Augmenting Graph Convolutional Neural Networks with Highpass Filters

Fatemeh Ansarizadeh(✉), David B. Tay, Dhananjay Thiruvady, and Antonio Robles-Kelly

School of IT, Deakin University, Waurn Ponds, VIC 3216, Australia
{f.ansarizadeh,david.tay,dhananjay.thiruvady, antonio.robles-kelly}@deakin.edu.au

Abstract. In this paper, we propose a graph neural network that employs high-pass filters in the convolutional layers. To do this, we depart from a linear model for the convolutional layer and consider the case of directed graphs. This allows for graph spectral theory and the connections between eigenfunctions over the graph and Fourier analysis to employ graph signal processing to obtain an architecture that "concatenates" low and high-pass filters to process data on a connected graph. This yields a method that is quite general in nature applicable to directed and undirected graphs and with clear links to graph spectral methods, Fourier analysis and graph signal processing. Here, we illustrate the utility of our graph convolutional approach to the classification using citation datasets and knowledge graphs. The results show that our method provides a margin of improvement over the alternative.

Keywords: Citation graph · Graph convolutional neural networks · Knowledge graph

1 Introduction

Recent breakthroughs in machine learning techniques have resulted in substantial progress in a wide range of areas, from image classification to natural language understanding. Predominately, for convolutional neural networks (CNNs), the input data, *i.e. images*, can be viewed as structured in an Euclidean space and hence abstracted onto a planar graph on a lattice. However, in many fields of research, including social networks and brain connectomes, input instances can have non-planar structure and hence, be defined over non-Euclidean spaces. Indeed, developing efficient implementations of CNNs for high-dimensional, non-Euclidean domains such as non-planar graphs, polygonal meshes or manifolds [4] its not a straightforward task [5].

The recent advent of graph neural networks provides the ideal basis for applying machine learning algorithms to datasets whose instances are not structured as a lattice and that require the capacity to process more general graphs. In particular, applying CNNs on data architectures in non-Euclidean domains was initially conceived in 1998 [6] and further developed by Scarselli *et al.* [7]. The resulting approach – graph neural networks (GNNs) – provide an efficient way to model problems as a graph, which can be easily integrated with neural networks. GNNs generally pose the problem of learning at each vertex of a graph using its neighborhood information. Early approaches modelled this local support using an information-theoretic standpoint whereby a signal on a graph with N vertices can be viewed as the counterpart of a discrete-time signal with N samples in classical signal processing. This has given rise to graph signal processing, where the most serious hurdle on the application of classical signal processing methods on graphs concerns discrete-time signals whereby dependencies arising from the irregular data domain are difficult to detect [8].

Neural networks on graphs have been proposed elsewhere [2,5,7,12,22]. In this paper, we propose a layer-wise GCN which employs a localized first-order approximation of spectral graph convolution to process data on a connected graph. This sort of architectures were presented in [22] and later advanced by [12]. The main difference between these and that presented in this paper stems from the fact that our architecture not only contains the equivalent of low-pass filters, but augments these with high-pass ones. To combine low and high-pass filters, we depart from graph-spectral theory. This yields a method to "concatenate" the low and high pass filter responses on the graph.

2 Background

In this paper, the focus is on the connections between graph convolutional networks (GCNs) and spectral graph theory. Recall that graph-spectral methods [13] allow for a direct link to be made with the Fourier transform through Chebychev polynomials. Thus, spectral methods carry a natural notion of frequency conveyed through the natural relationship between eigenfuctions over the graph, their corresponding eigenvalues, and the natural frequencies or "modes" of connected graphs computed making use of the graph Laplacian. For example, eigenvalues that are small in value correspond to eigenfunctions on the graph that vary smoothly and change gradually and, hence, can be viewed as representing low frequencies. In contrast, large eigenvalues correspond to rapidly changing functions, *i.e.* high frequencies. This is analogous to the treatment of eigenvalues and eigenvectors in classical Fourier analysis.

Throughout the paper we will extensively use graphs. Formally, in a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, $\mathcal{V}$ and $\mathcal{E}$ are the set of vertices and edges, respectively. For the purposes of inference and learning, instances are abstracted as vertices and the relationship between them correspond to edges in the graph. For instance, when applied to our sample citation datasets, the vertices $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$ represent documents under consideration and the edges $\mathcal{E} = \{e_1, e_2, \cdots, e_m\}$ stand for whether a document cites another or not, *i.e.* an edge $e_k = (v_i, v_j)$ implies that document v_j cites v_i.

As mentioned above, graph-spectral methods are based upon the Laplacian or adjacency matrix [15]. Recall that the graph connectivity can be described by the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, where its element $a_{i,j}$ is the weight of the edge connecting nodes i and j, and a zero weight indicates no connection. A graph can have directed or undirected edges, whereby, for the latter case $\boldsymbol{A}$ is symmetric. The normalized graph Laplacian matrix defined as $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ [9], where $\boldsymbol{D}$ is a diagonal matrix called the degree matrix (the i^{th} diagonal element correspond to the degree of the node indexed i) and $\boldsymbol{I}$ is the $N \times N$ identity matrix. Thus, $[\boldsymbol{D}]_{i,i} \equiv \sum_j a_{i,j}$ and $[\boldsymbol{D}]_{i,i}$ represents the sum of the weights of all the edges incident to vertex i. An important property of the normalized Laplacian matrix (for undirected graphs) is that it is a real, symmetric, positive semidefinite matrix. As a result, the normalized Laplacian matrix can be factorized as $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where $\boldsymbol{U} = [\boldsymbol{u}_0|\boldsymbol{u}_1|\cdots|\boldsymbol{u}_{N-1}]$ is the orthogonal matrix of eigenvectors. These eigenvectors correspond to the eigenvalues given by the diagonal matrix $\boldsymbol{\Lambda} \equiv \text{diag}(\lambda_0, \lambda_1, \cdots, \lambda_{N-1})$, *i.e.* $\boldsymbol{\Lambda}_{ii} = \lambda_i$. These eigenvalues define the spectrum of the graph and provide a frequency interpretation of the "modes" for the graph. Small eigenvalues correspond to low frequencies and vice-versa.

Many graph signal processing methods are spectrally-based [8] as it allows for the analysis of the signals in terms of their frequency content. In a connected graph, the eigenvalues of the normalized graph Laplacian, $\boldsymbol{\Lambda} = [\lambda_0, \lambda_1, \cdots, \lambda_{N-1}]$, satisfy the inequality, $0 = \lambda_0 < \lambda_1 \leqslant \cdots \leqslant \lambda_{Max} \leqslant 2$. Moreover, the eigenvectors of the normalized Laplacian matrix form an orthonormal space, *i.e.* $\boldsymbol{u}_i^T\boldsymbol{u}_j = \delta(i-j)$. Let a graph signal $x \in \boldsymbol{R}^N$ represent a feature vector for all nodes of a graph, where x_i is the feature value for the i^{th} node. The graph Fourier transform of the signal x is defined as $\hat{x} \equiv \boldsymbol{\mathcal{F}}(x) \equiv \boldsymbol{U}^T x$, where $\hat{x}$ is the transformed signal. The inverse graph Fourier transform is defined as $x = \boldsymbol{\mathcal{F}}^{-1}(\hat{x}) \equiv \boldsymbol{U}\hat{x}$.

The transformed signal $\hat{x}$ possesses the coordinates of the graph signal in the new orthogonal space $\boldsymbol{U}$. Therefore, the input signal x can be expressed as $x = \sum_i \hat{x}_i\boldsymbol{u}_i$, which is the inverse graph Fourier transform expression. Considering this, the spectral convolution in the Fourier domain between a signal and a filter $g_\theta \equiv \text{diag}(\theta)$, where $\theta \in \boldsymbol{R}^N$, is given by $g_\theta\hat{x} = g_\theta(\boldsymbol{U}^T x)$. This is equivalent to the multiplication of each diagonal element of g_θ with each element of $\hat{x}$. The filter output signal $z \in \boldsymbol{R}^N$ in the vertex domain is then obtained by taking the inverse Fourier transform of $g_\theta\hat{x} = g_\theta(\boldsymbol{U}^T x)$ as $z \equiv g_\theta * x = \boldsymbol{U}g_\theta\boldsymbol{U}^T x$.

The above definition of convolution is the basis of all spectral convolutions on graph neural networks. Note that the only distinction among these spectral convolutions relies on the properties of the filter g_θ. In a machine learning setting, θ is the vector of learnable parameters. Furthermore, this spans a matrix Θ where each column corresponds to a vector θ. This matrix can be viewed as the span of "channels", where each of these corresponds to a different filter g_θ. In practice, the computational complexity of the above graph convolution is of order $\mathcal{O}(N^3)$, which makes it impractical for large graphs. To circumvent this issue, Chebyshev polynomials are used to approximate the filter g_θ and thereby reducing the complexity to $\mathcal{O}(N)$ [5]. Thus, the filter g_θ is approximated by Chebyshev polynomials of the diagonal matrix of eigenvalues as $g_\theta = \sum_{i=0}^{K} \theta_i T_i(\hat{\boldsymbol{\Lambda}})$, where $\hat{\boldsymbol{\Lambda}} = 2\boldsymbol{\Lambda}/\lambda_{max} - \boldsymbol{I}$ contain the normalized eigenvalues in the interval $[-1, 1]$, and $T_i(x)$ are Chebyshev polynomials of the first kind.

By defining $\tilde{\boldsymbol{L}} \equiv 2\boldsymbol{L}/\lambda_{Max} - \boldsymbol{I}$ and using the property $T_i(\tilde{\boldsymbol{L}}) = \boldsymbol{U}\boldsymbol{T}_i(\tilde{\boldsymbol{\Lambda}})\boldsymbol{U}^T$, the filter output $g_\theta \hat{x} = g_\theta(\boldsymbol{U}^T x)$ can be written as $z = g_\theta * x = \sum_{i=0}^{K} \theta_i \boldsymbol{T}_i(\tilde{\boldsymbol{L}})x$. Therefore, we can infer that approximating the filter by Chebyshev polynomials makes GCNs to be locally supported in the vertex space, *i.e.* only signal values in a neighbourhood of a given vertex are needed to compute the output value of the vertex [8]. Using a first-order approximation of $g_\theta * x = \sum_{i=0}^{K} \theta_i \boldsymbol{T}_i(\tilde{\boldsymbol{L}})x$, with $K = 1$ and $\lambda_{Max} = 2$, the simplified version of graph convolution becomes $g_\theta * x = \theta_0 \boldsymbol{x} - \theta_1 \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{x}$ [12]. To preclude over-fitting by limiting the number of parameters, we assume $\theta = \theta_0 = -\theta_1$. After substituting this equality in previous equation we get $g_\theta * x = \theta\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{x}$.

We now consider multiple channels of input/output and introduce non-linearities. This yields

$$\boldsymbol{H} = \boldsymbol{g_\theta} * \boldsymbol{X} = f\left(\boldsymbol{\Theta}\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{X}\right), \tag{1}$$

where $\boldsymbol{X}$ is a matrix of node feature vectors X_i, and $f(\cdot)$ is called an activation function.

Since the numerical experiments show instability in the above GCN [9], we opt to solve the problem via re-normalization. To this end, $\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is replaced with $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\tilde{\boldsymbol{A}} \equiv \boldsymbol{A}+\boldsymbol{I}$ and $\tilde{\boldsymbol{D}}$ is the diagonal degree matrix $\tilde{\boldsymbol{D}}_{\boldsymbol{i},\boldsymbol{i}} = \sum_j \tilde{\boldsymbol{A}}_{(\boldsymbol{i},\boldsymbol{j})}$. As a result, the output of a single layer is then calculated using this equation

$$\boldsymbol{H} = f\left(\boldsymbol{\Theta}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}\right), \tag{2}$$

Thus, the forward propagation rule applied in a multi-layer graph convolutional networks (GCNs) follows an iterative scheme given by:

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right). \tag{3}$$

In Eq. (3), the filter $\boldsymbol{\theta}$ and activation function $f(\cdot)$ are replaced by $\boldsymbol{W}$ and $\boldsymbol{\sigma}$, respectively. Additionally, the state of the l^{th} layer is represented by $\boldsymbol{H}^{(l)}$, where the initial value of the state is $\boldsymbol{H}^{(0)} = \boldsymbol{X}$.

3 Convolutional Filters for GCNs

The filter in equation $g_\theta * x = \theta\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{x}$ is essentially a low-pass one, which only captures low frequency components of a signal/feature[8]. This low-pass filter essentially computes a weighted average of the signal values in a localized neighbourhood about a given vertex. Here, by introducing high-pass filtering, we aim to capture those high frequency components, which represent variation of the signal values about the localized neighbourhood.

3.1 Layer-Wise Linear Model

To commence, we depart from a localized first-order approximation of spectral graph convolution, which comprises an input layer, two convolutional layers and one output layer. In previous section, the symmetric normalization in Eq. (3) is only valid for undirected graphs. For the more general case of directed graphs, left-normalization (also called row-normalization) is used [10]. Therefore, symmetric normalization in Eq. (3), $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$, is replaced by $\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}$. This yields the forward propagation rule on directed graphs as

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right), \tag{4}$$

where l represents the current layer index and $l+1$ corresponds to that for the layer immediately after. We further simplify this equation by replacing $\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}$ by $\widehat{\boldsymbol{A}}$, which is defined below. This gives the final propagation rule as

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right). \tag{5}$$

3.2 Low and High-Pass Filters

We now turn our attention to the modification of $\tilde{\boldsymbol{A}}$, so as to incorporate both low and high-pass filters, in each convolutional layer. Consider N instances in each dataset and let $\boldsymbol{A}$ to be a symmetric adjacency matrix. We now define the following matrices:

$$\tilde{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A} \qquad \text{and} \qquad \tilde{\boldsymbol{B}} = \boldsymbol{I} - \boldsymbol{A} \tag{6}$$

where $\boldsymbol{I}$ is the identity matrix. To understand these equations in terms of its spectral filtering characteristics, consider the normalized Laplacian matrix which can be written as $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{A}$. The relationship between each eigenvalue of the Laplacian and the adjacency matrices is given by $\lambda = 1 - \mu$, [16], where λ and μ are the eigenvalues of Laplacian and adjacency matrices, respectively. The (diagonal) matrix form of $\lambda = 1 - \mu$ is $\boldsymbol{\Lambda} = \boldsymbol{I} - \boldsymbol{M}$. Using the definition of eigenvalues and eigenvectors, we have $\boldsymbol{L}\boldsymbol{U} = \boldsymbol{\lambda}\boldsymbol{U}$, and furthermore, using $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{A}$, we get:

$$(\boldsymbol{I} - \boldsymbol{A})\,\boldsymbol{U} = \boldsymbol{\Lambda}\boldsymbol{U} \Longrightarrow \boldsymbol{A}\boldsymbol{U} = (\boldsymbol{I} - \boldsymbol{\Lambda})\,\boldsymbol{U}. \tag{7}$$

Recall that the spectrum of normalized Laplacian matrix lies in range $0 \leqslant \lambda \leqslant 2$. Therefore, since $\lambda = 1 - \mu$, the spectrum of the corresponding adjacency matrix is $-1 \leqslant \mu \leqslant 1$. From a graph signal processing viewpoint, the frequency response function $1+\mu$ represents a low-pass filter corresponding to the matrix $\tilde{\boldsymbol{A}}$. When $\lambda = 0$, corresponding to the lowest frequency, $\mu = 1$ and the filter function value is $1 + \mu = 2$, corresponding to a large gain. When $\lambda = 2$, corresponding to the highest frequency, $\mu = -1$ and the filter function value is $1 + \mu = 0$, corresponding to a zero gain. The opposite is true for the high-pass filter, with frequency response function $1 - \mu$. This function takes the lowest and highest values at the smallest $\lambda = 0$ and largest $\lambda = 2$, respectively, and corresponds to the matrix $\tilde{\boldsymbol{B}}$.

3.3 Incorporating Filters into the Convolutional Layers

Here, we proceed to apply row normalization to $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$ as expressed in Eqs. (6). To compute $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$:

$$\widehat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}} \qquad \text{and} \qquad \widehat{\boldsymbol{B}} = \tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{B}}, \tag{8}$$

where $\tilde{\boldsymbol{D}}$ is the degree matrix and $\tilde{\boldsymbol{D}}_{i,i} = \sum_j \tilde{\boldsymbol{A}}$. These two matrices, $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$, are sparse since $\tilde{\boldsymbol{D}}$ is a diagonal matrix. This sparsity is crucial since real-world datasets are often sparse and hence this approach prevents over-fitting [17]. Here, $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ represent the low-pass and high-pass filters, respectively. To combine the effect of both filters, we concatenate $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ horizontally. Hence, in $\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right)$, $\widehat{\boldsymbol{A}}$ is replaced by $\left[\widehat{\boldsymbol{A}} \,\vdots\, \widehat{\boldsymbol{B}}\right]$, which results in the rule $\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\left[\widehat{\boldsymbol{A}} \,\vdots\, \widehat{\boldsymbol{B}}\right]\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right)$. To weight each of the filter matrices separately, the state or feature matrix, $\boldsymbol{H}$, is multiplied by $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ individually. This gives $\left[\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)} \,\vdots\, \widehat{\boldsymbol{B}}\boldsymbol{H}^{(l)}\right]$. The purpose here is to be able to modify the classic forward propagation rule so as to incorporate the low-pass and high-pass filters. Note that, up to this step, the propagation rule is as follows:

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\left[\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)} \,\vdots\, \widehat{\boldsymbol{B}}\boldsymbol{H}^{(l)}\right]\boldsymbol{W}^{(l)}\right). \tag{9}$$

Here, we initialize the weight matrix making use of the Glorot or Xavier method expressed in [18]. Since we are interested in analysing the influence of low-pass and high-pass filters, the weight matrices for each of them is initialized separately. Hence, two different weight matrices are allocated to the low-pass and high-pass filters. The weight matrix, $\boldsymbol{W}$, is a concatenated matrix formed by two other weight matrices as $\boldsymbol{\Delta}$ and $\boldsymbol{\Gamma}$. To be mathematically compatible, these two sub-matrices must be concatenated vertically $\boldsymbol{W}^{(l)} = \begin{bmatrix}\boldsymbol{\Delta}^{(l)} \\ \hdashline \boldsymbol{\Gamma}^{(l)}\end{bmatrix}$. The final forward propagation rule can then be written as:

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{\sigma}\left(\left[\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)} \,\vdots\, \widehat{\boldsymbol{B}}\boldsymbol{H}^{(l)}\right]\begin{bmatrix}\boldsymbol{\Delta}^{(l)} \\ \hdashline \boldsymbol{\Gamma}^{(l)}\end{bmatrix}\right). \tag{10}$$

Now $\boldsymbol{\Delta}^{(l)}$ is multiplied by $\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)}$ and $\widehat{\boldsymbol{B}}\boldsymbol{H}^{(l)}$ is multiplied by $\boldsymbol{\Gamma}^{(l)}$. By concatenating $\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)}$ and $\widehat{\boldsymbol{B}}\boldsymbol{H}^{(l)}$ horizontally, both the low and high-pass filters can simultaneously influence the output at each node. Note that the elements of the resulting matrix, inside the activation function, consists of terms arising from both the low and the high-pass filters. Furthermore, each layer's output forms the input of the next layer after passing through a nonlinear activation function, $\boldsymbol{\sigma}$.

4 Implementation Issues

Figure 1 shows the schematic of our GCNN implementation. In our implementation, the feature matrix $\boldsymbol{X}$, is provided at input. According to equation $g_\theta * x = \theta\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{x}$, in the first layer, $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, we have:

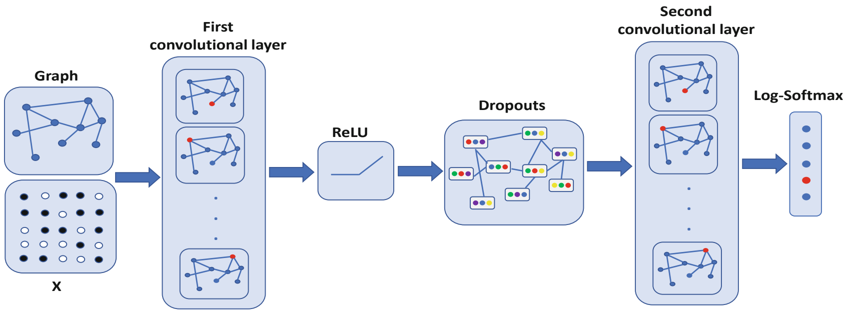

Fig. 1. A schematic representing our two-layer convolutional graph neural network. Each convolutional layer learns a hidden representation by aggregating feature information using local support. After the first layer, a ReLU function and dropout with probability of 0.5 is applied. In the output layer, a log-softmax is used for classification.

$$\boldsymbol{H}^{(1)} = ReLU\left(\left[\widehat{\boldsymbol{A}}\boldsymbol{X} \,\vdots\, \widehat{\boldsymbol{B}}\boldsymbol{X}\right]\left[\frac{\boldsymbol{\Delta}^{(0)}}{\boldsymbol{\Gamma}^{(0)}}\right]\right). \tag{11}$$

The output of Eq. (11) is the input to the next layer, which can be expressed as:

$$\boldsymbol{H}^{(2)} = \left[\widehat{\boldsymbol{A}}\boldsymbol{H}^{(1)} \,\vdots\, \widehat{\boldsymbol{B}}\boldsymbol{H}^{(1)}\right]\left[\frac{\boldsymbol{\Delta}^{(1)}}{\boldsymbol{\Gamma}^{(1)}}\right]. \tag{12}$$

In our network, multiplication of the combined low and high-pass filtering, expressed in $\left[\widehat{\boldsymbol{A}} \,\vdots\, \widehat{\boldsymbol{B}}\right]$, with the features matrix, is followed by the first convolutional layer. The process of convolution entails multiplication by the weight matrix. After convolution in the first layer, a *ReLU* activation function is applied. To prevent over-fitting in the model, after compiling the first layer, a dropout is introduced to the architecture. After the second convolutional layer, a log-Softmax loss is used to perform classification for mutually exclusive classes in the output layer.

5 Experiments

We now illustrate the utility of our GCNN for purposes of classification. To this end, we show results and provide comparison with the method in [12]. Our choice of alternative is based upon the notion that, as mentioned earlier, our

Table 1. Statistics of the citation network datasets used and results yielded by our method and the alternative in [12]

Dataset	Nodes	Edges	Classes	Features	Method in [12]	Our method
Cora	2078	5429	7	1433	80.1%	82.5%
Citeseer	3327	4732	6	3703	67.9%	69%

Table 2. Statistics of the knowledge graph dataset WebKB used in our experiments and results yielded by our method and the alternative in [12]

Dataset	Nodes	Edges	Classes	Features	Method in [12]	Our method
Cornell	195	304	5	1703	90%	90%
Texas	187	328	5	1703	42%	47.4%
Washington	230	446	5	1703	55.6%	77.8%
Wisconsin	265	530	5	1703	56%	60%

architecture is based on first order graph convolution approximation first presented in [22] and later advanced by [12].

All the datasets used to investigate the performance of our proposed approach were trained for 200 epochs using the Adam optimizer, which involves a first-order gradient-based optimization. Also, for purpose of providing a fair comparison, we have set all hyper-parameters of our approach to the same values as those reported in [12]. We implemented our network in Python with all layers defined according to Fig. 1 with a learning rate of 0.01 and a dropout rate of 0.5. In order to avoid over-training the model, a L-2 regularisation method with the term coefficient 5×10^{-4} is used in all our experiments.

5.1 Datasets

For our experiments, we have used three widely available datasets. These are the Cora [19][1], Citeseer [20][2] and WebKB[3] datasets. Here, and for the sake of consistency in our comparison with the alternative in [12], both the Citeseer and WebKB datasets were pre-processed before feeding them to the GCNs. A data cleaning was applied to the CiteSeer dataset as described in [21]. In case of the Cora dataset, we have used the same dataset splits as those employed in [12]. Regarding the other two datasets, we have used random 60%-20%-20-% dataset splits for training, validation and testing. All necessary matrices are constructed according to the metadata files provided with each dataset, which contain the information on nodes and edges necessary for constructing the adjacency matrices for the graph neural network. The number of nodes, features, classes and other information for the datasets have been summarised in Tables 1 and 2.

5.2 Classification Results

The results of our proposed approach are summarized and compared with the method in [12] in Table 1 and Table 2, which show the average accuracy and improvement over 10 runs, where each run has a unique random initialization. From these tables, we can appreciate improvements across datasets as compared

[1] The dataset can be accessed at https://relational.fit.cvut.cz/dataset/CORA.
[2] Widely available at http://networkrepository.com/citeseer.php.
[3] For more information on WebKB, go to http://www.cs.cmu.edu/~webkb/.

to the approach in the alternative, which only employs low-pass filters. The WebKB-Cornell dataset is the only case where our approach shows the same level of accuracy as the alternative. Breaking down the results, for the Cora and Citeseer datasets, we find slightly improved accuracy whereas some of the differences in the WebKB dataset are substantial. The reasons for more improvement in accuracy by adding highpass filters in these datasets can be attributed to the structure of the datasets. In particular, we see that the WebKB datasets consist of a smaller number of nodes, edges and classes. Moreover, the proportion of the number of edges to nodes are: (1) Cora = 2.6, (2) Citeseer = 1.3, (3) WebKB – Cornell = 1.5, (4) WebKB – Texas = 1.7, (5) WebKB – Washington = 1.94 and (6) WebKB – Wisconsin = 2. This hints that, increasing the density of the graph also implies increasing the classification accuracy. This is consistent with the results, where large gains can be seen with small problem instances whose graph structure is denser. Another point of interest is the level of accuracy across these datasets. We notice that the accuracy in the Cora dataset is higher as compared to that yielded for the Citeseer dataset. This is consistent for both our method and the alternative. Furthermore, we see that the range is large for the WebKB datasets (47.4% to 90%), which is not dissimilar to previous results. This is also consistent to the intuitive notion that, in a dense graph, the local support is provided by more edges within a particular neighbourhood, leading to improved classification accuracy.

6 Conclusions

In this paper, we have presented a method to integrate low and high-pass filters into convolutional layers in GCNNs. Our method is quite general in nature and applies to both, directed and undirected graphs. We used concepts from spectral graph theory, and exploited the relationship between eigenvalues and the modes of the graph to incorporate high-frequency information in the learning process. We illustrated the utility of our method for classification making use of widely available citation datasets and compared our results against those yielded by an alternative. In our experiments, our method outperforms the alternative, providing a margin of improvement in the classification accuracy.

References

1. Dou, W., Zhang, X., Liu, J., Chen, J.: HireSome-II: towards privacy-aware cross-cloud service composition for big data applications. IEEE Trans. Parallel Distrib. Syst. **2**(26), 455–466 (2013)
2. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. In: International Conference on Learning Representations (2016)
3. Fukushima, K., Miyake, S., Ito, T.: Neocognitron: a neural network model for a mechanism of visual pattern recognition. IEEE Trans. Syst. Man Cybern. B Cybern. **2**(5), 826–834 (1983)
4. TaubinÝ, G.: Geometric signal processing on polygonal meshes. In: Proceedings of EUROGRAPHICS 2000: state of the art report (2000)

5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
6. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, pp. 2278–2324. IEEE (1998)
7. Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **2**(20), 61–80 (2008)
8. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag. **2**(30), 83–98 (2013)
9. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**, 4–24 (2020)
10. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: a review of methods and applications. ArXiv preprint arXiv:1812.08434 (2018)
11. Spielman, D.A.: Algorithms, graph theory, and linear equations in Laplacian matrices. In: 4th Proceedings of the International Congress of Mathematicians, pp. 2698–2722. Plenary Lectures and Ceremonies Vols. II-IV: Invited Lectures, World Scientific (2010)
12. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017)
13. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013)
14. Sarkar, S., Boyer, K.L.: Quantitative measures of change based on feature organization: Eigenvalues and Eigenvectors. Comput. Vis. Image Underst. **17**(1), 110–136 (1998)
15. Narang, S.K., Ortega, A.: Perfect reconstruction two-channel wavelet filter banks for graph structured data. IEEE Trans. Signal Process. **60**(6), 2786–2799 (2012)
16. Weiss, Y.: Segmentation using eigenvectors: a unifying view. In: International Conference on Computer Vision, pp. 975–982 (1999)
17. Xiang, G., Wei, H., Ongming, G.: Exploring structure-adaptive graph learning for robust semi-supervised classification. In: 2020 IEEE International Conference on Multimedia and Expo (ICME), pp. 1–6 (2020)
18. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256 (2010)
19. McCallum, A.K., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. Inf. Retrieval **3**(2), 127–163 (2000)
20. Giles, C.L., Bollacker, K.D., Lawrence, S.: CiteSeer: an automatic citation indexing system. In: Proceedings of the third ACM conference on Digital libraries, pp. 89–98 (1998)
21. Wang, Y., et al.: A data cleaning method for citeseer dataset. In: Cellary, W., Mokbel, M.F., Wang, J., Wang, H., Zhou, R., Zhang, Y. (eds.) WISE 2016. LNCS, vol. 10041, pp. 35–49. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48740-3_3
22. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations (2014)