a general powerful framework for learning deep graph representations from attributed graphs that are naturally inductive for use in across-network learning tasks. DeepGL overcomes many limitations of existing work and has the following key properties:

- **Novel framework**: This paper presents a deep hierarchical inductive graph representation learning framework called DeepGL for large (attributed) networks that generalizes for discovering both node and edge features. DeepGL searches a space of relational functions (representing features) that are expressed as compositions of relational feature operators applied to a set of base features. The framework is flexible with many interchangeable components, expressive, and shown to be effective for a wide variety of applications.
- **Inductive representation learning**: Contrary to existing node embedding methods, DeepGL is naturally inductive by learning relational functions that generalize for computation on any arbitrary graph and therefore supports across-network transfer learning tasks.
- **Space efficiency**: While most existing methods learn dense high-dimensional feature vectors that are often impractical for large graphs (*e.g.*, too large to fit in-memory), DeepGL is *space-efficient* by learning a sparse graph representation that requires up to 6x less space than existing work.
- **Fast, 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 3.
- **Hierarchical graph representation**: DeepGL learns hierarchical graph representations with multiple layers where each successive layer uses the output from the previous layer as input to derive features of a higher-order.
- **Interpretable and explainable**: Unlike existing embedding methods, DeepGL learns interpretable and explainable features.

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).[4]

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

Table 1: Summary of notation. Matrices are bold upright roman letters; vectors are bold lowercase letters.

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
$\mathcal{G}$	set of graph elements $\{g_1, g_2, \cdots\}$ (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 $\Gamma(g_i) = \{g_j \in \mathcal{G} \mid \text{dist}(g_i, g_j) \leq \ell\}$
$\text{dist}(g_i, g_j)$	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 i-th element of $\mathbf{x}$ for graph element g_i
$\lvert\mathbf{X}\rvert$	number of nonzeros in a matrix $\mathbf{X}$
$\mathcal{F}$	set of *feature definitions/functions* from DeepGL
$\mathcal{F}_k$	k-th feature layer (where k is the depth)
f_i	relational function (definition) of $\mathbf{x}_i$
$\mathbf{\Phi}$	set of relational operators $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$
$\mathbb{K}(\cdot)$	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

2.1 Base Graph Features

The first step of DeepGL (Alg. 1) is to derive a set of *base graph features*[5] 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 decompose G into its smaller subgraph components called graphlets (network motifs) [1] using local graphlet decomposition methods [3] and concatenate the graphlet count-based feature vectors to the feature matrix $\mathbf{X}$. 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.

We also 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:

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

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 [4]. Given a node v and an integer ℓ, the ℓ-egonet of v is defined as the set of graph elements ℓ-hops away from v (*i.e.*, distance at most ℓ) and all edges and nodes between that set. The external and within-egonet features for nodes are provided

[5]The term *graph feature* refers to an edge or node feature.

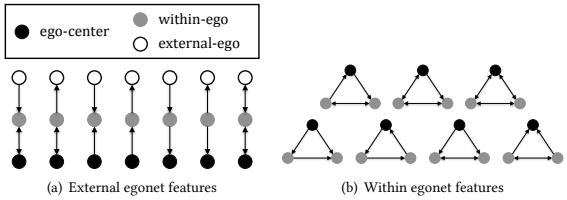

Figure 1: Egonet Features. The set of base (ℓ=1 hop)-egonet graph features. (a) the external egonet features; (b) the within egonet features. See the legend for the vertex types: ego-center (●), within-egonet vertex (●), and external egonet vertices (○).

in Figure 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 element (node, edge) in G.

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 the initial base features.

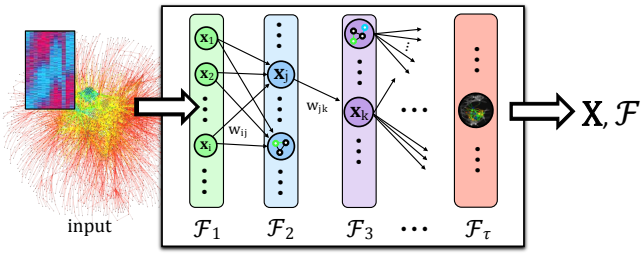

Figure 2: Overview of the DeepGL architecture for graph representation learning. Let $\mathbf{W} = [w_{ij}]$ be a matrix of feature weights where w_{ij} (or W_{ij}) is the weight between the feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Notice that $\mathbf{W}$ has the constraint that $i < j < k$ 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* $\mathcal{F}_h = \{\cdots, f_k, \cdots\}$ 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.

2.2 Relational Function Space & Expressivity

In this section, we formulate the space of relational functions[6] that can be expressed and searched over by DeepGL. A relational function (feature) in DeepGL is defined as a composition of relational feature operators applied to an initial base feature $\mathbf{x}$. Recall that unlike recent node embedding methods [13, 23, 32], 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* $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$. A few relational feature operators are defined formally in Table 2; see [28] (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: (*i*) the initial base features (derived using the graph structure, initial attributes given as input, or both), (*ii*) a set of *relational feature operators* $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$, (*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 $\Phi_p \in \mathbf{\Phi}$, 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.[7] 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 (Figure 2). Intuitively, just as filters are used in Convolutional Neural Networks (CNNs) [12], 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 Functions. We can also derive a wide variety of *relational functions* compositionally by adding and multiplying relational functions (*e.g.*, $\Phi_i + \Phi_j$, and $\Phi_i \times \Phi_j$). 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* is analogous an AND operation as two instances are close if both relational functions have large values.

2.3 Searching the Relational Function Space

A general and flexible framework for DeepGL is given in Alg. 1. Recall that DeepGL begins by deriving a set of base features which are used as a basis for learning deeper and more discriminative

[6] The terms graph function and relational function are used interchangeably.

[7] 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.

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 αM graph elements (edges/nodes) 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. 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 (Figure 2) 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 $\mathcal{F}_2, \mathcal{F}_3, \cdots, \mathcal{F}_\tau$ are derived as follows (Alg. 1 Lines 4-11): 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 $\mathbf{\Phi}$ to each of the novel features $f_i \in \mathcal{F}_{\tau-1}$ learned in the previous layer (Alg. 1 Line 5). An algorithm for deriving a feature layer is provided in Alg. 2. Next, the feature vectors from layer $\mathcal{F}_\tau$ are transformed in Line 7 as discussed previously.

The resulting features in layer τ are then evaluated. The feature evaluation routine (in Alg. 1 Line 8) 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 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 (Alg. 3) called in Line 8 of Alg. 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 (Alg. 3 Line 2-6). More formally, let E_F denote the set of connections representing dependencies between features:

$$E_F = \left\{(i,j) \mid \forall (i,j) \in |\mathcal{F}| \times |\mathcal{F}| \; s.t. \; \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) > \lambda\right\} \tag{2}$$

Table 2: Definitions of a few relational feature operators. 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 $S \in \left\{\Gamma_\ell(g_i), \Gamma^+_\ell(g_i), \Gamma^-_\ell(g_i)\right\}$. The relational operators generalize to ℓ-distance neighborhoods (*e.g.*, $\Gamma_\ell(g_i)$ where ℓ is the distance). Note $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_i & \cdots \end{bmatrix} \in \mathbb{R}^M$ where x_i is the i-th element of $\mathbf{x}$ for g_i.

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

Algorithm 1 DeepGL: Deep Inductive Graph Representation Learning Framework

Require:
a (un)directed graph $G = (V, E)$; a set of relational feature operators $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$, 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 features (see text 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 $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$ to features $\begin{bmatrix} \cdots & \mathbf{x}_i & \mathbf{x}_{i+1} & \cdots \end{bmatrix}$ given as output in the previous layer $\mathcal{F}_{\tau-1}$ (via Alg. 2). Add feature vectors to $\mathbf{X}$ and functions/def. to $\mathcal{F}_\tau$.
6: Diffuse new feature vectors via a feature diffusion process (Eq. 3)
7: Transform feature vectors of layer $\mathcal{F}_\tau$
8: Evaluate the features (functions) in layer $\mathcal{F}_\tau$ using the criterion $\mathbb{K}$ to score feature pairs along with a feature selection method to select a subset (Alg. 3).
9: Discard features from $\mathbf{X}$ that were pruned (not in $\mathcal{F}_\tau$) and set $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_\tau$
10: Set $\tau \leftarrow \tau + 1$ and initialize $\mathcal{F}_\tau$ to $\varnothing$ for next feature layer
11: **until** no new features emerge **or** the max number of layers is reached
12: **return** $\mathbf{X}$ and the set of relational functions (definitions) $\mathcal{F}$

The result is a *weighted feature dependence graph* $\mathcal{G}_F$. 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 $\{C_1, C_2, \ldots\}$ where each set $C_k \in C$ represents features that are dependent (though not necessarily pairwise dependent). To partition the feature graph $\mathcal{G}_F$, Alg. 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 Alg. 3 the earliest feature in each connected component $C_k = \{..., f_i, ..., f_j, ...\} \in C$ 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 $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_\tau$ (Alg. 1: Line 9). Next, Line 10 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 4-11).

In contrast to node embedding methods that output only a *node* feature matrix $\mathbf{X}$, DeepGL also outputs the (hierarchical) relational functions $\mathcal{F}$ corresponding to the learned features. 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

Algorithm 2 Derive a feature layer using the features from the previous layer and the set of relational feature operators $\mathbf{\Phi} = \{\Phi_1, \cdots, \Phi_K\}$.

1 **procedure** FeatureLayer(G, $\mathbf{X}$, $\mathbf{\Phi}$, $\mathcal{F}$, $\mathcal{F}_\tau$, 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 $\Phi \in \mathbf{\Phi}$ **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** $\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) > \lambda$ **then**
6 $E_F = E_F \cup \{(i, j)\}$
7 $W_{ij} = \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j)$
8 Partition $\mathcal{G}_F$ using conn. components $C = \{C_1, C_2, \ldots\}$
9 **parallel for each** $C_k \in C$ **do** ▷ Remove features
10 Find f_i s.t. $\forall f_j \in C_k : i < j$.
11 Remove C_k from $\mathcal{F}_\tau$ and set $\mathcal{F}_\tau \leftarrow \mathcal{F}_\tau \cup \{f_i\}$

as the relational functions can be used to derive embeddings for new nodes or even graphs.

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

$$\bar{\mathbf{X}}^{(t)} = \mathbf{D}^{-1}\mathbf{A}\bar{\mathbf{X}}^{(t-1)} \tag{3}$$

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 $t = 1, 2, ..., T$ 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

$$\bar{\mathbf{X}}^{(t)} = (1-\theta)\mathbf{L}\bar{\mathbf{X}}^{(t-1)} + \theta\mathbf{X}, \qquad \text{for } t = 1, 2, \ldots \tag{4}$$

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

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2} \tag{5}$$

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 9 of Alg. 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 Computational Complexity

Recall that M is the number of edges, N is the number of nodes, and F is the number of features.

2.5.1 Learning. The total computational complexity of the *edge representation learning* from the DeepGL framework is:

$$O\big(F(M + MF)\big) \tag{6}$$

For *node representation learning*, the time complexity of DeepGL is:

$$O\big(F(M + NF)\big) \tag{7}$$

Thus, in both cases, the runtime of representation learning in DeepGL is linear in the number of edges. As an aside, the initial graphlet features are computed using fast and accurate estimation methods, see Ahmed *et al.* [3].

2.5.2 Inductive relational functions. We now state the computational complexity of directly computing the set of inductive relational functions (feature definitions) which were previously learned on another arbitrary graph. Computation of the relational functions $\mathcal{F}$ on another arbitrary graph is important for inductive across-network learning tasks. Given the set of learned relational functions $\mathcal{F}$, the total computational complexity of the *edge relational functions* is:

$$O(FM) \tag{8}$$

Similarly, the time complexity of the *node relational functions* is also $O(FM)$. Thus, the runtime of deriving the edge and node relational functions in DeepGL 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 2.5.1). The key difference is that features are not evaluated when deriving the relational functions directly. In contrast, representation learning in DeepGL scores the features at each layer.

3 EXPERIMENTS

This section demonstrates the effectiveness of the proposed framework.

3.1 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:

$$\mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\big|\{(x_{ik}, x_{jk}),\ \forall k = 1, \ldots, N \mid x_{ik} = x_{jk}\}\big|}{N} \tag{9}$$

where x_{ik} and x_{jk} are the k-th feature value of the N-dimensional vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. Unless otherwise mentioned, we 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 $\mathbf{\Phi} \in \big\{\Phi_{\text{mean}}, \Phi_{\text{sum}}, \Phi_{\text{prod}}, \{\Phi_{\text{mean}}, \Phi_{\text{sum}}\}, \{\Phi_{\text{prod}}, \Phi_{\text{sum}}\}, \{\Phi_{\text{prod}}, \Phi_{\text{mean}}\}\big\}$. 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% 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 node2vec [13], DeepWalk [23], and LINE [32]. For node2vec, we use the hyperparameters and grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [13]. The experimental setup mentioned in [13] 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 [26].[8]

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 $\lvert\mathbf{x}_i - \mathbf{x}_j\rvert$	**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

3.2 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; $|\mathbf{x}_i - \mathbf{x}_j|$ and $(\mathbf{x}_i - \mathbf{x}_j)^{\circ 2}$ is the WEIGHTED-L_1 and WEIGHTED-L_2 binary operators, respectively.[9] 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% across all graphs and binary operators.

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.

3.3 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, 23, 32]. 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, 23]. We focus first on node representations since existing methods are limited to only node features. Results are shown in Figure 3. In all cases, the node representations learned by DeepGL are extremely sparse and significantly more space-efficient than node2vec [13] as observed in Figure 3. 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 (Figure 3). Moreover, the density of node and edge representations from DeepGL is between $\lfloor 0.162,\ 0.334 \rfloor$ for nodes and $\lfloor 0.164,\ 0.318 \rfloor$ for edges and up to 6× more space-efficient than existing methods.

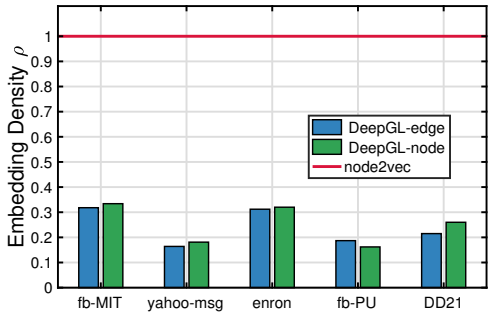

Figure 3: DeepGL requires up to 6x less space than node2vec and other methods that learn dense embeddings.

Notably, recent node embedding methods not only output dense node features, but are also real-valued and often negative (*e.g.*, [13, 23, 32]). 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 10.2GB, whereas DeepGL uses only 0.768GB (assuming a modest 0.3 density) — a significant reduction in space by a factor of 13.

3.4 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 node2vec [13] which is designed specifically to be *scalable* for large graphs and shown to be faster than DeepWalk and LINE. Default parameters are used for each method. In Figure 4, we observe that DeepGL is significantly faster and more scalable than node2vec. In particular, node2vec takes 1.8 days (45.3 hours) for 10 million nodes, whereas DeepGL finishes in only 15 minutes; see Figure 4. Strikingly, this is 182 times faster than node2vec.

[8] See http://networkrepository.com/ for data description and statistics
[9] Note $\mathbf{x}^{\circ 2}$ is the element-wise Hadamard power; $\mathbf{x}_i \odot \mathbf{x}_j$ is the element-wise product.

Table 4: AUC scores for node classification

graph	$\|C\|$	DeepGL	node2vec
DD242	20	**0.730**	0.673
DD497	20	**0.696**	0.660
DD68	20	**0.730**	0.713
ENZYMES118	2	**0.779**	0.610
ENZYMES295	2	**0.872**	0.588
ENZYMES296	2	**0.823**	0.610

3.5 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 Figure 5, 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` on a machine with 4 Intel Xeon E5-4627 v2 3.3GHz CPUs. Other graphs and machines gave similar results.

3.6 Node Classification

For node classification, we use the *i.i.d.* variant of RSM [29] since it is able to handle multiclass problems in a direct fashion (as opposed to indirectly, *e.g.*, one-vs-rest) and consistently outperformed other indirect approaches such as LR and SVM. In particular, RSM assigns a test vector $\mathbf{x}_i$ to the class that is most similar *w.r.t.* the training vectors (*i.e.*, feature vectors of the nodes with known labels); see [29] for further details. Similarity is measured using the RBF kernel and RBF's hyperparameter σ is set using cross-validation with a grid search over $\sigma \in \{0.001, 0.01, 0.1, 1\}$. Results are shown in Table 4. In all cases, we observe that DeepGL significantly outperforms node2vec across all graphs and node classification problems including both binary and multiclass problems. Further, DeepGL achieves the best improvement in AUC on ENZYMES295 of 48%. As an aside, results for DeepWalk and LINE were removed for brevity since node2vec outperformed them in all cases.

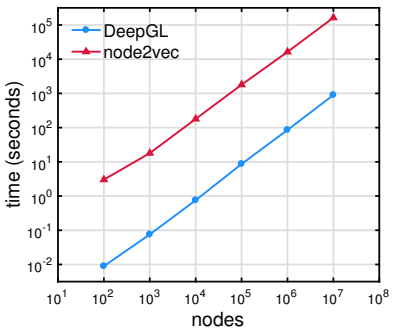

Figure 4: Runtime comparison on Erdös-Rényi graphs with an average degree of 10. The proposed approach is shown to be orders of magnitude faster than node2vec [13].

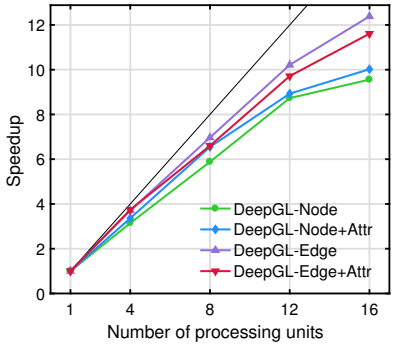

Figure 5: Parallel speedup of different variants from the DeepGL framework. See text for discussion.

4 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, 22, 23, 32]. In particular, recent methods that apply the popular word2vec framework to learn node embeddings [13, 23, 32]. The proposed DeepGL framework differs from these methods in six fundamental ways: (1) DeepGL learns complex relational functions that generalize for across-network transfer learning. Features learned from DeepGL on one graph can be extracted from another graph for transfer learning tasks such as network alignment, graph similarity, role discovery, temporal graph modeling, among others. (2) DeepGL learns sparse features and thus is extremely space-efficient for large networks. (3) DeepGL learns important and useful edge *and* node representations whereas existing work is limited to *node features* [13, 23, 32]. (4) DeepGL naturally supports attributed graphs. (5) DeepGL is fast and efficient with a runtime that is linear in the number of edges. (6) DeepGL is also completely parallel and shown in Section 3 to scale strongly.

There is also another related body of work focused on attributed graphs. Recently, Huang *et al.* [14] proposed a label informed embedding method for attributed networks. This approach assumes the graph is labeled and uses this information to improve predictive performance. However, this work is significantly different. First and foremost, while DeepGL is able to naturally support attributed graphs, this work does not focus on such graphs. Moreover, DeepGL does not require attributes or class labels on the nodes. Another important fundamental difference is that DeepGL learns features representing relational functions that generalize for extraction on any other arbitrary graph. The relational functions naturally represent higher-order structures when based on lower-order subgraph features (Figure 2). DeepGL also learns features that are sparse and therefore space-efficient for large graphs. Moreover, it is fast with a runtime that is linear in the number of edges and is completely parallel with strong scaling. There has also been some recent work on heterogeneous network embeddings [10, 11, 37], semi-supervised network embeddings [15, 38], and methods for improving the learned representations [31, 35, 36]. This work investigates entirely different problems than the one discussed in this paper.

We can also use the inferred embeddings for *graph-based transfer learning*. This is possible since DeepGL learns relational functions that generalize across-networks and therefore are easily extracted on another arbitrary graph. Other key differences were summarized previously in Section 1.

Higher-order network analysis: Other methods use high-order network properties (such as graphlet frequencies) as features for graph classification [34]. Graphlets (network motifs) are small induced subgraphs and have been used for graph classification [34], role discovery [2], and visualization and exploratory analysis [1]. However, our work focuses on using graphlet frequencies as base features for learning node and edge representations from large networks. To the best of our knowledge, this paper is the first to use network motifs (including all motifs of size 3, 4, and 5 vertices) as base features for graph representation learning.

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, 23, 32] (see Section 3). In contrast, existing embedding methods store 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.

5 CONCLUSION

We proposed DeepGL, a general, flexible, and highly expressive framework for learning deep node and edge features that generalize for across-network transfer learning tasks. Each feature learned by DeepGL corresponds to a composition of relational feature operators applied over a base feature. Thus, features learned by DeepGL are interpretable and naturally generalize for across-network transfer learning tasks as they can be derived on any arbitrary graph. 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 functions (features) that generalize for graph-based transfer learning *and* large (attributed) graphs
- **Space-efficient** requiring up to 6× less memory
- **Fast** with up to 182× speedup in runtime performance
- **Accurate** with a mean improvement in AUC of 20% 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.
- **Parallel** with strong scaling results.

REFERENCES

[1] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient Graphlet Counting for Large Networks. In *ICDM*. 10.
[2] Nesreen K. Ahmed, Ryan A. Rossi, Theodore L. Willke, and Rong Zhou. 2017. Edge Role Discovery via Higher-order Structures. In *PAKDD*. Springer.
[3] Nesreen K. Ahmed, Theodore L. Willke, and Ryan A. Rossi. 2016. Estimation of Local Subgraph Counts. In *IEEE BigData*. 586–595.
[4] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. *PAKDD* (2010), 410–421.
[5] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *DMKD* 29, 3 (2015), 626–688.
[6] Mohammad Al Hasan and Mohammed J Zaki. 2011. A survey of link prediction in social networks. In *Social Network Data Analytics*. Springer, 243–275.
[7] Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning* 2, 1 (2009), 1–127.
[8] Yoshua Bengio. 2013. Deep learning of representations: Looking forward. In *SLSP*. Springer, 1–37.
[9] Adrien Bibal and Benoît Frénay. 2016. Interpretability of machine learning models and representations: an introduction. In *Proc. ESANN*. 77–82.
[10] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *SIGKDD*. 119–128.
[11] Ting Chen and Yizhou Sun. 2017. Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification. In *WSDM*. 295–304.
[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT Press.
[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[14] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. 731–739.
[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[16] Mehmet Koyutürk, Yohan Kim, Umut Topkara, Shankar Subramaniam, Wojciech Szpankowski, and Ananth Grama. 2006. Pairwise alignment of protein interaction networks. *JCB* 13, 2 (2006), 182–199.
[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR Workshop*.
[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.
[20] Jennifer Neville and David Jensen. 2000. Iterative classification in relational data. In *AAAI Workshop on Learning Statistical Models from Relational Data*. 13–20.
[21] Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. 2013. Graph metrics for temporal networks. In *Temporal Networks*. Springer, 15–40.
[22] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *arXiv:1605.05273*.
[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.
[24] Robert Pienta, James Abello, Minsuk Kahng, and Duen Horng Chau. 2015. Scalable graph exploration and visualization: Sensemaking challenges and opportunities. In *BigComp*.
[25] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. 2004. Defining and identifying communities in networks. *PNAS* 101, 9 (2004), 2658–2663.
[26] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. http://networkrepository.com
[27] Ryan A. Rossi and Nesreen K. Ahmed. 2015. Role Discovery in Networks. *TKDE* 27, 4 (2015), 1112–1131.
[28] Ryan A. Rossi, Luke K. McDowell, David W. Aha, and Jennifer Neville. 2012. Transforming graph data for statistical relational learning. *JAIR* 45, 1 (2012), 363–441.
[29] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2016. Relational Similarity Machines. In *KDD MLG*. 1–8.
[30] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2017. Deep Feature Learning for Graphs. In *arXiv:1704.08829*. 11.
[31] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.
[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding.. In *WWW*.
[33] Alfredo Vellido, José David Martín-Guerrero, and Paulo JG Lisboa. 2012. Making machine learning models interpretable.. In *ESANN*, Vol. 12. 163–172.
[34] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *JMLR* 11 (2010), 1201–1242.
[35] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. 1225–1234.
[36] Jason Weston, Frédéric Ratle, and Ronan Collobert. 2008. Deep learning via semi-supervised embedding. In *ICML*. 1168–1175.
[37] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2017. Embedding of Embedding (EOE): Joint Embedding for Coupled Heterogeneous Networks. In *WSDM*. ACM, 741–749.
[38] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).