LouvainNE: Hierarchical Louvain Method for High Quality and Scalable Network Embedding*

Ayan Kumar Bhowmick
Indian Institute of Technology, Kharagpur, India
ayankb@iitkgp.ac.in

Koushik Meneni
Indian Institute of Technology, Kharagpur, India
koushik190498@gmail.com

Maximilien Danisch
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France
maximilien.danisch@lip6.fr

Jean-Loup Guillaume
L3I, University of La Rochelle, France
jean-loup.guillaume@univ-lr.fr

Bivas Mitra
Indian Institute of Technology, Kharagpur, India
bivas@cse.iitkgp.ac.in

ABSTRACT

Network embedding, that aims to learn low-dimensional vector representation of nodes such that the network structure is preserved, has gained significant research attention in recent years. However, most state-of-the-art network embedding methods are computationally expensive and hence unsuitable for representing nodes in billion-scale networks. In this paper, we present *LouvainNE*, a hierarchical clustering approach to network embedding. Precisely, we employ *Louvain*, an extremely fast and accurate community detection method, to build a hierarchy of successively smaller subgraphs. We obtain representations of individual nodes in the original graph at different levels of the hierarchy, then we aggregate these representations to learn the final embedding vectors. Our theoretical analysis shows that our proposed algorithm has quasi-linear runtime and memory complexity. Our extensive experimental evaluation, carried out on multiple real-world networks of different scales, demonstrates both (i) the scalability of our proposed approach that can handle graphs containing tens of billions of edges, as well as (ii) its effectiveness in performing downstream network mining tasks such as network reconstruction and node classification.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**;
• **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

Network embedding; scalability; real-world graph algorithms

1 INTRODUCTION

Representation learning on graphs, or network embedding [11, 19], involves a mapping of nodes in the graph to a low-dimensional vector space, such that the topological structure of the network is preserved. Such learned embedding vectors can be efficiently used as features for carrying out various network mining tasks. Most of the existing network embedding approaches employ random walks on graphs, matrix factorization techniques and deep learning architectures [20, 36, 44] to represent nodes. However, these methods are computationally expensive for networks containing billions of edges.

Recently, few embedding methods [9, 28, 29] that have proposed hierarchical approaches to learn node embeddings have been developed. For instance, Ma et al. [29] captures the latent hierarchical taxonomy denoting categories of different granularity in the learned vertex representations. *HARP* [9] and *MILE* [28] repeatedly coarsens the original graph into a series of smaller graphs. Next, vertex representations of the coarsened graphs are learned using state-of-the-art embedding approaches, followed by a refinement step to obtain the final node embeddings of original graph. However, graph convolution network and gradient updates introduce a high computational overhead. Recently proposed *RandNE* [54] is faster than other approaches on large-scale networks, however it compromises with the embedding quality unless its numerous parameters are accurately tuned. Another recent method is *ProNE* [53] which learns high quality embeddings, but do not scale well for networks with billions edges.

In this paper, we leverage the notion of community structures present in real networks [17] as an effective mechanism to compute node embeddings. The nodes present in a single community have similar types and are densely connected among themselves [31]. Hence, placing those nodes closely in the embedding space may facilitate multiple graph mining tasks such as node clustering and node classification, as well as network reconstruction and link prediction. Partitioning the graph recursively into a series of coarsened subgraphs (communities) can help to capture similarity between nodes at different levels of proximity: the recursive partitioning essentially creates a hierarchy of communities in the network, where (a) the nodes present in the same community at the top level of the hierarchy indicates a cluster of similar nodes with higher-order structural relationships and (b) communities lower down in the hierarchy preserve the neighborhood relationship between connected pairs of nodes. Hence, generating the embedding vectors for a node from its presence in the communities at various levels of the hierarchy preserves the embedding quality and makes it suitable for various graph mining tasks. Side by side, one may use state-of-the-art fast community detection algorithms to make this process scalable for large-scale networks [32].

The major contribution of this paper is to develop a fast and scalable network embedding framework, *LouvainNE*, which is able

*This research was partially supported by the DST - CNRS funded Indo - French collaborative project "Evolving Communities and Information Spreading" and French National Agency (ANR) under the JCJC project LiMass http://bit.ly/LiMass.

to generate high quality embeddings for networks containing tens of billions of edges. First, we define the problem of scalable network embedding and explain the scope of community structures present in the network to efficiently learn node embedding vectors (Section 2). Leveraging these insights, we develop *LouvainNE*, a framework for node embedding based on hierarchical community detection. The development of *LouvainNE* involves three major steps: (a) A hierarchical clustering method is proposed to build a hierarchy of subgraphs. We use the *Louvain* algorithm [5] to recursively coarsen a large graph into smaller communities and construct the hierarchy of subgraphs. (b) Next, we generate the level specific node embeddings for each subgraph in the hierarchy. We propose two different approaches to embed each subgraph (i) stochastic embedding (ii) standard embedding. (c) Finally, we combine the obtained embeddings at different levels of the hierarchy into the final embedding of individual nodes in the graph (Section 3). We introduce various real-world network datasets and describe the state-of-the-art network embedding methods used as baselines (Section 4). We perform extensive evaluation of the quality of the learned embedding vectors using *LouvainNE* on various downstream graph mining tasks such as network reconstruction and node classification. Our evaluation demonstrates the effectiveness of *LouvainNE* in learning high quality embedding vectors, which significantly outperform state-of-the-art methods, especially for large-scale datasets (Section 5). We explore the different variants of *LouvainNE* implementations as well as investigate the effect of tuning the model hyperparameters on the performance of *LouvainNE* (Section 6). Finally, we perform scalability evaluation on multiple real-world networks with up to 20 billions of edges which shows that *LouvainNE* scales linearly with the size of the network, while state-of-the-art algorithms fail to generate embeddings in a reasonable amount of time (up to 5 days). We present the related work in Section 8.

2 PROBLEM STATEMENT AND KEY IDEA

In this section, we first formulate the problem of scalable network embedding. Next, we provide the detailed intuition behind the proposed methodology.

2.1 Notations and Problem Definition

Let $G = (\mathcal{V}, \mathcal{E})$ denote an undirected graph where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ is the set of $N = |\mathcal{V}|$ nodes and $\mathcal{E}$ is the set of $M = |\mathcal{E}|$ undirected edges in G. In this paper, our objective is to find a mapping function f that learns a low-dimensional embedding vector $\mathbf{y_v}$ of dimension d for every node $v \in G$ where $d << N$ is the predefined number of dimensions of the embedding. Mathematically, we learn the function $f : \mathcal{V} \rightarrow \mathcal{R}^d$ such that $\forall v \in \mathcal{V}$, $f(v) = \mathbf{y_v}$. The function f is learned in such a way that it preserves various network properties in the embedding space. For instance, (a) preserving the neighborhood proximity between connected node pairs (b) keeping the similar albeit non-adjacent nodes relatively close to each other (c) ensuring that dissimilar and non-adjacent nodes are placed far away (d) preserving higher-order structural relationships in the embedding space (e) representing the graph topology, are essential requirements of learning embedding vectors.

Apart from generating high quality node embedding vectors, our aim is to ensure that the developed algorithm is fast and highly scalable for large-scale networks. It should be able to learn embedding vectors in a network consisting of a few billion nodes and edges efficiently in a reasonable time, without compromising the quality of embedding for effectively performing various downstream network mining tasks.

2.2 Embedding and community: Key idea behind *LouvainNE*

In real-world networks, the distribution of links is inhomogeneous, with high concentrations of links within specific groups of nodes, and low concentrations between these groups [10]. This feature of real networks results in the formation of community structure, where nodes inside the community are more densely connected, than with the rest of the network [51]. Communities in a network are groups of nodes which probably share common properties and/or play similar roles within the network. We envision the notion of community as a useful tool to compute node embeddings. Indeed, it is intuitive to place nodes in the same community closely in the embedding space, compared to the nodes in different communities. The node embedding vectors learned from this community structure may facilitate to cluster the similar nodes in the graph, as well as infer the node types effectively. Moreover, dense connections within community will also help to preserve the higher-order structural relationships. Unfortunately, vanilla community structure does not capture the neighborhood property of the connected node pairs. However, if we construct a hierarchy of subgraphs by repeatedly identifying smaller sub-communities from the larger communities in the network, this will help to preserve the local proximity between node pairs in the network, since the neighbors of a node are more likely to be in the same sub-community lower down in the hierarchy. This will effectively preserve the neighborhood relationship between two nodes and will help to accurately reconstruct the original network. Finally, the presence of a wide variety of fast community detection algorithms in the literature [32] capable of detecting communities in massive-scale graphs in a very short time, paves the way of utilizing community structure for developing scalable node embedding algorithms. In this paper, we leverage these aforesaid observations to develop *LouvainNE*.

3 DEVELOPMENT OF *LouvainNE*

LouvainNE involves three broad steps for learning the embeddings in large-scale networks. First, we propose a fast method to **create a hierarchy** of partitions of the original graph. Next, we generate **level specific node embeddings** for each partition in the hierarchy. Finally, we compute embedding vectors of each individual node of the original graph by **combining the embeddings** of all the partitions that contain this node. The detail follows.

3.1 Hierarchy construction

We use a graph partitioning algorithm to compute a partition $\mathcal{P}$ of the set of nodes $\mathcal{V}$ of a graph $G = (\mathcal{V}, \mathcal{E})$, i.e., to compute a set of nonempty subsets of $\mathcal{V}$ such that every node $v \in \mathcal{V}$ belongs to exactly one of these subsets. Then, for each set of nodes $S \in \mathcal{P}$,

we build the subgraph $G[S]$ induced in G by S and repeat the partitioning process on $G[S]$ for every set S. This induced subgraph construction and partitioning is repeated recursively until the partitioning algorithm gives a trivial singleton partition that cannot be further decomposed. This happens for instance if the considered graph is a single node or a clique. In the end, when a single set of nodes is obtained, we create a partition of singletons $\{v\}$ for each node v in that set. This procedure is general and any graph partitioning algorithm could be used. However a fast and reliable algorithm must be used to ensure the quality and scalability and we therefore use here the *Louvain* algorithm [5].

This recursive procedure can be represented as a tree, which we depict in Figure 1 (left for a graphical representation of the nested partitions and middle for the corresponding tree). The constructed tree has the following properties: (i) each tree-node contains a subset of nodes of the input graph, (ii) the root node contains all nodes in the graph, (iii) the children of a given tree-node contain the sets of nodes corresponding to the partition of the subgraph induced by the set of nodes in the tree-node and (iv) each leaf contains a single node and there is a single leaf for every node of the original graph.

We detail this procedure in Algorithm 1. The procedure is recursive and calls itself in line 10. The input consists in the graph of interest and a partition function, called in line 3. We used the Louvain algorithm to partition a graph. The output consists in the hierarchical tree.

Algorithm 1 Hierarchy construction

```
1:  RECPART(G)
2:  function RECPART(G)
3:      C ← PARTITION(G)        ▷ We use Louvain for partitioning
4:      if C contains a single set of nodes S then
5:          for each node v in S do
6:              output leaf {v}
7:      else
8:          for each cluster S in C do
9:              output internal tree-node S
10:             RECPART(G[S])
```

3.2 Generating level specific embedding vectors

Once the hierachy is obtained, we compute an embedding vector for each one of the tree-nodes except the root node. We propose two embedding techniques.

(1) **Standard embedding.** Given any tree-node, including the root but except the leafs, we construct a weighted undirected meta-graph, where the nodes are the children of the considered tree-node and the weighted-edge between two children S_1 and S_2 is given by:

$$w_{S_1S_2} = \frac{|E_{S_1S_2}|}{|S_1| \cdot |S_2|}.$$

Here $E_{S_1S_2}$ denotes the set of edges between the sets of nodes S_1 and S_2 in the original graph. The normalisation $\frac{1}{|S_1| \cdot |S_2|}$ allows to obtained more relevant weights when the sets S are of different sizes (two large sets of nodes may have a large number of edges between them compared to two small sets). We note that other weight functions can be used, we defer this study to future work. Once the meta-graphs are obtained, we apply a standard graph embedding algorithm (for instance, *node2vec* [20] or *DeepWalk* [34]) to obtain an embedding vector in dimension d of each one of these meta-graphs independently.

(2) **Stochastic embedding.** For each tree-node, except the root, we simply generate a random vector of dimension d from the standard normal distribution $Normal(0, 1)$ with zero mean and unit variance. If d is large enough, then any two random vectors will be roughly at the same distance (curse of dimensionality) which is a good property for what we do next. Interestingly, here, we somehow leverage the curse of dimensionality.

In the case of Standard embedding, we note that the size of the meta-graphs are much smaller compared to the size of the original graph G, we can thus learn node embeddings of high quality in a very short time. We also note that a given edge of the original graph intervenes only in a single one of the obtained meta-graph (as a contribution to a weighted edge). This makes the overall time to compute all the embeddings no slower than the time that it would require to compute the embedding of the original graph with the chosen standard embedding method if the chosen standard embedding method has a linear running time. And it makes it much faster if it has running time slower than linear, say a quadratic running time.

In the case of Stochastic embedding, we observe that the embedding step does not directly depend on the structure of the input graph and it only depends on the tree. The input graph can thus be omitted as input for that step, which takes as input the tree only. This step is thus extremely fast as it only consists in generating a random vector for each tree-node.

3.3 Combining embeddings at different levels

Finally, we compute the embedding vectors $\mathbf{y_v}$ for each node $v \in G$ such that (i) the pairwise neighborhood relationship is preserved and (ii) similar nodes are close to one-another in the embedding space.

We assume that the leaf $\{v\}$ (where v is the node of interest) of the tree is at depth h. We denote y_v^t for $1 \leq t \leq h$ the embedding vectors of the h tree-nodes that lie on the path from the root the the leaf $\{v\}$ (excluding the root, but including the leaf).

We combine these h different embedding vector components in order to obtain the final embedding vector y_v of node v. For that purpose, we introduce a parameter α ($0 < \alpha < 1$) that regulates the weighting of the embedding vector components at different hierarchical levels t. We compute the final embedding vector as follows:

$$\mathbf{y_v} = \sum_{t=1}^{h} \alpha^{t-1} \mathbf{y_v^t} \tag{1}$$

Equation 1 ensures that the contribution of the vector components $\mathbf{y_v^t}$ gradually diminishes with increase in t as we move down the hierarchy. Hence, two nodes which belong to the same community very low down in the hierarchy will be placed extremely close to each other in the embedding space, compared to two nodes

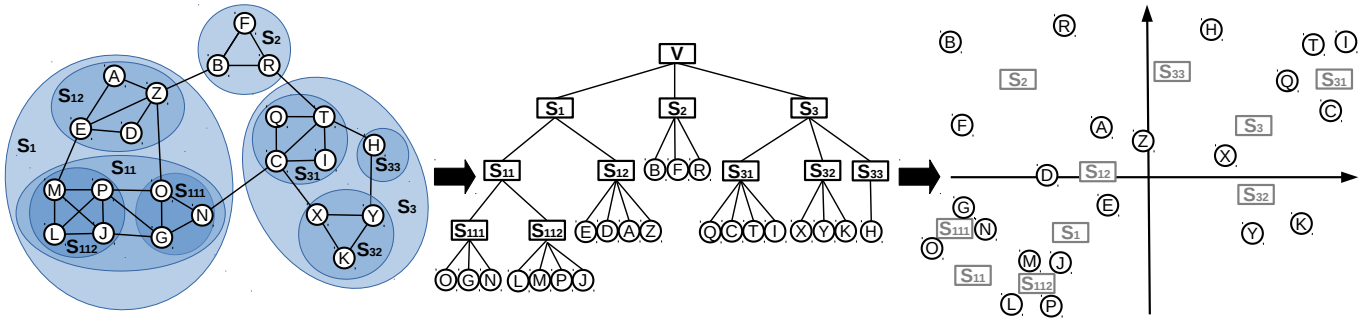


Figure 1: From graph to hierarchical tree to embedding

that are part of the same community at a top level of the hierarchy but belong to different communities lower down in the hierarchy. Nevertheless, in the second case too, the embedding vectors will be placed relatively close. This helps to preserve the neighborhood relationship between nodes in the embedding space, as well as place similar nodes (part of the same community) relatively close in the embedding space.

We illustrate this step of converting a hierarchy into an embedding in Figure 1 (middle and right). For illustration purposes, we set the number of dimensions to 2 and represent the non-leaf tree-nodes in shaded grey at the coordinates given by Equation 1, but where the sum is truncated at the depth of the tree-node.

4 EXPERIMENTAL SETUP

In this section, we describe the datasets and the state-of-the-art algorithms used for evaluating the proposed *LouvainNE* algorithm.

4.1 Datasets

We use datasets of different scales for our experiments, see Table 1 for detailed statistics. We consider *Blogcatalog* as the moderate-scale and *Youtube* and *Flickr* as the large-scale datasets. We will consider larger graphs in Section 7.

(a) Blogcatalog (BC): This dataset contains the network of social relations [1] between bloggers on the *Blog catalog* website with labels reflecting their categories of interest (say technology).

(b) Youtube dataset: This is a social network [2] consisting of all user-to-user links in the *Youtube* video sharing website. The labels denote groups that are subscribed to by users on *Youtube* such as gaming.

(c) Flickr dataset: This is a network dataset [2] where nodes are images shared on Flickr. Edges are formed between images sharing common metadata in Flickr such as same location, submitted to same gallery, sharing common tags, images taken by friends, labels such as animal etc.

Dataset	# Nodes	# Edges	# Labels
Blogcatalog	10312	333983	39
Youtube	1138499	2990443	47
Flickr	1715255	15555042	195

Table 1: Dataset statistics

4.2 Competing algorithms

We consider the following recent state-of-the-art methods for evaluating the performance of *LouvainNE*:

(a) HARP: This is a meta-strategy [9] for embedding graphs preserving higher-order structural features. It employs a hierarchical embedding approach where it first coarsens the graph using edge & star collapsing, followed by applying standard embedding to learn representations on the coarsest graph. Finally, it refines the embeddings from the coarsest to the finest graph. We used the implementation of the authors in [9] with the default parameters: the walk length= 10, number of walks= 40 and window size= 10; *node2vec* is used to embed the coarsest graph with the return parameter p and in-out parameter q, selected by grid search over the values $\{0.25, 0.5, 1, 2, 4\}$.

(b) MILE: [28] developed a framework similar to [9] where it scales up the performance of a standard embedding method even for large graphs. The proposed methodology includes a hybrid matching strategy to coarsen the graph and a graph convolution network to compute the refined final embeddings of the original graph. We use the authors' implementation with default parameters: the number of coarsening levels= 2, learning rate= 0.001. We use *DeepWalk* as the base embedding method and *GCN* is used to refine embeddings with self-loop weight= 0.05.

(c) RandNE: [54] proposed a simple random projection based network embedding approach using an iterative projection procedure, that helps to learn embeddings for billion-scale networks. We used the default parameter settings recomanded by the authors: with $q = 1$, weights=$[1, 0.1]$ and using the adjacency matrix for network reconstruction. For classification, we have used the transition matrix with parameter values $q = 3$ and weights=$[1, 10^2, 10^4, 10^5]$.

(d) NetSMF: [35]relies on sparse matrix factorization and on spectral sparsification of a dense random-walk matrix polynomial. We have used the implementation provided by the authors with default parameter settings of context window size $T = 10$, $b = 1$ and number of samples $M = 10^3 \times T \times m$ where m is the number of edges in the graph.

(e) ProNE: [53] developed a fast and scalable network embedding framework, which relies on spectral propagation to enhance the quality of learned embeddings. We used the authors' implementation with default parameter settings where term number of the Chebyshev expansion $k = 10$, $\mu = 0.1$ and $\theta = 0.5$.

Dataset	HARP	RandNE	MILE	NetSMF	ProNE	LouvainNE
Blogcat.	0.733	0.712	0.702	0.725	**0.747**	0.683
Youtube	0.883	0.818	0.907	0.932	0.917	**0.939**
Flickr	0.806	0.797	0.899	0.902	**0.913**	0.908

Table 2: AUC scores for network reconstruction

Dataset	HARP	RandNE	MILE	NetSMF	ProNE	LouvainNE
Blogcat.	0.316	0.308	0.264	**0.334**	0.323	0.306
Youtube	0.305	0.303	0.304	**0.307**	0.296	**0.307**
Flickr	0.384	0.385	0.386	0.356	0.361	**0.389**

Table 3: Micro-F1 scores for node classification

(f) LouvainNE: Our proposed solution. Unless otherwise specified, we use the *Stochastic embedding* variant and set the value of the parameter $\alpha = 0.01$.

Unless otherwise specified, we have used the embedding dimension $d = 128$ for all the methods including *LouvainNE*.

5 EMBEDDING QUALITY EVALUATION

In this section, we evaluate the quality of the learned node representations obtained from *LouvainNE* against the one obtained from the competing algorithms based on standard downstream tasks.

5.1 Network reconstruction

In this task, the learned embedding vectors are expected to well reconstruct the graph [33]. A good network embedding algorithm should ensure that the embedding vectors can preserve the original network topology, such that they can be used to reconstruct the network. Precisely, we aim to predict the links in the original network by ranking the node pairs based on the similarity of their learned embedding vectors. The larger the similarity between embedding vectors of a node pair, the higher is the probability for the node pair to be connected by a link.

Evaluation procedure. We use the following two standard evaluation metrics for the task of network reconstruction:

(a) precision@K: This metric [44] is measured as the fraction of node pairs in the top-K ranked pairs that are connected by an edge. In order to rank the node pairs, we compute the Euclidean distance between embedding vectors of all node pairs and rank them in increasing order of Euclidean distance (smaller Euclidean distance denotes larger similarity). For computing *precision@k* for network reconstruction, we rely on the following two ways depending on graph size:

1. Classical evaluation: For a graph G of size N, we compute the Euclidean distance for all $\binom{N}{2}$ pairs in the graph and sort these pairs in increasing order of Euclidean distance. Then we compute *precision@K* for top-K pairs over the entire ranked list of $\binom{N}{2}$ pairs. We follow this method to compute *precision@K* only for the moderate-scale network of *Blogcatalog*.

2. Scalable evaluation: The classical evaluation becomes infeasible for the large-scale datasets of *Youtube* and *Flickr* since the total number of node pairs in these networks are of the order of 10^{13}. Hence, we perform graph sampling to reduce the number of nodes and edges in the network and perform network reconstruction considering only the node pairs in the sampled graph. For graph sampling, we rely on the *forestfire* technique [22] to obtain a subgraph of the original graph. We fix the size of the sampled subgraph as the randomly chosen 1% nodes of the original graph. We then rank the node pairs in this sampled subgraph based on the computed Euclidean distance in increasing order. We compute *precision@K* considering only the node pairs belonging to this subgraph. We repeat this process 100 times and report the average *precision@K*. We apply this evaluation technique on the large-scale networks of *Youtube* and *Flickr*.

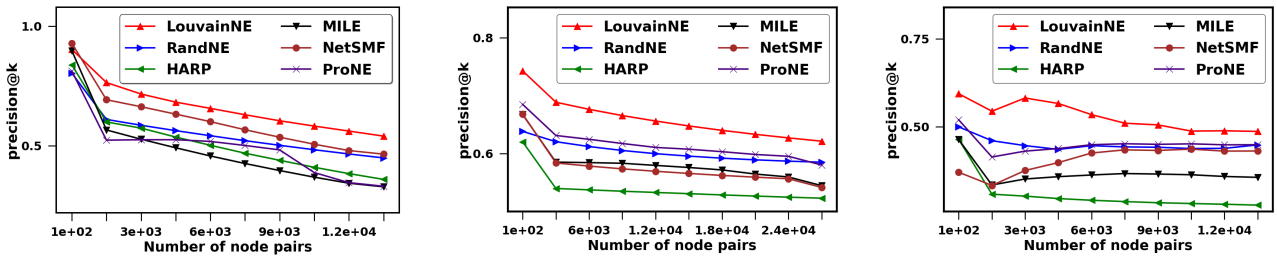
Dataset	HARP	RandNE	MILE	NetSMF	ProNE	LouvainNE
Blogcat.	**0.190**	0.182	0.150	0.179	0.160	0.167
Youtube	0.153	0.189	0.155	0.190	**0.192**	0.189
Flickr	0.251	0.249	0.251	0.248	0.250	**0.254**

Table 4: Macro-F1 scores for node classification

(b) Area Under the Curve (AUC): This metric [16] measures the probability that a randomly selected adjacent pair of nodes (positive sample) is ranked higher than a randomly selected non-adjacent pair of nodes (negative sample), in terms of similarity between their respective embedding vectors. As the exact *AUC* is resource intensive to compute due to the size of the considered networks, we compute an estimation of it by sampling. We sample an adjacent pair of nodes and a non-adjacent pair of nodes; we check whether the sampled adjacent pair is more similar than the sampled non-adjacent pair. Here the similarity between a pair of nodes is measured in terms of Euclidean distance between the learned embedding vectors of the two nodes (the lower the Euclidean distance, higher is the similarity). We repeat this procedure a large number of times (100 times the number of edges in the corresponding graph) and report the estimated *AUC*.

Results. To show the effectiveness of our proposed *LouvainNE* method on network reconstruction task, we plot the *precision@K* over increasing values of K for various datasets in Figure 2. We observe that *LouvainNE* outperforms the state-of-the-art methods across all datasets in terms of the *precision@K*, though the method *ProNE* performs comparably to *LouvainNE* on *Blogcatalog* and *Youtube* datasets. Moreover, in Table 2, we show that the proposed *LouvainNE* achieves good *AUC* scores outperforming all state-of-the-art methods except *ProNE*. Importantly, *LouvainNE* has comparable performance to *ProNE* in terms of *AUC* for the large-scale *Youtube* and *Flickr* datasets.

This superior performance of *LouvainNE* for the network reconstruction task is due to the fact that it relies on the construction of hierarchy of subgraphs based on successively applying community detection on the original graph (Section 3.1). This ensures that the majority of a node's neighbors will continue to belong to the same community as the given node lower down in the hierarchy. Subsequently, the embedding and combining steps ensure that a node is placed very close to its neighbors in the embedding space. Hence, connected node pairs will have highly similar embedding vectors compared to disconnected pairs, thereby preserving the pairwise neighborhood relationships in embedding space.



(a) Blogcatalog (b) Youtube (c) Flickr

Figure 2: precision@K for network reconstruction

5.2 Node classification

In this section, we evaluate the quality of embeddings generated by *LouvainNE* for node classification. Since a node can have one or multiple ground truth labels in all our used datasets, the resultant problem of classifying nodes is a multi-label classification problem.

Evaluation procedure. We apply the generated embedding vectors as features in a supervised learning framework to classify a node into the corresponding ground truth label(s). Specifically, we use the classifier chain technique [37] of performing multi-label classification, that takes into account label correlations by constructing a chain of binary classifiers, equal to the total number of ground truth labels. We implement a *Logistic Regression model* to train this multi-label classifier. We randomly sample 80% of the vertices as the training set and evaluate the classifier performance on the remaining vertices that form the test set (20%). We predict the label(s) of a node in the test set and report the *Micro-F1* and *Macro-F1* scores averaged over 100 iterations.

Results. In Tables 3 and 4, we observe that the proposed *LouvainNE* outperforms all state-of-the-art methods for the large-scale *Youtube* and *Flickr* datasets. However, for the moderate scale *Blogcatalog* dataset, *NetSMF* (*Micro-F1*) and *HARP* (*Macro-F1*) performs slightly better. This stems from the variations in the structural characteristics across the different networks, which impact the outcome of the multi-label node classification task. The superior performance of *LouvainNE* on the node classification task can be attributed to the graph embedding step of *LouvainNE* (Section 3.2) that assigns the same embedding vector to all nodes in a single community (reflecting nodes of similar labels) at a given hierarchy level. This ensures that the resultant embedding vectors of nodes belonging to the same community, obtained after the graph combining step (Section 3.3), will be placed very close to each other in the embedding space.

6 DRILLING DOWN *LouvainNE*

In this section, we explore the variants of *LouvainNE* implementations as well as investigate the impact of the model parameters on the performance of *LouvainNE*.

6.1 Embedding variants of *LouvainNE*

We compare the performance of variants of *LouvainNE* with different *level specific embedding* implementations in Section 3.2. We implement *node2vec* [20] and *DeepWalk* [34] as *Standard embedding* techniques as well as the *Stochastic embedding* and evaluate the performance on downstream tasks. First, we compare the *AUC* of network reconstruction across the model variants in Table 5. Next, we compare the performance of multi-label node classification in Table 6. We observe that all the variants of *LouvainNE* perform pretty uniformly across the tasks. Nevertheless, *DeepWalk* implementation of *Standard embedding* is the best performing model variant for the moderate scale *Blogcatalog* dataset, whereas *Stochastic embedding* is the best performing variant of *LouvainNE* for the large-scale *Youtube* dataset.

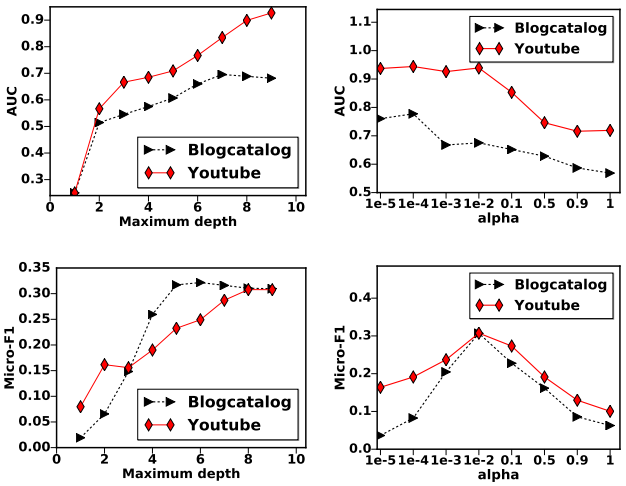


Figure 3: Parameter tuning of *LouvainNE*: (top-left) *AUC* as a function of the maximum depth; (top-right) *AUC* for network reconstruction as a function of α; (bottom-left) Micro-F1 as a function of the maximum depth; (bottom-right) Micro-F1 for node classification as a function of α.

6.2 Parameter tuning

Here we investigate the impact of (i) varying the maximum number of levels (the maximum depth) h_{max} in the hierarchy and (ii)

Dataset	Stochastic	Node2vec	DeepWalk
Blogcatalog	0.683	0.691	**0.695**
Youtube	**0.939**	0.923	0.925

Table 5: *AUC* for network Reconstruction of the variants of *LouvainNE*: Stochastic, Node2vec and DeepWalk.

Dataset	Stochastic	Node2vec	DeepWalk
Blogcatalog	0.306 (**0.167**)	0.311 (0.162)	**0.314** (0.164)
Youtube	**0.307** (**0.189**)	0.305 (0.186)	0.305 (0.187)

Table 6: Performance for node classification of *LouvainNE* (Stochastic, Node2vec and DeepWalk) in terms of *Micro-F1* (*Macro-F1* in parentheses)

varying the parameter α, while combining embeddings at different levels (Section 3.3), on the performance of *LouvainNE*. In that section, we only consider the stochastic variant of *LouvainNE*.

(i) Effect of varying maximum depth h_{max}: We vary maximum depth h_{max} from 1 to 9 (as 9 is the maximum number of levels encountered while building the hierarchy on *Youtube* and *Blogcatalog*). In that, we modify the stopping condition in the recursive Algorithm 1 by adding, line 4: "or if the depth is equal to h_{max}". We then generate the embedding vectors $\mathbf{y}_\mathbf{u}^\mathbf{t}$ using *Stochastic embedding* and perform the combining of embedding vector components. In Figure 3 (top-left) and Figure 3 (bottom-left), we compare the respective *AUC* for network reconstruction and *Micro-F1* for multi-label node classification using the embedding vectors obtained by *LouvainNE* over different values of h_{max} on *Blogcatalog* and *Youtube*. We observe that *AUC* is pretty low for $h_{max} = 1$ for both datasets, since the top level communities (due to their size) fail to discriminate (in terms of learned embedding vectors) between the connected and disconnected node pairs. Similar observation holds true in case of *Micro-F1* for node classification. As t increases, the performance improves gradually since the embedding vectors can better distinguish between neighbors and non-neighbors inside a community, as we go down in the hierarchy.

(b) Effect of varying α: We vary the value of α in the range $[10^{-5}, 1]$ and generate node embeddings for each value of α. We plot the *AUC* for network reconstruction using the embedding vectors obtained for different values of α for *Blogcatalog* and *Youtube* in Figure 3 (top-right). We observe that *AUC* is pretty high for low values of α and decreases as α increases. This reveals that lower values of α enable the learned embeddings to better preserve the neighborhood relationships. It is rather intuitive, as for very small values of α and given a node, the nodes closer to that node will be the ones sharing the same community at the lower hierachical level. We also plot the *Micro-F1* for multi-label node classification as a function of α in Figure 3 (bottom-right). Here we observe that, contrarily to the problem of network reconstruction, the lower value of α is not the better for classification. We indeed observe a cap shape where medium values of α, say $\alpha = 0.01$, perform much better than low or high values. This may be due to a trade-off between (i) the fact that low values of α are better as they put nodes sharing the same community at a low hierachical level relatively close to one-another and (ii) the smoothness of variation in the vectors required by the subsequent machine learning algorithm used to predict the label (logistic regression).

7 SCALABILITY EVALUATION

In this section, we push *LouvainNE* to its limits in order to demonstrate that it can deal with the largest publicly available real-world graphs. In order to evaluate our implementation of *LouvainNE* (with the *Stochastic embedding* variant), we gathered three real-world graphs with at least one billion edges (detailed statistics are presented in Table 7). We carried out these experiments on a Linux machine equipped with a processor Intel Xeon CPU E5- 2660 @ 2.60 GHz and with 512 GB of RAM DDR4 2133 MHz. We implemented LouvainNE efficiently in C[1].

We observed that none of the existing implementations of state-of-the-art methods (including *ProNE* that shows comparable performance to *LouvainNE* on downstream tasks) can compute an embedding of such large graphs using such a computer withing a reasonable amount of time. We thus re-implemented *RandNE* efficiently in C[2], in order to have a competitive baseline to compare against and evaluate the scalability performance of *LouvainNE*.

We present the results in Table 7 where we report the time to compute the hierarchy of *LouvainNE*, as well as the overall running time of *LouvainNE* and *RandNE* for the embedding dimensions $d = 16$, 128 and 512. We observe that when d is very small then *RandNE* is faster than *LouvainNE*, but for medium or large values of d *LouvainNE* is faster. For $d = 128$, *LouvainNE* is nearly 5 times faster than *RandNE* and for $d = 512$, *LouvainNE* is nearly 10 times faster.

Complexity analysis: The hierarchy construction step by applying *Louvain* [5] takes $O(M \cdot H)$ time, where M is the number of links (*Louvain* has a linear running time) and H is the height of the hierarchical tree (note that we observe in practice that H is very small, say around 10 or less). We note that this first step does not depend on the number of dimensions d of the embedding. The generation of level specific embedding vectors of dimension d and combining embeddings take $O(d \cdot N)$ time (where N is the number of nodes), leading to an overall time complexity of $O(M \cdot H + d \cdot N)$. The memory complexity is in $O(M \cdot H)$ in the worst case (we store the input graph (in an adjacency array datastructure), as well as a subgraph for each level of the hierarchy).

The time complexity of *RandNE* is in $\Theta(N \cdot d^2 + M \cdot d)$ [54]. *RandNE* is thus significantly slow when d is large on large graphs (both terms $N \cdot d^2$ and $M \cdot d$ are problematic). This is demonstrated experimentally in Table 7.

We also note that, the construction of the hierarchy takes a significant part of the overall running time of *LouvainNE* (say, 50% for $d = 512$). In case of competing hierarchical approaches such as *HARP* and *MILE*, either the graph coarsening step is complicated (for *HARP*) or the graph combining step based on *GCN* is slow (for *MILE*). This results in significantly increasing the overall running time, which makes these competing approaches unsuitable for embedding large-scale networks.

[1]Publicly available implementation: https://github.com/maxdan94/LouvainNE

[2]Publicly available implementation: https://github.com/maxdan94/RandNE. We do not perform the Gram-Schmidt process to obtain orthogonal projections as it takes important additional time and memory. We use $q = 2$ in the experiments.

				$d = 16$		$d = 128$		$d = 512$	
Network	**# Nodes**	**# Edges**	**Hierarchy**	**LouvainNE**	**RandNE**	**LouvainNE**	**RandNE**	**LouvainNE**	**RandNE**
Twitter09 [8]	5.2×10^7	1.6×10^9	1h21m	1h25m21s	**48m27s**	**1h57m**	5h49m	**3h47m**	22h59m
Friendster [27]	1.2×10^8	1.8×10^9	2h31m	2h37m	**1h31m**	**3h17m**	11h26m	**5h36m**	45h24m
MOLIERE [40]	3.0×10^7	3.3×10^9	4h44m	4h46m	**2h25m**	**5h5m**	17h57m	**6h10m**	71h13m

Table 7: Running time for the hierarchy construction step of *LouvainNE* and overall running time of *LouvainNE* and *RandNE*

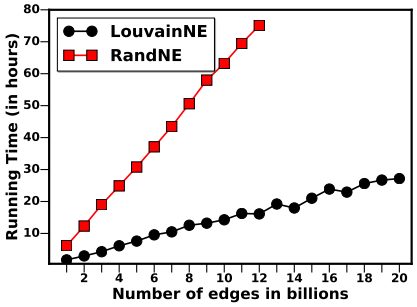


Figure 4: Linear time complexity of *LouvainNE* and *RandNE*

Finally, we evaluate the running time of *LouvainNE* as a function of the number of edges M. In order to do so, we relied on a publicly available Twitter graph (users and follow links) crawled in 2012 [18]. We ignore edge orientations since this is a directed graph. This graph has around 500 million users and the undirected version has 20 billion edges. We sampled subgraphs of different sizes in terms of number of edges and we measure the running time of *LouvainNE* for $d = 128$ on each of these subgraphs comparing it against *RandNE*. We present our results in Fig. 4 where we observe that the running time of *LouvainNE* and *RandNE* scales linearly with the number of edges, but that *LouvainNE* is about 5 times faster than *RandNE*.

We conclude that our implementation of *LouvainNE* is faster than our implementation of *RandNE* for medium values of d and much faster for large values of d. However, we note that *RandNE* (without the Gram-Schmidt process to obtain orthogonal projections) is embarrassingly parallel and offers a good degree of parallelism. *LouvainNE* can also be made parallel, but we defer the study of its parallelism to future work.

8 RELATED WORK

Earliest approaches on network embedding [3, 39, 42] relied on various dimensionality reduction techniques for mapping nodes to a low-dimensional vector space from high-dimensional adjacency matrix such that adjacent nodes get placed close to each other in the embedding space. However, these methods, with time complexity in $\approx O(|V|^2)$, were only suitable to process small graphs.

Recently, most of the existing state-of-the-art network embedding approaches that are based on random walks [13, 20, 24, 26, 34, 38, 55] are inspired by the word2vec model [30] for learning representations of words. These methods rely on generating large number of walks for training the model, thus considerably slowing down the learning process. On the other hand, network embedding approaches employing matrix factorization techniques [6, 14, 23, 33, 36, 49] are inherently slow since they learn node representations using a dense objective matrix, often with millions of rows and columns making them both memory and computationally infeasible. In addition, there also exist some embedding methods that learn node representations on graphs mainly capturing local neighborhood information using the knowledge of vertex-vertex connections [41, 45, 48] or capturing the non-linearity of graphs by employing deep learning methods that needs to optimize huge number of parameters [7, 12, 21, 43, 44, 47]. Besides these, algorithms have been developed for embedding nodes in heterogeneous networks [13], signed networks [4, 25, 46, 52] and attributed networks [46, 50]. However, all these methods mainly capture the local structural relationships as well as suffer from high computational complexity.

More recently, there have been few works on hierarchical representation learning [15, 29]. Probably, the closest approach to ours are *HARP* [9] and *MILE* [28]. However, these approaches relies on coarsening the original graph by repeatedly aggregating nodes based on structural similarity followed by embedding the coarsest graph using a state-of-the-art method (*Deepwalk*, *node2vec* etc.). The embeddings of the coarsest graph are then successively refined to get the final embeddings of the original graph. Our approach differs in the way we construct the hierarchy of communities as well as the embedding and combining steps to obtain the final embeddings from this hierarchy.

Few works have relied on spectral graph sparsification techniques [35, 53] and random projection [54] for obtaining scalable network embedding. However, none of these approaches except (our re-implementation of) *RandNE* [54] scales to networks with tens of billions of edges.

9 CONCLUSION

We leverage the notion of community structure to develop *LouvainNE*, a scalable graph embedding framework relying on three steps: (a) constructing a hierarchy of subgraphs (b) computing level specific embeddings for each subgraph in the hierarchy and (c) combining these level specific embeddings. We use the *Louvain* algorithm to recursively partition the graph and obtain the hierarchy.

We have shown that *LouvainNE* leads to high-quality embeddings, for downstream graph mining tasks, relatively to the state-of-the-art. We have shown that our implementation of *LouvainNE* is able to process graphs with tens of billions of edges, that its running time scales linearly with the number of edges and that it is much faster than its fastest competitor (our re-implementation of *RandNE*).

For future work, we would like to study the parallelism of *LouvainNE*, use partitioning algorithms other than *Louvain*, as well as

investigate other level specific embedding methods and combining steps. Can we use the hierarchy directly as input to machine learning algorithms?

REFERENCES

[1] [n.d.]. Dataset : BlogCatalog3. http://socialcomputing.asu.edu/datasets/BlogCatalog3. [Online; accessed 17-Aug-2019].

[2] [n.d.]. IMC 2007 Data Sets. http://socialnetworks.mpi-sws.org/data-imc2007.html. [Online; accessed 17-Aug-2019].

[3] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. NIPS, 585–591.

[4] Ayan Kumar Bhowmick, Koushik Meneni, and Bivas Mitra. 2019. On the Network Embedding in Sparse Signed Networks. PAKDD, 94–106.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. CIKM, 891–900.

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. AAAI, 1145–1152.

[8] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. ICWSM.

[9] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. AAAI, 2127–2134.

[10] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical review E* 70, 6 (2004), 066111.

[11] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. NIPS, 3844–3852.

[13] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. KDD, 135–144.

[14] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Spectral graph wavelets for structural role similarity in networks. (2018).

[15] Lun Du, Zhicong Lu, Yun Wang, Guojie Song, Yiming Wang, and Wei Chen. 2018. Galaxy Network Embedding: A Hierarchical Community Structure Preserving Approach. IJCAI, 2079–2085.

[16] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[17] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[18] Maksym Gabielkov, Ashwin Rao, and Arnaud Legout. 2014. Studying Social Networks at Scale: Macroscopic Anatomy of the Twitter Social Graph. In *ACM Sigmetrics 2014*. Austin, United States. https://hal.inria.fr/hal-00948889

[19] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[20] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. KDD, 855–864.

[21] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. NIPS, 1024–1034.

[22] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* (2013).

[23] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. WSDM, 731–739.

[24] Mohammad Mehdi Keikha, Maseud Rahgozar, and Masoud Asadpour. 2018. Community aware random walk for network embedding. *Knowledge-Based Systems* 148 (2018), 47–54.

[25] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. SIDE: Representation Learning in Signed Directed Networks. WWW, 509–518.

[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[27] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[28] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2018. MILE: A Multi-Level Framework for Scalable Graph Embedding. *arXiv preprint arXiv:1802.09612* (2018).

[29] Jianxin Ma, Peng Cui, Xiao Wang, and Wenwu Zhu. 2018. Hierarchical Taxonomy Aware Network Embedding. KDD, 1920–1929.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. NIPS, 3111–3119.

[31] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[32] Günce Keziban Orman, Vincent Labatut, and Hocine Cherifi. 2011. Qualitative comparison of community detection algorithms. In *International conference on digital information and communication technology and its applications*. Springer, 265–279.

[33] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. KDD, 1105–1114.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. KDD, 701–710.

[35] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. WWW, 1509–1520.

[36] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. WSDM, 459–467.

[37] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2009. Classifier chains for multi-label classification. ECML PKDD, 254–269.

[38] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. KDD, 385–394.

[39] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290 (2000), 2323–2326.

[40] Justin Sybrandt, Michael Shtutman, and Ilya Safro. 2017. Moliere: Automatic biomedical hypothesis generation system. KDD, 1633–1642.

[41] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. WWW, 1067–1077.

[42] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290 (2000), 2319–2323.

[43] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. 11 (2010), 3371–3408.

[44] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. KDD, 1225–1234.

[45] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. AAAI, 2508–2515.

[46] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. CIKM, 137–146.

[47] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. SIAM, 327–335.

[48] Suhang Wang, Jiliang Tang, Fred Morstatter, and Huan Liu. 2016. Paired restricted boltzmann machine for linked data. CIKM, 1753–1762.

[49] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. AAAI, 203–209.

[50] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. IJCAI, 2111–2117.

[51] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.

[52] Shuhan Yuan, Xintao Wu, and Yang Xiang. 2017. Sne: signed network embedding. PAKDD, 183–195.

[53] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. 2019. ProNE: Fast and Scalable Network Representation Learning. IJCAI.

[54] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale Network Embedding with Iterative Random Projection. ICDM, 787–796.

[55] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. AAAI, 2942–2948.