# Spectral Graph Theory and Deep Learning on Graphs

Karalias Nikolaos
Supervisor: Tefas Anastasios

Department of Informatics
Aristotle University of Thessaloniki

# Acknowledgements

I would like to thank my supervisor Tefas Anastasios for providing useful advice throughout the course of this year as well as being patient and allowing me the freedom to study the topics that I considered the most interesting in the field.
I would also like to thank my parents for their constant support and understanding.

# Abstract

A significant challenge in machine learning problems is learning meaningful representations that encode all the information that is relevant to a given task. Neural networks focus on learning parameters based on the ability to successfully represent *individual* samples of a dataset. Our goal is to learn representations that combine a sample's individual characteristics and its relationships with other samples by utilizing the concept of the graph convolution. We employ methods and tools from the emerging field of deep learning on graphs to achieve that. Our models implement approximations to the graph spectral definition of the graph convolution, on graphs defined over batches of the dataset. We report better performance than regular convolutional networks on image classification tasks.

# Contents

# Chapter 1

# Introduction

The efficient representation of data has been one of the main objectives of machine learning algorithms throughout the years. For that reason, much of the effort in deploying machine learning algorithms goes into the design of feature extraction, preprocessing and data transformations. Feature engineering is important but can be expensive both in terms of time and effort. In many areas of articial intelligence, information retrieval and data mining, one is often confronted with intrinsically low dimensional data lying in a very high dimensional space. Ideally, one would like to create models that can get a low dimensional representation without the need for manual investigation and time consuming trial and error. In recent years, the explosive growth of deep learning has provided many alternatives for this task of obtaining efficient representations. Deep learning architectures have made machine learning algorithms less dependent on handcrafted features, as the models are capable of learning themselves how to represent the data. The models now incorporate a lot of our prior knowledge and assumptions in their architecture and parameters. Fully automating the process of learning is the ultimate goal in the quest of creating AI that can autonomously model and understand sensory input. Consequently, scientists favor models that reduce the amount of human intervention in the process as much as possible.

## Deep Learning

Deep learning algorithms are special cases of representation learning with the property that they learn multiple levels of representation. Deep learning algorithms often employ shallow (single-layer) representation learning algorithms as subroutines. The central concept behind all deep learning methodology is the automated discovery of abstraction, with the belief that more abstract representations of data such as images, video and audio signals tend to be more useful: they represent the semantic content of the data, divorced from the low-level features of the input. Deep architectures lead to abstract representations because more abstract concepts can often be constructed in terms of less abstract ones [5].

## The notion of spectrum

In Latin, the word spectrum means “image” or “apparition”. It was first introduced by Isaac Newton, referring to the visible spectrum of light. The notion of spectrum is ubiquitous in the fields of physical science and engineering. The electromagnetic spectrum and in particular the visible spectrum, i.e the range of frequencies of electromagnetic radiation that we as humans can perceive, is the occurence of this concept that is probably closest to our everyday experience. On the other hand in a context far removed from traditional human intuition, the spectrum (eigenvalues) of a Hermitian operator associated with an observable is the set of possible outcomes of a quantum measurement. It is clear that the spectrum is a notion deeply woven into different levels of reality. The ability to operate on spectral representations of entities is conceptually appealing with deep mathematical implications that is constantly utilized by scientists. It can often provide insight into problems where the original setup is not easy to understand or interpret. For many years now, popular approaches to common tasks like audio/image compression and filtering have utilized the tools of Fourier analysis to obtain spectral representations of the data. These representations allowed one to effectively manipulate the contents of these signals by working with the intuitive concept of frequencies. Likewise, in this work the concept of spectrum will provide us with the platform to discuss many of the approaches to solving problems in machine learning.

## Data on graphs

Graph-structured data have become ever more prevalent. Some examples are social networks, protein or gene regulation networks, chemical pathways and protein structures, or the growing body of research in program flow analysis. To analyze and understand this data, one needs data analysis and machine learning methods that can handle large-scale graph data sets. Graphs provide a concise way of modeling relationships between objects. In fact, in many applications knowing only the relationships between the entities on a given problem can be more informative about the structure of the data and lead to better solutions than just being aware of the entities themselves. This is one of the main motivations in the learning model we present in this work. Certain types of data, for example images, reside on graphs with highly regular structure such as grids. This allows one to treat the graph as a euclidean domain and use well defined operations. For general graphs that can have arbitrary structure, this is not the case. This means that fundamental operations such as the convolution are not well defined on general graphs. That constitutes an important obstacle in the effort to create a generalized framework, capable of processing arbitrarily structured data. Challenges like this one have motivated researchers from various fields of computer science to formulate graph generalizations of standard operations in an attempt to tackle an ever increasing amount of graph-based problems. Here, once again the notion of spectrum will come into play. Graphs themselves have spectral representations, in terms of the matrices that they are associated with, which can be used to shed more light on the structure of a graph and also lay the foundation for definitions of operations that are hard to establish in the nodal domain.

We will be discussing these ideas in detail and utilizing them in our own methods and experiments. In this work we are going to assume familiarity with some basic knowledge of linear algebra, graph theory and Fourier analysis. We will be providing the relevant context and definitions whenever that is possible and as long as it does not detract from the main points that we are seeking to establish. This work is structured as follows. In the second chapter, we introduce some of the main ideas and applications of spectral graph theory in order to gain an understanding of the interplay between graphs and their spectra. Then, in chapter three we focus on deep learning with an emphasis on graph neural networks. There, the ideas of spectral graph theory appear again as the spectral formulation of graph neural networks was at the core of its rise into popularity within the field.

# Chapter 2

# Spectral Graph Theory

## 2.1 Introduction

Spectral graph theory is the field concerned with the study of the eigenvectors and eigenvalues of the matrices that are naturally associated with graphs. One of the goals is to determine important properties of the graph from its graph spectrum. A lot of invariant properties of the graph that are fundamental in understanding it are closely associated with its spectrum. As we will later explore in more detail, spectral graph theory has interesting links to mathematical areas such as differential geometry and spectral Riemannian geometry.

Apart from theoretical interest it also has a wide range of applications. Some of the earliest applications of spectral graph theory have been in chemistry. Since then, it has been successfully applied to a plethora of problems in biology, communication networks and more recently image processing, computer graphics, and data science. Graph spectra also naturally arise in various problems of theoretical physics and quantum mechanics, such as energy minimization in Hamiltonian systems. We will mostly focus on the data science and signal processing applications and the relevant theory behind them since they are directly linked to the central topic of this work.

The spectrum of a graph $G$ refers to the set of its eigenvalues. The definitions of the graph spectrum are not entirely consistent in the literature as different authors prefer to focus on the eigenvalues of different matrices of the graph. In [15] the spectrum of a graph $G$ is defined as the set of the eigenvalues of its Laplacian matrix, while in [51] and [36] the eigenvalues of the adjacency matrix are used instead. Additionally, many of the methods for clustering and graph drawing that will be discussed later on in this chapter are referred to as spectral and they all rely on the eigenvalues and eigenvectors of different matrices.

Finally, it should be noted that most of the material presented in this chapter is going to address undirected graphs. Undirected graphs are easier to study as they are generally associated with symmetric operators, which in turn are known from the spectral theorem to have real eigenvalues and a set of eigenvectors that form an orthonormal basis. In contrast, the matrices for directed graphs may not necessarily even be diagonalizable. Thus, the theory for the spectra of directed graphs is still being developed while in the undirected case there is already a large body of work that has been completed.

## 2.2 Notation

Scalars $x, y$ are written in lower case font while vectors $\mathbf{x}, \mathbf{y}$ in bold lower case font. Matrices are written as bold uppercase letters $\mathbf{X}, \mathbf{Y}$. The $i$-th element of a vector $\mathbf{x}$ is denoted $x_i$. The element $(i, j)$ of a matrix $\mathbf{A}$ is written as $\mathbf{A}(i, j) = a_{i,j}$. For a generic matrix the eigenvalues will be $\lambda$ and the eigenvectors $\mathbf{v}$. For specific matrices the notation is clarified throughout the chapter. Functions are usually named $f, g$. For graph vertices, we often use the vertex index $i$ in the place of $v_i$ for simplicity. This work covers many subjects in mathematics, from graph theory and discrete quantities to continuous quantities and manifold operators. Thus, the notation can become cumbersome when discussing certain ideas from different angles. Therefore, we did our best to clearly specify all the symbols throughout the work whenever they appear.

## 2.3 Core concepts and properties

### 2.3.1 Matrices of a graph and definitions

An undirected graph $G$ is represented by a pair $(V, E)$ where $V$ is the set of vertices $v_1, v_2, \dots, v_n$ and $E$ the set of edges of the graph. A pair $\{i, j\} \in E$ if vertices $v_i$ and $v_j$ are adjacent. We denote the edge connecting $v_i$ and $v_j$ by $e_{i,j}$. A weighted undirected graph is represented by $(V, E, w)$ where $w : V \times V \to \mathbb{R}$ and satisfies

$$w_{i,j} = w_{j,i} \tag{2.1}$$

and

$$w_{i,j} \geq 0. \tag{2.2}$$

There are various operators that can be associated with a graph. The most commonly known operator of a graph $G$ is the **adjacency matrix**. It is a symmetric matrix defined by

$$\mathbf{A}(i, j) = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

For a weighted graph $(V, E, w)$ the definition is similar, the **weighted adjacency matrix** of the graph

$$\mathbf{W}(i,j) = \begin{cases} w_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

$\mathbf{W}$ reduces to $\mathbf{A}$ in the case where we restrict the weights to be $\{0,1\}$, so from now on we will be defining the rest of the important matrices in the more general weighted terms. The **weighted degree** of a vertex is the total weight of the edges it participates in. The **degree matrix** of a graph is defined as

$$\mathbf{D}(i,j) = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

Thus, the degree matrix is a diagonal matrix with entries the weighted degree $d_i$ of the vertex $i$ defined as

$$d_i = \sum_{j:(i,j)\in E} w_{i,j} \tag{2.6}$$

The **combinatorial degree** counts the number of edges that the vertex participates in. Our definitions moving on will involve the weighted degree unless specified otherwise. An operator derived from the adjacency matrix that encodes the dynamics of a random walk is called the **walk matrix**. It is defined

$$\mathbf{W_{rw}}(i,j) = \begin{cases} \frac{w_{i,j}}{d_i} & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

With these terms defined, we are ready to provide the definition of the **Laplacian** operator of a graph $G$ and its different versions,

$$\mathbf{L}(i,j) = \begin{cases} d_i - w_{i,i} & \text{if } i = j \\ -w_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

The Laplacian in its normalized form is also one of the key operators that we are going to need. It is defined by

$$\mathbf{L_{sym}}(i,j) = \begin{cases} 1 - \frac{w_{i,i}}{d_i} & \text{if } i = j \\ -\frac{w_{i,j}}{\sqrt{d_i d_j}} & \text{if } (i,j) \in E. \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

This normalized version of the Laplacian has been studied extensively and is the preferred version in the standard spectral graph theory textbook by Chung as it is claimed that its eigenvalues relate well to graph invariants and are consistent with eigenvalues in spectral geometry and sthochastic processes. [15]. Finally, we have the **random walk Laplacian**

$$\mathbf{L_{rw}}(i,j) = \begin{cases} 1 - \frac{w_{i,i}}{d_i} & \text{if } i = j \text{ and } d_i \neq 0 \\ -\frac{w_{i,j}}{d_i} & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

These definitions can be expressed in a more compact way using matrix notation. Specifically, we have

- $\mathbf{W_{rw}} = \mathbf{D^{-1}W}$ for the walk matrix.
- $\mathbf{L} = \mathbf{D} - \mathbf{W}$, for the Laplacian.
- $\mathbf{L_{sym}} = \mathbf{D^{-1/2}LD^{-1/2}}$ for the symmetric normalized Laplacian.
- $\mathbf{L_{rw}} = \mathbf{D^{-1}L}$. for the random walk Laplacian.

The random walk Laplacian derives its name from the walk matrix as it can be written

$$\mathbf{L_{rw}} = \mathbf{D^{-1}L} = \mathbf{D^{-}1(D - W)} = \mathbf{I} - \mathbf{D^{-1}W}. \tag{2.11}$$

Similarly, the same type of relationship holds for the normalized Laplacian

$$\mathbf{L_{sym}} = \mathbf{D^{-1/2}LD^{-1/2}} = \mathbf{D^{-1/2}(D - W)D^{-1/2}} = \mathbf{I} - \mathbf{D^{-1/2}WD^{-1/2}} \tag{2.12}$$

Finally, we mention a version of the Laplacian that has not been utilized as much in applications and that is the **signless Laplacian**

$$|\mathbf{L}|(i,j) = \begin{cases} d_i + w_{i,i} & \text{if } i = j \\ w(i,j) & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

which can be more compactly written as

$$|\mathbf{L}| = \mathbf{D} + \mathbf{W}. \tag{2.14}$$

The spectral properties of this matrix have been explored by Cvetkovic et al. [20], however we are not going to focus on the signless Laplacian in this chapter. Before moving forward, it has to be clarified that the results that are going to be presented apply to both the simple and the weighted versions of the matrices. A final matrix that needs to be mentioned is the **incidence matrix B** of a graph. It is an $|V| \times |E|$ matrix

$$\mathbf{B}(v, e_{i,j}) = \begin{cases} \sqrt{w_{i,j}} & \text{if } v = i \\ -\sqrt{w_{i,j}} & \text{if } v = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

Given this definition, the Laplacian matrix $\mathbf{L}$ can also be derived from the incidence matrix as

$$\mathbf{L} = \mathbf{B}\mathbf{B}^{\mathbf{T}}. \tag{2.16}$$

What is important to note in this definition is that the order of the vertices in $e_{i,j}$ is not important as long as one of the vertices participating on the edge is positive and the other negative [15].

Finally, we define some quantities that are commonly used in the literature. The first is the **volume** of a graph $G$, defined by

$$vol(G) = \sum_i d_i \tag{2.17}$$

The **distance** $d_G(i,j)$ between $i, j$ in a graph, counts the number of edges that participate in the shortest path that connects $i$ and $j$. The **diameter** $D$ of a graph is the maximum distance between any two vertices in $G$.

### 2.3.2 Operators on vertices

We provide some motivation behind the need to study graph eigenvalues and eigenvectors. If $\mathbf{A}$ is a square matrix, and $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ its set of eigenvectors that form a basis, we know that

$$\mathbf{A}\mathbf{v}_i = \lambda \mathbf{v}_i. \tag{2.18}$$

Multiplication of a vector $\mathbf{x}$ with the matrix $\mathbf{A}$ can be viewed through the lens of the eigenvectors. The vector $\mathbf{x}$ can be written as

$$\mathbf{x} = \sum_i c_i \mathbf{v}_i, \tag{2.19}$$

and therefore the multiplication with $\mathbf{A}$ is

$$\mathbf{A}^k\mathbf{x} = \sum_i c_i \mathbf{A}^k \mathbf{v}_i = \sum_i c_i.\lambda_i^k \mathbf{v}_i. \tag{2.20}$$

This is important because the operators often associated with undirected graphs are symmetric. Symmetric operators admit an eigenvalue decomposition into a set of orthogonal eigenvectors that form a basis. This is known as the **spectral theorem**. Therefore, matrix multiplications with powers of these operators are easier to study through their spectrum (eigenvalues), since any vector can be expressed in the eigenbasis of the operator.

The vertices of a graph $G$ can be mapped to a vector $\mathbf{x} \in \mathbb{R}^{|V|}$. The vector $\mathbf{x}$ can be viewed as a function $x : V \to R$. In fact, that is exactly the approach in [15], [109]. However we are going to use vector notation to keep it more in line with the rest of the material in this work. That vector can be used to represent entities that reside on the graph's vertices. An intuitive example could be a graph whose vertices contain the pixel intensities of a digital image. The matrix operator associated with the graph $G$ then usually represents some action on the values of the vertices, in this case the pixel intensities. The action of the adjacency matrix on a vertex $v_i$, described by the matrix-vector multiplication $\mathbf{Wx}$, is a sum of the vertex values in the neighborhood of $v_i$. In weighted graphs, each neighboring vertex contribution in the sum is weighted by its corresponding edge weight. That is, if $x_i$ is the real value assigned to the $i$-th vertex of the graph, then the effect of $\mathbf{Wx}$ on that value is

$$x_i = \sum_{j \in N(i)} w_{i,j} x_j \tag{2.21}$$

where $N(i)$ denotes the neighborhood of the vertex $i$, i.e the set of vertices connected to $i$. For the walk matrix, the value at $x_i$ is

$$x_i = \sum_{j \in N(i)} \frac{x_j w_{i,j}}{d_j} \tag{2.22}$$

Recall that a random walk is a process that begins at some vertex, then moves to a random neighbor of that vertex, and then a random neighbor of that vertex, and so on. Random walks arise in many models in mathematics and physics. The theory of random walks is very closely related to a number of other branches of graph theory. Basic properties of a random walk are determined by the spectrum of the graph, and also by electric resistance of the electric networks naturally associated with graphs. There are other processes that can be defined on a graph, usually describing some process of diffusion such as chip-firing, load-balancing in distributed networks etc. whose basic parameters are closely related with random walks [78].

The walk matrix is used to study the evolution of the probability distribution of a random walk. If $\mathbf{x} \in \mathbb{R}^n$ is a probability distribution on the vertices, then $\mathbf{W}_{\mathbf{rw}}^{\mathbf{T}}\mathbf{x}$ is the probability distribution obtained by selecting a vertex according to $\mathbf{x}$, and then selecting a random neighbor of that vertex. As the eigenvalues and eigenvectors of $\mathbf{W}_{\mathbf{rw}}$ provide information about the behavior of a random walk on $G$, they also provide information about the graph [109]. Notice that in terms of matrix operations, we described the action as the matrix-vector multiplication with the tranpose of $\mathbf{W}_{\mathbf{rw}}$ in order to remain consistent with the literature.

For the Laplacian operator, the product $\mathbf{Lx}$ acts on the value at vertex $i$ as follows

$$
\begin{aligned}
x_i &= \sum_{j \in N(i)} x_j l_{i,j} \\
&= \sum_{j \in N(i)} x_j (d_j - w_{i,j})
\end{aligned}
$$

$$
\begin{aligned}
&= (d_i - w_{i,i}) x_i - \sum_{\substack{j \in N(i) \\ i \neq j}} x_j w_{i,j} \\
&= \sum_{j \in N(i)} w_{i,j} x_i - w_{i,i} x_i - \sum_{\substack{j \in N(i) \\ i \neq j}} x_j w_{i,j} \\
&= \sum_{\substack{j \in N(i) \\ i \neq j}} x_i w_{i,j} - \sum_{\substack{j \in N(i) \\ i \neq j}} x_j w_{i,j}
\end{aligned}
$$

Thus, the action is described as

$$x_i = \sum_{j \in N(i)} w_{i,j} (x_i - x_j) \tag{2.23}$$

which matches that of the discrete Laplace operator.

Finally for the incidence matrix the action is $\mathbf{y} = \mathbf{B^T x}$; it maps the values from $\mathbb{R}^V$ to $\mathbb{R}^E$. That is why it is also called the **coboundary operator**. So for the $y_{i,j}$ that corresponds to edge $e_{i,j}$ we have

$$y_{ij} = \sqrt{(w_{i,j})} (x_i - x_j). \tag{2.24}$$

### 2.3.3 Spectrum of a graph and its properties

Having established the basic definitions we can now move on to defining the spectrum of a graph. To avoid confusion between different terminologies, we are going to follow the same convention as [10] and refer to the eigenvalues of the adjacency matrix and the Laplacian matrix as the **adjacency spectrum** and the **Laplacian spectrum** respectively. Before we move on, a few clarifications need to be made regarding the notation that we will be following. For the eigenvalues of the adjacency matrix we have

$$a_1 \geq a_2 \geq \cdots \geq a_n \tag{2.25}$$

and the corresponding eigenvectors

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$$

while for the eigenvalues of the Laplacian

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n. \tag{2.26}$$

and their eigenvectors

$$\mathbf{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_n]$$

For the normalized symmetric Laplacian we have the eigenvalues

$$0 = \nu_1 \leq \nu_2 \leq \cdots \leq \nu_n \leq 2 \tag{2.27}$$

and their eigenvectors

$$\mathbf{\Psi} = [\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_n].$$

For the signless Laplacian $|L|$ the eigenvalues are

$$q_1 \leq q_1 \cdots \leq q_n. \tag{2.28}$$

To avoid more cumbersome notation for the rest of the matrices we will be using $\lambda$ for the eigenvalues and $\mathbf{v}$ for the eigenvectors in the general case of any other matrix. These symbols coincide with the eigenvalues of the Laplacian and eigenvectors of the adjacency matrix but it will be explicitly specified and made clear by the context whenever that is the case.

The definitions of the symmetric normalized Laplacian and the random walk Laplacian hint at relationships between their spectrum and the spectrum of $\mathbf{W_{rw}}$ and $D^{-1/2}WD^{-1/2}$ respectively. Recall that

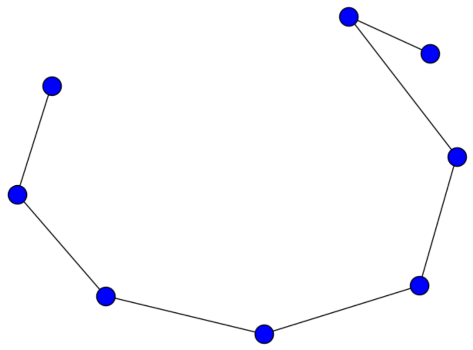
$$\mathbf{L_{rw}} = \mathbf{D^{-1}L} = \mathbf{I} - \mathbf{D^{-1}W}.$$

It is straightforward to demonstrate that $\mathbf{L_{rw}}$ and $\mathbf{W_{rw}}$ share the same eigenvectors, but with eigenvalues $1 - \lambda$.

$$\begin{aligned}
\mathbf{L_{rw}v} &= \lambda\mathbf{v} \\
\mathbf{D^{-1}Lv} &= \lambda\mathbf{v} \\
\mathbf{D^{-1}}(\mathbf{D} - \mathbf{W})\mathbf{v} &= \lambda\mathbf{v} \\
\mathbf{Iv} - \mathbf{D^{-1}Wv} &= \lambda\mathbf{v} \\
\mathbf{Iv} - \lambda\mathbf{v} &= \mathbf{D^{-1}Wv} \\
(1 - \lambda)\mathbf{v} &= \mathbf{D^{-1}Wv}
\end{aligned}$$

The same holds for how $\mathbf{L_{sym}}$ eigenvalues and eigenvectors relate to the eigenvalues and eigenvectors of $\mathbf{D^{-1/2}WD^{-1/2}}$. In a similar way it can also be shown that the adjacency eigenvectors are the same as the Laplacian eigenvectors when a graph is regular. A **regular graph** is a graph where all the vertices have the same degree. There, the Laplacian can be expressed as $\mathbf{L} = k\mathbf{I} - \mathbf{W}$, where $k$ is the degree of every vertex so it should be clear how the same type of proof will work.

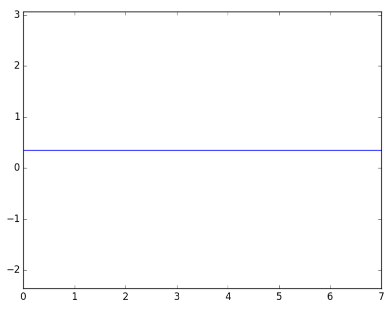
We will now present a few basic facts regarding the spectrum of a graph and how it relates to its structure. The eigenvectors of the Laplacian matrix are also called its **harmonic eigenfunctions** as they can be viewed as maps from the vertices to the eigenvector components. . That is because as we will see soon the eigenvectors have a natural interpretation as the fundamental oscillation modes of a graph. Let us take a look at a simple example.



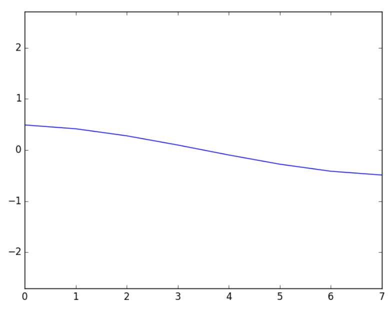
.

Figure 2.1: The $P_8$ path graph.

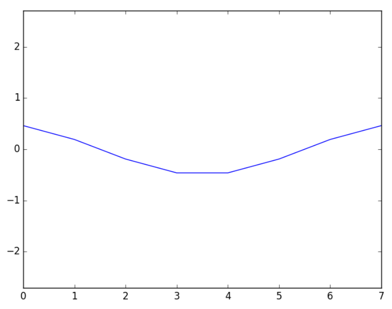
We compute the Laplacian eigenvectors of the graph and plot their values. It is clearly noticeable on the plots that the eigenvectors corresponding to larger eigenvalues represent higher frequency oscillations. The number of zero-crossings increases by one from one eigenvector to the next, starting of course from the all ones vector which is a known eigenvector for every Laplacian matrix and corresponds to the eigenvalue zero. It is easy to prove that $\mathbf{1}$ is an eigenvector of the Laplacian by observing that the rows of the matrix always sum to zero.



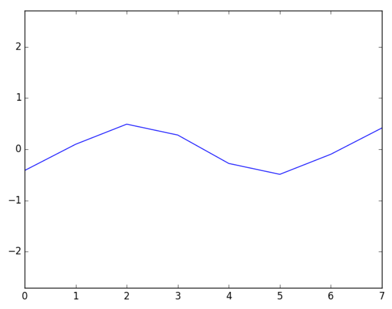
(a) $\boldsymbol{\phi}_1$



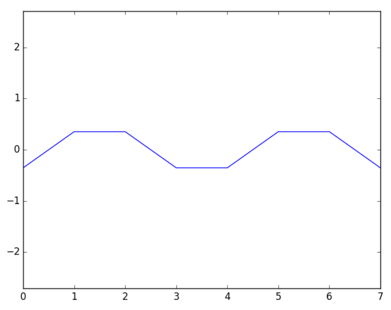
(b) $\boldsymbol{\phi}_2$



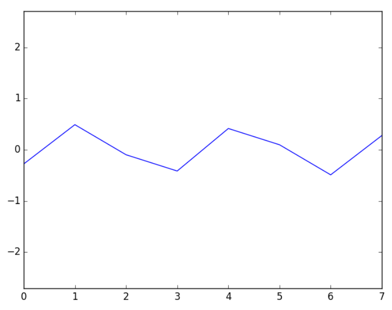
(a) $\boldsymbol{\phi}_3$



(b) $\boldsymbol{\phi}_4$

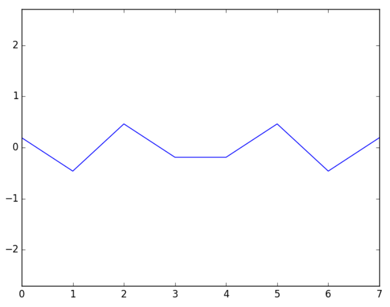


(a) $\boldsymbol{\phi}_5$

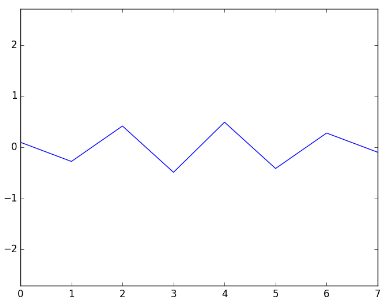


(b) $\boldsymbol{\phi}_6$

Similar patterns on the eigenvectors can be observed for the other matrices as well. It should be pointed out that for the adjacency matrix, since the order of the eigenvalues is reversed, the eigenvectors corresponding to the high frequency oscillations occur first. This intuitive interpretation in terms of oscillations is straightforward in simple structures like path graphs, however this may not be exactly the case for every eigenvector in more complicated graphs. This property of the graph eigenvectors has led many to consider these eigenfunctions as analogues of the Fourier basis for signals defined on graphs. We will present more on that subject later on in the chapter. Also more intuition will be provided into why eigenvectors have this structure, once some important concepts have been introduced. At this point we also have to mention another significant detail. The values of the adjacency and Laplacian matrices depend on the ordering of the vertices. Laplacians with reordered vertices are **permutationally similar**.



(a) $\boldsymbol{\phi}_7$

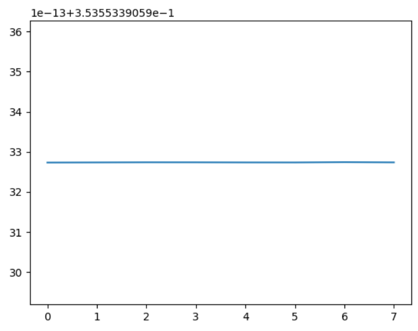


(b) $\boldsymbol{\phi}_8$

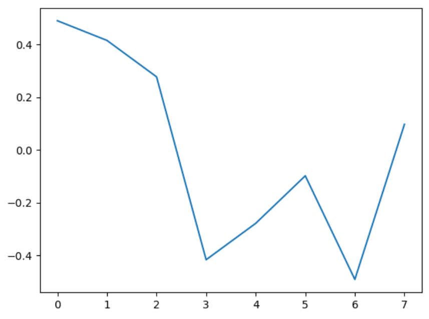
That is, let $\mathbf{L_p}$ be a Laplacian matrix of a graph $G$ whose vertices have been permuted. It can be expressed in terms of the standard Laplacian as follows

$$\mathbf{L_p} = \mathbf{P^T L P} \tag{2.29}$$

where $\mathbf{P}$ is a permutation matrix. Similar matrices have the same eigenvalues and therefore a graph with its vertices permuted will have the same adjacency and Laplacian spectrum as before.

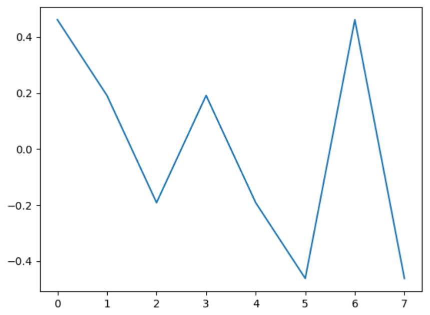


(a) $\boldsymbol{\phi}_1$

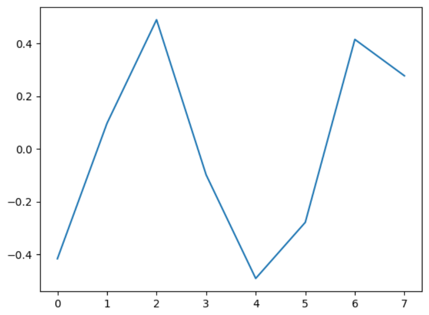


(b) $\boldsymbol{\phi}_2$

This is the first time we encounter the notion of invariance in a more concrete sense. A different way to phrase the similarity of graph permutations is to say that the graph's spectrum is **invariant** to permutations.



(a) $\boldsymbol{\phi}_3$



(b) $\boldsymbol{\phi}_4$

Consider the first 4 eigenvectors of the permuted Laplacian of a path graph. Unlike the previous case, we notice in the plots for the permuted eigenvectors that they do not resemble oscillatory modes any more. For an eigenvector of $\mathbf{L_p}$ we know that

$$\mathbf{L_p}\boldsymbol{\phi} = \lambda\boldsymbol{\phi}$$

but $\mathbf{L_p}$ can be expressed in terms of $\mathbf{L}$. We can write the equation above as

$$\mathbf{P^T L P}\boldsymbol{\phi} = \lambda\boldsymbol{\phi}.$$

A permutation matrix $\mathbf{P}$ is orthogonal , that is

$$\mathbf{P^T P} = \mathbf{I}. \tag{2.30}$$

Thus we get

$$\mathbf{L}(\mathbf{P}\boldsymbol{\phi}) = \lambda(\mathbf{P}\boldsymbol{\phi}) \tag{2.31}$$

This means that an eigenvector of the Laplacian $\mathbf{L}$ is a row-permuted eigenvector of the permuted Laplacian $\mathbf{L}_p$ with the same eigenvalue. For example switching the vertices $i, j$ of a graph implies switching the $i, j$ *rows and columns* of the Laplacian matrix and switching the $i, j$ rows of its corresponding eigenvectors. A graph that is a vertex-permuted version of $G$ is said to be **isomorphic** to $G$.

Next, we list a few more basic facts and definitions about graphs and their spectra [51]. These will not be explored in detail but serve to highlight the relationship between a graph and the spectrum of its associated matrices.

- Two graphs are called **cospectral** if they have the same spectrum. The definition of cospectral usually involves the adjacency spectrum in the literature [36]. Here, we will be explicitly specifying with regard to which spectrum the graphs are cospectral. An example can be seen in Figure 2.8.

- Let $d_{min}$ and $d_{max}$ denote the minimum and maximum degrees of a weighted graph $G$ without isolated vertices, respectively. The following holds

$$\frac{1}{dmax}\lambda_i \leq \nu_i \leq \frac{1}{d_{min}}\lambda_i \tag{2.32}$$

$$\frac{1}{d_{max}}q_{n-i} \leq 2 - \nu_i \leq \frac{1}{d_{min}}q_{n-i} \tag{2.33}$$

$$2d_{min} \leq \lambda_i + q_{n-i} \leq 2d_{max} \tag{2.34}$$

- The multiplicity of $\lambda_1$ counts the number of connected components in a graph.

- A graph $G$ is connected if and only if $\nu_2 > 0$.

- The multiplicity of 2 for the normalized Laplacian eigenvalues counts the number of bipartite connected components that have at least two vertices.

- A weighted graph is bipartite with no isolated vertices if and only if the spectrum of the normalized Laplacian is symmetric around 1.

- For a simple complete graph without isolated vertices, that contains $n \geq 2$ vertices we have

$$\nu_2 = \frac{n}{n-1} \tag{2.35}$$

If the graph is not complete then $\nu_2 \leq 1$.

- If $G$ is a simple graph on $n$ vertices, without isolated vertices, then we have

$$\nu_n \geq \frac{n}{n-1} \tag{2.36}$$

- For a connected weighted graph $G$ on $n$ vertices the following holds

$$\nu_n - \nu_2 \geq \frac{2}{n-1}\sqrt{(n-1)\mathbf{1^T}D^{-1}\mathbf{WD^{-1}1} - n} \tag{2.37}$$

-

$$\sqrt{\frac{\nu_n}{\nu_2}} + \sqrt{\frac{\nu_2}{\nu_n}} \geq 2\sqrt{(1-\frac{1}{n})(1-\frac{1}{n}\mathbf{1^T D^{-1} W D^{-1} 1})} \tag{2.38}$$

- If $D$ is the diameter of a graph $G$ then

$$\nu_2 \geq \frac{1}{Dvol(G)}. \tag{2.39}$$

- The $\mathbf{1}$ vector is an eigenvector of the adjacency matrix of a graph $G$ if and only if the graph is $a_1$-regular.

As it can be seen above, several results have been proven that relate the eigenvalues and eigenvectors of a graph to other quantities that provide information about its structure, such as the volume,the diameter and the vertex degrees of the graph.

The notion of cospectral graphs also implies that not all graphs can be uniquely determined by their spectrum. The question of which graphs can be identified by their spectrum remains still an open problem, even though it was posed more than 50 years ago in [56]. In general, a more modest approach is to focus on which properties of certain families of graphs can be determined by their spectra. Van Dam et al. [117] provide a more detailed review of the problem and its intricacies. Table 1 contains some of the properties of a graph that can be discerned by looking at the spectrum of the different matrices. A 'No' answer indicates the existence of two non-isomorphic graphs which have the same spectrum but differ in the indicated structure.

| **Matrix** | **bipartite** | **#components** | **#bipartite components** | **#edges** |
|---|---|---|---|---|
| Adjacency | Yes | No | No | Yes |
| Laplacian | No | Yes | No | Yes |
| Signless Laplacian | No | No | Yes | Yes |
| Normalized Laplacian | Yes | Yes | Yes | No |

Table 2.1: Table with properties that can be determined from each of the matrices, adapted from [51]

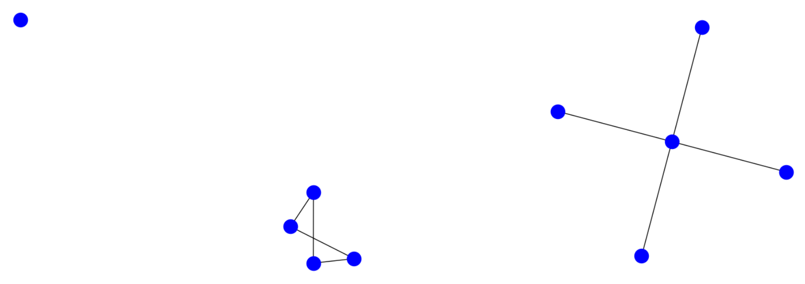


Figure 2.8: Two graphs that are cospectral with respect to the adjacency matrix. The isolated vertex belongs to the graph in the left.

### 2.3.4 Quadratic forms

Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{A}$ a symmetric $n \times n$ matrix. The **quadratic form** is

$$\mathbf{x^T A x}. \tag{2.40}$$

Quadratic forms arise constantly in the study of graph spectra and are also part of many important applications and heuristics. Furthermore, the **Rayleigh quotient** of $\mathbf{A}$ is known to be

$$\frac{\mathbf{x^T A x}}{\mathbf{x^T x}}. \tag{2.41}$$

The **Courant-Fischer theorem** asserts that if $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n$ are $\mathbf{A}$'s eigenvalues, and then

$$\lambda_k = \max_{\substack{S \subseteq \mathbb{R}^n \\ dim(S)=k}} \min_{\substack{\mathbf{x} \in S \\ \mathbf{x} \neq 0}} \frac{\mathbf{x^T A x}}{\mathbf{x^T x}} \tag{2.42}$$

and also

$$\lambda_k = \min_{\substack{T \subseteq \mathbb{R}^n \\ dim(T)=n-k+1}} \max_{\substack{\mathbf{x} \in T \\ \mathbf{x} \neq 0}} \frac{\mathbf{x^T A x}}{\mathbf{x^T x}} \tag{2.43}$$

That is, the Rayleigh quotient is minimized by the set eigenvalues of $\mathbf{A}$, with the $k$-th eigenvalue obtaining the maximum value of the quotient over all subspaces of dimension k. In the second expression for subspaces of dimension $n-k+1$ the $k$-th eigenvalue obtains the minimum value among those maximizing the quotient. From this theorem we can deduce that

$$\lambda_1 = \max_{\mathbf{x} \in \mathbb{R}^n} \frac{\mathbf{x^T A x}}{\mathbf{x^T x}} \tag{2.44}$$

and

$$\lambda_n = \min_{\mathbf{x} \in \mathbb{R}^n} \frac{\mathbf{x^T A x}}{\mathbf{x^T x}} \tag{2.45}$$

This is called the **variational characterization** of the eigenvalues of matrix $\mathbf{A}$. Given these facts, it is also straight forward to deduce that for the values $\lambda$ that optimize the quotient, $\mathbf{x}$ is the corresponding eigenvector.

For the Laplacian matrix $\mathbf{L}$ the quadratic form is expressed as

$$\mathbf{x^T L x} = \sum_{(i,j) \in E} w_{i,j}(x_i - x_j)^2. \tag{2.46}$$

This quadratic form describes the variation of the values on the graph. Its values will be small whenever the values of $\mathbf{x}$ do not vary too much along the edges. An $n \times n$ matrix is positive semidefinite if its quadratic form is nonnegative for all vectors $\mathbf{x} \in \mathbb{R}^n$. An equivalent definition is that all its eigenvalues are nonnegative. For the Laplacian this is already established and therefore we can also say that

$$\mathbf{x^T L x} \geq 0. \tag{2.47}$$

The variational characterization of eigenvalues can help provide additional insight in the case of the Laplacian matrix. Recall how the eigenvectors corresponding to larger eigenvalues had incrementally more zero crossings and matched higher frequency oscillations. It is easily shown that the eigenvector corresponding to the smallest eigenvalue minimizes the Laplacian quadratic form since $\lambda_1 = 0$ for all Laplacians. That is

$$\boldsymbol{\phi_1^T} \mathbf{L} \boldsymbol{\phi} = \boldsymbol{\phi^T} \lambda_1 \boldsymbol{\phi} = 0. \tag{2.48}$$

The all ones eigenvector $\boldsymbol{\phi}_1$ has indeed the maximum 'smoothness' as its values do not vary. This is also reflected in the next eigenvectors which have values that change faster. Remember that $\boldsymbol{\phi}_8$ of the path graph was the highest frequency eigenvector; it maximizes the Laplacian quadratic form for the path graph.

The quadratic forms for the Laplacian and normalized Laplacian are central in understanding the applications of spectral graph theory in problems like graph partitioning, dimensionality reduction and clustering as the eigenvectors and eigenvalues arise naturally as solutions to optimization problems that use these quadratic forms as objectives.

### 2.3.5 Relationships with Manifold Operators

First we establish the relevant definitions relevant to the discussion, as they are presented in [9]. A d-dimensional **manifold** $\chi$ is a topological space where each point $x$ has a neighborhood that is topologically equivalent to a d-dimensional Euclidean space, called the **tangent space** and denoted by $T_x\chi$. The collection of tangent spaces at all points is referred to as the **tangent bundle** and denoted by $T\chi$. On each tangent space an inner product is defined $\langle \cdot, \cdot \rangle_{T_x\chi} : T_x\chi \times T_x\chi \to \mathbb{R}$ which is assumed to depend smoothly on the position of $x$. This inner product is called a **Riemannian metric** in differential geometry. A manifold equipped with a metric is called a **Riemannian manifold**. A Riemannian manifold can be embedded on a Euclidean space by using the structure of the Euclidean space to induce a Riemannian metric. Embeddings are not necessarily unique, two different realizations of a Riemannian metric are called **isometries**. Isometries do not affect the metric structure of the manifold and consequently preserve any quantities that can be expressed in terms of the Riemannian metric. These quantities are called **intrinsic**. Properties related to the specific realization of the manifold in Euclidean space are called **extrinsic**. A **scalar field** is a smooth real function $f : \chi \to \mathbb{R}$ on the manifold. A **tangent vector field** $F : \chi \to T\chi$ is a mapping attaching a tangent vector $F(x) \in T_x\chi$ to each point $x$. Let $L^2(\chi)$ and $L^2(T\chi)$ be the Hilbert spaces of scalar and vector fields on manifolds respectively., with the following inner products:

$$\langle f, g \rangle_{L^2(\chi)} = \int_\chi f(x)g(x)dx$$
$$\langle F, G \rangle_{L^2(T\chi)} = \int_\chi \langle F(x), G(x) \rangle_{T_x\chi} dx$$

where $dx$ denotes a d-dimensional volume element induced by the Riemannian metric. The operator $\nabla f : L^2(\chi) \to L^2(T\chi)$ is called the **intrinsic gradient** and is similar to the classical notion of the gradient defining the direction of the steepest change of the function at a point. The difference is that the direction is a tangent vector. The operator $\text{div} f : L^2(T\chi) \to L^2(\chi)$ acting on tangent vector fields and adjoint to the gradient operator

$$\langle F, \nabla f \rangle_{L^2(T\chi)} = \langle \nabla^* F, f \rangle_{L^2(\chi)} = \langle -\text{div} F, f \rangle_{L^2(\chi)} \tag{2.49}$$

is called the **intrinsic divergence** . The **Laplace-Beltrami operator** $\Delta : L^2(\chi) \to L^2(\chi)$ acting on scalar fields is defined as

$$\Delta f = -\text{div}(\nabla f) \tag{2.50}$$

This operator is self-adjoint(symmetric). Finally we also have,

$$\langle \nabla f, \nabla f \rangle_{L^2(T\chi)} = \langle \Delta f, f \rangle_{L^2(\chi)} = \langle f, \Delta f \rangle_{L^2(\chi)} . \tag{2.51}$$

The left side of this equation is known as the **Dirichlet energy** in physics and measures the smoothness of a scalar field on the manifold.

Now that the definitions are set up, we are going to point out the similarities of these operators with the discrete operators we defined at the start of the chapter. We will be focusing on the unweighted versions (i.e set $w_{i,j} = 1$ in our definitions) since it is easier to draw the parallels. Things become more involved in the weighted case. First we focus our attention on (2.49). Consider the coboundary operator that we defined earlier. Remember that the action $\mathbf{B^T x}$ acts as a difference operator, mapping a function $x : V \to R$ defined on the nodes of a graph, to this function's differences. The operator $\mathbf{B}$ can function as an analogue to the divergence operator above, as it is the transpose (adjoint) of the difference operator. Also notice how in (2.50), the Laplacian operator is defined as the divergence of the gradient. This is reminiscent of the Laplacian matrix property $\mathbf{Lx} = \mathbf{BB^T x}$ where $\mathbf{B^T}$ behaves like the gradient, $\mathbf{B}$ like the divergence operator and of course $\mathbf{L}$ is the analogue to the Laplace-Beltrami operator. In (2.51) we see that the Dirichlet energy is the inner product of the gradient with itself. We form the same inner product with the incidence matrix,

$$||\mathbf{B^T x}||^2 = (\mathbf{B^T x})^\mathbf{T}(\mathbf{B^T x}) = \mathbf{x^T B B^T x} = \mathbf{x^T L x} \tag{2.52}$$

which gives us the Laplacian quadratic form that is also a measure of smoothness on the graph. The eigenfunctions of the Laplace-Beltrami operator are the smoothest functions in the sense of Dirichlet energy, as are the eigenvectors of the Laplacian matrix in terms of its quadratic form. We can clearly recognize that the Laplacian quadratic form works on a graph as a discrete analogue of the Dirichlet energy.

In applications, there are certain properties that are desired for the discrete Laplacians so that they resemble the continuous operator as much as possible. For a smooth surface $S$ , Wardetzky et al. [122] describe the core properties of the smooth Laplace-Beltrami operator which we reproduce here:

- $\Delta f = 0$ whenever $f$ is constant.
- Symmetry: $\langle \Delta f, g \rangle = \langle f, \Delta g \rangle$ whenever $f, g$ are sufficiently smooth and vanish along the boundary of $S$
- Local support: for any pair $p \neq q$ of points, $\Delta f(p)$ is independent of $f(q)$. Altering the function value at a distant point will not affect the action of the Laplacian locally.
- Linear precision: $\Delta f = 0$ whenever $S$ is a part of the Euclidean plane, and $f = ax + by + c$ is a linear function on the plane
- Maximum principle: harmonic functions (those for which $\Delta f = 0$ in the interior of $S$) have no local maxima (or minima) at interior points.
- Positive semi-definiteness: The Dirichlet energy is nonnegative. $\langle f, \Delta f \rangle_{L^2(S)} \geq 0$ whenever $f$ is sufficiently smooth and vanishes along the boundary of $S$.

On a triangular surface mesh $M$, the set of desired properties for the discrete Laplacian is:

- Symmetry: $w_{i,j} = w_{j,i}$ Motivation: Real symmetric matrices have real eigenvalues and orthogonal eigenvectors.
- Locality: Weights are associated to mesh edges (1-ring support), so that $w_{i,j} = 0$ if $(i, j) \notin E_M$. Changing the function value $f_j$ will not alter the Laplacian's action on vertex $i$, $Lf_i$, if $i, j \notin E_M$. Motivation: Smooth Laplacians govern diffusion processes via $f_t = -\Delta f$. When discretized via random walks on graphs, $w_{i,j}$ are transition probabilities along the edges of $M$.

- Linear precision: $Lf_i = 0$ at each interior vertex whenever $M$ is a straight line embedded into the plane and $f$ is a linear function on the plane, point sampled at the vertices of $M$. This is equivalent to requiring that

  $$\sum_j w_{i,j}(x_i - x_j) = 0. \tag{2.53}$$

  for all interior vertices $i$, where $\mathbf{x} \in \mathbb{R}^{2|V|}$ denotes the vector of positions of the vertices in the plane. This is desirable in many graphics applications for purposes such as denoising without vertex drift etc.

- Positive weights $w_{i,j} \geq 0$ whenever $i \neq j$. Additionally, it is required that for each vertex $i$ there is at least one vertex $j$ such that $w_{i,j} > 0$. Motivation: Positive weights are a sufficient condition for a discrete maximum principle like the one in the continuous case we mentioned above. They assure that flow travels from regions of higher to regions of lower potential. It also establishes barycentric coordinates by setting

  $$y_{i,j} = \frac{w_{i,j}}{\sum_{j \neq i} w_{i,j}} \quad \text{so that} \quad \sum_{j \neq i} y_{i,j} = 1.$$

  That means $f$ is a discrete harmonic ($Lf_i = 0$ at all interior vertices ) if and only if $f_i$ is a convex combination of its neighbors.

- Positive semi-definiteness: $L$ is symmetric positive semidefinite with respect to the standard inner product and has a one-dimensional kernel. This is to keep it in line with the continuous analogue of having a nonnegative Dirichlet energy.

- Convergence $L_n \to \Delta_n$ in the sense that solutions to the discrete Dirichlet problem, involving $L_n$, converge to the solution of the smooth Dirichlet problem, involving $\Delta$ under appropriate refinement conditions and in appropriate norms. The convergence of discrete Laplacians to their manifold operators has been studied in [108] and [45]. Under certain assumptions, it is possible for the discrete operators to converge to their continuous counterparts.

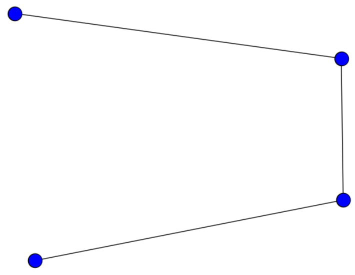
The main result in [122] is that not all of these natural properties can be satisfied simultaneously. Thus, a 'perfect' discrete Laplacian does not exist.

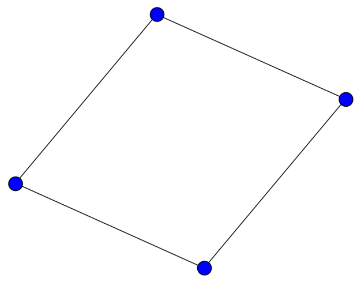
## 2.4 Graph Products

We will introduce three fundamental graph products, the Cartesian graph product, the direct product, and the strong product. The **Cartesian product** of graphs

$G$ and $H$, denoted by $G \diamond H$ is a graph whose vertex set is $V_G \times V_H$ of sets. Two vertices $vh$ and $v'h'$ are adjacent precisely if $v = v'$ and $(h, h') \in E_H$, or $(v, v') \in E_G$ and $h = h'$. Thus,

$$V_{G\diamond H} = \{vh : v \in V_G \text{ and } h \in V_H\} \tag{2.54}$$
$$E_{G\diamond H} = \{(vh, v'h') : v = v', (h, h') \in E_H, \text{ or } (v, v') \in E_G, h = h'\} \tag{2.55}$$


(a) Path graph $P_4$


(b) Cycle graph $C_4$

The graphs $G$ and $H$ are called factors of the product $G \diamond H$. An example of this product is demonstrated in Figure 2.10. The **direct product** of $G$ and $H$, denoted by $G \times H$ is a graph product whose vertex set is $V_G \times V_H$ and for which vertices $vh$ and $v'h'$ are adjacent precisely if $(v, v') \in E_G$ and $(h, h') \in E_H$. Thus,

$$V_{G\times H} = \{vh : v \in V_G \text{ and } h \in V_H\} \tag{2.56}$$
$$E_{G\times H} = \{(vh, v'h') : (v, v') \in E_G \text{ and } (h, h') \in E_H\} \tag{2.57}$$

An example of the direct product is presented in Figure 2.11. The direct product has appeared in the literature with varues names such as tensor product, Kronecker product, relational product, categorical product etc.

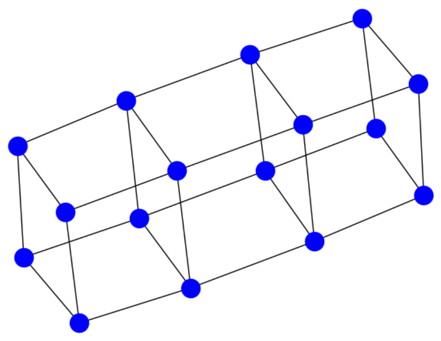

.

Figure 2.10: The $P_4 \diamond C_4$.

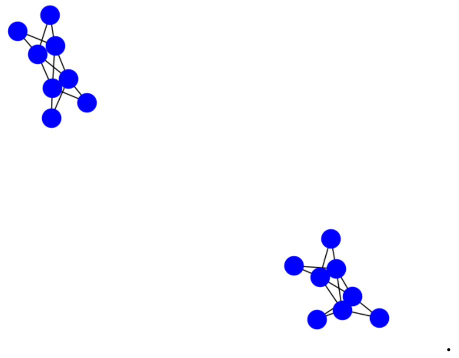


Figure 2.11: The $P_4 \times C_4$.

Finally, the **strong product** of $G$ and $H$ is the graph product denoted by $G \boxtimes H$ and defined as

$$V_{G\boxtimes H} = \{vh : v \in V_G \text{ and } h \in V_H\} \tag{2.58}$$
$$E_{G\boxtimes H} = E_{G\diamond H} \cup E_{G\times H} \tag{2.59}$$

Other names that have been used for this are strong direct product or symmetric composition. Figure 2.12 shows an example of the strong product.

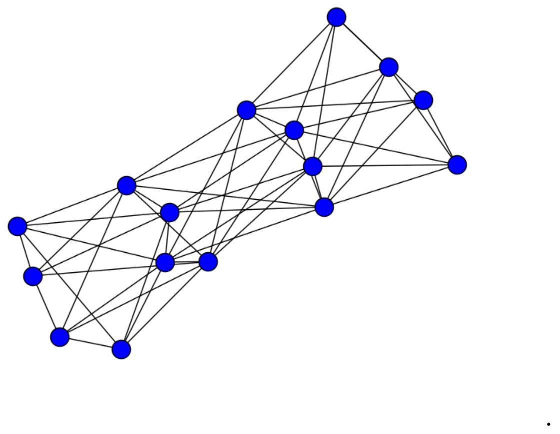


Figure 2.12: The $P_4 \boxtimes C_4$.

The adjacency matrix of product graphs can be described in terms of the adjacency matrices of the product factors. Let $\mathbf{I}_n$ be the $n \times n$ identity matrix. The adjacency matrices of the graph products can be written as

$$\mathbf{A}_{G\diamond H} = \mathbf{A}_G \otimes \mathbf{I}_{|V_H|} + \mathbf{I}_{|V_G|} \otimes \mathbf{A}_H \tag{2.60}$$
$$\mathbf{A}_{G\boxtimes H} = \mathbf{A}_G \otimes \mathbf{I}_{|V_H|} + \mathbf{I}_{|V_G|} \otimes \mathbf{A}_H + \mathbf{A}_G \otimes \mathbf{A}_H \tag{2.61}$$
$$\mathbf{A}_{G\times H} = \mathbf{A}_G \otimes \mathbf{A}_H \tag{2.62}$$

These relationships allow us to deduce the adjacency spectra of these graph products. Let $a^G$ and $a^H$ be the adjacency eigenvalues of graphs $G$ and $H$ respectively, with corresponding eigenvectors $\mathbf{v}^G$ and $\mathbf{v}^H$. For all three products it can be shown that their eigenvectors $\mathbf{v}^{GH}$ are

$$\mathbf{v}^{GH} = \mathbf{v}^G \otimes \mathbf{v}^H. \tag{2.63}$$

Now, let $i = 1, 2 \ldots, |V_G|,\ j = 1, 2, \ldots, |V_H|$,for the Cartesian product we have

$$\mathbf{A}_{G\diamond H}(\mathbf{v}_i^G \otimes \mathbf{v}_j^H) = (a_i^G + a_j^H)(\mathbf{v}_i^G \otimes \mathbf{v}_j^H). \tag{2.64}$$

For the direct product

$$\mathbf{A}_{G\times H}(\mathbf{v}_i^G \otimes \mathbf{v}_j^H) = (a_i^G a_j^H)(\mathbf{v}_i^G \otimes \mathbf{v}_j^H) \tag{2.65}$$

and finally for the strong product

$$\mathbf{A}_{G\boxtimes H}(\mathbf{v}_i^G \otimes \mathbf{v}_j^H) = (a_i^G + a_j^G + a_i^G a_j^H)(\mathbf{v}_i^G \otimes \mathbf{v}_j^H) \tag{2.66}$$

For the Laplacian spectrum only the Cartesian product has been analytically described so far. The Laplacian of the Cartesian product is written

$$\mathbf{L}_{G\diamond H} = \mathbf{L}_G \otimes \mathbf{I}_{|V_H|} + \mathbf{I}_{|V_G|} \otimes \mathbf{L}_H \tag{2.67}$$

and thus its spectrum can be written as

$$\mathbf{L}_{G\diamond H}(\boldsymbol{\phi}_i^G \otimes \boldsymbol{\phi}_j^H) = (\lambda_i^G + \lambda_j^H)(\boldsymbol{\phi}_i^G \otimes \boldsymbol{\phi}_j^H) \tag{2.68}$$

The Laplacian spectrum of the direct and the strong product have not been analytically expressed yet although there are some heuristic approximations that seem to produce good enough results in some cases [97].

## 2.5 Spectral Graph Drawing

An interesting problem is that of creating a visual representation of a graph's vertices and their connectivity. The earliest approach was formulated by Tutte in [116], where each node was drawn as a weighted centroid of its neighbors. The boundary was set by some nodes whose coordinates were manually selected and served as anchor points in order to place the rest of the nodes. The selection of those anchor points was arbitrary which was the main problem with this approach.

Here we will consider a simple spectral approach to the problem of drawing a graph. We are concerned with visualizing a graph using only its connectivity matrix. Initially, the problem will come down to mapping the graph vertices to a vector of real numbers. After that is established, the approach will be extended to produce a pair of coordinates for each vertex. Now let $\mathbf{x} \in \mathbb{R}^{|V|}$ be the vector of real numbers that the graph vertices are mapped on. A desired property of our mapping is that neighboring vertices should be close to each other on the real line. At this point, recall that the Laplacian quadratic form

$$\mathbf{x}^{\mathbf{T}}\mathbf{L}\mathbf{x} = \sum_{(i,j)\in E} w_{i,j}(\mathbf{x}_i - \mathbf{x}_j)^2 \tag{2.69}$$

provides a measure of smoothness on the graph. Specifically, larger differences between values of neighboring vertices mean higher values of the Laplacian quadratic form. Therefore, it was suggested by Hall in [41], that the assignment of real values to the vertices should minimize the Laplacian quadratic form. To avoid degenerate solutions like the all zeros vector, an additional constraint can be imposed on the euclidean norm of $\mathbf{x}$ so that

$$\mathbf{x}^{\mathbf{T}}\mathbf{x} = ||\mathbf{x}||^2 = 1. \tag{2.70}$$

However a degenerate solution is still possible, since all the vertices can be mapped to $1/\sqrt{n}$. Therefore we also require

$$\mathbf{1}^{\mathbf{T}}\mathbf{x} = 0. \tag{2.71}$$

From the Courant-Fischer Theorem , the solution satisfying (2.70) that minimizes (2.69) is the eigenvector corresponding to the smallest eigenvalue. For the Laplacian matrix, that eigenvector is the all ones vector with eigenvalue zero which is not allowed by our constraint (2.71) and thus we are led to choose the eigenvector corresponding to the second smallest eigenvalue of the Laplacian.

For a visual representation of the graph we are going to need a pair of coordinates. Having successfully mapped the vertices on the real line, the 2D case requires a slight modification. Based on (2.69) we can minimize the equivalent expression for the 2D case

$$\sum_{(i,j)\in E} ((\mathbf{x}(i) - \mathbf{x}(j))^2 + (\mathbf{y}(i) - \mathbf{y}(j))^2)$$

which is just

$$\mathbf{x^T L x} + \mathbf{y^T L y}.$$

Constraints (2.70) and (2.71) apply to $\mathbf{y}$ as well.

$$\mathbf{y^T y} = ||\mathbf{y}||^{\mathbf{2}} = 1 \tag{2.72}$$

$$\mathbf{1^T y} = 0. \tag{2.73}$$

A remaining degenerate solution is $\mathbf{x} = \mathbf{y} = \boldsymbol{\phi}_2$. To avoid this one, the constraint imposed is

$$\mathbf{x^T y} = 0. \tag{2.74}$$

i.e requiring that the two vectors are orthogonal. This finally leads us to

$$\mathbf{x} = \boldsymbol{\phi}_2 \tag{2.75}$$

$$\mathbf{y} = \boldsymbol{\phi}_3. \tag{2.76}$$

Thus, the eigenvectors corresponding to the second and third smallest eigenvalues provide us with pairs of coordinates for each node. Instead of the Laplacian, approaches using the eigenvectors of the adjacency matrix have also been formulated. In fact, Koren in [65] suggests using the degree-normalized eigenvectors, i.e the eigenvectors of the walk matrix which is $\mathbf{D^{-1}A}$. An interesting consequence of this approach is that it produces the non-degenerate drawing with the smallest deviations from the centroid of neighbors and therefore addresses the problem with Tutte's centroids that we mentioned in the beginning of the section.

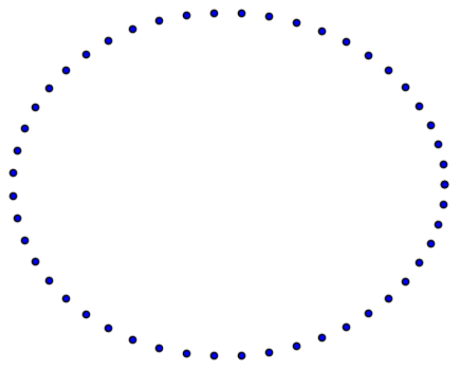
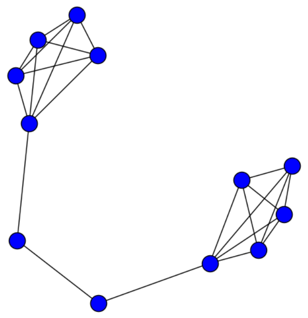


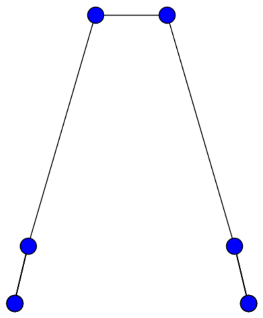
Figure 2.13: Plot of points with $\mathbf{x} = \boldsymbol{\phi_1}$ and $\mathbf{y} = \boldsymbol{\phi_2}$

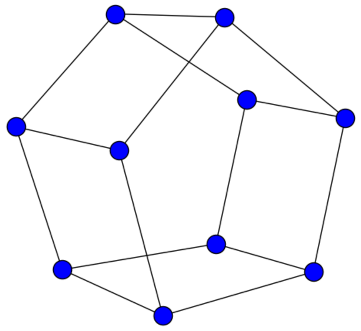
Moving on, we will be using the term "spectral layout" interchangeably with "spectral drawing". Next, we will take a look at a few examples of spectral drawings with the eigenvectors of the Laplacian. First we produce a set of points uniformly on the unit circle. Then we form their weighted adjacency matrix $\mathbf{W}$ whose elements are

$$w_{i,j} = e^{-\frac{|x_i - x_j|^2}{1.45}} \tag{2.77}$$

and the respective Laplacian matrix from $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Plotting with the eigenvectors corresponding to the second and third smallest eigenvalue accurately produces a ring as shown in Figure 2.13.


(a) Barbell graph


(b) Spectral drawing of barbell graph


(a) Circular ladder graph

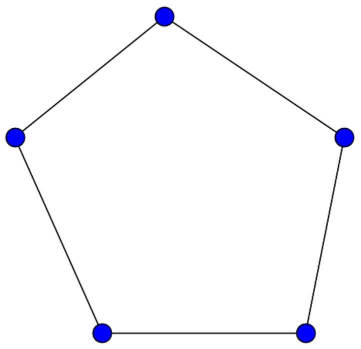

(b) Spectral drawing of circular ladder graph

Figure 2.15: Cases of node overlap in the spectral layout

In the first one we have the barbell graph followed by its spectral drawing. It consists of two complete graphs on 5 nodes and a 2 node path graph connecting them. On the second one we have the circular ladder graph. This one consists of 5 connected pairs forming a ring and each pair connects with two other pairs. Every vertex has degree 3 which means this is a regular graph. In these examples, only half of the nodes are visible in the spectral drawing due to overlap. While this can definitely be a drawback when drawing graphs, the tendency of the mapping to cluster points together can be utilized to our advantage. This will be the focus of the next section. Spectral graph embeddings can provide an intuitive perspective on a graph, especially when the only information available is the graph's connectivity matrix. However, as we saw in the examples above, the quality of the spectral embedding can vary and for many graphs the drawing may not give a clear picture of the graph's nodes. For a more detailed discussion on spectral graph drawing we recommend Koren's paper [65]. Also, for a general overview of various graph drawing methods see the survey in [47].

## 2.6 Spectral Clustering

The second smallest eigenvalue of the Laplacian matrix has been itself the focus of extensive study. The eigenvalue $\lambda_2$ is zero if and only if a graph is not connected. The better a graph is connected, the higher the value of $\lambda_2$ is going to be. This is why Fiedler in his 1973 paper [25] called $\lambda_2$ the **algebraic connectivity** of a graph. The corresponding eigenvector is often referred to as the **Fiedler vector** or the **characteristic valuation** of the graph.

Spectral clustering includes all methods and techniques that partition a set of $n$ objects into clusters by using the eigenvectors of the connectivity matrices that describe the relationships between these $n$ objects. It has been effectively implemented in various problems including community detection in graphs [27], image segmentation [104], fMRI analysis [19] and 3D mesh segmentation [77].

In the previous section on spectral graph drawing, we demonstrated how the eigenvector corresponding to $\lambda_2$ provides a mapping of vertices to real numbers that minimizes the Laplacian quadratic form $\mathbf{x^T L x}$. We can further use this mapping to partition a graph. This is achieved by choosing a real number $t$ and partitioning the nodes depending on whether they satisfy $\phi_v > t$ or not. Fiedler proved in [26] that for all $t \leq 0$ the set of nodes that obey $\phi_v \geq t$ form a connected component. As an example, the graph in Figure 2.16 has $\lambda_2 = 0.432$. and was generated with the Holme and Kim algorithm [52] that produces graphs with power law degree distribution and approximate clustering.

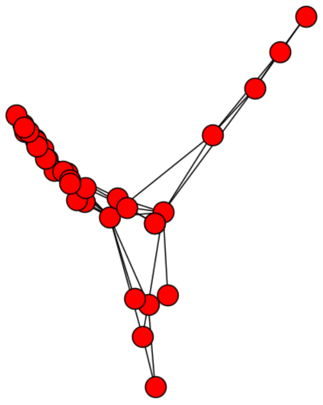
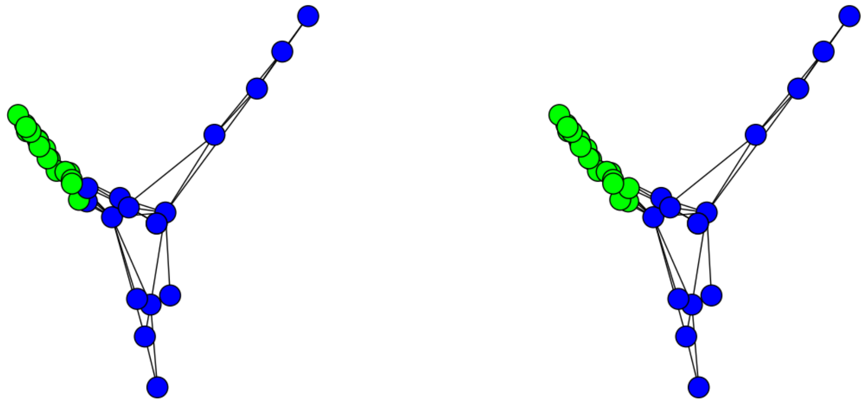


Figure 2.16: Spectral layout of power law graph on 34 nodes with p=0.63 and m=2



(a) Partitioned for $t = \tilde{\mu}$ (b) Partitioned for $t = 0$

Figure 2.17: Partitioned power law graph using the Fiedler vector

In figure 2.17 we can see the result of the partitioning on a spectral layout for two different values of $t$. Common choices are $t = 0$ and $t = \tilde{\mu}$ where $\tilde{\mu}$ represents the median element of $\boldsymbol{\phi}_2$. The graph in Figure 2.18 was drawn with the force directed placement algorithm by Fruchterman and Reingold [29]. This illustrates that the elements of the Fiedler eigenvector can indeed provide a partitioning of the vertices.

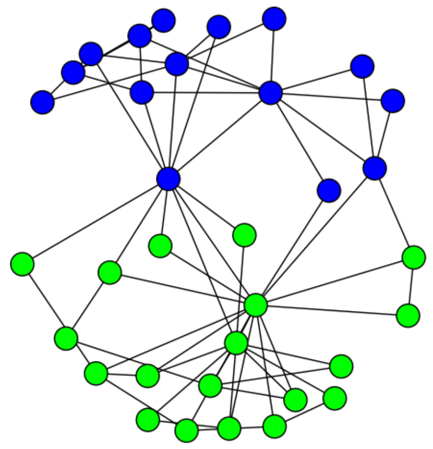


Figure 2.18: Partitioned power law graph drawn with a spring layout with $t = 0$

This spectral partitioning method is heuristic and does not guarantee an optimum result. Graph bisection is in fact an NP-complete problem. A number of variations can be considered on these heuristics, and all of them are concerned with choosing an appropriate value of $t$ for the bisection. Apart from the choices of $t = \tilde{\mu}$ and $t = 0$ that were already presented there are a couple of alternatives that were presented in [40]:

- Look for a large gap in the sorted list of the Fiedler vector components and set $t$ such that the vertices are partitioned depending on whether they are above or below the gap
- On the sorted list of the Fielder vector components, consider setting $t = i$ where $i$ is the index that provides the best ratio cut.

The ratio cut is a quantity used to evaluate the quality of a cut. Let $S$ be a part of the set of all vertices $V$, i.e $S \subset V$ . Let $\partial S$ be the set of edges with exactly one endpoint in $S$. $\partial S$ is called the **edge boundary** of $S$ :

$$\partial S = \{\{u, v\} \in E(G) : u \in S \text{ and } v \notin S\} \tag{2.78}$$

The **vertex boundary** of $S$ is similarly defined as the set of vertices in $S$ that have an edge connecting them to vertices in $\tilde{S}$ with $\tilde{S} = V - S$. A well known measure that evaluates the quality of a cut is the ratio cut. The **ratio cut** is defined as

$$R(S) = \frac{w(\partial S)}{|S||\tilde{S}|} \tag{2.79}$$

where $w(\partial(S))$ denote the sum of the weights of the edges in $\partial S$. Further heuristics can be implemented that depend on the ratio cut. For example, one could experiment with other eigenvectors instead of the Fiedler vector for the graph partitioning, and select the one that gives the best values for $R(S)$. [39] Next, define the **conductance** [57] of a cut as

$$h_G(S) = \frac{w(\partial S)}{\min(vol(S), vol(\tilde{S}))} \tag{2.80}$$

The conductance of a full graph, denoted by $h_G$ is the minimum conductance over all the cuts of the graph, i.e

$$h_G = \min_{S \subset V} h_G(S) \tag{2.81}$$

The conductance of the graph, also known as the **Cheeger constant**, provides a measure of how "well-knit" the graph is. The conductance of a clustering is the minimum conductance of its clusters. The famous **Cheeger inequality** relates the conductance of a graph with the eigenvalues of the normalized Laplacian,

$$\nu_2 > \frac{h_G^2}{2}. \tag{2.82}$$

There are variants of this inequality that can be proved that use the walk matrix instead of the normalized Laplacian, or that relate the eigenvalues of the unnormalized Laplacian to quantities like the isoperimetric ratio [14].

The methodology of partitioning a graph using its eigenvector components can be used to obtain a multi-way partitioning; a clustering of the vertices of the graph into more than two groups. After obtaining two clusters from the initial partitioning, the same methodology can be recursively applied to each cluster to further partition them into more clusters. A stopping criterion for the recursion recommended by Shi and Malik was a quantity similar to the conductance, which they defined as the **normalized cut**

$$Ncut(S, \tilde{S}) = \frac{cut(S)}{assoc(S, V)} + \frac{cut(S)}{assoc(\tilde{S}, V)} \tag{2.83}$$

, with the *cut* and *assoc* quantities defined as

$$cut(S) = \sum_{(u,v) \in \partial S} w_{u,v} \tag{2.84}$$

$$assoc(S, V) = \sum_{u \in S,\ v \in V} w_{u,v}. \tag{2.85}$$

The quantity $cut(S)$ is the sum of edge weights in the set $\partial S$. Therefore

$$cut(S) = w(\partial S), \tag{2.86}$$

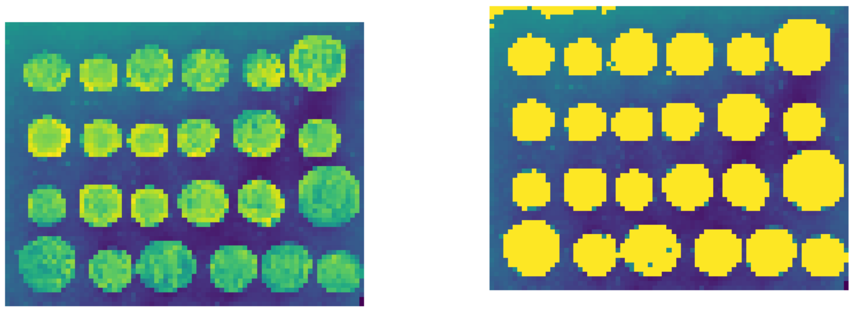
while $assoc(S, V)$ represents the total connections from nodes in $S$ to all vertices $V$ in the graph, i.e

$$assoc(S, V) = vol(S). \tag{2.87}$$

A simple recursive algorithm for clustering can be summarized as follows [104]:

1. Given a graph $G = (V, E)$, set up the Laplacian matrix $\mathbf{L}$ and the degree matrix $\mathbf{D}$
2. Solve the eigenvalue problem of $\mathbf{D^{-1}Lv} = \lambda\mathbf{v}$
3. Use the eigenvector corresponding to the second smallest eigenvalue to bipartition the graph, deciding the partition point $t$ based on where the $Ncut(S, \tilde{S})$ is minimized.
4. Decide if the partition should be further subdivided. Halt if the $Ncut$ value is above a certain threshold.
5. Recursively bipartition the segmented parts.

In their paper, Shia et al. also ensure the stability of the cut by monitoring the histogram of the eigenvector's components and ignoring eigenvectors with smoothly varying values.



(a) Original image

(b) Image segmented in two groups, one is highlighted.

Figure 2.19: Image segmentation with the normalized spectral clustering algorithm.

There are certain disadvantages that come with this recursive approach, such as the eigenvalue problem that is being solved, even though only the second eigenvector is used. It would be desirable to utilize more of the eigenvectors and simultaneously form the required clusters. Recall that for spectral graph drawing, we utilized the values of the third eigenvector to map the vertices of the graph on two dimensions and how that could be extended to more dimensions. In [86] Ng et al. propose the following algorithm for simultaneous spectral clustering:

1. Form the Weighted adjacency and Degree matrices, $\mathbf{W}$ and $\mathbf{D}$
2. Solve the eigenvalue problem $\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\boldsymbol{\psi} = \nu\boldsymbol{\psi}$ and form the matrix $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_k]$ by stacking the k eigenvectors corresponding to the k largest eigenvalues as columns.
3. Normalize the rows of $\boldsymbol{\Psi}$ ($l_2$ norm)
4. Cluster the rows of $\boldsymbol{\Psi}$ into k clusters using the K-means algorithm.
5. Each row maps to a node in the graph. Therefore, every node belongs to the cluster that its corresponding row was mapped to.

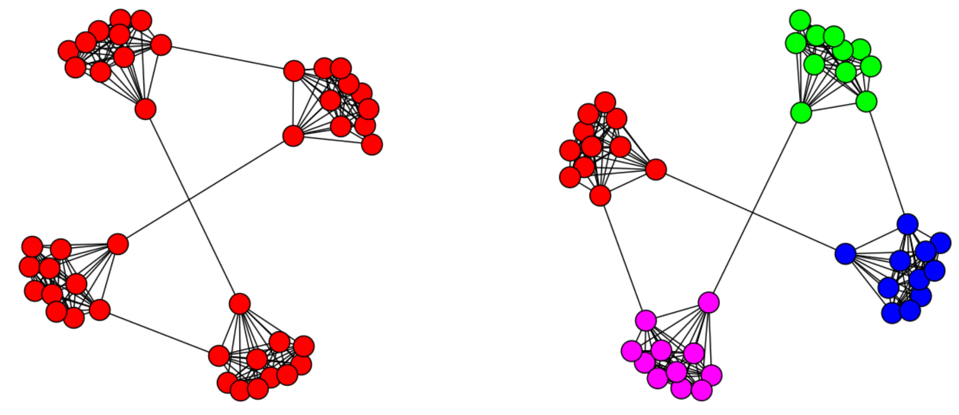

(a) Connected caveman graph (b) Clustered connected caveman graph

Figure 2.20: A basic example of spectral clustering

Similar in spirit to the graph drawing examples earlier, this algorithm essentially maps the nodes of the graph to the rows of the top k eigenvectors, obtaining a representation for each node in $\mathbb{R}^k$. After the mapping, the points are then clustered with the k-means algorithm. The points form tight clusters after the mapping as we already observed in some of the drawing examples, making it easier for the K-means method to obtain good results. Examples of that can be seen in Figures 2.19 and 2.20.

The requirement to solve an eigenvalue problem of course implies a significant computational cost. Thankfully, there are efficient methods to compute the first eigenvectors of matrices, the most popular ones being the power method or Krylov subspace methods such as the Lanczos method [121].

To get a better understanding of how this algorithm works, we are going to consider the ideal case where the points belonging to different clusters are infinitely far apart. Also, let the number of clusters be $k$ and for the sake of simplicity let the nodes be ordered together based on which cluster they belong (remember that node order does not affect eigenvalues and only permutes the elements within the eigenvectors). Based on the previous assumptions, the graph has $k$ connected components and therefore its adjacency and Laplacian matrices will be block diagonal, i.e of the form:

$$\mathbf{L_{sm}} = \begin{bmatrix} \mathbf{L_{sm}^{(1,1)}} & \dots & \dots & 0 \\ 0 & \mathbf{L_{sm}^{(2,2)}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \mathbf{L_{sm}^{(k,k)}} \end{bmatrix} \mathbf{W} = \begin{bmatrix} \mathbf{W^{(1,1)}} & \dots & \dots & 0 \\ 0 & \mathbf{W^{(2,2)}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \mathbf{W^{(k,k)}} \end{bmatrix} \tag{2.88}$$

We are working with $\mathbf{L_{sm}} = \mathbf{D^{-1/2}WD^{-1/2}}$ in this instance, and its eigenvalue decomposition is the union of the eigenvalue decompositions of its diagonal blocks, where the eigenvectors are appropriately padded with zeros. We know that for this matrix $\mathbf{L_{sm}} = \mathbf{I} - \mathbf{L_{sym}}$, so it is straightforward to show that we are using the eigenvectors of the normalized Laplacian, with the eigenvalues adjusted to be $1 - \nu_i$. Therefore the matrix $\mathbf{\Psi}$ will have the form

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{\Psi^{(1,1)}} & \dots & \dots & 0 \\ 0 & \mathbf{\Psi^{(2,2)}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \mathbf{\Psi^{(k,k)}} \end{bmatrix} \tag{2.89}$$

where $\mathbf{\Psi^{(i,i)}}$ are the eigenvectors corresponding to the block $\mathbf{L_{sym}^{(i,i)}}$. Let $\mathbf{Y^{(i,i)}}$ be the $\mathbf{\Psi^{(i,i)}}$ eigenvector block after it has been row-normalized. Then, the following proposition holds.

**Proposition 1** *Let* $\mathbf{W}$*'s off-diagonal blocks be zero and each of the* $k$ *clusters be connected. There exist* $k$ *orthonormal vectors* $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k$ *so that* $\mathbf{Y}$*'s rows, denoted by* $\mathbf{y}_j^i$ *where* $i$ *iterates through the blocks, satisfy*

$$\mathbf{y}_j^i = \mathbf{r}_i. \tag{2.90}$$

This guarantees that there are $k$ mutually orthogonal points mapped via the row normalization to the surface of the unit k-sphere, around which $\mathbf{Y}$'s rows will cluster. These clusters, correspond exactly to the true clustering of the original data. The above proposition holds in the idealized case where the off diagonal elements are zero and the clusters are perfectly distinguishable. In most applications, the nodes are not that well organized and the goal is to manage to recover similar performance guarantees like proposition 1. This is achieved by studying a perturbed version of the adjacency matrix $\hat{\mathbf{W}} = \mathbf{W} + \mathbf{E}$ and when its eigenvectors approximate those of $\mathbf{W}$.

For the sake of brevity, we have omitted certain details that are important for a complete understanding of the algorithm. For a more complete treatment, as well as a breakdown of the general perturbed case we refer the reader to the original paper by Ng et al. [86]. It is also important to mention that several versions of spectral clustering have been implemented with variations on the similarity metric, the type of matrix used (random walk Laplacian, normalized Laplacian etc.) or the parameters of the kernel. Different versions have different performance guarantees; for example the normalized Laplacian optimizes the Ncut quantity and thus both of the following:

1. Inter-cluster distance. Distance between points that belong to different clusters is maximized.

2. Intra-cluster distance. Distance between points that belong to the same cluster is minimized.

This is achieved by minimizing $cut(S)$(first objective) *and* maximizing $vol(S)$ (second objective). The unnormalized version only minimizes $cut(S)$ as it can be shown to optimize the cut ratio. Furthermore, from a statistical perspective the consistency of the clustering can only be guaranteed for normalized clustering and not for the unnormalized version as $n \to \infty$. However, the normalized Laplacian $\mathbf{L_{sym}}$ variant of spectral clustering can also run into problems if the there are many low degree vertices in the graph. This is why von Luxburg in [121] recommends using $\mathbf{L_{rw}}$, since it seems to be the most consistent in terms of overall performance.

Spectral clustering has been very successful in a multitude of applications. It empirically appears to allow capturing such salient features of a data set as its main clusters and submanifolds. This is unlike other manifold learning methods like LLE and Isomap which assume a single manifold and have not been designed to say something about the modes of the distribution [7].

## 2.7 Graph signal processing

Fourier analysis is one of the most important tools of digital signal processing. The concept of the Fourier transform is fundamental to many applications in the field. In the classical DSP setting, one often makes use of the frequency spectrum representation uncovered by the Fourier transform of the digital signal for the purposes of filtering.

On the real line, the complex exponentials $e^{i\omega x}$ defining the Fourier transform are eigenfunctions of the one-dimensional Laplacian operator $\frac{d^2}{dx^2}$. The inverse Fourier transform

$$f(x) = \frac{1}{2\pi} \int_{\mathbb{R}} \hat{f}(\omega) e^{i\omega x} d\omega \tag{2.91}$$

can be seen as the expansion of $f$ in terms of the eigenfunctions of the Laplacian operator. [42]. In [113] the concept of a Fourier transform on a surface signal was first introduced. The property of the Laplacian eigenfunctions to represent the vibrational modes of a surface was exploited for the purpose of surface smoothing. In follow up works [114] [115], Taubin further developed the framework that defines the Fourier transform on the nodes of a graph signal as the projection of the signal onto the eigenspace of its Laplacian. Let $\mathbf{x}$ be a signal defined on an undirected graph $G$ and

$$\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\boldsymbol{T}} \tag{2.92}$$

the decomposition of the Laplacian into its eigenvalues and eigenvectors, then the **Graph Fourier Transform** (GFT) is defined as

$$\hat{\mathbf{x}} = \mathbf{\Phi}^{\boldsymbol{T}}\mathbf{x}. \tag{2.93}$$

where $\hat{\mathbf{x}} = [\hat{x_1}, \hat{x_2}, \dots, \hat{x_n}]$ can be thought of as the graph frequencies of the signal. The inverse GFT is

$$\mathbf{x} = \mathbf{\Phi}\hat{\mathbf{x}} \tag{2.94}$$

In [96], the definition provided includes any operator associated with a graph, although our focus will remain on the Laplacian for the rest of this section.

In the standard DSP setting, smoothing can be done by computing the discrete Fourier transform of a signal, then discarding its high frequency components and then reverting back to the original representation with the inverse transform. A similar operation can be carried out here expressed as

$$\mathbf{x_{out}} = \mathbf{\Phi}\mathbf{H}\mathbf{\Phi}^{\boldsymbol{T}}\mathbf{x}, \tag{2.95}$$

where $\mathbf{H} = \mathbf{diag}(h(0), h(1), \dots, h(N))$. For a low pass operation, $\mathbf{H}$ could have the form

$$\begin{bmatrix} \mathbf{I_n} & 0 \\ 0 & 0 \end{bmatrix} \tag{2.96}$$

where $n$ is a suitably chosen cutoff. This means that the filtering would preserve the first $n$ low frequency components and drop the rest. In figure 2.21 we demonstrate a simple example of this operation.

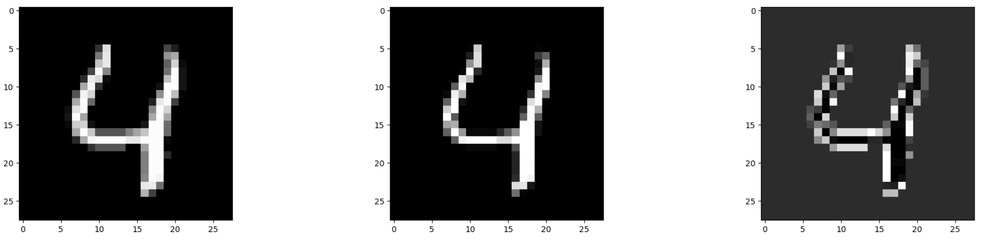


Figure 2.21: (a)Original image. (b)Image reconstructed from the first two eigenvectors(dropping the rest of the frequencies). (c) Image reconstructed from all the eigenvectors except the first two.

An important part of filtering operations in classical DSP is the convolution of a signal $x$ with a filter $h$, defined as

$$x_{out}(t) = \int_{\mathbb{R}} x(\tau)h(t-\tau)d\tau = (x * h)(t). \tag{2.97}$$

An important property is that convolution in the temporal domain corresponds to multiplication in the frequency domain. This is known as the convolution theorem

$$\int_{\mathbb{R}} x(\tau)h(t-\tau)d\tau = \int_{\mathbb{R}} e^{i\omega t}\hat{x}(\omega)\hat{h}(\omega)d\omega. \tag{2.98}$$

That property allows one to design filters with specified frequency characteristics that operate on the spatial domain. Notice that the definition involves the term $h(t-\tau)$ which implies translation. However, it is not clear how translation can be defined on a graph. Therefore, defining the convolution can also be tricky. One way to do it is by taking advantage of the convolution theorem and using the graph spectral domain for the definition. Let $\mathbf{x}, \mathbf{h} \in \mathbb{R}^{|V|}$ two signals defined on the graph $G(V, E)$. Then the graph convolution can be defined as

$$\mathbf{x} *_G \mathbf{h} = \mathbf{\Phi}((\mathbf{\Phi^T x}) \odot (\mathbf{\Phi^T h})). \tag{2.99}$$

Thus, convolving the graph signals can be viewed as element-wise multiplication of their graph spectral representations. The graph spectral definition is lacking compared to the classical, in the sense that we still have to compute the eigendecomposition of the graph Laplacian in order to perform the filtering. For large graphs this proves to be a considerable obstacle, since the eigendecomposition is computationally expensive $O(|V|^3)$. Working on the spectrum does not guarantee any localization of the filter in the spatial domain. This could also be a problem depending on the application.

There are two possible approaches. The first one deals mostly with the computational concern and makes use of the properties of product graphs that we described in an earlier section. That is, if we know that a graph can be expressed as a Cartesian product of factor graphs, then its spectrum can also be computed through the spectrum of its factors. We are focusing on the Cartesian product here since our definition of the GFT involves the Laplacian (for the adjacency spectrum any of the other two products can be computed from its factors). We can demonstrate this using a simple example. Consider the $n$th dimensional hypercube graph. The vertices are $2^n$, so for large values of $n$ the Laplacian eigenvectors would be really expensive to compute. It is known that the $n$ dimensional hypercube can be expressed as the Cartesian product of $n$ edge graphs. An edge graph is a graph that contains two vertices that are connected by an edge. It is therefore sufficient to compute the Laplacian spectrum of the edge and then using

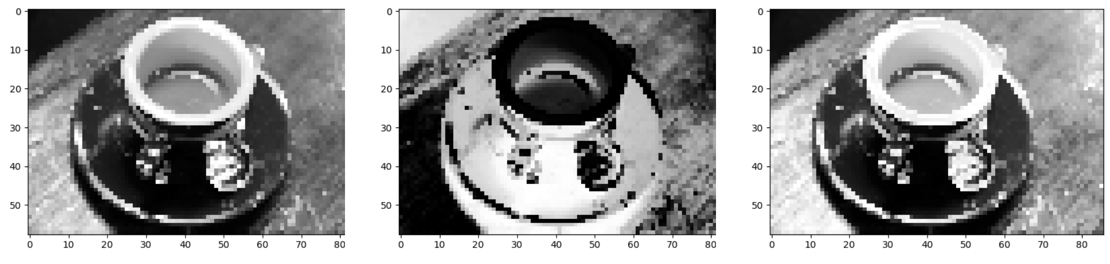
$$\mathbf{L}_{G\diamond H}(\boldsymbol{\phi}_i^G \otimes \boldsymbol{\phi}_j^H) = (\lambda_i^G + \lambda_j^H)(\boldsymbol{\phi}_i^G \otimes \boldsymbol{\phi}_j^H)$$

express the eigenvalues of the hypercube as sums of the eigenvalues of edges and the eigenvectors as Kronecker products of the edge eigenvectors. This would give us the GFT of the hypercube graph. The eigenvectors of the edge graph, correspond to the first two column vectors of the DFT matrix. A $n$-dimensional discrete Fourier transform can be computed from Kronecker products of $n$ 1-D DFT matrices [1]. Consequently, the eigenvectors of the hypercube match the basis vectors of the $n-$ dimensional DFT. Of course, this is an easy case since the hypercube has special structure that can be exploited. A general Fast Fourier Transform algorithm does not exist for general graphs as they do not exhibit the same regularities.

In the second approach, we would like to be able to operate directly on the values of the nodes in a way that that allows us to manipulate the graph spectral characteristics of the signal. We can get a hint of how to do this by looking at the standard multiplication with the Laplacian matrix,

$$\mathbf{Lx} = (\mathbf{\Phi\Lambda\Phi^T})\mathbf{x} = \mathbf{\Phi\Lambda}(\mathbf{\Phi^T x}) \tag{2.100}$$

Multiplication with the Laplacian first computes the signal's graph spectrum, then scales the graph frequency components according to the eigenvalues $\lambda$ and finally reverts back by multiplying with the inverse GFT.



(a) (b) (c)

Figure 2.22: (a)original image, (b)image filtered with $\mathbf{I} - \mathbf{L}$,(c) image filtered with $\mathbf{I} - \mathbf{L_{sym}}$. The eigenvalues of $\mathbf{L}$ are not bounded like $\mathbf{L_{sym}}$ which can lead to more unpredictable results.

Taubin in [114] recognized the cost of computing the eigenvectors and formulated graph spectral filtering in terms of the Laplacian. Specifically, he formed filters of the type

$$\mathbf{x}' = (\mathbf{I} - c\mathbf{L})^k\mathbf{x} \tag{2.101}$$

where $c$ a chosen parameter and the power $k$ is just iterative application of $\mathbf{I} - \mathbf{cL}$, $k$ times. He observed that this was equivalent to multiplying $\hat{x}_i$ with $(1 - c\lambda_i)^k$. However this type of filtering gets dominated by the DC eigenvector and the rest of the frequencies vanish. Taubin called this effect shrinkage. In order to address that he designed filters using the Chebyshev polynomial [114]. This entails filtering the signal with a polynomial of the Laplacian $h(\mathbf{L})$ which corresponds to filtering the graph frequencies of the signal with a polynomial on the eigenvalues $h(\lambda)$. Additionally, the localization of the filter in the spatial domain can also be controlled with the parameter $k$ as each multiplication with $\mathbf{L}^k$ operates on a $k$-hop neighborhood of each node. We will see more on that approach in the next chapter on the subject of graph convolutional networks.

While graph signal processing has been utilized in graphics for mesh processing [73], the field began drawing more attention after works that extended the basic GFT framework to wavelets on graphs [42] [105], works that set the theoretical and practical foundations and related its concepts to their counterparts in digital signal processing [106] [95] [96], as well as works like [31] that expressed the bilateral filter, a well known image processing filter, in terms of the graph spectrum of a digital image. Since then, the theory has been developed further but the concepts have also found applicability in the field of data science and machine learning. Explaining how these relate to applications in machine learning and neural networks will be one of the central themes of the next chapter.

# Chapter 3

# Deep Learning and Graph Neural Networks

## 3.1 Brief History

The foundations of neural networks and deep learning begin with the work of McCulloch and Pitts [82] who first described the behavior of neurons (thresholded units) in a systematic way that connects them to computation. They observed that neuronal activity can be modeled by logic propositions due to the binary nature of the neuron excitations. A few years later Kleene [64] connected the framework of Pitts and McCulloch to finite automata. These formulations about neurons did not explicitly address the concept of learning. Some of the first ideas about learning came from Hebb [44]. In the following decades Rosenblatt's Perceptron [92] was one of the first neural network models that was trained in a supervised way. In [54] Ivankhenko described what could be condisdered a precursor of a deep neural network as he was able in that period to train networks with multiple hidden layers. The 80s was the decade where a lot of the ideas about training neural networks were firmly established and successfully applied. In [30] Fukushima proposed the neocognitron. There he described a network with hierarchical structure that contained receptive fields; two-dimensional arrays of variables that were learned through training. He recognized that the lower levels of the architecture learned local features and patterns while the higher levels more global characteristics. The neocognitron also implemented a version of pooling. That decade also saw the rise of backpropagation algorithm for training neural network weights. The first formulation of backpropagation on neural networks was presented in [124] by Werbos, and later on independently by Lecun [69] and Rumelhart et al [93]. During this period Hinton et al. also formulated the Restricted Boltzmann machine (RBM). In the following years, Lecun et al. described in [71] [72] the basic form of convolutional neural networks as they are known today, trained with backpropagation. Additionally, some of the first deep RNNs were developed by Schmidhuber [100] early in the 90s as well as the famous LSTM [50] later on in that decade . By the end of the 90s kernel machines [118], [119] [101] [102], had overtaken the field by achieving state of the art results in various tasks and contests. The SVM dominance lasted until halfway through the 2000s, and while the progress of neural networks did not stop [7], the papers [48] [49] [6] revived interest in deep feedforward architectures and sparked the deep learning "revolution". They demonstrated that deep architectures could be efficiently trained by backpropagation after a greedy layer-wise training procedure. The interest in these architectures further spiked as hardware improved and training became more efficient [91], and in 2012 a deep convolutional network won the ImageNet [67]. The network achieved almost half the error rates of the then-best competing approaches. This was accomplished with the use of ReLU units [35], dropout [110] as well as data augmentation methods.

## 3.2 Convolutional neural networks

Convolutional neural networks (CNNs) have been at the heart of the deep learning revolution, achieving state of the art results at various classification tasks in recent years, such as image recognition [107] and sentence classification [60]. Convolutional networks typically work well on domains where there is additional structure between features, such as the grid structure of the pixels of an image.

CNNs are designed to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays containing pixel intensities in the three colour channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images. There are four key ideas behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers. The architecture of a typical ConvNet is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU. All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks.

The most prominent example of this is the standard architecture of convolutional neural networks (CNNs), which typically consist of alternating convolutional layers and pooling layers. The convolutional layers endow these models with the property of translation equivariance. The pooling layers provide local translation invariance, by combining the feature representations extracted by the convolutional layers in a way that is position-independent within a local region of space. Together, these layers allow CNNs to learn a hierarchy of feature detectors, where each successive layer sees a larger part of the input and is progressively more robust to local variations in the layer below.

We mention some of the important assumptions we make about the data when working with convolutional networks, as presented in [9]. Consider a compact d-dimensional Euclidean domain $\Omega = [0,1]^d \subset \mathbb{R}^d$ on which square integrable functions $f \in L^2(\Omega)$ are defined. For images these can be thought of as functions on the unit square $\Omega = [0,1]^2$. In a generic supervised learning setting, an unknown function $y : L^2(\Omega) \to \mathcal{Y}$ is observed on a training set

$$\{f_i \in L^2(\Omega),\ y_i = y(f_i)\}_{i \in \mathcal{I}}. \tag{3.1}$$

For supervised classification , the target space $\mathcal{Y}$ can be considered discrete with $|\mathcal{Y}|$ being the number of classes. Two core assumptions about the function $y$ are:

- Stationarity. Let
$$\mathcal{T}_i f(x) = f(x - i) \quad x, i \in \Omega \tag{3.2}$$
be a translation operator acting on functions $f \in L^2(\Omega)$. The first assumption is that the function $y$ is either invariant or equivariant with respect to the translation operator. **Invariance** is defined as
$$y(\mathcal{T}_i f) = y(f) \quad \text{for any } f \in L^2(\Omega) \tag{3.3}$$
and **equivariance** as
$$y(\mathcal{T}_i f) = \mathcal{T}_i y(f). \tag{3.4}$$
This means that if we translate an image and then feed it through a convolution layer, the output will be the same as if we had fed the original image through the convolution layer and then translated the output.

- Local deformations and scale separation. A deformation $\mathcal{L}_\tau$ where $\tau : \Omega \to \Omega$ is a smooth vector field, acts on $L^2(\Omega)$ as
$$\mathcal{L}_\tau f(x) = f(x - \tau(x)). \tag{3.5}$$
Deformations can model local translations, changes in point of view, rotations and frequency transpositions. Most tasks in computer vision are stable with respect to local deformations. In translation invariant tasks we have
$$|y(\mathcal{L}_\tau f) - y(f)| \approx ||\nabla \tau|| \tag{3.6}$$
It follows from this that we can extract sufficient statistics at a lower spatial resolution by downsampling demodulated localized filter responses without losing approximation power. An important consequence of this is that long-range

dependencies can be broken into multi-scale local interaction terms, leading to hierarchical models in which spatial resolution is progressively reduced.

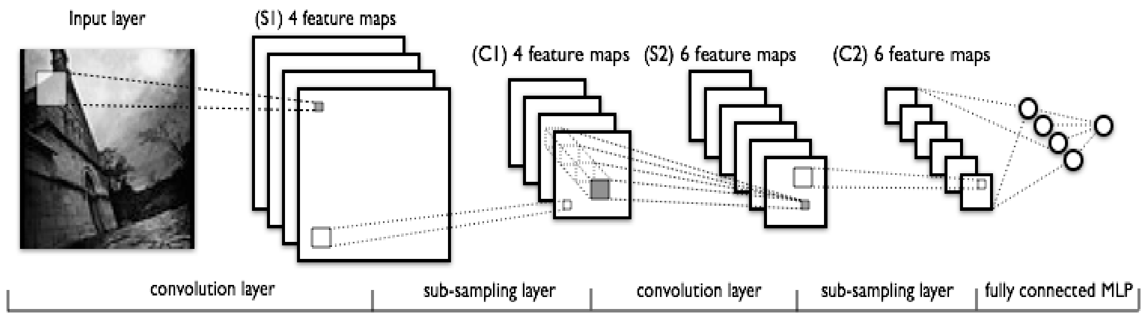


Figure 3.1: Illustration of a convolutional neural network. Image from deeplearning.net

## 3.3 Recent Developments

In this section we will be taking a look at recent works in the field that we consider to be impactful or that bring up new and interesting ideas. This is by no means a comprehensive list as there are many high quality papers being released by various authors all around the world. This only reflects the preferences of the author.

### 3.3.1 Group equivariant CNNs and Steerable CNNs

In [17], Cohen and Welling introduce the group equivariant convolutional neural network (G-CNN) . In that paper they make use of ideas from the mathematical field of group theory in order to exploit the symmetries of the data in a more efficient way on CNNs. Here are some of the basic concepts they presented.

A transformation of an object is called a **symmetry** if the object is invariant to that transformation. Additionally, given two symmetry transformations, their composition should also be a symmetry. Symmetry transformations also admit an inverse transformation which itself is also a symmetry. A set of transformations with these properties is called a **symmetry group** .

A regular CNN computes the feature masks by translating a convolution kernel on the grid of the image. The authors achieve state of the art results on datasets like CIFAR10 by replacing regular convolution layers by group convolutions. Essentially, instead of restricting the convolution masks to translations on the grid, they compute feature masks for all the symmetry transformations that are contained on certain symmetry groups. For example, the group $p4m$ consists of all compositions of translations, mirror reflections, and rotations by 90 degrees about any center of rotation in the grid.

This lead to their next work on steerable CNNs [18]. Steerable filters were first introduced in [28] . To give a sense of the concept of steerability we are going to see one of the basic examples that are usually presented in this case. Consider a Gaussian $G$ written in coordinates $x$ and $y$:

$$G(x, y) = e^{-(x^2+y^2)}. \tag{3.7}$$

The first derivative of the Gaussian in the $x$ direction is

$$G_1^{0^\circ} = \frac{\partial}{\partial x} e^{-x^2+y^2)} = -2xe^{-(x^2+y^2)} \tag{3.8}$$

and for the $y$ direction

$$G_1^{90^\circ} = \frac{\partial}{\partial y} e^{-x^2+y^2)} = -2ye^{-(x^2+y^2)}. \tag{3.9}$$

To achieve a filter $G_1$ at an arbitrary orientation we can take a linear combination of the two derivatives

$$G_1^{\theta} = \cos(\theta) G_1^{0^\circ} + \sin(\theta) G_1^{90^\circ}. \tag{3.10}$$

Thus, to get an image filtered at an arbitrary angle, one can take linear combinations of the images convolved with $G_1^{0^\circ}$ and $G_1^{90^\circ}$. In [18], the authors sought to generalize the ideas from group equivariant CNNs even further. For example, on G-CNNs the groups they tested considered filters that were rotated by multiples of 90 degree angles. The notion of steerability naturally leads one to consider filters of all possible angles that can identify any pose. Thus, they further expand on their ideas about group CNNs and develop the framework of steerable CNN which includes group CNNs as a special case. Going into more detail would require a fair amount of groundwork on representation theory and context which is outside the scope of our work here.

### 3.3.2 Generative Adversarial Networks

Generative adversarial networks (GANs) proposed by Goodfellow et al. [37] saw a massive increase in popularity in the last couple of years with dozens of new GAN variants being proposed [75] [13] [76] [2]. It is framework for estimating generative models via an adversarial process, in which two models are trained simultaneously. A model $G$ (generator) that captures the data distribution, and a discriminative model $D$ (discriminator) that estimates the probability that a sample came from the training data rather than $G$. The training procedure for $G$ is to maximize the probability of $D$ making a mistake, and for $D$ to maximize the probability of assigning the correct label to both samples from $G$ and the the dataset. The generator is mapping input noise variables to the data space via a multilayer perceptron. Similarly, the discriminator $D(\mathbf{x})$ is also a multilayer perceptron that outputs a single scalar which is the probability that the input vector $\mathbf{x}$ came from the data rather than the generator.

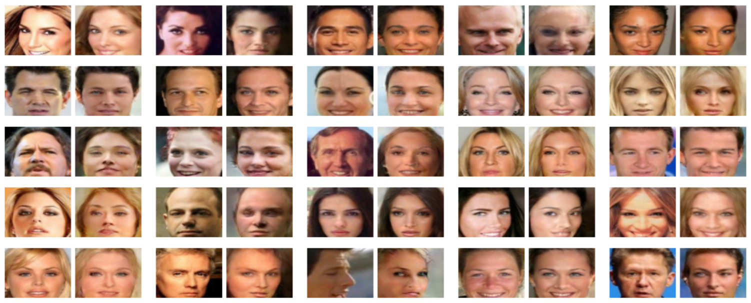


Figure 3.2: Images generated from a GAN.

### 3.3.3 Attention

One of the most curious aspects of the human visual system is the presence of attention. Instead of compressing an entire image into a static representation, attention allows for salient features to dynamically come to the forefront as needed. This is especially important when there is a lot of clutter in an image. Using representations that reduce information in an image down to the most salient objects is one effective solution that has become more popular in recent years [84] [127] [79]. One application of that is caption generation for images. The image can be divided into $n$ parts and representations of each can be computed by a CNN. These representations (feature vectors denoted by $\mathbf{a}_i$) are often called annotation vectors. Then, an LSTM is generating new words with the attention mechanism focusing on the relevant parts of the images. The attention mechanism takes as input the annotation vectors and the hidden state of the LSTM at time $t-1$ and computes positive annotation weights $w_i$ that encode the relevance of each location $i$. Then, using the computed weights and the annotation vectors, a function $\phi$ outputs the context vector, a dynamic representation of the relevant part of the image input at time $t$. Two types of attention can be distinguished in that setting. Soft attention and hard attention.

In hard attention, for the $t$-th word the context $\hat{z}_t$ is a Multinoulli random variable with

$$p(s_{t,i} = 1 | s_{j<t}, \mathbf{a}) = w_{t,i} \tag{3.11}$$

where $s_{t,i}$ is an indicator one-hot variable set to 1 if the $i$-th location is the one used for the visual features. In other words, the output context is a sampled annotation vector based on a Multinoulli distribution that is parametrized by the annotation weights $w_i$.

Soft attention is a deterministic variant of the attention mechanism. Given annotation vectors and annotation weights, the expected annotation vector is computed, where $w_i$ represent the probabilities. Thus, the output of the mechanism is a weighted context. This is a differentiable model and therefore it is easily trainable end-to-end with backpropagation.

### 3.3.4 Neural Style transfer

The key finding of the work in [32] is that the representations of content and style in the Convolutional Neural Network are separable. That is, both representations can be manipulated independently to produce new, perceptually meaningful images.

Higher layers in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. In contrast, reconstructions from the lower layers simply reproduce the exact pixel values of the original image. Therefore the feature responses in higher layers of the network are considered the content of the representation. To obtain a representation of the style of an input image, they use a feature space originally designed to capture texture information. This feature space is built on top of the filter responses in each layer of the network. It consists of the correlations between the different filter responses over the spatial extent of the feature maps. By including the feature correlations of multiple layers, they obtain a stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement. In other words, by using a large pretrained network (VGG) one can train an image with gradient descent using two objectives:

- Squared error loss between *content* features of the generated image and the original image. These have the form of a stack of feature maps in vector form on a given layer.

- Squared error loss between *style* features of the generated image and the original image. These have the form of the Gram matrix of the feature maps on a given layer.

As a quick demonstration we have generated some images with style transfer using VGG. The images in 3.5 have been generated by combining the source images in 3.4 and the style of the reference image in 3.3.

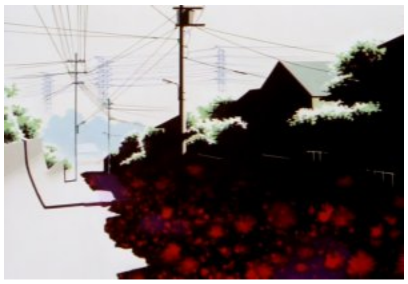


Figure 3.3: Reference image.

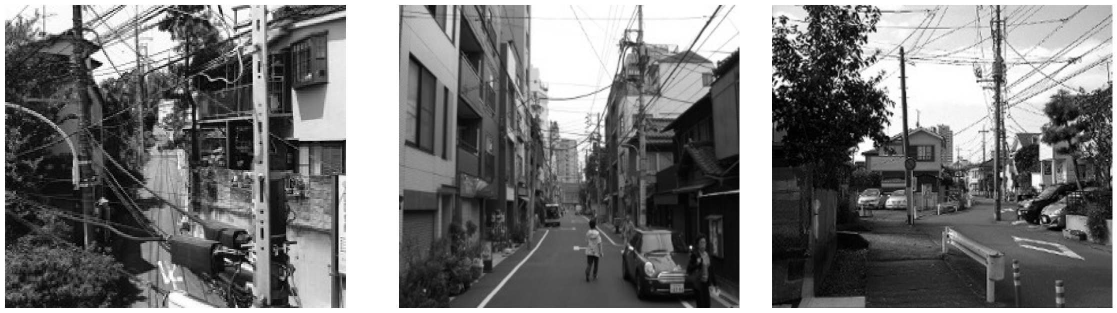


Figure 3.4: Original images.

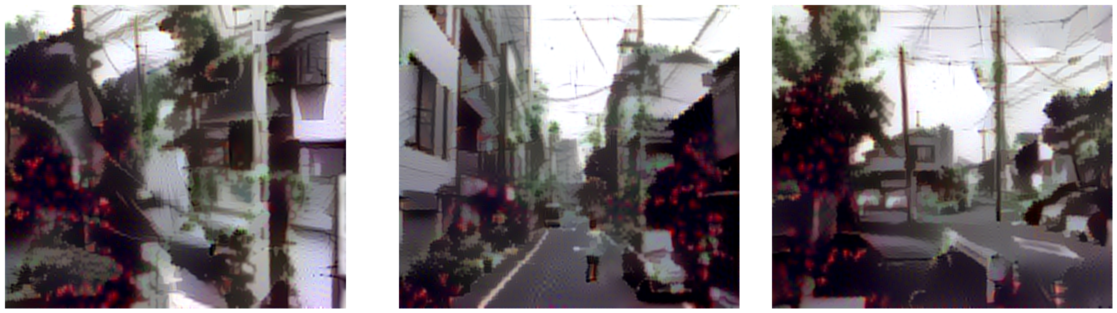


Figure 3.5: Generated images.

### 3.3.5 Connections to other fields

While the performance of deep learning models has skyrocketed in the past decade, the theoretical underpinnings are still investigated in order to explain 'why does deep learning work so well'. The paper by Mehta et al. [83] points to a connection between the renormalization group (RG) in physics and deep learning. They demonstrated that there is a one-to-one mapping between RBM-based Deep Neural Networks and the variational renormalization group. RG is an iterative coarse-graining scheme that allows for the extraction of relevant features (i.e operators) as a physical system is examined at different length scales.

One of the standard ways of studying the RG is the Ising model. The one-dimensional Ising model describes a collection of binary spins $v_i$ organized along a one-dimensional lattice with lattice spacing $a$. Such a system is described by a Hamiltonian of the form

$$H = -J \sum_i v_i v_{i+1} \tag{3.12}$$

where J is a ferromagnetic coupling that energetically favors configurations where neighboring spins align. A generalization of the Ising model is the Potts model. The Potts model has been connected to neural networks for solving optimization problems in [90] as well as used for ICA [125] and signal recovery [111]. Additionally, the Potts model has connections to graph theory via the Tutte polynomial [123] and invariants in knot theory [55].

The renormalization group plays a central role in our modern understanding of statistical physics and quantum field theory. A central finding of RG is that the long distance physics of many disparate physical systems are dominated by the same long distance fixed points. This gives rise to the idea of universality. Many microscopically dissimilar systems exhibit macroscopically similar properties at long distances. Physicists have developed elaborate technical machinery for exploiting fixed points and universality to identify the salient long distance features of physics systems. Therefore, a sensible question to ask is if any of the mathematical tools developed by physicists can be employed in the field of deep learning [83].

Finally, works like [89] and [16] as well as the steerable CNNs discussed earlier view deep learning through the lens of group theory and group representations. Incidentally, group representation theory has also been extensively studied in physics as for example Lie groups are intimately connected to very successful theories like the Standard Model.

## 3.4 Graph Neural Networks

Graph convolutional networks (GCNs) are the models that aim to generalize convolutional layers to data that reside on arbitrary graphs. It is a new subfield of machine learning that has grown significantly in the past few years and has found applications in quantum chemistry [33], chemoinformatics [59], brain networks [68] and disease prediction [88]. GCNs have also had success as substitutes of standard methods within the machine learning field on tasks like semi-supervised learning [62] [80], learning variational autoencoders [63], matrix completion [9], community detection [11], neural machine translation and natural language processing [3] [81] .

In [4] the authors proposed a regularization term that enforced local smoothness on graph data. This meant that the graph structure was encoded in the loss function and could be used to train kernel machines. There have been various attempts to generalize neural networks to the graph setting. Initially, Scarselli et al., proposed the graph neural network (GNN) which was an extension of recurrent neural networks. In that network a contraction mapping was iteratively applied until the convergence of the dynamical system, and the weights were trained with a regular gradient descent strategy [38] [98] [99]. However, there was not much progress with these types of models until the work by Bruna et al. [12] on spectral networks and the spectral formulation of graph convolutions which was further expanded upon in [46].

### 3.4.1 Graph convolutions

The core concept behind these networks is the graph convolution. One of the basic properties of the convolution is that it is equivalent to multiplication in the Fourier domain. That is the famous result known as the convolution theorem. Similarly, when defining the graph convolution, an intuitive approach is for it to have the same property; to be a multiplication operation in the spectrum of the graph's Laplacian (graph Fourier domain). Based on this observation a possible definition of the graph convolution is the following.

Let $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\mathbf{T}}$ be the eigenvalue decomposition of the graph's Laplacian, and $\mathbf{f}, \mathbf{y} \in \mathbb{R}^N$ two signals defined on the graph $G(V, E)$.

$$\mathbf{f} *_G \mathbf{y} = \mathbf{\Phi}((\mathbf{\Phi}^{\mathbf{T}}\mathbf{f}) \odot (\mathbf{\Phi}^{\mathbf{T}}\mathbf{y})) \tag{3.13}$$

where $*_G$ denotes the graph convolution on $G$ and $\odot$ denotes the Hadamard product. The products $\mathbf{\Phi}^{\mathbf{T}}\mathbf{f}$ and $\mathbf{\Phi}^{\mathbf{T}}\mathbf{y}$ denote the graph Fourier transform.

From that definition a natural way to define a graph convolution layer in a neural network is to learn a vector $\mathbf{w}$ of spectral multipliers (weights) that will operate on the graph Fourier domain of the signal. A desired property of these convolutions would be small support on the feature (spatial) domain. By default, this is not guaranteed in the layer defined above. Smoothness in the spectrum implies spatial decay, therefore Henaff et al. in [46] achieved localization by multiplying the weight vector with an interpolation kernel.

However, the requirement of the eigenvector matrix $\mathbf{\Phi}$ implies scalability problems in this approach. It would be desirable to avoid the computation of the eigendecomposition.

We can express the output of a graph convolution layer as

$$\mathbf{x} *_G \mathbf{w} = \mathbf{\Phi} diag(\mathbf{w}) \mathbf{\Phi}^{\mathbf{T}}\mathbf{x} \tag{3.14}$$

That is, we consider the graph convolution with $\mathbf{w}$ as elementwise multiplication of $\mathbf{\Phi}^{\mathbf{T}}\mathbf{x}$ with $\mathbf{w}$ followed by the inverse trasnform. Defferrard et al. extended this framework in [21] and expressed

$$\mathbf{x} *_G \mathbf{w} = w(\mathbf{L})\mathbf{x} \tag{3.15}$$

in terms of the Chebyshev polynomial $T_k(x)$. The order k Chebyshev polynomial can be computed as

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{3.16}$$

with $T_0 = 1$ and $T_1 = x$. Specifically, we have

$$w(\mathbf{L}) = \sum_{k=0}^{K-1} w_k T_k(\tilde{\mathbf{L}}) \tag{3.17}$$

where $T_k(\tilde{\mathbf{L}})$ is the Chebyshev polynomial of order $k$ evaluated at the scaled Laplacian $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}_N$.

This extension of the framework is more efficient. From (3.16) we see that the Chebyshev polynomial can be computed recursively. This formulation is also localized on the spatial domain due to the product with the Laplacian operator. The degree of the Chebyshev polynomial controls the localization of the filter; the order k Laplacian includes k-hop neighborhoods in the filtering process. This polynomial corresponds to filtering the spectral domain with the Chebyshev polynomial of order k evaluated at $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}_N$, a diagonal matrix of scaled eigenvalues that lie in [-1,1].

Kipf et al. [62] demonstrated competitive results on node classification of citation networks and knowledge graphs with a simple first order linear approximation

$$\mathbf{g} *_G \mathbf{x} \approx \theta(\mathbf{I} + \mathbf{D^{-1/2}}\mathbf{A}\mathbf{D^{-1/2}})\mathbf{x} \tag{3.18}$$

with $\theta$ being a tunable parameter. The complete filtering layer included also a fully connected weight matrix $\mathbf{W}$ and a ReLU activation $\sigma(\mathbf{x})$ so the forward pass is

$$f(\mathbf{x}) = \sigma\left(\mathbf{g} *_{\mathbf{G}} (\mathbf{W}\mathbf{x})\right). \tag{3.19}$$

This is a semi supervised approach, as multiplication of the data with the similarity matrix $\mathbf{D^{-1/2}AD^{-1/2}}$ tends to naturally cluster the data in an unsupervised way, which can then be followed up by supervised training. An important detail in this implementation was that the graph was provided externally and not formed by the data. The success of this approach was attributed by the authors to its resemblance to the Weisfeiler-Lehman algorithm [24] [103]. Briefly, the main idea of the algorithm is to repeatedly assign values to vertices and their neighborhoods until convergence. Let $x_1, x_2, \ldots, x_n$ be the values assigned to the vertices of a graph $G$. Each iteration proceeds with the following steps: For all vertices $v$ in $G$

- Compute a hash of $v$'s neighboring values $x_1, x_2, \ldots, x_k$.
- Assign the value of the hash to $v$ for the next iteration.

This repeats until all the vertices have been assigned a unique value. This algorithm can also be utilized to check for graph isomorphism by comparing the values assigned to two different graphs. If the values are different then the graphs are not isomorphic. It is important to note that this is not guaranteed to converege and fails for graphs that have a lot of symmetry, e.g stars, rings etc.

Repeated application of the filtering in (3.20) can lead to numerical instability. For that reason the proposed a renormalization procedure so that the filtering becomes

$$\mathbf{g} *_G \mathbf{x} \approx \theta(\mathbf{I} + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2})\mathbf{x} \tag{3.20}$$

where the adjacency and the degree matrices have been adjusted so that

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \tag{3.21}$$

and the diagonal degree matrix $\tilde{\mathbf{D}}$ with entries

$$\tilde{d}_i = \sum_j \tilde{\mathbf{A}}(i,j). \tag{3.22}$$

At this point it is worth noticing that

$$(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}) = \mathbf{2I} - \mathbf{L_{sym}} = \tilde{\mathbf{L}}_{\mathbf{sym}}. \tag{3.23}$$

The eigenvalues of the symmetric Laplacian lie in the $[0,2]$ interval and therefore the eigenvalues of the filter in (3.23) are reversed. Filtering a vector $\mathbf{x}$ with powers $k$ of that filter, where $k$ a sufficiently large positive integer, would make $\mathbf{x}$ converge to the eigenvector corresponding to the largest eigenvalue (appropriately scaled) of the filter. This is the main observation behind the popular power method for estimating eigenvalues of a matrix. The top eigenvector of $\tilde{\mathbf{L}}_{\mathbf{sym}}$ is the constant eigenvector corresponding to eigenvalue 2. From there, a strategy commonly known as **deflation** can be utilized to compute the rest of the eigenvalues. Typically, a rank one modification is applied to the original matrix so as to displace the top eigenvalue, while keeping all other eigenvalues unchanged. The rank one modification is chosen so that the second eigenvalue becomes the one with largest modulus of the modified matrix and therefore, the power method can now be applied to the new matrix to extract the second largest eigenvalue and eigenvector. [94] Of course the leading eigenvector is known, thus $\tilde{\mathbf{L}}_{\mathbf{sym}}$ can be modified as

$$\tilde{\mathbf{L}}'_{\mathbf{sym}} = \tilde{\mathbf{L}}_{\mathbf{sym}} - 2(\mathbf{D^{1/2}1})(\mathbf{D^{1/2}1})^{\mathbf{T}} \tag{3.24}$$

Now the power method with powers $k$ of that matrix would approximate the second largest (Fiedler) vector. As we have already discussed in the spectral graph drawing and spectral clustering sections of the previous chapter, that would lead to an optimal assignment of values to the nodes of the graph in terms of the Rayleigh quotient of the filter. This in turn could be utilized for the purposes of segmentation/clustering.

It is also worth pointing out another connection of these approaches to the theory for eigenvalue approximation. Consider the filter presented in (3.17). The vectors produced by the filter lie in the Krylov subspace of $\tilde{\mathbf{L}}$ as they are essentially linear combinations of positive powers of $\tilde{\mathbf{L}}$. The **Krylov subspace** of $\tilde{\mathbf{L}}$ is defined as

$$\mathcal{K}_N = \operatorname{span}\{\mathbf{x}, \tilde{\mathbf{L}}\mathbf{x}, \tilde{\mathbf{L}}^2\mathbf{x}, \ldots, \tilde{\mathbf{L}}^{N-1}\mathbf{x}\}. \tag{3.25}$$

It is easy to notice that this space is the same that is spanned by the vectors of the power method iteration. Furthermore, if one incorporates a classical Gram-Schmidt orthogonalization process for each step of the power method the result is going to be the well known Arnoldi method. This method constructs an orthogonal basis of the subspace $\mathcal{K}_N$ [94]. Additionally it can be shown that the eigenvectors produced from the Arnoldi method are good approximations of the eigenvectors of $\tilde{\mathbf{L}}$. It is conceivable that a process like that could be utilized to achieve graph spectral filtering by approximating the eigenvectors in a feedforward neural network. The first filtering layers could implement the power iteration along with the orthogonalization in a ResNet [43] manner, followed by learning a spectral multiplication kernel like the one in [46].

There have also been other vertex domain formulations for the graph convolution such as the ones in [87] and [120]. Vialatte et al. proposed a convolution operator defined for two signals $\mathbf{f}, \mathbf{y}$ as

$$\mathbf{f} *_G \mathbf{y} = (\mathbf{f}^{\mathbf{T}} \otimes \mathbf{A})\mathbf{y} \tag{3.26}$$

where $\otimes$ denotes the Kronecker product and $\mathbf{A}$ is an allocation matrix with binary entries. They were able to demonstrate that this operation reduces to regular convolutions in regular domains (grids).

Finally, Monti et al. in [85] define a patch operator $D$ that acts on function $f$ defined on the graph nodes $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ as

$$D_j(x)f = w_j(\mathbf{u}(x, y))f(y), \quad j = 1, \ldots, J \tag{3.27}$$

where $w$ are pseudo-coordinates associated with nodes $y \in N(x)$

$$\mathbf{w}_{\mathbf{\Theta}}(\mathbf{u}) = [w_1(\mathbf{u}), \ldots, w_J(\mathbf{u})] \tag{3.28}$$

parametrized by some trainable parameters $\mathbf{\Theta}$. For the continuous case, it is remarked that $f$ can be a function on points $x$ on a manifold and the sum will be appropriately replaced by the corresponding integral. Using the above, they define this generalization of spatial convolution

$$(f * g)(x) = \sum_{j=1}^{J} g_j D_j(x) f. \tag{3.29}$$

In fact, the authors were able to represent some of the previously mentioned models as special instances of their generalized framework.

### 3.4.2 Graph Pooling

One of the foundational building blocks for regular convolutional networks is the pooling layer. Pooling helps reduce the dimensionality of the data and augments the hierarchical structure of the features extracted from a convolutional net. While it is straightforward to do max or mean pooling on a regular grid or on a line, generalizing this to arbitrary graphs can be challenging.

One of the prevalent approaches that has been successful in [85] and [21] utilizes the Graclus graph pooling algorithm which works as follows [22]. Given a graph $G = (V, E)$, the algorithm coarsens the graph into $G_1, G_2, \dots G_n$ by repeatedly transforming it into smaller graphs such that $|V_1| > |V_2| > \dots |V_m|$. To coarsen the graph from $G_i$ to $G_{i+1}$ sets of nodes in $G_i$ are combined into 'supernodes' in $G_{i+1}$. This is achieved by first visiting each node in random order. For each vertex $i$ that is not marked, merge $i$ with its unmarked neighbor $j$ that maximizes

$$\frac{w_{i,j}}{w(i)} + \frac{w_{i,j}}{w(j)} \tag{3.30}$$

where $w_{i,j}$ the edge weight and $w(i), w(j)$ the weights of the vertices $i$ and $j$. Mark $i$ and $j$ and proceed to the next unmarked vertex. If all neighbors of unmarked $i$ are themselves marked then $i$ is marked but not merged with any vertex. This whole process can be repeated to get multiple levels of coarsening for the graph. The criterion in (3.30) is reminiscent of the ratios we presented for clustering in the previous chapter. Indeed, if the denominators of the fractions (vertex weights) are such that

$$w(i) = d_i \tag{3.31}$$

where $d_i$ the degree of vertex $i$, then this criterion optimizes the normalized cut. Otherwise, if the denominators are set to 1 then the Kernighan-Lin objective is optimized. This objective is a slight modification of the ratio cut that we have already discussed.

Alternatively, the authors in [112] introduced the graph embed layer. This layer trains an embedding matrix $\mathbf{V}_{emb}$ that pools the vertices so that

$$\mathbf{x}_{pooled} = \mathbf{V^T}_{emb}\mathbf{x} \quad \text{and} \tag{3.32}$$

$$\mathbf{A}_{pooled} = \mathbf{V^T}_{emb}\mathbf{A}\mathbf{V} \tag{3.33}$$

where $\mathbf{A}$ is the adjacency matrix of the graph. Ultimately, it remains an open question which type of pooling works best for which graphs.

### 3.4.3 Parameters

While the graph convolution is the main building block of these models, there are many additional parameters and design choices that are crucial when tackling any graph problem. In previous works, models built around the graph convolution tend to have a smaller number of parameters [12]. That can be interpreted as an advantage or a drawback depending on the application. For example, when filtering signals on graphs with highly regular structure such as grids (images), the symmetric nature of the Laplacian and adjacency operators leads to filters that are more restricted when compared to the standard convolution operator. That is, if we consider a graph defined on the pixels of the image, we will have $w(i, j) = w(j, i)$ for two pixels $i, j$. We can se This in turn implies that the filters will not be able to properly recognize orientation (i.e they are rotation invariant), in contrast to regular convolutions where the pixels $i, j$ will be weighted differently depending on the current position of the weight kernel. Such et al. proposed a way to remedy that in [112]. They describe a filtering operation as

$$\mathbf{x_{out}} = \sum_c \mathbf{H}_c \mathbf{x}_c + b \tag{3.34}$$

where $b$ is a bias term and

$$\mathbf{H}_c = \sum_l h_l \mathbf{A}_l. \tag{3.35}$$

There, the adjacency matrices $\mathbf{A}_l$ are constructed in a way that encodes directionality. Here are some examples of how to achieve that. First, for the vectorized form of an image we map pixels from their 2D coordinates to a positive integer as follows. Consider a pixel with $(i, j)$ coordinates and the image width is $W$. Its position in vectorized form is

$$p = iW + j. \tag{3.36}$$

For two pixels, $p_1$ and $p_2$, the oriented adjacency matrices can be defined as follows.

$$\mathbf{A}{\uparrow}(p_1, p_2) = \begin{cases} 1, & \text{if } p_1 - W = p_2 \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{A}{\downarrow}(p_1, p_2) = \begin{cases} 1, & \text{if } p_1 + W = p_2 \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{A}{\leftarrow}(p_1, p_2) = \begin{cases} 1, & \text{if } p_1 - 1 = p_2 \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{A}{\rightarrow}(p_1, p_2) = \begin{cases} 1, & \text{if } p_1 + 1 = p_2 \\ 0, & \text{otherwise} \end{cases}$$

For diagonal orientations an example is

$$\mathbf{A}{\nwarrow}(p_1, p_2) = \begin{cases} 1, & \text{if } p_1 - W - 1 = p_2 \\ 0, & \text{otherwise.} \end{cases}$$

Similar matrices can be defined for the rest of the diagonal directions. From (3.34) we can see that the filtering of the graph can be written as matrix-vector multiplication of the vertices with a linear combination of the oriented adjacency matrices. These definitions naturally sacrifice the symmetry of these matrices to obtain the orientation. This is equivalent to considering variations of directed graphs on the same vertex set. Unfortunately, much less is known about the spectral properties of directed graphs since as we mentioned in the previous chapter there is no guarantee of diagonalizability of the adjacency/Laplacian matrices. Thus, it seems like one practically has to abandon the nice spectral properties of symmetric matrices to achieve directionality. The authors were able to demonstrate performance on image classification that is similar to regular CNNs and thus consider this a successful generalization of graph neural networks that incorporate orientation.

There have been various works that extend regular CNNs in ways that exploit various symmetries of data more efficiently [18] [23] [17].

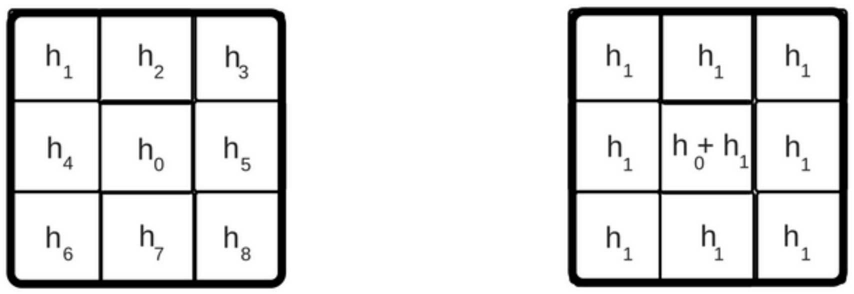


Figure 3.6: Left: Standard 2D convolution kernel. Right: Graph convolution with $h_0\mathbf{I} + h_1\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$.

An important distinction when it comes to classification tasks is graph classification versus node classification. The state-of-the-art in graph classification was currently dominated by a family of methods referred to as graph kernels, which compute the similarity between two graphs as the sum of the pairwise similarities between some of their substructures, and then pass the similarity matrix computed on the entire dataset to a kernel-based supervised algorithm such as the Support

Vector Machine for classification. Graph kernels mainly vary based on the substructures they use, which range from random walks and shortest paths to subgraphs, to name a few. Graph kernels have been very successful, but they suffer several limitations, among which is their high time complexity. For example, populating the kernel matrix requires computing the similarity between every two graphs in the training set, which can have a high computational cost. Training therefore becomes increasingly more expensive as the dataset gets bigger.

In [62], the task was formulated as a node classification problem. The graph convolution there filters the whole dataset with a fixed graph and the parameters that are adjusted are the fully connected layers that the graph convolution precedes. In some sense it can be considered a fully connected network equipped with a diffusion operator. On the other hand, in [85] the authors also dealt with image classification of the MNIST dataset as a graph classification problem, where each digit was represented by a graph of its superpixels. There, the graph was a trainable parameter of the framework as the weights of the patch operator in (3.29) are expressed

$$w_j(\mathbf{u}) = e^{-\frac{1}{2}(\mathbf{u}-\mu_\mathbf{j})\mathbf{\Sigma^{-1}}(\mathbf{u}-\mu_\mathbf{j}))} \tag{3.37}$$

with $\mathbf{\Sigma}$ and $\boldsymbol{\mu}$ being trainable parameters. In [74] et al. the authors work in a similar vein; they learn Laplacians for each data sample using the generalized Mahalanobis distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^\mathbf{T}\mathbf{\Sigma}(\mathbf{x}-\mathbf{y})} \tag{3.38}$$

with $\mathbf{\Sigma}$ a learnable parameter.

It should be clear by now that even though certain concepts like the graph convolution are at the core of graph neural network models, there is a lot of room for experimentation and design choices. Using an externally provided graph for the data or learning a graph in supervised/unsupervised ways can all be valid strategies and the best choice seems to largely be problem dependent. When additional information about the nature of the data is known (in the sense that e.g images are associated with grid graphs), then providing that additional information to the model can yield improved results.

# Chapter 4

# Batch and Dataset Convolutions

## 4.1 Our approach

Our overarching goal is to describe the building blocks that could be utilized for multiple levels of processing in a feedforward architecture. For some levels of processing, this has already been carried out to some extent in other works. The main contribution here is to propose graph convolution layers that filter input samples based on dataset information. This is done primarily at the batch level via a matrix of pairwise distances between data vectors in the batch, which can be further modified to include global information through what we call a reference structure of the dataset. Along with the already developed notion of convolutions on features, these building blocks are sufficient for a holistic graph approach to learning, since we can operate on three different levels:

- Feature level
- Batch level
- Dataset level

While most of the works in the literature focus on replacing the standard convolutional layers, our goal is to provide additional building blocks which can be combined with existing models to improve performance. In this work is to describe *batch convolutions* and *dataset convolutions* which use the vertex domain implementation of convolutions on graphs [21], for the purpose of incorporating additional information from the dataset in the training process of each sample. This is achieved by defining graphs on each training batch whose nodes are the training samples and then performing graph filtering. We combine this with regular convolutional and pooling layers to improve performance on image classification.

Based on these 3 levels of processing we mentioned, we propose the following graph convolutions:

- Feature convolutions
- Batch convolutions
- Dataset convolutions

This terminology was chosen because it presents the models concisely in terms of their core functionality and it also sets the context properly to discuss the overarching goal of these experiments. The structure is going to be the following. First, we are going to present the core properties that are shared among these different approaches. Next, we will describe each of these approaches separately and discuss experiments and implementation details for each one of them. Finally, we will conclude with some general observations and comments as well as discussion regarding future work and open questions.

### 4.1.1 Models

Our work is based on a graph spectral layer that applies a transform on each incoming minibatch. We define the weight matrix $\mathbf{W}$ where the weights $w_{i,j}$ represent the similarity between $i$ and $j$, as given by the standard exponential parametrized by a trainable $\sigma$ parameter

$$w_{i,j} = e^{\frac{||x_i - x_j||^2}{\sigma}}. \tag{4.1}$$

Let $\mathbf{L} = \mathbf{D} - \mathbf{W}$ be the Laplacian matrix of the graph, and $\mathbf{D}$ the diagonal degree matrix with entries $d_i = \sum_{j \in N(i)} w_{i,j}$, where $N(x)$ is the set of nodes in the immediate (1-hop) neighborhood of $x$.

For the symmetric normalized Laplacian $\mathbf{L_{sym}}$, we have

$$\mathbf{L_{sym}} = \mathbf{D^{-1/2} L D^{-1/2}} = \mathbf{I} - \mathbf{D^{-1/2} W D^{-1/2}} \tag{4.2}$$

Then, if for instance $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\mathbf{T}$ is the $N \times D$ minibatch matrix of $N$ samples and $D$ features, the operation

$$f(\mathbf{X}) = \mathbf{L_{sym} X} \tag{4.3}$$

can be written as in (3.14), i.e it is the graph convolution $\mathbf{x}_i *_G \boldsymbol{\lambda}$ of the data samples $\mathbf{x}_i$ in the minibatch with the eigenvalues of $\mathbf{L_{sym}}$.

This graph convolution acts as a linear spectral filter $f(\lambda) = 1 - \lambda$ of the eigenvalues of $\mathbf{D^{-1/2} W D^{-1/2}}$. To achieve a more general filter, we parametrize (4.3) as

$$f(\mathbf{X}) = (a\mathbf{I} + b\mathbf{D^{-1/2} W D^{-1/2}})\mathbf{X} \tag{4.4}$$

where $a, b$ are trainable parameters. This gives the eigenvalue filter the form of $f(\lambda) = a - b\lambda$.

### 4.1.2 Experimental Setup

The learning rate was set to 0.001 and all the models were trained for 40 epochs using the Adam algorithm [61] with fixed batch size of 60 samples. We noticed slightly better results but slower convergence with larger batch sizes at some experiments but 60 was the maximum that could run 'crash-free' on a 2GB graphics card. The coefficients $a, b$ in (4.4) were initialized as normal random variables in the way described in [34]. An approach that wo Many of the models we experimented with usually involve a combination of graph convolutional layers and regular convolutions.

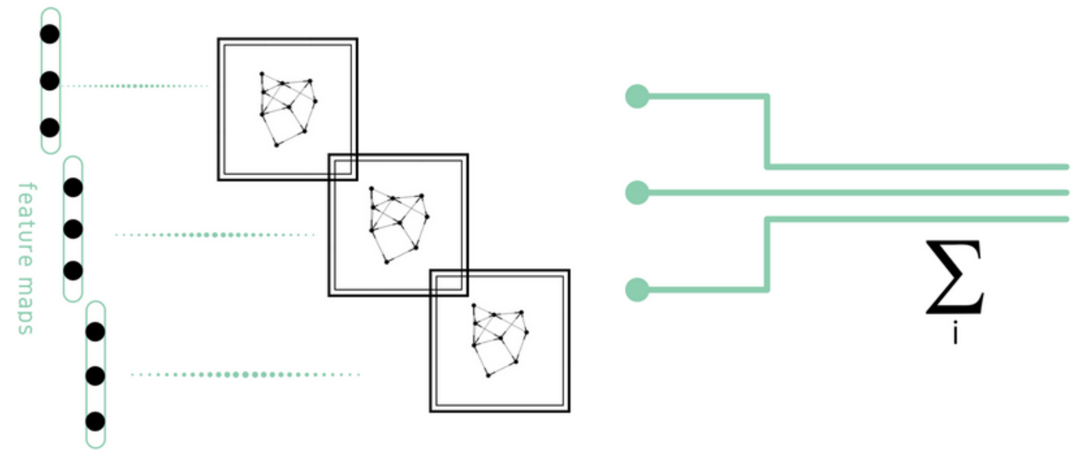


Figure 4.1: Schematic representation of the graph convolution layer. Each input feature map is graph convolved on graphs trained with different parameters $\sigma$ and then summed together to produce an output feature map.

### 4.1.3 Datasets

**MNIST** This is the well known MNIST dataset [70]. We are using a training set of 9600 digits in all of the cases, and testing on 10k others. The validation set also contains 9600 digits. These numbers are consistent throughout all experiments with all of the different models.
**Fashion MNIST** This is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples [126]. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is

essentially a harder version of MNIST that shares the same image size and structure of training and testing splits. We are using the same training and test size in this one as on regular MNIST.
**CIFAR10** The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [66]. Again we consider train,test and validation sets that contain 9600 samples each.

### 4.1.4 Feature Convolutions

The **feature convolution** (SC) layer is the convolution defined on the graph whose vertices correspond to an individual sample's features. For example inn the case of images, the nodes of the graph would be the pixel intensities. The function $f(\mathbf{X})$ in (4.4) becomes a filtering operation for the vectorized images. The model we describe here frames the image classification problem as graph classification. This means that for each image a new graph is formed based on the learned parameters and the graphs are constantly adjusted in each epoch as the parameters are updated. The model learns a set of parameters for the graph kernel which could be multiple for each input feature vector.

So, for a tensor $\mathbf{Y}$ of dimensions $M \times N \times D$ where $M$ is the number of feature maps, the filtering operation $f(\mathbf{X})$ is carried out by the tensor of dimensions $M \times N \times D \times D$. In other words, the filtering acts on the features (pixels) of each input image. Then, similar to regular to 2D convolutions, the filtered versions of the feature maps are summed to produce one output feature map. This can be thought of as a convolutional layer where the weights instead of being locally restricted by the predefined grid neighborhoods (kernel size), they determined by the exponential in 4.1 and each sample can be connected to any other without spatial restrictions. One could impose a similar restriction on the graph convolutions by adjusting the distance metric. This can be similar to the formulation of the bilateral filter in [31], which would mean that for our weights in 4.1,

$$w_{i,j} = e^{\frac{||x_i - x_j||^2}{\sigma_1}} e^{\frac{||p_i - p_j||^2}{\sigma_2}}. \tag{4.5}$$

where $p_i, p_j$ can be the coordinates of the pixels $i, j$ on the grid.

It quickly becomes apparent that this formulation of graph convolutions can have high memory costs, considering the fact that at the top level an image can have thousands of pixels, which will result in even larger adjacency/Laplacian matrices. Coupled with the fact that multiple such matrices are used per feature map in the BCS layer, the computational burden of certain experiments can become almost prohibitive. For that reason we were not able to adequately experiment with many of these models. Even with small images from the MNIST and fashionMNIST datasets training a very simple model without many output feature masks can take multiple hours.

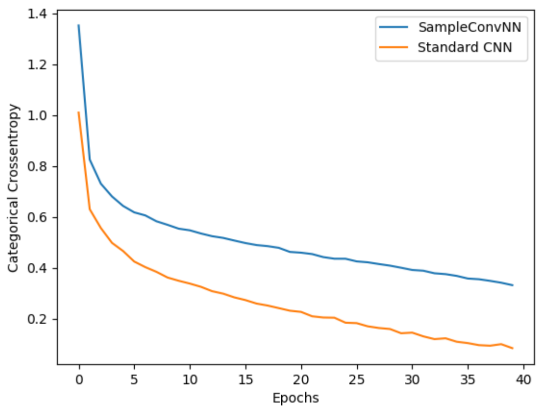


Figure 4.2: Feature convolution training loss convergence. SampleConvNN represents the network that implements feature convolutions.

To alleviate that we restricted our experiments to a downsampled version of MNIST. Essentially, an additional pooling layer is stacked on top of the model, and thus the image size is cut in half before the first filtering layer. Even with a setup like that, a model like the one presented in the diagram of figure 7 took more than a minute for each training epoch, which results in slightly less than an hour for each complete training session. Table 4.1 shows the results of an SC layer based model on FashionMNIST. The number next to *SC* indicates the number of output feature maps for that layer. For all the models that were tested, max-pooling layers were included between each convolution layer. As we can see in 4.2, the model converges slower than the CNN. Of course as we have already explained earlier in the parameter section, we would not expect this approach to outperform regular CNNs as it does not account for directionality.

| | FashionMNIST(%) |
|---|---|
| conv32-conv32-fc600 | 86.489 |
| SC_4-SC_16-fc600 | 83.199 |
| fc 600 | 61.012 |

Table 4.1: Feature convolution accuracy on FashionMNIST.

### 4.1.5 Batch Convolutions

In this layer, which we call **batch convolution** layer (BC layer) the graph is such that the nodes of the graph represent the feature vectors of the layer's input batch. In contrast to the feature convolution layer where the graph nodes represent the features The problem is framed as a node classification task, since each node represents a batch sample. For a tensor $\mathbf{Y}$ of dimensions $M \times N \times D$ where $M$ is the number of feature maps, the output of the layer is the matrix $\mathbf{Y}_{out}$ with the same dimensions as $\mathbf{Y}$, whose feature maps of dimension $N \times D$ have been filtered as it has been described in (4.4). The graph is constructed from the data so in total there are $M$ parameters from (4.1), plus $M$ pairs of $a, b$ coefficients for (4.4).

We also implement a different version of this approach which we call the Batch Convolution Sum (BCS) layer, $\mathbf{f}(\mathbf{X})$ is calculated in the same way, but this time the output is $N \times D \times M'$ tensor, where $M'$ is the number of desired featured maps in the output. The $j$th $N \times D$ output feature map is calculated as

$$\mathbf{Y}_{out}^{j} = \sum_{i}^{M} f(\mathbf{X}^{i}). \tag{4.6}$$

In other words, we learn graph filters for each feature map

| | FashionMNIST(%) | MNIST(%) | CIFAR10(%) |
|---|---|---|---|
| BC-conv32-conv32-fc600 | 86.843 | 98.593 | 61.850 |
| BC-conv32-BC-conv32-fc600 | 86.958 | 97.541 | 54.468 |
| BC-conv32-BC-conv32-BC-fc600 | 86.916 | 97.281 | - |
| BCS-conv32-conv32-fc600 | 87.385 | 98.677 | 38.906 |
| BCS-conv32-BCS-conv32-fc600 | 86.989 | 95.895 | - |
| BCS-conv32-BCS-conv32-BCS-fc600 | 86.739 | 96.885 | - |
| conv32-BCS-conv32-BCS-fc600 | - | - | 50.520 |
| conv32-conv32-fc600 | 86.489 | 98.545 | 61.770 |

Table 4.2: Accuracy of models on image classification. Some models were not tested on certain datasets due to the computational cost.

and then sum over all the filtered maps to produce the output feature map. This resembles the structure of a standard convolutional layer where regions of each input vector are convolved with a different kernel and summed across all input vectors.

For this layer, we have $3M$ (the same as the BC layer) parameters per output feature map. Here we have $M'$ output feature maps, thus the total number of parameters of the layer is $M'3M$. Both the BC and the BCS layers are combined with 2D convolution and max pooling layers followed by RELU non-linearities.

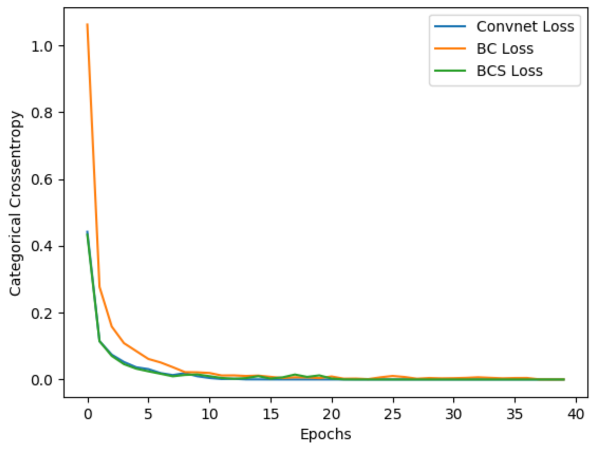


Figure 4.3: Comparison of BC and BCS loss during training with standard convolutions

A potential vulnerability of the batch layer has to do with the fact that any input batch can vary significantly and thus learning the graph metric (kernel parameters) on batches with high variability might not be the best strategy. It would be desirable to somehow standardize the distances in a way that makes learning the graph parameters easier.

### 4.1.6 Dataset Convolutions

The **dataset convolution** performs a graph convolution on the graph whose nodes are the feature vectors of the batch. This is similar to batch convolutions in the sense that we are again working on node classification and filtering the batches with a similarity matrix. The key difference between this approach and the models described in the previous section that utilize the batch convolution lies in they way we compute the weight matrix $\mathbf{W}$. The $N \times N$ weight matrix of the minibatch is

$$\mathbf{W}(i,j) = \sum_{k=1}^{L} \left( \sum_{l=1}^{L} z_{i,l} c_{l,k} \right) z_{k,j} \tag{4.7}$$

where $\mathbf{c_i} = [c_{i,1}, c_{i,2}, \dots, c_{i,L}]$ is the vector of distances from a point $i$ to the rest of the points in a set of reference points $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, \dots, \mathbf{y_L}]$, while $\mathbf{z_j} = [z_{i,1}, z_{i,2}, \dots, z_{i,L}]$ is the vector of distances from a point $i$ in the minibatch to the points $\mathbf{y_1}, \mathbf{y_2}, \dots, \mathbf{y_L}$.

We can express $\mathbf{W}$ more compactly with matrix notation. Let $\mathbf{Z} = [\mathbf{z_1}, \mathbf{z_2}, \dots, \mathbf{z_L}]^T$ and $\mathbf{C} = [\mathbf{c_1}, \mathbf{c_2}, \dots, \mathbf{c_L}]$ be $N \times L$ and $L \times L$ matrices respectively. Then

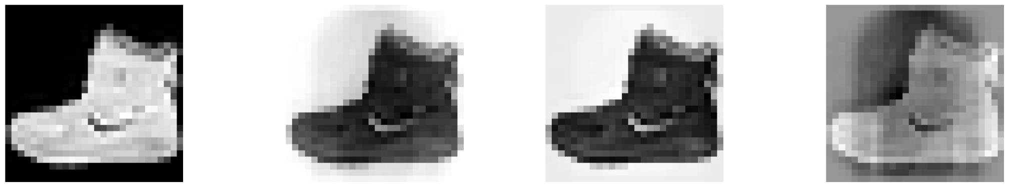


Figure 4.4: Feature maps produced by a BCS layer filtering directly the input images.

$$\mathbf{W} = \mathbf{Z}\mathbf{C}\mathbf{Z}^{\mathbf{T}}. \tag{4.8}$$

The matrix $\mathbf{L}$ is symmetric and so is $\mathbf{W}$. The entry $(i, j)$ of the matrix $\mathbf{W}$ can be interpreted as the distance traveled from vector $\mathbf{x_i}$ in the minibatch to vector $\mathbf{x_j}$ in the minibatch when passing through the reference points $\mathbf{Y}$. This is also easily generalizable to paths with length more than 1 in the reference graph. For distances through paths of length $k$ we can compute

$$\mathbf{W}_{(k)} = \mathbf{Z}\mathbf{C}^{k}\mathbf{Z}^{\mathbf{T}}. \tag{4.9}$$

That is, the distance between two samples in the batch passes through $k$-step paths in the reference structure (cluster centers).

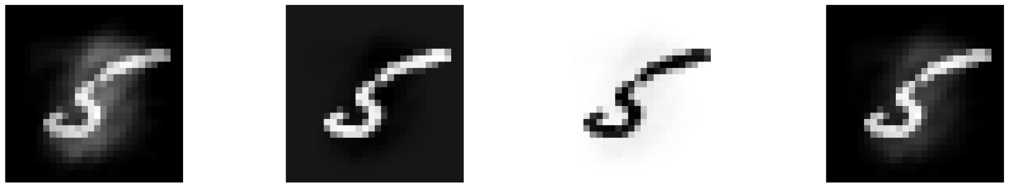


Figure 4.5: Feature maps produced by the DC layer operating directly on the input images.

Of course the obvious question here is what choice of reference points should be made. In our experiments, the reference points are selected from the cluster centers of the dataset. Specifically, for MNIST and fashionMNIST we first cluster the training set with the standard K-Means [58] clustering algorithm. Then we obtain a set of 50 cluster centers, 5 from each class and compute the distance matrix $\mathbf{C}$. During execution, $\mathbf{C}$ is passed onto the dataset convolution layer, while the matrix $\mathbf{Z}$ is calculated as each minibatch passes through the model. In our experiments we tested only $k = 1$ paths. Also, for the purpose of numerical stability, both $\mathbf{C}$ and $\mathbf{Z}$ were scaled by a factor of 0.01.

| | FashionMNIST(%) | MNIST(%) |
|---|---|---|
| DC_Conv32-Conv32-FC600 | 87.614 | 98.729 |
| Conv32-Conv32-FC600 | 86.489 | 98.545 |

Table 4.3: CNN augmented with DC layer vs regular CNN.

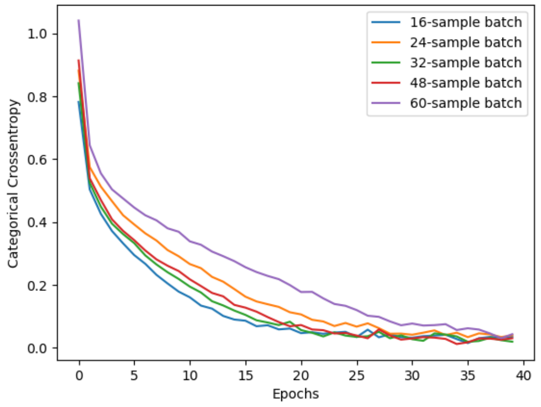


Figure 4.6: Loss over training epochs with different batch sizes for the BCS layer.

The dataset convolution layer seeks to rectify the problem of stability that we mentioned in the batch convolution, by using those 'proxy' points as a way to calibrate the distances between feature vectors in the batch. The dataset convolutions were able to obtain the best performance across all the models we tested. At this stage it is hard to tell if this performance boost justifies the extra burden of computing cluster centers for a whole dataset. On larger datasets the computational overhead could affect training times significantly.

### 4.1.7 Discussion and Results

The convergence rates for each batch size can be seen in Figure 4.6. The $\sigma$ parameters were intitialized to 0.01. We found that random initializations would make convergence during training more difficult. Figure 4.3 shows the convergence of the BC and BCS models compared to regulard convolutions. For the BCS layer, we fix the number of output feature maps on all experiments at 16. An interesting observation was that BCS layers between convolutional layers with less output feature masks than the preceeding convolution still achieve the same performance. That is, a Conv32-BCS4-Conv32 configuration did not have a significantly worse accuracy than a model with Conv32-BCS16-Conv32. Stacking BCS and BC layers can affect convergence drastically because the sample values can get quickly out of the [0,1] interval. In [62], the graph filter is renormalized after each subsequent application. An alternative solution was to use a batch normalization layer [53] in between BC/BCS layers, which was successful in maintaining the data on a reasonable range of values.

For our approach we combined different elements of the models we already described in the previous section. Namely, we use a 'diffusion-like' operator on the batch with a similar goal in mind as the one achieved in [62]; that is to push data samples that are close with each other even closer in order to achieve a more clustered representation. The difference is that the two parameters we use instead of the one in the paper by Kipf et al. allow for a sample to also subtract its neighbors from itself. Also, these layers are learning various distance metrics as is the case in [85] and [74]. Finally, the sum over all filtered feature maps resembles the filtering setup in [112] and equation (3.34).

For the layers that use a graph structure on batches, during testing time we predict test samples in batches from the test set. An alternative would be to predict each test sample using a batch of nearest neighbors or random samples from the training set. Assuming that the test and train set come from approximately the same distribution, then using samples from the training set does not really impact the error. Indeed, when testing like that we found no differences. For the nearest neighbor case, the implementation introduces additional computational costs and seems to perform slightly worse.

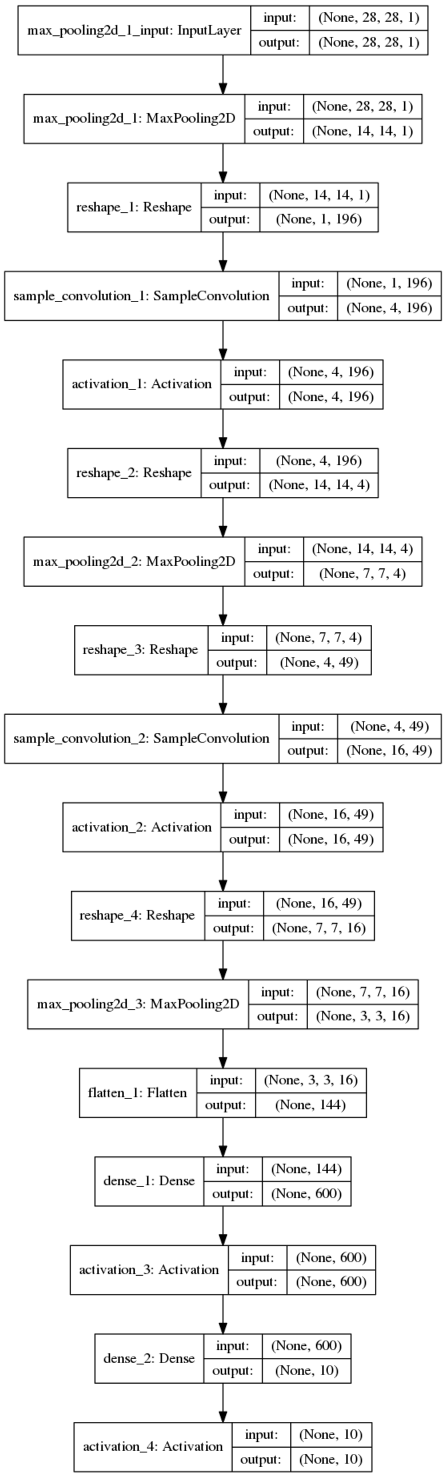
## 4.2 Conclusion and Future work

To conclude, we have presented three types of graph convolution layer, each forming a graph on different levels of the neural network pipeline. Variants of the feature convolution have been already discussed in previous works so the novelty in the approach presented here lies mostly in the other two layers. We were able to demonstrate that these models are indeed trainable and can be utilized to improve performance of standard layers. The performance boosts are marginal for now and only for certain models but they definitely present us with possible new directions that can be further explored. One of the main visions that we were not able to cover in this work experimentally, is the combination of all three of the layers that were presented in a unified architecture. We could not test these models consistently enough in order to present a clear picture as they are costly in terms of computation. Therefore, this is a potential avenue for more research and experimentation and is left as future work. Based on our current assessment of these models in terms of their performance, the number of parameters could also be one of the hindering factors. While it is tempting to consider models that have a minimal amount of parameters and mostly rely on the data to solve any given problem, in practice we often find that increasing the complexity of the model (this often manifests as increased depth on more common deep learning models) can yield significant performance improvements.

For the feature convolution as well as the broader class of spatial and spectral filtering networks, there is yet a lot of ground to cover as the field is still very young and thriving. We highlighted some potentially interesting connections between spectral estimation methods in numerical analysis and the currently popular approaches for localized filtering as well as some attempts to alleviate the drawbacks of symmetric matrices when it comes to recognizing orientation by considering matrices that correspond to specific directions on the grid. To add to that, new insights from training standard deep learning models can easily bleed into the graph neural network setting as there are a lot of common qualities between them. In fact, these models are at the intersection of many kinds of engineering and mathematical analysis which means that there is a lot of potential for novel insights and perspectives that will enrich the currently existing framework. On a more practical level there are already examples of this such as the node dropout presented in [8]. On a theoretical level we have mentioned the intriguing connections between (spectral) graph theory, physics and fields of mathematics like group and knot theory. Advances in the theory of data representation will also be invaluable for these models since they seem (at least for now) to require a fair amount of design choices and parameter tuning. Understanding what is the best setup that matches particular kinds of data is still a hard problem even among the most popular deep learning models, hence finding ways to streamline the process of model selection would make graph neural networks a more attractive alternative. Ultimately, it is easy to maintain a positive outlook as it seems that we have only begun to scratch the surface of what is possible.

# Appendices

## Model examples



max_pooling2d_1_input: InputLayer | input: (None, 28, 28, 1) | output: (None, 28, 28, 1)
max_pooling2d_1: MaxPooling2D | input: (None, 28, 28, 1) | output: (None, 14, 14, 1)
reshape_1: Reshape | input: (None, 14, 14, 1) | output: (None, 1, 196)
sample_convolution_1: SampleConvolution | input: (None, 1, 196) | output: (None, 4, 196)
activation_1: Activation | input: (None, 4, 196) | output: (None, 4, 196)
reshape_2: Reshape | input: (None, 4, 196) | output: (None, 14, 14, 4)
max_pooling2d_2: MaxPooling2D | input: (None, 14, 14, 4) | output: (None, 7, 7, 4)
reshape_3: Reshape | input: (None, 7, 7, 4) | output: (None, 4, 49)
sample_convolution_2: SampleConvolution | input: (None, 4, 49) | output: (None, 16, 49)
activation_2: Activation | input: (None, 16, 49) | output: (None, 16, 49)
reshape_4: Reshape | input: (None, 16, 49) | output: (None, 7, 7, 16)
max_pooling2d_3: MaxPooling2D | input: (None, 7, 7, 16) | output: (None, 3, 3, 16)
flatten_1: Flatten | input: (None, 3, 3, 16) | output: (None, 144)
dense_1: Dense | input: (None, 144) | output: (None, 600)
activation_3: Activation | input: (None, 600) | output: (None, 600)
dense_2: Dense | input: (None, 600) | output: (None, 10)
activation_4: Activation | input: (None, 10) | output: (None, 10)

Figure 7: Example of a feature convolution model

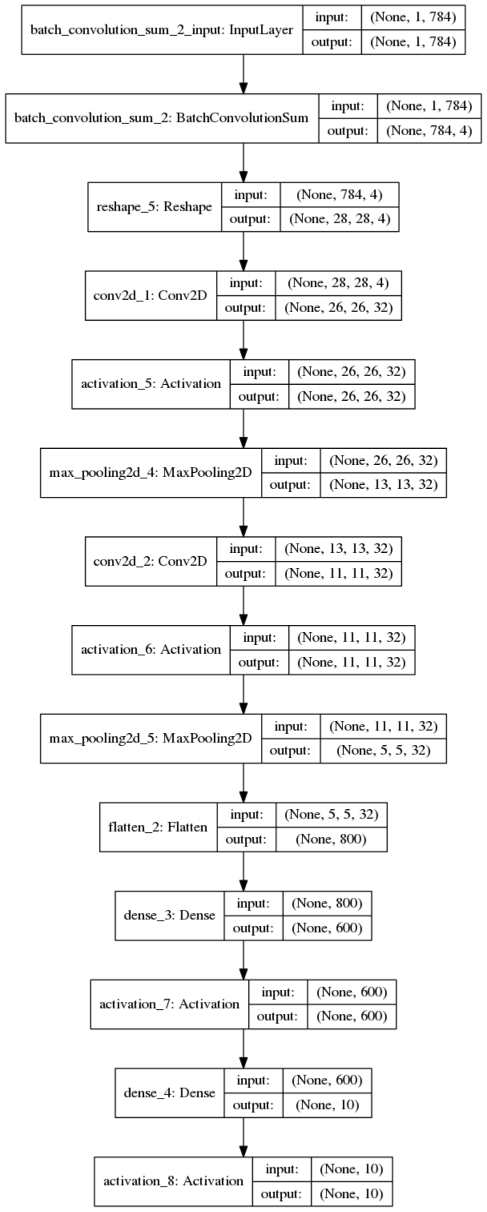


Figure 8: Example of a batch convolution model

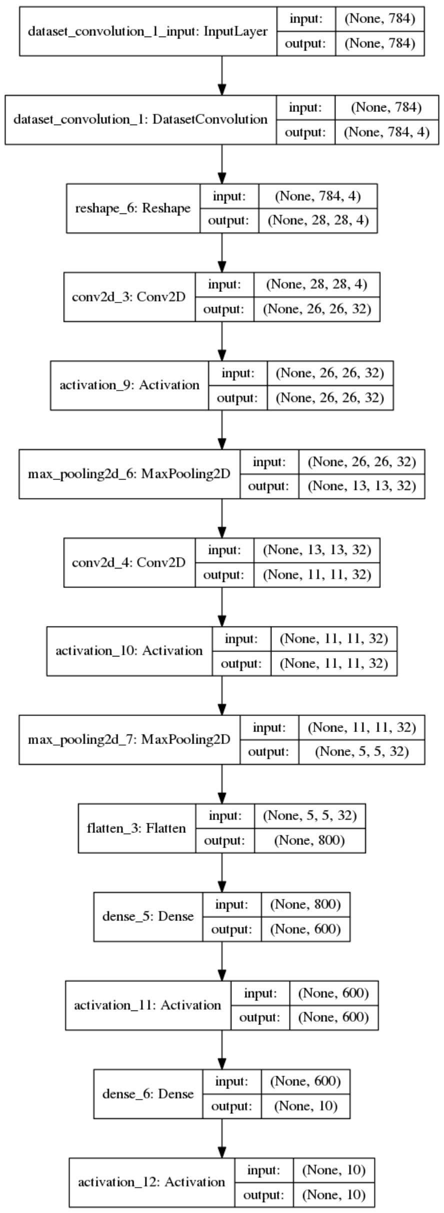

dataset_convolution_1_input: InputLayer
input: (None, 784)
output: (None, 784)
dataset_convolution_1: DatasetConvolution
input: (None, 784)
output: (None, 784, 4)
reshape_6: Reshape
input: (None, 784, 4)
output: (None, 28, 28, 4)
conv2d_3: Conv2D
input: (None, 28, 28, 4)
output: (None, 26, 26, 32)
activation_9: Activation
input: (None, 26, 26, 32)
output: (None, 26, 26, 32)
max_pooling2d_6: MaxPooling2D
input: (None, 26, 26, 32)
output: (None, 13, 13, 32)
conv2d_4: Conv2D
input: (None, 13, 13, 32)
output: (None, 11, 11, 32)
activation_10: Activation
input: (None, 11, 11, 32)
output: (None, 11, 11, 32)
max_pooling2d_7: MaxPooling2D
input: (None, 11, 11, 32)
output: (None, 5, 5, 32)
flatten_3: Flatten
input: (None, 5, 5, 32)
output: (None, 800)
dense_5: Dense
input: (None, 800)
output: (None, 600)
activation_11: Activation
input: (None, 600)
output: (None, 600)
dense_6: Dense
input: (None, 600)
output: (None, 10)
activation_12: Activation
input: (None, 10)
output: (None, 10)

Figure 9: Example of a dataset convolution model

# Index

# Bibliography

[1] Myoung An, Chao Lu, et al. *Mathematics of multidimensional Fourier transform algorithms.* Springer Science & Business Media, 2012.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*, 2017.

[4] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on riemannian manifolds. *Machine learning*, 56(1-3):209–239, 2004.

[5] Yoshua Bengio. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer, 2013.

[6] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[7] Yoshua Bengio, Pascal Vincent, Jean-François Paiement, O Delalleau, M Ouimet, and N LeRoux. Learning eigenfunctions of similarity: linking spectral clustering and kernel pca. Technical report, Technical Report 1232, Departement dInformatique et Recherche Oprationnelle, Universite de Montreal, 2003.

[8] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[9] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[10] Andries E Brouwer and Willem H Haemers. *Spectra of graphs.* Springer Science & Business Media, 2011.

[11] Joan Bruna and Xiang Li. Community detection with graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

[12] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[13] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[14] Fan Chung. Discrete isoperimetric inequalities. In *DMTCS*, pages 24–24, 1996.

[15] Fan RK Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[16] Taco Cohen. *Learning transformation groups and their invariants*. PhD thesis, Masters thesis, University of Amsterdam, 2013.

[17] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999, 2016.

[18] Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016.

[19] R Cameron Craddock, G Andrew James, Paul E Holtzheimer, Xiaoping P Hu, and Helen S Mayberg. A whole brain fmri atlas generated via spatially constrained spectral clustering. *Human brain mapping*, 33(8):1914–1928, 2012.

[20] Dragoš Cvetković and Slobodan K Simić. Towards a spectral theory of graphs based on the signless laplacian, i. *Publications de l'Institut Mathématique*, 85(99):19–33, 2009.

[21] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3837–3845, 2016.

[22] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11), 2007.

[23] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *arXiv preprint arXiv:1602.02660*, 2016.

[24] Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

[25] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[26] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[27] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[28] William T Freeman, Edward H Adelson, et al. The design and use of steerable filters. *IEEE Transactions on Pattern analysis and machine intelligence*, 13(9):891–906, 1991.

[29] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[30] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[31] Akshay Gadde, Sunil K Narang, and Antonio Ortega. Bilateral filter: Graph spectral interpretation and extensions. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 1222–1226. IEEE, 2013.

[32] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[33] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[34] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[35] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[36] Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[38] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 729–734. IEEE, 2005.

[39] Stephen Guattery and Gary L Miller. On the performance of spectral graph partitioning methods. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242. Society for Industrial and Applied Mathematics, 1995.

[40] Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

[41] Kenneth M Hall. An r-dimensional quadratic placement algorithm. *Management science*, 17(3):219–229, 1970.

[42] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[44] Donald Olding Hebb. *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[45] Matthias Hein, Jean-Yves Audibert, and Ulrike Von Luxburg. From graphs to manifolds–weak and strong pointwise consistency of graph laplacians. In *International Conference on Computational Learning Theory*, pages 470–485. Springer, 2005.

[46] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[47] Ivan Herman, Guy Melançon, and M Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

[48] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[49] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[51] Leslie Hogben. *Handbook of linear algebra*. CRC Press, 2006.

[52] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.

[53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[54] Alexey Grigorevich Ivakhnenko. The group method of data handling, a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1968.

[55] Vaughan Jones. On knot invariants related to some statistical mechanical models. *Pacific Journal of Mathematics*, 137(2):311–334, 1989.

[56] Mark Kac. Can one hear the shape of a drum? *The american mathematical monthly*, 73(4):1–23, 1966.

[57] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[58] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[59] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[60] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[61] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[63] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[64] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[65] Yehuda Koren. Drawing graphs by eigenvectors: theory and practice. *Computers & Mathematics with Applications*, 49(11):1867–1888, 2005.

[66] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[68] Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Distance metric learning using graph convolutional networks: Application to functional brain networks. *arXiv preprint arXiv:1703.02161*, 2017.

[69] Yann LeCun. Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85, Paris, France*. 1985.

[70] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[71] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[72] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[73] Bruno Lévy and Hao Richard Zhang. Spectral mesh processing. In *ACM SIGGRAPH 2010 Courses*, page 8. ACM, 2010.

[74] Ruoyu Li and Junzhou Huang. Learning graph while training: An evolving graph convolutional neural network. *arXiv preprint arXiv:1708.04675*, 2017.

[75] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017.

[76] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477, 2016.

[77] Rong Liu and Hao Zhang. Segmentation of 3d meshes through spectral clustering. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 298–305. IEEE, 2004.

[78] László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2:1–46, 1993.

[79] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[80] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *arXiv preprint arXiv:1704.06199*, 2017.

[81] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

[82] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[83] Pankaj Mehta and David J Schwab. An exact mapping between the variational renormalization group and deep learning. *arXiv preprint arXiv:1410.3831*, 2014.

[84] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[85] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016.

[86] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[87] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

[88] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrerro Moreno, Ben Glocker, and Daniel Rueckert. Spectral graph convolutions on population graphs for disease prediction. *arXiv preprint arXiv:1703.03020*, 2017.

[89] Arnab Paul and Suresh Venkatasubramanian. Why does deep learning work?-a perspective from group theory. *arXiv preprint arXiv:1412.6621*, 2014.

[90] Carsten Peterson and Bo Söderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(01):3–22, 1989.

[91] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM, 2009.

[92] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[93] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[94] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems: Revised Edition*. SIAM, 2011.

[95] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

[96] Aliaksei Sandryhaila and Jose MF Moura. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE Signal Processing Magazine*, 31(5):80–90, 2014.

[97] Hiroki Sayama. Estimation of laplacian spectra of direct and strong product graphs. *Discrete Applied Mathematics*, 205:160–170, 2016.

[98] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[99] Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 666–672. IEEE, 2005.

[100] Jürgen Schmidhuber. Learning algorithms for networks with internal and external feedback. In *IN DS TOURETZKY, JL ELMAN, TJ SEJNOWSKI, GE HINTON, PROC OF THE CONNECTIONIST MODELS SUMMER SCHOOL, PAGES 52-61. SAN MATEO, CA: MORGAN KAUFMANN, 1990.* Citeseer, 1990.

[101] Bernhard Schölkopf, Christopher JC Burges, and Alexander J Smola. *Advances in kernel methods: support vector learning*. MIT press, 1999.

[102] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[103] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[104] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[105] David I Shuman, Mohammadjavad Faraji, and Pierre Vandergheynst. Semi-supervised learning with spectral graph wavelets. In *Proceedings of the International Conference on Sampling Theory and Applications (SampTA)*, number EPFL-CONF-164765, 2011.

[106] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[107] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[108] Amit Singer. From graph to manifold laplacian: The convergence rate. *Applied and Computational Harmonic Analysis*, 21(1):128–134, 2006.

[109] Daniel A Spielman. Spectral graph theory and its applications. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 29–38. IEEE, 2007.

[110] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[111] Martin Storath, Andreas Weinmann, and Laurent Demaret. Jump-sparse and sparse recovery using potts functionals. *IEEE Transactions on Signal Processing*, 62(14):3654–3666, 2014.

[112] Felipe Petroski Such, Shagan Sah, Miguel Dominguez, Suhas Pillai, Chao Zhang, Andrew Michael, Nathan Cahill, and Raymond Ptucha. Robust spatial filtering with graph convolutional neural networks. *arXiv preprint arXiv:1703.00792*, 2017.

[113] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM, 1995.

[114] Gabriel Taubin, Tong Zhang, and Gene Golub. Optimal surface smoothing as filter design. *Computer VisionECCV'96*, pages 283–292, 1996.

[115] G TaubinÝ. Geometric signal processing on polygonal meshes. 2000.

[116] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.

[117] Edwin R Van Dam and Willem H Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003.

[118] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[119] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

[120] Jean-Charles Vialatte, Vincent Gripon, and Grégoire Mercier. Generalizing the convolution operator to extend cnns to irregular domains. *arXiv preprint arXiv:1606.01166*, 2016.

[121] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[122] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *Symposium on Geometry processing*, pages 33–37, 2007.

[123] Dominic JA Welsh and Criel Merino. The potts model and the tutte polynomial. *Journal of Mathematical Physics*, 41(3):1127–1152, 2000.

[124] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[125] Jiann-Ming Wu and Shih-Jang Chiu. Independent component analysis using potts models. *IEEE Transactions on Neural Networks*, 12(2):202–211, 2001.

[126] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[127] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.