Graph Convolutions and Machine Learning

Acknowledgments

I am indebted to my thesis advisor, Professor Salil Vadhan, for his helpful comments throughout the writing process. His support substantially improved both the content of the thesis and its presentation. I am grateful also to Professor Michael Bronstein of the Università della Svizzera italiana (Switzerland) and Tel Aviv University, a 2017–2018 Fellow of the Radcliffe Institute for Advanced Study at Harvard, and to Federico Monti, PhD candidate at the Università della Svizzera italiana, both of whom helped me to come to understand this area and deep learning more broadly. My exposure to this topic and to the work discussed in Section 4.4 of this paper came as a result of my work as a research assistant to Professor Bronstein at the Radcliffe Institute. I am grateful for his generous support.

Contents

Chapter 1

Introduction

In recent years deep learning has produced significant improvements in the field of machine learning. Some of the greatest successes have come from the application of convolutional neural networks to images and audio. Convolutional neural networks have recently surpassed human performance on some image classification tasks [19]. Convolutional networks have also seen commercial deployment. Reportedly, all of the approximately two billion images uploaded to Facebook each day pass through four convolutional neural networks for content filtering and annotation [23]. In this work, we will present a generalization to new data domains of some of the core operations used inside these networks. Specifically, this paper considers graph settings where the operation of convolution is not as easily defined as in Euclidean domains. This allows building networks similar to those on Euclidean data sets and carrying out deep learning on data with irregular network structure.

Many of the problems considered in machine learning contexts—specifically *supervised* learning—can be phrased as problems of function approximation. As a specific illustration, consider the case where one has a data set of images and would like to approximate a function labeling them:

$$f : \{\text{Image}\} \longrightarrow \{\text{Label}\}, \tag{1.1}$$

where {Image} is the space of all images of some size and {Label} is a set of discrete labels, describing the content of the image ("dog," "cat," etc.). This function f is ill-defined and operates on a domain of very high dimension. Both of these factors make it impossible—or nearly so—to make practical use of an exact construction of f, so instead we will seek an approximation.

Traditional approaches to such problems are largely human-driven, requiring great manual effort to develop models of the task. Humans select and populate a basis set in an effort to capture the important aspects of the problem, by which the target function may be approximated. In many cases these models take the form of complex, parameterized statistical models or sets of fundamental features which are then tuned to build a model.

Deep learning, by contrast, generally constructs models which are less theoretically complex but more highly parameterized. Neural networks are a particular instance of such models. In their

simplest form they are constructed from layers of alternating linear and non-linear maps

$$X_{(\ell+1)} = \phi(W_{(\ell)}X_{(\ell)}) \tag{1.2}$$

where $X_{(\ell)}$ is a vector produced by the previous layer, $W_{(\ell)}$ is a matrix representing a linear map (the parameters of the layer), and ϕ is a non-linear map applied over the entries of the resulting vector. A common choice of non-linearity is the ReLU:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1.3}$$

Each individual layer is simple and limited, but combining many such layers produces a structure capable of approximating many functions.

Neural networks are fitted to a data set through training. To complete the model, a *loss function* is produced which expresses in a single number the "error" of the network. Generally these numerically encode the accuracy of the results of the network over the training data. Critically, both the loss function and all layers of the network are differentiable. The network is trained by updating the layer parameters $W_{(\ell)}$ through gradient descent. The gradient of the loss function is computed for each parameter and these are iteratively modified in order to decrease the loss value. If this process successfully converges, the network may provide a good approximation for the particular task. Modern machine learning software packages are capable of automatically computing and applying gradients through very deep networks, so what remains is largely to design a suitable network by choosing the composition of the layers.

Producing neural networks which are sufficiently rich to approximate complicated functions often requires them to have many layers and consequently many parameters. This presents many difficulties, among which are the computational cost of the network training and the large amounts of training data required to successfully train these networks. One approach to reducing these difficulties is to build networks which have been specially tailored to a particular problem domain. If certain properties of the machine learning problem are "built in" to the network this can allow the constructed networks to have fewer parameters and to more successfully solve the targeted task. An example of this approach is the convolutional neural network.

1.1 Convolutional Neural Networks

Convolutional neural networks are neural networks in which the linear map in some layers is replaced with a convolution (see Definition 1.1.1). This produces layers of the form:

$$X_{(\ell+1)} = \phi(X_{(\ell)} * g_{(\ell)}) \tag{1.4}$$

where $g_{(\ell)}$ is a convolution "kernel." In the discrete case of images the convolution kernel can be imagined as a small matrix of values which is walked over the input image, computing a two-dimensional analogue of the dot product at each location. Multiple kernels can also be combined

to jointly process a set of input signals and produce multiple output signals. Such constructions yield convolution kernels represented as higher-dimensional arrays.

Definition 1.1.1 (Convolution)**.** The convolution operation is defined as follows:

$$(f * g)(x) = \int_{\mathbb{R}^d} f(t) \cdot g(x - t)\ dt$$

where $f, g : \mathbb{R}^d \longrightarrow \mathbb{R}$ are two real-valued signals on a d-dimensional Euclidean space.

The convolution operation is commonly applied in signal processing contexts. In these applications, though, the convolution kernels are often hand-crafted for a particular task. In a deep learning context, the convolution kernels are produced in the same way as the parameters in the linear layers from Equation 1.2: that is, trained by gradient descent.

One primary benefit of convolutions is that they bring parameter reductions. In the discrete neural network context, convolution kernels are manipulated as small vectors or arrays of values. The size of these kernels can be explicitly controlled independently of the size of the input. Thus, the number of parameters in a single layer can be limited to an almost arbitrarily small number.

A second benefit is that incorporating convolutions in a network can, for some applications, bring increased network performance at a smaller number of parameters. The small convolution kernels used in neural networks are localized. Localized, that is, in the sense that a particular output of a convolutional layer depends only on the values of a small set of neighboring input values. For images and audio, for example, it is reasonable to expect that features of interest are local, and thus may be measured without considering values from across the entire input signal. Further, the convolution operators also add shift-invariance to the network. Again, for images and audio problems this is often desirable. Classifying the objects in an image should not depend on where within the image these objects are located. The content of an image or of an audio sample is not affected by translations in space or time. Because the convolution operator is swept across the entire input, this invariant is enforced.

The linear maps of a plain neural network can encode the behavior of a convolutional layer, but encoding this structure into the network operations themselves makes this possible with vastly fewer parameters. However, a linear layer with N inputs and M outputs requires an $M \times N$ matrix of parameters. A convolutional layer with support K producing M outputs requires $M \times K$ parameters to encode. For small K, this is much more efficient and independent of the size of the input data, which can be large. This reduction in parameters also serves to reduce the amount of training data needed to tune a network.

1.2 Deep Learning on Non-Euclidean Data

The data sets discussed above as illustrations of deep learning problems have a regular, Euclidean structure. The input values of images and audio are easily represented as grids of numbers of an appropriate dimension. Many modern data sets, however, do not have such a simple structure:

social, citation and transportation networks as well as networks imposed on a data set through some measure of similarity are examples of data sets that have a complex structure. Such structures are more rich than Euclidean data sets, but are consequently more complex and difficult to process.

The benefits of convolutions on Euclidean data sets, namely localization and parameter reductions, are similarly desirable for problems defined on non-Euclidean domains. Additionally, it is reasonable to expect that such data sets might exhibit locality similar to that seen in images and audio. Information about users in a social network or papers in a citation network, for example, is likely more strongly dependent on close neighbors than on distant members. Given these considerations, it would be desirable to produce networks with benefits analogous to those of a standard convolutional neural network: locality and parameter reduction. However, the operation of convolution does not directly generalize to non-Euclidean domains and will require the development of a suitable analogy.

Just as convolutional neural networks built on existing techniques in signal processing, there has been work to generalize signal processing techniques to graphs [32]. Combining these techniques with deep learning methods and suitable data sets may enable similarly significant improvements to machine learning results on graph data. One approach to generalize convolutions to graphs makes use of notions from spectral graph theory. We present this generalization in Chapter 2. In Chapter 3 we consider some computational issues that arise in practical deep learning applications. Finally, in Chapter 4 we present some works which make use of graph convolutions as well as some extensions of these network architectures. Among the extensions presented in this chapter is an original extension of graph convolutional neural networks developed as a result of research in which the author of this paper participated.

1.3 Contributions of this Thesis

This paper provides a detailed description of a generalization of convolutions to data defined over graphs and discusses applications of such an operator in neural networks. The paper makes an expository contribution to the existing literature in this area through its detailed discussion of certain basic deep learning concepts in the introductory sections, as well as the detailed development of the relevant theorems and background presented in Chapter 2. This should make the discussion more accessible to readers who are unfamiliar with the subject areas combined here. In particular, Section 2.4 provides some justification for accepting the analogy used to define convolutions on graphs by examining a link to the Discrete Fourier Transform and makes use of group characters in doing so. This particular method of presenting this link has not, as far as the author is aware, been used in a similar context.

Similarly, the discussion of computational concerns in Chapter 3 provides a detailed discussion of issues that arise in applying graph convolutions as a part of neural networks. This chapter discusses in similar detail an approach to reduce the computational cost of the graph convolutions in this case. This detailed treatment of both the theoretical background and computational concerns is provided in order to enhance the usefulness of the exposition to readers with less familiarity

with deep learning in general.

An additional contribution of this paper is its collection and presentation of references to current research using the techniques discussed here. Finally, this paper contributes to the development of new techniques for deep learning on graphs through its presentation of an extension of graph convolutions developed during a research project on which the author worked as a research assistant. This extension demonstrates an original approach to processing information on directed graphs. The graph convolutions discussed in Chapter 2 require that the graphs considered be undirected; this new method seeks to lift this restriction. The results of this research are preliminary, but they yield improvements in testing over a graph data set.

Chapter 2

Convolutions on Graphs

In order to produce convolutional neural networks on graph-structured data we require a corresponding notion of convolution. However, the convolution integral given in Definition 1.1.1 would require us first to make sense of translation. In a Euclidean space this is readily understood, but it is not clear how to interpret $g(x - t)$ for a graph signal.

In order to define a convolution on a graph we will proceed by *analogy*. To this end, we search for some property of convolutions that we can realize in a graph domain. We find such properties in the fact that in the frequency domain—under the Fourier transform—convolutions become multiplication, and in the relationship of the Fourier transform with the Laplacian operator which is an object of study in spectral graph theory. We begin by introducing the standard Euclidean forms of these objects.

2.1 The Fourier Transform and Convolutions

The Fourier transform expresses a real-valued function as a combination of various frequencies of complex exponentials $e^{-2\pi i\xi}$ [7], [32], [33]. A function $f : \mathbb{R} \longrightarrow \mathbb{R}$ can be written as a combination of frequencies of such exponentials, each scaled by $\hat{f}$, the Fourier transform of f. That is,

$$f(x) = \int_{\mathbb{R}} \hat{f}(\xi) e^{2\pi i x\xi} \, d\xi \qquad \text{where} \qquad \hat{f}(\xi) = \int_{\mathbb{R}} f(t) e^{-2\pi i \xi t} \, dt. \tag{2.1}$$

In general, for $\hat{f}$ to exist we require that f be bounded and continuous [14]:

$$\int_{\mathbb{R}} |f(x)| \, dx < \infty, \tag{2.2}$$

so that integrals may be taken over the real line. Provided that f meets this condition and $\hat{f}$ is similarly well-behaved, the Fourier transform provides a reversible conversion of f from the *spatial* domain (or, in standard signal processing, the *time* domain) to the *frequency* or *spectral* domain. We limit discussion of the properties and requirements of the standard Fourier transform as we

will work to generalize the Fourier transform to graphs. Discussion of the formal properties of the Euclidean Fourier transform can be found in standard texts on the subject [7], [14].

The Convolution Theorem shows that convolution in the spatial domain corresponds to a point multiplication in the spectral domain. This property will be important for our generalization.

Theorem 2.1.1 (Convolution Theorem)**.** *Convolution in the spatial domain corresponds to multiplication in the spectral domain. That is,* $\widehat{(f * g)} = \hat{f} \cdot \hat{g}$

Proof. We briefly prove Theorem 2.1.1 [21]. This requires use of some calculus.

$$\begin{aligned}
\widehat{(f * g)}(\xi) &= \int_{\mathbb{R}} (f * g)(t) e^{-2\pi i \xi t} \, dt \\
&= \int_{\mathbb{R}} \left(\int_{\mathbb{R}} f(u) \cdot g(t-u) \, du \right) e^{-2\pi i \xi t} \, dt \\
&= \int_{\mathbb{R}} \int_{\mathbb{R}} f(u) \cdot g(t-u) e^{-2\pi i \xi t} \, dt \, du \\
&= \int_{\mathbb{R}} f(u) \left(\int_{\mathbb{R}} g(t-u) e^{-2\pi i \xi t} \, dt \right) du
\end{aligned}$$

where we changed the order of integration by application of Fubini's Theorem. Next we make a change of variables with $v = t - u$ so that $t = v + u$ and $dt = dv$:

$$\begin{aligned}
\int_{\mathbb{R}} f(u) \left(\int_{\mathbb{R}} g(v) e^{-2\pi i \xi (v+u)} \, dv \right) du &= \left(\int_{\mathbb{R}} f(u) e^{-2\pi i \xi u} \, du \right) \cdot \left(\int_{\mathbb{R}} g(v) e^{-2\pi i \xi v} \, dv \right) \\
&= \hat{f}(\xi) \cdot \hat{g}(\xi)
\end{aligned}$$

□

Theorem 2.1.1 will be a major component in our definition of a convolution operator on graphs. The other main component will come from the relationship between the Fourier transform and the Laplacian operator.

Definition 2.1.2 (Euclidean Laplacian Operator)**.** Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. The Laplacian of f, denoted Δf, is the sum of all pure second-order partial derivatives of f: $\Delta f = \sum_{\ell=1}^{n} \frac{\partial^2 f}{\partial x_\ell^2}$.

The complex exponentials that appear in the Fourier transform are eigenfunctions of the Laplacian [8], [32], [33]. In the single variable case:

$$\Delta e^{-2\pi i \xi x} = -4\pi^2 \xi^2 e^{-2\pi i \xi x}. \tag{2.3}$$

When we take the exponentials $e^{-2\pi i \xi}$ as elements of $L^2([0,1])$, the space of square-integrable functions on the interval:

$$f \in L^2([0,1]) \implies \int_0^1 |f(x)|^2 \, dx < \infty \text{ and exists}, \tag{2.4}$$

with inner product

$$\langle f, g \rangle = \int_0^1 f(t) \cdot \overline{g(t)}\, dt, \tag{2.5}$$

and $\xi \in \mathbb{Z}$ then the exponentials are also orthonormal. This gives us values for the inner product of our "basis" functions [21]:

$$\begin{aligned} \xi, \xi' \in \mathbb{Z} \implies \langle e^{-2\pi i \xi}, e^{-2\pi i \xi'} \rangle &= \int_0^1 e^{-2\pi i \xi t} \cdot \overline{e^{-2\pi i \xi' t}}\, dt \\ &= \int_0^1 e^{-2\pi i (\xi - \xi') t}\, dt = \begin{cases} 1 & \text{if } \xi = \xi' \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{2.6}$$

We note that this integral is zero when $\xi \neq \xi'$ as over $[0, 1]$ each exponential will go through a full period. From this perspective the Fourier Transform gives us a decomposition of a function f into a combination of eigenfunctions of the Laplacian operator and under suitable restrictions these are also orthogonal. This will be our point of departure for generalizing the convolution operator to graphs.

2.2 Graph Laplacians

The connection between our definitions for Euclidean space and our generalization for graphs is derived from spectral graph theory. Before continuing, we first define undirected graphs and their associated graph Laplacian matrices [12]. These matrices will be the central objects that we will use to generalize convolutions [8], [32].

Definition 2.2.1 (Weighted Undirected Graph)**.** We define an undirected graph $G = (V, E, W)$ with V a set of vertices, E a set of edges and W its weighted adjacency matrix.

For W we require that $(v_i, v_j) \in E \implies W_{i,j} = W_{j,i} > 0$ and $(v_i, v_j) \notin E \implies W_{i,j} = W_{j,i} = 0$. This ensures that edges have positive weight.

Definition 2.2.2 (Graph Laplacian)**.** Let $d_i = \sum_{j=1}^{n} W_{i,j}$ be the weighted degree of vertex i and let D be an $n \times n$ diagonal matrix such that $D_{i,i} = d_i$. We then define the *unnormalized* graph Laplacian to be $L = D - W$ and the *normalized* graph Laplacian to be $\mathcal{L} = D^{-1/2} L D^{-1/2}$. We take $d_i^{-1/2} = 0$ if $d_i = 0$.

The connection to the Euclidean Laplacian can be motivated through a discretization of the gradient and divergence operators on graphs [8]. In the Euclidean case for a function f, the Laplacian is the divergence of the gradient: $\Delta f = \operatorname{div} \nabla f$.

Now taking $f : V \longrightarrow \mathbb{R}$ to be a function defined on the vertices of a graph G the operations of gradient and divergence may be discretized. The gradient may be viewed as an operator taking a function on the vertices of the graph to a function on the edges, and the divergence may be viewed

as an operator which gathers values from an edge function at the vertices. That is:

$$(\nabla f)(i,j) = \sqrt{W_{i,j}} \cdot (f(i) - f(j)) \tag{2.7}$$

$$(\operatorname{div} F)(i) = \sum_{j:(i,j)\in E} \sqrt{W_{i,j}} \cdot F(i,j) \tag{2.8}$$

where $f : V \longrightarrow \mathbb{R}$ and $F : E \longrightarrow \mathbb{R}$ so that $\nabla f : E \longrightarrow \mathbb{R}$ and $\operatorname{div} F : V \longrightarrow \mathbb{R}$.

These operators can be produced on the discrete graph space from the vertex-edge adjacency matrix [17]. Let A be this matrix; it has dimension $|V| \times |E|$ and entries given by:

$$A_{v,e} = \begin{cases} \pm\sqrt{W_{v,w}} & \text{if } e = (v,w), \text{ that is incident to } v \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

Each edge has two nonzero entries in this matrix, one for each possible sign. These values induce an orientation (though not a direction) on the edge which will not affect the results for our purposes. With this we can realize the "gradient" as application of A^T, and the divergence as application of A. With these definitions we examine $\operatorname{div} \nabla f$:

$$\begin{aligned} (\operatorname{div} \nabla f)(i) &= \sum_{j:(i,j)\in E} \sqrt{W_{i,j}} \cdot \nabla f(i,j) = \sum_{j:(i,j)\in E} W_{i,j}\big(f(i) - f(j)\big) \\ &= D_{i,i} f(i) - \sum_{j\in N_i} W_{i,j} \cdot f(j) = (Lf)_i \\ &\implies AA^T = L \end{aligned} \tag{2.10}$$

where N_i denotes the one-hop neighborhood of vertex i. This also shows, as noted above, that $L = AA^T$ [17]. Equation 2.10 recovers the unnormalized graph Laplacian as a generalization of the Euclidean Laplacian operator. $\mathcal{L}$ performs a similar operation, although with normalization by vertex degrees.

As discussed in Section 2.1 the complex exponentials can be viewed as, in some sense, diagonalizing the Euclidean Laplacian operator. Similarly, the graph Laplacians L and $\mathcal{L}$ can be diagonalized.

Theorem 2.2.3 (Graph Laplacians Eigenbasis)**.** *There exists an orthonormal matrix U such that $L = U^T \Lambda U$ where Λ is a diagonal matrix. An analogous decomposition of $\mathcal{L}$ exists.*

Proof. For undirected graphs, L and $\mathcal{L}$ are symmetric real-valued matrices, and therefore they have a complete orthonormal basis of eigenvectors. Take U^T to have these vectors as columns, then $U^{-1} = U$ with U^T are the needed matrices. The results used in this proof are provided in texts on Linear Algebra [4]. □

The fact that L and $\mathcal{L}$ have eigenbases will allow us to generalize convolution to graphs. Our procedure will make use of the *spectrum* of our graph, its set of eigenvalues. Lemma 2.2.4 summarizes some properties of the graph spectrum.

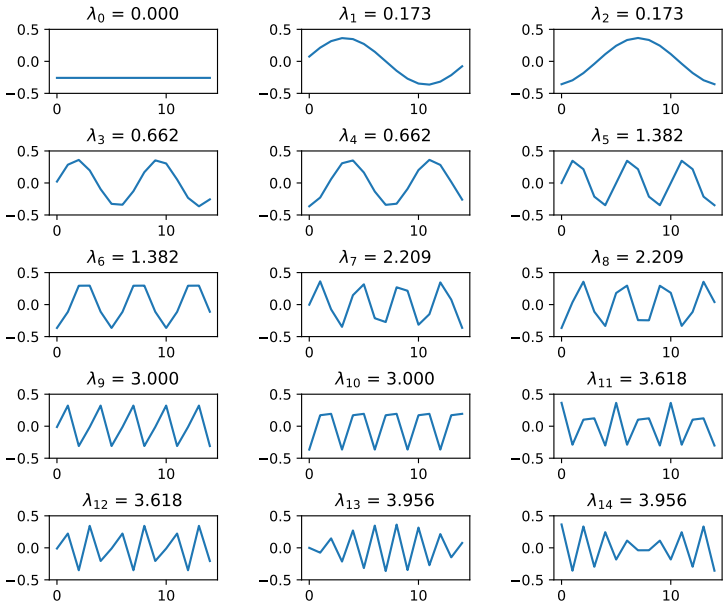

Figure 2.1: Eigenvectors of a 15-cycle [1], [29].

Lemma 2.2.4 (Graph Spectrum)**.** *In increasing order, the eigenvalues of L satisfy*

$$0 = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{n-1}.$$

The same applies to the eigenvalues of $\mathcal{L}$.

Proof. Because L is real and symmetric its eigenvalues are real [4]. We note that $\lambda_0 = 0$ as the all ones vector $\vec{1}$ has eigenvalue zero.

$$(L\vec{1})_i = (D\vec{1})_i - (W\vec{1})_i = d_i - \sum_j W_{i,j} = d_i - d_i.$$

In fact, by similar logic we have a zero eigenvector for each connected component of the graph G. Take v_i to have ones on the connected component and zeros elsewhere, and repeat the above computation [12]. $\mathcal{L}$ has an eigenvector with eigenvalue zero by a similar analysis, after appropriately scaling the entries of the vector.

The non-negativity of the eigenvalues follows from the fact that L is positive semidefinite which can be shown from the fact that it decomposes as $L = AA^T$ as in Equation 2.10 [4]. □

One other point of similarity between the eigenbasis of the Graph Laplacian and the complex exponentials used in the Fourier transform is a loosely analogous notion of frequency. The Fourier transform decomposes a continuous signal into a combination of basic frequencies. In some sense

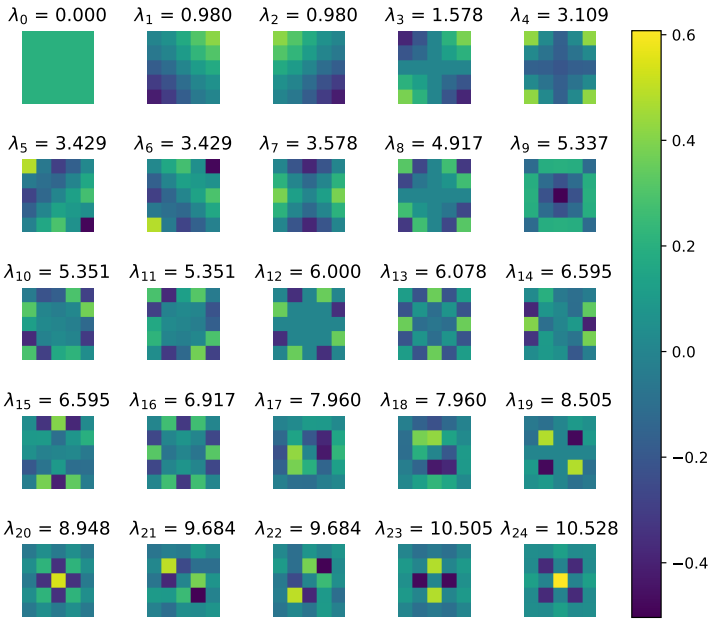

Figure 2.2: Eigenvectors of a 5 × 5 Euclidean grid.

the graph eigenvectors in order of increasing eigenvalue capture a similar notion of frequency, although this can be more difficult to define and analyze.

In a particularly simple case, eigenvectors on a simple cycle of vertices generate values which have a wave-like pattern with frequencies that increase with the eigenvalues [1]. This mirrors the increasing frequency of the complex exponentials of the Fourier transform. A plot of such eigenvectors is included in Figure 2.1. Similar results hold for graphs which are "shift invariant" in the sense that we can order the vertices to produce a circulant Laplacian matrix [17]. In this case, the eigenvalues and eigenvectors of the graph Laplacian will behave similarly to the complex exponentials used in Fourier transforms in Euclidean space. This is discussed further in Section 2.4.

Another case which can be visualized is the case of a finite grid. We produce a grid of vertices each joined to neighbors within unit distance under the ℓ_∞ norm: that is, immediate neighbors in all cardinal directions, and along diagonals. Eigenvectors for such a grid are included in Figure 2.2. Here, a notion of frequency is less obvious. One way to conceptualize frequency is the number of "zero crossings" [32]. For larger eigenvalues, neighboring vertices are more likely to have values opposite in sign. This trend is borne out in other graph topologies and is visible on the grid, as shown in Figure 2.2.

2.3 Spectral Convolutions

We will define a convolution operation on graphs making use of the spectral domain obtained through the Laplacian. First we define an analogue of the Fourier Transform for functions on graph vertices.

Definition 2.3.1 (Graph Fourier Transform)**.** Let $f : V \longrightarrow \mathbb{R}$. Define the Fourier Transform of f as $\hat{f} = Uf$ where U is as in Theorem 2.2.3.

The transformation U performs a change of basis on the vertex function f, representing it as a linear combination of Laplacian eigenfunctions. This can be viewed as a discrete analogue of the Fourier Transform based on our discussion in Section 2.1. We make use of this operation to define convolution on vertex functions by analogy with the Convolution Theorem (2.1.1). This approach is used in several works [8], [20], [32].

Definition 2.3.2 (Graph Convolution)**.** Let $f, g : V \longrightarrow \mathbb{R}$. Define $f * g = U^T(Uf \odot Ug)$, where $\odot$ denotes a point multiplication: $(u \odot v)_i = u_i \cdot v_i$.

Notice that Definition 2.3.2 respects the Convolution Theorem, as it was constructed to do:

$$\widehat{(f * g)} = U(U^T(Uf \odot Ug)) = Uf \odot Ug = \hat{f} \odot \hat{g}. \tag{2.11}$$

The convolution kernel g, above, can be defined as a function over the vertices $g : V \longrightarrow \mathbb{R}$ or can be defined directly over the graph spectrum, as a function of the eigenvalues. We can take $g : \mathbb{R} \longrightarrow \mathbb{R}$ and represent it as a vector $\vec{g} = [g(\lambda_0), \dots, g(\lambda_{n-1})]$. Then our convolution becomes:

$$f * g = U^T(Uf \odot \vec{g}) = U^T(Uf \odot g(\Lambda)). \tag{2.12}$$

Through the graph Fourier Transform these representations of convolution kernels are largely interchangeable. We can view the kernel g as a convolution with a function on the vertices of G; as a function on the real line; or as a function only on the eigenvalues of the graph Laplacian. One difference, however, lies in the "expressiveness" of the convolution kernels which can be defined through these two methods. When defining a kernel over the graph spectrum—rather than over the vertices—the response on eigenvectors with eigenvalues of multiplicity greater than one is forced to be equal. These repeated eigenvalues have vectors which are often equivalent under symmetries of the graph, or over different connected components. Examples of symmetric eigenvectors can be clearly seen in Figure 2.2. Some repeated frequencies also appear in Figure 2.1. Even though the spectral kernel construction loses the ability to distinguish among repeated eigenvalues, we will make use of these alternate kernel representations when we analyze properties of graph convolutions due to the many computational benefits that come from this approach, as discussed in Chapter 3.

To make use of this convolution operation in a neural network we can form convolution layers using the above operator. We then update the convolution kernels g by gradient descent as usual. These convolutions are made of simple matrix and point multiplications. Existing deep learning frameworks are capable of computing gradients through these operations, which makes including these spectral convolutions in a neural network relatively straightforward.

2.4 Discrete Fourier Transform

One particularly interesting instance of the graph Fourier transform given in Definition 2.3.1 results when it is applied to torus graphs which have regular, cyclic structure. In this case the eigenvectors which appear in U form the basis used in the standard Discrete Fourier Transform (for an illustration see Figure 2.1). This special case of our definition lends some support to accepting it as a generalization of the Fourier Transform to general graph structures.

The Discrete Fourier Transform is a discretization of the standard Fourier transform for finite sequences of values $x_0, \dots x_N$. This can alternatively be viewed as sampling a periodic function f on $[-\pi, \pi]$ at evenly spaced points $t_k = \frac{2\pi k}{N}$ with $k = -N/2, \dots, \frac{N}{2} - 1$ for even N [31]. An analogous subdivision of the interval exists for odd N. The Discrete Fourier Transform then represents f at these points as:

$$f(t_k) = \sum_{n=-N/2}^{N/2-1} c_n e^{2\pi i k n/N} \tag{2.13}$$

for some coefficients c_n (which are the result of the Discrete Fourier Transform) [31]. The exponentials $e^{2\pi i k n/N}$ form the Discrete Fourier Transform basis.

Our path to recovering these from our graph Fourier transform will make use of Cayley graphs (Definition 2.4.1). Since we are working towards a correspondence with the Discrete Fourier Transform we will consider only Abelian groups, although Cayley graphs can be realized more generally.

Definition 2.4.1 (Cayley Graph)**.** Let $\mathcal{G}$ be an Abelian group and $S \subset \mathcal{G}$. We may then define a graph $G = \Gamma(\mathcal{G}, S)$ with vertices given by the members of G and edges defined by

$$E = \{(g, s + g) \mid g \in \mathcal{G}, s \in S\}.$$

In particular, if $s \in S \implies -s \in S$ the resulting graph will be undirected [9].

To produce torus graphs of arbitrary dimension, we take our group $\mathcal{G}$ to be a product of cyclic groups $\mathcal{G} = C_{n_1} \cdot \times \cdot C_{n_\ell}$ [1]. Each cyclic group has a generator g_i of order n_i. We take these and their inverses, each paired with the additive identity element 0 from each cyclic group as our set S. That is, $S = \{0 \times \cdots \times \pm g_i \times \cdots \times 0 \mid \forall i\}$. This produces a discrete torus graph $\Gamma(\mathcal{G}, S)$. Each vertex is connected to $|S|$ neighbors, making this graph $|S|$-regular. From our discussion in Definition 2.4.1, this graph is undirected and thus has a symmetric adjacency matrix W as described in Definition 2.2.2. Here we take all entries to satisfy $W_{i,j} \in \{0, 1\}$. In connecting this to the Discrete Fourier Transform we will make use of the relationship between W and the characters of the group $\mathcal{G}$. The necessary theory of group characters is introduced in various sources on algebra [3].

[1] As this group is finite and Abelian, we could also insist that each cyclic group have prime power order and produce an isomorphic group, by the structure theorem [3].

Theorem 2.4.2 (Characters and Eigenvectors)**.** *Each character χ_i of the finite Abelian group $\mathcal{G}$ induces an eigenvector on adjacency matrix W of the Cayley graph $G = \Gamma(\mathcal{G}, S)$, when the character is realized as the vector $\vec{\chi}_{i_g} = \chi_i(g)$ for $g \in \mathcal{G}$.*

Proof. We note that as $\mathcal{G}$ is finite and Abelian, its characters are one-dimensional [3, Sec. 10.4]. Therefore $\chi_i : \mathcal{G} \longrightarrow \mathbb{C}^\times$ is a group homomorphism. From this we can verify that $\vec{\chi}_i$ is indeed an eigenvector of W, as in [35]. Consider the g^{th} entry of $W\vec{\chi}_i$, corresponding to $g \in \mathcal{G}$.

$$\begin{aligned}(W\vec{\chi}_i)_g &= \sum_{h\in\mathcal{G}} W_{g,h}\vec{\chi}_{ih} = \sum_{h\in N_g} W_{g,h}\vec{\chi}_{ih} \\ &= \sum_{h=sg} \vec{\chi}_{ih} \\ &= \sum_{s\in S} \vec{\chi}_{ig+s} \\ &= \vec{\chi}_{ig} \cdot \sum_{s\in S} \vec{\chi}_{is}\end{aligned}$$

where N_g denotes the set of vertices (equivalently group elements) adjacent to g in the Cayley graph. The intermediate steps make use of the fact that our adjacency matrix W is binary, so any nonzero entries (for edges) have value 1, and the fact that our characters respect the operation of $\mathcal{G}$.

The above demonstrates that $\vec{\chi}_i$ is indeed an eigenvector of W and also that its eigenvalue is given by $\lambda_i = \sum_{s\in S} \vec{\chi}_{is}$. □

The above theorem shows only that characters induce eigenvectors of the adjacency matrix W while our graph Fourier transform makes use of the graph Laplacian $L = D - W$. We note however that in the case of the Cayley graphs these matrices will share eigenvectors. Because the Cayley graphs are regular, with each vertex having $|S|$ neighbors, we note that $D = |S|I$ so that $L = D - W = |S|I - W$. Therefore if $\vec{x}$ is an eigenvector of W with eigenvalue λ we have:

$$L\vec{x} = (|S|I - W)\vec{x} = |S|I\vec{x} - W\vec{x} = |S|\vec{x} - \lambda\vec{x} = (|S| - \lambda)\vec{x} \tag{2.14}$$

which demonstrates that $\vec{x}$ is also an eigenvector of L.

Now we return to our specific finite Abelian group $\mathcal{G}$ introduced above. Because $\mathcal{G}$ is finite and Abelian it has $|G|$ irreducible characters [3], and thus a full basis of eigenvectors of W. Because the characters of such groups are one-dimensional (as discussed in the proof of Theorem 2.4.2), we have that $\chi_i(0) = 1$ where 0 is the element of $\mathcal{G}$ consisting of the identity element from each component cyclic group. Then taking $g_i \in S$, the generator of cyclic group C_{n_i}, and using the fact that every character, χ, is a group homomorphism, we have

$$\chi(g_i)^{n_i} = \chi(n_i \cdot g_i) = \chi(0) = 1. \tag{2.15}$$

Therefore $\chi(g_i) = \omega_{n_i}^k$ a power of a primitive n_i^{th} root of unity. These have the form $e^{2\pi ik/n_i}$ which is the form of the Discrete Fourier Transform basis elements. Therefore, the character values on

this graph are given by combinations of Discrete Fourier Transform exponentials, and these are the eigenvectors of W and L for the Cayley graph of such groups. Thus, these exponentials give rise to the elements of U used in the Graph Fourier Transform.

For real-valued symmetric matrices such as L, we note that we can also take only the real components of these eigenvectors, $\mathrm{Re}\,\vec{\chi}_i$. Therefore the real eigenvectors of the resulting Laplacians will be given by the cosine components of the roots of unity through Euler's Formula. This explains the values that are plotted in Figure 2.1, at least up to combinations of vectors in each eigenspace. Therefore our Graph Fourier Transform reduces to the standard Discrete Fourier Transform in this special case of Cayley graphs for finite Abelian groups.

Chapter 3

Computational Concerns

As noted in Section 2.3 modern deep learning frameworks simplify the inclusion of graph convolutional layers in a neural network architecture. Nevertheless there are several considerations that affect the usefulness of our spectral convolutions.

3.1 Issues with Spectral Convolutions

3.1.1 Kernel Orientation

One might think that if G takes the form of a Euclidean grid that Definition 2.3.2 will fully recover standard Euclidean convolutions. However, if we define our kernel g on the graph spectrum as in Equation 2.12 this will not be the case. This can be illustrated by considering a permutation matrix P. If we reorder the vertices of the graph we permute the entries of the Laplacian:

$$P^T L P = P^T U^T \Lambda U P. \tag{3.1}$$

The permuted Laplacian has the same spectrum as the original Laplacian. In particular, permutations which are symmetries of the graph will produce a Laplacian identical to the original. Due to the spectral construction we have no notion of orientation for our convolution kernels. In computer vision applications, for example, it is common for convolution kernels to detect oriented edges. In the spectral construction, however, we cannot leverage a notion of orientation. That is, spectral filters are isotropic [13]. One solution to this would be to construct convolution kernels spatially as functions over the vertices of the graph. This approach, however, has perhaps even more significant issues with the number of parameters (see Section 3.1.2).

3.1.2 Parameter Count

Another—perhaps more pressing—concern is the number of parameters required to determine a spectral convolution kernel. In our construction above, the convolution kernels require specifying values either for each vertex or for each eigenvalue. In both cases this is $O(n)$ parameters where

$n = |V|$, the number of vertices. One of the significant benefits of convolutions in Euclidean space is the ability to specify kernels using a number of parameters independent of the size of the input signal while under this spectral construction these values are linked.

3.1.3 Spatial Support

Closely related to this is the issue of the spatial support, the distance over which the convolution combines values. Euclidean convolutions are explicitly localized over a bounded region. Kernels are often defined over discretized regions as small matrices of parameters. These have support only within the region of defined values. By contrast, it is not as clear how arbitrary spectral multipliers g are localized over the vertices of a graph. Especially for applications where the dataset likely exhibits locality (for example citation or social networks), this may be undesirable. We seek a method to produce convolution kernels that are *localized* and manipulate information in a local way—or at least approximately local.

3.1.4 Diagonalization

Our spectral Graph convolution from Definition 2.3.2 requires fully diagonalizing the Laplacian L. The computational cost to do this with standard algorithms scales as $O(n^3)$ which may quickly become prohibitive for very large graphs. Furthermore, while L will be sparse for many real-world and constructed graphs, its Eigenbasis matrix U may be significantly more dense. This will require $O(n^2)$ dense matrix-vector multiplication as a part of the convolutional filtering [8], [13]. Ideally, our convolution operator should take advantage of the sparsity of L and allow us to avoid the need to diagonalize the graph Laplacian.

3.2 Convolution Kernel Bases

We will not be able, in general, to resolve the issue of isotropic convolution kernels as this is intrinsic to the spectral approach (Section 3.1.1), but we will offer an approach to alleviate the other issues raised in Section 3.1.

Convolution kernels defined over the eigenvalues of the graph will be isotropic as in the graph spectrum the notion of orientation is lost. See Figures 2.2 and 2.1 for a visualization of eigenvectors on a grid and cycle, respectively. Note, in particular, that for any symmetries in the graph, we have a set of eigenvectors which display the same symmetry.

Our approach to reduce the parameter count of the convolution kernels is to restrict the parameterizations. We do this by selecting a (small) set of basis kernels and producing spectral multipliers as a combination of these. By choosing a basis set of polynomials we will also resolve most of the issues raised above in Section 3.1 (all except for the issue of isotropic convolution kernels).

Ideally our basis would also produce kernels which are localized. One way to view this requirement is by analogy with the Euclidean case. For Euclidean spaces *smooth* spectral kernels

correspond to spatially localized weights [8], [10]:

$$\int_{\mathbb{R}} |x^k f(x)|^2 \, dx = \int_{\mathbb{R}} \left| \frac{\partial^k \hat{f}(\xi)}{\partial \xi^k} \right|^2 \, d\xi. \tag{3.2}$$

This equality in the Euclidean case is due to the Plancherel Theorem and induction on identities for the derivative of $\hat{f}$ [14]:

$$\hat{f}'(\xi) = \hat{g}(\xi) \qquad \text{where} \qquad g(x) = -2\pi i x f(x). \tag{3.3}$$

From this we derive that if $\hat{f}$ is smooth in the sense of having square-integrable k^{th} order derivatives, then f must decay faster than x^k, yielding spatial localization. One approach to inducing localization is to choose a spectral basis that is smooth and assume that this smooth-localized relationship continues to hold for graph convolutions. In fact, the Plancherel Theorem does generalize in a discrete sense to the graph domain:

Theorem 3.2.1 (Plancherel Theorem for Graph Spectra)**.** *The Euclidean Plancherel Theorem states that:*

$$\int_{\mathbb{R}} |f(x)|^2 \, dx = \int_{\mathbb{R}} |\hat{f}(\xi)|^2 \, d\xi$$

Because our graph convolutions take place over a discrete spectrum this statement becomes:

$$\sum_{\ell=0}^{n-1} |f_\ell|^2 = \sum_{\ell=0}^{n-1} |\hat{f}_{\lambda_\ell}|^2.$$

Proof. We note that $\sum_{\ell=0}^{n-1} |f_\ell|^2 = \langle f, f \rangle$ and $\sum_{\ell=0}^{n-1} |\hat{f}_{\lambda_\ell}|^2 = \sum_{\ell=0}^{n-1} |U f_{\lambda_\ell}|^2 = \langle Uf, Uf \rangle$. From this we see that this statement follows from the fact that U is orthonormal and thus preserves inner products of coordinates [4]. □

The analogy with the Plancherel theorem and Equation 3.2 is often used to describe the effect of spectral kernels on the localization of the resulting convolutions [8], [10]. Here, however, we approach this from the perspective of polynomial approximations which is discussed in [18] for the purpose of studying wavelets on graphs. The perspective of polynomial approximation is more directly related to the localization properties we wish to discuss and provides a clearer mechanism for linking spectral responses to spatial localization. We begin with a few definitions and will return to this topic in Section 3.3.3.

Definition 3.2.2 (Shortest Path Distance)**.** Let L be a graph Laplacian. The shortest path distance $d(i, j)$ between $i, j \in V$ is the shortest route along the edges of the graph from i to j. Equivalently, $d(i, j) = \min\{n \mid \delta_i^T L^n \delta_j \neq 0\}$. Take $d(i, j) = \infty$ if no path from i to j exists.

The formulation of shortest path distance from the graph Laplacian follows from the fact that the Laplacian is a localized difference operator [12], [18]. We demonstrate this here:

Lemma 3.2.3 (Localization of the Laplacian)**.** *Let $i, j \in V$, the graph Laplacian satisfies the relationship $d(i,j) > k \implies L^k_{i,j} = 0$*

Proof. We reproduce the proof of this statement given in [18]. We expand the matrix power L^k into its nested sum and find:

$$L^k_{i,j} = \sum_{\ell \in V^{k-1}} L_{i,\ell_1} L_{\ell_1,\ell_2} \cdots L_{\ell_{k-1},j}$$

where we sum over all sequences of length k (with $k-1$ "intermediate" vertices). Recall that $L_{i,j} = 0$ if $i \neq j$ and if there is *not* an edge between them. For this value to be nonzero at least one of the terms in the sum must be nonzero. This implies the existence of a path following the ℓ_i. This path then has length k. If there is no such path, each term will necessarily be zero as one of the "steps" will involve a non-existent edge. □

Lemma 3.2.3 says that L^k is k-localized under our definition of k-localized, given below.

Definition 3.2.4 (k-Localized)**.** A spectral kernel g is k-localized if $(\delta_i * g)_j = 0$ whenever $d(i,j) > k$. Intuitively this specifies that the kernel depends only on vertex values within a distance of k of a root vertex, i.

Definition 3.2.4 suggests that to produce spectral kernels that are k-localized, we can use a basis set which is k-localized or approximately so. Several basis sets have been used by different authors [8]: such as a basis of cubic splines (working from the idea of spectral smoothness discussed above) [10], or a basis of Chebyshev polynomials [13].

3.3 Polynomial Bases

The results for the Laplacian in Lemma 3.2.3 suggest a particularly attractive set of basis functions: polynomials over the graph spectrum. We define a kernel on the spectrum (as opposed to spatially on the vertices) $g : \mathbb{R} \longrightarrow \mathbb{R}$. Because $L = U^T \Lambda U$, if $g(x) = \sum_{\ell=0}^{k} a_\ell x^k$ is a polynomial of degree k this produces a filter:

$$\begin{aligned} U^T g(\Lambda) U &= U^T \begin{bmatrix} g(\lambda_0) & & \\ & \ddots & \\ & & g(\lambda_{n-1}) \end{bmatrix} U = U^T \left(\sum_{\ell=0}^{k} a_\ell \Lambda^k \right) U \\ &= \left(\sum_{\ell=0}^{k} a_\ell (U^T \Lambda U)^k \right) = g(L). \end{aligned} \tag{3.4}$$

Per Equation 3.4 polynomial filters can be realized not only as polynomials in the graph spectrum, but also as polynomials in the graph Laplacian itself.

With the exception of the issues of isotropic kernels, this approach solves all of the issues discussed in Section 3.1. A reduction in the parameter count can be achieved by limiting the degree of polynomials to k, producing kernels with $k+1$ parameters to learn. This learning

complexity is controllable independently of the size of the input graph. Further, because L^k is perfectly k-localized, so too are polynomial filters of degree k. This makes it possible to produce filters with configurable spatial support.

Perhaps most importantly for practical applications, applying these filters does not require diagonalizing the Laplacian. The filters are constructed of linear combinations of the original Laplacian matrix and do not require explicitly determining the spectrum of the graph. Applying the filter takes the form of a matrix multiplication by $g(L)$. Importantly, as L is highly sparse for many real-world networks, the filter matrix $g(L)$ will also have high sparsity for low values of k relative to the diameter of the graph. Multiplication by the sparse matrix L is proportional to the number of nonzero entries, that is, the edge count of the graph [13]. Leveraging the sparsity of L and avoiding expensive matrix diagonalization makes spectral convolutions more scalable to large graphs.

However, while L may be highly sparse, as the polynomial degrees rise the induced powers of the Laplacian L^k become increasingly dense. This is related to the radii of the graph's connected components. Once k becomes larger than one of these radii, the Laplacian will become dense on this component. This presents a computational concern when polynomial filters which have wide spatial support are desired.

One approach used to alleviate this issue is an application of graph coarsening, roughly corresponding to "pooling" used in standard convolutional neural networks. In such a setting the number of vertices is decreased by combining related vertices according to some measure. The Laplacian matrix for such pooled graphs is then significantly smaller so that the increased density is less of a concern. At the same time, because related vertices have been combined, a filter with the same spatial support on the pooled graph can correspond to a larger extent on the original graph. These approaches can be "spatial" in which vertices to be combined are selected within the graph structure, for example by greedily combining neighbors in order to minimize some score. Such an approach is used in [13] applying the coarsening phase of the algorithm described in [15]. Another approach is to cluster vertices "spectrally" by embedding them into Euclidean space through the Laplacian eigenvectors. One method is to apply the well-known k-means clustering to these embeddings [26]. Another possible approach may be to sparsify the higher powers of the Laplacian matrix itself. Efficient sparsification methods for Laplacian-type matrices have been proposed as in [11]. As far as the author is aware, such methods have not been applied in a machine learning setting but, for dense powers of the Laplacian, they may help alleviate some of the computational concerns.

Next we discuss two polynomial bases which are commonly applied in this context.

3.3.1 Chebyshev Polynomials

One common basis often used in polynomial approximation theory is the set of Chebyshev polynomials [6], [34]. These have several benefits for polynomial approximation [6]:

- Ease of computation – the Chebyshev polynomials are related to the Fourier transform so that the Fast Fourier Transform may be used to compute the coefficients of an approximation [34].

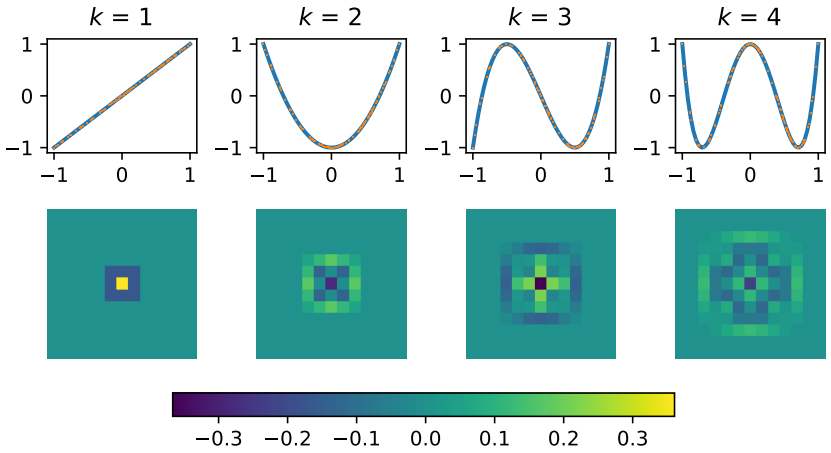

Figure 3.1: Chebyshev basis polynomials of order k convolved with a delta function.

- Fast convergence – even at relatively low degrees they are able to approximate many functions with low error.
- Completeness – the Chebyshev polynomials form a basis of functions on the interval $[-1, 1]$.

Of these benefits, only the last is likely to be especially useful in the context of graph convolution kernels. The ease of computing coefficients for a Chebyshev series is unlikely to be relevant in this context, as we set out to learn a spectral filter, and not to approximate an existing function. Using the Chebyshev basis to construct a polynomial by gradient descent may not benefit as significantly from the numerical properties of the Chebyshev polynomials which are used in approximations. The Chebyshev polynomials are used in existing works to produce spatially localized graph convolutional filters. One such work is discussed in Section 4.2 [13].

Definition 3.3.1 (Chebyshev Polynomials)**.** The Chebyshev Polynomials (of the first kind) can be computed by the recurrence relation [6], [13], [18], [34]:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x). \end{aligned}$$

As noted above, one benefit of the Chebyshev polynomials is their orthogonality over the interval $[-1, 1]$. This orthogonality is computed with respect to the inner product [6]:

$$\rho(T_m, T_n) = \int_{-1}^{1} \frac{1}{\sqrt{1 - x^2}} \cdot T_m(x) \cdot T_n(x)\, dx. \tag{3.5}$$

With this the Chebyshev polynomials form an orthogonal basis for $L^2([-1, 1], 1/\sqrt{1 - x^2})$ [13].

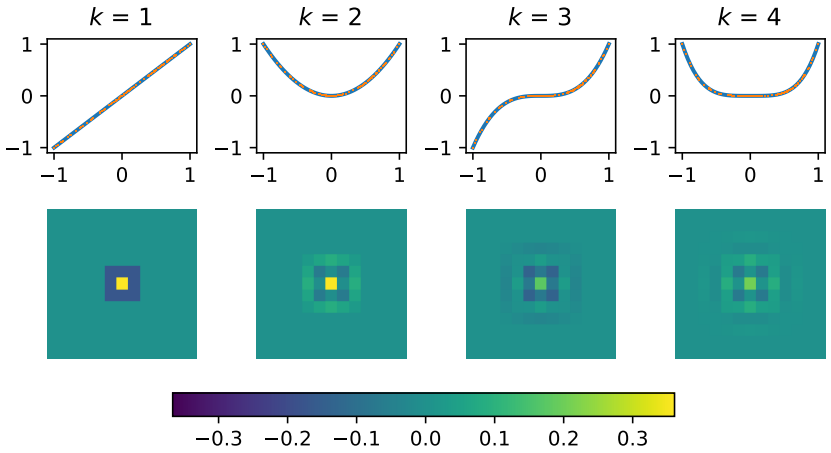

Figure 3.2: Monomial basis polynomials of order k convolved with a delta function.

The Chebyshev polynomials are designed to act over the interval $[-1, 1]$ while the graph spectrum does not necessarily lie within this range. To solve this problem we can re-normalize the Laplacian [13]

$$\tilde{L} = 2L/\lambda_{n-1} - I. \tag{3.6}$$

This produces $\tilde{L}$ which has eigenvalues in the range $[-1, 1]$. This normalization requires only the largest eigenvalue λ_{n-1}, which can be efficiently approximated using the power method [16].

A plot of the convolution responses of a delta function with Chebyshev basis polynomials is included in Figure 3.1. In each plot the delta function is positioned at the center vertex and a convolution with the corresponding Chebyshev polynomial is performed. The resulting values are plotted. The values of each Chebyshev polynomial over the real line of the graph spectrum are plotted in the top row. The lighter points on each line indicate the locations of the eigenvalues of the graph spectrum.

Because our kernels are defined on at most n distinct eigenvalues and because the Chebyshev polynomials are complete, by taking $T_0, \ldots, T_n$ we can produce any spectral kernel. This is equivalent to the trivial statement that any convolution kernel is n-localized.

3.3.2 Monomials

Another set of polynomial basis functions is simply the set of monomials. While these are generally not used in numerical approximation due to their ill-conditioning [6], [34] they may be suitable for the purpose of constructing spectral kernels.

A plot of the monomial basis functions of several degrees and the resulting convolution responses to a delta function are included in Figure 3.2. The configuration is the same as that used in Figure 3.1. From the figure we see that the monomial basis functions do appear less well-behaved

in their responses to the delta function. Even though they are k-localized and, like the Chebyshev basis, do respond to values at a distance of k from the weight of the delta function, the responses are weaker at higher orders. The Chebyshev polynomials plotted in Figure 3.1 have a stronger response further from the center at higher degrees. It is possible that the Chebyshev polynomials' stronger convolution response at higher degrees may produce better performance when used to produce spectral convolution kernels. Experiments empirically comparing the performance of these two bases across a wide variety of applications and neural network configurations would make for useful future work.

3.3.3 Polynomial Approximations

A natural direction to explore is the correspondence between convolution kernels defined spectrally and their corresponding localization. This is somewhat related to the "uncertainty principle" which, in the Euclidean case, describes a trade-off between spatial and spectral/frequency localization. In the graph case, however, this relationship is not nearly as well-defined. It is possible for a graph signal to be fully localized in both the vertex and spectral domains. For example, consider a graph with an isolated vertex. A signal with weight only on the isolated vertex will be an Eigenvector of the graph Laplacian and thus be fully localized in the graph spectrum.

Some work has been done to produce an uncertainty principle on graphs [1], [2], [29]. These works present results on the tradeoffs between spectral vs. spatial localization for graph signals. For our purposes, however, we are interested in the localization of graph convolution kernels. Our approach to analyzing this problem is to consider the quality of an approximation by degree-k polynomials as a measure of approximate k-localization. If the spectrum of a kernel is well-approximated by a polynomial of degree k then we expect the kernel to be approximately k-localized. Such measurements are not relevant for spectral kernels constructed from polynomials as these will have perfect k-localization. Nevertheless, we discuss this approach, illustrate several polynomial approximations and plot the effects of the resulting kernels on a Euclidean grid in order to explore the effects of spectral "smoothness" on the localization of the resulting convolution kernel.

Rather than appealing to analogy with the Euclidean Fourier transform, considering the quality of a low-order polynomial approximation as a measure of spectral kernel smoothness provides an alternative justification for selecting a set of basis functions to construct kernels. Rather than relating boundedness of derivatives to spatial decay, we relate polynomial approximability to localization of the convolution kernel.

Drawing on [18] we note that differences in the spectral coefficients can be used to bound differences in the spatial representations of the kernels (and thus in their responses to signals on the graph). Let g, g' be two convolution kernels defined on a graph G. We note that

$$U^T(Uf \odot Ug) - U^T(Uf \odot Ug') = U^T(Uf \odot Ug - Uf \odot Ug') = U^T(Uf \odot (Ug - Ug')). \tag{3.7}$$

Therefore clearly the deviation in spectral coordinates has significant influence on the spatial response. In fact, we can produce a bound on the "spatial" differences of the convolution kernel

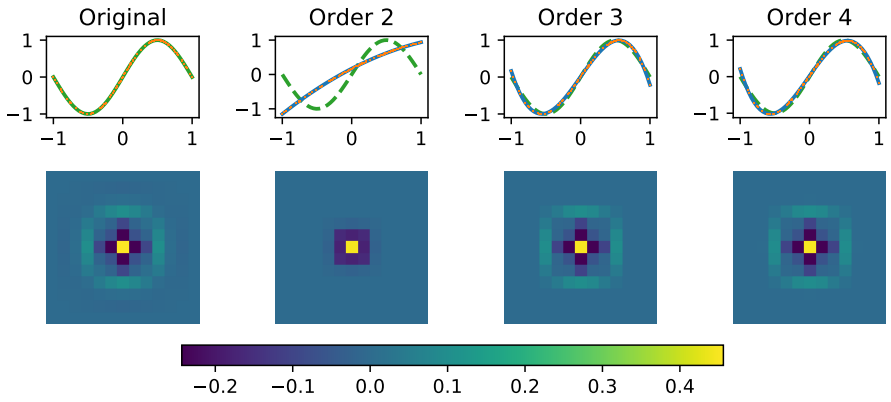

Figure 3.3: The effects of polynomial approximation on a spectral kernel: $\lambda \mapsto \sin(\pi\lambda)$.

from the maximum deviation of the kernel function on the graph spectrum. This is detailed in Theorem 3.3.2.

Theorem 3.3.2 (Spectral Deviations and Spatial Response). *Let g, g' be two graph spectral kernels. If $|g(\lambda_\ell) - g'(\lambda_\ell)| \leq M$ for all λ_ℓ then $|\delta_j^T(\delta_i * g) - \delta_j^T(\delta_i * g')| \leq M$ for all $i, j \in V$.*

Proof. We proceed with the proof given in [18].

$$\begin{aligned}
|\delta_j^T(\delta_i * g) - \delta_j^T(\delta_i * g')| &= |\langle \delta_j,\ (U^T g(\Lambda)U - U^T g'(\Lambda)U)\delta_i \rangle| \\
&= |\langle \delta_j,\ U^T[g(\Lambda) - g'(\Lambda)]U\delta_i \rangle| \\
&= |\langle U\delta_j,\ [g(\Lambda) - g'(\Lambda)]U\delta_i \rangle| \\
&= \left|\sum_\ell v_{\ell,j}[g(\lambda_\ell) - g'(\lambda_\ell)]v_{\ell,i}\right|.
\end{aligned}$$

Here, we have made use of the fact that U preserves inner products as discussed in Theorem 3.2.1.

$$\left|\sum_\ell v_{\ell,j}[g(\lambda_\ell) - g'(\lambda_\ell)]v_{\ell,i}\right| \leq M \cdot \sum_\ell |v_{\ell,j}v_{\ell,i}| \leq M \cdot \left(\sum_\ell |v_{\ell,j}|^2\right)^{1/2}\left(\sum_\ell |v_{\ell,i}|^2\right)^{1/2} = M$$

where we have used the Cauchy-Schwartz inequality and the fact that the v_ℓ form an orthonormal basis. Therefore computing an inner product over their transposes can be at most 1. □

We note that the spatial difference discussed in Theorem 3.3.2, $|\delta_j^T(\delta_i * g) - \delta_j^T(\delta_i * g')|$, is effectively a bound on the deviation in the values of the resulting convolution matrices $U^T g(\Lambda)U$. By varying the values of i and j we select such entries. Therefore the maximum spectral deviation directly gives us a bound on the maximal deviation in the values of the convolution operator on the graph vertices.

One result of this is that the extent to which a function over the eigenvalues of the graph spectrum can be approximated by polynomials can be used as a proxy to measure its localization

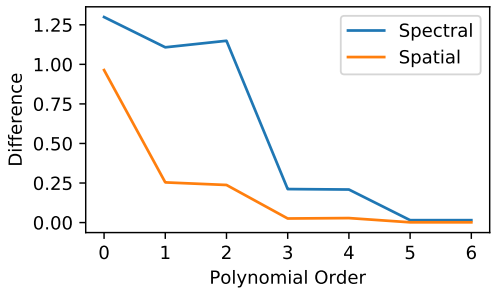

Figure 3.4: Spectral and spatial approximation errors for a kernel of $\lambda \mapsto \sin(\pi\lambda)$.

when used as a convolution kernel. In a sense, the closer the spectral kernels are to a polynomial of degree k, the better they can be approximated by this polynomial in the graph Laplacian which is k-localized. An example of such an approximation is included in Figure 3.3. Visibly, as the order of the polynomial approximation (over a Chebyshev basis) increases, the function of the eigenvalues approaches the "true" values. At the same time the result of a convolution with a delta function on the central vertex also approaches the response of the true kernel. As in previous plots, the graph is a square grid with unit distance connections under the ℓ_∞ norm.

The spectral deviation upper bound is illustrated for the same kernel in Figure 3.4. As discussed in Theorem 3.3.2, the maximal absolute difference over the eigenvalues of the graph spectrum produces an upper bound over the entries in the resulting convolution matrix. The results in this figure show that the $\sin(\pi\lambda)$ kernel is quite well-approximated by a polynomial of degree three. This indicates that the convolution kernel should be largely 3-localized. Indeed, by examining the plots on the Euclidean grid in Figure 3.3, this appears to be the case. The true sine kernel places very low (although nonzero) weight on vertices beyond an ℓ_∞ distance of 3. Its degree three approximation, by contrast, is perfectly 3-localized. The quality of this approximation serves as illustration that this kernel is *almost* 3-localized, and based on Figure 3.4 appears to be 5-localized.

Chapter 4

Applications and Extensions

As discussed in Section 2.3, making use of graph convolutions in a neural network can be accomplished by incorporating a convolution as a network layer:

$$X_{(\ell+1)} = \phi(U^T g_\Theta(\Lambda) U X_{(\ell)}) \tag{4.1}$$

where U is the Graph Fourier Transform matrix (Definition 2.3.1), $g_\Theta(\Lambda)$ is the graph convolution kernel which is learned by gradient descent over its parameters Θ, ϕ is a non-linear function applied over the values of the signal and the X are the inputs and outputs of the layer. Several papers have made use of the techniques discussed above. The exact methods can vary by altering the form of the graph convolutional layer shown in Equation 4.1 or by choosing a particular construction for the convolution kernel g_Θ (a set of basis functions, for example as discussed in Section 3.2).

These techniques can also be applied to several types of machine learning problems. In some cases, as in those discussed in Sections 4.1 and 4.2, the authors validate the architectures on image classification tasks, comparing their results to standard convolutional networks on data sets such as MNIST [24] or ImageNet [30][1]. In others, as considered in Sections 4.3 and 4.4, these techniques are applied to a vertex classification task. In these tasks, each vertex has a label and several associated features. The network is provided with labels for only some vertices and must infer labels for a set of hidden vertices. It is trained to minimize error in these classifications.

In general when compared against traditional convolutional networks on image classification tasks, spectral convolutions tend to achieve results comparable to the standard architectures. On graph-type problems, the use of graph convolutions in a neural network can yield better performance than approaches which do not make use of these operators [22]. Section 4.4 presents an original method of extending graph convolutions to incorporate information from *directed* edges. Below we briefly summarize a number of such applications of graph convolutions in neural networks, and discuss this original extension.

[1] These are image datasets which are commonly used in machine learning tasks. MNIST is a collection of small grayscale images of handwritten digits while ImageNet is a large set of photographs labeled according to their contents.

4.1 Spline Spectral Kernels

In [10] Bruna, Zaremba, Szlam, *et al.* consider methods to construct convolution-type neural networks on graph structured data. The authors propose both a spatial construction and a spectral convolution network architecture. The spatial construction multiplies signals by matrices which are forced to be sparse outside of defined neighborhoods, and the spectral construction is of the basic type discussed in Chapter 2 and makes use of a basis set of cubic splines to construct their convolution kernels. Working by analogy with the spatial localization-spectral smoothness relationship of the Euclidean Fourier transform, the authors select a basis of cubic splines (that is, functions given piecewise by cubic polynomials).

The authors test their architecture on the MNIST data set of handwritten digits. However, to distinguish it from the standard Euclidean case, they produce an irregular graph topology by sampling a fixed set of pixels from the images and linking neighbors in this perforated grid. The spectral construction achieves a 1.8% error rate while using 5×10^3 parameters [10]. Based on the reported results this approach performs less well in terms of accuracy than does their pure spatial construction (a 1.3% error rate) but uses far fewer learned parameters (1.6×10^5). The spline kernel spectral construction achieves a comparable error rate while using several orders of magnitude fewer parameters. The work in [20] applies a network architecture similar to that used in [10] to the ImageNet data set and attains accuracy values within half a percent of a standard convolutional neural network for images. These results for images validate that the graph spectral approach with smooth spectral kernels can achieve similar performance to that of a spatial convolutional neural network.

4.2 Chebyshev Polynomial Kernels

In [13], Defferrard, Bresson, and Vandergheynst make use of the Chebyshev polynomial basis described in Section 3.3.1 and discuss the benefits of polynomial bases in general as was presented in Section 3.3.

The authors construct several different graph spectral networks and test them on several data sets including image (MNIST) and textual data (a set of newsgroup postings). On the MNIST image dataset the authors test both the Chebyshev basis as well as the basis of splines. The authors find little difference between these two bases for graph convolutional networks but do observe a performance difference of 0.2% between the graph convolutions and a standard convolutional network. The authors indicate that they believe this performance loss is due to the lack of orientation in the spectral convolutional kernels, an issue discussed in Section 3.1.1.

4.3 Semi-Supervised Classification

The papers discussed above validate the spectral convolution network architectures on data sets that are often used with more common neural network architectures. The work in [22] designs a

modified graph convolutional neural network and tests it against explicitly graph-type datasets. These include three citation networks, consisting of academic articles classified into subject areas. As described in the introduction to this chapter, the networks are trained to classify vertices based on associated features. The graph convolutional network is trained with known labels on a small fraction of the data set (5%, thus "semi-supervised") and its performance is evaluated based on its output for vertices that were not labeled during training.

Kipf and Welling [22] use a simplified convolution-type operation in their graph convolution layers. Even though this method deviates somewhat from the spectral convolutions discussed in Chapters 2 and 3 it is nevertheless derived from the approach, so we include the results here. The convolution model used by the authors is:

$$X_{(\ell+1)} = \phi(\tilde{D}^{-1/2}\tilde{W}\tilde{D}^{-1/2}X_{(\ell)}\Theta) \tag{4.2}$$

where $\tilde{W} = W + I$ is the sum of the (binary) graph adjacency matrix with the identity matrix to add self-connections to each vertex. Next, $\tilde{D}$ is the associated degree matrix to $\tilde{W}$ just as in the normalized graph Laplacian in Definition 2.2.2; Θ is a matrix of parameters which are to be learned by gradient descent; and $X_{(\ell)}$ is a matrix of features, one for each vertex in the graph, arriving from the previous layer.

This model is a constrained version of the spectral convolution kernels that could be defined using the models described in Sections 4.1 or 4.2. By using $\tilde{W} = W + I$ we are effectively limited to kernels which are 1-localized in the graph. In this model, the authors combine two convolutional layers of the form given by Equation 4.2. The first outputs 16 features, and the second classifies the vertices into the correct number of categories for the dataset by assigning each vertex a point in the appropriate probability simplex.

This model is compared against other approaches which do not make use of graph convolutions and it generally obtains improvements of about 5% in classification accuracy for each data set above these [22]. Their results illustrate the applicability of graph convolutional neural networks in solving classification tasks and their simplified layer model provides an example of a method to dramatically reduce the number of layer parameters if required.

4.4 Directed Graphs

The approach described in this section was developed as a part of research on which the author worked as an assistant to Professor Michael Bronstein, a 2017–2018 visiting fellow at the Radcliffe Institute for Advanced Study at Harvard University [28].

Designing spectral convolutions through the eigenbasis of the graph Laplacian, as described in Definition 2.3.2, makes use of the fact that the Laplacian is symmetric and thus has a full basis of eigenvectors. This translates into a requirement that the graphs used to produce this operator be undirected. In the case of many real-world networks (citation networks, some friendship graphs, etc.), *directed* edges may reasonably be expected to provide additional insight into the graph and the behavior of data defined over it.

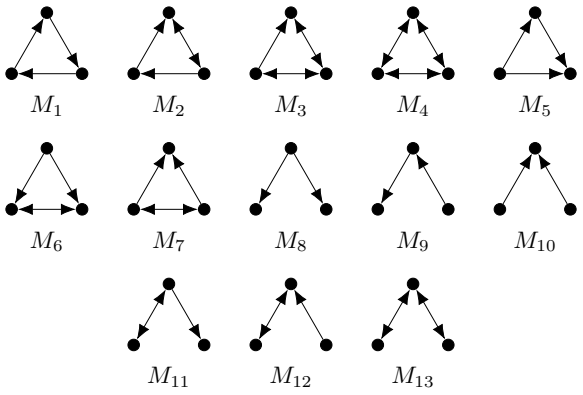

Figure 4.1: The set of three vertex motifs [5], [27].

One approach to lifting this restriction is to build graph neural networks that incorporate multiple generalized Laplacians (generalized Laplacians are discussed briefly in [12]). The approach used here to define such matrices is to modify edge weights according to subgraph structures which appear in the network. As a particular set of such structures, we consider the full set of directed three-vertex subgraphs up to permutations of the vertices, called "motifs" [5], [27]. Figure 4.1 illustrates the full set of these motifs as considered in [28].

Using these motifs, a motif Laplacian can be constructed by counting, for each edge $(i, j) \in E$, the number of times the pair i, j participate in an instance of motif M_i. This produces a symmetric weighted adjacency matrix. This matrix can then be used to construct a normalized Laplacian exactly as discussed in Definition 2.2.2. We have again constructed a symmetric matrix, but here the values still incorporate some notion of the directed structure of the graph. By building a network which incorporates these matrices into its structure, the neural network can be allowed to leverage some amount of the graph's directed information.

Further, these matrices, when viewed over the original graph connectivity and weights, produce convolution filters which are *anisotropic*. Due to the motif-derived weights, the motif Laplacians have "preferred" directions, spreading information more strongly along edges which participate many times in a given motif structure.

To evaluate such an extension, we tested a neural network in the Semi-Supervised Classification setting as presented in Section 4.3 [22]. In our case we made use of a directed version of the CORA citation network data set which includes 19,793 papers, each of which has a vector of 8,710 features and is sorted into one of 70 categories. The number of features was reduced by selecting the first 130 PCA components. The motif Laplacians matrices were constructed following routines derived from the SNAP codebase, a graph processing library from a research group at Stanford [25].

To incorporate these motifs we generalize the polynomial filter basis to include multiple Laplacian matrices. This produces multivariate matrix polynomials. Because these polynomials are over matrices which multiply non-commutatively, the number of possible terms is prohibitively

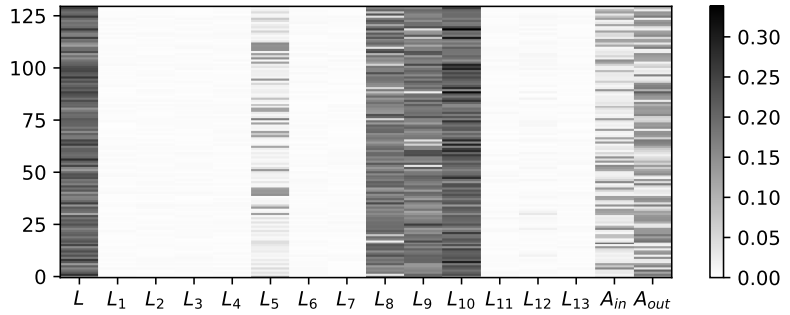

Figure 4.2: Attention values for first layer in attention scoring network [28].

high. Therefore, to reduce the number of terms in the polynomial, and thus parameters in the network, we restrict ourselves to non-mixed terms, that is, polynomials of the form:

$$P_{\Theta}^{D}(L_1, \dots, L_k) = \theta_{0,0} I + \sum_{d=1}^{D} \sum_{\ell=1}^{k} \theta_{d,\ell} L_{\ell}^{d} \tag{4.3}$$

where the $\theta_{d,\ell}$ are the network parameters which are to be trained over the data set.

Still, for practical applications, the full set of thirteen motifs produces a large number of parameters even for polynomial filters of relatively low degree. Therefore, we selected a subset of the relevant motif matrices by first training a network with filters of degree 1 over the full set of possible motif Laplacians and selected those which had the highest "attention score," which measured the contribution of each motif to the determinations made by this network. The result of this scoring from the first layer of the network is illustrated in Figure 4.2 which plots the relative contributions of each matrix across the 130 PCA components.

From this process the matrices which appeared most relevant to the network's operation were L (the undirected Laplacian matrix); the motif Laplacians L_5, L_8 and L_9; and matrices A_{in} and A_{out}. These last two are asymmetric binary adjacency matrices indicating incoming and outgoing edges, respectively. Although these matrices are not symmetric and thus do not produce a graph convolution operator in our usual sense, they nevertheless were relevant in the attention scoring network's operation. The motif Laplacian L_{10}, while relevant, was excluded due to its density, which significantly slowed network training.

With these matrices we constructed a network consisting of two convolutional layers followed by a layer applying a linear operator ("fully-connected"). The network was then trained with 10% of the data set labeled and evaluated on another 10% fraction whose labels were withheld [28]. The performance of the motif-based network was compared to a network producing convolutions with standard Chebyshev polynomials on the undirected graph Laplacian, L. The results of these two networks are plotted in Figure 4.3 and the numerical values are included in Table 4.1.

As is visible in Figure 4.3, incorporating the directed matrices in addition to the standard undirected Laplacian produces a network which consistently outperforms the baseline, even if

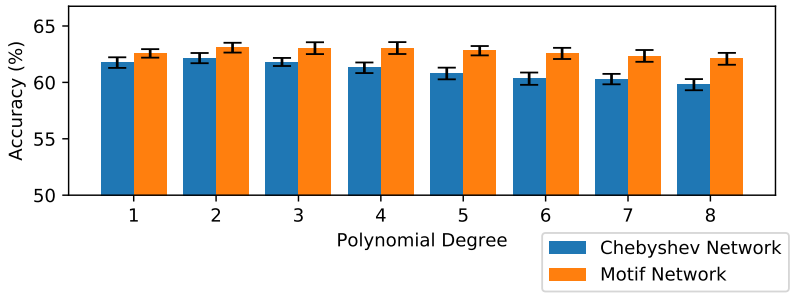

Figure 4.3: Accuracy of motif network vs. undirected Chebyshev baseline [28].

Degree	Chebyshev		Motif	
	Accuracy (%)	Parameters	Accuracy (%)	Parameters
1	61.75	94 k	62.57	95 k
2	62.15	128 k	63.08	131 k
3	61.81	162 k	63.03	166 k
4	61.30	196 k	63.04	202 k
5	60.79	229 k	62.81	237 k
6	60.33	263 k	62.57	272 k
7	60.29	297 k	62.35	308 k
8	59.80	331 k	62.09	343 k

Table 4.1: Motif network performance and trainable parameter counts [28].

only by a small amount. The motif-based network achieves this performance while increasing the number of parameters to train by a small fraction, as indicated in Table 4.1 [28]. These results are preliminary and tested on a single data set. It is possible that on other data sets, where the directed information is yet more important, this approach could yield a larger improvement. Similar experiments on other data remain to be done.

Bibliography

[1] A. Agaskar and Y. M. Lu, "An uncertainty principle for functions defined on graphs", in *Proceedings SPIE Wavelets and Sparsity*, vol. 8138, Sep. 2011. DOI: 10.1117/12.894359.

[2] ——, "A spectral graph uncertainty principle", *IEEE Transactions on Information Theory*, vol. 59, no. 7, Jul. 2013. DOI: 10.1109/TIT.2013.2252233.

[3] M. Artin, *Algebra*, 2nd ed. Prentice Hall, 2011.

[4] S. Axler, *Linear Algebra Done Right*, 2nd ed., ser. Undergraduate Texts in Mathematics. Springer, 1996.

[5] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks", *Science*, vol. 353, no. 6295, Jul. 2016. DOI: 10.1126/science.aad9029.

[6] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, 2nd ed. Dover Publications, 2001.

[7] R. Bracewell, *The Fourier Transform and Its Applications*. McGraw-Hill Book Company, 1965.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning, Going beyond euclidean data", *IEEE Signal Processing Magazine*, vol. 34, no. 4, Jul. 2017. DOI: 10.1109/MSP.2017.2693418.

[9] A. E. Brouwer and W. H. Haemers, *Spectra of Graphs*, ser. Universitext. Springer, 2012.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs", Published on arXiv, May 2014, [Online]. Available: https://arxiv.org/abs/1312.6203.

[11] D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S.-H. Teng, "Spectral sparsification of random-walk matrix polynomials", Published on arXiv, Feb. 2015, [Online]. Available: https://arxiv.org/abs/1502.03496.

[12] F. R. K. Chung, *Spectral Graph Theory*, ser. Regional Conference Series in Mathematics 92. American Mathematical Society, 1997.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering", in *Proceedings of Advances in Neural Information Processing Systems 29 (NIPS)*, 2016. [Online]. Available: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.

[14] A. Deitmar, *A First Course in Harmonic Analysis*, ser. Universitext. Springer, 2002.

[15] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: A multilevel approach", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, Nov. 2007. DOI: 10.1109/TPAMI.2007.1115.

[16] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Johns Hopkins University Press, 1989.

[17] L. J. Grady and J. R. Polimeni, *Discrete Calculus, Applied Analysis on Graphs for Computational Science*. Springer, 2010.

[18] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory", *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, p. 3, 2011. DOI: 10.1016/j.acha.2010.04.005.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification", in *Proceedings of 2015 IEEE Conference on Computer Vision*, Dec. 2015. DOI: 10.1109/ICCV.2015.123.

[20] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data", Published on arXiv, Jun. 2015, [Online]. Available: https://arxiv.org/abs/1506.05163.

[21] Y. Katznelson, *An Introduction to Harmonic Analysis*, 3rd ed. Cambridge University Press, 2004.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Apr. 2017. [Online]. Available: https://arxiv.org/abs/1609.02907.

[23] Y. LeCun, "Deep learning and the future of artificial intelligence", Public Lecture at the Institute for Pure and Applied Mathematics, UCLA, Feb. 2018, [Online]. Available: https://www.ipam.ucla.edu/programs/workshops/new-deep-learning-techniques.

[24] Y. LeCun, C. Cortes, and C. J. C. Burges, *The MNIST database of handwritten digits*, Web. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[25] J. Leskovec and R. Sosič, "SNAP: A general-purpose network analysis and graph-mining library", *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 1, Oct. 2016. DOI: 10.1145/2898361.

[26] U. von Luxburg, "A tutorial on spectral clustering", Max Planck Institute for Biological Cybernetics, Tech. Rep. TR-149, Aug. 2006.

[27] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks", *Science*, vol. 298, no. 5594, Oct. 2002. DOI: 10.1126/science.298.5594.824.

[28] F. Monti, K. Otness, and M. M. Bronstein, "Motifnet: A motif-based graph convolutional network for directed graphs", Published on arXiv, Feb. 2018, [Online]. Available: https://arxiv.org/abs/1802.01572.

[29] N. Perraudin, B. Ricaud, D. I. Shuman, and P. Vandergheynst, "Global and local uncertainty principles for signals on graphs", Published on arXiv, Mar. 2016, [Online]. Available: https://arxiv.org/abs/1603.03030.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge", *International Journal of Computer Vision*, vol. 115, no. 3, 2015. DOI: 10.1007/s11263-015-0816-y.

[31] V. Serov, *Fourier Series, Fourier Transform and Their Applications to Mathematical Physics*, ser. Applied Mathematical Sciences. Springer, 2017, vol. 197.

[32] D. I. Shuman, S. K. Narang, P. Fossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs, Extending high-dimensional data analysis to networks and other irregular domains", *IEEE Signal Processing Magazine*, vol. 30, no. 3, May 2013. DOI: 10.1109/MSP.2012.2235192.

[33] T. Tao, "Fourier transform", Preprint from author's website, 2007, [Online]. Available: http://www.math.ucla.edu/~tao/preprints/fourier.pdf.

[34] L. N. Trefethen, *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.

[35] L. Trevisan, *Cayley graphs of abelian groups*, Online, Berkeley course notes, 2016. [Online]. Available: https://people.eecs.berkeley.edu/~luca/expanders2016/.