Coordinate-descent for learning orthogonal matrices through Givens rotations

Uri Shalit URI.SHALIT@MAIL.HUJI.AC.IL
ICNC-ELSC & Computer Science Department, The Hebrew University of Jerusalem, 91904 Jerusalem Israel
The Gonda Brain Research Center, Bar Ilan University, 52900 Ramat-Gan, Israel

Gal Chechik GAL.CHECHIK@BIU.AC.IL
The Gonda Brain Research Center, Bar Ilan University, 52900 Ramat-Gan, Israel

Abstract

Optimizing over the set of orthogonal matrices is a central component in problems like sparse-PCA or tensor decomposition. Unfortunately, such optimization is hard since simple operations on orthogonal matrices easily break orthogonality, and correcting orthogonality usually costs a large amount of computation.

Here we propose a framework for optimizing orthogonal matrices, that is the parallel of coordinate-descent in Euclidean spaces. It is based on *Givens-rotations*, a fast-to-compute operation that affects a small number of entries in the learned matrix, and preserves orthogonality.

We show two applications of this approach: an algorithm for tensor decompositions used in learning mixture models, and an algorithm for sparse-PCA. We study the parameter regime where a Givens rotation approach converges faster and achieves a superior model on a genome-wide brain-wide mRNA expression dataset.

1. Introduction

Optimization over orthogonal matrices – matrices whose rows and columns form an orthonormal basis of $\mathbb{R}^d$ – is central to many machine learning optimization problems. Prominent examples include *Principal Component Analysis* (PCA), *Sparse PCA*, and *Independent Component Analysis (ICA)*. In addition, many new applications of tensor orthogonal decompositions were introduced recently, including Gaussian Mixture Models, Multi-view Models and Latent Dirichlet Allocation (e.g., Anandkumar et al. (2012a); Hsu & Kakade (2013)).

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

A major challenge when optimizing over the set of orthogonal matrices is that simple updates such as matrix addition usually break orthonormality. Correcting by orthonormalizing a matrix $V \in \mathbb{R}^{d\times d}$ is typically a costly procedure: even a change to a single element of the matrix, may require $O(d^3)$ operations in the general case for re-orthogonalization.

In this paper, we present a new approach for optimization over the manifold of orthogonal matrices, that is based on a series of sparse and efficient-to-compute updates that operate **within the set of orthonormal matrices**, thus saving the need for costly orthonormalization. The approach can be seen as the equivalent of coordinate descent in the manifold of orthonormal matrices. Coordinate descent methods are particularly relevant for problems that are too big to fit in memory, for problems where one might be satisfied with a partial answer, or in problems where not all the data is available at one time (Richtárik & Takáč, 2012).

We start by showing that the orthogonal-matrix equivalent of a single coordinate update is applying a single *Givens rotation* to the matrix. In section 3 we prove that for a differentiable objective the procedure converges to a local optimum under minimal conditions, and prove an $O(1/T)$ convergence rate for the norm of the gradient. Sections 4 and 5 describe two applications: (1) sparse PCA, including a variant for streaming data; (2) a new method for orthogonal tensor decomposition. We study how the performance of the method depends on the problems hyperparameters using synthetic data, and demonstrate that it achieves superior accuracy on an application of sparse-PCA for analyzing gene expression data.

2. Coordinate descent on the orthogonal matrix manifold

Coordinate descent (CD) is an efficient alternative to gradient descent when the cost of computing and applying a gradient step at a single coordinate is small relative to com-

puting the full gradient. In these cases, convergence can be achieved with a smaller number of computing operations, although using a larger number of (faster) steps.

Applying coordinate descent to optimize a function involves choosing a coordinate basis, usually the standard basis. Then calculating a directional derivative in the direction of one of the coordinates. And finally, updating the iterate in the direction of the chosen coordinate. To generalize CD to operate over the set of orthogonal matrices, we need to generalize these ideas of directional derivatives and updating the orthogonal matrix in a "straight direction".

In the remaining of this section, we introduce the set of orthogonal matrices, $\mathcal{O}_d$, as a Riemannian manifold. We then show that applying coordinate descent to the Riemannian gradient amounts to multiplying by Givens rotations. Throughout this section and the next, the objective function is assumed to be a differentiable function $f : \mathcal{O}_d \to \mathbb{R}$.

2.1. The orthogonal manifold and Riemannian gradient

The orthogonal matrix manifold $\mathcal{O}_d$ is the set of $d \times d$ matrices U such that $UU^T = U^TU = I_d$. It is a $\frac{d(d+1)}{2}$ dimensional smooth manifold, and is an embedded submanifold of the Euclidean space $R^{d\times d}$ (Absil et al., 2009).

Each point $U \in \mathcal{O}_d$ has a tangent space associated with it, a $\frac{d(d-1)}{2}$ dimensional vector space, that we will use below in order to capture the notion of 'direction' on the manifold. The tangent space is denoted $T_U\mathcal{O}_d$, and defined by $T_U\mathcal{O}_d = \{Z \in \mathbb{R}^{d\times d}, Z = U\Omega : \Omega = -\Omega^T\} = USkew(d)$, where $Skew(d)$ is the set of skew-symmetric $d \times d$ matrices.

2.1.1. Geodesic directions

The natural generalization of straight lines to the manifold context are *geodesic curves*. A geodesic curve is locally the shortest curve between two points on the manifold, or equivalently, a curve with no acceleration tangent to the manifold (Absil et al., 2009). For a point $U \in \mathcal{O}_d$ and a "direction" $U\Omega \in T_U\mathcal{O}_d$ there exists a single geodesic line that passes through U in direction Ω. Fortunately, while computing a geodesic curve in the general case might be hard, computing it for $\mathcal{O}_d$ has a closed form expression: $\gamma : (-1, 1) \to \mathcal{O}_d$, $\gamma(\theta) = U\text{Expm}(\theta\Omega)$, where $\gamma(\theta)$ with $\theta \in (-1, 1)$ is the parameterization of the curve, and Expm is the matrix exponential function.

In the special case where the operator $Expm(\Omega)$ is applied to a skew-symmetric matrix Ω, it maps Ω into an orthogonal matrix [1]. As a result, $\gamma(\theta) = U\text{Expm}(\theta\Omega)$ is also an orthogonal matrix for all θ.

[1] Because $\text{Expm}(\Omega)\text{Expm}(\Omega)^T = \text{Expm}(\Omega)\text{Expm}(\Omega^T) = \text{Expm}(\Omega)\text{Expm}(-\Omega) = I$

2.1.2. The directional derivative

In analogy to the Euclidean case, the Riemannian directional derivative of f in the direction of a vector $U\Omega \in T_U\mathcal{O}_d$ is defined as the derivative of a single variable function which involves looking at f along a single curve (Absil et al., 2009):

$$\nabla_\Omega f(U) \equiv \frac{\mathrm{d}}{\mathrm{d}\theta} f(\gamma(\theta))\Big|_{\theta=0} = \frac{\mathrm{d}}{\mathrm{d}\theta} f(U\text{Expm}(\theta\Omega))\Big|_{\theta=0}. \tag{1}$$

Note that $\nabla_\Omega f(U)$ is a scalar. The definition means that the directional derivative is f' with f restricted to the geodesic curve going through U in the direction $U\Omega$.

2.1.3. The directional update

Since the Riemannian equivalent of walking in a straight line is walking along the geodesic curve, taking a step of size $\eta > 0$ from a point $U \in \mathcal{O}_d$ in direction $U\Omega \in T_U\mathcal{O}_d$ amounts to:

$$U_{next} = U\text{Expm}\left(\eta\Omega\right), \tag{2}$$

We also have to define the orthogonal basis for $Skew(d)$. Here we use $\{e_ie_j^T - e_je_i^T : 1 \le i < j \le d\}$. We denote each basis vector as $H_{ij} = e_ie_j^T - e_je_i^T, 1 \le i < j \le d$.

2.2. Givens rotations as coordinate descent

Coordinate descent is a popular method of optimization in Euclidean spaces. It can be more efficient than computing full gradient steps when it is possible to (1) compute efficiently the coordinate directional derivative, and (2) apply the update efficiently. We will now show that in the case of the orthogonal manifold, applying the update (step 2) can be achieved efficiently. The cost of computing the coordinate derivative (step 1) depends on the specific nature of the objective function f, and we we show below several cases where that can be achieved efficiently.

Let H_{ij} be a coordinate direction, let $\nabla_{H_{ij}} f(U)$ be the corresponding directional derivative, and choose step size $\eta > 0$. A straightforward calculation based on Eq. 2 shows that the update $U_{next} = U\text{Expm}(-\eta H_{ij})$ obeys

$$\text{Expm}(-\eta H_{ij}) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & cos(\eta) & \cdots & -sin(\eta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & sin(\eta) & \cdots & cos(\eta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

This matrix is known as a *Givens rotation* (Golub & Van Loan, 2012) and is denoted $G(i,j,-\eta)$. It has $cos(\eta)$ at the (i,i) and (j,j) entries, and $\pm sin(\eta)$ at the (j,i) and (i,j) entries. It is a simple and sparse orthogonal matrix. For a dense matrix $A \in \mathbb{R}^{d\times d}$, the linear operation $A \mapsto AG(i,j,\eta)$ rotates the i^{th} and j^{th} columns of A by an angle η in the plane they span. Computing this operation costs $6d$ multiplications and additions. As a result, computing Givens rotations successively for all $\frac{d(d-1)}{2}$ coordinates H_{ij} takes $O(d^3)$ operations, the same order as ordinary matrix multiplication. Therefore the relation between the cost of a single Givens relative to a full gradient update is the same as the relation between the cost of a single coordinate update and a full update is in Euclidean space. We note that any determinant-1 orthogonal matrix can be decomposed into at most $\frac{d(d-1)}{2}$ Givens rotations.

2.3. The Givens rotation coordinate descent algorithm

Based on the definition of Givens rotation, a natural algorithm for optimizing over orthogonal matrices is to perform a sequence of rotations, where each rotation is equivalent to a coordinate-step in CD.

To fully specify the algorithm we need two more ingredients: (1) Selecting a schedule for going over the coordinates and (2) Selecting a step size. For scheduling, we chose here to use a random order of coordinates, following many recent coordinate descent papers (Richtárik & Takáč, 2012; Nesterov, 2012; Patrascu & Necoara, 2013).

For choosing the step size η we use exact minimization, since we found that for the problems we aim to solve, using exact minimization was usually the same order of complexity as performing approximate minimization (like using an Armijo step rule Bertsekas (1999); Absil et al. (2009)).

Based on these two decisions, Algorithm (1) is a random coordinate minimization technique.

Algorithm 1 Riemannian coordinate minimization on $\mathcal{O}_d$

Input: Differentiable objective function f, initial matrix $U_0 \in \mathcal{O}_d$
$t = 0$
while not converged **do**
1. Sample uniformly at random a pair $(i(t), j(t))$ such that $1 \le i(t) < j(t) \le d$.
2. $\theta_{t+1} = \underset{\theta}{\operatorname{argmin}}\ f\left(U_t \cdot G(i,j,\theta)\right)$.
3. $U_{t+1} = U_t \cdot G(i,j,\theta_{t+1})$.
4. $t = t+1$.
end while
Output: U_{final}.

3. Convergence rate for Givens coordinate minimization

In this section, we show that under the assumption that the objective function f is differentiable Algorithm 1 converges to critical point of the function f, and the only stable convergence points are local minima. We further show that the expectation w.r.t. the random choice of coordinates of the squared l_2-norm of the Riemannian gradient converges to 0 with a rate of $O(\frac{1}{T})$ where T is the number of iterations. The proofs, including some auxiliary lemmas, are provided in the supplemental material. Overall we provide the same convergence guarantees as provided in standard non-convex optimization (e.g., Nemirovski (1999); Bertsekas (1999)).

Definition 1. *Riemannian gradient*
The Riemannian gradient $\nabla f(U)$ of f at point $U \in \mathcal{O}_d$ is the matrix $U\Omega$, where $\Omega \in Skew(d)$, $\Omega_{ji} = -\Omega_{ij} = \nabla_{ij} f(U), 1 \le i < j \le d$ is defined to be the directional derivative as given in Eq. 1, and $\Omega_{ii} = 0$. The norm of the Riemannian gradient $||\nabla f(U)||^2 = Tr(\nabla f(U) \nabla f(U)^T) = ||\Omega||^2_{fro}$.

Definition 2. *A point $U_* \in \mathcal{O}_d$ is* asymptotically stable *with respect to Algorithm (1) if it has a neighborhood $\mathcal{V}$ of U_* such that all sequences generated by Algorithm (1) with starting point $U_0 \in \mathcal{V}$ converge to U_*.*

Theorem 1. *Convergence to local optimum*
(a) The sequence of iterates U_t of Algorithm (1) satisfies: $\lim_{t\to\infty} ||\nabla f(U_t)|| = 0$. This means that the accumulation points of the sequence $\{U_t\}_{t=1}^{\infty}$ are critical points of f.
(b) Assume the critical points of f are isolated. Let U_* be a critical point of f. Then U_* is a local minimum of f if and only if it is asymptotically stable with regard to the sequence generated by Algorithm (1).

Definition 3. *For an iteration t of Algorithm (1), and a set of indices $(i(t), j(t))$, we define the auxiliary single variable function g_t^{ij} :*

$$g_t^{ij}(\theta) = f\left(U_t \cdot G(i,j,\theta)\right), \tag{3}$$

Note that g_t^{ij} are differentiable and periodic with a period of 2π. Since $\mathcal{O}_d$ is compact and f is differentiable there exists a single Lipschitz constant $L(f) > 0$ for all g_t^{ij}.

Theorem 2. *Rate of convergence*
Let f be a continuous function with L-Lipschitz directional derivatives [2]. Let U_t be the sequence generated by Algorithm 1. For the sequence of Riemannian gradients $\nabla f(U_t) \in T_{U_t}\mathcal{O}_d$ we have:

$$\max_{0\le t\le T} E\left[||\nabla f(U_t)||_2^2\right] \le \frac{L \cdot d^2 \left(f(U_0) - f_{min}\right)}{T+1} \quad . \tag{4}$$

[2] Because $\mathcal{O}_d$ is compact, any function f with a continuous second-derivative will obey this condition.

The proof is a Riemannian version of the proof for the rate of convergence of Euclidean random coordinate descent for non-convex functions (Patrascu & Necoara, 2013) and is provided as supplemental material.

4. Sparse PCA

Principal component analysis (PCA) is a basic dimensionality reducing technique used throughout the sciences. Given a data set $A \in \mathbb{R}^{d\times n}$ of n observations in d dimensions, the principal components are a set of orthogonal vectors $z_1, z_2, \ldots, z_m \in \mathbb{R}^d$, such that the variance $\sum_{i=1}^m z_i^T AA^T z_i$ is maximized. The data is then represented in a new coordinate system $\hat{A} = Z^T A$ where $Z = [z_1, z_2, \ldots, z_m] \in \mathbb{R}^{d\times m}$.

One drawback of ordinary PCA is lack of interpretability. In the original data A, each dimension usually has an understandable meaning, such as the level of expression of a certain gene. The dimensions of $\hat{A}$ however are typically linear combinations of all gene expression levels, and as such are much more difficult to interpret. A common approach to the problem of finding *interpretable* principal components is Sparse PCA (Zou et al., 2006; Journée et al., 2010; d'Aspremont et al., 2007; Zhang et al., 2012; Zhang & Ghaoui, 2012). SPCA aims to find vectors z_i as in PCA, but which are also sparse. In the gene-expression example, the non-zero components of z_i might correspond to a few genes that explain well the structure of the data A.

One of the most popular approaches for solving the problem of finding sparse principal components is the work by Journée et al. (2010). In their paper, they formalize the problem as finding the optimum of the following constrained optimization problem to find the sparse basis vectors Z:

$$\underset{U\in\mathbb{R}^{n\times m}, Z\in\mathbb{R}^{d\times m}}{\operatorname{argmax}} \; Tr(Z^T AU) - \gamma \sum_{ij} |Z_{ij}| \qquad (5)$$

$$s.t.\;\; U^T U = I_m, \sum_{i=1}^d Z_{ij}^2 = 1 \;\forall j = 1 \ldots m \quad .$$

Journée et al. provide an algorithm to solve Eq. 5 that has two parts: The first and more time consuming part finds an optimal U, from which optimal Z is then found. We focus here on the problem of finding the matrix U. Note that when $m = n$, the constraint $U^T U = I_m$ implies that U is an orthogonal matrix.

We use a second formulation of the optimization problem, also given by Journée et al. in section 2.5.1 of their paper:

$$\underset{U\in\mathbb{R}^{n\times m}}{\operatorname{argmax}} \sum_{j=1}^m \sum_{i=1}^d [|(A\cdot U)_{ij}| - \gamma]_+^2$$
$$s.t.\;\; U^T U = I_m,$$

where n is the number of samples, d is the input dimensionality and m is the number of PCA components computed. This objective is once-differentiable and the objective matrix U grows with the number of samples n.

4.1. Givens rotation algorithm for the full case $m = n$

If we choose the number of principal components m to be equal to the number of samples n we can apply Algorithm ((1)) directly to solve the optimization problem of Eq. 6. Explicitly, at each round t, for choice of coordinates (i, j) and a matrix $U_t \in \mathcal{O}_d$, the resulting coordinate minimization problem is:

$$\begin{aligned}\underset{\theta}{\operatorname{argmin}} - \sum_{j=1}^m \sum_{i=1}^d [|(AU_t G(i,j,\theta))_{ij}| - \gamma]_+^2 = \\ \underset{\theta}{\operatorname{argmin}} - \sum_{k=1}^d [|cos(\theta)(AU_t)_{ki} + sin(\theta)(AU_t)_{kj}| - \gamma]_+^2 + \\ [| - sin(\theta)(AU_t)_{ki} + cos(\theta)(AU_t)_{kj}| - \gamma]_+^2 \end{aligned} \qquad (6)$$

Algorithm 2 Riemannian coordinate minimization for sparse PCA

Input: Data matrix $A \in \mathbb{R}^{d\times n}$, initial matrix $U_0 \in \mathcal{O}_n$, sparsity parameter $\gamma \geq 0$
$t = 0$
$AU = A \cdot U_0$.
while not converged **do**
1. Sample uniformly at random a pair $(i(t), j(t))$ such that $1 \leq i(t) < j(t) \leq n$.
2. $\theta_{t+1} = \underset{\theta}{\operatorname{argmax}}$ $\sum_{k=1}^d ([|cos(\theta)(AU)_{ki(t)} + sin(\theta)(AU)_{kj(t)}| - \gamma]_+^2$ $+[| - sin(\theta)(AU)_{ki(t)} + cos(\theta)(AU)_{kj(t)}| - \gamma]_+^2)$.
3. $AU = AU \cdot G(i(t), j(t)), \theta_{t+1})$.
4. $t = t + 1$.
end while
5. $Z = solveForZ(AU, \gamma)$ // Algorithm 6 of Journée et al. (2010).
Output: $Z \in \mathbb{R}^{d\times n}$

See Algorithm (2) for the full procedure. In practice, there is no need to store the matrices U_t in memory, and one can work directly with the matrix AU_t. Evaluating the expression in Eq. 6 for a given θ requires $O(d)$ operations, where d is the dimension of the data. We found in practice that optimizing Eq. 6 required an order of 5-10 evaluations. Overall each iteration of Algorithm (2) requires $O(d)$ operations.

4.2. Givens rotation algorithm for the case $m < n$

The major drawback of Algorithm (2) is that it requires the number of principal components m to be equal to the num-

ber of samples n. This kind of "full dimensional sparse PCA" may not be necessary when researchers are interested to obtain a small number of components. We therefore develop a streaming version of Algorithm (2). For a small given m, we treat the data as if only m samples exist at any time, giving an intermediate model $AU \in \mathbb{R}^{d\times m}$. After a few rounds of optimizing over this subset of samples, we use a heuristic to drop one of the previous samples and incorporate a new sample. This gives us a streaming version of the algorithm because in every phase we need only m samples of the data in memory. The full details of the algorithm are given in the supplemental material.

4.3. Experiments

Sparse PCA attempts to trade-off two variables: the fraction of data variance that is explained by the model's components, and the level of sparsity of the components. In our experiment, we monitor a third important parameter, the number of floating point operations (FLOPS) performed to achieve a certain solution. To compute the number of FLOPS we counted the number of additions and multiplications computed on each iteration. This does not include pointer arithmetic.

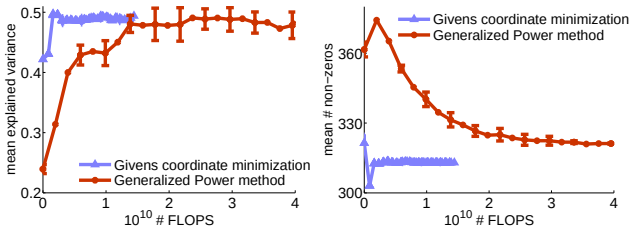
We first examined Algorithm 2 for the case where $m = n$. We used the prostate cancer gene expression data by Singh et al. (2002). This dataset consists of the gene expression levels for 52 tumor and 50 normal samples over 12,600 genes, resulting in a $12,600 \times 102$ data matrix.

We compared the performance of our approach with that of the *Generalized Power Method* of Journée et al. (2010). We focus on this method for comparisons because both methods optimize the same objective function, which allows to characterize the relative strengths and weaknesses of the two approaches.

As can be seen in Figure 1, the Givens coordinate minimization method finds a sparser solution with better explained variance, and does so faster than the generalized power method.

We tested the streaming version of the coordinate descent algorithm for sparse PCA (Algorithm 5, supp. material) on a recent large gene expression data set collected from of six human brains (Hawrylycz et al., 2012). Overall, each of the 20K human genes was measured at 3702 different brain locations, and this data can be used to study the spatial patterns of mRNA expression across the human brain. We again compared the performance of our approach with that of the *Generalized Power Method* of Journée et al. (2010).

We split the data into 5 train/test partitions, with each train set including 2962 examples and each test set including 740 examples. We evaluated the amount of variance explained by the model on the test set. We use the adjusted variance procedure suggested in this case by Zou et al. (2006), which takes into account the fact that the sparse principal components are not orthogonal.



(a) explained variance (b) number of non-zeros

Figure 1. (a) The explained variance as function of FLOPS of the coordinate minimization method from Algorithm 2 and of the generalized power method by Journée et al. (2010), on a prostate cancer gene expression dataset. (b) The number of non-zeros in the sparse PCA matrix as function of FLOPS of the coordinate minimization method from Algorithm 2 and of the generalized power method by Journée et al. (2010), on a prostate cancer gene expression dataset. The size of the sparse PCA matrix is $12,600 \times 102$.

For the Generalized Power Method we use the greedy l_1 version of Journée et al. (2010), with the parameter μ set to 1. We found the greedy version to be more stable and to be able to produce sparse solutions when the number of components was $m > 1$. We used values of γ ranging from 0.01 to 0.2, and two stopping conditions: "convergence", where the algorithm was run until its objective converged within a relative tolerance level of 10^{-4}, and "early stop" where we stopped the algorithm after 14% of the iterations required for convergence. For our algorithm we used the same range of γ values, and an early-stop condition where the algorithm was stopped after using 14% of the samples.

Figure 2 demonstrates the tradeoff between floating point operations and explained variance for SPCA with 3, 5 and 10 components and with 3 sparsity levels: 5%, 10% and 20%. Using low dimensions is often useful for visual exploration of the data. Each dot represents one instance of the algorithm, run with a certain value of γ and stopping criterion. To avoid clutter we only show instances which performed best in terms of explained variance or few FLOPS.

When strong sparsity is required (5% or 10% sparsity), the Givens-rotation coordinate descent algorithm finds solutions faster (blue rectangles are more to the left in Figure 2), and these solutions are similar or better in terms of explained variance. For low-dimensional less sparse solutions (20% sparsity) we find that the generalized power method finds comparable or better solutions using the same computational cost, but only when the number of components is small, as seen in Figure 2.c,f,i.

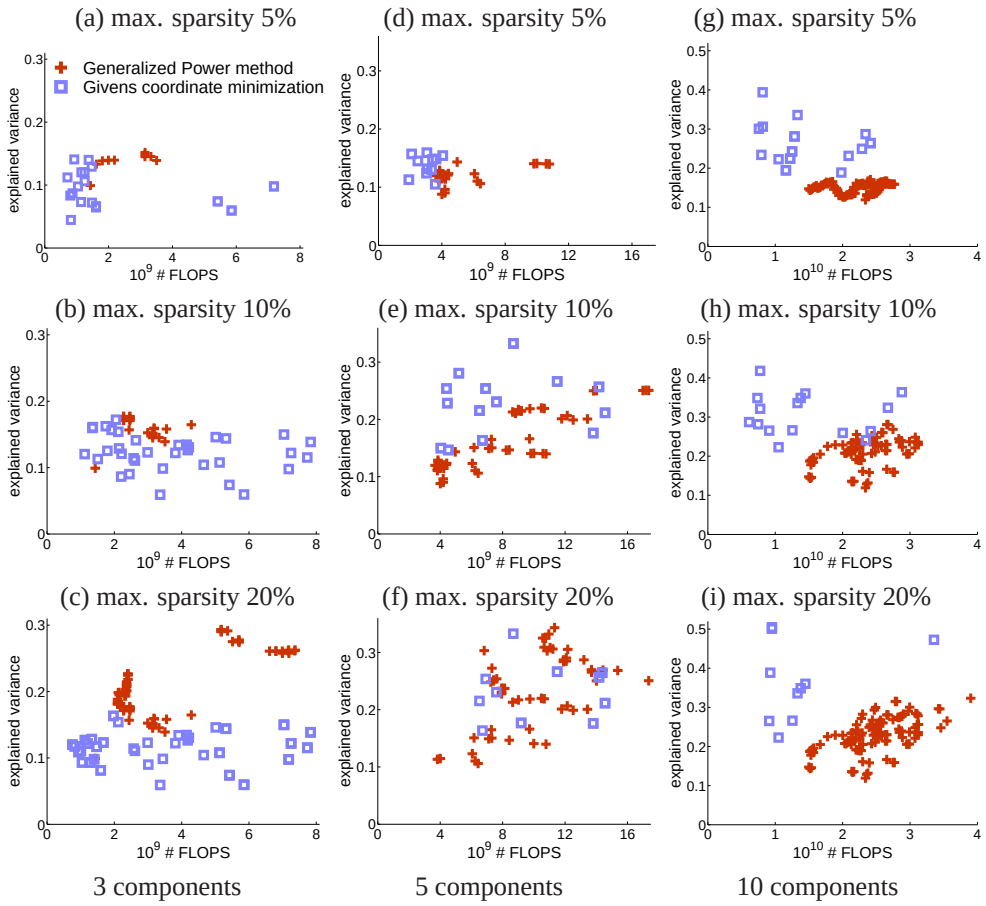


Figure 2. The tradeoff between explained variance and computational cost for 3, 5 and 10-component sparse PCA models applied to human gene expression data. The models are constrained for maximum sparsity of 5% (a), (d) & (g), 10% (b), (e) & (h) and 20% (c), (f) & (i). Red pluses indicate the Generalized Power method (Journée et al., 2010); blue squares represent the Givens coordinate procedure. See Subsection 4.3 for experimental conditions. Explained variance was adjusted following Zou et al. (2006).

5. Orthogonal tensor decomposition

Recently it has been shown that many classic machine learning problem such as Gaussian Mixture Models and Latent Dirichlet Allocation can be solved efficiently by using 3rd order moments (Anandkumar et al., 2012a; Hsu & Kakade, 2013; Anandkumar et al., 2012b;c; Chaganty & Liang, 2013). These methods ultimately rely on finding an orthogonal decomposition of 3-way tensors $T \in \mathbb{R}^{d\times d\times d}$, and reconstructing the solution from this decomposition. Here we show that the problem of finding an orthogonal decomposition for a tensor $T \in \mathbb{R}^{d\times d\times d}$ can be naturally cast as an optimization problem over the orthogonal matrix manifold. We apply Algorithm 1 to this problem, and compare its performance on a task of finding a Gaussian Mixture Model with a state-of-the-art tensor decomposition method, the robust Tensor Power Method (Anandkumar et al., 2012a). We find that the Givens coordinate method consistently finds better solutions when the number of mixture components is large.

5.1. Orthogonal tensor decomposition

The problem of tensor decomposition is very hard in general (Kolda & Bader, 2009). However, a certain class of tensors known as *orthogonally decomposable* tensors are easier to decompose, as has been discussed recently by Anandkumar et al. (2012a); Hsu & Kakade (2013) and others. Here we introduce the problem of orthogonal tensor decomposition, and provide a new characterization of the solutions to the decomposition problem as extrema of an optimization problem on the orthogonal matrix manifold.

The resulting algorithm is similar to one recently proposed by Ishteva et al. (2013). However, we aim for full diagonalization, while they focus on finding a good low-rank approximation. This results in different objective functions: ours involves third-order polynomials on $\mathcal{O}_d$, while Ishteva et al.'s results in sixth-order polynomials on the low-rank compact Stiefel manifold. Diagonalizing the tensor T is attainable in our case thanks to the strong assumption that

it is orthogonally decomposable. Nonetheless, both methods are extensions of Jacobi's eigenvalue algorithm to the tensor case, in different setups.

We start with preliminary notations and definitions. We focus here on symmetric tensors $T \in \mathbb{R}^{d\times d\times d}$. A third-order tensor is symmetric if its values are identical for any permutation σ of the indices: with $T_{i_1 i_2 i_3} = T_{i_{\sigma(1)} i_{\sigma(2)} i_{\sigma(3)}}$.

We also view a tensor T as a trilinear map. $T : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$: $T(v_1, v_2, v_3) = \sum_{a,b,c=1}^{d} T_{abc} v_{1a} v_{2b} v_{3c}$.

Finally, we also use the three-form tensor product of a vector $u \in \mathbb{R}^d$ with itself: $u \otimes u \otimes u \in \mathbb{R}^{d\times d\times d}$, $(u \otimes u \otimes u)_{abc} = u_a \cdot u_b \cdot u_c$. Such a tensor is called a *rank-one* tensor.

Definition 4. *A symmetric tensor T is orthogonally decomposable if there exists an orthonormal set $v_1, \dots v_d \in \mathbb{R}^d$, and positive scalars $\lambda_1, \dots \lambda_d > 0$ such that:*

$$T = \sum_{i=1}^{d} \lambda_i (v_i \otimes v_i \otimes v_i). \tag{7}$$

Unlike matrices, most symmetric tensors are not orthogonally decomposable. However, as shown by Anandkumar et al. (2012a); Hsu & Kakade (2013); Anandkumar et al. (2013), several problems of interest, notably Gaussian Mixture Models and Latent Dirichlet Allocation do give rise to third-order moments which are orthogonally decomposable in the limit of infinite data.

The goal of orthogonal tensor decomposition is, given an orthogonally decomposable tensor T, to find the orthogonal vector set $v_1, \dots v_d \in \mathbb{R}^d$ and the scalars $\lambda_1, \dots \lambda_d > 0$.

We now show that finding an orthogonal decomposition can be stated as an optimization problem over $\mathcal{O}_d$:

Theorem 3. *Let $T \in R^{d\times d\times d}$ have an orthogonal decomposition as in Definition 4, and consider the optimization problem*

$$\max_{U\in\mathcal{O}_d} f(U) = \sum_{i=1}^{d} T(u_i, u_i, u_i), \tag{8}$$

where $U = [u_1\ u_2\ \dots\ u_d]$. The stable stationary points of the problem are exactly orthogonal matrices U such that $u_i = v_{\pi(i)}$ for a permutation π on $[d]$. The maximum value they attain is $\sum_{i=1}^{d} \lambda_i$.

The proof is given in the supplemental material.

5.2. Coordinate minimization algorithm for orthogonal tensor decomposition

We now adapt Algorithm 1 for solving the problem of orthogonal tensor decomposition of a tensor T, by maximizing the objective function 8, $f(U) = \sum_{i=1}^{d} T(u_i, u_i, u_i)$.

For this we need to calculate the form of the function $g_t^{ij}(\theta) = f\left(U \cdot G(i,j,\theta)\right)$. We have:

$$g_t^{ij}(\theta) = f\left(U \cdot G(i,j,\theta)\right) = \sum_{k\neq i,j}^{d} T(u_k, u_k, u_k) + T\left(\tilde{u}_i,\ \tilde{u}_i,\ \tilde{u}_i\right) + T\left(\tilde{u}_j,\ \tilde{u}_j,\ \tilde{u}_j\right).$$

where we used $\tilde{u}_i = cos(\theta)u_i + sin(\theta)u_j$ and $\tilde{u}_j = cos(\theta)u_j - sin(\theta)u_i$.

Denote by $\tilde{T}$ the tensor such that: $\tilde{T}_{ijk} = T(u_i, u_j, u_k)$. We will abuse notation and denote $\tilde{T} = T(U, U, U)$. The tensor $\tilde{T}$ is the three-way multiplication of T by the matrix U. This is the lifting of the matrix operation $\tilde{M} = M(U,U) = UMU^T$ to the tensor domain.

Collecting terms, using the symmetry of T and some basic trigonometric identities, we then have:

$$\begin{aligned} g_t^{ij}(\theta) =& cos^3(\theta)\left(\tilde{T}_{iii} + \tilde{T}_{jjj} - 3\tilde{T}_{ijj} - 3\tilde{T}_{jii}\right) \\ &+ sin^3(\theta)\left(\tilde{T}_{iii} - \tilde{T}_{jjj} - 3\tilde{T}_{ijj} + 3\tilde{T}_{jii}\right) \\ &+ cos(\theta)\left(3\tilde{T}_{ijj} + 3\tilde{T}_{jii}\right) \\ &+ sin(\theta)\left(3\tilde{T}_{ijj} - 3\tilde{T}_{jii}\right). \end{aligned} \tag{9}$$

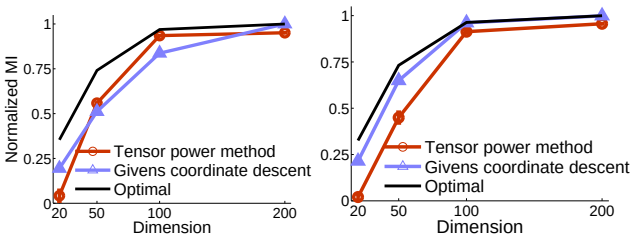
In each step of the algorithm, we maximize $g_t^{ij}(\theta)$ over $-\pi \leq \theta < \pi$. The scalar function g_t^{ij} has at most 3 maxima that can be obtained in closed form solution, and thus can be maximized in constant time.

Algorithm 3 Riemannian coordinate maximization for orthogonal tensor decomposition

Input: Symmetric tensor $T \in \mathbb{R}^{d\times d\times d}$.
Initialize $t = 0$, $\tilde{T}^0 = T$, $U_0 = I_d$.
while not converged **do**
1. Sample uniformly at random a pair $(i(t), j(t))$ such that $1 \leq i(t) < j(t) \leq d$.
2. Obtain $\tilde{T}^t_{iii}$, $\tilde{T}^t_{jjj}$, $\tilde{T}^t_{ijj}$, $\tilde{T}^t_{jii}$.
3. $\theta_t = \operatorname{argmax}_{\theta} g_t^{ij}(\theta)$, where g_t^{ij} is defined as in 9.
4. $\tilde{T}^{t+1} = \tilde{T}^t\left(G(i,j,\theta_t), G(i,j,\theta_t), G(i,j,\theta_t)\right)$.
// Three way multiplication of $\tilde{T}^t$ by $G(i,j,\theta_t)$.
5. $U_{t+1} = U_t G(i,j,\theta_t)$.
6. $t = t + 1$.

end while
Output: U_{final}.

The most computationally intensive part of Algorithm 3 is line 4. Multiplying a tensor by the Givens rotation $G(i,j,\theta)$ only affects tensor entries on the i-th and j-th slice. This requires $O(d^2)$ operations per iteration. In Section D of the supplemental material we provide a different version of this



(a) 10,000 samples (b) 200,000 samples

Figure 3. Clustering performance in terms of normalized MI of the Givens algorithm vs. the tensor power method of Anandkumar et al. (2012a). Clustering by fitting a GMM from samples drawn from a 20-component GMM with varying dimension, using 3rd order moments. Reconstruction is performed from (a) 10K and (b) 200K samples. Blue line with triangles marks the Givens coordinate method. Red line with circles marks the tensor power method, and the black line is the optimal performance if all GMM parameters are known.

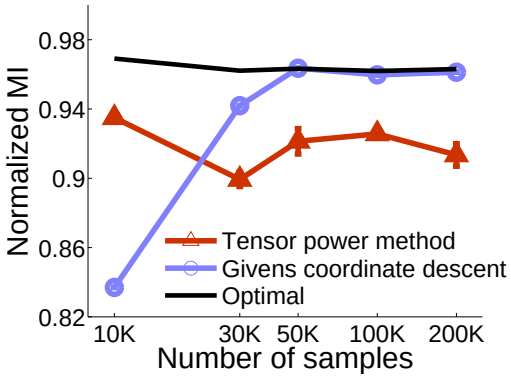


Figure 4. Same task as Figure 3, but for fixed dimension $d = 100$ and varying number of samples.

algorithm which does not require calculating the tensor T. Instead, it operates directly on the data points, calculating cross products on demand. This version of the algorithm has complexity per step of $O(\#samples)$ instead.

5.3. Experiments

Hsu & Kakade (2013) and Anandkumar et al. (2012a) have recently shown how fitting a Gaussian Mixture Model (GMM) with common spherical covariance can be reduced to orthogonally decomposing a third moment tensor. We evaluate the Givens coordinate minimization algorithm using this problem. We compare with a state of the art tensor decomposition method, the robust tensor power method, as given in Anandkumar et al. (2012a).

We generated GMMs with the following parameters: number of dimensions in $\{10, 20, 50, 100, 200\}$, number of samples in $\{10K, 30K, 50K, 100K, 200K\}$. We used 20 components, each with a spherical variance of 2. The centers were sampled from a Gaussian distribution with an inverse-Wishart distributed covariance matrix. Given the samples, we constructed the 3rd order moment, decomposed it, and reconstructed the model following the procedure in Anandkumar et al. (2012a). We then clustered the samples according to the reconstructed model, and measured the *normalized mutual information* (NMI) (Manning et al., 2008) between the learned clustering and the true clusters.

Figure 3 compares the performance of the two methods with the optimal NMI across dimensions. The coordinate minimization method outperforms the tensor power method for the large sample size (200K), whereas for small sample size (10K) the tensor power method performs better for the intermediate dimensions. In Figure 4 we see the performance of both algorithms across all sample sizes for dimension $= 100$. We see that the coordinate minimization method again performs better for larger sample sizes. We observed this phenomenon for 50 components as well, and for mixture models with larger variance.

6. Conclusion

We described a framework to efficiently optimize differentiable functions over the manifold of orthogonal matrices. The approach is based on Givens rotations, which we show can be viewed as the parallel of coordinate updates in Euclidean spaces. We prove the procedure's convergence to a local optimum. Using this framework, we developed algorithms for two unsupervised learning problems: Finding sparse principal components; and learning a Gaussian mixture model through orthogonal tensor decomposition. Our method poses an alternative to the tensor power method for orthogonal tensor decompositions. Our alternative extends the way the Jacobi eigenvalue algorithm is an alternative to the matrix power method for matrix decompositions.

We expect that the proposed framework can be further extended to other problems requiring learning over orthogonal matrices including ICA. Moreover, coordinate descent approaches have some inherent advantages and are sometimes better amenable to parallelization. Developing distributed Givens-rotation algorithms would be an interesting future research direction.

Acknowledgments

We wish to thank the anonymous reviewers for numerous improvements to the paper, and Haim Avron for fruitful discussions. U.S. is a recipient of the Google Europe Fellowship in Machine Learning, and this research is supported in part by this Google Fellowship. G.C. was supported by the Israeli science foundation grant 1090/12, and by a Marie Curie reintegration grant PIRG06-GA-2009-256566.

References

Absil, P-A, Mahony, Robert, and Sepulchre, Rodolphe. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

Anandkumar, Anima, Ge, Rong, Hsu, Daniel, Kakade, Sham M, and Telgarsky, Matus. Tensor decompositions for learning latent variable models. *arXiv preprint arXiv:1210.7559*, 2012a.

Anandkumar, Anima, Ge, Rong, Hsu, Daniel, and Kakade, Sham M. A tensor spectral approach to learning mixed membership community models. *arXiv preprint arXiv:1302.2684*, 2013.

Anandkumar, Animashree, Foster, Dean P, Hsu, Daniel, Kakade, Sham M, and Liu, Yi-Kai. A spectral algorithm for latent dirichlet allocation. *arXiv preprint arXiv:1204.6703*, 2012b.

Anandkumar, Animashree, Hsu, Daniel, and Kakade, Sham M. A method of moments for mixture models and hidden markov models. *arXiv preprint arXiv:1203.0683*, 2012c.

Bertsekas, Dimitri P. *Nonlinear programming*. Athena Scientific, 1999.

Chaganty, Arun Tejasvi and Liang, Percy. Spectral experts for estimating mixtures of linear regressions. *arXiv preprint arXiv:1306.3729*, 2013.

d'Aspremont, Alexandre, El Ghaoui, Laurent, Jordan, Michael I, and Lanckriet, Gert RG. A direct formulation for sparse pca using semidefinite programming. *SIAM review*, 49(3):434–448, 2007.

Golub, Gene H and Van Loan, Charles F. *Matrix computations*, volume 3. JHUP, 2012.

Hawrylycz, Michael J, Lein, S, Guillozet-Bongaarts, Angela L, Shen, Elaine H, Ng, Lydia, Miller, Jeremy A, van de Lagemaat, Louie N, Smith, Kimberly A, Ebbert, Amanda, Riley, Zackery L, et al. An anatomically comprehensive atlas of the adult human brain transcriptome. *Nature*, 489(7416):391–399, 2012.

Hsu, Daniel and Kakade, Sham M. Learning mixtures of spherical gaussians: moment methods and spectral decompositions. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pp. 11–20. ACM, 2013.

Ishteva, Mariya, Absil, P-A, and Van Dooren, Paul. Jacobi algorithm for the best low multilinear rank approximation of symmetric tensors. *SIAM Journal on Matrix Analysis and Applications*, 34(2):651–672, 2013.

Journée, Michel, Nesterov, Yurii, Richtárik, Peter, and Sepulchre, Rodolphe. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

Kolda, Tamara G and Bader, Brett W. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Manning, Christopher D, Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

Nemirovski, A. Optmization II Numerical Methods for Nonlinear Continuous Optimization. 1999.

Nesterov, Yu. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Patrascu, Andrei and Necoara, Ion. Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *arXiv preprint arXiv:1305.4027*, 2013.

Richtárik, Peter and Takáč, Martin. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, pp. 1–38, 2012.

Singh, Dinesh, Febbo, Phillip G, Ross, Kenneth, Jackson, Donald G, Manola, Judith, Ladd, Christine, Tamayo, Pablo, Renshaw, Andrew A, D'Amico, Anthony V, Richie, Jerome P, et al. Gene expression correlates of clinical prostate cancer behavior. *Cancer cell*, 1(2):203–209, 2002.

Zhang, Youwei and Ghaoui, Laurent El. Large-scale sparse principal component analysis with application to text data. *arXiv preprint arXiv:1210.7054*, 2012.

Zhang, Youwei, dAspremont, Alexandre, and El Ghaoui, Laurent. Sparse pca: Convex relaxations, algorithms and applications. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, pp. 915–940. Springer, 2012.

Zou, Hui, Hastie, Trevor, and Tibshirani, Robert. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.