2018 IEEE International Conference on Big Knowledge

Discriminative Graph Autoencoder

Haifeng Jin, Qingquan Song, Xia Hu
Department of Computer Science and Engineering, Texas A&M University
College Station, United States
{jin, song_3134, xiahu}@tamu.edu

***Abstract*—With the abundance of graph-structured data in various applications, graph representation learning has become an effective computational tool for seeking informative vector representations for graphs. Traditional graph kernel approaches are usually frequency-based. Each dimension of a learned vector representation for a graph is the frequency of a certain type of substructure. They encounter high computational cost for counting the occurrence of predefined substructures. The learned vector representations are very sparse, which prohibit the use of inner products. Moreover, the learned vector representations are not in a smooth space since the values can only be integers. The state-of-the-art approaches tackle the challenges by changing kernel functions instead of producing better vector representations. They can only produce kernel matrices for kernel-based methods and not compatible with methods requiring vector representations. Effectively learning smooth vector representations for graphs of various structures and sizes remains a challenging task. Motivated by the recent advances in deep autoencoders, in this paper, we explore the capability of autoencoder on learning representations for graphs. Unlike videos or images, the graphs are usually of various sizes and are not readily prepared for autoencoder. Therefore, a novel framework, namely discriminative graph autoencoder (DGA), is proposed to learn low-dimensional vector representations for graphs. The algorithm decomposes the large graphs into small subgraphs, from which the structural information is sampled. The DGA produces smooth and informative vector representations of graphs efficiently while preserving the discriminative information according to their labels. Extensive experiments have been conducted to evaluate DGA. The experimental results demonstrate the efficiency and effectiveness of DGA comparing with traditional and state-of-the-art approaches on various real-world datasets and applications, e.g., classification and visualization.**

***Index Terms*—autoencoder, graph, kernel**

I. Introduction

Graphs are widely used to represent macrostructures of relational instances, such as airline networks, publication connections, and social communications [1]. Beyond the single graph setting, multiple microstructures are also ubiquitous in various real-world data such as protein graphs, molecular expressions and control flows. In such cases, each data instance is a graph instead of a node resulting in higher complexity and difficulty in analysis and applications. Graph representation learning aims at deriving informative low-dimensional representations to prepare graphs for a variety of graph mining tasks such as graph classification [2] and clustering [3].

A widely used approach to graph representation learning is the graph kernel, which measures the similarity between graphs with vector inner product [4]. The key idea of graph kernels is to predefine a set of representative substructures and count their frequency in the graphs, known as frequency-based methods. There are several inherent limitations of these approaches. First, graph kernels are computationally expensive. The complexity of identifying and counting representative substructures is often high [5]. Even the cost of counting some simple substructures other than subgraphs (e.g., shortest paths) [6] is still $O(\sum v_i)$ in producing a vector representation $\mathbf{v} = (v_1, v_2, ..., v_n)$ for one graph. Second, the vector representations are sparse because a graph can only contain limited types of substructures. The sparse representations lead to an unsatisfactory performance of similarity measurement which is critically influential in many data mining tasks such as classification and clustering [7]. Third, the vector space of the learned representations is discrete. Values in the learned representations can only be integers. Such rigid representations usually negatively affect the applications, especially visualization [8]. Some recent efforts have been devoted to improve the performance by varying the kernel functions [1], [7], [9]–[11]. However, these methods only provide pairwise similarity measurements but not vector representations, which is essential in many downstream applications.

Autoencoder has been successfully applied to many real-world applications such as image representations learning [12], machine translation [13] and video representation learning [14]. The success of autoencoder motivates us to explore its nice properties which could be a better fit for our problem. First, the values in the learned representations are continuous. The information is more fine-grained than that in the raw integer representations. Second, the produced representation length is flexible. The user could define a proper length of the learned representation to avoid the sparsity problem [15]. Third, it is efficient by avoiding the complicated feature engineering process. In this paper, we propose to investigate whether a novel computational framework based on autoencoder could better tackle the challenges in learning graph representations.

This is a nontrivial task to design an autoencoder for learning graph representations because of the following reasons. First, unlike image or video data, graphs are not readily prepared for an autoencoder which requires a vector input, while graphs are structural data. Second, graphs are of various sizes. The architecture of the autoencoder needs to be modified to take different sizes of input. Third, graphs have hierarchical information. For example, computer programs have subroutines as subgraphs; and chemical compounds have functional groups as subgraphs. To take full advantage of hierarchical

978-1-5386-9125-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICBK.2018.00033

information, the autoencoder needs to consider both overview and details of the graphs.

In addition, the discriminative information in the graphs could be important. Graph data usually have valuable class label information with them. For example, computer programs may be labeled as malicious or benign; and chemical compounds may be labeled as acid or alkaline. The labels make the representations more meaningful for the downstream applications, e.g., make classification more accurate or make the visualization more illustrative. While discriminative information could be useful in learning graph representations, it is simply ignored in many related work [16]. Thus we also propose to study how the discriminative information could be naturally embedded in the learned graph representation.

To tackle the above challenges, in this paper, we study the problem of learning graph representations. We aim at answering the following questions. (1) How to efficiently and effectively learn vector representations from graphs? (2) How to incorporate the discriminative information in the graphs into the learned representation based on class labels? By investigating these questions, a novel method to learning graph vector representations is proposed, namely Discriminative Graph Autoencoder (DGA). We summarize the main contributions of this paper as follows:

- A novel graph representation learning method DGA is proposed, which is able to efficiently and effectively learn a smooth vector representation.
- Present an autoencoder architecture to leverage the discriminative information in graphs based on class labels.
- Validate DGA effectiveness and efficiency through classification and visualization on real-world datasets.

II. Related Work

We summarize the related work into two categories: graph kernels and neural network based approaches.

A. Graph Kernel-Based Approaches

Graph kernels extend kernel-based methods on graphs. Most of the work maps graphs to vector space by counting the presence of specific substructures, which could be considered as frequency-based methods.

They use the presence of a certain set of sub-structures as the features of a graph. There are three main approaches for the state-of-art methods, namely limited-sized subgraphs [17]–[23], graph kernels based on subtree patterns [4], [24], and graph kernels based on walks and paths [6], [25], [26].

Some works improve traditional graph kernels and their performance on downstream tasks, such as solving the problem of the dominance of certain dimensions of the learned vector representations [8], find discriminative subgraphs as substructures to learn better vector representations for classification tasks [16], [27]. Besides, Kong et al. [28] try to reduce the labeling cost for graph data for classification tasks. Saigo et al. [29] proposed a method to collect informative patterns progressively through mathematical programming.

Inevitably, these solutions all suffer from the following problems. First, the complexity is high. For the subgraph based methods, it is extremely expansive even to calculate the number of occurrences of a subgraph in a given graph. For other methods, the complexity is at least $O(\sum v_i)$, which is the sum of all the values in the vector representation, where the learned vector representation is $V = [v_1, v_2, \ldots, v_n]$. Second, the vector representation is sparse. Even two similar graphs may not share many non-zero dimensions in their vector representations, which significantly degraded the performance of the downstream tasks. Third, the values in the vector representations are discrete values, which would result in a nonsmooth vector space. The vector representations would be rigid and lose more information in the original graph. The effects of such rigidity are shown in the experiments.

Besides the traditional frequency-based approaches, some state-of-the-art work aims to solve the problems above. Yanardag et al. [7] proposed a method to measure the similarity between the substructures selected. It is more accurate in measuring the similarity values. Based on this work, Narayanan et al. [1] proposed a new way to learn vector presentations for subgraphs to better solve the sparsity problem. In addition, some other state-of-the-art work [9]–[11] solve the sparsity and discrete problem by changing the kernel function. However, as we mentioned before, they can solve the sparsity problem but cannot produce vector representation. It is compatible with kernel-based methods like SVM, but not compatible with other methods which require vector representations of the input, such as neural networks.

B. Deep Neural Network Based Approaches

Deep learning has been widely applied to feature extraction. Work has been done on network data [30], image data and text data [31]. The embedding and feature-learning techniques all try to map one form of data into the latent space, so that every data instance is represented in a vector representation preserving its original properties, which is much easier for further learning or processing than the original form of the data. Instead of text, image or a node in a network as a data instance, we have graphs as data pieces. However, only a few works focus on graph representations.

Duvenaud et al. [32] also proposed a method for running convolutional neural networks on molecular data in graph form. It generalizes standard molecular feature extraction methods based on circular fingerprints. Scarselli et al. [33] proposed a graph neural network which used recurrent neural networks for graph data. It maps the nodes of a graph and one of its node to the Euclidean space which can process various types of graphs. It is useful for rooted graphs since a root node has to be selected for the learning process. Li et al. [34] modified the graph neural network to use gated recurrent units and modern optimization techniques. These two works only did unsupervised learning which did not take the valuable labeling data into consideration.

The modern convolutional neural networks and recurrent neural networks are also involved in the graph similarity

measurement problem. Niepert et al. [5] proposed a method for preprocessing the graph data to be in the input format of a convolutional neural network. Their emphasis is placed on fast formatting graphs into suitable inputs of the neural network, which is similar to our graph sampling method. It is used as the baseline method for efficiency evaluation in the experiment part. More details are introduced in the experiment section.

III. Problem Statement

Notations and the mathematical definition of the core problem to solve are presented as follows.

1) Notations: Given a set of graphs $\mathbb{G} = \{G_1, ..., G_n\}$ where n is the number of graphs, each graph is denoted as $G = (V, E) \in \mathbb{G}$, where $V = \{v_1, ..., v_{|V|}\}$ denotes its vertex set and $E = \{e_{uv} | \text{if an edge connecting } u \text{ and } v \text{ exists}\}$ is its edge set. $l(e)$ and $l(v)$ denote the labels of edge e and vertex v, respectively. $p(v)$ denotes the index of v in the canonical permutation of vertices in G. $Y = \{y_1, y_2, \ldots, y_n\}$ denotes the graph level label of G. A_G is the adjacency matrix of G.

$$A_G(u, v) = \begin{cases} l(e), & \text{if } e_{uv} \in E \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$\bigoplus$ denotes an aggregated concatenation operation on a set of elements. For example, $\bigoplus_{x \in X} x$ denotes a long vector, matrix, or tensor, which is the result of concatenating all the elements in X. [**a** **b**] denotes the concatenate **a** and **b**. $vec(\cdot)$ is the vectorization (flatten) function for matrix, it concatenate each column of the matrix to become a column vector. $\circ$ denotes the function composition operation.

Based on the notations described above, we formally define our problem as follows: *Given a set of graphs $\mathbb{G}$ and their labels Y, the goal is to map each graph $G_i \in \mathbb{G}$ to a d-dimensional vector representation $\boldsymbol{h}_i \in \mathbb{R}^d$, i.e., Learning a mapping function $\varphi : \{G, y\} \rightarrow \boldsymbol{h}$ which produces informative representations of each graph G while preserving the discriminative information according to y.*

IV. Discriminative Graph Autoencoder

In this section, the proposed method Discriminative Graph Autoencoder (DGA) is introduced to deal with the previously mentioned challenges. While some recent work [1], [7] has put their focus on addressing the sparsity problem, they can only provide a pairwise similarity matrix for all graph pairs instead of a vector representation of each graph. Applications are thus constrained to kernel-based algorithms like SVM instead of general data mining algorithms, e.g. neural networks, nor any visualization can be done, which requires low-dimensional vector representations.

Our approach tackles the sparsity problem and produces vector representations simultaneously. The key ideas of DGA are shown in Figure 1. Given an input graph G to be encoded, we first decompose it into several subgraphs. These subgraphs and the adjacency matrix of G are then inputted into an encoder network. The output of the encoder corresponds to the vector representation of G we intend to learn, i.e., $\varphi(G)$. To preserve the graph structures and the discriminative

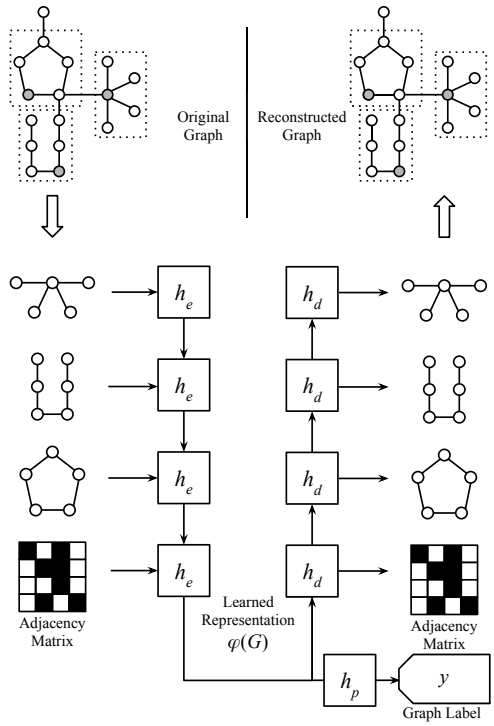


Fig. 1: Discriminative Graph Autoencoder Architecture

information of labels, $\varphi(G)$ is used for predicting the graph label y of G and is further input into a decoder network to reconstruct the original input subgraphs and adjacency matrix. Equipped with this model architecture, the mapping function $\varphi(\cdot)$ could be decomposed into two functions as follows:

$$\varphi = h_e \circ F. \tag{2}$$

where $F(\cdot)$ denotes the mapping function for graph sampling and $h_e(\cdot)$ represents the encoding function.

Graph sampling is an effective way to collect information from a graph. The graph kernels only seek predefined substructures and use their frequencies of occurrence to represent a graph. Graphs that only contain few of the predefined substructures may obtain sparse representations, making the performance of similarity measurement less effective. In addition, some graphs may contain substructures beyond those predefined substructures since the list is not comprehensive, thus the corresponding information cannot be captured by the representations. Our key idea of the graph sampling function $F(G)$ is to use arbitrary subgraphs found in each graph to represent the graphs instead of searching for certain predefined types of substructures, which preserves more structural information comparing to frequency-based approaches. For each input graph, a certain number of subgraphs are extracted around the selected vertices in the graph, which are combined with the adjacency matrix of the entire graph to prepare the input to autoencoder.

The autoencoding is used to capture the structural information from the sampled inputs for representation learning.

It can learn a smooth representation and avoid the sparsity problem. To capture discriminative information in the graph autoencoder, the key idea of the proposed algorithm is to let the autoencoder reconstruct the input and predict the graph label y_i from the learned representation $\varphi(G_i)$ at the same time. It is achieved by defining a new objective function instead of using the original autoencoder objective function which purely minimizing the reconstruction error between the input and output.

A. Graph Sampling

Our goal of graph sampling is to sample the raw structural data from the graph G and pack the information in a matrix $F(G)$ with a predefined shape. A natural way of graph sampling is to use adjacency matrices which may contain structural information of the graphs. However, two problems prevent us from using this solution. First, adjacency matrices are usually of different sizes. Take data flow graphs of computer programs as an example. Different programs may contain different numbers of variables corresponding to different numbers of vertices, resulting in different sizes of adjacency matrices. Second, adjacency matrices are vulnerable to changes in the vertex order. Two graphs of the same structure may have significant differences in their adjacency matrices if their vertices are in different orders.

To tackle the challenges, we propose a graph sampling method that uses subgraphs together with the adjacency matrix to present the information in a graph as shown in Equation 3.

$$F(G) = \Big[\bigoplus_{v \in C} (f \circ g)(v) \quad vec(A_G) \Big]^T, \tag{3}$$

where $\bigoplus$ is the concatenation operation on all the vectors in a set, $f(\cdot)$ is the vectorization operation on a single subgraph, and $g(\cdot)$ is the subgraph extraction operation around the given vertex $v \in C$, C is a specially selected subset of vertex set V, $vec(\cdot)$ is the matrix vectorization function, A_G is the adjacency matrix of graph G.

$F(G)$ is a matrix consists of multiple vectors. Each of the vectors encodes the information of a vectorized subgraph except the last vector, which encodes the information of the truncated adjacency matrix of G. A_G is part of $F(G)$ because the autoencoder needs to have an overview of the entire graph and the relationship between the subgraphs. It is also essential for reconstructing the entire graph after decoding.

$F(\cdot)$ is decomposed into the selection of central vertices C, subgraph extraction $g(\cdot)$, and vectorization function $f(\cdot)$, which are introduced in the following sections.

1) Vertex Selection: A set of vertices C is selected as the central vertices for subgraph extraction. Before the selection, it is necessary to put an order to the information in the graph which is essential for the following encoding part. Even two similar graphs if their vertices are arranged in a different order, the adjacency matrices could be not similar at all. The goal of sorting the vertices is to decrease the difference in the adjacency matrices of the graphs and subgraphs when the graphs are similar to each other, and increase the difference in the adjacency matrices of the graphs and subgraphs when they are not similar. The goal is achieved by canonical labeling [35]. After the canonical labeling, it is easier to tell whether graphs are similar or not from their adjacency matrices. Canonical labeling gives each vertex v in the graph $G = \{V, E\}$ a unique index $p(v) \in \{x | 1 \leq x \leq |V|\}$, $p(v_i) = p(v_j)$ if and only if $i = j$. The indices are the ranks of vertices sorted in canonical order. It is an indication of relations between vertices in two isomorphic graphs. For example, G and $G' = \{V', E'\}$ are isomorphic, $v_i \in V$ should be identical to $v'_j \in V'$ if $p(v_i) = p'(v'_j)$.

BLISS [36] is one of the most famous algorithms in graph isomorphism. We follow their idea to index the vertices in the graph according to their structural information of the graphs without the label information $l(v)$ or $l(e)$, which makes our method more general. Some graphs already have clear orders of the vertices within the graphs, for which the indexing process is not necessary.

With the label $p(v)$ we are able to select the central vertices set C according to Equation 4.

$$C = \{v | p(v) = ax + 1, 0 \leq x \leq s - 1, x \in \mathbb{Z}\}, \tag{4}$$

$$\begin{aligned} a = \underset{i \in \mathbb{Z}}{\operatorname{argmax}} (s-1) * i + 1, \\ \text{s.t.} (s-1) * i + 1 \leq |V|, \end{aligned} \tag{5}$$

where a is the gap between two selected vertices in the sequence of vertices sorted by their $p(v)$, s is the user-defined size of central vertices set C. The vertices are selected with a common interval of a. The goal is to evenly distribute the selected vertices in the graph. In Equation 5, $(s-1) * i + 1$ is the index of the last vertex in C. a is the interval between the indices of the selected vertices, so that the vertices are evenly separated. It is selected to maximize the index of the last vertex in C to ensure the indices of the selected vertices are distributed in V.

2) Subgraph Extraction: Subgraph extraction function $g(\cdot)$ is defined to extract a subgraph around a given vertex.

In Algorithm 1, it shows how one subgraph is extracted from G with a selected vertex as $central$ and a user-defined size $|V_{g(v)}| = r$, where $V_{g(v)}$ is the vertex set of the extracted subgraph. It is similar to a breadth-first search (BFS) with a priority queue optimization. The search starts from the vertex $central$ and ends when the number of vertices reaches the subgraph size limit r. The priority queue is always able to select the next vertex by comparing their $p(v)$ value, which can always break the tie between vertices. From line 2 to 5, the queue is initialized with a single object $central$ in it. From line 6 to 15, the subgraph keeps expanding until reaches r vertices. From line 7 to 8, the priority queue with vertex comparator pops out the best vertex candidate u and adds it to the subgraph according to the following rules. First, it selects the vertices with the shortest distance from the $central$. Second, it selects the v which equals to $\operatorname{argmin}_{v \in V} p(v)$ to further break the ties. From line 9 to 14, it pushes the neighbors of u into the priority queue as potential candidates for subgraph expanding.

Algorithm 1 Subgraph Extraction

Require: $G, central, r$
Ensure: R

```
function SUBGRAPH EXTRACTION(G, central, r)
    queue ← PriorityQueue
    queue.comparator ← Vertex Comparator
    queue ← {central}
    visited(central) ← TRUE
    while |R| < r do
        u ← queue.pop()
        R.add(u)
        for v do in u's neighbour
            if not visited v then
                queue.push(v)
                visited(v) ← TRUE
            end if
        end for
    end while
    return subgraph(R)
end function
```

3) Vectorization: As shown in Equation 6, the vectorization function $f(\cdot)$ takes a subgraph as input and put its information into a compatible form with the autoencoder h_e.

$$f(g) = \Big[vec(A_g)^T \quad (\bigoplus_{v \in V_g} l(v))^T\Big]^T, \tag{6}$$

where g is the input graph, A_g is the adjacency matrix of g, $\bigoplus$ is the concatenate operation on a set of elements, V_g is the vertex set of g. Notably, the order of the vertices in g is first sorted according to $p(v)$ where $v \in V_g$. The adjacency matrix A_g and the concatenate operation is all sorted according to the vertex order. The final output of $f(g)$ is a matrix, the first part of which is the vectorized adjacency matrix A_g. The rest of the row is the sequence of the labels of the vertices in the subgraph. Therefore, the size of the final output is $|V_g| \times |V_g| + |V_g|$. It contains all the information we need for the subgraph and ready to be encoded.

B. Autoencoding

A straightforward architecture for the autoencoder is multilayer perceptron [37], However, it is not flexible with different sizes of input and cannot leverage discriminative information. We use it as one of our baseline methods in the experiments. Thus, we proposed a novel architecture with a separate branch to leverage the discriminative information of the graph labels. Motivated by the "sequence to sequence learning model" [38], two LSTM [39] networks are used to better fit the various sizes of the graphs.

As shown in Figure 1, the discriminative autoencoder consists of the encoder $h_e(\cdot)$, decoder $h_d(\cdot)$, and predictor $h_p(\cdot)$. After graph sampling, the vectorized subgraphs of graph G is input into the encoder h_e. The encoder produces a low dimensional smooth vector representation of G denoted as $\varphi(G)$. $\varphi(G)$ is input to the predictor h_p to produce a label prediction for G denoted as $\hat{y}$. It is also input to h_d to reconstruct the original input graph. The two branches starting from the learned representation corresponds to two terms, $\mathcal{L}_1$ and $\mathcal{L}_2$, in the objective function,. $\mathcal{L}_1$ is defined for minimizing the autoencoder reconstruct error for the information in the input graph G. $\mathcal{L}_2$ is defined for minimizing the error of predicting the label of the input graph G from the learned representation $\varphi(G)$.

1) Encoding: For the encoder LSTM network, the vectorized information $F(G)$ of a graph of shape $(s+1,\ r*(r+1))$ is input to LSTM in $s+1$ steps, where $s = |C|$ is the number of subgraphs extracted, $r = |V_g(v)|$ is the size of each graph. Then, among the $s+1$ output of the encoder LSTM network, only the last one is collected. This collected output is the learned representation $\varphi(G)$.

2) Decoding: $\varphi(G)$ is used by the decoder LSTM to reconstruct the input. The decoder LSTM network uses $\varphi(G)$ as the input of the first step. For the rest s steps of the LSTM decoder, only padding vectors filled with ones are used as input. So there are total $s+1$ steps for the input to the decoder LSTM. To reconstruct the $s+1$ subgraphs input to the encoder, all $s+1$ outputs are collected from the decoder.

The non-discriminative graph autoencoder is optimized by minimizing the following objective function:

$$\mathcal{L}_1(\theta) = -\frac{1}{n}\sum_{i=1}^{n} L((h_d \circ h_e \circ F)(G), F(G)) \tag{7}$$

where $L(\cdot,\cdot)$ is the categorical cross-entropy. θ is the parameter set of the autoencoder. The output subgraphs should be in reverse order to ease the optimization of the autoencoder [13].

With the decoded adjacency matrix and the information of the subgraphs. The graph can be reconstructed in its original space. Apply the graph sampling procedure on the adjacency matrix to map the vertices to the vertices in the subgraphs, so that the reconstructed vertex labels in the subgraphs can be mapped to the graph.

3) Discriminative Information: Besides the structural information of the input graph G, the label information y could also be used to increase the rate of discriminative information in the learned representations among graphs.

To leverage discriminative information, the learned representation $\varphi(G)$ is used as shown in Equation 8.

$$\mathcal{L}_2(\theta) = -\frac{1}{n}\sum_{i=1}^{n} L((h_p \circ h_e \circ F)(G_i), \mathbf{y}_i), \tag{8}$$

where $\mathbf{y}$ is the binarized label of the graphs. $\mathbf{y}_{i,j} = 1$ if $y_i = j$, otherwise $\mathbf{y}_{i,j} = 0$, θ is the parameter set of the discriminative autoencoder, $h_p(\cdot)$ is a single-layer perceptron predictor use softmax as activation function, $L(\cdot,\cdot)$ is categorical cross-entropy. Optimizing against this loss function, it would force the autoencoder to focus on the information that can distinguish one class of graphs from the other classes, instead of treating each class of graphs equally.

TABLE I: Datasets Statistics

Datasets \ Statistics	$\|G\|$	$\overline{\|V\|}$	$\overline{\|E\|}$	Class
MUTAG	188	17	39	2
NCI1	4110	29	64	2
PTC	344	25	51	2
PROTEIN	1113	39	145	2

TABLE II: Time For Graph Sampling

Datasets \ Methods	PCSN	DGA
MUTAG	00.83s	00.75s
NCI1	25.15s	21.36s
PTC	01.74s	01.61s
PROTEIN	10.60s	08.70s

4) DGA Objective Function: The autoencoder is optimized against the sum of two loss functions for representation learning, which is balanced by the parameter λ shown in Equation 9. The overall objective function requires the learned representations not only contain as much information in the extracted subgraphs, but also incorporate the discriminative information in graph labels.

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} L((h_d \circ h_e \circ F)(G_i), F(G_i)) + \lambda L((h_p \circ h_e \circ F)(G_i), \mathbf{y}_i). \quad (9)$$

C. Time Complexity

The time complexity of Discriminative Graph Autoencoder is analyzed as follows. The complexity of the discriminative autoencoder depends on many complex parameters. Moreover, it is not the bottleneck of the proposed method. Therefore, the time complexity analysis focuses on the graph sampling part. The complexity of the vertex indexing is $O(|V|)$, which is the complexity of BLISS algorithm. The complexity for a simple breadth-first search to extract a subgraph of size r is $O(r)$. However, a priority queue is used for finding the best vertices which put additional complexity to the method. The time complexity of each push or pop operation of the priority queue is $O(\log r)$. There are r operations in total, where r is the size of the subgraph. Therefore, the complexity for extracting one subgraph is $O(r \log r)$. Vectorize one subgraph would cost $O(r^2)$, which is the size of the output vector $f(g)$. The total complexity of $f(g(v))$ is $O(r^2 + r \log r) = O(r^2)$. So the complexity for generating s subgraphs is $O(r^2 s)$. Thus, the overall complexity of the graph sampling is $O(|V| + r^2 s)$.

V. Experiments

We empirically evaluate the representations learned from the proposed model DGA on two different tasks, i.e., classification and visualization. Three questions are mainly analyzed: (1) How efficient is DGA in learning graph representations? (2) How effective are the learned representations for graph classification? (3) How informative is the visualization of the learned vector representations in low dimensional space?

A. Experimental Setup

1) Datasets: The datasets used are benchmark datasets for graph classification, which consist of MUTAG, NCI1, PROTEIN, and PTC. MUTAG [40], NCI1 [41], and PTC [42] are datasets of chemical compounds. PROTEIN [43] is a dataset of the protein structures. The number of examples for each class is balanced. The statistics information of these datasets are shown in Table I, where $|G|$ is the number of graphs in each dataset, $\overline{|V|}$ and $\overline{|E|}$ denote the average number of vertices and edges. $Class$ is the number of different classes in the labels of the graphs in the datasets.

2) Baselines: Four different types of baseline methods are used for comparison as follows. First, Graphlet Kernels (GK) [23], Shortest-Path graph kernels (SP) [6], fast subtree kernels (WL) [4] are used as traditional baselines. They are frequency-based methods of counting subgraphs, vertex pairs and subtrees respectively. Second, the advanced baselines are Deep Graph Kernels [7] and PCSN [5]. Deep Graph Kernels are three graph kernels derived from the three traditional approaches above, namely DGK, DSP, and DWL. Third, we also implemented a naive approach of Multi-Layer Perceptron autoencoder (MLP), which only uses the adjacency matrix as input. Fourth, to show the effectiveness of discriminative information, a non-discriminative graph autoencoder (GA) is implemented, which only uses $\mathcal{L}_1$ as the loss function.

3) Parameter Setting: For three traditional baseline methods, we follow the parameter setting in the original paper. For deep graph kernels, the length of the subgraph embedding is set to 128. For PCSN, we set $w = k = 12$, which are the sizes of the sampling information in the graphs. The following parameters are set to DGA for the experiments. 1. Number of subgraphs selected $s = 12$. 2. Size of each subgraph $r = 12$. 3. The length of the final embedding is 128. 4. $\lambda = 1$ in the loss function $\mathcal{L}(\theta)$. These parameters are set based on cross-validation, the parameter analysis.

B. Efficiency

The efficiency of graph sampling is evaluated and compared with the state-of-the-art method PCSN [5] on four datasets shown in Table II. We follow similar experimental settings in [5]. Since the complexity of LSTM autoencoder is much lower than the graph sampling process, the evaluation is mainly targeting the efficiency of graph sampling, which is potentially the bottleneck of the efficiency of the entire process. The number of subgraphs and the size of the subgraphs are set to 12 for both of the methods.

The total time for PCSN and DGA in graph sampling and network input generating is shown in Table II. The results show that Discriminative Graph Autoencoder performs slightly better than PCSN on all benchmark datasets. Our proposed DGA method is very efficient since it is very careful in using the expensive traditional graph isomorphism algorithm as subroutines. It uses vertex indexing only once and without considering the labels. The subgraph sampling is also boosted with priority queue optimization.

TABLE III: Classification Accuracy with Standard Deviation

	MUTAG	NCI1	PTC	PROTEIN
GK	81.66±2.11	62.28±0.29	57.26±1.41	71.67±0.55
SP	85.22±2.43	73.00±0.24	58.24±2.44	75.07±0.54
WL	83.48±6.51	80.13±0.50	56.97±2.01	72.92±0.56
DGK	82.66±2.11	62.48±0.25	57.32±1.13	71.68±0.50
DSP	87.44±2.72	73.55±0.51	60.08±2.55	75.68±0.54
DWL	82.94±2.68	**80.31±0.46**	59.17±1.56	73.30±0.82
PCSN	92.63± 4.21	78.59± 1.89	60.00± 4.82	75.89± 2.76
MLP	76.05±9.71	67.05±2.39	59.31±4.96	73.76±3.54
GA	92.57±5.84	72.74±1.75	70.93±3.89	77.45±3.29
DGA	**93.63±5.21**	74.55±1.46	**71.24±4.60**	**77.71±2.37**

C. Graph Classification

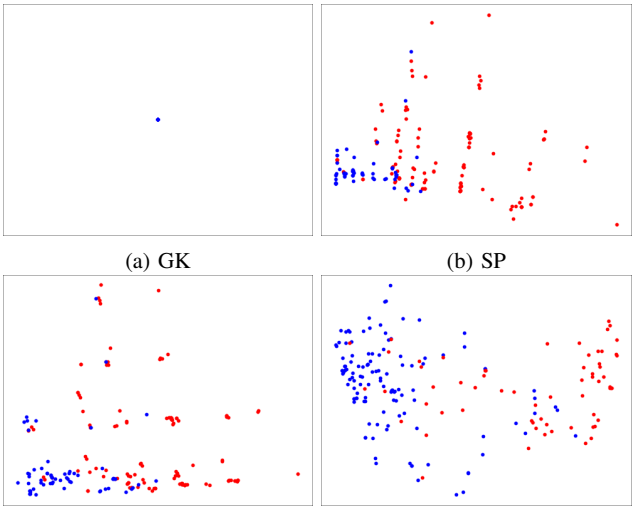
Graph classification aims at assigning unlabeled graphs into target categories based on available labeled training graphs. In this section, we evaluate DGA on the four datasets stated above with the graph classification task.

Experimental Setting For each of the algorithms to be tested, the following steps are conducted to produce the results. First, the datasets are divided to conduct a 10-fold cross-validation. Second, during each fold, each of the algorithms is trained on the corresponding training dataset and then used to convert all the testing graphs into vector representations. Finally, an SVM is trained on the converted training dataset (i.e., labeled vector representations) and tested on the converted testing dataset.

Results Table III shows the accuracy of all the experiments, which is defined as the quotient of the number of correctly classified instances divided by the total number of instances in the testing dataset. From the results, we can see that our method has the highest accuracy on most of the datasets. DGA has a significant increase in accuracy in PTC. By comparing the GA and DGA, we can see the power of discriminative information in the representations. DGA has an extra loss function of $\mathcal{L}_2$ to keep the representation focusing more on the discriminative information in the graphs. By this loss function, the accuracy rises on four of the datasets.

D. Graph Visualization

One useful application of graph representation learning is to produce meaningful visualizations that layout graphs on a low-dimensional space. The visualization performance is also an indicator of the quality of the representations. DGA is compared with three traditional baselines, GK, SP, and WL, on MUTAG dataset, and achieved a significant increase in this task. Deep graph kernels and PCSN are not selected in this experiment as they are not capable of generating vector representations of graphs. To achieve the best performance for the baselines during the experiment, they are first trained to produce high dimensional representations. Then, PCA is used for reducing the dimensionality of these representations to 2. The proposed method DGA is directly trained to produce a 2-D representation. The representations learned from all four methods are visualized in a 2-D space shown in Figure 2.



(a) GK (b) SP

(c) WL (d) DGA

Fig. 2: Visualization of Learned Representations

Each point in the figure is the projection of one representation learned from a graph in the dataset. The colors of the points correspond to the class labels $L(G)$ of the graphs. In Figure 2a, the GK cannot provide valid visualization of the learned representations since all instances are mapped to one point. This is mainly because the learned representations are long and extremely sparse, which results in a concentration effect after PCA. SP and WL methods have better performances in visualization which are shown in Figure 2b and Figure 2c. However, it clearly shows striped or grid patterns in the distributions of points. The main reason is the vector space of the learned representations is not smooth. The representations are rigid integer coordinations representing head counts for substructures and cannot represent the fine-grained information in graphs. The representations are so rigid and stick to the integer coordinated grids in the hyperspace that even after the PCA dimensionality reduction the grids can still be seen. The visualization results using the representations generated from DGA is shown in Figure 2d. Comparing to other methods, it has three advantages: (1) it can directly produce valid low-dimensional data representation; (2) the vector space of the learned representations is smooth, which avoids the damage caused by the rigidity in the representations; (3) it can use the discriminative information to better visualize the differences between different classes of graphs.

VI. Conclusion and Future Work

Learning vector representations from graphs can derive informative low-dimensional representations to benefit many data mining tasks on graph data. To this end, we propose discriminative graph autoencoder (DGA) to learn smooth vector representations for graphs, which also leverages discriminative information based on the graph labels. Specifically, it samples subgraphs from each of the graphs and vectorizes them to feed to the discriminative autoencoder, and then the

autoencoder is optimized for two goals: reconstructing the subgraphs and predicting the labels. Experiments on real-world datasets demonstrate that DGA effectively and efficiently learns vector representations from graphs which performed well on classification and visualization tasks.

In the future, we will aim at specific downstream tasks or special types of graphs. For example, in neural architecture search (NAS), the meta-learning model needs to learn the relations between the computational graphs and the performances of the neural networks [44]. However, it is challenging to use graphs as the input to the meta-learning model. New graph representation learning strategies can be developed to convert the computational graphs of neural networks into vectors.

VII. Acknowledgements

This work is, in part, supported by DARPA (#FA8750-17-2-0116) and NSF (#IIS-1718840 and #IIS-1750074).

References

[1] A. Narayanan, M. Chandramohan *et al.*, "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *arXiv preprint arXiv:1606.08928*, 2016.

[2] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *Advances in neural information processing systems*, 2005.

[3] K. Riesen and H. Bunke, *Graph classification and clustering based on vector space embedding*. World Scientific, 2010.

[4] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *Advances in neural information processing systems*, 2009.

[5] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016.

[6] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005.

[7] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[8] ——, "A structural smoothing framework for robust graph comparison," in *Advances in neural information processing systems*, 2015.

[9] F. Costa and K. De Grave, "Fast neighborhood subgraph pairwise distance kernel," in *International conference on machine learning*, 2010.

[10] F. Orsini, P. Frasconi, and L. De Raedt, "Graph invariant kernels," in *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence*, 2015.

[11] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, 2016.

[12] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval." in *ESANN*, 2011.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014.

[14] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, 2015.

[15] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.

[16] X. Yan, P. S. Yu, and J. Han, "Graph indexing based on discriminative frequent structure analysis," *TODS*, 2005.

[17] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proceedings of the 2002 SIAM International Conference on Data Mining*, 2002.

[18] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Subdue: Compression-based frequent pattern discovery in graph data," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, 2005.

[19] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008.

[20] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *Proceedings of the 2009 SIAM International Conference on Data Mining*, 2009.

[21] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010.

[22] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *International conference on machine learning*, 2005.

[23] N. Shervashidze, S. Vishwanathan *et al.*, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009.

[24] N. Shervashidze, P. Schweitzer *et al.*, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, 2011.

[25] L. Hermansson, F. D. Johansson, and O. Watanabe, "Generalized shortest path kernel on graphs," in *International Conference on Discovery Science*, 2015.

[26] B. L. Douglas, "The weisfeiler-lehman method and graph isomorphism testing," *arXiv preprint arXiv:1101.5211*, 2011.

[27] N. Jin, C. Young, and W. Wang, "Gaia: graph classification using evolutionary computation," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010.

[28] X. Kong, W. Fan, and P. S. Yu, "Dual active feature and sample selection for graph classification," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.

[29] H. Saigo, S. Nowozin *et al.*, "gboost: a mathematical programming approach to graph classification and regression," *Machine Learning*, 2009.

[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016.

[31] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[32] D. K. Duvenaud, D. Maclaurin *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015.

[33] F. Scarselli, M. Gori *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, 2009.

[34] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[35] L. Babai, "Graph isomorphism in quasipolynomial time," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016.

[36] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*, 2007.

[37] L. Deng, M. L. Seltzer *et al.*, "Binary coding of speech spectrograms using a deep auto-encoder," in *INTERSPEECH*, 2010.

[38] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *Journal of Symbolic Computation*, 2014.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[40] A. Debnath, d. C. R. Lopez *et al.*, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity." *Journal of medicinal chemistry*, 1991.

[41] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, 2008.

[42] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, 2003.

[43] K. M. Borgwardt, C. S. Ong *et al.*, "Protein function prediction via graph kernels," *Bioinformatics*, 2005.

[44] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," *arXiv preprint arXiv:1806.10282*, 2018.