# A Comprehensive Comparison of Unsupervised Network Representation Learning Methods

Megha Khosla, Avishek Anand, Vinay Setty



**Abstract**—There has been appreciable progress in unsupervised network representation learning (UNRL) approaches over graphs recently with flexible random-walk approaches, new optimization objectives and deep architectures. However, there is no common ground for systematic comparison of embeddings to understand their behavior for different graphs and tasks. In this paper we theoretically group different approaches under a unifying framework and empirically investigate the effectiveness of different network representation methods. In particular, we argue that most of the UNRL approaches either explicitly or implicit model and exploit context information of a node. Consequently, we propose a framework that casts a variety of approaches – random walk based, matrix factorization and deep learning based – into a unified context-based optimization function. We systematically group the methods based on their similarities and differences. We study the differences among these methods in detail which we later use to explain their performance differences (on downstream tasks).

We conduct a large-scale empirical study considering 9 popular and recent UNRL techniques and 11 real-world datasets with varying structural properties and two common tasks – node classification and link prediction. We find that there is no single method that is a clear winner and that the choice of a suitable method is dictated by certain properties of the embedding methods, task and structural properties of the underlying graph. In addition we also report the common pitfalls in evaluation of UNRL methods and come up with suggestions for experimental design and interpretation of results.

## 1 Introduction

There has been a resurgence of unsupervised methods for network embeddings for graphs in the last five years [21], [31], [19]. This is primarily due to improvements in modelling and optimization techniques using neural network based approaches, and their utility in a wide variety of prediction and social network analysis tasks such as link prediction [12], vertex classification [6], recommendations [36], knowledge-base completion [14] etc.

**Lack of a comprehensive study.** In spite of their success, there is a lack of an in depth systematic study of the differences between various embedding approaches. Prior works have mainly focused on studying similarities between different embedding approaches using unifying theoretical frameworks [13], [22]. As we show in our experiments (cf. Section 7), evaluation studies accompanying each new approaches mostly focus on the experimental regimes where they perform well and omit the scenarios where they might perform suboptimally. Comprehensive large-scale studies comparing these approaches under different experimental conditions are missing altogether to the best of our knowledge. Thus a fundamental practical question remains largely unanswered: From a comparative standpoint, *Which unsupervised representation network representation learning (UNRL) approaches for nodes are most effective for different graph types and task types?* This begs for a common theoretical and experimental ground for systematic comparison of embeddings that are trained in an unsupervised manner.

In this paper, we fill this gap by first proposing a common theoretical framework that focuses on the differences between the various UNRL approaches. Secondly, we perform a comprehensive experimental evaluation with 9 embedding methods (cf. Table 1) of different methods – random walks, edge modeling, matrix factorization and deep learning – that includes some of the earliest approaches for learning network representations to the latest deep learning based approaches and 11 datasets (cf. Table 2). We note that in the scope of this work we consider only unsupervised methods in transductive scenario.

**Unifying framework.** Our common theoretical framework for understanding UNRL approaches is inspired by the observation that most of the unsupervised learning approaches explicitly or implicitly model the *context* of a vertex. That is vertices in similar contexts are embedded closer to each other and different context farther away. A vertex can be in the context of another if they are in the immediate neighborhood, reachable by truncated random walks, or they are in the same community/cluster etc. We posit that the objective functions of various approaches can be generalized into a common objective function based on their context modelling decisions. First, casting the context as a context graph helps us to identify differentiating properties between UNRL approaches. Secondly, we are also able to reason about the differences between methods when it comes to exploiting this context information. In this paper we show that we can cast a wide variety of approaches– e.g. that employ *random walks* [21], [6], [39], [31], *neighborhood modelling* [28], *matrix factorization* [22], [19] and *deep learning* [7], [33] – into our unified context-based

- *M. Khosla, and A. Anand are with the L3S Research center, Leibniz Universität, Hannover, Germany. E-mail: {khosla,anand}@L3S.de*
- *V. Setty is with the Department of Electrical Engineering and Computer Science, University of Stavanger, Norway. E-mail: vsetty@acm.org*

optimization function.

**Comprehensive Experimental Evaluation.** In our evaluation of UNRL methods we investigate the *conceptual differences between the embedding approaches that result in performance differences on downstream tasks.*

First, using graphs with diverse structural characteristics we argue about the utility of several approaches. We carefully chose 11 large network datasets (5 undirected and 6 directed) with diverse properties that are popular from social networks, citation networks, and collaboration networks domains. With focus on reproducibility and large scale analysis, we chose at least one dataset used in each of the original papers and try to be as close to the authors original experimental setup as possible on two most popular tasks – node classification and link prediction.

Second, in addition to performing a large-scale study with a large number of baselines we also find limitations in the experimental setup of earlier approaches. In particular, for evaluating link prediction performance in case of directed graphs most of the earlier works only check for the existence of an edge between a pair of nodes and ignore directionality of the edge.

Finally, we also question the claimed superiority of various embedding methods in the node classification task, wherein naïve (yet effective) baselines are not considered. We surprisingly find that for several of the datasets comparable of even better performance is achieved by our naïve baseline.

**Key findings.** Our study does not propose a winner or a loser but highlights the strengths and weaknesses of approaches under different graph and task characteristics. Some of the key finding of our study are as follows:

- Context plays a major role in the performance of the link prediction task. For directed graphs, using both node and context information is crucial for better performance. On the contrary for undirected graphs, we surprisingly find that explicitly exploiting context has no perceivable advantage.
- Structural properties of the input graph indeed have an effect on the performance of the UNRL embeddings. Graph properties like *clustering coefficient* and *reciprocity* of graphs greatly impact the performance of UNRL approaches.
- Lastly we question the claimed superiority of various embedding methods on node classification task, wherein for several of the datasets comparable or better performance is achieved by a naïve baseline using just the immediate neighbor information in the graph space.

We believe that this work will provide a common ground to understand a class of unsupervised node embedding methods from both conceptual and empirical perspectives that is still missing in the present literature. Our results can serve as guideline for researchers and industry practitioners in the choice of the embedding methods for an input graph and a specific task.

## 2 Related Work

With increasing number of unsupervised embedding methods, it has become extremely difficult to objectively compare and choose appropriate methods for a given dataset.

Several existing surveys focus on categorization of various network embedding techniques with respect to the methodology such as random walks, matrix factorization and edge modeling etc [2], [3]. But they fail to provide any unifying framework to compare and gain deeper understanding of various methods. Other works which do provide a common framework only focus on demonstrating the equivalence of various methods to matrix factorization [13], [22], [35] but do not consider the differences between the methods and their impact on task performance. Several other surveys [8], [34] consider a wide range of unsupervised and semi-supervised embedding methods without any empirical comparison. More importantly, the theoretical categorization and comparison in these surveys does not directly correspond with explaining why some methods are superior to other methods under certain circumstances.

Surveys which include empirical comparison [5], [38] focus only on effect of training data size for various tasks or the effect of varying hyperparameters on task performance. In [24], the authors consider semi-supervised node classification and demonstrate the effect of different train/validation/test splits and hyperparameters on the performance of several graph neural network models.

In summary, to the best of our knowledge, we are the first to compare UNRL methods using (1) a common unifying theoretical framework based on the concept of context, (2) structural properties of the underlying graph and (3) large-scale experiments to demonstrate the properties of various methods observed by the theoretical framework.

We remark that the scope of this work is limited to unsupervised, transductive methods and non-attributed graphs. We include most popular representative methods which follow our common objective function in our study. We omit other unsupervised methods like variational graph autoencoder [10] which is a generative model, ARGA [20] and NetRA [37] which combine graph convolution networks (GCNs) with graph adversarial networks (GANs), Graph2Gauss [1] and DVNE [40] which model uncertainty in network representation. Semi-supervised methods like graph attention networks [32] are also not considered in our study. None of these follow our common framework and hence deserve a separate study.

## 3 Theoretical Framework and Research Questions

In this paper we first build a common theoretical framework in which we can conveniently cast the objective functions of the *random walk*, *matrix factorization* and *deep learning* based Unsupervised Network Representation Learning (UNRL) methods. We argue that most of the unsupervised learning approaches explicitly or implicitly model the *context* of a vertex by embedding vertices in similar contexts closer and different contexts farther away from each other. For example, a vertex can be in the context of another if they are in the immediate neighborhood, reachable by a truncated random walks, in the same community/cluster etc. Hence, differences in the context definition would entail different embeddings. In what follows, we introduce the notion of a *context graph* and introduce a common objective function

into which all of the methods under investigation can be mapped.

Formally, given a graph $G = (V, E)$ we are interested in learning low dimensional representations of each node $v \in V$ such that similar nodes in $V$ are embedded closer. These representations are then used for downstream tasks for example predicting missing links in $G$ or in node classification task where the goal is to predict missing node labels. Note that as we do not consider additional node or edge attributes in this work, the similarity information among the vertices is inferred from the topological structure of $G$.

### 3.1 Context Graph

In order to understand how various methods differ in their definitions and treatment of similarity, we begin by constructing an auxiliary *directed* and *weighted* graph $\mathcal{C}$ from $G$ where $\mathcal{C} = (V, E')$ such that higher the weight of edge $(u, v) \in E'$ higher the similarity among nodes $u$ and $v$. We call $\mathcal{C}$ as the *context graph* and for each edge $(u, v) \in E'$, $v$ is the called the *context* of node $u$. Let $C$ denote the corresponding adjacency matrix of $\mathcal{C}$ with $c_{i,j}$ denoting $(i, j)th$ element in $C$.

We denote the d-dimensional node and context representations of nodes in $\mathcal{C}$ by $\Phi \in \mathbb{R}^{|V|} \times \mathbb{R}^d$ and $\theta \in \mathbb{R}^{|V|} \times \mathbb{R}^d$ respectively. We are then interested in learning $\Phi$ and $\theta$ while minimizing the following loss

$$\mathcal{J} = -\sum_{i,j} c_{i,j} \cdot f_1(\Phi_i, \theta_j), \tag{1}$$

where $f_1$ is monotonically increasing in $\Phi_i \cdot \theta_j$. Note that by minimizing (1) a vertex will be embedded closer to its context; and therefore vertices sharing same context will be embedded closer by transitivity. In this work we consider DEEPWALK [21], NODE2VEC [6], APP [39], LINE-2 [28], SDNE [33] employing the above form of the loss function. We also include two methods which use matrix factorization objectives–HOPE [19] and NETMF [22], based on their equivalence to the above objective demonstrated in [13].

In some works such as VERSE [31], LINE-1 [28], unsupervised GRAPHSAGE [7] the loss function ignores the context, i.e., learning only $\Phi_i$ by minimizing the following loss function.

$$\mathcal{J}' = -\sum_{i,j} c_{i,j} \cdot f_1(\Phi_i, \Phi_j). \tag{2}$$

While many of the considered methods can be explained under a unified framework based on their similarities in their objective functions (as also done by previous works [22], [13]), we are interested in understanding their differences due to their modelling decisions.

In the rest of this section, we elaborate these differences based on (1) how they *define context*, (2) how they *exploit context* and (3) how they *optimize their objectives*. In Section 3.2 we list 4 different schemes of defining context with each scheme having examples of two embedding methods. For these chosen methods, we then focus on exploitation of *context* in Section 3.3. We then elaborate on various optimization approaches used by these methods in Section 3.4. The main discussed differences and similarities among the studied methods is also summarized in Table 1. Finally, in Section 3.5, we formulate a set of research questions which are answered based on the differences elaborated below and the experimental results in Section 7.

**Notations.** Before we elaborate the above mentioned differences with respect to various methods, we define the notations used in this paper. Let $A$ and $D$ denote the adjacency and degree matrix of $G$ respectively. Let $\mathbf{P} = D^{-1}A$ be the transition matrix. Let $d_v$ denotes the degree of a vertex $v \in V$ in $G$. For directed graphs, we denote $d_v^+$ as the indegree and $d_v^-$ denotes the outdegree of $v$. We represent $vol(G)$ as the total sum of all edge weights in $G$.

### 3.2 Different Schemes of Defining Context

In this section, we compare various approaches with respect to the different ways in which the context graph defined in Section 3.1. The simplest possible context matrix which can be used is the adjacency matrix itself, i.e., nodes sharing a link are similar to each other. In this case the context graph is same as the original graph $G$. While some methods directly use similarity notions like Katz similarity [9] (e.g., HOPE), Personalized PageRank (e.g., VERSE) to quantify similarity among vertices, other methods explore higher order neighborhoods via random walks and quantify similarity among nodes by their co-occurrence in these walks (e.g., DEEPWALK and NODE2VEC).

#### 3.2.1 Random Walk Based Context

In random walk based methods, the higher order neighborhoods are usually sampled to define the context graph. Roughly, a vertex pair $(u, v)$ co-occurring in a random walk will correspond to two directed edges in the context graph : $u \to v$ and $v \to u$. In the first edge $v$ serves as a context for $u$ while for the second edge $u$ is the context. We explain below more precisely the context graphs of two popular methods under this category, namely DEEPWALK and NODE2VEC.

**DEEPWALK, NODE2VEC .** These methods employ truncated random walks of length $T$ from each vertex $v \in V$ to create vertex sequence, say $W_v$. In particular, for each $i \in W_v$ and for each $j \in W[k - r : k + r]$ (r is the window size), $(i, j)$ forms an edge in the corresponding context graph. While DEEPWALK performs a uniform random walk, NODE2VEC follows a 2nd order random walk.

More specifically, from [22], for any pair of vertices $i, j \in V$ for DEEPWALK's walk lengths $T$ and window size $r$ we have

$$\frac{c_{i,j}}{\sum_{u,v} c_{u,v}} \xrightarrow{p} \frac{1}{2T} \sum_{r=1}^{T} \left( \frac{d_i}{vol(G)} \cdot (\mathbf{P}^r)_{i,j} + \frac{d_j}{vol(G)} \cdot (\mathbf{P}^r)_{j,i} \right) \tag{3}$$

On the other hand NODE2VEC first computes a second order transition probability to sample the next vertex in the walk as defined below.

$$\mathbf{P}_{u,v,w} = \frac{T_{u \to v \to w}}{\sum_w T_{u \to v \to w,}}$$

where

$$T_{u \to v \to w} = \begin{cases} \frac{1}{p}, \text{if } (u, v) \in E (v, w) \in E, u = w \\ 1, \text{if } (u, v) \in E (v, w) \in E, u \neq w, (u, w) \in E \\ \frac{1}{q}, \text{if } (u, v) \in E (v, w) \in E, u \neq w, (u, w) \notin E \\ 0, \text{otherwise} \end{cases}.$$

| Algorithm | Symmetric Context | Learnt Embeddings | Used Embeddings | Loss | Optimization |
|---|---|---|---|---|---|
| DEEPWALK | ✓ | $\Phi, \theta$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \theta_k)}$ | Hierarchical Softmax |
| NODE2VEC | ✓ | $\Phi, \theta$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \theta_k)}$ | Negative Sampling (NS) |
| APP | ✗ | $\Phi, \theta$ | $\Phi, \theta$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \theta_k)}$ | NS |
| VERSE | ✗ | $\Phi, \theta$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \Phi_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \Phi_k)}$ | NS |
| LINE-1 | ✓ | $\Phi, \theta$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{1}{1+\exp(-\Phi_i \cdot \Phi_j)}$ | NS |
| LINE-2 | ✗ | $\Phi, \theta$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \theta_k)}$ | NS |
| NETMF | ✗ | $\Phi, \theta$ | $\Phi$ | $\|\|C - \Phi \cdot \theta\|\|_F^2$ | Matrix Factorization (MF) |
| HOPE | ✗ | $\Phi, \theta$ | $\Phi, \theta$ | $\|\|C - \Phi \cdot \theta\|\|_F^2$ | MF |
| SDNE | ✗ | $\Phi$ | $\Phi$ | see Equation (11) | Deep Autoencoders |
| Unsupervised GRAPHSAGE | ✓ | $\Phi$ | $\Phi$ | $-\sum_{i,j} c_{i,j} \log \frac{\exp(\Phi_i \cdot \Phi_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \Phi_k)}$ | NS with Neighborhood Aggregation |

TABLE 1: A summary of *Network Representation Learning* algorithms with respect to Context and Optimization

Under assumptions of infinite length walks on undirected graphs, it can be shown that for NODE2VEC

$$\frac{c_{i,j}}{\sum_{u,v} c_{u,v}} \xrightarrow{p} \frac{1}{2T} \sum_{r=1}^{T} \sum_{k} \left( \mathbf{X}_{k,i} (\mathbf{P}_{\mathbf{k},\mathbf{i},\mathbf{j}})^r + \mathbf{X}_{k,j} (\mathbf{P}_{\mathbf{j},\mathbf{k},\mathbf{i}})^r \right), \tag{4}$$

where $\mathbf{X}$ represents a stationary distribution of the second order random walk. We observe that the respective context graphs have the following properties.

**Property 1.** *For* DEEPWALK *and* NODE2VEC*'s context graphs vertex and contexts are indistinguishable by construction, for example $c_{i,j}$ and $c_{j,i}$ will be identical even if the underlying graph $G$ is directed.*

**Property 2.** *The parameter $p$ as used by* NODE2VEC *will have no effect for a directed graph with zero reciprocity*[1] *as there are no back edges.*

**Property 3.** *For any triplets $u, v, w$ we have $\mathbf{P}_{v,w} = \mathbf{P}_{u,v,w}$ when $u, v, w$ does not form a triangle and there are no back edges, i.e., $u \neq w$. This implies that* NODE2VEC*'s biased walks might not give any additional advantage in case of graphs with low clustering coefficient and zero reciprocity.*

#### 3.2.2 Personalized PageRank Based Context

**APP and VERSE.** These methods use random walks with restarts to draw vertex pairs. In particular, every time a walk starts from a vertex chosen uniformly at random; a walk is continued with probability $1 - \alpha$ where $\alpha$ is the predefined restart probability. The first and last vertices of this walk forms a directed edge in the corresponding context graph with the first node of the walk being the source node. For any vertices $i, j \in V$ we compute the theoretical estimate of $\frac{c_{i,j}}{\sum_i c_{i,j}}$.

**Proposition 1.** *Let $i \in V$ be uniformly chosen as done in* APP*. Let $j \in V$ be such that the shortest hop distance between $i$ and $j$ be $h$. For some vertex $j \in V$, $j$ is the last vertex with probability $O(\frac{1}{|V|}(1-\alpha)^h \cdot \alpha \cdot (\mathbf{P}^h)_{i,j})$, i.e.,*

$$\frac{c_{i,j}}{\sum_{u,v} c_{u,v}} = O\left( \frac{1}{|V|} (1-\alpha)^h \cdot \alpha \cdot (\mathbf{P}^h)_{i,j} \right).$$

1. In a directed network, the reciprocity equals to the proportion of edges for which an edge in the opposite direction exists

The proof for above proposition is provided in the supplementary material. We observe that the context graph used by APP and VERSE is considerably different from that used by DEEPWALK and NODE2VEC. Note Equations (3) and (4) imply $c_{i,j} = c_{j,i}$ for DEEPWALK and NODE2VEC whereas this is not the case for APP and VERSE, where their values depend on the neighborhood structures of nodes as shown in Figure 1 where there is a higher probability of reaching from node $u$ to $v$ than vice-versa. We therefore have the following property.

**Property 4.** *For any $i, j \in V$, $c_{i,j}$ is not always equal to $c_{j,i}$, i.e., $C$ is not always a symmetric matrix or the similarity relation between vertices is not always symmetric.*

#### 3.2.3 Adjacency Based Context

**LINE and SDNE.** Both of these methods directly uses the given graph as its context graph, i.e., $C = A$. They aim to embed vertices closer which have either links between them (optimizing for first order proximity) or share common 1-hop neighborhood (optimizing for second order proximity). They specifically differ in their exact formulations of loss functions and optimization strategies which will be discussed in detail in Section 3.4. Corresponding to LINE, we study both of its variants : LINE-1 ( optimizing only first-order proximity) and LINE-2 (optimizing only second-order proximity). LINE-1+2 is obtained by normalizing and concatenating the embedding vectors from LINE-1 and LINE-2

**Special Case of Unsupervised GRAPHSAGE.** GRAPHSAGE uses a two layer deep neural architecture where in each layer $k$ a node $v \in V$ computes its representation $h^k$ as an aggregation of representations (from previous layer) of its neighbors, $\{h^{(k-1)(u)}, \forall u \in N(v)\}$. The parameters of aggregation functions are learnt using the loss function similar to DEEPWALK. In other words, GRAPHSAGE also optimizes for embedding vertices closer which are more similar with respect to the context matrix generated using Equation (3), where, an embedding vector of a node is a function of embedding vectors of its immediate neighbors. Intuitively this implies that nodes having links between them will be embedded closer. For GRAPHSAGE we report

the best results corresponding to one of its four aggregators (Mean, MeanPool, MaxPool and LSTM). In addition, we study GCN variant of GRAPHSAGE where the aggregator function is the graph convolution network. Note that we used the unsupervised and transductive variant of GRAPHSAGE for this work.

#### 3.2.4 Direct Matrix Based Context

**NETMF.** NETMF is derived from a theoretical analysis of DEEPWALK and directly factorizes the context matrix with $(i, j)$th element given by

$$c_{i,j} = \log\left(\frac{vol(G)}{kT}\left(\frac{1}{d_j}\cdot(\sum_{r=1}^{T}\mathbf{P}^r)_{i,j}\right)\right) \quad (5)$$

where $T, r, k$ are hyperparameters and correspond to walk length, window size and negative samples in DEEPWALK.

We remark here that while DEEPWALK explicitly encodes similarity between vertices as given by Equation (3), using the equivalence of SGNS[15] optimization to matrix factorization, [22] proposes that DEEPWALK implicitly factorizes the context matrix where each element given by Equation (5). Note that the focus of this work is not to validate/invalidate this connection but understand the kind of node similarities different methods try to encode in the latent representations of the nodes. DEEPWALK and NETMF are therefore not only different from their optimization techniques but also their respective context graphs representing similarities between vertices.

**HOPE.** This approach preserves the asymmetric role information of the nodes by approximating high-order proximity measures like Katz measure [9], Rooted PageRank [26] etc. We study the version of HOPE where it uses Katz similarity matrix as the context matrix as it also gives us a different type of context graph to compare with. For example, the context graph generated for Rooted PageRank is quite similar to the ones used by VERSE and APP. In a Katz similarity matrix, each entry $c_{i,j}$ is a weighted summation over the path set between two vertexes. More specifically,

$$c_{i,j} = \sum_{\ell=1}^{\infty}\beta^k(A^k)_{i,j},$$

where $\beta$ is a decay parameter and determines how fast the weight of the path decay with growing length.

### 3.3 Exploitation of Context

Methods differ in their learning and usage of context representations. While some methods completely ignore context and only learn node representations, other methods learn both node and context representations but only utilize node representations for the downstream task. There is yet another class of methods which in addition to learning two representations also use both of them for downstream tasks.

**DEEPWALK, NETMF, LINE-2 and NODE2VEC.** These methods learn both node and context representations but use only node representation for the downstream tasks.

**GRAPHSAGE, LINE-1, VERSE and SDNE.** All of these methods learn a single representation per node and ignore the context representation.

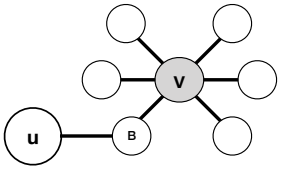


Fig. 1: Asymmetric local structures in Undirected Graphs

**APP and HOPE.** Both these methods learn two representations per node as well use both the representations for downstream tasks. They infact use the context representation to represent the node in its destination role if the original graph $G$ is directed.

**Difference between APP and VERSE.** APP and VERSE both perform random walks with restarts to compute their respective context graphs. As already discussed (cf. Property 4), the similarities encoded by the context graph in their case are not symmetric, yet VERSE ignores this asymmetries and attempts to encode the similarities between vertices in a single embedding space. This is quite contrary to its motivation of encoding Pensonalized PageRank (PPR) which is by construction asymmetric, i.e., $PPR(i, j)$ is not always equal to $PPR(j, i)$, where $PPR(i, j)$ represents the PPR of $i$ with respect to $j$ .

### 3.4 Differences in Optimization Methods

Optimization methods span from direct matrix factorization, deep autoencoders, negative sampling and neighborhood aggregation methods.

**Hierarchical softmax [17] and Negative Sampling [15].** DEEPWALK, NODE2VEC, LINE-2, APP models $f_1$ in Equation (1) as logarithm of probability for pair $(i, j)$ sharing an edge in the context graph, i.e.,

$$f_1(\Phi_i, \theta_j) = \log\frac{\exp(\Phi_i\cdot\theta_j)}{\sum_k\exp(\Phi_i\cdot\theta_k)} \quad (6)$$

VERSE uses exactly the same form of $f_1$ except that it uses only ignores context representation, i.e, it defines $f_1$ as

$$f_1^{verse}(\Phi_i, \Phi_j) = \log\frac{\exp(\Phi_i\cdot\Phi_j)}{\sum_k\exp(\Phi_i\cdot\Phi_k)} \quad (7)$$

Since exact computation of $f_1$ would require computations over all vertex-pairs which would be very expensive. Instead these methods make use of approximations namely hierarchical softmax and negative sampling. Hierarchical softmax is only used by DEEPWALK. Other methods employ negative sampling.

For LINE-1, the corresponding function is given as

$$f_1^{LINE1}(\Phi_i, \Phi_j) = \log\frac{1}{1+\exp(-\Phi_i\cdot\Phi_j)} \quad (8)$$

and it further approximates it using negative sampling.

**Neighborhood Aggregation and Negative Sampling.** GRAPHSAGE trains a set of aggregator functions that learn to aggregate feature information from a node's local neighborhood. Like others GRAPHSAGE uses an unsupervised loss and its context graph corresponding to the loss function

is same as that of DEEPWALK. Instead of directly learning the embeddings as done by other methods, GRAPHSAGE learns the parameters of the aggregator functions via stochastic gradient descent.

**Deep Autoencoders.** SDNE uses a multi-layer auto-encoder model to capture non-linear structures based on first- and second-order proximities. By reconstructing first order proximity, the model aims to embed vertices closer which have links between them with the corresponding loss function given by

$$\mathcal{L}_1 = \sum_{i,j} c_{i,j} ||\Phi_i - \Phi_j||^2. \tag{9}$$

Drawing parallel to Equation (2) we have $f_1^{SDNE} = ||\Phi_i - \Phi_j||^2$ For preserving second order proximity it uses the adjacency matrix as input to the autoencoder. Denoting row $i$ of matrix $C$ by $c_i$ the reconstruction process will make the vertices with similar neighborhood structures have similar latent representations, i.e, the following loss function will be minimized

$$\mathcal{L}_2 = \sum_i ||c_i - g(\Phi_i) \odot \beta||^2, \tag{10}$$

where $g$ is a decoder function. $\mathcal{L}_2$ is an auxiliary reconstruction loss and is restricted to a node rather than a pair of nodes and hence is of different form than Equation (2). The contribution of these two proximities is controlled by the hyperparameter $\alpha$ such that setting $\alpha = 0$ will switch to only preserving second order proximity. Another hyperparameter $\beta$ controls the reconstruction of zero elements in the adjacency matrix of the training graph. For simplicity we state the loss function without the regularization term

$$\mathcal{J} = \mathcal{L}_2 + \alpha \mathcal{L}_1 \tag{11}$$

**Remark 1.** *Like* SDNE, LINE *also aims preserve first and second order proximities. But unlike* LINE, SDNE *uses a deep neural network and performs joint optimization as opposed to learning two separate embeddings and later concatenating them.*

**Matrix Factorization.** HOPE and NETMF compute low rank decomposition of their respective context matrices. While HOPE uses both factors for downstream task denoting the first factor as the source representation of the vertex and the second as target representation, NETMF only uses one representation matrix for downstream tasks. Their loss function is given as

$$\mathcal{J} = ||C - \Phi \cdot \theta||_F^2, \tag{12}$$

where $||.||_F$ denotes the Forbenius norm. Table 1 summarizes the list of embedding methods along with the corresponding properties with respect to defining and exploiting context and loss functions.

### 3.5 Research Questions

Based on the differences due to context and optimization methods described above, we formulate the following research questions.

**RQ 1.** *How does the choice of different context schemes defined in Section 3.2 affect the performance of downstream tasks? And to what extent this performance is influenced by the structural properties of the underlying graph?*

**RQ 2.** *How does different ways of exploiting the context listed in Section 3.3, affect the performance of network representation learning methods? Which combination of downstream tasks and input graphs could benefit from the explicit use of context embeddings?*

**RQ 3.** *How does choice of optimization method (listed in Section 3.4) affect the performance? Do deep models always outperform the shallow models?*

We answer these research questions in Section 7 based on the observations from extensive experimental comparison and summarize the answers in Section 7.3.

## 4 TASK DESCRIPTION

In this section we describe two most popular tasks used for empirically comparing various UNRL methods – Link Prediction (LP) and Node Classification (NC). We also discus the shortcomings of previous works with respect to these tasks and propose new experimental settings to overcome the same.

### 4.1 Link Prediction (LP)

The aim of the link prediction task is to predict missing edges given a network with a fraction of removed edges. In the literature there have been slightly different yet similar experimental settings. A fraction of edges is removed randomly to serve as the *test split* while the residual network can be utilized for training. The test split is balanced with negative edges sampled from random vertex pairs that have no edges between them. While removing edges randomly, we make sure that no node is isolated, otherwise the representations corresponding to these nodes can not be learned.

For directed graphs in addition to the existence of an edge it is also desirable to learn about the directionality of the edge. Therefore, for directed graphs we inverse a fraction of positive edges in the test split in order to create negative edges. For example given an edge $(a, b)$ in the test split we check if $(b, a)$ is also an edge. If not, we replace another negative edge with $(b, a)$ in the negative edge list of the test split. It is trivial to note that methods employing a single representation of vertex embeddings would not be able to simultaneously predict the existence of edge $(a, b)$ and non existence of edge $(b, a)$.

Tables 3 and 4 present the ROC-AUC (Area Under the Receiver Operating Characteristic Curve) scores for undirected and directed graphs respectively. For each embedding, the inner product of two node representations normalized by the sigmoid function is employed as the similarity/link-probability measurement.

**Remark 2.** *We remark that most of the previous works are lacking in the sense that they only evaluate if the method predicts a link and ignore the edge directionality for directed graphs hence giving an unfair treatment to methods designed specifically for directed graphs like* HOPE *and* APP.

**Remark 3.** *Note that the difficulty of link prediction in directed graphs will be influenced by its* reciprocity. *For example for*

*graphs with reciprocity zero, single representation would not be able to simultaneously predict the existence of edge in one direction and its non-existence in the other direction.*

### 4.2 Multilabel Node Classification (NC)

Given a graph, each node has one or more labels. We report the Micro-F1 and Macro-F1 scores after a 5-fold multi-label classification using one-vs-rest logistic regression. The main motivation behind using embeddings for this task is the assumption that node neighborhood dictates the node labels. For example a republican would have more republican friends than democrats. We use 5 undirected and 3 directed networks for this task. The three directed networks with labels are the citation networks wherein an edge represents a citation relationship.

**New Baseline.** In order to better judge the difficulty of predicting labels for a particular graph we propose a naïve baseline, which we call, MAX-VOTE. In this approach, in order to assign a label to a node, only the known labels of its immediate neighborhood are considered. In MAX-VOTE, we first split the datasets into training and test set (80-20) and the labels are assigned for the nodes in test set using only the labels of the neighbors which are part of the training set but not in the test set. For a given node with $k$ labels in the ground truth, we assign it the most frequent $k$ labels of its labelled immediate neighbors. If less than $k$ neighboring nodes are labelled or the neighbors of a node have less than $k$ labels, remaining labels are chosen randomly from the list of all possible labels in the graph. The pseudo-code for subroutine to label a node is shown in Algorithm 1 where $\ell$ denotes the total number of label classes.

**Remark 4.** *By* homophily *in node classification we understand that the similar nodes share the same label. Our baseline method* MAX-VOTE *quantifies* homophily *when similarity is limited to similarity between immediate neighbors.*

**Algorithm 1** Subroutine to label a node with MAX-VOTE

```
function LABEL(v, N(v), k)
    for (i = 0, 1, ... ℓ) do
        L(i) = 0
    for (i ∈ N(v)) do
        if (i is labelled) then
            for j ∈ labels(i) do
                L(j) = L(j) + 1
    Choose the most frequent k labels in L to label v
```

In the next section, we describe various structural properties of the underlying graphs which further aid in explaining the performance differences of various methods.

## 5 Structural Properties

In order to quantify the impact that different kinds of graphs have on the performance of the node representations, we consider diameter, reciprocity, clustering coefficient, transitivity and spectral separation.

In order to compute diameter ($D$), edge directions are not considered. In networks that are not connected, the diameter of the largest connected component is reported. In a directed network, the reciprocity ($r$) equals the proportion of edges for which an edge in the opposite direction exists, i.e., that are reciprocated, i.e., $r = \frac{1}{m}|\{(u,v) \in E \mid (v,u) \in E\}|$.

The local clustering coefficient of a vertex quantifies how probable it is for $v$ to form a clique of size 3 with its neighbors. Formally, if $d(v)$ is the degree of $v$, then local clustering coefficient of $v$ is defined as

$$c(u) = \frac{|(u,w) \in E \mid (u,v) \in E, (v,w) \in E|}{\binom{d(v)}{2}}.$$

For directed graphs, the local clustering coefficient of a node $u$ equals the proportion of directed 2-paths starting from $u$ that are completed by a third edge oriented in the same direction as the 2-path. The clustering coefficient ($C$) of graph $G$ is then defined as the average of the local clustering coefficients of its vertices. We denote the directed clustering coefficient by $C_{dir}$.

Transitivity ($T$) measures the extent to which two nodes are related in a network that are connected by an edge is transitive. It is defined as the ratio of the number of vertex triplets forming a triangle to the total number of triads (subgraphs of 3 vertices). For directed graphs, the transitivity ($T_{dir}$) equals the proportion of directed 2-paths that are completed by a third edge oriented in the same direction as the 2-path.

The spectral separation ($S$) equals the largest absolute eigenvalue of the adjacency matrix divided by the second largest absolute eigenvalue. Low values (just slightly larger than one) indicate many independent substructures in the network.

## 6 Experimental Setup

We empirically validate the impact of various differences among the 9 embedding methods (cf. Table 1) on task performance. For reproducibility we used the authors' implementations whenever available and performed hyperparameter tuning whenever applicable. We provide a detailed description of parameter settings, hardware and software setup in the supplementary material (Section 2). We consider 11 popular real world datasets consisting of six social network graphs, four citations networks and an authorship network with their structural properties summarized in Table 2. We consider two tasks LP and NC defined in Section 4.

**Social Network Graphs:** BlogCatalog, Flickr and Youtube are social networks with users as nodes and friendship between them as undirected edges. All these datasets also have multiple labels per node for each group or community the user belongs to. Reddit on the other hand is an artificially generated network by [7], a post-to-post graph, connecting Reddit posts if the same user comments on both. The labels for the nodes in Reddit graph represent the subreddit (communities) they belong to. Since each post can only belong to one subreddit, each node has only one label. Finally, Twitter and Epinion are unlabelled, directed graphs modeling the follower and trust between users respectively.

**Citation Graphs:** DBLP-Ci, CoCit, Cora, PubMed are directed graphs representing academic citation networks, with nodes as papers and edges representing the citations between them. DBLP-Ci and Cora are parsed from scientific

| Category | Dataset | Type | $\|V\|$ | $\|E\|$ | $r$ | $D$ | $C$ | $T$ | $C_{dir}$ | $T_{dir}$ | $S$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Social | BlogCatalog [29], [30] | undir. | 10K | 333K | - | 5 | 0.463 | 0.0914 | - | - | 2.18 |
| | Flickr [29], [30] | undir. | 80K | 5.90M | - | 6 | 0.165 | 0.1875 | - | - | 2.06 |
| | Youtube [16] | undir. | 1.13M | 2.99M | - | 21 | 0.080 | 0.0062 | - | - | 1.19 |
| | Reddit [7] | undir. | 231K | 11.6M | - | 10 | 0.169 | 0.0458 | - | - | 1.47 |
| | Twitter [4] | dir. | 465K | 834K | 0.3% | 8 | 0.0006 | 0. 0152 | 0.0002 | 0.013 | 1.05 |
| | Epinion [23] | dir. | 75K | 508K | 40.52% | 15 | 0.1378 | 0.0657 | 0.0982 | 0.0902 | 1.74 |
| Citation | DBLP-Ci [11] | dir. | 12.5K | 49K | 46.4% | 10 | 0.1169 | 0.0620 | 0.039 | 0.0967 | 1.39 |
| | CoCit [25] | dir. | 44K | 195K | 0% | 25 | 0.1419 | 0.0806 | 0.0826 | 0.0913 | 1.07 |
| | Cora [27] | dir. | 23K | 91K | 5.00% | 20 | 0.2660 | 0.1169 | 0.169 | 0.221 | 1.03 |
| | PubMed [18] | dir. | 19K | 44k | 0.07% | 18 | 0.0602 | 0.0537 | 0.0325 | 0.0530 | 1.14 |
| Collaboration | DBLP-Au [28] | undir. | 1.2M | 10.3M | - | 24 | 0.635 | 0.1718 | - | - | 1.0005 |

TABLE 2: A summary of benchmark datasets for evaluating network representation learning.

publications from the Computer Science community. While DBLP-Ci is unlabelled, Cora has multilabels representing the sub-communities in Computer Science such as "Machine Learning", "Databases" etc. CoCit is a labeled citation graph from Microsoft Academic Graph, with labels representing conferences in which the papers were published. Finally, PubMed, is a citation graph derived from the medical literature database pertaining to diabetes classified into one of three classes of diabetes.

**Collaboration Network:** Finally, DBLP-Au is a collaboration network of authors of scientific papers from DBLP Computer Science bibliography. An undirected edge between two authors represents a common publication. There may be multiple edges between the authors if they collaborated on multiple papers, but we only consider single edges.

## 7 Results and Discussion

### 7.1 Link Prediction

Main results for the LP task for both undirected (cf. Table 3) and directed graphs (cf. Table 4) are summarized below:

1) For undirected graphs, PPR based methods APP and VERSE are more or less the best performing methods in all datasets (cf. Table 3).
2) LINE which directly uses adjacency matrix as context matrix outperforms random walk based methods for undirected graphs (cf. Section 7.1.2).
3) For directed graphs with low reciprocity, context plays a major role (cf. 7.1.2) and methods encoding and using two embedding spaces for source and target roles of nodes should be used for directed link prediction.
4) Deeper models do not have a considerable advantage over the shallow ones for this task (cf. Section 7.1.3).

#### *7.1.1 Different Schemes of Context.*

In this section, we investigate in detail the performance difference potentially caused by differences in the definition of the context as questioned in RQ1.

**Random Walk Based.** We first ponder over the usefulness of expensive biased walks employed by NODE2VEC and establish that the biases can in fact cause differences in performance as compared to simpler DEEPWALK for graphs with certain structural properties. We begin by quantitatively and qualitatively examining the utility of biased random walks used in NODE2VEC in building its context graph. From properties 2 and 3 we infer that the biased walks of NODE2VEC will not produce any significant gains for graphs with low clustering coefficient and low reciprocity for example Twitter which is also evident in the empirical results (see Table 4). On the contrary, for undirected graphs with high clustering ratio like BlogCatalog, one observes a relatively higher standard deviation (computed mean and standard deviations provided in Supplementary Material) from the mean of scores computed with 25 combinations of the $p$ and $q$ parameters. Similarly, for directed graphs with high reciprocity and high clustering coefficient, the choice of parameters $p$ and $q$ matters for NODE2VEC. Notable differences are observed for directed dataset Epinion with high reciprocity and clustering coefficient where NODE2VEC outperforms DEEPWALK by $72.86\%$ for the case when only random negative edges exist in the test set. We also observe that for other directed graph with high reciprocity and clustering coefficient, NODE2VEC performs better than DEEPWALK. Hence we infer that the biased walks in NODE2VEC can produce considerably different results from DEEPWALK for graphs with high clustering coefficient, high diameter and high reciprocity (in case of directed graphs).

**Adjacency based.** We observe that for link prediction on undirected graphs, LINE performs better than DEEPWALK and NODE2VEC. Note that LINE uses adjacency matrix as its context graph. We observe that for Youtube with lowest clustering co-efficient and transitivity, LINE outperforms DEEPWALK and NODE2VEC by $26.99\%$ whereas for Flickr with transitivity of $0.1875$, the gain is $3.1\%$. On the other hand, for DBLP-Au and Flickr with high clustering coefficient and transitivity, LINE outperforms these methods by a smaller margin. All of the observations lead to the conclusion that LINE performs comparable or better as compared to DEEPWALK and NODE2VEC, with the performance becoming much better for graphs with low clustering coefficient and transitivity. SDNE on the other hand performs worse than LINE and other methods (for more discussion see Section 7.1.3). **Direct Matrix based.**

NETMF is designed specifically for undirected graphs and HOPE for directed graphs. In the original paper NETMF was not compared for the task of link prediction. NETMF could only be run for smaller graphs and there is no clear advantage of using NETMF over other methods for link prediction task. Of the three datasets we observe that NETMF performs better than DEEPWALK, NODE2VEC and LINE for BlogCatalog and Reddit with low transitivity but high clustering coefficient. HOPE while using two embedding spaces to encode a vertex in its source and target roles outperforms most of the single embedding based methods

| *method* | BlogCatalog | Youtube | Reddit | DBLP-Au | Flickr |
|---|---|---|---|---|---|
| DEEPWALK | 0.527 | 0.586 | 0.897 | 0.850 | 0.772 |
| NODE2VEC | 0.556 | 0.652 | 0.892 | 0.949 | 0.821 |
| VERSE | **0.878** | 0.884 | 0.973 | **0.994** | 0.918 |
| APP | 0.790 | 0.871 | **0.974** | 0.994 | **0.928** |
| NetMF | 0.659 | ✗ | 0.949 | ✗ | 0.604 |
| LINE-1+2 | 0.612 | **0.894** | 0.949 | 0.989 | 0.839 |
| LINE-1 | 0.495 | 0.758 | 0.947 | 0.989 | 0.830 |
| LINE-2 | 0.400 | 0.823 | 0.833 | 0.896 | 0.694 |
| GraphSage | 0.619 | 0.778 | 0.936 | 0.912 | 0.734 |
| GraphSage-GCN | 0.661 | 0.813 | 0.941 | 0.975 | 0.779 |
| SDNE | 0.519 | ✗ | ✗ | ✗ | 0.483 |

TABLE 3: Link prediction results for undirected graphs using 50% edges as training data. ✗ indicates the corresponding method failed to finish for the given dataset.

| | Cora | | | Twitter | | | DBLP-Ci | | | Epinion | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *method* | 0% | 50% | 100% | 0% | 50% | 100% | 0% | 50% | 100% | 0% | 50% | 100% |
| DEEPWALK | 0.836 | 0.669 | 0.532 | 0.536 | 0.522 | 0.501 | 0.868 | 0.680 | 0.503 | 0.538 | 0.560 | 0.563 |
| NODE2VEC | 0.840 | 0.649 | 0.526 | 0.500 | 0.500 | 0.500 | 0.889 | 0.697 | 0.503 | 0.930 | 0.750 | 0.726 |
| VERSE | 0.875 | 0.688 | 0.500 | 0.52 | 0.510 | 0.501 | 0.809 | 0.654 | 0.503 | **0.955** | **0.753** | **0.739** |
| APP | 0.865 | **0.841** | **0.833** | 0.723 | 0.638 | 0.555 | **0.957** | **0.838** | 0.722 | 0.639 | 0.477 | 0.455 |
| HOPE | 0.784 | 0.734 | 0.718 | **0.981** | **0.980** | **0.979** | 0.756 | 0.737 | **0.732** | 0.807 | 0.718 | 0.716 |
| LINE-1+2 | 0.735 | 0.619 | 0.518 | 0.009 | 0.255 | 0.500 | 0.319 | 0.404 | 0.501 | 0.658 | 0.622 | 0.617 |
| LINE-1 | 0.781 | 0.644 | 0.526 | 0.007 | 0.007 | 0.254 | 0.312 | 0.405 | 0.501 | 0.744 | 0.677 | 0.668 |
| LINE-2 | 0.693 | 0.598 | 0.514 | 0.511 | 0.507 | 0.503 | 0.642 | 0.572 | 0.503 | 0.555 | 0.544 | 0.543 |
| GRAPHSAGE | 0.902 | 0.707 | 0.531 | 0.659 | 0.602 | 0.504 | 0.806 | 0.656 | 0.503 | 0.814 | 0.672 | 0.658 |
| GRAPHSAGE-GCN | **0.927** | 0.721 | 0.534 | 0.589 | 0.539 | 0.502 | 0.856 | 0.670 | 0.503 | 0.816 | 0.668 | 0.668 |
| SDNE | 0.613 | 0.557 | 0.507 | ✗ | ✗ | ✗ | 0.569 | 0.540 | 0.501 | 0.601 | 0.560 | 0.551 |

TABLE 4: Link Prediction Results for directed graphs with (1) random negative edges in test set (2) 50% of the test negative edges created by reversing true edges of the test set (3) when all true edges of test set are reversed to create negative edges in the test set. ✗ indicates the corresponding method failed to finish for the given dataset.

for directed link prediction but is still mostly outperformed by APP, exceptions being for Twitter and Epinion. Interesting to note is that it is better in predicting the edge direction than APP (see results corresponding to 100% edge reversal in Table 4). In summary, Katz based context graph as used by HOPE performs best for directed graphs with very low reciprocity and low clustering coefficient for example Twitter. **GRAPHSAGE.** GRAPHSAGE-GCN outperforms DEEPWALK, NODE2VEC and SDNE for most of the directed and undirected graphs. Interestingly, GRAPHSAGE does not seem to be more advantageous than LINE for undirected graphs for link prediction. Surprisingly for directed graphs with high reciprocity, GRAPHSAGE-GCN outperforms not only LINE but sometimes also HOPE and APP when only random negative edges are considered in the test set (see the column corresponding to $0\%$ for Cora, DBLP-Ci and Epinion in Table 4).

#### 7.1.2 Exploitation of Context

We recall that both methods APP and VERSE use similar context graphs, but the main difference is that VERSE uses single embedding space, while APP uses two different embedding spaces. The advantage of using two embedding spaces by APP is not straightforward. As per the arguments of the authors, there might exist asymmetries in undirected graphs due to differing local properties of a node for example degree. This argument is still insufficient to interpret the use of the embedding space to predict missing links. For example, consider that we would like to predict the existence or a non existence of link between the nodes $u$ and $v$. Using two embedding spaces might result in prediction of link between $u$ and $v$ but not between $v$ and $u$. This can happen when destination representation of $v$ is embedded closer to source embedding of $u$ but the source embedding of $v$ is far away from destination $u$. Note that such a result is probable for example for nodes $u$ and $v$ as shown in Figure 1. For undirected link prediction, other methods for example LINE-1 and GRAPHSAGE-GCN learning only a single representation perform relatively better than DEEPWALK, NODE2VEC and LINE-2 which learn both node and context representations.

**Effect of Reciprocity in Directed Link Prediction.** For directed graphs with low reciprocity, learning and using two embedding matrices per node is more intuitive as these two matrices represent the two roles of a node (source and target respectively). We observe that single embedding based methods are insufficient to capture the directed relationship in graphs. We report results corresponding to

| | BlogCatalog | | PubMed | | Cora | | Reddit | | Flickr | | Youtube | | CoCit | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *method* | mi. | ma. | mi. | ma. | mic. | mac. | mic. | mac. | mic. | mac. | mic. | mac. | mic. | mac. |
| DEEPWALK | 42.15 | 28.48 | 73.96 | 71.34 | 64.98 | 51.53 | 94.40 | 92.01 | **42.20** | **31.00** | 47.09 | 39.89 | 41.92 | 30.07 |
| NODE2VEC | 42.46 | **29.16** | 72.36 | 68.54 | 65.74 | 49.12 | 94.11 | 91.73 | 42.11 | 30.57 | **48.41** | **42.04** | 41.64 | 28.18 |
| VERSE | 35.51 | 21.77 | 71.24 | 68.68 | 60.87 | 45.52 | 92.87 | 89.69 | 35.70 | 23.00 | 45.12 | 37.28 | 40.17 | 27.56 |
| APP | 20.60 | 5.39 | 69.00 | 65.20 | 64.58 | 47.03 | 77.11 | 56.28 | 24.26 | 4.21 | 45.04 | 36.61 | 40.34 | 28.06 |
| HOPE | n.a | n.a | 63.00 | 54.6 | 26.23 | 1.22 | n.a | n.a | n.a | n.a | n.a | n.a | 16.66 | 1.91 |
| NETMF | **43.29** | 29.04 | 73.66 | 71.11 | 63.38 | 46.16 | 91.99 | 86.92 | 37.44 | 21.55 | ✗ | ✗ | 40.42 | 28.7 |
| LINE-1+2 | 41.01 | 25.02 | 62.29 | 59.79 | 54.04 | 41.83 | **94.50** | **92.08** | 41.46 | 27.65 | 48.22 | 41.51 | 37.71 | 26.75 |
| LINE-1 | 41.54 | 24.28 | 55.65 | 53.83 | 62.36 | 47.19 | 94.31 | 91.96 | 40.92 | 26.19 | 47.49 | 41.17 | 36.10 | 25.70 |
| LINE-2 | 36.70 | 18.80 | 56.81 | 51.71 | 51.05 | 35.37 | 94.30 | 91.81 | 40.49 | 24.24 | 47.46 | 39.97 | 31.4 | 20.59 |
| GRAPHSAGE | 19.28 | 5.07 | 77.90 | 76.39 | 67.07 | 44.78 | 89.94 | 82.28 | 25.52 | 5.84 | 40.45 | 29.97 | 43.71 | 30.52 |
| GRAPHSAGE-GCN | 26.76 | 10.82 | **79.19** | **77.85** | 69.64 | 51.64 | 91.65 | 86.88 | 29.66 | 9.69 | 42.54 | 32.54 | 44.08 | 30.73 |
| SDNE | 26.40 | 12.29 | 46.41 | 32.32 | 32.43 | 8.27 | ✗ | ✗ | 29.10 | 10.53 | ✗ | ✗ | 21.67 | 9.53 |
| MAX-VOTE | 32.71 | 19.60 | 76.81 | 75.25 | **71.96** | **57.21** | 93.26 | 90.11 | 34.60 | 22.48 | 28.96 | 25.65 | **44.66** | **33.39** |

TABLE 5: Multilabel Node Classification results in terms of Micro-F1 and Macro-F1. All results are mean of 5-fold cross validations. ✗ indicates the corresponding method failed to finish for the given dataset.

the LP for directed graphs in Table 4. Note that we test three settings: for $0\%$, we use random negative edges in the test set, for $50\%$ and $100\%$ we force the model to not only predict the right edges but also decide on the edge direction by using the reverse of true (positive) edges in the test set as negative edges (if possible). Methods using two representations per vertex, HOPE and APP outperform single embedding based methods for all directed datasets except for Epinion which has a high reciprocity.

#### 7.1.3 Differences in Optimization.

We observe that the joint optimization of first and second order objectives using deep auto-encoder by SDNE does not provide any additional performance gains as compared to LINE and other methods. We could not run SDNE for bigger datasets because of its prohibitive memory requirements imposed by the input adjacency matrix.

GRAPHSAGE shares the same unsupervised loss function as DEEPWALK but instead of learning directly embeddings it learns parameters of neighborhood aggregation functions. Even though it is not the best performing methods but when compared to its counterpart sharing the same loss function, it performs much better than DEEPWALK for link prediction in directed and undirected graphs.

### 7.2 Node Classification

In this section, we look at the results from node classification (Table 5). We also present additional experiments to measure the learning rate of different methods for the NC task in supplementary material (Section 4). In contrast to the earlier link prediction task, node classification is a supervised task that includes external information in the form of labels. The effectiveness of an unsupervised representation for vertex classification is the extent to which it can reconcile varying degrees of homophily. We observe that DEEPWALK is either the best performing approach or reasonably competitive in most of the datasets. Note that this is both *surprising and counter intuitive* since it was the earliest proposed approach. This calls into question the utility of its other variants for example biased random walk methods such as NODE2VEC.

As mentioned in Remark 4 the homophily baseline (MAX-VOTE) measures the degree of label similarity among neighboring nodes. Lower values indicate low neighborhood homophily where a node is less likely to share the label as that of its neighbours. In Section 7.2.2 we investigate the effect of neighborhood homophily in detail and its consequence on utility of edge direction in directed graphs.

#### 7.2.1 Different Schemes of Context.

**Random Walk based.** Both DEEPWALK and NODE2VEC perform quite well in this task. Taking advantage of longer walks exploiting similarities with higher order neighborhoods, both of these methods perform specifically well when MAX-VOTE's performance's drops, i.e., when neighboring nodes might not have same labels.

**PPR based.** In contrast to link prediction, for node classification task PPR based context matrix is not the best performing.

**Direct Matrix based.** We observe that NETMF has the best Macro-F1 for NC task on BlogCatalog and close to DEEPWALK in all other cases. We could not run NETMF for large datasets because of prohibitive memory requirements limiting any further analysis. HOPE performs poorly for all datasets. We believe that as HOPE is tied to a particular similarity matrix, it is limited to a certain type of task and cannot be generalized.

#### 7.2.2 Exploitation of Context

We believe that for directed graphs, edge directionality has little effect on the labels on the nodes. We verify this hypothesis through the empirical performance of various methods as shown in Table 5 and as discussed below.

**Baseline - MAX-VOTE** First, we observe that our naïve MAX-VOTE baseline outperform other methods for all directed graphs. We note that for MAX-VOTE, edge direction has been ignored.

**APP and HOPE.** For all directed graphs, methods ignoring the context representation outperform HOPE and

| Algorithm | Favourable Task | Favourable/Unfavourable Label Properties | Favourable Graph Properties |
|---|---|---|---|
| DEEPWALK | NC | Robust for different label distributions | High Spectral separation |
| NODE2VEC | NC | Robust for different label distributions | High Clustering Coefficient, High Reciprocity |
| APP | LP | - | Directed Graphs, Low Spectral Separation |
| VERSE | LP | - | Undirected or directed with high reciprocity |
| LINE | LP, NC | Low Similarity among labels of neighboring nodes | Undirected, low clustering coefficient, low transitivity |
| NETMF | NC | Robust for different label distributions | Undirected |
| HOPE | LP | - | Directed Graphs with Low Reciprocity and low Clustering coefficient |
| GRAPHSAGE-GCN | LP, NC | High Label similarity among immediate neighbors | Undirected graphs with high clustering coefficient |

TABLE 6: Summary of Main Results corresponding to best performing methods, favourable task along with favourable graph and label properties are listed. Note that only those methods are included which are best performing in at least one dataset in at least one task.

APP which use both vertex and context representation for node classification.

**VERSE and LINE-1.** VERSE which only learns vertex representation, hence ignoring the role of context performs better than APP which shares the same context graph as VERSE but additionally learns and uses context representations. Moreover, LINE-1 which specifically only learns vertex representation hence ignoring the edge directionality outperforms LINE-2 (designed to take directionality into account) in almost all datasets (except PubMed).

**DEEPWALK, NODE2VEC, LINE-2, NETMF.** As already observed in Property 1 the context matrix used by these methods is symmetric even if the underlying graph is directed. Consider for example, a directed random walk of length 1 with vertex sequence $(v1, v2)$ with window length 1. Now the following vertex-context pairs are considered for training: $(v1, v2)$ and $(v2, v1)$, thereby ignoring the edge direction between $v1$ and $v2$. Note that this is not the case for LINE-2 which would only consider $(v1, v2)$ for training. So in the above described sense, DEEPWALK and NODE2VEC are still ignoring edge directionality, hence the role of context while training even if they operate on directed random walks.

#### 7.2.3 Differences in Optimization

Similar to link prediction, LINE outperforms SDNE which uses deep autoencoders. GRAPHSAGE-GCN outperforms most of the other methods when the baseline MAX-VOTE performs well, i.e., when there is a high degree of similarity among neighboring nodes. This is also understandable since GRAPHSAGE-GCN constructs node representations as convolutions of its immediate neighbor representations which explains its good performance when there exist a strong homophily in label distribution of neighboring nodes.

### 7.3 Answers to Research Questions

In the following we summarize the above analysis while answering the research questions from Section 3.5. The main results pertaining to favourable tasks and dataset properties for best performing methods are summarized in Table 6.

**RQ 1.** From the experimental results described above, it is clear that the choice of different schemes of context is dependent on both task and underlying graph properties. For example, for the LP task, PPR based context graph construction provides best results for both undirected and directed graphs. Similarly, for directed graphs, with low reciprocity, Katz similarity based context graph provides best results. Biased walks on the other hand have advantages in link prediction for graphs with high clustering coefficient, high transitivity and high reciprocity. With low clustering coefficients and transitivity LINE-1 performs much better for link prediction. Random walk based methods are more robust in the task of node classification while neighborhood aggregation based methods perform best if there is a high similarity among labels of neighboring nodes.

**RQ 2.** Exploitation of context embeddings plays an important role in LP task for directed graphs where context representations should also be used along with vertex representations for predicting links. Lower the reciprocity in directed graphs, more important is the role of context. For undirected graphs, the role of context is not well defined and in general methods learning only single node representations outperform others modelling both node and context. Explicit modelling of edge direction via context in directed graphs does not provide any advantage in NC task.

**RQ 3.** In general, single layer models using negative sampling work better for both LP and NC tasks. Neighborhood aggregation learnt via SGNS objective works best for node classification when there is a high similarity among labels among neighboring nodes. Optimizing first and second order proximities using negative sampling based objective as done by LINE is better than using deep autoencoders to encode these proximities as done by SDNE.

Finally, we also provide best practices and caveats for the practitioners in the supplementary material (Section 4).

## 8 CONCLUSIONS

In this paper, we systematically studied the important but unexplored problem of analyzing differences between widely used network representation learning approaches. We provide a first ever exhaustive evaluation benchmark with 11 datasets and 9 popular network embedding methods. Our analysis provided several non-intuitive insights which are beneficial for practitioners and academics to carry out experiments and apply network embedding techniques for graphs with different properties and different tasks.

As future work we plan to study semi-supervised node embedding methods on similar lines.

## REFERENCES


[1] A. Bojchevski and S. Gnnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

[2] H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE*, 30(9):1616–1637, 2018.

[3] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *TKDE*, 2018.

[4] M. De Choudhury, Y.-R. Lin, H. Sundaram, K. S. Candan, L. Xie, A. Kelliher, et al. How does the data sampling strategy impact the discovery of information diffusion in social media? *Icwsm*, 10:34–41, 2010.

[5] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[6] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

[7] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[8] W. L. Hamilton, Z. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40:52–74, 2017.

[9] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[10] T. N. Kipf and M. Welling. Variational graph auto-encoders. In *NeurIPS BDL*, 2016.

[11] M. Ley. The dblp computer science bibliography: Evolution, research issues, perspectives. In *SPIRE*, pages 1–10. Springer, 2002.

[12] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.

[13] X. Liu, T. Murata, K.-S. Kim, C. Kotarasu, and C. Zhuang. A general view for network embedding as matrix factorization. In *WSDM*, pages 375–383, 2019.

[14] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, and H. Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *ISWC*, pages 3–20. Springer, 2018.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[16] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*. ACM, 2007.

[17] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[18] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU. Query-driven active surveying for collective classification. In *MLG*, 2012.

[19] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114. ACM, 2016.

[20] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI*, pages 2609–2615, 2018.

[21] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710. ACM, 2014.

[22] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, pages 459–467, 2018.

[23] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *ISWC*, pages 351–368. Springer, 2003.

[24] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[25] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-j. P. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. In *WWW*, pages 243–246. ACM, 2015.

[26] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *SIGCOMM*, pages 322–335. ACM, 2009.

[27] L. Šubelj and M. Bajec. Model of complex networks based on citation dynamics. In *WWW*, pages 527–530. ACM, 2013.

[28] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[29] L. Tang and H. Liu. Relational learning via latent social dimensions. In *SIGKDD*, pages 817–826. ACM, 2009.

[30] L. Tang and H. Liu. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM*, pages 1107–1116, 2009.

[31] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller. Verse: Versatile graph embeddings from similarity measures. In *The Web Conference*, pages 539–548, 2018.

[32] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Li, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[33] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234. ACM, 2016.

[34] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[35] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang. Network representation learning with rich text information. In *IJCAI*, 2015.

[36] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, pages 974–983. ACM, 2018.

[37] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang. Learning deep network representations with adversarially regularized autoencoders. In *SIGKDD*, pages 2663–2671. ACM, 2018.

[38] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 2018.

[39] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao. Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948, 2017.

[40] D. Zhu, P. Cui, D. Wang, and W. Zhu. Deep variational network embedding in wasserstein space. In *SIGKDD*, pages 2827–2836. ACM, 2018.

# A Comprehensive Comparison of Unsupervised Network Representation Learning Methods (Suplementary Material)

Megha Khosla[1], Avishek Anand[1], and Vinay Setty [2]

[1]L3S Research Center and Leibniz Universität Hannover, Germany
[2]Department of Electrical Engineering and Computer Science, University of Stavanger, Norway

March 19, 2019

## 1 Missing Proofs

*Proof of Proposition 1.* Note that $i$ is chosen uniformly at random , i.e., the probability of choosing $i$ as the starting vertex is $\frac{1}{|V|}$. For an $h$ hop walk starting from fixed $i$, the probability that $j$ is the chosen neighbor is given by the $(i,j)th$ element of the transition matrix over $h$-hops, i.e., $\mathbf{P}^h_{i,j}$.

Again the probability that the walk will stop in exactly $h$ hops is $(1-\alpha)^h \cdot \alpha$. Again there might exist paths of length greater than $h$ from $i$ to $j$; say of lengths $h_1, h_2 \dots h_k$. Then probability that pair $(i,j)$ will be sampled is given by

$$\Pr((i,j) \text{ is sampled}) = \frac{1}{|V|} \sum_{h' \geq h} (1-\alpha)^{h'} \cdot \alpha \cdot (\mathbf{P}^{h'})_{i,j} \tag{1}$$

$$\leq \frac{1}{|V|} \sum_{h' \geq h} (1-\alpha)^{h'} \cdot \alpha \cdot (\mathbf{P}^{h})_{i,j}$$

$$= \frac{1}{|V|} (1-\alpha)^{h} \cdot \alpha \cdot (\mathbf{P}^{h})_{i,j} \sum_{h' \geq 1} (1 + (1-\alpha)^{h'})$$

$$\leq \frac{2}{|V|} (1-\alpha)^{h} \cdot \alpha \cdot (\mathbf{P}^{h})_{i,j}$$

□

## 2 Parameter Settings

Here we describe all the tunable hyperparameters which are common across methods. Unless specified explicitly we use default parameters provided by

the author implementations. For all methods we fix the embedding dimensions $d = 128$ as it is the most common practice in the literature.

**Random Walk:** For all methods which rely on random walks, we set the target walk length $t = 40$ and number of walks $r = 80$ as it provided the best results. For all methods using SGNS, we set the negative sample size $ns = 5$ and window size $w = 10$. For all methods we also set number of worker threads to 32 since we observed a minor variation in performance with different number of worker threads. This is due to the way random walks are performed in parallel.

For NODE2VEC there are two hyperparameters $p$ and $q$, for biased random walks. The authors recommend exploring the parameters $p, q \in \{0.25, 0.5, 1, 2, 4\}$. Since that results in 25 combinations, it is very expensive to explore these parameters for all datasets especially for large datasets such as Youtube and Reddit. For these datasets we fix the $p = 0.25, q = 4$ which were the best performing parameters in most cases. We summarize the best performing parameters in Table **??** and **??** along with mean and standard deviation of the accuracy values for different values of $p$ and $q$. As you can observe, these parameters do not play a huge role in the performance of NODE2VEC as the standard deviation is quite low in most cases.

For VERSE, we fix $\alpha = 0.85$, which is the default setting used for personalized page rank algorithm in [**?**]. We omit the variation HVERSE which is nothing but the best performing accuracies after hyperparameter exploration in the original paper since it is too expensive.

For APP, no information is provided in the original paper about the optimal parameters, therefore we iterate through the node list 80 times, in each iteration we run 10 random walks per node, thereby totalling 800 random walks per node as we do with all the random walk based methods.

For LINE, we run experiments with $T = 10$ billion samples and $s = 5$ negative samples, as described by the authors in their paper [**?**]. In addition, we also compare three variants of LINE: (1) LINE-1 (LINE with first-order proximity), (2) LINE-2 (LINEwith second-order proximity) and (3) LINE-1+2 which is obtained by normalizing and concatenating the 64-dimensional embedding vectors from LINE-1 and LINE-2.

**Matrix Factorization:** For HOPE, we set the attenuation factor $\beta = 0.01$ for all datasets except PubMed for all tasks. For PubMed, best results were obtained at $\beta = 0.5$. Choosing optimal $\beta$ is difficult as only a rough guideline is available, i.e., $\beta$ should be less than 1 divided by spectral radius of adjacency matrix to ensure the convergence of Katz measure. The authors reported best results for Cora at $\beta = 0.1$. We therefore searched for best value of $\beta$ lying close 0.1 and reported the best results.

For NETMF, we set number of eigenpairs (rank) $h = 256$ for BlogCatalog and $h = 16384$ for Flickr as suggested by the authors in their paper [**?**]. For rest of the datasets we set the default value of $h = 256$. We also observed that setting negative sample value $ns = 5$ as with other random walk approaches resulted in significantly worse performance in some cases. Therefore, we resorted to the default value of $ns = 1$. In addition, for NetMF, the authors provide two

Table 1: NODE2VEC parameters for link prediction task

| Dataset | $p$ | $q$ | mean | stddev |
|---|---|---|---|---|
| BlogCatalog | 4 | 4 | 0.543 | 0.0048 |
| Flickr | 2 | 4 | 0.811 | 0.0031 |
| Reddit | 4 | 0.25 | 0.889 | 0.0027 |
| DBLP-Au | 0.25 | 4 | 0.947 | 0.0008 |
| Youtube | 0.25 | 4 | 0.615 | 0.0167 |
| Cora | 4 | 4 | 0.838 | 0.00083 |
| Twitter | 0.25 | 4 | 0.500 | 0.0000 |
| DBLP-Ci | 0.5 | 4 | 0.800 | 0.0343 |
| Epinion | 0.25 | 4 | 0.888 | 0.0233 |

Table 2: NODE2VEC parameters for node classification task

| Dataset | $p$ | $q$ | Mean mic.F1 | Stddev mic.F1 | Mean mac.F1 | Stddev mac.F1 |
|---|---|---|---|---|---|---|
| BlogCatalog | 0.25 | 4 | 41.87 | 0.587 | 28.44 | 0.610 |
| PubMed | 0.25 | 0.25 | 72.01 | 0.230 | 68.03 | 0.238 |
| Cora | 0.25 | 4 | 65.30 | 0.268 | 47.66 | 0.737 |
| Reddit | 0.25 | 4 | - | - | - | - |
| Flickr | 0.25 | 2 | 41.57 | 0.990 | 29.53 | 1.971 |
| Youtube | 0.25 | 4 | - | - | - | - |
| CoCit | 0.5 | 0.25 | 41.56 | 0.059 | 28.05 | 0.102 |

different ways to compute Eigenvector decomposition, by specifying the parameters –small and –large which corresponds to small and large window lengths respectively. For smaller datasets we tried both and report the best performing numbers but for large datasets such as Flickr and Reddit, NETMF could only finish with –small. For many larger datasets such as Youtube and DBLP-Au, NETMF crashed by exhausting main memory before finishing training. Since, NETMF requires symmetric adjacency matrix as input, for node classification task, we convert the directed graphs to undirected and create a symmetric matrix. However, for link prediction, such a conversion does not make sense since we consider the directionality of the edges for link prediction task.

**Deep Learning:** For deep learning methods we consider GRAPHSAGE and SDNE. Since the authors do not provide any implementation for SDNE, we use a public implementation in keras. For BlogCatalog we use the hyperparameters such as hidden layer size recommended by the authors in their paper [**?**]. However, for Flickr the recommended later configuration resulted in "ResourcesExhausted" error. Furthermore, we explored $\alpha = 50, 100$ and $\beta = 1, 5, 10$

parameters for SDNE with Flickr dataset without any significant improvements.

For GraphSAGE, since we only deal with transductive, unsupervised setting in this paper we only use the unsupervised version. GraphSAGE provides four aggregators: Mean, MeanPool, MaxPool and LSTM. We repeat all experiments with each aggregator and report the best values. We also include a variant of GraphSAGE with GCN aggregator (GraphSAGE-GCN). Since it is significantly different from other GraphSAGE aggregators, we report it separately. There are several hyperparameters such as learning rate, dropout, epochs, batch size etc. It is extremely expensive to tune all these parameters for all the datasets. Instead, we follow the recommendations of the authors and explore the learning rate in 0.001, 0.0001, 0.00002 [**?**, **?**]. For GraphSAGE, in [**?**], the authors perform a grid search over several of these hyperparameters and they recommend "Mean" aggregator, along with learning rate of 0.0001, dropout 0.4 for inductive setting. The authors also recommend using "–model_size big" option for unsupervised setting which we follow. The results of GraphSAGE could be further improved by performing more exhaustive exploration of hyperparameters. However, we do not expect any contradictions to our findings.

# 3 Hardware and Software

We train all embeddings on Linux servers with 80 core Intel Xeon 2.40GHz CPU, 1TB main memory running "Scientific Linux" distribution. For algorithms which need GPUs we use Nvidia Tesla P100 GPU units with 16GB memory. Most algorithms were executed using Python 2.7 with the exception of APP and HOPE which are implemented in C++ and MatLab respectively. For SDNE, keras 2.4.4 with tensorflow 1.11 backend and GraphSAGE was executed with tensorflow 1.11.

# 4 Discussion and Best Practices

From our experiments we conclude that while using unsupervised methods for downstream tasks such as link prediction and node classification it is important to be cognizant to graph properties, label distribution and certain best practices in the experimental setup. We performed one additional experiment for node classification, common in many papers, for observing the trends in learning improvements when the training data is steadily increased. The practice employed by notable works [**?**, **?**] fix a training data sample and take the complement as the test set. We contend this by instead fixing the test data set (to 20% of the input) as choosing a variable test set is misleading. We increase the training data in steps of 10% (cf. Figure **??**). Note that we do not report all the approaches due to legibility of the plot (but they show similar trend).

We observe not surprisingly that the performance increases with increasing training data and plateaus at 40% of the training data with some exceptions.

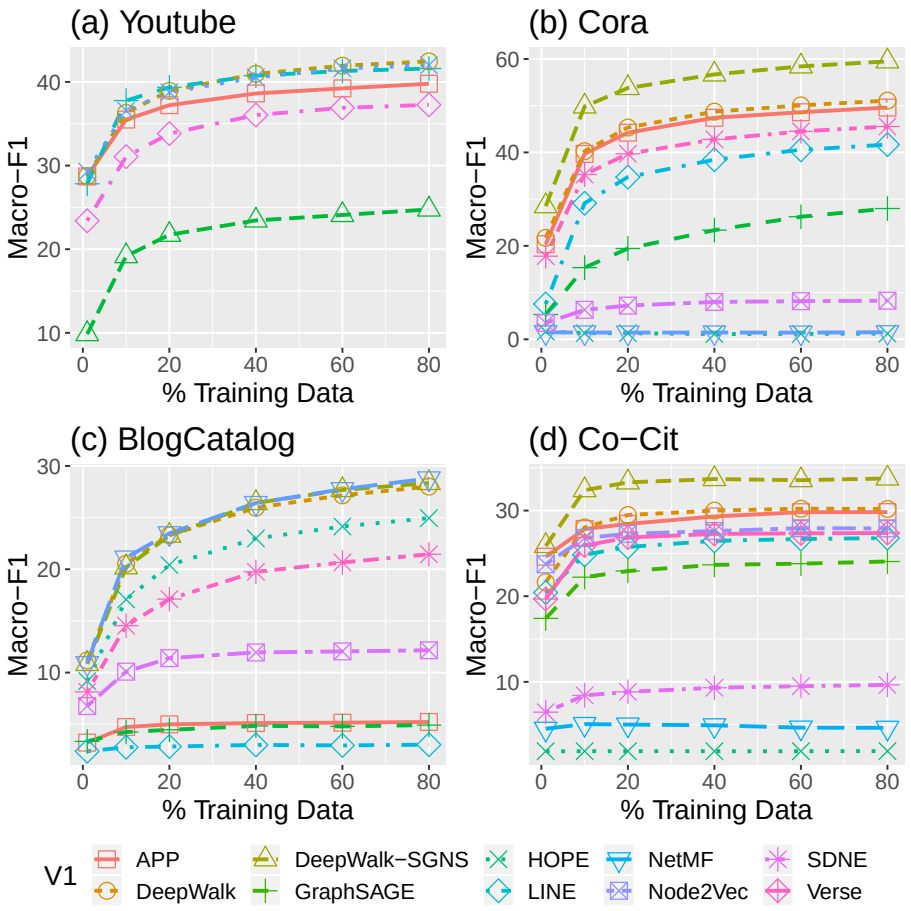


Figure 1: Learning rate with increasing Training data size. All runs are averaged over 5 splits of 20% Test data.

GraphSAGE still continues to learn with increasing training data for Cora and BlogCatalog. On the contrary most of the approaches for CoCit already converge to their final performance values at 20% of training data. This suggests, contrary to small training datasets in earlier works that consider as small as 1% training data size, one should at least consider at least 20% of training data while reporting performance values.

**Threats to validity.** We chose the datasets in a manner that at least one of the datasets is used in the paper for an approach. We further chose to experiment with the authors implementation as much as possible except for APP and SDNE. We also were able to replicate the results mentioned in the original paper except SDNE for NC task on BlogCatalog and Flickr. Finally, we re-trained models as and when necessary and made them stronger using

newer datasets or reverted to the best parameters suggested in the original papers. However, we did not explore all hyper-parameters in all approaches due to their sheer combinatorial search space. We report and verify all the structural properties as mentioned in Konect [**?**] and compute those which are missing.

## 4.1 Advice to practitioners

In employing node embeddings for tasks like node classification and link prediction some of the key aspects to bear in mind in the choice of the approach are the following.

1. When considering an undirected graph for link prediction PPR based methods such as VERSE are efficient and robust that ignores the traditional context modeling.

2. For doing link predictions in directed graphs almost always node and context embedding pairs like APP and HOPE should be preferred. Only in cases when the reciprocity of the graph is high the other approaches become competitive. In terms of evaluation one should carefully construct test sets with negative edges as reversed positive edges to evaluate directionality.

3. For node classification the degree of homophily should be precomputed and that should drive the choice of the method. For high degree of label homophily among neighboring nodes neighborhood aggregation based deep learning approaches outperform others, while DEEPWALK is a robust choice for low homophily graphs.