Graph Neural Networks: Techniques and Applications

Zhiqian Chen

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science Applications

Chang-Tien Lu, Chair
Ing-Ray Chen
Narendran Ramakrishnan
Alireza Haghighat
Feng Chen

June 30, 2020
Falls Church, Virginia

Keywords: graph neural networks, deep learning, graph signal processing

Graph Neural Networks: Techniques and Applications

Zhiqian Chen

ABSTRACT

Effective information analysis generally boils down to the geometry of the data represented by a graph. Typical applications include social networks, transportation networks, the spread of epidemic disease, brain's neuronal networks, gene data on biological regulatory networks, telecommunication networks, knowledge graph, which are lying on the non-Euclidean graph domain. To describe the geometric structures, graph matrices such as adjacency matrix or graph Laplacian can be employed to reveal latent patterns. This thesis focuses on the theoretical analysis of graph neural networks and the development of methods for specific applications using graph representation. Four methods are proposed, including rational neural networks for jump graph signal estimation, RemezNet for robust attribute prediction in the graph, ICNet for integrated circuit security, and CNF-Net for dynamic circuit deobfuscation.

For the first method, a recent important state-of-art method is the graph convolutional networks (GCN) nicely integrate local vertex features and graph topology in the spectral domain. However, current studies suffer from drawbacks: graph CNNs rely on Chebyshev polynomial approximation which results in oscillatory approximation at jump discontinuities since Chebyshev polynomials require degree $\Omega(\text{poly}(1/\epsilon))$ to approximate a jump signal such as $|x|$. To reduce complexity, RatioanlNet is proposed to integrate rational function and neural networks for graph node level embeddings. For the second method, we propose a method for function approximation which suffers from several drawbacks: non-robustness and infeasibility issue; neural networks are incapable of extracting analytical representation; there is no study reported to integrate the superiorities of neural network and Remez. This work proposes a novel neural network model to address the above issues. Specifically, our method utilizes the characterizations of Remez to design objective functions. To avoid the infeasibility issue and deal with the non-robustness, a set of constraints are imposed inspired by the equioscillation theorem of best rational approximation. The third method proposes an approach for circuit security. Circuit obfuscation is a recently proposed defense mechanism to protect digital integrated circuits (ICs) from reverse engineering. Estimating the deobfuscation runtime is a challenging task due to the complexity and heterogeneity of graph-structured circuit, and the unknown and sophisticated mechanisms of the attackers for deobfuscation. To address the above-mentioned challenges, this work proposes the first graph-based approach that predicts the deobfuscation runtime based on graph neural networks. The fourth method proposes a representation for dynamic size of circuit graph. By analyzing SAT attack method, a conjunctive normal form (CNF) bipartite graph is utilized to characterize the complexity of this SAT problem. To overcome the difficulty in capturing the dynamic size of the CNF graph, an energy-based kernel is proposed to aggregate dynamic features.

Graph Neural Networks: Techniques and Applications

Zhiqian Chen

GENERAL AUDIENCE ABSTRACT

Graph data is pervasive throughout most fields, including pandemic spread network, social network, transportation roads, internet, and chemical structure. Therefore, the applications modeled by graph benefit people's everyday life, and graph mining derives insightful opinions from this complex topology. This paper investigates an emerging technique called graph neural newton (GNNs), which is designed for graph data mining.

There are two primary goals of this thesis paper: (1) understanding the GNNs in theory, and (2) apply GNNs for unexplored and values real-world scenarios.

For the first goal, we investigate spectral theory and approximation theory, and a unified framework is proposed to summarize most GNNs. This direction provides a possibility that existing or newly proposed works can be compared, and the actual process can be measured. Specifically, this result demonstrates that most GNNs are either an approximation for a function of graph adjacency matrix or a function of eigenvalues. Different types of approximations are analyzed in terms of physical meaning, and the advantages and disadvantages are offered. Beyond that, we proposed a new optimization for a highly accurate but low efficient approximation. Evaluation of synthetic data proves its theoretical power, and the tests on two transportation networks show its potentials in real-world graphs.

For the second goal, the circuit is selected as a novel application since it is crucial, but there are few works. Specifically, we focus on a security problem, a high-value real-world problem in industry companies such as Nvidia, Apple, AMD, etc. This problem is defined as a circuit graph as apply GNN to learn the representation regarding the prediction target such as attach runtime. Experiment on several benchmark circuits shows its superiority on effectiveness and efficacy compared with competitive baselines.

This paper provides exploration in theory and application with GNNs, which shows a promising direction for graph mining tasks. Its potentials also provide a wide range of innovations in graph-based problems.

To my wife, and my son.

Acknowledgments

First and foremost, I express my sincerest gratitude to my advisor and mentor, Dr. Chang-Tien Lu who has supported me throughout my Ph.D. research with patience and knowledge whilst allowing me sufficient flexibility to work in my own way. I attribute the level of my Ph.D. degree to his encouragement and effort. Beyond that, he has also been a great mentor in my career and life.

I am very thankful to all the committee members for their important guidance for the completion of my dissertation. Thank Dr. Naren Ramakrishnan for offering his fantastic leadership of our big project which supported my research. His deep expertise benefits my research work very much. Thank Dr. Ing-Ray Chen for providing insightful comments and suggestions, and his course regarding network also gave me with broad knowledge. Thank Dr. Alireza Haghighat for his unique perspective from different domain, and his course in Monte Carlo offered me the opportunity to harvest knowledge about sample methodology. Their knowledge of research pointed me in alternative directions that I might not recognize otherwise.

I would like to especially thank the enormous and continuous help from Dr. Feng Chen and Dr. Liang Zhao during these years on my research. With a great personality and research passion, they have always been examples of me.

I would like to express appreciation to my friends in the Spatial Data Management Laboratory: Kaiqun Fu, Taoran Ji, Lei Zhang, Fanglan Chen, Ting Hua, Yen-Cheng Lu, Xuchao Zhang, Jianfeng He, Shuo Lei, Aziz, Lulu, to name a few - with whom I have shared unique moments throughout this time. They made the journey enjoyable with many happy memories. Especially, I would like to express my thanks to Kaiqun, who helped me a lot from the first day I came to the U.S..

Finally, I would like to dedicate a special thanks to my parents for all of their love and support, and to my wife, Jie, for soothing my soul and taking care of our lovely son.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Effective information analysis generally boils down to the geometry of the data represented by a graph. Typical applications include social networks[71], transportation networks[10], spread of epidemic disease[104], brain's neuronal networks[97], gene data on biological regulatory networks[30], telecommunication networks[34], knowledge graph[85], which are lying on non-Euclidean graph domain. To describe the geometric structures, graph matrices such as adjacency matrix or graph Laplacian can be employed to reveal latent patterns.

In recent years, many problems are being revisited with deep learning tools. Convolutional neural networks(ConvNets) emerging in recent years are at the heart of deep learning, and the most prominent strain of neural networks in research. ConvNets have revolutionized computer vision[70], natural language processing[25], computer audition[43], reinforcement learning[101, 127], and many other areas. However, ConvNets are designed for grid data such as image, which belongs to the Euclidean domain. Graph data is non-Euclidean which makes it difficult to employ typical ConvNets. To bridge the gap, Bruna et al. [18][53] generalized spectral convolutional operation which requires expensive steps of spectral decomposition and matrix multiplication. Hammond et al.[50] first introduced truncated Chebyshev polynomial for estimating wavelet in graph signal processing. Based on this polynomial approximation, Defferrard et al.[32] designed ChebNet which contains a novel neural network layer for the convolution operator in the spectral domain. Kipf and Welling[65] simplified ChebNet by assuming the maximum of eigenvalues is 2 and fixing the order to 1, which boosts both effectiveness and efficiency. Li et al.[79] found that this simplified ChebNet is an application of Laplacian smoothing, which implies that current studies are only effective on the smooth signal.

This research focuses on the design of neural network approach for modeling graph data, and development of methods for application such as circuit deobfuscation. Specifically, we propose graph neural networks with rational function kernel to improve accuracy especially under challenging scenario, and a set of methodology for security defense under integrated circuit case. Details are described as follows:

- **Uniform Framework for Graph Neural Networks:** The success of deep learning has been widely recognized in many machine learning tasks during the last decades, ranging from image classification and speech recognition to natural language understanding. As an extension of deep learning, Graph neural networks (GNNs) are designed to solve the non-Euclidean problems on graph-structured data which can hardly be handled by general deep learning techniques. Existing GNNs under various mechanisms, such as random walk, PageRank, graph convolution, and heat diffusion, are designed for different types of graphs and problems, which makes it difficult to compare them directly. Previous GNN surveys focus on categorizing current models into independent groups, lacking analysis regarding their internal connection. This paper proposes a unified framework and provides a novel perspective that can widely fit existing GNNs into our framework methodologically. Specifically, we survey and categorize existing GNN models into the spatial and spectral domains, and reveal connections among subcategories in each domain. Further analysis establishes a strong link across the spatial and spectral domains.

- **Approximation for Graph Convolution:** In recent years, many problems are being revisited with deep learning tools. Convolutional neural networks(ConvNets) emerging in recent years are at the heart of deep learning, and the most prominent strain of neural networks in research. ConvNets have revolutionized computer vision[70], natural language processing[25], computer audition[43], reinforcement learning[101, 127], and many other areas. However, ConvNets are designed for grid data such as image, which belongs to the Euclidean domain. Graph data is non-Euclidean which makes it difficult to employ typical ConvNets. To bridge the gap, Bruna et al. [18][53] generalized spectral convolutional operation which requires expensive steps of spectral decomposition and matrix multiplication. Hammond et al.[50] first introduced truncated Chebyshev polynomial for estimating wavelet in graph signal processing. Based on this polynomial approximation, Defferrard et al.[32] designed ChebNet which contains a novel neural network layer for the convolution operator in the spectral domain. Kipf and Welling[65] simplified ChebNet by assuming the maximum of eigenvalues is 2 and fixing the order to 1, which boosts both effectiveness and efficiency. Li et al.[79] found that this simplified ChebNet is an application of Laplacian smoothing, which implies that current studies are only effective on the smooth signal.

- **Robust approximation for Graph Approximation:** The need for approximation arises due to that one has a limited set of data points and wants to determine the underlying functional form. A suitable approximation by simpler functions can save a considerable amount of computation. For example, the function value may be the result of many complex calculations, and it may be time-consuming to calculate each function value. By approximating closed form, one could obtain approximate function values much quicker. A recent example is graph convolutional networks[32, 65] which apply Chebyshev approximation to avoid expensive matrix decomposition and multiplication. Another benefit of simple representation is that some following manipulations(e.g., dif-

ferentiation or integration) can be performed easily. Also, the closed form approximation explores the direction of machine learning on small data because the traditional method only requires very few data to approximate the underlying function. Approximation techniques are concerned how approximate complex functions with simpler functions, quantitatively characterizing the errors. This is often done with polynomial or rational approximations. Polynomial function models have well-understood properties. However, they are notorious for oscillations between exact-fit values and have insufficient capacity in modeling asymptotic phenomena. On the contrary, rational function models can accept a much wider range of shape than polynomial family, and they are typically smoother and significantly less oscillatory than polynomial models. Furthermore, a rational function can model complicated structure with a fairly low degree in both the numerator and denominator. Theoretically, polynomials require degree $\Omega(\text{poly}(1/\epsilon))$ to approximate functions near singularities and on an unbounded domain, while rational functions only need $\mathcal{O}(\text{poly}\log(1/\epsilon))$ to achieve ϵ-close[84, 136].

- **Graph based Circuit Deobfuscation Runtime Estimation** Over the past years, many of the leading semiconductor companies have become fabless because the increasing costs and complexity confine them not to design, test, fabricate, and package ICs. The expense of making a new semiconductor fab was estimated to be the $5.0 billion [46, 159]. Due to the high cost of building, operating, managing, and maintaining silicon manufacturing facilities, many companies have always been fabless in recent years. The considerable high capital costs on semiconductor manufacturing motivate most companies to outsource their designed integrated circuits (ICs) to the contract foundries for fabrication. Despite the reduced cost and other benefits, this trend has led to ever-increasing security risks such as concerns of risks on IC counterfeiting, piracy and unauthorized overproduction by the contract foundries [46, 132]. The overall financial risk caused by such counterfeit and unauthorized ICs was estimated to be over $169 billion per year [96]. The major threats from the attackers arise from reverse engineering an IC by fully identifying its functionality by stripping it layer-by-layer and extracting the unveiling gate-level netlist. To prevent such reverse engineering, IC *obfuscation* techniques have been extensively researched in recent years [158]. The general idea is to obfuscate some gates in an IC so that their gate type cannot be determined by reverse engineering optically, yet they preserve the functionality same as the original gates.

- **CNF based Circuit Deobfuscation Runtime Estimation** SAT attacker techniques were highly effective until very recent progress of the attacking techniques based on logical attackers were invented and widely applied [37]. This is due to that there are limited types of gates (e.g., AND, OR, XOR) in IC, so the attackers can just brute force all the possible combinations of types for all obfuscated gates to find out the one that functions identically to the targeted IC. However, brute force is usually prohibitively expensive. More recently, efficient methods such as satisfiability (SAT) based attacks have been proposed which have attracted enormous attention [86]. The runtime of SAT attack to

reverse engineer the IC mostly depends on the complexity of the obfuscated IC, which can vary from milliseconds to days or years. Therefore, a successful obfuscation requires attackers a prohibitive amount of time (i.e., many years) to deobfuscate. However, gates to obfuscate come at a heavy cost in finance, power, and space, such trade-off forces us to search for optimal layout instead of purely increasing their quantity. Therefore, the best set of gates for being obfuscated maximizes the runtime for deobfuscating. Mathematically, the ICs deobfuscation problem can be considered equivalently as solving the Boolean satisfiability (SAT) of a conjunctive normal form (CNF). This research is to discover the connect pattern between CNF representation and runtime.

1.1 Research Issues

This research aims to investigate and develop neural networks based efficient and effective techniques for graph data. The major research issues are stated as follows:

- **Accurate approximation for graph convolution:** Current studies on graph ConvNet heavily rely on polynomial approximation, which makes it difficult to estimate jump signals. It is widely recognized that **(1)** polynomial approximation suffers from Gibbs phenomenon, which means polynomial function oscillate and overshoot near discontinuities[139]; **(2)** Applying a higher order of polynomials could dramatically reduce the oscillation, but also incurs an expensive computational cost. **(3)** Polynomials require degree $\Omega(\text{poly}(1/\epsilon))$ to approximate functions near singularities and on an unbounded domain, while rational functions only need $\mathcal{O}(\text{poly}\log(1/\epsilon))$ to achieve ϵ-close[84, 136]. However, it is non-trivial to apply rational approximation. Polynomial-based method can easily transfer the function on eigenvalues to the same function on graph Laplacian so that matrix multiplication by eigenvector can be avoided. It is not easy for rational approximation to do so due to the additional denominator.

- **Robust approximation for graph convolution:** The existing approximation for graph convolution still face several critical challenges: **(1) Traditional method for deriving closed form is non-robust.** As state of the art, Remez algorithm theoretically guarantees the optimality of rational approximation. Nevertheless, it requires to carefully choose the degree of the rational function. Otherwise, this algorithm often makes itself infeasible due to the singular coefficient matrix, or it may cause large approximation errors. **(2) General neural networks do not represent function using an analytical expression.** Deep neural networks have been shown to outperform previous state-of-the-art machine learning techniques due to the tremendous ability in approximation.Notwithstanding, its intrinsic property hinders itself from deducing a simple closed form, which makes it unable to enjoy the benefit of low computation complexity.**(3) There is no study found to combine Remez and deep learning for function approximation.** Without proper constraints, there is no guarantee for neural networks to approach to the optimal

approximation, while Remez derives an accurate expression only under very harsh conditions. Although deep neural networks and Remez are widely known for their different advantages in function approximation, we did not find any work that jointly exploits their abilities while remedying these defects.

- **Circuit Deobfuscation Runtime Estimation based on Circuit:** SAT based attack techniques were highly effective until very recent progress of the attacking techniques based on logical attackers were invented and widely applied [37]. This is due to that there are limited types of gates (e.g., AND, OR, XOR) in IC, so the attackers can just brute force all the possible combinations of types for all obfuscated gates to find out the one that functions identically to the targeted IC. However, brute force is usually prohibitively expensive. More recently, efficient methods such as satisfiability (SAT) based attacks have been proposed which have attracted enormous attention [86]. The runtime of the SAT attack to decrypt the IC mostly depends on the complexity of the obfuscated IC, which can vary from milliseconds to days or years. Therefore, a successful obfuscation requires attackers a prohibitive amount of time (i.e., many years) to deobfuscate. Moreover, gates to obfuscate come at a heavy cost in finance, power, and space; such trade-off forces us to search for optimal layout instead of purely increasing their quantity. Therefore, the best set of gates for being obfuscated maximizes the runtime for deobfuscating. However, until now it is still generally based on human heuristics or experience, which is seriously arbitrary and suboptimal [64]. This is because it is unable to "try and error" all the different ways of obfuscation, as there are millions of combinations to try and the runtime for each try (i.e., to run the attacker) can be days, weeks, or years. pre-estimate the runtime for all the different ways of obfuscation in order to find out the most time-consuming one.

- **Circuit Deobfuscation Runtime Estimation based on CNF:** This research topic is vastly under-explored because of its significant challenges: **1) Difficulty in characterizing the hidden and sophisticated algorithmic mechanism of attackers.** Over the recent years, a large number of deobfuscation methods have been proposed with various techniques [64]. In order to practically beat the attackers, methods with sophisticated theories, rules, and heuristics have been proposed and adopted. The behavior of such highly-nonlinear and strongly-coupling systems is prohibitive for conventional simple models (e.g., linear regression and support vector machine [13]) to characterize. **2) Difficulty in extracting determinant features from discrete and dynamic graph-structured ICs.** The inputs of the runtime estimation problem is the ICs with selected gates obfuscated. Conventional feature extraction methods are not intuitive to be applied to such type of varying-structured data without significant information loss. Hence, it is highly challenging to intactly formulate and seamlessly integrate them as mathematical forms that can be input to conventional machine learning models. **3) Requirement on high efficiency and scalability for deobfuscation runtime estimation.** The key to the defense against deobfuscation is the speed. The faster the defender can estimate the deobfuscation runtime for each candidate set of obfuscated gates, the more candidate sets the defender

can estimate, and hence the better the obfuscation effect will be. Moreover, the estimation speed of deobfuscation runtime must not be sensitive to different obfuscation strategies in order to make the defender strategy controllable.

1.2 Contributions

The major proposed research contributions can be stated as follows:

Accurate approximation for graph convolution:

- **Propose a neural network model based on rational function for recovering jump discontinuities**: To estimate the jump signal, our proposed method integrates rational approximation and spectral graph operation to avoid matrix multiplication by eigenvectors. For graph signal regression task, expensive matrix inversion can be circumvented by graph Fourier transform.
- **Develop an efficient algorithm for model parameters optimization**: Remez algorithm is theoretically optimal, but it is often not practical especially when approximating discrete signal. To alleviate this issue, the stopping rules of Remez algorithm are relaxed to initialize the neural networks parameters.
- **Provide theoretical analysis for the proposed method on jump signal**: For understanding the behaviors of polynomial and rational function on jump discontinuities, a uniform representation is proposed to analyze convergence rate regarding the order number theoretically.
- **Conducting extensive experiments for performance evaluations**[1]: The proposed method was evaluated on synthetic and real-world data. Experimental results demonstrate that the proposed approach runs efficiently and consistently outperforms the best of the existing methods.

Robust approximation for graph convolution:

- **Proposing a novel framework that integrates the strengths of rational approximation and neural networks**: A neural network framework, namely RemezNet, is proposed for approximating function that utilizes optimality characterization of best rational approximation. RemezNet takes advantage of the neural network in function approximation, to avoid the drawbacks of traditional Remez, such as infeasibility issue.
- **Developing a set of objectives for optimizing approximation performance**: To achieve optimal approximation, we enforce the equioscillation theorem and the minimax theorem derived from traditional Remez into the proposed model. Combining those constraints with general regression loss, the optimality of rational representation can be identified.

[1] https://github.com/aquastar/RationalGraphNet

- **Designing an efficient initialization for parameter update**: To expedite convergence of RemezNet, we proposed a special-purpose initialization policy that exploits equioscillation characterization and Padé normalization. This initialization shows significant superiority over state of the art initializations for normal neural networks.
- **Conducting extensive experiments for performance evaluations**: The proposed method was evaluated on benchmark functions with challenging singularity. Experimental results demonstrate that the proposed approach runs efficiently and consistently overcomes the disadvantage of traditional Remez.

Circuit based deobfuscation runtime estimation:

- **Proposing a new framework, ICNet, for deobfuscation runtime estimation based on graph deep learning**. We propose the first graph neural network for the circuit, where it is directly modeled using the gate connectivity. After learning the representation of the circuit, a neural network is trained to the relationship between its representation and the runtime.
- **Designing an interpretable component for extracting domain rules.** Utilizing the attention mechanism, the importance of features regarding runtime is derived during the learning process. Features are evaluated by importance ranking, and the only important features are kept to boost the efficiency with little accuracy loss.
- **Conducting systematical experimental evaluations and analyses on real-world datasets.** Standard benchmark with several circuits is used to evaluate the proposed method and competitive baselines. The result shows constant superiority of the proposed ICNet beyond the baselines.

CNF based circuit deobfuscation runtime estimation:

- **Formulating a graph learning framework for predicting deobfuscation runtime.** We formulate obfuscation runtime prediction as a graph learning problem. In the proposed method, a graph-based model is built by transforming obfuscated ICs into a CNF graph. The model then learns the relationship between the complexity of CNF graph and deobfuscation runtime.
- **Proposing a new framework, CNF-Net, for deobfuscation runtime estimation based on graph deep learning.** To model SAT-based deobfuscation, a CNF graph with multi-order is proposed to derive informative representations from obfuscated ICs. Such end-to-end deep graph regressor can automatically extract the discriminative features that are determinants to the estimation of the deobfuscation runtime to achieve accurate runtime prediction.

- **Designing an energy-based neural layer to process varying size of graph data.** To unify the dynamic topology of CNF graph, this work innovatively leverages the energy of restricted Boltzmann machine to quantify the complexity of CNF. The bipartivity of CNF graph is highly utilized to optimize the computational cost.
- **Conducting comprehensive experimental evaluations and analyses on multiple datasets.** The proposed method is compared with several state-of-the-art methods on six real-world benchmarks. The analyses of the performance and effectiveness demonstrate the advantage of our method.

1.3 Thesis Organization

The remainder of this research thesis is organized as follows. Chapter 2 focuses the an approximation technique for graph convolution, compares different kernels, designs a new function constraint, and presents experiments results and discussions. Chapter 3 describes the proposed robust framework for function approximation and presents effective parameter inference algorithm with deep neural networks. Chapter 4 presents the a novel task on circuit deobfuscation task based on graph modeling, and proposes a graph based neural network, and then proposes effective algorithm for parameter optimization. Chapter 5 develops a CNF based framework for estimating deobfuscation runtime, and then present a scalable algorithm for the model optimization. Chapter 6 illustrates the research plan for this thesis together with the schedule and current publications.

Chapter 2

Uniform Framework for Graph Neural Networks

This chapter first describes the fundamental concepts of graph neural network, including graph convolution, approximation theory and spectral graph theory. It then presents literature surveys on graph neural networks with examples. In this chapter, a generalized framework is proposed to summary main-strain graph neural networks.

2.1 Introduction

The effectiveness of deep learning [73] has been widely recognized in various machine learning tasks [117, 120, 55, 147, 92] during the last decades, achieving remarkable success on Euclidean data. Recent decades has witnessed a great number of emerging applications where effective information analysis generally boils down to the non-Euclidean geometry of the data represented by a graph, such as social networks [71], transportation networks [10], spread of epidemic disease [104], brain's neuronal networks [97], gene data on biological regulatory networks [30], telecommunication networks [34], and knowledge graph [85]. Such non-Euclidean problems on graph-structured data can hardly be handled by general deep learning techniques. Modeling data by the graph is challenging due to that graph data is irregular, i.e., each graph has a variable size of nodes, and each node in a graph has a different number of neighbors, rendering some operations such as convolutions not directly applicable to the graph structure. Recently, there has been increasing interest in extending deep learning for graph data. Inspired by the success of deep learning, ideas are borrowed from deep learning models to handle the intrinsic complexity of the graph. This rising trend attracts increasing interest in the machine learning community, and a large number of GNN models are developed based on various theories [19, 66, 32, 48, 7, 141].

Despite GNNs dominate graph representation learning in recent years, there is still a limited

understanding of their representational power and physical meaning. The lack of understanding of GNNs significantly hinders the comparison and improvement of state-of-the-art methods. This gap also makes it challenging to extend GNNs to many domains such as business intelligence or drug development, since black box models may be associated with uncontrollable risks. Therefore, there is a pressing need to demystify GNNs, which motivates researchers to explore a generalized framework for GNNs [153, 41, 160]. However, these works can only explain few GNNs, and the interpretation for the majority of GNNs is still missing.

There exist a large number of different mechanisms among current GNNs, such as random walk, Page Rank, attention model, low-pass filter, message passing, and so on. These methods can be classified into several coarse-grained groups [102, 49, 162, 165, 148] such as spectral [19, 66, 32] and spatial domain [48, 7, 141]. However, the current taxonomies fail to provide an understanding of the connections among different GNN models. Elucidating the underlying mechanisms of GNNs, and understanding connections among all types of GNNs is still at the forefront of GNNs research [153, 146, 81, 80]. This work is not trivial since the mechanisms behind existing GNNs are not inherently consistent, so their internal connection remains unclear. This gap incurs difficulty in understanding GNNs and comparing emerging methods. Previous surveys of GNNs [102, 49, 162, 165, 148] focus on categorizing current models into independent groups and expounding each group separately without analysis regarding their relationship.

The objective of this paper is to provide a unified framework to generalize GNNs, bridging the gap among the existing works in spatial and spectral domains which are currently deemed as independent. The main focus of this work is the connection among GNNs from a theoretical perspective, going beyond existing taxonomies and designing a new scheme for GNNs. Our research is unique in how it links present works of various categories of GNNs.

2.1.1 Graph Neural Networks in Spatial and Spectral Domain

The problem of graph neural networks has been extensively studied recently. However, the existence of various GNNs challenges the model choice, since they cannot be understood straightforwardly except empirical study on limited benchmarks . First, some GNNs are designed by spectral theory , and some others are proposed from spatial perspective. The inherent incompatibility between spectral and spatial methodologies makes it impossible to be compared. Second, even in each domain, models are very different, which prevents us from understanding their advantages and disadvantages.

Firstly, we briefly introduce the proposed framework, including spatial and spectral domains, and present their internal connection. Then detailed subcategories from the spatial and spectral domains are provided respectively, and several popular GNN examples are used to illustrate our taxonomies in each subcategory.

The focus on a unified framework offers to advance current understanding regarding how

GNNs work. This study aims to exploit a series of graph theory and spectral domain knowledge to explore the linkage between major categories such as spatial- and spectral-based methods. In this survey, we present a review of current research regarding graph neural networks in a dual domain perspective. The major contribution of this section is two-fold:

1. **Connecting physical meaning of spectral and spatial domains.** These unique properties of spectral- and spatial-based GNNs determines that the basic concepts, principles and physical meanings of them are substantially different from each other. Therefore an overview of basic concepts and properties of spectral- and spatial-based GNNs can facilitate a better understanding of the challenges, opportunities and necessity of applying GNN.

2. **Unifying spectral and spatial domains respectively.** The effectiveness of GNN on many applications has encouraged a large body of literature in extending GNN. On the one hand, GNNs in spectral and spatial domain can be explained respectively. On the other hand, the development of GNNs in two domains is highly imbalanced - some tasks are extensively studied, while others have not been sufficiently investigated. This ignorance makes current GNN studies separate from each other. For insufficiently studied GNNs, it is necessary to give formal definition and uniform perspective with promising research directions that can enrich current research. We will show that each domain can well unified using independent theory.

The organization this survey are summarized as follows:

1. We give an overview of basic concepts of graph neural networks in Section 2. We discuss methods to represent graph, spectral-base GNN, spatial-based GNN, and necessary fundamentals.

2. We classify the graph neural networks into spectral and spatial domains. From Section 3 to Section 5, we review well- studied GNN models in each domain with representative algorithms.

3. Graph neural networks is in the early stages of development. We discuss some representative model for each domain, including several that have not yet received sufficient attention in the literature.

The readers of this survey are expected to have some basic understanding of graph such as adjacency matrices, and graph Laplacian, graph signal processing techniques such as clustering and eigen-decomposition, deep learning techniques such as fully connected layers.

2.1.2 Related Surveys and Differences

A few comprehensive surveys about graph neural networks have been compiled recently [165, 162, 148, 17, 49]. However, most surveys focus on collecting recently emerging works, and dividing the literature into independent categories instead of analyzing their sharing and underlying mechanism. Specifically, a comprehensive survey provides a overview of different examples of graph neural network, categorizing them into spatial and spectral based methods [17]. A recent survey [165] provides taxonomy on graph types, training methods, propagation steps. [162] summarizes current progress of graph neural network into semi-supervised (graph convolution), unsupervised (graph auto-encoder), and recent advancement (graph recurrent neural network and graph reinforcement learning). [148] proposes a similar taxonomy which includes graph convolution, graph auto-encoder, graph recurrent neural network, and spatial-temporal graph neural networks.

There also exists many surveys about graph neural networks in a particular perspective. For example, a comprehensive and focused survey [74] is conducted on the field of graph neural networks with attention mechanism. Another example [115] offers a perspective that several graph neural networks with negative sampling can be unified into a matrix factorization framework in an analytical form. Another similar work [87] proposes a general view demonstrating equivalent between network embedding approaches and matrix factorization by two objectives: one is for similar nodes; the other is for distant nodes. One dedicated survey provides a unified paradigm for the systematic categorization and analysis on various existing heterogeneous network embeddings algorithms, creating four benchmark datasets with various properties and friendly interfaces for ten popular algorithms.

Compared to previous surveys which merely present independent categories in their taxonomies, our proposed framework aims to go across the boundary of several seemingly separate categories, bridging the gap between two major categories (i.e., spatial- and spectral-based methods) and connecting all subcategories by strong theoretical support. This survey is also generalization of several previous work including generalizing graph convolution networks as Laplacian smoothing [80], graph neural networks as Weisfeiler-Lehman graph [153].

2.2 Problem Setup and Preliminary

In this section, we outlines basic concepts, necessary preliminary, and problem setup of node embedding which is the major task in the literature.

[Graph] A graph is defined as $\mathcal{G} = (, \mathbb{E}, \mathbf{A})$, where is a set of n nodees, $\mathbb{E}$ represents edges. An entry $v_i \in$ denotes a node, $e_{i,j} = \{v_i, v_j\} \in \mathcal{E}$ indicates an edge between node i and j. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is defined by $\mathbf{A}_{i,j} = 1$ iff there is a link between node i and j. Graph signal or node attributes $\mathbf{X} \in \mathbb{R}^{N \times D}$ is a feature matrix with each entry $x_i \in \mathbf{X}$ representing the feature vector on node i. In this survey, we focus on node embeddings

Notations	Descriptions
$\mathcal{G}$	A graph.
	The set of nodes in a graph.
$\mathbb{E}$	The set of edges in a graph.
$\mathbf{A}, \tilde{\mathbf{A}}$	The adjacency matrix and its normalization.
$\mathbf{L}, \tilde{\mathbf{L}}$	The graph Laplacian matrix and its normalization.
v	A node $v \in$.
e_{ij}	An edge $e_{ij} \in \mathbb{E}$.
$\lambda_i \in \mathbf{\Lambda}$	Eigenvalue(s).
$\mathbf{U}, \mathbf{U}^\intercal$	Eigenvector matrix and its transpose.
$\mathbf{u}_i \in \mathbf{U}, \mathbf{u}^\intercal{}_i \in \mathbf{U}^\intercal$	Single eigenvector and its transpose.
$\mathbf{D}$	The degree matrix of $\mathbf{A}$.$\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$.
$\mathbf{X} \in \mathbf{R}^{N \times d}$	The feature matrix of a graph.
$\mathbf{Z} \in \mathbf{R}^{N \times b}$	New node feature matrix.
$\mathbf{H} \in \mathbf{R}^{N \times b}$	The node hidden feature matrix.
$\mathbf{h}_v \in \mathbf{R}^b$	The hidden feature vector of node v.
N	node number
b	dimension size of hidden feature
$\odot$	Element-wise product.
$\mathbf{\Theta}, \theta$	Learnable model parameters.
$\mathbf{P}(\cdot), \mathbf{Q}(\cdot)$	Polynomial function.
$\mathcal{N}(v)$	Directed neighbors of node v

Table 2.1: Commonly used notations

in undirected graph which is the most simple and common scenario. Learning node-level embeddings can be summarized as

$$\mathbf{Z} = f_\Theta(\mathcal{G}, \mathbf{X}), \tag{2.1}$$

where Θ indicates the parameters of the model. We aim to find a $f_\Theta(\cdot)$ which can integrate graph structure and original node attributes, outputting a new node embedding $\mathbf{Z}$. There is multiple options for $\mathcal{G}$ listed in Table 2.2.

In this survey, we represents graph using graph Laplacian or adjacency matrix, since there is no experimental or theoretical evidence showing any consistent advantages of each single filter listed Table 2.2 [144]. This survey focuses on two major categories, i.e., spectral and spatial methods, and two related definitions are listed below for understanding this paper.

[Spatial Method] Treating graph Laplacian $\mathbf{L}$ [22] as spatial connectivity among nodes, spatial method integrate $\mathbf{L}$ and signal $\mathbf{X}$:

$$\mathbf{Z} = f(\mathcal{G}, \mathbf{X}) = f(\mathbf{L}, \mathbf{X}). \tag{2.2}$$

Therefore, spatial methods focus on finding a function $f(\cdot)$.

[Spectral Method] Graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A} \in \mathbb{R}^{N \times N}$ where $\mathbf{D}$ is degree matrix. Due to its generalization ability [15] , the normalized Laplacian is defined as $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\,\mathbf{L}\,\mathbf{D}^{-\frac{1}{2}}$. The Laplacian $\mathbf{L}$ is diagonalized by the Fourier basis $\mathbf{U}^\intercal$ (i.e., graph Fourier transform) [125, 167]: $\tilde{\mathbf{L}} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal$ where $\mathbf{\Lambda}$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues (i.e., $\mathbf{\Lambda}_{ii} = \lambda_i$), and $\mathbf{U}$ is also called eigenvectors. The graph Fourier transform of a signal $\mathbf{X}$ residing on nodes is represented as $\hat{\mathbf{X}} = \mathbf{U}^\intercal\,\mathbf{X} \in \mathbb{R}^{N \times N}$ and the inverse graph Fourier transform is $\mathbf{X} = \mathbf{U}\,\hat{\mathbf{X}}$. A graph convolution operation is defined in the Fourier domain such that

$$f_1 * f_2 = \mathbf{U}\left[(\mathbf{U}^\intercal f_1) \odot (\mathbf{U}^\intercal f_2)\right], \tag{2.3}$$

where $\odot$ is the element-wise product, and f_1/f_2 are two signals defined on node domain. It follows that a node signal $f_2 = \mathbf{X}$ is filtered by spectral signal $\hat{f}_1 = \mathbf{U}^\intercal f_1 = \mathbf{g}$ as:

$$\mathbf{Z} = f(\tilde{\mathbf{L}}, \mathbf{X}) = \mathbf{U}\left[\mathbf{g}(\mathbf{\Lambda}) \odot (\mathbf{U}^\intercal\,\mathbf{X})\right] = \mathbf{U}\,\mathbf{g}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\,\mathbf{X} \tag{2.4}$$

where $\mathbf{g}$ is known as frequency response function of filter $\tilde{\mathbf{L}}$. Therefore, spectral methods is defined as learning $\mathbf{g}(\cdot)$.

In mathematics, approximation theory is concerned how to approximate complex functions using simpler functions, quantitatively characterizing the errors. One useful case of approximating function is to significantly reduce the computational overhead of operations on complex functions. This is typically done with polynomial or rational approximations. Polynomials are familiar and comfortable, but rational functions seem complex and specialized, and rational functions are more powerful than polynomials at approximating functions in

Notations	Descriptions
$\mathbf{A}$	Adjacency matrix
$\mathbf{L}$	Graph Laplacian
$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$	Adjacency with self loop
$\mathbf{D}^{-1}\mathbf{A}$	Random walk row normalized adjacency
$\mathbf{A}\mathbf{D}^{-1}$	Random walk column normalized adjacency
$\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$	Symmetric normalized adjacency
$\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$	Left renormalized adjacency, $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$
$\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}$	Right renormalized
$\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$	Symmetric renormalized)
$(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}})^k$	Powers of left renormalized adjacency
$(\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1})^k$	Powers of right renormalized adjacency

Table 2.2: Graph Representations

challenging cases. Basic properties of rational function are described in books of complex analysis [4, 139, 107, 114, 24, 112, 2, 168, 16, 98, 119]. As a popular and important polynomial approximation, Chebyshev approximation is first introduced as spectral filter for graph convolution: [Chebyshev Approximation for Graph Convolution] A real symmetric graph Laplacian $\mathbf{L}$ can be decomposed as $\mathbf{L} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^{-1} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal$. Chebyshev approximation on spectral filter $\mathbf{g}$ is applied [50, 32] so that:

$$
\begin{aligned}
&\mathbf{g} * x = \mathbf{U}\,\mathbf{g}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\,x \\
&\approx \mathbf{U}\sum_k \theta_k T_k(\tilde{\mathbf{\Lambda}})\,\mathbf{U}^\intercal\,x && (\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\mathbf{\Lambda} - \mathbf{I_N}) \\
&= \sum_k \theta_k T_k(\tilde{\mathbf{L}})x && (\mathbf{U}\,\mathbf{\Lambda}^k\,\mathbf{U}^\intercal = (\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)^k)
\end{aligned}
$$

A most popular graph convolutional network [66] further simplifies it:

$$
\begin{aligned}
&\mathbf{g} * x \\
\approx &\theta_0 \mathbf{I_N} x + \theta_1 \tilde{\mathbf{L}} x && (\text{expand to 1st order}) \\
= &\theta_0 \mathbf{I_N} x + \theta_1 (\frac{2}{\lambda_{max}} \mathbf{L} - \mathbf{I_N}) x && (\tilde{\mathbf{L}} = \tfrac{2}{\lambda_{max}} \mathbf{L} - \mathbf{I_N}) \\
= &\theta_0 \mathbf{I_N} x + \theta_1 (\mathbf{L} - \mathbf{I_N}) x && (\lambda_{max} = 2) \\
= &\theta_0 \mathbf{I_N} x - \theta_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} x && (\mathbf{L} = \mathbf{I_N} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \\
= &\theta_0 (\mathbf{I_N} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) x && (\theta_0 = -\theta_1) \\
= &\theta_0 (\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}) x && (\text{renormalization}: \tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_N}, \\
& && \tilde{\mathbf{D}}_{ii} = \textstyle\sum_j \mathbf{A}_{ij}),
\end{aligned}
$$

2.3 The proposed taxonomy

We provide a way to understand the spectral method in the spatial perspective and vice versa. A cross-domain perspective is introduced to integrate spatial and spectral methods into a unified framework. Figure 2.1 illustrates our framework, and corresponding research objectives are described in the following sections. The proposed framework categorizes GNNs into the spatial (A0) and spectral (B0) groups, each of which is further divided into three subcategories respectively. We illustrate the proposed framework in two aspects as follows.

2.3.1 Inside the Spatial and Spectral Domain

This subsection shows the structural relationship inside the spatial and spectral domain respectively. The spatial-based methods can be classified into three subcategories and there is specialization and generalization relationship among them:

(A1) Local Aggregation $\rightleftarrows$ (A2) Connectivity Order $\rightleftarrows$ (A3) Propagation Type,

where it is generalization from left to right, and specialization from right to left. Specifically, (A1) summarizes all methods that aggregate first order neighbors, (A2) contains approaches involving higher order neighbors, and (A3) covers (A1)/(A2) with bidirectional propagation. Therefore, (A1) is generalized as (A2) when involving higher order neighbors, and (A2) can be generalized as (A3) after adding bidirectional label propagation.

Similarly, the spectral-based methods are categorized into three subcategories:

(B1) Frequency Aggregation $\rightleftarrows$ (B2) Approximation Order $\rightleftarrows$ (B3) Approximation Type,

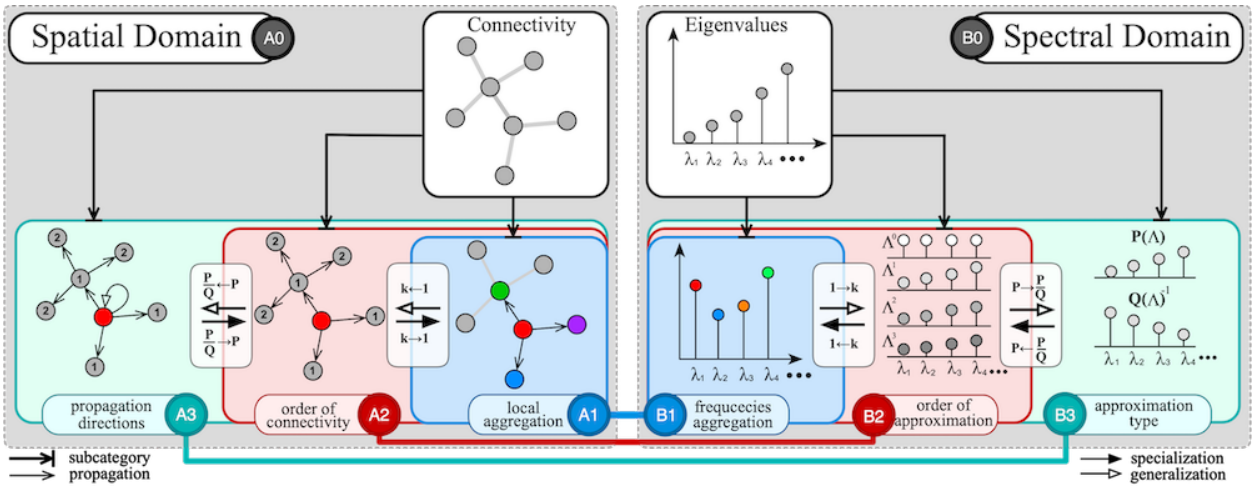


Figure 2.1: Illustration of major graph neural operations and their relationship. Spatial and spectral methods are divided into three groups, respectively. Group A1, A2, and A3 are strongly-correlated by generalization and specialization, so are group B1, B3, and B3. The equivalence relationship among them is marked in the same color.

where it is generalization from left to right, and specialization from right to left. Concretely, (B1) outlines all models that aggregate frequency components using linear approximation, while (B2) uses polynomial approximation, and (B3) covers rational approximation. Therefore, (B1) can be generalized as (B2) if replacing linear approximation with polynomial approximation, (B2) is generalized as (B3) if replacing polynomial approximation with rational approximation.

Sections 3 and 4 will discuss details of these two threads respectively and exemplify using several graph neural networks.

2.3.2 Between the Spatial and Spectral Domain

In this subsection, we elaborate the relationship across the boundary between the spatial and spectral domain. By transforming the analytical form of these subcategories, we found three relations of equivalence as below. First equivalence is between (A1) and (B1):

$$\text{(A1) } \underline{\text{Local Aggregation}} \rightleftarrows \text{(B1) } \underline{\text{Frequency Aggregation}},$$

which means that *Local Aggregation* adjusts weights on neighbors corresponds to adjusting weights on frequency components using linear function in *Frequency Aggregation*. (A2) and (B2) are the same in terms of actual operation, i.e.,

$$\text{(A2) } \underline{\text{Connectivity Order}} \rightleftarrows \text{(B2) } \underline{\text{Approximation Order}}.$$

This indicates that aggregating higher orders of neighbors in *Connectivity Order* can be rewritten as the sum of different orders of frequency components in *Approximation Order*. The last equivalence is

$$\text{(A3) } \underline{\text{Propagation Type}} \rightleftarrows \text{(B3) } \underline{\text{Approximation Type}},$$

in which *Propagation Type* defines a label propagation with bidirectional propagation, while *Approximation Type* adjusts filter function with rational approximation.

2.3.3 Pros and Cons

Since there exist equivalence relationships, we analyze the merits of three groups:

- **(A1) & (B1)**: This category considers the first order neighbors (i.e., directed neighbors), and aggregate the representations of neighbors by tuning the weights of each neighbor. In spectral perspective, it often applies linear filter function on eigenvalues with negative parameter, i.e., $\mathbf{g}(\mathbf{\Lambda}) = -\mathbf{\Lambda} + a$. This is also called low-pass filtering, since the low-frequency components are assigned with higher weight by g than its original value (i.e., eigenvalues). The major advantage of this category is (1) its low computational cost, and (2) many real-world scenarios subjects to local similarity assumption (i.e., neighbors are similar). The main drawback is that there is no guarantee that the local similarity exists in every network.
- **(A2) & (B2)**: Beyond the first order neighbors, this category involves higher-order neighbors, which increase the capacity in modeling a more complex relationship among the neighborhood. In terms of spectral perspective, it has theoretical superiority over (A1)/(B1), since (A2)/(B2) is a polynomial approximation as a spectral filter, while (A1)/(B1) is linear regression. Therefore, one shortcoming is also the price paid for its coverage on border neighbors, i.e., higher computational complexity than (A1)/(B1). Another defect is the over-smoothing issue (i.e., all nodes are similar) if the order is set too large, and there is no golden rule about which order is proper since it depends on the data. Note that K-layer (A1) or (B1) is equivalent to K-order of (A2)/(B2), so there is also an over-smooth issue if stacking K-layer (A1) or (B1).
- **(A3) & (B3)**: To alleviate the over-smooth issue, (A3)/(B3) introduce bi-directional propagation to limit the intensity of uni-directional propagation. This advantage can be explained as the superiority of rational approximation ((A3)/(B3)) over polynomial approximation ((A2)/(B2)). Specifically, rational approximation is more powerful and accurate especially in estimating some challenging signals such as discontinuity [139, 114, 24, 112, 2, 16, 98].

Therefore, we can summarize their pros and cons, as shown in Fig. 2.2: there is a trade-off between computational efficiency and generalization power, so the category selection lies in the data complexity and the efficiency requirement.

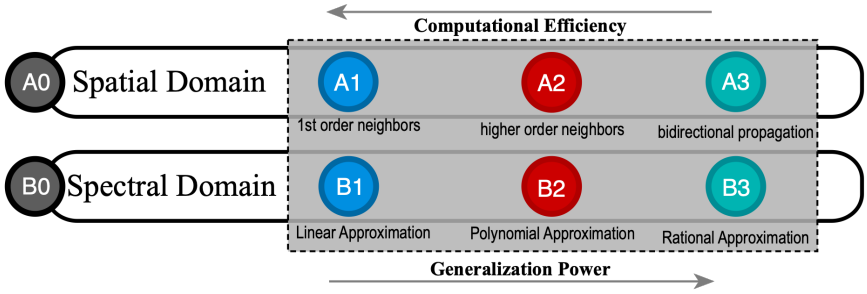


Figure 2.2: Comparison of each category.

2.4 Spatial-based GNNs (A0)

Several important aspects are often discussed in the existing literature of spatial methods, such as self-loop, normalization, high-order neighbors, aggregation, and combination among nodes. Based on these operations, we propose a new taxonomy of graph neural networks, categorizing spatial-based GNNs into three groups:

2.4.1 Local Aggregation (A1)

A number of works [111, 153, 152, 41, 48, 141] can be treated as learning the aggregation scheme among first order neighbors (i.e., direct neighbors). This aspect focuses on adjusting the weights for node and its neighbors to reveal the pattern regarding the supervision signal. Formally, updated node embeddings, $\mathbf{Z}(v)$, can be written as:

$$\mathbf{Z}(v_i) = \Phi(v_i)\,\mathbf{h}(v_i) + \sum_{u_j \in \mathcal{N}(v_i)} \Psi(u_j)\,\mathbf{h}(u_j), \tag{2.5}$$

where u_j denotes a neighbor of node v_i, $\mathbf{h}(\cdot)$ is their representations, and Φ/Ψ indicate the weight functions. First item on the right hand side denotes the representation of node v_i, while the second represents the update from its neighbors. Applying random walk normalization (i.e., dividing neighbors by degree of the current node), Eq. (2.5) can be written as:

$$\mathbf{Z}(v_i) = \Phi(v_i)\,\mathbf{h}(v_i) + \sum_{u_j \in \mathcal{N}(v_i)} \Psi(u_j)\frac{\mathbf{h}(u_j)}{d_i}, \tag{2.6}$$

or symmetric normalization:

$$\mathbf{Z}(v_i) = \Phi(v_i)\,\mathbf{h}(v_i) + \sum_{u_j \in \mathcal{N}(v_i)} \Psi(u_j)\frac{\mathbf{h}(u_j)}{\sqrt{d_i d_j}}, \tag{2.7}$$

where d_i represents the degree of node v_i. Normalization has better generalization capacity, which is not only due to some implicit evidence but also because of a theoretical proof on

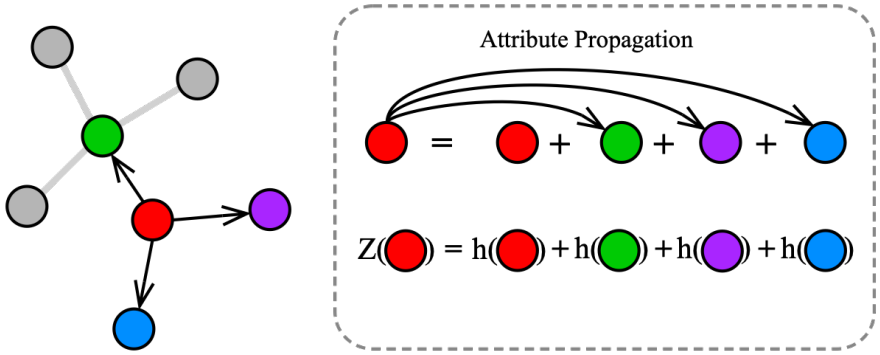


Figure 2.3: Illustration of A1

performance improvement [61]. In a simplified configuration, weights for the neighbors (Ψ) are the same. Therefore, they can be rewritten in matrix form as:

$$\mathbf{Z} = \phi\,\mathbf{X} + \psi\,\mathbf{D}^{-1}\,\mathbf{A}\,\mathbf{X} = (\phi\,\mathbf{I} + \psi\,\mathbf{D}^{-1}\,\mathbf{A})\,\mathbf{X} \tag{2.8}$$

or

$$\mathbf{Z} = \phi\,\mathbf{X} + \psi\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{X} = (\phi\,\mathbf{I} + \psi\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{-\frac{1}{2}})\,\mathbf{X}, \tag{2.9}$$

where ϕ and ψ are the weights. Eq. (2.8) and (2.9) can be generalized as the same form:

$$\mathbf{Z} = (\phi\,\mathbf{I} + \psi\tilde{\mathbf{A}})\,\mathbf{X} \tag{2.10}$$

where $\tilde{\mathbf{A}}$ denotes normalized $\mathbf{A}$, which could be implemented by random walk or symmetric normalization. As shown in Fig. 2.3, the new representation of the current node (in red) is updated as the sum of the previous representations of itself and its neighbors. A1 may adjust the weights of the neighbors.

Several state-of-the-art methods are selected to illustrate this schema:

Graph Convolutional Network (GCN) By simplifying Chebyshev polynomial for graph convolutional filter [32], GCN [66] adds a self-loop to nodes, and applies a *renormalization* trick which changes degree matrix from $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ to $\hat{\mathbf{D}}_{ii} = \sum_j (\mathbf{A} + \mathbf{I})_{ij}$. Specifically, GCN can be written as:

$$\mathbf{Z} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{X} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\hat{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{X} = (\mathbf{I} + \tilde{\mathbf{A}})\,\mathbf{X},$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\tilde{\mathbf{A}}$ is normalized adjacency matrix with self loop. Therefore, Eq. (2.4.1) is equivalent to Eq. (2.10) when setting $\phi = 0$ and $\psi = 1$ with the *renormalization* trick, and GCN takes the sum of each node and average of its neighbors as new node embeddings.

GraphSAGE Computing intermediate representations of each node and its neighbors, GraphSAGE [48] then applies a aggregation among its neighbors. Take mean aggregator as example, it averages a node with its neighbors by:

$$\mathbf{Z}(v_i) = \text{MEAN}\left(\{\mathbf{h}(v_i)\} \cup \{\mathbf{h}(u_j), \forall u_j \in \mathcal{N}(v_i)\}\right), \tag{2.11}$$

where $\mathbf{h}$ indicates the intermediate representation, and $\mathcal{N}$ denotes the neighbor nodes. Eq. (2.11) can be written in matrix form after implementing MEAN using symmetric normalization:

$$\mathbf{Z} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{X} = (\mathbf{I}+\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.12}$$

which is equivalent to Eq. (2.10) with $\phi = 1$ and $\psi = 1$. Note that the key difference between GCN and GraphSAGE is the normalization: the former is symmetric normalization and the latter is random walk normalization.

Graph Isomorphism Network (GIN) Inspired by Weisfeiler-Lehman (WL) test, GIN developed conditions to maximize the power of GNN, proposing a simple architecture, Graph Isomorphism Network (GIN). With strong theoretical support, GIN generalizes the WL test and [153] updates node representations as:

$$\mathbf{Z} = (1+\epsilon)\cdot\mathbf{h}(v) + \sum_{u_j\in\mathcal{N}(v_i)} \mathbf{h}_(\,u_j) = [(1+\epsilon)\,\mathbf{I}+\mathbf{A}]\,\mathbf{X}, \tag{2.13}$$

which is equivalent to Eq. (2.10) with $\phi = 1+\epsilon$ and $\psi = 1$. Note that **GIN** dose not perform normalization.

Graph Attention Model (GAT) GAT [141] applies attention mechanism by adjusting neighbors' weights, instead of using uniform weights in many related works:

$$\mathbf{Z} = (W_{att}\otimes\mathbf{A})\,\mathbf{X}, \tag{2.14}$$

where $W_{att}\in\mathbb{R}^{N\times N}$ is a matrix, and calculated by a forward neural network $W_{att}(i,j) = f(\mathbf{h}_i,\mathbf{h}_j)$ with a pair of node representations as input. MoNet [102] is similar to GAT, since its update follows:

$$\mathbf{Z}(v) = \sum_{u\in\mathcal{N}(v)} w_j(\mathbf{u}(\mathbf{h}_i,\mathbf{h}_j))\,\mathbf{h}_j \tag{2.15}$$

where $\mathbf{u}$ is a d-dimensional vector of pseudo-coordinates $\mathbf{u}(x,y)$, and

$$w_j(\mathbf{u}) = \exp\left(-\frac{1}{2}\left(\mathbf{u}-\boldsymbol{\mu}_j\right)^\top\boldsymbol{\Sigma}_j^{-1}\left(\mathbf{u}-\boldsymbol{\mu}_j\right)\right), \tag{2.16}$$

and μ_j are learnable d $\times$ d and d $\times$ 1 covariance matrix and mean vector of a Gaussian kernel, respectively. Let $W_{Monet} = w(u(\cdot))$ as a weight function of a pair of node representations representation, then it is also a attention model:

$$\mathbf{Z} = (W_{MoNet}\otimes\mathbf{A})\,\mathbf{X}, \tag{2.17}$$

These works do not consider updating nodes with their original representations, i.e., $\phi = 0$ and singular value ψ is replaced with matrix parameter W in Eq. (2.10). However, it is easy to extend them with self node.

2.4.2 Order of Connectivity (A2)

To collect richer local structure, several studies [7, 32, 146, 134, 45] involve higher orders of neighbors. Since direct neighbors (i.e., first-order neighbors) are not always sufficient for representing the node. On the other hand, large order usually averages all node representations, causing an over-smoothing issue and losing its focus on the local neighborhood [80]. This motivates many models to tune the aggregation scheme on different orders of neighbors. Therefore, proper constraint and flexibility of orders are critical for node representation. High order of neighbors has been proved to characterize challenging signal such as Gabor-like filters [1]. Formally, this type of work can be written as:

$$\mathbf{Z}(v_i) = \phi\,\mathbf{h}(v) + \overbrace{\sum_{u_j^{(1)} \in \mathbf{N}(v)} \psi_j^{(1)}\,\mathbf{h}(u_j^{(1)})}^{\text{1st order neighbor}} + \overbrace{\sum_{u_j^{(2)} \in \bigcup_j \mathbf{N}(u_j^{(1)})} \psi_j^{(2)}\,\mathbf{h}(u_j^{(2)})}^{\text{2nd order neighbor}} \quad ...+$$
$$\overbrace{\sum_{u_j^{(k+1)} \in \bigcup_j \mathbf{N}(u_j^{(k)})} \psi_j^{(k+1)}\,\mathbf{h}(u_j^{(k+1)})}^{\text{k-th order neighbor}} +..., \tag{2.18}$$

where $u_i^{(n)}$ indicates a n-th order neighbors of node v. Eq. (2.18) can be rewritten in matrix form:

$$\mathbf{Z} = (\phi\,\mathbf{I} + \sum_{j=1}^{k} \psi_j\,\mathbf{A}^j)\,\mathbf{X} = \mathbf{P}(\mathbf{A})\,\mathbf{X}, \tag{2.19}$$

where $\mathbf{P}(\cdot)$ is a polynomial function. Applying normalization, Equation (2.19) can be rewritten in matrix form as:

$$\mathbf{Z} = (\phi\,\mathbf{I} + \sum_{j=1}^{k} \psi_i(\mathbf{D}^{-\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{-\frac{1}{2}})^j)\,\mathbf{X} = (\phi\,\mathbf{I} + \sum_{i=1}^{k} \psi_i\,\tilde{\mathbf{A}}^i)\,\mathbf{X} \tag{2.20}$$
$$= (\sum_{i=0} \psi_i\,\tilde{\mathbf{A}}^i)\,\mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.21}$$

where $\phi = \psi_0$, and $\mathbf{A}$ could also be normalized by random walk normalization: $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\,\mathbf{A}$. As shown in Fig. 2.4, the new representation of the current node (in red) is updated as the sum of the previous representations of itself, its first and second order neighbors. A2 can adjust the weights among those representations. Several existing works are analyzed below, showing that they are variants of Eq. (2.19) or (2.21):

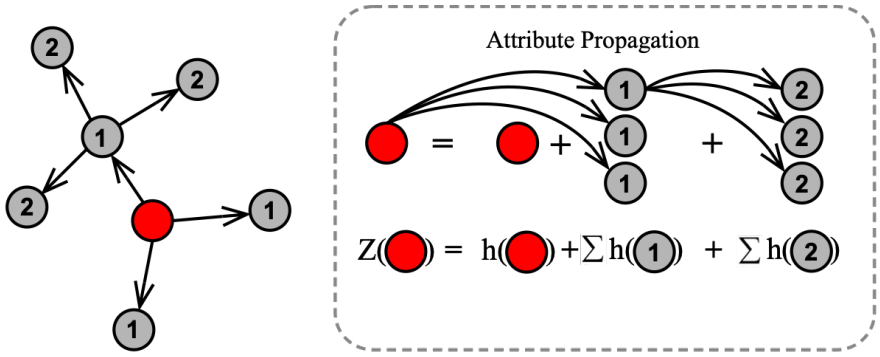


Figure 2.4: Illustration of A2.

ChebNet [50]

first introduced truncated Chebyshev polynomial for estimating wavelet in graph signal processing. Based on this polynomial approximation, Defferrard et al.[32] designed ChebNet which embeds a novel neural network layer for the convolution operator. Specifically, ChebNet is written as:

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\,\mathbf{X} = (\tilde{\theta_0}\,\mathbf{I} + \tilde{\theta_1}\tilde{\mathbf{L}} + \tilde{\theta_2}\tilde{\mathbf{L}}^2 + ...)\,\mathbf{X}, \tag{2.22}$$

where $T_k(\cdot)$ denotes the Chebyshev polynomial and θ_k is the Chebyshev coefficient. $\tilde{\theta}$ is the coefficient after expansion and reorganization. Since $\tilde{\mathbf{L}} = \mathbf{I} - \tilde{\mathbf{A}}$, we have:

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\,\mathbf{X} = [\tilde{\theta_0}\,\mathbf{I} + \tilde{\theta_1}(\mathbf{I} - \tilde{\mathbf{A}}) + \tilde{\theta_2}(\mathbf{I} - \tilde{\mathbf{A}})^2 + ...]\,\mathbf{X}, \tag{2.23}$$

which can be reorganized as:

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\,\mathbf{X} = (\phi\,\mathbf{I} + \sum_{i=1}^{k} \psi_i\,\tilde{\mathbf{A}}^i)\,\mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.24}$$

which is exactly Eq. (2.21)

DeepWalk [111] is a random walk based model that is integrated with deep learning technique. DeepWalk first draws a group of random paths from graph and applies a skip-gram algorithm to extract node features. Assuming the number of samples is large enough, then the transfer probability of random walk on a graph can be written as:

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1}\,\mathbf{A}, \tag{2.25}$$

with random walk normalization. Let the window size of skip-gram be $2t + 1$ and the current node is the (t+1)-th one, the farthest neighbor current node can reach is a t-th order one. If

the training is sufficient and samples are adequate, the node will converge to its neighbors. Therefore, the updated representation is as follows:

$$\mathbf{Z} = \frac{1}{t+1}(\mathbf{I} + \tilde{\mathbf{A}} + \tilde{\mathbf{A}}^2 + ... + \tilde{\mathbf{A}}^t)\,\mathbf{X} = \frac{1}{t+1}\,\mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X} \tag{2.26}$$

Diffusion convolutional neural networks (DCNN) [7] considers using a degree-normalized transition matrix, i.e., renormalized adjacency matrix: $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\,\mathbf{A}$:

$$\mathbf{Z} = W \odot \tilde{\mathbf{A}}^*\,\mathbf{X}, \tag{2.27}$$

where $\tilde{\mathbf{A}}^*$ denotes a tensor containing the power series of $\tilde{\mathbf{A}}$, and the $\odot$ operator represents element-wise multiplication. It can be transformed as:

$$\mathbf{Z} = (\psi_1\,\tilde{\mathbf{A}} + \psi_2\,\tilde{\mathbf{A}}^2 + \psi_3\,\tilde{\mathbf{A}}^3 + ...)\,\mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.28}$$

Node2Vec [45] defines a 2nd order random walk to control the balance between BFS (breath first search) and DFS (depth first search). If a random walk traverses edge (t, v) and stops at node v. The transition probabilities to next stop x from node v is defined as:

$$\alpha(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{2.29}$$

where d_{tx} denotes the shortest path between nodes t and x. d_{tx}=0 indicates a 2nd order random walk returns to its source node, (i.e., t), while d_{tx}=1 means that this walk goes to a BFS node, and d_{tx}=2 to a DFS node. The parameters p and q control the distribution of these three cases. Assuming the random walk is sufficiently sampled, Node2Vec can be rewritten in matrix form:

$$\mathbf{Z} = (\frac{1}{p} \cdot \overbrace{\mathbf{I}}^{\text{source}} + \overbrace{\tilde{\mathbf{A}}}^{\text{BFS}} + \frac{1}{q}\overbrace{(\tilde{\mathbf{A}}^2 - \tilde{\mathbf{A}})}^{\text{DFS}})\,\mathbf{X}, \tag{2.30}$$

which can be transformed and reorganized as:

$$\mathbf{Z} = [\frac{1}{p}\,\mathbf{I} + (1 - \frac{1}{q})\,\tilde{\mathbf{A}} + \frac{1}{q}\,\tilde{\mathbf{A}}^2]\,\mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.31}$$

where transition probabilities $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\,\mathbf{A}$ is random walk normalized adjacency matrix.

LINE[85] / SDNE[142] consider first order and second order neighbors as the constraints for learning node embeddings. first order: the nodes representation is forced to be similar to its neighbors, which is equivalent to:

$$\mathbf{Z} = \tilde{\mathbf{A}}\,\mathbf{X}\,. \tag{2.32}$$

Second order: the pair of nodes are forced to be similar if their neighbors are similar, which is equivalent to make 2nd order neighbors similar, therefore we can get the 2nd order connectivity by taking the power of the originial adjacency:

$$\mathbf{Z} = \tilde{\mathbf{A}}^2 \mathbf{X}. \tag{2.33}$$

Then the final learned node embeddings are formulated as:

$$\mathbf{Z} = \tilde{\mathbf{A}} \mathbf{X} + \alpha \tilde{\mathbf{A}}^2 \mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}}) \mathbf{X}. \tag{2.34}$$

Since LINE use concatenation between the representations constrained by first and second order, $\alpha = 1$; For SDNE, α is pre-defined.

Simple Graph Convolution (SGC) [146] removes non-linear function between neighboring graph convolution layers, and combine graph propagation in one single layer:

$$\mathbf{Z} = \tilde{\mathbf{A}}^K \mathbf{X} \tag{2.35}$$

where $\tilde{\mathbf{A}}$ is renormalized adjacency matrix, i.e., $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{A} \tilde{\mathbf{D}}^{-\frac{1}{2}}$, where $\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is degree matrix with self loop. Therefore, it can be easily rewritten as:

$$\mathbf{Z} = (0 \cdot \mathbf{I} + 0 \cdot \tilde{\mathbf{A}} + 0 \cdot \tilde{\mathbf{A}}^2 + ... + 1 \cdot \tilde{\mathbf{A}}^K) \mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}}) \mathbf{X} \tag{2.36}$$

Propagation Directions (A3) Most works merely consider label propagation from the node to its neighbors (i.e., gathering information form its neighbors) but ignore propagation in reverse direction. Reverse propagation means that labels or attributes can be propagated back to itself with probabilities, or restart propagating with a certain probability. This reverse behavior can avoid over-smoothing issue [67]. Note that [A2] can also alleviate over-smoothing issue by manually adjusting the order number, while [A3] can automatically fit the proper order number. Several works explicitly or implicitly implement reverse propagation by applying rational function on the adjacency matrix [21, 67, 81, 91, 60, 75, 12].

Since general label propagation is implemented by multiplying graph Laplacian, reverse propagation could be implemented by multiplying inverse graph Laplacian as:

$$\mathbf{Z} = \mathbf{P}(\tilde{\mathbf{A}}) \mathbf{Q}(\tilde{\mathbf{A}})^{-1} \mathbf{X} = \frac{\mathbf{P}(\tilde{\mathbf{A}})}{\mathbf{Q}(\tilde{\mathbf{A}})} \mathbf{X}, \tag{2.37}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are two different polynomial functions, and the bias of $\mathbf{Q}$ is often set to 1. As shown in Fig. 2.5, the new representations of the current node (in red) is updated the representations as the previous one with probability P, and as that of neighbors with probability (1-P). The difference of A3 beyond A2 is that A3 can avoid over-smooth issue since it can keeps part of the original representation no matter how many iteration it performs.

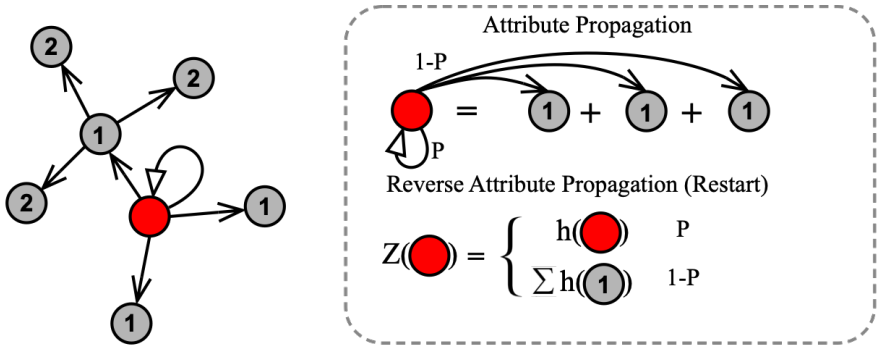


Figure 2.5: Illustration of A3.

Auto-Regressive label propagation (LP) [166, 163, 11] is a widely used methodology for graph-based learning. The objective of LP is two-fold: one is to extract embeddings that match with the label, the other is to become similar to neighboring vertices. The label can be treated as part of node attributes, so we have:

$$\mathbf{Z} = (\mathbf{I} + \alpha\, \tilde{\mathbf{L}})^{-1}\, \mathbf{X} = \frac{\mathbf{I}}{\mathbf{I} + \alpha(\mathbf{I} - \tilde{\mathbf{A}})}\, \mathbf{X} = \frac{\mathbf{I}}{(1+\alpha)\, \mathbf{I} - \alpha\, \tilde{\mathbf{A}}}\, \mathbf{X}, \tag{2.38}$$

which is equivalent to the form of Eq. (2.37), i.e., $\mathbf{P} = \mathbf{I}$ and $\mathbf{Q} = (1+\alpha)\, \mathbf{I} - \alpha\, \tilde{\mathbf{A}}$.

Personalized PageRank (PPNP) [67] can obtain node's representation via teleport (restart) probability α which is the ratio of keeping the original representation $\mathbf{X}$, i.e., no propagation. (1-α) is the ratio of performing the normal label propagation:

$$\mathbf{Z} = \alpha \left(\mathbf{I} - (1-\alpha)\, \tilde{\mathbf{A}}\right)^{-1} \mathbf{X} = \frac{\alpha}{\mathbf{I} - (1-\alpha)\, \tilde{\mathbf{A}}}\, \mathbf{X}, \tag{2.39}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\, \mathbf{A}$ is random walk normalized adjacency matrix with self loop. Eq. (2.58) is with a rational function whose numerator is a constant.

ARMA filter [12] utilize ARMA filter for approximating any desired filter response function, which can be written in the spatial domain as:

$$\mathbf{Z} = \frac{b}{\mathbf{I} - a\, \tilde{\mathbf{A}}}\, \mathbf{X}\,. \tag{2.40}$$

Note that ARMA filter is an unnormalized version of PPNP. When a+b=1, ARMA becomes PPNP.

RationalNet [21] proposes a general rational function and optimized by Remez algorithm, and the analytic form is exactly Eq. (2.37)

CayleyNets [75] applies rational filtering function in the complex domain.

Remark: The computational cost of [A3] is expensive since it involves the inverse of matrix. Typical solution is to apply iterative algorithms [12, 67, 75].

The major difference between rational spectral filter and polynomial spectral filter is whether there is polynomial of inverse graph Laplacian. The inverse of the graph Laplacian is a key object in online learning algorithms for graphs [54]. By using the concept of conductance, if f is a heat distribution over the vertexes, then $\mathbf{L}(f)$ indicates the flux induced by f over the graph. Then by the representer theorem [6, 122], namely that $f(\mathcal{V}_i) = \mathbf{L}^{-1}(\mathbf{L}(f))$ could be interpreted as saying that the heat at each vertex can be expressed concerning or derived from the flux through every vertex.

Thus, if $\mathbf{L}$ sends a heat distribution f over each node to flux through each vertex, then $\mathbf{L}^{-1}$ sends some of the fluxes over the graph back to the original heat distribution (i.e., keep part of fluxes itself). Going back to the graph learning application, we first translate our updated "heat distribution" to flux through all of those nodes by calculating $\mathbf{P}(\mathbf{L}(f))$. M-th degree of $\mathbf{P}(\cdot)$ means that each vertex can update M-th neighbors at most. Then using another updated flux in the reverse direction, $\mathbf{Q}(\mathbf{L}(f))^{-1}$ will adjust or reduce flux within N-th neighbors.

For polynomial spectral filter, more layers or higher degree involve more neighbors, and the model thereby should have better capacity. In practice, if the number of layers or degree is more than what is needed, it is unavoidable to be over-smooth (e.g., all nodes are similar). However, the best number of layer or degree is difficult to calculate unless trying all possible options. On the other hand, "sending back" behavior of rational spectral filter alleviates the over-smoothing issue [67], since it always limits the out-degree flux even if excessive graph convolutional layers or approximation degrees are added.

2.4.3 Connection among spatial methods

Three groups of spatial methods introduced above (i.e., A1, A2, A3) are strongly connected under *generalization* and *specialization* relationship, as shown in Fig. (2.1): **(1) Generalization:** *Local Aggregation* can be extended to *Order of Connectivity* by adding more neighbors of higher order. *Order of Connectivity* can be upgraded to *Propagation Direction* by adding reverse propagation; **(2) Specialization:** *Local Aggregation* is a special case of *Order of Connectivity* when setting the the order to 1. *Order of Connectivity* is a special case of *Propagation Direction* if removing reverse propagation.

2.5 Spectral-based GNNs (B0)

Spectral-based GNN models are built on spectral graph theory which applies eigen-decomposition and analyzes the weight-adjusting function (i.e., filter function) on eigenvalues of graph matrices. The weights yielded by filter function are assigned to frequency components (eigenvectors)

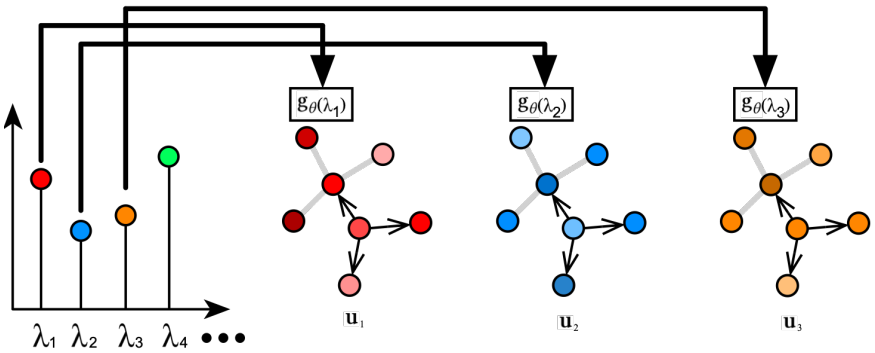


Figure 2.6: Illustration of B1.

for reconstructing the target signal. Based on spectral operation, we propose a new taxonomy of graph neural networks, categorizing spectral-based GNNs into three subgroups:

2.5.1 Frequency Aggregation (B1)

There exist numerous works that can be boiled down to adjusting weights of frequency components in the spectral domain. The goal of filter function is to adjust eigenvalues (i.e., the weights of eigenvectors) to fit the target output. Many of them are proving to be low-pass filters [81], which means that only low-frequency components are emphasized, i.e., the first few eigenvalues are enlarged, and the others are reduced. There exist a large number of works that can be understood as adjusting weights of frequency component during aggregation. Specifically, a linear function of $\mathbf{g}$ is employed:

$$\mathbf{Z} = (\sum_{i=0}^{l} \theta_i \lambda_i \, \mathbf{u}_i \, \mathbf{u}^\intercal{}_i) \, \mathbf{X} = \mathbf{U} \, \mathbf{g}_\theta(\mathbf{\Lambda}) \, \mathbf{U}^\intercal \, \mathbf{X}, \tag{2.41}$$

where $\mathbf{u}_i$ is the i-th eigenvector, $\mathbf{g}$ is *frequency filter function* controlled by parameters θ, and selected l lowest frequency components. The goal of $\mathbf{g}$ is to change the weights of eigenvalues to fit the target output. As shown in Fig. 2.6, A1 updates the weights of eigenvectors $(\mathbf{u}_1, \mathbf{u}_2)$ as $\mathbf{g}_\theta(\lambda)$ which is a linear function. Several state-of-the-art methods introduced in the last section are analyzed to illustrate this scheme:

Graph Convolutional Network [66] can be rewritten in spectral domain as:

$$\mathbf{Z} = \tilde{\mathbf{A}} \, \mathbf{X} = (\mathbf{I} - \tilde{\mathbf{L}}) \, \mathbf{X} = \mathbf{U}(1 - \mathbf{\Lambda}) \, \mathbf{U}^\intercal \, \mathbf{X} \tag{2.42}$$

Therefore, the frequency response function is $\mathbf{g}(\mathbf{\Lambda}) = 1 - \mathbf{\Lambda}$ which is a low-pass filter, i.e.,smaller eigenvalue will be adjusted to a large value, in which small eigenvalue corresponds to low frequency component.

GraphSAGE Considering the MEAN aggregation as example, we can rewrite GraphSAGE [48] in matrix form:

$$\mathbf{Z} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{X} = (\mathbf{I}+\tilde{\mathbf{A}})\,\mathbf{X} = (2\,\mathbf{I}-\tilde{\mathbf{L}})\,\mathbf{X} = \mathbf{U}(2-\mathbf{\Lambda})\,\mathbf{U}^{\intercal}\,\mathbf{X}, \tag{2.43}$$

so the frequency response function is $\mathbf{g}(\mathbf{\Lambda}) = 2 - \mathbf{\Lambda}$. Note that GraphSAGE's normalization is different from GCN, since GCN has renormalization trick.

Graph Isomorphism Network (GIN)

Multi-Layer neural network is capable of fit the scale (i.e., normalization) [66], so GIN [153] can be rewritten as:

$$\mathbf{Z} = \mathbf{D}^{-\frac{1}{2}}[(1+\epsilon)\,\mathbf{I}+\mathbf{A}]\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{X} = \mathbf{D}^{-\frac{1}{2}}[(2+\epsilon)\,\mathbf{I}-\tilde{\mathbf{L}}]\,\mathbf{D}^{-\frac{1}{2}}\,\mathbf{X} = \mathbf{U}(2+\epsilon-\mathbf{\Lambda})\,\mathbf{U}^{\intercal}\,\mathbf{X}\,. \tag{2.44}$$

GIN can be seen as a generalization of GCN or GraphSAGE without normalized adjacency matrix $\mathbf{A}$. The frequency response function is $\mathbf{g}(\mathbf{\Lambda}) = 2 + \epsilon - \mathbf{\Lambda}$

Summary

By showing concrete examples, we found that (B1) generalized the methods that apply linear low-pass filtering. The only difference among them is that the bias is different (i.e., 1 for GCN, 2 for GraphSAGE, and 2+ϵ for GIN). Therefore, we study the influence of bias on the filter function, and define a metric:

$$w\left(\lambda_i\right) = \frac{|bias - \lambda_i|}{\sum_j |bias - \lambda_j|}, \tag{2.45}$$

which indicates the adjusted and normalized eigenvalue, i.e., the proportion of the corresponding eigenvector. An large adjusted value means that the filtering will enlarge the influence of corresponding eigenvector. The range of the eigenvalue is [0, 2) for the normalized Laplacian matrix [22], so we have:

$$w(\lambda_i) = \frac{|bias - \lambda_i|}{\sum_j |bias - \lambda_j|} = \frac{bias - \lambda_i}{N \cdot bias - \sum_j \lambda_j} = \overbrace{\frac{-1}{N \cdot bias - \sum_j \lambda_j}}^{slope}\lambda_i + \overbrace{\frac{bias}{N \cdot bias - \sum_j \lambda_j}}^{intercept}, \tag{2.46}$$

when $bias > 2$. We study the slope and intercept of the above function regarding bias in filter function $\mathbf{g}$:

$$w_{slope}(bias) = \frac{-1}{N \cdot bias - \sum_j \lambda_j}, w_{intercept}(bias) = \frac{bias}{N \cdot bias - \sum_j \lambda_j}.$$

- $bias < \lambda$: The slope is positive, which means that the filter function is high-pass. Several examples are drawn in Fig. 2.7(a): as the bias increases, the slope becomes larger, which means that larger positive weights are assigned to high-frequency spectral components.

- $bias > \lambda$: The slope is negative, which means that the filter function is low-pass. Several examples are drawn in Fig. 2.7(b): as the bias increases, the slope becomes smaller, which means that larger positive weights are assigned to low-frequency spectral components.

Note that $bias > 2$ also ensure the non-negativity of all adjusted eigenvalue, since negative eigenvalue is hard to explain.

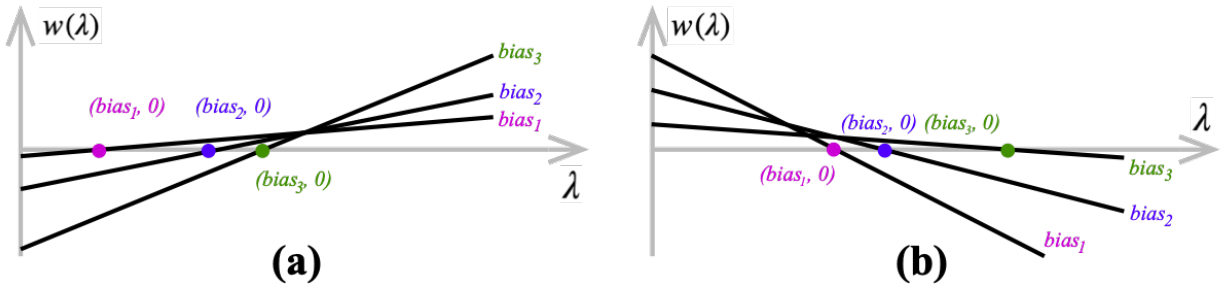


Figure 2.7: Filter function analysis ($bias_1 < bias_2 < bias_3$:(a) when $bias < \bar{\lambda}$; (b)when $bias > \bar{\lambda}$

2.5.2 Order of Approximation (B2)

Considering higher order of frequency, filter function can approximate any smooth filter function, because it is equivalent to applying the polynomial approximation. Therefore, introducing higher-order of frequencies boosts the representation power of filter function in simulating spectral signal. Formally, this type of work can be written as:

$$\mathbf{Z} = (\sum_{i=0}^{l} \sum_{j=0}^{k} \theta_j \lambda_i^j \, \mathbf{u}_i \, \mathbf{u}^{\intercal}{}_i) \, \mathbf{X} = \mathbf{U} \, \mathbf{P}_\theta(\mathbf{\Lambda}) \, \mathbf{U}^{\intercal} \, \mathbf{X}, \tag{2.47}$$

where $\mathbf{g}(\cdot) = \mathbf{P}_\theta(\cdot)$ is a polynomial function.

As shown in Fig. 2.8, A2 updates the weights of eigenvectors $(\mathbf{u}_1, \mathbf{u}_2)$ as $\mathbf{P}_\theta(\lambda)$ which is a polynomial function.

ChebNet [50] can be rewritten as:

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}}) \, \mathbf{X} = (\tilde{\theta_0} \, \mathbf{I} + \tilde{\theta_1} \tilde{\mathbf{L}} + \tilde{\theta_2} \tilde{\mathbf{L}}^2 + ...) \, \mathbf{X}, \tag{2.48}$$

where $T_k(\cdot)$ is the Chebyshev polynomial and θ_k is the Chebyshev coefficient. $\tilde{\theta}$ is the coefficient after expansion and reorganization. Therefore, we can rewrite it as:

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}}) \, \mathbf{X} = \mathbf{U}(\tilde{\theta_0} \cdot 1 + \tilde{\theta_1} \, \mathbf{\Lambda} + \tilde{\theta_2} \mathbf{\Lambda}^2 + ...) \, \mathbf{U}^{\intercal} \, \mathbf{X}, \tag{2.49}$$

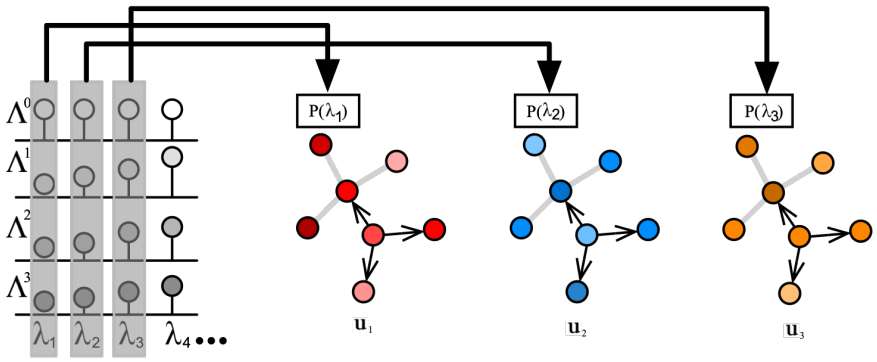


Figure 2.8: Illustration of B2.

where $\mathbf{g}(\mathbf{\Lambda}) = \tilde{\theta_0} \cdot 1 + \tilde{\theta_1}\,\mathbf{\Lambda} + \tilde{\theta_2}\mathbf{\Lambda}^2 + ... = \mathbf{P}(\mathbf{\Lambda})$.

DeepWalk [111] updates representation as:

$$
\begin{aligned}
\mathbf{Z} &= \frac{1}{t+1}(\mathbf{I} + \tilde{\mathbf{A}} + \tilde{\mathbf{A}}^2 + ... + \tilde{\mathbf{A}}^t)\,\mathbf{X} \\
&= \frac{1}{t+1}(\mathbf{I} + (\mathbf{I} - \tilde{\mathbf{L}}) + (\mathbf{I} - \tilde{\mathbf{L}})^2 + ... + (\mathbf{I} - \tilde{\mathbf{L}})^t)\,\mathbf{X} \\
&= \frac{1}{t+1}(2\,\mathbf{I} + (-1 - 2 - 3 - ...)\,\tilde{\mathbf{L}} + (1 + 3 + 6 + ...)\,\tilde{\mathbf{L}}^2 + ... \\
&((-1)^{t-1} + \begin{pmatrix} 1 \\ t \end{pmatrix}(-1)^{t-1})\,\tilde{\mathbf{L}}^{t-1} + (-1)^t\,\tilde{\mathbf{L}}^t)\,\mathbf{X} \\
&= (\theta_0\,\mathbf{I} + \theta_1\,\tilde{\mathbf{L}} + \theta_2\,\tilde{\mathbf{L}}^2 + ... + \theta_t\,\tilde{\mathbf{L}}^t)\,\mathbf{X} \\
&= \mathbf{U}(\theta_0 + \theta_1\,\mathbf{\Lambda} + \theta_2\,\mathbf{\Lambda}^2 + ... + \theta_t\,\mathbf{\Lambda}^t)\,\mathbf{U}^\intercal\,\mathbf{X},
\end{aligned}
$$

where $\mathbf{g}(\mathbf{\Lambda}) = \theta_0 + \theta_1\,\mathbf{\Lambda} + \theta_2\,\mathbf{\Lambda}^2 + ... + \theta_t\,\mathbf{\Lambda}^t$, and all parameters θ_i are determined by the predefined step size t.

Diffusion convolutional neural networks Diffusion convolutional neural networks (DCNN) [7] considers using a degree-normalized transition matrix, i.e., renormalized adjacency matrix: $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}\,\mathbf{A}$. For a graph,

$$\mathbf{Z} = W \odot \tilde{\mathbf{A}}^*\,\mathbf{X}, \tag{2.50}$$

where $\tilde{\tilde{\mathbf{A}}}^*_{tijl}$ denotes an tensor containing the power series of $\tilde{\mathbf{A}}$, and the $\odot$ operator represents element-wise multiplication. It can be transformed as:

$$\mathbf{Z} = (\theta_1\,\tilde{\mathbf{A}} + \theta_2\,\tilde{\mathbf{A}}^2 + \theta_3\,\tilde{\mathbf{A}}^3 + ...)\,\mathbf{X} = \mathbf{P}(\tilde{\mathbf{A}})\,\mathbf{X}, \tag{2.51}$$

which is equivalent to ChebNet, and parameters θ_i are learnable.

Node2Vec Node2Vec [45] can be rewritten in matrix form:

$$\mathbf{Z} = (\frac{1}{p} \cdot \overbrace{\mathbf{I}}^{\text{source}} + \overbrace{\tilde{\mathbf{A}}}^{\text{BFS}} + \frac{1}{q} \overbrace{(\tilde{\mathbf{A}}^2 - \tilde{\mathbf{A}})}^{\text{DFS}}) \mathbf{X}, \tag{2.52}$$

which can be transformed and reorganized after substitute $\tilde{\mathbf{A}} = \mathbf{I} - \tilde{\mathbf{L}}$:

$$\mathbf{Z} = [(1 + \frac{1}{p}) \mathbf{I} - (1 + \frac{1}{q}) \tilde{\mathbf{L}} + \frac{1}{q} \tilde{\mathbf{L}}^2] \mathbf{X} = \mathbf{U}[(1 + \frac{1}{p}) - (1 + \frac{1}{q}) \tilde{\Lambda} + \frac{1}{q} \tilde{\Lambda}^2] \mathbf{X}. \tag{2.53}$$

Therefore, Node2Vec's frequency response function integrate is a second order function of $\tilde{\Lambda}$ with predefined parameters, i.e., p and q.

Simple Graph Convolution Simple Graph Convolution (SGC) [146] removes non-linear function between neighboring graph convolution layers, and combine graph propagation in one single layer:

$$\mathbf{Z} = \tilde{\mathbf{A}}^K \mathbf{X} \tag{2.54}$$

where $\tilde{\mathbf{A}}$ is renormalized adjacency matrix, i.e., $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{A} \tilde{\mathbf{D}}^{-\frac{1}{2}}$. Therefore, it can be easily rewritten as:

$$\begin{aligned}
\mathbf{Z} &= (\mathbf{I} - \tilde{\mathbf{L}})^K \mathbf{X} \\
&= [\binom{K}{0} \mathbf{I} + \binom{K}{1} \tilde{\mathbf{L}}^1 + \binom{K}{2} \tilde{\mathbf{L}}^2 + \cdots + \tilde{\Lambda}^n] \mathbf{X} \\
&= \mathbf{U}[\binom{K}{0} + \binom{K}{1} \tilde{\Lambda}^1 + \binom{K}{2} \tilde{\Lambda}^2 + \cdots + \tilde{\Lambda}^n] \mathbf{U}^\intercal \mathbf{X}
\end{aligned}$$

Improved GCN To increase filter strength of GCN and produce smoother features, multiple layers have to be stacked, which will introduce numerous trainable parameters and may lead to severe overfitting especially when label rate is small. To fix this issue, Improved GCN (IGCN) [81] is proposed as:

$$\mathbf{Z} = \tilde{\mathbf{L}}^k \mathbf{X} = \mathbf{U} \tilde{\Lambda}^k \mathbf{U}^\intercal \mathbf{X} \tag{2.55}$$

Summary

In theory, polynomial approximation becomes more accurate as the order increases [4, 139, 107, 114, 24]. Note that linear approximation (B1) can be treated as a polynomial approximation with order of 1. We study polynomial approximation on $sign(x)$ function, showing the difference among all the examples listed in (B2). As shown in Fig. 2.9(a), linear function cannot well approximate $sign(x)$, since it is difficult for any straight line to fit a jump signal. When applying polynomial approximation, the situation become much better as shown in Fig. 2.9(b). If increasing the order of polynomial function , the variance will significantly

reduced (Fig. 2.9(c)). In sum, the higher order of polynomial approximation is more accurate, and yet incurs higher computational complexity. Therefore, Node2Vec/LINE/SDNE with an order of 2 have relatively low approximation power than the others (i.e., ChebNet, DeepWalk, Diffusion CNN, Simple Graph Convolution), since the order of the latter is predefined and could be as large as possible.

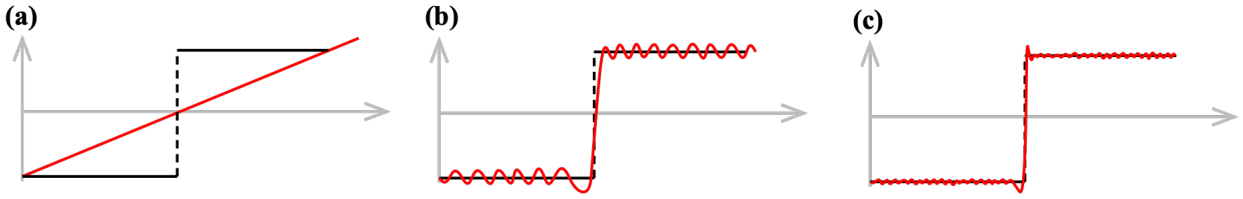


Figure 2.9: Approximation for $sign(x)$: (a) linear approximation (b) polynomial approximation with low orders, (c) polynomial approximation with high orders.

2.5.3 Approximation Type (B3)

Although polynomial approximation is widely used and empirically effective, it only works when applying on a smooth signal in the spectral domain. However, there is no guarantee that any real world signal is smooth. Therefore, the rational approximation is introduced to improve the accuracy of non-smooth signal modelling. Rational kernel based method can be written as:

$$\mathbf{Z} = (\sum_{i}^{l} \frac{\sum_{j=0}^{k} \theta_j \lambda_i^j}{\sum_{m=1}^{n} \phi_m \lambda_i^m + 1} \mathbf{u}_i \, \mathbf{u}^\intercal{}_i) \, \mathbf{X} = \mathbf{U} \frac{\mathbf{P}_\theta(\mathbf{\Lambda})}{\mathbf{Q}_\phi(\mathbf{\Lambda})} \mathbf{U}^\intercal \mathbf{X}, \tag{2.56}$$

where $\mathbf{g}(\cdot) = \frac{\mathbf{P}_\theta(\cdot)}{\mathbf{Q}_\phi(\cdot)}$ is a rational function, and $\mathbf{P}, \mathbf{Q}$ are independent polynomial functions. Spectral methods process graph as a signal in the frequency domain.

As shown in Fig. 2.10, A1 updates the weights of eigenvectors $(\mathbf{u}_1, \mathbf{u}_2)$ as $\mathbf{g}_\theta(\lambda)$ which is a rational function.

Auto-Regressive filter Label propagation (LP) [166, 163, 11] is a prevail methodology for graph-based learning. The objective of LP contains twofold: one is to extract embeddings that agrees with the label, the other is to be similar with neighboring vertices. Label can be treated as part of node attributes, there, we can have:

$$\mathbf{Z} = (\mathbf{I} + \alpha \, \tilde{\mathbf{L}})^{-1} \mathbf{X} = \mathbf{U} \frac{1}{1 + \alpha(1 - \mathbf{\Lambda})} \mathbf{U}^\intercal \mathbf{X}, \tag{2.57}$$

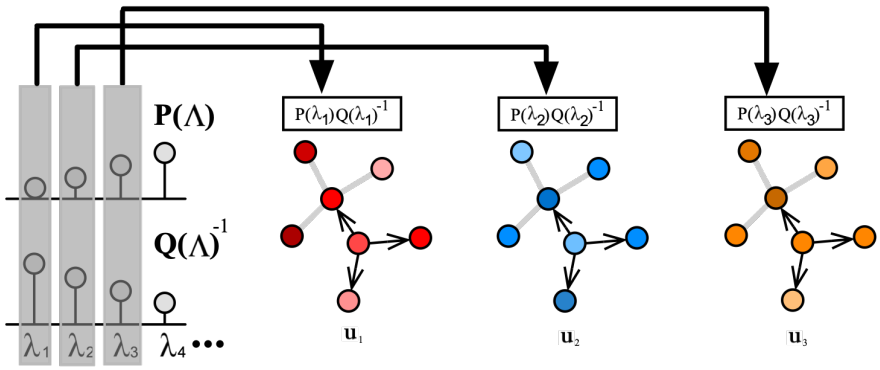


Figure 2.10: Illustration of B3.

PPNP Personalized PageRank (PPNP) [67] can obtain node's representation via teleport (restart) probability which indicate the ratio of keeping the original representation:

$$\mathbf{Z} = \frac{\alpha}{\mathbf{I} - (1-\alpha)(\mathbf{I} - \tilde{\mathbf{L}})}\mathbf{X} = \mathbf{U}\frac{\alpha}{\alpha\,\mathbf{I} + (1-\alpha)\,\mathbf{\Lambda}}\mathbf{U}^\intercal\,\mathbf{X}, \tag{2.58}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\,\mathbf{A}$ is random-walk normalized adjacency matrix with self-loop. Eq. 2.58 is with a rational function whose numerator is a constant.

ARMA filter [12] can be rewritten in the spectral domain with substituting $\tilde{\mathbf{A}} = \mathbf{I} - \tilde{\mathbf{L}}$ as:

$$\mathbf{Z} = \frac{b}{\mathbf{I} - a(\mathbf{I} - \tilde{\mathbf{L}})}\mathbf{X} = \frac{b}{(1-a)\,\mathbf{I} + a\,\mathbf{\Lambda}}\mathbf{X}\,. \tag{2.59}$$

Note that ARMA filter is an unnormalized version of PPNP. When a+b=1, ARMA filter becomes PPNP.

RationalNet [21] proposes a general rational function and optimized by Remez algorithm.

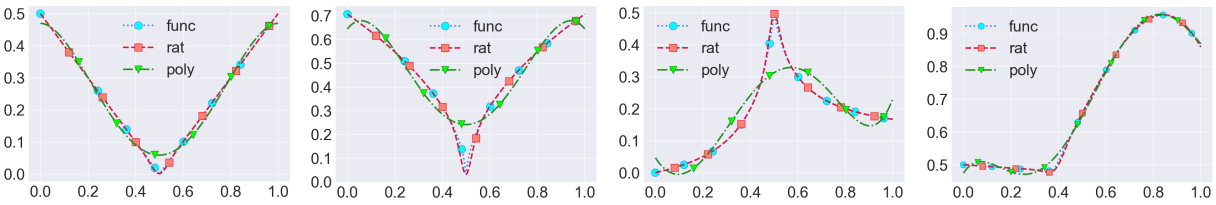


Figure 2.11: Rational (rat) and polynomial (poly) approximation for several functions with discontinuity (func). From left to right: $\sqrt{|x-0.5|}$; $|x-0.5|$; $\frac{x}{10|x-0.5|+1}$; $max(0.5, sin(x + x^2)) - \frac{x}{20}$

2.5.4 Connection among spectral methods

There is a strong-tie among the above-mentioned three groups of spectral methods in the perspective of *generalization* and *specialization*, as shown in Fig. (2.1): **(1) Generalization:** *Frequency Aggregation* can be extended to *Order of Connectivity* by adding more higher order of eigenvalues, i.e., 1←k. *Order of Connectivity* can be upgraded to *Approximation Type* if the denominator of filter function is not 1; **(2) Specialization:** *Frequency Aggregation* is a special case of *Order of Connectivity* by setting the highest order to 1. *Order of Connectivity* is a special case of *Approximation Type* by setting the denominator of filter function to 1.

2.6 Complexity Analysis

As shown in Fig. (2.2), there is a trade-off between computational efficiency and generalization power. Polynomial approximation requires cost of $\mathcal{O}(K|\mathcal{E}|)$ [32] where K is the order number and $|\mathcal{E}|$ is the number of edges. Accordingly, linear approximation needs $\mathcal{O}(|\mathcal{E}|)$. It is difficult for the linear function to fit any non-linear signal, such as any curve function. If there is any non-linearity, the polynomial approximation is qualified to estimate the signal. For rational approximation, the quick algorithm need $\mathcal{O}(n^2)$ [68].

Next, we analyze the required degree given fixed error. Linear function is exclusive since its order is fixed as one. According to existing analysis [21], rational approximation will need degree:

$$n \leq (\ln \frac{C}{\epsilon})^2 = \mathcal{O}(poly \log(1/\epsilon)) \tag{2.60}$$

given error ϵ, where C is a constant. According to Theorem 3.5.3, polynomial approximation needs degree

$$n \leq \frac{C\beta}{\epsilon} = \mathcal{O}(1/\epsilon), \tag{2.61}$$

where $\beta \in (0.278, 0.286)$ [140].

2.7 Conclusion

In this survey, a unified framework is proposed to summarize the state-of-the-art GNNs, providing a new perspective for understanding GNNs of different mechanisms. By analytically categorizing current GNNs into the spatial and spectral domains and further dividing them into subcategories, our analysis reveals that the subcategories are not only strongly connected by generalization and specialization relations within their domain, but also by equivalence relation across the domains. We demonstrate the generalization power of our proposed framework by reformulating numerous existing GNN models. The above survey of the state-of-the-art graph neural networks, showing that GNNs is still a young research area.

Increasing number of emerging GNN models [39, 155, 20, 145, 149, 108] makes the theoretical understanding [90, 106] a urgent need. Therefore, the next-generation GNNs are expected to be more interpretable and transparent to the application [47, 8, 160].

Chapter 3

Rational Neural networks

This chapter proposes a rational graph filter that is more accurate than the popular polynomial filter. We study the theoretical property of rational filter and then demonstrate its effectiveness in estimation singular signal that contains discontinuities.

3.1 Introduction

Effective information analysis generally boils down to the geometry of the data represented by a graph. Typical applications include social networks[71], transportation networks[10], spread of epidemic disease[104], brain's neuronal networks[97], gene data on biological regulatory networks[30], telecommunication networks[34], knowledge graph[85], which are lying on non-Euclidean graph domain. To describe the geometric structures, graph matrices such as adjacency matrix or graph Laplacian can be employed to reveal latent patterns.

In recent years, many problems are being revisited with deep learning tools. Convolutional neural networks(ConvNets) emerging in recent years are at the heart of deep learning, and the most prominent strain of neural networks in research. ConvNets have revolutionized computer vision[70], natural language processing[25], computer audition[43], reinforcement learning[101, 127], and many other areas. However, ConvNets are designed for grid data such as image, which belongs to the Euclidean domain. Graph data is non-Euclidean which makes it difficult to employ typical ConvNets. To bridge the gap, Bruna et al. [18][53] generalized spectral convolutional operation which requires expensive steps of spectral decomposition and matrix multiplication. Hammond et al.[50] first introduced truncated Chebyshev polynomial for estimating wavelet in graph signal processing. Based on this polynomial approximation, Defferrard et al.[32] designed ChebNet which contains a novel neural network layer for the convolution operator in the spectral domain. Kipf and Welling[65] simplified ChebNet by assuming the maximum of eigenvalues is 2 and fixing the order to 1, which boosts both effectiveness and efficiency. Li et al.[79] found that this simplified ChebNet is an application

of Laplacian smoothing, which implies that current studies are only effective on the smooth signal.

Current studies on graph ConvNet heavily rely on polynomial approximation, which makes it difficult to estimate jump signals. Fig. 4.1 shows the behaviors of polynomial and rational function on jump discontinuity: rational approximation fits the functions considerably better than polynomials. It is widely recognized that **(1)** polynomial approximation suffers from Gibbs phenomenon, which means polynomial function oscillate and overshoot near discontinuities[139]; **(2)** Applying a higher order of polynomials could dramatically reduce the oscillation, but also incurs an expensive computational cost. **(3)** Polynomials require degree $\Omega(\text{poly}(1/\epsilon))$ to approximate functions near singularities and on an unbounded domain, while rational functions only need $\mathcal{O}(\text{poly}\log(1/\epsilon))$ to achieve ϵ-close[84, 136]. However, it is non-trivial to apply rational approximation. Polynomial-based method can easily transfer the function on eigenvalues to the same function on graph Laplacian so that matrix multiplication by eigenvector can be avoided. It is not easy for rational approximation to do so due to the additional denominator.

In this paper, the advantage of the rational function in approximation is transferred to spectral graph domain. Specifically, we propose a rational function based neural networks(RationalNet), which can avoid matrix multiplication by eigenvectors. To alleviate the local minimum problem of the neural network, a relaxed Remez algorithm is employed for parameter initialization. Our theoretical analysis shows that rational functions converge much faster than polynomials on jump signal. In a nutshell, the key innovations are:

- **Propose a neural network model based on rational function for recovering jump discontinuities**: To estimate the jump signal, our proposed method integrates rational approximation and spectral graph operation to avoid matrix multiplication by eigenvectors. For graph signal regression task, expensive matrix inversion can be circumvented by graph Fourier transform.
- **Develop an efficient algorithm for model parameters optimization**: Remez algorithm is theoretically optimal, but it is often not practical especially when approximating discrete signal. To alleviate this issue, the stopping rules of Remez algorithm are relaxed to initialize the neural networks parameters.
- **Provide theoretical analysis for the proposed method on jump signal**: For understanding the behaviors of polynomial and rational function on jump discontinuities, a uniform representation is proposed to analyze convergence rate regarding the order number theoretically.
- **Conducting extensive experiments for performance evaluations**[1]: The proposed method was evaluated on synthetic and real-world data. Empirical results demonstrate the effectiveness and efficiency of the proposed approach.

[1]https://github.com/aquastar/RationalGraphNet

The rest of the paper is organized as follows. Section 6.2 reviews existing work in this area. Necessary preliminary is presented in section 3.3. Section 6.4 elaborates a rational function based model for graph signal regression task. Section 3.5 presents algorithm description and theoretical analysis. In Section 6.5, experiments on synthetic and real-world data are analyzed. This paper highlights the important findings in Section 6.6.

3.2 Related Work

The proposed method draws inspiration from the field of approximation theory, spectral graph theory and recent works using neural networks. In what follows, we provide a brief overview of related work in these fields.

3.2.1 Approximation theory

In mathematics, approximation theory is about approximating complex function with simpler functions, quantitatively characterizing the errors. This is a practical tool that can significantly reduce the computational overhead of operating on complex function. Typically, polynomial or rational approximations are applied. Polynomials are popular and easy to use, whereas rational functions seem complex but more powerful than polynomials at approximating functions in challenging case such as jump discontinuities. Basic properties of rational function are described in books of complex analysis[4, 139, 107, 114, 24, 112, 2, 168, 16, 98, 119].

3.2.2 Spectral graph theory

Spectral graph theory is the study of the properties of a graph in relationship to the eigenvalues, and eigenvectors of matrices of the graph[22, 44, 29]. Many graphs and convolution methods have been proposed recently. The spectral convolution methods ([18, 32, 65, 17]) are the mainstream algorithm developed as the graph convolution. Because their theory is based on the spectral graph analysis ([125, 126]). The polynomial approximation is firstly proposed by [50]. Inspired by this, graph convolutional neural networks (GCNNs) ([32]) is a successful attempt at generalizing the powerful convolutional neural networks (CNNs) in dealing with Euclidean data to modeling graph-structured data. Kipf and Welling proposed a simplified type of GCNNs[65], called graph convolutional networks (GCNs). The GCN model naturally integrates the connectivity patterns and feature attributes of graph-structured data and outperforms many state-of-the-art methods significantly. Li et al.[79] found that GCN is actual an application of Laplacian smoothing, which is inconsistent with GCN's motivation. In sum, this thread of work calculates the average of vertexes within Nth-order neighbors.

In this paper, we focus on the effectiveness of the approximation technique on graph signal.

Under graph signal regression task, the superiority of rational function beyond polynomial function is analyzed, and a rational function based neural network is proposed.

3.3 Preliminaries

We focus processing graph signals defined on undirected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{V}$ is a set of n vertexes, $\mathcal{E}$ represents edges and $\mathcal{W} = [w_{ij}] \in \{0, 1\}^{n \times n}$ is an unweighted adjacency matrix. A signal $x : \mathcal{V} \to \mathbb{R}$ defined on the nodes may be regarded as a vector $x \in \mathbb{R}^n$. Combinatorial graph Laplacian[22] is defined as $\mathbf{L} = D - \mathcal{W} \in \mathbb{R}^{n \times n}$ where D is degree matrix.

The Laplacian is diagonalized by the Fourier basis $\mathbf{U}^\intercal$: $\mathbf{L} = \mathbf{U}\,\Lambda\,\mathbf{U}^\intercal$ where Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e., $\Lambda_{ii} = \lambda_i$. The graph Fourier transform of a signal $x \in \mathbb{R}^n$ is defined as $\hat{x} = \mathbf{U}^\intercal\, x \in \mathbb{R}^n$ and the inverse graph Fourier transform inverse is $x = \mathbf{U}\,\hat{x}$[125, 126, 167]. To enable the formulation of fundamental operations such as filtering in the vertex domain, the convolution operator on graph is defined in the Fourier domain such that $f_1 * f_2 = \mathbf{U}\,[(\mathbf{U}^\intercal\, f_1) \odot (\mathbf{U}^\intercal\, f_2)]$, where $\odot$ is the element-wise product, and f_1/f_2 are two signals defined on vertex domain. It follows that a vertex signal $f_2 = x$ is filtered by spectral signal $\hat{f}_1 = \mathbf{U}^\intercal\, f_1 = \mathbf{g}$ as:

$\mathbf{g} * x = \mathbf{U}\,[\mathbf{g}(\mathbf{\Lambda}) \odot (\mathbf{U}^\intercal\, f_2)] = \mathbf{U}\,\mathbf{g}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\, x$.Note that a real symmetric matrix $\mathbf{L}$ can be decomposed as $\mathbf{L} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^{-1} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal$ since $\mathbf{U}^{-1} = \mathbf{U}^\intercal$. D. K. Hammond et al. and Defferrard et al.[50, 32] apply polynomial approximation on spectral filter $\mathbf{g}$ so that:

$$
\begin{aligned}
&\mathbf{g} * x = \mathbf{U}\,\mathbf{g}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\, x \\
\approx\ &\mathbf{U} \sum_k \theta_k T_k(\tilde{\mathbf{\Lambda}})\,\mathbf{U}^\intercal\, x && (\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\,\mathbf{\Lambda} - \mathbf{I_N}) \\
=\ &\sum_k \theta_k T_k(\tilde{\mathbf{L}}) x && (\mathbf{U}\,\mathbf{\Lambda}^k\,\mathbf{U}^\intercal = (\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)^k)
\end{aligned}
$$

Kipf and Welling[65] simplifies it by:

$$
\begin{aligned}
&\mathbf{g} * x \\
\approx &\theta_0 \mathbf{I_N}\, x + \theta_1 \tilde{\mathbf{L}} x && \text{(expand to 1st order)} \\
= &\theta_0 \mathbf{I_N}\, x + \theta_1 (\frac{2}{\lambda_{max}} \mathbf{L} - \mathbf{I_N}))x && (\tilde{\mathbf{L}}{=}\tfrac{2}{\lambda_{max}} \mathbf{L} - \mathbf{I_N}) \\
= &\theta_0 \mathbf{I_N}\, x + \theta_1 (\mathbf{L} - \mathbf{I_N}))x && (\lambda_{max}{=}2) \\
= &\theta_0 \mathbf{I_N}\, x - \theta_1\, \mathbf{D}^{-\frac{1}{2}}\, \mathbf{A}\, \mathbf{D}^{-\frac{1}{2}}\, x && (\mathbf{L}{=}\mathbf{I_N} - \mathbf{D}^{-\frac{1}{2}}\, \mathbf{A}\, \mathbf{D}^{-\frac{1}{2}}) \\
= &\theta_0 (\mathbf{I_N} + \mathbf{D}^{-\frac{1}{2}}\, \mathbf{A}\, \mathbf{D}^{-\frac{1}{2}})x && (\theta_0{=}-\theta_1) \\
= &\theta_0 (\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}})x && (\text{renormalization:}\tilde{\mathbf{A}}{=}\mathbf{A} + \mathbf{I_N}, \\
& && \tilde{\mathbf{D}}_{ii}{=}\textstyle\sum_j \mathbf{A}_{ij}),
\end{aligned}
$$

rewrite the above GCN in matrix form: $g_\theta * X \approx (\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}) X \Theta$, which leads to *symmetric normalized Laplacian* with raw feature. GCN has been analyzed GCN in [79] using smoothing Laplacian[135]: $y = (1-\gamma)x_i + \gamma \sum_j \frac{\tilde{a}_{ij}}{d_i} x_j = x_i - \gamma(x_i - \sum_j \frac{\tilde{a}_{ij}}{d_i} x_j)$,where γ is a weight parameter between the current vertex x_i and the features of its neighbors x_j, d_i is degree of x_i, and y is the smoothed Laplacian. This smoothing Laplacian has a matrix formïijZ

$$
\begin{aligned}
Y &= x - \gamma \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{L}} x \\
&= (\mathbf{I_N} - \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{L}}) x && (\gamma = 1) \\
&= (\mathbf{I_N} - \tilde{\mathbf{D}}^{-1} (\tilde{\mathbf{D}} - \tilde{\mathbf{A}}))x && (\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}) \\
&= \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} x.
\end{aligned}
$$

The above formula is *random walk normalized Laplacian* as a counterpart of *symmetric normalized Laplacian*. Therefore, GCN is nothing but a first-order Laplacian smoothing which averages neighbors of each vertex.

3.4 Model description

This section formally defines the task of graph signal recovering and then describes our proposed RationalNet which aims to characterize the jump signal in the spectral domain.

3.4.1 Problem Setting

All the related works integrate graph convolution estimator and fully-connected neural layers for a classification task. This classification can be summarized as:

$$Y = f(\mathcal{G}, x)\Theta, \tag{3.1}$$

where Θ indicates the parameters of normal neural network layers connecting the output of f and the label Y, such as fully-connected layers and softmax layers for classification. Moreover, f is a neural network layer implemented by approximation techniques. However, whether the success is due to the neural networks(Θ) or the convolution approximation method(f) remains unknown. To focus on the analysis of approximation on f, a graph signal regression task is proposed to evaluate the performance of the convolution approximators f. Regression task directly compares the label and the output of f, removing the distraction of Θ.

Given a graph $\mathcal{G}$, raw feature x, and training signal on the part of vertexes, Y_{train}, our goal is to recover signal values, Y_{test}, on test nodes. Formally, we want to find a $f(\cdot)$ so that:

$$Y = f(\mathcal{G}, x).$$

If the raw features are good enough for the regression task, whether the effectiveness is due to f or x is difficult to verify. Therefore, one reasonable option for x is the uniform signal in spectral domain. Specifically, $x = \sum_i \mathbf{U}_i$ and $\hat{x} = \mathbf{U}^\intercal x = \mathbb{1} = \{1, 1, ..., 1\}$, which means that x represents eigenbasis of graph structure in spectral domain. Each entry of vector $\hat{x}_i$ indicates one eigenvector in the spectral domain. The physical meaning of the convolution operation is how to select eigenbasis in spectral domain to match the graph signal Y. Representing $\mathcal{G}$ with graph Laplacian, the regress task is formulated as:

$$Y = f(\mathbf{L}, \mathbf{U}) = \mathbf{U}\, \mathrm{g}_\theta(\mathbf{\Lambda})\, \mathbf{U}^\intercal \sum_i \mathbf{U}, \tag{3.2}$$

where g_θ is the spectral filter to approximate.

3.4.2 RationalNet

RationalNet is a neural network that has a rational function as the kernel for graph convolution operator. As shown in Fig. 3.1, graph connection information, i.e., graph Laplacian $\mathbf{L}$, is converted into spectral domain by inverse Fourier transform $\mathbf{U}^\intercal$. The same transform is operated on features X. Then the element-wise product of them in spectral domain is convolution or spectral filtering. After that, the convolution signal is approximated by a rational function whose parameters are learned during training. The new features of nodes, Z, are obtained by Fourier transform $\mathbf{U}$. Finally, the residues between the new features and ground truth are calculated for the backpropagation algorithm. Similar to polynomial

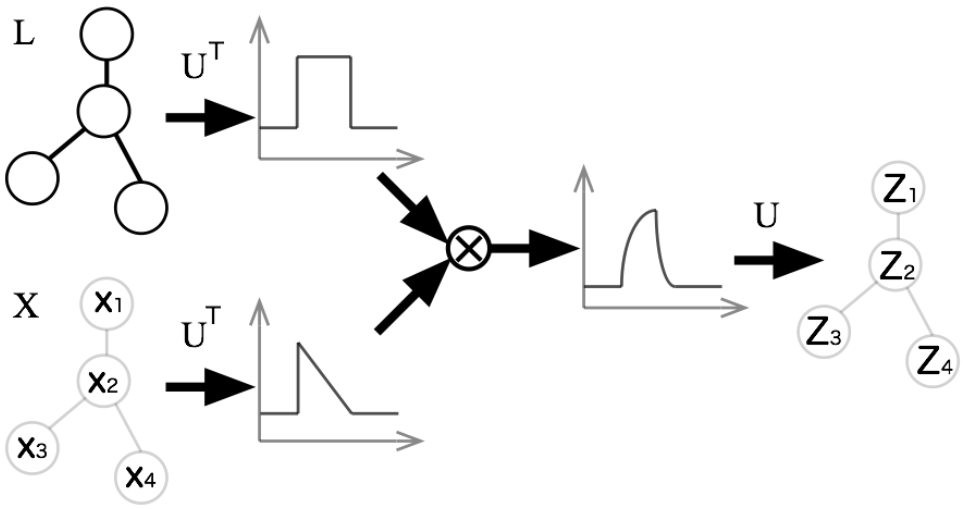


Figure 3.1: Structure of RationalNet.

approximation on graph domain such as ChebNet[32] or GCN[65], RationalNet approximates the spectral filter by a widely used type of rational function, i.e., Padé approximator, which is defined as:

$$R(x) = \frac{\sum_{i=0}^{m} \psi_i x^i}{\sum_{j=0}^{n} \phi_j x^j}, \phi_0 = 1, \phi_j, \psi_i \in \mathbb{R}. \tag{3.3}$$

Applying to graph convolution operator, we have:

$$\begin{aligned}
\mathrm{g}_\theta * x &= \mathbf{U}\, \mathrm{g}_\theta\, \mathbf{U}^\intercal\, x && \text{(convolution theorem)} \\
&\approx \mathbf{U}\, \frac{\sum_{i=0}^{m} \psi_i \tilde{\mathbf{\Lambda}}^i}{1 + \sum_{j=1}^{n} \phi_j \tilde{\mathbf{\Lambda}}^j}\, \mathbf{U}^\intercal\, x && (\tilde{\mathbf{\Lambda}} = \frac{\mathbf{\Lambda}}{\lambda_{max}}) \\
&= \mathbf{U}\, \frac{\mathbf{P}(\mathbf{\Lambda})}{\mathbf{Q}(\mathbf{\Lambda})}\, \mathbf{U}^\intercal\, x, && \text{(define P and Q)}
\end{aligned}$$

where $\mathbf{\Lambda}$ represents a diagonal matrix whose entries are eigenvalues, $\mathrm{g}_\theta = R$, and $\theta = \{\psi, \phi\}$. The division $\frac{P}{Q}$ is element-wise. The inverse of matrix Q(x) is equivalent to applying reciprocal operation on its diagonal entries, so the equation can be rewritten as:

$$\mathbf{U}\, \mathbf{P}(\mathbf{\Lambda})\, \mathbf{Q}(\mathbf{\Lambda})^{-1}\, \mathbf{U}^\intercal\, x.$$

Applying matrix rules, it's easy to have:

$$
\begin{aligned}
&= \mathbf{U}\,\mathbf{P}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\,\mathbf{U}\,\mathbf{Q}(\mathbf{\Lambda})^{-1}\,\mathbf{U}^\intercal\,x && (\mathbf{U}^\intercal{=}\mathbf{U}^{-1}, \mathbf{U}^\intercal\,\mathbf{U}{=}\mathbf{I_N}) \\
&= \big[\,\mathbf{U}\,\mathbf{P}(\mathbf{\Lambda})\,\mathbf{U}^\intercal\,\big]\big[\,\mathbf{U}\,\mathbf{Q}(\mathbf{\Lambda})^{-1}\,\mathbf{U}^\intercal\,\big]x && \\
&= \big[\,\mathbf{P}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)\big]\big[\,\mathbf{U}\,\mathbf{Q}(\mathbf{\Lambda})^{-1}\,\mathbf{U}^\intercal\,\big]x && (\mathbf{U}\,\mathbf{\Lambda}^k\,\mathbf{U}^\intercal{=}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)^k) \\
&= \big[\,\mathbf{P}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)\big]\big[(\mathbf{Q}(\mathbf{\Lambda})\,\mathbf{U}^{-1})^{-1}\,\mathbf{U}^\intercal\,\big]x && (A^{-1}B^{-1}{=}(BA)^{-1}) \\
&= \big[\,\mathbf{P}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)\big]\big[(\mathbf{U}\,\mathbf{Q}(\mathbf{\Lambda})\,\mathbf{U}^{-1})^{-1}\big]x && (\mathbf{U}^\intercal{=}\mathbf{U}^{-1}) \\
&= \big[\,\mathbf{P}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)\big]\big[(\mathbf{U}\,\mathbf{Q}(\mathbf{\Lambda})\,\mathbf{U}^\intercal)^{-1}\big]x && (\mathbf{U}^\intercal{=}\mathbf{U}^{-1}) \\
&= \big[\,\mathbf{P}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)\big]\big[(\mathbf{Q}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal))^{-1}\big]x && (\mathbf{U}\,\mathbf{\Lambda}^k\,\mathbf{U}^\intercal{=}(\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)^k)
\end{aligned}
$$

Since $\mathbf{U}\,\mathbf{\Lambda}^k\,\mathbf{U}^\intercal = (\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^\intercal)^k$, we can rewrite the equation above as:

$$\mathrm{g}_\theta * x = \mathbf{P}(\mathbf{L})\,\mathbf{Q}(\mathbf{L})^{-1}x, \tag{3.4}$$

where $\mathbf{P}(x) = \sum_{i=0}^{m}\psi_i x^i$ and $\mathbf{Q}(x) = \sum_{j=0}^{n}\phi_j x^j$. Note that $\mathbf{P}(\mathbf{L})\,\mathbf{Q}(\mathbf{L})^{-1}x = \mathbf{Q}(\mathbf{L})^{-1}\,\mathbf{P}(\mathbf{L})x$ in our case. Based on polynomial approximation, RationalNet only adds a inverse polynomial function $\mathbf{Q}(\mathbf{L})^{-1}$. Therefore, polynomial approximation (GCN/ChebNet) on graph is a special case of RationalNet when $\mathbf{Q}(\mathbf{L})^{-1} = \mathbf{I}$.

Computing inverse of a matrix is still of high complexity $\mathcal{O}(n^3)$ (Gauss-Jordan elimination method) in Eq. 3.4, especially for large matrix. This can be avoided by transferring vertex graph signal and raw features into the spectral domain. Therefore, the Eq. 3.12 can be rewritten as:

$$\hat{Y} = \frac{\mathbf{P}(\mathbf{\Lambda})}{\mathbf{Q}(\mathbf{\Lambda})}\hat{x}, \tag{3.5}$$

where $\hat{Y} = \mathbf{U}^\intercal Y$ is the graph signal projected into spectral domain by graph Fourier transform , and $\hat{x} = \mathbf{U}^\intercal x$ is the graph Fourier transform of raw feature. By this step, we only need compute reciprocal of eigenvalues, rather than computing matrix multiplication and inversion at each update. Eq. 2 can be obtained via left multiplying both sides of Eq. 3.2 by transpose of eigenvectors. Note that Eq. 3.5 is applicable when there is no other layers between the output Y and the convolution operation. In contrast, Eq. 3.4 can be used not only in regression task like Eq. 3.2, but also in classification where there exist additional neural networks as described in Eq. 3.1. RationalNet has complexity $\mathcal{O}(|\mathcal{E}|)$ for Eq. 3.5 and $\mathcal{O}(|\mathcal{E}|^3)$ for Eq. 3.4.

3.4.3 Relaxed Remez Algorithm for initialization

RationalNet is powerful at approximating a function. However, it is often stuck in a local optimum due to the neural network optimization, which makes rational function not always better than the polynomial approximation. Remez exchange algorithm[119] is an iterative

algorithm finding simple approximations to functions. Nevertheless, the drawback of this minimax algorithm is that the order for optimum is unknown and the stopping condition is not often practical. To improve RationalNet's initialization, a relaxed Remez algorithm is proposed.

As Theorem **24.1** (equioscillation characterization of best approximants, [139]) states: Given the order of numerator(m) and denominator(n), and a real function f that is continuous in [p, q], there exists a unique best rational approximation R^* defined as Eq. 4.3. This means that the R^* optimizes the minimax error:

$$R^* = \arg\min_{R} \max_{x\in[p,q]} |f - R|. \tag{3.6}$$

A rational function R is equal to R^* iff $f - R$ equioscillates between at least m+n+2 extreme points, or say the error function reaches the absolute maximum with alternating sign:

$$f(x_d) - R(x_d) = (-1)^d E, d \in [0, m + n + 1], \tag{3.7}$$

where d indicates the index of data point, and E represents the max of residuals: $E = max_{x_d}|f(x_d) - R(x_d)|$. For rational function approximation, there is a product of E with ϕ_j in the equations, which results in nonlinearity. Therefore, these equations can only be solved iteratively. By linearzing the euations, The iteration formula can be defined as:

$$\sum_{i=0}^{m} \psi_i x_d^i - \left[f\left(x_d\right) - (-1)^d E_r\right] \sum_{j=1}^{k} \phi_j x_d^j = f\left(x_d\right) - (-1)^d E_{r+1}, \tag{3.8}$$

where r indicate the iteration index. Eq. 3.8 is obtained by removing nonlinear terms of $(E_r - E_{r+1})\phi_j x_d^j$ in Eq. 3.7. This procedure can converge in a reasonable time ([93]). Expanding Eq. 3.8 for all sampled data points $x_0, x_1, ..., x_d$, it can be rewritten as:

$$\begin{bmatrix} x_0^0 & ... & x_0^m & (E_r - y_0)x_0^1 & ... & (E_r - y_0)x_0^n & (-1)^0 \\ x_1^0 & ... & x_1^m & (E_r - y_0)x_1^1 & ... & (E_r - y_0)x_1^n & (-1)^1 \\ x_2^0 & ... & x_2^m & (E_r - y_0)x_2^1 & ... & (E_r - y_0)x_2^n & (-1)^2 \\ ... & & & & & & \\ x_K^0 & ... & x_K^m & (E_r - y_0)x_K^1 & ... & (E_r - y_0)x_K^n & (-1)^K \end{bmatrix} \begin{bmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ .. \\ \psi_m \\ \phi_1 \\ \phi_2 \\ \phi_3 \\ .. \\ \phi_n \\ E_{r+1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ ... \\ y_K \end{bmatrix}, \tag{3.9}$$

where K=m+n+1. Starting from an assumed initial guess $E_{r=0}$, this set of linear equations are used to solve unknown ψ_i, ϕ_j and E_{r+1}, when two successive values of E_r are under predefined threshold such as $|E_{r+1} - E_r|$ is less than 1e-6. Constructing a rational function with new coefficients, Remez computes the error residuals. If absolute value of the residuals δ are not great than $|E|$, the optimal coefficients are obtained. Otherwise, Remez calculates the roots of rational function and constructs a new set of control points by collecting the extremes in each interval of roots, and repeat the computation of Eq. 3.8 until residuals δ are not great than $|E|$. However, this stopping rule is not often satisfied, which makes the algorithm stuck in dead loop. Therefore, we add an iteration limit for avoid dead loop. The relaxed Remez algorithm could be summarized as follows:

1. **Prepare training data**
    - Specify the degree of interpolating rational function.
    - Pick $m+n+2$ points from the data points $X = \{x_0, x_1, ..., x_{m+n+1}\}$ with equal interval. Under this discrete setting, the distances between any neighbors are considered equal if the data distribution are dense
2. **Optimization by equioscillation constraint**
    - Solve coefficients and E by Eq. 3.9
    - Form a new rational function R with new coefficients
    - Calculate residual errors
    - Repeat until E converges or $|E_{r+1} - E_r|$ is less than 1e-6
3. **Check stopping rule**
    - Calculate residual errors
    - Stops if the absolute value of any residual is not numerically greater than $|E|$.
    - Otherwise, find the n+m+1 roots of the rational function and find the points at which the error function attains its extreme value. Rerun the algorithm with this new set of training data from the second step.

We have considered an algorithm for obtaining minimax approximation when the function could be evaluated at any point inside the interval. In our case, the function is known only at a set of discrete points, since eigenvalues are not continuous. However, this problem is no essentially different form the continuous case if the set of points is rather dense in the target interval [p, q]. We simply assume that eigenvalue samples are dense enough, since we often normalized eigenvalues into the range [0,1], several hundreds of points are thereby sufficient. For example, our smallest size of the synthetic graph consists of 500 nodes, so there are 500 eigenvalues distributed in [0, 1], which should be enough for approximation.

If the degree of rational function is large, then the system of equations could be ill-conditioned. Sometime, the linear system of equations 3.9 is singular, which make the solution vector(ψ_i, ϕ_j, E_{r+1}) under-determined. We traverse all possible m/n pairs given the maximum of m and n. The relaxed algorithm discards any m/n if singular matrix error occurs.

We found that the residuals δ are not smaller than $|E|$ for some m/n pairs, and the algorithm continues to output the same values. In such case, the algorithm stops if the maximum and minimum residuals ($\delta_{min,max}$) converge or they satisfy $\delta_{0,1,\ldots,m+n+1} < |E|$.

3.5 Algorithm and Theoretical analysis

This section elaborates algorithm details and analyzes its convergence rate on jump discontinuity.

3.5.1 Algorithm description and complexity analysis

The Algorithm 1 first calculate graph Laplacian(line 1) and spectral decomposition(line 3), and convert vertex signal into spectral domain by inverse Fourier transform(line 2). From given m, n, algorithm 1 traverse all possible m/n pairs (line 4). Picking up $m+n+2$ points with equal intervals, the optimal error is calculated (line 10). After convergence, optimal m/n and ψ_i/ϕ_i are determined (line 11). Then algorithm calculates the residuals(line 13). If the stopping rule is not satisïňĄed, decrease the order of denominator or numerator in turns and repeat the same process, otherwise, output the parameters of rational function. With optimal parameters, graph convolution operation is calculated by rational approximation (line 19). Then we conduct typical neural networks optimization.

Solving a system of linear equations(line 6-15) has a complexity of at most $\mathcal{O}((n+m+2)^3)$ and at least $\mathcal{O}((n+m+2)^2)$. However, $m+n+2$ is often small in practice such as 12 in our experiments, since large $m+n+2$ will exponentially increase the complexity and violate its original intention. Therefore, the actual complexity is $\mathcal{O}(1)$. Since the initialization identifies a configuration near the optimum, training neural network(line 17-23) dose not take too much time.

3.5.2 Theoretical analysis

This section first represents jump discontinuity using a function(Eq. 3.10). Then convergence rate of rational function on jump discontinuity is analyzed(Theorem 3.5.1). With the help of Lemma 3.5.2, we prove Theorem 3.5.1. Similarly, convergence rate of polynomial function(Theorem 3.5.3) is also provided.

We found that $f_{\sigma=1} = a|x| + bx$ and $f_{\sigma=2} = a\frac{|x|}{x} + bx$ can characterize single jump discontinuity. For example, when a=b=1/2 and σ=0, $f_{1,2}$ is ReLU function. It is $sign(x)$ when a=1 and b=1, and σ=1. Thus, $f_{1,2}$ rotates or change the angle between two lines at jump discontinuity

Algorithm 1: RationalNet

Input: a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$,
rational function order: m,
graph signal on nodes: Y(i), i $\in 1, 2, ..., |\mathcal{V}|$
Output: a rational function with parameters: ψ_i and ϕ_i

compute graph Laplacian: $\mathbf{L} = A - D$
compute spectral signal by graph Fourier transform: $\hat{Y} = \mathbf{U}^\intercal Y$
perform eigen decomposition: $\mathbf{L} = \mathbf{U}\,\Lambda\,\mathbf{U}^\intercal$
n $\leftarrow$ m
// initialize parameters by a relexed Remez
repeat
    Pick m + n + 2 points $x_0, x_1, ..., x_{m+n+1}$ from full data X with equal interval
    r = 0, $E_r = 0$
    repeat
        solve $\psi_{0\sim m}, \phi_{1\sim n}, E_{r+1}$ ▷ Eq. 3.8 or 3.9
    until $E_{r+1} - E_r$ *convergence*;
    form a Padé rational function $R_{\psi,\phi}$ with $\psi_{0\sim m}, \phi_{1\sim n}$
    compute residues $\delta_d = |\hat{Y}(d) - R(x_d)|$
    m $\leftarrow$ m-1 or n $\leftarrow$ n-1 in turns.
until δ *convergence or* $\delta_{min,max} < |E|$;
// initialize a rational function with the above coefficients
repeat
    form a Padé rational function $R_{\psi,\phi}$ with $\psi_{0\sim m}, \phi_{1\sim n}$ obtained in the above repeat loop
    $R(\mathbf{L})x = \mathbf{P}(\mathbf{L})\,\mathbf{Q}(\mathbf{L})^{-1}x$ ▷ Eq. 3.4 or 3.5
    $\theta = \{\psi_i, \phi_j\}$
    compute the mean error function $\mathcal{L} = \mathbf{MSE}(R(x) - Y)$
    compute derivatives to update parameters: $\theta \leftarrow \theta + \beta\nabla_\theta\mathcal{L}$, where β is learning rate
until MSE *converges*;

based on $|x|$ and x. These two functions can be rewritten in an uniform formula:

$$f_{1,2} = a\frac{|x|}{x^{\sigma\in\{0,1\}}} + bx \tag{3.10}$$

where $a, b \in \mathbb{R}$.

Theorem 3.5.1 (convergence rate of rational approximation on jump discontinuity)**.** *Given $n \geq 5$ and $b \geq 1$, there exist a rational function $R_n(x)$ of degree n that satisfies*

$$\sup_{x\in[-c,c]} |f_{1,2} - R_n(x)| \leq Ce^{-\sqrt{n}}.$$

In our proof of Theorem 3.5.1, for $n \in \mathbb{N}$, we follow [103] and define the Newman polynomial: $\mathbf{N_n}(x) := \prod_{i=1}^{n-1}(x + \alpha_n^i)$, where $\alpha_n := e^{-1/\sqrt{n}}$. To approximate jump discontinuity, define $A_n(x)$ as Newman approximation:$A_n(x) := x\frac{\mathbf{N_n}(x)-\mathbf{N_n}(-x)}{\mathbf{N_n}(x)+\mathbf{N_n}(-x)}$.

Lemma 3.5.2. *Given $n \in [5, \infty) \cap \mathbb{Z}$, $c \in [1, +\infty)$, $\sigma \in \{0, 1\}$*

$$\sup_{x\in[-c,c]} \left| \frac{|x|}{x^\sigma} - \frac{cA_n(x/c)}{x^\sigma} \right| \leq 3ce^{-\sqrt{n}}. \tag{3.11}$$

proof for Lemma 3.11. **If σ=0**, left of Eq. 3.11 is equivalent to

$$\left| |x| - cA_n(\frac{x}{c}) \right| = \left| c(|\frac{x}{c}| - A_n(\frac{x}{c})) \right| = c \left| |\frac{x}{c}| - A_n(\frac{x}{c}) \right|.$$

Since $|\frac{x}{c}|$ and $A_n(\frac{x}{c})$ are both even, it suffices to consider the case when $0 \leq x \leq c$. **For** $x \in [0, c\alpha_n^n = ce^{-\sqrt{n}}]$, since $\mathbf{N_n}(x) \geq \mathbf{N_n}(-x) \geq 0$ so that $\frac{a}{c}A_n(\frac{a}{c}) \geq 0$:

$$c \left| |\frac{x}{c}| - A_n(\frac{a}{c}) \right| \leq c \left| |\frac{x}{c}| \right| = x \leq ce^{-\sqrt{n}} < 3ce^{-\sqrt{n}}.$$

For $x \in (c\alpha_n^n = ce^{-\sqrt{n}}, c]$:

$$\begin{aligned}
& c \left| |\frac{x}{c}| - A_n(\frac{x}{c}) \right| \\
=& c \left| \frac{x}{c} - A_n(\frac{x}{c}) \right| && (\frac{x}{c} > 0) \\
=& c \left| \frac{x}{c} - \frac{x}{c}\frac{\mathbf{N_n}(\frac{x}{c}) - \mathbf{N_n}(-\frac{x}{c})}{\mathbf{N_n}(\frac{x}{c}) + \mathbf{N_n}(-\frac{x}{c})} \right| && (\text{definition of } A_n) \\
=& 2x \left| \frac{\mathbf{N_n}(\frac{x}{c})}{\mathbf{N_n}(-\frac{x}{c})} + 1 \right|^{-1} \\
\leq& 2c\frac{x}{c} \left[\left| \frac{\mathbf{N_n}(\frac{x}{c})}{\mathbf{N_n}(-\frac{x}{c})} \right| - |-1| \right]^{-1} && (|a-b| \geq |a| - |b|) \\
\leq& 2c \left[\left| \frac{\mathbf{N_n}(\frac{x}{c})}{\mathbf{N_n}(-\frac{x}{c})} \right| - 1 \right]^{-1} && (\frac{x}{c} \leq 1) \\
\leq& \frac{2c}{e^{\sqrt{n}} - 1} && (\text{Lemma 3.2, Ch. 7, [89]}) \\
\leq& \frac{2c}{e^{\sqrt{n}} - \frac{1}{3}e^{\sqrt{n}}} && (\frac{e^{\sqrt{n}}}{3} \geq \frac{e^{\sqrt{5}}}{3} \approx \frac{3.19}{3} > 1) \\
=& 3ce^{-\sqrt{n}}.
\end{aligned}$$

If σ=1, Following same procedure as σ=0, we have:

$$\left| \frac{|x|}{x} - \frac{cA_n(x/c)}{x} \right| \leq 3ce^{-\sqrt{n}}.$$

□

proof for Theorem 3.5.1. Applying Lemma 3.11:

$$\begin{aligned}|f_1 - R(x)| =& \left|(ax + b|x|) - (ax + bcA_n(\frac{x}{c}))\right| \\ =& b\left||x| - cA_n(\frac{x}{c})\right| \leq 3bce^{-\sqrt{n}}.\end{aligned}$$

Similarly, we have:

$$\begin{aligned}|f_2 - R(x)| =& \left|(ax + b\frac{|x|}{x}) - (ax + b\frac{cA_n(x/c)}{x}))\right| \\ =& b\left|\frac{|x|}{x} - \frac{cA_n(x/c)}{x}\right| \leq 3bce^{-\sqrt{n}}.\end{aligned}$$

In sum,

$$\sup_{x\in[-c,c]} |f_{1,2} - R_n(x)| \leq Ce^{-\sqrt{n}},$$

where $C = 3bc$ in Theorem 3.5.1 □

By Bernstein's theorem[3], polynomials can approximate a function with:

$$||x| - P_n(x)| \leq \frac{\beta}{n},$$

where $P_n(x)$ is a polynomial function of degree of n, and $\beta \approx 2.801$[140]. Using the same settings for the rational function, we have a similar result for polynomials:

Theorem 3.5.3 (convergence rate of polynomial approximation on jump discontinuity)**.** *Given $n \in [5, \infty) \cap \mathbb{Z}$, $c \in [1, +\infty)$, $\sigma \in \{0, 1\}$:*

$$\sup_{x\in[-c,c]} |f_{1,2} - P_n(x)| \leq \frac{C\beta}{n},$$

where $P_n(x)$ is a polynomial function of degree n, and C=3bc.

In a nutshell, when the order is large or equal to 5, polynomial converges linearly regarding the order number, while rational function converges exponentially.

3.5.3 Effect Analysis of Inverse Graph Laplacian in Graph Convolution

The major difference between rational spectral filter and polynomial spectral filter is whether there is polynomial of inverse graph Laplacian. The inverse of the graph Laplacian is a key object in online learning algorithms for graphs [54]. By using the concept of conductance, if f is a heat distribution over the vertexes, then $\mathbf{L}(f)$ indicates the flux induced by f over the graph. Then by the representer theorem [6, 122], namely that $f(\mathcal{V}_i) = \mathbf{L}^{-1}(\mathbf{L}(f))$ could be interpreted as saying that the heat at each vertex can be expressed concerning or derived from the flux through every vertex.

Thus, if $\mathbf{L}$ sends a heat distribution f over each node to flux through each vertex, then $\mathbf{L}^{-1}$ sends some of the fluxes over the graph back to the original heat distribution (i.e., keep part of fluxes itself). Going back to the graph learning application, we first translate our updated "heat distribution" to flux through all of those nodes by calculating $\mathbf{P}(\mathbf{L}(f))$. M-th degree of $\mathbf{P}(\cdot)$ means that each vertex can update M-th neighbors at most. Then using another updated flux in the reverse direction, $\mathbf{Q}(\mathbf{L}(f))^{-1}$ will adjust or reduce flux within N-th neighbors.

For polynomial spectral filter, more layers or higher degree involve more neighbors, and the model thereby should have better capacity. In practice, if the number of layers or degree is more than what is needed, it is unavoidable to be over-smooth (e.g., all nodes are similar). However, the best number of layer or degree is difficult to calculate unless trying all possible options. On the other hand, "sending back" behavior of rational spectral filter alleviates the over-smoothing issue klicpera2018predict, since it always limits the out-degree flux even if excessive graph convolutional layers or approximation degrees are added.

3.6 Evaluation

This section elaborates evaluation with a detailed analysis of the behaviors of the proposed method on synthetic and real-world graphs.

3.6.1 Training Setting and Baselines

The input include a graph Laplacian $\mathbf{L}$, a graph signal residing on each vertex Y, and raw feature x . In a nutshell, we aims at finding a function f that satisfies $Y = f(\mathbf{L}, x)$:

$$Y = \mathbf{U}\, \mathrm{g}_\theta(\mathbf{\Lambda})\, \mathbf{U}^\intercal\, x = \mathrm{g}_\theta(\mathbf{L})x, \tag{3.12}$$

where $\mathrm{g}_\theta(\mathbf{\Lambda})$ is set to be jump function such $|x|$ and $sign(x)$. In previous works, raw features and filtering signal are fed into the model to fit the graph signal. However, raw features have an impact on fitting graph signal, which distracts the analysis of filtering behaviors. As discussed in Section 6.4, x is set to be eigenvector $\mathbf{U}$ which is a uniform signal in spectral

domain, so that we can focus on the behaviors of approximation methods. We compare Rational Net(RatNet or RNet) against several stat of art regression models:

- Linear Regression(LR)
- Polynomial Regression(PR) [131]
- Passive Aggressive Regression(PAR) [27]
- LASSO[137]
- Epsilon-Support Vector Regression(SVR)[129]. Three kernels were applied: linear(L), polynomial(P) and RBF(R).
- Ridge Regression(RR)[105]
- Bayesian Ridge Regression(BR)[94],
- Automatic Relevance Determination(ARD)[138]
- Elastic Net(EN)[169]
- Orthogonal Matching Pursuit(OMP)[95]
- SGD Regression
- Huber Regression [56]
- ChebNet[32]. PolyNet is proposed by replacing Chebyshev polynomial with normal polynomial.

3.6.2 Experiments on Synthetic Data

To validate the effectiveness of RationalNet, we conduct a simulated test with synthetic data. The task is to recover signal on the vertexes, which is a regression problem. Specifically, we generated a graph comprised of several subgroups. The edge amount for each vertex in the same subgroup is randomly chosen between 0 and 8, while the links among different subgroups are sampled between 0 and 3. Experiments were conducted on a 500-node and a 1000-node graph. Two types of jump signals are fed into this network structure: $|x|$ and $sign(x)$. Since all eigenvalues are normalized into range [0, 1], jump points of $|x|$ and $sign(x)$ are moved into the same range. Specifically, we used $|x - 0.5|$ and $sign(x - 0.5)$. Detailed results are shown in Table 3.2 and 3.1.

In 1000-node graph test on $|x|$(first two columns in Table 3.1), PolyFit achieved the second lowest MSE(0.0016 for S-ERR). PolyNet's MSE(0.0016) is the same as that of PolyFit, which shows the power of polynomial regression. Chebyshev polynomial(ChebNet) dose not improve PolyNet, which implies that neural network might approximate the best coefficients of polynomials no matter what type of polynomial is used. LR, RR, LASSO, EN, OMP, BR, ARD, SGD SVR(L/P) performed at the same level(0.0015-0.0018). Our method(5e-6) significantly outperformed all the baselines by a large margin. Both the errors in spectral domain and vertex domain show the advantage of RationalNet. The Similarly, PolyFit and

Method	S-ERR($\|x\|$)	V-ERR($\|x\|$)	S-ERR($sign(x)$)	V-ERR($sign(x)$)
SVR-R	.0044±.0000	.0043±.0000	.3840±.0000	.2573±.0000
SVR-L	.0165±.0000	.0111±.0000	.3218±.0000	.2799±.0000
SVR-P	.0179±.0000	.0131±.0000	.3587±.0000	.2573±.0000
LR	.0161±.0000	.0110±.0000	.3211±.0000	.2788±.0000
RR	.0160±.0000	.0110±.0000	.3199±.0000	.2786±.0000
LASSO	.0157±.0000	.0137±.0000	.5581±.0000	.5087±.0000
EN	.0157±.0000	.0137±.0000	.5969±.0000	.5438±.0000
OMP	.0161±.0000	.0110±.0000	.3211±.0000	.2788±.0000
BR	.0161±.0000	.0110±.0000	.3210±.0000	.2788±.0000
ARD	.0161±.0000	.0110±.0000	.3210±.0000	.2788±.0000
SGD	.0152±.0000	.0116±.0000	.3191±.0001	.2795±.0003
PAR	.2871±.0997	.2740±.1033	1.0370±.8892	.9745±.8418
Huber	.0202±.0000	.0123±.0000	.3219±.0000	.2794±.0000
PolyFit	.0016±.0000	.0010±.0000	.2057±.0000	.1703±.0000
ChebNet	.0021±.0000	1.1904±.0052	.2058± .0067	.2084± .0043
PolyNet	.0016±.0000	.0038±.0000	.2011±.0095	.2001±.0056
RNet	**5.2971e-6±1.2501e-8**	**.0001±.00000**	**.0103±.0001**	**.0153±.0006**

Table 3.1: 1000-node graph test: s-err indicates error in spectral domain, while v-err represents error in vertex domain.

PolyNet and SVR(R) performed better than all the baselines except RationalNet. Our method still achieves the lowest MSE(0.004619 for S-ERR). The 500-node graph experiment(Table 3.1)) also demonstrates the effectiveness of RationalNet.

Regression behaviors on synthetic data is shown in Fig. 3.2. Methods (SVR(L), Ridge, OMP, LASSO, Linear regression, ENet, ARD, Huber, etc.) fitted the $|x|$(upper of Fig. 3.2) using a straight Line, while better baselines(SVR(R)), PolyFit, PolyNet, ChebNet) approximate the function with curves. RationalNet almost overlapped with the target function which makes its MSE very small(5e-6). Similarly, in lower part of Fig.3.2, the methods using straight lines(SVR(L), Ridge, OMP, SGD, LASSO, BR, Huber LR, ENet, ARD) performed relatively bad. Fitting with curves, PolyFit, PolyNet, ChebNet improved the performance by a large margin. Similarly, RationalNet overlapped the signal and achieved the lowest error.

Since RationalNet initializes parameters by a relaxed Remez algorithm, we analyze the performance of neural networks and Remez respectively. As shown in Table 3.3, the first two rows show the MSE of Remez only. Compared with Remez algorithm, RationalNet without Remez initialization(3rd and 4th lines) performed badly. On the contrary, RationalNet with Remez initialization(5th and 6th lines) improved the Remez by 56.26% and 81.39%(7th line) for $|x|$ and $sign(x)$ separately in spectral domain, which also reduces their MSE in vertex domain by 9.37% and 14.93%(8th line). This result illustrates that Remez and RationalNet

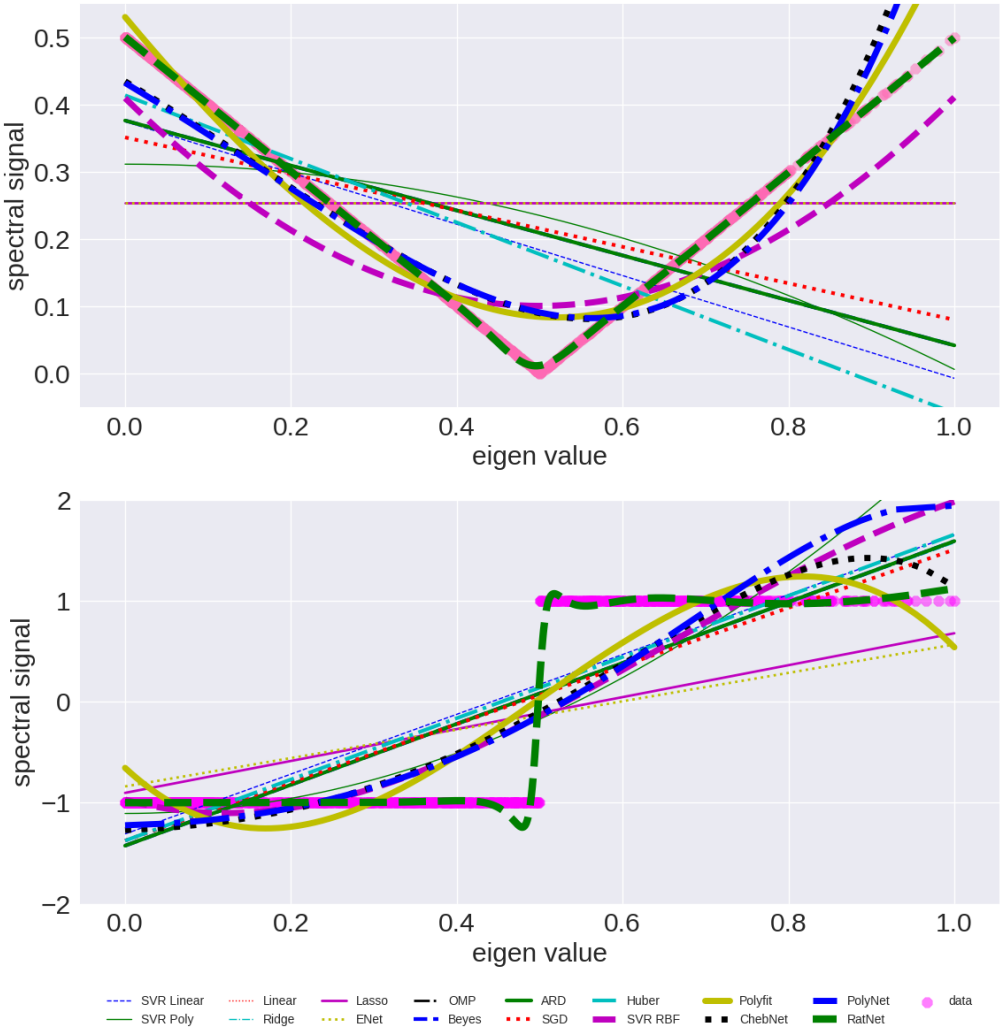


Figure 3.2: Regression comparison on $|x|$ and $sign(x)$.

Method	S-ERR(|x|)	V-ERR(|x|)	S-ERR($sign(x)$)	V-ERR($sign(x)$)
SVR-R	.0043±.0000	.0044±.0000	.2691±.0000	.2867±.0000
SVR-L	.0148±.0000	.0131±.0000	.2612±.0000	.2748±.0000
SVR-P	.0137±.0000	.0138±.0000	.2784±.0000	.2875±.0000
LR	.0140±.0000	.0130±.0000	.2582±.0000	.2734±.0000
RR	.0140±.0000	.0130±.0000	.2579±.0000	.2741±.0000
LASSO	.0135±.0000	.0137±.0000	.4723±.0000	.4865±.0000
EN	.0135±.0000	.0137±.0000	.5260±.0000	.5374±.0000
OMP	.0140±.0000	.0130±.0000	.2582±.0000	.2734±.0000
BR	.0140±.0000	.0130±.0000	.2581±.0000	.2734±.0000
ARD	.0140±.0000	.0130±.0000	.2581±.0000	.2734±.0000
SGD	.0135±.0000	.0138±.0000	.2597±.0007	.2764±.0008
PAR	.4026±.3980	.3982±.3954	.7412±.5682	.7456±.5029
Huber	.0158±.0000	.0135±.0000	.2581±.0000	.2734±.0000
PolyFit	.0010±.0000	.0011±.0000	.1488±.0000	.1699±.0000
ChebNet	.0044±.0000	.0044±.0000	.2025±.0000	.2115±.0004
PolyNet	.0016±.0000	.0016±.0000	.2059±.0000	.2083±.0004
RNet	**.0001±.0000**	**.0001±.0000**	**.0108±.0001**	**.1479±.0001**

Table 3.2: 500-node graph test: s-err indicates error in spectral domain, while v-err represents error in vertex domain.

cannot find the optimum independently. Therefore, it is reasonable to integrate these two methods for optimizing the coefficients.

3.6.3 Case Study on Real-world Scenario

In this section, we study a traffic congestion signal on Minnesota state-level road network [2] and Fairfax county-level road network VA[3][14]. Specifically, the signal is a high-pass filtering which can be written as $\zeta = \frac{sign(x-0.5)+1}{2}$ in Fourier domain. ζ is a threshold function sets the output to 0 when normalized eigenvalues $\in [0, 0.5)$, and 1 for $\in (0.5, 1]$. Therefore, this function filter out signal of low frequency. The physical meaning of the convolutional operation is a weight function that chooses the eigenbasis(φ_i) to fit the traffic signal Y. The top line of Fig. 3.3 shows several examples in eigen space of Minnesota road networks. First two sub figures are the 2nd and 3rd eigenvector φ_1, φ_2 on vertex domain: φ_1 emphasizes the south of Minnesota(red area), while φ_2 highlights the capital St. Paul and its biggest city Minneapolis. Note that the 1st eigenvector φ_0 is a constant vector for any connected graph. φ_1, φ_2 correspond to λ_1, λ_1, which represent the first two lowest frequencies. As these figures

[2]https://www.cise.ufl.edu/research/sparse/matrices/Gleich/minnesota.html
[3]https://github.com/gboeing/osmnx

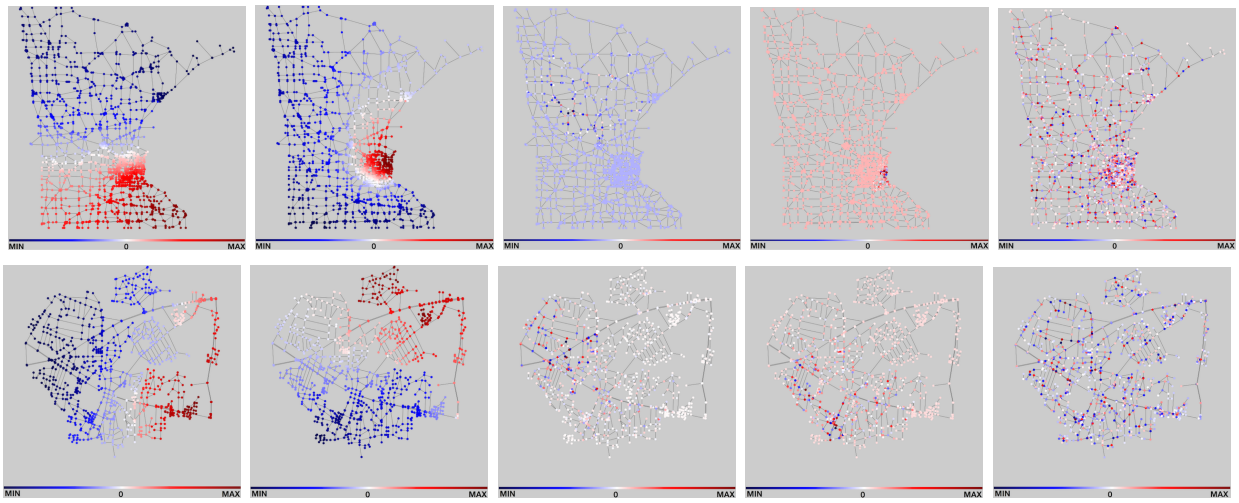


Figure 3.3: Top line: Minnesota road network. From left to right are φ_1,φ_2,φ_{1001},φ_{1002}, and ζ. $\mathcal{E}_{\varphi_1}$=0.00084ïijN̆ $\mathcal{E}_{\varphi_2}$=0.00207, $\mathcal{E}_{\varphi_{2001}}$=4.03932, $\mathcal{E}_{\varphi_{2002}}$=4.04661, $\mathcal{E}_{\zeta}$=15384.10112. **Bottom line**: Fairfax road network. From left to right are φ_1,φ_2,φ_{701},φ_{702}, and ζ. $\mathcal{E}_{\varphi_1}$=0.00250, $\mathcal{E}_{\varphi_2}$=0.00296, $\mathcal{E}_{\varphi_{701}}$=3.82741, $\mathcal{E}_{\varphi_{702}}$=3.81540, $\mathcal{E}_{\zeta}$=6376.54224

show, low frequencies represent smooth signals, which means that the neighbors of each node are likely to have similar signal value. By contrast, high-frequency basis captures non-smooth component as the 3rd and 4th sub figures show: signal values vary frequently in some areas. Combining top 50% high-frequency eigenbasis, the last sub figure shows the ζ signal on the graph. In addition, the degree of non-smoothness of signal regard graph structure can be evaluated quantitatively by Dirichlet energy([125]): $\mathcal{E}_{\varphi_i} = \varphi_i^{\intercal}\,\mathbf{L}\,\varphi_i$. Dirichlet energy of examples is shown in the caption of Fig. 3.3. Eigenvectors of low frequency($\varphi_{1,2}$) are smooth, so their Dirichlet energies are low. While high-frequency eigenvectors are less smooth since their Dirichlet energy is higher(around 4.04). Summing up the top 50% high frequencies,

	$\lvert x\rvert$	$sign(x)$
Remez (spectral)	4.531041e-6	0.057233
Remez (vertex)	0.000105	0.109641
RNet w/o Remez (spectral)	0.000145	1.364790
RNet w/o Remez (vertex)	0.000252	0.887602
RNet w/ Remez (spectral)	1.981569e-6	0.010645
RNet w/ Remez (vertex)	9.569891e-5	0.093268
Improved (spectral)	56.26%	81.39%
Improved (vertex)	9.37%	14.93%

Table 3.3: Remez and RationalNet on 1000-node graph: MSE improvement in spectral and vertex domain

Method	S-ERR(FF)	V-ERR(FF)	S-ERR(MI)	V-ERR(MI)
SVR-R	.0364±.0000	.0406±.0000	.0393±.0000	.0358±.0000
SVR-L	.0652±.0000	.0599±.0000	.0670±.0000	.0627±.0000
SVR-P	.1226±.0000	.1014±.0000	.0518±.0000	.0499±.0000
LR	.0640±.0000	.0595±.0000	.0662±.0000	.0621±.0000
RR	.0639±.0000	.0595±.0000	.0662±.0000	.0621±.0000
LASSO	.2026±.0000	.2030±.0000	.2141±.0000	.2138±.0000
EN	.1595±.0000	.1594±.0000	.1609±.0000	.1592±.0000
OMP	.0640±.0000	.0595±.0000	.0662±.0000	.0621±.0000
BR	.0640±.0000	.0595±.0000	.0662±.0000	.0621±.0000
ARD	.0640±.0000	.0595±.0000	.0662±.0000	.0621±.0000
SGD	.0639±.0001	.0598±.0000	.0664±.0000	.0622±.0000
PAR	.4960±.3273	.4948±.3200	.4255±.4575	.4222±.4588
Huber	.0646±.0000	.0597±.0000	.0666±.0000	.0624±.0000
PolyFit	.0346±.0000	.0382±.0000	.0384±.0000	.0346±.0000
ChebNet	.0468±.0006	.0468±.0006	.2336±.0094	.2336±.0094
PolyNet	.0468±.0006	.0468±.0006	.0490±.0049	.0490±.0009
RNet	**.0064±.0007**	**.0064±.0007**	**.0046±.0012**	**.0046±.0006**

Table 3.4: Regression comparison on Fairfax(FF) and Minnesota(MI) road networks. s-err indicates error in spectral domain, while v-err represents error in vertex domain.

the Dirichlet energy of ζ in the last sub figure is very large(15384.10). The bottom line of Fig. 3.3 shows similar examples from Fairfax road networks. φ_1 highlights Fair City Mall(red area) and the road to this mall, while φ_2 underlines Fairfax Circle Shopping Center and a residential neighborhoods nearby. Similarly, φ_{701} and φ_{702} show two non-smooth graph signals. Summing up top 50% high frequencies, the 5th sub figure exhibits an extremely non-smooth signal. Characterizing non-smooth graph signal or high frequencies is not a trivial task. Therefore, approximating this high pass filtering is significantly challenging. Table 3.4 shows

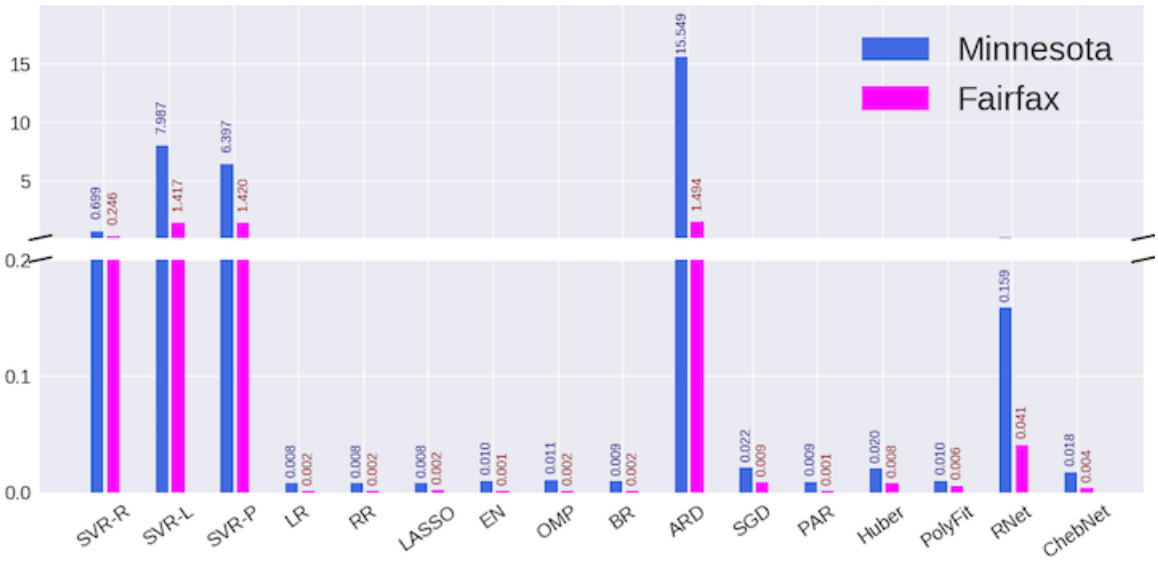


Figure 3.4: Comparison of average running time in seconds.

similar results as in synthetic data: The proposed method still performed much better(3e-5) than the baselines. PolyFit achieved the second best level(0.0008), ChebNet, PolyNet and SVR(RBF) are generally good(0.0039,0.0039 and 0.0055), this is probably because they fitted the target with curves. The methods using straight lines have highest level of MSE(around 0.01). The results on another dataset, Fairfax road, also show that RationalNet has huge advantage beyond the baselines.

Fig. 3.4 shows the comparison of running time on two real-world networks. Minnesota dataset contains 2642 vertexes, while Fairfax network consists of 993 nodes. Most baseline methods are efficient such as LR, RR, LASSO, EN, OMP, BR, PAR. They finish computing within around 0.01 second on Minnesota graph and 0.002 second on Fairfax graph. SGD, PolyFit, and Huber only require 0.02 and 0.01 for Minnesota and Fairfax network respectively. SVR group performed slower, but they complete the calculation within 10 seconds for Minnesota graph and 2 seconds for Fairfax. ARD needs around 15 seconds and 1.4 seconds separately, which is the slowest baseline. Note that the number for RationalNet and ChebNet in Fig. 3.4 is the time for each iteration. RationalNet took 0.159 seconds for one update on Minnesota network, and 0.041 seconds on Fairfax network. In practice, RationalNet often converges within 300 iterations, which takes less than one minute for both datasets. Due to the

complexity of computation, it is natural that RationalNet is slower than its counterpart ChebNet and several baselines. However, it shows that our algorithm can run reasonably fast in real-world datasets. Our case study on real-world graph justifies that RationalNet can accurately estimate the high pass filter within a reasonable time.

3.7 Conclusion

In this paper, we have introduced a neural network model for graph signal recovering. To estimate jump discontinuity, a rational function is employed due to its powerful ability of approximation. The proposed method can avoid multiplication with the eigenvector matrix. With the help of a relaxed Remez algorithm, RationalNet can identify the optimal configuration. In theory, RationalNet obtains exponential convergence rate on jump signal, significantly fast than the polynomial-based approximation. Experiments on synthetic datasets suggest that the proposed RationalNet model is capable of model typical jump function accurately.

Chapter 4

Robust Approximation for Graph Convolution

In this chapter, we propose a framework that serves accurate and robust approximation for graph convolution. Inspired by optimization theory for traditional rational approximation, the proposed frameworks integrate multiple theoretical constraints in a deep neural network to achieve robust and quick estimation for graph filter.

4.1 Introduction

The need for approximation arises due to that one has a limited set of data points and wants to determine the underlying functional form. A suitable approximation by simpler functions can save a considerable amount of computation. For example, the function value may be the result of many complex calculations, and it may be time-consuming to calculate each function value. By approximating closed form, one could obtain approximate function values much quicker. A recent example is graph convolutional networks[32, 65] which apply Chebyshev approximation to avoid expensive matrix decomposition and multiplication. Another benefit of simple representation is that some following manipulations(e.g., differentiation or integration) can be performed easily. Also, the closed form approximation explores the direction of machine learning on small data because the traditional method only requires very few data to approximate the underlying function.

Approximation techniques are concerned how approximate complex functions using simpler functions, quantitatively characterizing the errors. This is often done with polynomial or rational approximations. Polynomial function models have well-understood properties. However, they are notorious for oscillations between exact-fit values and have insufficient capacity in modeling asymptotic phenomena. On the contrary, rational function models can accept a much wider range of shape than polynomial family, and they are typically smoother and significantly less oscillatory than polynomial models. Furthermore, a rational function can model complicated structure with a fairly low degree in both the numerator

Features\ Model	Remez	Neural Net	RemezNet
Analytical Form	√	×	√
Few Parameters	√	×	√
Theoretical Optimality	√	×	√
Small Data	√	×	√
Robustness/Feasibility	×	√	√
Free of Hard Threshold	×	√	√

Table 4.1: Features of Remez, Neural Net and RemezNet

and denominator. Theoretically, polynomials require degree $\Omega(\text{poly}(1/\epsilon))$ to approximate functions near singularities and on an unbounded domain, while rational functions only need $\mathcal{O}(\text{poly log}(1/\epsilon))$ to achieve ϵ-close[84, 136].

Numerous studies have shown the universal approximation property of neural networks in function approximations[136, 84, 38, 9]. For example, Word2Vec is found to approximate co-occurrence matrix factorization[77, 76, 83], avoiding the expensive computational cost of matrix decomposition. The conjecture behind Word2Vec is that a word can be represented by its neighbors, which is equivalent to calculating word co-occurrence. Thus, Word2Vec is another version of matrix decomposition without a high computational overhead. Similarly, SpectralNet[124] approximates the eigen-decomposition on similarity matrix by imposing feature orthogonalization, preventing itself from unaffordable computational expense. This shows that neural networks are capable of approximating underlying function if proper constraints are applied.

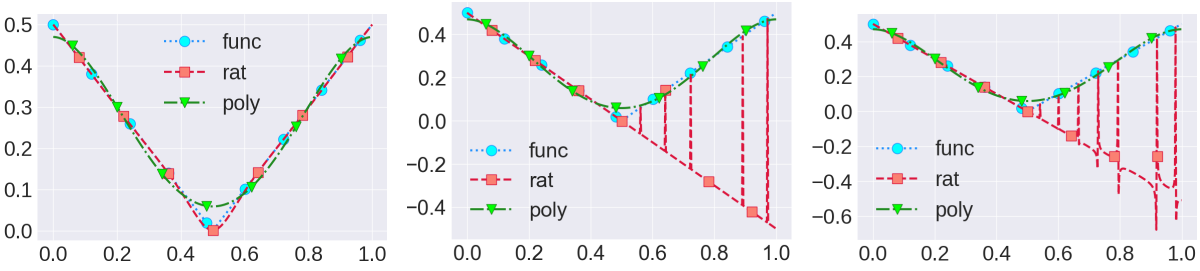


Figure 4.1: Rational(rat) and polynomial(poly) approximation are compared for a non-smooth functions($func = |x - 0.5|$). degrees of rat from left to right: 5, 7, 9. The results of degree 7 and 9 are highly underfitting.

However, the existing methods for function approximation still face several critical challenges: **(1) Traditional method for deriving closed form is non-robust.** As state of the art, Remez algorithm theoretically guarantees the optimality of rational approximation. Nevertheless, it requires to carefully choose the degree of the rational function. Otherwise, this algorithm often makes itself infeasible due to the singular coefficient matrix, or it may cause large approximation errors. Fig. 4.1 illustrates the issue of non-robustness regarding degree. **(2)**

General neural networks do not represent function using an analytical expression. Deep neural networks have been shown to outperform previous state-of-the-art machine learning techniques due to the tremendous ability in approximation. Notwithstanding, its intrinsic property hinders itself from deducing a simple closed form, which makes it unable to enjoy the benefit of low computation complexity. **(3) There is no study found to combine Remez and deep learning for function approximation.** Without proper constraints, there is no guarantee for neural networks to approach to the optimal approximation, while Remez derives an accurate expression only under very harsh conditions. Although deep neural networks and Remez are widely known for their different advantages in function approximation, we did not find any work that jointly exploits their abilities while remedying these defects.

In sum, as shown in Table 4.1: Remez has strong theoretical support to generate an accurate analytical form with few data and few parameters, but it is very sensitive to the degree, and its convergence relies on the hard threshold. While neural networks do not give simple closed form, and it often requires a large number of parameters and data. Besides, theoretical optimality of analytical approximation for neural networks is still missing.

To simultaneously address all these technical challenges, this paper presents a novel neural network named RemezNet which is capable of handling the infeasibility problem and secure the model's robustness. To characterize the optimal solution, equioscillation constraint is employed so that the optimality is theoretically guaranteed. By incorporating minimax constraints of the Remez algorithm, we propose an objective function that minimizes the error value at extreme points, which ensures the global optimum. An initialization policy is designed for RemezNet inspired by the equioscillation theorem. The rational representation is initialized with Padé normalization and alternating sign to boost the convergence performance. In a nutshell, the key innovations of our study are summarized below:

- **Proposing a novel framework that integrates the strengths of rational approximation and neural networks**: A neural network framework, namely RemezNet, is proposed for approximating function that utilizes optimality characterization of best rational approximation. RemezNet takes advantage of the neural network in function approximation, to avoid the drawbacks of traditional Remez, such as infeasibility issue.
- **Developing a set of objectives for optimizing approximation performance**: To achieve optimal approximation, we enforce the equioscillation theorem and the minimax theorem derived from traditional Remez into the proposed model. Combining those constraints with general regression loss, the optimality of rational representation can be identified.
- **Designing an efficient initialization for parameter update**: To expedite convergence of RemezNet, we proposed a special-purpose initialization policy that exploits equioscillation characterization and Padé normalization. This initialization shows significant superiority over state of the art initializations for normal neural networks.
- **Conducting extensive experiments for performance evaluations**[1]: The proposed method

[1]The code for replication will be released upon acceptance

was evaluated on benchmark functions with challenging singularity. Experimental results demonstrate that the proposed approach runs efficiently and consistently overcomes the disadvantage of traditional Remez.

In this chapter, we first review related works and then present the proposed model and describe the algorithm process. Next, we show the evaluation. Finally, we conclude by summarizing the study's important findings.

4.2 Related Work

This section provides a brief overview of related work in approximation theory and deep learning.

4.2.1 Approximation Theory and Remez Algorithm

Approximation theory is concerned with how to approximate functions with simpler functions. This is typically done with polynomial or rational approximations. Polynomials are familiar and comfortable, while rational functions seem complex and specialized. However, rational functions are more powerful than polynomials at approximating functions in challenging such as jump discountinouties. Basic properties of rational function are described in the literature [107, 114, 112]. In the past, function approximation was handled just by hand [23]. A minimax framework for this problem was originally done by Chebyshev [130] by stating equioscillation theorem that gave a necessary and sufficient condition for a minimax approximation function[31]. However, this series of works is not working well in practice. A Russian mathematician, Remez, introduced another algorithm that computes the minimax approximation [118, 109].

4.2.2 Deep Learning as Function Approximation

Neural networks have a wide range of reputation in function approximations [136, 84, 38, 9]. Neural networks can solve the old problem with the same constraint, and often improve the effectiveness and efficiency beyond traditional methods. Word2vec[100] is a group of related models that are used to produce word embeddings. Levy et al., found that Word2Vec is an approximation to PPMI matrix factorization [77, 76, 83]. As we all know, Word2Vec is more efficient than general matrix decomposition. Similarly, SpectralNet [124] naturally approximates the spectral decomposition to unseen data points, which also solves the out-of-sample-extension issue and scalability issue of spectral clustering. That is to say, neural networks can achieve the same goal with less cost if suitable constraints are employed.

The connection between Remez and deep learning has been well surveyed [136, 52], but their collaboration for closed form extraction has never been investigated. Inspired by these related works, we exert the power of deep neural networks and theoretical support of the Remez algorithm to solve function approximation problem.

4.3 RemezNet

This section formally defines the task of rational approximation and then describes our proposed model.

4.3.1 Problem Setting

Given a finite set of reference data points of a unknown function, i.e., $\{(x_0, y_0), (x_1, y_1), (x_2, y_2), ...\}$ where $x_i \in [a, b]$, and wants to determine the underlying rational representation $R(x)$ so that the residues r between $R(x)$ and original function $f(x)$ is minimized:

$$R(x)^* = \arg\min_{R(x)} r = \arg\min_{R(x)} \int_x |f(x) - R(x)| \, \mathrm{d}x, \tag{4.1}$$

where $x \in [a, b]$ and $R(x)$ is a rational function:

$$R(x) = R_{mn}(x) = \frac{P_m(x)}{Q_n(x)} = \frac{\sum_{i=0}^{\mu} \psi_i x^i}{\sum_{j=0}^{\nu} \phi_j x^j}, \phi_j, \psi_i \in \mathbb{R}, \tag{4.2}$$

where R_{mn} is a rational function whose maximum numerator degree is m and maximum denominator degree is n, i.e., $\mu \leq m$ and $\nu \leq n$. Therefore, we can rewrite Eq. 4.1 as:

$$R(x)^* = \arg\min_{\phi,\psi} \int_x |f(x) - \frac{\sum_{i=0}^{\mu} \psi_i x^i}{\sum_{j=0}^{\nu} \phi_j x^j}| \, \mathrm{d}x. \tag{4.3}$$

In practice, function can only be evaluated in a finite set of discrete points. Following Remez, two functions are considered to be identical if their difference is small enough. Therefore, the rational approximation is obtained when model optimizes the difference:

$$R = \arg\min_{R_{mn}} \max_{a \leq x \leq b} |f(x) - R_{mn}(x)|. \tag{4.4}$$

Without loss of generality, $[a, b]$ can be set to $[0, 1]$, since any function can be normalized and shifted by transformation $f(x) = \frac{x-a}{b-a}$.

4.3.2 Model Overview

RemezNet is a single layer neural network that has a rational function kernel. There are no more neural layers such as fully connected layers. Because the goal of RemezNet is to derive an analytical form, and any additional neural layer will make the learned denominator and numerator meaningless in analytical representation since the denominator and numerator are the coefficients of an intermediate function.

The model workflow is shown in Fig. 4.2: The input includes $m + n + 2$ of references points x and their function values Y. x is prepared as a power series for efficient matrix multiplication. Remez parameters Φ, Ψ are initialized by a special policy called **Padé and alternating sign initialization(PASI)**(see D in next subsection). Multiplying power series of x by Φ, Ψ separately, $P(x)$ and $Q(x)$ are obtained. After calculating $R(x)$, Remez computes three objectives which are **Minimax Loss(MM)**, **Equioscillation Loss(EO)** and **Mean Square Loss(MSE)**(see A/B/C in next subsection). Following the general training process of deep neural networks, the model performs back-propagation to update the parameters Φ, Ψ until convergence.

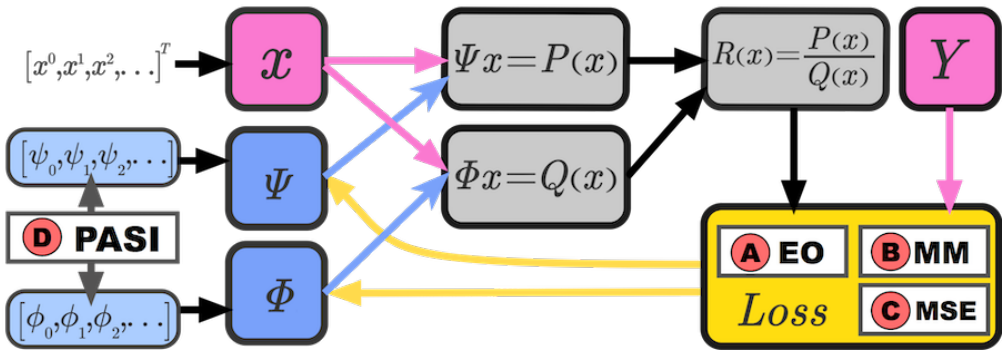


Figure 4.2: RemezNet training process on reference point x.

4.3.3 Model Details

By optimality theorem of minimax approximation[93], the optimal $R(x)$ is obtained by solving the system of equations:

$$f(x_d) - R_{mn}(x_d) = (-1)^d E, \tag{4.5}$$

where leveled error $E > 0$ and x_d is d-th reference points within $[a, b]$. The minimax approximation can be constructed by an iterative process referred to as the second algorithm of Remez[112]. Expanding Eq. (4.5), the equations become:

$$\sum_{j=0}^{\nu} \phi_j x_d^j f(x_d) - \sum_{j=0}^{\nu} \phi_j x_d^j (-1)^d E - \sum_{i=0}^{\mu} \psi_i x_d^i = 0, \tag{4.6}$$

where the term $\sum_{j=\nu}^{n} \phi_j x_d^j (-1)^d E$ is nonlinear since both ϕ_j and E are unknown. Remez[112] decomposes this term into $\sum_{j=\nu}^{n} \phi_j x_d^j (-1)^d E_l$ and $(-1)^d E_{l+1}$, where E_l is the value evaluated at lth iteration. $E_{l=0}$ can be set to 0 to linearize the system of non-linear equations:

$$\sum_{j=0}^{\nu} \phi_j x_d^j f(x_d) - \sum_{j=\nu}^{n} \phi_j x_d^j (-1)^d E_l - \sum_{i=0}^{\mu} \psi_i x_d^i = (-1)^d E_{l+1}, \tag{4.7}$$

then Remez proceeds to do solve this system of equation iteratively until E_l converges. However, this simplification incurs intrinsic inaccuracy and cannot ensure the feasibility of the system of equations. To avoid improper simplification, we propose a neural network to solve the non-linear Eq. 4.5 as it is. The challenge is that non-linear optimization is still a non-trivial problem for neural networks, and the model will be stuck in a local minimum if the constraint is not well designed. Following the optimality characterization of the rational approximation on approximating function, we propose four constraints, **[A] Equioscillation loss; [B] Minimax loss; and [C] Mean square loss; [D] Padé and alternating sign initialization**, as follows:

[A] Equioscillation Loss(EO):

Based on equioscillation of characterization of best approximants [139], a real function f has a unique best approximation $R^* \in R_{mn}$, and a function is equal to R^* if and only if $f - R$ equioscillates between at least $m + n + 2 - d$ extreme points, where d is the defect of r in R_{mn}. It is proved that the Equioscillation is equivalent to optimality[139].

Unfortunately, there is no theoretical evidence that characterizes the optimal setting about μ and ν. Without loss of generality, we assume that there is no share zeros, and μ, ν are set to be m, n respectively. By the equioscillation theorem[99], the model requires at least $\tau = m+n+2-\delta$ extreme points, where the defeat number $\delta = \min\{m-\mu, n-\nu\}$. We require that the residues $r = \{r_0, r_1, ..., r_{\tau-1}\}$ have alternating signs to satisfy the equioscillation. Given the degrees of rational function, i.e., m and n, there are two options of sign sequence for residues: setting the first term to positive or negative. Due to the symmetry, these two options are equivalent. Formally,

$$\forall r_d, r_d \cdot (-1)^{d+\sigma} = \|f(x_d) - R(x_d)\| \cdot (-1)^{d+\sigma} \geq 0,$$

where $d \in [0, \tau - 1] \subset \mathbb{Z}$ is the reference index and $\sigma \in \{0, 1\}$. Without loss of generality, RemezNet simplifies sign configuration by enforcing the sign of first residue to be positive, i.e., $\sigma = 0$. Accordingly, the second residue is automatically set to negative, the third residue is positive and so on. Therefore, penalty is the absolute value of the residue if it dose not satisfy equioscillation constraint, and 0 for the others. Specifically, for the residue at dth: $r_d = f(x_d) - R_{mn}(x_d)$, we have:

$$\mathcal{L}_{EO} = \sum_{d}^{\tau-1} \mathcal{L}_{eo}, \tag{4.8}$$

where $\mathcal{L}_{eo}$ is defined as:

$$\mathcal{L}_{eo} = \begin{cases} |r_d| & (-1)^d * r_d < 0 \\ 0 & (-1)^d * r_d \geq 0 \end{cases}$$

[B] Minimax Loss(MM):

Remez algorithm is a minimax approximation algorithm. The convergence theorems for the Remez algorithm on rational approximation are given by [116]. According to the minimax characterization theorem[114], the optimal function minimizes the maximum value of residues:

$$\min \|f(x) - R(x)\|_\infty, x \in \mathcal{E}$$

where $\mathcal{E}$ are extreme points collection of residue function. However, the reference points x_d cannot be easily initialized using extreme points. Therefore, reference points are iteratively exchanged by Remez algorithm to approach extreme points. In practice, the model initializes reference points as Chebyshev nodes[139] or points with equal interval. Since Chebyshev nodes has no significant superiority and require additional computation cost, reference points are initialized randomly with equal intervals. From the equioscillation theorem, it follows that the error function has $m + n + 1$ zeros, i.e., z_i. We compute the roots using any numerical method and consider the $m + n + 2$ intervals:

$$[0, z_0], [z_0, z_1], [z_1, z_2]..., [z_{m+n-1}, z_{m+n}], [z_{m+n}, 1], \tag{4.9}$$

where $z_0, z_1, ..., z_n$ are the $m + n + 1$ roots. For each interval above, we find the point at which the error functions attains its extreme points such as maximum or minimum value. We can do this by differentiating the error function and locating the minimum or maximum in each interval. If it happens that the extreme point doesnâĂŹt exist, the model computes the value of the error at the two end points and take the one with the largest absolute value. This provides us with a new set of points:

$$x_0^*, x_1^*, ..., x_{m+n+1}^* \tag{4.10}$$

where $0 \leq x_0^* \leq z_0 \leq x_1^* \leq z_1 \leq ... \leq z_{m+n} \leq x_{m+n+1}^* \leq 1$. This new set of points that will be used in the next iteration. RemezNet continues the iteration until a stopping criterion is met. Given the current reference points, RemezNet minimizes the maximum of residues:

$$\mathcal{L}_{MM} = \|f(x) - R(x)\|_\infty, \tag{4.11}$$

[C] Mean Squared Error(MSE):

Following general regression task, RemezNet applied mean squared error as another constraint:

$$\mathcal{L}_{MSD} = \sum_d^n \|f(x_d) - R(x_d)\|. \tag{4.12}$$

Therefore, the full objective is:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{EO} + \lambda_2 \mathcal{L}_{MM} + \lambda_3 \mathcal{L}_{MSD}, \tag{4.13}$$

where $\lambda_{\{1,2,3\}}$ controls the relative importance of these objectives. In the experiments, all λ are set to 1. Therefore, we aim to solve:

$$\phi^*, \psi^* = \arg\min \mathcal{L}\,. \tag{4.14}$$

[D] Padé and Alternating Sign Initialization(PASI):

Different from tradition method, the neural network often starts from a random position and try to approach the optimal configuration. It is widely recognized that neural network optimization is often stuck in a local minimum, so the initialization is critical to the convergence performance.

Solving the system of linear equations of Eq. 4.7 will fail when the square matrix is singular, and it is non-robust concerning rounding errors even when the matrix is nonsingular. To work with it robustly, the idea of Padé approximation is introduced to initialize model parameters. Padé approximation is a special case of rational approximation, and usually robust when functions contain poles, because of the use of rational functions. Padé approximants of order (m, n) exist if there exist two polynomials $P_m(x)$ and $Q_n(x)$ such that:

$$[i] f(x) - \frac{P_m(x)}{Q_n(x)} = \mathcal{O}(x^{m+n+1}) \tag{4.15}$$

$$[ii] \phi_0 = 1. \tag{4.16}$$

As this definition of Padé approximation states, the optimal solution normalize the bias of denominator to 1 (Eq. 4.16), which is our first initialization policy.

Second initialization constraint is inferred from equioscillation and the specific domain(i.e., [0,1]). Enforcing some reference points in the target domain, i.e., $0 \leq x_0 \leq x_1 \leq x_2 \leq ...x_{m-1} \leq 1$, the numerator becomes:

$$P_m(x) \tag{4.17}$$

$$= \prod_{i=0}^{m-1} (x - x_i) = (x - x_0)(x - x_1)...(x - x_{m-1}) \tag{4.18}$$

$$= x^m - x^{m-1} \sum_{i}^{m-2} \xi_i + x^{m-2} \sum_{i,j}^{m-2} x_i x_j - x^{m-3} \sum_{i,j,k}^{m-3} x_i x_j x_k + ... \tag{4.19}$$

$$= \sum_{i=0}^{m} (-1)^i x^{m-i} \underbrace{\sum_{i,j,...,k}^{m-i}}_{\text{i element(s)}} \overbrace{(x_i x_j ... x_k)}^{\text{i element(s)}}, \tag{4.20}$$

Algorithm 2: RemezNet

Input: a small set of sampled points: x_i, y_i

Output: a rational function with parameters: ψ_i and ϕ_i

Predefine m, n

Initialize parameters ψ and ϕ [D] ▷ Eq. 4.16, 4.21 and 4.22

Pick $m+n+2$ reference points $x_0, x_1, ..., x_{m+n+1}$, and prepare a power series of each x_i

$\mathcal{L}_{global} = 0$

//Outer loop iterates on different set of reference points

repeat

  //Inner loop iterates on different model parameters

  repeat

    compute *equioscillation losses* [A] ▷ Eq. 4.8

    compute *minimax losses* [B] ▷ Eq. 4.11

    compute *mean square losses* [C] ▷ Eq. 4.12

    compute total loss $\mathcal{L}$ ▷ Eq. 4.13

    compute gradient: $\nabla_{\psi,\phi}\mathcal{L}$

    update parameters: $\psi, \phi = \psi, \phi + \alpha\nabla_{\psi,\phi}\mathcal{L}$

  until $\mathcal{L}$ *converges*;

  $\Delta = \| \mathcal{L} - \mathcal{L}_{global} \|$.

  exchange reference points by choosing extreme values in each interval. ▷ Eq. 4.9 and 4.10.

  $\mathcal{L}_{global} = \mathcal{L}$.

until Δ *converges* $\mathcal{L}$ *is small enough*;

where $\sum_{i,j,...,k} x_i x_j ... x_k$ is the sum of all possible combinations, and each subscript, $i, j, ..., k$, belongs to the domain $[0, m-i] \subset \mathbb{Z}$. Therefore, the sign of each term in Eq. 4.20, $(-1)^i x^{m-i} \sum_{i,j,...,k}^{m-i} (x_i x_j ... x_k)$, is determined by $(-1)^i$, since the other terms are positive. Consequently, assigning terms with alternating signs is natural choice, i.e., {1,-1,1,-1,1,-1,...}, or

$$P_m(x) = 1 \cdot x^m - 1 \cdot x^{m-1} + 1 \cdot x^{m-2} + ... + (-1)^i \cdot x^{m-i} + ... \tag{4.21}$$

Similarly, denominator is initialized as:

$$Q_n(x) = 1 \cdot x^n - 1 \cdot x^{n-1} + 1 \cdot x^{n-2} + ... + (-1)^i \cdot x^{n-i} + ... \tag{4.22}$$

4.3.4 Algorithm Description

The Algorithm 2 first predefines desired degrees(line 1) and initializes coefficients of a rational function using **PASI**(line 2). Then, $m + n + 2$ reference points are sampled with equal intervals(line 3). The losses are calculated (line 4.8, 4.11, 4.12) given the current reference points, and sum of losses is computed with finely tuned weight parameters.(line 4.13). Following typical neural networks, model parameters are updated by back propagation(line 14). After convergence on current reference points, RemezNet exchanges reference points with a new set of points to seek lower loss(line 18). If the loss on current references converges, the algorithm stops. Otherwise, RemezNet continues to exchange the reference points and repeat the same learning process(from line 8 to 15).

4.4 Evaluation

This section elaborates evaluation with a detailed analysis of the behaviours of the proposed method. Specifically, subsection [I] introduces experiment settings, and subsection [II] evaluates the effectiveness of the proposed initialization policy. In subsection [III], we present intuitive results that show the proposed losses are critical to approximation performance. To demonstrate the effectiveness and efficiency of RemezNet, several challenging baselines are compared in subsection [IV]. In last subsection [V], the robustness of Remez and RemezNet is analyzed.

Two functions with discontinuities in [0, 1] are discussed: $|x - 0.5|$ and $sign(x - 0.5)$, since they are often considered to be challenging benchmark in literatures [139]. To keep the notation simple, we use $|x|$ and $sign(x)$ to represent the original functions. The degree of RemezNet in evaluation is set to 6 since rational function shows a theoretical advantage over polynomials when the degree is not smaller than 5 [103]. A higher degree means better capacity but incurs more computation cost, so the degree is set to be the minimum. 500 discrete

points are sampled for each function. Only a small number of points are used as input(each iteration needs $m + n + 2 = 14$ points, and often 2 or 3 iterations are sufficient); the other points are for the test. In the evaluations, we study the behaviors of different initializations and analyzes their difference quantitatively. We compare **PASI** against several popular initializations for deep neural networks: Kaiming Normal(**KNORM**)/Uniform(**KUNI**)[51], and Xavier Normal(**XNORM**)/Uniform(**XUNI**)[42]. To prove the superiority of the proposed model, we compare RemezNet against state of the art regression models including many that have analytical expression:

- Linear Regression(LR)
- Polynomial Regression(Polyfit) [131]
- Passive Aggressive Regression(PAR) [27]
- LASSO [137]
- Support Vector Regression(SVR) [129]. 3 kernels were applied: linear(L), polynomial(P) and RBF(R).
- Ridge Regression(RR) [105]
- Bayesian Ridge Regression(BR) [94],
- Automatic Relevance Determination(ARD) [138]
- Elastic Net(EN) [169]
- Orthogonal Matching Pursuit(OMP) [95]
- SGD Regression
- Huber Regression [56]

4.4.1 Experiment configuration

4.4.2 Initialization evaluation

This subsection study the behaviors of Padé normalization and Alternating Sign Initialization(ASI) respectively. Fig. 4.3 and 4.4 shows the convergence results by different initializations (**ASI**, **XNORM**, **XUNI**; **KNORM**, **KUNI** respectively) for $|x|$ and $sign(x)$. In each subfigure, upper one is without Padé normalization, and lower one is with Padé normalization. Overall, there is significant advantage of Padé normalization over that without Padé normalization. Specifically, Padé normalization improved **ASI** (error: 0.006 93), **XNORM** (38.361 10), **XUNI** (0.093 95), **KNORM** (0.043 95), **KUNI** (27.523 61) on $|x|$ by 98.77%(error:$8.489\,63 \times 10^{-5}$), 99.99%($8.954\,99 \times 10^{-5}$), 99.76%(0.00021), 99.76%(0.00010), 99.95%(0.01188) respectively. Note that **ASI** and **XNORM** performed much better than the others and at the same error level(10^{-5}). Similarly for $sign(x)$, Padé normalization improved **ASI** (0.462670), **XNORM** (10280.983650), **XUNI** (0.564945), **KNORM** (0.462945),

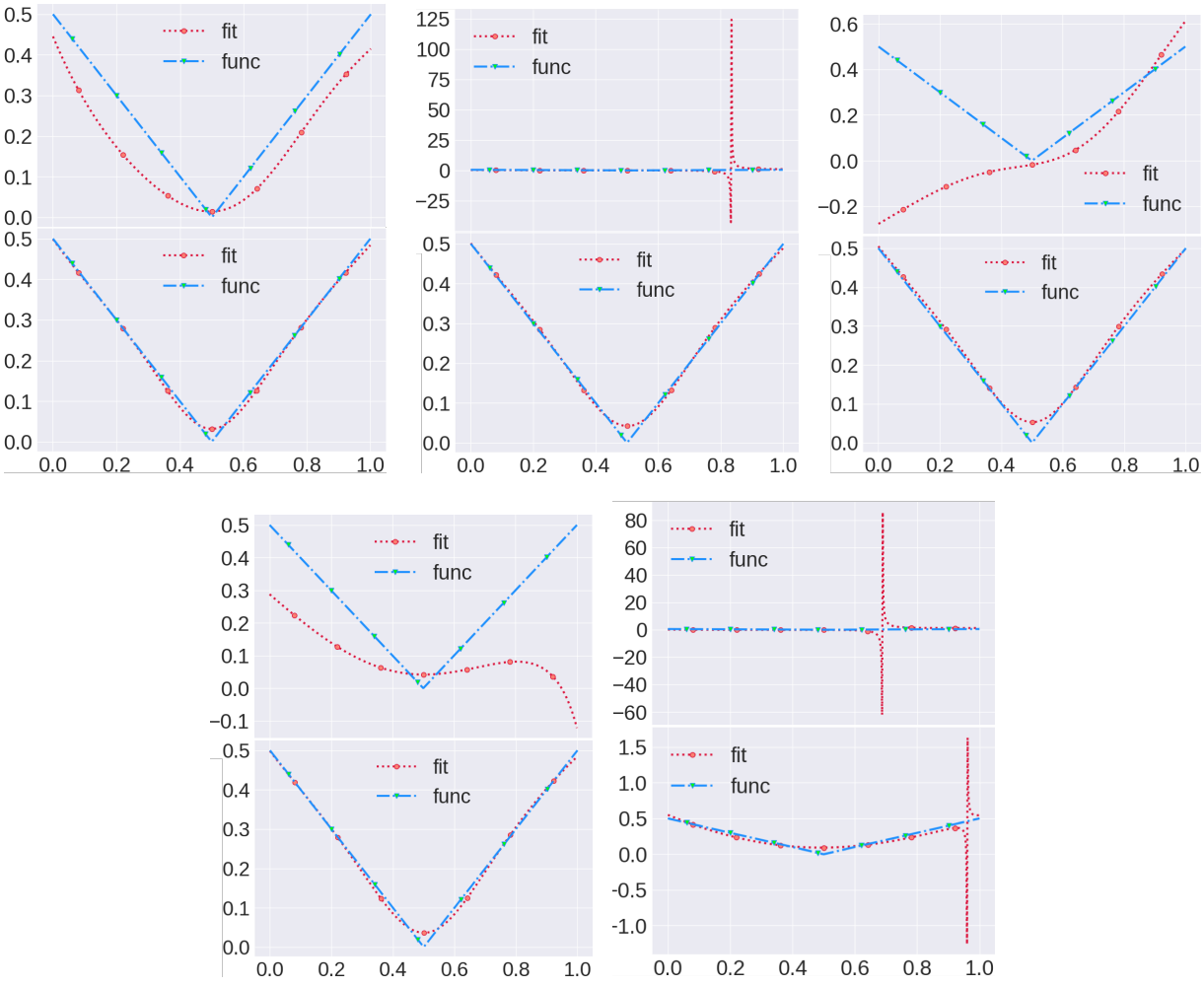


Figure 4.3: Approximation results by different initialization methods for $|x|$.

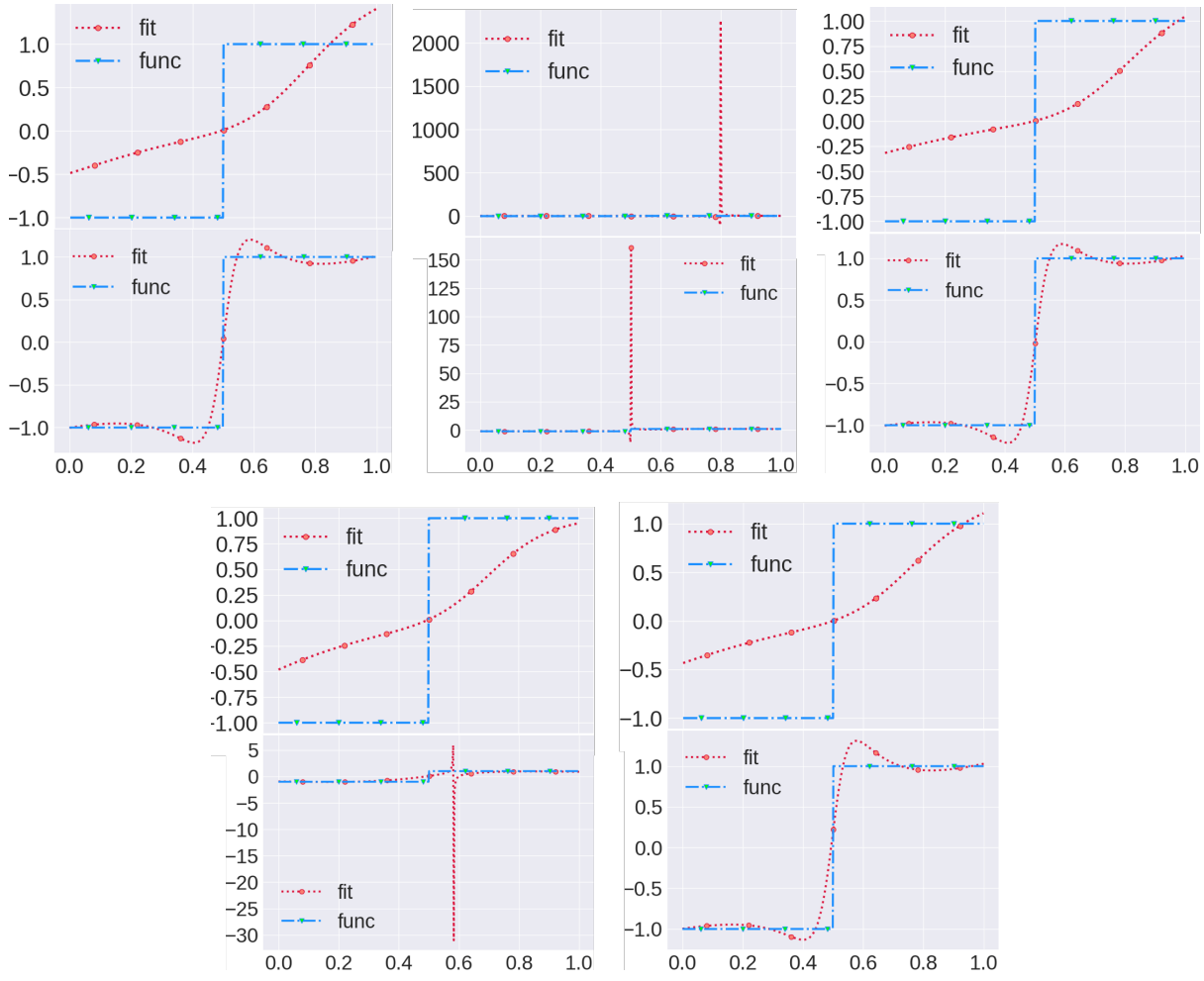


Figure 4.4: Approximation results by different initialization methods for $sign(x)$.

loss	$\|x\|$	$sign(x)$
MSE w/o PASI	3.524×10^{-2}	5.8346×10^{-1}
MSE	4.8×10^{-4}	3.370×10^{-2}
OSC	1.9×10^{-4}	5.020×10^{-2}
MM	1.2×10^{-4}	5.813×10^{-2}
MSE + OSC + MM	$8.489\,63 \times 10^{-5}$	3.203×10^{-2}

Table 4.2: Loss study: error on $|x|$ and $sign(x)$.

KUNI (0.490364) by 93.07% (0.032034), 99.99% (0.032965), 94.47% (0.031188), 92.18% (0.036182), 93.40% (0.032323). In $sign(x)$ case, the Padé normalization reduced the errors of approximations by all the initializations to the same level. Note that the error of **ASI** (0.032034) only larger than that of **XUNI** (0.031188) by a very small margin.

By these experiments, it concludes that Padé normalization makes a significant contribution to approximation results, and **ASI** performed robustly across different functions. Therefore, it is reasonable for their combination (lower part of each subfigure in the first column) to outperform the others.

4.4.3 Loss evaluation

This subsection studies the performance of the three losses separately, showing the necessity of the combination of them. RemezNet is tested under every single loss, and under the sum of these losses respectively. Each loss was evaluated on the same reference points for both functions. This set of points are initialized randomly with an equal interval. In Table 4.2, the combination of three losses outperformed any single loss by a large margin. For example, comparing the approximation errors on $|x|$, their combination ($8.489\,63 \times 10^{-5}$) outperformed **MM** (0.00012) by 29.25%, **MSE** (0.00048) by 82.31%, **OSC** (0.00019) by 55.31%. For $sign(x)$, their combination is still the best. The setting **MSE w/o PASI** means that a neural network with MSE loss only and it does not apply Padd́ normalization. This general network caused the largest errors in both tasks, which proves that neural networks with general constraint do not work in this problem. We can conclude that our proposed objectives are critical to the approximation performance.

4.4.4 Function Approximation

Fig. 4.5 shows that RemezNet(black) outperformed all the baselines. Quantitatively, Table 4.3 lists comparison of their mean square error. For $|x|$, SVR(RBF) and PolyFit outperformed all the other baselines. RemezNet further improved their errors by 98.41% and 93.15%. Similarly, SVR(RBF) and PolyFit achieved the best performance of the baselines on $sign(x)$,

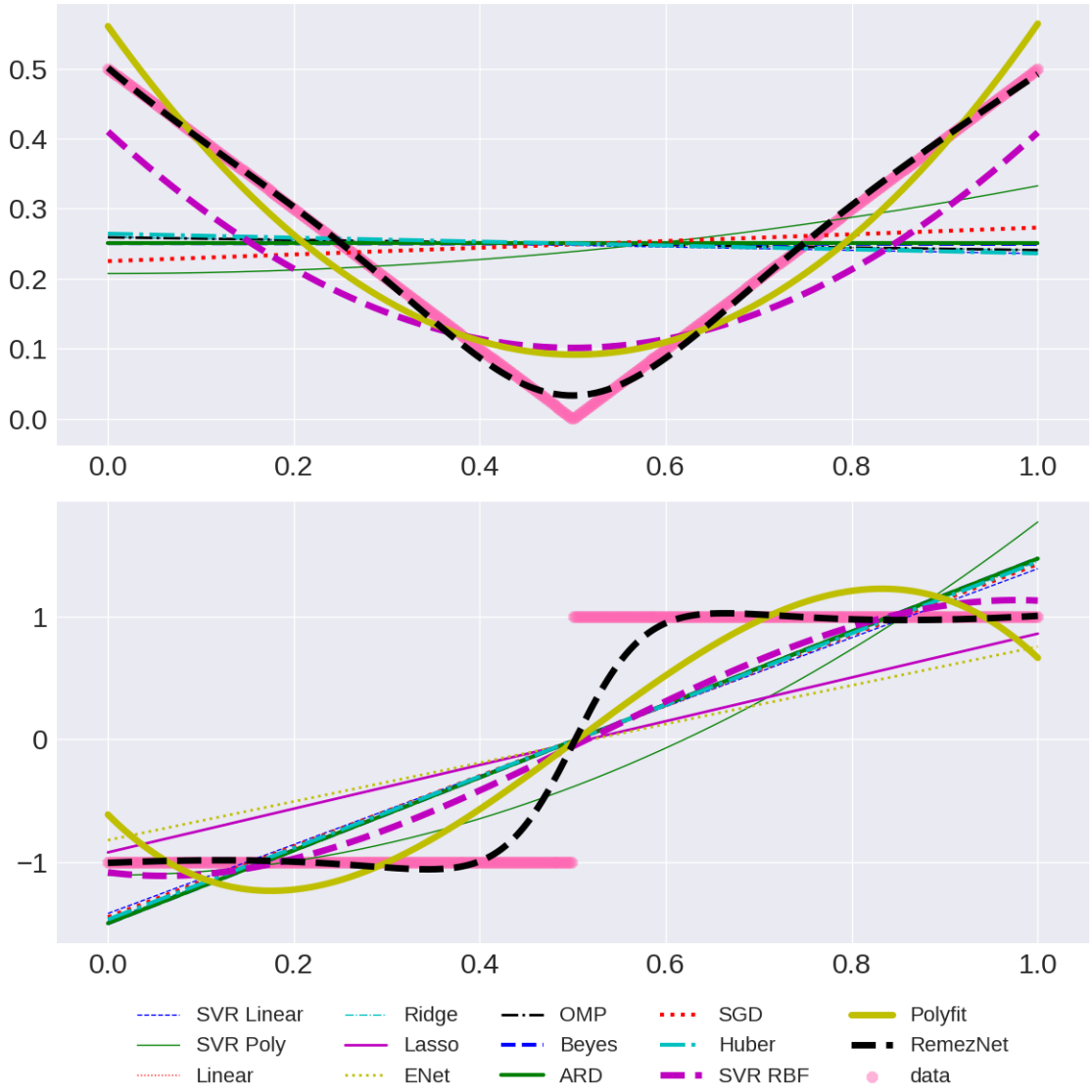


Figure 4.5: Approximation comparison on $|x|$ and $sign(x)$.

our model is still better than them by 73.94% and 65.05%. Fig. 4.6 shows the comparison

Method	$\|x\|$	$sign(x)$
SVR RBF	0.0056±0.0000	0.1888±0.0000
SVR Linear	0.0210±0.0000	0.2520±0.0000
SVR Poly	0.0198±0.0000	0.3173±0.0000
LR	0.0209±0.0000	0.2500±0.0000
RR	0.0209±0.0000	0.2501±0.0000
LASSO	0.0209±0.0000	0.3705±0.0000
EN	0.0209±0.0000	0.4168±0.0000
OMP	0.0209±0.0000	0.2500±0.0000
BR	0.0209±0.0000	0.2500±0.0000
ARD	0.0209±0.0000	0.2500±0.0000
SGD	0.0212±0.0000	0.2511±0.0003
PAR	0.2007±0.1118	0.6193±0.2730
Huber	0.0209±0.0000	0.2502±0.0000
PolyFit	0.0013±0.0000	0.1408±0.0000
RemezNet	8.8986×10^{-5} $\pm 1.3536 \times 10^{-5}$	**0.0492±0.0018**

Table 4.3: Baselines: error comparison on $|x|$ and $sign(x)$

of running time. Most baseline methods are efficient such as LR, RR, LASSO, EN, OMP, BR, PAR. They finish computing within 0.01 second on both cases. SGD, PolyFit, and Huber only require 0.02 and 0.01 respectively. SVR group performed slower, completing the calculation within 10 seconds. ARD needs around 15 seconds and 1.4 seconds separately, which is the slowest baseline. RemezNet took 19 seconds for $sign(x)$, and 4.033 seconds on $|x|$, which is ever slower than ARD. However, efficiency is not the advantage of RemezNet, but it shows that our algorithm can run reasonably fast.

4.4.5 Remez and RemezNet

RemezNet is designed to avoid infeasibility of traditional Remez. Table 4.4 shows that RemezNet can always give a solution, while Remez is very sensitive to the degree number and it's difficult to find a solution given degree. Using the same test in Table 4.3, the result of Remez for $|x|$ are $1.956\,296\,282\,055\,538\,3 \times 10^{-6}$(degree=6) and $1.300\,079\,294\,436\,022 \times 10^{-7}$(degree=8), which is better than that of RemezNet (8.8986×10^{-5}). The results of Remez for $sign(x)$ are $0.041\,031\,774\,788\,093\,89$(degree=6) and $0.020\,770\,376\,978\,777\,08$(degree=8), which are slightly better than RemezNet. In a word, RemezNet approximates the functions with reasonably small error compared with traditional Remez, and prevent the model from infeasibility issue.

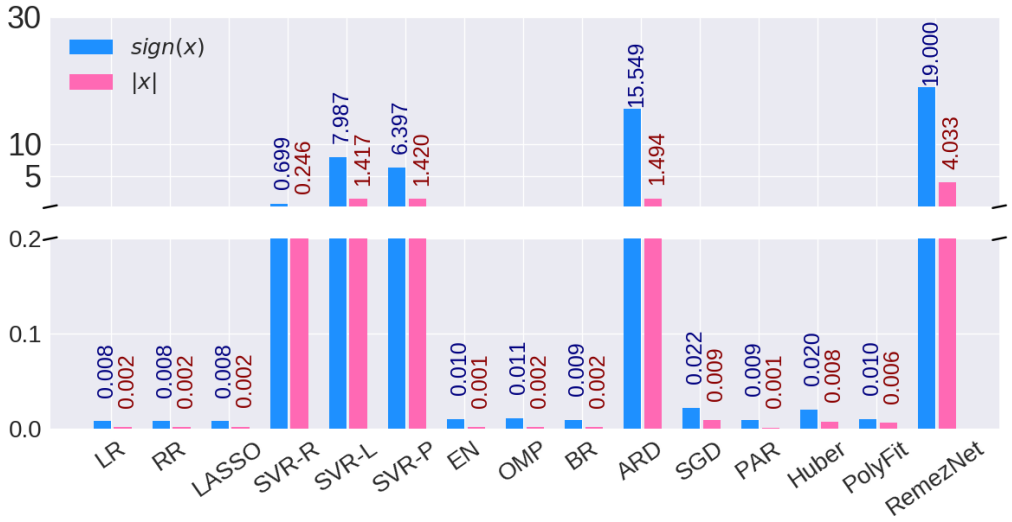


Figure 4.6: Comparison of average running time in seconds.

z		$\lvert x-z\rvert$				$sign(x-z)$			
		6	7	8	9	6	7	8	9
0.5	RemezNet	√	√	√	√	√	√	√	√
	Remez	√	×	√	×	√	×	√	×
0.3	RemezNet	√	√	√	√	√	√	√	√
	Remez	×	×	×	×	×	×	×	×
0.8	RemezNet	√	√	√	√	√	√	√	√
	Remez	×	×	×	×	×	×	×	×

Table 4.4: Feasibility comparison. √ means that solution is available, while × indicates that it's not solvable.

4.5 Conclusion

In this paper, we have introduced RemezNet for analytical function approximation. RemezNet jointly utilizes the advantages of the Remez algorithm and deep neural networks to estimate the parameters of the closed form. Inspired by the basic idea behind Remez, the proposed method employs the equioscillation theorem and the minimax characterization of best rational approximation. To solve the non-robustness, RemezNet initializes parameters constrained by Padé normalization and alternating sign. Extensive experiments suggest that RemezNet is capable of approximating functions accurately and robustly.

Chapter 5

Circuit based Deobfuscation Runtime Estimation

This chapter proposes the first graph neural network for circuit deobfuscation problem. By adapting the exclusive features of the circuit and its obfuscation problem, ICNet is designed to fit this scenario and automatically learns the relationship between deobfuscation time and circuit topology.

5.1 Introduction

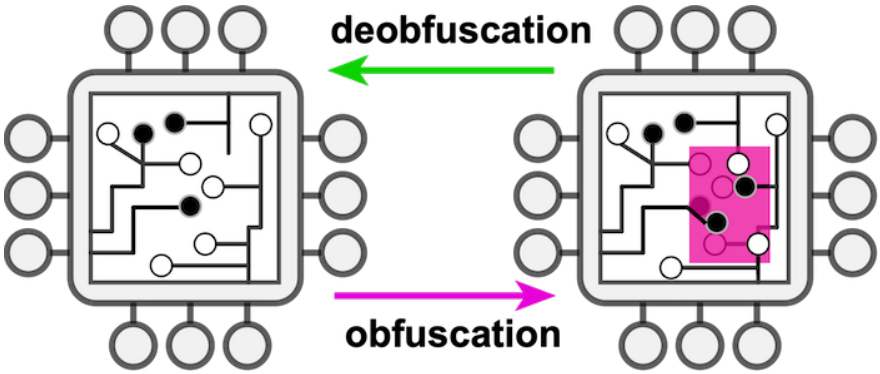


Figure 5.1: Illustration of obfuscation and deobfuscation

Over the past years, many of the leading semiconductor companies have become fabless because the increasing costs and complexity confine them not to design, test, fabricate, and package ICs. The expense of making a new semiconductor fab was estimated to be the $5.0 billion [46, 159]. Due to the high cost of building and maintenance, many companies have been fablessin recent years. The considerable high capital costs on semiconductor manufacturing

motivate most companies to outsource their designed integrated circuits (ICs) to the contract foundries for fabrication. Despite the reduced cost and other benefits, this trend has led to ever-increasing security risks such as concerns of risks on IC counterfeiting, piracy and unauthorized overproduction by the contract foundries [46, 132]. The overall financial risk caused by such counterfeit and unauthorized ICs was estimated to be over $169 billion per year [96]. The major threats from the attackers arise from reverse engineering an IC by fully identifying its functionality by stripping it layer-by-layer and extracting the unveiling gate-level netlist. To prevent such reverse engineering, IC *obfuscation* techniques have been extensively researched in recent years [158]. The general idea is to obfuscate some gates in an IC so that their gate type cannot be determined by reverse engineering optically, yet they preserve the functionality same as the original gates. As shown in Fig. (5.1), obfuscation is a process which selects a part of the circuit (in pink) and modifies the structure of them. The functionality of these encrypted gates can be retrieved only if correct keys are provided at the input.

Such techniques were highly effective until very recent progress of the attacking techniques based on logical attackers were invented and widely applied [37]. This is due to that there are limited types of gates (e.g., AND, OR, XOR) in IC, so the attackers can just brute force all the possible combinations of types for all obfuscated gates to find out the one that functions identically to the targeted IC. However, brute force is usually prohibitively expensive. More recently, efficient methods such as satisfiability (SAT) based attacks have been proposed which have attracted enormous attention [86].

The runtime of the SAT attack to decrypt the IC mostly depends on the complexity of the obfuscated IC, which can vary from milliseconds to days or years. Moreover, gates to obfuscate come at a heavy cost in finance, power, and space; such trade-off forces us to search for optimal layout instead of purely increasing their quantity. Therefore, the best set of gates for being obfuscated maximizes the runtime for deobfuscating. However, until now it is still generally based on human heuristics or experience, which is seriously arbitrary and suboptimal [64]. This is because it is unable to "try and error" all the different ways of obfuscation, as there are millions of combinations to try and the runtime for each try (i.e., to run the attacker) can be days, weeks, or years.

To address this issue, this paper focuses on efficient and scalable ways to estimate the runtime for an attacker to deobfuscate an obfuscated IC. This research topic is vastly under-explored because of its significant challenges: **1) Difficulty in characterizing the hidden and sophisticated algorithmic mechanism of attackers.** Over the recent years, a large number of deobfuscation methods have been proposed with various techniques [64]. In order to practically beat the attackers, methods with sophisticated theories, rules, and heuristics have been proposed and adopted. The behavior of such highly-nonlinear and strongly-coupling systems is prohibitive for conventional simple models (e.g., linear regression or support vector machine) to characterize. **2) Difficulty in extracting determinant features from discrete and dynamic graph-structured ICs.** The inputs of the runtime estimation problem is the ICs with selected gates obfuscated. Conventional feature extraction methods are not intuitive to be

applied to such type of varying-structured data without significant information loss. Hence, it is highly challenging to intactly formulate and seamlessly integrate them as mathematical forms that can be input to conventional machine learning models. **3) Lack of approach that extracts interpretable patterns.** The key to the defense against deobfuscation is to obtain interpretable rules so that the engineer can understand and directly apply them for obfuscation task. This requires the model to have the capacity of integrating domain based knowledge and generate explicable rules.

This work addresses all the above challenges and proposes the first generic framework for deobfuscation runtime prediction, based on graph neural networks. By concretely formulating ICs and the camouflaged gates as multi-attributed graphs, this work innovatively leverages and extends the graph deep learning methods to instantiate a graph regressor. Such end-to-end deep graph regressor can characterize the underlying and sophisticated cognitive process of the attacker for deobfuscating the ICs. It can also automatically extract the discriminative patterns that are determinants to the estimation of the deobfuscation runtime to achieve accurate runtime prediction. The major contributions of this paper are:

- **Proposing a new framework, ICNet, for deobfuscation runtime estimation based on graph deep learning**. We propose the first graph neural network for the circuit, where it is directly modeled using the gate connectivity. After learning the representation of the circuit, a neural network is trained to the relationship between its representation and the runtime.
- **Designing an interpretable component for extracting domain rules.** Utilizing the attention mechanism, the importance of features regarding runtime is derived during the learning process. Features are evaluated by importance ranking, and the only important features are kept to boost the efficiency with little accuracy loss.
- **Conducting systematical experimental evaluations and analyses on real-world datasets.** Standard benchmark with several circuits is used to evaluate the proposed method and competitive baselines. The result shows constant superiority of the proposed ICNet beyond the baselines.

In this chapter, Section 6.2 reviews existing work. Section 6.4 elaborates a graph deep learning model for deobfuscation task. In Section 6.5, evaluation on real-world benchmarks is present. This paper summarizes the important findings in Section 6.6.

5.2 Background and Related Work

This section discussed the logic obfuscation and SAT attacks followed by graph convolutional networks as background.

5.2.1 Logic Obfuscation and SAT Attacks

Logic obfuscation often referred to as logic locking [157] is a hardware security solution that facilitates to hide the IP using key-programmable logic gates. The activation of the obfuscated IP is accomplished in a trusted regime before releasing the product into the market, thereby reducing the probability to obtain the secret configuration keys by the attacker. During the activation phase, the correct key is applied to these key-programmable gates to recover the correct functionality of the IC/IP. Although obfuscation schemes try to minimize the probability of determining the correct key by an attacker and avoid making pirated and illegal copies, introducing SAT attack shows that these schemes can be broken [133].

Different SAT-hard schemes such as [151, 156] are proposed Furthermore, new obfuscation schemes that focus on non-Boolean Behavior of circuits [150], that are not convertible to an SAT circuit is proposed for SAT resilience. Some of such defenses include adding cycles into the design [121]. By adding cycles into the design may cause that the SAT attack gets stuck in the infinite loop. However, advanced SAT-based attacks such as cycSAT [164] can extract the correct key despite employing such defenses.

To ensure that the proposed defense ensures robustness against SAT attacks, the defenders need to run the rigorous simulations which could range from few minutes up to a few days. The work in [123] utilizes neural network with single-bit supervision to predict whether a given circuit in Conjunctive Normal Form (CNF) can be decrypted or not. However, this is limited to determining for few kinds of SAT-solvers, but cannot be applied to SAT-hard solutions such as SMT-SAT [161], a superset of SAT attacks. However, with proposed GCN based predictor, the defender can determine the deobfuscation time in a single run of GCN, which consumes a few seconds.

5.2.2 Graph Neural Networks

Spectral graph theory is the study of the properties of a graph in relationship to the spectral properties of the graph matrices. Many graphs and convolution methods have been proposed recently. The spectral convolution methods [32, 65] are the mainstream algorithms designed as the graph convolution. Their theory is based on the graph spectral analysis [125]. The polynomial approximation is firstly proposed by [50]. Inspired by this, graph convolutional neural networks (GCNNs) ([32]) is a successful attempt at generalizing the powerful convolutional neural networks (CNNs) in dealing with Euclidean data to modeling graph-structured data. Kipf and Welling proposed a simplified type of GCNNs[65], called graph convolutional networks (GCNs). The GCN model naturally integrates the connectivity patterns and attributes of graph-structured data and outperforms many state-of-the-art methods significantly. With rational function, GCN can model non-smooth signal in spectral domain[21].

5.3 Methodology

This section introduces the problem setting, and presents the proposed deobfuscation time prediction.

5.3.1 Problem Setup

First, circuit is modeled as a graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, where $\mathcal{V}$ is a set of n vertexes, $\mathcal{E}$ represents edges and $\mathcal{A} = [a_{ij}] \in \{0,1\}^{n \times n}$ is an unweighted adjacency matrix. A signal x defined on the nodes may be regarded as a vector $x \in \mathbb{R}^F$. Combinatorial graph Laplacian is defined as $\mathbf{L} = \mathcal{D} - \mathcal{A} \in \mathbb{R}^{n \times n}$ where $\mathcal{D}$ is degree matrix.

Accordingly, we formulate the estimation of running time on IC as a regression task. Specifically, given the circuit (denoted as $\mathcal{G}$) of the IC before obfuscation, and x is attributes attached to gates. The goal of this work is to predict the runtime T_i by a runtime prediction function $f : (\mathcal{G}, x_i) \to T_i$, where T_i is typically a non-deterministic value.

5.3.2 ICNet

GCN is neural networks that extract vector embeddings for vertex based on averaging properties value of neighbors. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where V and E are nodes and edges respectively. The matrix $X \in \mathbb{R}^{n \times F}$ denotes attribute of all nodes, where F is the dimension of the attribute vector. Formally, GCN performs graph convolution σ as:

$$Z^{j+1} = \sigma(\tilde{A}XW_j) \in \mathbb{R}^{n \times H^{j+1}}, \tag{5.1}$$

where $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is normalized graph matrix, and W_j is weight matrix. j indicates the index of GCN iteration.

ICNet is a neural network that is based on graph convolution operator. As shown in Figure (5.2), ICNet encodes the obfuscated circuit on the left hand into two components: (1) **graph structure** $\mathcal{G}$: Complete set of local connection is often used to represent the graph structure. Typically, a graph Laplacian is employed, since it contains gate-wise connection. (2) **gate attributes** x : gate-level information can be encoded as a numerical vector as an input feature. Such information could include gate type, or whether it is obfuscated. However, the convolutional operator of GCN is not suitable for the circuit, which is due to two technical issues:

- **graph Laplacian operator L:** the graph Laplacian will make the graph convolutional operator behavior as label propagation, i.e., the attributes of each gate are similar to its neighbors. This is called *smoothness assumption*. This does not fit the domain

knowledge: gate type or encryption location of each gate does not determine its neighbors' type or mask.

- **attribute aggregation:** After applying graph convolutional operator, new features $z \in \mathbb{R}^{n \times Z}$ are generated. However, there is a clear methodology for aggregating multiple attributes, except employing *sum* or *mean* function. However, *sum* or *mean* further strength the smoothness assumption by averaging neighbors and hereby violate domain knowledge.

Therefore, we cope with the technical issues above by proposing several policies: replacing graph Laplacian $\mathbf{L}$ with adjacency matrix $\mathbf{A}$, and proposing attention based method for attribute-wise and dimension-wise aggregation.

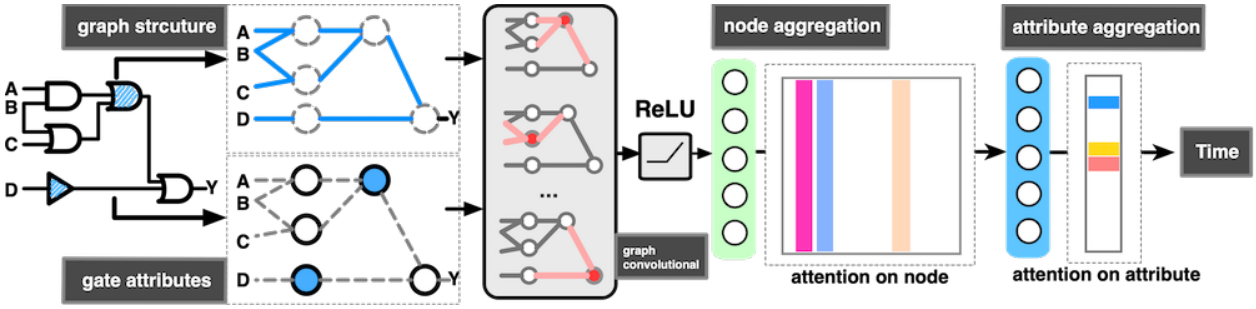


Figure 5.2: ICNet structure: the model conducts graph convolutional operation to fuse information from graph structure and gate attributes. Then attention neural networks are performed for the attribute and gate aggregation. Last few layers are fully connected to predict the runtime.

Graph representation $\mathcal{G} \rightarrow \mathcal{A}$

Graph Laplacian $\mathbf{L}$ weighs each node as N_i using degree matrix $\mathcal{D}$ (i is the index of the row in graph Laplacian or degree matrix), and weighs the sum of weights of its neighbors as $-N_i$ due to $-\mathcal{A}$. Consequently, they are canceled out when gate representation are aggregated using sum, especially their representation is close. Then the model can hardly learn the relationship between their $sum(\cdot)/mean(\cdot)$ of residues and labeled time. This can be easily solved by replacing graph representation $\mathcal{G}$ with adjacency matrix $\mathcal{A}$, so that the proposed model only consider neighbors as input when considering on each gate. This representation can avoid smoothness assumption which is not compatible under IC scenario.

Attention based aggregation

Generally, *sum* or *mean* function is a typical methods for aggregating node attribute into lower dimensional vector. This treats voting from each gate equally, which is not fit in theory.

For example, the encrypted gate should be weighed higher, since it impose more difficulty on obfuscation task; gate types also have a significant impact on runtime. Therefore, a more flexible way is build to neural network to automatically learn attribute aggregation. To fit whole-graph level regression task, the proposed method designs two aggregation neural components based on attention mechanism for node-level and attribute-level. Formally, attribute based attention is calculated as:

$$\begin{aligned} a_i &= \frac{exp(e_i)}{\sum_i exp(e_i)}, \\ c_i &= \sum_i a_i attr_i, \end{aligned} \tag{5.2}$$

where $attr_i$ represents ith attribute, and $e_i = \theta_{attr_i}$. This attention shows which attribute contributes more to the obfuscation time. Similarly, node-wise attention is utilized for node-level aggregation by setting $e_i = \theta_{node_i}$ in (5.2). To make this iattention interpretable, W in Eq. (5.1) is removed:

$$attr_i = \sigma(\mathcal{A}X)_i. \tag{5.3}$$

Complete workflow of ICNet is shown in Fig. (5.2). ICNet is actually a generalization of a state-of-the-art model:

Lemma 5.3.1. *ICNet is a generalization of MoNet[102], while MoNet is a special case of ICNet when setting the attribute aggregation to sum function.*

Proof. We denote x a vertex of a graph and consider $y \in \mathcal{N}(x)$ in the neighborhood of x. MoNet defines a pseudo-coordinates $\mathbf{u}(x, y)$. Therefore, the patch operator can be written in a general form:

$$(f * g)(x) = \sum_{j=1}^{J} g_j D_j(x) f = \sum_{j=1}^{J} g_j \sum_{y \in \mathcal{N}(x)} w_j(\mathbf{u}(x, y)) f(y), \tag{5.4}$$

where $j \in [1, J]$ is the index of frequency component, or the index of vertex. Rewrite Eq. (5.4) in matrix form, we have:

$$(f * g)(x) = \mathbf{GWF}, \tag{5.5}$$

where $\mathbf{G} = \{g_1, g_2, ..., g_J\} \mathbb{R}^{n \times n}$ is frequency component of graph Laplacian or adjacency, $\mathbf{W} = \{w_{1,1}, w_{1,2}, ..., w_{J,J}\} \mathbb{R}^{n \times n}$ represents weight matrix and $\mathbf{F} = \{f_1, f_2, ..., f_J\} \mathbb{R}^{n \times F}$ denotes attribute matrix. By setting $\mathbf{W} = \mathbf{I}$, Eq. (5.5) becomes GCN[65]. $\mathbf{W}$ can be treated as an implementation of attention on nodes. Adding attributes aggregation for Eq. (5.5) will turn it into ICNet:

$$(f * g)(x) = \mathbf{GWF\Theta_{attr}}. \tag{5.6}$$

Therefore, MoNet is a special case of ICNet when setting $\Theta_{attr} = \mathbf{I}$ □

Algorithm 3: ICNet

Input: A circuit graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, gate attribute set: $x_j(i)$, i $\in 1, 2, ..., |\mathcal{V}|$ for each encryption instance D_j, the real runtime Y_j for instance D_j

Output: a neural network function with parameters θ

1 // data preparing
2 calculate $\mathcal{A}$ which is the adjacency matrix of $\mathcal{G}$
3 split instances D into training/testing set D_{train}/D_{test}
4 // update ICNet
5 initialize θ with Gaussian distribution, attention parameters are initialized using uniform distribution.
6 **repeat**
7 perform graph convolution on D_{train} ▷ Eq. (5.3)
8 aggregate nodes with attention model ▷ Eq. (5.2)
9 aggregate attributes with attention model ▷ Eq. (5.2)
10 calculate predicted runtime $T_i = MLP(c_i)$
11 calculate loss function $\mathcal{L} = \|Y - T\|$
12 compute derivatives to update parameters: $\theta \leftarrow \theta + \beta \nabla_\theta \mathcal{L}$, where β is learning rate
13 **until** θ *convergence*;
14 predict runtime of D_{test} instances

5.3.3 Algorithm Description

Algorithm 1 first prepare graph adjacency as circuit connection representation(line 2). To fit the machine learning schema, the whole dataset is split into training and testing dataset. Each dataset is then split into small batch size to improve learning efficiency(line 3). ICNet training is an iterative process which updates the model until the residues are small enough or converged(line 6-13). First, the model parameters are initialized by Gaussian distribution and uniform distribution for attention component. In each iteration, a batch of the training set is selected randomly. By equation (5.2), the model computes the aggregation of nodes and attributes (line 8-9). Then it is integrated with fully connected layers to predict runtime(line 10). Following update schema of normal neural networks, ICNet calculates the residues between real runtime and prediction(line 11), and then update parameters by the derivatives regarding the parameters themselves with learning rate(line 12). After convergence, ICNet evaluates testing data (line 13).

5.4 Evaluation

This section elaborates evaluation of the proposed method ICNet with competitive graph deep learning methods including: GCN[65] and ChebNet[32]. We also compare against several

stat-of-the-art regression models[1]:

- Linear Regression(LR)
- LASSO
- Epsilon-Support Vector Regression(SVR).
- Ridge Regression(RR)
- Elastic Net(EN)
- Orthogonal Matching Pursuit(OMP)
- SGD Regression
- Least Angle Regression(LARS)
- Theil-Sen Estimators(Theil)

These regression models does not model graph using Laplacian or adjacency matrix, since they can only accept attribute vector. Therefore, the input are encoded as mean or sum on concatenation of Laplacian or adjacency matrix and gate attributes.

5.4.1 Data Preprocessing

We evaluate ICNet[2] on ISCA-85 benchmarks for one replacement policy and SAT solver [133] that employs lingering solver. However, this can be applied to any of the circuits as well as replacement policies, as the neural graph networks learn the patterns and are not confined to any circuit or replacement policy or SAT solver. The datasets are obtained by running SAT algorithm [133, 132] on real-world ISCA-85 benchmarks: First, we take one circuit and select a random gate and replace it with LUT of fixed size (LUT size 4 in current work). To deobfuscate, we implement SAT attack algorithm [133, 132] with the obfuscated circuit netlist as input. We monitor the time that sat takes to decode the key, which is the deobfuscation time. The proposed model is evaluated on two datasets:

- **Dataset 1**: the total number of the encryption location ranges from 1 to 350, which is for testing sensitivity to the number of encrypted quantity of gates.
- **Dataset 2**: the total number of the encryption location ranges from 1 to 3, this is for testing if the model can handle very small value.

For graph deep learning methods, graph is represented using Laplacian matrix or adjacency matrix, while for general regression baselines, the graph Laplacian or adjacency matrix is

[1]https://scikit-learn.org/stable/modules/linear_model.html
[2]Data and codes will available at [hidden for double blind]

summed or averaged across gates. Though the evaluations showed here are mere proof-of-concept of how powerful the proposed GCN based deobfuscation runtime prediction is, it can be applied to an SAT-hardening solution utilizing any replacement policy, LUT size and other SAT parameters, by retraining GCN.

5.4.2 Experiment configuration

The attributes of gate used in experiments include

- **gate mask**: if the gate is encrypted, the value is set to 1, otherwise 0.
- **gate type**: {AND, NOR, NOT, NAND, OR, XOR}.

For graph deep learning model(ChebNet and ICNet), the graph structure is represented using graph Laplacian matrix or adjacency matrix. These model employ ADAM [72] optimizer and will stop learning when the learning loss is converged. The implementation of our model will be available online. All the baselines and the proposed model are evaluated on two different attribute sets, since gate type is useful or not is unknown. Two attribute sets are evaluated:

- **Location**: Only the location of encrypted gate is included
- **All attribute**: Both **Location** and gate type is included.

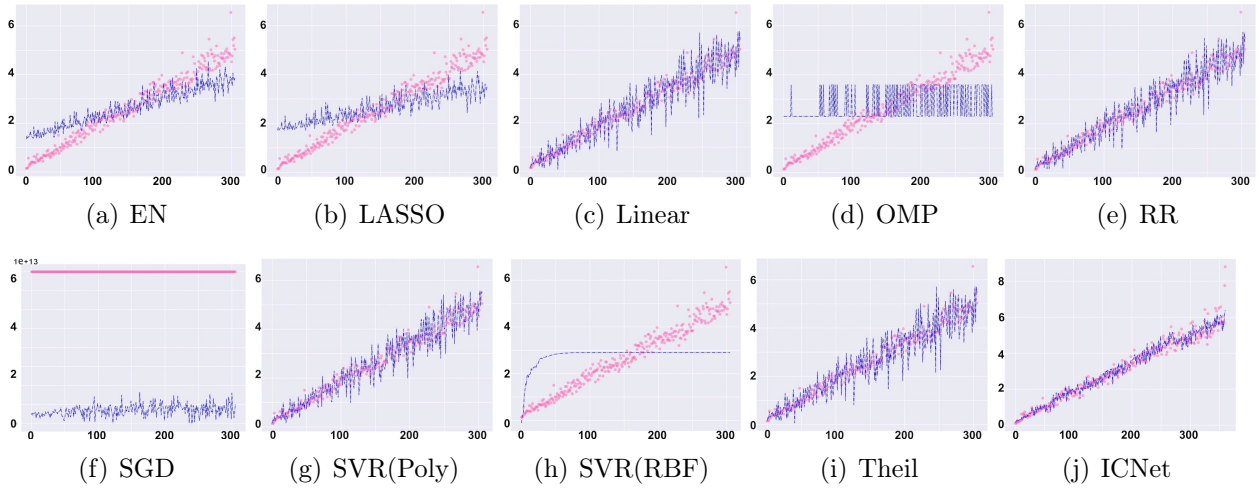
For node aggregation, we apply *sum* and *mean* since they are popular. Deep learning model can have another node aggregation method, i.e., learning by a neural network automatically. Therefore, in the results, $ChebNet-NN$ and $ICNet-NN$ denote the automatic version. It is expected that a deep neural network can learn an optimal aggregation which is not worse than our assumption, i.e., the sum or mean.

5.4.3 Regression Results

In the dataset 1 experiment(Table 5.1, all methods achieved acceptable mean square error except SGD (sum) which did not learn a reasonable model to predict the runtime, since the value is very large (at e+25/+26 scale). Most regression methods are sensitive to the aggregation method. For example, only using location attribute, MSE of RR is 0.2319 when using sum, but it got 2.1508 when using the mean function. Sensitive models include SVR, LASSO, and EN. The best of the regression baselines is LR and Theil, which achieved around MSE of 0.22. On the other hand, graph deep learning model ChebNet is slightly better than the best regression model. However, ChebNet is not stable and sensitive to the aggregation method and attribute set, since it may yield a very large error. Our model, ICNet, is stable to the attribute and aggregation setting and outperformed all the other methods, i,e, 0.11001 of

	Location		All attributes	
Method	Sum	Mean	Sum	Mean
SVR RBF	1.6791	0.6784	1.6675	0.6739
SVR Poly	0.1913	2.1890	0.1696	2.2091
SGD	2.1450e+25	2.1823	1.0430e+26	2.2072
LR	0.2839	0.2284	0.2449	0.2253
RR	0.2309	2.1508	0.2058	2.1738
LASSO	0.9213	2.1843	1.0127	2.2083
EN	0.5763	2.1843	0.6409	2.2083
OMP	1.8182	1.9192	1.8651	2.0337
LARS	1.9968	2.1277	2.0434	2.1833
Theil	0.2948	0.2238	0.2385	0.2277
ChebNet	0.1484	8.8370e+33	0.1761	0.1760
ChebNet-NN	0.17858		3.8549e+27	
GCN	0.3364	0.4149	0.2496	0.3290
GCN-NN	0.1811		0.1606	
ICNet	0.1534	0.1256	0.2390	0.1902
ICNet-NN	**0.0843**		**0.1367**	

Table 5.1: Regression Performance (Mean Square Error) on Dataset 1



(a) EN (b) LASSO (c) Linear (d) OMP (e) RR

(f) SGD (g) SVR(Poly) (h) SVR(RBF) (i) Theil (j) ICNet

Figure 5.3: Illustrative comparison between predictions and real values: Pink dot are real runtime, blue lines are the predictions. x-axis is data index in testing data while y-axis is runtime value. ICNet fits the real data with least variance than the others.

MSE. Note that ICNet-NN is better than ICNet with the sum or mean, which demonstrates that there exists a better aggregation method, and a deep neural network can learn this function automatically. Note that ICNet is always better than GCN under any settings,

	Location		All attribute	
Method	Sum	Mean	Sum	Mean
SVR RBF	0.0051	0.0048	0.0050	0.0051
SVR Poly	0.0048	0.0048	0.0048	0.0051
SGD	7.6301e+25	0.0045	2.0675e+26	0.0049
LR	6.9063ee+23	4.6521e+20	7.2916e+25	5.8600e+23
RR	0.0070	0.0045	0.0065	0.0049
LASSO	0.0047	0.0045	0.0046	0.0049
EN	0.0047	0.0045	0.0046	0.0049
OMP	0.0047	0.0045	0.0045	0.0049
PAR	0.0054	0.1918	0.0051	0.3143
LARS	0.0047	0.0045	0.0046	0.0049
Theil	N/A	N/A	N/A	N/A
ChebNet	0.0047	0.0045	4.3570e+28	0.0048
ChebNet-NN	**0.0043**		0.0047	
GCN	0.0061	0.0046	0.0048	0.0050
GCN-NN	0.0050		0.1606	
ICNet	0.0049	0.0047	**0.0040**	0.0043
ICNet-NN	0.0051		0.0048	

Table 5.2: Regression Performance (Mean Square Error) on Dataset 2

which shows that our improvement works on circuit scenario.

While in dataset 2, it is more challenging, since all the runtime is small and the model has to be very precise to achieve low MSE. All methods at almost the same level of MSE. Once again, some of the regression models are not stable such as SGD and LR. Graph deep learning method includes ChebNet and ICNet still at the best error level. ChebNet can achieve the best level but sensitive to the settings, while ICNet is insensitive to all configuration. ICNet-NN is still the best method, and it outperformed its mean and sum version. Moreover, ICNet is more stable than GCN and ChebNet.

Prediction behavior analysis

Since there is little difference in dataset 2, we choose several competitive baselines in dataset one experiments. Several baselines performed very badly such as OMP and SGD which only output values around a constant level. SVR(RBF) is also bad and yield constant value when the real runtime is larger than a threshold. The results of EN and LASSO is positively related to the real values, but the correlation parameters are significantly different from the truth. Linear, RR, SVR(POLY) and Theil predicted the values that are relatively closer than that of the other baselines, but with high variance. The proposed method, ICNet, not only predicted the value very precisely but also with small variance.

5.4.4 Case Study: Attentions on Attributes

The subsection studies whether the attention mechanism can identify which attribute and highlight latent rules. Several circuits are evaluated as shown in Table (6.8). Gate number consistently attracted higher attention than the gate type by 9.64% on average. This guides us to study the correlation between real runtime and gate number. The Pearson(P) and Spearman(S) correlation are 0.8238 and 0.9722 on average in Table (6.8), which show there does exist a strong correlation. Then a linear regression model was performed to extract the quantitative relationship as practical rules: take circuit c7553 as an example, the runtime is 2.37% of runtime. Different circuits show different linear parameters. This gives us a convenient message that can accurately predict the deobfuscation time and could serve circuit obfuscation task.

circuit	gate #	gate type	corr(P/S)	linear param
c7553	56.40%	43.59%	0.8754 / 0.9345	0.0237
c499	54.39%	47.05%	0.8149 / 0.9965	0.1300
c2670	52.94%	47.05%	0.7769 / 0.9753	0.0559
c1335	56.27%	43.72%	0.8282 / 0.9846	0.0599

Table 5.3: Case study: attributes and extracted rules.

5.5 Conclusion

In this work, we have introduced a neural network model for recovering SAT runtime on ICs. To properly fuse graph structure and gate attributes, an enhanced graph convolutional operator is introduced. The proposed method can avoid attribute propagation which is in the original GCN but not suitable for ICs. An attention based component is designed to derive interpretable rules for obfuscation engineer. Experiments on real-world datasets suggest that the proposed model is capable of modeling the runtime regarding the circuit graph accurately.

Chapter 6

CNF based Circuit Deobfuscation Runtime Estimation with Survival Analysis

This chapter formulates circuit deobfuscation as a binary satisfiability (SAT) problem. Our proposed model, CNFNet, is designed for an SAT graph, i.e., dynamic size of bipartite graphs. Utilizing energy model from physics, CNFNet calculates an energy indicator as the representation of each deobfuscation instance. The model is proved to be effective in quickly estimating the deobfuscation runtime without running simulation on multiple popular benchmarks.

6.1 Introduction

The considerable high capital costs on semiconductor manufacturing motivate most semiconductor companies to outsource their designed integrated circuits (ICs) to the contract foundries for fabrication. Despite the reduced cost and other benefits, this trend has led to ever-increasing security risks such as IC counterfeiting, piracy and unauthorized overproduction by the contract foundries [132]. The overall financial risk caused by such counterfeit and unauthorized ICs was estimated to be over $169 billion per year [96]. The major threats from the attackers arise from reverse engineering an IC by fully identifying its functionality by stripping it layer-by-layer and extracting the unveiling gate-level netlist. To prevent such reverse engineering, IC *obfuscation* techniques have been extensively researched in recent years [158]. The general idea is to obfuscate some gates in an IC so that their gate type cannot be determined by reverse engineering optically, yet they preserve the functionality same as the original gates. As shown in Fig. (6.1)(a), obfuscation is a process which selects a part of the circuit (in pink) and modifies the structure of them, whose functionality can be retrieved only if correct keys are provided at the input.

Such techniques were highly effective until very recent progress of the attacking techniques

based on logical attackers were invented and widely applied [37]. This is due to that there are limited types of gates (e.g., AND, OR, XOR) in IC, so the attackers can just brute force all the possible combinations of types for all obfuscated gates to find out the one that functions identically to the targeted IC. However, brute force is usually prohibitively expensive. More recently, efficient methods such as satisfiability (SAT) based attacks have been proposed which have attracted enormous attention [86].

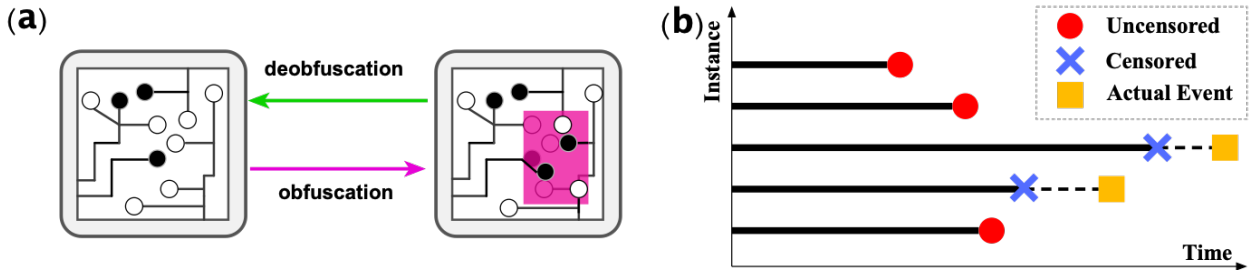


Figure 6.1: (a) An illustration of obfuscation and deobfuscation. (b) An illustration demonstrating the survival analysis problem.

Such techniques were highly effective until very recent progress of the attacking techniques based on logical attackers was invented and widely applied [37]. More recently, efficient methods such as satisfiability (SAT) based attacks have been proposed, which have attracted enormous attention [86]. The runtime of the SAT attack to reverse engineer the IC mostly depends on the complexity of the obfuscated IC, which can vary from milliseconds to days or years. Therefore, a successful obfuscation requires attackers a prohibitive amount of time (i.e., many years) to deobfuscate. However, gates to obfuscate come at a high cost in finance, power, and space, such trade-off forces us to search for optimal layout instead of purely increasing their quantity. Therefore, the best set of gates for being obfuscated maximizes the runtime for deobfuscating. However, until now it is still generally based on human heuristics or experience, which is seriously arbitrary and suboptimal [64]. This is because it is unable to "try and error" all the different ways of obfuscation, as there are millions of combinations to try and the runtime for each try (i.e., to run the attacker) can be days, weeks, or years. This situation also causes the early stop of numerous attack simulations since the unknown attack times may be dramatically long, and the computational resource is limited. Such incomplete simulations cannot be used by normal regression models due to their inaccurate labels (i.e., early stop time). These incomplete records are similar to censored data in survival analysis [143], as shown in Fig. 6.1(b). The actual event of censored data (i.e., actual time of finishing deobfuscation) is unknown, while the censored time point is recorded. For uncensored data, the recorded time is the exact time of completing deobfuscation.

To address this issue, this paper focuses on efficient and scalable ways to estimate the runtime for an attacker to deobfuscate an obfuscated IC. This research topic is vastly under-explored because of its significant challenges: **1) Difficulty in characterizing the hidden and sophisticated algorithmic mechanism of attackers.** Over the recent years, a large number of deobfuscation methods have been proposed with various techniques [64]. In order to

practically beat the attackers, methods with sophisticated theories, rules, and heuristics have been proposed and adopted. The behavior of such highly-nonlinear and strongly-coupling systems is prohibitive for conventional simple models (e.g., linear regression and support vector machine [13]) to characterize. **2) Difficulty in extracting determinant features from discrete and dynamic graph-structured ICs.** The inputs of the runtime estimation problem is the ICs with selected gates obfuscated. Conventional feature extraction methods are not intuitive to be applied to such type of varying-structured data without significant information loss. Hence, it is highly challenging to intactly formulate and seamlessly integrate them as mathematical forms that can be input to conventional machine learning models. **3) Requirement on high efficiency and scalability for deobfuscation runtime estimation.** The key to the defense against deobfuscation is the speed. The faster the defender can estimate the deobfuscation runtime for each candidate set of obfuscated gates, the more candidate sets the defender can estimate, and hence the better the obfuscation effect will be. Moreover, the estimation speed of deobfuscation runtime must not be sensitive to different obfuscation strategies in order to make the defender strategy controllable.

This work addresses all the above challenges and proposes the first generic framework for deobfuscation runtime prediction, based on Conjunctive Normal Form (CNF) graph representation for obfuscated Circuits. The major contributions of this paper are:

- **Formulating a graph learning framework for predicting deobfuscation runtime.** We formulate obfuscation runtime prediction as a graph learning problem. In the proposed method, a graph-based model is built by transforming obfuscated ICs into a CNF graph. The model then learns the relationship between the complexity of CNF graph and deobfuscation runtime.
- **Proposing a new framework, CNF-Net, for deobfuscation runtime estimation based on graph deep learning.** To model SAT-based deobfuscation, a CNF graph with multi-order is proposed to derive informative representations from obfuscated ICs. Such end-to-end deep graph regressor can automatically extract the discriminative features that are determinants to the estimation of the deobfuscation runtime to achieve accurate runtime prediction.
- **Designing an energy-based neural layer to process varying size of graph data.** To unify the dynamic topology of CNF graph, this work innovatively leverages the energy of restricted Boltzmann machine to quantify the complexity of CNF. The bipartivity of CNF graph is highly utilized to optimize the computational cost.
- **Conducting comprehensive experimental evaluations and analyses on multiple datasets.** The proposed method is compared with several state-of-the-art methods on six real-world benchmarks. The analyses of the performance and effectiveness demonstrate the advantage of our method.

The rest of the paper is organized as follows: Section 6.2 reviews existing work in this

area. Section 6.3 formulates the runtime prediction problem. Section 6.4 elaborates a novel energy-based operator for graph and presents model structure. In section 6.5, experiments on multiple datasets is presented. This paper concludes by summarizing the study's important findings in Section 6.6.

6.2 Background and Related Work

Here, we discuss the logic obfuscation and SAT attacks followed by graph convolutional networks and the relevant works.

Logic Obfuscation and SAT Attacks: Logic obfuscation often referred to as logic locking [157] is a hardware security solution that facilitates to hide the Intellectual Property (IP) using key-programmable logic gates. Although obfuscation schemes try to minimize the probability of determining the correct key by an attacker, and avoid making pirated and illegal copies, introducing SAT attack shows that these schemes can be broken [133]. In order to perform SAT attack, the attacker is required to have access to the functional IC along with the obfuscated netlist, and different SAT-hard schemes such as [151, 156] are proposed. Furthermore, new obfuscation schemes that focus on non-Boolean behavior of circuits [150], that are not convertible to an SAT circuit is proposed for SAT resilience. Some of such defenses include adding cycles into the design [121]. By adding cycles into the design may cause that the SAT attack gets stuck in the infinite loop, however, advanced SAT-based attacks such as cycSAT [164] can extract the correct key despite employing such defenses. Before the proposed defense ensures robustness against SAT attacks, the defenders need to run the rigorous simulations which could range from few minutes up to few days or even years.

Graph Neural Networks: Many graphs and geometric convolution methods have been proposed recently for modeling graph data. The spectral convolution methods [32, 65] are the mainstream algorithms developed as the graph convolution methods. Their theory is based on the graph Fourier analysis [125]. The polynomial approximation is firstly proposed by [50]. Inspired by this, graph convolutional neural networks (GCNNs) [32] successfully generalize the powerful convolutional neural networks (CNN) in dealing with Euclidean data to modeling graph-structured data. Kipf and Welling proposed a simplified type of GCNNs [65], called graph convolutional networks (GCNs). The GCN model naturally integrates the connectivity patterns and feature attributes of graph-structured data and outperforms many state-of-the-art methods. Another category of graph CNN is spatial domain method such as Diffusion CNNs or DCNN [7] which considered a diffusion process on the graph. DCNN generate different features by applying a diffusion kernel of different size. MoNet[102] generalized spectral and spatial methods in the same framework.

Runtime Distribution(RTD): Runtime predictions have boosted a wide range of meta-algorithmic tasks such as algorithm selection [154], model configuration [57], generating

hard benchmark [78], gaining insights info instance [128], and algorithm performance[58]. Predicting algorithm runtimes focuses on mean runtimes. [59] predicted single parameter of exponential function and [5] predicted two parameters of log normal and exponential RTD. [40] proposed neural network to learn a distribution of the runtime. [88] proposed to use text format as input to obtain representation. [36] solving hard combinatorial problem in AI include cases with high variations in runtime.

6.3 Problem Setup

This section presents the formulation and challenges of deobfuscation runtime prediction for integrated circuits (ICs).

Mathematically, the ICs deobfuscation problem can be considered equivalently as solving the Boolean satisfiability (SAT) of a conjunctive normal form (CNF). To see this, specifically, Fig. 6.2 exemplifies for us this process as three steps. **Step 1** shows the obfuscated IC where several gates (the two blue gates) have been encrypted into the new blue components with additional key inputs (i.e., $x1$, $x2$, and $x3$) shown in **Step 2**. These new components can be equivalent to the original IC only when the key inputs are correctly inferred. This will be achieved only when the inferred circuit in Step 2 and the original one in Step 1 will always produce the same output value of Y given the same values of the inputs A, B, C, D. Such a problem is routinely formulated as a CNF expression shown in **Step 3**, and the solving of it is a standard SAT problem, which has been proved to be NP-complete [26, 63]. This means that its solving runtime cannot be tightly estimated theoretically and hence an accurate runtime prediction method is imperative.

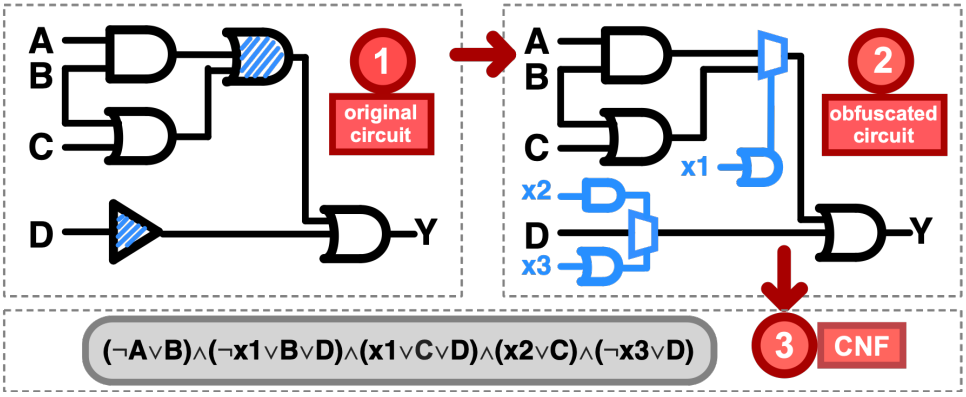


Figure 6.2: Illustration of transformation from original circuit to obfuscated circuit, and then to CNF representation.

Therefore, the runtime prediction is dependent on the CNF expression, because it stores all the necessary information for the runtime estimation as mentioned above. As a standard

formula of a Boolean expression, CNF is a conjunction of one or more clauses, where a clause is a disjunction of literals. In other words, in CNF, different clauses are connected by "AND" operators while each clause consists of one or more literals (or their negations) connected by "OR" operators. In practice, the CNF formation of an obfuscated IC can be conventionally generated through a set of handwritten rules [132]. Hence, we formally present the problem formulation of this paper as follows:

Problem Formulation: Given the CNF (denoted as CNF_i) of the ith obfuscated IC, the goal of this work is to predict the runtime T_i by a runtime prediction function $f : CNF_i \rightarrow T_i$, where T_i is typically a non-deterministic value.

The above is a new, yet extremely challenging research problem that involves three major technical challenges: **1) Difficulty in representing the CNF in an intact and structured way for a machine learning model.** CNF, though typically written as a sequence, is mathematically not a sequence as the order among different clauses is meaningless. Moreover, one literal can appear in multiple clauses with or without their negation forms, which further complicates its representation. However, there is no such an existing machine learning model that is designed for directly treating such CNF expression as input without information loss, while extracting hand-crafted features from it will surely lose information and be easily biased. **2) Difficulty in learning the mapping from the CNF to the runtime.** Different from conventional inputs of machine learning models, CNF inherently endorses logical operators (typically discrete) instead of numerical operators. Moreover, it is imperative yet, extremely difficult to automatically learn the determinant features that decide how "time-consuming" deobfuscating a CNF is. **3) Non-deterministic and non-normal distributed output.** Although theoretically SAT over CNF is an NP-complete problem, the empirical runtime complexity is typically much less than exponential. However, the runtime of the SAT could also vary due to the different initialization and hyper parameters, following a distribution which is not necessarily a standard normal distribution. Such non-deterministic non-normal-distributed output is not able to be handled by conventional prediction models.

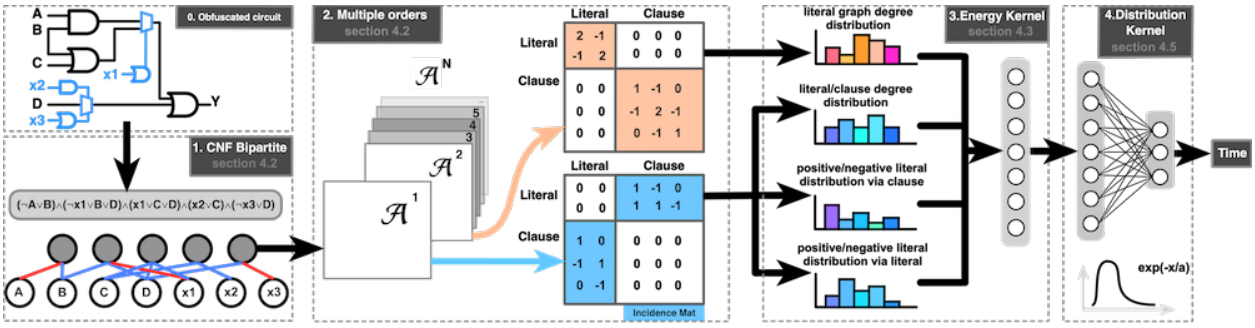


Figure 6.3: Architecture of CNF-Net: 1) extract CNF graph from obfuscated circuit; 2) derive multiple order information from graph representation; 3) apply energy kernel to aggregate intermediate features; 4) utilize distribution layer to sample runtime.

6.4 The proposed model: CNF-Net

To address the above challenges, we first present a comprehensive representation of the CNF graph as a bipartite graph with its multi-order analysis and then elaborate our model named CNF-Net based on our novel clause-literal bipartite graph operators.

6.4.1 Model Architecture

This section presents a high-level workflow of our CNF-Net. As shown in Fig. 6.3, an obfuscated IC is encoded into the corresponding CNF, which will be mathematically formulated as a bipartite graph, called *CNF bipartite graph* as introduced in Section 6.4.2. Also, following the proposed method in section 6.4.2, multiple order information is extracted from the CNF bipartite graph by calculating the power series of its adjacency matrix. By leveraging the property of the bipartite graph, the computational cost is significantly reduced. Without loss of generality, the first two orders are considered in this model, and it is easy to extend to any order. As discussed in Section 6.4.2, the odd order information only needs incidence matrix which is less than 25% of the adjacency matrix of the CNF bipartite graph, while the even order information can also save the similar amount of storage and computation. This feature will significantly reduce the computational cost compared with typical graph neural networks. After extract raw features of *CNF bipartite graph*, an energy-based[113] kernel is proposed to model the dynamic-size data. This new kernel calculates the energy which identifies the complexity of corresponding CNF bipartite graph. Considering the non-deterministic property of SAT runtime, a distribution kernel is leveraged in order to produce the final output.

6.4.2 CNF Bipartite Graph Representation

The CNF of an obfuscated circuit is modeled as an undirected and signed bipartite graph which uses one node type for clauses and the other for literals. This *CNF bipartite graph* is exemplified in Fig. 6.4 and defined as $\mathcal{G}(E, V^{literal}, V^{clause})$, where $V^{literal}, V^{clause}$ indicate the set of literal and clause nodes, respectively. The sign of an edge between a literal l and a clause c denotes whether l is negated or not in c. That means, the edge value is:

- 1 (positively connected), if l is in c, and l is positive;
- -1 (negatively connected), if l is in c, and l is negative;
- 0 (disconnected), if l is not in c.

Based on the above formulation, we denote the i-th CNF bipartite graph as $\mathcal{G}_i(E_i, V_i^{literal}, V_i^{clause})$, with adjacency matrix $\mathbf{A}_i \in \mathbb{R}^{N_i \times N_i}$, where $N_i = |V_i^l| + |V_i^c|$ is the total number of literal and

clause nodes. Typically, the CNFs for different obfuscated of the same circuit (or different circuits) are different, leading to $\mathbf{A}_i \neq \mathbf{A}_j$ given $i \neq j$.

The CNF bipartite graph provides a comprehensive representation of the CNF without information loss, and hence enables the end-to-end frameworks to sufficiently learn any important features automatically. Moreover, we find that the CNF bipartite graph is a powerful representation such that its multi-order version can also capture additional meanings of a CNF. Specifically, here, the 1st- and 2nd-order information is studied first and then expand to higher orders. This multi-order information is used as an immediate input representation of an obfuscated IC to explore the patterns associated with the deobfuscation runtime estimation.

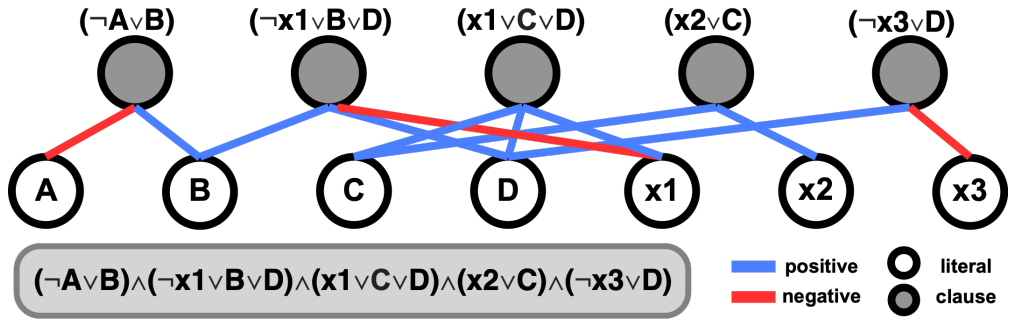


Figure 6.4: From CNF to 1st order graph.

1st and 2nd order CNF bipartite graph information: The 1st order information is the connectivity between literals and clause, i.e., adjacency matrix $\mathbf{A}$. Note that $\mathbf{A}$ is shown on the left side of Fig. 6.5, which only has zero values for the diagonal blocks and non-zero values in other parts, each of which is an incident matrix. Since the incidence matrix is symmetric in $\mathbf{A}$, only one blue block is sufficient to represent $\mathbf{A}$ without any information loss, leading to significant computational savings. The 2nd order graph holds 2nd order connectivity which can be obtained by taking a square of the adjacency matrix, i.e., $\mathbf{A}^2$. Then the numbers in $\mathbf{A}^2$ indicate the connections only between the same type of nodes. As shown on the right side of Fig. 6.5, the numbers in the diagonal blocks are all zero, and non-zero otherwise, which exactly the reverse case of the 1st order adjacency matrix. This implies that there is no need to feed the whole $\mathbf{A}^2$ into the model as well. Section 6.4.3 will show a method which fully utilizes this property to dramatically reduce the computation cost.

Odd and even order information: Following this, the 3rd order adjacency matrix indicated the connectivity with 3rd order neighbors and had zero values again in the diagonal blocks while zeros elsewhere, the same as the 1st order one shown on the left of Fig. 6.5. Similarly, the 4th order one is the same way as that of the 2nd order one, so on so forth. This means we can save significant amount of computations when considering higher-order graph information in the same way as we discuss the case for 1st- and 2nd-order graph. There is a trade-off between efficiency and order number since more orders mean more information but take more computational resource. Therefore, we only consider the 1st and 2nd order to improve

efficiency, and it can be easily extended to any order as needed.

6.4.3 Energy-based Operators for CNF Graph

The next task is to effectively learn the mapping from the CNF bipartite graph to the non-deterministic runtime, as well as the discriminative features during the prediction process. This task cannot be effectively handled by existing graph classification or regression models because of the unique properties of both the input and output. Different from the conventional graphs, the correlation among the neighboring nodes in the CNF bipartite graph does not indicate "proximity" or "similarity" but instead indicates logical relation with signed edges in a variable-size bipartite graph. To address this unique issue, novel graph encoder layers have been proposed by leveraging and extending the energy of restricted Boltzmann machine (RBM).

RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form an undirected bipartite graph where the two types of nodes are commonly referred to as the "visible" and "hidden" units respectively.

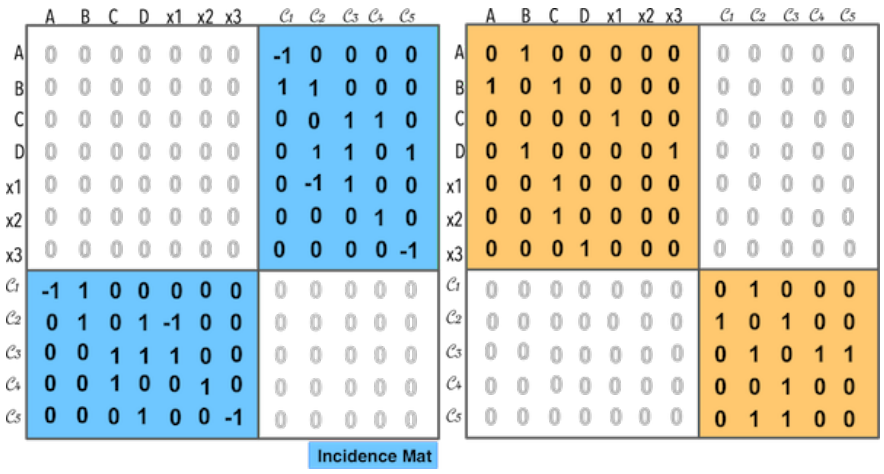


Figure 6.5: A simple example showing (left): adjacency matrix of 1st order graph: $\mathcal{A}$, which models the example in Fig. 6.4; and (right): adjacency matrix of 2nd order graph: $\mathcal{A}^2$. There are 5 clauses: clause $\mathcal{C}_1 = \{\neg A, B\}$, clause $\mathcal{C}_2 = \{\neg x1, B, D\}$, clause $\mathcal{C}_3 = \{x1, C, D\}$, clause $\mathcal{C}_4 = \{x2, C\}$, clause $\mathcal{C}_5 = \{\neg x3, D\}$

RBMs for CNF bipartite graphs

By innovatively treating literals and clauses as visible and hidden units, the CNF bipartite graph can be modeled by RBM. The energy of the original RBM is hence defined as:

$$E(\mathbf{v}, h) = -\sum_m a_m v_m - \sum_n b_n h_n - \sum_m \sum_m v_i w_{m,n} h_n \tag{6.1}$$

or in matrix notation:

$$E(\mathbf{v}, h) = \underbrace{-\mathbf{a}^\top \mathbf{v}}_{\text{visible}} - \underbrace{\mathbf{b}^\top \mathbf{h}}_{\text{hidden}} - \underbrace{\mathbf{v}^\top \mathbf{W} \mathbf{h}}_{\text{interaction}}, \tag{6.2}$$

where **v** and **h** are the values of visible and hidden nodes respectively, while **a, b, W** are weights to learn. In sum, the first component is energy of visible nodes, the second one is of hidden nodes, and the last is the interaction energy between visible and hidden nodes. In terms of CNF bipartite graph, **v** and **h** are the representations of a literal and a clause, respectively. Similarly, an energy form is defined for characterizing a CNF:

$$E = -\alpha \cdot E_{literal} - \beta \cdot E_{clause} - \gamma \cdot E_{interaction} \tag{6.3}$$

where $E_{literal}$, E_{clause} and $E_{interaction}$ are the energies of literals, clauses, and their connections, and α, β, γ are the weights of them, respectively. Since SAT runtime estimation over CNF is a highly nonlinear process, traditional linear function has been generalized by us into a new non-linear version:

$$E = f_\Phi(E_{literal}, E_{clause}, E_{interaction}), \tag{6.4}$$

where f_Φ is a neural network function controlled by parameter Φ. In the following, we study bipartite connection energy $E_{interaction}$ first and then $E_{literal}, E_{clause}$ in turn. Based on RBM, $E_{interaction}$ is defined as linear function of literals:

$$E_{interaction} = \sum_m \sum_n v_m w_{m,n} h_n, \tag{6.5}$$

where v_m is a literal and h_n is a clause in one single CNF bipartite graph $\mathcal{G}_i$. However, $E_{interaction}$ is not necessarily a sum function. Therefore, we generalize $E_{interaction}$ inspired by generalizing convolutional graph layers into bipartite graphs. However, most existing graph deep learning operators focus on graphs with fixed topology while the size and topologies of CNF bipartite graph vary across different instances dramatically. To solve this problem, we design a kernel for aggregating interaction information in one graph. Specifically, a d-dimensional vector of pseudo-coordinates is associated with $[\mathbf{v}, \mathbf{h}]$, we define a weighting kernel $Z_\Theta(\cdot, \cdot)$, so that for one CNF bipartite graph $\mathcal{G}_i$, we have:

$$E_{interaction} = \sum_m \sum_n Z_\Theta(\mathbb{E}(\mathbf{v}_m, \mathbf{h}_n)) \cdot \mathbb{E}(\mathbf{v}_m, \mathbf{h}_n), \tag{6.6}$$

where $Z_\Theta(\cdot)$ projects the $[\mathbf{v}, \mathbf{h}]$ into a new value as the weight of $[\mathbf{v}, \mathbf{h}]$, and $\mathbb{E}(\mathbf{v}_m, \mathbf{h}_n)$ represents the interaction or edge value between $\mathbf{v}_m$ and $\mathbf{h}_n$ such as 1, -1 or 0. Note that Z_Θ function is implemented by neural networks and controlled by fixed-size parameters Θ. Similarly, we further generalize $E_{literal}$, E_{clause} as:

$$E_{literal} = \sum_m Z_\Theta(\mathbf{v}_m) \cdot \mathbf{v}_m, \tag{6.7}$$

and

$$E_{clause} = \sum_n Z_\Theta(\mathbf{h}_n) \cdot \mathbf{h}_n, \tag{6.8}$$

where $\mathbf{v}$ and $\mathbf{h}$ indicate attributes of literal and clause respectively. Therefore, the final model for CNF is:

$$\begin{aligned} E = f_\Phi(&\sum_m Z_\Theta(\mathbf{v}_m) \cdot \mathbf{v}_m, \sum_n Z_\Theta(\mathbf{h}_n) \cdot \mathbf{h}_n, \\ &\sum_m \sum_n Z_\Theta(\mathbb{E}(\mathbf{v}_m, \mathbf{h}_n)) \cdot \mathbb{E}(\mathbf{v}_m, \mathbf{h}_n)), \end{aligned} \tag{6.9}$$

Energy model for 1st-order graph operators

Eq. (6.9) above dose not consider the sign of the edges between literals and clauses. Hence, positive and negative information is encoded separated: $E = \{E^+, E^-\}$. To capture the sign information, the corresponding incidence matrix $INC \in \mathbb{R}^{|V^{literal}| \times |V^{clause}|}$ is utilized:

- **Normalized positive and negative edge distribution in clause scope (NPNC)**: we count positive and negative edges for each clause, and take normalization of both positive and negative counts, so there are two values for each clause, i.e., c^{pos} and c^{neg}. If there exist $|V^{clause}|$ clauses, there will be $2|V^{clause}|$ number of features:

$$\begin{aligned} &\left\langle c^+_{clause}(0), c^-_{clause}(0) \right\rangle, \left\langle c^+_{clause}(1), c^-_{clause}(1) \right\rangle, ..., \\ &\left\langle c^+_{clause}(|V^{clause}| - 1), c^-_{clause}(|V^{clause}| - 1) \right\rangle, \end{aligned} \tag{6.10}$$

  which can be obtained by column-wise summation on positive values only and negative only of incidence matrix.

- **Normalized positive and negative edge distribution in literal scope (NPNL)**: Similarly, positive and negative degrees are counted for each literal, and the normalization per literal is taken. there will be $2|V^{literal}|$ number of features:

$$\begin{aligned} &\left\langle c^+_{literal}(0), c^-_{literal}(0) \right\rangle, \left\langle c^+_{literal}(1), c^-_{literal}(1) \right\rangle, ..., \\ &\left\langle c^+_{literal}(|V^{literal}| - 1), c^-_{literal}(|V^{literal}| - 1) \right\rangle, \end{aligned} \tag{6.11}$$

which can be obtained by row-wise summation on positive value only and negative value only of the incidence matrix.

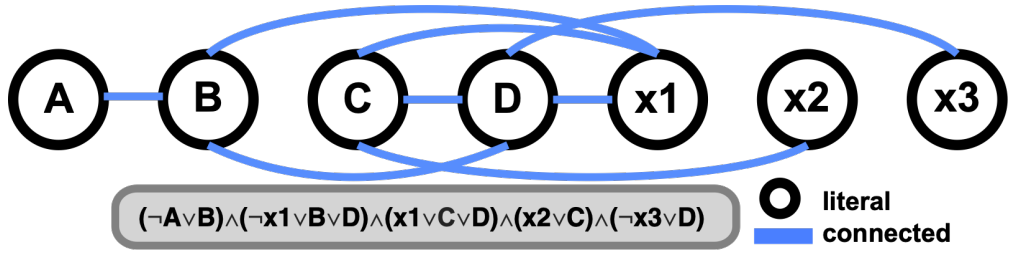


Figure 6.6: Illustration of transformation from the CNF to literal-literal graph.

In sum, positive- and negative-valued edges are treated as separate operators, so we have 2 feature maps for NPNC after normalization:

$$\left\{\frac{c^{+}_{literal}(i)}{c^{+}_{literal}(i)+c^{-}_{literal}(i)}\right\}_{i=0}^{|V^{literal}|-1}, \left\{\frac{c^{+}_{literal}(i)}{c^{+}_{literal}(i)+c^{-}_{literal}(i)}\right\}_{i=0}^{|V^{literal}|-1}. \tag{6.12}$$

Similarly, another two feature maps for NPNL:

$$\left\{\frac{c^{+}_{clause}(i)}{c^{+}_{clause}(i)+c^{-}_{clause}(i)}\right\}_{i=0}^{|V^{clause}|-1}, \left\{\frac{c^{+}_{clause}(i)}{c^{+}_{clause}(i)+c^{-}_{clause}(i)}\right\}_{i=0}^{|V^{clause}|-1}, \tag{6.13}$$

where $\{\cdot\}_i^N$ denotes a list of N items from index i to N. Eq. (6.12) and (6.13) then are fed as 4 features maps into Eq. (6.6).

Energy Model for the 2nd-order graph operators

The 2nd order of graph with adjacency matrix $\mathbf{A}^2$ denotes the literal-wise and clause-wise mutual information, which corresponds to the top left and bottom right block respectively in the right subfigure of Fig. 6.5. Their physical meaning is:

- **literal-wise**: frequency of two literals appearing in the same clause.
- **clause-wise**: frequency of two clause sharing the same literal.

Intuitively, whether two literals share the same clause or not is more important than how many times they share. To further emphasize this important trait and reduce the computational complexity, our model only distinguish zero with non-zero values in 2nd order graph information(Fig. 6.6), which means that we only consider if: two literals co-appear in the same clause or two clauses share the same literal. This can be obtained by setting all non-zero value of $\mathbf{A}^2$ to 1. This "0/1" values mean if two literal appear in the same clause for at least one time. Therefore, Eq. (6.7) or (6.8) is applied to calculate the energy of this graph, and two feature maps are built for the model.

6.4.4 Deep Survival Analysis

Due to the difficulty of obtaining labeled instance with large runtime (i.e., hard instance), since it may take days, months, or even years. Therefore, there exist numerous records that stop early, and its exact runtime is unknown, which will be referred to as "censored" data in this context. Censored data cannot be provided as labels for the regression model, but still offers information about runtime. For example, it a simulation stop early within 1 hour, the real runtime is probably larger than 1 hour.

Survival analysis is introduced to integrate uncensored data with censored data, improving the performance of regression performance. Specifically, we borrow the idea of parametric survival analysis [82] that can be used to predict the time. It finds the best parameters by optimizing the likelihood function of all instances:

$$L(\beta) = \overbrace{\prod_{\delta_i=0} f\left(T_i, \beta\right)}^{uncensored} \overbrace{\prod_{\delta_i=1} S\left(T_i, \beta\right)}^{censored}, \tag{6.14}$$

where $\delta_i = 0$ and $\delta_i = 1$ mean uncensored and censored data, $_i$ denotes predicted time, and β is the model parameter. Note that taking logarithm on Eq. 6.14 will convert the product into sum function, i.e., $\log L(\beta) = \sum_{\delta_i=0} \log f\left(T_i, \beta\right) + \sum_{\delta_i=1} \log S\left(T_i, \beta\right)$. $f(t)$, $S(t)$ indicate death density function and survival function respectively [143]. Under survival analysis, $f(t)$ is defined as the probability that the instance dies at time t, while $S(t)$ indicates the probability that the instance can survive for longer than a certain time t. However, Eq. 6.14 is designed for parametric model in which $f(t)$ and $S(t)$ have analytical forms, which is difficult to be determined especially in complex and real-world scenario.

Inspired by these concepts, we design a new objective function, imposing similar regularization to Eq. 6.14. To further improve the accuracy of prediction, a consistence loss is added to reconcile uncensored loss and censored loss. Therefore, the proposed loss consists of three components with weight parameters α and β:

$$\mathcal{L} = \mathcal{L}_{uncensored} + \alpha \mathcal{L}_{censored} + \beta \mathcal{L}_{consist}. \tag{6.15}$$

Uncensored Loss is designed to represent the regression loss for the uncensored data. We define $f_{reg}(x)$ as regression model and the uncensored loss is written as :

$$\mathcal{L}_{uncensored} = \sum_{\delta=0} \| f_{reg}(X) - \log Y \|^2, \tag{6.16}$$

where X is the features of instance, and Y is the labelled runtime of uncensored data (i.e., exact runtime). Both death density function and survival function are exponential based due to the death-age relationship. The exponential base is also employ in f_{reg}, that is why our target is $\log Y$ rather than Y. **Censored Loss** characterizes the capacity that distinguishes

whether an instance exceed a censor threshold or not. It is defined using binary cross entropy:

$$\mathcal{L}_{censored} = -\sum_{\delta \in \{0,1\}} \delta \log f_{class}(X), \tag{6.17}$$

where $f_{class}(x)$ is a binary classifier. Note that uncensored loss and censored loss corresponds to those two components in Eq. 6.14. However, there is a implicit connection between $f(t)$ and $S(t)$ [143]:

$$f(t) = (1 - S(t))t = -S(t)t, \tag{6.18}$$

and they share the same parameter β. Therefore, a regularization is proposed to implement this mechanism between f_{ref} and f_{class}. **Consistence Loss** is a constraint that forces f_{reg} to predict the right label as f_{class} does. Intuitively, if an instance is censored, f_{reg} must output a value that is large than the threshold even though exact runtime is unknown for the censored instance. Specifically, consistence loss is defined as a ReLU function:

$$\mathcal{L}_{consist} = ReLU[(f_{reg} - \Gamma) \cdot (1 - 2\delta)], \tag{6.19}$$

where Γ is a fixed censor threshold, and $(1 - 2\delta)$ is a mapping that transfer $\delta \in \{0, 1\}$ to $\delta \in \{1, -1\}$. Therefore, when $f_{reg} > \Gamma$ (i.e., the test instance is hard instance, and should be censored) and $\delta = 0$ (i.e., uncensored), it is a inconsistent. In this case, we add a punishment as above: $(f_{reg} - \Gamma)$ and $(1 - 2\delta)$ always have the same sign if inconsistence happens. ReLU is employed to remove consistent cases from loss.

6.4.5 Relationship with Spatial Graph NN

As a state-of-art of the spatial graph neural networks model, the superiority of DCNN [7] beyond the other graph deep learning models is the capacity in handling graphs with variable sizes and considering multiple orders, which is also one advantage of our model. Our study demonstrates that DCNN is a special case of our CNF-NET:

Lemma 6.4.1. *CNF-Net is a generalization of DCNN, while DCNN is a special case of CNF-Net when setting the feature aggregation to mean function.*

Proof. DCNN can be extended to whole graph modeling by taking the mean activation over the features on diffusion $P_t^* X_t$:

$$Z = f(W^c \odot 1_{N_t}^T P_t^* X_t / N_t) = f(W^c \odot \underbrace{(\frac{1}{N_t})_{N_t}^\top}_{\text{aggregation}} P_t^* X_t), \tag{6.20}$$

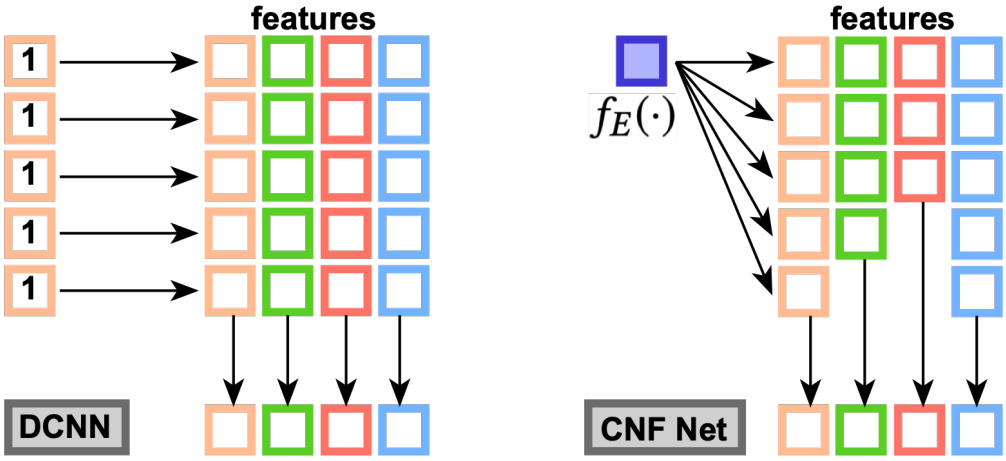


Figure 6.7: Comparison with DCNN

where $(\frac{1}{N_t})_{N_t}$ is a $N_t \times 1$ vector of value $\frac{1}{N_t}$), t indicates the graph index, and P^* is power series of adjacency matrix. Following the same representation, we can rewrite CNF-Net as:

$$Z = f(W^c \odot \underbrace{f_E(\cdot)}_{\text{aggregation}} P_t^* X_t), \tag{6.21}$$

where $f_E(\cdot)$ represents a vector of $\left[Z_\Theta(\phi(i))\right]_{i=0}^{d_{feat}-1}$, and d_{feat} indicate dimension of a feature and $\phi(i)$ is the i-th value along a feature. Therefore $f_E(\cdot)$ is a vector of dynamic size. □

This difference brings two advantages: (1) The feature aggregation is learned from the data rather than average, which is extremely important for the high non-linearity in deobfuscation runtime estimation; (2) CNF-Net can work for the case where the number of feature maps changes across different graphs, while DCNN still requires the number to be fixed. As shown in Fig. 6.7, $f_E(\cdot)$ calculate a weight vector for each number of feature maps by repeatedly applying the one-dimensional function $Z_\Theta \cdot$

6.4.6 Algorithm Description

The algorithm (4) first prepare CNF graph adjacency as obfuscated circuit representation (line 2). Then power series are extracted based on CNF graph (line 3). The model computes feature maps based on power series 4. All model parameters are initialized by Gaussian distribution (line 5 to 6). In each training iteration, energy-based kernel is applied to literals, clauses and their interaction (line 9 to 10). The aggregation of energy is treated as features of the targeted obfuscated IC. To model the variance of runtime for similar or exactly the same instance, a distribution kernel is applied in the last layer. This distribution kernel can be replaced with any other suitable distributions such as logarithmic or exponential kernel.

Algorithm 4: CNF-Net

Input: $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, ... \mathcal{G}_i ..., \mathcal{G}_N\}$, the real runtime T_i for instance $\mathcal{G}_i$

Output: a neural network function with parameters Φ, Θ, parameters of fully connected layers(τ), and distribution parameter σ

// data preparing

transform from obfuscated circuit $\mathcal{G}_i$ into CNF representation CNF_i and derive bipartite graph

extract adjacency matrix $\mathbf{A}$ from this bipartite graph and calculate power series : $\{\mathbf{A}^1, \mathbf{A}^2, ..., \mathbf{A}^N\}$ ▷ Eq. (6.12) and (6.13)

compute distribution features based on power series ▷ Eq. (6.7)

$\theta = \{\Theta, \Phi, \tau, \mu, \sigma\}$

initialize θ with standard Gaussian

// update CNF-Net

repeat

  apply the energy kernel on laterals and clauses. ▷ Eq. (6.7)/ (6.8)

  apply the energy kernel on the connection between laterals and clauses ▷ Eq. (6.6)

  calculate the overall energy E and then feed E into fully connected layer ▷ Eq. (6.4)

  get a intermediate predicted value t, and apply distribution e^t as predicted time calculate residues $\delta = T - e^t$

  compute derivatives to update parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta \delta$, where α is learning rate

until δ *convergence*;

The algorithm (4) presents such idea using normal distribution. Then typical parameter optimization(i.e., Adam) of deep neural networks is employed.

6.5 Evaluation

In this section, the performance of the proposed model CNF-Net is evaluated. All the experiments were conducted on a 64-bit machine with Intel(R) Xeon(R) CPU E3-2136 3.30GHz with 32.0GB memory.[1]

6.5.1 Benchmark datasets

For the experimental setup, we have used the ISCAS-89 benchmarks[2], and obfuscation instances on 6 different circuits are selected as 6 benchmark datasets (c432, c499, c880, c1355, c1908 and c2670). These benchmarks are synthesized using the Synopsys DC Compiler

[1] Datasets and codes will be available at (hidden for anonymous submission)

[2] `http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/verilog/`

	MSE						Pearson						Spearman					
	c432	c499	c880	c1355	c1908	c2670	c432	c499	c880	c1355	c1908	c2670	c432	c499	c880	c1355	c1908	c2670
EN	17.422	16.836	18.488	19.873	17.550	17.223	0.333	0.112	0.436	0.046	0.301	0.440	0.625	0.491	0.726	0.609	0.553	0.704
LARS	17.454	16.421	18.546	19.872	17.593	17.313	0.314	0.038	0.331	-0.000	0.393	0.332	-0.123	0.215	**0.888**	0.150	0.437	**0.895**
LASSO	17.342	17.120	18.358	19.896	17.407	17.124	0.572	0.076	0.515	-0.038	0.441	0.533	0.417	0.353	0.377	0.300	0.374	0.622
LR	17.315	17.518	18.281	19.861	17.591	16.744	0.582	-0.088	0.550	0.105	0.355	0.727	0.426	0.162	0.215	0.208	0.347	0.122
OMP	17.421	16.976	18.521	19.881	17.433	17.304	0.314	0.038	0.331	-0.000	0.393	0.332	-0.123	0.215	**0.888**	0.150	0.437	**0.895**
PAR	17.452	16.226	18.618	19.876	17.596	17.409	0.150	0.172	0.306	0.105	0.220	0.318	**0.794**	**0.927**	**0.869**	**0.947**	**0.812**	**0.871**
Ridge	17.379	17.032	18.467	19.879	17.469	17.154	0.489	0.079	0.445	0.025	0.384	0.503	0.526	0.394	0.462	0.477	0.371	0.594
SGD	68.412	72.478	65.875	73.447	72.771	75.613	-0.159	0.145	0.226	-0.121	0.202	-0.332	-0.808	**0.887**	**0.812**	-0.941	**0.894**	-0.895
SVR	17.446	16.259	18.613	19.876	17.612	17.450	0.737	0.092	0.375	0.016	0.080	0.213	0.411	0.533	0.476	0.540	0.775	**0.839**
Theil	20.350	20.089	18.512	19.941	20.063	17.106	0.012	-0.038	0.546	0.033	0.007	0.615	0.354	0.365	0.288	0.562	0.551	0.240
DistNet	174.0315	18.406	9.3729	12.8988	22.8172	7.0459	0.175	0.638,	0.5472	0.265	**0.8915**	0.4034	-0.1229	**0.8043**	0.774	**0.8903**	0.5021	0.0995
DCNN	**4.458**	**3.897**	**7.431**	**5.356**	**4.353**	**6.312**	-0.061	0.034	0.203	-0.099	-0.021	-0.031	0.030	-0.005	0.213	-0.037	-0.043	-0.030
CNF-Net	5.758	4.164	7.719	6.602	20.063	7.273	**0.736**	**0.686**	**0.770**	**0.878**	**0.710**	**0.855**	**0.794**	**0.858**	**0.888**	**0.944**	0.5519	**0.884**

Table 6.1: Prediction performance on 6 benchmark circuits: MSE, Pearson and Spearman correlation

with the help of 32/28nm Generic library[3]. The synthesized netlist is further converted to the bench format as per the SAT-solver requirements [132]. The in-house developed script replaces the gates with the Look-up-table of size 2, as described in [62]. The output of the script is the obfuscated bench file along with the adjacency matrix. Obfuscated bench file along with the original benchmark are given to the SAT-solver as input parameters, and SAT-solver tries to find the key. The time taken by the SAT-solver to find the correct key is the deobfuscation time.

6.5.2 Runtime Prediction Task

Baselines

We compare against several state of art regression models[4]: Linear Regression(LR),Passive Aggressive Regression(PAR) [27], LASSO [137], Epsilon-Support Vector Regression (SVR), Ridge Regression (RR) [105], Elastic Net(EN) [169], Orthogonal Matching Pursuit (OMP) [95], SGD Regression, Least Angle Regression (LARS) [35], Theil-Sen Estimators(Theil) [28]. In addition, a neural network for predicting runtime work DistNet [36] is employed. General regression models cannot directly learn pattern on graph data, so the experiments prepare several predefined features such as: *size of clause, size of literal, ratio of clause and literal* and so on(i.e., feature 1-27 of Fig 2 in [33]). From state-of-art graph deep learning models, DCNN [7] is chosen, since it can learn graph embeddings directly on a set of dynamic-size graphs.

[3] `https://www.synopsys.com`

[4] https://scikit-learn.org/stable/modules/linear_model.html

Metrics

In this experiment, the mean square error(MSE) of runtime prediction task were utilized as the metric for performance evaluation. Another metrics is the Spearman and Pearson correlation of prediction runtime, which represents the capacity in prediction trend of runtime.

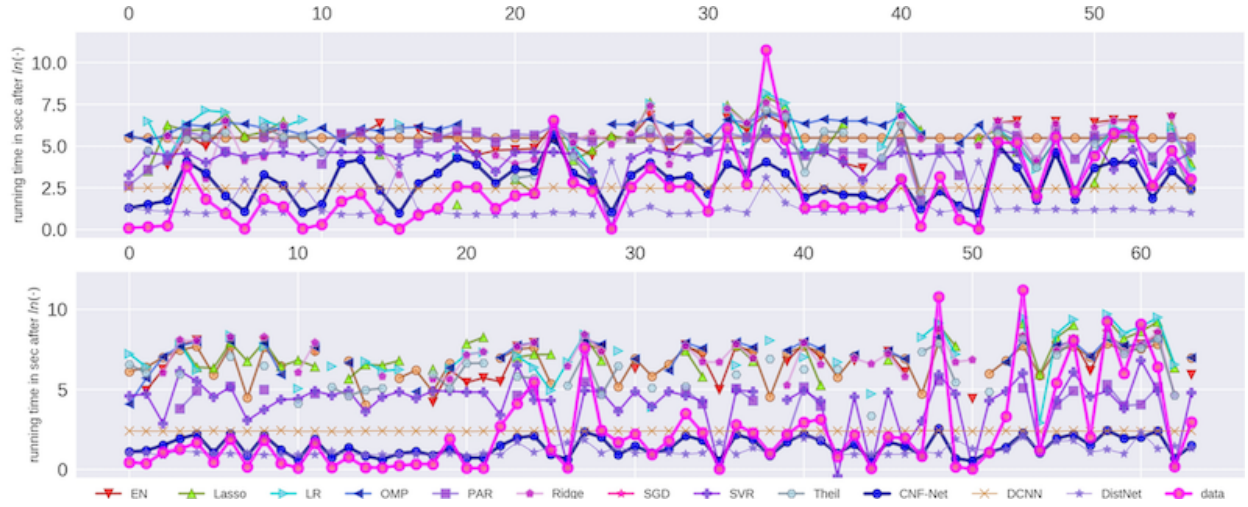


Figure 6.8: Prediction performance on samples from two datasets c432 and c880(negative values are removed): x-axis is the data index and y-axis denote predicted runtime compared with real runtime(label of*data*)

Performance analysis

Table (6.1) shows the MSE after taking ln(·), and two correlation for all the methods on 6 datasets. DCNN achieved the best MSE score throughout all datasets, while our method is the second best. MSE scores of regression models are much higher, which shows their ineffectiveness. On the contrary, this observation implies the effectiveness of the proposed methods in considering and utilizing the graph features. Regarding correlation, CNF-Net outperformed all the other baselines. This advantage confirms our model can well identify the trend of runtime. In terms of the correlation between predicted runtime and real runtime, DCN's scores are significantly small, i.e., less than 20% of the Pearson coefficient. While CNF-Net achieves around 70% correlation and up to 87% of Pearson coefficient at c1355 dataset. The same advantage of CNF-Net shows up in Spearman evaluation: the correlation score is more than 80% for five datasets, which verifies its effectiveness in estimating the runtime trend. The performance on Spearman of PAR is similar to CNF-Net, which also achieved high correlations. Overall, CNF-net is balanced and has good performance in predicting runtime task. However, the inconsistent performance of DCNN using different metrics shows the insufficiency of these metrics. Therefore, random samples were evaluated on all the methods to illustrate the differences among them. As shown in Fig. (6.10), 56

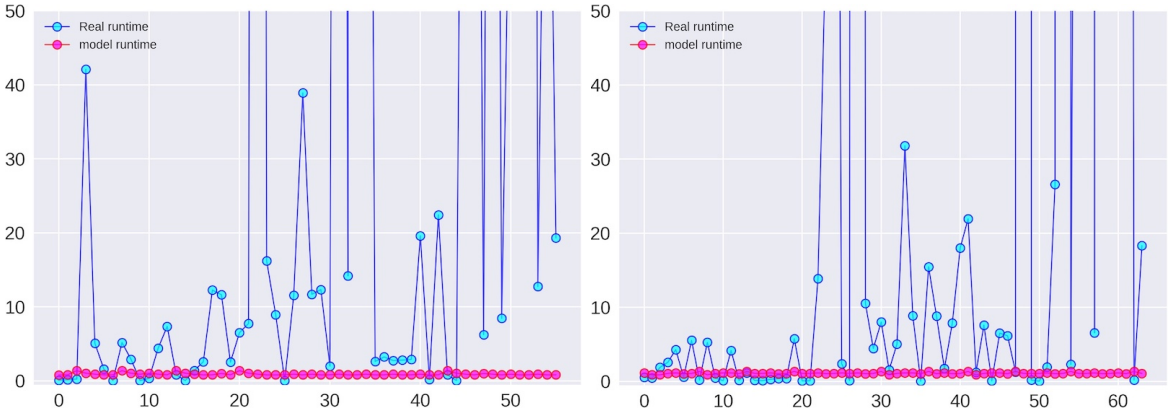


Figure 6.9: Real runtime compared with prediction time

samples from dataset c432 and 64 instances from c880 were used to perform evaluation one by one. DCNN predicted runtime by a constant value which is close to the mean of the distribution. This can help DCNN to achieve low MSE, but it is useless in practice since DCNN cannot distinguish high values from low values. CNF-Net can almost match the trend of these samples except extremely high values(Note all values listed are $\ln(\cdot)$ of the original values for plotting)

Prediction efficiency

Our task is to predict runtime efficiently. Otherwise, there is no practical value for this method. As shown in Fig. (6.9), prediction time (pink) using the proposed model is compared with real runtime in blue (i.e., running SAT-based attack). CNF-Net took a stable and little time to predict the runtime, which demonstrates its efficiency.

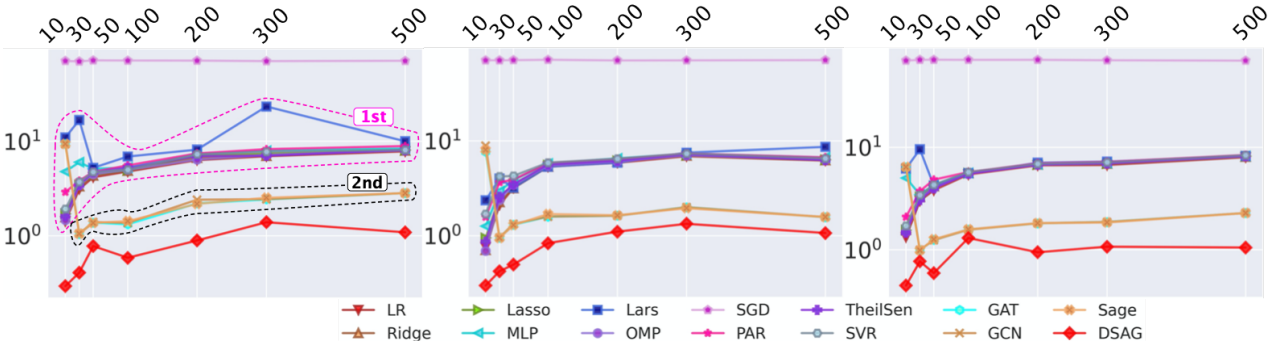


Figure 6.10: Performance Comparison on Benchmark. X-axis is the threshold (in seconds) for censoring the datasets, while y-axis indicates MSE. Left to right are c432, c499, c880 datasets.

Performance of Survival Analysis: Fig. 6.10 shows the performance comparison among

Figure 6.11: Ablation Test for DSAG: *neither* means no censor and consistence loss, while *both* indicates using both censor and consistence loss. Left to right are c432, c499, c880 datasets.

	10	30	50	100	200	300	500
Linear	3.1612e+00	2.1793e+01	6.3305e+01	1.1966e+02	5.5299e+02	1.0539e+03	2.5017e+03
Ridge	3.8178e+00	2.7043e+01	7.6082e+01	1.3196e+02	7.7698e+02	1.1116e+03	2.9605e+03
Lasso	4.5429e+00	3.0166e+01	8.6493e+01	1.4276e+02	1.1425e+03	1.6504e+03	3.1290e+03
MLP	1.2003e+02	4.0424e+02	1.6597e+02	1.9531e+02	1.5453e+03	2.8539e+03	4.3616e+03
Lars	6.1069e+04	1.9969e+07	1.8832e+02	9.9658e+02	3.5244e+03	1.4317e+10	2.2377e+04
OMP	3.3470e+00	3.0490e+01	8.7874e+01	1.4623e+02	6.0280e+02	1.2861e+03	3.0199e+03
SGD	6.0536e+30	2.8940e+30	1.8046e+31	1.3786e+31	1.1936e+31	4.9351e+30	8.8286e+30
PAR	1.7315e+01	4.5457e+01	1.1420e+02	2.6134e+02	1.7128e+03	3.8690e+03	7.4445e+03
Theil	4.0361e+00	3.0620e+01	1.1132e+02	1.8472e+02	1.0692e+03	1.1447e+03	3.2790e+03
SVR	5.8683e+00	4.0667e+01	1.1367e+02	1.5026e+02	1.4733e+03	2.3860e+03	3.4118e+03
GAT	1.1647e+04	1.8566e+00	2.9676e+00	2.7502e+00	8.0696e+00	1.0336e+01	1.6019e+01
GCN	1.1147e+04	1.8642e+00	3.0435e+00	2.9381e+00	1.0127e+01	1.0484e+01	1.6038e+01
Sage	1.2215e+04	1.8586e+00	2.9289e+00	3.1756e+00	7.8375e+00	1.1503e+01	1.5929e+01
DSAG	3.4140e-01	5.0372e-01	1.1775e+00	7.9674e-01	1.4410e+00	3.0248e+00	1.9715e+00

Table 6.2: Performance Comparison on c432

baselines and the proposed method. There are four groups in the first subfigure: 1st group as highlighted contains most normal regression models, 2nd group as marked are graph neural network models, and the remaining two are SGD (top and purple) and DSAG (bottom and red). It is obvious that normal regression, such as linear model, can hardly characterize non-linearity. Introducing graph topology, the 2nd group improves the normal regression models significantly (Note that the y-axis is in log scale). Further, the proposed DSAG improves the 2nd group dramatically. Several regression models in 1st group have higher errors in smaller thresholds (e.g., 10 and 30), which is because a smaller threshold keeps less data for regression. On the contrary, the error of DSAG decreases significantly, since DSAG sill can utilize the censored data. This superiority of DSAG is consistent throughout all three datasets. Detailed numbers of runtime prediction is provided in Table 6.2, 6.3 and 6.4.

To study each component in survival analysis, an ablation test on the proposed three losses. Fig. 6.11 shows the performance using different loss combinations. If using neither censor loss or consistence loss, the model only applies MSE as the only objective and is inclined

	10	30	50	100	200	300	500
Linear	1.1325e+00	7.2451e+00	2.3523e+01	2.0986e+02	3.6043e+02	9.9268e+02	4.9533e+02
Ridge	1.0289e+00	8.5923e+00	2.2871e+01	2.1248e+02	4.2973e+02	1.0718e+03	5.1005e+02
Lasso	1.5789e+00	1.1583e+01	3.2713e+01	2.5499e+02	4.7528e+02	1.1959e+03	5.5367e+02
MLP	2.5265e+00	1.8087e+01	4.6606e+01	3.2670e+02	6.9622e+02	1.5558e+03	8.1077e+02
Lars	9.6250e+00	5.4189e+01	2.3605e+01	2.1020e+02	3.6203e+02	1.8526e+03	5.7809e+03
OMP	9.9897e-01	9.7863e+00	2.9870e+01	2.3973e+02	4.3046e+02	1.1364e+03	5.7320e+02
SGD	7.0854e+30	7.8799e+30	9.1582e+30	1.8021e+31	5.7861e+30	6.6850e+30	1.0180e+31
PAR	3.8185e+00	3.7536e+01	4.8887e+01	3.3999e+02	6.2005e+02	1.5992e+03	8.1596e+02
Theil	1.3708e+00	1.2147e+01	3.1260e+01	2.7528e+02	4.7761e+02	1.3549e+03	5.3149e+02
SVR	4.4410e+00	6.6629e+01	7.0891e+01	3.6558e+02	6.5791e+02	1.6024e+03	7.0729e+02
GAT	2.2144e+03	1.6008e+00	2.6829e+00	3.9028e+00	4.0831e+00	6.3761e+00	3.7974e+00
GCN	7.5068e+03	1.5998e+00	2.7845e+00	3.9008e+00	4.0753e+00	6.3694e+00	3.8145e+00
Sage	3.2712e+03	1.5777e+00	2.6410e+00	4.4357e+00	4.1685e+00	6.0335e+00	3.8338e+00
DSAG	3.4719e-01	5.2389e-01	6.4042e-01	1.3016e+00	2.0157e+00	2.7992e+00	1.9103e+00

Table 6.3: Performance Comparison on c499

	10	30	50	100	200	300	500
Linear	2.7754e+00	1.9417e+01	4.4460e+01	2.2991e+02	7.9404e+02	8.3623e+02	3.0158e+03
Ridge	3.8257e+00	2.6167e+01	6.3753e+01	2.6860e+02	8.8931e+02	9.9023e+02	3.3501e+03
Lasso	3.7568e+00	2.5577e+01	6.4954e+01	2.6742e+02	8.9272e+02	1.0711e+03	3.6088e+03
MLP	1.5144e+02	3.1003e+01	7.1854e+01	2.8338e+02	9.3473e+02	1.1511e+03	3.8307e+03
Lars	5.0124e+02	1.4411e+04	6.8009e+01	2.7046e+02	1.1785e+03	1.4192e+03	3.8229e+03
OMP	3.4928e+00	2.3846e+01	5.4433e+01	2.6425e+02	8.9263e+02	1.0718e+03	3.6112e+03
SGD	3.2375e+30	1.0856e+31	1.2071e+31	1.0439e+31	1.0808e+31	5.7292e+30	3.1060e+30
PAR	7.1275e+00	4.1192e+01	1.2508e+02	3.0963e+02	1.0876e+03	1.3523e+03	4.5832e+03
Theil	3.3507e+00	2.0203e+01	5.3617e+01	2.5220e+02	8.9015e+02	1.0767e+03	3.8655e+03
SVR	4.5527e+00	3.0250e+01	7.3403e+01	3.1302e+02	1.0250e+03	1.2775e+03	4.3970e+03
GAT	6.0733e+02	1.7071e+00	2.4918e+00	3.8402e+00	5.1399e+00	5.5019e+00	8.8471e+00
GCN	6.1568e+02	1.7124e+00	2.4739e+00	3.8181e+00	5.1608e+00	5.4951e+00	8.8469e+00
Sage	6.3035e+02	1.6968e+00	2.5505e+00	3.8779e+00	5.1838e+00	5.3649e+00	8.8540e+00
DSAG	5.6682e-01	1.1624e+00	8.1095e-01	2.6881e+00	1.5796e+00	1.9264e+00	1.8682e+00

Table 6.4: Performance Comparison on c880

	10	30	50	100	200	300	500
neither	8.3943e-01	7.1515e-01	1.1436e+00	7.6344e-01	5.8402e+00	1.5851e+00	2.2145e+00
censor	4.0860e-01	7.0290e-01	9.6342e-01	3.0432e+00	1.1402e+00	1.6648e+00	2.6891e+00
consist	3.8342e-01	7.4995e-01	8.7840e-01	1.1675e+00	1.2938e+00	1.4941e+00	4.1502e+00
both	3.4140e-01	5.0372e-01	1.1775e+00	7.9674e-01	1.4410e+00	3.0248e+00	1.9715e+00

Table 6.5: Ablation test on c432

	10	30	50	100	200	300	500
neither	1.9976e-01	3.3945e-01	2.7204e-01	7.2224e-01	8.9144e-01	1.0160e+01	2.4844e+00
censor	4.4624e-01	4.5780e-01	4.3349e-01	1.2885e+00	1.2926e+00	1.5043e+00	9.1251e+00
consist	8.0903e-01	4.7088e-01	4.5495e-01	9.3713e-01	3.8482e+00	1.4911e+00	1.0419e+00
both	3.4719e-01	5.2389e-01	6.4042e-01	1.3016e+00	2.0157e+00	2.7992e+00	1.9103e+00

Table 6.6: Ablation test on c499

to output bigger errors such as the threshold of 300 in the second dataset and threshold of 200 in the third dataset. If using censor loss only, the model also has large errors such as threshold of 300 in the first dataset, threshold of 500 in the second, and threshold of 100 in the third. The remaining two combinations, i.e., consistence loss only and *both*, have similar performance and outperforms the other two (i.e, *neither* and *censor only*). *both* contains the censor loss and consistence loss, but the consistence loss performed better than censor loss only. These two losses both compare the label and the representation of the censored data, but the difference is that **censor loss** use a classification model with cross-entropy, while *consist loss* is associated with regression component. Therefore, it is natural that *consist loss* is better than *censor loss* since regression model has loss value with fine granularity. Combining *consist loss* and *censor loss*, *both* achieves the level of the better one, i.e., *consist loss*. Therefore, we can conclude that *consist loss* is the key factor that improves the one without survival analysis (i.e., *neither* in red). Detailed numbers of ablation test is provided in Table 6.5, 6.6 and 6.7.

6.5.3 Study of Model Paramters

This subsection studies the function $Z_\Phi(\cdot)$ to explore if the neural network learns a reasonable function. Correlation is evaluated between several frequently-used math function and $Z_\Phi(\cdot)$. These math functions include: $x^3 + x^2 + x(polynomial), sin(\cdot), log(\cdot), exp(\cdot), \frac{1}{x}$. The data

	10	30	50	100	200	300	500
neither	3.1560e-01	4.8206e-01	7.7573e-01	1.2841e+00	1.9549e+00	1.7395e+00	3.3581e+00
censor	5.2762e-01	1.2513e+00	9.1296e-01	1.9529e+00	1.6656e+00	7.6368e+00	4.4233e+00
consist	1.0473e+00	5.2228e-01	2.0164e+00	1.2262e+00	1.2743e+00	2.3473e+00	1.6471e+00
both	5.6682e-01	1.1624e+00	8.1095e-01	2.6881e+00	1.5796e+00	1.9264e+00	1.8682e+00

Table 6.7: Ablation test on c880

	Poly		Log		Exp		Sin		$\frac{1}{x}$	
	P	S	P	S	P	S	P	S	P	S
c432	0.92	1.00	0.96	1.00	0.69	1.00	-0.08	-0.08	-0.17	-1.00
c499	0.92	1.00	0.96	1.00	0.69	1.00	-0.08	-0.08	-0.17	-1.00
c880	0.92	1.00	0.96	1.00	0.69	1.00	-0.08	-0.08	-0.17	-1.00
c1355	-0.92	-1.00	-0.96	-1.00	-0.69	-1.00	0.08	0.08	0.17	1.00
c1908	-0.92	-1.00	-0.96	-1.00	-0.69	-1.00	0.08	0.08	0.17	1.00
c2670	0.92	1.00	0.96	1.00	0.69	1.00	-0.08	-0.08	-0.17	-1.00

Table 6.8: Correlation test for neural network function

points are sampled in (0, 100]. As shown in Table 6.8, $log(\cdot)$ has the highest scores in both correlation. Note that we consider strong correlation no matter positive or negative strong correlation, since both hold valuable information as function. If substituting $log(\cdot)$ back into Eq. (6.6), (6.7) or (6.8):

$$E = \sum Z_{\Theta}(x)x = \sum \log(x)x, \tag{6.22}$$

where x denotes one literal, clause or their edge. Eq. (6.22) is reverse value of entropy which is defined as $E = -\sum \log(x)x$. This shows that the neural network learned a function which is strongly and positively related to entropy.

6.6 Conclusion

This paper presents a novel graph neural networks framework to identify the security level of obfuscated ICs by predicting runtime of SAT attack. There is no existing work reported to address the critical challenges such as learning on the dynamic size of ICs after obfuscation. Our work proposed to employ a bipartite graph as a representation of obfuscated ICs and characterize this graph by multi-order information without any information loss. Inspired by the convolutional operator, an energy-based kernel is designed to aggregate dynamic features of the graph representation. By utilizing the special properties of the bipartite graph under our case, the computational cost is further reduced. Extensive experiments on several benchmarks demonstrated the advantageous performance of the proposed model over the existing method for runtime prediction task.

Chapter 7

Completed Work and Future Work

The proposed research aims on the design of an effective and efficient approach for graph data, and development of a specific method for graph-based application. We are focused on two major types of direction, including method-wise and the application-wise research.

For the problem of inaccurate polynomial approximation for graph convolution, we have introduced a neural network model for graph signal recovering. To estimate jump discontinuity, a rational function is employed due to its powerful ability of approximation. The proposed method can avoid multiplication with the eigenvector matrix. With the help of a relaxed Remez algorithm, RationalNet can identify the optimal configuration. In theory, RationalNet obtains exponential convergence rate on the jump signal, significantly fast than the polynomial-based approximation. Experiments on synthetic datasets suggest that the proposed RationalNet model is capable of model typical jump function accurately.

For the problem of non-robust issue of approximation techniques, we have introduced RemezNet for analytical function approximation. RemezNet jointly utilizes the advantages of the Remez algorithm and deep neural networks to estimate the parameters of the closed form. Inspired by the basic idea behind Remez, the proposed method employs the equioscillation theorem and the minimax characterization of best rational approximation. To solve the non-robustness, RemezNet initializes parameters constrained by Padé normalization and alternating sign. Extensive experiments suggest that RemezNet is capable of appa roximating functions accurately and robustly.

For the problem of netlist based circuit deobfuscation runtime estimation, this thesis presents a novel graph neural networks framework to identify the security level of obfuscated ICs by predicting the runtime of SAT attack. There is no existing work reported to address the critical challenges such as learning on the dynamic size of ICs after obfuscation. Our work proposed to employ a bipartite graph as a representation of obfuscated ICs and characterize this graph by multi-order information without any information loss. Inspired by the convolutional operator, an energy-based kernel is designed to aggregate dynamic features of the graph

representation. By utilizing the special properties of the bipartite graph under our case, the computational cost is further reduced. Extensive experiments on several benchmarks demonstrated the advantageous performance of the proposed model over the existing method for the runtime prediction task.

7.1 Research Tasks

7.1.1 Understanding Graph Convolution Methodology [A]

- **Evaluate polynomial function and rational function under function approximation task [A1]:** Two often used approximation are evaluated to verify the effectiveness in graph convolution operation.
- **Propose rational neural networks for estimate jump discontinuities in spectral domain [A2]:** To improve the traditional approximation kernel, a rational function is integrated into neural network to enhance the accuracy ini estimating graph signal.
- **Design a robust approximation for graph convolutional operation [A3]:** Rational function is more accurate but not robust, a set of theoretical constraints are employed to build a robust rational graph neural network.
- **Evaluate rational neural networks for multiple dimension signal and on more real world datasets [A4]:** One dimension graph signal is intuitive and easy to interpret, multiple dimension graph signal is however more practical and useful in real world case.
- **Identify the condition when rational neural network works [A5]:** Rational neural network is effective, but which type of case is suitable for rational function remains unknown. Both experimental and theoretical framework will be designed to analyze the rational model.
- **Review: connect spectral and spatial domain [A6]:** To understand the graph neural network community, a set of most important literatures will be collected and analyzed in both spectral and spatial framework.

7.1.2 Circuit Deobfuscation [B]

- **Propose the first graph neural networks for circuit deobfuscation task based on netlist [B1]:** By modeling circuit with graph structure, we propose a netlist based approach to predict the attach runtime.
- **Design a graph neural networks for circuit deobfuscation task based on CNF representation [B2]:** The attack is actually based on CNF representation, a CNF based

model is logically reasonable. A CNF based graph and a corresponding energy model to characterize a set of dynamic size of instances.

- **Identify the features of circuit obfuscation which incurs high attack runtime [B3]:**Obfuscation which cause high attack runtime is the most important case, therefore, the feature of this type of obfuscation remains unknown. Experimental evaluation based on domain knowledge are conducted to discovery the obfuscation pattern.
- **Optimize estimation performance across small and large runtime case [B4]:** current model is not robust across instances with small and large runtime, how to build a robust model is important for real world task.

7.1.3 Chemical compound design aid [C]

- **Unsupervised study for energy material [C1]:** Although machine learning has gained great interest in the discovery of functional materials, the advancement of reliable models is impeded by the scarcity of available materials property data. Here we propose and demonstrate a distinctive approach for materials discovery using unsupervised learning, which does not require labeled data and thus alleviates the data scarcity challenge. Using solid-state Li-ion conductors as a model problem, unsupervised learning utilizes a limited quantity of conductivity data to cluster a broad range of Li-containing materials that narrows a high-throughput screening of a large candidate list to a prioritized list for further accurate screening.
- **Property prediction for energy material in catalyst effect [C2]:** The prediction of hydrogen release ability is indispensable to evaluating hydrogen storage performance of LiBH4-based mixtures before experimentation. In order to achieve this goal, ensemble machine learning is employed to automatically infer the relationship between factors (i.e., sample preparation, mixing conditions and operational variables) and target (H 2 release amount), providing exceptional insight into hydrogen release ability.
- **Property prediction based on Dd crystal structure [C3]:** Recent graph neural networks have numerous attempts in molecular or material modeling, which is one of the most important practical applications. However, most of them focus on 2d chemical structure, missing rich information embedded in the 3D crystal structure. This direction aims to model 3D crystal structure with a novel graph neural network.
- **Automatic compound design by generative graph model [C4]:** By learning on a large set of chemical compound with hydrogen labels, a generative graph model is proposed to quick find suitable material design.

7.1.4 Brain Networks [D]

- **Build a graph representation for dynamic signal in brain network [D1]:** A brain network is important to understand the health status of human, and they are highly related. A graph model is build for the following task such as disease classification.
- **Functional connectivity prediction by structural connectivity [D2]:** The interplay between the brain's function and structure is of immense interest in neuroscience. The physical structure connectivity (SC) among brain's neurons is often used to infer the status of a patient. GNN provides a possible tool to model the complexity of SC and connect it with target properties.

7.1.5 Urban Computing [E]

- **Graph Neural Network with Kalman Filter for Traffic Speed Prediction [E2]:** The development of Intelligent Transportation Systems (ITS) requires high quality traffic data collection. Despite the advance of techniques, missing values still widely exist in traffic data collection. There are many reasons for missing data in the traffic flow, such as malfunction of the sensor, manual system closure and errors in signal transmission. We treat spatial neighbors as an estimate to recover missing data along with self estimate. Specifically, spatial neighbors are integrated by Krig graph convolutional network, and then updated as an Kalman gain beyond self estimate along the time line.
- **Graph Neural Network with Hierarchical Observation [E2]:** In the transportation or geo-location system, high-level and low-level observations are partially overlapped but provide different information. This work utilizes road-level and sensor level monitoring to predict traffic incident impact.
- **Graph Neural Network for Crime Detection in Road Networks [E3]:** People can perceive the safety level of a city block based on the physical appearances of the streets. We study the correlations between the physical appearances of the urban locations and the crime inference under the considerations of the aforementioned limitations in the urban perception community.

7.2 Future Work

The future direction of this work will be finding insightful understanding of GNNs [47, 8, 160], and novel application in various domains. To deeply understand GNNs, one possible way is to investigate the white-box model such as graph wavelet [50], and windowed graph Fourier transform. White-box models aim to identify every detail of the learning process and may reveal explicit patterns. The other way for explainability is to apply the black-box model such

as causal inference [110] and influence function [69]. Black-box models derive the behavior pattern by sufficient statistics given variables. On the other hand, applications with GNNs are very limited due to the homophily of most GNNs (i.e., neighbors are always similar). Many graphs are challenging to model with the current generation of GNNs, such as the transportation network. Exploring broader applications makes up for the deficiency and enhances the existing GNNs. These two directions facilitate the development of each other mutually and will be further investigated alternatively.

7.3 Publications

1. Ying Zhang, Xingfeng He, **Zhiqian Chen**, Qiang Bai, Charles A.Roberts, Debasish Banerjee, Tomoya Matsunaga, Yifei Mo, Chen Ling. Unsupervised Discovery of Solid-State Lithium Ion Conductors. **Nature Communications** (IF: 11.880, h5-index: 260), 2019. DOI: 10.1038/s41467-019-13214-1

2. Zhao Ding, **Zhiqian Chen**, Leon Shaw, Tianyi Ma, Wenhui Ma, Chang-Tien Lu. Predicting the Hydrogen Release Ability of $LiBH_4$-based Mixtures by Ensemble Machine Learning. **Energy Storage Materials**. DOI: 10.1016/j.ensm.2019.12.010 (accepted, IF: 15.97)

3. Subhodip Biswas, Fanglan Chen, Andreea Sistrunk, Nathan Self, **Zhiqian Chen**, Chang-Tien Lu, Naren Ramakrishnan, Geospatial Clustering for Balanced and Proximal Schools, *Proceedings of the Tenth AAAI Symposium on Educational Advances in Artificial Intelligence* (**AAAI-EAAI**), New York, NY, February 8-9, 2020. (accepted)

4. **Zhiqian Chen**, Gaurav Kolhe, Setareh Rafatirad, Chang-Tien Lu, Sai Manoj Pudukotai Dinakarrao , Houman Homayoun and Liang Zhao. Estimating the Circuit Deobfuscating Runtime based on Graph Deep Learning. Design, Automation, and Test in Europe (**DATE**), 2020. (acceptance rate: 26%, accepted)

5. Subhodip Biswas, Fanglan Chen, **Zhiqian Chen**, Andreea Sistrunk, Nathan Self, Kaiqun Fu, Chang-Tien Lu, Naren Ramakrishnan. REGAL: A Regionalization framework for school boundaries. *Proceeding of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (**SIGSPATIAL**), Paper ID: 258, Chicago, IL, Nov. 5-8, 2019. DOI: 10.1145/3347146.3359377

6. Taoran Ji, **Zhiqian Chen**, Nathan Self, Kaiqun Fu, Chang-Tien Lu, Naren Ramakrishnan. Patent Citation Dynamics Modeling via Multi-Attention Recurrent Networks. *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (**IJCAI**), pp. 2621-2627, Macao, China, Aug. 10-16, 2019. DOI: 10.24963/ijcai.2019/364 (Acceptance rate: 17.8%)

7. Zhao Ding, Shaoyuan Li, Yang Zhou, **Zhiqian Chen**, Weijie Yang, Wenhui Ma, Leon Shaw. $LiBH_4$ for hydrogen storage - New perspectives. **Nano Materials Science**, 2019. DOI: 10.1016/j.nanoms.2019.09.003

8. **Zhiqian Chen**, Feng Chen, Rongjie Lai, Xuchao Zhang, Chang-Tien Lu. Rational Neural Networks for Approximating Jump Discontinuities of Graph Convolution Operator. *Proceedings of the IEEE International Conference on Data Mining* (**ICDM**), pp. 59-68, Singapore, Nov. 17-20, 2018. DOI: 10.1109/ICDM.2018.00021 (Acceptance rate: 8.86% Full Paper)

9. Kaiqun Fu, **Zhiqian Chen**, Chang-Tien Lu. StreetNet: Preference Learning with Convolutional Neural Network on Urban Crime Perception. *Proceeding of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (**SIGSPATIAL**), Seattle, WA, Nov. 6-9, 2018. DOI: 10.1145/3274895.3274975

10. Manu Shukla, **Zhiqian Chen**, Chang-Tien Lu. DIMPL: A Distributed In-Memory Drone Flight Path Builder System. **Journal of Big Data**, Springer, July 2018, Vol. 5, No.24. DOI: 10.1186/s40537-018-0134-7

11. Xuchao Zhang, Liang Zhao, **Zhiqian Chen**, Chang-Tien Lu. Distributed Self-Paced Learning in Alternating Direction Method of Multipliers. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (**IJCAI**), pp. 3148-3154, Stockholm, Sweden, July 13-19, 2018. https://www.ijcai.org/proceedings/2018/437, DOI: 10.24963/ijcai.2018/437 (Acceptance rate: 20.46%)

12. Bingsheng Wang*, **Zhiqian Chen***, Arnold P. Boedihardjo, Chang-Tien Lu. Virtual Metering: An Efficient Water Disaggregation Algorithm via Non-Intrusive Load Monitoring. *ACM Transactions on Intelligent Systems and Technology* (**TIST**), Vol. 9, Issue 4, Article No. 39, February 2018. DOI: 10.1145/3141770

13. **Zhiqian Chen**, Chih-Wei Wu, Yen-Cheng Lu, Alexander Lerch, Chang-Tien Lu. Learning to Fuse Music Genres with Generative Adversarial Dual Learning. *Proceedings of the IEEE International Conference on Data Mining* (**ICDM**), pp. 817-822, New Orleans, Louisiana, Nov. 18-21, 2017. DOI: 10.1109/ICDM.2017.98 (Acceptance rate: 19.9%)

14. Xuchao Zhang, Liang Zhao, **Zhiqian Chen**, Arnold P. Boedihardjo, Chang-Tien Lu. Trendi: Tracking Stories in News and Microblogs via Emerging, Evolving and Fading Topics. *Proceedings of the IEEE International Conference on Big Data* (**Big Data**), pp. 1590-1599, Boston, MA, Dec. 11-14, 2017. DOI: 10.1109/BigData.2017.8258093

15. Xuchao Zhang, **Zhiqian Chen**, Liang Zhao, Arnold P. Boedihardjo, Chang-Tien Lu. TRACES: Generating Twitter Stories via Shared Subspace and Temporal Smoothness. Proceedings of the IEEE International Conference on Big Data (**Big Data**), pp. 1688-1693, Boston, MA, Dec. 11-14, 2017. DOI: 10.1109/BigData.2017.8258093

16. **Zhiqian Chen**, Xuchao Zhang, Arnold P. Boedihardjo, Jing Dai, Chang-Tien Lu. Multimodal Storytelling via Generative Adversarial Imitation Learning. *Proceeding of the 26th International Joint Conference on Artificial Intelligence* (**IJCAI**), Melbourne, Australia, August 19-25, 2017. DOI: 10.24963/ijcai.2017/554 (Acceptance rate: 26%)

17. Xuchao Zhang, **Zhiqian Chen**, Weisheng Zhong, Arnold P. Boedihardjo, Chang-Tien Lu. Storytelling in heterogeneous Twitter entity network based on hierarchical cluster routing. *Proceedings of the IEEE International Conference on Big Data* (**Big Data**), pp. 1522-1531, Washington, DC, Dec. 5-8, 2016. DOI: 10.1109/BigData.2016.7840760

18. Manu Shukla, **Zhiqian Chen**, Chang-Tien Lu. DIFPL: Distributed drone flight path builder system. *Proceedings of the 1st International Conference on Geographical Information Systems Theory, Applications and Management* (**GISTAM**), Barcelona, Spain, April 28-30, 2015. DOI: 10.1186/s40537-018-0134-7 (Best Student Paper Award)

19. **Zhiqian Chen**, Wenya Feng. Detecting Impolite Crawler by Using Time Series Analysis. *Proceedings of 25th IEEE International Conference on Tools with Artificial Intelligence* (**ICTAI**), pp. 123-126, Herndon, VA, USA, November 4-6, 2013. DOI 10.1109/ICTAI.2013.28

Bibliography

[1] Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. *arXiv preprint arXiv:1905.00067* (2019).

[2] Achieser, N. I. *Theory of approximation*. Courier Corporation, 2013.

[3] Achiezer, N. Theory of approximation (transl. by cj hyman), ungar, new york, 1956. *Google Scholar*.

[4] Ahlfors, L. V. Complex analysis: an introduction to the theory of analytic functions of one complex variable. *New York, London* (1953), 177.

[5] Arbelaez, A., Truchet, C., and O'Sullivan, B. Learning sequential and parallel runtime distributions for randomized algorithms. In *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on* (2016), IEEE, pp. 655–662.

[6] Argyriou, A., Micchelli, C. A., and Pontil, M. When is there a representer theorem? vector versus matrix regularizers. *Journal of Machine Learning Research 10*, Nov (2009), 2507–2529.

[7] Atwood, J., and Towsley, D. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems* (2016), pp. 1993–2001.

[8] Baldassarre, F., and Azizpour, H. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686* (2019).

[9] Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory 39*, 3 (1993), 930–945.

[10] Bell, M. G., Iida, Y., et al. Transportation network analysis.

[11] Bengio, Y., Delalleau, O., and Le Roux, N. 11 label propagation and quadratic criterion.

[12] Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional arma filters. *CoRR* (2019).

[13] Bishop, C. M., and Mitchell, T. M. Pattern recognition and machine learning.

[14] Boeing, G. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems 65* (2017), 126–139.

[15] Bollobás, B. *Extremal graph theory*. Courier Corporation, 2004.

[16] Boyd, J. P. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

[17] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine 34*, 4 (2017), 18–42.

[18] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)* (2013).

[19] Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014* (2014), pp. http–openreview.

[20] Chen, Y., Wu, L., and Zaki, M. J. Reinforcement learning based graph-to-sequence model for natural question generation. *arXiv preprint arXiv:1908.04942* (2019).

[21] Chen, Z., Chen, F., Lai, R., Zhang, X., and Lu, C.-T. Rational neural networks for approximating jump discontinuities of graph convolution operator. *ICDM* (2018).

[22] Chung, F. R. *Spectral graph theory*. No. 92. American Mathematical Soc., 1997.

[23] Cody, W. A survey of practical rational and polynomial approximation of functions. *Siam Review 12*, 3 (1970), 400–423.

[24] Cohen, H. *Numerical approximation methods*. Springer, 2011.

[25] Collobert, R., and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (2008), ACM, pp. 160–167.

[26] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (1971), ACM, pp. 151–158.

[27] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. Online passive-aggressive algorithms. *Journal of Machine Learning Research 7*, Mar (2006), 551–585.

[28] Dang, X., Peng, H., Wang, X., and Zhang, H. Theil-sen estimators in a multiple linear regression model. *Olemiss. edu* (2008).

[29] Das, K. C. The laplacian spectrum of a graph. *Computers & Mathematics with Applications 48*, 5-6 (2004), 715–724.

[30] Davidson, E. H., Rast, J. P., Oliveri, P., Ransick, A., Calestani, C., Yuh, C.-H., Minokawa, T., Amore, G., Hinman, V., Arenas-Mena, C., et al. A genomic regulatory network for development. *science 295*, 5560 (2002), 1669–1678.

[31] Davis, P. J. *Interpolation and approximation*. Courier Corporation, 1975.

[32] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems* (2016), pp. 3844–3852.

[33] Devlin, D., and O'Sullivan, B. Satisfiability as a classification problem. In *PROC. OF THE 19TH IRISH CONF. ON ARTIFICIAL INTELLIGENCE AND COGNITIVE SCIENCE* (2008).

[34] Drew, J. H., and Liu, H. Diagnosing fault patterns in telecommunication networks, Sept. 23 2008. US Patent 7,428,300.

[35] Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. Least angle regression. *The Annals of statistics 32*, 2 (2004), 407–499.

[36] Eggensperger, K., Lindauer, M., and Hutter, F. Neural networks for predicting algorithm runtime distributions. In *IJCAI* (2018), pp. 1442–1448.

[37] El Massad, M., Garg, S., and Tripunitara, M. V. Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes. In *NDSS* (2015).

[38] Eldan, R., and Shamir, O. The power of depth for feedforward neural networks. In *Conference on Learning Theory* (2016), pp. 907–940.

[39] Errica, F., Podda, M., Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893* (2019).

[40] Gagliolo, M., and Schmidhuber, J. A neural network model for inter-problem adaptive online time allocation. In *International Conference on Artificial Neural Networks* (2005), Springer, pp. 7–12.

[41] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1263–1272.

[42] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS* (2010), pp. 249–256.

[43] Graves, A., Mohamed, A.-r., and Hinton, G. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (2013), IEEE, pp. 6645–6649.

[44] Grone, R., Merris, R., and Sunder, V. S. The laplacian spectrum of a graph. *SIAM Journal on Matrix Analysis and Applications 11*, 2 (1990), 218–238.

[45] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), ACM, pp. 855–864.

[46] Guin, U., Forte, D., and Tehranipoor, M. Anti-counterfeit techniques: from design to resign. In *Microprocessor Test and Verification (MTV), 2013 14th International Workshop on* (2013), IEEE, pp. 89–94.

[47] Guo, R., Li, J., and Liu, H. Learning individual treatment effects from networked observational data. *ACM International Conference on Web Search and Data Mining* (2020).

[48] Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* (2017), pp. 1024–1034.

[49] Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[50] Hammond, D. K., Vandergheynst, P., and Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis 30*, 2 (2011), 129–150.

[51] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV* (2015), pp. 1026–1034.

[52] Helmke, U., and Williamson, R. C. Rational parametrizations of neural networks. In *NIPS* (1993), pp. 623–630.

[53] Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[54] Herbster, M., Pontil, M., and Wainer, L. Online learning over graphs. In *Proceedings of the 22nd international conference on Machine learning* (2005), ACM, pp. 305–312.

[55] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine 29* (2012).

[56] Huber, P. J. Robust statistics. In *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 1248–1251.

[57] Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization* (2011), Springer, pp. 507–523.

[58] Hutter, F., Hoos, H. H., and Leyton-Brown, K. Identifying key algorithm parameters and instance features using forward selection. In *International Conference on Learning and Intelligent Optimization* (2013), Springer, pp. 364–381.

[59] Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence 206* (2014), 79–111.

[60] Isufi, E., Loukas, A., Simonetto, A., and Leus, G. Autoregressive moving average graph filtering. *IEEE Transactions on Signal Processing 65*, 2 (Jan 2017), 274–288.

[61] Johnson, R., and Zhang, T. On the effectiveness of laplacian normalization for graph semi-supervised learning. *Journal of Machine Learning Research 8*, Jul (2007), 1489–1517.

[62] Kamali, H. M., Azar, K. Z., Gaj, K., Homayoun, H., and Sasan, A. Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection. *in IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (2018), 1–6.

[63] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[64] Khaleghi, S., and Rao, W. Hardware obfuscation using strong pufs. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (2018), IEEE, pp. 321–326.

[65] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)* (2017).

[66] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. *ICLR* (2017).

[67] Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank.

[68] Klicpera, J., Bojchevski, A., and GÃijnnemann, S. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations* (2019).

[69] Koh, P. W., and Liang, P. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* (2017).

[70] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[71] Lazer, D., Pentland, A. S., Adamic, L., Aral, S., Barabasi, A. L., Brewer, D., Christakis, N., Contractor, N., Fowler, J., Gutmann, M., et al. Life in the network: the coming age of computational social science. *Science (New York, NY) 323*, 5915 (2009), 721.

[72] Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning* (2011), Omnipress, pp. 265–272.

[73] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436.

[74] Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K., and Koh, E. Attention models in graphs: A survey. *ACM Trans. Knowl. Discov. Data 13*, 6 (Nov. 2019).

[75] Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing 67*, 1 (2018), 97–109.

[76] Levy, O., and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *NIPS* (2014), pp. 2177–2185.

[77] Levy, O., Goldberg, Y., and Dagan, I. Improving distributional similarity with lessons learned from word embeddings. *TACL 3* (2015), 211–225.

[78] Leyton-Brown, K., Nudelman, E., and Shoham, Y. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM (JACM) 56*, 4 (2009), 22.

[79] Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI* (2018).

[80] Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[81] Li, Q., Wu, X.-M., Liu, H., Zhang, X., and Guan, Z. Label efficient semi-supervised learning via graph filtering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[82] Li, Y., Xu, K. S., and Reddy, C. K. Regularized parametric regression for high-dimensional survival analysis. In *Proceedings of the 2016 SIAM International Conference on Data Mining* (2016), SIAM, pp. 765–773.

[83] Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., and Chen, E. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *IJCAI* (2015), pp. 3650–3656.

[84] Liang, S., and Srikant, R. Why deep neural networks for function approximation? In *International Conference on Learning Representations (ICLR)* (2017).

[85] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. Learning entity and relation embeddings for knowledge graph completion. In *AAAI* (2015), vol. 15, pp. 2181–2187.

[86] Liu, D., Yu, C., Zhang, X., and Holcomb, D. Oracle-guided incremental sat solving to reverse engineer camouflaged logic circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016* (2016), IEEE, pp. 433–438.

[87] Liu, X., Murata, T., Kim, K.-S., Kotarasu, C., and Zhuang, C. A general view for network embedding as matrix factorization. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (2019), pp. 375–383.

[88] Loreggia, A., Malitsky, Y., Samulowitz, H., and Saraswat, V. A. Deep learning for algorithm portfolios. In *AAAI* (2016), pp. 1280–1286.

[89] Lorentz, G. G., von Golitschek, M., and Makovoz, Y. *Constructive approximation: advanced problems*, vol. 304. Springer Berlin, 1996.

[90] Loukas, A. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199* (2019).

[91] Loukas, A., Simonetto, A., and Leus, G. Distributed autoregressive moving average graph filters. *IEEE Signal Processing Letters 22*, 11 (Nov 2015), 1931–1935.

[92] Luong, T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (2015), pp. 1412–1421.

[93] Macgregor, P. Numerical methods for scientists and engineers by h. m. antia. 404–405.

[94] MacKay, D. J. Bayesian interpolation. *Neural computation 4*, 3 (1992), 415–447.

[95] Mallat, S. G., and Zhang, Z. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing 41*, 12 (1993), 3397–3415.

[96] Markit, I. Ihs technology press release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry, 2012.

[97] Marx, V. High-throughput anatomy: charting the brain's networks. *Nature 490*, 7419 (2012), 293.

[98] Mason, J. C., and Handscomb, D. C. *Chebyshev polynomials*. CRC Press, 2002.

[99] Mayans, R. The chebyshev equioscillation theorem. *Journal of Online Mathematics and Its Applications 6* (2006).

[100] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *NIPS* (2013), pp. 3111–3119.

[101] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[102] Monti, F., Boscaini, D., and Masci, J. Geometric deep learning on graphs and manifolds using mixture model cnns.

[103] Newman, D. J., et al. Rational approximation to $|x|$. *The Michigan Mathematical Journal 11*, 1 (1964), 11–14.

[104] Newman, M. E. Spread of epidemic disease on networks. *Physical review E 66*, 1 (2002), 016128.

[105] Ng, A. Y. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning* (2004), ACM, p. 78.

[106] Oono, K., and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification.

[107] Pachon, R. *Algorithms for polynomial and rational approximation*. PhD thesis, University of Oxford, 2010.

[108] Park, C. W., and Wolverton, C. Developing an improved crystal graph convolutional neural network framework for accelerated materials discovery, 2019.

[109] Parks, T., and McClellan, J. Chebyshev approximation for nonrecursive digital filters with linear phase. *IEEE Transactions on Circuit Theory 19*, 2 (1972), 189–194.

[110] Pearl, J., et al. Causal inference in statistics: An overview. *Statistics surveys 3* (2009), 96–146.

[111] Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), ACM, pp. 701–710.

[112] Petrushev, P. P., and Popov, V. A. *Rational approximation of real functions*, vol. 28. Cambridge University Press, 2011.

[113] Poultney, C., Chopra, S., Cun, Y. L., et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems* (2007), pp. 1137–1144.

[114] Powell, M. J. D. *Approximation theory and methods*. Cambridge university press, 1981.

[115] Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (2018), pp. 459–467.

[116] Ralston, A. Rational chebyshev approximation by remes' algorithms. *Numerische Mathematik 7*, 4 (1965), 322–330.

[117] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[118] Remes, E. Sur le calcul effectif des polynomes dâĂŹapproximation de tchebichef. *CR Acad. Sci. Paris 199* (1934), 337–340.

[119] Remez, E. Y. Sur la détermination des polynômes dâĂŹapproximation de degré donnée. *Comm. Soc. Math. Kharkov 10* (1934), 41–63.

[120] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.

[121] Roshanisefat, S., Mardani Kamali, H., and Sasan, A. Srclock: Sat-resistant cyclic logic locking for protecting the hardware. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI* (2018), GLSVLSI '18.

[122] Schölkopf, B., Herbrich, R., and Smola, A. A generalized representer theorem computational learning theory ed d helmbold and b williamson, 2001.

[123] Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a SAT solver from single-bit supervision. *ArXiv abs/1802.03685* (2018).

[124] Shaham, U., Stanton, K., Li, H., Nadler, B., Basri, R., and Kluger, Y. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587* (2018).

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

[126] Shuman, D. I., Ricaud, B., and Vandergheynst, P. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis 40*, 2 (2016), 260–291.

[127] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484–489.

[128] Smith-Miles, K., and Lopes, L. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research 39*, 5 (2012), 875–889.

[129] Smola, A. J., and Schölkopf, B. A tutorial on support vector regression. *Statistics and computing 14*, 3 (2004), 199–222.

[130] Steffens, K.-G. *The history of approximation theory: from Euler to Bernstein*. Springer Science & Business Media, 2007.

[131] Stigler, S. M. Gergonne's 1815 paper on the design and analysis of polynomial regression experiments. *Historia Mathematica 1*, 4 (1974), 431–439.

[132] Subramanyan, P., Ray, S., and Malik, S. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on* (2015), IEEE, pp. 137–143.

[133] Subramanyan, P., Ray, S., and Malik, S. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on* (2015), IEEE, pp. 137–143.

[134] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *WWW* (2015), ACM.

[135] Taubin, G. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM, pp. 351–358.

[136] Telgarsky, M. Neural networks and rational functions. In *International Conference on Machine Learning* (2017), pp. 3387–3393.

[137] Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[138] Tipping, M. E. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research 1*, Jun (2001), 211–244.

[139] Trefethen, L. N. *Approximation theory and approximation practice*, vol. 128. Siam, 2013.

[140] Varga, R. S., and Carpenter, A. J. On the bernstein conjecture in approximation theory. *Constructive Approximation 1*, 1 (Dec 1985), 333–348.

[141] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[142] Wang, D., Cui, P., and Zhu, W. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), pp. 1225–1234.

[143] Wang, P., Li, Y., and Reddy, C. K. Machine learning for survival analysis: A survey. *ACM Computing Surveys (CSUR) 51*, 6 (2019), 1–36.

[144] Wang, Y., Hu, Z., Ye, Y., and Sun, Y. Demystifying graph neural network via graph filter assessment, 2020.

[145] Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[146] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International Conference on Machine Learning* (2019), pp. 6861–6871.

[147] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[148] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[149] Xie, T., and Grossman, J. C. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett. 120* (Apr 2018), 145301.

[150] Xie, Y., and Srivastava, A. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *ACM/EDAC/IEEE Design Automation Conference (DAC)* (2017), pp. 1–6.

[151] Xie, Y., and Srivastava, A. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018), 1–1.

[152] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[153] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations* (2019).

[154] Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research 32* (2008), 565–606.

[155] Yang, F., Yang, Z., and Cohen, W. W. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems* (2017), pp. 2319–2328.

[156] Yasin, M., Mazumdar, B., Rajendran, J. J. V., and Sinanoglu, O. Sarlock: Sat attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2016).

[157] Yasin, M., Rajendran, J. J., Sinanoglu, O., and Karri, R. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35*, 9 (Sept 2016), 1411–1424.

[158] Yasin, M., Sengupta, A., Nabeel, M. T., Ashraf, M., Rajendran, J. J., and Sinanoglu, O. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 1601–1618.

[159] Yeh, A. Trends in the global ic design service market. *DIGITIMES research* (2012).

[160] Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. Gnn explainer: A tool for post-hoc explanation of graph neural networks.

[161] Zamiri Azar, K., Mardani Kamali, H., Homayoun, H., and Sasan, A. SMT-attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks. In *Transaction of Cryptography Hardware and Embedded Systems* (2019).

[162] Zhang, Z., Cui, P., and Zhu, W. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* (2018).

[163] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. Learning with local and global consistency. In *Advances in neural information processing systems* (2004), pp. 321–328.

[164] Zhou, H., Jiang, R., and Kong, S. Cycsat: Sat-based attack on cyclic logic encryptions. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (Nov 2017), pp. 49–56.

[165] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[166] Zhu, X., Ghahramani, Z., and Lafferty, J. D. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)* (2003), pp. 912–919.

[167] Zhu, X., and Rabbat, M. Approximating signals supported on graphs. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on* (2012), IEEE, pp. 3921–3924.

[168] Ziegel, E. Numerical recipes: the art of scientific computing, 1987.

[169] Zou, H., and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67*, 2 (2005), 301–320.