VECODER - VARIATIONAL EMBEDDINGS FOR COMMUNITY DETECTION AND NODE REPRESENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we study how to simultaneously learn two highly correlated tasks of graph analysis, i.e., community detection and node representation learning. We propose an efficient generative model called **VECODER** for jointly learning **V**ariational **E**mbeddings for **Co**mmunity **De**tection and node **R**epresentation. VECODER assumes that every node can be a member of one or more communities. The node embeddings are learned in such a way that connected nodes are not only "closer" to each other but also share similar community assignments. A joint learning framework leverages community-aware node embeddings for better community detection. We demonstrate on several graph datasets that VECODER effectively outperforms many competitive baselines on all three tasks i.e. node classification, overlapping community detection and non-overlapping community detection. We also show that VECODER is computationally efficient and has quite robust performance with varying hyperparameters.

1 INTRODUCTION

Graphs are flexible data structures that model complex relationships among entities, i.e. data points as nodes and the relations between nodes via edges. One important task in graph analysis is community detection, where the objective is to cluster nodes into multiple groups (communities). Each community is a set of densely connected nodes. The communities can be overlapping or non-overlapping, depending on whether they share some nodes or not. Several algorithmic (Ahn et al., 2010; Derényi et al., 2005) and probabilistic approaches (Gopalan & Blei, 2013; Leskovec & Mcauley, 2012; Wang et al., 2017; Yang et al., 2013) to community detection have been proposed. Another fundamental task in graph analysis is learning the node embeddings. These embeddings can then be used for downstream tasks like link prediction (Gao et al., 2011; Wang et al., 2017), graph visualization (Tang et al., 2016; Wang et al., 2016) and classification (Cao et al., 2015; Tang et al., 2015).

In the literature, these tasks are treated separately. Recently two approaches, namely ComE (Cavallari et al., 2017) and vGraph (Sun et al., 2019), have been proposed to learn node embeddings and detect communities simultaneously in a unified framework. Several studies have shown that community detection is improved by incorporating the node representation in the learning process (Cao et al., 2015; Kozdoba & Mannor, 2015; Tian et al., 2014). The intuition is that the global structure of graphs learned during community detection can provide useful context for node embeddings and vice versa. Fig. 1 presents an overview of the current state of research for joint learning of these two tasks.

Both existing approaches (ComE and vGraph) learn two embeddings for each node. One node embedding is used for the node representation task. The second node embedding is the "context" embedding of the node which aids in community detection. As ComE is based on DeepWalk (Perozzi et al., 2014), it inherits "context" embedding from it for learning the neighbourhood information of the node. vGraph requires two node embeddings for parametrizing two different distributions. In contrast, we propose learning a single community-aware node representation which is directly used for both tasks. In this way, we not only get rid of an extraneous node embedding but also reduce the computational cost.

In this paper, we propose an efficient generative model called **VECODER** for jointly learning both community detection and node representation. The underlying intuition behind VECODER is that

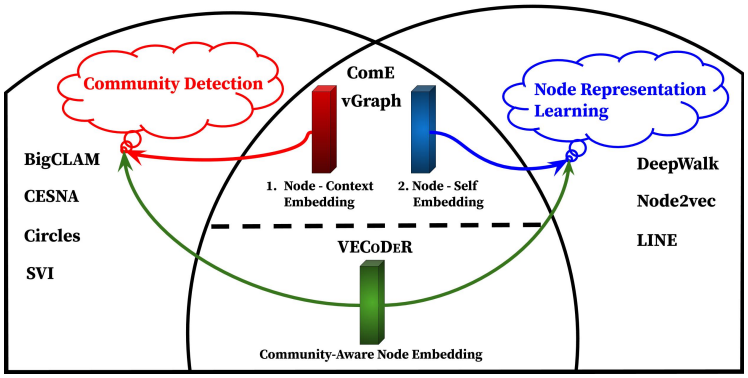


Figure 1: VECODER and competitive baselines for community detection and node representation

every node can be a member of one or more communities. However, the node embeddings should be learned in such a way that connected nodes are "closer" to each other than unconnected nodes. Moreover, connected nodes should have similar community assignments. Formally, we assume that for i-th node, the node embeddings $\boldsymbol{z}_i$ are generated from a prior distribution $p(\boldsymbol{z})$. Given $\boldsymbol{z}_i$, the community assignments c_i are sampled from $p(c_i|\boldsymbol{z}_i)$, which is parameterized by node and community embeddings. In order to generate an edge (i, j), we sample another node embedding $\boldsymbol{z}_j$ from $p(\boldsymbol{z})$ and respective community assignment c_j from $p(c_j|\boldsymbol{z}_j)$. Afterwards the node embeddings and the respective community assignments of node pairs are fed to a decoder. The decoder ensures that embeddings of both the nodes and the communities of connected nodes share high similarity with each other. This enables learning such node embeddings that are useful for both community detection and node representation tasks (see Fig. 1).

We validate the effectiveness of our approach on several real-world graph datasets. In Sec. 4, we show empirically that VECODER is able to outperform the baseline methods including the direct competitors (ComE and vGraph) on all three tasks i.e. node classification, overlapping community detection and non-overlapping community detection. Furthermore, we compare the computational cost of training different algorithms. VECODER is up to 40x more time-efficient than its competitors. We also conduct hyperparameter sensitivity analysis which demonstrates the robustness of our approach. For qualitative analysis, we also present graph visualizations generated via VECODER . Our main contributions are summarized below:

- We propose an efficient generative model called **VECODER** for joint community detection and node representation learning.
- We adopt a novel approach and argue that a single node embedding is sufficient for learning both the representation of the node itself and its context.
- VECODER significantly outperforms the state-of-the-art methods (ComE and vGraph) on all metrics of interest, for joint community detection and node representation learning.
- Training VECODER is extremely time-efficient in comparison to ComE and vGraph.

2 RELATED WORK

Community Detection. Early community detection algorithms are inspired from clustering algorithms (Xie et al., 2013). For instance, spectral clustering (Tang & Liu, 2011) is applied to the graph adjacency matrix for extracting the communities. Similarly, several matrix factorization based methods have been proposed to tackle the community detection problem. For example, BIGCLAM (Yang & Leskovec (2013)) treats the problem as a non-negative matrix factorization (NMF) task. It aims to recover the node-community affiliation matrix and learns the latent factors which represent community affiliations of nodes. Another method CESNA(Yang et al. (2013)) extends BIGCLAM by modelling the interaction between the network structure and the node attributes. The performance of matrix factorization methods is limited due to the capacity of the bi-linear models. Some generative models, like vGraph (Sun et al., 2019), Circles (Leskovec & Mcauley (2012)) etc, have also

been proposed to detect communities in a graph. But these methods suffer from high computational complexity. We present a comparison of training time with VECODER in the experiments.

Node Representation Learning. Many successful algorithms which learn node representation in an unsupervised way are based on random walk objectives (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016; Hamilton et al., 2017) . However, Gilmer et al. (2017) recently showed that graph convolutions encoder models greatly reduce the need for using the random-walk based training objectives. This is because the graph convolutions enforce that the neighboring nodes have similar representations. Some known issues with random-walk based methods (e.g. DeepWalk, ComE etc) are: (1) They sacrifice the structural information of the graph by putting over-emphasis on the proximity information (Ribeiro et al., 2017) and (2) great dependence of the performance on hyperparameters (walk-length, number of hops etc) (Perozzi et al., 2014; Grover & Leskovec, 2016). ComE(Cavallari et al. (2017)) is inspired from DeepWalk (a random-walk based method) and jointly learns community embeddings as well as node embeddings. There are some methods which we do not include in this paper because of their high computational complexity. For instance, GraRep (Cao et al., 2015) has complexity cubic to the number of nodes.

Joint community detection and node representation learning. In the literature, several attempts have been made to tackle both these tasks in a single framework (Cavallari et al., 2017; Jia et al., 2019; Tsitsulin et al., 2018; Tu et al., 2018; Wang et al., 2017). Most of these methods propose an alternate optimization process, i.e. learn node embeddings and improve community assignments with them and vice versa (Cavallari et al., 2017; Tu et al., 2018). Some methods have high computational complexity , i.e. quadratic to the number of nodes in a graph, e.g. M-NMF (Wang et al. (2017)) and DNR (Yang et al., 2016a). Two existing approaches (ComE and vGraph) appear to be less prone to aforementioned problems. However, their method requires learning two embeddings for each node for simultaneously tackling the two tasks. Unlike them, VECODER learns a single community-aware node representation which is directly used for both tasks.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

Suppose a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ and a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ of F-dimensional node features, N being the number of nodes. Given K as the number of communities, we aim to jointly learn the node embeddings and the community embeddings following a variational approach such that: (1) One or more communities can be assigned to every node and (2) the node embeddings can be used for both community detection and node classification.

3.2 VARIATIONAL MODEL

Generative Model: Let us denote the latent node embedding and community assignment for i-th node by the random variables $\boldsymbol{z}_i \in \mathbb{R}^d$ and c_i respectively. The generative model is given by:

$$p(\boldsymbol{A}) = \int \sum_{\boldsymbol{c}} p(\boldsymbol{Z}, \boldsymbol{c}, \boldsymbol{A}) d\boldsymbol{Z}, \tag{1}$$

where $\boldsymbol{c} = [c_1, c_2, \cdots, c_N]$ and the matrix $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_N]$ stacks the node embeddings. The joint distribution in (1) is mathematically expressed as

$$p(\boldsymbol{Z}, \boldsymbol{c}, \boldsymbol{A}) = p(\boldsymbol{Z})\, p_\theta(\boldsymbol{c}|\boldsymbol{Z})\, p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}), \tag{2}$$

where θ denotes the model parameters. Let us denote elements of $\boldsymbol{A}$ by a_{ij}. Considering $\boldsymbol{z}_i$ and $c_i|\boldsymbol{z}_i$ to be $i.i.d$ random variables, the joint distributions in (2) can be factorized as

$$p(\boldsymbol{Z}) = \prod_{i=1}^{N} p(\boldsymbol{z}_i) \tag{3} \qquad p_\theta(\boldsymbol{c}|\boldsymbol{Z}) = \prod_{i=1}^{N} p_\theta(c_i|\boldsymbol{z}_i) \tag{4}$$

$$p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}) = \prod_{i,j} p_\theta(a_{ij}|c_i, c_j, \boldsymbol{z}_i, \boldsymbol{z}_j), \tag{5}$$

where Eq. (5) assumes that the *edge decoder* $p_\theta(a_{ij}|c_i, c_j, \boldsymbol{z}_i, \boldsymbol{z}_j)$ depends only on $c_i, c_j, \boldsymbol{z}_i$ and $\boldsymbol{z}_j$.

Inference Model: We aim to learn the model parameters θ such that $\log(p_\theta(\boldsymbol{A}))$ is maximized. In order to ensure computational tractability, we introduce the approximate posterior

$$q_\phi(\boldsymbol{Z}, \boldsymbol{c}|\mathcal{I}) = \prod_i q_\phi(\boldsymbol{z}_i, c_i|\mathcal{I}) = \prod_i q_\phi(\boldsymbol{z}_i|\mathcal{I}) q_\phi(c_i|\boldsymbol{z}_i, \mathcal{I}), \tag{6}$$

where $\mathcal{I} = (\boldsymbol{A}, \boldsymbol{X})$ if node features are available, otherwise $\mathcal{I} = \boldsymbol{A}$. We maximize the corresponding ELBO bound (for derivation, refer to the supplementary material), given by

$$\mathcal{L}_{ELBO} \approx -\sum_{i=1}^{N} D_{KL}(q_\phi(\boldsymbol{z}_i|\mathcal{I}) \,||\, p(\boldsymbol{z}_i)) - \sum_{i=1}^{N} \frac{1}{M} \sum_{m=1}^{M} D_{KL}(q_\phi(c_i|\boldsymbol{z}_i^{(m)}, \mathcal{I}) \,||\, p_\theta(c_i|\boldsymbol{z}_i^{(m)}))$$
$$+ \sum_{i,j} \mathbb{E}_{(\boldsymbol{z}_i, \boldsymbol{z}_j, c_i, c_j) \sim q_\phi(\boldsymbol{z}_i, \boldsymbol{z}_j, c_i, c_j|\mathcal{I})} \Big\{ \log\Big(p_\theta(a_{ij}|c_i, c_j, \boldsymbol{z}_i, \boldsymbol{z}_j) \Big) \Big\}, \tag{7}$$

where $D_{KL}(.||.)$ represents the KL-divergence between two distributions. The distribution $q_\phi(\boldsymbol{z}_i, \boldsymbol{z}_j, c_i, c_j|\mathcal{I})$ in the third term of Eq. (7) is factorized into two conditionally independent distributions i.e.

$$q_\phi(\boldsymbol{z}_i, \boldsymbol{z}_j, c_i, c_j|\mathcal{I}) = q_\phi(\boldsymbol{z}_i, c_i|\mathcal{I}) q_\phi(\boldsymbol{z}_j, c_j|\mathcal{I}). \tag{8}$$

3.3 Design Choices

In Eq. (3), $p(\boldsymbol{z}_i)$ is chosen to be the standard gaussian distribution for all i. The corresponding approximate posterior $q_\phi(\boldsymbol{z}_i|\mathcal{I})$ in Eq. (6), used as node embeddings encoder, is given by

$$q_\phi(\boldsymbol{z}_i|\mathcal{I}) = \mathcal{N}\big(\boldsymbol{\mu}_i(\mathcal{I}), \text{diag}(\boldsymbol{\sigma}^2{}_i(\mathcal{I}))\big). \tag{9}$$

The parameters of $q_\phi(\boldsymbol{z}_i|\mathcal{I})$ can be learnt by any encoder network e.g. graph convolutional network (Kipf & Welling (2016a)), graph attention network (Veličković et al. (2017)), GraphSAGE (Hamilton et al. (2017)) or even two matrices to learn $\boldsymbol{\mu}_i(\mathcal{I})$ and $\text{diag}(\boldsymbol{\sigma}^2{}_i(\mathcal{I}))$. Samples are then generated using reparametrization trick (Doersch (2016)).

For parameterizing $p_\theta(c_i|\boldsymbol{z}_i)$ in Eq. (4), we introduce community embeddings $\{\boldsymbol{g}_1, \cdots, \boldsymbol{g}_K\}$; $\boldsymbol{g}_k \in \mathbb{R}^d$. The distribution $p_\theta(c_i|\boldsymbol{z}_i)$ is then modelled as the softmax of dot products of $\boldsymbol{z}_i$ with $\boldsymbol{g}_k$, i.e.

$$p_\theta(c_i = k|\boldsymbol{z}_i) = \frac{\exp(< \boldsymbol{z}_i, \boldsymbol{g}_k >)}{\sum_{\ell=1}^{K} \exp(< \boldsymbol{z}_i, \boldsymbol{g}_\ell >)}. \tag{10}$$

The corresponding approximate posterior $q_\phi(c_i = k|\boldsymbol{z}_i, \mathcal{I})$ in Eq. (6) is affected by the node embedding $\boldsymbol{z}_i$ as well as the neighborhood. To design this, our intuition is to consider the similarity of $\boldsymbol{g}_k$ with the embedding $\boldsymbol{z}_i$ as well as with the embeddings of the neighbors of the i-th node. The overall similarity with neighbors is mathematically formulated as the average of the dot products of their embeddings. Afterwards a hyperparameter α is introduced to control the bias between the effect of $\boldsymbol{z}_i$ and the set $\mathcal{N}_i$ of the neighbors of the i-th node. Finally, a softmax is applied as follows

$$q_\phi(c_i = k|\boldsymbol{z}_i, \mathcal{G}) = \frac{\exp\Big(\alpha < \boldsymbol{z}_i, \boldsymbol{g}_k > + (1-\alpha) \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} < \boldsymbol{z}_j, \boldsymbol{g}_k > \Big)}{\sum_{\ell=1}^{K} \exp\Big(\alpha < \boldsymbol{z}_i, \boldsymbol{g}_\ell > + (1-\alpha) \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} < \boldsymbol{z}_j, \boldsymbol{g}_\ell > \Big)}. \tag{11}$$

Hence, Eq. (11) ensures that graph structure information is employed to learn community assignments instead of relying on an extraneous node embedding as done in (Sun et al., 2019; Cavallari et al., 2017). Finally, the choice of edge decoder in Eq. (5) is motivated by the intuition that the nodes connected by edges have a high probability of belonging to the same community and vice versa. Therefore we model the edge decoder as:

$$p_\theta(e_{ij}|c_i = \ell, c_j = m, \boldsymbol{z}_i, \boldsymbol{z}_j) = \frac{\sigma(< \boldsymbol{z}_i, \boldsymbol{g}_m >) + \sigma(< \boldsymbol{z}_j, \boldsymbol{g}_\ell >)}{2}. \tag{12}$$

For better reconstructing the edges, Eq. (12) makes use of the community embeddings, node embeddings and community assignment information simultaneously. This helps in learning better node representations by leveraging the global information about the graph structure via community detection. On the other hand, this also forces the community assignment information to exploit the local graph structure via node embeddings and edge information.

3.4 Practical Aspects

The third term in Eq. (7) is estimated in practice using the samples generated by the approximate posterior. This term is equivalent to the negative of binary cross-entropy (BCE) loss between observed edges and reconstructed edges. Since community assignment follows a categorical distribution, we use Gumbel-softmax (Jang et al. (2016)) for backpropagation of the gradients. As for the second term of Eq. (7), it is also enough to set $M = 1$, i.e. use only one sample per input node.

For inference, non-overlapping community assignment can be obtained for i-th node as

$$\mathcal{C}_i = \underset{k \in \{1, \cdots, K\}}{\arg\max} \; q_\phi(c_i = k | \boldsymbol{z}_i, \mathcal{I}). \tag{13}$$

To get overlapping community assignments for i-th node, we can threshold its weighted probability vector at ϵ, a hyperparameter, as follows

$$\mathcal{C}_i = \left\{ k \;\middle|\; \frac{q_\phi(c_i = k | \boldsymbol{z}_i, \mathcal{I})}{\max_\ell q_\phi(c_i = \ell | \boldsymbol{z}_i, \mathcal{I})} \geq \epsilon \right\}, \quad \epsilon \in [0, 1]. \tag{14}$$

3.5 Complexity

Computation of dot products for all combinations of node and community embeddings takes $\mathcal{O}(NKd)$ time. Solving Eq. (11) further requires calculation of mean of dot products over the neighborhood for every node, which takes $\mathcal{O}(|\mathcal{E}|K)$ computations overall as we traverse every edge for every community. Finally, we need softmax over all communities for every node in Eq. (10) and Eq. (11) which takes $\mathcal{O}(NK)$ time. Eq. (12) takes $\mathcal{O}(|\mathcal{E}|)$ time for all edges as we have already calculated the dot products. As a result, the overall complexity becomes $\mathcal{O}(|\mathcal{E}|K + NKd)$. This complexity is quite low compared to other algorithms designed to achieve similar goals (Cavallari et al., 2017; Wang et al., 2017; Yang et al., 2016a).

4 Experiments

4.1 Datasets

We have selected 15 different datasets ranging from 270 to 126,842 edges. For non-overlapping community detection and node classification, we use 5 the citation datasets (Bojchevski & Günnemann (2017); Yang et al. (2016b)). The remaining datasets (Leskovec & Mcauley (2012); Yang & Leskovec (2015)), used for overlapping community detection, are taken from SNAP repository (Leskovec & Krevl (2014)). Following (Sun et al., 2019), we take 5 biggest ground truth communities for youtube, amazon and dblp.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	K	$\lvert F\rvert$	Overlap
CiteSeer	3327	9104	6	3703	N
CiteSeer-full	4230	10674	6	602	N
Cora	2708	10556	7	1433	N
Cora-ML	2995	16316	7	2879	N
Cora-full	19793	126842	70	8710	N
fb0	333	2519	24	N/A	Y
fb107	1034	26749	9	N/A	Y
fb1912	747	30025	46	N/A	Y
fb3437	534	4813	32	N/A	Y
fb348	224	3192	14	N/A	Y
fb414	150	1693	7	N/A	Y
fb698	61	270	13	N/A	Y
youtube	5346	24121	5	N/A	Y
amazon	794	2109	5	N/A	Y
dblp	24493	89063	5	N/A	Y

Table 1: Every dataset has $|\mathcal{V}|$ nodes, $|\mathcal{E}|$ edges, K communities and $|F|$ features. $|F|$ = N/A means that either the features were missing or not used.

4.2 Baselines

For overlapping community detection, we compare with the following competitive baselines: **BIGCLAM** (Yang & Leskovec (2013)) formulates community detection as a non-negative matrix factorization (NMF) task. **CESNA** (Yang et al. (2013)) extends BIGCLAM by statistically modelling

F1 Score (%) / jaccard (%)									
Dataset	**Bigclam**	**CESNA**	**Circles**	**SVI**	**vGraph**	**vGraph+**	**ComE**	**VECODER**	**Δ(%)**
fb0	29.5 / 18.5	28.1 / 17.3	28.6 / 18.6	28.1 / 17.6	24.4 / 14.6	26.1 / 15.9	31.1 / 19.5	**34.7 / 24.7**	11.65 / 26.69
fb107	39.3 / 27.5	37.3 / 27.0	24.7 / 15.5	26.9 / 17.2	28.2 / 18.3	31.8 / 21.7	39.7 / 28.7	**59.7 / 46.8**	50.45 / 63.22
fb1684	50.4 / 38.0	51.2 / 38.7	28.9 / 18.7	35.9 / 24.7	42.3 / 29.2	43.8 / 32.7	52.9 / 40.3	**56.4 / 42.5**	06.63 / 05.48
fb1912	34.9 / 24.1	34.7 / 23.9	26.2 / 16.7	28.0 / 20.1	25.8 / 18.6	37.5 / 28.0	28.7 / 18.5	**45.8 / 37.3**	22.16 / 33.37
fb3437	19.9 / 11.5	20.1 / 11.7	10.1 / 05.5	15.4 / 09.0	20.9 / 12.0	22.7 / 13.3	21.3 / 12.5	**50.2 / 36.2**	121.57 / 172.67
fb348	49.6 / 35.9	53.8 / 40.0	51.8 / 39.3	46.1 / 33.6	55.4 / 41.0	53.1 / 40.5	46.2 / 34.4	**58.2 / 43.5**	04.98 / 06.07
fb414	58.9 / 47.1	60.1 / 47.3	48.4 / 34.2	38.9 / 29.3	64.7 / 51.8	66.9 / 55.9	55.3 / 42.2	**69.6 / 58.4**	04.05 / 04.56
fb698	54.2 / 41.9	58.7 / 45.9	35.2 / 22.6	40.3 / 30.0	54.0 / 43.6	59.5 / 47.7	45.8 / 33.8	**64.0 / 50.4**	07.61 / 05.66
Youtube	43.7 / 29.3	38.4 / 24.2	36.0 / 22.1	41.4 / 28.7	50.7 / 34.3	52.2 / 34.8	65.5 / 52.5	**67.3 / 53.3**	02.72 / 01.60
Amazon	46.4 / 35.1	46.8 / 35.0	53.3 / 36.7	47.3 / 36.4	53.3 / 36.9	53.2 / 36.9	50.1 / 34.6	**58.1 / 41.9**	09.08 / 13.35
Dblp	23.6 / 13.8	35.9 / 22.3	36.2 / 23.3	33.7 / 20.9	39.3 / 25.0	39.9 / 25.1	47.1 / 27.9	**53.9 / 37.3**	14.44 / 33.64

Table 2: Performance on datasets with overlapping communities. Best result for every dataset is in bold. The column $\mathbf{\Delta}(\%)$ shows the percentages by which VECODER outperforms the second best.

the interaction between the network structure and the node attributes. **Circles** (Leskovec & Mcauley (2012)) introduces a generative model for community detection in ego-networks by learning node similarity metrics for every community. **SVI** (Gopalan & Blei (2013)) formulates membership of nodes in multiple communities by a Bayesian model of networks. **vGraph** (Sun et al. (2019)) simultaneously learns node embeddings and community assignments by modelling the nodes as being generated from a mixture of communities. **vGraph+**, a variant further incorporates regularization to weigh local connectivity. **ComE** (Cavallari et al. (2017)) jointly learns community and node embeddings by using gaussian mixture model formulation.

For non-overlapping community detection and node classification, in addition to vGraph and ComE, we compare VECODER with the following baselines: **DeepWalk** (Perozzi et al. (2014)) makes use of SkipGram (Mikolov et al. (2013)) and truncated random walks on network to learn node embeddings. **LINE** (Tang et al. (2015)) learns node embeddings while attempting to preserve first and second order proximities of nodes. **Node2Vec** (Grover & Leskovec (2016)) learns the embeddings using biased random walk while aiming to preserve network neighborhoods of nodes.

4.3 SETTINGS

For overlapping community detection, we learn mean and log-variance matrices of 16-dimensional node embeddings. We set $\alpha = 0.9$ and $\epsilon = 0.3$ in all our experiments. Following Kipf & Welling (2016b), we first pre-train a variational graph autoencoder. We perform gradient descent with Adam optimizer (Kingma & Ba (2014)) and learning rate $= 0.01$. Community assignments are obtained using Eq. (14). For the baselines, we employ the results reported by Sun et al. (2019). For evaluating the performance, we use *F1-score* and *Jaccard similarity*.

For non-overlapping community detection, since the default implementations of all the baselines use 128 dimensional embeddings, for VECODER we also use $d = 128$ for fair comparison. Eq. (13) is used for community assignments. For vGraph, we use the code provided by the authors. We employ *normalized mutual information (NMI)* and *adjusted random index (ARI)* as evaluation metrics.

For node classification, we follow the split strategy used in various previous works (Yang et al., 2016b; Kipf & Welling, 2016a; Veličković et al., 2018), i.e. 20 nodes per class for training. We train logistic regression using LIBLINEAR (Fan et al. (2008)) solver as our classifier. For the algorithms that do not use node features, we train the classifier by appending the raw node features with the learnt embeddings. For evaluation, we use *F1-macro* and *F1-micro* scores.

All the reported results are the average over five runs. Further implementation details can be found in the code: https://anonymous.4open.science/r/1d95bf8f-8ce3-4870-a454-07db463b419f.

4.4 RESULTS

Tab. 2 gives the F1-score and Jaccard similarity for the datasets with overlapping communities. It can be seen that VECODER outperforms all the competitive baselines, often by considerable margins. This shows the capability of VECODER to learn multiple community assignments and hence reinforces the intuition behind the design of Eq. (11). For the datasets with non-overlapping communities, VECODER significantly outperforms other algorithms in community detection as well as node classification. It gives the best NMI results as shown in Tab. 3. As for ARI, VECODER

NMI(%) / ARI(%)							
Dataset	**ComE**	**DeepWalk**	**LINE**	**Node2Vec**	**vGraph**	**VECODER**	**Δ(%)**
CiteSeer	18.8 / 19.3	08.8 / 10.7	08.7 / 04.0	14.9 / 10.6	09.0 / 04.3	**38.5 / 35.2**	105.11 / 82.57
CiteSeer-full	32.8 / 30.2	15.4 / 22.0	13.0 / 09.3	22.3 / 20.3	07.6 / 02.6	**59.0 / 60.3**	79.88 / 99.74
Cora	39.6 / 42.2	39.7 / 36.6	32.8 / 22.1	39.7 / 34.4	26.4 / 11.5	**52.7 / 45.1**	32.65 / 06.90
Cora-ML	47.6 / **56.8**	43.2 / 45.8	42.3 / 38.5	39.6 / 49.6	29.8 / 27.2	**56.3** / 49.8	18.38 / -12.26
Cora-full	51.2 / **29.0**	48.5 / 24.4	40.3 / 13.8	48.1 / 22.4	41.7 / 14.6	**55.2** / 28.8	07.83 / -00.83

Table 3: Non-overlapping community detection results. Best result for every dataset is in bold. The column **Δ**(%) shows the performance of VECODER relative to the best among the baselines.

F1-macro(%) / F1-micro(%)							
Dataset	**ComE**	**DeepWalk**	**LINE**	**Node2Vec**	**vGraph**	**VECODER**	**Δ(%)**
CiteSeer	47.8 / 48.8	37.7 / 39.4	29.2 / 30.2	38.2 / 39.3	26.1 / 27.2	**64.8 / 68.2**	35.54 / 39.67
CiteSeer-full	58.6 / 58.9	50.4 / 51.5	39.8 / 40.2	50.2 / 51.2	25.5 / 25.6	**76.8 / 77.0**	31.01 / 30.82
Cora	69.4 / 69.9	66.2 / 66.4	53.7 / 53.3	65.3 / 64.9	38.3 / 37.8	**73.1 / 75.6**	05.35 / 08.09
Cora-ML	77.4 / 78.5	72.6 / 72.8	67.3 / 67.6	74.1 / 74.6	52.2 / 54.5	**80.2 / 82.0**	03.59 / 04.47
Cora-full	41.9 / 48.1	39.2 / 43.3	31.0 / 31.8	38.0 / 41.5	28.0 / 30.8	**43.1 / 49.6**	02.86 / 03.14

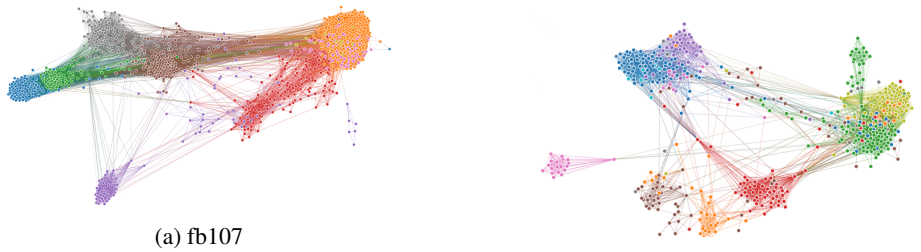
Table 4: Node classification results. Best result for every dataset is in bold. The column **Δ**(%) shows the percentage by which VECODER outperforms the second best.

performs the best in 3 out of 5 datasets. It is second best in case of *Cora-ML* and achieves similar performance to ComE on *Cora-full* dataset. On node classification, VECODER consistently achieves the best scores as shown by the **Δ** column in Tab. 4. This supports our claim that the embeddings learnt by VECODER can simultaneously be used for community detection as well as node classification because of the implicit relatedness in these tasks.

These results also show that a single community-aware node embedding is sufficient to aid in both the node representation and community assignment tasks. This is also qualitatively demonstrated by graph visualizations of node embeddings (obtained via t-SNE (Maaten & Hinton, 2008)) and inferred communities for two datasets, fb107 and fb3437, presented in Fig. 2.

4.5 HYPERPARAMETER SENSITIVITY

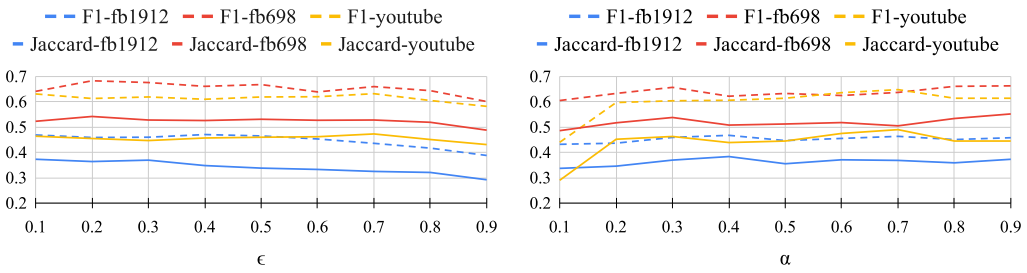
We study the dependence of VECODER on ϵ and α by evaluating on three datasets of different sizes: fb698($N = 61$), fb1912($N = 747$) and youtube($N = 5346$). We sweep for $\epsilon = \{0.1, 0.2, \cdots, 0.9\}$. For demonstrating effect of α, we fix $\epsilon = 0.3$ and sweep for $\alpha = \{0.1, 0.2, \cdots, 0.9\}$. The average results of five runs for ϵ and α are given in Fig. 3a and Fig. 3b respectively. Overall VECODER is quite robust to the change in the values of ϵ and α. In case of ϵ, we see a general trend of decrease in performance when the threshold ϵ is set quite high e.g. $\epsilon > 0.7$. This is because the datasets contain overlapping communities and a very high ϵ will cause the algorithm to give only the most probable community assignment instead of potentially providing multiple communities per node. However, for a large part of sweep space, the results are almost consistent. When ϵ is fixed and α is changed, the results are mostly consistent except when α is set to a very low value e.g. $\alpha < 0.2$.



(a) fb107

(b) fb3437

Figure 2: Graph visualization with community assignments (better viewed in color)



(a) Effect of ϵ. Overall a slight decrease in scores can be observed after $\epsilon = 0.7$ mark.

(b) Effect of α. The scores generally tend to decrease for very small values of α.

Figure 3: Effect of hyperparameters on the performance. F1 and Jaccard scores are in solid and dashed lines respectively.

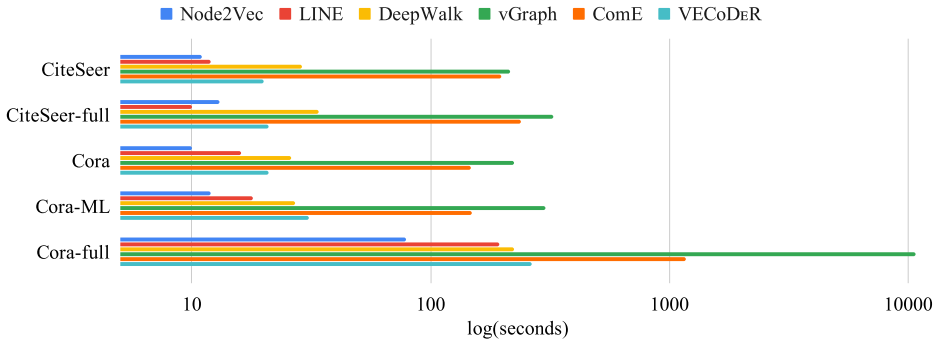


Figure 4: Comparison of running times of different algorithms.

Eq. (11) shows that in such a case the node itself is almost neglected and VECODER tends to assign communities based upon neighborhood only, which causes a slight decrease in the performance.

4.6 TRAINING TIME

Now we compare the training times of different algorithms in Fig. 4. As some of the baselines are more resource intensive than others, we select aws instance type `g4dn.4xlarge` for fair comparison of training times. For vGraph, we train for 1000 iterations and for VECODER for 1500 iterations. For all other algorithms we use the default parameters as used in section 4.3. We observe that the methods that simply output the node embeddings take relatively less time compared to the algorithms that jointly learn node representations and community assignments i.e. VECODER , ComE and vGraph. Among these algorithms VECODER is the most time efficient. It consistently trains in less time compared to ComE and vGraph. For instance, it is about 12 times faster than ComE for *CiteSeer-full* and about 40 times faster compared to vGraph for *Cora-full* dataset. This provides evidence for lower computational complexity of VECODER in Section 3.5.

5 CONCLUSION

We propose a scalable generative method VECODER to simultaneously perform community detection and node representation learning. Our novel approach learns a single community-aware node embedding for both the representation of the node and its context. VECODER is scalable due to its low complexity, i.e. $\mathcal{O}(|\mathcal{E}|K + NKd)$. The experiments on several graph datasets show that VECODER consistently outperforms all the competitive baselines on node classification, overlapping community detection and non-overlapping community detection tasks. Moreover, training the VECODER is highly time-efficient than its direct competitors (ComE and vGraph).

REFERENCES

Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761–764, 2010.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pp. 891–900, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450337946. doi: 10.1145/2806416.2806512. URL https://doi.org/10.1145/2806416.2806512.

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 *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 377–386, 2017.

Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005.

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

Sheng Gao, Ludovic Denoyer, and Patrick Gallinari. Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1169–1174, 2011.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Prem K Gopalan and David M Blei. Efficient discovery of overlapping communities in massive networks. *Proceedings of the National Academy of Sciences*, 110(36):14534–14539, 2013.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Yuting Jia, Qinqin Zhang, Weinan Zhang, and Xinbing Wang. Communitygan: Community detection with generative adversarial nets. In *The World Wide Web Conference*, pp. 784–794, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Mark Kozdoba and Shie Mannor. Community detection via measure space embedding. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 2890–2898, Cambridge, MA, USA, 2015. MIT Press.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pp. 539–547, 2012.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 385–394, 2017.

Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. vgraph: A generative model for joint community detection and node representation learning. In *Advances in Neural Information Processing Systems*, pp. 514–524, 2019.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015.

Jiliang Tang, Charu Aggarwal, and Huan Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM international conference on data mining*, pp. 54–62. SIAM, 2016.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pp. 1293–1299. AAAI Press, 2014.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*, pp. 539–548, 2018.

Cunchao Tu, Xiangkai Zeng, Hao Wang, Zhengyan Zhang, Zhiyuan Liu, Maosong Sun, Bo Zhang, and Leyu Lin. A unified framework for community detection and network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(6):1051–1065, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1225–1234, 2016.

Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, volume 17, pp. 203–209, 2017.

Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)*, 45(4):1–35, 2013.

Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 587–596, 2013.

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

Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pp. 1151–1156. IEEE, 2013.

Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. Modularity based community detection with deep learning. In *IJCAI*, volume 16, pp. 2252–2258, 2016a.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016b.