Manifold Learning with Tensorial Network Laplacians

A thesis

presented to

the faculty of the Department of Mathematics

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Mathematical Sciences

by

Scott Sanders

May 2021

ABSTRACT

Manifold Learning with Tensorial Network Laplacians

by

Scott Sanders

The interdisciplinary field of machine learning studies algorithms in which functionality is dependent on data sets. This data is often treated as a matrix, and a variety of mathematical methods have been developed to glean information from this data structure such as matrix decomposition. The Laplacian matrix, for example, is commonly used to reconstruct networks, and the eigenpairs of this matrix are used in matrix decomposition. Moreover, concepts such as SVD matrix factorization are closely connected to manifold learning, a subfield of machine learning that assumes the observed data lie on a low-dimensional manifold embedded in a higher-dimensional space. Since many data sets have natural higher dimensions, tensor methods are being developed to deal with big data more efficiently. This thesis builds on these ideas by exploring how matrix methods can be extended to data presented as tensors rather than simply as ordinary vectors.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION AND FOUNDATIONAL CONCEPTS

We begin by exploring the Laplacian operator, tensors and manifolds in a mathematical context and then relate these topics to current concepts in machine learning.

1.1 Laplacian Operator

The Laplacian operator is a second-order differential operator given by the divergence of the gradient of a function on Euclidean space [27]. Denoted by Δ, the Laplacian of a twice-differentiable function f is

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$$

where ∇f is the *total derivative* of f, which is often called the gradient operator.

Definition 1.1 (Laplace Operator) *[27] The Laplace Operator is a differential operator representing the divergence of the gradient of a function, where the divergence of a vector field is a vector operator that produces a scalar and is the sum of partial derivatives of f.*

For a function $f : \mathbb{R}^n \to \mathbb{R}$, the Laplacian of f can be written as the sum of partial second derivatives:

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}.$$

The operator is named after Pierre-Simon de Laplace, a mathematician, physicist and astronomer who developed the concept in order to apply Newtonian gravitation to the entire solar system [4]. Because his work laid the foundation for the scientific study of heat, electricity and magnetism in addition to proving the stability of the universe, the equation $\Delta f = 0$ is now known as Laplace's equation and is a partial differential equation representing a variety of physical systems [4].

1.2 Tensors

The term *tensor* was originally derived from the Latin word *tensus* which means tension or stress [20] as its early use was to represent tensile forces in mechanical systems. Tensors as "generalizations of matrices to higher dimensions" [15] are popular in the fields of engineering and physics. More recently, tensors have been used in machine learning to provide "structured representation of data format and ability to reduce the complexity of multidimensional arrays" [7]. Despite being intimidating for many, a solid theoretical foundation will be helpful in demystifying tensors.

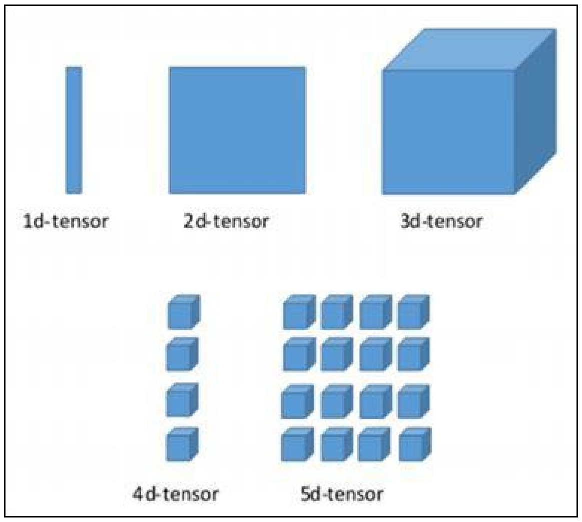

Figure 1: Visualizing tensors of different dimensions [15].

The *order* of a tensor is the number of its dimensions [15]. Scalars can be thought of as 0-order tensors as a scalar can be referenced without the use of an index. Vectors are first-order tensors since coefficients can be identified using one index, and matrices are second-order tensors since a coefficient is referenced by its row and column position. Tensors may also have three or more dimensions. Such tensors are difficult to visualize because working in four or more dimensions is not always intuitive, but tensors and their notation are useful in making sense of higher-dimensional space.

Definition 1.2 (Tensor) *[2]* $\mathbb{R}^{I_1 \times I_2 \times \ldots \times I_n}$ *is the vector space of multidimensional arrays with* n *indices. An element of this space is called a tensor.*

If each $I_j = m$ for all $j = 1, \ldots, n$, then tensor $\mathcal{X}$ has n indices and m^n coefficients. Furthermore, we recognize tensors as both the structure of an n-dimensional array and a mathematical algebra for manipulating such structures. We often write tensors as

$$\mathcal{T}_k^\ell = a_{i_1,\ldots,i_k}^{i_1,\ldots,i_\ell}.$$

For example, $\mathcal{T}_0^0$ is a scalar, $\mathcal{T}_1^0$ is a row vector and $\mathcal{T}_0^1$ is a column vector. The order of a tensor is $\ell + k$ where ℓ is the number of upper indices and k is the number of lower indices. The upper indices are referred to as the *contravariant* indices, and the lower indices are referred to as the *covariant* indices. Tensors with both covariant and contravariant indices are sometimes referred to as mixed type [20]. Furthermore, the rank R of a tensor is the number of minimum first order tensors necessary to reproduce the tensor [16].

We define the *outer product* as a function that takes two vectors as input and outputs a tensor denoted by $\mathbf{a} \circledcirc \mathbf{b}$ for vectors $\mathbf{a}$ and $\mathbf{b}$. For example, if $\mathbf{a}$ and $\mathbf{b}$ are in the column space $\mathbb{R}^n$ then $\mathbf{a} \circledcirc \mathbf{b} = \mathbf{a}\mathbf{b}^T$. It is important to note that tensors can be thought of as a sum of outer products [15]:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circledcirc \mathbf{a}_r^{(2)} \circledcirc \cdots \circledcirc \mathbf{a}_r^{(N)}.$$

We can break up this notation into smaller units.

Definition 1.3 (Dyad) *[20] Dyads are order 1 and rank 2 tensors formed by combining two vectors with the outer product.*

For instance, $\mathbf{b}_i \circledcirc \mathbf{c}_j$ is a dyad. Hence, linear combinations of dyads can be used to represent tensors.

Tensors can be reordered to adjust dimensions for a variety of reasons. *Vectorization*, for instance, maps a given matrix $X \in \mathbb{R}^{I \times J}$ into a vector by stacking its columns vertically. Analogously, *matricization* reshapes a tensor into a matrix [15]. A variety of methods exist for this process as discussed in section 1.4 of this chapter.

1.3 Basic Topology and Manifolds

Topology is defined in [1] as the the branch of mathematics dealing with qualitative geometric information, including connectivity information. Despite being quite abstract, topology has fascinating applications across the field of mathematics such as modeling high dimensional data structures.

Definition 1.4 (Topological Space) *[8] A topological space consists of some set X and open subsets T that that satisfy four conditions:*

1. *The empty set is in T.*

2. *X is in T.*

3. The intersection of a finite number of sets in T is also in T.

4. The union of an arbitrary number of sets in T is also in T.

Topological spaces are the most general type of mathematical space, and many other spaces can be constructed by adding additional properties. Euclidian space, for instance, is the set of all n-tuples (finite lists of length n) of real numbers $(x_1, x_2, \ldots, x_n)$ and is any nonnegative integer dimension $\mathbb{R}^n$. We can define a space that is central to this thesis.

Definition 1.5 (Manifold) *[8] A manifold is a topological space that locally resembles Euclidean space.*

For example, the earth is spherical, but, locally, the surface of the earth is flat and operations can be done on a simple two-coordinate system.

Manifold's often have nice properties. Central to some machine learning techniques is *the manifold hypothesis* which states that real-world high dimensional data (such as images) lie on low-dimensional manifolds embedded in a high-dimensional space [8] where *embedding* is the representation of topological objects in different spaces while preserving connective and algebraic properties.

Additional topological vocabulary is needed to understand manifolds. A *chart* on a smooth manifold is a 1-1 infinitely differentiable mapping from an open set on the manifold to a local coordinate system in a Euclidian space. If the domains of two charts overlap, then the charts are *compatible* if the the composition of one chart with the inverse of other chart is an infinitely differentiable map from Euclidean space to itself. Charts with domains that do not overlap are also considered compatible. An *atlas* is a collection of mutually compatible charts that cover a manifold. A smooth manifold is a topological space with an atlas of smooth, compatible charts.

A smooth manifold with a metric tensor (a function that defines distance between points in a space) is called a *Riemannian* manifold. With inner products varying smoothly between points, Riemannian manifolds have many useful properties [8]. An example of a Riemannian manifold is a *torus*, denoted $\mathbb{T}^n$. If it is a surface of genus one, then it possesses a single hole. Figure 2, illustrates some examples of manifolds in $\mathbb{R}^3$.

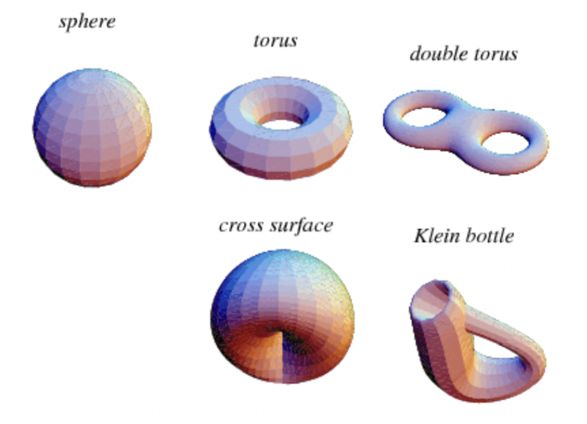

Figure 2: Examples of manifolds [8].

Keng emphasizes in [8] that "manifolds are all about mappings. Mapping from the manifold to a local coordinate system in Euclidean space using a chart; mapping from one local coordinate system to another coordinate system; and mapping a curve or function on a manifold to local coordinates."

1.4 Graphs and The Laplacian Matrix

In mathematics, a *graph* is a collection of vertices and connections among some (possibly empty) subset of those vertices [26]. We denote a graph G as

$$G = (V, E)$$

where V is a set of vertices and E is a collection of edges between those vertices. Two vertices are *adjacent* if they are connected by an edge, and the *degree of a vertex* refers to the number of vertices adjacent to a given vertex. There exists a "remarkable interplay between graphs and manifolds" [13] that forms a conceptual basis for manifold learning techniques that allow machine learning algorithms to take advantage of a data set's inherent geometric structure. This interplay between graphs and manifolds appears in the Laplacian operator's ability to represent structural information from both types of objects. Indeed, the Laplacian can be used to model a graph as a discrete form of a manifold [13]. Using this bridge to develop new learning methods is a growing field of research [13].

The Laplacian operator has many forms including a matrix form known as the *Network Laplacian.* This discrete operator measures "to what extent a graph differs at one vertex from its values at nearby vertices [28]." The network laplacian is a matrix that can be defined using two other matrices: the adjacency matrix and degree matrix of a graph. We begin by defining the adjacency matrix of a graph.

Definition 1.6 (Adjacency Matrix) *[28] For graph $G(V, E)$ with $|V| = n$ and*

$e_1, e_2, \ldots, e_m \in E$, the adjacency matrix, A, is an $n \times n$ square matrix where

$$a_{i,j} = \begin{cases} 1 & \textit{if edges } e_i \textit{ and } e_j \textit{ are adjacent} \\ \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

The adjacency matrix is composed of 1's and 0's, a 1 indicating whether the vertices in the graph are connected. Next, we define the degree matrix.

Definition 1.7 (Degree Matrix) *[28] For graph $G(V, E)$ with vertex set $|V| = n$ and edge set E, the degree matrix, D, is an $n \times n$ square matrix where*

$$d_{i,j} = \begin{cases} \deg(v_i) & \textit{if } i = j \\ \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

The degree matrix is a diagonal matrix of vertex degrees and represents the connectedness of each node. We can now define the Laplacian matrix.

Definition 1.8 (Network Laplacian) *[28] Given an unweighted, undirected graph $G = (V, E)$ with vertices $|V| = n$ and edges E, the $n \times n$ Laplacian Matrix is*

$$L = D - A$$

The Network Laplacian is both a linear transformation and a matrix itself, so it can be classified as a *matrix operator*. This matrix includes information about each vertex, the connectivity of that vertex, and the strength of those connections. This structure appears in many machine learning applications with a particular example being image gradient blending as seen in [14]. Images as data sets are grids of discrete pixel intensities, and discrete matrix operations on image data allow for editing and efficient storage.

1.5 Meshes and The Laplace-Beltrami Operator

A *polygon mesh* is a set of vertices, edges, and faces that approximates more complex geometric objects. As seen in Figure 3, meshes are important for creating and manipulating computer graphics. Meshes with many points are difficult to edit,

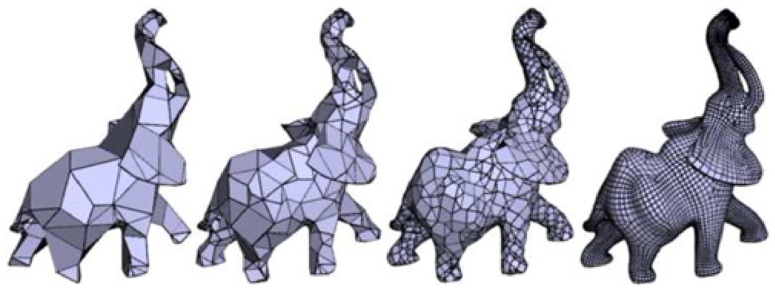

Figure 3: Mesh approximations of an element with various levels of complexity [11].

but mesh reconstruction done with Laplacian methods enhances editing ability [19]. *Differential coordinates* reduce global coordinates to an individual coordinate system at each point in the mesh. Furthermore, these coordinates are helpful for describing points relative to their neighbors [10] and are defined using the Laplacian on a manifold.

The *Laplace-Beltrami operator* is a generalization of the Laplace operator to functions defined on a Riemannian manifold. When applied to each vertex on a mesh/-manifold, the resulting vector represents the difference between the vertex and the vertices in a small neighborhood [19]. Using those vectors, one can reconstruct the mesh up to a rigid transformation [19]. Note that every smooth manifold has a Laplace–Beltrami operator that can be used to study the manifold itself.

1.6 Other Matrix Methods

This section explores matrix operations and their current uses in machine learning.

Spectral Clustering is a widely used technique for exploratory data analysis that uses eigenvalues and eigenvectors of a network Laplacian [22]. The goal is to group individuals on the basis of connectivity instead of compactness as illustrated in Figure 4. Such separation is possible if a set of data is sampled from a topological space that

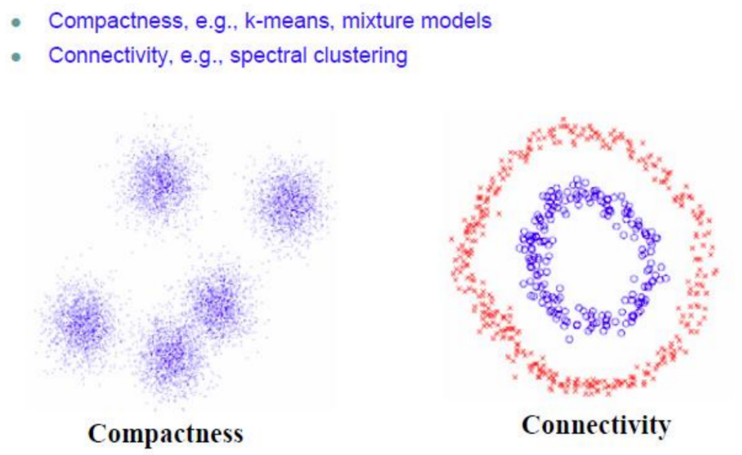

Figure 4: Compactness versus connectivity [22].

can be modeled by an appropriately constructed network, and often this topology is in the form of a manifold. The fundamental idea for this method comes from the Spectral Theorem:

Theorem 1.9 (The Spectral Theorem) *[22] Any symmetric matrix S has the form $S = Q\Lambda Q^T$.*

- *Q is an orthogonal matrix composed of the eigenvectors of S.*
- *Λ is the diagonal matrix of corresponding eigenvalues.*

The Spectral Theorem works on square matrices by decomposing them into eigenvector components and eigenvalues which represent their relative importance or weight. Since most matrices are not square, the Singular Value Decomposition is often used:

Theorem 1.10 (Singular Value Decomposition) *[32] Consider an $m \times n$ real matrix A with rank r. Matrix A can be written in* ***singular value decomposition*** *form as*

$$A = U\Sigma V^* = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \ s.t\ \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r.$$

Additionally,

$$A^*A = V\Sigma^2 V^* \text{ and } AA^* = U\Sigma^2 U^*.$$

In practice, the decomposition of a Network Laplacian yields insights into the structure of a network. In particular, spectral clustering can identify areas of low connectivity in a network and identify the boundaries of different groups [22]. In fact, these boundaries are found with a minimization problem using the Fiedler eigenvector, the normalized eigenvector of the second smallest eigenvalue of a Laplacian graph matrix [9]. This clustering method is relatively simple and more powerful than other clustering algorithms such a k-means clustering [22].

Diagonal matrices are special matrices where all coefficients other than those on the main diagonal are zeros. They have useful properties such as easy determinant calculations and letting one raise matrices to powers. The *diagonalization* of a matrix M, if it exists, is a diagonal matrix D for which there is an invertible matrix P such that $D = P^{-1}MP$. A diagonalization retains some properties of the original matrix [29]. This useful tool is equivalent to finding the eigenvalues of a matrix [29]. Matrix decomposition often involves diagonalization.

An $n \times n$ matrix A is *positive definite* if

$$\mathbf{x}^T A\mathbf{x} > 0$$

for all nonzero vectors $\mathbf{x} \in \mathbb{R}^n$ [6]. Furthermore, a positive definite matrix A is symmetric and has positive eigenvalues. Positive definite matrices also have applications in machine learning. *Convexity* is an important condition in optimization problems in which the value of the artithmetic mean of any two points on the function is greater than every value of the function within the domain of those two points [24]. Positive definiteness implies convexity [6].

Principle Component Analysis (PCA) is a statistical technique that reduces the dimensionality of datasets by creating new uncorrelated variables that successively maximize variance. Finding such new variables, the principal components, reduces to solving an eigenvalue/eigenvector problem, and the new variables are defined by the dataset at hand [21]. PCA is useful technique for data exploration that has not yet been completely generalized to tensors.

The Moore-Penrose Pseudo Inverse is important because the problem of finding the inverse of a matrix is *ill-conditioned*, which means that small changes in a coefficient of a matrix can cause large differences in the inverse of a matrix. Fortunately, a well-conditioned pseudoinverse can always be constructed, even for non-square matrix.

Definition 1.11 (Moore-Penrose Pseudo Inverse) *[30] Let A be an $m \times n$ matrix. The pseudo inverse, denoted A^+, is an $m \times n$ matrix that satisfies*

1. $AA^+A = A$

2. $A^+AA^+ = A^+$

3. $(AA^+)^T = AA^+$

4. $(A^+A)^T = A^+A.$

It can be shown that the Moore-Penrose pseudoinverse of a matrix always exists. In particular, the pseudoinverse can be applied to ill-conditioned linear systems. This is known as multilinear regression and is a common practice in the field of statistics.

The selection of matrix methods mentioned in this chapter is only a small fraction of what can be done using the algebraic properties of matrices and vectors.

1.7 Poisson Equation

Poisson's equation is an elliptic partial differential equation and is a generalization of Laplace's equation. The equation is

$$\Delta\phi = f$$

where Δ is the Laplace operator and ϕ and f are functions on a manifold [14]. Furthermore, f can be thought of as the divergence of a gradient of some function. Applications of Poisson's equation are most often found in physics, but the equation appears many times in the realms of mathematics and computing as well. One method explored in section 1.8 is using Poisson's equation is gradient mixing for image blending [34]. Poisson's equation is another example of operations defined on tensors.

For example, many image-blending problems are based on Poisson equations whose solutions are estimated using matrix methods. To begin, the continuous version of

Poisson's equation above has discrete counterpart of

$$L\mathbf{x} = f$$

where L is a positive definite matrix. For example, the objective is to insert the image of the hot air balloon into the picture of the empty parking lot in such a way that the alteration cannot be detected. Problems like this are common in the areas of computer vision and photo editing [34]. For this example, images have shape (700,700,3) meaning they have a height and width of 700 pixels along with three separate layers of color values (RGB).

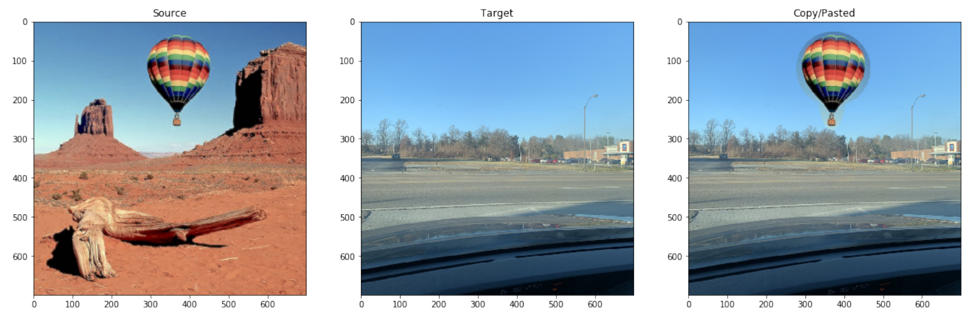

Figure 5: Balloon copy-pasted into new background without any blending.

Images are made up of pixels, each with a specific color value. Because of this inherent structure, matrices and tensors are often used to represent images in computing. Given a pixel (x, y) with intensity value I, the Laplacian, $L(x, y)$, is defined as

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}.$$

Using the Python libraries *pytorch* and *kornia*, the source and target images are converted to tensors. Convolution is an integral

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

where function f is modified by another function g, blending the two functions in a sense [23]. A *Laplacian filter* is the discrete second order derivative of convolution and is commonly used for edge detection [12] and is shown in Figure 6. A kernel is a

0	-1	0
-1	4	-1
0	-1	0

Figure 6: Laplacian filter.

matrix of weights which are multiplied with the input to extract relevant features [5]. An image can be thought of a discrete grid of color values, and the process shown in Figure 7 illustrates a kernel operating on an pixel.

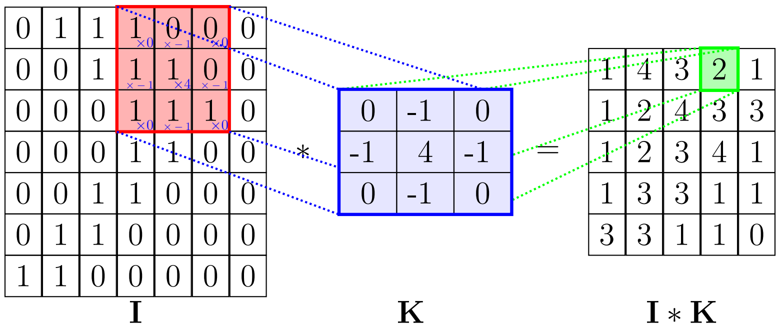

Figure 7: Kernel matrix operating on an image.

This filter smooths the given tensors with a Laplacian kernel, by convolving it to each channel [17]. Figure 8 shows the output (gradient) of kornia's Laplacian filter after it is converted back to an image matrix where differences in color value intensities are clearly visible.

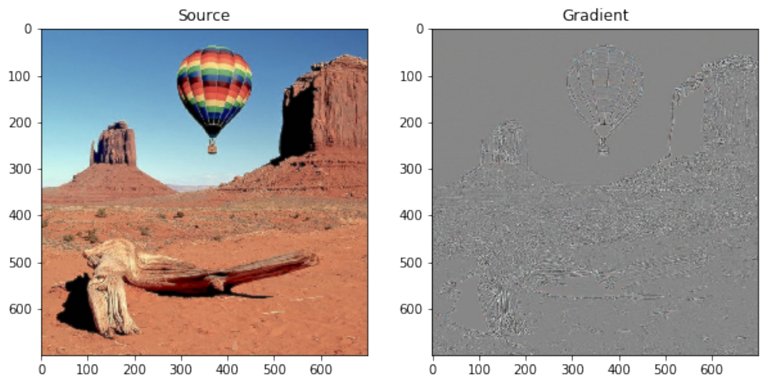

Figure 8: Color-based gradient of balloon image.

The two gradient images are combined using a predefined black and white mask image. We now have a gray scale representation of the change in color values for the blended image, but the color matrices have not yet been solved. Hence, we define a function to create a discrete Laplacian Matrix of a given size with *scipy*, a Python library built on another library, numpy, that defines numerical arrays. Note that an input of $m \times n$ where m is the number of rows and n the number of columns, results in a matrix of size $(mn) \times (mn)$.

The Poisson equation is

$$\Delta f = \text{div } \mathbf{v}$$

where $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ is the Laplacian and div $\mathbf{v} = \text{div } \nabla g = \text{div } (u, v) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of the gradient. Hence,

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

and this leads us to the linear system where the Laplacian Matrix is L and the blended Laplacian is B. Matrix B is flattened, meaning it is transformed from size $m \times n$ to a vector of size $(mn) \times 1$. We use spsolve, a Python library for solving sparse linear systems, to get a solution to the sparse linear system

$$L\mathbf{x} = \mathbf{b}.$$

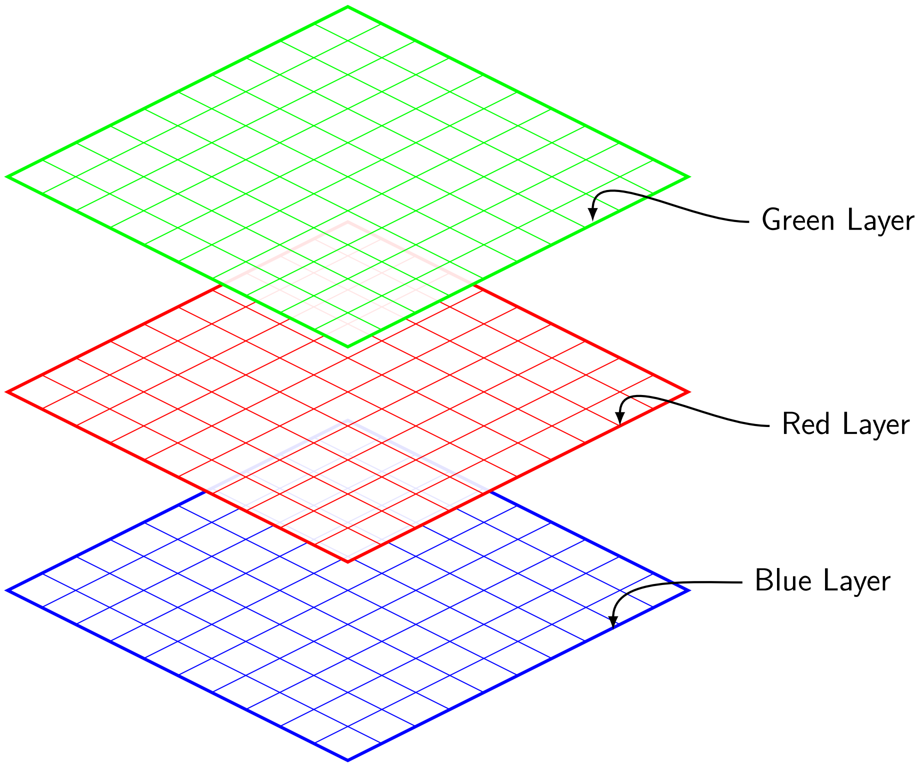

Figure 9: Three color layers of an image.

The idea behind this function is to solve the color equation for each of the three layers of the image, treating each layer as a unique matrix as seen in Figure 9. Once all three levels are solved, we can recombine them using numpy. Finally, we see the result in Figure 10: an image transferred from the source to the target in a way that the shape (gradient) of the balloon is maintained, but the colors have been adjusted to its new surroundings.

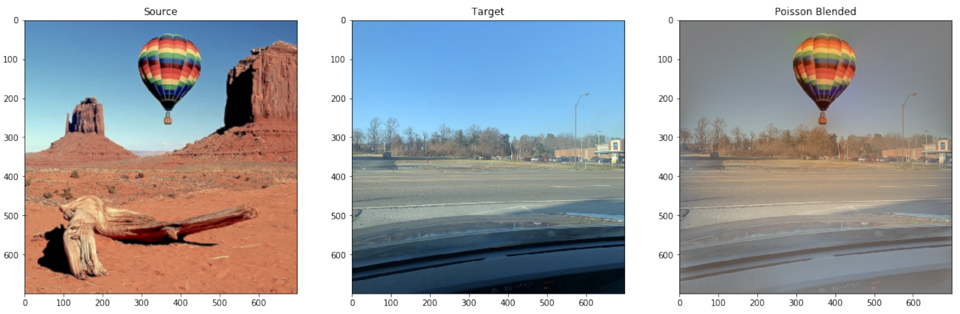

Figure 10: Balloon inserted into new background with gradient blending.

While not perfect, the image blended using the Poisson equation looks much more authentic than the first attempt to transfer the image. The code for this project gives anyone a powerful and relatively simple method to create deceptively convincing, photo-shopped images and videos that are often referred to as “deep-fakes.” Still, issues arise when color or texture differences between the source and target images are very large. This shows how ideas are often developed in a continuous sense but implemented with discrete methods and linear algebra.

1.8 Problem Statement

The Laplacian is often defined in terms of tensors, but we use matrix methods to study it. The central focus of this thesis will be extending matrix ideas to tensor concepts for the Laplacian. We seek to develop methods for dealing with vector-valued graphs and implement an example with image blending treating the image as a tensor instead of multiple matrices.

To address this problem, we explore tensors and their properties in Chapter 2. We investigate methods of representing and combining tensors in addition to defining a vector space and building on that definition of a tensor space. Also, we develop the concept of a vector valued function space and will show that linear transformations on certain vector spaces can be represented as tensors.

In chapter 3, we continue to motivate the tensor problem by discussing the least squares problem and how it looks in a tensorial context. We attempt to solve a tensor decomposition problem with a feature of random walks on a network and seek a method of calculating eigentensors. We then apply these methods to the image blending problem from this section.

2 IN-DEPTH TENSOR DESCRIPTION

2.1 Tensor Operations

Tensors can be combined using the *direct tensor product*. To illustrate this method, consider two tensors $\mathcal{A} = [a_{ij}^{k}] \in \mathcal{V}_2^1$ and $\mathcal{B} = [b_k^{vw}] \in \mathcal{V}_1^2$. We multiply element-wise using the outer product [15]:

$$\mathcal{A} \otimes \mathcal{B} = \sum_k a_{ij}^{k} b_k^{vw} = \mathcal{C} = [c_{ij}^{vw}] \in \mathcal{V}_2^2.$$

This is one method of combining two tensors to get a third tensor. The k index cancels as the two tensors are combined. This notation using covariant and contravariant indices and summing over identical entries when combined is called *Einstein Summation* notation, but it is not the only way to represent tensors. In particular, we define a tensorial method of combining two matrices that differs from traditional matrix multiplication.

Definition 2.1 (Kronecker Product) *The Kronecker Product between two arbitrarily sized matrices* $\mathbf{A} \in \mathbb{R}^{R\times J}$ *and* $\mathbf{B} \in \mathbb{R}^{K\times L}$ *is defined*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}.$$

The Kronecker product is a generalization of the outer product of two vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively. Tensor notation and the notation for the Kronecker product are often both denoted $\otimes$, but the tensor product is defined on tensors while the Kronecker product is defined on matrices.

Some basic properties of the Kronecker product are discussed in [18]. The Kronecker product of an $p \times q$ matrix and a $r \times s$ matrix is dimension $(pr) \times (qs)$. A property known since the 1890's is that the Kronecker product of identity matrices I_n and I_m is the identity matrix [18]

$$I_n \otimes I_m = I_{nm}.$$

The homogeneity of the Kronecker product (KRON 1 in [18])

$$(\alpha A) \otimes B = A \otimes (\alpha B) = \alpha(A \otimes B)$$

for scalar α and matrices A and B implies that the placement of multiplication with a scalar does not matter. The relation (KRON 7 in [18])

$$(A \otimes B)(C \otimes D) = (AC) \otimes (DC)$$

for matrices A, B, C and D shows that the product of two Kronecker products is another Kronecker product. These properties are applied in proofs found later in this chapter.

Norms are functions that take vectors or matrices as input and returns a scalar. They are used to calculate the length of vectors and evaluate the error of models in machine learning [3]. Norms are denoted by $\| \cdot \|$ and a subscript indicating which norm is being used. There are many different norms, and one relevant to this research is the *Frobenius norm.*

Definition 2.2 (Frobenius Norm) *[25] The Frobenius norm on an $m \times n$ matrix A is the square root of the absolute squares of all elements in the matrix. It can be* *thought of as a vector norm. Note its two forms*

$$\|A\|_{\mathcal{F}} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}|^2} = \sqrt{Tr(AA^H)}$$

where A^H is the conjugate transpose of A.

The concept of a norm can also be extended to tensors where the norm $\|\mathcal{X}\|_{\mathcal{T}}$ of tensor $\mathcal{X}$ is the square root of the sum of all entries in $\mathcal{X}$ squared.

2.2 Tensor Decompositions

A few methods for tensor decomposition exist and are useful in machine learning techniques such as dimensionality reduction and latent factor analysis [15]. *Canonical polyadic decomposition*, also known as CP-Decomposition or CANDECOMP, is a common generalization of the matrix SVD in which tensors are expressed as the sum of rank-one tensors [16]. To achieve CP decomposition, the following generalized minimization problem must be solved:

$$\min_{\hat{\mathcal{X}}} ||\mathcal{X} - \hat{\mathcal{X}}|| \text{ where } \hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circledcirc \mathbf{a}_r^{(2)} \circledcirc \ldots \circledcirc \mathbf{a}_r^{(n)}.$$

The rank-one tensors are normalized at unit length, and the scalings are stored in the λ_r value. One drawback to this method is that the rank is necessary for approximation. There is no trivial algorithm in computing the rank of a tensor as it the problem in NP-hard (a computational problem where the solution is easy to verify but difficult to compute) [15]. In practice, most algorithms fit for multiple ranks and then choose the best approximation [16].

The *Tucker decomposition* is similar to CANDECOMP, but a core tensor is derived with scalings along each axis. The minimization problem for Tucker decomposition is

$$\min_{\hat{\mathcal{X}}} ||\mathcal{X} - \hat{\mathcal{X}}|| \text{ where } \hat{\mathcal{X}} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_n=1}^{R_n} g_{r_1,r_2,\ldots,r_n} \mathbf{a}_{i_1 r_1}^{(1)} \circledcirc \mathbf{a}_{i_1 r_1}^{(2)} \circledcirc \ldots \circledcirc \mathbf{a}_{i_n r_n}^{(n)}$$

where the sum of the $g_{r_1,r_2,\ldots,r_n}$ terms is some tensor $\mathcal{G}$ that is the same dimension as the tensor being decomposed. Note that the Tucker decomposition is generally not unique as the core structure is arbitrary [16]. Furthermore, if $g_{r_1,r_2,\ldots,r_n} = 0$ for all $r_1 \neq r_2 \neq \ldots \neq r_n$ then the problem reduces to the CP-Decomposition [16]. These methods are useful despite their flaws, and we can use these ideas to develop our solution.

2.3 Vector Spaces and Function Spaces

A fundamental concept in linear algebra is that of a *vector space* which is a set of vectors that is closed under vector addition and vector multiplication and satisfies certain conditions [33]:

- Commutativity
- Associativity of vector addition
- Additive and scalar multiplication identity
- Existence of additive inverse
- Associativity of scalar multiplication

- Distributivity of scalar sums

- Distributivity of vector sums.

A list of vectors $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$ is a basis for a vector space if and only if every $\mathbf{v} \in V$ can be written uniquely as

$$\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_n\mathbf{v}_n.$$

If V is an n-dimensional vector space, then $V \cong \mathbb{R}^n$. Moreover, vector spaces are often categorized by their number of independent components. Two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are *linearly independent* if $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 = 0$ only when $a_1, a_2 = 0$. The *basis* of a vector space V is a subset of linearly independent vectors that spans V. Hence, the number of linearly independent components in a vector space is equal to the number of vectors in its basis. A frequently-used type of basis is the orthonormal basis where all vectors in the basis satisfy the conditions of normality and orthogonality. For some vector space V where $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n] \in V$ form an orthonormal basis, the length of each $\mathbf{v}_i$ is 1, so all vectors are *normal*. Also, all vectors on V are *orthogonal* such that $\langle \mathbf{v}_i, \mathbf{v}_j \rangle_V = 0$ if $i \neq j$ and $\langle \mathbf{v}_i, \mathbf{v}_j \rangle_V = 1$ if $i = j$.

Vector spaces are useful in decompositions, and properties can be added to build other spaces with use in tensor decomposition [16]. We define the inner product.

Definition 2.3 (Inner Product) *[16] The inner product on a real vector space V is the mapping $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ such that for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$ and $\alpha, \beta \in \mathbb{R}$.*

1. $\langle \mathbf{x}, \mathbf{y} + \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$ *(Linearity)*

2. $\langle \alpha\mathbf{x}, \mathbf{y} \rangle = \alpha\langle \mathbf{x}, \mathbf{y} \rangle$ *(Homogeneity)*

3. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ *(Symmetry)*

4. $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$, $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ *if and only if* $x = 0$ *(Nonnegativity and Positive Definiteness)*

The inner product maps two vectors to a scalar is also called the *dot product.* Furthermore, a vector space with an inner product is called an *inner product space.* Every inner product space has a norm defined by

$$||\mathbf{x}||_p = (\langle \mathbf{x}, \mathbf{x} \rangle)^{1/p} = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \quad [16].$$

A *complete* space is one in which all Cauchey sequences converge. With this condition, we define a Banach space.

Definition 2.4 (Banach Space) *[16] A Banach space is a normed vector space that is a complete metric space with respect to the metric derived from its norm.*

A special case occurs with the Banach space on the $p = 2$ norm.

Definition 2.5 (Hilbert Space) *[16] A Hilbert space is a normed vector space that is a complete inner product space with respect to the norm*

$$||\mathbf{x}||_2 = (\langle \mathbf{x}, \mathbf{x} \rangle)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$$

Note that every finite dimensional Hilbert space has an orthonormal basis [16]. $\mathcal{T}_j^i(\mathbb{R})$ is a vector space with dimension $i + j$, and $\ell(V)$ is a vector space of functions that map some set of vertices V to $\mathbb{R}$. This is also known as a function space.

Definition 2.6 (Function Space) *[16] A Function space is the set of all real-valued functions on a set X, denoted*

$$\ell(X) = \{f : X \to \mathbb{R}\}.$$

Function spaces are also vector spaces and have many of the same properties. For instance, if X is a finite set, then $\ell(X) \cong \mathbb{R}^n$ where $n = |X|$, and $fg \in \ell(X)$ for all $f, g \in \ell(X)$. Moreover, there exists an orthonormal basis for every $\ell(X)$.

2.4 Linear Transformations between Vector Spaces

In considering bringing matrix methods into the realm of tensors, we consider transformations of these vector and function spaces. We first define a common transformation.

Definition 2.7 (Linear Transformation) *[16] Let V and W be vector spaces. The mapping*

$$T : V \to W$$

is a linear transformation if

$$T(\alpha\mathbf{v} + \beta\mathbf{w}) = \alpha T(\mathbf{v}) + \beta T(\mathbf{w}) \textit{ for all } \mathbf{v}, \mathbf{w} \in V \textit{ and } \alpha, \beta \in \mathbb{R}.$$

Consider $T : \ell(G, \mathbb{R}) \to \ell(G, \mathbb{R})$ which is a linear transformation. Note that

$$(Tf)(i) = (Af)(i) = \sum_{i=1}^{n} a_{ij} f(j)$$

where A is a matrix. We can denote the standard basis as δ_i where for each $g_j \in G_{im}$, we get $\delta_i(g_j) = 1$ when $i = j$ and $\delta_i(g_j) = 0$ when $i \neq j$. Hence,

$$(Tf)(i) = \begin{bmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \begin{bmatrix} \delta_1(i) & \cdots & \delta_1(m) \end{bmatrix}.$$

We see here that linear transformations on vector spaces can be represented as matrices.

A *vector valued function space* is defined as

$$\ell(G, \mathbb{R}^n) = \{f : G \to \mathbb{R}^n\} = V \otimes \mathbb{R}^n$$

where V is a vector space. Our goal is to learn more about V, and we return to the image example for context. Note that a function space mapping a graph to $\mathbb{R}^2$ infers we are working with a black and white image with only one matrix layer. $\mathbb{R}^3$ is a color image with three layers, and $\mathbb{R}^4$ can represent video data as there are three color dimensions and a fourth time dimension. We remain within the realm of color images but note that these concepts should extend to higher dimensions. Suppose G_{im} is a network graph organized in a grid representing pixels in an $n \times m$ image as shown in Figure 7. Also, let each node on the graph contains the red, green and blue color values for the image.

We have

$$\ell(G_{im}, \mathbb{R}^3) = \{f : G_{im} \to \mathbb{R}^3\}$$

and know that for each row $i = 1, \ldots, n$ and column $j = 1, \ldots, m$ we get $f(i,j) \in \ell(G_{im}, \mathbb{R}^3) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ or a vector with three color values. In other words, we have a

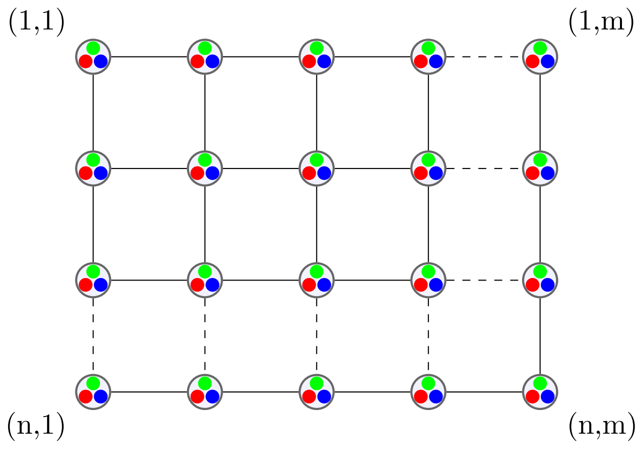

Figure 11: Network grid graph representation.

vector valued mapping. In tensor notation with a third index $k = 0, 1, 2$ we have that

$$f(i, j, 0) = X_0$$

$$f(i, j, 1) = X_1$$

$$f(i, j, 2) = X_2$$

where each X is one of three color layers.

Now consider $L : \ell(G_{im}, U) \to \ell(G_{im}, U)$ where $U \in \mathbb{R}^3$ is composed of vector valued functions. Each $f(g_j)$ is a vector that takes $\mathbb{R}^3 \to \mathbb{R}^3$, and so f can be thought of as a vector of vectors. Furthermore, the linear transformation L can be written as

$$(Lf) = \begin{bmatrix} A_{1,1} & \cdots & A_{1,m} \\ \vdots & & \vdots \\ A_{n,1} & \cdots & A_{n,m} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} A_{1,1}\mathbf{f}_1 + & \cdots & +A_{1,m}\mathbf{f}_m \\ \vdots & & \vdots \\ A_{n,1}\mathbf{f}_1 + & \cdots & +A_{n,m}\mathbf{f}_m \end{bmatrix}.$$

where the individual entries in L are all matrices and the transformation itself is a matrix of matrices which is a type of tensor. We are able to define a tensor space.

Definition 2.8 (Tensor Space) *$\ell(V, \mathbb{R}^n)$ is a vector space of $\mathbb{R}^n$ valued functions on V.*

Hence, linear transformations on $\ell(V, \mathbb{R}^n)$ are tensors.

2.5 Determining Tensor Order

We want to investigate the structure of mappings between tensors. Consider a simple case with $\mathbf{x} \in \mathbb{Z}_m$ and $\mathbf{y} \in \mathbb{R}^n$ and $\mathcal{A}$ a tensor. Then

$$f(\mathbf{x}) = \mathbf{y} \implies \mathbf{x} \xrightarrow{\mathcal{A}} \mathbf{y}$$

and in tensor notation we have

$$\mathbf{x}_m \mathcal{A} = \mathbf{y}^n \implies \mathcal{A} \in \mathcal{T}_0^{mn}$$

which is a two tensor. If we change $\mathbb{R}^n$ to $\mathbb{R}_n$ then

$$\mathbf{x}_m \mathcal{A} = \mathbf{y}_n \implies \mathcal{A} \in \mathcal{T}_n^{m}$$

which is another two tensor.

Consider

$$\ell(G; V) = \{f : G \to V\}$$

where V is an n-dimensional vector space and G is a group of nodes on a network. We want to determine if f is a two-tensor and what type it is. For instance, it could be $\mathcal{T}_0^2$ (a list of rows), $\mathcal{T}_2^0$ (a list of columns) or $\mathcal{T}_1^1$ (a 2D array). We now focus on proving the order of a tensor. If we let $f \in \ell(\mathbb{Z}_m, \mathbb{R}^n)$ then f is a linear transformation that maps elements of $\mathbb{Z}_m$ to $\mathbb{R}^n$. Furthermore,

$$\mathbf{f} = \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(m-1) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} x_{1,0} \\ \vdots \\ x_{n,0} \end{bmatrix} \\ \begin{bmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} x_{1,m-1} \\ \vdots \\ x_{n,m-1} \end{bmatrix} \end{bmatrix}$$

and the output from the function f appears to be a tensor. Every tensor space is also a vector space, and this can be easily shown. We let the function $f = \mathcal{A}$ and assume $\mathcal{A}$ is a tensor. Hence,

$$f(\mathbf{x}) = \mathbf{y} \implies \mathbf{x} \xrightarrow{\mathcal{A}} \mathbf{y}$$

and in tensor notation we have

$$\mathbf{x}_m \mathcal{A} = \mathbf{y}^n \implies \mathcal{A} \in \mathcal{T}_0^{mn}$$

which is a two tensor.

If we change $\mathbb{R}^n$ to $\mathbb{R}_n$ then we get

$$\mathbf{f} = \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(m-1) \end{bmatrix} = \begin{bmatrix} [x_{1,0}, \ldots, x_{n,0}] \\ [x_{1,1}, \ldots, x_{n,1}] \\ \vdots \\ [x_{1,m-1}, \ldots, x_{n,m-1}] \end{bmatrix}$$

which is also seems to be a tensor but of a different shape. Hence,

$$\mathbf{x}_m \mathcal{A} = \mathbf{y}_n \implies \mathcal{A} \in \mathcal{T}_n^{m}$$

which is another two tensor. From these examples we are able to deduce that the dimensions of a higher order linear transformation is dependent on the specified input and output tensors.

2.6 Conceptualization of Eigentensors

Graphs are related to manifolds, and machine learning techniques such as spectral embedding uses graphs and the global structure of manifolds. Furthermore, the Laplacian is used to compare graphs using eigenvectors. Hence, manifold learning is equivalent to finding eigenvector representations of Laplacian matrices. The eigenproblem, determining the eigentensors and eigenvalues of a given tensor, for tensor Laplacians is of great importance to this thesis.

The tensor product can be thought of as a generalization of the outer product and linear combinations of such. Consider $V = \ell(G_1) = \{f : G_1 \to \mathbb{R}\}$ and $U = \ell(G_2) = \{g : G_2 \to \mathbb{R}\}$. We define the tensor product between these two vector spaces as

$$V \otimes U = \ell(G_1 \times G_2) = \{f : (g_1, g_2) \to \mathbb{R}\}.$$

Furthermore, we can use the tensor product to construct a basis for $V \otimes U$ using $\mathbf{b}$ and $\mathbf{c}$. We formalize this statement with a theorem.

Theorem 2.9 *If $W = V \otimes U$ where $\mathbf{b} = [b_1, b_2, \dots, b_m]$ is a basis for vector space $V = \mathbb{R}^m$ and $\mathbf{c} = [c_1, c_2, \dots, c_n]$ is a basis for vector space $U = \mathbb{R}^n$ then $f \in W$ if and only if there exists α_{ij} such that*

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} (\mathbf{b}_i \circledcirc \mathbf{c}_j).$$

The key takeaway is that we obtain bases from tensor products, and a *tensor space* can be thought of as a linear combination of the tensor product of the basis vectors. We denote the inner product on V as $\langle\cdot,\cdot\rangle_V$ and on U as $\langle\cdot,\cdot\rangle_U$ and show how operations on vector spaces extend nicely to this new space.

Theorem 2.10 *With $W = V \otimes U$, suppose $e_1, \ldots, e_m$ is an orthonormal basis for $V \in \mathbb{R}^m$ and $f_1, \ldots, f_n$ is an orthonormal basis for $U \in \mathbb{R}^n$. If we define the inner product on W as*

$$\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_k \circledcirc \mathbf{f}_\ell \rangle_W = \langle \mathbf{e}_i, \mathbf{e}_k \rangle_V \langle \mathbf{f}_j, \mathbf{f}_\ell \rangle_U$$

then

1. *the definition of $\langle\cdot,\cdot\rangle_W$ on dyads of the orthonormal bases extend via linearity to an inner product on W, and*

2. *$\mathbf{e}_i \circledcirc \mathbf{f}_j$ where $i = 1, \ldots, m$ and $j = 1, \ldots, n$ is an orthonormal basis for W.*

Proof: *By Theorem 2.9, we know that the dyad $\mathbf{e}_i \circledcirc \mathbf{f}_j$ forms a basis on W. Thus, there exists some $x, y \in W$ where*

$$x = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} (\mathbf{e}_i \circledcirc \mathbf{f}_j) \text{ and } y = \sum_{i=1}^{m} \sum_{j=1}^{n} \beta_{ij} (\mathbf{e}_i \circledcirc \mathbf{f}_j)$$

for $\alpha, \beta \in \mathbb{R}$. Furthermore,

$$\begin{aligned}
\langle x, y\rangle_W &= \left\langle \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j), \sum_{i=1}^{m}\sum_{j=1}^{n} \beta_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j) \right\rangle_W \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij}\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_i \circledcirc \mathbf{f}_j\rangle_W \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij}\langle \mathbf{e}_i, \mathbf{e}_i\rangle_V \langle \mathbf{f}_j, \mathbf{f}_j\rangle_U \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij}
\end{aligned}$$

as a result of linearity and the fact that $\mathbf{e}$ and $\mathbf{f}$ are bases. We show this is an inner product:

1. *Additivity: Let $z = \sum_{i=1}^{m}\sum_{j=1}^{n} \iota_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j) \in W$. Note that*

$$x + y = \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij} + \beta_{ij})(\mathbf{e}_i \circledcirc \mathbf{f}_j)$$

and

$$\begin{aligned}
\langle x + y, z\rangle_W &= \left\langle \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij} + \beta_{ij})(\mathbf{e}_i \circledcirc \mathbf{f}_j), \sum_{i=1}^{m}\sum_{j=1}^{n} \iota_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j) \right\rangle_W \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij} + \beta_{ij})\iota_{ij}\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_i \circledcirc \mathbf{f}_j\rangle_W \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij} + \beta_{ij})\iota_{ij} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij}\iota_{ij} + \beta_{ij}\iota_{ij}) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} (\alpha_{ij}\iota_{ij}) + \sum_{i=1}^{m}\sum_{j=1}^{n} (\beta_{ij}\iota_{ij}) \\
&= \langle x, z\rangle_W + \langle y, z\rangle_W.
\end{aligned}$$

2. *Homogeneity: For some $a \in \mathbb{R}$ note that*

$$\begin{aligned}\langle ax, y\rangle_W &= \sum_{i=1}^{m}\sum_{j=1}^{n} a(\alpha_{ij}\beta_{ij}) \\ &= a\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij} \\ &= a\langle x, y\rangle_W.\end{aligned}$$

3. *Symmetry: We see that*

$$\langle x, y\rangle_W = \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij} = \sum_{j=1}^{n} \alpha_{ij}\beta_{ij}\sum_{i=1}^{m} = \langle y, x\rangle_W$$

since multiplication of scalars is commutative.

4. *Positive Definiteness: We have already shown that with*

$$y = \sum_{i=1}^{m}\sum_{j=1}^{n} \beta_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j) \text{ and } z = \sum_{i=1}^{m}\sum_{j=1}^{n} \iota_{ij}(\mathbf{e}_i \circledcirc \mathbf{f}_j)$$

then $\langle y, z\rangle_W = \sum_{i=1}^{m}\sum_{j=1}^{n}(\beta_{ij}\iota_{ij}) \geq 0$, so let $z = \sum_{i=1}^{m}\sum_{j=1}^{n} \iota_{k\ell}(\mathbf{e}_k \circledcirc \mathbf{f}_\ell)$ and there be some $\langle y, z\rangle_W = 0$. Hence,

$$\begin{aligned}0 &= \langle y, z\rangle_W \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij}\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_k \circledcirc \mathbf{f}_\ell\rangle_W \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij}\beta_{ij}\langle \mathbf{e}_i, \mathbf{e}_k\rangle_V\langle \mathbf{f}_j, \mathbf{f}_\ell\rangle_U\end{aligned}$$

This result shows that $\langle \mathbf{e}_i, \mathbf{e}_k\rangle_V = 0$ or $\langle \mathbf{f}_j, \mathbf{f}_\ell\rangle_U = 0$. Indeed, both $\langle \mathbf{e}_i, \mathbf{e}_j\rangle_V = 0$ and $\langle \mathbf{f}_j, \mathbf{f}_\ell\rangle_U = 0$ since $\mathbf{e}$ and $\mathbf{f}$ are orthonormal bases. The converse statement *holds as well where*

$$\begin{aligned}\langle y, z\rangle_W &= \sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij}\beta_{ij}\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_k \circledcirc \mathbf{f}_\ell\rangle_W \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij}\beta_{ij}\langle \mathbf{e}_i, \mathbf{e}_k\rangle_V\langle \mathbf{f}_j, \mathbf{f}_\ell\rangle_U \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij}\beta_{ij}(0)(0) = 0\end{aligned}$$

so $\langle \mathbf{v}, \mathbf{v}\rangle_W = 0$ if and only if $\mathbf{v} = 0$. Thus, $\langle \cdot, \cdot\rangle_W$ meets the criteria of an inner product.

Finally, we show that the basis $\mathbf{e}_i \circledcirc \mathbf{f}_j$ is orthonormal. By the definition of the inner product on W,

$$\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_i \circledcirc \mathbf{f}_j\rangle_W = \langle \mathbf{e}_i, \mathbf{e}_i\rangle_V\langle \mathbf{f}_j, \mathbf{f}_j\rangle_U = (1)(1) = 1$$

and

$$\langle \mathbf{e}_i \circledcirc \mathbf{f}_j, \mathbf{e}_k \circledcirc \mathbf{f}_\ell\rangle_W = \langle \mathbf{e}_i, \mathbf{e}_k\rangle_V\langle \mathbf{f}_j, \mathbf{f}_\ell\rangle_U = (0)(0) = 0.$$

Hence, $\mathbf{e}_i \circledcirc \mathbf{f}_j$ is an orthonormal basis on W and $\langle \cdot, \cdot\rangle_W$ is an inner product. □

We now have the tools to investigate eigentensors. Note that we can represent linear transformations as linear combinations of Kronecker products which are equivalent to linear combinations of dyads. Hence, we are able to construct a representation of linear transformations on a tensor space using the Kronecker product. Using properties of the Kronecker product, we are able to define the eigenproblem in terms of tensors.

Theorem 2.11 *Suppose $A \in \mathcal{M}_n(\mathbb{R})$ and $B \in \mathcal{M}_m(\mathbb{R})$. Note that this implies $A \otimes B \in \mathcal{M}_{nm}(\mathbb{R})$. Consider $A\mathbf{v} = \lambda\mathbf{v}$ where $\mathbf{v} \in \mathbb{R}^n$ and $B\mathbf{w} = \gamma\mathbf{w}$ where $\mathbf{w} \in \mathbb{R}^m$.*

Again note that $\mathbf{v} \otimes \mathbf{w} \in \mathbb{R}^{mn}$. *Show that*

$$(A \otimes B)(\mathbf{v} \otimes \mathbf{w}) = \lambda\gamma(\mathbf{v} \otimes \mathbf{w})$$

Proof: *Using the relationship (KRON 7) from section 2.1, note*

$$\begin{aligned}(A \otimes B)(\mathbf{v} \otimes \mathbf{w}) &= (A\mathbf{v}) \otimes (B\mathbf{w}) \\ &= (\lambda\mathbf{v}) \otimes (\gamma\mathbf{w}) \\ &= \lambda\gamma(\mathbf{v} \otimes \mathbf{w}).\end{aligned}$$

Thus, $\lambda\gamma$ *is an eigenvalue of* $A \otimes B$ *and* $\mathbf{v} \otimes \mathbf{w}$ *is the corresponding eigenvector.*

□

Similar to Theorem 2.11, the linear combination of eigenvectors produces something that functions as an eigentensor. Finally, we show that this eigentensor satisfies the definition of a basis.

Theorem 2.12 *Suppose* $A = A^T$ *and* $B = B^T$ *(*A *and* B *are symmetric). Show that the eigenvector* $\mathbf{v_i} \otimes \mathbf{w_j}$ *forms a basis for a tensor space.*

Proof:

Consider two Hilbert spaces H_1 *and* H_2 *and their inner products* $\langle \cdot, \cdot \rangle_1$ *and* $\langle \cdot, \cdot \rangle_2$. *We construct a new space* $q : H_1 \times H_2 \to H_1 \otimes H_2$ *and define an inner product*

$$\langle \phi_1 \otimes \phi_2, \psi_1 \otimes \psi_2 \rangle_q = \langle \phi_1, \psi_1 \rangle_1 \langle \phi_2, \psi_2 \rangle_2 \textit{ for } \phi_1, \phi_2 \in H_1 \textit{ and } \psi_1, \psi_2 \in H_2$$

such that we have an inner product space. Since both A *and* B *are square matrices, they can be decomposed as* $Q\Lambda Q^{-1}$ *where the columns of* Q *are orthogonal vectors and* Λ *is a diagonal matrix with the corresponding eigenvalues. Furthermore,* $A \in \mathcal{M}_n(\mathbb{R})$

and $B \in \mathcal{M}_m(\mathbb{R})$ implies $\mathbf{v}_i \in \mathbb{R}^n$ and $\mathbf{w}_j \in \mathbb{R}^m$. Also, $\mathbf{v}_i$ and $\mathbf{w}_j$ are bases for A and B respectively. If we let $A \in H_1$ and $B \in H_2$ then $H_1 \otimes H_2$ is the same dimension as $A \otimes B$. Then $\mathbf{v}_i \otimes \mathbf{w}_j$ is of the same dimension as well, and by Theorem 2.10, we know it forms a basis for the tensor space as well.

Because $\mathbf{v}_i \otimes \mathbf{w}_j$ is a set of independent vectors of the same dimension as $H_1 \otimes H_2$ then we conclude that it forms a basis for the tensor space. $\square$

Eigentensors can be defined for specific problems, but higher order complexity makes a generalized eigentensor difficult to define. However, eigentensors, especially those of the Laplacian tensor, are crucial to bridging the gap between matrix methods and tensor methods.

3 RESEARCH AND APPLICATION

3.1 Limitations of Tensor Methods

While tensor decomposition algorithms exist, they are not as powerful as analogous matrix operations. For instance, both CPD and Tucker Decomposition deconstruct tensors into important parts similar to Singular Value Decomposition. SVD requires a diagonal matrix with entries $x_{ij} = 0$ if $i \neq j$, and extending this idea to tensors may look like a tensor where $t_{ijk} = 0$ if $i \neq j$ or $j \neq k$ or $i \neq k$. These three conditions cannot be met simultaneously, so there are always different ways to create a diagonal tensor based on its shape. The lack of a unique, generalized SVD method for tensors has spawned various tensor methods with specific use cases.

For another example of limitations in tensor methods, recall from the Poisson image blending example the *least squares minimization* problem

$$\mathbf{x}_* = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|A\mathbf{x} - \mathbf{b}\|_2^2 \text{ where } A \in \mathbb{R}^{m\times n} \text{ and } \mathbf{b} \in \mathbb{R}^m.$$

Solving this problem is useful in evaluating error in statistical models and regression analysis. The solution is of the form $\mathbf{x}_* = A^+\mathbf{b}$ with $\mathbf{x}_* \in \mathbb{R}^n$ where A^+ is the Moore-Penrose pseudoinverse. If we change $\mathbf{b}$ into a matrix B and use the Frobenius norm then the problem becomes more complex.

$$X_* = \underset{X\in\mathbb{R}^{n\times \ell}}{\operatorname{argmin}} \|AX - B\|_{\mathcal{F}}^2 \text{ where } A \in \mathbb{R}^{m\times n} \text{ and } B \in \mathbb{R}^{m\times \ell}.$$

We are able to solve this new minimization problem and show that the result will be a matrix instead of a vector. Lemma 3.1 provides a useful relationship between the Frobenius norm and the vector norm.

Lemma 3.1 *For any matrix $M = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_\ell]$ we have*

$$\|M\|_{\mathcal{F}}^2 = \|\mathbf{m}_1\|_2^2 + \ldots + \|\mathbf{m}_\ell\|_2^2.$$

Proof: *Let $M = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_\ell]$ be a matrix in $\mathbb{R}^{n\times\ell}$. Note that the Frobenius norm of M is*

$$\begin{aligned}
\|M\|_{\mathcal{F}} &= \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{\ell}|m_{ij}|^2} \\
&= \sqrt{\sum_{j=1}^{\ell}\sum_{i=1}^{n}|m_{ij}|^2} \\
&= \sqrt{\sum_{j=1}^{\ell}[\mathbf{m}_1^2, \mathbf{m}_2^2, \ldots, \mathbf{m}_\ell^2]} \\
&= \sqrt{[\mathbf{m}_1^2 + \mathbf{m}_2^2 + \ldots + \mathbf{m}_\ell^2]}
\end{aligned}$$

Hence, we square the Frobenius norm in this form to reach the desired result.

$$\|M\|_{\mathcal{F}}^2 = [\mathbf{m}_1^2 + \mathbf{m}_2^2 + \ldots + \mathbf{m}_\ell^2] = \|\mathbf{m}_1\|_2^2 + \|\mathbf{m}_2\|_2^2 + \ldots + \|\mathbf{m}_\ell\|_2^2.$$

□

We see the Forbenius norm acting as a vector norm for every column vector in a given matrix. Thus, we determine how the output of the least squares problem changes when B is a matrix.

Proposition 3.2 *The solution $X_* = \underset{X\in\mathbb{R}^{n\times\ell}}{\operatorname{argmin}}\|AX - B\|_{\mathcal{F}}^2 = A^+B$ where $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times\ell}$ is a matrix.*

Proof: *Let* $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell]$ *and* $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_\ell]$. *Note*

$$AX - B = [A\mathbf{x}_1 - \mathbf{b}_1, A\mathbf{x}_2 - \mathbf{b}_2, \ldots, A\mathbf{x}_\ell - \mathbf{b}_\ell]$$

and we see the problem can be viewed as solving the simple iteration least squares minimization problem ℓ *times. Hence,*

$$\begin{aligned}\underset{X\in\mathbb{R}^{n\times\ell}}{\operatorname{argmin}}\|AX - B\|_{\mathcal{F}}^2 &= \underset{X\in\mathbb{R}^{n\times\ell}}{\operatorname{argmin}}\left(\sum_{k=1}^{\ell}\|A\mathbf{x}_k - \mathbf{b}_k\|^2\right) && \textit{by Lemma 3.1}\\ &= \sum_{k=1}^{\ell}\left(\underset{X\in\mathbb{R}^{n\times\ell}}{\operatorname{argmin}}\|A\mathbf{x}_k - \mathbf{b}_k\|^2\right).\end{aligned}$$

Since $\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\|A\mathbf{x}_i - \mathbf{b}_i\|_2^2 = A^+\mathbf{b}_i$ *we can deduce that*

$$X_* = \sum_{k=1}^{\ell}\left(A^+\mathbf{b}_k\right) = [A^+\mathbf{b}_1, A^+\mathbf{b}_2, \ldots, A^+\mathbf{b}_\ell] = A^+B.$$

With $A^+ \in \mathbb{R}^{n\times m}$ *and* $B \in \mathbb{R}^{m\times\ell}$ *we conclude that* X_* *is a matrix of size* $n \times \ell$. □

Finding a matrix solution to the least squares problem is straightforward, but solving the least squares problem when $\mathcal{A}$ and $\mathcal{B}$ are tensors is not straightforward. For example, consider

$$\mathcal{X}_* = \underset{\mathcal{X}}{\operatorname{argmin}}\|\mathcal{A}\mathcal{X} - \mathcal{B}\|_{\mathcal{T}}^2 \text{ where } \mathcal{A} \text{ and } \mathcal{B} \text{ are tensors.}$$

As shown in chapter 2, the product of two tensors is another tensor, so we assume the solution $\mathcal{X}_*$ is a tensor as well. In determining what space $\mathcal{X}$ is in, we encounter issues with the problem being ill-defined. For instance, if you have some rank three tensor $\mathcal{A} = [a_{ijk}]$ and a rank one tensor $\mathcal{B} = [b_\ell]$ then a system $\mathcal{A} \otimes \mathcal{X} = \mathcal{B}$ implies $\mathcal{X}$ is a two-tensor, but there are multiple ways to combine a rank two tensor and a rank three tensor. Possibilities in this scenario include

$$\sum_{i,j} a_{ijk}x_{ij}, \sum_{j,k} a_{ijk}x_{jk} \text{ or } \sum_{i,k} a_{ijk}x_{ik}$$

all of which output a rank one tensor. This non-uniqueness adds difficulty to generalizing the Moore-Penrose pseudoinverse for tensors. Furthermore, the Moore-Penrose pseudoinverse is the matrix of a linear transformation, so we must develop a concept of linear transformations on tensor spaces before properly defining the pseudoinverse of a tensor.

The tensorized least squares problem can be solved with an algorithm, and we consider $\min\limits_{D,F} \|A - DF\|_{\mathcal{F}}^2$ given the constraints $\text{rank}(D) = \text{rank}(F) = r$ to see an example of one.

1. Fix F and solve the minimization problem for D such that $D_{*1} = AF^{+}$.

2. Fix D_{*1} and solve the minimization problem for F such that $F_{*1} = D_{*1}^{+}A$.

3. Fix F_{*1} and solve the minimization problem for D such that $D_{*2} = AF_{*1}^{+}$.

4. Fix D_{*2} and solve the minimization problem for F such that $F_{*2} = D_{*2}^{+}A$.

5. Repeat this process until the optimal D_* and F_* are found.

This is known as the *alternating least squares* algorithm and plays a central role in CP-decomposition. While very useful, the tensor rank is required as input which creates problems [15].

Tensor decomposition algorithms have proved useful in deep learning, but there is room for improvement. For example, CANDECOMP and Tucker decomposition do not utilize the covariant and contravariant structure of tensors and can be clunky in implementation. Hence, we seek an alternative method.

3.2 Random Walk on a Network

A *random walk* is a discrete series of steps [31]. If someone flips a coin with equal probability of heads and tails and moves a step to the left if they get heads and to the right if they get tails, then the path and ending location after n coinflips is a simple random walk. Random walks can also be iterated over network structures and yield interesting mathematical insights. Suppose we have the graph shown in Figure 12.

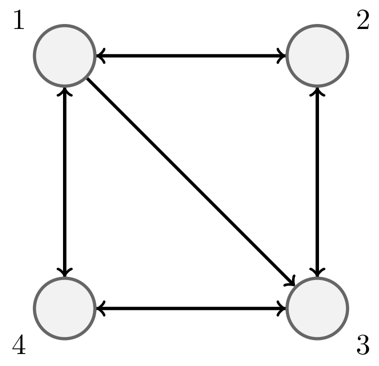

Figure 12: Example of a directed graph.

A *probability transition matrix* can model flows through a network. Every node is represented as a row and a column, and entries in this matrix correspond to the likelihood of moving between the nodes. The probability matrix for the graph in Figure 12 is

$$P = \begin{bmatrix} 0 & P_{12} & P_{13} & P_{14} \\ P_{21} & 0 & P_{23} & 0 \\ 0 & 0 & 0 & P_{34} \\ P_{41} & 0 & 0 & 0 \end{bmatrix}$$

Note that P is *row stochastic* meaning all row entries sum to 1. A *natural random walk* occurs when all edges from a given node have the same weight. Moreover, each row has the same probability entries. Hence,

$$P_{natural} = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix}$$

is the natural random walk for the network shown in Figure 12. We utilize a feature of these random walks illustrate their connection to tensors.

Importantly, there exists a connection between a natural random walk and the *edge-weighted Laplacian*, denoted $L_w = S^T W S$ where W is a diagonal matrix of edge weights and S is an incidence matrix. Note that the S^T matrix will map vertexes to edges, W from edged to edges, and S maps edged back to vertexes. An *incidence matrix* is a logical matrix that shows the relationship between two classes of objects and includes information about the weights and adjacency structure of a graph. The incidence matrix for the graph in Figure 12 is

$$S = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{bmatrix}$$

where the columns represent vertices and the rows represent edges. A weight matrix can be represented for this graph as

$$\begin{bmatrix} w_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_9 \end{bmatrix}.$$

The Laplacian in this case is represented as $L = S^T W S$. If W is an identity matrix then $S^T S = L = V\Sigma^2 V^T$ and $S = U\Sigma V^T$. If each vertex is represented by a vector, the weight matrix is

$$\begin{bmatrix} [W_1] & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & [W_2] & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & [W_3] & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & [W_4] & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & [W_5] & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & [W_6] & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & [W_7] & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & [W_8] & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & [W_9] \end{bmatrix}$$

where each W_n is a square $n \times n$ matrix in which n is the size of the vertex vectors. A Network Laplacian matrix in this context would look like

$$L_w = \begin{bmatrix} D_{11} & -W_{12} & \cdots & -W_{1n} \\ -W_{21} & D_{22} & \cdots & -W_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -W_{m1} & -W_{m2} & \cdots & D_{mn} \end{bmatrix}$$

such that $D_{ii} = \sum_i^n W_{ij}$. The connecting idea is the equation

$$P = D^{-1}A = D^{-1}(D - L) = I - D^{-1}L$$

where A is the adjacency matrix and D is the degree matrix. We can also define the

Laplacian matrix with only the degree matrix and probability matrix where

$$L = D(I - P).$$

The matrix P is assumed to be invertible and diagonal, but these conditions are not always present. We see that with just the probability matrix we are able to calculate the Laplacian on a graph and vice versa.

A unique property of the natural random walk matrix is that the columns (rows?) of $P_{natural}$ will converge when high powers of the matrix are taken. This means that the matrix $P^n_{natural}$ for a sufficiently large n reaches equilibrium. Since each column of $P^n_{natural}$ converges to a single value, all rows are identical, and we notice that one of these rows acts as an eigenvector for $P_{natural}$. This concept of raising a transition matrix to some power holds if $P_{natural}$ is a tensor, and we calculate some examples in Python in Appendix A.2. We use this fact to derive the concept of an eigentensor of a Laplacian in section 3.3.

3.3 Eigentensors of a Network Laplacian

We now reach the fundamental question of this thesis. Consider an edge-weighted Laplacian L_u on the function space $\ell(G, U) = \{f : G \to U\}$ for a graph G and a vector space $U \in \mathbb{R}^n$. Indeed, a linear transformation on this function space is a tensor, and we attempt to derive the eigenequation for this tensor.

As shown in chapter 2, we can represent the transformation on a tensor space using the Kronecker product. The structure of this function space may be $\ell(G, U) = L_u \otimes I$ for an unweighted network. A linear transformation L in this unweighted context looks like

$$(Lf)(i) = \sum_{i\sim j}(f(i) - f(j))$$

where $\sum_{i\sim j} f(i) = f(i)\sum_{i\sim j} 1 = d_i$ and d_i is the degree of the ith vertex. For the case where every node on a network has the same weights, $\ell(G, U) = L_u \otimes W$ where W is some matrix of weights on U. The Laplacian matrix in which every term is multiplied by the same matrix W using the Kronecker product is

$$L_u \otimes W = \begin{bmatrix} d_1W & -1W & \cdots & 0W \\ -1W & d_2W & \cdots & -1W \\ \vdots & \vdots & \ddots & \vdots \\ 0W & -1W & \cdots & d_nW \end{bmatrix}.$$

With weighted nodes then d_i becomes the sum of the weights of a row and the transformation is

$$(Lf)(i) = \sum_{i\sim j} w_{ij}(f(i) - f(j)).$$

It is reasonable to assume that each node might have weights W_{ij} where each W_{ij} is different:

$$L_u \otimes W_{ij} = \begin{bmatrix} d_1W_{11} & -1W_{12} & \cdots & 0W_{1n} \\ -1W_{21} & d_2W_{22} & \cdots & -1W_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0W_{m1} & -1W_{m2} & \cdots & d_nW_{mn} \end{bmatrix}.$$

Hence, a linear transformation on our weighted, vector valued example L_u on $\ell(G, U)$ where $\mathbf{f}(i) = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_n \end{bmatrix} \in U$ is of the form

$$(L_u\mathbf{f})(i) = \sum_{i\sim j} W_{ij}(\mathbf{f}(i) - \mathbf{f}(j))$$

and we note that the W_{ij} terms are now matrices where $W : U \to U$ is a linear transformation.

The eigenproblem in this context looks like

$$(L_u\mathbf{v})(i) = \lambda\mathbf{v}(i) = \sum_{i\sim j} W_{ij}(\mathbf{v}(i) - \mathbf{v}(j))$$

for every $i \in V$ and $G = (V, E)$. To solve the eigentensor problem we assume $W_{ij} = W_{ji}$ and solve for the $\mathbf{v}(i)$ term in the eigenequation. We let $W_{ij} = w_{ij}I_m$ where $I_m : V \to V$ then the eigenvectors reduce to that of a regular Laplacian. Also note that if $\mathbf{v}_j \in \ker(W_{ij})$ for every $j = 1, \ldots, n$ then we get

$$\lambda\mathbf{v}_i = \left(\sum_{j\sim i} W_{ij}\right)\mathbf{v}_i$$

and we are left with a simpler eigenvalue problem. We then let $W_{ij} = \mathbf{e}_j \circledcirc \mathbf{e}_i$ where $\mathbf{e}_i$ is an orthonormal basis for V. Then $W_{ij}\mathbf{e}_j = 0$ and

$$\lambda\mathbf{v}_i = \left(\sum_{j\sim i} \mathbf{e}_j \circledcirc \mathbf{e}_i\right)\mathbf{v}_i$$

such that our representation is a tensor defined as a linear combination of dyads.

However, more than likely, it is incorrect to assume $W_{ij} = W_{ji}$. The correct statement is that $W_{ij} = W_{ji}^T$ meaning we may not have a self-adjoint matrix. Notice that if we let $W_{ij} = \mathbf{e}_i\mathbf{e}_j^T + \mathbf{e}_j\mathbf{e}_i^T$ then

$$W_{ij}\mathbf{e}_j = \mathbf{e}_i\mathbf{e}_j^T\mathbf{e}_j + \mathbf{e}_j\mathbf{e}_i^T\mathbf{e}_j = \mathbf{e}_i(1) + \mathbf{e}_j(0) = \mathbf{e}_i$$

and we see that $W_{ij} = W_{ij}^T$. If we assume $W_{ij}\mathbf{v}_j = \alpha\mathbf{v}_i$ then

$$\begin{aligned}
\lambda\mathbf{v}_i &= \sum_{j\sim i}\left(W_{ij}\mathbf{v}_i - W_{ij}\mathbf{v}_j\right) \\
&= \sum_{j\sim i}\left(W_{ij}\mathbf{v}_i - \alpha\mathbf{v}_i\right) \\
&= \sum_{j\sim i}\left(W_{ij} - \alpha I\right)\mathbf{v}_i.
\end{aligned}$$

This is also a linear combination of dyads and a more generalized eigentensor. Hence, we can solve eigenvalue problem for the Laplacian for $i = 1, \dots, n$ or for each vertex. Moreover, the random walk over the probability transition matrix provides a method of computing this solution.

3.4 Poisson Blending Application

When running the image blending algorithm, we are really solving

$$\mathbf{x}_* = \operatorname{argmin} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

for each matrix layer where A is a sparse Laplacian matrix. The output is three vectors, say $\mathbf{x}_{*red}, \mathbf{x}_{*green}$ and $\mathbf{x}_{*blue}$ that must me reshaped and recombined into an image. In theory, we can say $\mathbf{x}_* = A^+\mathbf{b}$, but the pseudoinverse might not be sparse.

Our goal has been to theorize what happens when we have some tensor $\mathcal{B}$ and a vertex weighted Laplacian $\mathcal{A}$. Thus, we set up the problem

$$\mathcal{X}_* = \operatorname{argmin} \|\mathcal{A}\mathcal{X} - \mathcal{B}\|_{\mathcal{T}}^2.$$

Finding the eigentensors of $\mathcal{A}$ lets us find a basis for the tensor space and decompose $\mathcal{A} = V\Sigma V^T$ using the tensor product. If we let $\mathcal{X} = VY$ and $\mathcal{B} = VC$ for some matrices Y and C, then our problem reduces to

$$\begin{aligned}
\operatorname{argmin} \|A\mathcal{X} - \mathcal{B}\|^2 &= \operatorname{argmin} \|(V\Sigma V^T)(VY) - (VC)\|^2 \\
&= \operatorname{argmin} \|(V\Sigma Y) - (VC)\|^2 \\
&= \operatorname{argmin} \|V(\Sigma Y - C)\|^2 \\
&= \operatorname{argmin} \|(\Sigma Y - C)\|^2 && \text{since } V \text{ is unitary}
\end{aligned}$$

as a result of V being unitary. This is a much easier system to solve. Thus, *the eigenvectors of a tensor allow us to solve higher order linear equations.*

First, let's investigate the dimensionality of the problem. A color photo has $m \times n$ pixels in three colors layers, so the picture can be represented as a $m \times n \times 3$ tensor. The blended gradient image, previously represented by $\mathbf{b}$ is also of size $m \times n \times 3$. In the matrix algorithm, $m \times n$ slices are flattened to create $(nm) \times 1$ vectors, so consider $\mathcal{X}, \mathcal{B} \in \mathcal{T}_1^2$. Hence, the transformation between these two tensors can be represented as a tensor as well. Note

$$\mathcal{X} \xrightarrow{\mathcal{A}} \mathcal{B} \implies \mathcal{A} \in \mathcal{T}_2^2$$

and we see that $\mathcal{A}$ can be a tensor of dimension $3 \times (mn) \times 3 \times (mn)$. Indeed, tensor $\mathcal{A}$ can be imagined as a $3 \times (mn)$ matrix of $3 \times (mn)$ matrices.

Now we have a Laplacian tensor, and we know the eigentensors of $\mathcal{A}$ are useful. With this and the methods derived in section 3.2, we are able to rewrite our tensorized problem statement in terms of matrices:

$$\begin{aligned}
\mathcal{X}_* &= \operatorname{argmin} \|A\mathcal{X} - \mathcal{B}\|^2 \\
&= \operatorname{argmin} \|(\Sigma Y - C)\|^2 \\
&= \operatorname{argmin} \left\| \left(\left(\sum_{j \sim i} W_{ij} - \alpha I \right) Y - C \right) \right\|^2
\end{aligned}$$

and solve for each vertex. This greatly reduces the complexity of the problem.

From a computational standpoint, we can relate the concept of eigenvectors on a natural random walk to the weighted Laplacian. Suppose we derive some L_{color} Laplacian for an image and calculate a natural random walk P_{color}. We calculate an eigenvector $\mathbf{p}$ by raising P_{color} to a sufficiently high number of powers such that

$$P_{\text{color}}\mathbf{p} = \Lambda\mathbf{p}.$$

Using the facts that

$$P_{\text{color}} = (I - D^{-1}L_{\text{color}}) \text{ and } L_{\text{color}} = D(I - P_{\text{color}})$$

we see that

$$\begin{aligned} P_{\text{color}}\mathbf{p} &= \Lambda\mathbf{p} \\ &= (I - D^{-1}L_{\text{color}})\mathbf{p} \\ &= \mathbf{p} - (D^{-1}L_{\text{color}})\mathbf{p} \end{aligned}$$

and we are able to solve for $L_{\text{color}}\mathbf{p}$. Hence,

$$\begin{aligned} L_{\text{color}}\mathbf{p} &= D(\mathbf{p} - P_{\text{color}}\mathbf{p}) \\ &= D(I - P_{\text{color}})\mathbf{p} \\ &= D(I - \Lambda)\mathbf{p} \end{aligned}$$

and we get the eigentensor of the weighted Laplacian built on $\mathbf{p}$. With this tensorized least squares problem, we are able to calculate our blended image in one step instead of three and maintain the relationships between the color channels.

3.5 Conclusion and Future Work

Many data sets have a naturally occurring higher dimensions that cannot be well-captured by matrices. For instance, scientific experiments often measure two variables over a series of trials, and matrix methods are iterated over the samples. Instead, tensor methods allow one to preserve the timing of the trials in the data and may lead to more efficient learning methods. Moreover, we are able to build on existing matrix methods such as diagonalization, spectral decomposition, singular value decomposition, and finding pseudoinverses by building on the concept of eigenvectors on a tensor. Despite the obstacle of non-uniqueness, we derive a basis found when considering equilibrium reached in a natural random walk.

BIBLIOGRAPHY

[1] Gunnar Carlsson. Topology and data, 2009.

[2] Jérémy Charlier, Radu State, and Jean Hilger. Modeling smart contracts activities: A tensor based approach. *CoRR*, abs/1905.09868, 2019.

[3] CHIRAG676. Must known vector norms in machine learning. Analytics Vidyha, March 2021.

[4] Encyclopedia.com. Laplace, pierre-simon, marquis de. Complete Dictionary of Scientific Biography., April 2021.

[5] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171 – 1220, 2008.

[6] Jonathan Hui. Machine learning linear algebra — special matrices, February 2019.

[7] Yuwang Ji, Qiang Wang, Xuan Li, and Jie Liu. A survey on tensor techniques and applications in machine learning. *IEEE Access*, 7:162950–162990, 2019.

[8] Brian Keng. Manifolds: A gentle introduction.

[9] Debra Knisley and Jeff Knisley. Vertex-weighted graphs and their applications. *Utilitas Mathematica*, 94, 07 2014.

[10] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rossi, and H.P. Seidel. Differential coordinates for interactive mesh editing. In *Proceedings Shape Modeling Applications, 2004.*, pages 181–190, 2004.

[11] Adrien Maglo, Clément Courbet, Pierre Alliez, and Céline Hudelot. Progressive compression of manifold polygon meshes. *Computers Graphics*, 36(5):349–359, 2012. Shape Modeling International (SMI) Conference 2012.

[12] Kusuma Wardhani Mega, Xiangru Yu, and Jinping Li. Comparative analysis of color edge detection for image segmentation. In *Proceedings of the 2018 International Conference on Computing and Pattern Recognition*, ICCPR '18, pages 93–101, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Luke Melas-Kyriazi. The mathematical foundations of manifold learning, 2020.

[14] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.

[15] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning, 2017.

[16] Justin Reising. Function space tensor decomposition and its application in sports analytics. Master's thesis, East Tennessee State University, December 2019.

[17] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[18] Kathrin Schäcke. On the kronecker product kathrin schäcke. 2013.

[19] F. Schmidt. The laplace-beltrami-operator on riemannian manifolds the laplace-beltrami-operator on riemannian manifolds. 2014.

[20] Taha Sochi. Introduction to tensor calculus, 2016.

[21] Jolliffe Ian T. and Cadima Jorge. Principal component analysis: a review and recent developments, 2016.

[22] Ulrike von Luxburg. A tutorial on spectral clustering, 2007.

[23] Eric W. Weisstein. Colvolution. From MathWorld–A Wolfram Web Resource.

[24] Eric W. Weisstein. Convex function. From MathWorld–A Wolfram Web Resource.

[25] Eric W. Weisstein. Frobenius norm. From MathWorld–A Wolfram Web Resource.

[26] Eric W. Weisstein. Graph. From MathWorld–A Wolfram Web Resource.

[27] Eric W. Weisstein. Laplacian. From MathWorld–A Wolfram Web Resource.

[28] Eric W. Weisstein. Laplacian matrix. From MathWorld–A Wolfram Web Resource.

[29] Eric W. Weisstein. Matrix diagonalization. From MathWorld–A Wolfram Web Resource.

[30] Eric W. Weisstein. Moore-penrose matrix inverse. From MathWorld–A Wolfram Web Resource.

[31] Eric W. Weisstein. Random walk. From MathWorld–A Wolfram Web Resource.

[32] Eric W. Weisstein. Singular value decomposition. From MathWorld–A Wolfram Web Resource.

[33] Eric W. Weisstein. Vector space. From MathWorld–A Wolfram Web Resource.

[34] Lingzhi Zhang, Tarmily Wen, and Jianbo Shi. Deep image blending, 2019.

APPENDICES

A Code Implementation

A.1 Poisson Blending with Matrix Methods

```
#Import Packages

%matplotlib inline
from matplotlib import pyplot as plt

import numpy as np
import pandas as pd
import torch
import kornia
import scipy.sparse
import os

from kornia.filters import Laplacian
from torchvision import datasets, transforms
from scipy.sparse.linalg import spsolve, cg

# specify kernel size
ker_size = 3

# empty lot in Johnson City
emptyLot = plt.imread('colocafe2.jpg')
emptyLot = emptyLot / emptyLot.max()
plane = plt.imread('utahball.jpg')
plane = plane / plane.max()
mask2 = plt.imread('utahmask.jpg')
mask2 = mask2 / mask2.max()

# blend pictures based on mask
blendpic = mask2*emptyLot + (1-mask2)*plane

# create a transform to transform the images to tensors
to_tensor = transforms.ToTensor()

# 'Unsqueeze' the image tensor to fit to the kornia filter
   's dimension requirements BxCxHxW
source2 = torch.unsqueeze(to_tensor(plane), dim=0)

# 'Unsqueeze' the image tensor to fit to the kornia filter
   's dimension requirements BxCxHxW
target2 = torch.unsqueeze(to_tensor(emptyLot), dim=0)
```

```
# Check shape of tensor
source2.shape, target2.shape

# take the Laplacian of the tensor using a 3x3 kernel size
lapball = Laplacian(source2, ker_size)
# convert the tensor back to image
imagelapball = kornia.tensor_to_image(lapball)
# take the Laplacian of the tensor using a 3x3 kernel size
laplot = Laplacian(target2, ker_size)
# convert the tensor back to image
imagelaplot = kornia.tensor_to_image(laplot)


# example output
fig,ax = plt.subplots(1,2,figsize = (10,20))
ax[0].imshow(plane)
ax[0].set_title('Source')
ax[1].imshow((imagelapball - imagelapball.min())/(
   imagelapball.max() - imagelapball.min()))
ax[1].set_title('Gradient');

# mixes the balloon of the source with everything but the
   balloon area of the target
blend_lap380 = mask2*imagelaplot + (1-mask2)*imagelapball

# We define a function to create a Laplacian Matrix of a
   given size with "scipy."
import scipy.sparse

def Laplacian_matrix(n, m):
    mat_D = scipy.sparse.lil_matrix((m, m), dtype = int)
    mat_D.setdiag(-1, -1)
    mat_D.setdiag(8)
    mat_D.setdiag(-1, 1)

    mat_A = scipy.sparse.block_diag([mat_D] * n, dtype =
       int).tolil()

    mat_A.setdiag(-1,  1*m)
    mat_A.setdiag(-1, -1*m)
    mat_A.setdiag(-1,  1*m+1)
    mat_A.setdiag(-1, -1*m+1)
    mat_A.setdiag(-1,  1*m-1)
    mat_A.setdiag(-1, -1*m-1)

    return mat_A

from scipy.sparse.linalg import spsolve, cg
```

```
mat_A380 = Laplacian_matrix( *blend_lap380.shape[:-1] ).
   tocsc()
# tocsc() converts matrix to Compressed Sparse Column
   format

# solve for each level of the three color values (RGB)
ImgChans380 = []
for i in range(1):
    mat_b380 = blend_lap380[:,:,i].flatten()
    ImgChans380.append( spsolve(mat_A380, mat_b380) )
    print(i)

# blends three images into one
Blended380 = np.stack( [ img.reshape(blend_lap380.shape
   [:-1]) for img in ImgChans380], axis = 2)
Blended380

# plot output
Blended380 = 1 - (Blended380 - Blended380.min())/(
   Blended380.max() - Blended380.min())

fig,ax = plt.subplots(1,3,figsize = (20,20))
ax[0].imshow(plane)
ax[0].set_title('Source')
ax[1].imshow(emptyLot)
ax[1].set_title('Target');
ax[2].imshow(Blended380)
ax[2].set_title('Poisson␣Blended');
```

A.2 Calculating the Eigenvector of a Natural Random Walk

```
#Import Packages

%matplotlib inline
from matplotlib import pyplot as plt

import numpy as np
import pandas as pd
import networkx as nx
import torch
import kornia
import scipy.sparse
import os
from scipy.sparse.linalg import spsolve, cg
np.set_printoptions(threshold=np.inf)

import tensorly as tl
```

```
from numpy.linalg import matrix_power

# simple case of a natural walk
Pnat = np.array([[1/4, 1/4, 1/4, 1/4],
                 [1/3, 1/3, 1/3, 0],
                 [0, 1/3, 1/3, 1/3],
                 [1/3, 0, 1/3, 1/3]])

Pnat, Pnat.sum(axis = 1)

# results from the simple case
res = np.linalg.matrix_power(Pnat, 100)
res, Pnat.T @ res[0,:]

# vector valued case
P2 = np.kron(Pnat, np.eye(2))
# change values to add mixing terms
P2[0,1] = 0.05
P2[0,0] = 0.2
P2

# high powers result
res2 = np.linalg.matrix_power(P2, 500)
res2, res2[0,:], res2[1,:]

# showing the results are eigenvectors
P2.T @ res2[0,:], P2.T @ res2[1,:]
```