Computationally Efficient Methods for Sparse Tensor Signal Processing

by

Ishan Maduranga Wickramasingha

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg

Abstract

Many modern applications solve multidimensional problems using linear algebra by vectorizing multidimensional signals (Tensor). However, the size of the vectorized signal increases in polynomial order with the number of dimensions of the signal (Order of the Tensor). Therefore solving large multidimensional problems using vectorized signals is computationally infeasible.

This research aimed to develop novel methods that could efficiently solve large multidimensional problems using significantly less computational resources. We studied Sparsity, Tensors, and Multilinear Algebra during this research, and we developed several tensor-based algorithms using multilinear algebra to process large multidimensional signals efficiently.

Sparse signal representations result in simpler and faster processing and lower memory storage requirements. However, obtaining a sparse signal representation of a large multidimensional signal by solving a sparse linear least-squares problem is computationally infeasible. Therefore, in this thesis, we develop the *Tensor Least Angle Regression* (T-LARS) algorithm, a generalization of *Least Angle Regression* (LARS) that could efficiently solve large L_0 or large L_1 constrained sparse multilinear least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ.

Sparse weighted multilinear least-squares is a generalization of the sparse multilinear least-squares problem, where prior information about, e.g., parameters and data is incorporated by multiplying both sides of the original problem by a typically diagonal weights matrix. If the diagonal weight matrix does not have a Kronecker structure similar to the dictionary matrix, we could not use T-LARS to solve this problem efficiently. Therefore, we introduced the *Weighted Tensor Least Angle Regression* (WT-LARS) algorithm to efficiently solve the sparse weighted multilinear least-squares problem for a non-separable weight matrix.

The T-LARS could not be initialized with a solution outside of the Pareto curve because it will violate the optimality conditions of T-LARS. Therefore, we developed the *Tensor Dynamic Least Angle Regression* (TD-LARS) algorithm, a multilinear generalization of the one-dimensional L_1-Homotopy algorithm to efficiently solve multilinear L_1 minimization problems using nonzero initial solutions of close problems located on or off of the Pareto curve.

We also introduced the Multilinear Elastic Net problem by generalizing the one-dimensional Elastic Net problem, which solves a strictly convex L_1 and L_2 constrained multilinear least-squares problem, and it has the best properties of both L_1 and L_2 minimization problems. The dictionary of the Multilinear Elastic Net problem has a partitioned Kronecker structure, which could not be efficiently solved with T-LARS. Therefore, we introduced the *Tensor Elastic Net* (T-NET) algorithm to efficiently solve the Multilinear Elastic Net problem by utilizing the partitioned Kronecker structure of the dictionary matrix.

Learned dictionaries could be used in classification or regression tasks. However, regression and classification performance could be improved significantly by supervised learning of task-specific dictionaries. Therefore, we extended the one-dimensional task-driven dictionary learning (TDDL) to develop the *tensor task-driven dictionary learning* (T-TDDL) that could work as an efficient online data-driven or task-driven dictionary learning algorithm for supervised and semi-supervised learning of *mode-n* dictionaries and *mode-n* model parameters. We also presented a compressed sensing extension to the T-TDDL formulation to efficiently solve large tensor task-driven dictionary learning problems.

Experimental results show the validity and performance of T-LARS, WT-LARS, TD-LARS, and T-NET in obtaining sparse multilinear representations of multidimensional signals and the performance of T-TDDL in multidimensional regression and classification tasks.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my academic advisor Dr. Sherif Sherif for his tremendous support and guidance throughout my Ph.D. research and his patience, motivation, enthusiasm, and immense knowledge. Thank you for your insightful feedback and encouragement that pushed me to grow as a research scientist and take the extra mile.

I would like to take this opportunity to thank my Ph.D. advisory committee members, Dr. Pradeepa Yahampath and Dr. Andrew Goertzen, for their insightful comments, valuable support, and encouragement throughout the Ph.D. research.

I would also like to extend my deepest gratitude to my family members and parents for their love, support, encouragement, and patience throughout my Ph.D. research. I want to thank my wife, Dr. Randima Hettiarachchi, for her continuous support and encouragement throughout the Ph.D. and for being there to discuss my research ideas, and my daughter Tanushi for bearing with me, especially when I had to finalize my Ph.D. research and write the Ph.D. thesis during the COVID-19 pandemic. Sadly, I had to remove the random letters contributed by my 2-year-old daughter from the thesis.

I would not be here without my mother, Kusum Wickramasingha, and my late father, W.A. Sirisoma, my first teachers who believed in me since I was young, and I'm eternally grateful for their unconditional love, support, encouragement, and sacrifices. Therefore, I would like to dedicate this thesis to my family and parents.

I'm grateful to the University of Manitoba for providing me with financial support through the University of Manitoba Graduate Fellowship (UMGF), which allowed me to focus entirely on the Ph.D. research. Finally, I would like to thank my colleagues and friends for their constant support, feedback, and encouragement and the staff of the University of Manitoba for providing a supportive working environment, especially during the COVID-19 pandemic.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

1. Introduction

Many modern applications process multidimensional signals, where each dimension (mode) of the signal has a physical meaning such as space, time, or frequency. The 3D/4D images generated by Magnetic resonance imaging (MRI), Positron emission tomography (PET), or optical coherence tomography (OCT) are such example applications in Biomedical Imaging. Typically, multidimensional signals in such applications are generated by sampling multivariate functions and stored as multidimensional arrays (Tensors).

Processing and storing multidimensional signals quickly become computationally expensive as the number of dimensions (modes) increases. For example, a 3D signal, $\mathcal{X} \in \mathbb{R}^{100\times100\times100}$, has one million ($10^6$) samples and a 4D signal, $\mathcal{Y} \in \mathbb{R}^{100\times100\times100\times100}$ has hundred million (10^8) samples.

The motivation behind this research came from exploring the possibility of solving full-wave simulation of light propagation inside a 5mm×5mm×5mm tissue sample when a light source with $1\mu m$ central wavelength is used, which requires solving the scalar scattering equation for a volume of $5000\lambda \times 5000\lambda \times 5000\lambda$. We could use the Method of Moments (MoM) [1]–[3] to solve the scalar scattering equation by converting it to a linear system of the form $\boldsymbol{Ax} = \boldsymbol{b}$, where the matrix $\boldsymbol{A}$ has 1.25×10^{11} rows and columns, or 1.5625×10^{22} elements. Therefore, to store the uncompressed matrix $\boldsymbol{A}$ in double precision, 125 zettabytes of memory is required, which is more than the total data storage capacity available in the world as of 2021 [4].

This research aimed to develop novel methodologies that could efficiently solve large multidimensional problems using much lower computational resources. To achieve this aim, we researched several topics such as Sparsity, Tensors, and Multilinear Algebra [5]–[7].

Sparse signal representation has gained much interest due to its ability to represent large signals using a few nonzero samples. A sparse signal representation usually results in simpler and faster processing and lower memory storage requirements for fewer coefficients [8], [9]. Obtaining sparse signal representations typically involves solving L_0 or L_1 constrained least-squares problems [10], [11]. Many methods have been proposed to solve the sparse least-squares problem, including *Matching Pursuit* (MP) [12], *Orthogonal Matching Pursuit* (OMP) [13], *Lasso* also known as *Basis Pursuit* (BP) [14], [15], and *Least Angle Regression* (LARS) [15].

Most of the multidimensional problems are currently solved using such methods based on linear algebra by vectorizing multidimensional signals. However, none of the above methods are suitable for obtaining sparse signal representations of large multidimensional signals because they require extensive computational power and memory.

We explored the possibility of using tensors and multilinear algebra to obtain sparse signal representations of multidimensional signals efficiently. Even though the term tensor has a specific mathematical definition in physics, it has been widely accepted in many disciplines, e.g., mathematics, signal processing, and statistics, to mean a multidimensional array. Tensor of order one is a vector; tensor of order two is a matrix; tensors of order three or higher are called higher-order tensors.

A multilinear representation of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_n \times \dots \times I_N}$ could be obtained by multiplying each mode of the tensor by a *mode-n* matrix $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$; $n \in \{1,2,\dots,N\}$ [5]–[7]. The multilinear representation has an equivalent vectorized form, in which the vectorized tensor $\text{vec}(\mathcal{X})$ is multiplied by a separable Kronecker matrix $\boldsymbol{\Phi} = (\boldsymbol{\Phi}^{(N)} \otimes \boldsymbol{\Phi}^{(N-1)} \otimes \dots \otimes \boldsymbol{\Phi}^{(2)} \otimes \boldsymbol{\Phi}^{(1)})$ to obtain the vectorized representation of the tensor. A sparse multilinear representation of a tensor could be obtained by solving a sparse multilinear least-squares problem. See section 2.1 and section 3.2 for more details.

An earlier generalization of OMP, known as *Kronecker-OMP* [16], was developed to solve the L_0 constrained multilinear least-squares problem for large multidimensional signals. However, its memory usage and computation time increase fast with the number of problem dimensions and iterations.

Elrewainy and Sherif earlier developed the *Kronecker Least Angle Regression* (K-LARS) algorithm to solve either large L_0 or large L_1 constrained sparse least-squares problems (overdetermined) efficiently, with a particular Kronecker form $\boldsymbol{A} \otimes \boldsymbol{I}$, for all critical values of the regularization parameter λ [17]. They used K-LARS to sparsely fit one-dimensional multi-channel hyperspectral imaging data to a Kronecker model $\boldsymbol{A} \otimes \boldsymbol{I}$.

By using tensors and multilinear algebra, we develop the *Tensor Least Angle Regression* (T-LARS) algorithm [18] in chapter 3, a generalization of K-LARS that could efficiently solve either large L_0 or large L_1 constrained sparse multilinear least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ, and which has lower computational complexity and lower memory usage than *Kronecker-OMP*.

Computing a sparse signal representation of a large dense signal requires many dense measurements and considerable computational resources. Compressed sensing solves this problem by projecting the dense signal to a sparse domain using a sensing matrix $\boldsymbol{Z}$, where a small number of nonzero measurements are obtained in the sparse domain [19], [20]. Kronecker compressed sensing [16], [21] generalizes the compressed sensing [19], [20] formulation to multidimensional signals, where a sparse representation is obtained using Kronecker dictionaries. Kronecker compressed sensing problems, where the sensing matrix and the dictionary are separable [16], [21], could be efficiently solved using our T-LARS algorithm.

Sparse weighted multilinear least-squares is a generalization of the sparse multilinear least-squares problem, where prior information about, e.g., parameters and data is incorporated by multiplying both sides of the original problem by a typically diagonal weights matrix [22]. We could use T-LARS to solve this problem efficiently if the diagonal weight matrix has a Kronecker structure similar to the dictionary matrix. Typically, these arbitrary diagonal weights matrices are non-Kronecker, leading to a linear least-squares problem that could be very large to store or solve practically. Therefore, we generalized T-LARS to develop the *Weighted Tensor Least Angle Regression* (WT-LARS) algorithm to efficiently solve either L_0 or L_1 constrained weighted sparse multilinear least-squares problems for a non-separable diagonal weights matrix.

Efficiently solving either large L_0 or large L_1 constrained sparse multilinear least-squares problems is essential to obtain sparse multilinear representations of large multidimensional signals. We could initialize T-LARS with an L_1 solution located on the Pareto curve [23] and obtain an L_1 solution with a lower residual error, where the Pareto curve contains every solution to a linear/multilinear least-squares problem. However, we could not initialize T-LARS with any solution outside of the Pareto curve because it will violate the optimality conditions of T-LARS. Asif & Romberg [24] introduced the *L_1-Homotopy* method to dynamically update the solutions of the one-dimensional L_1 minimization problems, using the previous solution as the initial solution for a streaming set of measurements. Therefore, we extend T-LARS and the one-dimensional L_1-Homotopy method to develop the *Tensor Dynamic Least Angle Regression* (TD-LARS) algorithm, which could be used to obtain the solutions to L_1 constrained multilinear least-squares problems efficiently by initializing with non-zero solutions of close L_1 minimization problems located on or off of the Pareto curve.

A sparse signal representation could be obtained by solving an L_0 constrained sparse least-squares problem, which is a nonconvex problem [12], [13]. *Lasso,* also known as *Basis Pursuit* (BP) [14], [25], solves a relaxed L_1 constrained least-squares problem, which is a convex problem, to obtains a sparse signal representation. *Ridge Regression* solves a strictly convex L_2 constrained least-squares problem, nevertheless it could not be used to obtain a sparse signal representation [26]. Zou and Hastie developed the Elastic Net to improve the performance of L_1 constrained least-squares problem by adding an additional L_2 constraint [27], [28]. Elastic Net solves a strictly convex problem, to obtain a sparse solution when both regularization coefficients of L_1 and L_2 are nonzero. Elastic Net selects all the coefficients from a group of highly correlated coefficients, and it could also obtain more than n nonzero coefficients for a n dimensional signal. The one-dimensional Elastic Net problem could be easily solved using the LARS algorithm.

A sparse signal representation of a multidimensional signal with better statistical properties could be obtained by solving a multilinear Elastic Net problem with both L_1 and L_2 constraints. However, the dictionary in the multilinear Elastic Net problem has a partitioned Kronecker structure, which could not be efficiently solved using T-LARS. Therefore, we develop the *Tensor Elastic Net* (T-NET) algorithm to solve the multilinear Elastic Net problem efficiently.

We could use fixed or learned separable dictionaries in obtaining a sparse multilinear representation of multidimensional signals using our T-LARS, WT-LARS, TD-LARS, T-NET, algorithms, or *Kronecker-OMP*. However, the dictionaries learned from the data are much more efficient in obtaining sparse representations than fixed dictionaries [29].

Roemer et al. [30] introduced *Tensor Method of Optimal Directions* (T-MOD) and *Kronecker Higher-Order SVD* (K-HOSVD) algorithms to learn data-driven separable dictionaries to solve multilinear problems by generalizing one-dimensional data-driven dictionary learning algorithms, *Method of Optimal Direction*(MOD) [31], and K-SVD [32], respectively.

Learned dictionaries could be used in classification or regression tasks [33]–[35]. However, regression and classification performance could be improved significantly by supervised learning of task-specific dictionaries [36], [37]. Mairal et al. introduced a generalized *Task-Driven Dictionary Learning*(TDDL) framework for supervised learning of dictionaries and model parameters to solve one-dimensional problems [38].

Many multidimensional classification and regression problems have been solved using the TDDL formulation after vectorizing multidimensional data [39]–[41]. However, using T-DDL formulation for large multidimensional tasks is computationally infeasible. Compared to vectorized tensors, sparse multi-linear representation of tensors requires significantly lower memory and computational resources.

Therefore, we extend the T-DDL framework using multi-linear algebra to develop the *Tensor Task-Driven Dictionary Learning* (T-TDDL), an efficient multi-linear task-driven dictionary learning framework to learn task-specific *mode-n* dictionaries and *mode-n* model parameters jointly for classification or regression tasks. We used our T-NET algorithm developed in this thesis to obtain the sparse multi-linear representations of tensors in the sparse coding step of T-TDDL. We have also developed a compressed sensing extension for T-TDDL.

1.1. Thesis Contributions

1. Development of the *Tensor Least Angle Regression* (T-LARS) - A computationally efficient algorithm to solve both L_0 and L_1 sparse multilinear least-squares problems.
    a. Formulation of the sparse multilinear least-squares problem.
    b. Developed the *Tensor Least Angle Regression* (T-LARS) algorithm by extending Least angle Regression(LARS) [15] to solve both L_0 and L_1 constrained multilinear least-squares problems and implemented it in Matlab. The Matlab implementation of T-LARS is published on Github.

   c. We presented experimental results to compare the performance of *Kronecker-OMP* and T-LARS in obtaining the sparse representation of 3D signals when solving both L_0 and L_1 sparse multilinear least-squares problems.

2. Development of the *Weighted Tensor Least Angle Regression* (WT-LARS) - A computationally efficient algorithm to obtain a sparse signal representation of multidimensional signals using weighted samples.
   a. Formulation of the sparse weighted tensor least-squares problem.
   b. Developed the *Weighted Tensor Least Angle Regression* (WT-LARS) algorithm by extending T-LARS and implemented it in Matlab.
   c. We successfully solved the inpainting problem using WT-LARS to remove foreground objects from color images and presented them in WT-LARS experimental results.
3. Development of the *Tensor Dynamic Least Angle Regression* (TD-LARS) - A computationally efficient algorithm to solve the tensor L_1 minimization problem by using L_1 solution of a close problem as the initial solution.
   a. Formulation of the *Tensor Dynamic Least Angle Regression* (TD-LARS) formulation by extending the vector-based L_1-Homotopy formulation [24], [42] .
   b. Developed the *Tensor Dynamic Least Angle Regression* (TD-LARS) algorithm by extending T-LARS and L_1-Homotopy algorithm and implemented it in Matlab.
   c. We presented experimental results to compare the performance of T-LARS and TD-LARS in obtaining the sparse representation of 3D signals when the L_1 solution of a close problem is available.
4. Development of the *Tensor Elastic Net* (T-NET) - A computationally efficient algorithm to solve the multilinear Elastic Net problem.
   a. Formulated the multilinear Elastic Net problem by extending one-dimensional Elastic Net [28].
   b. Developed the *Tensor Elastic Net* (T-NET) algorithm by extending T-LARS and implemented it in Matlab.
   c. We presented experimental results to compare T-LARS and the performance of T-NET in obtaining the sparse representation of 3D signals using overcomplete DCT dictionaries with different mutual coherences.

5. Development of the *Tensor Task-Driven Dictionary Learning* (T-TDDL) - A computationally efficient task-driven dictionary learning algorithm to learn *mode-n* dictionaries and *mode-n* model parameters to predict a tensor $\mathcal{Y}$ from a tensor $\mathcal{X}$.
    a. Developed the *Tensor Task-Driven Dictionary Learning* (T-TDDL) formulation by extending the one-dimensional *Task-Driven Dictionary Learning* formulation[38].
    b. Developed the *Tensor Task-Driven Dictionary Learning (T-TDDL)* algorithm and implemented it in Matlab.
    c. Multilinear generalization of the projected stochastic gradient descent algorithm to optimize Kronecker matrices efficiently while keeping the Kronecker structure intact.
    d. Developed the compressed sensing extension to the T-TDDL algorithm.
    e. Presented calculations for T-TDDL regression, binary classification, and multiclass classification applications.
    f. We presented experimental results for solving the tensor regression and tensor binary classification problems by learning *mode-n* dictionaries, *mode-n* model parameters, and *mode-n* sensing matrices using T-TDDL.

1.1.1. Related publications

1. **I. Wickramasingha**, M. Sobhy, A. Elrewainy, and S. S. Sherif, "Tensor least angle regression for sparse representations of multidimensional signals," *Neural Comput.*, vol. 32, no. 9, pp. 1697–1732, Sep. 2020,https://doi.org/10.1162/neco_a_01304.

2. **I. Wickramasingha**, M. Sobhy, and S. S. Sherif, "Sparsity in Bayesian Signal Estimation," in *Bayesian Inference*, vol. 37, no. 2, J. P. Tejedor, Ed. InTech, 2017, https://doi.org/10.5772/intechopen.70529.

3. **I. Wickramasingha** and S. S. Sherif, "Multilinear Compressed Sensing using Tensor Least Angle Regression (T-LARS)," Accepted to the International Conference on Digital Signal Processing, Chengdu, China, Feb. 2022.

4. **I. Wickramasingha**, and S. S. Sherif, (2021). "Weighted Tensor Least Angle Regression for solving weighted sparse multilinear least-squares problems" (submitted to *IEEE Signal Processing Letters*, 2021).

5. **I. Wickramasingha**, and S. S. Sherif, “Tensor Elastic Net for Sparse Multilinear Regression With Elastic Net Constraint,” (submitted to *Journal of Statistical Software*, 2021).

1.1.2. Publications Plan

1. **I. Wickramasingha**, and S. S. Sherif, “Tensor Dynamic Least Angle Regression for Efficiently Solving Tensor L_1 Minimization Problems with Non-zero Initial Solutions,” (in preparation for submission to *IEEE Transactions on Circuits and Systems for Video Technology*).

2. **I. Wickramasingha**, and S. S. Sherif, “Tensor Task-Driven Dictionary Learning,” (in preparation for submission to *Pattern Recognition*).

3. B. Mezgebo, **I. Wickramasingha**, B. Kordi, and S. S. Sherif, “Gradient-Based Multidimensional Signal Recovery from Incomplete Samples in Arbitrary Separable Dictionaries,” (in preparation for submission to *IEEE Transactions on Image Processing*).

1.2. Thesis Outline

This thesis is structured as follows.

Chapter 2 discusses background theory on Tensors, Multilinear Algebra, the one-dimensional Sparse Signal Representation problem, and the Sparse tensor signal representation problem.

Chapter 3 presents our *Tensor Least Angle Regression* (T-LARS) algorithm, a computationally efficient algorithm to solve both L_0 and L_1 sparse multilinear least-squares problems.

Chapter 4 presents our *Weighted Tensor Least Angle Regression* (WT-LARS) algorithm, which could be used to obtain a sparse signal representation of multidimensional signals using weighted samples.

Chapter 5 presents our *Tensor* Dynamic *Least Angle Regression* (TD-LARS) algorithm, which could be used to obtain the L_1 solution of a multilinear sparse least-squares problem efficiently by using the L_1 solution of a close problem.

Chapter 6 presents our *Tensor Elastic Net* (T-NET) algorithm, a computationally efficient algorithm to solve the multilinear elastic net problem.

Chapter 7 presents our *Tensor Task-Driven Dictionary Learning* (T-TDDL), a computationally efficient task-driven dictionary learning framework to learn *mode-n* dictionaries and *mode-n* model parameters to predict a tensor $\mathcal{Y}$ from a tensor $\mathcal{X}$.

Finally, Chapter 8 provides the conclusions and future directions.

Chapter 2

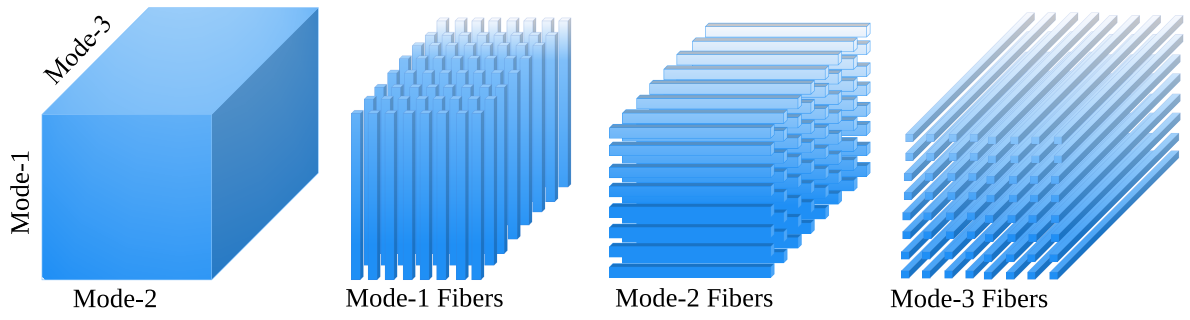
2. Tensors and Sparse Signal Representation

2.1. Tensors and Multilinear Algebra

The term tensor has a specific mathematical definition in physics, but it has been widely accepted in many disciplines, e.g., signal processing and statistics, to mean a multidimensional array (Multi-way Arrays, ND Array). A vector is a first-order tensor; a matrix is a second-order tensor; an N-dimensional array is an N^{th} order tensor, whose N dimensions are also known as modes [6], [43]. The N^{th} order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ has N modes, with dimensions, I_1, $I_2, \ldots,$ I_N, where vectors along a specific mode, n, are called *mode-n* fibers. For example, for the 3rd order tensor shown in Figure 2.1, vectors along *mode-1* are called *mode-1* fibers, and vectors along *mode-2* and *mode-3* are called *mode-2* and *mode-3* fibers, respectively.

Vectorization and *mode-n* Matricization of Tensors [5]–[7] are two important tensor reshaping operations. As the names imply, vectorization generates a vector, and matricization generates a matrix.

Figure 2.1. A 3rd-order tensor and *mode-n* fibers



2.1.1. Vectorization of a tensor

Tensors are vectorized by stacking *mode-1* fibers in reverse lexicographical order, where this vectorization is denoted by $\mathrm{vec}(\mathcal{X})$.

$$\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N} \longrightarrow \mathrm{vec}(\mathcal{X}) \in \mathbb{R}^{I_\mathrm{N} I_{N-1} \ldots I_1}$$

In $\mathrm{vec}(\mathcal{X})$, I_1 varies the fastest and I_N varies the slowest. The vector index l corresponding to the tensor element $x_{i_1 \ldots i_n \ldots i_N} \in \mathcal{X}$ is given by,

$$l = i_1 + \sum_{p=2}^{\mathrm{N}} (i_p - 1) I_1 I_2 \ldots . I_{p-1} \tag{2.1}$$

Proposition 2.1 shows how to obtain tensor indices $\{i_1 \ldots i_N\}$, corresponds to the vector index l.

Proposition 2.1*: Let l be the vector index of an element v_l in $vec(\mathcal{X}) \in \mathbb{R}^{I_N I_{N-1} \ldots I_1}$ and I_n be the dimension of the mode-n of the tensor $\mathcal{X}$ where, $n \in \{1,2,\ldots,N\}$. The tensor indices $i_n; n \in \{1,2,\ldots,N\}$, corresponding to the vector element v_l, could be obtained by,*

$$i_n = \left\lceil \frac{l}{I_1 \times \ldots \times I_{n-1}} - \sum_{p=n+1; p \le N}^{N} (i_p - 1) \prod_{q=n; q>0}^{p-1} I_q \right\rceil \tag{2.2}$$

where $\lceil * \rceil$ indicate the ceiling function. For example,

$$i_1 = \lceil l - (i_N - 1) I_{N-1} \times \ldots . \times I_1 - \cdots - (i_2 - 1) I_1 \rceil$$

$$\vdots$$

$$i_{N-1} = \left\lceil \frac{l}{I_1 \times \ldots \times I_{N-2}} - (i_N - 1) I_{N-1} \right\rceil$$

$$i_N = \left\lceil \frac{l}{I_1 \times \ldots \times I_{N-1}} \right\rceil$$

The proof is in Appendix A.1.

2.1.2. *Mode-n* Matricization of a tensor

A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ has N different modes $(1,2, \ldots, N)$. Consider splitting these modes into two different disjoint sets: $\{1, \ldots, N\} = P \cup Q$ with $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_{N-k}\}$. Tensor $\mathcal{X}$ could be matricized by merging the group P into row indices and the group Q into column indices [44]. The *mode-n* tensor matricization is a special case of tensor matricization, where the *mode-n* indices i_n are the row indices, and all other tensor modes are merged to get column indices. Therefore, in tensor *mode-n* matricization, *mode-n* fibers become the columns of the resulting matrix. We note that ordering of these columns is not consistent across the literature [5], [45]. Therefore, we use the reverse lexicographical order $(I_N \ldots I_{n+1} I_{n-1} \ldots I_1)$, for the column ordering of the tensor *mode-n* matricization in this thesis. In the reverse lexicographical order, I_1 varies the fastest and I_N varies the slowest. Let $\boldsymbol{X}_{(n)}$ denote *mode-n* matricization of a tensor $\mathcal{X}$.

$$\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N} \longrightarrow \boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times (I_N \ldots I_{n+1} I_{n-1} \ldots I_1)}$$

Let ${x^{(n)}}_{i_n j} \in \boldsymbol{X}_{(n)}$ be the tensor element $x_{i_1 \ldots i_n \ldots i_N} \in \mathcal{X}$, in the *mode-n* matrix. Therefore, the column index j is given by [5], [6], [45],

$$j = 1 + \sum_{p=1; p \neq n}^{\mathrm{N}} (i_p - 1) I_1 \ldots I_{n-1} I_{n+1} \ldots I_{p-1} \tag{2.3}$$

2.1.3. Tensor *Mode-n* Product

Another important operation that could be performed on a tensor is its *mode-n* product, i.e., its multiplication by a matrix along one of its modes. Let the tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ and a matrix $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_\mathrm{n} \times I_n}$, then their *mode-n* product, $\mathcal{Y}_n \in \mathbb{R}^{I_1 \times \ldots \times J_\mathrm{n} \times \ldots \times I_N}$, is denoted as

$$\mathcal{Y}_n = \mathcal{X} \times_n \boldsymbol{\Phi}^{(n)} \tag{2.4}$$

In the *mode-n* product, all *mode-n* fibers of the tensor are multiplied by the matrix $\boldsymbol{\Phi}^{(n)}$. Therefore (2.4) could be written as

$$\boldsymbol{Y}_{(n)} = \boldsymbol{\Phi}^{(n)} \boldsymbol{X}_{(n)} \tag{2.5}$$

where $\boldsymbol{Y}_{(n)}$ is the *mode-n* matricization of the tensor $\mathcal{Y}_n$ and $\boldsymbol{X}_{(n)}$ is the *mode-n* matricization of the tensor $\mathcal{X}$. A *mode-n* product could be thought of as a linear transformation of the *mode-n* fibers of a tensor.

2.1.4. Other Important Products

This section presents some important products used throughout the thesis; Tensor outer product, Kronecker Product, Khatri-Rao product, and Hadamard Product [5]–[7].

2.1.4.1. Tensor Outer Product - $\mathcal{A} \circ \mathcal{B}$

Let $\mathcal{A} \in \mathbb{R}^{I_1 \times \dots \times I_M}$ be a mode M tensor and $\mathcal{B} \in \mathbb{R}^{J_1 \times \dots \times J_N}$ be a mode N tensor. Therefore the tensor outer product $\mathcal{C} \in \mathbb{R}^{I_1 \times \dots \times I_M \times J_1 \times \dots \times J_N}$ is a mode $(M+N)$ tensor.

$$\mathcal{A} \circ \mathcal{B} = \mathcal{C} \in \mathbb{R}^{I_1 \times \dots \times I_M \times J_1 \times \dots \times J_N}$$

with

$$c_{i_1 \dots i_M j_1 \dots j_N} = a_{i_1 \dots i_M} b_{j_1 \dots j_N}$$

Where $a_{i_1 \dots i_M} \in \mathcal{A}$, $b_{j_1 \dots j_N} \in \mathcal{B}$ and $c_{i_1 \dots i_M j_1 \dots j_N} \in \mathcal{C}$.

2.1.4.2. Kronecker Product - $\boldsymbol{A} \otimes \boldsymbol{B}$

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be matrices. Therefore the Kronecker product denoted by $\boldsymbol{A} \otimes \boldsymbol{B}$ is given by

$$\boldsymbol{A}_{n_1 \times m_1} \otimes \boldsymbol{B}_{n_2 \times m_2} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \dots & a_{1m_1}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \dots & a_{2m_1}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_1 1}\boldsymbol{B} & a_{n_1 2}\boldsymbol{B} & \dots & a_{n_1 m_1}\boldsymbol{B} \end{bmatrix}_{n_1 n_2 \times m_1 m_2}$$

where a_{ij} are elements of the matrix $\boldsymbol{A}$.

2.1.4.3. Khatri-Rao Product - $\boldsymbol{A} \odot \boldsymbol{B}$

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be matrices with the same number of columns. Therefore the Khatri-Rao product denoted by $\boldsymbol{A} \odot \boldsymbol{B}$ is defined as,

$$\boldsymbol{A}_{n_1 \times m} \odot \boldsymbol{B}_{n_2 \times m} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \quad \boldsymbol{a}_2 \otimes \boldsymbol{b}_2 \quad \dots \quad \boldsymbol{a}_m \otimes \boldsymbol{b}_m]_{n_1 n_2 \times m}$$

where $\boldsymbol{a}_1, \dots, \boldsymbol{a}_m$ are the column vectors of $\boldsymbol{A}$ and $\boldsymbol{b}_1, \dots, \boldsymbol{b}_m$ are the column vectors of $\boldsymbol{B}$.

2.1.4.4. Hadamard Product - $\boldsymbol{A} \circledast \boldsymbol{B}$

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be matrices with the same number of rows and columns. Therefore the Hadamard product denoted by $\boldsymbol{A} \circledast \boldsymbol{B}$ is defined as,

$$\boldsymbol{A}_{n\times m} \circledast \boldsymbol{B}_{n\times m} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1m}b_{1m} \\ a_{21}b_{21} & a_{22}b_{22} & \dots & a_{2m}b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \dots & a_{nm}b_{nm} \end{bmatrix}_{n\times m}$$

where a_{ij} are elements of the matrix $\boldsymbol{A}$ and b_{ij} are elements of the matrix $\boldsymbol{B}$.

2.1.5. Multilinear Transformation of Tensors

The *mode-n* product could be thought of as a linear transformation of the *mode-n* fibers of a tensor. Therefore the multilinear transformation of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$ could be defined as

$$\mathcal{Y} = \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} \qquad (2.6)$$

where, $\boldsymbol{\Phi}^{(n)}$; $n \in \{1,2,\ldots,N\}$ are matrices with dimensions $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n\times I_n}$; $n \in \{1,2,\ldots,N\}$ and $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_n\times\ldots\times J_N}$ [6], [43]. In the multilinear transformation, each mode $n \in \{1,2,\ldots,N\}$ of the tensor is transformed by the respective *mode-n* matrix $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n\times I_n}$; $n \in \{1,2,\ldots,N\}$ to get the transformed tensor $\mathcal{Y}$.

In mathematics, the equation (2.6) is widely known as the Tucker decomposition [5], where a tensor $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_n\times\ldots\times J_N}$ is decomposed to obtain a core tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$ multiplied by a set of factor matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n\times I_n}$; $n \in \{1,2,\ldots,N\}$ along each mode.

Figure 2.2 Multilinear transformation of a 3rd-order tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times I_3}$

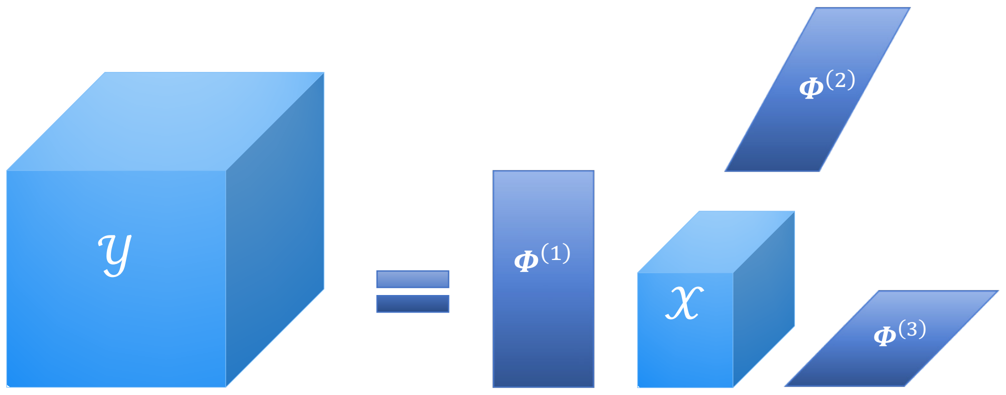

$\mathcal{Y}$
=
$\boldsymbol{\Phi}^{(1)}$
$\mathcal{X}$
$\boldsymbol{\Phi}^{(2)}$
$\boldsymbol{\Phi}^{(3)}$

Figure 2.2 shows the multilinear transformation $\mathcal{Y} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ of a 3rd-order core tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ obtained by multiplying each mode of the core tensor $\mathcal{X}$ with a *mode-n* matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_\mathrm{n} \times I_n}$; $n \in \{1,2,3\}$. Therefore as shown in Figure 2.2, a large tensor $\mathcal{Y}$ could be represented using a smaller core tensor and a smaller set of factor matrices $\boldsymbol{\Phi}^{(n)}$.

The multilinear transformation could also be written as a product of $\mathrm{vec}(\mathcal{X})$ and the Kronecker product of matrices $\boldsymbol{\Phi}^{(n)}$; $n \in \{1,2,\ldots,N\}$ [6], [43].

$$\mathrm{vec}(\mathcal{Y}) = (\boldsymbol{\Phi}^{(N)} \otimes \boldsymbol{\Phi}^{(N-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)})\mathrm{vec}(\mathcal{X}) \tag{2.7}$$

Let

$$\boldsymbol{\Phi} = (\boldsymbol{\Phi}^{(N)} \otimes \boldsymbol{\Phi}^{(N-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(2)} \otimes \boldsymbol{\Phi}^{(1)}) \tag{2.8}$$

Therefore,

$$\mathrm{vec}(\mathcal{Y}) = \boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) \tag{2.9}$$

Equation (2.9) is a linear system of equations with a separable dictionary $\boldsymbol{\Phi}$, input $\mathrm{vec}(\mathcal{X})$, and output $\mathrm{vec}(\mathcal{Y})$.

2.2. Sparse Signal Representation

A sinusoid is a dense signal in the time domain, but it could be uniquely represented with its frequency and phase. Therefore a sinusoid signal has a sparse signal representation in the frequency domain.

Consider a linear system, where a finite-dimensional signal $\boldsymbol{y}$ is represented using a dictionary $\boldsymbol{\Phi}$, and a coefficient vector $\boldsymbol{x}$.

$$\boldsymbol{y} = \boldsymbol{\Phi x} + \boldsymbol{e} \tag{2.10}$$

where $\boldsymbol{e}$ is the error term.

In sparse signal representations, a signal $\boldsymbol{y}$ is represented in a sparse domain using a dictionary $\boldsymbol{\Phi}$ where most of its coefficients $\boldsymbol{x}$ are zero. Sparse signal representations require less memory storage, simpler and faster processing, and fewer computational resources than their dense counterparts.

The dictionary $\boldsymbol{\Phi}$ could be a basis, frame, or a tight frame(union of basis) [8], [9]. Common dictionaries include Fourier dictionary, Discrete Cosine Transform (DCT) dictionary, Wavelet dictionaries, Chirplet dictionaries, or a union of any of the above dictionaries. The dictionaries could also be learned from the data using dictionary learning algorithms [31], [32][32].

The least-squares problem provides a mathematical framework for solving the inverse problem of (2.10) to obtain coefficients $\boldsymbol{x}$. Let the vectors $\boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{x} \in \mathbb{R}^n$ and the dictionary $\boldsymbol{\Phi} \in \mathbb{R}^{m\times n}$ in (2.10), where m and n are the row rank and column rank of the dictionary $\boldsymbol{\Phi}$ respectively. The least-squares method minimizes the L_2 norm of the residual vector $\boldsymbol{r} = \boldsymbol{\Phi}\,\boldsymbol{x} - \boldsymbol{y}$ [46]–[48].

$$\widehat{\boldsymbol{x}} = \arg\min_{x} \|\boldsymbol{\Phi}\,\boldsymbol{x} - \boldsymbol{y}\,\|_2^2 \tag{2.11}$$

An inverse problem is well-posed if it satisfies the three Hadamard conditions; Existence, Uniqueness, and Continuity [9], [49], [50]. For example, if the dictionary $\boldsymbol{\Phi} \in \mathbb{R}^{m\times n}$ is a square matrix, where $m = n,$ has a full rank and well-conditioned, the inverse problem of (2.10) is well posed [51]. Therefore, for a well-posed problem, $\boldsymbol{y} \in \mathcal{R}(\boldsymbol{\Phi})$, the solution of the least-squares problem is also the solution of the linear system in (2.10).

The inverse problems for overdetermined and underdetermined linear systems are ill-posed. Because the overdetermined linear systems, where $m > n$, violate the existence condition, and the underdetermined linear systems, where $m < n$, violate the uniqueness condition [9], [50]–[54].

However, the least-squares methods could obtain a desirable approximate solution to ill-posed problems. Generally, overdetermined problems could be solved with the least-squares methods without additional constraints. However, additional constraints are necessary to find a unique solution to underdetermined problems [52].

2.2.1. Sparse Least-Squares Problems

The sparsity could be imposed by adding a sparsity constraint to the least-squares problem. We could formulate the sparse least-squares problem as an L_p minimization problem [9],

$$\widehat{\boldsymbol{x}} = \arg\min_{x} \|\boldsymbol{\Phi}\, \boldsymbol{x} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{x}\|_p \tag{2.12}$$

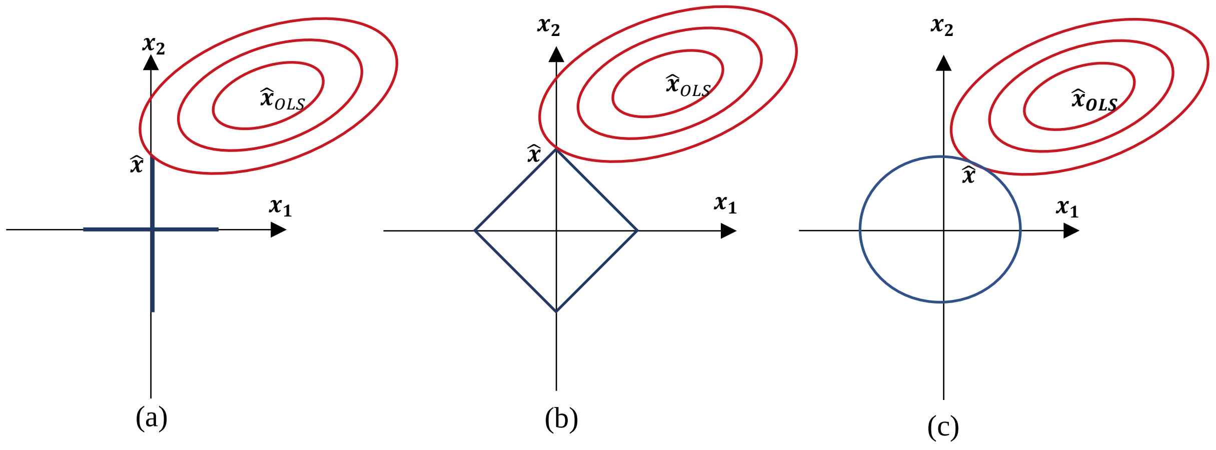
where the *p-norm* of a vector is defined as,

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{\frac{1}{p}} \tag{2.13}$$

Norms with $p < 1$ are called pseudo-norms since they do not satisfy the triangular inequality. We note that in (2.12), it is a nonconvex optimization problem for $0 \leq p < 1$ and a convex optimization problem for $p \geq 1$. The sparsest solutions are given when $p = 0$, and the sparsity reduces as p increases.

Let us consider a 2-dimensional sparse least-squares problem. Figure 2.3 a) shows that the contours of the least-squares error(in red) and the unit ball for the L_0 constraint(in blue) always intersect at an axis to obtain a sparse solution $\widehat{\boldsymbol{x}}$ (e.g. $x_1 = 0$ at $\widehat{\boldsymbol{x}}$), where $\widehat{\boldsymbol{x}}_{OLS}$ is the solution for the unconstrained least-squares problem.

Figure 2.3. Contours of the least-squares error and the unit ball for a) L_0 norm and b) L_1 norm c) L_2 norm



The L_0 constrained least-squares problem is a non-convex optimization problem that can be solved using hard thresholding if $\boldsymbol{\Phi}$ is orthogonal [8]. Matching Pursuit (MP) [12], Orthogonal Matching Pursuit (OMP) [13], Group OMP [55], and Least Angle Regression(LARS) [15] are some other algorithms for solving the L_0 constrained least-squares problem.

We could slightly relax the sparsity constraint to obtain the convex L_1 constrained least-squares problem. Figure 2.3 b) shows the contours of the least-squares error and the unit ball of L_1 norm has a high chance of intersecting at an axis to obtain a sparse solution $\hat{\boldsymbol{x}}$. If $\boldsymbol{\Phi}$ is orthogonal, the L_1 constrained least-squares problem could be solved using soft thresholding [8]. Basis Pursuit(BP) [14], Iterative Thresholding [56], Lasso [25], LARS [15], and Grouped LARS/Lasso [57] are a few other algorithms to solve the L_1 constrained least-squares problem.

The L_2 constrained least-squares problem, also known as ridge regression [26], solves a strictly convex problem. However, as shown in Figure 2.3 c), the contours of the least-squares error and the unit ball of L_2 norm has a less chance of intersecting at an axis. Therefore, the L_2 constrained least-squares problem, could not be used to obtain a sparse signal representation.

2.2.2. Obtaining Optimum Sparse Signal Representations

An optimum sparse signal representation should be able to represent a given signal using a few elementary atoms from a dictionary. However, it is impossible to find an ideal dictionary to obtain optimum sparse signal representations for all signals [8].

Orthogonal bases ($\boldsymbol{\Phi} \in \mathbb{R}^{m \times n} |\, m = n$), a dictionary of minimum size, could be designed to obtain sparse signal representations of signals efficiently by hard thresholding, which solves an L_0 minimization problem in (2.12), or by soft thresholding, which solves an L_1 minimization problem in (2.12).

However, a sparse signal representation obtained over a single basis or a smaller frame is not always optimum for an arbitrary signal [8], [53], [58]. For example, if a signal has time localized components, the Fourier basis fails to obtain an optimum sparse signal representation. Whereas, if the Fourier transform of the signal has components with narrow high-frequency support, wavelet bases fail to obtain an optimum sparse signal representation [12]. Therefore, a union of both Fourier and a wavelet basis could obtain a better sparse signal representation for a signal with both time localized and localized frequency components than either basis alone.

Therefore, richer overcomplete dictionaries ($\boldsymbol{\Phi} \in \mathbb{R}^{m \times n} |\, n > m$) could be used to obtain optimum sparse signal representations of complicated signals [8], [53]. Typically, sparse signal representations are obtained by solving an L_p constrained least-squares problem in (2.12) using OMP, BP, or LARS. However, the computational requirement for solving the sparse signal representation problem increases significantly with the size of the dictionary. Therefore, overcomplete dictionaries are typically designed as a union of a few orthogonal bases(tight frames) or frames.

Fixed dictionaries are designed for sparse representation of certain regularities of signals. The Fourier transform promotes a sparse representation of uniformly regular functions, and Discrete Cosine Transform (DCT) could be used to obtain real-valued representations of such signals, with applications in signal compression. Wavelet bases promote the sparse representation of piecewise continuous signals, including transients and singularities [8]. Therefore, wavelets could be used to represent edges in images efficiently. Wavelet packets [59], steerable wavelets [60], curvelets [61], contourlets [62], and bandelets [63] could be used to represent specific types of edges in images [8], [53]. Typically dictionaries learned for a specific type of signal obtain better sparse signal representations than fixed dictionaries [53].

2.3. Sparse Tensor Signal Representation

The size of tensors quickly grows with the number of modes and dimensions along each mode. Therefore, obtaining sparse tensor signal representations enables solving large tensor problems because sparse tensors have fewer non-zero coefficients and require significantly lower computer power and memory than their dense counterparts.

As shown in (2.6), a finite dimensional tensor signal $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_\mathrm{n}\times\ldots\times J_\mathrm{N}}$ could be represented using a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$ and a set of *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_\mathrm{n}\times I_n}$; $n \in \{1,2,\ldots,N\}$. Each *mode-n* dictionary matrix $\boldsymbol{\Phi}^{(n)}$ could be a basis, frame, or a tight frame. The *mode-n* dictionary matrices could be selected independently to obtain an optimum representation of each tensor mode.

For example, consider 30 frames of an RGB video with the resolution 640×480. The corresponding tensor $\mathcal{Y} \in \mathbb{R}^{640\times480\times3\times30}$, could be represented using four *mode-n* dictionary matrices $\{\boldsymbol{\Phi}^{(1)},\cdots,\boldsymbol{\Phi}^{(4)}\}$. Therefore, we could select $\boldsymbol{\Phi}^{(1)}$, and $\boldsymbol{\Phi}^{(2)}$ to be a union of DCT dictionary and a wavelet dictionary to represent each video frame, $\boldsymbol{\Phi}^{(3)}$ to be an identity matrix to represent three RGB channels, and $\boldsymbol{\Phi}^{(4)}$ to be a wavelet dictionary to obtain a time-frequency representation of temporal variations in video frames.

Obtaining a sparse signal representation of a large tensor is a computationally challenging problem.

2.3.1. Sparse Multilinear Least-squares Problem

The equation (2.7) is an equivalent vector formulation of (2.6). Therefore, a sparse tensor $\mathcal{X}$ could be obtained by rewriting (2.7) as an L_p minimization problem [9],

$$\widetilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) - \mathrm{vec}(\mathcal{Y})\|_2^2 + \lambda\|\mathrm{vec}(\mathcal{X})\|_p \tag{2.14}$$

where $\boldsymbol{\Phi} = \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}\right)$, λ is a regularization parameter.

Equation (2.14) is an L_p constrained linear least-squares problem, where for a large tensors $\mathcal{X}$, and $\mathcal{Y}$, solving the L_p minimization problem by constructing the Kronecker dictionary $\boldsymbol{\Phi}$, might be computationally infeasible.

Therefore, we could reformulate (2.14) as a multilinear least-squares problem using (2.6) and (2.7),

$$\tilde{\mathcal{X}} = \arg\min_{\mathcal{X}} \left\| \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} - \mathcal{Y} \right\|_2^2 + \lambda \|\mathcal{X}\|_p \qquad (2.15)$$

where $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ and $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$.

The L_p constrained least-square problems in (2.14) and (2.15) are equivalent. However, the results of (2.14) are in vector form, and the results of (2.15) are in tensor form.

2.3.2. Thesis Problem Statement

A large tensor problem could be solved efficiently by using a sparse tensor signal representation, which is typically obtained by solving an L_p constrained linear least-squares problem in (2.14). However, obtaining a sparse signal representation of a tensor $\mathcal{Y}$, by solving (2.14) requires constructing and inverting a significantly large Kronecker dictionary matrix $\boldsymbol{\Phi}$.

For example, a third-order tensor $\mathcal{Y} \in \mathbb{R}^{100 \times 100 \times 100}$, would require constructing a Kronecker dictionary, $\boldsymbol{\Phi} \in \mathbb{R}^{10^6 \times 10^6}$, with at least 1 trillion (10^{12}) elements, in solving (2.14) using method discussed in section 2.2.1, and a fourth-order tensor $\mathcal{Y} \in \mathbb{R}^{100 \times 100 \times 100 \times 100}$ would require constructing a Kronecker dictionary $\boldsymbol{\Phi} \in \mathbb{R}^{10^8 \times 10^8}$, with at least 10 quadrillion (10^{16}) elements.

Therefore, solving (2.14) for a tensor problem using OMP, BP, LARS, or any other one-dimensional method discussed in section 2.2.1 quickly become computationally intractable as the number of modes and the dimensions of each tensor mode increases.

The main objective of this research is to develop novel methods that could efficiently obtain sparse signal representations of tensors to solve large multidimensional problems efficiently. The novel methods developed in this research is primarily based on the relationship between (2.14) and (2.15).

This thesis presents four novel methods, developed in chapters 3, 4, 5, and 6, by extending their one-dimensional counterparts using tensors and multilinear algebra. These methods obtain sparse signal representations of large tensors efficiently by solving variations of (2.15) without explicitly constructing or inverting large matrices such as the Kronecker dictionary matrix $\boldsymbol{\Phi}$. Instead our methods use much smaller *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}$ in their calculations.

These four methods use fixed or previously learned *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$ to obtain a sparse signal representation $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ of a large tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$. Our fifth method developed in chapter 7, which is also an extension of its one-dimensional counterpart using tensors and multilinear algebra, could be used for online learning of *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$ in (2.15) using data tensors $\mathcal{Y}$.

Also, the fifth method is a tensor task-driven dictionary learning framework that uses the sparse signal representations of tensors obtained using our novel methods to efficiently solve large multidimensional supervised or semi-supervised machine learning problems such as tensor regression or tensor classifications. This method predicts a tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ from a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$, when $\mathcal{X}$ is associated with $\mathcal{Y}$, by jointly learning *mode-n* dictionaries and *mode-n* model parameters online.

Chapter 3

3. Tensor Least Angle Regression (T-LARS)

Sparse signal representations have gained much interest recently in both signal processing and statistical communities. Compared to *Orthogonal Matching Pursuit* (OMP) [13], and *Basis Pursuit* (BP) [25], [27], that solve the L_0 and L_1 constrained sparse least-squares problems respectively, for a specific value of the regularization parameter λ, *Least Angle Regression* (LARS) [15] is a computationally efficient method to solve both problems for all critical values of λ. However, these methods are not suitable for solving large multidimensional sparse least-squares problems, as they would require extensive computational power and memory. An earlier generalization of OMP, known as *Kronecker-OMP* [16], was developed to solve the L_0 problem for large multidimensional sparse least-squares problems. However, its memory usage and computation time increase fast with the number of problem dimensions and iterations. In this chapter, we develop a generalization of LARS, *Tensor Least Angle Regression* (T-LARS) [18] that could efficiently solve either large L_0 or large L_1 constrained multidimensional sparse least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ, and which has lower computational complexity and lower memory usage than *Kronecker-OMP*. To demonstrate the validity and performance of our T-LARS algorithm, we used it to successfully obtain different sparse representations of two relatively large 3-D brain images, using fixed and learned separable over-complete dictionaries, by solving both L_0 and L_1 constrained sparse least-squares problems and compared with *Kronecker-OMP*. We also present the multilinear compressed sensing problem, and we compared *Kronecker-OMP* and T-LARS in reconstructing 3D brain images using compressed sensed samples.

3.1. Introduction

Sparse signal representations have gained much interest recently in both Signal Processing and Statistics communities. A sparse signal representation usually results in simpler and faster processing, in addition to lower memory storage requirements for fewer coefficients [8], [9]. However, finding optimal sparse representations for different signals is not a trivial task [8]. Therefore, redundant signal representations using overcomplete dictionaries have been introduced to facilitate finding more sparse representations for different signals [8], [9], [12], [13]. Under complete dictionaries could also be used to obtain approximate signal representations [64]–[66]. We note that at their core, such signal representation problems typically involve solving a least-squares problem [10], [11].

A number of methods have been proposed to solve the sparse least-squares problem, including the *Method of Frames* (MOF) [67], *Matching Pursuit* (MP) [12], *Orthogonal Matching Pursuit* (OMP) [13], *Best Orthogonal Basis* (BOB) [68], *Lasso* also known as *Basis Pursuit* (BP) [25], [27], and *Least Angle Regression* (LARS) [15]. *Matching Pursuit* (MP) and *Orthogonal Matching Pursuit* (OMP) obtain sparse signal representations by solving a non-convex L_0 constrained least-squares problem [69]. Matching Pursuits are heuristic methods that construct sparse signal representations by sequentially adding atoms from a given dictionary in a *greedy*, i.e., non-globally optimal manner. *Basis Pursuit* (BP) relaxes the non-convex L_0 constrained optimization problem to solve a convex L_1 constrained least-squares problem instead [14]. In both problem formulations, a regularization parameter λ determines the trade-off between the representation error of the signal and its sparsity, as shown in the Pareto curve in [23]. A common approach to obtaining a sparse signal representation using *Basis Pursuit* is to solve the optimization problem multiple times, for different values of λ, before choosing the most suitable solution for the application at hand [23].

Compared to the above methods, *Least Angle Regression* efficiently solves the L_0, or with a slight modification, the L_1 constrained least-squares problem for all critical values of the regularization parameter λ [15]. However, even LARS is not suitable for large-scale problems as it would require multiplication and inversion of very large matrices [70]. For example, for m unknown variables and n equations, the LARS algorithm has $\mathrm{O}(m^3 + \mathrm{n}m^2)$ computational complexity [15].

LARS and other algorithms to solve sparse least-squares problems are directly applicable to one-dimensional signals. Therefore, multidimensional signals that are represented by tensors, i.e., multidimensional arrays, would need to be vectorized first to enable the application of these methods [7], [45], [71]. For I^N number of vectorized variables, a dictionary of the size $I^N \times J^N$, where $J > I$ for an overcomplete dictionary, is required to solve the sparse linear least-squares problem, using LARS and other algorithms mentioned above. N is the order of the tensor, also known as the number of modes, and I is the dimension of each mode. Therefore, the number of vectorized unknown variables I^N would increase exponentially with the order of the tensor N. Thus, such problems would quickly become increasingly large and computationally intractable. For example, a 3D tensor with 100 unknown variables in each mode has a total of 1 million unknowns, which requires a dictionary with at least 1 trillion (10^{12}) elements, whereas a 4D tensor with 100 unknown variables in each mode has a total of 100 million unknowns, which requires a dictionary of at least ten quadrillion (10^{16}) elements.

Mathematically separable signal representations, i.e., using separable dictionaries, have been typically used for multidimensional signals, as they are simpler and easier to obtain than non-separable representations [16], [72]. Caiafa *et al*. introduced Kronecker-OMP, a generalization of OMP that could represent multidimensional signals, represented by tensors, using separable dictionaries [16]. They also developed the N-BOMP algorithm to exploit block-sparse structures in multidimensional signals. However, similar to OMP, Kronecker-OMP could only obtain an approximate nonglobally optimal solution of the nonconvex L_0 constrained sparse least-squares problem [17], [23]. Also, there is currently no computationally efficient method to obtain a sparse representation of a multidimensional signal by solving the convex L_1 constrained sparse least-squares problem for all critical values of the regularization parameter λ. However, two of our co-authors, Elrewainy and Sherif, earlier developed the *Kronecker Least Angle Regression* (K-LARS) algorithm to efficiently solve either large L_0 or large L_1 sparse least-squares problems (overdetermined) with a particular Kronecker form $\boldsymbol{A} \otimes \boldsymbol{I}$, for all critical values of the regularization parameter λ. They used K-LARS to sparsely fit one-dimensional multi-channel hyperspectral spectral imaging data to a Kronecker model $\boldsymbol{A} \otimes \boldsymbol{I}$ [17].

In this chapter, we develop a generalization of K-LARS, *Tensor Least Angle Regression* (T-LARS) [18] that could efficiently solve either large L_0 or large L_1 multidimensional sparse least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ. We also discuss the compressed sensing problem and use T-LARS to efficiently obtain a sparse representation of large tensors using compressed sensed samples.

The applications of T-LARS include compression of large multidimensional signals, e.g., multidimensional biomedical images, videos, satellite imaging, communication. The T-LARS could also be used in the sparse coding methods of the tensor dictionary learning algorithms such as the Tensor Method of Optimal Directions(T-MOD) and Kronecker Higher-Order SVD (K-HOSVD) [30] to learn *mode-n* dictionaries efficiently. Also, T-LARS with T-MOD or K-HOSVD could be used to efficiently solve tensor regression problems [72], [73]. In this chapter, we used T-LARS to represent 3D MRI brain images and 3D PET-CT brain images using significantly lower coefficients than the number of elements in the original signals.

This chapter is organized as follows: Section 3.2 includes a brief introduction to tensors, tensor operations, multilinear sparse least-squares problems, and multilinear compressed sensing. In Section 3.3, we review Least Angle Regression (LARS) and describe our Tensor Least Angle Regression (T-LARS) algorithm in detail. Section 3.4 presents the computational complexity of our T-LARS algorithm and compares its computational complexity with that of Kronecker-OMP. Section 3.5 provides experiment results of applying both T-LARS and Kronecker-OMP. We present our conclusions in Section 3.6.

3.2. Problem Formulation

3.2.1. Tensors and multilinear transformations

The term tensor has a specific mathematical definition in physics, but it has been widely accepted in many disciplines, e.g., signal processing and statistics, to mean a multidimensional array. Therefore, a vector is a first-order tensor, and a matrix is a second-order tensor. An *N*-dimensional array is an N^{th} order tensor, whose N dimensions are also known as modes [6], [43]. The N^{th} order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ has N modes, with dimensions, $I_1,\ I_2, \ldots,\ I_N$, where vectors along a specific mode, n, are called *mode-n* fibers.

Vectorization and *mode-n* matricization of tensors [6], [43] are two important tensor reshaping operations. As the names imply, the vectorization of a tensor generates a vector, and the matricization of a tensor generates a matrix. Tensors are vectorized by stacking *mode-1* fibers in reverse lexicographical order, where this vectorization is denoted $\mathrm{vec}(\mathcal{X})$.

$$\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N} \longrightarrow \mathrm{vec}(\mathcal{X}) \in \mathbb{R}^{I_\mathrm{N} I_{N-1}\ldots I_1}$$

In *mode-n* tensor matricization, *mode-n* fibers become the columns of the resulting matrix. We note that such ordering of these columns is not consistent across the literature [5], [45]. In this chapter, we use reverse lexicographical order $(I_N \ldots I_{n+1} I_{n-1} \ldots I_1)$ for the column ordering in *mode-n* tensor matricization. In such reverse lexicographical order, I_1 varies the fastest and I_N varies the slowest. Let $\boldsymbol{X}_{(n)}$ denote *mode-n* matricization of a tensor $\mathcal{X}$

$$\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N} \longrightarrow \boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n\times(I_N\ldots I_{n+1}I_{n-1}\ldots I_1)}$$

Another important operation that could be performed on a tensor is its *mode-n* product, i.e., its multiplication by a matrix along one of its modes. Let the tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$ and a matrix $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_\mathrm{n}\times I_n}$, then their *mode-n* product, $\mathcal{Y}_n \in \mathbb{R}^{I_1\times\ldots\times J_\mathrm{n}\times\ldots\times I_N}$, is denoted as

$$\mathcal{Y}_n = \mathcal{X} \times_n \boldsymbol{\Phi}^{(n)} \tag{3.1}$$

where all *mode-n* fibers of the tensor are multiplied by the matrix $\boldsymbol{\Phi}^{(n)}$. Equation (3.1) could also be written as $\boldsymbol{Y}_{(n)} = \boldsymbol{\Phi}^{(n)}\boldsymbol{X}_{(n)}$, where $\boldsymbol{Y}_{(n)}$ and $\boldsymbol{X}_{(n)}$ are *mode-n* matricizations of tensors $\mathcal{X}$ and $\mathcal{Y}$, respectively. A *mode-n* product could be thought of as a linear transformation of the *mode-n* fibers of a tensor. Therefore, a multilinear transformation of a tensor $\mathcal{X}$ could be defined as

$$\mathcal{Y} = \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} \tag{3.2}$$

where, $\boldsymbol{\Phi}^{(n)}$; $n \in \{1,2,\ldots,N\}$ are matrices with dimensions $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_\mathrm{n}\times I_n}$; $n \in \{1,2,\ldots,N\}$ and $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_\mathrm{n}\times\ldots\times J_\mathrm{N}}$ [6], [43]. This multilinear transformation could also be written as a product of $\mathrm{vec}(\mathcal{X})$ and the Kronecker product of matrices $\boldsymbol{\Phi}^{(n)}$; $n \in \{1,2,\ldots,N\}$ [6], [43].

$$\mathrm{vec}(\mathcal{Y}) = \left(\boldsymbol{\Phi}^{(N)} \otimes \boldsymbol{\Phi}^{(N-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}\right)\mathrm{vec}(\mathcal{X}) \tag{3.3}$$

We note that (3.3) is a linear system relating to $\boldsymbol{x} = \mathrm{vec}(\mathcal{X})$ and $\boldsymbol{y} = \mathrm{vec}(\mathcal{Y})$. If matrix $\boldsymbol{\Phi}$ represents a separable dictionary, i.e.,

$$\boldsymbol{\Phi} = (\boldsymbol{\Phi}^{(N)} \otimes \boldsymbol{\Phi}^{(N-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(2)} \otimes \boldsymbol{\Phi}^{(1)}) \tag{3.4}$$

Then (3.3) describes a representation of $\boldsymbol{y} = \mathrm{vec}(\mathcal{Y})$ using a dictionary $\boldsymbol{\Phi}$, where $\boldsymbol{x} = \mathrm{vec}(\mathcal{X})$ represents its coefficients ($\boldsymbol{y} = \boldsymbol{\Phi x}$). Similarly, we could think of (3.2) as a representation of tensor $\mathcal{Y}$ using dictionaries $\boldsymbol{\Phi}^{(n)}$; $n \in \{1,2,\dots,N\}$, where tensor $\mathcal{X}$ represents its coefficients.

3.2.2. Sparse Multilinear Least-squares Problem

A sparse multilinear representation of (3.3) could be obtained by rewriting it as an L_p minimization problem [9],

$$\tilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\| \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}\right) \mathrm{vec}(\mathcal{X}) - \mathrm{vec}(\mathcal{Y}) \right\|_2^2 + \lambda \|\mathrm{vec}(\mathcal{X})\|_p \quad (3.5)$$

where λ is a regularization parameter.

Alternatively, using (3.2) and (3.3), (3.5) could be written as

$$\tilde{\mathcal{X}} = \arg\min_{\mathcal{X}} \left\| \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} - \mathcal{Y} \right\|_2^2 + \lambda \|\mathcal{X}\|_p \quad (3.6)$$

where $\mathcal{Y} \in \mathbb{R}^{J_1 \times \dots \times J_n \times \dots \times J_N}$ and $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_n \times \dots \times I_N}$.

For the same values of p, the L_p minimization problems in (3.5) and (3.6) are equivalent, even though their formulations are in vector and tensor forms, respectively. We note that for $0 \leq p < 1$, (3.5) and (3.6) are nonconvex optimization problems, while for $p \geq 1$, they are convex optimization problems. The most sparse solution of (3.5) and (3.6) would be obtained when $p = 0$, $\text{i.e.}, L_0$ constrained problem, but its sparsity would be reduced as p increases.

The L_0 minimization problem ($p = 0$) is a nonconvex optimization problem, whose vector formulation, problem (3.5), could be solved approximately, i.e., non-globally, using Orthogonal Matching Pursuit (OMP) or Least Angle Regression (LARS) [23]. The L_1 minimization problem ($p = 1$) is a convex optimization problem, whose vector formulation, problem (3.5), could be exactly, i.e., globally, solved using Basis Pursuit (BP) or LARS [73]. However, OMP, BP, and LARS share a serious drawback in that they are not suitable for solving very large sparse least-squares problems as they involve multiplication and inverting of very large matrices.

As a way of overcoming this drawback, Caiafa *et al*. proposed Kronecker-OMP, a tensor-based generalization of OMP, for solving sparse multilinear least-squares problems. However, similar to OMP, Kronecker-OMP could only obtain a non-globally optimal solution of the nonconvex L_0 constrained sparse least-squares problem [73].

3.2.3. Multilinear Compressed Sensing

Consider a multilinear transformation of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ by *mode-n* dictionary matrices $\boldsymbol{D}^{(n)} \in \mathbb{R}^{L_n \times I_n}$; $n \in \{1,2,\ldots,N\}$ to obtain a large tensor $\mathcal{A} \in \mathbb{R}^{L_1 \times \ldots \times L_n \times \ldots \times L_n}$.

$$\mathcal{A} = \mathcal{X} \times_1 \boldsymbol{D}^{(1)} \times_2 \boldsymbol{D}^{(2)} \times_3 \cdots \times_N \boldsymbol{D}^{(N)} \tag{3.7}$$

The main objective of the multilinear compressed sensing is to obtain a sparse coefficient tensor $\mathcal{X}$ by sampling the large tensor signal $\mathcal{A}$ in a sparse domain, where a much smaller sample tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}; \forall n\, J_n \leq L_n$, is obtained. The smaller sample tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ is a projection of the tensor $\mathcal{A}$ to a sparse domain using *mode-n* sensing matrices $\boldsymbol{Z}^{(n)} \in \mathbb{R}^{J_n \times L_n}$; $n \in \{1,2,\ldots,N\}$, where $\mathcal{Y} = \mathcal{A} \times_1 \boldsymbol{Z}^{(1)} \times_2 \cdots \times_N \boldsymbol{Z}^{(N)}$ [16], [21], [74].

Therefore, the relationship between the sparse coefficient tensor $\mathcal{X}$ and the compressed sensing samples tensor $\mathcal{Y}$ is given by,

$$\mathcal{Y} = \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} \tag{3.8}$$

Where $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n} = \boldsymbol{Z}^{(n)}\boldsymbol{D}^{(n)}; \forall\, n \in \{1,2,\ldots,N\}$.

As shown in (3.7), the tensor $\mathcal{A}$ could be reconstructed using the sparse coefficient tensor $\mathcal{X}$, and *mode-n* dictionary matrices $\boldsymbol{D}^{(n)} \in \mathbb{R}^{L_n \times I_n}$; $n \in \{1,2,\ldots,N\}$, where the sparse coefficient tensor $\mathcal{X}$, could be calculated using the much smaller sample tensor $\mathcal{Y}$ by solving a sparse multilinear least-squares problem,

$$\widetilde{\mathcal{X}} = \arg\min_{\mathcal{X}} \left\| \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)} - \mathcal{Y} \right\|_2^2 + \lambda \|\mathcal{X}\|_p \tag{3.9}$$

Equation (3.6) and (3.9) are L_p constrained multilinear least-squares problems. A sparse tensor $\mathcal{X}$ could be obtained by solving either L_0 or L_1 constrained minimization problems. As we discussed in section 3.2.2, Kronecker-OMP could be used to obtain a non-globally optimal solution of the nonconvex L_0 constrained multilinear least-squares problem.

In this chapter, we develop Tensor Least Angle Regression (T-LARS), a computationally efficient method to solve either large L_0 or L_1 constrained multilinear least-squares problem in (3.6) and (3.9) for all critical values of the regularization parameter λ.

3.3. Tensor Least Angle Regression (T-LARS)

Least angle regression (LARS) is a computationally efficient method to solve either L_0 or L_1 constrained minimization problem in vector form, problem (3.5), for all critical values of the regularization parameter λ [15]. In this chapter, we develop a generalization of LARS, Tensor Least Angle Regression (T-LARS), to solve large sparse tensor least-squares problems to, for example, obtain sparse representations of multidimensional signals using a separable dictionary as described by (3.3). As shown below, our T-LARS calculations are performed without explicitly generating or inverting large matrices, thereby keeping its computational complexity and memory requirement relatively low. Both T-LARS and Kronecker-OMP algorithms use the *Schur complement* inversion formula for inverting large matrices without explicitly inverting them [16].

3.3.1. Least Angle Regression (LARS)

Least angle regression (LARS) solve the L_0 or L_1 constrained minimization problem in (3.5) for all critical values of the regularization parameter λ. LARS starts with a very large value of λ that results in an empty active columns matrix, $\boldsymbol{\Phi}_I$, and a solution $\tilde{\boldsymbol{x}}_{t=0} = \boldsymbol{0}$. The set I denotes an active set of the dictionary $\boldsymbol{\Phi}$, i.e., column indices where the optimal solution $\tilde{\boldsymbol{x}}_t$ at iteration t, is nonzero, and I^c denotes its corresponding inactive set. Therefore, $\boldsymbol{\Phi}_I$ contains only the active columns of the dictionary $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}_{I^c}$ contains only its inactive columns.

At each iteration t, a new column is either added (L_0) to the active set I or a new column is either added or removed (L_1) from the active set I, and λ is reduced by a calculated value δ_t^*. As a result of such iterations, new solutions with an increased number of coefficients that follow a piecewise linear path are obtained until a predetermined residual error ε is obtained. One important characteristic of LARS is that the current solution at each iteration is the optimum sparse solution for the selected active columns.

Initialization of LARS includes setting the active set to an empty set, I= {}, the initial solution vector $\tilde{\boldsymbol{x}}_0 = 0$, the initial residual vector $\boldsymbol{r}_0 = \boldsymbol{y}$ and initial regularization coefficient $\lambda_1 =$

$\max(\boldsymbol{c}_1)$ where, $\boldsymbol{c}_1 = \boldsymbol{\Phi}^T \boldsymbol{r}_0$. The optimal solution $\widetilde{\boldsymbol{x}}_t$ at any iteration, t must satisfy the following two optimality conditions,

$$\left\| \boldsymbol{\Phi}_{I^c}^T \boldsymbol{r}_t \right\|_\infty \leq \lambda_t \tag{3.10}$$

$$\boldsymbol{\Phi}_I^T \boldsymbol{r}_t = -\lambda_t \boldsymbol{z}_t \tag{3.11}$$

where, $\boldsymbol{r}_t$ is the residual vector at iteration t, $\boldsymbol{r}_t = \boldsymbol{y} - \boldsymbol{\Phi}\widetilde{\boldsymbol{x}}_t$, and $\boldsymbol{z}_t$ is the sign sequence of $\boldsymbol{c}_t$ on the active set I.

The condition in (3.11) ensures that the magnitude of the correlation between all active columns and the residual is equal to $|\lambda_t|$ at each iteration, and the condition in (3.10) ensures that the magnitude of the correlation between the inactive columns and the residual is less than or equal to $|\lambda_t|$.

For L_1 constrained minimization problem, at each iteration, if an inactive column violates the condition (3.10), it is added to the active set, and if an active column violates the condition (3.11), it is removed from the active set. For L_0 constrained minimization problem only the columns that violate the condition (3.10) are added to the active set at each iteration.

For a given active set I, the optimal solution $\widetilde{\boldsymbol{x}}_t$ could be written as

$$\widetilde{\boldsymbol{x}}_t = \begin{cases} \left(\boldsymbol{\Phi}_{I_t}{}^T \boldsymbol{\Phi}_{I_t}\right)^{-1} \left(\boldsymbol{\Phi}_{I_t}{}^T \boldsymbol{y} - \lambda_t \boldsymbol{z}_t\right), & \text{on } I \\ 0, & \text{Otherwise} \end{cases} \tag{3.12}$$

where, $\boldsymbol{z}_t$ is the sign sequence of $\boldsymbol{c}_t$ on the active set I, and $\boldsymbol{c}_t = \boldsymbol{\Phi}^T \mathrm{r}_{t-1}$ is the correlation vector of all columns of the dictionary $\boldsymbol{\Phi}$ with the residual vector $\boldsymbol{r}_{t-1}$ at iteration *t*. The Least Angle Regression (LARS) algorithm is summarized below.

Algorithm 3.1: Least Angle Regression (LARS)

```
Input: $LARS\_mode = L_1$ or $L_0$; stopping criterion: residual error: $\varepsilon$, or number of non-zero coefficients: $K$; normalized $\boldsymbol{y}$; dictionary $\boldsymbol{\Phi}$
Initialization: Residual: $\boldsymbol{r}_0 = \boldsymbol{y}$; $\boldsymbol{x} = \boldsymbol{0}$; active set: $I = \{\}$;
1.  $\boldsymbol{c}_1 = \boldsymbol{\Phi}^T \boldsymbol{r}_0$
2.  $[\lambda_1, column\_idx] = max(\boldsymbol{c}_1)$;
3.  $I = \{column\_idx\}$;
4.  while stopping criterion not reached
5.      $\boldsymbol{z}_t = \mathrm{sign}\,(\boldsymbol{c}_t(I))$
6.      $\boldsymbol{d}_t = \left(\boldsymbol{\Phi}_{I_t}^T \boldsymbol{\Phi}_{I_t}\right)^{-1} \boldsymbol{z}_t$
7.      $\boldsymbol{v}_t = \boldsymbol{\Phi}^T \boldsymbol{\Phi}_{I_t} \boldsymbol{d}_t$
8.      for $i = 1$ to column length of $\boldsymbol{\Phi}$ do
9.        if $i \in I^c$
10.           $\delta_t^+ = \min\left\{\frac{\lambda_t - \boldsymbol{c}_t(i)}{1 - \boldsymbol{v}_t(i)}, \frac{\lambda_t + \boldsymbol{c}_t(i)}{1 + \boldsymbol{v}_t(i)}\right\}$
11.           if $\delta_t^+ < \delta_t^*$
12.             $\delta_t^* = \delta_t^+$; $column\_idx = i$; add_column = True;
13.           end
14.       elseif $LARS\_mode == L_1$
15.           $\delta_t^- = -\frac{\boldsymbol{x}_{t-1}(i)}{\boldsymbol{d}_t(i)}$
16.           if $\delta_t^- < \delta_t^*$
17.             $\delta_t^* = \delta_t^-$; $column\_idx = i$; add_column = False;
18.           end
19.       end
20.     end for
21.     $\boldsymbol{x} = \boldsymbol{x} + \delta_t^* \boldsymbol{d}_t$
22.     $\lambda_t = \lambda_t - \delta_t^*$
23.     $\boldsymbol{c}_t = \boldsymbol{\Phi}^T \boldsymbol{r}_t$
24.     $\boldsymbol{r}_t = \boldsymbol{r}_{t-1} - \delta_t^* \boldsymbol{\Phi}_{I_t} \boldsymbol{d}_t$
25.     if add_column == True
26.       $I = I + \{column\_idx\}$
27.     else
28.       $I = I - \{column\_idx\}$
29.     end
30. end while
31. return $I, \boldsymbol{x}$
```

3.3.2. Tensor Least Angle Regression (T-LARS) Algorithm

Tensor Least Angle Regression (T-LARS) is a generalization of Least Angle Regression (LARS) to solve the sparse multilinear least-squares problem in (3.6) using tensors and multilinear algebra. Unlike LARS, T-LARS does not calculate large matrices such as the Kronecker dictionary, $\boldsymbol{\Phi}$ in (3.4), which is required in vectorized sparse multilinear least-squares problems. Instead, T-LARS uses much smaller *mode-n* dictionaries for calculations. A mapping between column indices of dictionary $\boldsymbol{\Phi}$ and column indices of *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ is essential in T-LARS calculations (See Appendix 1).

Required inputs to the T-LARS algorithm are the tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$, *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ and the stopping criterion as a residual tolerance ε or the maximum number of non-zero coefficients K (K-sparse representation). The output is the solution tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$.

First, normalize the tensor $\mathcal{Y}$ and columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ to have a unit L_2 norm. Note that normalizing columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ ensure normalization of the separable dictionary $\boldsymbol{\Phi}$ in (3.4) (See Appendix 2). For notational simplicity in the following sections, we will use $\mathcal{Y}$ to represent the normalized tensor and $\boldsymbol{\Phi}^{(n)}$ to represent normalized dictionary matrices.

Gram matrices are used in several steps of T-LARS. For a large separable dictionary, $\boldsymbol{\Phi}$, its Gram matrix $\boldsymbol{G} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}$ would be large as well. Therefore, explicitly building this Gram matrix and using it in computations could be very inefficient for large problems. Instead, T-LARS uses Gram matrices of *mode-n* dictionary matrices, $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \ldots, \boldsymbol{\Phi}^{(N)}$, defined as $\boldsymbol{G}^{(1)}, \boldsymbol{G}^{(2)}, \ldots, \boldsymbol{G}^{(N)}$. We could obtain a Gram matrix $\boldsymbol{G}^{(n)}; n \in \{1, \cdots, N\}$ for each *mode-n* dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ by,

$$\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T} \boldsymbol{\Phi}^{(n)} \tag{3.13}$$

The tensor $\mathcal{C}_1$ is the correlation between the tensor $\mathcal{Y}$ and the *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$.

$$\mathcal{C}_1 = \mathcal{Y} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \ldots \times_n \boldsymbol{\Phi}^{(n)^T} \times_{n+1} \ldots \times_N \boldsymbol{\Phi}^{(N)^T} \tag{3.14}$$

The tensor $\mathcal{C}_1$ could be calculated efficiently as N *mode-n* products, and the initial correlation vector is obtained by vectorizing $\mathcal{C}_1$, where, $\boldsymbol{c}_1 = \operatorname{vec}(\mathcal{C}_1)$ (See Appendix 3).

T-LARS requires several parameters to be initialized before starting the iterations. The regularization parameter λ_1 is initialized to the maximum value of the correlation vector $\boldsymbol{c}_1$ and the corresponding most correlated column $\boldsymbol{\phi}_{I_1}$ from the separable dictionary, $\boldsymbol{\Phi}$ is added to the initial active set *I*. The initial residual tensor $\mathcal{R}_0$ is set to $\mathcal{Y}$ and the initial solution vector $\boldsymbol{x}_0$ and the initial direction vector $\boldsymbol{d}_0$ is set to $\mathbf{0}$. Initial step size δ_0^* is also set to 0. T-LARS starts the iterations at $t = 1$ to run until a stopping criterion is reached.

- Initial residual tensor: $\mathcal{R}_0 = \mathcal{Y}$
- Initial solution vector: $\widetilde{\boldsymbol{x}}_0 \ = \ \mathbf{0}$
- Initial direction vector: $\boldsymbol{d}_0 \ = \ \mathbf{0}$
- Initial step size: δ_0^* = 0
- Initial regularization parameter: $\lambda_1 = \ \max(\boldsymbol{c}_1)$
- Active set: $I \ = \ \{\boldsymbol{\phi}_{I_1}\}$
- Start iterations at $t = 1$

The following calculations are performed at every iteration $t = 1, 2, \ldots$ of the T-LARS algorithm until the stopping criterion is reached.

3.3.2.1. Obtain the inverse of the Gram matrix of the active columns of the dictionary

We obtain the Gram matrix of the active columns of the dictionary $\boldsymbol{G}_t = \boldsymbol{\Phi}_{I_t}{}^T \boldsymbol{\Phi}_{I_t}$ at each iteration t. The size of this Gram matrix would either increase (dictionary column addition) or decrease (dictionary column removal) with each iteration t. Therefore, for computational efficiency, we use the *Schur complement* inversion formula to calculate $\boldsymbol{G}_t^{-1}$ from $\boldsymbol{G}_{t-1}^{-1}$, thereby avoiding its full calculation [16], [75], [76].

a) Updating the Gram matrix after adding a new column $\boldsymbol{k_a}$ to the active set

Let the column $k_a \in I$ be the new column added to the active matrix. Given $\boldsymbol{G}_{t-1}^{-1}$, the inverse of the Gram matrix $\boldsymbol{G}_t^{-1}$ could be calculated using the *Schur complement* inversion formula for a symmetric block matrix [77]–[79],

$$\boldsymbol{G}_t^{-1} = \begin{bmatrix} \boldsymbol{F}_{11}^{-1} & \alpha\boldsymbol{b} \\ \alpha\boldsymbol{b}^{\mathrm{T}} & \alpha \end{bmatrix} \tag{3.15}$$

where, $\boldsymbol{F}_{11}^{-1} = \ \boldsymbol{G}_{t-1}^{-1} + \ \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$, $\boldsymbol{b} \ = \ -\boldsymbol{G}_{t-1}^{-1}\mathbf{g}_{\mathrm{a}}$ and $\alpha \ = 1 \,/\, \left(\mathrm{g}_{(k_a,k_a)} + \ \mathbf{g}_{\mathrm{a}}^T\boldsymbol{b}\right)$ and the column vector $\mathbf{g}_{\mathrm{a}}^T$ is given by,

$$\mathbf{g}_{\mathrm{a}}^T \ = \ [\mathrm{g}_{(k_1,k_a)} \quad \cdots \ \mathrm{g}_{(k_n,k_a)} \cdots \quad \mathrm{g}_{(k_{a-1},k_a)}]_{1\times a-1}$$

The elements, $\mathrm{g}_{(k_n,k_a)}$ of $\mathbf{g}_{\mathrm{a}}^{T}$ are elements of the gram matrix, $\boldsymbol{G_t}$ that are obtained using *mode-n* gram matrices $\boldsymbol{G}^{(n)}; n \in \{1, \cdots, N\}$.

$$\mathrm{g}(k_n, k_a) \;= \mathrm{g}^{(N)}\big(k_{n_N}, k_{a_N}\big) \otimes \ldots \otimes \mathrm{g}^{(1)}\big(k_{n_1}, k_{a_1}\big)$$

where, $k_{n_N} \cdots k_{n_1}$ are the tensor indices corresponds to the column index k_n and $k_{a_N} \cdots k_{a_1}$are the tensor indices corresponds to the column index k_a (See Appendix 1).

b) Updating the Gram matrix after removing a column $\boldsymbol{k_r}$ from the active set

Let the column $k_r \in I$ be the column removed from the active set. We move column k_r and row k_r of $\boldsymbol{G}_{t-1}^{-1}$ to become its last column and last row, respectively. We denote this new matrix as $\widehat{\boldsymbol{G}}_{t-1}^{-1}$. By using the *Schur complement* inversion formula for a symmetric block matrix, the inverse $\widehat{\boldsymbol{G}}_{t-1}^{-1}$ could be interpreted as

$$\widehat{\boldsymbol{G}}_{t-1}^{-1} \;= \begin{bmatrix} \boldsymbol{F}_{11}^{-1} & \alpha\boldsymbol{b} \\ \alpha\boldsymbol{b}^{\mathrm{T}} & \alpha \end{bmatrix}_{\mathrm{N}\times\mathrm{N}} \tag{3.16}$$

where, $\boldsymbol{F}_{11}^{-1} = \boldsymbol{G}_t^{-1} + \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$. Therefore, we could calculate the inverse of the Gram matrix at iteration t as [75], [76],

$$\boldsymbol{G}_t^{-1} \;= \boldsymbol{F}_{11}^{-1} \,-\, \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}} \tag{3.17}$$

Both $\boldsymbol{F}_{11}^{-1}$ and $\alpha\boldsymbol{b}\boldsymbol{b}^{T}$ could be easily obtained from ($\widehat{\boldsymbol{G}}_{t-1}^{-1}$) as follows (MATLAB notation)

$$\boldsymbol{F}_{11}^{-1} \;=\; \widehat{\boldsymbol{G}}_{t-1}^{-1}(1{:}N-1, 1{:}N-1) \tag{3.18}$$

$$\alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}} \;= \frac{\widehat{\boldsymbol{G}}_{t-1}^{-1}(1{:}N-1,\ N)\widehat{\boldsymbol{G}}_{t-1}^{-1}(N,\ 1{:}N-1)}{\widehat{\boldsymbol{G}}_{t-1}^{-1}(N,\ N)} \tag{3.19}$$

3.3.2.2. Obtain direction vector d_t

The direction vector, along which the solution $\boldsymbol{x}$ follows in a piecewise linear fashion when an active column is added to or removed from the active set, is given by

$$\boldsymbol{d}_t = \boldsymbol{G}_t^{-1}\boldsymbol{z}_t \tag{3.20}$$

where, $\boldsymbol{z}_t \;=\; sign\big(\boldsymbol{c}_t(I)\big)$, i.e., the sign sequence of the correlation vector over the active set.

3.3.2.3. Obtain v_t

A vector $\boldsymbol{v}_t$ could be defined as

$$\boldsymbol{v}_t = \boldsymbol{\Phi}^T \boldsymbol{\Phi}_{I_t} \boldsymbol{d}_t \tag{3.21}$$

This vector $\boldsymbol{v}_t$ could be efficiently obtained as a multilinear transformation of a direction tensor $\mathcal{D}_\mathrm{t}$ by the Gram matrices $\boldsymbol{G}^{(n)};\ n \in \{1, \cdots, N\}$

$$\mathcal{V}_\mathrm{t} = \mathcal{D}_\mathrm{t} \times_1 \boldsymbol{G}^{(1)} \times_2 \ldots \times_n \boldsymbol{G}^{(2)} \times_{n+1} \ldots \times_N \boldsymbol{G}^{(N)} \tag{3.22}$$

where $\mathrm{vec}(\mathcal{D}_\mathrm{t}(I)) = \boldsymbol{d}_t$, and $\mathcal{D}_\mathrm{t}(I^c) = 0$. We note that $\mathrm{vec}(\mathcal{V}_\mathrm{t}) = \boldsymbol{v}_t$.

3.3.2.4. Obtain the correlation vector c_t

As $\boldsymbol{c_1}$ would be obtained at initialization, the following calculations are needed only any iteration $t \geq 2$. The correlation vector $\boldsymbol{c_t}$ is given by

$$\boldsymbol{c_t} = \boldsymbol{\Phi}^T \mathrm{vec}(\mathcal{R}_{t-1}) \tag{3.23}$$

where $\mathcal{R}_{t-1}$ is the residual tensor from the previous iteration.

Since

$$\mathrm{vec}(\mathcal{R}_{t-1}) = \mathrm{vec}(\mathcal{R}_{t-2}) - \delta_{t-1}^* \boldsymbol{\Phi}_{I_{t-1}} \boldsymbol{d}_{t-1} \tag{3.24}$$

we could update the correlation vector $\boldsymbol{c_t}$ by,

$$\boldsymbol{c_t} = \boldsymbol{\Phi}^T \mathrm{vec}(\mathcal{R}_{t-2}) - \delta_{t-1}^* \boldsymbol{\Phi}^T \boldsymbol{\Phi}_{I_{t-1}} \boldsymbol{d}_{t-1} \tag{3.25}$$

Substituting (3.21) and (3.23) into (3.25), we obtain an update for the correlation

$$\boldsymbol{c_t} = \boldsymbol{c}_{t-1} - \delta_{t-1}^* \boldsymbol{v}_{t-1} \tag{3.26}$$

3.3.2.5. Calculate step size δ^*

The minimum step size for adding a new column to the active set is given by,

$$\delta_t^+ = \min_{i \in I^c} \left\{ \frac{\lambda_t - \boldsymbol{c}_t(i)}{1 - \boldsymbol{v}_t(i)}, \frac{\lambda_t + \boldsymbol{c}_t(i)}{1 + \boldsymbol{v}_t(i)} \right\} \tag{3.27}$$

The minimum step size for removing a column from the active set is given by,

$$\delta_t^- = \min_{i \in I} \left\{ -\frac{\boldsymbol{x}_{t-1}(i)}{\boldsymbol{d}_t(i)} \right\} \tag{3.28}$$

Therefore, the minimum step size for L_1 constrained sparse least-squares problem is

$$\delta_t^* = \min \{\delta_t^+, \delta_t^-\}. \tag{3.29}$$

For the L_0 constrained sparse least-squares problem, only new columns are added to the active set at every iteration. Therefore, the minimum step size for L_0 constrained sparse least-squares problem is $\delta_t^* = \delta_t^+$.

3.3.2.6. Update the solution $\tilde{x}_t$, regularization parameter, and residual

The current solution $\tilde{\boldsymbol{x}}_\mathrm{t} = \mathrm{vec}(\tilde{\mathcal{X}}_t)$ is given by,

$$\tilde{\boldsymbol{x}}_t = \tilde{\boldsymbol{x}}_{t-1} + \delta_t^* \boldsymbol{d}_t \tag{3.30}$$

Update λ_{t+1} and $\mathcal{R}_t$ for the next iteration

$$\lambda_{t+1} = \lambda_t - \delta_t^* \tag{3.31}$$

The residual tensor $\mathcal{R}_t$ could be calculated as,

$$\mathcal{R}_t = \mathcal{R}_{t-1} - \delta_t^* \mathcal{D}_\mathrm{t} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} \tag{3.32}$$

3.3.2.7. Check stopping criterion

Check if either of the following stopping criteria has been reached

$$\|\mathcal{R}_t\|_2 < \varepsilon \tag{3.33}$$

or

$$length(I) \geq K \tag{3.34}$$

where $\|\mathcal{R}_t\|_2 = \left\| \mathcal{Y} - \tilde{\mathcal{X}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)} \right\|_2$ is the L_2 norm of the residual error after the iteration *t*, where for a normalized $\mathcal{Y}$ and column normalized *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)}$,

$$0 \leq \|\mathcal{R}_t\|_2 \leq 1$$

T-LARS algorithm solves the sparse tensor least-squares problem in (3.6) to obtain a sparse solution $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ for a tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ using *N* *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in$

$\mathbb{R}^{J_n \times I_n}; \forall\ n \in \{1,..N\}$. Tensor $\mathcal{Y}$ and N *mode-n* dictionaries $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \dots, \boldsymbol{\Phi}^{(N)}$, are the inputs to the T-LARS algorithm, where $N \geq 1$. T-LARS algorithm could be used to solve underdetermined, square, or overdetermined sparse tensor least-squares problems, where the *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}; \forall\ n \in \{1,..N\}$, are over-complete dictionaries $(J_n < I_n)$, complete dictionaries $(J_n = I_n)$ or under-complete dictionaries $(J_n > I_n)$, respectively.

The complete T-LARS algorithm is summarized below (Matlab notation). T-LARS algorithm given in this section solves the L_p sparse separable least-squares problem when T-LARS_mode is set to L_p.

Algorithm 3.2: Tensor Least Angle Regression (T-LARS)

```
Input: T-LARS_mode = $L_1$ or $L_0$, normalized tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \dots \times J_n \times \dots \times J_N}$; normalized dictionary
matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$; $n \in \{1,..N\}$; stopping criterion: residual tolerance: $\varepsilon$ or number of non-
zero coefficients: $K$
Initialization: Residual: $\mathcal{R}_0 = \mathcal{Y}$; $\boldsymbol{x}_0 = 0$; active set: $I = \{\}$;
1.  $\mathcal{C}_1 = \mathcal{R}_0 \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \dots \times_n \boldsymbol{\Phi}^{(n)^T} \times_{n+1} \dots \times_N \boldsymbol{\Phi}^{(N)^T}$
2.  $\boldsymbol{c}_1 = \mathrm{vec}(\mathcal{C}_1)$
3.  $[\lambda_1, column\_idx\ ] = max(\boldsymbol{c}_1)$
4.  $I = \{column\_idx\}$
5.  for $n = 1$ to $N$ do
6.      $\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T} \boldsymbol{\Phi}^{(n)}$
7.  end for
8.  while stopping criterion not reached ($\|\mathcal{R}_{t-1}\|_2 > \varepsilon$ or $length(I) < K$)
9.      $\boldsymbol{z}_t = sign\,(\boldsymbol{c}_t(I))$
10.     $\boldsymbol{G}_t^{-1}$ = updateInverseGramMatrix($\boldsymbol{G}_{t-1}^{-1}$, $\{\boldsymbol{G}^{(1)}, \dots, \boldsymbol{G}^{(N)}\}$, I, add_column, column_idx) %
        See Section (3.3.2.1)
11.     $\boldsymbol{d}_t = \boldsymbol{G}_t^{-1} \boldsymbol{z}_t$
12.     $\mathrm{vec}(\mathcal{D}_t(I)) = \boldsymbol{d}_t$
13.     $\mathcal{V}_t = \mathcal{D}_t \times_1 \boldsymbol{G}^{(1)} \times_2 \dots \times_n \boldsymbol{G}^{(2)} \times_{n+1} \dots \times_N \boldsymbol{G}^{(N)}$
14.     $\boldsymbol{v}_t = \mathrm{vec}(\mathcal{V}_t)$
15.     $\delta_{t\ 1}^{+} = (\lambda_t - \boldsymbol{c}_t(I^c))./(1 - \boldsymbol{v}_t(I^c))$     % "./" - Elementwise division
16.     $\delta_{t\ 2}^{+} = (\lambda_t + \boldsymbol{c}_t(I^c))./(1 + \boldsymbol{v}_t(I^c))$
17.     $\delta_t^{-} = -\boldsymbol{x}_{t-1}./\ \boldsymbol{d}_t$
18.     $[\delta_t^*,\ column\_idx] = min\ \left(\delta_{t\ 1}^{+}, \delta_{t\ 2}^{+}\right)$
19.     add_column == True
20.     if T-LARS_mode == $L_1$ && $min\,(\delta_t^-) < \delta_t^*$
21.         $[\delta_t^*,\ column\_idx] = min\,(\delta_t^-)$
22.         add_column = False
23.     end
24.     $\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \delta_t^* \boldsymbol{d}_t$
25.     $\lambda_{t+1} = \lambda_t - \delta_t^*$
26.     $\boldsymbol{c}_{t+1} = \boldsymbol{c}_t - \delta_t^* \boldsymbol{v}_t$
27.     $\mathcal{R}_t = \mathcal{R}_{t-1} - \delta_t^* \mathcal{D}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)}$
```

```
28.         if $add\_column == True$
29.             $I = I + \{column\_idx\}$
30.         else
31.             $I = I - \{column\_idx\}$
32.         end
33. end while
34. return $I, \boldsymbol{x}$
```

3.4. Algorithm Computational Complexity

In this section, we analyze the computational complexity of our T-LARS algorithm and compare it with the computational complexity of Kronecker-OMP. We show that the computational complexity of T-LARS is significantly lower than Kronecker-OMP when solving sparse tensor least-squares problems.

3.4.1. The Computational complexity of T-LARS

Let K be the number of iterations used in T-LARS, $P = I_1 \times \ldots \times I_n \times \ldots \times I_N$ is the number of atoms in the Kronecker dictionary $\boldsymbol{\Phi}$, and $Q = J_1 \times \ldots \times J_n \times \ldots \times J_N$ be the total number of elements in tensor $\mathcal{Y}$. In a typical sparse solution obtained by T-LARS $K \ll P$. In the following analysis, we refer to Algorithm 3.2 above, which describes the T-LARS algorithm.

Step 1 of the T-LARS algorithm runs only once and has a computational complexity of,

$$\left(I_1 Q + \frac{I_1 I_2 Q}{J_1} + \cdots + \frac{\mathrm{P}Q}{J_1 \ldots J_{\mathrm{n}-1} \times I_{n+1} \ldots I_N} + \cdots + P\mathrm{J}_{\mathrm{N}} \right)$$

Step 10 of the T-LARS algorithm obtains the inverse Gram matrix, using *Schur complement* inversion, and has complexity $\mathcal{O}(Pk + k^3)$ where k is the iteration number whose maximum value is K. The computational complexity of step 10 for column addition is $(PK + 4\sum_{k=1}^{K} k^2)$ and the computational complexity of step 10 for column removal is $(2\sum_{k=1}^{K} k^2)$. Therefore, the maximum computational complexity of step 10 is given by,

$$PK + 4\sum_{k=1}^{K} k^2$$

Steps 13 and 27 of the T-LARS algorithm involve multilinear transformations. In both steps $\mathcal{D}_{\mathrm{t}}$ is a sparse tensor with at most k non-zero entries at any iteration k. Therefore, for K iterations, the computational complexities of step 13 and step 27 are,

$$2\sum_{k=1}^{K}\sum_{n=1}^{N} kI_n$$

and

$$2\sum_{k=1}^{K}\sum_{n=1}^{N} kJ_n + 2KQ$$

respectively.

3.4.1.1. Case of overcomplete mode-n dictionaries

For over-complete *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_{\mathrm{n}}\times I_n};\ J_{\mathrm{n}} < I_n,\, n \in \{1,\cdots,N\}$, step13 of the T-LARS algorithm would have higher computational complexity compared to step 27. Therefore, the computational complexity for most computationally intensive steps of the T-LARS algorithm would be less than,

$$\left(\begin{array}{c}\left(I_1Q + \dfrac{I_1I_2Q}{J_1} + \cdots + \dfrac{\mathrm{P}Q}{J_1 \dots J_{\mathrm{n}-1} \times I_{n+1} \dots I_N} + \cdots + P\mathrm{J}_{\mathrm{N}}\right) + \left(PK + 4\sum_{k=1}^{K} k^2\right) \\ + \left(2\sum_{k=1}^{K}\sum_{n=1}^{N} kI_n + 2KQ\right)\end{array}\right) \qquad (3.35)$$

3.4.2. Comparison of computational complexities of Kronecker-OMP and T-LARS

Caiafa *et al*. earlier analyzed the computational complexity of Kronecker-OMP to solve the problem (5) given $\mathcal{Y} \in \mathbb{R}^{J_1\times\dots\times J_n\times\dots\times J_N}, J_{\mathrm{n}} = J; \forall\, n \in \{1,\cdots,N\}$ and *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J\times I}$. From [16], after K iterations, the combined computational complexity of Kronecker-OMP was given by

$$\left(\begin{array}{c} 2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right)K + \left(2I^N K \ + 7\sum_{k=1}^{K} k^{2N}\right) + \\ \left((2NJ + N \ + \ 4) \ \sum_{k=1}^{K} k^N\right) + \left((N(N-1) + 3\ )KJ^N\right) \end{array} \right) \tag{3.36}$$

To obtain the computational complexity of T-LARS to solve the problem (3.6), given $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$, $J_\text{n}; \forall\, n \in \{1, \cdots, N\}$ and *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J \times I}$, we substitute $I_\text{n} = I$ and $J_\text{n} = J;\quad \forall\, n \in \{1, \cdots, N\}$, in (3.35) to obtain

$$\left(2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right) + \left(I^N K + 4\sum_{k=1}^{K} k^2\right) + \ 2NI\sum_{k=1}^{K} k + \ 2KJ^N \right) \tag{3.37}$$

Table 3.1. Term by term comparison of the computational complexity of Kronecker-OMP and T-LARS given in (3.36) and (3.37)

	Kronecker-OMP	T-LARS
1st Term	$2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right)K$	$2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right)$
2nd Term	$2I^N K + \ 7\sum_{k=1}^{K} k^{2N}$	$I^N K + 4\sum_{k=1}^{K} k^2$
3rd Term	$(2NJ + N \ + \ 4)\sum_{k=1}^{K} k^N$	$2NI\sum_{k=1}^{K} k$
4th Term	$(N(N-1) + 3\ )\ KJ^N$	$2KJ^N$

Table 1. shows a term by term comparison of the computational complexity of Kronecker-OMP and T-LARS given in (3.36) and (3.37), respectively.

On comparing (3.36) and (3.37), we note that the first term of the computational complexity of T-LARS is more than K times lower than the first term of the computational complexity of Kronecker-OMP.

$$2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right) < 2I^N J\left(\frac{1-\left(\frac{J}{I}\right)^N}{1-\frac{J}{I}}\right)K \tag{3.38}$$

On comparing (3.36) and (3.37), we note that the second term of the computational complexity of T-LARS is $O(I^N K + K^3)$ while the second term of the computational complexity of Kronecker-OMP is $O(I^N K + K^{2N+1})$. Therefore, for $N \geq 2$ and the same number of iterations

$$I^N K + 4\sum_{k=1}^{K} k^2 < \ 2I^N K + \ 7\sum_{k=1}^{K} k^{2N}. \tag{3.39}$$

On comparing (3.36) and (3.37), we note that the third term of the computational complexity of T-LARS is $O(K^2)$ while the third term of the computational complexity of Kronecker-OMP is $O(K^{N+1})$. Therefore, for $N \geq 2$ and the same number of iterations

$$2NI\sum_{k=1}^{K} k < (2NJ + N \ + \ 4)\sum_{k=1}^{K} k^N \tag{3.40}$$

On comparing (3.36) and (3.37), we note that both fourth terms of the computational complexity of T-LARS and the fourth term of the computational complexity of Kronecker-OMP are $O(J^N)$. Therefore,

$$2KJ^N < (N(N-1) + 3\ )KJ^N \tag{3.41}$$

Therefore, from (3.38), (3.39), (3.40), and (3.41), we observe that the computational complexity of our T-LARS algorithm is significantly lower than Kronecker-OMP when solving sparse tensor least-squares problems with $N \geq 2$ with the same number of iterations.

For multi-dimensional problems, $N \geq 2$, typically $K \gg I$, therefore, the 2nd terms of the computational complexities of both T-LARS and Kronecker-OMP dominate over all other terms. Therefore, for K iterations, the asymptotic computational complexities of T-LARS and Kronecker-OMP are $O(I^N K + K^3)$ and $O(I^N K + K^{2N+1})$, respectively.

3.5. Experimental Results

In this section, we present experimental results to compare the performance of Kronecker-OMP and T-LARS when used to obtain sparse representations of 3-D brain images using both fixed and learned *mode-n* overcomplete dictionaries.

3.5.1. Experimental Datasets

For our computational experiments, we obtained a 3D MRI brain image, and a 3D PET-CT brain image, from publicly available datasets.

Our used 3D MRI brain image consists of $175 \times 150 \times 10$ voxels and was obtained from the *OASIS-3: Longitudinal Neuroimaging, Clinical, and Cognitive Dataset for Normal Aging and Alzheimer's Disease* [80]. This 3D MRI image shows a region in the brain of a 38-year-old male patient with a tumor in his right frontal lobe.

Our used 3D PET-CT brain image consisted of $180 \times 160 \times 10$ voxels and was obtained from the *Cancer Genome Atlas Lung Adenocarcinoma (TCGA-LUAD) data collection* [81]. This 3D PET-CT image shows a region in the brain of a 38-year-old female patient.

3.5.2. Experimental Setup

We compared the performance of T-LARS and Kronecker-OMP when used to obtain different sparse representations for our 3-D MRI and PET-CT brain images by solving L_0 and L_1 constrained sparse multilinear least-squares problems, using both fixed and learned overcomplete dictionaries. We also compared the performance of T-LARS and Kronecker-OMP when used to obtain sparse representations of 3D PET-CT images using compressed sensed samples.

Our fixed *mode-n* overcomplete dictionaries were unions of a Discrete Cosine Transform (DCT) dictionaries and a Symlet wavelet packet with four vanishing moments dictionaries. In this case of using fixed *mode-n* dictionaries, we obtained the required 3D sparse representations by solving either the 3-D L_0 or L_1 minimization problem.

Our learned *mode-n* overcomplete dictionaries were learned using the Tensor-Method of Optimal Directions (T-MOD) [30] algorithm. We used T-MOD to learn, three overcomplete *mode-n* dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{32\times38}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{32\times38}$and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times12}$, using random patches, $32 \times$

32×10 voxels, with a 10% overlap from either one of our used 3-D brain images (MRI or PET-CT) [29], [82]. In this case of using learned dictionaries, we obtained the required 3D sparse representations by solving either a 4-D (due to the use of image patches) L_0 or L_1 minimization problem. For fair comparison of the performance of T-LARS and Kronecker-OMP, we generated our results using two learned dictionaries, $\boldsymbol{\Phi}_{KOMP}$ and $\boldsymbol{\Phi}_{TLARS}$, that were obtained using Kronecker-OMP and T-LARS as the sparse coding algorithm used by T-MOD, respectively.

To compare the performance of T-LARS and Kronecker-OMP when used to solve L_0 and L_1 constrained sparse multilinear least-squares problems, we designed the following experiments to obtain sparse representations of our used 3-D brain images under different conditions.

1. Experiment 1: Fixed *mode-n* dictionaries - 3D L_0 minimization problem
2. Experiment 2: Learned *mode-n* dictionaries($\boldsymbol{\Phi}_{KOMP}$) - 4D L_0 minimization problem
3. Experiment 3: Learned *mode-n* dictionaries($\boldsymbol{\Phi}_{TLARS}$) - 4D L_0 minimization problem
4. Experiment 4: Fixed *mode-n* dictionaries - 3D L_1 minimization problem
5. Experiment 5: Learned *mode-n* dictionaries($\boldsymbol{\Phi}_{TLARS}$) - 4D L_1 minimization problem

All our experimental results were obtained using a MATLAB implementation of T-LARS and Kronecker-OMP on an MS-Windows machine: 2 Intel Xeon CPUs E5-2637 v4, 3.5GHz, 32GB RAM, and NVIDIA Tesla P100 GPU with 12GB memory.

3.5.3. Experimental Results for 3D MRI Brain Images

In this section, we compare the performance of T-LARS and Kronecker-OMP to obtain *K*-sparse representations of our 3D MRI brain image, $\mathcal{Y}$, $175 \times 150 \times 10$ voxels. by solving the L_0 constrained sparse tensor least-squares problem. We also obtained similar *K*-sparse representations using T-LARS by solving the L_1 optimization problem. Table 3.2 summarizes our results for the 1-5 experiments described in Section 5.2. In all experiments, the algorithms were stopped when the number of non-zero coefficients *K* reached 13,125, which is 5% of the number of elements in $\mathcal{Y}$. We note that in Table 2, the number of iterations for L_1 optimization problems is larger than *K* because, as shown in Algorithm 3.2, at each iteration, T-LARS could either add or remove non-zero coefficients to or from the solution.

Table 3.2. Summary of experimental results for our 3D MRI brain image

Experiment	Image Size	Optimization Problem	Dictionary Type	# of Iterations	Computation Time (sec) K-OMP	Computation Time (sec) T-LARS
1	$175\times150\times10$	L_0	Fixed	13,125	20,144	**434**
2	$32\times32\times10\times36$	L_0	Learned ($\boldsymbol{\Phi}_{KOMP}$)	13,125	25,002	**394**
3	$32\times32\times10\times36$	L_0	Learned ($\boldsymbol{\Phi}_{TLARS}$)	13,125	22,646	**400**
4	$175\times150\times10$	L_1	Fixed	14,216	-	**495**
5	$32\times32\times10\times36$	L_1	Learned ($\boldsymbol{\Phi}_{TLARS}$)	14,856	-	**490**

Experiment 1: Figure 3.1 and Figure 3.2 show obtained experimental results for representing our 3D MRI brain image using three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{175\times351}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{150\times302}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times26}$, by solving the L_0 minimization problem, using both T-LARS and Kronecker-OMP. The residual error of the reconstructed 3-D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.0839$ (8.39 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.0624$ (6.24 %).

Figure 3.1. Original 3D MRI brain image (a), its reconstruction using 5% non-zero coefficients ($K = 13{,}125$) obtained by Kronecker-OMP (b) and T-LARS (c) using fixed *mode-n* overcomplete dictionaries (Experiment 1)

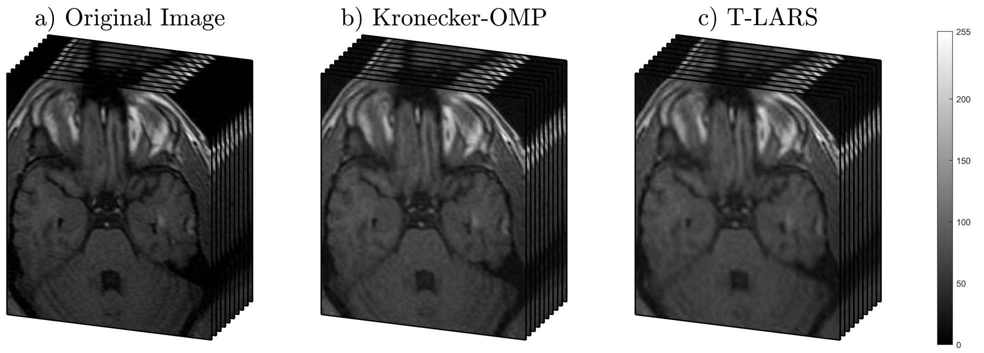
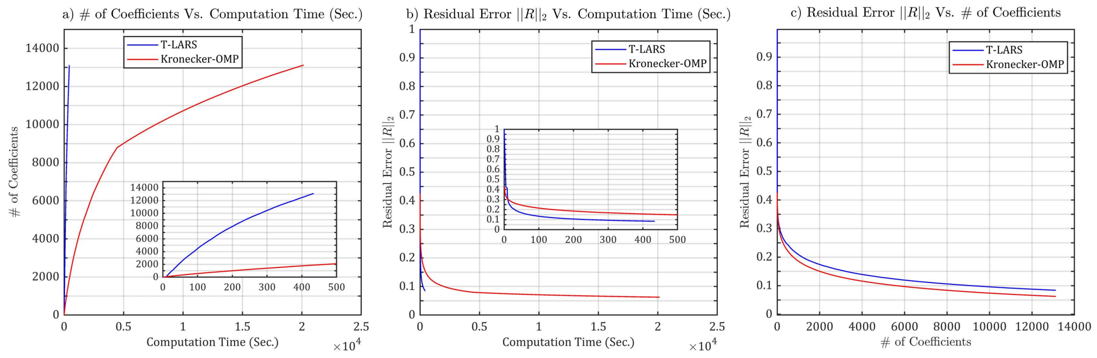


Figure 3.2. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying Kronecker-OMP and T-LARS to our 3D MRI brain image and using fixed *mode-n* overcomplete dictionaries (Experiment 1)



Experiment 2 & 3: Figure 3.3 and Figure 3.4 show obtained experimental results for representing our 3D MRI brain image using our learned overcomplete dictionaries, $\boldsymbol{\Phi}_{KOMP}$ and $\boldsymbol{\Phi}_{TLARS}$, by solving the L_0 minimization problem, using both T-LARS and Kronecker-OMP. For the $\boldsymbol{\Phi}_{KOMP}$ dictionary, the residual error of the reconstructed 3-D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.1368$ (13.68 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.1143$ (11.43 %). For the $\boldsymbol{\Phi}_{TLARS}$ dictionary, the residual error of the reconstructed 3-D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.1127$ (11.27 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.0955$ (9.55 %).

Figure 3.3. Original 3D MRI brain image (a), its reconstructions using 5% non-zero coefficients ($K = 13{,}125$), (b) - (e), the difference images, (f) - (i) obtained using Kronecker-OMP and T-LARS, using our learned overcomplete dictionaries (Experiment 2 & 3)

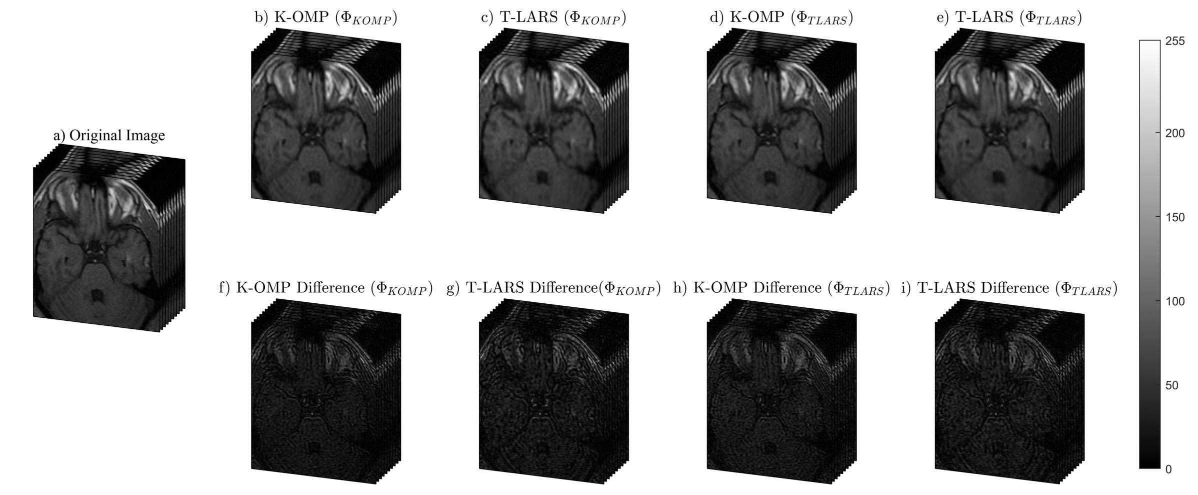
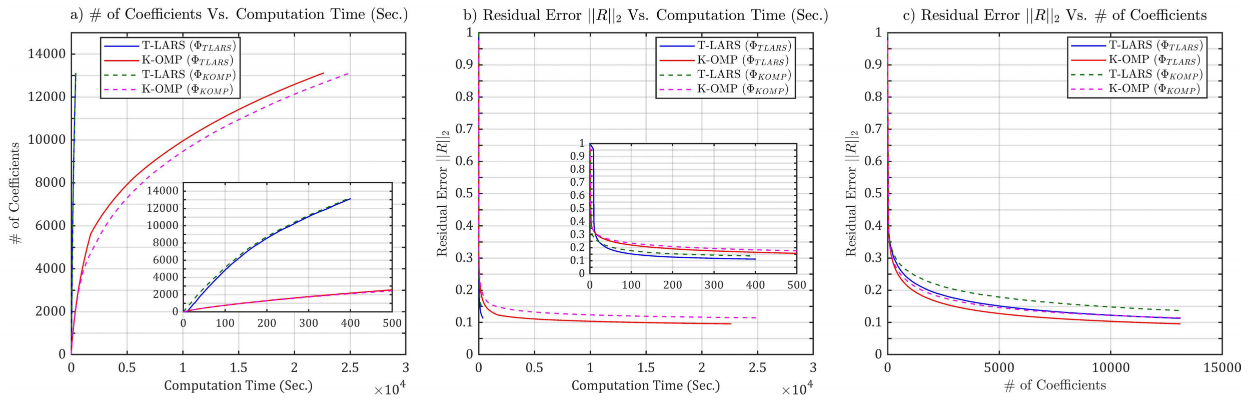

a) Original Image
b) K-OMP (Φ_{KOMP})
c) T-LARS (Φ_{KOMP})
d) K-OMP (Φ_{TLARS})
e) T-LARS (Φ_{TLARS})
f) K-OMP Difference (Φ_{KOMP})
g) T-LARS Difference(Φ_{KOMP})
h) K-OMP Difference (Φ_{TLARS})
i) T-LARS Difference (Φ_{TLARS})
255
200
150
100
50
0

Figure 3.4. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying Kronecker-OMP and T-LARS to our 3D MRI brain image and using our learned overcomplete dictionaries (Experiment 2 & 3)



Experiment 4: Figure 3.5 and Figure 3.6 show obtained experimental results for representing our 3D MRI brain image using three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{175\times351}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{150\times302}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times26}$, by solving the L_1 minimization problem, using T-LARS. The residual error of the reconstructed 3D image obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.121$ (12.1 %).

Figure 3.5. Original 3D MRI brain image (a), its reconstruction using 5% non-zero coefficients ($K = 13{,}125$) obtained by T-LARS using fixed *mode-n* overcomplete dictionaries (b), and the difference image (c) (Experiment 4)

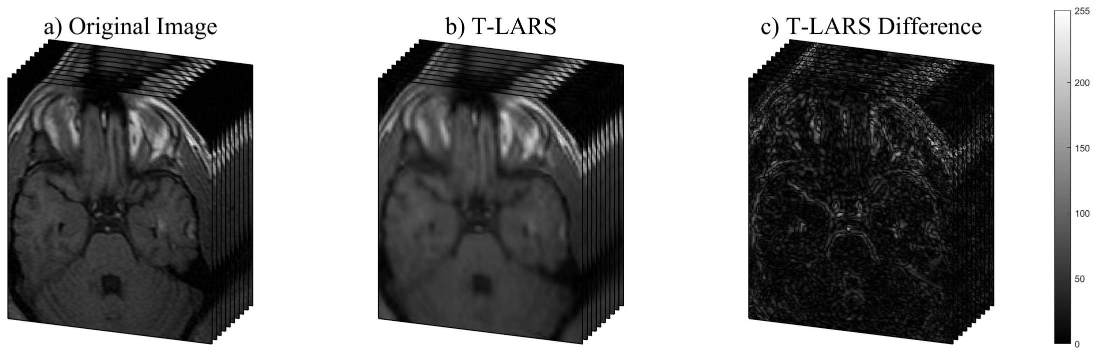
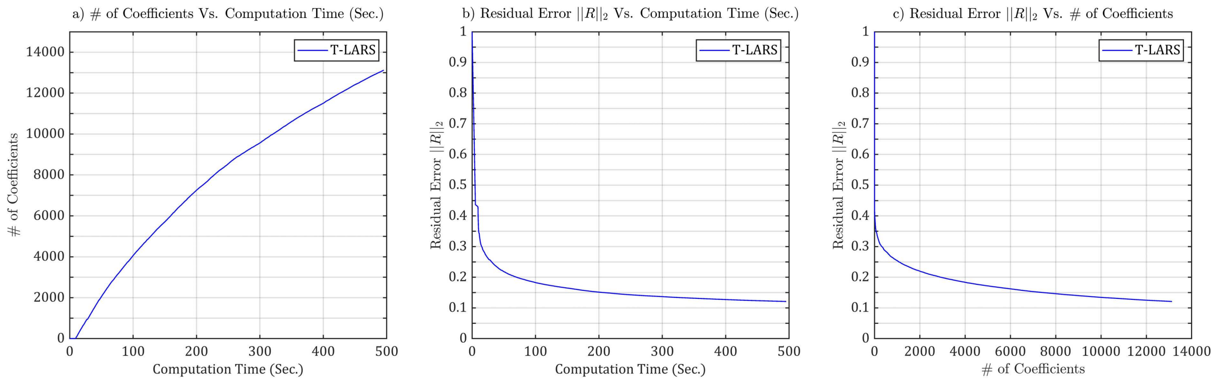


Figure 3.6. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying T-LARS to our 3D MRI brain image and using fixed *mode-n* overcomplete dictionaries (Experiment 4)



Experiment 5: Figure 3.7 and Figure 3.8 show obtained experimental results for representing our 3D MRI brain image using our learned over-complete dictionary, $\boldsymbol{\Phi}_{TLARS}$, by solving the L_1 minimization problem, using T-LARS. The residual error of the reconstructed 3D image was $\|\mathcal{R}\|_2 = 0.138$ (13.8 %).

Figure 3.7. Original 3D MRI brain image (a), and its reconstruction using 5% non-zero coefficients ($K = 13{,}125$) obtained by T-LARS using our learned over-complete dictionary (b), and the difference image (c) (Experiment 5).

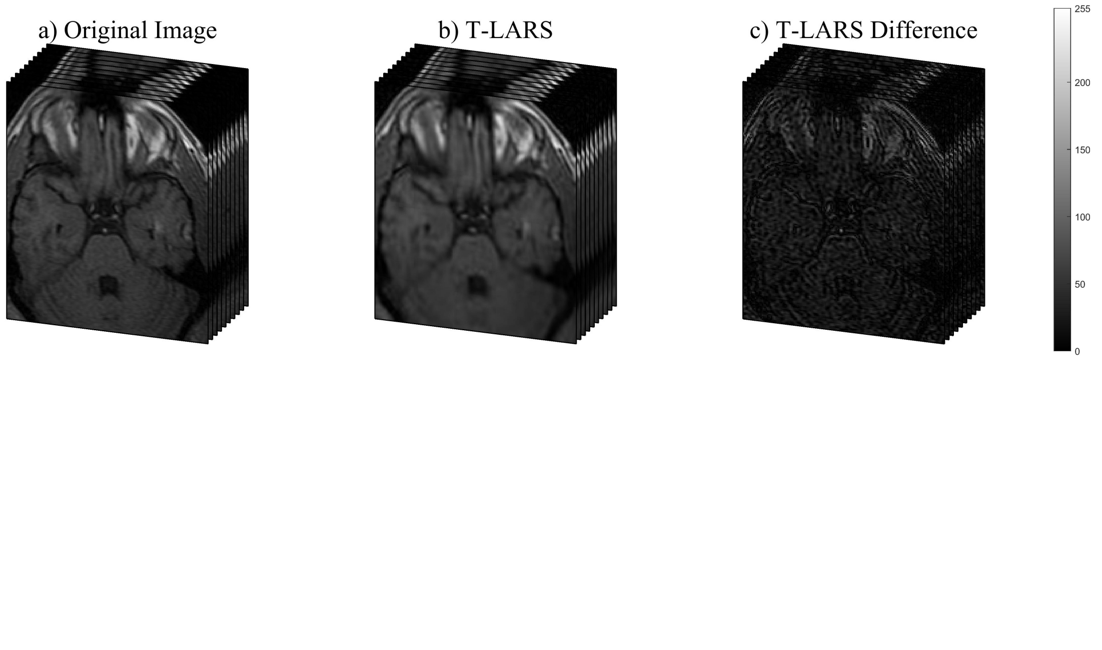
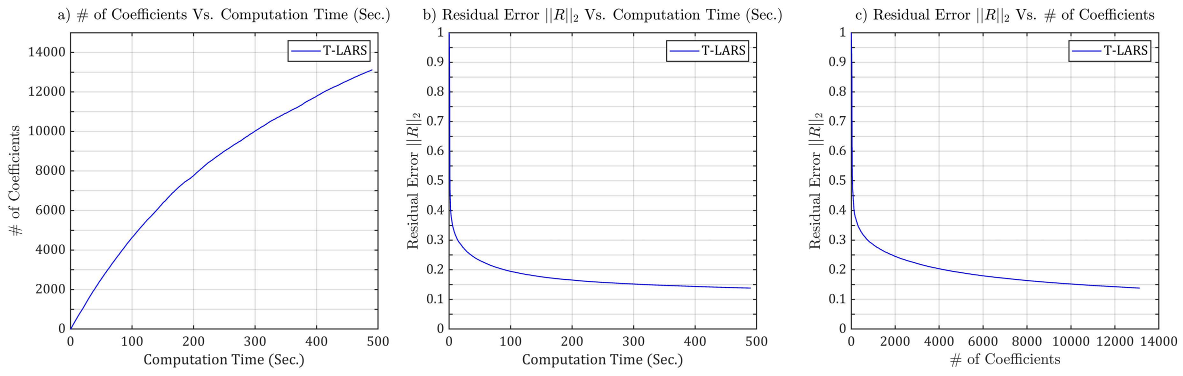


Figure 3.8. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. the number of non-zero coefficients, obtained by applying T-LARS to our 3D MRI brain image and using our learned overcomplete dictionary (Experiment 5).



3.5.4. Experimental Results for 3D PET-CT Brain Images

In this section, we compare the performance of T-LARS and Kronecker-OMP to obtain K-sparse representations of our 3D PET-CT brain image, $\mathcal{Y}$, $180 \times 160 \times 10$ voxels. by solving the L_0 constrained sparse tensor least-squares problem. We also obtained similar K-sparse representations using T-LARS by solving the L_1 optimization problem. Table 3.2 summarizes our results for the 1-5 experiments described in Section 5.2. In all experiments, the algorithms were stopped when the number of non-zero coefficients K reached 14,400, which is 5% of the number of elements in $\mathcal{Y}$. We note that in Table 2, the number of iterations for L_1 optimization problems is larger than K because, as shown in Algorithm 3.2, at each iteration, T-LARS could either add or remove non-zero coefficients to or from the solution.

Table 3.3. Summary of experimental results for our 3D PET-CT brain image

Experiment	Image Size	Optimization Problem	Dictionary Type	# of Iterations	Computation Time (sec) K-OMP	Computation Time (sec) T-LARS
1	180×160×10	L_0	Fixed	14,400	29,529	**505**
2	32×32×10×42	L_0	Learned ($\boldsymbol{\Phi}_{KOMP}$)	14,400	33,453	**476**
3	32×32×10×42	L_0	Learned ($\boldsymbol{\Phi}_{TLARS}$)	14,400	31,083	**490**

4	180×160×10	L_1	Fixed	16,059	-	**591**
5	32×32×10×42	L_1	Learned ($\boldsymbol{\Phi}_{TLARS}$)	18,995	–	**744**

Experiment 1: Figure 3.9 and Figure 3.10 show obtained experimental results for representing our 3D PET-CT brain image using three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{180\times 364}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{160\times 320}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times 26}$, by solving the L_0 minimization problem, using both T-LARS and Kronecker-OMP. The residual error of the reconstructed 3D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.054$ (5.4 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.0368$ (3.68 %).

Figure 3.9. Original PET-CT brain image (a), its reconstruction using 5% non-zero coefficients ($K = 14{,}400$) obtained by Kronecker-OMP (b) and T-LARS (c) using fixed *mode-n* overcomplete dictionaries (Experiment 1)

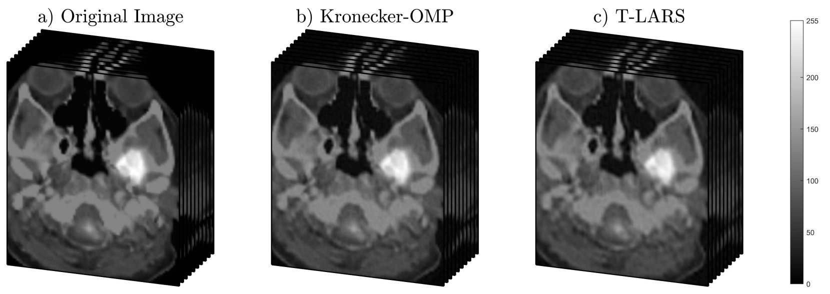
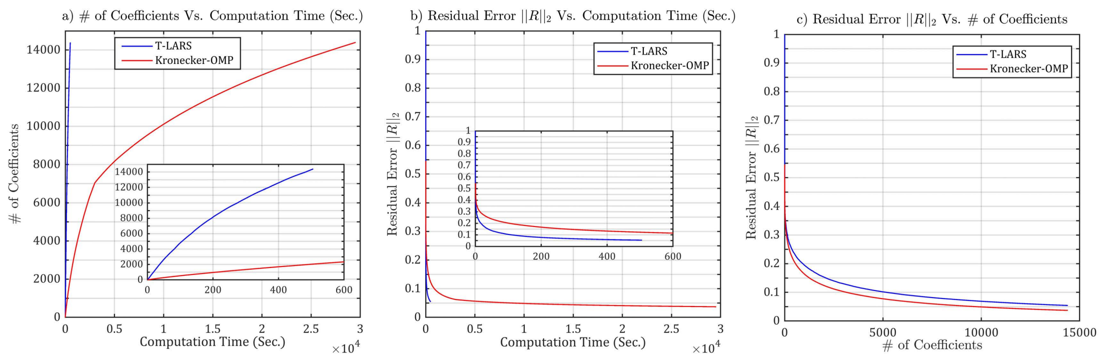


Figure 3.10. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying Kronecker-OMP and T-LARS to our 3D PET-CT brain image and using fixed *mode-n* overcomplete dictionaries (Experiment 1)



Experiment 2 & 3: Figure 3.11 and Figure 3.12 show obtained experimental results for representing our 3D PET-CT brain image using our learned overcomplete dictionaries, $\boldsymbol{\Phi}_{KOMP}$ and $\boldsymbol{\Phi}_{TLARS}$, by solving the L_0 minimization problem, using both T-LARS and Kronecker-OMP. For the $\boldsymbol{\Phi}_{KOMP}$ dictionary, the residual error of the reconstructed 3D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.096$ (9.6 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.077$ (7.7 %). For the $\boldsymbol{\Phi}_{TLARS}$ dictionaries, the normalized residual error of the reconstructed 3D images obtained using T-LARS was $\|\mathcal{R}\|_2 = 0.0877$ (8.77 %) and Kronecker-OMP was $\|\mathcal{R}\|_2 = 0.0722$ (7.22 %).

Figure 3.11. Original 3D PET-CT brain image (a), its reconstructions using 5% non-zero coefficients ($K = 14{,}400$), (b) - (e), the difference images, (f) - (i) obtained using Kronecker-OMP and T-LARS, using our learned overcomplete dictionaries (Experiment 2 & 3)

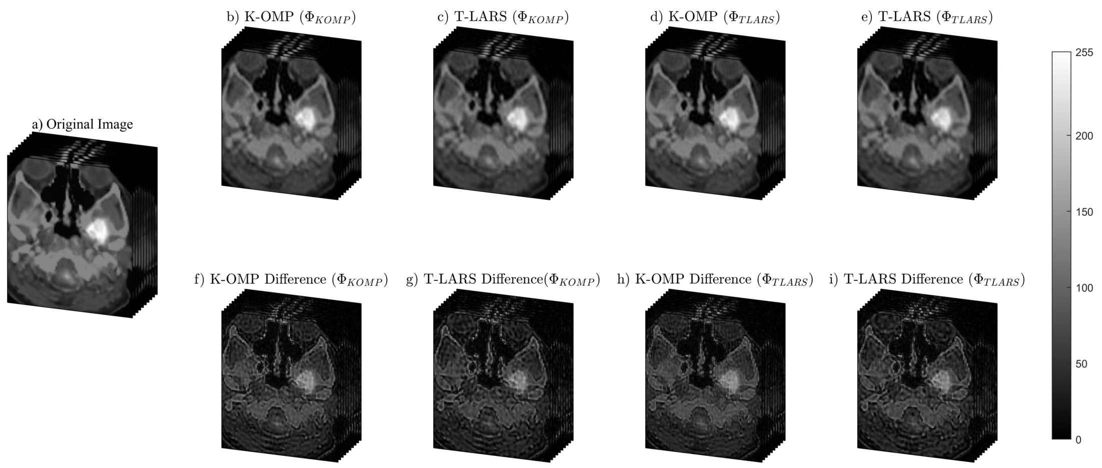
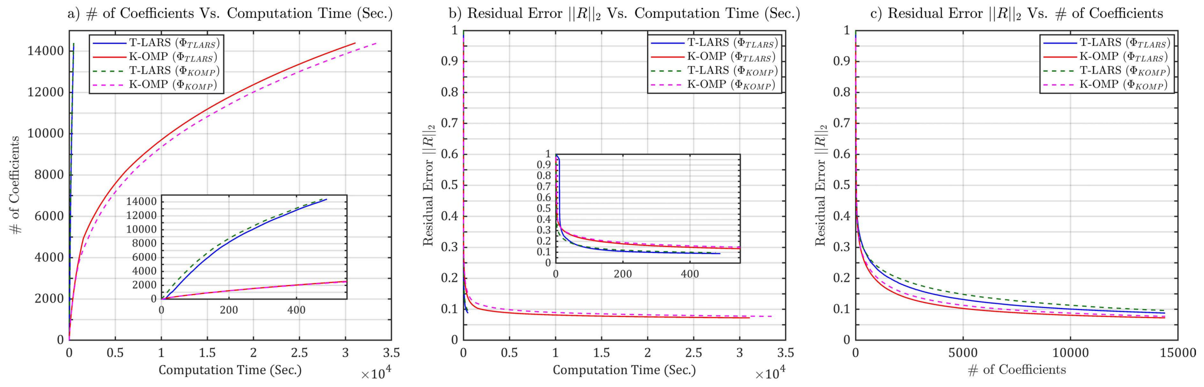

a) Original Image
b) K-OMP (Φ_{KOMP})
c) T-LARS (Φ_{KOMP})
d) K-OMP (Φ_{TLARS})
e) T-LARS (Φ_{TLARS})
f) K-OMP Difference (Φ_{KOMP})
g) T-LARS Difference(Φ_{KOMP})
h) K-OMP Difference (Φ_{TLARS})
i) T-LARS Difference (Φ_{TLARS})
255
200
150
100
50
0

Figure 3.12. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying Kronecker-OMP and T-LARS to our 3D PET-CT brain image and using our learned overcomplete dictionaries (Experiment 2 &3)



Experiment 4: Figure 3.13 and Figure 3.14 shows the experimental results for representing 3D PET-CT brain images using three fixed overcomplete *mode-n* dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{180\times364}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{160\times320}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times26}$, by solving the L_1 minimization problem, using T-LARS. The

residual error of the reconstructed 3D PET-CT brain images obtained using T-LARS is $\|\mathcal{R}\|_2 = 0.0838$ (8.38 %).

Figure 3.13. Original 3D PET-CT brain image (a), its reconstruction using 5% non-zero coefficients ($K = 14{,}400$) obtained by T-LARS using fixed *mode-n* overcomplete dictionaries (b), and the difference image (c) (Experiment 4)

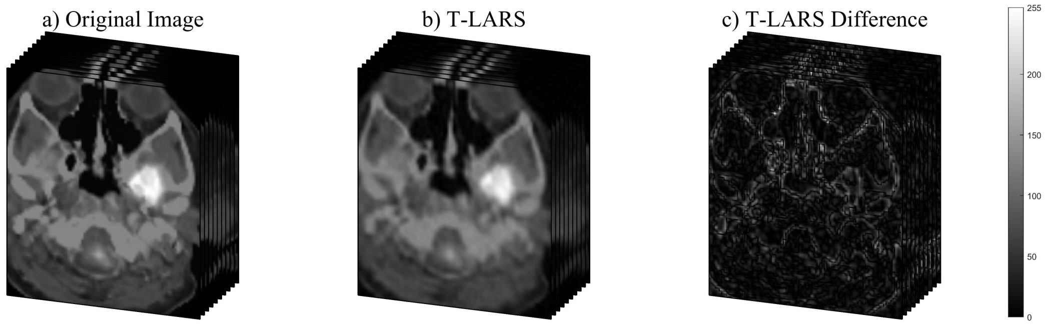
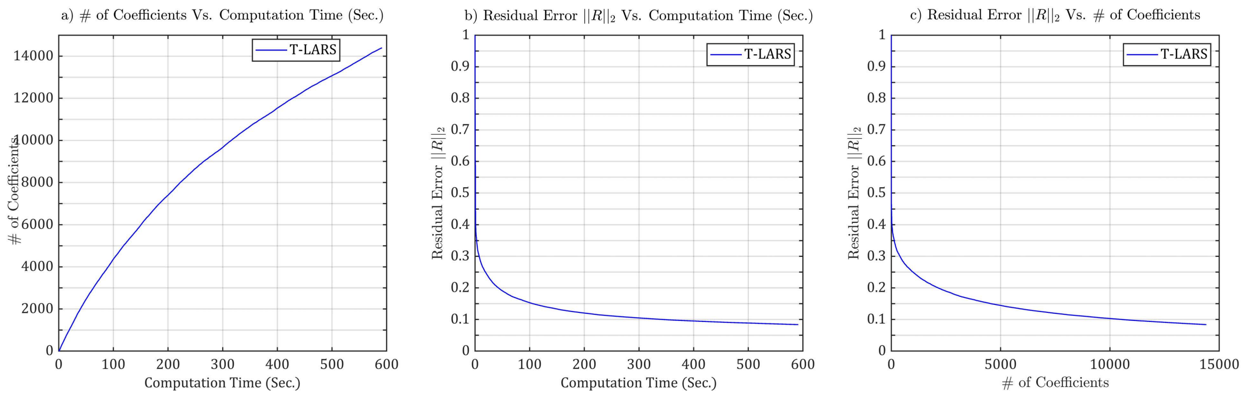


Figure 3.14. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying T-LARS to our 3D PET-CT brain image and using fixed *mode-n* overcomplete dictionaries (Experiment 4)



Experiment 5: Figure 3.15 and Figure 3.16 shows the experimental results for representing the 3D PET-CT brain images using our learned overcomplete dictionary, $\boldsymbol{\Phi}_{TLARS}$, by solving the L_1 minimization problem, using T-LARS. The residual error of the reconstructed 3D PET-CT brain images obtained using T-LARS is $\|\mathcal{R}\|_2 = 0.106$ (10.6 %).

Figure 3.15. Original 3D PET-CT brain image (a), and its reconstruction using 5% non-zero coefficients ($K = 14{,}400$) obtained by T-LARS using our learned overcomplete dictionary (b), and the difference image (c) (Experiment 5)

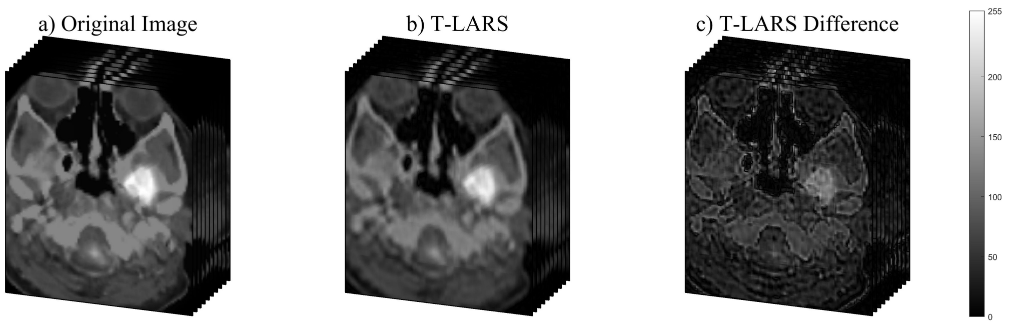
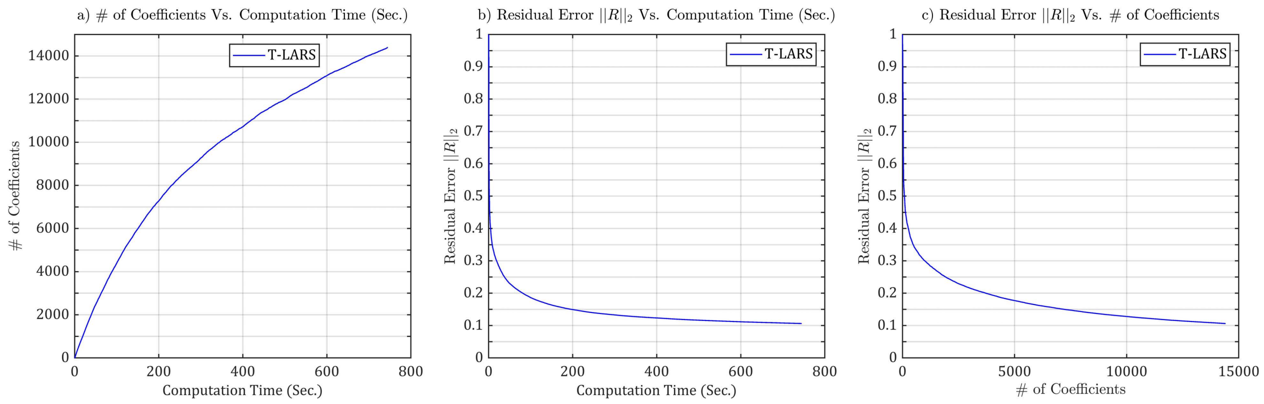


Figure 3.16. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. number of non-zero coefficients, obtained by applying T-LARS to our 3D PET-CT brain image and using our learned overcomplete dictionary (Experiment 5)



3.5.5. Experimental Results for Reconstructing 3D PET-CT Brain Images Using Compressed Sensing Samples

For our compressed sensing experiment, we obtained a tensor $\mathcal{A} \in \mathbb{R}^{150\times150\times10}$ having $150 \times 150 \times 10$ voxels from the 3D PET-CT brain images dataset. Then, we generated the compressed sensing samples tensor $\mathcal{Y} \in \mathbb{R}^{113\times113\times10}$ from the 3D PET-CT brain images tensor, $\mathcal{A} \in \mathbb{R}^{150\times150\times10}$, by projecting the tensor $\mathcal{A}$ using three *mode-n* Gaussian random sensing matrices, $\boldsymbol{Z}^{(1)} \in \mathbb{R}^{113\times150}$, $\boldsymbol{Z}^{(2)} \in \mathbb{R}^{113\times150}$ and $\boldsymbol{Z}^{(3)} \in \mathbb{R}^{10\times10}$. The resulting sampling ratio is $\frac{113\times113\times10}{150\times150\times10} = 0.5675$.

The used overcomplete *mode-n* dictionaries $\boldsymbol{D}^{(1)} \in \mathbb{R}^{150\times302}$, $\boldsymbol{D}^{(2)} \in \mathbb{R}^{150\times302}$ and $\boldsymbol{D}^{(3)} \in \mathbb{R}^{10\times10}$ were a union of a Symlet wavelet packet with four vanishing moments dictionary and a discrete cosine transform (DCT) dictionary.

We solved an L_0 minimization problem using Kronecker-OMP and T-LARS, and an L_1 minimization problem using T-LARS to recover the sparse tensor $\mathcal{X} \in \mathbb{R}^{113\times113\times10}$ using the compressed sensing samples tensor $\mathcal{Y} \in \mathbb{R}^{113\times113\times10}$ for three *mode-n* dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{113\times302}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{113\times302}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times10}$, where $\boldsymbol{\Phi}^{(n)} = \boldsymbol{Z}^{(n)}\boldsymbol{D}^{(n)}; \forall\, n \in \{1,2,\dots,N\}$.

Table 3.4 summarizes experiment results for reconstructing the 3D PET-CT image, $\mathcal{A} \in \mathbb{R}^{150\times150\times10}$, using compressed sensing samples tensor $\mathcal{Y} \in \mathbb{R}^{113\times113\times10}$ and three *mode-n* dictionaries, $\boldsymbol{\Phi}^{(1)}$, $\boldsymbol{\Phi}^{(2)}$ and $\boldsymbol{\Phi}^{(3)}$. The number of compressed sensing samples in the tensor $\mathcal{Y}$ is only 56.75% of the tensor signal $\mathcal{A}$.

In the experimental results shown in Table 3.4, we obtained a *K-Sparse* solution for the 3D PET-CT images, using $K = 15{,}323$ coefficients, which is only 12% of the elements in $\mathcal{Y}$. As shown in the Table 3.4, the L_1 minimization problem took more iterations compared to the L_0 minimization problems because T-LARS only adds columns to the active set *I* when solving the L_0 minimization problems, and T-LARS either adds or removes columns from the active set when solving the L_1 minimization problems.

Table 3.4. Summary of compressed sensing experimental results

Algorithm	Optimization Problem	#of Iterations	Residual Error	Computation Time (sec)
K-OMP	L_0	15,323	0.1144	28,045
T-LARS	L_0	15,323	0.1221	496
T-LARS	L_1	17,229	0.1217	612

Figure 3.17 and Figure 3.18 show experimental results for reconstructing the 3D PET-CT brain image $\mathcal{A}$ from the compressed sensing samples using T-LARS and Kronecker-OMP.

Figure 3.17. Original 3D PET-CT brain image (a), Reconstructed 3D PET-CT brain image using 12% non-zero coefficients ($K = 15{,}323$) obtained by solving an L_0 minimization problem using Kronecker-OMP (b) and T-LARS (c), and solving a L_1 minimization problem using T-LARS (d), and respective differences (e), (f), and (g) in our compressed sensing experiment.

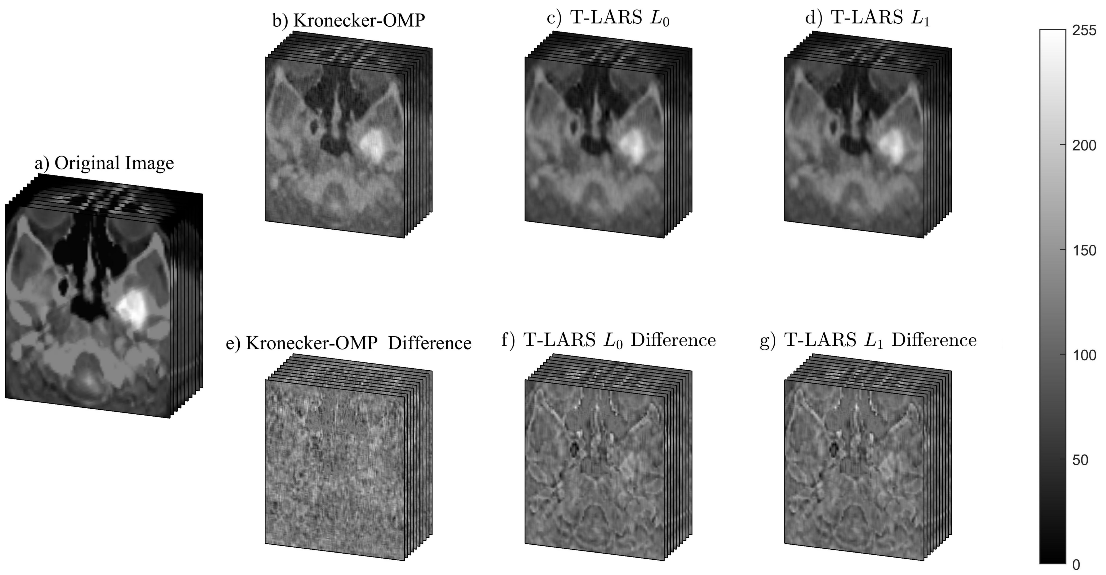
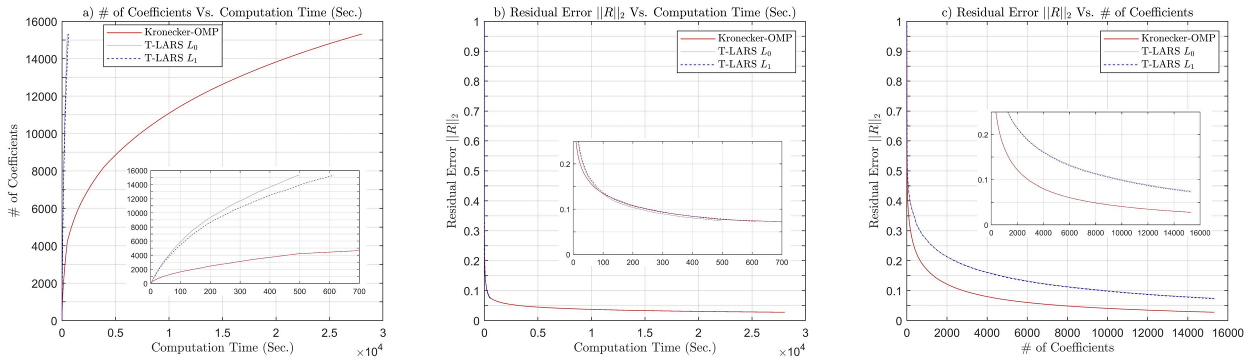

a) Original Image
b) Kronecker-OMP
c) T-LARS L_0
d) T-LARS L_1
e) Kronecker-OMP Difference
f) T-LARS L_0 Difference
g) T-LARS L_1 Difference
255
200
150
100
50
0

Figure 3.18. a) Number of non-zero coefficients vs. computation time; b) Residual error vs. computation time c) Residual error vs. the number of non-zero coefficients, for both Kronecker-OMP and T-LARS for our compressed sensing experiment.



3.6. Conclusions

In this chapter, we developed *Tensor Least Angle Regression* (T-LARS), a generalization of *Least Angle Regression*, to efficiently solve either large *L_0* or *L_1* constrained multi-dimensional (tensor) sparse least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ. An earlier generalization of OMP, known as *Kronecker-OMP*, has been developed to solve the L_0 problem for large multi-dimensional sparse least-squares problems. To demonstrate the validity and performance of our T-LARS algorithm, we used it to successfully obtain different K-sparse signal representations of two 3-D brain images, using fixed and learned separable over-complete dictionaries, by solving 3D and 4D, L_0 and L_1 constrained sparse least-squares problems. Our different numerical experiments demonstrate that our T-LARS algorithm is significantly faster (46 - 70 times) than Kronecker-OMP in obtaining K-sparse solutions for multilinear least-squares problems. However, the K-sparse solutions obtained using Kronecker-OMP always have a slightly lower residual error (1.55% - 2.25%) than ones obtained by T-LARS. These numerical results confirm our analysis in Section 3.4.2 that showed that the computational complexity of T-LARS is significantly lower than the computational complexity of *Kronecker-OMP*.

We also discussed the multilinear compressed sensing problem, and we compared Kronecker-OMP and T-LARS in reconstructing 3D PET-CT brain images, using compressed sensing samples and fixed *mode-n* over-complete dictionaries by solving 3D, L_0 and L_1 constrained multilinear least-squares problems. Our experimental results demonstrate that the T-LARS is 56 times faster than Kronecker-OMP in reconstructing the 3D PET-CT brain images using compressed sensing samples. Therefore, as future work, we plan to exploit this significant computational efficiency of T-LARS to develop more computationally efficient Kronecker dictionary learning methods.

Chapter 4

4. Weighted Tensor Least Angle Regression (WT-LARS)

Sparse weighted multilinear least-squares is a generalization of the sparse multilinear least-squares problem, where prior information about, e.g., parameters and data is incorporated by multiplying both sides of the original problem by a typically diagonal weights matrix [22]. If the diagonal weight matrix has a similar Kronecker structure to the dictionary matrix, we could use the *Tensor Least Angle Regression* (T-LARS) [18] algorithm developed in chapter 3 to solve this problem efficiently. Typically, introducing arbitrary diagonal weights would result in a non-Kronecker least-squares problem that could be very large to store or solve practically. In this chapter, we generalize the Tensor Least Angle Regression (T-LARS) algorithm developed in chapter 3 to efficiently solve either L_0 or L_1 constrained multilinear least-squares problems with arbitrary diagonal weights for all critical values of their regularization parameter. To demonstrate the validity of our new *Weighted Least Angle Regression* (WT-LARS) algorithm, we used it to successfully solve three different image inpainting problems by obtaining sparse representations of binary-weighted images.

4.1. Introduction

Weighted least squares is a generalization of the least-squares (LS) problem, where prior information about parameters and data is incorporated by multiplying both sides of the original LS problem by a typically diagonal weights matrix. Applications of weighted least-squares in Signal Processing include signal restoration [83], [84], source localization in wireless networks [85]–[87], adaptive filters [86], [88], [89], and image smoothing [90]. In Statistics, weighted least-squares regression is often used to reduce bias from non-informative data samples [91], [92]. Also, a best linear unbiased estimator (BLUE) is obtained by using the inverse of the data covariance matrix as the weights matrix [93].

Recently, sparsity has become a commonly desired characteristic of a least-squares solution [8], [9]. Because of its relatively small number of non-zero values, a sparse solution could result in faster processing with lower computer storage requirements [8], [9]. A sparse solution is usually obtained by solving a least-squares problem while minimizing either the L_0 norm of the solution (non-convex optimization problem) or minimizing the L_1 norm of the solution (convex optimization problem). Several methods have been proposed to solve sparse least-squares problems, including the *Method of Frames* [67], *Matching Pursuit* (MP) [12], *Orthogonal Matching Pursuit* (OMP) [13], *Best Orthogonal Basis* [68], *Least Absolute Shrinkage and Selection Operator* (LASSO) that is also known as *Basis Pursuit* [14], [15], and *Least Angle Regression* (LARS) [15]. Both MP and OMP solve the L_0 constrained least-squares problem [69] using sequential heuristic steps that add solution coefficients in a greedy, i.e., non-globally optimal, way. LASSO relaxes the non-convex L_0 constrained least-squares problem to solve the convex L_1 constrained least-squares problem instead [14]. Among the above solution methods, only *Least Angle Regression*(LARS) could efficiently solve both the L_0 and, with a slight modification, L_1 constrained least-squares problem for all critical values of their regularization parameters. This parameter is required to balance the minimization of the LS residual with the minimization of the norm of the solution [15].

In addition to incorporating *a priori* information, weights also could be introduced to sparse least-squares problems to improve the L_1 minimization problem results [94], [95]. Candès *et al.* also used a reweighted L_1 minimization approach to enhance sparsity in compressed sensing [96]. Also, weighted L_1 constrained least-squares regression has been used to extract information from large data sets for statistical applications [97], [98]. We note that sparse weighted least-squares problems could be solved using any of the above optimization methods.

Multilinear least-squares is a multidimensional generalization of least-squares [5], [7], [18], where the least-squares matrix has a Kronecker structure [16], [72]. Sparse multilinear least-squares could be either an L_0 constrained or an L_1 constrained multilinear least-squares problem. Caiafa and Cichocki introduced a generalization of OMP, *Kronecker-OMP*, to solve the L_0 constrained sparse multilinear least-squares problem [16]. Elrewainy and Sherif [17] developed *Kronecker Least Angle Regression* (*K*-LARS) to efficiently solve both L_0 and L_1 constrained sparse least-squares having a specific Kronecker matrix form, $\boldsymbol{A} \otimes \boldsymbol{I}$, for all critical values of the regularization parameter. To overcome this limitation, in chapter 3 we developed the Tensor Least angle Regression (T-LARS) [18], a generalization of K-LARS that does not require any special form of the LS matrix beyond being Kronecker. T-LARS solves either large L_0 or large L_1 constrained, sparse multilinear least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ with significantly lower computational complexity and memory usage than Kronecker-OMP.

Weighted multilinear least-squares is a generalization of multilinear least-squares that introduces a typically diagonal weight matrix to both sides of the original LS problem. Since an arbitrary diagonal weight matrix would not be Kronecker, the weighted LS matrix would lose its original Kronecker structure, resulting in a potentially very large non-Kronecker LS matrix. Thus, solving these weighted sparse multilinear least-squares problems could become highly impractical, as it would require significant memory and computational power.

Therefore, in this chapter, we extend T-LARS to Weighted Tensor Least Angle Regression (WT-LARS) that could solve both L_0 and L_1 constrained sparse weighted multilinear least-squares problems efficiently for all critical values of the regularization parameter.

Weighted multilinear least-squares problems could be used to include prior information about parameters and tensor data to a multilinear least-squares problem. Therefore, WT-LARS could be used to solve multidimensional counterparts of applications of weighted least squares such as tensor signal restoration, video smoothing, and reduce bias in multilinear regression applications. In the experimental results, we used WT-LARS to solve image inpainting problems successfully using binary-weighted images.

This chapter is organized as follows: Section 4.2 includes a brief introduction to the sparse weighted tensor least-squares problem. Section 4.3 describes our Weighted Tensor Least Angle Regression (WT-LARS) algorithm in detail. Section 4.4 provides results of applying WT-LARS to solve three different image inpainting problems by obtaining sparse representations of binary-weighted RGB images. We present our conclusions in Section 4.5.

4.2. Problem Formulation

4.2.1. Sparse weighted tensor least-squares problem

A multilinear transformation of a tensor $\mathcal{X}$ could be defined as, $\mathcal{Y} = \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}$, where $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ and $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ are *N^th^* order tensors, with the equivalent vectorized form

$$\boldsymbol{\Phi}\text{vec}(\mathcal{X}) = \text{vec}(\mathcal{Y}) \tag{4.1}$$

Where $\boldsymbol{\Phi} \in \mathbb{R}^{J \times I}$, and $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$.

Let $\boldsymbol{W} = \boldsymbol{S}^H \boldsymbol{S}$, be a diagonal weight matrix. We could obtain a weighted linear transformation [22] of (4.1) as

$$\boldsymbol{S\Phi}\text{vec}(\mathcal{X}) = \boldsymbol{S}\text{vec}(\mathcal{Y}) \tag{4.2}$$

A sparse solution of the weighted linear system in (4.2) could be obtained by solving an L_p (*p = 0 or p = 1*) minimization problem,

$$\tilde{\mathcal{X}} = \arg\min_{\mathcal{X}} \|\boldsymbol{S\Phi}\text{vec}(\mathcal{X}) - \boldsymbol{S}\text{vec}(\mathcal{Y})\|_2^2 + \lambda \|\text{vec}(\mathcal{X})\|_p \tag{4.3}$$

If $\boldsymbol{S}$ is a Kronecker matrix, then $\boldsymbol{S\Phi} = \left(\boldsymbol{S}^{(N)}\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{S}^{(1)}\boldsymbol{\Phi}^{(1)}\right)$ and we could use T-LARS [18] developed in chapter 3, to obtain a sparse solution for either L_0 or L_1 optimization problem in (4.3) efficiently. However, $\boldsymbol{S}$ is not typically Kronecker, so $\boldsymbol{S\Phi}$ would not have a Kronecker structure, and (4.3) should be solved as a potentially very large vectorized (one-dimensional) sparse least-squares problem which could be very challenging in terms of memory and computational power requirements. Therefore, in this chapter, we develop Weighted Tensor Least Angle Regression (WT-LARS), a computationally efficient method, to solve either L_0 or L_1 constrained sparse weighted multilinear least-squares problems in (4.3) for an arbitrary diagonal weights matrix $\boldsymbol{W} = \boldsymbol{S}^H \boldsymbol{S} \in \mathbb{R}^{J \times J}$.

4.2.2. Calculating the mutual coherence of a large weighted Kronecker dictionary

The mutual coherence μ [14], [99] is an important parameter for analyzing the uniqueness and the accuracy of a *K-sparse* solution. The mutual coherence for a weighted dictionary $\boldsymbol{S\Phi}$ is given by,

$$\mu = \max_{i \neq j} \left| (\boldsymbol{\Phi}^{\boldsymbol{H}} \boldsymbol{W} \boldsymbol{\Phi})_{ij} \right| \tag{4.4}$$

Where $\boldsymbol{W} = \boldsymbol{S}^H \boldsymbol{S}$.

For an arbitrary diagonal weights matrix $\boldsymbol{W}$, the matrix $\boldsymbol{\Phi}^{\boldsymbol{H}} \boldsymbol{W} \boldsymbol{\Phi}$ does not have a Kronecker structure. Therefore, calculating μ for a large dictionary matrix $\boldsymbol{S\Phi}$ by constructing the matrix $\boldsymbol{\Phi}^{\boldsymbol{H}} \boldsymbol{W} \boldsymbol{\Phi}$ is computationally infeasible. However, we could efficiently calculate the mutual coherence using,

$$\mu = \max_{i \neq j; 1 \leq j \leq J} \left| \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)} \right)^{\boldsymbol{H}} \boldsymbol{W} \boldsymbol{\Phi}_j \right| \tag{4.5}$$

Where $\boldsymbol{\Phi}_j$ is the jth column of the Kronecker matrix $\boldsymbol{\Phi}$ and we could efficiently calculate each column vector $\left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)} \right)^{\boldsymbol{H}} \boldsymbol{W} \boldsymbol{\Phi}_j$ using the full multilinear product between the *mode-n* matrices $\boldsymbol{\Phi}^{(n)}$ and the column vector $\boldsymbol{W}\boldsymbol{\Phi}_j$.

4.3. Weighted Tensor Least Angle Regression (WT-LARS)

In this section, we develop the Weighted Tensor Least Angle Regression (WT-LARS) by extending Tensor Least Angle Regression (T-LARS) to solve the sparse tensor least-squares problem in (4.2) for weights $\boldsymbol{W} = \boldsymbol{S}^H \boldsymbol{S}$ and Kronecker dictionaries $\boldsymbol{\Phi}$.

Inputs to WT-LARS are the data tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$, *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$, the diagonal weight matrix $\boldsymbol{W} = \boldsymbol{S}^H \boldsymbol{S}$, and the stopping criterion as a residual tolerance ε or the maximum number of non-zero coefficients K (K-sparse representation). The output is the solution tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$.

WT-LARS requires weighted data $\boldsymbol{S}\mathrm{vec}(\mathcal{Y})$, and columns of the weighted dictionary $\boldsymbol{S\Phi}$ to have a unit L_2 norm. Normalized weighted data could be easily calculated by $\mathcal{Y}_W = \boldsymbol{S}\mathrm{vec}(\mathcal{Y}) / \|\boldsymbol{S}\mathrm{vec}(\mathcal{Y})\|_2$. However, the dictionary matrix $\boldsymbol{S\Phi}$ does not have a Kronecker structure. Hence, normalizing *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}$ does not ensure normalization of the columns of $\boldsymbol{S\Phi}$. Therefore, in WT-LARS, we use the normalized weighted dictionary matrix $\boldsymbol{\Phi}_W = \boldsymbol{S\Phi Q}$ instead of the normalized dictionary matrix $\boldsymbol{\Phi}$ in T-LARS, where $\boldsymbol{Q}$ is a diagonal matrix,

$$\boldsymbol{Q}_{i,i} = \frac{1}{\|(\boldsymbol{S\Phi})_i\|_2} \tag{4.6}$$

Where $(\boldsymbol{S\Phi})_i$ is the i^{th} column of the weighted dictionary matrix $\boldsymbol{S\Phi}$. We can efficiently calculate the diagonal matrix $\boldsymbol{Q}$ as,

$$diag(\boldsymbol{Q}) = \mathbf{1}./\sqrt{(\boldsymbol{\Phi}^{*2})^{\boldsymbol{T}}\, diag(\boldsymbol{W})} \tag{4.7}$$

Where, $\boldsymbol{\Phi}^{*2}$ [100] denotes the Hadamard square of $\boldsymbol{\Phi}$, such that $\boldsymbol{\Phi}_{i,j}^{*2} = \left(\boldsymbol{\Phi}_{i,j}\right)^2$, "./" denotes elementwise division, and $diag(\boldsymbol{Q})$ and $diag(\boldsymbol{W})$ are diagonal vectors of $\boldsymbol{Q}$ and $\boldsymbol{W}$ respectively. We could efficiently calculate $(\boldsymbol{\Phi}^{*2})^{\boldsymbol{T}} diag(\boldsymbol{W})$ using the full multilinear product.

WT-LARS solves the L_0 or L_1 constrained minimization problems in (4.3) for all critical values of the regularization parameter λ. WT-LARS starts with a large value of λ that results in an empty active set $I = \{\}$, and a solution $\widetilde{\mathcal{X}}_{t=0} = 0$. The set I denotes an active set of columns of the dictionary $\boldsymbol{\Phi}_W$, i.e., column indices where the optimal solution $\widetilde{\mathcal{X}}_t$ at iteration t, is nonzero, and I^c denotes its corresponding inactive set. Therefore, $\boldsymbol{\Phi}_{W_I}$ contains only the active columns of the dictionary $\boldsymbol{\Phi}_W$ and $\boldsymbol{\Phi}_{W_{I^c}}$ contains only its inactive columns.

At each iteration t, a new column is either added (L_0) to the active set I or a new column is either added or removed (L_1) from the active set I, and λ is reduced by a calculated value δ_t^*.

As a result of such iterations, new solutions with an increased number of coefficients that follow a piecewise linear path are obtained until a predetermined residual error ε or a predetermined number of active columns K is obtained.

The regularization parameter λ is initialized to the maximum of the correlation $\boldsymbol{c}_1$, between the columns of $\boldsymbol{\Phi}_W$ and the initial residual $\boldsymbol{r}_0 = \text{vec}(\mathcal{Y})$.

$$\boldsymbol{c}_1 = \boldsymbol{\Phi}_W^T \boldsymbol{r}_0 \tag{4.8}$$

Since $\boldsymbol{\Phi}_W^T = \boldsymbol{Q}\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{S}$, we can easily calculate $\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{S}\boldsymbol{r}_0$ using the full multilinear product as

$$\acute{\mathcal{C}}_1 = \mathcal{R}_{\boldsymbol{S}_0} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)^T} \tag{4.9}$$

where $\text{vec}\left(\mathcal{R}_{\boldsymbol{S}_0}\right) = \boldsymbol{S}\boldsymbol{r}_0$ and $\boldsymbol{c}_1 = \boldsymbol{Q}\text{vec}\left(\acute{\mathcal{C}}_1\right)$. The column index corresponding to the maximum correlation $\boldsymbol{c}_1$ is added to the active set. For a given active set I, the optimal solution $\widetilde{\mathcal{X}}_t$ at any iteration t, could be written as

$$\mathrm{vec}(\tilde{\mathcal{X}}_t) = \begin{cases} \left(\boldsymbol{\Phi}_{W_{I_t}}^{\ T}\boldsymbol{\Phi}_{W_{I_t}}\right)^{-1}\left(\boldsymbol{\Phi}_{W_{I_t}}^{\ T}\mathrm{vec}(\mathcal{Y}) - \lambda_t \boldsymbol{z}_t\right), \text{on } I \\ 0, \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{Otherwise} \end{cases} \tag{4.10}$$

where, $\boldsymbol{z}_t$ is the sign sequence of $\boldsymbol{c}_t$ on the active set I, and $\boldsymbol{c}_t = \boldsymbol{\Phi}_W^T \boldsymbol{r}_{t-1}$ is the correlation vector of all columns of the dictionary $\boldsymbol{\Phi}_W$ with the residual $\boldsymbol{r}_{t-1}$ at any iteration t.

The optimal solution at any iteration, t must satisfy the following two optimality conditions,

$$\boldsymbol{\Phi}_{W_{I_t}}^{\ T}\boldsymbol{r}_t = -\lambda_t \boldsymbol{z}_t \tag{4.11}$$

$$\left\| \boldsymbol{\Phi}_{W_{I_t^c}}^{\ T}\boldsymbol{r}_t \right\|_\infty \le \lambda_t \tag{4.12}$$

where, $\boldsymbol{r}_t = \mathrm{vec}(\mathcal{Y}) - \boldsymbol{\Phi}_W \mathrm{vec}(\tilde{\mathcal{X}}_t)$ is the residual at iteration t, and $\boldsymbol{z}_t$ is the sign sequence of the correlation $\boldsymbol{c}_t$ at iteration t, on the active set I. The condition in (4.11) ensures that the magnitude of the correlation between all active columns of $\boldsymbol{\Phi}_W$ and the residual is equal to $|\lambda_t|$ at each iteration, and the condition in (4.12) ensures that the magnitude of the correlation between the inactive columns of $\boldsymbol{\Phi}_W$ and the residual is less than or equal to $|\lambda_t|$.

At each iteration t, λ_t is reduced by a small step size δ_t^*, until a condition in either (4.11) or (4.12) violates. For L_0, and L_1 constrained minimization problems, if an inactive column violates the condition (4.12), it is added to the active set, and for L_1 constrained minimization problems, if an active column violates the condition (4.11), it is removed from the active set.

As λ is reduced by δ_t^*, the solution $\tilde{\mathcal{X}}_t$ change by $\delta_t^* \boldsymbol{d}_t$ along a direction $\boldsymbol{d}_t$, where $\boldsymbol{d}_{I_t^c} = 0$ and $\boldsymbol{d}_{I_t} = \boldsymbol{G}_t^{-1}\boldsymbol{z}_t$, and $\boldsymbol{G}_t^{-1}$ is the inverse of the Gram matrix of the active columns of the dictionary $\boldsymbol{G}_t = \boldsymbol{\Phi}_{W_{I_t}}^{\ T}\boldsymbol{\Phi}_{W_{I_t}}$.

The size of this Gram matrix would either increase (dictionary column addition) or decrease (dictionary column removal) with each iteration t. Therefore, for computational efficiency, we use the *Schur complement* inversion formula to calculate $\boldsymbol{G}_t^{-1}$ from $\boldsymbol{G}_{t-1}^{-1}$, thereby avoiding its full calculation [18], [101]. See Appendix C.1 for updating the inverse of the Gram matrix using the *Schur complement* inversion formula.

The smallest step size for L_1 constrained sparse least-squares problem $\delta_t^* = \min\{\delta_t^+, \delta_t^-\}$ is the minimum of δ_t^+, minimum step size for adding a column, and δ_t^-, minimum step size for removing a column. The minimum step size for removing a column from the active set is given by,

$$\delta_t^- = \min_{i\in I}\left\{-\frac{\boldsymbol{x}_{t-1}(i)}{\boldsymbol{d}_t(i)}\right\} \tag{4.13}$$

The minimum step size for adding a new column to the active set is given by,

$$\delta_t^+ = \min_{i\in I^c}\left\{\frac{\lambda_t - \boldsymbol{c}_t(i)}{1-\boldsymbol{v}_t(i)}, \frac{\lambda_t + \boldsymbol{c}_t(i)}{1+\boldsymbol{v}_t(i)}\right\} \tag{4.14}$$

where

$$\boldsymbol{v}_t = \boldsymbol{\Phi}_W^T \boldsymbol{\Phi}_W \boldsymbol{d}_t \tag{4.15}$$

Since $\boldsymbol{\Phi}_W = \boldsymbol{S\Phi Q}$ **,** We can efficiently calculate $\boldsymbol{v}_t$ using two full multilinear products.

Let $\boldsymbol{v}_t = \boldsymbol{Q}\text{vec}(\acute{\mathcal{V}}_t)$, where

$$\acute{\mathcal{V}}_t = \mathcal{U}_{w_t} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \dots \times_N \boldsymbol{\Phi}^{(N)^T} \tag{4.16}$$

And $\text{vec}(\mathcal{U}_{w_t}) = \boldsymbol{W}\text{vec}(\acute{\mathcal{D}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \dots \times_N \boldsymbol{\Phi}^{(N)})$, and $\text{vec}(\acute{\mathcal{D}}_t) = \boldsymbol{Q}\boldsymbol{d}_t$.

The residual $\boldsymbol{r}_t$ is calculated at the end of each iteration using,

$$\boldsymbol{r}_t = \boldsymbol{r}_{t-1} - \delta_t^* \boldsymbol{\Phi}_W \boldsymbol{d}_t \tag{4.17}$$

We can efficiently calculate $\boldsymbol{\Phi}_W \boldsymbol{d}_t$ as

$$\boldsymbol{\Phi}_W \boldsymbol{d}_t = \boldsymbol{S}\text{vec}(\acute{\mathcal{D}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}) \tag{4.18}$$

WT-LARS stops at a predetermined residual error $\|\boldsymbol{r}_t\|_2 \le \varepsilon$ or when a predetermined number of active columns K is obtained. The residual error $\|\boldsymbol{r}_t\|_2 = \left\|\text{vec}(\mathcal{Y}) - \boldsymbol{\Phi}_W \text{vec}(\widetilde{\mathcal{X}}_t)\right\|_2$ is the L_2 norm of the residual error after the iteration *t*, where $0 \le \|\boldsymbol{r}_t\|_2 \le 1$ for a normalized $\mathcal{Y}$ and column normalized weighted dictionary $\boldsymbol{\Phi}_{W_{I_t}}$.

4.3.1. Weighted Tensor Least Angle Regression Algorithm

Algorithm 4.1: Weighted Tensor Least Angle Regression (WT-LARS)

Input: *WT-LARS_mode* $= L_1$ *or* L_0, *normalized tensor* $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_n\times\ldots\times J_N}$; *Mode-n dictionary matrices* $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n\times I_n}$; $n \in \{1,..N\}$; *Diagonal Weights Matrix* $\boldsymbol{W} \in \mathbb{R}^{(J_1\times\ldots\times J_N)\times(J_1\times\ldots\times J_N)}$; *Stopping criterion: residual tolerance:* ε *or number of non-zero coefficients:* K

Initialization: $\boldsymbol{S} = \sqrt{\boldsymbol{W}}$, *Residual:* $\boldsymbol{r}_0 = \boldsymbol{S}vec(\mathcal{Y})$; $\boldsymbol{x}_0 = 0$; *active set:* $I = \{\}$;

```
1.  $diag(\boldsymbol{Q}) = \mathbf{1}./\sqrt{(\boldsymbol{\Phi}^2)^T\, diag(\boldsymbol{W})}$
2.  $\mathrm{vec}(\mathcal{R}_{\boldsymbol{S}_0}) = \boldsymbol{S}\boldsymbol{r}_0$
3.  $\mathcal{C}_1 = \mathcal{R}_{\boldsymbol{S}_0} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \ldots \times_N \boldsymbol{\Phi}^{(N)^T}$
4.  $\boldsymbol{c}_1 = \boldsymbol{Q}\mathrm{vec}(\mathcal{C}_1)$
5.  $[\lambda_1, column\_idx\,] = max(\boldsymbol{c}_1)$
6.  $I = \{column\_idx\}$
7.  while $(\|\boldsymbol{r}_{t-1}\|_2 < \varepsilon$ or $length(I) < K)$
8.      $\boldsymbol{z}_t = sign\,(\boldsymbol{c}_t(I))$
9.      $\boldsymbol{G}_t^{-1}$ = updateWeightedInverseGramMatrix($\boldsymbol{G}_{t-1}^{-1}$, $\boldsymbol{W}$, $\boldsymbol{Q}$, $\{\boldsymbol{\Phi}^{(1)}, \ldots, \boldsymbol{\Phi}^{(N)}\}$, I, add_column,
        column_idx) % See Appendix C.1
10.     $\boldsymbol{d}_{I_t} = \boldsymbol{G}_t^{-1}\boldsymbol{z}_t$
11.     $\mathrm{vec}(\acute{\mathcal{D}}_t) = \boldsymbol{Q}\boldsymbol{d}_t$
12.     $\mathcal{U}_t = \acute{\mathcal{D}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \ldots \times_N \boldsymbol{\Phi}^{(N)}$
13.     $\mathrm{vec}(\mathcal{U}_{w_t}) = \boldsymbol{W}\mathrm{vec}(\mathcal{U}_t)$
14.     $\mathcal{V}_t = \mathcal{U}_{w_t} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \ldots \times_N \boldsymbol{\Phi}^{(N)^T}$
15.     $\boldsymbol{v}_t = \boldsymbol{Q}vec(\mathcal{V}_t)$
16.     $\delta_{t\ 1}^{+} = (\lambda_t - \boldsymbol{c}_t(I^c))./(1 - \boldsymbol{v}_t(I^c))$     % "./" - Elementwise division
17.     $\delta_{t\ 2}^{+} = (\lambda_t + \boldsymbol{c}_t(I^c))./(1 + \boldsymbol{v}_t(I^c))$
18.     $\delta_t^- = -\boldsymbol{x}_{t-1}./\,\boldsymbol{d}_t$
19.     $[\delta_t^*,\ column\_idx] = min\,(\delta_{t\ 1}^{+}, \delta_{t\ 2}^{+})$
20.     add_column == True
21.     If WT-LARS_mode == $L_1$ && min $(\delta_t^-) < \delta_t^*$
22.         $[\delta_t^*,\ column\_idx] = min\,(\delta_t^-)$
23.         add_column = False
24.     end
25.     $\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \delta_t^*\boldsymbol{d}_t$
26.     $\lambda_{t+1} = \lambda_t - \delta_t^*$
27.     $\boldsymbol{c}_{t+1} = \boldsymbol{c}_t - \delta_t^*\boldsymbol{v}_t$
28.     $\acute{\mathcal{R}}_t = \acute{\mathcal{D}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}$
29.     $\boldsymbol{r}_t = \boldsymbol{r}_{t-1} - \delta_t^*\boldsymbol{S}\mathrm{vec}(\acute{\mathcal{R}}_t)$
30.     if add_column == True
31.         $I = I + \{column\_idx\}$
32.     else
```

```
33.         $I = I - \{column\_idx\}$
34.     **end**
35. **end while**
36. **return** $I, \boldsymbol{x}$
```

4.4. Experimental Results

In this section, we present experimental results for WT-LARS using inpainting as an example. For experiments shown in Figure 4.1 and Figure 4.2, we obtained fenced images from the Image datasets for MSBP deformable lattice detection Algorithm [102], and for the experiment shown in Figure 4.3, we obtained a landscape image from the DIV2K dataset [103].

Our experimental results were obtained using a MATLAB implementation of T-LARS and Kronecker-OMP on an MS-Windows machine: 2 Intel Xeon CPUs E5-2637 v4, 3.5GHz, 32GB RAM, and NVIDIA Tesla P100 GPU with 12GB memory.

4.4.1. Inpainting Experiment

In this experiment, we use WT-LARS for inpainting. After applying zero weights to the missing data, we obtained a sparse representation of the inpainted image using WT-LARS.

In our experimental results shown in Figure 4.1 and Figure 4.2, we obtained a fenceless image by considering pixels behind the fences as missing data. Figure 4.1 a) and Figure 4.2 a) show the original images with fences, and Figure 4.1 b) and Figure 4.2 b) show the respective masks applied to each pixel of the original image, where black indicates zero and white indicate one. Figure 4.1 c) and Figure 4.2 c) show the reconstructed fenceless images using the sparse representation of images behind fences obtained by WT-LARS.

We obtained RGB image patches, $200 \times 200 \times 3$ pixels, from the original images in Figure 4.1 a) and Figure 4.2 a). For each patch, we obtained a weighted *K*-sparse representation using WT-LARS, with 10% nonzero coefficients, for three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{200\times400}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{200\times400}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{3\times4}$, by solving a L_1 constrained sparse weighted least-squares problem. Weights consists of zeros for the pixels that belong to the fence in the original images and ones for everywhere else. Used fixed *mode-n* overcomplete dictionaries were a union of a Discrete Cosine Transform (DCT) dictionaries and a Symlet wavelet packet with four vanishing moments dictionaries. In the experimental results shown in Figure 4.1 and Figure 4.2, the RGB patches with the minimum number of nonzero samples had 79,834 and 92,748 nonzero samples, respectively.

Figure 4.1. a) Original image with a fence b) Weights image with zero weights for the fence c) WT-LARS reconstructed image (Fence Removed)

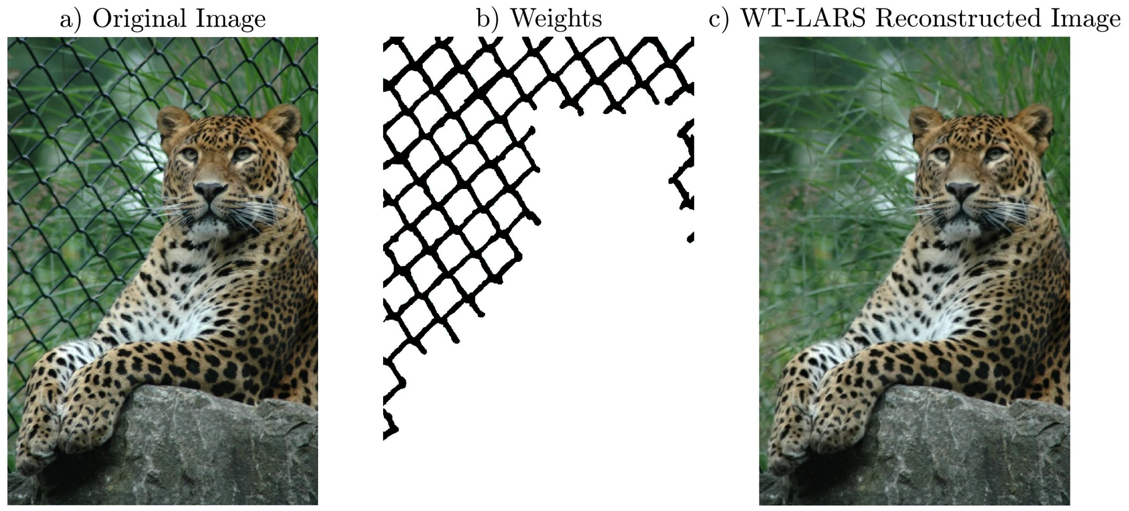
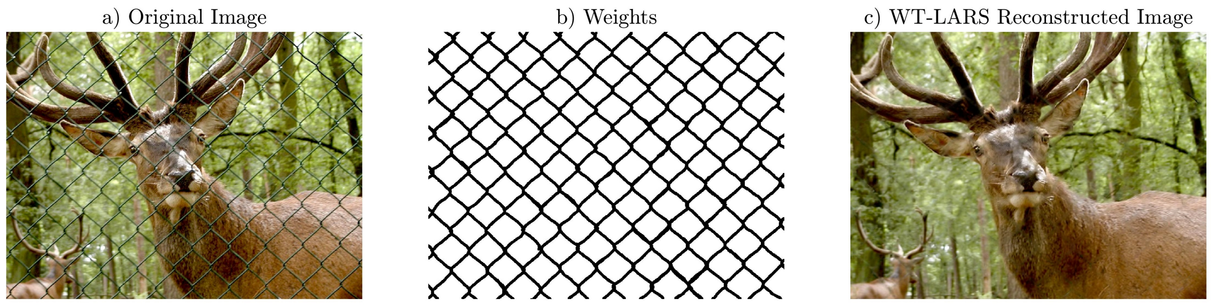


Figure 4.2. a) Original image with a fence b) Weights image with zero weights for the fence c) WT-LARS reconstructed image (Fence Removed)

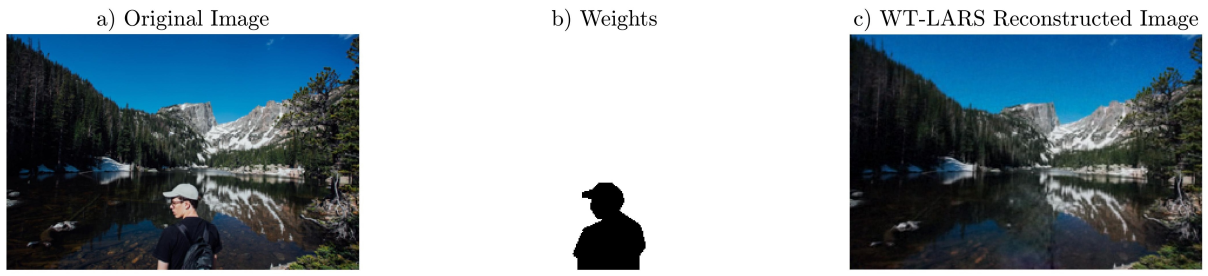


In the experimental results shown in Figure 4.3, we used WT-LARS to obtain a landscape image occluded by a person in Figure 4.3 a). Figure 4.3 b) shows the weights, and Figure 4.3 c) shows the inpainting result after removing the person from the foreground of the landscape image.

The RGB images in Figure 4.3 a) are a scaled version of the original image with $200 \times 300 \times 3$ pixels. We obtained a weighted K-sparse representation for the scaled image in Figure 4.3 a) using WT-LARS, with 20% non-zero coefficients, for three fixed *mode-n* overcomplete dictionaries,

$\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{200\times400}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{300\times604}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{3\times4}$, by solving a weighted L_1 constrained sparse least-squares problem.

Figure 4.3. a) Original image with a person b) Weights image with zero weights for the person c) WT-LARS reconstructed image (Person Removed)



Weights consist of zeros for the pixels belonging to the person in the original image and ones for everywhere else. Used fixed *mode-n* overcomplete dictionaries were a union of a Discrete Cosine Transform (DCT) dictionaries and a Symlet wavelet packet with four vanishing moments dictionaries. In the experimental results shown in Figure 4.3, a total of 170,829 nonzero samples have been used to obtain a sparse signal representation of the landscape image. Therefore, the inpainting results in, Figure 4.1 c), Figure 4.2 c) and Figure 4.3 c) clearly show that WT-LARS could be successfully used to approximate missing/incomplete data.

4.5. Conclusions

Sparse weighted multilinear least-squares is a generalization of the sparse multilinear least-squares problem, where both sides of the Kronecker LS system are multiplied by an arbitrary diagonal weights matrix. These arbitrary weights would result in a potentially very large non-Kronecker least-squares problem that could be impractical to solve as it would require significant memory and computational power.

This chapter extended the T-LARS algorithm, developed in chapter 3 [18], to become the Weighted Tensor Least Angle Regression (WT-LARS) algorithm that could efficiently solve either L_0 or L_1 constrained multilinear least-squares problems with arbitrary diagonal weights for all critical values of their regularization parameter λ. To validate our new WT-LARS algorithm, we used it to solve three image inpainting problems. In our experimental results using WT-LARS shown in Figure 4.1 and Figure 4.2, we obtained the exact sparse signal representation of RGB images behind fences after applying zero weights to the pixels representing the fences. In the experimental result using WT-LARS shown in Figure 4.3, we successfully obtained an exact sparse signal representation of an RGB landscape image occluded by a person by applying zero weights to the pixels representing this person. These results demonstrate the validity and usefulness of our new Weighted Least Angle Regression (WT-LARS) algorithm.

Chapter 5

5. Tensor Dynamic Least Angle Regression (TD-LARS)

The Tensor Least Angle Regression (T-LARS) [18] developed in chapter 3 is a computationally efficient method to solve large L_0 or L_1 constrained sparse multilinear least-squares problems for all critical values of the regularization parameter λ. We could initialize T-LARS with an L_1 solution located on the Pareto curve [23] and obtain an L_1 solution with a lower residual error, where the Pareto curve contains every solution to a linear/multilinear least-squares problem. However, we could not initialize T-LARS with a solution outside of the Pareto curve because it will violate the optimality conditions of T-LARS. Therefore, this chapter extends T-LARS and the one-dimensional L_1-Homotopy method [24] to develop the *Tensor Dynamic Least Angle Regression (TD-LARS)* algorithm to obtain a solution to an L_1 constrained multilinear least-squares problem when initialized with a non-zero initial solution located on or off of the Pareto curve. Therefore, with TD-LARS, we could efficiently obtain a solution to a multilinear L_1 minimization problem by initializing with an L_1 solution of a close problem.

5.1. Introduction

Efficiently solving either large L_0 or large L_1 constrained sparse multilinear least-squares problems is essential to obtain sparse multilinear representations of large multi-dimensional signals. Caiafa and Cichocki introduced Kronecker-OMP, a generalization of OMP, for solving nonconvex L_0 constrained sparse multilinear least-squares problems [16]. Elrewainy and Sherif [17] developed the Kronecker Least Angle Regression (K-LARS) algorithm to solve either L_0 or L_1 sparse least-squares problems efficiently (overdetermined) with a Kronecker form $\boldsymbol{A} \otimes \boldsymbol{I}$, for all critical values of the regularization parameter λ. In chapter 3 we have developed the Tensor Least angle Regression (T-LARS) [18], a generalization of Least angle Regression (LARS) [15], to solve large L_0 or large L_1 constrained sparse multilinear least-squares problems efficiently for all critical values of the regularization parameter λ, with lower computational complexity and memory usage than Kronecker-OMP [18].

T-LARS is a Homotopy algorithm that typically starts with an empty active set $I = \{\}$, and the solution $\mathcal{X} = 0$ for both L_0 or L_1 minimization problems, where the active set contains the column indices corresponding to nonzero coefficients.

Each L_1 constrained linear/multilinear least-squares problem has a unique Pareto curve [23], which contains all possible L_1 solutions to a particular L_1 constrained linear/multilinear least-squares problem, where the *X-axis* of the Pareto curve is the L_1 norm of the solution, $\|\mathcal{X}\|_1$; *Y-axis* is the L_2 norm of the residual error; The gradient is the regularization parameter λ.

If we initialize an L_1 constrained multilinear least-squares problem with a nonzero initial solution tensor, $\hat{\mathcal{X}}$ with the corresponding regularization parameter, $\hat{\lambda}$, located on the Pareto curve, we could use T-LARS to obtain an L_1 solution with $\lambda < \hat{\lambda}$. However, T-LARS could not use an initial solution, which is not located on the Pareto curve, because it would violate the optimality conditions of T-LARS.

Initializing an L_1 constrained least-squares problem with an arbitrary initial solution on or off the Pareto curve would have applications in many disciplines, including Signal Processing, Statistics, and Machine Learning.

Typically, in dynamic programming [104], image and video coding, and compression [105], [106], a large problem is broken down into overlapping small sub-problems, and the solutions of each sub-problem are combined to obtain the final solution. Since two close sub-problems with small condition numbers have close solutions, we could efficiently solve one sub-problem by initializing with the solution of the other sub-problem.

Transfer Learning is used widely in the Statistics and machine learning community due to its ability to transfer knowledge acquired in previous tasks to learn a new task efficiently [107], [108]. The parameter transfer approach in transfer learning assumes the two models are close and share common parameters or prior distributions [107], [109]. Kumagai & Kanamori [109], Maurer et al. [110], and Raina et al. [34] worked on parameter transfer in sparse coding, where the parameters transferred were the dictionaries learned from data, but they did not transfer the coefficients. However, one could improve parameter transfer in sparse coding-based dictionary learning methods and in regression by transferring the L_1 solution of an L_1 minimization problem to another L_1 minimization problem.

Asif & Romberg [24] introduced an L_1-Homotopy method to dynamically update the solutions of one-dimensional L_1 minimization problems, using the previous solution as the initial solution for a streaming set of measurements. The L_1-Homotopy method successfully update the L_1 solution, $\hat{\boldsymbol{x}}$, to obtain the L_1 solution to the new L_1 minimization problem when the signal or the dictionary change.

Therefore, in this chapter, we extend the T-LARS and the one-dimensional L_1-Homotopy algorithm to develop the Tensor Dynamic Least Angle Regression (TD-LARS) algorithm to obtain the solution to an L_1 constrained multilinear least-squares problem efficiently by initializing with a nonzero initial solution tensor $\hat{\mathcal{X}}$ that is located on or off the Pareto curve.

TD-LARS would allow using the solution of an L_1 constrained multilinear least-squares problem to solve another problem efficiently when the data tensor or the dictionary changes slightly if the condition number of the multilinear least-squares problem is small [9], [111]. Therefore, TD-LARS will have applications in multiple areas, including sparse representation of multi-dimensional streaming signals, compressing multidimensional biomedical signals, video encoding, transfer learning in tensor regression, and parameter transfer in multilinear dictionary learning.

This chapter is organized as follows: Section 5.2 provides the background theory into the sparse multilinear least-squares problem and Tensor Least Angle Regression (T-LARS). We describe the problem formulation and the Tensor Dynamic Least Angle Regression algorithm (TD-LARS) in Section 5.3. Section 5.4 provides experiment results of applying both TD-LARS and T-LARS to L_1 constrained sparse multilinear least-squares problem. We present our conclusions in Section 5.5.

5.2. Background

5.2.1. Sparse Multilinear Least-squares Problem

A multilinear transformation of a tensor $\mathcal{X}$ could be defined as, $\mathcal{Y} = \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}$, where $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$ and $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$ are N^{th} order tensors, with the equivalent vector form

$$\boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) = \mathrm{vec}(\mathcal{Y}) \tag{5.1}$$

Where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$; $n \in \{1, .. N\}$.

A sparse solution of the linear system in (5.1) could be obtained by rewriting it as an L_p minimization problem,

$$\widetilde{\mathcal{X}} = \arg\min_{\mathcal{X}} \| \boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) - \mathrm{vec}(\mathcal{Y}) \|_2^2 + \lambda \|\mathcal{X}\|_p \tag{5.2}$$

5.2.2. Tensor Least Angle Regression (T-LARS)

We could use T-LARS [18] developed in chapter 3 to obtain a sparse solution efficiently for L_0 or L_1 constrained sparse multilinear least-squares problem in (5.2) for all critical values of the regularization parameter λ.

T-LARS starts with a large value of λ, which results in an empty active set $I = \{\}$, and a solution $\widetilde{\mathcal{X}}^*_{t=0} = 0$. The set I denotes an active set of columns of the dictionary $\boldsymbol{\Phi}$, i.e., column indices where the optimal solution $\widetilde{\mathcal{X}}^*_t$ at iteration t, is nonzero, and I^c denotes its corresponding inactive set. Therefore, $\boldsymbol{\Phi}_I$ contains only the active columns of the dictionary $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}_{I^c}$ contains only its inactive columns.

At each iteration t, a new column is either added or removed from the active set I, and λ is reduced by a calculated value δ^*_t and the solution $\widetilde{\mathcal{X}}^*_t$ is moved in a direction $\boldsymbol{d}_t$.

The optimal solution at any iteration, t must satisfy the following two optimality conditions,

$$\boldsymbol{\Phi}^T_{I_t}\left(\boldsymbol{\Phi}\mathrm{vec}\big(\widetilde{\mathcal{X}}_t\big) - \mathrm{vec}(\mathcal{Y})\right) = -\lambda_t \boldsymbol{z}_t \tag{5.3}$$

$$\left\| \boldsymbol{\Phi}^T_{I^c_t}\left(\boldsymbol{\Phi}\mathrm{vec}\big(\widetilde{\mathcal{X}}_t\big) - \mathrm{vec}(\mathcal{Y})\right) \right\|_\infty \le \lambda_t \tag{5.4}$$

where, λ_t is the regularization parameter at iteration t and $\boldsymbol{z}_t$ is the sign sequence of the nonzero coefficients of $\mathrm{vec}(\widetilde{\mathcal{X}}_t)$ on the active set I.

T-LARS obtain a new solution at each iteration t, with an increasing number of coefficients, which follows a piecewise linear path until obtaining a predetermined number of active columns K or reaching a predetermined residual error ε.

5.3. Tensor Dynamic Least Angle Regression (TD-LARS)

In this section, we develop the Tensor Dynamic Least Angle Regression (TD-LARS) algorithm by extending the one-dimensional L_1-Homotopy algorithm [24], [42], and T-LARS [18] that we developed in chapter 3 to efficiently obtain the solution to an L_1 constrained multilinear least-squares problem using a solution of a slightly different problem.

Let us assume we have a sparse coefficient tensor $\widehat{\mathcal{X}}$, with support $\hat{I}$ and sign sequence $\hat{\boldsymbol{z}}$, where $\hat{\boldsymbol{z}} = \mathrm{sign}\left(\mathrm{vec}(\widehat{\mathcal{X}})\right)$, which is close to the solution of (5.2). Our objective is to obtain the L_1 solution to (5.2) efficiently, by using $\widehat{\mathcal{X}}$ as the initial solution.

If $\widehat{\mathcal{X}}$ is not the L_1 solution of (5.2) for a certain λ, T-LARS could not use $\widehat{\mathcal{X}}$ as the initial solution because $\widehat{\mathcal{X}}$ would violate the optimality conditions in (5.3) and (5.4). Therefore, the Tensor Dynamic Least Angle Regression problem is formulated by adding an extra term to (5.2) to satisfy the optimality conditions for a given initial solution.

5.3.1. Problem Formulation

Asif & Romberg [24], [42] introduced an *L₁*-Homotopy method to dynamically update the solutions of one-dimensional *L₁* minimization problems, using the previous solution as the initial solution.

$$F(\boldsymbol{x}) = \underset{\mathcal{X}}{arg\,min}\frac{1}{2}\|\boldsymbol{\Phi x} - \boldsymbol{y}\,\|_2^2 + \lambda\|\boldsymbol{x}\|_1 + (1-\epsilon)\boldsymbol{u}^T\boldsymbol{x} \tag{5.5}$$

Therefore, we could formulate the Tensor Dynamic Least Angle Regression problem by extending the vector-based L_1-Homotopy formulation in (5.5) as [24], [42].

$$F(\mathcal{X}) = \underset{\mathcal{X}}{\arg\min}\frac{1}{2}\|\boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) - \mathrm{vec}(\mathcal{Y})\,\|_2^2 + \lambda\|\mathcal{X}\|_1 \; + \epsilon\boldsymbol{u}^T\mathrm{vec}(\mathcal{X}) \tag{5.6}$$

where $0 \leq \epsilon \leq 1$ and $\boldsymbol{u} \in \mathbb{R}^{I_1 \cdots I_N}$ is a vector.

$$\frac{\partial F(\mathcal{X})}{\partial \mathrm{vec}(\mathcal{X})} = \boldsymbol{\Phi}^T \big(\boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}) - \mathrm{vec}(\mathcal{Y})\big) + \lambda \mathrm{vec}(\partial \|\mathcal{X}\|_1) + \epsilon \boldsymbol{u} = 0 \tag{5.7}$$

Where, $\partial\|\mathcal{X}\|_1$ denotes the sub-differential of the L_1 norm that could be described as,

$$\mathrm{vec}(\partial\|\mathcal{X}\|_1) = \left\{ \boldsymbol{g} \in \mathbb{R}^{I_1 \cdots I_N} \middle| \begin{array}{ll} g_i = +1, & \text{where } x_i > 0 \\ g_i = -1, & \text{where } x_i < 0 \\ g_i \in [-1, +1], & \text{where } x_i = 0 \end{array} \right. \tag{5.8}$$

Where x_i is the i^{th} element of $\mathrm{vec}(\mathcal{X})$.

Using the sub-differential $\boldsymbol{g} = \mathrm{vec}(\partial\|\mathcal{X}\|_1)$, we could describe the optimality condition $0 \in \partial f(\mathcal{X}^*)$ for a tensor $\mathcal{X}^*$ as,

$$\lambda \boldsymbol{g} + \boldsymbol{\Phi}^T (\boldsymbol{\Phi}\mathrm{vec}(\mathcal{X}^*) - \mathrm{vec}(\mathcal{Y})) + \epsilon \boldsymbol{u} = 0 \tag{5.9}$$

Where $\|\boldsymbol{g}\|_\infty \leq 1$ and $\boldsymbol{g}^T \mathrm{vec}(\mathcal{X}^*) = \|\mathcal{X}^*\|_1$

5.3.2. Tensor Dynamic Least Angle Regression (TD-LARS) Formulation

The objective of TD-LARS is to start with a nonzero initial solution $\widehat{\mathcal{X}}$, with a support $\hat{I}$ and efficiently obtain the solution to (5.2) for a given λ. Similar to T-LARS, we normalize the data tensor $\mathcal{Y}$, and the columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ to have a unit L_2 norm. Note that normalizing columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ ensure normalization of the separable dictionary $\boldsymbol{\Phi}$ [18]. For notational simplicity in the following sections, we will use $\mathcal{Y}$ to represent the normalized data tensor and $\boldsymbol{\Phi}^{(n)}$ to represent normalized dictionary matrices.

The TD-LARS algorithm starts at $t = 0$ and $\epsilon_t = 1$, where ϵ_t denotes the ϵ of (5.6) at any iteration t. At each iteration t, ϵ_t is decreased by a small value δ_t until ϵ_t goes to zero. Note that when $\epsilon_t = 0$, both problems (5.2) and (5.7) are identical. Therefore the solution of (5.7) at $\epsilon_t = 0$ is also the solution of (5.2) for a specific λ.

From (5.8) and (5.9), we could define the optimality conditions for TD-LARS at any iteration t, and for any $0 \leq \epsilon_t \leq 1$ as,

$$\boldsymbol{\Phi}_{I_t}^T \left(\boldsymbol{\Phi}\mathrm{vec}(\widetilde{\mathcal{X}}_t) - \mathrm{vec}(\mathcal{Y})\right) + \epsilon_t \boldsymbol{u}_{I_t} = -\lambda \boldsymbol{z}_t \tag{5.10}$$

$$\left|\boldsymbol{\Phi}_{I_t^c}^T\left(\boldsymbol{\Phi}\mathrm{vec}(\widetilde{\mathcal{X}}_t) - \mathrm{vec}(\mathcal{Y})\right) + \epsilon_t \boldsymbol{u}_{I_t^c}\right| \leq \lambda \tag{5.11}$$

Where $\widetilde{\mathcal{X}}_t$ is the optimal solution, $\boldsymbol{z}_t = \mathrm{sign}\left(\mathrm{vec}(\widetilde{\mathcal{X}}_t)\right)$, I_t denotes the active set, and I_t^c denotes the inactive set at any iteration t.

TD-LARS starts at, $t = 0$, $\epsilon_t = 1, \mathcal{X} = \widehat{\mathcal{X}}$ and the active set $\hat{I}$. At $\epsilon_t = 1$ the initial optimum solution $\widehat{\mathcal{X}}$ should satisfy the optimality conditions in (5.10) and (5.11). Therefore, for $\widehat{\mathcal{X}}$ to become the initial optimum solution of (5.6), we should define $\boldsymbol{u}$ as,

$$\boldsymbol{u} = -\boldsymbol{\Phi}^T\left(\boldsymbol{\Phi}\mathrm{vec}(\widehat{\mathcal{X}}) - \mathrm{vec}(\mathcal{Y})\right) - \lambda\hat{\boldsymbol{z}} \tag{5.12}$$

Where $\hat{\boldsymbol{z}} = \mathrm{sign}\left(\mathrm{vec}(\widehat{\mathcal{X}})\right)$ on the active set $\hat{I}$ and zero everywhere else.

At each iteration t, we decrease ϵ_t by a small value δ_t, and the optimal solution $\mathrm{vec}(\widetilde{\mathcal{X}}_t)$ is updated by $\delta_t\boldsymbol{d}_t$ along a direction $\boldsymbol{d}_t$,

$$\left(\boldsymbol{\Phi}_{I_t}^T\left(\boldsymbol{\Phi}\mathrm{vec}(\widetilde{\mathcal{X}}_t) - \mathrm{vec}(\mathcal{Y})\right) + \epsilon_t\boldsymbol{u}_{I_t} + \delta_t\left(\boldsymbol{\Phi}_{I_t}^T\boldsymbol{\Phi}\boldsymbol{d}_t - \boldsymbol{u}_{I_t}\right)\right) = -\lambda\boldsymbol{z}_t \tag{5.13}$$

$$\left|\underbrace{\boldsymbol{\Phi}_{I_t^c}^T\left(\boldsymbol{\Phi}\mathrm{vec}(\widetilde{\mathcal{X}}_t) - \mathrm{vec}(\mathcal{Y})\right) + \epsilon_t\boldsymbol{u}_{I_t^c}}_{\boldsymbol{p}_t} + \delta_t\underbrace{\left(\boldsymbol{\Phi}_{I_t^c}^T\boldsymbol{\Phi}\boldsymbol{d}_t - \boldsymbol{u}_{I_t^c}\right)}_{\boldsymbol{v}_t}\right| \leq \lambda \tag{5.14}$$

We obtain the update direction $\boldsymbol{d}_t$, by setting $\boldsymbol{\Phi}_{I_t}^T\boldsymbol{\Phi}\boldsymbol{d}_t - \boldsymbol{u}_{I_t} = 0$.

$$\boldsymbol{d}_t = \begin{cases} \boldsymbol{G}_t^{-1}\boldsymbol{u}_{I_t}, & \text{on } I \\ 0, & \text{Otherwise} \end{cases} \tag{5.15}$$

where $\boldsymbol{G}_t^{-1}$ is the inverse of the Gram matrix $\boldsymbol{G}_t = \left(\boldsymbol{\Phi}_{I_t}^T\boldsymbol{\Phi}_{I_t}\right)$. The size of this Gram matrix would either increase (dictionary column addition) or decrease (dictionary column removal) with each iteration t. Therefore, for computational efficiency, we use the *Schur complement* inversion formula to calculate $\boldsymbol{G}_t^{-1}$ from $\boldsymbol{G}_{t-1}^{-1}$, thereby avoiding its full calculation (See section 3.3.2.1).

The optimal solution $\mathrm{vec}(\widetilde{\mathcal{X}}_t)$ is moved in the direction $\boldsymbol{d}_t$ until a condition in (5.10) and (5.11) violates. If the condition in (5.11) is violated, an additional column of $\boldsymbol{\Phi}_{I^c}$ should be added to the active set I; If the condition in (5.10) is violated an active column of $\boldsymbol{\Phi}_I$ must be removed.

The smallest step size that would violate the condition in (5.11) is given by,

$$\delta_t^+ = \min_{i \in I^c} \left\{ \frac{\lambda - \boldsymbol{p}_t(i)}{\boldsymbol{v}_t(i)}, \frac{-\lambda - \boldsymbol{p}_t(i)}{\boldsymbol{v}_t(i)} \right\} \tag{5.16}$$

The smallest step size that would violate condition in (5.10) is given by,

$$\delta_t^- = \min_{i \in I} \left\{ -\frac{\widetilde{\boldsymbol{x}}_{t-1}(i)}{\boldsymbol{d}_t(i)} \right\} \tag{5.17}$$

Where $\widetilde{\boldsymbol{x}}_{t-1}$ = vec$\left(\widetilde{\mathcal{X}}_{t-1}\right)$.

Therefore, $\delta_t^* = \min\,(\delta_t^+, \delta_t^-)$, is the smallest step size that would violate one of the optimality conditions given in (5.10) or (5.11). If $\delta_t^* = \delta_t^+$ an additional column $i \in I^c$ is going to be added to the active set I and if $\delta_t^* = \delta_t^-$ a column $i \in I$ is removed from the active set I.

The solution $\widetilde{\mathcal{X}}_t$ is updated as

$$\mathrm{vec}\left(\widetilde{\mathcal{X}}_t\right) = \mathrm{vec}\left(\widetilde{\mathcal{X}}_{t-1}\right) + \delta_t^* \boldsymbol{d}_t \tag{5.18}$$

ϵ_{t+1} is updated as

$$\epsilon_{t+1} = \epsilon_t - \delta_t^* \tag{5.19}$$

TD-LARS evaluate $\boldsymbol{d}_t$, and δ_t^* for each iteration t and update the solution $\widetilde{\mathcal{X}}_t$ before continuing to the next iteration. The TD-LARS algorithm stops when $(\epsilon_t - \delta_t^*) \leq 0$.

5.3.3. Tensor Dynamic Least Angle Regression Algorithm (TD-LARS)

Inputs to TD-LARS are the data tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$, initial solution tensor $\widehat{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$, *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}; n \in \{1, \cdots, N\}$ where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$, and the regularization parameter λ. The output is the solution tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_n \times \ldots \times I_N}$.

Algorithm 5.1 shows the complete TD-LARS algorithm using MATLAB notation.

Algorithm 5.1: Tensor Dynamic Least Angle Regression (TD-LARS)

```
Input: normalized Tensor $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_n\times\ldots\times J_N}$; initial solution $\widehat{\mathcal{X}} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$; $L_1$ regularization parameter$\lambda$; normalized dictionary matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n\times I_n}$; $n \in \{1,..N\}$;

Initialization: Initial Residual: $\mathcal{R}_0 = \left(\widehat{\mathcal{X}} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)} - \mathcal{Y}\right)$; $\varepsilon$=1; active set: $I$ = {indices of $(\widehat{\mathcal{X}} \neq 0)$};
1.  $\mathcal{C}_1 = \mathcal{R}_0 \times_1 {\boldsymbol{\Phi}^{(1)}}^T \times_2 \ldots \times_n {\boldsymbol{\Phi}^{(n)}}^T \times_{n+1} \ldots \times_N {\boldsymbol{\Phi}^{(N)}}^T$
2.  $\boldsymbol{c_1} = \mathrm{vec}(\mathcal{C}_1)$
3.  $\boldsymbol{z} = \begin{cases} sign\left(\mathrm{vec}(\widehat{\mathcal{X}})\right), & on\ I \\ 0, & Otherwise \end{cases}$
4.  $\boldsymbol{u} = -\boldsymbol{c_1} - \boldsymbol{\lambda z}$
5.  $\boldsymbol{x}_0 = \mathrm{vec}(\widehat{\mathcal{X}})$
6.  for $n=1$ to $N$, do
7.      $\boldsymbol{G}^{(n)} = {\boldsymbol{\Phi}^{(n)}}^T \boldsymbol{\Phi}^{(n)}$
8.  end for
9.  $\boldsymbol{G}_0^{-1} = \left(\boldsymbol{\Phi}_I^T \boldsymbol{\Phi}_I\right)^{-1}$
10. while $\varepsilon > 0$
11.     $\boldsymbol{G}_t^{-1}$ = updateInverseGramMatrix ($\boldsymbol{G}_{t-1}^{-1}$, {$\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(N)}$}, I, add_column, column_idx)
          % See section 3.3.2.1 [18]
12.     $\boldsymbol{d}_{I_t} = \boldsymbol{G}_t^{-1} \boldsymbol{u}_{I_t}$
13.     $\mathrm{vec}(\mathcal{D}_t) = \boldsymbol{d}_t$
14.     $\boldsymbol{p}_t = \boldsymbol{c}_t + \varepsilon \boldsymbol{u}$
15.     $\mathcal{Q}_t = \mathcal{D}_t \times_1 \boldsymbol{G}^{(1)} \times_2 \ldots \times_n \boldsymbol{G}^{(n)} \times_{n+1} \ldots \times_N \boldsymbol{G}^{(N)}$
16.     $\boldsymbol{q}_t = \mathrm{vec}(\mathcal{Q}_t)$
17.     $\boldsymbol{v}_t = \boldsymbol{q}_t - \boldsymbol{u}$
18.     $\delta_{t_1}^+ = (-\lambda - \boldsymbol{p}_{I_t^c})./\boldsymbol{v}_{I_t^c}$  % "./" - Elementwise division
19.     $\delta_{t_2}^+ = (\lambda - \boldsymbol{p}_{I_t^c})./\boldsymbol{v}_{I_t^c}$
20.     $\delta_t^- = -\boldsymbol{x}_{I_{t-1}}./\boldsymbol{d}_{I_t}$
21.     $[\delta_t^*, index] = min\ (\delta_{t_1}^+, \delta_{t_2}^+)$
22.     add_column == True
23.     if $min\ (\delta_t^-) < \delta_t^*$
24.         $[\delta_t^*, index] = min\ (\delta_t^-)$
25.         add_column = False
26.     end
27.     $\boldsymbol{x}_{I_t} = \boldsymbol{x}_{I_{t-1}} + \delta_t^* \boldsymbol{d}_{I_t}$
28.     $\varepsilon_{t+1} = \varepsilon_t - \delta_t^*$
29.     $\boldsymbol{c}_{t+1} = \boldsymbol{c}_t + \delta_t^* \boldsymbol{q}_t$
30.     $\mathcal{R}_t = \mathcal{R}_{t-1} + \delta_t^* \mathcal{D}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)}$
31.     if add_column == True
32.         $I = I +$ {index}
33.     else
34.         $I = I -$ {index}
35.     end
```

```
36. end while
37. vec( $\mathcal{X}$ ) = $\boldsymbol{x}$
38. return $I, \mathcal{X}$
```

5.4. Experimental Results

This section presents experimental results to compare the performance of the T-LARS and the TD-LARS algorithms when used to obtain a sparse representation of 3D signals using overcomplete *mode-n* dictionaries. When comparing, T-LARS starts at a solution $\mathcal{X} = 0$ and TD-LARS starts at a nonzero initial solution $\widehat{\mathcal{X}}$.

For our computational experiments, we obtained two successive RGB video frames of a color video with a frame rate of 30 frames/Sec. and two successive 3D MRI images from a sequence of 3D MRI images from publicly available datasets.

The two RGB video frames used in our experiments consist of $232 \times 424 \times 3$ voxels, and they are the first two frames of a video of an Acorn Woodpecker obtained from the VB100 Video Bird Dataset [112].

The two 3D MRI images used in the experiments consist of $100 \times 75 \times 10$ voxels, and they are the corresponding sub-tensors of two successive 3D MRI images obtained from a sequence of 3D MRI images (4DMRI dataset) of respiratory liver motion obtained from the computer vision laboratory of ETH Zurich [113], [114].

We obtained our experimental results using a MATLAB implementation of TD-LARS and T-LARS on an MS-Windows machine: 2 Intel Xeon CPUs E5-2637 v4, 3.5GHz, 32GB RAM, and NVIDIA Tesla P100 GPU with 12GB memory.

5.4.1. Obtaining Sparse Representations of Successive RGB Video Frames Using TD-LARS

In this experiment, we used TD-LARS to obtain the sparse representation of the RGB video frame "Frame 2" by using the sparse representation of the previous RGB video frame "Frame 1," and we compared the performance with T-LARS, where Frame 1 and Frame 2 each consisted of $232 \times 424 \times 3$ voxels.

For a given data tensor $\mathcal{Y}$, *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$; $n \in \{1, .. N\}$ and a regularization parameter λ, both T-LARS and TD-LARS should obtain the same L_1 solution to (2). Therefore, to compare the accuracy and speed of the L_1 solutions obtained using TD-LARS and T-LARS, we obtained the L_1 solution of RGB video Frame 2 using TD-LARS and T-LARS for three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{232\times464}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{424\times848}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{3\times4}$ and a fixed regularization parameter $\hat{\lambda}$.

The *mode-n* dictionaries, $\boldsymbol{\Phi}^{(1)}$ and $\boldsymbol{\Phi}^{(2)}$ were a union of a Discrete Cosine Transform (DCT) dictionaries and a Symlet wavelet packet with four vanishing moments dictionaries, and the *mode-n* dictionary $\boldsymbol{\Phi}^{(3)}$ was a union of an Identity matrix with a dc column where each element is 1. The regularization parameter $\hat{\lambda}$ was selected at the residual error $\|R\| = 0.02$ of the L_1 solution of Frame 2.

Figure 5.1 a) and Figure 5.1 b) show the original RGB video Frame 1 and Frame 2, respectively. Figure 5.1 c) shows the exaggerated difference between the original Frame 1 and original Frame 2, where $difference = 10 \times (Frame\ 1 - Frame\ 2)$. Figure 5.1 d) and Figure 5.1 e) show the reconstructed Frame 1 and Frame 2, respectively, at the residual error $\|R\| = 0.02$, using the L_1 solutions obtained by T-LARS. Figure 5.1 e) shows the reconstructed Frame 2 obtained using TD-LARS by using the L_1 solution of Frame 1 as the initial solution.

Figure 5.2 a) shows the change of the parameter ε with iterations. At $\varepsilon = 1$, the TD-LARS solution $\mathcal{X}$ is same as the L_1 solution $\hat{\mathcal{X}}$ of Frame 1, and as ε goes to zero, the TD-LARS solution $\mathcal{X}$ goes to the L_1 solution of Frame 2.

Figure 5.2 b) shows the residual error vs. $\|\mathcal{X}\|_1$, which is also called the Pareto curve [23]. As shown in the Pareto curve, both L_1 solutions of Frame 1 and Frame 2 are close. The zoomed graph of Figure 5.2 b) shows the TD-LARS solution starts away from both Pareto curves of Frame 1 and Frame 2 and reach the L_1 solution of Frame 2, when the residual error reaches $\|R\| = 0.02$. Figure 5.2 c) shows the number of iterations required to reach the residual error $\|R\| = 0.02$, where T-LARS took 8,385 iterations to obtain 8,166 active columns and 8,295 iterations to obtain 8,090 active columns for Frame 1 and Frame 2, respectively. TD-LARS only took 764 iterations to obtain the 8090 active columns of the L_1 solution of Frame 2 to reach the residual error $\|R\| = 0.02$. As shown in Figure 5.2 d), T-LARS took 215s and 211s to obtain the L_1 solutions of Frame 1 and

Frame 2, respectively, whereas TD-LARS obtained the L_1 solution of Frame 2 in just 22s, which is just 10% of the time it took for T-LARS to obtain the L_1 solution of Frame 2.

Figure 5.1. a) Original RGB video Frame 1 b) Original RGB video Frame 2 c) The difference between the original RGB video Frame 1 and the original RGB video Frame 2 d) T-LARS reconstructed RGB video Frame 1 e) T-LARS reconstructed RGB video Frame 2 f) TD-LARS reconstructed RGB video Frame 2

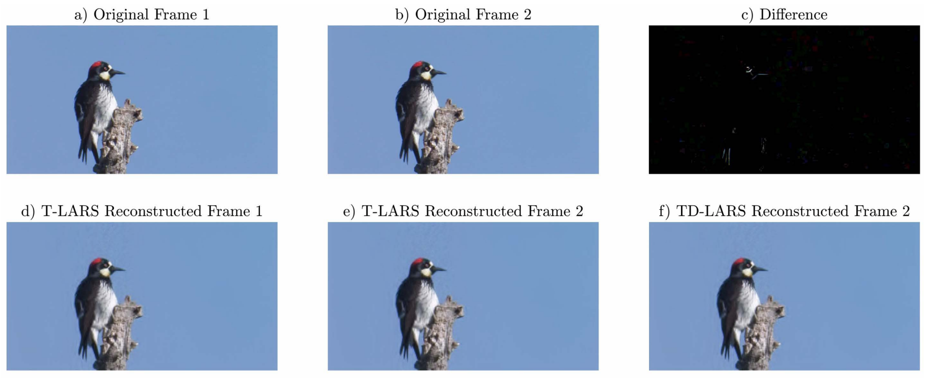
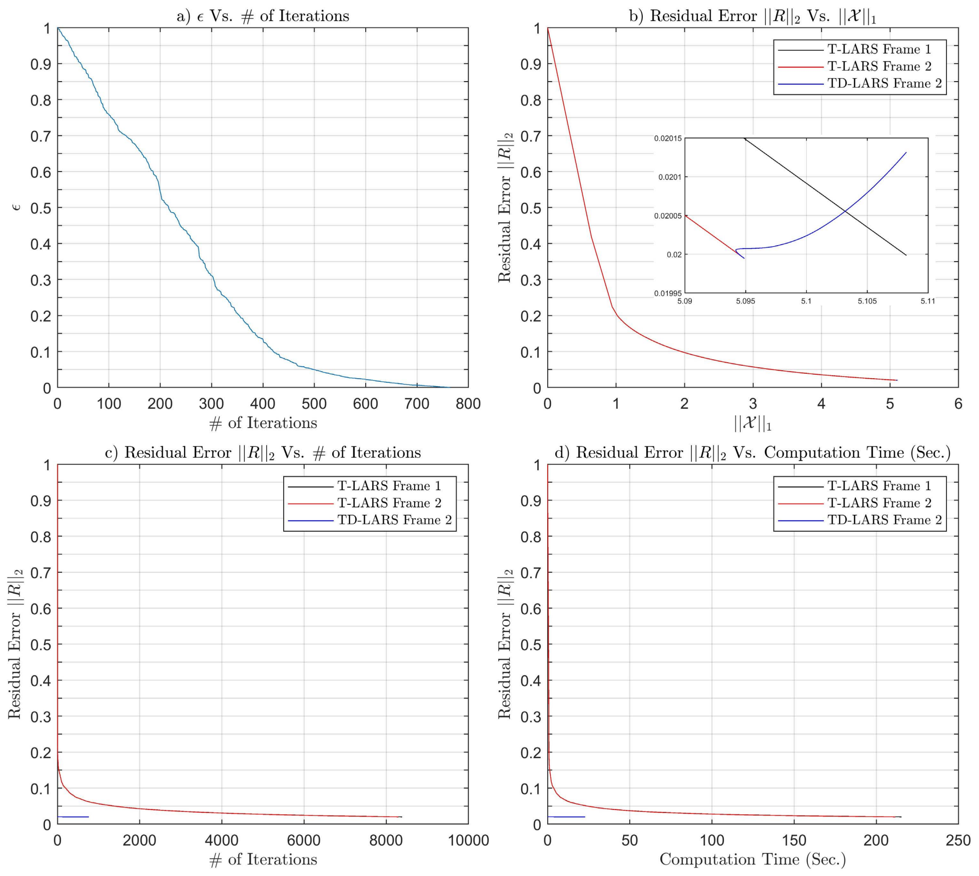

a) Original Frame 1
b) Original Frame 2
c) Difference
d) T-LARS Reconstructed Frame 1
e) T-LARS Reconstructed Frame 2
f) TD-LARS Reconstructed Frame 2

Figure 5.2. a) ε vs. the number of iterations b) Residual error vs. $\|\mathcal{X}\|_1$ c) Residual error vs. number of iterations d) Residual error vs. computation time (Sec.), obtained by applying T-LARS and TD-LARS to our RGB video Frame 1 and Frame 2



5.4.2. Obtaining Sparse Representations of Successive 3D MRI Images Using the TD-LARS

In this experiment, we used TD-LARS to obtain the sparse representation of the 3D MRI Image "3D MRI Image 2" by using the L_1 solution of the previous 3D MRI Image "3D MRI Image 1" as the initial solution and compared the performance with T-LARS, where 3D MRI Image 1 and 3D MRI Image 2 each consisted of $100 \times 75 \times 10$ voxels.

To compare the accuracy and speed of the L_1 solutions obtained using TD-LARS and T-LARS, we obtained the L_1 solution of the 3D MRI Image 2 using TD-LARS and T-LARS for three fixed *mode-n* overcomplete dictionaries, $\boldsymbol{\Phi}^{(1)} \in \mathbb{R}^{100\times 204}$, $\boldsymbol{\Phi}^{(2)} \in \mathbb{R}^{75\times 155}$ and $\boldsymbol{\Phi}^{(3)} \in \mathbb{R}^{10\times 26}$ and a fixed regularization parameter $\hat{\lambda}$.

The mode-n dictionaries, $\boldsymbol{\Phi}^{(1)}$, $\boldsymbol{\Phi}^{(2)}$ and $\boldsymbol{\Phi}^{(3)}$ were a union of Discrete Cosine Transform (DCT) dictionaries and a Symlet wavelet packet with four vanishing moments dictionaries. The regularization parameter $\hat{\lambda}$ was selected at the residual error $||R|| = 0.075$ of the L_1 solution of Frame 2.

Figure 5.3 a) and Figure 5.3 b) show the original 3D MRI Image 1 and 3D MRI Image 2. Figure 5.3 c) shows the difference between the original 3D MRI Image 1 and the original 3D MRI Image 2. Figure 5.3 c) shows a significant difference between the original 3D MRI Image 1 and the original 3D MRI Image 2. Figure 5.3 d) and Figure 5.3 e) show the reconstructed 3D MRI Image 1 and 3D MRI Image 2, respectively, at the residual error $||R|| = 0.075$, using L_1 solutions obtained by T-LARS. Figure 5.3 e) shows the reconstructed 3D MRI Image 2 obtained using TD-LARS by using the L_1 solution of the 3D MRI Image 1 as the initial solution.

Figure 5.4 a) shows the change of the parameter ε with iterations. At $\varepsilon = 1$, the TD-LARS solution $\mathcal{X}$ is same as the L_1 solution $\hat{\mathcal{X}}$ of the 3D MRI Image 1, and as ε goes to zero, the TD-LARS solution $\mathcal{X}$ goes to the L_1 solution of the 3D MRI Image 2.

Figure 5.3. a) Original 3D MRI Image 1 b) Original 3D MRI Image 2 c) The difference between the original 3D MRI Image 1 and the original 3D MRI Image 2 d) T-LARS reconstructed 3D MRI Image 1 e) T-LARS reconstructed 3D MRI Image 2 f) TD-LARS reconstructed 3D MRI Image 2

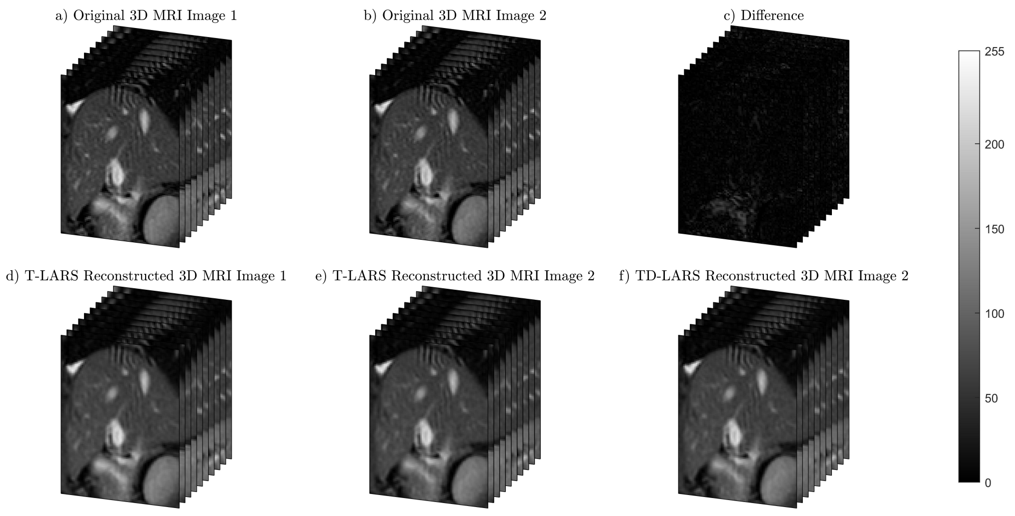

a) Original 3D MRI Image 1
b) Original 3D MRI Image 2
c) Difference
d) T-LARS Reconstructed 3D MRI Image 1
e) T-LARS Reconstructed 3D MRI Image 2
f) TD-LARS Reconstructed 3D MRI Image 2
255
200
150
100
50
0

Figure 5.4. a) ε vs. the number of iterations b) Residual error vs. $\|\mathcal{X}\|_1$ c) Residual error vs. number of iterations d) Residual error vs. computation time (Sec.), obtained by applying T-LARS and TD-LARS to our 3D MRI Image 1 and 2

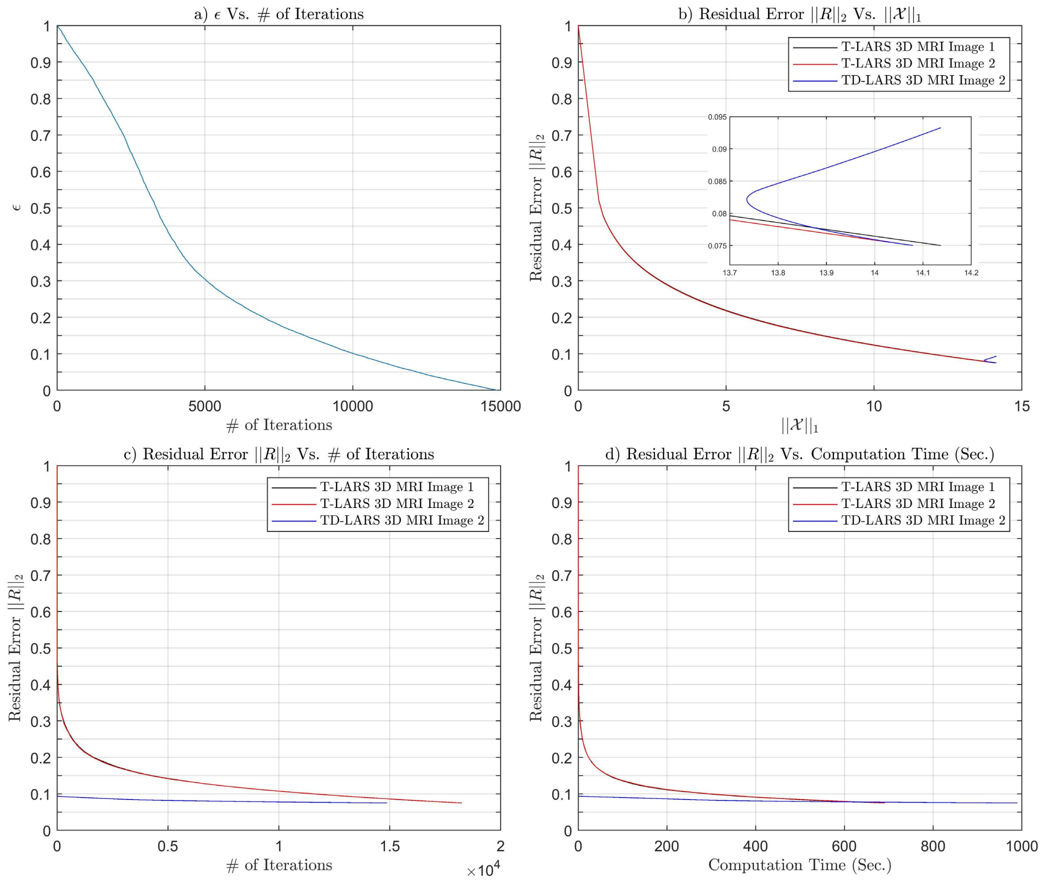


Figure 5.4 b) shows the residual error vs. $\|\mathcal{X}\|_1$, which is also called the Pareto curve [23]. The zoomed graph of Figure 5.4 b) shows the TD-LARS solution starts further away from the Pareto curves of the 3D MRI Images and reach the L_1 solution of 3D MRI Image 2, when the residual error reaches $\|R\| = 0.075$. Figure 5.4 c) shows the number of iterations required to reach the residual error $\|R\| = 0.075$, where T-LARS took $18{,}246$ iterations to obtain $16{,}381$ active columns and $18{,}197$ iterations to obtain $16{,}372$ active columns for the 3D MRI Image 1, and 3D MRI Image 2, respectively. TD-LARS took $14{,}882$ iterations to obtain the $16{,}372$ active columns of the L_1 solution of the 3D MRI Image 2 to reach the residual error $\|R\| = 0.075$. As shown in Figure 5.4 d), T-LARS took 686s and 691s to obtain the L_1 solution of the 3D MRI Image 1 and 3D MRI Image 2, respectively, whereas TD-LARS took 990s to obtain the L_1 solution of the 3D MRI Image 2.

Even though TD-LARS required 18% fewer iterations than T-LARS to obtain the L_1 solution of the 3D MRI Image 2, TD-LARS required an additional 299s, which is 43% more than the time required by T-LARS. T-LARS starts with an empty active set, and the size of the active set increases with iterations, whereas TD-LARS starts with a large active set. Therefore, TD-LARS required more time than T-LARS to obtain the L_1 solution of the 3D MRI Image 2.

Figure 5.3 c) shows a significant difference between the 3D MRI Image 1 and 3D MRI Image 2. Therefore, when the two L_1 solutions are not close, TD-LARS requires a significant amount of computation time compared to T-LARS to obtain the L_1 solution.

5.5. Conclusions

Our Tensor Dynamic Least Angle Regression (TD-LARS) algorithm is a multilinear generalization of the one-dimensional L_1-Homotopy algorithm developed by Asif & Romberg to efficiently solve multilinear L_1 minimization problems by using a nonzero initial solution. By initializing TD-LARS with a close solution, we could obtain the desired solution to an L_1 constrained multilinear least-squares problem more efficiently than solving it using T-LARS.

For experimental results, we obtained a sparse multilinear representation of an RGB video frame and a 3D MRI Image using TD-LARS by initializing with the L_1 solution of the previous video frame and the previous 3D MRI Image in a sequence, respectively. Experimental results show that the TD-LARS solution starts away from the Pareto curves of the respective L_1 minimization problems and reaches the Pareto curve when the gradient is equal to the respective regularization parameter $\hat{\lambda}$.

The normalized difference between the two original RGB video frames is much smaller, at 0.0047, compared to the normalized difference between the two 3D MRI Images, which is 0.0709. Therefore, TD-LARS obtains the L_1 solution of the video frame “Frame 2” much faster than T-LARS, where TD-LARS just took 10% of the time taken for T-LARS. However, TD-LARS required 43% more time than T-LARS to obtain the L_1 solution of the 3D MRI Image 2, even if it obtained the solution in 18% fewer iterations. Usually, TD-LARS starts with a large active set, and T-LARS starts with an empty active set and increases its size with iterations. Therefore, TD-LARS requires significantly more time than T-LARS to run an equal number of iterations. Therefore, TD-LARS requires more iterations and computation time than T-LARS to obtain the L_1 solution of a multilinear L_1 minimization problem when the normalized difference between the images increases.

However, when the two problems are close, like in the video frames example, TD-LARS could be used to obtain the solutions of L_1 constrained multilinear least-squares problems much more efficiently than any other available method. Therefore, TD-LARS will have applications in multiple areas, including sparse representation of multi-dimensional streaming signals, video encoding, transfer learning in regression, and parameter transfer in dictionary learning.

Chapter 6

6. Tensor Elastic Net (T-NET)

A sparse representation of a multi-dimensional signal could be obtained efficiently by solving either L_0 or L_1 constrained sparse multilinear least-squares problem using the Tensor Least Angle Regression (T-LARS) [18] algorithm developed in chapter 3. The L_0 minimization problem is nonconvex, and the slightly relaxed L_1 minimization problem is convex. Even though the L_2 minimization problem is strictly convex, the L_2 solution is not sparse. Zou and Hastie proposed the Elastic Net formulation [27], [28] to obtain sparse solutions to one-dimensional problems by solving strictly convex L_1 and L_2 constrained sparse linear least-squares problems. The one-dimensional Elastic Net problems could be easily solved using Least Angle Regression(LARS) [15]. This chapter proposes a multilinear Elastic Net (multi-dimensional) formulation by extending the Elastic Net (one-dimensional) to solve the strictly convex L_1, and L_2 constrained sparse multilinear least-squares problems. However, the dictionary in the multilinear Elastic Net problem has a partitioned Kronecker structure, which could not be efficiently solved using T-LARS. Therefore, in this chapter, we develop the *Tensor Elastic Net (T-NET)* algorithm to efficiently solve the multilinear Elastic Net problem using the partitioned Kronecker structure of the dictionary matrix.

6.1. Introduction

A sparse signal representation could be obtained by solving a L_0 constrained sparse least-squares problem, which is a nonconvex problem [12], [13]. *Lasso,* also known as *Basis Pursuit* (BP) [14], [25], solves a relaxed L_1 constrained least-squares problem, which is a convex problem, to obtains a sparse signal representation. Efron *et al.* introduced *Least Angle Regression* (LARS) [15], a computationally efficient method to solve both L_0 and with a slight modification L_1 constrained least-squares problems. Even though, *Ridge Regression*, solves a strictly convex L_2 constrained least-squares problems, it could not be used to obtain a sparse signal representation [26].

Basis pursuit, which solves the L_1 constrained least-squares problem selects a single nonzero coefficient from a group of highly correlated coefficients. For a n dimensional signal, Basis pursuit could only select at most n coefficients due to the nature of the convexity of L_1 constrained least-squares problems. To obtain an accurate sparse signal representation, solutions of both L_0 and L_1 constrained least-squares problems could select at most $S < (1 + \mu^{-1})/2$ nonzero coefficients, where S is the sparsity and μ is the coherence of the dictionary [73]. Therefore, as the coherence of the dictionary increases, the number of nonzero coefficients that could be selected for a sparse signal representation decreases.

Zou and Hastie developed the Elastic Net to improve the performance of L_1 constrained least-squares problems by adding an additional L_2 constraint [27], [28]. Elastic Net solves a strictly convex problem to obtain a sparse solution when both regularization coefficients of L_1 and L_2 are nonzero. Elastic Net selects all the coefficients from a group of highly correlated coefficients, and it could also obtain more than n nonzero coefficients for a n dimensional signal. However, due to the group selection, for a given residual error, the Elastic Net usually includes more nonzero coefficients than L_0 and L_1 minimization problems. Therefore, the Elastic Net is an important tool to obtain a sparse representation of a signal when the number of atoms in the dictionary is much higher than the signal's dimension. Elastic Net problems could be easily solved using the LARS algorithm [28].

Sparse representations of multi-dimensional signals are simpler and easier to obtain using separable dictionaries than non-separable dictionaries [16], [72]. Caiafa and Cichocki introduced Kronecker-OMP, a generalization of OMP, to represent multi-dimensional signals, using separable dictionaries, by solving a nonconvex L_0 constrained sparse tensor least-squares problem [16]. Elrewainy and Sherif developed the Kronecker Least Angle Regression (K-LARS) algorithm to efficiently solve either large L_0 or large L_1 sparse least-squares problems (overdetermined) with a Kronecker form $\boldsymbol{A} \otimes \boldsymbol{I}$, for all critical values of the regularization parameter λ. By extending K-LARS, authors have previously developed Tensor Least angle Regression (T-LARS) [18] in chapter 3, a generalization of LARS, to solve large L_0 or large L_1 constrained, sparse tensor least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ and with lower computational complexity and memory usage than Kronecker-OMP.

Usually, a small dictionary has a higher coherence than a large dictionary for the same frame. Therefore, a Kronecker dictionary, a Kronecker product of smaller dictionaries, has a higher coherence than a non-Kronecker dictionary of the same size for the same frame [115]. Therefore, a sparse signal representation of a multi-dimensional signal would be obtained more efficiently and accurately by solving a multilinear Elastic Net problem with both L_1 and L_2 constraints.

The dictionary matrix in a multilinear Elastic Net problem does not have a Kronecker structure. Therefore multilinear Elastic Net problems could not be efficiently solved with T-LARS. However, as shown in section 6.2, the dictionary matrix in the multilinear Elastic Net problem has a partitioned Kronecker structure. Therefore, in this chapter, we develop the Tensor Elastic Net (T-NET) algorithm by exploiting the partitioned Kronecker structure of the dictionary matrix to solve multilinear Elastic Net problems efficiently.

T-LARS could be used to solve a multilinear generalization of a LASSO-based regression model [25], [116], and similar to LASSO, it could be a poor variable selection(feature selection in machine Learning) method for tensor regression models with highly coherent predictor variables. Also, LASSO-based regression models could not select all the variables from a group of highly correlated variables and more than N predictor variables for a tensor with N elements. Therefore, T-NET allows obtaining robust solutions with better statistical properties to sparse tensor signal representation problems and tensor regression problems than T-LARS. Therefore, T-NET could be used to obtain optimum sparse signal representations of large multidimensional signals such as 3D/4D biomedical images, videos, satellite images, hyperspectral images using large over complete *mode-n* dictionaries with high coherence. T-NET also allows better convergence when used in the sparse coding step of the tensor dictionary learning algorithms such as the Tensor Method of Optimal Directions(T-MOD) and Kronecker Higher-Order SVD(K-HOSVD) [30] than T-LARS.

This chapter is organized as follows: In Section 6.2, we describe the problem formulation, Tensor Elastic Net Formulation, and the Tensor Elastic Net algorithm. Section 6.3 provides experiment results of applying both T-NET and T-LARS to a sparse multi-dimensional signal representation problem. We present our conclusions in Section 6.4.

6.2. Tensor Elastic Net

6.2.1. Problem Formulation

The Elastic Net [28] minimizes a linear least-squares problem with both L_1 and L_2 constraints. Similarly, we can define a multilinear Elastic Net to obtain a sparse tensor solution $\widetilde{\mathcal{X}}$ by minimizing a multilinear least-squares problem with both L_1 and L_2 constraints.

$$\widetilde{\mathcal{X}} = \underset{\mathcal{X}}{\operatorname{argmin}} \left\| \mathcal{Y} - \mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)} \right\|_2^2 + \gamma_1 \|\mathcal{X}\|_1 + \gamma_2 \|\mathcal{X}\|_2^2 \qquad (6.1)$$

where $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_n \times \dots \times I_N}$, $\mathcal{Y} \in \mathbb{R}^{J_1 \times \dots \times J_n \times \dots \times J_N}$, *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}; n \in \{1, \cdots, N\}$. γ_1 and γ_2 are regularization parameters for L_1 and L_2 constraints, respectively.

We could write the equivalent vector formulation of (6.1) as,

$$\widetilde{\mathcal{X}} = \underset{\widetilde{\mathcal{X}}}{\operatorname{argmin}} \ \|\operatorname{vec}(\mathcal{Y}) - \boldsymbol{\Phi}\operatorname{vec}(\mathcal{X}) \ \|_2^2 + \gamma_1 \|\operatorname{vec}(\mathcal{X})\|_1 + \gamma_2 \|\operatorname{vec}(\mathcal{X})\|_2^2 \qquad (6.2)$$

where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ is a separable dictionary.

Let

$$\boldsymbol{\Psi} = \frac{1}{\sqrt{1+\gamma_2}} \begin{vmatrix} \boldsymbol{\Phi} \\ \sqrt{\gamma_2}\, \boldsymbol{I} \end{vmatrix} \qquad (6.3)$$

And

$$\operatorname{vec}(\mathcal{Y}^*) = \begin{vmatrix} \operatorname{vec}(\mathcal{Y}) \\ \mathbf{0} \end{vmatrix} \qquad (6.4)$$

Where $\boldsymbol{\Psi} \in \mathbb{R}^{(J_1 \times \dots \times J_N + I_1 \times \dots \times I_N) \times (I_1 \times \dots \times I_N)}$ is a partitioned dictionary matrix, $\boldsymbol{I} \in \mathbb{R}^{(I_1 \times \dots \times I_N) \times (I_1 \times \dots \times I_N)}$ is an identity matrix, $\operatorname{vec}(\mathcal{Y}^*) \in \mathbb{R}^{(J_1 \dots J_N + I_1 \dots I_N)}$ and $\mathbf{0} \in \mathbb{R}^{I_1 \dots I_N}$ is a zero vector.

Therefore, we can reformulate (6.2) as,

$$\widetilde{\mathcal{X}}^* = \underset{\mathcal{X}^*}{\operatorname{argmin}} \|\operatorname{vec}(\mathcal{Y}^*) - \boldsymbol{\Psi}\operatorname{vec}(\mathcal{X}^*) \ \|_2^2 + \lambda \|\operatorname{vec}(\mathcal{X}^*)\|_1 \qquad (6.5)$$

Where $\lambda = \frac{\gamma_1}{\sqrt{1+\gamma_2}}$.

The Elastic Net solution $\widetilde{\mathcal{X}}$ is given by

$$\widetilde{\mathcal{X}} = \sqrt{1+\gamma_2}\ \widetilde{\mathcal{X}}^* \tag{6.6}$$

Equation (6.5) is a vector L_1 minimization problem, which can be solved using LARS [15].

We introduced Tensor Least Angle Regression (T-LARS) [18] in chapter 3, which is a computationally efficient algorithm to solve multilinear L_1 minimization problems with a separable dictionary $\boldsymbol{\Phi}$. However, the matrix $\boldsymbol{\Psi}$ in (6.5) is a nonseparable partitioned matrix, which does not have a Kronecker structure. Therefore, both T-LARS and LARS are computationally inefficient at solving the L_1 minimization problem in (6.5) because they require to construct large matrices such as the dictionary matrix $\boldsymbol{\Psi}$.

However, both the matrices $\boldsymbol{\Phi}$ and $\boldsymbol{I}$ in the partitioned matrix $\boldsymbol{\Psi}$ are individually separable. Therefore, in this chapter, we extend T-LARS to develop the Tensor Elastic Net (T-NET) algorithm to solve the multilinear Elastic Net problem in (6.5) efficiently, by using the partitioned Kronecker structure of $\boldsymbol{\Psi}$, without constructing large matrices.

6.2.2. Tensor Elastic Net Formulation

Tensor Elastic Net (T-NET) is an extension of the Tensor Least Angle Regression (T-LARS) to solve the multilinear Elastic Net problem in (6.5) using tensors and multilinear algebra. T-NET does not construct large matrices such as the partitioned dictionary, $\boldsymbol{\Psi}$, which is required in solving (6.5) using LARS [15] or T-LARS [18] developed in chapter 3. Instead, T-NET uses much smaller *mode-n* dictionaries $\boldsymbol{\Phi}^{(\mathrm{n})}; n \in \{1,\cdots,N\}$ for calculations.

Inputs to T-NET are the data tensor $\mathcal{Y} \in \mathbb{R}^{J_1\times\ldots\times J_n\times\ldots\times J_N}$, *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}; n \in \{1,\cdots,N\}$ where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$, L_2 regularization parameter γ_2, and the stopping criterion as a residual tolerance ε or the maximum number of non-zero coefficients K (K-sparse representation). The output is the Elastic Net solution tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_n\times\ldots\times I_N}$.

T-NET requires data tensor $\mathcal{Y}$, and columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1,\cdots,N\}$ to have a unit L_2 norm. Note that normalizing columns of each dictionary $\boldsymbol{\Phi}^{(n)}; n \in \{1,\cdots,N\}$ ensure normalization of the separable dictionary $\boldsymbol{\Phi}$. For notational simplicity in the following sections, we will use $\mathcal{Y}$ to represent the normalized tensor and $\boldsymbol{\Phi}^{(n)}$ to represent normalized *mode-n* dictionary matrices.

T-NET solves the L_1 constrained minimization problems in (6.5) for all critical values of the regularization parameter λ. T-NET starts with a large value of λ, which results in an empty active set $I = \{\}$, and a solution $\widetilde{\mathcal{X}}_{t=0}^{*} = 0$. The set I denotes an active set of columns of the dictionary $\boldsymbol{\Psi}$, i.e., column indices where the optimal solution $\widetilde{\mathcal{X}}_{t}^{*}$ at iteration t, is nonzero, and I^c denotes its corresponding inactive set. Therefore, $\boldsymbol{\Psi}_I$ contains only the active columns of the dictionary $\boldsymbol{\Psi}$ and $\boldsymbol{\Psi}_{I^c}$ contains only its inactive columns.

At each iteration t, a new column is either added or removed from the active set I, and λ is reduced by a calculated value δ_t^*. As a result of such iterations, new solutions with an increased number of coefficients that follow a piecewise linear path are obtained until a predetermined residual error ε or a predetermined number of active columns K is obtained.

The regularization parameter λ is initialized to the maximum of the correlation $\boldsymbol{c}_1$, between the columns of $\boldsymbol{\Psi}$ and the initial residual $\mathcal{R}_0^* = \mathcal{Y}^*$.

$$\boldsymbol{c}_1 = \boldsymbol{\Psi}^T \text{vec}(\mathcal{R}_0^*) = \frac{1}{\sqrt{1+\gamma_2}} \left[\boldsymbol{\Phi}^T | \sqrt{\gamma_2}\, \boldsymbol{I}\right] \begin{vmatrix} \text{vec}(\mathcal{Y}) \\ \boldsymbol{0} \end{vmatrix} \tag{6.7}$$

Therefore,

$$\boldsymbol{c}_1 = \frac{1}{\sqrt{1+\gamma_2}} \boldsymbol{\Phi}^T \text{vec}(\mathcal{Y}) \tag{6.8}$$

Since $\boldsymbol{\Phi}^T$ is a Kronecker matrix, we could easily calculate the initial correlation $\boldsymbol{c}_1$ using the full multilinear product as

$$\mathcal{C}_1 = \frac{1}{\sqrt{1+\gamma_2}} \mathcal{Y} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)^T} \tag{6.9}$$

Where $\boldsymbol{c}_1 = \text{vec}(\mathcal{C}_1)$. The column index corresponding to the maximum correlation $\boldsymbol{c}_1$ is added to the active set I.

For a given active set I, the optimal solution $\widetilde{\mathcal{X}}_t^*$ at any iteration t, could be written as

$$\text{vec}(\widetilde{\mathcal{X}}_t^*) = \begin{cases} \left(\boldsymbol{\Psi}_{I_t}^T \boldsymbol{\Psi}_{I_t}\right)^{-1} \left(\boldsymbol{\Psi}_{I_t}^T \text{vec}(\mathcal{Y}^*) - \lambda_t \boldsymbol{z}_t\right), on\ I \\ 0, \qquad \text{Otherwise} \end{cases} \tag{6.10}$$

where, λ_t is the regularization parameter λ at iteration t, $\boldsymbol{z}_t$ is the sign sequence of $\boldsymbol{c}_t$ on the active set I, and $\boldsymbol{c}_t = \boldsymbol{\Psi}^T \text{vec}(\mathcal{R}_{t-1}^*)$ is the correlation vector between the columns of the dictionary $\boldsymbol{\Psi}$ and the residual $\text{vec}(\mathcal{R}_{t-1}^*)$ at any iteration t.

The optimal solution at any iteration, t must satisfy the following two optimality conditions,

$$\boldsymbol{\Psi}_{I_t}^T \text{vec}(\mathcal{R}_t^*) = -\lambda_t \boldsymbol{z}_t \tag{6.11}$$

$$\left\| \boldsymbol{\Psi}_{I_t^c}^T \text{vec}(\mathcal{R}_t^*) \right\|_\infty \leq \lambda_t \tag{6.12}$$

where, $\text{vec}(\mathcal{R}_t^*) = \text{vec}(\mathcal{Y}^*) - \boldsymbol{\Psi}\text{vec}(\widetilde{\mathcal{X}}_t^*)$ is the residual at iteration t, and $\boldsymbol{z}_t$ is the sign sequence of the correlation $\boldsymbol{c}_t$ at iteration t, on the active set I.

The condition in (6.11) ensures that the magnitude of the correlation between all active columns of $\boldsymbol{\Psi}$ and the residual is equal to $|\lambda_t|$ at each iteration t, and the condition in (6.12) ensures that the magnitude of the correlation between the inactive columns of $\boldsymbol{\Psi}$ and the residual is less than or equal to $|\lambda_t|$.

At each iteration t, λ_t is reduced by a small step size δ_t^*, until a condition in either (6.11) or (6.12) violates. If an active column violates the condition (6.11), it is removed from the active set, and if an inactive column violates the condition (6.12), it is added to the active set.

As λ_t is reduced by δ_t^*, the solution $\widetilde{\mathcal{X}}_t^*$ change by $\delta_t^* \boldsymbol{d}_t$ along a direction $\boldsymbol{d}_t$, where $\boldsymbol{d}_{I_t^c} = 0$ and $\boldsymbol{d}_{I_t} = \boldsymbol{G}_t^{-1} \boldsymbol{z}_t$. Matrix $\boldsymbol{G}_t^{-1}$ is the inverse of the Gram matrix of the active columns of the dictionary $\boldsymbol{G}_t = \boldsymbol{\Psi}_{I_t}^T \boldsymbol{\Psi}_{I_t}$.

The size of the Gram matrix would either increase (dictionary column addition) or decrease (dictionary column removal) with each iteration t. Therefore, for computational efficiency, we use the *Schur complement* inversion formula, similar to T-LARS, to calculate $\boldsymbol{G}_t^{-1}$ from $\boldsymbol{G}_{t-1}^{-1}$ thereby avoiding its full calculation [18], [101]. See Appendix E.1 for updating the inverse of the Gram matrix using the *Schur complement* inversion formula.

The smallest step size $\delta_t^* = \min\{\delta_t^+, \delta_t^-\}$ is the minimum of δ_t^+, minimum step size for adding a column, and δ_t^-, minimum step size for removing a column. The minimum step size for removing a column from the active set is given by,

$$\delta_t^- = \min_{i \in I}\left\{-\frac{\boldsymbol{x}_{t-1}(i)}{\boldsymbol{d}_t(i)}\right\} \tag{6.13}$$

Where $\boldsymbol{x}_{t-1} = \mathrm{vec}\left(\tilde{\mathcal{X}}_{t-1}^*\right)$. The minimum step size for adding a new column to the active set is given by,

$$\delta_t^+ = \min_{i \in I^c}\left\{\frac{\lambda_t - \boldsymbol{c}_t(i)}{1 - \boldsymbol{v}_t(i)}, \frac{\lambda_t + \boldsymbol{c}_t(i)}{1 + \boldsymbol{v}_t(i)}\right\} \tag{6.14}$$

where

$$\boldsymbol{v}_t = \boldsymbol{\Psi}^T\boldsymbol{\Psi}\boldsymbol{d}_t = \frac{1}{1+\gamma_2}\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{d}_t + \gamma_2\boldsymbol{d}_t\right) \tag{6.15}$$

This vector $\boldsymbol{v}_t$ could be efficiently obtained as a multilinear transformation of the direction tensor $\mathcal{D}_{\mathrm{t}}$ by *mode-n* Gram matrices $\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T}\boldsymbol{\Phi}^{(n)}$; $n \in \{1, \cdots, N\}$.

$$\mathcal{V}_{\mathrm{t}} = \frac{1}{1+\gamma_2}\left(\mathcal{D}_{\mathrm{t}} \times_1 \boldsymbol{G}^{(1)} \times_2 \ldots \times_N \boldsymbol{G}^{(N)} + \gamma_2\mathcal{D}_{\mathrm{t}}\right) \tag{6.16}$$

Where $\boldsymbol{v}_t = \mathrm{vec}(\mathcal{V}_{\mathrm{t}})$ and $\mathrm{vec}(\mathcal{D}_{\mathrm{t}}) = \boldsymbol{d}_t$. The correlation vector $\boldsymbol{c_t}$ at iteration t, is $\boldsymbol{c_t} = \boldsymbol{\Psi}^T\mathrm{vec}(\mathcal{R}_{t-1}^*)$, where $(\mathcal{R}_{t-1}^*)$ is the residual tensor from the previous iteration. Since $\mathrm{vec}(\mathcal{R}_{t-1}^*) = \mathrm{vec}(\mathcal{R}_{t-2}^*) - \delta_{t-1}^*\boldsymbol{\Psi}_{t-1}\boldsymbol{d}_{t-1}$,

$$\boldsymbol{c_t} = \boldsymbol{\Psi}^T\mathrm{vec}(\mathcal{R}_{t-2}^*) - \delta_{t-1}^*\boldsymbol{\Psi}^T\boldsymbol{\Psi}\boldsymbol{d}_{t-1} \tag{6.17}$$

We could update the correlation vector $\boldsymbol{c_t}$ by

$$\boldsymbol{c_t} = \boldsymbol{c}_{t-1} - \delta_{t-1}^*\boldsymbol{v}_{t-1} \tag{6.18}$$

At the end of each iteration T-NET update $\tilde{\mathcal{X}}_t^*$, λ_{t+1} using the following equations

$$\tilde{\mathcal{X}}_t^* = \tilde{\mathcal{X}}_{t-1}^* + \delta_t^*\mathcal{D}_{\mathrm{t}} \tag{6.19}$$

$$\lambda_{t+1} = \lambda_t - \delta_t^* \tag{6.20}$$

T-NET stops at a predetermined residual error $\|\mathcal{R}_t\|_2 < \varepsilon$ or when a predetermined number of active columns K is obtained, where the Elastic Net residual tensor $\mathcal{R}_t$ for the Elastic Net solution $\widetilde{\mathcal{X}}_t$ is given by, $\mathcal{R}_t = \mathcal{Y} - \widetilde{\mathcal{X}}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}$. Therefore, the residual tensor $\mathcal{R}_t$ could be easily obtained using,

$$\mathcal{R}_t = \mathcal{R}_{t-1} - \delta_t^* \mathcal{D}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \boldsymbol{\Phi}^{(2)} \times_3 \cdots \times_N \boldsymbol{\Phi}^{(N)} \qquad (6.21)$$

For a normalized tensor $\mathcal{Y}$ and column normalized *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)}$, the L_2 norm of the residual is $0 \leq \|\mathcal{R}_t\|_2 \leq 1$.

The Elastic Net solution is given by

$$\widetilde{\mathcal{X}}_t = \sqrt{1+\gamma_2}\, \widetilde{\mathcal{X}}_t^* \qquad (6.22)$$

6.2.3. Tensor Elastic Net Algorithm

The complete T-NET algorithm is summarized below (Matlab notation).

Algorithm 6.1: Tensor Elastic Net (T-NET)

```
Input: normalized tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$; normalized dictionary matrices $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{J_n \times I_n}$; $n \in \{1,..N\}$; $L_2$ regularization parameter $\gamma_2$; stopping criterion: residual tolerance: $\varepsilon$ or number of non-zero coefficients: $K$
Initialization: Residual: $\mathcal{R}_0^* = \mathcal{Y}^*$; $\boldsymbol{x}_0^* = 0$; active set: $I = \{\}$;
1.  $\mathcal{C}_1 = \frac{1}{\sqrt{1+\gamma_2}} \mathcal{R}_0^* \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \ldots \times_N \boldsymbol{\Phi}^{(N)^T}$
2.  $\boldsymbol{c}_1 = \mathrm{vec}(\mathcal{C}_1)$
3.  $[\lambda_1, column\_idx] = max(\boldsymbol{c}_1)$
4.  $I = \{column\_idx\}$
5.  for $n = 1$ to $N$ do
6.      $\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T} \boldsymbol{\Phi}^{(n)}$
7.  end for
8.  while stopping criterion not reached($\|\mathcal{R}_{t-1}\|_2 > \varepsilon$ or length($I$) $< K$)
9.      $\boldsymbol{z}_t = sign(\boldsymbol{c}_t(I))$
10.     $\boldsymbol{G}_t^{-1}$ = updateInverseGramMatrix($\boldsymbol{G}_{t-1}^{-1}, \{\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(N)}\}$, $I$, $\gamma_2$, add_column, column_idx)%See Appendix E.1
11.     $\boldsymbol{d}_{I_t} = \boldsymbol{G}_t^{-1} \boldsymbol{z}_t$
12.     $\mathrm{vec}(\mathcal{D}_t) = \boldsymbol{d}_t$
13.     $\mathcal{V}_t = \frac{1}{1+\gamma_2} \left(\mathcal{D}_t \times_1 \boldsymbol{G}^{(1)} \times_2 \ldots \times_N \boldsymbol{G}^{(N)} + \gamma_2 \mathcal{D}_t\right)$
14.     $\boldsymbol{v}_t = \mathrm{vec}(\mathcal{V}_t)$
15.     $\delta_{t\ 1}^+ = (\lambda_t - \boldsymbol{c}_t(I^c))./(1 - \boldsymbol{v}_t(I^c))$   % "./"- Elementwise division
16.     $\delta_{t\ 2}^+ = (\lambda_t + \boldsymbol{c}_t(I^c))./(1 + \boldsymbol{v}_t(I^c))$
17.     $\delta_t^- = -\boldsymbol{x}_{t-1}./\, \boldsymbol{d}_t(I)$
18.     $[\delta_t^*, column\_idx] = min\left(\delta_{t\ 1}^+, \delta_{t\ 2}^+\right)$
```

```
19.         add_column == True
20.         if min $(\delta_t^-) < \delta_t^*$
21.             $[\delta_t^*,\ column\_idx]$ = min $(\delta_t^-)$
22.             add_column = False
23.         end
24.         $\tilde{\boldsymbol{x}}_t^* = \tilde{\boldsymbol{x}}_{t-1}^* + \delta_t^* \boldsymbol{d}_{I_t}$
25.         $\lambda_{t+1} = \lambda_t - \delta_t^*$
26.         $\boldsymbol{c}_{t+1} = \boldsymbol{c}_t - \delta_t^* \boldsymbol{v}_t$
27.         $\mathcal{R}_t = \mathcal{R}_{t-1} - \delta_t^* \mathcal{D}_t \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_N \boldsymbol{\Phi}^{(N)}$
28.         if add_column == True
29.             $I = I + \{column\_idx\}$
30.         else
31.             $I = I - \{column\_idx\}$
32.         end
33. end while
34. $\text{vec}(\tilde{\mathcal{X}}_{I_t}^*) = \tilde{\boldsymbol{x}}_t^*$
35. $\mathcal{X} = \sqrt{1+\gamma_2}\ \tilde{\mathcal{X}}_t^*$    %Elastic Net Solution
36. return $I, \mathcal{X}$
```

6.3. Experimental Results

This section presents experimental results to compare T-LARS and T-NET's performance to obtain sparse representations of 3D images using overcomplete DCT dictionaries with different mutual coherence values.

For our experiments shown in Figure 6.1 and Figure 6.2, we obtained 3D OCT mouse brain images from the Mendeley dataset [117], and for our experiments shown in Figure 6.3 and Figure 6.4, we obtained RGB video frames from the vid4 dataset [118]. Our experimental results were obtained using a MATLAB implementation of T-LARS and T-NET on an MS-Windows machine: 2 Intel Xeon CPUs E5-2637 v4, 3.5GHz, 32GB RAM, and NVIDIA Tesla P100 GPU with 12GB memory.

6.3.1. Experimental Setup

We compared the performance of T-LARS and T-NET when used to obtain sparse representations for our 3D OCT mouse brain images and RGB video frames by solving sparse multilinear least-squares problems using overcomplete DCT dictionaries with different coherence (μ) values.

We obtained overcomplete DCT dictionaries by oversampling the DCT basis to add non-orthogonal atoms between orthogonal atoms [32], [119].

$$\boldsymbol{D}_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right) \quad k = 0, \frac{N}{M}, \cdots, N - \frac{N}{M} \tag{6.23}$$

Where N is the number of rows and M is the number of atoms in the overcomplete DCT dictionary $\boldsymbol{D}$. For a fixed N, as M increases, the coherence (μ) of the dictionary also increases.

6.3.2. Experimental Results for 3D OCT Mouse Brain Images

In this experiment, we compare the performance of T-LARS and T-NET, to obtain *K*-sparse representations of 3D OCT mouse brain images, $\mathcal{Y}$, 70 × 100 × 10 voxels, using five sets of *mode-n* overcomplete DCT dictionaries $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}$ and $\boldsymbol{\Phi}^{(3)}$ with different coherence values.

Table 6.1 shows the overcomplete DCT *mode-n* dictionary sizes and coherence of the Kronecker Dictionary $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(3)} \otimes \boldsymbol{\Phi}^{(2)} \otimes \boldsymbol{\Phi}^{(1)}$ for each experiment.

Table 6.1. DCT *mode-n* dictionary sizes and coherence of the Kronecker Dictionary $\boldsymbol{\Phi}$

Exp.	Columns to rows ratio $\left(\frac{M}{N}\right)$	Size of $\boldsymbol{\Phi}^{(1)}$	Size of $\boldsymbol{\Phi}^{(2)}$	Size of $\boldsymbol{\Phi}^{(3)}$	Coherence of $\boldsymbol{\Phi}$ (μ)
1	1.25	70×87	100×125	10×12	0.4173
2	2	70×140	100×200	10×20	0.9012
3	3	70×210	100×300	10×30	0.9839
4	5	70×350	100×500	10×50	0.9985
5	10	70×700	100×1000	10×100	0.9999

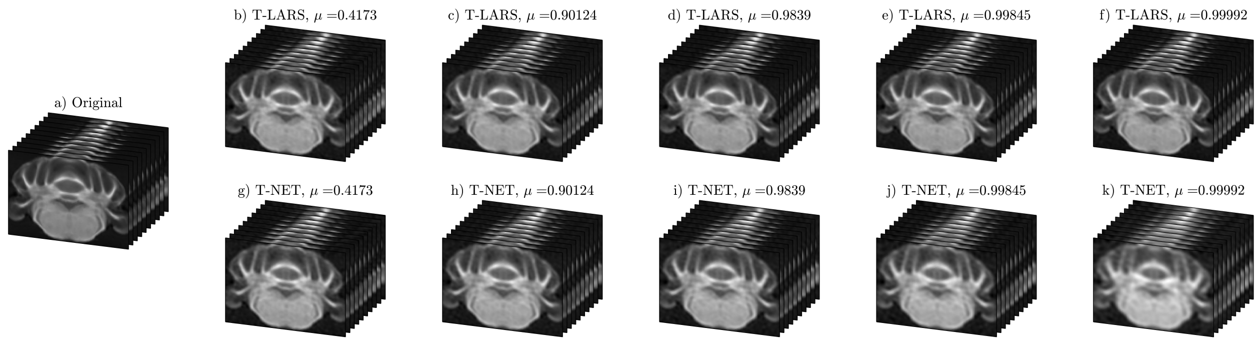
We obtained 10% non-zero coefficients, K=7,000 nonzero coefficients, for each experiment using T-LARS and T-NET to represent 3D OCT mouse brain images, where we used $\gamma_2 = 0.1$ for the regression coefficients of the L_2 norm of the solution in T-NET. Figure 6.1 and Figure 6.2 show the experimental results for representing our 3D OCT mouse brain images, using K=7,000 nonzero coefficients, over overcomplete DCT dictionaries with different coherence values (μ) shown in Table 6.1. As the columns to rows ratio (M/N) in mode-n DCT dictionaries increases, coherence of the Kronecker dictionary $\boldsymbol{\Phi}$ increases.

Table 6.2 shows the number of iterations, computation time, and residual error for representing 3D OCT mouse brain images using T-LARS and T-NET for experiments 1-5.

Table 6.2. Experimental results for T-LARS and T-NET to represent 3D OCT mouse brain images using overcomplete DCT dictionaries with different coherence values

Exp.	Coherence of $\boldsymbol{\Phi}$ (μ)	T-LARS			T-NET		
		Number of Iterations	Computation Time (Sec)	Residual Error	Number of Iterations	Computation Time (Sec)	Residual Error
1	0.4173	7144	**128.54**	**0.0467**	**7052**	137.38	0.0532
2	0.9012	7846	149.35	**0.0419**	**7102**	**137.66**	0.0619
3	0.9839	9416	198.28	**0.0410**	**7238**	**149.17**	0.0728
4	0.9985	12960	420.29	**0.0408**	**7392**	**228.28**	0.0891
5	0.9999	21604	2979.73	**0.0405**	**7916**	**944.74**	0.1156

Figure 6.1. Original 3D OCT mouse brain image (a) and its reconstruction using 10% nonzero coefficients (K = 7,000) obtained by T-LARS (b)-(f) and T-NET (g)-(k) using our overcomplete DCT dictionaries with different coherence values (μ).

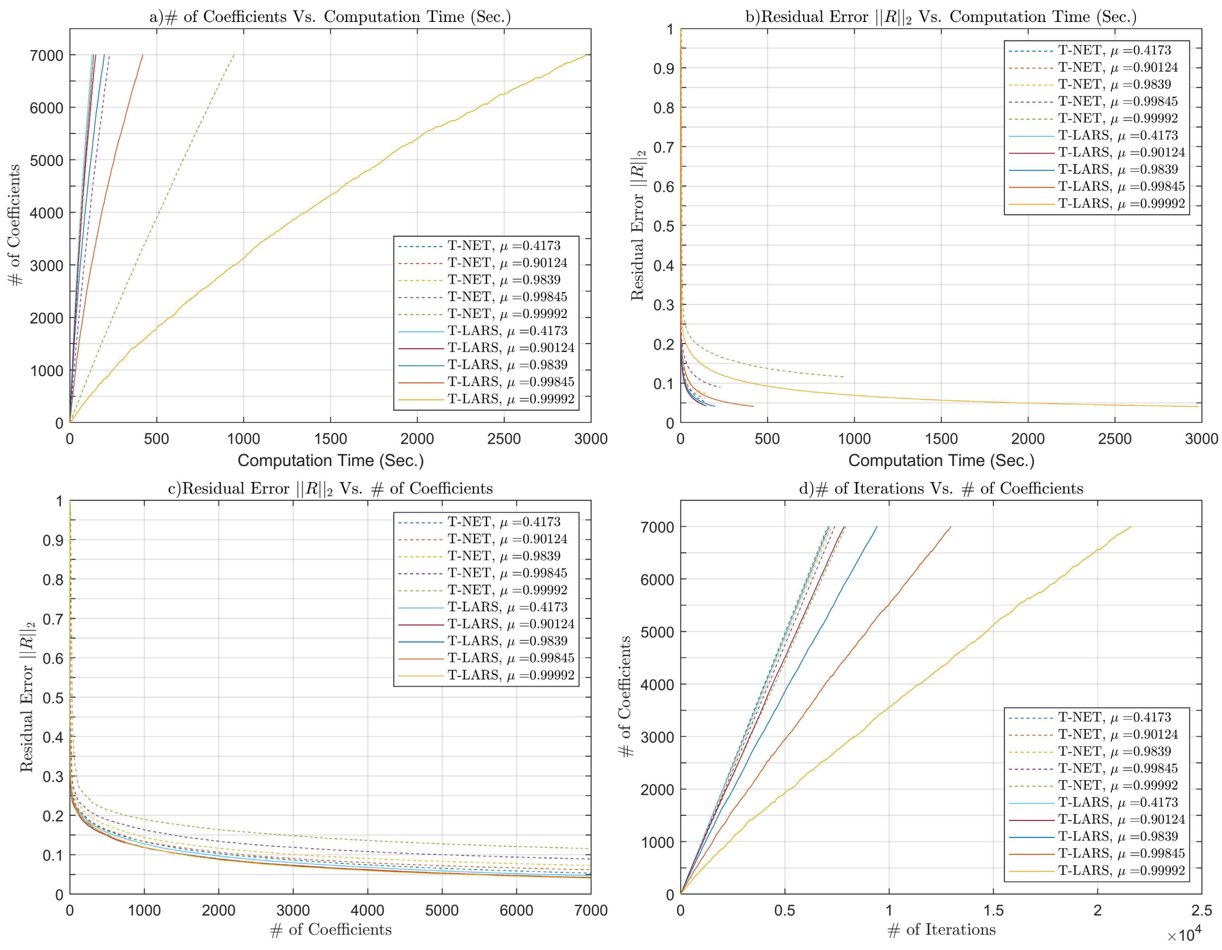


In both Figure 6.2 and Table 6.2, as the Kronecker dictionary's coherence increases, the number of iterations and the computations time required to obtain 7,000 nonzero coefficients using T-LARS increases significantly compared to T-NET. However, the residual error slightly decreases in T-LARS and increases in T-NET with the coherence for each experiment.

T-LARS only keep one atom from a group of coherent atoms. Therefore, when the dictionary's coherence is high, T-LARS adds and removes coherent atoms until it is left with one atom per group, resulting in a significantly large number of iterations to obtain 7000 nonzero coefficients.

Due to the grouping effect of T-NET, coherent atoms are grouped in the active set. Therefore, T-NET requires fewer iterations than T-LARS to obtain 7000 non-zero coefficients. Also, due to the grouping effect, T-NET requires more non-zero coefficients to obtain the same residual error as T-LARS in dictionaries with higher coherence.

Figure 6.2. (a) Number of nonzero coefficients versus computation time. (b) Residual error versus computation time. (c) Residual error versus the number of nonzero coefficients. (a) The number of nonzero coefficients versus the number of iterations, obtained by applying T-LARS and T-NET to our 3D OCT mouse brain image using overcomplete DCT dictionaries with different coherence values (μ).



6.3.3. Experimental Results for RGB video

In this experiment, we compared the performance of T-LARS and T-NET, to obtain K-sparse representations of a 4D signal, $\mathcal{Y}$, five RGB video frames of a 144 × 176 video, with 144 × 176 × 3× 5 voxels, using five sets of *mode-n* overcomplete dictionaries $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \boldsymbol{\Phi}^{(3)}$ and $\boldsymbol{\Phi}^{(4)}$ with different coherence values. Our selected *mode-n* overcomplete dictionaries $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}$, and $\boldsymbol{\Phi}^{(4)}$ are overcomplete DCT dictionaries and $\boldsymbol{\Phi}^{(3)}$ is an Identity matrix with a dc column where each element is 1. Table 6.1 shows the overcomplete DCT *mode-n* dictionary sizes and coherence of the Kronecker Dictionary $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(4)} \otimes \boldsymbol{\Phi}^{(3)} \otimes \boldsymbol{\Phi}^{(2)} \otimes \boldsymbol{\Phi}^{(1)}$ for each experiment.

Table 6.3. DCT *mode-n* dictionary sizes and coherence of the Kronecker Dictionary $\boldsymbol{\Phi}$

Exp.	Columns to rows ratio $\left(\frac{M}{N}\right)$	Size of $\boldsymbol{\Phi}^{(1)}$	Size of $\boldsymbol{\Phi}^{(2)}$	Size of $\boldsymbol{\Phi}^{(3)}$	Size of $\boldsymbol{\Phi}^{(4)}$	Coherence of $\boldsymbol{\Phi}$ (μ)
6	1.25	144×180	176×220	3×4	5×6	0.5773
7	2	144×288	176×352	3×4	5×10	0.9040
8	3	144×432	176×528	3×4	5×15	0.9848
9	5	144×720	176×880	3×4	5×25	0.9986
10	7	144×1008	176×1232	3×4	5×35	0.9996

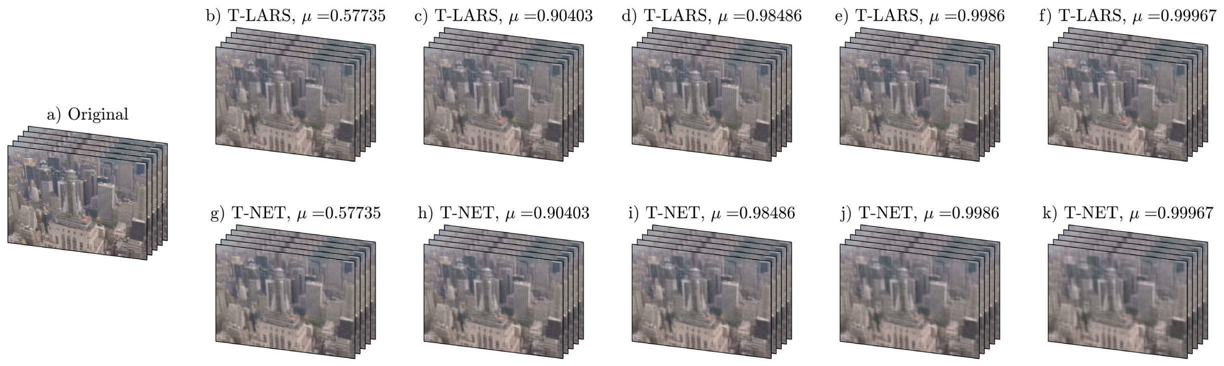
We obtained 4% non-zero coefficients, K=15,206 nonzero coefficients, for each experiment using T-LARS and T-NET to represent the RGB video, where we used $\gamma_2 = 0.1$ for the regression coefficients of the L_2 norm of the solution in T-NET. Figure 6.3 and Figure 6.4 show the experimental results for representing our RGB video, using K=15,206 nonzero coefficients, over overcomplete DCT dictionaries with different coherence values (μ) shown in Table 6.3. As the columns to rows ratio (M/N) in mode-n DCT dictionaries increases, coherence of the Kronecker dictionary $\boldsymbol{\Phi}$ increases.

Table 6.4 shows the number of iterations, computation time, and residual error for representing RGB video using T-LARS and T-NET for experiments 6-10.

Table 6.4. Experimental results for T-LARS and T-NET to represent RGB video using overcomplete DCT dictionaries with different coherence values.

Exp.	Coherence of $\boldsymbol{\Phi}$ (μ)	T-LARS			T-NET		
		Number of Iterations	Computation Time (Sec)	Residual Error	Number of Iterations	Computation Time (Sec)	Residual Error
6	0.5773	15380	539	**0.0623**	**15232**	**480**	0.3293
7	0.9040	16767	705	**0.0611**	**15454**	**592**	0.3377
8	0.9848	19697	1320	**0.0611**	**15750**	**938**	0.3671
9	0.9986	26033	5792	**0.0614**	**16401**	**2953**	0.4189
10	0.9996	32676	29680	**0.0615**	**16939**	**9859**	0.4576

Figure 6.3. Original RGB video (a) and its reconstruction using 4% nonzero coefficients (K = 15,206) obtained by T-LARS (b)-(f) and T-NET (g)-(k) using our overcomplete DCT dictionaries with different coherence values (μ).

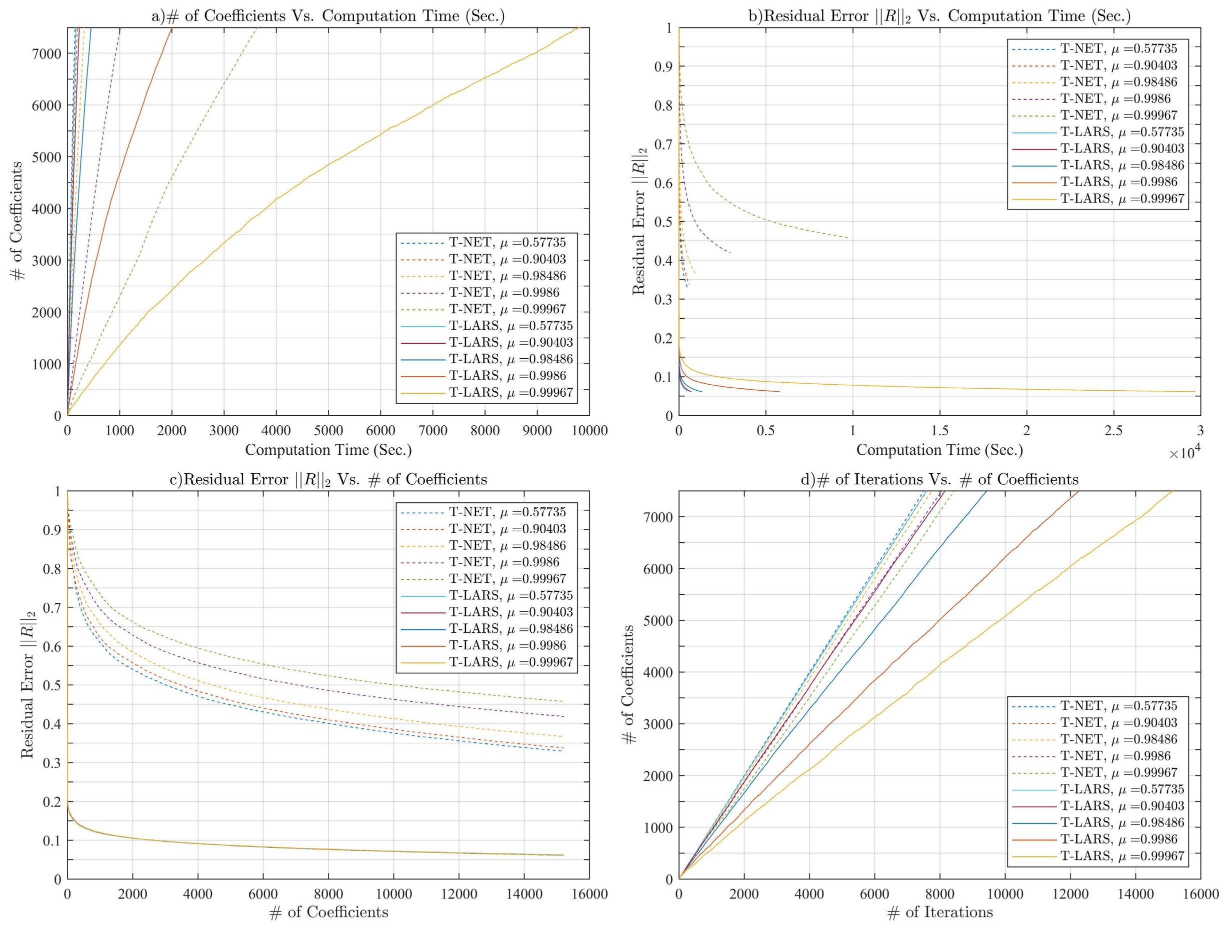


In both Figure 6.4 and Table 6.4, as the Kronecker dictionary's coherence increases, the number of iterations and the computations time required to obtain 15,206 nonzero coefficients using T-LARS increases significantly compared to T-NET. However, the residual error slightly decreases in T-LARS and increases in T-NET with the coherence for each experiment.

T-LARS only keep one atom from a group of coherent atoms. Therefore, when the dictionary's coherence is high, T-LARS adds and removes coherent atoms until it is left with one atom per group, resulting in a significantly large number of iterations to obtain 15,206 nonzero coefficients.

Due to the grouping effect of T-NET, coherent atoms are grouped in the active set. Therefore, T-NET requires fewer iterations than T-LARS to obtain 15,206 non-zero coefficients. Also, due to the grouping effect, T-NET requires more non-zero coefficients to obtain the same residual error as T-LARS in dictionaries with higher coherence.

Figure 6.4. (a) Number of nonzero coefficients versus computation time. (b) Residual error versus computation time. (c) Residual error versus the number of nonzero coefficients. (a) The number of nonzero coefficients versus the number of iterations, obtained by applying T-LARS and T-NET to our RGB video using overcomplete DCT dictionaries with different coherence values (μ).



6.4. Conclusions

Sparse signal representation of a multi-dimensional signal could be easily obtained using Kronecker dictionaries by solving a sparse multilinear least-squares problem, using T-LARS, which could be used to solve both L_0 and L_1 constrained multilinear least-squares problems efficiently. The L_0 minimization problem is a non-convex problem, and the relaxed L_1 minimization problem is a convex problem. Even though, L_2 minimization problem is strictly convex; it does not provide a sparse solution. Also, both L_0 and L_1 minimization problems have an upper limit for selecting the number of coefficients for a unique and accurate solution based on the dictionary's coherence. The group selection ability is important in some applications; however, the L_1 minimization problem does not have the group selection ability.

Tensor Elastic Net solves a strictly convex L_1 and L_2 constrained multilinear least-squares problem, which has the best properties of both L_1 and L_2 minimization problems such as sparsity and group selection ability. In addition to the group selection ability, Tensor Elastic Net can obtain more than n nonzero coefficients for a signal with n elements. Therefore, Tensor Elastic Net is ideal for solving multilinear sparse least-squares problems with highly coherent dictionaries.

The dictionary in tensor Elastic Net problem has a partitioned Kronecker structure, which could not be efficiently solved with T-LARS. Therefore, we introduced the Tensor Elastic Net (T-NET) algorithm in this chapter to efficiently solve the tensor Elastic Net problem using the partitioned Kronecker structure of the dictionary matrix.

Experimental results show that both T-LARS and T-NET behave similarly in solving the multilinear sparse representation problem for dictionaries with lower coherence. As the dictionary's coherence increases, T-LARS requires a large number of iterations and a much longer time to obtain K-Sparse solutions, whereas, for T-NET, the required number of iterations or the required time does not change significantly. However, due to group selection ability, the T-NET solution always has a higher residual error than the T-LARS solution.

Therefore, T-NET could be used to obtain a robust solution with better statistical properties to the sparse multilinear least-squares problem than T-LARS. We will be using T-NET as the primary tool to solve the sparse least-squares problems in many applications, including the Tensor Task Driven Dictionary Learning (T-TDDL) in chapter 7.

Chapter 7

7. Tensor Task-Driven Dictionary Learning (T-TDDL)

Sparse multilinear representations of multi-dimensional signals over fixed or learned separable dictionaries could be obtained efficiently using the four tensor-based algorithms developed in the previous chapters of this thesis (T-LARS [18], T-NET, TD-LARS, WT-LARS), or Kronecker-OMP [16]. However, the dictionaries learned from the data are much more efficient in obtaining sparse representations than fixed dictionaries [29].

Learned dictionaries could be used in classification or regression tasks [33]–[35]. However, regression and classification performance could be improved significantly by supervised learning of task-specific dictionaries. Mairal et al. introduced a generalized task-driven dictionary learning (TDDL) framework for supervised learning of dictionaries and model parameters to solve one-dimensional regression and classification problems [38].

The TDDL formulation solves multi-dimensional regression or classification tasks using vectorized data tensors. Therefore, using TDDL formulation for large multi-dimensional regression or classification tasks is computationally infeasible. Compared to vectorized tensors, sparse multi-linear representation of tensors requires significantly lower memory and computational resources. Therefore, this chapter extends the TDDL framework using tensor and multi-linear algebra to develop the Tensor Task-Driven Dictionary Learning (T-TDDL), an efficient multi-linear task-driven dictionary learning framework to learn task-specific *mode-n* dictionaries and *mode-n* model parameters jointly for classification or regression tasks. We use the T-NET algorithm developed in chapter 6 for the sparse coding step of the T-TDDL. This chapter also presents a compressed sensing extension for T-TDDL and calculations for regression, binary classification, and multiclass classification applications.

7.1. Introduction

Tensor-based algorithms for solving multi-dimensional problems are gaining much popularity among signal processing, machine learning, and statistics communities [5]–[7]. Tensors quickly grow in size with the number of modes and dimensions of each mode, and processing such large tensors requires significant computational resources. Instead, using a sparse representation of tensors results in fewer computations and lower memory storage requirements for fewer coefficients [8], [9]. Sparse multilinear representations of tensors are easier to obtain, using separable dictionaries, than linear representations, using non-separable dictionaries, and require significantly lower computational resources [6], [7], [16], [18], [72].

Caiafa and Cichocki introduced Kronecker-OMP, a generalization of OMP, to obtain sparse multilinear representations by solving a nonconvex L_0 constrained sparse tensor least-squares problem [16]. Authors have developed the Tensor Least angle Regression (T-LARS) [18] in chapter 3 to obtain sparse multilinear representations by solving L_0, or L_1 constrained, sparse multilinear least-squares problems for all critical values of the regularization parameter λ and with lower computational complexity and memory usage than Kronecker-OMP. By extending T-LARS, authors have developed Tensor Elastic NET (T-NET) in chapter 6, a computationally efficient algorithm to solve the multilinear Elastic Net problem [28].

The dictionaries learned from the data are much more efficient in obtaining sparse representations than fixed dictionaries [29]. Roemer et al. [30] introduced T-MOD and K-HOSVD algorithms to learn data-driven *mode-n* dictionaries to solve multilinear problems by generalizing one-dimensional data-driven dictionary learning algorithms, Method of Optimal Direction(MOD) [31], and K-SVD [32], respectively. Roemer used one-dimensional sparse coding methods in the sparse coding step of the T-MOD and K-HOSVD, requiring a significant amount of computational resources for solving data-driven tensor dictionary learning problems. However, we could efficiently solve large data-driven tensor dictionary learning problems using T-LARS [18], T-NET, or Kronecker-OMP [16] in the sparse coding step of T-MOD and K-HOSVD.

Learned dictionaries could be used in classification or regression tasks [33]–[35]. However, regression and classification performance could be improved significantly by supervised learning of task-specific dictionaries [36], [37]. Mairal et al. introduced a generalized task-driven dictionary learning(TDDL) framework for supervised learning of dictionaries and model parameters to solve one-dimensional problems [38]. Many multi-dimensional classification and regression problems have been solved using the TDDL formulation after vectorizing multi-dimensional data [39]–[41]. Recent extensions to Task-driven dictionary learning include Multi-modal task-driven dictionary learning [120] and Task-driven dictionary learning in a distributed online setting [121]. However, as far as we know, there is no method available for supervised learning of *mode-n* dictionaries and *mode-n* model parameters to solve a specific task.

Therefore, we extend the one-dimensional TDDL formulation to develop the tensor task-driven dictionary learning(T-TDDL) framework, which could work as an online data-driven or task-driven dictionary learning algorithm for supervised or semi-supervised learning of *mode-n* dictionaries and *mode-n* model parameters to solve specific tasks. We also present a multilinear compressed sensing extension to T-TDDL to learn *mode-n* task-driven dictionaries and model parameters efficiently for large data tensors. The T-TDDL framework could also be used for unsupervised learning of *mode-n* dictionaries in an online data-driven multilinear dictionary learning formulation similar to the online tensor dictionary learning algorithm (OTDL) [122].

The T-TDDL formulation could be used to solve multivariate multilinear regression [123], [124], and tensor classifications problems efficiently by learning *mode-n* dictionaries and *mode-n* model parameters to predict a tensor $\mathcal{Y} \in \mathbb{R}^{Q_1 \times \ldots \times Q_N}$ from a tensor $\mathcal{X} \in \mathbb{R}^{P_1 \times \ldots \times P_N}$. We could use different loss functions with T-TDDL formulations to efficiently solve a wide range of supervised, semi-supervised and unsupervised, tensor machine learning problems. Therefore, T-TDDL could solve a wide range of multidimensional machine learning problems, including problems in weather prediction, classifying multidimensional biomedical images such as 3D/4D MRI, 3D/4D CT, or 3D/4D PET, Chemometric analysis, Communications, augmented reality, and virtual reality. In the experimental results of this chapter, we used the T-TDDL multilinear formulation to solve a $4X$ video super-resolution problem, binary classification of 3D MRI, and multiclass classification of 3D CAD models using square, logistic regression, and Softmax cross-entropy loss functions, respectively.

This chapter is organized as follows: Section 7.2 includes a brief introduction of one-dimensional task-driven dictionary learning(TDDL) formulation. We describe our Tensor Task-Driven Dictionary Learning (T-TDDL) formulation for learning *mode-n* dictionaries and *mode-n* model parameters in detail in Section 7.3. Section 7.4 provides sample applications of T-TDDL, and Section 7.5 presents the compressed sensing extension to the T-TDDL. Section 7.6 presents experiment results of applying T-TDDL to multi-dimensional regression, binary classification, and multiclass classification tasks. We present our conclusions in Section 7.7.

7.2. Task Driven Dictionary Learning

In one-dimensional Task-driven dictionary learning (TDDL), we want to predict a vector $\boldsymbol{y} \in \mathbb{R}^Q$ from a vector $\boldsymbol{x} \in \mathbb{R}^P$, when $\boldsymbol{x}$ is associated with the vector $\boldsymbol{y}$, by supervised learning of dictionary $\boldsymbol{D} \in \mathbb{R}^{P\times U}$ and model parameters $\boldsymbol{W} \in \mathbb{R}^{Q\times U}$. Once we learn $\boldsymbol{D}$ and $\boldsymbol{W}$ using TDDL, $\boldsymbol{y}$ can be predicted using model parameters $\boldsymbol{W}$, and a sparse representation $\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D}) \in \mathbb{R}^U$ of $\boldsymbol{x}$, obtained using the dictionary $\boldsymbol{D}$. The vector $\boldsymbol{y}$ could be a finite set of labels in a classification task or a subset of $\mathbb{R}^Q$ in a regression task.

Task-driven dictionary learning formulation [38] consists of jointly learning $\boldsymbol{D}$ and $\boldsymbol{W}$ by solving,

$$\underset{\mathbf{D}\in\mathcal{D},\mathbf{W}\in\mathcal{W}}{\arg\min}\, f(\boldsymbol{D},\boldsymbol{W}) + \frac{v}{2}\|\boldsymbol{W}\|_2 \tag{7.1}$$

Where $\mathcal{D}$ and $\mathcal{W}$ are convex sets. To prevent the L_2 norm of $\boldsymbol{D}$ being arbitrarily large, the convex set $\mathcal{D}$ satisfy the constraint $\mathcal{D} \triangleq \{\boldsymbol{D} \in \mathbb{R}^{P\times U} \text{ s.t.} \forall u \in \{1,\cdots,U\}, \|\boldsymbol{d}_u\|_2 \leq 1\}$.

The convex function $f(\boldsymbol{D},\boldsymbol{W})$ is defined as

$$f(\boldsymbol{D},\boldsymbol{W}) \triangleq (1-\mu)\mathbb{E}_{\boldsymbol{y},\boldsymbol{x}}\left[l_s\big(\boldsymbol{y},\boldsymbol{W},\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D})\big)\right] + \mu\mathbb{E}_{\boldsymbol{x}}[l_u(\boldsymbol{x},\boldsymbol{D})] \tag{7.2}$$

Where $l_s\big(\boldsymbol{y},\boldsymbol{W},\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D})\big)$ is a supervised twice continuously differentiable loss function, $l_u(\boldsymbol{x},\boldsymbol{D})$ is an unsupervised twice continuously differentiable loss function, and $\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D})$ is the sparse solution of the following Elastic Net [27] problem,

$$\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D}) \triangleq \underset{\boldsymbol{\alpha}^*\in\mathbb{R}^U}{\arg\min}\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{D\alpha}\|_2^2 + \lambda_1\|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\alpha}\|_2 \tag{7.3}$$

A supervised ($\mu = 0$), semi-supervised ($1 > \mu > 0$), or unsupervised ($\mu = 1$), task-driven dictionary learning formulations are obtained depending on the value of μ in (7.2). The unsupervised dictionary learning formulation ($\mu = 1$) in (7.2), is also known as data-driven dictionary learning [38], [125].

7.3. Tensor Task Driven Dictionary Learning(T-TDDL)

In Tensor task-driven dictionary learning(T-TDDL), we want to predict a tensor $\mathcal{Y} \in \mathbb{R}^{Q_1 \times \dots \times Q_N}$ from a tensor $\mathcal{X} \in \mathbb{R}^{P_1 \times \dots \times P_N}$, when the tensor $\mathcal{X}$ is associated with the tensor $\mathcal{Y}$ by supervised learning of Kronecker dictionary $\boldsymbol{D} \in \mathbb{R}^{P \times U}$ and Kronecker model parameters $\boldsymbol{W} \in \mathbb{R}^{Q \times U}$, where $P = \prod_{n=1}^{N} P_n$, $Q = \prod_{n=1}^{N} Q_n$ and $U = \prod_{n=1}^{N} U_n$. For example, the tensor $\mathcal{Y}$ could be a finite set of labels in a classification task or a subset of $\mathbb{R}^{Q_1 \times \dots \times Q_N}$ in a regression task.

When the tensors $\mathcal{X}$ and $\mathcal{Y}$ are significantly smaller, we could vectorize them as $\boldsymbol{x} = \mathrm{vec}(\mathcal{X})$ and $\boldsymbol{y} = \mathrm{vec}(\mathcal{Y})$ respectively, and use the TDDL formulation in (7.1) to jointly learn the dictionary $\boldsymbol{D}$ and model parameters $\boldsymbol{W}$.

However, as the number of elements in tensors $\mathcal{X} \in \mathbb{R}^{P_1 \times \dots \times P_N}$ and $\mathcal{Y} \in \mathbb{R}^{Q_1 \times \dots \times Q_N}$ increases, the TDDL formulation in (7.1) quickly becomes computationally infeasible. Note that a dictionary matrix $\boldsymbol{D} \in \mathbb{R}^{P \times U}$, has PU elements. Therefore a third-order cubical tensor $\mathcal{X} \in \mathbb{R}^{100 \times 100 \times 100}$, has 10^6 elements, requires learning a dictionary matrix $\boldsymbol{D} \in \mathbb{R}^{10^6 \times 10^6}$, with 10^{12} elements, when $P = U = 10^6$, and uncompressed double-precision storage of $\boldsymbol{D}$, requires 8TB of memory. However, if $\mathcal{X} \in \mathbb{R}^{100 \times 100 \times 100 \times 100}$ is a fourth-order cubical tensor, has 10^8 elements, the square dictionary $\boldsymbol{D} \in \mathbb{R}^{10^8 \times 10^8}$, has 10^{16} elements, and the uncompressed double-precision storage of $\boldsymbol{D} \in \mathbb{R}^{10^8 \times 10^8}$, requires 80PB of memory.

Therefore, TDDL is not suitable for learning task-driven dictionaries for tensors $\mathcal{X}$ and $\mathcal{Y}$, except when they are significantly smaller.

In section 7.3.1, we formulate the Tensor task-driven dictionary learning (T-TDDL), a computationally efficient generalization of the task-driven dictionary learning(TDDL) framework [38] to efficiently learn *mode-n* dictionaries and *mode-n* model parameters to predict a tensor $\mathcal{Y}$ from a tensor $\mathcal{X}$ when the tensor $\mathcal{X}$ is associated with the tensor $\mathcal{Y}$**.**

7.3.1. Proposed Formulation

We could formulate our tensor task-driven dictionary learning (T-TDDL) to learn $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$ and $\boldsymbol{W} = \boldsymbol{W}^{(N)} \otimes \cdots \otimes \boldsymbol{W}^{(1)}$ jointly by solving,

$$\underset{\substack{\{\boldsymbol{D}^{(1)},\cdots,\boldsymbol{D}^{(N)}\}\in\mathcal{D},\\ \{\boldsymbol{W}^{(1)},\cdots,\boldsymbol{W}^{(N)}\}\in\mathcal{W}}}{\arg\min}\ f(\boldsymbol{D},\boldsymbol{W}) + \frac{v}{2}\|\boldsymbol{W}\|_2 \tag{7.4}$$

The convex function $f(\boldsymbol{D},\boldsymbol{W})$ is defined as

$$f(\boldsymbol{D},\boldsymbol{W}) \triangleq (1-\mu)\mathbb{E}_{\mathcal{Y},\mathcal{X}}\left[l_s\big(\mathcal{Y},\boldsymbol{W},\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D})\big)\right] + \mu\mathbb{E}_{\mathcal{X}}[l_u(\mathcal{X},\boldsymbol{D})] \tag{7.5}$$

Where $\mathcal{X} \in \mathbb{R}^{P_1\times\ldots\times P_N}$, and $\mathcal{Y} \in \mathbb{R}^{Q_1\times\ldots\times Q_N}$, are tensors of order N, $\boldsymbol{D}^{(n)} \in \mathbb{R}^{P_n\times U_n}; \forall n \in \{1,\cdots,N\}$, are *mode-n* dictionaries, $\boldsymbol{W}^{(n)} \in \mathbb{R}^{Q_n\times U_n}; \forall n \in \{1,\cdots,N\}$, are *mode-n* model parameters, and $\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D}) \in \mathbb{R}^{U_1\ldots U_n}$ is the Elastic net solution of,

$$\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D}) \triangleq \underset{\boldsymbol{\alpha}^*\in\mathbb{R}^U}{\arg\min}\frac{1}{2}\|\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}\|_2^2 + \lambda_1\|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\alpha}\|_2 \tag{7.6}$$

A supervised ($\mu = 0$), semi-supervised ($1 > \mu > 0$), or unsupervised ($\mu = 1$), tensor task-driven dictionary learning formulations are obtained depending on the value of μ in (7.5). The unsupervised tensor dictionary learning formulation ($\mu = 1$) in (7.5), could be used for solving online tensor data-driven dictionary learning problems [30], [122].

Authors have developed the Tensor Elastic Net (T-NET) in chapter 6 by extending the Tensor Least Angle Regression (T-LARS) [18] developed in chapter 3, which is a robust, computationally efficient algorithm to solve the multilinear elastic net problem in (7.6) for Kronecker structured dictionaries, $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$.

7.3.2. Optimization

As discussed before, the dictionary $\boldsymbol{D}$ and the model parameters $\boldsymbol{W}$ could be huge matrices for a large T-TDDL problem. Therefore, directly optimizing such $\boldsymbol{D}$ and $\boldsymbol{W}$ requires a massive amount of computational resources.

Therefore, in T-TDDL, we jointly optimize the Kronecker dictionaries $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$ and Kronecker parameters $\boldsymbol{W} = \boldsymbol{W}^{(N)} \otimes \cdots \otimes \boldsymbol{W}^{(1)}$, without explicitly constructing them, by jointly optimizing *mode-n* dictionaries $\boldsymbol{D}^{(n)} \in \mathbb{R}^{P_n \times U_n}; \forall n \in \{1, \cdots, N\}$ and *mode-n* parameter matrices $\boldsymbol{W}^{(n)} \in \mathbb{R}^{Q_n \times U_n}; \forall n \in \{1, \cdots, N\}$ using the projected stochastic gradient descent algorithm [126], [127].

Please refer to [38] for the proof of differentiability and gradients of $f(\boldsymbol{D}, \boldsymbol{W})$, w.r.t $\boldsymbol{W}$ and $\boldsymbol{D}$. In the following sections, we extend gradient calculations in [38] to obtain the gradients of the objective function in (7.4) (denoted by $\mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$), w.r.t $\boldsymbol{W}^{(n)}$ and $\boldsymbol{D}^{(n)}$, where

$$\mathrm{g}(\boldsymbol{D}, \boldsymbol{W}) = f(\boldsymbol{D}, \boldsymbol{W}) + \frac{v}{2} \|\boldsymbol{W}\|_2 \tag{7.7}$$

For notational simplicity, from here on, we denote the supervised loss function $l_s\big(\mathcal{Y}, \boldsymbol{W}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})\big)$ in (7.5) as l_s, and the unsupervised loss function $l_u(\mathcal{X}, \boldsymbol{D})$ as l_u, $\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})$ as $\boldsymbol{\alpha}^*$.

7.3.2.1. Gradient of $g(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{W}^{(n)}$

We individually optimize each *mode-n* parameter matrix, $\boldsymbol{W}^{(n)}$ using the stochastic gradient descent algorithm to optimize the Kronecker parameter matrix $\boldsymbol{W}$. Therefore we obtain the gradient of $\mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{W}^{(n)}$ as

$$\nabla_{\boldsymbol{W}^{(n)}} \mathrm{g}(\boldsymbol{D}, \boldsymbol{W}) = (1 - \mu) \mathbb{E}_{\mathcal{Y}, \mathcal{X}} \left[\nabla_{\boldsymbol{W}^{(n)}} l_s \right] + \frac{v}{2} \nabla_{\boldsymbol{W}^{(n)}} \|\boldsymbol{W}\|_2 \tag{7.8}$$

Proposition 7.1: *Let $f(\boldsymbol{\Phi})$ be a continuously differentiable function and $\boldsymbol{\Phi} \in \mathbb{R}^{P \times Q}$ be a Kronecker matrix, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n \times J_n}; \forall\, n \in \{1, \cdots, N\}$. Therefore, the gradient $\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi})$; $\forall n \in \{1, \cdots, N\}$ is given by,*

$$\left[\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi}) \right]_{i,j} = Tr\left(\left(\nabla_{\boldsymbol{\Phi}} f(\boldsymbol{\Phi}) \right)^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \right) \tag{7.9}$$

The proof is Appendix F.1.

By applying Proposition 7.1, we could obtain $\nabla_{\boldsymbol{W}^{(n)}} l_s$ as a function of $\nabla_{\boldsymbol{W}} l_s$.

$$\left[\nabla_{\boldsymbol{W}^{(n)}} l_s \right]_{i,j} = Tr\left((\nabla_{\boldsymbol{W}} l_s)^T \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{W}_{i,j}^{(n)}} \right) \tag{7.10}$$

Calculating $\nabla_{\boldsymbol{W}^{(n)}} l_s$ directly using (7.10) is not computationally efficient due to the multiplication of large matrices. However, as shown in section 7.4, we could further simplify the gradient calculation in (7.10), for a given supervised loss function l_s.

***Proposition 7.2**: Let $\boldsymbol{\Phi} \in \mathbb{R}^{P\times Q}$ be a Kronecker matrix, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$, $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n\times J_n}; \forall\ n \in \{1, \cdots, N\}$, and $\|\boldsymbol{\Phi}\|_2$ is the L_2 norm of $\boldsymbol{\Phi}$. Therefore, the gradient $\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2$ is given by,*

$$\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2 = 2\gamma_{\boldsymbol{\Phi}^{(n)}} \boldsymbol{\Phi}^{(n)} \qquad (7.11)$$

Where

$$\gamma_{\boldsymbol{\Phi}^{(n)}} = \prod_{m=1, m\neq n}^{N} Tr\left(\boldsymbol{\Phi}^{(m)^T} \boldsymbol{\Phi}^{(m)}\right)$$

The proof is in Appendix F.2.

Applying Proposition 7.2 to $\frac{v}{2}\nabla_{\mathbf{W}^{(n)}} \|\boldsymbol{W}\|_2$,

$$\frac{v}{2}\nabla_{\mathbf{W}^{(n)}} \|\boldsymbol{W}\|_2 = v\gamma_{\boldsymbol{W}^{(n)}} \boldsymbol{W}^{(n)} \qquad (7.12)$$

Where,

$$\gamma_{\boldsymbol{W}^{(n)}} = \prod_{m=1, m\neq n}^{N} Tr\left(\boldsymbol{W}^{(m)^T} \boldsymbol{W}^{(m)}\right)$$

Therefore, the gradient of g$(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{W}^{(n)}$ is,

$$\nabla_{\mathbf{W}^{(n)}} \mathrm{g}(\boldsymbol{D}, \boldsymbol{W}) = (1-\mu)\nabla_{\boldsymbol{W}^{(n)}} l_s + v\gamma_{\boldsymbol{W}^{(n)}} \boldsymbol{W}^{(n)} \qquad (7.13)$$

Where $\left[\nabla_{\boldsymbol{W}^{(n)}} l_s\right]_{i,j} = Tr\left((\nabla_{\boldsymbol{W}} l_s)^T \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{W}_{i,j}^{(n)}}\right);\ \forall\ i \in \{1, \dots, Q_n\},\ j \in \{1, \dots, U_n\}$

7.3.2.2. Gradient of $g(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{D}^{(n)}$

We individually optimize each *mode-n* dictionary matrix, $\boldsymbol{D}^{(n)}$ using the stochastic gradient descent algorithm to optimize the Kronecker dictionary matrix $\boldsymbol{D}$. Therefore we obtain the gradient of g$(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{D}^{(n)}$ as

$$\nabla_{\boldsymbol{D}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W}) = (1-\mu)\mathbb{E}_{\mathcal{Y},\mathcal{X}}\left[\nabla_{\boldsymbol{D}^{(n)}} l_s\right] + \mu\mathbb{E}_{\mathcal{X}}\left[\nabla_{\boldsymbol{D}^{(n)}} l_u\right] \tag{7.14}$$

Using Proposition 7.1, the gradient of the supervised loss function $\nabla_{\boldsymbol{D}^{(n)}} l_s$ could be written as

$$\left[\nabla_{\boldsymbol{D}^{(n)}} l_s\right]_{i,j} = Tr\left((\nabla_{\boldsymbol{D}} l_s)^T \frac{\partial \boldsymbol{D}}{\partial \boldsymbol{D}_{i,j}^{(n)}}\right) \tag{7.15}$$

In [38], it shows that,

$$\nabla_{\boldsymbol{D}} l_s = \left(-\boldsymbol{D}\boldsymbol{\beta}^*\boldsymbol{\alpha}^{*T} + (\mathrm{vec}(\mathcal{X}) - \boldsymbol{D}\boldsymbol{\alpha})\boldsymbol{\beta}^{*T}\right) \tag{7.16}$$

Where $\boldsymbol{\beta}_{I^c}^* = 0$ and, $\boldsymbol{\beta}_I^* = (\boldsymbol{D}_I^T\boldsymbol{D}_I + \lambda_2\boldsymbol{I})^{-1}\nabla_{\boldsymbol{\alpha}_I} l_s$, where I is the active set and I^c is the inactive set of the Elastic net solution.

Since $Tr(A+B) = Tr(A) + Tr(B)$ and $Tr(AB) = Tr(BA)$, from (7.15) and (7.16) we could write,

$$\left[\nabla_{\boldsymbol{D}^{(n)}} l_s\right]_{i,j} = -Tr\left((\boldsymbol{D}\boldsymbol{\beta}^*)^T\left(\frac{\partial \boldsymbol{D}}{\partial \boldsymbol{D}_{i,j}^{(n)}}\boldsymbol{\alpha}^*\right)\right) + Tr\left((\mathrm{vec}(\mathcal{X}) - \boldsymbol{D}\boldsymbol{\alpha})^T\left(\frac{\partial \boldsymbol{D}}{\partial \boldsymbol{D}_{i,j}^{(n)}}\boldsymbol{\beta}^*\right)\right) \tag{7.17}$$

Proposition 7.3: *Let f be a function of tensor $\mathcal{X} \in \mathbb{R}^{J_1\times\cdots\times J_N}$, tensor $\mathcal{Y} \in \mathbb{R}^{I_1\times\cdots\times I_N}$ and a Kronecker matrix $\boldsymbol{\Phi} \in \mathbb{R}^{P\times Q}$, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n\times J_n}$; $\forall\, n \in \{1,\cdots,N\}$. If*

$$\frac{\partial f}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = Tr\left(\mathrm{vec}(\mathcal{Y})^T\left(\frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\mathrm{vec}(\mathcal{X})\right)\right)$$

Then $\frac{\partial f}{\partial \boldsymbol{\Phi}^{(n)}}$ is given by,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}^{(n)}} = \mathcal{Y}_{(n)}\left(\mathcal{X}_{(n)}\boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}}^T\right)^T \tag{7.18}$$

Where, $\boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}} = \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n+1)} \otimes \boldsymbol{\Phi}^{(n-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}\right)$, $\mathcal{Y}_{(n)}$ is the mode-n matricization of the tensor $\mathcal{Y}$ and $\mathcal{X}_{(n)}$ is the mode-n matricization of the tensor $\mathcal{X}$.

The proof is in Appendix F.3.

Using Proposition 7.3 on (7.17), we obtain

$$\nabla_{\boldsymbol{D}^{(n)}} l_s = -(\boldsymbol{D\beta}^*)_{(n)} \left(\boldsymbol{\alpha}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T + (\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}^*)_{(n)} \left(\boldsymbol{\beta}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T \tag{7.19}$$

Where $\boldsymbol{\Psi}_{\boldsymbol{D}^{(n)}} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(n+1)} \otimes \boldsymbol{D}^{(n-1)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$, $\boldsymbol{\beta}^*_{I^c} = 0$ and $\boldsymbol{\beta}^*_I = (\boldsymbol{D}_I^T \boldsymbol{D}_I + \lambda_2 \boldsymbol{I})^{-1} \nabla_{\boldsymbol{\alpha}_I} l_s$. Matrices $(\boldsymbol{D\beta}^*)_{(n)}$, $\boldsymbol{\alpha}^*{}_{(n)}$, $(\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}^*)_{(n)}$, and $\boldsymbol{\beta}^*{}_{(n)}$, are *mode-n* matricization of respective vectors.

Using Proposition 7.1, the gradient of the unsupervised loss function $\nabla_{\boldsymbol{D}^{(n)}} l_u$ could be written as

$$\left[\nabla_{\boldsymbol{D}^{(n)}} l_u\right]_{i,j} = Tr\left((\nabla_{\boldsymbol{D}} l_u)^T \frac{\partial \boldsymbol{D}}{\partial \boldsymbol{D}^{(n)}_{i,j}} \right) \tag{7.20}$$

Since $\nabla_{\boldsymbol{D}} l_u = (\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime})\boldsymbol{\alpha}^{*\prime T}$ and $Tr(AB) = Tr(BA)$

$$\left[\nabla_{\boldsymbol{D}^{(n)}} l_u\right]_{i,j} = Tr\left((\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime})^T \frac{\partial \boldsymbol{D}}{\partial \boldsymbol{D}^{(n)}_{i,j}} \boldsymbol{\alpha}^{*\prime} \right) \tag{7.21}$$

By using Proposition 7.3 on (7.21) we could obtain $\nabla_{\boldsymbol{D}^{(n)}} l_u$ as,

$$\nabla_{\boldsymbol{D}^{(n)}} l_u = \left(\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime}\right)_{(n)} \left(\boldsymbol{\alpha}^{*\prime}{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T \tag{7.22}$$

Therefore, the gradient of $\mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{D}^{(n)}; n \in \{1, \cdots, N\}$ is,

$$\nabla_{\boldsymbol{D}^{(n)}} \mathrm{g}(\boldsymbol{D}, \boldsymbol{W}) = \left(\begin{array}{c} (1-\mu) \left(\begin{array}{c} -(\boldsymbol{D\beta}^*)_{(n)} \left(\boldsymbol{\alpha}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T \\ + (\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}^*)_{(n)} \left(\boldsymbol{\beta}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T \end{array} \right) \\ -\mu \left(\left(\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime}\right)_{(n)} \left(\boldsymbol{\alpha}^{*\prime}{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T \right) \end{array} \right) \tag{7.23}$$

where $(\boldsymbol{D\beta}^*)_{(n)}$, $\left(\boldsymbol{\alpha}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T$, $\left(\boldsymbol{\alpha}^{*\prime}{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T$, $(\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}^*)_{(n)}$, $\left(\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime}\right)_{(n)}$, and $\left(\boldsymbol{\beta}^*{}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{D}^{(n)}}\right)^T$ are *mode-n* matrices that are significantly smaller in size, compared to the separable dictionary matrix $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$. After tensorizing $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$, we could efficiently calculate *mode-n* matrices in (7.23) as N *mode-n* products (full multilinear product) with *mode-n* dictionary matrices $\boldsymbol{D}^{(1)}, \cdots, \boldsymbol{D}^{(N)}$ [5], [7].

7.3.3. Tensor Task-Driven Dictionary Learning Algorithm

In T-TDDL and T-NET large Kronecker matrices such as $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$ and $\boldsymbol{W} = \boldsymbol{W}^{(N)} \otimes \cdots \otimes \boldsymbol{W}^{(1)}$ are not constructed explicitly. Instead, smaller *mode-n* dictionaries $\boldsymbol{D}^{(1)}, \cdots, \boldsymbol{D}^{(N)}$ and $\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(N)}$ are used in computations, thereby significantly reducing memory usage and improving computational efficiency.

Algorithm 7.1: Tensor Task-Driven Dictionary Learning (T-TDDL)

Input*: $p(\mathcal{Y}, \mathcal{X})$ (a way to draw i.i.d samples of $p(\mathcal{Y}, \mathcal{X})$); $\lambda_1, \lambda_2, v \in \mathbb{R}$ (regularization parameters); $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)} \in \mathcal{D}$ (initial mode-n dictionaries); $\boldsymbol{W} = \boldsymbol{W}^{(N)} \otimes \cdots \otimes \boldsymbol{W}^{(1)} \in \mathcal{W}$ (initial mode-n parameters); T (number of iterations); μ, t_0, ρ(learning rate parameter);*

Initialization*: Initialize $\boldsymbol{D}^{(1)} \cdots \boldsymbol{D}^{(N)}$ and $\boldsymbol{W}^{(1)} \cdots \boldsymbol{W}^{(N)}$randomly or using a previous solution for transfer learning*

```
1.  for $t = 1$ to $T$ do
2.      Draw subtensors $\mathcal{Y}_t, \mathcal{X}_t$ from $p(\mathcal{Y}, \mathcal{X})$.
3.      Sparse coding: compute $\boldsymbol{\alpha}^*$ using T-NET
            $\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D}) \triangleq \underset{\boldsymbol{\alpha}^* \in \mathbb{R}^U}{arg\ min} \frac{1}{2} \|vec(\mathcal{X}) - \boldsymbol{D\alpha}\|_2^2 + \lambda_1 \|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\alpha}\|_2$
4.      Compute the active set:
            $I = \{k \in \{1, \cdots, K\} : \boldsymbol{\alpha}^*(k) \neq 0\}$
5.      Compute $\boldsymbol{\beta}^*$: set $\boldsymbol{\beta}^*_{I^c} = 0$ and
            $\boldsymbol{\beta}^*_I = (\boldsymbol{D}_I^T \boldsymbol{D}_I + \lambda_2 \boldsymbol{I})^{-1} \nabla_{\boldsymbol{\alpha}_I} l_s(\mathcal{Y}, \boldsymbol{W}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D}))$
6.      Choose the learning rate $\rho_t \leftarrow min\left(\rho, \rho \frac{t_0}{t}\right)$
7.      for $n = 1$ to $N$ do
          Update the parameters by a projected gradient step
            $\boldsymbol{W}^{(n)} \leftarrow \Pi_{\mathcal{W}} \left[ \boldsymbol{W}^{(n)} - \rho_t \left( (1 - \mu) \nabla_{\boldsymbol{W}^{(n)}} l_s + \frac{v}{2} \nabla_{\boldsymbol{W}^{(n)}} \|\boldsymbol{W}\|_2 \right) \right]$
            $\boldsymbol{D}^{(n)} \leftarrow \Pi_{\mathcal{D}} \left[ \boldsymbol{D}^{(n)} - \rho_t \left( (1 - \mu) \nabla_{\boldsymbol{D}^{(n)}} l_s - \mu \nabla_{\boldsymbol{D}^{(n)}} l_u \right) \right]$
8.      end for
9.  end for
10. return $\boldsymbol{D}^{(1)}, \cdots, \boldsymbol{D}^{(N)}$ (learned dictionaries), $\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(N)}$ (learned parameters)
```

In the T-TDDL algorithm, the learning rate parameter ρ_t is calculated using the $\rho_t = min\left(\rho, \rho \frac{t_0}{t}\right)$ [38], where for the first t_0 iterations the learning rate is ρ and after t_0 the learning rate is reduced at the rate of $\frac{t_0}{t}$. The notation $\Pi_{\mathcal{W}}$ and $\Pi_{\mathcal{D}}$ denotes an orthogonal projection on the set $\mathcal{W}$ and $\mathcal{D}$ respectively.

7.4. Applications

This section presents three example applications of our T-TDDL algorithm in multi-dimensional regression, binary classifications, and multiclass classification. We select a twice differentiable supervised loss function for each multi-dimensional application and present gradient calculations of $g(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. *mode-n* dictionaries $\boldsymbol{D}^{(n)}$ and *mode-n* model parameter matrices $\boldsymbol{W}^{(n)}$.

7.4.1. Regression

In T-TDDL multi-dimensional regression applications, the tensor $\mathcal{Y}$ is a subset of $\mathbb{R}^{Q_1\times\ldots\times Q_N}$ and the objective is to predict the tensor $\mathcal{Y}$ from a tensor $\mathcal{X} \in \mathbb{R}^{P_1\times\ldots\times P_N}$ by supervised learning of the Kronecker dictionary $\boldsymbol{D} \in \mathbb{R}^{P\times U}$ and Kronecker model parameters $\boldsymbol{W} \in \mathbb{R}^{Q\times U}$. We select the square loss as the supervised loss function for the multi-dimensional regression application. However, any other twice differentiable multi-dimensional regression loss function could be selected.

We could define the supervised regression loss function as,

$$l_s\big(\mathcal{Y}, \boldsymbol{W}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})\big) = \frac{1}{2}\|\mathrm{vec}(\mathcal{Y}) - \boldsymbol{W\alpha}\|_2^2 \qquad (7.24)$$

The gradient $\nabla_{\boldsymbol{W}^{(n)}} g(\boldsymbol{D}, \boldsymbol{W})$ given in (7.13) depends on the gradient of the supervised loss function $\nabla_{\boldsymbol{W}^{(n)}} l_s$. In (7.10) we obtained the gradient $\nabla_{\boldsymbol{W}^{(n)}} l_s$ as a function of the gradient $\nabla_{\boldsymbol{W}} l_s$. Since

$$\nabla_{\boldsymbol{W}} l_s = -(\mathrm{vec}(\mathcal{Y}) - \boldsymbol{W\alpha}^*)\boldsymbol{\alpha}^{*T} \qquad (7.25)$$

And $Tr(AB) = Tr(BA)$, from (7.10) and (7.25) we obtain,

$$\left[\nabla_{\boldsymbol{W}^{(n)}} l_s\right]_{i,j} = -Tr\left((\mathrm{vec}(\mathcal{Y}) - \boldsymbol{W\alpha}^*)^T \left(\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{W}_{i,j}^{(n)}}\right)\boldsymbol{\alpha}^*\right) \qquad (7.26)$$

Therefore, using *Proposition 7.3*, we could obtain $\nabla_{\boldsymbol{W}^{(n)}} l_s$ as,

$$\nabla_{\boldsymbol{W}^{(n)}} l_s = -(\mathrm{vec}(\mathcal{Y}) - \boldsymbol{W\alpha}^*)_{(n)}\big(\boldsymbol{\alpha}^*{}_{(n)}\boldsymbol{\Psi}_{\boldsymbol{W}^{(n)}}^T\big)^T \qquad (7.27)$$

where

$$\boldsymbol{\Psi}_{\boldsymbol{W}^{(n)}} = \boldsymbol{W}^{(N)} \otimes \ldots \otimes \boldsymbol{W}^{(n+1)} \otimes \boldsymbol{W}^{(n-1)} \otimes \cdots \otimes \boldsymbol{W}^{(1)}$$

The gradient $\nabla_{\boldsymbol{D}^{(n)}} \mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$ is given in (7.23), where $\boldsymbol{\beta}_I^*$ depends on $\nabla_{\boldsymbol{\alpha}_I} l_s$. Therefore, for the supervised regression loss function,

$$\nabla_{\boldsymbol{\alpha}_I} l_s = -\boldsymbol{W}^T(\mathrm{vec}(\mathcal{Y}) - \boldsymbol{W}\boldsymbol{\alpha}^*) \tag{7.28}$$

After learning the Kronecker dictionary $\boldsymbol{D}$ and the Kronecker model parameters $\boldsymbol{W}$, we could predict $\mathcal{Y} \in \mathbb{R}^{Q_1 \times \ldots \times Q_N}$ for a new tensor $\mathcal{X} \in \mathbb{R}^{P_1 \times \ldots \times P_N}$ as $\mathrm{vec}(\mathcal{Y}) = \boldsymbol{W}\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})$, where $\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})$ is the Elastic net solution of (7.3).

7.4.2. Binary Classification

In the T-TDDL binary classification applications, the objective is to predict a scalar $y = \{-1, +1\}$ from a tensor $\mathcal{X} \in \mathbb{R}^{P_1 \times \ldots \times P_N}$ by supervised learning of the Kronecker dictionary $\boldsymbol{D} \in \mathbb{R}^{P \times U}$ and Kronecker model parameters vector $\boldsymbol{w} \in \mathbb{R}^{1 \times U}$, where $\boldsymbol{D} = \boldsymbol{D}^{(N)} \otimes \cdots \otimes \boldsymbol{D}^{(1)}$ and $\boldsymbol{w} = \boldsymbol{w}^{(N)} \otimes \cdots \otimes \boldsymbol{w}^{(1)}$. We select the logistic regression loss function as the supervised binary classification loss function [38].

$$l_s\big(y, \boldsymbol{w}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})\big) = ln\big(1 + e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})}\big) \tag{7.29}$$

The gradient $\nabla_{\boldsymbol{w}^{(n)}} \mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$ in (7.13) depends on the gradient of the supervised loss function $\nabla_{\boldsymbol{w}^{(n)}} l_s$. From (7.10) we could obtain the gradient $\nabla_{\boldsymbol{w}^{(n)}} l_s$ as a function of the gradient $\nabla_{\boldsymbol{w}} l_s$. The gradient of the supervised binary classification loss function w.r.t $\boldsymbol{w}$ is,

$$\nabla_{\boldsymbol{w}} l_s = \frac{-e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*} y\boldsymbol{\alpha}^{*T}}{1 + e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}} \tag{7.30}$$

Since $Tr(AB) = Tr(BA)$, from (7.10) and (7.30) we obtain,

$$\left[\nabla_{\boldsymbol{w}^{(n)}} l_s\right]_j = Tr\left(\frac{-y}{1 + e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}} \frac{\partial \boldsymbol{w}}{\partial \boldsymbol{w}_j^{(n)}} \boldsymbol{\alpha}^*\right) \tag{7.31}$$

Therefore, using Proposition 7.3, we could obtain $\nabla_{\boldsymbol{w}^{(n)}} l_s$ as,

$$\nabla_{\boldsymbol{w}^{(n)}} l_s = \left(\frac{-ye^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}}{1 + e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}}\right)\left(\boldsymbol{\alpha}^*{}_{(n)} \boldsymbol{\Psi}_{\boldsymbol{w}^{(n)}}^T\right)^T \tag{7.32}$$

Where, $\boldsymbol{\Psi}_{\boldsymbol{w}^{(n)}} = \boldsymbol{w}^{(N)} \otimes \cdots \otimes \boldsymbol{w}^{(n+1)} \otimes \boldsymbol{w}^{(n-1)} \otimes \cdots \otimes \boldsymbol{w}^{(1)}$, $\boldsymbol{w}^{(n)} \in \mathbb{R}^{1\times U_n}; \forall n \in \{1, \cdots, N\}$ and $\boldsymbol{\alpha}^*{}_{(n)}$ is the *mode-n* matricization of the vector $\boldsymbol{\alpha}^*$.

The gradient of $\mathrm{g}(\boldsymbol{D}, \boldsymbol{W})$ w.r.t. $\boldsymbol{D}^{(n)}$ is given in (7.23), where $\boldsymbol{\beta}_I^*$ depends on the gradient of the supervised binary classification loss function $\nabla_{\boldsymbol{\alpha}_I} l_s$. Therefore, for the supervised binary classification loss function,

$$\nabla_{\boldsymbol{\alpha}_I} l_s = \frac{-ye^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}\boldsymbol{w}^T}{1 + e^{-y\boldsymbol{w}\boldsymbol{\alpha}^*}} \tag{7.33}$$

After learning the Kronecker dictionary $\boldsymbol{D}$ and the Kronecker model parameters vector $\boldsymbol{w}$, a new tensor $\mathcal{X}$ could be classified according to the sign of $\boldsymbol{w}\boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})$.

7.4.3. Multiclass Classification

In the T-TDDL multiclass classification applications, the objective is to predict a finite set of labels $\mathcal{Y} \in \{1, \cdots, Q\}$ with $Q > 2$ from a tensor $\mathcal{X} \in \mathbb{R}^{P_1\times\cdots\times P_N}$ by supervised learning of the Kronecker dictionary $\boldsymbol{D} \in \mathbb{R}^{P\times U}$ and model parameters $\boldsymbol{W} \in \mathbb{R}^{Q\times U}$.

We could formulate a multiclass classifier using a binary classifier in a one-vs-all or one-vs-one configurations [38], [120]. However, binary classifiers in one-vs-all or one-vs-one configurations have scalability issues. It is also possible to formulate the multiclass classification problem as a regression problem using a binary vector for $\mathcal{Y}$, where the k^{th} element of the binary vector is one for a class k and zero everywhere else [38].

A multiclass loss function could solve multiclass classification problems efficiently using an all-vs-all configuration. Therefore, we use the Softmax cross-entropy loss function for solving multi-dimensional multiclass classification problems. We could define the Softmax cross-entropy loss function for classifying K classes as,

$$l_s\big(\boldsymbol{y}, \boldsymbol{W}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D})\big) = -\sum_{k=1}^{K} 1_{\{y=k\}} ln\left(\frac{e^{\boldsymbol{I}_k^T\boldsymbol{W}\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D})}}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D})}}\right) \tag{7.34}$$

Where $\boldsymbol{I}_1, \dots, \boldsymbol{I}_K$ are the columns of the identity matrix $\boldsymbol{I} = [\boldsymbol{I}_1 \dots \boldsymbol{I}_K] \in \mathbb{R}^{K\times K}$, and $1_{\{.\}}$ is the indicator function.

The gradient $\nabla_{\boldsymbol{W}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W})$ given in (7.13) depends on the gradient of the supervised loss function $\nabla_{\boldsymbol{W}^{(n)}}l_s$. In (7.10) we obtained the gradient $\nabla_{\boldsymbol{W}^{(n)}}l_s$ as a function of the gradient $\nabla_{\boldsymbol{W}}l_s$. We could calculate the gradient $\nabla_{\boldsymbol{W}}l_s$ for the Softmax cross-entropy loss function as,

$$\nabla_{\boldsymbol{W}}l_s = \left(\frac{\gamma}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*}} - \boldsymbol{I}_y\right)\boldsymbol{\alpha}^{*T} \tag{7.35}$$

Where $\gamma \in \mathbb{R}^K$ and $\gamma_k = e^{\boldsymbol{I}_k^T\boldsymbol{W}\boldsymbol{\alpha}^*}$; $\forall\, k = \{1,\dots,K\}$.

Since $Tr(AB) = Tr(BA)$, from (7.10) and (7.35) we obtain,

$$\left[\nabla_{\boldsymbol{W}^{(n)}}l_s\right]_{i,j} = Tr\left(\left(\frac{\gamma}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*}} - \boldsymbol{I}_y\right)^T \frac{\partial\boldsymbol{W}}{\partial\boldsymbol{W}_{i,j}^{(n)}}\boldsymbol{\alpha}^*\right) \tag{7.36}$$

We could use Proposition 7.3 to calculate $\nabla_{\boldsymbol{W}^{(n)}}l_s$ as,

$$\nabla_{\boldsymbol{W}^{(n)}}l_s = \left(\frac{\gamma_{(n)}}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*}} - \boldsymbol{I}_{y_{(n)}}\right)\left(\boldsymbol{\alpha}^*{}_{(n)}\boldsymbol{\Psi}_{\boldsymbol{W}^{(n)}}^T\right)^T \tag{7.37}$$

Where $\boldsymbol{\alpha}^*{}_{(n)}, \gamma_{(n)}$ and $\boldsymbol{I}_{y_{(n)}}$ are *mode-n* matricization of $\boldsymbol{\alpha}^*, \gamma$ and $\boldsymbol{I}_y$, respectively.

The gradient of $\mathrm{g}(\boldsymbol{D},\boldsymbol{W})$ w.r.t. $\boldsymbol{D}^{(n)}$ is given in (7.23), where $\boldsymbol{\beta}_I^*$ depends on the gradient of the supervised multiclass classification loss function $\nabla_{\boldsymbol{\alpha}_I}l_s$. Therefore, for the supervised Softmax cross-entropy loss function, $\nabla_{\boldsymbol{\alpha}_I}l_s$ is given by,

$$\nabla_{\boldsymbol{\alpha}_I}l_s = \boldsymbol{W}^T\left(\frac{\gamma}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*}} - \boldsymbol{I}_{\boldsymbol{y}}\right) \tag{7.38}$$

After learning $\boldsymbol{D}$ and $\boldsymbol{W}$, a new tensor $\mathcal{X}$ is classified according to the class k with the maximum probability $p(\boldsymbol{y}|\mathcal{X})$, where

$$p(\boldsymbol{y}|\mathcal{X}) = \underset{k\in\{1,\dots,k\}}{\arg\max}\frac{e^{\boldsymbol{I}_k^T\boldsymbol{W}\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D})}}{\sum_{c=1}^{K} e^{\boldsymbol{I}_c^T\boldsymbol{W}\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D})}} \tag{7.39}$$

7.5. Compressed Sensing Extension

Computations involving large multi-dimensional signals require many dense samples and extensive computational resources. Compressed sensing solves this problem by projecting the dense signal to a sparse domain using a sensing matrix $\boldsymbol{Z}$, where a small number of nonzero samples are obtained in the sparse domain [19], [20]. Therefore, we could significantly improve the performance of T-TDDL by using compressed sensing to project large tensors $\mathcal{X} \in \mathbb{R}^{P_1 \times \dots \times P_N}$ to a sparse domain before learning dictionaries and model parameters for predicting a tensor $\mathcal{Y}$ from a large tensor $\mathcal{X}$.

We could define the compressed sensing extension to the T-TDDL as,

$$\underset{\substack{\{\boldsymbol{D}^{(1)},\cdots,\boldsymbol{D}^{(N)}\}\in\mathcal{D},\\ \{\boldsymbol{W}^{(1)},\cdots,\boldsymbol{W}^{(N)}\}\in\mathcal{W}\\ \{\boldsymbol{Z}^{(1)},\cdots,\boldsymbol{Z}^{(N)}\}\in\mathcal{Z}}}{\arg\min} \; f(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) + \frac{v_1}{2}\|\boldsymbol{W}\|_2 + \frac{v_2}{2}\|\boldsymbol{Z}\|_2 \tag{7.40}$$

Where $\mathcal{Z}$ is a convex set, $\boldsymbol{Z} \in \mathbb{R}^{M\times P}$ is a Kronecker sensing matrix with $\boldsymbol{Z} = \boldsymbol{Z}^{(N)} \otimes \cdots \otimes \boldsymbol{Z}^{(1)}$, $\boldsymbol{Z}^{(n)} \in \mathbb{R}^{M_n \times P_n}; \forall n \in \{1,\cdots,N\}$, $\boldsymbol{D} \in \mathbb{R}^{M\times U}$ and $\boldsymbol{D}^{(n)} \in \mathbb{R}^{M_n\times U_n}; \forall n \in \{1,\cdots,N\}$.

The convex function $f(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})$ is defined as

$$f(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) \triangleq (1-\mu)\mathbb{E}_{\mathcal{Y},\mathcal{X}}\left[l_{cs}\left(\mathcal{Y},\boldsymbol{W},\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D},\boldsymbol{Z})\right)\right] + \mu\mathbb{E}_{\mathcal{X}}[l_{cu}(\mathcal{X},\boldsymbol{D},\boldsymbol{Z})] \tag{7.41}$$

For notational simplicity, we denote the compressed sensing-based supervised loss function $l_{cs}\left(\mathcal{Y},\boldsymbol{W},\boldsymbol{\alpha}^*(\mathcal{X},\boldsymbol{D},\boldsymbol{Z})\right)$ as l_{cs} and the unsupervised loss function $l_{cu}(\mathcal{X},\boldsymbol{D},\boldsymbol{Z})$ as l_{cu}.

Let $\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})$ be the objective function of the compressed sensing extension of the T-TDDL, where

$$\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) = f(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) + \frac{v_1}{2}\|\boldsymbol{W}\|_2 + \frac{v_2}{2}\|\boldsymbol{Z}\|_2 \tag{7.42}$$

Therefore we could obtain the $\nabla_{\boldsymbol{Z}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})$ as

$$\nabla_{\boldsymbol{Z}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) = (1-\mu)\nabla_{\boldsymbol{Z}^{(n)}}l_{cs} + \mu\nabla_{\boldsymbol{Z}^{(n)}}l_{cu} + \frac{v_2}{2}\nabla_{\boldsymbol{Z}^{(n)}}\|\boldsymbol{Z}\|_2 \tag{7.43}$$

By applying Proposition 7.1, we could obtain $\nabla_{\boldsymbol{Z}^{(n)}}l_{cs}$ as a function of $\nabla_{\boldsymbol{Z}}l_{cs}$.

$$\left[\nabla_{\boldsymbol{Z}^{(n)}} l_{\mathrm{cs}}\right]_{i,j} = Tr\left((\nabla_{\boldsymbol{Z}} l_{cs})^T \frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{Z}_{i,j}^{(n)}} \right) \tag{7.44}$$

In [38], it shows that, $\nabla_{\boldsymbol{Z}} l_{cs} = \boldsymbol{D}\boldsymbol{\beta}^{*}\mathrm{vec}(\mathcal{X})^T$, Where $\boldsymbol{\beta}^{*}_{I^c} = 0$ and $\boldsymbol{\beta}^{*}_{I} = (\boldsymbol{D}_I^T\boldsymbol{D}_I + \lambda_2\boldsymbol{I})^{-1}\nabla_{\boldsymbol{\alpha}_I} l_{\mathrm{cs}}$. Therefore, by applying $\nabla_{\boldsymbol{Z}} l_{cs}$ to (7.44),

$$\left[\nabla_{\boldsymbol{Z}^{(n)}} l_{\mathrm{cs}}\right]_{i,j} = Tr\left((\boldsymbol{D}\boldsymbol{\beta}^{*})^T \left(\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{Z}_{i,j}^{(n)}} \mathrm{vec}(\mathcal{X}) \right)\right) \tag{7.45}$$

Where $Tr(AB) = Tr(BA)$.

Therefore, by using Proposition 7.3 on (7.45) we obtain the gradient of the supervised loss function $\nabla_{\boldsymbol{Z}^{(n)}} l_{cs}$ as,

$$\nabla_{\boldsymbol{Z}^{(n)}} l_{\mathrm{cs}} = (\boldsymbol{D}\boldsymbol{\beta}^{*})_{(n)} \left(\mathcal{X}_{(n)} \boldsymbol{\Psi}^T_{\boldsymbol{Z}^{(n)}}\right)^T \tag{7.46}$$

Where $\boldsymbol{\Psi}_{\boldsymbol{Z}^{(n)}} = \boldsymbol{Z}^{(N)} \otimes \cdots \otimes \boldsymbol{Z}^{(n+1)} \otimes \boldsymbol{Z}^{(n-1)} \otimes \cdots \otimes \boldsymbol{Z}^{(1)}$

By applying Proposition 7.1, we could obtain $\nabla_{\boldsymbol{Z}^{(n)}} l_{cu}$ as a function of $\nabla_{\boldsymbol{Z}} l_{cu}$.

$$\left[\nabla_{\boldsymbol{Z}^{(n)}} l_{cu}\right]_{i,j} = Tr\left((\nabla_{\boldsymbol{Z}} l_{cu})^T \frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{Z}_{i,j}^{(n)}} \right) \tag{7.47}$$

Since $\nabla_{\boldsymbol{Z}} l_{cu} = \left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D}\boldsymbol{\alpha}^{*'}\right)\mathrm{vec}(\mathcal{X}')^T$ and $Tr(AB) = Tr(BA)$

$$\left[\nabla_{\boldsymbol{Z}^{(n)}} l_{cu}\right]_{i,j} = Tr\left(\left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D}\boldsymbol{\alpha}^{*'}\right)^T \frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{Z}_{i,j}^{(n)}} \left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D}\boldsymbol{\alpha}^{*'}\right) \right) \tag{7.48}$$

Therefore, by using Proposition 7.3 on (7.48) we obtain the gradient of the unsupervised loss function $\nabla_{\boldsymbol{Z}^{(n)}} l_{cu}$ as,

$$\nabla_{\boldsymbol{Z}^{(n)}} l_{cu} = \left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D}\boldsymbol{\alpha}^{*'}\right)_{(n)} \left(\mathcal{X}'_{(n)} \boldsymbol{\Psi}_{\boldsymbol{Z}^{(n)}}\right)^T \tag{7.49}$$

By applying Proposition 7.2 to $\frac{v_2}{2}\nabla_{\boldsymbol{Z}^{(n)}} \|\boldsymbol{Z}\|_2$ we obtain,

$$\frac{v_2}{2}\nabla_{\boldsymbol{Z}^{(n)}} \|\boldsymbol{Z}\|_2 = v_2 \gamma_{\boldsymbol{Z}^{(n)}} \boldsymbol{Z}^{(n)} \tag{7.50}$$

where,

$$\gamma_{\boldsymbol{Z}^{(n)}} = \prod_{m=1,m\neq n}^{N} Tr\left(\boldsymbol{Z}^{(m)^T}\boldsymbol{Z}^{(m)}\right)$$

Therefore, the gradient of the objective function $\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})$ w.r.t. $\boldsymbol{Z}^{(n)}; \forall n \in \{1,\cdots,N\}$ is,

$$\nabla_{\boldsymbol{Z}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) = \begin{pmatrix} (1-\mu)\left((\boldsymbol{D\beta}^*)_{(n)}\left(\mathcal{X}_{(n)}\boldsymbol{\Psi}_{\boldsymbol{Z}^{(n)}}\right)^T\right) \\ +\,\mu\left(\left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime}\right)_{(n)}\left(\mathcal{X}'_{(n)}\boldsymbol{\Psi}_{\boldsymbol{Z}^{(n)}}\right)^T\right) \\ +\,v_2\gamma_{\boldsymbol{Z}^{(n)}}\boldsymbol{Z}^{(n)} \end{pmatrix} \tag{7.51}$$

Similarly, we could obtain the gradient of $\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})$ w.r.t. $\boldsymbol{W}^{(n)}; \forall n \in \{1,\cdots,N\}$ and $\boldsymbol{D}^{(n)}; \forall n \in \{1,\cdots,N\}$ as,

$$\nabla_{\boldsymbol{W}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) = (1-\mu)\nabla_{\boldsymbol{W}^{(n)}}l_{cs} + v_1\gamma_{\boldsymbol{W}^{(n)}}\boldsymbol{W}^{(n)} \tag{7.52}$$

and

$$\nabla_{\boldsymbol{D}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z}) = \begin{pmatrix} (1-\mu)\begin{pmatrix} -\,(\boldsymbol{D\beta}^*)_{(n)}\left(\boldsymbol{\alpha}^*{}_{(n)}\boldsymbol{\Psi}_n{}^T\right)^T \\ +\,(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}) - \boldsymbol{D\alpha}^*)_{(n)}\left(\boldsymbol{\beta}^*{}_{(n)}\boldsymbol{\Psi}_n{}^T\right)^T \end{pmatrix} \\ -\mu\left(\left(\boldsymbol{Z}\mathrm{vec}(\mathcal{X}') - \boldsymbol{D\alpha}^{*\prime}\right)_{(n)}\left(\boldsymbol{\alpha}^{*\prime}{}_{(n)}\boldsymbol{\Psi}_n{}^T\right)^T\right) \end{pmatrix} \tag{7.53}$$

At each iteration of the compressed sensing extension of the T-TDDL algorithm, we update the *mode-n* parameters $\boldsymbol{W}^{(n)}$, *mode-n* dictionaries $\boldsymbol{D}^{(n)}$and *mode-n* sensing matrices $\boldsymbol{Z}^{(n)}$ by a projected gradient step.

$$\boldsymbol{W}^{(n)} \leftarrow \Pi_{\mathcal{W}}\left[\boldsymbol{W}^{(n)} - \rho_t\left(\nabla_{\boldsymbol{W}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})\right)\right] \tag{7.54}$$

$$\boldsymbol{D}^{(n)} \leftarrow \Pi_{\mathcal{D}}\left[\boldsymbol{D}^{(n)} - \rho_t\left(\nabla_{\boldsymbol{D}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})\right)\right] \tag{7.55}$$

$$\boldsymbol{Z}^{(n)} \leftarrow \Pi_{\mathcal{Z}}\left[\boldsymbol{Z}^{(n)} - \rho_t\left(\nabla_{\boldsymbol{Z}^{(n)}}\mathrm{g}(\boldsymbol{D},\boldsymbol{W},\boldsymbol{Z})\right)\right] \tag{7.56}$$

Where $\Pi_{\mathcal{W}}$, $\Pi_{\mathcal{D}}$ and $\Pi_{\mathcal{Z}}$ are respective orthogonal projections on the convex sets $\mathcal{W}$, $\mathcal{D}$, and $\mathcal{Z}$.

7.6. Experimental Results

This section presents experimental results for the performance of the T-TDDL algorithm when used to solve supervised and semi-supervised multi-dimensional regression, binary classification, and multiclass classification problems.

We solved a multi-dimensional super-resolution task for our supervised and semi-supervised multi-dimensional regression experiments to obtain 4X upscaled videos from low-resolution videos. We obtained training and test videos from the vid4 dataset [118], a publicly available super-resolution dataset.

We obtained 3D-CT chest scan images from the MOSMEDDATA dataset [110], a 3D chest CT dataset with covid-19 related findings for our binary classification experiments. We used the compressed sensing extension of T-TDDL to learn task-driven *mode-n* dictionaries, *mode-n* sensing matrices, and *mode-n* model parameters to distinguish healthy people from patients with COVID-19 Pneumonia.

We obtained labeled 3D-CAD models belonging to ten different classes, the ModelNet10 dataset, from the Princeton ModelNet dataset [128] for our multiclass classification experiments. We used the compressed sensing extension of T-TDDL to learn task-driven *mode-n* dictionaries, *mode-n* sensing matrices, and *mode-n* model parameters to classify 3D-CAD models into ten classes.

We obtained our experimental results using a MATLAB implementation of T-TDDL on an MS-Windows machine: 2 Intel Xeon CPUs E5-2637 v4, 3.5GHz, 32GB RAM, and NVIDIA Tesla P100 GPU with 12GB memory.

7.6.1. Regression Experiment

This experiment uses the T-TDDL to learn *mode-n* dictionaries and *mode-n* model parameters for a multi-dimensional super-resolution task. We divided the used vid4 [118] 4X super-resolution dataset for training and testing, where each set contains two low-resolution color videos of 144×180 and 120×180 and two high-resolution ground-truth videos of 720×480 and 720×576, respectively. The low-resolution color videos have sixteen times fewer pixels than the high-resolution ground-truth videos.

In this experiment, we used the multi-dimensional regression loss function in section 7.4.1 as the supervised loss function $l_s(\mathcal{Y}, \boldsymbol{W}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D}))$ in both supervised and semi-supervised T-TDDL formulations. In the supervised T-TDDL formulation, we set $\mu = 0$ and in semi-supervised formulation we set $\mu = 0.1$.

We used T-TDDL to learn, three overcomplete task-driven *mode-n* dictionaries, $\boldsymbol{D}^{(1)} \in \mathbb{R}^{8\times10}$, $\boldsymbol{D}^{(2)} \in \mathbb{R}^{8\times10}$and $\boldsymbol{D}^{(3)} \in \mathbb{R}^{4\times5}$, and three *mode-n* model parameter matrices, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{32\times10}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{32\times10}$ and $\boldsymbol{W}^{(3)} \in \mathbb{R}^{4\times5}$, to predict a super-resolution tensor $\mathcal{Y} \in \mathbb{R}^{32\times32\times4}$ from a low-resolution tensor $\mathcal{X} \in \mathbb{R}^{8\times8\times4}$.

We set the hyperparameters of T-TDDL as $\upsilon = 10^{-5}$, $\lambda_2 = 0.001$, learning rate $\rho = 0.1$ and t_0 to 20% of the total iterations.

At each iteration of the T-TDDL, the tensor $\mathcal{X} \in \mathbb{R}^{8\times8\times4}$ was randomly sampled from the low-resolution training videos to obtain $8 \times 8 \times 4$ patches, and we set the tensor $\mathcal{Y} \in \mathbb{R}^{32\times32\times4}$ to the corresponding $32 \times 32 \times 4$ patch from its high-resolution training video.

Figure 7.1 and Figure 7.2 show our super-resolution experimental results for the two testing videos obtained using supervised T-TDDL formulation ($\mu = 0$) and semi-supervised T-TDDL formulation ($\mu = 0.1$).

Figure 7.1. a) Original low-resolution video, b) 4X super-resolution video obtained using supervised T-TDDL ($\mu = 0$), c) 4X super-resolution video obtained using semi-supervised T-TDDL ($\mu = 0.1$), d) High-resolution ground-truth video, and difference videos e) and f) (Super Resolution Experiment 1)

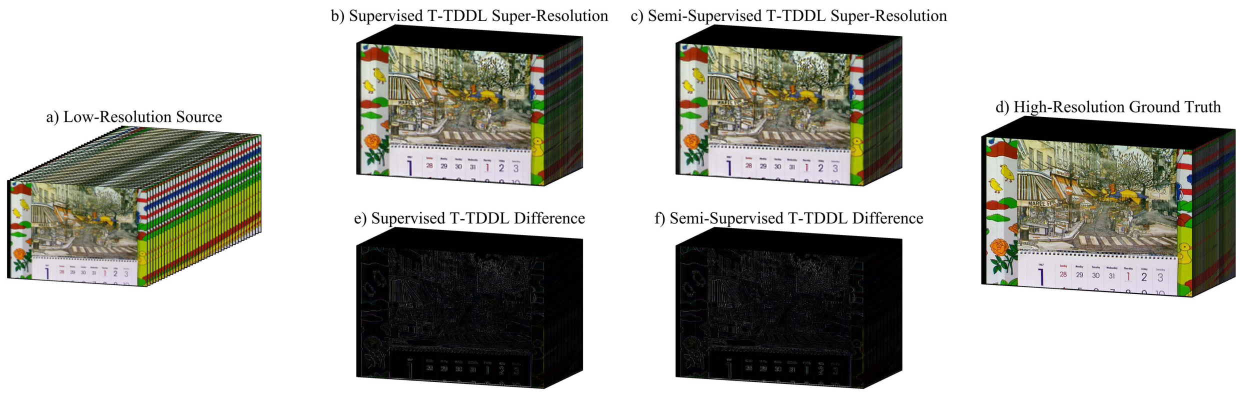

b) Supervised T-TDDL Super-Resolution
c) Semi-Supervised T-TDDL Super-Resolution
a) Low-Resolution Source
d) High-Resolution Ground Truth
e) Supervised T-TDDL Difference
f) Semi-Supervised T-TDDL Difference

Figure 7.2. a) Original low-resolution video, b) 4X super-resolution video obtained using supervised T-TDDL ($\mu = 0$), c) 4X super-resolution video obtained using semi-supervised T-TDDL ($\mu = 0.1$), d) High-resolution ground-truth video, and difference videos e) and f) (Super Resolution Experiment 2)

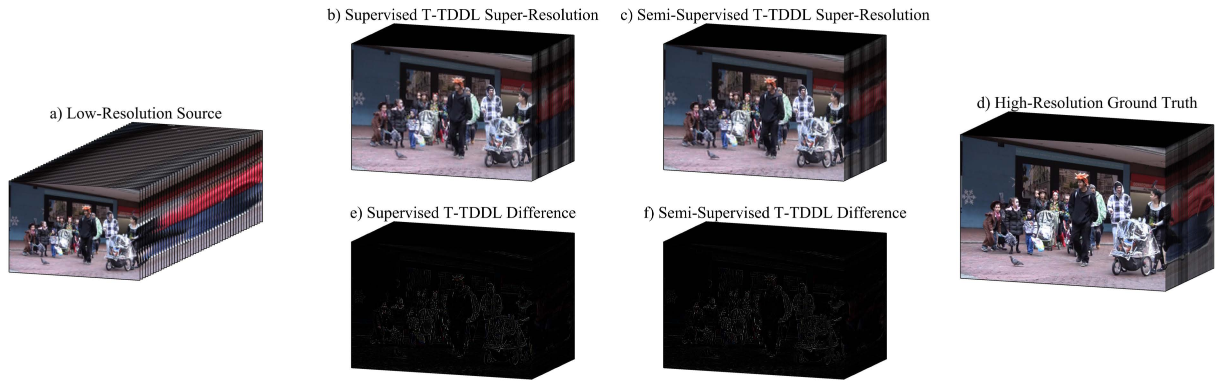

a) Low-Resolution Source
b) Supervised T-TDDL Super-Resolution
c) Semi-Supervised T-TDDL Super-Resolution
d) High-Resolution Ground Truth
e) Supervised T-TDDL Difference
f) Semi-Supervised T-TDDL Difference

Table 7.1 summarizes the performance comparison of Supervised T-TDDL, ($\mu = 0$), and Semi-Supervised T-TDDL($\mu = 0.1$) in obtaining 4X upscaled super-resolution videos for both super-resolution experiments 1 and 2. Compared performance metrics in Table 7.1 are $\|\mathcal{Y} - \hat{\mathcal{Y}}\|_2$, the norm of the difference between the ground truth video, $\mathcal{Y}$, and the super resolution video, $\hat{\mathcal{Y}}$, Measure of structural similarity (SSIM) [128], and Peak signal to noise ratio (PSNR). In $\|\mathcal{Y} - \hat{\mathcal{Y}}\|_2$, lower numbers indicate better results and in SSIM and PSNR higher numbers indicate better results.

Table 7.1. Comparison of Super-resolution experimental results for T-TDDL ($\mu = 0$) and Semi-Supervised T-TDDL ($\mu = 0.1$)

	Metric	T-TDDL ($\mu = 0$)	Semi-Supervised T-TDDL ($\mu = 0.1$)
Super-Resolution Experiment 1	$\|\mathcal{Y} - \hat{\mathcal{Y}}\|_2$	0.1957	**0.1950**
	PSNR	18.38	**18.42**
	SSIM	0.6066	**0.6083**
Super-Resolution Experiment 2	$\|\mathcal{Y} - \hat{\mathcal{Y}}\|_2$	0.1396	**0.1349**
	PSNR	24.09	**24.39**
	SSIM	0.7993	**0.8032**

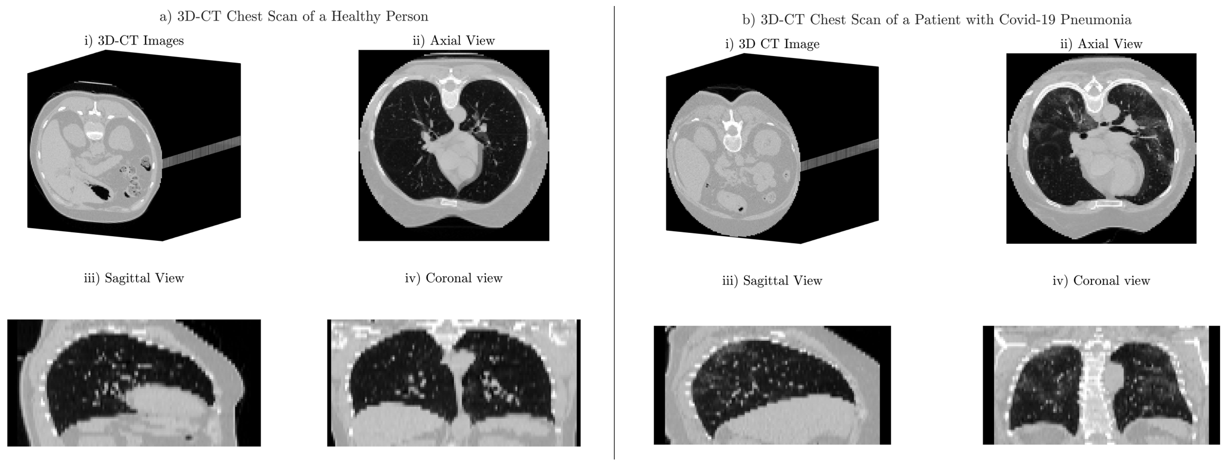
As shown in Table 7.1, the 4X video super-resolution experimental results obtained using the semi-supervised T-TDDL formulation ($\mu = 0.1$) outperforms the supervised T-TDDL formulation in all three metrics.

7.6.2. Binary Classification Experiment

In the T-TDDL binary classification experiment, we used the compressed sensing extension of the T-TDDL to learn task-driven *mode-n* dictionaries, *mode-n* model parameters, and *mode-n* sensing matrices to classify 3D-CT chest scans with COVID-19 Pneumonia and compared with the results obtained using a 3D Convolutional Neural Network(3D-CNN) model.

We obtained 3D-CT chest scan images from the MOSMEDDATA dataset [129], a 3D chest CT dataset with covid-19 related findings. Figure 7.3 shows samples of 3D-CT chest scans of a healthy person and a patient with COVID-19 Pneumonia from our dataset. The hazy gray patches (Increased CT Attenuation) in the Axial, Sagittal, and Coronal view of the Covid-19 patient's lungs in Figure 7.3 b) are called ground-glass opacities (GGO) [130], which indicate abnormalities in the lungs. In this experiment, we train T-TDDL and 3D-CNN to distinguish COVID-19 patients with Pneumonia (Figure 7.3 b)) from healthy people (Figure 7.3 a)) using the presence of ground-glass opacities (GGO) in 3D-CT chest scans.

Figure 7.3. 3D, Axial, Sagittal, Coronal view of a) 3D-CT chest scan of a healthy person and b) a 3D-CT chest scan of a COVID-19 patient with Pneumonia



Our dataset consisted of 100 labeled 3D-CT chest scans of patients with COVID-19 associated Pneumonia and 100 labeled 3D-CT chest scans of healthy people. We normalized the 3D-CT images after applying a threshold between -1000 and 400 and rescaled 3D-CT images to have $128 \times 128 \times 64$ voxels. Then, we split the 3D-CT images dataset into train and test with a 70:30 ratio. The training set consisted of 140 3D-CT images, and the testing set consisted of 60 3D-CT images. We augmented the training data during the training of both 3D-CNN and T-TDDL by rotating the 3D-CT images along the longitudinal axis by a random angle.

We used T-TDDL to learn three *mode-n* sensing matrices $\boldsymbol{Z}^{(1)} \in \mathbb{R}^{16\times128}$, $\boldsymbol{Z}^{(2)} \in \mathbb{R}^{16\times128}$and $\boldsymbol{Z}^{(3)} \in \mathbb{R}^{16\times64}$, three overcomplete task-driven *mode-n* dictionaries, $\boldsymbol{D}^{(1)} \in \mathbb{R}^{16\times32}$, $\boldsymbol{D}^{(2)} \in \mathbb{R}^{16\times32}$and $\boldsymbol{D}^{(3)} \in \mathbb{R}^{16\times32}$ and three *mode-n* model parameter vectors, $\boldsymbol{w}^{(1)} \in \mathbb{R}^{1\times32}$, $\boldsymbol{w}^{(2)} \in \mathbb{R}^{1\times32}$ and $\boldsymbol{w}^{(3)} \in \mathbb{R}^{1\times32}$, to predict the class y from 3D-CT image tensor $\mathcal{X} \in \mathbb{R}^{128\times128\times64}$. We used the *mode-n* sensing matrices $\boldsymbol{Z}^{(1)}, \boldsymbol{Z}^{(2)}$ and $\boldsymbol{Z}^{(3)}$ to project the image tensor $\mathcal{X} \in \mathbb{R}^{128\times128\times64}$ to a much smaller tensor $\mathcal{X}_z \in \mathbb{R}^{16\times16\times16}$ before using T-NET to obtain a sparse representation, which is much more computationally efficient than obtaining a sparse representation of $\mathcal{X} \in \mathbb{R}^{128\times128\times64}$ directly using T-NET.

We used the logistic regression loss function [38], as shown in section 7.4.2, as the supervised binary classification loss function $l_s\big(y, \boldsymbol{w}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D}, \boldsymbol{Z})\big)$ in a supervised T-TDDL formulation where $\mu = 0$. We set the hyperparameters of T-TDDL as $v = 10^{-5}$, $\lambda_2 = 0.001$, learning rate $\rho = 0.1$ and t_0 to 20% of the total iterations in one epoch. Each epoch consisted of a maximum of $100{,}000$ iterations, and we stopped training at each epoch when all *mode-n* sensing matrices, *mode-n* dictionaries, and *mode-n* model parameters were converged or when the maximum number of iterations was reached. We ran six epochs to obtain the training results.

We compared our T-TDDL binary classification results with the results obtained by training a 3D-Convolutional Neural Network Model (3D-CNN). For the 3D-CNN model, we used the 17-layer 3D-CNN architecture proposed by Zunair et al. [131] for classifying 3D-CT images. We trained the 3D-CNN for 100 epochs with early stopping to achieve the maximum classification accuracy.

Table 7.2 Binary Classification Report

		Precision	Recall	F_1-score	Support
3D-CNN	COVID-19 Pneumonia	0.69	**0.90**	**0.78**	30
	Normal	**0.86**	0.60	0.71	30
	accuracy			0.75	60
T-TDDL	COVID-19 Pneumonia	**0.77**	0.77	0.77	30
	Normal	0.77	**0.77**	**0.77**	30
	accuracy			**0.77**	60

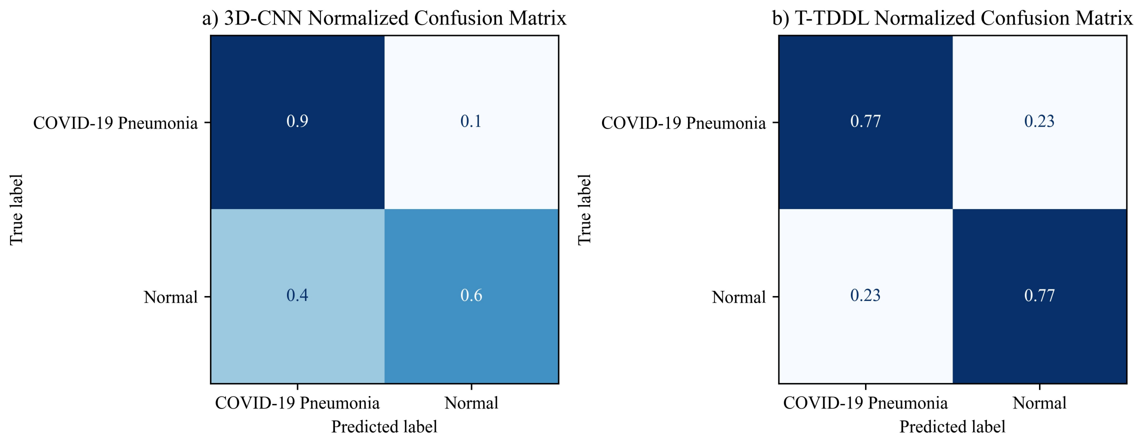
Table 7.2 shows the classification report for both 3D-CNN and T-TDDL. The precision of classification is given by $Precision = True\ Positive/(True\ Positive\ +\ False\ Positive)$, and the recall is $Recall = True\ Positive/(True\ Positive\ +\ False\ Negative)$. The F_1-score [132] is the harmonic mean of the precision and recall, which is given by,

$$F_1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

We measured the classification accuracy using the F_1-score, which is given in Table 7.2, where T-TDDL with compressed sensing extension achieved a classification accuracy of 0.77, whereas the 3D-CNN model only achieved a classification accuracy of 0.75, classifying 3D-CT images with COVID-19 Associated Pneumonia.

Figure 7.4 a) shows the normalized confusion matrix for the 3D-CNN binary classification experiment, and Figure 7.4 b) shows the normalized confusion matrix for the T-TDDL Binary Classification Experiment. The Confusion matrices in Figure 7.4 show that the 3D-CNN model is biased toward classifying 3D-CT images for the COVID-19 Pneumonia class, whereas the T-TDDL shows no special bias toward any class.

Figure 7.4. Normalized confusion matrices for a) 3D-CNN and b) T-TDDL Binary Classification Experiment



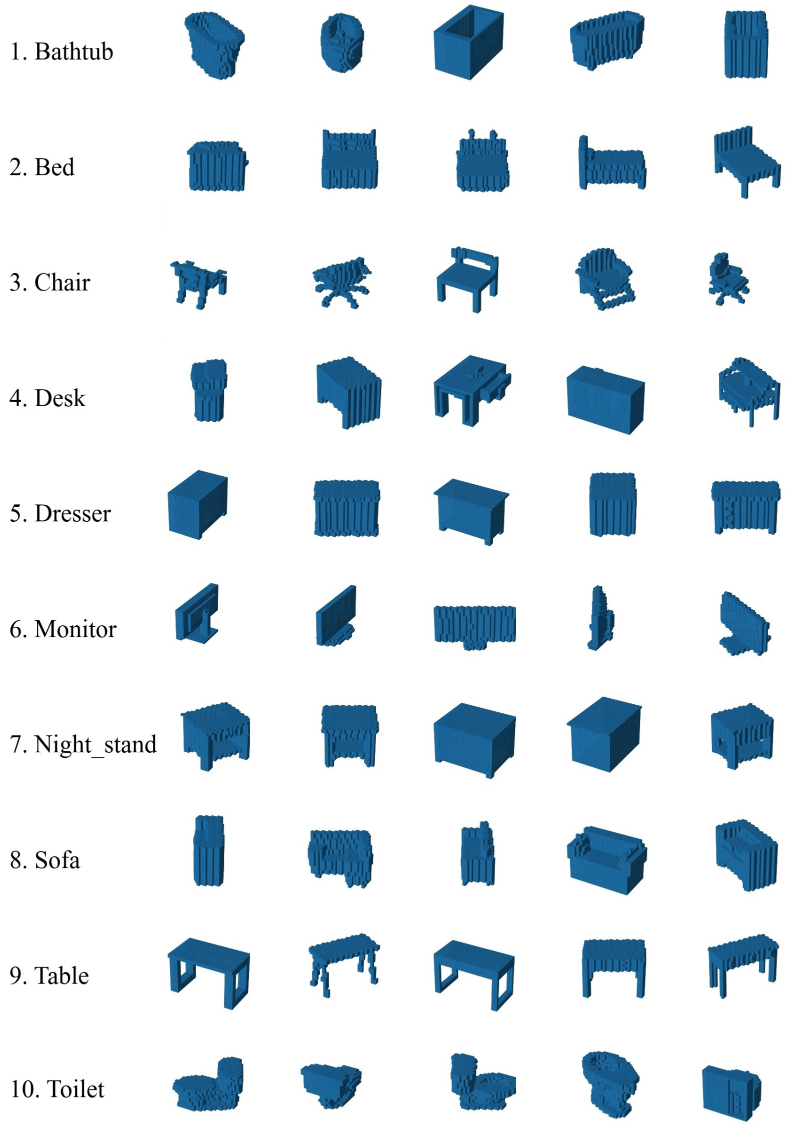
7.6.3. Multiclass Classification Experiment

In the T-TDDL multiclass classification experiment, we used the compressed sensing extension of the T-TDDL to learn task-driven *mode-n* dictionaries, *mode-n* model parameters, and *mode-n* sensing matrices to classify 3D-CAD models and compared the results with a 3D Convolutional Neural Network(3D-CNN) model.

We obtained 3D-CAD models, the ModelNet10 dataset, from the Princeton ModelNet dataset [133], consisting of 3D-CAD models belonging to ten different classes. Each 3D-CAD model had $30 \times 30 \times 30$ voxels. Our training dataset consisted of 47,892 labeled 3D-CAD models, and our testing dataset consisted of 10,896 labeled 3D-CAD models. Figure 7.5 shows sample 3D-CAD models for each class from the ModelNet10 dataset.

We used T-TDDL to learn three *mode-n* sensing matrices $\boldsymbol{Z}^{(1)} \in \mathbb{R}^{15\times30}$, $\boldsymbol{Z}^{(2)} \in \mathbb{R}^{15\times30}$and $\boldsymbol{Z}^{(3)} \in \mathbb{R}^{15\times30}$, three over-complete task-driven *mode-n* dictionaries, $\boldsymbol{D}^{(1)} \in \mathbb{R}^{15\times30}$, $\boldsymbol{D}^{(2)} \in \mathbb{R}^{15\times30}$and $\boldsymbol{D}^{(3)} \in \mathbb{R}^{115\times30}$ and three *mode-n* model parameter matrices, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{10\times30}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{1\times30}$ and $\boldsymbol{W}^{(3)} \in \mathbb{R}^{1\times30}$, to predict a class vector $\boldsymbol{y}$ from 3D-CAD model tensor $\mathcal{X} \in \mathbb{R}^{30\times30\times30}$. We used the *mode-n* sensing matrices $\boldsymbol{Z}^{(1)}, \boldsymbol{Z}^{(2)}$ and $\boldsymbol{Z}^{(3)}$ to project the tensor $\mathcal{X} \in \mathbb{R}^{30\times30\times30}$ to a much smaller tensor $\mathcal{X}_z \in \mathbb{R}^{15\times15\times15}$ before using T-NET to obtain a sparse representation, which is computationally efficient than obtaining a sparse representation of $\mathcal{X} \in \mathbb{R}^{30\times30\times30}$ directly using T-NET.

Figure 7.5 Sample 3D-CAD models from the ModelNet10 dataset



We used the Softmax cross-entropy loss function, as shown in section 7.4.3, as the supervised multiclass classification loss function $l_s\big(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}^*(\mathcal{X}, \boldsymbol{D}, \boldsymbol{Z})\big)$ in a supervised T-TDDL formulation where $\mu = 0$. We set the hyperparameters of T-TDDL as $v = 10^{-5}$, $\lambda_2 = 0.001$, learning rate $\rho = 0.1$ and t_0 to 20% of the total iterations in one epoch. Each epoch consisted of a maximum of 100,000 iterations, and we stopped training at each epoch when all *mode-n* sensing matrices, *mode-n* dictionaries, and *mode-n* model parameters were converged or when the maximum number of iterations was reached. We ran ten epochs to obtain the training results.

We compared our T-TDDL multiclass classification results with the results obtained by training a 3D-Convolutional Neural Network Model (3D-CNN). For the 3D-CNN model, we used the 14-layer 3D-CNN model for classifying 3D-CAD models. We trained the 3D-CNN for 100 epochs with early stopping to achieve the maximum classification accuracy.

Table 7.3 Multiclass Classification Report

	Class	Precision	Recall	F_1-score	Support
3D-CNN	Bathtub	0.72	**0.92**	**0.8**	600
	Bed	**0.93**	**0.99**	**0.96**	1200
	Chair	0.64	**0.99**	0.78	1200
	Desk	0.62	**0.81**	**0.7**	1032
	Dresser	**0.76**	**0.84**	**0.8**	1032
	Monitor	**0.95**	**0.99**	**0.97**	1200
	Night Stand	**0.73**	**0.76**	**0.74**	1032
	Sofa	0.92	**0.97**	**0.95**	1200
	Table	0.88	0.67	0.76	1200
	Toilet	0	0	0	1200
	Accuracy			0.7861	10896
T-TDDL	Bathtub	**0.85**	0.69	0.76	600
	Bed	0.76	0.94	0.84	1200
	Chair	**0.85**	0.9	**0.87**	1200
	Desk	**0.68**	0.65	0.66	1032
	Dresser	0.75	0.82	0.78	1032
	Monitor	0.92	0.91	0.91	1200
	Night Stand	0.72	0.7	0.71	1032
	Sofa	**0.95**	0.81	0.87	1200
	Table	**0.9**	**0.76**	**0.83**	1200
	Toilet	**0.79**	**0.88**	**0.83**	1200
	Accuracy			**0.8149**	10896

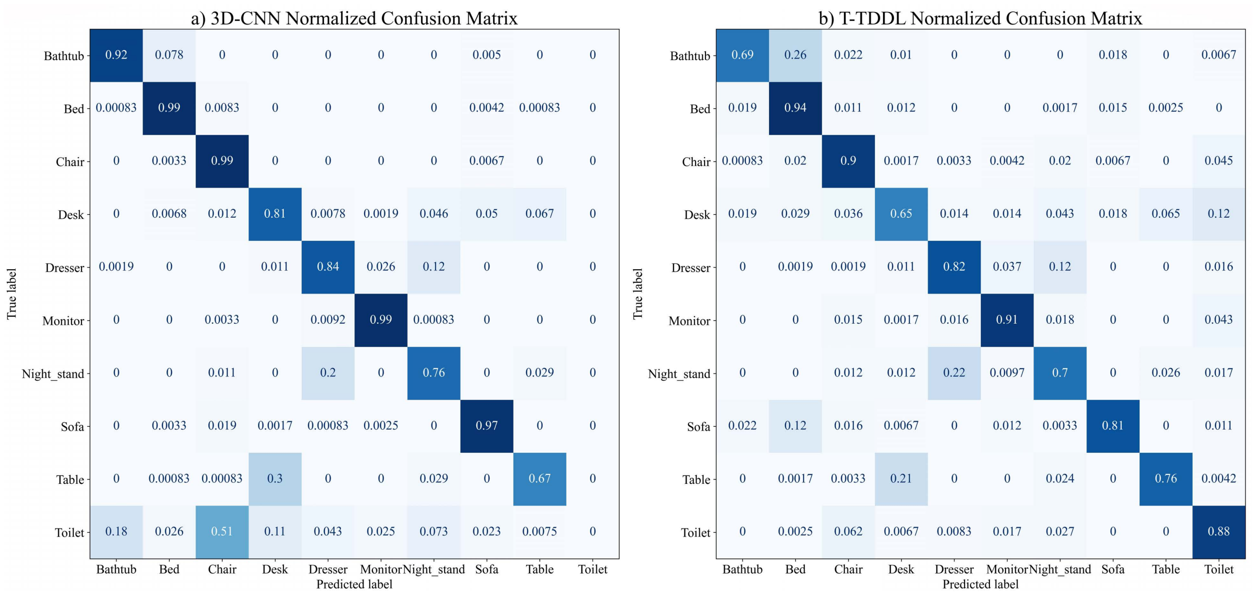
Table 7.3 shows the classification report for both 3D-CNN and T-TDDL. The precision of classification is $Precision = True\ Positive/(True\ Positive\ +\ False\ Positive)$, and the recall is $Recall = True\ Positive/(True\ Positive\ +\ False\ Negative)$. The F_1-score [132] is the harmonic mean of the precision and recall, which is given by,

$$F_1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

We measured the classification accuracy using the F_1-score, which is given in Table 7.3, where T-TDDL with compressed sensing extension achieved a classification accuracy of 0.8149, whereas the 3D-CNN model only achieved a classification accuracy of 0.7861, classifying 3D-CAD models.

Figure 7.6 a) shows the normalized confusion matrix for the 3D-CNN multiclass classification experiment, and Figure 7.6 b) shows the normalized confusion matrix for the T-TDDL multiclass classification experiment. The Confusion matrices in Figure 7.6 show that the 3D-CNN model failed to predict the Toilet class and shows a bias toward some classes, whereas the T-TDDL shows no special bias toward any class.

Figure 7.6. Normalized confusion matrices for a) 3D-CNN and b) T-TDDL Multiclass Classification Experiment



7.7. Conclusions

This chapter extended the one-dimensional TDDL formulation to develop the tensor task-driven dictionary learning(T-TDDL) framework that could work as an efficient online data-driven or task-driven dictionary learning algorithm for supervised and semi-supervised learning of *mode-n* dictionaries and *mode-n* model parameters. We have also presented a compressed sensing extension to the T-TDDL formulation for efficiently solving large tensor task-driven dictionary learning problems. Section IV presented three example applications in Regression, Binary Classification, Multiclass classification, and the gradient calculations of the respective loss functions.

We solved a 4X video super-resolution task for the T-TDDL regression experiment, using both supervised T-TDDL formulation $(\mu = 0)$ and the semi-supervised T-TDDL formulations $(\mu = 0.1)$. Our supervised T-TDDL regression formulation achieved SSIM [128] of 0.6066 and 0.7993 for the two example videos. The semi-supervised T-TDDL regression formulation achieved SSIM of 0.6083 and 0.8032, respectively, outperforming the experimental results of the supervised T-TDDL regression formulation.

The compressed sensing extension of the T-TDDL binary classification formulation achieved a higher classification accuracy, F_1-score of 0.77, compared to the 3D-CNN model, which achieved the F_1-score of 0.75 in classifying the 3D-CT chest scans images of COVID-19 patients with Pneumonia from a labeled 3D-CT chest scans images dataset. In our binary classification experiment, the T-TDDL formulation showed no bias towards any class, whereas the 3D-CNN model showed a bias towards the 3D-CT images of patients with COVID-19 Pneumonia.

Also, the compressed sensing extension of the T-TDDL multiclass classification formulation achieved a higher classification accuracy, F_1-score of 0.8149, compared to the 3D-CNN model, which achieved the F_1-score of 0.7861 in classifying the 3D-CAD models of the ModelNet10 dataset. Therefore the T-TDDL framework could be used for accurately solving multi-dimensional classification problems.

Unlike CNN, the T-TDDL framework could be used for solving N dimensional regression or classification problems without extra modification. Therefore, the T-TDDL framework could be used for solving tensor task-driven dictionary learning problems accurately and efficiently.

Multi-modal learning is gaining much popularity in machine learning communities. The T-TDDL formulation could easily extend to efficiently solve tensor multi-modal task-driven dictionary learning problems [120]. Furthermore, the T-TDDL formulation could also extend to solve tensor task-driven dictionary learning problems in an agent-based distributed online setting using the formulations given by Koppel et al. [121].

Therefore, with various loss functions and formulations, the T-TDDL framework could be used to efficiently solve a wide range of tensor task-driven dictionary learning problems and online data-driven dictionary learning problems.

Chapter 8

8. Conclusions And Future Directions

8.1. Conclusions

This research's main objective was to develop novel methodologies that could efficiently solve large multi-dimensional problems using available limited computational resources. We researched several topics such as Sparsity, Tensors, and Multilinear Algebra to achieve this.

Many signal processing, machine learning, and statistical applications solve multi-dimensional problems using linear algebra after vectorizing multi-dimensional signals. As the number of dimensions increases, multi-dimensional signals quickly grow in size, and solving such problems becomes computationally infeasible.

Therefore, we looked into obtaining a sparse signal representation of large multi-dimensional signals, which results in simpler and faster processing and less memory storage requirements. However, obtaining a sparse signal representation of large multi-dimensional signals by solving a sparse linear least-square problem also requires a significantly large amount of computational resources. As the size of the multi-dimensional signal increases, it quickly becomes computationally infeasible.

A multilinear representation of a tensor (multi-dimensional signal) is obtained by multiplying each mode of the tensor by a smaller *mode-n* matrix, which requires significantly less computational resources. An earlier generalization of OMP, known as *Kronecker-OMP* [16], was developed to solve the L_0 constrained least-squares problem for large multi-dimensional signals. We developed the *Tensor Least Angle Regression* (T-LARS), a generalization of *Least Angle Regression (LARS)*, to efficiently solve large L_0 or L_1 constrained multilinear least-squares problems (underdetermined or overdetermined) for all critical values of the regularization parameter λ.

To demonstrate the validity and performance of our T-LARS algorithm, we used it to successfully obtain different sparse representations of two relatively large 3D brain images, using fixed and learned separable over-complete dictionaries, by solving both L_0 and L_1 constrained sparse least-squares problems. Our numerical experiments demonstrate that our T-LARS algorithm is significantly faster (46 - 70 times) than Kronecker-OMP in obtaining K-sparse solutions for multilinear least-squares problems. However, the K-sparse solutions obtained using Kronecker-OMP always have a slightly lower residual error (1.55% - 2.25%) than ones obtained by T-LARS. We also presented experimental results to compare Kronecker-OMP and T-LARS in obtaining a sparse representation of 3D brain images using compressed sensed samples by solving a Kronecker compressed sensing problem. Therefore, T-LARS could be an important tool for numerous multi-dimensional signal processing and regression applications.

Sparse weighted multilinear least-squares is a generalization of the sparse multilinear least-squares problem, where prior information about, e.g., parameters and data is incorporated by multiplying both sides of the original problem by a typically diagonal weights matrix [22]. If the diagonal weight matrix has a similar Kronecker structure to the dictionary matrix, we could use the T-LARS algorithm developed in chapter 3 to solve this problem efficiently. Typically, introducing arbitrary diagonal weights would result in a non-Kronecker least-squares problem that could be very large to store or solve practically. Therefore, we introduced the *Weighted Tensor Least Angle Regression* (WT-LARS) algorithm, which could efficiently solve the weighted tensor least-squares problem for an arbitrary diagonal weight matrix. In the experimental results, we solved the image inpainting problems using WT-LARS by obtaining sparse representations of RGB images using binary-weighted samples to demonstrate the validity of WT-LARS. We successfully obtained sparse representations of RGB images behind fences, using 10% nonzero coefficients, and a sparse representation of a landscape image occluded by a person, using 20% nonzero coefficients.

We could initialize T-LARS with an L_1 solution located on the Pareto curve [23] and obtain an L_1 solution with a lower residual error, where the Pareto curve contains every solution to a linear/multilinear least-squares problem. However, we could not initialize T-LARS with any solution outside of the Pareto curve because it will violate the optimality conditions of T-LARS. Therefore, we developed the *Tensor Dynamic Least Angle Regression* (TD-LARS) algorithm, a multilinear generalization of the L_1-Homotopy algorithm [24] to efficiently solve multilinear L_1 minimization problems using nonzero initial solutions located on or off of the Pareto curve. Our experimental results show that TD-LARS obtains the solution to an L_1 minimization problem much faster than solving it from the $\mathcal{X} = 0$ initial solution using T-LARS when a close initial solution is available.

L_0 minimization problem is a non-convex problem, and the slightly relaxed L_1 minimization problem is a convex problem. Even though, the L_2 minimization problem is strictly convex; it does not provide a sparse solution. Also, both L_0 and L_1 minimization problems have an upper limit for selecting the maximum number of coefficients for a unique and accurate solution based on the dictionary's coherence. Also, the L_0 or L_1 minimization problem does not have the group selection ability, which is important in certain applications.

Therefore in this thesis, we introduced the Multilinear Elastic Net problem by generalizing the one-dimensional Elastic Net problem [27], [28]. Multilinear Elastic Net solves a strictly convex L_1 and L_2 constrained multilinear least-squares problem and it has, the best properties of both L_1 and L_2 minimization problems. In addition to the group selection ability, Multilinear Elastic Net can obtain more than n nonzero coefficients for a signal with n elements. Therefore, Multilinear Elastic Net is ideal for solving multilinear sparse least-squares problems with highly coherent dictionaries.

The dictionary in Multilinear Elastic Net problem has a partitioned Kronecker structure, which could not be efficiently solved with T-LARS. Therefore, we introduced the *Tensor Elastic Net* (T-NET) algorithm to efficiently solve the Multilinear Elastic Net problem by utilizing the partitioned Kronecker structure of the dictionary matrix. Experimental results show T-NET has better statistical properties than T-LARS, such as the group selection ability and ability to solve problems with highly coherent dictionaries.

We could use fixed or learned separable dictionaries in obtaining a sparse multilinear representation of multi-dimensional signals using T-LARS, WT-LARS, TD-LARS, T-NET, or Kronecker-OMP. However, the dictionaries learned from the data are much more efficient in obtaining sparse representations than fixed dictionaries [29].

Roemer et al. [30] introduced Tensor Method of Optimal Directions (T-MOD) and Kronecker Higher-Order SVD (K-HOSVD) algorithms to learn data-driven separable dictionaries to solve multilinear problems by generalizing one-dimensional data-driven dictionary learning algorithms, Method of Optimal Direction(MOD) [31], and K-SVD [32], respectively. Roemer used one-dimensional sparse coding methods in the sparse coding step of the T-MOD and K-HOSVD algorithms, requiring a significantly large amount of computational resources for solving data-driven tensor dictionary learning problems. However, we could solve large data-driven tensor dictionary learning problems efficiently by using T-LARS [18], T-NET, or Kronecker-OMP [16] in the sparse coding step of the T-MOD and K-HOSVD algorithms.

Learned dictionaries could be used in classification or regression tasks [33]–[35]. However, regression and classification performance could be improved significantly by supervised learning of task-specific dictionaries [36], [37].

This thesis extended the one-dimensional TDDL formulation [38] to develop the *Tensor Task-Driven Dictionary Learning* (T-TDDL) that could work as an efficient online data-driven or task-driven dictionary learning algorithm for supervised and semi-supervised learning of *mode-n* dictionaries and *mode-n* model parameters. We have also presented a compressed sensing extension to the T-TDDL formulation for efficiently solving large tensor task-driven dictionary learning problems. Section 7.4 presented example applications in Regression, Binary Classification, and Multiclass classification, and the gradient calculations of the respective loss functions.

A supervised, semi-supervised, or unsupervised T-TDDL formulations could be obtained depending on the value of μ. The unsupervised tensor dictionary learning formulation could be used to solve the online tensor data-driven dictionary learning problems [30], [122].

To demonstrate the validity and performance of our T-TDDL framework, we used the T-TDDL framework to solve multi-dimensional regression, binary classification, and multiclass classification problems and presented experimental results. The experimental results show that the 4X super-resolution videos (4D tensor), obtained using the semi-supervised T-TDDL regression formulation, outperform the supervised T-TDDL regression formulations. In the binary classification experiment, we used the compressed sensing extension of the T-TDDL to learn *mode-n* dictionaries, *mode-n* model parameters, and *mode-n* sensing matrices to classify 3D-CT chest scan images of patients with COVID-19 associated pneumonia from 3D-CT chest scan images of healthy people. We compared our T-TDDL results with the results obtained using a 17-layer 3D-CNN model designed explicitly for classifying 3D-CT images. We used the F_1-score to measure the classification accuracy. Our T-TDDL formulation achieved a binary classification accuracy of 0.77, which is higher than the binary classification accuracy of 0.75 achieved by the 3D-CNN model. Furthermore, the compressed sensing extension of the T-TDDL multiclass classification achieved a higher classification accuracy of 0.8149 than the 3D-CNN model, which only achieved a classification accuracy of 0.7861 in classifying the 3D-CAD models of the ModelNet10 dataset.

Unlike CNN, the T-TDDL formulation could be used for solving N dimensional regression or classification problems without extra modification. Therefore, the T-TDDL framework could be efficiently used for a wide range of multi-dimensional machine learning applications with various task-specific loss functions.

8.2. Future Directions

This research introduced several tensor-based algorithms for efficiently obtaining a sparse multilinear representation of multi-dimensional signals under different conditions. To demonstrate the validity and performance of our algorithms, we presented a few example applications for each algorithm.

Most state-of-the-art applications in Signal Processing, Machine Learning, and Statistics currently solve multi-dimensional problems using linear algebra after vectorizing multi-dimensional signals. The concepts and algorithms introduced in this research enable using tensors and multilinear algebra in future large multi-dimensional applications.

T-LARS, TD-LARS, WT-LARS, and T-NET could be used to solve large multi-dimensional problems such as multi-dimensional regression, representation, and compression problems in various fields. Also, the TD-LARS algorithm could be used for transfer learning applications in multi-dimensional regression.

The T-LARS and T-NET algorithms could be extended to solve multi-dimensional problems with grouped variables by extending group LARS [134] and group adaptive Elastic Net [135]. Also, T-LARS and convex-LAR [136] could be extended to develop a generalized algorithm to obtain a sparse solution for any multilinear convex loss function.

One of the limitations of T-LARS, TD-LARS, WT-LARS, and T-NET is that, as the number of selected active coefficients increases, the Gram matrix gets bigger and increases the usage of computational resources resulting in longer execution time. Therefore, available computational resources limit the number of active coefficients that could be selected. Since the Gram matrix is symmetric, in our algorithms, we store the lower triangle part of the Gram matrix in memory and use it in computations for faster processing. As a future research direction, one could improve the above algorithms or develop novel algorithms to solve the sparse multilinear least-squares problems without explicitly building and storing large Gram matrices.

The T-TDDL is a general multi-dimensional framework for learning online multi-dimensional data-driven or task-driven *mode-n* dictionaries, *mode-n* model parameters, and *mode-n* sensing matrices. T-TDDL could be used for many multi-dimensional machine learning and signal processing applications using different task-specific loss functions. The compressed sensing extension allows solving huge multi-dimensional problems using T-TDDL, where *mode-n* dictionaries and *mode-n* model parameters could be learned after projecting the input, using *mode-n* sensing matrices, to a smaller manageable size.

Multi-modal learning is gaining much popularity in machine learning communities. The T-TDDL formulation could easily extend to efficiently solve tensor multi-modal task-driven dictionary learning problems [120]. Furthermore, the T-TDDL formulation could also extend to solve tensor task-driven dictionary learning problems in an agent-based distributed online setting using the formulations given by Koppel et al. [121].

The initial motivation behind this research was to solve the full-wave simulation of light propagation inside a 5mm×5mm×5mm tissue sample, when a light source with $1\mu m$ central wavelength is used, which requires solving the scalar scattering equation for a volume of $5000\lambda \times 5000\lambda \times 5000\lambda$. Solving the linear system, resulting from the scalar scattering equations when Method of Moments (MoM) is applied, was computationally infeasible even with the available supercomputers. This Ph.D. research has laid the foundation and developed tensor-based methods for efficiently solving such large multidimensional problems. Therefore, as a future direction, one could use our tensor-based methods to solve the full-wave simulation of light propagation inside a tissue problem for a large tissue sample.

References

[1] W. P. Elderton and N. L. Johnson, “Method of Moments,” *Syst. Freq. Curves*, vol. 82, pp. 12–34, 2010, doi: 10.1017/cbo9780511569654.004.

[2] M. M. Ney, “Method of Moments as Applied to Electromagnetic Problems,” *IEEE Trans. Microw. Theory Tech.*, vol. MTT-33, no. 10, pp. 972–980, 1985, doi: 10.1109/TMTT.1985.1133158.

[3] K. Maleknejad and M. Nosrati Sahlan, “The method of moments for solution of second kind Fredholm integral equations based on B-spline wavelets,” *Int. J. Comput. Math.*, vol. 87, no. 7, pp. 1602–1616, 2010, doi: 10.1080/00207160802406523.

[4] D. Reinsel, J. Gantz, and J. Rydning, “Data Age 2025: The Digitization of the World From Edge to Core,” *Seagate, IDC*, no. November, p. 28, 2018, Accessed: Mar. 15, 2021. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[5] T. G. Kolda and B. W. Bader, “Tensor Decompositions and Applications,” *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009, doi: 10.1137/07070111X.

[6] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, “Tensor Decomposition for Signal Processing and Machine Learning,” *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017, doi: 10.1109/TSP.2017.2690524.

[7] A. Cichocki *et al.*, “Tensor decompositions for signal processing applications: From two-way to multiway component analysis,” *IEEE Signal Processing Magazine*, vol. 32, no. 2. pp. 145–163, 2015, doi: 10.1109/MSP.2013.2297439.

[8] S. Mallat, *A Wavelet Tour of Signal Processing*. Elsevier, 2009.

[9] I. Wickramasingha, M. Sobhy, and S. S. Sherif, “Sparsity in Bayesian Signal Estimation,” in *Bayesian Inference*, vol. 37, no. 2, J. P. Tejedor, Ed. InTech, 2017.

[10] D. L. Donoho, “For most large underdetermined systems of linear equations the minimal ℓ 1-norm solution is also the sparsest solution,” *Commun. Pure Appl. Math.*, vol. 59, no. 6, pp. 797–829, Jun. 2006, doi: 10.1002/cpa.20132.

[11] D. L. Donoho and M. Elad, “Optimally sparse representation in general (nonorthogonal) dictionaries via 1 minimization,” *Proc. Natl. Acad. Sci.*, vol. 100, no. 5, pp. 2197–2202, Mar. 2003, doi: 10.1073/pnas.0437847100.

[12] S. G. Mallat and Z. Zhang, “Matching Pursuits With Time-Frequency Dictionaries,” *IEEE*

Trans. Signal Process., vol. 41, no. 12, pp. 3397–3415, 1993, doi: 10.1109/78.258082.

[13] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, “Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition,” in *Conference Record of the Asilomar Conference on Signals, Systems & Computers*, 1993, vol. 1, pp. 40–44, doi: 10.1109/acssc.1993.342465.

[14] S. S. Chen, D. L. Donoho, and M. A. Saunders, “Atomic decomposition by basis pursuit,” *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, Jan. 1998, doi: 10.1137/S1064827596304010.

[15] B. Efron *et al.*, “Least angle regression,” *Ann. Stat.*, vol. 32, no. 2, pp. 407–499, Apr. 2004, doi: 10.1214/009053604000000067.

[16] C. F. Caiafa and A. Cichocki, “Computing Sparse Representations of Multidimensional Signals Using Kronecker Bases,” *Neural Comput.*, vol. 25, no. Ci, pp. 1–35, Jan. 2012, doi: 10.1162/NECO_a_00385.

[17] A. Elrewainy and S. S. Sherif, “Kronecker least angle regression for unsupervised unmixing of hyperspectral imaging data,” *Signal, Image Video Process.*, vol. 14, no. 2, pp. 359–367, Mar. 2020, doi: 10.1007/s11760-019-01562-w.

[18] I. Wickramasingha, M. Sobhy, A. Elrewainy, and S. S. Sherif, “Tensor least angle regression for sparse representations of multidimensional signals,” *Neural Comput.*, vol. 32, no. 9, pp. 1697–1732, Sep. 2020, doi: 10.1162/neco_a_01304.

[19] D. L. Donoho, “Compressed sensing,” *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006, doi: 10.1109/TIT.2006.871582.

[20] E. J. Candès, “Compressive sampling,” in *International Congress of Mathematicians, ICM 2006*, 2006, vol. 3, pp. 1433–1452, doi: 10.4171/022-3/69.

[21] M. F. Duarte and R. G. Baraniuk, “Kronecker compressive sensing,” *IEEE Trans. Image Process.*, vol. 21, no. 2, pp. 494–504, Feb. 2012, doi: 10.1109/TIP.2011.2165289.

[22] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. 1999.

[23] E. van den Berg and M. P. Friedlander, “Probing the Pareto Frontier for Basis Pursuit Solutions,” *SIAM J. Sci. Comput.*, vol. 31, no. 2, pp. 890–912, Jan. 2009, doi: 10.1137/080714488.

[24] M. S. Asif and J. Romberg, “Sparse Recovery of Streaming Signals Using L1-Homotopy,”

IEEE Trans. Signal Process., vol. 62, no. 16, pp. 4209–4223, Aug. 2014, doi: 10.1109/TSP.2014.2328981.

[25] R. Tibshirani, “Regression Selection and Shrinkage via the Lasso,” *Journal of the Royal Statistical Society B*, vol. 58, no. 1. WileyRoyal Statistical Society, pp. 267–288, 1996, doi: 10.2307/2346178.

[26] N. R. Draper and H. Smith, “Ridge Regression,” vol. 87, no. 10, Wiley, 1998, pp. 387–400.

[27] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 67, no. 2, pp. 301–320, Nov. 2005, doi: 10.1111/j.1467-9868.2005.00503.x.

[28] H. Zou and T. Hastie, “Regression Shrinkage and Selection via the Elastic Net, with Applications to Microarrays,” *J. R. Stat. Soc. Ser. B*, vol. 67, no. 1, pp. 301–320, 2003, [Online]. Available: http://webdocs.cs.ualberta.ca/~mahdavif/ReadingGroup/Papers/10.1.1.9.6188.pdf.

[29] M. Elad and M. Aharon, “Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries,” *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006, doi: 10.1109/TIP.2006.881969.

[30] F. Roemer, G. Del Galdo, and M. Haardt, “Tensor-based algorithms for learning multidimensional separable dictionaries,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, May 2014, pp. 3963–3967, doi: 10.1109/ICASSP.2014.6854345.

[31] K. Engan, S. O. Aase, and J. H. Husøy, “Multi-frame compression: Theory and design,” *Signal Processing*, vol. 80, no. 10, pp. 2121–2140, Oct. 2000, doi: 10.1016/S0165-1684(00)00072-4.

[32] M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation,” *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, 2006, doi: 10.1109/TSP.2006.881199.

[33] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng, “Shift-invariant sparse coding for audio classification,” in *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence, UAI 2007*, Jun. 2007, pp. 149–158, Accessed: Apr. 28, 2020. [Online]. Available: http://arxiv.org/abs/1206.5241.

[34] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, “Self-taught learning: Transfer learning from unlabeled data,” in *ACM International Conference Proceeding Series*, 2007, vol. 227, pp. 759–766, doi: 10.1145/1273496.1273592.

[35] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, “Robust face recognition via sparse representation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, 2009, doi: 10.1109/TPAMI.2008.79.

[36] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, “Discriminative learned dictionaries for local image analysis,” 2008, doi: 10.1109/CVPR.2008.4587652.

[37] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, “Supervised dictionary learning,” in *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, 2009, pp. 1033–1040.

[38] J. Mairal, F. Bach, and J. Ponce, “Task-driven dictionary learning,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, pp. 791–804, Apr. 2012, doi: 10.1109/TPAMI.2011.156.

[39] X. Sun, N. M. Nasrabadi, and T. D. Tran, “Task-driven dictionary learning for hyperspectral image classification with structured sparsity constraints,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 8, pp. 4457–4471, 2015, doi: 10.1109/TGRS.2015.2399978.

[40] H. Hu, B. Wohlberg, and R. Chartrand, “Task-driven dictionary learning for inpainting,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, May 2014, no. 2, pp. 3543–3547, doi: 10.1109/ICASSP.2014.6854260.

[41] Y. Zhang, X. Mou, G. Wang, and H. Yu, “Tensor-Based Dictionary Learning for Spectral CT Reconstruction,” *IEEE Trans. Med. Imaging*, vol. 36, no. 1, pp. 142–154, 2017, doi: 10.1109/TMI.2016.2600249.

[42] M. S. Asif, “Dynamic Compressive Sensing : Sparse Recovery Algorithms for Streaming Signals and Video,” no. August, 2013, [Online]. Available: http://users.ece.gatech.edu/~sasif/Research/asif-dissertation-2013.pdf.

[43] S. Hawe, M. Seibert, and M. Kleinsteuber, “Separable dictionary learning,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013, pp. 438–445, doi: 10.1109/CVPR.2013.63.

[44] D. Kressner and C. Tobler, “htucker – A Matlab toolbox for tensors in hierarchical,” no. 4, pp. 1–28, 2013.

[45] T. G. Kolda, “Multilinear operators for higher-order decompositions.,” Albuquerque, NM, and Livermore, CA, Apr. 2006. doi: 10.2172/923081.

[46] H. W. Sorenson, “Least-squares estimation: from Gauss to Kalman: The Gaussian concept of estimation by least squares, originally stimulated by astronomical studies, has provided the basis for a number of estimation theories and techniques during the ensuing 170 years—

prob," *IEEE Spectr.*, vol. 7, no. 7, pp. 63–68, Jul. 1970, doi: 10.1109/MSPEC.1970.5213471.

[47] J. Taylor, "The geometry of least squares in the 21st century," *Bernoulli*, vol. 19, no. 4, pp. 1449–1464, 2013, doi: 10.3150/12-BEJSP15.

[48] P. Deuflhard and A. Hohmann, *Numerical analysis in modern scientific computing: An introduction*, vol. 47, no. 8–9. Springer, 2004.

[49] J. Hadamard, "Sur les problems aux derivees patielles et leur signification physique," *Princet. Uni. Bull.*, vol. 13, pp. 49–52, 1902.

[50] J. L. Fernández-Martínez, Z. Fernández-Muñiz, J. L. G. Pallero, and L. M. Pedruelo-González, "From Bayes to Tarantola: New insights to understand uncertainty in inverse problems," *J. Appl. Geophys.*, vol. 98, pp. 62–72, 2013, doi: 10.1016/j.jappgeo.2013.07.005.

[51] P. C. Hansen, *Rank-Deficient and Discrete Ill-Posed Problem (Numerical Aspects of Linear Inversion).pdf*. SIAM, 1987.

[52] Å. Björk, *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[53] M. Elad, *Sparse and Redunant Representations*, vol. 53, no. 9. New York, NY: Springer New York, 2013.

[54] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*, vol. 120. Society for Industrial and Applied Mathematics, 2005.

[55] A. Lozano, G. Swirszcz, and N. Abe, "Group orthogonal matching pursuit for logistic regression," *J. Mach. Learn. Res.*, vol. 15, pp. 452–460, 2011, Accessed: May 22, 2018. [Online]. Available: http://proceedings.mlr.press/v15/lozano11a/lozano11a.pdf.

[56] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Nov. 2004, doi: 10.1002/cpa.20042.

[57] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 68, no. 1, pp. 49–67, 2006, doi: 10.1111/j.1467-9868.2005.00532.x.

[58] D. L. Donoho and J. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994, doi: 10.1093/biomet/81.3.425.

[59] R. R. Coifman, "Wavelet Analysis and Signal Processing," in *IN WAVELETS AND THEIR*

APPLICATIONS, 1990, pp. 59–68.

[60] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, “Shiftable Multiscale Transforms,” *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 587–607, 1992, doi: 10.1109/18.119725.

[61] J. L. Starck, E. J. Candès, and D. L. Donoho, “The curvelet transform for image denoising,” *IEEE Trans. Image Process.*, vol. 11, no. 6, pp. 670–684, Jun. 2002, doi: 10.1109/TIP.2002.1014998.

[62] M. N. Do and M. Vetterli, “The contourlet transform: An efficient directional multiresolution image representation,” *IEEE Trans. Image Process.*, vol. 14, no. 12, pp. 2091–2106, Dec. 2005, doi: 10.1109/TIP.2005.859376.

[63] E. Le Pennec and S. Mallat, “Bandelet image approximation and compression,” *Multiscale Model. Simul.*, vol. 4, no. 3, pp. 992–1039, Jul. 2005, doi: 10.1137/040619454.

[64] D. M. Malioutov, M. Çetin, and A. S. Willsky, “Homotopy continuation for sparse signal representation,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2005, vol. V, pp. 733–736, doi: 10.1109/ICASSP.2005.1416408.

[65] I. Tosic and P. Frossard, “Dictionary Learning,” *IEEE Signal Process. Mag.*, vol. 28, no. 2, pp. 27–38, Mar. 2011, doi: 10.1109/MSP.2010.939537.

[66] K. Kreutz-Delgado, J. F. Murray, B. D. Rao, K. Engan, T.-W. Lee, and T. J. Sejnowski, “Dictionary learning algorithms for sparse representation.,” *Neural Comput.*, vol. 15, no. 2, pp. 349–396, Feb. 2003, doi: 10.1162/089976603762552951.

[67] I. Daubechies, “The Wavelet Transform, Time-Frequency Localization and Signal Analysis,” *IEEE Trans. Inf. Theory*, vol. 36, no. 5, pp. 961–1005, 1990, doi: 10.1109/18.57199.

[68] R. R. Coifman and M. V. Wickerhauser, “Entropy-based algorithms for best basis selection,” *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 713–718, Mar. 1992, doi: 10.1109/18.119732.

[69] J. A. Tropp, “Greed is good: Algorithmic results for sparse approximation,” *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2231–2242, Oct. 2004, doi: 10.1109/TIT.2004.834793.

[70] J. Yang, Y. Peng, W. Xu, and Q. Dai, “Ways to sparse representation: An overview,” *Sci. China, Ser. F Inf. Sci.*, vol. 52, no. 4, pp. 695–703, 2009, doi: 10.1007/s11432-009-0045-5.

[71] E. Acar, D. M. Dunlavy, and T. G. Kolda, “A scalable optimization approach for fitting canonical tensor decompositions,” *J. Chemom.*, vol. 25, no. 2, pp. 67–86, 2011, doi: 10.1002/cem.1335.

[72] J. Sulam, B. Ophir, M. Zibulevsky, and M. Elad, “Trainlets: Dictionary Learning in High Dimensions,” *IEEE Trans. Signal Process.*, vol. 64, no. 12, pp. 3180–3193, 2016, doi: 10.1109/TSP.2016.2540599.

[73] D. L. Donoho and Y. Tsaig, “Fast solution of ℓ1-Norm minimization problems when the solution may be sparse,” *IEEE Trans. Inf. Theory*, vol. 54, no. 11, pp. 4789–4812, Nov. 2008, doi: 10.1109/TIT.2008.929958.

[74] Y. Rivenson and A. Stern, “Compressed imaging with a separable sensing operator,” *IEEE Signal Process. Lett.*, vol. 16, no. 6, pp. 449–452, Jun. 2009, doi: 10.1109/LSP.2009.2017817.

[75] F. Rosário, F. A. Monteiro, and A. Rodrigues, “Fast matrix inversion updates for massive MIMO detection and precoding,” *IEEE Signal Process. Lett.*, vol. 23, no. 1, pp. 75–79, Jan. 2016, doi: 10.1109/LSP.2015.2500682.

[76] D. Goldfarb, “Modification Methods for Inverting Matrices and Solving Systems of Linear Algebraic Equations,” *Math. Comput.*, vol. 26, no. 120, pp. 829–829, 1972, doi: 10.1090/S0025-5718-1972-0317527-4.

[77] P. M. Pardalos, *Convex optimization theory*, vol. 25, no. 3. Cambridge University Press, 2010.

[78] W. W. Hager, “Updating the Inverse of a Matrix,” *SIAM Rev.*, vol. 31, no. 2, pp. 221–239, Jun. 1989, doi: 10.1137/1031049.

[79] Å. Björck, *Numerical Methods in Matrix Computations*, vol. 59. Cham: Springer International Publishing, 2015.

[80] P. J. LaMontagne *et al.*, “OASIS-3: LONGITUDINAL NEUROIMAGING, CLINICAL, AND COGNITIVE DATASET FOR NORMAL AGING AND ALZHEIMER’S DISEASE,” *Alzheimer’s Dement.*, vol. 14, no. 7S_Part_2, pp. P138–P138, Jul. 2018, doi: 10.1016/j.jalz.2018.06.2231.

[81] K. Clark *et al.*, “The cancer imaging archive (TCIA): Maintaining and operating a public information repository,” *J. Digit. Imaging*, vol. 26, no. 6, pp. 1045–1057, Dec. 2013, doi: 10.1007/s10278-013-9622-7.

[82] L. Zhai, Y. Zhang, H. Lv, S. Fu, and H. Yu, “Multiscale Tensor Dictionary Learning

Approach for Multispectral Image Denoising," *IEEE Access*, vol. 6, pp. 51898–51910, 2018, doi: 10.1109/ACCESS.2018.2868765.

[83] A. Repetti, E. Chouzenoux, and J. C. Pesquet, "A penalized weighted least squares approach for restoring data corrupted with signal-dependent noise," *Eur. Signal Process. Conf.*, no. Eusipco 2012, pp. 1553–1557, 2012.

[84] J. Wang, H. Lu, J. Wen, and Z. Liang, "Multiscale penalized weighted least-squares sinogram restoration for low-dose X-ray computed tomography," *IEEE Trans. Biomed. Eng.*, vol. 55, no. 3, pp. 1022–1031, 2008, doi: 10.1109/TBME.2007.909531.

[85] K. W. Cheung, H. C. So, W. K. Ma, and Y. T. Chan, "Least Squares Algorithms for Time-of-Arrival-Based Mobile Location," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1121–1128, 2004, doi: 10.1109/TSP.2004.823465.

[86] Y. Zou, H. Liu, and Q. Wan, "An Iterative Method for Moving Target Localization Using TDOA and FDOA Measurements," *IEEE Access*, vol. 6, no. 15, pp. 2746–2754, 2017, doi: 10.1109/ACCESS.2017.2785182.

[87] P. Tarrío, A. M. Bernardos, J. A. Besada, and J. R. Casar, "A new positioning technique for RSS-based localization based on a weighted least squares estimator," *ISWCS'08 - Proc. 2008 IEEE Int. Symp. Wirel. Commun. Syst.*, pp. 633–637, 2008, doi: 10.1109/ISWCS.2008.4726133.

[88] S. C. K. Chan, "Adaptive weighted least squares algorithm for Volterra signal modeling," *IEEE Trans. Circuits Syst. I Fundam. Theory Appl.*, vol. 47, no. 4, pp. 545–554, 2000, doi: 10.1109/81.841856.

[89] M. De Courville and P. Duhamel, "Adaptive filtering in subbands using a weighted criterion," *IEEE Trans. Signal Process.*, vol. 45, no. 6, p. 1675, 1997, doi: 10.1109/icassp.1995.480341.

[90] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. N. Do, "Fast global image smoothing based on weighted least squares," *IEEE Trans. Image Process.*, vol. 23, no. 12, pp. 5638–5653, Dec. 2014, doi: 10.1109/TIP.2014.2366600.

[91] D. Ruppert and M. P. Wand, "Multivariate Locally Weighted Least Squares Regression," *Ann. Stat.*, vol. 22, no. 3, pp. 1346–1370, 2007, doi: 10.1214/aos/1176325632.

[92] L. Magee, "Improving survey-weighted least squares regression," *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 60, no. 1, pp. 115–126, Jan. 1998, doi: 10.1111/1467-9868.00112.

[93] J. P. Romano and M. Wolf, "Resurrecting weighted least squares," *J. Econom.*, vol. 197,

no. 1, pp. 1–19, 2017, doi: 10.1016/j.jeconom.2016.10.003.

[94] M. P. Friedlander, H. Mansour, R. Saab, and Ö. Yilmaz, "Recovering compressively sampled signals using partial support information," *IEEE Trans. Inf. Theory*, vol. 58, no. 2, pp. 1122–1134, 2012, doi: 10.1109/TIT.2011.2167214.

[95] M. A. Khajehnejad, W. Xu, A. S. Avestimehr, and B. Hassibi, "Weighted l1 minimization for sparse recovery with prior information," *IEEE Int. Symp. Inf. Theory - Proc.*, pp. 483–487, 2009, doi: 10.1109/ISIT.2009.5205716.

[96] E. J. Candès, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted$\ell 1$ minimization," *J. Fourier Anal. Appl.*, vol. 14, no. 5–6, pp. 877–905, Dec. 2008, doi: 10.1007/s00041-008-9045-x.

[97] L. C. Bergersen, I. K. Glad, and H. Lyng, "Weighted lasso with data integration," *Stat. Appl. Genet. Mol. Biol.*, vol. 10, no. 1, 2011, doi: 10.2202/1544-6115.1703.

[98] T. Shimamura, S. Imoto, R. Yamaguchi, and S. Miyano, "Weighted lasso in graphical Gaussian modeling for large gene network estimation based on microarray data.," *Genome Inform.*, vol. 19, pp. 142–153, Nov. 2007, doi: 10.1142/9781860949852_0013.

[99] D. L. Donoho and X. Huo, "Uncertainty principles and ideal atomic decomposition," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2845–2862, 2001, doi: 10.1109/18.959265.

[100] C. Bocci, E. Carlini, and J. Kileel, "Hadamard products of linear spaces," *J. Algebr.*, vol. 448, pp. 595–617, 2016, doi: 10.1016/j.jalgebra.2015.10.008.

[101] Q. Zhao *et al.*, "Higher order partial least squares (HOPLS): A generalized multilinear regression method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 7, pp. 1660–1673, 2013, doi: 10.1109/TPAMI.2012.254.

[102] M. Park, K. Brocklehurst, R. T. Collins, and Y. Liu, "Deformed lattice detection in real-world images using mean-shift belief propagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 10, pp. 1804–1816, 2009, doi: 10.1109/TPAMI.2009.73.

[103] E. Agustsson and R. Timofte, "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017, vol. 2017-July, pp. 1122–1131, doi: 10.1109/CVPRW.2017.150.

[104] A. Lew and H. Mauch, "Introduction to dynamic programming," *Studies in Computational Intelligence*, vol. 38. Springer, Berlin, Heidelberg, pp. 3–43, 2007, doi: 10.1007/978-3-540-37014-7_1.

[105] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards*. Boston, MA: Springer US, 1995.

[106] I. Bocharova and I. Bocharova, *Video-coding standards*. 2012.

[107] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.

[108] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, “A survey of transfer learning,” *J. Big Data*, vol. 3, no. 1, p. 9, Dec. 2016, doi: 10.1186/s40537-016-0043-6.

[109] W. Kumagai and T. Kanamori, “Risk bound of transfer learning using parametric feature mapping and its application to sparse coding,” *Mach. Learn.*, vol. 108, no. 11, pp. 1975–2008, Nov. 2019, doi: 10.1007/s10994-019-05805-2.

[110] A. Maurer, M. Pontil, and B. Romera-Paredes, “Sparse coding for multitask and transfer learning,” in *30th International Conference on Machine Learning, ICML 2013*, 2013, no. PART 2, pp. 1002–1010.

[111] P. Fieguth, *Statistical Image Processing and Multidimensional Modeling*, vol. 58, no. 12. New York, NY: Springer New York, 2011.

[112] Z. Ge *et al.*, “Exploiting Temporal Information for DCNN-Based Fine-Grained Object Classification,” *2016 Int. Conf. Digit. Image Comput. Tech. Appl. DICTA 2016*, 2016, doi: 10.1109/DICTA.2016.7797039.

[113] A. Tovaglieri, “Research Collection,” *Brisk Bin. Robust Invariant Scalable Keypoints*, vol. 15, no. 3, pp. 12–19, 2011, doi: 10.3929/ethz-a-010782581.

[114] P. C. Martin von Siebenthal^1, “Respiratory Organ Motion from 4DMRI.” www.vision.ethz.ch/4dmri (accessed Jun. 26, 2018).

[115] H. Zanddizari, S. Rajan, and H. Zarrabi, “Increasing the quality of reconstructed signal in compressive sensing utilizing Kronecker technique,” *Biomed. Eng. Lett.*, vol. 8, no. 2, pp. 239–247, 2018, doi: 10.1007/s13534-018-0057-4.

[116] B. Mailhé, S. Lesage, R. Gribonval, F. Bimbot, and P. Vandergheynst, “Shift-invariant dictionary learning for sparse representations: extending K-SVD,” 2008. Accessed: Aug. 29, 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00350165.

[117] J. Lefebvre, A. Castonguay, and F. Lesage, “OCT Mouse Brain Templates,” vol. 1. Mendeley Data, 2017, doi: 10.17632/33wfgxpmp8.1.

[118] J. Caballero *et al.*, “Real-time video super-resolution with spatio-temporal networks and

motion compensation," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-Janua, pp. 2848–2857, doi: 10.1109/CVPR.2017.304.

[119] S. Christen, C. Studer, and G. Pope, "Dictionary Learning for Super-Resolution," 2010.

[120] S. Bahrampour, N. M. Nasrabadi, A. Ray, and W. K. Jenkins, "Multimodal Task-Driven Dictionary Learning for Image Classification," *IEEE Trans. Image Process.*, vol. 25, no. 1, pp. 24–38, Jan. 2016, doi: 10.1109/TIP.2015.2496275.

[121] A. Koppel, G. Warned, and E. Stump, "Task-driven dictionary learning in distributed online settings," *Conf. Rec. - Asilomar Conf. Signals, Syst. Comput.*, vol. 2016-Febru, no. 2, pp. 1114–1118, 2016, doi: 10.1109/ACSSC.2015.7421313.

[122] R. Zhao and Q. Wang, "Learning Separable Dictionaries for Sparse Tensor Representation: An Online Approach," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 66, no. 3, pp. 502–506, Mar. 2019, doi: 10.1109/TCSII.2018.2862900.

[123] Y. Su, X. Gao, X. Li, and D. Tao, "Multivariate multilinear regression," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 42, no. 6, pp. 1560–1573, 2012, doi: 10.1109/TSMCB.2012.2195171.

[124] E. F. Lock, "Tensor-on-Tensor Regression," *J. Comput. Graph. Stat.*, vol. 27, no. 3, pp. 638–647, 2018, doi: 10.1080/10618600.2017.1401544.

[125] I. Rish and G. Grabarnik, *Sparse Modeling: Theory, Algorithms, and Applications*. 2015.

[126] L. Bottou, "Stochastic Gradient Descent Tricks," vol. 1, no. 1, Springer, Berlin, Heidelberg, 2012, pp. 421–436.

[127] B. Rozovskii and M. Yor, *Stochastic Approximation and Recursive Algorithms and Applications*, vol. 35. New York: Springer-Verlag, 2003.

[128] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.

[129] S. P. Morozov *et al.*, "MosMedData: Chest CT scans with COVID-19 related findings dataset," *medRxiv*. 2020, doi: 10.1101/2020.05.20.20100362.

[130] C. Bao, X. Liu, H. Zhang, Y. Li, and J. Liu, "Coronavirus Disease 2019 (COVID-19) CT Findings: A Systematic Review and Meta-analysis," *J. Am. Coll. Radiol.*, vol. 17, no. 6, pp. 701–709, Jun. 2020, doi: 10.1016/j.jacr.2020.03.006.

[131] H. Zunair, A. Rahman, N. Mohammed, and J. P. Cohen, “Uniformizing Techniques to Process CT Scans with 3D CNNs for Tuberculosis Prediction,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Oct. 2020, vol. 12329 LNCS, pp. 156–168, doi: 10.1007/978-3-030-59354-4_15.

[132] Y. Sasaki, “The truth of the F-measure,” 2015. Accessed: Feb. 23, 2021. [Online]. Available: https://www.cs.odu.edu/~mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07.pdf.

[133] Z. Wu *et al.*, “3D ShapeNets: A deep representation for volumetric shapes,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June, pp. 1912–1920, doi: 10.1109/CVPR.2015.7298801.

[134] L. Meier, S. Van De Geer, and P. Bühlmann, “The group lasso for logistic regression,” *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 70, no. 1, pp. 53–71, 2008, doi: 10.1111/j.1467-9868.2007.00627.x.

[135] J. Hu, J. Huang, and F. Qiu, “A group adaptive elastic-net approach for variable selection in high-dimensional linear regression,” *Sci. China Math.*, vol. 61, no. 1, pp. 173–188, Jan. 2018, doi: 10.1007/s11425-016-0071-x.

[136] W. Xiao, Y. Wu, and H. Zhou, “ConvexLAR: An Extension of Least Angle Regression,” *J. Comput. Graph. Stat.*, vol. 24, no. 3, pp. 603–626, Jul. 2015, doi: 10.1080/10618600.2014.962700.

[137] P. Lancaster and H. K. Farahat, “Norms on Direct Sums and Tensor Products,” vol. 26, no. 118, pp. 401–414, 1972.

[138] K. Brandt Petersen Michael Syskind Pedersen *et al.*, “The Matrix Cookbook.” Accessed: Nov. 15, 2018. [Online]. Available: http://www.math.uwaterloo.ca/~hwolkowi//matrixcookbook.pdf.

[139] J. R. Magnus, “On the concept of matrix derivative,” *J. Multivar. Anal.*, vol. 101, no. 9, pp. 2200–2206, 2010, doi: 10.1016/j.jmva.2010.05.005.

[140] J. R. Schott and D. A. Harville, “Matrix Algebra from a Statistician’s Perspective,” *J. Am. Stat. Assoc.*, vol. 93, no. 443, p. 1236, 1998, doi: 10.2307/2669871.

Appendix A

A.1 Mapping of Tensor Indices to Corresponding Vector Indices of $\text{vec}(\mathcal{X}) \in \mathbb{R}^{I_N I_{N-1} \dots I_1}$

Proposition 2.1: *Let l be the vector index of an element v_l in $vec(\mathcal{X}) \in \mathbb{R}^{I_N I_{N-1} \dots I_1}$ and I_n be the dimension of the mode-n of the tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_n \times \dots \times I_N}$ where, $n \in \{1,2,\dots,N\}$. The tensor indices $i_n; n \in \{1,2,\dots,N\}$, corresponding to the vector element v_l, can be obtained by,*

$$i_n = \left\lceil \frac{l}{I_1 \times \dots \times I_{n-1}} - \sum_{p=n+1; p \le N}^{N} \left(i_p - 1\right) \prod_{q=n; q>0}^{p-1} I_q \right\rceil \tag{A.1}$$

where $\lceil * \rceil$ indicate the ceiling function. For example,

$$i_1 = \lceil l - (i_N - 1) I_{N-1} \times \dots . \times I_1 - \dots - (i_2 - 1) I_1 \rceil$$

$$\vdots$$

$$i_{N-1} = \left\lceil \frac{l}{I_1 \times \dots \times I_{N-2}} - (i_N - 1) I_{N-1} \right\rceil$$

$$i_N = \left\lceil \frac{l}{I_1 \times \dots \times I_{N-1}} \right\rceil$$

Proof: Note that i_n for all $n \in \{1,2,\dots,N\}$ are integers and $1 \le i_n \le I_\text{n}$

From (2.1),

$$l = i_1 + (i_2 - 1) I_1 + \dots + (i_N - 1) I_1 \times \dots \times I_{N-1} \tag{A.2}$$

$$\Rightarrow i_N - 1 = \underbrace{\frac{l}{I_1 \times \dots \times I_{N-1}}}_{S_N} - \underbrace{\frac{i_1 + (i_2 - 1) I_1 + \dots + (i_{N-1} - 1) I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}}}_{f_N}$$

$$i_N = S_N + (1 - f_N) \tag{A.3}$$

Since $i_n \le I_\text{n} \ \forall \, n \in \{1,2,\dots,N\}$,

$$f_N \le \frac{I_1 + (I_2 - 1) I_1 + \dots + (I_{N-1} - 1) I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}} = 1 \tag{A.4}$$

Also $1 \leq i_n \;\; \forall\, n \in \{1,2,\dots,N\}$ and,

$$f_N = \frac{1 + (1-1)I_1 + \cdots + (1-1)I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}} > 0 \qquad (\mathrm{A.5})$$

Therefore, from (A.4) and (A.5),

$$0 < f_N \leq 1 \qquad (\mathrm{A.6})$$

$$\Longrightarrow 0 \leq 1 - f_N < 1 \qquad (\mathrm{A.7})$$

Since $i_N \in \mathbb{Z}$, from (A.3) and (A.7),

$$\Rightarrow i_N = \lceil i_N - (1 - f_N) \rceil = \lceil S_N \rceil$$

$$i_N = \left\lceil \frac{l}{I_1 \times \dots \times I_{N-1}} \right\rceil \qquad (\mathrm{A.8})$$

$\lceil * \rceil$ indicate the ceiling of the value.

Similarly,

$$i_{N-1} - 1 = \underbrace{\frac{l}{I_1 \times \dots \times I_{N-2}} - (i_N - 1)I_{N-1}}_{S_{N-1}} - \underbrace{\frac{i_1 + (i_2 - 1)I_1 + \cdots + (i_{N-1} - 1)I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-2}}}_{f_{N-1}}$$

$$i_{N-1} = S_{N-1} + (1 - f_{N-1})$$

$$0 \leq (1 - f_{N-1}) < 1$$

Since $i_{N-1} \in \mathbb{Z}$ and $0 \leq (1 - f_{N-1}) < 1$,

$$i_{N-1} = \lceil i_{N-1} - (1 - f_{N-1}) \rceil = \lceil S_{N-1} \rceil$$

$$i_{N-1} = \left\lceil \frac{l}{I_1 \times \dots \times I_{N-2}} - (i_N - 1)I_{N-1} \right\rceil$$

Similarly, for $\forall\, n \in \{1,2,\dots,N\}$, $0 \leq (1 - f_n) < 1$

$$i_n = \lceil i_n - (1 - f_n) \rceil = \lceil S_n \rceil$$

$$i_n = \left\lceil \frac{l}{I_1 \times \dots \times I_{n-1}} - \sum_{p=n+1;\, p \leq N}^{N} \left(i_p - 1\right) \prod_{q=n;\, q>0}^{p-1} I_q \right\rceil$$

Appendix B

B.1 Mapping of column indices of dictionary $\boldsymbol{\Phi}$ to column indices of *mode-n* dictionaries

T-LARS avoids the construction of large matrices such as the separable dictionary $\boldsymbol{\Phi}$ in (3.4). Instead, T-LARS uses *mode-n* dictionaries for calculations. The following mapping between column indices of dictionary $\boldsymbol{\Phi}$ and column indices of *mode-n* dictionaries $\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$ is essential in T-LARS calculations.

The arbitrary column $\boldsymbol{\Phi}_k$ is the k^{th} column of the separable dictionary $\boldsymbol{\Phi}$ in (3.4), which is given by the Kronecker product of the columns of the dictionary matrices $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \dots, \boldsymbol{\Phi}^{(N)}$, $n \in \{1, \cdots, N\}$.

$$\boldsymbol{\Phi}_k \ = \ \phi_{i_N}^{(N)} \otimes \phi_{i_{N-1}}^{(N-1)} \otimes \dots \ \otimes \phi_{i_1}^{(1)} \qquad (\mathrm{B.1})$$

The column indices $(i_N, i_{N-1}, \dots, i_1)$ are the indices of the columns of the dictionary matrices $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \dots, \boldsymbol{\Phi}^{(N)}$. The column index k of the separable dictionary $\boldsymbol{\Phi}$ is given by [45],

$$k \ = i_1 + \sum_{n=2}^{N} (i_n - 1) I_1 I_2 \dots . I_{n-1} \qquad (\mathrm{B.2})$$

where, $I_1, I_2, \dots, I_N$ are the dimensions of the columns of the dictionary matrices $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \dots, \boldsymbol{\Phi}^{(N)}$, respectively. The following proposition shows how to obtain the column indices of the dictionary matrices $(i_N, i_{N-1}, \dots, i_1)$ corresponds to the column index k of the separable dictionary $\boldsymbol{\Phi}$.

Proposition B.1: Let k be the column index of the separable dictionary column vector $\boldsymbol{\Phi}_k$ and I_n be the dimension of the columns of each dictionary matrix $\boldsymbol{\Phi}^{(n)}$; $n \in \{1, \cdots, N\}$. In $(\mathrm{B.1})$, the corresponding column indices i_n; $n \in \{1, \cdots, N\}$ of each dictionary matrix $\phi_{i_n}^{(n)}$ is given by,

$$i_n = \left\lceil \frac{k}{I_1 \times \dots \times I_{n-1}} - \sum_{p=n+1; p \leq N}^{N} \left(i_p - 1\right) \prod_{q=n; q>0}^{p-1} I_q \right\rceil \qquad (\mathrm{B.3})$$

where $\lceil * \rceil$ indicate the ceiling function. For example,

$$i_1 = \lceil k - (i_N - 1)I_{N-1} \times \dots . \times I_1 - \dots - (i_2 - 1)I_1 \rceil$$

$$\vdots$$

$$i_{N-1} = \left\lceil \frac{k}{I_1 \times \dots \times I_{N-2}} - (i_N - 1)I_{N-1} \right\rceil$$

$$i_N = \left\lceil \frac{k}{I_1 \times \dots \times I_{N-1}} \right\rceil$$

Proof: We note that i_n ; $\forall\, n \in \{1,2,\dots,N\}$ are integers and $1 \leq i_n \leq I_\mathrm{n}$

From $(\mathrm{B}.2)$,

$$k = i_1 + (i_2 - 1)I_1 + \dots + (i_N - 1)I_1 \times \dots \times I_{N-1} \qquad (\mathrm{B}.4)$$

Therefore,

$$i_N - 1 = \underbrace{\frac{k}{I_1 \times \dots \times I_{N-1}}}_{S_N} - \underbrace{\frac{i_1 + (i_2 - 1)I_1 + \dots + (i_{N-1} - 1)I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}}}_{f_N}$$

$$i_N = S_N + (1 - f_N) \qquad (\mathrm{B}.5)$$

Since $i_n \leq I_\mathrm{n}$; $\forall\, n \in \{1,2,\dots,N\}$,

$$f_N \leq \frac{I_1 + (I_2 - 1)I_1 + \dots + (I_{N-1} - 1)I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}} = 1 \qquad (\mathrm{B}.6)$$

Also since $1 \leq i_n$; $\forall\, n \in \{1,2,\dots,N\}$, we have

$$f_N = \frac{1 + (1 - 1)I_1 + \dots + (1 - 1)I_1 \times \dots \times I_{N-2}}{I_1 \times \dots \times I_{N-1}} > 0 \qquad (\mathrm{B}.7)$$

Therefore, from $(\mathrm{B}.6)$ and $(\mathrm{B}.7)$,

$$0 < f_N \leq 1 \qquad (\mathrm{B}.8)$$

$$0 \leq 1 - f_N < 1 \qquad (\mathrm{B}.9)$$

Since $i_N \in \mathbb{Z}$, then from $(\mathrm{B}.5)$ and $(\mathrm{B}.9)$ we have

$$i_N = \lceil i_N - (1 - f_N) \rceil = \lceil S_N \rceil$$

$$i_N = \left\lceil \frac{k}{I_1 \times \ldots \times I_{N-1}} \right\rceil \tag{B.10}$$

where $\lceil * \rceil$ indicate the ceiling function.

Similarly,

$$i_{N-1} - 1 = \underbrace{\frac{k}{I_1 \times \ldots \times I_{N-2}} - (i_N - 1) I_{N-1}}_{s_{N-1}} - \underbrace{\frac{i_1 + (i_2 - 1) I_1 + \cdots + (i_{N-1} - 1) I_1 \times \ldots \times I_{N-2}}{I_1 \times \ldots \times I_{N-2}}}_{f_{N-1}}$$

$$i_{N-1} = s_{N-1} + (1 - f_{N-1})$$

$$0 \leq (1 - f_{N-1}) < \ 1$$

Since $i_{N-1} \in \mathbb{Z}$ and $0 \leq (1 - f_{N-1}) < 1$,

$$i_{N-1} \ = \ \lceil i_{N-1} - (1 - f_{N-1}) \rceil = \lceil S_{N-1} \rceil$$

$$i_{N-1} = \left\lceil \frac{k}{I_1 \times \ldots \times I_{N-2}} - (i_N - 1) I_{N-1} \right\rceil$$

Similarly, for $\forall\, n \in \{1,2, \ldots, N\}$,

$$0 \leq (1 - f_n) < \ 1$$

$$i_n \ = \ \lceil i_n - (1 - f_n) \rceil = \lceil S_n \rceil$$

$$i_n = \left\lceil \frac{k}{I_1 \times \ldots \times I_{n-1}} - \sum_{p=n+1; p \leq N}^{N} \left(i_p - 1 \right) \prod_{q=n; q>0}^{p-1} I_q \right\rceil$$

B.2 Normalization of the tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \ldots \times J_n \times \ldots \times J_N}$

Compute

$$\hat{\mathcal{Y}} \ = \ \frac{\mathcal{Y}}{\|\mathcal{Y}\|_2} \tag{A.11}$$

where, $\|\mathcal{Y}\|_2 = \sqrt{\langle \mathcal{Y}, \mathcal{Y} \rangle} = \left(\sum_{j_1}^{J_1} \dots \sum_{j_N}^{J_N} y_{j_1 j_2 \dots j_N}^2\right)^{\frac{1}{2}}$

B.3 Normalization of columns of the separable dictionary $\boldsymbol{\Phi}$ to have a unit L_2 norm

The column $\boldsymbol{\Phi}_k$ in $(\mathrm{B.1})$, is the k^{th} column of the separable dictionary $\boldsymbol{\Phi}$. Normalization of each column vector $\boldsymbol{\Phi}_k$ of the separable dictionary is given by,

$$\widehat{\boldsymbol{\Phi}}_k = \frac{\boldsymbol{\Phi}_k}{\|\boldsymbol{\Phi}_k\|_2} \tag{B.12}$$

Proposition B.2: *Normalization of the column* $\boldsymbol{\Phi}_k$ *in* $(\mathrm{B.1})$ *is given by the Kronecker product of the normalization of the dictionary columns* $\boldsymbol{\phi}_{i_N}^{(N)}, \boldsymbol{\phi}_{i_{N-1}}^{(N-1)}, \dots, \boldsymbol{\phi}_{i_1}^{(1)}$

$$\widehat{\boldsymbol{\Phi}}_k = \widehat{\boldsymbol{\phi}}_{i_N}^{(N)} \otimes \widehat{\boldsymbol{\phi}}_{i_{N-1}}^{(N-1)} \otimes \dots \otimes \widehat{\boldsymbol{\phi}}_{i_1}^{(1)} \tag{B.13}$$

Proof: The *L2* norm of the Kronecker product of vectors is the product of *L2* norms of these vectors [137], i.e.,

$$\|\boldsymbol{\Phi}_k\|_2^2 = \left\|\boldsymbol{\phi}_{i_N}^{(N)} \otimes \boldsymbol{\phi}_{i_{N-1}}^{(N-1)} \otimes \dots \otimes \boldsymbol{\phi}_{i_1}^{(1)}\right\| = \left\|\boldsymbol{\phi}_{i_N}^{(N)}\right\|_2^2 \times \left\|\boldsymbol{\phi}_{i_{N-1}}^{(N-1)}\right\|_2^2 \times \dots \times \left\|\boldsymbol{\phi}_{i_1}^{(1)}\right\|_2^2 \tag{B.14}$$

From $(\mathrm{B.12})$ and $(\mathrm{B.13})$,

$$\widehat{\boldsymbol{\Phi}}_k = \frac{\boldsymbol{\Phi}_k}{\|\boldsymbol{\Phi}_k\|_2} = \frac{\boldsymbol{\phi}_{i_N}^{(N)}}{\left\|\boldsymbol{\phi}_{i_N}^{(N)}\right\|_2^2} \otimes \dots \otimes \frac{\boldsymbol{\phi}_{i_1}^{(1)}}{\left\|\boldsymbol{\phi}_{i_1}^{(1)}\right\|_2^2} \tag{B.15}$$

Therefore,

$$\widehat{\boldsymbol{\Phi}}_k = \widehat{\boldsymbol{\phi}}_{i_N}^{(N)} \otimes \widehat{\boldsymbol{\phi}}_{i_{N-1}}^{(N-1)} \otimes \dots \otimes \widehat{\boldsymbol{\phi}}_{i_1}^{(1)}$$

B.4 Obtaining the initial correlation tensor $\mathcal{C}_1$

In T-LARS, the initial correlation vector $\boldsymbol{c}_1$ is obtained by taking the correlation between all columns of $\boldsymbol{\Phi}$ and the vectorization of the tensor $\mathcal{Y}$,

$$\boldsymbol{c}_1 = \left(\boldsymbol{\Phi}^{(N)^T} \otimes \boldsymbol{\Phi}^{(N-1)^T} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)^T}\right) \operatorname{vec}(\mathcal{Y}) \tag{B.16}$$

We could also represent $(\mathrm{B}.16)$ as a multilinear transformation of the tensor $\mathcal{Y}$ [45],

$$\mathcal{C}_1 = \mathcal{Y} \times_1 \boldsymbol{\Phi}^{(1)^T} \times_2 \ldots \times_n \boldsymbol{\Phi}^{(n)^T} \times_{n+1} \ldots \times_N \boldsymbol{\Phi}^{(N)^T} \tag{B.17}$$

The tensor $\mathcal{C}_1$ is the correlation between the tensor $\mathcal{Y}$ and the *mode-n* dictionary matrices $\boldsymbol{\Phi}^{(n)}$; $n \in \{1, \cdots, N\}$. The tensor $\mathcal{C}_1$ could be calculated efficiently as N *mode-n* products.

B.5 Creating a Gram matrix for each *mode-n* dictionary $\boldsymbol{\Phi}^{(n)}$

Gram matrices are used in several steps of T-LARS. For a large separable dictionary, $\boldsymbol{\Phi}$, its Gram matrix would be large as well. Therefore, explicitly building this Gram matrix and using it in computations could be very inefficient for large problems. Therefore, we developed T-LARS to use Gram matrices of *mode-n* dictionary matrices, $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \ldots, \boldsymbol{\Phi}^{(N)}$, defined as $\boldsymbol{G}^{(n)}$; $n \in \{1, \cdots, N\}$, instead of the Gram matrix $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$,s

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)^T}\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n)^T}\boldsymbol{\Phi}^{(n)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)^T}\boldsymbol{\Phi}^{(1)} \tag{B.18}$$

We can obtain, Gram matrix $\boldsymbol{G}^{(n)}$ for each *mode-n* dictionary $\boldsymbol{\Phi}^{(n)}$ by,

$$\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T}\boldsymbol{\Phi}^{(n)} \tag{B.19}$$

The total sizes of the Gram matrices $\boldsymbol{G}^{(n)}$; $n \in \{1, \cdots, N\}$ would be much smaller than the Gram matrix $\boldsymbol{G} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$, thereby allowing faster calculations and requiring less computer storage.

Appendix C

C.1 Obtain the inverse of the Gram matrix of the active columns of the dictionary in WT-LARS

Let the column $k_a \in I$ be the new column added to the active matrix. Given $\boldsymbol{G}_{t-1}^{-1}$, the inverse of the Gram matrix $\boldsymbol{G}_t^{-1}$ could be calculated using the *Schur complement* inversion formula for a symmetric block matrix [77]–[79],

$$\boldsymbol{G}_t^{-1} = \begin{bmatrix} \boldsymbol{F}_{11}^{-1} & \alpha\boldsymbol{b} \\ \alpha\boldsymbol{b}^{\mathrm{T}} & \alpha \end{bmatrix} \qquad (\mathrm{B.}\,20)$$

where, $\boldsymbol{F}_{11}^{-1} = \boldsymbol{G}_{t-1}^{-1} + \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$, $\boldsymbol{b} = -\boldsymbol{G}_{t-1}^{-1}\mathbf{g}_{\mathrm{a}}$ and $\alpha = 1 / \left(\mathrm{g}_{(k_a,k_a)} + \mathbf{g}_{\mathrm{a}}^T\boldsymbol{b}\right)$ and the column vector $[\mathbf{g}_{\mathrm{a}} \;\; \mathrm{g}_{(k_a,k_a)}]^T$ is given by,

$$[\mathbf{g}_{\mathrm{a}} \;\; \mathrm{g}_{(k_a,k_a)}]^T = \boldsymbol{\Phi}_{W}{}_{I_t}^{T}\boldsymbol{\phi}_{k_a}$$

Where $\boldsymbol{\phi}_{k_a}$ is the k^{th} column of $\boldsymbol{\Phi}_W$ and $\mathrm{g}_{(k_a,k_a)}$ is the last element of the vector $\boldsymbol{\Phi}_{W}{}_{I_t}^{T}\boldsymbol{\phi}_{k_a}$.

Since $\boldsymbol{\Phi}_W = \boldsymbol{S\Phi Q}$, we can easily calculate $\mathbf{g}_{\mathrm{a}}$ and $\mathrm{g}_{(k_a,k_a)}$ as, $[\mathbf{g}_{\mathrm{a}} \;\; \mathrm{g}_{(k_a,k_a)}]^T = \boldsymbol{Q}_{k_a,k_a}\boldsymbol{Q}_{I_t}^T\boldsymbol{\Phi}_{I_t}^T\boldsymbol{W}\boldsymbol{\phi}_k$, where $\boldsymbol{\phi}_k$ is the k^{th} column of $\boldsymbol{\Phi}$.

Please refer to T-LARS [18] in chapter 3 for updating the inverse of the Gram matrix $\boldsymbol{G}_{t-1}^{-1}$ to obtain $\boldsymbol{G}_t^{-1}$ after removing a column $k_a \in I$, which is identical in both WT-LARS and T-LARS.

Appendix D

D.1 Obtain the inverse of the Gram matrix of the active columns of the dictionary in TD-LARS

Let the column $k_a \in I$ be the new column added to the active matrix. Given $\boldsymbol{G}_{t-1}^{-1}$, the inverse of the Gram matrix $\boldsymbol{G}_t^{-1}$ could be calculated using the *Schur complement* inversion formula for a symmetric block matrix [77]–[79],

$$\boldsymbol{G}_t^{-1} = \begin{bmatrix} \boldsymbol{F}_{11}^{-1} & \alpha\boldsymbol{b} \\ \alpha\boldsymbol{b}^{\mathrm{T}} & \alpha \end{bmatrix} \qquad (\mathrm{B}.21)$$

where, $\boldsymbol{F}_{11}^{-1} = \boldsymbol{G}_{t-1}^{-1} + \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$, $\boldsymbol{b} = -\boldsymbol{G}_{t-1}^{-1}\mathbf{g}_{\mathrm{a}}$ and $\alpha = 1 / \left(\mathrm{g}_{(k_a,k_a)} + \mathbf{g}_{\mathrm{a}}^T\boldsymbol{b}\right)$ and the column vector $[\mathbf{g}_{\mathrm{a}} \quad \mathrm{g}_{(k_a,k_a)}]^T$ is given by,

$$[\mathbf{g}_{\mathrm{a}} \quad \mathrm{g}_{(k_a,k_a)}]^T = \boldsymbol{\Phi}_{W}{}_{I_t}^{T}\boldsymbol{\phi}_{k_a}$$

Where $\boldsymbol{\phi}_{k_a}$ is the k^{th} column of $\boldsymbol{\Phi}_W$ and $\mathrm{g}_{(k_a,k_a)}$ is the last element of the vector $\boldsymbol{\Phi}_{W}{}_{I_t}^{T}\boldsymbol{\phi}_{k_a}$.

Since $\boldsymbol{\Phi}_W = \boldsymbol{S\Phi Q}$, we can easily calculate $\mathbf{g}_{\mathrm{a}}$ and $\mathrm{g}_{(k_a,k_a)}$ as, $[\mathbf{g}_{\mathrm{a}} \quad \mathrm{g}_{(k_a,k_a)}]^T = \boldsymbol{Q}_{k_a,k_a}\boldsymbol{Q}_{I_t}^T\boldsymbol{\Phi}_{I_t}^T\boldsymbol{W}\boldsymbol{\phi}_k$, where $\boldsymbol{\phi}_k$ is the k^{th} column of $\boldsymbol{\Phi}$.

Please refer to T-LARS [18] in chapter 3 for updating the inverse of the Gram matrix $\boldsymbol{G}_{t-1}^{-1}$ to obtain $\boldsymbol{G}_t^{-1}$ after removing a column $k_a \in I$, which is identical in both TD-LARS and T-LARS.

Appendix E

E.1 Obtain The Inverse Of The Gram Matrix Of The Active Columns Of The Dictionary in T-NET

Let the column $k_a \in I$ be the new column added to the active matrix. Given $\boldsymbol{G}_{t-1}^{-1}$, the inverse of the Gram matrix $\boldsymbol{G}_t^{-1}$ could be calculated using the *Schur complement* inversion formula for a symmetric block matrix [77]–[79],

$$\boldsymbol{G}_t^{-1} = \begin{bmatrix} \boldsymbol{F}_{11}^{-1} & \alpha\boldsymbol{b} \\ \alpha\boldsymbol{b}^{\mathrm{T}} & \alpha \end{bmatrix} \tag{B.22}$$

where, $\boldsymbol{F}_{11}^{-1} = \boldsymbol{G}_{t-1}^{-1} + \alpha\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$, $\boldsymbol{b} = -\frac{1}{1+\nu_2}\boldsymbol{G}_{t-1}^{-1}\mathbf{g}_{\mathrm{a}}$ and $\alpha = (1+\nu_2) \,/\, \left(\mathrm{g}_{(k_a,k_a)} + \nu_2 + \mathbf{g}_{\mathrm{a}}^T\boldsymbol{b}\right)$ and the column vector $\mathbf{g}_{\mathrm{a}}^T$ is given by,

$$\mathbf{g}_{\mathrm{a}}^T = [\mathrm{g}_{(k_1,k_a)} \quad \cdots \; \mathrm{g}_{(k_n,k_a)} \cdots \quad \mathrm{g}_{(k_{a-1},k_a)}]_{1\times a-1}$$

The elements, $\mathrm{g}_{(k_n,k_a)}$ of $\mathbf{g}_{\mathrm{a}}^T$ are elements of the gram matrix, $\boldsymbol{G}_t$, that are obtained using *mode-n* gram matrices $\boldsymbol{G}^{(n)} = \boldsymbol{\Phi}^{(n)^T}\boldsymbol{\Phi}^{(n)}; n \in \{1, \cdots, N\}$.

$$\mathrm{g}(k_n, k_a) = \mathrm{g}^{(N)}\left(k_{n_N}, k_{a_N}\right) \otimes \ldots \otimes \mathrm{g}^{(1)}\left(k_{n_1}, k_{a_1}\right)$$

where, $k_{n_N} \cdots k_{n_1}$ are the tensor indices corresponds to the column index k_n and $k_{a_N} \cdots k_{a_1}$are the tensor indices corresponds to the column index k_a [18].

Please refer to T-LARS [18] in chapter 3 for updating the inverse of the Gram matrix $\boldsymbol{G}_{t-1}^{-1}$ to obtain $\boldsymbol{G}_t^{-1}$ after removing a column $k_a \in I$, which is identical in both T-NET and T-LARS.

Appendix F

F.1 Proof of Proposition 7.1

Proposition 7.1: *Let $f(\boldsymbol{\Phi})$ be a continuously differentiable function and $\boldsymbol{\Phi} \in \mathbb{R}^{P\times Q}$ be a Kronecker matrix, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and each mode-n matrix $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n\times J_n}; \forall\, n \in \{1, \cdots, N\}$. Therefore, the gradient $\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi})\ ; \forall n \in \{1, \cdots, N\}$ is given by,*

$$\left[\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi})\right]_{i,j} = Tr\left(\left(\nabla_{\boldsymbol{\Phi}} f(\boldsymbol{\Phi})\right)^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}^{(n)}_{i,j}}\right) \tag{B.23}$$

Proof:

$$\left[\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi})\right]_{i,j} = \frac{\partial f(\boldsymbol{\Phi})}{\partial \boldsymbol{\Phi}^{(n)}_{i,j}} \tag{B.24}$$

Let us apply the chain rule [138]–[140] to (B. 24),

$$\frac{\partial f(\boldsymbol{\Phi})}{\partial \boldsymbol{\Phi}^{(n)}_{i,j}} = \sum_{p=1}^{P}\sum_{q=1}^{Q} \frac{\partial f(\boldsymbol{\Phi})}{\partial \boldsymbol{\Phi}_{p,q}} \frac{\partial \boldsymbol{\Phi}_{p,q}}{\partial \boldsymbol{\Phi}^{(n)}_{i,j}} \tag{B.25}$$

Where $P = \prod_{n=1}^{N} I_{\mathrm{n}}$ and $Q = \prod_{n=1}^{N} J_{\mathrm{n}}$.

Therefore,

$$\left[\nabla_{\boldsymbol{\Phi}^{(n)}} f(\boldsymbol{\Phi})\right]_{i,j} = Tr\left(\left(\nabla_{\boldsymbol{\Phi}} f(\boldsymbol{\Phi})\right)^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}^{(n)}_{i,j}}\right) \tag{B.26}$$

F.2 Proof of Proposition 7.2

***Proposition 7.2*:** *Let $\boldsymbol{\Phi} \in \mathbb{R}^{P\times Q}$ be a Kronecker matrix, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$, $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n\times J_n}; \forall\ n \in \{1, \cdots, N\}$, and $\|\boldsymbol{\Phi}\|_2$ is the L_2 norm of $\boldsymbol{\Phi}$. Therefore, the gradient $\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2$ is given by,*

$$\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2 = 2\gamma_{\boldsymbol{\Phi}^{(n)}} \boldsymbol{\Phi}^{(n)} \tag{B.27}$$

Where

$$\gamma_{\boldsymbol{\Phi}^{(n)}} = \prod_{m=1, m\neq n}^{N} Tr\left({\boldsymbol{\Phi}^{(m)}}^T \boldsymbol{\Phi}^{(m)}\right)$$

Proof:

Using Proposition 7.1, we could write,

$$\left[\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2\right]_{i,j} = Tr\left(\left(\frac{\partial \|\boldsymbol{\Phi}\|_2}{\partial \boldsymbol{\Phi}}\right)^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right) \tag{B.28}$$

$$\Longrightarrow \left[\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2\right]_{i,j} = 2Tr\left(\boldsymbol{\Phi}^T \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right) \tag{B.29}$$

However, $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and,

$$\frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)} \tag{B.30}$$

Since $\mathrm{Tr}(\mathrm{A} \otimes \mathrm{B}) = \mathrm{Tr}(\mathrm{A})\mathrm{Tr}(\mathrm{B})$

$$\begin{aligned} &\left[\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2\right]_{i,j} \\ &= 2\left(\prod_{m=1, m\neq n}^{N} Tr\left({\boldsymbol{\Phi}^{(m)}}^T \boldsymbol{\Phi}^{(m)}\right)\right) Tr\left({\boldsymbol{\Phi}^{(n)}}^T \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right) \end{aligned} \tag{B.31}$$

However,

$$Tr\left({\boldsymbol{\Phi}^{(n)}}^T \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right) = \boldsymbol{\Phi}_{i,j}^{(n)} \tag{B.32}$$

Therefore,

$$\nabla_{\boldsymbol{\Phi}^{(n)}} \|\boldsymbol{\Phi}\|_2 = 2\gamma_{\boldsymbol{\Phi}^{(n)}} \boldsymbol{\Phi}^{(n)} \tag{B.33}$$

Where

$$\gamma_{\boldsymbol{\Phi}^{(n)}} = \prod_{m=1, m \neq n}^{N} Tr\left(\boldsymbol{\Phi}^{(m)^T} \boldsymbol{\Phi}^{(m)} \right)$$

F.3 Proof of Proposition 7.3

***Proposition 7.3*:** Let $f(\mathcal{X}, \mathcal{Y}, \boldsymbol{\Phi})$ be a function of tensor $\mathcal{X} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$, tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$and a Kronecker matrix $\boldsymbol{\Phi} \in \mathbb{R}^{P \times Q}$, where $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$ and $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{I_n \times J_n}$; $\forall\, n \in \{1, \cdots, N\}$. If

$$\frac{\partial f}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = Tr\left(\text{vec}(\mathcal{Y})^T \left(\frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \text{vec}(\mathcal{X}) \right) \right) \qquad (\text{B}.34)$$

Then $\frac{\partial f}{\partial \boldsymbol{\Phi}^{(n)}}$ is given by,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}^{(n)}} = \mathcal{Y}_{(n)} \left(\mathcal{X}_{(n)} \boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}}^T \right)^T \qquad (\text{B}.35)$$

Where, $\boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}} = \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n+1)} \otimes \boldsymbol{\Phi}^{(n-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)} \right)$, $\mathcal{Y}_{(n)}$ is the *mode-n* matricization of the tensor $\mathcal{Y}$ and $\mathcal{X}_{(n)}$ is the *mode-n* matricization of the tensor $\mathcal{X}$.

Proof:

We could rewrite $(\text{B}.34)$ as an inner product between the tensor $\mathcal{Y}$ and the multilinear transformation of the tensor $\mathcal{X}$ as,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = \langle \mathcal{Y}, \left(\mathcal{X} \times_1 \boldsymbol{\Phi}^{(1)} \times_2 \cdots \times_n \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \cdots \times_N \boldsymbol{\Phi}^{(N)} \right) \rangle \qquad (\text{B}.36)$$

Where $\frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = \boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}$

Therefore,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = \langle \mathcal{Y}_{(n)}, \frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} \mathcal{X}_{(n)} \boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}}^{T} \rangle \tag{B.37}$$

Where $\mathcal{X}_{(n)}$ and $\mathcal{Y}_{(n)}$ are *mode-n* matricization of respective tensors and,

$$\boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}} = \left(\boldsymbol{\Phi}^{(N)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(n+1)} \otimes \boldsymbol{\Phi}^{(n-1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(1)}\right) \tag{A.38}$$

The element $\left(\frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right)_{i,j} = 1$ and 0 everywhere else in the gradient matrix $\left(\frac{\partial \boldsymbol{\Phi}^{(n)}}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}}\right)$. Therefore,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}_{i,j}^{(n)}} = \langle \left(\mathcal{Y}_{(n)}\right)_{i,*}, \left(\mathcal{U}_{(n)}{}^{T}\right)_{*,j} \rangle \tag{B.39}$$

Where $\left(\mathcal{Y}_{(n)}\right)_{i,*}$ denote the ith row of the *mode-n* matrix $\mathcal{Y}_{(n)}$, $\mathcal{U}_{(n)} = \mathcal{X}_{(n)} \boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}}^{T}$, and $\left(\mathcal{U}_{(n)}{}^{T}\right)_{*,j}$ denotes the *j*th column of the transposed *mode-n* matrix $\mathcal{U}_{(n)}{}^{T}$.

Therefore,

$$\frac{\partial f}{\partial \boldsymbol{\Phi}^{(n)}} = \mathcal{Y}_{(n)} \mathcal{U}_{(n)}{}^{T} = \mathcal{Y}_{(n)} \left(\mathcal{X}_{(n)} \boldsymbol{\Psi}_{\boldsymbol{\Phi}^{(n)}}^{T}\right)^{T} \tag{B.40}$$