UNIVERSITY OF CALIFORNIA SAN DIEGO

Local-to-global Perspectives on Graph Neural Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree

in

Doctor of Philosophy

by

Chen Cai

Committee in charge:

Professor Jingbo Shang, Chair
Professor Yusu Wang, Co-Chair
Professor Gal Mishne
Professor Rose Yu

2023

EPIGRAPH

Some people may sit back and say, I want to solve this problem and they sit down and say, "How do I solve this problem?" I don't. I just move around in the mathematical waters, thinking about things, being curious, interested, talking to people, stirring up ideas; things emerge and I follow them up. Or I see something which connects up with something else I know about, and I try to put them together and things develop. I have practically never started off with any idea of what I'm going to be doing or where it's going to go. I'm interested in mathematics; I talk, I learn, I discuss and then interesting questions simply emerge. I have never started off with a particular goal, except the goal of understanding mathematics.

– Michael Atiyah

Talk is cheap. Show me the code.

– Linus Torvalds

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to express my gratitude to my Ph.D. advisor, Prof. Yusu Wang. Her guidance, support, and encouragement have been invaluable throughout my doctoral studies. I took a meandering path to computer science and made a switch from computational topology to graph representation learning in the middle of my Ph.D. Such a transition will not be possible without her support. I am grateful for the patience, kindness, vision, and academic freedom she offered.

Besides Yusu, I also want four professors who shaped my research career. I would like to thank Prof. Victoria Sadovskaya for the opportunity of attending the MASS (Mathematics Advanced Study Semesters) program at Penn State University when I was an undergrad. It is one of the best times I will always remember. I also want to thank Prof. Yakov Pesin, who taught me the Erlangen program in the projective geometry course. It converts me into a geometer and leads me to geometric deep learning many years later. I want to thank Prof. Joseph Mitchell from Stony Brook University and Prof. Jie Gao from Rutgers University for introducing me to the field of algorithm design and computational geometry. Joe's kindness and devotion to computational geometry are contagious and left a mark on me. Jie's advice of grounding self in important applications balanced my tendency of going too abstract.

I benefited much from two summer schools, "Mathematical Methods for High-Dimensional Data Analysis" summer school organized by Ulrich Bauer, where I am intrigued by the topological data analysis course given by Steve Oudot, and London Geometry and Machine Learning (LOGML) summer school, where I had the opportunity to work with Haggai Maron on equivariant subgraph aggregation networks. I also want to thank Simons Institute for the Theory of Computing, and Institute for Pure and Applied Mathematics (IPAM) for making most talks available to the public. They have always been great learning resources throughout the years.

I am fortunate to work with many excellent researchers from different disciplines across the world, from whom I learned and enjoyed so much. Thank Bahador Bahmani, Goonmeet Bajaj, Gopinath Balamurugan, Beatrice Bevilacqua, Michael M. Bronstein, Fabrizio Frasca, Saket Gurukar, Rafael Gómez-Bombarelli, Truong Son Hy, Moniba Keymanesh, Woojin Kim, Saravana Kumar, Derek Lim, Ran Ma, Lucas Magee, Pranav Maneriker, Haggai Maron, Facundo Memoli, Benjamin Kurt Miller, Anasua Mitra, Srinivasan Parthasarathy, Vedang Patel, Balaraman Ravindran, Tess Smidt, Aakash Srinivasan, Jian Tang, Priyesh Vijayan, Nikolaos Vlassis, Wujie Wang, Yusu Wang, Dingkang Wang, Teng-Fong Wong, Zeyu Xiong, Rose Yu, Jie Zhang. I want to especially thank Haggai Maron for being a great mentor and continuing to push the limit of equariance research. I want to thank Wujie Wang for turning a serendipity meetup in NeurIPS into a fruitful collaboration and Mingkai Xu for a long-term discussion on AI4Science. I want to thank Tess Smidt and Mario Geiger for the help with e3nn.

I would like to pay special tributes to the researchers I look up to. I want to thank Taco Cohen, Max Welling, Risi Kondor, Leonidas Guibas, Jure Leskovec, Haggai Maron, Joan Bruna, and Michael M. Bronstein for their pioneering work on graph neural networks and geometric deep learning. I want to thank Sanjeev Arora and Mikhail Belkin for their original perspectives on deep learning theory.

I want to thank the friends and mentors I met during my internships. Thank you Stephan Eismann and Raphael Townshend for the great opportunity to work on RNA structural prediction in Atomic.ai. Thank Nick Boyd, Brandon Anderson, Paul Raccuglia, Ruth Brillman, Yuan Zhang and Walid Chaabene for mentorship in Atomic.ai, Google, and Amazon.

I want to thank my friends Shuaicheng Chang, Justin Eldridge, Tao Hou, Like Hui, Wuwei Lan, Tianqi Li, Rui Li, Albert Liang, Lucas Magee, Dingkang Wang, Jiayuan Wang, Cheng Xin, Ryan Slechta, Jiankai Sun, Dayu Shi, Longhua Wu, Hao Zhang, Qi Zhao and Shi Zong in the Ohio State University and Tristan Brugere, Sam Chen, Truong Son Hy, Akbar Rafiey, Puoya Tabaghi, Xingyue Xia, Rui Wang, Libin Zhu and Zhengchao Wan from UC San Diego. I also want to thank Huang Fang, Xueyu Mao, Haiyan Yin, and Yingxue Zhou, whom I got to know during internships and kept in touch since. I want to especially thank Siyuan Ma for many discussions and insights. I want to thank my long-time friends Yiming Chen, Shashan Shu, and Tianyu Zuo who have been with me for more than ten years. I enjoy the time with all my friends and thank you all for making my Ph.D. life so much fun.

I would like to thank Prof. Jingbo Shang, Prof. Rose Yu, and Prof. Gal Mishne who agree to serve on my committee and provide insightful comments and suggestions.

Lastly, I want to thank my wife for her love and support and my parents for raising me as a curious and independent person. This thesis is dedicated to them.

VITA

2015	Bachelor of Science, China Agricultural University
2017	Master of Science, Stony Brook University
2020	Master of Science, Ohio State University
2023	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Chen Cai, Yusu Wang., “Convergence of invariant graph networks.", *International Conference on Machine Learning (ICML)*, 2022.

Chen Cai, Truong Son Hy, Rose Yu, Yusu Wang., “ On the connection between MPNN and Graph Transformer”, *International Conference on Machine Learning (ICML)*, 2023.

Chen Cai, Dingkang Wang, Yusu Wang., “Graph coarsening with neural networks”, *International Conference on Learning Representations (ICLR)*, 2022.

ABSTRACT OF THE DISSERTATION

Local-to-global Perspectives on Graph Neural Networks

by

Chen Cai

in Doctor of Philosophy

University of California San Diego, 2023

Professor Jingbo Shang, Chair
Professor Yusu Wang, Co-Chair

Message Passing Neural Networks (MPNN) has been the leading architecture for machine learning on graphs. Its theoretical study focuses on increasing expressive power and overcoming over-squashing & over-smoothing phenomena. The expressive power study of MPNN suggests that one needs to move from local computation to global modeling to gain expressive power in terms of the Weisfeiler-Lehman hierarchy. My dissertation centers around understanding the theoretical property of global GNN, its relationship to the local MPNN, and how to use local MPNN for coarse-graining. In particular, it consists of three parts:

Convergence of Invariant Graph Network. One type of global GNNs is the so-called

Invariant graph networks (IGN). In the first part, we aim to study the convergence behavior of IGNs, where a similar understanding has already been provided for the local MPNNs. We investigate the convergence of one powerful GNN, Invariant Graph Network (IGN) over graphs sampled from graphons. We first prove the stability of linear layers for general k-IGN (of order k) based on a novel interpretation of linear equivariant layers. Building upon this result, we prove the convergence of k-IGN under the model of [124], where we access the edge weight but the convergence error is measured for graphon inputs. Under the more natural (and more challenging) setting of [78] where one can only access 0-1 adjacency matrix sampled according to edge probability, we first show a negative result that the convergence of any IGN is not possible. We then obtain the convergence of a subset of IGNs, denoted as IGN-small, after the edge probability estimation. We show that IGN-small still contains functions rich enough to approximate spectral GNNs arbitrarily well. Lastly, we perform experiments on various graphon models to verify our statements.

The Connection between MPNN and Graph Transformer. In the second part, we study the connection between local GNN (MPNN) and global GNN (Graph Transformer). Previous work [82] shows that with proper position embedding, GT can approximate MPNN arbitrarily well, implying that GT is at least as powerful as MPNN. Here we study the inverse connection and show that MPNN with virtual node (VN), a commonly used heuristic with little theoretical understanding, is powerful enough to arbitrarily approximate the self-attention layer of GT. In particular, we first show that if we consider one type of linear transformer, the so-called Performer/Linear Transformer, then MPNN + VN with only $O(1)$ depth and $O(1)$ width can approximate a self-attention layer in Performer/Linear Transformer. Next, via a connection between MPNN + VN and DeepSets, we prove the MPNN + VN with $O(n^d)$ width and $O(1)$ depth can approximate the self-attention layer arbitrarily well, where d is the input feature dimension. Lastly, under some (albeit rather strong) assumptions, we provide an explicit construction of MPNN + VN with $O(1)$ width and $O(n)$ depth approximating the self-attention layer in GT arbitrarily well.

Graph Coarsening with Neural Networks. Finally, one way to obtain global information via local MPNN is through graph coarsening, where at a coarser level, edges among super-nodes represent more global connections. However, when performing graph coarsening, one hope to be able to preserve the original graph's properties. The specific property we aim to preserve is its spectral property, which can capture long-range interaction in graphs (e.g., the behavior of random walks). In the last part, we first propose a framework for measuring the quality of coarsening algorithm and show that depending on the goal, we need to carefully choose the Laplace operator on the coarse graph and associated projection/lift operators. Motivated by the observation that the current choice of edge weight for the coarse graph may be sub-optimal, we parametrize the weight assignment map with GNN and train it to improve the coarsening quality in an unsupervised way. Through extensive experiments on both synthetic and real networks, we demonstrate that our method significantly improves common graph coarsening methods under various metrics, reduction ratios, graph sizes, and graph types. It generalizes to graphs of larger size (25x of training graphs), is adaptive to different losses (differentiable and non-differentiable), and scales to much larger graphs than previous work.

Chapter 1

Introduction

1.1 Background

Graphs are flexible representations for modeling complex objects, such as road networks, protein interaction networks, social networks, molecules, and so on. From the methodology perspective, modeling functions on general graphs naturally requires handling greater variability compared to deep learning on images (2d grid) and sequences (1d line graph). This implies that the design of graph neural networks is more challenging than common techniques in processing images and sequences such as CNN, RNN, and Transformer. The study of machine learning & deep learning on graphs is therefore of great theoretical interest and practical significance, and has been extensively studied in recent years [84, 143, 58, 57, 148, 20, 56, 15, 18, 164].

The purpose of this thesis is to provide a local-to-global perspective on the graph neural network (GNN), a leading machine learning architecture for processing graphs. The local approach to GNN results in Message Passing Neural Network (MPNN) that is widely used in practice, which includes popular models like GAT [143], GCN [84], and GraphSAGE [58]. However, the theoretical study of MPNN reveals its limitations, such as limited expressive power, over-smoothing, and over-squashing. Take the expressive power as an example, it is well known MPNN can not be more expressive in terms of distinguishing non-isomorphic graphs than the 1-WL (weisfeiler-lehman) test [156]. Increasing the expressive power of GNN beyond 1-WL has been extensively studied [11, 109, 5, 53, 129]. On the high level, most work requires some elements of global modeling, in the form of modeling high-order interaction or subgraph aggregation. This motivates the study of the global approach to learning on graphs such as Invariant Graph Network (IGN) [110] and Graph Transformer (GT) [86, 159].

The first part of the thesis in Chapter 2 introduces the Invariant Graph Network (IGN), a global GNN, and provides a systematic study of its convergence property. Convergence is closely related to generalization, a central topic in graph neural network research and machine learning in general.

The second part of the thesis in Chapter 3 studies the connection between local MPNNs and global Graph Transformers. It connects the local approach (MPNN) and global approach (Graph Transformer), with DeepSets and Invariant Graph Network (IGN) serving as the conceptual bridge.

One common approach to model long-range interaction modeling on irregular domain is graph coarsening. In Chapter 4, the last part of the thesis, we study the creative use of MPNN to perform graph coarsening. MPNN offers an alternative to classical optimization techniques, with the advantage of generalizing to graphs of different sizes.

We next give a short introduction of models that appeared in the thesis and then provide the outline of the thesis in Section 1.2 and list my contributions in Section 1.3.

1.1.1 Message Passing Neural Network (MPNN)

Message Passing Neural Networks (MPNNs) are a class of neural networks designed to handle structured data, specifically graph-like structures. MPNNs are capable of capturing the complex relations between nodes in a graph, making them ideal for a variety of tasks ranging from social network analysis to chemical structure prediction. They operate through a process known as “message passing”, where nodes in the graph exchange and aggregate information iteratively, thereby enabling the network to learn a representation of the whole graph based on local node features and their connections.

Specifically, the t-th iteration of messaging passing takes the following form

$$m_v^{t+1} = \sum_{w \in N(v)} M_t\left(h_v^t, h_w^t, e_{vw}\right)$$
$$h_v^{t+1} = U_t\left(h_v^t, m_v^{t+1}\right)$$

where h_v^{t+1}, the hidden representation for node v, are updated based on messages m_v^{t+1}. M_t is the message functions, and U_t is the vertex update functions. Many popular graph networks such as GCN [84], GAT [143], and GraphSage [58] can be realized under the MPNN framework.

1.1.2 Invariant Graph Network (IGN)

Invariant Graph Network (IGN) is a class of global GNN that treats graphs as order 2 tensors. Just as CNN interleaves the linear and nonlinear layers, IGN follows the same approach to building graph networks. As there is no canonical node order, the linear layer needs to be both linear and permutation equivariant. Such linear and permutation equivariance constraints dramatically reduce the degrees of freedom. [110] characterizes the space of linear and permutation equivariant functions from tensor of order l to tensor of order m, i.e. all linear permutation equivariant functions from $\mathbb{R}^{n^l}$ to $\mathbb{R}^{n^m}$ is of dimension $Bell(l+m)$. Note that the dimension of space is independent of graph nodes n, and it is the reason why IGN can generalize to graphs of different sizes.

Depending on the order of intermediate tensor representation, IGN can be parameterized by k-IGN, where k is the largest tensor order. It is shown 2-IGN can arbitrarily approximate MPNN, and therefore as least has the 1-WL expressive power. In general, k-IGN has the expressive power of k-WL in terms of distinguishing non-isomorphic graphs, which implies that IGN is a class of highly expressive GNN that deserve further investigation.

1.1.3 Graph Transformer (GT)

We next introduce another class of global GNN, Graph Transformer (GT). Because of the great successes of Transformers in natural language processing (NLP) [142, 151] and recently in computer vision [39, 45, 101], there is great interest in extending transformers for graphs. In particular, it encodes the graph structure into the position embeddings and then applies the Transformer layer to mix the features. One common belief of advantage of graph transformer over MPNN is its capacity in capturing long-range interactions while alleviating over-smoothing [96, 114, 21] and over-squashing in MPNN [3, 140].

Fully-connected Graph transformer [43] was introduced with eigenvectors of graph Laplacian as the node positional encoding (PE). Various follow-up works proposed different ways of PE to improve GT, ranging from an invariant aggregation of Laplacian's eigenvectors in SAN [86], pair-wise graph distances in Graphormer [159], relative PE derived from diffusion kernels in GraphiT [112], and recently Sign and Basis Net [100] with a principled way of handling sign and basis invariance. Other lines of research in GT include combining MPNN and GT [153, 120], encoding the substructures [25], and efficient graph transformers for large graphs [152].

1.2 Outline of Thesis

In Chapter 2, we introduce a class of GNN model named Invariant Graph Network (IGN). IGN is parameterized by the k, order of intermediate tensor representation. The k is a hyperparameter that can be tuned to balance the expressive power (in terms Weisfeiler-Lehman hierarchy) and the computational complexity. We study the convergence property of IGN. The problem is stated as follows: given a continuous graphon model where we can sample a sequence of graphs G_i, we are interested in whether the output of IGN $\phi(G_i)$ will converge. Convergence relates the output of IGN given a sequence of graphs sampled from the same continuous models. Convergence is easier to study than generalization, a central topic in GNN research because we restrict the variability of input graphs to come from the same model. Therefore, studying the convergence may shed light on the generalization of GNN [16].

In Chapter 3, we study the connection between local MPNN and global Graph Transformer. Local MPNN has been widely studied while the global Graph Transformer is a new model that receives a lot of attention recently. One advantage of the Graph Transformer is that its Transformer backbone allows more effective feature mixing than MPNN, which will require many layers to pass information when the radius of the input graph is large. Previous work [82] shows that with proper position embedding, GT can approximate MPNN arbitrarily well, implying that GT is at least as powerful as MPNN. In this chapter, we study the inverse connection and show that MPNN with virtual node (VN), a commonly used heuristic with little theoretical understanding, is powerful enough to arbitrarily approximate the self-attention layer of GT. Our work draws a tighter connection between local and global GNN [23].

In Chapter 4, we study the problem of graph coarsening, which aims to reduce the size of the graph while preserving the essential property. Despite rich graph coarsening literature, there is only limited exploration of data-driven methods in the field. In this chapter, we propose a novel data-driven graph coarsening method based on the message-passing neural network (MPNN) [19].

1.3 Contributions

The first family of popular global GNN we look into is the so-called Invariant graph networks (IGN). In Chapter 2, we study the convergence of k-IGN under two models, the edge weight continuous model, and the edge probability discrete model. We first provide a novel interpretation of the linear permutation equivariant basis of k-IGN for any k, which is interesting on its own. Based on such interpretations, we prove the convergence of k-IGN under the edge weight continuous model. Under the more challenging edge probability discrete model, we first show that convergence of k-IGN is not possible. We then showed that under the preprocessing step of edge smoothing (used in Graphon estimation), we can retain the convergence property of a subset of k-IGN, named IGN-small, under the edge probability discrete model. Lastly, we characterize that IGN-small in some sense is not too small as it still contains the rich class of functions that can approximate spectral GNN arbitrarily well.

Another popular class of global GNN is Graph Transformer (GT). GT recently has emerged as a new paradigm of graph learning algorithms, outperforming the previously popular Message Passing Neural Network (MPNN) on multiple benchmarks. In Chapter 3, we provide a systematic study of the approximation power of MPNN with virtual node (VN). In particular, In particular, we first show that if we consider one type of linear transformer, the so-called Performer/Linear Transformer (Choromanski et al., 2020; Katharopoulos et al., 2020), then MPNN + VN with only $O(1)$ depth and $O(1)$ width can approximate a self-attention layer in Performer/Linear Transformer. Next, via a connection between MPNN + VN and DeepSets, we prove the MPNN + VN with $O(n^d)$ width and $O(1)$ depth can approximate the self-attention layer arbitrarily well, where d is the input feature dimension. Lastly, under some assumptions, we provide an explicit construction of MPNN + VN with $O(1)$ width and O(n) depth approximating the self-attention layer in GT arbitrarily well. On the empirical side, we demonstrate that 1) MPNN + VN is a surprisingly strong baseline, outperforming GT on the recently proposed Long Range Graph Benchmark (LRGB) dataset, 2) our MPNN + VN improves over early implementation on a wide range of OGB datasets.

Finally, one way to obtain global information via local MPNN is through graph coarsening, where at a coarser level, connections among super-nodes represent more global connections. However, when performing graph coarsening, one hopes to be able to preserve the original graph's properties. The specific property we aim to preserve is its spectral property, which can capture long-range interaction in graphs (e.g., the behavior of random walks). In Chapter 4, we explored the use of data-driven methods for graph coarsening. We leverage the recent progress of deep learning on graphs for graph coarsening. We first propose a framework for measuring the quality of coarsening algorithm and show that depending on the goal, we need to carefully choose the Laplace operator on the coarse graph and associated projection/lift operators. Motivated by the observation that the current choice of edge weight for the coarse graph may be sub-optimal, we parametrize the weight assignment map with graph neural networks and train it to improve the coarsening quality in an unsupervised way. Through extensive experiments on both synthetic and real networks, we demonstrate that our method significantly improves common graph coarsening methods under various metrics, reduction ratios, graph sizes, and graph types. It generalizes to graphs of larger size ($25\times$ of training graphs), is adaptive to different losses (differentiable and non-differentiable), and scales to much larger graphs than previous work.

Chapter 2

Convergence of Invariant Graph Networks

2.1 Introduction

In this chapter, we focus on the the convergence property of a class of powerful global GNN called Invariant Graph Networks (IGN). Although theoretical properties of GNN such as expressive power [111, 80, 109, 51, 5, 53, 11] and over-smoothing [96, 114, 21, 168] of GNNs have received much attention, their convergence property is less understood. In this chapter, we systematically investigate the convergence of one of the most powerful families of GNNs, the *Invariant Graph Network (IGN)* [110]. Different from message passing neural network (MPNN) [54], it treats graphs and associated node/edge features as monolithic tensors and processes them in a permutation equivariant manner. 2-IGN can approximate the message passing neural network (MPNN) arbitrarily well on the compact domain. When allowing the use of high-order tensor as the intermediate representation, k-IGN is shown at least as powerful as k-WL test. As the tensor order k goes to $O(n^4)$, it achieves the universality and can distinguish all graphs of size n [111, 80, 5].

The high level question we are interested in is the convergence and stability of GNNs. In particular, given a sequence of graphs sampled from some generative models, does a GNN performed on them also converge to a limiting object? This problem has been considered recently, however, so far, the studies [124, 78] focus on the convergence of *spectral GNNs*, which encompasses several models [13, 33] including GCNs with order-1 filters [84]. However, it is known that the expressive power of GCN is limited. Given that 2(k)-IGN is strictly more powerful than GCN [156] in terms of separating graphs[1] and its ability to achieve universality, it is of great interest to study the convergence of such powerful GNN. In fact, it is posted as an open question in [79] to study convergence for models more powerful than spectral GNNs and higher order GNNs. This is the question we aim to study in this chapter.

Contributions. We present the first convergence study of the powerful k-IGNs (strictly more powerful than the Spectral GNN which previous work studied). We first analyze the building block of IGNs: linear equivariant layers, and develop a stability result for such layers. The case of 2-IGN is proved via case analysis while the general case of k-IGN uses a novel interpretation of the linear equivariant layers which we believe is of independent interest.

There have been two existing models of convergence of spectral GNNs for graphs sampled from graphons developed in [124] and [78], respectively. Using the model of [124] (denoted by the *edge weight continuous model*) where we access the edge weight but the convergence error is measured between *graphon inputs* (see Section 2.4 for details), we obtain analogous convergence results for k-IGNs. The results cover both deterministic and random sampling for **k-IGN** while [124] only covers deterministic sampling for the much weaker **Spectral GNN**s.

Under more natural (and more challenging) setting of [78] where one can only access 0-1 adjacency matrix sampled according to edge probability (called the *edge probability discrete model*), we first show a negative result that in general the convergence of all IGNs is not possible. Building upon our earlier stability result, we obtain the convergence of a subset of IGN, denoted as IGN-small, after a step of edge probability estimation. We show that IGN-small still contains rich function class that can approximate Spectral GNN arbitrarily well. Lastly, we perform experiments on various graphon models to verify our statements.

[1] In terms of separating graphs, k-IGN $>$ 2-IGN $=$ GIN $>$ GCN for $k > 2$.

Table 2.1. Linear equivariant maps for $\mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ and $\mathbb{R}^{[0,1]^2} \to \mathbb{R}^{[0,1]^2}$. $\mathbf{1}$ is a all-one vector of size $n \times 1$ and $\mathrm{I}_{u=v}$ is the indicator function.

Operations	Discrete	Continuous	Partitions
1-2: The identity and transpose operations	$T(A) = A$	$T(W) = W$	$\{\{1,3\},\{2,4\}\}$
	$T(A) = A^T$	$T(W) = W^T$	$\{\{1,4\},\{2,3\}\}$
3: The diag operation	$T(A) = \mathrm{Diag}(\mathrm{Diag}^*(A))$	$T(W)(u,v) = W(u,v)\mathrm{I}_{u=v}$	$\{\{1,2,3,4\}\}$
4-6: Average of rows replicated on rows/ columns/ diagonal	$T(A) = \frac{1}{n}A\mathbf{1}\mathbf{1}^T$	$T(W)(*,u) = \int W(u,v)dv$	$\{\{1,4\},\{2\},\{3\}\}$
	$T(A) = \frac{1}{n}\mathbf{1}(A\mathbf{1})^T$	$T(W)(u,*) = \int W(u,v)dv$	$\{\{1,3\},\{2\},\{4\}\}$
	$T(A) = \frac{1}{n}\mathrm{Diag}(A\mathbf{1})$	$T(W)(u,v) = \mathrm{I}_{u=v}\int W(u,v')dv'$	$\{\{1,3,4\},\{2\}\}$
7-9: Average of columns replicated on rows/ columns/ diagonal	$T(A) = \frac{1}{n}A^T\mathbf{1}\mathbf{1}^T$	$T(W)(*,v) = \int W(u,v)du$	$\{\{1\},\{2,4\},\{3\}\}$
	$T(A) = \frac{1}{n}\mathbf{1}(A^T\mathbf{1})^T$	$T(W)(v,*) = \int W(u,v)du$	$\{\{1\},\{2,3\},\{4\}\}$
	$T(A) = \frac{1}{n}\mathrm{Diag}(A^T\mathbf{1})$.	$T(W)(u,v) = \mathrm{I}_{u=v}\int W(u',v)du'$	$\{\{1\},\{2,3,4\}\}$
10-11: Average of all elements replicated on all matrix/ diagonal	$T(A) = \frac{1}{n^2}(\mathbf{1}^TA\mathbf{1})\cdot\mathbf{1}\mathbf{1}^T$	$T(W)(*,*) = \int W(u,v)dudv$	$\{\{1\},\{2\},\{3\},\{4\}\}$
	$T(A) = \frac{1}{n^2}(\mathbf{1}^TA\mathbf{1})\cdot\mathrm{Diag}(\mathbf{1})$.	$T(W)(u,v) = \mathrm{I}_{u=v}\int W(u',v')du'dv'$	$\{\{1\},\{2\},\{3,4\}\}$
12-13: Average of diagonal elements replicated on all matrix/diagonal	$T(A) = \frac{1}{n}(\mathbf{1}^T\mathrm{Diag}^*(A))\cdot\mathbf{1}\mathbf{1}^T$	$T(W)(*,*) = \int \mathrm{I}_{u=v}W(u,v)dudv$	$\{\{1,2\},\{3\},\{4\}\}$
	$T(A) = \frac{1}{n}(\mathbf{1}^T\mathrm{Diag}^*(A))\cdot\mathrm{Diag}(\mathbf{1})$	$T(W)(u,v) = \mathrm{I}_{u=v}\int W(u',u')du'$	$\{\{1,2\},\{3,4\}\}$
14-15: Replicate diagonal elements on rows/columns	$T(A) = \mathrm{Diag}^*(A)\mathbf{1}^T$	$T(W)(u,v) = W(u,u)$	$\{\{1,2,4\},\{3\}\}$
	$T(A) = \mathbf{1}\mathrm{Diag}^*(A)^T$	$T(W)(u,v) = W(v,v)$	$\{\{1,2,3\},\{4\}\}$

Table 2.2. Linear equivariant maps for $\mathbb{R}^{n} \to \mathbb{R}^{n\times n}$ and $\mathbb{R}^{[0,1]} \to \mathbb{R}^{[0,1]^2}$.

Operations	Discrete	Continuous	Partitions
1-3: Replicate to diagonal/rows/columns	$T(A) = \mathrm{Diag}(A)$	$T(W)(u,v) = \mathrm{I}_{u=v}W(u)$	$\{\{1,2,3\}\}$
	$T(A)_{i,j} = A_i$	$T(W)(u,v) = W(u)$	$\{\{1,3\},\{2\}\}$
	$T(A)_{i,j} = A_j$	$T(W)(u,v) = W(v)$	$\{\{1,2\},\{3\}\}$
4-5: Replicate mean to diagonal/all matrix	$T(A)_{i,i} = \frac{1}{n}A\mathbf{1}$	$T(W)(u,v) = \mathrm{I}_{u=v}\int W(u)du$	$\{\{1\},\{2,3\}\}$
	$T(A)_{i,j} = \frac{1}{n}A\mathbf{1}$	$T(W)(u,v) = \int W(u)du$	$\{\{1\},\{2\},\{3\}\}$

Table 2.3. Linear equivariant maps for $\mathbb{R}^{n\times n} \to \mathbb{R}^{n}$ and $\mathbb{R}^{[0,1]^2} \to \mathbb{R}^{[0,1]}$.

Operations	Discrete	Continuous	Partitions
1-3: Replicate diagonal/row mean/ columns mean	$T(A) = \mathrm{Diag}^*(A)$	$T(W)(u) = W(u,u)$	$\{\{1,2,3\}\}$
	$T(A) = \frac{1}{n}A\mathbf{1}$	$T(W)(u) = \int W(u,v)dv$	$\{\{1,2\},\{3\}\}$
	$T(A) = \frac{1}{n}A^T\mathbf{1}$	$T(W)(u) = \int W(u,v)du$	$\{\{1,3\},\{2\}\}$
4-5: Replicate mean of all elements/ mean of diagonal	$T(A)_i = \frac{1}{n^2}\mathbf{1}^TA\mathbf{1}$	$T(W)(u) = \int W(u,v)dudv$	$\{\{1\},\{2\},\{3\}\}$
	$T(A)_i = \frac{1}{n}\mathbf{1}^T\mathrm{Diag}(\mathrm{Diag}^*(A))\mathbf{1}$	$T(W)(u) = \int \mathrm{I}_{u,v}W(u,v)dudv$	$\{\{1,2\},\{3\}\}$

2.2 Preliminaries

2.2.1 Notations

To talk about convergence/stability, we will consider graphs of different sizes sampled from a generative model. Similar to the earlier work in this direction, the specific general model we consider is a graphon model.

Graphons. A graphon is a bounded, symmetric and measurable function $W : [0,1]^2 \to [0,1]$. We denote the space of graphon as $\mathscr{W}$. It can be intuitively thought of as an undirected weighted graph with an uncountable number of nodes: roughly speaking, given $u_i, u_j \in [0,1]$, we can consider there is an edge (i, j) with weight $W(u_i, u_j)$. Given a graphon W, we can sample **unweighted** graphs of any size from W, either in a deterministic or stochastic manner. We defer the definition of the sampling process until we introduce the edge weight continuous model in Section 2.4 and edge probability discrete model in Section 2.5.

Tensor. Let $[n]$ denote $\{1,...,n\}$. A tensor X of order k, called a *k-tensor*, is a map from $[n]^{\otimes k}$ to $\mathbb{R}^d$. If we specify a name name_i for each axis, we then say X is indexed by $(\text{name}_1, ..., \text{name}_k)$. With slight abuse of notation, we also write that $X \in \mathbb{R}^{n^k \times d}$. We refer to d as the *feature dimensions* or the *channel dimensions*. If $d = 1$, then we have a k-tensor $\mathbb{R}^{n^k \times 1} = \mathbb{R}^{n^k}$. Although the name for each axis acts as an identifier and can be given arbitrarily, we will use *set* to name each axis in this chapter. For example, given a 3-tensor X, we use $\{1\}$ to name the first axis, $\{2\}$ for the second axis, and so on. The benefits of doing so will be clear in Section 2.3.2.

Partition. A partition of $[k]$, denoted as γ, is defined to be a set of disjoint sets $\gamma := \{\gamma_1, ..., \gamma_s\}$ with $s \leqslant k$ such that the following condition satisfies, 1) for all $i \in [s], \gamma_i \subset [k]$, 2) $\gamma_i \cap \gamma_j = \emptyset, \forall\, i, j \in [s]$, and 3) $\cup_{i=1}^{s} \gamma_i = [k]$. We denote the space of all partitions of $[k]$ as Γ_k. Its cardinality is called the k-th *bell number* $\text{bell}(k) = |\Gamma_k|$.

Other conventions. By default, we use 2-norm (Frobenius norm) to refer ℓ_2 norm for all vectors/matrices and L_2 norm for functions on $[0,1]$ and $[0,1]^2$. $\|\cdot\|_2$ or $\|\cdot\|$ denotes the 2 norm for discrete objects while $\|W\|_{L_2} := \int\int W(u,v)dudv$ denotes the norm for continuous objects. Similarly, we use $\|\cdot\|_\infty$ and $\|\cdot\|_{L_\infty}$ to denotes the infinity norm. When necessary, we use $\|\cdot\|_{L_2([0,1])}$ to specify the support explicitly. We use $\|\cdot\|_{\text{spec}}$ to denote spectral norm. Φ_c and Φ_d refers to the continuous IGN and discrete IGN respectively. We sometimes call a function $f : [0,1] \to \mathbb{R}^d$ a *graphon signal*. Given $A \in \mathbb{R}^{n^k \times d_1}, B \in \mathbb{R}^{n^k \times d_2}$, $[A,B]$ is defined to be the concatenation of A and B along feature dimensions, i.e., $[A,B] \in \mathbb{R}^{n^k \times (d_1+d_2)}$. See Table 3.2 for the full symbol list.

2.2.2 Invariant Graph Network

Definition 1. *An Invariant Graph Network (IGN) is a function $\Phi : \mathbb{R}^{n^2 \times d_0} \to \mathbb{R}^d$ of the following form:*

$$F = h \circ L^{(T)} \circ \sigma \circ \cdots \circ \sigma \circ L^{(1)}, \tag{2.1}$$

where each $L^{(t)}$ is a linear equivariant (LE) layer [110] from $\mathbb{R}^{n^{k_{t-1}} \times d_{t-1}}$ to $\mathbb{R}^{n^{k_t} \times d_t}$ (i.e., mapping a k_{t-1} tensor with d_{t-1} channels to a k_t tensor with d_t channels), σ is nonlinear activation function, h is a linear invariant layer from k_T-tensor $\mathbb{R}^{n^{k_T} \times d_T}$ to vector in $\mathbb{R}^d$. d_t is the channel number, and k_t is tensor order in t-th layer.

Let $\text{Diag}(\cdot)$ be the operator of constructing a diagonal matrix from vector and $\text{Diag}^*(\cdot)$ be the operation of extracting a diagonal from a matrix. Under the IGN framework, we view a graph with n nodes as a 2-tensor: In particular, given its adjacency matrix A_n of size $n \times n$ with node features $X_n \in \mathbb{R}^{n \times d_{\text{node}}}$ and edge features $E_{n\times n} \in \mathbb{R}^{n^2 \times d_{\text{edge}}}$, the input of IGN is the concatenation of $[A_n, \text{Diag}(X_n), E_{n\times n}] \in \mathbb{R}^{n^2 \times (1+d_{\text{node}}+d_{\text{edge}})}$ along different channels. We drop the subscript when there is no confusion. We use 2-**IGN** to denote the IGN whose largest tensor order within any intermediate layer is 2, while k-**IGN** is one whose largest tensor order across all layers is k. We use IGN to refer to the general IGN for any order k.

Without loss of generality, we consider input and output tensor to have a single channel. Consider all linear equivariant maps from $\mathbb{R}^{n^\ell}$ to $\mathbb{R}^{n^m}$, denoted as $\text{LE}_{\ell+m}$. [110] characterizes the basis of the space of $\text{LE}_{\ell,m}$. It turns out that the cardinality of the basis equals to the bell

Table 2.4. Summary of important notations.

Symbol	Meaning
$\mathbf{1}_n$	all-one vector of size $n \times 1$
$\|\cdot\|_2/\|\cdot\|_{L_2}$	2-norm for matrix/ graphon
$\|\cdot\|_\infty/\|\cdot\|_{L_\infty}$	infinity-norm for matrix/graphon
$[\cdot,\cdot]$	Given $A \in \mathbb{R}^{n^k \times d_1}, B \in \mathbb{R}^{n^k \times d_2}$, $[A,B]$ is the concatenation of A and B along feature dimension. $[A,B] \in \mathbb{R}^{n^k \times (d_1+d_2)}$.
$W:[0,1]^2 \to [0,1]$	graphon
$X \in \mathbb{R}^{[0,1]\times d}$	1D signal
$\mathscr{W}$	space of graphons
$\|\cdot\|_{\text{pn}}$	partition-norm. When the underlying norm is L_∞ norm, we also use $\|\cdot\|_{\text{pn}-\infty}$.
I	indicator function
I	interval
SGNN	spectral graph neural networks, defined in Equation (2.14)
$\text{LE}_{\ell,m}$	linear equivariant maps from ℓ-tensor to m-tensor
	Notations related to sampling
W_n	Induced piecewise constant graphon from fixed grid
$\widetilde{W_n}$	Induced piecewise constant graphon from random grid
$\widetilde{W_{n,E}}$	Induced piecewise constant graphon from random grid, but resize the all individual blocks to be of equal size (also called chessboard graphon in this chapter). $\widetilde{W_{n,E}}(I_i \times I_j) = W(u_{(i)}, u_{(j)})$
$W_{n\times n}$	$n \times n$ matrix sampled from W; $W_{n\times n}(i,j) = W(u_i,u_j)$
$\widehat{W}_{n\times n} \in \mathbb{R}^{n\times n}$	the estimated edge probability from graphs sampled according to edge probability discrete model from [165]
$\widetilde{x_n} \in \mathbb{R}^{n\times d}$	sampled signal $[\widetilde{x_n}]_i := X(u_i)$
X_n	induced 1D piecewise graphon signal from fixed grid
$\widetilde{X_n}$	induced 1D piecewise graphon signal from random grid
S_U	normalized sampling operator for random grid. $S_U f(i,j) = \frac{1}{n}(f(u_{(i)}), f(u_{(j)})$
S_n	normalized sampling operator for fixed grid. $S_n f(i,j) = \frac{1}{n}(f(\frac{i}{n}), f(\frac{j}{n}))$
$\text{RMSE}_U(x,f)$	$\left(n^{-1}\sum_{i=1}^n \|x_i - f(u_i)\|^2\right)^{1/2}$ for 1D signal; $\left(n^{-2}\sum_i\sum_j \left\|x_{i,j} - f\left(u_i,u_j\right)\right\|^2\right)^{1/2}$ for 2D case
α_n	a parameter that controls the sparsity of sample graphs. Set to be 1 in the chapter.
	Notations related to IGN
$\text{bell}(k)$	Bell number: number of partitions of $[k]$. $\text{bell}(2)=2, \text{bell}(3)=5, \text{bell}(4)=15, \text{bell}(5)=52$...
Γ_k	space of all partitions of $[k]$
$\mathscr{I}_k$	the space of indices. $\mathscr{I}_k := \{(i_1,...,i_k) \| i_1 \in [n],...,i_k \in [n]\}$. Elements of $\mathscr{I}_k$ is denoted as $\boldsymbol{a}$
$\gamma \in [k]$	partition of $[k]$. For example $\{\{1,2\},\{3\}\}$ is a partition of [3]. The total number of partitions of $[k]$ is $\text{bell}(k)$.
$\boldsymbol{a} \in \gamma$	$\boldsymbol{a}$ satisfies the equivalence pattern of γ. For example, $(x,x,y) \in \{\{1,2\},\{3\}\}$ where $x,y,z \in [n]$.
$\gamma < \beta$	given two partitions $\gamma,\beta \in \Gamma_k$, $\gamma < \beta$ if γ is finer than β. For example, $\{1,2,3\} < \{\{1,2\},\{3\}\}$.
$\boldsymbol{B}_\gamma$	$l+m$ tensor; tensor representation of $\text{LE}_{l,m}$ maps. we differentiate T_γ (operators) from $\boldsymbol{B}_\gamma$ (tensor representation of operators)
$\mathscr{B}$	a basis of the space of linear equivariant operations from ℓ-tensor to m-tensor. $\mathscr{B} = \{T_\gamma \| \gamma \in \Gamma_{l+k}\}$
T_c/T_d	linear equivariant layers for graphon (continuous) and graphs (discrete)
Φ_c/Φ_d	IGN for graphon (continuous) and graphs (discrete)
$L^{(i)}$	i-th linear equivariant layer of IGN
L	normalized graph Laplacian
T_i	basis element of the space of linear equivariant maps; sometimes also written as T_γ.

number $\mathrm{bell}(\ell+m)$, thus depending only on the order of input/output tensor and independent from graph size n. As an example, we list a specific basis of the space of LE maps for 2-IGN (thus with tensor order at most 2) in Tables 2.1 to 2.3 when input/output channel numbers are both 1. Extending the LE layers to multiple input/output channels is straightforward, and can be achieved by parametrizing the LE layers according to indices of input/output channel. See Remark 1. Note that one difference of the operators in Tables 2.1 to 2.3 from those given in the original paper is that here we normalize all operators appropriately w.r.t. the graph size n. (This normalization is also in the official implementation of the IGN paper.) This is necessary when we consider the continuous limiting case.

Remark 1 (multi-channel IGN contains MLP)**.** *For simplicity, in the main text, we focus on the case when the input and output tensor channel number is 1. The general case of multiple input and output channels is presented in Equation 9 of [110]. The main takeaway is that permutation equivariance does not constrain the mixing over feature channels, i.e., the space of linear equivariant maps from* $\mathbb{R}^{n^\ell \times d_1} \to \mathbb{R}^{n^m \times d_2}$ *if of dimension* $d_1 d_2 \mathrm{bell}(l+m)$*. Therefore IGN contains MLP.*

To talk about convergence, one has to define the continuous analog of IGN for graphons. In Tables 2.1 to 2.3 we extend all LE operators defined for graphs to graphons, resulting in the continuous analog of 2-IGN, denoted as 2-cIGN or Φ_c in the remaining text. Similar operation can be done in general for k-IGN as well, where the basis elements for k-IGNs will be described in Section 2.3.2.

Definition 2 (2-cIGN)**.** *By extending all* LE *layers for* 2*-IGN to the graphon case as shown in Tables 2.1 to 2.3, we can definite the corresponding 2-cIGN via Eq.* (2.1).

2.2.3 Edge Probability Estimation from [165]

We next restate the setting and theorem regarding the theoretical guarantee of the edge probability estimation algorithm.

Definition 3. *For any $\delta, A_1 > 0$, let $\mathscr{F}_{\delta;L}$ de note a family of piecewise Lipschitz graphon functions $f : [0,1]^2 \to [0,1]$ such that (i) there exists an integer $K \geq 1$ and a sequence $0 = x_0 < \cdots < x_K = 1$ satisfying $\min_{0 \leqslant s \leqslant K-1} (x_{s+1} - x_s) \geq \delta$, and (ii) both $|f(u_1,v) - f(u_2,v)| \leqslant A_1 |u_1 - u_2|$ and $|f(u,v_1) - f(u,v_2)| \leqslant A_1 \mid v_1 - v_2 \mid$ hold for all $u, u_1, u_2 \in [x_s, x_{s+1}], v, v_1, v_2 \in [x_t, x_{t+1}]$ and $0 \leqslant s,t \leqslant K-1$*

Assume that α_n = 1. It is easy to see that the setup considered in [165] is slightly more general than the setup in [78]. The statistical guarantee of the edge smoothing algorithm is stated below.

Theorem 2.2.1 ([165]). *Assume that A_1 is a global constant and $\delta = \delta(n)$ depends on n, satisfying $\lim_{n\to\infty} \delta/(n^{-1}\log n)^{1/2} \to \infty$. Then the estimator $\tilde{P}$ with neighborhood $\mathscr{N}_i$ defined in [165] and $h = C(n^{-1}\log n)^{1/2}$ for any global constant $C \in (0,1]$, satisfies $\max_{f \in \mathscr{F}_{\delta;A_1}} \mathrm{pr}\{d_{2,\infty}(\tilde{P},P)^2 \geq C_1(\frac{\log n}{n})^{1/2}\} \leqslant n^{-C_2}$ where C_1 and C_2 are positive global constants. Here, $d_{2,\infty}(P,Q) = n^{-1/2}\|P-Q\|_{2,\infty} = \max_i n^{-1/2}\|P_i - Q_i\|_2$.*

2.3 Stability of Linear Layers in IGN

In this section, we first show a stability result for a single linear layer of IGN. That is, given two graphon W_1, W_2, we show that if $\|W_1 - W_2\|_{\mathrm{pn}}$ is small, then the distance between the objects after applying a single LE layer remain close. Here $\|\cdot\|_{\mathrm{pn}}$ is a partition-norm that will be introduced in a moment. Similar statements also hold for the discrete case when the input is a graph. We first describe how to prove stability for 2-(c)IGN as a warm-up. We then prove it for k-(c)IGN, which is significantly more interesting and requires a new interpretation of the elements in a specific basis of the space of LE operators in [110].

A the general LE layer $T : \mathbb{R}^{n^\ell} \to \mathbb{R}^{n^m}$can be written as $T = \sum_\gamma c_\gamma T_\gamma$, where $T_\gamma \in \mathscr{B} := \{T_\gamma | \gamma \in \Gamma_{\ell+m}\}$ is the basis element of the space of $\mathrm{LE}_{\ell,m}$ and c_γ are denoted as filter coefficients. Hence proving the stability of T can be reduced to showing the stability for each element in $\mathscr{B}$, which we focus from now on.

2.3.1 Stability of Linear Layers of 2-IGN

A natural way to show stability is by showing that the spectral norm of each LE operator in a basis is bounded. However, even for 2-IGN, as we see some LE operator requires replicating “diagonal elements to all rows" (e.g., operator 14-15 in Table 2.1), and has unbounded spectral norm. To address this challenge, we need a more refined analysis. In particular, below we will introduce a “new" norm that treats the diagonal differently from non-diagonal elements for the 2-tensor case. We term it *partition-norm* as later when handling high order *k*-IGN, we will see that this norm arises naturally w.r.t. the partition of index set of tensors.

Definition 4 (Partition-norm)**.** *The* partition-norm *of 2-tensor $A \in \mathbb{R}^{n^2}$ is defined as* $\|A\|_{\text{pn}} := (\frac{\|Diag^*(A)\|_2}{\sqrt{n}}, \frac{\|A\|_2}{n})$. *The continuous analog of the partition-norm for graphon $W \in \mathscr{W}$ is defined as* $\|W\|_{\text{pn}} = \left(\sqrt{\int W^2(u,u)du}, \sqrt{\iint W^2(u,v)dudv}\right)$.

We refer to the first term as the normalized diagonal norm *and the second term as the* normalized matrix norm. *Furthermore, we define operations like addition/comparison on the partition-norm simply as component-wise operations. For example, $\|A\|_{pn} \leq \|B\|_{pn}$ if each of the two terms of A is at most the corresponding term of B.*

As each term in partition-norm is a norm on different parts of the input, the partition-norm is also a norm. By summing over the finite feature dimension both for finite and infinite cases, the definition of the partition-norm can be extended to multi-channel tensors $\mathbb{R}^{n^2 \times d}$ and its continuous version $\mathbb{R}^{[0,1]^2 \times d}$. See Section 2.9.1 for details.

The following result shows that each basis operation for 2-IGN, shown in Tables 2.1, 2.2 and 2.3, is stable w.r.t. the partition-norm. Hence a LE layer consisting of a finite combination of these operations will remain stable. The proof is via a case-by-case analysis and can be found in Section 2.9.1.

Proposition 2.3.1. *For all* LE *operators $T_i : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$ of discrete 2-IGN listed in Table 2.1, $\|T_i(A)\|_{\text{pn}} \leqslant \|A\|_{pn}$ for any $A \in \mathbb{R}^{n^2}$. Similar statements hold for $T_i : \mathbb{R}^n \to \mathbb{R}^{n^2}$ and $T_i : \mathbb{R}^{n^2} \to \mathbb{R}^n$ in Tables 2.2 and 2.3. In the case of continuous 2-cIGN, the stability also holds.*

Remark 2. *Note that this also implies that given $W_1, W_2 \in \mathscr{W}$, we have that $||T_i(W_1) - T_i(W_2)||_{pn} \leq ||W_1 - W_2||_{pn}$. Similarly, given $A_1, A_2 \in \mathbb{R}^{n^2 \times 1} = \mathbb{R}^{n^2}$, we have $||T_i(A_1) - T_i(A_2)||_{pn} \leq ||A_1 - A_2||_{pn}$.*

2.3.2 Stability of Linear Layers of k-IGN

We now consider the more general case of k-IGN. In principle, the proof of 2-IGN can still be extended to k-IGN, but going through all bell(k) number of elements of LE basis of k-IGN one by one can be quite cumbersome. In the next two subsections, we provide a new interpretation of elements of the basis of space of $\mathrm{LE}_{\ell,m}$ in a unified framework so that we can avoid a case-by-case analysis. Such an interpretation, detailed in Section 2.3.3, is potentially of independent interest. First, we need some notations.

Definition 5 (Equivalence pattern)**.** *Given a k-tensor X, denote the space of its indices $\{(i_1, ..., i_k) \mid i_1 \in [n], ..., i_k \in [n]\}$ by $\mathscr{I}_k$. Given X, $\gamma = \{\gamma_1, ..., \gamma_d\} \in \Gamma_k$ and an element $\boldsymbol{a} = (a_1, ..., a_k) \in \mathscr{I}_k$, we say $\boldsymbol{a} \in \gamma$ if $i, j \in \gamma_l$ for some $l \in [d]$ always implies $a_i = a_j$. Alternatively, we also say $\boldsymbol{a}$ satisfies the equivalence pattern of γ if $\boldsymbol{a} \in \gamma$.*

As an example, suppose $\gamma = \{\{1,2\},\{3\}\}$. Then $(x,x,y) \in \gamma$ while $(x,y,z) \notin \gamma$. Equivalence patterns can induce "slices"/sub-tensors of a tensor.

Definition 6 (Slice/sub-tensor of $X \in \mathbb{R}^{n^k \times 1}$ for $\gamma \in \Gamma_k$)**.** *Let $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor indexed by $(\{1\}, ..., \{k\})$. Consider a partition $\gamma = \{\gamma_1, ..., \gamma_{k'}\} \in \Gamma_k$ of cardinality $k' \leqslant k$. The* slice (sub-tensor) of X induced by γ *is a k'-tensor X_γ, indexed by $(\gamma_1, ..., \gamma_{k'})$, and defined to be $X_\gamma(j_1, ..., j_{k'}) := X(\iota_\gamma(j_1, ..., j_{k'}))$ where $j_\cdot \in [n]$ and $\iota_\gamma(j_1, ..., j_{k'}) \in \gamma$. $\iota_\gamma : [n]^{k'} \to [n]^k$ is defined to be $\iota_\gamma(j_1, ..., j_{k'}) := (i_1, ..., i_k)$ such that $\{a,b\} \subseteq \gamma_c$ implies $i_a = i_b := j_c$. Here $a, b \in [k], c \in [k']$. As an example, we show five slices of a 3-tensor in Figure 2.1.*

Consider the LE operators from $\mathbb{R}^{n^\ell}$ to $\mathbb{R}^{n^m}$. Each such map T_γ can be represented by a matrix of size $n^\ell \times n^m$ which can further considered as a $(\ell + m)$-tensor $\boldsymbol{B}_\gamma$. [110] showed that a specific basis for such operators can be characterized as follows: Each basis element will correspond to one of the bell$(\ell+m)$ partitions in $\Gamma_{\ell+m}$. In particular, given a partition $\gamma \in \Gamma_{\ell+m}$, we have a corresponding basis LE operator T_γ and its tensor representation $\mathbf{B}_\gamma$ defined as follows:

$$\text{for any } \boldsymbol{a} \in \mathscr{I}_{\ell+m}, \ \ \mathbf{B}_\gamma(\boldsymbol{a}) = \begin{cases} 1 & \boldsymbol{a} \in \gamma \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The collection $\mathscr{B} = \{T_\gamma \mid \gamma \in \Gamma_{\ell+m}\}$ form a basis for all $\mathrm{LE}_{\ell,m}$ maps. In Section 2.3.3, we will provide an interpretation of each element of $\mathscr{B}$, making it easy to reason its effect on an input tensor using a unified framework.

Before the main theorem, we also need to extend the partition-norm in Definition 4 from 2-tensor to high-order tensor. Intuitively, for $X \in \mathbb{R}^{n^k}$, $\|X\|_{\text{pn}}$ has bell(k) components, where each component corresponds to the normalized norm of X_γ, the slice of X induced by $\gamma \in \Gamma_k$. See Figure 2.1 for examples of slices of a 3-tensor. The partition-norm of input and output of a $\mathrm{LE}_{\ell,m}$ will be of dimension bell(ℓ) and bell(m) respectively. See Section 2.9.1 for details.

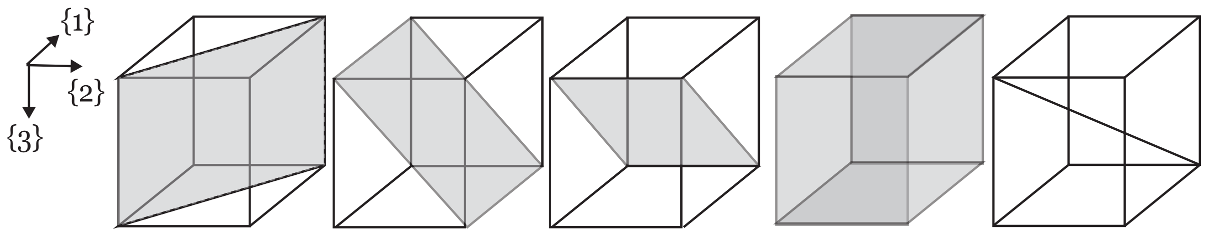


Figure 2.1. Five possible "slices" of a 3-tensor, corresponding to bell(3) = 5 partitions of [3]. From left to right: a) $\{\{1,2\},\{3\}\}$ b) $\{\{1\},\{2,3\}\}$ c) $\{\{1,3\},\{2\}\}$ d) $\{\{1\},\{2\},\{3\}\}$ e) $\{\{1,2,3\}\}$.

The following theorem characterizes the effect of each operator in $\mathscr{B}$ in terms of partition-norm of input and output, generalizing Theorem 2.3.1 from matrix to high order tensor.

Theorem 2.3.2 (Stability of LE layers for k-IGN)**.** *Let $T_\gamma : \mathbb{R}^{[0,1]^\ell} \to \mathbb{R}^{[0,1]^m}$ be a basis element of the space of* $\mathrm{LE}_{\ell,m}$ *maps where $\gamma \in \Gamma_{\ell+m}$. If $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\text{bell}(\ell)}$, then the partition-norm of $Y := T_\gamma(X)$ satisfies $\|Y\|_{pn} \leqslant \varepsilon \mathbf{1}_{\text{bell}(m)}$ for all $\gamma \in \Gamma_{\ell+m}$.*

The proof relies on a new interpretation of elements of $\mathscr{B}$ in k-IGN. We give only an intuitive sketch using an example in the next subsection. See Section 2.9.1 for the proof.

2.3.3 Interpretation of Basis Elements

For better understanding, we color the input axis $\{1,...,\ell\}$ as red and output axis $\{\ell+1,...,\ell+m\}$ as blue. Each T_γ corresponds to one partition γ of $[\ell+m]$.

For any partition $\gamma \in \Gamma_{l+k}$, we can write this set as disjoint union $\gamma = S_1 \cup S_2 \cup S_3$ where S_1 is a set of set(s) of input axis, and S_3 is a set of set(s) of output axis. S_2 is a set of set(s) where each set contains both input and output axis. With slight abuse of notation, we omit the subscript γ for S_1, S_3, S_3 when its choice is fixed or clear, and denote $\{\ell+1,...,\ell+m\}$ as $\ell+[m]$. As an example, one basis element of the space of $\mathrm{LE}_{3,3}$ maps is $\gamma = \{\{1,2\},\{3,6\},\{4\},\{5\}\}$

$$\underbrace{S_1 = \{\{1,2\}\}}_{\text{Only has input axis}} \cup \underbrace{S_2 = \{\{3,6\}\}}_{\substack{\text{has both} \\ \text{input and output axis}}} \cup \underbrace{S_3 = \{\{4\},\{5\}\}}_{\text{only has output axis}} \tag{2.3}$$

where $1,2,3$ specifies the axis of input tensor and $4,5,6$ specifies the axis of the output tensor. Recall that there is a one-to-one correspondence between the partitions over $[\ell+m]$ and the base

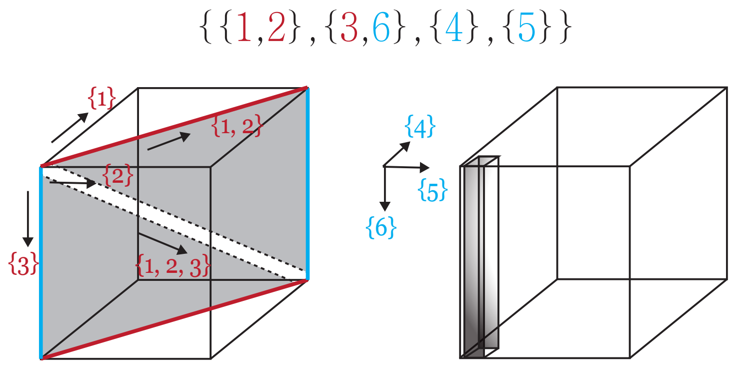


Figure 2.2. An illustration of the one basis element of the space of $\mathrm{LE}_{3,3}$. It selects area spanned by axis $\{1,2\}$ and $\{3\}$, average over the axis $\{1,2\}$, and then align the resulting 1D tensor with axis $\{6\}$, and finally replicate the slices along axis $\{4\}$ and $\{5\}$ to fill in the whole cube on the right.

elements in $\mathscr{B}$ as in Eqn (2.3.2). The basis element T_γ corresponding to $\gamma = S_1 \cup S_2 \cup S_3$ operates on an input tensor $X \in \mathbb{R}^{n^\ell}$ and produce an output tensor $Y \in \mathbb{R}^{n^m}$ as follows:

> Given input X, (step 1) obtain its slice X_γ on Π_1 (selection axis), (step 2) average X_γ over Π_2 (reduction axis), resulting in $X_{\gamma,\text{reduction}}$. (step 3) Align $X_{\gamma,\text{reduction}}$ on Π_3 (alignment axis) with Y_γ and (step 4) replicate Y_γ along Π_4 (replication axis), resulting $Y_{\gamma,\text{replication}}$, a slice of Y. Entries of Y outside $Y_{\gamma,\text{replication}}$ will be set to be 0.

In general, Π_i can be read off from S_1-S_3. See Section 2.9.1 for details. As a running example, Figure 2.2 illustrates the basis element corresponding to $\gamma = S_1 \cup S_2 \cup S_3$ where $S_1 = \{\{1,2\}\} \cup S_2 = \{\{3,6\}\} \cup S_3 = \{\{4\},\{5\}\}$. In the first step, given 3-tensor X, indexed by $\{\{1\},\{2\},\{3\}\}$ we select slices of interest X_γ on $\Pi_1 = \{\{1,2\},\{3\}\}$, colored in grey in the left cube of Figure 2.2. In the second step, we average X_γ over axis $\Pi_2 = \{\{1,2\}\}$ to reduce 2-tensor X_γ, indexed by $\{\{1,2\},\{3\}\}$ to a 1-tensor $X_{\gamma,\text{reduction}}$, indexed by $\{\{3\}\}$. In the third step, the $X_{\gamma,\text{reduction}}$ is aligned with $\Pi_3 = \{\{6\}\}$, resulting in the grey cuboid Y_γ indexed by $\{\{6\}\}$, shown in the right cube in Figure 2.2. Here the only difference between $X_{\gamma,\text{reduction}}$ and Y_γ is the index name of two tensors. In the fourth step, we replicate the grey cuboid Y_γ over axis $\Pi_4 = \{\{4\},\{5\}\}$ to fill in the cube, resulting in $Y_{\gamma,\text{replication}}$, indexed by $\{\{3\},\{4\},\{5\}\}$. Note in general $Y_{\gamma,\text{replication}}$ is a slice of Y and does have to be the same as Y.

These steps are defined formally in the Section 2.9. For each of the four steps, we can control the partition-norm of output for each step (shown in Theorem 2.9.2), and therefore control the partition-norm of the final output for every basis element. See Section 2.9.1 for full proofs.

2.4 Convergence of IGN in the Edge Weight Continuous Model

[124] consider the convergence of $\|\Phi_c(W) - \Phi_c(W_n)\|_{L_2}$ in the *graphon space*, where W is the original graphon and W_n is a piecewise constant graphon induced from graphs of size n sampled from W (to be defined soon). We call this model as the *edge weight continuous model*. The main result of [125] is the convergence of *continuous* spectral GNN in the *deterministic* sampling case where graphs are sampled from W deterministically. Leveraging our earlier

stability result of linear layers of continuous IGNs in Theorem 2.3.2, we can prove an analogous convergence result of cIGNs in the edge weight continuous model for both the deterministic and random sampling cases.

Setup of the edge weight continuous model. Given a graphon $W \in \mathscr{W}$ and a signal $X \in \mathbb{R}^{[0,1]\times d}$, the input of cIGN will be $[W, \mathrm{Diag}(X)] \in \mathbb{R}^{[0,1]^2\times(1+d)}$. In the random sampling setting, we sample a graph of size n from W by setting the following edge weight matrix and discrete signal:

$$[\widetilde{A_n}]_{ij} := W(u_i, u_j) \quad \text{and} \quad [\widetilde{x_n}]_i := X(u_i) \tag{2.4}$$

where u_i is the i-th smallest point from n i.i.d points sampled from uniform distribution on $[0,1]$. We further lift the discrete graph $(\widetilde{A_n}, \widetilde{x_n})$ to a piecewise-constant graphon $\widetilde{W_n}$ with signal $\widetilde{X_n}$. Specifically, partition $[0,1]$ to be $I_1 \cup \ldots \cup I_n$ with $I_i = (u_i, u_{i+1}]$. We then define

$$\begin{aligned} \widetilde{W_n}(u,v) &:= [\widetilde{A_n}]_{ij} \times \mathrm{I}(u \in I_i)\mathrm{I}(v \in I_j) \quad \text{and} \\ \widetilde{X_n}(u) &:= [\widetilde{x_n}]_i \times \mathrm{I}(u \in I_i) \end{aligned} \tag{2.5}$$

where I is the indicator function. Replacing the random sampling with fixed grid, i.e., let $u_i = \frac{i-1}{n}$, we can get the deterministic edge weight continuous model, where W_n and X_n can be defined similarly as the lifting of a discrete sampled graph to a piecewise constant graphon. Note that $\widetilde{W_n}$ is a piecewise constant graphon where each block is not of the same size, while all blocks W_n are of size $\frac{1}{n} \times \frac{1}{n}$. We use $\widetilde{\cdot}$ to emphasize that $\widetilde{W_n}/\widetilde{X_n}$ are random variables, in contrast to the deterministic W_n/X_n.

We also need a few assumptions on the input and IGN.

AS1. *The graphon W is A_1-Lipschitz, i.e. $|W(u_2,v_2) - W(u_1,v_1)| \leqslant A_1(|u_2-u_1| + |v_2-v_1|)$.*

AS2. *The filter coefficients c_γ are upper bounded by A_2.*

AS3. *The graphon signal X is A_3-Lipschitz.*

AS4. *The activation functions in IGNs are normalized Lipschitz, i.e.* $|\rho(x) - \rho(y)| \leqslant |x-y|$*, and* $\rho(0) = 0$*.*

Such four assumptions are quite natural and also adopted in [124]. With AS 1-4, we have the following key proposition. The proof leverages the stability of linear layers for k-IGN from Theorem 2.3.2; see Section 2.9.2 for details.

Proposition 2.4.1 (Stability of Φ_c)**.** *If cIGN* $\Phi_c : \mathbb{R}^{[0,1]^2 \times d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ *satisfy AS2, AS4 and* $\|W_1 - W_2\|_{pn} \leqslant \varepsilon \mathbf{1}_2$*, then* $\|\Phi_c(W_1) - \Phi_c(W_2)\|_{pn} = \|\Phi_c(W_1) - \Phi_c(W_2)\|_{L_2} \leqslant C(A_2)\varepsilon$ *. The same statement still holds if we change the underlying norm of Partition-norm from* L_2 *to* L_∞*.*

Remark 3. *Statements in Theorem 2.9.6 holds for discrete IGN* Φ_d *as well.*

From AS3 we can also bound the difference between the original signal X and the induced signal (X_n and $\widetilde{X_n}$).

Lemma 2.4.2. *Let* $X \in \mathbb{R}^{[0,1] \times d}$ *be an* A_3*-Lipschitz graphon signal satisfying AS3, and let* $\widetilde{X_n}$ *and* X_n *be the induced graphon signal as in Eqs.* (2.4) *and* (2.5)*. Then we have i)* $\|X - X_n\|_{pn}$ *converges to 0 and ii)* $\|X - \widetilde{X_n}\|_{pn}$ *converges to 0 in probability.*

We have the similar statements for W as well.

Lemma 2.4.3. *If* W *satisfies AS1,* $\|W - W_n\|_{\text{pn}}$ *converges to 0.* $\|W - \widetilde{W_n}\|_{\text{pn}}$ *converges to 0 in probability.*

The following main theorem (for k-cIGN of any order k) of this section can be shown by combining Theorem 2.9.6 with Theorems 2.4.2 and 2.4.3; see Section 2.9.2 for details.

Theorem 2.4.4 (Convergence of cIGN in the edge weight continuous model)**.** *Under the fixed sampling condition, IGN converges to cIGN, i.e.,* $\|\Phi_c([W, Diag(X)]) - \Phi_c([W_n, Diag(X_n)])\|_{L_2}$ *converges to 0.*

An analogous statement hold for the random sampling setting, where $\|\Phi_c([W, Diag(X)]) - \Phi_c([\widetilde{W_n}, Diag(\widetilde{X_n})])\|_{L_2}$ *converges to 0 in probability.*

2.5 Convergence of IGN in the Edge Probability Discrete Model

In this section, we will consider the convergence setup of [78], which we call the *edge probability discrete model*. The major difference from the edge weight continuous model of [124] is that (1) we only access 0-1 adjacency matrix instead of full edge weights and (2) the convergence error is measured in the graph space (instead of graphon space).

This model is more natural. However, we will first show a negative result that in general IGN does not converge in the edge probability discrete model in Section 2.5.2. This motivates us to consider a relaxed setting where we estimate the edge probability from data. With this extra assumption, we can prove the convergence of IGN-small, a subset of IGN, in the edge probability discrete model in Section 2.5.3. Although this is not entirely satisfactory, we show that nevertheless, the family of functions that can be represented by IGN-small is still rich enough to for example approximate any spectral GNN arbitrarily well.

2.5.1 Setup: Edge Probability Continuous Model

We first state the setup and results of [78]. We keep the notation close to the original paper for consistency. A random graph model (P, W, f) is represented as a probability distribution P uniform over latent space $\mathscr{U} = [0,1]$, a symmetric kernel $W : \mathscr{U} \times \mathscr{U} \to [0,1]$ and a bounded function (graph signal) $f : \mathscr{U} \to \mathbb{R}^{d_z}$. A random graph G_n with n nodes is then generated from (P, W, f) according to latent variables $U := \{u_1, ..., u_n\}$ as follows:

$$\forall j < i \leqslant n: \quad \text{graph node } u_i \overset{iid}{\sim} P, \quad z_i = f(u_i), \tag{2.6}$$

$$\text{graph edge } a_{ij} \sim \text{Ber}\left(\alpha_n W(u_i, u_j)\right) \tag{2.7}$$

where Ber is the Bernoulli distribution and α_n controls the sparsity of sampled graph. Note that in our case, we assume that the sparsification factor $\alpha_n = 1$ (which is the classical graphon model). We define a degree function by $d_{W,P}(\cdot) := \int W(\cdot,u)dP(u)$. We assume the following

$$\|W(\cdot,u)\|_{L_\infty} \leqslant c_{\max}, \quad d_{W,P}(u) \geqslant c_{\min}, \tag{2.8}$$

$$W(\cdot,u) \text{ is } \left(c_{\text{Lip.}}, n_{\mathscr{U}}\right)\text{-piecewise Lipschitz.} \tag{2.9}$$

A function $f : \mathscr{U} \to \mathbb{R}$ is said to be $\left(c_{\text{Lip.}}, n_{\mathscr{U}}\right)$-piecewise Lipschitz if there is a partition $\mathscr{U}_1, \ldots, \mathscr{U}_n$ of $\mathscr{U}$ such that, for all u, u' in the same $\mathscr{U}_i$, we have $|f(u) - f(u')| < c_{Lip.}d(u,u')$. We introduce two normalized sampling operator S_U and S_n that sample a continuous function to a discrete one over n points. For a function $W' : \mathscr{U}^{\otimes k} \to \mathbb{R}^{d_{\text{out}}}$, $S_U W'(i_1, \ldots, i_k) := (\frac{1}{\sqrt{n}})^k (W'(u_{(i_1)}), \ldots, W'(u_{(i_k)}))$ where $u_{(i)}$ is the i-th smallest number over n uniform random samples over $[0,1]$ and $i_1, \ldots, i_k \in [n]$. Similarly, $S_n W'(i_1, \ldots, i_k) := (\frac{1}{\sqrt{n}})^k \left(W'(\frac{i_1}{n}), \ldots, W'(\frac{i_k}{n})\right)$ Note that the normalizing constant will depend on the dimension of the support of W'. We have $\|S_U W'\|_2 \leqslant \|W'\|_{L_\infty}$ and $\|S_n W'\|_2 \leqslant \|W'\|_{L_\infty}$.

To measure the convergence error, we consider root mean square error at the node level: for a signal $x \in \mathbb{R}^{n^2 \times d_{\text{out}}}$ and latent variables U, we define $\text{RMSE}_U(f,x) := \|S_U f - \frac{x}{n}\|_2 = (n^{-2} \sum_{i=1}^n \sum_{j=1}^n \left\|f\left(u_i, u_j\right) - x(i,j)\right\|^2)^{1/2}$. Again, there is a dependency on the input dimension – the normalization term n^{-2} will need to be adjusted when the input order is different from 2.

2.5.2 Negative Result

Theorem 2.5.1. *Given any graphon W with $c_{max} < 1$ and an IGN architecture (fix hyperparameters like number of layers), there exists a set of parameters θ such that convergence of IGN$_\theta$ to cIGN$_\theta$ is not possible, i.e.,* $\text{RMSE}_U(\Phi_c([W, Diag(X)]), \Phi_d([A_n, Diag(\widetilde{x}_n)]))$ *does not converge to 0 as $n \to \infty$, where A_n is 0-1 matrix generated according to Eq.* (2.7)*, i.e., $A_n[i][j] = a_{i,j}$.*

The proof of Theorem 2.5.1 hinges on the fact that the input to IGN in discrete case is 0-1 matrix while the input to cIGN in the continuous case has edge weight upper bounded by $c_{\max} < 1$. The margin between 1 and $c_{\max}$ makes it easy to construct counterexamples. See

Section 2.9.3 for details.

Theorem 2.5.1 states that we cannot expect every IGN will converge to its continuous version cIGN. As the proof of this theorem crucially uses the fact that we can only access 0-1 adjacency matrix, a natural question is what if we can estimate the edge probability from the data? Interestingly, we can obtain the convergence of for a subset of IGNs (which is still rich enough), called IGN-small, in this case.

2.5.3 Convergence of IGN-small

Let $\widehat{W}_{n\times n}$ be the estimated $n \times n$ edge probability matrix from A_n. $\widetilde{W_n}$ is the induced graphon defined in Eq. (2.5). To analyze the convergence error for general IGN after edge probability estimation, we first decompose the convergence error of the interest using triangle inequality. Assuming the output is 1-tensor, then

$$\begin{aligned}
&\mathrm{RMSE}_U(\Phi_c(W), \Phi_d(\widehat{W}_{n\times n})) \\
&= \|S_U \Phi_c(W) - \frac{1}{\sqrt{n}} \Phi_d(\widehat{W}_{n\times n})\| \\
&\leqslant \underbrace{\|S_U \Phi_c(W) - S_U \Phi_c(\widetilde{W_n})\|}_{\text{First term: discretization error}} + \underbrace{\|S_U \Phi_c(\widetilde{W_n}) - \Phi_d S_U(\widetilde{W_n})\|}_{\text{Second term: sampling error}} \\
&+ \underbrace{\|\Phi_d S_U(\widetilde{W_n}) - \frac{1}{\sqrt{n}} \Phi_d(\widehat{W}_{n\times n})\|}_{\text{Third term: estimation error}}
\end{aligned} \tag{2.10}$$

The three terms measure the different sources of error. First-term is concerned with the discretization error, which can be controlled via a property of S_U and Theorem 2.9.6. The Second term concerns the sampling error from the randomness of U. This term will vanish if we consider only S_n instead of S_U under the extra condition stated below. The third term concerns the edge probability estimation error, which can also be controlled by leveraging existing literature on the

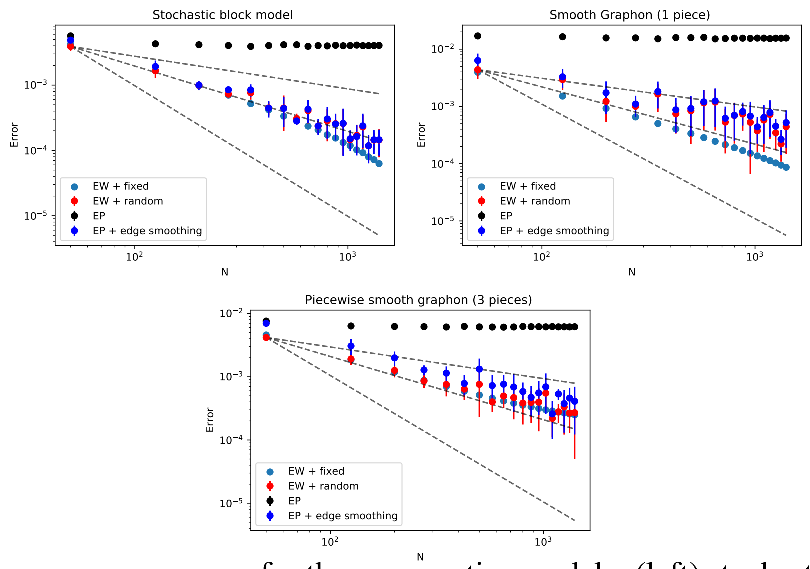


Figure 2.3. The convergence error for three generative models: (left) stochastic block model, (middle) smooth graphon, (right) piece-wise smooth graphon. EW and EP stands for edge weight continuous model (Eq. (2.4)) and edge probability discrete model (Eq. (2.7)).

statistical guarantee of the *edge probability estimation* algorithm from [165]. [2]

Controlling the second term is more involved. This is also the place where we have to add an extra assumption to constrain the IGN space in order to achieve convergence after edge smoothing.

Definition 7 (IGN-small)**.** *Let $\widetilde{W_{n,E}}$ be a graphon with "chessboard pattern" [3], i.e., it is a piecewise constant graphon where each block is of the same size. Similarly, define $\widetilde{X_{n,E}}$ as the 1D analog. IGN-small denotes a subset of IGN that satisfies $S_n\Phi_c([\widetilde{W_{n,E}}, Diag(\widetilde{X_{n,E}})]) = \Phi_d S_n([\widetilde{W_{n,E}}, Diag(\widetilde{X_{n,E}})])$.*

Theorem 2.5.2 (convergence of IGN-small in the edge probability discrete model)**.** *Assume AS 1-4, and let $\widehat{W}_{n\times n}$ be the estimated edge probability that satisfies $\frac{1}{n}\|W_{n\times n} - \widehat{W}_{n\times n}\|_2$ converges to 0 in probability. Let Φ_c, Φ_d be continuous and discrete IGN-small. Then*
$\mathrm{RMSE}_U\left(\Phi_c\left([W, Diag(X)]\right), \Phi_d\left([\widehat{W}_{n\times n}, Diag(\tilde{x}_n)]\right)\right)$ *converges to 0 in probability.*

[2]For better readability, here we only use the W as input instead of $[W, \mathrm{Diag}(X)]$. Adding $\mathrm{Diag}(X)$ into the input is easy and is included in the full proof in Section 2.9.3.

[3]See full definition in Definition 12.

We leave the detailed proofs in Section 2.9.3 with some discussion on the challenges for achieving full convergence results in the Remark 6. We note that Theorem 2.5.2 has a practical implication: It suggests that in practice, for a given unweighted graph (potentially sampled from some graphon), it may be beneficial to first perform edge probability estimation before feeding into the general IGN framework, to improve the architecture's stability and convergence.

Finally, although the convergence of IGN-small is not entirely satisfactory, it contains some interesting class of functions that can approximate any spectral GNN arbitrarily well. See Section 2.9.4 for proof details.

Theorem 2.5.3. *IGN-small can approximates spectral GNN (both discrete and continuous ones) arbitrarily well on the compact domain in the $\|\cdot\|_{L_\infty}$ sense.*

2.6 Experiments

We experiment 2-IGN on three graphon models of increasing complexity: Erdoes Renyi graph with $p = 0.1$, stochastic block model of 2 blocks of equal size and probability matrix $[[0.1, 0.25], [0.25, 0.4]]$, a Lipschitz graphon model with $W(u,v) = \frac{u+v+1}{4}$, and a piecewise Lipschitz graphon with $W(u,v) = \frac{u\%\frac{1}{3}+v\%\frac{1}{3}+1}{4}$ where % is modulo operation. Similar to [78], we consider untrained IGN with random weights to assess how convergence depends on the choice of architecture rather than learning. We use a 5-layer IGN with hidden dimension 16. We take graphs of different sizes as input and plot the error in terms of the norm of the output difference. The results are plotted in Figure 3.3.

As suggested by the Theorem 2.4.4, for both deterministic and random sampling, the error decreases as we increase the size of the sampled graph. Interestingly, if we take the 0-1 adjacency matrix as the input, the error does not decrease, which aligns with the negative result in Theorem 2.5.1. We further implement the edge smoothing algorithm [47] and find that after the edge probability estimation, the error again decreases, as implied by Theorem 2.5.2. We remark that although Theorem 2.5.2 works only for IGN-small, our experiments for the general 2-IGN with randomized initialized weights still show encouraging convergence results. Understanding the convergence of general IGN after edge smoothing is an important direction that we will leave for further investigation.

2.7 Related Work

One type of convergence in deep learning concerns the limiting behavior of neural networks when the width goes to infinity [73, 41, 4, 91, 40]. In that regime, the gradient flow on a normally initialized, fully connected neural network with a linear output layer in the infinite-width limit turns out to be equivalent to kernel regression with respect to the Neural Tangent Kernel [73].

Another type of convergence concerns the limiting behavior of neural networks when the depth goes to infinity. In the continuous limit, models such as residual networks, recurrent neural network decoders, and normalizing flows can be seen as an Euler discretization of an ordinary differential equation [150, 27, 106, 126].

The type of convergence we consider in this chapter concerns when the input objects converge to a limit, does the output of some neural network over such sequence of objects also converge to a limit? In the context of GNNs, such convergence and related notion of stability and transferability have been studied in both graphon [124, 78, 50, 125] and manifold setting [85, 95]. In the manifold setting, the analysis is closely related to the literature on convergence of Laplacian operator [155, 149, 9, 10, 35].

Lastly, after ICML 2022 conference (where the papaer this chapter is based on is published) it is brought to our attention that the characterization of linear permutation equivariant layers in k-IGN bears similarity in [1]. The pooling and broadcasting operations in [1] are the same as what we call the "averaging" and "replication" operations in this chapter. This is discussed in details in Remark 4.

2.8 Concluding Remarks

in this chapter, we investigate the convergence property of a powerful GNN, Invariant Graph Network. We first prove a general stability result of linear layers in IGNs. We then prove a convergence result under the model of [124] for both 2-IGN and high order k-IGN. Under the model of [78] we first show a negative result that in general the convergence of every IGN is not possible. Nevertheless, we pinpoint the major roadblock and prove that if we preprocess input graphs by edge smoothing [165], the convergence of a subfamily of IGNs, called IGN-small, can be obtained. As an attempt to quantify the size of IGN-small, we also show that IGN-small contains a rich class of functions that can approximate any spectral GNN.

2.9 Missing proofs

2.9.1 Missing Proofs from Section 2.3

Extension of Partition-norm

There are three ways of extending Partition-norm 1) extend the definition of partition-norm to multiple channels 2) changing the underlying norm from L_2 norm to L_∞ norm, and 3) extend Partition-norm defined for 2-tensor to k-tensor.

First recall the definition partition-norm.

Definition 4 (Partition-norm)**.** *The* partition-norm *of 2-tensor $A \in \mathbb{R}^{n^2}$ is defined as* $\|A\|_{\text{pn}} := (\frac{\|Diag^*(A)\|_2}{\sqrt{n}}, \frac{\|A\|_2}{n})$*. The continuous analog of the partition-norm for graphon $W \in \mathscr{W}$ is defined as* $\|W\|_{\text{pn}} = \left(\sqrt{\int W^2(u,u)du}, \sqrt{\iint W^2(u,v)dudv}\right)$.

We refer to the first term as the normalized diagonal norm *and the second term as the* normalized matrix norm. *Furthermore, we define operations like addition/comparison on the partition-norm simply as component-wise operations. For example, $\|A\|_{pn} \leq \|B\|_{pn}$ if each of the two terms of A is at most the corresponding term of B.*

To extend partition-norm to signal $A \in \mathbb{R}^{n^2 \times d}$ of multiple channels, we denote $A = [A_{\cdot,1} \in$

$\mathbb{R}^{n^2\times 1}, ..., A_{\cdot,d} \in \mathbb{R}^{n^2\times 1}]$ where $[\cdot,\cdot]$ is the concatenation along channels. $\|A\|_{\text{pn}} := \sum_{i=1}^{d} \|A_{\cdot,i}\|_{\text{pn}}$. both for multi-channel signal both for graphs and graphons.

Another way of generalizing Partition-norm is to change the L_2 to L_∞ norm. We denote the resulting norm as $\|\cdot\|_{\text{pn}-\infty}$. For $W \in \mathscr{W}$, $\|W\|_{\text{pn}-\infty} := (\max_{u\in[0,1]} W(u,u), \max_{u\in[0,1],v\in[0,1]} W(u,v))$. The discrete case and high order tensor case can be defined similarly as the L_2 case.

The last way of extending Partition-norm to k-tensor $X \in \mathbb{R}^{n^k\times 1}$ is to define the norm for each slice of X, i.e., $\|X\|_{\text{pn}} := ((\frac{1}{\sqrt{n}})^{|\gamma_1|}\|X_{\gamma_1}\|_2, ..., \frac{1}{\sqrt{n}})^{|\gamma_{\text{bell}(k)}|}\|X_{\gamma_{\text{bell}(k)}}\|_2)$ where $\gamma_\cdot \in \Gamma_k$. Note how we order $(\gamma_1, ..., \gamma_{\text{bell}(k)})$ can be arbitrary as long as the order is used consistent.

Proof of stability of linear layer for 2-IGN

Proposition 2.9.1. *For all* LE *operators $T_i : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$ of discrete 2-IGN listed in Table 2.1, $\|T_i(A)\|_{\text{pn}} \leqslant \|A\|_{pn}$ for any $A \in \mathbb{R}^{n^2}$. Similar statements hold for $T_i : \mathbb{R}^n \to \mathbb{R}^{n^2}$ and $T_i : \mathbb{R}^{n^2} \to \mathbb{R}^n$ in Tables 2.2 and 2.3. In the case of continuous 2-cIGN, the stability also holds.*

Proof. The statements hold in both discrete and continuous cases. Without loss of generality, we only prove the continuous case by going over all linear equivariant maps $\mathbb{R}^{[0,1]^2} \to \mathbb{R}^{[0,1]^2}$ in Table 2.1.

- 1-3: It is easy to see that the partition-norm does not increase for all three cases.
- 4-6: It is enough to prove case 4 only. Since $T(W)(*,u) = \int W(u,v)dv$, diagonal norm $\|\text{Diag}(T(W))\|_{L_2}^2 = \int(\int W(u,v)dv)^2 du \leqslant \iint W^2(u,v)dudv$. For matrix norm: $\|T(W)\|_{L_2}^2 = \|\text{Diag}(T(W))\|_{L_2} \leqslant \iint W^2(u,v)dudv$. Therefore the statement holds for this linear equivariant operation.
- 7-9: same as case 4-6.
- 10-11: It is enough to prove the first case: average of all elements replicated on the whole matrix. The diagonal norm is the same as the matrix norm. Both norms are decreasing so we are done.

- 12-13: It is enough to prove only case 12. Since diagonal norm is equal to matrix norm, and diagonal norm is decreasing by Jensen's inequality we are done.
- 14-15: Since matrix norm is the same as diagonal norm, which stays the same so we are done.

As shown in all cases for any $W \in \mathscr{W}$ with $\|W\|_{\text{pn}} < (\varepsilon, \varepsilon)$, $\|T_i(W)\|_{\text{pn}} < (\varepsilon, \varepsilon)$. Therefore we finish the proof for $\mathbb{R}^{[0,1]^2} \to \mathbb{R}^{[0,1]^2}$. We next go over all linear equivariant maps $\mathbb{R}^{[0,1]} \to \mathbb{R}^{[0,1]^2}$ in Table 2.2 and prove it case by case.

- 1-3: It is enough to prove the second case. It is easy to see diagonal norm is preserved and $\|T(W)\|_2 = \|W\|_2 \leqslant \varepsilon$. Therefore $\|T(W)\|_{\text{pn}} \leqslant (\varepsilon, \varepsilon)$.
- 4-5: It is enough to prove the second case. Norm on diagonal is no larger than $\|W\|$ by Jensen's inequality. The matrix norm is the same as the diagonal norm therefore also no large than ε. Therefore $\|T(W)\|_{\text{pn}} \leqslant (\varepsilon, \varepsilon)$.

Last, we prove the cases for $\mathbb{R}^{[0,1]^2} \to \mathbb{R}^{[0,1]}$.

For cases 1-3, it is enough to prove case 2. Since the norm of the output is no large than the matrix norm of input by Jensen's inequality, we are done. Similar reasoning applies to cases 4-5 as well. □

Proof of Theorem 2.3.2

We need a few definitions and lemmas first.

Definition 8 (axis of a tensor)**.** *Given a k-tensor $X \in \mathbb{R}^{n^k \times 1}$ indexed by* $(\text{name}_1, ..., \text{name}_k)$. *The axis of X, denoted as ax(X), is defined to be ax$(X) :=$* $(\text{name}_1, ..., \text{name}_k)$.

As an example, the aixs of the first grey sub-tensor in Figure 2.5a, which is a 2-tensor, is $\{\{1,2\},\{3\}\}$.

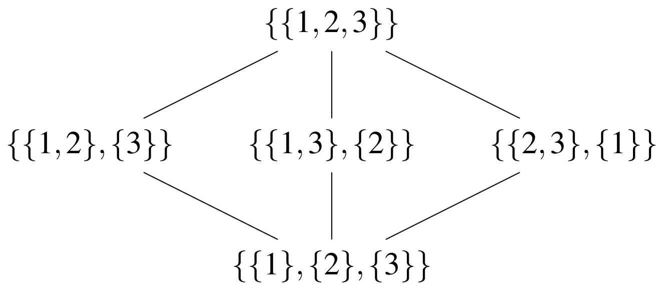


Figure 2.4. Space of partitions forms a Hasse diagram under the partial order defined in Definition 10.
Top to bottom corresponds to coarse partition to finer partition.

Definition 9 (replication of a tensor)**.** *Given a k-tensor $X \in \mathbb{R}^{n^k \times 1}$ indexed by $(1, ..., k)$, replicating X over new axis $(k+1, ..., k+d)$ means that the resulting new tensor X' of $k+d$ dimension is $X'(i_1, ..., i_k, *, ..., *) := X(i_1, ..., i_k)$.*

Definition 10 (partial order of partitions)**.** *Given two partitions of $[k]$, denoted as $\gamma = \{\gamma_1, ..., \gamma_{d_1}\}$ and $\beta = \{\beta_1, ..., \beta_{d_2}\}$, we say γ is finer than β, denoted as $\gamma < \beta$, if and only if 1) $\gamma \neq \beta$ and 2) for any $\beta_j \in \beta$, there exists $\gamma_i \in \gamma$ such that $\beta_j \subseteq \gamma_i$.*

For example, $\{\{1,2,3\}\}$ is finer than $\{\{1,2\},\{3\}\}$ but $\{\{1,2\},\{3\}\}$ is not comparable with $\{\{1,3\},\{2\}\}$. Note that space of partitions forms a Hasse diagram under the partial order defined above (each set of elements has a least upper bound and a greatest lower bound, so that it forms a lattice). See Figure 2.4 for an example.

Definition 11 (average a k-tensor X over Π)**.** *Let $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor indexed by $\{\{1\}, ..., \{k\}\}$. Without loss of generality, let $\Pi = \{\{1\}, ..., \{d\}\}$. Denote the resulting $(k-d)$-tensor X', indexed by $\{\{d+1\}, ..., \{k\}\}$. By averaging X over Π, we mean*

$$X'(\cdot) := \frac{1}{n^d} \sum_{t \in \mathcal{I}_d} X(t, \cdot).$$

The definition can be extended to $\mathbb{R}^{[0,1]^k}$ by replacing average with integral.

Lemma 2.9.2 (properties of partition-norm)**.** *We list some properties of the partition-norm. Although all lemmas are stated in the discrete case, the continuous version also holds. The statements also holds for $\|\cdot\|_{pn-\infty}$ as well.*

(a) Let $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor and denote one of its slices $X' \in \mathbb{R}^{n^{k'} \times 1}$ with $k' \leqslant k$. If $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k)}$, then $\|X'\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k')}$.

(b) Let $k' < k$. Let $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor and $X' \in \mathbb{R}^{n^{k'} \times 1}$ be the resulting k'-tensor after averaging over $k-k'$ axis of X. If $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k)}$, then $\|X'\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k')}$.

(c) Let $k' > k$. Let $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor and X' be the resulting k'-tensor after replicating X over $k'-k$ axis of X'. If $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k)}$, then $\|X'\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k')}$.

(d) Let $k' < k$ and $X \in \mathbb{R}^{n^k \times 1}$ be a k-tensor such that it has only one non-zero slice X_γ of order k', i.e., if $\boldsymbol{a} \in \mathscr{I}_k, X(\boldsymbol{a}) \neq 0$, it implies $\boldsymbol{a} \in \gamma$. If $\|X_\gamma\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k')}$, then $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k)}$.

Proof. We prove statements one by one. Note that although the proof is done for L_2 norm, we do not make use of any specific property of L_2 norm and the same proof can be applied to L_∞ as well. Therefore all statements in the lemma apply to $\|\cdot\|_{\mathrm{pn}-\infty}$ as well.

1. By the definition of partition-norm and slice in Definition 6, we know that any slice of X' is also a slice of X, therefore any component of $\|X'\|_{\mathrm{pn}}$ will be upper bounded by ε, which concludes the proof.

2. Without loss of generality, we can assume that $k' = k-1$ as the general case can be handled by induction. Let the axis of X that is averaged over is axis $\{1\}$. To bound $\|X'\|_{\mathrm{pn}}$, we need to bound the normalized norm of any slice of X'. Let $X'_{\gamma'}$ be arbitrary slice of X'. Since X' is obtained by averaging over axis 1 of X, we know that $X'_{\gamma'}$ is the obtained by averaging over axis of 1 of X_γ, a slice of X, where $\gamma := \gamma' \cup \{\{1\}\}$. Since $\|X\|_{\mathrm{pn}} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(k)}$, we know that $(\frac{1}{\sqrt{n}})^{|\gamma|}\|X_\gamma\| \leqslant \varepsilon$. By Jensen's inequality, we have $(\frac{1}{\sqrt{n}})^{|\gamma'|}\|X'_{\gamma'}\| \leqslant (\frac{1}{\sqrt{n}})^{|\gamma|}\|X_\gamma\|$, and

therefore $(\frac{1}{\sqrt{n}})^{|\gamma'|}\|X'_{\gamma'}\| \leqslant \varepsilon$. Since $(\frac{1}{\sqrt{n}})^{|\gamma'|}\|X'_{\gamma'}\| \leqslant \varepsilon$ holds for arbitrary slice of X', we conclude that $\|X'\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(k')}$.

The proof above only handles the case of $k' = k-1$. The general case where $k - k' > 1$ can be handled by evoking the proof above multiple times for different reduction axis.

3. Similar to the Theorem 2.9.2 (b), we can handle general case by performing induction. Therefore without loss of generality, we assume X is indexed by $(\{1\},...,\{k\})$ and X' is indexed by $(\{1\},...,\{k+1\})$. Just as the last case, without loss of generality we assume that X' is obtained by replicating X over 1 new axis, denoted as $\{k+1\}$. In other words, $\text{ax}(X') = \text{ax}(X) \cup \{\{k+1\}\}$.

   To control $\|X'\|_{\text{pn}}$, we need to bound $(\frac{1}{\sqrt{n}})^{|\gamma|}\|X'_{\gamma}\|$ where $\gamma \in \Gamma_{k+1}$. Since X' is obtained from X by replicating it over $\{k+1\}$, $(\frac{1}{\sqrt{n}})^{|\gamma|}\|X'_{\gamma}\| = (\frac{1}{\sqrt{n}})^{|\beta|}\|X_{\beta}\|$ where $\beta = \gamma|_{[k]}$. As $\|X\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(k)}$, it implies that $(\frac{1}{\sqrt{n}})^{|\gamma|}\|X'_{\gamma}\| \leqslant \varepsilon$ holds for any $\gamma \in \Gamma_{k'}$. Therefore we conclude that $\|X'\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(k')}$.

4. To bound $\|X\|_{\text{pn}}$, we need to bound the normalized norm of any slice of X. Let X_{β} be arbitrarily slice of X where $\beta \in \Gamma_k$. Since γ and β are partitions of $[k]$, there exist partitions that are finer than both β and γ, where the notion of finer between two partitions is defined in Definition 10. Among all partitions that satisfy such conditions, denote the most coarse one as $\alpha \in \Gamma_k$. This can be done because the Γ_k is finite. Note that $|\alpha| < |\beta|$ and $|\alpha| < |\gamma|$.

   Since X_{α} is a slice of X_{γ} and $\|X_{\gamma}\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(k')}$, $(\frac{1}{\sqrt{n}})^{|\alpha|}\|X_{\alpha}\| \leqslant \varepsilon$ according to Theorem 2.9.2 (a). As X_{α} is the slice of X_{β} (implies $\|X_{\alpha} \leqslant X_{\beta}\|$) and α is the most coarse partition that is finer than β and γ (implies $\|X_{\alpha}\| \geqslant \|X_{\beta}\|$ we have $\|X_{\beta}\| = \|X_{\alpha}\|$. This implies $(\frac{1}{\sqrt{n}})^{|\beta|}\|X_{\beta}\| \leqslant (\frac{1}{\sqrt{n}})^{|\alpha|}\|X_{\alpha}\| \leqslant \varepsilon$.

   As $(\frac{1}{\sqrt{n}})^{k'}\|X_{\beta}\| \leqslant \varepsilon$ holds for arbitrary slice β of X, we conclude that $\|X\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(k)}$.

□

Now we are ready to prove the main theorem.

Theorem 2.3.2 (Stability of LE layers for k-IGN)**.** *Let $T_\gamma : \mathbb{R}^{[0,1]^\ell} \to \mathbb{R}^{[0,1]^m}$ be a basis element of the space of* $\mathrm{LE}_{\ell,m}$ *maps where $\gamma \in \Gamma_{\ell+m}$. If $\|X\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(\ell)}$, then the partition-norm of $Y := T_\gamma(X)$ satisfies $\|Y\|_{pn} \leqslant \varepsilon \mathbf{1}_{\mathrm{bell}(m)}$ for all $\gamma \in \Gamma_{\ell+m}$.*

Proof. Without loss of generality, we first consider discrete cases of mapping from $X \in \mathbb{R}^{n^\ell}$ to $Y \in \mathbb{R}^{n^m}$. In general, each element T_γ of linear permutation equivariant basis can be identified with the following operation on input/output tensors.

> Given input X, (step 1) obtain its subtensor X_γ on a certain Π_1 (selection axis), (step 2) average X_γ over Π_2 (reduction axis), resulting in $X_{\gamma,\mathrm{reduction}}$. (step 3) Align $X_{\gamma,\mathrm{reduction}}$ on Π_3 (alignment axis) with Y_γ and (step 4) replicate Y_γ along Π_4 (replication axis), resulting $Y_{\gamma,\mathrm{replication}}$, a slice of Y. Entries of Y outside $Y_{\gamma,\mathrm{replication}}$ will be set to be 0. In general, Π_i can be read off from S_1-S_3.

Π_1-Π_4 corresponds to different axis of input/output tensor and can be read off from different parts of $S_\gamma = S_1 \cup S_2 \cup S_3$ as we introduced in the main text. Note such operation can be naturally extended to the continuous case, as done in Tables 2.1 to 2.3 for 2-IGN. We next give detailed explanations of each step.

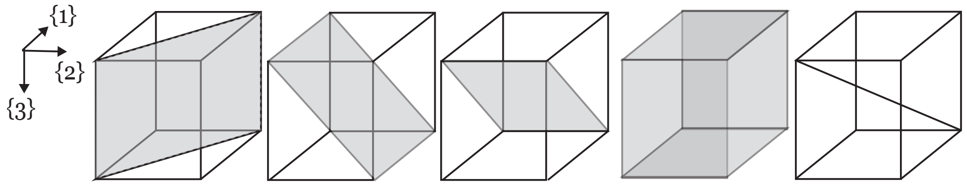


Figure 2.5. Five "slices" of a 3-tensor, corresponding to bell(3) = 5 partitions of [3]. From left to right: a) $\{\{1,2\},\{3\}\}$ b) $\{\{1\},\{2,3\}\}$ c) $\{\{1,3\},\{2\}\}$ d) $\{\{1\},\{2\},\{3\}\}$ e) $\{\{1,2,3\}\}$.

First step ($X \to X_\gamma$): select X_γ from X via Π_1.

Π_1 corresponds to

$$S|_{[\ell]} = \{s \cap [l] \mid s \in S \text{ and } s \cap [l] \neq \emptyset\}.$$

It specifies the what parts (such as diagonal part for 2-tensor) of the input ℓ-tensor is under consideration. We denote the resulting subtensor as X_γ. See Definition 6 for formal definition.

As an example in Equation (2.3), Π_1 corresponds to $\{\{1,2\},\{3\}\}$, meaning we select a 2-tensor with axises $\{1,2\}$ and $\{3\}$. Note that the cardinality $|S|_{[\ell]}| = |(S_1 \cup S_2)|_{[\ell]}| \leqslant l$ encodes the order of X_γ.

Second step ($X_\gamma \to X_{\gamma,\mathbf{reduction}}$): average of X_γ over Π_2. Π_2 corresponds axes in $S_1 \subset S|_{[\ell]}$, which tells us along what axis to average over X_γ. It will reduce the tensor X_γ of order $|S_1| + |S_2|$, indexed by $S|_{[\ell]}$, to a tensor of order $|S|_{[\ell]}| - |S_1| = |S_2|$, indexed by $S_2|_{[l]}$. Recall the definition of "averaging" in Definition 11.

In the example of Figure 2.6, this corresponds to averaging over axis $\{\{1,2\}\}$, reducing 2-tensor (indexed by axis $\{1,2\}$ and $\{3\}$) to 1-tensor (indexed by axis $\{3\}$). The normalization factor in the discrete case is $n^{|S_1|}$. We denote the tensor after reduction as $X_{\gamma,\text{reduction}}$.

As the second step performs tensor order reduction, we end up with a tensor $X_{\gamma,\text{reduction}}$ of order $|S_2|$. The next two steps will describe how to fill in the output tensor Y using $X_{\gamma,\text{reduction}}$. To fill in Y, we will first align $X_{\gamma,\text{reduction}}$ with Y_γ, a subtensor of Y, in the third step. We then replicate Y_γ on Π_4 in the fourth step, resulting in $Y_{\gamma,\text{replication}}$, a sub-tensor of Y. Finally, we fill all entries of Y outside the subtensor Y_γ to be zero.

Third step ($X_{\gamma,\mathbf{reduction}} \to Y_\gamma$): align $X_{\gamma,\mathbf{reduction}}$ with Y_γ. To fill in Y_γ, we need to specify how the resulting $|S_2|$-tensor $X_{\gamma,\text{reduction}}$ is *aligned* with a certain $|S_2|$-subtensor Y_γ of Y. After all, there are many ways of selecting a $|S_2|$-tensor from Y, which is indexed by $\{\{l+1\},...,\{\ell+m\}\}$. Specifically, set Y_γ be the $|S_2|$-tensor indexed by $S_2|_{\ell+[m]}$. We next define the precise relationship between $X_{\gamma,\text{reduction}}$ and Y_γ. $X_{\gamma,\text{reduction}}$ is indexed by $S_2|_{[l]}$ while Y_γ is indexed by $S_2|_{l+[m]}$ and defined to be $Y_\gamma(\cdot) = X_{\gamma,\text{reduction}}(\cdot)$. In the example of Figure 2.6, $X_{\gamma,\text{reduction}}$ is a 1D tensor indexed by $\{3\}$ and Y_γ (the grey cuboid on the right cube of Figure 2.6) is indexed by $\{6\}$.

Fourth step ($Y_\gamma \to Y_{\gamma,\mathbf{replication}}$): replicating Y_γ over Π_4. Π_4 corresponds to axes in S_3. It will be used to specify along what axis (axes) we will replicate the $|S_2|$-tensor Y_γ over. Recall that Y_γ is indexed by $S_2|_{l+[m]}$. Let $Y_{\gamma,\text{replication}}$ be a subtensor of $Y \in \mathbb{R}^{n^l}$ indexed by $(S_2 \cup S_3)|_{l+[m]}$. Obviously, the tensor Y_γ output from the Third step is a subtensor of $Y_{\gamma,\text{replication}}$. Without loss of generality, let the first $|S_2|$ component are indexed by $S_2|_{l+[m]}$ and the rest components are indexed by $S_3|_{l+[m]}$. The mathematical definition of the fourth step is then $Y_{\gamma,\text{replication}}(\cdot,t) := Y_\gamma(\cdot)$ for all $t \in [n]^{|S_3|}$. Note that the order of $Y_{\gamma,\text{replication}}$ can be smaller than order of Y.

The example in Equation (2.3) has $S_3 = \{\{4\},\{5\}\}$, which means that we will replicate the 1-tensor along axis $\{4\}$ and $\{5\}$. Note that in general, we do not have to fill in the whole m-tensor (think about copy row average to diagonal in Table 2.1).

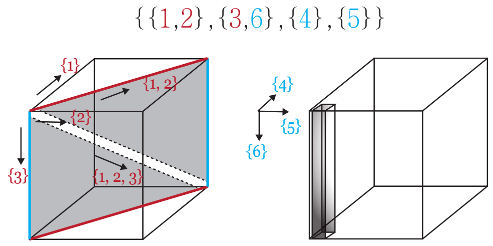


Figure 2.6. An illustration of the one basis element of the space of $\text{LE}_{3,3}$. It selects area spanned by axis $\{1,2\}$ and $\{3\}$, average over the axis $\{1,2\}$, and then align the resulting 1D tensor with axis $\{6\}$, and finally replicate the slices along axis $\{4\}$ and $\{5\}$ to fill in the whole cube on the right.

After the interpretation of general linear equivariant maps in k-IGN, We now show that if $\|X\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(\ell)}$, then $T_\gamma(X) \leqslant \varepsilon \mathbf{1}_{\text{bell}(m)}$ holds for all γ. This can be done easily with the use of Theorem 2.9.2.

For any partition of $[\ell+m]$ γ, according to the first step we are mainly concerned about the $\|X_\gamma\|_{\text{pn}}$ instead of $\|X\|_{\text{pn}}$. Since X_γ is a slice of X, then if $\|X\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(\text{ord}(X))}$, by Theorem 2.9.2 (a), then $\|X_\gamma\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(|S_1|+|S_2|)}$.

According to the second step and Theorem 2.9.2 (b), we can also conclude that $\|X_{\gamma,\text{reduction}}\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(|S_2|)}$.

For the third step of align $X_{\gamma,\text{reduction}}$ with Y_γ, it is quite obvious that $\|Y_\gamma\|_{\text{pn}} = \|X_{\gamma,\text{reduction}}\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(|S_2|)}$.

For the fourth step of replicating Y_γ over Π_4 to get $Y_{\gamma,\text{replication}}$, by Theorem 2.9.2 (c), we have $\|Y_{\gamma,\text{replication}}\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(|S_2|+|S_3|)}$.

Lastly, we evoke Theorem 2.9.2 (d) to get $\|Y\|_{\text{pn}} \leqslant \varepsilon \mathbf{1}_{\text{bell}(m)}$, which concludes our proof.

$\square$

Remark 4 (On the difference from Incidence Networks for Geometric Deep Learning.)**.** *A recent preprint Incidence Networks for Geometric Deep Learning [1] characterize the linear equivariant maps between incidence tensor, which encodes the combinatorial structure of graphs and its higher order analog simplicial complex and polytopes. [1] characterizes the linear permutation equivariant maps in terms of pooling and broadcasting operations. The pooling and broadcasting operations is the same as the averaging and replication operation defined in Definition 11 and Definition 9.*

The main difference of [1] from this chapter is 1) their motivation is to characterize the linear permutation equivariant maps between incidence tensors while in this chapter, the similar characterization (in the case of linear permutation equivariant maps of k-IGN) serves as a building block for our convergence proof; 2) the characterization in [1] is slightly more general as incidence tensor can have different length for different axis while tensors considered in our case has the same length across all axis.

2.9.2 Missing Proofs from Section 2.4 (Edge Weight Continuous Model)

First we need a lemma on the distribution of gaps between n uniform sampled points on $[0,1]$.

Lemma 2.9.3. *Let $u_{(i)}$ be n points uniformly sampled on $[0,1]$, sorted from small to large with $u_{(0)} = 0$ and $u_{(n+1)} = 1$. Let $D_i = u_{(i)} - u_{(i-1)}$. All D_is have same distribution, which is* $\mathrm{Beta}(1,n)$*. In particular, expectation of D_i $\mathbb{E}(D_i) = \frac{1}{n+1}$, $\mathbb{E}(D_i^2) = \frac{2}{(n+1)(n+2)}$, $\mathbb{E}(D_i^3) = \frac{6}{(n+1)(n+2)(n+3)}$.*

Proof. By a symmetry argument, it is easy to see that all intervals follow the same distribution. For the first interval, the probability all the n points are above x is $(1-x)^n$ so the density of the length of the first (and so each) interval is $n(1-x)^{n-1}$. This is a Beta distribution with parameters $\alpha = 1$ and $\beta = n$ The expectation of higher moments follows easily. Note that although the intervals are identically distributed, they are not independently distributed, since their sum is 1. □

Lemma 2.9.4. *Let $X \in \mathbb{R}^{[0,1]\times d}$ be an A_3-Lipschitz graphon signal satisfying AS3, and let $\widetilde{X_n}$ and X_n be the induced graphon signal as in Eqs.* (2.4) *and* (2.5). *Then we have i) $\|X - X_n\|_{pn}$ converges to 0 and ii) $\|X - \widetilde{X_n}\|_{pn}$ converges to 0 in probability.*

Proof. We first bound the $\|X - X_n\|_{L2[0,1]}$ and $\|X - \widetilde{X_n}\|_{L2[0,1]}$. For the first case, partitioning the unit interval as $I_i = [(i-1)/n, i/n]$ for $1 \leqslant i \leqslant n$ (the same partition used to obtain x_n, and thus X_n, from X), we can use the Lipschitz property of X to derive

$$\|X - X_n\|^2_{L_2(I_i)} \leqslant A_3^2 \int_0^{1/n} u^2 du = \frac{A_3^2}{3n^3}$$

We can then write $\|X - X_n\|^2_{L_2([0,1])} = \sum_i \|X - X_n\|^2_{L_2(I_i)} \leqslant \frac{A_3}{3n^2}$, which implies that $\|X - X_n\|_{L_2([0,1])} \leqslant \sqrt{\frac{A_3}{3n^2}}$.

For the second case, since $\|X - \widetilde{X_n}\|^2_{L_2([0,1])} = \sum_i \|X - \widetilde{X_n}\|^2_{L_2(I_i)}$, we will bound the $\left\|X - \widetilde{X_n}\right\|^2_{L_2(I_i)}$. As

$$\left\|X - \widetilde{X_n}\right\|^2_{L_2(I_i)} \leqslant A_3^2 \int_0^{D_i} u^2 du = A_3 D_i^3/3$$

therefore

$$\left\|X - \widetilde{X_n}\right\|^2_{L_2(I)} = \sum_i \left\|X - \widetilde{X_n}\right\|^2_{L_2(I_i)} \leqslant A_3/3 \sum_i D_i^3$$

where D_i stands for the length of I_i, which is a random variable due to the random sampling.

According to Theorem 2.9.3, all D_i are identically distributed and follows the Beta distribution $B(1, n-1)$. The expectation $E(D_i^3) = \frac{6}{n(n+1)(n+2)}$. Since by Jensen's inequality $E(\sqrt{Y}) \leqslant \sqrt{E(Y)}$ holds for any positive random variable Y, $E(\sqrt{\frac{A_3}{3}\sum_i D_i^3}) \leqslant \sqrt{E(\frac{A_3}{3}\sum_i D_i^3)} = \sqrt{\frac{A_3}{3}\frac{1}{n(n+2)}} = \Theta(\frac{1}{n})$. Using Markov inequality, we can then upper bound the

$$P(\|X - \widetilde{X_n}\|_{L_2(I)} \geq \varepsilon) \leqslant P(\sqrt{\frac{A_3}{3}\sum_i D_i^3} \geq \varepsilon) \leq \frac{E(\sqrt{\frac{A_3}{3}\sum_i D_i^3})}{\varepsilon} = \Theta(\frac{1}{n\varepsilon}) \tag{2.11}$$

Since the $P(\|X-\widetilde{X_n}\|_{L_2(I)} \geq \varepsilon)$ goes to 0 as n increases, we conclude that $\|X-\widetilde{X_n}\|_{\mathrm{pn}}$ converges to 0 in probability. $\square$

Lemma 2.9.5. *If W satisfies AS1, $\|W-W_n\|_{\mathrm{pn}}$ converges to 0. $\|W-\widetilde{W_n}\|_{\mathrm{pn}}$ converges to 0 in probability.*

Proof. For the first case, partitioning the unit interval as $I_i = [(i-1)/n, i/n]$ for $1 \leqslant i \leqslant n$, we can use the graphon's Lipschitz property to derive

$$\|W-W_n\|_{L_1(I_i\times I_j)} \leqslant A_1 \int_0^{1/n}\int_0^{1/n} |u|dudv + A_1 \int_0^{1/n}\int_0^{1/n} |v|dvdu = \frac{A_1}{2n^3} + \frac{A_1}{2n^3} = \frac{A_1}{n^3}.$$

We can then write $\|W-W_n\|_{L_1([0,1]^2)} = \sum_{i,j} \|W-W_n\|_{L_1(I_i\times I_j)} \leqslant n^2\frac{A_1}{n^3} = \frac{A_1}{n}$ which, since $W-W_n : [0,1]^2 \to [-1,1]$, implies $\|W-W_n\|_{L_2([0,1]^2)} \leqslant \sqrt{\|W-W_n\|_{L_1([0,1]^2)}} \leqslant \sqrt{\frac{A_1}{n}}$. The second last inequality holds because all entries of $W-W_n$ lies in $[-1,1]$.

Similarly, $\|\mathrm{Diag}(W-W_n)\|_{L_2[0,1]} \leqslant \sqrt{\|\mathrm{Diag}(W-W_n)\|_{L_1[0,1]}} \leqslant \sqrt{2nA_1\int_0^{1/n} udu} = \sqrt{\frac{A_1}{n}}$. Therefore we conclude the first part of the proof.

For the second case, diagonal norm is similar to the proof of Theorem 2.4.2 so we only focus on the $\|W-W_n\|_{L_2([0,1]^2)}$. Since $W-\widetilde{W_n} : [0,1]^2 \to [-1,1]$ implies

$$\|W-\widetilde{W_n}\|_{L_2([0,1]^2)} \leqslant \sqrt{\|W-\widetilde{W_n}\|_{L_1([0,1]^2)}} = \sqrt{\sum_{i,j} \|W-\widetilde{W_n}\|_{L_1(I_i\times I_j)}}$$

where

$$\|W-\widetilde{W_n}\|_{L_1(I_i\times I_j)} \leqslant A_1 \int_{I_v}\int_{I_u} |u|dudv + A_1 \int_{I_u}\int_{I_v} |v|dvdu = \frac{A_1}{2}(D_iD_j^2 + D_jD_i^2)$$

Therefore

$$\|W-\widetilde{W_n}\|_{L_2([0,1]^2)} \leqslant \sqrt{\|W-\widetilde{W_n}\|_{L_1([0,1]^2)}} = \sqrt{\sum_{i,j} \frac{A_1}{2}(D_jD_i^2 + D_iD_j^2)} = \sqrt{A_1 \sum_i D_i^2} \quad (2.12)$$

where we use the $\sum_i D_i = 1$ for the last equality. Since by Jensen's inequality $E(\sqrt{Y}) \leqslant \sqrt{E(Y)}$ for any positive random variable Y, $E(\sqrt{\sum_i D_i^2}) \leqslant \sqrt{E(\sum_i D_i^2)} = \Theta(\frac{1}{\sqrt{n}})$ since $E(D_i^2) = \Theta(\frac{1}{n^2})$ by Theorem 2.9.3. By Markov inequality, we then bound

$$P(\|W - \widetilde{W_n}\|_{L_2([0,1]^2)} > \varepsilon) \leq P(\sqrt{\|W - \widetilde{W_n}\|_{L_1([0,1]^2)}} > \varepsilon) \leqslant \frac{E(\sqrt{\sum_i D_i^2})}{\varepsilon} \leq \Theta(\frac{1}{\sqrt{n}\varepsilon})$$

□

Therefore, we conclude that both $\|W - W_n\|_{\text{pn}}$ and $\|W - \widetilde{W_n}\|_{\text{pn}}$ converges to 0.

Proposition 2.9.6 (Stability of Φ_c). *If cIGN $\Phi_c : \mathbb{R}^{[0,1]^2 \times d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ satisfy AS2, AS4 and $\|W_1 - W_2\|_{pn} \leqslant \varepsilon \mathbf{1}_2$, then $\|\Phi_c(W_1) - \Phi_c(W_2)\|_{pn} = \|\Phi_c(W_1) - \Phi_c(W_2)\|_{L_2} \leqslant C(A_2)\varepsilon$. The same statement still holds if we change the underlying norm of Partition-norm from L_2 to L_∞.*

Proof. Without loss of generality, it suffices to prove for 2-IGN as k-IGN follows the same proof with the constant being slightly different. Since we have proved stability of every linear layers of IGN in Theorem 2.3.2, the general linear layer T is just a linear combinations of individual linear basis, i.e. $T = \sum_\gamma c_\gamma T_\gamma$ where $c_i \leqslant A_2$ for all i according to AS2. Without loss of generality, We can assume $T(X)$ is of order 2 and have

$$\begin{aligned}
\|T(W_1) - T(W_2)\|_{\text{pn}} &= \|\sum_i c_\gamma T_\gamma (W_1 - W_2)\|_{\text{pn}} \\
&\leqslant \sum_i \|c_\gamma T_\gamma (W_1 - W_2)\|_{\text{pn}} \\
&\leqslant (\sum |c_\gamma|\varepsilon, \sum |c_\gamma|\varepsilon) = (15A_2\varepsilon, 15A_2\varepsilon)
\end{aligned}$$

To extend the result to nonlinear layer, note that AS4 ensures the 2-norm shrinks after passing through nonlinear layers. Therefore $\|\sigma \circ T(X) - \sigma \circ T(Y)\|_{\text{pn}} \leqslant \|T(X) - T(Y)\|_{\text{pn}} = \|T(X - Y)\|_{\text{pn}} \leqslant 15A_2\|X - Y\|_{\text{pn}}$. Repeating such process across layers, we finish the proof of the L_2 case.

The extension to L_∞ is similar to the case of L_2 norm. The main modification is to change the definition of the partition-norm from L_2 norm on different slices (corresponding to different partitions of $[\ell]$ where ℓ is the order of input) to L_∞ norm. The extension to the case where input and output tensor is of order ℓ and m is also straightforward according to Theorem 2.3.2.

$\square$

Theorem 2.4.4 (Convergence of cIGN in the edge weight continuous model)**.** *Under the fixed sampling condition, IGN converges to cIGN, i.e.,* $\|\Phi_c([W, Diag(X)]) - \Phi_c([W_n, Diag(X_n)])\|_{L_2}$ *converges to 0.*

An analogous statement hold for the random sampling setting, where $\|\Phi_c([W, Diag(X)]) - \Phi_c([\widetilde{W_n}, Diag(\widetilde{X_n})])\|_{L_2}$ *converges to 0 in probability.*

Proof. By Theorem 2.9.6, it suffices to prove that $\|[W, \mathrm{Diag}(X)]) - [W_n, \mathrm{Diag}(X_n)]\|_{\mathrm{pn}}$ and $\|[W, \mathrm{Diag}(X)]) - [\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\|_{\mathrm{pn}}$ goes to 0.

$\|[W, \mathrm{Diag}(X)]) - [W_n, \mathrm{Diag}(X_n)]\|_{\mathrm{pn}}$ is upper bounded by $(\Theta(\frac{1}{n^{1.5}}), \Theta(\frac{1}{n^{1.5}}))$ according to Theorems 2.4.2 and 2.4.3, which decrease to 0 as n increases. Therefore we finish the proof of convergence for the deterministic case.

For the random sampling case, by Theorems 2.4.2 and 2.4.3, we know that both $\|W - \widetilde{W_n}\|_{L_2([0,1]^2)}$ and $\|X - \widetilde{X_n}\|_{L_2(I)}$ goes to 0 as n increases in probability at the rate of $\Theta(\frac{1}{n^{1.5}})$. Therefore we can also conclude that the convergence of IGN in probability according to Theorem 2.9.6. $\square$

2.9.3 Missing Proof from Section 2.5 (Edge Probability Continuous Model)

Missing Proof for Section 2.5.2

Theorem 2.5.1. *Given any graphon W with $c_{max} < 1$ and an IGN architecture (fix hyper-parameters like number of layers), there exists a set of parameters θ such that convergence of IGN_θ to $cIGN_\theta$ is not possible, i.e.,* $\mathrm{RMSE}_U(\Phi_c([W, Diag(X)]), \Phi_d([A_n, Diag(\widetilde{x_n})]))$ *does* *not converge to 0 as $n \to \infty$, where A_n is 0-1 matrix generated according to Eq.* (2.7)*, i.e., $A_n[i][j] = a_{i,j}$.*

Proof. Given a fixed IGN architecture Φ_c that maps input $\mathbb{R}^{n^2 \times d_1}$ to $\mathbb{R}^{n^k \times d_2}$, it suffices to show the case of $k = 1$ and $d_2 = 1$. Under the case of $k = 1$ and $d_2 = 1$, it suffice to show that single layer IGN may not converge. Let $IGN = \sigma \circ L^{(1)}$ have only one linear layer, and let the input to IGN be A in the discrete case and W in the continuous case. For simplicity, we assume that graphon W is constant p on $[0,1]^2$. As A consists of only 0 and 1 and all entries of W is below $c_{\max}$, we can set weights of IGN such that its first linear layer consists of only identity map and bias term. By choosing bias term to be any number between $[-1, -c_{\max}]$, $L^{(1)}$ map any number no large than $c_{\max}$ to negative and maps 1 to positive.

Therefore $L^{(1)}(W) = 0$ and $L^{(1)}(A)$ is a positive number $c \in \mathbb{R}^+$ on entries (i,j) where $A(i,j) = 1$. Let σ be ReLU and $L^{(2)}$ be average of all entries. We can see that c$IGN(W) = 0$ for all n while $IGN(A)$ converges to $\sigma(c)p$ as n increases.

As the construction above only relies on the fact that there is a separation between $c_{\max}$ and 1 (but not on size n), it can be extended to deeper IGNs , which means the gap between c$IGN(W)$ and $IGN(A)$ will not decrease as n increases. In the general case of W not being constant, the only difference is that $IGN(A)$ will converge to be $\sigma(c)p^*$ where p^* is a different constant that depends on W. Therefore we conclude the proof. □

Remark 5. *The reason that the same argument does not work for spectral GNN is that spectral GNN always maintains Ax in the intermediate layer. In contrast, IGN keeps both A and Diag(x) in separate channels, which makes it easy to isolate them to construct counterexamples.*

Missing Proofs from Section 2.5.3

Notation. For any $P, Q \in \mathbb{R}^{n \times n}$, define $d_{2,\infty}$, the normalized $2,\infty$ matrix norm, by $d_{2,\infty}(P,Q) = n^{-1/2}\|P - Q\|_{2,\infty} := \max_i n^{-1/2}\|P_{i,\cdot} - Q_{i,\cdot}\|_2$ where $P_{i,\cdot}, Q_{i,\cdot}$ are i-th row of P and Q, respectively. Note that $d_{2,\infty}(P,Q) \geq \frac{1}{n}\|P - Q\|_2$.

Let S_U be the sampling operator for W, i.e., $S_U(W) = \frac{1}{n}[W(U_i, U_j)]_{n\times n}$. Note that as U is randomly sampled, S_U is a random operator. Denote S_n as sampling on a fixed equally spaced grid of size $n \times n$, i.e. $S_nW = \frac{1}{n}[W(\frac{i}{n}, \frac{j}{n})]_{n\times n}$. S_n is a fixed operator when n is fixed.

Let $\widehat{W}_{n\times n}$ be the estimated edge probability from graphs A sampled from W. Let $\widetilde{W_n}$ be the piece-wise constant graphon induced from sample U as Eq. (2.5). Similarly, denote $W_{n\times n}$ be the $n\times n$ matrix realized on sample U, i.e., $W_{n\times n}[i, j] = W(u_i, u_j)$. It is easy to see that $S_U(W) = \frac{1}{n}W_{n\times n}$. Let $\widetilde{W_{n,E}}$ be the graphon induced by $W_{n\times n}$ with $n\times n$ blocks of the same size. In particular, $\widetilde{W_{n,E}}(I_i \times I_j) = W(u_{(i)}, u_{(j)})$ where $I_i = [\frac{i-1}{n}, \frac{i}{n}]$. E in the subscript is the shorthand for the "blocks of equal size". Similarly we can also define the 1D analog of $\widetilde{W_n}$ and $\widetilde{W_{n,E}}$, $\widetilde{X_n}$ and $\widetilde{X_{n,E}}$.

Proof strategy. We first state five lemmas that will be used in the proof of Theorem 2.5.2. Theorem 2.9.7 concerns the property of normalized sampling operator S_U and S_n. Theorems 2.9.8 and 2.9.9 concern the convergence of $\|\widetilde{W_n} - W\|_{L_\infty}$ and $\|\widetilde{W_{n,E}} - W\|_{L_\infty}$. Theorem 2.9.10 characterize the effects of linear equivariant layers T and IGN Φ on L_∞ norm of the input and output. Theorem 2.9.11 bounds the L_∞ norm of the difference of stochastic sampling operator S_U and the deterministic sampling operator S_n. Theorem 2.5.2 is built on the results from five lemmas and the existing result on the theoretical guarantee of edge probability estimation from [165].

The convergence some lemmas states is almost surely convergence. Convergence almost surely implies convergence in probability, and in this chapter, all theorems concern convergence in probability. Note that proofs of Theorems 2.9.7 to 2.9.9 and 2.9.11 for the W and X are almost the same. Therefore without loss of generality, we mainly prove the case of W.

Definition 12 (Chessboard pattern)**.** *Let $u_i = \frac{i-1}{n}$ for all $i \in [n]$. A graphon W is defined to have chessboard pattern if and only if there exists a n such that W is a piecewise constant on $[u_i, u_{i+1}] \times [u_j, u_{j+1}]$ for all $i, j \in [n]$. Similarly, $f : [0, 1] \to \mathbb{R}$ has 1D chessboard pattern if there exists n such that f is a piecewise constant on $[u_i, u_{i+1}]$ for all $i \in [n]$.*

See Figure 2.7 for examples and counterexamples.

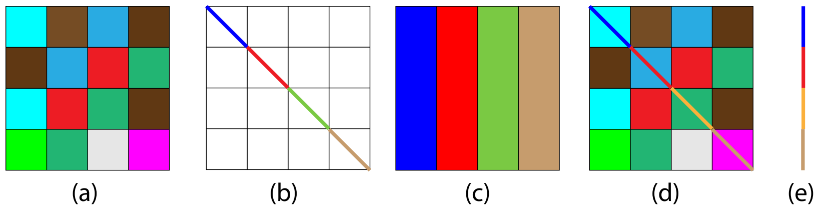


Figure 2.7. (a) and (c) has chessboard pattern. (e) has 1D chessboard pattern. (d) does not has the chessboard pattern. (b) is of form $\mathrm{Diag}(\widetilde{f_{n,E}})$ and also does not have chessboard pattern, but in the case of IGN approximating Spectral GNN, (b) is represented in the form of c).

Lemma 2.9.7 (Property of S_n and S_U)**.** *We list some properties of sampling operator S_U and S_n*

1. *$S_U \circ \sigma = \sigma \circ S_U$. Similar result holds for S_n as well.*
2. *$\|S_U f_{1d}\| \leqslant \|f_{1d}\|_{L_\infty}$*

   where $f_{1d} : [0,1] \to \mathbb{R}$. Similar result holds for $f_{2d} : [0,1]^2 \to \mathbb{R}$ and S_n as well.

Lemma 2.9.8. *Let W be $[0,1]^2 \to \mathbb{R}$ and X be $[0,1] \to \mathbb{R}$. If W is Lipschitz, $\|\widetilde{W_n} - W\|_{L_\infty}$ converges to 0 in probability. If X is Lipschitz, $\|\widetilde{X_n} - X\|_{L_\infty}$ converges to 0 in probability.*

Proof. Without loss of generality, we only prove the case for W. By the Lipschitz condition of W, if suffices to bound the $Z_n = \max_{i=1}^n D_i$ where D_i is the length of i-th interval $|u_{(i)} - u_{(i-1)}|$. Characterizing the distribution of the length of largest interval is a well studied problem [121, 118, 65]. It can be shown that Z_n follows $P(Z_n \leqslant x) = \sum_{j=0}^{n+1} \begin{pmatrix} n+1 \\ j \end{pmatrix} (-1)^j (1 - jx)_+^n$ with the expectation $E(Z_k) = \frac{1}{n+1} \sum_{i=1}^{n+1} \frac{1}{i} = \Theta(\frac{\log n}{n})$. By Markov inequality, we conclude that $\|\widetilde{W_n} - W\|_{L_\infty}$ converges to 0 in probability.

□

Lemma 2.9.9. *Let W be $[0,1]^2 \to \mathbb{R}$ and X be $[0,1] \to \mathbb{R}$. If W is Lipschitz, $\|\widetilde{W_{n,E}} - W\|_{L_\infty}$ converges to 0 almost surely. If X is Lipschitz, $\|\widetilde{X_{n,E}} - X\|_{L_\infty}$ converges to 0 almost surely.*

Proof. As $\widetilde{W_{n,E}}$ is a piecewise constant graphon and W is Lipschitz, we only need to examine $\max_{i,j} \|(W - \widetilde{W_{n,E}})(\frac{i}{n}, \frac{j}{n})\|$.

It is easy to see that $(W-\widetilde{W_{n,E}})(\frac{i}{n},\frac{j}{n}) = W(\frac{i}{n},\frac{j}{n}) - W(u_{(i)},u_{(j)})$ where $u_{(i)}$ stands for the i-th smallest random variable from uniform i.i.d. samples from $[0,1]$. By the Lipschitz condition of W, if suffices to bound $\|\frac{i}{n}-u_{(i)}\|+\|\frac{j}{n}-u_{(j)}\|$. Glivenko-Cantelli theorem tells us that the L_∞ of empirical distribution F_n and cumulative distribution function F converges to 0 almost surely, i.e., $\sup_{u\in[0,1]}|F(u)-F_n(u)| \to 0$ almost surely. Since $\max_i \|u_{(i)}-\frac{i}{n}\| = \sup_{u\in\{u_{(1)},\ldots,u_{(n)}\}}|F(u)-F_n(u)| \leqslant \sup_{u\in[0,1]}|F(u)-F_n(u)|$ when $F(u)=u$ (cdf of uniform distribution), we conclude that $\|\widetilde{W_{n,E}}-W\|_{L_\infty}$ converges to 0 almost surely.

$\square$

We also need a lemma on the property of the linear equivariant layers T.

Lemma 2.9.10 (Property of T_c and σ)**.** *Let σ be nonlinear layer. Let T_c be a linear combination of elements of basis of the space of linear equivariant layers of cIGN, with coefficients upper bounded. We have the following property about T_c and σ*

1. *If W is Lipschitz, $T_c(W)$ is piecewise Lipschitz on diagonal and off-diagonal. Same statement holds for $\Phi_c(W)$.*
2. *$S_n\circ\sigma(\widetilde{W_{n,E}}) = \sigma\circ S_n(\widetilde{W_{n,E}})$.*

Proof. We prove two statements one by one.

1. We examine the linear equivariant operators from $\mathbb{R}^{[0,1]^2}$ to $\mathbb{R}^{[0,1]^2}$ in Table 2.1. There are some operations such as "average of rows replicated on diagonal" will destroy the Lipschitz condition of $T_c(W)$ but $T_c(W)$ will still be piecewise Lipschitz on diagonal and off-diagonal. Since σ will preserve the Lipschitzness, $\Phi_c(W)$ is piecewise Lipschitz on diagonal and off-diagonal.
2. This is easy to see as σ acts on input pointwise.

$\square$

Lemma 2.9.11. *Let W be $[0,1]^2 \to \mathbb{R}$*

1. *If W is Lipschitz, $\|S_U W - S_n W\|$ converges to 0 almost surely. Similarly, if X is Lipschitz, $\|S_U Diag(X) - S_n Diag(X)\|$ converges to 0 almost surely.*

2. *If W is piecewise Lipschitz on S_1 and S_2 where S_1 is the diagonal and S_2 is off-diagonal, then $\|S_U W - S_n W\|$ converges to 0 almost surely.*

Proof. Since the case of X is essentially the same with that of W, we only prove the case of W.

1. As $n\|S_U W - S_n W\|_\infty \geq \|S_U W - S_n W\|$, it suffices to prove that $n\|S_U W - S_n W\|_\infty = \max_{i,j}|W(u_{(i)}, u_{(j)}) - W(\frac{i}{n}, \frac{j}{n})|$ converges to 0 almost surely. Similar to Theorem 2.9.9, using Lipschitz condition of W and Glivenko-Cantelli theorem concludes the proof.

2. This statement is stronger than the one above. The proof of the last item can be adapted here. As W is A_1 Lipschitz on off-diagonal region and A_2 Lipschitz on diagonal,

$$\begin{aligned} n\|S_U W - S_n W\|_\infty &= \max_{i,j}\left|W(u_{(i)}, u_{(j)}) - W(\frac{i}{n}, \frac{j}{n})\right| \\ &= \max\left(\max_{i\neq j}\left|W(u_{(i)}, u_{(j)}) - W(\frac{i}{n}, \frac{j}{n})\right|, \max_{i=j}\left|W(u_{(i)}, u_{(j)}) - W(\frac{i}{n}, \frac{j}{n})\right|\right). \end{aligned}$$

   Using Lipschitz condition on diagonal and off-diagonal part of W and Glivenko-Cantelli theorem concludes the proof.

□

With all lemmas stated, we are ready to prove the main theorem.

Theorem 2.5.2 (convergence of IGN-small in the edge probability discrete model)**.** *Assume AS 1-4, and let $\widehat{W}_{n\times n}$ be the estimated edge probability that satisfies $\frac{1}{n}\|W_{n\times n} - \widehat{W}_{n\times n}\|_2$ converges to 0 in probability. Let Φ_c, Φ_d be continuous and discrete IGN-small. Then* $\text{RMSE}_U\left(\Phi_c\left([W, Diag(X)]\right), \Phi_d\left([\widehat{W}_{n\times n}, Diag(\widetilde{x_n})]\right)\right)$ *converges to 0 in probability.*

Proof. Using the triangle inequality

$$
\begin{aligned}
&\mathrm{RMSE}_U\left(\Phi_c\left([W,\mathrm{Diag}(X)]\right),\Phi_d\left([\widehat{W}_{n\times n},\mathrm{Diag}(\widetilde{x_n})]\right)\right)\\
&=\left\|S_U\Phi_c\left([W,\mathrm{Diag}(X)]\right)-\frac{1}{\sqrt{n}}\Phi_d\left([\widehat{W}_{n\times n},\mathrm{Diag}(\widetilde{x_n})]\right)\right\|\\
&=\|S_U\Phi_c\left([W,\mathrm{Diag}(X)]\right)-S_U\Phi_c\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)\\
&+S_U\Phi_c\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)-\Phi_d S_U\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{x_n})]\right)\\
&+\Phi_d S_U\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)-\frac{1}{\sqrt{n}}\Phi_d([\widehat{W}_{n\times n},\mathrm{Diag}(\widetilde{X_n})])\|\\
&\leqslant\underbrace{\left\|S_U\Phi_c\left([W,\mathrm{Diag}(X)]\right)-S_U\Phi_c\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)\right\|}_{\text{First term: discretization error}}\\
&+\underbrace{\left\|S_U\Phi_c\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)-\Phi_d S_U\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)\right\|}_{\text{Second term: sampling error}}\\
&+\underbrace{\left\|\Phi_d S_U\left([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]\right)-\frac{1}{\sqrt{n}}\Phi_d\left([\widehat{W}_{n\times n},\mathrm{Diag}(\widetilde{x_n})]\right)\right\|}_{\text{Third term: estimation error}}
\end{aligned}
\tag{2.13}
$$

The three terms measure the different sources of error. The first term is concerned with the discretization error. The second term concerns the sampling error from the randomness of U. This term will vanish if we consider only S_n instead of S_U for IGN-small. The third term concerns the edge probability estimation error.

For the first term, it is similar to the sketch in Section 2.5.3. $\|S_U\Phi_c([W,\mathrm{Diag}(X)])-S_U\Phi_c([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})])\|=\|S_U(\Phi_c([W,\mathrm{Diag}(X)])-\Phi_c([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})]))\|$, if suffices to upper bound $\|\Phi_c([W,\mathrm{Diag}(X)])-\Phi_c([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})])\|_{L_\infty}$ according to property of S_U in Theorem 2.9.7. Since $\|\Phi_c([W,\mathrm{Diag}(X)])-\Phi_c([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})])\|_{L_\infty}\leqslant C(\|W-\widetilde{W_n}\|_{L_\infty}+\|\mathrm{Diag}(X)-\mathrm{Diag}(\widetilde{X_n})\|_{L_\infty})$ by Theorem 2.9.6, and $\|W-\widetilde{W_n}\|_{L_\infty}$ converges to 0 in probability according to Theorem 2.9.8, we conclude that the first term will converges to 0 in probability.

For the third term $\|\Phi_d S_U([\widetilde{W_n},\mathrm{Diag}(\widetilde{X_n})])-\frac{1}{\sqrt{n}}\Phi_d([\widehat{W}_{n\times n},\mathrm{Diag}(\widetilde{x_n})])\|$

$= \| \frac{1}{\sqrt{n}}(\Phi_d([W_{n\times n}, \mathrm{Diag}(\widetilde{x_n})]) - \Phi_d([\widehat{W}_{n\times n}, \mathrm{Diag}(\widetilde{x_n})]))\| = \|\Phi_d([W_{n\times n}, \mathrm{Diag}(\widetilde{x_n})]) - \Phi_d([\widehat{W}_{n\times n}, \mathrm{Diag}(\widetilde{x_n})])\|_{\mathrm{pn}}$,

it suffices to control the $\|[W_{n\times n}, \mathrm{Diag}(\widetilde{x_n})] - [\widehat{W}_{n\times n}, \mathrm{Diag}(\widetilde{x_n})]\|_{\mathrm{pn}} = \frac{1}{n}\|W_{n\times n} - \widehat{W}_{n\times n}\|_2 \leqslant \|W_{n\times n} - \widehat{W}_{n\times n}\|_{2,\infty}$, which will also goes to 0 in probability as n increases according to the statistical guarantee of edge probability estimation of neighborhood smoothing algorithm [165], stated in Theorem 2.2.1. Therefore by Theorem 2.9.6, the third term also goes to 0 in probability.

Therefore the rest work is to control the second term $\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - \Phi_d S_U\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right)\|$. Again, we use the triangle inequality

$$\begin{aligned}
&\text{Second term}\\
&= \left\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - \Phi_d S_U\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right)\right\|\\
&\leqslant \left\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\| + \left\|S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right) - \Phi_d S_U\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right)\right\|\\
&= \left\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\| + \left\|S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right) - \Phi_d S_n([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})])\right\|\\
&= \left\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\|\\
&\leqslant \left\|S_U\Phi_c\left([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - S_U\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\| + \left\|S_U\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\|\\
&= \underbrace{\left\|S_U\left(\Phi_c([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]\right) - \Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\|}_{\text{term } a} + \underbrace{\left\|(S_U - S_n)\Phi_c\left([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})]\right)\right\|}_{\text{term } b}
\end{aligned}$$

The second equality holds because $S_U([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]) = S_n([\widetilde{W_{n,E}}, \widetilde{X_{n,E}}])$ by definition of $\widetilde{W_{n,E}}$ and IGN-small (See Remark 6 for more discussion). The third equality holds by the definition of IGN-small. We will bound the term a) $\|S_U(\Phi_c([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]) - \Phi_c([\widetilde{W_{n,E}}, \widetilde{X_{n,E}}]))\|$ and b) $\|(S_U - S_n)\Phi_c([\widetilde{W_{n,E}}, \mathrm{Diag}(\widetilde{X_{n,E}})])\|$ next.

For term a) $\|S_U(\Phi_c([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]) - \Phi_c([\widetilde{W_{n,E}}, \widetilde{X_{n,E}}]))\|$, if suffices to prove that $\|\Phi_c([\widetilde{W_n}, \mathrm{Diag}(\widetilde{X_n})]) - \Phi_c([\widetilde{W_{n,E}}, \widetilde{X_{n,E}}]))\|_{L_\infty}$ converges to 0 in probability. According to Theorem 2.9.6, it suffices to bound the $\|[\widetilde{W_n}, \widetilde{X_n}] - [\widetilde{W_{n,E}}, \widetilde{X_{n,E}}]\|_{L_\infty}$. Because $[\widetilde{W_n}, \widetilde{X_n}] - [\widetilde{W_{n,E}}, \widetilde{X_{n,E}}]\|_{L_\infty}$ $= \|\widetilde{W_n} - \widetilde{W_{n,E}}\|_{L_\infty} + \|\mathrm{Diag}(\widetilde{X_n}) - \mathrm{Diag}(\widetilde{X_{n,E}})\|_{L_\infty}) \leqslant \|\widetilde{W_n} - W\|_{L_\infty} + \|\widetilde{W_{n,E}} - W\|_{L_\infty} + \|\mathrm{Diag}(\widetilde{X_n}) - \mathrm{Diag}(X)\|_{L_\infty} + \|\mathrm{Diag}(\widetilde{X_{n,E}}) - \mathrm{Diag}(X)\|_{L_\infty}$, we only need to upper bound $\|\widetilde{W_n} - W\|_{L_\infty}$, $\|\widetilde{W_{n,E}} -$

$W\|_{L_\infty}$, $\|\text{Diag}(\widetilde{X_n}) - \text{Diag}(X)\|_{L_\infty})$ and $\|\text{Diag}(\widetilde{X_{n,E}}) - \text{Diag}(X)\|_{L_\infty})$, which are proved by Theorem 2.9.8 and Theorem 2.9.9 respectively.

For term b) $\|(S_U - S_n)\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right)\|$

$$\begin{aligned}
&\left\|(S_U - S_n)\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right)\right\| \\
&= \left\|(S_U\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right)\right\| \\
&\leqslant \left\|(S_U\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right) - S_U\Phi_c([W, \text{Diag}(X)])\right\| + \|S_U\Phi_c([W, \text{Diag}(X)]) - S_n\Phi_c([W, \text{Diag}(X)])\| \\
&+ \left\|S_n\Phi_c([W, \text{Diag}(X)]) - S_n\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right)\right\| \\
&= \left\|(S_U(\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right) - \Phi_c([W, \text{Diag}(X)]))\right\| + \|S_U\Phi_c([W, \text{Diag}(X)]) - S_n\Phi_c([W, \text{Diag}(X)])\| \\
&+ \left\|S_n(\Phi_c\left([\widetilde{W_{n,E}}, \text{Diag}(\widetilde{X_{n,E}})]\right) - \Phi_c([W, \text{Diag}(X)]))\right\|
\end{aligned}$$

For the first and last term, by the property of S_U, S_n and Φ_c, it suffices to bound $\|W - \widetilde{W_{n,E}}\|_{L_\infty}$ and $\|\text{Diag}(X) - \text{Diag}(\widetilde{X_{n,E}})\|_{L_\infty}$. Without loss of generality, We only prove the case for W. As $\|W - \widetilde{W_{n,E}}\|_{L_\infty}$ converges to 0 almost surely by Theorem 2.9.9, we conclude that the first and last term converges to 0 almost surely (therefore in probability). For the second term $\|S_U\Phi_c([W, \text{Diag}(X)]) - S_n\Phi_c([W, \text{Diag}(X)])\|$, $\Phi_c([W, \text{Diag}(X)])$ is piecewise Lipschitz on diagonal and off-diagonal according to Theorem 2.9.10 , and it converges to 0 almost surely according to the second part of Theorem 2.9.11.

As all terms converge to 0 in the probability or almost surely, we conclude that $\|S_U\Phi_c([W, \text{Diag}(X)]) - \Phi_d([\widehat{W}_{n\times n}, \text{Diag}(\widetilde{X_n})])\|$ converges to 0 in probability. □

Remark 6. *Note that we can not prove $S_n \cdot \Phi_c(\widetilde{W_{n,E}}) = \Phi_d \cdot S_n(\widetilde{W_{n,E}})$ in general. The difficulty is that starting with $\widetilde{W_{n,E}}$ of chessboard pattern, after the first layer, pattern like Figure 2.7(e) may appear in $\sigma \circ T_1(\widetilde{W_n})$. If T_2 is just a average/integral to map $\mathbb{R}^{n^2\times 1}$ to $\mathbb{R}$, then $S_n \circ T_2 \circ \sigma \circ T_1(\widetilde{W_n}) = T_2 \circ \sigma \circ T_1(\widetilde{W_n})$ will not be equal to $T_2 \circ \sigma \circ T_1(S_n\widetilde{W_n})$. The reason is that both $\sigma \circ T_1(\widetilde{W_n})$ and $\sigma \circ T_1(S_n\widetilde{W_n})$ will no longer be of chessboard pattern (Figure 2.7(e) may occur). The diagonal in the $\sigma \circ T_1(\widetilde{W_n})$ has no effect after taking integral in T_2 as it is of measure 0. On the other hand, the diagonal in the matrix $\sigma \circ T_1(S_n\widetilde{W_n})$ will affect the average. Therefore in* *general,* $S_n\Phi_c(\widetilde{W_{n,E}}) = \Phi_d S_n(\widetilde{W_{n,E}})$ *does not hold.*

2.9.4 IGN-small can Approximate Spectral GNN

Definition of Spectral GNN. The spectral GNN (SGNN) here stands for GNN with multiple layers of the following form $\forall j = 1,\dots d_{\ell+1}$,

$$z_j^{(\ell+1)} = \sigma\left(\sum_{i=1}^{d_\ell} h_{ij}^{(\ell)}(L)z_i^{(\ell)} + b_j^{(\ell)}1_n\right) \in \mathbb{R}^n \tag{2.14}$$

where $L = D(A)^{-\frac{1}{2}}AD(A)^{-\frac{1}{2}}$ stands for normalized adjacency,[4] $z_j^\ell, b_j^\ell \in \mathbb{R}$ denotes the embedding and bias at layer ℓ. d_ℓ stands for the number of output channels in ℓ-th layer. $h : \mathbb{R} \to \mathbb{R}, h(\lambda) = \sum_{k\geq 0}\beta_k\lambda^k, h(L) = \sum_k \beta_k L^k$, i.e., we apply h to the eigenvalues of L when it is diagonalizable. Extending h to multiple input output channels which are indexed in i and j, we have $h_{ij}^{(\ell)}(\lambda) = \sum_k \beta_{ijk}^{(\ell)}\lambda^k$. By defining all components of spectral GNN for graphon, the continuous version of spectral GNN can also be defined. See [78] for details.

We first prove IGN can approximate spectral GNN arbitrarily well, both for discrete SGNN and continuous SGNN. Next, we show that such IGN belongs to IGN-small. We need the following simple assumption to ensure the input lies in a compact domain.

AS5. *There exists an upper bound on* $\|x\|_{L_\infty}$ *for the discrete case and* $\|X\|_{L_\infty}$ *in the continuous case.*

AS6. *min*$(D(A)_{\text{mean}}) \geq c_{\min}$ *where* $D(A)_{\text{mean}}$ *is defined to be* $\frac{1}{n}$*Diag*$(A\mathbf{1})$. *The same lower bound holds for graphon case.*

Lemma 2.9.12. *Assume AS1-AS6 and DMD arbitrarily well in* L_∞ *sense on a compact domain*

Proof. Given diagonal matrix D and matrix M, to implement DMD with linear equivariant layers of 2-IGN, we first use operation 14-15 in Table 2.1 to copy diagonal elements in D to rows and

[4]We follow the same notation as [78], which is different from the conventional notation.

columns of two matrix D_{row} and D_{col}. Then calculating DMD becomes entry-wise multiplication of three matrix $D_{\text{row}}, M, D_{\text{col}}$. Assuming all entries of D and M lies in a compact domain, we can use MLP (which is part of IGN according to Remark 1) to approximate multiplication arbitrarily well [31, 67]. for illustration.

To implement $\frac{1}{n}Mx$ with linear equivariant layers of 2-IGN, first map x into a diagonal matrix $\text{Diag}(x)$ and concatenate it with M as the input $[\text{Diag}(x), M] \in \mathbb{R}^{n\times n\times 2}$ to 2-IGN. Apply "copy diagonal to all columns" to the first channel and use MLP to uniformly approximates up to arbitrary precision ε the multiplication of first channel with the second channel. Then use operation "copy row mean" to map $\mathbb{R}^{n\times n} \to \mathbb{R}^{n}$ to get the $\frac{1}{n}Mx$ within ε precision. See Figure 2.8. $\square$

Remark 7. *Linear layers in 2-IGN can not implement matrix-matrix multiplication in general. When we introduce the matrix multiplication component, the expressive power of GNN in terms of WL test provably increases from 2-WL to 3-WL [109]).*

Theorem 2.9.13. *Given n, ε, and $\text{SGNN}_{\theta_1}(n)$, there exists a 2-IGN $\text{IGN}_{\theta_2}(n)$ such that it approximates $\text{SGNN}_{\theta_1}(n)$ on a compact set (support of input feature x_n) arbitrarily well in L_∞ sense.*

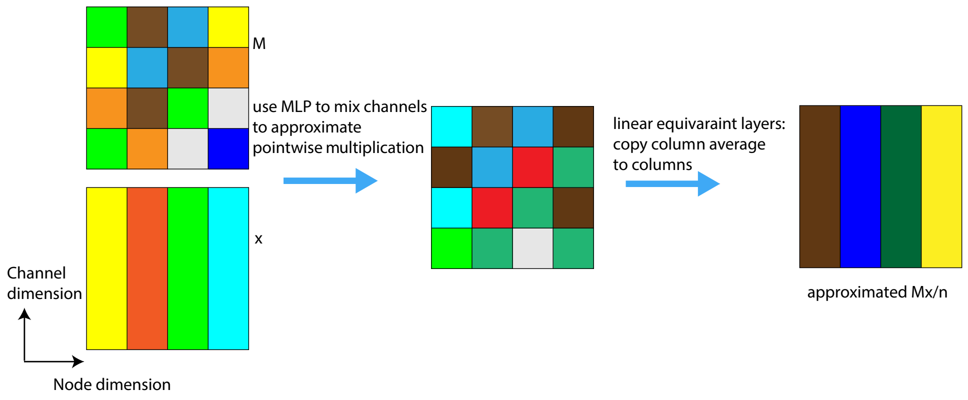

M
use MLP to mix channels to approximate pointwise multiplication
linear equivaraint layers: copy column average to columns
x
Channel dimension
Node dimension
approximated Mx/n

Figure 2.8. An illustration of how we approximate the major building blocks of SGNN: $\frac{1}{n}Ax$.

Proof. Since IGN and SGNN has the same non-linearity. To show that IGN can approximate SGNN, it suffices to show that IGN can approximate linear layer of SGNN, which further boils down to prove that IGN can approximate Lx.

Here we assume the input of 2-IGN is $A \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^{n\times d}$. We need to first show how $L = D(A)^{-\frac{1}{2}}AD(A)^{-\frac{1}{2}}$ can be implemented by linear layers of IGN. This is achieved by noting that $L = \frac{1}{n}D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}$ where $D(A)_{\text{mean}}$ is normalized degree matrix $\frac{1}{n}\text{Diag}(A\mathbf{1})$. Representing L as $\frac{1}{n}D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}$ ensures that all entries in A and $D(A)_{\text{mean}}$ lies in a compact domain, which is crucial when we extending the approximation proof to the graphon case.

Now we show how $Lx = \frac{1}{n}D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}x$ is implemented. First, it is easy to see that 2-IGN can calculate exactly $D(A)_{\text{mean}}$ using equivariant layers. Second, as approximating a) $f(a,b) = ab$ and b) $f(a) = \frac{1}{\sqrt{a}}$ can achieved by MLP on compact domain, approximating $D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}$ can also achieved by 2-IGN layers according to Theorem 2.9.12. Third, we need to show $\frac{1}{n}D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}x$ can also be implemented. This is proved in Theorem 2.9.12.

There are two main functions we need to approximate with MLP: a) $f(x) = 1/\sqrt{a}$ and b) $f(a,b) = ab$.

For a) the input is entries of $D(A)_{\text{mean}}$ which lie in $[0,1]$. By classical universal approximation theorem [31, 67], we know MLP can approximate a) arbitrarily well.

For b) the input is $(D(A)_{\text{mean}}^{-1/2}, A)$ for normalized adjacency matrix calculation, and (L,x) for graph signal convolution.

To ensure the uniform approximation, we need to ensure all of them lie in a compact domain. This is indeed the case as all entries in $D(A)_{\text{mean}}, A, x$ are all upper bounded

1. every entry in A is either 0 or 1 therefore lies in a compact domain.

2. similarly, all entries $D(A)_{\text{mean}}$ lies in $[c_{\min}, 1]$ by AS6, and therefore $D(A)_{\text{mean}}^{-\frac{1}{2}}$ also lies in a compact domain. As $L(A)$ is the multiplication of $D(A)_{\text{mean}}^{-1/2}, A, D(A)_{\text{mean}}^{-1/2}$, every entry of $L(A)$ also lies in compact domain.

3. input signal x has bounded l_∞-norm by assumption AS5.

4. all coefficient for operators is upper bounded and independent from n by AS2.

Since we showed the $L(A)x$ can be approximated arbitrarily well by IGN, repeating such processes and leveraging the fact that L has bounded spectral norm, we can then approximate $L^k(A)x$ up to ε precision. The errors ε depend on the approximation error of the MLP to the relevant function, the previous errors, and uniform bounds as well as uniform continuity of the approximated functions. $\square$

Theorem 2.9.14. *Given ε, and a spectral GNN $c\mathrm{SGNN}_{\theta_1}$, there exists a continuous 2-IGN $c\mathrm{IGN}_{\theta_2}$such that it approximates $c\mathrm{SGNN}_{\theta_1}$ on a compact set (input feature X) arbitrarily well.*

Proof. In the continuous case, $Lx = \frac{1}{n}D(A)_{\text{mean}}^{-\frac{1}{2}}AD(A)_{\text{mean}}^{-\frac{1}{2}}x$ in the discrete case will be replaced with $D(W)^{-\frac{1}{2}}WD(W)^{-\frac{1}{2}}X$ where $D(W)$ is a diagonal graphon defined to be $D(W)(i,i) = \int_0^1 W(i,j)dj$.

We show that all items listed in proof of Theorem 2.9.13 still holds in the continuous case

- we consider the W instead in the continuous case, where all entries still lies in a compact domain $[0,1]$.

- similarly all entries of the continuous analog of $D(A)_{\text{mean}}, D(A)_{\text{mean}}^{-\frac{1}{2}}$, and $T(W)$ also lies in a compact domain according to AS6.

- the statements about input signal X and the coefficient for linear equivariant operators also holds in the continuous setting.

Therefore we conclude the proof. Now we are ready to prove that those IGN that can approximate SGNN well is a subset of IGN-small. $\square$

Lemma 2.9.15. *With slight abuse of notation, let $\widetilde{W_{n,E}}$ be graphon of chessboard pattern. Let $\widetilde{X_{n,E}}$ be a graphon signal with 1D chessboard pattern. $S_n \circ \widetilde{W_{n,E}}\widetilde{X_{n,E}} = (S_n\widetilde{W_{n,E}})(S_n\widetilde{X_{n,E}})$.*

Proof. Since $S_n \circ \widetilde{W_{n,E}}\widetilde{X_{n,E}} = S_n \circ \int_{j\in[0,1]} \widetilde{W_{n,E}}(i,j)\widetilde{X_{n,E}}(j)dj = \left(\ldots, \frac{1}{\sqrt{n}} \int_{j\in[0,1]} \widetilde{W_{n,E}}(\frac{i}{n}, j)\widetilde{X_{n,E}}(j), \ldots\right)$, it suffices to analyze i-th component $\frac{1}{\sqrt{n}} \int_{j\in[0,1]} \widetilde{W_{n,E}}(\frac{i}{n}, j)\widetilde{X_{n,E}}(j)$.

Since $\widetilde{W_{n,E}}, \widetilde{X_{n,E}}$ are of chessboard pattern, we can replace integral with summation.

$$
\begin{aligned}
S_n \circ \widetilde{W_{n,E}}\widetilde{X_{n,E}}(i) &= \frac{1}{\sqrt{n}} \int_{j\in[0,1]} \widetilde{W_{n,E}}(\frac{i}{n}, j)\widetilde{X_{n,E}}(j) \\
&= \frac{1}{\sqrt{n}} \frac{1}{n} \sum_{j\in[n]} \widetilde{W_{n,E}}(\frac{i}{n}, \frac{j}{n})\widetilde{X_{n,E}}(\frac{j}{n}) \\
&= \sum_{j\in[n]} \frac{1}{n} \widetilde{W_{n,E}}(\frac{i}{n}, \frac{j}{n})(S_n\widetilde{X_{n,E}})(j) \\
&= \sum (S_n\widetilde{W_{n,E}})(i,j)(S_n\widetilde{X_{n,E}})(j) \\
&= \left((S_n\widetilde{W_{n,E}})(S_n\widetilde{X_{n,E}})\right)(i)
\end{aligned}
$$

Which concludes the proof. Note that our proof does make use of the property of multiplication between two numbers. □

Remark 8. *The whole proof only relies on that $\widetilde{W_{n,E}}$ and $\widetilde{X_{n,E}}$ have checkerboard patterns. Therefore replacing the multiplication with other operations (such as a MLP) will still hold.*

Theorem 2.5.3. *IGN-small can approximates spectral GNN (both discrete and continuous ones) arbitrarily well on the compact domain in the $\|\cdot\|_{L_\infty}$ sense.*

Proof. To prove this, we only need to show that $S_n\Phi_{c,\text{approx}}([\widetilde{W_{n,E}}, \widetilde{f_{n,E}}]) = \Phi_{d,\text{approx}}S_n([\widetilde{W_{n,E}}, \widetilde{f_{n,E}}])$. Here $\Phi_{c,\text{approx}}$ and $\Phi_{d,\text{approx}}$ denotes those specific IGN in Theorems 2.9.13 and 2.9.14 constructed to approximate SGNN.

To build up some intuition, let Φ_{SGNN} denotes the spectral GNN that Φ_{approx} approximates. it is easy to see that $S_n\Phi_{c,\text{SGNN}}([\widetilde{W_{n,E}}, \widetilde{f_{n,E}}]) = \Phi_{d,\text{SGNN}}S_n([\widetilde{W_{n,E}}, \widetilde{f_{n,E}}])$ due to Theorem 2.9.15 and Theorem 2.9.10.2. To show the same holds for Φ_{approx}, note that the only difference between $\widetilde{W_{n,E}}\widetilde{f_{n,E}}$ implemented by SGNN and approximated by Φ_{approx} is that Φ_{approx} use MLP to simulate multiplication between numbers. According to Remark 8, the approximated version of $\widetilde{W_{n,E}}\widetilde{f_{n,E}}$ still commutes with S_n.

Since nonlinear layer σ in Φ_{approx} also commutes with S_n according to Theorem 2.9.10.2, we can combine the result above and conclude that Φ_{approx} commutes with S_n. Therefore Φ_{approx} belongs to IGN-small, which finishes the proof. $\square$

Chapter 3

On the Connection Between MPNN and Graph Transformer

3.1 Introduction

In this chapter, we study the connection between MPNN and Graph Transformer. MPNN (Message Passing Neural Network) [54] has been the leading architecture for processing graph-structured data. Recently, transformers in natural language processing [142, 75] and vision [45, 59] have extended their success to the domain of graphs. There have been several pieces of work [159, 153, 86, 120, 82] showing that with careful position embedding [100], graph transformers (GT) can achieve compelling empirical performances on large-scale datasets and start to challenge the dominance of MPNN.

Table 3.1. Summary of approximation result of MPNN + VN on self-attention layer. n is the number of nodes and d is the feature dimension of node features. The dependency on d is hidden.

	Depth	Width	Self-Attention	Note
Theorem 3.3.1	$O(1)$	$O(1)$	Approximate	Approximate self attention in Performer [29]
Theorem 3.4.2	$O(1)$	$O(n^d)$	Full	Leverage the universality of equivariant DeepSets
Theorem 3.5.2	$O(n)$	$O(1)$	Full	Explicit construction, strong assumption on $\mathcal{X}$
Theorem 3.10.7	$O(n)$	$O(1)$	Full	Explicit construction, more relaxed (but still strong) assumption on $\mathcal{X}$

MPNN imposes a sparsity pattern on the computation graph and therefore enjoys linear complexity. It however suffers from well-known over-smoothing [96, 114, 21] and over-squashing [3, 140] issues, limiting its usage on long-range modeling tasks where the label of one

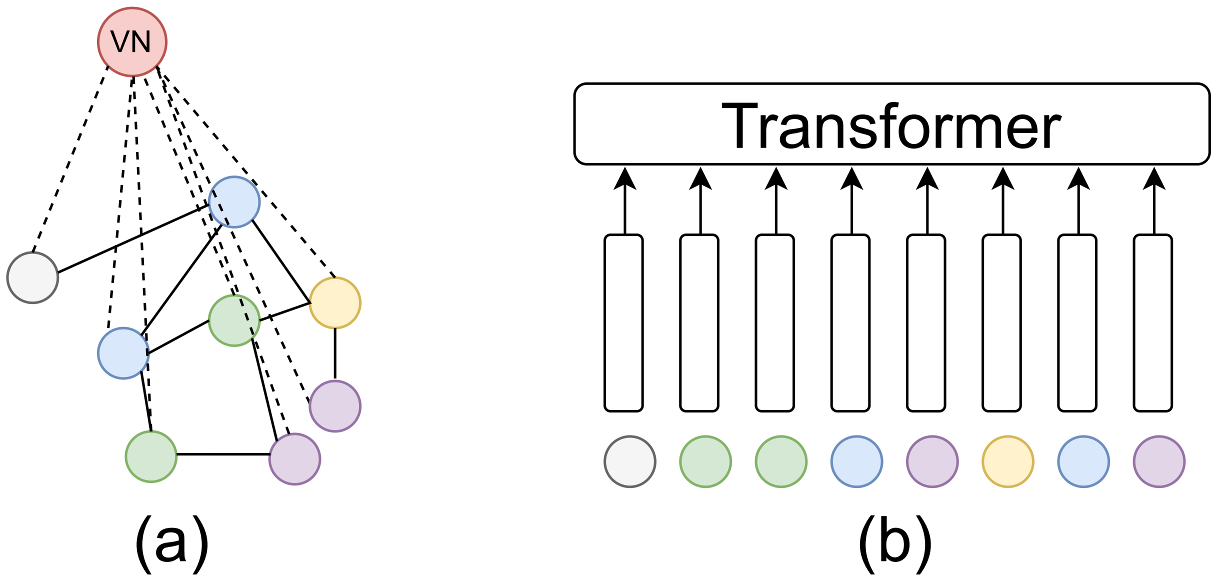


Figure 3.1. MPNN + VN and Graph Transformers.

node depends on features of nodes far away. GT relies purely on position embedding to encode the graph structure and uses vanilla transformers on top. [1] It models all pairwise interactions directly in one layer, making it computationally more expensive. Compared to MPNN, GT shows promising results on tasks where modeling long-range interaction is the key, but the quadratic complexity of self-attention in GT limits its usage to graphs of medium size. Scaling up GT to large graphs remains an active research area [152].

Theoretically, it has been shown that graph transformers can be powerful graph learners [82], i.e., graph transformers with appropriate choice of token embeddings have the capacity of approximating linear permutation equivariant basis, and therefore can approximate 2-IGN (Invariant Graph Network), a powerful architecture that is at least as expressive as MPNN [110]. This raises an important question that *whether GT is strictly more powerful than MPNN*. Can we approximate GT with MPNN?

One common intuition of the advantage of GT over MPNN is its ability to model long-range interaction more effectively. However, from the MPNN side, one can resort to a simple trick

[1]GT in this chapter refers to the practice of tokenizing graph nodes and applying standard transformers on top [159, 82]. There exists a more sophisticated GT [86] that further conditions attention on edge types but it is not considered in this chapter.

to escape locality constraints for effective long-range modeling: the use of an additional *virtual node (VN)* that connects to all input graph nodes. On a high level, MPNN + VN augments the existing graph with one virtual node, which acts like global memory for every node exchanging messages with other nodes. Empirically this simple trick has been observed to improve the MPNN and has been widely adopted [54, 69, 68] since the early beginning of MPNN [54, 8]. However, there is very little theoretical study of MPNN + VN [72].

In this work, we study the theoretical property of MPNN + VN, and its connection to GT. We systematically study the representation power of MPNN + VN, both for certain approximate self-attention and for the full self-attention layer, and provide a depth-width trade-off, summarized in Table 3.1. In particular,

- With $O(1)$ depth and $O(1)$ width, MPNN + VN can approximate one self-attention layer of Performer [29] and Linear Transformer [77], a type of linear transformers [139].
- Via a link between MPNN + VN with DeepSets [162], we prove MPNN + VN with $O(1)$ depth and $O(n^d)$ width (d is the input feature dimension) is permutation equivariant universal, implying it can approximate self-attention layer and even full-transformers.
- Under certain assumptions on node features, we prove an explicit construction of $O(n)$ depth $O(1)$ width MPNN + VN approximating 1 self-attention layer arbitrarily well on graphs of size n. Unfortunately, the assumptions on node features are rather strong, and whether we can alleviate them will be an interesting future direction to explore.
- Empirically, we show 1) that MPNN + VN works surprisingly well on the recently proposed LRGB (long-range graph benchmarks) datasets [44], which arguably require long-range interaction reasoning to achieve strong performance 2) our implementation of MPNN + VN is able to further improve the early implementation of MPNN + VN on OGB datasets and 3) MPNN + VN outperforms Linear Transformer [77] and MPNN on the climate modeling task.

3.2 Preliminaries

We denote $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ the concatenation of graph node features and positional encodings, where node i has feature $\boldsymbol{x}_i \in \mathbb{R}^d$. When necessary, we use $\boldsymbol{x}_j^{(l)}$ to denote the node j's feature at depth l. Let $\mathscr{M}$ be the space of multisets of vectors in $\mathbb{R}^d$. We use $\mathscr{X} \subseteq \mathbb{R}^{n\times d}$ to denote the space of node features and the $\mathscr{X}_i$ be the projection of $\mathscr{X}$ on i-th coordinate. $\|\cdot\|$ denotes the 2-norm. $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ denotes the concatenation of $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$. $[n]$ stands for the set $\{1, 2, ..., n\}$.

Definition 13 (attention). *We denote key and query matrix as $\boldsymbol{W}_K, \boldsymbol{W}_Q \in \mathbb{R}^{d\times d'}$, and value matrix as $\boldsymbol{W}_V \in \mathbb{R}^{d\times d}$ [2]. Attention score between two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{d\times 1}$ is defined as $\alpha(\boldsymbol{u}, \boldsymbol{v}) = \mathrm{softmax}(\boldsymbol{u}^T \boldsymbol{W}_Q (\boldsymbol{W}_K)^T \boldsymbol{v})$. We denote $\mathscr{A}$ as the space of attention α for different $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$. We also define unnormalized attention score $\alpha'(\cdot,\cdot)$ to be $\alpha'(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T \boldsymbol{W}_Q (\boldsymbol{W}_K)^T \boldsymbol{v}$. Self attention layer is a matrix function $\boldsymbol{L} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ of the following form: $\boldsymbol{L}(\boldsymbol{X}) = \mathrm{softmax}(\boldsymbol{X}\boldsymbol{W}_Q(\boldsymbol{X}\boldsymbol{W}_K)^T)\boldsymbol{X}\boldsymbol{W}_V$.*

3.2.1 MPNN Layer

Definition 14 (MPNN layer [54]). *An MPNN layer on a graph G with node features $\boldsymbol{x}^{(k)}$ at k-th layer and edge features $\boldsymbol{e}$ is of the following form*

$$\boldsymbol{x}_i^{(k)} = \gamma^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in\mathscr{N}(i)}\phi^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

Here $\gamma : \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}^d$ is update function, $\phi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{d_e} \to \mathbb{R}^{d'}$ is message function where d_e is the edge feature dimension, $\tau : \mathscr{M} \to \mathbb{R}^d$ is permutation invariant aggregation function and $\mathscr{N}(i)$ is the neighbors of node i in G. Update/message/aggregation functions are usually parametrized by neural networks. For graphs of different types of edges and nodes, one can further extend MPNN to the heterogeneous setting. We use $1, ..., n$ to index graph nodes and $\boldsymbol{n}$ to denote the virtual node.

[2]For simplicity, we assume the output dimension of self-attention is the same as the input dimension. All theoretical results can be extended to the case where the output dimension is different from d.

Definition 15 (heterogeneous MPNN + VN layer)**.** *The heterogeneous MPNN + VN layer operates on two types of nodes: 1) virtual node and 2) graph nodes, denoted as vn and gn, and three types of edges: 1) vn-gn edge and 2) gn-gn edges and 3) gn-vn edges. It has the following form*

$$\boldsymbol{x}_{\boldsymbol{n}}^{(k)} = \gamma_{\boldsymbol{n}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in[n]}\phi_{vn\text{-}gn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right) \tag{3.1}$$

for the virtual node, and

$$\begin{aligned}\boldsymbol{x}_i^{(k)} = \gamma_{gn}^{(k)}(\boldsymbol{x}_i^{(k-1)}, &\tau_{j\in\mathcal{N}_1(i)}\phi_{gn\text{-}vn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\\ &+\tau_{j\in\mathcal{N}_2(i)}\phi_{gn\text{-}gn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i})\right)\end{aligned} \tag{3.2}$$

for graph node. Here $\mathcal{N}_1(i)$ for graph node i is the virtual node and $\mathcal{N}_2(i)$ is the set of neighboring graph nodes.

Our proof of approximating self-attention layer $\boldsymbol{L}$ with MPNN layers does not use the graph topology. Next, we introduce a simplified heterogeneous MPNN + VN layer, which will be used in the proof. It is easy to see that setting $\phi_{\text{gn-gn}}^{(k)}$ to be 0 in Definition 15 recovers the simplified heterogeneous MPNN + VN layer.

Definition 16 (simplified heterogeneous MPNN + VN layer)**.** *A simplified heterogeneous MPNN + VN layer is the same as a heterogeneous MPNN + VN layer in Definition 15 except we set $\theta_{gn\text{-}gn}$ to be 0. I.e., we have*

$$\boldsymbol{x}_{\boldsymbol{n}}^{(k)} = \gamma_{\boldsymbol{n}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in[n]}\phi_{vn\text{-}gn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

for the virtual node, and

$$\boldsymbol{x}_i^{(k)} = \gamma_{gn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in\mathcal{N}_1(i)}\phi_{gn\text{-}vn}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

for graph nodes.

Intuitively, adding the virtual node (VN) to MPNN makes it easy to compute certain quantities, for example, the mean of node features (which is hard for standard MPNN unless the depth is proportional to the diameter of the graph). Using VN thus makes it easy to implement for example the mean subtraction, which helps reduce over-smoothing and improves the performance of GNN [158, 166]. See more connection between MPNN + VN and over-smoothing in Section 3.6.4.

3.2.2 Assumptions

We have two mild assumptions on feature space $\mathscr{X} \subset \mathbb{R}^{n\times d}$ and the regularity of target function $\boldsymbol{L}$.

AS1. $\forall i \in [n], \boldsymbol{x}_i \in \mathscr{X}_i, \|\boldsymbol{x}_i\| < C_1$. *This implies $\mathscr{X}$ is compact.*

AS2. $\|\boldsymbol{W}_Q\| < C_2, \|\boldsymbol{W}_K\| < C_2, \|\boldsymbol{W}_V\| < C_2$ *for target layer $\boldsymbol{L}$. Combined with AS1 on $\mathscr{X}$, this means $\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is both upper and lower bounded, which further implies $\sum_j e^{\alpha'(\boldsymbol{x}_i,\boldsymbol{x}_j)}$ be both upper bounded and lower bounded.*

3.2.3 Notations

We provide a notation table for references.

3.3 $O(1)$-Depth $O(1)$-Width MPNN + VN for Unbiased Approximation of Attention

The standard self-attention takes $O(n^2)$ computational time, therefore not scalable for large graphs. Reducing the computational complexity of self-attention in Transformer is active research [139]. In this section, we consider self-attention in a specific type of efficient transformers, Performer [29] and Linear Transformer [77].

Table 3.2. Summary of important notations.

Symbol	Meaning
$\boldsymbol{X} \in \mathscr{X} \subset \mathbb{R}^{n\times d}$	graph node features
$\boldsymbol{x}_i \in \mathbb{R}^{1\times d}$	graph node i's feature
$\tilde{\boldsymbol{x}}_i \in \mathbb{R}^{1\times d}$	approximated graph node i's feature via attention selection
$\mathscr{M}$	A multiset of vectors in $\mathbb{R}^d$
$\boldsymbol{W}_Q^{(l)}, \boldsymbol{W}_K^{(l)}, \boldsymbol{W}_V^{(l)} \in \mathbb{R}^{d\times d'}$	attention matrix of l-th self-attention layer in graph transformer
$\mathscr{X}$	feature space
$\mathscr{X}_i$	projection of feature space onto i-th coordinate
$\boldsymbol{L}_i^{\text{ds}}$	i-th linear permutation equivariant layer in DeepSets
$\boldsymbol{L}, \boldsymbol{L}'$	full self attention layer; approximate self attention layer in Performer
$\boldsymbol{z}_{\boldsymbol{n}}^{(l)}, \boldsymbol{z}_i^{(l)}$	virtual/graph node feature at layer l of heterogeneous MPNN + VN
$\alpha_{\boldsymbol{n}}$	attention score in MPNN + VN
$\alpha(\cdot,\cdot)$	normalized attention score
$\alpha_{\text{GATv2}}(\cdot,\cdot)$	normalized attention score with GATv2
$\alpha'(\cdot,\cdot)$	unnormalized attention score. $\alpha'(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{u}\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v}^T$
$\alpha'_{\text{GATv2}}(\cdot,\cdot)$	unnormalized attention score with GATv2. $\alpha'_{\text{GATv2}}(\boldsymbol{u},\boldsymbol{v}) := \boldsymbol{a}^T\,\text{LeakyReLU}\left(\boldsymbol{W}\cdot[\boldsymbol{u}\Vert\boldsymbol{v}]+\boldsymbol{b}\right)$
$\mathscr{A}$	space of attentions, where each element $\alpha \in \mathscr{A}$ is of form $\alpha(\boldsymbol{u},\boldsymbol{v}) = \text{softmax}(\boldsymbol{u}\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v}^T)$
C_1	upper bound on norm of all node features $\Vert\boldsymbol{x}_i\Vert$
C_2	upper bound on the norm of $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$ in target $\boldsymbol{L}$
C_3	upper bound on the norm of attention weights of $\alpha_{\boldsymbol{n}}$ when selecting $\boldsymbol{x}_i$
$\gamma^{(k)}(\cdot,\cdot)$	update function
$\theta^{(k)}(\cdot,\cdot)$	message function
$\tau(\cdot)$	aggregation function

One full self-attention layer $\boldsymbol{L}$ is of the following form

$$\boldsymbol{x}_i^{(l+1)} = \sum_{j=1}^{n} \frac{\kappa\left(\boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{W}_K^{(l)}\boldsymbol{x}_j^{(l)}\right)}{\sum_{k=1}^{n}\kappa\left(\boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{W}_K^{(l)}\boldsymbol{x}_k^{(l)}\right)} \cdot \left(\boldsymbol{W}_V^{(l)}\boldsymbol{x}_j^{(l)}\right) \tag{3.3}$$

where $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the softmax kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) := \exp(\boldsymbol{x}^T\boldsymbol{y})$. The kernel function can be approximated via $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \langle\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y})\rangle_{\mathscr{V}} \approx \phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$ where the first equation is by Mercer's theorem and $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$ is a low-dimensional feature map with random transformation. For Performer [29], the choice of ϕ is taken as $\phi(\boldsymbol{x}) = \frac{\exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2}\right)}{\sqrt{m}}\left[\exp\left(\boldsymbol{w}_1^T\boldsymbol{x}\right), \cdots, \exp\left(\boldsymbol{w}_m^T\boldsymbol{x}\right)\right]$ where $\boldsymbol{w}_k \sim \mathscr{N}(0, I_d)$ is i.i.d sampled random variable. For Linear Transformer [77], $\phi(\boldsymbol{x}) = \mathrm{elu}(\boldsymbol{x}) + 1$.

By switching $\kappa(\boldsymbol{x}, \boldsymbol{y})$ to be $\phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$, and denote $\boldsymbol{q}_i = \boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{k}_i = \boldsymbol{W}_K^{(l)}\boldsymbol{x}_i^{(l)}$ and $\boldsymbol{v}_i = \boldsymbol{W}_V^{(l)}\boldsymbol{x}_i^{(l)}$, the approximated version of Equation (3.3) by Performer and Linear Transformer becomes

$$\begin{aligned} \boldsymbol{x}_i^{(l+1)} &= \sum_{j=1}^{n} \frac{\phi\left(\boldsymbol{q}_i\right)^T\phi\left(\boldsymbol{k}_j\right)}{\sum_{k=1}^{n}\phi\left(\boldsymbol{q}_i\right)^T\phi\left(\boldsymbol{k}_k\right)} \cdot \boldsymbol{v}_j \\ &= \frac{\left(\phi\left(\boldsymbol{q}_i\right)^T\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j\right)^T}{\phi\left(\boldsymbol{q}_i\right)^T\sum_{k=1}^{n}\phi\left(\boldsymbol{k}_k\right)}. \end{aligned} \tag{3.4}$$

where we use the matrix multiplication association rule to derive the second equality.

The key advantage of Equation (3.4) is that $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)$ and $\sum_{j=1}^{n}\phi(\boldsymbol{k}_j)\otimes\boldsymbol{v}_j$ can be approximated by the virtual node, and shared for all graph nodes, using only $O(1)$ layers of MPNNs. We denote the self-attention layer of this form in Equation (3.4) as $\boldsymbol{L}_{\text{Performer}}$. Linear Transformer differs from Performer by choosing a different form of $\phi(\boldsymbol{x}) = \mathrm{Relu}(\boldsymbol{x}) + 1$ in its self-attention layer $\boldsymbol{L}_{\text{Linear-Transformer}}$.

In particular, the VN will approximate $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)$ and $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j$, and represent it as its feature. Both $\phi\left(\boldsymbol{k}_j\right)$ and $\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j$ can be approximated arbitrarily well by an MLP with constant width (constant in n but can be exponential in d) and depth. Note that $\phi(\boldsymbol{k}_j)\otimes\boldsymbol{v}_j \in \mathbb{R}^{dm}$

but can be reshaped to 1 dimensional feature vector.

More specifically, the initial feature for the virtual node is $\mathbf{1}_{(d+1)m}$, where d is the dimension of node features and m is the number of random projections ω_i. Message function + aggregation function for virtual node $\tau\phi_{\text{vn-gn}} : \mathbb{R}^{(d+1)m} \times \mathscr{M} \to \mathbb{R}^{(d+1)m}$ is

$$\begin{aligned} \tau_{j\in[n]}\phi^{(k)}_{\text{vn-gn}}(\cdot, \{\boldsymbol{x}_i\}_i) = [&\sum_{j=1}^{n} \phi\left(\boldsymbol{k}_j\right), \\ &\texttt{ReshapeTo1D}(\sum_{j=1}^{n} \phi\left(\boldsymbol{k}_j\right) \otimes \boldsymbol{v}_j)] \end{aligned} \tag{3.5}$$

where $\texttt{ReshapeTo1D}(\cdot)$ flattens a 2D matrix to a 1D vector in raster order. This function can be arbitrarily approximated by MLP. Note that the virtual node's feature dimension is $(d+1)m$ (where recall m is the dimension of the feature map ϕ used in the linear transformer/Performer), which is larger than the dimension of the graph node d. This is consistent with the early intuition that the virtual node might be overloaded when passing information among nodes. The update function for virtual node $\gamma_{\boldsymbol{n}} : \mathbb{R}^{(d+1)m} \times \mathbb{R}^{(d+1)m} \to \mathbb{R}^{(d+1)m}$ is just coping the second argument, which can be exactly implemented by MLP.

VN then sends its message back to all other nodes, where each graph node i applies the update function $\gamma_{\text{gn}} : \mathbb{R}^{(d+1)m} \times \mathbb{R}^{d} \to \mathbb{R}^{d}$ of the form

$$\begin{aligned} &\gamma_{\text{gn}}(\boldsymbol{x}_i, [\sum_{j=1}^{n} \phi\left(\boldsymbol{k}_j\right), \texttt{ReshapeTo1D}(\sum_{j=1}^{n} \phi\left(\boldsymbol{k}_j\right) \otimes \boldsymbol{v}_j)]) \\ &= \frac{\left(\phi\left(\boldsymbol{q}_i\right) \sum_{j=1}^{n} \phi\left(\boldsymbol{k}_j\right) \otimes \boldsymbol{v}_j\right)^T}{\phi\left(\boldsymbol{q}_i\right)^T \sum_{k=1}^{n} \phi\left(\boldsymbol{k}_k\right)} \end{aligned} \tag{3.6}$$

to update the graph node feature.

As the update function γ_{gn} can not be computed exactly in MLP, what is left is to show that error induced by using MLP to approximate $\tau\phi_{\text{vn-gn}}$ and γ_{gn} in Equation (3.5) and Equation (3.6) can be made arbitrarily small.

Theorem 3.3.1. *Under the AS1 and AS2, MPNN + VN of $O(1)$ width and $O(1)$ depth can* *approximate $\boldsymbol{L}_{\text{Performer}}$ and $\boldsymbol{L}_{\text{Linear-Transformer}}$ arbitrarily well.*

Proof. We first prove the case of $\boldsymbol{L}_{\text{Performer}}$. We can decompose our target function as the composition of $\tau_{j\in[n]}\phi^{(k)}_{\text{vn-gn}}$, γ_{gn} and ϕ. By the uniform continuity of the functions, it suffices to show that 1) we can approximate ϕ, 2) we can approximate operations in γ_{gn} and $\tau\phi_{\text{vn-gn}}$ arbitrarily well on the compact domain, and 3) the denominator $\phi\left(\boldsymbol{q}_i\right)^T\sum_{k=1}^{n}\phi\left(\boldsymbol{k}_k\right)$ is uniformly lower bounded by a positive number for any node features in $\mathscr{X}$.

For 1), each component of ϕ is continuous and all inputs $\boldsymbol{k}_j, \mathbf{q}_j$ lie in the compact domain so ϕ can be approximated arbitrarily well by MLP with $O(1)$ width and $O(1)$ depth [31].

For 2), we need to approximate the operations in γ_{gn} and $\tau\phi_{\text{vn-gn}}$, i.e., approximate multiplication, and vector-scalar division arbitrarily well. As all those operations are continuous, it boils down to showing that all operands lie in a compact domain. By assumption AS1 and AS2 on $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$ and input feature $\mathscr{X}$, we know that $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i$ lies in a compact domain for all graph nodes i. As ϕ is continuous, this implies that $\phi(\boldsymbol{q}_i), \sum_{j=1}^{n}\phi(\boldsymbol{k}_j)\otimes\boldsymbol{v}_j$ lies in a compact domain (n is fixed), therefore the numerator lies in a compact domain. Lastly, since all operations do not involve n, the depth and width are constant in n.

For 3), it is easy to see that $\phi\left(\boldsymbol{q}_i\right)^T\sum_{k=1}^{n}\phi\left(\boldsymbol{k}_k\right)$ is always positive. We just need to show that the denominator is bound from below by a positive constant. For Performer, $\phi(\boldsymbol{x}) = \frac{\exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2}\right)}{\sqrt{m}}\left[\exp\left(\boldsymbol{w}_1^T\boldsymbol{x}\right), \cdots, \exp\left(\boldsymbol{w}_m^T\boldsymbol{x}\right)\right]$ where $\boldsymbol{w}_k \sim \mathscr{N}\left(0, I_d\right)$. As all norm of input $\boldsymbol{x}$ to ϕ is upper bounded by AS1, $\exp(\frac{-\|\boldsymbol{x}\|_2^2}{2})$ is lower bounded. As m is fixed, we know that $\|\boldsymbol{w}_i^T\boldsymbol{x}\| \leq \|\boldsymbol{w}_i\|\|\boldsymbol{x}\|$, which implies that $\boldsymbol{w}_i^T\boldsymbol{x}$ is lower bounded by $-\|\boldsymbol{w}_i\|\|\boldsymbol{x}\|$ which further implies that $\exp(\boldsymbol{w}_i^T\boldsymbol{x})$ is lower bounded. This means that $\phi\left(\boldsymbol{q}_i\right)^T\sum_{k=1}^{n}\phi\left(\boldsymbol{k}_k\right)$ is lower bounded.

For Linear Transformer, the proof is essentially the same as above. We only need to show that $\phi(\boldsymbol{x}) = \text{elu}(\boldsymbol{x}) + 1$ is continuous and positive, which is indeed the case. $\square$

Besides Performers, there are many other different ways of obtaining linear complexity. In Section 3.7.2, we discuss the limitation of MPNN + VN on approximating other types of efficient transformers such as Linformer [147] and Sparse Transformer [28].

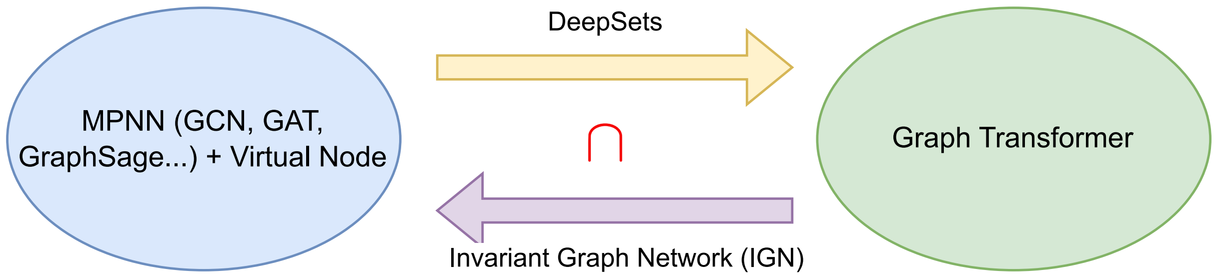

DeepSets
MPNN (GCN, GAT, GraphSage...) + Virtual Node
$\cap$
Graph Transformer
Invariant Graph Network (IGN)

Figure 3.2. The link between MPNN and GT is drawn via DeepSets in Section 3.4 of this chapter and Invariant Graph Network (IGN) in [82]. Interestingly, IGN is a generalization of DeepSets [110].

3.4 $O(1)$ Depth $O(n^d)$ Width MPNN + VN

We have shown that the MPNN + VN can approximate self-attention in Performer and Linear Transformer using only $O(1)$ depth and $O(1)$ width. One may naturally wonder whether MPNN + VN can approximate the self-attention layer in the *full* transformer. In this section, we show that MPNN + VN with $O(1)$ depth (number of layers), but with $O(n^d)$ width, can approximate 1 self-attention layer (and full transformer) arbitrarily well.

The main observation is that MPNN + VN is able to exactly simulate (not just approximate) equivariant DeepSets [162], which is proved to be universal in approximating any permutation invariant/equivariant maps [162, 131]. Since the self-attention layer is permutation equivariant, this implies that MPNN + VN can approximate the self-attention layer (and full transformer) with $O(1)$ depth and $O(n^d)$ width following a result on DeepSets from [131].

We first introduce the permutation equivariant map, equivariant DeepSets, and permutation equivariant universality.

Definition 17 (permutation equivariant map)**.** *A map $\boldsymbol{F}: \mathbb{R}^{n\times k} \to \mathbb{R}^{n\times l}$ satisfying $\boldsymbol{F}(\sigma \cdot \boldsymbol{X}) = \sigma \cdot \boldsymbol{F}(\boldsymbol{X})$ for all $\sigma \in S_n$ and $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is called permutation equivariant.*

Definition 18 (equivariant DeepSets of [162])**.** *Equivariant DeepSets has the following form $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{L}^{ds}_m \circ \nu \circ \cdots \circ \nu \circ \boldsymbol{L}^{ds}_1(\boldsymbol{X})$, where $\boldsymbol{L}^{ds}_i$ is a linear permutation equivariant layer and ν is a nonlinear layer such as ReLU. The linear permutation equivariant layer in DeepSets has* *the following form* $\boldsymbol{L}_i^{ds}(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{A} + \frac{1}{n}\mathbf{1}\mathbf{1}^T\boldsymbol{X}\boldsymbol{B} + \mathbf{1}\boldsymbol{c}^T$*, where* $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d_i \times d_{i+1}}$*,* $\boldsymbol{c} \in \mathbb{R}^{d_{i+1}}$ *is the weights and bias in layer i, and* ν *is ReLU.*

Definition 19 (permutation equivariant universality)**.** *Given a compact domain* $\mathscr{X}$ *of* $\mathbb{R}^{n \times d_{in}}$*, permutation equivariant universality of a model* $\boldsymbol{F} : \mathbb{R}^{n \times d_{in}} \to \mathbb{R}^{n \times d_{out}}$ *means that for every permutation equivariant continuous function* $\boldsymbol{H} : \mathbb{R}^{n \times d_{in}} \to \mathbb{R}^{n \times d_{out}}$ *defined over* $\mathscr{X}$*, and any* $\varepsilon > 0$*, there exists a choice of m (i.e., network depth),* d_i *(i.e., network width at layer i) and the trainable parameters of* $\boldsymbol{F}$ *so that* $\|\boldsymbol{H}(\boldsymbol{X}) - \boldsymbol{F}(\boldsymbol{X})\|_\infty < \varepsilon$ *for all* $\boldsymbol{X} \in \mathscr{X}$*.*

The universality of equivariant DeepSets is stated as follows.

Theorem 3.4.1 ([131])**.** *DeepSets with constant layer is universal. Using ReLU activation the width* $\omega := max_i d_i$ *(*d_i *is the width for i-th layer of DeepSets) required for universal permutation equivariant network satisfies* $\omega \leq d_{out} + d_{in} + \begin{pmatrix} n + d_{in} \\ d_{in} \end{pmatrix} = O(n^{d_{in}})$*.*

We are now ready to state our main theorem.

Theorem 3.4.2. *MPNN + VN can simulate (not just approximate) equivariant DeepSets:* $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$*. The depth and width of MPNN + VN needed to simulate DeepSets is up to a constant factor of the depth and width of DeepSets. This implies that MPNN + VN of* $O(1)$ *depth and* $O(n^d)$ *width is permutation equivariant universal, and can approximate self-attention layer and transformers arbitrarily well.*

Proof. Equivariant DeepSets has the following form $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{L}_m^{\text{ds}} \circ \nu \circ \cdots \circ \nu \circ \boldsymbol{L}_1^{\text{ds}}(\boldsymbol{X})$, where $\boldsymbol{L}_i^{\text{ds}}$ is the linear permutation equivariant layer and ν is an entrywise nonlinear activation layer. Recall that the linear equivariant layer has the form $\boldsymbol{L}_i^{\text{ds}}(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{A} + \frac{1}{n}\mathbf{1}\mathbf{1}^T\boldsymbol{X}\boldsymbol{B} + \mathbf{1}\boldsymbol{c}^T$. As one can use the same nonlinear entrywise activation layer ν in MPNN + VN, it suffices to prove that MPNN + VN can compute linear permutation equivariant layer $\boldsymbol{L}^{\text{ds}}$. Now we show that 2 layers of MPNN + VN can exactly simulate any given linear permutation equivariant layer $\boldsymbol{L}^{\text{ds}}$.

Table 3.3. Baselines for `Peptides-func` (graph classification) and `Peptides-struct` (graph regression). The performance metric is Average Precision (AP) for classification and MAE for regression. **Bold**: Best score.

Model	**# Params.**	`Peptides-func`		`Peptides-struct`	
		Test AP before VN	**Test AP after VN** $\uparrow$	**Test MAE before VN**	**Test MAE after VN** $\downarrow$
GCN	508k	0.5930±0.0023	0.6623±0.0038	0.3496±0.0013	**0.2488±0.0021**
GINE	476k	0.5498±0.0079	0.6346±0.0071	0.3547±0.0045	0.2584±0.0011
GatedGCN	509k	0.5864±0.0077	0.6635±0.0024	0.3420±0.0013	0.2523±0.0016
GatedGCN+RWSE	506k	0.6069±0.0035	**0.6685±0.0062**	0.3357±0.0006	0.2529±0.0009
Transformer+LapPE	488k	0.6326±0.0126	-	0.2529±0.0016	-
SAN+LapPE	493k	0.6384±0.0121	-	0.2683±0.0043	-
SAN+RWSE	500k	0.6439±0.0075	-	0.2545±0.0012	-

Specifically, at layer 0, we initialized the node features as follows: The VN node feature is set to 0, while the node feature for the i-th graph node is set up as $\boldsymbol{x}_i \in \mathbb{R}^d$.

At layer 1: VN node feature is $\frac{1}{n}\mathbf{1}\mathbf{1}^T\boldsymbol{X}$, average of node features. The collection of features over n graph node feature is $\boldsymbol{X}\boldsymbol{A}$. We only need to transform graph node features by a linear transformation, and set the VN feature as the average of graph node features in the last iteration. Both can be exactly implemented in Definition 16 of simplified heterogeneous MPNN + VN.

At layer 2: VN node feature is set to be 0, and the graph node feature is $\boldsymbol{X}\boldsymbol{A}+\frac{1}{n}\mathbf{1}\mathbf{1}^T\boldsymbol{X}\boldsymbol{B}+\mathbf{1}\boldsymbol{c}^T$. Here we only need to perform the matrix multiplication of the VN feature with $\boldsymbol{B}$, as well as add a bias $\boldsymbol{c}$. This can be done by implementing a linear function for γ_{gn}.

It is easy to see the width required for MPNN + VN to simulate DeepSets is constant. Thus, one can use 2 layers of MPNN + VN to compute linear permutation equivariant layer $\boldsymbol{L}_i^{\text{ds}}$, which implies that MPNN + VN can simulate 1 layer of DeepSets exactly with constant depth and constant width (independent of n). Then by the universality of DeepSets, stated in Theorem 3.4.1, we conclude that MPNN + VN is also permutation equivariant universal, which implies that the constant layer of MPNN + VN with $O(n^d)$ width is able to approximate any continuous equivariant maps. As the self-attention layer $\boldsymbol{L}$ and full transformer are both continuous and equivariant, they can be approximated by MPNN + VN arbitrarily well. $\square$

Thanks to the connection between MPNN + VN with DeepSets, there is no extra assumption on $\mathscr{X}$ except for being compact. The drawback on the other hand is that the upper bound on the computational complexity needed to approximate the self-attention with wide MPNN + VN is worse than directly computing self-attention when $d > 2$.

3.5 $O(n)$ Depth $O(1)$ Width MPNN + VN

The previous section shows that we can approximate a full attention layer in Transformer using MPNN with $O(1)$ depth but $O(n^d)$ width where n is the number of nodes and d is the dimension of node features. In practice, it is not desirable to have the width depend on the graph size.

In this section, we hope to study MPNN + VNs with $O(1)$ width and their ability to approximate a self-attention layer in the Transformer. However, this appears to be much more challenging. Our result in this section only shows that for a rather restrictive family of input graphs (see Assumption 3 below), we can approximate a full self-attention layer of transformer with an MPNN + VN of $O(1)$ width and $O(n)$ depth. We leave the question of MPNN + VN's ability in approximate transformers for more general families of graphs for future investigation.

We first introduce the notion of $(\boldsymbol{V}, \delta)$ separable node features. This is needed to ensure that VN can approximately select one node feature to process at each iteration with attention $\alpha_{\boldsymbol{n}}$, the self-attention in the virtual node.

Definition 20 (($\boldsymbol{V}, \delta$) separable by α)**.** *Given a graph G of size n and a fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_n]$ and $\bar{\alpha} \in \mathscr{A}$, we say node feature $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ of G is $(\boldsymbol{V}, \delta)$ separable by some $\bar{\alpha}$ if the following holds. For any node feature $\boldsymbol{x}_i$, there exist weights $\boldsymbol{W}_K^{\bar{\alpha}}, \boldsymbol{W}_Q^{\bar{\alpha}}$ in attention score $\bar{\alpha}$ such that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j\neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i) + \delta$. We say set $\mathscr{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ if every element $\boldsymbol{X} \in \mathscr{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$.*

The use of $(\boldsymbol{V}, \delta)$ separability is to approximate hard selection function arbitrarily well, which is stated below and proved in Section 3.10.1.

Table 3.4. Test performance in graph-level OGB benchmarks [69]. Shown is the mean ± s.d. of 10 runs.

Model	ogbg-molhiv	ogbg-molpcba	ogbg-ppa	ogbg-code2
	AUROC ↑	**Avg. Precision ↑**	**Accuracy ↑**	**F1 score ↑**
GCN	0.7606 ± 0.0097	0.2020 ± 0.0024	0.6839 ± 0.0084	0.1507 ± 0.0018
GCN+virtual node	0.7599 ± 0.0119	0.2424 ± 0.0034	0.6857 ± 0.0061	0.1595 ± 0.0018
GIN	0.7558 ± 0.0140	0.2266 ± 0.0028	0.6892 ± 0.0100	0.1495 ± 0.0023
GIN+virtual node	0.7707 ± 0.0149	0.2703 ± 0.0023	0.7037 ± 0.0107	0.1581 ± 0.0026
SAN	0.7785 ± 0.2470	0.2765 ± 0.0042	–	–
GraphTrans (GCN-Virtual)	–	0.2761 ± 0.0029	–	0.1830 ± 0.0024
K-Subtree SAT	–	–	0.7522 ± 0.0056	0.1937 ± 0.0028
GPS	0.7880 ± 0.0101	0.2907 ± 0.0028	0.8015 ± 0.0033	0.1894 ± 0.0024
MPNN + VN + NoPE	0.7676 ± 0.0172	0.2823 ± 0.0026	0.8055 ± 0.0038	0.1727 ± 0.0017
MPNN + VN + PE	0.7687 ± 0.0136	0.2848 ± 0.0026	0.8027 ± 0.0026	0.1719 ± 0.0013

Lemma 3.5.1 (approximate hard selection). *Given $\mathscr{X}$ is $(\boldsymbol{V},\delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d}$, $\bar{\alpha} \in \mathscr{A}$ and $\delta > 0$, the following holds. For any $\varepsilon > 0$ and $i \in [n]$, there exists a set of attention weights $\boldsymbol{W}_{i,Q}, \boldsymbol{W}_{i,K}$ in i-th layer of MPNN + VN such that $\alpha_{\boldsymbol{n}}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1-\varepsilon$ for any $\boldsymbol{x}_i \in \mathscr{X}_i$. In other words, we can approximate a hard selection function $f_i(\boldsymbol{x}_1,...,\boldsymbol{x}_n) = \boldsymbol{x}_i$ arbitrarily well on $\mathscr{X}$ by setting $\alpha_{\boldsymbol{n}} = \bar{\alpha}$.*

With the notation set up, We now state an extra assumption needed for deep MPNN + VN case and the main theorem.

AS3. *$\mathscr{X}$ is $(\boldsymbol{V},\delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d}$, $\bar{\alpha} \in \mathscr{A}$ and $\delta > 0$.*

Theorem 3.5.2. *Assume AS 1-3 hold for the compact set $\mathscr{X}$ and $\boldsymbol{L}$. Given any graph G of size n with node features $\boldsymbol{X} \in \mathscr{X}$, and a self-attention layer $\boldsymbol{L}$ on G (fix $\boldsymbol{W}_K, \boldsymbol{W}_Q, \boldsymbol{W}_V$ in α), there exists a $O(n)$ layer of heterogeneous MPNN + VN with the specific aggregate/update/message function that can approximate $\boldsymbol{L}$ on $\mathscr{X}$ arbitrarily well.*

The proof is presented in the Section 3.10. On the high level, we can design an MPNN + VN where the *i*-th layer will select $\tilde{\boldsymbol{x}}_i$, an approximation of $\boldsymbol{x}_i$ via attention mechanism, enabled by Theorem 3.5.1, and send $\tilde{\boldsymbol{x}}_i$ to the virtual node. Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes and computes the approximation of $e^{\alpha(\boldsymbol{x}_i,\boldsymbol{x}_j)}, \forall j \in [n]$. Repeat such procedures *n* times for all graph nodes, and finally, use the last layer for attention normalization. A slight relaxation of AS3 is also provided in Section 3.10.

Table 3.5. Evaluation on PCQM4Mv2 [68] dataset. For GPS evaluation, we treated the *validation* set of the dataset as a test set, since the *test-dev* set labels are private.

Model	**PCQM4Mv2**			
	Test-dev MAE ↓	**Validation MAE ↓**	**Training MAE**	**# Param.**
GCN	0.1398	0.1379	n/a	2.0M
GCN-virtual	0.1152	0.1153	n/a	4.9M
GIN	0.1218	0.1195	n/a	3.8M
GIN-virtual	0.1084	0.1083	n/a	6.7M
GRPE [115]	0.0898	0.0890	n/a	46.2M
EGT [71]	0.0872	0.0869	n/a	89.3M
Graphormer [133]	n/a	0.0864	0.0348	48.3M
GPS-small	n/a	0.0938	0.0653	6.2M
GPS-medium	n/a	0.0858	0.0726	19.4M
MPNN + VN + PE (small)	n/a	0.0942	0.0617	5.2M
MPNN + VN + PE (medium)	n/a	0.0867	0.0703	16.4M
MPNN + VN + NoPE (small)	n/a	0.0967	0.0576	5.2M
MPNN + VN + NoPE (medium)	n/a	0.0889	0.0693	16.4M

3.6 Experiments

We benchmark MPNN + VN for three tasks, long range interaction modeling in Section 3.6.2 and OGB regression tasks in Section 3.6.3. The code is available https://github.com/Chen-Cai-OSU/MPNN-GT-Connection.

3.6.1 Dataset Description

ogbg-molhiv and **ogbg-molpcba** [69] are molecular property prediction datasets adopted by OGB from MoleculeNet. These datasets use a common node (atom) and edge (bond) featurization that represent chemophysical properties. The prediction task of ogbg-molhiv is a binary classification of molecule's fitness to inhibit HIV replication. The ogbg-molpcba, derived from PubChem BioAssay, targets to predict the results of 128 bioassays in the multi-task binary classification setting.

ogbg-ppa [153] consists of protein-protein association (PPA) networks derived from 1581 species categorized into 37 taxonomic groups. Nodes represent proteins and edges encode the normalized level of 7 different associations between two proteins. The task is to classify which of the 37 groups does a PPA network originate from.

ogbg-code2 [153] consists of abstract syntax trees (ASTs) derived from the source code of functions written in Python. The task is to predict the first 5 subtokens of the original function's name.

OGB-LSC PCQM4Mv2 [68] is a large-scale molecular dataset that shares the same featurization as ogbg-mol* datasets. It consists of 529,434 molecule graphs. The task is to predict the HOMO-LUMO gap, a quantum physical property originally calculated using Density Functional Theory. True labels for original test-dev and test-challange dataset splits are kept private by the OGB-LSC challenge organizers. Therefore for the purpose of this chapter, we used the original validation set as the test set, while we left out random 150K molecules for our validation set.

3.6.2 MPNN + VN for LRGB Datasets

We experiment with MPNN + VN for Long Range Graph Benchmark (LRGB) datasets. Original paper [44] observes that GT outperforms MPNN on 4 out of 5 datasets, among which GT shows significant improvement over MPNN on `Peptides-func` and `Peptides-struct` for all MPNNs. To test the effectiveness of the virtual node, we take the original code and modify the graph topology by adding a virtual node and keeping the hyperparameters of all models unchanged.

Results are in Table 3.3. Interestingly, such a simple change can boost MPNN + VN by a large margin on `Peptides-func` and `Peptides-struct`. Notably, with the addition of VN, GatedGCN + RWSE (random-walk structural encoding) after augmented by VN **outperforms all transformers** on `Peptides-func`, and GCN outperforms transformers on `Peptides-struct`.

3.6.3 Stronger MPNN + VN Implementation

Next, by leveraging the modularized implementation from GraphGPS [120], we implemented a version of MPNN + VN with/without extra positional embedding. Our goal is not to achieve SOTA but instead to push the limit of MPNN + VN and better understand the source of the performance gain for GT. In particular, we replace the GlobalAttention Module in GraphGPS with DeepSets, which is equivalent to one specific version of MPNN + VN. We tested this specific version of MPNN + VN on 4 OGB datasets, both with and without the use of positional embedding. The results are reported in Table 3.4. Interestingly, even without the extra position embedding, our MPNN + VN is able to further improve over the previous GCN + VN & GIN + VN implementation. The improvement on **ogbg-ppa** is particularly impressive, which is from 0.7037 to 0.8055. Furthermore, it is important to note that while MPNN + VN does not necessarily outperform GraphGPS, which is a state-of-the-art architecture using both MPNN, Position/structure encoding and Transformer, the difference is quite small – this however, is achieved by a simple MPNN + VN architecture.

We also test MPNN + VN on large-scale molecule datasets PCQMv2, which has 529,434 molecule graphs. We followed [120] and used the original validation set as the test set, while we left out random 150K molecules for our validation set. As we can see from Table 3.5, MPNN + VN + NoPE performs significantly better than the early MPNN + VN implementation: GIN + VN and GCN + VN. The performance gap between GPS on the other hand is rather small: 0.0938 (GPS) vs. 0.0942 (MPNN + VN + PE) for the small model and 0.0858 (GPS) vs. 0.0867 (MPNN + VN + PE) for the medium model.

3.6.4 Connection to Over-Smoothing Phenomenon

Over-smoothing refers to the phenomenon that deep GNN will produce same features at different nodes after too many convolution layers. Here we draw some connection between VN and common ways of reducing over-smoothing. We think that using VN can potentially help alleviate the over-smoothing problem. In particular, we note that the use of VN can simulate some strategies people use in practice to address over-smoothing. We give two examples below.

Example 1: In [166], the two-step method (center & scale) PairNorm is proposed to reduce the over-smoothing issues. In particular, PairNorm consists of 1) Center and 2) Scale

$$\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_i \tilde{\mathbf{x}}_i$$

$$\dot{\mathbf{x}}_i = s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_i ||\tilde{\mathbf{x}}_i^c||_2^2}}$$

Where $\tilde{\mathbf{x}}$ is the node features after graph convolution and s is a hyperparameter. The main component for implementing PairNorm is to compute the mean and standard deviation of node features. For the mean of node features, this can be exactly computed in VN. For standard deviation, VN can arbitrarily approximate it using the standard universality result of MLP [5]. If we further assume that the standard deviation is lower bounded by a constant, then MPNN + VN can arbitrarily approximate the PairNorm on the compact set.

Example 2: In [158] mean subtraction (same as the first step of PairNorm) is also introduced to reduce over-smoothing. As mean subtraction can be trivially implemented in MPNN + VN, arguments in [158] (with mean subtraction the revised power Iteration in GCN will lead to the Fiedler vector) can be carried over to MPNN + VN setting.

In summary, introducing VN allows MPNN to implement key components of [158, 166], we think this is one reason why we observe encouraging empirical performance gain of MPNN + VN.

3.7 On the Limitation of MPNN + VN

Although we showed that in the main part of this chapter, MPNN + VN of varying depth/width can approximate the self-attention of full/linear transformers, this does not imply that there is no difference in practice between MPNN + VN and GT. Our theoretical analysis mainly focuses on approximating self-attention without considering computational efficiency. In this section, we mention a few limitations of MPNN + VN compared to GT.

3.7.1 Representation Gap

The main limitation of deep MPNN + VN approximating full self-attention is that we require a quite strong assumption: we restrict the variability of node features in order to select one node feature to process each iteration. Such assumption is relaxed by employing stronger attention in MPNN + VN but is still quite strong.

For the large width case, the main limitation is the computational complexity: even though the self-attention layer requires $O(n^2)$ complexity, to approximate it in wide MPNN + VN framework, the complexity will become $O(n^d)$ where d is the dimension of node features.

We think such limitation shares a similarity with research in universal permutational invariant functions. Both DeepSets [162] and Relational Network [128] are universal permutational invariant architecture but there is still a representation gap between the two [169]. Under the restriction to analytic activation functions, one can construct a symmetric function acting on sets of size n with elements in dimension d, which can be efficiently approximated by the Relational Network, but provably requires width exponential in n and d for the DeepSets. We believe a similar representation gap also exists between GT and MPNN + VN and leave the characterization of functions lying in such gap as the future work.

3.7.2 On the Difficulty of Approximating Other Linear Transformers

In Section 3.3, we showed MPNN + VN of $O(1)$ width and depth can approximate the self-attention layer of one type of linear transformer, Performer. The literature on efficient transformers is vast [139] and we do not expect MPNN + VN can approximate many other efficient transformers. Here we sketch a few other linear transformers that are hard to approximate by MPNN + VN of constant depth and width.

Linformer [147] projects the $n \times d$ dimension keys and values to $k \times d$ suing additional projection layers, which in graph setting is equivalent to graph coarsening. As MPNN + VN still operates on the original graph, it fundamentally lacks the key component to approximate Linformer.

We consider various types of efficient transformers effectively generalize the virtual node trick. By first switching to a more expansive model and reducing the computational complexity later on, efficient transformers effectively explore a larger model design space than MPNN + VN, which always sticks to the linear complexity.

3.7.3 Difficulty of Representing SAN Type Attention

In SAN [86], different attentions are used conditional on whether an edge is presented in the graph or not, detailed below. One may wonder whether we can approximate such a framework in MPNN + VN.

In our proof of using MPNN + VN to approximate regular GT, we mainly work with Definition 16 where we do not use any gn-gn edges and therefore not leverage the graph topology. It is straightforward to use gn-gn edges and obtain the different message/update/aggregate functions for gn-gn edges non-gn-gn edges. Although we still achieve the similar goal of SAN to condition on the edge types, it turns out that we can not arbitrarily approximate SAN.

Without loss of generality, SAN uses two types of attention depending on whether two nodes are connected by the edge. Specifically,

$$\begin{aligned}
\hat{\boldsymbol{w}}_{ij}^{k,l} &= \begin{cases} \frac{\boldsymbol{Q}^{1,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{1,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{1,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\boldsymbol{Q}^{2,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{2,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{2,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{otherwise} \end{cases} \\
w_{ij}^{k,l} &= \begin{cases} \frac{1}{1+\gamma} \cdot \text{softmax}\left(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}\right) & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\gamma}{1+\gamma} \cdot \text{softmax}\left(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}\right) & \text{otherwise} \end{cases}
\end{aligned} \tag{3.7}$$

where $\circ$ denotes element-wise multiplication and $\boldsymbol{Q}^{1,k,l}, \boldsymbol{Q}^{2,k,l}, \boldsymbol{K}^{1,k,l}, \boldsymbol{K}^{2,k,l}, \boldsymbol{E}^{1,k,l}, \boldsymbol{E}^{2,k,l} \in$

$\mathbb{R}^{d_k \times d}$. $\gamma \in \mathbb{R}^+$is a hyperparameter that tunes the amount of bias towards full-graph attention, allowing flexibility of the model to different datasets and tasks where the necessity to capture long-range dependencies may vary.

To reduce the notation clutter, we remove the layer index l, and edge features, and also consider only one-attention head case (remove attention index k). The equation is then simplified to

$$\begin{gathered}
\hat{\boldsymbol{w}}_{ij} = \left\{ \begin{array}{lr} \frac{\boldsymbol{Q}^1 \boldsymbol{h}_i^l \circ \boldsymbol{K}^1 \boldsymbol{h}_j^l}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\boldsymbol{Q}^2 \boldsymbol{h}_i^l \circ \boldsymbol{K}^2 \boldsymbol{h}_j^l}{\sqrt{d_k}} & \text{otherwise} \end{array} \right\} \\
w_{ij} = \left\{ \begin{array}{lr} \frac{1}{1+\gamma} \cdot \operatorname{softmax}\left(\sum_d \hat{\boldsymbol{w}}_{ij}\right) & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\gamma}{1+\gamma} \cdot \operatorname{softmax}\left(\sum_d \hat{\boldsymbol{w}}_{ij}\right) & \text{otherwise} \end{array} \right\}
\end{gathered} \tag{3.8}$$

We will then show that Equation (3.8) can not be expressed (up to an arbitrary approximation error) in MPNN + VN framework. To simulate SAN type attention, our MPNN + VN framework will have to first simulate one type of attention for all edges, and then simulate the second type of attention between gn-gn edges by properly offset the contribution from the first attention. This seems impossible (although we do not have rigorous proof) as we cannot express the difference between two attention in the new attention mechanism.

3.8 Related Work

Virtual node in MPNN. The virtual node augments the graph with an additional node to facilitate the information exchange among all pairs of nodes. It is a heuristic proposed in [54] and has been observed to improve the performance in different tasks [68, 69]. Surprisingly, its theoretical properties have received little study. To the best of our knowledge, only a recent paper [72] analyzed the role of the virtual node in the link prediction setting in terms of 1) expressiveness of the learned link representation and 2) the potential impact on under-reaching and over-smoothing.

Graph transformer. Because of the great successes of Transformers in natural language processing (NLP) [142, 151] and recently in computer vision [39, 45, 101], there is great interest in extending transformers for graphs. One common belief of advantage of graph transformer over MPNN is its capacity in capturing long-range interactions while alleviating over-smoothing [96, 114, 21] and over-squashing in MPNN [3, 140].

Fully-connected Graph transformer [43] was introduced with eigenvectors of graph Laplacian as the node positional encoding (PE). Various follow-up works proposed different ways of PE to improve GT, ranging from an invariant aggregation of Laplacian's eigenvectors in SAN [86], pair-wise graph distances in Graphormer [159], relative PE derived from diffusion kernels in GraphiT [112], and recently Sign and Basis Net [100] with a principled way of handling sign and basis invariance. Other lines of research in GT include combining MPNN and GT [153, 120], encoding the substructures [25] and efficient graph transformers for large graphs [152].

Deep Learning on Sets. Janossy pooling [113] is a framework to build permutation invariant architecture for sets using permuting & averaging paradigm while limiting the number of elements in permutations to be $k < n$. Under this framework, DeepSets [162] and PointNet [119] are recovered as the case of $k = 1$. For case $k = 2$, self-attention and Relation Network [128] are recovered [145]. Although DeepSets and Relation Network [128] are both shown to be universal permutation invariant, recent work [169] provides a finer characterization on the representation gap between the two architectures.

3.9 Concluding Remarks

in this chapter, we study the expressive power of MPNN + VN under the lens of GT. If we target the self-attention layer in Performer and Linear Transformer, one only needs $O(1)$-depth $O(1)$ width for arbitrary approximation error. For self-attention in full transformer, we prove that heterogeneous MPNN + VN of either $O(1)$ depth $O(n^d)$ width or $O(n)$ depth $O(1)$ width (under some assumptions) can approximate 1 self-attention layer arbitrarily well. Compared to early results [82] showing GT can approximate MPNN, our theoretical result draws the connection from the inverse direction.

On the empirical side, we demonstrate that MPNN + VN remains a surprisingly strong baseline. Despite recent efforts, we still lack good benchmark datasets where GT can outperform MPNN by a large margin. Understanding the inductive bias of MPNN and GT remains challenging. For example, can we mathematically characterize tasks that require effective long-range interaction modeling, and provide a theoretical justification for using GT over MPNN (or vice versa) for certain classes of functions on the space of graphs? We believe making processes towards answering such questions is an important future direction for the graph learning community.

3.10 Missing Proofs

In this section, we show the missing proofs of $O(n)$ heterogeneous MPNN + VN Layer with $O(1)$ width can approximate 1 self attention layer arbitrarily well.

3.10.1 Assumptions

A special case of $(\boldsymbol{V},\delta)$ separable is when $\delta = 0$, i.e., $\forall i, \bar{\alpha}(\boldsymbol{x}_i,\boldsymbol{v}_i) > \max_{j\neq i}\bar{\alpha}(\boldsymbol{x}_j,\boldsymbol{v}_i)$. We provide a geometric characterization of $\boldsymbol{X}$ being $(\boldsymbol{V},0)$ separable.

Lemma 3.10.1. *Given $\bar{\alpha}$ and $\boldsymbol{V}$, $\boldsymbol{X}$ is $(\boldsymbol{V},0)$ separable by $\bar{\alpha} \iff \boldsymbol{x}_i$ is not in the convex hull spanned by $\{\boldsymbol{x}_j\}_{j\neq i}$. $\iff$ there are no points in the convex hull of $\{\boldsymbol{x}_i\}_{i\in[n]}$.*

Proof. The second equivalence is trivial so we only prove the first equivalence. By definition, $\boldsymbol{X}$ is $(\boldsymbol{V},0)$ separable by $\bar{\alpha} \iff \bar{\alpha}(\boldsymbol{x}_i,\boldsymbol{v}_i) > \max_{j\neq i}\bar{\alpha}(\boldsymbol{x}_j,\boldsymbol{v}_i)\forall i\in[n] \iff \langle \boldsymbol{x}_i, \boldsymbol{W}_Q^{\bar{\alpha}}\boldsymbol{W}_K^{\bar{\alpha},T}\boldsymbol{v}_i\rangle > \max_{j\neq i}\langle \boldsymbol{x}_j, \boldsymbol{W}_Q^{\bar{\alpha}}\boldsymbol{W}_K^{\bar{\alpha},T}\boldsymbol{v}_i\rangle\forall i\in[n]$.

By denoting the $\boldsymbol{v}_i' := \boldsymbol{W}_Q^{\bar{\alpha}}\boldsymbol{W}_K^{\bar{\alpha},T}\boldsymbol{v}_i \in \mathbb{R}^d$, we know that $\langle \boldsymbol{x}_i,\boldsymbol{v}_i'\rangle > \max_{j\neq i}\langle \boldsymbol{x}_j,\boldsymbol{v}_i'\rangle \forall i\in[n]$, which implies that $\forall i\in[n], \boldsymbol{x}_i$ can be linearly seprated from $\{\boldsymbol{x}_j\}_{j\neq i} \iff \boldsymbol{x}_i$ is not in the convex hull spanned by $\{\boldsymbol{x}_j\}_{j\neq i}$, which concludes the proof. $\square$

Lemma 3.10.2 (approximate hard selection)**.** *Given $\mathscr{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d}$, $\bar{\alpha} \in \mathscr{A}$ and $\delta > 0$, the following holds. For any $\varepsilon > 0$ and $i \in [n]$, there exists a set of attention weights $\boldsymbol{W}_{i,Q}, \boldsymbol{W}_{i,K}$ in i-th layer of MPNN + VN such that $\alpha_{\boldsymbol{n}}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1-\varepsilon$ for any $\boldsymbol{x}_i \in \mathscr{X}_i$. In other words, we can approximate a hard selection function $f_i(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \boldsymbol{x}_i$ arbitrarily well on $\mathscr{X}$ by setting $\alpha_{\boldsymbol{n}} = \bar{\alpha}$.*

Proof. Denote $\bar{\alpha}'$ as the unnormalized $\bar{\alpha}$. As $\mathscr{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$, by definition we know that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j\neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i) + \delta$ holds for any $i \in [n]$ and $\boldsymbol{x}_i \in \mathscr{M}$. We can amplify this by multiple the weight matrix in $\bar{\alpha}$ by a constant factor c to make $\bar{\alpha}'(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j\neq i} \bar{\alpha}'(\boldsymbol{x}_j, \boldsymbol{v}_i) + c\delta$. This implies that $e^{\bar{\alpha}'(\boldsymbol{x}_i, \boldsymbol{v}_i)} > e^{c\delta} \max_{j\neq i} e^{\bar{\alpha}'(\boldsymbol{x}_j, \boldsymbol{v}_i)}$. This means after softmax, the attention score $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i)$ will be at least $\frac{e^{c\delta}}{e^{c\delta}+n-1}$. We can pick a large enough $c(\delta, \varepsilon)$ such that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1-\varepsilon$ for any $\boldsymbol{x}_i \in \mathscr{X}_i$ and $\varepsilon > 0$. $\square$

Proof Intuition and Outline. On the high level, i-th MPNN + VN layer will select $\tilde{\boldsymbol{x}}_i$, an approximation i-th node feature $\boldsymbol{x}_i$ via attention mechanism, enabled by Theorem 3.5.1, and send $\tilde{\boldsymbol{x}}_i$ to the virtual node. Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes and computes the approximation of $e^{\alpha(\boldsymbol{x}_i, \boldsymbol{x}_j)}, \forall j \in [n]$. Repeat such procedures n times for all graph nodes, and finally, use the last layer for attention normalization.

The main challenge of the proof is to 1) come up with message/update/aggregation functions for heterogeneous MPNN + VN layer, which is shown in Section 3.10.2, and 2) ensure the approximation error, both from approximating Aggregate/Message/Update function with MLP and the noisy input, can be well controlled, which is proved in Section 3.10.4.

We will first instantiate the Aggregate/Message/Update function for virtual/graph nodes in Section 3.10.2, and prove that each component can be either exactly computed or approximated to an arbitrary degree by MLP. Then we go through an example in Section 3.10.3 of approximate self-attention layer $\boldsymbol{L}$ with $O(n)$ MPNN + VN layers. The main proof is presented in Section 3.10.4, where we show that the approximation error introduced during different steps is well controlled.

Lastly, in Section 3.10.5 we show assumption on node features can be relaxed if a more powerful attention mechanism `GATv2` [12] is allowed in MPNN + VN.

3.10.2 Aggregate/Message/Update Functions

Let $\mathscr{M}$ be a multiset of vectors in $\mathbb{R}^d$. The specific form of Aggregate/Message/Update for virtual and graph nodes are listed below. Note that ideal forms will be implemented as MLP, which will incur an approximation error that can be controlled to an arbitrary degree. We use $\boldsymbol{z}_{\boldsymbol{n}}^{(k)}$ denotes the virtual node's feature at l-th layer, and $\boldsymbol{z}_i^{(k)}$ denotes the graph node i's node feature. Iteration index k starts with 0 and the node index starts with 1.

virtual node

At k-th iteration, virtual node i's feature $\boldsymbol{z}_i^{(k)}$ is a concatenation of three component $[\tilde{\boldsymbol{x}}_i, \boldsymbol{v}_{k+1}, 0]$ where the first component is the approximately selected node features $\boldsymbol{x}_i \in \mathbb{R}^d$, the second component is the $\boldsymbol{v}_i \in \mathbb{R}^d$ that is used to select the node feature in i-th iteration. The last component is just a placeholder to ensure the dimension of the virtual node and graph node are the same. It is introduced to simplify notation.

Initial feature is $\boldsymbol{z}_{\boldsymbol{n}}^{(0)} = [\boldsymbol{0}_d, \boldsymbol{v}_1, 0]$.

Message function + Aggregation function $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)} : \mathbb{R}^{2d+1} \times \mathscr{M} \to \mathbb{R}^{2d+1}$ has two cases to discuss depending on value of k. For $k = 1,2,...,n$,

$$\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}(\boldsymbol{z}_{\boldsymbol{n}}^{(k-1)}, \{\boldsymbol{z}_i^{(k-1)}\}_i) = \begin{cases} \sum_i \alpha_{\boldsymbol{n}}(\boldsymbol{z}_{\boldsymbol{n}}^{(k-1)}, \boldsymbol{z}_i^{(k-1)})\boldsymbol{z}_i^{(k-1)} & k = 1,2,...,n \\ \boldsymbol{1}_{2d+1} & k = n+1, n+2 \end{cases} \tag{3.9}$$

where $\boldsymbol{z}_{\boldsymbol{n}}^{(k-1)} = [\tilde{\boldsymbol{x}}_{k-1}, \boldsymbol{v}_k, 0]$. $\boldsymbol{z}_i^{(k-1)} = [\overbrace{\underbrace{\boldsymbol{x}_i}_{d \text{ dim}}, ..., ...}^{2d+1 \text{ dim}}]$ is the node i's feature, where the first d coordinates remain fixed for different iteration k. $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}$ use attention $\alpha_{\boldsymbol{n}}$ to approximately select k-th node feature $[\overbrace{\underbrace{\boldsymbol{x}_k}_{d \text{ dim}}, ..., ...}^{2d+1 \text{ dim}}]$. Note that the particular form of attention $\alpha_{\boldsymbol{n}}$ needed for soft selection is not important as long as we can approximate hard selection arbitrarily well. As the $\boldsymbol{z}_{\boldsymbol{n}}^{(k-1)}$ contains $\boldsymbol{v}_k$ and $\boldsymbol{z}_i^{(k-1)}$ contains $\boldsymbol{x}_i$ (see definition of graph node feature in Section 3.10.2), this step can be made as close to hard selection as possible, according to Theorem 3.10.5.

In the case of $k = n+1$, $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)} : \underbrace{\mathbb{R}^{2d+1}}_{\boldsymbol{n}} \times \underbrace{\mathscr{M}}_{\text{set of gn}} \to \mathbb{R}^d$ simply returns $\mathbf{1}_{2d+1}$. This can be exactly implemented by an MLP.

Update function $\gamma_{\boldsymbol{n}}^{(k)} : \underbrace{\mathbb{R}^{2d+1}}_{\boldsymbol{n}} \times \underbrace{\mathbb{R}^{2d+1}}_{gn} \to \mathbb{R}^{2d+1}$: Given the virtual node's feature in the last iteration, and the selected feature in virtual node $\boldsymbol{y} = [\boldsymbol{x}_k, ..., ...]$ with $\alpha_{\boldsymbol{n}}$,

$$\gamma_{\boldsymbol{n}}^{(k)}(\cdot, \boldsymbol{y}) = \begin{cases} [\boldsymbol{y}_{0:d}, \boldsymbol{v}_{k+1}, 0] & k = 1, ..., n-1 \\ [\boldsymbol{y}_{0:d}, \mathbf{0}_d, 0] & k = n \\ \mathbf{1}_{2d+1} & k = n+1, n+2 \end{cases} \tag{3.10}$$

where $\boldsymbol{y}_{0:d}$ denotes the first d channels of $\boldsymbol{y} \in \mathbb{R}^{2d+1}$. $\boldsymbol{y}$ denotes the selected node $\boldsymbol{z}_i$'s feature in Message/Aggregation function. $\gamma_{\boldsymbol{n}}^{(k)}$ can be exactly implemented by an MLP for any $k = 1, ..., n+2$.

Graph node

Graph node i's feature $\boldsymbol{v}_i \in \mathbb{R}^{2d+1}$ can be thought of as a concatenation of three components $[\underbrace{\boldsymbol{x}_i}_{d \text{ dim}}, \underbrace{\mathsf{tmp}}_{d \text{ dim}}, \underbrace{\mathsf{partialsum}}_{1 \text{ dim}}]$, where $\boldsymbol{x}_i, \in \mathbb{R}^d, \mathsf{tmp} \in \mathbb{R}^d$ [3], and $\mathsf{partialsum} \in \mathbb{R}$.

In particular, $\boldsymbol{x}_i$ is the initial node feature. The first d channel will stay the same until the layer $n+2$. $\mathsf{tmp} = \sum_{j\in\text{subset of}[n]} e^{\alpha'_{ij}} \boldsymbol{x}_j$ stands for the unnormalized attention contribution up to the current iteration. $\mathsf{partialsum} \in \mathbb{R}$ is a partial sum of the unnormalized attention score, which will be used for normalization in the $n+2$-th iteration.

Initial feature $\boldsymbol{z}_{\text{gn}}^{(0)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$.

[3] tmp technicially denotes the dimension of projected feature by W_V and does not has to be in $\mathbb{R}^d$. We use $\mathbb{R}^d$ here to reduce the notation clutter.

Message function + Aggregate function: $\tau_{j\in[n]}\phi^{(k)}_{gn\text{-}vn}:\mathbb{R}^{2d+1}\times\mathbb{R}^{2d+1}\to\mathbb{R}^{2d+1}$ is just "copying the second argument" since there is just one incoming message from the virtual node, i.e., $\tau_{j\in[n]}\phi^{(k)}_{\text{gn-vn}}(\boldsymbol{x},\{\boldsymbol{y}\})=\boldsymbol{y}$. This function can be exactly implemented by an MLP.

Update function $\gamma^{(k)}_{\text{gn}}:\underbrace{\mathbb{R}^{2d+1}}_{\text{gn}}\times\underbrace{\mathbb{R}^{2d+1}}_{\boldsymbol{n}}\to\mathbb{R}^{2d+1}$ is of the following form.

$$\gamma^{(k)}_{\text{gn}}([\boldsymbol{x},\mathsf{tmp},\mathsf{partialsum}],\boldsymbol{y})=\begin{cases}[\boldsymbol{x},\mathsf{tmp},\mathsf{partialsum}] & k=1\\ [\boldsymbol{x},\mathsf{tmp}+e^{\alpha'(\boldsymbol{x},\boldsymbol{y}_{0:d})}\boldsymbol{W}_V\boldsymbol{y}_{0:d}, & \\ \mathsf{partialsum}+e^{\alpha'(\boldsymbol{x},\boldsymbol{y}_{0:d})}] & k=2,...,n+1\\ [\frac{\mathsf{tmp}}{\mathsf{partialsum}},\mathbf{0}_d,0] & k=n+2\end{cases}\tag{3.11}$$

where $\alpha'(\boldsymbol{x},\boldsymbol{y}_{0:d})$ is the usual unnormalized attention score. Update function $\gamma^{(k)}_{\text{gn}}$ can be arbitrarily approximated by an MLP, which is proved below.

Lemma 3.10.3. *Update function $\gamma^{(k)}_{gn}$ can be arbitrarily approximated by an MLP from $\mathbb{R}^{2d+1}\times\mathbb{R}^{2d+1}$ to $\mathbb{R}^{2d+1}$ for all $k=1,...,n+2$.*

Proof. We will show that for any $k=1,...,n+2$, the target function $\gamma^{(k)}_{\text{gn}}:\mathbb{R}^{2d+1}\times\mathbb{R}^{2d+1}\to\mathbb{R}^{2d+1}$ is continuous and the domain is compact. By the universality of MLP in approximating continuous function on the compact domain, we know $\gamma^{(k)}_{\text{gn}}$ can be approximated to arbitrary precision by an MLP.

Recall that

$$\gamma_{\text{gn}}^{(k)}([\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}], \boldsymbol{y}) =$$
$$\begin{cases} [\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}] & k = 1 \\ [\boldsymbol{x}, \mathsf{tmp} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})} \boldsymbol{W}_V \boldsymbol{y}_{0:d}, & \\ \mathsf{partialsum} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})}] & k = 2, \ldots, n+1 \\ [\frac{\mathsf{tmp}}{\mathsf{partialsum}}, \boldsymbol{0}_d, 0] & k = n+2 \end{cases}$$

it is easy to see that $k = 1$, $\gamma_{\text{gn}}^{(1)}$ is continuous. We next show for $k = 2, \ldots, n+2$, $\gamma_{\text{gn}}^{(1)}$ is also continuous and all arguments lie in a compact domain.

$\gamma_{\text{gn}}^{(k)}$ is continuous because to a) $\alpha'(\boldsymbol{x}, \boldsymbol{y})$ is continuous b) scalar-vector multiplication, sum, and exponential are all continuous. Next, we show that four component $\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}$, $\boldsymbol{y}_{0:d}$ all lies in a compact domain.

$\boldsymbol{x}$ is the initial node features, and by AS1 their norm is bounded so $\boldsymbol{x}$ is in a compact domain.

tmp is an approximation of $e^{\alpha'_{i,1}} \boldsymbol{W}_V \boldsymbol{x}_1 + e^{\alpha'_{i,2}} \boldsymbol{W}_V \boldsymbol{x}_2 + \ldots$. As $\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is both upper and lower bounded by AS2 for all $i, j \in [n]$ and $\boldsymbol{x}_i$ is bounded by AS1, $e^{\alpha'_{i,1}} \boldsymbol{W}_V \boldsymbol{x}_1 + e^{\alpha'_{i,2}} \boldsymbol{W}_V \boldsymbol{x}_2 + \ldots$ is also bounded from below and above. tmp will also be bounded as we can control the error to any precision.

$\mathsf{partialsum}$ is an approximation of $e^{\alpha'_{i,1}} + e^{\alpha'_{i,2}} + \ldots$. For the same reason as the case above, $\mathsf{partialsum}$ is also bounded both below and above.

$\boldsymbol{y}_{0:d}$ will be $\tilde{\boldsymbol{x}}_i$ at i-th iteration so it will also be bounded by AS1.

Therefore we conclude the proof. $\square$

3.10.3 A Running Example

We provide an example to illustrate how node features are updated in each iteration.

Time 0: All nodes are initialized as indicated in Section 3.10.2. Virtual node feature $\boldsymbol{z}_{\boldsymbol{n}}^{(0)} = [\mathbf{0}_d, \boldsymbol{v}_1, 0]$. Graph node feature $\boldsymbol{z}_i^{(0)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$ for all $i \in [n]$.

Time 1:

For virtual node, according to the definition of $\tau_{j\in[n]}\phi^{(1)}_{\text{vn-gn}}$ in Equation (3.9), it will pick an approximation of $\boldsymbol{x}_1$, i.e. $\tilde{\boldsymbol{x}}_1$. Note that the approximation error can be made arbitrarily small. VN's node feature $\boldsymbol{z}_{\boldsymbol{n}}^{(1)} = [\tilde{\boldsymbol{x}}_1, \boldsymbol{v}_2, 0]$.

For i-th graph node feature, $\boldsymbol{z}_{\boldsymbol{n}}^{(0)} = \mathbf{1}_d$, and $\boldsymbol{z}_i^{(0)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$. According to $\gamma^{(k)}_{\text{gn}}$ in Equation (3.11), $\boldsymbol{z}_i^{(1)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$.

Time 2:

For the virtual node feature: similar to the analysis in time 1, VN's feature $\boldsymbol{z}_{\boldsymbol{n}}^{(2)} = [\tilde{\boldsymbol{x}}_2, \boldsymbol{v}_3, 0]$ now. Note that the weights and bias in $\tau_{j\in[n]}\phi^{(2)}_{\text{vn-gn}}$ will be different from those in $\tau_{j\in[n]}\phi^{(1)}_{\text{vn-gn}}$.

For i-th graph node feature, as $\boldsymbol{z}_{\boldsymbol{n}}^{(1)} = [\tilde{\boldsymbol{x}}_1, \boldsymbol{v}_2, 0]$ and $\boldsymbol{z}_i^{(1)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$, according to $\gamma^{(k)}_{\text{gn}}$ in Equation (3.11), $\boldsymbol{z}_i^{(2)} = [\boldsymbol{x}_i, \widetilde{e^{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1, \widetilde{e^{\alpha'_{i,1}}}]$. Here $\widetilde{\alpha'_{i,1}} := \alpha'(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_1)$. We will use similar notations in later iterations. [4]

Time 3:

Similar to the analysis above, $\boldsymbol{z}_{\boldsymbol{n}}^{(3)} = [\widetilde{\boldsymbol{x}_3}, \boldsymbol{v}_4, 0]$.

$\boldsymbol{z}_i^{(3)} = [\boldsymbol{x}_i, \widetilde{e^{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1 + \widetilde{e^{\alpha'_{i,2}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_2, \widetilde{e^{\alpha'_{i,1}}} + \widetilde{e^{\alpha'_{i,2}}}]$.

Time n:

$\boldsymbol{z}_{\boldsymbol{n}}^{(n)} = [\tilde{\boldsymbol{x}}_n, \mathbf{0}_d, 0]$.

$$\boldsymbol{z}_i^{(n)} = \boldsymbol{x}_i, \underbrace{\widetilde{e^{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1 + \ldots + \widetilde{e^{\alpha'_{i,n-1}}}\boldsymbol{W}_V\widetilde{\boldsymbol{x}_{n-1}}}_{n-1 \text{ terms}},$$
$$\underbrace{\widetilde{e^{\alpha'_{i,1}}} + \widetilde{e^{\alpha'_{i,2}}} + \ldots + \widetilde{e^{\alpha'_{i,n-1}}}}_{n-1 \text{ terms}}].$$

Time $n+1$:

According to Section 3.10.2, in $n+1$ iteration, the virtual node's feature will be $\mathbf{1}_d$.

[4]To reduce the notation clutter and provide an intuition of the proof, we omit the approximation error introduced by using MLP to approximate aggregation/message/update function, and assume the aggregation/message/update can be exactly implemented by neural networks. In the proofs, approximation error by MLP is handled rigorously.

$\boldsymbol{z}_i^{(n+1)} = [\boldsymbol{x}_i, \sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k, \sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}]$

Time $n+2$ (final layer):

For the virtual node, its node feature will stay the same.

For the graph node feature, the last layer will serve as a normalization of the attention score (use MLP to approximate vector-scalar multiplication), and set the last channel to be 0 (projection), resulting in an approximation of $[\boldsymbol{x}_i, \frac{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k}{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}}, 0]$. Finally, we need one more linear transformation to make the node feature become $[\frac{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k}{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}}, \mathbf{0}_d, 0]$. The first d channel is an approximation of the output of the self-attention layer for node i where the approximation error can be made as small as possible. This is proved in Section 3.10, and we conclude that heterogeneous MPNN + VN can approximate the self-attention layer $\boldsymbol{L}$ to arbitrary precision with $O(n)$ MPNN layers.

3.10.4 Controlling Error

On the high level, there are three major sources of approximation error: 1) approximate hard selection with self-attention and 2) approximate equation $\gamma_{\text{gn}}^{(k)}$ with MLPs, and 3) attention normalization in the last layer. In all cases, we aim to approximate the output of a continuous map $\boldsymbol{L}_c(\boldsymbol{x})$. However, our input is usually not exact $\boldsymbol{x}$ but an approximation of $\tilde{\boldsymbol{x}}$. We also cannot access the original map $\boldsymbol{L}_c$ but instead, an MLP approximation of $\boldsymbol{L}_c$, denoted as $\boldsymbol{L}_{\text{MLP}}$. The following lemma allows to control the difference between $\boldsymbol{L}_c(\boldsymbol{x})$ and $\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})$.

Lemma 3.10.4. *Let $\boldsymbol{L}_c$ be a continuous map from compact set to compact set in Euclidean space. Let $\boldsymbol{L}_{\text{MLP}}$ be the approximation of $\boldsymbol{L}_c$ by MLP. If we can control $\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|$ to an arbitrarily small degree, we can then control the error $\|\boldsymbol{L}_c(\boldsymbol{x})-\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$ arbitrarily small.*

Proof. By triangle inequality $\|\boldsymbol{L}_c(\boldsymbol{x})-\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\| \leq \|\boldsymbol{L}_c(\boldsymbol{x})-\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}))\| + \|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x})-\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$.

For the first term $\|\boldsymbol{L}_c(\tilde{\boldsymbol{x}})-\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$, by the universality of MLP, we can control the error $\|\boldsymbol{L}_c(\tilde{\boldsymbol{x}})-\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$ in arbitrary degree.

For the second term $\|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$, as $\boldsymbol{L}_{\text{MLP}}$ is continuous on a compact domain, it is uniformly continuous by Heine-Cantor theorem. This means that we can control the $\|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$ as long as we can control $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$, independent from different $\boldsymbol{x}$. By assumption, this is indeed the case so we conclude the proof. □

Remark 9. *The implication is that when we are trying to approximate the output of a continuous map $\boldsymbol{L}_c$ on the compact domain by an MLP $\boldsymbol{L}_{\text{MLP}}$, it suffices to show the input is 1) $\|\boldsymbol{L}_c - \boldsymbol{L}_{\text{MLP}}\|_\infty$ and 2) $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|$ can be made arbitrarily small. The first point is usually done by the universality of MLP on the compact domain [31]. The second point needs to be shown case by case.*

In the Section 3.10.3, to simplify the notations we omit the error introduced by using MLP to approximate aggregation/message/update functions (continuous functions on the compact domain of $\mathbb{R}^d$.) in MPNN + VN. Theorem 3.10.4 justify such reasoning.

Lemma 3.10.5 ($\tilde{\boldsymbol{x}}_i$ approximates $\boldsymbol{x}_i$. $\widetilde{\alpha'_{i,j}}$ approximates $\alpha'_{i,j}$.)**.** *For any $\varepsilon > 0$ and $x \in \mathscr{X}$, there exist a set of weights for message/aggregate functions of the virtual node such that $||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i|| < \varepsilon$ and $|\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}| < \varepsilon$.*

Proof. By Theorem 3.5.1 We know that $\widetilde{\alpha_{i,j}} := \widetilde{\alpha}(\boldsymbol{x}_i, \boldsymbol{x}_j) \to \delta(i-j)$ as $C_3(\varepsilon)$ goes to infinity. Therefore we have

$$||\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i|| = ||\sum_j \widetilde{\alpha_{i,j}} \boldsymbol{x}_j - \boldsymbol{x}_i|| = ||\sum (\widetilde{\alpha}_{i,j} - \delta(i-j))\boldsymbol{x}_j|| < \varepsilon \sum ||\boldsymbol{x}_j|| < nC_1\varepsilon \tag{3.12}$$

As n and C_1 are fixed, we can make the upper bound as small as we want by increasing C_3.

$|\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}| = |\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j) - \alpha'_{\text{MLP}}(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = |\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j) - \alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| + |\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j) - \alpha'_{\text{MLP}}(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = |\alpha'(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = (\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i)^T \boldsymbol{x}_j C_2^2 + \varepsilon < nC_1\varepsilon C_1 C_2^2 + \varepsilon = (nC_1^2 C_2^2 + 1)\varepsilon$. As $\alpha'_{i,j}, \widetilde{\alpha'_{i,j}}$ is bounded from above and below, it's easy to see that $|e^{\alpha'_{i,j}} - e^{\widetilde{\alpha'_{i,j}}}| = |e^{\alpha'_{i,j}}(1 - e^{\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}})| < C(1 - e^{\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}})$ can be controlled to arbitrarily degree. □

Theorem 3.5.2. *Assume AS 1-3 hold for the compact set $\mathscr{X}$ and $\boldsymbol{L}$. Given any graph G of size n* *with node features $\boldsymbol{X} \in \mathscr{X}$, and a self-attention layer $\boldsymbol{L}$ on G (fix $\boldsymbol{W}_K, \boldsymbol{W}_Q, \boldsymbol{W}_V$ in α), there exists a $O(n)$ layer of heterogeneous MPNN + VN with the specific aggregate/update/message function that can approximate $\boldsymbol{L}$ on $\mathscr{X}$ arbitrarily well.*

Proof. i-th MPNN + VN layer will select $\tilde{\boldsymbol{x}}_i$, an arbitrary approximation i-th node feature $\boldsymbol{x}_i$ via attention mechanism. This is detailed in the message/aggregation function of the virtual node in Section 3.10.2. Assuming the regularity condition on feature space $\mathscr{X}$, detailed in AS3, the approximation error can be made as small as needed, as shown in Theorems 3.5.1 and 3.10.5.

Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes, which computes an approximation of $e^{\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)}, \forall j \in [n]$. This step is detailed in the update function $\gamma_{\text{gn}}^{(k)}$ of graph nodes, which can also be approximated arbitrarily well by MLP, proved in Theorem 3.10.3. By Theorem 3.10.4, we have an arbitrary approximation of $e^{\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)}, \forall j \in [n]$, which itself is an arbitrary approximation of $e^{\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j)}, \forall j \in [n]$.

Repeat such procedures n times for all graph nodes, we have an arbitrary approximation of $\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k \in \mathbb{R}^d$ and $\sum_{k\in[n]} e^{\alpha'_{ik}} \in \mathbb{R}$. Finally, we use the last layer to approximate attention normalization $L_c(\boldsymbol{x}, y) = \frac{\boldsymbol{x}}{y}$, where $\boldsymbol{x} \in \mathbb{R}^d, y \in \mathbb{R}$. As inputs for attention normalization are arbitrary approximation of $\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k$ and $\sum_{k\in[n]} e^{\alpha'_{ik}}$, both of them are lower/upper bounded according to AS1 and AS2. Since the denominator is upper bounded by a positive number, this implies that the target function L_c is continuous in both arguments. By evoking Theorem 3.10.4 again, we conclude that we can approximate its output $\frac{\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k}{\sum_{k\in[n]} e^{\alpha'_{ik}}}$ arbitrarily well. This concludes the proof.

□

3.10.5 Relaxing Assumptions with More Powerful Attention

One limitation of Theorem 3.5.2 are assumptions on node features space $\mathscr{X}$: we need to 1) restrict the variability of node feature so that we can select one node feature to process each iteration. 2) The space of the node feature also need to satisfy certain configuration in order for

VN to select it. For 2), we now consider a different attention function for α_n in MPNN + VN that can relax the assumptions AS3 on $\mathscr{X}$.

More powerful attention mechanism. From proof of Theorem 3.5.2, we just need $\alpha(\cdot,\cdot)$ uniformly select every node in $\boldsymbol{X} \in \mathscr{X}$. The unnormalized bilinear attention α' is weak in the sense that $f(\cdot) = \langle \boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^T, \cdot \rangle$ has a linear level set. Such a constraint can be relaxed via an improved attention module `GATv2`. Observing the ranking of the attention scores given by `GAT` [143] is unconditioned on the query node, [12] proposed `GATv2`, a more expressive attention mechanism. In particular, the unnormalized attention score $\alpha'_{\text{GATv2}}(\boldsymbol{u},\boldsymbol{v}) := \boldsymbol{a}^T \,\text{LeakyReLU}\,(\boldsymbol{W} \cdot [\boldsymbol{u} \| \boldsymbol{v}] + \boldsymbol{b})$, where $[\cdot||\cdot]$ is concatenation. We will let $\alpha_n = \alpha_{\text{GATv2}}$ to select features in $\tau_{j\in[n]}\phi^{(k)}_{\text{vn-gn}}$.

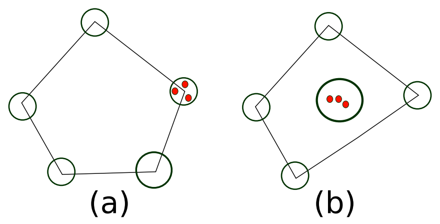


Figure 3.3. In the left figure, we have one example of $\mathscr{X}$ being $(\boldsymbol{V},\delta)$ separable, for which α can uniformly select any point (marked as red) $\boldsymbol{x}_i \in \mathscr{X}_i$. In the right figure, we change α_n in MPNN + VN to α_{GATv2}, which allows us to select more diverse feature configurations.

Lemma 3.10.6. *$\alpha'_{GATv2}(\cdot,\cdot)$ can approximate any continuous function from $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. For any $\boldsymbol{v} \in \mathbb{R}^d$, a restriction of $\alpha'_{GATv2}(\cdot,\boldsymbol{v})$ can approximate any continuous function from $\mathbb{R}^d \to \mathbb{R}$.*

Proof. Any function continuous in both arguments of α'_{GATv2} is also continuous in the concatenation of both arguments. As any continuous functions in $\mathbb{R}^{2d}$ can be approximated by α'_{GATv2} on a compact domain according to the universality of MLP [31], we finish the proof for the first statement.

For the second statement, we can write W as 2×2 block matrix and restrict it to cases where only $\boldsymbol{W}_{11}$ is non-zero. Then we have

$$\alpha'_{\text{GATv2}}(\boldsymbol{u},\boldsymbol{v}) = a^T \,\text{LeakyReLU}\left(\begin{bmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{W}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} + \boldsymbol{b} \right) = \boldsymbol{a}^T \,\text{LeakyReLU}\,(\boldsymbol{W}_{11}\boldsymbol{u} + \boldsymbol{b}) \tag{3.13}$$

which gives us an MLP on the first argument $\boldsymbol{u}$. By the universality of MLP, we conclude the proof for the second statement.

$\square$

Definition 21. *Given $\delta > 0$, We call $\mathscr{X}$ is δ nonlinearly separable if and only if* $\min_{i \neq j} d(\mathscr{X}_i, \mathscr{X}_j) > \delta$.

AS4. *$\mathscr{X}$ is δ nonlinearly separable for some $\delta > 0$.*

Proposition 3.10.7. *If $\mathscr{X} \subset \mathbb{R}^{n \times d}$ satisfies that $\mathscr{X}_i$ is δ-separated from $\mathscr{X}_j$ for any $i, j \in [n]$, the following holds. For any $\boldsymbol{X} \in \mathscr{X}$ and $i \in [n]$, there exist a α_{GATv2} to select any $\boldsymbol{x}_i \in \mathscr{X}_i$. This implies that we can arbitrarily approximate the self-attention layer $\boldsymbol{L}$ after relaxing AS3 to AS3'.*

Proof. For any $i \in [n]$, as $\mathscr{X}_i$ is δ-separated from other $\mathscr{X}_j, \forall j \neq i$, we can draw a region $\Omega_i \subset \mathbb{R}^d$ that contains $\mathscr{X}_i$ and separate $\mathscr{X}_i$ from other $\mathscr{X}_j (j \neq i)$, where the distance from $\mathscr{X}_i$ from other $\mathscr{X}_j$ is at least δ according to the definition of Definition 21. Next, we show how to construct a continuous function f whose value in $\mathscr{X}_i$ is at least 1 larger than its values in any other $\mathscr{X}_j$ $\forall j \neq i$.

We set the values of f in $\mathscr{X}_i$ to be 1.5 and values of f in $\mathscr{X}_j, \forall j \neq i$ to be 0. We can then interpolate f in areas outside of $\cup \mathscr{X}_i$ (one way is to set the values of $f(x)$ based on $d(x, \mathscr{X}_i)$, which results in a continuous function that satisfies our requirement. By the universality of α_{GATv2}, we can approximate f to arbitrary precision, and this will let us select any $\mathscr{X}_i$. $\square$

Chapter 4

Graph Coarsening with Neural Networks

4.1 Introduction

In this chapter, we look into another common way of global modeling for large graphs. As large scale-graphs become increasingly ubiquitous in various applications, they pose significant computational challenges to process, extract and analyze information. It is therefore natural to look for ways to simplify the graph while preserving the properties of interest.

There are two major ways to simplify graphs. First, one may reduce the number of edges, known as graph edge sparsification. It is known that pairwise distance (spanner), graph cut (cut sparsifier), eigenvalues (spectral sparsifier) can be approximately maintained via removing edges. A key result [137] in the spectral sparsification is that any dense graph of size N can be sparsified to $O(Nlog^cN/\varepsilon^2)$ edges in nearly linear time using a simple randomized algorithm based on the effective resistance.

Alternatively, one could also reduce the number of nodes to a subset of the original node set. The first challenge here is how to choose the topology (edge set) of the smaller graph spanned by the sparsified node set. On the extreme, one can take the complete graph spanned by the sampled nodes. However, its dense structure prohibits easy interpretation and poses computational overhead for setting the $\Theta(n^2)$ weights of edges. this chapter focuses on *graph coarsening*, which reduces the number of nodes by contracting disjoint sets of connected vertices.

The original idea dates back to the algebraic multigrid literature [123] and has found various applications in graph partitioning [61, 76, 89], visualization [60, 70, 146] and machine learning [90, 52, 134].

However, most existing graph coarsening algorithms come with two restrictions. First, they are *prespecified* and not adapted to specific data nor different goals.

Second, most coarsening algorithms set the edge weights of the coarse graph equal to the sum of weights of crossing edges in the original graph. This means the weights of the coarse graph is determined by the coarsening algorithm (of the vertex set), leaving no room for adjustment.

With the two observations above, we aim to develop a data-driven approach to better assigning weights for the coarse graph depending on specific goals at hand.

We will leverage the recent progress of deep learning on graphs to develop a framework to learn to assign edge weights *in an unsupervised manner* from a collection of input (small) graphs. This learned weight-assignment map can then be applied to new graphs (of potentially much larger sizes).

In particular, our contributions are threefold.

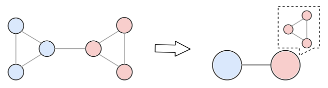
- First, depending on the quantity of interest $\mathscr{F}$ (such as the quadratic form w.r.t. Laplace operator), one has to carefully choose projection/lift operator to relate quantities defined on graphs of different sizes. We formulate this as the invariance of $\mathscr{F}$ under lift map, and provide three cases of projection/lift map as well as the corresponding operators on the coarse graph. Interestingly, those operators all can be seen as the special cases of doubly-weighted Laplace operators on coarse graphs [66].

- Second, we are the first to propose and develop a framework to learn the edge weights of the coarse graphs via graph neural networks (GNN) in an unsupervised manner. We show convincing results both theoretically and empirically that changing the weights is crucial to improve the quality of coarse graphs. many existing graph coarsening algorithms.

- Third, through extensive experiments on both synthetic graphs and real networks, we demonstrate that our method GOREN significantly improves common graph coarsening methods under different evaluation metrics, reduction ratios, graph sizes, and graph types. It generalizes to graphs of larger size (than the training graphs), adapts to different losses (so as to preserve different properties of original graphs), and scales to much larger graphs than what previous work can handle. Even for losses that are not differentiable w.r.t the weights of the coarse graph, we show training networks with a differentiable auxiliary loss still improves the result.

4.2 Proposed Approach: Learning Edge Weight with GNN

4.2.1 High-level overview



Our input is a non-attributed (weighted or unweighted) graph $G = (V, E)$. Our goal is to construct an appropriate "coarser" graph $\widehat{G} = (\widehat{V}, \widehat{E})$ that preserves certain properties of G. Here, by a "coarser" graph, we assume that $|\widehat{V}| << |V|$ and there is a surjective map $\pi : V \to \widehat{V}$ that we call the *vertex map*. Intuitively, (see figure on the right), for any node $\hat{v} \in \widehat{V}$, all nodes $\pi^{-1}(\hat{v}) \subset V$ are mapped to this *super-node* $\hat{v}$ in the coarser graph $\widehat{G}$. We will later propose a GNN based framework that can be trained using a collection of existing graphs *in an unsupervised manner*, so as to construct such a coarse graph $\widehat{G}$ for a future input graph G (presumably coming from the same family as training graphs) that can preserve properties of G effectively.

We will in particular focus on preserving properties of the *Laplace operator* $\mathcal{O}_G$ of G, which is by far the most common operator associated to graphs, and forms the foundation for spectral methods. Specifically, given $G = (V = \{v_1, \ldots, v_N\}, E)$ with $w : E \to \mathbb{R}$ being the weight function for G (all edges have weight 1 if G is unweighted), let W the corresponding $N \times N$ edge-weight matrix where $W[i][j] = w(v_i, v_j)$ if edge $(v_i, v_j) \in E$ and 0 otherwise. Set D to be the $N \times N$ diagonal matrix with $D[i][i]$ equal to the sum of weights of all edges incident to v_i. The standard *(un-normalized) combinatorial Laplace operator* of G is then defined as $L = D - W$. The *normalized Laplacian* is defined as $\mathcal{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$.

However, to make this problem as well as our proposed approach concrete, various components need to be built appropriately. We provide an overview here, and they will be detailed in the remainder of this section.

- Assuming that the set of super-nodes $\widehat{V}$ as well as the map $\pi : V \to \widehat{V}$ are given, one still need to decide how to set up the connectivity (i.e, edge set $\widehat{E}$) for the coarse graph $\widehat{G} = (\widehat{V}, \widehat{E})$. We introduce a natural choice in Section 4.2.2, and provide some justification for this choice.
- As the graph G and the coarse graph $\widehat{G}$ have the different number of nodes, their Laplace operators $\mathcal{O}_G$ and $\mathcal{O}_{\widehat{G}}$ of two graphs are not directly comparable. Instead, we will compare $\mathscr{F}(\mathcal{O}_G, f)$ and $\mathscr{F}(\mathcal{O}_{\widehat{G}}, \hat{f})$, where $\mathscr{F}$ is a functional intrinsic to the graph at hand (invariant to the permutation of vertices), such as the quadratic form or Rayleigh quotient. However, it turns out that depending on the choice of $\mathscr{F}$, we need to choose the precise form of the Laplacian $\mathcal{O}_{\widehat{G}}$, as well as the (so-called lifting and projection) maps relating these two objects, carefully, so as they are comparable. We describe these in detail in Section 4.2.3.
- In Section 4.2.4 we show that adjusting the weights of the coarse graph $\widehat{G}$ can significantly improve the quality of $\widehat{G}$. This motivates a learning approach to learn a strategy (a map) to assign these weights from a collection of given graphs. We then propose a GNN-based framework to do so in an unsupervised manner. Extensive experimental studies will be presented in Section 4.3.

4.2.2 Construction of Coarse graph

Assume that we are already given the set of super-nodes $\widehat{V} = \{\hat{v}_1, \ldots, \hat{v}_n\}$ for the coarse graph $\widehat{G}$ together with the vertex map $\pi : V \to \widehat{V}$ – There has been much prior work on computing the sparsified set $\widehat{V} \subset V$ and π [105, 104]; and if the vertex map π is not given, then we can simply define it by setting $\pi(v)$ for each $v \in V$ to be the nearest neighbor of v in $\widehat{V}$ in terms of graph shortest path distance in G [36].

To construct edges for the coarse graph $\widehat{G} = (\widehat{V}, \widehat{E})$ together with the edge weight function $\hat{w} : \widehat{E} \to \mathbb{R}$, instead of using a complete weighted graph over $\widehat{V}$, which is too dense and expensive, we set $\widehat{E}$ to be those edges “induced" from G when collapsing each *cluster* $\pi^{-1}(\hat{v})$ to its corresponding super-node $\hat{v} \in \widehat{V}$: Specifically, $(\hat{v}, \hat{v}') \in \widehat{E}$ if and only if there is an edge $(v, v') \in E$ such that $\pi(v) = \hat{v}$ and $\pi(v') = \hat{v}'$.

The weight of this edge is $\hat{w}(\hat{v}, \hat{v}') := \sum_{(v,v') \in E\left(\pi^{-1}(\hat{v}), \pi^{-1}(\hat{v}')\right)} w(v, v')$

where $E(A,B) \subseteq E$ stands for the set of edges crossing sets $A, B \subseteq V$;

i.e., $\hat{w}(\hat{v}, \hat{v}')$ is the total weights of all crossing edges in G between clusters $\pi^{-1}(\hat{v})$ and $\pi^{-1}(\hat{v}')$ in V. We refer to $\widehat{G}$ constructed this way the $\widehat{V}$*-induced coarse graph*.

As shown in [36], if the original graph G is the 1-skeleton of a hidden space X, then this induced graph captures the topological of X at a coarser level if $\widehat{V}$ is a so-called δ-net of the original vertex set V w.r.t. the graph shortest path metric.

Let $\widehat{W}$ be the edge weight matrix, and $\widehat{D}$ be the diagonal matrix encoding the sum of edge weights incident to each vertex as before. Then the standard combinatorial Laplace operator w.r.t. $\widehat{G}$ is simply $\widehat{L} = \widehat{D} - \widehat{W}$.

Relation to the operator of [104]. Interestingly, this construction of the coarse graph $\widehat{G}$ coincides with the coarse Laplace operator for a sparsified vertex set $\widehat{V}$ constructed by [104]. We will use this view of the Laplace operator later; hence we briefly introduce the construction of [104] (adapted to our setting):

Given the vertex map $\pi : V \to \widehat{V}$, we set a $n \times N$ matrix P by $P[r,i] = \begin{cases} \frac{1}{\left|\pi^{-1}(\hat{v}_r)\right|} & \text{if } v_i \in \pi^{-1}(\hat{v}_r) \\ 0 & \text{otherwise} \end{cases}$.

In what follows, we denote $\gamma_r := \left|\pi^{-1}(\hat{v}_r)\right|$ for any $r \in [1, n]$, which is the size of the cluster of $\hat{v}_r$ in V. P can be considered as the weighted projection matrix of the vertex set from V to $\widehat{V}$. Let P^+ denote the Moore-Penrose pseudoinverse of P, which can be intuitively viewed as a way to lift a function on $\widehat{V}$ (a vector in $\mathbb{R}^n$) to a function over V (a vector in $\mathbb{R}^N$). As shown in [104], P^+ is the $N \times n$ matrix where $P^+[i,r] = 1$ if and only if $\pi(v_i) = \hat{v}_r$. See Section 4.10.1 for a toy example.

Finally, [104] defines an operator for the coarsened vertex set $\widehat{V}$ to be $\tilde{L}_{\widehat{V}} = (P^+)^T L P^+$. Intuitively, $\widehat{L}$ operators on n-vectors. For any n-vector $\hat{f} \in \mathbb{R}^n$, $\tilde{L}_{\widehat{V}} \hat{f}$ first lifts $\hat{f}$ to a N-vector

$f = P^+ \hat{f}$, and then perform L on f, and then project it down to n-dimensional via $(P^+)^T$.

Proposition 4.2.1. *[104] The combinatorial graph Laplace operator $\widehat{L} = \widehat{D} - \widehat{W}$ for the $\widehat{V}$-induced coarse graph $\widehat{G}$ constructed above equals to the operator $\tilde{L}_{\widehat{V}} = (P^+)^T L P^+$.*

4.2.3 Laplace Operator for the Coarse Graph

We now have an input graph $G = (V, E)$ and a coarse graph $\widehat{G}$ induced from the sparsified node set $\widehat{V}$, and we wish to compare their corresponding Laplace operators. However, as $\mathcal{O}_G$ operates on $\mathbb{R}^N$ (i.e, functions on the vertex set V of G) and $\mathcal{O}_{\widehat{G}}$ operates on $\mathbb{R}^n$, we will compare them by their effects on "corresponding" objects.

[105, 104] proposed to use the quadratic form to measure the similarity between the two linear operators. In particular, given a linear operator A on $\mathbb{R}^N$ and any $x \in \mathbb{R}^N$, $\mathsf{Q}_A(x) = x^T A x$. The quadratic form has also been used for measuring spectral approximation under edge sparsification. The proof of the following result is in Section 4.10.1.

Proposition 4.2.2. *For any vector $\hat{x} \in \mathbb{R}^n$, we have that $\mathsf{Q}_{\widehat{L}}(\hat{x}) = \mathsf{Q}_L(P^+ \hat{x})$, where $\widehat{L}$ is the combinatorial Laplace operator for the $\widehat{V}$-induced coarse graph $\widehat{G}$ constructed above. That is, set $x := P^+ \hat{x}$ as the lift of $\hat{x}$ in $\mathbb{R}^N$, then $\hat{x}^T \widehat{L} \hat{x} = x^T L x$.*

Intuitively, this suggests that if later, we measure the similarity between L and some Laplace operator for the coarse graph $\widehat{G}$ based on a loss from quadratic form difference, then we should choose the Laplace operator $\mathcal{O}_{\widehat{G}}$ to be $\widehat{L}$ and compare $\mathsf{Q}_{\widehat{L}}(Px)$ with $\mathsf{Q}_L(x)$. We further formalize this by considering the *lifting map* $\mathcal{U} : \mathbb{R}^n \to \mathbb{R}^N$ as well as a projection map $\mathcal{P} : \mathbb{R}^N \to \mathbb{R}^n$, where $\mathcal{P} \cdot \mathcal{U} = Id_n$. Proposition 4.2.2 suggests that for quadratic form-based similarity, the choices are $\mathcal{U} = P^+, \mathcal{P} = P$, and $\mathcal{O}_{\widehat{G}} = \widehat{L}$. See the first row in Table 4.1.

On the other hand, eigenvectors and eigenvalues of a linear operator A are more directly related, via Courant-Fischer Min-Max Theorem, to its Rayleigh quotient $\mathsf{R}_A(x) = \frac{x^T A x}{x^T x}$. Interestingly, in this case, to preserve the Rayleigh quotient, we should change the choice of $\mathcal{O}_{\widehat{G}}$ to be the following *doubly-weighted Laplace operator* for a graph that is both edge and vertex weighted.

Table 4.1. Depending on the choice of $\mathscr{F}$ (quantity that we want to preserve) and $\mathscr{O}_G$, we have different projection/lift operators and resulting $\mathscr{O}_{\widehat{G}}$ on the coarse graph.

Quantity $\mathscr{F}$ of interest	$\mathscr{O}_G$	Projection $\mathscr{P}$	Lift $\mathscr{U}$	$\mathscr{O}_{\widehat{G}}$	Invariant under $\mathscr{U}$
Quadratic form Q	L	P	P^+	Combinatorial Laplace $\widehat{L}$	$\mathsf{Q}_L(\mathscr{U}\hat{x}) = \mathsf{Q}_{\widehat{L}}(\hat{x})$
Rayleigh quotient R	L	$\Gamma^{-1/2}(P^+)^T$	$P^+\Gamma^{-1/2}$	Doubly-weighted Laplace $\widehat{\mathsf{L}}$	$\mathsf{R}_L(\mathscr{U}\hat{x}) = \mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x})$
Quadratic form Q	$\mathscr{L}$	$\widehat{D}^{1/2}PD^{-1/2}$	$D^{1/2}(P^+)\widehat{D}^{-1/2}$	Normalized Laplace $\widehat{\mathscr{L}}$	$\mathsf{Q}_{\mathscr{L}}(\mathscr{U}\hat{x}) = \mathsf{Q}_{\widehat{\mathscr{L}}}(\hat{x})$

Specifically, for the coarse graph $\widehat{G}$, we assume that each vertex $\hat{v} \in \widehat{V}$ is weighted by $\gamma_{\hat{v}} := |\pi^{-1}(\hat{v})|$, the size of the cluster from G that got collapsed into $\hat{v}$. Let Γ be the vertex matrix, which is the $n \times n$ diagonal matrix with $\Gamma[r][r] = \gamma_{\hat{v}_r}$. The *doubly-weighted Laplace operator* for a vertex- and edge-weighted graph $\widehat{G}$ is then defined as:

$$\widehat{\mathsf{L}} = \Gamma^{-1/2}(\widehat{D} - \widehat{W})\Gamma^{-1/2} = \Gamma^{-1/2}\widehat{L}\Gamma^{-1/2} = (P^+\Gamma^{-1/2})^T L (P^+\Gamma^{-1/2}).$$

The concept of doubly-weighted Laplace for a vertex- and edge-weighted graph is not new, see e.g [30, 66, 157]. In particular, [66] proposes a general form of combinatorial Laplace operator for a simplicial complex where all simplices are weighted, and our doubly-weighted Laplace has the same eigenstructure as their Laplacian when restricted to graphs. See Section 4.10.1 for details. Using the doubly-weighted Laplacian for Rayleigh quotient based similarity measurement between the original graph and the coarse graph is justified by the following result (proof in Section 4.10.1).

Proposition 4.2.3. *For any vector $x \in \mathbb{R}^n$, we have that $\mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x}) = \mathsf{R}_L(P^+\Gamma^{-1/2}\hat{x})$. That is, set the lift of $\hat{x}$ in $\mathbb{R}^N$ to be $x = P^+\Gamma^{-1/2}\hat{x}$, then we have that $\frac{\hat{x}^T\widehat{\mathsf{L}}\hat{x}}{\hat{x}^T\hat{x}} = \frac{x^TLx}{x^Tx}$.*

Finally, if using the normalized Laplace $\mathscr{L}$ for the original graph G, then the appropriate Laplace operator for the coarse graph and corresponding projection/lift maps are listed in the last row of Table 4.1, with proofs in Section 4.10.1.

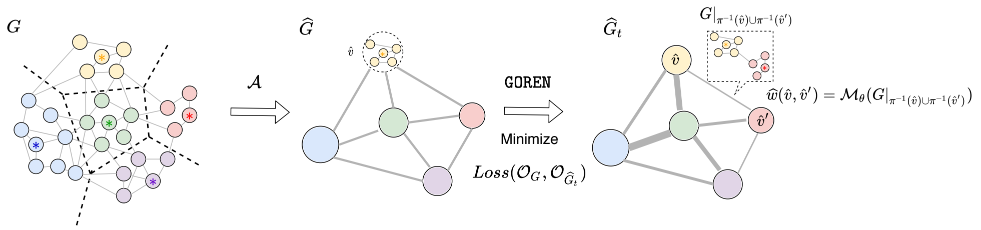

G
$\mathcal{A}$
$\widehat{G}$
$\hat{v}$
GOREN
Minimize
$Loss(\mathcal{O}_G, \mathcal{O}_{\widehat{G}_t})$
$\widehat{G}_t$
$G|_{\pi^{-1}(\hat{v})\cup\pi^{-1}(\hat{v}')}$
$\hat{w}(\hat{v},\hat{v}') = \mathcal{M}_\theta(G|_{\pi^{-1}(\hat{v})\cup\pi^{-1}(\hat{v}')})$
$\hat{v}'$

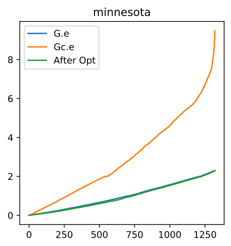
Figure 4.1. An illustration of learnable coarsening framework. Existing coarsening algorithm determines the topology of coarse graph $\widehat{G}$, while GOREN resets the edge weights of the coarse graph.

4.2.4 A GNN-based Framework for Constructing the Coarse Graph

In the previous section, we argued that depending on what similarity measures we use, appropriate Laplace operator $\mathscr{O}_{\widehat{G}}$ for the coarse graph $\widehat{G}$ should be used. Now consider the specific case of Rayleigh quotient, which can be thought of as a proxy to measure similarities between the low-frequency eigenvalues of the original graph Laplacian and the one for the coarse graph. As described above, here we set $\mathscr{O}_{\widehat{G}}$ as the doubly-weighted Laplacian $\widehat{\mathsf{L}} = \Gamma^{-1/2}(\widehat{D} - \widehat{W})\Gamma^{-1/2}$.

The effect of weight adjustments.

We develop an iterative algorithm with convergence guarantee (to KKT point in 6) for optimizing over edge weights of $\widehat{G}$ for better spectrum alignment. As shown in the figure on the right, after changing the edge weight of the coarse graph, the resulting graph Laplacian has eigenvalues much closer (almost identical) to the first n eigenvalues of the original graph Laplacian. More specifically, in this figure, $G.e$ and $Gc.e$ stand for the eigenvalues of the original graph G and coarse graph $\widehat{G}$ constructed by the so-called Variation-Edge coarsening algorithm [104]. “After-Opt" stands for the eigenvalues of coarse graphs when weights are optimized by our iterative algorithm. See Section 4.10.2 for the description of our iterative algorithm, its convergence results, and full experiment results.



A GNN-based framework for learning weight assignment map. The discussions above indicate that we can obtain better Laplace operators for the coarse graph by using betterinformed weights than simply summing up the weights of crossing edges from the two clusters.

More specifically, suppose we have a fixed strategy to generate $\widehat{V}$ from an input graph $G = (V,E)$. Now given an edge $(\hat{v},\hat{v}') \in \widehat{E}$ in the induced coarse graph $\widehat{G} = (\widehat{V},\widehat{E})$, we model its weight $\hat{w}(\hat{v},\hat{v}')$ by a *weight-assignment function* $\mu(G|_{\pi^{-1}(\hat{v})\cup\pi^{-1}(\hat{v}')})$, where $G|_A$ is the subgraph of G induced by a subset of vertices A. However, it is not clear how to setup this function μ. Instead, we will learn it from a collection of input graphs in an *unsupervised* manner. Specifically, we will parametrize the weight-assignment map μ by a learnable neural network $\mathscr{M}_\theta$.

See Figure 4.1 for an illustration.

In particular, we use Graph Isomorphism Network (GIN) [156] to represent $\mathscr{M}_\theta$. We initialize the model by setting the edge attribute of the coarse graph to be 1. Our node feature is set to be a 5-dimensional vector based on LDP (Local Degree Profile) [20]. We enforce the learned weight of the coarse graph to be positive by applying one extra ReLU layer to the final output. All models are trained with Adam optimizer with a learning rate of 0.001. See Section 4.3.3 for more details. We name our model as **G**raph c**O**arsening **R**efinem**E**nt **N**etwork (GOREN).

Given a graph G and a coarsening algorithm $\mathscr{A}$, the general form of loss is

$$Loss(\mathscr{O}_G, \mathscr{O}_{\widehat{G_t}}) = \frac{1}{k}\sum_{i=1}^{k} |\mathscr{F}(\mathscr{O}_G, f_i) - \mathscr{F}(\mathscr{O}_{\widehat{G_t}}, \mathscr{P}f_i)|, \tag{4.1}$$

where f_i is signal on the original graph (such as eigenvectors) and $\mathscr{P}f_i$ is its projection. We use $\mathscr{O}_{\widehat{G_t}}$ to denote the operator of the coarse graph *during training*, while $\mathscr{O}_{\widehat{G}}$ standing for the operator defined w.r.t. the coarse graph output by coarsening algorithm $\mathscr{A}$. That is, we will start with $\mathscr{O}_{\widehat{G}}$ and modify it to $\mathscr{O}_{\widehat{G_t}}$ during the training.

The loss can be instantiated for different cases in Table 4.1. For example, a loss based on quadratic form means that we choose $\mathscr{O}_G, \mathscr{O}_{\widehat{G_t}}$ to be the combinatorial Laplacian of G and $\widehat{G_t}$, and the resulting *quadratic loss* has the form:

$$Loss(L,\widehat{L}_t) = \frac{1}{k}\sum_{i=1}^{k} |f_i^T L f_i - (P f_i)^T \widehat{L}_t (P f_i)|. \tag{4.2}$$

It can be seen as a natural analog of the loss for spectral sparsification in the context of graph coarsening, which is also adopted in [104].

Similarly, one can use a loss based on the Rayleigh quotient, by choosing $\mathscr{F}$ from the second row of Table 4.1. Our framework for graph coarsening is flexible. Many different loss functions can be used as long as it is differentiable in the weights of the coarse graph. we will demonstrate this point in Section 4.5.

Finally, given a collection of training graphs $G_1, \ldots, G_m$, we will train for parameters in the module $\mathscr{M}_\theta$ to minimize the total loss on training graphs. When a test graph G_{test} is given, we simply apply $\mathscr{M}_\theta$ to set up weight for each edge in $\widehat{G_{test}}$, obtaining a new graph $\widehat{G_{test,t}}$. We compare $Loss(\mathscr{O}_{G_{test}}, \mathscr{O}_{\widehat{G_{test,t}}})$ against $Loss(\mathscr{O}_{G_{test}}, \mathscr{O}_{\widehat{G_{test}}})$ and expect the former loss is smaller.

4.3 Experiments Setup

In the following experiments, we apply six existing coarsening algorithms to obtain the coarsened vertex set $\widehat{V}$, which are Affinity [102], Algebraic Distance [26], Heavy edge matching [38, 122], as well as two local variation methods based on edge and neighborhood respectively [104], and a simple baseline (BL); See Section 4.3.2 for detailed descriptions. The two local variation methods are considered to be state-of-the-art graph coarsening algorithms [104]. We show that our `GOREN` framework can improve the qualities of coarse graphs produced by these methods.

4.3.1 Dataset

Synthetic Graphs

Erdős-Rényi graphs (ER). $G(n, p)$ where $p = \frac{0.1*512}{n}$

Random geometric graphs (GEO). The random geometric graph model places n nodes uniformly at random in the unit cube. Two nodes are joined by an edge if the distance between the nodes is at most radius r. We set $r = \frac{5.12}{\sqrt{n}}$.

Barabasi-Albert Graph (BA). A graph of n nodes is grown by attaching new nodes each with m edges that are preferentially attached to existing nodes with high degrees. We set m to be 4.

Watts-Strogatz Graph (WS). It is first created from a ring over n nodes. Then each node in the ring is joined to its k nearest neighbors (or $k-1$ neighbors if k is odd). Then shortcuts are created by replacing some edges as follows: for each edge (u,v) in the underlying "n-ring with k nearest neighbors" with probability p replace it with a new edge (u,w) with a uniformly random choice of existing node w. We set k,p to be 10 and 0.1.

Dataset from Loukas's paper

Yeast. Protein-to-protein interaction network in budding yeast, analyzed by [74]. The network has $N = 1458$ vertices and $M = 1948$ edges.

Airfoil. Finite-element graph obtained by airow simulation [117], consisting of $N = 4000$ vertices and $M = 11,490$ edges.

Minnesota [55]. Road network with $N = 2642$ vertices and $M = 3304$ edges.

Bunny [141]. Point cloud consisting of $N = 2503$ vertices and $M = 65,490$ edges. The point cloud has been sub-sampled from its original size.

Real Networks

Shape graphs (Shape). Each graph is KNN graph formed by 1024 points sampled from shapes from ShapeNet where each node is connected 10 nearest neighbors.

Coauthor-CS (CS) and Coauthor-Physics (Physics) are co-authorship graphs based on the Microsoft Academic Graph from the KDD Cup 2016 challenge. Coauthor CS has $N = 18,333$ nodes and $M = 81,894$ edges. Coauthor Physics has $N = 34,493$ nodes and $M = 247,962$ edges.

PubMed [132] has $N = 19,717$ nodes and $M = 44,324$ edges. Nodes are documents and edges are citation links.

Flickr [163] has $N = 89,250$ nodes and $M = 899,756$ edges. One node in the graph represents one image uploaded to Flickr. If two images share some common properties (e.g., same geographic location, same gallery, comments by the same user, etc.), there is an edge between the nodes of these two images.

4.3.2 Existing Graph Coarsening Methods

Heavy Edge Matching. At each level of the scheme, the contraction family is obtained by computing a maximum-weight matching with the weight of each contraction set (v_i, v_j) calculated as $w_{ij}/\max\{d_i, d_j\}$. In this manner, heavier edges connecting vertices that are well separated from the rest of the graph are contracted first.

Algebraic Distance. This method differs from heavy edge matching in that the weight of each candidate set $(v_i, v_j) \in E$ is calculated as $\left(\sum_{q=1}^{Q} \left(x_q(i) - x_q(j)\right)^2\right)^{1/2}$, where x_k is an N-dimensional test vector computed by successive sweeps of Jacobi relaxation. The complete method is described by [122], see also [26].

Affinity. This is a vertex proximity heuristic in the spirit of the algebraic distance that was proposed by [102] in the context of their work on the lean algebraic multigrid. As per the author suggests, the $Q = k$ test vectors are here computed by a single sweep of a Gauss-Seidel iteration.

Local Variation. There are two variations of local variation methods, edge-based local variation, and neighborhood-based local variation. They differ in how the contraction set is chosen. Edge-based variation is constructed for each edge, while the neighborhood-based variant takes every vertex and its neighbors as contraction set. What two methods have common is that they both optimize an upper bound of the restricted spectral approximation objective. In each step, they greedily pick the sets whose local variation is the smallest. See [104] for more details.

Baseline. We also implement a simple baseline that randomly chooses a collection of nodes in the original graph as landmarks and contract other nodes to the nearest landmarks. If there are multiple nearest landmarks, we randomly break the tie. The weight of the coarse graph is set to be the sum of the weights of the crossing edges.

4.3.3 Details of the Experimental Setup

Feature Initialization. We initialize the the node feature of subgraphs as a 5 dimensional feature based on a simple heuristics local degree profile (LDP) [20]. For each node $v \in G(V)$, let $DN(v)$ denote the multiset of the degree of all the neighboring nodes of v, i.e., $DN(v) = \{\text{degree}(u) | (u, v) \in E\}$. We take five node features, which are (degree(v), min(DN(v)), max(DN(v)),mean(DN(v)), std(DN(v))). In other words, each node feature summarizes the degree information of this node and its 1- neighborhood. We use the edge weight as 1 dimensional edge feature.

Optimization. All models are trained with Adam optimizer [83] with a learning rate of 0.001 and batch size 600. We use Pytorch [116] and Pytorch Geometric [49] for all of our implementation. We train graphs one by one where for each graph we train the model to minimize the loss for certain epochs (see hyper-parameters for details) before moving to the next graph. We save the model that performs best on the validation graphs and test it on the test graphs.

Model Architecture. The building block of our graph neural networks is based on the modification of Graph Isomorphism Network (GIN) that can handle both node and edge features. In particular, we first linear transform both node feature and edge feature to be vectors of the same dimension. At the k-th layer, GNNs update node representations by

$$h_v^{(k)} = \text{ReLU}\left(\text{MLP}^{(k)}\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k-1)} + \sum_{e=(v,u): u \in \mathcal{N}(v) \cup \{v\}} h_e^{(k-1)}\right)\right) \tag{4.3}$$

where $\mathcal{N}(v)$ is a set of nodes adjacent to v, and $e = (v; v)$ represents the self-loop edge. Edge features $h_e^{(k-1)}$ is the same across the layers.

We use average graph pooling to obtained the graph representation from node embeddings, i.e., $h_G = \text{MEAN}\left(\left\{h_v^{(K)} | v \in G\right\}\right)$. The final prediction of weight is $1 + \text{ReLu}(\Phi(h_G))$ where Φ is a linear layer. We set the number of layers to be 3 and the embedding dimension to be 50.

Time Complexity. In the preprocessing step, we need to compute the first k eigenvectors of Laplacian (either combinatorial or normalized one) of the original graph as test vectors. Those can be efficiently computed by Restarted Lanczos Method [94] to find the eigenvalues and eigenvectors.

In the training time, our model needs to recompute the term in the loss involving the coarse graph to update the weights of the graph neural networks for each batch. For loss involving Laplacian (either combinatorial or normalized Laplacian), the time complexity to compute the $x^T Lx$ is $O(|E|k)$ where $|E|$ is the number of edges in the coarse graph and k is the number of test vectors. For loss involving conductance, computing the conductance of one subset $S \subset E$ is still $O(|E|)$ so in total the time complexity is also $O(|E|k)$. In summary, the time complexity for each batch is linear in the number of edges of training graphs. All experiments are performed on a single Intel Xeon CPU E5-2630 v4@ 2.20GHz $\times$ 40 and 64GB RAM machine.

More concretely, for synthetic graphs, it takes a few minutes to train the model. For real graphs like CS, Physics, PubMed, it takes around 1 hour. For the largest network Flickr of 89k nodes and 899k edges, it takes about 5 hours for most coarsening algorithms and reduction ratios.

Hyperparameters. We list the major hyperparameters of GOREN below.

- epoch: 50 for synthetic graphs and 30 for real networks.
- walk length: 5000 for real networks. Note the size of the subgraph is usually around 3500 since the random walk visits some nodes more than once.
- number of eigenvectors k: 40 for synthetic graphs and 200 for real networks.
- embedding dimension: 50
- batch size: 600

- learning rate: 0.001

4.4 Proof of Concept

Table 4.2. The error reduction after applying GOREN.

Dataset	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Airfoil	91.7%	88.2%	86.1%	43.2%	73.6%
Minnesota	49.8%	57.2%	30.1%	5.50%	1.60%
Yeast	49.7%	51.3%	37.4%	27.9%	21.1%
Bunny	84.7%	69.1%	61.2%	19.3%	81.6%

As proof of concept, we show that GOREN can improve common coarsening methods on multiple graphs (see 4.3.1 for details). Following the same setting as [104], we use the relative eigenvalue error as evaluation metric. It is defined as $\frac{1}{k}\sum_{i=1}^{k}\frac{\left|\widehat{\lambda}_i-\lambda_i\right|}{\lambda_i}$, where $\lambda_i, \widehat{\lambda}_i$ denotes eigenvalues of combinatorial Laplacian L for G and doubly-weighted Laplacian $\widehat{\mathsf{L}}$ for $\widehat{G}$ respectively, and k is set to be 40. For simplicity, this error is denoted as *Eigenerror* in the remainder of the chapter. Denote the Eigenerror of graph coarsening method as l_1 and Eigenerror obtained by GOREN as l_2. In Table 4.2, we show the *error-reduction ratio*, defined as $\frac{l_1-l_2}{l_1}$. The ratio is upper bounded by 100.

Since it is hard to directly optimize Eigenerror, the loss function we use in our GOREN set to be the *Rayleigh loss* $Loss(\mathcal{O}_G, \mathcal{O}_{\widehat{G_t}}) = \frac{1}{k}\sum_{i=1}^{k}|\mathcal{F}(\mathcal{O}_G, f_i) - \mathcal{F}(\mathcal{O}_{\widehat{G_t}}, \mathcal{P}f_i)|$ where $\mathcal{F}$ is Rayleigh quotient, $\mathcal{P} = \Gamma^{-1/2}(P^+)^T$ and $\mathcal{O}_{\widehat{G}_t}$ being doubly-weighted Laplacian $\widehat{\mathsf{L}}_t$. In other words, We use Rayleigh loss as a differentiable proxy for the Eigenerror. As we can see in Table 4.2, GOREN reduces the Eigenerror by a large margin for *training* graphs, which serves as a sanity check for our framework, as well as for using Rayleigh loss as a proxy for Eigenerror. See Section 4.4.2 for full results where we reproduce the results in [104] up to small differences.

In Table 4.7, we will demonstrate this training strategy also generalizes well to unseen graphs.

Table 4.3. Loss: quadratic loss. Laplacian: combinatorial Laplacian for both original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.44 (16.1%)	0.44 (4.4%)	0.68 (4.3%)	0.61 (3.6%)	0.21 (14.1%)	0.18 (72.7%)
	ER	0.36 (1.1%)	0.52 (0.8%)	0.35 (0.4%)	0.36 (0.2%)	0.18 (1.2%)	0.02 (7.4%)
	GEO	0.71 (87.3%)	0.20 (57.8%)	0.24 (31.4%)	0.55 (80.4%)	0.10 (59.6%)	0.27 (65.0%)
	WS	0.45 (62.9%)	0.09 (82.1%)	0.09 (60.6%)	0.52 (51.8%)	0.09 (69.9%)	0.11 (84.2%)
Real	CS	0.39 (40.0%)	0.21 (29.8%)	0.17 (26.4%)	0.14 (20.9%)	0.06 (36.9%)	0.0 (59.0%)
	Flickr	0.25 (10.2%)	0.25 (5.0%)	0.19 (6.4%)	0.26 (5.6%)	0.11 (11.2%)	0.07 (21.8%)
	Physics	0.40 (47.4%)	0.37 (42.4%)	0.32 (49.7%)	0.14 (28.0%)	0.15 (60.3%)	0.0 (-0.3%)
	PubMed	0.30 (23.4%)	0.13 (10.5%)	0.12 (15.9%)	0.24 (10.8%)	0.06 (11.8%)	0.01 (36.4%)
	Shape	0.23 (91.4%)	0.08 (89.8%)	0.06 (82.2%)	0.17 (88.2%)	0.04 (80.2%)	0.08 (79.4%)

4.4.1 Synthetic Graphs

We train the `GOREN` on synthetic graphs from common graph generative models and test on larger unseen graphs from the same model. We randomly sample 25 graphs of size $\{512, 612, 712, ..., 2912\}$ from different generative models. If the graph is disconnected, we keep the largest component. We train `GOREN` on the first 5 graphs, use the 5 graphs from the rest 20 graphs as the validation set and the remaining 15 as test graphs. We use the following synthetic graphs: Erdős-Rényi graphs (ER), Barabasi-Albert Graph (BA), Watts-Strogatz Graph (WS), random geometric graphs (GEO). See Section 4.3.1 for datasets details.

We report both the loss $Loss(L, \widehat{L})$ of different algorithms (w/o learning) and the *relative improvement percentage* defined as $\frac{Loss(L,\widehat{L})-Loss(L,\widehat{L}_t)}{Loss(L,\widehat{L})}$ when `GOREN` is applied, shown in parenthesis.

Erdős-Rényi graphs (ER). $G(n, p)$ where $p = \frac{0.1*512}{n}$

As we can see in Table 4.3, for most methods, trained on small graphs, `GOREN` also performs well on test graphs of larger size across different algorithms and datasets – Again, the larger improvement percentage is, the larger the improvement by our algorithm is, and a negative value means that our algorithm makes the loss worse. Note the size of test graphs are on average $2.6\times$ the size of training graphs.

For ER and BA graphs, the improvement is relatively smaller compared to GEO and WS graphs. This makes sense since ER and BA graphs are rather homogenous graphs, leaving less room for further improvement.

4.4.2 Real Networks

We test on five real networks: Shape, PubMed, Coauthor-CS (CS), Coauthor-Physics (Physics), and Flickr (largest one with 89k vertices), which are much larger than datasets used in [62] ($\leq$ 1.5k) and [104] ($\leq$ 4k). Since it is hard to obtain multiple large graphs (except for the Shape dataset, which contains meshes from different surface models) coming from similar distribution, we bootstrap the training data in the following way. For the given graph, we randomly sample a collection of landmark vertices and take a random walk of length l starting from selected vertices. We take subgraphs spanned by vertices of random walks as training and validation graphs and the original graph as the test graph. See Section 4.3.1 for dataset details.

As shown in the bottom half of Table 4.3, across all six different algorithms, `GOREN` significantly improves the result among all five datasets in most cases. For the largest graph Flickr, the size of test graphs is more than $25\times$ of the training graphs, which further demonstrates the strong generalization.

4.5 Other Losses

Other differentiable loss. To demonstrate that our framework is flexible, we adapt `GOREN` to the following two losses. The two losses are both differentiable w.r.t the weights of coarse graph.

(1) Loss based on normalized graph Laplacian: $Loss(\mathscr{L}, \widehat{\mathscr{L}_t}) = \frac{1}{k}\sum_{i=1}^{k} |f_i^T \mathscr{L} f_i - (\mathscr{P} f_i)^T \widehat{\mathscr{L}_t}(\mathscr{P} f_i)|$. Here $\{f_i\}$ are the set of first k eigenvectors of the normalized Laplacian $\mathscr{L}$ of original grpah G, and $\mathscr{P} = \widehat{D}^{1/2} P D^{-1/2}$. (2) Conductance difference between original graph and coarse

Table 4.4. Relative eigenvalue error (Eigenerror) by different coarsening algorithm and the improvement (in percentage) after applying GOREN.

Dataset	Ratio	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Airfoil	0.3	0.262 (82.1%)	0.208 (64.9%)	0.279 (80.3%)	0.102 (-67.6%)	0.184 (69.6%)
	0.5	0.750 (91.7%)	0.672 (88.2%)	0.568 (86.1%)	0.336 (43.2%)	0.364 (73.6%)
	0.7	2.422 (96.4%)	2.136 (93.5%)	1.979 (96.7%)	0.782 (78.8%)	0.876 (87.8%)
Minnesota	0.3	0.322 (-5.0%)	0.206 (0.5%)	0.357 (-4.5%)	0.118 (-5.9%)	0.114 (-14.0%)
	0.5	1.345 (49.8%)	1.054 (57.2%)	0.996 (30.1%)	0.457 (5.5%)	0.382 (1.6%)
	0.7	4.290 (70.4%)	3.787 (76.6%)	3.423 (58.9%)	2.073 (55.0%)	1.572 (38.1%)
Yeast	0.3	0.202 (10.4%)	0.108 (5.6%)	0.291 (1.4%)	0.113 (6.2%)	0.024 (-58.3%)
	0.5	0.795 (49.7%)	0.485 (51.3%)	1.080 (37.4%)	0.398 (27.9%)	0.133 (21.1%)
	0.7	2.520 (60.4%)	2.479 (72.4%)	3.482 (52.9%)	2.073 (58.9%)	0.458 (45.9%)
Bunny	0.3	0.046 (32.6%)	0.217 (50.0%)	0.258 (74.4%)	0.007 (-328.5%)	0.082 (74.8%)
	0.5	0.085 (84.7%)	0.372 (69.1%)	0.420 (61.2%)	0.057 (19.3%)	0.169 (81.6%)
	0.7	0.182 (84.6%)	0.574 (78.6%)	0.533 (75.4%)	0.094 (45.7%)	0.283 (73.9%)

graph. $Loss = \frac{1}{k}\sum_{i=1}^{k}|\varphi(S_i) - \varphi(\pi(S_i))|$. $\varphi(S)$ is the conductance $\varphi(S) := \frac{\sum_{i\in S, j\in \bar{S}} a_{ij}}{\min(a(S), a(\bar{S}))}$ where $a(S) := \sum_{i\in S}\sum_{j\in V} a_{ij}$. We randomly sample k subsets of nodes $S_0, \ldots, S_k \subset V$ where

$|S_i|$ is set to be a random number sampled from the uniform distribution $U(|V|/4, |V|/2)$. Due to space limits, we present the result for conductance in Section 4.5.

Following the same setting as before, we perform experiments to minimize two different losses. As shown in Table 4.6 and Section 4.5, for most graphs and methods, GOREN still shows good generalization capacity and improvement for both losses. Apart from that, we also observe the initial loss for normalized Laplacian is much smaller than that for standard Laplacian, which might be due to that the fact that eigenvalues of normalized Laplacian are in $[0, 2]$.

Non-differentiable loss. In Section 4.4, we use Rayleigh loss as a proxy for training but the Eigenerror for validation and test. Here we train GOREN with Rayleigh loss but evaluate *Eigenerror* on *test* graphs, which is more challenging. Number of vectors k is 40 for synthetic graphs and 200 for real networks.

Table 4.5. Loss: quadratic loss. Laplacian: *combinatorial* Laplacian for both original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage. BL stands for the baseline.

Dataset	Ratio	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
BA	0.3	0.36 (6.8%)	0.22 (2.9%)	0.56 (1.9%)	0.49 (1.7%)	0.06 (16.6%)	0.17 (73.1%)
	0.5	0.44 (16.1%)	0.44 (4.4%)	0.68 (4.3%)	0.61 (3.6%)	0.21 (14.1%)	0.18 (72.7%)
	0.7	0.21 (32.0%)	0.43 (16.5%)	0.47 (17.7%)	0.4 (19.3%)	0.2 (48.2%)	0.11 (11.1%)
CS	0.3	0.25 (28.7%)	0.08 (24.8%)	0.05 (21.5%)	0.09 (15.6%)	0.0 (-254.3%)	0.0 (60.6%)
	0.5	0.39 (40.0%)	0.21 (29.8%)	0.17 (26.4%)	0.14 (20.9%)	0.06 (36.9%)	0.0 (59.0%)
	0.7	0.46 (55.5%)	0.57 (36.8%)	0.33 (36.6%)	0.28 (29.3%)	0.18 (44.2%)	0.09 (26.5%)
Physics	0.3	0.26 (35.4%)	0.36 (36.6%)	0.2 (29.7%)	0.1 (18.6%)	0.0 (-42.0%)	0.0 (2.5%)
	0.5	0.4 (47.4%)	0.37 (42.4%)	0.32 (49.7%)	0.14 (28.0%)	0.15 (60.3%)	0.0 (-0.3%)
	0.7	0.47 (60.0%)	0.53 (55.3%)	0.42 (61.4%)	0.27 (34.4%)	0.25 (67.0%)	0.01 (-4.9%)
Flickr	0.3	0.16 (5.3%)	0.17 (2.0%)	0.08 (4.3%)	0.18 (2.7%)	0.01 (16.0%)	0.02 (33.7%)
	0.5	0.25 (10.2%)	0.25 (5.0%)	0.19 (6.4%)	0.26 (5.6%)	0.11 (11.2%)	0.07 (21.8%)
	0.7	0.28 (21.0%)	0.31 (12.4%)	0.37 (18.7%)	0.33 (11.3%)	0.2 (17.2%)	0.2 (21.4%)
PubMed	0.3	0.17 (13.6%)	0.06 (6.2%)	0.03 (9.5%)	0.1 (4.7%)	0.01 (18.8%)	0.0 (39.9%)
	0.5	0.3 (23.4%)	0.13 (10.5%)	0.12 (15.9%)	0.24 (10.8%)	0.06 (11.8%)	0.01 (36.4%)
	0.7	0.31 (41.3%)	0.23 (22.4%)	0.14 (8.3%)	0.14 (-491.6%)	0.16 (12.5%)	0.05 (21.2%)
ER	0.3	0.25 (0.5%)	0.41 (0.2%)	0.2 (0.5%)	0.23 (0.2%)	0.01 (4.8%)	0.01 (5.9%)
	0.5	0.36 (1.1%)	0.52 (0.8%)	0.35 (0.4%)	0.36 (0.2%)	0.18 (1.2%)	0.02 (7.4%)
	0.7	0.39 (3.2%)	0.55 (2.5%)	0.44 (2.0%)	0.43 (0.8%)	0.23 (2.9%)	0.29 (10.4%)
GEO	0.3	0.44 (86.4%)	0.11 (65.1%)	0.12 (81.5%)	0.34 (80.7%)	0.01 (0.3%)	0.14 (70.4%)
	0.5	0.71 (87.3%)	0.2 (57.8%)	0.24 (31.4%)	0.55 (80.4%)	0.1 (59.6%)	0.27 (65.0%)
	0.7	0.96 (83.2%)	0.4 (55.2%)	0.33 (54.8%)	0.72 (90.0%)	0.19 (72.4%)	0.41 (61.0%)
Shape	0.3	0.13 (86.6%)	0.04 (79.8%)	0.03 (69.0%)	0.11 (69.7%)	0.0 (1.3%)	0.04 (73.6%)
	0.5	0.23 (91.4%)	0.08 (89.8%)	0.06 (82.2%)	0.17 (88.2%)	0.04 (80.2%)	0.08 (79.4%)
	0.7	0.34 (91.1%)	0.17 (94.3%)	0.1 (74.7%)	0.24 (95.9%)	0.09 (64.6%)	0.13 (84.8%)
WS	0.3	0.27 (46.2%)	0.04 (65.6%)	0.04 (-26.9%)	0.43 (32.9%)	0.02 (68.2%)	0.06 (75.2%)
	0.5	0.45 (62.9%)	0.09 (82.1%)	0.09 (60.6%)	0.52 (51.8%)	0.09 (69.9%)	0.11 (84.2%)
	0.7	0.65 (73.4%)	0.15 (78.4%)	0.14 (66.7%)	0.67 (76.6%)	0.15 (80.8%)	0.16 (83.2%)

Table 4.6. Loss: quadratic loss. Laplacian: normalized Laplacian for original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.13 (76.2%)	0.14 (45.0%)	0.15 (51.8%)	0.15 (46.6%)	0.14 (55.3%)	0.06 (57.2%)
	ER	0.10 (82.2%)	0.10 (83.9%)	0.09 (79.3%)	0.09 (78.8%)	0.06 (64.6%)	0.06 (75.4%)
	GEO	0.04 (52.8%)	0.01 (12.4%)	0.01 (27.0%)	0.03 (56.3%)	0.01 (-145.1%)	0.02 (-9.7%)
	WS	0.05 (83.3%)	0.01 (-1.7%)	0.01 (38.6%)	0.05 (50.3%)	0.01 (40.9%)	0.01 (10.8%)
Real	CS	0.08 (58.0%)	0.06 (37.2%)	0.04 (12.8%)	0.05 (41.5%)	0.02 (16.8%)	0.01 (50.4%)
	Flickr	0.08 (-31.9%)	0.06 (-27.6%)	0.06 (-67.2%)	0.07 (-73.8%)	0.02 (-440.1%)	0.02 (-43.9%)
	Physics	0.07 (47.9%)	0.06 (40.1%)	0.04 (17.4%)	0.04 (61.4%)	0.02 (-23.3%)	0.01 (35.6%)
	PubMed	0.05 (47.8%)	0.05 (35.0%)	0.05 (41.1%)	0.12 (46.8%)	0.03 (-66.4%)	0.01 (-118.0%)
	Shape	0.02 (84.4%)	0.01 (67.7%)	0.01 (58.4%)	0.02 (87.4%)	0.0 (13.3%)	0.01 (43.8%)

As shown in Table 4.7, our training strategy via Rayleigh loss can improve the eigenvalue alignment between original graphs and coarse graphs in most cases. Reducing Eigenerror is more challenging than other losses, possibly because we are minimizing a differentiable proxy (the Rayleigh loss). Nevertheless, improvement is achieved in most cases.

4.6 On the Use of GNN as Weight-Assignment Map.

Recall that we use GNN to represent a edge-weight assignment map for an edge $(\hat{u},\hat{v})$ between two super-nodes $\hat{u},\hat{v}$ in the coarse graph $\widehat{G}$. The input will be the subgraph $G_{\hat{u},\hat{v}}$ in the original graph G spanning the clusters $\pi^{-1}(\hat{u})$, $\pi^{-1}(\hat{v})$, and the crossing edges among them; while the goal is to compute the weight of edge $(\hat{u},\hat{v})$ based on this subgraph $G_{\hat{u},\hat{v}}$. Given that the input is a local graph $G_{\hat{u},\hat{v}}$, a GNN will be a natural choice to parameterize this edge-weight assignment map. Nevertheless, in principle, any architecture applicable to graph regression can be used for this purpose. To better understand if it is necessary to use the power of GNN, we replace GNN with the following baseline for graph regression. In particular, the baseline is a composition of mean pooling of node features in the original graph and a 4-layer MLP with embedding dimension 200 and ReLU nonlinearity. We use mean-pooling as the graph regression

Table 4.7. Loss: Eigenerror. Laplacian: combinatorial Laplacian for original graphs and doubly-weighted Laplacian for coarse ones. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage. † stands for out of memory.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.36 (7.1%)	0.17 (8.2%)	0.22 (6.5%)	0.22 (4.7%)	0.11 (21.1%)	0.17 (-15.9%)
	ER	0.61 (0.5%)	0.70 (1.0%)	0.35 (0.6%)	0.36 (0.2%)	0.19 (1.2%)	0.02 (0.8%)
	GEO	1.72 (50.3%)	0.16 (89.4%)	0.18 (91.2%)	0.45 (84.9%)	0.08 (55.6%)	0.20 (86.8%)
	WS	1.59 (43.9%)	0.11 (88.2%)	0.11 (83.9%)	0.58 (23.5%)	0.10 (88.2%)	0.12 (79.7%)
Real	CS	1.10 (18.0%)	0.55 (49.8%)	0.33 (60.6%)	0.42 (44.5%)	0.21 (75.2%)	0.0 (-154.2%)
	Flickr	0.57 (55.7%)	†	0.33 (20.2%)	0.31 (55.0%)	0.11 (67.6%)	0.07 (60.3%)
	Physics	1.06 (21.7%)	0.58 (67.1%)	0.33 (69.5%)	0.35 (64.6%)	0.20 (79.0%)	0.0 (-377.9%)
	PubMed	1.25 (7.1%)	0.50 (15.5%)	0.51 (12.3%)	1.19 (-110.1%)	0.35 (-8.8%)	0.02 (60.4%)
	Shape	2.07 (67.7%)	0.24 (93.3%)	0.17 (90.9%)	0.49 (93.0%)	0.11 (84.2%)	0.20 (90.7%)

component needs to be permutation invariant over the set of node features. However, this baseline ignores the detailed graph structure which GNN will leverage. The results for different reduction ratios are presented in the Table 4.11. We have also implemented another baseline where the MLP module is replaced by a simpler linear regression module. The results are worse than those of MLP (and thus also GNN) as expected, and therefore omitted from this chapter.

As we can see, MLP works reasonably well in most cases, indicating that learning the edge weights is indeed useful for improvement. On the other hand, we see using GNN to parametrize the map generally yields a larger improvement over the MLP, which ignores the topology of subgraphs in the original graph. A systematic understanding of how different models such as various graph kernels [87, 144] and graph neural networks affect the performance is an interesting question that we will leave for future work.

4.7 Visualization

We visualize the subgraphs corresponding to randomly sampled edges of coarse graphs. For example, in WS graphs, some subgraphs have only a few nodes and edges, while other

Table 4.8. Loss: quadratic loss. Laplacian: *normalized* Laplacian for both original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage. BL stands for the baseline.

Dataset	Ratio	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
	0.3	0.06 (68.6%)	0.07 (73.9%)	0.08 (80.6%)	0.08 (79.6%)	0.06 (79.4%)	0.01 (-15.8%)
BA	0.5	0.13 (76.2%)	0.14 (45.0%)	0.15 (51.8%)	0.15 (46.6%)	0.14 (55.3%)	0.06 (57.2%)
	0.7	0.22 (17.0%)	0.23 (5.5%)	0.24 (10.8%)	0.24 (9.7%)	0.23 (5.4%)	0.17 (36.8%)
	0.3	0.04 (50.2%)	0.03 (44.1%)	0.01 (-7.0%)	0.03 (50.1%)	0.0 (-135.0%)	0.01 (-11.7%)
CS	0.5	0.08 (58.0%)	0.06 (37.2%)	0.04 (12.8%)	0.05 (41.5%)	0.02 (16.8%)	0.01 (50.4%)
	0.7	0.13 (57.8%)	0.1 (36.3%)	0.09 (21.4%)	0.09 (29.3%)	0.05 (11.6%)	0.04 (10.8%)
	0.3	0.05 (32.3%)	0.04 (5.4%)	0.02 (-16.5%)	0.03 (69.3%)	0.0 (-1102.4%)	0.0 (-59.8%)
Physics	0.5	0.07 (47.9%)	0.06 (40.1%)	0.04 (17.4%)	0.04 (61.4%)	0.02 (-23.3%)	0.01 (35.6%)
	0.7	0.14 (60.8%)	0.1 (52.0%)	0.06 (20.9%)	0.07 (29.9%)	0.04 (11.9%)	0.02 (39.1%)
	0.3	0.05 (-29.8%)	0.05 (-31.7%)	0.05 (-21.8%)	0.05 (-66.8%)	0.0 (-293.4%)	0.01 (13.4%)
Flickr	0.5	0.08 (-31.9%)	0.06 (-27.6%)	0.06 (-67.2%)	0.07 (-73.8%)	0.02 (-440.1%)	0.02 (-43.9%)
	0.7	0.08 (-55.3%)	0.07 (-32.3%)	0.04 (-316.0%)	0.07 (-138.4%)	0.03 (-384.6%)	0.04 (-195.6%)
	0.3	0.03 (13.1%)	0.03 (-15.7%)	0.01 (-79.9%)	0.04 (-3.2%)	0.01 (-191.7%)	0.0 (-53.7%)
PubMed	0.5	0.05 (47.8%)	0.05 (35.0%)	0.05 (41.1%)	0.12 (46.8%)	0.03 (-66.4%)	0.01 (-118.0%)
	0.7	0.09 (58.0%)	0.09 (34.7%)	0.07 (68.7%)	0.07 (21.2%)	0.08 (67.2%)	0.03 (43.1%)
	0.3	0.06 (84.3%)	0.06 (82.0%)	0.05 (76.8%)	0.06 (80.5%)	0.03 (65.2%)	0.04 (80.8%)
ER	0.5	0.1 (82.2%)	0.1 (83.9%)	0.09 (79.3%)	0.09 (78.8%)	0.06 (64.6%)	0.06 (75.4%)
	0.7	0.12 (59.0%)	0.14 (52.3%)	0.12 (55.7%)	0.13 (57.1%)	0.08 (25.1%)	0.09 (50.3%)
	0.3	0.02 (73.1%)	0.01 (-37.1%)	0.01 (-4.9%)	0.02 (64.8%)	0.0 (-204.1%)	0.01 (-22.0%)
GEO	0.5	0.04 (52.8%)	0.01 (12.4%)	0.01 (27.0%)	0.03 (56.3%)	0.01 (-145.1%)	0.02 (-9.7%)
	0.7	0.05 (66.5%)	0.02 (39.8%)	0.02 (42.6%)	0.04 (66.0%)	0.01 (-56.2%)	0.02 (0.9%)
	0.3	0.01 (82.6%)	0.0 (41.9%)	0.0 (25.6%)	0.01 (87.3%)	0.0 (-73.6%)	0.0 (11.8%)
Shape	0.5	0.02 (84.4%)	0.01 (67.7%)	0.01 (58.4%)	0.02 (87.4%)	0.0 (13.3%)	0.01 (43.8%)
	0.7	0.03 (85.2%)	0.01 (78.9%)	0.01 (58.2%)	0.02 (87.9%)	0.01 (43.6%)	0.01 (59.4%)
	0.3	0.03 (78.9%)	0.0 (-4.4%)	0.0 (-7.2%)	0.04 (73.7%)	0.0 (-253.3%)	0.01 (60.8%)
WS	0.5	0.05 (83.3%)	0.01 (-1.7%)	0.01 (38.6%)	0.05 (50.3%)	0.01 (40.9%)	0.01 (10.8%)
	0.7	0.07 (84.1%)	0.01 (56.4%)	0.01 (65.7%)	0.07 (89.5%)	0.01 (62.6%)	0.02 (68.6%)

Table 4.9. Loss: conductance difference. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage. † stands for out of memory error.

Dataset	Ratio	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
BA	0.3	0.11 (82.3%)	0.08 (78.5%)	0.10 (74.8%)	0.10 (74.3%)	0.09 (79.3%)	0.11 (83.6%)
	0.5	0.14 (69.6%)	0.13 (31.5%)	0.14 (37.4%)	0.14 (33.9%)	0.13 (34.3%)	0.13 (56.2%)
	0.7	0.22 (48.1%)	0.20 (11.0%)	0.21 (22.4%)	0.21 (20.0%)	0.20 (13.2%)	0.21 (47.8%)
ER	0.3	0.10 (81.0%)	0.09 (74.7%)	0.10 (74.3%)	0.10 (72.5%)	0.09 (76.4%)	0.12 (79.0%)
	0.5	0.13 (64.0%)	0.14 (33.8%)	0.14 (33.6%)	0.14 (32.5%)	0.14 (31.9%)	0.12 (1.4%)
	0.7	0.20 (43.4%)	0.19 (10.1%)	0.20 (17.6%)	0.20 (17.2%)	0.19 (23.4%)	0.17 (15.7%)
GEO	0.3	0.10 (91.2%)	0.09 (87.0%)	0.10 (84.8%)	0.10 (85.5%)	0.10 (84.6%)	0.11 (92.3%)
	0.5	0.12 (88.1%)	0.13 (33.9%)	0.13 (32.6%)	0.13 (37.6%)	0.13 (35.3%)	0.13 (90.1%)
	0.7	0.21 (86.7%)	0.17 (21.9%)	0.19 (25.2%)	0.19 (27.3%)	0.19 (27.8%)	0.11 (72.4%)
Shape	0.3	0.10 (82.3%)	0.10 (86.8%)	0.09 (85.8%)	0.09 (86.3%)	0.09 (84.8%)	0.09 (92.0%)
	0.5	0.14 (33.2%)	0.13 (34.7%)	0.13 (34.6%)	0.13 (37.7%)	0.13 (40.8%)	0.12 (89.8%)
	0.7	0.17 (41.4%)	0.19 (23.4%)	0.20 (27.7%)	0.20 (34.0%)	0.20 (34.3%)	0.11 (76.8%)
WS	0.3	0.10 (86.7%)	0.09 (82.1%)	0.10 (84.3%)	0.10 (82.9%)	0.09 (81.9%)	0.10 (90.5%)
	0.5	0.13 (80.8%)	0.13 (31.2%)	0.13 (33.1%)	0.13 (27.7%)	0.13 (34.0%)	0.13 (86.5%)
	0.7	0.19 (45.3%)	0.19 (19.3%)	0.19 (27.0%)	0.19 (26.6%)	0.20 (27.1%)	0.11 (12.8%)
CS	0.3	0.11 (75.8%)	0.08 (86.8%)	0.12 (71.4%)	0.11 (62.6%)	0.11 (76.7%)	0.14 (87.9%)
	0.5	0.14 (48.3%)	0.12 (16.7%)	0.15 (50.0%)	0.11 (-7.2%)	0.11 (6.7%)	0.09 (9.6%)
	0.7	0.26 (40.1%)	0.22 (29.0%)	0.24 (35.0%)	0.24 (41.0%)	0.23 (35.2%)	0.17 (28.8%)
Physics	0.3	0.10 (81.7%)	0.07 (79.2%)	0.11 (73.6%)	0.10 (73.7%)	0.11 (79.0%)	0.13 (4.4%)
	0.5	0.13 (20.5%)	0.19 (39.7%)	0.15 (27.8%)	0.16 (31.7%)	0.15 (25.4%)	0.11 (-22.3%)
	0.7	0.24 (60.2%)	0.16 (26.1%)	0.23 (15.3%)	0.24 (16.5%)	0.23 (11.2%)	0.20 (35.9%)
PubMed	0.3	0.12 (42.8%)	0.10 (0.4%)	0.18 (3.6%)	0.18 (-0.2%)	0.19 (0.9%)	0.11 (26.4%)
	0.5	0.15 (19.7%)	0.19 (1.3%)	0.24 (-12.9%)	0.39 (3.7%)	0.39 (11.8%)	0.16 (16.0%)
	0.7	0.25 (27.3%)	0.33 (0.8%)	0.36 (0.0%)	0.31 (33.2%)	0.28 (35.3%)	0.23 (14.1%)
Flickr	0.3	0.11 (62.6%)	†	0.13 (52.5%)	0.13 (54.7%)	0.12 (74.2%)	0.16 (58.3%)
	0.5	0.09 (-34.5%)	†	0.15 (3.1%)	0.16 (3.4%)	0.15 (19.9%)	0.13 (-6.7%)
	0.7	0.19 (35.6%)	†	0.20 (6.0%)	0.28 (-3.1%)	0.29 (5.3%)	0.12 (-25.4%)

Table 4.10. Loss: Eigenerror. Laplacian: combinatorial Laplacian for original graphs and *doubly-weighted Laplacian* for coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage. † stands for out of memory error.

Dataset	Ratio	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
	0.3	0.19 (4.1%)	0.1 (5.4%)	0.12 (5.6%)	0.12 (5.0%)	0.03 (25.4%)	0.1 (-32.2%)
BA	0.5	0.36 (7.1%)	0.17 (8.2%)	0.22 (6.5%)	0.22 (4.7%)	0.11 (21.1%)	0.17 (-15.9%)
	0.7	0.55 (9.2%)	0.32 (12.4%)	0.39 (10.2%)	0.37 (10.9%)	0.21 (33.0%)	0.28 (-29.5%)
	0.3	0.46 (16.5%)	0.3 (56.9%)	0.11 (59.1%)	0.23 (38.9%)	0.0 (-347.6%)	0.0 (-191.8%)
CS	0.5	1.1 (18.0%)	0.55 (49.8%)	0.33 (60.6%)	0.42 (44.5%)	0.21 (75.2%)	0.0 (-154.2%)
	0.7	2.28 (16.9%)	0.82 (57.0%)	0.66 (53.3%)	0.73 (38.9%)	0.49 (73.4%)	0.34 (63.3%)
	0.3	0.48 (19.5%)	0.35 (67.2%)	0.14 (65.2%)	0.2 (57.4%)	0.0 (-521.6%)	0.0 (20.7%)
Physics	0.5	1.06 (21.7%)	0.58 (67.1%)	0.33 (69.5%)	0.35 (64.6%)	0.2 (79.0%)	0.0 (-377.9%)
	0.7	2.11 (19.1%)	0.88 (72.9%)	0.62 (66.7%)	0.62 (64.9%)	0.31 (70.3%)	0.01 (-434.0%)
	0.3	0.33 (20.4%)	†	0.16 (7.8%)	0.16 (9.1%)	0.02 (63.0%)	0.04 (-88.9%)
Flickr	0.5	0.57 (55.7%)	†	0.33 (20.2%)	0.31 (55.0%)	0.11 (67.6%)	0.07 (60.3%)
	0.7	0.86 (85.2%)	†	0.6 (32.6%)	0.57 (38.7%)	0.23 (92.2%)	0.21 (40.7%)
	0.3	0.56 (5.6%)	0.27 (13.8%)	0.13 (17.4%)	0.34 (10.6%)	0.06 (-0.4%)	0.0 (31.1%)
PubMed	0.5	1.25 (7.1%)	0.5 (15.5%)	0.51 (12.3%)	1.19 (-110.1%)	0.35 (-8.8%)	0.02 (60.4%)
	0.7	2.61 (8.9%)	1.12 (19.4%)	2.24 (-149.8%)	4.31 (-238.6%)	1.51 (-260.2%)	0.27 (75.8%)
	0.3	0.27 (-0.1%)	0.35 (0.4%)	0.15 (0.6%)	0.18 (0.5%)	0.01 (5.7%)	0.01 (-10.4%)
ER	0.5	0.61 (0.5%)	0.7 (1.0%)	0.35 (0.6%)	0.36 (0.2%)	0.19 (1.2%)	0.02 (0.8%)
	0.7	1.42 (0.8%)	1.27 (2.1%)	0.7 (1.4%)	0.68 (0.3%)	0.29 (3.5%)	0.33 (10.2%)
	0.3	0.78 (43.4%)	0.08 (80.3%)	0.09 (77.1%)	0.27 (82.2%)	0.01 (-524.6%)	0.1 (82.5%)
GEO	0.5	1.72 (50.3%)	0.16 (89.4%)	0.18 (91.2%)	0.45 (84.9%)	0.08 (55.6%)	0.2 (86.8%)
	0.7	3.64 (30.4%)	0.33 (86.0%)	0.25 (86.7%)	0.61 (93.0%)	0.15 (88.7%)	0.32 (79.3%)
	0.3	0.87 (55.4%)	0.12 (88.6%)	0.07 (56.7%)	0.29 (80.4%)	0.01 (33.1%)	0.09 (84.5%)
Shape	0.5	2.07 (67.7%)	0.24 (93.3%)	0.17 (90.9%)	0.49 (93.0%)	0.11 (84.2%)	0.2 (90.7%)
	0.7	4.93 (69.1%)	0.47 (94.9%)	0.27 (68.5%)	0.71 (95.7%)	0.25 (79.1%)	0.34 (87.4%)
	0.3	0.7 (32.3%)	0.05 (84.7%)	0.04 (58.9%)	0.44 (37.3%)	0.02 (75.0%)	0.06 (83.4%)
WS	0.5	1.59 (43.9%)	0.11 (88.2%)	0.11 (83.9%)	0.58 (23.5%)	0.1 (88.2%)	0.12 (79.7%)
	0.7	3.52 (45.6%)	0.18 (77.7%)	0.17 (78.2%)	0.79 (82.8%)	0.17 (90.9%)	0.19 (65.8%)

Table 4.11. Model comparison between MLP and GOREN . Loss: quadratic loss. Laplacian: combinatorial Laplacian for both original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage.

Dataset	Ratio	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
	0.3	0.27 (46.2%)	0.04 (4.1%)	0.04 (-38.0%)	0.43 (31.2%)	0.02 (-403.3%)	0.06 (67.0%)
WS + MLP	0.5	0.45 (62.9%)	0.09 (64.1%)	0.09 (15.9%)	0.52 (31.2%)	0.09 (31.6%)	0.11 (58.5%)
	0.7	0.65 (70.4%)	0.15 (57.6%)	0.14 (31.6%)	0.67 (76.6%)	0.15 (43.6%)	0.16 (54.0%)
	0.3	0.27 (46.2%)	0.04 (65.6%)	0.04 (-26.9%)	0.43 (32.9%)	0.02 (68.2%)	0.06 (75.2%)
WS + GOREN	0.5	0.45 (62.9%)	0.09 (82.1%)	0.09 (60.6%)	0.52 (51.8%)	0.09 (69.9%)	0.11 (84.2%)
	0.7	0.65 (73.4%)	0.15 (78.4%)	0.14 (66.7%)	0.67 (76.6%)	0.15 (80.8%)	0.16 (83.2%)
	0.3	0.13 (76.6%)	0.04 (-53.4%)	0.03 (-157.0%)	0.11 (69.3%)	0.0 (-229.6%)	0.04 (-7.9%)
Shape + MLP	0.5	0.23 (78.4%)	0.08 (-11.6%)	0.06 (67.6%)	0.17 (83.2%)	0.04 (44.2%)	0.08 (-1.9%)
	0.7	0.34 (69.9%)	0.17 (85.1%)	0.1 (73.5%)	0.24 (65.8%)	0.09 (74.3%)	0.13 (85.1%)
	0.3	0.13 (86.8%)	0.04 (79.8%)	0.03 (69.0%)	0.11 (69.7%)	0.0 (1.3%)	0.04 (73.6%)
Shape + GOREN	0.5	0.23 (91.4%)	0.08 (89.8%)	0.06 (82.2%)	0.17 (88.2%)	0.04 (80.2%)	0.08 (79.4%)
	0.7	0.34 (91.1%)	0.17 (94.3%)	0.1 (74.7%)	0.24 (95.9%)	0.09 (64.6%)	0.13 (84.8%)

subgraphs have some common patterns such as the dumbbell shape graph. For PubMed, most subgraphs have tree-like structures, possibly due to the edge sparsity in the citation network.

In Section 4.7, we visualize the weight difference between coarsening algorithms with and without learning. We also plot the eigenvalues of coarse graphs, where the first 40 eigenvalues of the original graph are smaller than the coarse ones. After optimizing edge weights via GOREN, we see both methods produce graphs with eigenvalues closer to the eigenvalues of the original graphs.

4.8 Related Work

Graph sparsification. Graph sparsification is firstly proposed to solve linear systems involving combinatorial graph Laplacian efficiently. [138, 136] showed that for any undirected graph G of N vertices, a spectral sparsifier of G with only $O(Nlog^cN/\varepsilon^2)$ edges can be constructed in nearly-linear time. [1] Later on, the time complexity and the dependency on the number

[1]The algorithm runs in $O(M.\text{polylog}N)$ time, where M and N are the numbers of edges and vertices.

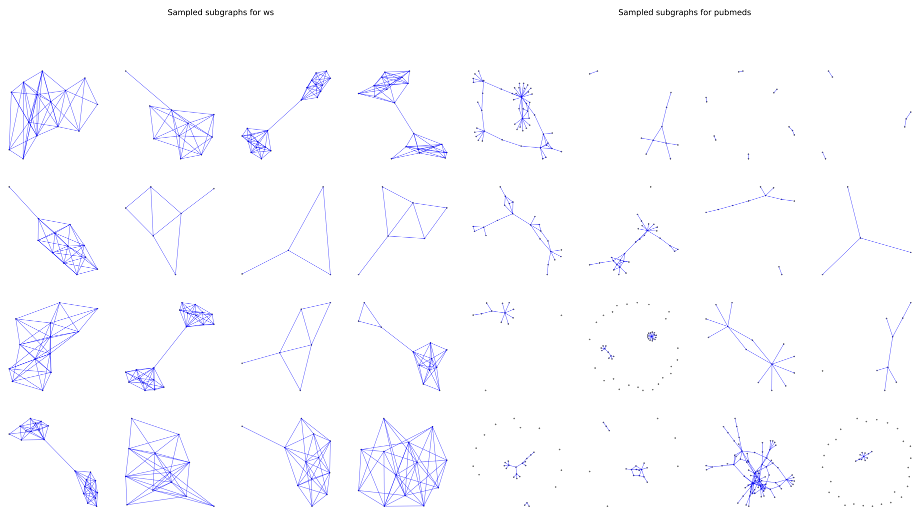


Figure 4.2. A collection of subgraphs corresponding to edges in coarse graphs (WS and PubMed) generated by variation neighborhood algorithm. Reduction ratio is 0.7 and 0.9 respectively.

of the edges are reduced by various researchers [7, 2, 93, 92].

Graph coarsening. Previous work on graph coarsening focuses on preserving different properties, usually related to the spectrum of the original graph and coarse graph. [105, 104] focus on the restricted spectral approximation, a modification of the spectral similarity measure used for graph sparsification. [62] develop a probabilistic framework to preserve inverse Laplacian.

Deep learning on graphs. As an effort of generalizing convolution neural network to the graphs and manifolds, graph neural networks is proposed to analyze graph-structured data. They have achieved state-of-the-art performance in node classification [84], knowledge graph completion [130], link prediction [34, 57], combinatorial optimization [98, 81], property prediction [42, 154] and physics simulation [127].

Deep generative model for graphs. To generative realistic graphs such as molecules and parse trees, various approaches have been taken to model complex distributions over structures and attributes, such as variational autoencoder [135, 107], generative adversarial networks (GAN) [32, 167], deep autoregressive model [99, 161, 97], and reinforcement learning type approach [160]. [167] proposes a GAN-based framework to preserve the hierarchical community structure

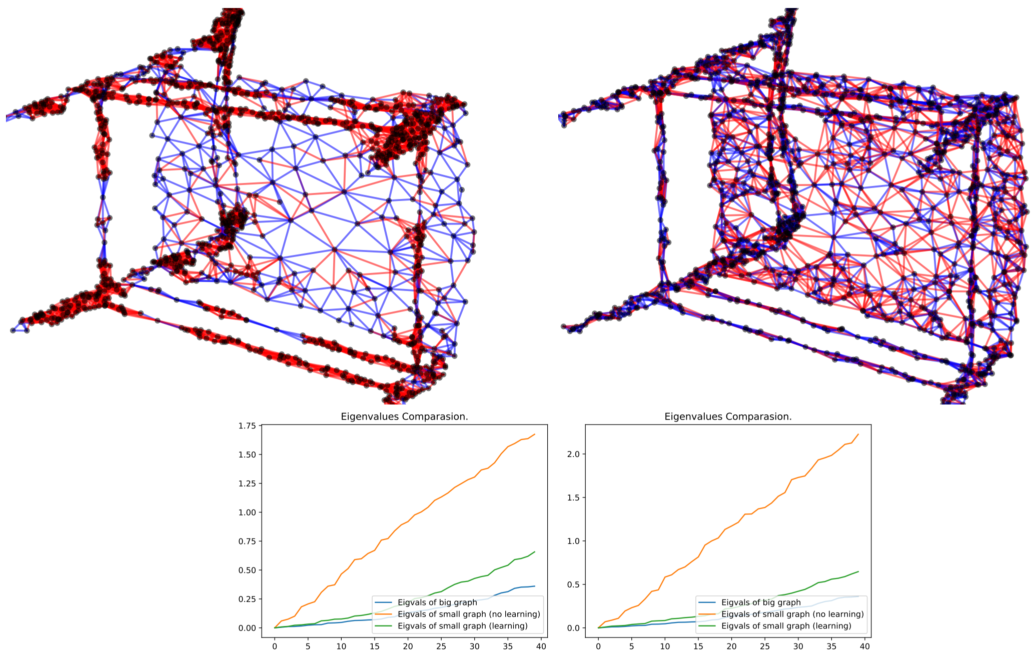


Figure 4.3. The first row illustrates the weight difference for two coarsening methods. Blue (red) edges denote edges whose learned weights is smaller (larger) than the default ones. The second row shows the spectrum of the original graph Laplacian, coarse graph w/o learning, and coarse graph w/ learning.

via algebraic multigrid method during the generation process. However, different from our approach, the coarse graphs in [167] are not learned.

4.9 Concluding Remarks

We present a framework to compare original graph and the coarse one via the properly chosen Laplace operators and projection/lift map. Observing the benefits of optimizing over edge weights, we propose a GNN-based framework to learn the edge weights of coarse graph to further improve the existing coarsening algorithms. Through extensive experiments, we demonstrate that our method `GOREN` significantly improves common graph coarsening methods under different metrics, reduction ratios, graph sizes, and graph types.

4.10 Missing Proofs

4.10.1 Choice of Laplace Operator

Laplace Operator on Weighted Simplicial Complex

Its most general form in the discrete case, presented as the operators on weighted simplicial complexes, is:

$$\mathscr{L}_i^{up} = W_i^{-1} B_i^T W_{i+1} B_i \quad \mathscr{L}_i^{down} = B_{i-1} W_{i-1}^{-1} B_{i-1}^T W_i$$

where B_i is the matrix corresponding to the coboundary operator δ_i, and W_i is the diagonal matrix representing the weights of i-th dimensional simplices. See [66] for details. When restricted to the graph (1 simplicial complex), we recover the most common graph Laplacians as special case of $\mathscr{L}_0^{up}$. Note that although the $\mathscr{L}_i^{up}$ and $\mathscr{L}_i^{down}$ is not symmetric, we can always symmetrize them by multiple a properly chosen diagonal matrix and its inverse from left and right without altering the spectrum.

Missing Proofs

We provide the missing proofs regarding the properties of the projection/lift map and the resulting operators on the coarse graph.

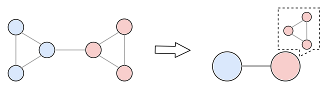


Figure 4.4. A toy example.

Recall as an toy example, a coarsening algorithm will take graph on the left in Section 4.10.1 and generate a coarse graph on the right, with coarsening matrix $P = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix}$, $P^+ = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$, $\Gamma = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$,

$\Pi = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix}$. Π in general is a $N \times N$ block matrix of rank n. All entries in each block Π_j is equal to $\frac{1}{\gamma_j}$ where $\gamma_j = \left|\pi^{-1}(\hat{v}_j)\right|$.

Table 4.12. Depending on the choice of $\mathscr{F}$ (quantity that we want to preserve) and $\mathscr{O}_G$, we have different projection/lift operators and resulting $\mathscr{O}_{\widehat{G}}$ on the coarse graph.

Quantity $\mathscr{F}$ of interest	$\mathscr{O}_G$	Projection $\mathscr{P}$	Lift $\mathscr{U}$	$\mathscr{O}_{\widehat{G}}$	Invariant under $\mathscr{U}$
Quadratic form Q	L	P	P^+	Combinatorial Laplace $\widehat{L}$	$\mathsf{Q}_L(\mathscr{U}\hat{x}) = \mathsf{Q}_{\widehat{L}}(\hat{x})$
Rayleigh quotient R	L	$\Gamma^{-1/2}(P^+)^T$	$P^+\Gamma^{-1/2}$	Doubly-weighted Laplace $\widehat{\mathsf{L}}$	$\mathsf{R}_L(\mathscr{U}\hat{x}) = \mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x})$
Quadratic form Q	$\mathscr{L}$	$\widehat{D}^{1/2}PD^{-1/2}$	$D^{1/2}(P^+)\widehat{D}^{-1/2}$	Normalized Laplace $\widehat{\mathscr{L}}$	$\mathsf{Q}_{\mathscr{L}}(\mathscr{U}\hat{x}) = \mathsf{Q}_{\widehat{\mathscr{L}}}(\hat{x})$

We first make an observation about projection and lift operator, $\mathscr{P}$ and $\mathscr{U}$.

Lemma 4.10.1. *$\mathscr{P} \circ \mathscr{U} = I$. $\mathscr{U} \circ \mathscr{P} = \Pi$.*

Proof. For the first case, it's easy to see $\mathscr{P} \circ \mathscr{U} = PP^+ = I$ and $\mathscr{U} \circ \mathscr{P} = P^+P = \Pi$.

For the second case, $\mathscr{P} \circ \mathscr{U} = \Gamma^{-1/2}(P^+)^T P^+ \Gamma^{-1/2} = \Gamma^{-1/2}\Pi\Gamma^{-1/2} = I$. $\mathscr{U} \circ \mathscr{P} = P^+\Gamma^{-1}(P^+)^T = I$.

For the third case,

$$\begin{aligned} \mathscr{P} \circ \mathscr{U} &= \widehat{D}^{1/2}PD^{-1/2}D^{1/2}(P^+)\widehat{D}^{-1/2} \\ &= \widehat{D}^{1/2}P(P^+)\widehat{D}^{-1/2} \\ &= \widehat{D}^{1/2}I\widehat{D}^{-1/2} = I. \end{aligned}$$

$$\begin{aligned}\mathscr{U}\circ\mathscr{P} &= D^{1/2}(P^{+})\widehat{D}^{-1/2}\widehat{D}^{1/2}PD^{-1/2}\\ &= D^{1/2}(P^{+})PD^{-1/2}\\ &= D^{1/2}\Pi D^{-1/2} = \Pi.\end{aligned}$$

□

Now we prove the three lemmas in the this chapter.

Proposition 4.10.2. *For any vector $\hat{x}\in\mathbb{R}^n$, we have that $\mathsf{Q}_{\widehat{L}}(\hat{x}) = \mathsf{Q}_L(P^{+}\hat{x})$. In other words, set $x := P^{+}\hat{x}$ as the lift of $\hat{x}$ in $\mathbb{R}^N$, then $\hat{x}^T\widehat{L}\hat{x} = x^T Lx$.*

Proof. $\mathsf{Q}_L(\mathscr{U}\hat{x}) = (\mathscr{U}\hat{x})^T L\mathscr{U}\hat{x} = \hat{x}(P^{+})^T LP^{+}\hat{x}^T = \hat{x}^T\widehat{L}\hat{x} = \mathsf{Q}_{\widehat{L}}(\hat{x})$ □

Proposition 4.10.3. *For any vector $x\in\mathbb{R}^n$, we have that $\mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x}) = \mathsf{R}_L(P^{+}\Gamma^{-1/2}\hat{x})$. That is, set the lift of $\hat{x}$ in $\mathbb{R}^N$ to be $x = P^{+}\Gamma^{-1/2}\hat{x}$, then we have that $\frac{\hat{x}^T\widehat{\mathsf{L}}\hat{x}}{\hat{x}^T\hat{x}} = \frac{x^T Lx}{x^T x}$.*

Proof. By definition $R_L(\mathscr{U}\hat{x}) = \frac{\mathsf{Q}_L(\mathscr{U}\hat{x})}{||\mathscr{U}\hat{x}||_2^2}$, $R_{\mathsf{L}}(x) = \frac{\mathsf{Q}_{\widehat{L}}(x)}{||x||_2^2}$. We will prove the lemma by showing $\mathsf{Q}_L(\mathscr{U}\hat{x}) = \mathsf{Q}_{\mathsf{L}}(x)$ and $||\mathscr{U}\hat{x}||_2^2 = ||x||_2^2$.

$$\begin{aligned}\mathsf{Q}_L(\mathscr{U}\hat{x}) &= (\mathscr{U}\hat{x})^T L\mathscr{U}\hat{x}\\ &= \hat{x}^T\Gamma^{-1/2}(P^{+})^T LP^{+}\Gamma^{-1/2}\hat{x}\\ &= \hat{x}^T\Gamma^{-1/2}\widehat{L}\Gamma^{-1/2}\hat{x}\\ &= \hat{x}^T\widehat{\mathsf{L}}\hat{x}\\ &= \mathsf{Q}_{\widehat{\mathsf{L}}}(\hat{x})\end{aligned}$$

$||\mathscr{U}\hat{x}||_2^2 = \hat{x}^T\Gamma^{-1/2}(P^{+})^T P^{+}\Gamma^{-1/2}\hat{x} = \hat{x}^T\hat{x} = ||\hat{x}||_2^2$. Since both numerator and denominator stay the same under the action of $\mathscr{U}$, we conclude $R_L(\mathscr{U}\hat{x}) = R_{\widehat{\mathsf{L}}}(\hat{x})$. □

Proposition 4.10.4. *For any vector $x\in\mathbb{R}^n$, we have that $\mathsf{Q}_{\widehat{\mathscr{L}}}(x) = \mathsf{Q}_{\mathscr{L}}(D^{1/2}P^{+}\widehat{D}^{1/2}x)$. That is, set the lift of $\hat{x}$ in $\mathbb{R}^N$ to be $x := D^{1/2}P^{+}\widehat{D}^{1/2}x$, then we have that $\hat{x}^T\widehat{\mathscr{L}}\hat{x} = x^T\mathscr{L}x$.*

Proof.

$$\begin{aligned}
\mathsf{Q}_{\mathscr{L}}(\mathscr{U}\hat{x}) &= (\mathscr{U}\hat{x})^T \mathscr{L}\mathscr{U}\hat{x} \\
&= \hat{x}\widehat{D}^{-1/2}(P^+)^T D^{1/2}\mathscr{L}D^{1/2}(P^+)\widehat{D}^{-1/2}\hat{x} \\
&= \hat{x}\widehat{D}^{-1/2}(P^+)^T L(P^+)\widehat{D}^{-1/2}\hat{x} \\
&= \hat{x}\widehat{D}^{-1/2}\widehat{L}\widehat{D}^{-1/2}\hat{x} \\
&= \hat{x}\widehat{\mathscr{L}}\hat{x} = \mathsf{Q}_{\widehat{\mathscr{L}}}(\hat{x})
\end{aligned}$$

□

4.10.2 Iterative Algorithm for Spectrum Alignment

Problem Statement

Given a graph G and its coarse graph $\widehat{G}$ output by existing algorithm $\mathscr{A}$, ideally we would like to set edge weight of $\widehat{G}$ so that spectrum of $\widehat{L}$ (denoted as $\mathfrak{L}\mathbf{w}$ below) has prespecified eigenvalues $\boldsymbol{\lambda}$, i.e,

$$\begin{gathered}
\mathfrak{L}\mathbf{w} = U\,\mathrm{Diag}(\boldsymbol{\lambda})U^T \\
\text{subject to } \mathbf{w} \geq 0, U^T U = I
\end{gathered} \tag{4.4}$$

We would like to make an important note that in general, given a sequence of non decreasing numbers $0 = \lambda_1 \leq \lambda_2, ..., \lambda_n$ and a coarse graph $\widehat{G}$, it is not always possible to set the edge weights (always positive) so that the resulting eigenvalues of graph Laplacian of $\widehat{G}$ is $\{0 = \lambda_1, \lambda_2, ..., \lambda_n\}$. We introduce some notations before we present the theorem. The theorem is developed in the context of *inverse eigenvalue problem* for graphs [6, 64, 48], which aims to characterize the all possible sets of eigenvalues that can be realized by symmetric matrices whose sparsity pattern is related to the topology of a given graph.

For a symmetric real $n \times n$ matrix M, the graph of M is the graph with vertices $\{1, ..., n\}$ and edges $\big\{\{i, j\} \mid b_{ij} \neq 0 \text{ and } i \neq j\big\}$. Note that the diagonal of M is ignored in determining

$\mathscr{G}(M)$. Let S_n be the set of real symmetric $n \times n$ matrices. For a graph $\widehat{G}$ with n nodes, define $\mathscr{S}(\widehat{G}) = \left\{ M \in S_n \mid \mathscr{G}(M) = \widehat{G} \right\}$.

Theorem 4.10.5. *[6, 64] If T is a tree, for any $M \in \mathscr{S}(T)$, the diameter of T is less than the number of distinct eigenvalues of M.*

For any graph $\widehat{G}$, its Laplacian (both combinatorial and normalized Laplacian) belongs to $\mathscr{S}(\widehat{G})$, the above theorem therefore applies. In other words, given a tree T and given a sequence of non-decreasing numbers $0 = \lambda_1 \leq \lambda_2, ... \lambda_n$, as long as the number of distinct values in sequences is less than the diameter of T, then this sequence can not be realized as the eigenvalues of graph Laplacian of T, no matter how we set the edge weights.

Therefore Instead of looking for the a graph with exact spectral alignment with original graph, which is impossible for some nondecreasing sequences as illustrated by the theorem 4.10.5, we relax the equality in equation 4.4 by instead minimizing the $||\mathfrak{L}\mathbf{w} - U \operatorname{Diag}(\boldsymbol{\lambda}) U^T||_F^2$. We first present an algorithm for the complete graph $\widehat{G}$ of size n. This algorithm is essentially the special case of [88]. We then show relaxing $\widehat{G}$ from the complete graph to the arbitrary graph will not change the convergence result. Before that, we introduce some notations.

Notation

Definition 22. *The linear operator $\mathfrak{L} : \mathbf{w} \in \mathbb{R}_+^{\frac{n(n-1)}{2}} \to \mathfrak{L}\mathbf{w} \in \mathbb{R}^{n \times n}$ is defined as*

$$[\mathfrak{L}\mathbf{w}]_{ij} = \begin{cases} -w_{i+d_j} & i > j \\ [\mathfrak{L}\mathbf{w}]_{ji} & i > j \\ \sum_{i \neq j} [\mathfrak{L}\mathbf{w}]_{ij} & i = j \end{cases}$$

where $d_j = -j + \frac{j-1}{2}(2n - j)$

A toy example is given to illustrate the operators, Consider a weight vector $\mathbf{w} =$

$[w_1, w_2, w_3, w_4, w_5, w_6]^T$, The Laplacian operator $\mathfrak{L}$ on $\mathbf{w}$ gives

$$\mathfrak{L}\mathbf{w} = \begin{bmatrix} \sum_{i=1,2,3} w_i & -w_1 & -w_2 & -w_3 \\ -w_1 & \sum_{i=1,4,5} w_i & -w_4 & -w_5 \\ -w_2 & -w_4 & \sum_{i=2,4,6} w_i & -w_6 \\ -w_3 & -w_5 & -w_6 & \sum_{i=3,5,6} w_i \end{bmatrix}$$

Adjoint operator $\mathfrak{L}^*$ is defined to satisfy $\langle \mathfrak{L}\mathbf{w}, Y \rangle = \langle \mathbf{w}, \mathfrak{L}^* Y \rangle$.

Complete Graph Case

Recall our goal is to

$$\begin{aligned} \underset{\mathbf{w},U}{\text{minimize}} \quad & \left\| \mathfrak{L}\mathbf{w} - U \operatorname{Diag}(\boldsymbol{\lambda}) U^T \right\|_F^2 \\ \text{subject to} \quad & \mathbf{w} \geq 0, U^T U = I \end{aligned} \tag{4.5}$$

Algorithm 1: Iterative algorithm for edge weight optimization

```
Input: coarse graph $\widehat{G}$, error tolerance $\varepsilon$, iteration limit $T$
Output: coarse graph with new edge weights
1 Initialize $U$ as random element in orthogonal group $O(n, \mathbb{R})$ and $t = 0$.
2 while $\varepsilon$ is smaller than the threshold or $t > T$ do
3     Update $\mathbf{w}^{t+1}, U^{t+1}$ according to 4.8 and Lemma 4.10.8
4     Compute Error $\varepsilon$
5     $t = t + 1$
6 From $w^t$, output coarse graph with new edge weights.
```

where $\boldsymbol{\lambda}$ is the desired eigenvalues of the smaller graph. One choice of $\boldsymbol{\lambda}$ can be the first n eigenvalues of the original graph of size N. $\mathbf{w}$ and U are variables of size $n(n-1)/2$ and $n \times n$.

The algorithm proceeds by iteratively updating U and $\mathbf{w}$ while fixing the other one.

Update for w: It can be seen when U is fixed, minimizing $\mathbf{w}$ is equivalent to a non-negative quadratic problem

$$\underset{\mathbf{w}\geq 0}{\text{minimize}} \quad f(\mathbf{w}) = \frac{1}{2}\|\mathfrak{L}\mathbf{w}\|_F^2 - \mathbf{c}^T\mathbf{w} \tag{4.6}$$

which is strictly convex where $\mathbf{c} = \mathfrak{L}^*(U\,\text{Diag}(\boldsymbol{\lambda})U^T)$. It is easy to see that the problem is strictly convex. However, due the the non-negativity constraint for $\mathbf{w}$, there is no closed form solution. Thus we derive a majorization function via the following lemma.

Lemma 4.10.6. *The function $f(w)$ is majorized at w_t by the function*

$$g(\mathbf{w}|\mathbf{w}^t) = f(\mathbf{w}^t) + (\mathbf{w}-\mathbf{w}^t)^T\nabla f(\mathbf{w}^t) + \frac{L_1}{2}\left\|\mathbf{w}-\mathbf{w}^t\right\|^2 \tag{4.7}$$

where $\mathbf{w}^t$ is the update from previous iteration an $L_1 = \|\mathfrak{L}\|_2^2 = 2n$.

After ignoring the constant terms in 4.7, the majorized problem of 4.6 at $\mathbf{w}^t$ is given

$$\underset{\mathbf{w}\geq 0}{\text{minimize}} \quad g(\mathbf{w}|\mathbf{w}^t) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \boldsymbol{a}^T\mathbf{w}, \tag{4.8}$$

where $a = \mathbf{w}^t - \frac{1}{L_1}\nabla f(\mathbf{w}^t)$ and $\nabla f(\mathbf{w}^t) = \mathfrak{L}^*(\mathfrak{L}\mathbf{w}^t) - \mathbf{c}$

Lemma 4.10.7. *From the KKT optimality conditions we can easily obtain the optimal solution to 4.7 as*

$$\mathbf{w}^{t+1} = (\mathbf{w}^t - \frac{1}{L_1}\nabla f(\mathbf{w}^t))^+$$

where $(x)^+ := \max(x,0)$.

Update for U: When $\mathbf{w}$ is fixed, the problem of optimizing U is equivalent to

$$\begin{aligned} \underset{U}{\text{minimize}} \quad & \text{tr}(U^T\mathfrak{L}\mathbf{w}U Diag(\boldsymbol{\lambda})) \\ \text{subject to} \quad & U^TU = I \end{aligned} \tag{4.9}$$

It can be shown that the optimal U at iteration t is achieved by $U^{t+1} = \text{eigenvectors}(L_w)$.

Lemma 4.10.8. *From KKT optimality condition, the solution to 4.9 is given by* $U^{t+1} = eigenvectors(\mathfrak{L}\mathbf{w})$.

The following theorem is proved at [88].

Theorem 4.10.9. *The sequence* $(\mathbf{w}^t, U^t)$ *generated by Algorithm 1 converges to the set of KKT points of 4.5.*

Non-complete Graph Case

The only complication in the case of the non-complete graph is that $\mathbf{w}$ has only $|E|$ number of free variables instead of $\frac{n(n-1)}{2}$ variables as the case of the complete graph. We will argue that w will stay at the subspace of dimension $|E|$ during the iteration.

For simplicity, given a non-compete graph $\widehat{G} = (\widehat{V}, \widehat{E})$, let us denote $\hat{v} = [n] = \{1, 2, ..., n\}$ and each edge will be represented as (i, j) where $i > j$, and $i, j \in [n]$. It is easy to see that we can map each edge (i, j) $(i > j)$ to k-th coordinate of $\mathbf{w}$ via $k = \Phi(i, j) = i - j + \frac{(j-1)(2p-j)}{2}$.

Let us denote $\overline{\mathbf{w}}$ (to emphasize its dependence on $\widehat{G}$, it is also denoted as $\mathbf{w}_{\widehat{G}}$ later.) to be the same as $\mathbf{w}$ on coordinates that corresponds to edges in G and 0 for the rest entries. In other words,

$$\overline{\mathbf{w}}[k] = \begin{cases} \mathbf{w}[k] & \text{if } \Phi^{-1}(k) \in E \\ 0 & \text{o.w.} \end{cases}$$

Similarly, for any symmetric matrix A of size $n \times n$

$$\overline{A}[i, j] = \begin{cases} A[i, j] & \text{if}(i, j) \in E \text{ or } (j, i) \in E \\ 0 & \text{o.w.} \end{cases}$$

Let us also define a $\widehat{G}$-subspace of $\mathbf{w}$ (denoted as $\widehat{G}$-subspace when there is no ambiguity) as $\{\overline{\mathbf{w}} | \mathbf{w} \in \mathbb{R}_+^{n(n-1)/2}\}$. What we need to prove is that if we initialize the algorithm with $\mathbf{w}_{\widehat{G}}$ instead of $\mathbf{w}$, $\mathbf{w}^t_{\widehat{G}}$ will remain in the $\widehat{G}$-subspace of $\mathbf{w}$ for any $t \in \mathbb{Z}_+$.

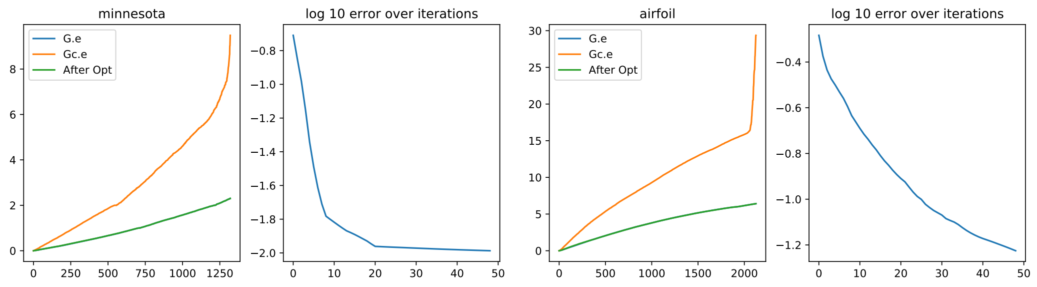


Figure 4.5. After optimizing edge weights, we can construct a smaller graph with eigenvalues much closer to eigenvalues of original graph. $G.e$ and $Gc.e$ stand for the eigenvalues of original graph and coarse graph. After-Opt stands for the eigenvalues of graphs where weights are optimized.

First, we have the following lemma.

Lemma 4.10.10. *We have*

1. $\mathfrak{L}\overline{w} = \overline{\mathfrak{L}w}$.
2. $\langle \overline{\mathbf{w}_1}, \mathbf{w}_2 \rangle = \langle \mathbf{w}_1, \overline{\mathbf{w}_2} \rangle = \langle \overline{\mathbf{w}_1}, \overline{\mathbf{w}_2} \rangle$.
3. $\mathfrak{L}^*\overline{Y} = \overline{\mathfrak{L}^*Y}$

Proof. Lemma 1 and 2 can be proved by definition. Now we prove the last lemma. For any $\mathbf{w} \in \mathbb{R}_+^{\frac{n(n-1)}{2}}$ and $Y \in \mathbb{R}^{n\times n}$

$$\langle \mathbf{w}, \mathfrak{L}^*\overline{Y} \rangle = \langle \mathfrak{L}\mathbf{w}, \overline{Y} \rangle = \langle \overline{\mathfrak{L}\mathbf{w}}, Y \rangle = \langle \mathfrak{L}\overline{\mathbf{w}}, Y \rangle = \langle \overline{\mathbf{w}}, \mathfrak{L}^*Y \rangle = \langle \mathbf{w}, \overline{\mathfrak{L}^*Y} \rangle$$

where the fourth equation follows from the definition of $\mathfrak{L}^*$ and the other equations directly follows from the previous two lemmas. Therefore $\mathfrak{L}^*\overline{Y} = \overline{\mathfrak{L}^*Y}$. □

Recall that we minimize the following objective when updating $\mathbf{w}_{\widehat{G}}$

$$\underset{\mathbf{w}_{\widehat{G}} \geq 0}{\text{minimize}} \quad \left\| \mathfrak{L}\mathbf{w}_{\widehat{G}} - U \,\text{Diag}(\boldsymbol{\lambda}) U^T \right\|_F^2$$

which is equivalent to be

$$\underset{\mathbf{w}_{\widehat{G}}\geq 0}{\text{minimize}} \quad \left\| \mathfrak{L}\mathbf{w}_{\widehat{G}} - \overline{U\,\text{Diag}(\boldsymbol{\lambda})U^T} \right\|_F^2 \tag{4.10}$$

Now following the same process for the case of complete graph. Equation 4.10 is equivalent to

$$\underset{\mathbf{w}_{\widehat{G}}\geq 0}{\text{minimize}} \quad f(\mathbf{w}_{\widehat{G}}) = \frac{1}{2}\|\mathfrak{L}\mathbf{w}_{\widehat{G}}\|_F^2 - \mathbf{c}^T\mathbf{w}_{\widehat{G}}$$

where $\mathbf{c} = \mathfrak{L}^*(\overline{U\,\text{Diag}(\boldsymbol{\lambda})U^T})$.

Use the same majorization function as the case of complete graph, we can get the following update rule

Lemma 4.10.11. *From the KKT optimality conditions we can easily obtain the optimal solution to as*

$$\mathbf{w}_{\widehat{G}}^{t+1} = (\mathbf{w}_{\widehat{G}}^t - \frac{1}{L_1}\nabla f(\mathbf{w}_{\widehat{G}}^t))^+$$

where $(x)^+ := \max(x,0)$ *and* $\nabla f(\mathbf{w}_{\widehat{G}}^t) = \mathfrak{L}^*(\mathfrak{L}\mathbf{w}_{\widehat{G}}^t - \overline{U\,\text{Diag}(\boldsymbol{\lambda})U^T})$.

Since $\nabla f(\mathbf{w}_{\widehat{G}}^t) = \mathfrak{L}^*(\mathfrak{L}\overline{\mathbf{w}^t}) - \overline{A}) = \mathfrak{L}^*(\overline{\mathfrak{L}\mathbf{w}^t} - \overline{A}) = \overline{\mathfrak{L}^*(\mathfrak{L}\mathbf{w}^t - A)}$ where $A = U\,\text{Diag}(\boldsymbol{\lambda})U^T$, therefore $\mathbf{w}_{\widehat{G}}^{t+1}$ will remain in the $\widehat{G}$-subspace if $\mathbf{w}_{\widehat{G}}^t$ is in the $\widehat{G}$-subspace. Since $\mathbf{w}_{\widehat{G}}^0$ is initialized inside $\widehat{G}$-subspace, by induction $\mathbf{w}_{\widehat{G}}^t$ stays in the $\widehat{G}$-subspace for any $t \in \mathbb{Z}^+$. Therefore, we conclude

Theorem 4.10.12. *In the case of non-complete graph, the sequence* $(\mathbf{w}^t, U^t)$ *generated by Algorithm 1 converges to the set of KKT points of 4.5.*

Remark: since for each iteration a full eigendecomposition is conducted, the computational complexity is $O(n^3)$ for each iteration, which is certainly prohibitive for large scale application. Another drawback is that the algorithm is not adaptive to the data so we have to run the same algorithm for graphs from the same generative distribution. The main takeaway of this algorithm is that it is possible to improve the spectral alignment of the original graph and coarse graph by optimizing over edge weights, as shown in Figure 4.5.

Chapter 5

Discussion and Future Directions

In this thesis, we discussed three works that offer a local-to-global perspective on graph neural networks. The first part of the thesis in Chapter 2 introduces a class of global GNN, the Invariant Graph Network (IGN), and provides a systematic study of its convergence property. The second part of the thesis in Chapter 3 studies the connection between local MPNNs and global Graph Transformers. It connects the local approach (MPNN) and global approach (Graph Transformer), with DeepSets and Invariant Graph Network (IGN) serving as the conceptual bridge. In the last part of the thesis at Chapter 4, we study the creative use of local MPNN to perform graph coarsening, a common subroutine used in modeling long-range interaction for large graphs. In the future, it would be interesting to explore the following directions.

Fine-grained understanding of the gap between MPNN and Graph Transformer. The approximation of the self-attention layer by MPNN establishes a link between MPNN + VN with Graph Transformer. However, it does not imply that MPNN + VN will achieve equivalent empirical performance as Graph Transformer. In fact, we still observe a gap between the two. We think such limitation shares a similarity with research in universal permutational invariant functions. Both DeepSets [162] and Relational Network [128] are universal permutational invariant architecture but there is still a representation gap between the two [169]. Under the restriction to analytic activation functions, one can construct a symmetric function acting on sets of size n with elements in dimension d, which can be efficiently approximated by the

Relational Network, but provably requires width exponential in n and d for the DeepSets. We believe a similar representation gap also exists between GT and MPNN + VN and leave the characterization of functions lying in the gap as future work.

Memory efficient global GNN. Both IGN and GT require large memory consumption and high computational complexity, which limits their application to cases such as fluid dynamics, sea temperature forecasting, and congestion prediction in chip design, where modeling long-range interaction is crucial. The current heuristic resorts to techniques in the efficient transformer literature that is agnostic to the graph structure. We believe understanding the continuous dynamics of the underlying phenomenon and leveraging such inductive bias in global GNN design will be a promising direction.

Topological Methods for Graph Learning. Topology is a useful tool to understand the global property of the data and naturally fits into the approach to building GNN. Topological methods [37, 24] such as persistence diagrams [63, 14, 22], multiparameter persistence modules [103, 17], simplicial complex networks [46] are promising tools to be integrated into future graph learning.

Graph Coarsening. For graph coarsening work in Chapter 4, there are three directions to pursue. first, the topology of the coarse graph is currently determined by the coarsening algorithm. It would be desirable to parametrize existing methods by neural networks so that the entire process can be trained end-to-end. Second, extending to other losses (maybe non-differentiable) such as [108] which involves inverse Laplacian remains interesting and challenging. Third, as there is no consensus on what specific properties should be preserved, understanding the relationship between different metrics and the downstream task is important.

Bibliography

[1] Marjan Albooyeh, Daniele Bertolini, and Siamak Ravanbakhsh. Incidence networks for geometric deep learning. *arXiv preprint arXiv:1905.11460*, 2019.

[2] Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 237–245, 2015.

[3] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[4] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

[5] Waïss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

[6] Francesco Barioli and Shaun Fallat. On two conjectures regarding an inverse eigenvalue problem for acyclic symmetric matrices. *The Electronic Journal of Linear Algebra*, 11, 2004.

[7] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[8] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, and Ryan Faulkner. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[9] Mikhail Belkin, Jian Sun, and Yusu Wang. Discrete laplace operator on meshed surfaces. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 278–287, 2008.

[10] Mikhail Belkin, Jian Sun, and Yusu Wang. Constructing laplace operator from point clouds in r^d. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1031–1040. SIAM, 2009.

[11] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021.

[12] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[14] Chen Cai. Sanity check for persistence diagrams. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*.

[15] Chen Cai, Yunfeng Cai, Mingming Sun, and Zhiqiang Xu. Group representation theory for knowledge graph embedding. *arXiv preprint arXiv:1909.05100*, 2019.

[16] Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. *arXiv preprint arXiv:2301.11956*, 2023.

[17] Chen Cai, Woojin Kim, Facundo Mémoli, and Yusu Wang. Elder-rule-staircodes for augmented metric spaces. *SIAM Journal on Applied Algebra and Geometry*, 5(3):417–454, 2021.

[18] Chen Cai, Nikolaos Vlassis, Lucas Magee, Ran Ma, Zeyu Xiong, Bahador Bahmani, Teng-Fong Wong, Yusu Wang, and WaiChing Sun. Equivariant geometric learning for digital rock physics: estimating formation factor and effective permeability tensors from morse graph. *International Journal for Multiscale Computational Engineering*, 21(5), 2023.

[19] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*, 2021.

[20] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

[21] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

[22] Chen Cai and Yusu Wang. Understanding the power of persistence pairing via permutation test. *arXiv preprint arXiv:2001.06058*, 2020.

[23] Chen Cai and Yusu Wang. Convergence of invariant graph networks. In *International Conference on Machine Learning*, pages 2457–2484. PMLR, 2022.

[24] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for classifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2573–2582. PMLR, 2019.

[25] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.

[26] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, 2011.

[27] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

[28] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[29] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, and Lukasz Kaiser. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[30] Fan RK Chung and Robert P Langlands. A combinatorial laplacian with vertex weights. *journal of combinatorial theory, Series A*, 75(2):316–327, 1996.

[31] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[32] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[33] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[34] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[35] Tamal K Dey, Pawas Ranjan, and Yusu Wang. Convergence, stability, and discrete approximation of laplace spectra. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 650–663. SIAM, 2010.

[36] Tamal Krishna Dey, Fengtao Fan, and Yusu Wang. Graph induced complex on point data. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 107–116, 2013.

[37] Tamal Krishna Dey and Yusu Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.

[38] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, and Sylvain Gelly. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[40] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685. PMLR, 2019.

[41] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

[42] 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.

[43] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[44] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv preprint arXiv:2206.08164*, 2022.

[45] Stéphane d'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pages 2286–2296. PMLR, 2021.

[46] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial neural networks. *arXiv preprint arXiv:2010.03633*, 2020.

[47] Justin Eldridge, Mikhail Belkin, and Yusu Wang. Graphons, mergeons, and so on! In *Advances in Neural Information Processing Systems*, pages 2307–2315, 2016.

[48] Shaun M Fallat, Leslie Hogben, Jephian C-H Lin, and Bryan L Shader. The inverse eigenvalue problem of a graph, zero forcing, and related parameters. *Notices of the American Mathematical Society*, 67(2), 2020.

[49] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[50] Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

[51] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020.

[52] Matan Gavish, Boaz Nadler, and Ronald R Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *ICML*, pages 367–374, 2010.

[53] Floris Geerts. The expressive power of kth-order invariant graph networks. *arXiv preprint arXiv:2007.12035*, 2020.

[54] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[55] David Gleich. Matlabbgl. a matlab graph library. *Institute for Computational and Mathematical Engineering, Stanford University*, 2008.

[56] Saket Gurukar, Priyesh Vijayan, Balaraman Ravindran, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, and Anasua Mitra. Benchmarking and analyzing unsupervised network representation learning and the illusion of progress.

[57] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, and Vedang Patel. Network representation learning: Consolidation and renewed bearing. *arXiv preprint arXiv:1905.00987*, 2019.

[58] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[59] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, and Yixing Xu. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 2022.

[60] David Harel and Yehuda Koren. A fast multi-scale method for drawing large graphs. In *International symposium on graph drawing*, pages 183–196. Springer, 2000.

[61] Bruce Hendrickson and Robert W Leland. A multi-level algorithm for partitioning graphs. *SC*, 95(28):1–14, 1995.

[62] Gecia Bravo Hermsdorff and Lee Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. In *Advances in Neural Information Processing Systems*, pages 7734–7745, 2019.

[63] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. *Advances in neural information processing systems*, 30, 2017.

[64] Leslie Hogben. Spectral graph theory and the inverse eigenvalue problem of a graph. *The Electronic Journal of Linear Algebra*, 14, 2005.

[65] Lars Holst. On the lengths of the pieces of a stick broken at random. *Journal of Applied Probability*, 17(3):623–634, 1980.

[66] Danijela Horak and Jürgen Jost. Spectra of combinatorial laplace operators on simplicial complexes. *Advances in Mathematics*, 244:303–336, 2013.

[67] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[68] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[69] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[70] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[71] Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665, 2022.

[72] EunJeong Hwang, Veronika Thost, Shib Sankar Dasgupta, and Tengfei Ma. An analysis of virtual nodes in graph neural networks for link prediction. In *Learning on Graphs Conference*, 2022.

[73] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[74] Hawoong Jeong, Sean P Mason, A-L Barabási, and Zoltan N Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.

[75] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*, 2021.

[76] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[77] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[78] Nicolas Keriven, Alberto Bietti, and Samuel Vaiter. Convergence and stability of graph convolutional networks on large random graphs. *arXiv preprint arXiv:2006.01868*, 2020.

[79] Nicolas Keriven, Alberto Bietti, and Samuel Vaiter. On the universality of graph neural networks on large random graphs. *arXiv preprint arXiv:2105.13099*, 2021.

[80] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32:7092–7101, 2019.

[81] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[82] Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *arXiv preprint arXiv:2207.02505*, 2022.

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

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

[85] Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. Surface networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2540–2548, 2018.

[86] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[87] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[88] Sandeep Kumar, Jiaxi Ying, Jose Vinicius de Miranda Cardoso, and Daniel Palomar. Structured graph learning via laplacian spectral constraints. In *Advances in Neural Information Processing Systems*, pages 11651–11663, 2019.

[89] Dan Kushnir, Meirav Galun, and Achi Brandt. Fast multiscale clustering and manifold identification. *Pattern Recognition*, 39(10):1876–1891, 2006.

[90] Stephane Lafon and Ann B Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1393–1403, 2006.

[91] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32:8572–8583, 2019.

[92] Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th annual acm sigact symposium on theory of computing*, pages 678–687, 2017.

[93] Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018.

[94] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, volume 6. Siam, 1998.

[95] Ron Levie, Wei Huang, Lorenzo Bucci, Michael Bronstein, and Gitta Kutyniok. Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research*, 22(272):1–59, 2021.

[96] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[97] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[98] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[99] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019.

[100] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

[101] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[102] Oren E Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

[103] David Loiseaux, Mathieu Carriere, and Andrew J Blumberg. Efficient approximation of multiparameter persistence modules. *arXiv preprint arXiv:2206.02026*, 2022.

[104] Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

[105] Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. *arXiv preprint arXiv:1802.07510*, 2018.

[106] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[107] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 7113–7124, 2018.

[108] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. Got: An optimal transport framework for graph comparison. In *Advances in Neural Information Processing Systems*, pages 13876–13887, 2019.

[109] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.

[110] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[111] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International conference on machine learning*, pages 4363–4371. PMLR, 2019.

[112] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

[113] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

[114] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[115] Wonpyo Park, Woong-Gi Chang, Donggeon Lee, and Juntae Kim. Grpe: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

[116] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[117] Robert Preis and Ralf Diekmann. Party-a software library for graph partitioning. *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71, 1997.

[118] Ronald Pyke. Spacings. *Journal of the Royal Statistical Society: Series B (Methodological)*, 27(3):395–436, 1965.

[119] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[120] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

[121] Alfréd Rényi. On the theory of order statistics. *Acta Mathematica Academiae Scientiarum Hungarica*, 4(3-4):191–231, 1953.

[122] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

[123] John W Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid methods*, pages 73–130. SIAM, 1987.

[124] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[125] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Graph neural networks: Architectures, stability, and transferability. *Proceedings of the IEEE*, 109(5):660–682, 2021.

[126] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020.

[127] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

[128] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

[129] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

[130] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[131] Nimrod Segol and Yaron Lipman. On universal equivariant set networks. *arXiv preprint arXiv:1910.02421*, 2019.

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

[133] Yu Shi, Shuxin Zheng, Guolin Ke, Yifei Shen, Jiacheng You, Jiyan He, Shengjie Luo, Chang Liu, Di He, and Tie-Yan Liu. Benchmarking graphormer on large-scale molecular modeling datasets. *arXiv preprint arXiv:2203.04810*, 2022.

[134] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2015.

[135] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422. Springer, 2018.

[136] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[137] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

[138] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[139] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 2020.

[140] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.

[141] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

[142] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[143] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[144] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *The Journal of Machine Learning Research*, 11:1201–1242, 2010.

[145] Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Michael A Osborne, and Ingmar Posner. Universal approximation of functions on sets. *Journal of Machine Learning Research*, 23(151):1–56, 2022.

[146] Chris Walshaw. A multilevel algorithm for force-directed graph drawing. In *International Symposium on Graph Drawing*, pages 171–182. Springer, 2000.

[147] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[148] Wujie Wang, Minkai Xu, Chen Cai, Benjamin Kurt Miller, Tess Smidt, Yusu Wang, Jian Tang, and Rafael Gómez-Bombarelli. Generative coarse-graining of molecular conformations. *arXiv preprint arXiv:2201.12176*, 2022.

[149] Max Wardetzky. Convergence of the cotangent formula: An overview. *Discrete differential geometry*, pages 275–286, 2008.

[150] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[151] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, and Morgan Funtowicz. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[152] Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

[153] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

[154] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

[155] Guoliang Xu. Discrete laplace–beltrami operators and their convergence. *Computer aided geometric design*, 21(8):767–784, 2004.

[156] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[157] Shijie Xu, Jiayan Fang, and Xiang-Yang Li. Weighted laplacian and its theoretical applications. *arXiv preprint arXiv:1911.10311*, 2019.

[158] Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

[159] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[160] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, pages 6410–6421, 2018.

[161] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[162] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[163] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[164] Jie Zhang, Chen Cai, George Kim, Yusu Wang, and Wei Chen. Composition design of high-entropy alloys with deep sets learning. *npj Computational Materials*, 8(1):89, 2022.

[165] Yuan Zhang, Elizaveta Levina, and Ji Zhu. Estimating network edge probabilities by neighborhood smoothing. *arXiv preprint arXiv:1509.08588*, 2015.

[166] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[167] Dawei Zhou, Lecheng Zheng, Jiejun Xu, and Jingrui He. Misc-gan: A multi-scale generative model for graphs. *Frontiers in Big Data*, 2:3, 2019.

[168] Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[169] Aaron Zweig and Joan Bruna. Exponential separations in symmetric neural networks. *arXiv preprint arXiv:2206.01266*, 2022.