Deep Kernel Network Embedding

Bo Zhang, Xiaoming Zhang, *Member, IEEE*, Feiran Huang, Ming Lu, and Shuai Ma

Abstract—This paper concerns the problem of network embedding (NE), whose aim is to learn a low-dimensional representation for each node in networks. We shed a new light to solve the sparsity problem where most of nodes including the new arrival nodes have little knowledge with respect to the network. A novel paradigm is proposed to integrate the multiple heterogeneous information from the subgraphs covering the target node instead of only the target node. Particularly, a probabiltiy distribution in subgraph space is contructed for each node, which is more effective to express the distinctive feature over the vertex domain compared to the traditional shallow representations. We boost NE performance by defining the convolution operation over the subgraph distributions that are efficient to evaluate and learn. Our method expliots the advantages of kernel method and deep learning such that the context semantics of subgraph distributions of nodes with dense links is transferred to the sparse nodes effectively via sharing model parameters. Experiments on four real-world network datasets demonstrate that our approach significantly outperforms state-of-the-art methods, especially on the representation learning for the nodes newly joining in the network.

Index Terms—Network embedding, kernel method, kernel mean embedding, convolutional neural network

1 Introduction

Information networks are ubiquitous in the real world with examples such as social and communication networks, paper citation networks and the World Wide Web. One of the fundamental problems, i.e. Network Embedding (NE), is how to learn useful network representations, in which each vertex is presented as a low-dimensional vector [1], [2], [3]. As a result, mining information in networks, such as information retrieval [4], classification [5], link prediction [6] and network alignment [7], [8], can be directly conducted in the low-dimensional space.

In recent years, numerous network embedding methods have been proposed [1], [9], [10], which leverage the adjacency information to constrain the similarity of the latent representations of a pair of vertices. However, many real-world networks are sparse in which most of nodes have little knowledge with respect to the links. The sparsity problem challenges the traditional NE methods, which means that only utilizing the very limited observed links is not enough to obtain a satisfactory performance [11]. One line research of handling sparsity is to introduce node attribute as the complementary information to the structural topology in NE [12], [13]. The different types of information contribute to the representation in a complementary manner. On the other side, the collective knowledge also provides clues for handling sparsity [14], [15]. For example, the nodes may have similar structure and attribute information with the neighbors or other nodes within the same community. It is reasonable to exploit the information existing in the neighbors [16] or the community [17] to improve the representation learning for the nodes with sparse information. Moreover, in dynamic networks, there exist some new nodes that arrive and join in the network at any time. The arrival nodes suffer from the sparsity problem more seriously since they have little information of both the structure and attribute about the networks. How to efficiently learn high-quality representations for them without retraining the model is an important problem as well [18].

In this paper, we provide a new train of thought and method for solving the sparsity problem. We observe that the complete geometric structures and semantics of a node may be preserved in some substructures, i.e, subgraphs. For example, a node connects with the neighbors to form the neighborhood subgraphs of the node, while from a high-level view, a node usually belongs to some community subgraphs where the nodes have dense connections internally. Therefore, a probability distribution $\mathbf{P}^v$ over the subgraph space can be formulated to represent the node v. A collection of v's subgraphs constructed from the network can be viewed as many instances obtained from the distribution $\mathbf{P}^v$. Based on this assumption, a learnt function f (e.g., a neural network) can be designed to identify the key feature by computing the expectation over $\mathbf{P}^v$, i.e., $\mathbb{E}_{G\sim\mathbf{P}^v} f(G)$. The features filtered by multiple functions form the final node representations. This strategy has several advantages over traditional ideas [1], [9] with respect to solving the sparsity problem. First, the distribution of each node is more capable to express the distinctive characteristics and features over vertex domain in a higher dimension compared to the shallow

- *Bo Zhang, Xiaoming Zhang, and Ming Lu are with the Key Laboratory of Aerospace Network Security, Ministry of Industry and Information Technology, School of Cyber Science and Technology, Beihang University, Beijing 100191, China. E-mail: {zhangnet, yolixs, luming0413}@buaa.edu.cn.*
- *Feiran Huang is with the College of Cyber Security, Jinan University, Guangzhou, Guangdong 510632, China. E-mail: huangfr@ieee.org.*
- *Shuai Ma is with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China. E-mail: mashuai@buaa.edu.cn.*

Manuscript received 9 February 2020; revised 8 February 2022; accepted 15 February 2022. Date of publication 23 February 2022; date of current version 1 May 2023.
This work was supported in part by the Beijing Natural Science Foundation of China under Grant 4182037, and in part by the National Natural Science Foundation of China under Grant 61925203.
(Corresponding author: Xiaoming Zhang.)
Recommended for acceptance by X. Xiao.
Digital Object Identifier no. 10.1109/TKDE.2022.3153053

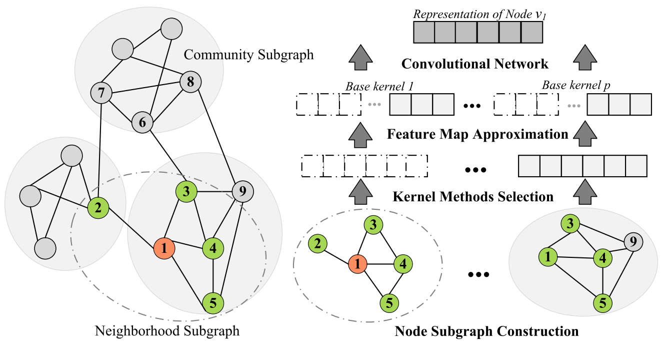

Fig. 1. An illustration of the proposed paradigm. The neighborhood and community subgraphs are constructed for v_1. We introduce kernel methods and sketching techniques to approximate feature maps based on different base kernels for each constructed graph. Then, these feature maps are integrated into convolutional neural networks to learn node representation of v_1.

representation vectors. Second, it defines a unified function f to process all types of the heterogeneous information including attributes, links and the collective knowledge. Compared to other methods that define multiple functions for different types of information [19], [20], it is adaptable and extendible to learn the latent correlation among heterogeneous information simultaneously. Last, it is also effective to embed new nodes. We take the distribution instead of shadow sparse information (e.g., adjacency matrix) as the input, where the non-linear structures of new nodes can be captured without retraining the model.

The primary bottlenecks of our strategy are the definition of the function f and the evaluation of the expectation $\mathbb{E}_{G\sim\mathbf{P}^v}f(G)$. Kernel methods [21], [22], [23] have shown promising results in the task of measuring the similarity between two objects. Roughly, the kernel methods first project the objects into feature map vectors and then compute an inner product between the vectors in reproducing kernel Hilbert space (RKHS). Kernel mean embedding (KME) [24] has recently emerged as a powerful tool to extend the feature map to the space of probability distributions. Building upon kernel methods and KME, the expectation $\mathbb{E}_{G\sim\mathbf{P}^v}f(G)$ can be evaluated as the inner product between the feature maps of $\mathbf{P}^v$ and the function f in RKHS. Meanwhile, convolution offers an efficient architecture to extract highly meaningful statistical patterns in large-scale and high-dimensional datasets. Therefore, the function f is designed as a convolution over the probability distribution $\mathbf{P}^v$ to learn a unique representation for each node.

A novel paradigm consisting of four components is proposed to realize our strategy efficiently: (i) constructing neighborhood and community subgraphs for each node; (ii) an effective kernel function to project the distribution of each node into explicit feature map according to the type of graph; (iii) approximating the explicit feature maps with various base kernels by sketching technique [25], [26], such that the convolutional operator can be performed in a low-dimension space; (iv) a deep convolution architecture to collapse all the feature maps into a unique representation. Fig. 1 illustrates an example of our paradigm. To verify our paradigm, we instantiate it as a deep Kernel Network Embedding (KNE) model to support the attribute networks. We use KONG kernel [23] to embed the subgraphs into RKHS and approximate the feature maps of the distribution $\mathbf{P}^v$. Also, the random walk strategy and triplet network [27] are also introduced to capture different-order proximities and community structure. We obtain precise bounds of the approximation accuracy and computational complexity for representation generation. The main contributions are outlined as follows:

- We shed a new light to solve sparsity problem by transforming the vertex domain into a probability distribution. In this assumption, the convolution operation over distributions is defined to process multiple types of the heterogeneous information in a unified manner.
- We propose a novel paradigm to evaluate the expectation $\mathbb{E}_{G\sim\mathbf{P}^v}f(G)$ efficiently. It is highly flexible and can be expended to model various types of networks.
- Extensive experiments are conducted on four real network datasets, and the results demonstrate the superiority of KNE by comparing with state-of-the-art baseline methods.

2 Related Work

2.1 Network Embedding

Network embedding (NE) aims at learning low-dimensional node representations to preserve the network structures. Some recent works introduce representations learning into network embedding. For example, DeepWalk [9] bridges the gap between network embeddings and word embeddings by treating nodes as words and generating short random walk as sentences. Then, neural language model such as Skip-gram [28] is applied on these random walks to obtain network embedding. Many subsequent works on network embeddings have followed the two-phase framework proposed in DeepWalk, with variations in both phases. Based on this framework, LINE [1] and node2vec [2] are proposed respectively with different random walk procedures. The work in [29] proposes a sampling strategy which gradually selects difficult negative context nodes with training process going on to learn node representations. The work in [30] leverages the adversarial training method to improve the robustness of DeepWalk and achieves better generalization ability. VERSE [31] embeds Personalized PageRank algorithm into the framework of Noise Contrastive Estimation where positive samples can be efficiently obtained by simulating α-discounted random walks. The other important line of network embedding is to perform matrix factorization on the proximity matrices to preserve the high-order proximities including LIE [32], HOPE [33], NetMF [34], ProNE [35] and STRAP [36].

Usually, nodes and edges in real-world networks are often associated with additional information called attributes. One line research focuses on exploiting node attributes. TADW [37] incorporates the text feature into the matrix factorization process. The work proposed in [18] leverages node attributes as complementary information to reconstruct the network structural topology of incomplete network. ANSE [19] and STNE [38] respectively introduce deep Multi-Layer Perception (MLP) and sequence-to-sequence (seq2seq) [39] model to fuse the content and structure information from the raw input. Pan *et al.* [40] introduce DeepWalk to handle the attribute networks. DANE [41] learns two autoencoders to capture the high-order proximity in the structure and attribute information. However, these approaches are still not effective to address the problem of sparsity, since the input is mainly the sparse adjacency vector of the target node.

2.2 Kernel Methods and Scale Algorithms

2.2.1 Kernels and Feature Maps

A popular approach to measure the similarity between structured objects is to use kernel methods. Let $\mathcal{X}$ be a nonempty set. The function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a *kernel* if there exist a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ and a map $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that $\forall x, y \in \mathcal{X}$,

$$k(x, y) = <\phi(x), \phi(y)>_{\mathcal{H}},$$

where $\phi(x)$ is the explicit feature map of x and can be written as $k(\cdot, x)$ without ambiguity. Let $x, y \in \mathbb{R}^d$ for $d \geq 1$, and let $p \geq 1$ be an integer and $c \geq 0$ be a positive real value. Then,

$$k(x, y) = (<x, y> + c)^p,$$

is a valid kernel called the Polynomial Kernel. A linear combination of kernels is also a kernel. If the base kernel is valid, then the convolutional kernel is also valid. In a RKHS $\mathcal{H}$, the kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ satisfies the reproducing property, i.e., $<f, \phi(x)>_{\mathcal{H}} = <f, k(\cdot, x)>_{\mathcal{H}} = f(x)$ for $\forall f \in \mathcal{H}$ and $\forall x \in \mathcal{X}$.

The strings can be compared using spectrum string kernel [42]. Let Σ_k^* be the set of all strings with exactly k characters. The string kernel compares the distribution of k-grams between s_1 and s_2 as follows:

$$k(s_1, s_2) := \sum_{t \in \Sigma_k^*} \#_t(s_1) \#_t(s_2),$$

where $\#_t(s)$ is the frequency of t in s. The explicit feature map for the string kernel is thus the k-gram frequency vector $\boldsymbol{\psi}(s) \in \mathbb{N}^{|\Sigma_k^*|}$ such that $\boldsymbol{\psi}_i(s) = \#_t(s)$, where t is the ith k-gram in the explicit enumeration of all k-grams.

Consider a base kernel where $\mathcal{X} = \mathbb{R}^n$ and $\phi : \mathbb{R}^n \mapsto \mathbb{R}^D$. Note here that D can be very large or even infinite. An active area of research [43] is to design scalable algorithms to compute the low-dimensional approximation of the explicit feature map $z : \mathbb{R}^D \mapsto \mathbb{R}^d$ such that $d \ll D$ and $k(\mathbf{x}, \mathbf{y}) \approx <z(\phi(\mathbf{x})), z(\phi(\mathbf{y}))>$, such as Count-Sketch [25] and Tensor-Sketch [26].

2.2.2 Count-Sketch and Tensor-Sketch

We introduce Count-Sketch [25] to achieve scalability. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, we compute sketches $z(\mathbf{x}), z(\mathbf{y}) \in \mathbb{R}^b$ such that $z^\top(\mathbf{x}) z(\mathbf{y}) \approx \mathbf{x}^\top \mathbf{y}$ and $b < d$ controls the approximation quality. A key property of Count-Sketch is that it is a linear projection, i.e., $z(\mathbf{x}) + z(\mathbf{y}) = z(\mathbf{x} + \mathbf{y})$.

Some high-level kernels can capture the non-linear relation. Considering the kernel $(\mathbf{x}^\top \mathbf{y})^p = <\mathbf{x}^{(p)}, \mathbf{y}^{(p)}>, p \in \mathbb{N}_+$, the explicit feature maps $\mathbf{x}^{(p)}$ and $\mathbf{y}^{(p)}$ of $\mathbf{x}$ and $\mathbf{y}$ are their p-level tensor product, i.e., the d^p-dimension vector is formed by taking the product of all subsets of p coordinates of $\mathbf{x}$ or $\mathbf{y}$ [26]. Therefore, computing the explicit feature map and then sketching it using Count-Sketch requires $O(d^p)$ time. Accordingly, using Tensor-Sketch [26], a sketch of size b for a p-level tensor product is computed in time $O(p(d + b \log b))$. The Tensor-Sketch is an effective tool to compute sketches $z^{(p)}(\mathbf{x}), z^{(p)}(\mathbf{y}) \in \mathbb{R}^b$ such that $<z^{(p)}(\mathbf{x}), z^{(p)}(\mathbf{y})> \approx (\mathbf{x}^\top \mathbf{y})^p$ and $b < d^p$ controls the approximation quality. This efficient algorithm motivates the approximation of the the polynomial kernel $(\mathbf{x}^\top \mathbf{y} + c)^p$ and cosine kernel $(\mathbf{x}^\top \mathbf{y} / (\|\mathbf{x}\|_2 \|\mathbf{y}\|_2))^p$ by adding an extra dimension of value $\sqrt{c}$ to all data points and normalizing the points separately.

3 Problem Statement

The input is a graph denoted as $\mathcal{G} = (V, E, C, \ell)$, where $V = \{v_i\}$ represents vertices, $E \subseteq V \times V$ represents the links between vertices, and $C = \{c_i\}$ denotes the attributes of nodes. Node attributes are features that attain an attribute sequence $c_i = (w_1, w_2, \ldots, w_m), m \geq 0$. The labeling function $\ell : V \mapsto \{0,1\}^{\mathcal{L}}$ assigns each vertex a community label, where $\{0,1\}^{\mathcal{L}}$ is the set of $\mathcal{L}$ dimensional 0/1-valued vectors and $\mathcal{L}$ is the the number of communities in the graph. The nth element in vector $\ell(v)$ is an indicator that the vertex v is within the community n. Given two vertex v and u, if $\ell(v) \wedge \ell(u) \neq \mathbf{0}$, it indicates that the vertex v and u share some common communities. Given $G = (V_G, E_G)$ and $H = (V_H, E_H)$, H is the *induced subgraph* of G if $V_H \subseteq V_G$ and E_H consists of all of the edges in E_G that have both endpoints in V_H. For a vertex v, we define its *neighborhood subgraph* as $\mathcal{N}^v = (V_{\mathcal{N}}, E_{\mathcal{N}}, \tau_v)$ which is the induced subgraph of $\mathcal{G}$. The vertex v's neighbors and links between them are included in $\mathcal{N}^v$. The ordering function τ_v defines a fixed order permutation on $V_{\mathcal{N}}$. The graph $\mathcal{M}^v = (V_{\mathcal{M}}, E_{\mathcal{M}})$ is a *community subgraph* of vertex v, if and only if $\mathcal{M}^v$ is the induced subgraph of $\mathcal{G}$, where $\ell(v')_t = \ell(v)_t = 1$ and $\ell(v'')_t \neq 1$ if $\exists t \leq \mathcal{L}$ for $\forall v' \in V_{\mathcal{M}}$ and $\forall v'' \in V - V_{\mathcal{M}}$, and $\ell(u)_j$ is the jth value of $\ell(u)$. There may exist multiple neighborhood and communities subgraphs for each vertex. All the subgraphs of the vertex v are assembled along with the attribute c_v to form a set G^v.

3.1 Definition of Convolution Over Vertex Domain

To address the problem of sparsity, the representation of each node is learnt based on the neighborhood and community subgraphs instead of the nodes. Let $\mathcal{X}_G$ denote the space of subgraphs. Each vertex v can be formulated as a probability measure $\mathbf{P}^v(\mathbf{X})$ on $\mathcal{X}_G$ and the set G^v can be viewed as a set of subgraph instances of v obtained from the probability $\mathbf{P}^v$. Given a specific function $f : \mathcal{X}_G \mapsto \mathbb{R}$, the expectation $\mathbb{E}_{G \sim \mathbf{P}^v}(f(G))$ over the distribution $\mathbf{P}^v$ can be viewed as a feature of v that is identified by a filter f. Therefore, our goal is to define a collection of f denoted as graph filter F to perform convolution over the distribution of the specific vertex, i.e., evaluate the expectation $\mathbb{E}_F[\mathbb{E}_{G \sim \mathbf{P}^v}(f(G))]$. Kernel mean embedding (KME) [24] has recently emerged as a powerful tool to learn statistical features from the probability distributions. The idea of KME is to extend the feature map ϕ to the space of probability distributions. KME represents a probability distribution $\mathbf{P}^v(\mathbf{X})$ by an element in RKHS endowed by a kernel k, defined as follows.

Definition 1 (Kernel Mean Embedding). *Let $\mathbf{P}^v(\mathbf{X})$ denote a Borel probability measure on $\mathcal{X}_G$. The Kernel Mean Embedding of $\mathbf{P}^v$ in $\mathcal{X}_G$ into an RKHS $\mathcal{H}$ endowed with a reproducing kernel k is defined by a mapping $\mu_{\mathbf{P}^v} : \mathcal{X}_G \mapsto \mathbb{R} \in \mathcal{H}$:*

$$\mu_{\mathbf{P}^v} = \mathbb{E}_{G\sim\mathbf{P}^v}[k(\cdot, G)] = \int_{G\in\mathcal{X}_G} k(\cdot, G)\mathrm{d}\mathbf{P}^v.$$

For a positive definite kernel k, the similarity between two vertices v and u can be defined as the inner product of KMEs of their subgraph distributions in RKHS $\mathcal{H}$, i.e.,

$$< \mu_{\mathbf{P}^v}, \mu_{\mathbf{P}^u} >_{\mathcal{H}} = \mathbb{E}_{G\sim\mathbf{P}^v, G'\sim\mathbf{P}^u} k(G, G')$$

Next, we provide the conditions under which the embedding $\mu_{\mathbf{P}^v}$ exists and belongs to $\mathcal{H}$.

Lemma 1 (Existence of $\mu_{\mathbf{P}^v}$). *If $\mathbb{E}_{G\sim\mathbf{P}^v}[\sqrt{k(G,G)}] < \infty$, $\mu_{\mathbf{P}^v} \in \mathcal{H}$.*

Proof. The main proof comes from [44]. The linear operator $T_{\mathbf{P}^v} f := \mathbf{E}_{\mathbf{P}^v} f(G)$ for all $f \in \mathcal{H}$ is bounded under the assumption, since

$$|T_{\mathbf{P}^v} f| = |\mathbb{E}_{\mathbf{P}^v} f(G)| \underset{(a)}{\leq} \mathbb{E}_{\mathbf{P}^v}|f(G)| \quad = \mathbb{E}_{\mathbf{P}^v}|< f, \phi(G) >_{\mathcal{H}}| \\ \leq \mathbb{E}_{\mathbf{P}^v}\left(\sqrt{k(G,G)}\|f\|\right),$$

where Jensen's inequality is used in (a). Hence by the Riesz representer theorem [45], there exists a $\mu_{\mathbf{P}^v} \in \mathcal{H}$ such that $T_{\mathbf{P}^v} f = < f, \mu_{\mathbf{P}^v} >_{\mathcal{H}}$. □

Given a function $f \in \mathcal{H}$, we can define a linear operation $L_f : \mathcal{H} \mapsto \mathbb{R}$ over the vertex domain,

$$\begin{aligned} L_f\mu_{\mathbf{P}^v} &= < f, \mu_{\mathbf{P}^v} >_{\mathcal{H}}, = < f, \mathbb{E}_{G\sim\mathbf{P}^v}[k(\cdot, G)] >_{\mathcal{H}} \\ &= \mathbb{E}_{G\sim\mathbf{P}^v} < f, k(\cdot, G) >_{\mathcal{H}} \\ &\underset{(b)}{=} \mathbb{E}_{G\sim\mathbf{P}^v} f(G), \end{aligned}$$

where in (b) the reproducing property is leveraged. The features of distribution $\mathbf{P}^v$ can be identified with multiple functions f, which are learnt from the data. Given the set of subgraphs G^v, the linear operation L_f over the vertex v can be estimated empirically by $\hat{L_f}\mu_{\mathbf{P}^v} = \frac{1}{|G^v|}\sum_{G\in G^v} < f, \phi(G) >_{\mathcal{H}}$.

Compared to the linear operation, convolution offers an efficient architecture to extract highly meaningful statistical patterns in large-scale and high-dimensional datasets. Therefore, motivated by the power of kernel methods, convolution is generalized to subgraph data for node representation learning.

Definition 2 (Convolution over vertex domain). *Let $\mathbb{F}$ be a Borel probability measure on $\mathcal{H}$. The convolution operation $L_{\mathbb{F}}$ over the vertex v is defined by a mapping:*

$$\begin{aligned} &L_{\mathbb{F}} : \mathcal{H} \mapsto \mathbb{R}, \\ &\quad \mu_{\mathbf{P}^v} \mapsto \mathbb{E}_{f\sim\mathbb{F}}\left[L_f \mu_{\mathbf{P}^v}\right]. \end{aligned}$$

Formally, the vertex v is filtered by $L_{\mathbb{F}}$:

$$L_{\mathbb{F}}\mu_{\mathbf{P}^v} = \mathbb{E}_{f\sim\mathbb{F}}\left[L_f\mu_{\mathbf{P}^v}\right] = \mathbb{E}_{f\sim\mathbb{F}}[\mathbb{E}_{G\sim\mathbf{P}^v}[f(G)]].$$

Let the graph filter $F = \{f_t \in \mathcal{H}\}$ be a set of instances obtained from the probability distribution $\mathbb{F}$. The convolution $L_{\mathbb{F}}$ over the vertex v can be estimated empirically as:

$$\hat{L_{\mathbb{F}}}\mu_{\mathbf{P}^v} = \frac{1}{|G^v||F|}\sum_{f\in F}\sum_{G\in G^v} < f, \phi(G) >_{\mathcal{H}} . \tag{1}$$

3.2 Paradigm

Based on the definition of convolution $L_{\mathbb{F}}$ over the vertex domain, a paradigm of deep learning of node representation is proposed. The paradigm consists of four parts summarized below.

Subgraph Construction. An efficient way to select neighbors of the vertex v is breath-first sampling (BFS), where the nodes within exactly q hops from v are assembled to construct the neighborhood subgraphs ($q \geq 1$). Thus, the neighborhood subgraphs can be easily generated by choosing multiple q with different values. Another approaches, such as depth-first sampling (DFS) and the combination of BFS and DFS, can also be used to generate the neighbors. On the other hand, the mature community detection algorithms [46], [47], [48] have shown promising results in grouping the nodes with dense connections into a community. Usually, a vertex may belong to multiple communities simultaneously. Based on the existing community detection algorithms, we can construct 0, 1 or several communities subgraphs for each vertex.

Kernel Function for Subgraph. The graph kernels are popular approaches to measure the similarity between two structured graphs. Most popular graph kernels are instances of the family of R-convolution kernels. The convolutional kernels decompose a given graph G into a set of substructures $\Gamma(G)$. The kernel between two graphs G and H is defined as following:

$$K(G, H) := \sum_{g\in\Gamma(G), h\in\Gamma(H)} \phi_k(g)\phi_k(h) = \phi(G)\phi(H),$$

where k is a base kernel to compare the two parts g and h. Notable examples include the random walk kernel [21], the shortest path kernel [49], the graphlet kernel [50] and the Weisfeiler-Lehman kernel [51]. An appropriate graph kernel function can be chosen to project the subgraphs of each node into explicit feature maps.

Feature Map Approximation. The dimension of explicit feature maps can be very large or even infinite. Therefore, the sketch technique is introduced to approximate the feature maps. Then, the convolution $L_{\mathbb{F}}$ can be computed in the low-dimensional vector space. Formally, given a subgraph G in G^v, we approximate the feature maps $\phi(G) \in \mathbb{R}^D$ into a b-dimensional vector $z(\phi(G))$ by Count-Sketch technique such that $b \ll D$. Now, the similarity between two subgraphs G_1 and G_2 can be measured by $k(G_1, G_2) = < \phi(G_1), \phi(G_2) > \approx < z(\phi(G_1)), z(\phi(G_2)) >$. However, the existing graph kernel is a linear kernel which cannot capture the non-linear relation. We extend it with the polynomial kernel. Given specific p, the explicit feature maps of polynomial kernel for subgraph G can be approximated as $z^{(p)}(\phi(G)) \in \mathbb{R}^b$ by Tensor-Sketch. By selecting L base kernels (polynomial kernels with different p), the subgraph G can be represented by L representations: $\{z^{(p)}(\phi(G))\}$. These feature maps with various base kernels can preserve different non-linear structures in multiple RKHSs. Ultimately, a vertex v is encoded in a set of feature maps as follows:

$$\mathcal{R}_v = \{ \mathcal{R}_v^{(p)} \mid p = 1, \ldots, L \},$$
$$\mathcal{R}_v^{(p)} = \{ z^{(p)}(\phi(G)) \mid G \in G^v \}.$$

Deep Representation Learning Model. A network embedding model E should be constructed to collapse all the feature maps in $\mathcal{R}_v$ into a single vector for each vertex v, i.e., the node representation $\mathbf{Emb}(v) = E(\mathcal{R}_v; \mathbf{\Theta}_e)$. The model E is required to define multiple filters F with different base kernels to perform convolution operation over vertex domain. The features identified by graph filters are passed to other network architectures from which the final node representations are generated. An objective function should be designed to force the model E to capture the network structures as well. In particular, we seek to optimize the following objective function, which maximizes the log-probability of observing a network context $\mathcal{Q}_v$ for a node v based on its feature representation, given E:

$$\max_{\mathbf{\Theta}_e} \sum_{v \in V} \log Pr(\mathcal{Q}_v | E(\mathcal{R}_v; \mathbf{\Theta}_e)). \tag{2}$$

The context $\mathcal{Q}_v$ is the key factor to determine what kind of structure the model E should be preserved. It can be be regarded as the supervised information to constrain the similarity of the latent representations of a pair of vertices. For example, to learn the first or higher order proximities (local structure), a popular way to generate $\mathcal{Q}_v$ is performing random walks in the network, such as DeepWalk [9] and node2vec [2]. If two nodes appear in the same walking sequence, they are treated as in the same context. On the other hand, to capture the community property (global structure), the context $\mathcal{Q}_v$ can be generated to include the nodes that share the same communities with v.

The proposed paradigm is highly flexible and can be expended to model the complexity graphs, e.g., directed graphs and attributed networks. These components in the paradigm can be adjusted for different applications manually.

4 Deep Kernel Network Embedding

Building upon the novel paradigm, a deep Kernel Network Embedding (KNE) model is proposed to perform embedding of attributed networks. The construction of node neighborhood and community subgraphs is beyond the scope of this paper. In the realization, we choose BFS to generate different neighborhood subgraphs with variant hop ranges while the communities are divided manually.

4.1 Approximating Feature Maps

The graph kernel KONG proposed in [23] defines a kernel between neighborhood subgraphs. KONG kernels are applied to generate and approximate the explicit feature maps of neighborhood subgraphs for various kernel functions using sketching techniques. Meanwhile, R-convolution kernels are selected to project the community subgraphs into feature maps by decomposing community subgraphs into a set of substructures, i.e., neighborhood subgraphs.

The key steps of generating feature maps are: (a) representing each node's subgraphs in G^v by strings, and (b) effectively computing and approximating k-gram frequency vector for each string s in a way which does not require storing the string s. Based on the string kernel, the feature map $\phi(G)$ of subgraph G in G^v is equal to its k-gram frequency vector denoted as $\boldsymbol{\psi}(SG)$, where SG is the string of subgraph G. To conduct network embedding, KNE feeds all sketches in $\mathcal{R}_v$ into a network model E. In the simplest case, only the last layer sketches in $\mathcal{R}_v$, i.e. $\mathcal{R}_v^{(L)}$, are selected to generate the node representations by concatenating them as follows:

$$\bigcup_{G \in G^v} z^{(L)}(\boldsymbol{\psi}(SG)). \tag{3}$$

We called this method as Linear KNE (LKNE).

Given a Count-Sketch module $z(\cdot)$, let's compute the sketches of attribute, neighborhood and community subgraphs for each vertex v in $\mathcal{R}_v^{(1)}$ as follows. The node attributes can be categorized into two types: a document and a set of discrete attributes (e.g., gender), both of which can be modeled by k-gram ($k \geq 1$ and $k = 1$) frequency vectors to preserve diverse semantics and properties. The attribute sketch $z(\boldsymbol{\psi}(c_v))$ can be easily computed by inputting the frequency vectors into the Count-Sketch module.

To represent a neighborhood subgraph N, KONG kernel is introduced to collect all the node attributes of the vertices by the order function τ_v and concatenate them as the string SN. Note here that it is not required to storing SN. Let pre_j and suf_j denote the $(k-1)$-prefix and $(k-1)$-suffix of c_j for the jth neighbor of vertex v. According to the linear projection of Count-Sketch, sketch of explicit feature maps for string SN is computed as following:

$$z(\boldsymbol{\psi}(SN)) = \sum_{u \in V_N} z(\boldsymbol{\psi}(c_u)) + \sum_{j=0}^{|V_N|-1} z(\boldsymbol{\psi}(suf_j \circ pre_{j+1})). \tag{4}$$

Each community subgraph can be decomposed into multiple substructures, i.e., neighborhood subgraphs, and each substructure is represented as a vector of k-gram frequencies. Therefore, according to R-convolution kernel methods, the feature maps of community subgraph M can be approximated as follows:

$$z(\boldsymbol{\psi}(SM)) = z\left(\sum_{N \in \mathcal{M}} \boldsymbol{\psi}(SN)) = \sum_{N \in \mathcal{M}} z(\boldsymbol{\psi}(SN)\right), \tag{5}$$

where $\mathcal{M}$ are the neighborhood subgraphs of the vertices within the community. Each node may have multiple different neighborhood subgraphs. In our experiments, the neighborhood subgraph with the maximum hop of BFS is used to compute the sketch of community subgraphs.

The Tensor-Sketch algorithm keeps p Count-Sketch modules with different hash functions to compute $z^{(p)}(\mathbf{x})$. The vector $\mathbf{x}$ is fed into each Count-Sketch. By constructing L Count-Sketch modules, we obtain L attribute, neighborhood and community sketches for each v. Then, p count sketches of v are collected to generate $\mathcal{R}_v^{(p)}$ using the Fast Fourier transform. Note here, for all the vertices, each layer of the sketches in $\mathcal{R}_v$ are computed with the same hash functions.

4.2 Learning Node Representations

Linear KNE has some drawbacks. The combination of all the sketches is linear. Meanwhile, the method cannot leverage all the layers in R_v simultaneously to capture different features under different RKHSs. We propose a model E to incorporate R_v into the representation of each v as follows.

4.2.1 *Convolutional Neural Networks*

For each layer $\mathcal{R}_v^{(p)}$ of vertex v, a convolutional layer is applied to learn the features. In particular, let $\mathbf{x}_k \in \mathbb{R}^b$ represents the kth sketch in $\mathcal{R}_v^{(p)}$ and the length of $\mathcal{R}_v^{(p)}$ is denoted as n. In general, let $\mathbf{x}_{k:(k+j)}$ refer to the concatenation of $j+1$ sketches from $\mathbf{x}_k$ to $\mathbf{x}_{(k+j)}$. A convolution operation with a filter $\mathbf{w}$ is applied to a window of $h \leq n$ sketches $\mathbf{x}_{k:(k+h-1)}$ to produce a new feature t_k as following:

$$t_k = f(\mathbf{w} \cdot x_{k:(k+h-1)} + \mathbf{b})$$

where $\mathbf{b} \in \mathbb{R}$ is a bias term and f is a non-linear function such as the rectified linear unit (ReLU). As the convolution $\hat{L_{\mathbb{F}}}$ evaluated in Eq. (1), a vertex v is filtered by a graph filter $\mathbf{w}$ in a specific RKHS and a feature t_k is identified with the filter which are learned from the feature maps. This filter is applied to each possible window of sketches to produce a feature vector $\mathbf{t} = [t_0, t_1, \ldots, t_{n-1}] \in \mathbb{R}^n$. To capture the most important features, a max-over-time pooling operation over the feature vector is conducted by taking the maximum value $\tilde{t} = \max\{\mathbf{t}\}$ as the feature.

We use F filters with H different window sizes to obtain multiple features for $\mathcal{R}_v^{(p)}$. Similarly, L convolution layers are applied for L layers of sketches in $\mathcal{R}_v$ to obtain FHL features. These features form the penultimate layer and are passed to the fully connected linear layers. Finally, the output of the last hidden layer is the learned representation of each node.

4.2.2 *Loss Functions*

As discussed above, the key problem of objective function is to design an appropriate context $\mathcal{Q}_v$ for different tasks. Let $\mathbf{y}_v$ be the node representation of the vertex v. If the vertex v_j is a member of $\mathcal{Q}_v$, the similarity between the representations of v_j and $\mathbf{y}_v$ should be reinforced. Instead, if the vertex v_j is not in the context of vertex v, the representations should be more different. For simplification, we assume that the likelihood of observing a context node is independent of observing any other context node given the source representation $\mathbf{y}_v$, i.e., $Pr(\mathcal{Q}_v|\mathbf{y}_v) = \prod_{u\in\mathcal{Q}_v} Pr(u|\mathbf{y}_v)$.

To learn the first or higher order proximities (local structure), we adopt a biased random walk procedure proposed by node2vec to generate $\mathcal{Q}_v$, which controls the random walk by balancing BFS and DFS. Inspired by the idea of negative sampling, the loss function for this goal is formulated as follows:

$$\begin{aligned}\mathbb{L}(v) &= \mathbb{E}_{v_j\sim\mathcal{Q}_v}\left[\log P(v_j|\mathbf{y}_v) + \lambda \mathbb{E}_{v_k\sim P_S}[\log(1-P(v_k|\mathbf{y}_v))]\right] \\ &\approx \frac{1}{|\mathcal{Q}_v|}\sum_{v_j\in\mathcal{Q}_v}\left(\log\sigma(\mathbf{y}_v\mathbf{u}_j) + \lambda\sum_{v_k\in\mathcal{S}_i}\log\sigma(-\mathbf{y}_v\mathbf{u}_k)\right) \qquad (6)\end{aligned}$$

where $\sigma(x) = 1/(1+e^{-x})$ and $U = \{\mathbf{u}_i\} \in \mathbb{R}^{|V|\times d}$ is a weight matrix between the last hidden layer and the output layer. $\mathcal{S}_i$ represents the negative samples which is the subset of $\{v_k|(v_i, v_k) \notin E\}$ and λ is the length of $\mathcal{S}_i$. $\mathbb{E}_{v_i\sim P_S}[\cdot]$ is the expectation when the instance obtained from negative samples conforms to the distribution $P_S = \frac{\#(c)}{|S|}$ (the distribution of the unigram c in S).

Some tasks require that KNE preserves the community structure in the node representations. It is easy to generate $\mathcal{Q}_v$ when all the nodes that share the same community with v are included. Usually, the length of $\mathcal{Q}_v$ is large and thus directly optimizing the loss function Eq. (2) is very time-consuming and space-consuming. Therefore, we introduce Triplet Network [27] to simplify the objective. Specifically, for each vertex v, a set $\mathcal{O}(v) = \{(v_i^+, v_i^-)|\ell(v_i^+) \wedge \ell(v) \neq \mathbf{0}, (v_i^-) \wedge \ell(v) = \mathbf{0}\}$ is defined to consist of π pairs of nodes. $P(v_i^+)$ ($P(v_i^-)$) is used to measure the similarity between y_v and the representation of v_i^+ (v_i^-). $P(v_i^*)$ ($* \in \{+,-\}$) can be evaluated by a softmax layer as following:

$$P(v_i^*) \approx d_i^* = \frac{\exp(\|\mathbf{y}_v - \mathbf{u}_i^*\|_2)}{\sum_{c\in\{+,-\}}\exp(\|\mathbf{y}_v - \mathbf{u}_i^c\|_2)},$$

where $\mathbf{u}_i^+, \mathbf{u}_i^- \in U$ represent the embedding of node v_i^+ and v_i^-. Then, Triplet Network [27] preserves the community information by maximizing the loss function as following:

$$\begin{aligned}\mathbb{L}_c(v) &= \mathbb{E}_{(v_i^+,v_i^-)\in\mathcal{O}(v)}\log\left(P(v_i^+)(1-P(v_i^-))\right) \\ &\approx \frac{2}{\pi}\frac{1}{|\mathcal{O}(v)|}\sum_{v_i^+\in\mathcal{O}(v)}\log d_i^+. \qquad (7)\end{aligned}$$

Overall, the deep model combines Eqs. (6) and (7) to jointly minimize the following objective function over all the nodes to learn the embedding as following:

$$\mathbb{L}_{mix} = -\frac{1}{|V|}\sum_{v\in V}(\mathbb{L}(v) + \alpha\mathbb{L}_c(v)). \qquad (8)$$

4.3 Analysis

4.3.1 *Sketching Performance*

The following theorem gives the additive error bound and time-space performance of our sketching algorithms.

Theorem 1. *Let $\mathcal{K}$ be a convolutional kernel whose base kernel is the polynomial kernel with parameter p, and let $\hat{\mathcal{K}}$ be $\mathcal{K}$'s approximation obtained by applying size-b sketches for explicit feature maps. R denotes an upper bound on the norm of the k-gram distribution vector* $\mathbf{x}$, *i.e.,* $\|\mathbf{x}\| \leq R$. *We can obtain: (i) the additive error bound, and (ii) time-space bounds, as follows.*

(i) A sketch with size $b = (\delta)^{-1-2n}$ *can be chosen such that* $\hat{\mathcal{K}}(v_i, v_j)$ *has an additive error of at most* $\mathcal{K}(v_i, v_j) + \sqrt{2}\delta^n R^{2p}$ *with a probability at least* $1-\delta$, *for* $\delta \in (0,1)$ *and* $n \geq 0$.

(ii) The sketches of all nodes can be computed in time $O(pv(s+nhk+m) + pv(m+n)b\log b)$ *and space* $O(pv(m+n)b)$, *where s is the maximum length of attribute data, h is the size of neighborhood and v is the number of vertices, n and m are the number of neighborhood and community subgraphs of each vertex respectively.*

Proof. (i) Let $C\mathbf{x}$ denote the Tensor-Sketch of vector $\mathbf{x}$. According to the main result from [26, Lemma 7], a standard application of Chebyshev's inequality yields an $(1\pm\varepsilon)$-multiplicative approximation of $(\mathbf{x}^\top\mathbf{y})^p$ with a probability larger than $\frac{2R^{4p}}{b\varepsilon^2}$,

$$P(|(C\mathbf{x})^\top C\mathbf{y} - (\mathbf{x}^\top\mathbf{y})^p| \geq \varepsilon) \leq \frac{(\mathbf{x}^\top\mathbf{y})^{2p} + (\|\mathbf{x}\|\|\mathbf{y}\|)^{2p}}{b\varepsilon^2} \leq \frac{2(\|\mathbf{x}\|\|\mathbf{y}\|)^{2p}}{b\varepsilon^2} \leq \frac{2R^{4p}}{b\varepsilon^2}.$$

According to Cauchy-Schwarz inequality, it holds that $(\mathbf{x}^\top\mathbf{y})^2 \leq (\|\mathbf{x}\|\|\mathbf{y}\|)^2$. Let δ denotes $\frac{2R^{4p}}{b\varepsilon^2}$. We choose $b = (\delta)^{-1-2n}, n \geq 0$ and the claimed additive error bound is given as follows:

$$P(|(C\mathbf{x})^\top C\mathbf{y} - (\mathbf{x}^\top\mathbf{y})^p| \leq \sqrt{2}\delta^n R^{2p}) \geq 1 - \delta.$$

(ii) A single Count-Sketch that summarizes the k-gram frequency vector of the string with length s can be computed in time $s - k + 1 = O(s - k)$ and space $O(b)$ for sketch size b. The computation for all the attributes in the vertices is in time $O(v(s - k))$ and space $O(vb)$. To obtain a neighborhood sketch of a vertex, we first sum up all the attributes sketches within its neighborhood in time $O(h)$. Then, the $k - 1$-prefix and $k - 1$-suffix of each attribute in the neighborhood are kept and concatenated to generate k-grams. This concatenation operation is computed in time $O(hk)$. Thus, n neighborhood sketches for v vertices are computed in $O(vnhk)$. Similarly, we sum up all the neighborhood sketches for m community subgraphs in time $O(vm)$. Therefore, for a single Count-Sketch, all the sketches of the graph can be computed in time $O(v(s + nhk + m))$ and space $O(v(m + n)b)$.

The Tensor-Sketch algorithm keeps p Count-Sketches per string and the k-gram distribution vector should be fed into each sketch. After generating a string, the p sketches of the string are converted to a single sketch using Fast Fourier transform in time $O(pb\log b)$. Meanwhile, p sketches of a string are stored in space $O(pb)$. □

4.3.2 *Embedding New Nodes*

Many real-world networks are dynamic and it is practically important to quickly obtain reasonable representations for the new nodes and adjust the embeddings for the original nodes with new links. The existing models, e.g., node2vec [2], SNDE [52] and ASNE [19], consider that the new node independently and is also time consuming. Our model utilizes the subgraphs as the inputs and exploits the correlations among node attributes, neighborhood and communities to learn the representations for new nodes. A significant advantage of our models is that we introduce kernel methods to project the structure information, i.e., the subgraphs of vertices, into scalable feature maps. These feature maps preserve plenty of information about the attributes, neighborhood and community features of vertices. However, the inputs of ASNE model are one-hot vectors that don't contain meaningful structure information and thus the embedding for the new node cannot effectively preserve the network structure without re-training. Specifically, according to Theorem 1, generating sketches for a new node is in time $O(s + nkh + m)$ and space $O((m + n)pb)$ which is unrelated with the total number of vertices in the network v but the global (community) information is still preserved since the Count-Sketch is a linear transformation, i.e., it holds $z(x + y) = z(x) + z(y)$. The community sketches are updated by adding the new neighborhood sketches in time $O(m)$. Given the sketches of a new node, the representation can be directly generated in time $O(1)$.

5 Experiments

In this section, we evaluate KNE on several real-world datasets by comparing it with the baseline models.

5.1 Datasets

Four network datasets, including two citation networks and two call graph networks, are used to evaluate the performance of KNE in two real-world applications, i.e., multiclass classification and link inference.

-*DBLP* and *Cora*: DBLP[1] and Cora[2] datasets are used to construct the citation networks. Each node and edge denote a paper and the reference between papers respectively. For DBLP dataset, the attributes are the concatenation of title and abstract after removing all the stop words. To represent the semantic information effectively, the top 5 k-gram terms with maximum TF-IDF values are extracted for each attribute. For Cora dataset, the node attributes are described by a 0/1-valued word vector indicating the absence/presence of the corresponding word in the dictionary. The Cora is the sparse network and thus there exist multiple isolated subgraphs which are considered as communities while we use publication venue as the communities for DBLP in experiments.

-*ARM* and *Intel*: To further evaluate the performance of the learned embedding in other domains such as software analysis, we collect PE (portable executable) in two systems with different instruction set architectures, i.e., ARM and Intel, and extract the call information from these binaries to construct call graph networks. The call graph is produced by program analysis tools to represent the relationships between subroutines in a computer program. Each node represents a subroutine and each edge represents a call from a subroutine to another. The node attributes are the sequence of instructions which are executed by the subroutine sequentially. ARM and Intel datasets respectively contain 306 and 75 call graphs classified into one of four and three classes. Each call graph in the network is viewed as a community.

To facilitate random walks on the edges, all of the four networks are treated as undirected graphs. Table 1 summarizes the statistics of the datasets.

5.2 Experimental Setup

Several state-of-the-art baseline methods are used for comparison.

-*node2vec* (N2V) [2]: It combines random walks and skip-gram to learn the representations based on the structure information only. node2vec designs a biased random walk procedure to explore diverse neighborhoods. Two hyperparameters p and q are introduced to control the random walk, which are set in the same way as the original paper. Note here that when p and q are set to be 1, *node2vec* degrades to *DeepWalk* [9].

1. http://arnetminer.org/citation (V4 version is used)
2. https://linqs.soe.ucsc.edu/data

TABLE 1
Statistics of Datasets

Datasets	ARM	Intel	Cora	DBLP
# Nodes	27,385	23,566	2,356	1,632,441
# Edges	53,962	50,970	9,486	4,569,390
# Classes	4	3	7	-
# Communities	306	75	103	7,708
# Max Community Nodes[1]	504	3112	931	23,730
# Min Community Nodes[1]	40	43	3	1
# Class Edges[2]	53,962	50,970	6,228	-
Class Edges ratio[3]	1.0	1.0	0.657	-
# k-Gram Terms[4]	1,375	1,437	1,433	5,853,106
k value	3	3	1	3

[1] *refers to the maximum/minimum number of nodes in communities.*
[2] *refers to the number of edges that both endpoints share the same classes.*
[3] *refers to the ratio of class edges to edges.*
[4] *refers to the number of all k-gram terms in node attributes.*

-*HOPE* [33]: It leverages a revised Singular Value Decomposition (SVD) to handle sparse similarity matrices. We report the results obtained running HOPE with the default parameters, i.e, Katz similarity as the similarity measure.

-*VERSE* [31]: It is an instance of random walk method that use Personalized PageRank (PPR) and SimRank as the proximities. *VERSE* learns the vertex embeddings by training a single-layer neural network. We use the original version of *VERSE* which simulates α-discounted random walk to train the skip-gram model. The decay factor α is set to the default value 0.85.

-*ProNE* [35]: It is a scalable and fast method that first initializes NE efficiently based on matrix factorization and then enhances the quality of embeddings by propagating them in the spectrally modulated space.

-*STRAP* [36]: It proposes the concept of transpose proximity to preserve the out-degree distributions and provide an consistent optimization strategy. *STRAP* computes the sparse PPR as its transpose proximity.

-*MVC-DB* [18]: It learns two embedding vectors for each node by preserving the network structure and the node properties respectively. Then, the embedding vectors are integrated by deep autoencoders.

-*ASNE* [19]: It adopts a deep neural network architecture to model the interrelations between structural information and attributes. The deep Multi-Layer Perception (MLP) is used to fuse the content and structure information from the raw input.

-*DANE* [41] It employs two auto-encoders for network structure and node attributes, and then encode the consistent and complementary information between them to capture the high non-linearity and preserve various proximities.

For all the baselines, the implementation released in the original works is used. Note here that *node2vec*, *DeepWalk*, *HOPE*, *VERSE*, *ProNE* and *STRAP* are designed to use only the structure information. For a fair comparison with our model, we further extend them to include attributes by concatenating the learned node representations with the attribute sketches. We denote the variants *DeepWalk+A*, *node2vec+A*, *HOPE+A*, *VERSE+A*, *ProNE+A* and *STRAP+A*.

Moreover, two variants of the proposed models are also implemented to verify the effectiveness of community information. In details, *KNE-C* and *LKNE-C* are the variations of *KNE* and the linear model *LKNE*, where the community sketch is removed from the input. For *KNE-C*, the loss function related to the communities is removed as well, i.e., the parameter α in Eq. (8) is set to be 0. The implementation of *KNE* is based on PyTorch[3]. As shown in Table 2, *KNE* is given different configurations for the datasets with different scale of networks. The output dimension is set to be 300. The *KNE* model is trained end-to-end using Adam [53] optimizer with batch size of 6,000 and learning rate of 0.001. The parameter α is tuned from 0.0 to 1.0, and we find 0.1 is the reasonable setting after validation. The nodes within exactly 1 and 2 hops are assembled to construct two neighborhood subgraphs for each node. The length of context $\mathcal{Q}_v$ is set to be 10. To achieve model stability, particularly when the network is dense, 10 context sets are used for each node to optimize the objective Eq. (6). The hyper-parameters λ and π in Eqs. (6) and (7) are empirically set to be 5 and 5 empirically.

TABLE 2
Configurations of KNE for Four Datasets

Datasets	b	F	H	L	# nodes in linear layer
ARM	300	150	(1,2,3)	3	1,350- 500 - 300
Intel	300	150	(1,2,3)	3	1,350- 500 - 300
DBLP	800	200	(1,2,3)	3	1800- 900 - 300
Cora	300	100	(1,2,3)	3	900- 450 - 300

5.3 Multi-Class Classification

In this experiment, we evaluate the effectiveness of different network representations through a multi-class classification task. The vertex representations generated from the embedding methods are used as the features to classify the vertices. Specifically, the LinearSVM package implemented by scikit-learn[4] is adopted to train the classifier. A portion of communities are randomly sampled as the training data. Then, the nodes in the remainder communities are used as the testing data. The proportion of training samples varies from 10% to 50% for the datasets ARM and Cora. Note here that the representation data is divided and organized as a series of communities in order to ensure that the community sketches in the training set will not appear in testing set. We run the experiment 5 times and report the average Micro-F1 and Macro-F1 scores.

The experiment results are shown in Table 3, in which *KNE* consistently outperforms the state-of-the-art baselines overall. It demonstrates that the learnt network representations of *KNE* can better generalize to the classification task. For ARM dataset, *KNE* and its variants can achieve more significantly improvement than baselines when the labelled data is limited. One also can see that the performance of *LKNE* and *KNE* is better than the variants (*LKNE-C* and *KNE-C*) without community sketches significantly. This is because that the nodes in the same communities share the same labels (refer to "# class edges" in Table 1) in ARM, demonstrating that the community sketches can effectively encode the class information in communities. In addition, our models have superior performance over the direct concatenation models (e.g., *node2vec+A*, *DeepWalk+A*, *HOPE+A*,

3. https://pytorch.org/
4. http://scikit-learn.org/

TABLE 3
Multi-Class Classification Results (F1-Score) on ARM and Cora Datasets

Datasets	ARM						Cora					
Methods	10%		30%		50%		10%		30%		50%	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
N2V [2]	0.246	0.267	0.268	0.276	0.263	0.270	0.494	0.498	0.582	0.593	0.721	0.743
DeepWalk+A [9]	0.290	0.313	0.449	0.467	0.549	0.565	0.454	0.489	0.543	0.561	0.718	0.719
N2V+A [2]	0.374	0.446	0.588	0.609	0.655	0.667	0.415	0.455	0.551	0.576	0.709	0.729
HOPE [33]	0.256	0.242	0.341	0.306	0.359	0.320	0.303	0.215	0.568	0.542	0.661	0.620
HOPE+A [33]	0.453	0.397	0.602	0.593	0.698	0.655	0.273	0.161	0.385	0.326	0.575	0.531
VERSE [31]	0.236	0.214	0.264	0.244	0.290	0.283	0.420	0.303	0.592	0.528	**0.740**	0.725
VERSE+A [31]	0.354	0.327	0.622	0.587	0.709	0.648	0.400	0.307	0.559	0.538	0.710	0.660
ProNE [35]	0.368	0.337	0.389	0.366	0.401	0.389	0.385	0.304	0.568	0.516	0.724	0.683
ProNE+A [35]	0.489	0.456	0.672	0.642	0.710	0.692	0.367	0.298	0.542	0.523	0.706	0.672
STRAP+A [36]	0.471	0.433	0.680	0.632	0.690	0.674	0.410	0.298	0.550	0.541	0.730	0.701
DANE [41]	0.732	0.756	0.766	0.789	0.803	0.810	0.512	0.557	**0.601**	**0.629**	0.701	0.724
MVC-DB [18]	0.711	0.738	0.738	0.760	0.800	0.806	0.346	0.365	0.428	0.441	0.578	0.618
ASNE [19]	0.237	0.279	0.644	0.658	0.686	0.691	0.390	0.427	0.387	0.447	0.638	0.659
LKNE	**0.986**	**0.987**	**0.992**	**0.992**	**0.979**	**0.979**	0.307	0.346	0.307	0.377	0.453	0.487
KNE	0.698	0.733	0.828	0.849	0.850	0.869	**0.534**	**0.561**	0.593	0.624	0.707	0.739
LKNE-C	0.872	0.887	0.903	0.903	0.872	0.876	0.270	0.338	0.293	0.360	0.454	0.490
KNE-C	0.695	0.708	0.736	0.736	0.714	0.726	0.514	0.553	0.588	0.615	0.737	**0.752**

VERSE+A, *ProNE+A* and *STRAP+A*) which verifies the positive effect of incorporating attributes and structure of networks into the embedding process rather than simple concatenation.

For Cora dataset, *KNE* and *KNE-C* only have slight improvement of the performance compared to *node2vec* in most tasks. The main reason is that compared to ARM dataset, the node attributes of Cora are sparse such that the feature maps cannot capture sufficient features to separate representations. Meanwhile, it is worth noting that *KNE* and *LKNE* have no advantages even disadvantages compared to *KNE-C* and *LKNE-C*. It mainly causes from the lower class edges ratio, i.e., the nodes in neighborhood and community subgraphs share different classes such that the aggregation of all nodes in subgraphs may disturb the node representations to preserve the true label information.

5.4 Link Inference

The goal of link inference is to recover missing links that are likely to exist between nodes but have not been recorded because of incomplete data. In this section, experiments are conducted to evaluate the performance of link inference, including the overall performance and the effectiveness of community information for link inference.

To perform link inference, we randomly hide a portion of the existing edges and use the rest of the network to learn the vertex representations. An edge $e = (v_s, v_e)$ is defined as an *intra-community edge* if and only if $\ell(v_s) \wedge \ell(v_e) \neq \mathbf{0}$. The hidden edges are denoted as positive samples. Meanwhile, the same number of unconnected node pairs is randomly selected as the negative links. Note here that there are two types of negative edges: inter-community edges and intra-community edges. Let T_c denote the ratio of negative intra-community edges to negative edges. After training, the representation for each vertex is learned to predict the unobserved links, i.e., the representation vectors are used to calculate the cosine similarity between two nodes. Area Under Curve (AUC) is used as the evaluation metric, which can find the optimal threshold to predict the positive and negative links automatically.

In the *first experiment*, we randomly hide 40 percentage of the existing links. T_c is set to 1, i.e., all the negative edges are generated between the nodes with the same community. Some of the observations and analysis on Table 4 are listed as follows:

- *KNE* and *KNE-C* are better than the baseline methods in most of the datasets. *KNE* significantly outperforms *DeepWalk* and *node2vec*, which reflects the contribution of attributes. However, compared to the original versions, the poor scores of *node2vec+A*, *VERSE* and *ProNE* indicate that simply concatenating attributes with the embedding vector is insufficient to leverage the rich content in attributes. It reveals the necessity of designing a more principled approach to incorporate multi-view data into the network embedding process. Although integrating the attribute into representations, *DANE*, *ANSE* and *MVC-DB* still obtain a worse performance

TABLE 4
Link Inference Results (AUC) on the Four Datasets

Methods	ARM	Intel	DBLP	Cora
DeepWalk [9]	0.652	0.685	0.876	0.758
node2vec [2]	0.662	0.692	0.892	0.759
node2vec+A [2]	0.650	0.685	0.888	0.759
HOPE [33]	0.647	0.705	0.896	0.770
VERSE [31]	0.776	0.755	0.913	**0.801**
VERSE+A [31]	0.774	0.750	0.914	0.779
ProNE [35]	0.754	0.741	0.901	0.766
ProNE+A [35]	0.732	0.740	0.892	0.759
STRAP [36]	0.783	0.754	0.911	0.780
DANE [41]	0.790	0.756	-	0.754
ANSE [19]	0.769	0.716	-	0.753
MVC-DB [18]	0.702	0.700	-	0.741
LKNE	0.647	0.660	0.768	0.553
KNE	0.810	0.764	**0.928**	0.795
LKNE-C	0.644	0.654	0.675	0.720
KNE-C	**0.814**	**0.774**	0.902	0.775

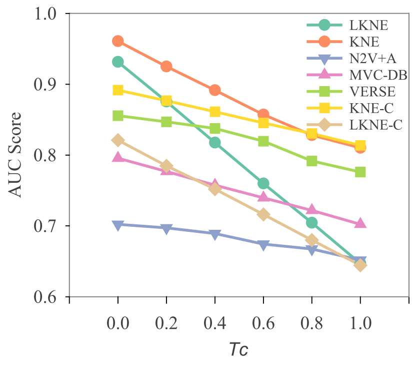

Fig. 2. Link inference for varying T_c under dataset ARM.

since the neighborhood and community structure information is not well exploited. Note here, for DBLP dataset, all the attributed NE models (i.e., *ASNE*, *DANE* and *MVCDB*) cannot handle the large scale networks because the width of their models is proportional to the number of vertices.

- *VERSE* shares the similar performance with *KNE* in Cora dataset but there are large margins in ARM and DBLP datasets. ARM and DBLP networks are sparse extremely after removing 40% percentage edges and therefore plenty of nodes will be orphan. *VERSE*, *ProNE*, *STRAP* and other structure-based methods suffer from the sparsity problem because they fail to handle these orphan nodes without any structure information. *ProNE* views NE as the sparse matrix factorization (MF) problem and introduces the rapid algorithm of MF to improve the running time by a large margin (As shown in Section 5.7). But, the attributes and a good fusing principle are also necessary for high-quality embeddings. *KNE* exploits the advantages of deep learning model such that the context semantics and latent patterns of nodes with dense links can be transferred to the orphan nodes efficiently via sharing model parameters.

- In most cases, the link inference performance of *LKNE* is the worst among the network embedding methods. The reasons are twofold. First, *LKNE* cannot capture the network structure when the attributes of network are sparse or missing. Second, *LKNE* purely relies on the topology of networks and thus is vulnerable to the incomplete networks when some links are missing. For ARM and Intel datasets, *KNE-C* achieves higher scores than *KNE* because the representations of two disconnected nodes within the same community can have high similarity if the community sketches are incorporated into the embeddings. Instead, for citation networks, *KNE* achieves the best performance as the average size of community in DBLP and Cora is rather small.

In the *second experiment*, we change T_c from 0.0 to 1.0 by randomly generating the negative samples as shown in Fig. 2. $T_c = 0.0$ is a common setting in most of researches where the negative links are constructed between any two nodes. When T_c is set to 0.0, one can see that *LKNE* and *KNE* outperform other methods at large margin since the node embeddings are made more discriminative by fusing the community information such that the negative links across communities can be easily detected. Therefore, we select $T_c = 1.0$ to test the link inference task instead of $T_c = 0$ for the fairness of comparison in the above *first experiment*. When the negative inter-community edges increase (T_c decreases), AUC scores of *KNE* and *LKNE* grow faster than other methods. It demonstrates that the representations learned by our methods can preserve diverse community information. However, when T_c is large, AUC score of *KNE* is slightly lower than that of *KNE-C*. Overall, *LKNE* and *KNE* can learn better representations by effectively utilizing the community structure as the complementary information.

5.5 Sparsity Sensitivity

In this subsection, the performance is evaluated for sparsity sensitivity analysis. We will fist describe how to build incomplete graphs based on the network links and node attributes, respectively. Then, the experiment results on the dataset ARM are analyzed under the setting $T_c = 1.0$.

Experiment on Incomplete Networks with Missing Links. A proportion of links are removed from the original networks to construct incomplete networks, where the links between nodes are more sparse. Specifically, the proportion of removed links increases from 0% to 100% to verify whether the models are vulnerable to sparsity. All the models are required to learn the corresponding embeddings on such incomplete networks. The performance of various methods is evaluated by conducting link inference and multi-class classification on the incomplete networks. The details of experiment results are shown in Fig. 3a. One can see that with the increase of removed links, the AUC curves of link inference and F1-score of classification decrease for most of the models. In link inference task, *VERSE* and *node2vec* with steep descent curves are very sensitive to the sparsity where they cannot utilize the attributes to complement the lack of structure information. The margin between *KNE* and *ANSE* becomes larger when the network is sparser. This is because

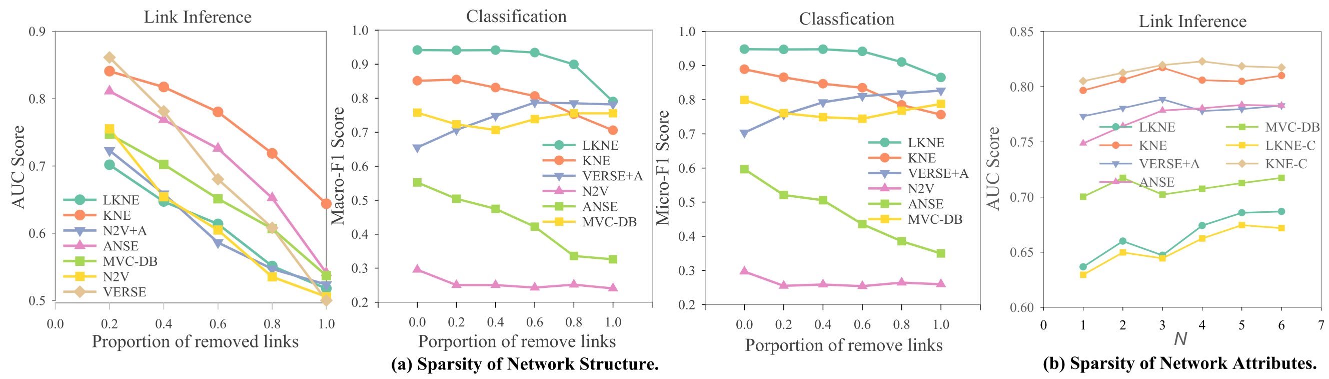

Fig. 3. Sparsity sensitivity of embedding w.r.t the proportion of removed links and node attributes.

TABLE 5
Multi-Class Classification Results (F1-Score) on ARM and Cora for New Nodes

Datasets	ARM						Cora					
Methods	10%		30%		50%		10%		30%		50%	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
MVC-DB [18]	0.676	0.704	0.751	0.789	0.766	0.802	0.398	0.473	0.356	0.483	0.517	0.550
VERSE [31]	0.234	0.212	0.258	0.239	0.284	0.264	0.430	0.343	0.586	0.530	**0.736**	**0.724**
VERSE+A [31]	0.432	0.402	0.621	0.586	0.709	0.653	0.402	0.338	0.548	0.527	0.706	0.673
LKNE	**0.911**	**0.919**	**0.973**	**0.976**	**0.985**	**0.986**	0.542	0.566	0.526	0.565	0.574	0.610
KNE	0.865	0.872	0.949	0.953	0.952	0.957	0.603	**0.633**	**0.610**	**0.648**	0.623	0.643
LKNE-C	0.699	0.729	0.799	0.817	0.829	0.844	0.341	0.427	0.341	0.448	0.412	0.476
KNE-C	0.607	0.629	0.770	0.787	0.799	0.814	0.606	0.630	0.590	0.612	0.596	0.646

KNE learns the multi-view correlation and exploits other related nodes' links to alleviate the link sparsity issue. When all the links are removed, *KNE* captures node similarity from attributes and community subgraphs such that the AUC score is higher 10% than other models. In classification task, the classification performance is affected by the attributes and community subgraphs for ARM dataset. By concatenating the attribute sketches generated by string kernels, *VERSE* has the ability to resist the sparsity of links (refers to *VERSE +A*). Note here that contrary to other models, the accuracy of *VERSE+A* increases with removing the links, because the embeddings from *VERSE* are not consistent with the attribute sketches. This demonstrates the necessity of designing a unified model to joint consider both structure and attribute features.

Experiment on Incomplete Networks with Varying Node Features. The features of node attributes can also be sparse, which are represented by k-gram frequency vectors. We generate 1-gram, 2-gram, ..., N-gram frequency vectors for each vertex and concatenate them as the node attributes. Then, the incomplete networks are constructed with varied settings of N-gram. By removing 40% edges from the network, the representation is learned for each node and link inference is conducted to evaluate the performance. The experiment result is reported in Fig. 3b. One can see that the scores of all methods increase slowly with the increase of N. *KNE* and *KNE-C* consistently outperform the baseline methods and *LKNE*. The result further certifies that *KNE* is more effective to alleviate the effect of sparsity problem.

5.6 New Node Embedding

Some of the previous models such as *ANSE* and *node2vec* cannot efficiently deal with the new nodes except for retraining the whole model, which is very time-consuming. We compare with the method *MVC-DB* which also don't require retraining the model for embedding new node. Meanwhile, to testify the effectiveness of *KNE*, we train *VERSE* with respect to the new nodes, and compare their embeddings with the immediate embeddings obtained from *KNE*. For *VERSE*, we sample multiple positive samples and negative samples from Personalized PageRank vectors of new nodes to obtain their embeddings by training the Noise Contrastive Estimation (NCE). To obtain new nodes for a network, we split the dataset into two parts with ratio of 8:2. That is, 80% of the nodes are used as the training nodes to train the network embedding models and 20% of the remaining nodes are considered as the new nodes. Meanwhile, the edges linking the new nodes with the training nodes are also removed. The baseline methods for new node embedding are *MVC-DB* (without retraining) and *VERSE* (only training over the new nodes). The *KNE* model is first optimized on the training nodes. For each new node, its attribute, neighborhood and community sketches can be computed in a short time and then these sketches are input into the trained *KNE* model to generate the node representation immediately. The experiment results of classification and link prognostication are analyzed as follows.

Performance Evaluation on Multi-Class Classification. Similar to the previous experiment, a proportion of the labeled training nodes are used to train LinearSVM. The new nodes are considered as the test data, and the classification results of new nodes are shown in Table 5. Compared with *KNE* and *KNE-C*, there is a phenomenal growth of F1-Scores when the community sketches are leveraged to generate the new node representations. It indicates that the community information plays a important role in classifying the new nodes. It worth noting that for ARM dataset, the classification performance of *KNE* for new nodes without training is even better than the nodes that are trained directly (shown in Table 3). This is caused by different test strategies: the neighborhood subgraphs of new nodes are gathered to update the community sketches which is critical for identifying the classes of new nodes while the test nodes in Section 5.3 don't share the same communities with training data. MVC-DB mainly utilizes the links and attribute information of the new node only, which is not effective to alleviate the sparsity problem of the new node. *KNE* and *LKNE* also have significant improvement over *VERSE* and *VERSE+A* although without retraining for new nodes that demonstrates the effectiveness of our models in incorporating information from the training nodes' attribute,

TABLE 6
Link Prognostication Results (AUC) for New Nodes

Methods	Intel		Cora		DBLP	
T_c	0.0	1.0	0.0	1.0	0.0	1.0
MVCDB [18]	0.669	0.619	0.758	0.696	-	-
VERSE [31]	0.807	0.761	**0.901**	**0.825**	0.902	**0.819**
VERSE+A [31]	0.813	0.768	0.882	0.813	0.895	0.812
LKNE	0.735	0.640	0.808	0.679	0.791	0.672
KNE	**0.838**	0.731	0.890	0.752	**0.907**	0.803
LKNE-C	0.701	0.664	0.765	0.717	0.707	0.605
KNE-C	0.812	**0.774**	0.868	0.804	0.868	0.810

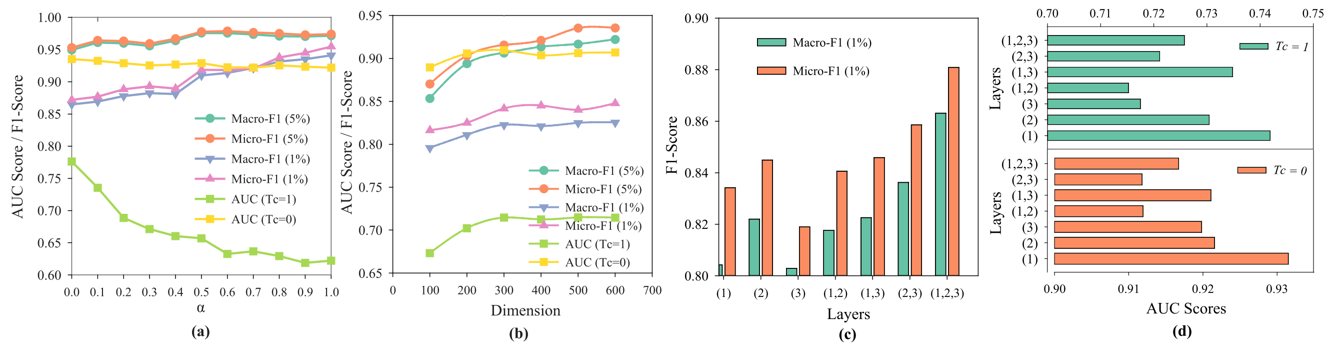

Fig. 4. Embedding Performance for new nodes w.r.t α, embedding dimension, layers and sketch size.

neighborhood and community information for new node embedding learning.

Performance Evaluation on Link Prognostication. To test the quality of generated embeddings, we develop a novel task called *link prognostication* that refers to infer the links between the existing nodes in the network and the new arrival nodes. It differs from link inference task (used in Section 5.4) in that link prognostication predicts the links that never exists instead of links that have existed in the network but been missing. For this purpose, the removed edges which link the new nodes with the training nodes in the original network are considered as the positive samples. We then randomly select the same number of unconnected node pairs as the negative samples of link. The ratio T_c of negative intra-community edges to negative links is set to 0.0 and 1.0 respectively. AUC is adopted to evaluate the performance, and Table 6 gives the experiment result. Specially, *KNE* outperforms *LKNE* and *MVC-DB* by around 5% on average under the settings of no retraining. Deep model adopted by *KNE* has a significant impact on preserving network topology structure compared to *LKNE*. Though *MVC-DB* uses deep autoencoder to capture the network structure, it is not effective to exploit the context information of the new nodes. Compared to *VERSE* and *VERSE+A* that are trained over the new nodes, the scores of *KNE* and *KNE-C* are slightly behind and even exceeds them in sparse Intel network. By fully exploiting the subgraph information and node content, the kernel method is more effective to predict the links for the new nodes even when they currently have very few links.

Discussion. Note here that VERSE performs slightly beyond VERSE+A in datasets Cora and DBLP in Tables 5 and 6. The main reason is that the attribute of individual contains noises, and thereby the simple concatenation of structure and attribute has a negative effect in link prognostication. However, in KNE, we aggregate the attributes of neighbors and communities to filter the noise in the individual nodes in the statistical sense.

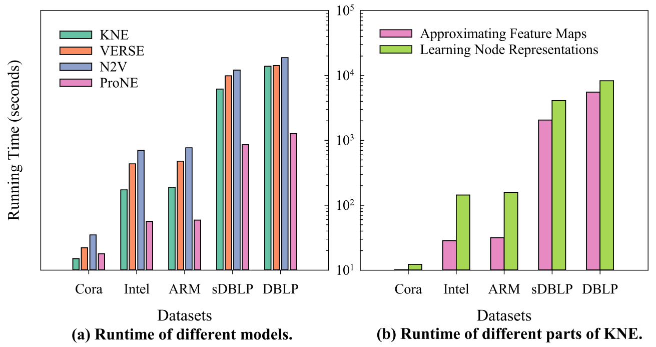

Fig. 5. Runtime results of different methods in (a) and different parts of *KNE* in (b).

5.7 Parameter Sensitivity and Time Sensitivity

There are several important parameters in the model, i.e., the dimension d of representation vector, hyper-parameters α used to adjust the importance of the community information, layers setting of $\mathcal{R}_v$ and sketch size b. This experiment is used to analyze the sensitivity of these parameters on dataset ARM, where the classification accuracy and AUC score of link inference for the new nodes are used for evaluation. The results are shown in Fig. 4. Generally, it is important to set the number of dimension for the latent embedding space for previous methods, but *KNE* is not very sensitive to d as shown in Fig. 4b. In Fig. 4a, the classification performance shows a reverse trend compared to that of link inference, and 0.1 is a reasonable choice to obtain a balance between classification and link inference. As shown in Figs. 4c and 4d, the sketch configuration affects the performance markedly. Fig. 4c shows that the combinations of all layers can achieve the best performances of classification, which indicates that each layer in $\mathcal{R}_v$ may preserve the structures and semantics information in each RKHS. The sketch size b is dependent on the size of k-gram terms for different datasets. However, when b is small, the model still performs well since the k-gram vectors are usually sparse. The figure shows that our model is not very sensitive to these parameters, which demonstrates the robustness of the model.

We now present the runtime results on four networks in Table 1. sDBLP is a subset of DBLP, which contains 465,322 nodes and 1,530,916 edges. We train *KNE* on a GPU RTX3090, where 3,000 instances are processed in parallel for each batch, and it is compared with other scalable methods, e.g., *VERSE*[5], *node2vec*[6] and *ProNE*[7], with C++ implementation and 16 running threads. For each method, we report the total wall-clock time, with graph loading and necessary preprocessing steps included. Due to different implementation and running environments, the experiment result in Fig. 5a merely provides an intuitive display of the time performance of *KNE* compared to the popular scalable methods. *KNE* is faster than

5. https://github.com/xgfs/verse
6. https://github.com/xgfs/node2vec-c
7. https://github.com/THUDM/ProNE

VERSE and *node2vec* when the number of nodes in the networks is small (e.g., Cora, ARM and Intel). However, this advantage is not sustainable when there are more nodes added to the network (e.g., sDBLP and DBLP). *KNE* processes 1.6×10^6 nodes in about 4 hours. *ProNE* performs well beyond *KNE* and other methods by a large margin. It mainly owes to the speed advantage of matrix factorization compared to the methods based on backpropagation and random walk. *KNE* and *ProNE* are two entirely different solutions that are designed to address different problems, i.e., sparse and large-scale problems respectively. *ProNE* is not effective to model sparse networks, where the abundant attributes cannot be effectively leveraged to supplement the structure information and thus the representation learning of the sparse nodes is affected. Our method is still scalable compared to other attribute-based methods e.g., DANE, ANSE and MVC-DB.

In addition, we analyze the runtime of different parts of *KNE*, i.e., generating sketches and training the convolutional network. Fig. 5b shows that the time of generating sketches increases when the network is larges. This is because we compute the sketches of each nodes linearly. Overall, although processing numerous attribute information, KNE is still comfortably the efficient and scalable method and obtains better time performance than *VERSE* and *node2vec* that only consider structure information.

6 Conclusion and Future Work

In this paper, we propose to take advantage of both the kernel methods and deep learning to address the sparsity problem of network embedding. Specifically, we propose a novel model deep kernel network embedding, which fully integrates the multiple information in the context of the target node. Experimental results on four real network datasets demonstrate the significant effectiveness, especially for generating representations for the new nodes. The novelty of this work is to alleviate the sparsity problem in network embedding by evaluating the expectation $\mathbb{E}_{G\sim \mathbf{P}^v} f(G)$ in an efficient way. This complements the current research which learns independent representation for each type of information based on the target node mainly.

There are several directions for future research. It would be interesting to investigate various types of graph kernels, such as directional graphs and weighted graphs, for network embedding.

References

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

[2] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.

[3] P. Cui, X. Wang, J. Pei, and W. Zhu, “A survey on network embedding,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, May 2019.

[4] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *Proc. 21st Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1753–1760.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[6] L. Backstrom and J. Leskovec, “Supervised random walks: Predicting and recommending links in social networks,” in *Proc. 4th ACM Int. Conf. Web Search Data Mining*, 2011, pp. 635–644.

[7] X. Du, J. Yan, and H. Zha, “Joint link prediction and network alignment via cross-graph embedding,” in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 2251–2257.

[8] S. Zhang and H. Tong, “FINAL: Fast attributed network alignment,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1345–1354.

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

[10] S. Cao, W. Lu, and Q. Xu, “GraRep: Learning graph representations with global structural information,” in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 891–900.

[11] C. Huang, B. Shi, X. Zhang, X. Wu, and N. V. Chawla, “Similarity-aware network embedding with self-paced learning,” in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2019, pp. 2113–2116.

[12] C. Shi, B. Hu, W. X. Zhao, and P. S. Yu, “Heterogeneous information network embedding for recommendation,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 357–370, Feb. 2019.

[13] Z. He, J. Liu, N. Li, and Y. Huang, “Learning network-to-network model for content-rich network embedding,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1037–1045.

[14] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, “Learning structural node embeddings via diffusion wavelets,” in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1320–1329.

[15] A. Zhiyuli, X. Liang, Y. Chen, and X. Du, “Modeling large-scale dynamic social networks via node embeddings,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 10, pp. 1994–2007, Oct. 2019.

[16] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, “Embedding temporal network via neighborhood formation,” in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 2857–2866.

[17] Q. Long, Y. Wang, L. Du, G. Song, Y. Jin, and W. Lin, “Hierarchical community structure preserving network embedding: A subspace approach,” in *Proc. ACM Int. Conf. Inf. Knowl. Manage.* 2019, pp. 409–418.

[18] D. Yang, S. Wang, C. Li, X. Zhang, and Z. Li, “From properties to links: Deep network embedding on incomplete graphs,” in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2017, pp. 367–376.

[19] L. Liao, X. He, H. Zhang, and T.-S. Chua, “Attributed social network embedding,” *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 12, pp. 2257–2270, Dec. 2018.

[20] S. Bandyopadhyay, N. Lokesh, and M. N. Murty, “Outlier aware network embedding for attributed networks,” in *Proc. 23rd AAAI Conf. Artif. Intell.*, 2019, pp. 12–19.

[21] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *Learn. Theory and Kernel Machines*, Berlin, Germany: Springer, 2003, pp. 129–143.

[22] P. Yanardag and S. V. N. Vishwanathan, “Deep graph kernels,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1365–1374.

[23] M. Draief, K. Kutzkov, K. Scaman, and M. Vojnovic, “KONG: Kernels for ordered-neighborhood graphs,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4051–4060.

[24] K. Muandet *et al.*, “Kernel mean embedding of distributions: A review and beyond,” *Foundations Trends Mach. Learn.*, vol. 10, no. 1–2, pp. 1–141, 2017.

[25] M. Charikar, K. Chen, and M. Farach-Colton, “Finding frequent items in data streams,” in *Proc. Int. Colloq. Automata Languages Program.*, 2002, pp. 693–703.

[26] N. Pham and R. Pagh, “Fast and scalable polynomial kernels via explicit feature maps,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 239–247.

[27] E. Hoffer and N. Ailon, “Deep metric learning using triplet network,” in *Proc. Int. Workshop Similarity-Based Pattern Recognit.*, 2015, pp. 84–92.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Proc. Int. Conf. Learn. Representations, Workshop Track*, 2013.

[29] H. Gao and H. Huang, “Self-paced network embedding,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1406–1415.

[30] Q. Dai, X. Shen, L. Zhang, Q. Li, and D. Wang, “Adversarial training methods for network embedding,” in *Proc. World Wide Web Conf.*, 2019, pp. 329–339.

[31] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *Proc. World Wide Web Conf.*, 2018, pp. 539–548.

[32] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[33] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1105–1114.

[34] 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 *Proc. 11th ACM Int. Conf. Web Search Data Mining*, 2018, pp. 459–467.

[35] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding, "ProNE: Fast and scalable network representation learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4278–4284.

[36] Y. Yin and Z. Wei, "Scalable graph embeddings via sparse transpose proximities," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1429–1437.

[37] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2015, pp. 4278–4284.

[38] J. Liu, Z. He, L. Wei, and Y. Huang, "Content to node: Self-translation network embedding," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1794–1802.

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[40] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 1895–1901.

[41] H. Gao and H. Huang, "Deep attributed network embedding," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 3364–3370.

[42] C. Leslie, E. Eskin, and W. S. Noble, "The spectrum kernel: A string kernel for SVM protein classification," *Pac. Symp. Biocomput.*, World Scientific: Singapore, 2002, pp. 566–575.

[43] J. Attenberg, A. Dasgupta, J. Langford, A. Smola, and K. Weinberger, "Feature hashing for large scale multitask learning," in *Proc. Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120.

[44] A. Smola, A. Gretton, L. Song, and B. Schölkopf, "A hilbert space embedding for distributions," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2007, pp. 13–31.

[45] M. Reed, *Methods of Modern Mathematical Physics: Functional Analysis*, Amsterdam, The Netherlands: Elsevier, 2012.

[46] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E Statist. Nonlinear Soft Matter Phys.*, vol. 76, no. 2, 2007, Art. no. 036106.

[47] J. Kim, S. Lim, J. Lee, and B. S. Lee, "Linkblackhole: Robust overlapping community detection using link embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2138–2150, Nov. 2019.

[48] D. Jin, X. You, W. Li, D. He, P. Cui, F. Fogelman-Soulié, and T. Chakraborty, "Incorporating network embedding into Markov random field for better community detection," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 160–167.

[49] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. 5th IEEE Int. Conf. Data Mining*, 2005, Art. no. 8.

[50] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proc. 12th Int. Conf. Artif. Intell. Statist.*, 2009, pp. 488–495.

[51] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[52] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1225–1234.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

Xiaoming Zhang received the BSc and MSc degrees in computer science and technology from the National University of Defense Technology, Changsha, China, in 2003 and 2007, respectively, and the PhD degree in computer science from Beihang University, Beijing, China, in 2012. He is currently with the School of Cyber Science and Technology, Beihang University, where he has been an assistant professor since 2012. He has authored or coauthored more than 40 papers, such as TOIS, TMM, TIP, TCYB, WWWJ, *Neurocomputing*, *Signal Processing*, ACM MM, AAAI, IJCAI, CIKM, ICMR, SDM, and EMNLP. His current research interests include social media analysis, image tagging, and text mining.

Feiran Huang (Member, IEEE) received the BSc degree from Central South University, Changsha, China, in 2011, and the PhD degree in computer software and theory from the School of Computer Science and Engineering, Beihang University, Beijing, China, in 2019. He is currently an assistant professor with the School of Information Scienc and Technology and College of Cyber Security, Jinan University, Guangzhou, China. He has authored or coauthored more than 30 papers, such as IEEE TIP, IEEE TCYB, IEEE TNNLS, ACM TOMM, ACM MM, ACM CIKM, and ACM ICMR. His research interests include social media analysis and multi-modal learning. He is an associate editor or an editorial borad menber for many journals, such as the *Computers*, *Electrical Engineering,* and *International Journal of Multimedia and Ubiquitous Engineering*. He is a member of ACM.

Ming Lu received the master's degree in software engineering in 2016 from Beihang University, Beijing, China, where he is currently working toward the PhD degree with the School of Cyber Science and Technology. His research interests include machine translation, multimodal data analysis, and data mining.

Shuai Ma received the PhD degrees from the University of Edinburgh, Edinburgh, U.K., in 2010, and Peking University, Beijing, China, in 2004. He is currently a professor with the School of Computer Science and Engineering, Beihang University, Beijing, China. He was a postdoctoral research fellow with the Database Group, University of Edinburgh, a summer intern with Bell Laboratories, Murray Hill, NJ, USA, and a visiting researcher of MRSA, Beijing, China. His current research interests include database theory and systems, social data and graph analysis, and data intensive computing. Dr. Ma was the recipient of the Best Paper Award for International Conference on Very Large Data Bases 2010 and the Best Challenge Paper Award for International Conference on Web Information Systems Engineering 2013.