# Decentralized Embedding Framework for Large-scale Networks

Mubashir Imran[1], Hongzhi Yin[1⋆], Tong Chen[1], Yingxia Shao[2], Xiangliang Zhang[3], and Xiaofang Zhou[1]

[1] School of Information Technology and Electrical Engineering, The University of Queenslands
m.imran@uq.net.au,{h.yin1,tong.chen}@uq.edu.au,zxf@itee.uq.edu.au
[2] Beijing University of Posts and Telecommunications shaoyx@bupt.edu.cn
[3] King Abdullah University of Science and Technology
xiangliang.zhang@kaust.edu.sa

**Abstract.** Network embedding aims to learn vector representations of vertices, that preserve both network structures and properties. However, most existing embedding methods fail to scale to large networks. A few frameworks have been proposed by extending existing methods to cope with network embedding on large-scale networks. These frameworks update the global parameters iteratively or compress the network while learning vector representation. Such network embedding schemes inevitably lead to a high cost of either high communication overhead or sub-optimal embedding quality. In this paper, we propose a novel *decentralized large-scale network embedding framework* called *DeLNE*. As the name suggests, DeLNE divides a network into smaller partitions and learn vector representation in a distributed fashion, avoiding any unnecessary communication overhead. Our proposed framework uses Variational Graph Convolution Auto-Encoders to embed the structure and properties of each sub-network. Secondly, we propose an embedding aggregation mechanism, that captures the global properties of each node. Thirdly, we propose an alignment function, that reconciles all sub-networks embedding into the same vector space. Due to the parallel nature of DeLNE, it scales well on large clustered environments. Through extensive experimentation on realistic datasets, we show that DeLNE produces high-quality embedding and outperforms existing large-scale network embeddings frameworks, in terms of both efficiency and effectiveness.

**Keywords:** Network Embedding · Distributed System · Auto-encoder · Embedding Alignment

## 1 Introduction

Learning continuous low-dimensional vector representations of nodes in a network has recently attracted substantial research interest. Data from networks

⋆ Corresponding author; contributing equally with the first author.

such as E-commerce platforms, scholarly libraries, social media, medicine, service providers [2, 5, 24, 25] etc. in its raw form is not directly applicable to emerging Machine Learning (ML) approaches, as they requires low-dimensional vector representations for computation. Formulating vector representations of these data networks, which can be utilized as an input to many ML systems, is identified as "Network Embedding" [1]. High-quality network embedding is imperative for accurately performing network inference tasks, e.g. link prediction, node classification, clustering, visualization and recommendation. State-of-the-art network embedding methods like node2vec, Line, DeepWalk and SDNE [3, 17, 20, 22] effectively preserve the network structure properties. However, in the case of large-scale networks, especially the dynamically growing networks, being confronted with more sophisticated network structure, these state-of-the-art network embedding methods fall short on both computational efficiency and embedding quality.

To tackle the crucial task of efficiently producing network embedding on large networks, some large-scale network embedding frameworks have been proposed. These frameworks mostly perform the task by: 1) *coarsening* the network (i.e. dividing it into smaller chunks), 2) *sharing the embedding parameters* and 3) utilizing *matrix multiplication* to reduce data dimensions. Yet, existing network coarsening techniques can hardly guarantee the embedding quality, as the degree of coarsening increases [12], due to loss of information (i.e. local structure). Sharing of global parameters introduces immense communication overhead, while matrix multiplication-based approaches incur high memory cost. These frameworks are further described in Section 5.

With the increasing availability of large-scale distributed systems and cloud-based resources, another possible approach to address the large-scale network embedding problem is to carry out representation learning in a decentralized fashion. Decentralized scheme is beneficial for effective utilization of cloud-based resources. It is faster to generate node embedding due to parallel computation of gradients, while avoiding synchronization of global parameters and being easily scalable in the case growing networks. But before we can enjoy the benefits of decentralized scheme, we must solve the following three challenges imposed: 1) producing quality partitions, 2) preserving network properties in the absence of global parameters, and 3) aligning the distributively learned node representations into the same vector space.

Commonly, large networks can be expressed as a collection of smaller well-defined communities [13], interconnected through border nodes, known as *anchor nodes* [8]. For example, in an academic network, if a scholar collaborates frequently with researchers from two different research areas, this scholar can be regarded as the anchor node between two research communities. Anchor nodes act as bridges in order to diffuse information between communities [8].

Motivated by the fact that *large networks consist of numerous communities* that share a variety of *anchor nodes*, we present *Decentralized Large-scale Network Embedding Framework (DeLNE)*. DeLNE learns node representations from large-scale networks in a distributed fashion. Our proposed model aims

to address the limitations of existing techniques by avoiding excessive communication overhead caused by parameter sharing. DeLNE identifies the optimal number of divisions for a given large-scale network, based on min-edge cut. It partitions the network into multiple well-defined communities with high neighborhood densities. Our framework then learns node embedding of each partition independently in a distributed environment. It then realigns the learned node representations into the same embedding space, by learning an alignment function. DeLNE stands out in its high scalability, even if the size of any partition grows or new sub-networks are introduced to the network. The main contributions of this paper are threefold:

- We advance the existing network embedding methods to cope with very large-scale networks in a parallel and distributed fashion. This allows for efficient computation on very large networks while preserving properties and structural information.
- In DeLNE, we present an aggregation function that captures the global properties of the sub-network, by actively fusing the information of neighbour nodes. We present an embedding alignment scheme that refines the distributed embedding onto the same embedding space.
- We conduct extensive experimentation on large-scale networks, to evaluate the effectiveness and efficiency of DeLNE. Experiments suggest that DeLNE outperforms the existing state-of-the-art and large-scale network embedding techniques.

## 2 Preliminaries

### 2.1 Notations and Definitions

In this section, we first explain the notations adopted throughout the paper (Table 1). For the ease of understanding, we also formally define the key technical terms in this paper as follows.

**Table 1.** The notations adopted in the paper

| Notation | Definition |
|---|---|
| $\mathcal{G}$ | An undirected network |
| $\mathcal{V}$ | A set of vertices belonging to network $\mathcal{G}$ |
| $\mathcal{E}$ | A set of edges belonging to network $\mathcal{G}$ |
| $\mathcal{F}$ | A set of features associated with $\mathcal{G}$ |
| $v_i$ | $i^{th}$ vertex $\in \mathcal{V}$ |
| $e_i$ | $i^{th}$ edge $\in \mathcal{E}$ |
| $d$ | Depth of network $\mathcal{G}$ from a vertex $v$ |
| $\mathcal{G}_S$ | **Super network:** Compressed representation of complete network |
| $\{\mathcal{G}_1, \mathcal{G}_2, ..\mathcal{G}_k\}$ | A $k$ way partitioned network |
| $v_a$ | Anchor node (a node having an edge in adjacent partition) |
| $\mathbf{A}$ | An adjacency matrix of the network $\mathcal{G}$ |
| $\mathbf{Z}_i$ | Embedding of sub-network $i$ |
| $\mathbf{z}$ | An embedded vector |

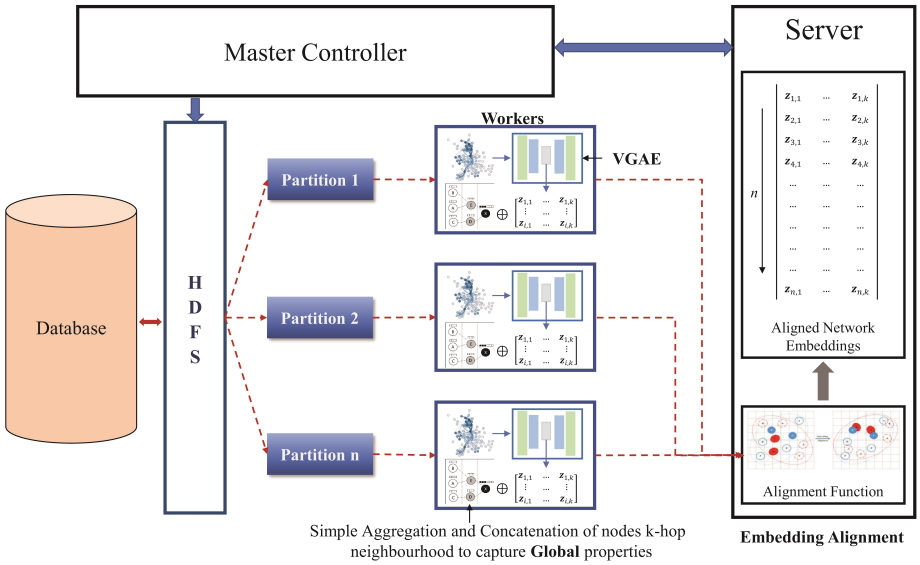


**Fig. 1.** The overview of DeLNE framework.

***Definition 1 (Super Network):*** *a network $\mathcal{G}$ constructed by collapsing its nodes (i.e. $\{\forall v \in \mathcal{G}\}$) using Heavy Edge Matching [12]. A single node in $\mathcal{G}_S$ represents numerous nodes in the original $\mathcal{G}$.*

***Definition 2 ($k^{th}$-Order Walk):*** *augmenting the neighbourhood $S$ of a vertex ($v \in \mathcal{G}$), from d=1 down to d=k. This augmentation is carried out in a breadth-first (BFS) search manner.*

***Definition 3 (Anchor Node):*** *the border nodes ($v_a$) of a partitioned network. An anchor node $v_a$ has an edge $e_a$ (i.e. anchor link) to the adjacent sub-network, connecting one partition to another.*

### 2.2 Problem Definition

Given a large-scale network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ having dimensions ($\mathbf{d} \ll |\mathcal{V}|$), we aim to learn node representations of a network $\mathcal{G}$ for its $k$ partitions $\{\mathcal{G}_1, \mathcal{G}_2, ..\mathcal{G}_k\}$ in a distributed fashion. We also want to learn an alignment function $M : \mathcal{L}^{|\mathbf{z}_s - \mathbf{z}_t|}$, that aligns (partitioned network using anchor nodes $\mathbf{z}_s$ of source graph to $\mathbf{z}_t$ of target graph) and produces consistent node embeddings of each sub-network. It should also preserve both local and global network structures, as well as the node properties.

## 3 Methodology

To enable parallel computing for large-scale network embedding, DeLNE implements a master worker framework, consisting of four main phases: 1) *network*

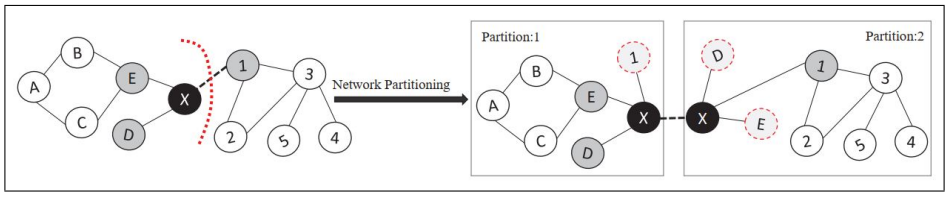


**Fig. 2.** Overlapping network partitioning by modifying multilevel graph partitioning algorithm. Node **X** is an anchor node connecting both partition-1 and partition-2. 1st order neighbour hood of **X** i.e. node **D** and **E** from partition-1 and node **1** from partition-2 are copied to the adjoining partitions.

*partitioning*, 2) local property-preserving *node embedding*, 3) global property-preserving *embedding aggregation*, and 4) *embedding refinement*, i.e. mapping sub-networks into the same embedding space. Fig. 1 presents an overview of the DeLNE architecture. Firstly, the master controller partitions the network using a non-overlapping network partitioning algorithm [9]. This divides the network into $k$ high-quality partitions. These partitions take the shape of communities, based on their structural properties. Two neighborhood partitions are connected via anchor nodes $v_a$. A copy of anchor node and its first-order neighbourhood is kept at both sides of the connecting partitions. Master controller assigns each partition to a unique worker. Each worker employs a Variational Graph Auto-encoder (VGAE) to learn low-dimensional node representations dedicated to a sub-network and a property aggregation function to capture global perspective of the sub-network. Finally, we align the node embedding into the same space by learning an alignment function supervised by the observed anchor nodes.

### 3.1 Network Partitioning

An important property exhibited by very large networks is their community structures property. This translates the network into number of smaller modules called clusters [13]. Each large network consists of several smaller communities that share inherit characteristics or interests, e.g. people with interest in a common sport like football or basketball, followers of a particular political party or opinion etc. It is therefore critical to partition a network, such that nodes sharing similar characteristics are grouped together, against those nodes which exhibit distinct characteristics. In other words, we aim to increase intra-cluster connectivity and decrease the inter-cluster connectivity.

In order to partition the network, we propose a variant of multilevel graph partitioning algorithm (METIS) [6]. As shown in Algorithm 1, multilevel partitioning approach partitions a given graph in three phases, namely coarsening, partitioning and uncoarsening. During the coarsening phase, the algorithm iteratively applies heavy-edge matching (HEM) to compute the edge matching, i.e. finding a contraction between the vertex with the lowest degree and the vertex with the highest weight $(v_1, v_2)$, such that weight between them is maximized.

Since the size of the corset graph is small, multilevel graph partitioning algorithm efficiently apply graph growing heuristic to compute partitioning. Boundary Kernighan-Lin (BKLR) algorithm is used to iteratively refine the corset partitions. DeLNE does not involve sharing of any parameters or communication between the partitions during the learning phase. Rather, we learn the node embedding of each sub-network independently and then map them into the same space. To align the sub-networks and preserve better structural properties of the anchor nodes, we make use of the non-overlapping partitioning (see Fig. 2). Anchor nodes(e.g. node **X** in Fig. 2) are copied to the linked partition (doted line in Fig. 2 represents such link), along with its $1^{st}$-order neighbourhood.

**ALGORITHM 1:** Multilevel Graph Partitioning

**Input:** $\mathcal{G} = (\mathcal{V}, \mathcal{E}), k$
**Output:** $[\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_k]$

**while** $|v \in \mathcal{G}| > k$ **do**
  sort in increasing order of degree($v \in \mathcal{G}$) **for** $\forall\ v \in \mathcal{G}$ **do in Parallel**
    collapse $v$ with neighbour having highest $w$
$[\mathcal{G}_{S1}, \mathcal{G}_{S2}, ...\mathcal{G}_{Sk}] \leftarrow$Partition($\mathcal{G}_S$)
**for** $\forall\ i \leftarrow k$ **do in Parallel**
  $\mathcal{G}_i \leftarrow$BKLR($\mathcal{G}_{Si}$)
  **if** *Neighbour*$(v : (v \in \mathcal{G}_i) \notin \mathcal{G}_i)$ **then**
    $\mathcal{G}_i \cup 1^{st}$order_neighbourhood($v$)
**return** $[\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_k]$

### 3.2 Base Embedding

Our low-dimensional vector representations should effectively preserve both network structure (for supporting the network reconstruction tasks) as well as network properties (for network inference tasks). Commonly used network embedding models include linear models e.g. matrix factorization [21, 23], skip-gram models [3, 17, 20] and non-linear Graph Convolution Networks (GCNs) [1, 4, 7] models. Traditional linear models can be classified as shallow models as they only captures first-order connections. Skip-gram models aim to learn embeddings from linear space, leading to limited expressiveness. Non-linear models e.g. GCNs address the previous problems, but heavily rely on high-quality labeled data and hardly adapt to unsupervised learning. While working with large-scale networks, due to their dynamics and node diversity, it is very difficult to associate high-quality and consistent labels with each individual node. To capture the non-linearity and preserve high-quality embeddings in the absence of labeled data, in DeLNE we employ Variational Graph Auto-Encoder (VGAE) [7], to support unsupervised learning task.

Variational auto-encoder (VAE) modifies traditional auto-encoders by taking in a distribution $q\phi(\mathbf{z}|v)$ rather than data points. By doing so the model is able to adapt with and infer on unseen data. VAE encoder accepts data point

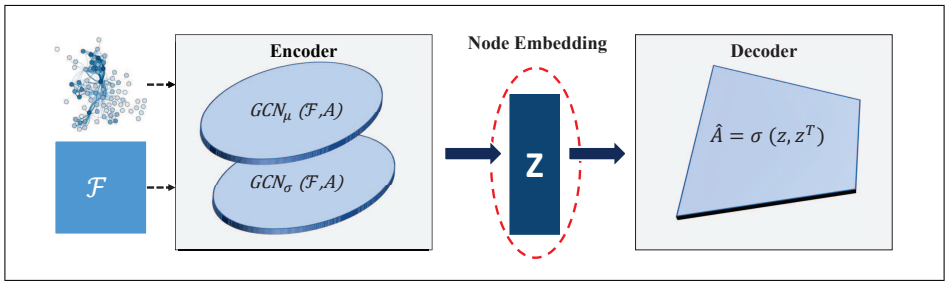


**Fig. 3.** VGAE, having two layer GCN as encoder and a inner product decoder.

$v$ and generates the means and standard deviation of the Gaussian distribution. The lower-dimensional vector representation $\mathbf{z}$ is sampled using this distribution, $q\phi(\mathbf{z}|v)$. The decoder takes in the embedding $\mathbf{z}$ and produces output using variational approximation $p\theta(v|\mathbf{z})$.

To apply the idea of VAE on a network, we deploy a VGAE, that is able to generate as well as predict unseen links and the structure. VGAE consists of two parts: 1)An inference model (encoder) and 2) A generative model (decoder). The inference model takes an adjacency matrix $\mathbf{A}$ and feature matrix $\mathcal{F}$ as its input. It generates the mean and standard deviation through a function $\theta$. As show in Fig. 3, this encoder is constructed using two layers of GCN, given as:

$$\mu = \mathrm{GCN}_\mu(\mathbf{A}, \mathcal{F}) \tag{1}$$

$$\log \sigma = \mathrm{GCN}_\sigma(\mathbf{A}, \mathcal{F}) \tag{2}$$

By combining the two layers together, we get:

$$\mathrm{GCN}(\mathbf{A}, \mathcal{F}) = \tilde{\mathbf{A}}\ Relu(W_0 \mathcal{F} \tilde{\mathbf{A}}) W_1 \tag{3}$$

where, $W_0$ and $W_1$ are the weight matrices of each layer. The generative model consists of the inner product among the latent variables and is given as:

$$p(\mathbf{A}|\mathbf{Z}) = \Pi_{i=1}^{n} \Pi_{j=1}^{n} p(\mathbf{A}_{(i,j)}|\mathbf{z}_i, \mathbf{z}_j) \tag{4}$$

where $\mathbf{A}_{(i,j)} \in \mathbf{A}$ and $p(\mathbf{A}_{(i,j)} = 1|\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$. Here $\sigma(\cdot)$ is a sigmoid function. The loss function for the optimization task, consists of two parts: 1)Reconstruction loss between the input and reconstructed adjacency and 2) Similarity loss, defined by KL-divergence.

$$L = -KL[q(Z|\mathcal{F}, \mathbf{A}||p(Z))] + E_{q(Z|\mathcal{F},\mathbf{A})}[\log(p(\mathbf{A}|\mathbf{Z}))] \tag{5}$$

### 3.3 Embedding Aggregation

Each sub-network exhibits a unique global prospective, depending on its structure. If two sub-networks have similar structures, they should also resemble each

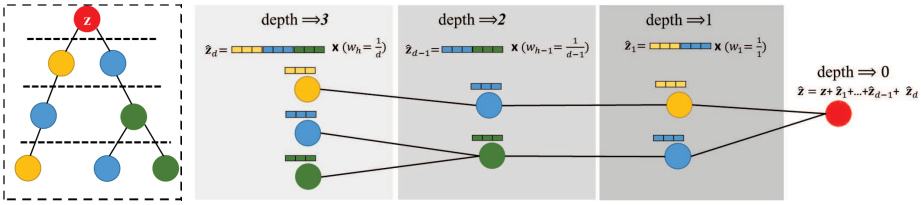


**Fig. 4.** Global aggregation augments walks from each node down to depth $d$, in a BFS fashion.

other in terms of their global characteristics [19]. In order to preserve this global perspective of a sub-network, we introduce an embedding aggregation function. This function aggregates the vector representations of vertices, that are $k^{th}$ order neighbour of $v$ in a breath first search (BFS) fashion, much in the same way as Weisfeiler-Lehman algorithm [19]. The resultant aggregation (i.e. $\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 ... + \hat{\mathbf{z}}_d$) is then concatenated with $\mathbf{z}$, as shown in Fig. 4. This process is repeated for the entire sub-network. At each level of the depth, a weight, inversely proportional to the depth is multiplied with aggregation. The aggregation and concatenation of each vertex is given as:

$$\hat{\mathbf{z}} = \mathbf{z} \oplus \sum_{l=1}^{d} w_l (\sum_{i=1}^{n} \mathbf{z}_{i,l}) \tag{6}$$

where, $\mathbf{z}$ is the base embedding of vertex $v$, $d$ is the depth, $n$ are the number of vertices at $d$ and $w_l = \frac{1}{d}$.

The benefit of this approach is that, for vertices having similar neighbourhood structures, their aggregated embeddings will be close to each other in the vector space. Correspondingly, if two sub-networks are alike in their structures, they will have highly similar node embeddings. In our experiments, we aggregated node embeddings down to $d = 4$.

### 3.4 Embedding Refinement

The key challenge faced while learning network representation in a distributed system is reconciling of sub-networks embeddings into the same embedding space. Directly projecting the embeddings within $k$ partitions into a unified vector space is a straightforward approach. However, it is impractical when dealing with large-scale networks as the huge amount of parameters used for embeddings makes it highly demanding on computational resources. So, in what follows, we describe an innovative embedding alignment approach.

As discussed earlier, a large network consists of numerous smaller communities [13]. These communities are interconnected with each other through an edge $e_B$, connecting a common (anchor) node $v_a$ [8]. For example, in a social network person $X$ can be a member of both a hockey and a football fan club, making him

the anchor node. In our framework we have partitioned the network in an overlapping fashion, such that we keep a copy of anchor node $v_a$ as $v_s, u_t$ (source and target) at both connected partitions of the network. For example, in Figure 2 node $\mathbf{X}$ can be considered as $v_s$ in partition-1 and $v_t$ in partition-2. Embeddings of each sub-network are constructed independently and possess separate latent representation. Hence they require alignment. Anchor nodes ($v_s, v_t$, having presence in multiple sub-networks) act as a bridge to align one embedding space with the other. Therefore with the help of these anchor nodes we translate one embedding space into the other embedding space, such that the anchor nodes are aligned. As a result of this translation process, all nodes in both sub-networks are aligned with the alignment of the anchor nodes. It is important to note that first-order neighbourhood of an anchor node is copied to each partition (Fig. 2). This helps in preserving similar local properties of the same (anchor) nodes belonging to different partitions, during the network embedding phase.

***Embedding Alignment:*** Assume we have a network partitioned into $k$ sub-networks and learned node embeddings $\mathbf{Z}_1, \mathbf{Z}_2, ...\mathbf{Z}_k$ of the respective sub-networks. Without loss of generality, let us assume that we want to align embeddings $\mathbf{Z}_1, \mathbf{Z}_2, ...\mathbf{Z}_k$ into the same embedding space, say $\mathbf{Z}_0$. The purpose here is to learn the mapping function $M$, supervised by anchor nodes $v_0^s, v_k^t \in J$, having embeddings $\mathbf{z}_0^s, \mathbf{z}_k^t$. Here J is a collection of marked anchor links. Motivated by Matrix Translation and Supervised Embedding Space Matching [15] and given $\mathbf{z}_0^s \in \mathbf{Z}_0$, our mapping function projects $\mathbf{z}_0^s \in \mathbf{Z}^0$ and $\mathbf{z}_k^t \in \mathbf{Z}^t$ into same space. Alignment Loss ($L_a$) is given as:

$$L_a = \sum_{u_0^s, u_k^t \in J} \| M(\mathbf{z}_0^s; \mathbf{w}) - \mathbf{z}_k^t \|_F \tag{7}$$

where $\mathbf{w}$ is the vector of weights, such that $\mathbf{z}_0^s \times w$ effectively approximates $\mathbf{z}_k^t$, while $\| \cdot \|_F$ represents Frobenius norm, that gives the distance between source embeddings and target embedding. The parameters of the mapping function $M$ can be obtained by minimizing the loss function given in equation (7), using scholastic gradient descent (SGD). To capture non-linear relationship between $v_0^s$ and $v_k^t$, we employ Multi-Layer Perceptron, having sigmoid $\sigma$ activation at each layer.

On receiving the embeddings from the first two workers, whose sub-networks are connected, we align the node embeddings into the same vector space. As more workers report their learned embeddings they are aligned into the initial space in parallel. Even if we only focus on mapping the anchor nodes, the learned mapping function is able to map the whole embedding space. This comes from the fact that the embeddings of nodes in the same network are considered to be in the same space. As we align the anchor nodes, the mapping function is able to align the whole embedding space as well.

## 4 Experiments

In this section we report our experimental findings on several large-scale datasets to showcase the effectiveness of DeLNE. Particularly, we aim to answer the following research questions:

- How is the **embedding quality** of DeLNE compared with state-of-the-art embedding methods for medium-sized networks?
- When handling large-scale networks, is DeLNE able to effectively **preserve network properties** compared with other large-scale network embedding frameworks?
- How **efficient** is DeLNE compared with existing large-scale embedding frameworks?

### 4.1 Experimental Environment

We carry out our experiments on two Linux-based servers. The master controller server has 256-GB of RAM, 40 CPU cores and NVIDIA GPU (GeForce GTX 1080 Ti), is utilized as the master node. Worker nodes are deployed as separate virtual machines on another server with 1024 GB RAM and 80 CPU cores.

### 4.2 Dataset

We test and compare our framework on five real-world datasets from different domains, as presented in Table 2. Wiki [23] contains 2,405 documents, having 17,981 links and 19 classes. Github [18] is a social network consisting of github developers, where edges represents following relationship. BlogCatalog [26] is a directory of social blog that manages the bloggers and their blogs. The YouTube [16] dataset consists of friendship groups and consists of over 1.1 million users and 4.9 million links. Flickr [16] dataset consists of edges that are formed between images shared between friends, submitted to similar galleries or groups etc.

**Table 2.** Statistics of datasets used in our experiments.

| **Dataset** | **Wiki** | **Github** | **BlogCatalog** | **Youtube** | **Flickr** |
|---|---|---|---|---|---|
| **#Nodes** | 2,405 | 37,700 | 88,784 | 1,134,890 | 1,715,255 |
| **#Edges** | 17,98 | 289,003 | 4,186,390 | 4,945,382 | 22,613,981 |
| **#Class Labels** | 19 | 2 | 39 | 47 | 20 |

### 4.3 Experimental Settings

To adjust the hyper-parameters, we adjust different settings of DeLNE and evaluate the performance on YouTube network. It is a large-scale network, having a large number of labeled nodes, making node classification task harder. For this reason YouTube network is widely adopted to test the performance by large-scale network embedding frameworks [11, 12, 27, 29]. To identify the optimal number of anchor nodes, we conduct our experiments by altering the number of anchor

nodes, kept at each partition. These anchor nodes are selected based on a decreasing order of their degrees. As indicated in Table 3, keeping 500 bridging nodes as anchor nodes outperforms other settings. We also run our experiments by altering the number of network partitioning ($\gamma$). For medium scale networks, we adjust $\gamma$ = 10,20 and 30. For Large graphs, we set $\gamma$ = 100,150 and 200.

**Table 3.** Effectiveness and efficiency of DeLNE for different setting of anchor nodes, on YouTube dataset.

| **Number of Anchor Nodes** | 0 | **500** | 1000 | 2000 |
|---|---|---|---|---|
| **Accuracy** | 0.27 | **0.51** | 0.47 | 0.42 |
| **Training Time** | 2.06 | **2.36** | 2.85 | 3.55 |

**Table 4.** Running time for network embedding on YouTube dataset. For DeLNE running time refers to the sum of training time at each cluster, global properties aggregation time and Embedding Alignment Time.

| **Embedding Framework** | **GPU** | **Clusters** | **Training Time (min)** |
|---|---|---|---|
| LINE | - | 1 | 92.2 |
| DeepWalk | - | 1 | 107.8 |
| SDNE | 1 | 1 | memory error |
| COSINE(DeepWalk) | - | 1 | 10.83 |
| COSINE(LINE) | - | 1 | 4.17 |
| GraphVite | 1 | 1 | 4.12 |
| DeLNE-100 | 1 | 100 | 2.36 |
| DeLNE-150 | 1 | 150 | **2.30** |
| DeLNE-200 | 1 | 200 | **2.30** |

### 4.4 Baselines

Given below are the state-of-the-art network embedding methods, we employ as baseline to compare effectiveness and efficiency of DeLNE. Experiments are conducted using both computationally expensive embedding methods as well as large-scale network embedding frameworks.

- **DeepWalk** [17] constructs node sequences by employing random walks to learn embeddings.
- **LINE** [20] optimizes the objective function of edge reconstruction in-order to construct embeddings. parameters were set as;
- **SDNE** [22] preserves non-linear local and global properties using a semi-supervised deep model.
- **COSINE** [27] learns embeddings on top of existing state-of-the-art embedding methods, using parameter sharing on partitioned networks.
- **SepNE** [11] is a flexible, local and global properties preserving network embedding algorithm which independently learns representations for different subsets of nodes.
- **GraphVite** [29] is a CPU/GPU hybrid system, that augments random walks on CPU in a parallel fashion. On GPU‘s it trains node embedding simultaneously by deploying state-of-the-art embedding methods.

### 4.5 Effectiveness of DeLNE as Compared to Traditional Embedding Methods

To test the effectiveness of our proposed model we deploy multi-label classification task. The main idea is how well a given model can predict whether a node belongs to a certain community (label) or not. We test our model against state-of-the-art embedding models using medium sized datasets i.e. Wiki, Github and BlogCatalog. As for larger datasets, which are computationally expensive and slower to train on state-of-the-art models, we employ large-scale embedding frameworks.

**Table 5.** Results of Multi-label prediction on Wiki, Github and BlogCatalog (BC), as compared with state-of-the-art embedding methods (micro-averaged F1 scores).

| | **Wiki** | | **Github** | | **BlogCatalog** | |
|---|---|---|---|---|---|---|
| **Labeled Nodes** | **10%** | **90%** | **10%** | **90%** | **10%** | **90%** |
| **DeepWalk** | 0.3265 | 0.4357 | 0.7417 | 0.7387 | 0.1578 | 0.2030 |
| **Line $1^{st}$order** | 0.3265 | 0.4357 | 0.7630 | 0.7689 | 0.1467 | 0.1989 |
| **Line $2^{nd}$order** | 0.3307 | 0.3307 | 0.7392 | 0.7469 | 0.1139 | 0.1533 |
| **SDNE** | 0.5044 | 0.6237 | 0.7766 | 0.7745 | 0.1734 | 0.1857 |
| **DeLNE-10** | **0.6575** | **0.6596** | 0.8397 | 0.8651 | 0.2151 | 0.2254 |
| **DeLNE-20** | 0.6486 | 0.6502 | **0.8405** | **0.8711** | **0.2405** | 0.2556 |
| **DeLNE-30** | 0.6331 | 0.6408 | 0.8331 | 0.8625 | 0.2403 | **0.2728** |

As Table 5 shows, our method out performs state-of-the-art network methods for all three datasets. This is because the structure of partitions plays an important role in identifying the node labels i.e, if the partitions form dense communities, better performance will be observed from our framework. For the smallest dataset, i.e. Wiki, DeLNE-10 outperforms all. DeLNE-20 works best for medium sized dataset.

### 4.6 Effectiveness of DeLNE as Compared to Large-scale Embedding Frameworks

To compute the effectiveness of our model for large-scale datasets, we predict labels for networks consisting of millions of nodes and billion of edges. To compare our results with state-of-the-art methods we employ large-scale embedding frameworks, since these datasets are computationally expensive.

Both for Youtube and Flickr networks, as shown in Table 6 and Table 7, DeLNE outperforms SepNE, COSINE and GraphVite. For Flickr network, DeepWalk and LINE settings of GraphVite slightly outperforms DeLNE. But as the number of labeled nodes are increased, DeLNE surpasses all the other frameworks.

### 4.7 Time Efficiency

Here we compare DeLNE with other large-scale network embedding methods regarding training efficiency. The time in Table 5, depicts training time plus

**Table 6.** Results of Multi-label prediction on Youtube (micro-averaged F1 scores).

| Labeled Nodes | 1% | 3% | 5% | 10% |
|---|---|---|---|---|
| **SepNE** | 0.2253 | 0.3361 | 0.3620 | 0.3882 |
| **COSINE(DeepWalk)** | 0.3650 | 0.4110 | 0.4230 | 0.4400 |
| **COSINE(LINE)** | 0.3631 | 0.4160 | 0.4271 | 0.4441 |
| **GraphVite (LINE)** | 0.3836 | 0.4217 | 0.4444 | 0.4625 |
| **GraphVite (DeepWalk)** | 0.3741 | 0.4212 | 0.4447 | 0.4639 |
| **DeLNE** | **0.4132** | **0.4360** | **0.4892** | **0.5130** |

**Table 7.** Results of Multi-label prediction on Flickr (micro-averaged F1 scores).

| Labeled Nodes | 1% | 3% | 5% | 10% |
|---|---|---|---|---|
| **SepNE** | 0.4269 | 0.4468 | 0.4562 | 0.4623 |
| **COSINE(DeepWalk)** | 0.4040 | 0.4140 | 0.4190 | 0.4230 |
| **COSINE(LINE)** | 0.4080 | 0.4180 | 0.4240 | 0.429 |
| **GraphVite (LINE)** | 0.6103 | **0.6201** | **0.6259** | 0.6305 |
| **GraphVite (DeepWalk)** | **0.6125** | 0.6144 | 0.6216 | 0.6298 |
| **DeLNM-100** | 0.5933 | 0.6136 | 0.6201 | **0.6312** |

aggregation time (maximum of all the workers) and network alignment time. It should be mentioned here that we do not compare time of SepNE, since it focuses on learning embeddings of certain components of a network and invites a sizeable computation overhead encase of embedding the entire network. By running efficiency tests on two large-scale datasets (YouTube and Flickr), we show our frame work outperforms all the base lines. We can also see that varying $\gamma$ in $\{100, 150, 200\}$, for DeLNE, does not improve efficiency too much.

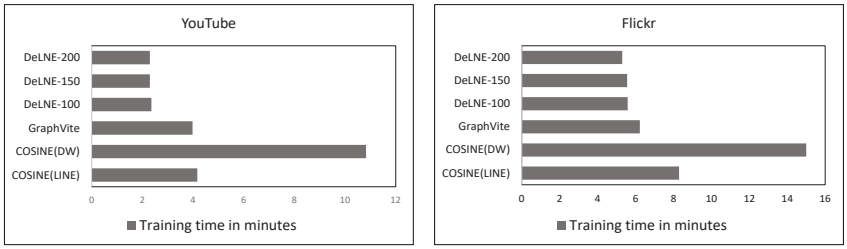


**Fig. 5. A:** Total Time in minutes taken to compute embeddings for Youtube dataset. **B:**Total Time in minutes taken to compute embeddings for Flickr dataset.

## 5 Related Work

State-of-the-art embedding methods can be divided in to two categories, namely structure-preserving and property-preserving embedding methods. Structure preserving methods such as node2vec [3] and DeepWalk [17] samples k-order random

walks on each node of a network. These walks are treated as sentences and Skip Gram is applied to obtain embedding. LINE [20] captures first and second-order proximity to learn embedding. SDNE [22], not only captures second-order proximity, but also non-linear structure using deep neutral networks. Property preserving embedding methods, such as MMDW[21] and TADW[23] deploy matrix factorization-based techniques to embed networks.

Large-scale network embedding frameworks, such as MILE [12], coarsens a very large network to perform network embedding and refine the embedding iteratively. This mechanism captures the global properties and transfers them down through refinement. A key limitation of this technique lies in it's ability to deal with the increasing network size. In this approach, the quality of embedding produced suffers, as the size of the network increases. In addition, refining embedding down from a super-node to the original nodes, is computationally expensive for large networks. Another approach to embed very large networks is parameter sharing. This technique is utilized by COSINE, GraphVite and PyTorch-BigGraph (PGB) [10, 27, 29]. These frameworks adopt non-overlapping network partitioning methods to create smaller partitions with distinct vertices. They update the global parameter, at each iteration of learning phases. A limitation to these approaches is there dependence on the bus-bandwidth. As the network size increases, the communication cost of updating the parameters over multiple iterations, aggravates.

Another variant network embedding method, Gaussian Random Projection (GRP), applied by RandNE [28] preserves high order structure using distributed multiplication, on the proximity matrix $S$. However, computing $S$ on a single machine still remains expensive, for large networks. In parallel settings, this technique requires each computing machine to keep a complete copy of a network's adjacency matrix. This requirement is not viable when embedding a very large network.

In contrast, DeLNE utilizes parallelism and distributive computation to increase efficiency, for large-scale networks. It preservers non-linearity using VGAE while preserving global properties through an aggregation function. DeLNE also aligns the embedding into same vector space, to insure the consistency of sub-networks embedding.

## 6 Conclusion

In this paper, we presented a novel decentralized large-scale network embedding framework, called DeLNE, which divides a network into multiple dense partitions and performs node embedding in a parallel manner over distributed servers. Our proposed framework uses Variational Graph Convolution Auto-Encoders to embed structure, as well as local and global properties of each sub-network into the vector space. In order to construct consistent embeddings of the entire partitioned network, while avoiding parameter sharing overhead, we learn a network alignment function. The alignment function maps the node embeddings received from distributed servers onto the same embedding space. Through extensive ex-

perimentation on real world datasets, we show that DeLNE produce high quality embedding and outperforms state-of-the-art as well as large-scale network embedding frameworks in terms of efficiency and effectiveness.

**Acknowledgement.** This work is supported by Australian Research Council (Grant No. DP190101985, DP170103954) and National Natural Science Foundation of China (Grant No. U1936104 and 61702015).

## References

1. Chen, H., Yin, H., Chen, T., Nguyen, Q.V.H., Peng, W.C. and Li, X.: Exploiting centrality information with graph convolutions for network representation learning. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 590–601 (2019)
2. Chen, H., Yin, H., Wang, W., Wang, H., Nguyen, Q.V.H., Li, X.: Pme: projected metric embedding on heterogeneous networks for link prediction. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1177–1186 (2018)
3. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864. ACM (2016)
4. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. pp. 1024–1034 (2017)
5. Imran, M., Akhtar, A., Said, A., Safder, I., Hassan, S.U.,Aljohani, N.R.,: Exploiting social networks of Twitter in altmetrics big data. In: 23rd international conference on science and technology indicators (STI 2018)
6. Karypis, G., Kumar, V.: Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0 (1995)
7. Kipf, T.N., Welling, M.: Variational graph auto-encoders. NIPS Workshop on Bayesian Deep Learning (2016)
8. Kong, X., Zhang, J., Yu, P.S.: Inferring anchor links across multiple heterogeneous social networks. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 179–188. ACM (2013)
9. LaSalle, D., Patwary, M.M.A., Satish, N., Sundaram, N., Dubey, P., Karypis, G.: Improving graph partitioning for modern graphs and architectures. In: Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms. p. 14. ACM (2015)
10. Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrstedt, L., Bose, A., Peysakhovich, A.: Pytorch-biggraph: A large-scale graph embedding system. In: Proceedings of The Conference on Systems and Machine Learning (2019)
11. Li, Z., Zhang, L., Song, G.: Sepne: Bringing separability to network embedding. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 4261–4268 (2019)
12. Liang, J., Gurukar, S., Parthasarathy, S.: Mile: A multi-level framework for scalable graph embedding. arXiv preprint arXiv:1802.09612 (2018)
13. Malliaros, F.D., Vazirgiannis, M.: Clustering and community detection in directed networks: A survey. Physics Reports **533**(4), 95–142 (2013)

14. Man, T., Shen, H., Huang, J., Cheng, X.: Context-adaptive matrix factorization for multi-context recommendation. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. pp. 901–910. ACM (2015)
15. Man, T., Shen, H., Liu, S., Jin, X., Cheng, X.: Predict anchor links across social networks via an embedding approach. In: IJCAI. vol. 16, pp. 1823–1829 (2016)
16. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. pp. 29–42. ACM (2007)
17. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
18. Rozemberczki, B., Allen, C., Sarkar, R.: Multi-scale attributed node embedding (2019)
19. Shervashidze, N., Schweitzer, P., Leeuwen, E.J.v., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(Sep), 2539–2561 (2011)
20. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. pp. 1067–1077. International World Wide Web Conferences Steering Committee (2015)
21. Tu, C., Zhang, W., Liu, Z., Sun, M., et al.: Max-margin deepwalk: Discriminative learning of network representation. In: IJCAI. vol. 2016, pp. 3889–3895 (2016)
22. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234. ACM (2016)
23. Yang, C., Liu, Z., Zhao, D., Sun, M., Chang, E.: Network representation learning with rich text information. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
24. Yin, H., Zou, L., Nguyen, Q.V.H., Huang, Z., Zhou, X.: Joint event-partner recommendation in event-based social networks. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 929–940. IEEE (2018)
25. Yin, H., Wang, Q., Zheng, K., Li, Z., Yang, J., Zhou, X.: Social influence-based group representation learning for group recommendation. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 566–577. IEEE (2019)
26. Zafarani, R., Liu, H.: Social computing data repository at ASU (2009), http://socialcomputing.asu.edu
27. Zhang, Z., Yang, C., Liu, Z., Sun, M., Fang, Z., Zhang, B., Lin, L.: Cosine: Compressive network embedding on large-scale information networks. arXiv preprint arXiv:1812.08972 (2018)
28. Zhang, Z., Cui, P., Li, H., Wang, X., Zhu, W.: Billion-scale network embedding with iterative random projection. In: 2018 IEEE International Conference on Data Mining (ICDM). pp. 787–796. IEEE (2018)
29. Zhu, Z., Xu, S., Tang, J., Qu, M.: Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In: The World Wide Web Conference. pp. 2494–2504. ACM (2019)