SPECTRAL GRAPH CONVULUTIONS

Sai Ajay Modukuri
Department of Computer Science and Engineering
Pennsylvania State University
State College, PA 16803
svm6277@psu.edu

Zeba Karishma
Department of Computer Science and Engineering
Pennsylvania State University
State College, PA 16803
zbk5052@psu.edu

February 19, 2022

ABSTRACT

Convolution filters are ubiquitous in almost all computer vision models and are also widely applied in audio related deep learning problems. They are quite efficient at feature extractions while keeping the number of parameters low. Convolution filters in Convolutional Neural Networks exhibit shift-invariance, translation-invariance, parameter sharing, etc. This is possible because of the 2-D structure of the images or the 1D structure of audio in the euclidean domain. However, convolutions cannot be directly applied to 3D structures like social networks, graphs, etc. In this work, we are interested in the generalization of CNNs from low-dimensional regular grids to signals defined on more general high dimensional domains, which are represented by graphs. For this purpose, we explore convolutions on graph structured data in the spectral domain. We will start with simple approximations of convolutions in the spectral domain and explore efficient ways of applying convolutions on graphs in the spectral domain.

***K*eywords** Graph Laplacian · Graph Spectral Convolutions · Chebyshev Convolution

1 Introduction

Convolutional Neural Networks offer an efficient architecture that allows to capture local stationary property of the input data and form multi-scale hierarchical patterns in large scale and high dimensional datasets. It primarily consists of the following layers:

- Convolution layer (CONV) - uses filters that perform convolution operations as it is scanning the input with respect to its dimensions. Its hyperparameters include the filter size and stride. The resulting output is called a feature map or activation map.
- Pooling layer (POOL) - is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. More specifically, max or average pooling is performed, where the maximum value or the average value of the current view is taken, respectively.
- Fully Connected layer (FC) - usually found at the end of CNN architectures are used to optimize objectives.

The key take away from the CNN is the Convolutional layer. The Convolutional layer consists of the number of convolution filters that are applied hierarchically on the entire image to extract/aggregate various features from an image.

In this work, we will start with a gentle introduction of graphs, graph notations, graph laplacians, spectral decomposition of graph laplacians and convolutions on graphs in the spectral domain. This work provides the mathematical and computational foundations for an efficient generalization of CNNs to graphs using tools available in graph signal processing (GSP). However, generalizing CNNs to graphs is not straightforward as the convolution and pooling operators are only defined for regular grids. The major bottleneck of generalizing CNNs to graphs and primary area of focus of this work is the definition of localized graph filters, which are efficient to evaluate and learn.

2 Background

Graphs offer a natural framework to generalize the low-dimensional grid structure and, by extension, the notion of convolution. In this work, we will discuss constructions of deep neural networks on graphs other than regular grids. Although, there are two constructions in general, one based upon a hierarchical clustering of the domain (aka *Spatial Construction*), and another based on the spectrum of the graph Laplacian (aka *Spectral Construction*). We present a formulation of CNNs in the context of spectral graph theory (Spectral Construction), which allows us to study complex geometric structures encoded by Graphs. User data on social networks or text documents on word embeddings are examples of data lying on irregular or non-Euclidean domains that can be structured with graphs.

A graph is pair $G = (V, E)$, where $V = \{v_1, v_2, \dots v_n\}$ is set of vertices and $E \subseteq V \times V$, is the set of pairs of nodes connecting vertices (u, v) .

Adjacency matrix A of a graph G is is an $n \times n$ matrix such that $a_{ij} = 1$ if there is a edge between i, j, else $a_{ij} = 0$. Adjancecny matrix A is symmetric i.e, $A^T = A$ and eigen values of A are real and orthogonal.

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, u_j\} \in E \\ 0 & otherwise \end{cases}$$

Degree matrix D of a given matrix G is an $n \times n$ matrix such that $d_{ii} = degree(node_i)$, where $degree(node_i)$ is the number of incoming and outgoing edge of vertex i.

$$degree(node_i) = d(v) = |\{u \in V | (v, u) \in E \ (u, v) \in E\}|$$
$$D(G) = diag(d_1, d_2, \dots, d_n)$$
$$D_{ii} = \sum_j A_{ij}$$

A **signal** $x : \mathcal{V} \to \mathbb{R}$ defined on the nodes of the graph may be regarded as a vector $x \in \mathbb{R}^n$ where x_i is the value of x at the i^{ith} node.

2.1 Graph Laplacian

Laplacian matrix $\mathcal{L}$ of a graph G is defined as $L = D - A$, where D,A are adjacency and degree matrix of G. For the graph laplacian matrix $\mathcal{L}$, $l_{ij} = -1$ if there exists no edge between i, j, $l_{ij} = 0$ if edge exists between i, j and $l_{ii} = degree(node_i)$. The normalized graph laplacian $\mathcal{L}$ is defined as

$$\mathcal{L} = D^{-1/2} A D^{-1/2}$$

And the normalized symmetric graph Laplacian is defined as

$$\mathcal{L} = I - D^{-1/2} A D^{-1/2}$$
$$\mathcal{L} = \begin{cases} 1 & \text{if u=v and d(u)>0} \\ -\frac{1}{\sqrt{d(u)d(v)}} & \text{if u neighbhor of v} \\ 0 & \text{otherwise} \end{cases}$$

Graph laplacian can be extended to weighted graphs $G = (V, E, w)$, where W is the associated weight matrix. In the case of weighted graphs, the combinatorial Laplacian is defined as $L = D - W$ and normalized Laplacian is defined as $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$. In the rest of our work, we use weighted laplacians unless stated otherwise.

As one can observe, laplacian matrix is a real semi-positive symmetric matrix. It's eigenvalue decomposition (spectral decomposition) results in complete set of orthogonal vectors $\{u_l\}_{l=0}^{n-1}$, known as the **graph Fourier modes** and their associated ordered real nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$, identified as the frequencies of the graph.

The Laplacian is diagonalized by the Fourier basis $U = [u_0, ..., u_{n-1}] \in \mathbb{R}^{n \times n}$ such that $L = U \Lambda U$ where $\Lambda = \text{diag}([\lambda_0, \dots, \lambda_{n-1}]) \in \mathbb{R}^{n \times n}$. Graph laplacian provides provide intresting insights about the graph. If two vertices are connected by an edge with a large weight, the values of the eigenvector at those locations are likely to be similar.

The eigenvectors associated with larger eigenvalues oscillate more rapidly and are more likely to have dissimilar values on vertices connected by an edge with high weight. In addition, the eigen vectors of laplacian forms orthogonal basis, which is used to performs graph fourier transforms.Therefore, graph laplacian can be written as:

$$L = U \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix} U^{-1} = U \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix} U^T, \tag{1}$$

Where U^{-1} could be replaced by U^T because $UU^T = I_N$. Fourier transformation could be expressed as:

$$\hat{h}(L) = U \begin{bmatrix} \hat{h}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{h}(\lambda_{N-1}) \end{bmatrix} U^T \tag{2}$$

The graph Fourier transform of a signal $x \in \mathbb{R}^n$ is then defined as $\hat{x} = U^T x \in \mathbb{R}^n$, and its inverse as $x = U\hat{x}$. On euclidean spaces, this transform enables the formulation of fundamental operations such as filtering.

2.2 Harmonic Analysis on Weighted Graphs

The frequency and smoothness relative to W are interrelated through Laplacian $L = D - W$. For a function, $f \in L$, the quadratic form of Laplacian is given by

$$\langle f, Lf \rangle = \sum_{i \sim j} (f(i) - f(j))^2 \tag{3}$$

Here, i, j are the nodes of the graph. The quadratic form measures the smoothness of the function f. Now if we consider $f(.)$ as neural network like differentiable function as in [3] and X as a matrix of node feature vectors X_i, the above equation can be rewritten as *(Refer to Proof in Appendix A.1)*:

$$\mathcal{L} = \sum_{i,j} W_{ij} \left\| f(X_i) - f(X_j) \right\|^2 = f(X)^\top L f(X) \tag{4}$$

And the same defintion of smoothness has been used in [1] for x, an m-dimensional vector,to define a smoothness functional $\|\nabla x\|_W^2$ at a node i is

$$\|\nabla x\|_W^2 (i) = \sum_j W_{ij} \left(x(i) - x(j)\right)^2$$

$$\|\nabla x\|_W^2 = \sum_i \sum_j W_{ij} [x(i) - x(j)]^2 \tag{5}$$

using this definition, the smoothest vector is a constant,

$$u_0 = \text{argmin}_{x \in \mathbb{R}^m, ||x||=1} ||\nabla x||_W^2 = \frac{1}{\sqrt{m}} \mathbf{1}_m. \tag{6}$$

Here, u_0 is also the eigenvector of L corresponding to eigen value 0, and each successive eigen vector is given as:

$$u_i = \text{argmin}_{x \in \mathbb{R}^m, ||x||=1, x \perp \{u_0, \cdots, u_{i-1}\}} ||\nabla x||_W^2 \tag{7}$$

The eigenvalues λ_i allow the smoothness of a vector x to be read off from the coefficients of x in $[u_0 \dots u_{m-1}]$, equivalently as the Fourier coefficients of a signal defined in a grid. Thus, just in the case of the grid, where the eigenvectors of the Laplacian are the Fourier vectors, diagonal operators on the spectrum of the Laplacian modulate the smoothness of their operands

2.3 Convolutions

Convulutions are widely employed in Computer vision. They work well on 2D data such as images, videos, audio, etc. In other words, convulutions are well defined on 2D data in euclidean space. Convulution between two functions $f, g : \omega \to \mathbb{R}$, in euclidean is defined as:

$$(f \star g)(t) = \int_{-\infty}^{+\infty} f(s)g(t-s)ds, \tag{8}$$

In CNNs, the convolution operation is the sum of element-wise multiplication (dot product) of the selected area of the image/grid with a filter. However, we cannot perform convolutions directly on 3D data i.e., convolution operations are not defined on graphs in the euclidean domain. Performing convolutions on 3D data is defined by the convolution theorem as below.

Theorem 1 *Convolution in the spatial domain is equivalent to multiplication in the Fourier domain, i.e., if $x \in \mathcal{R}^n$, the Fourier transform on x is $F(x) = U^T X$, where U is the orthogonal basis obtained from decomposing graph Laplacian, i.e. $L = U\Lambda U^T$*

$$x \star g = F^{-1}(F(x) \odot F(g))) = U\left(U^T x \odot U^T g\right) \tag{9}$$

Where $*$ is the convolution operation in the euclidian domain, $\odot$ is the Hadamard product. Hadamard product is defined as an element wise multiplication on two matrices of same shape/size. For two matrices A, B of same shape $m \times n$, $(A \odot B)_{ij} = (a_{ij})(b_{ij})$.

The convolution theorem defines convolutions as linear operators that diagonalize in the Fourier basis (represented by the eigenvectors of the Laplacian operator).

When the convulution theorem is applied on graphs, the above equation (9), in case of graphs with node features X with $n \times d$ dimensions (here n is the number of nodes and d is the number of features) and H convulution filters, on the layer $l + 1$ we can rewrite it as:

$$X^{l+1} = U(U^T X^l \odot V^T H^l) \tag{10}$$

2.3.1 Challenges

In the above equation, the filter H acts on the entire graph. Therefore, the dimensionality of the trainable filters depends on the number of nodes N in the graph. H also depends on the graph structure encoded in eigenvectors U.

When applying convolutions on graphs using the above methods, we face computational overhead as convolutions are applied on the entire dataset. To overcome this, we can generalize the duality of the grid. On the Euclidian grid, the decay of a function in the spatial domain is translated into smoothness in the Fourier domain, and vice-versa. Therefore, in order to learn a layer in which features will be not only shared across locations but also well localized in the original domain, one can learn spectral multipliers that are smooth. This smoothness is established in the (5). A simple, naive approach is choosing fixed cubic spline kernels as below:

$$H^l \approx \sum_{k=1}^{K} \alpha_k f_k \tag{11}$$

The dimensionality of f_k depends on the number of nodes n. The functions f_k are fixed cubic splines. These functions are not learnable. These functions are particularly efficient if $K << N$. Thus this approach reduces the number of trainable parameters and also defines convolution filters whose size in independent of the number of nodes in the graph, thus allowing for convolutions, which can scale well with the size of the graphs.

While this solves the issue of (9), where using convolutional filters size need not have to be equal to N, it is still computationally heavy. This is due to the complexity of eigenvalue decomposition, which is $O(N^3)$. Moreover, multiplication with graph Fourier basis costs an additional $O(N^2)$ computations. The other problem with this approach is the trained model is closely related to the eigenvectors of the graph. This can be a problem if the train and test sets have different structures. Finally, since the filters are not completely trainable, the models trained using this approach will be quite limited in their modeling capacity.

Some of these problems can be overcome by using Chebyshev graph convolutions discussed in the next section.

3 FAST APPROXIMATE CONVOLUTIONS ON GRAPH

3.1 Spectral Graph Convolutions

Spectral convolutions on Graph is defined as the multiplication of a signal $x \in \mathbb{R}^n$ (a scalar for every node) with a filter $g_\theta = diag(\theta)$ parametrized by $\theta \in \mathbb{R}^n$ in the Fourier domain given by:

$$y = g_\theta(L)x = g_\theta\left(U\Lambda U^T\right)x = Ug_\theta(\Lambda)U^T x \tag{12}$$

A non-parametric filter, i.e. a filter whose parameters are all free, would be defined as

$$g_\theta(\Lambda) = diag(\theta), \tag{13}$$

where the parameter $\theta \in \mathbb{R}_n$ is a vector of Fourier coefficients. However, non-parametric filters are not localized in space and their learning complexity is in $\mathcal{O}(n)$, the dimensionality of the data.

Smoothness defined using cubic spline kernel in Eq. 11 can be rewritten using polynomial filter as follows:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \tag{14}$$

where the parameter $\theta \in \mathbb{R}_K$ is a vector of polynomial coefficients.

Using polynomial parametrization for localized filters as defined in equation 14, the value at vertex j of the filter g_θ centered at vertex i is given by

$$\left(g_\theta(L)\delta_i\right)_j = \left(g_\theta(L)\right)_{i,j} = \sum_k \theta_k \left(L^k\right)_{i,j} \tag{15}$$

where the kernel is localized via a convolution with a Kronecker delta function $\delta_i \in \mathbb{R}^n$. By Theorem 2, $d(i,j) > K$ implies $(L^K)_{i,j} = 0$, where d is the shortest path distance, i.e. the minimum number of edges connecting two vertices on the graph. Consequently, spectral filters represented by K^{th}- order polynomials of the Laplacian exactly have K parameters, also know as K-localized, and their learning complexity is $\mathcal{O}(K)$, the support size of the filter. This overcomes the heavy computation of $O(N)$ faced in learning non-parametric filters. Using this approach, we reduce the learning complexity to $O(K)$ (where K is the size of the filter). This is similar to learning complexity of filters in classic CNN on grids.

3.2 Chebyshev Graph Convolution

Although the above formulation reduces the number of learnable parameters for a filter to K, we still have to perform multiplication in Fourier basis U. To reduce the complexity of filter operation from $O(n^2)$ operations caused by multiplication with Fourier basis, let's formulate $g_\theta(L)$ as a polynomial function that can be computed recursively from L, as K multiplications by a sparse L that costs $O(K|\mathcal{E}|) \ll O\left(n^2\right)$.

For this purpose, Chebyshev polynomial $T_k(x)$ of order k is chosen, which can be computed by the stable recurrence relation.

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \text{ with } T_0 = 1 \text{ and } T_1 = x$$

These polynomials form an orthogonal basis for $L^2\left([-1,1], dy/\sqrt{1-y^2}\right)$ with respect to the measure $dy/\sqrt{1-y^2}$. A filter can thus be parametrized as the truncated expansion $g_\theta(\nabla)$

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \tag{16}$$

of order $K-1$, where the parameter $\theta \in \mathbb{R}_K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}_{nxn}$ is the Chebyshev polynomial of order k evaluated at $T_k(\tilde{\Lambda}) = 2\Lambda/\lambda_{max} - I_n$, a diagonal matrix of scaled eigenvalues that lie in $[-1,1]$.

Chebyshev polynomial $T_k(\tilde{\Lambda})$ is a matrix with dimensions same as $\tilde{\Lambda}$ (i.e. n×n, where n = number of nodes). Elements of matrix $T_k(\hat{\Lambda})$ are obtained by applying Chebyshev polynomial definition element wise.

$$T_0(\tilde{\Lambda}) = \begin{bmatrix} T_0\left(\frac{2}{\lambda_{\max}}(\lambda_1 - 1)\right) & & & \\ & T_0\left(\frac{2}{\lambda_{\max}}(\lambda_2 - 1)\right) & & \\ & & \ddots & \\ & & & T_0\left(\frac{2}{T_{\max}}(\lambda_N)\right) \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} = I_N$$

Therefore, $T_0(\tilde{\Lambda}) = I_N$ and $T_1(\tilde{\Lambda}) = \tilde{\Lambda}$

The filtering operation can then be written as

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})x, \tag{17}$$

where $T_k(\tilde{\Lambda}) \in \mathbb{R}^{nxn}$is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $T_k(\tilde{\Lambda}) = 2L/\lambda_{max} - I_n$.

Using this formulation, we can avoid the costly eigenvalue decomposition, which is a $O(N^3)$ operation. This brings down the entire filtering operation $y = g_\theta(L)x$ to $O(K|\mathcal{E}|)$, where $\mathcal{E}$ is the number of edges. The derivation of this convolution approximation using Chebyshev polynomial and graph laplacian has been shown in A.2

3.3 Localization

We know that if the kernel g is localized in the spectral domain, then the associated spectral graph will all be localized in frequency.

Theorem 2 *Let G be a weighted graph, with adjacency matrix A. Let B equal the adjacency matrix of the binarized graph, i.e. Bm,n = 0 if Am,n = 0, and $B_{m,n} = 1$ if $A_{m,n} > 0$. Let $\hat{B}$ be the adjacency matrix with unit loops added on every vertex, e.g.$\hat{B}_{m,n} = B_{m,n}$ for $m \neq n$ and $\hat{B}_{m,n} = 1$ for $m = n$.*

Then for each $s > 0$, $(\hat{B}^s)_{m,n}$ equals the number of paths of length s connecting m and n, and $(\hat{B}^s)_{m,n}$ equals the number of all paths of length $r \leq s$ connecting m and n.

4 Applications of Fast Graph Convolutions

In this section, we describe how fast convolutions achieved by first-order approximation of spectral graph convolutions can therefore be used to build a neural network model. We show two significant applications of the fast Chebyshev graph convolutions.

4.1 Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

[2] paper designs a localized convolutional filter using the spectral approach. It defines convolution filters using a localization operator on graphs via convolutions with a Kronecker delta implemented in the spectral domain.Where, the output feature map is calculated using Spectral Graph Convolution coupled with fast Chebyshev approximations as described in Section 3. Therefore, the j^{th} output feature map of the sample s is given by

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L)x_{s,i} \in \mathbb{R}^n \tag{18}$$

where the $x_{s,i}$ are the input feature maps and the $F_{in} \times F_{out}$ vectors of Chebyshev coefficients $\theta_{i,j} \in \mathbb{R}^K$ are the layer's trainable parameters. The following two gradients are calculated to train the CNN architecture

$$\frac{\partial E}{\partial \theta_{i,j}} = \sum_{s=1}^{S} [\bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1}]^T \frac{\partial E}{\partial y_{s,j}} \quad \text{and} \quad \frac{\partial E}{\partial x_{s,i}} = \sum_{j=1}^{F_{\text{out}}} g_{\theta_{i,j}}(L)\frac{\partial E}{\partial y_{s,j}} \tag{19}$$

where E is the loss energy over a mini-batch of S samples.

4.2 SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

[3] paper takes a graph-based semi-supervised learning approach to classify nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. The label information is smoothed over the graph by using a graph Laplacian regularization term in the loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \left\| f(X_i) - f(X_j) \right\|^2 = f(X)^\top L f(X) \tag{20}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(.)$ can be a neural network like differentiable function, λ is a weighing factor and X is a matrix of node feature vectors X_i. Again, L is the unnormalized graph laplacian of the undirected graph.

It considers a multi-layer Graph Convolution Network with the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right) \tag{21}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. $\sigma(.)$ denotes an activation function, such as the ReLU(·) = max(0, ·). $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$. The above propagation rule was motivated via a first-order approximation of localized spectral filters on graphs as described in Section 3.A neural network model based on graph convolutions is built by stacking multiple convolutional layers of the form of Eq. 17, each layer followed by a point-wise non-linearity. The K-localized convolution given by Chebyshev approximation is used to define a layer-wise linear GCN model by limiting the layer-wise convolution operation to K = 1 (see Eq. 17), i.e. a function that is linear w.r.t. L and therefore a linear function on the graph Laplacian spectrum.It provides us with convolutional filter functions by stacking multiple such layers, but we are not limited to the explicit parameterization given by, e.g., the Chebyshev polynomials.

Furthermore, approximating $\lambda_{\max} \approx 2$,Eq. 17 simplies to:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \tag{22}$$

with two free parameters θ'_0 and θ'_1.

The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the k^{th}-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model. In practice, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This leaves us with the following expression:

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x \tag{23}$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$. Note that $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range [0, 2]. Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, we introduce the following renormalization trick:

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow D^{-\frac{1}{2}} A \dot{D}^{-\frac{1}{2}}, \tag{24}$$

with $\tilde{A} = A + I_N$ and $D_{ii} = \sum_j A_{ij}$. We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \tag{25}$$

where $\theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity $O(|\varepsilon| FC)$, as $A\hat{X}$ can be efficiently implemented as a product of a sparse matrix with a dense matrix.

5 Discussion

In the context of computer vision and machine learning, the graph Laplacian defines how node features will be updated if we stack several graph neural layers in the form of formula

$$X^{(l+1)} = \mathcal{A} X^{(l)} W^{(l)} \tag{26}$$

So, given graph Laplacian L, node features $X \in \mathbb{R}^{N \times C}$ and spectral filters W , a spectral convolution on C dimensional graphs can be performed by repeating the convolution for each channel and then sum over C as in signal/image convolution. However, spectral convolution of signals on regular grids using the Fourier Transform creates the following problems:

1. the dimensionality of trainable weights (spectral filters) W depends on the number of nodes N in a graph;
2. W also depends on the graph structure encoded in eigenvectors V.

These issues prevent the model from scaling to datasets with large graphs of variable structure. To solve this issue, smooth filters as discussed in Section 2.3 in the spectral domain were proposed. While it solves the first issue, this smoothing method does not address the second issue.

Furthermore, spectral convolution calculates eigen decomposition which has huge complexity of $O(N^3)$. More specifically, keeping the graph Laplacian in a dense format in RAM is infeasible. Since the trained model is closely related to the eigenvectors V of the graph, it can be a difficulty if training and test graphs have very different structures. Moreover, if you use some smoothing of filters in the frequency domain like splines discussed above, then your filters become more localized, and the problem of adapting to new graphs seems to be even less noticeable. However, the models will still be quite limited.

Using Chebyshev polynomials for fast approximations solves these problems by avoiding the computation of costly eigen-decomposition and by detaching the filters from the eigenvectors. It utilizes a very useful parameter K, which determines the locality of filters. Informally: for $K = 1$, we feed just node features $X^{(l)}$ to our GNN; for K=2, we feed $X^{(l)}$ and $AX^{(l)}$; for K=3, we feed $X^{(l)}$, $AX^{(l)}$ and $A^2X^{(l)}$ and so forth.So basically, increasing K in the Chebyshev convolution increases the total number of trainable parameters. For example, for K=2, our weights $W^{(l)}$ will be 2C×F instead of just C×F. And adding more training parameters means the model is more difficult to train, and more data must be labeled for training.

Note that to satisfy the orthogonality of the Chebyshev basis, A assumes no loops in the graph, so that in each i-th row of matrix product $AX^{(l)}$ we will have features of the neighbors of node i, but not the features of node i itself. Features of node i will be fed separately as a matrix $X^{(l)}$.

If K equals the number of nodes N, the Chebyshev convolution closely approximates a spectral convolution so that the receptive field of filters will be the entire graph. But, as in the case of convolutional networks, we do not want our filters to be as big as the input images, so in practice, K takes reasonably small values.

So, GCN of [3] essentially "merged" matrices of node features$X^{(l)}$ and $AX^{(l)}$into a single N×C matrix. As a result, the model has two times fewer parameters to train compared to Chebyshev convolution with K=2, yet has the same receptive field of 1 hop. The main trick involves adding "self-loops" to the graph by adding an identity matrix I to A before computing the Laplacian and normalizing it, so now in each i-th row of matrix product $AX^{(l)}$ we will have features of the neighbors of node i, as well as features of node i. The main difference between the two methods is that in the Chebyshev convolution, we recursively loop over K to capture features in the K-hop neighborhood. We can stack such GCN or Chebyshev layers interleaved with nonlinearities to build a Graph Neural Network.

However, some of the limitations faced by GCN [3] are as follows: (1) With full-batch gradient descent, memory requirement grows linearly in the size of the dataset. Therefore, large graphs will not fit into GPU memory and will require to be trained on CPU. (2) The current model needs to be updated to handle edge features and directed graphs as the original directed graph can be represented as an undirected bipartite graph with additional nodes that represent edges in the original graph.

6 Conclusion

In this work, we explore spectral graph convolutions. We leverage convolution theorem in the spectral domain and build upon this theorem to introduce simple non parametric filters. We further this idea to define more efficient approximations for filters without the need for eigen value decomposition, thereby considerably reducing the time complexity involved in performing convolutions in the spectral domain. Using these methods, we discuss applications such as classification in semi-supervised setting.

References

[1] Joan Bruna and Wojciech Zaremba and Arthur Szlam and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv preprint arXiv:1312.6203*, 2014.

[2] Michaël Defferrard and Xavier Bresson and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering *arXiv preprint arXiv:1606.09375*, 2017.

[3] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, 2017.

[4] *https://personal.utdallas.edu/ hkokel/articles/GraphConvolutionalNetwork.html*.

A APPENDIX

A.1 Laplacian in quadratic form

For any $f \in L$, the quadratic form of weighted Laplacian is

$$\mathcal{L} = \sum_{i,j} W_{ij} \left\| f\left(X_i\right) - f\left(X_j\right) \right\|^2 = f(X)^\top L f(X)$$

$Proof$:

$$\begin{aligned}
f(X)^T \Delta f(X) &= f(X)^T (D - W) f(X) \\
&= f\left(X\right)^\top D f\left(X\right) - f\left(X\right)^\top W f\left(X\right) \\
&= \sum_{i=1}^{n} D_{ii} f\left(X_i\right)^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} f\left(X_i\right) f\left(X_j\right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} f\left(X_i\right)^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} f\left(X_i\right) f\left(X_j\right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left(f\left(X_i\right)^2 - f\left(X_i\right) f\left(X_j\right) \right) \\
&= \sum_{\{i,j\} \in E} \left(f\left(X_i\right) - f\left(X_j\right) \right)^2 \\
&= \sum_{i \leq j} W_{ij} \left\| f\left(X_i\right) - f\left(X_j\right) \right\|^2
\end{aligned}$$

A.2 Using Graph Laplacian to Approximate Convolution

The convolution theorem in graphs represents following equation: $L = U \Lambda U^\top$, where $U \Lambda U^\top$ is the eigen decomposition of the graph laplacian matrix. The complexity of finding eigen-decomposition is $O(n^3)$.

Lemma 1: $\left(U \Lambda U^\top\right)^k = U \Lambda^k U^\top$

$Proof$:

$$\begin{aligned}
\left(U \Lambda U^\top\right)^2 &= \left(U \Lambda U^\top\right) \left(U \Lambda U^\top\right) \\
&= U \Lambda \left(U^\top U\right) \Lambda U^\top \\
&= U \Lambda I \Lambda U^\top \\
&= U \Lambda^2 U^\top \\
\text{Similarly, } \left(U \Lambda U^\top\right)^k &= U \Lambda^k U^\top
\end{aligned}$$

Lemma 2: $U T_k(\tilde{\Lambda}) U^\top = T_k(\tilde{L})$

$Proof$:

$$\begin{aligned}
U T_0(\tilde{\Lambda}) U^\top = U I_N U^\top = I_N = T_0(\tilde{L}) \\
U T_1(\tilde{\Lambda}) U^\top = U \tilde{\Lambda} U = \tilde{L} = T_1(\tilde{L})
\end{aligned}$$

$$\begin{aligned}
J_2(\tilde{\Lambda}) U^\top &= U \left(2 \tilde{\Lambda} T_1(\tilde{\Lambda}) - T_0(\tilde{\Lambda}) \right) U^\top \\
&= 2 U \tilde{\Lambda} T_1(\tilde{\Lambda}) U^\top - U T_0(\tilde{\Lambda}) U^\top \\
&= 2 U \tilde{\Lambda} U^\top U T_1(\tilde{\Lambda}) U^\top - U T_0(\tilde{\Lambda}) U^\top \\
&= 2 \tilde{L} T_1(\tilde{L}) - T_0(\tilde{L}) \\
&= T_2(\tilde{L}) \\
\text{Similarly, } U T_k(\tilde{\Lambda}) U^\top &= T_k(\tilde{L})
\end{aligned}$$

Using the above two results, we can approximate the convolution without needing eigen-decomposition, directly using graph laplacian,

$$\begin{aligned} g_{\theta'}(\Lambda) \star x &\approx U \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) U^\top \mathrm{x} \\ &= \sum_{k=0}^{K} \theta'_k U T_k(\tilde{\Lambda}) U^\top \mathrm{x} \\ &= \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) \mathrm{x} \end{aligned}$$

Using Chebyshev polynomial, the complexity of the graph convolution has successfully reduced from $O(n^3)$ to $O(|\varepsilon|)$, i.e. from cubic in number of nodes (n) to linear in number of edges ($|\varepsilon|$).Since $T_k(\tilde{L})$ has $2 * \varepsilon$ non-zero elements, multiplication $\theta'_k T_k(\tilde{L})\mathrm{x}$ is a sparse matrix multiplication which can be done in $O(|\varepsilon|)$.

A.3 GCN and spectral convolution

GCN can be seen as approximation of spectral convolution with $K = 1$, $\lambda_{max} = 2$ and $\theta = \theta'_0 = \theta'_1$

$$\begin{aligned} g_{\theta'}(\Lambda) \star x &\approx \sum_{k=0}^{1} \theta'_k T_k(\tilde{L}) x \\ &= \theta'_0 T_0(\tilde{L}) x + \theta'_1 T_1(\tilde{L}) x \\ &= \theta'_0 I_N x + \theta'_1 \left(\frac{2}{\lambda_{\max}} L - I_N \right) \\ &= \theta'_0 x + \theta'_1 (L - I_N) \quad \boxed{\lambda_{max} = 2} \\ &= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad \boxed{L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}} \\ &= \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \quad \boxed{\theta = \theta'_0 = -\theta'_1} \\ &= \theta \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) x \quad \boxed{\underbrace{I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}}_{\text{renormalization trick}}, \text{ with } \tilde{A} = A + I_N} \\ &= \theta \hat{A} x \end{aligned}$$

Finally,

$$\begin{aligned} g_{\theta'}(\Lambda) \star x &= diag(\theta)\, T_k(\tilde{L})\, X \\ &= T_k(\tilde{L})\, X\, diag(\theta) \end{aligned}$$

With $X \in \mathbb{R}^{n \times f_i}$, we have $Z = \hat{A} X W$ which is graph convolution. Complexity of matrix multiplication is $O(n f_i f_o)$. Since $\hat{A}$ is a sparse matrix with ε non-zero elements, the complexity is $O(|\varepsilon| f_i f_o)$