JMLR: Workshop and Conference Proceedings 44 (2015) 116-129 NIPS 2015

The 1st International Workshop "Feature Extraction: Modern Questions and Challenges"

Convolutional Dictionary Learning through Tensor Factorization

Furong Huang FURONGH@UCI.EDU
Animashree Anandkumar A.ANANDKUMAR@UCI.EDU
Electrical Engineering and Computer Science Dept.
University of California, Irvine
Irvine, USA 92697, USA

Editor: Dmitry Storcheus

Abstract

Tensor methods have emerged as a powerful paradigm for consistent learning of many latent variable models such as topic models, independent component analysis and dictionary learning. Model parameters are estimated via CP decomposition of the observed higher order input moments. In this paper, we extend tensor decomposition framework to models with invariances, such as convolutional dictionary models. Our tensor decomposition algorithm is based on the popular alternating least squares (ALS) method, but with additional shift invariance constraints on the factors. We demonstrate that each ALS update can be computed efficiently using simple operations such as fast Fourier transforms and matrix multiplications. Our algorithm converges to models with better reconstruction error and is much faster, compared to the popular alternating minimization heuristic, where the filters and activation maps are alternately updated.

Keywords: Tensor CP decomposition, convolutional dictionary learning, convolutional ICA, blind deconvolution.

1. Introduction

Feature or representation learning forms a cornerstone of modern machine learning. Representing the data in the relevant feature space is critical to obtaining good performance in challenging machine learning tasks in speech, computer vision and natural language processing. A popular representation learning framework is based on dictionary learning. Here, the input data is modeled as a linear combination of dictionary elements. However, this model fails to incorporate natural domain-specific invariances such as shift invariance and results in highly redundant dictionary elements, which makes inference in these models expensive.

These shortcomings can be remedied by incorporating invariances into the dictionary model, and such models are known as convolutional models. Convolutional models are ubiquitous in machine learning for image, speech and sentence representations (Zeiler et al., 2010; Kavukcuoglu et al., 2010; Bristow et al., 2013), and in neuroscience for modeling neural spike trains (Olshausen, 2002; Ekanadham et al., 2011). Deep convolutional neural networks are a multi-layer extension of these models with non-linear activations. Such models have revolutionized performance in image, speech and natural language processing (Zeiler et al., 2010; Kalchbrenner et al., 2014). The convolutional dictionary learning model posits that the input signal x is generated as a linear combination of convolutions of unknown dictionary elements or *filters* $f_1^*, \dots f_L^*$ and unknown *activation maps* $w_1^*, \dots w_L^*$:

$$x = \sum_{i \in [L]} f_i^* * w_i^*, \tag{1}$$

where $[L] := 1, \ldots, L$. The vector w_i^* denotes the activations at locations, where the corresponding filter f_i^* is active.

In order to learn the model in (1), usually a square loss reconstruction criterion is employed:

$$\min_{f_i, w_i : \|f_i\|=1} \|x - \sum_{i \in [L]} f_i * w_i\|^2. \tag{2}$$

The constraints ($\|f_i\| = 1$) are enforced, since otherwise, the scaling can be exchanged between the filters f_i and the activation maps w_i. Also, an additional regularization term (for example an ℓ_1 term on the w_i's) is usually added to the above objective to promote sparsity on w_i.

A popular heuristic for solving (2) is based on alternating minimization (Bristow and Lucey, 2014), where the filters f_i are optimized, while keeping the activations w_i fixed, and vice versa. Each alternating update can be solved efficiently (since it is linear in each of the variables). However, the method is computationally expensive in the large sample setting since each iteration requires a pass over all the samples, and in modern machine learning applications, the number of samples can run into billions. Moreover, alternating minimization has multiple spurious local optima, and reaching the global optimum of (2) is NP-hard in general. This problem is severely amplified in the convolutional setting due to additional symmetries, compared to the usual dictionary learning setting (without the convolutional operation). Due to shift invariance of the convolutional operator, shifting a filter f_i by some amount, and applying a corresponding negative shift on the activation w_i leaves the objective in (2) unchanged. Can we design alternative methods for convolutional dictionary learning that are scalable to huge datasets?

1.1 Summary of Results

In this paper, we propose a novel framework for learning convolutional models through tensor decomposition. We consider inverse method of moments to estimate the model parameters via decomposition of higher order (third or fourth order) moment tensors. When the inputs x are generated from a convolutional model in (1), with independent activation maps w_i^*, i.e. a convolutional ICA model, we show that the cumulant tensors have a CP decomposition, whose components correspond to filters and their *circulant* shifts. We propose a novel method for tensor decomposition when such circulant constraints are imposed on the components of the tensor decomposition.

Our tensor decomposition method is a constrained form of the popular alternating least squares (ALS) method[1]. We show that the resulting optimization problem in each tensor ALS iteration can be solved in closed form, and uses simple operations such as Fast Fourier transforms (FFT) and matrix multiplications. These operations have a high degree of parallelism: for estimating L filters, each of length n, we require $O(\log n + \log L)$ time and $O(L^2 n^3)$ processors. Note that there is *no* dependence on the number of data samples N, since the empirical moment tensor can be computed in one data pass, and the ALS iterations only updates the filters. This is a huge saving in running time, compared to the alternate minimization method which requires a pass over data in each step to decode all the activation maps w_i. The running time of alternating minimization

1. The ALS method for tensor decomposition is not to be confused with the alternating minimization method for solving (2). While (2) acts on data samples and alternates between updating filters and activation maps, tensor ALS operates on averaged moment tensors and alternates between different modes of the tensor decomposition.

is $O(\max(\log n \log L, \log n \log N))$ per iteration with $O(\max(\frac{nNL}{\log N}, \frac{nNL}{\log L}))$ processors, and when $N \gg Ln^2$, which is the typical scenario, our method is hugely advantageous. Our method avoids decoding the activation maps in each iteration since they are averaged out in the input moment tensor, on which the ALS method operates and we only estimate the filters f_i in the learning step. In other words, the activation maps w_i's After filter estimation, the activation maps are easily estimated using (2) in one data pass. Thus, our method is highly parallel and scalable to huge datasets.

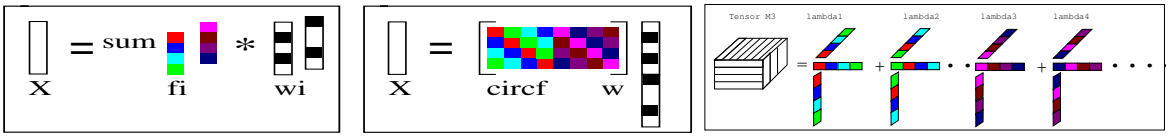
We carefully optimize computation and memory costs by exploiting tensor algebra and circulant structure, due to the shift invariance of the convolutional model. We implicitly carry out many of the operations and do not form large (circulant) matrices and minimize storage requirements. Preliminary experiments further demonstrate superiority of our method compared to alternating minimization. Our algorithm converges accurately and much faster to the true underlying filters compared to alternating minimization. Moreover, it results in much lower reconstruction error, while alternating minimization tends to get stuck in spurious local optima. Our algorithm is also orders of magnitude faster than the alternating minimization.

1.2 Related Works

The special case of (1) with one filter $(L = 1)$ is a well studied problem, and is referred to as *blind deconvolution* (Hyvärinen et al., 2004). In general, this problem is not identifiable, i.e. multiple equivalent solutions can exist (Choudhary and Mitra, 2014). It has been documented that in many cases alternating minimization produces trivial solutions, where the filter $f = x$ is the signal itself and the activation is the identity function (Levin et al., 2009). Therefore, alternative techniques have been proposed, such as convex programs, based on nuclear norm minimization (Ahmed et al., 2014) and imposing hierarchical Bayesian priors for activation maps (Wipf and Zhang, 2013). However, there is no analysis for settings with more than one filter. Incorporating Bayesian priors has shown to reduce the number of local optima, but not completely eliminate them (Wipf and Zhang, 2013; Krishnan et al., 2013). Moreover, Bayesian techniques are in general more expensive than alternating minimization.

The extension of blind deconvolution to multiple filters is known as convolutive blind source separation or convolutive independent component analysis (ICA) (Hyvärinen et al., 2004). Previous methods directly reformulate convolutive ICA as an ICA model, without incorporating the shift constraints. Moreover, reformulation leads to an increase of number of hidden sources from L to nL in the new model, where n is the input dimension, which are harder to separate and computationally more expensive. Other methods are based on performing ICA in the Fourier domain, but the downside is that the new mixing matrix depends on the angular frequency, and leads to permutation and sign indeterminacies of the sources across frequencies. Complicated interpolation methods (Hyvärinen et al., 2004) overcome these indeterminacies. In contrast, our method avoids all these issues. We do not perform Fourier transform on the input, instead, we employ FFTs at different iterations of our method to estimate the filters efficiently.

The dictionary learning problem without convolution has received much attention. Recent results show that simple iterative methods can learn the globally optimal solution (Agarwal et al., 2014; Arora et al., 2014). In addition, tensor decomposition methods provably learn the model, when the activations are independently drawn (the ICA model) (Anandkumar et al., 2014) or are sparse (the sparse coding model) (Anandkumar et al., 2015). In this work, we extend the tensor decomposition methods to efficiently incorporate the shift invariance constraints imposed by the convolution operator.



(a) (b) **(c)** Convolutional tensor decomposition

Figure 1: Convolutional tensor decomposition for learning convolutional ICA models.(a) The convolutional generative model with 2 filters. (b) Reformulated model where $\mathcal{F}^*$ is column-stacked circulant matrix. (c) The third order cumulant is decomposed as filters.

2. Model and Formulation

Notation Let $[n] := \{1, 2, \ldots, n\}$. For a vector v, denote the i^{th} element as $v(i)$. For a matrix M, denote the i^{th} row as M^i and j^{th} column as M_j. For a tensor $T \in \mathbb{R}^{n\times n\times n}$, its $(i_1, i_2, i_3)^{\text{th}}$ entry is denoted by $[T]_{i_1,i_2,i_3}$. A *column-stacked* matrix M consisting of M_i's (with same number of rows) is $M := [M_1, M_2, \ldots, M_L]$. Similarly, a *row-stacked* matrix M from M_i's (with same number of columns) is $M := [M_1; M_2; \ldots; M_L]$.

Cyclic Convolution The 1-dimensional (1-D) n-cyclic convolution $f * w$ between vectors f and w is defined as $v = f *_n w,\ v(i) = \sum_{j\in[n]} f(j)w((i-j+1) \mod n)$. Note that the linear convolution is the combination without the modulo operation (i.e. cyclic shifts) above. n-Cyclic convolution is equivalent to linear convolution, when n is at least twice the support length of both f and w (Oppenheim and Willsky, 1997), which will be assumed. We drop the notation n in $*$ for convenience. Cyclic convolution in (2) is equivalent to $f * w = \mathsf{Cir}(f) \cdot w$, and

$$\mathsf{Cir}(f) := \sum_p f(p)G_p \in \mathbb{R}^{n\times n}, \quad (G_p)^i_j := \delta\left\{((i-j) \mod n) = p-1\right\}, \quad \forall p \in [n]. \tag{3}$$

defines a circulant matrix. A circulant matrix $\mathsf{Cir}(f)$ is characterized by the vector f, and each column corresponds to a cyclic shift of f.

Properties of circulant matrices Let F be the discrete Fourier transform matrix whose (m,k)-th entry is $F^m_k = \omega_n^{(m-1)(k-1)}, \forall m, k \in [n]$ where $\omega_n = \exp(-\frac{2\pi i}{n})$. If $U := \sqrt{n}F^{-1}$, U is the set of eigenvectors for all $n \times n$ circulant matrices (Gray, 2005). Let the Discrete Fourier Transform of a vector f be $\mathsf{FFT}(f)$, we express the circulant matrix $\mathsf{Cir}(f)$ as

$$\mathsf{Cir}(f) = U\mathsf{diag}(F \cdot f)U^{\mathsf{H}} = U\mathsf{diag}(\mathsf{FFT}(f))U^{\mathsf{H}}. \tag{4}$$

This is an important property we use in algorithm optimization to improve computational efficiency.

Column stacked circulant matrices We will extensively use column stacked circulant matrices $\mathcal{F} := [\mathsf{Cir}(f_1), \ldots, \mathsf{Cir}(f_L)]$, where $\mathsf{Cir}(f_j)$ is the circulant matrix corresponding to filter f_j.

2.1 Convolutional Dictionary Learning/ICA Model

We assume that the input $x \in \mathbb{R}^n$ is generated as

$$x = \sum_{j\in[L]} f_j^* * w_j^* = \sum_{j\in[L]} \mathsf{Cir}(f_j^*)w_j^* = \mathcal{F}^* \cdot w^*, \tag{5}$$

where $\mathcal{F}^* := [\mathsf{Cir}(f_1^*), \mathsf{Cir}(f_2^*), \ldots, \mathsf{Cir}(f_L^*)]$ is the concatenation or column stacked version of circulant matrices and w^* is the *row-stacked* vector $w^* := [w_1^*; w_2^*; \ldots w_L^*] \in \mathbb{R}^{nL}$. Recall that

$\mathsf{Cir}(f_l^*)$ is circulant matrix corresponding to filter f_l^*, as given by (4). Note that although $\mathcal{F}^*$ is a n by nL matrix, there are only nL free parameters. We never explicitly form the estimates $\mathcal{F}$ of $\mathcal{F}^*$, but instead use filter estimates f_l's to characterize $\mathcal{F}$. In addition, we can handle additive Gaussian noise in (5), but do not incorporate it for simplicity.

Activation Maps: For each observed sample x, the activation map w_i^* in (5) indicates the locations where each filter f_i^* is active and w^* is the *row-stacked* vector $w^* := [w_1^*; w_2^*; \dots w_L^*]$. We assume that the coordinates of w^* are drawn from some product distribution, i.e. different entries are independent of one another and we have the independent component analysis (ICA) model in (5). When the distribution encourages sparsity, e.g. Bernoulli-Gaussian, only a small subset of locations are active, and we have the *sparse coding* model in that case. We can also extend to dependent distributions such as Dirichlet for w^*, along the lines of (Blei et al., 2003), but limit ourselves to ICA model for simplicity.

Learning Problem: Given access to N i.i.d. samples, $X := [x^1, x^2, \dots, x^N] \in \mathbb{R}^{n \times N}$, generated according to the above model, we aim to estimate the true filters f_i^*, for $i \in [L]$. Once the filters are estimated, we can use standard decoding techniques, such as the square loss criterion in (2) to learn the activation maps for the individual maps. We focus on developing a novel method for filter estimation in this paper.

3. Form of Cumulant Moment Tensors

Tensor Preliminaries We consider 3rd order tensors in this paper but the analysis is easily extended to higher order tensors. For tensor $T \in \mathbb{R}^{n \times n \times n}$, its $(i_1, i_2, i_3)^{\text{th}}$ entry is denoted by $[T]_{i_1,i_2,i_3}, \forall i_1 \in [n], i_2 \in [n], i_3 \in [n]$. A flattening or unfolding of tensor $T \in \mathbb{R}$ is the column-stacked matrix of all its slices, given by $\text{unfold}(T) := [[T]_{:,:,1}, [T]_{:,:,2}, \dots, [T]_{:,:,n}] \in \mathbb{R}^{n \times n^2}$. Define the Khatri-Rao product for vectors $u \in \mathbb{R}^a$ and $v \in \mathbb{R}^b$ as a *row-stacked* vector $[u \odot v] := [u(1)v; u(2)v; \dots; u(a)v] \in \mathbb{R}^{ab}$. Khatri-Rao product is also defined for matrices with same columns. For $M \in \mathbb{R}^{a \times c}$ and $M' \in \mathbb{R}^{b \times c}$, $M \odot M' := [M_1 \odot M_1', \dots, M_c \odot M_c',] \in \mathbb{R}^{ab \times c}$, where M_i denotes the i^{th} column of M.

Cumulant The third order cumulant of a multivariate distribution is a third order tensor, which uses (raw) moments up to third order. Let $\mathrm{C}_3 \in \mathbb{R}^{n \times n^2}$ denote the unfolded version of third order cumulant tensor, it is given by

$$\mathrm{C}_3 := \mathbb{E}[x(x \odot x)^\top] - \text{unfold}(Z) \tag{6}$$

where $[Z]_{a,b,c} := \mathbb{E}[x_a]\mathbb{E}[x_b x_c] + \mathbb{E}[x_b]\mathbb{E}[x_a x_c] + \mathbb{E}[x_c]\mathbb{E}[x_a x_b] - 2\mathbb{E}[x_a]\mathbb{E}[x_b]\mathbb{E}[x_c],\ \forall a, b, c \in [n]$.

Under the convolution ICA model in Section 2.1, we show that the third order cumulant has a nice tensor form, as given below.

Lemma 1 (Form of Cumulants) *The unfolded third order cumulant* C_3 *in* (6) *has the following decomposition form*

$$\mathrm{C}_3 = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* (\mathcal{F}_j^* \odot \mathcal{F}_j^*)^\top = \mathcal{F}^* \Lambda^* (\mathcal{F}^* \odot \mathcal{F}^*)^\top, \quad \textit{where } \Lambda^* := \mathsf{diag}(\lambda_1^*, \lambda_2^*, \dots, \lambda_{nL}^*) \tag{7}$$

where $\mathcal{F}_j^*$ *denotes the* j^{th} *column of the* column-stacked *circulant matrix* $\mathcal{F}^*$ *and* λ_j^* *is the third order cumulant corresponding to the (univariate) distribution of* $w^*(j)$.

For example, if the l^{th} activation is drawn from a Poisson distribution with mean $\tilde{\lambda}$, we have that $\lambda_l^* = \tilde{\lambda}$. Note that if the third order cumulants of the activations, i.e. λ_j^*'s, are zero, we need to consider higher order cumulants. This holds for zero-mean activations and we need to use fourth order cumulant instead. Our method extends in a straightforward manner for higher order cumulants.

$$\mathcal{F} = \begin{array}{|c|c|c|} \hline \mathrm{blk}_1(\mathcal{F}) & \dots & \mathrm{blk}_L(\mathcal{F}) \\ \hline \end{array} \qquad \boldsymbol{\Psi} = \begin{array}{|c|c|c|} \hline \mathrm{blk}_1^1(\boldsymbol{\Psi}) & \cdots & \mathrm{blk}_L^1(\boldsymbol{\Psi}) \\ \hline \cdots & \cdots & \cdots \\ \hline \mathrm{blk}_1^L(\boldsymbol{\Psi}) & \cdots & \mathrm{blk}_L^L(\boldsymbol{\Psi}) \\ \hline \end{array}$$

Figure 2: (a) Blocks of the column-stacked circulant matrix $\mathcal{F}$. (b) Blocks of the row-and-column-stacked diagonal matrices $\boldsymbol{\Psi}$. $\mathrm{blk}_j^i(\boldsymbol{\Psi})$ is diagonal.

The decomposition form in (7) is known as the CANDECOMP/PARAFAC (CP) decomposition form (Anandkumar et al., 2014) (the usual form has the decomposition of the tensor and not its unfolding, as above). We now attempt to recover the unknown filters f_i^* through decomposition of the third order cumulants C_3. This is formally stated below.

Objective Function: Our goal is to obtain filter estimates f_i's which minimize the Frobenius norm $\|\cdot\|_{\mathbb{F}}$ of reconstruction of the cumulant tensor C_3,

$$\begin{aligned} &\min_{\mathcal{F}} \quad \|\mathrm{C}_3 - \mathcal{F}\Lambda\,(\mathcal{F}\odot\mathcal{F})^\top\|_F^2, \\ &\text{s.t. } \mathrm{blk}_l(\mathcal{F}) = U\mathsf{diag}(\mathsf{FFT}(f_l))U^{\mathsf{H}},\ \|f_l\|_2 = 1, \quad \forall l\in[L], \quad \Lambda = \mathsf{diag}(\lambda). \end{aligned} \tag{8}$$

where $\mathrm{blk}_l(\mathcal{F})$ denotes the l^{th} circulant matrix in $\mathcal{F}$. The conditions in (8) enforce $\mathrm{blk}_l(\mathcal{F})$ to be circulant and for the filters to be normalized. Recall that U denotes the eigenvectors for circulant matrices. The rest of the paper is devoted to devising efficient methods to solve (8).

Throughout the paper, we will use $\mathcal{F}_j$ to denote the j^{th} column of $\mathcal{F}$, and $\mathrm{blk}_l(\mathcal{F})$ to denote the l^{th} circulant matrix block in $\mathcal{F}$. Note that $\mathcal{F}\in\mathbb{R}^{n\times nL}$, $\mathcal{F}_j\in\mathbb{R}^n$ and $\mathrm{blk}_l(\mathcal{F})\in\mathbb{R}^{n\times n}$.

4. Alternating Least Squares for Convolutional Tensor Decomposition

To solve the non-convex optimization problem in (8), we consider the alternating least squares (ALS) method with *column stacked* circulant constraint. We first consider the asymmetric relaxation of (8) and introduce separate variables $\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$ for filter estimates along each of the modes to fit the third order cumulant tensor C_3. We then perform alternating updates by fixing two of the modes and updating the third one.

$$\min_{\mathcal{F}} \quad \|\mathrm{C}_3 - \mathcal{F}\Lambda\,(\mathcal{H}\odot\mathcal{G})^\top\|_F^2 \text{ s.t. } \mathrm{blk}_l(\mathcal{F}) = U\cdot\mathsf{diag}(\mathsf{FFT}(f_l))\cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l\in[L] \tag{9}$$

Similarly, $\mathcal{G}$ and $\mathcal{H}$ have the same column-stacked circulant matrix constraint and are updated similarly in alternating steps. The diagonal matrix Λ is updated through normalization.

We now introduce the *Convolutional Tensor* (CT) Decomposition algorithm to efficiently solve (9) in closed form, using simple operations such as matrix multiplications and fast Fourier Transform

(FFT). We do not form matrices $\mathcal{F}, \mathcal{G}$ and $\mathcal{H} \in \mathbb{R}^{n \times nL}$, which are large, but only update them using filter estimates $f_1, \ldots, f_L, g_1, \ldots, g_L, h_1, \ldots h_L$. Denote

$$\mathrm{M} := \mathrm{C}_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger, \tag{10}$$

where † denotes pseudoinverse. Let $\mathrm{blk}_l(\mathrm{M})$ and $\mathrm{blk}_l(\Lambda)$ denote the l^{th} blocks of M and Λ. We have a closed form solution for filter update, once we have computed M, and we present the main result as follows.

Theorem 2 *[Closed form updates] The optimal solution f_l^{opt} for (27) is given by*

$$f_l^{\text{opt}}(p) = \frac{\sum\limits_{i,j \in [n]} \| \mathrm{blk}_l(\mathrm{M})_j \|^{-1} \cdot \mathrm{blk}_l(\mathrm{M})_j^i \cdot I_{p-1}^q}{\sum\limits_{i,j \in [n]} I_{p-1}^q}, \qquad \forall p \in [n], q := (i-j) \mod n. \tag{11}$$

Further $\Lambda = \mathsf{diag}(\lambda)$ is updated as $\lambda(i) = \| \mathrm{M}_i \|$, for all $i \in [nL]$. Note that I_{p-1}^q denotes the $(q, (p-1))^{th}$ element of the identity matrix.

Proof Sketch: Using the property of least squares, the optimization problem in (9) is equivalent to

$$\min_{\mathcal{F}} \|\mathrm{C}_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger \Lambda^\dagger - \mathcal{F}\|_F^2 \text{ s.t. } \mathrm{blk}_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}}, \ \|f_l\|_2^2 = 1, \forall l \in [L] \tag{12}$$

when $(\mathcal{H} \odot \mathcal{G})$ and Λ are full column rank. The full rank condition requires $nL < n^2$ or $L < n$, and it is a reasonable assumption since otherwise the filter estimates are redundant. In practice, we can additionally regularize the update to ensure full rank condition is met. Since (26) has block constraints, it can be broken down in to solving L independent sub-problems

$$\min_{f_l} \left\| \mathrm{blk}_l(\mathrm{M}) \cdot \mathrm{blk}_l(\Lambda)^\dagger - U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}} \right\|_F^2 \quad s.t. \quad \|f_l\|_2^2 = 1, \forall l \in [L] \tag{13}$$

Our proof for the closed form solution is similar to the analysis in (Eberle and Maciel, 2003), where they proposed a closed form solution for finding the closest circulant/toeplitz matrix. For a detailed proof of Theorem 2, see Appendix B. □

Thus, the reformulated problem in (27) can be solved in closed form efficiently. A bulk of the computational effort will go into computing M in (10). Computation of M requires $2L$ fast Fourier Transforms of length n filters and simple matrix multiplications without explicitly forming $\mathcal{G}$ or $\mathcal{H}$. We make this concrete in the next section. The closed form update after getting M is highly parallel. With $O(n^2 L / \log n)$ processors, it takes $O(\log n)$ time.

5. Algorithm Optimization to Reduce Memory and Computational Costs

We now focus on estimating $\mathrm{M} := \mathrm{C}_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger$ in (10). If done naively, this requires inverting $n^2 \times nL$ matrix and multiplication of $n \times n^2$ and $n^2 \times nL$ matrices with $O(n^6)$ time. However, forming and computing with these matrices is very expensive when n (and L) are large. Instead, we utilize the properties of circulant matrices and the Khatri-Rao product $\odot$ to efficiently carry out these computations implicitly. We present our final result on computational complexity of the proposed method. Recall that n is the filter size and L is the number of filters.

Lemma 3 ***[Computational Complexity]*** *With multi-threading, the running time of our algorithm for n dimensional input and L number of filters is $O(\log n + \log L)$ per iteration using $O(L^2 n^3)$ processors.*

Note that before the iterative updates, we compute the third order cumulant[2] C_3 once whose computational complexity is $O(\log N)$ with $\frac{N}{\log N}$ processors, where N is the number of samples. However, this operation is not iterative. In contrast, alternating minimization (AM) requires pass over all the data samples in each iteration, while our algorithm requires only one pass of the data.

The parallel computational complexity of AM is as follows. In each iteration of AM, computing the derivative with respect to either filters or activation maps requires NL number of FFTs (requires $O(NLn \log n)$ serial time), and the degrees of parallelism are $O(Nn \log L)$ and $O(Nn \log n)$ respectively. Therefore with multi-threading, the running time of AM is $O(\max(\log n \log L, \log n \log N))$ per iteration using $O(\max(\frac{nNL}{\log N}, \frac{nNL}{\log L}))$ processors. Comparing with Lemma 3, we find that our algorithm is advantageous in the regime of $N \geq Ln^2$, which is the typical regime in applications.

Let us describe how we utilize various algebraic structures to obtain efficient computation.

Property 1 (Khatri-Rao product): $((\mathcal{H} \odot \mathcal{G})^\top)^\dagger = (\mathcal{H} \odot \mathcal{G})((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$, where $.\star$ denotes element-wise product.

Computational Goals: Find $((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ first and multiply the result with $\mathrm{C}_3(\mathcal{H} \odot \mathcal{G})$ to find M.

We now describe in detail how to carry out each of these steps.

5.1 Challenge: Computing $((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$

A naive implementation to find the matrix inversion $((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ is very expensive. However, we incorporate the stacked circulant structure of $\mathcal{G}$ and $\mathcal{H}$ to reduce computation. Note that this is not completely straightforward since although $\mathcal{G}$ and $\mathcal{H}$ are column stacked circulant matrices, the resulting product whose inverse is required, is *not* circulant. Below, we show that however, it is partially circulant along different rows and columns.

Property 2 (Block circulant matrix): The matrix $(\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G})$ consists of row and column stacked circulant matrices.

We now make the above property precise by introducing some new notation. Define column stacked identity matrix $\mathbf{I} := [I, \ldots, I] \in \mathbb{R}^{n \times nL}$, where I is $n \times n$ identity matrix. Let $\mathbf{U} := \mathrm{Blkdiag}(U, U, \ldots U) \in \mathbb{R}^{nL \times nL}$ be the block diagonal matrix with U along the diagonal. The first thing to note is that $\mathcal{G}$ and $\mathcal{H}$, which are column stacked circulant matrices, can be written as

$$\mathcal{G} = \mathbf{I} \cdot \mathbf{U} \cdot \mathsf{diag}(v) \cdot \mathbf{U}^{\mathsf{H}}, \quad v := [\mathsf{FFT}(g_1); \mathsf{FFT}(g_2); \ldots; \mathsf{FFT}(g_L)], \tag{14}$$

where $g_1, \ldots, g_L$ are the filters corresponding to $\mathcal{G}$, and similarly for $\mathcal{H}$, where the diagonal matrix consists of FFT coefficients of the respective filters $h_1, \ldots, h_L$.

By appealing to the above form, we have the following result. We use the notation $\mathrm{blk}^i_j(\mathbf{\Psi})$ for a matrix $\mathbf{\Psi} \in \mathbb{R}^{nL \times nL}$ to denote $(i,j)^{\text{th}}$ block of size $n \times n$.

Lemma 4 (Form of $(\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G})$) *We have*

$$((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger = \mathbf{U} \cdot \mathbf{\Psi}^\dagger \cdot \mathbf{U}^{\mathsf{H}}, \tag{15}$$

2. Instead of computing the cumulant tensor C_3, a randomized sketch can be computed efficiently, following the recent work of (Wang et al., 2015), and the ALS updates can be performed efficiently without forming the cumulant tensor C_3.

where $\mathbf{\Psi} \in \mathbb{R}^{nL \times nL}$ has L by L blocks, each block of size $n \times n$. Its $(j,l)^{th}$ block is given by

$$\mathrm{blk}_l^j(\mathbf{\Psi}) = \mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(g_j)) \cdot \mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(h_j)) \cdot \mathsf{diag}(\mathsf{FFT}(g_l)) \cdot \mathsf{diag}(\mathsf{FFT}(h_l)) \in \mathbb{R}^{n \times n} \quad (16)$$

Therefore, the inversion of $(\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G})$ can be reduced to the inversion of row-and-column stacked set of diagonal matrices which form $\mathbf{\Psi}$. Computing $\mathbf{\Psi}$ simply requires FFT on all $2L$ filters $g_1, \ldots, g_L$ and $h_1, \ldots, h_L$, i.e. $2L$ FFTs, each on length n vector. We propose an efficient iterative algorithm to compute $\mathbf{\Psi}^\dagger$ via block matrix inversion theorem(Golub and Van Loan, 2012) in Appendix C.

5.2 Challenge: Computing $\mathrm{M} = \mathrm{C}_3(\mathcal{H} \odot \mathcal{G}) \cdot ((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$

Now that we have computed $((\mathcal{H}^\top \mathcal{H}) . \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ efficiently, we need to compute the resulting matrix with $\mathrm{C}_3(\mathcal{H} \odot \mathcal{G})$ to obtain M. We observe that the m^{th} row of the result M is given by

$$M^m = \sum_{j \in [nL]} \mathbf{U}^j \mathsf{diag}^{\mathsf{H}}(z)\, \Phi^{(m)} \mathsf{diag}(v)\, (\mathbf{U}^j)^{\mathsf{H}} \mathbf{U}^j \mathbf{\Psi}^\dagger \mathbf{U}^{\mathsf{H}}, \quad \forall m \in [nL], \quad (17)$$

where $v := [\mathsf{FFT}(g_1); \ldots; \mathsf{FFT}(g_L)]$, $z := [\mathsf{FFT}(h_1); \ldots; \mathsf{FFT}(h_L)]$ are concatenated FFT coefficients of the filters, and

$$\Phi^{(m)} := \mathbf{U}^{\mathsf{H}} \mathbf{I}^\top \Gamma^{(m)} \mathbf{I} \mathbf{U}, \quad [\Gamma^{(m)}]_j^i := [\mathrm{C}_3]_{i+(j-1)n}^m, \quad \forall i, j, m \in [n] \quad (18)$$

Note that $\Phi^{(m)}$ and $\Gamma^{(m)}$ are fixed for all iterations and need to be computed only once. Note that $\Gamma^{(m)}$ is the result of taking m^{th} row of the cumulant unfolding C_3 and matricizing it. Equation (17) uses the property that ${\mathrm{C}_3}^m(\mathcal{H} \odot \mathcal{G})$ is equal to the diagonal elments of $\mathcal{H}^\top\, \Gamma^{(m)}\, \mathcal{G}$.

We now bound the cost for computing (17). (1) Inverting $\mathbf{\Psi}$ takes $O(\log L + \log n)$ time with $O(n^2 L^2/(\log n + \log L))$ processors according to appendix C. (2) Since $\mathsf{diag}(v)$ and $\mathsf{diag}(z)$ are diagonal and $\mathbf{\Psi}$ is a matrix with diagonal blocks, the overall matrix multiplication in equation (17) takes $O(L^2 n^2)$ time serially with $O(L^2 n^2)$ degree of parallelism for each row. Therefore the overall serial computation cost is $O(L^2 n^3)$ with $O(L^2 n^3)$ degree of parallelism. With multi-threading, the running time is $O(1)$ per iteration using $O(L^2 n^3)$ processes. (3) FFT requires $O(n \log n)$ serial time, with $O(n)$ degree of parallelism. Therefore computing $2L$ FFT's takes $O(\log n)$ time with $O(Ln)$ processors.

Combining the above discussion, it takes $O(\log L + \log n)$ time with $O(L^2 n^3)$ processors.

6. Experiments: Comparison with Alternating Minimization

We compare our convolutional tensor decomposition framework with solving equation (2) using alternating (between filters and activation map) minimization method where gradient descent is employed to update f_i and w_i alternatively. The error comparison between our proposed convolutional tensor algorithm and the alternating minimization algorithm is in figure 3a. We evaluate the errors for both algorithms by comparing the reconstruction of error and filter recovery error[3]. Our algorithm converges much faster to the solution than the alternating minimization algorithm. In fact, alternating minimization leads to spurious solution where the reconstruction error is significantly

3. Note that circulant shifts of the filters result in the same reconstruction error, and we report the lowest error between the estimated filters and all circulant shifts of the ground-truth.

larger compared to the error achieved by the tensor method. The error bump in the reconstruction error curve in figure 3a for tensor method is due to the random initialization following deflation of one filter, and estimation of the second one. The running time is also reported in figure 3b and 3c between our proposed convolutional tensor algorithm and the alternating minimization. Our algorithm is orders of magnitude faster than the alternating minimization. Both our algorithm and alternating minimization scale linearly with number of filters. However convolutional tensor algorithm is almost constant time with respect to the number of samples, whereas the alternating minimization scales linearly. This results in huge savings in running time for large datasets.

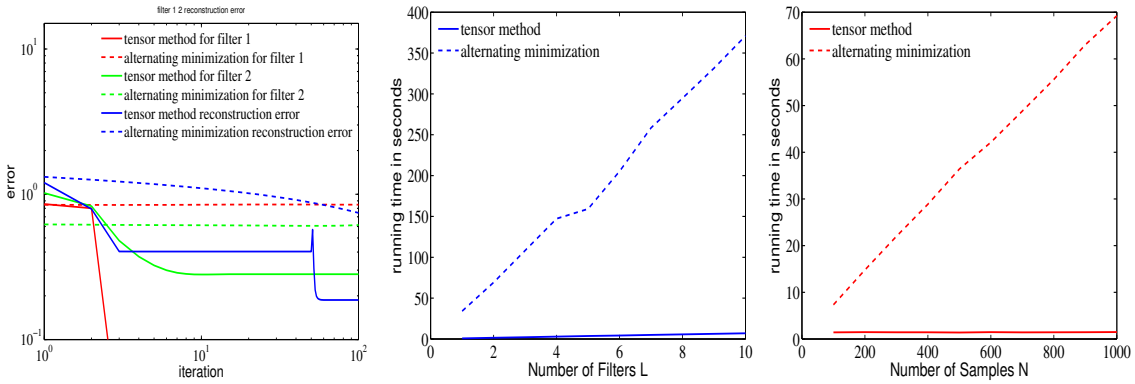


Figure 3: (a) Error comparison between our convolutional tensor method (proposed CT) and the baseline alternate minimization method (baseline AM). (b) Running time comparison between our proposed CT and the baseline AM method under varying L. (c) Running time comparison between CT and AM method under varying N.

7. Conclusion

In this paper, we proposed a novel tensor decomposition framework for learning convolutional dictionary models. Unlike the popular alternating minimization, our method avoids expensive decoding of activation maps in each step and can reach better solutions with faster run times. We derived efficient updates for tensor decomposition based on modified alternating least squares, and it consists of simple operations such as FFTs and matrix multiplications. Our framework easily extends to convolutional models for higher dimensional signals (such as images), where the circulant matrix is replaced with block circulant matrices (Gray, 2005). More generally, our framework can handle general group structure, by replacing the FFT operation with the appropriate group FFT (Kondor, 2008). By combining the advantages of tensor methods with a general class of invariant representations, we thus have a powerful paradigm for learning efficient latent variable models and embeddings in a variety of domains.

ACKNOWLEDGMENTS

We thank Cris Cecka for discussion on fast implementation of block matrix inverse, and we thank the initial discussions with Majid Janzamin and Hanie Sedghi on Toeplitz matrices (Eberle and Maciel, 2003).

References

A. Agarwal, A. Anandkumar, P. Jain, P. Netrapalli, and R. Tandon. Learning Sparsely Used Overcomplete Dictionaries. In *Conference on Learning Theory (COLT)*, June 2014.

Ali Ahmed, Benjamin Recht, and Justin Romberg. Blind deconvolution using convex programming. *Information Theory, IEEE Transactions on*, 60(3):1711–1732, 2014.

Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

Animashree Anandkumar, Rong Ge, and Majid Janzamin. Learning overcomplete latent variable models through tensor methods. In *Conference on Learning Theory (COLT)*, June 2015.

Sanjeev Arora, Rong Ge, and Ankur Moitra. New algorithms for learning incoherent and overcomplete dictionaries. In *Conference on Learning Theory (COLT)*, June 2014.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Hilton Bristow and Simon Lucey. Optimization methods for convolutional sparse coding. *arXiv preprint arXiv:1406.2407*, 2014.

Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 391–398. IEEE, 2013.

Sunav Choudhary and Urbashi Mitra. Sparse blind deconvolution: What cannot be done. In *Information Theory (ISIT), 2014 IEEE International Symposium on*, pages 3002–3006. IEEE, 2014.

Maria Gabriela Eberle and Maria Cristina Maciel. Finding the closest toeplitz matrix. *Computational & Applied Mathematics*, 22(1):1–18, 2003.

Chaitanya Ekanadham, Daniel Tranchina, and Eero P Simoncelli. A blind sparse deconvolution method for neural spike identification. In *Advances in Neural Information Processing Systems*, pages 1440–1448, 2011.

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

Robert M Gray. Toeplitz and circulant matrices: A review. *Communications and Information Theory*, 2(3):155–239, 2005.

Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann L Cun. Learning convolutional feature hierarchies for visual recognition. In *Advances in neural information processing systems*, pages 1090–1098, 2010.

Imre Risi Kondor. *Group theoretical methods in machine learning*. 2008.

Dilip Krishnan, Joan Bruna, and Rob Fergus. Blind deconvolution with non-local sparsity reweighting. *arXiv preprint arXiv:1311.4029*, 2013.

Anat Levin, Yair Weiss, Fredo Durand, and William T Freeman. Understanding and evaluating blind deconvolution algorithms. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1964–1971. IEEE, 2009.

Bruno A Olshausen. Sparse codes and spikes. *Probabilistic models of the brain: Perception and neural function*, pages 257–272, 2002.

Alan V Oppenheim and Alan S Willsky. *Signals and systems*. Prentice-Hall, 1997.

Yining Wang, Hsiao-Yu Tung, Alexander Smola, and Animashree Anandkumar. Fast and guaranteed tensor decomposition via sketching. In *Proc. of NIPS*, 2015.

David Wipf and Haichao Zhang. Revisiting bayesian blind deconvolution. *arXiv preprint arXiv:1305.2362*, 2013.

Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Robert Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

Appendix for Convolutional Dictionary Learning through Tensor Factorization

Appendix A. Cumulant Form

In Anandkumar et al. (2014), it is proved that in ICA model, the cumulant of observation x is decomposed into multi-linear transform of a diagonal cumulant of h. Therefore, we aim to find the third order cumulant for input x.

As we know that the r^{th} order moments for variable x is defined as

$$\mu_r := \mathbb{E}[x^r] \in \mathbb{R}^{n \times n \times n} \tag{19}$$

Let us use $[\mu_3]_{i,j,k}$ to denote the $(i,j,k)^{\text{th}}$ entry of the third order moment. The relationship between 3^{th} order cumulant $\kappa_3$and 3^{th} order moment $\mu_3$is

$$[\kappa_3]_{i,j,k} = [\mu_3]_{i,j,k} - [\mu_2]_{i,j}[\mu_1]_k - [\mu_2]_{i,k}[\mu_1]_j - [\mu_2]_{j,k}[\mu_1]_i + 2[\mu_1]_i[\mu_1]_j[\mu_1]_k \tag{20}$$

Therefore the shift tensor is in this format: We know that the shift term

$$[Z]_{a,b,c} := \mathbb{E}[x_a^i]\mathbb{E}[x_b^i x_c^i] + \mathbb{E}[x_b]\mathbb{E}[x_a x_c^i] + \mathbb{E}[x_c]\mathbb{E}[x_a x_b] - 2\mathbb{E}[x_a]\mathbb{E}[x_b]\mathbb{E}[x_c], \quad a,b,c \in [n] \tag{21}$$

It is known from Anandkumar et al. (2014) that cumulant decomposition in the 3 order tensor format is

$$\mathbb{E}[x \otimes x \otimes x] - Z = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* \otimes \mathcal{F}_j^* \otimes \mathcal{F}_j^* \tag{22}$$

Therefore using the Khatri-Rao product property,

$$\text{unfold}(\sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* \otimes \mathcal{F}_j^* \otimes \mathcal{F}_j^*) = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* (\mathcal{F}_j^* \odot \mathcal{F}_j^*)^\top = \mathcal{F}^* \Lambda^* \left(\mathcal{F}^* \odot \mathcal{F}^*\right)^\top \tag{23}$$

Therefore the unfolded third order cumulant is decomposed as $\mathrm{C}_3 = \mathcal{F}^* \Lambda^* \left(\mathcal{F}^* \odot \mathcal{F}^*\right)^\top$.

Appendix B. Proof for Main Theorem 2

Our optimization problem is

$$\min_{\mathcal{F}} \quad \|\mathrm{C}_3 - \mathcal{F}\Lambda \left(\mathcal{H} \odot \mathcal{G}\right)^\top\|_F^2 \text{ s.t. } \text{blk}_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}}, \ \|f_l\|_2^2 = 1, \forall l \in [L], \tag{24}$$

where we denote $D := \Lambda \left(\mathcal{H} \odot \mathcal{G}\right)^\top$ for simplicity. Therefore the objective is to minimize $\|\mathrm{C}_3 - \mathcal{F}D\|_F^2$. Let the SVD of D be $D = P\Sigma Q^\top$. Since the Frobenius norm remains invariant under orthogonal transformations and full rank diagonal matrix (Eberle and Maciel, 2003), it is obtained that

$$\|\mathrm{C}_3 - \mathcal{F}\, D\|_F^2 = \|\mathrm{C}_3 - \mathcal{F}\, P\Sigma Q^\top\|_F^2 = \|\mathrm{C}_3\, Q\Sigma^\dagger - \mathcal{F}\, P\|_F^2 = \|\mathrm{C}_3\, Q\Sigma^\dagger P^\top - \mathcal{F}\|_F^2 \tag{25}$$

Therefore the optimization problem in (9) is equivalent to

$$\min_{\mathcal{F}} \|\mathrm{C}_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger \Lambda^\dagger - \mathcal{F}\|_F^2 \text{ s.t. } \mathrm{blk}_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \quad (26)$$

when $(\mathcal{H} \odot \mathcal{G})$ and Λ are full column rank.

The full rank condition requires $nL < n^2$ or $L < n$, and it is a reasonable assumption since otherwise the filter estimates are redundant. Since (26) has block constraints, it can be broken down in to solving L independent sub-problems

$$\min_{f_l} \left\| \mathrm{blk}_l(\mathrm{M}) \cdot \mathrm{blk}_l(\Lambda)^\dagger - U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}} \right\|_F^2 \quad s.t. \quad \|f_l\|_2^2 = 1, \forall l \in [L]. \quad (27)$$

Appendix C. Parallel Inversion of $\mathbf{\Psi}$

We propose an efficient iterative algorithm to compute $\mathbf{\Psi}^\dagger$ via block matrix inversion theoremGolub and Van Loan (2012).

Lemma 5 *(Parallel Inversion of row and column stacked diagonal matrix) Let $J^L = \mathbf{\Psi}$ be partitioned into a block form:*

$$J^L = \begin{bmatrix} J^{L-1} & O \\ R & \mathrm{blk}_L^L(\mathbf{\Psi}) \end{bmatrix}, \quad (28)$$

where $O := \begin{bmatrix} \mathrm{blk}_L^1(\mathbf{\Psi}) \\ \vdots \\ \mathrm{blk}_L^{L-1}(\mathbf{\Psi}) \end{bmatrix}$, and $R := \left[\mathrm{blk}_{L-1}^1(\mathbf{\Psi}), \ldots, \mathrm{blk}_{L-1}^L(\mathbf{\Psi})\right]$. After inverting $\mathrm{blk}_L^L(\mathbf{\Psi})$ which takes $O(1)$ time using $O(n)$ processors, there inverse of $\mathbf{\Psi}$ is achieved by

$$\mathbf{\Psi}^\dagger = \begin{bmatrix} (J^{L-1} - O\mathrm{blk}_L^L(\mathbf{\Psi})^{-1}R)^{-1} & -(J^{L-1})^{-1}O(\mathrm{blk}_L^L(\mathbf{\Psi}) - R(J^{L-1})^{-1}O)^{-1} \\ -\mathrm{blk}_L^L(\mathbf{\Psi})^{-1}R(J^{L-1} - O\mathrm{blk}_L^L(\mathbf{\Psi})^{-1}R)^{-1} & (\mathrm{blk}_L^L(\mathbf{\Psi}) - R(J^{L-1})^{-1}O)^{-1} \end{bmatrix} \quad (29)$$

assuming that J^{L-1} and $\mathrm{blk}_L^L\,\mathbf{\Psi}$ are invertible.

This again requires inverting R, O and J^{L-1}. Recursively applying these block matrix inversion theorem, the inversion problem is reduced to inverting L^2 number of n by n diagonal matrices with additional matrix multiplications as indicated in equation (29).

Inverting a diagonal matrix results in another diagonal one, and the complexity of inverting $n \times n$ diagonal matrix is $O(1)$ with $O(n)$ processors. We can simultaneous invert all blocks. Therefore with $O(nL^2)$ processors, we invert all the diagonal matrices in $O(1)$ time. The recursion takes L steps, for step $i \in [L]$ matrix multiplication cost is O($\log nL$) with $O(n^2L/\log(nL))$ processors. With L iteration, one achieves $O(\log n + \log L)$ running time with $O(n^2L^2/(\log L + \log n))$ processors.