CS168: The Modern Algorithmic Toolbox
Lecture #10: Tensors, and Low-Rank Tensor Recovery

Tim Roughgarden & Gregory Valiant*

May 2, 2024

Last lecture discussed singular value decomposition (SVD), and we saw how such decompositions reveal structure about the matrix in question, allowing us to possibly de-noise the matrix, compress the matrix, fill in missing entries, etc. This lecture is about tensors, and we will see an analog of this sort of decomposition that is, in a quantifiable sense, much stronger than the matrix analog.

1 Introduction to Tensors

Tensor methods for data science are relatively new, and there is ongoing research on how to leverage tensor methods to extract features and structure from a dataset.

A tensor is just like a matrix, but with more dimensions:

Definition 1.1 A $n_1 \times n_2 \times \ldots \times n_k$ k-tensor is a set of $n_1 \cdot n_2 \cdot \ldots \cdot n_k$ numbers, which one interprets as being arranged in a k-dimensional hypercube. Given such a k-tensor, A, we can refer to a specific element via $A_{i_1,i_2,\ldots,i_k}$.

A 2-tensor is simply a matrix, with $A_{i,j}$ referring to the i,jth entry. You should think of a $n_1 \times n_2 \times n_3$ 3-tensor as simply a stack of n_3 matrices, where each matrix has size $n_1 \times n_2$. The entry $A_{i,j,k}$ of such a 3-tensor will refer to the i,jth entry of the kth matrix in the stack.

Remark 1.2 For our purposes (and in most computer science applications involving data), the above definition of tensors suffices. Tensors are very useful in physics, in which case they are viewed as more geometric objects, and are endowed with some geometric notion of what it means to change the coordinate system. We won't worry about this, though be aware that you might come across a significantly more confusing definition of a tensor at some point. . . .

1.1 Examples of Tensors

We have all seen plenty of 2-tensors (i.e. matrices). Below we list a few examples of higher order tensors that you might encounter.

Example 1.3 (k-grams) Given a body of text, and some ordering of the set of words (for example, just alphabetical ordering, $w_1, \ldots, w_n$, we can associate a k-tensor, A, defined by setting entry $A_{i_1,\ldots,i_k}$ equal to the number of times the sequence of words $w_{i_1}, w_{i_2}, \ldots, w_{i_k}$ occurs in the text. For example, if we have n distinct words, we could extract an $n \times n \times n$ 3-tensor from a corpus of text, where $A_{i,j,k}$ represents the number of times in the corpus that the ith, jth, and kth words occur in sequence.

Example 1.4 (The Moment Tensor) Suppose we have some data $s_1, s_2, \ldots, s_n$ representing independent draws from some high-dimensional distribution in $\mathbb{R}^d$. That is, each $s_i \in \mathbb{R}^d$. The mean of this data is simply a vector of length d. The covariance matrix of this data is represented by a $d \times d$ matrix, whose i, jth entry is the empirical estimate of $\mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]$, where X_i denotes the ith coordinate of a sample from the distribution. We can also consider higher moments: the $d \times d \times d$ 3-tensor representing the third order moments, has entries $A_{i,j,k}$ representing the empirical estimate of $\mathbb{E}\left[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])[(X_k - \mathbb{E}[X_k])\right]$. This is simply given by the following expression in terms of the data $s_1, \ldots$: letting m_i, m_j, m_k denote the average value of the ith, jth, and kth components of the datapoints $s_1, \ldots,$,

$$M_{i,j,k} = \frac{1}{n} \sum_{\ell=1}^{n} (s_{\ell_i} - m_i)(s_{\ell_j} - m_j)(s_{\ell_k} - m_k),$$

where s_{ℓ_i} denotes the value in the ith dimension of datapoint s_ℓ. We can define higher order tensors analogously, corresponding to higher order moments.

Example 1.5 (Batches of Data) In applied ML circles, a cannonical $b \times n \times d$ 3-tensor corresponds to a batch of b datapoints, each of which is a length n sequence, with each element of the sequence represented as a d-dimensional vector. In the setting of a large language model, each datapoint would be a sequence of n words, and each word would be represented as a length d vector. In this case, if each of the b datapoints were selected uniformly at random from a dataset, then there would not be much structure along the b dimension.

1.2 The Rank of a Tensor

The rank of a tensor is defined analogously to the rank of a matrix. Recall that a matrix M has rank r if it can be written as $M = UV^t$, where U has r columns, and V has r columns. Letting $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$ denote these columns, we have

$$M = \sum_{i=1}^{r} u_i v_i^t.$$

Each term $u_i v_i^t$ is the *outer-product* of two vectors, and this expression represents M as a sum of r rank one matrices, where the ith matrix $B_i = u_i v_i^t$ has entries $B_{j,k} = u_i(j)v_i(k)$, namely the product of the jth entry of vector u_i and the kth entry of vector v_i.

Tensor rank is defined analogously. A tensor has rank one if it can be expressed as the outer-product of a set of vectors, and a tensor is rank k if it can be written as a sum of k rank 1 tensors. We now formally define the outer-product, and describe the notation we use to represent the vector outer-product.

Definition 1.6 Given vectors u, v, w, of lengths n, m, and ℓ, respectively, their *tensor product* or *outer product* is the $n \times m \times \ell$ rank one 3-tensor denoted $A = u \otimes v \otimes w$ with entries $A_{i,j,k} = u_i v_j w_k$.

The above definition extends naturally to higher dimensional tensors:

Definition 1.7 Given vectors $v_1, v_2, \ldots, v_k$, of lengths $n_1, n_2, \ldots, n_k$, their *tensor product* is denoted $v_1 \otimes v_2 \otimes \ldots \otimes v_k$, and represents the $n_1 \times n_2 \times \ldots \times n_k$ k-tensor A with entry $A_{i_1,i_2,\ldots,i_k} = v_1(i_1) \cdot v_2(i_2) \cdot \ldots \cdot v_k(i_k)$.

Example 1.8 For example, given

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, v_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, v_3 = \begin{pmatrix} 10 \\ 20 \end{pmatrix}, .$$

$v_1 \otimes v_2 \otimes v_3$ is a $3 \times 2 \times 2$ 3-tensor, that can be thought of as a stack of two 3×2 matrices

$$M_1 = \begin{pmatrix} -10 & 10 \\ -20 & 20 \\ -30 & 30 \end{pmatrix}, M_2 = \begin{pmatrix} -20 & 20 \\ -40 & 40 \\ -60 & 60 \end{pmatrix}.$$

We are now ready to define the rank of a tensor, which will correspond to our definition of the rank of a matrix in the case that we are referring to a 2-tensor:

Definition 1.9 A 3-tensor A has rank r if there exists 3 sets of r vectors, $u_1, \ldots, u_r$, $v_1, \ldots, v_r$ and $w_1, \ldots, w_r$ such that

$$A = \sum_{i=1}^{r} u_i \otimes v_i \otimes w_i.$$

The definition of rank for general k-tensors is analogous.

2 Differences between Matrices and Tensors

In general, most of what you know about linear algebra for matrices does NOT apply to k-tensors for $k \geq 3$. Below is a brief list of notable differences between tensors and matrices:

1. For matrices, the best rank-k approximation (where best is in the sense of the Frobenius norm–namely minimizing the sum of the element-wise squared differences) can be found by iteratively finding the best rank-1 approximation, and then subtracting it off. In other words, for a matrix M, the best rank 1 approximation of M is the same as the best rank 1 approximation of the matrix M_2 defined as the best rank 2 approximation of M. Because of this, if uv^t is the best rank 1 approximation of M, then $rank(M - uv^t) = rank(M) - 1$.

   For k-tensors with $k \geq 3$, this is not always the case. If $u \otimes v \otimes w$ is the best rank 1 approximation of 3-tensor A, it is possible that $rank(A - u \otimes v \otimes w) \geq rank(A)$.

2. For matrices with entries in $\mathbb{R}$, there is no point in looking for a low-rank decomposition in terms of vectors whose entries are complex numbers, because if the matrix has real entries, the lowest rank representation using complex numbers is the same as the rank that can be obtained using vectors with all real valued entries, namely $rank_{\mathbb{R}}(M) = rank_{\mathbb{C}}(M)$. For k-tensors, with $k \geq 3$, this is not always the case, it can be that the rank over complex vectors is smaller than the rank over real vectors, even if the entries in the tensor are real-valued.

3. We don't really know how to argue about the rank of 3-tensors: for example, with probability 1, if you pick the entries of an $n \times n \times n$ 3-tensor independently at random from the interval $[0, 1]$, the rank will be on the order of n^2, however we don't know how to describe any explicit construction of $n \times n \times n$ tensors whose rank is greater than $n^{1.1}$, for all n.

4. Computing the rank of matrices is computationally easy (ie polynomial time via computing the via SVD). Computing the rank of 3-tensors is NP-hard.

5. As we will explore in the following section, despite the above point, if a 3-tensor has factors that are linearly independent (note: this is a much WEAKER assumption than requiring them to be orthogonal), then 1) its rank can be efficiently computed, 2) its low-rank representation is *unique*, 3) this representation can be efficiently recovered. This fact underlies many of the compelling use-cases for tensors and tensor factorization.

3 Low-Rank Tensors

Recall that a low-rank representation of a matrix M, is *not* unique. For $M = UV^t$, where both U and V have r columns, for any $r \times r$ invertible matrix C, we have $M = UCC^{-1}V^t = (UC)(C^{-1}V^t)$, and hence the columns of UC, and the rows of $C^{-1}V^t$ form a different rank r representation of M. This lack of uniqueness of low-rank representations is frustrating if we hope to interpret the various factors.

One of the earlier pioneers of low-rank approximation of matrices was the British psychologist/statistician Charles Spearman. One of his early experiments was to give a number

of academic tests to a number of students, and form the matrix M in which entry $M_{i,j}$ represented the performance of the ith student on the jth test. He realized that M was very close to a rank 2 matrix, and went on to conjecture that this might arise via the following explanation: suppose the ith student has two number m_i, v_i representing their mathematical ability and verbal ability. Suppose further that the jth test can basically be represented as two number, t_j, q_j representing that tests' mathematical and verbal components. If this model were correct, then $M_{i,j} \approx m_i t_j + v_i q_j$, and hence M would be close to the rank 2 matrix UV^t, where the two columns of U represent the students' math/verbal abilities, and the two columns of V represent the tests' math/verbal components. Unfortunately, the rank-2 representation is not unique, as mentioned in the previous paragraph, and hence even if this model of the world were true, the rank 2 representation recovered would not correspond to this model.

Amazingly, once one goes from 2-tensors (matrices) to 3-tensors, low-rank decompositions end up being essentially unique!

Theorem 3.1 *Given a 3-tensor A of rank k s.t. there exists three sets of linearly independent vectors, $(u_1, \ldots, u_k), (v_1, \ldots, v_k), (w_1, \ldots, w_k)$, s.t.*

$$A = \sum_{i=1}^{k} u_i \otimes v_i \otimes w_i,$$

then this rank k decomposition is unique (up to scaling the vectors by a constant), and these factors can be efficiently recovered.

To give a simple illustration of the above theorem, suppose we conducted Spearman's experiment, except we added an extra dimension—suppose we administered each test to each student in one of three different settings (i.e. a setting in which classical music is playing, a setting with a distracting video playing, and a control setting). Let M denote the corresponding 3-tensor, with $M_{i,j,k}$ denoting the performance of student i on test j in setting k. Suppose the true model of the world is as follows: as above, for every student there are two numbers representing their math/verbal ability, and every test can be regarded as having a math/verbal component; additionally, for each setting, there is some scaling of the math performance resulting from that setting, and a scaling of the verbal performance resulting from that setting. Hence $M_{i,j,k}$ can be approximated by multiplying the math ability of the student with the math component of the test and the math boost-factor of the setting, and then adding the corresponding product from the verbal components. Theorem 3.1 asserts that, provided the vector of student math abilities is not identical (up to a constant rescaling) to the vector of verbal abilities, and the 2 vectors of math/verbal test components are not identical up to rescaling, and the 2 vectors of math/verbal setting boosts are not identical up to rescaling, then this is the unique factorization of this tensor, and we will be able to recover these exact factors.

3.1 Quick Discussion

As mentioned in previous lectures, one can often interpret the top two or three singular vectors of a dataset. Perhaps the main reason that the 4th, 5th, etc. singular vectors are often less interpretable, is that—because they must be orthogonal to the first components—they end up not representing the clean/interpretable phenomena that one might hope. The beauty of Theorem 3.1 is that the factors do not need to be orthogonal; as long as they are linearly independent, then they can be recovered uniquely. More broadly, tensor methods offer one hope for enabling useful features to be extracted from data in an unsupervised setting.

3.2 The Algorithm

We now describe the algorithm alluded to in Theorem 3.1. This algorithm was originally proposed in the 1970's but has been reinvented several times, in several different contexts, and is often referred to as "Jenrich's Algorithm". Do not worry too much about the details—the main step that might be unfamiliar is the computation of an eigen-decomposition of a matrix $M = QSQ^{-1}$ where S is the diagonal matrix of eigenvalues, and the columns of Q are eigenvectors. Note that in Lecture 7 and 8, we only looked at eigenvectors of a symmetric matrix XX^t, in which case $Q^t = Q^{-1}$. Here, the matrix in question, M, might not be symmetric and hence the eigenvectors need not be orthogonal to each other.

Algorithm 1
TENSOR DECOMPOSITION

Given an $n \times n \times n$ tensor $A = \sum_{i=1}^{k} u_i \otimes v_i \otimes w_i$, with $(u_1, \ldots, u_k)$, and $(v_1, \ldots, v_k)$, and $(w_1, \ldots, w_k)$ linearly independent, the following algorithm will output the lists of u's,v's, and w's.

- Choose random unit vectors $x, y \in \mathbb{R}^n$.
- Define the $n \times n$ matrices A_x, A_y, where A_x is defined as follows: consider A as consisting of a stack of n $n \times n$ matrices. Let A_x be the weighted sum of these n matrices, where the weight given to the ith matrix is x_i–namely the ith element of vector x. Define A_y analogously to be the weighted sum of the matrices, with the ith matrix being scaled by y_i.
- Compute the eigen-decompositions of $A_x A_y^{-1} = QSQ^{-1}$, and $A_x^{-1} A_y = (Y^t)^{-1} T Y^t$.
- We will show that with probability 1, the entries of diagonal matrix S will be unique, and will be inverses of the entries of diagonal matrix T. The vectors $u_1, \ldots, u_k$ are the columns of Q corresponding to nonzero eigenvalues, and the vectors $v_1, \ldots, v_k$ will be the columns of Y, where v_i corresponds to the reciprocal of the eigenvalue to which u_i corresponds.
- Given the u_i's and v_i's, we can now solve a linear system to find the w_i's, or imagine rotating the whole tensor A and repeating the algorithm to recover the w's.

Before analyzing the above algorithm, we note that if the original tensor A is $n \times m \times p$, rather than $n \times n \times n$, then the above algorithm continues to work, provided we compute the eigen-decomposition of the matrices $A_x A_y^+$ and $A_x^+ A_y$, where M^+ denotes the "pseudo-inverse" of M, which is the analog of inverses for non-square matrices.

To analyze the above algorithm, we first argue that $A_x = \sum_{i=1}^{k} \langle w_i, x \rangle u_i v_i^t$, and, similarly, $A_y = \sum_{i=1}^{k} \langle w_i, y \rangle u_i v_i^t$.

Lemma 3.2 *For A_x and A_y as defined in the algorithm,*

$$A_x = \sum_{i=1}^{k} \langle w_i, x \rangle u_i v_i^t, \text{ and } A_y = \sum_{i=1}^{k} \langle w_i, y \rangle u_i v_i^t..$$

Proof: First consider the case where $k = 1$. Hence $A = u_1 \otimes v_1 \otimes w_1$, and the ith matrix in the stack corresponding to A is $w_1(i) \cdot u_1 v_1^t$. Hence the contribution of this matrix to the matrix A_x is defined to be $x_i \cdot w_1(i) \cdot u_1 v_1^t$, and hence the matrix A_x is simply $u_1 v_1^t \sum_i x_i \cdot w(i) = \langle w_1, x \rangle u_1 v_1^t$. Since A is simply a sum of these rank 1 factors, A_x and A_y are simply the sum of the corresponding rank 1 components, weighted appropriately. ■

Given the above lemma, $A_x = UDV^t$ where the columns of U are the vectors u_i, and the columns of V are the vectors v_i, and D is a diagonal matrix with ith entry $\langle w_i, x\rangle$. Similarly, $A_y = UEV^t$, where the ith diagonal entry of E is $\langle w_i, y\rangle$. Given this,

$$A_x A_y^{-1} = UDV^t(V^t)^{-1}E^{-1}U^{-1} = U(DE^{-1})U^{-1},$$

and similarly

$$A_x^{-1}A_y = (V^t)^{-1}D^{-1}U^{-1}UEV^t = (V^t)^{-1}D^{-1}EV^t.$$

The correctness of the algorithm now follows from the uniqueness of the eigen-decomposition in the case that the eigenvalues are distinct. Without belaboring the details, for a random choice of vectors x, y, provided the u's, v's and w's are linearly independent, with probability 1 the eigenvalues of the above matrices will be distinct. Hence we can recover the list of $u's$ and the list of $v's$.

How do we match up the right u with the corresponding v? That is, we want to ensure that we know that u_1 belongs to the same factor as v_1, rather than, say, grouping u_1 with v_3. This is easy, as the eigenvalues of $A_x A_y^{-1}$ are given by the diagonal matrix DE^{-1} and hence are reciprocals of the eigenvalues of $A_x^{-1}A_y$, so if u_1 is the eigenvector of $A_x A_y^{-1}$ with largest eigenvalue λ_1, then v_1 will be the eigenvector of $A_x^{-1}A_y$ with smallest eigenvalue which will be equal to $1/\lambda_1$.