Semi-Supervised Learning in Graph Neural Networks: A Spectral Filtering Approach

Anish Anand

Semi-Supervised Learning in Graph Neural Networks: A Spectral Filtering Approach

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Anish Anand

[Roll No: CS-2023]

under the guidance of

Dr. Swagatam Das.

Associate Professor and Head,
Electronics and Communication Sciences Unit

Indian Statistical Institute
Kolkata-700108, India
July 2022

To my family and my Supervisor.

CERTIFICATE

This is to certify that the dissertation entitled **"Semi-Supervised Learning in Graph Neural Networks: A Spectral Filtering Approach'** submitted by **Anish Anand** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Dr. Swagatam Das
Associate Professor and Head,
Electronics and Communications Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my deepest gratitude to my advisor, *Prof. Dr. Swagatam Das*, ECSU, Indian Statistical Institute, Kolkata, for his guidance, continuous support, and encouragement.

I would also like to thank *Kushal Bose* Senior Research Fellow, ECSU, Indian Statistical Institute, Kolkata, for his valuable suggestions and discussions.

My deepest thanks to all the faculties of the Indian Statistical Institute, for their valuable suggestions and discussions which contributed to my research work.

I owe my thanks to many people who have helped me in many ways these years. I want to thank my family for their love and support. I am indebted to ISI for providing me with an opportunity to learn and work in a lively scientific environment. I finally want to mention some of the teachers, scientists, and friends with whom I have interacted during these years: Mandar Mitra, Sandip Das, Malay Bhattacharya, Azad, Sachin, Akhil, and Ayush.

Anish Anand
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

In this thesis, we investigated the general framework of Graph Neural Network for node and graph classification tasks. We studied the phenomenon of over smoothing and conducted experiments to validate them. We propose a novel spectral-based Legendre Filter based on the Legendre polynomial to learn node features on graph-structured data. We also described various aggregation schemes that can be employed with the Legendre Filter to further improve information aggregation. Furthermore, we proposed a novel algorithm that changes graph topology based on some heuristics to improve overall classification accuracy. For Semi-Supervised learning, we demonstrated that our proposed method performs better than GCN and Chebyshev Filter on Citation Network datasets. Our proposed model outperformed GAT on Citeseer and PubMed. For the full-supervised learning task, we showed that our method outperforms all three baselines; GCN, GAT, and Chebyshev Filter on Citation Network and WebKB datasets. We further showed that our method outperforms deep GNN models like GCNII, JKNet on the WebKB dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Deep Learning on Graphs

Graphs provide a natural way to represent the data. Data from different domains like social networks, chemistry, brain networks, citation network, transportation network, and e-commerce product networks can be better represented through graphs. Classical deep learning models work with tabular data assuming that each data point is independent and identically distributed. But the assumption is inherently flawed as we saw that data points in many data across different systems are related to each other. Classical Deep Learning techniques have been proven powerful in the representational learning task and have achieved benchmark performance in the areas like Natural Language Processing, Computer Vision, Anamoly Detection, etc. However, classical deep learning methods consider data points independent and therefore do not consider other data points while generating features. Since deep learning on graphs takes into consideration the topology of graph-structured data, generating the features takes into account the input features of other data points. Graph Neural Network has more representational capacity for graph-structured data than classical deep learning models. Semi-Supervised learning comes into play when we have a small number of labeled data and a huge number of unlabelled data. As we will demonstrate in this work Graph, Neural Network significantly outperforms classical deep learning methods on such a task.

1.2 Contents covered

Now we briefly explain contents covered in subsequent chapters:

- Chapter 2: In this chapter, we studied the basic properties of Graph Laplacian and its use in Graph Fourier Transform. Next, we investigated a general framework for Graph Neural networks for node/graph classification tasks. Next, we studied different approaches to designing filters and reviewed a couple of them. We also investigated the phenomenon of over-smoothing in Graph Convolutional Networks. Finally, we explored some deep architectures in Graph Neural Networks.

- Chapter 3: In this chapter, we propose our novel spectral-based filter, namely, Legendre Filter. Next, we discuss a particular case of our proposed filter. We then generalize Legendre Filter to suggest Jacobi Filter based on Jacobi polynomials. Finally, we propose a novel algorithm that changes graph topology based on some heuristics to improve the model's overall accuracy.

- Chapter 4: In this chapter, we conducted an over-smoothing experiment and validated our conjecture empirically given in the previous chapter. Next, for the semi-supervised learning task, we compared our model's performance with three baselines; Chebyshev Filter, GCN, and GAT on Citation Network, CoraML, Amazon Photo, DBLP, Coauthor CS. We also include other SOTAs for comparison. We compared our model's performance with shallow and deep models on Citation Network and Web Network datasets for the full-supervised learning task.

Chapter 2

Background Literature and Related Works

2.1 Basic Properties of Graph

We begin with the definition of the graph and its properties.

Definition 1 *A graph $G = \{V, E\}$ is defined by the node set $V = \{v_1, v_2, ..., v_n\}$ and edge set $E = \{e_1, e_2, ..., e_m\}$.*

Definition 2 *Given a graph $G = \{V, E\}$ such that $|V| = n$. An adjacency matrix $\mathbf{A}$ is defined as $n \times n$ matrix such that $A_{ij} = 1$ if and only if v_i and v_j is connected by an edge.*

Definition 3 *Given a graph $G = \{V, E\}$, degree of vertex v is defined as number of adjacent vertices of v.*

Theorem 1 *Given a graph $G = \{V, E\}$, following result holds true.*

$$\sum_{v \in V} deg(v) = 2|E|$$

2.1.1 Centrality

In a graph, centrality is the measure of how important a node is in the graph. There are many types of centrality measures, we mention a few of them.

- Degree Centrality: The importance of a node can be decided on the basis of the number of edges connected to it. Hence, degree centrality is defined as

$$cen(v) = degree(v)$$

- Eigenvalue Centrality [3]: Degree centrality has one drawback. It gives equal importance to all the adjacent nodes. Eigenvalue centrality is the eigenvector of the adjacency matrix for a particular eigenvalue.

$$A \cdot w = \lambda \cdot w$$

  Here w contains the centrality score for all the nodes.

2.2 The Graph Laplacian

Given a graph $G = \{V, E\}$ with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, graph Laplacian [3] is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

There are many versions of the Laplacian matrix we mention two of them.

- Normalized Laplacian [3]: Normalized Laplacian is defined as

$$\mathbf{L}_{nor} = I - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$$

- Random Walk Laplacian [3]: RW Laplacian is defined as

$$\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L}$$

  RW Laplacian can also be thought of transition probability matrix of Random Walk on the graph.

We state the following result from Chung [3]. Given a vector x and simple graph Laplacian L following equation holds true

$$x^T L x = \sum_{v_i \in V} \sum_{v_j \in N(v_i)} (x[i] - x[j])^2$$

Here $N(v)$ represents neighbouring nodes of v. We mention the following theorem from Chung [3]

Theorem 2 *Given a graph $G = \{V, E\}$, the eigenvalues of L are positive.*

Proof:
Let λ and x be the eigenvalue and normalized eigenvector of L. Thus we have $x^T x = 1$. From above equation it follows that,

$$0 \leq x^T L x = \lambda x^T x = \lambda$$

. □

2.3 Graph Signal Processing

A graph signal comprises a graph $G = \{V, E\}$ and a mapping function on the vertex set V to $\mathbf{R}^d$ space.

$$f : V \rightarrow \mathbf{R}^d$$

Consider a single-channel signal f. If the values change slowly as we traverse through the graph via the edges, then we say the signal f is smooth. It can be observed that the signal is smooth if and only if the value of $f^T L f$ is small.

2.3.1 Graph Fourier Transform

Let f be a single channel graph signal. Suppose eigenvalue decomposition of graph Laplacian L is given by $L = U \Lambda U^T$. Here i^{th} column correspond to the i^{th} eigenvector u_i of the graph Laplacian L.
Graph Fourier transform [25] of f is given by:

$$\hat{f} = U^T f$$

By observing following equation

$$u_i^T L u_i = \lambda_i$$

we conclude that smoothness of the eigenvector u_i is measured by the eigenvalue λ_i. Spectral graph signal $\hat{f}$ can be transformed back to spatial graph signal f using Inverse Fourier Transform [25].

$$f = U\hat{f}$$

.

2.4 The GNN Framework

Graph Neural Networks are a collection of methods that seeks to apply deep learning neural network to graph-structured data. Classical deep learning techniques are designed to work well with tabular data, but it performs poorly when it comes to graph-structured data. The research on GNNs goes back to Sperduti et al. [27] where the neural network model was applied to graph-structured data, which motivated further research (Gori et al. [16]) on using deep learning method on graphs. Graph neural networks can be thought of as representational learning on graphs. For node-focused tasks, it aims to learn better node representation which helps with the node classification task. Graph-focused tasks aim to learn the representation for the entire graph required for the graph classification task. Now we describe Graph Neural Networks (Scarselli et al. [5]) framework for both node and graph-focused tasks.

2.4.1 Framework for Node Classification Task

A framework of GNN for node classification task [14] can be viewed as composition of graph filtering layer and application of non linear activation function. We denote the filter by F and activation function by a. Let $G = \{V, E\}$ be a graph with adjacency matrix A. Let $X \in \mathbf{R}^{n \times d}$ be the feature matrix. A GNN framework with l Graph Filtering layers and $l-1$ activation function is given by

$$X^{(i)} = F_i(A, a_{i-1}(X^{(i-1)}))$$

Here, $X^{(0)}$ is the input feature, and $X^{(i)} \in \mathbf{R}^{n \times d_i}$ is the output feature obtained after application of i^{th} filter. a_0 is identity function, i.e. we don't apply activation function to input feature matrix $X^{(0)}$.

2.4.2 Framework for Graph Classification Task

A framework of GNN for graph classification [9] tasks comprises the graph filtering process, transformation through an activation function, and Graph Pooling [5] layer. Graph pooling layer takes adjacency matrix and feature matrix as input and returns modified adjacency matrix and new feature matrix. We denote the graph pooling operation by $Pool()$. The general architecture consists of k blocks. Each block consists of a series of graph filtering layers and activation functions. Graph Pooling layer is applied to each of the blocks. Let input to the block with k filtering layer is given by (A^{in}, X^{in}). The output of the block after the Pooling layer is:

$$X^{(i)} = F_i(A^{in}, a_i(X^{in})) \ for \ i = 1, 2, ..., k.$$

$$A^{out}, X^{out} = Pool(A^{in}, X^{(k)})$$

Graph Classification task can be viewed as a hierarchical process that slowly coarsens the graph to generate graph-level features.

2.5 Training Parameters for Graph Neural Network

2.5.1 Training Parameters for Node Classification Task

Let $G = \{V, E\}$ be a graph with feature matrix and adjacency matrix given by X and A respectively. We divide the vertex set $V = V_l \sqcup V_{ul}$ into disjoint subsets of labeled vertices and unlabeled vertices. The task of classification is to train a model based on the labeled vertex set V_l to predict the label of the unlabeled vertex set V_{ul}. Suppose a GNN model is given by $GNNModel()$. For this task, any GNN model takes the

entire graph as input. Suppose out of the final layer of the model is X^{out}.

$$X^{out} = GNNModel(A, X; W_g)$$

where W represents the model parameters to be trained. X^{out} is passed through softmax layer [8].

$$\hat{Y} = Softmax(X^{out}; W_s)$$

where $\hat{Y} \in \mathbf{R}^{n \times c}$, here c is the number of classes. We could also write the above simply as

$$\hat{Y} = g_{GNN}(A, X, W)$$

where $W = W_g \; and \; W_s$. Given a loss function $l(\cdot|\cdot)$ (usually we take Cross Entropy Loss Function [8] for classification task) we train the model to minimize the following objective function

$$\mathbf{L}_{train} = \sum_{v \in V_l} l(g_{GNN}(A, X, W)_i, y_i)$$

where y_i is the true class label.

2.5.2 Training Parameters for Graph Classification Task

Suppose we have a training set $S_{train} = \{G_i, y_i\}$ where G_i is the graph and y_i is the label of the graph. The task of graph classification is to train a model based on the training set S_{train} to predict the label of the unlabeled graphs. Let $GNNModel()$ be the GNN model which takes graph G_i as input and generates graph level feature.

$$X_i^{out} = GNNModel(A, G_i; W_g)$$

where $X_i^{out} \in \mathbf{R}^{1 \times d}$. The output is passed through softmax layer to produce $\hat{Y}_i$.

$$\hat{Y} = Softmax(X^{out}; W_s)$$

We could also write the above simply as

$$\hat{Y} = g_{GNN}(G, W)$$

where $W = W_g \; and \; W_s$. Same as above model is trained to minimize the following objective

$$\mathbf{L}_{train} = \sum_{G_i \in S_{train}} l(g_{GNN}(G_i, W), y_i)$$

2.6 Graph Filters

There are many different ways to design graph filters. Almost all of them can be categorized into two categories:

- Spectral-Based Filter: Spectral-Based filter uses spectral graph theory to design filtering layers in the spectral domain.
- Spatial-Based Filter: Spatial-Based Filter uses the graph topology and graph structure information for feature refining.

  As we will show in the next section, these two approaches are related. A spectral-Based filter can be viewed as a Spatial-Based filter.

2.6.1 Spectral-Based Filter

We consider single-channel graph signal $X \in \mathbf{R}^n$. After applying Graph Fourier Transform [25], we get following

$$\hat{X} = U^T X$$

where i^{th} column of U correspond to i^{th} eigenvector with λ_i as eigenvalue of L . The i^{th} element of the $\hat{X}$ is the i^{th} Fourier component of u_i with corresponding frequency λ_i. Now we use a function $m()$ to modulate the frequency of the graph signal $\hat{X}$. The function m is defined on frequency set of graph signal. Following it, we apply inverse Graph Fourier Transform to get the smoothened signal $\hat{X}'$.

$$\hat{X}' = U \cdot m(\Lambda) \cdot U^T X.$$

where Λ is the diagonal eigenvalue matrix. Since we do not know which frequencies (eigenvalues) are important or how to modulate them, the problem thus becomes finding the best-modulating function m for the given data. However, we can learn

the function m through a data-driven approach.

2.6.2 Poly-Filter

Defferrard et al. [4] modelled the function using K^{th} order polynomial.

$$m(\Lambda) = \sum_{i=0}^{K} w_k \Lambda^K$$

It was shown by them that equation $Um(\Lambda)U^T$ can be transformed in terms of Laplacian L. Hence there is no need for eigenvalue decomposition of L.

$$\begin{aligned} Um(\Lambda)U^T &= \sum_{i=0}^{K} w_k U\Lambda^K U^T \\ &= \sum_{i=0}^{K} w_k (U\Lambda U^T \cdot U\Lambda U^T \cdots U\Lambda U^T) \\ &= \sum_{i=0}^{K} w_k L^K \end{aligned}$$

Hence the smoothened signal $\hat{X}$ is given by

$$\hat{X} = Um(\Lambda)U^T X = \sum_{i=0}^{K} w_k L^K X.$$

We state the following lemma described by Hammond et al [10].

Lemma 1 *Let G be the graph, and L be the corresponding graph Laplacian matrix. Then following equation holds true*

$$L_{ij}^k = 0 \quad if\ d(v_i, v_j) > k$$

Output signal of node v_i can be given by

$$\hat{X}[i] = \sum_{v_j \in V} \left(\sum_{l=0}^{K} w_i L_{ij}^l \right) X[i]$$

From above lemma we have $L^l_{ij} = 0$ if $d(v_i, v_j) > l$. It follows that only nodes are involved in the calculation that lies in the $K\text{-}th$ hop neighborhood of the node v_i. Since only those nodes are involved in calculating the output feature of node v_i that lies in $K\text{-}th$ hop neighborhood, the Poly-Filter can also be thought of as aggregating information from nodes that lie in $K-th$ hop neighborhood of v_i. Thus, Poly-Filter can also be thought of spatial-based filter.
Poly-Filter has many advantages but suffers from a major drawback. The basis used in Poly-Filter is not orthogonal. Hence parameters become unstable during the learning process.

Chebyshev Filter

To alleviate the above problem, Defferrard et al. [4] proposed to use Chebyshev polynomial instead of the standard basis. Since Chebyshev polynomials are orthogonal in $[-1, 1]$. It makes the coefficients independent and hence are stable during the training process. Chebyshev polynomials are given by following recursive relation

$$T_k(x) = 2kT_{k-1}(x) - T_{k-2}(x)$$

The mapping function m based on Chebyshev polynomial is given by

$$m(\Lambda) = \sum_{i=0}^{K} W_i T_k(\tilde{\Lambda})$$

where $\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I$ is transformed eigenvalue matrix. Output feature after applying Cheby-Filter is given by

$$\hat{X} = \sum_{i=0}^{K} T_i(\tilde{L}) X W_i$$

where $\tilde{L}$ is transformed Laplacian given by

$$\tilde{L} = \frac{2L}{\lambda_{max}} - I$$

One Limitation of Poly-Filer is that it fails to capture sharp changes in the frequency response of the graph. The use of ARMA filter [13] to build graph convolution layer has been proposed by Bianchi et al.[14].

2.6.3 Graph Convolutional Networks (GCN)

Kipf and Welling [14] proposed a simple but effective filter by simplifying Cheby-Filter and using renormalization trick. It has been proved in Spectral Graph Theory (Chung. et al. [1]) that the eigenvalues of normalized graph Laplacian satisfy the following:

$$0 = \lambda_1 < \lambda_2 < \cdots < \lambda_{max} < 2$$

Cheby-Filter was simplified by taking $K = 1$ and $\lambda_{max} = 2$. It follows that

$$m(\Lambda) = w_0 T_0(\tilde{\Lambda}) + w_1 T_1(\tilde{\Lambda}) = w_0 I + w_1(\Lambda - I)$$

Suppose x be a single channel graph signal, then the output signal x' after simplyfying Cheby-Filter is given by

$$\begin{aligned} x' &= U m(\Lambda) U^T x \\ &= w_0 x + w_1 U(\Lambda - I) U^T x \\ &= w_0 x - w_1(L - I)x \\ &= w_0 x - w_1 D^{-1/2} A D^{-1/2} x \end{aligned}$$

where $L = I - D^{-1/2} A D^{-1/2}$ is normalized graph Laplacian. Taking $w = w_0 = -w_1$ we get

$$x' = w(I + D^{-1/2} A D^{-1/2})x$$

Since the eigenvalues of $(I + D^{-1/2} A D^{-1/2})$ lies in $[0, 2]$ stacking GCN might lead to numerical instability. To solve this problem, they proposed a renormalization trick where they replaced A by $\tilde{A}$ and D by $\tilde{D}$, where $\tilde{A}$ and $\tilde{D}$ are adjacency matrix and degree matrix after adding self-loops. Final feature becomes

$$x' = w(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2})x$$

For d_{in} dimensional graph signal $X \in \mathbf{R}^{n \times d_{in}}$ output features after applying GCN is given by

$$X' = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X W$$

where $X' \in \mathbf{R}^{n \times d_{out}}$ and $W \in \mathbf{R}^{d_{in} \times d_{out}}$ is the parameter matrix. Feature vector X'_i for node v_i with feature vector X_i can also be written as

$$X'_i = \sum_{v_j \in N(v_i) \cup \{v_i\}} \frac{1}{\sqrt{d_i d_j}} X_j W$$

Thus GCN can also be viewed as node aggregating information from immediate neighbours.

2.7 Spatial-Based Filter

Now we discuss spatial-based filter.

2.7.1 GraphSAGE

Hamilton et al. [9] proposed GraphSAGE model that is based on sampling and aggregating information from immediate neighbors to update node features. Given a node v_i with feature vector X_i the output feature is generated using the following process:

- Sampling: $N_S(v_i) = \texttt{SAMPLE}(N(v_i), k)$
- Aggregation: $X'_{N_S(v_i)} = \texttt{AGGREGATE}(X_j, \forall v_j \in N_S(v_i))$
- Update: $X'_i = \sigma([X_i, X'_{N_S(v_i)}]W)$

Sampling function `SAMPLE`() samples k nodes from the immediate neighbors of v_i. `AGGREGATE`() uses some aggregation scheme to combine the information from the sampled nodes. Finally, the output feature is generated by concatenating the input feature with the aggregated feature, followed by linear transformation and passing through the activation function. There are different types of aggregation schemes, we list a few of them:

- Mean Aggregator: Mean aggregator simply takes a channel-wise mean of the sampled nodes.

- LSTM Aggregator: LSTM aggregator considers sample features as sequential data and applies LSTM [12] architecture to generate the output feature.

- Pooling Aggregator: Pooling Aggregator applies max, min, or avg pooling to combine the information of the sampled nodes. Before applying Pooling, features are usually transformed by passing through a linear layer followed by an activation layer.

2.7.2 Graph Attention Networks (GAT)

Velikovic et al. [28] introduced attention-based architecture for the node classification tasks. New feature for a node v_i is generated by first assigning importance scores to the nodes in its immediate neighborhood. Based on the importance score, information from the neighboring nodes is aggregated. Let $N(v_i)$ be the set of immediate neighbors of v_i and h_i be the feature vector. Importance score for each $v_j \in N(v_i) \cup \{v_i\}$ is computed as

$$e_{ij} = a(Wh_i, Wh_j)$$

where $a()$ is the shared attention function given by

$$a(Wh_i, Wh_j) = LeakyReLU(b^T[Wh_i, Wh_j])$$

where $[\cdot, \cdot]$ is concatenation operation, $LeakyReLU$ [32] is an activation function, and W, b are parameter matrix. Importance scores are computed by normalizing e_{ij}'s using softmax layer

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{v_k \in N(v_i) \cup \{v_i\}} \exp(e_{ik})}$$

Finally new node feature h'_i is computed using following equation

$$h'_i = \sum_{v_j \in N(v_i) \cup \{v_i\}} \alpha_{ij} W h_i$$

Furthermore K independent attention mechanism are performed in parallel to obtain intermediate features using above equation. Finally all the intermediate features are

concatenated to get the final node feature.

$$h_i' = ||_{k=1}^{K} \sum_{v_j \in N(v_i) \cup \{v_i\}} \alpha_{ij}^k W^k h_i$$

2.7.3 Monti-Filter

Monti et al. [17] proposed a mixture model network (MonNet) to perform convolution operations on graph-structured data. For each $v_j \in N(v_i)$ a pseudo co-ordinate is defined as

$$c(v_i, v_j) = \left(\frac{1}{\sqrt{d_i}}, \frac{1}{\sqrt{d_j}} \right)$$

To measure the relation between the nodes v_i and v_j , Gaussian kernel [18] is applied on the pseudo co-ordinate to get

$$e_{ij} = \exp\left(-\frac{1}{2} (c(v_i, v_j) - \mu)^T \Sigma^{-1} (c(v_i, v_j) - \mu) \right)$$

where both μ, Σ are the parameter matrix to be learned. Finally, node features are generated using following equation

$$h_i' = \sum_{v_j \in N(v_i)} e_{ij} h_j.$$

2.8 Over-Smoothing

Before discussing over-smoothing, we briefly discuss why GCN works? Consider a single-layer GCN applied on feature matrix X. We have

$$Y = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X W$$

Laplacian Smoothing explained in Taubin et al. [18] on feature matrix X with adjacency matrix A is given by

$$Y = (I - \gamma \tilde{D}^{-1} \tilde{L}) X$$

where $\tilde{L} = \tilde{D} - \tilde{A}$ and γ controls the smoothing. If we set $\gamma = 1$ and replace normalized Laplacian $\tilde{D}^{-1}\tilde{L}$ with symmetrically normalized Laplacian $\tilde{D}^{-1/2}\tilde{L}\tilde{D}^{-1/2}$ we get

$$Y = (I - \tilde{D}^{-1/2}\tilde{L}\tilde{D}^{-1/2})X = (I - \tilde{D}^{-1/2}(\tilde{D} - \tilde{A})\tilde{D}^{-1/2})X. = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X$$

This shows that GCN is a special case of Laplacian Smoothing. Nodes that are connected by an edge, Laplacian Smoothing makes their representation similar. Consequently, it helps with classification task. This explains why GCN works really well. Now, if we stack couple of layers of GCN unlike classical Convolutional Network [8] average accuracy drops dramatically. It happens because all the node representation become too much similar to each other. This phenomenon is called Over-Smoothing. The following theorem due to Li et al[13]. formalizes the problem of over-smoothing.

Theorem 3 *Let G be connected non-bipartite graph such that A is its adjacency matrix. Suppose we have a single channel input $h \in \mathbf{R}^n$. Then the following holds true*

$$\lim_{k\to\infty} (\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^k h = \lambda_1 u_1$$

where u_1 is the eigenvector of $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ corresponding to the largest eigenvalue and $\lambda_1 = u_1^T h$.

Proof:

Let $\tilde{L} = I - \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ be normalized Laplacian. The eigenvalues of $\tilde{L}$ are given by $0 = \theta_1 < \theta_2 < \cdots < \theta_n < 2$ with corresponding eigenvectors $u_1, u_2, .., u_n$. In matrix form eigenvalue decomposition of $\tilde{L}$ is given by

$$\tilde{L} = U\Lambda U^T$$

we can write $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ as

$$\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} = I - \tilde{L} = U(I - \Lambda)U^T$$

Hence the eigenvalues of $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is given by $1 = 1-\theta_1 > 1-\theta_2 > \cdots > 1-\theta_n > -1$ with corresponding eigenvectors given by $u_1, u_2, ..., u_n$. Now we have

$$\begin{aligned}
\lim_{k\to\infty}(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^k h &= \lim_{k\to\infty}(U(I-\Lambda)U^T)^k h \\
&= \lim_{k\to\infty} U(I-\Lambda)^k U^T h \\
&= Udiag(1,0,0,,,0)U^T h \\
&= u_1 \cdot (u_1^T h) \\
&= \lambda_1 u_1
\end{aligned}$$

It can be shown that $u_1 = \tilde{D}^{-1}1$. It follows that the node does not contain any other information other than the degree of the node. In the theorem, we didn't use the activation function. However, Oono and Suzuki [19] showed similar results using the ReLU [19] activation function. Interestingly, they showed that the use of ReLU activation actually speeds up the phenomenon of over-smoothing. □

2.9 DropEdge

To tackle over-smoothing Rong et al. [24] proposed the Drop Edge technique; at each training epoch, each edge is selected at random with probability p and is dropped from the adjacency matrix. Following it model is trained on the graph with the new adjacency matrix. Mathematically, it can be written as

$$A_{drop} = Unif(A, 1-p) \tag{2.1}$$

where $Unif()$ uniformly samples each edge and retains the corresponding edge with probability $1-p$. The author also claims that the Drop Edge technique also helps with the problem of over-fitting. The reasoning given is that; Updating node feature can be seen as taking the weighted sum of all the neighboring nodes feature. In each epoch, Drop Edge only allows selecting randomly sampled nodes from the set of neighboring nodes to perform the aggregation. This helps to prevent overfitting.

2.10 Approximate Personalized Propagation of Neural Predictions (APPNP)

Approximate Personalized Propagation of Neural Predictions (APPNP) (Klicpera et al. [15])is inspired by the Google PageRank [20] algorithm that allows the model to go back to the initial residual with a teleportation probability of α. Let X be the input feature matrix, and $f_\theta(\cdot)$ be a fully connected neural network. Initial residuals are calculated using the following equation

$$h_0 = f_\theta(X)$$

In the next step, output features for K iteration are computed as follows:

$$h^{(k+1)} = (1-\alpha)\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}h^{(k)} + \alpha \cdot h_0 \quad for\ 0 \leq k \leq K-2$$

Finally, output features are passed through the softmax layer for classification

$$h^{(K)} = Softmax((1-\alpha)\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}h^{(K-1)} + \alpha \cdot h_0)$$

Till now we have discussed shallow models in Graph Neural Networks. Attempts have been made to build deep models like ResGCN [22], IncepGCN [24], JKNet [33], etc in Graph Neural Networks.

2.11 Geometric GCN

Geometric GCN (Pei et al. [21]) employs a three-stage aggregation scheme to update the node features. We describe them below.

- Node embedding: This is a basic module that maps each node v to the latent space. Give a node v with feature vector x_v, the mapping function is given by

  $$f : v \rightarrow z_v$$

  We can think z_v to be the node's position in the latent space.

- Structural neighborhood: Next phase of aggregation builds structural neigh-

borhood on the basis of latent space feature and node's feature, and graph geometry. given by

$$N(v) = \{N_g(v), N_s(v), \tau\}$$

. Here $N_g(v)$ is the set of immediate neighbors of v, $N_s(v) = \{u \ \ |d(z_v, z_u) < \rho\}$ is the neighbors of v in latent space and τ is relation operator. Relational operator τ is a function by

$$\tau(z_u, z_v) \rightarrow r$$

. Here $r \in R$ is geometric relationship between latent variables.

- Bi-level aggregation: Final aggregation combines two functions defined above. Let $h_v^0 = x_v$, the node's feature h_v^l at th layer is updated by

$$\begin{aligned} e_{(i,r)}^{v,l+1} &= p(\{h_u^l \ \ |u \in N_i(v), \tau(z_u, z_v) = r\}), \forall i \in \{g, s\}, i \in R. \\ m_v^{l+1} &= q_{i\in\{g,s\},i\in R}(e_{(i,r)}^{v,l+1}, (i, r)) \\ h_v^{l+1} &= \sigma(W_l \cdot m_v^{l+1}). \end{aligned}$$

The first equation says that nodes that have the same geometric relationship r and that are in the same neighborhood i are aggregated using some aggregation function p. Generated features in the first equation are then aggregated in the next equation using aggregation function q. Finally, intermediate feature m_v^{l+1} is passed through linear layer and subsequently through non-linear activation function to get the updated feature h_v^{l+1}.

2.12 Jumping Knowledge Network (JKNet)

Jumping Knowledge Network (Xu et al. [33]) is an architecture that learns to combine information from different neighborhoods to generate a feature vector. As we discussed, stacking k layers of GCN can be viewed as aggregating information from k-hop neighborhood. Suppose $h_1, h_2, ..., h_k$ are intermediate feature vector generated by k layers of GCN such that i^{th} block generates h_i. These feature vectors can be viewed as information aggregated from different localities. JKNet combines all these feature vectors by some aggregation scheme to produce the final output feature. This allows the architecture to learn to combine information aggregated from different

neighborhoods.
Some commonly used aggregation schemes are

- Concatenation: One simple way to generate output feature is to concatenate all intermediate features $\texttt{CONCATENATE}(h_1, h_2, ..., h_K)$. Sometimes it's desirable to pass the concatenated feature through a linear layer.

- Max Pooling: We could perform channel-wise max-pooling to produce the output feature. For i^{th} channel, i^{th} channel of output feature is given by

$$h^i = max(\{h_l^i \quad : \ 1 \leq l \leq k\})$$

  Channel that represents global property of the graph can learn from higher order neighbors.

- LSTM attention: For each node v attention mechanism identifies the useful neighborhood range. For each node v the features $h_1^{(v)}, h_2^{(v)}, ..., h_k^{(v)}$ are fed to a bi-LSTM [8] to generate forward and backward features f_l^v, f_l^v for each layer respectively. Following it concatenated feature $[f_l^v, f_l^v]$ is passed through a linear layer to produce importance score s_l^v. Finally $\{s_l^v\}$ is passed through softmax layer to get attention scores. Hence final feature h^v is computed as

$$h = \sum_{l=1}^{K} Softmax(s_l^v) \cdot h_l^v$$

  .

2.13 GCNII

Chen et al. [2] combined the ideas mentioned below to build a deep architecture for Graph Neural Networks.

- Initial Residual Connection: This idea is borrowed from APPNP [15] that allows the model to go back to initial residuals with teleportation probability α to generate a feature vector.

- Identity Mapping: The idea of Identity mapping was introduced in ResNet [11] to alleviate the problem of vanishing gradient in a very deep model. This allowed model to go deep and achieve great representational capacity.

Let X be the feature matrix. Feature matrix is passed through a fully connected layer to obtain initial residuals h_0.

$$h_0 = f_\theta(X)$$

For a L layer model update rule is given by

$$h^{(l+1)} = \sigma\left(\left((1-\alpha_l)\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}h^{(l)} + \alpha_l h_0\right)\left((1-\beta_l)I_n + \beta_l W_l\right)\right)$$

where α_l controls the effect of initial residuals h_0 at layer l and β_l controls the effect of identity mapping. $\beta_l = \log(\lambda/l + 1)$ where λ is a hyperparameter.

Chapter 3

Proposed Approach

3.1 Preliminary

For a graph G with features matrix X, the graph filtering operation is given by

$$X' = U\gamma(\Lambda)U^T.$$

Here $L = U^T\Lambda U$ is the spectral decomposition of the graph Laplacian L and γ is a function that modulates the frequency component of the graph signal. While designing the spectral filter, one obvious approach is to give full freedom to the set of trainable parameters, that is

$$\gamma(\lambda_k) = w_k$$

One limitation of the above approach is for a large graph number of trainable parameters will be huge, making it difficult to train. To address this problem polynomial filtering(Poly-Filter) operation was proposed by Defferrard et al. The function γ is given by truncated K^{th} order polynomial

$$\gamma(\Lambda) = \sum_{i=0}^{K} \Theta_i \Lambda^i$$

Poly-Filter is a simple and effective spectral-based filter that offers many advantages. One limitation of Poly-Filter is that the basis used is not orthogonal, and hence the coefficients are dependent on each other, making them unstable under updates

during the learning process. To alleviate this problem, we propose Legendre Filter (Leg-Filter) based on the Legendre polynomial.

3.2 Legendre Filter

We first briefly discuss the properties of Legendre polynomial [23]. Legendre Polynomials are a set of orthogonal polynomials that satisfy the following recurrence relation:

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x) \; for \; n \geq 2$$

with $P_0(x) = 1, P_1(x) = x$. Legendre polynomials are orthogonal in $[-1, 1]$ given by following equation,

$$\begin{aligned} \int_{-1}^{1} P_n(x)P_m(x) &= \frac{2}{2n+1} \quad if \; n = m \\ &= 0 \quad if \; n \neq m \end{aligned}$$

Hence the function γ is given by

$$\gamma(\Lambda) = \sum_{i=0}^{K} \Theta_i P_i(\Lambda)$$

Since Legendre Polynomials are orthogonal on range $[-1, 1]$ we shift the eigenvalues of graph Laplacian in the same range, hence shifted eigenvalue matrix Λ is given by

$$\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I$$

For a graph with feature matrix X, feature X' obtained after applying Legendre filter is given by

$$X' = \sum_{i=0}^{K} U P_i(\tilde{\Lambda}) U^T X \Theta_i$$

Lemma 2 *For a graph G with Laplacian L following relation is true $\forall i \geq 0$.*

$$U P_i(\tilde{\Lambda}) U^T = P_i(\tilde{L})$$

Here modified Laplacian $\tilde{L} = \frac{2L}{\lambda_{max}} - I$.

Proof:

We prove the lemma using induction. For $i = 0$ we have

$$UP_0(\tilde{\Lambda})U^T = UU^T = I.$$

For $i = 1$ we have,

$$\begin{aligned} UP_1(\tilde{\Lambda})U^T &= U\tilde{\Lambda}U^T = U\left(\frac{2\Lambda}{\lambda_{max}} - I\right)U^T \\ &= \frac{2U\Lambda U^T}{\lambda_{max}} - UU^T \\ &= \frac{2L}{\lambda_{max}} - I = \tilde{L} = P_1(\tilde{L}). \end{aligned}$$

Suppose for $i = k-1$ and $i = k$ the equation holds true.

$$\begin{aligned} UP_{k-1}(\tilde{\Lambda})U^T &= P_{k-1}(\tilde{L}) \\ UP_k(\tilde{\Lambda})U^T &= P_k(\tilde{L}) \end{aligned}$$

For $i = k+1$,

$$\begin{aligned} UP_{k+1}(\tilde{\Lambda})U^T &= U\left(\frac{1}{k+1}\left((2k+1)\tilde{\Lambda}P_k(\tilde{\Lambda}) - kP_{k-1}(\tilde{\Lambda})\right)\right)U^T \\ &= \frac{2k+1}{k+1}U\tilde{\Lambda}U^T UP_k(\tilde{\Lambda})U^T - \frac{k}{k+1}UP_{k-1}(\tilde{\Lambda})U^T \\ &= \frac{2k+1}{k+1}\tilde{L}P_k(\tilde{L}) - \frac{k}{k+1}P_{k-1}(\tilde{L}) \\ &= \frac{1}{k+1}\left((2k+1)\tilde{L}P_k(\tilde{L}) - kP_{k-1}(\tilde{L})\right) \\ &= \frac{1}{k+1}(k+1)P_{k+1}(\tilde{L}) = P_{k+1}(\tilde{L}). \end{aligned}$$

For $i = k+1$ equation holds true, hence form induction the equation is true for all $i \geq 0$. □

We define unnormalized Legendre Convolution operation on graph G with feature

matrix X as

$$X' = \sum_{i=0}^{K} P_k(\tilde{L}) X \Theta_i.$$

We further normalize the Legendre polynomial $P_k(x)$ by multiplying it by a factor of $\sqrt{\frac{2}{2k+1}}$,

$$\tilde{P}_k(\tilde{L}) = \sqrt{\frac{2}{2k+1}} P_k(\tilde{L})$$

Finally, we define the normalized Legendre Convolution operation on graph G with feature matrix X:

$$X' = \sum_{i=0}^{K} \tilde{P}_k(\tilde{L}) X \Theta_i.$$

3.2.1 Special Case

For simplicity, we consider a single-channel graph signal. Filter operation on single-channel graph for $k = 2$ is given by

$$\gamma(\tilde{\Lambda}) = \theta_0 I + \theta_1 \tilde{\Lambda} + \frac{\theta_2}{2}(3\tilde{\Lambda}^2 - I)$$

We take following assumptions; $\theta_2 = 2\theta_0, \theta_1 = 3\theta_0, \theta_0 = \theta$. It follows that,

$$\begin{aligned} \gamma(\tilde{\Lambda}) &= \theta I + 3\theta\tilde{\Lambda} + 2\frac{\theta}{2}(3\tilde{\Lambda}^2 - I) \\ &= 3\theta(\tilde{\Lambda} + \tilde{\Lambda}^2) \end{aligned}$$

We consider normalized graph Laplacian L given by $L = I - D^{-1/2}AD^{-1/2}$. Eigenvalues of the graph Laplacian L are $0 = \lambda_1 < \lambda_2 \cdots < \lambda_n < 2$. Hence, we make the following approximation:

$$\lambda_{max} = 2$$

It follows that $\tilde{\Lambda} = \Lambda - I$. Therefore we have following equation,

$$\begin{aligned} U\gamma(\tilde{\Lambda})U^T &= 3\theta U(\Lambda - I + (\Lambda - I)^2)U^T \\ &= 3\theta U((\Lambda - I + \Lambda^2 - 2\Lambda + I)U^T \\ &= 3\theta U(\Lambda^2 - \Lambda)U^T \\ &= 3\theta(L^2 - L) \\ &= \theta'(-(I - D^{-1/2}AD^{-1/2})D^{-1/2}AD^{-1/2}) \\ &= \theta'((D^{-1/2}AD^{-1/2})^2 - D^{-1/2}AD^{-1/2}) \end{aligned}$$

For a graph with a d-dimensional feature above operation takes the following form:

$$X' = ((D^{-1/2}AD^{-1/2})^2 - D^{-1/2}AD^{-1/2})X\Theta$$

Legendre [31], Chebychev [29], and other orthogonal polynomials are special cases of Jacobi polynomial. Hence, we generalize the Legendre filter and consequently define Jacobi Filter.

3.3 Jacobi Filter

First, we briefly discuss some properties of the Jacobi polynomial [30]. Jacobi polynomials are given by following recurrence relation,

$$\begin{aligned} 2n(n+\alpha+\beta)(2n+\alpha+\beta-2)P_n^{(\alpha,\beta)}(x) &= (2n+\alpha+\beta-1)\{(2n+\alpha+\beta) \\ &\quad (2n+\alpha+\beta-2)x + \alpha^2 - \beta^2)P_{n-1}^{(\alpha,\beta)}(x)\} \\ &\quad -2(n+\alpha+1)(n+\beta+1) \\ &\quad (2n+\alpha+\beta)P_{n-2}^{(\alpha,\beta)}(x) \quad for\ n \geq 2 \end{aligned}$$

Jacobi Polynomial for $n = 0, 1$ are given by

$$\begin{aligned} P_0^{(\alpha,\beta)}(x) &= 1 \\ P_1^{(\alpha,\beta)}(x) &= (\alpha+1) + (\alpha+\beta+1)\frac{x-1}{2} \end{aligned}$$

Jacobi Polynomials satisfies the following orthogonality condition:

$$\int_{-1}^{1}(1-x)^{\alpha}(1-x)^{\beta}P_n^{(\alpha,\beta)}(x)P_m^{(\alpha,\beta)}(x) = \frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1}\frac{\Gamma(\alpha+n+1)\Gamma(\beta+n+1)}{\Gamma(n+\alpha+\beta+1)n!}\delta_{nm}$$

For a graph with feature matrix X, application of Jacobi Filter on feature matrix results in X' given by

$$X' = \sum_{i=0}^{K} U P_i^{(\alpha,\beta)}(\tilde{\Lambda})U^T X\Theta_i$$

Using the above lemma, the unnormalized Jacobi Convolution operation is given by

$$X' = \sum_{i=0}^{K} P_k^{(\alpha,\beta)}(\tilde{L})X\Theta_i.$$

We normalize the Jacobi Polynomial to get,

$$P_n^{(\tilde{\alpha},\beta)} = \sqrt{\frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1}\frac{\Gamma(\alpha+n+1)\Gamma(\beta+n+1)}{\Gamma(n+\alpha+\beta+1)n!}}P_n^{(\alpha,\beta)}$$

Finally, we define the Jacobi Convolution operation on graph G with feature matrix X:

$$X' = \sum_{i=0}^{K} P_n^{(\tilde{\alpha},\beta)}(\tilde{L})X\Theta_i.$$

3.4 Aggregation Schemes

As we discussed earlier Legendre Filter of order K can be thought as aggregating information upto K-th hop neighbors. We propose an aggregation scheme to update nodes's feature. We fix the order (K) of the Legendre Filter. For $k \leq K$ and node v, let $h_v^{(k)}$ denote the node feature after applying Legendre Filter of order k. For each node we have $h_v^{(1)}, h_v^{(2)}, h_v^{(3)}, ..., h_v^{(K)}$. Updated node feature h'_v can be given as

$$h'_v = \texttt{AGGREGATION}(\{h_v^{(1)}, h_v^{(2)}, h_v^{(3)}, ..., h_v^{(K)}\})$$

Following can we used to aggregate collected node features:

- **Mean Pooling**: We could simply take mean of $\{h_v^{(1)}, h_v^{(2)}, h_v^{(3)}, ..., h_v^{(K)}\}$ to up-

date the feature of the node v.

- **Stochastic Mean Pooling**: We fix some feature $h_0 = h_v^l$ for $l \leq K$. Next, during training we choose each node feature h_v^i for $i \neq l$ with some probability p. Suppose we get $\{h_v^{k_1}, h_v^{k_2}, ..., h_v^{k_n}\}$ after choosing each feature with probability p. Finally we take mean of $\{h_0, h_v^{k_1}, h_v^{k_2}, ..., h_v^{k_n}\}$ to update node feature v.

- **LSTM Aggregator**: Features $\{h_v^{(1)}, h_v^{(2)}, h_v^{(3)}, ..., h_v^{(K)}\}$ can be treated as sequential data since h_v^i represent information aggregated upto i-th hop neighbors. Updated node feature is given as

$$h'_v = \sum_{i=1}^{K} \alpha_i h_v^i$$

  Following the approach of JKNet [33], we learn the weights $\alpha'_i s$ using LSTM architecture. Node features $\{h_v^{k_1}, h_v^{k_2}, ..., h_v^{k_n}\}$ is fed to LSTM and output of each LSTM block is passed through linear layer to get the score s_v^i. The score s_v^i is passed through the Softmax layer to get the weights $\alpha'_i s$.

3.5 Adaptive Edge Algorithm

In this part of the work, we focus on Graph Topology to improve the accuracy of the node classification task. During our experiments, we observed that the average accuracy of higher degree nodes was significantly greater than the lower degree nodes. The results of these experiments can be found in the next chapter. This led to the following conjecture:

Conjecture 1 *The average classification accuracy of nodes with a higher degree is significantly greater than nodes with a lower degree.*

We develop an adaptive Edge algorithm that changes the graph topology by adding supernodes to the graph and also using various parameters like degree information and predictive confidence.
Now, we propose the algorithm. The algorithm is divided into two phases. First, we fix a particular model and a dataset. We train the model for a maximum of k times, and the model is selected amongst them with the best validation accuracy for

the next phase. In the second phase of the algorithm, we add as many supernodes to the graph as there are classes in the dataset. Next, we uniquely assign a label to each of the supernodes. Now we add edges between training nodes and supernodes belonging to the same class. For the rest of the nodes, we first get the predicted class label by the model selected in the first phase. Next, we add edges between them and supernodes based on the predicted class label and degree info. We also remove interclass edges between the nodes based on predicted label and degree info. Finally, we train the model on the modified graph. Pseudo Code for the same is given below.

Algorithm 1 `BEST_MODEL(INPUT)`

1. *for* $i = 1$ *to* K :.
2. $Initialize\ model, best\_val = 9999.$
3. $val\_loss = model.fit(data).val\_loss().$
4. $if\ val\_loss < best\_val$:
5. $best\_model = model$
6. *return best_model*

Algorithm 2 `ADD_EDGE()`

1. *for i in data.num_num_classes* :
2. *add new node* v_i *to graph.*
3. $Initialize\ feature\ of\ v_i, set\ class\ label\ i.$
4. *for each v in data.training_set* :
5. $if\ v.class = i$:
6. $add\ edge\ e = (v, v_i)$
7. *model* = `BEST_MODEL()`:
8. *for each v not in data.training_set*:
9. $label = model.predict(v).$
10. $if\ deg[v] \geq deg\_conf\ and\ pred[v] \geq pred\_conf$:
11. $add\ edge\ e = (v, v_{label})$
12. *return data.*

Algorithm 3 `REMOVE_EDGE()`

1. *for each e indata.edge_list* :
2. *let* $e = (v, w)$.
3. *if* $deg[v] \geq deg\_conf$ *and* $deg[w] \geq deg\_conf$:.
4. *if* $pred[v] \geq pred\_conf$ *and* $pred[w] \geq pred\_conf$:
5. *Remove edge e from data.edge_list.*
6. *return data.*

Now we define the terms used in the algorithm. `BEST_MODEL()` runs the model K times and returns the model with the best validation accuracy. The deg_conf allows us to connect only higher degree vertices with supernode. Similarly, $pred\_conf$ allows us to only those vertices whose log-softmax prediction is above the threshold. During our experiments, we observed that the algorithm works well on models with high classification accuracy.

Chapter 4

Experiments and Results

In this section, we measure the performance of the proposed Legendre Filter and Adaptive Edge algorithm. We also report the results and plots of other experiments.

4.1 DATASET

For the semi-supervised node classification task, we use the benchmark standard Citation Network dataset [34] - Cora, Citeseer, PubMed. In this dataset, nodes are the documents, and edges correspond to the citations. Node features are the bag-of-word representation of the document. Each node's label belongs to one of the academic topics. We also use Cora_ ML, and DBLP from the full Citation Network dataset [7]. Next, we use the Coauthor CS dataset [26]. Here node represents the author, and the edges between two nodes exist if they Co-authored a paper. Each node's label belongs to its respective field of study. Finally, we use the Amazon Photo dataset [26]. Here, nodes represent goods, and there is an edge between the nodes if both of them are bought frequently. The class label of each node is their respective product category.
For the full-supervised node classification task, we also included Web Network datasets [21]- Cornell, Wisconsin, and Texas along with the Citation Network datasets. Here, each node is a web page, and edges correspond to hyperlinks. Each node's feature is a bag-of-word representation of the respective web page. Each node's label belongs to one of five categories- student, project, course, staff, and faculty.

Table 4.1: Dataset Statistics

Datasets	Nodes	Edges	Features	Classes
Cora	2,708	10,556	1,433	7
Citeseer	3,327	9,104	3,703	6
PubMed	19,717	88,648	500	3
Cora_ML	2,995	16,316	2,879	7
DBLP	17,716	105,734	1,639	4
Coauthor CS	18,333	163,788	6,805	15
Amazon Photo	7,650	238,162	745	8
Cornell	183	295	1,703	5
Wisconsin	259	499	1,703	5
Texas	183	309	1,703	5

4.2 OverSmoothing Experiment

It has been shown that a Poly-Filter of degree K can be simulated by stacking K layers of GCN. In this experiment we plot average accuracy of GCN, Legendre-Filter and Cheby-Filter for different values of K ($1 \leq K \leq 10$). For each value of K average accuracy for each model is reported after performing 100 runs of the experiment. It can be seen that GCN suffers hugely from oversmoothing. As for the Poly-Filter our proposed Legendre-Filter performs significantly better compared to Cheby-Filter for large values of K.

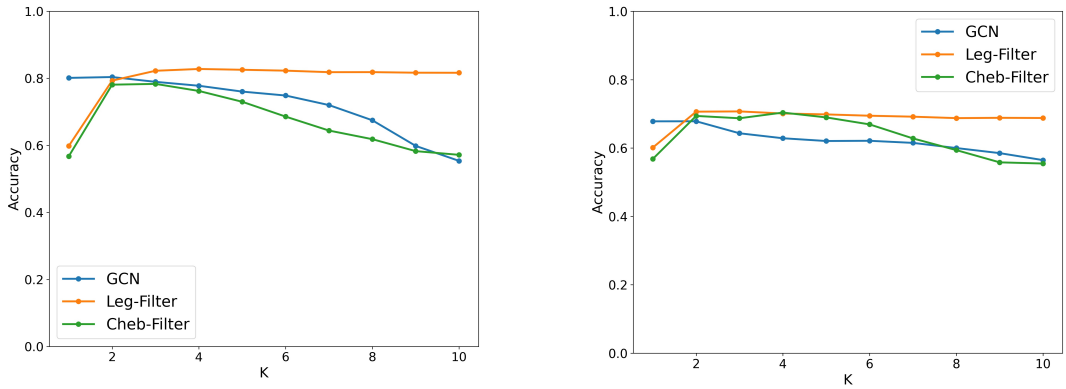

Figure 4.1: From left to right: Cora and CiteSeer. Average Accuracy of GCN, Legendre-Filter and Cheby-Filter for different values ogf K.

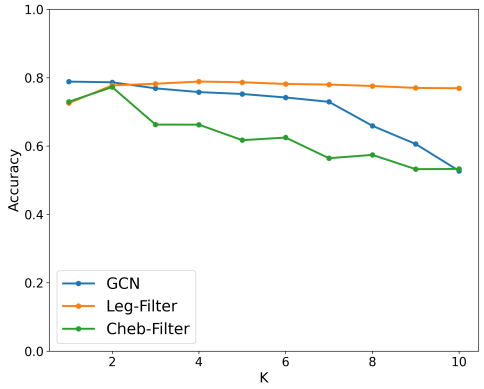

Figure 4.2: PubMed Dataset. Average Accuracy of GCN, Legendre-Filter and Cheby-Filter for different values of K.

4.3 Node Degree vs Accuracy Test

In this section we present the empirical results supporting *Conjecture* 1. For the experiment we fixed the $degree = k$ for the given dataset. Following this, we computed the average accuracy of the nodes corresponding to $degree = k$. We repeated this process for all possible node degree's in the graph. Citation network dataset was used for this experiment, and Leg-Filter was used to train the model. Results are mentioned in the Table 4.2.

Table 4.2: Average accuracy of Lower-Degree nodes (all the nodes whose degree$\leq$ 3) and Higher-Degree nodes(all the nodes whose degree$>$ 3).

Datasets	**Lower-Degree** (Avg.Accuracy)	**Higher-Degree** (Avg.Accuracy)	**Num-Nodes** (Lower Degree)	**Num-Nodes** (Higher Degree)
Cora	79.59	85.43	1068	1640
Citeseer	65.09	76.84	2174	1153
PubMed	76.57	83.21	12451	7266

We also observed that overall accuracy of the dataset is better if the dataset has greater number of Higher-Degree nodes.

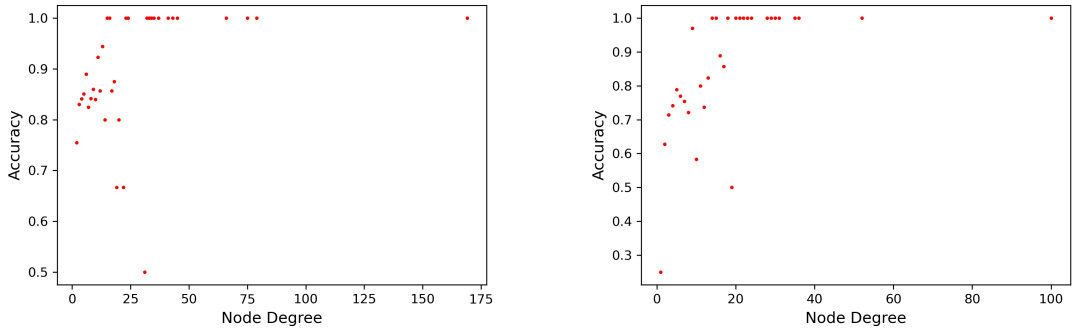

Figure 4.3: From left to right: Cora and CiteSeer. Average Accuracy of nodes is plotted for different node degree.

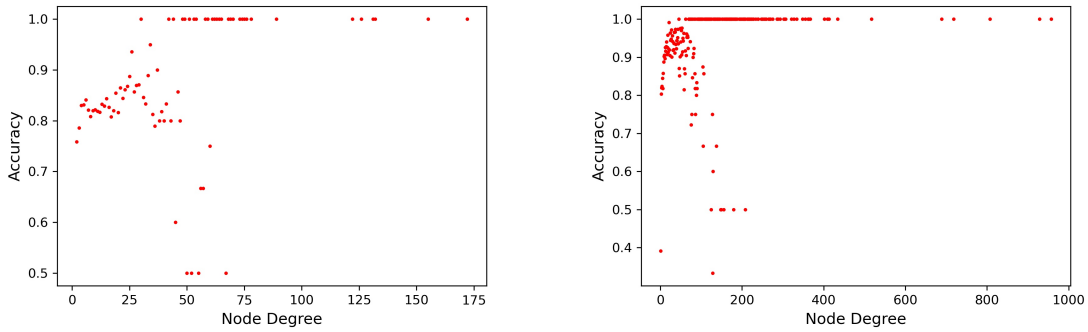

Figure 4.4: From left to right: PubMed and Amazon Photos. Average Accuracy of nodes is plotted for different node degree.

4.4 Semi-Supervised Node Classification

- **Baselines and Experimental Setup**: For Semi-Supervised node classification task we use the fixed standard training/validation/testing splits for Citation network-Cora, CiteSeer, PubMed dataset as taken in GCN [Kipf & Welling]. We take 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for testing. For Cora_ML, DBLP, Amazon Photo, and Coauthor CS we took our fixed training/validation/testing splits. In this work we compare our proposed Legendre-Filter and Legendre-Filter with stochastic mean aggregation(Legendre Filter-M) with Cheby-Filter,GCN and GAT, but we also include other SOTA; PPNP, IncepGCN, JKNET, GCNII for reference. We used Pytorch Geometric [6] to implement out proposed Legendre Filter. For optimization, we use Adam Optimizer. We use elu activation function for

all tasks. All the hyperparameters were tuned based on the validation set. Early stopping is used if validation loss does not decrease after 100 epochs. On all the datasets model is trained for 200 epochs. Details on hyperparameters can be found in Table 4.6.

- **Comparisions with SOTAs**: We used Normalized Legendre Filter for all of the classification task. We also trained Legendre Filter with stochastic mean pooling (Legendre Filter-M) on cora dataset. Table 4.3 reports mean classification accuracy along with the standard deviation on the test set of different models after performing 100 runs of the experiment. For all the models we reuse the metrics provided in Chen et al. For Chebyshev Filter we reuse the metrics provided in Kipf & Welling. On Cora our model performs better than Chebyshev Filter and GCN. On Citeseer and PubMed our model performs better than Chebyshev, GCN and GAT. Interestingly, on PubMed our shallow model performs marginally better than deep models like JKNet and IncepGCN. On Amazon Photo and Cora_ML Legendre filter performs better than Chebyshev Filter and GAT, but performs marginally poor than GCN. On Coauthor CS Legendre filter performs better than all three baselines. Finally, on DBLP Legendre Filter performs significantly better than all three baselines.

Table 4.3: For Semi-Supervised Learning task: Average Classification accuracy results(%) on Citation Network Dataset; Cora, CiteSeer and PubMed. Number in the parenthesis refer to number of layers used in corresponding deep models.

Method	**Cora**	**CiteSeer**	**PubMed**
Chebyshev Filter	81.2	69.8	74.4
GCN	81.5	70.3	79.0
GAT	83.1	70.8	78.5
APPNP	83.3	71.8	80.1
JKNet	81.1 (4)	69.8 (16)	78.1 (32)
JKNet(Drop)	83.3 (4)	72.6 (16)	79.2 (32)
Incep(Drop)	83.5 (64)	72.7 (4)	79.5 (4)
GCNII	85.5 (64)	73.4 (32)	80.2(16)
Legendre Filter	**82.67** $\pm$0.5	**71.35** $\pm$ 0.6	**79.73** $\pm$ 0.4
Legendre Filter-M	**82.88** $\pm$0.6	**-**	**79.86** $\pm$0.5

Table 4.4: For Semi-Supervised Learning task : Average Classification accuracy results(%) on Amazon Photos (Amz.), Coauthor CS (CoCS.), Cora_ML and DBLP. Results are reported on our fixed training/validation/testing split.

Method	**Amz.**	**CoCS.**	**Cora_ML**	**DBLP**
Chebychev Filter	86.68	91.14	76.14	75.72
GCN	91.64	92.56	81.59	71.17
GAT	89.55	92.16	80.93	72.25
Legendre Filter	**90.62** $\pm$0.5	**92.89**$\pm$0.3	**81.17**$\pm$0.7	**77.06**$\pm$1.8

4.5 Full Supervised Node Classification

- **Baselines and Experimental Setup**: Again, we use a normalized Legendre Filter for this task. We use 6 datasets for the task- Cora, CiteSeer, PubMed, Cornell, Wisconsin and Texas. For each of the datasets, in each run we randomly split nodes of each class such that 60%, 20% and 20% is for training, validation and testing, respectively. Classification accuracy is measured by taking an average of accuracy over 10 runs. Adam optimizer is used for optimization. Early stopping is used if validation loss does not decrease after 100 epochs. On all the datasets model is trained for 200 epochs. Details of hyperparameter can be found in Table 4.5.

- **Comparisions with SOTAs** We use a normalized Legendre Filter for all the datasets. Average classification accuracy for all the datasets after performing 10 runs is reported in Table 4.5. On all the datasets Legendre filter performs better than all the shallow models. On Cora, Citeseer and Pubmed our model performance is comparable with the deep models. On Cornell our model performs better than almost all the deep models. On Texas and Wisconsin our model performs better than all the deep models.

Table 4.5: For Full-Supervised Lrearning Task: Average Classification accuracy results(%) on Cora, CiteSeer, PubMed, Cornell, Texas and Wisconsin.

Method	**Cora**	**CiteS.**	**PubM.**	**Corn.**	**Texa.**	**Wisc.**
GCN	85.77	73.68	88.13	52.70	52.16	45.88
GAT	86.37	74.32	87.62	54.32	58.38	49.41
Geom-GCN-I	85.19	77.99	90.05	56.76	57.58	58.24
Geom-GCN-P	84.93	75.14	88.09	60.81	67.57	64.12
Geom-GCN-S	85.27	74.71	84.75	55.68	59.73	56.67
APPNP	87.87	76.53	89.40	73.51	65.41	69.02
JKNet	85.25 (16)	75.85 (8)	88.94 (64)	57.30 (4)	56.49 (32)	48.82 (8)
JKNet(Drop)	87.46 (16)	75.96 (8)	89.45 (64)	61.08 (4)	57.30 (32)	50.59 (8)
Incep(Drop)	86.86 (8)	76.83 (8)	89.18 (4)	61.62 (16)	57.84 (8)	50.20 (8)
GCNII	88.49 (64)	77.08 (64)	89.57 (64)	74.86 (16)	69.46 (32)	74.12 (16)
GCNII*	88.01 (64)	77.13 (64)	90.30 (64)	76.49 (16)	77.84 (32)	81.57 (16)
Legendre Filter	**88.45**	**77.34**	**89.67**	**76.48**	**80.81**	**84.4**

4.6 Hyper-Parameter Details

We used two layers of Legendre filter for all the datasets and experiments. We only used dropout on second layer for all the datasets and experiments. We used elu activation function. K denotes the order of Legendre polynomial and weight decay is regularization factor.

Table 4.6: Hyper-parameters of Legendre Filter for Table 4.3 and Table 4.4.

Datasets	**Hyper-parameters**
Cora	K = 5, lr: 0.005, hidden: 16, dropout: 0.8, weight decay = 0.0005
Cora (Leg-M)	K = 6, lr: 0.005, hidden: 16, layer1 dropout: 0.4, layer2 dropout: 0.8, mean pool prob. = 0.8, i= 5 ($h_0 = h_v^i$), weight decay = 0.0005
Citeseer	K = 4, lr: 0.01, hidden: 16, dropout: 0.6, weight decay = 0.005
PubMed	K = 4, lr: 0.01, hidden: 16, dropout: 0.8, weight decay = 0.001
PubMed (Leg-M)	K = 5, lr: 0.01, hidden: 16, layer1 dropout: 0.4, layer2 dropout: 0.8, mean pool prob. = 0.9, i= 5 ($h_0 = h_v^i$), weight decay = 0.0001
Amaz.	K = 4, lr: 0.01, hidden: 16, dropout: 0.8, weight decay = 0.00005
Coauth.	K = 4, lr: 0.005, hidden: 16, dropout: 0.8, weight decay = 0.0005
Cora_ML	K = 4, lr: 0.01, hidden: 16, dropout: 0.8, weight decay = 0.0005
DBLP	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.0005

Table 4.7: Hyper-parameters of Legendre Filter for Table 4.5.

Datasets	Hyper-parameters
Cora	K = 5, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.0005
Citeseer	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.0005
PubMed	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.0001
Corn.	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.00005
Wisc.	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.00005
Texa.	K = 4, lr: 0.05, hidden: 16, dropout: 0.8, weight decay = 0.0001

4.7 Adaptive Edge experiment

We perform semi-supervised learning using adaptive edge algorithm on Amazon datset. Accuracies before and after applying adaptive edge algorithm using GCN, Legendre Filter and GAT are mentioned in table.

Table 4.8: For Semi-Supervised Learning task : Average Classification accuracy results(%) on Amazon Photos are performing 100 runs of experiment. Results are reported on out fixed training/validation/testing split.

AdapEdge	**GCN**	**Leg-Filter**	**GAT**
Before	91.64	90.62	89.55
After	**92.41** ±0.1	**90.89** ±0.1	**91.85** ±0.5

4.8 T-SNE Plots

In this section we report t-SNE plots for last layer of Legendre Filter for all the datasets used in semi-supervised learning. Plots refer to two dimensional t-SNE plots of the outputs of last layer of Legendre filter for different datasets.

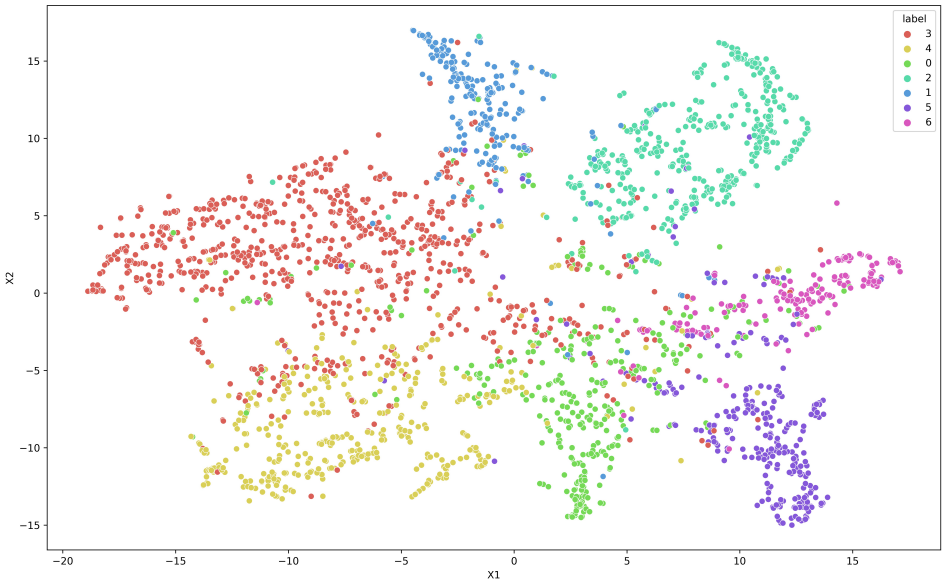

Figure 4.5: Dataset: Cora

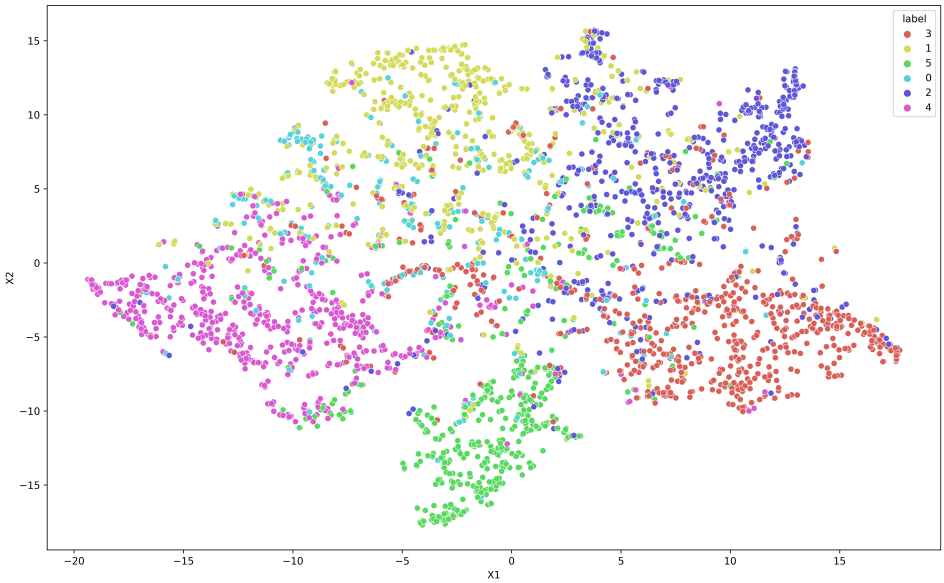

Figure 4.6: Dataset: Citeseer

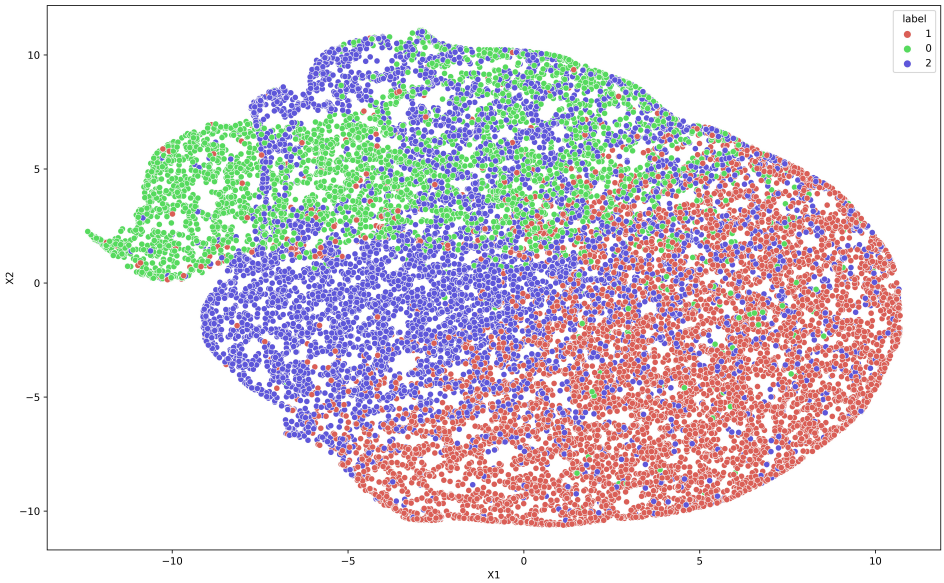

Figure 4.7: Dataset: PubMed.

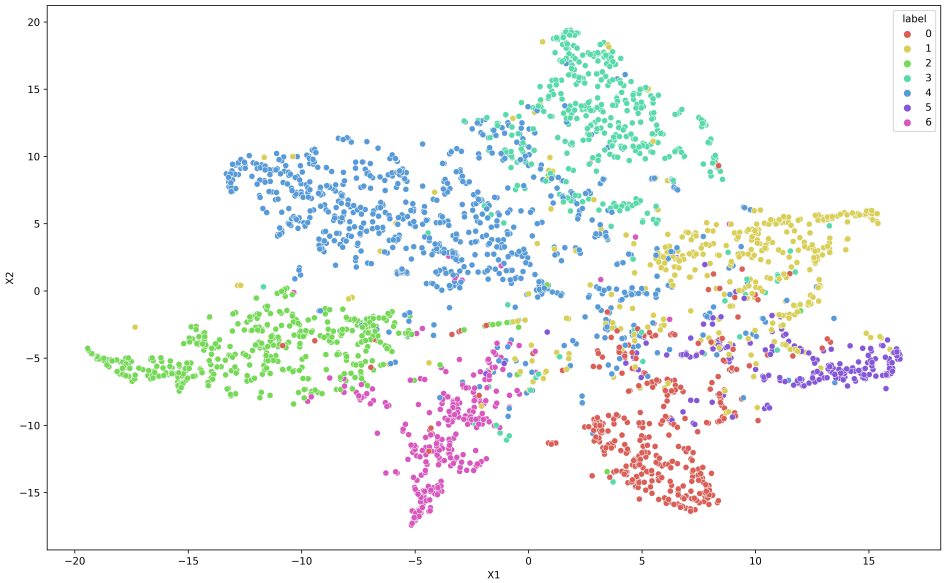

Figure 4.8: Dataset: Cora_ML.

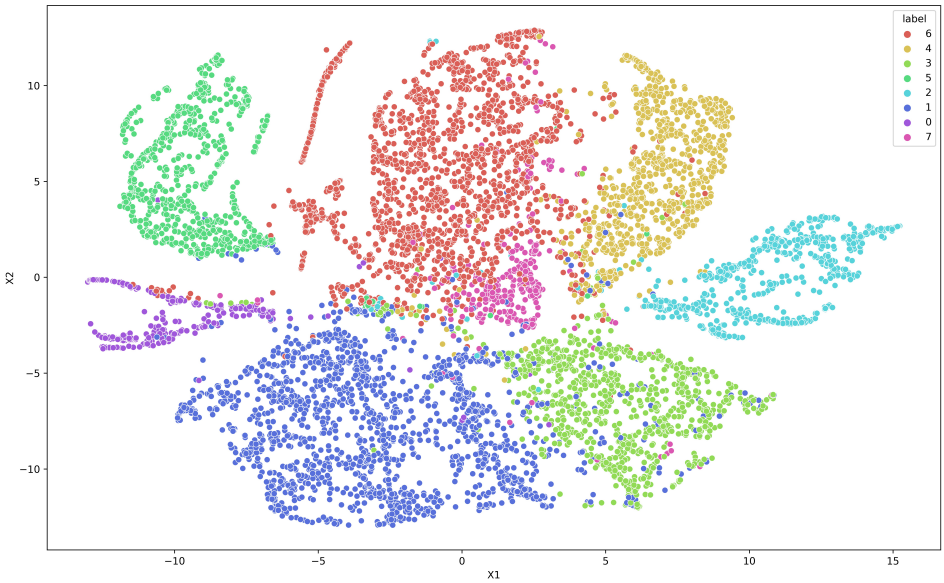

Figure 4.9: Dataset: Amazon Photo.

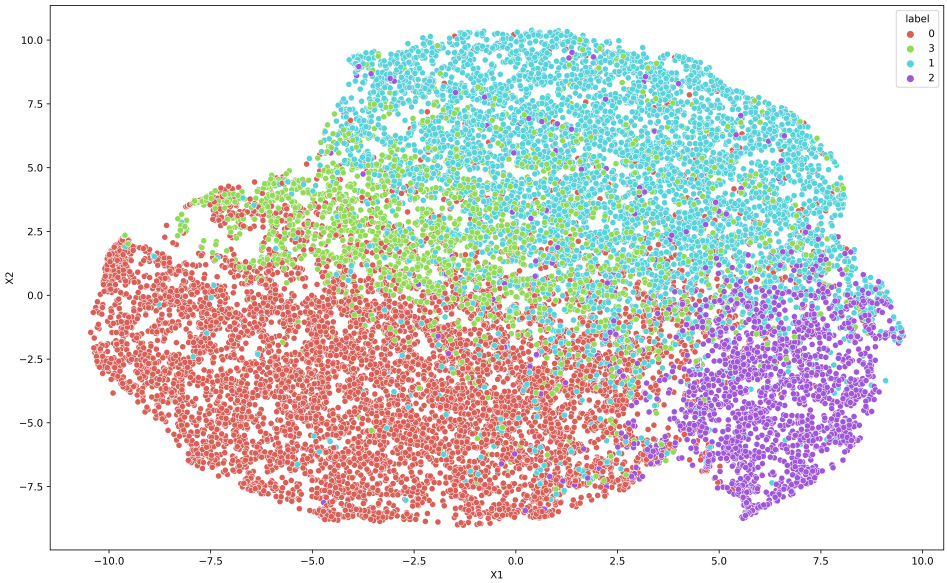

Figure 4.10: Dataset: DBLP.

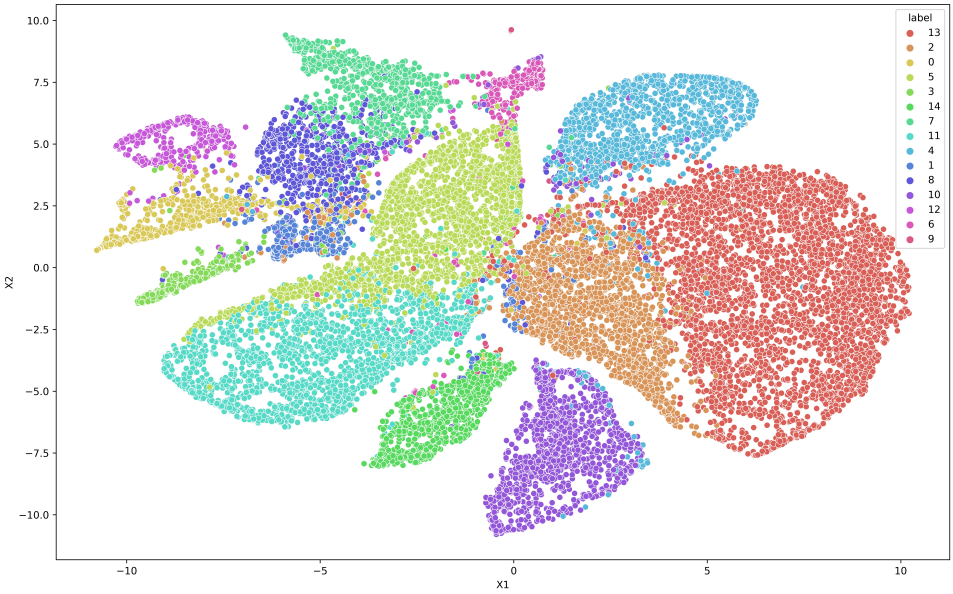

Figure 4.11: Dataset: Coauthor CS.

Chapter 5

Future Work and Conclusion

5.1 Conclusion

In this work, we proposed a novel spectral-based filter, Legendre-Filter, for a semi-supervised node classification task for graph-structured data. Furthermore, we proposed a novel algorithm that changes graph topology based on degree information and other heuristics to improve the overall accuracy of a given model. Our proposed Legendre Filter performs better than GCN, GAT, and Chebyshev Filter for the semi-supervised learning task on almost all the datasets. For full-supervised learning, our model performed better than all three baselines. Furthermore, on a few datasets, our model even performed better than deep models like JKNet, GCNII, etc.

5.2 Future Work

First, we briefly discuss the limitations of our approach. Being a shallow model, our model has a limited representational capacity. Being a spectral-based filter, it suffers from the same limitations as any other spectral-based filter.
One approach to improve the representational capacity of deep models like GCNII is to combine an attention-based approach like GAT to compute the attention matrix and combine it with the GCN block to generate output features.

Bibliography

[1] Bianchi, F.M., Grattarola, D., Livi, L., Alippi, C.: Graph neural networks with convolutional ARMA filters. CoRR abs/1901.01343 (2019), http://arxiv.org/abs/1901.01343

[2] Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 1725–1735. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/chen20v.html

[3] Chung, F.R.K.: Spectral Graph Theory. American Mathematical Society (1997)

[4] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. CoRR abs/1606.09375 (2016), http://arxiv.org/abs/1606.09375

[5] F. Scarselli, M. Gori, A.C.T.M.H., Monfardini, G.: The graph neural network model (2009)

[6] Fey, M., Lenssen, J.E., Weichert, F., Müller, H.: Splinecnn: Fast geometric deep learning with continuous b-spline kernels. CoRR abs/1711.08920 (2017), http://arxiv.org/abs/1711.08920

[7] Fu, X., Zhang, J., Meng, Z., King, I.: MAGNN: metapath aggregated graph neural network for heterogeneous graph embedding. CoRR abs/2002.01680 (2020), https://arxiv.org/abs/2002.01680

[8] Goodfellow, I.J., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA, USA (2016), http://www.deeplearningbook.org

[9] Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. CoRR abs/1706.02216 (2017), http://arxiv.org/abs/1706.02216

[10] Hammond, D.K., Vandergheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory (2009), https://arxiv.org/abs/0912.3848

[11] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)

[12] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)

[13] Isufi, E., Loukas, A., Simonetto, A., Leus, G.: Distributed time-varying graph filtering. CoRR abs/1602.04436 (2016), http://arxiv.org/abs/1602.04436

[14] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907 (2016), http://arxiv.org/abs/1609.02907

[15] Klicpera, J., Bojchevski, A., Günnemann, S.: Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. CoRR abs/1810.05997 (2018), http://arxiv.org/abs/1810.05997

[16] M. Gori, G.M., Scarselli, F.: A new model for learning in graph domains. in Proc. of IJCNN 2, 729–734 (2005)

[17] Monti, F., Bronstein, M.M., Bresson, X.: Geometric matrix completion with recurrent multi-graph neural networks. CoRR abs/1704.06803 (2017), http://arxiv.org/abs/1704.06803

[18] Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)

[19] Oono, K., Suzuki, T.: Optimization and generalization analysis of transduction through gradient boosting and application to multi-scale graph neural networks. CoRR abs/2006.08550 (2020), https://arxiv.org/abs/2006.08550

[20] Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (November 1999), http://ilpubs.stanford.edu:8090/422/, previous number = SIDL-WP-1999-0120

[21] Pei, H., Wei, B., Chang, K.C., Lei, Y., Yang, B.: Geom-gcn: Geometric graph convolutional networks. CoRR abs/2002.05287 (2020), https://arxiv.org/abs/2002.05287

[22] Pei, Y., Huang, T., van Ipenburg, W., Pechenizkiy, M.: Resgcn: Attention-based deep residual modeling for anomaly detection on attributed networks. CoRR abs/2009.14738 (2020), https://arxiv.org/abs/2009.14738

[23] Polynomials, L.: Properties of Legendre Polynomials

[24] Rong, Y., Huang, W., Xu, T., Huang, J.: The truly deep graph convolutional networks for node classification. CoRR abs/1907.10903 (2019), http://arxiv.org/abs/1907.10903

[25] Sandryhaila, A., Moura, J.M.F.: Discrete signal processing on graphs: Graph fourier transform. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 6167–6170 (2013)

[26] Shchur, O., Mumme, M., Bojchevski, A., Günnemann, S.: Pitfalls of graph neural network evaluation. CoRR abs/1811.05868 (2018), http://arxiv.org/abs/1811.05868

[27] Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks 8(3), 714–735 (1997)

[28] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2017), https://arxiv.org/abs/1710.10903

[29] Wikipedia contributors: Chebyshev polynomials — Wikipedia, the free encyclopedia (2022), https://en.wikipedia.org/w/index.php?title=Chebyshev_polynomials&oldid=1095221317, [Online; accessed 7-July-2022]

[30] Wikipedia contributors: Jacobi polynomials — Wikipedia, the free encyclopedia (2022), https://en.wikipedia.org/w/index.php?title=Jacobi_polynomials&oldid=1069717800, [Online; accessed 7-July-2022]

[31] Wikipedia contributors: Legendre polynomials — Wikipedia, the free encyclopedia (2022), https://en.wikipedia.org/w/index.php?title=Legendre_polynomials&oldid=1093551457, [Online; accessed 7-July-2022]

[32] Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network (2015), https://arxiv.org/abs/1505.00853

[33] Xu, K., L.C.T.Y.S.T.K.K., Jegelka: Representation learning on graphs with jumping knowledge networks. ICML (2018), https://arxiv.org/pdf/1806.03536

[34] Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. CoRR abs/1603.08861 (2016), http://arxiv.org/abs/1603.08861