Ph.D. DISSERTATION

Unsupervised Representation Learning for Homogeneous, Heterogeneous, and Tree-like Graphs

동종, 이종, 그리고 나무 형태의 그래프를 위한
비지도 표현 학습

BY

Jiwoong Park

AUGUST 2022

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Unsupervised Representation Learning for Homogeneous, Heterogeneous, and Tree-like Graphs

동종, 이종, 그리고 나무 형태의 그래프를 위한
비지도 표현 학습

BY

Jiwoong Park

AUGUST 2022

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Unsupervised Representation Learning for Homogeneous, Heterogeneous, and Tree-like Graphs

동종, 이종, 그리고 나무 형태의 그래프를 위한
비지도 표현 학습

지도교수 최 진 영

이 논문을 공학박사 학위논문으로 제출함

2022년 8월

서울대학교 대학원

전기 정보 공학부

박 지 웅

박지웅의 공학박사 학위 논문을 인준함

2022년 8월

Abstract

The goal of unsupervised graph representation learning is extracting useful node-wise or graph-wise vector representation that is aware of the intrinsic structures of the graph and its attributes. These days, designing methodology of unsupervised graph representation learning based on graph neural networks has growing attention due to their powerful representation ability. Many methods are focused on a homogeneous graph that is a network with a single type of node and a single type of edge. However, as many types of relationships exist in this world, graphs can also be classified into various types by structural and semantic properties. For this reason, to learn useful representations from graphs, the unsupervised learning framework must consider the characteristics of the input graph. In this dissertation, we focus on designing unsupervised learning models using graph neural networks for three graph structures that are widely available: homogeneous graphs, tree-like graphs, and heterogeneous graphs.

First, we propose a symmetric graph convolutional autoencoder which produces a low-dimensional latent representation from a homogeneous graph. In contrast to the existing graph autoencoders with asymmetric decoder parts, the proposed autoencoder has a newly designed decoder which builds a completely symmetric autoencoder form. For the reconstruction of node features, the decoder is designed based on Laplacian sharpening as the counterpart of Laplacian smoothing of the encoder, which allows utilizing the graph structure in the whole processes of the proposed autoencoder architecture. In order to prevent the numerical instability of the network caused by the Laplacian sharpening introduction, we further propose a new numerically stable form of the Laplacian sharpening by incorporating the signed graphs. The experimental results of clustering, link prediction and visualization tasks on homogeneous graphs strongly support that the proposed model is stable and outperforms various state-of-the-art algorithms.

Second, we analyze how unsupervised tasks can benefit from learned representations in hyperbolic space. To explore how well the hierarchical structure of unlabeled data can be represented in hyperbolic spaces, we design a novel hyperbolic message passing autoencoder whose overall auto-encoding is performed in hyperbolic space. The proposed model conducts auto-encoding the networks via fully utilizing hyperbolic geometry in message passing. Through extensive quantitative and qualitative analyses, we validate the properties and benefits of the unsupervised hyperbolic representations of tree-like graphs.

Third, we propose the novel concept of metanode for message passing to learn both heterogeneous and homogeneous relationships between any two nodes without meta-paths and meta-graphs. Unlike conventional methods, metanodes do not require a predetermined step to manipulate the given relations between different types to enrich relational information. Going one step further, we propose a metanode-based message passing layer and a contrastive learning model using the proposed layer. In our experiments, we show the competitive performance of the proposed metanode-based message passing method on node clustering and node classification tasks, when compared to state-of-the-art methods for message passing networks for heterogeneous graphs.

keywords: Unsupervised Graph Representation Learning, Graph Neural Networks, Autoencoder, Contrastive Learning

student number: 2018-35712

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A fundamental problem of machine learning is learning useful representations from high-dimensional data. There are many (semi-) supervised representation learning methods that achieve good performances for downstream tasks [2–5] on several data domains such as images and graphs. In recent years, with the success of deep learning, various large-scale real-world datasets have been collated [2, 6–8]. However, the larger these datasets and the closer they are to the real world, the expense and effort required to label the data increases proportionally. Also, supervised representation learning might suffer deteriorated performances due to the lacked generalization capability from limited training data and noisy labels. Thus, unsupervised representation learning is an increasingly viable approach to extract useful representation from real-world datasets.

Among the various data domain, unsupervised representation learning of graph-structured data is one of the most important machine learning subjects. This is because, a graph representing objects and their relationships exists everywhere in our world. Social relations between people, hierarchy in an organization, links between web pages, bonds between atoms in a molecule, and purchase records between user and item are some of the representative examples. Thus, we can perceive our world via understanding the properties of graphs and extracting knowledge from them.

Recently, Graph Neural Networks (GNNs) [3, 9–11] are de facto models for unsupervised graph representation learning. Since GNNs were first proposed, the majority of efforts in this field have been aimed at learning representations for homogeneous graphs with a single type of object and a single type of relation. However, graph-structured datasets in real-world applications are not limited to a single type of nodes and edges and do not share the same structure. For example, relations between a hierarchy of organizations and web pages have a hierarchical structure and can be represented as trees, and there are different types of bonds between atoms or molecules, such as ionic bonds and covalent bonds. Thus, to extract accurate and useful representation from graphs, designing graph representation learning architectures which take into account the characteristics of each graph structure is important.

During Ph.D., my current research interest is to answer the following question: *In an unsupervised environment, how to extract useful knowledge from relationships between objects?* In my work, I aim to achieve this by understanding the unique characteristics of graphs in various domains and using it as an inductive bias. Since there are various types of relationships in this world, the graph of each domain has its own unique characteristics. My work focuses on three representative graph structures among various kinds of graphs: *homogeneous graph, tree-like graph*, and *heterogeneous graph*. I try to answer the following questions: i) how to extract a latent representation of a node in the homogeneous graph?; ii) how to learn accurate representations from tree-like graphs?; iii) how to learn the intricate structure from different types of nodes and relations in heterogeneous graphs effectively and efficiently? Below I briefly describe my approaches to answer the above questions by designing domain-specific unsupervised graph representation learning frameworks.

Homogeneous graph. A homogeneous graph indicates there exists a single type of node and a single type of edge (relation) in graphs. A citation network whose nodes are papers and edges representing citation between papers is a typical example of homogeneous graphs. Many previous works tried to learn a low-dimensional latent representation of nodes using autoencoder framework. However, due to the partially trainable frameworks, they have a limited learning capability. To tackle this issue, we introduced a novel decoder layer to design a fully trainable autoencoder framework from understanding how the encoder of previous models works on the homogeneous graph. This work is included in Chapter. 3.

Tree-like graph. Among the many types of relationships, hierarchical relationships are one of the most common types in our world. For example, we can find hierarchies in the organizational structure of corporations and governments or in biological classifications between species. It is well known that graph underlying hierarchical relations between nodes show a tree-like structure. Existing methods usually extract a latent node representation of tree-like graph in Euclidean space as many machine learning methods did. However, due to the nature of the tree that the number of leaf nodes grows exponentially with the depth of the tree, recent analysis reveals that Euclidean space is not an appropriate space to learn tree-like graphs. Thus, we introduced a novel autoencoder framework operating in hyperbolic spaces that can be considered as a continuous version of the discrete tree to effectively learn representations of tree-like graphs. Also, we designed a self-attention mechanism that adopts the hyperbolic distance between node features. This work is included in Chapter. 4.

Heterogeneous graph. A heterogeneous graph is a network with multiple types of nodes and edges. For example, a citation network might have multiple types of nodes (papers, authors, institutions, and subjects of papers) and multiple types of edges (writing and be affiliated with). Most existing methods rely on the pre-defined composition of different types of nodes in advance of training a model to learn intricate structures from multiple types of nodes and edges. However, it is hard to know whether the compositions given in advance is conducive to effective learning. Most of heterogeneous graphs are given only sparse relationships between different node types. Motivated by this property, we proposed the concept of a virtual node that does not require any preprocessing step and can help to effectively learn the relation of diverse relations in heterogeneous graphs. This work is included in Chapter. 5.

Chapter 2

Representation Learning on Graph-Structured Data

2.1 Basic Introduction

A graph that can model the objects and interaction between them as nodes and edges respectively can be found everywhere in our world. Social interaction between people, citation relation between papers, recommendation system, chemical bonds between atoms, and biological classification between species are some of the representative examples. Therefore, understanding and extracting patterns from interaction between nodes in graphs is one of the long standing machine learning research fields. In the real world, since there exist tons of different types of interactions (relations), graphs can be classified into numerous types by structural and semantic properties. For instance, a network may have a single type of node and a single type of edge (e.g. homogeneous graph), or multiple types of nodes and edges (e.g. heterogeneous graph). In some cases, the network can have a tree structure if there exist hierarchical relations between nodes.

By building powerful models that can analyze relations between objects and understand the properties, we can get insights from real-world complex networks. For a very long time, attempts to learn graphs based on a mathematical basis have been actively made, and recently, it has become possible to extract useful information about the structure of graphs in various areas with the power of neural networks. In this chapter, we provide basic notations on graphs, traditional methodology such as graph statistic and spectral approaches, Graph Neural Networks (GNNs) model, and architectures of unsupervised learning model for graphs.

2.1.1 Notations

A graph can be represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ is a node set and $\mathcal{E}$ is an edge set. An adjacency matrix $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ encodes information of pairwise relations between nodes. A degree matrix $D \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is a diagonal matrix whose diagonal element $D_{ii} = \sum_j A_{ij}$ refers to the degree value of each node.

If the graph is undirected, since $(u, v), (v, u) \in \mathcal{E}$ for $u, v \in \mathcal{V}$, the adjacency matrix A is symmetric, while the graph is directed, the corresponding A is asymmetric. If the graph is weighted, the edge can have a continuous weights. In this case, the adjacency matrix is no longer binary matrix and can take any real-values.

2.2 Traditional Approaches

There exist numerous methods to understand and analyze the graphs. In this section, we will briefly review some early attempts among them such as graph statistics, neighborhood overlap, graph kernel, spectral approaches, and random walks on graphs. We will assume that the given graph is undirected for simplicity.

2.2.1 Graph Statistics

To extract features from graphs, the most straightforward way is utilizing statistics on graphs. The distribution of degree values about the connection of nodes or the frequency of small specific structures such as triangles helps to understand the characteristics of the network.

Node Centrality

Node centrality measures the importance of the node in the graph. There are various definitions of centrality depending on which criterion is used to measure the importance. Among them, we will explain betweenness centrality, closeness centrality, and eigenvector centrality.

i) Betweenness centrality [12] measures the importance of node using shortest path. If a node u appears many times on the shortest path between other nodes, than that node have a large centrality score $c(u)$ as follows:

$$c(u) = \sum_{s \neq u \neq t} \frac{\text{number of shortest paths between s and t containing u}}{\text{number of shortest paths between s and t}}. \tag{2.1}$$

ii) Closeness centrality [13] measures the importance of nodes using shortest path distance. If the shortest path distance between node u and all other node are small, then that node have a large centrality:

$$c(u) = \frac{1}{\sum_{u \neq v} \text{shortest path distance between u and v}}. \tag{2.2}$$

iii) Eigenvector centrality [14] measures the node importance by considering the importance of neighbors. Eigenvector centrality $c(u)$ defines the centrality as the sum of the centrality of the neighbor nodes:

$$c(u) = \frac{1}{\lambda} \sum_{v \in \mathcal{N}_G(u)} c(v), \tag{2.3}$$

where λ is a normalizing constant. The equation (2.3) can be reformulated as $\lambda \mathbf{c} = A\mathbf{c}$, where $\mathbf{c}$ is the centrality vector and A is the adjacency matrix. It can be observed that $\mathbf{c}$ is the eigenvector of the adjacency matrix. The eigenvector corresponding to the largest eigenvalue of the adjacency matrix is used for the centrality vector.

Clustering Coefficient

Clustering coefficient measures the how much the node of the graph are well connected together. The definition of local clustering coefficient [15] is:

$$c(u) = \frac{\text{number of edges among neighbor nodes}}{\binom{d_u}{2}} \in [0, 1], \tag{2.4}$$

where d_u is the degree value of node u. If the neighbor nodes in $\mathcal{N}_{\mathcal{G}}(u)$ are well connected with each other, then $c(u)$ have a large value.

2.2.2 Neighborhood Overlap

Although many methods are focusing on the prediction of node-wise or graph-wise properties, prediction of relation such as link prediction is also very important task of graph representation learning fields. When predicting the relation between two nodes, it is essential to consider how close they are. How many neighbors two nodes share with each other (neighborhood overlap) provides very important information to know the relationship between them. We will introduce local neighborhood overlap: Common neighbors, Jaccard's coefficient, and Adamic-Adar index, and global neighborhood overlap: Katz index. From now on, we define S_{ij} as the value about the relation between two nodes $v_i, v_j \in \mathcal{V}$.

Common Neighbors

Common neighbors measures the relation between two nodes as the number of shared neighbors as follows:

$$S_{ij} = |\mathcal{N}_{\mathcal{G}}(i) \cap \mathcal{N}_{\mathcal{G}}(j)|. \tag{2.5}$$

Common neighbors is one of the simplest methods and cannot measure the relation between two nodes if they are located far apart.

Jaccard's Coefficient

Jaccard's coefficient normalizes the closeness of neighbors of two target nodes into unit interval $[0, 1]$:

$$S_{ij} = \frac{|\mathcal{N}_{\mathcal{G}}(i) \cap \mathcal{N}_{\mathcal{G}}(j)|}{|\mathcal{N}_{\mathcal{G}}(i) \cup \mathcal{N}_{\mathcal{G}}(j)|}. \tag{2.6}$$

If the neighbor sets of two nodes $\mathcal{N}_{\mathcal{G}}(i), \mathcal{N}_{\mathcal{G}}(j)$ are identical, then $S_{ij} = 1$.

Adamic-Adar Index

Unlike common neighbors and Jaccard's coefficient, Adamic-Adar index [16] consider second order relations:

$$S_{ij} = \sum_{m \in \mathcal{N}_{\mathcal{G}}(i) \cap \mathcal{N}_{\mathcal{G}}(j)} \frac{1}{\log |\mathcal{N}_{\mathcal{G}}(m)|}. \tag{2.7}$$

Although two target nodes share many neighbors, if the neighbor $m \in \mathcal{N}_{\mathcal{G}}(i) \cap \mathcal{N}_{\mathcal{G}}(j)$ has a high degree value, then S_{ij} can have a small value. The intuition of Adamic-Adar index is that the common neighbors with large neighbors are less significant when considering the relation between two target nodes.

Katz Index

The limitation of local neighborhood overlap such as common neighbors, Jaccard's coefficient, and Adamic-Adar index is that if two nodes v_i, v_j do not share any neighbors, then S_{ij} is always 0. However, although the two nodes do not have local neighbor, they can locate in the same community or be connected later. Global neighborhood overlap overcomes this issue by considering the overall structure of the graph. Katz index [17] measures the relation between two nodes by counting the number of walks of any number:

$$S_{ij} = \sum_{k=1}^{\infty} \beta^k A_{ij}^k, \tag{2.8}$$

where A is the adjacency matrix and $\beta (0 < \beta < 1)$ is a discount factor. Since A_{ij}^k is equal to the number of walks of length k between v_i and v_j, Katz index can consider all walks between two nodes by the power of adjacency matrix. It can be noticed that Katz index assigns more weights β^k to the walks of short lengths.

2.2.3 Graph Kernel

Suppose that we want to classify multiple graphs. Then, it is essential to measure the structural similarity between the graphs. The well-known method to measure the similarity between the graphs is graph kernel method. The kernel methods measure the similarity between two graphs $\mathcal{G}_1, \mathcal{G}_2$ via kernel function $\mathcal{K}(\cdot, \cdot)$ that is equal to an inner product in Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$: $\mathcal{K}(\mathcal{G}_1, \mathcal{G}_2) = \langle \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) \rangle_{\mathcal{H}}$, where $\phi(\mathcal{G})$ is a feature representation of the graph.

Graphlet Kernel

Graphlets [18] are small, induced, and non-isomorphic subgraph structures. The key idea of graphlet kernel [19] is using bag-of-graphlet representation of the graph. By counting the occurrence of different graphlets, we can define graphlet count vector $f_{\text{graphlet}} \in \mathbb{R}^{n_k}$, where n_k is the number of graphlets and i-th component of f_{graphlet} indicates the frequency of occurrence of i-th graphlet. After computing the graphlet count vectors of two graphs $f_{\text{graphlet}}(\mathcal{G}_1), f_{\text{graphlet}}(\mathcal{G}_2)$, we normalize each feature vector. Then, we can define the kernel function of graphlet kernel as $\mathcal{K}_{\text{graphlet}}(\mathcal{G}_1, \mathcal{G}_2) = f_{\text{graphlet}}(\mathcal{G}_1)^T f_{\text{graphlet}}(\mathcal{G}_2)$. If two graphs share similar frequency of graphlet occurrence, then the result of kernel function is a high value. The limitation of graphlet kernel is that counting graphlets requires high computational burden, since counting the size of k graphlets for a graph $\mathcal{G}$ takes $|\mathcal{V}|^k$.

Weisfeiler-Lehman Kernel

Weisfeiler-Lehman (WL) kernel [20] is more efficient feature descriptor compared to graphlet kernel. The key idea of WL kernel is utilizing neighbor structure of each node by iteratively aggregating labels of neighbors. After deriving node-level features, WL kernel computes a graph-level feature by aggregating node-level features. To achieve this, WL kernel applies WL algorithm (color refinements) [21]:

1. Assign an initial color $c^0(v)$ to each node $v \in \mathcal{V}$. For most of graphs, we can simply assign the initial color following the degree value d_v.

2. Iteratively refine node label as $c^{i+1}(v) = \text{HASH}\big(\{c^i(v), \{c^i(u)\}_{u \in \mathcal{N}_\mathcal{G}(v)}\}\big)$, where HASH maps different inputs to different labels.

3. Repeat Step 2 K times and derive the final node label $c^K(v)$. Then $c^K(v)$ summarizes the structure of K-hop ego networks of each node v.

After finishing color refinement, we define a color count vector of the graph $f_{\text{WL}}(\mathcal{G})$ by counting the occurrence of colors. Then WL kernel computes the similarity between two graphs as $\mathcal{K}_{\text{WL}}(\mathcal{G}_1, \mathcal{G}_2) = f_{\text{WL}}(\mathcal{G}_1)^T f_{\text{WL}}(\mathcal{G}_2)$. Since the time complexity of WL kernel is linear in the number of edges $|\mathcal{E}|$, WL kernel is computationally efficient for sparse graphs.

2.2.4 Spectral Approaches

Spectral graph theory [22] focuses the behavior of spectrum of matrix that represents graph structure. From the graph spectrum, we can deduce the properties and structures of the graph. At first, we explain graph Laplacian, the most important graph matrix.

Graph Laplacian

Suppose that the graph $\mathcal{G}$ is undirected and its adjacency matrix and degree matrix are A and D, respectively. An essential operator in spectral graph theory is the graph Laplacian L, whose definition is $L = D - A$, and $L = U \Lambda U^T$ where the graph Laplacian can be diagonalized by the Fourier basis $U = [u_1, \ldots, u_{|\mathcal{V}|}] \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\Lambda = diag([\lambda_1, \ldots, \lambda_{|\mathcal{V}|})] \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ where $\{u_i\}_{i=1}^{|\mathcal{V}|}$ are the eigenvectors and $\{\lambda_i\}_{i=1}^{|\mathcal{V}|}$ are the non-negative eigenvalues of graph Laplacian ($0 = \lambda_1 \leq \ldots \leq \lambda_{|\mathcal{V}|}$). There exist two normalized version of L: symmetric graph Laplacian L_{sym} and random walk graph Laplacian L_{rw} are defined by $L_{sym} = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $L_{rw} = I_n - D^{-1}A$ respectively, where $I_n \in \mathbb{R}^{n \times n}$ denotes an identity matrix. Now, we provide some important properties of L:

1. L is symmetric and positive semi-definite.
2. $x^T Lx = \frac{1}{2}\sum_{u\in\mathcal{V}}\sum_{v\in\mathcal{V}} A_{uv}(x[u] - x[v])^2 \geq 0$ for all $x \in \mathbb{R}^{|\mathcal{V}|}$.
3. The smallest eigenvalue λ_1 is 0 and its corresponding eigenvector u_1 is the one vector $\mathbf{1}$.
4. The multiplicity of eigenvalue 0 is equal to the number of connected components in the graph.
5. The eigenvalues of graph Laplacian are the union of the spectra of each connected component.

Graph Cut

Suppose we want to find partitions $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k (\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k = \mathcal{V})$ of the graph such that maximizing the number of edges of intra-community and minimizing the number of edges of inter-community. To find the optimal partition, we define graph cut that minimizes:

$$\text{cut}(\mathcal{A}_1, ..., \mathcal{A}_k) = \frac{1}{2}\sum_{i=1}^{k} W(\mathcal{A}_i, \bar{\mathcal{A}}_i), \tag{2.9}$$

where $\bar{\mathcal{A}}_i$ is the complement of $\mathcal{A}_i$ and $W(\mathcal{A}_i, \bar{\mathcal{A}}_i) = \sum_{i\in\mathcal{A}_i, j\in\bar{\mathcal{A}}_i} A_{ij}$. However, directly minimizing (2.9) might generate implausible partitions since the cut value does not consider relative sizes between partitions.

To solve the problem of graph cut, RatioCut [23] and normalized cut (NCut) [24] consider the size of the partition as the number of nodes in partition and degree value, respectively. The definition of ratio cut and normalized cut are described as below:

$$\text{RatioCut}(\mathcal{A}_1, ..., \mathcal{A}_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(\mathcal{A}_i, \bar{\mathcal{A}}_i)}{|\mathcal{A}_i|}, \tag{2.10}$$

$$\text{NCut}(\mathcal{A}_1, ..., \mathcal{A}_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(\mathcal{A}_i, \bar{\mathcal{A}}_i)}{\text{vol}(\mathcal{A}_i)}, \tag{2.11}$$

where $|\mathcal{A}_i|$ is the number of nodes in the partition $\mathcal{A}_i$ and $\text{vol}(\mathcal{A}_i) = \sum_{u\in\mathcal{A}_i} d_u$ is the summation of degree value of nodes in partition $\mathcal{A}_i$.

Now we explain the relation between RatioCut, NCut and the spectrum of graph Laplacian. For simplicity we use the case of $k = 2$, where $\mathcal{A}$ and $\bar{\mathcal{A}}(\mathcal{A} \cup \bar{\mathcal{A}} = \mathcal{V})$. At first, we approximate RatioCut$(\mathcal{A}, \bar{\mathcal{A}})$ using graph Laplacian:

$$\min_{\mathcal{A}\subset\mathcal{V}} \text{RatioCut}(\mathcal{A}, \bar{\mathcal{A}}). \tag{2.12}$$

We rewrite the problem more convenient way by defining the vector $f \in \mathbb{R}^{|\mathcal{V}|}$ as follows:

$$f[u] = \begin{cases} \sqrt{\frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|}}, & \text{if } u \in \mathcal{A}, \\ -\sqrt{\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|}}, & \text{if } u \in \bar{\mathcal{A}}. \end{cases} \tag{2.13}$$

Now, we can represent (2.12) as the equation of graph Laplacian L:

$$f^T L f = \frac{1}{2}\sum_{u,v\in\mathcal{V}} A_{uv}(x[u] - x[v])^2 \tag{2.14}$$

$$= \sum_{u\in\mathcal{A}, v\in\bar{\mathcal{A}}} A_{uv}\Big(\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|} + \frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|}\Big)^2 \tag{2.15}$$

$$= \text{cut}(\mathcal{A}, \bar{\mathcal{A}})\Big(\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|} + \frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|} + 2\Big) \tag{2.16}$$

$$= \text{cut}(\mathcal{A}, \bar{\mathcal{A}})\Big(\frac{|\mathcal{A}| + |\bar{\mathcal{A}}|}{|\bar{\mathcal{A}}|} + \frac{|\mathcal{A}| + |\bar{\mathcal{A}}|}{|\mathcal{A}|}\Big) \tag{2.17}$$

$$= |\mathcal{V}|\text{RatioCut}(\mathcal{A}, \bar{\mathcal{A}}). \tag{2.18}$$

Since the vector f of (2.13) is orthogonal to the one vector $\mathbf{1}$ ($f \perp \mathbf{1}$) and $||f||^2 = |\mathcal{V}|$, we can rewrite (2.12) as follows:

$$\begin{aligned} \min_{\mathcal{A}\subset\mathcal{V}} \quad & f^T L f \\ \text{s.t.} \quad & f \perp \mathbf{1} \\ & \|f\|^2 = |\mathcal{V}|. \end{aligned} \tag{2.19}$$

However, since (2.19) is NP-hard problem due to the discrete nature, we have to relax the problem as follows:

$$\begin{aligned} \min_{f\in\mathbb{R}^{|\mathcal{V}|}} \quad & f^T L f \\ \text{s.t.} \quad & f \perp \mathbf{1} \\ & \|f\|^2 = |\mathcal{V}|. \end{aligned} \tag{2.20}$$

By Rayleigh-Ritz theorem [25], the solution of (2.20) is the eigenvector of the second smallest eigenvalue of L: u_2. After obtaining u_2, we can derive the partition by following the sign of the elements:

$$\begin{cases} i \in \mathcal{A}, & \text{if } u_2[i] \geq 0, \\ i \in \bar{\mathcal{A}}, & \text{if } u_2[i] < 0. \end{cases} \tag{2.21}$$

Now, we approximate NCut($\mathcal{A}, \bar{\mathcal{A}}$) using graph Laplacian:

$$\min_{\mathcal{A}\subset\mathcal{V}} \text{NCut}(\mathcal{A}, \bar{\mathcal{A}}). \tag{2.22}$$

The process of approximating NCut is similar to the case of the RatioCut. At first, we define the vector $f \in \mathbb{R}^{|\mathcal{V}|}$ as follows:

$$f[u] = \begin{cases} \sqrt{\frac{\text{vol}(\bar{\mathcal{A}})}{\text{vol}(\mathcal{A})}}, & \text{if } u \in \mathcal{A}, \\ -\sqrt{\frac{\text{vol}(\mathcal{A})}{\text{vol}(\bar{\mathcal{A}})}}, & \text{if } u \in \bar{\mathcal{A}}. \end{cases} \tag{2.23}$$

Then, we can rewrite (2.22):

$$\begin{aligned} \min_{\mathcal{A}\subset\mathcal{V}} \quad & f^T L f \\ \text{s.t.} \quad & Df \perp \mathbf{1} \\ & f^T D f = \text{vol}(\mathcal{V}). \end{aligned} \tag{2.24}$$

Then, we relax the above problem:

$$\begin{aligned} \min_{f\in\mathbb{R}^{|\mathcal{V}|}} \quad & f^T L f \\ \text{s.t.} \quad & Df \perp \mathbf{1} \\ & f^T D f = \text{vol}(\mathcal{V}). \end{aligned} \tag{2.25}$$

We once again transform the problem after substituting f:

$$\begin{aligned} \min_{g\in\mathbb{R}^{|\mathcal{V}|}} \quad & g^T D^{-1/2} L D^{-1/2} g \\ \text{s.t.} \quad & g \perp D^{1/2}\mathbf{1} \\ & \|g\|^2 = \text{vol}(\mathcal{V}), \end{aligned} \tag{2.26}$$

where $g = D^{1/2}f$. By Rayleigh-Ritz theorem, the solution of (2.26) is the eigenvector of the second smallest eigenvalue of symmetric graph Laplacian $L_{sym} = D^{-1/2}LD^{-1/2}$.

Spectral Clustering

The goal of spectral clustering [26] is partitioning the nodes of the graph into K clusters. The processes of spectral clustering are as follows:

1. Compute the symmetric graph Laplacian L_{sym}.
2. Obtain the eigenvectors $U = [u_1, ..., u_K] \in \mathbb{R}^{|\mathcal{V}|\times K}$ of the smallest K eigenvalue of L_{sym}.

3. Normalize the row of U to become a unit vector.

4. Conduct K-means clustering on each row of U, and obtain K clusters.

For a more detailed explanation of graph cut and spectral clustering, refer [27].

2.3 Node Embeddings I: Factorization and Random Walks

In this section, we describe the shallow node embedding methods based on matrix factorization and random walks on graphs.

2.3.1 Factorization-based Methods

Matrix factorization methods aim to obtain node embeddings by decomposing the graph matrix such as the adjacency matrix or graph Laplacian.

i) Laplacian Eigenmap [28] minimizes the following loss function to get the k - dimensional node embedding $H \in \mathbb{R}^{|\mathcal{V}| \times k}$:

$$\begin{aligned} \min_{H \in \mathbb{R}^{|\mathcal{V}| \times k}} \quad & tr(H^T L H) \\ \text{s.t.} \quad & H^T D H = I. \end{aligned} \tag{2.27}$$

The solution of the above problem is the eigenvectors of the k smallest eigenvalues of graph Laplacian. If the two nodes are connected by edges with large weight, then they will have similar embeddings, since (2.27) can be reformulated as $tr(H^T L H) = \sum_{i,j} \|h_i - h_j\|^2 A_{ij}$, where $h_i \in \mathbb{R}^k$ is the node embedding of i-th node (i-th row of H).

ii) Graph factorization [29] decomposed the adjacency matrix to obtain node embeddings H:

$$\min_{H \in \mathbb{R}^{|\mathcal{V}| \times k}} \quad \|HH^T - A\|_2^2. \tag{2.28}$$

Since the above problem can be seen as measure the distance between the adjacency matrix and the inner product between node embeddings, graph factorization targets first-order similarity.
iii) GraRep [30] aims to learn node embeddings that aware higher-order similarity by decomposing the power of the adjacency matrix A^p:

$$\min_{H \in \mathbb{R}^{|\mathcal{V}| \times k}} \quad \|HH^T - A^p\|_2^2. \tag{2.29}$$

2.3.2 Random Walk-based Methods

Unlike graph factorization and GraRep, random walk methods obtain node embeddings in a stochastic way. If two nodes are occur frequently in each other's short random walks, then they will share similar embeddings.
i) DeepWalk [31] aims to obtain node embeddings that aware random walks. At first, DeepWalk conducts short random walks starting from each node in the graphs and collects the multiset $\mathcal{N}_{rw}(u)$ whose elements are visited nodes during random walks starting from node u. Then DeepWalk optimizes the node embeddings of node u, h_u, by maximizing the following loss function L:

$$L = -\sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{rw}(u)} \log(P(v|h_u)), \tag{2.30}$$

where $P(v|h_u) = \frac{\exp(h_u^T h_v)}{\sum_{l=1}^{|\mathcal{V}|} \exp(h_u^T h_l)}$. $P(v|h_u)$ is the probability that node u and v are co-occur on the random walks. However, optimizing the loss function (2.30) is computationally inefficient, since summing over every node in the graph is required. To solve this issue, DeepWalk introduced hierarchical softmax.
ii) When conducting random walks, node2vec [32] applies two types of random walks based on breadth first search (BFS) and depth first search (DFS). By adjusting hyperparmeters to balance between BFS and DFS, the node embeddings of node2vec can aware both of local structure and global structure of the graph. Also, node2vec

approximates the loss function of DeepWalk (2.30) by introducing negative sampling strategy as follows:

$$\log\Big(\frac{\exp(h_u^T h_v)}{\sum_{l=1}^{|\mathcal{V}|}\exp(h_u^T h_l)}\Big) \approx \log(\sigma(h_u^T h_v)) - \sum_{i=1}^{m}\log(\sigma(h_u^T h_{n_i})), n_i \sim P_{\mathcal{V}}, \tag{2.31}$$

where σ is the sigmoid function and $P_{\mathcal{V}}$ denotes random distribution over nodes in the graph. By sampling m random negative samples, node2vec can avoid the case of normalizing against all nodes in the graph.

2.4 Node Embeddings II: Graph Neural Networks

In this section, we explain Graph Neural Networks, the powerful deep node embedding method.

2.4.1 Overview of Framework

About 15 years ago, the concepts of Graph Neural Networks (GNNs) were first proposed [33, 34]. The main purpose of GNNs is learning vector representation of a node h_i or a graph $h_{\mathcal{G}}$ by leveraging both graph structure and node features. The majority of modern GNNs adopts the architecture of Message Passing Neural Networks (MPNNs) [9] where each node update their messages (node features or representation) by exchanging messages with its immediate neighbor nodes $\mathcal{N}_{\mathcal{G}}$ iteratively as described in Eq. (2.32).

$$h_i^{l+1} = \text{COMBINE}\Big(h_i^l, \text{AGGREGATE}\big(h_j^l | j \in \mathcal{N}_{\mathcal{G}}(i)\big)\Big), \tag{2.32}$$

where h_i^l is the representation of node v_i at l-th layer. We set $h_i^0 = x_i$, where x_i is the feature of v_i. There are two major functions in the architecture of MPNNs: COMBINE and AGGREGATE. AGGREGATE is a function for aggregating messages from the neighbor nodes. COMBINE is a function for updating the node representation by taking its own representation from the previous layer and the aggregated messages from neighbors as inputs. Depending on which of the AGGREGATE and COMBINE functions are used, we can classify GNNs models. In a case of obtaining a vector representation of a graph for graph classification, the READOUT function which aggregates the representations of every node in the graphs at the final layer is applied:

$$h_{\mathcal{G}} = \text{READOUT}(\{h_i^L | i \in \mathcal{V}\}). \tag{2.33}$$

Many methods apply some simple permutation invariant functions as READOUT such as sum pooling or max pooling [35, 36].

2.4.2 Representative Models

Now, we explain three most well-known GNNs models: Graph Convolutional Networks, Graph Attention Networks, GraphSAGE, Jumping Knowledge Networks, and Graph Isomorphism Networks.

Graph Convolutional Networks

Graph Convolutional Networks (GCN) [37] integrates AGGREGATE and COMBINE functions by applying element-wise mean pooling on the representations of itself and neighbor nodes as follows:

$$h_i^{l+1} = \text{ReLU}\Big(W \cdot \text{MEAN}\big\{h_j^l | j \in \mathcal{N}_{\mathcal{G}}(i) \cup \{i\}\big\}\Big), \tag{2.34}$$

where W denotes a trainable weight matrix. When aggregating messages, GCN layer assigns an equal (degree-normalized) weight to every messages of neighbors including itself. GCN was applied to semi-supervised node classification task and showed superior performances on citation networks.

Graph Attention Networks

Graph Attention Networks (GAT) [38] has an integrated step of AGGREGATE and COMBINE functions similar to GCN [37]. The main difference from GCN is that, unlike isotropic aggregation in GCN, GAT assigns importance to each neighboring node and performs anisotropic aggregation as follows:

$$h_i^{l+1} = \text{ReLU}\Big(W \cdot \sum_{j \in \mathcal{N}_{\mathcal{G}}(i) \cup \{i\}} \alpha_{ij}^l h_j^l\Big), \tag{2.35}$$

$$\alpha_{ij}^l = \frac{\exp(\text{LeakyReLU}(a^T[W \cdot h_i^l | W \cdot h_j^l])}{\sum_{k \in \mathcal{N}_{\mathcal{G}}(i) \cup \{i\}} \exp(\text{LeakyReLU}(a^T[W \cdot h_i^l | W \cdot h_k^l])}), \tag{2.36}$$

where α_{ij}^l denotes an attention score at l-th layer representing the importance of node v_j to node v_i, and a is a weight vector. There also exists a multi-head version of GAT to stabilize the learning process. Due to the attention score, GAT is more interpretable than other GNNs models, and shows superior performances on transductive and inductive node classification tasks.

GraphSAGE

Graph Sample and Aggregate (GraphSAGE) [39] applies mean pooling, max pooling, or LSTM function as AGGREGATE function and concatenation as COMBINE function. The below is GraphSAGE in a case of max pooling aggregator:

$$h_i^{l+1} = \text{ReLU}\Big(W \cdot \big[h_i^l | \text{MAX}\{\text{ReLU}(W \cdot h_j^l) | j \in \mathcal{N}_{\mathcal{G}}(i)\}\big]\Big). \tag{2.37}$$

For learning on large graphs, each layer of GraphSAGE does not aggregate messages from every neighbor nodes and only aggregate messages of sampled neighbors for each node. Due to the sampling strategy, GraphSAGE shows the improved scalability and runtime.

Jumping Knowledge Networks

Each node of the graph has its own local structure. For instance, some nodes who are connected to many neighbors are hubs, while some nodes are isolated nodes or have a few neighbors. So, when we apply the same number of message passing layers to every nodes in the graph, some nodes might lose their own meaning due to the mixing of too much messages from neighbors. On the other hands, some nodes can suffer limited information aggregation due to the extremely sparse local structures. To circumvent this issue, when computing the final representation of each node h_i before classifier, Jumping Knowledge Networks (JK-Nets) [40] aggregates every intermediate representations $h_i^0, h_i^1, ..., h_i^L$ to let the model adapts the effective range of neighbor of each node as follows:

$$h_i = f_{\text{JK}}(h_i^0, h_i^1, \cdots, h_i^L), \tag{2.38}$$

where f_{JK} denotes the aggregation function for intermediate representations such as concatenation, max-pooling, or LSTM-attention. By making the model to consider adaptive neighbor size for each node, JK-Nets showed the improved transductive and inductive node classification performances.

Graph Isomorphism Networks

One of the goal of graph learning model is mapping two nodes to the same location in the representation space, if they have an identical node feature and same local structures. Thus, to learn the same representation for the same subgraph structure, AGGREGATE function should be injective. However, max aggregation of GraphSAGE and mean aggregation of GCN are not injective, which may limit their expressive power. To make the model injective, Graph Isomorphism Networks (GIN) [41] proposed sum aggregation as follows:

$$h_i^{l+1} = \text{MLP}^l\Big((1+\epsilon^l)h_i^l + \sum_{j \in \mathcal{N}_\mathcal{G}(i)} h_j^l\Big), \tag{2.39}$$

where MLP is multi-layer perceptrons and ϵ is for discriminating the representation of itself from the those of neighbors. Since sum aggregator is injective, GIN can be more powerful model compared to GCN or GraphSAGE. GIN shows the improved graph classification performances on social networks and biochemical molecules compared to graph kernel [20] and diffusion-based graph convolution method [42]. For more comprehensive explanation about graph neural networks, refer to [43].

2.5 Learning in Unsupervised Environments

Unsupervised representation learning on graph-structured data is long-lasting important problem in machine learning fields. The earliest attempts were focused on dimensionality reduction that tries to learn low-dimensional representation of graphs. Some representative works are Multi-Dimensional Scaling [44], Isometric Mapping [45], and Laplacian Eigenmaps [28]. Then, methods of matrix factorization on graph shift operator [29,30,46] and random-walk on graphs [31,32,47,48] were proposed. Recently, due to its representation power and the surge of research on unsupervised (self-supervised) learning, GNNs are de facto models for unsupervised graph representation learning. There exist numerous deep graph unsupervised learning models based on GNNs and they can be classified into two categories: predictive coding and contrastive coding. In the below, we explain some predictive and contrastive coding methods utilizing GNNs architectures.

2.5.1 Predictive Coding

The predictive learning methods on graph-structured data aim to train the encoder f using the input data as supervisory signals. The representative predictive learning architecture is autoencoder [49] that is composed of the encoder f that maps the input to the low-dimensional latent space and the decoder g that maps the representation of latent space to the reconstruction of the input (supervisory signal from input data). The graph autoencoder models can be classified according to which part of the input data is used as a supervisory signal (or reconstruction target): 1) feature reconstruction, 2) structure reconstruction.

Autoencoders

i) Feature reconstruction: Marginalized Graph AutoEncoder (MGAE) [50] reconstructs the node attributes from the corrupted node attributes. After randomly corrupting node

attributes, MGAE takes that as an input of stacked encoder based on GCN [3].
ii) Structure reconstruction: Variational Graph AutoEncoder (VGAE) [51] is the earliest attempts to apply graph neural networks as an encoder of autoencoder frameworks. After obtaining latent representation from encoders composed of GCN [3] layer, VGAE reconstruct the graph structure (adjacency matrix) using the inner-product decoder. VGAE achieves improved link prediction performances compared to random-walk model. Adversarially Regularized Variational Graph Autoencoder (ARVGA) [52] adds adversarial regularization to VGAE model [51]. By regularizing the latent space, ARVGA makes the latent representations follow a prior distribution and achieves robust representation. Semi-Implicit Graph Variational AutoEncoder (SIG-VAE) [53] applies semi-implicit hierarchical variational distribution to VGAE model [51] along with Bernoulli-Poisson link decoder.

2.5.2 Contrastive Coding

The contrastive learning on graph-structured data has been greatly influenced by the advancements in self-supervised learning in image and language domains. After generating two views from single graph, the goal of contrastive learning is maximizing agreements between similar semantic information (positive samples), while minimizing agreements between non-similar semantic information (negative samples) [54]. We briefly introduce graph augmentation for generating multiple views of graphs, and some representative works.

Graph Augmentation

Due to the irregular nature of graphs, it is extremely difficult to directly apply image augmentation data to the graph domain. The early attempts of graph augmentation is heuristics such as node dropping, edge perturbation, attribute masking, and subgraph sampling [54, 55]. By randomly perturbing graph structures such as dropping a fraction of nodes and their connected edges (this can be considered as cutout [56] on visual domain), adding or removing a fraction of edges, and sampling subgraphs by graph diffusion on seed nodes, contrastive learning aims to learn encoder f robust on structure perturbation. There also exist attempts to perturb node attributes by randomly masking elements to zero or node attributes shuffling [1].

Representative Models

i) Deep Graph Infomax (DGI) [1]: DGI is the first attempt that introduces contrastive learning to GNNs. At first, DGI generates a negative view of graph $\tilde{\mathcal{G}}$ by randomly shuffling node feature matrix. To learn the representation of each node, DGI applies GCN [37] or GraphSAGE [39] as the encoder network f. The architecture of DGI shares the same encoder network for both the original graph and the corrupted graph to learn the representation of each node. h_i and $\tilde{h}_i$ on node $v_i \in \mathcal{V}$ denote the outputs of the encoder network for the original graph and the corrupted graph, respectively. DGI extracts global summary vector s of the original graph by applying mean pooling $\mathrm{s} = \sigma(\frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|} h_i)$, where σ denotes the logistic sigmoid function. Then, DGI utilizes a contrastive objective with binary cross entropy loss function between positive samples (h_i, s) and negative samples $(\tilde{h}_i, \mathrm{s})$ as below:

$$L = \frac{1}{2|\mathcal{V}|}\Big(\sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_{\mathcal{G}}[\log D(h_i, \mathrm{s})] + \sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_{\tilde{\mathcal{G}}}[\log(1 - D(\tilde{h}_i, \mathrm{s}))]\Big), \tag{2.40}$$

where $D(h_i, \mathrm{s}) = \sigma(h_i^T W \mathrm{s})$ denotes the discriminator function which is a bilinear network (W is a learnable matrix). Maximizing the objective function L is equal to maximize the mutual information between the representation from the original graph h_i and the global summary vector s from the original graph. By conducting a local-global contrastive framework, DGI can learn the node representation that considers not only the local neighbor of each node but also the overall graph structure. On node classification tasks, DGI shows competitive performances compared to semi-supervised learning models.

ii) MVGRL [57] generates two views that consider the local structure of each graph

and the global structure of the graph, respectively. For generating local view, MVGRL randomly dropped a fraction of edges of the original graph. For global view, MVGRL derives diffusion matrix $\mathbf{S}$ by conducting graph diffusion [58] such as personalized pagerank or heat kernel. The framework of MVGRL shares the same encoder network for both the local view and the global view to learn both locality-aware and global-aware node representations. Thus, MVGRL also contrast between local view and global view like DGI [1], Both on node classification and graph classification tasks, MVGRL showed superior performances compared to existing unsupervised graph learning methods and graph kernel methods.
iii) GraphCL [55] contrasts two graph-wise representations. They proposes four graph augmentation methods: node dropping, edge perturbation, attribute masking, and subgraph sampling. After adopting the framework of SimCLR [59], one of the representative contrastive learning model on visual domain, they introduces graph contrastive learning framework. By applying the proposed augmentation methods on two graph domains, biochemical molecules and social networks, they empirically shows the role of graph augmentation on each domain. On semi-supervised learning, unsupervised representation learning, adversarial robustness, and transfer learning, they showed the validity of graph contrastive learning for GNNs pre-training.

2.6 Applications

After learning the node representation, machine learning models for graphs are validated through applications such as classification or relation prediction.

2.6.1 Classifications

Node Classification

The goal of node classification is predict the label of each node. Under semi-supervised conditions, nodes of the graph are split into training set, validation set, and test set.

In training stage, the model is trained by the node labels of training set. The best model is chosen by measuring the accuracy or loss value of validation set, and the chosen model predicts the label of the test node set that was not seen during training stage. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{C}|$-class node classification task aims to learn GNNs classifier $f : v \to \{1, 2, ..., |\mathcal{C}|\}$ that assigns labels to nodes $v \in \mathcal{V}$ in the graph. This work has been actively researched for a long time, and there exist many attempts to learn accurate node representations of high homophily [3, 60, 61] graphs that have a high probability that the two connected nodes share the same ground truth label. Recently, some approaches were proposed to accurately classify the node of high heterophily [62, 63] graphs that have a high probability that the edge is connect the two nodes with different labels.

Subgraph Classification

Given subgraphs $\mathcal{S} = \{S_1, ..., S_n\}$, $|\mathcal{C}|$-class subgraph classification task aims the GNNs model to learn a function $f : \mathcal{S} \to \{1, 2, ..., |\mathcal{C}|\}$ that assigns labels to subgraphs within a graph $\mathcal{G}$. This task has not been studied more actively than the node-wise or graph-wise tasks, and a representative work tries to classify protein subgraphs in a human protein-protein interaction network [64].

Graph Classification

Graph classification is conducted on the vector representation of the graph after aggregating representation of every nodes of each graph. Given a set of graphs $\mathbb{G} = \{\mathcal{G}_1, ..., \mathcal{G}_n\}$, $|\mathcal{C}|$-class graph classification task aims to learn a function that maps each graph to its corresponding labels: $f : \mathbb{G} \to \{1, 2, ..., |\mathcal{C}|\}$. Numerous graph pooling models [35,65,66] and GNNs with improved representation power [41, 67, 68] achieves superior performances on classifying social networks and biochemical molecules.

2.6.2 Link Prediction

Link prediction task is for predicting the relations between nodes. The probability of the edge between two nodes is measured using two node representations. The edges of the graph is split into training set, validation set, and test set. The model is trained using the edges of training set. The best model is chosen by measuring the prediction accuracy or loss value of validation edge set, and the chosen model predict the relation probability of the test edge set that was not seen during training stage. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, link prediction task aims to learn a mapping function that computes the probability of edge existence $f : (v_i, v_j) \rightarrow e_{ij}$, where e_{ij} indicates the edge probability score of a node pair (v_i, v_j). By leveraging the representation power of GNNs, recent attempts [51, 69, 70] show superior performances on link prediction tasks compared to heuristics such as common neighbors, Adamic-Adar index [16], or Katz index [17].

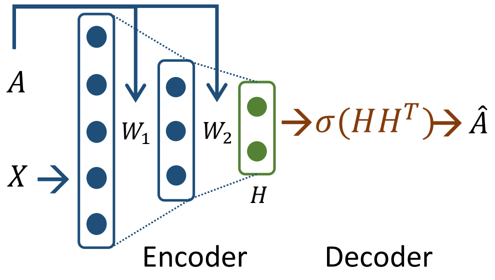
Chapter 3

Autoencoder Architecture for Homogeneous Graphs

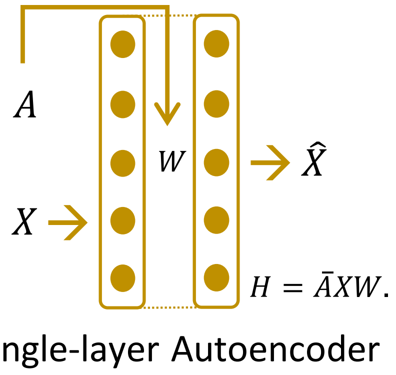
3.1 Overview

A graph, which consists of a set of nodes and edges, is a powerful tool to seek the geometric structure of data. There are various applications using graphs in the machine learning and data mining fields such as node clustering [26], dimensionality reduction [28], social network analysis [71], chemical property prediction of a molecular graph [10], and image segmentation [24]. However, conventional methods for analyzing a graph have several problems such as low computational efficiency due to eigendecomposition or singular value decomposition, or only showing a shallow relationship between nodes.

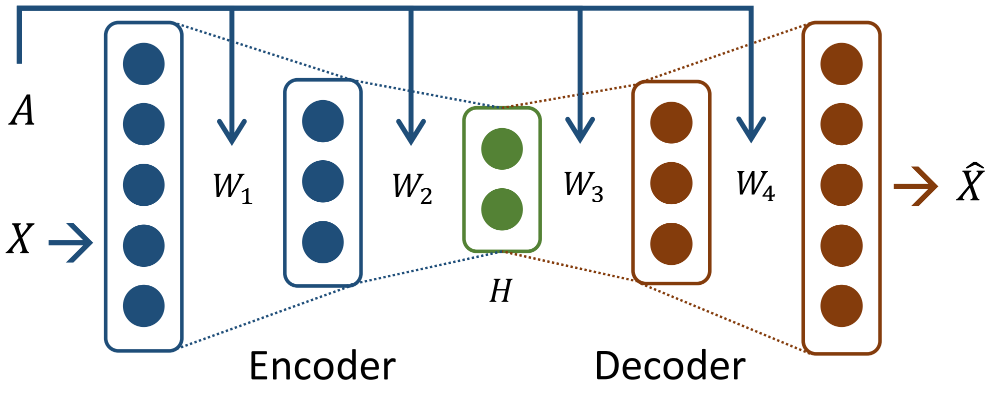
In recent years, an emerging field called *geometric deep learning* [11], generalizes deep neural network models to non-Euclidean domains such as meshes, manifolds, and graphs [3, 72, 73]. Among them, finding deep latent representations of geometrical structures of graphs using an autoencoder framework is getting growing attention. The first attempt is VGAE [51] which consists of a Graph Convolutional Network (GCN) [3] encoder and a matrix outer-product decoder as shown in Figure 3.1 (a). As a variant of VGAE, ARVGA [52] has been proposed by incorporating an adversarial approach to VGAE. However, VGAE and ARVGA were designed to reconstruct the



(a) VGAE [51]



(b) MGAE [50]



(c) Proposed autoencoder

Figure 3.1: Architectures of existing graph convolutional autoencoders and proposed one. A, X, H and W denote the affinity matrix (structure of graph), node attributes, latent representations and the learnable weight of network respectively.

affinity matrix A instead of node feature matrix X. Hence, the decoder part cannot be learnable, therefore, the graphical feature cannot be used at all in the decoder part. These facts can degrade the capability of graph learning. Following that, MGAE [50] has been proposed, which uses stacked single layer graph autoencoder with linear activation function and marginalization process as shown in Figure 3.1 (b). However, since the MGAE reconstructs the feature matrix of nodes without hidden layers, it cannot manipulate the dimension of the latent representation and performs a linear mapping. This is a distinct limitation in finding a latent representation that clearly reveals the structure of the graph.

To overcome the limitation of the existing graph convolutional autoencoders, in this chapter, we propose a novel graph convolutional autoencoder framework which has symmetric autoencoder architecture and uses both graph and node attributes in both the encoding and decoding processes as illustrated in Figure 3.1 (c). Our design of the decoder part is motivated from the analysis in a recent paper [74], that the encoder of VGAE [51] can be interpreted as a special form of Laplacian smoothing [75] that computes the new representation of each node as a weighted local average of neighbors and itself. This interpretation has inspired us to design a decoder to perform *Laplacian sharpening*, which is a counterpart of Laplacian smoothing. To realize a decoder to do Laplacian sharpening, we express Laplacian sharpening in the form of Chebyshev polynomial and newly reformulate it in a numerically stable form by utilizing a signed graph [76].

In computer vision fields, there is a popular assumption that, even though image datasets are high-dimensional in their ambient spaces, they usually reside in multiple low-dimensional subspaces [77]. Thus, especially for image clustering tasks, we apply the concept of subspace clustering, which has such an assumption about the input data in its own definition, to our graph convolutional autoencoder framework. Specifically, to find a latent representation and a latent affinity matrix simultaneously, we merge a subspace clustering cost function into the reconstruction cost of the autoencoder. Contrary to the conventional subspace clustering cost function [78,79], we could derive a computationally efficient cost function.

The main contributions of this work are summarized as follows:

- We propose the first completely symmetric graph convolutional autoencoder which utilizes both the structure of the graph and node attributes through the whole encoding-decoding process.

- We derive a new numerically stable form of decoder preventing the numerical instability of the neural network.
- We design a computationally efficient subspace clustering cost to find both latent representation and a latent affinity matrix simultaneously for image clustering tasks.

In experiments, the validity of the proposed components is shown by doing ablation experiments on our architecture and cost function. Also, the superior performance of the proposed method is validated by comparing it with the state-of-the-art methods and visualizing the graph clustered by our framework.

3.2 Preliminaries

3.2.1 Spectral Convolution on Graphs

A spectral convolution on a graph [80] is the multiplication of an input signal $x \in \mathbb{R}^n$ with a spectral filter $g_\theta = diag(\theta)$ parameterized by the vector of Fourier coefficients $\theta \in \mathbb{R}^n$ as follows:

$$g_\theta * x = U g_\theta U^T x, \tag{3.1}$$

where U is the matrix of eigenvectors of the symmetric graph Laplacian $L_{sym} = U \Lambda U^T$. $U^T x$ is the graph Fourier transform of the input x, and g_θ is a function of the eigenvalues of L_{sym}, i.e., $g_\theta(\Lambda)$, where Λ is the diagonal matrix of eigenvalues of L_{sym}. However, this operation is inappropriate for large-scale graphs since it requires an eigendecomposition to obtain the eigenvalues and eigenvectors of L_{sym}. To avoid computationally expensive operations, the spectral filter $g_\theta(\Lambda)$ was approximated by K^{th} order Chebyshev polynomials in previous works [81]. By doing so, the spectral convolution on the graph can be approximated as

$$g_\theta * x \approx U \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) U^T x = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}_{sym}) x, \tag{3.2}$$

where $T_k(\cdot)$ and θ' denote the Chebyshev polynomials and a vector of the Chebyshev coefficients respectively. $\tilde{\Lambda}$ is $\frac{2}{\lambda_{max}}\Lambda - I_n$, λ_{max} denotes the largest eigenvalue of L_{sym} and $\tilde{L}_{sym}$ is $U\tilde{\Lambda}U^T = \frac{2}{\lambda_{max}}L_{sym} - I_n$. The approximated model above is used as a building block of a convolution on graphs in [82].

In the GCN [3], the Chebyshev approximation model was simplified by setting $K = 1$, $\lambda_{max} \approx 2$ and $\theta = \theta'_0 = -\theta'_1$. This makes the spectral convolution simplified as follows:

$$g_\theta * x \approx \theta(I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x. \tag{3.3}$$

However, repeated application of $I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ can cause numerical instabilities in neural networks since the spectral radius of $I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is 2, and the Chebyshev polynomials form an orthonormal basis when its spectral radius is 1. To circumvent this issue, the GCN uses renormalization trick:

$$I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \to \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tag{3.4}$$

where $\tilde{A} = A + I_n$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Since adding self-loop on nodes to an affinity matrix cannot affect the spectral radius of the corresponding graph Laplacian matrix [83], this renormalization trick can provide a numerically stable form of $I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ while maintaining the meaning of each elements as follows:

$$(I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})_{ij} = \begin{cases} 1 & i = j \\ A_{ij}/\sqrt{D_{ii}D_{jj}} & i \neq j \end{cases} \tag{3.5}$$

$$(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})_{ij} = \begin{cases} 1/(D_{ii} + 1) & i = j \\ A_{ij}/\sqrt{(D_{ii} + 1)(D_{jj} + 1)} & i \neq j. \end{cases} \tag{3.6}$$

Finally, the forward-path of the GCN can be expressed by

$$H^{(m+1)} = \xi(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(m)}\Theta^{(m)}), \tag{3.7}$$

where $H^{(m)}$ is the activation matrix in the m^{th} layer and $H^{(0)}$ is the input nodes' feature matrix X. $\xi(\cdot)$ is a nonlinear activation function like ReLU($\cdot$) = $\max(0, \cdot)$, and $\Theta^{(m)}$ is a trainable weight matrix. The GCN presents a computationally efficient convolutional process (given the assumption that $\tilde{A}$ is sparse) and achieves an improved accuracy over the state-of-the-art methods in semi-supervised node classification task by using features of nodes and geometric structure of graph simultaneously.

3.2.2 Laplacian Smoothing

Li et al. [74] demystify GCN [3] and show that GCN is a special form of Laplacian smoothing [75]. Laplacian smoothing is a process that calculates a new representation of the input as a weighted local average of its neighbors and itself. When we add a self-loop on the nodes, the affinity matrix becomes $\tilde{A} = A + I_n$ and the degree matrix becomes $\tilde{D} = D + I_n$. Then, the Laplacian smoothing equation is given as follows:

$$x_i^{(m+1)} = (1-\gamma)x_i^{(m)} + \gamma \sum_j \frac{\tilde{A}_{ij}}{\tilde{D}_{ii}} x_j^{(m)}, \tag{3.8}$$

where $x_i^{(m+1)}$ is the new representation of $x_i^{(m)}$, and γ $(0 < \gamma \leq 1)$ is a regularization parameter which controls the importance between itself and its neighbors. We can rewrite the above equation in a matrix form as follows:

$$\begin{aligned} X^{(m+1)} &= (1-\gamma)X^{(m)} + \gamma \tilde{D}^{-1}\tilde{A}X^{(m)} \\ &= X^{(m)} - \gamma(I_n - \tilde{D}^{-1}\tilde{A})X^{(m)} \\ &= X^{(m)} - \gamma \tilde{L}_{rw} X^{(m)}. \end{aligned} \tag{3.9}$$

If we set $\gamma = 1$ and replace $\tilde{L}_{rw}$ with $\tilde{L}_{sym}$, then Eq. (3.9) is changed into $X^{(m+1)} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X^{(m)}$ and this equation is the same as the renormalized version of spectral convolution in Eq. (3.7). From the above interpretation, Li et al. explain that the superior performance of GCN in semi-supervised node classification task is due to Laplacian smoothing which makes the features of nodes in the same clusters become similar.

3.3 Methodology

In this section, we propose a novel graph convolutional autoencoder framework, named as GALA (Graph convolutional Autoencoder using LAplacian smoothing and sharpening). In GALA, there are M layers in total, from the first to $\frac{M}{2}$th layers for the encoder and from the $\left(\frac{M}{2}+1\right)$th to Mth layers for the decoder where M is an even number. The encoder part of GALA is designed to perform the computationally efficient spectral convolution on the graph with a numerically stable form of Laplacian smoothing in the Eq. (3.7) [3]. Along with this, its decoder part is designed to be a special form of Laplacian sharpening [75], unlike the existing VGAE-related algorithms. By this decoder part, GALA reconstructs the feature matrix of nodes directly, instead of yielding an affinity matrix as in the existing VGAE-related algorithms whose decoder parts are incomplete. Furthermore, to enhance the performance of image clustering, we devise a computationally efficient subspace clustering cost term which is added to the reconstruction cost of GALA.

3.3.1 Laplacian Sharpening

Because the encoder performs Laplacian smoothing that makes the latent representation of each node similar to those of its neighboring nodes, we design the decoder part to perform Laplacian sharpening as the counterpart of Laplacian smoothing. Laplacian sharpening is a process that makes the reconstructed feature of each node farther away from the centroid of its neighbors, which accelerates the reconstruction along with the reconstruction cost and is governed by

$$x_i^{(m+1)} = (1+\gamma)x_i^{(m)} - \gamma \sum_j \frac{A_{ij}}{D_{ii}} x_j^{(m)}, \tag{3.10}$$

where $x_i^{(m+1)}$ is the new representation of $x_i^{(m)}$, and γ is the regularization parameter which controls the importance between itself and its neighbors. The matrix form of Eq.

(3.10) is given by

$$\begin{aligned} X^{(m+1)} &= (1+\gamma)X^{(m)} - \gamma D^{-1}AX^{(m)} \\ &= X^{(m)} + \gamma(I_n - D^{-1}A)X^{(m)} \\ &= X^{(m)} + \gamma L_{rw}X^{(m)}. \end{aligned} \tag{3.11}$$

Analogous to the encoder, we set $\gamma = 1$ and replace L_{rw} with L_{sym}. Similar to Eq. (3.3), we can express Laplacian sharpening in the form of Chebyshev polynomial and simplify it with $K = 1$, $\lambda_{max} \approx 2$, and $\theta = \frac{1}{2}\theta'_0 = \theta'_1$. Then, a decoder layer can be expressed by

$$H^{(m+1)} = \xi((2I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})H^{(m)}\Theta^{(m)}), \tag{3.12}$$

where $H^{(m)}$ is the matrix of the activation in the m^{th} layer, $2I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is a special form of Laplacian sharpening, $\xi(\cdot)$ is the nonlinear activation function like ReLU$(\cdot)$ = $\max(0,\cdot)$, and $\Theta^{(m)}$ is a trainable weight matrix. However, since the spectral radius of $2I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is 3, repeated application of this operator can be numerically instable. Hence, as GCN finds a numerically stable form of Chebyshev polynomials, we have to find a numerically stable form of Laplacian sharpening while maintaining its meaning.

3.3.2 Numerically Stable Laplacian Sharpening

To find a new representation of Laplacian sharpening whose spectral radius is 1, we use a signed graph [76]. A signed graph is denoted by $\Gamma = (\mathcal{V}, \mathcal{E}, \hat{A})$ which is induced from the unsigned graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where each element in $\hat{A}$ has the same absolute value with A, but its sign is changed into minus or keeps plus. The degree matrix of the signed graph Γ is denoted by $\hat{D}$ which is obtained from $\hat{A}$. In the signed graph, a problem occurs when calculating the degree matrix $\hat{D}$ by the conventional way that may cancel the mixed signed weights in summation and so fails to yield the degree value representing the connectivity of a node to its neighbors. Thus, by following the practice for signed graphs, we calculate the degree of each node by $\hat{D}_{ii} = \sum_j |\hat{A}_{ij}|$ that has the same value (degree of connectivity) as in the unsigned graph. By using $\hat{A}$ and $\hat{D}$, we can construct an unnormalized graph Laplacian $\hat{L} = \hat{D} - \hat{A}$ and symmetric graph Laplacian $\hat{L}_{sym} = I_n - \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ of the signed graph. From Theorem 1 of [76], the range of the eigenvalue of $\hat{L}_{sym}$ is $[0, 2]$, thus the spectral radius of $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is 1 for any choice of $\hat{A}$. Using this result, instead of Eq. (3.12), we use a numerically stable form of Laplacian sharpening with spectral radius of 1, given by

$$H^{(m+1)} = \xi(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(m)}\Theta^{(m)}). \tag{3.13}$$

The remaining issue is to choose $\hat{A}$ induced from A so that $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ maintains the meaning of each element of $2I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ in Eq. (3.12). To achieve this, we map all weights of the unsigned A to negative weights and adding a self-loop with a weight value 2 to each node, that is, $\hat{A} = 2I_n - A$ and $\hat{D} = 2I_n + D$. Then, each element of $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is obtained by

$$(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})_{ij} = \begin{cases} 2/(D_{ii}+2) & i = j \\ -A_{ij}/\sqrt{(D_{ii}+2)(D_{jj}+2)} & i \neq j, \end{cases} \tag{3.14}$$

which has the same meaning with the original one given by

$$(2I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})_{ij} = \begin{cases} 2 & i = j \\ -A_{ij}/\sqrt{D_{ii}D_{jj}} & i \neq j. \end{cases} \tag{3.15}$$

From Eqs. (3.13), (3.14) and (3.15), the numerically stable decoder layer of GALA is given as

$$H^{(m+1)} = \xi(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(m)}\Theta^{(m)}), (m = \tfrac{M}{2}, ..., M-1), \tag{3.16}$$

where $\hat{A} = 2I_n - A$ and $\hat{D} = 2I_n + D$. The encoder part of GALA is constructed by using Eq. (3.7) as in GCN [3] as

$$H^{(m+1)} = \xi(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(m)}\Theta^{(m)}), (m = 0, ..., \tfrac{M}{2}-1), \tag{3.17}$$

where $H^{(0)} = X$ is the feature matrix of the input nodes, $\tilde{A} = I_n + A$ and $\tilde{D} = I_n + D$. The complexity of propagation functions, Eqs. (3.16) and (3.17), are both $\mathcal{O}(mpc)$,

Table 3.1: Effectiveness of various decoders

	Cora			Citeseer		
	ACC	NMI	ARI	ACC	NMI	ARI
Eq. (3.7)	0.5628	0.4074	0.3289	0.5296	0.2588	0.2437
Eq. (3.12)	0.5999	0.4274	0.3775	0.5915	0.3177	0.3126
Eq. (3.16)	**0.7459**	**0.5767**	**0.5315**	**0.6932**	**0.4411**	**0.4460**

where m is the cardinality of edges in the graph, p is the feature dimension of the previous layer, and c is the feature dimension of the current layer.

Since the complexity is linear in the number of edges in the graph, the proposed algorithm is computationally efficient (given the assumption that A is sparse). Also, from Eq. (3.17), since GALA decodes the latent representation using both the graph structure and node features, the enhanced decoder of GALA can help to find more distinct latent representation.

In Table 3.1, we show the reason why the Laplacian smoothing is not appropriate to the decoder and the necessity of numerically stable Laplacian sharpening by node clustering experiments (the higher values imply the more correct results). Laplacian smoothing decoder (Eq. 3.7) shows the lowest performances, since Laplacian smoothing which makes the representation of each node similar to those of its neighboring nodes conflicts with the purpose of reconstruction cost. A numerically instable form of Laplacian sharpening decoder (Eq. 3.12) shows higher performance compared to smoothing decoder because the role of Laplacian sharpening coincide with reconstructing the node feature. The performance of proposed numerically stable Laplacian sharpening decoder (Eq. 3.16) significantly higher than others, since it solves instability issue of neural network while maintaining the meaning of original Laplacian sharpening.

The basic cost function of GALA is given by

$$\min_{\bar{X}} \quad \frac{1}{2}\|X - \bar{X}\|_F^2, \tag{3.18}$$

where $\bar{X}$ is the reconstructed feature matrix of nodes, the column of $\bar{X}$ corresponds to the output of the decoder for an input feature of a node, and $\|\cdot\|_F$ denotes the Frobenius norm.

3.3.3 Subspace Clustering Cost for Image Clustering

It is a well-known assumption that image datasets are often drawn from multiple low-dimensional subspaces, although their data dimensions are high. Accordingly, subspace clustering, which has such an assumption about the input data in its own definition, has shown prominent clustering performance on various image datasets. Hence, we add an element of subspace clustering to the proposed method in the case of image clustering tasks. Among the various subspace clustering models, we add Least Squares Regression (LSR) [84] model for computational efficiency. Then the cost function for training of GALA becomes

$$\min_{\bar{X},H,A_H} \quad \frac{1}{2}\|X-\bar{X}\|_F^2 + \frac{\lambda}{2}\|H-HA_H\|_F^2 + \frac{\mu}{2}\|A_H\|_F^2, \tag{3.19}$$

where $H \in \mathbb{R}^{k\times n}$ denotes the latent representations (i.e., the output of the encoder), $A_H \in \mathbb{R}^{n\times n}$ denotes the affinity matrix which is a new latent variable for subspace clustering, and λ, μ are the regularization parameters. The second term of Eq. (3.19) aims at the self-expressive model of subspace clustering and the third term of Eq. (3.19) is for regularizing A_H. If we only consider minimizing A_H, the problem becomes:

$$\min_{A_H} \quad \frac{\lambda}{2}\|H-HA_H\|_F^2 + \frac{\mu}{2}\|A_H\|_F^2. \tag{3.20}$$

We can easily obtain the analytic solution $A_H^* = (H^TH + \frac{\mu}{\lambda}I_n)^{-1}H^TH$ by the fact that LSR model is quadratic on the variable A_H. By substituting A_H^* derived from Eq. (3.20) to Eq. (3.19), the cost function becomes

$$\min_{\bar{X},H} \quad \frac{1}{2}\|X-\bar{X}\|_F^2 + \frac{\lambda}{2}\left\|H-H(H^TH+\frac{\mu}{\lambda}I_n)^{-1}H^TH\right\|_F^2 + \frac{\mu}{2}\left\|(H^TH+\frac{\mu}{\lambda}I_n)^{-1}H^TH\right\|_F^2. \tag{3.21}$$

However, minimizing the above problem is computationally expensive since it requires the inverse of n by n matrix and lots of matrix multiplications, and we need a computationally-efficient form.

The singular value decomposition (SVD) of H can simplify the tangled cost function. Let $svd(H) = U\Sigma V^T$, where $U \in \mathbb{R}^{k\times k}$ $(UU^T = U^TU = I_k)$ is a unitary matrix composed of the left singular vectors of H, $V \in \mathbb{R}^{n\times n}$ $(VV^T = V^TV = I_n)$ is those of the right singular vectors of H, and $\Sigma \in \mathbb{R}^{k\times n}$ is a diagonal matrix whose diagonal elements are non-zero singular values $\{\sigma_i\}_{i=1}^k$ of H. Then the second term of Eq. (3.21), $\|H - H(H^TH + \frac{\mu}{\lambda}I_n)^{-1}H^TH\|_F^2$ can be simplified as $\|\mu(\mu I_k + \lambda HH^T)^{-1}H\|_F^2$ from that

$$\begin{aligned} H - H(H^TH + \frac{\mu}{\lambda}I_n)^{-1}H^TH &= U(\Sigma - \Sigma(\Sigma^T\Sigma + \frac{\mu}{\lambda}I_n)^{-1}\Sigma^T\Sigma)V^T \\ &= U\mu(\mu I_k + \lambda\Sigma\Sigma^T)^{-1}\Sigma V^T \\ &= \mu(\mu I_k + \lambda HH^T)^{-1}H, \end{aligned} \tag{3.22}$$

where the diagonal elements of the diagonal matrices at right hand sides of the first and the second equal signs are identical to $\{\mu\sigma_i/(\mu + \lambda\sigma_i^2)\}_{i=1}^k$.

Also, the third term of Eq. (3.21), $\|(H^TH + \frac{\mu}{\lambda}I_n)^{-1}H^TH\|_F^2$ can be simplified as $\|\lambda H^T(\mu I_k + \lambda HH^T)^{-1}H\|_F^2$ from that

$$\begin{aligned} (H^TH + \frac{\mu}{\lambda}I_n)^{-1}H^TH &= V(\Sigma^T\Sigma + \frac{\mu}{\lambda}I_n)^{-1}\Sigma^T\Sigma V^T \\ &= V\Sigma^T\lambda(\mu I_k + \lambda\Sigma\Sigma^T)^{-1}\Sigma V^T \\ &= \lambda H^T(\mu I_k + \lambda HH^T)^{-1}H, \end{aligned} \tag{3.23}$$

where the diagonal elements of the diagonal matrices at right hand side of the first and the second equal sign are identical to $\{\lambda\sigma_i^2/(\mu+\lambda\sigma_i^2)\}_{i=1}^k$ and 0 for from the $(k+1)$-th to n-th diagonal elements. From the above calculation, Eq. (3.21) is transformed into

$$\min_{\bar{X},H} \; \frac{1}{2}\|X - \bar{X}\|_F^2 + \frac{\lambda}{2}\|\mu(\mu I_k + \lambda HH^T)^{-1}H\|_F^2 + \frac{\mu}{2}\|\lambda H^T(\mu I_k + \lambda HH^T)^{-1}H\|_F^2. \tag{3.24}$$

We can attain a more simplified version of Eq. (3.24) by merging the second term and the third term by using SVD and the final computationally-efficient subspace clustering cost function is as follows:

$$\min_{\bar{X},H} \quad \frac{1}{2}\|X-\bar{X}\|_F^2 + \frac{\mu\lambda}{2} tr((\mu I_k + \lambda H H^T)^{-1} H H^T). \tag{3.25}$$

Furthermore, the affinity matrix is given by $A_H^* = (H^{*T}H^* + \frac{\mu}{\lambda}I_n)^{-1}H^{*T}H^*$ which requires an $n \times n$ matrix inversion. Its computationally-efficient version using a $k \times k$ matrix inversion is given by $A_H^* = \lambda H^{*T}(\mu I_k + \lambda H^* H^{*T})^{-1}H^*$.

$$\min_{\bar{X},H} \quad \frac{1}{2}\|X-\bar{X}\|_F^2 + \frac{\mu\lambda}{2} tr((\mu I_k + \lambda H H^T)^{-1} H H^T), \tag{3.26}$$

where $tr(\cdot)$ denotes the trace of the matrix. The above problem can be solved by $k \times k$ matrix inversion instead of $n \times n$ matrix inversion. Since the dimension of latent representation (k) is much smaller than the number of nodes (n), this simplification can reduce the computational burden significantly from $\mathcal{O}(n^3)$ to $\mathcal{O}(k^3)$.

3.3.4 Training

We train GALA to minimize Eq. (3.18) by using the ADAM algorithm [85]. We train GALA deterministically by using the full batch in each training epoch and stop when the cost is converged, so the number of epochs of each dataset varies. Note here that using the full batch during training is a common approach in neural networks based on spectral convolution on graph. Specifically, we set the learning rate to 1.0×10^{-4} for training and train GALA in an unsupervised way without any cluster labels. When the subspace clustering cost is added to reconstruction cost for image clustering tasks, we use pre-training and fine-tuning strategies similar to the ones in [78] to train GALA. First, in the pre-training stage, the training method is the same as that of minimizing Eq. (3.18). After pre-training, we fine-tune GALA to minimize Eq. (3.26) using ADAM. As in the pre-training, we train GALA deterministically by using full batch in each training epoch, and we set the number of epochs of the fine-tuning stage as 50 for all dataset. We set the learning rate to 1.0×10^{-6} for fine-tuning.

After the training process are over, we construct k-nearest neighborhood graph using attained latent representations H^*. Then we perform spectral clustering [26] and get the clustering performance. In the case of image clustering, after all training processes are over, we construct the optimal affinity matrix A_H^* noted in the previous subsection by using the attained latent representation matrix H^* from GALA. Then we perform spectral clustering [26] on the affinity matrix and get the optimal clustering with respect to our cost function.

3.4 Experiments

3.4.1 Datasets

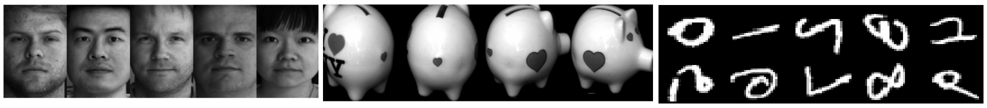
Table 3.2: Summary of datasets

	# Nodes	Dimension	Classes	# Edges
Cora [8]	2708	1433	7	5429
Citeseer [8]	3312	3703	6	4732
Wiki [86]	2405	4973	17	17981
Pubmed [8]	19717	500	3	44338
COIL20 [87]	1440	1024	20	–
YALE [88]	5850	1200	10	–
MNIST [89]	10000	784	10	–

We have used four network datasets (Cora, Citeseer, Wiki, and Pubmed) and three image datasets (COIL20, YALE, and MNIST) for clustering tasks. Every network dataset has a feature matrix X and an affinity matrix A, and every image dataset has a feature matrix X only. The details of each dataset are as follows:

Cora [8] is a citation network between scientific publications which consists of 2,708 nodes where their feature dimension is 1,433 and there are 5,429 edges between nodes. The number of the clusters is 7.

Citeseer [8] is also a citation network between scientific publications, which has 3,312 nodes with 3,703 feature dimensions, and 4,732 edges between nodes. The number of the clusters of Citeseer is 6.



(a) YALE (b) COIL20 (c) MNIST

Figure 3.2: Sample images of three image datasets

Wiki [86] is a network dataset whose number of nodes is 2,405, the dimension of the features is 4,973, and there are 17,981 edges between nodes. The number of the clusters of Wiki is 17. We observed that several papers [50], [86] have written the number of the clusters of Wiki as 19. Although nodes are labeled up to 19, the actual cardinality of the labels is 17.

Pubmed [8] is a citation network which consists of 19,717 nodes where their feature dimension is 500 and there are 44,338 edges between nodes. The number of the clusters is 3.

COIL20 [87] is the Columbia Object Image Library dataset. There are 1,440 images for 20 objects and each object contains 72 images since the pose changes every 5 degrees $(360° = 72 \times 5°)$. The size of each image is 32 $\times$ 32, so the feature dimension is 1024. The number of the clusters of COIL20 is 20.

YALE [88] is a face dataset which contains 5,850 face images with 9 poses and 65 illumination variations. Original images were cropped to 30 $\times$ 40 pixels, so the feature dimension of each face image is 1200. The number of the clusters is 10.

MNIST [89] is a handwritten digit dataset. There are 60,000 training samples and 10,000 test samples and among them, we use 10,000 test samples for image clustering. The size of each digit image is 28 $\times$ 28, so the feature dimension is 784. The number of the clusters of MNIST is 10.

The sample images of each image dataset are described in Figure 3.2.

3.4.2 Experimental Settings

To measure the performance of node clustering task, we use three metrics: accuracy (ACC), normalized mutual information (NMI), and adjusted rand index (ARI) as in [50]. We report the mean values of the three metrics for each algorithm after executing 50 times, and the higher values imply the more correct results. For link prediction task, we partitioned the dataset following the work of GAE [51], and reported mean scores and standard errors of Area Under Curve (AUC) and Average Precision (AP) with 10 random initializations.

We conduct our algorithm on a GPU environment (a Nvidia GTX 1080Ti GPU) in TensorFlow [90]. For the Cora dataset, we set the encoder's number of layers to two (1600, 400 neurons). For the Citeseer dataset, we set the number of layers of the encoder to two (2000, 500 neurons). For the Wiki dataset, we set the encoder's number of layers to one (500 neurons). For the Pubmed dataset, we set the encoder's number of layers to two (600, 100 neurons). For the COIL20 dataset, we set $\lambda = 9.0$, $\mu = 1.0$, and the number of layers of the encoder to three (1100, 800, 500 neurons). For the YALE dataset, we set $\lambda = 3.0 \times 10$, $\mu = 1.0$, and the encoder's number of layers to three (1300, 800, 500 neurons). For the MNIST dataset, we set $\lambda = 5.0 \times 10$, $\mu = 1.0$, and the number of layers of the encoder to three (800, 700, 500 neurons). The encoder and decoder have symmetrical structures for all datasets. We set the parameters of the compared methods following the instructions of their papers. Among the parameter sets noted in each paper, we reported the best results [50, 52]. For the image datasets, we construct k-nearest neighborhood graphs using each dataset's feature matrix. We set k as 9, 9, and 20 for COIL20, YALE, and MNIST respectively. Also, we normalize the feature of each image on unit interval for all image datasets.

3.4.3 Comparing Methods

We compare the performance of 15 algorithms. Compared algorithms can be categorized into four groups as described below:

- **i) Using features only.** ‘Kmeans’ [91] is the K-means clustering based on only the features of the data, which is the baseline clustering algorithm in our experiment.

- **ii) Using network structures only.** ‘Spectral’ [26] is a spectral clustering algorithm using eigendecomposition on graph Laplacian. ‘Big-Clam’ [92] is a large-scale community detection algorithm utilizing a variant of nonnegative matrix factorization. ‘DeepWalk’ [31] learns the latent social representation of nodes using local information through a neural network. ‘GraEnc’ [93] is a graph-encoding neural network derived from the relation between autoencoder and spectral clustering. ‘DNGR’ [94] generates a low-dimensional representation of each node by using a graph structure and a stacked denoised autoencoder.

- **iii) Using both.** ‘Circles’ [95] is an algorithm which discovers social circles through a node clustering algorithm. ‘RTM’ [96] presents a relational topic model of documents and links between the documents. ‘RMSC’ [97] is a robust multi-view spectral clustering algorithm which can handle noises in the data and recover transition matrix through low-rank and sparse decomposition. ‘TADW’ [86] interprets DeepWalk from the view of matrix factorization and incorporates text features of nodes.

- **iv) Using both with spectral convolution on graphs.** ‘GAE’ [51] is the first attempt to graft the spectral convolution on graphs onto autoencoder framework. ‘VGAE’ [51] is the variational variant of GAE. ‘MGAE’ [50] is an autoencoder which combines the marginalization process with spectral convolution on graphs. ‘ARGA’ [52] learns the latent representation by adding an adversarial model to a non-probabilistic variant of VGAE. ‘ARVGA’ [52] is an algorithm which adds an adversarial model to VGAE.

3.4.4 Node Clustering

The experimental results of node clustering are presented in Table 3.3. It can be observed that for every dataset, the methods which use features and network structures

Table 3.3: Experimental results of node clustering

	Cora			Citeseer			Wiki		
	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
Kmeans [91]	0.4922	0.3210	0.2296	0.5401	0.3054	0.2786	0.4172	0.4402	0.1507
Spectral [26]	0.3672	0.1267	0.0311	0.2389	0.0557	0.0100	0.2204	0.1817	0.0146
Big-Clam [92]	0.2718	0.0073	0.0011	0.2500	0.0357	0.0071	0.1563	0.0900	0.0070
DeepWalk [31]	0.4840	0.3270	0.2427	0.3365	0.0878	0.0922	0.3846	0.3238	0.1703
GraEnc [93]	0.3249	0.1093	0.0055	0.2252	0.0330	0.0100	0.2067	0.1207	0.0049
DNGR [94]	0.4191	0.3184	0.1422	0.3259	0.1802	0.0429	0.3758	0.3585	0.1797
Circles [95]	0.6067	0.4042	0.3620	0.5716	0.3007	0.2930	0.4241	0.4180	0.2420
RTM [96]	0.4396	0.2301	0.1691	0.4509	0.2393	0.2026	0.4364	0.4495	0.1384
RMSC [97]	0.4066	0.2551	0.0895	0.2950	0.1387	0.0488	0.3976	0.4150	0.1116
TADW [86]	0.5603	0.4411	0.3320	0.4548	0.2914	0.2281	0.3096	0.2713	0.0454
VGAE [51]	0.5020	0.3292	0.2547	0.4670	0.2605	0.2056	0.4509	0.4676	0.2634
MGAE [50]	0.6844	0.5111	0.4447	0.6607	0.4122	0.4137	0.5146	0.4852	0.3490
ARGA [52]	0.6400	0.4490	0.3520	0.5730	0.3500	0.3410	0.3805	0.3445	0.1122
ARVGA [52]	0.6380	0.4500	0.3740	0.5440	0.2610	0.2450	0.3867	0.3388	0.1069
GALA	**0.7459**	**0.5767**	**0.5315**	**0.6932**	**0.4411**	**0.4460**	**0.5447**	**0.5036**	**0.3888**

Table 3.4: Experiment results on Pubmed dataset

	ACC	NMI	ARI
Kmeans [91]	0.5952	0.3152	0.2817
Spectral [26]	0.5282	0.0971	0.0620
GAE [51]	0.6861	0.2957	0.3046
VGAE [51]	0.6887	0.3108	0.3018
MGAE [50]	0.5932	0.2822	0.2483
ARGA [52]	0.6807	0.2757	0.2910
ARVGA [52]	0.5130	0.1169	0.0777
GALA	**0.6939**	**0.3273**	**0.3214**

simultaneously show better performance than the methods which use only one of them. Furthermore, among the methods which use both features and network structures, algorithms with neural network models which exploit spectral convolution on graphs present outstanding performance since they can learn deeper relationships between nodes than the methods which do not use spectral convolution on graphs. In every experiments, GALA shows superior performance to other methods. Especially, for the Cora dataset, GALA outperforms VGAE, which is the first graph convolution autoencoder framework, by about 24.39%, 24.75% and 27.68%, and MGAE, which is the state-of-the-art graph convolutional autoencoder algorithm, by about 6.15%, 6.56% and 8.68% on ACC, NMI and ARI, respectively. The better performance of GALA comes from the better decoder design based on the numerically stable form of Laplacian sharpening both and full utilizing of graph structure and node attributes in the whole autoencoder framework.

Furthermore, we conduct another node clustering experiment on a large network dataset (Pubmed), and the results are reported in Table 3.4. We can observe that GALA outperforms every baselines and state-of-the-art graph convolution algorithms. Although Kmeans clustering, a baseline algorithm, shows higher performance over several graph convolution algorithms on NMI and ARI, the proposed method presents better performances.

3.4.5 Image Clustering

The experimental results of image clustering are presented in Table 3.5. We report both GALA's performance of reconstruction cost only case and the subspace clustering cost added case (GALA+SCC). It can be seen that GALA outperforms several baselines and the state-of-the-art graph convolution algorithms for most of the cases. Also, for every case, the proposed subspace clustering cost term contributes to improve the performance of the image clustering. On the YALE dataset, notably, we can observe that the proposed subspace clustering cost term significantly enhances the image clustering performance

Table 3.5: Experimental results of image clustering

	COIL20			YALE			MNIST		
	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
Kmeans [91]	0.6118	0.7541	0.5545	0.7450	0.8715	0.7394	0.5628	0.5450	0.4213
Spectral [26]	0.6806	0.8324	0.6190	0.5793	0.7202	0.4600	0.6496	0.7204	0.5836
GAE [51]	0.6632	0.7420	0.5514	0.8520	0.8851	0.8122	0.7043	0.6535	0.5534
VGAE [51]	0.6847	0.7465	0.5627	0.9157	0.9358	0.8873	0.7163	0.7149	0.6154
MGAE [50]	0.6507	0.7889	0.6004	0.8203	0.8550	0.7636	0.5807	0.5820	0.4362
ARGA [52]	0.7271	0.7895	0.6183	0.9309	0.9394	0.8961	0.6672	0.6759	0.5552
ARVGA [52]	0.7222	0.7917	0.6240	0.8727	0.8803	0.7944	0.6328	0.6123	0.4909
GALA	0.8000	0.8771	0.7550	0.8530	0.9486	0.8647	0.7384	0.7506	0.6469
GALA+SCC	**0.8229**	**0.8851**	**0.7579**	**0.9933**	**0.9860**	**0.9854**	**0.7426**	**0.7565**	**0.6675**

and achieves nearly perfect accuracy.

3.4.6 Ablation Studies

We validate the effectiveness of the proposed stable decoder and the subspace clustering cost by image clustering experiments on the three image datasets (COIL20, YALE and MNIST). There are four configurations as shown in Table 3.6. We would like to note that the reconstruction cost only (Eq. 3.18) is a subset of subspace clustering cost (Eq. 3.26), thus the last configuration is the full proposed method. Reported numbers are mean values after executing 50 times. It can be clearly noticed that the numerically stable form of Laplacian sharpening and subspace clustering cost are helpful to find the latent representations which reflect the graph structures certainly and using both components can boost the performance of clustering. In addition, it can be seen that the stable decoder with the reconstruction cost only outperforms the state-of-the-art algorithms in most cases because GALA can utilize the graph structure in the whole processes of the autoencoder architecture.

Table 3.6: Effects of stable decoder and subspace clustering cost

	COIL20			YALE			MNIST		
	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
Unstable decoder and reconstruction cost (Eq. 3.12, Eq. 3.18)	0.5961	0.7986	0.5492	0.7205	0.9028	0.7530	0.6589	0.7397	0.5983
Unstable decoder and sub. clustering cost (Eq. 3.12, Eq. 3.26)	0.7104	0.8074	0.6429	0.7810	0.8710	0.7130	0.6734	0.7211	0.6028
Stable decoder and reconstruction cost (Eq. 3.16, Eq. 3.18)	0.8000	0.8771	0.7550	0.8530	0.9486	0.8646	0.7384	0.7506	0.6469
Stable decoder and sub. clustering cost (Eq. 3.16, Eq. 3.26)	**0.8229**	**0.8851**	**0.7579**	**0.9933**	**0.9860**	**0.9854**	**0.7426**	**0.7565**	**0.6675**

3.4.7 Link Prediction

We provide some results on link prediction task on Citeseer dataset. For link prediction task, we minimized the below cost function that added link prediction cost of GAE [51] to the reconstruction cost, where H is the latent representation, $\hat{A} = \text{sigmoid}(HH^T)$ is the reconstructed affinity matrix and γ is the regularization parameter.

$$\min_{\bar{X},H} \quad \frac{1}{2}\|X - \bar{X}\|_F^2 + \gamma \mathbb{E}_H[\log p(\hat{A}|H)]. \tag{3.27}$$

The results are shown in Table 3.7, and our model outperforms the compared methods in terms of the link prediction task as well as the node clustering task.

3.4.8 Visualization

One of the key ideas of the proposed autoencoder is that the encoder makes the feature of each node becomes similar with its neighbors, and the decoder makes the features of each node distinguishable with its neighbors using the geometrical structure of the graphs. To validate the proposed model, we visualize the distribution of learned latent

Table 3.7: Experimental results of link prediction on Citeseer

	AUC	AP
GAE [51]	89.5 $\pm$ 0.04	89.9 $\pm$ 0.05
VGAE [51]	90.8 $\pm$ 0.02	92.0 $\pm$ 0.02
ARGA [52]	91.9 $\pm$ 0.003	93.0 $\pm$ 0.003
ARVGA [52]	92.4 $\pm$ 0.003	93.0 $\pm$ 0.003
GALA	**94.4** $\pm$ **0.009**	**94.8** $\pm$ **0.010**

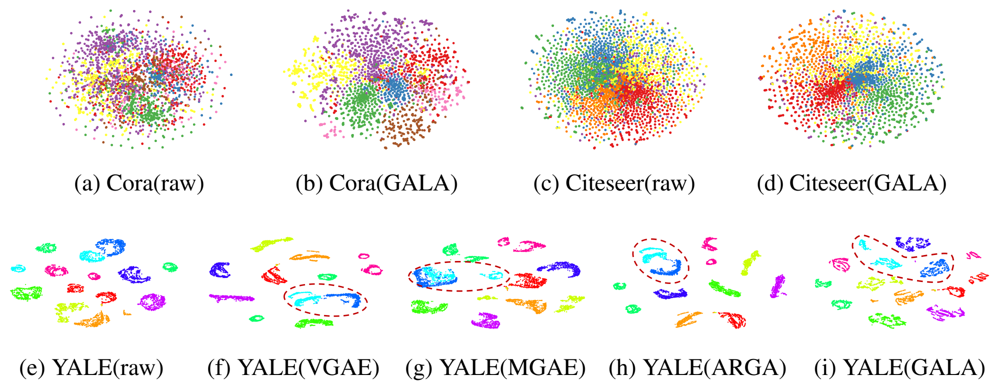


Figure 3.3: The two-dimensional visualizations of raw features of each node and the latent representations of compared methods and GALA for Cora, Citeseer and YALE are presented. The same color indicates the same cluster.

representations and the input features of each node in two-dimensional space using *t-SNE* [98] as shown in Figure 3.3. From the visualization, we can see that GALA is well-clustering the data according to their corresponding labels even though GALA performs in an unsupervised manner. Also, we can see through the red dotted line in embedding results of the latent representation on YALE that GALA embeds the representation of nodes better than the compared methods by minimizing inter-cluster affinity and maximizing intra-cluster affinity.

3.5 Summary

In this chapter, we proposed a novel autoencoder framework which can extract low-dimensional latent representations from a homogeneous graph. We designed a symmetric graph convolutional autoencoder architecture where the encoder performs Laplacian smoothing while the decoder performs Laplacian sharpening. Also, to prevent numerical instabilities, we designed a new representation of Laplacian sharpening with spectral radius one by incorporating the concept of the signed graph. To enhance the performance of image clustering tasks, we added a subspace clustering cost term to the reconstruction cost of the autoencoder. Experimental results on the network and image datasets demonstrated the validity of the proposed framework and had shown superior performance over various graph-based clustering algorithms.

Chapter 4

Autoencoder Architecture for Tree-like Graphs

4.1 Overview

Recently, many works [99–105] utilize hyperbolic geometry [106] to learn representations by understanding the underlying nature of the data domains. It is well known that complex networks contain latent hierarchies between large groups and the divided subgroups of nodes and can be approximated as trees that grow exponentially with their depth [106]. Based on this fact, previous works which involve graphs [99–101,107–109] showed the effectiveness of learning representation using hyperbolic spaces (a continuous version of trees) where distances increase exponentially when moving away from the origin. More recently, few works [103, 104, 110] have been conducted which learn more powerful representations via conducting message passing (graph convolution) [3, 9, 61] in hyperbolic spaces.

In addition, it has been successfully shown that grafting hyperbolic geometry onto computer vision tasks is promising [105]. They observed a high degree of hyperbolicity [111] in the activations of image datasets obtained from pre-trained convolutional networks. Also, it has been shown that the hyperbolic distance between learned embeddings and the origin of the Poincaré ball could be considered as a measurement of the model's confidence. Using these analyses, [105] added a single layer of hyperbolic neural networks [101] to deep convolutional networks and showed the benefits of hyperbolic embeddings on few-shot learning and person re-identification. Another work [112] also demonstrated the suitability of hyperbolic embeddings on zero-shot learning. However, most of the existing hyperbolic representation learning works [103–105, 110, 112] mainly focus on a supervised setting, and the effect of hyperbolic geometry on unsupervised representation learning has not been explored deeply so far [109, 113, 114].

In this chapter, we explore the benefits of hyperbolic geometry to carry out unsupervised representation learning upon various data domains. Our motivation is to learn high-quality node embeddings of the graphs that are hierarchical and tree-like without supervision via considering the geometry of the embedding space. To do so, we present a novel hyperbolic graph convolutional autoencoder (HGCAE) by combining hyperbolic geometry and message passing [9]. Every layer of HGCAE performs message passing in the hyperbolic space and its corresponding tangent space where curvature values can be trained. This is primarily in contrast to the Poincaré variational autoencoder (P-VAE) [113] whose latent space is the Poincaré ball and conducts message passing in Euclidean space. The HGCAE conducts auto-encoding the graphs from diverse data domains, such as images or social networks, in the hyperbolic space such as the Poincaré ball and hyperboloid. To fully utilize hyperbolic geometry for representation learning, we adopt a geometry-aware attention mechanism [102] when conducting message passing. Through extensive experiments and analyses using the learned representation in the hyperbolic latent spaces, we present the following observations on hierarchically structured data:

- The proposed autoencoder, which combines message passing based on geometry-aware attention and hyperbolic spaces, can learn useful representations for downstream tasks. On various networks, the proposed method achieves state-of-the-art results on node clustering and link prediction tasks.

- Image clustering tasks can benefit from embeddings in hyperbolic latent spaces. We achieve comparable results to state-of-the-art image clustering results by learning representations from the activations of neural networks.

- Hyperbolic embeddings of images, the results of unsupervised learning, can recognize the underlying data structures such as a class hierarchy without any supervision of ground-truth class hierarchy.
- We show that the sample's hyperbolic distance from the origin in hyperbolic space can be utilized as a criterion to choose samples, therefore improving the generalization ability of a model for a given dataset.

4.2 Preliminaries

4.2.1 Hyperbolic Embeddings

Hyperbolic embedding of images. Khrulkov et al. [105] validated hyperbolic embeddings of images via measuring the degree of hyperbolicity of image datasets. Many datasets such as CIFAR10/100 [6], CUB [7] and MiniImageNet [115] showed high degrees of hyperbolicity. In particular, the ImageNet dataset [116] is organized by following the hierarchical structure of WordNet [117]. These observations suggest that hyperbolic geometry can be beneficial in analyzing image manifolds by capturing not only semantic similarities but also hierarchical relationships between images. Furthermore, Khrulkov et al. [105] empirically showed that the distance between the origin and the image embeddings in the Poincaré ball can be regarded as the measure of the model's confidence. They observed that the samples which are easily classified are located near the boundary, while those more ambiguous samples lie near the origin of the hyperbolic space. Recent works of hyperbolic image embeddings [105, 112] add one or two layers of hyperbolic layers [101] after Euclidean convolutional networks.
Graph auto-encoding via hyperbolic geometry. Some recent works [113, 114, 118] attempted to auto-encode graphs in hyperbolic space. Their models attempted to learn latent representations in the hyperbolic space via grafting hyperbolic geometry onto

a variational autoencoder model [119]. [113, 114] encoded the node representation via message passing [3] in Euclidean space, then the encoded representation was projected onto the hyperbolic space. Similar to these concurrent models, our autoencoder framework learns latent node representations of the graph in hyperbolic latent spaces. Differing from these models, our work considers hyperbolic geometry throughout the auto-encoding process. Each encoder and decoder layer of the proposed model conducts message passing by utilizing geometry-aware attention in the hyperbolic space and its tangent space.

4.2.2 Hyperbolic Geometry

A real, smooth manifold $\mathcal{M}$ is a set of points x, that is locally similar to linear space. At each point $x \in \mathcal{M}$, the tangent space at x, $\mathcal{T}_x\mathcal{M}$, is a real vector space whose dimensionality is same as $\mathcal{M}$. A Riemannian manifold is defined as a tuple $(\mathcal{M}, g)$ that is possessing metric tensor $g_x : \mathcal{T}_x\mathcal{M} \times \mathcal{T}_x\mathcal{M} \to \mathbb{R}$ on the tangent space $\mathcal{T}_x\mathcal{M}$ at each point $x \in \mathcal{M}$ [120]. The metric tensor provides geometric notions such as geodesic, angle and volume. There exist mapping between the manifold and the tangent space: exponential map and logarithmic map. The exponential map $\exp_x : \mathcal{T}_x\mathcal{M} \to \mathcal{M}$ projects the vector on the tangent space $\mathcal{T}_x\mathcal{M}$ back to the manifold $\mathcal{M}$, while the logarithmic map $\log_x : \mathcal{M} \to \mathcal{T}_x\mathcal{M}$ is the inverse mapping of the exponential map as $\log_x(\exp_x(v)) = v$.

The hyperbolic space is a Riemannian manifold with constant negative sectional curvature equipped with hyperbolic geometry. This paper deals with two hyperbolic spaces; *'Poincaré ball'* and *'hyperboloid'*. The *Poincaré ball* $\mathbb{P}$ is highly effective for visualizing and analyzing the hyperbolic latent space. Meanwhile, the *hyperboloid* $\mathbb{H}$ can provide stable optimization since, unlike distance function of *Poincaré ball*, there is no division in the distance function [100].

Poincaré ball. The n-dimensional Poincaré ball with constant negative curvature

$K(K<0)$ $(\mathbb{P}^n_K, g^{\mathbb{P}_K}_x)$ is defined:

$$\mathbb{P}^n_K = \{x \in \mathbb{R}^n : \|x\|^2 < -1/K\}, \tag{4.1}$$

where $\|\cdot\|$ denotes Euclidean norm. The metric tensor is $g^{\mathbb{P}_K}_x = (\lambda^K_x)^2 g^{\mathbb{E}}_x$, where $\lambda^K_x = \frac{2}{1+K\|x\|^2}$ is the conformal factor and $g^{\mathbb{E}}_x = \mathrm{diag}([1,1,\dots 1])$ denotes Euclidean metric tensor. The origin of $\mathbb{P}^n_K$ is $\mathbf{o} = (0,\dots,0) \in \mathbb{R}^n$. The distance between two points $x, y \in \mathbb{P}^n_K$ is defined as

$$d_{\mathbb{P}^n_K}(x,y) = \frac{1}{\sqrt{-K}} \operatorname{arcosh}\left(1 - \frac{2K\|x-y\|^2}{(1+K\|x\|^2)(1+K\|y\|^2)}\right). \tag{4.2}$$

For points $x \in \mathbb{P}^n_K$, tangent vector $v \in \mathcal{T}_x\mathbb{P}^n_K$, and $y \neq \mathbf{0}$, the exponential map $\exp_x : \mathcal{T}_x\mathbb{P}^n_K \to \mathbb{P}^n_K$ and the logarithmic map $\log_x : \mathbb{P}^n_K \to \mathcal{T}_x\mathbb{P}^n_K$ are defined as:

$$\exp^K_x(v) = x \oplus_K \left(\tanh(\frac{\sqrt{-K}\lambda^K_x\|v\|}{2})\frac{v}{\sqrt{-K}\|v\|}\right), \tag{4.3}$$

$$\log^K_x(y) = \frac{2}{\sqrt{-K}\lambda^K_x} \operatorname{arctanh}\left(\sqrt{-K}\|u\|\right)\frac{u}{\|u\|}, \tag{4.4}$$

where $u = -x \oplus_K y$ and $\oplus_K$ denotes Möbius addition [121] for $x, y \in \mathbb{P}^n_K$ as

$$x \oplus_K y = \frac{(1 - 2K\langle x,y\rangle - K\|y\|^2)x + (1+K\|x\|^2)y}{1 - 2K\langle x,y\rangle + K^2\|x\|^2\|y\|^2}. \tag{4.5}$$

Hyperboloid. The hyperbolic space is a Riemannian manifold with constant negative sectional curvature equipped with hyperbolic geometry, and the hyperboloid model is one of the multiple equivalent hyperbolic models. For $x, y \in \mathbb{R}^{n+1}$, the Lorentz inner product $\langle\cdot,\cdot\rangle_{\mathcal{L}}$ is defined as $\langle x,y\rangle_{\mathcal{L}} = -x_0y_0 + \sum_{i=1}^n x_iy_i$. The n-dimensional hyperboloid with constant negative curvature $K(K<0)$ is defined as $(\mathbb{H}^n_K, g^{\mathbb{H}_K}_x)$:

$$\mathbb{H}^n_K = \{x \in \mathbb{R}^{n+1} : \langle x,x\rangle_{\mathcal{L}} = 1/K, x_0 > 0\}. \tag{4.6}$$

The metric tensor is $g^{\mathbb{H}_K}_x = \mathrm{diag}([-1,1,\dots 1])$, and the origin of the hyperboloid model is $\mathbf{o} = (1/\sqrt{|K|}, 0, \dots, 0) \in \mathbb{R}^{n+1}$. The distance between two points $x, y \in \mathbb{H}^n_K$ is defined as

$$d_{\mathbb{H}^n_K}(x,y) = \frac{1}{\sqrt{-K}} \operatorname{arcosh}(K\langle x,y\rangle_{\mathcal{L}}). \tag{4.7}$$

For points $x \in \mathbb{H}^n_K$, tangent vector $v \in \mathcal{T}_x\mathbb{H}^n_K$, and $y \neq \mathbf{0}$, $\exp_x : \mathcal{T}_x\mathbb{H}^n_K \to \mathbb{H}^n_K$ and $\log_x : \mathbb{H}^n_K \to \mathcal{T}_x\mathbb{H}^n_K$ are defined as

$$\exp^K_x(v) = \cosh(s)x + \sinh(s)\frac{v}{s}, \tag{4.8}$$

$$\log^K_x(y) = \frac{\operatorname{arcosh}(K\langle x, y\rangle_{\mathcal{L}})}{\sqrt{K^2\langle x, y\rangle^2_{\mathcal{L}} - 1}}(y - K\langle x, y\rangle_{\mathcal{L}}x), \tag{4.9}$$

where $s = \sqrt{-K}\|v\|_{\mathcal{L}}$ and $\|x\|_{\mathcal{L}} = \sqrt{\langle x, x\rangle_{\mathcal{L}}}$.

Mapping between two models. Two hyperbolic models, Poincaré ball and hyperboloid, are equivalent and transformations between two models retain many geometric properties including isometry. There exist diffeomorphisms $p_{\mathbb{H}\to\mathbb{P}}$ and $p_{\mathbb{P}\to\mathbb{H}}$ between the two models, Poincaré ball $\mathbb{P}^n_K$ and hyperboloid $\mathbb{H}^n_K$ [103, 110], as follows:

$$p_{\mathbb{H}\to\mathbb{P}}(x_0, x_1, \ldots, x_n) = \frac{(x_1, \ldots, x_n)}{\sqrt{|K|}x_0 + 1}, \tag{4.10}$$

$$p_{\mathbb{P}\to\mathbb{H}}(x_1, \ldots, x_n) = \frac{(\frac{1}{\sqrt{|K|}}(1 - K\|x\|^2), 2x_1, \ldots, 2x_n)}{1 + K\|x\|^2}. \tag{4.11}$$

4.3 Methodology

HGCAE is designed to fully utilize hyperbolic geometry in the auto-encoding process along with leveraging the power of graph convolutions via a geometry-aware attention mechanism. Each layer conducts message passing in hyperbolic space whose curvature value is trainable. Before conducting message passing, we need to map the given input data points, x^{Euc}, defined in Euclidean space to the hyperbolic manifold. We map the Euclidean feature into hyperbolic manifold via $h^1_i = \exp^{K_1}_{\mathbf{o}}(x^{Euc}_i)$, where K_1 and h^1_i denote a trainable curvature value and the i-th node's representation of the first layer respectively. When the hyperbolic space is hyperboloid model, we use $(0, x^{Euc}) \in \mathbb{R}^{n+1}$ as an input of an exponential map as [103] did. The overall architecture of HGCAE is presented in Fig. 4.1.

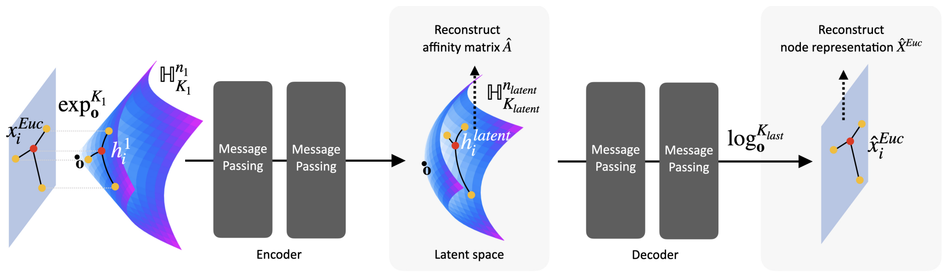

x_i^{Euc}
$\exp_{\mathbf{o}}^{K_1}$
$\mathbb{H}^{n_1}_{K_1}$
h_i^1
$\mathbf{o}$
Message Passing
Message Passing
Encoder
Reconstruct affinity matrix $\hat{A}$
$\mathbb{H}^{n_{latent}}_{K_{latent}}$
h_i^{latent}
Latent space
Message Passing
Message Passing
Decoder
$\log_{\mathbf{o}}^{K_{last}}$
Reconstruct node representation $\hat{X}^{Euc}$
$\hat{x}_i^{Euc}$

Figure 4.1: The overall architecture of HGCAE in a two-layer autoencoder (*i.e.* the encoder and decoder have two layers each) whose hyperbolic space is hyperboloid. This figure describes three things: 1) how the node of the graph (red dot) conducts message passing (Eq. (4.12) and (4.15)) with its neighbors (yellow dot), 2) the process of embedding the output of encoder in hyperboloid latent space (blue-purple space), and 3) reconstruction of Euclidean node attributes at the end of the decoder.

4.3.1 Geometry-Aware Message Passing

Linear transformation. Message passing in the HGCAE consists of two steps: the linear transformation of a message and aggregating messages from neighbors. The i-th node's message passing result at the l-th layer z_i^l is as follows:

$$z_i^l = \exp_{\mathbf{o}}^{K_l}\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^l \left(W^l \log_{\mathbf{o}}^{K_l}(h_j^l) + b^l\right)\right), \tag{4.12}$$

where W^l, b^l, $\mathcal{N}(i)$, and α_{ij}^l denote a weight matrix, a bias term, the set of direct neighbors of node i including itself, and the relative importance (attention score) of the neighbor node j to the node i at the l-th layer respectively. Based on [101], we map the points in the hyperbolic manifold to the tangent space via the logarithmic map since the linear transformation cannot be performed directly in hyperbolic spaces. Then, the messages are linearly transformed on the tangent space of the origin in which inherits many properties of the ambient Euclidean space.

Aggregation. After performing linear transformation, we aggregate messages from neighbors via an attention mechanism. The majority of message passing algorithms that use attention mechanisms learn the relative importance of each node's neighbors based on node feature not only in Euclidean space [61] but also in hyperbolic space [103]. However, only considering node features for learning their relative importance does not take into account the geometry of the space, and this might result in an imprecise attention score. To make full use of the Riemannian metric of the hyperbolic manifolds, we adopt a geometry-aware attention mechanism [102] by utilizing the distance between linearly transformed node features on the hyperbolic space. Let $y_i^l = W^l \log_{\mathbf{o}}^{K_l}(h_i^l) + b^l$, then the attention score at the l-th layer in Eq. (4.12) is:

$$\alpha_{ij}^l = \frac{\exp(-\beta^l d^2_{\mathcal{M}_{K_l}}(y_i^l, y_j^l) - \gamma^l)}{\sum_{p \in \mathcal{N}(i)} \exp(-\beta^l d^2_{\mathcal{M}_{K_l}}(y_i^l, y_p^l) - \gamma^l)}, \tag{4.13}$$

where $d_{\mathcal{M}_{K_l}}(\cdot,\cdot)$, β^l, and γ^l denote the distance on the hyperbolic space with curvature value K_l, and trainable parameters of the l-th layer respectively. After every step of message passing, we map the representation on the tangent space to the hyperbolic manifold via the exponential map.

4.3.2 Nonlinear Activation

The nonlinear activations, σ, such as ReLU can be directly applied to the points in the Poincaré ball, in contrast to the points on the hyperboloid [110]. Thus, when the hyperboloid model is used, we map the points to the Poincaré ball using Eq. (4.10) first. Next, we apply the nonlinear activation in the Poincaré ball and then return the result to the hyperboloid using the Eq. (4.11).

Since the curvature value of each layer in HGCAE is trainable, each layer can have different curvature values from other layers. Thus, a step for locating the result of the nonlinear activation in the hyperbolic space having a curvature value of the next layer is required. First, we map the results of the nonlinear activation to the tangent space of the current layer, $\mathcal{T}_{\mathbf{o}}\mathcal{M}_{K_l}$, using logarithmic map, $\log_{\mathbf{o}}^{K_l}$. Next, the points in the tangent space are mapped to the next layer's hyperbolic space via an exponential map of the next layer $\exp_{\mathbf{o}}^{K_{l+1}}$. The equations for performing such nonlinear activation and mapping to the hyperbolic space of the next layer in the cases of Poincaré ball and hyperboloid are as follows respectively:

$$h_i^{l+1} = \exp_{\mathbf{o}}^{K_{l+1}}\Big(\log_{\mathbf{o}}^{K_l}\left(\sigma(z_i^l)\right)\Big), \tag{4.14}$$

$$h_i^{l+1} = \exp_{\mathbf{o}}^{K_{l+1}}\Big(\log_{\mathbf{o}}^{K_l}\left(p_{\mathbb{P}\to\mathbb{H}}(\sigma(p_{\mathbb{H}\to\mathbb{P}}(z_i^l)))\right)\Big). \tag{4.15}$$

4.3.3 Loss Function

Our HGCAE reconstructs both the affinity matrix (graph structure) A and the Euclidean node attributes X^{Euc} at the end of the encoder and the decoder, respectively. To reconstruct the Euclidean node attributes $\hat{X}^{Euc}$, the aggregated representations in the hyperbolic space of the decoder's last layer are mapped to the tangent space of the origin $\mathcal{T}_{\mathbf{o}}\mathcal{M}$. Then, the loss of representations $\mathcal{L}_{REC-X}$ is defined as the mean square error between X^{Euc} and $\hat{X}^{Euc}$: $\frac{1}{N}\|X^{Euc} - \hat{X}^{Euc}\|^2$. For reconstructing the structure of the graph, the hyperbolic distance between the latent representations (the output of the encoder) of two nodes is utilized. To calculate the probability score of an edge which links between two nodes, we adopt the Fermi-Dirac distribution [99, 106], $\hat{A}_{ij} = [e^{(d^2_{\mathcal{M}^K}(h_i,h_j)-r)/t} + 1]^{-1}$, where h_i, $\hat{A}$, r, and t denote the latent representation of node i, the reconstructed affinity matrix, and hyperparameters respectively. The loss function for the affinity matrix is defined by the cross entropy loss with negative sampling: $\mathcal{L}_{REC-A} = \mathbb{E}_{q(H|X,A)}[\log p(\hat{A}|H)]$, where $q(H|X,A) = \prod_{i=1}^{N} q(h_i|X,A)$. The overall loss function of HGCAE is

$$\mathcal{L} = \mathcal{L}_{REC-A} + \lambda\mathcal{L}_{REC-X}, \tag{4.16}$$

where λ is a regularization parameter. λ serves to control the relative importance between the attributes and structure.

4.4 Experiments

This section explores the effectiveness of unsupervised hyperbolic embeddings on various data domains via quantitative and qualitative analyses. We use 9 real-world

Table 4.1: Dataset statistics.

Dataset	Node	Edge	Attribute	Class
Phylogenetic [122, 123]	344	343	-	-
CS PhDs [124]	1,025	1,043	-	-
Diseases [125, 126]	516	1,188	-	-
Cora [8]	2,708	5,429	1,433	7
Citeseer [8]	3,312	4,552	3,703	6
Wiki [86]	2,405	17,981	4,973	17
Pubmed [8]	19,717	44,338	500	3
BlogCatalog [127]	5,196	171,743	8,189	6
Amazon Photo [128]	7,650	119,081	745	8
ImageNet-10 [129]	13,000	-	27,648	10
ImageNet-Dogs [129]	19,500	-	27,648	15
ImageNet-BNCR	11,700	-	27,648	9

complex network datasets and 3 image datasets. For node clustering and link prediction tasks on the 9 network datasets, we evaluate HGCAE-P and HGCAE-H, which denote HGCAE models whose latent spaces are Poincaré ball and hyperboloid, respectively. For the tasks of image clustering and visual data analysis, we use HGCAE-P because Poincaré ball is a powerful tool for visualizing and analyzing properties of hyperbolic visual embeddings. The statistics of the datasets are summarized in Table 4.1.

4.4.1 Datasets

Network Datasets. Phylogenetic tree [122, 123] models the generic heritage. CS PhDs [124] represents the relationship between Ph.D. candidates and their advisors in computer science fields. Diseases [125, 126] is a biological network expressing the relationship between diseases. Cora [8], Citeseer [8], Pubmed [8], and Wiki [86] are citation networks whose nodes are scientific papers or web pages and edges represent citation relationships between any two papers or links between any two web pages.

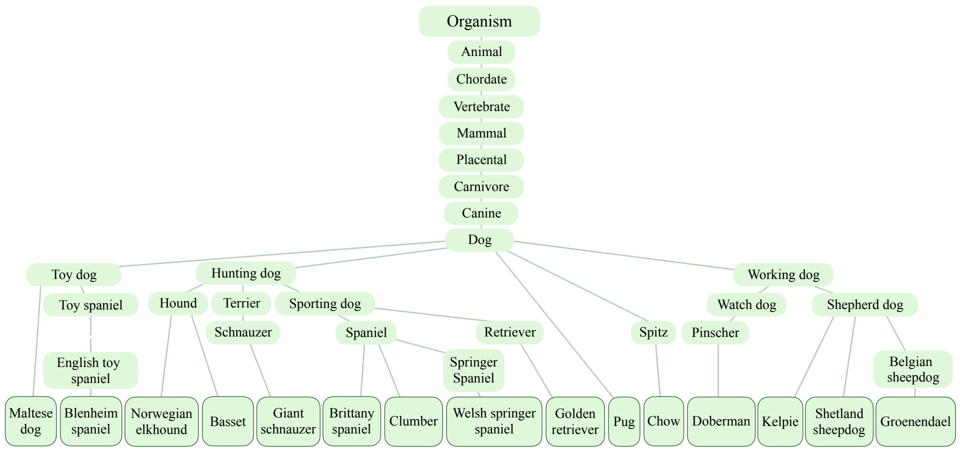


Figure 4.2: Class hierarchy of ImageNet-Dogs[1].

BlogCatalog [127] models a social network among bloggers in the online community. Attribute and label of a node represent the description of each blog and the interest of a blogger, respectively. Amazon Photo [128] is a part of Amazon co-purchase networks whose nodes are goods and edges represent purchase correlations between any two goods. A node attribute indicates the bag-of-words for goods' reviews and its label denotes a product category.

Image Datasets. ImageNet-10 [129] and ImageNet-Dogs [129] are subsets of the ImageNet dataset [2]. ImageNet-10 consists of $13,000$ images from 10 randomly selected subjects. ImageNet-Dogs are $19,500$ images from 15 randomly selected dog breeds. The class hierarchy of ImageNet-Dogs is illustrated in Fig. 4.2. We have constructed a new dataset, ImageNet-BNCR, via randomly choosing 3 leaf classes per root. We chose three roots, *Artifacts*, *Natural objects*, and *Animal*. Thus, there exist 9 leaf classes, and each leaf class contains $1,300$ images in ImageNet-BNCR dataset. For every dataset used for the image clustering task, we used only the training set without the validation set, and images were resized to $96 \times 96 \times 3$.

[1]`http://image-net.org/index`

4.4.2 Compared Methods.

Node Clustering and Link Prediction. We compared HGCAE with seven state-of-the-art unsupervised message passing models which mainly conduct in Euclidean space. GAE [51], VGAE [51], ARGA [52], and ARVGA [52] are graph autoencoders that reconstruct only the affinity matrix using a non-parametric decoder which is not learnable. MGAE [50] is a stacked one-layer graph autoencoder that reconstructs only the node attributes via a linear activation function. GALA [130] is a graph autoencoder that reconstructs only the node attributes through learnable parametric encoder and decoder. DBGAN [131] is a distribution-induced bidirectional generative adversarial network that estimates the structure-aware prior distribution of the representations. GAE [51], VGAE [51], ARGA [52], ARVGA [52], and GALA [130] are constrained to have two-layer autoencoder models, since they report that two-layer structures show the best performances. In the case of MGAE [50] which is a stacked one-layer autoencoder model, we have stacked the layer up to three and reported the best performances. For DBGAN [131], we followed the number of layers in the literature. For every compared method, we followed the hyperparameters in the literature.

Image Clustering. Extensive baselines and state-of-the-art image clustering methods were compared. Several traditional methods including k-means clustering (Kmeans) [91], spectral clustering (SC) [132], agglomerative clustering (AC) [133], and nonnegative matrix factorization (NMF) [134] were also compared. For the representation-based clustering methods, AE [135], CAE [136], SAE [137], DAE [138], DCGAN [139], DeCNN [140], SWWAE [141], and VAE [119] were adopted. Besides, the state-of-the-art image clustering methods including JULE [142], DEC [143], DAC [129], DDC [144], DCCM [145], and PICA [146] were employed. For every compared method, we followed the experimental details in the literature.

4.4.3 Experimental Details.

For every experiment and analysis, HGCAE has two encoder layers and two decoder layers. The dimension of each layer for HGCAE was set to one of $\{2^3, 2^4, ..., 2^{11}\}$. We optimized HGCAE using Adam [147] with learning rate 0.01. As reported in [103], we observe that Euclidean optimization [147] is much more stable than Riemannian optimization [148]. Because of exponential and logarithmic maps, the parameters of our model can be optimized using Euclidean optimization. We experimented with HGCAE for two cases, fixing the curvature of all layers or learning the curvature of each layer, then we reported the best performances. In the case of fixing the curvature of all layers, the curvature K was set to one of $\{-0.1, -0.5, -1, -2\}$. The regularization parameter λ of Eq. (4.16) was set to one of $\{10^{-6}, 10^{-5}, ..., 10^3\}$. The initial values of trainable parameters β and γ in Eq. (4.13) were set to 0. We searched the best hyperparameters which suited well to each dataset by random search. For visual datasets, we construct the mutual k nearest neighbors graph, A, as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } x_i \in \mathrm{NN}_k(x_j) \wedge x_j \in \mathrm{NN}_k(x_i) \\ 0 & \text{otherwise,} \end{cases} \tag{4.17}$$

where x_i and $\mathrm{NN}_k(x_i)$ denote the feature and k Euclidean nearest neighbor set of the i-th image respectively. We set $k = 20$ and $k = 10$ for ImageNet-10 and ImageNet-Dogs, respectively.

Details of Node Clustering and Link Prediction. For the link prediction task, we divided the edges into training edges, validation edges, and test edges as 85%, 5%, and 10%, then we used validation edges for the model convergence. During training for the link prediction task, we only reconstructed training edges in $\mathcal{L}_{REC-A} = \mathbb{E}_{q(H|X,A)}[\log p(\hat{A}|H)]$. For the node clustering task, every edge is reconstructed by the output of the encoder during training. The performance of node clustering was obtained by running k-means clustering [91] on the latent representations (output of the encoder) in the tangent space of the last layer of the encoder.

Details of Image Clustering. The performance of HGCAE on the image clustering task was obtained by running k-means clustering [91] on the latent representations (output of the encoder) in the tangent space of the last layer of the encoder.

Details of Convolutional Autoencoder. We extracted 1000-dimensional features by training a convolutional autoencoder (CAE) [136] on the ImageNet-10 [129] and ImageNet-BNCR datasets on the experiment of Section 4.4.6. We used the encoder part and decoder part as VGG-16 network [149] and five deconvolution layers [140] respectively. We optimized CAE using Adam [147] with learning rate 0.0001 and obtained the feature after 100 epochs.

Details of Image Classification. We obtained the latent representation of ImageNet-10 [129] and ImageNet-BNCR by training CAE on the experiments of Section 4.4.7. For the image classification task, we trained the VGG-11 [149] classifier. We trained the classifier using stochastic gradient descent [150] and used the learning rate scheduler as in [151]. When adding further samples in every training epoch, high, middle, and low HDO samples were chosen by $n\%$ of the original data closest to the boundary, $n\%$ of the original data closest to the median of distance histogram, and $n\%$ of the original data closest to the origin, respectively. We set n for ImageNet-10 and ImageNet-BNCR to 30 and 50 respectively. The learning rates of ImageNet-10 and ImageNet-BCNR were set to 0.01 and 0.0005 respectively. When training BaselineFL, we tried $\{0.5, 1.0, 2.0\}$ for γ in focal loss [152] and reported the best performances. There has been recent research on manipulating the gradient updates based on the prediction difficulty, anchor loss (AL) [153], and we have tried to report the classification performance of AL as well as FL. However, due to the several NaN issues of official AL implementation[2], we could not report the performance of AL.

Table 4.2: Link prediction performances.

	Cora		Citeseer		Wiki		Pubmed		BlogCatalog		Amazon Photo	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
GAE [51]	0.910	0.920	0.895	0.899	0.930	0.948	0.964	0.965	0.840	0.841	0.956	0.948
VGAE [51]	0.914	0.926	0.908	0.920	0.936	0.950	0.944	0.947	0.844	0.846	0.971	0.966
ARGA [52]	0.924	0.932	0.919	0.930	0.934	0.947	**0.968**	0.971	0.857	0.850	0.961	0.954
ARVGA [52]	0.924	0.926	0.924	0.930	0.947	0.948	0.965	0.968	0.837	0.828	0.927	0.909
GALA [130]	0.929	0.937	0.944	0.948	0.936	0.931	0.915	0.897	0.774	0.765	0.918	0.910
DBGAN [131]	0.945	0.951	0.945	0.958	-	-	**0.968**	**0.973**	-	-	-	-
HGCAE-P	0.948	0.947	0.960	0.963	**0.955**	**0.962**	0.962	0.960	**0.896**	**0.886**	**0.982**	**0.976**
HGCAE-H	**0.956**	**0.955**	**0.967**	**0.970**	0.952	0.958	0.962	0.960	0.857	0.850	0.972	0.966

4.4.4 Node Clustering and Link Prediction

Comparison to embeddings in Euclidean latent space. We evaluated the usefulness of hyperbolic representations by the performances of downstream tasks on citation [8, 86], social [127], and co-purchase [128] networks. We compared against the state-of-the-art unsupervised message passing models [50–52, 130, 131] which mainly conduct in Euclidean space. Similar to evaluation metrics used in [130], we used area under curve (AUC) and average precision (AP) to evaluate the performance of the link prediction task while using accuracy (ACC) and normalized mutual information (NMI) for evaluating the node clustering task.

The results of link prediction and node clustering are presented in Tables 4.2 and 4.3 respectively. From the results, we can see that our HGCAE, with the representations of hyperbolic latent spaces, outperforms the existing methods, which use Euclidean latent spaces. Our superior results over their Euclidean counterparts support the fact that unsupervised learning with message passing benefit from the geometry of hyperbolic spaces.

Comparison to embeddings of hyperbolic graph autoencoder. To validate the architecture of HGCAE, we compared its performance with the Poincaré variational

[2] `https://github.com/slryou41/AnchorLoss`

Table 4.3: Node clustering performances.

	Cora		Citeseer		Wiki		Pubmed		BlogCatalog		Amazon Photo	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
Kmeans [91]	0.492	0.321	0.540	0.305	0.417	0.440	0.595	0.315	0.180	0.007	0.267	0.122
GAE [51]	0.532	0.434	0.505	0.246	0.460	0.468	0.686	0.295	0.284	0.112	0.390	0.337
VGAE [51]	0.595	0.446	0.467	0.260	0.450	0.467	0.688	0.310	0.269	0.097	0.418	0.376
MGAE [50]	0.684	0.511	0.660	0.412	0.514	0.485	0.593	0.282	0.423	0.202	0.594	0.475
ARGA [52]	0.640	0.449	0.573	0.350	0.458	0.437	0.680	0.275	0.464	0.270	0.577	0.499
ARVGA [52]	0.638	0.450	0.544	0.261	0.386	0.338	0.513	0.116	0.450	0.250	0.455	0.395
GALA [130]	0.745	0.576	0.693	0.441	**0.544**	**0.503**	0.693	0.327	0.400	0.251	0.512	0.485
DBGAN [131]	0.748	0.560	0.670	0.407	-	-	0.694	0.324	-	-	-	-
HGCAE-P	0.746	0.572	0.693	0.422	0.459	0.467	**0.748**	**0.377**	0.550	0.325	0.781	0.696
HGCAE-H	**0.767**	**0.599**	**0.715**	**0.453**	0.530	0.435	0.711	0.347	**0.741**	**0.578**	**0.817**	**0.722**

Table 4.4: Link prediction task compared with P-VAE.

	Phylogenetic		CS PhDs		Diseases	
	AUC	AP	AUC	AP	AUC	AP
VGAE [51]	0.542	0.540	0.565	0.564	0.898	0.918
P-VAE [113]	0.590	0.555	0.598	0.567	0.923	**0.936**
HGCAE-P	**0.688**	**0.712**	**0.673**	**0.640**	**0.926**	0.914

autoencoder (P-VAE) [113], whose latent space is the Poincaré ball and conducts its message passing in Euclidean space. Three networks, phylogenetic tree [122, 123], Ph.D. advisor-student relationships [124], and disease relationships [125, 126], were used for evaluating performance on link prediction. The latent space of both P-VAE and HGCAE-P is a 5-dimensional Poincaré ball. We report the results in Table 4.4. The proposed HGCAE-P outperforms P-VAE for most cases of the datasets since HGCAE-P considers hyperbolic geometry in the whole auto-encoding processes.

Visualization of citation network. We explored the latent representations of GAE [51] and our models on the Cora dataset [8] by constraining the latent space as a 2-dimensional hyperbolic or Euclidean space. The result is given in Fig. 4.3. On the

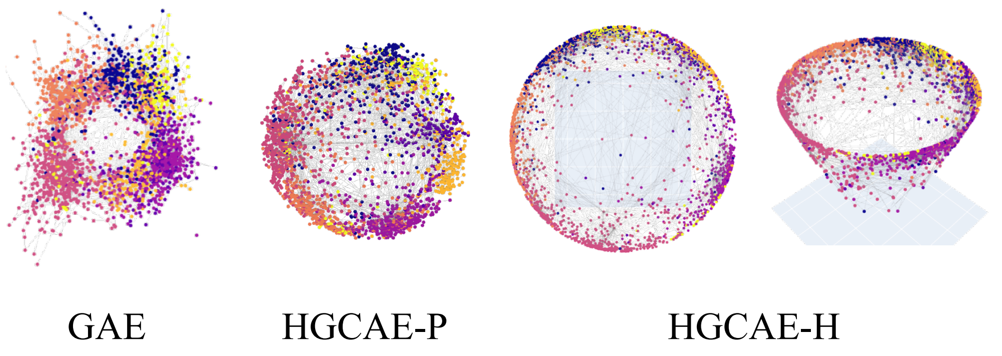
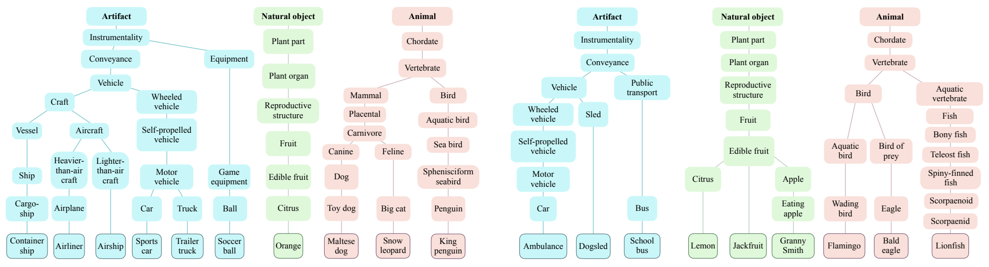


Figure 4.3: 2-dimensional embeddings in Euclidean, Poincaré ball, and hyperboloid latent space on Cora dataset. Same color indicates same class. On hyperbolic latent spaces, most of the nodes are located on the boundary and well-clustered with the nodes in the same class.

results of HGCAE, most of the nodes are located on the boundary of hyperbolic spaces and well-clustered with the nodes in the same class.

4.4.5 Image Clustering



(a) ImageNet-10 (b) ImageNet-BNCR

Figure 4.4: Class hierarchy of ImageNet-10 and ImageNet-BNCR[3].

In this experiment, we illustrate that image clustering can benefit from hyperbolic geometry. The training sets of ImageNet-10 and ImageNet-Dogs [129], which are

[3]http://image-net.org/index

Table 4.5: Image clustering performances.

	ImageNet-10			ImageNet-Dogs		
	ACC	NMI	ARI	ACC	NMI	ARI
Kmeans [91]	0.241	0.119	0.057	0.105	0.055	0.020
SC [132]	0.274	0.151	0.076	0.111	0.038	0.013
AC [133]	0.242	0.138	0.067	0.139	0.037	0.021
NMF [134]	0.230	0.132	0.065	0.118	0.044	0.016
AE [135]	0.317	0.210	0.152	0.185	0.104	0.073
CAE [136]	0.253	0.134	0.068	0.134	0.059	0.022
SAE [137]	0.325	0.212	0.174	0.183	0.112	0.072
DAE [138]	0.304	0.206	0.138	0.190	0.104	0.078
DCGAN [139]	0.346	0.225	0.157	0.174	0.121	0.078
DeCNN [140]	0.313	0.186	0.142	0.175	0.098	0.073
SWWAE [141]	0.323	0.176	0.160	0.158	0.093	0.076
VAE [119]	0.334	0.193	0.168	0.179	0.107	0.079
JULE [142]	0.300	0.175	0.138	0.138	0.054	0.028
DEC [143]	0.381	0.282	0.203	0.195	0.122	0.079
DAC [129]	0.527	0.394	0.302	0.275	0.219	0.111
DDC [144]	0.577	0.433	0.345	-	-	-
DCCM [145]	0.710	0.608	0.555	0.383	0.321	0.182
PICA† [146]	0.850	0.782	0.733	0.324	0.336	0.179
PICA‡ [146]	0.828	0.763	0.692	0.352	0.353	0.201
PICA‡ [146]+HAE	0.821	0.759	0.686	0.338	0.347	0.200
PICA‡ [146]+GAE [51]	0.854	**0.792**	0.737	0.344	0.350	0.199
PICA‡ [146]+HGCAE-P	**0.855**	0.790	**0.741**	**0.387**	**0.360**	**0.226**

† Numbers from literature.

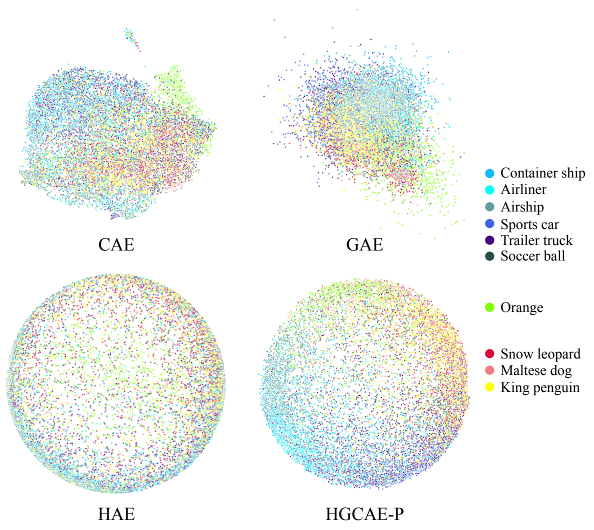
‡ Numbers from our experiments on the official pre-trained networks[4].

subsets of ImageNet [2], are used for evaluation. In the manner of the researches [101,102,105] which impose hyperbolic geometry on the activations of neural networks, we used the activations of PICA [146], one of the most recent models developed for deep image clustering. After obtaining activations from the pre-trained networks of PICA, we built the graph by mutual k nearest neighbors between activations. Then, both the activations and the graph were used as inputs of HGCAE-P. Extensive baselines and state-of-the-art image clustering methods [91, 119, 129, 132–146] were compared. Furthermore, we also trained two autoencoder models, GAE [51], and hyperbolic autoencoder (HAE) whose layers are hyperbolic feed-forward layers [101]. The image clustering results are reported in Table 4.5. The metrics, ACC, NMI and Adjusted Rand Index (ARI), were used for evaluation. The results demonstrate that applying hyperbolic geometry along with using additional information of the approximated image manifold via nearest neighbor graphs can achieve better results than the Euclidean counterparts. We can also observe that HAE, the autoencoder which naively applies hyperbolic geometry, does not work well, while our model performs better via the message passing fully utilizing hyperbolic geometry.

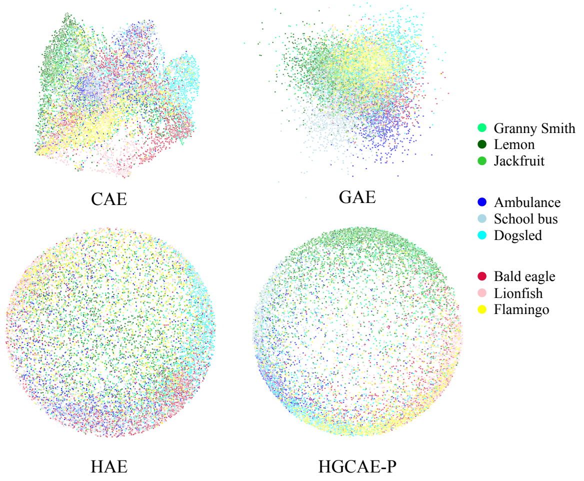
4.4.6 Structure-Aware Unsupervised Embeddings

In this experiment, we observe the unsupervised hyperbolic image embeddings' ability to recognize the latent structure of visual datasets that have hierarchical structures. ImageNet [2] is constructed following the hierarchy of WordNet [117], therefore, its classes of ImageNet-10 [129] also have hierarchical structures. However, it is difficult to explore the effectiveness of hyperbolic embeddings since the classes of ImageNet-10 are biased to a certain root. Thus, we have constructed a new dataset, *ImageNet-BNCR*, that has a *B*alanced *N*umber of *C*lasses across *R*oots. For *ImageNet-BNCR*, we have chosen three roots, *Artifact, Natural objects*, and *Animal*, which have a large number of leaf classes. Each root contains balanced child nodes of *{Ambulance, Dogsled, School bus}*, *{Lemon, Jackfruit, Granny Smith}*, and *{Flamingo, Bald eagle, Lionfish}*, respectively. On the leaf classes of ImageNet-10, *{Container ship, Airliner, Airship, Sports car, Trailer truck, Soccer ball}*, *{Orange}*, and *{Maltese dog, Snow leopard, King penguin}* are the child nodes of the roots *Artifact, Natural objects*, and *Animal*, respectively. The class hierarchies of ImageNet-10 and ImageNet-BNCR are shown in Fig. 4.4.

We extracted 1000-dimensional features by training a convolutional autoencoder



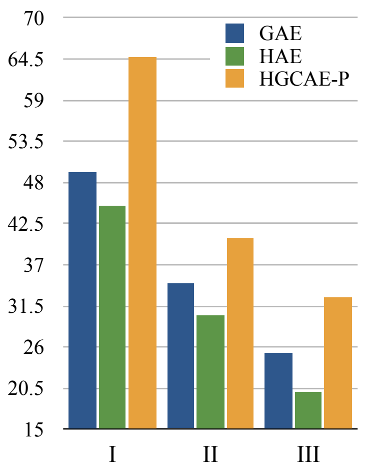
(a) ImageNet-10



(b) ImageNet-BNCR

Figure 4.5: 2-dimensional embeddings of CAE, GAE, HAE, and HGCAE-P on ImageNet-10 and ImageNet-BNCR. Hyperbolic representations belonging to the same root are close to each other near the boundary of the space.

(CAE) [136] on the ImageNet-10 and ImageNet-BNCR datasets. Then, after building the graph using mutual k nearest neighbors between extracted features, we trained three autoencoder models (HGCAE-P, GAE [51], and HAE) whose latent space is

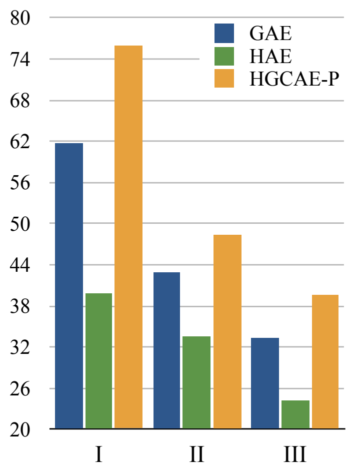


I. *Artifact, Natural object, Animal*

II. *Craft, Wheeled vehicle, Equipment, Citrus, Mammal, Bird*

III. *10 leaf classes*

(a) ImageNet-10



I. *Artifact, Natural object, Animal*

II. *Vehicle, Public transport, Edible fruit, Bird, Aquatic vertebrate*

III. *9 leaf classes*

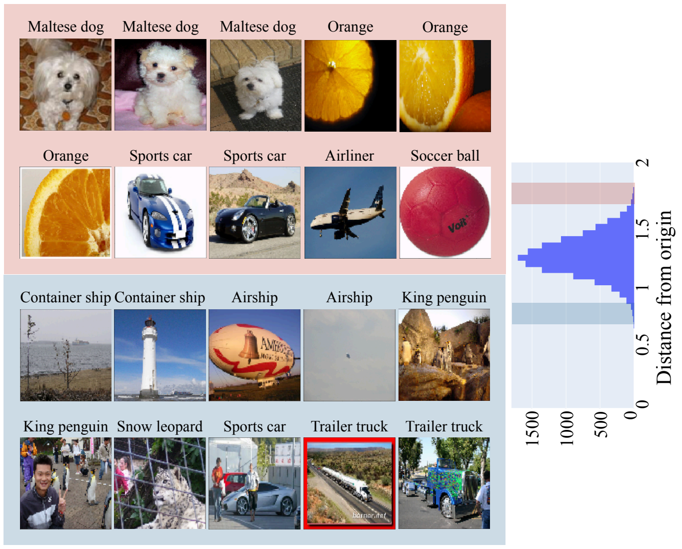
(b) ImageNet-BNCR

Figure 4.6: Clustering accuracy (%) according to the hierarchy of classes on ImageNet-10 and ImageNet-BNCR.

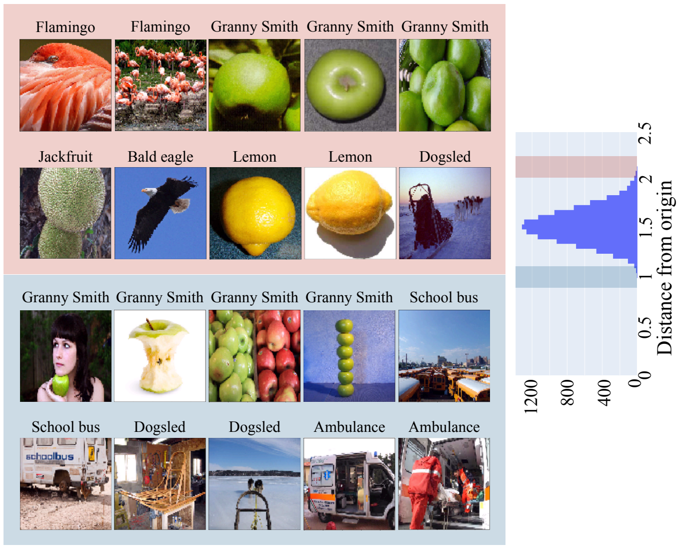
2-dimensional without the ground truth hierarchy structure of labels. The embedding results of the 1000-dimensional CAE features via UMAP [154] and three autoencoders are presented in Fig. 4.5. We can observe that the embeddings of HGCAE-P are better clustered than others, according to the classes of each root in Fig. 4.4. On the ImageNet-10, in the same root *Artifact*, the embeddings of descendants of *Craft* and *Wheeled vehicle* are clustered respectively. The embeddings of the ImageNet-BNCR are clustered more distinctly according to the root of class hierarchy than with ImageNet-10. On the other hand, the embeddings of the root *Natural objects*, *{Lemon, Jackfruit, Granny Smith}*, are located closer to each other since the geodesic distance between each leaf label is small. Our distinction from HAE implies that the additional information on image manifolds approximated by nearest neighbor graphs is helpful. In contrast to the representations of CAE and GAE, we can see that the hyperbolic representations belonging to the same root are located near the boundary of the space. In addition, to quantitatively validate the ability to recognize the latent hierarchical structure of the data without direct learning of label hierarchy, we cluster 2-dimensional embeddings of the three auto-encoders with three ground truth label settings according to the class hierarchy in Fig. 4.4: I. Root nodes, II. Internal nodes, and III. Leaf nodes. The quantitative results (clustering accuracy) on ImageNet-10 and ImageNet-BNCR are reported in Fig. 4.6. HGCAE-P outperforms GAE and HAE in every label hierarchy settings. This might be because the leaf classes whose parent is the same are closely embedded with each other. This analysis empirically demonstrates that unsupervised hyperbolic image embeddings can recognize the latent structure of the visual data that has a hierarchical structure.

4.4.7 Hyperbolic Distance to Filter Training Samples

In this experiment, we show that hyperbolic distance can help to choose training samples beneficial to the generalization ability of neural networks. To this end, we obtained the latent embeddings of ImageNet-10 [129] and ImageNet-BNCR via HGCAE-P model. Then, the hyperbolic distance (Eq. (4.2)) of each embedding from the origin was computed. Fig. 4.7 shows some samples near the boundary or near the origin in the histogram of the hyperbolic distance from embeddings to the origin. We can see that the samples near the boundary can be easily classified, whereas those near the origin are harder to classify. In general, the easy samples are not influential to learn an exact decision boundary. On the other hand, the hard samples make the decision boundary over-fitted, i.e., they work like noises located at the soft margin region near the decision boundary [155]. This illustration intuitively shows that the *H*yperbolic *D*istance from the *O*rigin (HDO) of a sample could give a clue which samples are influential or beneficial to learn the decision boundary crucial for the generalization ability of a classifier.

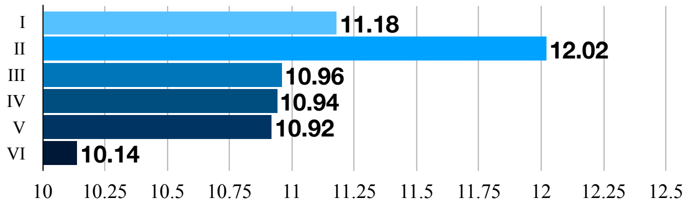


(a) ImageNet-10

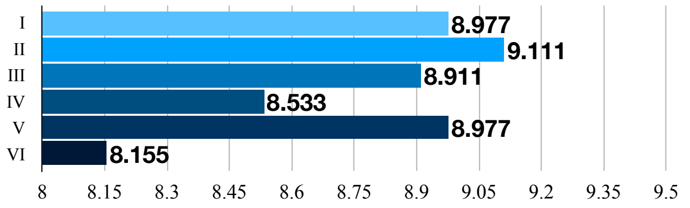


(b) ImageNet-BNCR

Figure 4.7: Histogram and images according to the hyperbolic distance from the origin (HDO) on ImageNet-10 and ImageNet-BNCR. The feature of images inside red (blue) color box have high (low) HDO, so are located near the boundary (origin) of hyperbolic space.



(a) ImageNet-10



(b) ImageNet-BNCR

I. Baseline.	IV. Baseline + High HDO data.
II. BaselineFL.	V. Baseline + Low HDO data.
III. Baseline + Random data.	VI. Baseline + Middle HDO data.

Figure 4.8: Top-1 classification error (%) on ImageNet-10 and ImageNet-BNCR.

To verify this intuition, we conducted an experiment on the image classification task. On ImageNet-10 and ImageNet-BNCR, we trained the VGG-11 [149] classifier by adding further samples near the boundary/median of the distance histogram/origin to the original dataset in every training epoch and evaluated the network via each class' validation set in ImageNet [2]. We compared our results with six settings: I. Baseline: original data with cross-entropy loss, II. BaselineFL: original data with focal loss (FL) [152][5], III. Baseline + Random data adding, IV. Baseline + High HDO data adding, V. Baseline + Low HDO data adding, and VI. Baseline + Middle HDO data adding.

The image classification results are given in Fig. 4.8. As expected, the case V of adding low HDO data in the histogram show similar performances with the baseline. The case IV of adding high HDO data contributes the performance improvements, but the case VI of adding middle HDO data demonstrates the best performance among the

[5]The focal loss tries to focus gradient updates on the samples that the classifier hard to classify.

compared settings. This result empirically verifies that the middle HDO samples are beneficial to learn a reasonable decision boundary which increases the generalization ability of a neural network. Since the supporting samples marginally apart from the decision boundary are crucial for the generalization performance [155], the HDO related with the generalization performance can be interpreted as a measure proportional to the distance of a sample from the decision boundary for a given classification task. In conclusion, we can utilize HDO as a criterion to choose samples for improving the generalization ability of a model for a given dataset.

4.4.8 Ablation Studies

Through link prediction experiments, we validated the effectiveness of two components: learning in the hyperbolic spaces and reconstructing both the graph structure and the node attributes. The experiment was conducted on two citation networks, Cora [8] and Citeseer [8], then the results for link prediction task are presented in Table 4.6. The baseline model is GAE [51], which conducts graph convolution in Euclidean space, does not use an attention mechanism, and reconstructs only the affinity matrix A. In Ablation I, reconstructing both the node attribute X^{Euc} and the graph structure A (Eq. (4.16)) are added to the baseline settings. In Ablation II, operating in hyperbolic space with fixed curvature K is added to Ablation I. In Ablation III, operating in hyperbolic space with fixed curvature K and the geometry-aware attention mechanism (Eq. (4.13)) are added to baseline settings. The results between Ablation I and Ablation II show that the message passing in the hyperbolic space is more effective than that in Euclidean space. Also, the performance gap between Ablation III and Proposed I shows that it is helpful to learn a representation that reflects both the structure of the network and the attributes of each node in hyperbolic space. This component is also valid in Euclidean space, as shown in the gap between Baseline and Ablation I. As shown in the gap between Proposed I and II, the fixed K and the trainable K show similar performance to each other for some datasets, but training K gives an efficient training scheme without multiple learning for searching the best K.

4.5 Further Discussions

4.5.1 Connection to Contrastive Learning

The hyperbolic geometry can be extended to contrastive learning [59]. A recent study [156] has uncovered the link between contrastive learning and deep metric learning. In this respect, it is becoming more significant to find the informative (hard) negative samples, embeddings that are difficult to distinguish from anchors, beyond uniform sampling [157]. Our work empirically showed that *H*yperbolic *D*istance from the *O*rigin (HDO) is an effective criterion for selecting samples without supervision for better generalization. The concept of HDO could be extended to informative negative sampling. Since the embeddings hard to discriminate is equal to those that are hard to classify by the model, the samples near the origin of hyperbolic space can be the impactful negative samples to increase the ability of the unsupervised contrastive learning.

4.5.2 Failure Cases of Hyperbolic Embedding Spaces

The inductive bias of hyperbolic representation learning is assuming that there exist hierarchical relationships in the dataset. Thus if the structure of the graph modeling the relation between data points is close to a tree, the hyperbolic space, a continuous version of a tree, is a suitable latent space. However, not all datasets' latent structures have the topological properties of the tree. For instance, datasets obtained from omnidirectional sensors of drones and autonomous cars are indeed more suitable to latent hyperspherical manifold rather than the hyperbolic manifold [158].

Table 4.6: Ablation studies on link prediction task: The baseline model is GAE which conducts graph convolution in Euclidean space, does not use an attention mechanism and reconstructs only the graph structure A.

	Reconstruct both A and X	Geometry aware attention	in hyperbolic space fixing K	in hyperbolic spaces learning K	Cora		Citeseer	
					AUC	AP	AUC	AP
Baseline: GAE [51]	×	×	×	×	91.0	92.0	89.5	89.9
Ablation I	√	×	×	×	92.7	92.1	94.0	94.8
Ablation II	√	×	√	×	94.6	94.4	95.9	96.3
Ablation III	×	√	√	×	94.5	94.8	96.1	96.4
Proposed I: HGCAE	√	√	√	×	95.4	**95.5**	**96.7**	**97.0**
Proposed II: HGCAE	√	√	×	√	**95.6**	**95.5**	96.5	96.8

[5] `https://github.com/Raymond-sci/PICA`

4.6 Summary

In this chapter, we explored the properties of unsupervised hyperbolic representations. We derived the representations from geometry-aware message passing autoencoders whose whole operations were conducted in hyperbolic spaces. Then, we conducted extensive experiments and analyses on the low-dimensional latent representations in hyperbolic spaces. The experimental results support the conclusion that taking advantage of hyperbolic geometry can improve the performances of unsupervised tasks; node clustering, link prediction, and image clustering. We observed that the proposed method could yield unsupervised hyperbolic image embeddings reflecting the latent structure of the visual datasets that have a hierarchical structure. Lastly, we demonstrated that the hyperbolic distance from origin for a sample could be utilized to determine the additional data crucial for a classifier's generalisation ability.

Chapter 5

Contrastive Learning for Heterogeneous Graphs

5.1 Overview

In recent years, Graph Neural Networks (GNNs) [3, 39, 40, 60, 61] have become the de facto standard for representation learning on graph-structured data. Among the differing architectures for GNNs, Message Passing Neural Networks (MPNNs) [9, 67] in which nodes exchange messages (i.e., representations) along edges, are some of the most well-known and effective mechanisms. Since GNNs were first proposed, the majority of efforts in this field have been aimed at learning representations for homogeneous graphs with a single type of nodes and a single type of edges (i.e., relationships). However, graph-structured datasets in real-world applications are not limited to a single type of nodes and edges. For instance, in the movie network of Figure 5.1 (a), there exist multiple types of nodes (movies, actors, directors, and producers) and multiple types of edges (acting, filming, and producing). This kind of graph that has multiple types of nodes and edges is called *heterogeneous information network* or *heterogeneous graph* [159, 160]. To capture complex relations in heterogeneous graphs, the representation learning model must consider the distinct nature of multiple types of nodes and edges. Thus simply plugging heterogeneous graphs into conventional MPNNs devoted to homogeneous graphs is inadequate because the MPNNs cannot distinguish

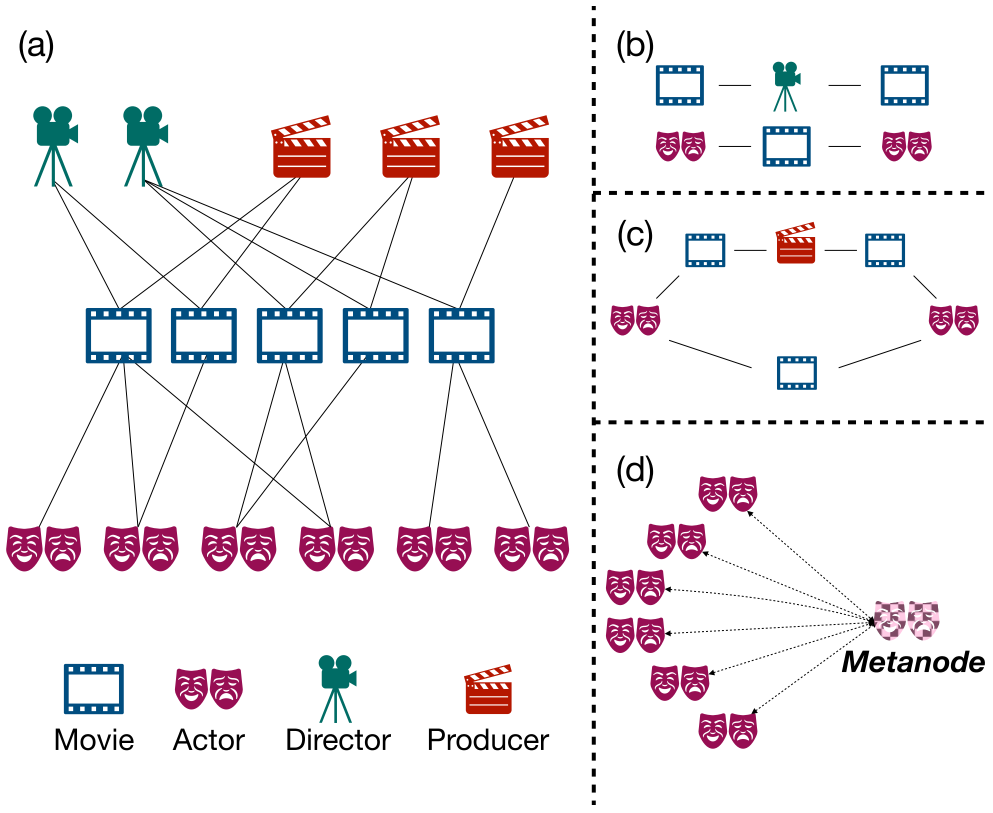


Figure 5.1: (a) An example of heterogeneous movie networks. There exists four types of nodes: *movie, actor, director*, and *producer*. (b) Example of two meta-paths (i.e., movie-director-movie and actor-movie-actor) which are compositions of different types of nodes. (c) A meta-graph which is a composition of multiple meta-paths. (d) The proposed metanode: each metanode aggregates messages of all nodes of each type and returns the aggregated message to each node when passing messages to the next layer.

multiple node and edge types. To deal with this problem, recently, Heterogeneous Graph Neural Networks (HGNNs) [161–168] have been proposed to extract useful knowledge from heterogeneous graphs by leveraging the power of GNNs.

Most of HGNNs are focusing on semi-supervised conditions to learn the representation of nodes by utilizing the ground-truth label of each node. However, real-world datasets are mostly unlabeled and the process of gathering ground truth labels is often an expensive and time-intensive task. Thus, unsupervised representation learning on heterogeneous graphs has become one of the major challenges in graph-structured data learning, as it can pave the way to make use of large amounts of unlabeled data. Especially, when there does not exist any given supervisory signal, it is highly important to contemplate the unique characteristics of heterogeneous graphs. As an example in Figure 5.1, most of the heterogeneous graphs are k-partite graphs whose nodes can be divided into k independent sets. Due to the nature of k-partite graphs, all that is given are sparse inter-type relations (i.e., edges between different types of nodes). However, using only these inter-type relations is not enough to extract useful knowledge from the intricate relations in the data. To resolve this problem, most HGNNs rely on additional predefined relational information, whether their conditions are semi-supervised or unsupervised. The most commonly used methods are *meta-path* [169] and *meta-graph* [170, 171], each of which are a composition of different types of nodes and multiple meta-paths as shown in Figure 5.1 (b) and (c). As we will show later, nearly all meta-paths implicitly derive intra-type relations (i.e., relations between same type of nodes) by manipulating given inter-type relations.

However, there exist three major problems of using predefined methods such as meta-paths for unsupervised heterogeneous graph learning. Firstly, there exist certain limitations on inducing intra-type relations from predefined inter-type relations. When the given inter-type relations are sparse or noisy, induced intra-type relations can also be affected. Secondly, the appropriate composition of nodes and edges (designing meta-paths and meta-graphs) for representation learning requires significant domain-specific knowledge. Thus, it is extremely hard to know which combinations of nodes and edges is suitable for learning useful representations in unsupervised environments. Lastly, although there exist attempts to learn appropriate meta-paths beyond given ones [165], several multiplications of the adjacency matrix are required. Due to the high computational cost of multiple matrix multiplications, their method is limited to very small datasets [172]. Also, it cannot be applied to unlabeled data. This is because their method requires ground truth labels of nodes.

To circumvent the above limitations of current methods, we propose a novel concept of ***metanode*** to construct a simple and powerful MPNN for learning heterogeneous graphs. Metanodes are virtual nodes that exist as many as the number of node types in the heterogeneous graph. Each metanode is connected to all nodes in each type as illustrated in Figure 5.1 (d). By introducing metanodes, message passing is no longer limited to sparse inter-type relations, and every node can directly perform message passing with other nodes of the same type via metanodes. To do so, we can enrich the information on the relationship by adding explicit intra-type relations to the given inter-type relations. There are three advantages of using metanode-based message passing compared to conventional methods. Firstly, since metanodes can model intra-type relations explicitly, we no longer need to reason intra-type relations indirectly. Thus, we do not require any predefined tools such as meta-paths or meta-graphs. Secondly, since each node can exchange messages with all nodes of the same type in a single step via metanodes, it is possible to learn distant but informative nodes through only a few message passing layers. Lastly, since the calculation of metanode representation is simple, new relationships can be established with a very small amount of computation. After introducing the concept of metanode, we propose a metanode-based message passing layer to learn both intra- and inter-type relations effectively. Then we propose a contrastive model for heterogeneous graph learning where the encoder consists of our metanode-based message passing layers.

Through nodewise downstream tasks on four real world heterogeneous graph datasets, we validate the proposed metanode-based message passing neural networks. We confirm that our metanode-based model learns rich relational information and shows competitive performance compared to existing state-of-the-arts relying on meta-paths.

Table 5.1: Predefined meta-paths of real-world datasets. In this table, it can be noticed that most of $\mathcal{R}$ are inter-type relations and $\mathcal{P}$ target on intra-type relations by setting the same type of nodes at both ends of $\mathcal{P}$.

Dataset	$\mathcal{A}$	$\mathcal{R}$	$\mathcal{P}$
DBLP	A, P, T, C	A-P, P-T, P-C	APA, APCPA, APTPA
IMDB	M, D, A	M-D, M-A	MDM, MAM
ACM	P, A, S	P-A, P-S	PAP, PSP
AMiner	P, A, R	P-A, P-R	PAP, PRP
Freebase	M, D, A, P	M-D, M-A, M-P	MAM, MDM, MPM
Last.FM	U, A, T	U-U, U-A, A-T	UU, UAU, UATAU, AUA, AUUA, ATA
Yelp	U, B, Co, Ci, Ca	U-U, U-B, U-Co, B-Ci, B-Ca	UBU, UCoU, UBCiBU, UBCaBU, BUB, BCiB, BCaB, BUCoUB
Douban	U, M, G, L, D, A, T	U-U, U-G, U-M, U-L, M-D, M-T, M-A	MUM, MTM, MDM, MAM, UMU, UMAMU, UMDMU, UMTMU

5.2 Preliminaries

5.2.1 Meta-path

A meta-path [169] $\mathcal{P}$ is defined as a path that has a form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$ (abbreviated as $A_1A_2 \cdots A_{l+1}$) which describes relation between A_1 and $A_{l+1} \in \mathcal{A}$ with a composition of relations $R_1, R_2, \dots, R_l \in \mathcal{R}$, where $\mathcal{A}$ and $\mathcal{R}$ denote sets of node types and edge types of heterogeneous graphs, respectively. Each meta-path can describe a semantic relation between nodes at both ends of the meta-path. For instance, in Figure 5.1 (b), the meta-path of movie-director-movie can describe the relationship between two movies by which director filmed them. Nearly all meta-paths of real-world datasets [159, 163, 166, 173] are implicitly composed for intra-type relations by setting the same type of nodes at both ends of $\mathcal{P}$ using given inter-type relations $\mathcal{R}$ as shown in the Table 5.1.

5.2.2 Representation Learning on Heterogeneous Graphs

Throughout several years, there exist some efforts to learn representations of heterogeneous graphs from random-walk based methods [174–177] to GNNs methods [161–163, 165, 168, 173, 178–181]. Nowadays, HGNNs leveraging the power of

GNNs show remarkable learning ability of intricate relations of multiple types of nodes and edges both of semi-supervised and unsupervised conditions. For instance, in semi-supervised learning, HAN [162] proposed attention-based MPNNs using meta-paths to take into account each semantic meaning of meta-paths. Further, MAGNN [173] considered all the nodes constituting each meta-path to respect intermediate semantic information. GTN [165] is an advanced method to learn new relations (meta-paths) in heterogeneous graphs by multiple multiplications of the adjacency matrix. DiffMG [181] also tries to learn task-specific meta-graphs using neural architecture search method. In unsupervised learning, HetGNN [162] learns a fixed size of correlated neighbors using random walk with restart. NSHE [179] samples sub-graphs and learns via multi-task learning to preserve relations in heterogeneous graphs. Whether the target condition of models is semi-supervised or unsupervised, it can be noticed that many heterogeneous graph learning model relies on predefined meta-paths or meta-graphs. However, depending on meta-paths is undesirable since the results of meta-paths based message passing cannot directly learn the relationships between same type of nodes. Also, it is difficult to know whether meta-paths given in advance is conducive to effective learning.

5.2.3 Contrastive methods for Heterogeneous Graphs

Unsupervised learning on graph-structured data is one of the fundamental problems of machine learning. Early attempts using GNNs [51, 52, 130, 182] were focused on an auto-encoder framework with random-walk based loss function [32]. However, learned representations of these previous models are limited since their models overemphasize proximity or locality information of graph. To overcome this problem, contrastive learning, one of the self-supervised learning methods, are attracting attention. Starting from a pioneering work [1], various contrastive models [55, 57, 63, 183] have been proposed in the field of homogeneous graph learning. Along with the above works, there are some efforts to learn representations of heterogeneous graphs via contrastive methods. HDMI [184] utilizes higher-order mutual information via considering relations of raw features of nodes, learned representations and global summary vectors. HDGI [167] is a contrastive model whose encoder is meta-path based neighborhood aggregation MPNNs and contrasts between an original graph and a corrupted graph. HeCo [166] is another contrastive model that contrasts local representation result of direct neighborhood aggregation and higher-order representation that of meta-path based neighborhood aggregation. Current state-of-the-arts contrastive methods commonly use meta-paths to learned representations. However, they can suffer the limitations of using meta-paths as we mentioned in the above paragraph. Thus, how to effectively learn unsupervised representations of heterogeneous graphs without any predefined information is one of the major tasks to be solved.

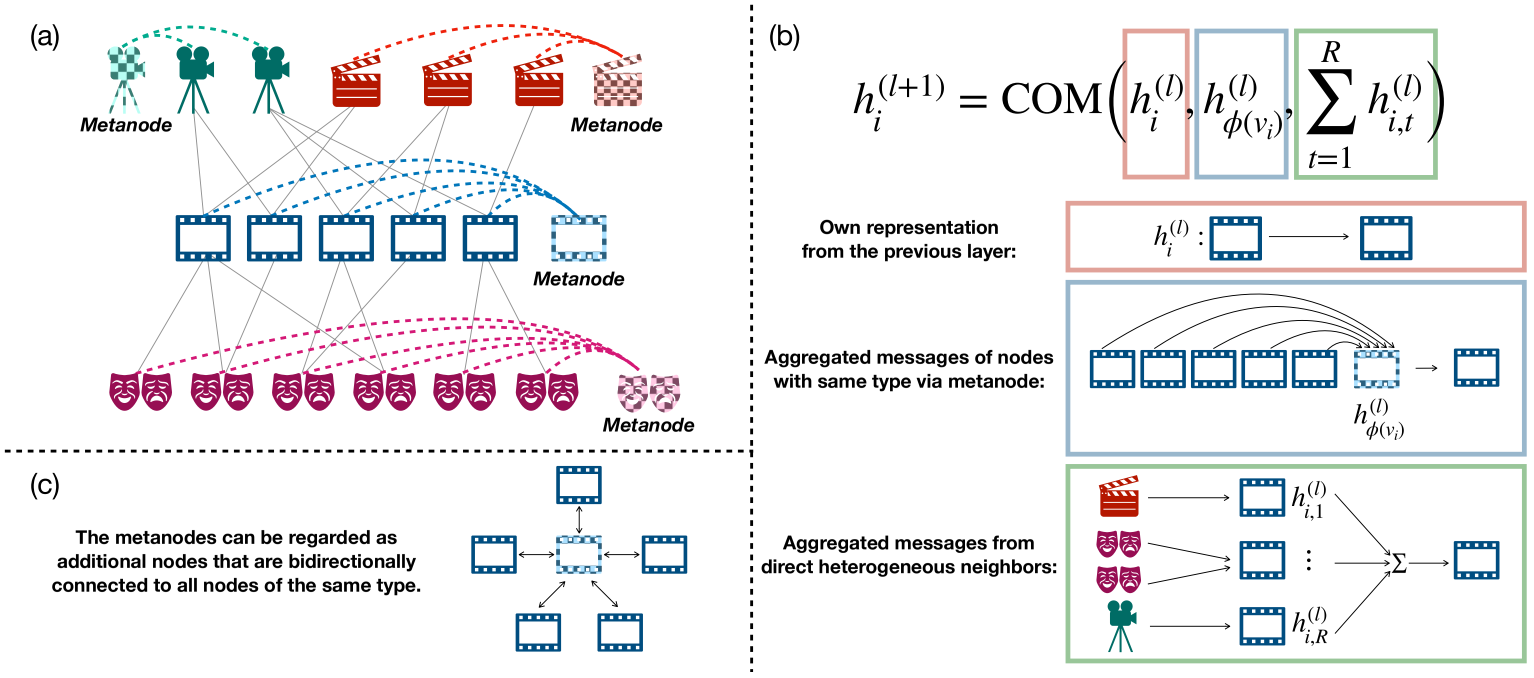
5.3 Methodology

In this section, we present metanode concept, metanode-based message passing layer (MN-MPL), and a contrastive model for heterogeneous graph with encoder adopting the MN-MPL. The graphical descriptions of the metanode, MN-MPL, and contrastive model are illustrated in Figures 5.2 and 5.3.

5.3.1 Definitions

Definition 5.3.1.1. Heterogeneous graph. A graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \phi, \psi)$ is a heterogeneous network with multiple types of nodes and edges, where $\mathcal{V}$ and $\mathcal{E}$ denote the node set and edge set, respectively. Each node $v \in \mathcal{V}$ and edge $e \in \mathcal{E}$ are associated with node type mapping function $\phi(v) : \mathcal{V} \rightarrow \mathcal{A}$ and edge type mapping function $\psi(e) : \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{A}$ and $\mathcal{R}$ denote sets of node types and edge types with $|\mathcal{A}| + |\mathcal{R}| > 2$, respectively. $\mathcal{V}_j$ denotes the node set having j-th node type, that is, $\bigcup_{j=1}^{|\mathcal{A}|} \mathcal{V}_j = \mathcal{V}$.

Definition 5.3.1.2. Metanode. Metanodes $\{\mathbf{v}_j\}_{j=1}^{|\mathcal{A}|}$ are additional nodes that exist as many as the number of node types in the heterogeneous graph. Each metanode $\mathbf{v}_j$ is



(a)
Metanode
Metanode
Metanode
Metanode
(b)
$h_i^{(l+1)} = \mathrm{COM}\left(h_i^{(l)}, h_{\phi(v_i)}^{(l)}, \sum_{t=1}^{R} h_{i,t}^{(l)}\right)$
Own representation from the previous layer:
$h_i^{(l)}$:
Aggregated messages of nodes with same type via metanode:
$h_{\phi(v_i)}^{(l)}$
Aggregated messages from direct heterogeneous neighbors:
$h_{i,1}^{(l)}$
$h_{i,R}^{(l)}$
Σ
(c)
The metanodes can be regarded as additional nodes that are bidirectionally connected to all nodes of the same type.

Figure 5.2: (a) The concept of metanode scheme. Each metanode (checkerboard pattern) is connected to all nodes in the node set of each type using extended edge sets (colored dash line). (b) The proposed metanode-based message passing layer (MN-MPL) takes three components as in Eq. (5.1), indicated by red, blue, and green boxes. (c) The metanode aggregates node messages of same type and returns the aggregated message to every nodes of same type.

connected to all nodes in $\mathcal{V}_j$ via added edges that connect each metanode and the nodes in each node type. The added edges enable message passing between each metanode $\mathbf{v}_j$ and all nodes in $\mathcal{V}_j$.

We illustrate an example of a heterogeneous graph introducing metanodes in Figure 5.2 (a). Introducing a concept of metanode enables explicit modelling of intra-type relations that is hard to infer due to the k-partite structural characteristic of heterogeneous graphs. Compared to conventional methods of using meta-paths or meta-graphs that infer intra-type relations from given inter-type relations indirectly, metanodes can directly establish intra-type relations. Also, unlike meta-paths or meta-graphs that are predefined before learning, metanodes do not require any prior domain knowledge or predefined steps.

5.3.2 Metanode-based Message Passing Layer

We now propose a novel message passing layer using metanodes. The proposed layer takes three components as input: the representation of the previous layer, the metanode representation, and aggregated messages from direct heterogeneous neighbors as shown in Figure 5.2 (b). For a node $v_i \in \mathcal{V}$ who has R different types of immediate neighbors, the metanode-based message passing layer (MN-MPL) is defined by

$$h_i^{(l+1)} = \text{COM}\Big(h_i^{(l)}, h_{\phi(v_i)}^{(l)}, \sum_{t=1}^{R} h_{i,t}^{(l)}\Big), \tag{5.1}$$

where $h_i^{(l)}, h_{\phi(v_i)}^{(l)}$, and $h_{i,t}^{(l)}$ denote the representation of i-th node at l-th layer, the metanode representation of type of i-th node $\phi(v_i)$, and the aggregated representation from t type neighbors of v_i, respectively. For the combination function COM, we apply concatenation $\text{COM}(x_1, \ldots, x_k) = \|_{j=1}^{k} x_j$ or summation $\text{COM}(x_1, \ldots, x_k) = \sum_{j=1}^{k} x_j$. COM includes a nonlinear MLP after concatenation or summation. The metanode representation $h_{\phi(v_i)}^{(l)}$ is defined by sum pooling: $\sum_{j \in \mathcal{V}_{\phi(v_i)}} h_j^{(l)}$, mean pooling: $\frac{1}{|\mathcal{V}_{\phi(v_i)}|} \sum_{j \in \mathcal{V}_{\phi(v_i)}} h_j^{(l)}$, or max pooling: $\max\{h_j^{(l)}\}_{j \in \mathcal{V}_{\phi(v_i)}}$, where $\max$ denotes element-wise max function. To aggregate messages from R types of direct heterogeneous neighbors of v_i, we aggregate messages of each different type of direct neighbors separately first $\{h_{i,1}^{(l)}, \ldots, h_{i,R}^{(l)}\}$, then take a summation to make a single vector representation: $\sum_{t=1}^{R} h_{i,t}^{(l)}$.

The proposed MN-MPL has three major advantages over conventional message passing on heterogeneous graphs. Firstly, each node can exchange messages with nodes of the same type by taking the aggregated messages through metanodes as an input during the proposed message passing process as shown in Figure 5.2 (c). Thus, the proposed message passing scheme makes full use of both inter-type relations via direct heterogeneous neighbors and intra-type relations via metanode representations. Compared to conventional message passing schemes that infer intra-type relations indirectly using predefined meta-paths or meta-graphs, the proposed layer does not require any indirect infer or pre-processing steps before learning. Secondly, through metanode, each node can easily exchange messages with distant nodes without as many layers as the length of meta-path. To learn distant but informative nodes, conventional methods require meta-paths or message passing layers with a length equal to the shortest path distance between two nodes. On the other hand, since each metanode connects all nodes of each type, nodes within each type can consider the others as one-hop neighbors. Thus, distant but informative nodes can be learned with only a small number of MN-MPLs. Lastly, the cost of message passing between intra-type nodes are extremely low. As explained in the paragraph above, the computation of metanode representations is a simple sum, mean or max pooling of the node representations, and requires absolutely no burdensome computational process. Thus, the computation cost of establishing unseen relations of heterogeneous graphs using metanodes is extremely low compared to existing methods such as [165] requiring several adjacency matrix multiplications that attempts an efficient variant [185] very recently.

5.3.3 Contrastive Learning Framework

In this subsection, we explain a framework of contrastive learning using MN-MPL. We apply MN-MPL to the contrastive framework of Deep Graph Infomax [1]. At first, because each node type has attributes of different dimensions, we project each different attribute into a common latent space whose dimension is d using one layer transformation network:

$$h_i^{(0)} = \zeta(W_{\phi(v_i)} x_i + b_{\phi(v_i)}), \tag{5.2}$$

where $x_i \in \mathbb{R}^{d_{\phi(v_i)}}, W_{\phi(v_i)} \in \mathbb{R}^{d \times d_{\phi(v_i)}}, b_{\phi(v_i)} \in \mathbb{R}^d$, and ζ denote the node attribute of v_i, a transformation matrix, a bias vector for type $\phi(v_i)$, and the nonlinear activation, respectively. For constrastive learning, we apply a corruption function C to generate a negative graph $\tilde{G} = C(G)$. We select the corruption function C as type-wise random permutation of node feature matrix $H^{(0)} \in \mathbb{R}^{N \times d}$, where N denotes the number of nodes in the graph. By applying C, each node is given the features of other nodes of the same type.

To learn the representation of each node, we apply MN-MPLs as the encoder network. We share the same encoder network for both the original graph and the corrupted graph to learn the representation of each node. h_i and $\tilde{h}_i$ on node $v_i \in \mathcal{V}$ denote the outputs of the encoder network for the original graph and the corrupted graph, respectively. We extract global summary vector s of the original graph by applying mean pooling $\mathrm{s} = \sigma(\frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} h_i)$, where σ denotes the logistic sigmoid function. Then, we utilize a contrastive objective with binary cross entropy loss function between positive examples (h_i, s) and negative samples $(\tilde{h}_i, \mathrm{s})$ as below:

$$L = \frac{1}{2|\mathcal{V}|} \Big(\sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_G[\log D(h_i, \mathrm{s})] + \sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_{\tilde{G}}[\log(1 - D(\tilde{h}_i, \mathrm{s}))] \Big), \tag{5.3}$$

where $D(h_i, \mathrm{s}) = \sigma(h_i^T W \mathrm{s})$ denotes the discriminator function which is a bilinear network (W is a learnable matrix). Maximizing the objective function L is equal to maximize the mutual information between the representation from the original graph

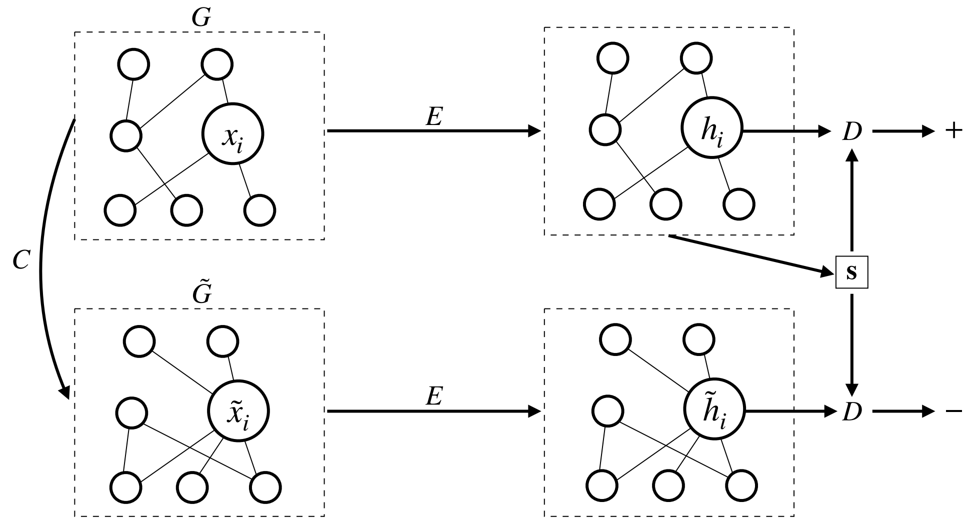


Figure 5.3: Overview of the contrastive model. C, D, E, and s denote a corruption function for generating negative samples, a discriminator function, an encoder network which is composed of our MN-MPL, and a global summary vector, respectively. We referred to the graphical description of [1].

h_i and the global summary vector s from the original graph. The graphical overview of the contrastive learning framework is illustrated at Figure 5.3.

5.4 Experiments

To verify the validity of the proposed metanode-based message passing scheme, we applied our method to node clustering and node classification tasks on the target node type of each dataset. Further, we analyzed the quality of learned representation of nodes and the effectiveness of proposed method via visualization and additional analysis.

Table 5.2: Statistics of datasets.

Dataset	$\mathcal{A}$	$\mathcal{R}$	$\mathcal{P}$
DBLP	Author (A): 4,057 Paper (P): 14,328 Term (T): 7,723 Conference (C): 20	A-P: 19,645 P-T: 85,810 P-C: 14,328	APA APTPA APCPA
ACM	Paper (P): 4,019 Author (A): 7,167 Subject (S): 60	P-A: 13,407 P-S: 4,019	PAP PSP
AMiner	Paper (P): 6,564 Author (A): 13,329 Reference (R): 35,890	P-A: 18,007 P-R: 58,831	PAP PRP
Freebase	Movie (M): 3,492 Director (D): 2,502 Actor (A): 33,401 Producer (P): 4,459	M-D: 3,762 M-A: 65,341 M-P: 6,414	MDM MAM MPM

5.4.1 Experimental Details

Datasets.

We validated our proposed contrastive learning model based on our MN-MPL using four real-world heterogeneous graph datasets. The statistics of datasets are presented in Table 5.2.

- **DBLP**[1] [173] is a subset of bibliography website of computer science fields. There exist four types of nodes: 4,057 authors, 14,328 papers, 7,723 terms, and 20 confer-

[1] `https://github.com/cynricfu/MAGNN`

Table 5.3: Summary of node classification results ($\% \pm \sigma$).

Datasets	Metric	Split	n2vec	SAGE	GAE	mp2vec	HERec	HetGNN	HAN	DGI	DMGI	HeCo	MN (ours)
DBLP	Ma-F1	20	48.75±1.0	71.97±8.4	90.90±0.1	88.98±0.2	89.57±0.4	89.51±1.1	89.31±0.9	87.93±2.4	89.94±0.4	91.28±0.2	**92.60±0.3**
		40	55.94±1.0	73.69±8.4	89.60±0.3	88.68±0.2	89.73±0.4	88.61±0.8	88.87±1.0	88.62±0.6	89.25±0.4	90.34±0.3	**92.78±0.5**
		60	58.15±0.7	73.86±8.1	90.08±0.2	90.25±0.1	90.18±0.3	89.56±0.5	89.20±0.8	89.19±0.9	89.46±0.6	90.64±0.3	**92.70±0.2**
	Mi-F1	20	48.92±1.0	71.44±8.7	91.55±0.1	89.67±0.1	90.24±0.4	90.11±1.0	90.16±0.9	88.72±2.6	90.78±0.3	91.97±0.2	**93.00±0.3**
		40	56.06±1.1	73.61±8.6	90.00±0.3	89.14±0.2	90.15±0.4	89.03±0.7	89.47±0.9	89.22±0.5	89.92±0.4	90.76±0.3	**93.14±0.5**
		60	58.58±0.8	74.05±8.3	90.95±0.2	91.17±0.1	91.01±0.3	90.43±0.6	90.34±0.8	90.35±0.8	90.66±0.5	91.59±0.2	**93.42±0.2**
	AUC	20	74.84±0.7	90.59±4.3	98.15±0.1	97.69±0.0	98.21±0.2	97.96±0.4	98.07±0.6	96.99±1.4	97.75±0.3	98.32±0.1	**99.05±0.1**
		40	78.54±0.6	91.42±4.0	97.85±0.1	97.08±0.0	97.93±0.1	97.70±0.3	97.48±0.6	97.12±0.4	97.23±0.2	98.06±0.1	**98.59±0.1**
		60	81.74±0.4	91.73±3.8	98.37±0.1	98.00±0.0	98.49±0.1	97.97±0.2	97.96±0.5	97.76±0.5	97.72±0.4	98.59±0.1	**99.20±0.0**
ACM	Ma-F1	20	71.96±1.1	47.13±4.7	62.72±3.1	51.91±0.9	55.13±1.5	72.11±0.9	85.66±2.1	79.27±3.8	87.86±0.2	88.56±0.8	**89.46±0.5**
		40	73.76±0.8	55.96±6.8	61.61±3.2	62.41±0.6	61.21±0.8	72.02±0.4	87.47±1.1	80.23±3.3	86.23±0.8	87.61±0.5	**89.19±0.4**
		60	74.03±0.8	56.59±5.7	61.67±2.9	61.13±0.4	64.35±0.8	74.33±0.6	88.41±1.1	80.03±3.3	87.97±0.4	89.04±0.5	**89.67±0.3**
	Mi-F1	20	70.27±1.4	49.72±5.5	68.02±1.9	53.13±0.9	57.47±1.5	71.89±1.1	85.11±2.2	79.63±3.5	87.60±0.8	88.13±0.8	**89.09±0.5**
		40	73.14±1.0	60.98±3.5	66.38±1.9	64.43±0.6	62.62±0.9	74.46±0.8	87.21±1.2	80.41±3.0	86.02±0.9	87.45±0.5	**88.86±0.4**
		60	72.86±1.0	60.72±4.3	65.71±2.2	62.72±0.3	65.15±0.9	76.08±0.7	88.10±1.2	80.15±3.2	87.82±0.5	88.71±0.5	**89.43±0.4**
	AUC	20	86.31±0.8	65.88±3.7	79.50±2.4	71.66±0.7	75.44±1.3	84.36±1.0	93.47±1.5	91.47±2.3	96.72±0.3	96.49±0.3	**96.77±0.2**
		40	86.75±0.6	71.06±5.2	79.14±2.5	80.48±0.4	79.84±0.5	85.01±0.6	94.84±0.9	91.52±2.3	96.35±0.3	96.40±0.4	**96.99±0.1**
		60	88.11±0.6	70.45±6.2	77.90±2.8	79.33±0.4	81.64±0.7	87.64±0.7	94.68±1.4	91.41±1.9	96.79±0.2	96.55±0.3	**97.61±0.0**
AMiner	Ma-F1	20	60.77±1.5	42.46±2.5	60.22±2.0	54.78±0.5	58.32±1.1	50.06±0.9	56.07±3.2	51.61±3.2	59.50±2.1	71.38±1.1	**73.45±0.5**
		40	67.64±1.1	45.77±1.5	65.66±1.5	64.77±0.5	64.50±0.7	58.97±0.9	63.85±1.5	54.72±2.6	61.92±2.1	73.75±0.5	**75.52±0.6**
		60	68.55±0.1	44.91±2.0	63.74±1.6	60.65±0.3	65.53±0.7	57.34±1.4	62.02±1.2	55.45±2.4	61.15±2.5	**75.80±1.8**	75.09±0.5
	Mi-F1	20	66.01±2.0	49.68±3.1	65.78±2.9	60.82±0.4	63.64±1.1	61.49±2.5	68.86±4.6	62.39±3.9	63.93±3.3	78.81±1.3	**80.53±0.6**
		40	73.05±1.3	52.10±2.2	71.34±1.8	69.66±0.6	71.57±0.7	68.47±2.2	76.89±1.6	63.87±2.9	63.60±2.5	80.53±0.7	**82.26±0.5**
		60	73.55±1.1	51.36±2.2	67.70±1.9	63.92±0.5	69.76±0.8	65.61±2.2	74.73±1.4	63.10±3.0	62.51±2.6	**82.46±1.4**	82.02±0.3
	AUC	20	86.18±0.9	70.86±2.5	85.39±1.0	81.22±0.3	83.35±0.5	77.96±1.4	78.92±2.3	75.89±2.2	85.34±0.9	90.82±0.6	**93.29±0.3**
		40	90.57±0.5	74.44±1.3	88.29±1.0	88.82±0.2	88.70±0.4	83.14±1.6	80.72±2.1	77.86±2.1	88.02±1.3	92.11±0.6	**94.85±0.2**
		60	90.71±0.5	74.16±1.3	86.92±0.8	85.57±0.2	87.74±0.5	84.77±0.9	80.39±1.5	77.21±1.4	86.20±1.7	92.40±0.7	**94.06±0.2**
Freebase	Ma-F1	20	55.60±1.3	45.14±4.5	53.81±0.6	53.96±0.7	55.78±0.5	52.72±1.0	53.16±2.8	54.90±0.7	55.79±0.9	**59.23±0.7**	58.92±0.7
		40	57.58±1.2	44.88±4.1	52.44±2.3	57.80±1.1	59.28±0.6	48.57±0.5	59.63±2.3	53.40±1.4	49.88±1.9	61.19±0.6	**62.73±0.7**
		60	55.54±1.2	45.16±3.1	50.65±0.4	55.94±0.7	56.50±0.4	52.37±0.8	56.77±1.7	53.81±1.1	52.10±0.7	**60.13±1.3**	60.10±0.8
	Mi-F1	20	58.75±1.3	54.83±3.0	55.20±0.7	56.23±0.8	57.92±0.5	56.85±0.9	57.24±3.2	58.16±0.9	58.26±0.9	**61.72±0.6**	61.48±0.1
		40	60.59±1.2	57.08±3.2	56.05±2.0	61.01±1.3	62.71±0.7	53.96±1.1	63.74±2.7	57.82±0.8	54.28±1.6	64.03±0.7	**65.86±0.6**
		60	58.44±1.2	55.92±3.2	53.85±0.4	58.74±0.8	58.57±0.5	56.84±0.7	61.06±2.0	57.96±0.7	56.69±1.2	63.61±1.6	**63.63±0.1**
	AUC	20	73.20±1.1	67.63±5.0	73.03±0.7	71.78±0.7	73.89±0.4	70.84±0.7	73.26±2.1	72.80±0.6	73.19±1.2	76.22±0.8	**76.66±0.8**
		40	75.25±1.0	66.42±4.7	74.05±0.9	75.51±0.8	76.08±0.4	69.48±0.2	77.74±1.2	72.97±1.1	70.77±1.6	78.44±0.5	**79.61±0.7**
		60	74.20±1.4	66.78±3.5	71.75±0.4	74.78±0.4	74.89±0.4	71.01±0.5	75.69±1.5	73.32±0.9	73.17±1.4	78.04±0.4	**78.43±0.8**

ences. The target node type is 'author' and has four ground truth labels: *Database, Data mining, Artificial Intelligence*, and *Information Retrieval*.

- **ACM**[2] [179] is a heterogeneous graph of papers that are published at *KDD, SIGMOD, SIGCOMM, MobiCOMM*, and *VLDB*. There exist three types of nodes: 4,019 papers, 7,167 authors, and 60 subjects. The target node type is 'paper' and has three ground truth labels: *Database, Wireless Communication*, and *Data Mining*.

[2] `https://github.com/Andy-Border/NSHE`

- **AMiner**[3] [186] is a citation heterogeneous information network. There exist three types of nodes: 6,564 papers, 13,329 authors, and 35,890 references. The target node type is ‘paper’ and has three ground truth labels.
- **Freebase**[4] [187] is a movie heterogeneous information network. There exist four types of nodes: 3,492 movies, 2,502 directors, 33,401 actors, and 4,459 producers. The target node type is ‘movie’ and three ground truth labels: *Action, Comedy*, and *Drama*.

Baselines.

We compared our method with graph representation learning methods in three categories: unsupervised homogeneous models, unsupervised heterogeneous models, and semi-supervised heterogeneous model.

- **Unsupervised homogeneous models:** node2vec (n2vec) [32] is a random-walk based model which learns representation solely depending on structure of homogeneous graphs. GraphSAGE (SAGE) [39] is a homogeneous MPNN that enforces nearby nodes having similar representations by random-walk based objective function. GAE [51] is a graph auto-encoder model whose encoder network is a graph convolutional network [3] and decoder network is an inner-product module. This model is optimized by random-walk based link prediction objective function. DGI [1] is a contrastive learning model on homogeneous graphs, in which positive examples from the original graph contrast with negative examples from the corrupted graph by optimization process using Eq. (5.3).
- **Unsupervised heterogeneous models:** Metapath2vec (mp2vec) [174] learns representation of nodes in heterogeneous graphs by random-walk on meta-paths and heterogeneous skip-gram model [188]. This paper and the below papers that use the

[3]`https://tinyurl.com/2p9x557w`

[4]`https://github.com/dingdanhao110/Conch`

meta-paths below use the meta-paths in Table 5.2. HERec [178] is a graph representation model using the random-walk based on meta-paths and DeepWalk [31]. This model is optimized by the specific recommendation task. HetGNN [162] is a heterogeneous MPNN that samples the fixed size of heterogeneous neighbor nodes using random walk with restart. This model utilizes bi-directional LSTM module [189] to aggregate the sampled neighbors and is optimized by random-walk objective function. DMGI [190] is a heterogeneous contrastive learning method by introducing consensus regularization framework. Similar to DGI [1], in DMGI, positive examples from the original graph contrast with negative examples from the corrupted graph. HeCo [166] is state-of-the-art in unsupervised representation learning methods for heterogeneous graphs. In this method, the representation from network-schema-based encoder contrasts with that of meta-path-based encoder. The method also contains several techniques such as view masking mechanism and positive sample selection strategy.

- **Semi-supervised heterogeneous model:** HAN [163] proposes a meta-path-based attention mechanism for heterogeneous graphs, which uses two-step attention mechanisms: meta-path-based neighbor node-level attention and semantic-level attention to learn the importance of each meta-path.

Experimental settings.

For comparison methods based on random-walk, we followed the settings of [166]. Specifically, for metapath2vec, HERec, and HetGNN, the number of walks per node, the walk length, and the window size were set to $40, 100$, and 5, respectively. For GraphSAGE, GAE, DGI, metapath2vec, and HERec, the performances of all meta-path instances were measured and the best performance was reported. For parameter settings other than those mentioned above, we followed the original setting of each paper.

In the experimental setup of our method, we did not use any meta-paths or meta-graphs. We applied Xavier uniform distribution for the parameter initialization [191] and used ADAM [147] for optimization. When conducting transformation of initial node features, Eq. (5.2), we applied batch normalization [192] before nonlinear activation. For COM in Eq. (5.1), we used summation for DBLP and ACM, and used concatenation for AMiner and Freebase. For the direct heterogeneous neighbor aggregation of our MN-MPL, we assigned GraphSAGE [39] modules as many numbers as the edge types in the dataset to compute $h_{i,1}^{(l)}, \ldots, h_{i,R}^{(l)}$. There do not exist features for 'author', 'subject' node types in ACM and every node type in AMiner and Freebase. In these cases, some methods assign one-hot vectors as a unique node identifier for those types that do not have features. However, this one-hot vector strategy does not suitable for contrastive learning methods that generate negative samples by random permutation of features, including ours. This is because a one-hot vector of negative samples can still serve as a unique node identifier after random permutations and cannot produce a useful supervising signal for contrastive learning. Thus, we used node2vec [32] to extract the structural feature of each node that does not have features. The feature extraction is done after removing all information about the types of nodes and edges by transforming from a heterogeneous graph to a homogeneous graph. Unlike some methods that apply message passing to the nodes of target node type only, we apply our MN-MPL to nodes of every type in the dataset.

5.4.2 Node Classification

We conducted node classification to see how useful the learned representation of the metanode-based contrastive learning is. For each dataset, we selected $20, 40, 60$ nodes per class for training set, $1,000$ nodes for validation set, and $1,000$ nodes for test set. We trained and tested a single layer of logistic regression classifier, and used Macro-F1, Micro-F1, and AUC for evaluation metrics. The average value and standard deviation after executing each model 10 times are reported in Table 5.3.

The results demonstrate that our methods can achieve outstanding results compared to the existing homogeneous models and heterogeneous models. Especially, in most

Table 5.4: Summary of node clustering results (%).

	DBLP		ACM		AMiner		Freebase	
	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
n2vec	21.48	14.70	41.71	34.77	32.04	14.36	16.43	17.27
SAGE	51.50	36.40	29.20	27.72	15.74	10.10	9.05	10.49
GAE	72.59	77.31	27.42	24.49	28.58	20.90	19.03	14.10
mp2vec	73.55	77.70	48.43	34.65	30.80	25.26	16.47	17.32
HERec	70.21	73.99	47.54	35.67	27.82	20.16	19.76	19.36
HetGNN	69.79	75.34	41.53	34.81	21.46	26.60	12.25	15.01
DGI	59.23	61.85	51.73	41.16	22.06	15.93	18.34	11.29
DMGI	70.06	75.46	51.66	46.64	19.24	20.09	16.98	16.91
HeCo	74.51	80.17	56.87	56.94	32.26	28.64	**20.38**	**20.98**
MN (ours)	**77.19**	**82.25**	**59.23**	**60.31**	**35.32**	**30.60**	19.05	18.87

cases, the proposed method (MN), that does not rely on any predefined composition (meta-paths or meta-graphs) of hetrogeneous nodes, shows competitive results compared to state-of-the-arts (mp2vec, DMGI, HeCo, etc.) in unsupervised heterogeneous models. Also, it can be seen that our method shows outstanding performances compared to even with a semi-supervised model (HAN). We have also observed that, for AMiner and Freebase datasets where the feature of the target node type is not given, homogeneous models can achieve similar performances with heterogeneous models. Specifically, n2vec and GAE show classification performances close to those of several heterogeneous models such as mp2vec, HERec, HetGNN. We conjecture that the reason for this result is that the node feature plays an important role in distinguishing different types of nodes of heterogeneous graphs.

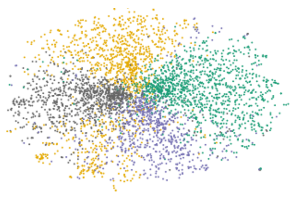
5.4.3 Node Clustering

We conducted node clustering by applying k-means clustering algorithm to the learned representation of each model. The clustering performance is measured by Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI). Table 5.4 reports the average value after executing each model 10 times to consider random initialization of k-means clustering algorithm.

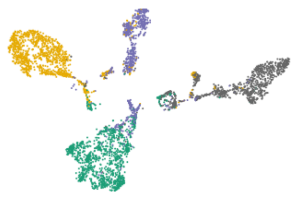
For most cases, the proposed method shows outstanding performances compared to state-of-the-art models. The results of DMGI, HeCo, and our method demonstrate that the contrastive learning framework is effective to learn representations of heterogeneous graphs in unsupervised environments. We observed that clustering performance of every model shows poor performance on Freebase compared to other datasets, DBLP, ACM, and AMiner. Similar to [173]'s analysis on IMDB movie dataset, we guess the cause of this result comes from the noisy labels of the movie genres. Every movie can has multiple genres, but for classification task, only one genre was selected as a label among them. As an evidence for this conjecture, we found that different movie genre labels, *Action, Adventure*, and *Crime*, were used in another paper [193] for the same Freebase dataset we used that labelled *Action, Comedy*, and *Drama*.

5.4.4 Visualization

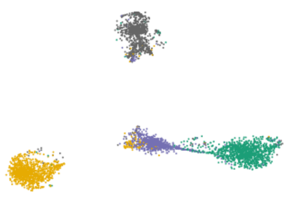
For qualitative analysis of the proposed method, we visualize the learned representation of target node type of each dataset. By using t-SNE [194], we obtained 2-dimensional projections of learned representations. The visualization results are shown in Figure 5.4. To measure the quality of projected representations of each model, we calculated a silhouette score [195]. Our method can distinguish node classes better than comparison methods. The projected representations of n2vec overlaps a lot among other classes due to the limited learning ability of n2vec. The projected representations of HeCo show discriminability among classes. However, in some cases, nodes of the same class are not grouped, resulting in a lower silhouette score compared to our method.


(a) DBLP

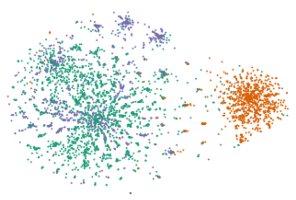
n2vec: 0.1944


(b) DBLP

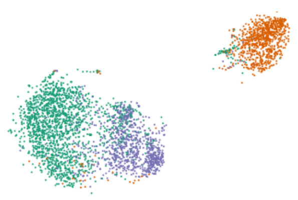
HeCo: 0.520


(c) DBLP

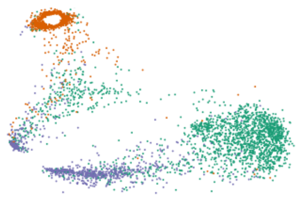
MN: 0.6351


(d) ACM

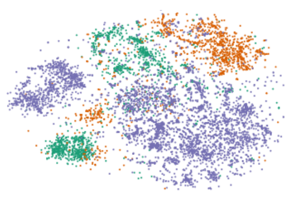
n2vec: 0.2035


(e) ACM

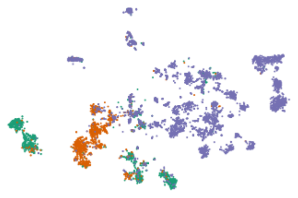
HeCo: 0.4436


(f) ACM

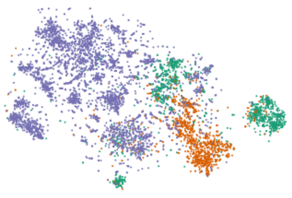
MN: 0.4743


(g) AMiner

n2vec: 0.0037


(h) AMiner

HeCo: 0.1681


(i) AMiner

MN: 0.1692

Figure 5.4: The two-dimensional projections of learned representations of n2vec, HeCo, and our methods (MN) for DBLP, ACM, and AMiner are illustrated. Silhouette score of each projected representations are provided below each subfigure. Same color of nodes share same class label.

5.4.5 Effectiveness of Metanodes

We conducted further experiment to validate the effectiveness of the proposed metanode. We designed an extended model of HeCo [166], one of the state-of-the-art contrastive learning model, via introducing metanode. In the framework of HeCo, there are two encoders, network schema encoder and meta-path encoder, for contrastive learning.

Table 5.5: Effectiveness of metanode via extended HeCo model on node classification task (%).

Datasets	Metric	Split	HeCo	HeCo+MN
DBLP	Ma-F1	20	91.28	**91.41**
		40	90.34	**90.83**
		60	90.64	**91.15**
	Mi-F1	20	91.97	**92.04**
		40	90.76	**91.20**
		60	91.59	**91.77**
	AUC	20	**98.32**	98.17
		40	98.06	**98.12**
		60	**98.59**	98.39
ACM	Ma-F1	20	88.56	**90.06**
		40	87.61	**88.95**
		60	89.04	**89.27**
	Mi-F1	20	88.13	**89.79**
		40	87.45	**88.83**
		60	88.71	**88.95**
	AUC	20	96.49	**97.40**
		40	96.40	**96.97**
		60	96.55	**96.59**
AMiner	Ma-F1	20	71.38	**71.74**
		40	**73.75**	73.14
		60	75.80	**76.68**
	Mi-F1	20	78.81	**78.99**
		40	80.53	**81.22**
		60	82.46	**83.11**
	AUC	20	**90.82**	90.35
		40	92.11	**92.44**
		60	92.40	**92.90**

We introduced metanode representation on meta-path encoder of HeCo by adding metanode representation to the representations of each node after aggregating messages from meta-path neighbors and before calculating semantic level attention (between Eq. (6) and Eq. (7) of HeCo paper). We applied mean pooling to compute metanode representation and measured the node classification performance on DBLP, ACM, and

AMiner under the same experimental environments. The node classification results of HeCo+MN, the metanode-based extended model, and HeCo are presented in Table 5.5. For most cases, we can see that introducing metanode to HeCo that relies on meta-paths can improve the classification performances. In contrast to HeCo+MN, in the original meta-path encoder of HeCo, each node can only aggregate its local meta-path neighbors of same type. By introducing metanodes, representation of HeCo+MN can learn both global structures of the same type such as distributions from mean pooling, and local neighbors of the same type. The improved performance of HeCo+MN in Table 5.5 confirms that existing heterogeneous graph learning models can benefit from the introduction of metanodes.

5.5 Summary

To diversify relations for comprehensive learning of heterogeneous graphs, we proposed a simple and powerful concept of metanode from understanding of unique structural characteristics of heterogeneous graphs. Each metanode enables each node to easily exchange messages with any nodes having same type with itself. By introducing metanodes, each node can take into account information both of heterogeneous neighbors and its corresponding node type. In addition, we proposed a metanode-based message passing layer (MN-MPL) and a contrastive learning framework using MN-MPL. The proposed method is free from problems depending on predefined additional tools such as meta-paths or meta-graphs. The proposed method was validated qualitatively and quantitatively by conducting node classification and node clustering tasks on real-world heterogeneous graphs. We observed competitive performances across various tasks compared to state-of-the-arts that are elaborately designed to use meta-paths. Our results illustrate that meta-paths or meta-graphs are not essential to unsupervised learning of heterogeneous graphs.

In the future, we will investigate more about message passing using metanodes.

Through metanode-based message passing, messages can be received from nodes of the same type, but not all of the same type have the same importance. Thus, by introducing attention modules such as transformer [196], our model can enable more effective message exchange through computation of importance among nodes of same type.

Chapter 6

Conclusions

In this dissertation, we proposed unsupervised graph learning models using graph neural networks for three representative graph structures: homogeneous graphs, tree-like graphs, and heterogeneous graphs. First, we have proposed a novel autoencoder framework which can extract low-dimensional latent representations from a homogeneous graph. We designed a symmetric graph convolutional autoencoder architecture where the encoder performs Laplacian smoothing while the decoder performs Laplacian sharpening, the opposite of smoothing operation. Also, to prevent numerical instabilities, we designed a new representation of Laplacian sharpening with spectral radius 1 by incorporating the concept of the signed graph. Second, we have explored the properties of unsupervised hyperbolic representations. We derived the representations from geometry-aware message passing autoencoders whose whole operations were conducted in hyperbolic spaces. We introduced a self-attention mechanism using the distance between node representations in hyperbolic space when conducting message passing to consider information of hierarchical relations containing in hyperbolic geometry. Third, we have proposed a simple and powerful concept of metanode from understanding of unique structural characteristics of heterogeneous graphs to diversify relations for comprehensive learning of heterogeneous graphs. Each metanode enables each node to easily exchange messages with any nodes having same type with itself. By introducing metanodes, each node can take into account information both of heterogeneous neighbors and its corresponding node type. In addition, we proposed a metanode-based message passing layer and a contrastive learning framework. The proposed method is free from problems depending on predefined additional tools such as meta-paths or meta-graphs. We conducted extensive experiments and analyses on the low-dimensional node representations obtained from the proposed unsupervised graph representation learning methods. The proposed methods were validated on several graph-related downstream tasks such as node clustering, node classification, and link prediction, and showed improved performances due to the model architecture that considers the properties of each graph structure.

Bibliography

[1] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[5] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. *arXiv preprint arXiv:1801.07455*, 2018.

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[7] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[8] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.

[11] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[12] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[13] Alex Bavelas. Communication patterns in task-oriented groups. *The journal of the acoustical society of America*, 22(6):725–730, 1950.

[14] Mark EJ Newman. The mathematics of networks. *The new palgrave encyclopedia of economics*, 2(2008):1–12, 2008.

[15] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[16] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[17] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[18] Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

[19] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pages 488–495. PMLR, 2009.

[20] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[21] AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[22] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[23] Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

[24] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[25] Helmut Lutkepohl. Handbook of matrices. *Computational statistics and Data analysis*, 2(25):243, 1997.

[26] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[27] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[28] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 14, 2001.

[29] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

[30] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 701–710. ACM, 2014.

[32] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 855–864, 2016.

[33] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks*, volume 2, pages 729–734, 2005.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[35] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[36] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

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

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[39] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[40] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

[42] James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[43] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[44] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

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

[46] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

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

[48] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[49] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[50] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 889–898. ACM, 2017.

[51] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[52] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI*, pages 2609–2615, 2018.

[53] Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Semi-implicit graph variational auto-encoders. *Advances in neural information processing systems*, 32, 2019.

[54] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021.

[55] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

[56] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[57] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. *arXiv preprint arXiv:2006.05582*, 2020.

[58] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13366–13378, 2019.

[59] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607, 2020.

[60] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871, 2019.

[61] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[62] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

[63] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.

[64] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

[65] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743. PMLR, 2019.

[66] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*, 2021.

[67] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[68] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020.

[69] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[70] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[71] David Lazer, Alex Sandy Pentland, Lada Adamic, Sinan Aral, Albert Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009.

[72] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017.

[73] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. *arXiv preprint arXiv:1712.00268*, 2017.

[74] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[75] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer graphics and Interactive techniques*, pages 351–358. ACM, 1995.

[76] Hong Hai Li and Jiong Sheng Li. Note on the normalized laplacian eigenvalues of signed graphs. *Australas. J. Combin*, 44:153–162, 2009.

[77] René Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

[78] Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *Advances in Neural Information Processing Systems*, pages 24–33, 2017.

[79] Pan Zhou, Yunqing Hou, and Jiashi Feng. Deep adversarial subspace clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1596–1604, 2018.

[80] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[81] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[82] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[83] Rostislav I Grigorchuk and Andrzej Zuk. On the asymptotic spectrum of random walks on infinite families of graphs. *Random Walks and Discrete Potential Theory (Cortona, 1997), Sympos. Math*, 39:188–204, 1999.

[84] Can-Yi Lu, Hai Min, Zhong-Qiu Zhao, Lin Zhu, De-Shuang Huang, and Shuicheng Yan. Robust and efficient subspace segmentation via least squares re-

gression. In *European conference on computer vision*, pages 347–360. Springer, 2012.

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

[86] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In *IJCAI*, pages 2111–2117, 2015.

[87] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-20). 1996.

[88] Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):643–660, 2001.

[89] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[90] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[91] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[92] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM International Conference on Web Search and Data Mining*, pages 587–596. ACM, 2013.

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

[94] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[95] Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, pages 539–547, 2012.

[96] Jonathan Chang and David Blei. Relational topic models for document networks. In *Artificial Intelligence and Statistics*, pages 81–88, 2009.

[97] Rongkai Xia, Yan Pan, Lei Du, and Jian Yin. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *AAAI*, pages 2149–2155, 2014.

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

[99] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 6338–6347, 2017.

[100] Maximillian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pages 3779–3788, 2018.

[101] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems*, pages 5345–5355, 2018.

[102] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam

Santoro, and Nando de Freitas. Hyperbolic attention networks. In *International Conference on Learning Representations*, 2019.

[103] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 4869–4880, 2019.

[104] Gregor Bachmann, Gary Bécigneul, and Octavian-Eugen Ganea. Constant curvature graph convolutional networks. *arXiv preprint arXiv:1911.05076*, 2019.

[105] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6428, 2020.

[106] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

[107] Guillaume Bouchard, Sameer Singh, and Theo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *2015 AAAI Spring Symposium Series*, 2015.

[108] Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. In *Advances in Neural Information Processing Systems*, pages 1179–1187, 2014.

[109] Yoshihiro Nagano, Shoichiro Yamaguchi, Yasuhiro Fujita, and Masanori Koyama. A wrapped normal distribution on hyperbolic space for gradient-based learning. In *International Conference on Machine Learning*, pages 4693–4702, 2019.

[110] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems*, pages 8228–8239, 2019.

[111] Hervé Fournier, Anas Ismail, and Antoine Vigneron. Computing the gromov hyperbolicity of a discrete metric space. *Information Processing Letters*, 115(6-8):576–579, 2015.

[112] Shaoteng Liu, Jingjing Chen, Liangming Pan, Chong-Wah Ngo, Tat-Seng Chua, and Yu-Gang Jiang. Hyperbolic visual embedding learning for zero-shot recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9273–9281, 2020.

[113] Emile Mathieu, Charline Le Lan, Chris J. Maddison, Ryota Tomioka, and Yee Whye Teh. Continuous hierarchical representations with poincaré variational auto-encoders. In *Advances in Neural Information Processing Systems*, 2019.

[114] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Adversarial autoencoders with constant-curvature latent manifolds. *Applied Soft Computing*, 81:105511, 2019.

[115] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[116] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[117] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[118] Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. In *International Conference on Learning Representations*, 2019.

[119] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[120] Peter Petersen, S Axler, and KA Ribet. *Riemannian geometry*, volume 171. Springer, 2006.

[121] Abraham Albert Ungar. A gyrovector space approach to hyperbolic geometry. *Synthesis Lectures on Mathematics and Statistics*, 1(1):1–194, 2008.

[122] Wolfgang Karl Hofbauer, Laura Lowe Forrest, Peter M Hollingsworth, and Michelle L Hart. Preliminary insights from dna barcoding into the diversity of mosses colonising modern building surfaces. *Bryophyte Diversity and Evolution*, 38(1):1–22, 2016.

[123] MJ Sanderson, MJ Donoghue, W Piel, and T Eriksson. Treebase: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *American Journal of Botany*, 81(6):183, 1994.

[124] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory social network analysis with Pajek: Revised and expanded edition for updated software*, volume 46. Cambridge University Press, 2018.

[125] Kwang-Il Goh, Michael E Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007.

[126] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[127] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 817–826. ACM, 2009.

[128] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[129] Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Deep adaptive image clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5879–5887, 2017.

[130] Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6519–6528, 2019.

[131] Shuai Zheng, Zhenfeng Zhu, Xingxing Zhang, Zhizhe Liu, Jian Cheng, and Yao Zhao. Distribution-induced bidirectional generative adversarial network for graph representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7224–7233, 2020.

[132] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2005.

[133] K Chidananda Gowda and G Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, 10(2):105–112, 1978.

[134] Deng Cai, Xiaofei He, Xuanhui Wang, Hujun Bao, and Jiawei Han. Locality preserving nonnegative matrix factorization. In *IJCAI*, volume 9, pages 1010–1015, 2009.

[135] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

[136] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[137] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[138] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(12), 2010.

[139] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[140] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2528–2535. IEEE, 2010.

[141] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

[142] Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5147–5156, 2016.

[143] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pages 478–487, 2016.

[144] Jianlong Chang, Yiwen Guo, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Deep discriminative clustering analysis. *arXiv preprint arXiv:1905.01681*, 2019.

[145] Jianlong Wu, Keyu Long, Fei Wang, Chen Qian, Cheng Li, Zhouchen Lin, and Hongbin Zha. Deep comprehensive correlation mining for image clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8150–8159, 2019.

[146] Jiabo Huang, Shaogang Gong, and Xiatian Zhu. Deep semantic clustering by partition confidence maximisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8849–8858, 2020.

[147] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[148] Gary Becigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *International Conference on Learning Representations*, 2019.

[149] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[150] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[151] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

[152] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.

[153] Serim Ryou, Seong-Gyun Jeong, and Pietro Perona. Anchor loss: Modulating loss scale based on prediction difficulty. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5992–6001, 2019.

[154] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[155] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[156] Michael Tschannen, Josip Djolonga, Paul K Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. In *International Conference on Learning Representations*, 2019.

[157] Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations*, 2021.

[158] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *International Conference on Learning Representations*, 2018.

[159] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and Philip S Yu. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *arXiv preprint arXiv:2011.14867*, 2020.

[160] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[161] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional

networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[162] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 793–803, 2019.

[163] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *Proceedings of The Web Conference 2019*, pages 2022–2032, 2019.

[164] Kyung-Min Kim, Donghyun Kwak, Hanock Kwak, Young-Jin Park, Sangkwon Sim, Jae-Han Cho, Minkyu Kim, Jihun Kwon, Nako Sung, and Jung-Woo Ha. Tripartite heterogeneous graph propagation for large-scale social recommendation. *arXiv preprint arXiv:1908.02569*, 2019.

[165] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in Neural Information Processing Systems*, 32:11983–11993, 2019.

[166] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining 2021*, page 1726–1736, 2021.

[167] Yuxiang Ren, Bo Liu, Chao Huang, Peng Dai, Liefeng Bo, and Jiawei Zhang. Hdgi: An unsupervised graph neural network for representation learning in heterogeneous graph. In *AAAI Workshop*, 2020.

[168] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[169] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

[170] Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiao-Li Li. Semantic proximity search on graphs with metagraph-based learning. In *2016 IEEE 32nd International Conference on Data Engineering*, pages 277–288. IEEE, 2016.

[171] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. Meta structure: Computing relevance in large heterogeneous information networks. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1595–1604, 2016.

[172] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, page 1150–1160, 2021.

[173] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020.

[174] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 135–144, 2017.

[175] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings*

of the 2017 ACM on Conference on Information and Knowledge Management, pages 1797–1806, 2017.

[176] Jisu Jeong, Jeong-Min Yun, Hongi Keam, Young-Jin Park, Zimin Park, and Junki Cho. div2vec: Diversity-emphasized node embedding. *arXiv preprint arXiv:2009.09588*, 2020.

[177] Yu He, Yangqiu Song, Jianxin Li, Cheng Ji, Jian Peng, and Hao Peng. Hetespaceywalk: A heterogeneous spacey random walk for heterogeneous information network embedding. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 639–648, 2019.

[178] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(2):357–370, 2018.

[179] Jianan Zhao, Xiao Wang, Chuan Shi, Zekuan Liu, and Yanfang Ye. Network schema preserved heterogeneous information network embedding. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 2020.

[180] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. Heterogeneous graph structure learning for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, Vol.35*, pages 4697–4705, 2021.

[181] Yuhui Ding, Quanming Yao, Huan Zhao, and Tong Zhang. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, page 279–288, 2021.

[182] Jiwoong Park, Junho Cho, Hyung Jin Chang, and Jin Young Choi. Unsupervised hyperbolic representation learning via message passing auto-encoders.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5516–5526, 2021.

[183] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2019.

[184] Baoyu Jing, Chanyoung Park, and Hanghang Tong. Hdmi: High-order deep multiplex infomax. In *Proceedings of the Web Conference 2021*, pages 2414–2424, 2021.

[185] Seongjun Yun, Minbyul Jeong, Sungdong Yoo, Seunghun Lee, Sean S Yi, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks: Learning meta-path graphs to improve gnns. *arXiv preprint arXiv:2106.06218*, 2021.

[186] Binbin Hu, Yuan Fang, and Chuan Shi. Adversarial learning on heterogeneous information networks. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 120–129, 2019.

[187] Xiang Li, Danhao Ding, Ben Kao, Yizhou Sun, and Nikos Mamoulis. Leveraging meta-path contexts for classification in heterogeneous information networks. In *2021 IEEE 37th International Conference on Data Engineering*, pages 912–923. IEEE, 2021.

[188] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[189] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[190] Chanyoung Park, Donghyun Kim, Jiawei Han, and Hwanjo Yu. Unsupervised attributed multiplex network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5371–5378, 2020.

[191] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[192] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[193] Xiang Li, Ben Kao, Yudian Zheng, and Zhipeng Huang. On transductive classification in heterogeneous information networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 811–820, 2016.

[194] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(11), 2008.

[195] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[196] Devin Kreuzer, Dominique Beaini, William L Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *arXiv preprint arXiv:2106.03893*, 2021.

초 록

그래프 데이터에 대한 비지도 표현 학습의 목적은 그래프의 구조와 노드의 속성을 잘 반영하는 유용한 노드 단위 혹은 그래프 단위의 벡터 형태 표현을 학습하는 것이다. 최근, 그래프 데이터에 대해 강력한 표현 학습 능력을 갖춘 그래프 신경망을 활용한 비지도 그래프 표현 학습 모델의 설계가 주목을 받고 있다. 많은 방법들은 한 종류의 엣지와 한 종류의 노드가 존재하는 동종 그래프에 대한 학습에 집중을 한다. 하지만 이 세상에 수많은 종류의 관계가 존재하기 때문에, 그래프 또한 구조적, 의미론적 속성을 통해 다양한 종류로 분류할 수 있다. 그래서, 그래프로부터 유용한 표현을 학습하기 위해서는 비지도 학습 프레임워크는 입력 그래프의 특징을 제대로 고려해야만 한다. 본 학위논문에서 우리는 널리 접할 수 있는 세가지 그래프 구조인 동종 그래프, 트리 형태의 그래프, 그리고 이종 그래프에 대한 그래프 신경망을 활용하는 비지도 학습 모델들을 제안한다.

처음으로, 우리는 동종 그래프의 노드에 대하여 저차원 표현을 학습하는 그래프 컨볼루션 오토인코더 모델을 제안한다. 기존의 그래프 오토인코더는 구조의 전체가 학습이 불가능해서 제한적인 표현 학습 능력을 가질 수 있는 반면에, 제안하는 오토인코더는 노드의 피쳐를 복원하며,구조의 전체가 학습이 가능하다. 노드의 피쳐를 복원하기 위해서, 우리는 인코더 부분의 역할이 이웃한 노드끼리 유사한 표현을 가지게 하는 라플라시안 스무딩이라는 것에 주목하여 디코더 부분에서는 이웃 노드의 표현과 멀어지게 하는 라플라시안 샤프닝을 하도록 설계하였다. 또한 라플라시안 샤프닝을 그대로 적용하면 불안정성을 유발할 수 있기 때문에, 엣지의 가중치 값에 음의 값을 줄 수 있는 부호형 그래프를 활용하여 안정적인 라플라시안 샤프닝의 형태를 제안하였다. 동종 그래프에 대한 노드 클러스터링과 링크 예측 실험을 통하여

제안하는 방법이 안정적으로 우수한 성능을 보임을 확인하였다.

둘째로, 우리는 트리의 형태를 가지는 계층적인 관계를 가지고 있는 그래프의 노드 표현을 정확하게 학습하기 위하여 쌍곡선 공간에서 동작하는 오토인코더 모델을 제안한다. 유클리디언 공간은 트리를 사상하기에 부적절하다는 최근의 분석을 통하여, 쌍곡선 공간에서 그래프 신경망의 레이어를 활용하여 노드의 저차원 표현을 학습하게 된다. 이 때, 그래프 신경망이 쌍곡선 기하학에서 계층 정보를 담고 있는 거리의 값을 활용하여 노드의 이웃사이의 중요도를 활용하도록 설계하였다. 우리는 논문 인용 관계 네트워크, 계통도, 이미지 사이의 네트워크등에 대해 제안한 모델을 적용하여 노드 클러스터링과 링크 예측 실험을 하였으며, 트리의 형태를 가지는 그래프에 대해서 제안한 모델이 유클리디언 공간에서 수행하는 모델에 비해 향상된 성능을 보였다는 것을 확인하였다.

마지막으로, 우리는 여러 종류의 노드와 엣지를 가지는 이종그래프에 대한 대조 학습 모델을 제안한다. 우리는 기존의 방법들이 학습하기 이전에 충분한 도메인 지식을 사용하여 설계한 메타패스나 메타그래프에 의존한다는 단점과 많은 이종그래프의 엣지가 다른 노드 종류사이의 관계에 집중하고 있다는 점을 주목하였다. 이를 통해 우리는 사전과정이 필요없으며 다른 종류 사이의 관계에 더하여 같은 종류 사이의 관계도 동시에 효율적으로 학습하게 하는 메타노드라는 개념을 제안하였다. 또한 메타노드를 기반으로하는 그래프 신경망과 대조 학습 모델을 제안하였다. 우리는 제안한 모델을 메타패스를 사용하는 이종그래프 학습 모델과 노드 클러스터링 등의 실험 성능으로 비교해보았을 때, 비등하거나 높은 성능을 보였음을 확인하였다.

주요어: 비지도 그래프 표현 학습, 그래프 신경망, 오토인코더, 대조 학습

학번: 2018-35712