UNIVERSITY OF OREGON

Graph Representation Learning and Graph Classification

by

Sara Riazi

in the

Department of Computer and Information Science

June 2017

Abstract

Many real-world problems are represented by using graphs. For example, given a graph of a chemical compound, we want do determine whether it causes a gene mutation or not. As another example, given a graph of a social network, we want to predict a potential friendship that does not exist but it is likely to appear soon. Many of these questions can be answered by using machine learning methods if we have vector representations of the inputs, which are either graphs or vertices, depending on the problem. A general approach to extracting such vectors is to learn a latent vector representation for the vertices or the entire graph such that these vectors can be used in machine learning tasks, such as training a classifier or a predictive model (Bengio et al., 2013). The learned vectors should reflect the graph's structure and its attributes, including vertex and edge attributes. For example, two adjacent vertices in a graph should have similar vector representations. The problem of learning a latent vector representation for graphs or vertices is called graph representation learning or graph embedding. In this document, we mainly focus on recent developments in graph representation learning in different settings and its connection to various problems, such as graph classification or graph clustering.

1 Introduction

Many real-world problems are represented as graphs or networks in different computational domains, such as bioinformatics (Borgwardt et al., 2005a; Baldi and Pollastri, 2003), chemical informatics (Ralaivola et al., 2005; Wale et al., 2008), vision (Shi and Malik, 2000; Felzenszwalb and Huttenlocher, 2004), or social networks analysis (Liben-Nowell and Kleinberg, 2007; Backstrom and Leskovec, 2011; Agrawal et al., 2013).

In many of these problems, the input data is in the form of graphs, and the task is to predict either node and edge labels or graph labels.

For example, chemical compounds are described as the interaction graph of the forming elements, and based on the structure of the graph, the functionality of the compounds differs. Given some known functionality, such as being toxic on nontoxic as the labels of chemical compounds, the problem is to predict the functionality of an unseen compound structure. This prediction can be seen as classifying each graph as being either toxic or nontoxic.

Another interesting problem is learning protein-protein interactions. In this problem, each node in the graph is a protein and the edges shows if two proteins interact. However, studying the interaction of two proteins is expensive in term of the cost of required resource. Therefore, the problem is to predict the presence of edges among proteins given some known edges, so bio researchers can study only the interaction of proteins that their algorithm predicts the existence of an edge between them. This problem can also be considered as an edge label classification problem by supposing a complete graph among protein and classifying each label as either present or not-present.

Community detection, especially in social analysis, is very popular for understanding the structure of the network and how it grows. This problem is similar to classifying each node as being member of a community or not. However, the general problem is unsupervised since none of the memberships nor the number of communities are know in advance, but in some cases we already know the label of some nodes and we want to predict the labels of the others.

One general approach that helps us to solve these problems is to represent the graph data as a low dimensional vector representation and then use linear or non-linear classification algorithms such as support vector machines (Bishop, 2006).

There exists different ways for finding a representation for graph data, such as extracting structural features from the graph and use them as a representation. Graph kernels are well-known approaches that use these structural properties for comparing two graphs.

The more recent approach is to use neural networks to learn a latent representation. In these approaches the neural network is responsible for extracting the structural features as set of latent parameters, which are not explicitly observable from the structure of graphs.

Learning node representation is the problem of indicating each vertex of the graph using a low dimensional vector such that the vectors of similar vertices are located near each others in the euclidean space. Therefore, these low dimensional representation can be used in different algorithms for graph clustering and classification. This low-dimensional can also be used for visualizing graph data either by learning a 2-dimensional vector representation or learning a low-dimensional vector representation and learn another mapping to 2-dimensional space. A well-known algorithm for the latter approach is t-SNE (Maaten and Hinton, 2008), which takes low-dimensional vectors as the input and maps them to a 2D space.

The goal of this document is to discuss different approaches for representing graphs or nodes in low-dimensional vector spaces, and the use of these representation for learning from graph data, especially in task of label predication for nodes, edges and graphs.

The rest of this document is organized as follows: in Section 2, we introduce the main approaches for learning representation learning of word sequences, which have inspired many graph representation learning algorithms. In Section 3, we discuss general graph representation algorithms, then in Section 4, we see how we can learn better representation if we jointly train the representation learning and the classification and clustering task.

Deep neural networks are widely used for representation learning, in Section 5, we explore how we can exploit deep neural networks for learning graph representations. In Section 6 and Section 7 we discuss the main approaches for graph representation learning when associated information is available for every vertex or when we have different node or edge types. Section 8 explores the problem of graph classification using graph kernel methods and neural networks, and finally, Section 9 concludes the document and discusses possible future directions.

2 Sequence Representation

The structure of graphs can be captured by sets of random walks starting at every vertex of the graph. Each of these random walks forms a sequence of vertices. Therefore, the algorithms for word representation that uses sequences of words (sentences) as the input can also be exploited for graph representation Perozzi et al. (2014). Word representation or word embedding is an important tool in language modeling Bengio et al. (2003), which helps algorithms to extract similar words. The idea is that given a corpus, similar words would appear within similar context. A context of a word is the set of surrounding words in the same sentence.

The basic word representation is a 2-gram or one-word context, in which we only care about the co-occurrence of a word and its context that includes only one word. Basically two words are similar if they appear within similar context more often. An N-graph is the generalization of 2-gram which focuses on the appearance of similar words in similar contexts that have more than one word. The most well-known word representation learning algorithm is Skip-gram (Mikolov et al., 2013b,a), which we discuss in more detail.

Given a corpus with n words, originally each word w is represented using an one-hot-encoding vector, which is an n-dimensional binary vector that has one entry for each word in the corpus. The one-hot vector representation of w has only one non-zero element located in the column corresponding to word w. The objective is to learn a d-dimensional vector representation for each word, such that $d \ll n$ and similar words have close vector representations.

Skip-gram measures the similarity of words based on their context. The context of a target word w is a window of words surrounding the target word, which is called the context words c. The objective function of Skip-gram is to maximize the probability of predicting the context words given target words:

$$\max \sum_{w \in \mathcal{V}} \sum_{c \in V_c} \log P(c|w), \tag{1}$$

where $\mathcal{V}$ is word vocabulary and and $\mathcal{V}_c$ is the context vocabulary, which may be considered to be equal to $\mathcal{V}$. Skip-gram estimates $P(c|w)$ using a softmax function:

$$P(c|w) = \frac{exp(\Phi(w)^T \Phi(c))}{\sum_{c' \in \mathcal{V}} exp(\Phi(w)^T \Phi(c'))}, \tag{2}$$

where $\Phi(.)$ is a function from vocabulary space to a d-dimensional vector representation.

However, because the size of the context vocabulary is often very large, computing the denominator in the above softmax is prohibitive. To overcome this obstacle hierarchical softmax (Morin and Bengio, 2005) and negative sampling (Mikolov et al., 2013c; Dyer, 2014) have been widely used.

The idea of hierarchical softmax is to group words into classes in order to reduce the summation. If we can predict the class of each word, then we only have to do the summation for the words belonging to that class, which reduces the required computation significantly. Morin and Bengio (2005) propose using hierarchical clustering, in which they form a binary tree of classes, and each intermediate node only predicates whether the word belong to the left or right sub-classes. They use softmax, for prediction at intermediate node:

$$P(b_l = 1|w) = \sigma(\Psi(b_l)^T \Phi(w)), \tag{3}$$

where $\Psi(.)$ is the vector representation of each intermediate node and $\sigma(.)$ is the sigmoid function: $\sigma(x) : 1/(1 + \exp(-x))$. Since the variable of the intermediate nodes are binary, we don't need to compute the normalization constant by simply selecting $P(b_l = 0|w) = 1 - P(b_l = 1|w)$. Using hierarchical softmax, Relation 2 can be computed using:

$$P(c|w) = \prod_{l=1}^{\lceil \log |\mathcal{V}| \rceil} P(b_l|w), \tag{4}$$

which needs evaluating $\lceil \log |\mathcal{V}| \rceil$ different softmax functions and is exponentially more efficient than computing Relation 2. For example, Figure 1 shows a factorization for computing $P(v_3|\Phi(v1))$ as $P(b1 = 0|\Phi(v1))P(b2 = 1|\Phi(v1))P(b5 = 0|\Phi(v1))$.

Mikolov et al. (2013c) introduce negative sampling as another way to deal with the computational complexity of the normalization constant of the softmax relation (Relation 2). Negative sampling penalizes the co-occurrence of random context words and the target words. Therefore, the objective function of skip-gram becomes the following:

$$\sigma(\Phi(c)^T \Phi(w)) + \sum_{i=1}^{k} E_{c' \sim P_D}[\log \sigma(-\Phi(w)^T \Phi(c'))], \tag{5}$$

where P_D is an empirical unigram distribution: $P_D(c) = \frac{\#(c)}{D}$.

Levy and Goldberg (2014) show that optimizing the above objective function is similar to factorization of matrix M, whose elements, M_{ij}, are shifted point-wise mutual information

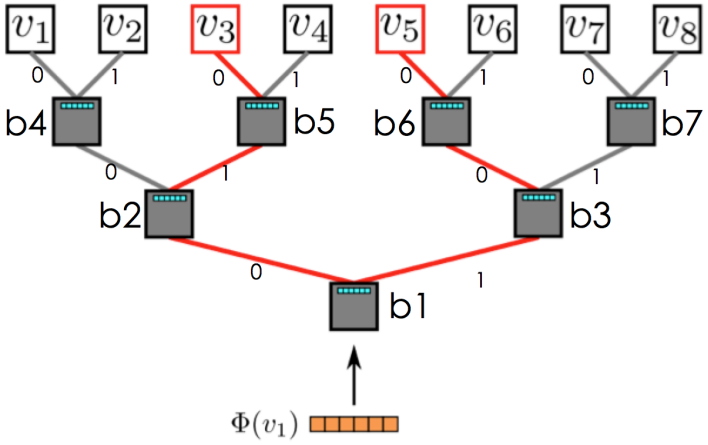

FIGURE 1: (Perozzi et al., 2014) Hierarchical softmax for computing $P(v_i|\Phi(v))$. Each intermediate node defines $P(b_l|\Phi(v))$, which be trained using logistic regression.

(PMI) of words and contexts: $M_{ij} = \log \frac{\#(w,c)|D|}{\#(w)\#(c)} - \log k$. To learn the representation using matrix factorization the goal is to reconstruct the matrix M as the linear product matrix $U \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $V \in \mathbb{R}^{|\mathcal{V}| \times d}$:

$$\min_{U,V} M - UV^T, \tag{6}$$

where there rows of U and V are the vector representations of target words and context words, respectively.

3 Learning Graph Representation

Similar to word representation, the goal of graph representation is to learn a low-dimensional vector for each vertex in the graph such that the vector representation carries the structural properties of the graph. Formally, for graph $G(\mathcal{V}, \mathcal{E})$ of vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, we want to learn a d-dimensional vector representation $\Phi(v)$ for each $v \in \mathcal{V}$ such that $d \ll |\mathcal{V}|$.

DeepWalk (Perozzi et al., 2014) suggests using a model similar to the Skip-gram model for learning $\Phi(v)$, which maps vertex v to its vector representation. DeepWalk relates each vertex to one word and the set of random walks on the graph G to the corpus. Using this relationship, DeepWalk successfully applies the Skip-gram model for learning the graph representation. DeepWalk generates a set of fixed-length random walks R_v starting at every vertex v of the graph. Then for every vertex v_j of random walk R_v, it considers the vertices surrounding v_j (in a window centered at v_j) as the context of v_j. Finally, the

representation vector of v_j, $\Phi(v_j)$, is calculated by optimizing the following relation:

$$\max_{\Phi} \sum_{i \in \{j-w,\cdots,j-1,j+1,\cdots j+w} \log P(v_i|\Phi(v_j)), \tag{7}$$

where the size of the window is $2w$.

DeepWalks uses hierarchical softmax to compute the probability of $P(v_i|\Phi(v_j))$.

Node2vec (Grover and Leskovec, 2016) is another successful representation learning approach for graphs which is similar to DeepWalk in term of objective function and using random walks, however it uses negative sampling instead of hierarchical softmax to overcome the intractability of Relation 2. Moreover, Node2vec defines the neighborhood of node v_j as its context, and introduce methods for extracting the neighborhood of a vertex. The main difference between the random walk exploration and neighborhood exploration is in introducing a search bias α, which controls selecting the next node to visit not only based on the current node, but also on the previous node. To select the next node to visit we need to sample from:

$$P(v_j = x|v_{j-1} = v, v_{j-2} = t) = \begin{cases} \frac{\alpha(t,x)w_{xv}}{Z} & \text{if (v,x)} \in E \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

where Z is the normalization constant, and α is defined based on the shortest path distant d_{tx} between node t and x as the following:

$$\alpha(t,x) = \begin{cases} \frac{1}{p} & \text{if } \mathrm{d}_{tx} = 0 \\ 1 & \text{if } \mathrm{d}_{tx} = 1 \\ \frac{1}{q} & \text{if } \mathrm{d}_{tx} = 2 \end{cases}, \tag{9}$$

where p and q are positive parameters that control neighborhood exploration. p controls how often the random walk revisits the previous node, and q controls how often the random walk explores nodes that are not immediate neighbors of the previous node. For example, for $p \gg 1$ and $q \ll 1$ results in random walks which are more likely emulating depth-firth search, while the random walks generated with $p > g \gg 1$ are more likely emulating breadth-first search.

Although DeepWalk and Node2vec are successful in learning graph representations, they mostly suffer from the fact that random walks only capture local structural properties of graphs; therefore, what they learn mostly depends on what random walks can capture.

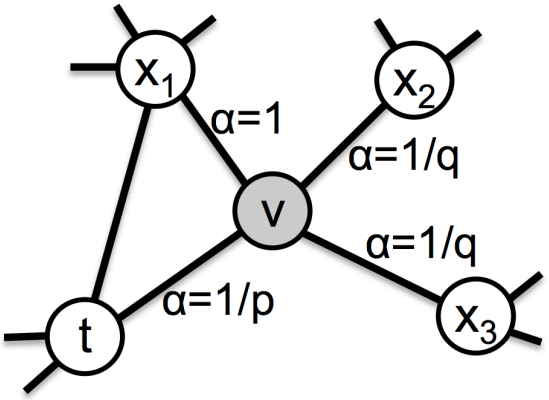

FIGURE 2: (Grover and Leskovec, 2016) Neighborhood exploration using search parameters p and q. The goal is to select the next node to visit given that the current and previous nodes are v and t, respectively.

Moreover, to learn the representation of large-scale graphs, they may need many random walks for each vertex which is prohibitive.

Tang et al. (2015b) address these problems by defining an objective function which directly depends on the structure of graph instead of relying on random walks for capturing the structure of input graphs. This objective function is based on the definition of proximity in graphs, which includes first-order proximity and second-order proximity.

First-order proximity is the pairwise similarity between two vertices v_i and v_j, defined as the joint probability distribution over both of them: $P(v_i, v_j) = \sigma(\Phi(v_i)^T \Phi(v_j))$.

Second-order proximity is the pairwise similarity between two vertices v_i and v_j that share similar context or neighborhood, and is defined using $P(v_i|v_j)$:

$$p(v_i|v_j) = \frac{\exp(\Phi(v_i)^T \Phi(v_j))}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_k)^T \Phi(v_j))} \tag{10}$$

Tang et al. (2015b) define empirical distributions based on the graph structure for both $P(v_i, v_j)$ and $P(v_i|v_j)$, and then minimize the distant between empirical distribution and the model defined as KL-divergence.

The empirical distribution for $P(v_i, v_j)$ is defined as $\frac{w_{ij}}{W}$, where W is the total weights of the edges in the graph and w_{ij} is the weight of the edge between vertex i and vertex j. Similarly, the empirical distribution for $P(v_i|v_j)$ is defined as $\frac{w_{ij}}{\sum_i w_{ij}}$.

Therefore, minimizing KL-divergence for these two models results in the objective functions O_1 and O_2 for the first-order and second-order proximity, respectively:

$$O_1 : \min_{\Phi} - \sum_{(i,j)\in\mathcal{E}} w_i j \log P(v_i, v_j)$$
$$O_2 : \min_{\Phi} - \sum_{(i,j)\in\mathcal{E}} w_i j \log P(v_i|v_j) \tag{11}$$

Tang et al. (2015b) experimentally show that optimizing either O_2 or $O_2 + O_1$ is learning a better representation comparing to DeepWalk.

3.1 Multi-Scale Representation

In general, Skip-gram based models only consider the linear interaction of vertices by considering the co-occurrence of context vertices and target vertices together. However, a random walk sequence captures multi-scale information from the graph. A random walk of length k can be considered as a sample from P^k, two samples from P^{k-1}, and so on, where P is the normalized transition matrix of the graph. Therefore, the amount of information about different scale is not balanced. To make a balanced multi-scale representation, Walklets Perozzi et al. (2016) extend the idea of DeepWalk by generating samples from different scales. Walklets construct skipped random walks, in which two adjacent vertices in the skipped random walk of scale k are $k-1$ vertex apart in the original random walk. Therefore, it can generate enough samples from all target scales, then use Skip-gram to learn a multi-scale representation for the graph.

Similar to Walklets, GraRep (Cao et al., 2015) learns multi-scale representation for a graph. For scale k, GraRep, factorizes P^k using SVD and find d-dimension representation, and finally concatenate all representation of different scales together.

Comparing to Walklets, GraRep using the whole P^k instead of samples from it, which makes it more computationally expensive. However, we can extract the representation of each scale easily from the representation learned by GraRep, which is not possible for the representation learned by Walklets.

Using matrix factorization for learning graph representation and random-walk based approaches are connected as Yang and Liu (2015) show that optimizing DeepWalk objective function is equal to factorizing matrix $M = \log \sum_1^t \frac{P^t}{t}$ following the formulation of Levy and Goldberg (2014) for relation of matrix factorization and Skip-gram.

4 Guided Representation Learning

The learned representation may be used for different optimizations for other tasks such as vertex clustering or classification. Jointly learning of representation and the target application results in a better-guided representation. Here, we discuss guided representation learning for both vertex classification and clustering.

4.1 Vertex Classification

Vertex classification is an important problem since many semi-supervised learning algorithms can be reduced to vertex classification (Zhu et al., 2005). The graph captures the similarity among the points, so every vertex represents one data point either labeled or unlabeled. This problem is also known as collective classification (Sen et al., 2008). Several methods have been proposed for collective classification such iterative classification (Sen et al., 2008). Label Propagation (Zhu and Ghahramani, 2002) is another well-known algorithm for vertex classification, in which the labels of the unlabeled points are decided based on the majority vote on the labels of the neighbors, and since many neighbors may be originally unlabeled; therefore, Label Propagation iteratively do the label assignment until it converges.

In general, in a semi-supervised setting, we are given the labels of a portion of data points, and the problem is to determine the labels of remaining unlabeled points. This setting is important since the number of labeled data is limited while we often access to considerable amount unlabeled data. The hypothesis of semi-supervised learning is to take the advantage of the unlabeled data to improve the performance of learning algorithms. However, it has been studied that an infinite number of unlabeled data is not always beneficial (Nadler et al., 2009).

Semi-supervised problems can be seen as the supervised problems with regularization on the graph of point similarities (Weston et al., 2012):

$$\min \sum_{i=1}^{N_l} l(y_i, f(x_i)) + \lambda \sum_{i,j=1}^{N_l+N_u} L(f(x_i), f(x_j), W_{ij}), \tag{12}$$

where N_l and N_u is the number of the labeled and unlabeled points, respectively, and W_{ij} is the weight of similarity of point i and point j. If we defined the regularization function

L as $W_{ij}\|f(x_i) - f(x_j)\|^2$ then $L = f^T \Delta f$, where $\Delta = D - W$ is the Laplacian of the similarity graph and D is a diagonal matrix $D_{ii} = \sum_j W_{ij}$.

Graph embedding has also been used for vertex classification (Tang et al., 2015b; Grover and Leskovec, 2016). In those setting the learned representation of labeled and unlabeled vertices is used as the input features in traditional classification algorithms such SVM. However, recently, Yang et al. (2016) show that we can get better classification performance if we jointly learn the graph embedding and train the classification algorithm. Yang et al. (2016) introduce a combined objective function for the joint learning:

$$\min -\frac{1}{N_l} \sum_{i}^{N_l} \log P(y_i|\mathbf{x}_i, \mathbf{e}_i) - \lambda \mathbb{E}_{(i,c,\gamma)} \log \sigma(\gamma \mathbf{w}_c^T \mathbf{e}_i), \tag{13}$$

where the first term is the supervised loss and the second term is the unsupervised term. In Relation 13, e_i and x_i are the representation and input feature of vertex i, respectively, and w_c is the representation of the context c. γ controls the negative samples such that if $\gamma = -1$, we randomly sample a context c from possible vertices. If $\gamma = 1$, we select the context based on the samples random walk similar to DeepWalk (Perozzi et al., 2014). Here, y_i is label of the labeled vertices. This setting is transductive, which means that it cannot classify an unseen point.

Yang et al. (2016) also give an inductive formulation for this setting by removing the dependence of the supervised part on embedding e_i. A classifier trained using inductive formulation can classify unseen data points.

4.2 Vertex Clustering

In graph clustering, the objective is to label each vertex with the community that it belongs to. However, similar to many data clustering algorithms such as K-means, the number of communities are not known in advance. The problem of community detection or graph clustering has been studied extensively. Among the developed algorithms modularity based community detection (Brandes et al., 2008), spectral clustering algorithms (Spielmat and Teng, 1996; Lin and Cohen, 2010), label propagation (Zhu and Ghahramani, 2002; Fujiwara and Irie, 2014; Ugander and Backstrom, 2013) have been more successful.

An interesting connection of graph clustering or vertex clustering to the representation learning is that we can use embedded vector of vertices as data points and run K-means clustering to find possible graph clustering. This is similar to spectral clustering, but in

spectral clustering, the representation is constructed using the eigenvectors, while in the representation learning, the embedded vector is learned using probabilistic methods.

We can improve the objective function with extra terms that guides the learning procedure to force the vectors of vertices in similar communities to be close to each other, especially if the target application of the representation learning is a vertex clustering application (Zheng et al., 2016; Tu et al., 2016; Wang et al., 2017).

To capture the community structure from the embedding, Zheng et al. (2016) use Gaussian mixture models (Bishop, 2006) to find the clusters given the embedding of the vertices. They suppose that the embedding of vertices of a cluster has a multivariate Gaussian distribution. Therefore, the embedding for the community k of the graph G is defined as tuple (Ψ_k, Σ_k), where Ψ_k and Σ_k are the mean and covariance matrix of a multivariate Gaussian distribution over the vertex embedding of G. As a result, community embedding characterizes the conditional probability of vertex v_i be a member of community k as a Gaussian distribution parameterized by Ψ_k and Σ_k:

$$P(v_i|z_i = k) = \mathcal{N}(\Phi(v_i); \Psi_k, \Sigma_k), \tag{14}$$

where z_i is the hidden variable indicating the community of vertex v_i. Therefore, the objective function for the community embedding becomes:

$$O_3 = \frac{\beta}{K} \sum_{i=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\Phi(v_i); \Psi_k, \Sigma_k), \tag{15}$$

where π_k is the fraction of nodes resides in the cluster k, N is the total number vertices in the graph, and K is the number of clusters. β controls the contribution of O_3 in the overall objective function with first-order and second-order proximity terms.

5 Learning Representation using Neural Networks

The goal of representation learning or embedding is to define a mapping $f : \mathcal{R}^n \rightarrow \mathcal{R}^d$ such that $d \ll n$. The function f is used to transform points from the original space into a low-dimensional space, which is similar to dimensionality reduction. Therefore, the approaches for dimensionality reduction can also be used to define the mapping. For example, singular value decomposition (SVD) is a well-known linear dimensional reduction method, which factorizes the matrix of data points X as $X \approx U\Lambda V^T$, where X is $m \times n$

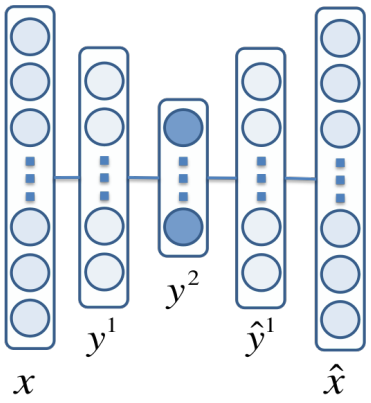

FIGURE 3: Simple stacked autoencoder with two layers of hidden variables.

matrix of m n-dimensional data points, and U and V are two $m \times d$ matrices. Λ is diagonal matrix of d singular values. In this setting the d-dimensional representation of each point x_i is defined as ΛV_i^T.

Another successfully applied dimensionality reduction is non-linear transformation. Laplacian eigenmaps (Belkin and Niyogi, 2003) try to learn a representation for data points considering the proximity of the points encoded as a similarity matrix W. To find the representation, we have to minimize the following objective function:

$$L = \sum_{i,j} W_{ij} \|f_i - f_j\|^2 = f^T \mathcal{L} f$$
$$s.t. f^T D 1 = 0, f^T D f = I, \tag{16}$$

where $\mathcal{L} = D - W$ is the laplacian and D is a diagonal matrix such that $D_{ii} = \sum_j W_{ij}$. The goal of the constraints is to make the representations balanced.

Another non-linear approach for learning representation is using neural networks. These type of neural networks are called autoencoders (Vincent et al., 2010). Figure **??** shows a simple autoencoder with two layers of hidden variables. The objective function of an autoencoder is minimizing the reconstruction loss: $\sum_{\mathbf{x} \in \mathcal{D}} \|\hat{\mathbf{x}} - \mathbf{x}\|$, where $\mathcal{D}$ is the training set of points. After learning the parameters, the embedded vector of an input data point is the value of the middle hidden layer for that input. Autoencoders are computationally more efficient than SVDs (the learning time of autoencoder is linear in the number of vertices, while the running time of SVD is quadratic in the number of vertices (Cao et al., 2016)).

Given an adjacency matrix A for a graph G, if we consider each row of matrix A as a sample point A_i, then we can use a stacked autoencoder to learn a representation for

each sample point. Here, for each vertex i, the corresponding input vector A_i contains the neighborhood information of vertex i. Therefore, the representation that we learn using a stacked autoencoder embeds the second-order proximity (Wang et al., 2016). However, since the most entries of A_i are zero, naively using autoencoder to minimize the reconstruction loss results in reconstructed vector $\hat{A}_i$ whose all of its entries are zero. To address this problem, Wang et al. (2016) use a modified loss function that increases the loss when the value of the entry is one. The modified loss function encourages the autoencoder to avoid predicting all zero vector:

$$L_2 = \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i}) \odot \mathbf{b}_\mathbf{i}\|_\mathbf{2}^\mathbf{2}, \tag{17}$$

where $\odot$ is element-wise multiplication, and b_{ij} is one when x_{ij} is zero and some value greater that one when $x_{ij} = 1$. Therefore, the model is forced to learn a meaningful representation by increasing the loss when $x_{ij} = 1$.

The aforementioned loss function only preserves the second-order proximity. To preserve the first-order proximity, we need the learned representation of two vertices be close to each other if those vertices are neighbors in the graph. There if A_{ij} is one we want to $g(A_I)$ and $g(A_j)$ be close to each other. Wang et al. (2016) introduce the following objective to satisfy the first-order proximity:

$$L_1 = \sum_{ij} A_{ij} \|\mathbf{y}_\mathbf{i}^{(\mathbf{K})} - \mathbf{y}_\mathbf{j}^{(\mathbf{K})}\|_\mathbf{2}^\mathbf{2} \tag{18}$$

Figure 4 shows how two autoencoders for two vertices interact to preserve the first-order proximity. It is worth mentioning that each of the instances of the autoencoder shares the weight vectors. By jointly optimizing L_1 and L_2, the learned representations satisfy both first-order and second-order proximity objectives. Wang et al. (2016) experimentally show that using stacked autoencoders results in better performance comparing to LINE (Tang et al., 2015b), which optimizes the first-order and second-order proximity objectives using a shallow neural model.

In another work, Cao et al. (2016) build on their previous work GraRep (Cao et al., 2015) by replacing SVD with a stacked denoising autoencoder. Moreover, instead of factorizing each P^k separately, they apply dimensionally reduction on combination of them: $r = \sum_{k=1}^{K} w(k)P^k$, where $w(t)$ is a non-decreasing function.

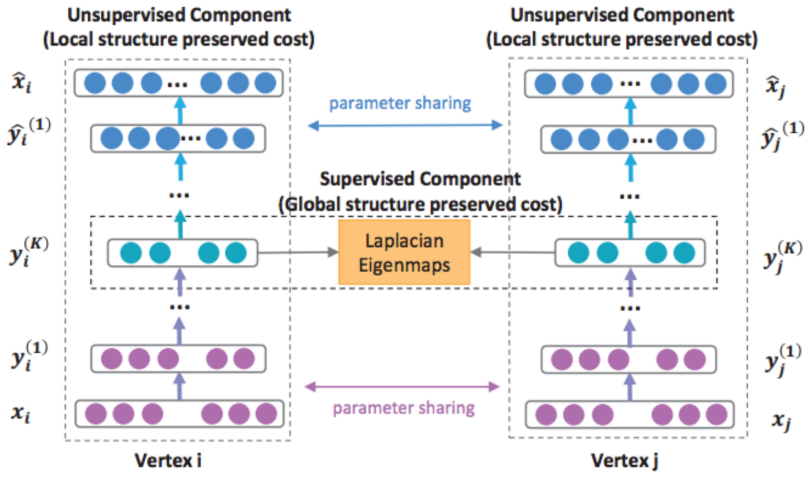

FIGURE 4: (Wang et al., 2016) Stacked autoencoder model for learning node representations. The learned representation for vertex i is $\mathbf{y_i^{(K)}}$

6 Information Enhanced Representation Learning

The method previously described for learning representation of graphs, only considers the structure of the graph for finding the representation, while in many real-world graphs, other information is attached to the users, which can improve the learned representation. For example, in the Twitter graph, each user is a vertex in the graph, and the edges show the friend and follower relationship. Suppose we are interested in learning the vertex representations such that the representations of the political tweets are close to each other. Such a representation can be used to group the political users together. Having the information about the users' tweets helps the representation to better distinguish between political and non-political users. This problem is known as collective classification (Sen et al., 2008), in which the labels of the subset of users are given as the training data. We study the classification of vertices in Section 4.

Yang et al. (2015) use both the structure of a graph and text features associated with each vertex to learn a representation, which is called text-associated deep walk (TADW). Yang et al. (2015) first show that the vertex representation learned by DeepWalk (Perozzi et al., 2014) can also be obtained by using matrix factorization. A similar result has been shown for learning word representation using skip-gram (Levy and Goldberg, 2014). In the matrix factorization version of DeepWalk, the objective is to find a factorization of a low-rank matrix M to minimize:

$$\min_{W,H} \sum_{ij} (M_{ij} - (W^T H)_{ij})^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2), \tag{19}$$

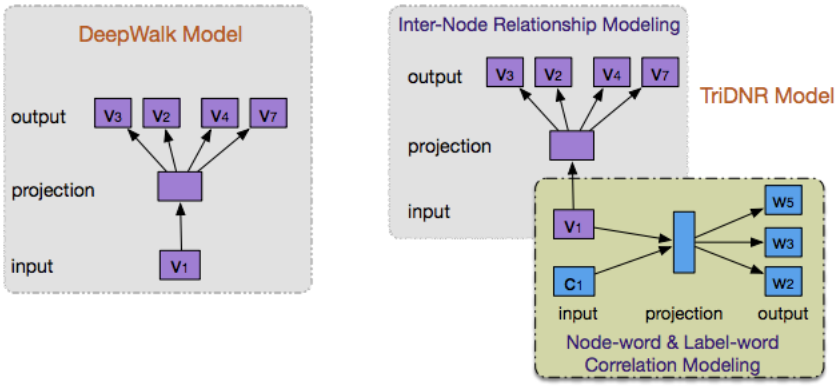

FIGURE 5: (Pan et al., 2016) The neural network model of the DeepWalk method (left), and the coupled neural network model of Tri-Party (right).

where $W \in \mathcal{R}^{|V| \times k}$ such that each rows shows the representation of the target vertices and $H \in \mathcal{R}^{k \times |V|}$ such that the columns show the representation of the context vertices (Yang et al., 2015). The matrix M is computed from the generated random walks:

$$M_{ij} = \log \frac{N(v_i, c_j)}{N(v_i)}, \tag{20}$$

where $N(.)$ and $N(.,.)$ count the occurrence and the co-occurrence of the vertices and contexts. In order to incorporate the text features, TADW supposes that the text features for each vertex is given as matrix $T \in \mathcal{R}^{f_t \times |V|}$, where f_t is the dimension of the text features. Therefore, TADW minimizes the following matrix factorization objective functions:

$$\min_{W,H} \sum_{ij} (M_{ij} - (W^T HT)_{ij})^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2), \tag{21}$$

where $H \in \mathcal{R}^{k \times f_t}$. In the task of vertex classification when the text features are available, TADW shows a considerable improvement over DeepWalk and other representation learning methods that only consider the structural properties of the graph. However, for large-scale graphs solving Relation 21 is very computationally expensive in terms of constructing matrix M and running matrix factorization methods such as SVD. To alleviate this problem, TADW constructs an approximation of M which reduces the accuracy of the algorithms.

To address the scalability problem, Pan et al. (2016) introduce a coupled neural network architecture that learn representation from three parties: graph structure, node contents and node labels (if is available otherwise it ignores this part). This architecture is called

Tri-Party. Figure 5 shows the architecture of Tri-Party as well as the neural network models of the DeepWalk method. The Tri-Party's neural network model can capture the inter-node occurrence similar to DeepWalk, while it can also capture node-word and label-word occurrence. Comparing to TADW, Tri-Party extracts the text features jointly with learning the representation, while TADW first learns the text features and then used it for learning the vertex representations. In tasks of vertex classification, Pan et al. (2016) show that Tri-Party is constantly better than TADW.

7 Heterogeneous Graph Embedding

Many existing algorithms for graph embeddings only consider the homogeneous graphs while nowadays, many real-world networks are heterogeneous. For example, in social networks, we can have multiple node types such as videos, images, text, and users, which are connected through edges with multiple types as well (Sun and Han, 2013). Some edges link single classes of objects, for example, image to image, while some others link different class of objects such as users to images.

A successful embedding methods for the heterogeneous should jointly consider the multiple content types and structural properties of the graphs.

The main problem of jointly learning the vertex representation is that relations among different vertex types have different semantic, so they are not comparable in general. Therefore, using methods such as DeepWalk (Perozzi et al., 2014) or LINE (Tang et al., 2015b) that does not discriminate among different relation types result in a poor performance in the presence of heterogeneous networks.

Two main approaches have been introduced to solve this problem. One approach is to separate the networks of different types and jointly learn a representation for all networks. In this approach, the heterogeneous network is union several homogeneous networks. Tang et al. (2015a) study learning a word embedding by constructing a heterogeneous networks, which consists, a network of word-word relations, a network of word-document relations, and a network of word-label relations. They have extended LINE's objective function (Equation 11) by introducing separate terms for different networks and then jointly optimizing them:

$$O = O_{ww} + O_{wd} + O_{wl}, \tag{22}$$

O_{ww}, O_{wd}, and O_{wl} are the objectives of word-word, word-document, word-label networks, respectively.

Similarly, Xu et al. (2017) separate a heterogeneous network into the constructing homogeneous networks and edges between these networks. As it learns a latent space representation for the vertices of each network using first order proximity similar to LINE (Tang et al., 2015b), it also considers the inter-network edges that connect two homogeneous networks. To represent these edges, Xu et al. (2017) defines the following joint probability over the nodes from two different networks:

$$p(u_i, v_j) = \frac{1}{1 + \exp(-\mathbf{u}_i^T M \mathbf{v}_j)}, \tag{23}$$

where M is $d_u \times d_v$ real-valued matrix that relates the latent space of two homogeneous networks and d_u and d_v are the dimension of the representations of the networks. For two homogeneous network $G_u(V_u, E_u)$ and $G_v(V_v, E_v)$ and crossing edges E_{uv}, the final objective of this model becomes as:

$$O = L_{pos} + L_{neg} + L_{reg}, \tag{24}$$

where L_{pos} emphasizes the existing edge, L_{neg} penalizes the model for considering an edge between two not-connected vertices, and L_{reg} is the regularization part: $L_{reg} = \lambda \sum_{i \in V_u} \|u_i\|^2 + \gamma \sum_{i \in V_v} \|v_i\|^2 + \|M\|$. Here the parameter γ, λ and β are for adjusting the regularization on different part of the objective function.

Xu et al. (2017) define L_{pos} as:

$$\begin{aligned} L_{pos} = -[& \sum_{(i,j) \in E_u} w_{ij}^u \log p(u_i, u_j) + \\ & \sum_{(i,j) \in E_v} w_{ij}^v \log p(v_i, v_j) + \\ & \sum_{(i,j) \in E_{uv}} w_{ij}^{uv} \log p(u_i, v_j)]. \end{aligned} \tag{25}$$

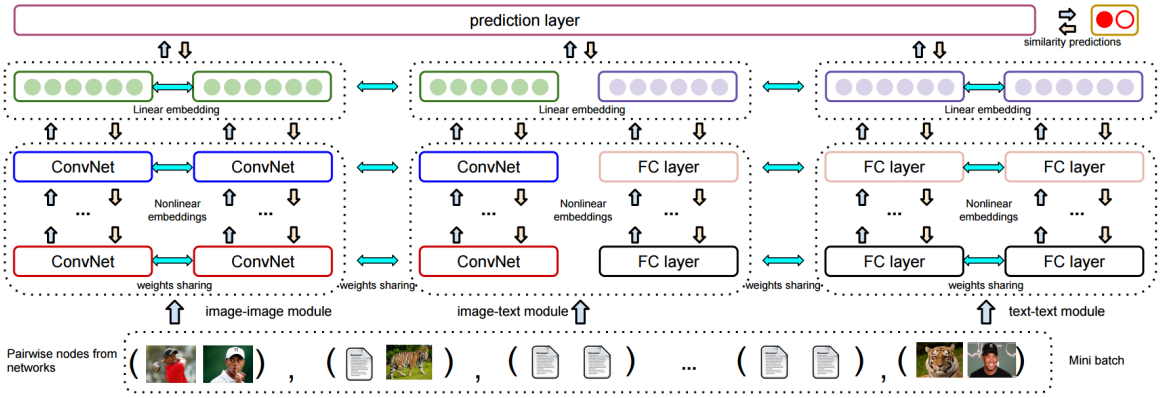

FIGURE 6: (Chang et al., 2015) The architecture of HNE. The image input is fed to convolutional neural networks and text input is fed to fully connected neural network. The output of both networks are mapped to the same latent space using linear transformations.

The goal of L_{neg} is penalize the model from presenting an edge between two vertices that are not connected:

$$\begin{aligned} L_{neg} = -[& \sum_{(i,j)\notin E_u} w_{ij}^{u} \log 1 - p(u_i, u_j) + \\ & \sum_{(i,j)\notin E_v} w_{ij}^{v} \log 1 - p(v_i, v_j) + \\ & \sum_{(i,j)\notin E_{uv}} w_{ij}^{uv} \log 1 - p(u_i, v_j)]. \end{aligned} \tag{26}$$

However, considering all pairs of vertices in a large-scale graph is impractical; therefore, Xu et al. (2017) only use a limited number sample from all possible pairs.

In another approach, Chang et al. (2015) introduce HNE which learns a representation for a heterogeneous network of image and text data. HNE uses convolutional neural networks for embedding image data and fully connected neural networks for embedding text. However, it supposes that the vector representation of text data and image data are not comparable, so it introduces matrices U and V, which map the vector representation of image and text data, respectively, into an r-dimensional space using linear transformations. Figure 6 shows the architecture of HNE.

Given $\mathbf{t}_j$ and $\mathbf{x}_j$ are text and image samples, respectively, the output representation of a convolutional neural network is a d_T dimensional vector $g_T(t_j)$ and the output representation of a fully connected neural network d_I dimensional vector $g_I(\mathbf{x}_j)$. The

linear transformation maps both of these vectors into a r-dimensional space:

$$\hat{g}_I(\mathbf{x}_j) = U^T g_I(\mathbf{x}_j) \tag{27}$$

$$\hat{g}_T(\mathbf{t}_j) = V^T g_T(\mathbf{t}_j) \tag{28}$$

Finally, HNE minimizes the following objective function over the set of parameters of convolutional neural network P_I, full connected neural network P_T, and linear transformation matrices U and V using stochastic gradient descent:

$$\begin{aligned} \min_{P_I, P_T, U, V} \quad & \frac{1}{N_{II}} \sum_{v_i, v_j \in V_I} L(\hat{g}_I(\mathbf{x}_i), \hat{g}_I(\mathbf{x}_j)) \\ & + \frac{\lambda_1}{N_{TT}} \sum_{v_i, v_j \in V_T} L(\hat{g}_T(\mathbf{t}_i), \hat{g}_T(\mathbf{t}_j)) \\ & + \frac{\lambda_2}{N_{TI}} \sum_{v_i \in V_T, v_j \in V_I} L(\hat{g}_T(\mathbf{t}_i), \hat{g}_I(\mathbf{x}_j)), \end{aligned} \tag{29}$$

where V_I and V_T are the set of image and text vertices, respectively, and N_{II}, N_{IT}, and N_{TT} are the number of image-image, image-text, and text-text in the heterogeneous graphs. The loss function L is defined as $L(\mathbf{y}_i, \mathbf{y}_j) = \log(1 + \exp(-A_{ij}\mathbf{y}_i^T\mathbf{y}_j))$, where A_{ij} is one if there exists an edge between the vertices represented by r-dimensional vectors $\mathbf{y}_i$ and $\mathbf{y}_j$, otherwise A_{ij} is zero. The loss function L forces the r-dimensional representation of vertices become close to each other if they are neighbours in the heterogeneous graph.

8 Classification of Graph Data

Classification of graph data has very important applications such as protein-protein interaction or predicting the functionality of the chemical compounds. In the graph classification problem, we are given a set of graphs $\{g_1, g_2, \cdots g_n\}$, in which only the labels of a subset of graphs are seen, and the goal is to predict the label of unseen graphs.

Graph classification is a difficult task known to be NP-hard since we can reduce it to check graph isomorphism. The basic method for comparing two graphs g_i and g_j is to compute the *edit-distance* between two graphs, which is the number of modification needed for transforming graph g_i to graph g_j.

8.1 Kernel Graphs

Kernel methods are one of the major development in machine learning. A kernel measures the similarity of the data points, which can be used for classification. For example, for a simple linear classification task $h(x) = \text{sign}(\mathbf{w}^T q(x) + b)$ can be rewritten as $h(x) = \text{sign}(\sum_i \alpha_i k(x, x_i) + b)$, where $k(x, x_i) = q(x)^T q(x)$ is the kernel function that measures the similarity of sample points using the extracted feature function $q(.)$.

Similarly, we can define $k(g_i, g_j)$ to measure the similarity of two graphs g_i and g_j. In this case $q(g_i)$ is the vector of features extracted from graph g_i. Different graph kernel methods merely differ based on how they extracted features from graphs. The most common graph kernels are random walk kernels (Borgwardt et al., 2005b), shortest-path kernels (Borgwardt and Kriegel, 2005), graphlet kernels (Shervashidze et al., 2009), and Weisfeiler-Lehman graph Kernels (Shervashidze et al., 2011).

In the following sections, we briefly discuss the computation of these graph kernels.

8.1.1 Random-Walk Graph Kernels

A random-Walk kernel $k_{sp}(g_i, g_j)$ measures the similarity of two labeled graphs g_i and g_j by comparing the random walks of the graphs. A label graph has a label $l^v \in \{l_1, l_2, \cdots l_k\}$ to every vertex $v \in V$. The vector $q(g_i)$ is formed by counting the number of tuples (l_s^i, l_d^j, m) generated from the graph, where l^i and l^j are the labels of source vertex s and destination vertex d in a walk, respectively, and m is the length of the walk.

8.1.2 Shortest-Path Graph Kernels

Shortest-path kernels (Borgwardt and Kriegel, 2005) are similar to random-walk graph kernels but the tuples (l_s^i, l_d^j, m) are formed from the shortest paths instead of walks. Shortest-path kernels are more accurate compare to random-walk kernels since the shortest paths are more direct compare to random walks. However, forming shortest-path kernels needs the computation of all shortest paths which is $O(n^3)$ so forming the shortest-path kernel is $O(n^4)$.

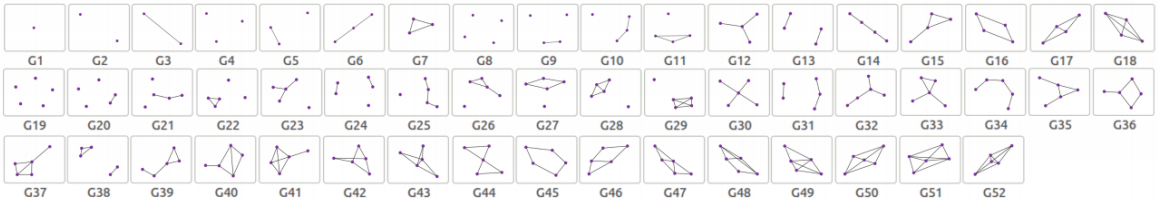

FIGURE 7: (Yanardag and Vishwanathan, 2015) The set of non-isomorphic graphlets with less than 6 vertices.

8.1.3 Graphlet Graph Kernels

The graphlet kernels (Shervashidze et al., 2009) count the number of substructures, graphlets, in the graphs. Figure 7 shows the set of non-isomorphic graphlets with fewer than six vertices. For this set of graphlets, $q(g_i)$ has 52 to entries, and each entry shows the normalized count of the corresponding graphlet in the graph g_i. For counting graphlets with k vertices in a graph with n vertices, we need to explore $\binom{n}{x}$ combinations, which has complexity $O(n^k)$. Therefore, graphlet counting is more practical with smaller k. To alleviate this problem, Shervashidze et al. (2009) suggest sampling the graphlets from the graph instead of exhaustive search.

8.1.4 Weisfeiler-Lehman Graph Kernels

This category of kernels is based counting the share subtrees in graphs. Weisfeiler-Lehman graph kernel uses an iterative relabeling process to compute the kernel values, and it designed for labeled graphs. Each relabeling iteration maps graph (V, E, L_h) to (V, E, L_{h+1}), in which only the labels of the vertices are different. In iteration h, each vertex i collects the labels of its neighbors as tuple of $(l_i, l_{N(i)})$, where l_i is the label of vertex i and $l_{N(i)}$ is the set of labels of the neighbors of vertex i (Figure 8.a). At the end of each iteration, the relabeling process assigns a new label for each tuple such that similar tuples get similar labels (Figure 8, part c, and d). The final feature for each graph is the concatenation of the label counts for each iteration (Figure 8.e).

8.2 Graph Classification Using Neural Network

The main problem of graph kernels is that the extracted features from the graphs are not independent (Narayanan et al., 2016; Yanardag and Vishwanathan, 2015). For example, one path of length l also includes the paths of length $l - 1$. Therefore, the vector

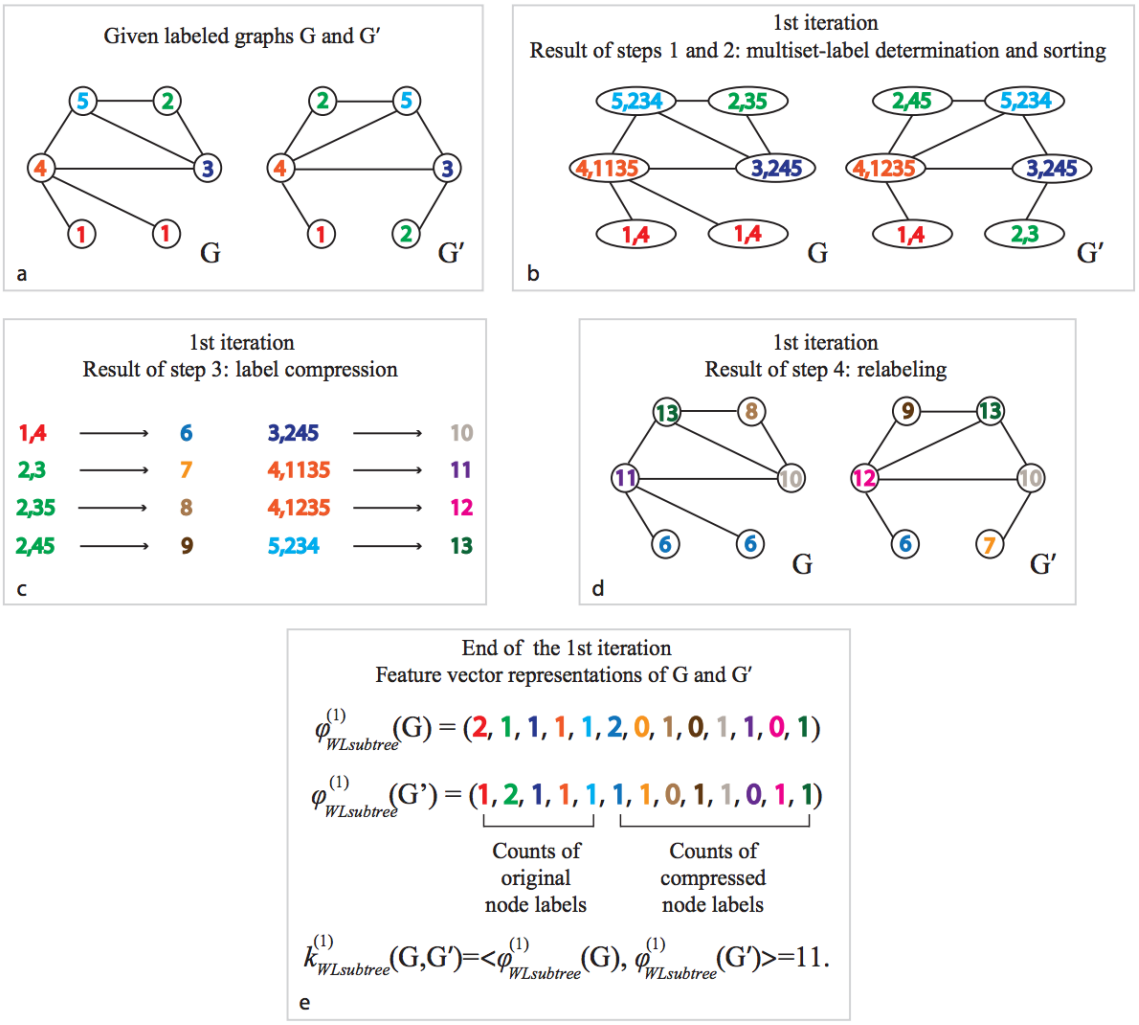

FIGURE 8: (Shervashidze et al., 2011) One iteration of the Weisfeiler-Lehman graph kernel.

representations of two graphs may become dissimilar although the substructures in these two graphs are similar. Yanardag and Vishwanathan (2015) shows this effect and names it as *diagonal dominance*, which is shown in Figure 9. The matrices show constructed Weisfeiler-Lehman kernels for classifying MUTAG dataset (Debnath et al., 1991).

To address this problem, Yanardag and Vishwanathan (2015) modify the graph kernels in to order to consider the similarity among the substructures used for feature extraction from the graphs:

$$K_d(g_i, g_j) = q(g_i)^T M q(g_j), \tag{30}$$

where M is the matrix express the similarity among the substructures. Yanardag and Vishwanathan (2015) use Skip-gram (see Section 2) to learn M. However, in order to

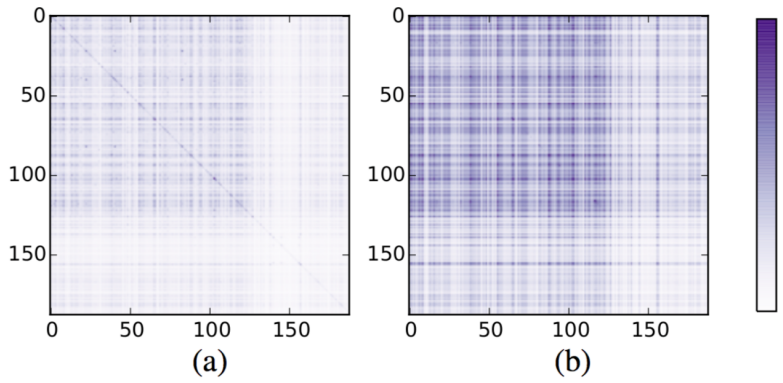

FIGURE 9: (Yanardag and Vishwanathan, 2015) a) Diagonal dominance effect for Weisfeiler-Lehman graph kernel on MUTAG dataset. b) Adding similarity matrix alleviate the problem.

use Skip-graph, we need to define the co-occurrence for the substructures. Yanardag and Vishwanathan (2015) define a graph of edit-distance over the substructures and consider two seen substructures as co-occurred if they are neighbors in the edit distance graph.

Using matrix M for emphasizing the similarity of the substructures increases the graph classification accuracy for the same feature extraction methods (Yanardag and Vishwanathan, 2015).

Another approach to consider graph structure in embedding is *subgraph2vec* (Narayanan et al., 2016), which tries to predict a subgraph context given each subgraph. Each subgraph is extracted as a breath-first-search tree rooted at each node based on Weisfeiler-Lehman relabeling process (Shervashidze et al., 2011), and the context of each subgraph is the rooted BFS tree extracted from all neighbors. Different from the aforementioned walk-based approaches, subgraph2vec benefits from non-linear context which is more suitable for graph data.

The state-of-the-art algorithm for graph classification defines a convolution neural network (CNNs) over input graphs (Niepert et al., 2016).

CNNs were successfully applied in different domains especially vision and natural language processing (Krizhevsky et al., 2012; Karpathy et al., 2014; Kim, 2014; Zhang et al., 2015). The main idea of CNNs to extract features from local regions of the input by moving a filter over different regions. However, in practice, we have a separate filter for each region, but they share their weight to emulate the convolution of a filter. A local region connected to a filter is called the receptive field of that filter. The convolution layer is followed by

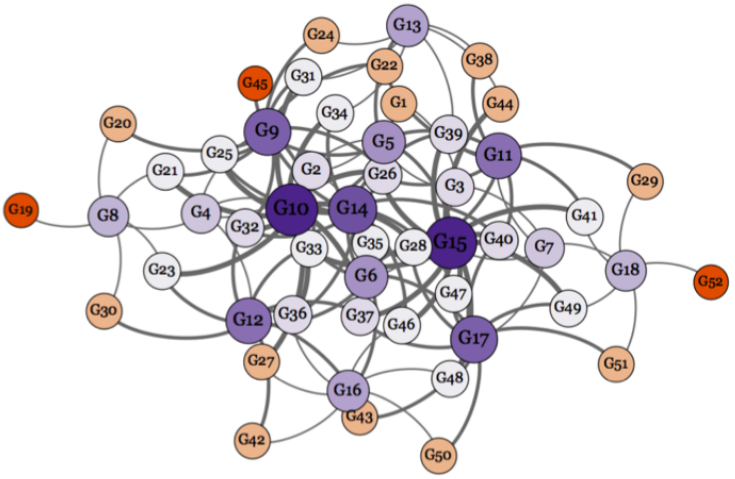

FIGURE 10: (Yanardag and Vishwanathan, 2015) Edit-distance graph of graphlets shown in Figure 7.

pooling layer which downsamples the output of the previous layer. The convolution layer and pooling layer may be used for extracting more than one common features.

In the model developed by Niepert et al. (2016), each receptive field of the convolution neural network receives a normalized subgraphs from the input graph.

However, the input of neural networks is fixed-length, but the graph data has a variable size. Therefor, we need to normalize the graph data. Niepert et al. (2016) introduce PATCHY-SAN algorithms to normalize the graph data, and then feed the normalized graph into a convolutional neural network. PATCHY-SAN suppose that the nodes of the input graphs have an ordering, so they are comparable. Given the ordering, it selects a set of w vertices as the root nodes from the ordered set of vertices. It picks root nodes with the stride of s, i.e. if it selects vertex i then the next vertex is $i + s$. Then, PATCHY-SAN extracts k nodes in the neighborhood of each root node using breath-first search, where k is the size of the receptive fields of the convolutional neural network. Finally, it normalizes each root node and its neighborhood based the closeness of each node in the neighborhood to the root node of that neighborhood. Figure 11 shows the process of normalizing graphs using PATCHY-SAN.

DeepGraphs (Li et al., 2016) is another approach for using deep learning for graph classification. However, instead of constructing receptive fields based on neighborhood extraction, it uses heat kernels (Chung, 2007; Bai and Hancock, 2004; Fang et al., 2015) to capture the features from the input graph. Heat kernels over graphs are defined using the eigenvalues and eigenvectors of the normalized graph Laplacian: $L_N = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$, where

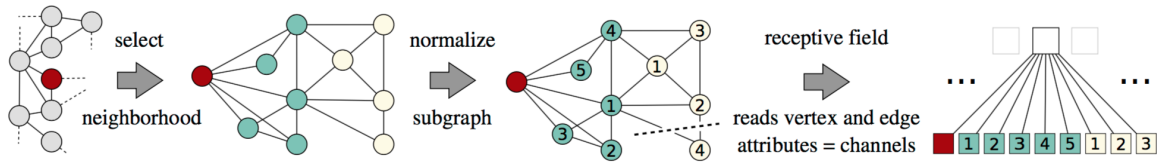

FIGURE 11: (Niepert et al., 2016) Normalizing the input graph in order to feed to the convolutional neural networks.

D is diagonal matrix of output weights: $D_{ii} = \sum_j W_i j$ and A is the adjacency graph. Then the heat kernel for diffusion step z is defined as a function of a pair of vertices i and j:

$$h_z(i,j) = \sum_k^{|V|} e^{\lambda_k z} \mu_k(i)\mu_k(j), \tag{31}$$

where $_k$ and μ_k are the kth eigenvalue and eigenvector of the input graph, respectively. Heat kernels have interesting properties, for example, every isomorphic graph has the same heat kernels, which makes it suitable for graph comparison and graph classification tasks. However, computing heat kernel is expensive, so instead, Li et al. (2016) use a heat kernel signature. A heat kernel signature over N diffusion steps is defined as a matrix $H \in \mathbb{R}^{|V|N}$, where $H_{iz} = h_z(i,i)$. Heat kernel signatures simplify the computation of heat kernels, while isomorphic graphs have similar heat kernel signatures. However, heat kernel signature H still depends on the ordering of vertices. To address this problem, Li et al. (2016) represent H with another matrix in which each column j is constructed based on the histogram of values on the column j of H. In this construction, the order of the vertices is not important anymore, which makes it possible to pass the matrix of histograms as the input to a deep neural network.

9 Conclusion and Future Directions

In this report, we have described representation learning for the graph data in various settings with shallow and deep neural networks. This setting includes homogeneous and heterogeneous networks as well as when associated information is available. We have also discussed how we can learn better representation by jointly learning that with clustering and classification algorithms.

The most promising future direction is to develop a representation learning algorithm that generalizes LINE (Tang et al., 2015b) to capture higher-order proximity. We believe an approach based on message-passing can be better capture higher-order proximity since two vertices with similar context receive similar messages. Moreover, a message-passing approach can easily be implemented over data-parallel graph frameworks such as GraphX (Gonzalez et al., 2014) to scale to very large graphs.

There are some possible directions mainly about jointly learning of the representation and different target application. For example, Zheng et al. (2016) enforce a flat clustering of the vertices, but in many applications, hierarchical graph clustering is more plausible, especially for analyzing the graph structure. Using learning a flat clustering and representation, and constructing the hierarchies from the representations may not be as accurate than jointly learning the hierarchical clustering and representation. We want to further study this specific problem as a future direction.

The representation learning approaches merely explore learning representations for the vertices of the graphs. However, as we have discussed, in the graph classification problems, we are given with different graphs and the problem is to classify the graphs, which depends on feature extractions. However, these feature extractions are expensive, so we want to explore ways to jointly learn vertex representation as well as representation for the whole graphs. Therefore, the graph representations can be used to train classifiers.

Bibliography

[Agrawal et al. 2013] Agrawal, Priyanka ; Garg, Vikas K. ; Narayanam, Ramasuri: Link Label Prediction in Signed Social Networks. In: *IJCAI*, 2013

[Backstrom and Leskovec 2011] Backstrom, Lars ; Leskovec, Jure: Supervised random walks: predicting and recommending links in social networks. In: *Proceedings of the fourth ACM international conference on Web search and data mining* ACM (Veranst.), 2011, p. 635–644

[Bai and Hancock 2004] Bai, Xiao ; Hancock, Edwin: Heat kernels, manifolds and graph embedding. In: *Structural, Syntactic, and Statistical Pattern Recognition* (2004), p. 198–206

[Baldi and Pollastri 2003] Baldi, Pierre ; Pollastri, Gianluca: The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. In: *Journal of Machine Learning Research* 4 (2003), No. Sep, p. 575–602

[Belkin and Niyogi 2003] Belkin, Mikhail ; Niyogi, Partha: Laplacian eigenmaps for dimensionality reduction and data representation. In: *Neural computation* 15 (2003), No. 6, p. 1373–1396

[Bengio et al. 2013] Bengio, Yoshua ; Courville, Aaron ; Vincent, Pascal: Representation learning: A review and new perspectives. In: *IEEE transactions on pattern analysis and machine intelligence* 35 (2013), No. 8, p. 1798–1828

[Bengio et al. 2003] Bengio, Yoshua ; Ducharme, Réjean ; Vincent, Pascal ; Jauvin, Christian: A neural probabilistic language model. In: *Journal of machine learning research* 3 (2003), No. Feb, p. 1137–1155

[Bishop 2006] Bishop, Christopher M.: Pattern recognition and machine learning. In: *Machine Learning* 128 (2006), p. 1–58

[Borgwardt and Kriegel 2005] Borgwardt, Karsten M. ; Kriegel, Hans-Peter: Shortest-path kernels on graphs. In: *Data Mining, Fifth IEEE International Conference on* IEEE (Veranst.), 2005, p. 8–pp

[Borgwardt et al. 2005a] Borgwardt, Karsten M. ; Ong, Cheng S. ; Schönauer, Stefan ; Vishwanathan, SVN ; Smola, Alex J. ; Kriegel, Hans-Peter: Protein function prediction via graph kernels. In: *Bioinformatics* 21 (2005), No. suppl 1, p. i47–i56

[Borgwardt et al. 2005b] Borgwardt, Karsten M. ; Ong, Cheng S. ; Schönauer, Stefan ; Vishwanathan, SVN ; Smola, Alex J. ; Kriegel, Hans-Peter: Protein function prediction via graph kernels. In: *Bioinformatics* 21 (2005), No. suppl 1, p. i47–i56

[Brandes et al. 2008] Brandes, Ulrik ; Delling, Daniel ; Gaertler, Marco ; Görke, Robert ; Hoefer, Martin ; Nikoloski, Zoran ; Wagner, Dorothea: On modularity clustering. In: *Knowledge and Data Engineering, IEEE Transactions on* 20 (2008), No. 2, p. 172–188

[Cao et al. 2015] Cao, Shaosheng ; Lu, Wei ; Xu, Qiongkai: Grarep: Learning graph representations with global structural information. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* ACM (Veranst.), 2015, p. 891–900

[Cao et al. 2016] Cao, Shaosheng ; Lu, Wei ; Xu, Qiongkai: Deep Neural Networks for Learning Graph Representations. (2016)

[Chang et al. 2015] Chang, Shiyu ; Han, Wei ; Tang, Jiliang ; Qi, Guo-Jun ; Aggarwal, Charu C. ; Huang, Thomas S.: Heterogeneous network embedding via deep architectures. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM (Veranst.), 2015, p. 119–128

[Chung 2007] Chung, Fan: The heat kernel as the pagerank of a graph. In: *Proceedings of the National Academy of Sciences* 104 (2007), No. 50, p. 19735–19740

[Debnath et al. 1991] Debnath, Asim K. ; Lopez, de Compadre R. ; Debnath, Gargi ; Shusterman, Alan J. ; Hansch, Corwin: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. In: *Journal of medicinal chemistry* 34 (1991), No. 2, p. 786–797

[Dyer 2014] Dyer, Chris: Notes on Noise Contrastive Estimation and Negative Sampling. In: *arXiv preprint arXiv:1410.8251* (2014)

[Fang et al. 2015] Fang, Yi ; Sun, Mengtian ; Ramani, Karthik: Heat-passing framework for robust interpretation of data in networks. In: *PloS one* 10 (2015), No. 2, p. e0116121

[Felzenszwalb and Huttenlocher 2004] Felzenszwalb, Pedro F. ; Huttenlocher, Daniel P.: Efficient graph-based image segmentation. In: *International journal of computer vision* 59 (2004), No. 2, p. 167–181

[Fujiwara and Irie 2014] Fujiwara, Yasuhiro ; Irie, Go: Efficient label propagation. In: *Proceedings of the 31st international conference on machine learning (ICML-14)*, 2014, p. 784–792

[Gonzalez et al. 2014] Gonzalez, Joseph E. ; Xin, Reynold S. ; Dave, Ankur ; Crankshaw, Daniel ; Franklin, Michael J. ; Stoica, Ion: GraphX: Graph processing in a distributed dataflow framework. In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. Broomfield, CO, USA, 2014 (OSDI 14), p. 599–613

[Grover and Leskovec 2016] Grover, Aditya ; Leskovec, Jure: node2vec: Scalable Feature Learning for Networks. In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016

[Karpathy et al. 2014] Karpathy, Andrej ; Toderici, George ; Shetty, Sanketh ; Leung, Thomas ; Sukthankar, Rahul ; Fei-Fei, Li: Large-scale video classification with convolutional neural networks. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, p. 1725–1732

[Kim 2014] Kim, Yoon: Convolutional neural networks for sentence classification. In: *arXiv preprint arXiv:1408.5882* (2014)

[Krizhevsky et al. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F. (Hrsg.) ; Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, p. 1097–1105. – URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[Levy and Goldberg 2014] Levy, Omer ; Goldberg, Yoav: Neural word embedding as implicit matrix factorization. In: *Advances in neural information processing systems*, 2014, p. 2177–2185

[Li et al. 2016] Li, Cheng ; Guo, Xiaoxiao ; Mei, Qiaozhu: DeepGraph: Graph Structure Predicts Network Growth. In: *arXiv preprint arXiv:1610.06251* (2016)

[Liben-Nowell and Kleinberg 2007] Liben-Nowell, David ; Kleinberg, Jon: The link-prediction problem for social networks. In: *journal of the Association for Information Science and Technology* 58 (2007), No. 7, p. 1019–1031

[Lin and Cohen 2010] Lin, Frank ; Cohen, William W.: Power iteration clustering. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, p. 655–662

[Maaten and Hinton 2008] Maaten, Laurens van d. ; Hinton, Geoffrey: Visualizing data using t-SNE. In: *Journal of Machine Learning Research* 9 (2008), No. Nov, p. 2579–2605

[Mikolov et al. 2013a] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Efficient estimation of word representations in vector space. In: *arXiv preprint arXiv:1301.3781* (2013)

[Mikolov et al. 2013b] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg S. ; Dean, Jeff: Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*, 2013, p. 3111–3119

[Mikolov et al. 2013c] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg S. ; Dean, Jeff: Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*, 2013, p. 3111–3119

[Morin and Bengio 2005] Morin, Frederic ; Bengio, Yoshua: Hierarchical Probabilistic Neural Network Language Model. In: *Aistats* Bd. 5 Citeseer (Veranst.), 2005, p. 246–252

[Nadler et al. 2009] Nadler, Boaz ; Srebro, Nathan ; Zhou, Xueyuan: Semi-supervised learning with the graph Laplacian: The limit of infinite unlabelled data. In: *Advances in neural information processing systems* 21 (2009)

[Narayanan et al. 2016] Narayanan, Annamalai ; Chandramohan, Mahinthan ; Chen, Lihui ; Liu, Yang ; Saminathan, Santhoshkumar: subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. In: *arXiv preprint arXiv:1606.08928* (2016)

[Niepert et al. 2016] Niepert, Mathias ; Ahmed, Mohamed ; Kutzkov, Konstantin: Learning Convolutional Neural Networks for Graphs. In: *arXiv preprint arXiv:1605.05273* (2016)

[Pan et al. 2016] Pan, Shirui ; Wu, Jia ; Zhu, Xingquan ; Zhang, Chengqi ; Wang, Yang: Tri-party deep network representation. In: *IJCAI*, 2016

[Perozzi et al. 2014] Perozzi, Bryan ; Al-Rfou, Rami ; Skiena, Steven: DeepWalk: Online Learning of Social Representations. In: *CoRR* abs/1403.6652 (2014). – URL http://arxiv.org/abs/1403.6652

[Perozzi et al. 2016] Perozzi, Bryan ; Kulkarni, Vivek ; Skiena, Steven: Walklets: Multiscale Graph Embeddings for Interpretable Network Classification. In: *arXiv preprint arXiv:1605.02115* (2016)

[Ralaivola et al. 2005] Ralaivola, Liva ; Swamidass, Sanjay J. ; Saigo, Hiroto ; Baldi, Pierre: Graph kernels for chemical informatics. In: *Neural networks* 18 (2005), No. 8, p. 1093–1110

[Sen et al. 2008] Sen, Prithviraj ; Namata, Galileo ; Bilgic, Mustafa ; Getoor, Lise ; Galligher, Brian ; Eliassi-Rad, Tina: Collective classification in network data. In: *AI magazine* 29 (2008), No. 3, p. 93

[Shervashidze et al. 2011] Shervashidze, Nino ; Schweitzer, Pascal ; Leeuwen, Erik Jan v. ; Mehlhorn, Kurt ; Borgwardt, Karsten M.: Weisfeiler-lehman graph kernels. In: *Journal of Machine Learning Research* 12 (2011), No. Sep, p. 2539–2561

[Shervashidze et al. 2009] Shervashidze, Nino ; Vishwanathan, SVN ; Petri, Tobias ; Mehlhorn, Kurt ; Borgwardt, Karsten M.: Efficient graphlet kernels for large graph comparison. In: *AISTATS* Bd. 5, 2009, p. 488–495

[Shi and Malik 2000] Shi, Jianbo ; Malik, Jitendra: Normalized cuts and image segmentation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (2000), No. 8, p. 888–905

[Spielmat and Teng 1996] Spielmat, Daniel A. ; Teng, Shang-Hua: Spectral partitioning works: Planar graphs and finite element meshes. In: *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on* IEEE (Veranst.), 1996, p. 96–105

[Sun and Han 2013] Sun, Yizhou ; Han, Jiawei: Mining heterogeneous information networks: a structural analysis approach. In: *ACM SIGKDD Explorations Newsletter* 14 (2013), No. 2, p. 20–28

[Tang et al. 2015a] Tang, Jian ; Qu, Meng ; Mei, Qiaozhu: Pte: Predictive text embedding through large-scale heterogeneous text networks. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM (Veranst.), 2015, p. 1165–1174

[Tang et al. 2015b] Tang, Jian ; Qu, Meng ; Wang, Mingzhe ; Zhang, Ming ; Yan, Jun ; Mei, Qiaozhu: Line: Large-scale information network embedding. In: *Proceedings of the 24th International Conference on World Wide Web* ACM (Veranst.), 2015, p. 1067–1077

[Tu et al. 2016] Tu, Cunchao ; Wang, Hao ; Zeng, Xiangkai ; Liu, Zhiyuan ; Sun, Maosong: Community-enhanced Network Representation Learning for Network Analysis. In: *arXiv preprint arXiv:1611.06645* (2016)

[Ugander and Backstrom 2013] Ugander, Johan ; Backstrom, Lars: Balanced label propagation for partitioning massive graphs. In: *Proceedings of the sixth ACM international conference on Web search and data mining* ACM (Veranst.), 2013, p. 507–516

[Vincent et al. 2010] Vincent, Pascal ; Larochelle, Hugo ; Lajoie, Isabelle ; Bengio, Yoshua ; Manzagol, Pierre-Antoine: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. In: *Journal of Machine Learning Research* 11 (2010), No. Dec, p. 3371–3408

[Wale et al. 2008] Wale, Nikil ; Watson, Ian A. ; Karypis, George: Comparison of descriptor spaces for chemical compound retrieval and classification. In: *Knowledge and Information Systems* 14 (2008), No. 3, p. 347–375

[Wang et al. 2016] Wang, Daixin ; Cui, Peng ; Zhu, Wenwu: Structural Deep Network Embedding. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 2016 (KDD

'16), p. 1225–1234. – URL http://doi.acm.org/10.1145/2939672.2939753. – ISBN 978-1-4503-4232-2

[Wang et al. 2017] Wang, Xiao ; Cui, Peng ; Wang, Jing ; Pei, Jian ; Zhu, Wenwu ; Yang, Shiqiang: Community Preserving Network Embedding. (2017)

[Weston et al. 2012] Weston, Jason ; Ratle, Frédéric ; Mobahi, Hossein ; Collobert, Ronan: Deep learning via semi-supervised embedding. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, p. 639–655

[Xu et al. 2017] Xu, Linchuan ; Wei, Xiaokai ; Cao, Jiannong ; Yu, Philip S.: Embedding of Embedding (EOE): Joint Embedding for Coupled Heterogeneous Networks. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* ACM (Veranst.), 2017, p. 741–749

[Yanardag and Vishwanathan 2015] Yanardag, Pinar ; Vishwanathan, SVN: Deep graph kernels. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM (Veranst.), 2015, p. 1365–1374

[Yang and Liu 2015] Yang, Cheng ; Liu, Zhiyuan: Comprehend deepwalk as matrix factorization. In: *arXiv preprint arXiv:1501.00358* (2015)

[Yang et al. 2015] Yang, Cheng ; Liu, Zhiyuan ; Zhao, Deli ; Sun, Maosong ; Chang, Edward Y.: Network Representation Learning with Rich Text Information. In: *IJCAI*, 2015, p. 2111–2117

[Yang et al. 2016] Yang, Zhilin ; Cohen, William W. ; Salakhutdinov, Ruslan: Revisiting semi-supervised learning with graph embeddings. In: *arXiv preprint arXiv:1603.08861* (2016)

[Zhang et al. 2015] Zhang, Xiang ; Zhao, Junbo ; LeCun, Yann: Character-level convolutional networks for text classification. In: *Advances in neural information processing systems*, 2015, p. 649–657

[Zheng et al. 2016] Zheng, Vincent W. ; Cavallari, Sandro ; Cai, Hongyun ; Chang, Kevin Chen-Chuan ; Cambria, Erik: From Node Embedding To Community Embedding. In: *arXiv preprint arXiv:1610.09950* (2016)

[Zhu and Ghahramani 2002] Zhu, Xiaojin ; Ghahramani, Zoubin: Learning from labeled and unlabeled data with label propagation. (2002)

[Zhu et al. 2005] Zhu, Xiaojin ; Lafferty, John ; Rosenfeld, Ronald: *Semi-supervised learning with graphs*. Carnegie Mellon University, language technologies institute, school of computer science, 2005