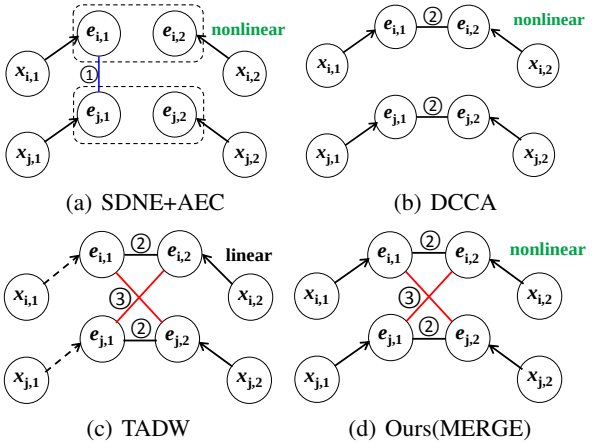
Unsupervised Multi-view Nonlinear Graph Embedding

Jiaming Huang[1,2] , Zhao Li[1]*, Vincent W. Zheng[3] , Wen Wen[2] , Yifan Yang[1] , Yuanmi Chen[1]
[1]Alibaba Group, Hangzhou, China
[2]School of Computers, Guangdong University of Technology, Guangzhou, China
[3]Advanced Digital Sciences Center, Singapore

Abstract

In this paper, we study the unsupervised multi-view graph embedding (UMGE) problem, which aims to learn graph embedding from multiple perspectives in an unsupervised manner. However, the vast majority of multi-view learning work focuses on non-graph data, and surprisingly there are limited work on UMGE. By systematically analyzing different existing methods for UMGE, we discover that cross-view and nonlinearity play a vital role in efficiently improving graph embedding quality. Motivated by this concept, we develop an unsupervised Multi-viEw nonlineaR Graph Embedding (MERGE) approach to model relational multi-view consistency. Experimental results on five benchmark datasets demonstrate that MERGE significantly outperforms the state-of-the-art baselines in terms of accuracy in node classification tasks without sacrificing the computational efficiency.



(a) SDNE+AEC (b) DCCA (c) TADW (d) Ours(MERGE)

Figure 1: A systematic comparison of different methods. $\mathbf{x}_{i,j}$ denotes the i^{th} instance (or node)'s j^{th}-view features, and $\mathbf{e}_{i,j}$ is its embedding. In (c) $\sim$ (d), we let view 1's features be network, and view 2's be content. ① denotes the cross-instance in single view. ② denotes the intra-instance–cross-view. ③ denotes the cross-instance–cross-view.

1 INTRODUCTION

Most of the recent work on graph embedding [25, 9, 5] focuses on the network information alone. In practice, a graph may consist of additional node features; *e.g.*, in a paper citation network, each paper node includes text content. Thus some pioneer work has started to consider the node features in graph embedding; *e.g.*, Planetoid [33] uses a semi-supervised framework to learn a network embedding for each node from its network features and a content embedding from its content features. Generally, network structure and node features are considered as two different "views" for a node in the graph. Motivated by *multi-view learning* [31, 11], we allow the two views to reinforce each other, so as to obtain a better graph embedding.

In this paper, we try to solve the *Unsupervised Multi-view Graph Embedding* (UMGE) problem. Given a graph with node features, we aim to learn a network embedding and a content embedding simultaneously for each node in an unsupervised manner. Most of the multi-view learning work focuses on non-graph data, such as image and caption features in [18], acoustic and articulatory features in speech [2], and so on. Combining network and node features has been popular for graph classification [17] and graph clustering [36]. However, it has been under-explored in the literature for UMGE [2, 32, 37]. Moreover, despite the success of these prior methods, their development appears ad hoc and underlying connections are not investigated yet.

*Zhao Li and Vincent W. Zheng are co-first authors.

To approach the problem of UMGE, we start with systematically analyzing the underlying connections among different prior methods in Fig. 1 (two-view example). For simple discussion, let us denote $\mathbf{x}_{i,k}$ as the features corresponding to the i^{th} instance in k^{th} view, and $\mathbf{e}_{i,k}$ as its embedding. A naive way to extend the single view graph embedding methods into multi-view scenarios is concatenating the graph embedding (learned in network structure only) and the node features directly. For example, as shown in Fig. 1(a), SDNE [27] learns the graph embedding taking no account of the node features. By using a deep AutoEncoder [4] to model the node features and concatenating these two kinds of embeddings as node representation, SDNE can be extended to multi-view scenarios. However, simply concatenating the two views has no guarantee to reinforce each other. Typically, multi-view learning focuses on enforcing intra-instance–cross-view consistency on irrelevant data. In Fig. 1(b), DCCA [2] first embeds each instance's multi-view features $\mathbf{x}_{i,k}$'s into low-dimensional representations $\mathbf{e}_{i,k}$'s, then enforces the maximal correlation between $\mathbf{e}_{i,1}$ and $\mathbf{e}_{i,2}$ for each i, which does not take the relational information into consideration. Some recent work exploits a different approach to model the relational information by enforcing cross-instance–cross-view consistency. In Fig. 1(c), TADW [32] recovers an adjacency matrix with the multiplication between a network embedding matrix and a text embedding matrix. Thus, two nodes with similar neighbors tend to have similar multi-view embedding.

Despite the recent advance of UMGE in network representation, there are still rooms for improvement, as current work has some limitations. First, the feature embedding in TADW is linear. However, in practice, data nonlinearity is common for both network features [27] and content features [30]. Second, the network structure is under-explored. For example, DCCA does not preserve any relational proximity. However, preserving network structure is useful for graph embedding [25]. TADW preserves the second-order proximity by using matrix decomposition, which is computationally expensive. It takes a complexity of $O(nnz(M) + |V|)$ to decompose the adjacency matrix M for a graph $G = (V, E)$, where V and E are the sets of nodes and edges respectively, and $nnz(M)$ is the number of non-zero entries in M. In other words, its complexity is at least $O(|E|+|V|)$, since $nnz(M) \geq |E|$. Therefore, our goal is to efficiently solve UMGE with nonlinearity. We consider nonlinear embedding for both views in order to deal with the nonlinear data nature. Besides, we also preserve the second-order proximity by enforcing cross-instance–cross-view consistency, yet with less computational cost compared against TADW.

In this paper, we propose a simple, yet effective unsupervised Multi-viEw nonlineaR Graph Embedding (MERGE) model. Our insights are two-fold. First, inspired by SDNE [27], MERGE encodes the nonlinearity of the network/content by taking the network/content features as input, and then applying a deep AutoEncoder [4] to learn a nonlinear network/content embedding for each node. Second, MERGE preserves the second-order proximity by extending DeepWalk [20] to the multi-view setting. On the one hand, DeepWalk preserves the second-order proximity by using one node's network embedding to interpret its "neighbor" node's context embedding. MERGE is easy to extend DeepWalk for using one node's network embedding to interpret its "neighbor" node's content embedding, so as to enforce cross-instance–cross-view consistency. Besides, MERGE also fully leverages this formulation to enforce intra-instance–cross-view consistency by using one node's network embedding to interpret its own content embedding. On the other hand, DeepWalk employs a graph sampling approach to achieve comparable performance, which takes an $O(|V| \log |V|)$ complexity with a hierarchical softmax objective function. MERGE further reduces the complexity to $O(|V|)$ by using *negative sampling* [19]. We back up this argument later with a detailed complexity analysis of the MERGE algorithm in Sec. 4.

Our contributions are summarize as follows. It is the first time that different existing methods for UMGE are systematically analyzed, where cross-view and non-linearity are discovered to be critical in improving graph embedding quality, and we develop a simple, yet effective MERGE model to efficiently incorporate nonlinearity in graph embedding. MERGE is evaluated on five benchmark datasets, and it exceedingly outperforms the state-of-the-art baselines by at least relatively 1.5% $\sim$ 17.9% (macro-F1) and 0.4% $\sim$ 14.9% (micro-F1) over all the datasets. Moreover, the computational complexity of MERGE scales linearly with $|V|$, which is more favorably applied to real-world scenarios.

2 RELATED WORD

Graph embedding is an important task for graph analytics. Generally, it aims to learn a vector representation for each node, such that two nodes being "close" on the graph have similar vectors. Earlier work on graph embedding, such as MDS [7], LLE [23], IsoMap [26] and Laplacian eigenmap [3], typically treats to solve the leading eigenvectors of graph affinity matrices as node embedding. Recent methods explore deep learning for graph embedding. For example, LINE [25] models the node closeness by using both first-order and second-order proximity, therefore two nodes either connect di-

rectly or share the common neighbors result in similar embedding. Node2Vec [9] considers how to sample paths for DeepWalk, with awareness of node homophily and structural role. Besides, there are a number of works study the heterogeneous graph embedding [24, 6, 16]. Most existing graph embedding work focuses on single-view setting; *i.e.*, only network structure is considered. Only little work considers a multi-view setting, such as COLDA [12], TADW [32] and GCN [13]. Yet, COLDA is supervised; GCN is semi-supervised; only TADW is unsupervised.

Multi-view learning is a machine learning paradigm, which handles the data with multiple views of features in its instances [28]. Some detailed surveys about multi-view learning are available in [31, 15]. A large portion of multi-view learning literatures focus on supervised or semi-supervised settings, such as recommendation [8] and classification [10, 35]. Some other work considers an unsupervised setting, but not particularly for representation learning. For example, in [14], multi-view learning is used to improve spectral clustering; whereas in [34], the focus is to tackle the corrupted view with intra-view and inter-view noises. An overall review of recent multi-view embedding models is in [29]. Among them, a notable method is DCCA [2]. It extends CCA [1] by learning embedding for each view with a deep neural network and then enforcing the maximal correlation across two views' embedding. However, some recent multi-view embedding work, including COLDA [12], and TADW [32], has paid less attention to graph data.

3 UNSUPERVISED MULTI-VIEW GRAPH EMBEDDING

The UMGE is formalized as follows. Given a graph $G = (V, E)$ as the input, each node $v_i \in V$ has a network feature vector $\mathbf{x}_{i,1} \in \mathbb{R}^{m_1}$ and a content feature vector $\mathbf{x}_{i,2} \in \mathbb{R}^{m_2}$. UMGE then generates two embedding vectors in a d-dimensional common space for each node as the output. For instance, the network features $\mathbf{x}_{i,1}$ is extracted from G's adjacency matrix, and the content features $\mathbf{x}_{i,2}$ is extracted from each node's text information or attributes. They are treated as two different views to represent node v_i's by generating the network embedding $\mathbf{e}_{i,1} \in \mathbb{R}^d$ and the content embedding $\mathbf{e}_{i,2} \in \mathbb{R}^d$ accordingly.

To tackle the problem, we propose MERGE, as shown in Fig. 1(d), to efficiently model the relational multi-view consistency with nonlinearity. MERGE enforces the cross-instance–cross-view and intra-instance–cross-view consistencies by DeepWalk style formulations with an inexpensive realization, which models the relational

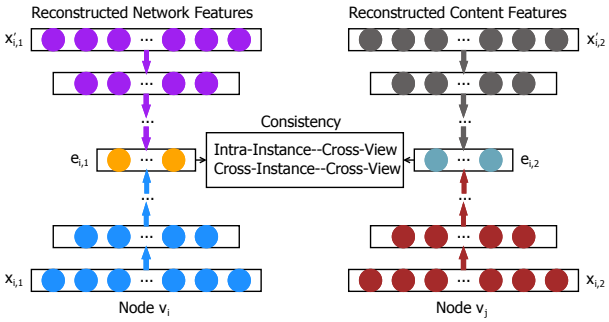


Figure 2: Illustration of the MERGE model.

information with second-order proximity and the multiple views correspondingly. Additionally, MERGE applies a deep neural network on the multi-view features and couples the nonlinear embedding outputs with the rich relational multi-view consistencies to model the feature nonlinearity. Fig. 2 summarizes the three major components of the proposed MERGE model:

- A direct formulation of intra-instance–cross-view and cross-instance–cross-view consistencies between node v_i's network embedding $\mathbf{e}_{i,1}$ and its "neighbor" node v_j's content embedding $\mathbf{e}_{j,2}$ (by setting i as a special "neighbor" node of itself, this component handles the intra-instance–cross-view as well);
- A nonlinear formulation of learning a network embedding $\mathbf{e}_{i,1}$ from node v_i's network features $\mathbf{x}_{i,1}$;
- A nonlinear formulation of learning a content embedding $\mathbf{e}_{j,2}$ from node v_j's content features $\mathbf{x}_{j,2}$.

By integrating all these components together, MERGE learns the embedding $\mathbf{e}_{i,1}$ and $\mathbf{e}_{j,2}$ from G. In the following sections, we introduce the methodology to model these consistencies.

Cross-instance–cross-view consistency for second-order proximity. Formally, we define this consistency between one node v_i's view-1 embedding $\mathbf{e}_{i,1}$ and another node v_j's view-2 embedding $\mathbf{e}_{j,2}$ as a probability $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$. To enforce this cross-instance–cross-view consistency between $\mathbf{e}_{i,1}$ and $\mathbf{e}_{j,2}$, MERGE maximizes $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$. Inspired by DeepWalk [20], which preserves second-order proximity and uses graph sampling for efficient realization, MERGE defines $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$ with a softmax function over $\mathbf{e}_{i,1}$ and $\mathbf{e}_{j,2}$:

$$p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1}) = \frac{\exp\left(\mathbf{e}_{j,2}^\top \mathbf{e}_{i,1}\right)}{\sum_{v_k \in V} \exp\left(\mathbf{e}_{k,2}^\top \mathbf{e}_{i,1}\right)}. \quad (1)$$

Following the similar intuition as DeepWalk, MERGE uses node v_i's network embedding $\mathbf{e}_{i,1}$ to interpret its "neighbor" v_j's content embedding $\mathbf{e}_{j,2}$. Different from DeepWalk, Eq. 1 does not contain any "context embedding" as additional parameters. Instead, MERGE uses one view's embedding to interpret the other view's embedding across the nodes. If two nodes have similar neighbors, the interpretations to the other view of these neighbors tend to be similar. Therefore MERGE still favors the second-order proximity with Eq. 1. Generally, it is possible to symmetrically define $p(\mathbf{e}_{i,1}|\mathbf{e}_{j,2}) \propto \exp\left(\mathbf{e}_{i,1}^\top \mathbf{e}_{j,2}\right)$. Since $\exp\left(\mathbf{e}_{i,1}^\top \mathbf{e}_{j,2}\right) = \exp\left(\mathbf{e}_{j,2}^\top \mathbf{e}_{i,1}\right)$, introducing $p(\mathbf{e}_{i,1}|\mathbf{e}_{j,2})$ tends to achieve a similar effect as Eq. 1. For simplicity, only $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$ is used to represent cross-instance–cross-view consistency in this paper.

Additionally, MERGE uses graph sampling to enforce $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$ on a size-controllable set of (v_i, v_j)'s to reduce computation cost. On the one hand, v_i and v_j are supposed to be "close" in G; otherwise it is not desirable to require their network embedding and content embedding to be consistent. We exploit a broad definition of "closeness" for v_i and v_j as [20, 9]; *i.e.*, a node v_j is the neighbor of v_i if v_j appears in a context window of v_i on a randomly sampled path from G. On the other hand, MERGE controls the number of "neighbors" of v_i by an adjustable amount of sampled paths from G. Usually, the number of sampled paths is set as a constant to avoid computing all the edges in G, which leads the complexity of TADW to be $O(|E|)$.

Intra-instance–cross-view consistency for multiple views. Formally, MERGE defines this consistency between node v_i's view-1 embedding $\mathbf{e}_{i,1}$ and its view-2 embedding $\mathbf{e}_{i,2}$ as a probability $p(\mathbf{e}_{i,2}|\mathbf{e}_{i,1})$. To enforce this intra-instance–cross-view consistency between $\mathbf{e}_{i,1}$ and $\mathbf{e}_{i,2}$, MERGE maximizes the likelihood $\prod_i p(\mathbf{e}_{i,2}|\mathbf{e}_{i,1})$. Considering the similar intuition as Eq. 1 that utilizes a given node's network embedding to interpret its content embedding, MERGE defines

$$p(\mathbf{e}_{i,2}|\mathbf{e}_{i,1}) = \frac{\exp\left(\mathbf{e}_{i,2}^\top \mathbf{e}_{i,1}\right)}{\sum_{v_k \in V} \exp\left(\mathbf{e}_{k,2}^\top \mathbf{e}_{i,1}\right)}. \tag{2}$$

In this regard, we simplify the objective function with a unified formulation about the cross-view consistency.

Nonlinearity for feature embedding. To handle the nonlinearity of each view's features, we propose to learn each v_i's embedding $\mathbf{e}_{i,k}$ as a nonlinear function of its corresponding feature vector $\mathbf{x}_{i,k}$. Specifically, we introduce AutoEncoder [4], a deep neural network, to enable this nonlinear embedding. For simplicity, we use a three-layer architecture as an example to explain how AutoEncoder works. For any input vector $\mathbf{z} \in \mathbb{R}^d$, we denote $\tanh(\mathbf{z})$ to return a vector value of the hyperbolic $\tanh$ function over each dimension of $\mathbf{z}$. For any view k, we model each $\mathbf{e}_{i,k}$ as

$$\mathbf{e}_{i,k} = \tanh(W_1^{(k)} \mathbf{x}_{i,k} + \mathbf{b}_1^{(k)}), \tag{3}$$

where $W_1^{(k)} \in \mathbb{R}^{d \times |V|}$, $\mathbf{x}_{i,k} \in \mathbb{R}^{m_k}$ and $\mathbf{b}_1^{(k)} \in \mathbb{R}^d$ are parameters. Here $\mathbf{x}_{i,k}$ is assumed to be reconstructible from $\mathbf{e}_{i,k}$ by a nonlinear function:

$$\mathbf{x}'_{i,k} = \tanh((W_2^{(k)})^\top \mathbf{e}_{i,k} + \mathbf{b}_2^{(k)}), \tag{4}$$

where $W_2^{(k)} \in \mathbb{R}^{d \times m_k}$ and $\mathbf{b}_2^{(k)} \in \mathbb{R}^{m_k}$ are parameters. Finally, MERGE minimizes the ℓ_2-distance between $\mathbf{x}_{i,k}$ and $\mathbf{x}'_{i,k}$ over all possible (i, k) pairs.

4 LEARNING WITH MERGE

In this section, we derive the overall objective function to compute the graph embedding. First of all, we consider the two cross-view consistencies in Eq. 1 and Eq. 2. Ideally, to enforce the two cross-view consistencies, we can directly maximize both $p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1})$ and $p(\mathbf{e}_{i,2}|\mathbf{e}_{i,1})$ for all the i's and their neighboring j's. However, since we need to sum up all the nodes in V, it is time consuming to compute the normalization terms in Eq. 1 and Eq. 2. DeepWalk exploits a hierarchical approximation to the softmax function in Eq. 1 and Eq. 2, which reduces the overall complexity to $O(|V| \log |V|)$. MERGE applies negative sampling [19] to replace the softmax function to further reduce the complexity. Specifically, instead of maximizing $\log p(\mathbf{e}_{j,2}|\mathbf{e}_{i,1}) + \log p(\mathbf{e}_{i,2}|\mathbf{e}_{i,1})$, MERGE introduces a surrogate function:

$$\begin{aligned} \Delta_{i,j} \quad = \quad & \log \sigma(\mathbf{e}_{j,2}^\top \mathbf{e}_{i,1}) \\ & + \textstyle\sum_{t=1}^{\ell_1} \mathbb{E}_{v_h \sim P_n(v_i)} [\log \sigma(-\mathbf{e}_{h,2}^\top \mathbf{e}_{i,1})]. \end{aligned} \tag{5}$$

Here $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function; $v_h \sim P_n(v_i)$ indicates sampling a node v_h as a *negative context* of node v_i from a distribution $P_n(v_i)$. We follow [19] to define $P_n(v_i) \propto deg(v_i)^{3/4}$, where $deg(v_i)$ is v_i's degree. Besides, $\mathbb{E}_{v_h \sim P_n(v_i)}[\cdot]$ is the expectation over $P_n(v_i)$. Denote the positive context set of node v_i as C_i, where $C_i \subset V$. Then, we define the objective function by enforcing the two cross-view consistencies as

$$O_1 = -\tfrac{1}{n} \textstyle\sum_{v_i \in V} \sum_{v_j \in C_i \cup v_i} \Delta_{i,j}, \tag{6}$$

where $n = \sum_{i:v_i \in V} |C_i \cup v_i|$ is the number of $\Delta_{i,j}$'s. By minimizing O_1, we enforce the two cross-view consistencies with second-order proximity. This O_1 formulation Eq. 6 also leads to reduce the complexity to $O(|V|)$, as we will see later.

Next, we consider the nonlinear embedding for two

Algorithm 1 Learning Algorithm for MERGE

Require: graph $G = (V, E)$ with node features $\mathbf{x}_{i,k}$'s, context window size ℓ_0, # of negative samples ℓ_1, sample path length τ, # of paths per node ℓ_3.
Ensure: embedding $\mathbf{e}_{i,k}$'s, parameters Θ.
1: Initialize the parameters in $\mathcal{M}$;
2: **while** not converged **do**
3: Paths $P \leftarrow$ SamplePath($G, \ell_0, \ell_1, \tau, \ell_3$);
4: Batches $\{(v_i, v_j, \lambda)\}, B \leftarrow$ ConstructBatch(P);
5: **for** each batch $\{(v_i, v_j, \lambda)\}$ **do**
6: $\nabla_{\mathcal{M}} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathcal{M}}$ by AdaDelta;
7: $\mathcal{M} \leftarrow$ GradDescent($\nabla_{\mathcal{M}}, \mathcal{M}$);
8: **end for**
9: **end while**

views. Specifically, we learn the nonlinear network embedding and nonlinear content embedding by minimizing:

$$\begin{cases} O_2 = \sum_{v_i \in V} \|\mathbf{x}'_{i,1} - \mathbf{x}_{i,1}\|^2 \\ O_3 = \sum_{v_i \in V} \|\mathbf{x}'_{i,2} - \mathbf{x}_{i,2}\|^2 \end{cases}. \tag{7}$$

Objective function. MERGE combines O_1, O_2 and O_3 together to learn $\mathbf{e}_{i,k}$'s. Denote our model $\mathcal{M}$ as $\{\mathbf{e}_{i,k}, \Theta | i = 1, ..., |V|, k = 1, 2\}$, which parameters are listed in $\Theta = \{W_1^{(k)}, W_2^{(k)}, \mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}\}$. The overall objective function to minimize is

$$\mathcal{L}(\mathcal{M}) = O_1 + \alpha O_2 + \beta O_3 + \gamma \Omega(\mathcal{M}), \tag{8}$$

where α, β, γ are a set of positive trade-off parameters. $\Omega(\mathcal{M})$ is a regularization term over $\mathcal{M}$; *e.g.*, it sums up the ℓ_2-norm of each parameter in $\mathcal{M}$.

Learning algorithm. To optimize $\mathcal{L}$ in Eq. 8, we use batch stochastic gradient descent algorithm. The detailed derivations of the parameter gradients are complex, especially when the layers of deep AutoEncoder increases. Due to space limit, we skip the gradient details in this paper. Empirically, these gradients can be automatically computed in the state-of-the-art deep learning toolkits. Since all the O_1, O_2 and O_3 are larger than zero, $\mathcal{L}$ is bounded, the minimization will converge in the end. Empirically, the MERGE algorithm converges quickly, as shown later in Sec. 5.

We summarize learning algorithm for MERGE in Alg. 1. In line 1, we initialize all the parameters $\mathbf{e}_{i,k}$'s and Θ. In line 3, we sample a set of paths from G, in order to identify the context neighbors for each node v_i. In particular, starting from each node $v \in V$, we sample ℓ_3 paths by random walk. Each path's length is τ. As we use negative sampling for O_1, each node v_i has both positive context and negative context. Thus, in line 4, we construct B batches, each of which has a set of tuples $\{(v_i, v_j, \lambda)\}$ from the sampled paths P. Here, v_j is v_i's positive context when $\lambda = 1$, negative when $\lambda = -1$. Empirically, we find the batch size range from 100 to 1000 working well. In lines $5 \sim 8$, we compute the batch gradient for each parameter in $\mathcal{M}$ and do gradient descent.

Algorithm complexity. We analyze the complexity of Alg. 1. Parameter initialization in line 1 takes $O(|V|)$. Path sampling in line 3 takes $O(|V|\tau\ell_3)$. In line 4, the following steps are implemented:

1. extracting all the v_i and its context neighbor pairs, which takes $O(|V|\tau\ell_3)$;
2. sampling ℓ_4 negative context nodes for v_i, which takes $O(\ell_4)$;
3. sampling B batches, each of size ℓ_4, which takes $O(B\ell_4)$.

In line 6, we compute the gradient of $\mathcal{L}$ w.r.t. each batch. For O_1, it takes $O(1)$ for a single node. For O_2, as each network feature vector has the length of $O(|V|)$, it takes $O(|V|)$ to compute the gradient. For O_3, it takes $O(1)$ for a single node. For Ω, it takes $O(|V|)$ since the network embedding has $O(|V|)$ dimensions. In total, line 6 takes $O(|V|\ell_4)$. Line 7 takes $O(|V|)$ to update the parameters. The complexity of Alg. 1 is $O(|V|) = O(|V| + |V|\tau\ell_3 + \ell_4 + B\ell_4 + B(|V|\ell_4 + |V|))$, which is much smaller than the TADW's $O(|E| + |V|)$.

5 EXPERIMENTS

In this section, we demonstrate the effectiveness of MERGE by comparing it against the current state-of-the-art baselines. MERGE shows a superior advantage over all other models in extensive experiments. In addition, we present a comprehensive analysis on MERGE in terms of its parameter sensitivity, algorithm convergence and computational complexity.

5.1 Experimental Setup

To quantitatively evaluate the quality of our node embedding, we follow [20, 5] to use node classification as the evaluation task. We feed the embedding output from each method into a logistic regression model for node classification. We use *micro-F1* (*i.e.*, the overall F1 w.r.t. all the classes) and *macro-F1* (*i.e.*, the average F1 w.r.t. each class) as the evaluation metrics. For each dataset, we randomly sample 50% of nodes as the test set D_{tst},

Table 1: Benchmark datasets.

Dataset	#(node)	#(edge)	#(class)
Cora	2708	5429	7
PubMed	19717	44338	3
CiteSeer	3264	4591	6
Wikipedia	2405	17981	19
MicroblogPCU	781	3315	2

30% as the training set D_{trn} and the remaining as the development set D_{dev}. We first train the model on D_{trn} and tune the hyperparameters with D_{dev}. Then we fix the hyperparameters, re-train our model on $D_{trn} \cup D_{dev}$ and test it on D_{tst}. We repeat this procedure for 10 times and report the average results. All experiments are implemented on a Linux server with 64GB memory, 24 Intel Xeon CPUs (2.40GHz) and 1 Intel Tesla K80 GPU.

Datasets. Five different benchmark datasets are used for evaluation:

1. Three paper citation networks[1] as used in [33], including *Cora*, *PubMed* and *CiteSeer*. In these networks, nodes are papers, edges are citations and classes are paper categories.
2. One web document network[2], *i.e.*, *Wikipedia*, as used in [32]. In this network, nodes are Wikipedia articles, edges are hyperlinks and classes are web page categories.
3. One microblog user network, *i.e.*, *MicroblogPCU*[3]. In this network, nodes are users, edges are followship and classes indicate spam users.

The network statistics of these datasets are summarized in Table 1. Next we introduce the node features of each dataset. For *Cora* and *Citeseer*, each node contains a preprocessed bag-of-words text feature vector. For *PubMed* and *Wikipedia*, each node corresponds a preprocessed *tf-idf* text feature vector. For *MicroblogPCU*, each node contains a raw document. As the vocabulary size is large and the text is noisy, we follow TADW [32] to apply PCA for dimensionality reduction on the node-by-text feature matrix of each dataset. Then we keep m_2 dimensions corresponding to the largest singular values, which retain 90% of the *energy* [21]. Thus each node has an m_2-dimensional content feature vector. Similarly, we also apply PCA to reduce each dataset's node adjacency matrix to m_1-dimensional network feature vector.

Baselines. To give a clear illustration on the contribution of cross-view consistency and model nonlinearity respectively, we classify our baselines into several categories:

1. Models which only use content information, such as *AEC*: AutoEncoder of Content applies AutoEncoder [4] to the content features.
2. Models which only explore relational information, such as *DeepWalk* [20], *Node2Vec* [9], *SDNE* [27] and *struc2vec* [22].
3. Models which naively concatenate a content feature vector and a network embedding vector. These approaches do not model the cross-view consistency, which include: *DeepWalk+AEC*, *Node2Vec+AEC*, *SDNE+AEC*, *struc2vec+AEC* and *AEG+AEC*, where AutoEncoder of Graph (AEG) is a method that applies AutoEncoder to the network features to learn a network embedding for each node.
4. *TADW* [32]: it learns both network embedding and content embedding by matrix decomposition, which only models the linearity.
5. *DCCA* [2]: it learns a network embedding and a content embedding for each node by applying a deep neural network to the corresponding features and enforcing the embedding to be maximally correlated. It does not consider relational information.

Parameters. MERGE has a similar set of hyperparameters as DeepWalk, which includes the context window size ℓ_0, the number of negative samples ℓ_1, the sample path length τ, the number of paths per node ℓ_3 and the embedding dimension d. Typically, we set $\ell_0 = 4, \ell_1 = 9, \tau = 10, \ell_3 = 10$ and $d = 128$ for all of the datasets. An exception is in *PubMed*, which is larger than the other datasets: we follow the suggestion in [19] to set a larger dimension $d = 256$ and a smaller number of negative samples $\ell_1 = 2$. In addition, we have tradeoff hyperparameters $\{\alpha, \beta, \gamma\}$. We use grid search over {1e-6, 1e-5, 1e-4, 1e-3, 1e-2} to tune each of them (results to be shown later). Typically, we set α = 1e-4, β = 1e-3, γ = 1e-5 for all the datasets.

For each baseline's hyperparameters, we take their suggested values as in references and fine tune them with the development set on each dataset. For DeepWalk and Node2Vec, we set $\ell_0 = 10, \tau = 10, \ell_3 = 80, d = 128$. For Node2Vec, we optimize p and q with a grid search over $p, q \in 0.25, 0.50, 1, 2, 4$ as suggested by [9] on each dataset's development set. For TADW, we use the optimal parameter values reported by [32] for *Cora*, *CiteSeer* and *Wikipedia*; besides, we set its hyperparameter k =

[1] http://linqs.umiacs.umd.edu/projects/projects/lbc
[2] http://linqs.cs.umd.edu/projects//projects/lbc/index.html
[3] https://archive.ics.uci.edu/ml/datasets/microblogPCU

200 for *PubMed* and $k = 100$ for *MicroblogPCU*. For DCCA, we set its hyperparameters α = 1e-4 and β = 1e-4. For SDNE, we optimize the hyperparameters of $\alpha \in \{1, 10, 100, 300, 500, 700\}$, $\beta \in \{1, 10, 30, 50, 100\}$, and $\gamma \in \{1e-4, 1e-3, 1e-2, 1e-1, 1, 10\}$ on the validation set. For struc2vec, we fine tune the hyperparameters of $\ell_0 \in (0, 10)$, $\tau \in \{10, 30, 50, 70, 90\}$ and $\ell_3 \in \{1, 5, 10, 15\}$ on the validation set. Besides, we set the maximum layer used in struc2vec as six and enable all of the optimization options in the paper. Note that it is time consuming to get the comparable results when the optimization options are disabled.

5.2 Model Performances

We report the results in Table 2. Note that all the improvements discussed below are calculated in relative values. For each dataset, we average the improvements over all the training ratios due to space limit. We draw conclusions as follows.

- Jointly considering both content and network information for graph embedding is critical, as indicated by the uncompetitive performance of AEC, DeepWalk, Node2Vec, SDNE, and struc2vec. In contrast, the naive method of concatenating content-based feature with network embedding vector already improves the overall performance. DeepWalk + AEC improves by 1% $\sim$ 28% over DeepWalk alone, Node2Vec + AEC improves by 1% $\sim$ 24%, SDNE + AEC improves by 1% $\sim$ 60% and struc2vec improves by 5% $\sim$ 320% in terms of macro-F1. Similarly, AEG+AEC improves AEC by 1% $\sim$ 24%. However, directly concatenating two embeddings does not exploit the cross-view relations among data, and thus cannot guarantee the performance improvements. For example, AEC outperforms both DeepWalk+AEC and Node2Vec+AEC in PubMed and CiteSeer.

- Utilization of either cross-instance consistency (DeepWalk+AEC, node2vec+AEC, AEG+AEC, SDNE+AEC, struc2vec+AEC) or cross-view consistency (DCCA) alone is not sufficient to model the relations in data. Particularly, by enforcing cross-instance–cross-view consistency, MERGE outperforms cross-instance-only baselines by at least 2% $\sim$ 17% (macro-F1) and 2% $\sim$ 9% (micro-F1) on average, and it improves DCCA by averagely 4% $\sim$ 20% (macro-F1) and 4% $\sim$ 17% (micro-F1). This again proves that maximizing the utility of both views for graph embedding, we need to carefully leverage the interdependency between the two views rather than simply concatenating them.

- Modeling data nonlinearity in graph embedding gives rise to an enhancement of performances. It is shown that MERGE is better than TADW by 6.87% $\sim$ 20% (macro-F1) and 4% $\sim$ 16% (micro-F1) on average.

- By comprehensively modeling cross-instance–cross-view consistency with nonlinearity, MERGE consistently and significantly outperforms all the baselines on all the five datasets with different training ratios. Specifically, in most datasets training with only 30% of labeled nodes, MERGE outperforms the baselines which are trained with 100% of the training data(micro-F1) except Wikipedia and MicroblogPCU . On average, MERGE improves the best baselines by 2.1% $\sim$ 10.3% (macro-F1) and 2.3% $\sim$ 9.3% (micro-F1) across all the datasets. The advantage of MERGE is more favorable especially with less training data. For example, in Cora, MERGE outperforms the baselines at least by 17.8% when using 10% training data and 6.2% when using 100% training data in terms of macro-F1.

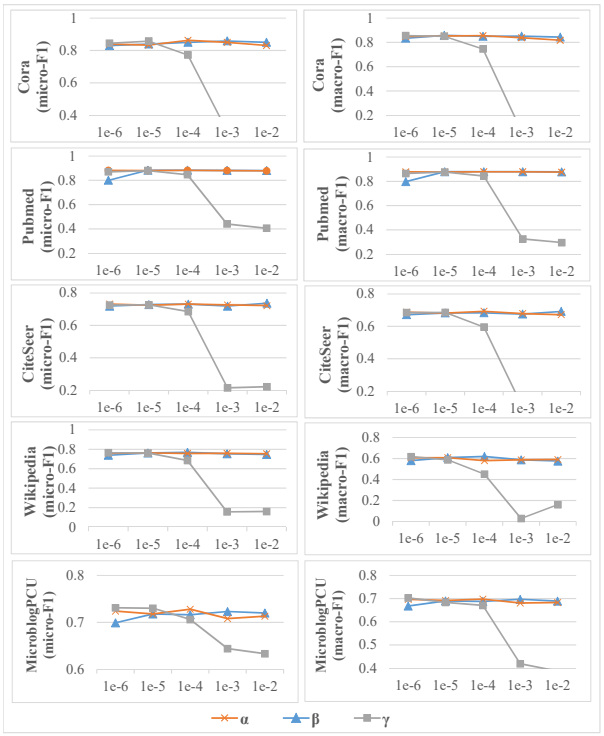


Figure 3: Results on parameter sensitivity.

5.3 Model Analysis

We validate the parameter sensitivity for MERGE by tuning our model parameters α, β and γ. As shown in

Table 2: Comparison with the baselines under different amounts of training labels. The bigger F1 value is better.

		macro-F1 (%)										micro-F1 (%)									
	% Labels	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Cora	AEC	27.8	43.7	52.1	54.8	59.4	62.1	64.0	65.2	66.0	67.0	44.5	54.9	60.1	62.8	65.6	67.2	68.9	69.5	70.3	70.9
	DeepWalk	61.8	68.1	71.2	72.8	73.9	74.8	75.8	76.2	76.7	77.1	64.2	69.7	72.5	74.3	75.0	76.1	76.9	77.3	77.6	78.1
	Node2Vec	63.1	70.2	72.8	74.5	76.0	76.5	77.4	77.9	78.1	78.3	65.7	72.0	74.4	76.1	77.5	77.9	78.9	79.3	79.5	79.7
	SDNE	60.1	67.7	70.3	71.4	72.4	73.9	74.6	75.1	75.7	76.2	62.9	69.4	71.6	72.7	73.7	74.9	75.6	76.1	76.6	77.0
	struc2vec	21.0	23.2	24.6	26.1	27.4	28.2	28.5	29.6	29.9	30.6	26.2	27.7	29.2	30.9	32.3	33.3	34.0	35.2	35.7	36.7
	DeepWalk+AEC	62.5	68.8	72.4	74.2	75.9	76.8	78.4	78.9	79.8	80.3	64.8	70.6	73.7	75.7	77.2	78.2	79.6	80.2	80.9	81.5
	Node2Vec+AEC	63.6	70.7	73.7	75.5	77.2	78.0	79.0	79.6	80.2	80.9	66.3	72.6	75.3	77.2	78.7	79.5	80.5	81.2	81.6	82.3
	AEG+AEC	37.9	57.6	66.3	69.0	72.8	75.2	77.0	77.9	79.0	79.7	51.0	64.0	69.8	72.8	75.5	77.4	79.1	79.7	80.6	81.1
	SDNE+AEC	66.3	71.7	73.7	75.2	76.0	77.1	77.9	78.2	79.0	79.7	69.8	73.8	75.5	76.9	77.7	78.6	79.5	79.9	80.6	81.1
	struc2vec+AEC	45.5	51.5	53.7	55.2	57.1	58.4	59.7	60.8	61.7	62.6	51.2	56.0	57.9	59.0	60.7	61.9	63.2	64.1	65.1	65.8
	TADW	59.3	68.3	70.2	70.7	71.6	72.0	72.4	72.4	73.1	73.1	65.5	71.1	72.8	73.4	74.1	74.3	74.8	74.9	75.3	75.5
	DCCA	58.8	66.4	68.6	69.7	71.0	71.6	72.8	72.9	73.3	73.5	64.0	69.4	71.4	72.5	73.5	73.9	75.0	75.3	75.5	75.7
	MERGE	**78.1**	**82.1**	**83.3**	**83.9**	**84.1**	**84.8**	**85.3**	**85.4**	**85.8**	**85.9**	**80.2**	**83.4**	**84.6**	**85.0**	**85.3**	**85.9**	**86.5**	**86.6**	**86.9**	**86.9**
PubMed	AEC	82.3	83.6	84.3	84.7	85.0	85.0	85.2	85.3	85.5	85.5	82.2	83.6	84.2	84.6	84.9	85.0	85.2	85.3	85.4	85.5
	DeepWalk	64.7	65.5	66.3	67.0	67.0	67.2	67.2	67.3	67.4	67.5	67.8	69.5	70.4	71.2	71.3	71.6	71.6	71.7	71.9	72.0
	Node2Vec	66.7	67.7	68.3	68.7	68.9	69.1	69.1	69.4	69.5	69.6	69.5	71.2	72.0	72.5	72.9	73.0	73.1	73.4	73.5	73.7
	SDNE	59.4	61.2	61.9	62.3	62.4	62.5	62.5	62.5	62.6	62.7	62.2	64.2	64.9	65.2	65.4	65.5	65.5	65.5	65.6	65.7
	struc2vec	39.5	41.4	42.2	42.6	43.1	43.4	43.4	43.7	43.8	43.7	42.2	44.9	46.0	46.7	47.3	47.7	47.8	48.2	48.4	48.4
	DeepWalk+AEC	78.7	83.4	84.9	85.7	86.3	86.5	86.7	86.9	87.1	87.2	79.2	83.7	85.2	86.0	86.6	86.7	86.9	87.2	87.3	87.4
	Node2Vec+AEC	79.0	83.7	85.1	85.9	86.4	86.6	86.9	87.0	87.1	87.2	79.6	83.9	85.4	86.1	86.6	86.8	87.0	87.2	87.3	87.4
	AEG+AEC	83.2	84.7	85.4	85.9	86.3	86.3	86.5	86.7	86.8	86.9	83.2	84.8	85.5	86.0	86.4	86.4	86.7	86.8	87.0	87.1
	SDNE+AEC	78.0	80.7	83.1	84.2	85.0	85.4	85.7	85.9	86.0	86.2	78.1	80.8	83.2	84.2	84.9	85.3	85.7	85.9	86.0	86.2
	struc2vec+AEC	75.9	76.8	79.4	80.9	82.2	83.2	83.8	84.2	84.6	84.7	75.6	76.8	79.3	80.9	82.2	83.1	83.7	84.1	84.5	84.6
	TADW	67.8	76.2	79.2	80.3	81.2	81.7	82.1	82.4	82.7	82.9	71.5	76.9	79.5	80.4	81.3	81.7	82.1	82.4	82.7	82.8
	DCCA	82.9	83.9	84.2	84.5	84.6	84.7	84.8	84.8	84.7	84.8	83.1	84.1	84.4	84.7	84.8	84.8	84.9	85.0	84.9	85.0
	MERGE	**86.2**	**87.2**	**87.6**	**87.8**	**88.1**	**88.1**	**88.3**	**88.3**	**88.4**	**88.5**	**86.6**	**87.5**	**87.9**	**88.1**	**88.4**	**88.4**	**88.6**	**88.6**	**88.7**	**88.8**
CiteSeer	AEC	45.4	55.2	57.7	58.8	59.5	60.2	60.1	61.0	61.3	61.6	55.3	64.3	66.8	67.7	68.3	69.0	69.1	69.4	69.7	69.8
	DeepWalk	39.7	43.3	45.9	47.3	48.4	49.5	49.6	50.0	50.2	50.7	43.4	46.6	49.6	51.1	52.5	53.7	54.0	54.5	54.9	55.5
	Node2Vec	40.4	45.5	48.3	50.5	52.0	52.6	52.9	53.3	54.2	53.8	44.1	49.1	52.3	54.5	56.3	56.9	57.3	57.9	58.6	58.5
	SDNE	33.6	37.2	38.9	39.7	40.3	41.0	42.0	42.3	42.6	42.8	37.8	41.6	43.1	44.4	45.2	45.8	46.7	47.0	47.2	47.6
	struc2vec	19.6	20.5	21.4	22.4	23.0	23.5	23.4	24.1	24.5	24.8	21.4	22.3	23.3	24.6	25.3	25.9	26.1	27.0	27.4	28.0
	DeepWalk+AEC	40.9	45.8	49.7	53.0	55.3	57.6	59.1	60.4	61.4	62.2	44.6	49.4	53.8	57.2	60.0	62.3	64.1	65.2	66.4	67.3
	Node2Vec+AEC	41.3	47.5	51.9	55.5	57.6	59.4	60.4	61.9	63.2	63.7	45.1	51.3	56.1	59.6	62.3	64.0	65.2	66.6	67.8	68.4
	AEG+AEC	49.0	57.8	60.1	61.1	62.2	62.8	62.6	63.6	64.0	64.4	58.6	66.7	69.1	70.0	70.7	71.1	71.4	71.6	71.9	72.2
	SDNE+AEC	53.7	56.9	59.1	60.6	61.2	62.0	63.4	63.8	64.3	64.6	57.4	61.1	63.5	65.0	66.1	66.7	68.0	68.5	69.1	69.5
	struc2vec+AEC	48.8	51.7	51.6	53.0	54.0	54.8	56.6	57.6	58.2	59.4	53.1	55.6	55.7	57.3	58.4	59.2	60.9	62.1	62.7	64.0
	TADW	58.8	62.0	62.8	63.4	64.1	64.8	64.7	65.5	65.6	65.7	65.5	67.5	68.4	68.7	69.5	69.9	70.1	70.5	70.7	70.8
	DCCA	50.3	55.2	57.3	58.2	58.9	59.8	60.2	60.6	61.1	61.2	55.3	60.2	62.2	63.2	64.0	64.6	65.3	65.5	65.9	66.2
	MERGE	**62.3**	**65.7**	**66.4**	**68.0**	**68.7**	**69.6**	**69.3**	**70.1**	**70.5**	**70.6**	**69.5**	**71.9**	**72.8**	**73.7**	**74.2**	**74.8**	**75.1**	**75.1**	**75.4**	**75.5**
Wikipedia	AEC	32.5	40.3	46.5	49.6	50.8	53.6	55.6	56.4	58.4	59.4	51.7	62.1	67.1	69.8	70.5	72.5	73.2	74.1	74.9	75.6
	DeepWalk	36.4	40.9	44.3	46.1	47.2	49.2	50.1	50.9	52.1	52.3	49.3	54.6	57.1	59.5	60.7	61.7	62.6	63.4	64.2	64.4
	Node2Vec	37.3	42.4	44.6	46.8	48.2	49.4	51.6	51.9	52.9	53.4	50.6	54.8	58.0	59.5	61.0	61.9	63.1	63.3	64.0	64.5
	SDNE	31.8	37.1	39.3	42.0	43.6	45.4	46.1	46.9	48.2	48.7	41.4	48.1	51.1	53.4	55.4	56.7	57.3	58.4	59.2	59.8
	struc2vec	7.8	8.6	9.5	10.2	10.3	10.6	11.1	11.1	11.5	11.5	12.7	13.5	14.4	15.1	15.0	15.3	16.0	16.0	16.3	16.5
	DeepWalk+AEC	37.1	42.3	46.5	49.0	51.1	53.1	55.7	56.5	58.9	59.2	50.0	56.1	60.1	62.7	65.2	66.4	68.7	69.8	71.3	72.1
	Node2Vec+AEC	37.9	43.7	46.5	49.6	51.7	53.8	56.3	57.9	59.6	60.2	51.4	56.3	60.4	62.9	65.3	66.6	68.5	70.0	71.3	72.1
	AEG+AEC	37.9	45.3	50.5	54.3	55.2	57.9	59.3	60.0	61.7	62.4	56.1	65.3	69.8	72.5	73.5	75.1	75.9	76.8	77.7	78.2
	SDNE+AEC	33.4	41.9	46.6	49.6	51.6	54.0	55.1	57.3	59.3	59.4	46.4	56.0	61.0	64.4	66.8	68.7	69.8	71.3	72.4	73.2
	struc2vec+AEC	11.2	18.9	27.0	32.4	36.4	40.0	42.6	45.0	46.9	48.4	19.0	29.8	40.3	47.7	52.4	56.6	59.7	62.1	64.3	65.8
	TADW	46.7	53.2	56.0	57.0	60.8	60.3	62.3	63.4	62.7	64.2	59.5	64.3	67.4	67.8	69.7	70.0	70.0	71.4	71.2	71.9
	DCCA	43.5	48.6	50.4	53.1	54.2	54.7	56.4	57.0	57.0	57.4	59.4	64.4	66.7	68.4	69.8	70.3	71.0	71.6	71.8	72.3
	MERGE	**50.4**	**57.9**	**59.4**	**62.1**	**63.7**	**64.4**	**66.9**	**67.7**	**68.2**	**69.2**	**68.2**	**73.0**	**75.0**	**76.6**	**77.4**	**77.9**	**78.9**	**79.4**	**79.7**	**80.1**
microblogPCU	AEC	39.3	44.4	48.4	52.0	52.4	52.8	58.2	59.6	59.2	60.2	63.2	64.4	66.0	67.4	67.0	67.8	69.4	70.1	69.8	69.9
	DeepWalk	60.3	60.8	58.5	59.1	63.1	66.6	64.9	66.1	64.1	65.6	63.9	64.9	63.3	63.0	66.0	70.1	68.8	69.9	68.1	69.7
	Node2Vec	58.5	59.4	56.9	57.9	60.8	62.9	62.9	65.0	63.6	63.4	62.3	63.8	62.6	62.3	64.4	67.4	68.0	70.0	68.4	68.3
	SDNE	53.5	53.9	53.2	58.6	59.1	58.2	57.1	56.9	58.3	61.8	55.6	58.8	57.7	61.3	64.9	63.6	61.3	63.4	63.0	65.6
	struc2vec	47.4	55.6	57.1	58.3	56.5	58.1	56.7	58.1	57.8	60.0	50.3	57.7	58.7	60.6	59.1	60.8	59.2	60.6	60.2	62.6
	DeepWalk+AEC	60.4	61.0	58.3	59.6	64.0	67.5	65.1	67.1	65.9	67.3	64.2	65.3	64.0	63.6	66.7	70.8	68.6	70.8	69.7	70.9
	Node2Vec+AEC	57.4	58.3	57.4	58.2	61.7	63.7	64.1	65.5	65.1	64.7	61.9	63.6	63.4	62.8	65.1	68.1	68.9	70.3	69.8	69.4
	AEG+AEC	43.9	47.8	51.0	57.8	57.0	62.4	64.4	67.0	65.6	67.4	63.8	65.1	66.7	68.7	67.9	71.6	71.8	74.3	72.9	73.8
	SDNE+AEC	54.0	58.3	61.1	64.6	64.8	66.9	67.7	68.4	69.1	69.3	56.4	60.0	62.3	65.8	66.2	68.3	69.3	70.0	70.7	71.0
	struc2vec+AEC	53.0	58.2	60.5	62.7	63.4	65.1	63.9	65.1	66.1	66.1	54.9	59.1	61.4	63.9	64.9	66.4	65.4	66.7	67.6	67.6
	TADW	56.8	59.7	60.5	60.6	62.7	62.7	63.1	62.8	63.2	62.7	64.4	68.3	69.2	69.4	71.4	71.2	71.9	71.8	71.7	71.6
	DCCA	53.8	54.6	57.5	60.4	61.1	60.5	63.6	62.8	64.4	65.4	63.8	66.1	67.1	68.3	69.0	69.1	71.0	70.0	71.3	72.0
	MERGE	**63.6**	**68.4**	**67.5**	**70.5**	**71.2**	**69.3**	**70.0**	**73.4**	**71.8**	**71.2**	**69.7**	**72.4**	**71.1**	**73.7**	**73.6**	**72.9**	**73.0**	**76.1**	**74.7**	**74.1**

only use content information | only use relational information
concatenate a content and network embedding | learns both network and content embedding

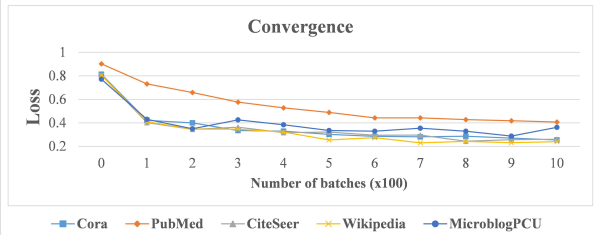


Figure 4: Results on convergence.

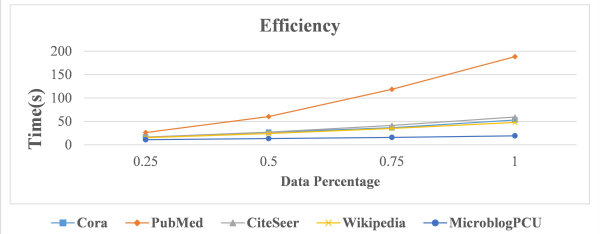


Figure 5: Results on running time.

Fig. 3, MERGE performs optimally when α and $\beta \in \{1e-5, 1e-4, 1e-3, 1e-2\}$ and $\gamma \in \{1e-6, 1e-5\}$. Detailed parameter settings are described in Sec. 5.1. Specifically, our model is relatively robust to the values of α and β. While taking large value for γ causes over-regularization of the model, which leads to poor performance. The suggested γ value is less than $1e-4$. We further validate the convergence and time complexity of MERGE. As shown in Fig. 4, the loss of the model normalized by the number of nodes converges quickly within a few iterations. In terms of asymptotic computation complexity, we test the algorithm complexity with different amounts of nodes (*i.e.*, different $|V|$ values). As shown in Fig. 5, it is observed that the running time of MERGE scales linearly with $|V|$.

6 CONCLUSION

In this paper, we systematically investigate the problem of UMGE, where multiple perspectives such as content and network features are exploited to learn graph embedding in an unsupervised fashion. We discover that cross-view and nonlinearity play a critical role in improving graph embedding quality. Therefore, we develop a simple, yet effective approach, Multi-viEw nonlineaR Graph Embedding (MERGE), to model relational multi-view consistency with nonlinearity. MERGE incorporates both network embedding and content embedding with a nonlinear AutoEncoder. Then, it enforces both intra-instance–cross-view consistency and cross-instance–cross-view consistency among nodes' network embedding with its neighbors' content embedding. The experimental results demonstrate that MERGE consistently outperforms the state-of-the-art baselines by at least relatively 2.12% $\sim$ 10.91% (macro-F1) and 2.29% $\sim$ 10.12% (micro-F1) on average over five public datasets, yet with $O(|V|)$ complexity. It is worthy to note that, in most cases, MERGE outperforms the baselines in favor of using relatively one third of training data yet with superior performance.

References

[1] ANDERSON, T. W. *An Introduction to Multivariate Statistical Analysis*, second ed. Wiley, New York, NY, 1984.

[2] ANDREW, G., ARORA, R., BILMES, J. A., AND LIVESCU, K. Deep canonical correlation analysis. In *ICML* (2013), pp. 1247–1255.

[3] BELKIN, M., AND NIYOGI, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS* (2001), pp. 585–591.

[4] BENGIO, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[5] CAO, S., LU, W., AND XU, Q. Grarep: Learning graph representations with global structural information. In *CIKM* (2015), pp. 891–900.

[6] CHANG, S., HAN, W., TANG, J., QI, G., AGGARWAL, C. C., AND HUANG, T. S. Heterogeneous network embedding via deep architectures. In *KDD* (2015), pp. 119–128.

[7] COX, T. F., AND COX, M. *Multidimensional Scaling, Second Edition*, 2 ed. Chapman and Hall/CRC, 2000.

[8] ELKAHKY, A. M., SONG, Y., AND HE, X. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW* (2015), pp. 278–288.

[9] GROVER, A., AND LESKOVEC, J. node2vec: Scalable feature learning for networks. In *KDD* (2016).

[10] HE, J., DU, C., ZHUANG, F., YIN, X., HE, Q., AND LONG, G. Online bayesian max-margin subspace multi-view learning. In *IJCAI* (2016), pp. 1555–1561.

[11] HOU, C., NIE, F., TAO, H., AND YI, D. Multi-view unsupervised feature selection with adaptive similarity and view weight. *IEEE Transactions on Knowledge and Data Engineering 29*, 9 (2017), 1998–2011.

[12] HUANG, Z., SHAN, S., ZHANG, H., LAO, S., AND CHEN, X. Cross-view graph embedding. In *ACCV* (2013), pp. 770–781.

[13] KIPF, T. N., AND WELLING, M. Semi-supervised classification with graph convolutional networks.

[14] LI, Y., NIE, F., HUANG, H., AND HUANG, J. Large-scale multi-view spectral clustering via bipartite graph. In *AAAI* (2015), pp. 2750–2756.

[15] LI, Y., YANG, M., AND ZHANG, Z. Multi-view representation learning: A survey from shallow methods to deep methods. *CoRR abs/1610.01206* (2016).

[16] LIU, Z., ZHENG, V. W., ZHAO, Z., AND LI, Z. Interactive paths embedding for semantic proximity search on heterogeneous graphs. In *KDD* (2018).

[17] MACSKASSY, S. A., AND PROVOST, F. J. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research 8* (2007), 935–983.

[18] MANSIMOV, E., PARISOTTO, E., BA, L. J., AND SALAKHUTDINOV, R. Generating images from captions with attention. In *ICLR* (2016).

[19] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *NIPS* (2013), pp. 3111–3119.

[20] PEROZZI, B., AL-RFOU, R., AND SKIENA, S. Deepwalk: Online learning of social representations. In *KDD* (2014), pp. 701–710.

[21] RAJARAMAN, A., AND ULLMAN, J. D. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[22] RIBEIRO, L. F., SAVERESE, P. H., AND FIGUEIREDO, D. R. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), ACM, pp. 385–394.

[23] ROWEIS, S. T., AND SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 5500 (2000), 2323–2326.

[24] TANG, J., QU, M., AND MEI, Q. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD* (2015), pp. 1165–1174.

[25] TANG, J., QU, M., WANG, M., ZHANG, M., YAN, J., AND MEI, Q. Line: Large-scale information network embedding. In *WWW* (2015), pp. 1067–1077.

[26] TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323.

[27] WANG, D., CUI, P., AND ZHU, W. Structural deep network embedding. In *KDD* (2016), pp. 1225–1234.

[28] WANG, S., DING, Z., AND FU, Y. Coupled marginalized auto-encoders for cross-domain multi-view learning. In *IJCAI* (2016), pp. 2125–2131.

[29] WANG, W., ARORA, R., LIVESCU, K., AND BILMES, J. A. On deep multi-view representation learning. In *ICML* (2015), pp. 1083–1092.

[30] XIE, R., LIU, Z., JIA, J., LUAN, H., AND SUN, M. Representation learning of knowledge graphs with entity descriptions. In *AAAI* (2016), pp. 2659–2665.

[31] XU, C., TAO, D., AND XU, C. A survey on multi-view learning. *CoRR abs/1304.5634* (2013).

[32] YANG, C., LIU, Z., ZHAO, D., SUN, M., AND CHANG, E. Y. Network representation learning with rich text information. In *IJCAI* (2015), pp. 2111–2117.

[33] YANG, Z., COHEN, W. W., AND SALAKHUTDINOV, R. Revisiting semi-supervised learning with graph embeddings. In *ICML* (2016), pp. 40–48.

[34] ZHANG, L., WANG, S., ZHANG, X., WANG, Y., LI, B., SHEN, D., AND JI, S. Collaborative multi-view denoising. In *KDD* (2016), pp. 2045–2054.

[35] ZHOU, D., AND BURGES, C. J. C. Spectral clustering and transductive learning with multiple views. In *ICML* (2007), pp. 1159–1166.

[36] ZHOU, Y., CHENG, H., AND YU, J. X. Graph clustering based on structural/attribute similarities. *PVLDB 2*, 1 (2009), 718–729.

[37] ZHUGE, W., NIE, F., HOU, C., AND YI, D. Unsupervised single and multiple views feature extraction with structured graph. *IEEE Transactions on Knowledge and Data Engineering 29*, 10 (2017), 2347–2359.