CS 270 Algorithms **Spring 2021**

Lecture 8: Tensors, Jennrich's algorithm, Independent component analysis

Lecturer: Prasad Raghavendra Scribe: Al Cheng, Kevin Lin, Ethan Qiu

8.1 Rank

We begin by defining a rank 1 matrix:

$$\text{rank 1 matrix } = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$
$$= uv^T$$

Then, the ijth entry is the product: $(uv^T)_{ij} = u_i v_j$

Working off of the definition of a rank 1 matrix, we proceed towards the definition of a general matrix:

Definition 8.1. A matrix M is rank k, if $M = \sum_{i=1}^{k} M_i$ with $rank(M_i) = 1$ and $M \neq \sum_{i=1}^{k-1} M_i$.

As such, the rank is the minimum number of rank 1 matrices you need to add to produce the rank k matrix.

8.1.1 Rank for Tensors

Definition 8.1.2. Tensor product, a rank one, third order tensor T is the tensor product of vectors u, v, w, with its entries being:

$$T_{i,j,k} = u_i v_j w_k$$

It follows then, that with u, v, w have dimensions v_1, v_2, v_3 respectively, then the dimension of T will then be $v_1 \times v_2 \times v_3$. An equivalent shorthand for tensor product, but much more commonly utilized:

$$T = u \otimes v \otimes w, \text{ for } u, v, w \in \mathbb{R}^m$$

Armed with the definition of a rank one tensor, we then proceed towards the definition of the rank of a tensor T.

Definition 8.1.3 The rank of a tensor T is the smallest k such that

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

Unfortunately, tensors turn out to be far less 'nice' than matricies.

8.1.2 Tensor Difficulties

Field Dependency of Rank

We consider the maximum rank of an $[n] \times [n] \times [n]$ tensor. This tensor has n^3 parameters in it. With a rank k tensor has $k3n$ parameters, then, the real upper bound of this n^3 parameter tensor would be $rank([n] \times [n] \times [n]) \geq \Omega(n^2)$. In general, the rank would be the dimension minus 1. However, the rank of the tensors turn out to be field-dependent.

For a real matrix M, its rank is independent upon whether if one is working over $\mathbb{R}$ or $\mathbb{C}$. However, the rank of a tensor depends on if you are working with $\mathbb{R}$ or $\mathbb{C}$. Allowing the use of real/complex numbers would impact the rank. We consider the following example from Moitra:

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

We can see that $rank_{\mathbb{R}}(T) \geq 3$, however, upon $\mathbb{C}$, we may write

$$T = \frac{1}{2}(\begin{bmatrix} 1 \\ -i \end{bmatrix} \otimes \begin{bmatrix} 1 \\ i \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -i \end{bmatrix} + \begin{bmatrix} 1 \\ i \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -i \end{bmatrix} \otimes \begin{bmatrix} 1 \\ i \end{bmatrix})$$

This adds another complication towards the rank of tensors.

Approximation of High Rank Tensors

For matrices, we consider a sequence of rank k matrices:

$$M_1, M_2, ..., M_i, ...$$

If we take the limit, $\lim_{t \to \infty} M_t$ is also a rank k matrix.

However, for tensors, there exists T such that $rank(T) >$ large n, but for all ϵ, there exists T' such that $rank(T) <$ small c, $\|T - T'\| < \epsilon$
For tensors, we can then have

$$T_1, T_2, ..., T_i, ...$$

With T_1, T_i is low rank, but its limit, $\lim_{t \to \infty} T_t$ is high rank. The means that the notion of rank is not robust, a tensor can be high rank, but we can get good approximations that are low rank.

NP-hardness of Computation

Tensor rank is NP-hard to compute. We cannot construct explicit tensor T whose rank is more than n. By explicit, we mean non-random, and certifiable high rank tensors. So it is difficult to get a deterministic algorithm that generates the entries of the matrix, generalize to all n. Especially considering that computing rank for tensors is expensive.

8.1.3 Eigendecomposition

For real symmetric matrix M, you can write $M = \sum \lambda_i v_i v_i^T$, with each v_i a eigenvector, with eigenvalue λ_i, and each v_i are orthogonal to each other.

Suppose, we have a tensor T such that

$$T = \sum_{i=1}^{n} a_i^{\otimes 3}, \text{ where } a_i^{\otimes 3} = a_i \otimes a_i \otimes a_i \text{ and } \langle a_i \rangle \text{ are orthogonal vectors.}$$

We are able to recover a_i. Consider that $T = \sum a_i \otimes a_i \otimes a_i$, we pick random $g \in \mathbb{R}^n$. We apply g to T's mode by creating new tensor $T[g,.,.] = \sum_{i=1}^{n} \langle a_i, g \rangle \cdot a_i \otimes a_i$. This operation, allows us to take a slice of T along a direction, taking a linear combination of the slices with coefficients $g_1, ...g_n$.

To further expand upon this, we can apply vectors to any subset of modes. We can do the following: $T[g,.,h] = \sum_{i=1}^{n} \langle a_i, g \rangle \cdot a_i \cdot \langle a_i, h \rangle$. This operation produces a matrix for which, we can compute eigenvectors, eigenvalues, etc.

$$\sum_{i=1}^{n} \langle a_i, g \rangle \cdot a_i \otimes a_i$$

We note that the eigenvectors are a_i. And the eigenvalues, if normalized appropriately, is $\langle a_i, g \rangle$. If a_i is a unit vector, then eigenvalues are $\langle a_i, g \rangle$, otherwise, you will pick up $\|a_i\|^3$.

$$(\sum_{j=1}^{n} \langle a_j, g \rangle \cdot a_j \otimes a_j) a_i = \lambda_i a_i$$

$$\sum_{j=1}^{n} \langle a_j, g \rangle (\cdot a_j \otimes a_j) a_i = \begin{cases} j \neq i, 0 \\ j = i, \langle a_i, g \rangle \cdot a_i \cdot \langle a_i, a_i \rangle = (\langle a_i, g \rangle \langle a_i, a_i \rangle) a_i \end{cases}$$

Then summed over all j one will obtain a multiple of a_i

One important thing to internalize regarding algorithms on Tensors, is that they are three dimensional array of numbers, but they have linear algebraic structure upon them: you can apply vectors, take linear combinations of the slices, etc.

In the special case describe above, there are multiple methods of recovering a_i. We include another:

$$T[x,x,x] = \sum T_{i,j,k} x_i x_j x_k$$

$$\text{If } T[x,x,x] = \sum a_i^{\otimes 3}, \text{ Then } \sum_{i=1}^{n} T_{i,j,k} x_i x_j x_k = \sum \langle a_i, x \rangle^3$$

If T has a eigendecomposition as described above, then the corresponding polynomial is a sum of cubes.

Theorem: $T[x,x,x]$ have local maxima on the vectors $a_1, ..., a_n$ on the unit ball. These are the only local maxima, and as such, by running gradient ascent, one can recover one of a_i.

8.2 Jennrich's Algorithm

We want to find a way to do minimum rank decomposition on tensors. Say we are given a tensor T, which can be expressed as a sum of r rank-one tensors $T = \sum_i^r u_i \otimes v_i \otimes w_i$, and we want to find the vectors $u_1, u_2, ..., u_r, v_1, v_2, ..., v_r, w_1, w_2, ..., w_r$. Unfortunately, it's impossible to recover the ordering of the vectors, so we can only hope to recover the vectors with some arbitrary reordering, and another caveat that different scalings of vectors u_i, v_i, w_i can result in the same rank-one tensor, so our recovery is reordered and vectors may be rescaled, but the rank-one tensors are the same.

Algorithm 1 Jennrich's Algorithm

Data: Tensor $T \in \mathbb{R}^{n\times n\times n}$
Result: Given some $n \times n \times n$ tensor T such that $T = \sum_i^r u_i \otimes v_i \otimes w_i$, find the $3r$ vectors, assuming $v_1, v_2, ..., v_r$ are linearly independent, $w_1, w_2, ..., w_r$ are linearly independent, and $u_1, u_2, ..., u_r$ are distinct.

Alg(T):
pick random $g \in \mathbb{R}^n$
define $M_g = \sum \langle u_i, g\rangle v_i \otimes w_i$
pick random $h \in \mathbb{R}^n$
define $M_h = \sum \langle u_i, h\rangle v_i \otimes w_i$
find $M_g M_h^{-1}$
columns of V = eigenvectors of $M_g M_h^{-1}$
rows of W = eigenvectors of $(M_g)^T (M_h^T)^{-1}$
solve linear system for U
return U, V, W

Proof:
We define D_h as the diagonal matrix of $\langle u_i, h\rangle$, and D_g as the diagonal matrix of $\langle u_i, g\rangle$. V's columns are the vectors $v_1, v_2, ..., v_r$. W's rows are the vectors $w_1, w_2, ..., w_n$. Thus, we get the following:

$$\begin{aligned} M_g &= V D_g W \\ M_h &= V D_h W \\ \therefore M_g M_h^{-1} &= V(D_g D_h^{-1})V^{-1} \end{aligned}$$

Recall that if a matrix $M = PDP^{-1}$ for some invertible P and diagonal D, then the eigenvectors of M are the columns of P. Since $(D_g D_h^{-1})$ is diagonal, we can use eigendecomposition to recover V.
We also get:

$$\begin{aligned} (M_g)^T (M_h^T)^{-1} &= (M_g)^T (M_h^{-1})^T \\ &= (M_h^{-1} M_g)^T \\ &= (W^{-1} D_h^{-1} V^{-1} V D_g W)^T \\ &= (W^{-1}(D_h^{-1} D_g)W)^T \\ &= W^T (D_h^{-1} D_g)^T (W^{-1})^T \end{aligned}$$

Similarly, since $(D_h^{-1} D_g)^T$ is diagonal, we can recover the rows of W by eigendecomposing to get the eigenvectors of $(M_g)^T (M_h^T)^{-1}$.
Note this algorithm is restricted by the condition $T = \sum_i^r u_i \otimes v_i \otimes w_i$ and can only decompose tensors of rank $\leq n$, as that is the maximum number of vectors we can recover.

8.3 Application: Independent Component Analysis

Suppose that we have samples $y = Ax + b$, where $x \in \mathbb{R}^n$ is a random vector with independent coordinates. For simplicity, assume each component of $x \in \{\pm 1\}$. A is some unknown invertible linear transformation, and b is some unknown shift. Our goal is to recover some permutation of A and b, which implies that you can recover X from y.

8.3.1 Cocktail Party Problem

Imagine you are at listening to a conversation at a cocktail party where there are independent voices all speaking together. The goal is to recover the individual voices from the mixed signal.

Given samples $y_1, \ldots, y_n$, one natural step is centering.

$$\hat{y_i} = y_i - \mathbb{E}[y_i]$$

This is the same as as subtracting b since $\mathbb{E}[Ax] = 0$ from the assumption of $x \in \{\pm 1\}^n$, so $\mathbb{E}[y] = \mathbb{E}[Ax] + \mathbb{E}[b] = b$. After centering, the problem becomes $y = Ax$.

Another useful technique here is *whitening*, putting the vectors in isotropic positions. After whitening, the covariance of the data becomes the identity matrix.

$$\begin{aligned}
\mathrm{Cov}[y] &= \mathbb{E}[(y - \mathbb{E}[y])(y - \mathbb{E}[y])^\intercal] \\
&= \mathbb{E}[yy^\intercal] = \mathbb{E}[Axx^\intercal A^\intercal] \\
&= A\mathbb{E}[xx^\intercal]A^\intercal \\
&= AA^\intercal
\end{aligned}$$

The last step follows by showing that $\mathbb{E}[xx^\intercal]$ is the identity matrix I. $\mathbb{E}[xx^\intercal]$ is the matrix with entries $\mathbb{E}[x_i x_j]$ for row i, and column j. Recall that x_i and x_j are random ± 1.

$$\mathbb{E}[x_i x_j] = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

From $AA^\intercal$ it is impossible to recover A because we could apply any rotation matrix U in the middle. Recall that a rotation matrix U is any matrix such that $U^\intercal U = I$.

$$AA^\intercal = (AU)(AU)^\intercal$$

This is an issue with low-rank decompositions in general. If a matrix is written as a low-rank decomposition, you could always insert a rotation matrix in the middle. It turns out, that adding in higher-order information, the rotation issue goes away.

Since we are getting samples from y, we estimate higher order tensors of y. For example, we could estimate the random tensor $\mathbb{E}[y \otimes y \otimes y \otimes y]$.

$$M_4 = \mathbb{E}[y \otimes y \otimes y \otimes y] - T$$

$$[T]_{a,b,c,d} = \mathbb{E}[y_a y_b]\mathbb{E}[y_c y_d] + \mathbb{E}[y_b y_c]\mathbb{E}[y_a y_d] + \mathbb{E}[y_a y_c]\mathbb{E}[y_b y_d]$$

$$M_4 = \sum_{i=1}^{n} \kappa_i * (A_i \otimes A_i \otimes A_i \otimes A_i)$$

where $k_i = (\mathbb{E}[x_i^2])^2 - 3$ and A_i are the columns of A. We assume that $\kappa_i \neq 0$. A_i can be recovered by Jennrich's algorithm.

Note that the the algorithm fails if $\mathbb{E}[x_i^4] = 3$, since we break the assumption that $\kappa_i \neq 0$. Assume that x_i are random Gaussian vectors. Then we know that $\mathbb{E}[x_i^4] = 3$. The Gaussian distribution is invariant under rotation. If x is is rotationally symmetric, then it is impossible to disambiguate the rotations. In fact, if $\mathbb{E}[x_i] = 0$, $\mathbb{E}[x_i^2] = 1$, and $\mathbb{E}[x_i^4] = 3$, then x_i is Gaussian.