TCNJ JOURNAL OF STUDENT SCHOLARSHIP VOLUME XI APRIL, 2009

HIGHER ORDER TENSOR OPERATIONS AND THEIR APPLICATIONS

Authors:
Emily Miller, The College of New Jersey
Scott Ladenheim, Syracuse University

Faculty Sponsor:
Carla Martin
Department of Mathematics and Statistics, James Madison University

ABSTRACT

More and more, real data obtained from experiments, databases, or samples is multi-dimensional in nature. Traditionally, such data is analyzed using two-dimensional methods, the versatile tools of linear algebra. However, there is a mathematical need to extend linear algebra to higher dimensions in order better to analyze, compress, or otherwise manipulate such multidimensional data. A matrix, the building block of linear algebra, is two-dimensional (rows and columns). A tensor is the higher dimensional equivalent and is considered an n-dimensional array of data. Extending linear algebra to tensors is highly non-trivial. In this paper, we give one possible extension of the most ubiquitous linear algebra factorization: the Singular Value Decomposition (SVD). First, we define the analogous operations of addition, multiplication, inverse and transpose for tensors in such a way that the set of $n \times n \times n$ invertible tensors forms an algebraic group under addition and multiplication. From these properties, we define the SVD for a tensor. We utilize the tensor SVD in two applications. The first uses the tensor SVD to compress a video file. This results in a file of reduced size, but without a loss in the original video's quality. The second application is a handwritten digit recognition algorithm. When given a sample digit, the algorithm recognizes and returns the correct value of the input digit with a high level of accuracy.

1 INTRODUCTION

Matrices have long been viewed as the basis of linear algebra for their unique properties and many useful applications in other fields of mathematics. However, matrices are limited because only two dimensions of data can be manipulated at any given time. Tensors, however, allow for much greater freedom in working with multidimensional data sets. A tensor is defined as an n-dimensional array of data, with $n > 2$. If we consider a vector $v \in \mathbb{R}^{n_1}$, and a matrix $A \in \mathbb{R}^{n_1 \times n_2}$, then a third order tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ can be thought of as a "cube" of data. These data cubes can be visualized as a collection of matrices, where each matrix forms a face or layer of the tensor.

An n-dimensional tensor is indexed by n indices. For example, if $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ then the ijk^{th} element is referred to as $a_{i,j,k}$. To perform the computations necessary for this paper, the program, Matlab,[1] was used extensively. Throughout this paper we refer to third and higher order tensors using this notation. For example, if $\mathcal{A} \in \mathbb{R}^{3 \times 3 \times 3}$, then the third face of this tensor is denoted as $\mathcal{A}(:,:,3)$. In a more general case, the ith face of third order tensor $\mathcal{A}$ is $\mathcal{A}(:,:,i)$. Finally, to avoid any confusion amongst arrays of differing dimensions, we use script letters to denote tensors, capital letters to denote matrices, and lower case letters to denote vectors and scalars.

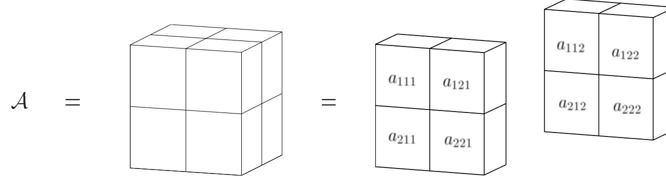


Figure 1: Visualizing a Tensor

2 TENSOR OPERATIONS

In this paper, we will concentrate on tensors. However, many of the concepts we will discuss are extensions of basic matrix operations and linear algebra concepts that can be found in a linear algebra textbook. We used Strang's *Linear Algebra and Its Applications* extensively. [4]

Basic Tensor Operations

Tensor Addition

Similar to matrices, two tensors $\mathcal{A}$ and $\mathcal{B}$ can be added if and only if their respective dimensions are equal. The following definition tells us how to add two tensors.

Definition 2.1. Given $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\mathcal{B} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$

$$\mathcal{A} + \mathcal{B} = \mathcal{C}$$

where $\mathcal{C} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $c_{i,j,k} = a_{i,j,k} + b_{i,j,k}$

Tensor Multiplication

Tensor multiplication, however, is not as straightforward as addition. The multiplication of two third order tensors $\mathcal{A}$ and $\mathcal{B}$ is computed as AB, where A is the block circulant matrix formed from the consecutive faces of $\mathcal{A}$, and B is the block column vector formed by consecutive faces of $\mathcal{B}$. For example, if $A_i = \mathcal{A}(:,:,i)$ and similarly for B_i then,

$$A = \begin{bmatrix} A_1 & A_n & A_{n-1} & \dots & A_3 & A_2 \\ A_2 & A_1 & A_n & \dots & A_4 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_n & A_{n-1} & A_{n-2} & \dots & A_2 & A_1 \end{bmatrix} \quad B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix}$$

Definition 2.2. Given $\mathcal{A}$ and $\mathcal{B}$

$$\mathcal{A} * \mathcal{B} = AB = \begin{bmatrix} A_1B_1 + A_nB_2 + \cdots + A_2B_n \\ A_2B_1 + A_1B_2 + \cdots + A_3B_n \\ \vdots \\ A_nB_1 + A_{n-1}B_2 + \cdots + A_1B_n \end{bmatrix}$$

Each block of the resulting block column vector forms a face of the tensor resulting from the multiplication of $\mathcal{A}$ and $\mathcal{B}$. Like matrices, the operation of tensor multiplication places some limitations on the size of the tensors to be multiplied. So we provide the following lemma.

Lemma 2.1. *Two tensors $\mathcal{A}$ and $\mathcal{B}$ can be multiplied if and only if $\mathcal{A} \in \mathbb{R}^{a\times b\times c}$ and $\mathcal{B} \in \mathbb{R}^{b\times d\times c}$. This multiplication yields $\mathcal{A} * \mathcal{B} = \mathcal{C} \in \mathbb{R}^{a\times d\times c}$*

Remark. From the definition of tensor multiplication and since matrix multiplication is itself an associative operation, it should be clear that tensor multiplication is an associative operation; thus:

$$(\mathcal{A} * \mathcal{B}) * \mathcal{C} = \mathcal{A} * (\mathcal{B} * \mathcal{C})$$

Tensor Identity and Inverse

Definition 2.3. The third order identity tensor, $\mathcal{I} \in \mathbb{R}^{m\times m\times n}$ is the tensor whose first face is the $m \times m$ identity matrix I, and whose subsequent faces are $m \times m$ zero matrices. The identity tensor, represented as a block column vector is:

$$\mathcal{I} = \begin{bmatrix} I \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

From this definition it is easy to see that $\mathcal{A} * \mathcal{I} = \mathcal{A}$ and $\mathcal{I} * \mathcal{A} = \mathcal{A}$

$$\mathcal{A} * \mathcal{I} = \begin{bmatrix} A_1 & A_m & A_{m-1} & \dots & A_3 & A_2 \\ A_2 & A_1 & A_m & \dots & A_4 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_m & A_{m-1} & A_{m-2} & \dots & A_2 & A_1 \end{bmatrix} * \begin{bmatrix} I \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} (A_1 \cdot I) + (A_m \cdot 0) + (A_{m-1} \cdot 0) + \dots + (A_3 \cdot 0) + (A_2 \cdot 0) \\ (A_2 \cdot I) + (A_1 \cdot 0) + (A_m \cdot 0) + \dots + (A_4 \cdot 0) + (A_3 \cdot 0) \\ \vdots \\ (A_m \cdot I) + (A_{m-1} \cdot 0) + (A_{m-2} \cdot 0) + \dots + (A_2 \cdot 0) + (A_1 \cdot 0) \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \mathcal{A}$$

Definition 2.4. An $n \times n \times n$ tensor $\mathcal{A}$ has an inverse $\mathcal{B}$ if and only if

$$\mathcal{A} * \mathcal{B} = \mathcal{I} \ \ and \ \ \mathcal{B} * \mathcal{A} = \mathcal{I}$$

An Important Remark

From the previous definitions, it follows that the set of all invertible $n \times n \times n$ tensors forms an algebraic group under the operations of addition and multiplication. For further information on groups and their properties, reference Papantonopoulou's *Algebra: Pure & Applied*. [2]

Tensor Transpose

Definition 2.5. The transpose of a third order tensor $\mathcal{C}$ is found by preserving the first face and then reversing the sequence of the subsequent faces of $\mathcal{C}$. $\mathcal{C}^{\mathcal{T}}$ can be represented as a block column vector in the following way:

$$C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_n \end{bmatrix} \qquad C^T = \begin{bmatrix} {C_1}^T \\ {C_n}^T \\ {C_{n-1}}^T \\ \vdots \\ {C_2}^T \end{bmatrix}$$

Orthogonal Tensors

Definition 2.6. A tensor, $\mathcal{Q} \in \mathbb{R}^{nxnxp}$, is orthogonal if and only if

$$\mathcal{Q} * \mathcal{Q}^{\mathcal{T}} = \mathcal{Q}^{\mathcal{T}} * \mathcal{Q} = \mathcal{I}.$$

In addition,

$$||\mathcal{Q} * \mathcal{A}||_F = ||\mathcal{A}||_F$$

where $||\mathcal{A}||_F$ is the Frobenius norm and is defined for $n \times m \times p$ tensors $\mathcal{A}$ as:

$$||\mathcal{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p} {a_{ijk}}^2}$$

Higher Order Tensor Operations

The operations defined in the previous sections are applicable only to third order tensors. However, it is possible to apply these concepts to higher dimensional tensors. The definitions for these higher order operations follow directly from the definitions in the previous section, except that they are performed in a recursive manner. For example, to multiply two fourth order tensors, we must first decompose them into third order tensors, and then multiply as defined earlier. Similarly, to multiply two fifth order tensors, we must decompose them first into fourth order tensors and then into third order tensors before multiplying. Higher order tensor multiplication and higher order tensor transposition both follow recursive processes. For clarity, in the following applications we focus only on the 3-dimensional case.

Matrix Singular Value Decomposition

Before we proceed into applications using our tensor operations, we first provide an application example of a very powerful matrix factorization, which we will later extend to tensors. This factorization is the singular value decomposition.

The singular value decomposition of an $m \times n$ matrix, A, is given by:

$$A = U \Sigma V^T = \sum_{i=1}^{r} \sigma_i U_i {V_i}^T \tag{1}$$

where U is an orthogonal $m \times m$ matrix formed by the unit eigenvectors of $A^T A$, Σ is the diagonal $m \times n$ matrix containing the singular values of $A^T A$ in decreasing order, and V^T is an orthogonal $n \times n$ matrix formed by the unit eigenvectors of $A^T A$. Every singular value in the Σ matrix is greater than or equal to zero. Furthermore, in the sum, r is the rank, U_i and V_i represent the ith columns of U and V respectively, and σ_i represents the ith singular value.

This decomposition provides useful information regarding the matrix A and has important applications to matrix computations. For example, the number of nonzero singular values of a given matrix specifies the rank, r of the matrix. The matrix SVD is useful in solving homogeneous linear equations, total least squares minimization, signal processing, pattern recognition, matrix approximation, and finding the range, null space, and column space of A.

Matrix SVD Applications

An application of the matrix SVD can be found in reducing the size of an image file without losing the original picture's visual quality. Since an image file can be treated as a data array, it can be transformed into a matrix of which the SVD can be computed. From this computation, the singular value matrix can be obtained. Generally, for the $m \times n$ case,

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_{n-1} & 0 \\ 0 & 0 & \dots & 0 & \sigma_n \end{bmatrix}$$

By plotting the singular values, a distinct drop can be seen such that after a certain σ_i

$$\sigma_i \gg \sigma_{i+1}, \sigma_{i+2}, \dots, \sigma_{n-1}, \sigma_n$$

These large singular values correspond to the dominant features of the original image file. Thus, eliminating the smallest singular values $(\sigma_{i+1}, \dots, \sigma_n)$ yields a compressed image that is of comparable quality. This is equivalent to choosing a $k < r$ and computing the SVD of a matrix A as a sum, yielding a close approximation.

$$A = \sum_{i=1}^{r} \sigma_i U_i V_i^T \approx \sum_{i=1}^{k} \sigma_i U_i V_i^T$$

The following example should remove any ambiguity of this process. By starting with an 800×800 pixel grayscale jpeg image file, we are able to create a 800×800 matrix containing the color data of the image. We then perform the SVD on the matrix and plot the singular values to note the point at which they begin to decrease rapidly. As you can see in Figure 2, this drop is recognizable even by plotting the first 25 singular values. By specifying a rank, r, about the value at which the singular values begin rapidly to decrease, we can remove underlying data that does very little for the quality of the original photo. This yields the optimal result: storing the smallest amount of data possible, while still maintaining as much of the original quality of the picture as possible.

If we were to begin with a 800×800 pixel image,[2] we could form an 800×800 matrix of the image data. Using the technique described above and choosing k=100, we are able to compress the original image and store it in a 100×100 matrix, thus cutting the amount of data stored by more than half. When a

suitable rank is chosen, the image returned is compressed in size, but very little is lost in the quality of the image. The original 800×800 pixel image is presented in Figure 3. When the full rank is used (k=800), the original image is returned, as in Figure 4(a). The images from Figure 4(b-e) present additional low rank approximations of the original image. There is a perceptible difference in quality as progressively lower ranks are used to reformulate the image. If the rank chosen is below a suitable level, the image will appear blurry and disjointed, as in Figures 4(d) and 4(e).

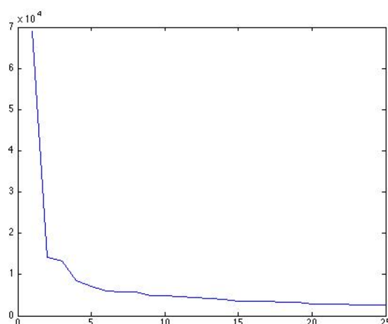


Figure 2: Singular Values

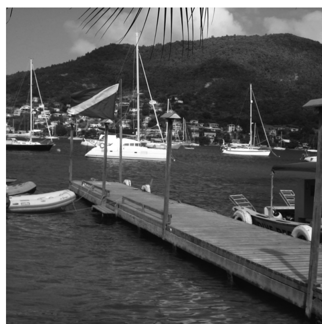
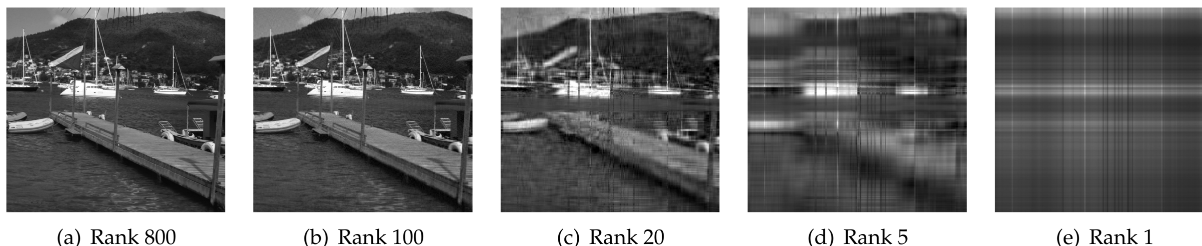


Figure 3: Original Image



(a) Rank 800 (b) Rank 100 (c) Rank 20 (d) Rank 5 (e) Rank 1

Figure 4: Low Rank Approximations of Original Image

Tensor Singular Value Decomposition

Similar to the SVD of a matrix, we can extend this concept to apply to tensors. The Tensor SVD is defined as follows:

Definition 2.7. Assume $\mathcal{A} \in \mathbb{R}^{m\times n\times p}$, then $\mathcal{A}$ can be decomposed into: $\mathcal{A} = \mathcal{U}\times\mathcal{S}\times\mathcal{V}^{\mathcal{T}}$ where $\mathcal{U} \in \mathbb{R}^{m\times m\times p}$, $\mathcal{S} \in \mathbb{R}^{m\times n\times p}$ and $\mathcal{V}^{\mathcal{T}} \in \mathbb{R}^{n\times n\times p}$. $\mathcal{U}$ and $\mathcal{V}$ are orthogonal tensors. Each face of the tensor $\mathcal{S}$ contains a matrix of singular values which decreases down the diagonal.

The result was obtained from the following process.
Assume that $\mathcal{A}$ is an $m \times n \times p$ tensor. Then,

$$(F \otimes I_m) \begin{bmatrix} A_1 & A_p & A_{p-1} & \dots & A_3 & A_2 \\ A_2 & A_1 & A_p & \dots & A_4 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_p & A_{p-1} & A_{p-2} & \dots & A_2 & A_1 \end{bmatrix} (F^* \otimes I_n) = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_p \end{bmatrix}$$

where F is the $p \times p$ discrete Fourier transform matrix, I_m is the $m \times m$ identity matrix, F^* is the conjugate transpose of F, and I_n is the $n \times n$ identity matrix. D is a block diagonal matrix formed by multiplying the Kronecker product of F and I_m by the block circulant of A by the Kronecker product of F^* and I_n. This implies that:

$$\begin{bmatrix} A_1 & A_p & A_{p-1} & \dots & A_3 & A_2 \\ A_2 & A_1 & A_p & \dots & A_4 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_p & A_{p-1} & A_{p-2} & \dots & A_2 & A_1 \end{bmatrix} = (F^* \otimes I_n) \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_p \end{bmatrix} (F \otimes I_m)$$

By taking the SVD of each D_i, we see that:

$$\begin{bmatrix} A_1 & A_p & A_{p-1} & \dots & A_3 & A_2 \\ A_2 & A_1 & A_p & \dots & A_4 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_p & A_{p-1} & A_{p-2} & \dots & A_2 & A_1 \end{bmatrix} = (F^* \otimes I_n) \begin{bmatrix} U_1\Sigma_1 V_1^T & & & \\ & U_2\Sigma_2 V_2^T & & \\ & & \ddots & \\ & & & U_p\Sigma_p V_p^T \end{bmatrix} (F \otimes I_m)$$

$$= (F^* \otimes I_n) \begin{bmatrix} U_1 & & & \\ & U_2 & & \\ & & \ddots & \\ & & & U_p \end{bmatrix} \begin{bmatrix} \Sigma_1 & & & \\ & \Sigma_2 & & \\ & & \ddots & \\ & & & \Sigma_p \end{bmatrix} \begin{bmatrix} V_1^T & & & \\ & V_2^T & & \\ & & \ddots & \\ & & & V_p^T \end{bmatrix} (F \otimes I_m)$$

By multiplying on the left and right of each block diagonal matrix above by the appropriate DFT matrices, the results are the $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$ block circulant matrices. Taking the first column of each, we reconstitute the faces to form the $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$ tensors.

$$= (F^* \otimes I_m) \begin{bmatrix} U_1 & & \\ & \ddots & \\ & & U_p \end{bmatrix} (F \otimes I_m)(F^* \otimes I_n) \begin{bmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_p \end{bmatrix} (F \otimes I_n)(F^* \otimes I_n) \begin{bmatrix} V_1^T & & \\ & \ddots & \\ & & V_p^T \end{bmatrix} (F \otimes I_n)$$

Important to note is that the tensor $\mathcal{S}$ has decreasing values along the diagonal of each face; furthermore, the singular values of the first face are larger than those of the second face, which are larger than those of the third. In general, the singular values of the ith face are greater than the singular values of the $i+1$ face.

The Fast Tensor SVD Algorithm

Performing the tensor SVD on large data sets, because of the tensor SVD definition, results in significant memory usage and long run times. In order to rectify this problem, we present the Fast Tensor SVD algorithm, which utilizes a Fast Fourier Transform.[1] As the name implies, the Fast Fourier Transform is computationally much faster than the use of discrete Fourier transform matrices. We give an example where our tensor $\mathcal{T} \in \mathbb{R}^{3\times3\times3}$, then extend the algorithm for $\mathcal{T} \in \mathbb{R}^{n\times n\times n}$. But first, a useful lemma.

Lemma 2.2. *Let FA=fft(A)*
Consider FAF^T

$$\begin{aligned} FAF^T &= F(((AF^T)^T)^T) \\ &= F((FA^T)^T) \\ &= F((\mathit{fft}(A^T)^T) \\ &= \mathit{fft}((\mathit{fft}(A^T)^T) \end{aligned}$$

Consider $\mathcal{T} \in \mathbb{R}^{3\times3\times3}$; the three faces of this tensor are:

$$\begin{bmatrix} t_{111} & t_{121} & t_{131} \\ t_{211} & t_{221} & t_{232} \\ t_{311} & t_{321} & t_{331} \end{bmatrix} \begin{bmatrix} t_{112} & t_{122} & t_{132} \\ t_{212} & t_{222} & t_{232} \\ t_{312} & t_{322} & t_{332} \end{bmatrix} \begin{bmatrix} t_{113} & t_{123} & t_{133} \\ t_{213} & t_{223} & t_{233} \\ t_{313} & t_{323} & t_{333} \end{bmatrix}$$

Note. For t_{ijk}, k designates the face of the tensor.
The algorithm works as such: First form a vector from the $(1,1)$ entry of each face; call this vector t; thus

$$t = \begin{bmatrix} t_{111} & t_{112} & t_{113} \end{bmatrix}$$

Create a circulant matrix from this vector t,

$$T = \begin{bmatrix} t_{111} & t_{113} & t_{112} \\ t_{112} & t_{111} & t_{113} \\ t_{113} & t_{112} & t_{111} \end{bmatrix}$$

From here, apply Lemma 2.2 to T. This will diagonalize the circulant matrix, creating $\hat{T}$:

$$\hat{T} = \begin{bmatrix} \hat{t}_{111} & 0 & 0 \\ 0 & \hat{t}_{112} & 0 \\ 0 & 0 & \hat{t}_{113} \end{bmatrix}$$

The diagonal entries of $\hat{T}$ then become the entries in a newly stored tensor $\mathcal{D}$, where $\hat{t}_{111}$ =d_{111}, $\hat{t}_{112}$=d_{112} and $\hat{t}_{113}$=d_{113}. Repeating this process for all possible (i,j) in $\mathcal{T}$, we obtain $\mathcal{D} \in \mathbb{R}^{3\times3\times3}$.

Note. The kth face of $\mathcal{D}$ is the same as the kth matrix of the block diagonal matrix in Definition 2.7. This technique is applied in order to save computational space.

With tensor $\mathcal{D}$, compute the SVD for each face, and store each U_i, Σ_i and V_i^T as a face of new tensors $\hat{\mathcal{U}}$, $\hat{\mathcal{S}}$ and $\hat{\mathcal{V}^{\mathcal{T}}}$. This corresponds to Definition 2.7, but rather than storing block diagonal matrices, with all the additional zero matrices, we use tensors in order to save computational space. We label these stored tensors with hats to remove ambiguity, since they are not the final $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$. At this point, we are dealing with $\hat{\mathcal{U}}$, $\hat{\mathcal{S}}$ and $\hat{\mathcal{V}^{\mathcal{T}}}$, each of which is an element of $\mathbb{R}^{3\times3\times3}$. To obtain our final $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$ tensors, first consider $\hat{\mathcal{U}}$ (the process for obtaining $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$ from $\hat{\mathcal{S}}$ and $\hat{\mathcal{V}^{\mathcal{T}}}$ is exactly the same). The faces of $\hat{\mathcal{U}}$ are:

$$\begin{bmatrix} \hat{u}_{111} & \hat{u}_{121} & \hat{u}_{131} \\ \hat{u}_{211} & \hat{u}_{221} & \hat{u}_{232} \\ \hat{u}_{311} & \hat{u}_{321} & \hat{u}_{331} \end{bmatrix} \begin{bmatrix} \hat{u}_{112} & \hat{u}_{122} & \hat{u}_{132} \\ \hat{u}_{212} & \hat{u}_{222} & \hat{u}_{232} \\ \hat{u}_{312} & \hat{u}_{322} & \hat{u}_{332} \end{bmatrix} \begin{bmatrix} \hat{u}_{113} & \hat{u}_{123} & \hat{u}_{133} \\ \hat{u}_{213} & \hat{u}_{223} & \hat{u}_{233} \\ \hat{u}_{313} & \hat{u}_{323} & \hat{u}_{333} \end{bmatrix}$$

Take a vector from the $(1,1)$ entry of each face, obtaining:

$$\hat{u} = \begin{bmatrix} \hat{u}_{111} & \hat{u}_{112} & \hat{u}_{113} \end{bmatrix}$$

Now, create a diagonal matrix $\hat{U}$ from the entries of this vector, i.e.:

$$\hat{U} = \begin{bmatrix} \hat{u}_{111} & 0 & 0 \\ 0 & \hat{u}_{112} & 0 \\ 0 & 0 & \hat{u}_{113} \end{bmatrix}$$

With this diagonal matrix, perform a back FFT, to recreate the circulant matrix U:

$$U = \begin{bmatrix} u_{111} & u_{113} & u_{112} \\ u_{112} & u_{111} & u_{113} \\ u_{113} & u_{112} & u_{111} \end{bmatrix}$$

Store the $(1,1)$ entry of U as the $(1,1)$ entry on the first face of the tensor $\mathcal{U}$. The $(2,1)$ entry of U is then stored as the $(1,1)$ entry of the second face of $\mathcal{U}$. Thus the $(i,1)$ entry of U is stored as the $(1,1)$ entry of the ith face of $\mathcal{U}$. This process is repeated for all (i,j) in $\hat{\mathcal{U}}$ to obtain the final $\mathcal{U}$. This same process is applied to both $\hat{\mathcal{S}}$ and $\hat{\mathcal{V}^{\mathcal{T}}}$ to obtain our final $\mathcal{S}$ and $\mathcal{V}^{\mathcal{T}}$.

For a tensor $\mathcal{T} \in \mathbb{R}^{n\times n\times n}$, the fast tensor SVD algorithm is the same. The only difference is that the depth vectors and matrices that are formed will be of size n and $n \times n$ respectively.

3 TENSOR APPLICATIONS

Tensor Compression Algorithms

Recall that a matrix A was approximated using a truncated portion of the singular values. We utilized two algorithms that approximate a tensor in similar manners.

Algorithm 1

Let $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and suppose its SVD is given by:

$$\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T = \sum_{i=1}^{min(n_1,n_2)} \mathcal{U}(:,i,:) * \mathcal{S}(i,i,:) * \mathcal{V}(:,i,:)^T \tag{2}$$

If we choose a $k < min(n_1, n_2)$, then summing to this k, we get an approximation of $\mathcal{A}$

$$\mathcal{A} \approx \sum_{i=1}^{k} \mathcal{U}(:,i,:) * \mathcal{S}(i,i,:) * \mathcal{V}(:,i,:)^T \tag{3}$$

Algorithm 2

For certain data sets, given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, it is useful to be able to manipulate a $k_1 < n_1$ and a $k_2 < n_2$. Suppose $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and that $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$. First, obtain matrices from each tensor, $\mathcal{A}, \mathcal{U}, \mathcal{S}$ and $\mathcal{V}^T$ by summing all of the faces:

$$\begin{aligned} A &= \sum_{i}^{n_3} \mathcal{A}(:,:,i) \\ U &= \sum_{i}^{n_3} \mathcal{U}(:,:,i) \\ S &= \sum_{i}^{n_3} \mathcal{S}(:,:,i) \\ V^T &= \sum_{i}^{n_3} \mathcal{V}^T(:,:,i) \end{aligned}$$

Since both $\mathcal{U}$ and $\mathcal{V}^T$ are orthogonal tensors, U and V are orthogonal matrices. Furthermore, since the faces of $\mathcal{S}$ are all diagonal matrices, then S will also be a diagonal matrix. Utilizing these facts it should be clear that:

$$A = USV^T \Rightarrow S = U^T AV$$

Choose a $k_1 < n_1$ and $k_2 < n_2$ to truncate the previous matrix SVD in the following manner:

$$A_{compressed} = \tilde{U}\tilde{S}\tilde{V}^T$$

where $\tilde{U} = U(:, 1 : k_1)$, $\tilde{S} = S(1 : k_1, 1 : k_2)$ and $\tilde{V}^T = V^T(1, 1 : k_2)$ and now, if k_1 and k_2 are chosen wisely, $A \approx A_{compressed}$. Now, for $i = 1, 2, \ldots, n_3$ compute $\tilde{U}^T \mathcal{A}(:, :, i)\tilde{V}$, call this new tensor $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times n_3}$. Lastly, compute $\mathcal{A}_{compressed}$ which is done in the following way:

$$\mathcal{A}_{compressed} = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \tilde{U}(:, i) \circ \tilde{V}(:, j) \circ \mathcal{T}(i, j, :) \tag{4}$$

where $\mathcal{A}_{compressed} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times n_3}$, $\tilde{U} \in \mathbb{R}^{n_1 \times k_1}$, $\tilde{V} \in \mathbb{R}^{n_2 \times k_2}$ and $\circ$ denotes the outer product.

Video Compression

The use of black and white videos, which are three dimensional in nature, provides us with a unique opportunity to apply our first algorithm as a compression strategy using the tensor SVD. Each black and white video has a dimension pertaining to each of the following: height, in pixels; width, in pixels; and time, in terms of the number of frames in the video file. Therefore, each frame in the video can be viewed as an independent image. This allows us a direct correlation from our image compression algorithm, for which we used a matrix SVD, to the first algorithm, for which we can use our tensor SVD. The tensor to be compressed will be built by converting each frame of the video into a matrix containing the pertinent image information. These matrices will then be compiled in consecutive order to form the tensor, representing the original video.

We will first calculate the tensor SVD of the video tensor. Since video files tend to be large and require large amounts of data storage and computations, we utilize the fast tensor SVD algorithm to perform the needed calculations. Similar to performing image compression, we will determine a rank, *r*, so that when the extraneous singular values are omitted, there is no detectable loss in quality. To minimize the number of values kept, while optimizing the quality and percent compression, we sort each of the singular values from each frame of the video and retain only the dominant singular values. Again, to save computing time and memory space in this algorithm, we compute the compression of the tensor as a sum. Recall the following sum:

$$A_{compressed} = \sum_{i=1}^{r} U(:, i, :) \times \Sigma(i, i, :) \times V(:, i, :)^T$$

This compressed tensor, formed from the sum shown above, can now be reconstituted into a video of comparable quality to the original, but containing significantly fewer stored data entries.[3]

Handwriting Data Compression Using the SVD

In order to examine yet another application of the tensor singular value decomposition in data compression, we manipulated a data set of 7291 handwritten digits, the digits being zero through nine. Our goal was to create a recognition algorithm, so that if given a test digit, a computer would recognize and return the value of that digit. The data used in this application was gathered from the United States Postal Service. The USPS database was obtained from the authors of *Handwritten digit classification using higher order singular value decomposition*. Previous work in this area has been done by Savas and Eldén [3].

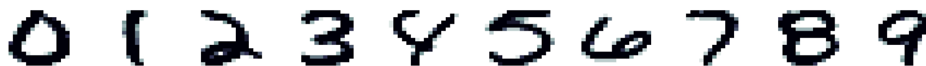


Figure 5: Sample Unblurred Digits, 0-9

Forming the Tensor

We formed our tensor from the data set by forming a 256×1194×10 tensor. Each 16×16 matrix, formed from the 16 pixel by 16 pixel digit image, was reshaped into a 256×1 column vector. Every version of each digit was then horizontally concatenated. However, this process did not form equal sized matrices that could be layered to form a tensor. The digits were not evenly distributed within the 7291 images.

0	1	2	3	4	5	6	7	8	9	Total
1194	1005	731	658	652	556	664	645	542	644	7291

Table 1: Digit Distribution in Handwriting Data

To remedy this problem, we duplicated the required number of columns for each digit containing less than 1194 versions, to fill the remaining columns. This created the desired 256×1194×10 tensor.

Blurring

In order to increase the correct recognition percentage of our algorithm, we used a Gaussian smoothing technique to blur each digit. Each digit was blurred under the two dimensional normal curve given by:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5}$$

where the mean is assumed to be at (0,0).

The heights of this curve correspond to the positional entries x and y of the point spread function we will utilize to blur the digits. This point spread function provides the weights, used later to compute a weighted average. Any point within the 5×5 square with the original point at its center will be called the neighborhood of the point, P, to be blurred. For each point in the original image data, we create a 5×5 neighborhood. The blurred point, P', is given by the weighted average of the original point P and its surrounding neighborhood, using the point spread function as the weights corresponding to the points of the original image.

This new point, P', is then placed in the same position in the new image, as in the original image. However, we need to pay close attention to the boundaries of the original image. In order for each point to have an adequate neighborhood, we expanded each original image by adding two pixel borders of white space around each 16×16 image, to form a 20×20 image. After performing the blur with the original 5×5 neighborhood, we treated the boundaries with a 3×3 neighborhood. The amount of blurring is dependent on the value of σ and also upon the position, given by x and y.

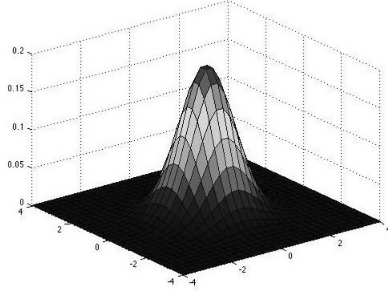


Figure 6: Normal curve in 2D, centered at (0,0)

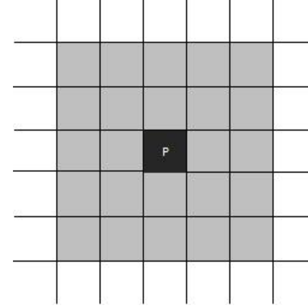


Figure 7: The neighborhood of the point, P

Recognition Algorithm

For this application, we utilize the second compression algorithm. Here, we focus on the $\mathcal{T}$ tensor. Recall that $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times n_3}$. First, we calculate the SVD of each face of the tensor. That is, for $\mu = 1, ..., n_3$, compute

$$\mathcal{T}(:, :, \mu) = U_\mu S_\mu {V_\mu}^T.$$

We now choose some $k < k_1$, at which the singular values begin to decrease rapidly, and set $B_\mu = U_u(:, [1 : k])$. Note that $B_\mu \in \mathbb{R}^{n_1 \times k}$. The digit to be recognized, d, is reshaped into a 400×1 vector and projected onto the space of pixels by letting $d_{k_1} = \tilde{U}^T d$. In order for the digit to be correctly recognized, we solve the minimization problem for

$$min_{x_\mu} ||d_{k_1} - B_\mu x_\mu||$$

We compute this using the QR decomposition of B_μ, which gives $||d_{k_1} - B_\mu {B_\mu}^T d_{k_1}||$. We compute this for each μ, and call

$$R(\mu) = ||d_{k_1} - B_\mu {B_\mu}^T d_{k_1}||_F \tag{6}$$

Since μ represents each digit, the match given by the algorithm will be denoted by the μ that results in the smallest $R(\mu)$.

Results

In order to determine the effectiveness of our recognition algorithm, we used a test set of 2007 digit images. We calculated the percent error by dividing the number of digits that were unrecognizable by our algorithm by the number of digits in the test set. The percent data reduction was calculated in the following manner:
The unblurred results were obtained from:

$$1 - \frac{k_1 \times k_2 \times 10}{256 \times 1194 \times 10} \tag{7}$$

The blurred results were obtained from:

$$1 - \frac{k_1 \times k_2 \times 10}{400 \times 1194 \times 10} \tag{8}$$

Table 2: Unblurred Digit Recognition

k_1/k_2	32	48	64
32	4.26% (99.67%)	4.18%(99.50%)	4.03% (99.33%)
48	3.68% (99.50%)	3.88%(99.25%)	3.81% (99.00%)
64	3.39% (99.33%)	3.70%(99.00%)	3.73% (98.66%)

Percent error and (percent data reduction) for selected values of k_1 and k_2.

Table 3: Blurred Digit Recognition

k_1/k_2	32	48	64
32	3.50% (99.79%)	3.42% (99.68%)	3.63% (99.57%)
48	3.25% (99.68%)	3.04% (99.52%)	3.10% (99.36%)
64	3.13% (99.57%)	3.00% (99.36%)	3.07% (99.14%)

Percent error and (percent data reduction) for selected values of k_1 and k_2.

The results from the recognition algorithm on the test set are shown in the tables above.

As you can see, we were able to achieve error rates of under four percent, while compressing the amount of data stored by over 99 percent.

4 CONCLUSIONS

Through our work with tensor operations, we have seen some very interesting results. First, from a theoretical aspect, because of the way in which we defined the operations of addition, multiplication, identity and inverse, and because of the associativity property of tensors, we were able to show that the set of all $n \times n \times n$ invertible tensors forms a group under the operations of addition and multiplication. We then extended the very useful and very powerful factorization of matrices, the singular value decomposition, to tensors, which allowed us to apply tensor operations to real data. In our first compression algorithm, we used the tensor SVD to compress the third order tensor formed from a video file. We were able greatly to reduce the amount of data stored while maintaining much of the original quality of the video. Finally, we saw an application in handwritten digit recognition. We used the tensor SVD to compress a tensor formed by handwritten digit data and then ran a recognition algorithm on the data. We were able to recognize the correct digit in over 95% of cases, while compressing the data by over 99%. Through these applications, we can see that working with tensors allows much greater freedom in data manipulation, as opposed matrices, which limit us to two dimensions of data. As computing power increases, work with tensors is becoming more and more manageable and vital to advancements in data analysis.

Future Work

In the future, we would like to continue working with tensors and adding efficiency to our computer algorithms, with which we performed our applications. Although the Fast Tensor SVD algorithm improved our computing time, it is still not where we would like it to be to test applications on a large scale. We would also like to extend our applications to fourth order and higher tensors of data. We provided two examples of third order tensor applications in this paper, but we have not yet tried to perform the same kind of analysis on a fourth order or higher tensor.

Acknowledgements

We would like to thank Professor Carla Martin, our adviser, for her help in this project. Her knowledge and her patience were endless and were crucial to our completion of this research. We would like to thank the Mathematics and Statistics Department at James Madison University for hosting the Research Experience for Undergraduates where this work was completed. We would also like to thank Professor Thomas Hagedorn from the Department of Mathematics and Statistics at The College of New Jersey for his help in the finalization and submission of this paper. Finally, we thank the National Science Foundation, for funding the project and making the research possible.
This research work was supported by NSF Grant: NSF-DMS 0552577.

References

[1] Germund Dahlquist and Åke Björck *Numerical Methods*, (Prentice Hall, Englewood Cliffs, NJ, 1974).

[2] Aigli Papantonopoulou *Algebra: Pure & Applied*, (Prentice Hall, Upper Saddle River, NJ, 2002).

[3] Berkant Savas and Lars Eldén, *Handwritten digit classification using higher order singular value decomposition*, (Pattern Recognition, Volume 40, Issue 3, March 2007, Pages 993-1003).

[4] Gilbert Strang, *Linear Algebra and Its Applications*, 4th *Ed.*, (Brooks Cole, Wellesley, MA, 2005).

Endnotes

[1] The MathWorks, Inc.
[2] Photograph taken by Scott Ladenheim.
[3] Original and compressed videos are found at:
http://www.math.jmu.edu/~carlam/research/videocompression.html