Investigating Expressive Power Capabilities of Graph Neural Networks

Capstone Report

Baimyrza Kalmyrzayev

Nazarbayev University
Department of Electrical and Computer Engineering
School of Engineering and Digital Sciences

This project report was created on TexStudio editing platform using LaTeX. All the figures were drawn using draw.io online software tool.

NAZARBAYEV UNIVERSITY

Electrical and Computer Engineering
Nazarbayev University
http://www.nu.edu.kz

Title:
Investigating Expressive Power Capabilities of Graph Neural Networks

Theme:
Graph Neural Networks

Project Period:
Fall 2024 - Spring 2025

Project Group:
Applications of Signal Processing Laboratory

Participant(s):
Baimyrza Kalmyrzayev

Supervisor(s):
Muhammad Tahir Akhtar

Copies: 1

Page Numbers: 50

Date of Completion:
April 22, 2025

Abstract:

The Capstone project investigates the expressive power of Graph Neural Networks (GNNs) from the perspective of graph filtering, specifically focusing on group of spectral GNNs that use polynomial approximations to model graph filters to learn from graph data. These models rely on graph signal processing (GSP) techniques, which are generalizations of classical signal processing over graphs. However, polynomial filters cannot fully capture the whole frequency range. Therefore, to address this problem, autoregressive moving average (ARMA) graph filters have been studied, which are based on rational function. However, they either have larger memory requirements or higher computational complexity depending on a specific model. Thus, a novel periodic ARMA GNN (pARMA-GNN) network is proposed, which is inspired by periodic ARMA graph filters. The main advantage of the proposed model is reduced memory requirements compared to other rational function based GNNs. Experiments on node classification in semi-supervised setting and graph classification have been conducted. Results show better performance of the proposed model compared to other baseline models in heterophilic datasets, while providing competitive results in homophilic datasets. As a part of ablation study, the experiment on impact of each component of the proposed model has been done.

Contents

Preface

Emergence of Graph Neural Networks (GNNs) has led to a lot of new real-world applications ranging from analysis of social networks and recommendation systems to weather prediction. Especially, the field of medicine and biochemistry have seen huge potential in using GNN models for various applications such as drug discovery and molecular property prediction. Developing novel models in these fields can improve upon existing models to get better performance across all applications. The results of this project can be used to these aforementioned applications providing more options to choose from based on specific needs of the use case. Moreover, GNNs are recent advancements in the world, especially in Kazakhstan; thus, this project can be used as a foundation to build upon for future works in this direction. The project studies GNNs from the perspective of graph signal processing, which is a strong analytical tool for analysis of data on graphs. Moreover, GNNs can be used in combination with other popular areas such as Computer Vision and Natural Language Processing to provide good baselines to deal with graph based data.

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Muhammad Tahir Akhtar for continuous support, providing needed resources and invaluable feedbacks, which were the essential part of this research project. I am equally grateful to Professor Hashim Ali from the Department of Computer Science for their invaluable co-supervision of this project and expertise in machine learning and willingness to provide additional perspective have contributed substantially to this work. I also appreciate my fellow research group members for their constructive discussions on the topic during weekly meetings. Finally, I would like to express my heartfelt gratitude to my mother, who has been supporting me throughout my life to achieve my goals.

Nazarbayev University, April 22, 2025

Baimyrza Kalmyrzayev
<baimyrza.kalmyrzayev@nu.edu.kz>

Chapter 1

Introduction

1.1 Background

Graph Neural Networks (GNNs) are the deep learning models that leverage graph data. They have become popular due to the availability of datasets with relational information, which can allow to create models with larger inductive bias represented in form of edges between nodes. The inductive bias can be incorporated into the models, which allows them to learn specific structures and patterns of the data such as cycles, motifs, etc. There are various applications of GNNs in different areas, including biochemistry and healthcare such as protein folding [1] and road networks such as transportation optimization in maps [2], etc.

The first GNN models have developed as an extension of recurrent neural networks, which is limited to only directed and acyclic graphs, to be able to process various types of graphs [3], [4]. Currently, GNNs are considered to be under Message Passing Neural Networks framework, which operates with message and readout functions to propagate information through a graph [5]. These models allow to extract important information without relying on preprocessing stage, which is the main issue with traditional machine learning approaches [6].

The theoretical fundamentals of GNNs can be described by Graph Signal Processing (GSP), which is an extension of classical signal processing to analyze signals defined on vertices of a graph or a network [7]. It allows to analyze graph signal in both vertex domain and spectral domain. Node space of a graph is suitable to discover local structures and patterns around a neighborhood of an interest node. Whereas, spectral domain provides global information about the smoothness or sharpness of all frequency components. Moreover, the analysis of spectral domain allows to model graph filters with desired properties by applying a certain type of functions.

GNNs emerge from the fundamental need to process graph-structured data (non-Euclidean). Graphs provide very strong abstraction for representing various

relationships between entities. The theory of GNNs are based on different disciplines such as graph theory, spectral theory and graph signal processing [6], [8].

An undirected graph is defined as $G = (\mathcal{V}, \mathcal{E})$ consisting of set of nodes $\mathcal{V} = \{\cdots, u_i, \cdots\}$ with $N = |\mathcal{V}|$ number of nodes and set of edges $\mathcal{E} = \{\cdots, (u_i, u_j), \cdots\}$. Nodes represent entities, while edges represent a relationship between these entities. The complex structure of graph-based data does not allow traditional machine learning models to fully capture hidden patterns; therefore, GNNs are specifically designed to handle such irregular data.

The core computational mechanism of GNNs is Message Passing framework, which allows to pass messages between neighboring nodes through recursive aggregation process [5]. This process is defined as

$$\boldsymbol{x}_i^{(t+1)} = \text{UPDATE}(\boldsymbol{x}_i^{(t)}, \text{AGGREGATE}(\{\boldsymbol{x}_j^{(t)} : u_j \in \mathcal{N}(u_i)\})) \tag{1.1}$$

where $\boldsymbol{x}_i^{(k)}$ is feature vector of node u_i, $\mathcal{N}(u_i)$ is a set of neighbors of node u_i, and UPDATE and AGGREGATE are learnable neural network functions. Fig. 1.1(a) depicts arbitrary undirected graph containing 6 nodes with colors indicating different feature values for each node. Fig. 1.1(b) shows the message passing mechanism for 2 layers, where each node receives aggregated information from its neighbors and then passes its own state to its neighbors. In this example, node 3 receives information from the whole network in 2 steps.

GNNs have shown considerable performance across various domains such as:

1. Molecular Property Prediction: drug discovery based on predicting molecular characteristics [9, 10].

2. Recommender Systems: user product interaction patterns [11].

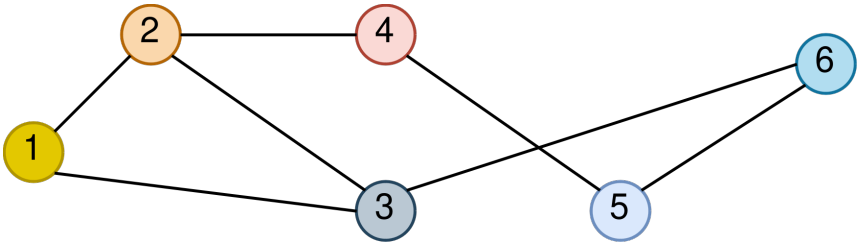
3. Social Network Analysis: user behavior patterns, community detection, influence propagation [12].

Even though GNNs have advanced significantly, there are several challenges in the field:

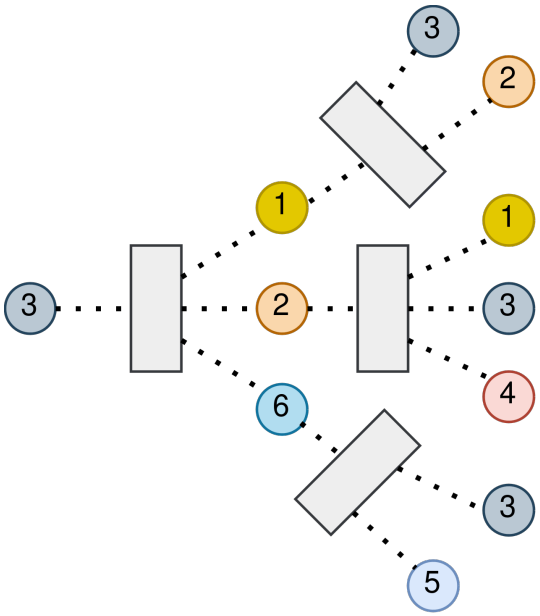
1. Over-smoothing problem: Increased depth of GNN models result in similar node representations making it hard to differentiate between nodes [13].

2. Interpretability: Understanding complex decision-making processes in GNNs.

3. Scalability: Computational complexity increases exponentially with graph size.



(a)



(b)

Figure 1.1: (a) An undirected graph with 6 nodes, (b) Message Passing mechanism of 2-hop neighborhood for node 3.

1.2 Literature Review

GNNs are generally divided into spatial and spectral models depending on the driving mechanism behind the model. Spatial GNNs use graph topology by aggregating and updating the information of a node from its neighborhood. Such simple implementation allows to apply various methods to spatial GNNs such as using attention mechanism [14, 15], defining new sampling techniques to improve model scalability to large graphs [16, 17], and adding additional features to increase model expressivity [18, 19, 20]. The expressive power of spatial GNNs mainly refers to what extent models can differentiate between two given graphs. Weisfeiler-Lehman (WL) graph isomorphism test, mostly named as 1-WL test, is used to define the expressive power of models [21], which is necessary, but not

sufficient condition for graphs to be isomorphic. The algorithm relies on iterative color-encoding technique, where each node is assigned initial color value as its state. Then, a node's state is updated by aggregating the states of its neighbors and using hash function to create new state of a node. This process is continued until the limit of iterations reached or nodes are in steady state. There are also researches that studied expressive power from different perspectives such as counting substructures [22], k-regular graphs [20], etc. Current spatial models surpassed 1-WL expressive test by using various methods such as positional encoding [19], higher-order relationships between nodes [23].

Spectral GNNs rely on spectral theory and may require eigenvalue decomposition of graph shift operator [24], which can be defined by graph Laplacian $\boldsymbol{L}$. The advantage of spectral models is that they can capture global information and are not restricted to local neighborhood due to eigenvalues and eigenvectors of a graph. The expressive power of spectral GNNs are evaluated from GSP perspective, specifically which frequency ranges they can capture [25]. This comes from the eigenvalue decomposition of graph Laplacian, which provides eigenvalues of graph topology that correspond to frequencies of a graph signal. Therefore, spectral methods mainly focus on developing models that can capture most important frequency information through graph filters. These models can be divided into three groups [26]: advanced filters that directly use eigenvalues, polynomial filters that use polynomial approximations, and linear filters that consider only linear relationship.

The main limitation of advanced filters is that they need eivenvalues as inputs that requires to compute eigenvalue decomposition, which is computationally expensive operations, especially for large graphs. One of the first GNN architectures that generalized convolution operator over graphs, Spectral Convolutional Neural Network (SpectralCNN) [24], has addressed this problem by truncating spectrum of a graph by considering only first k frequency components. However, the use of advanced graph filters in GNNs has been declined due to high complexity compared to other types of spectral filters and still remains as an open problem in the field.

Polynomial models does not rely directly on eigenvalues of a graph; thus, they do not require eigenvalue decomposition compared to advanced graph filters. Moreover, it has been found that they can be expressive as much as 3-WL test [27]. These models use K-order polynomial approximations as a basis leading to possible limited capabilities as reaching to larger neighborhood leads to exponential computational complexity. One of the first works in this direction is Chebyshev Network (ChebNet) [28], which is based on K-th order Chebyshev polynomials. ChebNet's expressive power and computational complexity directly depends on polynomial order K, which at higher orders can capture any arbitrary frequency component, but requires computing higher power polynomials. Despite theoretical

soundness, it has been found that ChebNet does not perform at expected level and can provide worse performance than linear models. ChebNetII [29] has been proposed to address this problem by applying reparameterization to the weight coefficients using Chebyshev interpolation. The main advantage of ChebNetII is solving ill-defined filter coefficient problem without increasing computational complexity of the model. Bernstein Network (BernNet) [30] uses Bernstein polynomials as a basis in constructing a graph filter. BernNet has also addressed the problem of ill-defined coefficients by using non-negative filter coefficients. The model can approximate any arbitrary spectral filters, even the complex ones such as comb and band-rejection filters. Jacobi Convolutional Network (JacobiConv) [31] is another spectral model based on Jacobi polynomials. JacobiConv has orthogonal bases and is a generalized version of other spectral models such as ChebNet, which means it is more expressive at the expense of higher computational complexity. Generalized PageRank GNN (GPR-GNN) [32] uses learnable weight coefficients for each propagation step, which corresponds to K-th order graph filter. This method allows the model to capture local structures at initial propagation steps and global relationship at later steps, while the model parameters will learn which steps are more important for a given task. Spectrum Graph Network (Spec-GN) [33] has proposed correlation-free model that can control the scale of spectrum and use generalized version of normalized adjacency matrix. This approach allows to create deeper networks providing a solution to over-smoothing problem, which is a widely known limitation of GNNs [13], where deep networks lead to similar indiscriminate features for nodes of all classes.

Linear models are the fastest ones due to their simpler design that use only linear function in each update step. However, as a result of such simplification, they suffer from poor performance on disassortative or heterophilic datasets, graphs where nodes with dissimilar features or different classes tend to connect with each other. The most well-known GNN model, Graph Convolutional Network (GCN) [34], is a linear model that uses first-order Chebyshev polynomial. GCN is well suited to assortative or homophilic graphs, where nodes with similar features or the same class tend to connect with each other, due to their update function that smooth out neighborhood information. However, it has been shown several times that GCN corresponds to low-pass filter [35, 25, 36], which makes it a bad choice for graph data with sharp changes along the neighboring nodes. Some work has been done to further amplify low-pass filtering ability of GCN by removing intermediate non-linearity and feature transformation steps such as Simple Graph Convolution (SGC) [35] and Graph Filters Neural Network (gfNN) [36]. There has been an attempt to improve the expressive power of linear models by applying different connectivity matrix to capture high-frequency components. For example, Frequency Adaptation GCN (FAGCN) [37] has incorporated attention mechanism via self-gating function, which learns the proportion of low-pass and high-pass

components in a graph structure. In the case of Adaptive GNN (AdaGNN) [38], authors have proposed deriving high-pass filtering through stacking several low-pass filters in addition to diagonal weight matrix that responsible for scaling each feature dimension. Adaptive Kernel GNN (AKGNN) [39] has proposed to a model that adaptively learns the weighted combination of all-pass and low-pass filters, which has been derived from the update function of GCN. Despite many attempts to make linear models to learn arbitrary frequency ranges of the underlying graph structure, they still cannot outperform polynomial or advanced filters on these sort of datasets.

1.3 Related Works

Polynomial models can be seen as a best choice for designing new spectral GNN models because of their ability to trade-off between the expressive power and computational complexity compared to advanced and linear filters. These models can still capture in theory arbitrary frequency components while not directly using computationally expensive operations such as eigenvalue decomposition. However, polynomial models cannot fully model sharp changes and require higher degree to reach larger neighborhood, which increases their computational complexity significantly. To address these problems, several research works have been conducted in the direction of rational based functions, specifically using Autoregressive Moving Average (ARMA) filters family. Due to their rational function, ARMA-based models can capture various frequency components, including high-pass and band-pass, using smaller degree polynomial. Two representatives of rational function based spectral GNNs are Cayley Network (CayleyNet) [40] and ARMA Convolutional Network (ARMAConv) [41].

CayleyNet [40] uses rational complex function to model narrow-bands more efficiently than previous spectral models. Apart from that, it uses spectral zooming parameter to control the spread of frequency spectrum, which is similar idea to one from Spec-GN [33]. The downside of CayleyNet is huge computational complexity, which may be several magnitudes higher than other spectral models.

ARMAConv [41] is based on first-order ARMA ($ARMA_1$) filters that are stacked in parallel to obtain various filtering behaviors. The main parameters of ARMAConv are number of stacked $ARMA_1$ filters K and the depth of layer T. The model can efficiently filter different frequency ranges and computationally efficient compared to CayleyNet. However, ARMAConv may require too much memory as each $ARMA_1$ filter has its own two sets of weight matrices. In certain cases, for example large graphs that require comparatively complex neural network, ARMAConv can be memory-intensive.

The afore-mentioned problems can be resolved using alternative to parallel $ARMA_1$ filters, specifically periodic $ARMA_1$ filters that utilize periodic weights

[42]. Such design choice allows to have the main strength of ARMAConv model, the ability to filter arbitrary frequency ranges, while lowering memory requirements of the model. Another design choice is to retain all-pass and high-pass components of original $ARMA_1$ filters, which has been coupled into one term in ARMAConv model. The rationale behind this decision is that stacking low-pass filters many times still have a smoothing effect, which must be avoided for heterophilic graphs. Thus, a novel periodic ARMA GNN (pARMA-GNN) model is proposed, which has capability to filter any frequency range with reduced memory requirements compared to existing ARMA-based spectral GNNs. Moreover, several modifications of pARMA-GNN are developed based on the type of given dataset, which further improves memory efficiency.

1.4 Motivation and Problem Statement

Modern popular machine learning models from Computer Vision (CV) and Natural Language Processing (NLP) are not suitable to work with graph data. However, most of the real-world data is mainly presented in a relational matter in a form of graph. As a result, most of the existing machine learning models are not able to fully capture the important patterns of graph data. In contrast, graph machine learning algorithms can be used to other forms of data such as image or text by converting them into graphs. Therefore, the main motivation of this Capstone project is to develop a GNN model, with several advantages over existing models providing a new state-of-the-art graph machine learning model to work with graph structured datasets.

1.5 Ethical and Professional Responsibilities

- **Ethical Responsibility:** One of the main ethical issues in machine learning models is biased data, which in turn leads to biased outputs. Models trained on such datasets can be used to misinform people or manipulate their opinion. Examples might be discrimination in social networks and recommendation systems that are based on graph neural networks.

  Another issue to consider is data privacy. In real-world applications, graph neural networks use interconnected data such as from social networks and healthcare, which can also include personal information of users. These case may occur due to the fact that graph neural networks heavily rely on relational information from node attributes and edge connections, which can introduce bias to some extent if these data are corrupted. Therefore, it is important to ensure data anonymity to protect users' privacy. Unprotected data might be leaked and further be used in fraudulent activities.

  To solve the problem with biased data, standardized datasets can be used in model training. These datasets are benchmarks to evaluate the performance of a model. The problems and flaws of these datasets have been researched previously and can be controlled. In the case of data privacy, clear guidelines must be established on using the developed model. Also, it is crucial to discuss the capabilities and limitations of the model by providing all relevant information and results of experiments.

- **Informed Judgments:** To make well-informed decisions during the project, two separate aspects must be considered, namely technical and societal. From a technical standpoint, it is important to stay up-to-date with recent studies on graph neural networks, graph signal processing and adjacent fields that apply aforementioned areas in their research. This requires regular literature review, experimenting with alternative approaches for the same problem to find a better solution. Also, it is beneficial to discuss the arisen problems with other researchers and experts in the field. One part of such discussions may include model scalability or computational complexity as both of these things have significant impact on the overall performance of the model.

  Nevertheless, considering only technical aspects is not enough to make fully well-informed decisions. There is also a societal impact that derives from technical decisions. This includes possible applications of graph neural networks, specifically which domains can use these models, what are the reasons to use them and how it will affect society. These applications may also include harmful activities, which must be addressed through continuous self-assessment and standard precautions. Thus, collaborating with researchers from areas such as ethics, sociology can be a possible solution to address

these questions.

- **Global Context:** The development of machine learning models have global implications because of interconnected relationships between countries. First of all, developed countries with better technological advancements can easily develop and deploy these models for various use cases such as weather forecasting, social network analysis, traffic monitoring or urban planning. In contrast, countries with limited computation resources or data availability might not be able to benefit from these models to the same extent as developed countries.

  Secondly, depending on the position of a country on using personal information, the rate of adoption of these models may vary. For instance, using graph neural networks for social network analysis may face challenges in societies with strict privacy regulations compared to those with less restrictive laws. Therefore, the data usage of the model must follow the laws of each individual country to avoid law violations. In contrast, these models can potentially lead to misuse or even ethical violations in regions with less strict regulations on personal data. Moreover, acceptance of machine learning models in certain sectors may take time depending on the legislative regulations of a country. This potentially can affect the effectiveness of existing technologies and development of new ones. Thus, it is aimed to develop an adaptable model which can be appropriately adjusted based on the needs of specific regions.

- **Economic Impact:** Short-term economic effects of developing such models mainly concentrated around research and development costs. These include financial investments into computation resources such as graphical processing units (GPUs) for training, data collection and team of researchers. However, most of the time these costs can be compensated by industry companies that are interested in potential applications of the given project.

  Long-term economic effects of the project can be considered as quite promising. Successfully developed and deployed graph neural network models can enable significant efficiency improvements in different fields. For example, graph neural networks can be applied to logistics networks by optimizing the operational costs. In the case of social networks, these models can be used to detect fraudulent activities and potential harmful actions by criminals.

  The usage of these technologies could also lead to innovations by creating new business models that were not possible before. Most known example of such a case is advancements of language models leading to a lot of new products using Large Language Models (LLM). This could contribute to economic growth and global competition between countries. Nevertheless, the major

concern regarding the economic impact of neural networks is the automation of certain jobs. There might be displacement of jobs leading to people losing their jobs.

- **Environmental Impact:** From the perspective of environmental impact, the main problem of the project is using computational resources for training and operating, which directly translates to energy consumption. Specifically, large-scale graphs require extensive amount of compute. This could contribute to increased carbon emissions if the energy sources are not renewable.

  Nevertheless, the potential environmental benefits of using these models can outweigh its drawbacks. Applying these models in certain areas can help to optimize existing processes and reduce wastes. For example, these models can help to improve Geographic Information Systems (GIS) by analyzing road traffics and enabling drivers to avoid traffic jams, which results in less carbon emissions from vehicles. In environmental science, currently used models in weather forecasting can be adopted to predict natural disasters and handle them.

  There are several measurements that can be taken to minimize the environmental harm of the project. First of all, more efficient and optimized algorithms will be used to reduce computational requirements of developing. Second of all, cloud service computing will be used that are much more environmentally friendly. Additionally, the research on less complex models with the same or even better expressive power than current state-of-the-art models can indirectly impact on reducing the need for more complex models, which indirectly affects environmental sustainability.

- **Societal Impact:** The most direct way of the project to impact society beneficially is in the healthcare sector. These models can be applied to drug discovery by using molecular information of chemical compounds and their properties, disease spread prediction by analyzing networks of social interactions. This could lead to a more efficient healthcare system allowing to save the lives of millions of people.

  They can be applied to social networks to improve community detection demographic analysis to enable more efficient communication strategies with these groups. This can help to support vulnerable groups of people.

  In the case of urban planning, graph neural networks can help to the development of smarter cities by optimizing the routes of public transportation, predicting maintenance needs for infrastructure. They can also help the energy distribution within cities.

  Specific indirect effects of such models can be on social structures and interactions. As these models become more widely used in online social network

analysis, they can potentially change how these online platforms operate. Such changes can influence information flow and social dynamics between communities. Moreover, these models can be misused in surveillance or personal data gathering, which must be carefully monitored and controlled. One way to prevent such course of actions is to focus on open-source contributions to enable other researchers and engineers to check for flaws in the models.

1.6 Report Organization

The Capstone Report is organized as follows. Section 2 provides a brief overview of graph signal processing, which serves as the foundation for developed methodology. Section 3 outlines the proposed methodology and presents an analysis of the model's frequency response. Section 4 discusses the experimental results. Finally, Section 5 concludes the report and highlights the potential directions for future work.

Chapter 2

Brief Overview of Graph Signal Processing

$\boldsymbol{X} \in \mathbb{R}^{N \times F}$ denotes a graph signal and the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ defines connections of nodes, where F is a dimension of the feature vector. Given that normalized adjacency matrix is defined as $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, let $\boldsymbol{L}$ denotes the normalized graph Laplacian given as:

$$\boldsymbol{L} = \boldsymbol{I} - \tilde{\boldsymbol{A}}, \tag{2.1}$$

where $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is an identity matrix and $\boldsymbol{D} \in \mathbb{R}^{N \times N}$ is a diagonal degree matrix defined as $D_{ii} = \sum_j A_{ij}$. The graph Laplacian has the eigenvalue decomposition given as:

$$\boldsymbol{L} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T, \tag{2.2}$$

where $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_N]$ is a matrix of eigenvectors and $\boldsymbol{\Lambda} = \text{diag}\{\lambda_1, \lambda_2, \cdots, \lambda_N\}$ is a diagonal matrix of eigenvalues. Irregular structure of graphs prohibits the use of convolution in node domain. Thus, convolution of one-channeled graph signal $\boldsymbol{x} \in \mathbb{R}^N$ and graph filter $\boldsymbol{g} \in \mathbb{R}^N$ is defined using graph Fourier transform on spectral domain as:

$$\boldsymbol{g} \circledcirc \boldsymbol{x} = \boldsymbol{\Phi}\big((\boldsymbol{\Phi}^T\boldsymbol{g}) \odot (\boldsymbol{\Phi}^T\boldsymbol{x})\big), \tag{2.3}$$

where $\circledcirc$ is a graph convolution defined on node space and $\odot$ is element-wise multiplication. Spectral filters replace graph Fourier transform of graph filter $\boldsymbol{\Phi}^T\boldsymbol{g}$ with parametrized diagonal matrix $\boldsymbol{G} = \text{diag}\{\theta_1, \theta_2, \cdots, \theta_N\}$. Thus, (2.3) can be rewritten as:

$$\boldsymbol{g} \circledcirc \boldsymbol{x} = \boldsymbol{\Phi}\boldsymbol{G}\boldsymbol{\Phi}^T\boldsymbol{x}. \tag{2.4}$$

The above formulation of graph convolution has led to development of numerous models. However, the main disadvantage of this method is computational complexity. The computation of eigenvalue decomposition is inefficient, especially for large graphs. Moreover, this method is not applicable to dynamic graphs that change over time. This is because small alteration in graph structure can completely change its spectral information leading to computing eigenvalue decomposition again. Thus, models based on (2.4) are suitable only for static graphs, which are often not a case for real-world applications.

Chapter 3

Proposed Methodology

The general formula for the frequency response of ARMA filters is given as:

$$H_K(\lambda) = \frac{\sum_{k=0}^{K-1} a_k \lambda^k}{1 + \sum_{k=1}^{K-1} b_k \lambda^k}, \tag{3.1}$$

where a_k is k-th moving average coefficient, b_k is k-th autoregressive coefficient, and K is the filter order. The rational function of ARMA model is well suited for designing graph filters that can handle arbitrary frequency ranges depending on the application. However, the given rational function involves computing inverse matrix when the given input and weight parameters of ARMA filter are matrices. Finding the inverse of a matrix is computationally expensive; thus, there have been proposed an iterative equation that models ARMA_1 filters given as [43]:

$$\boldsymbol{y}^{(t+1)} = \psi \boldsymbol{S} \boldsymbol{y}^{(t)} + \varphi \boldsymbol{x}, \tag{3.2}$$

where $y^{(t)}$ is an output of the filter or autoregressive output at time t, $\boldsymbol{S}$ is a graph shift operator that is a general form of connectivity matrix, and ψ and φ are scalar filter coefficients.

The frequency response of the model can be greatly improved by stacking several ARMA_1 filters; however, it costs much more memory. This problem has been addressed by introducing periodic weights into stacked ARMA_1 filters (ARMA_K) defined as:

$$\boldsymbol{y}^{(t+1)} = (\gamma^{(t)} \boldsymbol{I} + \psi^{(t)} \boldsymbol{S}) \boldsymbol{y} + \varphi^{(t)} \boldsymbol{x}, \tag{3.3}$$

where $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is an identity matrix and scalar parameters $\gamma^{(t)}$, $\psi^{(t)}$, $\varphi^{(t)}$ are periodic with period P as $\gamma^{(t)} = \gamma^{(t-iP)}$, $\psi^{(t)} = \psi^{(t-iP)}$, $\varphi^{(t)} = \varphi^{(t-iP)}$ with $i \in \mathbb{N}$.

Due to periodicity, the filter output $\boldsymbol{y}^{(t)}$ is only valid after each P iterations, when the input goes through the full cycle of weights.

Spectral GNN based on periodic ARMA$_K$ filters can be constructed by reformulating (3.3) into GNN setting, where parameters are learned by directly using available training dataset instead of using optimization techniques. Thus, a novel periodic ARMA GNN (pARMA-GNN) is proposed, which has capabilities to filter any arbitrary frequency components and it is architecture is shown in Figure 3.1. The proposed pARMA-GNN consists of M periodic ARMA Convolutional (pARMAConv) layers. The main two parameters of pARMA-GNN are period P and timestamps T, which is a number of full cycles in one forward pass of the model. Therefore, in each iteration, the input is propagated $T \cdot P$ times. Each pARMAConv layer consists of linear layer, which use case is explained in the following section and pARMAConv propagation steps, where activation function and dropout are applied after each propagation step.

3.1 Periodic ARMA Graph Neural Network with Scalar Weights

The first version of pARMA-GNN is defined using scalar weight parameters, which exactly follows (3.3). The convolution layer of pARMA-GNN scalar (pARMA-GNN-s) is defined as:

$$\boldsymbol{Y}^{(t+1)} = \sigma(w_I^{(t)}\boldsymbol{Y}^{(t)} + w_L^{(t)}\boldsymbol{L}\boldsymbol{Y}^{(t)} + v^{(t)}\boldsymbol{X}), \tag{3.4}$$

where $\boldsymbol{Y}^{(t)}$ is autoregressive model output at time t, $w_I^{(t)}$ is a scalar periodic identity autoregressive weight parameter at time t, $w_L^{(t)}$ is a scalar periodic Laplacian autoregressive weight parameter at time t, $v^{(t)}$ is a scalar periodic moving-average weight parameter at time t, and σ is a non-linear activation function such as parametric rectified linear unit (PReLU). The graph Laplacian L from (2.1) has been selected as a graph shift operator. The rationale behind this choice is filtering mechanism of the graph Laplacian. The identity part in (3.4) correspond to all-pass filter, while the graph Laplacian correspond to high-pass filter because it captures the differences between a node's signal value and those of its neighbors. Therefore, different filtering behaviors can be obtained, such as band-pass or high-pass, by periodically applying the given convolution layer. Proofs of these claims are given in the section on frequency analysis.

The initial autoregressive state $\boldsymbol{Y}^{(0)}$ is usually not available from the dataset. Thus, it can either be set to zero or computed from input matrix $\boldsymbol{X}$ by applying linear layer $\boldsymbol{Q}$ as $\boldsymbol{Y}^{(0)} = \boldsymbol{X}\boldsymbol{Q}$, where $\boldsymbol{Q}$. However, this approach increases memory requirements of the model as matrix $\boldsymbol{Q}$ is not fixed and also learned during training. In Figure 3.2, the left diagram is the implementation of pARMAConv layer for pARMA-GNN-s model. It requires initial autoregressive output, feature input

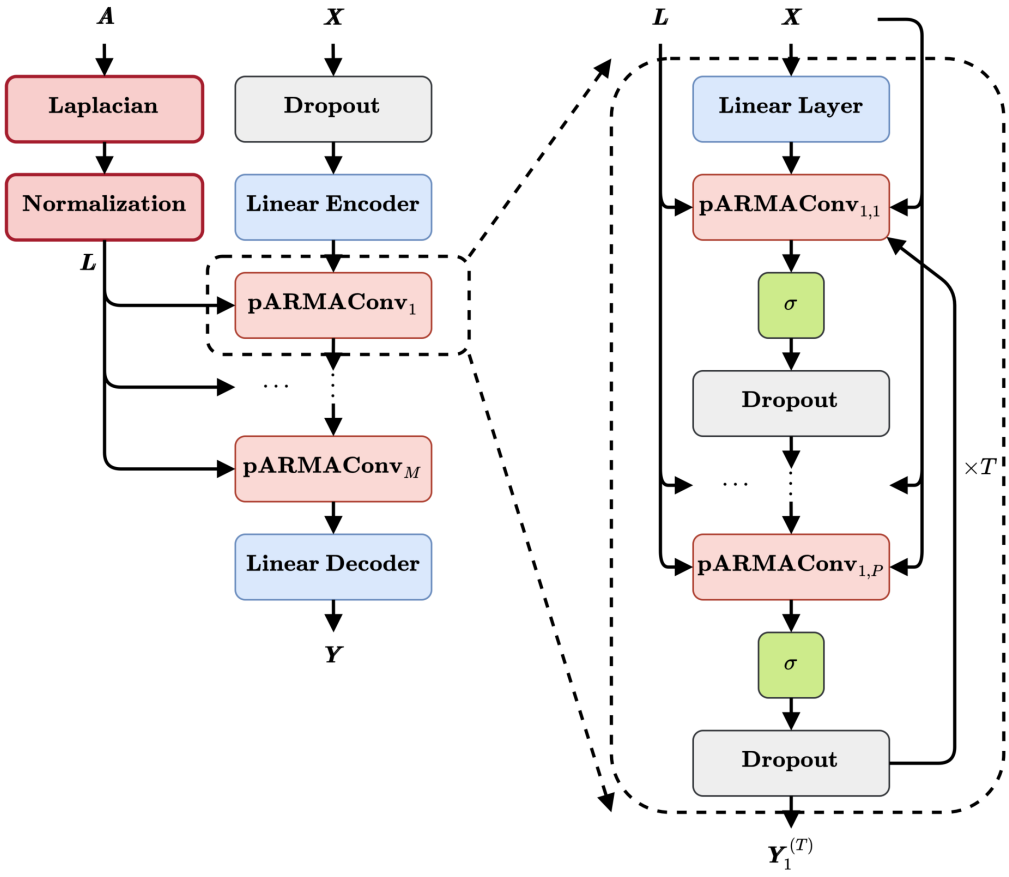


Figure 3.1: The general block diagram of the proposed pARMA-GNN architecture. The left block diagram shows information flow of the model consisting of linear encoder, M pARMAConv layers and linear decoder. The right block diagram shows inner block diagram of first pARMAConv layer consisting of P periodic propagation T times.

and normalized graph Laplacian for propagation. The bias term is included as additional feature.

3.2 Periodic ARMA Graph Neural Network with Matrix Weights

The second version of the proposed pARMA-GNN uses weight matrices instead of scalars. The convolution layer of pARMA-GNN matrix (pARMA-GNN-m) is given as:

$$\boldsymbol{Y}^{(t+1)} = \sigma(\boldsymbol{Y}^{(t)}\boldsymbol{W}_I^{(t)} + \boldsymbol{L}\boldsymbol{Y}^{(t)}\boldsymbol{W}_L^{(t)} + \boldsymbol{X}\boldsymbol{V}^{(t)}), \tag{3.5}$$

where $\boldsymbol{W}_I^{(t)}$ is a periodic identity autoregressive matrix weight parameter at time t, $\boldsymbol{W}_L^{(t)}$ is a periodic Laplacian autoregressive matrix weight parameter at time t, and

$\boldsymbol{V}^{(t)}$ is a periodic moving-average matrix weight parameter at time t.

The main reason for proposing matrix weighted version of pARMA-GNN is improving expressive power of the model. Scalar weights are very memory-efficient because there are only 3 parameters per pARMAConv layer; however, they apply the same filter strength to all feature dimensions and cannot learn complex interactions between feature channels. In this case, uniform filtering leads to limited expressiveness of the model. In contrast, pARMA-GNN-m has higher modeling capacity as it applies filters across feature dimensions at the expense of higher memory requirements. From the neural networks perspective, both pARMA-GNN-s and pARMA-GNN-m have their own use cases. For example, if the size of data is comparatively small, pARMA-GNN-s is expected to be better choice because it can potentially learn all important patterns in the data without spending too much memory resources. Whereas, pARMA-GNN-m is more well-suited for large datasets as its matrix weight parameters allow it to learn more complex relationships between features. Moreover, pARMA-GNN-m may overfit in small datasets due to large number of parameters losing its generalization power.

3.3 Modifications to the Periodic ARMA Graph Neural Network

Two modifications of the proposed pARMA-GNN have been developed to further improve its efficiency performance-wise and memory-wise.

To improve the adaptivity of pARMA-GNN, a periodic learnable parameter $\alpha^{(t)}$ is introduced, which controls the spectral filtering behavior of the model by balancing between identity and the graph Laplacian autoregressive terms. The update equation of adaptive pARMA-GNN (Ada-pARMA-GNN) for scalar weighted version is given as:

$$\boldsymbol{Y}^{(t+1)} = \sigma(\alpha^{(t)} w_I^{(t)} \boldsymbol{Y}^{(t)} + (1-\alpha^{(t)}) w_L^{(t)} \boldsymbol{L}\boldsymbol{Y}^{(t)} + v^{(t)} \boldsymbol{X}). \tag{3.6}$$

The same equation for matrix weighted pARMA-GNN is given as:

$$\boldsymbol{Y}^{(t+1)} = \sigma(\alpha^{(t)} \boldsymbol{Y}^{(t)} \boldsymbol{W}_I^{(t)} + (1-\alpha^{(t)}) \boldsymbol{L}\boldsymbol{Y}^{(t)} \boldsymbol{W}_L^{(t)} + \boldsymbol{X}\boldsymbol{V}^{(t)}), \tag{3.7}$$

Using adaptive parameter $\alpha^{(t)}$ helps the model to emphasize a specific filtering behavior depending on the underlying structure of the dataset. Specific application of Ada-pARMA-GNN is shown in section 4. In Figure 3.2, the middle block diagram is illustration of Ada-pARMAConv layer of Ada-pARMA-GNN-s model. It has additional weighting periodic coefficient $\alpha^{(t)}$ that improves the adaptability of autoregressive terms.

The memory requirements of the proposed pARMA-GNN can be further decreased by implying symmetric assumption on autoregressive weight parameters.

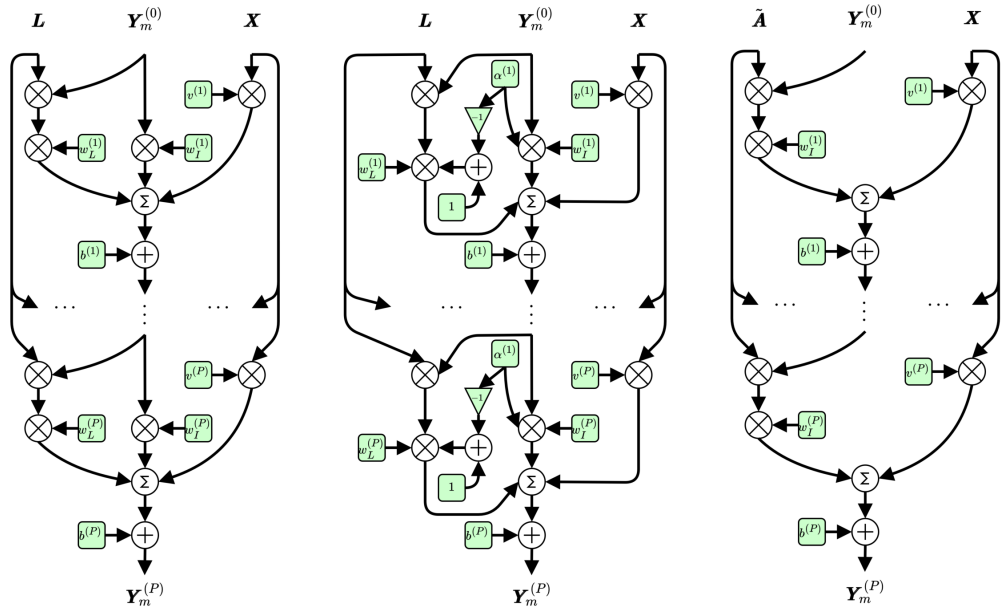


Figure 3.2: Block diagrams of pARMAConv layers for pARMA-GNN architecture in 1 timestamp at layer m. The left block diagram is original pARMAConv layer, the middle block diagram is Ada-pARMAConv layer that includes additional periodic weighting parameter $\alpha^{(t)}$, and the right block diagram is Simple-pARMAConv layer that uses normalized adjacency matrix and only one autoregressive weight.

Specifically, scalar weights can be set as $w_L^{(t)} = -w_I^{(t)}$ and matrix weights can be set as $\boldsymbol{W}_L^{(t)} = -\boldsymbol{W}_I^{(t)}$, which simplifies (3.4) and (3.5) into:

$$\boldsymbol{Y}^{(t+1)} = \sigma(w_I^{(t)} \tilde{\boldsymbol{A}} \boldsymbol{Y}^{(t)} + v^{(t)} \boldsymbol{X}) \tag{3.8}$$

and

$$\boldsymbol{Y}^{(t+1)} = \sigma(\tilde{\boldsymbol{A}} \boldsymbol{Y}^{(t)} \boldsymbol{W}_I^{(t)} + \boldsymbol{X} \boldsymbol{V}^{(t)}), \tag{3.9}$$

respectively. Simplified pARMA-GNN (Simple-pARMA-GNN) offers low-pass filtering by assuming normalized adjacency matrix $\tilde{\boldsymbol{A}}$ as the graph shift operator in its propagation steps. In theory, the expressive power of Simple-pARMA-GNN is lower than pARMA-GNN and Ada-pARMA-GNN; however, Simple-pARMA-GNN is less memory-intensive and can be used as an alternative to other spectral GNNs with low-pass filtering mechanism such as GCN. In Figure 3.2, the right block diagram provides the implementation of Simple-pARMAConv layer of Simple-pARMA-GNN-s model. As can be seen, the model has only one autoregressive term and replaces normalized graph Laplacian with normalized adjacency matrix.

3.4 Relationship between Periodic ARMA Graph Neural Network and Other Spectral Graph Neural Networks

Due to its recursive equation, pARMA-GNN can be seen as a generalization of several other spectral models. First of all, pARMA-GNN can approximate GNNs based on polynomials such as ChebNet because such models use order K finite polynomials.

In the case of GCN, it can be directly derived from pARMA-GNN under certain conditions. Specifically, when $P = 1$, $V^{(t)} = \mathbf{0}$ and $\boldsymbol{W}_L^{(t)} = -\boldsymbol{W}_I^{(t)}$, (3.5) simplifies to:

$$\boldsymbol{Y}^{(t+1)} = \sigma(\tilde{\boldsymbol{A}}\boldsymbol{Y}^{(t)}\boldsymbol{W}_I). \tag{3.10}$$

Taking into account that initial autoregressive weight can be defined by linear transformation of input matrix, (3.10) at step $t = 1$ can be rewritten as:

$$\boldsymbol{Y}^{(1)} = \sigma(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{Q}\boldsymbol{W}_I), \tag{3.11}$$

which corresponds to update equation of GCN with linearly transformed input matrix.

3.5 Frequency Response Analysis of the Periodic ARMA Graph Neural Network

In this section, frequency response of the proposed pARMA-GNN-s is derived and stability condition is provided. The two assumptions are needed for frequency response derivation:

- Initial autoregressive state $\boldsymbol{Y}^{(0)}$ is arbitrary;
- Non-linear activation function set to identity as $\sigma(\boldsymbol{x}) = \boldsymbol{x}$.

Then, first two iterations of recursion (3.4) can be calculated as:

$$\begin{aligned}\boldsymbol{Y}^{(1)} &= (w_I^{(0)}\boldsymbol{I} + w_L^{(0)}\boldsymbol{L})\boldsymbol{Y}^{(0)} + v^{(0)}\boldsymbol{X} \\ \boldsymbol{Y}^{(2)} &= (w_I^{(1)}\boldsymbol{I} + w_L^{(1)}\boldsymbol{L})(w_I^{(0)}\boldsymbol{I} + w_L^{(0)}\boldsymbol{L})\boldsymbol{Y}^{(0)} + v^{(0)}(w_I^{(1)}\boldsymbol{I} + w_L^{(1)}\boldsymbol{L})\boldsymbol{X} + v^{(1)}\boldsymbol{X}.\end{aligned} \tag{3.12}$$

Finally, the closed-form formula of recursion (3.4) from above iterations (3.12) can be obtained as:

$$\boldsymbol{Y}^{(T+1)} = \prod_{t=0}^{T}(w_I^{(t)}\boldsymbol{I} + w_L^{(t)}\boldsymbol{L})\boldsymbol{Y}^{(0)} + \sum_{t=0}^{T} v^{(t)} \prod_{k=t+1}^{T}(w_I^{(k)}\boldsymbol{I} + w_L^{(t)}\boldsymbol{L})\boldsymbol{X}. \tag{3.13}$$

From (3.13), autoregressive output at time $T + 1$ depends on weighted combination of initial autoregressive state $\boldsymbol{Y}^{(0)}$ and input feature matrix $\boldsymbol{X}$. The first term

in (3.13) is stability condition of the system because periodic product must be bounded to not let the output $\boldsymbol{Y}^{(T+1)}$ to explode. As a result, the stability condition of pARMA-GNN-s is given as:

$$|\prod_{t=0}^{T+1}(w_I^{(t)} + w_L^{(t)}\rho)| < 1, \tag{3.14}$$

where $\rho = \max\{|\lambda_i|\}$ is a spectral bound of the graph Laplacian. This stability condition ensures that the output is not increased indefinitely. The frequency response of the system is obtained by considering it in its limiting case when $T \to \infty$. Thus, first term in (3.13) approaches 0 due to stability condition and the output is given as:

$$\lim_{T\to\infty} \boldsymbol{Y}^{(T+1)} = \sum_{t=0}^{T} v^{(t)} \prod_{k=t+1}^{T} (w_I^{(k)}\boldsymbol{I} + w_L^{(t)}\boldsymbol{L})\boldsymbol{X}. \tag{3.15}$$

This means that output of the system does not depend on initial conditions. The further derivation of (3.15) is provided in [42], which is given as:

$$\boldsymbol{Y} = \Big(\boldsymbol{I} - \prod_{t=0}^{P-1}(w_I^{(t)}\boldsymbol{I} + w_L^{(t)}\boldsymbol{L})\Big)^{-1}\Big(\sum_{t=0}^{P-1} v^{(t)} \prod_{l=t+1}^{P-1} (w_I^{(l)}\boldsymbol{I} + w_L^{(l)}\boldsymbol{L})\Big)\boldsymbol{X}. \tag{3.16}$$

Then, by dividing both sides of (3.16) by input matrix $\boldsymbol{X}$ and considering a specific frequency λ, the frequency response of pARMA-GNN-s is defined as:

$$H(\lambda) = \frac{\sum_{t=0}^{P-1} v^{(t)} \prod_{l=t+1}^{P-1} (w_I^{(l)} + w_L^{(l)}\lambda)}{1 - \prod_{k=0}^{P-1}(w_I^{(k)} + w_L^{(k)}\lambda)}. \tag{3.17}$$

The obtained frequency response (3.17) is very flexible and has capabilities to filter any arbitrary frequency range. The main parameters that affect the behavior of filter are weights and period. It can be illustrated by setting each parameter as shown in Table 3.1 as an example, where period is set to 2. In this case, (3.12) is simplified as:

$$H(\lambda) = \frac{v^{(0)}(w_I^{(1)} + w_{\tilde{L}}^{(1)}\lambda) + v^{(1)}}{1 - (w_I^{(0)} + w_{\tilde{L}}^{(0)}\lambda)(w_I^{(1)} + w_{\tilde{L}}^{(1)}\lambda)}. \tag{3.18}$$

From above equation, it can be seen that denominator is second order polynomial, which allows to have low-pass, band-pass and high-pass filtering capability depending on weights. The results of frequency response plots are shown in Figure

Table 3.1: Weight parameter values for frequency response with $P = 2$.

Parameters	Low-pass	Band-pass	High-pass
$w_I^{(0)}$	0.8	0.5	0.05
$w_I^{(1)}$	0.9	0.5	0.05
$w_{\tilde{L}}^{(0)}$	-0.2	0.7	0.5
$w_{\tilde{L}}^{(0)}$	-0.2	-0.7	0.5
$v^{(0)}$	1.0	1.0	1.0
$v^{(1)}$	0.2	-1.0	-1.0

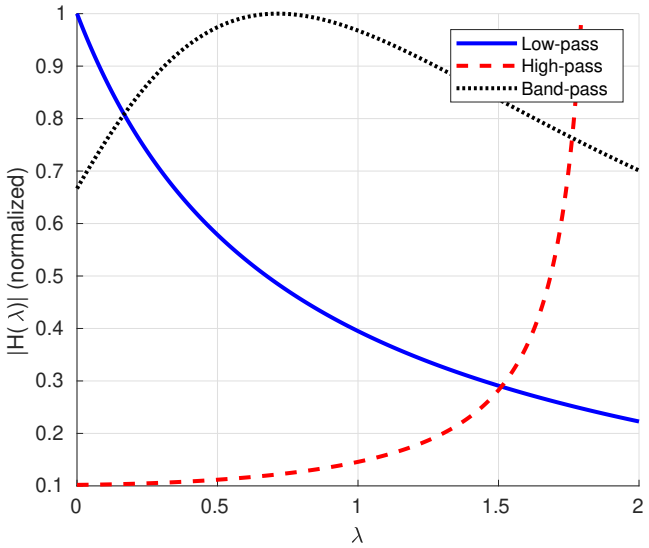


Figure 3.3: Normalized frequency responses of pARMA-GNN with low-pass, band-pass, and high-pass behaviors when $P = 2$.

3.3. It can be observed that the frequency response of the proposed pARMA-GNN-s is flexible and can model any filtering behavior including low-pass, band-pass and high-pass. The eigenvalues corresponding to frequency of graphs are bounded due to normalized graph Laplacian. The frequency responses are also normalized for better visual representation.

Chapter 4

Results and Discussions

4.1 Datasets and Performance Measurements

For the node classification task, two sets of datasets have been used [1]. The first group of datasets includes homophilic citation networks such as Cora, CiteSeer and PubMed from [44] with descriptions given in Table 4.1. Cora is the smallest dataset containing 2708 nodes, 5429 edges and 1433 dimensional features with the largest number of classes, 7 in total. CiteSeer has the highest feature dimension of 3703 with 3312 nodes, 4372 edges and 6 classes. Lastly, PubMed is the biggest graph with 19717 nodes, 44338 edges, 3 classes and feature dimension of 500. The second group consist of 6 heterophilic graphs, namely: Chameleon and Squirrel datasets from Wikipedia networks [45], induced Actor dataset [46], and university webpage networks Wisconsin, Texas and Cornell [46]. Chameleon and Squirrel are networks created from a collection of wikipedia webpages involving chameleons and squirrels, respectively. Actor dataset is extracted from film-director-actor-writer network, where actors starred in the same film are connected. Wisconsin, Texas and Cornell are networks created from a collection of university webpages. Heterophilic graphs are divided into 2 groups, where each group contains 3 graph datasets. Actor, Chameleon and Squirrel are considerably large graphs and their descriptions are given in Table 4.2, where it can be seen that Actor has the largest number of nodes while Squirrel has around 200000 edges, which is the highest number in all given datasets. The descriptions of small heterophilic datasets, Wisconsin, Texas and Cornell are given in Table 4.3, where graphs contain no more than 251 nodes and the highest number of edges is only 450. All descriptions also include the edge homophily ratio from [47] given as:

$$h = \frac{|\{(u_i, u_j) : (u_i, u_j) \in \mathcal{E} \wedge y_i = y_j\}|}{|\mathcal{E}|}, \tag{4.1}$$

[1] All datasets are obtained from public access and do not contain any sensitive information

Table 4.1: Descriptions of homophilic graph datasets used in node classification experiments.

Features	Cora	CiteSeer	PubMed
Number of nodes	2708	3327	19717
Number of edges	5278	4552	44324
Number of classes	7	6	3
Number of features	1433	3703	500
Homophily ratio (h)	0.81	0.74	0.80

Table 4.2: Descriptions of large heterophilic graph datasets used in node classification experiments.

Features	Actor	Chameleon	Squirrel
Number of nodes	7600	2277	5201
Number of edges	26659	31371	198353
Number of classes	5	5	5
Number of features	932	2325	2089
Homophily ratio (h)	0.22	0.23	0.22

where y_i is the label of node u_i. The given edge homophily ratio measures the fraction of edges that connect nodes with the same label. Thus, if h is close to 1, then most of nodes with the same label are connected with each other, meaning that underlying graph structure exhibits smooth changes across connected nodes. Such type of nodes can be classified using low-pass filtering. Whereas, homophily ratio close to 0 means that changes along edges are very sharp and variation in signals or feature vector between neighboring nodes are large. These graphs require high-pass filtering capability from a model to detect sharp changes and preserve information of each node instead of smoothing them out. As can be seen from description tables, homophilic graphs have high h, while heterophilic graphs have small h.

For the graph classification task, 3 datasets from TUDataset [48] have been used, namely: MUTAG, PROTEINS, IMDB-BINARY. In the case of IMDB-BINARY, it does not have feature matrix; thus, graph properties such as clustering coefficients and node degrees have been extracted and used as feature matrix.

All datasets have been used directly from PyTorch Geometric (PyG) Python library [49]. Heterophilic graphs have been converted into undirected graphs and self-loops have been removed. This is done because some graphs are initially given as directed and certain models do not require self-loops.

For all experiments, accuracy has been chosen as a performance metric, which

Table 4.3: Descriptions of small heterophilic graph datasets used in node classification experiments.

Features	Wisconsin	Texas	Cornell
Number of nodes	251	183	183
Number of edges	450	279	277
Number of classes	5	5	5
Number of features	1703	1703	1703
Homophily ratio (h)	0.18	0.06	0.12

is given as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.2}$$

where TP is true positives, TN is true negatives, FP is false positives, FN is false negatives.

However, it should be also noted that there are other important performance metrics that can be used such as: recall, precision, F1 score, ROC AUC. These metrics allow to get better understanding of the model's behavior. Aforementioned metrics will be used in the future experiments.

4.2 Implementation and Reproduction of Graph Neural Network Models

Two GNN architectures have been reviewed and implemented to better understand the mechanism behind them. Specifically, Graph Convolutional Network (GCN) [34] as an original graph convolutional and representative of linear spectral model and Graph Attention Network (GAT) [14] as an example of attention mechanism in graphs. The implementations are done using Python programming language and its deep learning libraries PyTorch and PyTorch Geometric (PyG). The specifications of custom implementations followed original ones as close as possible. There have been some adjustments due to computation limitations.

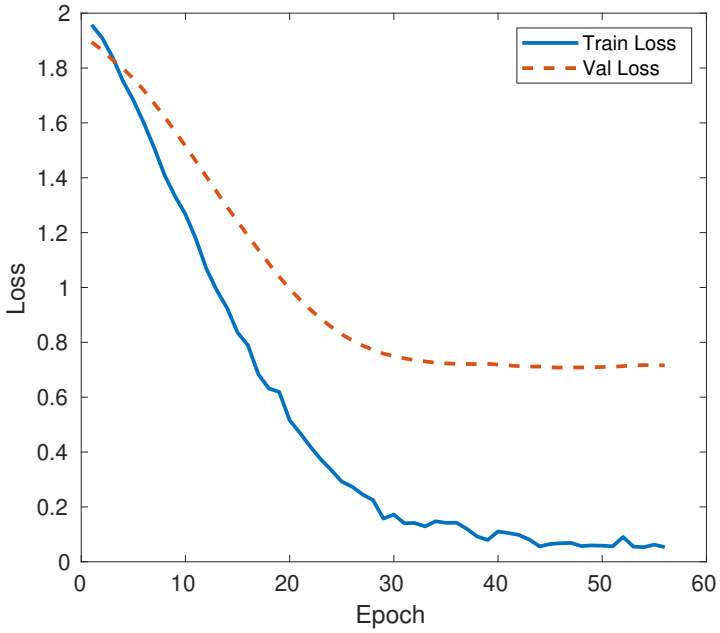
The first implemented GNN model is GCN. The results of custom and PyG training for Cora dataset are shown in Figure 4.1 and Figure 4.2 respectively. Whereas the custom and PyG implementation results for CiteSeer dataset are show in Figure 4.3 and Figure 4.4 respectively. The model relies on custom class *GCNLayer*, which is one layer of the model that performs forward pass. After that two separate models have been initialised, the first one uses custom layer while the second one uses *GCNConv* official implementation from PyG library. Models consist of two layers with early stopping on validation loss, dropout mechanism and L2 regularization to prevent overfitting. As can be seen from the results, custom

implementations show similar performance in both datasets. The general trend across all results is that models are over-fitting to training data; therefore, validation accuracy is significantly lower than training accuracy.

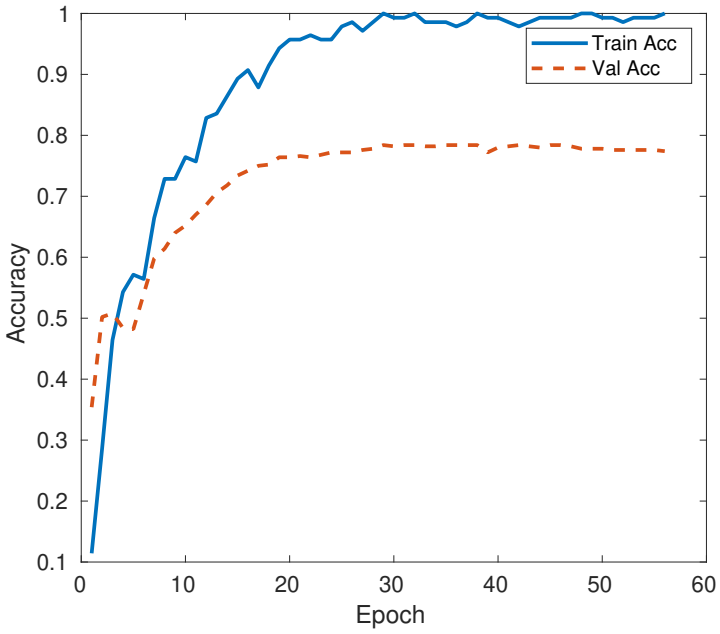
The second model is GAT that uses attention mechanism to compute the contribution weight of each neighboring node to the target node. GAT is more expressive than GCN because it assigns specific weight to each node instead of assigning normalized weights as GCN. The custom and PyG implementation results trained on Cora dataset are shown in Figure 4.5 and Figure 4.6 respectively. Whereas the custom and PyG implementation results for CiteSeer dataset are show in Figure 4.7 and Figure 4.8 respectively. The main component of the model is *GATLayer* module that implements the one layer of forward pass. The difference in this implementation from GCN is that all function passes have been written purely in tensor operations without any optimization techniques. The number of epochs have been set to 200, which is considerably smaller than in the original paper. This is done due to the computational complexity. Early stopping mechanism on validation loss and accuracy is applied in addition to L2 regularization and dropout mechanism. PyG implementation uses built-in *GATConv* layer. Again, it can be observed that custom implementations provides very similar performance results in training set for both datasets. However, it can also be noticed that PyG implementations has faster convergence, for example they reach 80% accuracy around 10 epochs earlier. Another difference is that PyG implementations has much smaller over-fit. As a result, they retain most of their performance for new data.

4.3 Structural Properties and Filtering Behavior of the Periodic ARMA Graph Neural Network

Initial experiments on homophilic and heterophilic datasets have shown interesting symmetric pattern in the autoregressive weights of the proposed pARMA-GNN-s. Discovered pattern does not depend on activation function and dataset property. The results of experiments on homophilic Cora dataset has shown in Figure 4.9(a) and Figure 4.9(a). Figure 4.9(a) shows trained pARMA-GNN-s parameters, where autoregressive weights exhibit symmetry to certain degree. This symmetry can be broken by applying previously proposed Ada-pARMA-GNN architecture in (3.6), which adds additional periodic weighting parameter. The results of using Ada-pARMA-GNN are given in Figure 4.9(a), which clearly shows that symmetry of autoregressive weights has been reduced significantly. The similar results have been obtained for heterophilic Chameleon dataset. The results of using pARMA-GNN-s are shown in Figure 4.10(a), while Figure 4.10(b) shows the results for Ada-pARMA-GNN. Again, the symmetry is broken, this time completely, after applying weighting parameter. The possible explanation for this symmetric pattern is that pARMA-GNN intrinsically trying to use normalized adjacency matrix $\tilde{\boldsymbol{A}}$ by

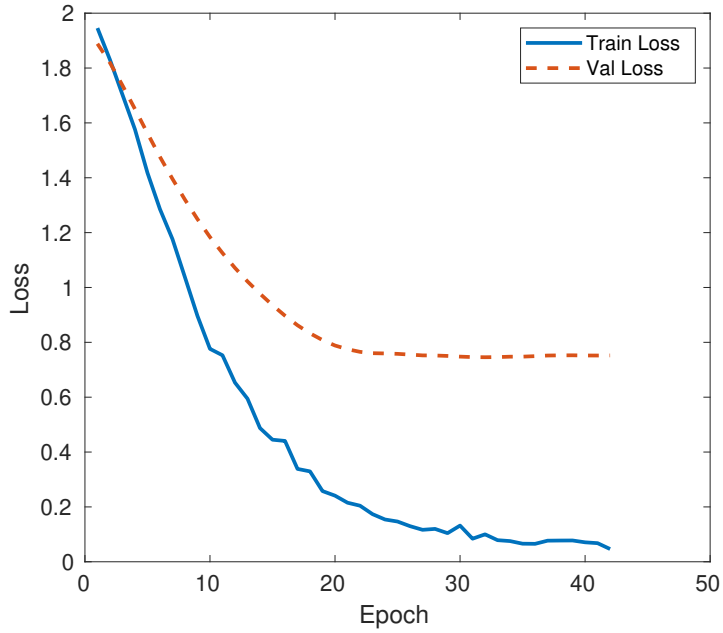


(a)

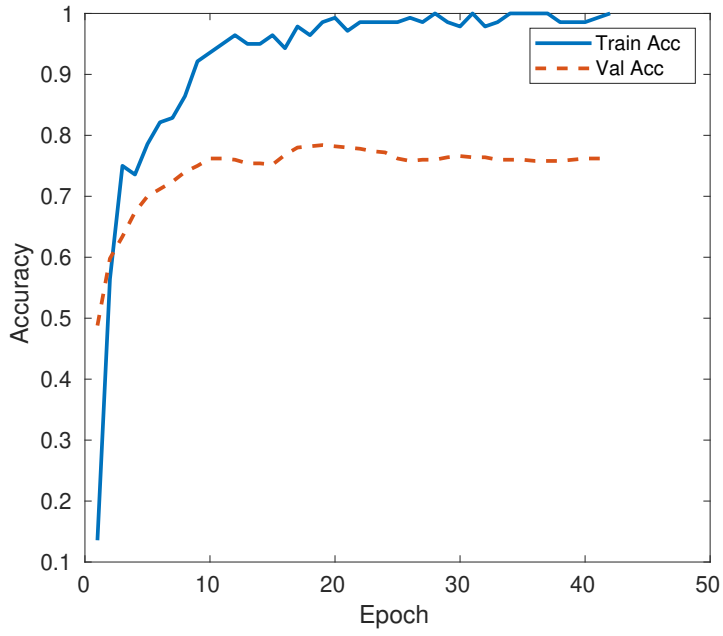


(b)

Figure 4.1: (a) Loss and (b) accuracy curves of training and validation data for custom implementation of GCN model trained on Cora dataset.

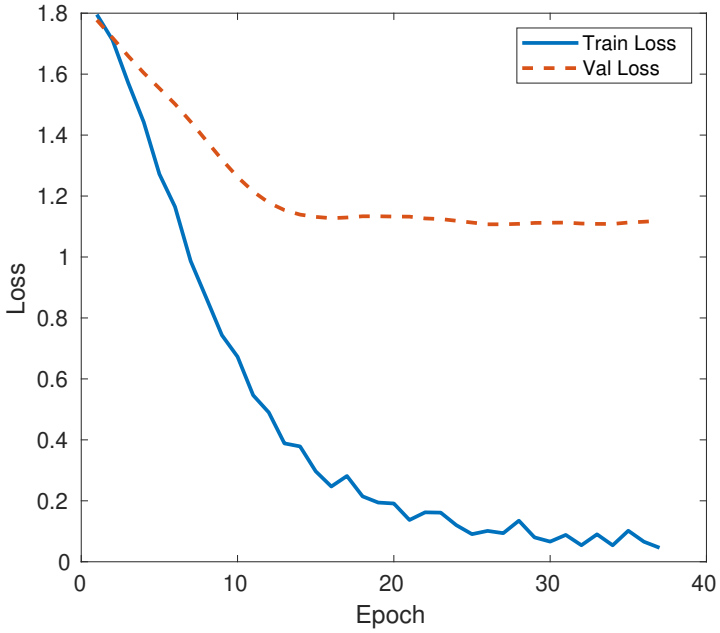


(a)

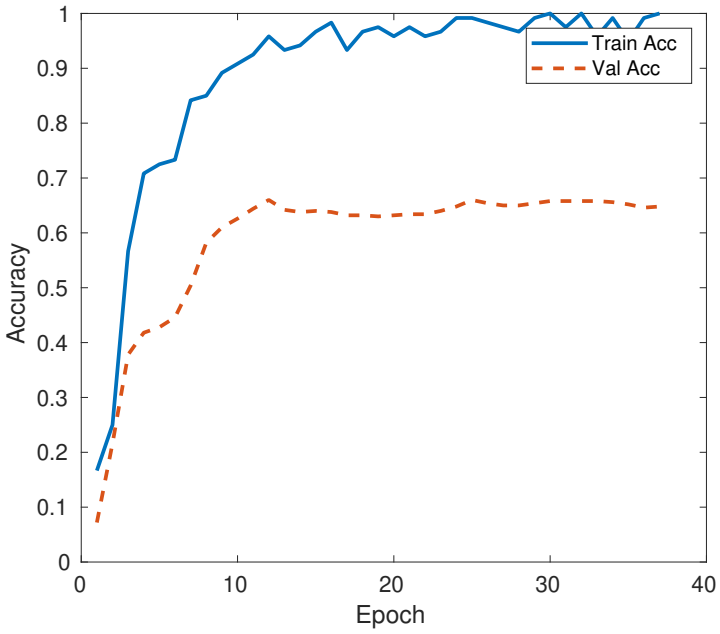


(b)

Figure 4.2: (a) Loss and (b) accuracy curves of training and validation data for PyG implementation of GCN model trained on Cora dataset.

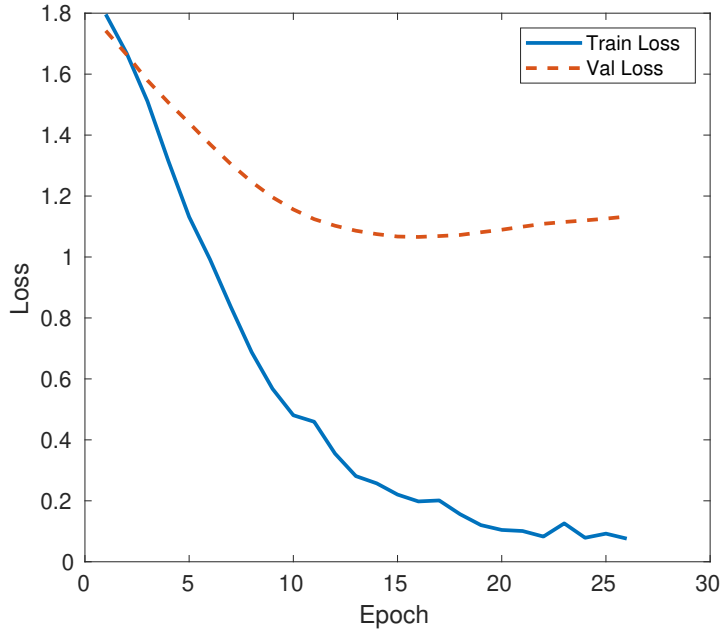


(a)

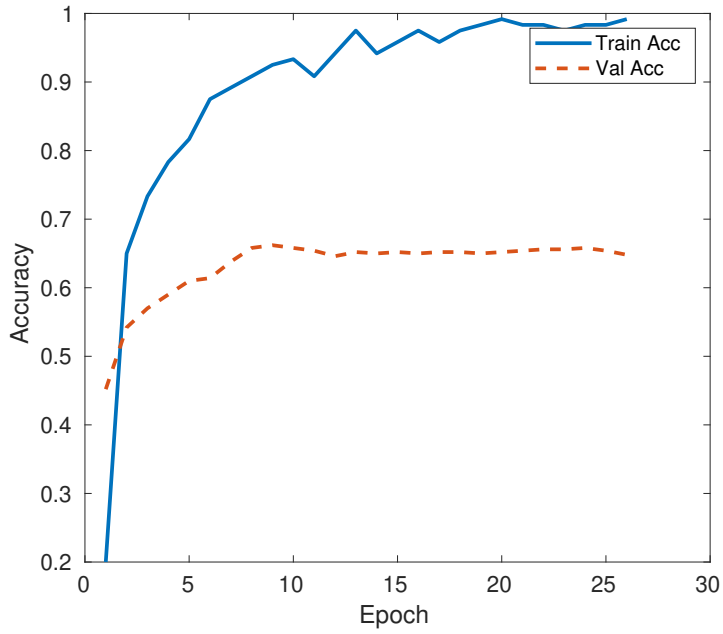


(b)

Figure 4.3: (a) Loss and (b) accuracy curves of training and validation data for custom implementation of GCN model trained on CiteSeer dataset.

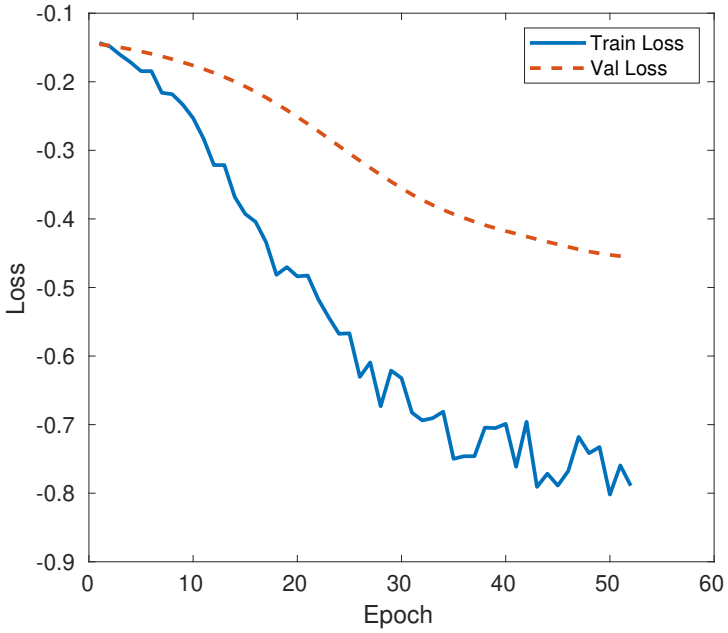


(a)

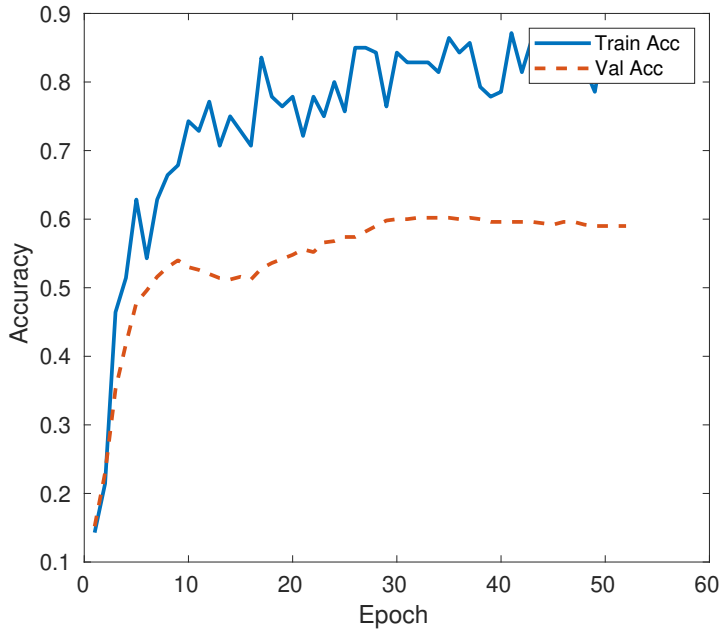


(b)

Figure 4.4: (a) Loss and (b) accuracy curves of training and validation data for PyG implementation of GCN model trained on CiteSeer dataset.

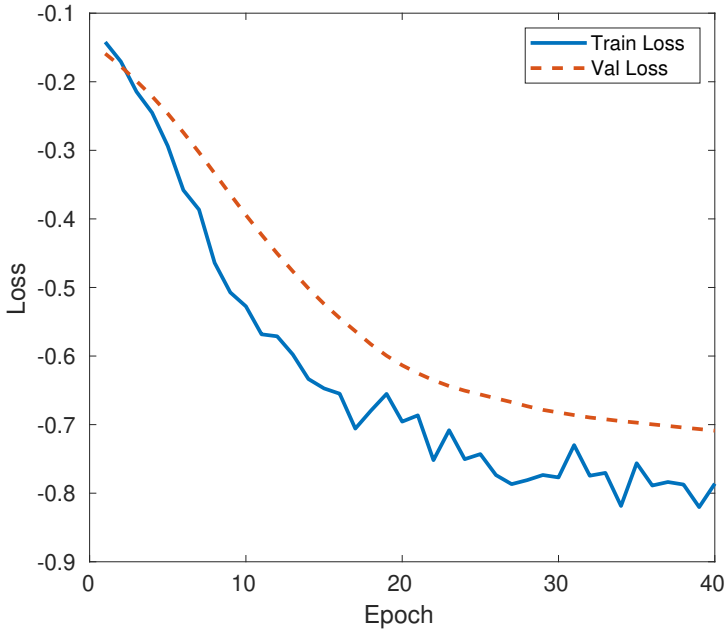


(a)

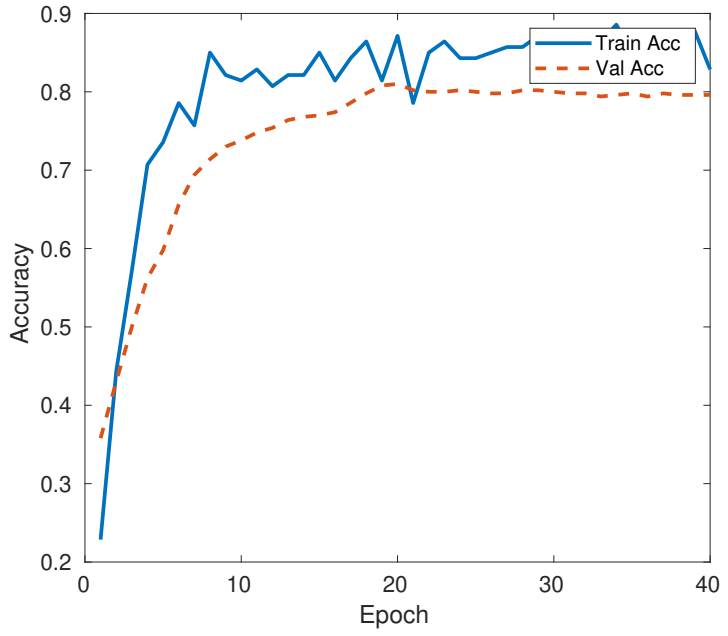


(b)

Figure 4.5: (a) Loss and (b) accuracy curves of training and validation data for custom implementation of GAT model trained on Cora dataset.

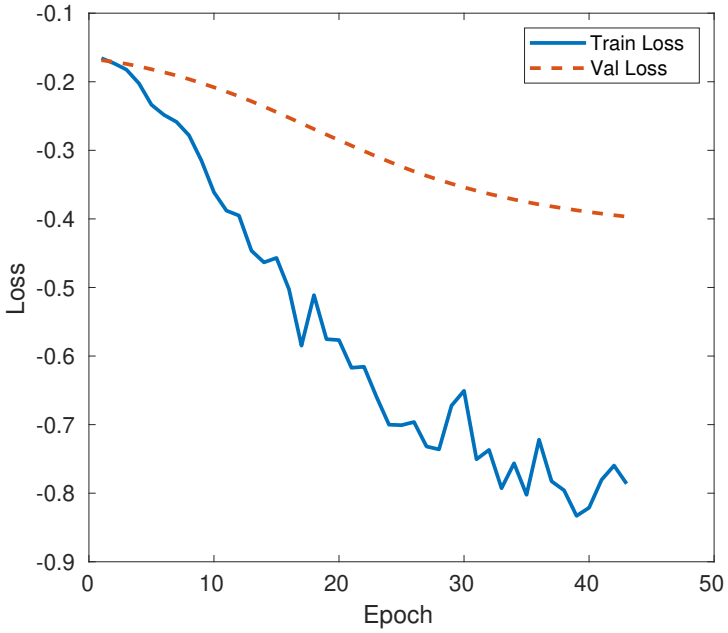


(a)

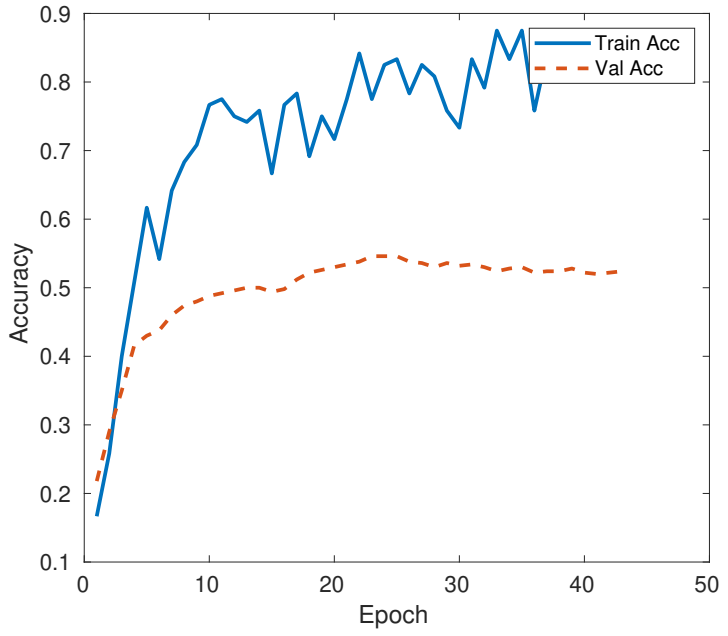


(b)

Figure 4.6: (a) Loss and (b) accuracy curves of training and validation data for PyG implementation of GAT model trained on Cora dataset.

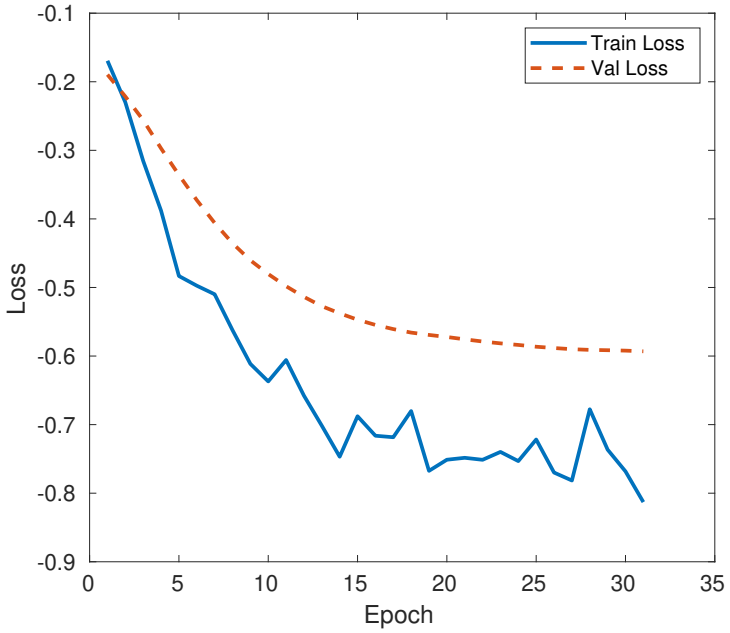


(a)

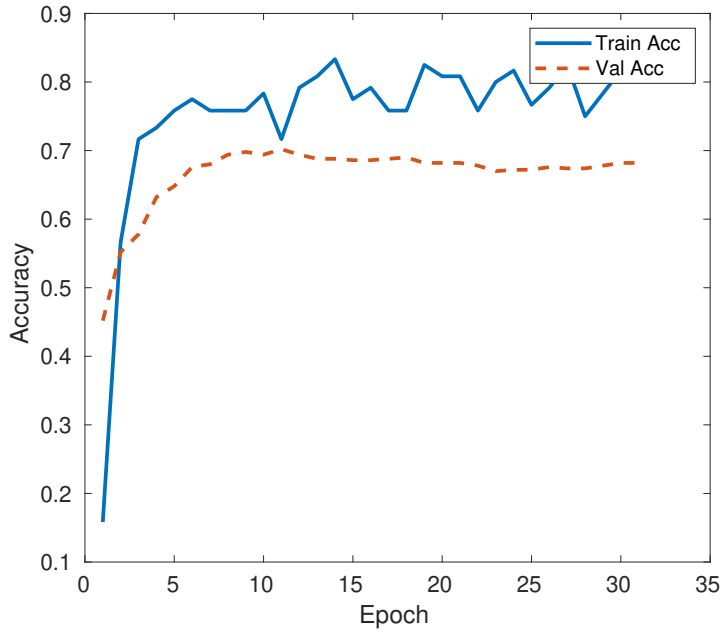


(b)

Figure 4.7: (a) Loss and (b) accuracy curves of training and validation data for custom implementation of GAT model trained on CiteSeer dataset.



(a)



(b)

Figure 4.8: (a) Loss and (b) accuracy curves of training and validation data for PyG implementation of GAT model trained on CiteSeer dataset.

forcing autoregressive weights to $w_I^{(t)} \approx w_L^{(t)}$. Even though the underlying principle of this property is unknown for now, it is not suitable for heterophilic graphs as they need to use graph shift operator that can capture high-frequency components, such as normalized graph Laplacian. Thus, applying previously proposed Ada-pARMAConv layer can in theory solve this problem. However, it is not recommended to use Ada-pARMAConv layer for homophilic graphs as they in fact work well with smoothing operators such as normalized adjacency matrix.

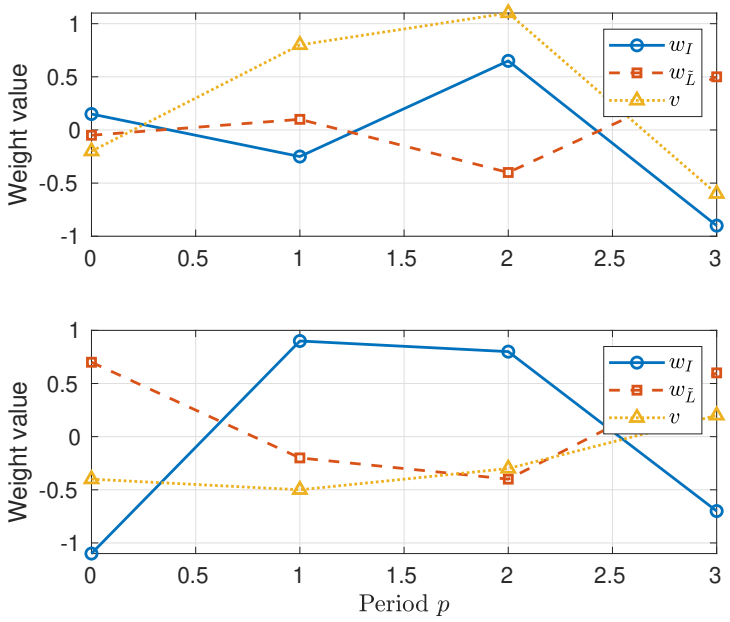
The above observation has led to an idea to proposition of Simple-pARMAConv layers defined in (3.8). The reason for that is the homophilic graphs do not lose any performance by replacing normalized graph Laplacian with normalized adjacency matrix, but it reduces the memory requirements.

In the case of filtering capabilities of pARMA-GNN in practice that has been previously shown theoretically, additional node classification experiments on synthetic data have been conducted. Specifically, a random graphs have been generated consisting of 200 nodes and randomly split into class -1 and class 1, where each class has 100 nodes. Then, the probability of connection between nodes of the same class (p) has been fixed to 0.1. Whereas the probability of connection between nodes of different classes (q) has been varied from 0.01 to 0.1 by adding 0.1 for each next graph. Nodes have single feature drawn from Gaussian distribution. The negative class has mean of -0.5 and standard deviation of 1, while the positive class has mean of 0.5 and the same standard deviation. This means that class features may overlap, so there is no ideal separation between two classes. The probability q directly controls the homophily ratio defined in (4.1). Higher probability of q lowers the homophily ratio meaning that at higher q the model must have high-pass filtering capabilities. For this experiment, GCN, ChebNet with $K = 3$, ARMAConv with $T = 1, K = 3$, pARMA-GNN with $T = 1, P = 2$ and pARMA-GNN with $T = 2, P = 2$ have been selected. The results are shown in Figure 4.11. As expected, GCN performs quite poorly at certain degrees of homophily and its accuracy drops to 70%. In contrast, polynomial models retain their performance at relatively good range with the lowest at 85%. Proposed pARMA-GNN with single timestamp has better results than pARMA-GNN with $T = 2$, which may be due to deeper network. However, both of the proposed models has good filtering capability and are competitive to ChebNet and ARMAConv models.

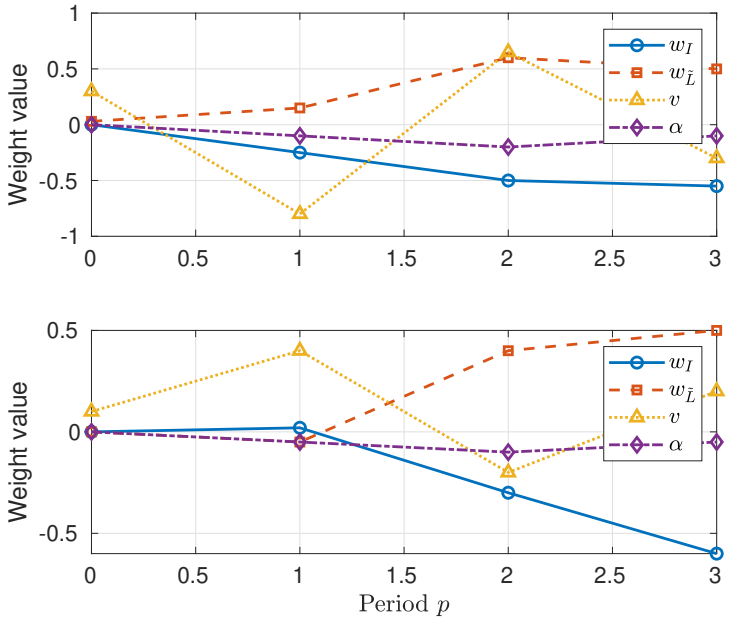
4.4 Semi-supervised Node Classification Task

For semi-supervised node classification, 11 models have been selected:

- GCN [34] as a baseline spectral model, which is based on linear approximation of Chebyshev polynomials;
- AdaGNN [38] as an adaptive model with the ability to filter high-frequency

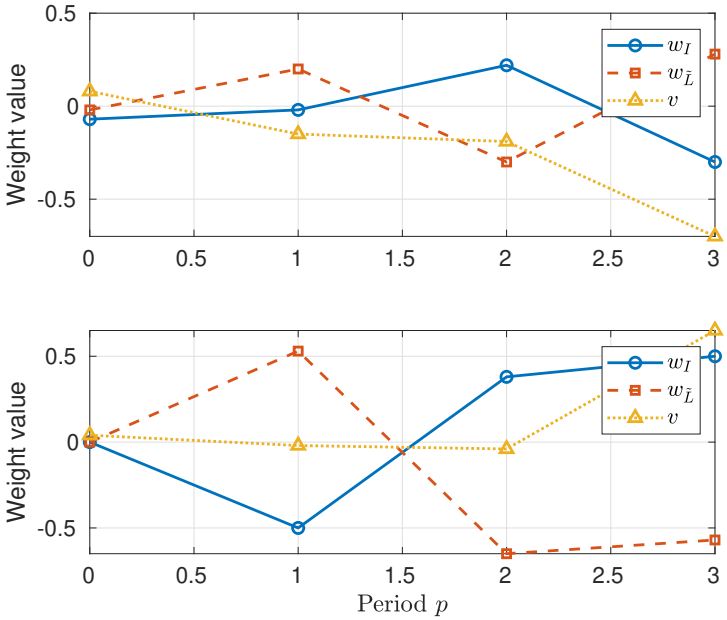


(a)

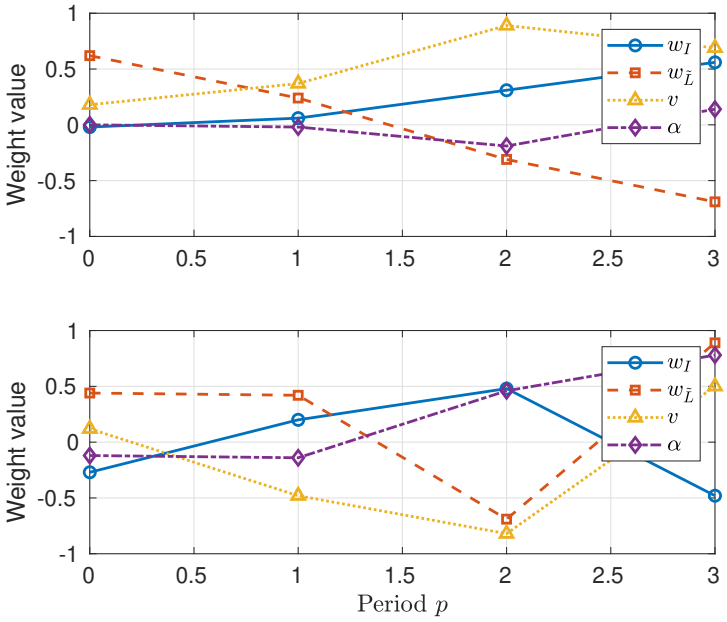


(b)

Figure 4.9: Visualization of learned weights for Cora dataset trained on 2-layer pARMA-GNN-s. The top plot is for layer 1 and the bottom plot is for layer 2. (a) Results using pARMAConv layer, (b) results using Ada-pARMAConv layer.



(a)



(b)

Figure 4.10: Visualization of learned weights for Chameleon dataset trained on 2-layer pARMA-GNN-s. The top plot is for layer 1 and the bottom plot is for layer 2. (a) Results using pARMAConv layer, (b) results using Ada-pARMAConv layer.

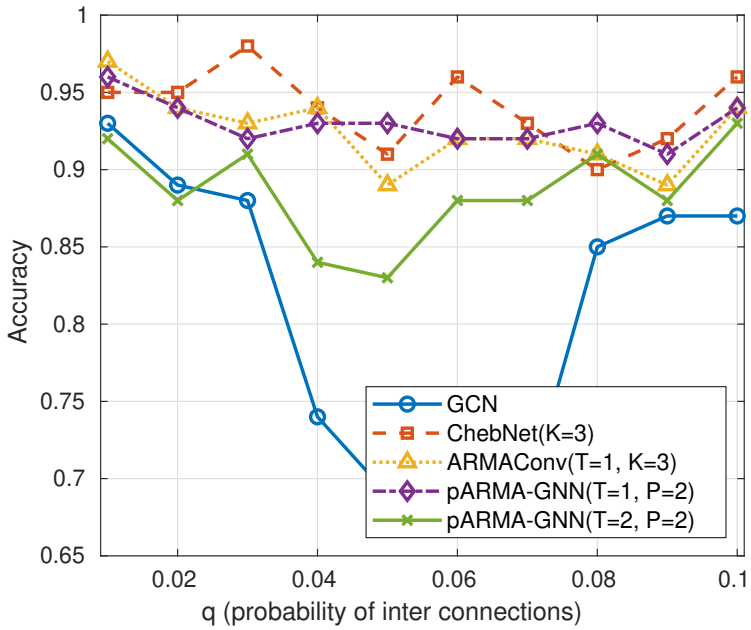


Figure 4.11: Accuracy performance for synthetic graphs with varying degree of homophily.

components;

- AKGNN [39] as another adaptive model capable of filtering different frequencies;
- BernNet [30] as polynomial filter that can model complex type of filters;
- ChebNet [28] as polynomial model that can approximate arbitrary filtering behaviors
- CayleyNet [40] as ARMA-based polynomial model with high precision in modeling band-pass filters;
- GPR-GNN [32] as polynomial model with the ability to handle both graph structure-specific and node features-specific information;
- JacobiNet [31] as generalized polynomial model with orthogonal bases;
- ARMAConv [41] as ARMA-based polynomial model and direct counterpart of the proposed pARMA-GNN;
- The proposed pARMA-GNN-s as ARMA-based polynomial model that utilizes scalar parameters and is less memory-intensive at the cost of lower expressive power;

Table 4.4: Accuracy performance results in semi-supervised node classification task for homophilic datasets.

Model	Cora	CiteSeer	PubMed
GCN[34]	79.06±1.42	63.03±1.19	74.36±2.54
AdaGNN[38]	76.20±2.46	56.70±4.85	66.36±1.63
AKGNN[39]	80.01±1.00	67.12±1.28	77.49±0.45
BernNet[30]	77.85±1.58	64.03±1.79	51.08±2.37
ChebNet[28]	72.36±3.18	62.97±2.49	73.02±3.04
CayleyNet[40]	72.71±5.12	52.39±3.23	OOM
GPR-GNN[32]	78.29±2.64	62.48±2.14	**78.01±0.78**
JacobiNet[31]	79.44±0.46	**67.98±0.85**	76.66±0.62
ARMAConv[41]	77.22±0.81	58.74±1.70	75.56±1.08
pARMA-GNN-s	**80.73±0.87**	64.68±1.18	77.04±0.58
pARMA-GNN-m	77.48±1.50	63.52±1.75	77.34±0.79

*OOM - out-of-memory

Table 4.5: Accuracy performance results in semi-supervised node classification task for large heterophilic datasets.

Model	Actor	Chameleon	Squirrel
GCN[34]	27.07±0.19	65.36±0.59	47.57±0.48
AdaGNN[38]	33.32±0.40	56.01±0.48	34.67±0.29
AKGNN[39]	33.25±0.16	55.03±0.38	42.23±0.84
BernNet[30]	30.22±0.39	42.86±0.57	22.90±0.25
ChebNet[28]	32.50±0.19	51.59±0.46	41.68±0.34
CayleyNet[40]	OOM	66.34±0.64	51.93±0.28
GPR-GNN[32]	33.33±0.35	57.97±0.25	36.02±0.24
JacobiNet[31]	32.12±0.21	44.07±0.32	31.49±0.30
ARMAConv[41]	23.98±0.29	57.91±0.25	44.63±0.27
pARMA-GNN-s	34.50±0.34	61.77±0.64	43.17±0.79
pARMA-GNN-m	**34.87±0.33**	**67.00±0.35**	**52.81±0.47**

*OOM - out-of-memory

- The proposed pARMA-GNN-m as ARMA-based polynomial model that uses matrix parameters offering higher filtering capabilities by requiring more memory.

All models have linear encoding layer that transforms input features into hidden dimension, several convolution layers depending on the specific model, and

Table 4.6: Accuracy performance results in semi-supervised node classification task for small heterophilic datasets.

Model	Wisconsin	Texas	Cornell
GCN[34]	49.53±0.77	49.19±0.94	43.86±1.37
AdaGNN[38]	83.36±1.22	80.35±1.05	<u>75.89±1.60</u>
AKGNN[39]	52.75±1.24	59.03±1.64	42.03±1.65
BernNet[30]	34.96±0.83	32.05±2.14	28.43±1.78
ChebNet[28]	76.96±1.32	76.62±1.00	62.00±1.19
CayleyNet[40]	80.55±1.12	78.89±1.89	66.22±1.73
GPR-GNN[32]	80.20±2.12	81.65±1.59	72.57±2.23
JacobiNet[31]	51.90±1.02	41.19±1.29	35.65±1.44
ARMAConv[41]	60.02±0.83	65.84±1.10	55.78±0.68
pARMA-GNN-s	**85.49±1.13**	**82.68±1.56**	**76.11±0.88**
pARMA-GNN-m	<u>84.00±0.72</u>	<u>81.97±2.10</u>	73.73±1.15

linear decoding layer that transforms features from hidden dimension to output class vector. The number of convolution layer directly depends on the model architecture, for example GCN includes only 2 convolution layers because it suffers from over-smoothing problem and deeper network deteriorates the performance while CayleyNet uses 1 convolution layer as the model is very complex and has the highest computational complexity among the tested models. The proposed pARMA-GNN-s and pARMA-GNN-m also used either 1 or 2 convolution layers depending on the dataset. This is because proposed models contain $T \cdot P$ iterations in one forward pass; thus, they propagate to larger neighborhood even with 1 layer. The polynomial degree for ChebNet, BernNet, CayleyNet and JacobiNet have been chosen from 3 to 5. Too small value of polynomial degree results in poor performance as polynomial models lose their expressive power heavily with smaller degrees, whereas higher degree leads to huge computational time as each of these models computes their corresponding polynomials during training or initialization. Each model has its own learning rate, dropout and L2 regularization. The graph shift operator of each model is set to one from their original paper. The number of epochs is set to 500 with early stopping set to 200 on validation loss. Each model has run 10 times to provide average accuracy and standard deviation results.

Table 4.4 presents accuracy results for homophilic datasets. The bold and underline indicate the best and second best results, respectively. The results for pARMA-GNN-s and pARMA-GNN-m indicate the best one from original models, and their modifications Ada-pARMA-GNN and Simple-pARMA-GNN. As can be

seen from the results, both models provide competitive performance, and pARMA-GNN-s is even the best model in Cora dataset. These results show that proposed pARMA-GNN architecture is able to filter low frequency components in a graph, utilizing their smoothness factor. The best results for CiteSeer and PubMed are also from polynomial models, which confirms their higher expressive power than linear models. However, AKGNN has also shown great performance, being second best model in all 3 datasets. In the case of ARMA-based models, proposed pARMA-GNN-s and pARMA-GNN-m outperforms both ARMAConv and CayleyNet by significant margin. Especially CayleyNet, which has one of the lowest results due to small polynomial degree. For PubMed, CayleyNet has had out-of-memory (OOM) problem due to graph size.

Table 4.5 shows the results for large heterophilic datasets. In this case, it can be observed that pARMA-GNN-m has the best results in all 3 datasets. pARMA-GNN-s scored second best in only Actor and has shown comparatively poor performance in other 2 datasets, while CayleyNet has got the second best performance in Chameleon and Squirrel with OOM in Actor dataset. Such huge difference in performance between pARMA-GNN-s and pARMA-GNN-m can be explained by their number of parameters and its effect on their expressive power. Such large graphs require considerable amount of parameters from models to capture their intrinsic patterns, where pARMA-GNN-m with its matrix parameters can easily learn that patterns. The same cannot be said on pARMA-GNN-s, which contains only $3P$ parameters per pARMAConv layer. As expected, linear models have not reached the performance level of polynomial models as they are unable to fully capture high-frequency information between neighboring nodes. In the case of ARMA-based models, pARMA-GNN-m has shown the best results, while ARMA-Conv is the worse one. The poor performance of ARMAConv can be hypothetically related to its graph shift operator, which is normalized adjacency matrix. Authors of ARMAConv have decided to use shifted version of graph Laplacian to bound the eigenvalues between -1 and 1 for stability purposes; however, by that design choice they essentially transformed their ARMA filter into low-pass filter. Even though it is fully possible to obtain other filtering behaviors by combination of low-pass filters, parallel stacking in the case of ARMAConv, the model still exhibits low-pass filtering behavior in each layer.

Table 4.6 demonstrates the results for small heterophilic datasets. As can be seen from the accuracy results, pARMA-GNN-s is the performing model with pARMA-GNN-m being second best in Wisconsin and Texas datasets. The most interesting observation is that pARMA-GNN-s performs better in small datasets while pARMA-GNN-s has significantly outperformed in large datasets. The rationale behind such good performance of pARMA-GNN-s is the same as for pARMA-GNN-m for large grahs. Small datasets does not need large amount of parameters to identify the pattern within underlying graph structure. In contrast, having a

Table 4.7: Accuracy measurements of pARMA-GNN-m models for ablation study in semi-supervised node classification task.

Model	Cora	CiteSeer	PubMed
pARMA-GNN-m	76.60±1.60	**62.91±2.24**	**77.05±0.93**
pARMA-GNN-m ($K = 1$)	75.53±1.87	60.20±1.50	74.41±9.18
pARMA-GNN-m ($T = 1$)	**76.80±1.11**	60.68±2.96	76.31±1.56

Table 4.8: Number of parameters of pARMA-GNN-m models for ablation study in semi-supervised node classification task.

Model	Cora	CiteSeer	PubMed
pARMA-GNN-m	56936	129543	26948
pARMA-GNN-m ($K = 1$)	**52712**	**125319**	**22724**
pARMA-GNN-m ($T = 1$)	65384	137991	35396

large number of parameters can be harmful due to possibility of over-fitting to the data. All linear models except AdaGNN has one of the lowest performances, which shows that the design choice of AdaGNN is capable of filtering heterophilic graphs. In the case of ARMA-based models, ARMAConv has again shown the poorest performance, which may be due to the same reasons discussed in results for large heterophilic graphs.

Additionally, ablation study has been conducted on fixing specific parameters separately to find out the individual contributions of them on the overall performance of pARMA-GNN-m. The proposed model have been compared against pARMA-GNN-m with no periodicity ($K = 1$) and pARMA-GNN-m with no timestamp ($T = 1$). Procedure is the same as in node classification experiment. Original pARMA-GNN-m is 2 layer network; thus, the timestamp of pARMA-GNN-m with no period is set to 4 while the period of pARMA-GNN-m with no timestamp is set to 4 to match the computational tree of original pARMA-GNN-m. This ensures that all networks reach the same neighborhood in each iteration. Number of parameters for each model has been also computed to highlight their difference in memory requirements.

The performance results and memory requirements are shown in Table 4.7 and Table 4.8, respectively. The accuracy performances show that pARMA-GNN-m offers the optimal solution beating other models in CiteSeer and PubMed and scoring second in Cora while being the second best in memory requirements. No periodicity results in lower number of parameters, but the performance suffers significantly as pARMA-GNN-m ($K = 1$) loses by 1-3% accuracy performance depending on a

Table 4.9: Accuracy performance measurements for graph classification task.

Model	MUTAG	PROTEINS	IMDB-BINARY
GCN [34]	71.75±13.23	69.91±5.11	**71.80±6.38**
AdaGNN [38]	69.77±13.79	59.57±4.47	56.81±7.35
GPR-GNN [32]	70.38±9.81	63.45±5.55	52.92±5.94
ARMAConv [41]	68.71±11.31	70.44±3.93	56.21±7.72
pARMA-GNN-s	66.52±9.80	59.58±4.70	53.34±4.83
pARMA-GNN-m	**74.47±11.76**	**70.88±3.48**	49.10±4.43

Table 4.10: Execution time (in minutes) of ARMA-based models on several graph datasets.

Model	CiteSeer	PubMed	Actor	Squirrel
CayleyNet [40]	10.22	OOM	OOM	213.87
ARMAConv [41]	3.19	**10.34**	39.86	37.33
pARMA-GNN-s	9.11	13.78	71.17	17.15
pARMA-GNN-m	**1.28**	16.24	**5.54**	**4.74**

dataset. In the case of pARMA-GNN-m ($T = 1$), it outperformed pARMA-GNN-m only on Cora dataset by 1%, but requires much larger memory. Such difference in number of parameters is explained by the fact that each layer does not have periodic weights; thus, it requires more memory for the same depth network.

4.5 Graph Classification Task

For graph classification, 6 models have been selected: GCN, AdaGNN, GPR-GNN, ARMAConv, pARMA-GNN-s, pARMA-GNN-m. All models have the same structure as in semi-supervised node classification experiments, except global mean pooling is added before final linear decoder layer. This is done to obtain the final embedding of the whole graph by combining all node embeddings. The procedure of experiments includes 10-fold cross validation, where training set is further split into 90% and 10% for training and validation sets. The dropout rate of 0.5 is used for all runs. The hidden dimension is set to 32. The number of epochs is 500 and early stopping is applied on validation loss with 200 steps. The aforementioned procedure is repeated 10 times.

The results of experiments are given in Table 4.9. The performance measurements show that pARMA-GNN-m has the best accuracy on MUTAG and PROTEINS, datasets with initial features. Whereas, it performs poorly on IMDB-

BINARY dataset, where the initial feature matrix has been created from graph properties of the data itself. Thus, it can be said that pARMA-GNN is not the best choice for graphs with no features or hand-crafted features as they may add additional structural information without adding any feature relationship between nodes. In contrast, GCN has the best performance on IMDB-BINARY by a large margin, which shows that GCN can be used without any specific features of nodes and gives good performance with only structural information. The results of pARMA-GNN-s may indicate that small number of parameters is not enough to model complex relationship between different graphs. In the case of ARMAConv, it has shown comparable results to pARMA-GNN-m beating it in IMDB-BINARY.

Additionally, time execution of ARMA-based models, CayleyNet, ARMAConv, pARMA-GNN-s, and pARMA-GNN-m have been tested, which is shown in Table 4.10. From the results, it can be seen that pARMA-GNN-m has overall the best execution time with only being third on PubMed dataset, which has the highest number of nodes. Apart from that, pARMA-GNN-s mostly loses to pARMA-GNN-m and has mixed results with ARMAConv. The reason for that is pARMA-GNN-s takes much more number of epochs to converge due to smaller number of parameters as it sometimes has taken all 500 epochs for training. Whereas, pARMA-GNN-m usually takes around 250 epochs to converge. It can also be observed that CayleyNet has the worst training time, especially for Squirrel dataset with the highest number of edges.

Chapter 5

Conclusion

5.1 Summary of Work Done

In this paper, two GNN models have been reproduced using custom implementation and compared to official implementations. The results of custom models have been considerably lower which may be due to several reasons such as better numerical stability in official implementations.

A novel GNN model, pARMA-GNN, is proposed utilizing periodic ARMA graph filters to reduce memory requirements and improve performance of spectral GNNs. Two variations of pARMA-GNN have been developed, specifically pARMA-GNN-s that uses scalar parameters and pARMA-GNN-m, which is based on matrix parameters. Both models have shown great performance on both homophilic and heterophilic datasets, highlighting the filtering capabilities of the proposed models. Additionally, two modifications have been introduced into pARMA-GNN architecture based on different assumptions on the structure of dataset. Ada-pARMA-GNN incorporates periodic weighting parameter that controls the influence of autoregressive terms, improving the adaptive ability of the model. Whereas, Simple-pARMA-GNN assumes that dataset is homophilic leading to simplified and less memory-intensive implementation of pARMA-GNN. The theoretical analysis including the derivation of frequency response of the model has been carried out, which has shown the ability of pARMA-GNN to model low-pass, band-pass, and high-pass behaviors.

As a conclusion of one-year work, the results of work done during Capstone I is in preparation for submission as a conference paper, while extended work performed during Capstone II is in preparation for submission as journal paper (as noted in List of Publications section).

5.2 Future Work

For future work, the several possible extensions of the current work are planned. These include studying common GNN problems such as over-smoothing and over-squashing that negatively affect the performance of the model. Further study of these problems are important because they allow to understand the underlying mechanism of the model in terms of information propagation. As a result of future studies, the expected outcome is to address the given problems and develop a solution. To achieve these goals, extensive literature review of corresponding problems and theoretical study of more advanced graph signal processing topics will be done.

List of Publications

1. B. Kalmyrzayev, H. Ali, and M. T. Akhtar, "On developing memory-efficient graph neural networks via periodic autoregressive moving average graph filtering," in *Proc. IEEE 35th Intern. Workshop Machine Learn. Signal Process. (MLSP 2025)*, Aug. 31–Sept. 3, 2025, Istanbul, Turkey. (*in preparation*)

2. B. Kalmyrzayev, H. Ali, and M. T. Akhtar, "Spectral filtering using periodic autoregressive moving average graph neural networks for heterophilic graphs," *IEEE Access*, 2025. (*in preparation*)

Bibliography

[1] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[2] Austin Derrow-Pinion et al. "Eta prediction with graph neural networks in google maps". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3767–3776.

[3] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IEEE. 2005, pp. 729–734.

[4] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[5] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[6] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 1–159.

[7] Antonio Ortega et al. "Graph signal processing: Overview, challenges, and applications". In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828.

[8] Michael M Bronstein et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[9] Kristof Schütt et al. "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions". In: *Advances in neural information processing systems* 30 (2017).

[10] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. "Directional message passing for molecular graphs". In: *International Conference on Learning Representations* (2020).

[11] Xiangnan He et al. "Lightgcn: Simplifying and powering graph convolution network for recommendation". In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 639–648.

[12] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[13] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper insights into graph convolutional networks for semi-supervised learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[14] Petar Veličković et al. "Graph attention networks". In: 2018.

[15] Shaked Brody, Uri Alon, and Eran Yahav. "How attentive are graph attention networks?" In: *International Conference on Learning Representations*. 2022.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in Neural Information Processing Systems* 30 (2017).

[17] Hanqing Zeng et al. "Graphsaint: Graph sampling based inductive learning method". In: 2020.

[18] Pan Li et al. "Distance encoding: Design provably more powerful neural networks for graph representation learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4465–4478.

[19] Jiaxuan You, Rex Ying, and Jure Leskovec. "Position-aware graph neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7134–7143.

[20] Jiaxuan You et al. "Identity-aware graph neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10737–10745.

[21] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), pp. 4–24.

[22] Zhengdao Chen et al. "Can graph neural networks count substructures?" In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 10383–10395.

[23] Christopher Morris et al. "Weisfeiler and leman go neural: Higher-order graph neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4602–4609.

[24] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *International Conference on Learning Representations* (2014).

[25] Muhammet Balcilar et al. "Analyzing the expressive power of graph neural networks in a spectral perspective". In: *International Conference on Learning Representations*. 2021.

[26] Deyu Bo et al. "A survey on spectral graph neural networks". In: *arXiv preprint arXiv:2302.05631* (2023).

[27] Muhammet Balcilar et al. “Breaking the limits of message passing graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 599–608.

[28] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in Neural Information Processing Systems* 29 (2016).

[29] Mingguo He, Zhewei Wei, and Ji-Rong Wen. “Convolutional neural networks on graphs with chebyshev approximation, revisited”. In: *Advances in neural information processing systems* 35 (2022), pp. 7264–7276.

[30] Mingguo He, Zhewei Wei, Hongteng Xu, et al. “Bernnet: Learning arbitrary graph spectral filters via bernstein approximation”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 14239–14251.

[31] Xiyuan Wang and Muhan Zhang. “How powerful are spectral graph neural networks”. In: *International conference on machine learning*. PMLR. 2022, pp. 23341–23362.

[32] Eli Chien et al. “Adaptive universal generalized pagerank graph neural network”. In: *arXiv preprint arXiv:2006.07988* (2020).

[33] Mingqi Yang et al. “A new perspective on the effects of spectrum in graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 25261–25279.

[34] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *International Conference on Learning Representations* (2017).

[35] Felix Wu et al. “Simplifying graph convolutional networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6861–6871.

[36] NT Hoang, Takanori Maehara, and Tsuyoshi Murata. “Revisiting graph neural networks: Graph filtering perspective”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 8376–8383.

[37] Deyu Bo et al. “Beyond low-frequency information in graph convolutional networks”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 5. 2021, pp. 3950–3957.

[38] Yushun Dong et al. “Adagnn: Graph neural networks with adaptive frequency response filter”. In: *Proceedings of the 30th ACM international conference on information & knowledge management*. 2021, pp. 392–401.

[39] Mingxuan Ju et al. “Adaptive kernel graph neural network”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. 6. 2022, pp. 7051–7058.

[40] Ron Levie et al. “Cayleynets: Graph convolutional neural networks with complex rational spectral filters”. In: *IEEE Transactions on Signal Processing* 67.1 (2018), pp. 97–109.

[41] Filippo Maria Bianchi et al. “Graph neural networks with convolutional arma filters”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2021), pp. 3496–3507.

[42] Elvin Isufi et al. “Autoregressive moving average graph filtering”. In: *IEEE Transactions on Signal Processing* 65.2 (2016), pp. 274–288.

[43] Andreas Loukas, Andrea Simonetto, and Geert Leus. “Distributed autoregressive moving average graph filters”. In: *IEEE Signal Processing Letters* 22.11 (2015), pp. 1931–1935.

[44] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. “Revisiting semi-supervised learning with graph embeddings”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 40–48.

[45] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. “Multi-scale attributed node embedding”. In: *Journal of Complex Networks* 9.2 (2021), cnab014.

[46] Hongbin Pei et al. “Geom-gcn: Geometric graph convolutional networks”. In: *Proceedings of International Conference on Learning Representations* (2020).

[47] Jiong Zhu et al. “Beyond homophily in graph neural networks: Current limitations and effective designs”. In: *Advances in neural information processing systems* 33 (2020), pp. 7793–7804.

[48] Christopher Morris et al. “Tudataset: A collection of benchmark datasets for learning with graphs”. In: 2020.

[49] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.