Complexity:
Avg: "Linear to size of Graph"
Worst: $O(|V|r\ell + |V| + K + T_1 \times (T_2 \cdot |V| \cdot K + E + |V|r\ell + |V|K)$

Shape of Data (worst):
Accuracy
Performance

Datasets:
① Karate Club
② BlogCatalog
③ Flickr
④ Wikipedia

$\boxed{\text{Name: ComE}}$

- Doesn't Really give any indication of ET performance
- Doesn't compare to non-embedding methods

uses conductance + NMI as metrics (std for communities??)

From Node Embedding To Community Embedding[*]

~~Vincent~~ W. Zheng[1], Sandro Cavallari[2], Hongyun Cai[1], Kevin Chen-Chuan Chang[3], Erik Cambria[2]
[1] Advanced Digital Sciences Center, Singapore;
[2] Nanyang Technological University, Singapore;
[3] University of Illinois at Urbana-Champaign, USA
{vincent.zheng, hongyun.c}@adsc.com.sg, sandro001@e.ntu.edu.sg,
kcchang@illinois.edu, cambria@ntu.edu.sg

Abstract

In this paper, we introduce a new setting for graph embedding, which considers embedding communities instead of individual nodes. Community embedding is useful as a natural community representation for applications, and it provides an exciting opportunity to improve community detection. Specifically, we see the interaction between community embedding and detection as a closed loop, through node embedding. On the one hand, we rely on node embedding to generate good communities and thus meaningful community embedding. On the other hand, we apply community embedding to improve node embedding through a novel community-aware higher-order proximity. This closed loop enables us to improve community embedding, community detection and node embedding at the same time. Guided by this insight, we propose *ComE*, the first community embedding method so far as we know. We evaluate ComE on multiple real-world data sets, and show ComE outperforms the state-of-the-art baselines in both tasks of community prediction and node classification. Our code is available at `https://github.com/andompesta/nodeembedding-to-communityembedding`.

1 Introduction

Traditionally, graph embedding focuses on individual *nodes*, which aims to output a vector representation for each node in the graph, such that two nodes "close" on the graph have similar vector representations in a low-dimensional space. Such node embedding has been shown very successful in preserving the network structure, and significantly improving a wide range of applications, including node classification [Cao *et al.*, 2015; Perozzi *et al.*, 2014], node clustering [Tian *et al.*, 2014; Yang *et al.*, 2016], link prediction [Grover and Leskovec, 2016; Ou *et al.*, 2016], graph visualization [Tang *et al.*, 2015; Wang *et al.*, 2016] and more [Fang *et al.*, 2016; Niepert *et al.*, 2016].

[*]This is a technical report for paper [Cavallari *et al.*, 2017].

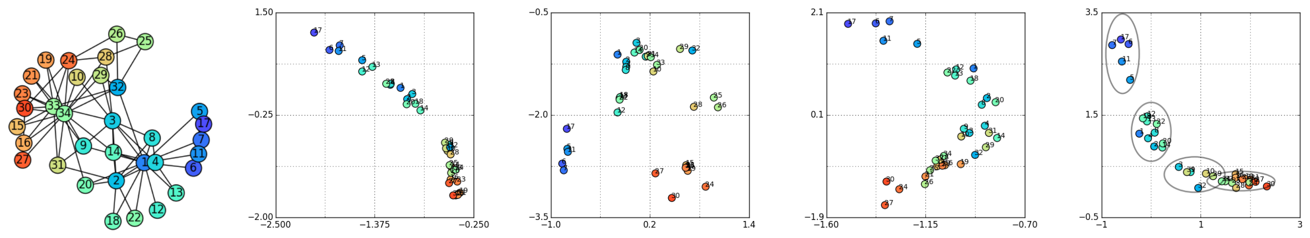
$$\begin{tikzcd}
\textit{Community Detection} \arrow[rr, "①"] & & \textit{Community Embedding} \arrow[d, "②"] \\
& \textit{Node Embedding} \arrow[ul, "③"] & \arrow[l]
\end{tikzcd}$$

Figure 1: Close loop for community detection & embedding.

In this paper, we study a new setting for graph embedding, which focuses on embedding *communities*. Generally, a "community embedding" is a representation for a community in a low-dimensional space. Since a community is a group of densely connected nodes, a community embedding cannot simply be a vector representation like a node embedding. Instead, a community embedding should be a set of *random variables*, for a distribution characterizing how the community member nodes spread in the low-dimensional space. For instance, if we consider multivariate Gaussian as the distribution, then a community embedding consists of a mean and a covariance. Surprisingly, despite the success of node embedding, the concept of community embedding is still missing in the literature (more discussions in Sect. 2).

Community embedding is useful. As a representation of community, it naturally supports community-level applications, such as visualizing communities to help generate insights for graph analysis, finding similar communities to assist community recommendation and so on. Besides, it also provides an exciting opportunity to improve community detection. Specifically, to find a useful community embedding, we first need to have a good community. Since recent graph embedding has shown to be an effective way for graph analytics, we wish to leverage node embedding to find good communities. Most recent node embedding methods such as DeepWalk [Perozzi *et al.*, 2014], LINE [Tang *et al.*, 2015] and node2vec [Grover and Leskovec, 2016] focus on preserving *first-order* and/or *second-order* proximity, which can ensure two nodes directly linked or sharing "context" to have similar embeddings. But they are not aware of community structures, as shown in Fig. 2(b)–2(d), where we visualize the node embedding for these methods on Zachary's karate club graph [Zachary, 1977]. Thus a simple pipeline approach to first detect communities and then aggregate their nodes' embeddings as community embedding (*i.e.*, ① in Fig. 1) is limited due to suboptimal detection. Community embedding can avoid such a limitation– suppose we already have some community embedding, then we can come back to optimize the

Dataset.



(a) Karate club graph (b) DeepWalk (c) LINE (d) node2vec (e) ComE (ours)

Figure 2: Graph embedding of the Zachary's karate club graph in a 2D space. In (b)–(e), we input the same data to different methods for graph embedding. For each node, we sample 10 paths, each of which has a length of 80. We set context window size as 5 and negative sampling size as 5. In LINE, we use mean pooling of the first-order proximity embedding and the second-order proximity embedding for each node. In node2vec, we set $p = 0.25$, $q = 0.25$. In ComE, we set $\alpha = 1$, $\beta = 1$. In (e), we also plot the community embeddings as four eclipses, characterizing the distribution of four detected communities.

node embedding (*i.e.*, ② in Fig. 1), by "squashing" the nodes to scatter closely near their community centers. The resulting node embedding thus becomes community-aware and hopefully improves the community detection (*i.e.*, ③ in Fig. 1). In Fig. 2(e), we show a promising result to have community-aware node embedding and better communities, with the help of community embedding.

To close the loop between community detection and community embedding is non-trivial. A shown in Fig. 1 ②, we try to use community embedding to guide node embedding, but it is not clear how to do it. There is work on using node embedding to improve community detection [Cao *et al.*, 2015; Kozdoba and Mannor, 2015; Tian *et al.*, 2014]. But there is very little work on the other way around using explicit community information to enhance node embedding [Yang *et al.*, 2016], and it needs extra supervision of must-links to enforce the community in node embedding.

Our insight for addressing the community embedding feedback is to see community embedding as providing a novel *community-aware higher-order proximity*, such that two nodes in the same community can be close in the low-dimensional space. In contrast with first-order proximity and second-order proximity, community embedding does not require two nodes to be directly linked or share many "contexts" for being close. There is little work on higher-order proximity [Cao *et al.*, 2015; Ou *et al.*, 2016], and they all rely on embedding a higher-order adjacency matrix, which can easily have a quadratic number of non-zero entries. As a result, their computation can be expensive. Besides, their higher-order proximity does not explicitly model community. Another advantage of seeing community embedding as a higher-order proximity is that, we can combine it with the first-order and the second-order proximity for node embedding through the well-established neural network framework.

Guided by the above insight, we propose *ComE*, the first Community Embedding model for graph analytics. To represent a community, we are inspired by the Gaussian Mixture Model [Bishop, 2006] to see each community as a Gaussian component. Thus we formulate a community embedding as a tuple of a mean vector indicating the center of a community and a covariance matrix indicating the spread of its members in a low-dimensional space. To realize the closed loop in Fig. 1, we develop an iterative inference algorithm for *ComE*. On the one hand, given node embeddings, we can detect and embed the communities, such that a good community assignment and a good community embedding should explain the node embedding well. On the other hand, given community assignment and community embedding, we can optimize the node embedding by enforcing all different orders of proximity. As shown in Sect. 4, our inference algorithm's complexity is linear to the graph size.

We summarize our contributions as follows.

- To the best of our knowledge, we are the first to introduce the concept of community embedding to graph analytics.
- We identify that communty embedding can improve community detection, thus closing the loop between them.
- We contribute a novel *ComE* model to close the loop and a scalable inference algorithm to detect communities and infer their embeddings at the same time.
- We evaluate *ComE* on three real-world data sets. It improves the state-of-the-art baselines by at least 2.5%–7.8% (NMI) and 1.1%–2.8% (conductance) in community detection, 9.52%–43.5% (macro-F1) and 6.9%–19.4% (micro-F1) in node classification.

2 Related Work

There has been an increasing amount of graph data, ranging from social networks such as Twitter to various information networks such as Wikipedia. An important question in graph analytics is how to represent a graph. Graph embedding is a popular graph representation framework, which aims to project a graph into a low-dimensional space for further applications [Tenenbaum *et al.*, 2000; Perozzi *et al.*, 2014].

In terms of the target to embed, most graph embedding methods focus on nodes. For example, earlier methods, such as MDS [Cox and Cox, 2000], LLE [Roweis and Saul, 2000], IsoMap [Tenenbaum *et al.*, 2000] and Laplacian eigenmap [Belkin and Niyogi, 2001], typically aim to solve the leading eigenvectors of graph affinity matrices as node embedding. Recent methods typically rely on neural networks to learn the representation for each node, with either shallow architectures [Xie *et al.*, 2016; Tang *et al.*, 2015; Grover and Leskovec, 2016] or deep architectures [Niepert *et al.*, 2016; Wang *et al.*, 2016; Chang *et al.*, 2015]. Other than node embedding, there is some attempt to learn edge embedding in a knowledge base [Luo *et al.*, 2015] or proximity embedding between two possibly distant nodes in a general heterogeneous graph [Liu *et al.*, 2017]. But there is no community embedding so far as we know.

In terms of the information to preserve, most graph embedding methods try to preserve first-order proximity and/or second-order proximity [Perozzi *et al.*, 2014; Grover and Leskovec, 2016; Tang *et al.*, 2015; Wang *et al.*, 2016]. Some recent attempts consider higher-order proximity, by factorizing a higher-order node-node proximity matrix by PageRank or Katz index [Cao *et al.*, 2015; Ou *et al.*, 2016]. Hence their higher-order proximity is based on the graph reachability via random walk, where the notion of community is missing. In contrast, our community embedding tries to preserve all the first-, second- and community-aware higher-order proximity.

In terms of interaction between node and community, there are a few graph embedding models that use node embedding to assist community detection [Tian *et al.*, 2014; Kozdoba and Mannor, 2015], but they do not have the notion of community in their node embedding. There is little work that allows community feedback to guide the node embedding [Yang *et al.*, 2016], but it lacks the concept of community embedding and its community feedback requires extra supervision on must-links. In contrast, we optimize node embedding, community embedding and community detection in a closed loop, and let them reinforce each other.

3 Problem Formulation

As *input*, we are given a graph $G = (V, E)$, where V is the node set and E is the edge set. Traditional graph embedding aims to learn a node embedding for each $v_i \in V$ as $\boldsymbol{\phi}_i \in \mathbb{R}^d$. In this paper, we introduce the concept of community embedding. Suppose there are K communities on the graph G. For each node v_i, we denote its community assignment as $z_i \in \{1, ..., K\}$. Motivated by Gaussian Mixture Model (GMM), we represent each community as a Gaussian component, which is characterized by a mean vector indicating the community center and a covariance matrix indicating its member nodes' spread. Formally, we define:

Definition 1 *A community k's embedding (where $k = 1, 2, ..., K$) in a d-dimensional space is a set of random variables $(\boldsymbol{\psi}_k, \Sigma_k)$, where $\boldsymbol{\psi}_k \in \mathbb{R}^d$ is the mean and $\Sigma_k \in \mathbb{R}^{d\times d}$ is the covariance for a multivariate Gaussian distribution.*

As *output*, we aim to learn both the community embedding $(\boldsymbol{\psi}_k, \Sigma_k)$ for each community $k \in \{1, ..., K\}$ and the node embedding $\boldsymbol{\phi}_i$ for each node $v_i \in V$ on the graph G.

Next, we model the closed loop in Fig. 1.

3.1 Community Detection and Embedding

Given node embedding, one straightforward way to detect communities and learn their embedding is to take a pipeline approach. For example, as shown in Fig. 1, one can first run K-means to detect communities, and then fit a Gaussian mixture for each community. However, such a pipeline approach lacks a unified objective function, thus hard to optimize later with node embedding. Alternatively, we can do community detection and embedding together in one single objective function based on Gaussian Mixture Model. That is, we consider each node v_i's embedding $\boldsymbol{\phi}_i$ as generated by a multivariate Gaussian distribution from a community $z_i = k$. Then, for all the nodes in V, we have the likelihood as

$$\prod_{i=1}^{|V|} \sum_{k=1}^{K} p(z_i = k) p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k), \quad (1)$$

where $p(z_i = k)$ is the probability of node v_i belonging to community k. For notation simplicity, we denote $p(z_i = k)$ as π_{ik}; thus we have $\pi_{ik} \in [0, 1]$ and $\sum_{k=1}^{K} \pi_{ik} = 1$. In community detection, these π_{ik}'s indicate the *mixed community membership* for each node v_i, and they are unknown. Besides, $p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k)$ is a multivariate Gaussian distribution defined as follows

$$p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k) = \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k). \quad (2)$$

In community embedding, the $(\boldsymbol{\psi}_k, \Sigma_k)$'s are unknown. By optimizing Eq. 1 *w.r.t.* π_{ik}'s and $(\boldsymbol{\psi}_k, \Sigma_k)$'s, we achieve community detection and embedding at the same time.

3.2 Node Embedding

Traditionally, node embedding focuses on preserving first- or second-order proximity. For example, to preserve first-order proximity, LINE [Tang *et al.*, 2015] enforces two neighboring nodes to have similar embedding by minimizing

$$O_1 = -\sum_{(v_i, v_j)\in E} \log \sigma(\boldsymbol{\phi}_j^T \boldsymbol{\phi}_i), \quad (3)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is a sigmoid function.

To preserve second-order proximity, LINE and DeepWalk [Perozzi *et al.*, 2014] both enforce two nodes sharing many "contexts" (i.e., neighbors within ζ hops) to have similar embedding. In this case, each node has two roles: a node for itself and a context for some other nodes. To differentiate such roles, LINE introduces an extra context embedding for each node v_j as $\boldsymbol{\phi}'_j \in \mathbb{R}^d$. Denote C_i as the set of contexts for node v_i. Then LINE adopts *negative sampling* [Mikolov *et al.*, 2013] to define a function for measuring how well v_i generates each of its contexts $v_j \in C_i$ as

$$\Delta_{ij} = \log \sigma(\boldsymbol{\phi}'^T_j \boldsymbol{\phi}_i) + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)}[\log \sigma(-\boldsymbol{\phi}'^T_l \boldsymbol{\phi}_i)], \quad (4)$$

where $v_l \sim P_n(v_l)$ denotes sampling a node $v_l \in V$ as a "negative context" of v_i according to a probability $P_n(v_l)$. We set $P_n(v_l) \propto r_l^{3/4}$ as proposed in [Mikolov *et al.*, 2013], where r_l is v_l's degree. In total, there are m negative contexts. Generally, maximizing Eq. 4 enforces node v_i's embedding $\boldsymbol{\phi}_i$ to best generate its positive contexts $\boldsymbol{\phi}'_j$'s, but not its negative contexts $\boldsymbol{\phi}'_l$'s. Then we can minimize the following objective function to preserve the second-order proximity:

$$O_2 = -\alpha \sum_{v_i\in V} \sum_{v_j\in C_i} \Delta_{ij}, \quad (5)$$

where $\alpha > 0$ is a trade-off parameter.

3.3 Closing the Loop

In order to close the loop in Fig. 1, we need to enable the feedback from community detection and community embedding to node embedding. Suppose we have identified the mixed community membership π_{ik}'s and the community embedding $(\boldsymbol{\psi}_k, \Sigma_k)$'s in Sect. 3.1. Then we can re-use Eq. 1 to enable such feedback, by seeing the node embedding $\boldsymbol{\phi}_i$'s as *unknown*. Effectively, optimizing Eq. 1 *w.r.t.* $\boldsymbol{\phi}_i$'s enforces the nodes $\boldsymbol{\phi}_i$'s within the same community to get closer to the corresponding community center $\boldsymbol{\psi}_k$. That is, two nodes sharing a community are likely to have similar embedding. Compared with the first- and second-order proximity, this design introduces a new *community-aware higher-order proximity* to node embedding, which is useful for community detection and embedding later. For example, in Fig. 2(a), node 3 and node 10 are directly linked, but node 3 is closer to the blue-green community, whereas node 10 is closer to the red-yellow community. Therefore, by only preserving first-order proximity, we may not tell their community membership's difference well. For another example, node 9 and node 10 share a number of one-hop and two-hop neighbors, but compared with node 10, node 9 is closer to the blue-green community. Therefore, by only preserving second-order proximity, we may not tell their community membership's difference well, as shown in Fig. 2(b) and Fig. 2(d).

Based on the closed loop, we will optimize community detection, community embedding and node embedding together. We have three types of proximity to consider for node embedding, including first-, second- and higher-order proximity. In general, there are two approaches to combine different types of proximity for node embedding: 1) "concatenation", *e.g.*, LINE first separately optimizes O_1 and O_2, then it concatenates the two resulting embedding for each node into a long vector as the final output; 2) "unification", *e.g.*, SDNE [Wang *et al.*, 2016] learns a single node embedding for each node to preserve both first- and second-order proximity at the same time. In this paper, to encourage the node embedding to unify multiple types of proximity, we adopt the unification approach, and leave the other approach as future work. Consequently, based on Eq. 1, we first define the objective function for community detection and embedding, as well as enforcing the higher-order proximity for node embedding as:

$$O_3 = -\frac{\beta}{K} \sum_{i=1}^{|V|} \log \sum_{k=1}^{K} \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k), \qquad (6)$$

where $\beta \geq 0$ is a trade-off parameter. Denote $\Phi = \{\boldsymbol{\phi}_i\}$, $\Phi' = \{\boldsymbol{\phi}'_i\}$, $\Pi = \{\pi_{ik}\}$, $\Psi = \{\boldsymbol{\psi}'_k\}$ and $\Sigma = \{\Sigma_k\}$ for $i = 1, ..., N, k = 1, ..., K$. Then, we also unify the first- and second-order proximity for node embedding, thus reaching the ultimate objective function for *ComE* as

$$\mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) = O_1(\Phi) + O_2(\Phi, \Phi') + O_3(\Phi, \Pi, \Psi, \Sigma). \qquad (7)$$

Our final optimization problem becomes:

$$(\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*) \leftarrow \underset{\forall k, diag(\Sigma_k) > \mathbf{0}}{\arg\min} \; \mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma), \qquad (8)$$

where $diag(\Sigma_k)$ returns the diagonal entries of Σ_k. We particularly introduce a constraint of $diag(\Sigma_k) > \mathbf{0}$ for each $k \in \{1, ..., K\}$ to avoid the singularity issue of optimizing $\mathcal{L}$. Similar to Gaussian Mixture model ([Bishop, 2006], there exists degenerated solutions for optimizing $\mathcal{L}$ without any constraint. That is, when a Gaussian component collapses to a single point, the $diag(\Sigma_k)$ becomes zero, which makes O_3 become negative infinity.

4 Inference

We decompose the optimization of Eq. 8 into two parts, and take an iterative approach to solve it. Specifically, we consider iteratively optimizing (Π, Ψ, Σ) with a constrained minimization given (Φ, Φ'), and optimizing (Φ, Φ') with an unconstrained minimization given (Π, Ψ, Σ). Empirically, this iterative optimization algorithm converges quickly with a reasonable initialization; *e.g.*, we initialize (Φ, Φ') by DeepWalk results in our experiments. We report the convergence in Sect. 5.3. Next we detail this iterative optimization.

Fix (Φ, Φ')**, optimize** (Π, Ψ, Σ). In this case, Eq. 7 is simplified as the negative log-likelihood of a GMM. According to [Bishop, 2006], we can optimize (Π, Ψ, Σ) by EM, and obtain a closed-form solution as

$$\pi_{ik} = \frac{N_k}{|V|}, \qquad (9)$$

$$\boldsymbol{\psi}_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} \boldsymbol{\phi}_i, \qquad (10)$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} (\boldsymbol{\phi}_i - \boldsymbol{\psi}_k)(\boldsymbol{\phi}_i - \boldsymbol{\psi}_k)^T, \qquad (11)$$

where $\gamma_{ik} = \frac{\pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k)}{\sum_{k'=1}^{K} \pi_{ik'} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_{k'}, \Sigma_{k'})}$ and $N_k = \sum_{i=1}^{|V|} \gamma_{ik}$.

Fix (Π, Ψ, Σ)**, optimize** (Φ, Φ'). In this case, Eq. 7 is simplified as optimizing the node embedding with three types of proximity. Due to the summation within the logarithm term of O_3, it is inconvenient to compute the gradient of $\boldsymbol{\phi}_i$. Thus we try to minimize an upper bound of $\mathcal{L}(\Phi, \Psi, \Sigma)$ instead:

$$O'_3 = -\frac{\beta}{K} \sum_{i=1}^{|V|} \sum_{k=1}^{K} \pi_{ik} \log \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k). \qquad (12)$$

It is easy to prove that $O'_3(\Phi; \Pi, \Psi, \Sigma) \geq O_3(\Phi; \Pi, \Psi, \Sigma)$, due to log-concavity $\sum_{i=1}^{|V|} \log \sum_{k=1}^{K} \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k) \geq \sum_{i=1}^{|V|} \sum_{k=1}^{K} \log \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k)$. As a result, we define

$$\mathcal{L}'(\Phi, \Phi') = O_1(\Phi) + O_2(\Phi, \Phi') + O'_3(\Phi; \Pi, \Psi, \Sigma),$$

and thus $\mathcal{L}'(\Phi, \Phi') \geq \mathcal{L}(\Phi, \Phi')$. We optimize $\mathcal{L}'(\Phi, \Phi')$ by *stochastic gradient descent* (SGD). For each $v_i \in V$, we have

$$\frac{\partial O_1}{\partial \boldsymbol{\phi}_i} = -\sum_{(i,j) \in E} \sigma(-\boldsymbol{\phi}_j^T \boldsymbol{\phi}_i) \boldsymbol{\phi}_j, \qquad (13)$$

$$\frac{\partial O_2}{\partial \boldsymbol{\phi}_i} = -\alpha \sum_{v_j \in C_i} \Big[\sigma(-\boldsymbol{\phi}'^T_j \boldsymbol{\phi}_i) \boldsymbol{\phi}'_j + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)} [\sigma(\boldsymbol{\phi}'^T_l \boldsymbol{\phi}_i)(-\boldsymbol{\phi}'_l)] \Big], \qquad (14)$$

$$\frac{\partial O'_3}{\partial \boldsymbol{\phi}_i} = \frac{\beta}{K} \sum_{k=1}^{K} \pi_{ik} \Sigma_k^{-1} (\boldsymbol{\phi}_i - \boldsymbol{\psi}_k). \qquad (15)$$

We also compute the gradient for context embedding as

$$\frac{\partial O_2}{\partial \boldsymbol{\phi}'_j} = -\alpha \sum_{v_i \in V} \Big[\delta(v_j \in C_i) \sigma(-\boldsymbol{\phi}'^T_j \boldsymbol{\phi}_i) \boldsymbol{\phi}_i + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)} [\delta(v_l = v_j) \sigma(\boldsymbol{\phi}'^T_l \boldsymbol{\phi}_i)(-\boldsymbol{\phi}_i)] \Big]. \qquad (16)$$

Algorithm 1 ComE

```
Require: graph $G = (V, E)$, #(community) $K$, #(paths per node) $\gamma$, walk length $\ell$, context size $\zeta$, embedding dimension $d$, negative context size $m$, parameters $(\alpha, \beta)$.
Ensure: node embedding $\Phi$, context embedding $\Phi'$, community assignment $\Pi$, community embedding $(\Psi, \Sigma)$.
1:  $\mathcal{P} \leftarrow$ SamplePath$(G, \ell)$;
2:  Initialize $\Phi$ and $\Phi'$;
3:  for $iter = 1 : T_1$ do
4:    for $subiter = 1 : T_2$ do
5:      Update $\pi_{ik}$, $\psi_k$ and $\Sigma_k$ by Eq. 9, Eq. 10 and Eq. 11;
6:    for all edge $(i, j) \in E$ do
7:      SGD on $\phi_i$ and $\phi_j$ by Eq. 13;
8:    for all path $p \in \mathcal{P}$ do
9:      for all $v_i$ in path $p$ do
10:       SGD on $\phi_i$ by Eq. 14;
11:       SGD on its context $\phi'_j$'s within $\zeta$ hops by Eq. 16;
12:   for all node $v_i \in V$ do
13:     SGD on $\phi_i$ by Eq. 15;
```

Algorithm and complexity. We summarize ComE in Alg. 1. In line 1, for each $v_i \in V$, we sample γ paths starting from v_i with length ℓ on G. In lines 4–5, we fix (Φ, Φ') and optimize (Π, Ψ, Σ) for community detection and embedding. In lines 6–13, we fix (Π, Ψ, Σ) and optimize (Φ, Φ') for node embedding. Specifically, we update node embedding by first-order proximity (lines 6–7), second-order proximity (lines 8–11) and community-aware higher-order proximity (lines 12–13).

We analyze time complexity of Alg. 1. Path sampling in line 1 takes $O(|V|\gamma\ell)$. Parameter initialization in line 2 takes $O(|V| + K)$. Community detection and embedding in line 5 takes $O(|V|K)$. Node embedding *w.r.t.* first-order proximity in lines 6–7 takes $O(|E|)$. Node embedding *w.r.t.* second-order proximity in lines 8–11 takes $O(|V|\gamma\ell)$. Node embedding *w.r.t.* community-aware higher-order proximity in lines 12–13 takes $O(|V|K)$. In total, the complexity is $O(|V|\gamma\ell + |V| + K + T_1 \times (T_2|V|K + |E| + |V|\gamma\ell + |V|K))$, which is linear to the graph size (i.e., $|V|$ and $|E|$).

5 Experiments

We want to quantitatively evaluate community embedding. Firstly, as an important application of community embedding, we evaluate the resulting community detection (Sect. 5.1). Secondly, as the loop between community detection and embedding is closed through node embedding, we also evaluate the resulting node embedding. In particular, following the previous work [Perozzi *et al.*, 2014; Tang *et al.*, 2015], we consider node classification (Sect. 5.2). Finally, we study our model's convergence and parameter sensitivity (Sect. 5.3).

Data sets. We use three public data sets[1], as listed in Table 1.

DATASETS

Table 1: Data sets.

	#(node)	#(edge)	#(node labels)	labels per node
BlogCatalog	10,312	333,983	39	single-label
Flickr	80,513	5,899,882	195	single-label
Wikipedia	4,777	184,812	40	multi-label

Evaluation metrics. In community detection, we use both *conductance* [Kloster and Gleich, 2014] and *normalized mutual information* (NMI) [Tian *et al.*, 2014]. In node classification, we use *micro-F1* and *macro-F1* [Perozzi *et al.*, 2014].

Baselines. We design baselines to answer two questions: 1) is it necessary to model community detection, community embedding and node embedding together? 2) is it necessary to enable the feedback from community embedding to community detection through node embedding? To answer the first question, we introduce a simple pipeline approach.

- *Pipeline*: it first detects communities by GMM based on the graph adjacency matrix. Then, it uses LINE to generate first- and second-order node embedding. Finally, it fits a multivariate Gaussian in each community based on its member nodes' embedding as the community embedding.

To answer the second question, we apply the state-of-the-art methods to generate node embedding, based on which we further use GMM to detect and embed communities.

- *DeepWalk* [Perozzi *et al.*, 2014]: it models second-order proximity for node embedding.
- *LINE* [Tang *et al.*, 2015]: it models both first- and second-order proximity for node embedding.
- *node2vec* [Grover and Leskovec, 2016]: it extends DeepWalk by exploiting homophily and structural roles.
- *GraRep* [Cao *et al.*, 2015]: it models random walk based higher-order proximity for node embedding.
- *SAE* [Tian *et al.*, 2014]: it uses sparse auto-encoder, which is shown to share a similar objective as spectral clustering, on the adjacency matrix for node embedding.

We try to compare with all baselines on all data sets, using the codes released by their authors. However, we cannot produce results for *node2vec* and *GraRep* on Flickr due to unmanageable out-of-memory errors on a machine with 64GB memory. Thus we exclude them from comparison on Flickr.

Parameters and environment. In *DeepWalk*, *node2vec* and ComE, we set $\gamma = \zeta = 10$, $\ell = 80$, $m = 5$. In *node2vec*, we set $p = q = 0.25$ according to the best results in [Grover and Leskovec, 2016]. We set the embedding dimension $d = 128$ for all methods. We set K as the number of unique labels in each data set. We run experiments on Linux machines with eight 3.50GHz Intel Xeon(R) CPUs and 16GB memory.

5.1 Community Detection

In community prediction, our goal is to predict the most likely community assignment for each node. Since our data sets are labeled, we set the number of communities K as the number of distinct labels in the data set. As an unsupervised task, we use the whole graph for learning embeddings and then predicting communities for each node. Note that in Wikipedia, one node has multiple labels, thus we predict the

[1]Available at http://socialcomputing.asu.edu/pages/datasets and https://snap.stanford.edu/node2vec/POS.mat.

Doesn't compare to non-embedding Methods.] - NO Timing eval.

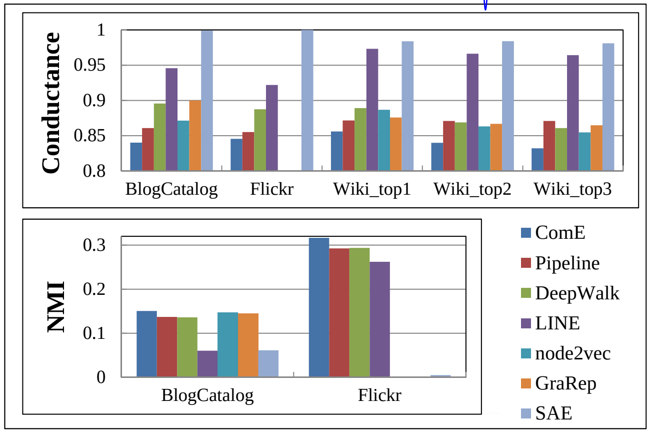


Figure 3: Results on community prediction. The smaller conductance is, the better. The bigger NMI is, the better.

top $N(N = 1, 2, 3)$ communities for each node, the results are denoted as Wiki_top$N(N = 1, 2, 3)$ in Fig. 3.

We compare ComE with the baselines for community prediction. We set $\alpha = 0.1, \beta = 0.01$ in ComE for all three data sets. As the baselines do not consider community in their embeddings, for fair comparison, we apply GMM over all the methods' node embedding outputs for community prediction. As shown in Fig. 3, ComE is consistently better than the baselines in terms of both conductance and NMI. Specifically, ComE improves the best baseline in all data sets by relative 1.1%–2.8% (conductance) and 2.5%–7.8% (NMI). This improvement suggests that, for community prediction, modeling community together with node embedding is better than doing them separately. Note that we do not calculate NMI for Wikipeida as it is multilabel dataset.

5.2 Node Classification

In node classification, our goal is to classify each node into one (or more) labels. We follow [Perozzi *et al.*, 2014] to first train the embeddings on the whole graph, then randomly split 10% (BlogCatalog and Wikipedia) and 90% (Flickr) of nodes as test data, respectively. We use the remaining nodes to train a classifier by LibSVM ($c = 1$ for all methods) [Chang and Lin, 2011]. We repeat 10 splits and report the average results.

We compare ComE with the baselines for node classification. We set $\alpha = 0.1$, $\beta = 0.01$ for BlogCatalog and Wikipedia, and $\alpha = 0.1$, $\beta = 0.1$ for Flickr. We vary the number of training data to build the classifiers for each method's node embeddings. As shown in Fig. 4, ComE is generally better than the baselines in terms of both macro-F1 and micro-F1. Specifically, ComE improves the best baselines by relatively 9.52%–43.5% (macro-F1) and 6.9%–19.4% (micro-F1), when using 70% (BlogCatalog and Wikipedia) and 8% (Flickr) of nodes for training. Our student t-tests show that all the above relative improvements are significant over the 10 data splits, with one-tailed p-values always less than 0.01. It is interesting to see ComE improves the baselines on node classification, since community embedding is after all unsupervised and it does not directly optimize the classification loss. This suggests that the higher-order proximity from community embedding does contribute to a better node embedding.

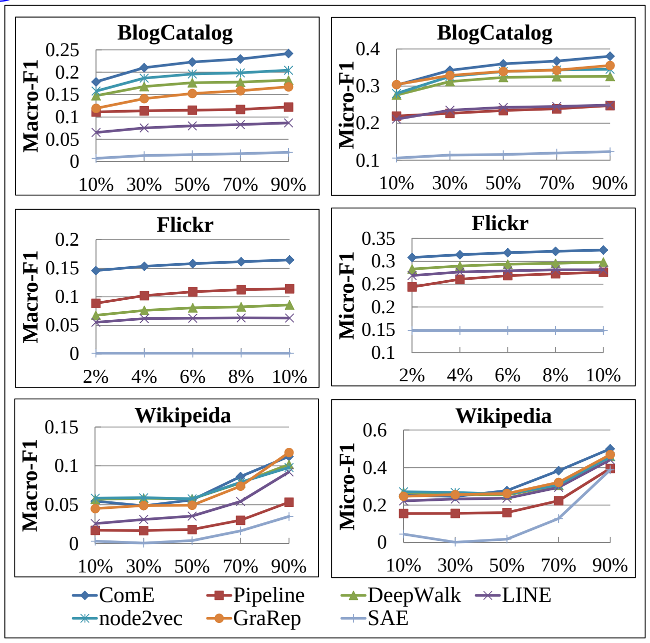


Figure 4: Results on node classification. The bigger macro-F1 and micro-F1 are, the better.

5.3 Model Study

We test different values for the model parameters α and β. As shown in Fig. 5, generally $\alpha = 0.1$ gives the best results for community prediction and node classification. This suggests keeping an appropriate trade off for the second-order proximity in the objective function is necessary. $\beta = 0.1$ is generally the best for both tasks. In general, when α and β are within the range of $[0.001, 1]$, the model performance is quite robust. We further validate the convergence and efficiency of ComE. As shown in Fig 6, the loss of ComE (normalized by $|V|$) converges quickly within a few iterations and the processing time of ComE is linear to the graph size (i.e., $|V|$ and $|E|$).

6 Conclusion

In this paper, we study the problem of embedding communities on graph. The problem is new because most graph embedding methods focus on individual nodes, instead of a group of nodes. We observe that community embedding and node embedding reinforce each other. On one hand, a good community embedding helps to get a good node embedding, because it preserves the community structure during embedding. On the other hand, a good node embedding also helps to get a good community embedding, as clustering is then done over the nodes with good representations. We jointly optimize node embedding and community embedding. We evaluate our method on the real-world data sets, and show that it outperforms the state-of-the-art baselines by at least 2.5%–7.8% (NMI) and 1.1%–2.8% (conductance) in community prediction, 9.52%–43.5% (macro-F1) and 6.9%–19.4%

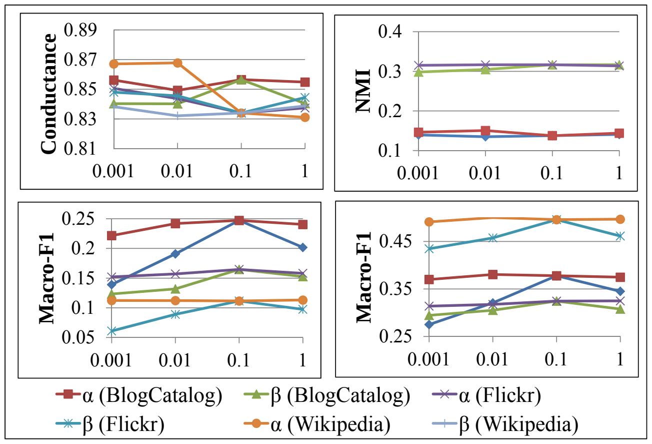


Figure 5: Impact of the parameters α and β. By default, $\alpha = 0.1$, $\beta = 0.1$.

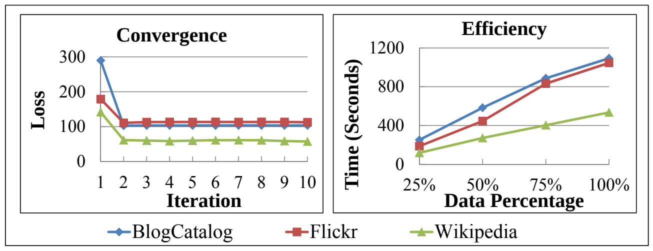


Figure 6: Model's Convergence and Efficiency

(micro-F1) in node classification.

References

[Belkin and Niyogi, 2001] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591, 2001.

[Bishop, 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

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

[Cavallari *et al.*, 2017] Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. Learning community embedding with community detection and node embedding on graphs. In *CIKM*, 2017.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[Chang *et al.*, 2015] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. Heterogeneous network embedding via deep architectures. In *KDD*, pages 119–128, 2015.

[Cox and Cox, 2000] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000.

[Fang *et al.*, 2016] Hanyin Fang, Fei Wu, Zhou Zhao, Xinyu Duan, Yueting Zhuang, and Martin Ester. Community-based question answering via heterogeneous social network learning. In *AAAI*, pages 122–128, 2016.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[Kloster and Gleich, 2014] Kyle Kloster and David F. Gleich. Heat kernel based community detection. In *KDD*, pages 1386–1395, 2014.

[Kozdoba and Mannor, 2015] Mark Kozdoba and Shie Mannor. Community detection via measure space embedding. In *NIPS*, pages 2890–2898, 2015.

[Liu *et al.*, 2017] Zemin Liu, Vincent W. Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*, 2017.

[Luo *et al.*, 2015] Yuanfei Luo, Quan Wang, Bin Wang, and Li Guo. Context-dependent knowledge graph embedding. In *EMNLP*, pages 1656–1661, 2015.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

[Ou *et al.*, 2016] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, pages 1105–1114, 2016.

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

[Roweis and Saul, 2000] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

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

[Tenenbaum *et al.*, 2000] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[Tian *et al.*, 2014] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299, 2014.

[Wang *et al.*, 2016] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016.

[Xie *et al.*, 2016] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, pages 2659–2665, 2016.

[Yang *et al.*, 2016] Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. Modularity based community detection with deep learning. In *IJCAI*, pages 2252–2258, 2016.

[Zachary, 1977] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.