



THE HONG KONG
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library

包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

LEARNING ON GRAPHS
WITH
GRAPH CONVOLUTION

QIMAI LI

PhD

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University
Department of Computing

Learning on Graphs
with
Graph Convolution

Qimai Li

A thesis submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy

August 2022

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Qimai Li

Abstract

Graph convolutional neural networks (GCNN) have been the model of choice for graph representation learning, which is mainly due to the effective design of graph convolution that computes the representation of a node by aggregating those of its neighbors. This thesis reveals the mechanisms behind graph convolution neural networks from the perspective of graph signal processing theory and focuses on developing theoretic algorithms for modeling complex, richly labeled, and large-scale graph-structured data, with applications spanning across computer vision, natural language processing, human action understanding, smart transportation, and malware detection.

We conducted systematic research on analyzing and extending GCNNs from different theoretical perspectives including graph signal processing and spectral graph theory. Our spatial analysis shows that the graph convolution in GCN is a special form of Laplacian smoothing, which is the key reason why GCN works, but it also brings the over-smoothing problem to deep GCN models. Our spectral analysis revisits GCN and classical label propagation methods under a graph filtering framework and shows that they extract useful data representations by a low-pass graph filter.

Our research also contributes to the development of efficient and more powerful GC-

NNs models, and various high-impact real-world applications. With the new theoretical insights, we have developed new, efficient, and more powerful models based on graph convolution for semi-supervised and unsupervised learning, including Improved Graph Convolutional Networks (IGCN), Generalized Label Propagation (GLP), Adaptive Graph Convolution (AGC). We also extend 1-D GCNN to 2-D GCNN so as to explore informative relational information among object attributes, and proposed Dimensionwise Separable 2-D Graph Convolution (DSGC).

The results have been published in various top AI conferences, including AAAI-18 [1], IJCAI-19 [2], CVPR-19 [3], KDD-21 [4], and WWW-22 [5].

Publications

Publications Arising from the Thesis

- **Qimai Li**, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2018*, pages 3538–3545, 2018. **(Cited 1700+ times)**
- **Qimai Li**, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9582–9591. 2019. **(Cited 130+ times)**
- **Qimai Li***, Xiaotong Zhang*, Han Liu*, Quanyu Dai, Xiao-Ming Wu. Dimensionwise Separable 2-D Graph Convolution for Unsupervised and Semi-Supervised Learning on Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD’ 21)*, pages 953–963, 2021.
- Xiaotong Zhang*, Han Liu*, **Qimai Li***, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4327–4333, 2019. **(Cited 140+ times)**
- Lu Fan*, **Qimai Li***, Bo Liu, Xiao-Ming Wu, Xiaotong Zhang, Fuyu Lv, Guli Lin, Sen Li, Taiwei Jin, and Keping Yang. Modeling User Behavior with Graph Convolution for Personalized Product Search. In *Proceedings of the Web Conference 2022*, 2022.

*equal contribution.

Other Publications during My PhD Study

- Xiaotong Zhang, Han Liu, **Qimai Li**, Xiao-Ming Wu. Adaptive Graph Convolution Methods for Attributed Graph Clustering. In submission to *Transactions on Knowledge and Data Engineering*. 2022
- Guangyuan Shi, **Qimai Li**, Wenlong Zhang, Jiaxin Chen, Xiao-Ming Wu. Recon: Reducing Conflicting Gradients From the Root For Multi-Task Learning. In submission to *the Eleventh International Conference on Learning Representations*. 2022.
- Quanyu Dai, Xiao-Ming Wu, Lu Fan, **Qimai Li**, Han Liu, Xiaotong Zhang, Dan Wang, Guli Lin, and Keping Yang. Personalized knowledge-aware recommendation with collaborative and attentive graph convolutional networks. *Pattern Recognition 128 (2022)*: 108628.
- Han Liu, Xiaotong Zhang, Xianchao Zhang, **Qimai Li**, and Xiao-Ming Wu. RPC: Representative possible world based consistent clustering algorithm for uncertain data. *Computer Communications 176 (2021)*: 128-137.
- Jiaxin Chen, Xiao-Ming Wu, Yanke Li, **Qimai Li**, Li-Ming Zhan, and Fu-lai Chung. A closer look at the training strategy for modern meta-learning. *Advances in Neural Information Processing Systems 33 (2020)*: 396-406.
- Guangfeng Yan*, Lu Fan*, **Qimai Li***, Han Liu, Xiaotong Zhang, Xiao-Ming Wu, and Albert YS Lam. Unknown intent detection using Gaussian mixture model with an application to zero-shot intent classification. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 1050-1060. 2020.
- Han Liu, Xiaotong Zhang, Lu Fan, Xuandi Fu, **Qimai Li**, Xiao-Ming Wu, and Albert YS Lam. Reconstructing capsule networks for zero-shot intent classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4799-4809. 2019.
- Yong Wang, Xiao-Ming Wu, **Qimai Li**, Jiatao Gu, Wangmeng Xiang, Lei Zhang, and Victor OK Li. Large margin meta-learning for few-shot classification. In *Workshop on Meta-Learning (MetaLearn 2018)@ NeurIPS 2018*. Neural Information Processing Systems Foundation., 2018.

*equal contribution.

Acknowledgement

First, I would like to express my sincerest gratitude to my supervisor, Prof. Xiao-Ming Wu for her mentorship during my PhD. Her patience, conscientiousness, enthusiasm, and optimism are something I will carry dearly into the rest of my life. Without her guidance and continuous encouragement, this thesis would hardly have been completed.

Second, I would like to thank Prof. Daniel Xiapu Luo, Prof. Fiona Yan Liu, and all other professors in Department of Computing for their guidance, help and support.

Third, I would like to thank all my lab mates for their friendship and support. I feel lucky to spend the past few years with them. Especially, I would like to thank Han Liu, Xiaotong Zhang, Jiaxin Chen, Lu Fan, and Li-Ming Zhan for the wonderful time we shared, from whom I have learned a lot.

Finally, I would also like to thank my family for their continuous and unparalleled love, help, and support. Without them, I would not be where I am today.

Contents

Abstract	i
Publications	iii
Acknowledgement	v
List of Figures	xii
List of Tables	xiv
1 Introduction	1
2 Background and Related Work	7
2.1 Graphs	7
2.1.1 Examples of Real-World Graphs and Learning Tasks	11
2.2 Graph-Based Semi-Supervised Learning	14
2.3 Graph-Based Unsupervised Clustering	17
2.4 Graph Convolutional Networks	20
3 Basics of Graph Signal Processing	23
3.1 Graph Laplacian	24
3.2 Signal Frequency and Fourier Transform	30
3.3 Normalized Laplacian	34
3.3.1 Frequency and Fourier Transform under Normalized Laplacian .	38
3.4 Graph Convolution	39
3.4.1 Graph Filter	39
3.4.2 Graph Convolution	40

3.4.3	Convolution Theorem	42
3.4.4	Unnormalized and Row-Normalized Convolution	45
3.4.5	Efficient Computation of Convolution	46
3.5	Low-pass Filters	47
3.6	Multi-dimensional Graph Signal Processing	51
3.6.1	Cartesian product of Graphs	51
3.6.2	Properties of Kronecker Product	54
3.6.3	Laplacian of Cartesian Product	56
3.6.4	Fourier Basis of Product Graph	58
3.6.5	Signals and Fourier Transform on Product Graph	59
3.6.6	Convolution on Product Graph	61
3.6.7	Multi-Dimensional Graph	63
4	Learning with 1-D Graph Convolution	71
4.1	Generalized Label Propagation	73
4.1.1	Label Propagation	74
4.1.2	Rethink Label Propagation	75
4.2	Revisit and Improve Graph Convolutional Networks	78
4.2.1	Spatial Analysis	79
4.2.2	Spectral Analysis	86
4.2.3	Improved Graph Convolutional Networks	89
4.3	Filter Design and Computation	90
4.4	Adaptive Graph Convolution for Clustering	96
4.4.1	k-Order Random Walk Graph Convolution	97
4.4.2	Clustering via Adaptive Graph Convolution	101

4.4.3	Algorithm Procedure and Time Complexity	103
5	Learning with 2-D Graph Convolutional	107
5.1	2-D Graph Convolution	111
5.1.1	2-D Graph Signal and Spectral Convolution	112
5.1.2	Fast Localized 2-D Graph Convolution	114
5.1.3	Dimensionwise Separable 2-D Graph Convolution (DSGC)	116
5.2	Variance Reduction by DSGC	117
5.2.1	Intra-class Variance Reduction by Object Graph Convolution	118
5.2.2	Intra-class Variance Reduction by Attribute Graph Convolution	121
5.3	Unsupervised and Semi-Supervised Learning with DSGC	125
5.3.1	Learning Frameworks	125
5.3.2	Implementation of Filters	126
6	Empirical Study	129
6.1	Semi-supervised Classification	129
6.1.1	Datasets and Settings	130
6.1.2	Results and Discussion	133
6.2	Unsupervised Clustering	137
6.2.1	Datasets and Settings	137
6.2.2	Result Analysis	139
6.3	2-D Convolution v.s. 1-D Convolution	142
6.3.1	Datasets	142
6.3.2	Variance Reduction and Visualization	144
6.3.3	Semi-supervised Node Classification	146
6.3.4	Node Clustering	149
7	Applications	151

7.1	Zero-Shot Image Classification	151
7.2	Personalized Product Search	154
7.3	Malware Detection and Analysis	159
7.4	Skeleton-based Action Recognition	161
7.4.1	Dynamic Human Skeletons	162
7.4.2	Spatial Temporal Graph Convolutional Networks	163
7.5	Traffic Flow Forecasting	165
8	Conclusion and Future Works	169

List of Figures

1.1	Research accomplishments.	4
2.1	Graph.	9
3.1	An example of graph product. Orange edges come from \mathcal{G}_1 ; blue ones come from \mathcal{G}_2	53
3.2	Mode- n filters and multi-dimensional graph filtering.	66
4.1	Response functions of AP filters.	76
4.2	Visualizing raw and filtered features.	78
4.3	Performance comparison of GCNs, label propagation, and our method for semi-supervised classification on the Cora citation network.	80
4.4	Vertex embeddings of Zachary’s karate club network with GCNs.	83
4.5	Response functions of the renormalization filters (RNM).	87
4.6	Effect of the renormalization trick. Left two figures plot points $(\lambda_i, p(\lambda_i))$. Right two figures plot points $(\tilde{\lambda}_i, p(\tilde{\lambda}_i))$	88
4.7	Response functions of AP, RNM, and RW filters, and their comparison.	94
4.8	Visualization of raw and filtered Cora features (using the RNM filter with different k).	95
4.9	Frequency response	98
4.10	t-SNE visualization of Cora’s raw and filtered node features with different k	102

5.1	t-SNE visualization of the “20 Newsgroups” dataset. (a) Raw features; (b) Filtered by the regular object graph convolution; (c) Filtered by our proposed attribute graph convolution; (d) Filtered by our proposed dimensionwise separable graph convolution (DSGC).	110
5.2	Conceptual illustration of DSGC by a toy example. The node representations (row vectors) obtained by DSGC (\mathbf{GXF}) are better than those in the original feature matrix (\mathbf{X}) or filtered by 1-D graph convolution (\mathbf{GX}), in the sense that nodes of the same class have more similar features.	116
6.1	Visualizing raw and filtered hand-written digits.	136
6.2	$\Delta_{\text{intra}}(k)$ and clustering performance w.r.t. k	140
6.3	t-SNE visualization of the “20 Newsgroups” dataset. (a) Raw features; (b) Filtered by the regular object graph convolution; (c) Filtered by our proposed attribute graph convolution; (d) Filtered by our proposed dimensionwise separable graph convolution (DSGC).	145
6.4	IntraVar/InterVar ratios. The lower, the better.	145
7.1	Spatial connections and temporal connections in a dynamic human skeleton.	163
7.2	Left: traffic flow as a heat map image [6]. Right: traffic flow as a sequence of graphs [7].	167

List of Tables

2.1	Notation Table I.	10
2.2	Example Learning Tasks.	13
3.1	\mathbb{Z}^n , the discretized Euclidean space studied by classical DSP, can be considered as a particular kind of graph — the square grids.	24
3.2	Let $\phi_s = \mathbf{D}^{1/2}\phi_r$, then the following four equations are equivalent. . .	37
3.3	Notation Table II.	50
3.4	N-dimensional discretized Euclidean space. $\mathbb{Z}^n = \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}$. . .	51
3.5	Notation Table III.	68
4.1	GCNs vs. Fully-connected networks	81
6.1	Dataset statistics.	131
6.2	Classification Accuracy with 20 labels per class.	133
6.3	Classification accuracy on citation networks and NELL.	134
6.4	Running time on citation networks and NELL.	135
6.5	Classification Accuracy on MNIST.	136
6.6	Dataset statistics.	138
6.7	Clustering performance.	141
6.8	Dataset statistics.	144
6.9	Classification accuracy on 20 NG, L-Cora, and Wiki.	147

6.10	Classification accuracy on four subsets of WebKB.	148
6.11	Baselines improved by DSGC.	148
6.12	Clustering performance.	150
7.1	Results for unseen classes in AWA2.	153
7.2	Features and relations.	159

Chapter 1

Introduction

Modern machine learning models, especially artificial neural networks, are mainly designed for Euclidean data, i.e., data points represented by n -dimensional vectors, such as audios, images, and videos. In the last decade, deep neural networks, boosted by the growing computational power of GPUs and the availability of a huge amount of training data, have achieved tremendous success in various tasks, including but not limited to understanding and generation of images, videos, and languages.

Other than Euclidean data, graph-structured data is ubiquitous in the real world. Examples include citation networks [8], social networks [9], traffic networks [10], protein-protein interaction networks [11], and knowledge graphs [12, 13]. However, models designed for Euclidean data can hardly be applied to graph-structured data, due to the large differences in data form.

Graph-structured data is different from Euclidean data in nature. First, it is not represented by a high-dimensional vector, but by a set of vertices connected by edges. The edges

are usually stored in the form of a (sparse) adjacency matrix. Second, the common i.i.d. assumption for Euclidean data, which assumes that samples are drawn from independent and identical distributions, does not hold for graph-structured data. Vertex samples of graph-structured data are not independent of each other but strongly correlated, because they are connected by edges. Hence, it is of great interest to design new deep-learning methods for graph-structured data.

Over the past few years, Graph Convolutional Neural Networks (GCNNs) [14–16] have emerged as a new class of promising models that generalize Convolutional Neural Networks (CNNs) from Euclidean space to graph domain and have been successfully applied in a variety of applications including text classification [15], skeleton-based human action recognition [17], traffic flow forecasting [18], zero-shot image recognition [19], and human tissue recognition [11].

The most popular GCNN model, referred to as graph convolutional networks (GCNs), was proposed for semi-supervised node classification by Kipf and Welling [15]. It naturally integrates graph connectivity patterns and node features with a simplified spatial graph convolutional filter and outperforms many state-of-the-art methods on many semi-supervised and unsupervised learning tasks on graphs. Nevertheless, it suffers from similar problems faced by other neural-network-based models. The working mechanisms of the GCN model were not clear, and the training of GCNs still requires a considerable amount of labeled data as a validation set for hyper-parameter tuning and model selection, which is a great disadvantage for semi-supervised learning and unsupervised learning. In addition, like other GCNN models, the training of GCNs usually takes a long time, because the mini-batch training can not be directly applied to graph neural networks. Due

to dense connections between node samples, there is no trivial way to split the training data into mini-batches without losing critical edge information. As such, a full-batch training strategy is often used to train GCNs, which is time-consuming and imposes high requirements on memory capacity, preventing the application of GCNs on large-scale networks and systems.

Moreover, another major limitation of GCNs is that they commonly adopt one-dimensional (1-D) graph convolution that operates on the object link graph to model node (object) relations and features, whose performance critically relies on the quality of the graph. However, real-life networks are often noisy and sparse. For example, in a web graph such as Wikipedia, a hyperlink between two webpages does not necessarily indicate that they belong to the same category, and mixing their features could be harmful for webpage classification or clustering. Moreover, it has been shown that many real-world networks are scale-free and there exist many low-degree nodes [20]. Since these nodes may have very few or even no links to other nodes, it is difficult, or even impossible, to do feature propagation for semi-supervised learning and unsupervised learning.

In this thesis, we aim to address the important issues of GCNs as mentioned above. We conduct a systematic study to analyze GCNs from the perspectives of graph signal processing [21, 22] and spectral graph theory [23, 24]. Our study contributes to the theoretical understanding of the mechanisms of GCNs and gives rise to simplified GCN models with improved efficiency and effectiveness. Our study also inspires various high-impact real-world applications in computer vision, natural language processing, recommender system, system security, and smart transportation. More specifically, we made the following contributions.

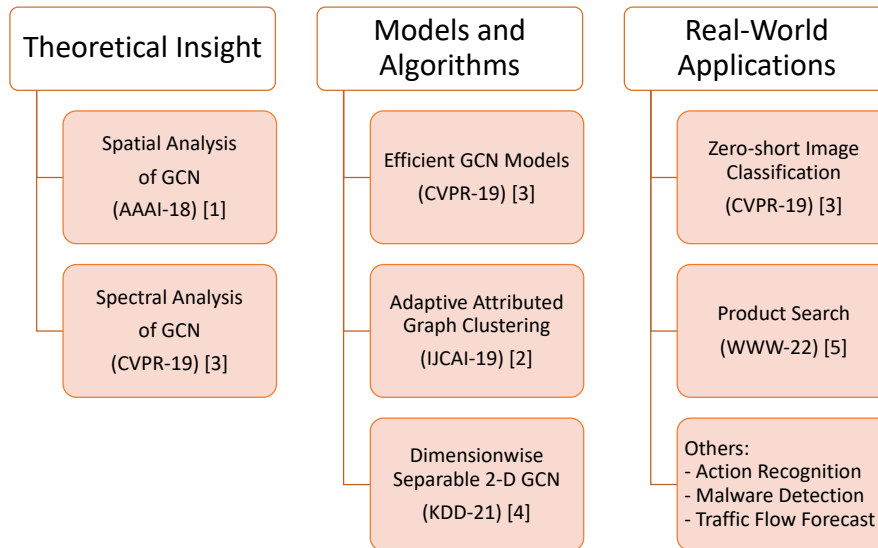


Figure 1.1: Research accomplishments.

Contribution 1: Theoretical insights of GCNs. We analyze GCNs from both spatial and spectral perspectives and reveal the fundamental mechanisms. Our spatial analysis [1] shows that the graph convolutional filter of GCNs is actually a special form of Laplacian smoothing, which is the key reason why GCN works, but it also brings up the over-smoothing problem of deep GCN models. Our spectral analysis [3] revisits GCN and classical label propagation methods under a graph filtering framework and shows that what they actually do is extracting useful data representations by a low-pass graph filter. The findings were published in AAAI-18 [1] and CVPR-19 [3].

Contribution 2: New efficient GCN models and learning algorithms. With the new theoretical insights, we have developed new, efficient, and more powerful models with 1-D and 2-D graph convolutional filters for unsupervised and semi-supervised learning, including Improved Graph Convolutional Networks (IGCN) [3], Generalized Label Propagation (GLP) [3], Adaptive Graph Convolution (AGC) [2] and Dimensionwise Separable 2-D Graph Convolution (DSGC) [4]. The effectiveness of our proposed models

and algorithms is verified by extensive experiments on semi-supervised and unsupervised learning tasks on graphs. The results were published in IJCAI-19 [2], CVPR-19 [3], and KDD-21 [4].

Contribution 3: Real-world applications. We have successfully applied our proposed models and algorithms to various real-world applications including zero-shot image recognition and personalized product search. We also discuss potential applications in skeleton-based action recognition, traffic flow forecasting and malware detection. Our models demonstrate higher accuracy than baselines in zero-shot image classification. Our method significantly improves the performance of state-of-the-art models for personalized product search. The results were published in CVPR-19 [3] and WWW-22 [5].

Thesis organization. Chapter 2 introduces the background of graph learning, defines the problem of graph-based semi-supervised and unsupervised learning, and provides an overview of existing literature. Chapter 3 gives a basic yet comprehensive introduction to graph signal processing, which serves as the main theoretical tool used in the following chapters. Chapter 4 revisits existing 1-D graph-convolution-based methods under the graph signal processing framework, reveals their mechanisms and drawbacks, and proposes new models for semi-supervised and unsupervised learning including IGCN, GLP, and AGC. Chapter 5 explores 2-D graph convolution to jointly model object links and attribute relations for graph representation learning and proposes computationally efficient dimensionwise separable 2-D graph convolution (DSGC). Chapter 6 presents comprehensive experiments to validate the theoretical analysis and proposed methods in Chapters 4 and 5. Chapter 7 discusses several real-world applications of graph convolution and presents exemplary and potential solutions.

Chapter 2

Background and Related Work

2.1 Graphs

Graphs are data structures widely used to model pairwise relations between objects. A graph is made up of vertices, which are connected by edges. In the machine learning context, every vertex is usually associated with a feature vector describing it.

Definition 1 (Graph). *A graph \mathcal{G} is an ordered triple*

$$\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X}), \tag{2.1}$$

comprising of a vertex set $\mathcal{V} = \{\nu_1, \dots, \nu_n\}$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, and an optional feature matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$.

An example of a graph is shown in Fig. 2.1. This definition differs from the common definition of a weighted graph in two ways.

First, the edge set \mathcal{E} and weight function are replaced by a single adjacency matrix \mathbf{A} ,

which encodes the edge weight between vertices. Sometimes, \mathbf{A} is also referred to as the topological structure of the graph. By convention, non-zero a_{ij} indicates a connection between ν_i and ν_j while zero value indicates no connection, and hence the edge set can be defined by $\mathcal{E} = \{(\nu_i, \nu_j) | a_{ij} \neq 0\}$. We consider graphs with non-negative weights ($a_{ij} \geq 0$), unless specified otherwise.

Second, every vertex represents an object (e.g., a document or an entity), which is usually described by a feature vector. The feature matrix \mathbf{X} is of size $n \times m$, where n is the number of vertices and m is the dimension of the feature vectors:

$$\mathbf{X} = \begin{bmatrix} \text{---} & \text{---} & \mathbf{x}_1^\top & \text{---} & \text{---} \\ \text{---} & \text{---} & \mathbf{x}_2^\top & \text{---} & \text{---} \\ & & \vdots & & \\ \text{---} & \text{---} & \mathbf{x}_n^\top & \text{---} & \text{---} \end{bmatrix}. \quad (2.2)$$

\mathbf{X} contains n rows. The i -th row is the feature vector associated with vertex ν_i . With \mathbf{X} given, \mathcal{G} is also referred to as an attributed graph, and \mathbf{X} is also referred to as attributes. If there are no given features, \mathbf{X} can be represented by an identity matrix \mathbf{I} or a zero matrix $\mathbf{0}$.

According to whether edges are weighted or not, graphs can be categorized into two types: weighted and unweighted. For unweighted graphs, all edge weights are 1, and the adjacency matrix is binary where 0/1 indicates the absence/presence of edges.

Definition 2 (Weighted and Unweighted Graph). *If the adjacency matrix \mathbf{A} is binary, i.e.,*

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \quad (2.3)$$

then the graph is unweighted because all edges have equal weight. Otherwise, the graph

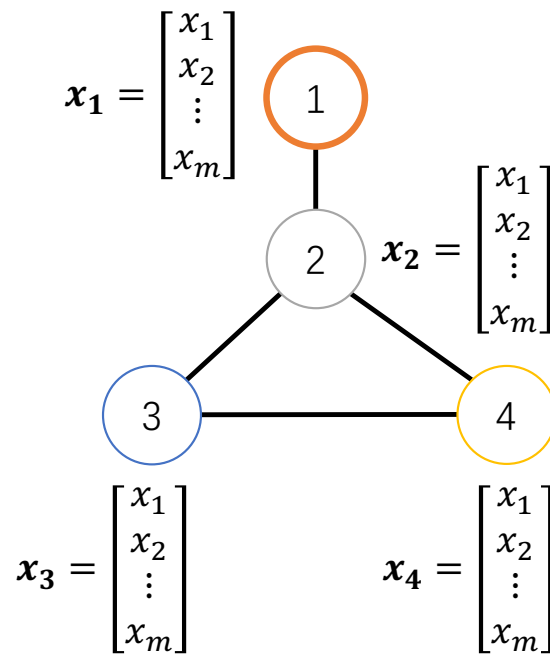


Figure 2.1: Graph.

is a weighted graph.

According to whether edges have direction or not, graphs can be categorized into two types: directed and undirected. If there is no direction associated with edges, we call it an undirected graph. In such a case, the existence of edge (ν_i, ν_j) indicates the existence of edge (ν_j, ν_i) , and $a_{ij} = a_{ji}$, which indicates the adjacency matrix is symmetric.

Definition 3 (Directed and Undirected Graph). *Graph \mathcal{G} is an undirected graph, if and only if the adjacency matrix is symmetric, i.e., $\mathbf{A} = \mathbf{A}^\top$. Otherwise, the graph is a directed graph.*

To turn a directed graph into an undirected graph, one can simply let $\mathbf{A} := (\mathbf{A} + \mathbf{A}^\top)/2$.

Table 2.1: Notation Table I.

Notation	Description
Bold upper-case letters	Matrices, such as \mathbf{A} , \mathbf{X} , \mathbf{Y}
Bold lower-case letters	Vectors, such as \mathbf{x} , \mathbf{z}
Plain lower-case letters	Scalars, matrix/vector elements, such as n , m , a_{ij}
\mathbf{A}^\top	Transpose of \mathbf{A}
$\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$	Graph
$\mathcal{V} = \{\nu_1, \dots, \nu_n\}$	Vertex set
$\mathcal{E} = \{(\nu_i, \nu_j) a_{ij} \neq 0\}$	Edge set
(ν_i, ν_j)	Edge from ν_i to ν_j
$\mathbf{A} = \{a_{ij}\} \in \mathbb{R}_+^{n \times n}$	Adjacency matrix
$a_{ij} \geq 0$	Weight between ν_i and ν_j
$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times m}$	Feature matrix
$\mathbf{x}_i \in \mathbb{R}^m$	Feature vector of ν_i
$n = \mathcal{V} $	Number of vertices
m	Number of feature dimension
$\mathbf{Y} \in \mathbb{R}^{n \times c}$	Label matrix (one-hot)
c	Number of classes
$\mathcal{L} = \{\nu_i\}_{i=0}^l$	Labeled vertex set
$\mathcal{U} = \{\nu_i\}_{i=l+1}^n$	Unlabeled vertex set

2.1.1 Examples of Real-World Graphs and Learning Tasks

Learning tasks on graphs can be divided into three levels: vertex-level, edge-level, and graph-level. According to whether the learning outcome is continuous or discrete, and whether the training procedure involves labels or not, we can divide the learning tasks into three categories: classification, regression, and clustering. Table 2.2 gives several examples for each category, which are described below in detail.

Citation networks [8, 25–27] are networks that record documents’ citation relationships. In citation networks, vertices are documents. A pair of vertices are connected by an edge if and only if one cites another. Besides the citation information between documents, each document is also associated with a feature vector, which encodes the document content. Commonly used features are bag-of-words, TF-IDF [28], and text embeddings [29, 30]. At the vertex level, we can perform topic analysis to classify documents into different topics. At the edge level, we can suggest for a document the papers which it should have cited but did not. If we include authors in the graph, we can also suggest reviewers for an academic paper.

Social networks [9] are networks that record social relationships. In social networks, vertices are people. A pair of vertices are connected by an edge if one person follows another. Besides the friendship information between users, each person is also associated with a feature vector extracted from their social profiles, which encodes the user’s interests, background, and demographic information. The most common task is to predict whether a user follows another, and recommend people to follow for users.

Traffic networks [10] are networks that record traffic patterns. In traffic networks, ver-

tices are locations, such as road intersections, stations, airports, harbors, cities, villages, and so on. These locations are connected by roads, railways, airways, and waterways. Each location also has its attributes, such as traffic volume, traffic density, and other information. The most common task is to predict the traffic flow at vertices, given historical traffic data.

Protein-protein interaction networks [11] are networks that record protein interactions. In protein networks, a pair of protein molecules are connected by an edge if they can react with each other to produce some new molecules. We classify protein roles, which rely on their cellular functions, and each graph corresponds to a specific human tissue.

Knowledge graphs [12, 13, 31] are heterogeneous information networks, which contain multiple types of entities, such as people, organizations, locations, movies, books, tools, vehicles, and countries. They also contain multiple types of relationships, such as “person A is a friend of person B”, “location A is a part of location B”, “actor A stars in movie B”, “author A writes book B”, or “company A produces product B”. Such heterogeneous relationships are referred to as facts or knowledge. Given a knowledge graph, we can obtain new knowledge by predicting whether there exists a certain type of relation between two entities.

3D mesh [35, 37] is a set of vertices, edges, and faces that represents the shape of a 3D object. The faces are usually triangles or other simple convex polygons. Each vertex is associated with a vector, describing its position, color, or other information. The possible learning tasks with 3D meshes are 3D object recognition, detection, segmentation, and retrieval.

Malware [36] is software with malicious purposes, such as privacy collection, remote

Table 2.2: Example Learning Tasks.

Vertex level	Classification	Document topic analysis [8] Sensor anomaly detection Protein role recognition [11]
	Regression	Traffic flow prediction [10] Sensor value forecast
	Clustering	Community detection[32] 3D Object segmentation
Edge level	Link prediction	Friend recommendation [33] Product search [5] Knowledge reasoning [34]
	Regression	Chemical bond energy prediction
Graph level	Classification	3D object classification [35] Malware detection [36]
	Clustering	3D content retrieval [37]

control, session hijacking, and remote monitoring. The development of code obfuscation techniques enables malware to adopt various camouflages and makes them harder to be detected and analyzed directly from the code level. However, malware can also be represented as various graphs with different abstraction levels, including control-flow graphs, data-flow graphs, and function call graphs, which are robust against code obfuscation. The tasks of interest are to detect malware and analyze the properties of different malware families.

In general, many kinds of relationships can be represented as graphs, and graph-structured data are omnipresent. They differ in field, form, scale, and nature. Some data come from computer science, physics, chemistry, or biology, while some are from social science,

finance, or economy. Some are text data, and some are images or videos. Small-scale data contain only dozens of nodes, while large-scale ones may contain millions or trillions of nodes. The prevalence of wide variety of graph-structured data makes learning on graphs an important research area.

2.2 Graph-Based Semi-Supervised Learning

It is well known that training a deep neural model typically requires a large amount of labeled data, which cannot be satisfied in many scenarios due to the high cost of labeling training data. To reduce the amount of data needed for training, we can adopt a semi-supervised learning paradigm, where a large amount of *unlabeled* data is utilized to boost training with typically a small amount of labeled data.

The problem of semi-supervised classification is defined as follows. We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$, where \mathcal{V} is the vertex set with n vertices, $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}_+^{n \times n}$ is the adjacency/affinity matrix, and $\mathbf{X} \in \mathbb{R}^{n \times m}$ is the feature matrix. The adjacency matrix \mathbf{A} encodes edge weights, and is non-negative and symmetric, with $a_{ij} = a_{ji} \geq 0$. The feature matrix \mathbf{X} contains feature vectors of vertices, where the i -th row is a m -dimensional feature vector of vertex ν_i . Vertices fall into one of the c classes. The vertex set are partitioned into two parts, a labeled set $\mathcal{L} = \{\nu_1, \dots, \nu_l\}$ and an unlabeled set $\mathcal{U} = \{\nu_{l+1}, \dots, \nu_n\}$ with $|\mathcal{L}| = l$, $|\mathcal{U}| = n - l$. In semi-supervised classification, only the labels of a small subset of vertices are given ($l \ll n$), and the goal is to predict the labels of the rest of the vertices.

Much research has shown that if properly used, unlabeled data can significantly improve

learning performance [38]. The critical issue is maximizing the effective utilization of structural and feature information in unlabeled data. In the past two decades, semi-supervised learning has been dominated by graph-based methods, among which the most widely studied one is label propagation. In recent years, due to the powerful feature extraction capability and the recent success of deep neural networks, there have been some successful attempts to revisit semi-supervised learning with neural-network-based models, including ladder network [39], semi-supervised embedding [40], planetoid [8], and graph convolutional networks [15].

In graph-based semi-supervised learning, the cluster assumption, which states that nearby vertices are likely to have the same labels, has been widely adopted in many methods, either explicitly or implicitly. Under the cluster assumption, the adjacency matrix A is also referred to as the affinity/similarity matrix, which encodes the similarity between vertices. Based on this assumption, there are several ways to carry out semi-supervised learning. One idea is to learn smooth low-dimensional embedding of data points by using Markov random walks [41], Laplacian eigenmaps [42], and spectral kernels [43, 44]. Another idea hinges on graph partition, where the cuts should agree with the labeled data and be placed in low-density regions [45–48]. Perhaps the most popular idea is to formulate a quadratic regularization framework to explicitly capture the consistency with the labeled data and the cluster assumption, which is known as label propagation [49–52]. Other methods include modified adsorption [53] and an iterative classification algorithm [25]. It was shown in Ekambaram et al. [54] and Girault et al. [55] that graph regularization in many of these methods can be interpreted as low-pass graph filters.

However, these methods are limited in their ability to incorporate vertex features for

prediction. In many applications, data instances come with feature vectors containing information not present in the graph. For example, in a citation network, the citation links between documents describe their citation relations, while the documents are represented by bag-of-words vectors which describe their contents. Many semi-supervised learning methods seek to jointly model the graph structure and feature attributes of data. The common idea is to regularize a supervised learner (e.g., support vector machines, neural networks) with some regularizer. For example, manifold regularization (LapSVM) [56] regularizes a support vector machine with a Laplacian regularizer. Deep semi-supervised embedding [40] regularizes a deep neural network with an embedding-based regularizer. Planetoid [8] also regularizes a neural network by jointly predicting the class label and the context of an instance.

Inspired by the success of convolutional neural networks (CNN) on grid-structured data such as images and videos, a series of works proposed a variety of graph convolutional neural networks [57–60] to extend CNN to general graph-structured data. These recently proposed graph convolutional neural networks adopt a different way to integrate graph topology and feature information, which is graph convolution, and have dominated modern graph-based semi-supervised learning. In their pilot work, ChebyNet [14] proposed to use a polynomial filter represented by k -th order polynomials of graph Laplacian via Chebyshev expansion to avoid the expensive eigen-decomposition. Graph convolutional networks (GCN) [15] further simplified ChebyNet by using a localized first-order approximation of spectral graph convolution and achieved promising results in semi-supervised learning. The GCN model naturally integrates the connectivity patterns and feature attributes of graph-structured data and outperforms many state-of-the-art methods significantly on some benchmarks. Other related GCNNs include

GraphSAGE [11], graph attention networks [16], FastGCN [61], dual graph convolutional neural network [62], stochastic GCN [63], Bayesian GCN [64], deep graph infomax [65], LanczosNet [66], etc. We refer readers to two comprehensive surveys [67, 68] for more discussions.

Despite the promising performance of GCN, it suffers from similar problems faced by other neural-network-based models. The working mechanisms of the GCN model for semi-supervised learning were unclear, and the training of GCNs still requires a considerable amount of labeled data for parameter tuning and model selection, which defeats the purpose of semi-supervised learning. Our papers [1, 3] address these drawbacks of GCN. First, we reveal its working mechanisms by showing that the success of GCN is due to performing Laplacian smoothing on data features, which is a smoothing filter in the spatial domain and a low-pass filter in the spectral domain. Second, guided by our theoretical analysis, we proposed a two-step framework that greatly improves accuracy and training speed.

2.3 Graph-Based Unsupervised Clustering

Attributed graph clustering [69] aims to cluster nodes of an attributed graph where each node is associated with a set of feature attributes. Attributed graphs widely exist in real-world applications, and clustering plays a vital role in detecting communities and analyzing the structure of these networks. However, attributed graph clustering requires joint modeling of graph structures and node attributes to make full use of available data, which presents significant challenges.

Some classical clustering methods, such as k -means, only deal with data features. In contrast, many graph-based clustering methods [70] only leverage graph connectivity patterns, e.g., user friendships in social networks, paper citation links in citation networks. Typically, these methods learn node embeddings using Laplacian eigenmaps, matrix factorization, random walks, or autoencoder. Methods based on graph Laplacian eigenmaps [71] assume that nodes with higher similarity should be mapped closer. Methods based on matrix factorization [72, 73] factorize the node adjacency matrix into node embeddings. Methods based on random walks [74, 75] learn node embeddings by maximizing the probability of the neighborhood of each node. Autoencoder-based methods [9, 76, 77] find low-dimensional node embeddings with the node adjacency matrix and then use the embeddings to reconstruct the adjacency matrix. Nevertheless, they usually fall short in attributed graph clustering, as they do not exploit informative node features such as user profiles in social networks and document contents in citation networks.

In recent years, various attributed graph clustering methods have been proposed, including methods based on generative models [78], spectral clustering [79], random walks [80], nonnegative matrix factorization [81], and graph convolutional networks (GCN) [15]. Attributed graph clustering [82] takes both node connectivity and features into account and thus differs from topology-only methods. Some model the interaction between graph connectivity and node features with generative models [78, 83–85]. Some apply nonnegative matrix factorization or spectral clustering to both the underlying graph and node features to get a consistent cluster partition [79–81, 86]. Some recent methods integrate node relations and features using GCN [15]. In particular, graph autoencoder (GAE) and graph variational autoencoder (VGAE) [87] learn node representations with a two-layer

GCN and then reconstruct the node adjacency matrix with autoencoder and variational autoencoder respectively. Marginalized graph autoencoder (MGAE) [88] learns node representations with a three-layer GCN and then applies marginalized denoising autoencoder to reconstruct the given node features. Adversarially regularized graph autoencoder (ARGE) and adversarially regularized variational graph autoencoder (ARVGE) [89] learn node embeddings by GAE and VGAE, respectively, and then use generative adversarial networks to enforce the node embeddings to match a prior distribution. In particular, GCN based methods such as GAE [87], MGAE [88], ARGE [89] have demonstrated state-of-the-art performance on several attributed graph clustering tasks.

Although graph convolution has been shown very effective in integrating structural and feature information, there is little study of how it should be applied to maximize clustering performance. Most existing methods directly use GCN as a feature extractor, where each convolutional layer is coupled with a projection layer, making it difficult to stack many layers and train a deep model. In fact, ARGE [89] and MGAE [88] use a shallow two-layer and three-layer GCN, respectively, in their models, which only take into account neighbors of each node in two or three hops away and hence may be inadequate to capture global cluster structures of large graphs. Moreover, all these methods use a fixed model and ignore the diversity of real-world graphs, which can lead to suboptimal performance.

To address these issues, we propose an adaptive graph convolution (AGC) method for attributed graph clustering [2]. The intuition is that nearby nodes tend to be in the same cluster, and clustering will become much easier if nearby nodes have similar feature representations. To this end, instead of stacking many layers as in GCN, we design a k -order graph convolution that acts as a low-pass graph filter on node features to obtain

smooth feature representations, where k can be adaptively selected using intra-cluster distance. AGC consists of two steps: 1) conducting k -order graph convolution to obtain smooth feature representations; 2) performing spectral clustering on the learned features to cluster the nodes. AGC enables easy use of high-order graph convolution to capture global cluster structures and allows the selection of an appropriate k for different graphs.

2.4 Graph Convolutional Networks

Graph convolutional neural networks (GCNNs) generalize traditional convolutional neural networks to the graph domain. There are mainly two types of GCNNs [90]: spatial GCNNs and spectral GCNNs. Spatial GCNNs view the convolution as a “patch operator” which constructs a new feature vector for each vertex using its neighborhood information. Spectral GCNNs define the convolution by decomposing a graph signal $\mathbf{x} \in \mathbb{R}^n$ (a scalar for each vertex) on the spectral domain and then applying a spectral filter g_θ (a function mapping eigenvalues of Laplacian \mathbf{L}_s) on the spectral components [21, 22, 91]. However, this model requires explicitly computing the Laplacian eigenvectors, which is impractical for real large graphs. A way to circumvent this problem is by approximating the spectral filter g_θ with Chebyshev polynomials up to K -th order [92]. Defferrard et al. [14] applied this to build a K -localized ChebyNet, where the convolutional filter is defined as a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K -th order:

$$g_{\theta'}(\mathbf{\Lambda}) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\mathbf{\Lambda}}), \quad (2.4)$$

where $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}} \mathbf{\Lambda} - \mathbf{I}$ are rescaled eigenvalues, and $\theta' \in \mathbb{R}^K$ is a vector of Chebyshev coefficients. By the approximation, the ChebyNet is actually spectrum-free and can be

evaluated in the spatial domain.

$$g_{\theta'}(\mathbf{L}_s) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\mathbf{L}}_s), \quad (2.5)$$

where $\tilde{\mathbf{L}}_s = \frac{2}{\lambda_{\max}} \mathbf{L}_s - \mathbf{I}$.

Kipf and Welling [15] simplified this model by limiting $K = 1$ and approximating the largest eigenvalue λ_{\max} of \mathbf{L}_s by 2. In this way, the filter becomes

$$g_{\theta'}(\mathbf{L}_s) = \theta'_0 \mathbf{I} + \theta'_1 (\mathbf{L}_s - \mathbf{I}) = \theta'_0 \mathbf{I} - \theta'_1 \mathbf{A}_s. \quad (2.6)$$

Furthermore, they restrict $\theta'_0 = -\theta'_1 \triangleq \theta$, where θ is the only Chebyshev coefficient left.

$$g_{\theta'}(\mathbf{L}_s) = \theta(\mathbf{I} + \mathbf{A}_s). \quad (2.7)$$

They further applied a renormalization trick to the graph, which is to add a self-loop to each vertex and replace $(\mathbf{I} + \mathbf{A}_s)$ by $\tilde{\mathbf{A}}_s$:

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}, \quad (2.8)$$

$$\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}, \quad (2.9)$$

$$\tilde{\mathbf{A}}_s = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}, \quad (2.10)$$

$$\mathbf{I} + \mathbf{A}_s \rightarrow \tilde{\mathbf{A}}_s. \quad (2.11)$$

Generalizing the above definition of convolution to a graph signal with m input channels, i.e., $\mathbf{X} \in \mathbb{R}^{n \times m}$ (each vertex is associated with an m -dimensional feature vector), Kipf and Welling [15] defined graph convolutional layer. The layer-wise propagation rule of this simplified model is:

$$\mathbf{H}^{(l+1)} = \sigma \left(\tilde{\mathbf{A}}_s \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right), \quad (2.12)$$

where $\mathbf{H}^{(l)}$ is the activations in the l -th layer with $\mathbf{H}^{(0)} = \mathbf{X}$, and $\mathbf{W}^{(l)}$ is the trainable weight matrix in l -th layer, σ is the activation function, e.g., $\text{ReLU}(\cdot)$. This simplified model is called graph convolutional networks (GCNs).

Chapter 3

Basics of Graph Signal Processing

As graph convolution becomes an essential component of GNNs, there is a foundational mathematical branch behind it — graph signal processing. The emerging field of signal processing on graphs [21, 22] merges the concepts in algebraic graph theory and spectral graph theory [23] with harmonic analysis. Graph signal processing generalizes classical harmonic analysis concepts, including signals, filters, convolution, convolution theorem, and sampling theorem from Euclidean space to graphs and develops a complete theory for analyzing and manipulating discrete graph signals. It serves as the primary theoretical tool for this thesis and will be briefly introduced in this chapter.

Classic harmonic analysis studies Euclidean signals, such as voice, images, and videos. All these signals are continuous by nature, but computers can only deal with discrete data, so they need to be sampled and discretized before being processed by computers. The branch of harmonic analysis that studies such discrete signals is discrete signal processing (DSP). In classic DSP, signals are functions defined on discretized Euclidean spaces, a.k.a., multidimensional integer space \mathbb{Z}^n . However, these signals can also be

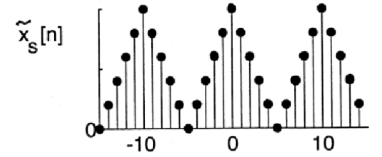
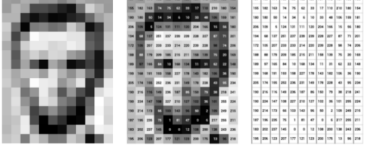
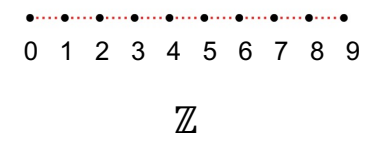
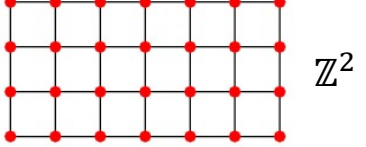
	1D	2D
Euclidean Signals		
Space structure as a Graph		

Table 3.1: \mathbb{Z}^n , the discretized Euclidean space studied by classical DSP, can be considered as a particular kind of graph — the square grids.

viewed as defined on a particular kind of graph — the square grids. For example, audios, images, and videos are signals defined on 1-D, 2-D, and 3-D grids, respectively. Audios are defined on integer set \mathbb{Z} . In space \mathbb{Z} , each integer is adjacent to two neighbors, its successor, and its predecessor, so the whole space has a chain-like structure as Table 3.1 shows. As typical 2D signals, images are defined on 2-D integer space \mathbb{Z}^2 . An image consists of many pixels arranged in rows and columns. Each pixel is adjacent to its four neighbors, the top, bottom, left, and right ones, so the space \mathbb{Z}^2 has a grid-like structure. More generally, the structure of multidimensional integer space \mathbb{Z}^n , from the viewpoint of graphs, can be considered as an n -dimensional square grid, hence the classic DSP theory is actually a special case of graph signal processing.

3.1 Graph Laplacian

Graph Signals Processing extends the concept of signal from regular grids to general graphs. Given a graph $\mathcal{G} = (\mathcal{V}, \mathbf{A})$ with vertex set \mathcal{V} and adjacency matrix \mathbf{A} , denote the number of vertices by $n = |\mathcal{V}|$. We only consider positive-weighted undirected graphs,

i.e., $\mathbf{A} = \mathbf{A}^\top$ and $a_{ij} \geq 0$ for all $i, j \leq n$.

Definition 4 (Graph signal). *A graph signal on \mathcal{G} is a real-valued function $x : \mathcal{V} \rightarrow \mathbb{R}$ that associates each vertex ν_i with a real value $x(\nu_i)$. All the associated real values are usually arranged into a vector*

$$\mathbf{x} = [x_1, \dots, x_n]^\top \in \mathbb{R}^n \quad (3.1)$$

with $x_i = x(\nu_i)$.

Graph signals are ubiquitous in graph-structured data. For example, every column of the feature matrix \mathbf{X} in Section 2.1 is a graph signal. If we take an image as a 2-D grid, then its RGB channels are three different signals on the 2-D grid. Given a sensor network, all sensor values form a graph signal. Given a social network, the ages of all people in the network form a graph signal.

In classic harmonic analysis, signals are decomposed into a linear combination of Fourier basis with different frequencies. In graph signal processing, eigenvectors and eigenvalues of the graph Laplacian play the same role of Fourier basis and frequencies in parallel with classical harmonic analysis. This section introduces graph Laplacian and its basic properties. For a more detailed discussion of graph Laplacian, we refer the readers to Von Luxburg [24] and Chung [23].

The ordinary Laplace operator, or simply Laplacian, is a differential operator on scalar fields that measures the divergence of its gradient. Graph Laplacian does the same thing.

Definition 5 (Gradient). *Given a signal \mathbf{x} , its gradient along a directed edge $\nu_i \rightarrow \nu_j$ is defined as*

$$(\nabla \mathbf{x})_{ij} = a_{ij}(x_i - x_j).$$

All gradients together form a gradient matrix $\nabla x \in \mathbb{R}^{n \times n}$.

The above definition of gradient also applies to undirected graphs, by taking each undirected edge $\{\nu_i, \nu_j\}$ as two directed edges $\nu_i \rightarrow \nu_j$ and $\nu_j \rightarrow \nu_i$. Note that, if there is no edge from ν_i to ν_j , then a_{ij} is 0 and so is gradient $(\nabla x)_{ij}$. Laplacian L computes the divergence of the gradient at each vertex. Let $\mathbf{y} = L\mathbf{x}$, then $\mathbf{y} \in \mathbb{R}^n$ and

$$y_i = \sum_j a_{ij}(x_i - x_j) \quad (3.2)$$

$$= x_i \sum_j a_{ij} - \sum_j a_{ij}x_j \quad (3.3)$$

$$= d_i x_i - \sum_j a_{ij}x_j. \quad (3.4)$$

where $d_i = \sum_j a_{ij}$ is the degree of vertex ν_i . We can arrange the degrees of all vertices into a diagonal matrix D with the degrees d_1, \dots, d_n on its diagonal.

$$D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \quad (3.5)$$

D is referred to as the degree matrix. It is easy to verify that $L\mathbf{x} = (D - A)\mathbf{x}$, so applying L to \mathbf{x} is equivalent to multiplying \mathbf{x} by a matrix $D - A$, and the matrix $D - A$ is the so-called Laplacian matrix.

Definition 6 (Laplacian matrix). *The unnormalized graph Laplacian matrix is defined as*

$$L = D - A. \quad (3.6)$$

Theorem 1 (Spectrum of unnormalized Laplacian). *If the graph is positively weighted and undirected, i.e., $A \geq 0$ and $A = A^\top$, then the graph Laplacian matrix L satisfies the following properties:*

1. For every signal $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{x}^\top \mathbf{L} \mathbf{x} = \frac{1}{2} \sum_{ij} a_{ij} (x_i - x_j)^2. \quad (3.7)$$

2. \mathbf{L} has non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. The smallest eigenvalue 0 has an all-one eigenvector $\vec{\mathbf{1}}$.

3. If the graph is connected, then the multiplicity of eigenvalue 0 is equal to 1, i.e., $\lambda_2 > 0$.

4. The largest eigenvalue $\lambda_n \leq 2d_m$ where $d_m = \max_i d_i$ is the largest vertex degree.

5. The largest eigenvalue $\lambda_n = 2d_m$ if and only if the graph is both regular and bipartite.

Proof.

1. We first prove Eq. (3.7):

$$\begin{aligned} \frac{1}{2} \sum_{ij} a_{ij} (x_i - x_j)^2 &= \frac{1}{2} \sum_{ij} a_{ij} x_i^2 + \frac{1}{2} \sum_{ij} a_{ij} x_j^2 - \sum_{ij} a_{ij} x_i x_j \\ &= \frac{1}{2} \sum_i d_i x_i^2 + \frac{1}{2} \sum_j d_j x_j^2 - \mathbf{x}^\top \mathbf{A} \mathbf{x} \\ &= \sum_i d_i x_i^2 - \mathbf{x}^\top \mathbf{A} \mathbf{x} \\ &= \mathbf{x}^\top \mathbf{D} \mathbf{x} - \mathbf{x}^\top \mathbf{A} \mathbf{x} = \mathbf{x}^\top \mathbf{L} \mathbf{x}. \end{aligned}$$

2. As a direct consequence of Eq. (3.7), \mathbf{L} is symmetric and positive semi-definite, so all its eigenvalues λ_i are non-negative. It is easy to verify that $\mathbf{L} \vec{\mathbf{1}} = \vec{\mathbf{0}}$, so the smallest eigenvalue $\lambda_1 = 0$, and all-one vector $\vec{\mathbf{1}}$ is the corresponding eigenvector.

3. If the graph is connected, then $\mathbf{L}\mathbf{x} = \vec{\mathbf{0}}$ implies $x_i = x_j$ for all i, j , so $\mathbf{x} = \alpha\vec{\mathbf{1}}$ for some $\alpha \in \mathbb{R}$. The dimension of the eigenspace of eigenvalue 0 is 1, and so is its multiplicity.

4. $\mathbf{x}^\top \mathbf{L}\mathbf{x}$ has the following upper bound.

$$\begin{aligned} \mathbf{x}^\top \mathbf{L}\mathbf{x} &= \frac{1}{2} \sum_{ij} a_{ij} (x_i - x_j)^2 \\ &\leq \sum_{ij} a_{ij} (x_i^2 + x_j^2) \\ &= \sum_i d_i x_i^2 + \sum_j d_j x_j^2 \\ &= 2 \sum_i d_i x_i^2. \end{aligned} \tag{3.8}$$

By the property of the largest eigenvalue and inequality (3.8), we have

$$\lambda_n = \max_{\mathbf{x}} \frac{\mathbf{x}^\top \mathbf{L}\mathbf{x}}{\mathbf{x}^\top \mathbf{x}} \tag{3.9}$$

$$\leq 2 \frac{\sum_i d_i x_i^2}{\|\mathbf{x}\|_2^2} \tag{3.10}$$

$$\leq 2d_m. \tag{3.11}$$

So, Property 4 is proved.

5. The equalities in (3.8) and (3.10) hold if and only if $x_i = -x_j$ for all connected vertex pairs (ν_i, ν_j) , which indicates the graph is bipartite. In this case, $\lambda_n = \frac{2}{n} \sum_i d_i$. If the equality in (3.11) also holds, then all nodes must have the same degree, which requires the graph to be regular. In summary, eigenvalues of \mathbf{L} are bounded by $2d_m$, and the maximum $2d_m$ is taken if and only if the graph is both regular and bipartite. \square

We can also derive a range for eigenvalues of \mathbf{A} from that of \mathbf{L} . We know that $\mathbf{A} = \mathbf{D} - \mathbf{L}$,

eigenvalues of \mathbf{D} fall in $[0, d_m]$, and eigenvalues of \mathbf{L} fall in $[0, 2d_m]$, so eigenvalues of the adjacency matrix \mathbf{A} fall in $[-d_m, d_m]$.

Disconnected Graph Another interesting fact about graph Laplacian is that the multiplicity of eigenvalues 0 is the number of connected components in the graph. Assume the graph contains k connected components $\mathcal{C}_1, \dots, \mathcal{C}_k$ with n_1, \dots, n_k vertices respectively. Without loss of generality, we assume vertices of every component are numbered consecutively, i.e., the first n_1 vertices belong to \mathcal{C}_1 , and the next n_2 vertices belong to \mathcal{C}_2 , and so on. Then the graph Laplacian matrix \mathbf{L} and the adjacency matrix \mathbf{A} is in block diagonal form:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & & \\ & \mathbf{L}_2 & & \\ & & \ddots & \\ & & & \mathbf{L}_k \end{bmatrix} \quad (3.12)$$

Every block is a graph Laplacian matrix of a connected component, and thus it has an eigenvalue 0 with multiplicity 1. Eigenvalues of sub-blocks are also eigenvalues of the whole graph, and the multiplicity equals the sum of multiplicity in sub-blocks. Eigenvalue 0 of the whole Laplacian \mathbf{L} has a multiplicity of k , equal to the number of connected components. The corresponding eigenspace is spanned by $\{\vec{\mathbf{1}}_{\mathcal{C}_1}, \dots, \vec{\mathbf{1}}_{\mathcal{C}_k}\}$, where $\vec{\mathbf{1}}_{\mathcal{C}_j}$ is indicator vector of component \mathcal{C}_j . If $\mathbf{x}_i \in \mathcal{C}_j$, then i -th component of $\vec{\mathbf{1}}_{\mathcal{C}_j}$ is 1, otherwise 0. These properties enable us to focus only on connected graphs because a disconnected graph can be easily divided into several connected components and discussed separately.

3.2 Signal Frequency and Fourier Transform

We rewrite Eq. (3.7) here.

$$\mathbf{x}^\top \mathbf{L} \mathbf{x} = \frac{1}{2} \sum_{ij} a_{ij} (x_i - x_j)^2. \quad (3.7)$$

This equation actually only sums over connected vertex pairs, as any unconnected vertex pair has 0 weight, i.e., $a_{ij} = 0$. Intuitively, $\mathbf{x}^\top \mathbf{L} \mathbf{x}$ computes the differences between each adjacent pair and measures how volatile the signal is. This observation allows us to define signal frequency.

Definition 7 (Signal Average Frequency). *The frequency of a signal \mathbf{x} is defined as the Rayleigh quotient for \mathbf{L} and \mathbf{x} :*

$$\omega(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{L} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}. \quad (3.13)$$

According to Theorem 1, we have the following facts about $\omega(\mathbf{x})$.

1. $\omega(\mathbf{x})$ is non-negative.
2. The more a signal \mathbf{x} differs between adjacent nodes, the higher $\omega(\mathbf{x})$ it has. A low-frequency signal appears to be “smooth”, and a high-frequency signal appears to be “rough”.
3. If the graph is connected, $\omega(\mathbf{x})$ is 0 if and only if the signal is constant at every vertex, i.e., $\mathbf{x} = \alpha \vec{\mathbf{1}}$ for some constant α . If the graph is not connected, $\omega(\mathbf{x}) = 0$ indicates \mathbf{x} is constant within each connected component.
4. When the graph is bipartite, $\omega(\mathbf{x})$ can reach its maximum $2d_m$. The signal \mathbf{x} with the highest frequency satisfies $x_i = -x_j$ for all connected vertex pairs (ν_i, ν_j) .

In other words, one step walk on the graph will go from the signal's crest to the signal's trough.

5. $\omega(\mathbf{x})$ does not depend on the scale of the signal, i.e., $\omega(\mathbf{x}) = \omega(\alpha\mathbf{x})$ for all $\alpha \neq 0$.

These facts make $\omega(\mathbf{x})$ consistent with the properties of frequency in classic harmonic analysis, which justifies our definition. Because \mathbf{L} is symmetric, it is diagonalizable and has the following eigen-decomposition

$$\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top, \quad (3.14)$$

where $\mathbf{\Lambda}$ is a diagonal matrix storing eigenvalues, and $\mathbf{\Phi}$ is the eigenbasis. Eigenvalues and eigenvectors are arranged in such an order that λ_i corresponds exactly to eigenvector ϕ_i :

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}, \quad \mathbf{\Phi} = \begin{bmatrix} | & | & & | \\ \phi_1 & \phi_2 & \cdots & \phi_n \\ | & | & & | \end{bmatrix}. \quad (3.15)$$

Since \mathbf{L} is symmetric, its eigenbasis $\mathbf{\Phi}$ is unitary, i.e.,

$$\mathbf{\Phi}^\top \mathbf{\Phi} = \mathbf{I}, \quad \mathbf{\Phi}^{-1} = \mathbf{\Phi}^\top. \quad (3.16)$$

The frequency of eigenbasis is exactly the corresponding eigenvalues. Formally, if (λ_i, ϕ_i) is a pair of eigenvalue and eigenvector of \mathbf{L} , then the frequency of ϕ_i is λ_i :

$$\omega(\phi_i) = \frac{\phi_i^\top \mathbf{L} \phi_i}{\phi_i^\top \phi_i} = \frac{\lambda_i \phi_i^\top \phi_i}{\phi_i^\top \phi_i} = \lambda_i. \quad (3.17)$$

In graph signal processing, $\mathbf{\Phi}$ serves as the Fourier basis. Fourier transform calculates the weight of these basis vectors in signals. Given the basis, we can define Fourier transform as follows.

Definition 8 (Fourier transform). *The Fourier transform is an $\mathbb{R}^n \rightarrow \mathbb{R}^n$ mapping, which computes the projection of a signal \mathbf{x} on eigenbasis Φ :*

$$\mathbf{c} = \Phi^\top \mathbf{x} \quad \text{with} \quad c_i = \phi_i^\top \mathbf{x}. \quad (3.18)$$

\mathbf{c} is referred to as *Fourier coefficients of \mathbf{x}* , and c_i is the “amplitude” of ϕ_i -component in \mathbf{x} .

Fourier transform enables us to analyze the signal in the frequency (spectral) domain. $|c_i|$ reflects how much ϕ_i -component \mathbf{x} contains. The larger the magnitude c_i has, the more ϕ_i -component is present in the signal. Given Fourier coefficients \mathbf{c} , we can recover the original signal by inverse Fourier transform.

Definition 9 (Inverse Fourier transform). *Inverse Fourier transform recovers the signal \mathbf{x} from its Fourier coefficients \mathbf{c} :*

$$\mathbf{x} = \Phi \mathbf{c} = \sum_i c_i \phi_i. \quad (3.19)$$

Eq. (3.19) also states that a signal \mathbf{x} can be decomposed into a sum of several components with different frequencies. $\Phi \mathbf{c}$ is referred to as the Fourier decomposition of \mathbf{x} .

In classic harmonic analysis, Parseval’s theorem states that the Fourier transform does not change the *energy* of the signal, which is also true in graph signal processing.

Definition 10 (Energy of a signal). *The energy $E(\mathbf{x})$ of a signal \mathbf{x} is defined as the squared ℓ_2 norm of \mathbf{x} :*

$$E(\mathbf{x}) = \sum_{i=1}^n x_i^2. \quad (3.20)$$

The energy of eigenbasis vectors is always one, i.e., $E(\phi_i) = 1$ for all i . If we decompose signal \mathbf{x} into n components as in Eq. (3.19), the energy of each component is equal to

the squared coefficient:

$$E(c_i \phi_i) = c_i^2. \quad (3.21)$$

Theorem 2 (Parseval's theorem). *Let \mathbf{c} be the Fourier coefficient of \mathbf{x} , then the energy of \mathbf{x} is the sum of squared Fourier coefficients, i.e., $E(\mathbf{x}) = E(\mathbf{c})$, and*

$$E(\mathbf{x}) = \sum_{i=1}^n E(c_i \phi_i) = \sum_{i=1}^n c_i^2 = E(\mathbf{c}), \quad \text{where } \mathbf{c} = \Phi^\top \mathbf{x}. \quad (3.22)$$

Proof. This theorem directly follows that the eigenbasis Φ is unitary:

$$E(\mathbf{x}) = \mathbf{x}^\top \mathbf{x} = \mathbf{x}^\top \Phi \Phi^\top \mathbf{x} = \mathbf{c}^\top \mathbf{c}.$$

□

Eq. (3.22) also tells us that the energy of ϕ_i -components is c_i^2 , and the energy of a signal can be considered as distributed in each Fourier component. In turn, the components' energy also affects the signal frequency in the following way.

Theorem 3. *The frequency of a signal \mathbf{x} is equal to a weighted average of the frequencies of Fourier basis, and the weights are components' energy:*

$$\omega(\mathbf{x}) = \sum_{i=1}^n \frac{c_i^2}{\|\mathbf{c}\|_2^2} \lambda_i, \quad \text{where } \mathbf{c} = \Phi^\top \mathbf{x}. \quad (3.23)$$

Proof. By $\mathbf{L} = \Phi \Lambda \Phi^\top$, we have

$$\omega(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{L} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} = \frac{\mathbf{x}^\top \Phi \Lambda \Phi^\top \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} = \frac{\mathbf{c}^\top \Lambda \mathbf{c}}{\mathbf{c}^\top \mathbf{c}} = \sum_{i=1}^n \frac{c_i^2}{\|\mathbf{c}\|_2^2} \lambda_i.$$

□

Eq. (3.23) tells us that the frequency of components decides the frequency of a signal.

The signal has a low frequency if most energy is distributed in low-frequency compo-

nents. The signal has a high frequency if most energy is distributed in high-frequency components.

In summary, the eigenbasis of Laplacian serves as the Fourier basis, and the eigenvalues are interpreted as frequency. Given the Fourier basis, we can define the Fourier transform and the inverse Fourier transform for graph signals. Fourier transform enables us to analyze the signal in the frequency (spectrum) domain. A signal \mathbf{x} can be decomposed into a sum of several components with different frequencies. The weight of each component is determined by its amplitude $|c_i|$. The larger the amplitude is, the more ϕ_i -component is present in the signal. If a particular (low or high) frequency component dominates the signal, then the signal frequency will be close to it.

3.3 Normalized Laplacian

In addition to unnormalized Laplacian \mathbf{L} , there are two types of normalized Laplacian: symmetrically normalized \mathbf{L}_s and row normalized \mathbf{L}_r . Sometimes they are also used to define frequency as unnormalized Laplacian does. \mathbf{L}_s and \mathbf{L}_r are related to two types of normalized adjacency matrices: symmetrically normalized \mathbf{A}_s and row normalized \mathbf{A}_r , which are given by the following equations:

$$\mathbf{A}_s = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}, \quad (3.24)$$

$$\mathbf{L}_s = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{A}_s, \quad (3.25)$$

$$\mathbf{A}_r = \mathbf{D}^{-1} \mathbf{A}, \quad (3.26)$$

$$\mathbf{L}_r = \mathbf{D}^{-1} \mathbf{L} = \mathbf{I} - \mathbf{A}_r. \quad (3.27)$$

L_s and A_s are still symmetric, as unnormalized L . L_r and A_r are not symmetric, but they are closely related to random walks because A_r is the transition probability matrix. Row sums of A_r are equal to 1, and row sums of L_r are equal to 0.

Theorem 4 (Spectra of normalized Laplacians). *L_r and L_s have the following properties.*

1. For every signal $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{x}^\top \mathbf{L}_s \mathbf{x} = \frac{1}{2} \sum_{ij} a_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2. \quad (3.28)$$

2. L_r and L_s have exactly the same eigenvalues. λ is an eigenvalue of L_r with eigenvector ϕ , if and only if λ is an eigenvalue of L_s with eigenvector $D^{1/2}\phi$.

3. L_r and L_s have n eigenvalues, which fall in the range $[0, 2]$, i.e.,

$$0 = \lambda_1 \leq \dots \leq \lambda_n \leq 2. \quad (3.29)$$

4. λ_n is 2 if and only if the graph is bipartite.

5. The smallest eigenvalue λ_1 is 0, and the corresponding eigenvectors are $\vec{\mathbf{1}}$ and $D^{1/2}\vec{\mathbf{1}}$ for L_s and L_r respectively.

Proof.

1. Eq. (3.28) follows Eq. (3.7):

$$\begin{aligned} \mathbf{x}^\top \mathbf{L}_s \mathbf{x} &= (D^{-1/2}\mathbf{x})^\top \mathbf{L} (D^{-1/2}\mathbf{x}) \\ &= \frac{1}{2} \sum_{ij} a_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2. \end{aligned}$$

2. If $\mathbf{L}_r \phi = \lambda \phi$, we can multiply both side by $\mathbf{D}^{1/2}$, and get

$$\begin{aligned}
 & \mathbf{L}_r \phi = \lambda \phi \\
 \iff & \mathbf{D}^{-1} \mathbf{L} \phi = \lambda \phi \\
 \iff & \mathbf{D}^{-1/2} \mathbf{L} \phi = \lambda \mathbf{D}^{1/2} \phi \\
 \iff & \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} (\mathbf{D}^{1/2} \phi) = \lambda \mathbf{D}^{1/2} \phi \\
 \iff & \mathbf{L}_s (\mathbf{D}^{1/2} \phi) = \lambda (\mathbf{D}^{1/2} \phi),
 \end{aligned}$$

so λ is also an eigenvalue of \mathbf{L}_s , and the corresponding eigenvector is $\mathbf{D}^{1/2} \phi$.

3. Eq. (3.28) indicates that \mathbf{L}_s is positive semi-definite, and thus all eigenvalues of \mathbf{L}_s are non-negative. Next, we consider the largest eigenvalue λ_n . Let $\mathbf{y} = \mathbf{D}^{-1/2} \mathbf{x}$, then $\mathbf{x}^\top \mathbf{L}_s \mathbf{x} = \mathbf{y}^\top \mathbf{L} \mathbf{y}$. Analogous to inequality (3.8), we have

$$\begin{aligned}
 \mathbf{x}^\top \mathbf{L}_s \mathbf{x} = \mathbf{y}^\top \mathbf{L} \mathbf{y} &= \frac{1}{2} \sum_{ij} a_{ij} (y_i - y_j)^2 \\
 &\leq \sum_{ij} a_{ij} (y_i^2 + y_j^2) = 2 \sum_i d_i y_i^2 \\
 &= 2 \mathbf{y}^\top \mathbf{D} \mathbf{y} = 2 \mathbf{x}^\top \mathbf{x},
 \end{aligned} \tag{3.30}$$

so the largest eigenvalue λ_n is less than or equal to 2:

$$\lambda_n = \max_{\mathbf{x}} \frac{\mathbf{x}^\top \mathbf{L}_s \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} \leq 2.$$

By property 2, \mathbf{L}_r has exact same eigenvalues as \mathbf{L}_s and they also fall in the range $[0, 2]$.

4. Equality in (3.30) holds, if and only if $x_i = -x_j$ for any two connected vertices ν_i and ν_j . The maximum 2 is achieved in such case, and the graph is bipartite.

5. It is easy to verify that

$$\mathbf{L}_r \vec{\mathbf{1}} = \vec{\mathbf{0}} = \mathbf{L}_s (\mathbf{D}^{1/2} \vec{\mathbf{1}}), \tag{3.31}$$

Table 3.2: Let $\phi_s = D^{1/2}\phi_r$, then the following four equations are equivalent.

	Symmetrically normalized	Row normalized	Eigenvalue range
Laplacian	$L_s\phi_s = \lambda\phi_s$	$L_r\phi_r = \lambda\phi_r$	$[0, 2]$
Adjacency	$A_s\phi_s = (1 - \lambda)\phi_s$	$A_r\phi_r = (1 - \lambda)\phi_r$	$[-1, 1]$

so property 4 is proved. \square

Spectra of A_s, A_r are closely related to spectra of L_s and L_r .

Theorem 5 (Spectra of normalized adjacency matrix). *Spectra of A_s, A_r relate to L_s, L_r in the following ways.*

1. A_r has exactly the same eigenvectors as L_r . λ is an eigenvalue of L_r with eigenvector ϕ , if and only if ϕ is also an eigenvector of A_r with eigenvalue $1 - \lambda$.
2. A_s has exactly the same eigenvectors with L_s . λ is an eigenvalue of L_s with eigenvector ϕ , if and only if ϕ is also an eigenvector of A_s with eigenvalue $1 - \lambda$.

Proof. By definitions of L_s and L_r , we have

$$\begin{aligned}
 L_s\phi &= \lambda\phi & L_r\phi &= \lambda\phi \\
 \Leftrightarrow (I - A_s)\phi &= \lambda\phi & \Leftrightarrow (I - A_r)\phi &= \lambda\phi \\
 \Leftrightarrow A_s\phi &= (1 - \lambda)\phi & \Leftrightarrow A_r\phi &= (1 - \lambda)\phi.
 \end{aligned}$$

A_r, L_r have same eigenvectors, and so do A_r, L_r . Relations among spectra of $L_s, L_r,$

A_s, A_r are summarized in Table 3.2 \square

3.3.1 Frequency and Fourier Transform under Normalized Laplacian

Theorem 4 shows that L_s and L_r share lots of similar properties with L , so we have the following alternatives to defining signal frequency:

$$\omega_s(\mathbf{x}) = \frac{\mathbf{x}^\top L_s \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}, \quad \omega_r(\mathbf{x}) = \frac{\mathbf{x}^\top L_r \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}. \quad (3.32)$$

$\omega_s(\mathbf{x})$ and $\omega_r(\mathbf{x})$ share most properties of $\omega(x)$ as in Section 3.2. In this case, the eigenbasis of L_s or L_r serves as the Fourier basis instead of the one of L . Assume we have the following eigen-decomposition for L_s and L_r^* :

$$L_s = \Phi_s \Lambda \Phi_s^\top, \quad L_r = \Phi_r \Lambda \Phi_r^{-1}. \quad (3.33)$$

We can still define the Fourier transform and the inverse Fourier transform.

Definition 11. *If the eigenbasis of L_s or L_r serves as the Fourier basis, then the Fourier transform is defined as*

$$\mathbf{c}_s = \Phi_s^\top \mathbf{x} \quad \text{or} \quad \mathbf{c}_r = \Phi_r^{-1} \mathbf{x}, \quad (3.34)$$

and the inverse Fourier transform is defined as

$$\mathbf{x} = \Phi_s \mathbf{c}_s \quad \text{or} \quad \mathbf{x} = \Phi_r \mathbf{c}_r. \quad (3.35)$$

Parseval's theorem is also true for L_s , and the signal frequency is still a weighted average over components' frequency by component energy:

$$E(\mathbf{x}) = \|\mathbf{c}_s\|_2^2, \quad \omega_s(\mathbf{x}) = \frac{1}{\|\mathbf{c}_s\|_2^2} \sum_i (c_s)_i^2 \lambda_i. \quad (3.36)$$

*Note that L_r is not symmetric, so its eigenbasis Φ_r is not unitary.

However, Eq. (3.36) is not true for \mathbf{L}_r , because Φ_r is not unitary.

The main advantage of normalized Laplacians over the unnormalized one is that their polynomials are also the polynomials of the corresponding adjacency matrix, which brings convenience to defining graph convolution in the next section.

3.4 Graph Convolution

3.4.1 Graph Filter

Analogous to classical signal processing, filters are transformations between signals.

Definition 12 (Filter). *A graph filter $h : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a function that takes a signal as the input and produces another signal:*

$$\mathbf{y} = h(\mathbf{x}). \quad (3.37)$$

There are various filters, among which linear filters are of particular interest to us.

Definition 13 (Linear filter). *A filter h is said to be linear if and only if for all graph signals $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$, linearity holds:*

$$h(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha h(\mathbf{x}) + \beta h(\mathbf{y}). \quad (3.38)$$

All filters satisfying Eq. (3.38) are indeed linear transformations. If filter $h(\cdot)$ is linear, there must exist a matrix \mathbf{H} such that

$$h(\mathbf{x}) = \mathbf{H}\mathbf{x}.$$

3.4.2 Graph Convolution

Convolution is a special family of linear filters. Here we introduce the whole story in the context of symmetrically normalized Laplacian L_s .

Definition 14. (Convolution) A graph convolution \mathbf{H} is a polynomial of adjacency matrix \mathbf{A}_s :

$$\mathbf{H} = p(\mathbf{A}_s) = \sum_{k=0}^K \theta_k \mathbf{A}_s^k, \quad (3.39)$$

where $p(\cdot)$ is a polynomial of degree K and $\boldsymbol{\theta} = [\theta_0, \dots, \theta_K]$ are polynomial coefficients.

We call $p(\cdot)$ the convolution kernel and K the size of the kernel.

Here, $p(\cdot)$ denotes a polynomial that can be evaluated on both scalars and square matrices.

For example, a polynomial $p(t) = 1 + 2t + 3t^2$, when evaluated on matrix \mathbf{A} , is $p(\mathbf{A}) = \mathbf{I} + 2\mathbf{A} + 3\mathbf{A}^2$.

Eq. (3.39) gives us a clear spatial interpretation of convolution — for each vertex, the convolution result is a weighted sum of nearby signal values, and the weights are $\boldsymbol{\theta}$. If there exists a path of length k from ν_i to ν_j , we say ν_j is a k -hop neighbor of ν_i . The non-zero elements are closely related to such k -hop neighbors:

$$(\mathbf{A}_s^k)_{ij} \neq 0 \iff \nu_j \text{ is a } k\text{-hop neighbor of } \nu_i,$$

so \mathbf{A}_s^k only connects a vertex with its k -hop neighbors. For example, $\mathbf{A}_s^0 = \mathbf{I}$ only connects each vertex with itself, so θ_0 is the weight for the vertex itself during convolution. $\mathbf{A}_s^1 = \mathbf{A}_s$ only connects each vertex with its neighbors, so θ_1 is the weight for these direct neighbors. $\mathbf{A}_s^2 = \mathbf{A}^2$ connects each vertex with its neighbors' neighbors, and so on. This spatial interpretation is consistent with the convolution definition in classic harmonic

analysis, which is also a weighted sum of neighbors. Denote the shortest-path distance[†] between two vertices by $d(\nu_i, \nu_j)$, we have the following theorem.

Theorem 6 (Locality of convolution). *For any convolutional filter $\mathbf{H} = p(\mathbf{A}_s)$, if the degree of polynomial $p(\cdot)$ is K , then the result of each vertex only relies on its K -hop neighborhood, i.e.,*

$$d(\nu_i, \nu_j) > K \implies H_{ij} = 0. \quad (3.40)$$

For any signal $\mathbf{y} = \mathbf{H}(\mathbf{x})$, we have

$$y_i = \sum_{\nu_j \in \mathcal{N}_K(\nu_i)} H_{ij} x_j. \quad (3.41)$$

where $\mathcal{N}_K(\nu_i) = \{\nu_j | d(\nu_i, \nu_j) \leq K\}$ is the neighbors of ν_i within K hops.

Proof. For $k = 0, 1, \dots, K$, if $d(\nu_i, \nu_j) > K$, then $(\mathbf{A}_s^k)_{ij} = 0$. Note that $\mathbf{H} = \sum_{k=0}^K \theta_k \mathbf{A}_s^k$, so Eq. (3.40) is true. Eq. (3.41) is a direct consequence of Eq. (3.40). \square

$\mathcal{N}_K(\nu_i) = \{\nu_j | d(\nu_i, \nu_j) \leq K\}$ is referred to as the reception field of ν_i . This theorem explains why polynomial degree K is named as the kernel size because it controls the size of the reception field.

In harmonic analysis, convolutional filters are linear and shift-invariant, and all linear shift-invariant filters are convolutional filters. This is also true in the graph domain. Given a signal \mathbf{x} , shifting it by one step is to multiply it by the adjacency matrix \mathbf{A}_s to get $\mathbf{A}_s \mathbf{x}$.

Definition 15 (Shift-invariant filter). *Shift-invariant filters are the filters satisfying*

$$\mathbf{A}_s h(\mathbf{x}) = h(\mathbf{A}_s \mathbf{x}), \quad (3.42)$$

[†]Shortest-path distance between two vertices is the length of the shortest path connecting them, denoted by $d(\nu_i, \nu_j)$.

i.e., filters that are inter-changeable with shifting operation.

Theorem 7 (Linear shift-invariant filter). *A filter is a convolution if and only if it is both linear and shift-invariant.*

Proof. Convolution is linear by definition. Here, we show it is also shift-invariant:

$$\mathbf{A}_s \mathbf{H} \mathbf{x} = (\mathbf{A}_s \mathbf{H}) \mathbf{x} = \left(\sum_{k=0}^K \theta_k \mathbf{A}_s^{k+1} \right) \mathbf{x} = (\mathbf{H} \mathbf{A}_s) \mathbf{x} = \mathbf{H} \mathbf{A}_s \mathbf{x}. \quad (3.43)$$

The proof for the other direction — all linear shift-invariant filters are convolutional filters, can be found in Sandryhaila and Moura [21]. \square

3.4.3 Convolution Theorem

Note that $\mathbf{L}_s = \mathbf{I} - \mathbf{A}_s$, so polynomials of \mathbf{L}_s and \mathbf{A}_s are also polynomials of each other.

Theorem 8. *A polynomial of \mathbf{A}_s with degree K is also a polynomial of \mathbf{L}_s with a degree at most K . A polynomial of \mathbf{L}_s with degree K is also a polynomial of \mathbf{A}_s with a degree at most K .*

Proof. By $\mathbf{L}_s = \mathbf{I} - \mathbf{A}_s$, we have

$$p(\mathbf{L}_s) = \sum_{k=0}^K \theta_k \mathbf{A}_s^k = \sum_{k=0}^K \theta_k (\mathbf{I} - \mathbf{A}_s)^k. \quad (3.44)$$

Apparently, the right side, after expansion, is a polynomial of \mathbf{A}_s with a degree at most K . The proof for the other direction is similar. \square

Theorem 8 shows that convolutional filters can also be defined as polynomials of \mathbf{L}_s :

$$\mathbf{H} = p(\mathbf{L}_s) = \sum_{k=0}^K \theta_k \mathbf{L}_s^k. \quad (3.45)$$

This definition is convenient for proving the convolution theorem. In classic harmonic analysis, the convolution theorem states that applying a convolution to a signal is equivalent to scaling the signal's Fourier coefficients in the frequency domain.

Theorem 9 (Convolution theorem). *Given a convolutional filter $\mathbf{H} = p(\mathbf{L}_s)$ with kernel $p(\cdot)$, if signals \mathbf{x}, \mathbf{y} satisfy $\mathbf{y} = \mathbf{H}\mathbf{x}$, then they have Fourier coefficients $\mathbf{c}^{(x)} = \Phi^\top \mathbf{x}$ and $\mathbf{c}^{(y)} = \Phi^\top \mathbf{y}$ such that*

$$\mathbf{c}^{(y)} = p(\Lambda)\mathbf{c}^{(x)}, \quad (3.46)$$

or in element-wise form

$$c_i^{(y)} = p(\lambda_i)c_i^{(x)}, \quad (3.47)$$

for $i = 1, 2, \dots, n$.

Proof. According to the properties of matrix polynomial, $p(\mathbf{L}_s)$ is similar to $p(\Lambda)$:

$$\begin{aligned} p(\mathbf{L}_s) &= \sum_{k=0}^K \theta_k \mathbf{L}_s^k = \sum_{k=0}^K \theta_k \Phi_s \Lambda^k \Phi_s^\top \\ &= \Phi_s \left(\sum_{k=0}^K \theta_k \Lambda^k \right) \Phi_s^\top = \Phi_s p(\Lambda) \Phi_s^\top, \end{aligned} \quad (3.48)$$

where $p(\Lambda)$ is a diagonal matrix with i -th diagonal element equal to $p(\lambda_i)$:

$$p(\Lambda) = \begin{bmatrix} \sum_k \theta_k \lambda_1^k & & & \\ & \sum_k \theta_k \lambda_2^k & & \\ & & \ddots & \\ & & & \sum_k \theta_k \lambda_n^k \end{bmatrix} = \begin{bmatrix} p(\lambda_1) & & & \\ & p(\lambda_2) & & \\ & & \ddots & \\ & & & p(\lambda_n) \end{bmatrix} \quad (3.49)$$

Substitute $p(\mathbf{L}_s)$ by Eq. (3.48), and the theorem is proved as follows:

$$\begin{aligned} \mathbf{y} &= \mathbf{H}\mathbf{x} \\ \implies \mathbf{y} &= \Phi_s p(\Lambda) \Phi_s^\top \mathbf{x} \\ \implies \Phi_s^\top \mathbf{y} &= p(\Lambda) \Phi_s^\top \mathbf{x} \\ \implies \mathbf{c}^{(y)} &= p(\Lambda) \mathbf{c}^{(x)}. \end{aligned}$$

In element-wise form,

$$c_i^{(y)} = p(\lambda_i)c_i^{(x)}. \quad (3.50)$$

□

Real-valued polynomial $p(\lambda) : \mathbb{R} \rightarrow \mathbb{R}$ is referred to as the frequency response function, because $p(\lambda_i)$ is the scaling ratio for the component of frequency λ_i . In practice, response functions are not necessarily a polynomial but can be any real-valued function, due to the following reason.

Definition 16. For any real-valued function $f(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$, it is evaluated on matrix \mathbf{L}_s

as

$$f(\mathbf{L}_s) = \mathbf{\Phi}_s f(\mathbf{\Lambda}) \mathbf{\Phi}_s^\top = \mathbf{\Phi}_s \begin{bmatrix} f(\lambda_1) & & & \\ & f(\lambda_2) & & \\ & & \ddots & \\ & & & f(\lambda_n) \end{bmatrix} \mathbf{\Phi}_s^\top, \quad (3.51)$$

i.e., apply $f(\cdot)$ to every eigenvalue of \mathbf{L}_s .

Theorem 10. For any real-valued function $f(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$, there exists a polynomial $p(\cdot)$ such that $f(\mathbf{L}_s) = p(\mathbf{L}_s)$, such that $f(\mathbf{L}_s)$ is a valid convolutional filter.

Proof. We only need to find a polynomial p such that

$$f(\lambda_i) = p(\lambda_i), \quad \text{for } i = 1, 2, \dots, n. \quad (3.52)$$

The interpolation theorem tells us that there exists a polynomial of degree at most $n - 1$ to satisfy the requirement. □

As a result, any real-valued function can serve as the response function of convolutional filters. We can extend the definition of convolution from polynomials $p(\mathbf{L}_s)$ to $f(\mathbf{L}_s)$. Ex-

tending available response functions to any real-valued functions gives us more freedom to design convolutional filters.

3.4.4 Unnormalized and Row-Normalized Convolution

Unnormalized Convolution We can also define convolution as a polynomial of unnormalized Laplacian L or adjacency matrix A :

$$\mathbf{H}_1 = p(L), \quad \mathbf{H}_2 = p(A). \quad (3.53)$$

The two definitions are not equivalent to the previous ones based on L_s and are not equivalent to each other either. \mathbf{H}_1 cannot be interpreted as the weighted sum of neighborhoods. \mathbf{H}_2 does not satisfy the convolution theorem unless we change the Fourier basis from the eigenbasis of L to the eigenbasis of A .

Row-normalized Convolution Another definition of convolution is based on row-normalized Laplacian L_r or adjacency matrix A_r :

$$\mathbf{H}_r = p_1(L_r) = p_2(A_r). \quad (3.54)$$

Since polynomials of L_r or A_r are also polynomials of each other, the two definitions are equivalent. The definitions relate to those based on symmetrically normalized Laplacian in the following way:

$$p_1(L_r) = D^{-1/2} p_1(L_s) D^{1/2}, \quad (3.55)$$

$$p_2(A_r) = D^{-1/2} p_2(A_s) D^{1/2}. \quad (3.56)$$

3.4.5 Efficient Computation of Convolution

If we want to apply a convolutional filter \mathbf{H} to a signal \mathbf{x} , the straightforward way is to calculate the filter itself first by

$$\mathbf{H} = p(\mathbf{A}_s) = \sum_{k=0}^K \theta_k \mathbf{A}_s^k, \quad (3.57)$$

and then apply it to \mathbf{x}

$$\mathbf{y} = \mathbf{H}\mathbf{x}. \quad (3.58)$$

Assume the graph has n vertices, and the degree of $p(\cdot)$ is K , then calculating single matrix multiplication requires $\mathcal{O}(n^3)$ operations and Eq. (3.57) requires $\mathcal{O}(K^2)$ matrix multiplications, so the overall computational complexity is $\mathcal{O}(K^2n^3)$, which is very high. Since polynomials can be calculated iteratively, e.g.,

$$1 + 2t + 3t^2 + 4t^3 = 1 + t(2 + t(3 + 4t))$$

, it inspires us to calculate $\mathbf{H}\mathbf{x}$ iteratively as follows:

$$\begin{aligned} \mathbf{y}^{(0)} &= \theta_K \mathbf{x}, \\ \mathbf{y}^{(1)} &= \theta_{K-1} \mathbf{x} + \mathbf{A}_s \mathbf{y}^{(0)}, \\ \mathbf{y}^{(2)} &= \theta_{K-2} \mathbf{x} + \mathbf{A}_s \mathbf{y}^{(1)}, \\ &\vdots \\ \mathbf{y}^{(K)} &= \theta_0 \mathbf{x} + \mathbf{A}_s \mathbf{y}^{(K-1)}. \end{aligned} \quad (3.59)$$

It is easy to verify that $\mathbf{y}^{(K)} = \mathbf{H}\mathbf{x}$. Each iteration takes $\mathcal{O}(n^2)$ operations, and there are K iterations, so the total complexity is $\mathcal{O}(Kn^2)$. In addition, if the graph is sparse, i.e., only a few elements in \mathbf{A}_s are non-zero, then the complexity is even lower. Denote the number of non-zero elements in \mathbf{A}_s by N with $N \ll n^2$, and then the complexity is $\mathcal{O}(KN)$.

3.5 Low-pass Filters

Recall the convolutional theorem,

$$f(\mathbf{L}_s) = \mathbf{\Phi} f(\mathbf{\Lambda}) \mathbf{c} = \sum_{i=0}^N f(\lambda_i) c_i \phi_i, \quad (3.60)$$

where $f(\lambda)$ is the frequency response function, and $f(\lambda_i)$ is the scaling coefficient for the component of frequency λ_i . A filter is said to be low-pass if it removes high-frequency components and reserves low-frequency components. Specifically, $f(\lambda)$ approaches 1, when λ approaches 0, and $f(\lambda)$ approaches 0, when λ is large. A basic fact is that eigenvalues of normalized Laplacian are bounded in $[0, 2]$ and eigenvalues of unnormalized Laplacian are bounded in $[0, 2d_m]$, where d_m is the graph degree. Thus, high frequency means frequencies close to 2 for normalized graphs, and those close to $2d_m$ for unnormalized graphs.

Ideal Low-Pass Filter An ideal low-pass filter has a cutoff frequency λ_{cut} . It removes all components with frequency higher than λ_{cut} and reserves those with frequency lower than λ_{cut} . The response function is in the form of

$$f(\lambda) = \begin{cases} 1 & \text{if } \lambda \leq \lambda_{\text{cut}} \\ 0 & \text{if } \lambda > \lambda_{\text{cut}}, \end{cases} \quad (3.61)$$

and the filter is

$$f(\mathbf{L}_s) = \sum_{0 \leq \lambda_i \leq \lambda_{\text{cut}}} \phi_i \phi_i^T.$$

However, computing such a filter requires eigen-decomposition of the graph Laplacian, which is impractical for large graphs.

Taubin Filter Taubin filter [93] is a polynomial approximation of the ideal low-pass filter within a fixed range, such as $[0, 2]$. It is defined as

$$p(\lambda) = \left[(1 - s\lambda)(1 - t\lambda) \right]^K, \quad (3.62)$$

$$p(\mathbf{L}_s) = \left[(1 - s\mathbf{L}_s)(1 - t\mathbf{L}_s) \right]^K, \quad (3.63)$$

where $s \in [0, \frac{1}{2}]$ and $t \leq -s$. Note that $p(0) = 1, p(s) = p(t) = 0$, and

$$\lambda \in [0, \frac{1}{t} + \frac{1}{s}] \implies p(\lambda) \in [1, +\infty), \quad (3.64)$$

$$\lambda \in (\frac{1}{t} + \frac{1}{s}, \frac{1}{s}] \implies p(\lambda) \in [0, 1). \quad (3.65)$$

Apparently, Taubin filter amplifies components of frequency in $[0, \frac{1}{t} + \frac{1}{s}]$, and suppresses those of frequency in $(\frac{1}{t} + \frac{1}{s}, \frac{1}{s}]$. When K is sufficiently large, $p(\lambda)$ approximates zero for $\lambda \in (\frac{1}{t} + \frac{1}{s}, 2]$. So, the cutoff frequency is $\lambda_{\text{cut}} = \frac{1}{t} + \frac{1}{s}$. If we already know the desired λ_{cut} , the parameters s, t can be chosen as follows:

$$\begin{cases} \frac{1}{s} = 2 \\ \lambda_{\text{cut}} = \frac{1}{t} + \frac{1}{s} \end{cases} \implies \begin{cases} s = \frac{1}{2} \\ t = \frac{1}{\lambda_{\text{cut}} - 2} \end{cases}. \quad (3.66)$$

Gaussian Filter Another typical low-pass filter is the Gaussian filter. It is a low-pass filter with a Gaussian response function

$$f(\lambda) = \exp\left(-\frac{\lambda^2}{\sigma^2}\right), \quad (3.67)$$

where parameter σ controls the strength of the filter. Small σ indicates strong smoothness.

Obtaining a precise Gaussian filter also requires to compute the eigen-decomposition of the graph Laplacian. However, we can approximate $f(\lambda)$ by its Taylor series centered around 0:

$$f(\lambda) \approx p(\lambda) = \sum_{k=0}^K (-1)^k \frac{1}{k!} \left(\frac{x}{\sigma}\right)^{2k}. \quad (3.68)$$

The larger K is, the more accurate the approximation is. Eigenvalues of normalized Laplacian are bounded in $[0, 2]$, so only a few terms are sufficient to get a good approximation. The final filter will be

$$p(\mathbf{L}_s) = \sum_{k=0}^K \frac{(-1)^k}{k! \sigma^{2k}} \mathbf{L}_s^{2k}. \quad (3.69)$$

We could also approximate $f(\lambda)$ in range $[0, 2]$ by Chebyshev series

$$f(\lambda) \approx p(\lambda) = \sum_{k=0}^K \theta_k T_k(\lambda - 1), \quad (3.70)$$

where $T_k(\cdot)$ is the k -th Chebyshev polynomial. The parameter is decided by

$$\theta_k = \int_0^2 T_k(\lambda - 1) f(\lambda). \quad (3.71)$$

In this case, the filter becomes

$$p(\mathbf{L}_s) = \sum_{k=0}^K \theta_k T_k(\mathbf{L}_s - \mathbf{I}) = \sum_{k=0}^K \theta_k T_k(-\mathbf{A}_s). \quad (3.72)$$

Random Walk Filter Another practical low-pass filter for machine learning is

$$p(\mathbf{L}) = \left(1 - \frac{1}{u} \mathbf{L}\right)^k, \quad (3.73)$$

where u is a proper upper bound of eigenvalues. For normalized Laplacian, u should be 2. For unnormalized Laplacian, if the degree d_m of the graph is known, u can be $2d_m$. Otherwise, d_m is bounded by the number of vertices n , so u can be $2n$. If the graph is a s -NN graph, d_m is bounded by $2s$, so u can be $4s$. The response function $f(\lambda)$ here is designed to go through points $(0, 1)$ and $(u, 0)$. The exponent k controls the strength of low-passness.

Table 3.3: Notation Table II.

Notation	Description
$\vec{\mathbf{1}}$ (Bold)	All-one vector
$\vec{\mathbf{0}}$ (Bold)	All-zero vector
\mathbf{I}	Identity matrix
\mathbf{x}, \mathbf{y}	Signals
$d_i = \sum_j a_{ij}$	Degree of ν_i
$\mathbf{D} = \text{diag}([d_1, \dots, d_n])$	(Diagonal) degree matrix
$\mathbf{L} = \mathbf{D} - \mathbf{A}$	Unnormalized Laplacian matrix
$\{\mathcal{C}_k\}$	Connected components of the graph
λ_i, ϕ_i	Eigenpair of the Laplacian matrix
$\omega(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{L} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}$	Signal frequency
$\mathbf{L} = \Phi \Lambda \Phi^{-1}$	Eigen-decomposition of Laplacian
$\mathbf{A}_s = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$	Symmetrically normalized adjacency matrix
$\mathbf{L}_s = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{A}_s$	Symmetrically normalized Laplacian
$\mathbf{A}_r = \mathbf{D}^{-1} \mathbf{A}$	Row normalized Laplacian matrix
$\mathbf{L}_r = \mathbf{D}^{-1} \mathbf{L} = \mathbf{I} - \mathbf{A}_r$	Row normalized adjacency matrix
\mathbf{H}	Linear filter, or hidden layer of neural networks
$d(\nu_i, \nu_j)$	Distance between (ν_i, ν_j)
$\mathcal{N}_k(\nu_i) = \{\nu_j d(\nu_i, \nu_j) \leq k\}$	k -hop neighborhood of ν_i
$\mathcal{N}(\nu_i) = \{\nu_j (\nu_i, \nu_j) \in \mathcal{E}\}$	Direct neighbors of ν_i
$p(\lambda) : \mathbb{R} \rightarrow \mathbb{R}$	Polynomial evaluated on scalar λ
$p(\mathbf{A}) : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n}$	Matrix polynomial evaluated on square matrix \mathbf{A}
$f(\lambda) : \mathbb{R} \rightarrow \mathbb{R}$	Real-valued function
$f(\mathbf{A}) : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n}$	Apply $f(\cdot)$ to eigenvalues of \mathbf{A}
$T_k(\cdot)$	k -order Chebyshev polynomial

3.6 Multi-dimensional Graph Signal Processing

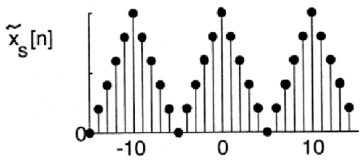
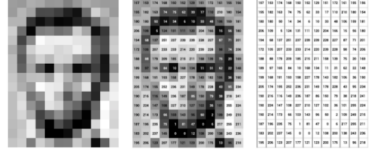
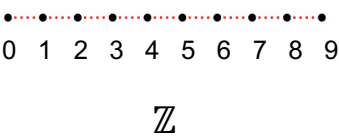
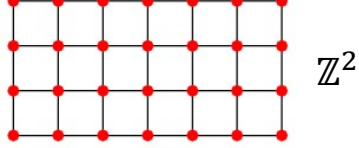
	1D	2-D
Euclidean Signals		
Discretized Euclidean space as a Graph		

Table 3.4: N-dimensional discretized Euclidean space. $\mathbb{Z}^n = \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}$.

3.6.1 Cartesian product of Graphs

In Euclidean space, signals are categorized according to the dimension of the space they lie in. For example, sound waves lie in 1-D Euclidean space, so they are 1-D signals; images lie in 2-D Euclidean space, so they are 2-D signals. An N -dimensional space is actually the Cartesian product of N 1-D spaces. The concepts of dimension and Cartesian product also exist in the graph domain. As Table 3.4 illustrates, the Cartesian product of two chains (graph representation of \mathbb{Z}) results in a 2-D grid, which is the graph representation of \mathbb{Z}^2 . In this section, we will briefly introduce multi-dimensional graph signal processing. Readers may refer to Kurokawa et al. [94] for a more comprehensive introduction.

Given two graphs $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathbf{A}^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathbf{A}^{(2)})$ with edge set $\mathcal{E}^{(1)}$ and $\mathcal{E}^{(2)}$, their Cartesian product $\mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$ is another graph with vertex set

$$\mathcal{V} = \mathcal{V}^{(1)} \times \mathcal{V}^{(2)} = \{(\nu_{i_1}^{(1)}, \nu_{i_2}^{(2)}) \mid i_1 = 1, \dots, n_1; i_2 = 1, \dots, n_2\}, \quad (3.74)$$

where $n_1 = |\mathcal{V}^{(1)}|$, $n_2 = |\mathcal{V}^{(2)}|$ are the numbers of vertices in $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$ respectively. The resulted product \mathcal{G} has $n_1 n_2$ vertices, and its adjacency matrix \mathbf{A} is of shape $n_1 n_2 \times n_1 n_2$.

The following formula specifies edge weights:

$$a_{(i_1 i_2), (j_1 j_2)} = a_{i_1 j_1}^{(1)} \mathbf{I}(i_2, j_2) + a_{i_2 j_2}^{(2)} \mathbf{I}(i_1, j_1), \quad (3.75)$$

where function $\mathbf{I}(\cdot, \cdot)$ is defined as

$$\mathbf{I}(i, j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \quad (3.76)$$

Readers may think of \mathbf{I} as an identity matrix of infinite size and i, j as element indices.

Intuitively, there is an edge between (ν_{i_1}, ν_{i_2}) and (ν_{j_1}, ν_{j_2}) in the product graph, if and only if $\nu_{i_1} = \nu_{j_1}$ and there is an edge between ν_{i_2}, ν_{j_2} in $\mathcal{G}^{(2)}$, or $\nu_{i_2} = \nu_{j_2}$ and there is an edge between ν_{i_1}, ν_{j_1} in $\mathcal{G}^{(1)}$, i.e.,

$$a_{(i_1 i_2), (j_1 j_2)} \neq 0 \iff \left((a_{i_1 j_1}^{(1)} \neq 0) \text{ and } (i_2 = j_2) \right) \text{ or } \left((i_1 = j_1) \text{ and } (a_{i_2 j_2}^{(2)} \neq 0) \right). \quad (3.77)$$

Eq. (3.75) describes edge weights of Cartesian product in element-wise manner. However, we can describe Eq. (3.75) more concisely with the help of *Kronecker product* and *Kronecker sum*. A Kronecker product [95] of $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{p \times q}$ is a matrix given by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11} \mathbf{B} & \cdots & a_{1n} \mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1} \mathbf{B} & \cdots & a_{mn} \mathbf{B} \end{bmatrix} \in \mathbb{R}^{mp \times nq}. \quad (3.78)$$

For two square matrices $\mathbf{A} \in \mathbb{R}^{m \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times n}$, their Kronecker sum is defined by

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_n + \mathbf{I}_m \otimes \mathbf{B}, \quad (3.79)$$

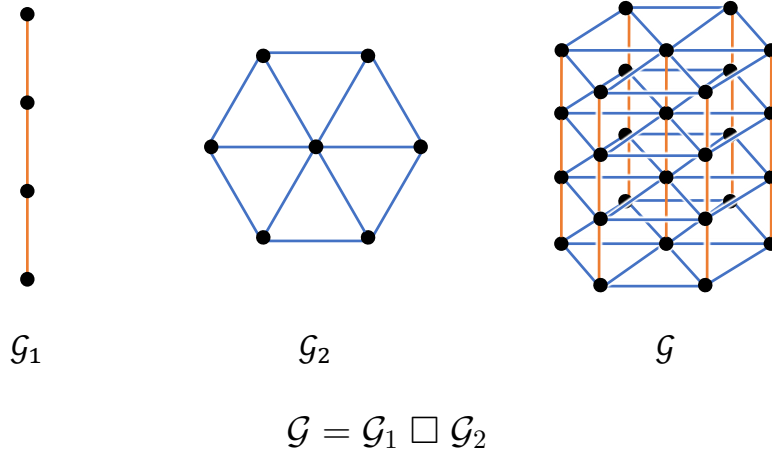


Figure 3.1: An example of graph product. Orange edges come from \mathcal{G}_1 ; blue ones come from \mathcal{G}_2 .

where \mathbf{I}_n and \mathbf{I}_m are identity matrices of size $n \times n$ and $m \times m$ respectively. It is easy to verify that the adjacency matrix \mathbf{A} of the product graph is the Kronecker sum of factor graphs' adjacency matrices $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$:

$$\mathcal{G}^{(1)} \square \mathcal{G}^{(2)} = (\mathcal{V}, \mathbf{A}) \implies \mathbf{A} = \mathbf{A}^{(1)} \oplus \mathbf{A}^{(2)}. \quad (3.80)$$

Then, we can define the Cartesian product of graphs.

Definition 17 (Cartesian product of graphs). *Given two graphs $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathbf{A}^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathbf{A}^{(2)})$, their Cartesian product, denoted by $\mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$, is another graph*

$$\mathcal{G} = (\mathcal{V}, \mathbf{A}) = \mathcal{G}^{(1)} \square \mathcal{G}^{(2)}, \quad (3.81)$$

where

$$\mathcal{V} = \mathcal{V}^{(1)} \times \mathcal{V}^{(2)} \text{ and } \mathbf{A} = \mathbf{A}^{(1)} \oplus \mathbf{A}^{(2)}. \quad (3.82)$$

We refer to \mathcal{G} as the product graph, and $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$ as factor graphs.

3.6.2 Properties of Kronecker Product

The rest of this section heavily relies on Kronecker product. Here, we introduce several important properties of Kronecker product. We refer readers to Wikipedia [95] for more properties about Kronecker product.

1. Kronecker Product is bilinear and associative:

$$\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{C}, \quad (3.83)$$

$$(\mathbf{B} + \mathbf{C}) \otimes \mathbf{A} = \mathbf{B} \otimes \mathbf{A} + \mathbf{C} \otimes \mathbf{A}, \quad (3.84)$$

$$(k\mathbf{A}) \otimes \mathbf{B} = \mathbf{A} \otimes (k\mathbf{B}) = k(\mathbf{A} \otimes \mathbf{B}), \quad (3.85)$$

$$(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}). \quad (3.86)$$

As a result, Kronecker sum and the graph Cartesian product are also associative:

$$(\mathbf{A}_1 \oplus \mathbf{A}_2) \oplus \mathbf{A}_3 = \mathbf{A}_1 \oplus (\mathbf{A}_2 \oplus \mathbf{A}_3), \quad (3.87)$$

$$(\mathcal{G}_1 \square \mathcal{G}_2) \square \mathcal{G}_3 = \mathcal{G}_1 \square (\mathcal{G}_2 \square \mathcal{G}_3). \quad (3.88)$$

2. Kronecker Product is non-commutative. Usually, $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$. However, they are permutation equivalent, i.e., there exist two permutation matrices \mathbf{P} and \mathbf{Q} such that

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{P}(\mathbf{B} \otimes \mathbf{A})\mathbf{Q}. \quad (3.89)$$

What's more, if \mathbf{A} and \mathbf{B} are square matrices, then $\mathbf{P} = \mathbf{Q}^\top$. As a consequence, $\mathbf{A} \oplus \mathbf{B}$ and $\mathbf{B} \oplus \mathbf{A}$ are generally not equal either, but permutation equivalent:

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{P}(\mathbf{B} \oplus \mathbf{A})\mathbf{Q}. \quad (3.90)$$

This property derives the following properties of the graph Cartesian product.

Theorem 11. $\mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$ and $\mathcal{G}^{(2)} \square \mathcal{G}^{(1)}$ are isomorphic.

Proof. Denote their adjacency matrices by \mathbf{A}_{12} and \mathbf{A}_{21} respectively, then

$$\mathbf{A}_{12} = \mathbf{A}^{(1)} \oplus \mathbf{A}^{(2)},$$

$$\mathbf{A}_{21} = \mathbf{A}^{(2)} \oplus \mathbf{A}^{(1)}.$$

Because \mathbf{A}_{12} and \mathbf{A}_{21} are square, there exists a permutation matrix \mathbf{P} , such that

$$\mathbf{A}_{12} = \mathbf{P}\mathbf{A}_{21}\mathbf{P}^\top.$$

In other words, \mathbf{A}_{12} and \mathbf{A}_{21} differ from each other merely by a vertex reordering, so

$\mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$ and $\mathcal{G}^{(2)} \square \mathcal{G}^{(1)}$ are isomorphic. \square

3. Mixed-product property:

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}), \quad (3.91)$$

where \mathbf{AC} and \mathbf{BD} are ordinary matrix products. This also applied to chained matrix multiplication:

$$(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \mathbf{A}_3)(\mathbf{B}_1 \otimes \mathbf{B}_2 \otimes \mathbf{B}_3) = (\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3) \otimes (\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3). \quad (3.92)$$

4. Transpose, inverse, and the diag operator are distributive over Kronecker product.

$$(\mathbf{A} \otimes \mathbf{B})^\top = \mathbf{A}^\top \otimes \mathbf{B}^\top, \quad (3.93)$$

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}, \quad (3.94)$$

$$\text{diag}(\mathbf{A} \otimes \mathbf{B}) = \text{diag}(\mathbf{A}) \otimes \text{diag}(\mathbf{B}), \quad (3.95)$$

$$\text{diag}^{-1}(\mathbf{a} \otimes \mathbf{b}) = \text{diag}^{-1}(\mathbf{a}) \otimes \text{diag}^{-1}(\mathbf{b}). \quad (3.96)$$

Here, operator $\text{diag} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n$ puts diagonal elements of a matrix into a vector, and

$\text{diag}^{-1} : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ creates a diagonal matrix from the given elements in a vector.

$$\mathbf{a} = \text{diag}(\mathbf{A}), \quad \text{where } a_i = A_{ii}, \quad (3.97)$$

$$\mathbf{A} = \text{diag}^{-1}(\mathbf{a}), \quad \text{where } A_{ii} = a_i \text{ and } A_{ij} = 0 \ \forall i \neq j. \quad (3.98)$$

3.6.3 Laplacian of Cartesian Product

The benefit of considering a graph as the product of other graphs is that its Laplacian and other concepts with Laplacian including Fourier transform and convolution, can be factored into the corresponding concepts of the factor graphs, thus greatly reducing the complexity of both analysis and computation.

Theorem 12 (Laplacian of Cartesian Product). *Denote the adjacency matrix of \mathcal{G} , $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$ by \mathbf{A} , $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and the degree matrix by \mathbf{D} , $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, and the Laplacian Matrix by \mathbf{L} , $\mathbf{L}^{(1)}$, $\mathbf{L}^{(2)}$, respectively. If $\mathcal{G} = \mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$, then*

$$\mathbf{A} = \mathbf{A}^{(1)} \oplus \mathbf{A}^{(2)}, \quad (3.99)$$

$$\mathbf{D} = \mathbf{D}^{(1)} \oplus \mathbf{D}^{(2)}, \quad (3.100)$$

$$\mathbf{L} = \mathbf{L}^{(1)} \oplus \mathbf{L}^{(2)}. \quad (3.101)$$

Proof. Eq. (3.99) is obtained by the definition of graph Cartesian Product. To prove Eq. (3.100), we consider the degree vector \mathbf{d} of graphs, which is equal to the row sum of the adjacency matrix \mathbf{A} :

$$\mathbf{d} \triangleq \text{diag}(\mathbf{D}) = \mathbf{A} \vec{\mathbf{1}}_{n_1 n_2},$$

where $\vec{\mathbf{1}}_{n_1 n_2}$ denotes an all-one vector of length $n_1 n_2$. For simplicity, the subscript, e.g., $n_1 n_2$, may be omitted if vector length can be inferred from the context. By facts of

$\vec{\mathbf{1}}_{n_1 n_2} = \vec{\mathbf{1}}_{n_1} \otimes \vec{\mathbf{1}}_{n_2}$ and mixed-product property of Kronecker product,

$$\begin{aligned}
\mathbf{d} &= \mathbf{A} \left(\vec{\mathbf{1}}_{n_1} \otimes \vec{\mathbf{1}}_{n_2} \right) \\
&= (\mathbf{A}^{(1)} \oplus \mathbf{A}^{(2)}) (\vec{\mathbf{1}} \otimes \vec{\mathbf{1}}) \\
&= (\mathbf{A}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}^{(2)}) (\vec{\mathbf{1}} \otimes \vec{\mathbf{1}}) \\
&= (\mathbf{A}^{(1)} \otimes \mathbf{I}) (\vec{\mathbf{1}} \otimes \vec{\mathbf{1}}) + (\mathbf{I} \otimes \mathbf{A}^{(2)}) (\vec{\mathbf{1}} \otimes \vec{\mathbf{1}}) \\
&= (\mathbf{A}^{(1)} \vec{\mathbf{1}}) \otimes (\mathbf{I} \vec{\mathbf{1}}) + (\mathbf{I} \vec{\mathbf{1}}) \otimes (\mathbf{A}^{(2)} \vec{\mathbf{1}}) \\
&= \mathbf{d}^{(1)} \otimes \vec{\mathbf{1}} + \vec{\mathbf{1}} \otimes \mathbf{d}^{(2)}.
\end{aligned}$$

By the fact that $\text{diag}^{-1}(\cdot)$ is distributive over Kronecker product,

$$\begin{aligned}
\mathbf{D} &= \text{diag}^{-1}(\mathbf{d}) \\
&= \text{diag}^{-1} \left(\mathbf{d}^{(1)} \otimes \vec{\mathbf{1}} \right) + \text{diag}^{-1} \left(\vec{\mathbf{1}} \otimes \mathbf{d}^{(2)} \right) \\
&= \text{diag}^{-1} \left(\mathbf{d}^{(1)} \right) \otimes \text{diag}^{-1}(\vec{\mathbf{1}}) + \text{diag}^{-1}(\vec{\mathbf{1}}) \otimes \text{diag}^{-1} \left(\mathbf{d}^{(2)} \right) \\
&= \mathbf{D}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{D}^{(2)} \\
&= \mathbf{D}^{(1)} \oplus \mathbf{D}^{(2)}.
\end{aligned}$$

Hence, Eq. (3.100) is proved. Next, we consider the Laplacian:

$$\begin{aligned}
\mathbf{L} &= \mathbf{D} - \mathbf{A} \\
&= (\mathbf{D}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{D}^{(2)}) - (\mathbf{A}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}^{(2)}) \\
&= (\mathbf{D}^{(1)} \otimes \mathbf{I} - \mathbf{A}^{(1)} \otimes \mathbf{I}) + (\mathbf{I} \otimes \mathbf{D}^{(2)} - \mathbf{I} \otimes \mathbf{A}^{(2)}) \\
&= (\mathbf{D}^{(1)} - \mathbf{A}^{(1)}) \otimes \mathbf{I} + \mathbf{I} \otimes (\mathbf{D}^{(2)} - \mathbf{A}^{(2)}) \\
&= \mathbf{L}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{L}^{(2)} \\
&= \mathbf{L}^{(1)} \oplus \mathbf{L}^{(2)}.
\end{aligned}$$

As such, Eq. (3.101) is proved. □

3.6.4 Fourier Basis of Product Graph

Eq. (3.101) tells us that the product graph's Laplacian is the Kronecker sum of those of the factor graphs. Moreover, the eigenvalues and eigenvectors of the Laplacian of the product graph also closely relate to those of the factor graphs.

Theorem 13. (*Fourier Basis of Product Graph*) For any eigenpair $\lambda_i^{(1)}, \phi_i^{(1)}$ of $\mathbf{L}^{(1)}$ and eigenpair $\lambda_j^{(2)}, \phi_j^{(2)}$ of $\mathbf{L}^{(2)}$, let

$$\lambda_{ij} \triangleq \lambda_i^{(1)} + \lambda_j^{(2)}, \quad (3.102)$$

$$\phi_{ij} \triangleq \phi_i^{(1)} \otimes \phi_j^{(2)}, \quad (3.103)$$

then ϕ_{ij} is the eigenvector of the Laplacian of the product graph with eigenvalue λ_{ij} , i.e.,

$$\mathbf{L}\phi_{ij} = \lambda_{ij}\phi_{ij}. \quad (3.104)$$

In other words, assume factor graphs' Laplacians have eigen-decomposition

$$\mathbf{L}^{(1)} = \mathbf{\Phi}^{(1)} \mathbf{\Lambda}^{(1)} \mathbf{\Phi}^{(1)\top}, \quad (3.105)$$

$$\mathbf{L}^{(2)} = \mathbf{\Phi}^{(2)} \mathbf{\Lambda}^{(2)} \mathbf{\Phi}^{(2)\top}, \quad (3.106)$$

then the Laplacian of the product graph has eigen-decomposition

$$\mathbf{L} = \mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^\top, \quad (3.107)$$

where

$$\mathbf{\Lambda} \triangleq \mathbf{\Lambda}^{(1)} \oplus \mathbf{\Lambda}^{(2)}, \quad \mathbf{\Phi} \triangleq \mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)}. \quad (3.108)$$

Proof. Notice that

$$\mathbf{L} = \mathbf{L}^{(1)} \oplus \mathbf{L}^{(2)} = \mathbf{L}^{(1)} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{n_1} \otimes \mathbf{L}^{(2)},$$

and

$$\begin{aligned} \mathbf{I}_{n_1} &= \mathbf{\Phi}^{(1)} \mathbf{I}_{n_1} \mathbf{\Phi}^{(1)\top}, \\ \mathbf{I}_{n_2} &= \mathbf{\Phi}^{(2)} \mathbf{I}_{n_2} \mathbf{\Phi}^{(2)\top}. \end{aligned} \tag{3.109}$$

Substitute \mathbf{I}_{n_1} and \mathbf{I}_{n_2} by Eq. (3.109), and apply mix-product property of Kronecker product, we get

$$\begin{aligned} \mathbf{L} &= \mathbf{L}^{(1)} \oplus \mathbf{L}^{(2)} = \mathbf{L}^{(1)} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{n_1} \otimes \mathbf{L}^{(2)} \\ &= (\mathbf{\Phi}^{(1)} \mathbf{\Lambda}^{(1)} \mathbf{\Phi}^{(1)\top}) \otimes (\mathbf{\Phi}^{(2)} \mathbf{I} \mathbf{\Phi}^{(2)\top}) + (\mathbf{\Phi}^{(2)} \mathbf{\Lambda}^{(2)} \mathbf{\Phi}^{(2)\top}) \otimes (\mathbf{\Phi}^{(1)} \mathbf{I} \mathbf{\Phi}^{(1)\top}) \\ &= (\mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)}) (\mathbf{\Lambda}^{(1)} \otimes \mathbf{I}) (\mathbf{\Phi}^{(1)\top} \otimes \mathbf{\Phi}^{(2)\top}) \\ &\quad + (\mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)}) (\mathbf{I} \otimes \mathbf{\Lambda}^{(2)}) (\mathbf{\Phi}^{(1)\top} \otimes \mathbf{\Phi}^{(2)\top}) \\ &= (\mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)}) (\mathbf{\Lambda}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{\Lambda}^{(2)}) (\mathbf{\Phi}^{(1)\top} \otimes \mathbf{\Phi}^{(2)\top}) \\ &= (\mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)}) (\mathbf{\Lambda}^{(1)} \oplus \mathbf{\Lambda}^{(2)}) (\mathbf{\Phi}^{(1)} \otimes \mathbf{\Phi}^{(2)})^\top \\ &\triangleq \mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^\top. \end{aligned}$$

□

This theorem can help us avoid explicitly calculating the eigen-decomposition of the Laplacian of the product graph in most cases. Instead, we may just calculate the eigen-decomposition of the Laplacian of the factor graphs, which reduces the computational complexity from $\mathcal{O}(n_1^3 n_2^3)$ to $\mathcal{O}(n_1^3 + n_2^3)$.

3.6.5 Signals and Fourier Transform on Product Graph

We may represent signals on a product graph as a vector, just as in previous sections, but organizing them in matrix form is more convenient and natural.

Definition 18. (*Signals on Product Graph*) Given graphs $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$ with n_1 nodes and n_2 nodes respectively, a signal defined on their Cartesian product $\mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$ is a matrix $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2}$, where x_{ij} is the value associated to node $(\nu_i^{(1)}, \nu_j^{(2)})$.

To represent a signal in vector form as in previous sections, we may vectorize \mathbf{X} :

$$\text{vec}(\mathbf{X}) = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{21} \\ x_{22} \\ \vdots \\ x_{n_1 n_2} \end{bmatrix}. \quad (3.110)$$

Operator $\text{vec}(\cdot)$ reshapes a matrix into a vector in row-major order, i.e., traversing the matrix row by row. Other material may define $\text{vec}(\cdot)$ in column-major order. The inverse operator $\text{vec}^{-1}(\cdot)$ reshapes a vector back to a matrix of proper size according to the context. With the help of $\text{vec}(\cdot)$, we may write the mixed Kronecker matrix-vector product as

$$(\mathbf{A} \otimes \mathbf{B}) \mathbf{x} = \text{vec}(\mathbf{A} \mathbf{X} \mathbf{B}^\top), \quad (3.111)$$

where $\mathbf{x} = \text{vec}(\mathbf{X})$ and $\mathbf{X} = \text{vec}^{-1}(\mathbf{x})$.

Theorem 14. (*Fourier Transform on Product Graph*) For product graph $\mathcal{G} = \mathcal{G}^{(1)} \square \mathcal{G}^{(2)}$ and a signal \mathbf{X} on \mathcal{G} (in matrix form), its Fourier coefficients \mathbf{c} satisfy

$$\mathbf{c} = \Phi \text{vec}(\mathbf{X}) = \text{vec}(\Phi^{(1)} \mathbf{X} \Phi^{(2)\top}). \quad (3.112)$$

The inverse transform is

$$\mathbf{X} = \Phi^{(1)\top} \text{vec}^{-1}(\mathbf{c}) \Phi^{(2)}. \quad (3.113)$$

This is a direct consequence of the fact of $\Phi = \Phi^{(1)} \otimes \Phi^{(2)}$ and mixed Kronecker matrix-vector product property. Similar to representing signals on the product graph, it is more convenient and natural to organize Fourier coefficients as a matrix.

Definition 19. *The spectrum of a signal on a product graph is a matrix $C \in \mathbb{R}^{n_1 \times n_2}$:*

$$C = \text{vec}^{-1}(\mathbf{c}) = \Phi^{(1)} \mathbf{X} \Phi^{(2)\top}. \quad (3.114)$$

We may recover the signal from the spectrum by

$$\mathbf{X} = \Phi^{(1)\top} C \Phi^{(2)}. \quad (3.115)$$

3.6.6 Convolution on Product Graph

As described in the previous sections, there are two equivalent definitions of graph convolution: one is in the spatial domain, and another is in the spectral domain. Here, we consider the convolution in the spectral domain. According to the convolution theorem, applying a convolution to a signal is equivalent to scaling the signal's Fourier coefficients in the frequency domain. Given a scaling factor $P \in \mathbb{R}^{n_1 \times n_2}$, we can apply it to the spectrum of \mathbf{X} by

$$\mathbf{Z} = \Phi^{(1)\top} (C \circ P) \Phi^{(2)}. \quad (3.116)$$

However, it requires to know the Fourier basis of \mathcal{G}_1 and \mathcal{G}_2 , which are computationally expensive to obtain. To alleviate this problem, we may parameterize the entries of the spectral kernel P as a two-variable polynomial $p(\cdot, \cdot)$ of eigenvalues $\lambda^{(1)}, \lambda^{(2)}$ such that $p_{ij} = p(\lambda_i^{(1)}, \lambda_j^{(2)})$, i.e.,

$$p_{ij} = p(\lambda_i^{(1)}, \lambda_j^{(2)}) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1, k_2} \left(\lambda_i^{(1)}\right)^{k_1} \left(\lambda_j^{(2)}\right)^{k_2}, \quad (3.117)$$

or in matrix form:

$$\mathbf{P} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} (\mathbf{\Lambda}^{(1)})^{k_1} \vec{\mathbf{1}}_{n_1 \times n_2} (\mathbf{\Lambda}^{(2)})^{k_2}, \quad (3.118)$$

where $\vec{\mathbf{1}}_{n_1 \times n_2}$ denotes all-one matrix of shape $n_1 \times n_2$. Substitute \mathbf{P} in Eq. (3.116) by

Eq. (3.118) and after some rearrangement, the graph convolution becomes

$$\begin{aligned} \mathbf{Z} &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} \mathbf{\Phi}^{(1)} \left(\mathbf{C} \circ \left((\mathbf{\Lambda}^{(1)})^{k_1} \vec{\mathbf{1}}_{n_1 \times n_2} (\mathbf{\Lambda}^{(2)})^{k_2} \right) \right) \mathbf{\Phi}^{(2)\top} \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} \mathbf{\Phi}^{(1)} (\mathbf{\Lambda}^{(1)})^{k_1} \left(\mathbf{C} \circ \vec{\mathbf{1}}_{n_1 \times n_2} \right) (\mathbf{\Lambda}^{(2)})^{k_2} \mathbf{\Phi}^{(2)\top} \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} \mathbf{\Phi}^{(1)} (\mathbf{\Lambda}^{(1)})^{k_1} \mathbf{C} (\mathbf{\Lambda}^{(2)})^{k_2} \mathbf{\Phi}^{(2)\top} \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} \mathbf{\Phi}^{(1)} (\mathbf{\Lambda}^{(1)})^{k_1} \mathbf{\Phi}^{(1)\top} \left(\mathbf{\Phi}^{(1)} \mathbf{C} \mathbf{\Phi}^{(2)\top} \right) \mathbf{\Phi}^{(2)\top} (\mathbf{\Lambda}^{(2)})^{k_2} \mathbf{\Phi}^{(2)\top} \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} \left(\mathbf{\Phi}^{(1)} (\mathbf{\Lambda}^{(1)})^{k_1} \mathbf{\Phi}^{(1)\top} \right) \mathbf{X} \left(\mathbf{\Phi}^{(2)} (\mathbf{\Lambda}^{(2)})^{k_2} \mathbf{\Phi}^{(2)\top} \right)^\top \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \theta_{k_1 k_2} (\mathbf{L}^{(1)})^{k_1} \mathbf{X} (\mathbf{L}^{(2)\top})^{k_2}. \end{aligned} \quad (3.119)$$

The second equality is because $(\mathbf{\Lambda}^{(1)})^{k_1}$ and $(\mathbf{\Lambda}^{(2)})^{k_2}$ are diagonal. Eq. (3.119) shows how to perform convolution purely in the spatial domain to avoid calculating the Fourier basis. Parameters $\Theta = \{\theta_{k_1 k_2}\} \in \mathbb{R}^{n_1 \times n_2}$ in Eq. (3.119) are called the (spatial) kernel of this convolution. Similar to Defferrard et al. [14], this spatial convolutional filter is localized. Denote by K_1 and K_2 the largest exponent of \mathbf{L}_1 and \mathbf{L}_2 in the polynomial, i.e., $k_1 > K_1$ or $k_2 > K_2$ implies $\theta_{k_1 k_2} = 0$, then Eq. (3.119) becomes

$$\mathbf{Z} = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} \theta_{k_1 k_2} (\mathbf{L}^{(1)})^{k_1} \mathbf{X} (\mathbf{L}^{(2)\top})^{k_2}. \quad (3.120)$$

In this way, kernel Θ is of size $(K_1 + 1) \times (K_2 + 1)$. The convoluted signal z_{ij} of vertex pair $(\nu_i^{(1)}, \nu_j^{(2)})$ only depends on the neighbourhood of $\nu_i^{(1)}$ within K_1 hops and the neighbourhood of $\nu_j^{(2)}$ within K_2 hops, so the filter is said to be K_1 -localized on \mathcal{G}_1 and K_2 -localized on \mathcal{G}_2 .

3.6.7 Multi-Dimensional Graph

In previous subsections, we introduced the Cartesian product of *two* graphs and the corresponding Laplacian, Fourier basis, and convolution, which is referred to as 2-D graph signal processing. However, most of the results in the 2-D case can be generalized to the Cartesian product of more than two graphs, which is referred to as multi-dimensional graph signal processing.

In this subsection, we consider the chained product as following:

$$\mathcal{G} = \mathcal{G}_1 \square \mathcal{G}_2 \square \dots \square \mathcal{G}_N. \quad (3.121)$$

Since there are N factor graphs, we refer to \mathcal{G} as an N -Dimensional graph or simply an N -D graph.

The order of factors in a Cartesian product does not matter much. Products of the same set of graphs in different orders are isomorphic. For example, Theorem 11 tells

$$\mathcal{G}_1 \square \mathcal{G}_2 \cong \mathcal{G}_2 \square \mathcal{G}_1,$$

where \cong denotes isomorphism.

Theorem 15. *For any permutation i_1, i_2, \dots, i_N of $1, 2, \dots, N$,*

$$\mathcal{G}_1 \square \mathcal{G}_2 \square \dots \square \mathcal{G}_N \cong \mathcal{G}_{i_1} \square \mathcal{G}_{i_2} \square \dots \square \mathcal{G}_{i_N}. \quad (3.122)$$

In other words, the graph Cartesian product is commutative under graph isomorphism equivalence.

Proof. We first prove that if factor graphs are isomorphic, products are also isomorphic.

Lemma 1. *For any graph $\mathcal{G}, \mathcal{G}_1, \mathcal{G}_2$, if $\mathcal{G}_1 \cong \mathcal{G}_2$, then*

$$\mathcal{G} \square \mathcal{G}_1 \cong \mathcal{G} \square \mathcal{G}_2 \cong \mathcal{G}_1 \square \mathcal{G} \cong \mathcal{G}_2 \square \mathcal{G}. \quad (3.123)$$

Consider their adjacency matrices $\mathbf{A}, \mathbf{A}_1, \mathbf{A}_2$. Since $\mathcal{G}_1 \cong \mathcal{G}_2$, there exists a permutation matrix \mathbf{P} such that

$$\mathbf{A}_1 = \mathbf{P}\mathbf{A}_2\mathbf{P}^\top.$$

Now we consider the adjacency matrices of the product graphs. By applying the mix-product property several times, we have

$$\begin{aligned} \mathbf{A} \oplus \mathbf{A}_1 &= \mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}_1 \\ &= \mathbf{A} \otimes (\mathbf{P}\mathbf{P}^\top) + \mathbf{I} \otimes (\mathbf{P}\mathbf{A}_2\mathbf{P}^\top) \\ &= (\mathbf{I} \otimes \mathbf{P})(\mathbf{A} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{P}^\top) + (\mathbf{I} \otimes \mathbf{P})(\mathbf{I} \otimes \mathbf{A}_2)(\mathbf{I} \otimes \mathbf{P}^\top) \\ &= (\mathbf{I} \otimes \mathbf{P})(\mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}_2)(\mathbf{I} \otimes \mathbf{P}^\top) \\ &= (\mathbf{I} \otimes \mathbf{P})(\mathbf{A} \oplus \mathbf{A}_2)(\mathbf{I} \otimes \mathbf{P}^\top). \end{aligned}$$

It is easy to verify that $\mathbf{I} \otimes \mathbf{P}$ is also a permutation matrix, so $\mathcal{G} \square \mathcal{G}_1$ is isomorphic with $\mathcal{G} \square \mathcal{G}_2$. By commutativity, we also have $\mathcal{G}_1 \square \mathcal{G} \cong \mathcal{G}_2 \square \mathcal{G}$. This lemma tells if factor graphs are isomorphic, then products are also isomorphic. Next, we prove that exchanging any two adjacent factors in the product chain will result in isomorphic products.

$$\begin{aligned} &\mathcal{G}_1 \square \cdots \mathcal{G}_i \square \mathcal{G}_j \cdots \mathcal{G}_N \\ &= \mathcal{G}_1 \square \cdots (\mathcal{G}_i \square \mathcal{G}_j) \cdots \mathcal{G}_N \\ &\cong \mathcal{G}_1 \square \cdots (\mathcal{G}_j \square \mathcal{G}_i) \cdots \mathcal{G}_N \\ &= \mathcal{G}_1 \square \cdots \mathcal{G}_j \square \mathcal{G}_i \cdots \mathcal{G}_N \end{aligned}$$

The second line and last line are from the associativity of graph Cartesian product (Section 3.6.2). By the fact that $\mathcal{G}_i \square \mathcal{G}_j$ is isomorphic with $\mathcal{G}_j \square \mathcal{G}_i$ (Theorem 11) and the lemma above, the third line holds.

Finally, we can get any permutation of factor graphs by exchanging adjacent factors for a finite number of times, and the products are kept isomorphic, so the theorem is proved. \square

This theorem enables us to ignore the order of factors in graph Cartesian product, so the below discussion applies to any order of product. We denote the chained Kronecker product and sum by

$$\bigotimes_{i=1}^N \mathbf{A}_i \triangleq \mathbf{A}_1 \otimes \mathbf{A}_2 \cdots \otimes \mathbf{A}_N, \quad (3.124)$$

$$\bigoplus_{i=1}^N \mathbf{A}_i \triangleq \mathbf{A}_1 \oplus \mathbf{A}_2 \cdots \oplus \mathbf{A}_N. \quad (3.125)$$

Theorem 16. Assume $\mathcal{G} = \mathcal{G}_1 \square \mathcal{G}_2 \square \dots \square \mathcal{G}_N$, and denote the degree matrices of factor graphs by $\mathbf{D}^{(i)}$, Laplacians by $\mathbf{L}^{(i)}$, Fourier basis by $\Phi^{(i)}$, eigenvalues by $\Lambda^{(i)}$, and those of the product graph by $\mathbf{D}, \mathbf{L}, \Phi, \Lambda$. Then we have

$$\begin{aligned} \mathbf{L} &= \bigoplus_{i=1}^N \mathbf{L}^{(i)}, & \mathbf{D} &= \bigoplus_{i=1}^N \mathbf{D}^{(i)}, \\ \Phi &= \bigotimes_{i=1}^N \Phi^{(i)}, & \Lambda &= \bigoplus_{i=1}^N \Lambda^{(i)}. \end{aligned}$$

Proof. This is a direct consequence of Theorems 12 and 13. \square

Definition 20. It is natural to organize a signal on N -dimensional graphs as an N -dimensional tensor

$$\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_N}, \quad (3.126)$$

and x_{i_1, i_2, \dots, i_N} is the value associated to vertex $(\nu_{i_1}^{(1)}, \nu_{i_2}^{(2)}, \dots, \nu_{i_N}^{(N)})$.

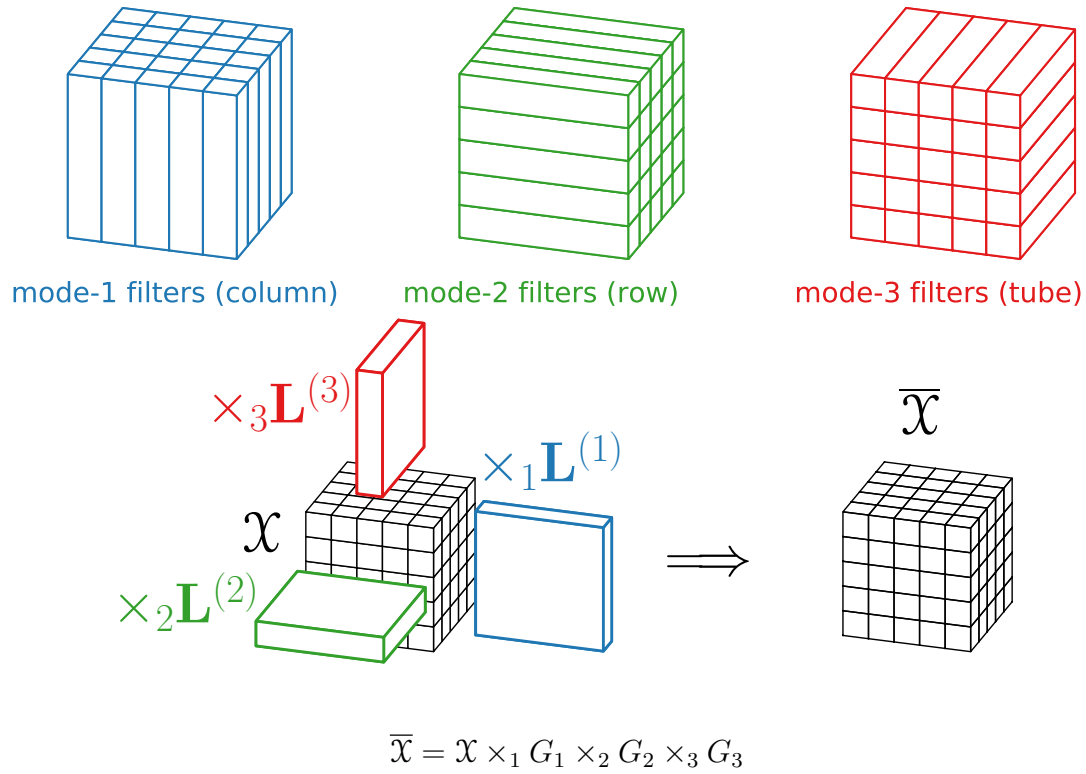


Figure 3.2: Mode- n filters and multi-dimensional graph filtering.

When $N = 1$, \mathcal{X} is a vector; and when $N = 2$, \mathcal{X} is a matrix.

The n -Mode Product Tensors can be multiplied together. The notations are more complex than matrix product, though. For a full treatment of tensor multiplication, see Kolda and Bader [96]. Here we only consider the tensor n -mode product, i.e., multiplying a tensor by a matrix (or a vector) in mode n . Such a product is actually a matrix-vector product applied to *fibers* of a tensor. Tensors' fibers are the higher-dimension analogous to the rows and columns of a matrix. A fiber is a vector defined by fixing the indices of all dimensions/modes but one. The columns of a matrix are mode-1 fibers, and the rows are mode-2 fibers. Similarly, a 3-dimensional tensor \mathcal{X} has three types of fibers: rows, columns, and tubes (Fig. 3.2). The n -mode (matrix) product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ with a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{A}$ and is of size

$I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. Elementwisely, we have

$$(\mathcal{X} \times_n \mathbf{A})_{i_1 i_2 \dots i_{n-1} i_n i_{n+1} \dots i_N} = \sum_j x_{i_1 i_2 \dots i_{n-1} j i_{n+1} \dots i_N} a_{j i_n}. \quad (3.127)$$

Theorem 17. (*Multi-Dimensional Fourier Transform and Inverse Transform*) *The spectrum of signal \mathcal{X} is a tensor \mathcal{C} of the same shape as \mathcal{X} , which can be obtained by*

$$\mathcal{C} = \mathcal{X} \times_1 \Phi^{(1)} \times_2 \Phi^{(2)} \cdots \times_N \Phi^{(N)}, \quad (3.128)$$

where $\mathcal{C} = \{c_{i_1 \dots i_N}\}$, and $c_{i_1 \dots i_N}$ is the coefficient of Fourier base $\phi_{i_1}^{(1)} \otimes \phi_{i_2}^{(2)} \otimes \cdots \otimes \phi_{i_N}^{(N)}$.

Given the spectrum \mathcal{C} , we can recover signal \mathcal{X} by

$$\mathcal{X} = \mathcal{C} \times_1 \Phi^{(1)\top} \times_2 \Phi^{(2)\top} \cdots \times_N \Phi^{(N)\top}. \quad (3.129)$$

Here, we just present the formula of multi-dimensional graph convolution in the spatial domain. One may verify that the N -D graph convolution also obeys the convolution theorem and Parseval's theorem.

Theorem 18. (*Multi-dimensional Graph Convolution*) *Assume $\mathcal{G} = \mathcal{G}_1 \square \mathcal{G}_2 \square \dots \square \mathcal{G}_N$, and denote Laplacians of factor graphs by $\mathbf{L}^{(i)}$. Given a signal \mathcal{X} on \mathcal{G} and a spatial convolution kernel $\Theta \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N}$, applying the convolution kernel to \mathcal{X} results in a new signal $\bar{\mathcal{X}}$ such that*

$$\bar{\mathcal{X}} = \sum_{k_1}^{n_1} \sum_{k_2}^{n_2} \cdots \sum_{k_N}^{n_N} \theta_{k_1 k_2 \dots k_N} \left(\mathcal{X} \times_1 (\mathbf{L}^{(1)})^{k_1} \times_2 (\mathbf{L}^{(2)})^{k_2} \cdots \times_N (\mathbf{L}^{(N)})^{k_N} \right). \quad (3.130)$$

The convolution is of size $K_1 \times K_2 \cdots \times K_N$ and K_i -localized on \mathcal{G}_i .

Table 3.5: Notation Table III.

Notation	Description
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{C}$	Tensors (Calligraphy letters)
$\vec{\mathbf{1}}$ (Bold)	All-one vector
$\vec{\mathbf{0}}$ (Bold)	All-zero vector
\mathbf{I}	Identity matrix
\mathbf{O}	All-zero matrix
$\mathcal{G}_1 \square \mathcal{G}_2$	Graph Cartesian product
$\mathcal{G}_1 \cong \mathcal{G}_2$	Graph isomorphism
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product
$\mathbf{A} \oplus \mathbf{B}$	Kronecker sum
$\mathbf{A} \circ \mathbf{B}$	Hadamard product (element-wise product)
$\mathcal{X} \times_n \mathbf{A}$	n -mode product
$\mathbf{x} = \text{vec}(\mathbf{X})$	Vectorize a matrix
$\mathbf{X} = \text{vec}^{-1}(\mathbf{x})$	Reshape a vector back to a matrix (with proper shape)
$\mathbf{d} = \text{diag}(\mathbf{D})$	Arrange diagonal elements of a matrix as a vector
$\mathbf{D} = \text{diag}^{-1}(\mathbf{d})$	Create a diagonal matrix from elements in vector \mathbf{d}
$\bigotimes_{i=0}^n \mathbf{A}_i$	Chained Kronecker product
$\bigoplus_{i=0}^n \mathbf{A}_i$	Chained Kronecker sum

Chapter Review

This chapter introduces the basic concepts of graph signal processing in detail, from graph Laplacian, graph signals, and signal frequency, to graph convolutional filters, Parseval's theorem, convolution theorem, and low-pass filters. These concepts and theorems serve as the main theoretical tools to analyze various convolution-based methods for learning on graphs in Chapter 4. The last section of this chapter generalizes these concepts to multi-dimensional graphs, which serve as the theoretical background for Chapter 5.

Chapter 4

Learning with 1-D Graph Convolution

Many interesting problems in machine learning have been studied with new deep learning methods. For graph-based semi-supervised learning, a recent important development is graph convolutional networks (GCNs) [15], which nicely integrate local vertex features and graph topology in the convolutional layers. Although the GCN model compares favorably with other state-of-the-art methods, its mechanisms are not clear, and it still requires a considerable amount of labeled data for validation and model selection.

Graph-based methods have been demonstrated as one of the most effective approaches for semi-supervised learning, as they can exploit the connectivity patterns between labeled and unlabeled data samples to improve learning performance. However, existing graph-based methods are either limited in their ability to jointly model graph structures and data features, such as the classical label propagation methods, or require a considerable amount of labeled data for training and validation due to high model complexity, such as the recent neural-network-based methods.

This chapter addresses graph-based semi-supervised learning and unsupervised learning from a graph filtering perspective. Specifically, we propose a graph filtering framework that injects graph similarity into data features by taking them as signals on the graph and applying a low-pass graph filter to extract useful data representations for classification or clustering, where label efficiency can be achieved by conveniently adjusting the strength of the graph filter. Interestingly, this framework unifies two seemingly very different methods — label propagation and graph convolutional networks.

Revisiting them under the graph filtering framework leads to new insights that improve their modeling capabilities and reduce model complexity. Especially, we reveal the fundamental mechanisms of graph convolutional networks via both spatial and spectral analysis. Our spatial analysis shows that the graph convolution of the GCN model is actually a special form of Laplacian smoothing, which is the crucial reason why GCN works, but it also brings up the over-smoothing problem of deep GCN models. Our spectral analysis revisits GCN and classical label propagation methods under a graph filtering framework and shows that what they actually do is to extract useful data representations by a low-pass graph filter.

With the new theoretical insights, we have developed new, efficient, and more powerful models based on graph convolution for semi-supervised and unsupervised learning, including Improved Graph Convolutional Networks (IGCN), Generalized Label Propagation (GLP), and Adaptive Graph Convolution (AGC). Experiments on various semi-supervised classification and clustering tasks on four citation networks and one semi-supervised regression task for zero-shot image recognition validate our findings and proposals.

4.1 Generalized Label Propagation

In this section, we first review label propagation methods (LP) in detail. Next, we cast LP in the context of graph signal processing and identify its three components: signal, filter, and classifier. Then we propose generalized label propagation methods by extending the three components of LP. Finally, we establish the connections between the graph convolution networks and GLP.

Label propagation was a popular method for semi-supervised learning in the early 2000s. The problem of semi-supervised classification is defined as follows. We consider on undirected graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$, where \mathcal{V} is the vertex set with $n = |\mathcal{V}|$ vertices, $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}_+^{n \times n}$ is the adjacency matrix, $\mathbf{X} \in \mathbb{R}^{n \times m}$ is the feature matrix. The adjacency matrix \mathbf{A} encodes edge weights, and is non-negative and symmetric, with $a_{ij} = a_{ji} \geq 0$. The feature matrix $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top$ contains features of each vertex, where $\mathbf{x}_i \in \mathbb{R}^m$ is the feature vector of vertex ν_i . Vertices fall into one of the c classes. The vertex set are partitioned into two part, labeled set $\mathcal{L} = \{\nu_1, \dots, \nu_l\}$ and unlabeled set $\mathcal{U} = \{\nu_{l+1}, \dots, \nu_n\}$ with $|\mathcal{L}| = l$, $|\mathcal{U}| = n - l$. In semi-supervised classification, only the labels of a small subset of vertices are given ($l \ll n$), and the goal is to predict the labels of the rest of the vertices. The labels are usually given in one-hot encodings \mathbf{y}_i , and arranged in a binary label matrix $\mathbf{Y} = \{y_{ij}\} \in \mathbb{R}^{n \times c}$. Label matrix \mathbf{Y} is also split into labeled part $\mathbf{Y}_{\mathcal{L}}$ and unlabeled part $\mathbf{Y}_{\mathcal{U}}$ such that $\mathbf{Y} = \mathbf{Y}_{\mathcal{L}} + \mathbf{Y}_{\mathcal{U}}$. Denote all-zero matrices of proper size by \mathbf{O} , then we have

$$\mathbf{Y} = \begin{bmatrix} \text{---} & \mathbf{y}_1 & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{y}_l & \text{---} \\ \text{---} & \mathbf{y}_{l+1} & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{y}_n & \text{---} \end{bmatrix} \quad \mathbf{Y}_{\mathcal{L}} = \begin{bmatrix} \text{---} & \mathbf{y}_1 & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{y}_l & \text{---} \\ & \mathbf{O}_{(n-l) \times c} & \end{bmatrix} \quad \mathbf{Y}_{\mathcal{U}} = \begin{bmatrix} & & \mathbf{O}_{l \times c} \\ & & \\ \text{---} & \mathbf{y}_{l+1} & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{y}_n & \text{---} \end{bmatrix} \quad (4.1)$$

4.1.1 Label Propagation

Label propagation methods [46, 49, 51] only take the graph weight matrix \mathbf{A} and the labeling matrix \mathbf{Y} as input without using the feature matrix \mathbf{X} . It should be noted that in real-world networks, e.g., citation networks, \mathbf{A} (citation links) naturally exists and contains different information with \mathbf{X} . The objective of the LP method is to find a prediction (embedding) matrix $\mathbf{Z} \in \mathbb{R}^{n \times c}$ of the same size as the labeling matrix \mathbf{Y} by minimizing both fitting error and manifold regularization penalty:

$$\mathbf{Z} = \arg \min_{\mathbf{Z}} \left\{ \underbrace{\|\mathbf{Z} - \mathbf{Y}_{\mathcal{L}}\|_F^2}_{\text{fitting error}} + \alpha \underbrace{\text{Tr}(\mathbf{Z}^\top \mathbf{L} \mathbf{Z})}_{\text{regularization}} \right\}, \quad (4.2)$$

where the graph Laplacian \mathbf{L} is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ with $d_i = \sum_j a_{ij}$ and $\mathbf{D} = \text{diag}(d_i)$ as the degree matrix. We can also use the row normalized graph Laplacian $\mathbf{L}_r = \mathbf{D}^{-1} \mathbf{L}$ or the symmetrically normalized graph Laplacian $\mathbf{L}_s = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}}$. In Eq. (4.2), the fitting term enforces the embedding matrix \mathbf{Z} to agree with the label matrix $\mathbf{Y}_{\mathcal{L}}$, while the regularization term enforces each column of \mathbf{Z} to be smooth along the edges. The scalar α is a balancing parameter. A closed-form solution to the unconstrained quadratic optimization problem can be obtained by taking the derivative of the objective function and setting it to zero:

$$\mathbf{Z} = (\mathbf{I} + \alpha \mathbf{L})^{-1} \mathbf{Y}_{\mathcal{L}}. \quad (4.3)$$

With the learned embedding matrix \mathbf{Z} , each unlabeled vertex v_i is then classified by simply finding the largest element in $Z(i, :)$ (some normalization may be applied to the columns of \mathbf{Z} first [46]).

4.1.2 Rethink Label Propagation

From the perspective of graph filtering, we show that LP comprises three components: signal, filter, and classifier. We can see from Eq. (4.3) that the input signal matrix of LP is simply the label matrix $\mathbf{Y}_{\mathcal{L}}$, which has c channels, and each column $\mathbf{Y}_{\mathcal{L}}(:, i)$ can be considered as a graph *signal*.

The convolutional *filter* of LP is:

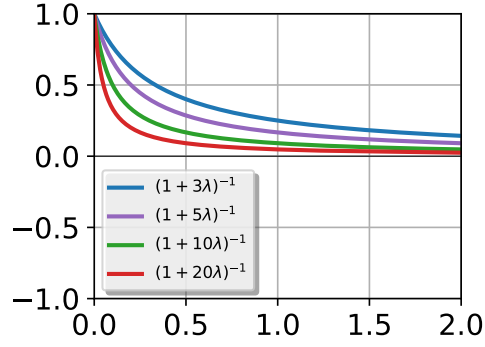
$$p_{\text{ap}}(\mathbf{L}) = (\mathbf{I} + \alpha\mathbf{L})^{-1} = \Phi(\mathbf{I} + \alpha\Lambda)^{-1}\Phi^{-1}, \quad (4.4)$$

with the frequency response function:

$$p_{\text{ap}}(\lambda) = \frac{1}{1 + \alpha\lambda}. \quad (4.5)$$

We name it the AP filter, as it encodes the absorption probability of random walks on graphs [97].

Note that this also holds for the normalized graph Laplacians. As shown in Fig. 4.1, the frequency response function of LP is low-pass. For any $\alpha > 0$, $p_{\text{ap}}(\lambda)$ is near 1 when λ is close to 0 and $p_{\text{ap}}(\lambda)$ decreases and approaches 0 as λ increases. Apply the filter on the i -th channel of signal $\mathbf{Y}_{\mathcal{L}}(:, i)$, it will produce a smooth signal $\mathbf{Z}(:, i)$ in which vertices in the same class have similar values and vertices in class i have larger values than others under the cluster assumption. The balancing parameter α controls the degree of manifold



$$p_{\text{ap}}(\lambda) = (1 + \alpha\lambda)^{-1}$$

Figure 4.1: Response functions of AP filters.

regularization. When α increases, the filter becomes more low-pass (Fig. 4.1) and will produce smoother embeddings. Finally, LP applies a nonparametric classifier on the embeddings to classify the unlabeled vertices.

$$\mathbf{Y} = \text{softmax}(\mathbf{Z}) \quad (4.6)$$

$$\text{predict}(\mathbf{x}_i) = \underset{j}{\text{argmax}} z_{ij} \quad (4.7)$$

Generalized Label Propagation

The above analysis shows that LP only takes the given graph W and the label matrix Y into account but excludes the feature matrix X . This is one of its major limitations in dealing with datasets that provide both W and X , e.g., citation networks. Here, we propose generalized label propagation (GLP) methods by naturally extending the three components of LP.

We propose a generalized label propagation (GLP) framework by naturally generalizing the three components of LP:

- Signal: Use the feature matrix X instead of the labeling matrix Y as the input signal.
- Filter: The filter G can be any low-pass, linear, shift-invariant filter.
- Classifier: The classifier can be any classifier trained on the embeddings of labeled vertices.

The low-pass, linear, shift-invariant graph filter G is applied to the feature matrix X to obtain a smooth feature matrix $\bar{X} \in \mathbb{R}^{n \times m}$:

$$\bar{X} = GX. \quad (4.8)$$

The next step is to train a supervised classifier (e.g., multilayer perceptron, convolutional neural networks, support vector machines, etc.) with the filtered features of labeled data and then apply the classifier to the filtered features of unlabeled data to predict their labels.

GLP combines graph and feature information naturally and seamlessly by Eq. (4.8) and allows taking advantage of a powerful supervised classifier. The rationale behind GLP is to learn representative feature vectors of each class to ease the downstream classification task. After being filtered by G , vertices in the same class are expected to have more similar and representative features (as shown in Fig. 4.2), making it much easier to train a good classifier with only a small set of samples.

Consider an extreme case where each class is a connected component of the graph. In such a case, we can learn perfect features by an extremely low-pass filter G , whose spectrum $p(\cdot)$ is unit impulse function, i.e., $p(0) = 1$ and $p(\lambda) = 0$ if $\lambda \neq 0$. We can compute $G = \Phi p(\Lambda) \Phi^{-1}$ in the spatial domain. In particular, $G_{ij} = \frac{1}{l_k}$ if v_i and v_j are of

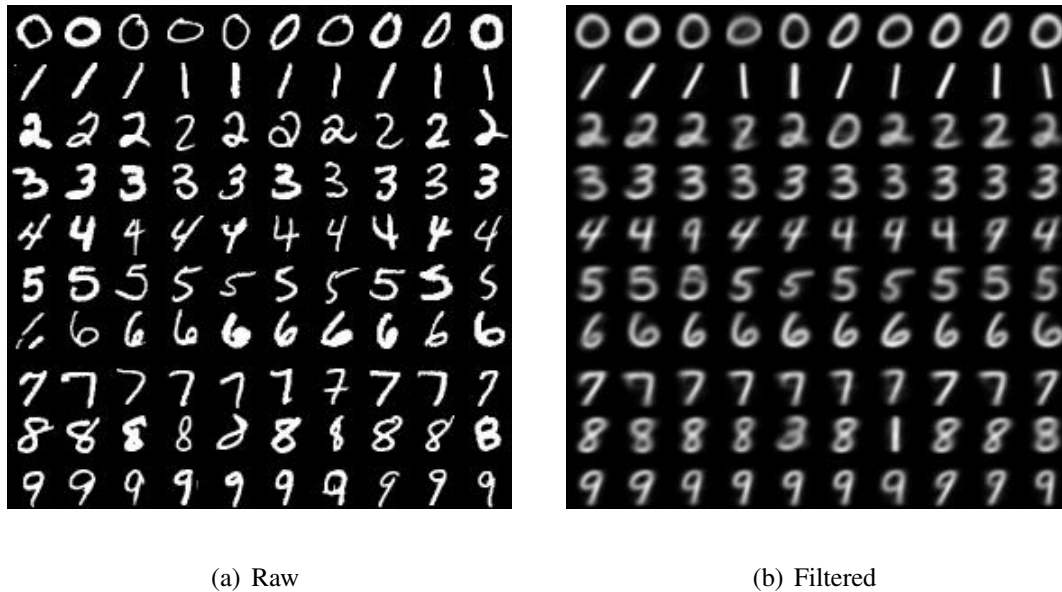


Figure 4.2: Visualizing raw and filtered features.

the same class, otherwise $G_{ij} = 0$, where l_k is the number of labeled samples in class k . After being filtered by G , vertices in the same class will have an identical feature vector, which is the class mean. Then any classifier that can correctly classify the labeled data will achieve 100% accuracy on the unlabeled data, and only one labeled example per class is needed to train the classifier.

4.2 Revisit and Improve Graph Convolutional Networks

The recently proposed graph convolutional networks (GCN) [15] have demonstrated superior performance in semi-supervised learning and attracted much attention. The GCN model consists of three steps. First, a so-called renormalization trick is applied on the adjacency matrix A by adding a self-loop to each vertex, resulting in a new adjacency matrix $\tilde{A} = A + I$ with the degree matrix $\tilde{D} = D + I$, which is then symmetrically

normalized as $\tilde{\mathbf{A}}_s = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$. Second, define the layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma \left(\tilde{\mathbf{A}}_s \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right), \quad (4.9)$$

where $\mathbf{H}^{(l)}$ is the matrix of activations fed to the l -th layer and $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{W}^{(l)}$ is the trainable weights, and σ is an activation function, e.g., $\text{ReLU}(\cdot) = \max(0, \cdot)$. The graph convolution is defined as multiplying the input of each layer with the renormalized adjacency matrix \mathbf{A}_s from the left, i.e., $\tilde{\mathbf{A}}_s \mathbf{H}^{(l)}$. The convoluted features are then fed into a projection matrix $\mathbf{W}^{(l)}$. Third, stack two layers up and apply a softmax function on the output features to produce a prediction matrix:

$$\mathbf{Z} = \text{softmax} \left(\tilde{\mathbf{A}}_s \text{ReLU} \left(\tilde{\mathbf{A}}_s \mathbf{X} \mathbf{W}^{(0)} \right) \mathbf{W}^{(1)} \right), \quad (4.10)$$

and then train the model with the cross-entropy loss on labeled samples.

4.2.1 Spatial Analysis

In this subsection, we demystify the GCN model for semi-supervised learning. In particular, we show that the graph convolution of the GCN model is simply a special form of Laplacian smoothing, which mixes the features of a vertex and its nearby neighbors. The smoothing operation makes the features of vertices in the same cluster similar, thus greatly easing the classification task, which is the key reason why GCNs work so well. However, it also brings potential concerns of over-smoothing. If a GCN is deep with many convolutional layers, the output features may be over-smoothed, and vertices from different clusters may become indistinguishable. Also, adding more layers to a GCN will make it much more difficult to train.

However, a shallow GCN model such as the two-layer GCN used in Kipf and Welling

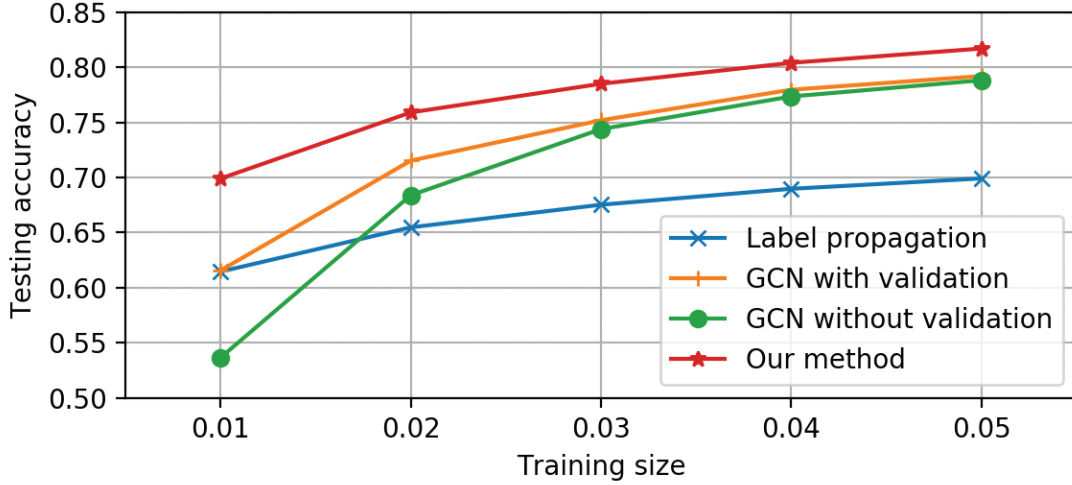


Figure 4.3: Performance comparison of GCNs, label propagation, and our method for semi-supervised classification on the Cora citation network.

[15] has its own limits. Besides it requires many additional labels for validation, it also suffers from the localized nature of the convolutional filter. When only a few labels are given, a shallow GCN cannot effectively propagate the labels to the entire data graph. As illustrated in Fig. 4.3, the performance of GCNs drops quickly as the training size shrinks, even for the one with 500 additional labels for validation.

Why GCNs Work

To understand the reasons why GCNs work so well, we compare them with the simplest fully-connected networks (FCNs), where the layer-wise propagation rule is

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{H}^{(l)}\mathbf{W}^{(l)}). \quad (4.11)$$

Clearly, the only difference between a GCN and a FCN is the graph convolution matrix $\tilde{\mathbf{A}}_s = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ (Eq. (4.10)) applied on the left of the feature matrix \mathbf{X} . To see the impact of the graph convolution, we tested the performances of GCNs and FCNs for

Table 4.1: GCNs vs. Fully-connected networks

	One-layer FCN	Two-layer FCN
FCN	0.530860	0.559260
GCN	0.707940	0.798361

semi-supervised classification on the Cora citation network with 20 labels in each class. The results can be seen in Table 4.1. Surprisingly, even a one-layer GCN outperformed a one-layer FCN by a very large margin.

Laplacian Smoothing Let us first consider a one-layer GCN. It actually contains two steps. 1) Generating a new feature matrix \mathbf{Z} from \mathbf{X} by applying the graph convolution:

$$\mathbf{Z} = \tilde{\mathbf{A}}_s \mathbf{X}. \quad (4.12)$$

2) Feeding the new feature matrix \mathbf{Y} to a fully connected layer. Clearly, graph convolution is the key to the huge performance gain.

Let us examine the graph convolution carefully. Suppose that we add a self-loop to each vertex in the graph, then the adjacency matrix of the new graph is $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. The Laplacian smoothing [93] on each channel of the input features is defined as

$$\mathbf{z}_i = (1 - \gamma)\mathbf{x}_i + \frac{\gamma}{d_i} \sum_j \tilde{a}_{ij} \mathbf{z}_j \quad (\text{for } 1 \leq i \leq n), \quad (4.13)$$

where $0 < \gamma \leq 1$ is a parameter that controls the weighting between the center vertex's features and its neighbors' features.. We can write the Laplacian smoothing in matrix form:

$$\mathbf{Z} = \mathbf{X} - \gamma \tilde{\mathbf{L}}_r \mathbf{X} = (\mathbf{I} - \gamma \tilde{\mathbf{L}}_r) \mathbf{X}, \quad (4.14)$$

where $\tilde{L}_r = I - \tilde{A}_r$, and $\tilde{A}_r = \tilde{D}^{-1}\tilde{A}$. By letting $\gamma = 1$, i.e., only using the neighbors' features, we have $Z = \tilde{A}_r X$, which is the standard form of Laplacian smoothing.

Now if we replace the row-normalized Laplacian \tilde{L}_r with the symmetrically normalized Laplacian \tilde{L}_s and let $\gamma = 1$, we have $Z = \tilde{A}_s X$, which is exactly the graph convolution in Eq. (4.12). We thus call the graph convolution a special form of Laplacian smoothing – symmetric Laplacian smoothing. Note that here the smoothing still includes the current vertex's features, as each vertex has a self-loop and is its own neighbor.

The Laplacian smoothing computes the new features of a vertex as the weighted average of itself and its neighbors. Since vertices in the same cluster tend to be densely connected, the smoothing makes their features similar, which makes the subsequent classification task much more manageable. As we can see from Table 4.1, applying the smoothing only once has already led to a huge performance gain.

Multi-layer Structure. We can also see from Table 4.1 that while the 2-layer FCN only slightly improves over the 1-layer FCN, the 2-layer GCN significantly improves over the 1-layer GCN by a large margin. This is because applying smoothing again on the activations of the first layer makes the output features of vertices in the same cluster more similar and further eases the classification task.

When GCNs Fail

We have shown that the graph convolution is essentially a type of Laplacian smoothing. A natural question is how many convolutional layers should be included in a GCN? it is certainly not that the more the better. On the one hand, a GCN with many layers is

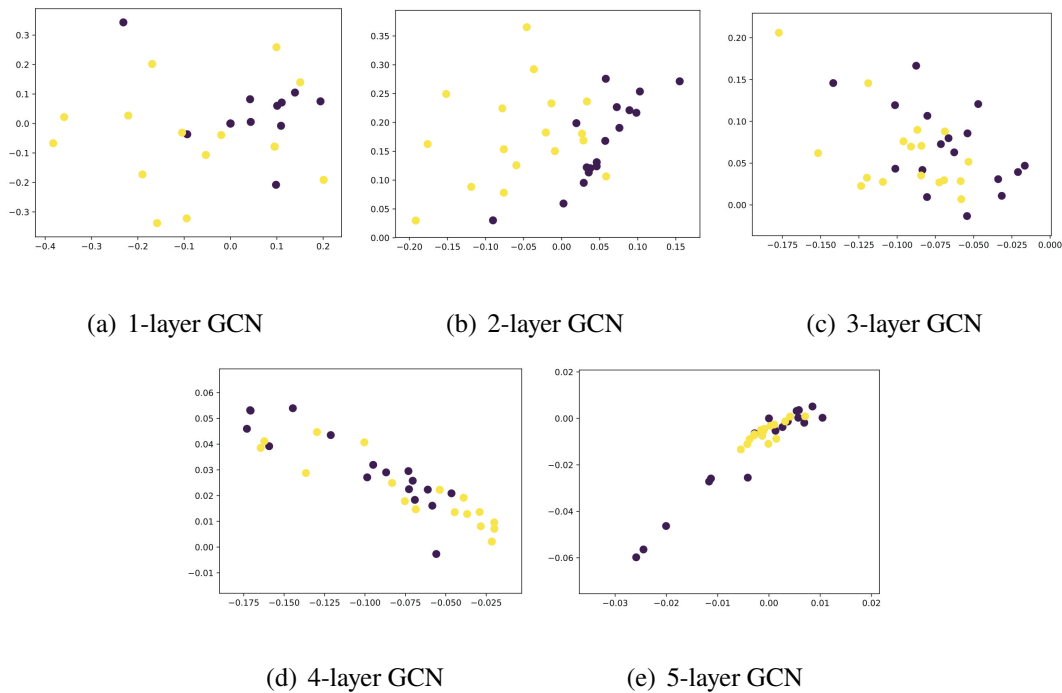


Figure 4.4: Vertex embeddings of Zachary's karate club network with GCNs.

difficult to train. On the other hand, repeatedly applying Laplacian smoothing may mix the features of vertices from different clusters and make them indistinguishable. In the following, we illustrate this point with a popular dataset.

We apply GCNs with different numbers of layers to the Zachary's karate club dataset [98], which has 34 vertices of two classes and 78 edges. The GCNs are untrained with the weight parameters initialized randomly as in Glorot and Bengio [99]. The dimension of the hidden layers is 16, and the dimension of the output layer is 2. The feature vector of each vertex is one-hot. The outputs of each GCN are plotted as two-dimensional points in Fig. 4.4. We can observe the impact of the graph convolution (Laplacian smoothing) on this small dataset. Apply the smoothing once, and points are not well-separated (Fig. 4.4(a)). Apply the smoothing twice, and the points from the two classes are separated relatively well. Apply the smoothing again and again, and the points are mixed (Fig. 4.4(c-e)). As this is a small dataset and vertices between two classes have

quite many connections, the mixing happens quickly. The mixing speed depends on the spectral gap of the filter (e.g., $\tilde{\mathbf{A}}_s$). Generally, the larger the spectral gap, the more quickly the mixing happens. A graph with dense connections tends to have an $\tilde{\mathbf{A}}_s$ with a large spectral gap.

In the following, we will prove that by repeatedly applying Laplacian smoothing many times, the features of vertices within each connected component of the graph will converge to the same values. For the case of symmetric Laplacian smoothing, they will converge to be proportional to the square root of the vertex degree.

Suppose that a graph \mathcal{G} has k connected components $\{\mathcal{C}_i\}_{i=1}^k$, and the indication vector for the i -th component is denoted by $\mathbf{1}^{(i)} \in \mathbb{R}^n$. This vector indicates whether a vertex is in component \mathcal{C}_i , i.e.,

$$\vec{\mathbf{1}}_j^{(i)} = \begin{cases} 1, & v_j \in \mathcal{C}_i \\ 0, & v_j \notin \mathcal{C}_i \end{cases} \quad (4.15)$$

Theorem 19. *If a graph has no bipartite components, then for any $\mathbf{x} \in \mathbb{R}^n$, and $\alpha \in (0, 1]$,*

$$\begin{aligned} \lim_{l \rightarrow +\infty} (\mathbf{I} - \alpha \mathbf{L}_r)^l \mathbf{x} &= [\vec{\mathbf{1}}^{(1)}, \vec{\mathbf{1}}^{(2)}, \dots, \vec{\mathbf{1}}^{(k)}] \boldsymbol{\theta}_1, \\ \lim_{l \rightarrow +\infty} (\mathbf{I} - \alpha \mathbf{L}_s)^l \mathbf{x} &= \mathbf{D}^{1/2} [\vec{\mathbf{1}}^{(1)}, \vec{\mathbf{1}}^{(2)}, \dots, \vec{\mathbf{1}}^{(k)}] \boldsymbol{\theta}_2, \end{aligned}$$

where $\boldsymbol{\theta}_1 \in \mathbb{R}^k$, $\boldsymbol{\theta}_2 \in \mathbb{R}^k$, i.e., they converge to a linear combination of $\{\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ and $\{\mathbf{D}^{1/2} \vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ respectively.

Proof. Recall properties of spectrums of \mathbf{L}_r and \mathbf{L}_s in Theorem 4. \mathbf{L}_r and \mathbf{L}_s have the same n eigenvalues (by multiplicity) with different eigenvectors. If a graph has no bipartite components, the eigenvalues all fall in $[0, 2)$. The eigenspaces of \mathbf{L}_r and \mathbf{L}_s corresponding to eigenvalue 0 are spanned by $\{\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ and $\{\mathbf{D}^{1/2} \vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ respectively.

For $\alpha \in (0, 1]$, the eigenvalues of $(\mathbf{I} - \alpha\mathbf{L}_r)$ and $(\mathbf{I} - \alpha\mathbf{L}_s)$ all fall into $(-1, 1]$, and the eigenspaces of eigenvalue 1 are spanned by $\{\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ and $\{\mathbf{D}^{1/2}\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ respectively. Since the absolute value of all eigenvalues of $(\mathbf{I} - \alpha\mathbf{L}_r)$ and $(\mathbf{I} - \alpha\mathbf{L}_s)$ are less than or equal to 1, after repeatedly multiplying them from the left, the result will converge to the linear combination of eigenvectors of eigenvalue 1, i.e. the linear combination of $\{\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ and $\{\mathbf{D}^{1/2}\vec{\mathbf{1}}^{(i)}\}_{i=1}^k$ respectively. \square

Note that since an extra self-loop is added to each vertex, there is no bipartite component in the graph. Based on the above theorem, over-smoothing will make the features indistinguishable and hurt the classification accuracy.

The above analysis raises potential concerns about stacking many convolutional layers in a GCN. Besides, a deep GCN is much more difficult to train. In fact, the GCN used in Kipf and Welling [15] is a 2-layer GCN. However, since the graph convolution is a localized filter – a linear combination of the feature vectors of adjacent neighbors, a shallow GCN cannot sufficiently propagate the label information to the entire graph with only a few labels. As shown in Fig. 4.3, the performance of GCNs (with or without validation) drops quickly as the training size shrinks. In fact, the accuracy of GCNs decreases much faster than the accuracy of label propagation. Since label propagation only uses graph information while GCNs utilize both graph structure and vertex features, it reflects the inability of the GCN model to explore the global graph structure.

Another problem with the GCN model in Kipf and Welling [15] is that it requires an additional validation set for early stopping in training, which is essentially using the prediction accuracy on the validation set for model selection. If we optimize a GCN on the training data without using the validation set, it will have a significant drop in

performance. As shown in Fig. 4.3, the performance of the GCN without validation drops much sharper than the GCN with validation. In Kipf and Welling [15], the authors used an additional set of 500 labeled data for validation, which is much more than the total number of training data. This is certainly undesirable as it defeats the purpose of semi-supervised learning. Furthermore, it makes comparing GCNs with other methods unfair, as other methods, such as label propagation, may not need the validation data at all.

4.2.2 Spectral Analysis

In this subsection, we analyze GCN in the frequency domain and explain its implicit design features, including the choice of the normalized graph Laplacian and the renormalization trick on the adjacency matrix.

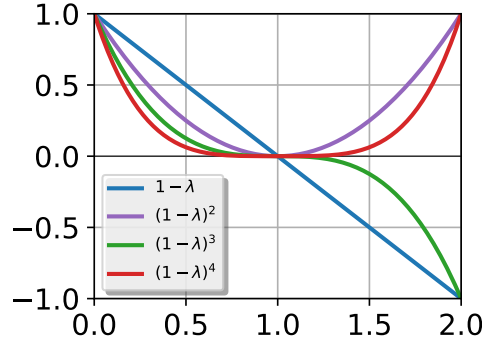
GCN conducts graph filtering on $\mathbf{H}^{(t)}$ in each layer with the filter $\tilde{\mathbf{A}}_s$. We have $\tilde{\mathbf{A}}_s = \mathbf{I} - \tilde{\mathbf{L}}_s$, where $\tilde{\mathbf{L}}_s$ is the symmetrically normalized graph Laplacian of the graph $\tilde{\mathbf{A}}$. Eigen-decompose $\tilde{\mathbf{L}}_s$ as $\tilde{\mathbf{L}}_s = \Phi \tilde{\Lambda} \Phi^{-1}$, then the filter becomes

$$\tilde{\mathbf{A}}_s = \mathbf{I} - \tilde{\mathbf{L}}_s = \Phi(\mathbf{I} - \tilde{\Lambda})\Phi^{-1}, \quad (4.16)$$

with a frequency response function

$$p(\tilde{\lambda}) = 1 - \tilde{\lambda}. \quad (4.17)$$

Clearly, as shown in Fig. 4.5, this function is linear and low-pass when $\lambda \in [0, 1]$, but not when $\lambda \in [1, 2]$.



$$p_{\text{rnm}}(\lambda) = (1 - \lambda)^k$$

Figure 4.5: Response functions of the renormalization filters (RNM).

Why Two-Layer structure? If we perform all the graph convolutions in Eq. (4.10) in the input layer by exchanging the renormalized adjacency matrix $\tilde{\mathbf{A}}_s$ in the second layer with the internal ReLU function, GCN is a special case of GLP, where the input signal matrix is X , the filter is $\tilde{\mathbf{A}}_s^2$, and the classifier is a two-layer multi-layer perceptron (MLP). One can also see that GCN stacks two convolutional layers because $\tilde{\mathbf{A}}_s^2$ is more low-pass than $\tilde{\mathbf{A}}_s$, which can be seen from Fig. 4.5 that $(1 - \lambda)^2$ is sort of more low-pass than $(1 - \lambda)$ by suppressing the large eigenvalues harder.

Why Use Normalized Graph Laplacian? GCN uses the normalized Laplacian L_s because the eigenvalues of L_s fall into $[0, 2]$ (Theorem 4), while those of the unnormalized Laplacian L are in $[0, +\infty]$. If using L , the frequency response in Eq. (4.17) will amplify eigenvalues in $[2, +\infty]$, which will introduce noise and undermine performance.

Why the Renormalization Trick Works? We illustrate the effect of the renormalization trick used in GCN in Fig. 4.6, where the frequency responses on the eigenvalues of L_s and \tilde{L}_s on the Cora citation network are plotted respectively. By adding a self-loop to

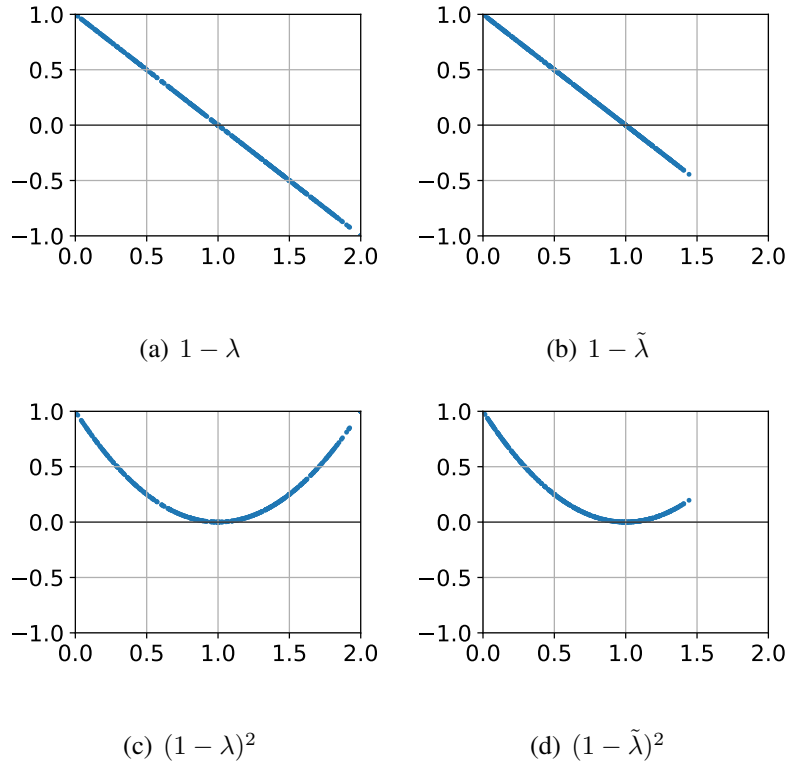


Figure 4.6: Effect of the renormalization trick. Left two figures plot points $(\lambda_i, p(\lambda_i))$. Right two figures plot points $(\tilde{\lambda}_i, p(\tilde{\lambda}_i))$.

each vertex, we can see that the range of eigenvalues shrinks from $[0, 2]$ to $[0, 1.5]$, which avoids amplifying eigenvalues near 2 and reduces noise. Hence, although the response function $(1 - \lambda)^k$ is not completely low-pass, the renormalization trick shrinks the range of eigenvalues and makes $\tilde{\mathbf{L}}_s$ resemble a low-pass filter. It can be proved that if the largest eigenvalue of \mathbf{L}_s is λ_m , then all the eigenvalues of $\tilde{\mathbf{L}}_s$ are no larger than $\frac{d_m}{d_m + 1} \lambda_m$, where d_m is the largest degree of all vertices.

Theorem 20. *If the largest eigenvalue of \mathbf{L}_s is λ_m , then the largest eigenvalue of $\tilde{\mathbf{L}}_s$ is no more than $\frac{d_m}{d_m + 1} \lambda_m$, where $d_m = \max_i d_i$ is the graph degree (the largest degree of vertices).*

Proof. According to the property of eigenvalue,

$$\begin{aligned}
\tilde{\lambda}_m &= \max_{\mathbf{x}} \frac{\mathbf{x}^\top \tilde{\mathbf{L}}_s \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} \\
&= \max_{\mathbf{x}} \frac{\mathbf{x}^\top (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} \mathbf{L} (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} \\
&= \max_{\mathbf{x}} \frac{\mathbf{x}^\top (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} \mathbf{D}^{\frac{1}{2}} \mathbf{L}_s \mathbf{D}^{\frac{1}{2}} (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}
\end{aligned} \tag{4.18}$$

Let $\mathbf{z} = \mathbf{D}^{\frac{1}{2}} (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}} \mathbf{x}$, then

$$\tilde{\lambda}_m = \max_{\mathbf{z}} \frac{\mathbf{z}^\top \mathbf{L}_s \mathbf{z}}{\mathbf{z}^\top (\mathbf{D} + \mathbf{I}) \mathbf{D}^{-1} \mathbf{z}} \tag{4.19}$$

$$\begin{aligned}
&= \max_{\mathbf{z}} \frac{\mathbf{z}^\top \mathbf{L}_s \mathbf{z}}{\sum_i \frac{d_i+1}{d_i} z_i^2} \\
&\leq \max_{\mathbf{z}} \frac{\mathbf{z}^\top \mathbf{L}_s \mathbf{z}}{\frac{d_m+1}{d_m} \sum_i z_i^2}
\end{aligned} \tag{4.20}$$

$$= \frac{d_m}{d_m+1} \max_{\mathbf{z}} \frac{\mathbf{z}^\top \mathbf{L}_s \mathbf{z}}{\mathbf{z}^\top \mathbf{z}} \tag{4.21}$$

$$= \frac{d_m}{d_m+1} \lambda_m$$

□

4.2.3 Improved Graph Convolutional Networks

A notable drawback of the current GCN model is that one cannot easily control filter strength. One has to stack multiple layers to increase filter strength and produce smoother features. However, since in each layer the convolution is coupled with a projection matrix by the ReLU, stacking many layers will introduce many trainable parameters. This may lead to severe overfitting when the label rate is low.

To fix this, we propose an improved GCN model (IGCN) by replacing the filter $\tilde{\mathbf{A}}_s$ with

$\tilde{\mathbf{A}}_s^k$:

$$\mathbf{Z} = \text{softmax} \left(\tilde{\mathbf{A}}_s^k \text{ReLU} \left(\tilde{\mathbf{A}}_s^k \mathbf{X} \mathbf{W}^{(0)} \right) \mathbf{W}^{(1)} \right). \tag{4.22}$$

We call $p_{\text{rnm}}(\tilde{\mathbf{L}}_s) = \tilde{\mathbf{A}}_s^k$ the renormalization (RNM) filter, with frequency response function

$$p_{\text{rnm}}(\tilde{\lambda}) = (I - \tilde{\lambda})^k. \quad (4.23)$$

IGCN can achieve label efficiency by using the exponent k to adjust the filter strength conveniently. This way, it can maintain a shallow structure with a reasonable number of trainable parameters to avoid overfitting.

4.3 Filter Design and Computation

Here, we give out several available low-pass filters and efficient ways to calculate them.

Absorption Probability

The absorption probability (AP) filter is the one used in LP:

$$\begin{aligned} p_{\text{ap}}(\mathbf{L}_s) &= (\mathbf{I} + \alpha \mathbf{L}_s)^{-1}, \\ p_{\text{ap}}(\mathbf{L}_r) &= (\mathbf{I} + \alpha \mathbf{L}_r)^{-1}, \\ p_{\text{ap}}(\mathbf{L}) &= (\mathbf{I} + \alpha \mathbf{L})^{-1}. \end{aligned} \quad (4.24)$$

We call p_{ap} the absorption probability filter, as it has an interpretation in absorption probability as shown in Wu et al. [97]. We have shown p_{ap} is low-pass in Section 4.1.2. However, the computation of $p_{\text{ap}}(\mathbf{L})$ involves matrix inversion, which is computationally expensive with complexity $\mathcal{O}(n^3)$. Fortunately, we can circumvent this problem by approximating p_{ap} by polynomials.

Theorem 21. *Let $\mathbf{L}_s, \mathbf{L}_r, \mathbf{L}$ be symmetrically normalized, row-normalized, and unnormalized Laplacian, respectively. For any $\alpha > 0$, the AP filters based on them can be*

approximated by

$$(\mathbf{I} + \alpha \mathbf{L}_s)^{-1} = \frac{1}{1 + \alpha} \sum_{k=0}^{+\infty} \left[\frac{\alpha}{1 + \alpha} \mathbf{A}_s \right]^k, \quad (4.25)$$

$$(\mathbf{I} + \alpha \mathbf{L}_r)^{-1} = \frac{1}{1 + \alpha} \sum_{k=0}^{+\infty} \left[\frac{\alpha}{1 + \alpha} \mathbf{A}_r \right]^k, \quad (4.26)$$

$$(\mathbf{I} + \alpha \mathbf{L})^{-1} = \sum_{k=0}^{\infty} \left[\left(\frac{1}{\alpha} \mathbf{I} + \mathbf{D} \right)^{-1} \mathbf{A} \right]^k (\mathbf{I} + \alpha \mathbf{D})^{-1}. \quad (4.27)$$

Proof. Consider sum of geometric series $1, t, t^2, \dots$ in \mathbb{R} :

$$\frac{1}{1 - t} = \sum_{k=0}^{\infty} t^k, \quad (|t| < 1). \quad (4.28)$$

This equation can also be applied to any matrix \mathbf{M} , whose eigenvalues $\lambda_1, \dots, \lambda_n$ are all of magnitude less than 1, i.e., $(|\lambda_i|) < 1$ for all i .

$$\{\text{eigenvalues of } \mathbf{M}\} \subset (-1, 1) \implies (\mathbf{I} - \mathbf{M})^{-1} = \sum_{k=0}^{\infty} \mathbf{M}^k. \quad (4.29)$$

Let $\mathbf{M} := \frac{\alpha}{1 + \alpha} (\mathbf{I} - \mathbf{L}_s) = \frac{\alpha}{1 + \alpha} \mathbf{A}_s$, whose eigenvalues are apparently of magnitude less than 1, then we get

$$\begin{aligned} \left(\frac{1}{1 + \alpha} \mathbf{I} + \frac{\alpha}{1 + \alpha} \mathbf{L}_s \right)^{-1} &= \sum_{k=0}^{\infty} \left[\frac{\alpha}{1 + \alpha} \mathbf{A}_s \right]^k \\ \implies (\mathbf{I} + \alpha \mathbf{L}_s)^{-1} &= \frac{1}{1 + \alpha} \sum_{k=0}^{\infty} \left[\frac{\alpha}{1 + \alpha} \mathbf{A}_s \right]^k. \end{aligned}$$

Similarly, Let $\mathbf{M} := \frac{\alpha}{1 + \alpha} (\mathbf{I} - \mathbf{L}_r) = \frac{\alpha}{1 + \alpha} \mathbf{A}_r$, whose eigenvalues are also of magnitude less than 1, then we get

$$(\mathbf{I} + \alpha \mathbf{L}_r)^{-1} = \frac{1}{1 + \alpha} \sum_{k=0}^{\infty} \left[\frac{\alpha}{1 + \alpha} \mathbf{A}_r \right]^k.$$

Above two equations are of the same form, because they are both Taylor series of $f(t) = (1 + \alpha t)^{-1}$ at $t = 1$.

Let $M := \left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{A}$, whose eigenvalues are also of magnitude less than 1, then we have

$$\left(\mathbf{I} - \left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{A}\right)^{-1} = \sum_{k=0}^{\infty} \left[\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{A}\right]^k. \quad (4.30)$$

$$\text{Left side} = \left[\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D} - \mathbf{A}\right)\right]^{-1} \quad (4.31)$$

$$= \left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D} - \mathbf{A}\right)^{-1}\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right) \quad (4.32)$$

$$= \left(\frac{1}{\alpha}\mathbf{I} + \mathbf{L}\right)^{-1}\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right) \quad (4.33)$$

$$= \left(\mathbf{I} + \alpha\mathbf{L}\right)^{-1}\left(\mathbf{I} + \alpha\mathbf{D}\right) \quad (4.34)$$

whence

$$\left(\mathbf{I} + \alpha\mathbf{L}\right)^{-1}\left(\mathbf{I} + \alpha\mathbf{D}\right) = \sum_{k=0}^{\infty} \left[\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{A}\right]^k \quad (4.35)$$

$$\Rightarrow \left(\mathbf{I} + \alpha\mathbf{L}\right)^{-1} = \sum_{k=0}^{\infty} \left[\left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{A}\right]^k \left(\mathbf{I} + \alpha\mathbf{D}\right)^{-1} \quad (4.36)$$

□

We can then compute $\mathbf{Z}_s = p_{\text{ap}}(\mathbf{L}_s)\mathbf{X}$ and $\mathbf{Z}_r = p_{\text{ap}}(\mathbf{L}_r)\mathbf{X}$ iteratively with

$$\begin{cases} \mathbf{Z}_s^{(0)} = \mathbf{X}, \\ \mathbf{Z}_s^{(k+1)} = \mathbf{X} + \frac{\alpha}{1+\alpha}\mathbf{A}_s\mathbf{Z}_s^{(k)}, \\ \mathbf{Z}_s \approx \frac{1}{1+\alpha}\mathbf{Z}_s^{(K)} \end{cases}, \quad \begin{cases} \mathbf{Z}_r^{(0)} = \mathbf{X}, \\ \mathbf{Z}_r^{(k+1)} = \mathbf{X} + \frac{\alpha}{1+\alpha}\mathbf{A}_r\mathbf{Z}_r^{(k)}, \\ \mathbf{Z}_r \approx \frac{1}{1+\alpha}\mathbf{Z}_r^{(K)} \end{cases}. \quad (4.37)$$

Or, compute $\mathbf{Z} = p_{\text{ap}}(\mathbf{L}_s)\mathbf{X}$ with

$$\begin{cases} \mathbf{Z}^{(0)} = \mathbf{X}, \\ \mathbf{Z}^{(k+1)} = \mathbf{X} + \left(\frac{1}{\alpha}\mathbf{I} + \mathbf{D}\right)^{-1}\mathbf{AZ}^{(k)} \\ \mathbf{Z} \approx \mathbf{Z}^{(K)}\left(\mathbf{I} + \alpha\mathbf{D}\right)^{-1} \end{cases} \quad (4.38)$$

Empirically, we find that $k = \lceil 4\alpha \rceil$ is enough to get a good approximation. Hence, the computational complexity is reduced to $\mathcal{O}(nm\alpha + Nm\alpha)$ (note that X is of size $n \times m$), where N is the number of nonzero entries in L , and $N \ll n^2$ when the graph is sparse.

Renormalization

The renormalization (RNM) filter is an exponential function of the renormalized adjacency filter used in GCN:

$$p_{\text{rnm}}(\tilde{\mathbf{L}}_s) = \left(\mathbf{I} - \tilde{\mathbf{L}}_s \right)^k. \quad (4.39)$$

We have shown in Section 4.2.2 that although the response function $p_{\text{rnm}}(\lambda) = (1 - \lambda)^k$ is not low-pass, the renormalization trick shrinks the range of eigenvalues of $\tilde{\mathbf{L}}_s$ and makes p_{rnm} resemble a low-pass filter. The exponent parameter k controls the low-pass effect of p_{rnm} . When $k = 0$, p_{rnm} is all-pass. When k increases, p_{rnm} becomes more low-pass. Note that for a sparse graph, matrix $(\mathbf{I} - \tilde{\mathbf{L}}_s)$ is also sparse. Hence, the fastest way to compute $\mathbf{Z} = p_{\text{rnm}}(\tilde{\mathbf{L}}_s)\mathbf{Z}$ is to left multiply \mathbf{X} by $(\mathbf{I} - \tilde{\mathbf{L}}_s)$ repeatedly for k times, which has the computational complexity $\mathcal{O}(Nmk)$.

Random Walk

We also propose to design a random walk (RW) filter:

$$p_{\text{rw}}(\mathbf{L}_r) = \left(\frac{1}{2}(\mathbf{I} + \mathbf{A}_r) \right)^k = \left(\mathbf{I} - \frac{1}{2}\mathbf{L}_r \right)^k. \quad (4.40)$$

We call p_{rw} the random walk filter because $\frac{1}{2}(\mathbf{I} + \mathbf{A}_r)$ is a stochastic matrix of a lazy random walk which at each step returns to the current state with probability $\frac{1}{2}$, and $(\frac{1}{2}(\mathbf{I} + \mathbf{A}_r))^k$ is the k -step transition probability matrix. Similarly, we can derive the

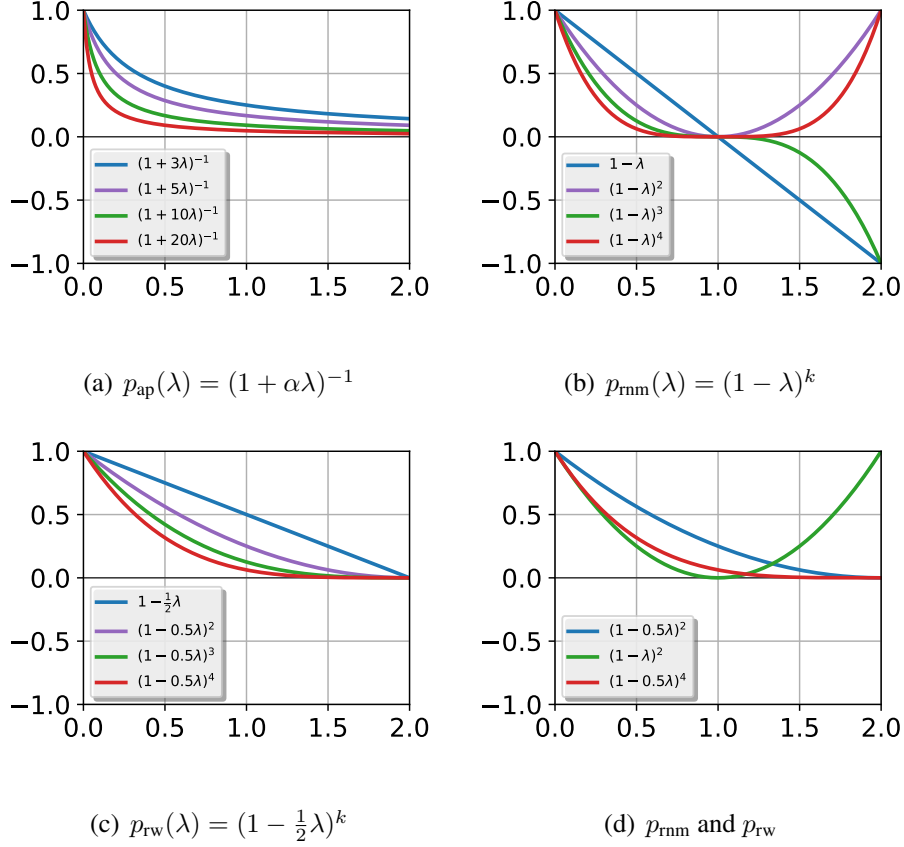


Figure 4.7: Response functions of AP, RNM, and RW filters, and their comparison.

response function of p_{rw} as

$$p_{\text{rw}}(\lambda) = (1 - \frac{1}{2}\lambda)^k. \quad (4.41)$$

Note that \mathbf{L}_r has the same eigenvalues with \mathbf{L}_s , with range $[0, 2]$. Unlike the RNM, p_{rw} is a typical low-pass filter on $[0, 2]$, as shown in Fig. 4.7(c). The fastest way to compute $\mathbf{Z} = p_{\text{rnm}}(\mathbf{L}_r)\mathbf{Z}$ is to left multiply \mathbf{X} by $(\mathbf{I} - \mathbf{L}_r)$ repeatedly for k times, and thus RW filter has the same complexity $\mathcal{O}(Nmk)$ as RNM filter.

Filter Strength

The strength of the AR, RNM, and RW filters is controlled by parameters α and k , respectively. However, choosing appropriate α and k for different application scenarios

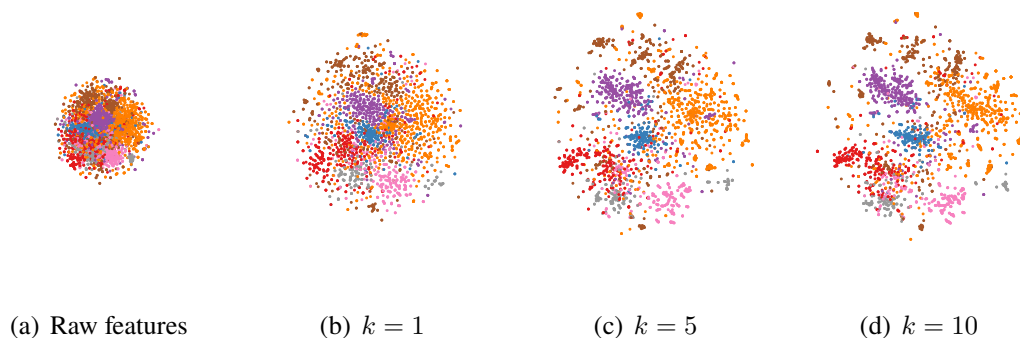


Figure 4.8: Visualization of raw and filtered Cora features (using the RNM filter with different k).

is non-trivial. An important factor that should be taken into account is the label rate. Intuitively, when there are very few labels in each class, one should increase filter strength so that distant nodes can have similar feature representations as the labeled nodes for ease of classification. However, over-smoothing often results in inaccurate class boundaries. Therefore, when the label rate is reasonably high, reducing filter strength would be desirable to preserve feature diversity to learn more accurate class boundaries.

Fig. 4.8 visualizes the raw and filtered features of Cora produced by the RNM filter and projected by t-SNE [100]. It can be seen that as k increases, the RNM filter produces smoother embeddings, i.e., the filtered features exhibit a more compact cluster structure, making classification possible with only a few labels.

Empirically, the following three response functions perform similarly to each other.

$$\begin{cases} p_{\text{rnm}}(\lambda) = (1 - \lambda)^k \\ p_{\text{rw}}(\lambda) = (1 - \frac{1}{2}\lambda)^{2k} \\ p_{\text{ap}}(\lambda) = (1 + 2k\lambda)^{-1} \end{cases} \quad (4.42)$$

For example, the curves of $(1 - \lambda)^2$ and $(1 - \frac{1}{2}\lambda)^4$ are very close as Fig. 4.7(d) shows, which implies that to have the same level of low-pass effect, k in p_{rw} should be set twice

as large as in p_{rnn} . This may be explained by the fact that the two functions $(1 - \lambda)^k$ and $(1 - \frac{1}{2}\lambda)^{2k}$ have the same derivative k at $\lambda = 0$.

However, all the above-mentioned principles for filter strength selection are based on intuition and qualitative. Even with these principles, α and k are still hyper-parameters requiring manual tuning. A method to quantitatively select filter strength is proposed in our paper Zhang et al. [2], which will be introduced in Section 4.4.

4.4 Adaptive Graph Convolution for Clustering

Attributed graph clustering [69] aims to cluster vertices of an attributed graph where each vertex is associated with a set of feature attributes. Attributed graphs widely exist in real-world applications such as social networks, citation networks, protein-protein interaction networks, etc. Clustering plays a vital role in detecting communities and analyzing the structures of these networks. However, attributed graph clustering requires joint modeling of graph structures and vertex attributes to make full use of available data, which presents great challenges. Recent progress on graph convolutional networks has proved that graph convolution is effective in combining structural and content information. Several recent methods based on it have achieved promising clustering performance on some real attributed networks. However, there is a limited understanding of how graph convolution affects clustering performance and how to use it properly to optimize performance for different graphs. Existing methods essentially use graph convolution of a fixed and low order that only takes into account neighbors within a few hops of each vertex, which under-utilizes vertex relations and ignores the diversity of graphs. In this section, we propose an adaptive graph convolution method for attributed graph clustering that exploits

high-order graph convolution to capture global cluster structure and adaptively selects the appropriate order for different graphs. We establish the validity of our method by theoretical analysis and extensive experiments on benchmark datasets. Empirical results show that our method compares favorably with state-of-the-art methods.

Graph Vertex Clustering Given a non-directed graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$, where $\mathcal{V} = \{\nu_1, \nu_2, \dots, \nu_n\}$ is a set of vertices with $|\mathcal{V}| = n$, and $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}^{n \times n}$ is the adjacency matrix, and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times m}$ is a feature matrix containing feature vectors of all vertices, where $\mathbf{x}_i \in \mathbb{R}^m$ is the feature vector of v_i . Our goal is to partition the vertices \mathcal{V} into c clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_c\}$.

4.4.1 k-Order Random Walk Graph Convolution

To make clustering easy, it is desired that vertices of the same class should have similar feature representations after graph filtering, which is achievable by low-pass filters. Here, we design a low-pass graph filter with the frequency response function

$$p(\lambda_q) = 1 - \frac{1}{2}\lambda_q. \quad (4.43)$$

As shown by the red line in Fig. 4.9, one can see that $p(\cdot)$ in Eq. (4.43) is decreasing and nonnegative on $[0, 2]$. Note that all the eigenvalues λ_q of the symmetrically normalized graph Laplacian L_s fall into the interval $[0, 2]$ [23], which indicates that $p(\cdot)$ in Eq. (4.43) is low-pass. The graph filter \mathbf{G} with $p(\cdot)$ in Eq. (4.43) as the frequency response function can then be written as

$$\mathbf{G} = \Phi p(\Lambda) \Phi^{-1} = \Phi \left(I - \frac{1}{2} \Lambda \right) \Phi^{-1} = I - \frac{1}{2} L_s. \quad (4.44)$$

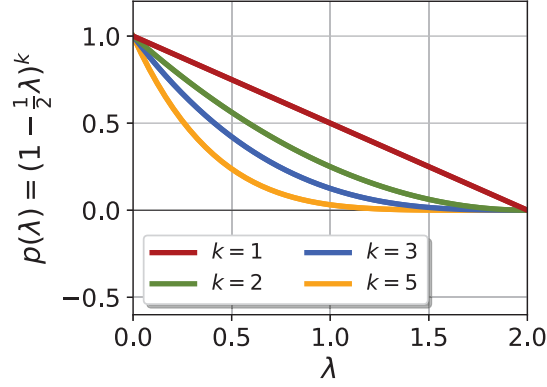


Figure 4.9: Frequency response

By performing graph convolution on the feature matrix X , we obtain the filtered embeddings Z :

$$Z = GX, \quad (4.45)$$

where $Z = [z_1, z_2, \dots, z_n]^\top$ are the filtered vertex features or embeddings. Applying such a low-pass graph filter on the feature matrix makes adjacent vertices have similar feature values within each channel, i.e., the graph signals are smooth. Based on the cluster assumption that nearby vertices are likely to be in the same cluster, performing graph convolution with a low-pass graph filter will make the downstream clustering task easier.

Note that the proposed graph filter in Eq. (4.44) is different from the graph filter used in GCN. The graph filter in GCN is $G = I - L_s$ with the frequency response function $p(\lambda) = 1 - \lambda$ [3], which is clearly not low-pass as its magnitude increase with λ for $\lambda \in (1, 2]$.

k -Order Graph Convolution However, the first-order graph convolution in Eq. (4.44) may not be adequate to achieve this, especially for large and sparse graphs, as it updates

each vertex v_i by the aggregation of its 1-hop neighbors only, without considering long-distance neighborhood relations. To capture global graph structures and facilitate clustering, we propose to use k -order graph convolution.

We define the k -order convolutional filter as

$$\mathbf{Z} = \mathbf{GX} = \left(\mathbf{I} - \frac{1}{2}\mathbf{L}_s\right)^k \mathbf{X}, \quad (4.46)$$

where k is a positive integer, and the corresponding frequency response in Eq. (4.46) is

$$p(\lambda_q) = 1 - \frac{1}{2}\lambda_q^k \cdot p_{\text{rw}}(\lambda) = \left(1 - \frac{1}{2}\lambda\right)^k. \quad (4.47)$$

As shown in Fig. 4.9, $p(\lambda)$ in Eq. (4.47) becomes more low-pass as k increases, indicating that the filtered vertex features \mathbf{Z} will be smoother.

The iterative calculation formula of k -order graph convolution is

$$\begin{aligned} \mathbf{Z}^{(0)} &= \mathbf{X}, \\ \mathbf{Z}^{(1)} &= \frac{1}{2} \left(\mathbf{Z}^{(0)} + \mathbf{A}_s \mathbf{Z}^{(0)} \right), \\ &\vdots \\ \mathbf{Z}^{(k)} &= \frac{1}{2} \left(\mathbf{Z}^{(k-1)} + \mathbf{A}_s \mathbf{Z}^{(k-1)} \right), \end{aligned} \quad (4.48)$$

and the final $\mathbf{Z}^{(k)}$ is the \mathbf{Z} we want.

From Eq. (4.48), one can easily see that k -order graph convolution updates the features of each vertex v_i by aggregating the features of its k -hop neighbours iteratively. As k -order graph convolution takes into account long-distance data relations, it can be helpful for capturing global graph structures to improve clustering performance.

Theoretical Analysis As k increases, k -order graph convolution will make the vertex features smoother on each dimension. In the following, we prove this using the Laplacian-

Beltrami operator $\omega(\cdot)$ defined in Eq. (3.32). Denote by \mathbf{x} a column of the feature matrix \mathbf{X} , which can be decomposed as $\mathbf{x} = \Phi \mathbf{c}$.

$$\omega_s(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{L}_s \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} = \frac{\mathbf{c}^\top \Lambda \mathbf{c}}{\|\mathbf{c}\|_2^2} = \frac{\sum_{i=1}^n \lambda_i c_i^2}{\sum_{i=1}^n c_i^2}. \quad (4.49)$$

Theorem 22. *If the frequency response function $p(\lambda)$ of a graph filter G is non-increasing and nonnegative for all λ_i , then for any signal \mathbf{x} and the filtered signal $\mathbf{z} = G\mathbf{x}$, we always have*

$$\omega_s(\mathbf{z}) \leq \omega_s(\mathbf{x}).$$

Proof. We first prove the following lemma by induction. The following inequality

$$S_a^{(n)} = \frac{\sum_{i=1}^n a_i T_i}{\sum_{i=1}^n a_i} \leq \frac{\sum_{i=1}^n b_i T_i}{\sum_{i=1}^n b_i} = S_b^{(n)} \quad (4.50)$$

holds, if $T_1 \leq \dots \leq T_n$ and $\frac{a_1}{b_1} \geq \dots \geq \frac{a_n}{b_n}$ with $\forall a_i, b_i \geq 0$. It is easy to verify that it holds when $n = 2$.

Now assume that $S_a^{(n-1)} \leq S_b^{(n-1)}$ holds. Then, consider the case of $S_a^{(n)}$ and $S_b^{(n)}$.

$$\begin{aligned} S_a^{(n)} &= \frac{\sum_{i=1}^n a_i T_i}{\sum_{i=1}^n a_i} = \frac{\sum_{i=1}^{n-1} a_i T_i + a_n T_n}{\sum_{i=1}^{n-1} a_i + a_n} \\ &= \frac{(\sum_{i=1}^{n-1} a_i) S_a^{(n-1)} + a_n T_n}{\sum_{i=1}^{n-1} a_i + a_n} \\ &\leq \frac{(\sum_{i=1}^{n-1} a_i) S_b^{(n-1)} + a_n T_n}{\sum_{i=1}^{n-1} a_i + a_n}. \end{aligned} \quad (4.51)$$

Since $S_b^{(n-1)} = \frac{\sum_{i=1}^{n-1} b_i T_i}{\sum_{i=1}^{n-1} b_i} \leq \frac{\sum_{i=1}^{n-1} b_i T_{n-1}}{\sum_{i=1}^{n-1} b_i} = T_{n-1}$, we have $S_b^{(n-1)} \leq T_n$. Also note

that $\frac{\sum_{i=1}^{n-1} a_i}{\sum_{i=1}^{n-1} b_i} \geq \frac{a_n}{b_n}$. Since the lemma holds when $n = 2$, we have

$$\frac{(\sum_{i=1}^{n-1} a_i) S_b^{(n-1)} + a_n T_n}{\sum_{i=1}^{n-1} a_i + a_n} \leq \frac{(\sum_{i=1}^{n-1} b_i) S_b^{(n-1)} + b_n T_n}{\sum_{i=1}^{n-1} b_i + b_n} = \frac{\sum_{i=1}^n b_i T_i}{\sum_{i=1}^n b_i} = S_b^{(n)}, \quad (4.52)$$

which shows that the inequality $S_a^{(n)} \leq S_b^{(n)}$ also holds. By induction, the above lemma holds for all $n = 2, 3, \dots$

We can now prove Theorem 1 using this lemma. For convenience, we arrange the eigenvalues λ_i of L_s in increasing order such that $0 \leq \lambda_1 \leq \dots \leq \lambda_n$. Since $p(\lambda)$ is nonincreasing and nonnegative, $p(\lambda_1) \geq \dots \geq p(\lambda_n) \geq 0$. Theorem 1 is then proved with the above lemma by letting

$$T_i = \lambda_i, \quad a_i = p^2(\lambda_i)z_i^2, \quad b_i = z_i^2. \quad (4.53)$$

□

Assuming that z_{k-1} and z_k are obtained by $(k-1)$ -order and k -order graph convolution, respectively, one can immediately infer from Theorem 1 that z_k is smoother than z_{k-1} . In other words, k -order graph convolution will produce smoother features as k increases. Since vertices in the same cluster tend to be densely connected, they are likely to have more similar feature representations with large k , which can benefit clustering.

4.4.2 Clustering via Adaptive Graph Convolution

We perform the classical spectral clustering method [24, 101] on the filtered feature matrix Z to partition the vertices of \mathcal{V} into c clusters, similar to [88]. Specifically, we first apply the linear kernel $\mathbf{K} = \mathbf{Z}\mathbf{Z}^T$ to learn pairwise similarity between vertices. Then we calculate $\mathbf{W} = \frac{1}{2}(|\mathbf{K}| + |\mathbf{K}^T|)$ to make sure that the similarity matrix is symmetric and nonnegative, where $|\cdot|$ means taking the absolute value of each element of the matrix. Finally, we perform spectral clustering on \mathbf{W} to obtain clustering results by computing the eigenvectors associated with the c largest eigenvalues of \mathbf{W} and then applying the k -means algorithm on the eigenvectors to obtain cluster partitions.

The central issue of k -order graph convolution is how to select an appropriate k . Although

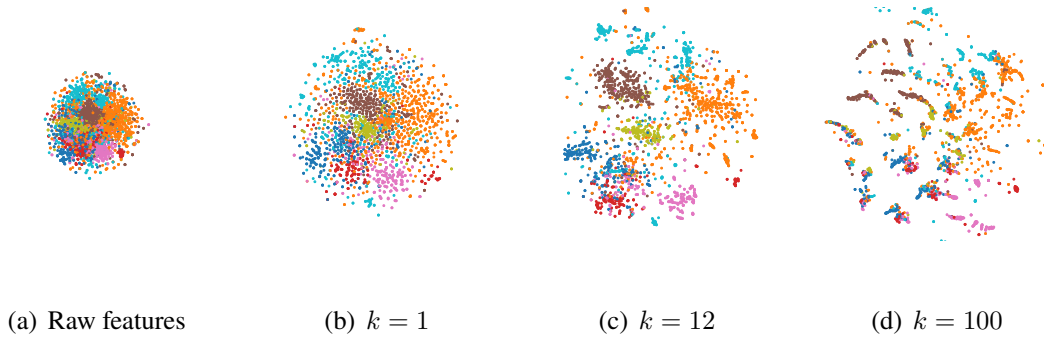


Figure 4.10: t-SNE visualization of Cora’s raw and filtered node features with different k .

k -order graph convolution can make nearby vertices have similar feature representations, k is definitely not the larger the better. Too large k will lead to over-smoothing, i.e., the features of vertices in different clusters are mixed and become indistinguishable. Fig. 4.10 visualizes the raw and filtered vertex features of the *Cora* citation network with different k , where the features are projected by t-SNE [100] and vertices of the same class are indicated by the same colour. It can be seen that the vertex features become similar as k increases. The data exhibits clear cluster structures with $k = 12$. However, with $k = 100$, the features are over-smoothed, and vertices from different clusters are mixed.

To adaptively select the order k , we use the clustering performance metric – internal criteria based on the information intrinsic to the data alone [102]. Here, we consider intra-cluster distance ($\text{intra}(\mathcal{C})$ for a given cluster partition $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_c\}$), which represents the compactness of the partition and is defined as:

$$\text{intra}(\mathcal{C}) = \frac{1}{c} \sum_{\mathcal{C}_k \in \mathcal{C}} \frac{1}{|\mathcal{C}_k|(|\mathcal{C}_k| - 1)} \sum_{\substack{v_i, v_j \in \mathcal{C}_k, \\ v_i \neq v_j}} \|z_i - z_j\|_2. \quad (4.54)$$

Note that inter-cluster distance can also be used to measure clustering performance given fixed data features. A good cluster partition should have a large inter-cluster distance and a small intra-cluster distance. However, by Theorem 1, the vertex features become

smoother as k increases, which could significantly reduce both intra-cluster and inter-cluster distances. Hence, the inter-cluster distance may not be reliable for measuring the clustering performance w.r.t. different k . So we propose to observe the variation of intra-cluster distance for choosing k .

Our strategy is to find the first local minimum of $\text{intra}(\mathcal{C})$ w.r.t. k . Specifically, we start from $k = 1$ and increment it by 1 iteratively. In each iteration t , we first obtain the cluster partition $\mathcal{C}^{(t)}$ by performing k -order ($k = t$) graph convolution and spectral clustering, then we compute $\text{intra}(\mathcal{C}^{(t)})$. Once $\text{intra}(\mathcal{C}^{(t)})$ is larger than $\text{intra}(\mathcal{C}^{(t-1)})$, we stop the iteration and set the chosen $k = t - 1$. More formally, consider $\Delta\text{intra}(t - 1) = \text{intra}(\mathcal{C}^{(t)}) - \text{intra}(\mathcal{C}^{(t-1)})$, the criterion for stopping the iteration is $\Delta\text{intra}(t - 1) > 0$, i.e., stops at the first local minimum of $\text{intra}(\mathcal{C}^{(t)})$. So, the final choice of cluster partition is $\mathcal{C}^{(t-1)}$. The benefits of this selection strategy are two-fold. First, it ensures finding a local minimum for $\text{intra}(\mathcal{C})$ that may indicate a good cluster partition and avoids over-smoothing. Second, it is time efficient to stop at the first local minimum of $\text{intra}(\mathcal{C})$.

4.4.3 Algorithm Procedure and Time Complexity

The overall procedure of AGC is shown in Algorithm 1. Denote by n the number of vertices, m the number of attributes, c the number of clusters, and N the number of nonzero entries of the adjacency matrix \mathbf{A} . Note that for a sparse \mathbf{A} , $N \ll n^2$. The time complexity of computing \mathbf{L}_s in the initialization is $\mathcal{O}(N)$. In each iteration, for a sparse \mathbf{L}_s , the fastest way of computing k -order graph convolution in (Eq. (4.46)) is to left multiply \mathbf{X} by $\mathbf{I} - \frac{1}{2}\mathbf{L}_s$ repeatedly for k times, which has the time complexity $\mathcal{O}(Ndk)$. In each iteration, the time complexity of performing spectral clustering on

Algorithm 1 AGC

Require: Vertex set \mathcal{V} , adjacency matrix \mathbf{A} , feature matrix \mathbf{X} , and maximum iteration number max_iter .

Ensure: Cluster partition \mathcal{C} .

- 1: Initialize $t = 0$ and $\text{intra}(\mathcal{C}^{(0)}) = +\infty$. Compute the symmetrically normalized graph Laplacian $\mathbf{L}_s = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.
 - 2: **repeat**
 - 3: Set $t := t + 1$ and $k := t$.
 - 4: Perform k -order graph convolution by Eq. (4.46) and get \mathbf{Z} .
 - 5: Apply the linear kernel $\mathbf{K} = \mathbf{Z}\mathbf{Z}^T$, and calculate the similarity matrix $\mathbf{W} = \frac{1}{2}(|\mathbf{K}| + |\mathbf{K}^T|)$.
 - 6: Obtain the cluster partition $\mathcal{C}^{(t)}$ by performing spectral clustering on \mathbf{W} .
 - 7: Compute $\text{intra}(\mathcal{C}^{(t)})$ by Eq. (4.54).
 - 8: **until** $\Delta\text{intra}(t - 1) > 0$ or $t > \text{max_iter}$
 - 9: Set $k := t - 1$ and $\mathcal{C} := \mathcal{C}^{(t-1)}$.
-

\mathbf{X} is $\mathcal{O}(n^2m + n^2c)$. The time complexity of computing $\text{intra}(\mathcal{C})$ in each iteration is $\mathcal{O}(\frac{1}{c}n^2m)$. Since c is usually much smaller than n and m , the time complexity of each iteration is approximate $\mathcal{O}(n^2m + Ndk)$. If Algorithm 1 iterates t times, the overall time complexity of AGC is $\mathcal{O}(n^2dt + Ndt^2)$. Unlike existing GCN-based clustering methods, AGC does not need to train the neural network parameters, which makes it time efficient.

Chapter Review

This chapter reveals the fundamental mechanisms of GCNs, provides new insights into GCNs for semi-supervised learning, and points out the over-smoothing problem. Our spectral analysis revisits GCN and classical label propagation methods under a unified graph filtering framework and shows the critical importance of low-pass graph filtering.

Inspired by the new theoretical insights, we propose an efficient decoupled framework for graph-based semi-supervised and unsupervised learning. We also provide guidelines to adaptively adjust the filter strength for semi-supervised learning and develop algorithms to properly select the parameter of convolutional filters.

Chapter 5

Learning with 2-D Graph

Convolutional

Existing GCN variants commonly use 1-D graph convolution that solely operates on the object link graph without exploring informative relational information among object attributes. This significantly limits their modeling capability and may lead to inferior performance on noisy and sparse real-world networks. This chapter explores 2-D graph convolution to jointly model object links and attribute relations for graph representation learning. Specifically, we propose a computationally efficient dimensionwise separable 2-D graph convolution (DSGC) for filtering node features. Theoretically, we show that DSGC can reduce the intra-class variance of node features on both the object dimension and the attribute dimension to learn more effective representations. Empirically, we demonstrate that by modeling attribute relations, DSGC achieves significant performance gain over state-of-the-art methods for node classification and clustering on a variety of real-world networks.

Learning effective node representations by utilizing graph structures and other aspects of information, such as node content, has proved very useful for unsupervised and semi-supervised learning on graphs. Previous approaches for unsupervised learning have explored applying non-negative matrix factorization, random walk statistics, and Laplacian eigenmaps on both graph structures and node attributes [103, 104] to learn node representations. For semi-supervised learning, a common way is to regularize a supervised classifier trained with node features by a Laplacian regularizer or an embedding-based regularizer to take into account graph structures, e.g., manifold regularization [56], manifold denoising [105], deep semi-supervised embedding [40], and Planetoid [8]. However, these methods model node connectivity and node content separately and hence may not be able to fully utilize the information.

In the past few years, graph convolutional neural networks (GCN) and variants have dominated the research of graph representation learning and achieved new state-of-the-art results in various learning tasks on graphs, especially in unsupervised learning [2, 88] and semi-supervised learning [3, 11, 15, 16]. The success is mainly attributed to graph convolution, a function that naturally combines node connectivity and node content for feature propagation and smoothing, which computes the representation of a node by aggregating the features of its neighbors. The effective utilization of both modalities of data gives the unique advantage to GCN-based methods over previous approaches, including topology-only models [74, 75] and graph regularization-based methods [8, 56]. Despite their empirical success, one major limitation of most existing GCN-based methods is that they commonly adopt one-dimensional (1-D) graph convolution that operates on the *object link graph* to model node (object) relations and features, whose performance

critically relies on the quality of the graph. However, real-life networks are often noisy and sparse. For example, in a web graph such as Wikipedia, a hyperlink between two web pages does not necessarily indicate that they belong to the same category. Mixing their features could be harmful to webpage classification or clustering. Moreover, it has been shown that many real-world networks are scale-free, and there exist many low-degree nodes [20]. Since these nodes may have very few or even no links to other nodes, it is difficult, and even impossible, to do feature propagation to endow them with similar features as other same-class nodes to facilitate downstream learning tasks.

To address the above limitation, we propose to explore data relations in another dimension by constructing an *attribute affinity graph*, which encodes relations between object attributes. The underlying assumption is attributes that indicate the same category should have strong relations. For example, in a citation network, object attributes are words, and documents of the AI category usually contain words such as “learning”, “robotics”, “machine”, “neural”, etc. These indicative words for the AI category should be more closely related than other non-indicative words. These informative relations can then be utilized for feature smoothing, similar to the use of object links. Notably, the attribute affinity graph can be a useful complement to the object link graph in learning node representations.

To formalize the above insight, we propose to perform graph convolution on the attribute affinity graph (Fig. 5.1 (c)). Further, we develop an efficient two-dimensional (2-D) graph convolution to perform feature smoothing on both the object link graph and the attribute affinity graph to learn more effective node representations (Fig. 5.1 (d)). Our main contributions are described as follows.

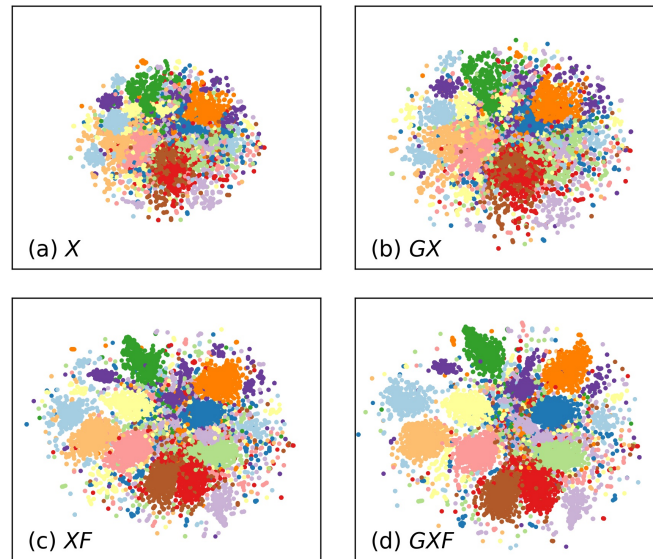


Figure 5.1: t-SNE visualization of the “20 Newsgroups” dataset. (a) Raw features; (b) Filtered by the regular object graph convolution; (c) Filtered by our proposed attribute graph convolution; (d) Filtered by our proposed dimensionwise separable graph convolution (DSGC).

- Methodology:** We propose to use 2-D graph convolution to jointly model object links, attribute relations, and object features for node representation learning. Furthermore, we develop a computationally efficient dimensionwise separable 2-D graph convolutional filter (DSGC), which is equivalent to performing 1-D graph convolution alternately on the object dimension and the attribute dimension, as illustrated in Fig. 5.2. Finally, we propose two learning frameworks based on DSGC for unsupervised node clustering and semi-supervised node classification.
- Theoretical insight:** We show that the regular 1-D graph convolution on the object link graph can reduce the intra-class variance of node features, which helps to explain the success of many existing methods. Further, we show that the same can be proved for graph convolution on a properly constructed attribute affinity

graph. Jointly, they provide a theoretical justification for DSGC.

- **Empirical study:** We implement DSGC for semi-supervised node classification and unsupervised node clustering and conduct extensive experiments on various real-world networks, including email networks, citation networks, and web graphs. The comparison with state-of-the-art methods demonstrates the advantages of DSGC over the regular 1-D graph convolution. Moreover, we show that DSGC can be easily plugged into some strong GCN-based methods to further improve their performance substantially.

5.1 2-D Graph Convolution

Graph signal processing has been an active research field in recent years. It generalizes basic concepts in harmonic analysis, including signals, filters, Fourier transform, and convolution, to the graph domain. Given a graph \mathcal{G} with a vertex set \mathcal{V} and an adjacency matrix \mathbf{A} , if we associate each vertex ν_i with a real value x_i , then a signal on \mathcal{G} can be defined as a vector $\mathbf{x} = [x_1, \dots, x_n]^\top$. Graph filters are defined as mappings between input and output signals. Let \mathbf{G} be a polynomial of \mathbf{A} (usually normalized), i.e., $\mathbf{G} = p(\mathbf{A})$, then \mathbf{G} is a legit convolutional filter on graph \mathcal{G} , and the corresponding 1-D graph convolution is defined as

$$\mathbf{z} = \mathbf{G}\mathbf{x}, \quad (5.1)$$

where \mathbf{z} is the output signal. Existing graph convolutional filters are all defined in this way and have achieved considerable success in various learning tasks on graphs.

However, previous research mainly focuses on designing and applying 1-D graph convolu-

tion. This section presents a 2-D graph convolution for learning node representations. A comprehensive introduction to multi-dimensional graph convolution is provided by [94]. Based on the theory developed by [94], we propose a localized 2-D graph convolution to circumvent the computationally intensive graph Fourier transform. Furthermore, we propose an even simpler dimensionwise separable 2-D graph convolution to model both object links and attribute relations efficiently.

5.1.1 2-D Graph Signal and Spectral Convolution

2-D Graph Signal A 2-D graph signal is a function defined on the Cartesian product of the vertex sets of two graphs. Formally, given two graphs \mathcal{G}_1 and \mathcal{G}_2 with n and m nodes respectively, and denote the vertex sets by \mathcal{V}_1 and \mathcal{V}_2 . A real-valued signal defined on them is a function $x : \mathcal{V}_1 \times \mathcal{V}_2 \rightarrow \mathbb{R}$. For convenience, we simply denote $x(\nu_i^{(1)}, \nu_j^{(2)})$ by x_{ij} and organize them as a matrix:

$$\mathbf{X} = (x_{ij}) \in \mathbb{R}^{n \times m}, \quad x_{ij} = x(\nu_i^{(1)}, \nu_j^{(2)}), \quad (5.2)$$

Signal \mathbf{X} associates each node pairs $(\nu_i^{(1)}, \nu_j^{(2)}) \in \mathcal{V}_1 \times \mathcal{V}_2$ with a real number x_{ij} . For example, the feature matrix given by usual node classification tasks is a 2-D signal defined on the object link graph and the attribute affinity graph.

2-D Graph Fourier Transform Define the graph Laplacian of \mathcal{G}_1 and \mathcal{G}_2 as $\mathbf{L}_1 = \mathbf{D}_1 - \mathbf{A}_1$ and $\mathbf{L}_2 = \mathbf{D}_2 - \mathbf{A}_2$, where $\mathbf{A}_1, \mathbf{A}_2$ are adjacency matrices and $\mathbf{D}_1, \mathbf{D}_2$ are the corresponding degree matrices. * Assuming two Laplacian matrices have the following

*Discussion below also applies to row-normalized Laplacian $\mathbf{L}_r = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, column-normalized Laplacian, $\mathbf{L}_c = \mathbf{I} - \mathbf{A}\mathbf{D}^{-1}$, and symmetric normalized Laplacian $\mathbf{L}_s = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

eigen-decomposition

$$\mathbf{L}_1 = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}, \quad \mathbf{L}_2 = \mathbf{V}\mathbf{M}\mathbf{V}^{-1}, \quad (5.3)$$

where $\mathbf{\Lambda}$, \mathbf{M} stores the eigenvalues λ_i, μ_j of two Laplacian matrices in their main diagonals, $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n]$ and $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_m]$ store the corresponding unit-length eigenvectors in their columns. This eigen-decomposition is always attainable for undirected graphs and nearly always attainable for directed graphs.

All $n \times m$ outer products $\mathbf{u}_i \mathbf{v}_j^\top$ together form a basis for space $\mathbb{R}^{n \times m}$. It is known as the 2-D graph Fourier basis – an analogy of the Fourier basis in classical harmonic analysis in the graph domain. A 2-D graph signal \mathbf{X} can be decomposed into this basis with coefficients s_{ij} :

$$\mathbf{X} = \sum_{ij} s_{ij} (\mathbf{u}_i \mathbf{v}_j^\top), \quad (5.4)$$

or in matrix form:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \quad \text{where } \mathbf{S} = (s_{ij}) \in \mathbb{R}^{n \times m}. \quad (5.5)$$

\mathbf{S} is called the spectrum or Fourier coefficients of signal \mathbf{X} and can be obtained by formula

$$\mathbf{S} = \mathbf{U}^{-1} \mathbf{X} (\mathbf{V}^{-1})^\top. \quad (5.6)$$

Eq. (5.6) is so-called 2-D graph Fourier transform; Eq. (5.5) is the corresponding inverse transform.

2-D Spectral Graph Convolution Given the decomposition above, we can now manipulate the spectrum of 2-D signals and define 2-D spectral graph convolution. Convolution is an operation that takes a signal as input and outputs another signal. By convolution theorem, convolution is equivalent to scaling entries of the spectrum. Thus, given a

signal \mathbf{X} with spectrum \mathbf{S} , a 2-D spectral graph convolution with \mathbf{X} as input is defined as:

$$\mathbf{Z} = \mathbf{U}(\mathbf{S} \circ \mathbf{P})\mathbf{V}^\top, \quad (5.7)$$

where \mathbf{P} is the convolution's spectral kernel (parameters in the spectral domain), and ' \circ ' is Hadamard (entry-wise) product. \mathbf{P} is also referred to as the frequency response of the convolution.

5.1.2 Fast Localized 2-D Graph Convolution

Although Eq. (5.7) well defines 2-D graph convolution, it is often impractical to perform convolution in the spectral domain due to the high cost of computing Fourier basis \mathbf{U} and \mathbf{V} . Similar to what [14] did to 1-D graph convolution, we propose 2-D spatial graph convolution here to avoid intensive computation. To achieve this goal, we need to parameterize the entries of spectral kernel \mathbf{P} as a two-variable polynomial $p(\cdot, \cdot)$ of eigenvalues λ, μ such that $p_{ij} = p(\lambda_i, \mu_j)$. Denote coefficients of the polynomial by $\Theta = (\theta_{k_1 k_2}) \in \mathbb{R}^{n \times m}$, then \mathbf{P} is parameterized as

$$p_{ij} = p(\lambda_i, \mu_j) = \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \lambda_i^{k_1} \mu_j^{k_2}. \quad (5.8)$$

or in matrix form:

$$\mathbf{P} = \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \mathbf{\Lambda}^{k_1} \vec{\mathbf{1}}_{n \times m} \mathbf{M}^{k_2}, \quad (5.9)$$

where $\vec{\mathbf{1}}_{n \times m}$ denotes all-one matrix of shape $n \times m$. Substitute Eq. (5.9) into Eq. (5.7)

and rearrange it, 2-D graph convolution will become

$$\begin{aligned}
\mathbf{Z} &= \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \mathbf{U} \left(\mathbf{S} \circ (\mathbf{\Lambda}^{k_1} \vec{\mathbf{1}}_{n \times m} \mathbf{M}^{k_2}) \right) \mathbf{V}^\top \\
&= \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \mathbf{U} \mathbf{\Lambda}^{k_1} \left(\mathbf{S} \circ \vec{\mathbf{1}}_{n \times m} \right) \mathbf{M}^{k_2} \mathbf{V}^\top \\
&= \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \mathbf{U} \mathbf{\Lambda}^{k_1} \mathbf{S} \mathbf{M}^{k_2} \mathbf{V}^\top \\
&= \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} (\mathbf{U} \mathbf{\Lambda}^{k_1} \mathbf{U}^{-1}) \mathbf{X} (\mathbf{V} \mathbf{M}^{k_2} \mathbf{V}^{-1})^\top \\
&= \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{m-1} \theta_{k_1 k_2} \mathbf{L}_1^{k_1} \mathbf{X} (\mathbf{L}_2^{k_2})^\top
\end{aligned} \tag{5.10}$$

Eq. (5.10) is called 2-D spatial graph convolution, as it manipulates the signal \mathbf{X} in the spatial domain. Eigen-decomposition of Laplacian and Fourier transformation of \mathbf{X} is no longer required. Parameters $\Theta = (\theta_{k_1 k_2})$ in Eq. (5.10) are called the (spatial) kernel of this convolution. Similar to [14], this spatial convolutional filter is localized. Denote by K_1 and K_2 the largest exponent of \mathbf{L}_1 and \mathbf{L}_2 in the polynomial, i.e., $k_1 > K_1$ or $k_2 > K_2$ implies $\theta_{k_1 k_2} = 0$, then Eq. (5.10) becomes

$$\mathbf{Z} = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} \theta_{k_1 k_2} \mathbf{L}_1^{k_1} \mathbf{X} (\mathbf{L}_2^{k_2})^\top. \tag{5.11}$$

In this way, kernel Θ only need to be of size $(K_1 + 1) \times (K_2 + 1)$. The convoluted signal z_{ij} of vertex pair $(\nu_i^{(1)}, \nu_j^{(2)})$ only depends on the neighbourhood of $\nu_i^{(1)}$ within K_1 hops and the neighbourhood of $\nu_j^{(2)}$ within K_2 hops, so the filter is said to be K_1 -localized on \mathcal{G}_1 and K_2 -localized on \mathcal{G}_2 .

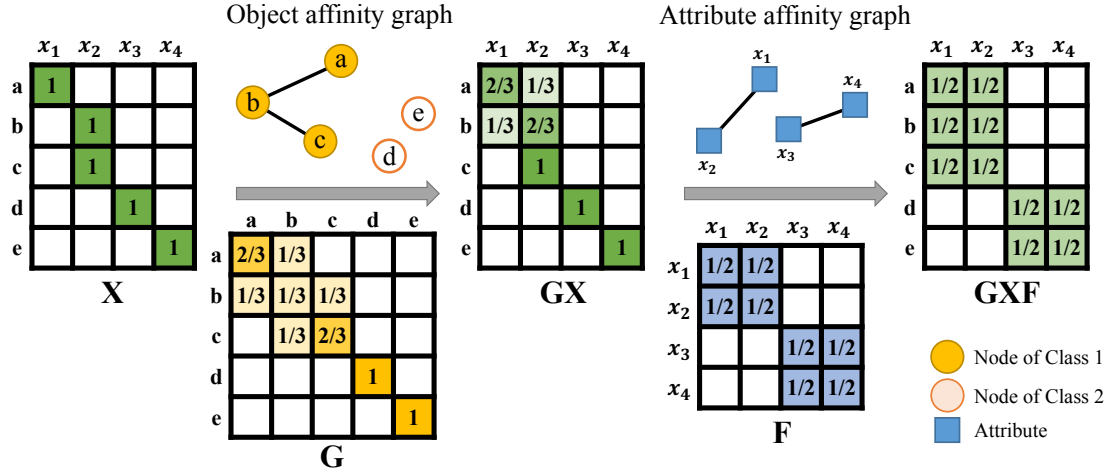


Figure 5.2: Conceptual illustration of DSGC by a toy example. The node representations (row vectors) obtained by DSGC (GXF) are better than those in the original feature matrix (X) or filtered by 1-D graph convolution (GX), in the sense that nodes of the same class have more similar features.

5.1.3 Dimensionwise Separable 2-D Graph Convolution (DSGC)

Although the above spatial graph convolution avoids the computationally expensive Fourier transform, its general form with kernel size $K_1 \times K_2$ still involves at least $K_1 \times K_2$ matrix multiplications. Inspired by the depth-wise separable convolution proposed in [106], we streamline spatial graph convolution by restricting the rank of Θ to be one. Consequently, Θ is able to be decomposed as an outer product of two vectors $\theta^{(1)} \in \mathbb{R}^n$ and $\theta^{(2)} \in \mathbb{R}^m$, i.e., $\Theta = \theta^{(1)}\theta^{(2)\top}$ and $\theta_{k_1 k_2} = \theta_{k_1}^{(1)}\theta_{k_2}^{(2)}$. Finally, the 2-D spatial graph convolution in Eq. (5.10) becomes

$$\mathbf{Z} = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} \theta_{k_1}^{(1)} \theta_{k_2}^{(2)} \mathbf{L}_1^{k_1} \mathbf{X} (\mathbf{L}_2^{k_2})^\top \quad (5.12)$$

$$= \left(\sum_{k_1=0}^{K_1} \theta_{k_1}^{(1)} \mathbf{L}_1^{k_1} \right) \mathbf{X} \left(\sum_{k_2=0}^{K_2} \theta_{k_2}^{(2)} \mathbf{L}_2^{k_2} \right)^\top = \mathbf{GXF}, \quad (5.13)$$

$$\text{where } \mathbf{G} = \sum_{k_1=0}^{K_1} \theta_{k_1}^{(1)} \mathbf{L}_1^{k_1} \text{ and } \mathbf{F} = \sum_{k_2=0}^{K_2} \theta_{k_2}^{(2)} (\mathbf{L}_2^{k_2})^\top. \quad (5.14)$$

We refer to Eq. (5.13) as dimensionwise separable graph convolution (DSGC), \mathbf{G} as the *object graph convolutional filter*, and \mathbf{F} as the *attribute graph convolutional filter*. The fastest way to compute it only requires $K_1 + K_2$ matrix multiplications, much less than the $K_1 \times K_2$ matrix multiplications needed by a general 2-D spatial graph convolution. Fig. Fig. 5.2 illustrates how \mathbf{G} and \mathbf{F} can work together to learn better node representations with DSGC.

5.2 Variance Reduction by DSGC

Given a data distribution, the lowest possible error rate a classifier can achieve is the Bayes error rate [107], which is caused by the intrinsic overlap between different classes and cannot be avoided. In this section, we show that DSGC, with proper filters, can reduce the intra-class variance of the data distribution while keeping class centers roughly unchanged, hence reducing the overlap between classes and improving learning performance.

Intra-class Variance and Inter-class Variance Suppose samples x_i and their labels y_i are observations of a random vector $\mathbb{X} = [\mathbb{X}_1, \dots, \mathbb{X}_m]^\top$ and a random variable \mathbb{Y} respectively. We define the variance of random vector \mathbb{X} as the sum of the variance of each dimension \mathbb{X}_j , i.e., the trace of the covariance matrix of \mathbb{X} .

$$\text{Var}(\mathbb{X}) = \sum_{j=1}^m \text{Var}(\mathbb{X}_j) = \text{Tr}(\text{Cov}(\mathbb{X})) \quad (5.15)$$

According to the law of total variance [108], the variance of \mathbb{X} can be divided into

intra-class variance and inter-class variance:

$$\text{Var}(\mathbb{X}) = \underbrace{\text{E}[\text{Var}(\mathbb{X}|\mathbb{Y})]}_{\text{Intra-class Variance}} + \underbrace{\text{Var}(\text{E}[\mathbb{X}|\mathbb{Y}])}_{\text{Inter-class Variance}}, \quad (5.16)$$

where the conditional variance $\text{Var}(\mathbb{X}|\mathbb{Y} = k)$ is the variance of class k and the conditional expectation $\text{E}[\mathbb{X}|\mathbb{Y} = k]$ is the k -th class center. Intra-class variance (IntraVar) measures the average divergence within each class, while inter-class variance (InterVar) measures the divergence among class centers. We are interested in the IntraVar/InterVar ratio. Desired node representations should have low intra-class variance (i.e., compact and dense for each class) and high inter-classes variance (large margins between classes).

5.2.1 Intra-class Variance Reduction by Object Graph Convolution

Commonly used object graph convolution reduces variance by averaging over the neighborhood. For any node ν_i , *object graph convolution* \mathbf{GX} produces a new feature vector $\mathbf{z}_i = \sum_j G_{ij} \mathbf{x}_j$. When \mathbf{G} is a stochastic matrix, the output feature vector \mathbf{z}_i is a weighted average of the neighbours of \mathbf{x}_i . Denote by \mathbb{Z} a random vector of \mathbf{z}_i . Intuitively, as long as each node i has enough same-class neighbors, \mathbb{Z} will have a smaller $\frac{\text{IntraVar}}{\text{InterVar}}$ ratio than \mathbb{X} .

Formally, assume that objects from the same class are connected with probability r , and objects from different classes are connected with probability q , i.e., the adjacency matrix \mathbf{A}_1 of the object graph obeys the following distribution:

	if $y_i = y_j$	if $y_i \neq y_j$
$\Pr(a_{ij} \neq 0)$	r	q
$\Pr(a_{ij} = 0)$	$1 - r$	$1 - q$

We also assume that classes are balanced, i.e., $\Pr(\mathbb{Y} = k) = 1/K$ for all k . Then, we

have the following theorem with the stochastic graph filter $\mathbf{G} = \mathbf{D}^{-1}\mathbf{A}_1$.

Theorem 23. *When q is sufficiently small, the IntraVar/InterVar ratio of \mathbb{Z} is less than or equal to that of \mathbb{X} , i.e.,*

$$\frac{\mathbb{E}[\text{Var}(\mathbb{Z}|\mathbb{Y})]}{\text{Var}(\mathbb{E}[\mathbb{Z}|\mathbb{Y}])} \leq \frac{\mathbb{E}[\text{Var}(\mathbb{X}|\mathbb{Y})]}{\text{Var}(\mathbb{E}[\mathbb{X}|\mathbb{Y}])}. \quad (5.17)$$

Proof. The proof consists of two parts. In the first part, we prove that inter-class variance is unchanged after object graph convolution when q approximates 0, i.e.,

$$\lim_{q \rightarrow 0} \text{Var}(\mathbb{E}[\mathbb{Z}|\mathbb{Y}]) = \text{Var}(\mathbb{E}[\mathbb{X}|\mathbb{Y}]). \quad (5.18)$$

In the second part, we prove that intra-class variance becomes smaller after object graph convolution, i.e.,

$$\mathbb{E}[\text{Var}(\mathbb{Z}|\mathbb{Y})] \leq \mathbb{E}[\text{Var}(\mathbb{X}|\mathbb{Y})], \quad (5.19)$$

when G is a stochastic matrix.

Part 1. Inter-class variance is unchanged. Since $z_i = \sum_j G_{ij} \mathbf{x}_j$, we have

$$\begin{aligned} \mathbb{E}[z_i | y_i = k] &= \sum_j \mathbb{E}[G_{ij}] \mathbb{E}[\mathbf{x}_j] \\ &= \sum_{j, y_j = k} \mathbb{E}[G_{ij}] \mathbb{E}[\mathbf{x}_j] + \sum_{j, y_j \neq k} \mathbb{E}[G_{ij}] \mathbb{E}[\mathbf{x}_j] \\ &= \frac{\sum_{j, y_j = k} \mathbb{E}[a_{ij}] \mathbb{E}[\mathbf{x}_j] + \sum_{j, y_j \neq k} \mathbb{E}[a_{ij}] \mathbb{E}[\mathbf{x}_j]}{\sum_j \mathbb{E}[a_{ij}]} \\ &= \frac{r \sum_{j, y_j = k} \mathbb{E}[\mathbb{X}|\mathbb{Y} = k] + q \sum_{j, y_j \neq k} \mathbb{E}[\mathbf{x}_j]}{\frac{N}{K}(r - q) + Nq} \\ &= \frac{\frac{N}{K} r \mathbb{E}[\mathbb{X}|\mathbb{Y} = k] + q \sum_j \mathbb{E}[\mathbf{x}_j] - q \sum_{j, y_j = k} \mathbb{E}[\mathbf{x}_j]}{\frac{N}{K}(r - q) + Nq} \\ &= \frac{\frac{N}{K}(r - q) \mathbb{E}[\mathbb{X}|\mathbb{Y} = k] + Nq \mathbb{E}[\mathbb{X}]}{\frac{N}{K}(r - q) + Nq} \\ &= \frac{(r - q) \mathbb{E}[\mathbb{X}|\mathbb{Y} = k] + Kq \mathbb{E}[\mathbb{X}]}{(r - q) + Kq} \end{aligned} \quad (5.20)$$

When q approximates 0, Eq. (5.20) approximates $E[\mathbb{X}|\mathbb{Y} = k]$, so

$$\begin{aligned} E[Z|\mathbb{Y} = k] &= \sum_{i, y_i=k} \Pr(Z = z_i | y_i = k) E[z_i | y_i = k] \\ &= \sum_{i, y_i=k} \Pr(Z = z_i | y_i = k) E[\mathbb{X} | \mathbb{Y} = k] \\ &= E[\mathbb{X} | \mathbb{Y} = k]. \end{aligned}$$

Take variance of both sides, and we get

$$\text{Var}(E[Z|\mathbb{Y}]) = \text{Var}(E[\mathbb{X}|\mathbb{Y}]). \quad (5.21)$$

Part 2. Intra-class variance becomes smaller. Denote by $\text{Cov}(\cdot, \cdot)$ the covariance of two random variables. We have the following inequality about variance.

$$\begin{aligned} \text{Var}\left(\sum_j G_{ij} \mathbf{x}_j\right) &= \sum_j G_{ij}^2 \text{Var}(\mathbf{x}_j) + \sum_{j,l} G_{ij} G_{il} \text{Cov}(\mathbf{x}_j, \mathbf{x}_l) \\ &\leq \sum_{j,l} G_{ij} G_{il} \sqrt{\text{Var}(\mathbf{x}_j)} \sqrt{\text{Var}(\mathbf{x}_l)} \\ &= \left(\sum_j G_{ij} \sqrt{\text{Var}(\mathbf{x}_j)}\right)^2. \end{aligned} \quad (5.22)$$

Consider the variance of filtering result z_i for each sample in class k . It is less than the variance of \mathbb{X} of that class:

$$\begin{aligned} \text{Var}(z_i | y_i = k) &= \text{Var}\left(\sum_j G_{ij} \mathbf{x}_j \middle| y_j = k\right) \\ &\leq \left(\sum_j G_{ij} \sqrt{\text{Var}(\mathbf{x}_j | y_j = k)}\right)^2 && \# \text{ by inequality (5.22)} \\ &= \left(\sum_j G_{ij} \sqrt{\text{Var}(\mathbb{X} | \mathbb{Y} = k)}\right)^2 \\ &= \left(\sqrt{\text{Var}(\mathbb{X} | \mathbb{Y} = k)}\right)^2 && \# \text{ since } \sum_j G_{ij} = 1 \\ &= \text{Var}(\mathbb{X} | \mathbb{Y} = k), \end{aligned}$$

Then the variance of random vector \mathbb{Z} for each class is less than the variance of \mathbb{X} of that class:

$$\begin{aligned}\text{Var}(\mathbb{Z}|\mathbb{Y} = k) &= \sum_{i, y_i=k} \Pr(\mathbb{Z} = \mathbf{z}_i|y_i = k)\text{Var}(\mathbf{z}_i|y_i = k) \\ &\leq \text{Var}(\mathbb{X}|\mathbb{Y} = k).\end{aligned}$$

Sum them over all classes:

$$\begin{aligned}\mathbb{E}[\text{Var}(\mathbb{Z}|\mathbb{Y})] &= \sum_k \Pr(\mathbb{Y} = k)\text{Var}(\mathbb{Z}|\mathbb{Y} = k) \\ &\leq \sum_k \Pr(\mathbb{Y} = k)\text{Var}(\mathbb{X}|\mathbb{Y} = k) \\ &= \mathbb{E}[\text{Var}(\mathbb{X}|\mathbb{Y})].\end{aligned}\tag{5.23}$$

Combining Eq. (5.21) and Eq. (5.23), we prove that when q is sufficiently small,

$$\frac{\mathbb{E}[\text{Var}(\mathbb{Z}|\mathbb{Y})]}{\text{Var}(\mathbb{E}[\mathbb{Z}|\mathbb{Y}])} \leq \frac{\mathbb{E}[\text{Var}(\mathbb{X}|\mathbb{Y})]}{\text{Var}(\mathbb{E}[\mathbb{X}|\mathbb{Y}])}.\tag{5.24}$$

□

This theorem tells that under the assumption that connected nodes are most likely to be of the same class, object graph convolution \mathbf{GX} can efficiently reduce the IntraVar/InterVar ratio.

5.2.2 Intra-class Variance Reduction by Attribute Graph Convolution

A proper attribute graph convolutional filter \mathbf{F} can also reduce the IntraVar/InterVar ratio. We use the convention that the random vector \mathbb{X} is a column vector, and hence the

attribute graph convolution $\mathbf{X}\mathbf{F}$ results in a new random vector $\mathbf{F}^\top\mathbb{X}$. We also assume that the node features are mean-centered, i.e., $\mathbb{E}[\mathbb{X}] = \mathbf{0}$.

Theorem 24. *If the attribute graph convolutional filter \mathbf{F} is a doubly stochastic matrix, then the output of attribute graph convolution has an intra-class variance less than or equal to that of \mathbb{X} , i.e.,*

$$\begin{aligned} \sum_i F_{ij} = \sum_j F_{ij} = 1 \text{ and } F_{ij} \geq 0, \forall i, j \\ \Rightarrow \mathbb{E}[\text{Var}(\mathbf{F}^\top\mathbb{X}|\mathbb{Y})] \leq \mathbb{E}[\text{Var}(\mathbb{X}|\mathbb{Y})]. \end{aligned}$$

Proof. We first prove a lemma that the variance of each class will not increase after attribute graph convolution, i.e., $\text{Var}(\mathbf{F}^\top\mathbb{X}|\mathbb{Y} = k) \leq \text{Var}(\mathbb{X}|\mathbb{Y} = k)$. Denote by $\text{Cov}(\cdot)$ the covariance matrix of a random vector. Based on our definition of variance at the beginning of Section 5.2, we have

$$\begin{aligned} \text{Var}(\mathbf{F}^\top\mathbb{X}|\mathbb{Y} = k) &= \text{Tr}(\text{Cov}(\mathbf{F}^\top\mathbb{X}|\mathbb{Y} = k)) \\ &= \text{Tr}(\mathbf{F}^\top \text{Cov}(\mathbb{X}|\mathbb{Y} = k) \mathbf{F}) && \# \text{ property of covariance} \\ &= \text{Tr}(\text{Cov}(\mathbb{X}|\mathbb{Y} = k) \mathbf{F}^\top) && \# \text{ cyclic property of trace} \\ &= \sum_{ij} \text{Cov}(\mathbb{X}_i, \mathbb{X}_j|\mathbb{Y} = k) (\mathbf{F}^\top)_{ij} && \# \text{ property of trace} \\ &\leq \sum_{ij} \sqrt{\text{Var}(\mathbb{X}_i|\mathbb{Y} = k)} \sqrt{\text{Var}(\mathbb{X}_j|\mathbb{Y} = k)} (\mathbf{F}^\top)_{ij} \end{aligned}$$

Let $\boldsymbol{\sigma} \in \mathbb{R}^m$ be a vector of variances with $\sigma_i \triangleq \sqrt{\text{Var}(\mathbb{X}_i|\mathbb{Y} = k)}$, then we have

$$\begin{aligned}
\text{Var}(\mathbf{F}^\top \mathbb{X} | \mathbb{Y} = k) &\leq \sum_{ij} \sigma_i \sigma_j (\mathbf{F}^\top)_{ij} \\
&= \boldsymbol{\sigma}^\top \mathbf{F}^\top \boldsymbol{\sigma} \\
&\leq \|\boldsymbol{\sigma}\|_2^2 && \# \text{ eigenvalues of } \mathbf{F} \text{ is no more than } 1 \\
&= \sum_i \text{Var}(\mathbb{X}_i | \mathbb{Y} = k) \\
&= \text{Var}(\mathbb{X} | \mathbb{Y} = k).
\end{aligned}$$

Next, we prove the theorem with the above lemma.

$$\begin{aligned}
\mathbb{E}[\text{Var}(\mathbf{F}^\top \mathbb{X} | \mathbb{Y})] &= \sum_k \Pr(\mathbb{Y} = k) \text{Var}(\mathbf{F}^\top \mathbb{X} | \mathbb{Y} = k) \\
&\leq \sum_k \Pr(\mathbb{Y} = k) \text{Var}(\mathbb{X} | \mathbb{Y} = k) \\
&= \mathbb{E}[\text{Var}(\mathbb{X} | \mathbb{Y})]
\end{aligned}$$

□

To achieve a low IntraVar/InterVar ratio, in addition to reducing intra-class variance, we also need to keep the class centers apart after convolution. This depends on the quality of the attribute affinity graph. A good attribute affinity graph should connect attributes that share similar expectations conditioned on \mathbb{Y} . Formally, each attribute \mathbb{X}_j has K conditional expectations w.r.t. \mathbb{Y} , which are denoted as a vector $\mathbf{e}_j = (\mathbb{E}[\mathbb{X}_j | \mathbb{Y} = 1], \dots, \mathbb{E}[\mathbb{X}_j | \mathbb{Y} = K]) \in \mathbb{R}^K$. We have the following.

Theorem 25. *If $\forall F_{ij} \neq 0$, $\|\mathbf{e}_i - \mathbf{e}_j\|_2 \leq \varepsilon$, then the distance between \mathbf{e}_j and $\hat{\mathbf{e}}_j = \sum_i F_{ij} \mathbf{e}_i$ is also less than or equal to ε , i.e.,*

$$\|\mathbf{e}_i - \mathbf{e}_j\|_2 \leq \varepsilon, \forall F_{ij} \neq 0 \quad \Rightarrow \quad \|\mathbf{e}_j - \hat{\mathbf{e}}_j\|_2 \leq \varepsilon,$$

and ε can be arbitrarily small with a proper \mathbf{F} .

Proof.

$$\begin{aligned}
\|\mathbf{e}_j - \hat{\mathbf{e}}_j\|_2 &= \left\| \mathbf{e}_j - \sum_i F_{ij} \mathbf{e}_i \right\|_2 \\
&= \left\| \sum_i F_{ij} (\mathbf{e}_j - \mathbf{e}_i) \right\|_2 && \# \text{ since } \sum_i F_{ij} = 1 \\
&\leq \sum_i F_{ij} \|\mathbf{e}_j - \mathbf{e}_i\|_2 && \# \text{ Cauchy-Schwarz inequality} \\
&\leq \sum_i F_{ij} \varepsilon = \varepsilon
\end{aligned}$$

Next, we prove that there exists such an \mathbf{F} that ε is 0. This is equivalent to finding a doubly stochastic \mathbf{F} satisfying $\sum_i F_{ij} \mathbf{e}_i = \mathbf{e}_j$ for all j . Given a trivial solution $\mathbf{F} = \mathbf{I}$, this equation is solvable. In most real-world attributed networks, the number of attributes is far greater than the number of classes, so the number of variables in this linear system is greater than the number of equations. Given that it is solvable, it must have an infinite number of solutions other than \mathbf{I} . Thus, ε can be arbitrarily small with a proper \mathbf{F} . \square

By Theorem 25, the conditional expectations of each attribute (i.e., class means) may change little after attribute graph convolution, and so does the inter-class variance. Combining Theorems 24 and 25, it suggests that a proper attribute affinity graph should connect attributes that have similar class means in order to achieve a low $\frac{\text{IntraVar}}{\text{InterVar}}$ ratio and improve performance.

5.3 Unsupervised and Semi-Supervised Learning with DSGC

5.3.1 Learning Frameworks

Given an attributed graph with node feature matrix \mathbf{X} , we can learn node representations \mathbf{Z} in an unsupervised manner by applying DSGC on \mathbf{X} , i.e.,

$$\mathbf{Z} = \mathbf{G}\mathbf{X}\mathbf{F}, \quad (5.25)$$

and then perform various downstream learning tasks with the node representations \mathbf{Z} .

Unsupervised Node Clustering Any standard clustering algorithm can be applied on \mathbf{Z} for clustering as long as it is suitable for the present data. In experiments, we use the popular spectral clustering method [24, 101] along with linear kernel $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top$.

Semi-supervised Node Classification After obtaining the unsupervised node representations \mathbf{Z} , we may adopt any proper supervised classifier and train it with \mathbf{Z} and a small portion of labels for semi-supervised classification. This two-step framework is semi-supervised in nature. In experiments, we choose a multi-layer perceptron with a single hidden layer as our classifier. In addition to the two-step framework, we can also plug DSGC into existing end-to-end graph-convolution-based methods. In experiments, we improve several popular methods, including GCN, GAT, and GraphSAGE, by replacing their 1-D graph convolution with DSGC. For example, to incorporate DSGC into the vanilla GCN, we can modify the first layer propagation of GCN as

$$\mathbf{H}^{(1)} = \sigma(\mathbf{G}\mathbf{X}\mathbf{F}\mathbf{W}^{(1)}), \quad (5.26)$$

where $\mathbf{H}^{(1)}$ is the hidden units in the first layer, $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times l}$ is the trainable parameters of GCN, and σ is an activation function such as ReLU.

Importantly, Eq. (5.26) can be considered as feeding a filtered feature matrix $\mathbf{X}\mathbf{F}$ instead of the raw feature matrix \mathbf{X} to GCN. Our above analysis shows that a proper attribute graph convolutional filter \mathbf{F} can reduce intra-class variance, making $\mathbf{X}\mathbf{F}$ much easier to classify and guarantees to help train a better model. Furthermore, the parameters of GCN are freely chosen from the parameter space $\mathbf{W}^{(1)}$, while the model trained by Eq. (5.26) is restricted in a subspace $\mathbf{F}\mathbf{W}^{(1)}$. Since the chosen filter \mathbf{F} is low-pass Section 5.3.2 and thus is nearly singular, $\mathbf{F}\mathbf{W}^{(1)}$ is a subspace of $\mathbb{R}^{m \times l}$ projected by \mathbf{F} . Model parameters in this subspace are generally better in generalization performance due to the variance reduction property of \mathbf{F} . However, the model learned by Eq. (5.26) can hardly be learned by GCN, since the subspace $\mathbf{F}\mathbf{W}^{(1)}$ has measure zero, which is a tiny subset of $\mathbb{R}^{m \times l}$.

5.3.2 Implementation of Filters

Object Graph Convolutional Filter In most cases, the object graph \mathbf{A}_1 is given as part of the dataset. All we need is to design the filter. There are various graph convolutional filters available [2, 3, 14, 15], but the key principle of filter design for semi-supervised and unsupervised learning is low-pass [3, 109]. Following this principle, we use the 2-order row-normalized affinity matrix as the object graph filter, i.e.,

$$\mathbf{G} = (\mathbf{I} - \mathbf{L}_r^{(1)})^2 = (\mathbf{D}_1^{-1} \mathbf{A}_1)^2. \quad (5.27)$$

Attribute Graph Convolutional Filter A key issue in implementing DSGC is to construct a suitable attribute affinity graph \mathbf{A}_2 . Possible ways to construct attribute

affinity graphs include extracting entity relation information from existing knowledge bases, building a similarity graph from features, or identifying correlations by domain knowledge. In experiments, we evaluate our methods on text dataset as in [11, 15, 16], and leverage two suitable attribute affinity graphs for text data described below.

Positive point-wise mutual information (PPMI) is a common tool for measuring the association between two words in computational linguistics [110]. PPMI between words w_i and w_j is defined by

$$a_{ij}^{(2)} = \text{PPMI}(w_i, w_j) = \left[\log \frac{\Pr(w_i, w_j)}{\Pr(w_i) \Pr(w_j)} \right]_+, \quad (5.28)$$

where $\Pr(w_i)$ is the probability of occurrence of word w_i , and $\Pr(w_i, w_j)$ is the probability of two words occurring together. If there is a semantic relation between two words, they tend to co-occur more frequently and thus share a high PPMI value. Here, we use PPMI between words as the corresponding weights in the attribute affinity matrix A_2 and symmetrically normalize it as Kipf and Welling [15].

Word embedding based k -NN graphs. Word embedding is a collection of techniques that map vocabularies to vectors in a Euclidean space. Embeddings of words are pre-trained vectors learned from the corpus with algorithms such as GloVe [111]. Since word embeddings capture semantic relations between words [112], they can be used for constructing an attribute affinity graph. We construct a k -NN graph from the embedding vectors by some proximity metric, such as the Euclidean distance.

With the constructed attribute affinity graph A_2 , we use one-step lazy random walk filter [3] in our experiments, i.e.,

$$F = \left(\mathbf{I} - \frac{1}{2} L_s \right) = \frac{1}{2} \left(\mathbf{I} + \mathbf{D}_2^{-1/2} \mathbf{A}_2 \mathbf{D}_2^{-1/2} \right). \quad (5.29)$$

Chapter Review

In this chapter, we systematically discuss how to build 2-D graph convolution neural networks and propose a simple yet efficient dimensionwise separable 2-D graph convolution (DSGC) for unsupervised and semi-supervised learning on graphs. We demonstrate that by exploiting attribute relations in addition to object relations, DSGC can learn better node representations than existing methods based on the regular 1-D graph convolution, leading to promising performance on node classification and clustering tasks.

Chapter 6

Empirical Study

In this chapter, we first demonstrate the effectiveness of our proposed methods with 1-D graph convolution, including GLP, IGCN, and AGC, on semi-supervised classification and unsupervised clustering tasks. Then, we demonstrate the effectiveness of our proposed dimensionwise separable 2-D graph convolution and its advantages over regular 1-D graph convolution, also on semi-supervised classification and unsupervised clustering.

6.1 Semi-supervised Classification

To validate the performance of our methods GLP and IGCN, we test our methods GLP and IGCN on three tasks.* 1) Semi-supervised document classification on citation networks, where nodes are documents and edges are citation links. The goal is to classify the documents into topics with only a few labeled documents. 2) Semi-supervised entity

*Code is available at <https://github.com/liqimai/Efficient-SSL>

classification on a knowledge graph. A bipartite graph is extracted from the knowledge graph [8], and there are two kinds of nodes: entity and relation, where the edges are between the entity and relation nodes. The goal is to classify the entity nodes with only a few labeled entity nodes. 3) Semi-supervised handwritten digit recognition. The goal is to recognize the handwritten digits with only a few labeled digits.

6.1.1 Datasets and Settings

We evaluate our methods on four citation networks – Cora, CiteSeer, PubMed [25], and Large Cora, one knowledge graph – NELL [113], and one handwritten digit recognition — MNIST [114]. The dataset statistics are summarized in Table 6.1.

Citation networks [25] Citation networks are networks that record documents’ citation relationships. In citation networks, vertices are documents, and edges are citation links. A pair of vertices are connected by an undirected edge if and only if one cites another. Besides the graph structure, each document is associated with a feature vector that encodes the document content. In three citation networks we tested on, CiteSeer, Cora, and PubMed, features are 0/1 vectors that have the same length as dictionary size and indicate words appearing in documents. The statistics of datasets are summarized in Table 6.1. On citation networks, we test two scenarios – 4 labels per class and 20 labels per class.

Never Ending Language Learning (NELL) [113] NELL is a knowledge graph introduced by Carlson et al., and processed by Yang et al.. In [8], Yang et al. extracted an

entity classification dataset from NELL and changed the knowledge graph into a single relation graph as below. For each relation type r , they create two new vertices r_1 and r_2 in the graph. For each triplet (e_1, r, e_2) , they create two edges (e_1, r_1) and (e_2, r_2) . We follow [15] to extend the number of features by assigning a unique one-hot representation for every relation node, effectively resulting in a 61,278-dim sparse feature vector per node. Dataset statistics are also in Table 6.1. On NELL, we test three scenarios – 0.1%, 1%, and 10% label rates.

MNIST [114] MNIST contains 70,000 images of handwritten digits from 0 to 9 of size 28×28 . Each image is represented by a dense 784-dimensional vector where each element is a gray intensity pixel value. A 20-NN graph is constructed based on the Euclidean distance between images. If the i -th image is within the j -th image’s 20 nearest neighbors or vice versa, then $w_{ij} = w_{ji} = 1$, otherwise $w_{ij} = w_{ji} = 0$.

Table 6.1: Dataset statistics.

Dataset	Type	Vertices	Edges	Classes	Features
CiteSeer	Citation network	3,327	4,732	6	3703
Cora	Citation network	2,708	5,429	7	1433
PubMed	Citation network	19,717	44,338	3	500
NELL	Knowledge graph	65,755	266,144	210	5414
MNIST	Handwritten Digits	70,000	2,060,504	10	784

Baselines. We compare GLP and IGCN with the state-of-the-art semi-supervised classification methods: manifold regularization (ManiReg) [56], semi-supervised embedding (SemiEmb) [40], DeepWalk [74], iterative classification algorithm (ICA) [25], Planetoid [8], graph attention networks (GAT) [16], multi-layer perceptron (MLP), LP [97],

and GCN [15].

Settings. We use MLP as the classifier of GLP and test GLP and IGCN with RNM and AP filters. We follow Kipf and Welling [15] to use a two-layer structure for all neural networks, including MLP, GCN, and IGCN. Guided by our analysis in Section 4.3, the filter parameters k and α should be set large if the label rate is low, and should be set small if the label rate is high. Specifically, when 20 labels per class on citation networks are available, or 10% entities of NELL are labeled, we set $k = 5$ for RNM and $\alpha = 10$ for AP filters in GLP. In other scenarios with fewer labels, we set $k = 10, \alpha = 20$ for GLP. The k, α chosen for IGCN is equal to the above k, α divided by the number of layers – 2. We follow Kipf and Welling [15] to set the parameters of MLP, GCN, and IGCN: for citation networks, we use a two-layer network with 16 hidden units, 0.01 learning rate, 0.5 dropout rate, and 5×10^{-4} L2 regularization, except that the hidden layer is enlarged to 64 units for Large Cora; for NELL, we use a two-layer network with 64 hidden units, 0.01 learning rate, 0.1 dropout rate, and 1×10^{-5} L2 regularization. For a more fair comparison with different baselines, we do not use a validation set for model selection as in Kipf and Welling [15]. Instead, we select the model with the lowest training loss in 200 steps. All results are averaged over 50 random splits of the dataset. We set α of LP to 100 for citation networks and 10 for NELL. Parameters of GAT are the same as Velickovic et al. [16]. Results of other baselines are taken from Yang et al. [8], Kipf and Welling [15].

Table 6.2: Classification Accuracy with 20 labels per class.

	Cora				CiteSeer				PubMed			
	SVM	DT	LR	MLP	SVM	DT	LR	MLP	SVM	DT	LR	MLP
GLP(Raw)	22.0	45.0	54.6	56.3	50.0	42.2	58.8	57.0	63.2	58.0	68.9	68.0
GLP(RNM)	55.8	64.1	74.8	80.2	62.8	50.9	66.8	68.2	69.5	65.8	73.7	76.3
GLP(RW)	55.8	65.2	74.8	80.3	62.9	50.5	67.0	68.3	69.5	67.8	73.8	76.3
GLP(AP)	51.8	63.7	74.6	81.0	62.0	52.4	67.3	69.3	70.4	67.6	74.2	77.2

6.1.2 Results and Discussion

Citation Networks and Knowledge Graph

GLP with Various Classifiers To demonstrate the benefit of GLP, our first experiment is to compare the raw features with the filtered features in training different supervised classifiers, including support vector machine (SVM), decision tree (DT), logistic regression (LR), and multilayer perceptron (MLP). The results are summarized in Table 6.2, where ‘Raw’ means using raw features. We can see that there is a huge improvement in classification accuracy for all classifiers and datasets with the smooth features produced by the three filters we proposed. This clearly demonstrates the advantage of filtered features over raw features.

GLP & IGCN v.s. Baselines The results of citation networks and NELL are summarized in Table 6.3, where the top 3 classification accuracies are highlighted in bold. Overall, GLP and IGCN perform best. Especially when the label rates are very low, they significantly outperform the baselines. Specifically, on citation networks, with 20 labels per class, GLP and IGCN perform slightly better than GCN and GAT but outperform

Table 6.3: Classification accuracy on citation networks and NELL.

Label rate	20 labels per class				4 labels per class				10%	1%	0.1%
	Cora	CiteSeer	PubMed	L-Cora	Cora	CiteSeer	PubMed	L-Cora	NELL		
ManiReg	59.5	60.1	70.7	-	-	-	-	-	63.4	41.3	21.8
SemiEmb	59.0	59.6	71.7	-	-	-	-	-	65.4	43.8	26.7
DeepWalk	67.2	43.2	65.3	-	-	-	-	-	79.5	72.5	58.1
ICA	75.1	69.1	73.9	-	62.2	49.6	57.4	-	-	-	-
Planetoid	75.7	64.7	77.2	-	43.2	47.8	64.0	-	84.5	75.7	61.9
GAT	79.5	68.2	76.2	67.4	66.6	55.0	64.6	46.4	-	-	-
MLP	55.1	55.4	69.5	48.0	36.4	38.0	57.0	30.8	63.6	41.6	16.7
LP	68.8	48.0	72.6	52.5	56.6	39.5	61.0	37.0	84.5	75.1	65.9
GCN	79.9	68.6	77.6	67.7	65.2	55.5	67.7	48.3	81.6	63.9	40.7
IGCN(RNM)	80.9	69.0	77.3	68.9	70.3	57.4	69.3	52.1	85.9	76.7	66.0
IGCN(AP)	81.1	69.3	78.2	69.2	70.3	58.0	70.1	52.5	85.4	75.7	67.4
GLP(RNM)	80.3	68.8	77.1	68.4	68.0	56.7	68.7	51.1	86.0	76.1	65.4
GLP(AP)	80.8	69.3	78.1	69.0	67.5	57.3	69.7	51.6	80.3	67.4	55.2

other baselines by a considerable margin. With 4 labels per class, GLP and IGCN significantly outperform all the baselines, demonstrating their label efficiency. On NELL, GLP and IGCN with the RNM filter and IGCN with the AP filter slightly outperform two very strong baselines – LP and Planetoid, and outperform other baselines by a large margin.

Compared with methods that only use graph information, e.g., LP and DeepWalk, the large performance gains of GLP and IGCN clearly come from leveraging both graph and feature information. Compared with methods that use both graph and feature information, e.g., GCN and GAT, GLP and IGCN are much more label efficient. The reason is that they allow stronger filters to extract higher-level data representations to improve performance when label rates are low, which can be easily achieved by increasing the filter parameters k and α . But this cannot be easily achieved in the original GCN. As explained in Section 4.2, to increase smoothness, GCN needs to stack many layers, but a deep GCN

Table 6.4: Running time on citation networks and NELL.

Label rate	20 labels per class				4 labels per class				10%	1%	0.1%
	Cora	CiteSeer	PubMed	L-Cora	Cora	CiteSeer	PubMed	L-Cora	NELL		
MLP	0.6s	0.6s	0.6s	0.8s	0.6s	0.5s	0.6s	0.6s	2.1s	1.1s	1.0s
LP	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.7s	1.8s	1.9s
GCN	1.3s	1.7s	9.6s	7.5s	1.3s	1.7s	9.8s	7.4s	33.5s	33.5s	33.2s
IGCN _(RNM)	1.2s	1.7s	10.0s	7.9s	1.3s	1.7s	10.3s	8.1s	42.4s	44.0s	46.6s
IGCN _(AP)	2.2s	2.6s	11.9s	11.0s	3.0s	3.4s	13.6s	13.6s	77.9s	116.0s	116.0s
GLP _(RNM)	0.9s	1.0s	0.6s	1.8s	0.7s	0.8s	0.6s	1.1s	35.9s	37.3s	38.5s
GLP _(AP)	1.0s	1.2s	0.7s	2.4s	0.8s	1.1s	0.8s	2.3s	57.4s	76.6s	78.6s

is difficult to train with few labels.

Training time Table 6.4 reports the total training time of the methods tested by us. All neural-network-based methods are trained for 200 epochs. We can see that GLP with the RNM filter runs much faster than GCN in most cases, and IGCN with the RNM filter has similar time efficiency as GCN.

Results of Image Data

The results of MNIST are summarized in Table 6.5. It can be seen clearly that for every case, our methods outperform all baselines significantly. The error rates are decreased by 38-62% when compared with GCN, by 37-68% when compared with CNN, and by 10-39% when compared with LP. The performance gain comes from two factors: 1) the smooth features produced by the graph filters, as shown in Fig. 6.1; 2) the powerful classifier CNN. This again shows the flexibility of the GLP framework – it can be coupled with any suitable classifier based on the data type. In contrast, GCN cannot take advantage of CNN in dealing with image data. Among the baselines, LP performs the best, but as

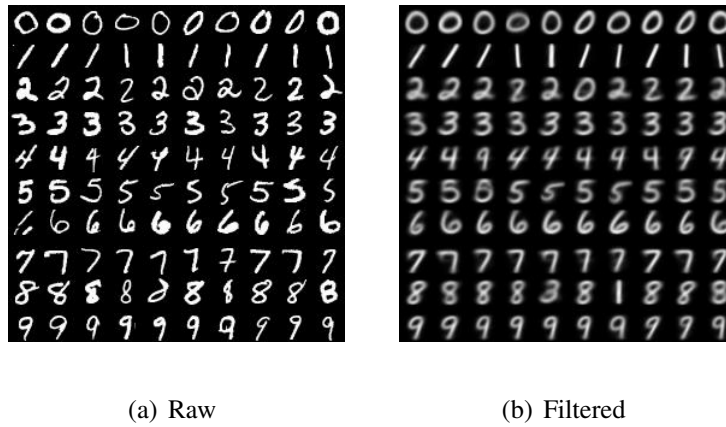


Figure 6.1: Visualizing raw and filtered hand-written digits.

Table 6.5: Classification Accuracy on MNIST.

Labels per class	10	20	30	40	50
GCN	83.3	85.6	90.1	90.6	91.3
LP	91.3	91.2	91.4	91.9	92.1
MLP	74.9	81.3	83.1	85.2	86.9
CNN	79.7	83.4	88.0	90.2	91.6
RNM+CNN	93.7	93.7	94.4	94.7	95.2
RW+CNN	93.5	93.8	94.6	94.7	95.1
AP+CNN	92.2	93.1	93.9	94.5	94.7

it cannot fully leverage the image features, its performance grows slowly as the number of labels increases.

6.2 Unsupervised Clustering

6.2.1 Datasets and Settings

We evaluate our method, adaptive graph convolution (AGC), on four benchmark attributed networks. *Cora*, *CiteSeer*, and *Pubmed* [87] are citation networks where nodes correspond to publications and are connected if one cites the other. *Wiki* [80] is a webpage network where nodes are webpages and are connected if one links to the other. The nodes in *Cora* and *CiteSeer* are associated with binary word vectors, and the nodes in *Pubmed* and *Wiki* are associated with tf-idf weighted word vectors. Table 6.6 summarizes the details of the datasets.

Cora A citation network where the nodes correspond to the publications and are connected by an undirected edge if one cites the other. There are 2708 nodes, 5429 edges in Cora, and each node is associated with a binary vector of 1433 dimensions. The nodes are classified into 7 categories. [87, 88]

CiteSeer A citation network with 3327 nodes and 4732 edges. The nodes correspond to the publications, which are represented by binary vectors of 3703 dimensions. The nodes are classified into 6 categories.

Pubmed A citation network with 19717 nodes and 44338 edges. The nodes correspond to the publications with 500 dimension feature vectors and are divided into 3 classes.

Dataset	#Nodes	#Edges	#Features	#Classes
Cora	2708	5429	1433	7
CiteSeer	3327	4732	3703	6
Pubmed	19717	44338	500	3
Wiki	2405	17981	4973	17

Table 6.6: Dataset statistics.

Wiki A webpage network where the nodes are webpages and are connected if one links to the other. There are 2405 nodes and 17981 edges in Wiki. The nodes are represented by 4973 dimension feature vectors and are divided into 17 classes.

Baselines and Evaluation Metrics

We compare AGC with three kinds of clustering methods. 1) Methods that only use node features: k -means and spectral clustering (Spectral-f) that construct a similarity matrix from the node features by the linear kernel. 2) Structural clustering methods that only use graph structures: spectral clustering (Spectral-g) that takes the node adjacency matrix as the similarity matrix, DeepWalk [74], and deep neural networks for graph representations (DNGR) [77]. 3) Attributed graph clustering methods that utilize both node features and graph structures: graph autoencoder (GAE) and graph variational autoencoder (VGAE) [87], marginalized graph autoencoder (MGAE) [88], and adversarially regularized graph autoencoder (ARGE) and variational graph autoencoder (ARVGE) [89].

To evaluate the clustering performance, we adopt three widely used performance measures [102]: clustering accuracy (Acc), normalized mutual information (NMI), and macro

F1-score (F1).

Parameter Settings

For AGC, we set $\text{max_iter} = 60$. For other baselines, we follow the parameter settings in the original papers. In particular, for DeepWalk, the number of random walks is 10, the number of latent dimensions for each node is 128, and the path length of each random walk is 80. For DNGR, the autoencoder is of three layers with 512 neurons and 256 neurons in the hidden layers, respectively. For GAE and VGAE, we construct encoders with a 32-neuron hidden layer and a 16-neuron embedding layer and train the encoders for 200 iterations using the Adam optimizer with a learning rate of 0.01. For MGAE, the corruption level p is 0.4, the number of layers is 3, and the parameter λ is 10^{-5} . For ARGE and ARVGE, we construct encoders with a 32-neuron hidden layer and a 16-neuron embedding layer. The discriminators are built by two hidden layers with 16 and 64 neurons, respectively. On *Cora*, *CiteSeer* and *Wiki*, we train the autoencoder-related models of ARGE and ARVGE for 200 iterations with the Adam optimizer, with the encoder learning rate and the discriminator learning rate both as 0.001; on *Pubmed*, we train them for 2000 iterations with the encoder learning rate 0.001 and the discriminator learning rate 0.008.

6.2.2 Result Analysis

We run each method 10 times for each dataset and report the average clustering results in Table 6.7, where the top 2 results are highlighted in bold. The observations are as follows.

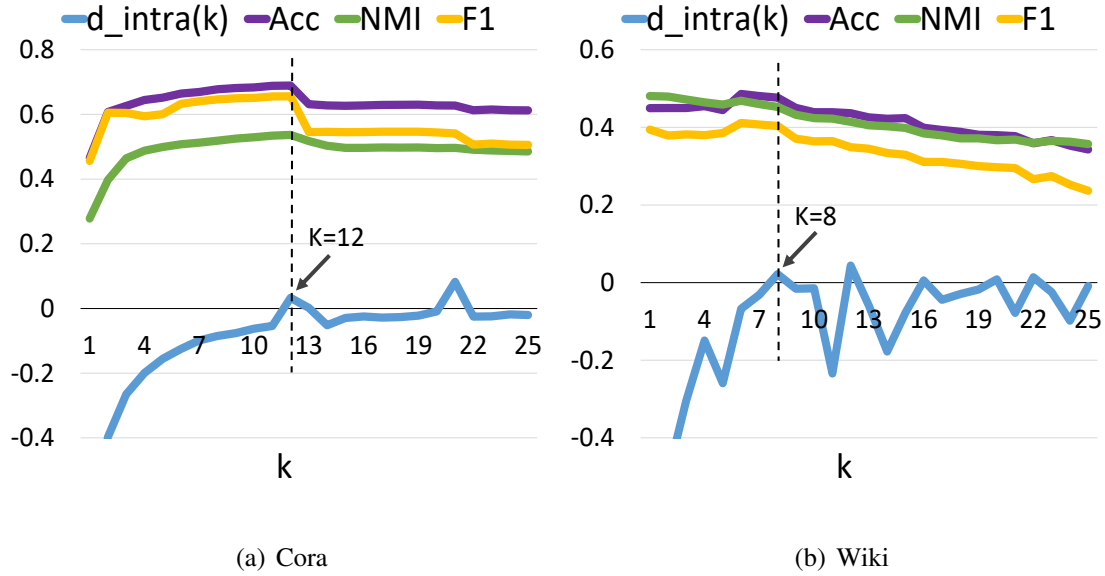


Figure 6.2: $\Delta_{intra}(k)$ and clustering performance w.r.t. k .

1) AGC consistently outperforms the clustering methods that only exploit either node features or graph structures by a very large margin due to the clear reason that AGC makes better use of available data by integrating both kinds of information, which can complement each other and greatly improve clustering performance.

2) AGC consistently outperforms existing attributed graph clustering methods that use both node features and graph structures. This is because AGC can better utilize graph information than these methods. In particular, GAE, VGAE, ARGE, and ARVGE only exploit the 2-hop neighborhood of each node to aggregate information, and MGAE only exploits the 3-hop neighborhood. In contrast, AGC uses k -order graph convolution with an automatically selected k to aggregate information within the k -hop neighborhood to produce better feature representations for clustering.

3) AGC outperforms the strongest baseline MGAE by a considerable margin on *Cora*, *CiteSeer*, and *Pubmed*, and is comparable to MGAE on *Wiki*. This is probably because *Wiki* is more densely connected than others, and aggregating information within the 3-

Methods	Input	Cora			CiteSeer			Pubmed			Wiki		
		Acc%	NMI%	F1%	Acc%	NMI%	F1%	Acc%	NMI%	F1%	Acc%	NMI%	F1%
<i>k</i> -means	Feature	34.65	16.73	25.42	38.49	17.02	30.47	57.32	29.12	57.35	33.37	30.20	24.51
Spectral-f	Feature	36.26	15.09	25.64	46.23	21.19	33.70	59.91	32.55	58.61	41.28	43.99	25.20
Spectral-g	Graph	34.19	19.49	30.17	25.91	11.84	29.48	39.74	3.46	51.97	23.58	19.28	17.21
DeepWalk	Graph	46.74	31.75	38.06	36.15	9.66	26.70	61.86	16.71	47.06	38.46	32.38	25.74
DNGR	Graph	49.24	37.29	37.29	32.59	18.02	44.19	45.35	15.38	17.90	37.58	35.85	25.38
GAE	Both	53.25	40.69	41.97	41.26	18.34	29.13	64.08	22.97	49.26	17.33	11.93	15.35
VGAE	Both	55.95	38.45	41.50	44.38	22.71	31.88	65.48	25.09	50.95	28.67	30.28	20.49
MGAE	Both	63.43	45.57	38.01	63.56	39.75	39.49	43.88	8.16	41.98	50.14	47.97	39.20
ARGE	Both	64.00	44.90	61.90	57.30	35.00	54.60	59.12	23.17	58.41	41.40	39.50	38.27
ARVGE	Both	63.80	45.00	62.70	54.40	26.10	52.90	58.22	20.62	23.04	41.55	40.01	37.80
AGC	Both	68.92	53.68	65.61	67.00	41.13	62.48	69.78	31.59	68.72	47.65	45.28	40.36

Table 6.7: Clustering performance.

hop neighborhood may be enough for feature smoothing. But it is not good enough for larger and sparser networks such as *CiteSeer*, and *Pubmed*, on which the performance gaps between AGC and MGAE are wider. AGC deals with the diversity of networks well via adaptively selecting a good k for different networks.

To demonstrate the validity of the proposed selection criterion $\Delta_{\text{intra}}(t-1) > 0$, we plot $\Delta_{\text{intra}}(k)$ and the clustering performance w.r.t. k on *Cora* and *Wiki* respectively in Fig. 6.2. One can see that when $\Delta_{\text{intra}}(k) > 0$, the corresponding Acc, NMI, and F1 values are the best or close to the best, and the clustering performance declines afterward. It shows that the selection criterion can reliably find a good cluster partition and prevent over-smoothing. The selected k for *Cora*, *CiteSeer*, *Pubmed*, and *Wiki* are 12, 55, 60, and 8 respectively, which are close to the best $k \in [0, 60]$ – 12, 35, 60, and 6 on these datasets respectively, demonstrating the effectiveness of the proposed selection criterion.

AGC is quite stable. The standard deviations of Acc, NMI and F1 are 0.17%, 0.42%, 0.01% on *Cora*, 0.24%, 0.36%, 0.19% on *CiteSeer*, 0.00%, 0.00%, 0.00% on *Pubmed*, and 0.79%, 0.17%, 0.20% on *Wiki*, all very small.

The running time of AGC (in Python, with NVIDIA Geforce GTX 1060 6GB GPU) on *Cora*, *CiteSeer*, *Pubmed*, and *Wiki* is 5.78, 62.06, 584.87, and 10.75 seconds respectively. AGC is a little slower than GAE, VGAE, ARGE, and ARVGE on *CiteSeer* but is more than twice faster on the other three datasets. This is because AGC does not need to train the neural network parameters as in these methods and hence is more time efficient even with a relatively large k .

6.3 2-D Convolution v.s. 1-D Convolution

To demonstrate the superiority of 2D convolution over 1D convolution, we conduct extensive experiments in semi-supervised node classification and node clustering on seven real-world networks, including 20 Newsgroups (20 NG) [115], Large Cora (L-Cora) [3, 116], Wikispeedia (Wiki) [117, 118], and four subsets of WebKB (Cornell, Texas, Wisconsin, and Washington).[†]

6.3.1 Datasets

The statistics of all datasets are summarized in Table 6.8, where the last row shows the intra-class edge ratio of the object link graph of each dataset, which can reflect the quality

[†]Note that we did not use the “Cora”, “CiteSeer” and “PubMed” datasets as in [8, 15, 25], since the attribute (word) lists are not provided.

of the graph.

20 Newsgroups (20 NG) [115] is an email discussion group, where each object is an email, and there are 18846 emails in total. Each email is represented by a 11697-dimension tf-idf feature vector. Two emails are connected by an edge if they reply to the same one.

Wikipedia (Wiki) [117, 118] is a webpage network in which the objects are 3767 Wikipedia web pages, and the edges are web hyperlinks. Each webpage is described by a 18316-dimension tf-idf vector. We removed several tiny classes, so the web pages distribute more evenly across the remaining 9 categories.

Large Cora (L-Cora) [116] is a citation network in which the objects are computer science research papers represented by 3780 dimensions of tf-idf values. Two papers are connected by an undirected edge if and only if one cites the other. These citation links form an object graph. After removing the papers that belong to no topic and those with no authors or title, a subset of 11881 papers is obtained [119]. We name this dataset “Large Cora” to distinguish it from the “Cora” dataset with 2708 papers used in [8, 15, 25].

WebKB [‡] is a webpage dataset collected by Carnegie Mellon University. We use its four sub-datasets: Cornell, Texas, Wisconsin, and Washington. The objects are web pages, and the edges are hyperlinks between them. The web pages are represented by tf-idf feature vectors and manually classified into five categories: student, project, course, staff, and faculty.

[‡]<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/>

Table 6.8: Dataset statistics.

Dataset	#vertices	#edges	#cls	#features	ratio of intra-class edges
20 NG	18,846	147,034	20	11,697	96.8%
Wiki	3,767	129,597	9	18,316	38.0%
L-Cora	11,881	64,898	10	3,780	76.5%
Corn.	247	384	5	3371	23.7%
Texa.	255	205	5	3371	19.8%
Wisc.	320	721	5	3371	26.3%
Wash.	265	417	5	3371	40.3%

6.3.2 Variance Reduction and Visualization

First of all, to verify our analysis in Section 5.2, we illustrate the variance reduction effect of both object graph convolution and attribute graph convolution. As shown in Fig. 6.4, 1-D graph convolution (GX or XF) already greatly reduces the IntraVar/InterVar ratio, and 2-D graph convolution (GXF) reduces it even further.

In Fig. 6.3, we visualize the results of performing graph convolution on the object features of 20 NG by t-SNE. It can be seen that both object graph convolution and attribute graph convolution can successfully reduce the overlap among classes, and 2-D graph convolution (DSGC) is more effective than 1-D.

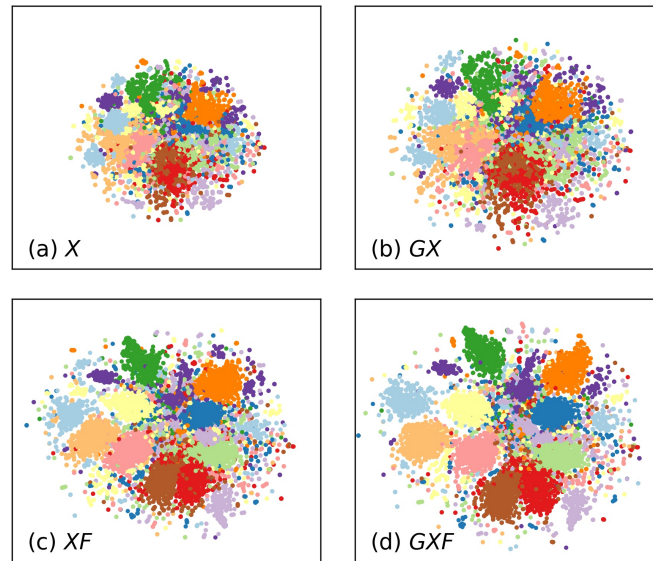


Figure 6.3: t-SNE visualization of the “20 Newsgroups” dataset. (a) Raw features; (b) Filtered by the regular object graph convolution; (c) Filtered by our proposed attribute graph convolution; (d) Filtered by our proposed dimensionwise separable graph convolution (DSGC).

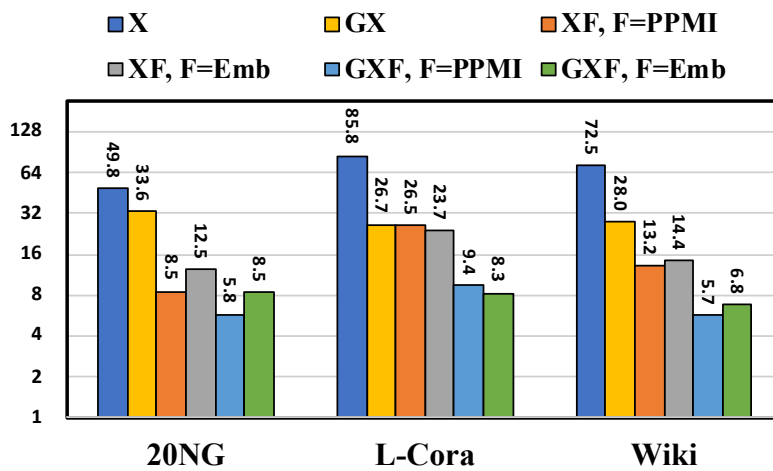


Figure 6.4: IntraVar/InterVar ratios. The lower, the better.

6.3.3 Semi-supervised Node Classification

Baselines We compare DSGC with the following baselines: label propagation (LP) [97], multi-layer perceptron (MLP), graph convolutional networks (GCN) [15], generalized label propagation (GLP) [3], GraphSAGE [11], graph attention networks (GAT) [16], deep graph infomax (DGI) [120], GCNII [121], jumping knowledge networks (JK-MaxPool & JK-Concat) [122], and GRAND [123]. We also try to improve GCN, GAT, and GraphSAGE by DSGC as described in Section Section 5.3.1. Our methods with mere object graph filter (\mathcal{G}) or mere attribute graph filter (\mathcal{F}) are also tested for the purpose of ablation study. PPMI and Emb denote attribute affinity graphs constructed by positive point-wise mutual information and word embedding, respectively, as described in Section 5.3.2).

Settings For 20 NG, L-Cora, and Wiki, we test two scenarios – 20 labels/class and 5 labels/class. We follow GCN [15] and many others to set aside a validation set containing 500 samples for hyper-parameter tuning. For the four small sub-datasets of WebKB, we randomly split them into 5/30/65% as train/valid/test set and ensure each class has at least 1 label. Hyper-parameters of all methods, including ours and baselines, are tuned by grid search according to validation. The reported results of all methods are averaged over 50 runs.

Performance Classification accuracies are summarized in Table 6.9 and 6.10. The top 2 accuracies are highlighted. Results of improved GAT, GCN, and GraphSAGE are shown in Table 6.11. The following observations can be made. Firstly, object graph convolution (\mathcal{G}) does not always help. On 20 NG and L-Cora, methods based

Table 6.9: Classification accuracy on 20 NG, L-Cora, and Wiki.

Dataset			20 NG		L-Cora		Wiki	
	G	F	20 labels/cls.	5 labels/cls.	20 labels/cls.	5 labels/cls.	20 labels/cls.	5 labels/cls.
MLP	\times	\times	63.76 \pm 0.17	38.67 \pm 0.38	52.97 \pm 0.41	39.56 \pm 0.85	67.23 \pm 0.25	54.41 \pm 0.66
LP [46]	\checkmark	\times	16.39 \pm 0.20	8.62 \pm 0.20	55.77 \pm 0.97	38.97 \pm 3.15	9.53 \pm 0.05	10.54 \pm 0.19
GLP [3]	\checkmark	\times	74.99 \pm 0.11	52.62 \pm 0.45	68.95 \pm 0.29	56.42 \pm 0.85	60.05 \pm 0.11	48.45 \pm 0.54
GCN [15]	\checkmark	\times	76.25 \pm 0.11	53.78 \pm 0.49	67.75 \pm 0.33	54.27 \pm 0.82	59.81 \pm 0.30	47.93 \pm 0.56
GAT [16]	\checkmark	\times	76.33 \pm 0.16	56.02 \pm 0.57	68.88 \pm 0.78	56.89 \pm 1.53	50.97 \pm 0.54	46.99 \pm 0.83
DGI [120]	\checkmark	\times	73.34 \pm 0.27	66.57 \pm 0.63	61.39 \pm 0.50	54.77 \pm 1.24	49.70 \pm 1.63	43.64 \pm 1.89
GraphSAGE [11]	\checkmark	\times	65.73 \pm 0.17	42.48 \pm 0.77	57.28 \pm 0.71	46.79 \pm 1.91	65.52 \pm 0.62	48.81 \pm 0.76
GCNII [121]	\checkmark	\times	77.41 \pm 0.12	58.10 \pm 0.85	68.18 \pm 0.19	57.02 \pm 0.55	43.65 \pm 1.03	35.98 \pm 4.45
JK-MaxPool [122]	\checkmark	\times	71.00 \pm 0.19	49.09 \pm 0.27	67.44 \pm 0.18	51.63 \pm 0.52	45.26 \pm 0.37	44.13 \pm 0.38
JK-Concat [122]	\checkmark	\times	72.24 \pm 0.13	49.76 \pm 0.27	67.47 \pm 0.19	51.96 \pm 0.46	47.24 \pm 0.30	45.21 \pm 0.29
GRAND [123]	\checkmark	\times	74.45 \pm 0.72	57.97 \pm 2.79	69.30 \pm 0.59	52.12 \pm 0.73	62.25 \pm 0.93	47.17 \pm 3.38
DSGC (GX)	\checkmark	\times	75.60 \pm 0.13	53.84 \pm 0.46	67.74 \pm 0.30	55.67 \pm 0.72	58.73 \pm 0.34	47.34 \pm 0.54
DSGC (XF)	\times	Emb	66.27 \pm 0.13	48.04 \pm 0.38	58.70 \pm 0.30	46.41 \pm 0.55	69.76 \pm 0.20	59.76 \pm 0.58
	\times	PPMI	75.36 \pm 0.11	59.61 \pm 0.34	61.01 \pm 0.23	48.31 \pm 0.62	69.91 \pm 0.21	60.13 \pm 0.61
DSGC (GXF)	\checkmark	Emb	76.53 \pm 0.15	59.91 \pm 0.31	69.81 \pm 0.26	58.63 \pm 0.75	60.50 \pm 0.26	49.69 \pm 0.56
	\checkmark	PPMI	81.69 \pm 0.12	68.94 \pm 0.32	70.20 \pm 0.24	59.43 \pm 0.68	58.84 \pm 0.26	48.51 \pm 0.54

* \checkmark and \times indicate using/not using G or F .

on it, like DSGC (GX), GCN, and GAT, all outperform MLP significantly. However, on Wiki and the four subsets of WebKB, object graph convolution severely harms the performance. This is because the hyperlink graphs are highly noisy. Only a small portion of edges connect nodes of the same class, much lower than that of 20 NG (96.8%) and L-Cora (76.5%) (see Table 6.8). This shows the limitation of object graph convolution. Secondly, attribute graph convolution works. As shown in Table 6.9, DSGC with mere F already outperforms MLP significantly. Especially, on Wiki and WebKB, object graph convolution fails while attribute graph convolution is still effective. Thirdly, 2-D graph convolution is useful. On datasets with good object link graphs like 20 NG and L-Cora, DSGC with both G and F performs much better than with either one of them only. Especially, DSGC (GXF) with PPMI achieves the best performance among all methods.

Table 6.10: Classification accuracy on four subsets of WebKB.

Method	F	Corn.	Texa.	Wisc.	Wash.
GCN [15]	\times	50.25 \pm 0.66	60.31 \pm 1.06	52.74 \pm 0.92	53.83 \pm 1.36
GLP [3]	\times	50.86 \pm 0.75	59.40 \pm 1.27	55.28 \pm 0.97	55.87 \pm 1.32
GAT [16]	\times	51.21 \pm 1.40	60.06 \pm 1.08	52.92 \pm 1.18	56.85 \pm 2.00
GRAND [123]	\times	49.01 \pm 1.04	57.38 \pm 0.26	49.30 \pm 0.22	42.09 \pm 1.25
DSGC (GX)	\times	51.22 \pm 0.77	59.35 \pm 1.26	56.26 \pm 0.93	55.77 \pm 1.30
DSGC (XF)	Emb	62.14 \pm 1.02	68.00 \pm 0.75	73.50 \pm 0.81	65.78 \pm 1.27
	PPMI	60.10 \pm 0.91	67.34 \pm 0.94	72.88 \pm 0.78	65.40 \pm 1.35
DSGC (GXF)	Emb	53.02 \pm 0.67	61.89 \pm 0.94	58.64 \pm 1.12	59.05 \pm 1.21
	PMI	52.35 \pm 0.52	61.83 \pm 0.91	56.40 \pm 1.04	57.01 \pm 1.20

Table 6.11: Baselines improved by DSGC.

Methods	F	20 NG	L-Cora	Wiki
GAT [16]	\times	76.33 \pm 0.16	68.88 \pm 0.78	50.97 \pm 0.54
	PPMI	78.01 \pm 0.30	67.38 \pm 0.65	55.43 \pm 0.51
GCN [15]	\times	76.25 \pm 0.11	67.75 \pm 0.33	59.81 \pm 0.30
	PPMI	81.60 \pm 0.10	67.87 \pm 0.25	61.33 \pm 0.28
GraphSAGE [11]	\times	65.73 \pm 0.17	57.28 \pm 0.71	65.52 \pm 0.62
	PPMI	76.27 \pm 0.33	60.23 \pm 1.81	67.26 \pm 0.52

* \times indicates not using F .

On datasets with bad object link graphs such as Wiki and the four subsets of WebKB, DSGC with both G and F improves upon DSGC with mere G and outperforms most baselines. Especially, DSGC with mere F achieves the best performance, which improves upon the best baseline by 3.63% and 5.72% in absolute accuracy in two scenarios.

Remarkably, it can be seen from Table 6.11 that by incorporating DSGC, the performance of baselines including GCN, GAT, and GraphSAGE is improved substantially in most cases, which again confirms that attribute graph convolution is a useful complement to object graph convolution.

6.3.4 Node Clustering

Baselines We test the proposed node clustering method with DSGC (Section 5.3.1) with or without G and F in five cases, and compare them with existing strong baselines including GAE and VGAE [87], MGAE [88], ARGE and ARVGE [89], and AGC [2]. We also compare them with the spectral clustering (Spectral) method that operates on a similarity graph constructed by a linear kernel.

Performance We adopt two widely-used clustering measures [102]: clustering accuracy (Acc) and normalized mutual information (NMI), and the results are shown in Table 6.12 with the top 2 results highlighted. We can make the following observations. 1) Attribute graph convolution is highly effective. On 20 NG, DSGC ($\mathbf{X}\mathbf{F}$) with PPMI outperforms most baselines by a very large margin. On Wiki, DSGC ($\mathbf{X}\mathbf{F}$) with PPMI or Emb significantly outperforms all the baselines. 2) 2-D graph convolution is beneficial, as validated in the classification experiments. On 20 NG, DSGC ($\mathbf{G}\mathbf{X}\mathbf{F}$) with PPMI can further improve upon the already very strong performance of DSGC ($\mathbf{X}\mathbf{F}$) with PPMI and performs the best; On L-Cora, DSGC ($\mathbf{G}\mathbf{X}\mathbf{F}$) with PPMI or Emb improves upon either DSGC ($\mathbf{G}\mathbf{X}$) or DSGC ($\mathbf{X}\mathbf{F}$) and outperforms most baselines significantly. On Wiki, DSGC ($\mathbf{X}\mathbf{F}$) performs better than DSGC ($\mathbf{G}\mathbf{X}\mathbf{F}$) due to the low-quality object link graph, as explained above.

Table 6.12: Clustering performance.

Dataset			20 NG		L-Cora		Wiki	
Method	G	F	Acc(%)	NMI(%)	Acc(%)	NMI(%)	Acc(%)	NMI(%)
Spectral	\times	\times	25.29 \pm 1.01	28.18 \pm 0.74	28.22 \pm 1.01	11.61 \pm 0.04	29.25 \pm 0.00	21.83 \pm 0.00
GAE [87]	\checkmark	\times	38.92 \pm 1.39	44.58 \pm 0.40	34.45 \pm 0.76	22.38 \pm 0.18	33.78 \pm 0.32	22.88 \pm 0.20
VGAE [87]	\checkmark	\times	25.04 \pm 0.81	25.72 \pm 0.77	29.45 \pm 1.25	17.53 \pm 0.15	33.83 \pm 0.45	21.46 \pm 0.19
MGAE [88]	\checkmark	\times	47.83 \pm 2.33	56.14 \pm 1.00	35.87 \pm 0.97	30.57 \pm 0.98	32.73 \pm 1.16	27.95 \pm 2.29
ARGE [89]	\checkmark	\times	42.04 \pm 0.50	44.13 \pm 0.91	36.07 \pm 0.05	27.74 \pm 0.01	26.49 \pm 0.10	17.17 \pm 0.05
ARVGE [89]	\checkmark	\times	21.10 \pm 0.61	21.79 \pm 0.49	26.45 \pm 0.03	12.94 \pm 0.01	33.82 \pm 0.13	21.42 \pm 0.11
AGC [2]	\checkmark	\times	38.83 \pm 0.84	47.08 \pm 1.57	41.76 \pm 0.01	33.65 \pm 0.01	32.74 \pm 0.01	24.90 \pm 0.01
DSGC (GX)	\checkmark	\times	38.42 \pm 0.66	46.28 \pm 0.93	38.26 \pm 0.02	30.66 \pm 0.02	31.43 \pm 0.09	24.16 \pm 0.18
DSGC (XF)	\times	Emb	28.99 \pm 0.06	33.22 \pm 0.10	30.80 \pm 0.56	17.46 \pm 0.21	35.45 \pm 0.91	33.44 \pm 0.66
	\times	PPMI	48.36 \pm 2.40	53.27 \pm 2.17	36.46 \pm 0.06	22.53 \pm 0.03	38.10 \pm 0.01	36.07 \pm 0.02
DSGC (GXF)	\checkmark	Emb	43.40 \pm 0.66	50.97 \pm 0.58	40.75 \pm 0.02	33.05 \pm 0.04	30.50 \pm 0.01	25.48 \pm 0.03
	\checkmark	PPMI	52.25 \pm 1.97	61.34 \pm 1.07	41.24 \pm 0.04	30.92 \pm 0.01	31.37 \pm 0.08	26.06 \pm 0.20

* \checkmark and \times indicate using/not using G or F .

Chapter Review

This chapter presents experimental results of our proposed methods compared with the state-of-the-art on various learning tasks. Sections 6.1 and 6.2 focus on 1-D graph convolution. Section 6.1 demonstrates the advantage of IGCN and GLP proposed in Chapter 4 over vanilla GCN and several other semi-supervised learning methods. Section 6.2 validates our proposed framework for unsupervised clustering and our proposed strategy AGC for adaptive filter strength selection. Section 6.3 focuses on 2-D graph convolution. The experiments in Section 6.3 verify the theorems in Chapter 5 and show the superiority of 2-D graph convolution, when properly used, over 1-D graph convolution.

Chapter 7

Applications

In this chapter, we discuss several real applications of graph convolution. We first show two applications of 1-D graph convolution, including zero-shot image classification and personalized product search. Then, we describe several potential applications of 2-D graph convolution, including malware detection, skeleton-based action recognition, and traffic flow forecasting.

7.1 Zero-Shot Image Classification

The proposed GLP and IGCN methods can also be used for semi-supervised regression. In Wang et al. [19], GCN was used for zero-shot image recognition with a regression loss. Here, we replace the GCN model used in Wang et al. [19] with GLP and IGCN to test their performance on the zero-shot image recognition task.

Zero-shot image recognition in Wang et al. [19] is to learn a visual classifier for the

categories with zero training examples, with only text descriptions of categories and relationships between categories. In particular, given a pre-trained CNN for known categories, Wang et al. [19] propose to use a GCN to learn the model/classifier weights of unseen categories in the last layer of the CNN. It first takes the word embedding of each category and the relations among all the categories (WordNet knowledge graph) as the inputs of GCN, then trains the GCN with the model weights of known categories in the last layer of the CNN and finally predicts the model weights of unseen categories.

Datasets. We evaluate our methods and baselines on the benchmark of ImageNet [124]. ImageNet is an image database organized according to the WordNet hierarchy. All categories of ImageNet form a graph through “is a kind of” relation. For example, drawbridges are a kind of bridge, bridges are a kind of construction, and construction is a kind of artifact. According to Wang et al. [19], the word embedding of each category is learned from Wikipedia by the GloVe text model [111].

Baselines. We compare our methods GLP and IGCN with six state-of-the-art zero-shot image recognition methods, namely Devise [125], SYNC [126], GCNZ [19], GPM [127], DGPM [127] and ADGPM [127]. The prediction accuracies of these baselines are taken from their papers. Notably, the GPM model is exactly our IGCN with $k = 1$.

Settings. There are 21K different classes in ImageNet. We split them into a training set, and a test set similarly as in Kampffmeyer et al. [127]. A ResNet-50 model was pre-trained on the ImageNet 2012 with 1k classes. The weights of these 1000 classes in the last layer of CNN are used to train GLP and IGCN for predicting the weights of the remaining classes. The evaluation of zero-shot image recognition is conducted on the AWA2 dataset [128], which is a subset of ImageNet. For IGCN and the classifier (MLP)

of GLP, we both use a two-layer structure with 2048 hidden units. We test $k = 1, 2, 3$ for IGCN and $k = 2, 4, 6$ for GLP. Results are averaged over 20 runs.

Table 7.1: Results for unseen classes in AWA2.

Method	Devise	SYNC	GCNZ	GPM	DGPM	ADGPM
Accuracy	59.7	46.6	68.0 (1840s)	77.3 (864s)	67.2 (932s)	76.0 (3527s)
Our Method	IGCN(RNM)			GLP(RNM)		
	k=1	k=2	k=3	k=2	k=4	k=6
Accuracy	77.9 (864s)	77.7 (1583s)	73.1 (2122s)	76.0 (12s)	75.0 (13s)	73.0 (11s)

Performance and Results Analysis. The results are summarized in Table 7.1, where the top 3 classification accuracies are highlighted in bold. We can see that IGCN with $k = 1, 2$ and GPM [127] perform the best, and outperform other baselines including Devise [125], SYNC [126], GCNZ [19], and DGPM [127] by a significant margin. GLP with $k = 2$ is the second best compared with the baselines, only slightly lower than GPM. We observe that smaller k achieves better performance on this task, probably because the diversity of features (classifier weights) should be preserved for the regression task [127]. This also explains why DGPM [127] (that expands the node neighborhood by adding distant nodes) does not perform very well. It is also worth noting that by replacing the 6-layer GCN in GCNZ with a 2-layer IGCN with $k = 3$ and a GLP with $k = 6$, the performance boosts from 68% to around 73%, demonstrating the low complexity and training efficiency of our methods. Another thing to notice is that GLP runs hundreds of times faster than GCNZ and tens of times faster than others.

7.2 Personalized Product Search

Product search is an essential module in online shopping platforms, which guides users to browse and purchase products from a massive collection of commodities. Products are mainly represented by short texts such as titles and descriptions, which may not always be informative. Other than texts, products are also associated with diverse relational data, including ontology, spec sheets, figures, etc. There are also various types of user-item interactions in e-commerce platforms. Users can browse, click, review, purchase a product or put it in their cart.

Utilizing such rich information for personalized product search would be highly desirable yet challenging. Existing methods mainly exploit text data and model product search as an information retrieval task [129–132]. However, they are limited in their ability to model user preferences, which is the core problem in product search. A common way to represent users is by the products they've visited during a period, but long-term historical user behavior usually contains noisy preference signals [129–131]. For computational efficiency, user behavior sequences are usually truncated, and only recent behaviors are considered. While this helps to eliminate noisy preference signals, short-term user behavior may not contain sufficient preference signals.

To improve user preference modeling and learn better product representations, we propose to utilize a global successive behavior graph and perform graph convolution over the graph to capture implicit and complex collaborative signals. Here, we adopt an efficient graph convolution layer with jumping connections and provide theoretical analysis to show its advantage.

Efficient Graph Convolution. In the past few years, graph convolutional networks (GCN) and variants have been successfully applied to learn useful graph node representations for various graph learning and mining tasks. In each layer of GCN, it performs feature propagation and transformation with connected nodes in the graph

$$\mathbf{H}^{(l)} = \sigma(\mathbf{A}_s \mathbf{H}^{(l-1)} \mathbf{W}^{(l)}), \quad (7.1)$$

where $\mathbf{H}^{(l)}$ is the node embeddings produced by layer l , $\mathbf{W}^{(l)}$ denotes trainable parameters, and σ is a non-linear function such as $\text{ReLU}(\cdot)$. However, as observed from our empirical study, the projection layers may distort the semantic product representations learned by the language model in methods such as ZAM or HEM. Hence, we propose to use graph convolution without the projection layers to enrich product representations. Following the efficient design in Li et al. [3], we remove the projection layers and activation layers. Further, we add a balancing parameter ω to control the strength of self-information:

$$\mathbf{H}^{(l)} = (\omega \mathbf{I} + (1 - \omega) \mathbf{A}_s) \mathbf{H}^{(l-1)}, \quad (7.2)$$

Jumping Graph Convolution Layer. Since user purchase behavior is often sparse, it is helpful to aggregate high-order information on the successive behavior graph to model potential user interest. However, the ordinary graph convolution suffers from the well-known over-smoothing problem, i.e., stacking too many convolution layers may make the node features (product representations) indistinguishable. To address this issue, Chen et al. [133] proposed GCNII that adds initial residual connections to each GCN layer. We follow the same design to add jumping connections, i.e., feeding each convolution layer an additional input of the initial product representations $\mathbf{H}^{(0)}$, i.e.,

$$\tilde{\mathbf{H}}^{(l)} = (\omega \mathbf{I} + (1 - \omega) \mathbf{A}_s) \left(\beta \mathbf{H}^{(0)} + (1 - \beta) \tilde{\mathbf{H}}^{(l-1)} \right), \quad (7.3)$$

where β is a weight parameter determining the portion of initial features. We call this jumping graph convolution layer.

We provide a theoretical analysis of the jumping group convolution by investigating the diversity of the convoluted product representations using the Laplacian-Beltrami operator $\Omega(\cdot)$ [23]. We compare the diversity of the convoluted product representations with jumping connections ($\tilde{\mathbf{H}}^{(l)}$ in Eq. (7.3)) and those without jumping connections ($\mathbf{H}^{(l)}$ in Eq. (7.2)). Laplacian-Beltrami operator measures the total variance of connected nodes:

$$\Omega(\mathbf{H}) = \sum_k \sum_{i,j} a_{ij} (\mathbf{H}_{i,k} - \mathbf{H}_{j,k})^2. \quad (7.4)$$

High $\Omega(\mathbf{H})$ indicates high diversity, and low $\Omega(\mathbf{H})$ indicates severe over-smoothing. The following theorem shows that the jumping connection can substantially alleviate the over-smoothing effect of graph convolution.

Theorem 26. *If the initial diversity $\Omega(\mathbf{H}^{(0)}) > 0$, then for any $l > 0$, $\beta \in (0, 1)$, and $\omega \in (0.5, 1)$, $\tilde{\mathbf{H}}^{(l)}$ is strictly more diverse than $\mathbf{H}^{(l)}$, i.e.,*

$$\Omega(\tilde{\mathbf{H}}^{(l)}) > \Omega(\mathbf{H}^{(l)}). \quad (7.5)$$

When l approaches infinity, jumping connection can prevent the diversity of product representations from collapsing to 0 (over-smoothing):

$$\lim_{l \rightarrow \infty} \Omega(\tilde{\mathbf{H}}^{(l)}) > \lim_{l \rightarrow \infty} \Omega(\mathbf{H}^{(l)}) = 0. \quad (7.6)$$

Proof. Let $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, then $\Omega(\cdot)$ becomes

$$\Omega(\mathbf{H}) = \sum_k \mathbf{H}_{:,k}^\top \mathbf{L} \mathbf{H}_{:,k}, \quad (7.7)$$

where $\mathbf{H}_{:,k}$ is the k -th column of \mathbf{H} . Denote arbitrary column of \mathbf{H} by h , then we only need to prove

$$(\tilde{h}^{(l)})^\top \mathbf{L} \tilde{h}^{(l)} = \Omega(\tilde{h}^{(l)}) > \Omega(h^{(l)}) = (h^{(l)})^\top \mathbf{L} h^{(l)}$$

and

$$\lim_{l \rightarrow 0} \Omega(\tilde{h}^{(l)}) > \lim_{l \rightarrow 0} \Omega(h^{(l)}) = 0.$$

Let $\mathbf{F} = (\omega \mathbf{I} + (1 - \omega) \mathbf{D}^{-1} \mathbf{A})$. From Eq. (7.2) and Eq. (7.3), we could obtain general formula for $h^{(l)}$ and $\tilde{h}^{(l)}$:

$$h^{(l)} = \mathbf{F}^l h^{(0)} \quad (7.8)$$

$$\tilde{h}^{(l)} = \left((1 - \beta)^l \mathbf{F}^l + \beta \sum_{k=1}^l (1 - \beta)^{k-1} \mathbf{F}^k \right) h^{(0)} \quad (7.9)$$

Denote the eigen-decomposition of \mathbf{L} by $\mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top$, where \mathbf{U} is the eigenbasis and $\mathbf{\Lambda}$ is a diagonal matrix storing corresponding eigenvalues, then

$$\mathbf{F} = (\mathbf{I} - (1 - \omega) \mathbf{L}) = \mathbf{U} (\mathbf{I} - (1 - \omega) \mathbf{\Lambda}) \mathbf{U}^\top = \mathbf{U} \mathbf{M} \mathbf{U}^\top \quad (7.10)$$

where $\mathbf{M} = \mathbf{I} - (1 - \omega) \mathbf{\Lambda}$. Denote $c = \mathbf{U}^\top h^{(0)}$ and substitute \mathbf{F} in Eq. (7.8) and (7.9) by Eq. (7.10):

$$\Omega(h^{(0)}) = c^\top \mathbf{\Lambda} c = \sum_i \lambda_i c_i^2, \quad (7.11)$$

$$\Omega(h^{(l)}) = c^\top \mathbf{M}^{2l} \mathbf{\Lambda} c = \sum_i (1 - (1 - \omega) \lambda_i)^{2l} \lambda_i c_i^2 \quad (7.12)$$

$$= \sum_i g_l^2(\lambda_i) \lambda_i c_i^2, \quad (7.13)$$

$$\Omega(\tilde{h}^{(l)}) = c^\top \left((1 - \beta)^l \mathbf{M}^l + \beta \sum_{k=1}^l (1 - \beta)^{k-1} \mathbf{M}^k \right)^2 \mathbf{\Lambda} c \quad (7.14)$$

$$= \sum_i f_l^2(\lambda_i) \lambda_i c_i^2, \quad (7.15)$$

where

$$g_l(\lambda) = (1 - (1 - \omega) \lambda)^l, \quad (7.16)$$

$$f_l(\lambda) = (1 - \beta)^l g_l(\lambda) + \beta \sum_{k=1}^l (1 - \beta)^{k-1} g_k(\lambda). \quad (7.17)$$

Now, we only need to compare $g_l(\lambda)$ and $f_l(\lambda)$. Notice that $\omega \in (0.5, 1)$ and λ is the eigenvalue of normalized graph laplacian \mathbf{L} , so $\lambda \in [0, 2]$ and $1 - (1 - \omega)\lambda \in (0, 1]$.

Then $g_l(\lambda)$ is positive and decreases as l increases:

$$g_{l_1}(\lambda) \geq g_{l_2}(\lambda) > 0, \text{ for any } l_2 > l_1. \quad (7.18)$$

The equality holds only if $\lambda = 0$. When it comes to $f_l(\lambda)$, we have

$$f_l(\lambda) \geq \left((1 - \beta)^l + \beta \sum_{k=1}^l (1 - \beta)^{k-1} \right) g_l(\lambda) = g_l(\lambda) > 0. \quad (7.19)$$

Given Eq. (7.13), (7.15) and (7.19), we can conclude

$$\Omega(\tilde{h}^{(l)}) \geq \Omega(h^{(l)}) \quad (7.20)$$

Notice that initial diversity $\Omega(h^{(0)}) > 0$, so there exists such λ_i that $\lambda_i c_i^2 > 0$ and $\lambda_i \neq 0$, then the equality does not hold and inequality Eq. (7.5) is proved.

Now we consider the limits of $f_l(\lambda)$ and $g_l(\lambda)$:

$$\lim_{l \rightarrow \infty} f_l(\lambda) = \frac{\beta(1 - (1 - \omega)\lambda)}{1 - (1 - \beta)(1 - (1 - \omega)\lambda)} > 0, \quad (7.21)$$

$$\lim_{l \rightarrow \infty} g_l(\lambda) = 0, \quad \text{for all } \lambda > 0. \quad (7.22)$$

As a consequence,

$$\lim_{l \rightarrow 0} \Omega(\tilde{h}^{(l)}) > \lim_{l \rightarrow 0} \Omega(h^{(l)}) = 0. \quad (7.23)$$

Eq. (7.6) is also proved. □

Let the rows of $\mathbf{H}^{(0)}$ represent the learned entity embeddings. We perform L iterations of graph convolution as in Eq. (7.3) and obtain the entity representations $\tilde{\mathbf{H}}^{(L)}$.

Experiments on Amazon review dataset [134, 135] show that our methods significantly improves over ZAM [129] by 7.76% to 41.39% in different domains, and also outperforms other state-of-art baselines, including HEM [130], DREM [136] and GraphSRRL [137].

Notation	Element	Description
X	x_{ij}	If APP _{<i>i</i>} invokes API _{<i>j</i>} , then $x_{ij} = 1$; otherwise $x_{ij} = 0$.
A	a_{ij}	Indicate whether two malware are similar to each other, constructed as Fan et al. [138].
B	b_{ij}	Co-block [139]. If API _{<i>i</i>} and API _{<i>j</i>} are invoked in the same code block, then $b_{ij} = 1$; otherwise, $b_{ij} = 0$.
P	p_{ij}	Co-package [139]. If API _{<i>i</i>} and API _{<i>j</i>} are provided by same package, then $p_{ij} = 1$; otherwise, $p_{ij} = 0$.

Table 7.2: Features and relations.

7.3 Malware Detection and Analysis

Malware are software that contains malicious behaviors. They have become a severe security issue on various platforms that cannot be ignored. It brings huge harm to users, including stealthy tariff consumption, privacy disclosure, and remote control. An in-depth study of malware is of great significance to the sound development of the application ecosystem. However, malware detection and analysis is challenging in that 1) malware are usually camouflaged as benign applications, so they are actually dominated by benign codes, and malicious codes only take a small portion even in malware; 2) obfuscation techniques enable malware to appear very differently while reserving the same malicious behaviors; 3) Analysis and labeling malware require considerable expert knowledge, and thus labeled data are scarce [36].

Feature Extraction We propose to detect and classify malware by our dimension-wise separable graph convolution (DSGC) network. The features we consider are sensitive APIs invoked by each application. Malware are software that contains malicious behav-

iors. To conduct malicious behaviors, malware must invoke some APIs to finish various tasks, such as sending texts, recording videos and audio, taking photos, or sending data to remote servers. All sensitive APIs invoked by a malware are obfuscation-invariant and well define apps' behavior. Assuming we have n APPs and m sensitive APIs, binary matrix $\mathbf{X} \in \{0, 1\}^{n \times m}$ indicates whether an APP invokes an API. An independent similarity graph between APPs is constructed as described in Fan et al. [138]. In short, APPs are transformed into function call graph (FCG), then embeddings are learned from the graph by *struc2vec* [140], and finally similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ are calculated from the distance between embeddings. Inspired by Hou et al. [139], we consider following two kinds of relations between APIs: 1) co-block, two APIs being invoked in the same code block. 2) co-package, two APIs being provided by the same package.

Classifier Denote the Laplacian of APP-APP graph by \mathbf{L}_1 and Laplacian of APP-API graph by \mathbf{L}_2 . We first perform a dimension-wise separable graph convolution to fuse the information encoded in feature \mathbf{X} , APP similarity graph \mathbf{A} and API similarity graph \mathbf{B} or \mathbf{P} :

$$\mathbf{Z} = \left(\sum_{k_1=0}^{K_1} \theta_{k_1}^{(1)} \mathbf{L}_1^{k_1} \right) \mathbf{X} \left(\sum_{k_2=0}^{K_2} \theta_{k_2}^{(2)} \mathbf{L}_2^{k_2} \right)^\top = \mathbf{G} \mathbf{X} \mathbf{F},$$

$$\text{where } \mathbf{G} = \sum_{k_1=0}^{K_1} \theta_{k_1}^{(1)} \mathbf{L}_1^{k_1} \text{ and } \mathbf{F} = \sum_{k_2=0}^{K_2} \theta_{k_2}^{(2)} (\mathbf{L}_2^{k_2})^\top.$$

Then, we apply a standard MLP to get final prediction.

$$\hat{\mathbf{y}}_i = \text{MLP}(\mathbf{z}_i) \quad (7.24)$$

The MLP is trained by minimizing cross-entropy loss.

$$\min \sum_i \text{CrossEntropy}(\mathbf{y}_i, \hat{\mathbf{y}}_i). \quad (7.25)$$

7.4 Skeleton-based Action Recognition

Human action recognition is to identify what action a person is performing in an image or video, which plays an important role in automated surveillance, human intention prediction, elderly behavior monitoring, human-computer interaction, content-based video retrieval, and video understanding [17, 141, 142]. There are multiple modalities from which action can be recognized, including RGB videos, depth videos, optical flow, and human skeletons. Among them, RGB videos are the most studied modality, upon which many vision-based methods are developed, such as CNNs.

Another modality of particular interest to us is dynamic human skeletons. Human skeletons model the human body as a set of 3D coordinates of joints, such as wrists, elbows, knees, hips, and shoulders, and these joints are connected by bones. A dynamic human skeleton is a series of consecutive frames of human skeletons, which records a person's actions in a period. The main advantages of skeleton-based methods are that it takes less storage space, less memory space, and less computation than vision-based methods. A typical 1080p colorful image contains $1920 \times 1080 \times 3 \approx 6.22\text{m}$ 8-bit integers. A 10-second 1080p 30fps video takes about 1.87 GB of memory space. Processing such video, and recognizing human action from it also requires tons of float operations. In contrast, a frame of a human skeleton only contains dozens of 3D coordinates of joints. Take NTU-RGB+D dataset as an example [142], each frame consists of 25 body joints, whose coordinates are stored as $25 \times 3 = 75$ float32 numbers. A 10-second 30fps dynamic human skeleton takes only 9KB memory space, less than RGB videos by about 200,000 times. As a consequence of the extremely small data scales, skeleton-based methods are also faster than vision-based ones by several orders of

magnitude. However, skeleton-based methods also have limitations. First, it is not good at recognizing actions that involve interactions with the environment. For actions such as playing basketball or sitting in a chair, vision-based methods could directly perceive the existence of basketballs or chairs and thus make correct recognition, while skeleton-based methods must guess what objects are being interacted with. Second, most of the current skeleton data are estimated from RGB videos, although it is not necessarily obtained from RGB videos. Skeleton estimation plus skeleton-based recognition is actually a two-stage vision-based method, which degraded the advantages brought by the data efficiency of human skeletons. However, skeleton data is not necessarily estimated from RGB videos. We can obtain skeletons from depth images, radar, LiDAR, and even directly from wearable devices.

7.4.1 Dynamic Human Skeletons

Formally, a human skeleton is a graph, where vertices \mathcal{V} are human joints and edges are bones. The 3D coordinates of joints are stored in a matrix $\mathbf{X} \in \mathbb{R}^{N \times 3}$. A finite time period is a chain graph with vertices $\{0, 1, 2 \dots\}$ and vertex t represents t -th timestamp. Consecutive timestamps are connected by edges. A dynamic human skeleton is a Cartesian product graph of time and skeleton. Each vertex in a dynamic human skeleton is a tuple (t, ν) represents a joint at t -th frame. There are two types of edges, temporal edges and spatial edges. Temporal edges connect the same joints at two consecutive frames.

$$(t, \nu) \sim (t + 1, \nu) \quad (7.26)$$

$$(t, \nu) \sim (t - 1, \nu) \quad (7.27)$$

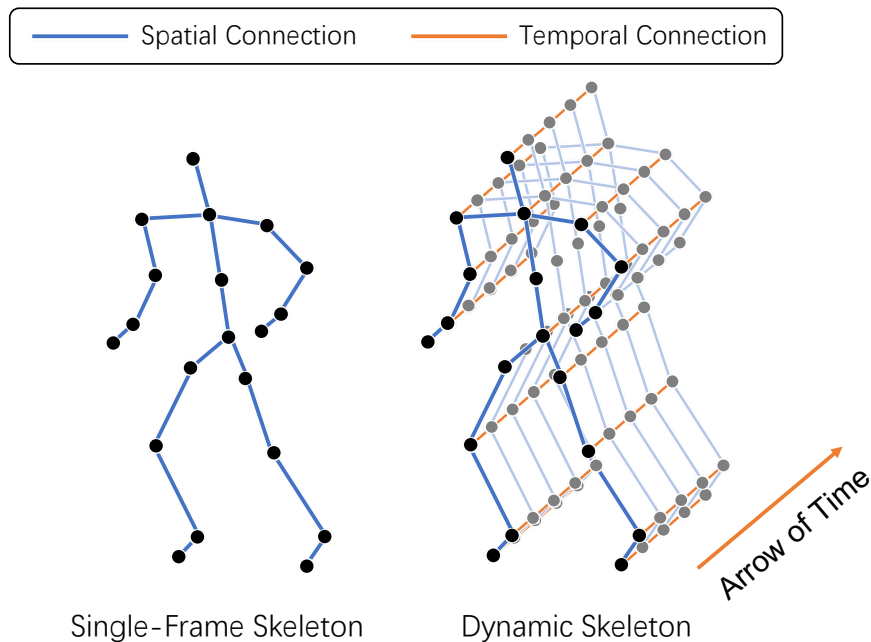


Figure 7.1: Spatial connections and temporal connections in a dynamic human skeleton.

Spatial edges connect joints within a frame that are spatially connected by bones.

$$(t, \nu_i) \sim (t, \nu_j) \iff \nu_i \sim \nu_j \quad (7.28)$$

Assume there are N joints and T frames, the coordinates of joints at different frames are stored in a tensor $\mathcal{X} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T] \in \mathbb{R}^{T \times N \times 3}$.

7.4.2 Spatial Temporal Graph Convolutional Networks

We design a general 2-D spatial-temporal graph convolution network to recognize human actions from dynamic human skeletons, which demonstrates the mixed usage of graph convolution and Euclidean convolution. The existing spatial-temporal graph convolution designed for action recognition [17, 143] are special variants of DSGC, which convolves in different dimensions separately. Different from existing ones, we exploit general 2-D convolution here for a higher model capacity.

Dimension-wise Separable Convolutional Layer The first several layers, as the main body of the network, are spatial-temporal graph convolutional layer. Denote the input of l -th layer by $\mathcal{X}^{(l-1)} \in \mathbb{R}^{T \times N \times C_{l-1}}$, and output by $\mathcal{X}^{(l)} \in \mathbb{R}^{T \times N \times C_l}$, then forward propagation formula for convolutional layer is

$$p_l(\mathbf{A}) = \sum_{k=0}^K \alpha_k \mathbf{A}^k, \quad (7.29)$$

$$\mathcal{X}^{(l)} = \mathcal{X}^{(l-1)} *_1 \boldsymbol{\theta}^{(l)} \times_2 p_l(\mathbf{A}) \times_3 \mathbf{W}^{(l)}. \quad (7.30)$$

“ $*_1$ ” is normal 1-D convolution at the temporal (first) dimension, and $\boldsymbol{\theta}^{(l)}$ is the kernel. “ \times_2 ” is n-mode product at the spatial (second) dimension. $p_l(\mathbf{A})$ is a graph convolutional filter, so “ $\times_2 p_l(\mathbf{A})$ ” is the spatial convolution. Finally, $\mathbf{W}^{(l)} \in \mathbb{R}^{C_{l-1} \times C_l}$ is a forward transform matrix, performed at the channel (third) dimension. In each layer, $\boldsymbol{\theta}^{(l)}$, α_k , and $\mathbf{W}^{(l)}$ are trainable parameters. The input to first layer is \mathcal{X} , i.e., $\mathcal{X}^{(0)} = \mathcal{X}$.

General 2-D Spatial-Temporal Convolution Besides above dimension-wise separable style convolution, we can also exploit general 2-D convolution to substitute Eq. (7.30) for a higher model capacity:

$$\mathcal{X}^{(l)} = \sum_i^{K_1} \sum_j^{K_2} \mathcal{X}^{(l-1)} \times_1 \mathbf{S}^i \times_2 \mathbf{A}^j \times_3 \mathbf{W}_{ij}^{(l)}, \quad (7.31)$$

where \mathbf{S} is the shift matrix, and $s_{ij} = 1$ if $i = j + 1$, $s_{ij} = 0$ otherwise.

Pooling Layer Different samples may have different lengths of time T , so a pooling at the temporal dimension is needed.

$$\mathbf{Z} = \text{Pool}_1(\mathcal{X}^{(l)}) \quad (7.32)$$

Pool_1 is a reduce operator performed at the temporal dimension, such as reduce sum, reduce mean or max pooling. After pooling, we get $\mathbf{Z} \in \mathbb{R}^{T \times C_l}$, whose shape is irrelevant to T , the length of time .

Softmax Layer Finally, we vectorize \mathbf{Z} and get the final classification result by a softmax layer.

$$\mathbf{z} = \text{vec}(\mathbf{Z}) \quad (7.33)$$

$$\mathbf{y} = \text{softmax}(\mathbf{W}\mathbf{z}) \quad (7.34)$$

\mathbf{W} is the trainable parameter.

The core of the network is spatial-temporal convolutional layers, which serve as a powerful feature extractor. It exploits two types of convolution to extract features from joint coordinates. Temporal convolution could extract time-related features, such as joint velocity and acceleration. Spatial convolution could extract space-related features, such as distance between joints, direction of bone, bone length, the angle between two bones, and the center of mass. Their combination could extract features that rely on both space and time, such as the rotational speed of a bone and the velocity of the mass center. Such spatial-temporal convolution provided plenty of useful features for downstream classification tasks.

7.5 Traffic Flow Forecasting

Another application of spatial-temporal convolution is traffic flow forecasting. Traffic Flow Forecasting is to forecast the traffic flow passing by each node in a traffic network,

given the historical flow of all nodes and some other helpful information about the network. There are mainly two types of methods to forecast traffic flow, CNN and GNN. CNN-based methods take traffic flow at a time point as a heat map image and treat it as a video prediction task [6, 144]. GNN-based methods treat traffic flow as a sequence of graphs and predict the flow by graph neural networks [7, 18, 145]. In general, modeling traffic networks as graphs is more natural, saving both memory space and computational resources.

Most GNN-based methods utilize Spatial-Temporal Graph Convolutional Networks (ST-GCN). Spatial-temporal convolutional (ST-Conv) layers, as the network's main body, rely on both spatial convolution and temporal convolution to extract features useful for downstream prediction. The existing ST-Conv filters designed for traffic flow forecasting [7, 18, 145] are also special variants of DSGC (Eq. (7.30)), which convolves in different dimensions separately. To improve model capacity, we propose to exploit general 2-D graph convolution.

The whole network architecture is similar to the one for skeleton-based action recognition, because traffic flow forecasting and skeleton-based action recognition are both dealing with spatial-temporal graphs. However, they differ in two ways: 1) Traffic flow forecasting is a node-level task, which require to predict for each spatial node, while action recognition is a graph-level task; 2) Traffic flow forecasting is a regression task, and action recognition is a classification task. Therefore, the ST-GCN designed for traffic flow forecasting shares nearly the same architecture as the one for action recognition, except that it uses a node-level prediction layer instead of a softmax layer, and a mean square loss instead of a cross-entropy loss.

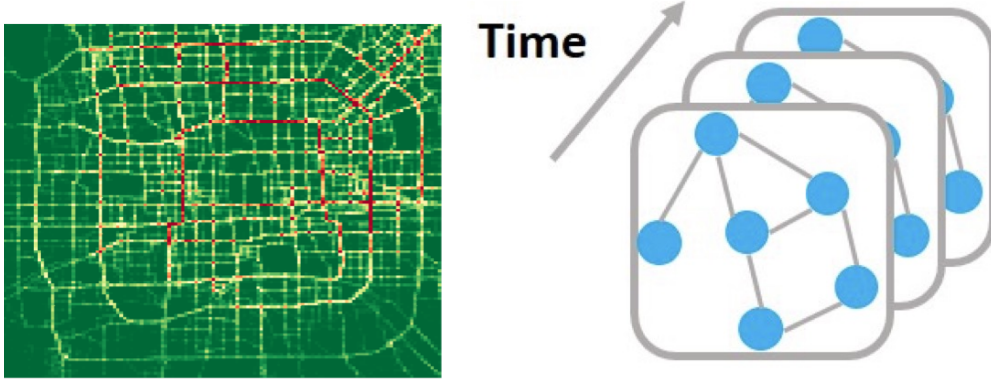


Figure 7.2: Left: traffic flow as a heat map image [6]. Right: traffic flow as a sequence of graphs [7].

Suppose there is a graph with N nodes and an adjacency matrix \mathbf{A} . Each node is described by a C -channel feature vector, containing its flow and other information at t -th time slice, like vehicle speed, wind speed, precipitation, etc. All features for t -th time slice are stored in a matrix $\mathbf{X}_t \in \mathbb{R}^{N \times C}$. We know all features before time T , stored in a tensor $\mathcal{X} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T] \in \mathbb{R}^{T \times N \times C}$, and we want to predict the traffic flow of all nodes $\mathbf{Y} = [\mathbf{y}_{T+1}, \mathbf{y}_{T+2}, \dots, \mathbf{y}_{T+T_p}] \in \mathbb{R}^{T_p \times N}$ after time T .

Spatial-Temporal Convolutional Layer. We exploit the general 2-D graph convolution as in Eq. (7.31).

Prediction Layer. After several convolutional layers, it follows the final prediction layer:

$$\hat{\mathbf{Y}} = \mathbf{W}_T(\mathcal{X}^{(l)} \times_c \mathbf{w}_c), \quad (7.35)$$

where $\mathbf{w}_c \in \mathbb{R}^{C_l}$ is a vector, and the product “ \times_c ” eliminates the channel dimension, i.e., $(\mathcal{X}^{(l)} \times_c \mathbf{w}_c) \in \mathbb{R}^{T \times N}$. $\mathbf{W}_T \in \mathbb{R}^{T_p \times T}$ is a transform matrix in the temporal dimension. Finally, we get the prediction of traffic flow, $\hat{\mathbf{Y}} \in \mathbb{R}^{T_p \times N}$, of all N nodes in the next T_p

time slices. The network is trained by minimizing the MSE loss between the prediction \hat{Y} and the ground truth Y :

$$\min \|Y - \hat{Y}\|_F^2. \quad (7.36)$$

Chapter Review

In this chapter, we introduce several real applications of graph convolution, including zero-shot image classification, personalized product search, malware detection, skeleton-based action recognition, and traffic flow forecasting. We discuss possible solutions for each application and present some preliminary results for some applications.

Chapter 8

Conclusion and Future Works

In this thesis, we have introduced a set of techniques for analyzing graph convolutional filters in both spatial and spectral domains. With these techniques, we reveal fundamental mechanisms of graph convolutional networks for unsupervised and semi-supervised learning. Our spatial analysis [1] shows that the graph convolution of GCN is actually a special form of Laplacian smoothing, which is the key reason why GCN works. It also brings up the over-smoothing problem, which is a fundamental limitation of deep GCN models. Our spectral analysis [3] revisits GCN and classical label propagation methods under a unified graph filtering framework, revealing critical insight that they both essentially learn data representations using a low-pass graph filter. The analysis provides useful guidelines for designing new graph convolutional filters and facilitates filter comparison.

With the theoretical analysis, we have developed new efficient models with graph convolution for semi-supervised and unsupervised learning, including Improved Graph Convolutional Networks (IGCN) [3], Generalized Label Propagation (GLP) [3], Adaptive

Graph Convolution (AGC) [2] and Dimensionwise Separable 2-D Graph Convolution (DSGC) [4]. IGCN extends vanilla GCN and is more flexible and efficient. GLP extends classic Label Propagation (LP) methods to deal with attributed graphs and is a simplified version of GCN. AGC is a simple node clustering method for attributed graphs, which efficiently uses an adaptive graph convolutional filter and outperforms state-of-the-art methods. DSGC generalizes beyond 1-D graph convolution and designs an efficient dimensionwise separable 2-D graph convolutional filter for unsupervised and semi-supervised learning on attributed graphs. The effectiveness of our proposed models and algorithms have been verified by extensive experiments carried out on real-world benchmark datasets.

Further, we have applied our proposed models and algorithms to various real-world applications across multiple areas. Particularly, we have demonstrated the effectiveness of our IGCN and GLP models in zero-shot image classification [3], and successfully applied IGCN to model user behaviour for personalized product search [5]. We have also discussed the potential application of our proposed 2-D graph convolutional filters to malware detection, skeleton-based action recognition, and traffic flow forecasting.

The significance of this thesis is three-fold:

- The thesis pioneers in analyzing GCNs from both spatial and spectral perspectives, revealing key insights and fundamental limitations of GCNs, and laying the foundation for many follow-up studies.
- The thesis proposes a set of efficient models and algorithms for unsupervised and semi-supervised learning on graphs and provides comprehensive experimental results, which can serve as solid baselines for future research.

- This thesis presents successful and potential applications of the proposed models and algorithms to a variety of real-world applications in computer vision, e-commerce, cyber security, and smart transportation.

Future work The following issues are left for future exploration.

1) Data-driven filter design. Most of existing GCN models use manually designed graph convolutional filters. The filter parameters are manually tuned and selected through extensive experiments. In Section 4.3, we draw a qualitative conclusion about how to decide filter strength according to label rate for semi-supervised node classification. When only a few labels are given, one should increase filter strength; and when label rate is high, reducing filter strength would be desirable. While this rule is helpful, it still requires manual tuning to decide filter parameters. It would be much more desirable if we can find a quantitative relation between filter parameters and data statistics (e.g., label rate, graph size, and node degree) to directly determine filter parameters. Further, it would be interesting to investigate if the filter parameters can be fully automatically learned like CNN parameters, i.e., setting the filter parameters as trainable and learning them from data during training.

2) Applications of 2-D and multi-dimension graph convolution. In Chapter 7, we discuss several potential applications of 2-D graph convolution. It would be interesting to implement the proposed 2-D graph convolutional filters and explore their potential for these applications. In Section 3.6, we introduce multi-dimension graph convolution. The application of multi-dimension graph convolution is also worth exploring. Potential applications include meteorological sensor network data analysis, recommender systems, and point cloud compression.

3) Scalability of graph neural networks. One of the most important issues for graph neural networks is scalability, which hinders their application in industry. Normal neural networks scale well to extremely large datasets of gigabyte or terabyte size, thanks to the batch training technique. However, applying batch training to graph-structured data is non-trivial because each batch can only cover the edges whose two end nodes both appear in the batch, and the cross-batch edges are discarded. As a result, nearly all graph neural networks resort to a full-batch training strategy, which is slow and requires large GPU memory. It is vital to design a proper batch splitting strategy that can efficiently cover as many edges as possible to improve the scalability of graph neural networks.

Bibliography

- [1] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2018*, pages 3538–3545. AAAI, 2018.
- [2] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4327–4333, 2019.
- [3] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9582–9591, 2019.
- [4] Qimai Li, Xiaotong Zhang, Han Liu, Quanyu Dai, and Xiao-Ming Wu. Dimensionwise separable 2-d graph convolution for unsupervised and semi-supervised learning on graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21)*, New York, NY, USA, 2021. ACM.
- [5] Lu Fan, Qimai Li, Bo Liu, Xiao-Ming Wu, Xiaotong Zhang, Fuyu Lv, Guli Lin, Sen

- Li, Taiwei Jin, and Keping Yang. Modeling user behavior with graph convolution for personalized product search. In *Proceedings of the Web Conference*, 2022.
- [6] Kun Ouyang, Yuxuan Liang, Ye Liu, Zekun Tong, Sijie Ruan, David Rosenblum, and Yu Zheng. Fine-grained urban flow inference. *IEEE transactions on knowledge and data engineering*, 2020.
- [7] Ken Chen, Fei Chen, Baisheng Lai, Zhongming Jin, Yong Liu, Kai Li, Long Wei, Pengfei Wang, Yandong Tang, Jianqiang Huang, et al. Dynamic spatio-temporal graph-based cnns for traffic flow prediction. *IEEE Access*, 8:185136–185145, 2020.
- [8] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48, 2016.
- [9] Fanghua Ye, Chuan Chen, and Zibin Zheng. Deep autoencoder-like nonnegative matrix factorization for community detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1393–1402. ACM, 2018.
- [10] Ning Wu, Wayne Xin Zhao, Jingyuan Wang, and Dayan Pan. Learning effective road network representation with hierarchical graph neural networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, pages 6–14, 2020.
- [11] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

- [12] Pengyang Wang, Kunpeng Liu, Lu Jiang, Xiaolin Li, and Yanjie Fu. Incremental mobile user profiling: Reinforcement learning with spatial knowledge graph for modeling event streams. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, pages 853–861, 2020.
- [13] Shichao Pei, Lu Yu, Guoxian Yu, and Xiangliang Zhang. REA: robust cross-lingual entity alignment between knowledge graphs. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, pages 2175–2184, 2020.
- [14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.
- [16] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.
- [17] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.
- [18] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting.

- In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.
- [19] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6857–6866, 2018.
- [20] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- [21] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 2013.
- [22] 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.
- [23] Fan RK Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.
- [24] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.
- [25] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3): 93–106, 2008.
- [26] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

- [27] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.
- [28] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.
- [29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [31] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.
- [32] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5): 75–174, 2010.
- [33] Xing Xie. Potential friend recommendation in online social network. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 831–835. IEEE, 2010.

- [34] Xiaojun Chen, Shengbin Jia, and Yang Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948, 2020.
- [35] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [36] FAN Ming, LIU Ting, LIU Jun, LUO Xiapu, YU Le, and GUAN Xiaohong. Android malware detection: A survey. *Scientia Sinica Informationis*, 50(8):1148–1177, 2020.
- [37] Yin-min Li, Zan Gao, Ya-bin Tao, Li-li Wang, and Yan-bing Xue. 3d object retrieval based on non-local graph neural networks. *Multimedia Tools and Applications*, 79(45):34011–34027, 2020.
- [38] X. Zhu and A.B Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.
- [39] Antti Rasmus, Mathias Berglund, Mikko Honkela, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
- [40] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *International conference on machine learning*, pages 1168–1175. PMLR, 2008.
- [41] M. Szummer and T Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing Systems*, pages 945–952, 2002.

- [42] M. Belkin and P Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine Learning*, 56(1):209–239, 2004.
- [43] O. Chapelle, J. Weston, and B Scholkopf. Cluster kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 601–608, 2003.
- [44] T. Zhang and R.K Ando. Analysis of spectral kernel design based semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2006.
- [45] A. Blum and S Chawla. Learning from labeled and unlabeled data using graph mincuts. In *International conference on machine learning*, pages 19–26, 2001.
- [46] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International conference on machine learning*, pages 912–919, 2003.
- [47] T Joachims. Transductive learning via spectral graph partitioning. In *International conference on machine learning*, pages 290–297, 2003.
- [48] A. Blum, J. Lafferty, M.R. Rwebangira, and R Reddy. Semi-supervised learning using randomized mincuts. In *International conference on machine learning*, page 13, 2004.
- [49] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, pages 321–328, 2004.

- [50] O. Chapelle and A Zien. Semi-supervised classification by low density separation. In *International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2005.
- [51] Y. Bengio, O. Delalleau, and N Le Roux. Label propagation and quadratic criterion. *Semi-supervised Learning*, pages 193–216, 2006.
- [52] B. Kveton, M. Valko, A. Rahimi, and L Huang. Semisupervised learning with max-margin graph cuts. In *International Conference on Artificial Intelligence and Statistics*, pages 421–428, 2010.
- [53] Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2009.
- [54] Venkatesan N Ekambaram, Giulia Fanti, Babak Ayazifar, and Kannan Ramchandran. Wavelet-regularized graph semi-supervised learning. In *Global Conference on Signal and Information Processing*, pages 423–426, 2013.
- [55] Benjamin Girault, Paulo Gonçalves, Eric Fleury, and Arashpreet Singh Mor. Semi-supervised learning for graph to signal mapping: A graph signal wiener filter interpretation. In *Conference on Acoustics, Speech and Signal Processing*, pages 1115–1119, 2014.
- [56] M. Belkin, P. Niyogi, and V Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(1):2399–2434, 2006.
- [57] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral net-

- works and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.
- [58] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.
- [59] 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.
- [60] James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.
- [61] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.
- [62] Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pages 499–508, 2018.
- [63] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 941–949, 2018.
- [64] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Pro-*

ceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 5829–5836, 2019.

- [65] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *Proceedings of the International Conference on Learning Representations (ICLR)*, volume 2, page 4, 2019.
- [66] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=BkedznAqKQ>.
- [67] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [68] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.
- [69] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.
- [70] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.
- [71] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

- [72] Shaosheng Cao, Wei Lu, and Qionghai 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. ACM, 2015.
- [73] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Thirty-first AAAI Conference on Artificial Intelligence*, pages 2429–2435, 2017.
- [74] 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.
- [75] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.
- [76] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.
- [77] Shaosheng Cao, Wei Lu, and Qionghai Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1145–1152, 2016.
- [78] Jonathan Chang and David Blei. Relational topic models for document networks. In *Artificial Intelligence and Statistics*, pages 81–88, 2009.

- [79] Rongkai Xia, Yan Pan, Lei Du, and Jian Yin. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2149–2155, 2014.
- [80] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2111–2117, 2015.
- [81] Xiao Wang, Di Jin, Xiaochun Cao, Liang Yang, and Weixiong Zhang. Semantic community identification in large attribute networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 265–271, 2016.
- [82] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 927–936, 2009.
- [83] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining (ICDM)*, pages 1151–1156. IEEE, 2013.
- [84] Dongxiao He, Zhiyong Feng, Di Jin, Xiaobao Wang, and Weixiong Zhang. Joint identification of network communities and semantics via integrative modeling of network topologies and node contents. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 116–124, 2017.
- [85] Aleksandar Bojchevski and Stephan Günnemann. Bayesian robust attributed graph

- clustering: Joint learning of partial anomalies and group structure. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [86] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. Community detection in attributed graphs: an embedding approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 338–345, 2018.
- [87] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.
- [88] 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.
- [89] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2609–2615, 2018.
- [90] 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.
- [91] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.
- [92] 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.
- [93] Gabriel Taubin. A signal processing approach to fair surface design. In *Conference on Computer Graphics and Interactive Techniques*, pages 351–358, 1995.
- [94] Takashi Kurokawa, Taihei Oki, and Hiromichi Nagao. Multi-dimensional graph fourier transform. *arXiv preprint arXiv:1712.07811*, 2017.
- [95] Crowd Source. Kronecker product - Wikipedia. https://en.wikipedia.org/wiki/Kronecker_product, 2022. [Online; accessed 24-June-2022].
- [96] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
- [97] Xiao-Ming Wu, Zhenguo Li, Anthony M. So, John Wright, and Shih-fu Chang. Learning with Partially Absorbing Random Walks. In *Advances in Neural Information Processing Systems*, pages 3077–3085, 2012.
- [98] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.
- [99] 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.
- [100] L.J.P. Van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

- [101] Pietro Perona and William Freeman. A factorization approach to grouping. In *ECCV*, pages 655–670, 1998.
- [102] Charu C Aggarwal and Chandan K Reddy. *Data Clustering: Algorithms and Applications*. CRC Press, Boca Raton, 2014.
- [103] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. *Network*, 11(9):12, 2016.
- [104] Xiao Huang, Jundong Li, and Xia Hu. Accelerated attributed network embedding. In *SIAM ICDM*, pages 633–641, 2017.
- [105] Matthias Hein and Markus Maier. Manifold denoising. In *Advances in Neural Information Processing Systems*, pages 561–568, 2007.
- [106] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [107] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013.
- [108] Charles M Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc, 2012.
- [109] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. In *25th International Conference on Pattern Recognition, ICPR’20*, 01 2021.

- [110] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.
- [111] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [112] Amir Bakarov. A survey of word embeddings evaluation methods. *CoRR*, abs/1801.09536, 2018.
- [113] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1306–1313, 2010.
- [114] Y. LeCun, L. Bottou, Y. Bengio, and P Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [115] Ken Lang. Newsweeder: Learning to filter netnews. In *International conference on machine learning*, pages 331–339, 1995.
- [116] Andrew McCallumzy, Kamal Nigamy, Jason Renniey, and Kristie Seymorey. Building domain-specific search engines with machine learning techniques. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace*, pages 28–39. Citeseer, 1999.
- [117] Robert West, Joelle Pineau, and Doina Precup. Wikispeedia: An online game for inferring semantic distances between concepts. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1598–1603, 2009.

- [118] Robert West and Jure Leskovec. Human wayfinding in information networks. In *WWW*, pages 619–628, 2012.
- [119] Claudio Saccá, Michelangelo Diligenti, and Marco Gori. Collective classification using semantic based regularization. In *2013 12th International Conference on Machine Learning and Applications*, volume 1, pages 283–286, 2013.
- [120] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.
- [121] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, 2020.
- [122] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018.
- [123] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, 2020.
- [124] 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.
- [125] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pages 2121–2129, 2013.
- [126] Soravit Changpinyo, Wei-Lun Chao, Boqing Gong, and Fei Sha. Synthesized classifiers for zero-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5327–5336, 2016.
- [127] Michael Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P Xing. Rethinking knowledge graph propagation for zero-shot learning. *arXiv preprint arXiv:1805.11724*, 2018.
- [128] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
- [129] Qingyao Ai, Daniel N Hill, SVN Vishwanathan, and W Bruce Croft. A zero attention model for personalized product search. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 379–388, 2019.
- [130] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W Bruce Croft. Learning a hierarchical embedding model for personalized product search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 645–654, 2017.

- [131] Keping Bi, Qingyao Ai, and W Bruce Croft. A transformer-based embedding model for personalized product search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1521–1524, 2020.
- [132] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. Learning latent vector spaces for product search. In *Proceedings of the 25th ACM international conference on information and knowledge management*, pages 165–174, 2016.
- [133] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.
- [134] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2015.
- [135] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 188–197, 2019.
- [136] Qingyao Ai, Yongfeng Zhang, Keping Bi, and W Bruce Croft. Explainable product search with a dynamic relation embedding model. *ACM Transactions on Information Systems (TOIS)*, 38(1):1–29, 2019.

- [137] Shang Liu, Wanli Gu, Gao Cong, and Fuzheng Zhang. Structural relationship representation learning with graph embedding for personalized product search. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 915–924, 2020.
- [138] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. Graph embedding based familial analysis of android malware using unsupervised learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 771–782. IEEE, 2019.
- [139] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1507–1515, 2017.
- [140] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394, 2017.
- [141] Amir Hossein Shabani, David A Clausi, and John S Zelek. Improved spatio-temporal salient feature detection for action recognition. In *The British Machine Vision Conference (BMVC)*, pages 1–12. Citeseer, 2011.
- [142] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019, 2016.

- [143] Qingqing Huang, Fengyu Zhou, Jiakai He, Yang Zhao, and Runze Qin. Spatial-temporal graph attention networks for skeleton-based action recognition. *Journal of Electronic Imaging*, 29(5):053003, 2020.
- [144] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Thirty-first AAAI Conference on Artificial Intelligence*, 2017.
- [145] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.