Tensor-based Regression Models and Applications

Thèse

Ming Hou

Doctorat en informatique
Philosophiæ doctor (Ph.D.)

Québec, Canada

Résumé

Avec l'avancement des technologies modernes, les tenseurs d'ordre élevé sont assez répandus et abondent dans un large éventail d'applications telles que la neuroscience informatique, la vision par ordinateur, le traitement du signal et ainsi de suite. La principale raison pour laquelle les méthodes de régression classiques ne parviennent pas à traiter de façon appropriée des tenseurs d'ordre élevé est due au fait que ces données contiennent des informations structurelles multi-voies qui ne peuvent pas être capturées directement par les modèles conventionnels de régression vectorielle ou matricielle. En outre, la très grande dimensionnalité de l'entrée tensorielle produit une énorme quantité de paramètres, ce qui rompt les garanties théoriques des approches de régression classique. De plus, les modèles classiques de régression se sont avérés limités en termes de difficulté d'interprétation, de sensibilité au bruit et d'absence d'unicité.

Pour faire face à ces défis, nous étudions une nouvelle classe de modèles de régression, appelés modèles de régression tensor-variable, où les prédicteurs indépendants et (ou) les réponses dépendantes prennent la forme de représentations tensorielles d'ordre élevé. Nous les appliquons également dans de nombreuses applications du monde réel pour vérifier leur efficacité et leur efficacité.

Abstract

With the advancement of modern technologies, high-order tensors are quite widespread and abound in a broad range of applications such as computational neuroscience, computer vision, signal processing and so on. The primary reason that classical regression methods fail to appropriately handle high-order tensors is due to the fact that those data contain multiway structural information which cannot be directly captured by the conventional vector-based or matrix-based regression models, causing substantial information loss during the regression. Furthermore, the ultrahigh dimensionality of tensorial input produces huge amount of parameters, which breaks the theoretical guarantees of classical regression approaches. Additionally, the classical regression models have also been shown to be limited in terms of difficulty of interpretation, sensitivity to noise and absence of uniqueness.

To deal with these challenges, we investigate a novel class of regression models, called tensor-variate regression models, where the independent predictors and (or) dependent responses take the form of high-order tensorial representations. We also apply them in numerous real-world applications to verify their efficiency and effectiveness.

Concretely, we first introduce *hierarchical Tucker tensor regression*, a generalized linear tensor regression model that is able to handle potentially much higher order tensor input. Then, we work on *online local Gaussian process for tensor-variate regression*, an efficient nonlinear GP-based approach that can process large data sets at constant time in a sequential way. Next, we present a computationally efficient online tensor regression algorithm with general tensorial input and output, called *incremental higher-order partial least squares*, for the setting of infinite time-dependent tensor streams. Thereafter, we propose a super-fast sequential tensor regression framework for general tensor sequences, namely *recursive higher-order partial least squares*, which addresses issues of limited storage space and fast processing time allowed by dynamic environments. Finally, we introduce *kernel-based multiblock tensor partial least squares*, a new generalized nonlinear framework that is capable of predicting a set of tensor blocks by merging a set of tensor blocks from different sources with a boosted predictive power.

Table des matières

Résumé **iii**

Abstract **v**

Table des matières **vii**

Abbreviations **ix**

List of Symbols **xi**

Liste des tableaux **xiii**

Liste des figures **xv**

Acknowledgements **xvii**

1 Introduction **1**
1.1 Context and Motivations for Tensor Regression 1
1.2 Tensor-based Regression Models and Their Applications 3
1.3 Main Contributions 5
1.4 Thesis Outline 6

2 Tensor Preliminaries **9**
2.1 Tensor Basics 10
2.2 Tensor Decompositions 17
2.3 Scaling up Tensor Decompositions 25
2.4 Conclusion 27

3 Tensor Regression Overview **29**
3.1 Tensor Regression 29
3.2 Our new contributions: the big picture 49
3.3 Conclusion 49

4 Hierarchical Tucker Tensor Regression **51**
4.1 Introduction 51
4.2 Hierarchical Tucker Decomposition (HTD) 52
4.3 $\mathcal{H}$-Tucker Tensor Regression Model 57
4.4 Parameter Estimation 59
4.5 Experimental Results 62

4.6 Conclusion . . . 65

5 Online Local Gaussian Process for Tensor Regression 67
5.1 Introduction . . . 67
5.2 Tensor GP Regression Review . . . 68
5.3 Tensor OLGP Regression . . . 69
5.4 Experimental Results . . . 72
5.5 Conclusion . . . 75

6 Incremental Higher-order Partial Least Squares Regression (IHOPLS) 77
6.1 Introduction . . . 77
6.2 High-order Partial Least Squares Regression (HOPLS) Review . . . 79
6.3 Incremental Higher-order Partial Least Square Regression (IHOPLS) . . . 79
6.4 Experimental Results . . . 82
6.5 Discussion . . . 86
6.6 Conclusion . . . 86

7 Recursive Higher-order Partial Least Squares Regression (RHOPLS) 89
7.1 Introduction . . . 89
7.2 Recursive Higher-order Partial Least Squares Regression (RHOPLS) . . . 90
7.3 Experimental Results . . . 98
7.4 Discussion . . . 109
7.5 Conclusion . . . 110

8 Partial Least Squares Regression: A Kernel-based Multiblock Tensor Approach 111
8.1 Introduction . . . 111
8.2 Kernel-based Multiblock Tensor PLS Regression (KMTPLS) . . . 112
8.3 Experimental Results . . . 118
8.4 Discussion . . . 126
8.5 Conclusion . . . 126

9 Conclusion 127
9.1 Summary of the Contributions . . . 127
9.2 Possible Future Work . . . 129

A Mathematical Background 131
A.1 Partial Least Squares Regression (PLS) . . . 131
A.2 Nonlinear Iterative Partial Least Squares PLS Regression (NIPALS-PLS) . . . 134
A.3 Linear Algebra Basics . . . 135
A.4 Generalized Linear Model (GLM) . . . 136
A.5 Maximum Likelihood Estimation (MLE) . . . 137

B Sketch of Proof 139
B.1 Sketch of Proof of Proposition 3 . . . 139

Bibliographie 141

Abbreviations

ADHD	Attention Deficit Hyperactivity Disorder Data Analysis
ADMM	Alternating Direction Method of Multiplier
ALTO	Accelerated Low-rank Tensor Online Learning
ALM	Augmented Lagrangian Method
ALS	Alternating Least Squares
ANN	Artificial Neural Networks
BCD	Block Component Decomposition
BIC	Bayesian Information Criterion
BRA	Block Relaxation Algorithm
CD	Coordinate Descent
CDMTR	Common and Discriminative multiblock Tensor Regression
CP	Canonical Decomposition/Parallel Factor Analysis
CNMF	Convolutive Nonnegative Matrix Factorization
C-PARAFAC	Convolutive Parallel Factor Analysis
ECoG	Electrocorticography
ED	Eigenvalue Decomposition
EEG	Electroencephalography
FMRI	Functional Magnetic Resonance Image
GLM	Generalized Linear Model
GP	Gaussian Process
HOOI	Higher-Order Orthogonal Iteration
HOPLS	Higher-Order Partial Least Squares
HOSVD	Higher-Order Singular Value Decomposition
HS	Hilbert Space
HTD	Hierarchical Tensor Decomposition
IHOPLS	Incremental Higher-Order Partial Least Squares
ITP	Iterative Tensor Projection
JS	Jensen-Shannon Divergence

KL	Kullback-Leibler Divergence
KMTPLS	Kernel-based Multiblock Tensor Partial Least Squares
KTPLS	Kernel-based Tensor Partial Least Squares
MHAD	Multimodal Human Action Database
MLE	Maximum Likelihood Estimation
MLMTL	Multilinear Multitask Learning
MMCR	Multiway Multiblock Covariates Regression
MRI	Magnetic Resonance Image
MTPLS	Multiblock Tensor Partial Least Squares
MTR	Multiblock Tensor Regression
MTTKRP	Matricized Tensor Times Khatri-Rao Product
NLL	Negative Log Likelihood
NPLS	N-Way Partial Least Squares
NIPALS	Nonlinear Iterative Partial Least Squares
OLGP	Online Local Gaussian Process
OMP	Orthogonal Matching Pursuit
PARAFAC2	Parallel Factor Analysis 2
PCA	Principal Components Analysis
PLS	Partial Least Squares
RHOPLS	Recursive Higher-Order Partial Least Squares
RMSE	Root Mean Square Error
RPLS	Recursive Partial Least Squares
RIP	Restricted Isometry Property
SED	Symmetric Eigenvalue Decomposition
SGD	Stochastic Gradient Descent
S-PARAFAC	Shifted Parallel Factor Analysis
SVD	Singular Value Decomposition
SVR	Support Vector Regression
Tensor GP	Tensor Gaussian Process
TPG	Tensor Projected Gradient
UMPM	Utrecht Multi-Person Motion
VAR	Vector Auto Regressive

List of Symbols

$a, b, c, \cdots$	Scalars
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \cdots$	Vectors
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \cdots$	Matrices
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \cdots$	Higher-order Tensors
$\Phi, \Psi, \cdots$	Higher-order Tensors in High Dimensional Hilbert Space
$\mathbf{A} \otimes \mathbf{B}$	Kronecker Product of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \odot \mathbf{B}$	Khatri-Rao Product of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} * \mathbf{B}$	Hadamard Product of $\mathbf{A}$ and $\mathbf{B}$
vec	Vectorization Operator
$\mathbf{A}_{(d)}$	d-mode Matricization of $\mathcal{A}$
$\mathbf{A}^{(d)}$	d-mode Factor Matrix
$\mathbf{a} \circ \mathbf{b}$	Outer Product of $\mathbf{a}$ and $\mathbf{b}$
$\mathcal{A} \circ \mathcal{B}$	Outer Product of $\mathcal{A}$ and $\mathcal{B}$
$\langle \mathcal{A}, \mathcal{B} \rangle$	Inner Product of $\mathcal{A}$ and $\mathcal{B}$
$\langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,C;1,\ldots,C}$	Contracted Product of $\mathcal{A}$ and $\mathcal{B}$ along First C Modes
$\mathcal{A} \times_d \mathbf{B}$	d-mode Product of $\mathcal{A}$ and $\mathbf{B}$
$\mathcal{A} \bar{\times}_d \mathbf{b}$	d-mode Product of $\mathcal{A}$ and $\mathbf{b}$
$\|\mathbf{A}\|_1$	$l1$ Norm of $\mathbf{A}$
$\|\mathcal{A}\|_\mathsf{F}$	Frobenius Norm of $\mathcal{A}$
$\|\mathcal{A}\|_\mathsf{tr}$	Overlapped Trace Norm of $\mathcal{A}$
$\|\mathcal{A}\|_\mathsf{scaled}$	Scaled Latent Trace Norm of $\mathcal{A}$
$\mathbf{A}^\mathsf{T}$	Transpose of $\mathbf{A}$
$\mathbf{A}^\dagger$	Pseudo Inverse of $\mathbf{A}$
$\mathbf{S}_\uparrow$	Vertical Shift Operator in Up Direction
$I, J, K, M, \cdots$	Upper Indices
$i, j, k, m, \cdots$	Running Indices

Liste des tableaux

3.1 Summarization of linear tensor regression models. 43
3.2 Summarization of nonlinear tensor regression models. 49
3.3 The overview of our new contributions. 49

4.1 Performance comparison for the misclassification error of $\mathcal{H}$-Tucker regression and Tucker regression model on ADHD data. 65

5.1 Computational complexity of tensor GP and tensor OLGP using product probabilistic tensor kernel. 71
5.2 Performance comparison for the prediction of movement of shoulder marker along x-axis on ECoG data, with data size=10 000. 72
5.3 Performance comparison for the prediction of movement of shoulder marker along x-axis on ECoG data, with data size=36 000 and $w_{gen} = 0.4$. 73

6.1 Performance comparison of IHOPLS, HOPLS and RNPLS for the averaged Q, RMSEP and total learning time on UMPM data. 83
6.2 Performance comparison of IHOPLS, HOPLS and RNPLS for the averaged Q, RMSEP and total learning time on ECoG data. 86

7.1 Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time with $I_0 = 200$ and $b = 2$ on UMPM data. 103
7.2 Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time with $I_0 = 20\%$ of the training set and $b = 2$ on ECoG data. 107
7.3 Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time for $L = [12, 16]$, $K = [3, 10]$, $b = 2$ on MHAD data. 109
7.4 Forecasting performance for $lag = 3$, trained with 50% of all the time series, $b = 1$ on CCDS data. 110

8.1 Performance comparison of KMTPLS and MMCR for the optimal Rank, Q and RMSEP on UMPM data. 118
8.2 Performance comparison of KMTPLS and MMCR for the averaged Q, RMSEP and learning time on MHAD data. 123

Liste des figures

2.1 Illustration of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. 9
2.2 Illustration of 1-mode fibers (left), 2-mode fibers (middle) and 3-mode fibers (right) of a third-order tensor. 10
2.3 Illustration of horizontal slices (left), vertical slices (middle) and frontal slices (right) of a third-order tensor. 11
2.4 Illustration of the matricization of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ in the first-mode (top) and second-mode (bottom). 13
2.5 Illustration of 2-mode tensor matrix multiplication $\mathcal{Y} = \mathcal{X} \times_2 \mathbf{A}$. 15
2.6 Illustration of tensor vector multiplication in all the modes of a third-order tensor. 16
2.7 Illustration of rank-one third-order tensor $\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \mathbf{a}^{(3)} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. . . 17
2.8 Illustration of R-component CP decomposition of a third-order tensor. 18
2.9 Illustration of PARAFAC2 of a third-order tensor. 21
2.10 Illustration of Tucker decomposition of a third-order tensor. 22
2.11 Illustration of block component decomposition of a third-order tensor. 25

3.1 An illustration of high-order partial least squares (HOPLS) for $M = 2$ and $L = 2$. 33
3.2 An illustration of tensor regression layer (TRL). 48

4.1 Illustration of a balanced binary dimension tree $\mathcal{T}$ for a 5-order tensor. 54
4.2 Illustration of $\mathcal{H}$-Tucker format for a 5-order tensor. 56
4.3 Performance comparison vs. number of samples for case of 4-order tensor. . . . 63
4.4 Performance comparison vs. number of samples for case of 5-order tensor. . . . 64

5.1 RMSE (top) and NLL (bottom) vs. number of training samples, $w_{gen} = 0.5$, $R = 6$. 74
5.2 Learning time vs. number of training samples, $w_{gen} = 0.5$, $R = 6$. 75
5.3 RMSE (top) and NLL (bottom) vs. number of local experts. 76

6.1 Prediction error (top) and CPU cost (bottom) of three methods over time, $R = 8$ and $\lambda = 4$ for IHOPLS and HOPLS for sequence length of 1050 and frequency at 5fps on UMPM. 84
6.2 Prediction error (top) and CPU cost (bottom) of three methods over time, $R = 8$ and $\lambda = 4$ for IHOPLS and HOPLS for sequence length of 5250 and frequency at 25fps on UMPM. 85
6.3 Prediction errors versus N_{max} for $R = 8$ $\lambda = 8$ $w_{gen} = 0.4$ on ECoG data. . . . 87

7.1 The RHOPLS scheme. The framework generates a set of initial factors for the initial data (**Step 0** red arrow). At every iteration, the framework first generates a set of incremental factors for the new data (**Step 1** yellow arrow). Then, the information contained in new data, represented in terms of factors, is added to current model by an appending operation (**Step 2** blue arrow). Next, the augmented set of factors are truncated back into the ones with original sizes to yield new loadings (**Step 3** purple arrow). The new individual core tensors are produced using an internal tensor representation of model (in terms of factors) under the projection of the new loadings (**Step 4** green arrow). 92
7.2 The initial approximation step of RHOPLS framework for $t = 0$. 93
7.3 The incremental approximation step of RHOPLS framework for $t = 1$. 94
7.4 The expansion step of RHOPLS framework for $t = 1$. 95
7.5 The compression step of RHOPLS framework for $t = 1$. 96
7.6 The projection step of RHOPLS framework for $t = 1$. 97
7.7 The whole RHOPLS scheme. 100
7.8 For "triangle" scenario, learning error and learning time versus the iteration. . . 101
7.9 For "table" scenario, learning error and learning time versus the iteration. . . . 102
7.10 The accuracy and learning time versus the number of latent vectors. 104
7.11 The accuracy and learning time versus the number of initial samples. 105
7.12 Performance versus sequence length (frequency). 106
7.13 An example of ground truth (150s time window) and the trajectories predicted by RHOPLS, HOPLS and RNPLS for Z-coordinate of the monkey's hand. . . . 108

8.1 Illustration of the framework of kernel-based multiblock tensor partial least squares (KMTPLS) regression. 114
8.2 Performance comparison of KMTPLS and MMCR for the Q versus the different combination of cameras on UMPM data. 119
8.3 Performance comparison of KMTPLS for the Q when using the optimal rank versus the relative importance α on UMPM data. 120
8.4 Visualization of ground truth and the trajectories predicted by MMCR and KMTPLS in the "table" scenario. 121
8.5 Performance comparison of KMTPLS and MMCR for the best Q, RMSEP versus the number of input blocks on jumping action on MHAD data. 124
8.6 Performance comparison of KMTPLS and MMCR for the best Q, RMSEP versus the number of input blocks on bending action on MHAD data. 125

A.1 Illustration of the framework of partial least squares (PLS) regression. 132

Acknowledgements

I would like to express my sincerest gratitude to my supervisor, Prof. Brahim Chaib-draa, for his tremendous support to help me open the door of research world and guide me through the entire Ph.D study with his great advices. His insightful instructions always inspire me to open my mind and discover novel ideas ; his great patience and continuous encouragement are source of my strength to keep trying and never give up ; his professional attitudes and activities in research influence me a lot and will benefit me for my whole research career.

I owe my gratitude to Dr. Yali Wang, who helps me a lot with valuable suggestions and guidance at early stage of my research.

I am grateful to Dr. Qibin Zhao and Prof. Andrzej Cichocki from RIKEN brain science institute, who provide me with a lot of precious advices and constructive comments.

I also thankful to Prof. Philippe Giguère, Prof. Claude-Guy Quimper and Prof. Jean-François Lalonde for their valuable guidance and wonderful courses, which expand my research horizons and help me lay solid foundations in research.

Finally, and most importantly, I dedicate this thesis to my great parents, my mom Jingyun Ma and my dad Jingbang Hou, who have always been there for me and supporting me with their deepest unconditional love.

Chapitre 1

Introduction

This chapter starts the thesis by introducing the context of our research and its motivations. Then, it gives the details about our contributions. Finally, it outlines the content of this thesis.

1.1 Context and Motivations for Tensor Regression

Regression is a statistical technique that attempts to study the relationship between two or more variables [Draper and Smith, 2014; Chatterjee and Hadi, 2015; Montgomery et al., 2015]. Specifically, regression analysis is able to predict one or more dependent variables (responses, outputs) from a set of independent variables (predictors, inputs), by exploring the correlations among these variables as well as explaining the factors behind the observed patterns. Regression is especially important because it can help you predict or analyze practically all types of data generated from complex systems in a large variety of applications. For instance, regression allows you to model property loss from fire as a function of a collection of variables such as degree of fire department involvement, response time and property value etc. If you find that response time is the key factor, you may need to build more fire stations. Extracting hidden structure and examining variables relationships, you may need to increase equipments or officers dispatched if you find that involvement variable is the key factor. Another example is that people may want to employ regression models to predict the amount of rainfall of next year based on the gauges from historical records.

The most commonly used regression models can typically be categorized, in terms of linearity, into linear regression and nonlinear regression. In particular, linear regression models describe relationship between dependent and independent variables using a linear function in parameters. Apart from simple regression model, linear regression models include multiple regression associated with multiple predictors, and multivariate regression corresponding to the case of linear regression model with multiple predictors and multiple responses. Normally, one may often encounter multivariate regression tasks where both the predictors and responses are arranged as vectors of variables or matrices of variables.

As an extension of ordinary linear regression models, the generalized linear model (GLM) [Nelder and Baker, 1972] is capable of modeling response variables via a particular distribution other than normal distribution from the exponential family, which unifies logistic regression [Hosmer and Lemeshow, 2000], multinomial regression, Poisson regression [Cameron and Trivedi, 2013] and so on. Additionally, the class of partial least squares (PLS) [Wold et al., 1984; Abdi, 2010] models also belongs to the category of linear regression models.

Unlike linear regression modelings, nonlinear regression models characterizing nonlinear dependencies in data are generally assumed to be parametric, in which responses are modeled as a function of a combination of nonlinear parameters and predictors, where nonlinear parameters usually take the form of an exponential function, trigonometric function, power function, etc. Nonparametric nonlinear regression models frequently appear in the context of machine learning including Gaussian process (GP) [Rasmussen and Williams, 2005], artificial neural networks (ANN) [Haykin and Network, 2004], decision trees [Quinlan, 1986], support vector regression (SVR) [Smola and Vapnik, 1997] and so on.

During the past few years, regular regression approaches, though being considered as mature techniques, have confronted new great challenges posed by numerous real-world regression tasks whose predictors and (or) responses take the form of high-order high-dimensional arrays with complex structures, also known as tensors [Kolda and Bader, 2009; Cichocki et al., 2009; Acar and Yener, 2009; Cichocki, 2013; Cichocki et al., 2015; Cong et al., 2015]. With the advancement of modern technologies, such high-order tensors (multiway arrays) are quite widespread and abound in a broad range of applications, including analytical chemistry [Andersen and Bro, 2003; Bro, 2006], computational neuroscience [Miwakeichi et al., 2004; Andersen and Rayens, 2004], computer vision [Vasilescu and Terzopoulos, 2002; Wang and Ahuja, 2008], industrial process control [Luo et al., 2015], data mining [Acar et al., 2005; Kolda et al., 2005] and so on.

For instance, wavelet-transformed multichannel electroencephalogram (EEG) data can be organized as a third-order tensor with modes *time* $\times$ *frequency* $\times$ *channels* [Miwakeichi et al., 2004; Cichocki, 2013; Cichocki et al., 2015; Cong et al., 2015]; fluorescence spectroscopic data can be arranged as a third-order tensor with modes *samples* $\times$ *excitation wavelengths* $\times$ *emission wavelengths* [Andersen and Bro, 2003]; video sequences can naturally be modeled as third-order tensor with modes *x-coordinates* $\times$ *y-coordinates* $\times$ *frames* [Wang and Ahuja, 2008] and face images with various conditions can be represented as a fifth-order tensor having modes *pixels* $\times$ *illuminations* $\times$ *expressions* $\times$ *viewpoints* $\times$ *identities* [Vasilescu and Terzopoulos, 2002]. In addition, the related tensor data analysis (multiway data analysis) [Kolda and Bader, 2009; Acar and Yener, 2009] approaches and tools have also been studied extensively in a variety of fields, with the fundamental goals of exploring the correlations among variables as well as discovering the underlying hidden patterns so as to find a a summarization of tensor data.

The primary reason why regular regression methods fail to appropriately handle high-order tensors is due to the fact that those data contain multiway structural information (i.e., the information about the patterns of correlations among different modalities in high-order tensors) which cannot be directly captured by the conventional vector-based or matrix-based regression models. For instance, in the application of brain imaging data analysis, we aim to establish the associations between the magnetic resonance imaging (MRI) brain images and the clinical outcomes. This problem can be formulated as a regression task with predictor variable taking the form of 3-order tensor and with response variable being the scalar clinical outcome, predicting whether the subject is healthy or not.

In order to address the high-order input, it is a common practice to apply classical regression techniques to tensorial brain images by first transforming them into vectors (matrices) via some vectorization (unfolding) operations and then feeding them to classical models. Nevertheless, there are two significant challenges as a consequence of this procedure. Firstly, the ultrahigh dimensionality of tensorial input like $3D$ or $4D$ brain images results in a huge amount of parameters, which undoubtedly leads to an overfitting situation as well as breaks the theoretical guarantees of classical regression approaches. Secondly, using such operations of vectorizing (or unfolding) as proposed will inevitably destroy underlying multiway structure contained in tensorial input and lose intrinsic correlations among the pixels, causing substantial information loss during the regression.

In addition to above issues, the classical regression models based on the matrix analysis approaches have also been shown to be inferior in terms of difficulty of interpretation, sensitivity to noise as well as absence of uniqueness [Acar and Yener, 2009]. Let us take the analysis of third-order multichannel EEG data as an example. If we unfold the predictor in the *channels* mode (i.e., resultant matrix with mode *channels* $\times$ *time-frequency*) and employ matrix-based regression model (i.e., unfolded PLS regression [Abdi, 2010], see Appendix A.1), then the set of factors (signatures, patterns) extracted from the *time-frequency* mode become very hard to interpret, leading to the difficulties in understanding the brain activities associated with those two factors (signatures, patterns).

1.2 Tensor-based Regression Models and Their Applications

Motivated by the extensive applicability of tensor data and the limitations of classical regression models, we propose to investigate in this thesis the *tensor based regression models and their applications*. More specifically, we aim to develop a novel class of regression models dealing with high-order tensorial predictors and (or) high-order tensorial responses and apply them in various real-life applications. The work is accomplished by making use of multiway factor modelings (i.e., canonical decomposition/parallel factor analysis (CP) model [Carroll and Chang, 1970; Harshman, 1970] or Tucker model [Tucker, 1963; De Lathauwer et al., 2000a])

and their related tensor (multiway) analysis approaches [Kolda and Bader, 2009; Cichocki et al., 2009; Acar and Yener, 2009]. In general, these models and analysis approaches are more beneficial over their matrix-based counterparts since they are designed to naturally represent and preserve the multiway nature of the tensorial data. Therefore, by combing multiway modeling ideas with classical regression models, we provided efficient tensor-variate regression models which are able to effectively capture the underlying inherent structural information in tensors, resulting in significantly improved predictability.

If we now model the previous MRI images using R-component CP model instead of explicit long vector, then the number of parameters will be drastically reduced from $\mathcal{O}(I^D)$ to $\mathcal{O}(DIR)$, where D is tensor order while I stands for the maximum size in each mode and $R \ll I$. In this case, not only the correlations among pixels are preserved, but also the CP-based model is in fact statistically simpler, which is not prone to overfitting for the case of MRI application where the number of observations is rather limited.

Once again, referring back to the previous EEG example, if we apply multiway PLS model [Bro, 1996] to third-order EEG array, we are able to extract a set of factors that individually accounts for the temporal pattern of brain activities from *time* mode as well as the spectral pattern of brain activities associated with *frequency* mode, respectively. The latent scores, extracted from *channels* mode, can easily be interpreted as the coefficients of the combination of different brain activity patterns, which are used to measure their relative contributions. Such advantage in the ease of interpretation can further result in the property of robustness to noise since the number of components in model can be reasonably specified according to the scientific assumption and experimental results.

Thus, tensor-variate regression, as we can see, is free of the above discussed limitations of conventional regression methods and is expected as a promising research direction. In this thesis, we investigate and extend several tensor-variate modeling strategies from different perspectives, including the aspect of multilinear model with potentially much higher-order tensor input, the aspect of nonlinear online model with scalar output, the aspect of multilinear online model with general tensor input and tensor output. We also investigate the aspect of model merging tensor blocks from different sources for boosting the predictive power. Furthermore, our proposed tensor-variate regression models and algorithms have been validated on a number of real-world representative applications from different disciplines such as the prediction of brain disease from medical MRI images in computational neuroscience, the reconstruction of limb trajectories from monkey's brain signals in neural signal processing, the estimation of human pose positions from video sequences in computer vision, etc.

In the following sections, we summarize our contributions and describe the outline of this thesis.

1.3 Main Contributions

1.3.1 Hierarchical Tucker Tensor Regression : Application to Brain Imaging Data Analysis

In [Hou and Chaib-draa, 2015], we present a novel generalized linear tensor regression model, which takes tensor-variate inputs as predictors and finds low-rank almost best approximation of regression coefficient arrays using hierarchical Tucker decomposition. With limited sample size, our model is highly compact and very efficient as it requires only $\mathcal{O}(DR^3 + DIR)$ parameters for order D tensors of mode size I and rank R. Thus, it avoids the exponential growth in D, in contrast to $\mathcal{O}(R^D + DIR)$ parameters of Tucker regression modeling. Our model also maintains the flexibility like classical Tucker regression by allowing distinct ranks on different modes according to a dimension tree structure. We validated our new model on synthetic data, and we also applied it to real-life MRI images to show its effectiveness in predicting the human brain disease of attention deficit hyperactivity disorder.

1.3.2 Online Local Gaussian Process for Tensor-Variate Regression : Application to Fast Reconstruction of Limb Movement from Brain Signal

In [Hou et al., 2015], we overcome the computational issues of existing tensor Gaussian process (tensor GP) regression approaches for large data sets by introducing a computationally-efficient tensor-variate regression approach in which the latent function is flexibly modeled by using online local Gaussian process (OLGP). By doing so, the large data set is efficiently processed by constructing a number of small-sized GP experts in an online fashion. Furthermore, we introduce two searching strategies to find local GP experts to make accurate predictions with a Gaussian mixture representation. Finally, we apply our approach to a real-life regression task, reconstruction of limb movements from brain signal, to show its effectiveness and scalability for large data sets.

1.3.3 Online Incremental Higher-order Partial Least Squares Regression for Fast Reconstruction of Motion Trajectories from Tensor Streams

The higher-order partial least squares (HOPLS) is considered as the state-of-the-art tensor-variate regression modeling for predicting a tensor response from a tensor input. However, the standard HOPLS can quickly become computationally prohibitive or merely intractable, especially when huge and time-evolving tensorial streams arrive over time in dynamic application environments. In [Hou and Chaib-draa, 2016], we present a computationally efficient online tensor regression algorithm, namely incremental higher-order partial least squares (IHOPLS), for adapting HOPLS to the setting of infinite time-dependent tensor streams. By incrementally clustering the projected latent variables in latent space and summarizing the previous data,

IHOPLS is able to recursively update the projection matrices and core tensors over time, resulting in greatly reduced costs in terms of both memory and running time while maintaining high prediction accuracy. For the experiments, we apply IHOPLS to two real-life applications, i.e., reconstruction of $3D$ motion trajectories from videos and ECoG streaming signals.

1.3.4 Fast Recursive Tensor Sequential Learning for Regression

In this specific work, we develop a super-fast sequential tensor regression framework, namely recursive higher-order partial least squares (RHOPLS) [Hou and Chaib-draa, 2017]. It addresses the great challenges posed by the limited storage space and fast processing time allowed by dynamic environments when dealing with very large-scale high-speed general tensor sequences. Smartly integrating a low-rank modification strategy of the Tucker into partial least squares (PLS), we efficiently update the PLS-based regression coefficients by effectively merging the new data into the previous low-rank approximation of the model at a small-scale factor (feature) level instead of the large raw data (observation) level. Unlike batch approaches, which require accessing the entire data, RHOPLS conducts a blockwise consecutive calculation scheme and thus it stores only a small set of factors. Our approach is orders of magnitude faster than all other methods while maintaining a highly comparable predictability with the cutting-edge batch methods, as verified on challenging real-life tasks.

1.3.5 Common and Discriminative Subspace Kernel-based Multiblock Tensor Partial Least Squares Regression

In [Hou et al., 2016], we introduce a new generalized nonlinear tensor regression framework called kernel-based multiblock tensor partial least squares (KMTPLS). With this framework, we can predict a set of dependent tensor blocks from a set of independent tensor blocks through the extraction of a small number of common and discriminative latent components. By considering both common and discriminative features, KMTPLS effectively fuses the information from multiple tensorial data sources and unifies the single and multiblock tensor regression scenarios into one general model. Moreover, in contrast to multilinear model, KMTPLS successfully addresses the nonlinear dependencies between multiple responses and predictor tensor blocks by combining kernel machines with joint Tucker decomposition, resulting in a significant performance gain in terms of predictability. An efficient learning algorithm for KMTPLS based on sequentially extracting common and discriminative latent vectors is also presented. Finally, as for the applications, we employ KMTPLS on a regression task in computer vision, i.e., reconstruction of human pose from multiview video sequences.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 gives an overview of the basic tensor definitions, its associated operations and the most commonly used tensor decomposition tools and

their extensions. Chapter 3 summarizes the concepts of tensor regression models and the recent advances in the work related to what we are proposing. The hierarchical Tucker tensor regression model is presented in Chapter 4. Chapter 5 introduces the nonlinear online local Gaussian process for regression task with tensor input and scalar output. Another two online tensor regression models based on PLS framework are introduced in Chapter 6 and 7, with the goal of addressing the problem of speed and storage complexity when dealing with general tensor input and tensor output. Chapter 8 proposes a generalized nonlinear tensor regression framework, as an effective tool of integrating tensorial data from different sources, for predicting a set of dependent tensor blocks from a set of independent tensor blocks. Finally, Chapter 9 concludes the thesis and discuss some future research directions.

Chapitre 2

Tensor Preliminaries

This chapter reviews several fundamental parts about tensor that are necessary for understanding tensor-variate regression in the remaining chapters. It begins by introducing some notations of tensor and some basic definitions of tensor algebra. Then it presents the key concept of tensor decomposition (factorization), also known as multiway models or multilinear models. This concept turns out to be an extremely useful tool to decompose a high-order tensor object into a collection of factors in low-dimensional subspace. Subsequently, it introduces several extended variants of tensor decomposition that are very useful for the regression problems as studied in this thesis. Finally, it gives a high-level overview of various strategies for scaling up tensor decompositions.

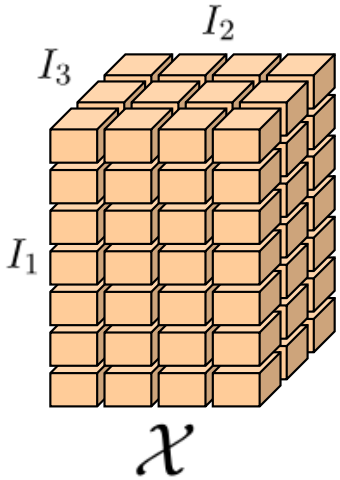


FIGURE 2.1 – Illustration of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

2.1 Tensor Basics

2.1.1 Tensor Definitions and Notations

Tensors [Kolda and Bader, 2009; Cichocki et al., 2009; Cichocki, 2013; Cichocki et al., 2015; Cong et al., 2015] are higher-order generalizations of vectors and matrices. They also refer to as multiway arrays of real numbers. Throughout the thesis, higher-order tensors will be denoted as $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ in calligraphy letters, where D is the *order* of $\mathcal{X}$. Matrices represented by boldface capital letters $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ are tensors of order two, while the vectors denote first-order tensors and are represented by boldface low-case letters $\mathbf{x} \in \mathbb{R}^{I_1}$. Here, the order D represents the number of dimensions and each dimension $d \in D$ of tensor is called *mode* or *way* [Tucker, 1963, 1964]. The number of variables I_d in the mode d is used to represent the dimensionality of that mode. In other words, D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ has D modes (ways) with dimensionality I_d in the mode d, where $d = 1, ..., D$. The ith entry of a vector $\mathbf{x}$ is denoted by x_i and the (i, j) entry of a matrix $\mathbf{X}$ is denoted by $x_{i,j}$. Likewise, we denote the entry $(i_1, i_2, ..., i_D)$ of D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ as $x_{i_1,i_2,...,i_D}$. An example of a third order tensor with three modes is illustrated in Figure 2.1.

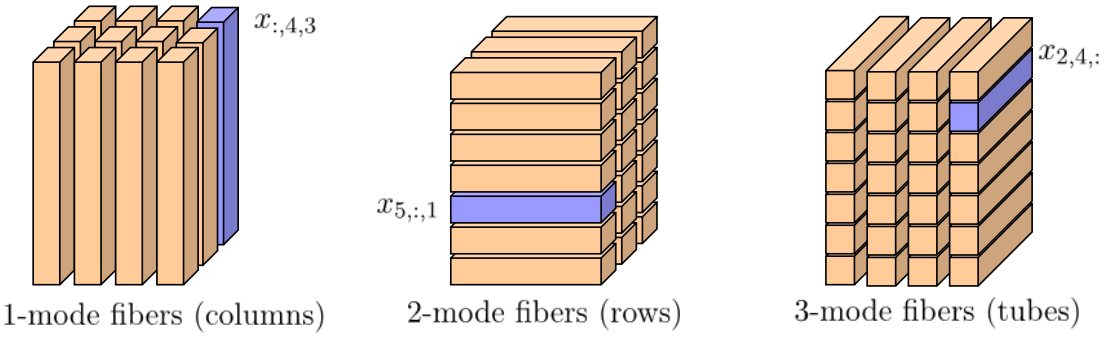


FIGURE 2.2 – Illustration of 1-mode fibers (left), 2-mode fibers (middle) and 3-mode fibers (right) of a third-order tensor.

The *d-mode vector* of tensor $\mathcal{X}$, also known as *d-mode fiber*, is an element of $\mathbb{R}^{I_d}$, which is obtained by varying the index I_d while keeping other indices fixed. For instance, for a third order tensor, a fiber $x_{:,4,3}$ from 1-mode fibers (columns), a fiber $x_{5,:,1}$ from 2-mode fibers (rows) and a fiber $x_{2,4,:}$ from 3-mode fibers (tubes) are shown in highlighted color in Figure 2.2, respectively. Here we use " : " to denote free indices that range over a specific mode.

Similarly, if we vary two indices by fixing the other one index in a third order tensor, then we will get a *slice* (*slab*), i.e., a horizontal slice $x_{3,:,:}$, a vertical slice $x_{:,4,:}$ and a frontal slice $x_{:,:,1}$ are shown in highlighted color in Figure 2.3.

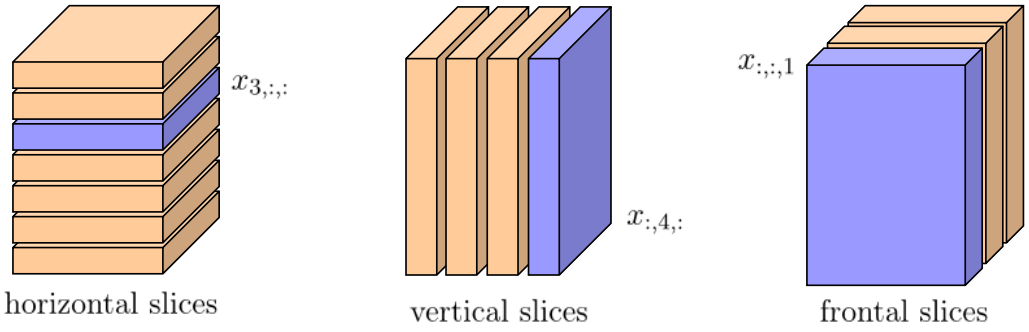


FIGURE 2.3 – Illustration of horizontal slices (left), vertical slices (middle) and frontal slices (right) of a third-order tensor.

2.1.2 Relevant Matrix Algebra

The *Kronecker product* [Smilde et al., 2005] between matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$ and matrix $\mathbf{Y} \in \mathbb{R}^{K \times L}$ yielding a block matrix of size $IK \times JL$ is denoted by symbol $\otimes$ as

$$\mathbf{X} \otimes \mathbf{Y} = \begin{pmatrix} x_{11}\mathbf{Y} & x_{12}\mathbf{Y} & \cdots & x_{1j}\mathbf{Y} \\ x_{21}\mathbf{Y} & x_{22}\mathbf{Y} & \cdots & x_{2j}\mathbf{Y} \\ \cdots & \cdots & \cdots & \cdots \\ x_{i1}\mathbf{Y} & x_{i2}\mathbf{Y} & \cdots & x_{ij}\mathbf{Y} \end{pmatrix} \in \mathbb{R}^{IK \times JL}, \tag{2.1}$$

and $\mathbf{X} \otimes \mathbf{Y}$ can also be expressed in form of columnwise Kronecker product as

$$\begin{aligned} \mathbf{X} \otimes \mathbf{Y} &= [\mathbf{x}_1 \otimes \mathbf{y}_1, \mathbf{x}_1 \otimes \mathbf{y}_2, \mathbf{x}_1 \otimes \mathbf{y}_3, ..., \mathbf{x}_J \otimes \mathbf{y}_{L-1}, \mathbf{x}_J \otimes \mathbf{y}_L] \in \mathbb{R}^{IK \times JL} \\ &= [vec(\mathbf{y}_1\mathbf{x}_1^\mathsf{T}), vec(\mathbf{y}_2\mathbf{x}_1^\mathsf{T}), vec(\mathbf{y}_3\mathbf{x}_1^\mathsf{T}), ..., vec(\mathbf{y}_{L-1}\mathbf{x}_J^\mathsf{T}), vec(\mathbf{y}_L\mathbf{x}_J^\mathsf{T})], \end{aligned} \tag{2.2}$$

where *vec* represents the vectorization operation.

The mixed-product property of Kronecker product, which will be found useful in this thesis, states that if $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{A}$ and $\mathbf{B}$ are conformable matrices whose dimensions are suitable for multiplications such that the matrix product $\mathbf{XY}$ and $\mathbf{AB}$ can be formed, then we have

$$(\mathbf{X} \otimes \mathbf{A})(\mathbf{Y} \otimes \mathbf{B}) = \mathbf{XY} \otimes \mathbf{AB}. \tag{2.3}$$

The *Khatri-Rao product* [Smilde et al., 2005] between matrix $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_J] \in \mathbb{R}^{I \times J}$ and $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_J] \in \mathbb{R}^{K \times J}$ having the equal number of columns is defined using columnwise Kronecker product of matrices as

$$\mathbf{X} \odot \mathbf{Y} = [\mathbf{x}_1 \otimes \mathbf{y}_1, \mathbf{x}_2 \otimes \mathbf{y}_2, ..., \mathbf{x}_J \otimes \mathbf{y}_J] \in \mathbb{R}^{IK \times J}. \tag{2.4}$$

The *Hadamard product* [Smilde et al., 2005] between matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$ and matrix $\mathbf{Y} \in \mathbb{R}^{I \times J}$ with equal matrix size is the entrywise product denoted by symbol $*$

$$\mathbf{X} * \mathbf{Y} = \begin{pmatrix} x_{11}y_{11} & x_{12}y_{12} & \cdots & x_{1j}y_{1j} \\ x_{21}y_{21} & x_{22}y_{22} & \cdots & x_{2j}y_{2j} \\ \cdots & \cdots & \cdots & \cdots \\ x_{i1}y_{i1} & x_{i2}y_{i2} & \cdots & x_{ij}y_{ij} \end{pmatrix} \in \mathbb{R}^{I \times J}. \tag{2.5}$$

2.1.3 Tensor Matricization

For high-order tensors, it is usually beneficial to be represented using vectors or matrices so that the powerful matrix analysis techniques can readily be applied to the lower subspaces.

The *d-mode matricization (unfolding, flattening)* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ [Kiers, 2000; De Lathauwer et al., 2000a], defined as $\mathbf{X}_{(d)} \in \mathbb{R}^{I_d \times I_1 \cdots I_{d-1} I_{d+1} \cdots I_D}$, is the process of rearranging the d-mode vectors (fibers) into the columns of the resulting matrix. In other words, according to a prespecified order, we collect all d-mode vectors of a tensor and then concatenate them side by side, leading to a matrix with size $I_d \times I_1 \cdots I_{d-1} I_{d+1} \cdots I_D$.

Mathematically, the mapping from an entry $(i_1, i_2, ..., i_D)$ of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ to an entry (i_d, j) of the unfolded matrix can be established by

$$j = 1 + \sum_{t \neq d}^{D} (i_t - 1) J_t \quad \text{with} \quad J_t = \prod_{p \neq t}^{t-1} i_p. \tag{2.6}$$

Figure 2.4 demonstrates the examples where a third order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is unfolded in the first-mode and second-mode, resulting in a matrix $\mathbf{X}_{(1)}$ of size $I_1 \times I_2 I_3$ and a matrix $\mathbf{X}_{(2)}$ of size $I_2 \times I_1 I_3$, respectively.

The *vectorization* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is defined as

$$vec(\mathcal{X}) = vec(\mathbf{X}_{(1)}), \tag{2.7}$$

that is, the vectorization of the corresponding first-mode unfolded matrix $\mathbf{X}_{(1)}$.

Specifically, the vectorization of a matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$ is obtained by stacking the matrix columns $\{\mathbf{x}_j\}_{j=1}^{J}$ into a long vector $\mathbf{x} \in \mathbb{R}^{IJ}$

$$\mathbf{x} = vec(\mathbf{X}) = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ ... \\ \mathbf{x}_J \end{pmatrix} \in \mathbb{R}^{IJ}. \tag{2.8}$$

Two important properties associated with the matrix vectorization operator are

$$vec(\mathbf{X})^{\mathsf{T}} vec(\mathbf{Y}) = trace(\mathbf{X}^{\mathsf{T}} \mathbf{Y}) \tag{2.9}$$

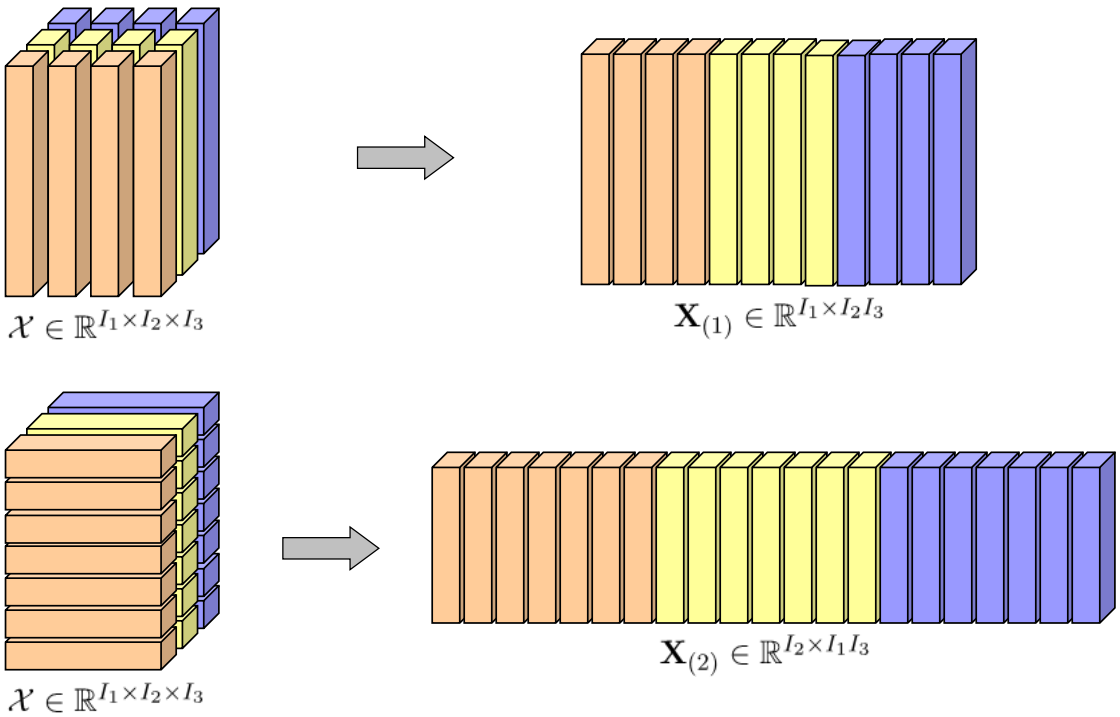


FIGURE 2.4 – Illustration of the matricization of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ in the first-mode (top) and second-mode (bottom).

and

$$vec(\mathbf{XYZ}) = (\mathbf{Z}^\mathsf{T} \otimes \mathbf{X})\, vec(\mathbf{Y}), \tag{2.10}$$

where $\otimes$ stands for the standard Kronecker product.

2.1.4 Tensor Multiplication

The *outer product* of tensor $\mathcal{X} \in \mathbb{R}^{J_1 \times \cdots \times J_P}$ and tensor $\mathcal{Y} \in \mathbb{R}^{K_1 \times \cdots \times K_Q}$, which produces a tensor of size $J_1 \times \cdots \times J_P \times K_1 \times \cdots \times K_Q$, is denoted as

$$\mathcal{Z} = \mathcal{X} \circ \mathcal{Y} \tag{2.11}$$

with each entry satisfying

$$z_{j_1,\ldots,j_P,k_1,\ldots,k_Q} = x_{j_1,\ldots,j_P} y_{k_1,\ldots,k_Q}. \tag{2.12}$$

The *inner product* between two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ having the same order and equal size is given by

$$z = \langle \mathcal{X}, \mathcal{Y} \rangle = \langle vec(\mathcal{X}), vec(\mathcal{Y}) \rangle, \tag{2.13}$$

where $z \in \mathbb{R}$ is a scalar. In entrywise form, we have

$$z = \langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_D=1}^{I_D} x_{i_1,\ldots,i_D} y_{i_1,\ldots,i_D}. \tag{2.14}$$

Having defined the tensor inner product, the Frobenius norm of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ follows the definition as

$$\|\mathcal{X}\|_{\mathsf{F}} = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}. \tag{2.15}$$

The *contracted product* between $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_C \times J_1 \times \cdots \times J_P}$ and $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_C \times K_1 \times \cdots \times K_Q}$ with equal size along the first C modes yields a tensor $\mathcal{Z} \in \mathbb{R}^{J_1 \times \cdots \times J_P \times K_1 \times \cdots \times K_Q}$ in the following element expression

$$\begin{aligned} z_{j_1,\ldots,j_P,k_1,\ldots,k_Q} &= \langle \mathcal{X}, \mathcal{Y} \rangle_{1,\ldots,C;1,\ldots,C}(j_1, \ldots, j_P, k_1, \ldots, k_Q) \\ &= \sum_{i_1=1}^{I_1} \cdots \sum_{i_C=1}^{I_C} x_{i_1,\ldots,i_C,j_1,\ldots,j_P} y_{i_1,\ldots,i_C,k_1,\ldots,k_Q}, \end{aligned} \tag{2.16}$$

which means the entries in the resulting tensor are obtained by summing out the product of each corresponding entry pair along the first few common indices between two tensor multipliers.

On one hand, it is a simple matter to see that the tensor inner product is actually a special case of the tensor contracted product when all the modes are in common between two tensor multipliers, e.g., $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_C}$ and $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_C}$

$$z = \langle \mathcal{X}, \mathcal{Y} \rangle = \langle \mathcal{X}, \mathcal{Y} \rangle_{1,\ldots,C;1,\ldots,C}. \tag{2.17}$$

On the other hand, with no common indices between two tensors, the tensor contracted product turns out to be the tensor outer product as introduced in (2.11)

$$\mathcal{Z} = \mathcal{X} \circ \mathcal{Y} = \langle \mathcal{X}, \mathcal{Y} \rangle_{0;0}. \tag{2.18}$$

The *tensor-matrix multiplication* is generalized from standard matrix multiplication via matrix matricization. Specifically, when a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is multiplied by a matrix $\mathbf{A} \in \mathbb{R}^{J_d \times I_d}$, it is first unfolded in the dth mode to obtain $\mathbf{X}_{(d)}$, then the matrix product $\mathbf{Y} = \mathbf{A}\mathbf{X}_{(d)}$ is computed. Finally, the resulting $\mathbf{Y}$ is reshaped back to a tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{d-1} \times J_d \times I_{d+1} \times \cdots \times I_D}$. This operation is depicted in Figure 2.5 for the case of 2-mode tensor matrix multiplication of a third order tensor. We refer such tensor-matrix multiplication as the *d-mode tensor matrix product* of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ and matrix $\mathbf{A} \in \mathbb{R}^{J_d \times I_d}$

$$\mathcal{Y} = \mathcal{X} \times_d \mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_{d-1} \times J_d \times I_{d+1} \times \cdots \times I_D}, \tag{2.19}$$

which can also be expressed in an elementwise form

$$y_{i_1,\ldots,i_{d-1},j_d,i_{d+1},\ldots,i_D} = \sum_{i_d=1}^{I_d} y_{i_1,\ldots,i_d,\ldots,i_D} a_{j_d,i_d}. \tag{2.20}$$

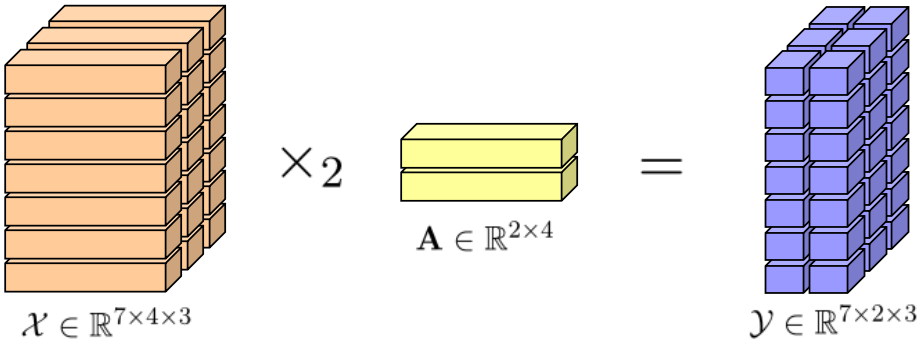


FIGURE 2.5 – Illustration of 2-mode tensor matrix multiplication $\mathcal{Y} = \mathcal{X} \times_2 \mathbf{A}$.

Assuming the conforming dimensions among the matrices and tensors, one can easily verify the following properties :

$$\begin{aligned} &\langle \mathcal{X} \times_d \mathbf{A}, \mathcal{Y} \rangle = \langle \mathcal{X}, \mathcal{Y} \times_d \mathbf{A}^\mathsf{T} \rangle, \\ &(\mathcal{X} \times_c \mathbf{A}) \times_d \mathbf{B} = (\mathcal{X} \times_d \mathbf{B}) \times_c \mathbf{A}, \quad c \neq d, \\ &(\mathcal{X} \times_d \mathbf{A}) \times_d \mathbf{B} = \mathcal{X} \times_d \mathbf{B}\mathbf{A}, \end{aligned} \tag{2.21}$$

and

$$(\mathcal{X} \times_1 \mathbf{A}^{(1)} \cdots \times_D \mathbf{A}^{(D)})_{(d)} = \mathbf{A}^{(d)} \mathbf{X}_{(d)} (\mathbf{A}^{(D)} \otimes \cdots \otimes \mathbf{A}^{(d+1)} \otimes \mathbf{A}^{(d-1)} \otimes \cdots \otimes \mathbf{A}^{(1)})^\mathsf{T}. \tag{2.22}$$

Similarly, *d-mode tensor vector product* between a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ and a vector $\mathbf{a} \in \mathbb{R}^{I_d}$ is defined as

$$\mathcal{Y} = \mathcal{X} \,\bar{\times}_d\, \mathbf{a} \in \mathbb{R}^{I_1 \times \cdots \times I_{d-1} \times I_{d+1} \times \cdots \times I_D}. \tag{2.23}$$

In element form, we get

$$y_{i_1,\ldots,i_{d-1},i_{d+1},\ldots,i_D} = \sum_{i_d=1}^{I_d} y_{i_1,\ldots,i_d,\ldots,i_D} a_{i_d}. \tag{2.24}$$

If we multiply tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ by a set of vectors $\{\mathbf{a} \in \mathbb{R}^{I_d}\}_{d=1}^{D}$, each of which is associated with one specific mode d, then the result is a scalar formulated by

$$y = \mathcal{X} \,\bar{\times}_1\, \mathbf{a}^{(1)} \,\bar{\times}_2\, \cdots \,\bar{\times}_D\, \mathbf{a}^{(D)} \in \mathbb{R}. \tag{2.25}$$

An example of tensor vector multiplication of a third order tensor is illustrated in Figure 2.6, in which a third order tensor $\mathcal{X} \in \mathbb{R}^{7\times 4\times 3}$ is sequentially multiplied by vectors $\mathbf{a}_1 \in \mathbb{R}^7$, $\mathbf{a}_2 \in \mathbb{R}^4$ and $\mathbf{a}_3 \in \mathbb{R}^3$ along the mode 1, 2 and 3, respectively, ending up with a scalar $y \in \mathbb{R}$.

When satisfying the relative ordering between the modes, the commutative property also holds for tensor vector product as below

$$(\mathcal{X} \,\bar{\times}_c\, \mathbf{a}) \,\bar{\times}_d\, \mathbf{b} = (\mathcal{X} \,\bar{\times}_d\, \mathbf{b}) \,\bar{\times}_c\, \mathbf{a} \quad \text{for} \quad c < d. \tag{2.26}$$

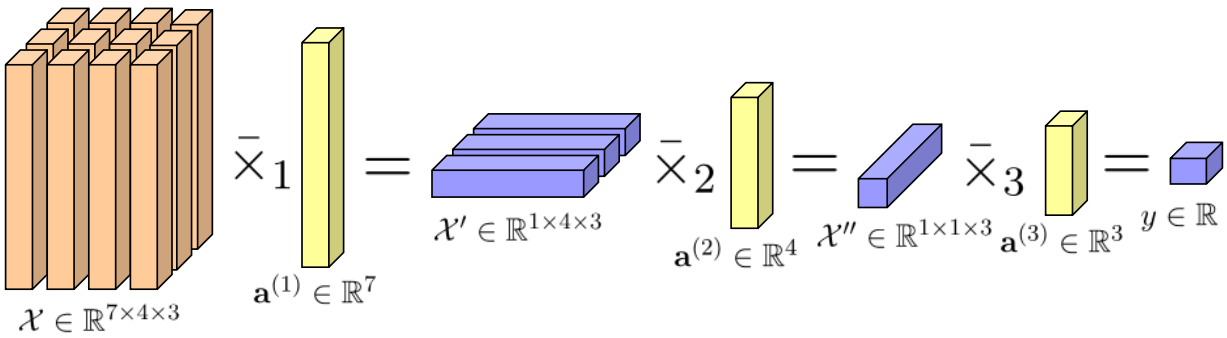


FIGURE 2.6 – Illustration of tensor vector multiplication in all the modes of a third-order tensor.

2.1.5 Tensor Rank

The *d-rank* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_D}$, denoted as $rank_d(\mathcal{X})$, is the column rank of the unfolding matrix $\mathbf{X}_{(d)}$. In other words, it is the dimension of the vector space spanned by d-mode vectors. In the context of d-mode matricization, we can establish that

$$rank_d(\mathcal{X}) = rank(\mathbf{X}_{(d)}). \tag{2.27}$$

A tensor $\mathcal{X}$ for which $r_d = rank_d(\mathcal{X})$ for $d = 1, ..., D$ is called a *rank-*$(r_1, r_2, ..., r_D)$ tensor, and the *D-tuple* $(r_1, r_2, ..., r_D)$ is defined as the *multilinear rank* of $\mathcal{X}$.

In addition to multilinear rank, an alternative notion of rank named tensor rank exists and it is built on the basis of rank-one tensor. The D-order tensor $\mathcal{X}$ is *rank-one tensor* if it consists of the outer product of D vectors $\{\mathbf{a}^{(d)} \in \mathbb{R}^{I_d}\}_{d=1}^{D}$

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(D)}, \tag{2.28}$$

where $\circ$ is the vector outer product introduced in (2.11). Writing this in an equivalent entry expression, it gives

$$x_{i_1,i_2,...,i_D} = \mathbf{a}^{(1)}_{i_1}\mathbf{a}^{(2)}_{i_2}\cdots\mathbf{a}^{(D)}_{i_D}. \tag{2.29}$$

An example of rank-one third-order tensor is shown in Figure 2.7, where it factorizes as the outer product of three vectors $\mathbf{a}^{(1)}$, $\mathbf{a}^{(2)}$ and $\mathbf{a}^{(3)}$ from three modes.

The *tensor rank*, namely $rank(\mathcal{X})$, is defined to be the minimum number of the sum of rank-one tensor that can exactly factorize tensor $\mathcal{X}$

$$rank(\mathcal{X}) := \arg\min\{R \in \mathbb{N} : \mathcal{X} = \sum_{r=1}^{R} \mathbf{a}^{(1)}_r \circ \mathbf{a}^{(2)}_r \circ \cdots \circ \mathbf{a}^{(D)}_r\}. \tag{2.30}$$

Particularly, for matrices (second-order tensors), the following equation holds

$$rank_1(\mathcal{X}) = rank_2(\mathcal{X}) = rank(\mathcal{X}). \tag{2.31}$$

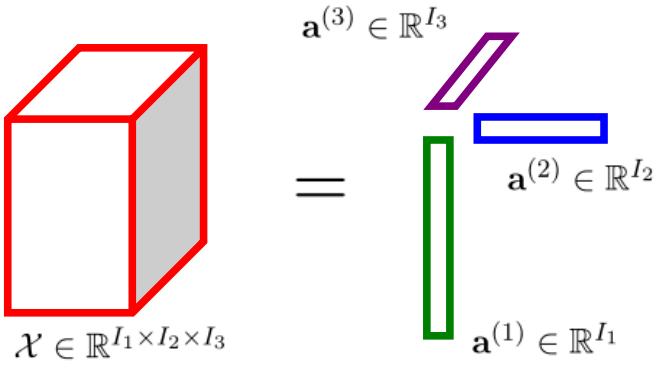


FIGURE 2.7 – Illustration of rank-one third-order tensor $\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \mathbf{a}^{(3)} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

2.2 Tensor Decompositions

The strength of tensor modelings lies in its associated decomposition tools that are capable of representing high-order data in terms of low-dimensional factors. The concept of tensor decomposition (factorization) was originally introduced by Hitchcock [1927, 1928]. Nowadays, higher-order tensor decompositions (factorizations) are frequently applied in a variety of fields such as chemometrics, neuroscience, image/video analysis and signal processing. Most of existing decomposition models can trace back to two fundamental decomposition formats, that are canonical decomposition/parallel factor analysis (or CANDECOMP/PARAFAC, or simply CP) [Carroll and Chang, 1970; Harshman, 1970; Kiers, 2000] decomposition and Tucker [Tucker, 1963; De Lathauwer et al., 2000a] decomposition.

2.2.1 CP decomposition

The *CP decomposition* [Carroll and Chang, 1970; Harshman, 1970; Kiers, 2000] initially introduced as the polyadic form [Hitchcock, 1927] of a tensor generalizes the bilinear factor models to multilinear data. Mathematically, the R-component CP model boils down to factorizing a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ as a linear combination of rank-one tensors as

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(D)} + \varepsilon \tag{2.32}$$

or in equivalent elementwise form

$$x_{i_1,i_2,\ldots,i_D} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_{i_1,r}^{(1)} \mathbf{a}_{i_2,r}^{(2)} \cdots \mathbf{a}_{i_D,r}^{(D)} + \varepsilon_{i_1,i_2,\ldots,i_D}, \tag{2.33}$$

where the symbol $\circ$ is the outer product operator. The normalized unit vector $\mathbf{a}_r^{(d)}$ for $r = 1, \ldots, R$ indicates the rth column of factor matrix $\mathbf{A}^{(d)} = [\mathbf{a}_1^{(d)}, \mathbf{a}_2^{(d)}, \ldots, \mathbf{a}_R^{(d)}] \in \mathbb{R}^{I_d \times R}$ from d mode. Furthermore, λ_r is the scalar weighting the different rank-one tensor components

and R is the tensor rank. ε is the residual with same size as tensor $\mathcal{X}$. For instance, the top part of Figure 2.8 demonstrates a R-component CP decomposition of a three-way array, where $\mathbf{a}_r^{(1)}, \mathbf{a}_r^{(2)}$ and $\mathbf{a}_r^{(3)}$ correspond to the factors associated with rth sub-tensor component extracted from the corresponding modes, respectively.

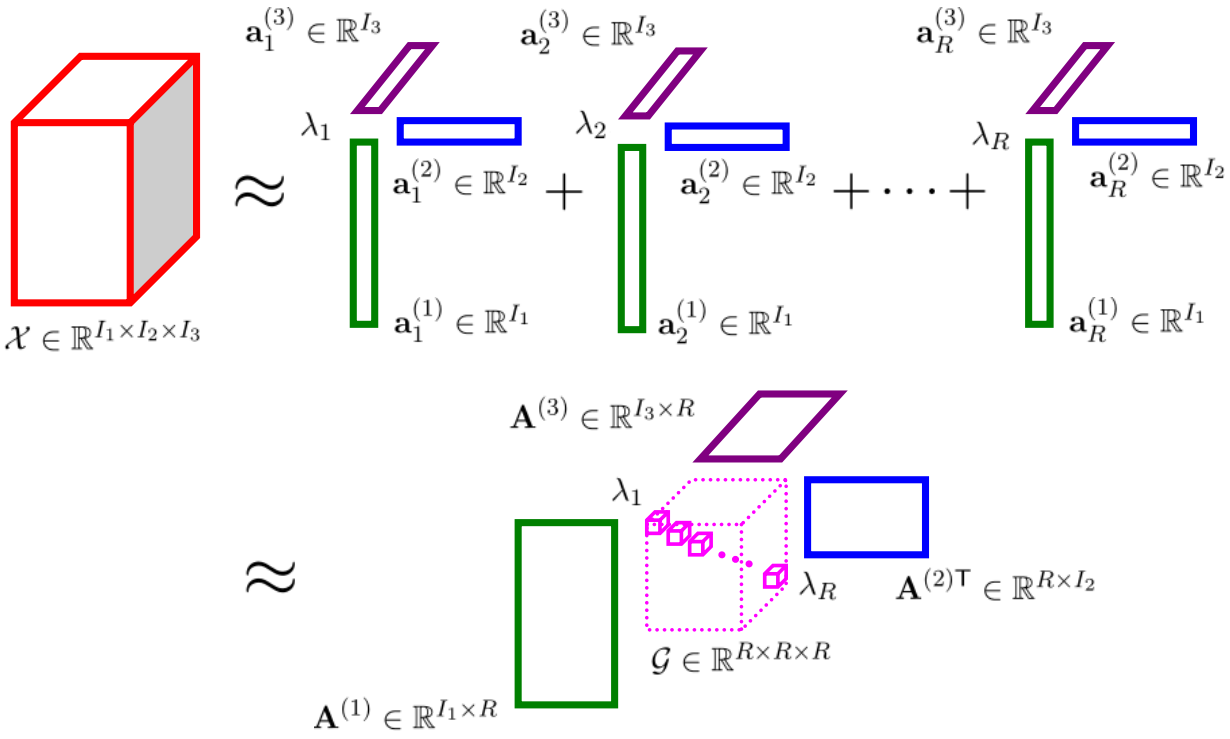


FIGURE 2.8 – Illustration of R-component CP decomposition of a third-order tensor.

The CP in equation (2.32) can also be converted into a more compact matrix form in terms of d-mode unfolding matrix $\mathbf{X}_{(d)}$ using Khatri-Rao product defined in (2.4)

$$\mathbf{X}_{(d)} = \mathbf{A}^{(d)} \Lambda (\mathbf{A}^{(D)} \odot \cdots \odot \mathbf{A}^{(d+1)} \odot \mathbf{A}^{(d-1)} \odot \cdots \odot \mathbf{A}^{(1)})^{\mathsf{T}} + \mathbf{E}, \tag{2.34}$$

where diagonal matrix $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_R)$ and $\mathbf{E}$ represents the residual. Applying vectorization property (2.10) to matrix form (2.34), we reach the vectorized form of CP as

$$vec(\mathcal{X}) = (\mathbf{A}^{(D)} \odot \cdots \odot \mathbf{A}^{(1)})\, vec(\Lambda) + vec(\varepsilon). \tag{2.35}$$

For the special case of third-order tensor, (2.32) can be rewritten in a alternative matrix form as

$$\mathbf{X}_i = \mathbf{A}^{(1)} \mathbf{D}_i \mathbf{A}^{(2)\mathsf{T}} + \mathbf{E}_i \quad \text{for} \quad i = 1, ..., I_3, \tag{2.36}$$

where $\mathbf{X}_i$ corresponds to the ith frontal slice of a third-order tensor $\mathcal{X}$. $\mathbf{D}_i = diag(a_{i,:}^{(3)})$ is a diagonal matrix with diagonal elements from the ith row of third mode factor matrix $\mathbf{A}^{(3)}$. $\mathbf{E}_i$ is the error associated with the ith frontal slice. Note that here in (2.36) the factor matrices $\{\mathbf{A}^{(d)}\}_{d=1}^{3}$ are not normalized.

For the simplicity of notation, the CP model can be expressed in a succinct way [Kolda and Bader, 2009] as

$$\mathcal{X} \approx [\Lambda, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(D)}]. \tag{2.37}$$

One favorable property about CP model is that, under certain mild conditions [Kruskal, 1989], one is able to obtain a unique CP factorization up to permutation indeterminacy and scaling indeterminacy. In other words, the CP model can be uniquely identified except that columns of factor matrices, i.e., $\mathbf{A}^{(d)} = [\mathbf{a}_1^{(d)}, \mathbf{a}_2^{(d)}, ..., \mathbf{a}_R^{(d)}]$ with $d = 1, ..., D$, can be arbitrarily reordered. It can be scaled as long as product of scaling coefficients of columns remains constant.

Nevertheless, such benefit is achieved at the cost of strict constraints imposed on the CP model, whose factors in distinct modes can only interact factorwise with each other. In other words, CP model is inadequate in terms of degrees of freedom. For example in Figure 2.8, the factor $\mathbf{a}_r^{(1)}$ in the first mode is required to communicate merely with the factor $\mathbf{a}_r^{(2)}$ in the second mode and the factor $\mathbf{a}_r^{(3)}$ in the third mode, leading to the same number of factors for all modes. Another disadvantage is that a CP approximation may be ill-posed and may produce unstable estimation of its components [Cichocki et al., 2009].

As a generalization of singular value decomposition (SVD) (see Appendix A.3) to high-order tensor, CP factorizes a tensor as a sum of rank-one tensors, which is analogous to SVD in the sense that SVD decomposes a matrix into a sum of rank-one matrices. Nevertheless, unlike SVD, the orthogonality of CP on factor matrices can hardly be satisfied.

It is worth noting that CP decomposition only demands the storage of $\mathcal{O}(DIR)$ entries in contrast to that of $\mathcal{O}(I^D)$ for $I = max\{I_d\}_{d=1}^D$, which become very attractive when $R \ll I$.

The classical algorithms for computing CP model [Carroll and Chang, 1970; Harshman, 1970] are based on the alternating least squares (ALS) style algorithm [Kroonenberg and De Leeuw, 1980] that estimates one factor matrix at a time by keeping the rest factor matrices fixed. Mathematically, the objective is to minimize the approximation error by iteratively solving the following least square problem, with respect to one specific factor matrix $\mathbf{A}^{(d)}$ at a time

$$\min \|\mathcal{X} - \bar{\mathcal{X}}\| = \min \|\mathbf{X}_{(d)} - \mathbf{A}^{(d)}(\mathbf{A}^{(D)} \odot \cdots \odot \mathbf{A}^{(d+1)} \odot \mathbf{A}^{(d-1)} \odot \cdots \odot \mathbf{A}^{(1)})^\mathsf{T}\|, \tag{2.38}$$

where $\odot$ refers to the Khatri-Rao operator. Thus, the solution of (2.38) regarding the d-mode factor matrix $\mathbf{A}^{(d)}$ can easily be computed as

$$\mathbf{A}^{(d)} = \mathbf{X}_{(d)}[(\mathbf{A}^{(D)} \odot \cdots \odot \mathbf{A}^{(d+1)} \odot \mathbf{A}^{(d-1)} \odot \cdots \odot \mathbf{A}^{(1)})^\mathsf{T}]^\dagger \tag{2.39}$$

with $\dagger$ being as the pseudo inverse. Following the similar pattern, the algorithm alternately updates factor matrices for all $d = 1..., D$ modes and stops when a certain convergence criterion is met. Though as a simple and practical solution for computing CP, the ALS-based algorithm does not guarantee a global optimum due to the non-convex property of (2.38).

Additionally, ALS-based algorithm has a slow convergence rate [Cichocki et al., 2009]. The CP-ALS procedure is summarized in Algorithm 1, in which equation (2.39) corresponds to line 6 and 7.

Algorithm 1 CP-ALS [Carroll and Chang, 1970; Harshman, 1970]

1: **Input**: D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, tensor rank R
2: **Output**: coefficients $\{\lambda_d\}_{d=1}^{D}$, factor matrices $\{\mathbf{A}^{(d)} \in \mathbb{R}^{I_d \times R}\}_{d=1}^{D}$
3: **Initialize**: randomly initialize $\{\mathbf{A}^{(d)}\}_{d=1}^{D}$
4: **repeat**
5: **for** $d = 1, ..., D$ **do**
6: $\mathbf{T}_d \leftarrow \mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)} * \cdots * \mathbf{A}^{(d-1)\mathsf{T}}\mathbf{A}^{(d-1)} * \mathbf{A}^{(d+1)\mathsf{T}}\mathbf{A}^{(d+1)} * \cdots * \mathbf{A}^{(D)\mathsf{T}}\mathbf{A}^{(D)}$
7: $\mathbf{A}^{(d)} \leftarrow \mathbf{X}_{(d)}(\mathbf{A}^{(D)} \odot \cdots \odot \mathbf{A}^{(d+1)} \odot \mathbf{A}^{(d-1)} \odot \cdots \odot \mathbf{A}^{(1)})\mathbf{T}_d^{\dagger}$
8: **end for**
9: **until** the convergence criterion is satisfied

One notable extension of standard CP model is referred to as PARAFAC2 [Harshman, 1972]. It was introduced to model third-order tensors with frontal slices having different dimensionality in one mode. For example, the data from industrial batch process control, e.g., *time duration* × *variables* × *batches*, usually varies in *time duration* mode due to the unavoidable disturbances and changes of operating conditions, leading to batch data with uneven-length frontal slices. In fact, PARAFAC2 simultaneously decomposes a collection of matrices with each having equal number of columns but different row size. Unlike CP model that applies the same factors across all frontal slices (e.g., $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ linked with $\mathbf{X}_i$ for $i = 1, ..., I_3$ in (2.36)), PARAFAC2 relaxes CP to allow for distinct factors associated with different frontal slices in the first mode. More formally, PARAFAC2 model can be derived from formulation (2.36) by enforcing additional constraint on factor matrix in the first mode as

$$\begin{aligned} \mathbf{X}_i &= \mathbf{A}_i^{(1)}\mathbf{D}_i\mathbf{A}^{(2)\mathsf{T}} + \mathbf{E}_i \\ \text{s.t} \quad \mathbf{A}_i^{(1)\mathsf{T}}\mathbf{A}_i^{(1)} &= \alpha \quad \text{for} \quad i = 1, ..., I_3, \end{aligned} \tag{2.40}$$

where the 1-mode factor matrix $\mathbf{A}_i^{(1)}$ corresponds to ith frontal slice. α is a constant. The constraint imposed on $\mathbf{A}_i^{(1)}$ indicates that the matrix product of $\mathbf{A}_i^{(1)}$ with its transpose is invariant for all frontal slices $\{\mathbf{X}_i\}_{i=1}^{I_3}$ of a third order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. Such constraint is introduced for the purpose of inducing the uniqueness properties of PARAFAC2 [Harshman, 1972]. Figure 2.9 shows an example of PARAFAC2 decomposition of an irregular third-order tensor having 4 frontal slices with different 1-mode sizes.

Another well-known extension of CP model called shifted PARAFAC (s-PARAFAC) is proposed by Harshman et al. [2003] with the aim to handle the shifting factors in sequential data, e.g., time series or spectral data. Compared to ordinary CP format, s-PARAFAC is relaxed to allow the factor matrix, in one mode of a third-order tensor, to be shifted by certain amount

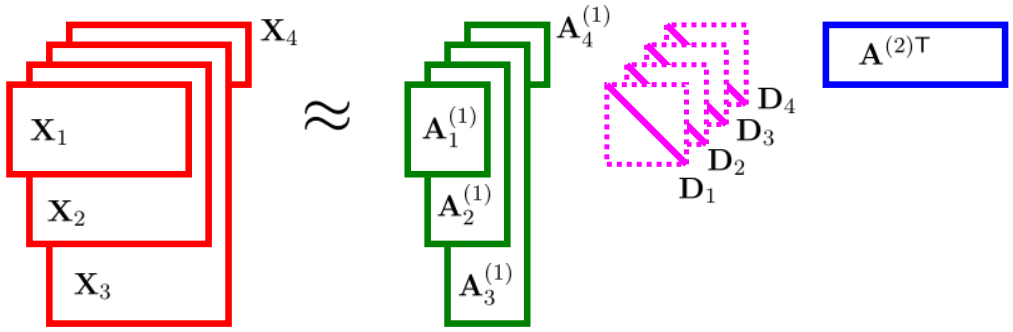


FIGURE 2.9 – Illustration of PARAFAC2 of a third-order tensor.

of positions. Concretely, s-PARAFAC for mode 2 looks something like

$$\mathbf{X}_i = \mathbf{A}^{(1)}\mathbf{D}_i S_{\mathbf{s}_i}(\mathbf{A}^{(2)})^{\mathsf{T}} + \mathbf{E}_i \quad \text{for} \quad i = 1, ..., I_3, \tag{2.41}$$

where almost all the notations are identical to that of (2.36) except that shift function $S_{\mathbf{s}_i}$ shifts all the columns of factor matrix $\mathbf{A}^{(2)}$ by some amount according to vector $\mathbf{s}_i \in \mathbb{R}^R$. The R elements in vector $\mathbf{s}_i$ are specified the ith row of shift matrix $\mathbf{S} \in \mathbb{R}^{I_3 \times R}$.

Generalizing convolutive nonnegative matrix factorization (CNMF) [Smaragdis, 2004] to high-order tensor, convolutive PARAFAC (c-PARAFAC) [Mørup and Schmidt, 2006] was proposed to model convolutive mixtures of multichannel time-frequency spectral data. Mathematically, the matrix representation is formulated in terms of frontal slice $\mathbf{X}_i$ as

$$\mathbf{X}_i = \mathbf{A}^{(1)} \sum_{\theta=0}^{\Theta-1} \mathbf{D}_i^{(\theta+1)} (\mathbf{S}_{\uparrow(\theta)}\mathbf{A}^{(2)})^{\mathsf{T}} + \mathbf{E}_i \quad \text{for} \quad i = 1, ..., I_3, \tag{2.42}$$

where $\mathbf{D}_i^{(\theta)}$ is a diagonal matrix whose entries are taken from the ith row of the θth factor matrix $\mathbf{A}_\theta$ in a set of factor matrices $\{\mathbf{A}_\theta^{(3)}\}_{\theta=1}^{\Theta}$ associated with the third mode. $\mathbf{S}_{\uparrow(\theta)}$ is denoted as vertical shift operator that enables us to shift matrix θ rows in the up direction, leaving the new rows shifted into the matrix from the bottom being zero.

2.2.2 Tucker decomposition

Another tensor decomposition in widespread use is called *Tucker decomposition* [Tucker, 1963; De Lathauwer et al., 2000a], which was introduced by relaxing the constraint in CP model to allow the arbitrary interaction of factors among different modes. The standard Tucker decomposition can be viewed as high-order principal component analysis. It converts high-order array $\mathcal{X}$ into a *core tensor* $\mathcal{G}$ that is transformed by an orthogonal *factor matrix* $\mathbf{A}^{(d)}$ along each mode d in the following way

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_D \mathbf{A}^{(D)} + \varepsilon \tag{2.43}$$

or equivalently in the outer product representation

$$\mathcal{X} = \sum_{j_1=1}^{R_1} \sum_{j_2=1}^{R_2} \cdots \sum_{j_d=1}^{R_D} g_{i_1,i_2,\ldots,i_D} \mathbf{a}_{j_1} \circ \mathbf{a}_{j_2} \circ \cdots \circ \mathbf{a}_{j_D} + \varepsilon \tag{2.44}$$

or in an equivalent scalar representation

$$x_{i_1,i_2,\ldots,i_D} = \sum_{j_1=1}^{R_1} \sum_{j_2=1}^{R_2} \cdots \sum_{j_d=1}^{R_D} g_{i_1,i_2,\ldots,i_D} a_{i_1,j_1} a_{i_2,j_2} \cdots a_{i_D,j_D} + \varepsilon_{i_1,i_2,\ldots,i_D}, \tag{2.45}$$

where $\mathbf{A}^{(d)} \in \mathbb{R}^{I_d \times R_d}$ is the factor matrix. R_d serves as the d-rank of $\mathcal{X}$ in mode d. $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_D}$ represents the core tensor indicating the interaction level among factor matrices from different modes. Thus, this core tensor $\mathcal{G}$ provides us with a much better way to capture the underlying multiway structure of tensor data. ε reflects the residuals.

A Tucker decomposition of a third-order tensor is given in Figure 2.10. When applying matricization in d-mode, Tucker reads

$$\mathbf{X}_{(d)} = \mathbf{A}^{(d)} \mathbf{G}_{(d)} (\mathbf{A}^{(D)} \otimes \cdots \otimes \mathbf{A}^{(d+1)} \otimes \mathbf{A}^{(d-1)} \otimes \cdots \otimes \mathbf{A}^{(1)})^{\mathsf{T}}, \tag{2.46}$$

and in the vectorization form, it becomes

$$vec(\mathcal{X}) = (\mathbf{A}^{(D)} \otimes \cdots \otimes \mathbf{A}^{(2)} \otimes \mathbf{A}^{(1)})\, vec(\mathcal{G}). \tag{2.47}$$

Analogous to CP model, the Tucker model can also be concisely expressed as [Kolda and Bader, 2009]

$$\mathcal{X} \approx [\mathcal{G}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(D)}]. \tag{2.48}$$

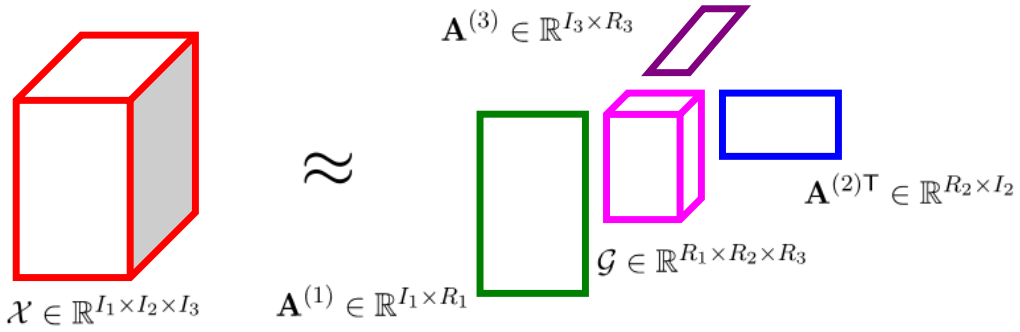


FIGURE 2.10 – Illustration of Tucker decomposition of a third-order tensor.

Hence, the total number of parameters for Tucker decomposition amounts to $\mathcal{O}(DIR + R^D)$ for $R = max\{R_d\}_{d=1}^{D}$, which is larger than that of CP but substantially smaller than $\mathcal{O}(I^D)$ of original tensor.

Essentially, CP model can be viewed as a special case of Tucker model in which the CP format can be reformulated into Tucker format with the core tensor being a superdiagonal core tensor, e.g, in following form through tensor matrix multiplication

$$\mathcal{X} = \Lambda \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_D \mathbf{A}^{(D)} + \varepsilon, \tag{2.49}$$

where $\Lambda \in \mathbb{R}^{R \times \cdots \times R}$ denotes the superdiagonal core tensor or super-identity tensor. In this formulation, a superdiagonal core tensor clearly indicates that the rth vector of factor matrix from one specific mode can only interact with the rth vector of factor matrix from another mode. Consequently, the nonzero entry in superdiagonal core tensor is nothing but a vector of coefficients that is used in weighting linear combination of rank-one tensors. This fact is clearly depicted in the bottom part of Figure 2.8.

The difference between the structure of core tensors suggests that Tucker is a much flexible model over CP, as the full dense core tensor enables us to explore more complicated correlation among factors from different modes. However, such flexibility gives rise to the rotational indeterminacy in Tucker [Kolda and Bader, 2009], e.g., the uniqueness of factor matrix is lost. To explain this rotational indeterminacy, suppose $\{\mathbf{B}^{(d}\}_{d=1}^{D}$ is a set of orthogonal rotators associated with corresponding mode, we are able to get the same model fit by multiplying the core tensor and factor matrix along d-mode with the rotator and its inverse, respectively. This is given by

$$\mathcal{X} = (\mathcal{G} \times_1 \mathbf{B}^{(1)\mathsf{T}} \times_2 \mathbf{B}^{(2)\mathsf{T}} \times_3 \cdots \times_D \mathbf{B}^{(D)\mathsf{T}}) \times_1 \mathbf{A}^{(1)}\mathbf{B}^{(1)} \times_2 \mathbf{A}^{(2)}\mathbf{B}^{(2)} \times_3 \cdots \times_D \mathbf{A}^{(D)}\mathbf{B}^{(D)} + \varepsilon. \tag{2.50}$$

As an another generalization of SVD to high-order tensor, Tucker decomposition can be computed using higher-order singular value decomposition (HOSVD) algorithm [De Lathauwer et al., 2000a] where the core tensor is all-orthogonal and factor matrices are orthonormal.

Specifically, in HOSVD the factor matrix $\mathbf{A}^{(d)}$ consisting of d-mode singular vectors can be directly found by the left singular vectors of d-mode matricization $\mathbf{Y}_{(d)}$ with singular value decomposition [Golub and Van Loan, 2012] (see Appendix A.3)

$$\mathbf{X}_{(d)} = \mathbf{A}^{(d)} \Sigma^{(d)} \mathbf{B}^{(d)\mathsf{T}}. \tag{2.51}$$

Given the factor matrix formed by d-mode singular vectors, the core tensor $\mathcal{G}$ can be computed according to

$$\mathcal{G} = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \times_2 \mathbf{A}^{(2)\mathsf{T}} \times_3 \cdots \times_D \mathbf{A}^{(D)\mathsf{T}}. \tag{2.52}$$

The HOSVD procedure is outlined in Algorithm 2, where line 4 and line 6 correspond to operation (2.51) and equation (2.52), respectively.

It is well known for matrices that the best rank-R approximation in least squares sense can be readily obtained from the truncated SVD. As for general higher-order tensors, the truncated

Algorithm 2 HOSVD [De Lathauwer et al., 2000a]

1: **Input**: D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, multilinear rank $(R_1, R_2, ..., R_D)$
2: **Output**: core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_D}$, factor matrices $\{\mathbf{A}^{(d)} \in \mathbb{R}^{I_d \times R_d}\}_{d=1}^{D}$
3: **for** $d = 1, ..., D$ **do**
4: $\quad \mathbf{A}^{(d)} \leftarrow R_d$ left singular vectors of $\mathbf{X}_{(d)}$
5: **end for**
6: $\mathcal{G} = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \times_2 \mathbf{A}^{(2)\mathsf{T}} \times_3 \cdots \times_D \mathbf{A}^{(D)\mathsf{T}}$

HOSVD does not provide the best rank-$(R_1, R_2, ..., R_D)$ approximation, only suboptimal solution is delivered [De Lathauwer et al., 2000b]. It is shown by Kolda [2003] that the factors in best rank-$(R_1+1, R_2+1, ..., R_D+1)$ approximation of a tensor does not necessarily contain the factors in its best rank-$(R_1, R_2, ..., R_D)$ approximation of that tensor. In contrast to HOSVD, when applying regular SVD to a matrix, the factors in best rank-$(R+1)$ approximation always contains the factors in its best rank-R approximation. Nevertheless, with good approximation guarantees, truncated HOSVD usually performs well in practice.

In order to improve the approximating ability of HOSVD, an iterative alternating least squares (ALS) based algorithm, known as higher-order orthogonal iteration (HOOI) [De Lathauwer et al., 2000b,a; Kroonenberg and De Leeuw, 1980], was proposed with the aim to minimize the approximation error in least squares, which is formulated as follows

$$\min \|\mathcal{X} - \bar{\mathcal{X}}\| = \min \|\mathcal{X} - \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_D \mathbf{A}^{(D)}\|. \tag{2.53}$$

Specially, HOOI usually uses HOSVD as a initialization, and the factor matrix $\mathbf{A}^{(d)}$ is estimated one at a time while fixing the rest factor matrices; this procedure iterates until convergence. Although being an iterative algorithm, HOOI is pretty efficient in the sense that it only needs to calculate the R $(R \ll I)$ left d-mode singular vectors in a dominant subspace. For instance, at each iteration, given other fixed factor matrices, we first project $\mathcal{X}$ onto a much smaller subspace to get $\mathcal{Y}^d$

$$\mathcal{Y}^d = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \cdots \mathbf{A}^{(d-1)\mathsf{T}} \times_{d-1} \mathbf{A}^{(d+1)\mathsf{T}} \times_{d+1} \cdots \times_D \mathbf{A}^{(D)\mathsf{T}}. \tag{2.54}$$

The procedure is summarized in Algorithm 3, where line 6 performs these projection operations of (2.54) for each mode before R left singular vectors from the d-mode are extracted.

One important extension of Tucker model, called block component decomposition (BCD) [De Lathauwer, 2008a,b; De Lathauwer and Nion, 2008], has been extensively investigated in depth in different approaches [Nion and De Lathauwer, 2008; Bro et al., 2009; Acar and Yener, 2009; Cichocki et al., 2009]. In general, BCD decomposes a tensor into the sum of subtensor components, each of which can be represented by a Tucker decomposition with same multilinear rank. Formally, BCD model with *rank*-$(R_1, R_2, ..., R_D)$ can be described as follows

Algorithm 3 HOOI [De Lathauwer et al., 2000b,a]

Input: D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, multilinear rank $(R_1, R_2, ..., R_D)$
Output: core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_D}$, factor matrices $\{\mathbf{A}^{(d)} \in \mathbb{R}^{I_d \times R_d}\}_{d=1}^{D}$
Initialize: initialize $\{\mathbf{A}^{(d)}\}_{d=1}^{D}$ using HOSVD
repeat
  for $d = 1, ..., D$ **do**
    $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \cdots \mathbf{A}^{(d-1)\mathsf{T}} \times_{d-1} \mathbf{A}^{(d+1)\mathsf{T}} \times_{d+1} \cdots \times_D \mathbf{A}^{(D)\mathsf{T}}$
    $\mathbf{A}^{(d)} \leftarrow R_d$ left singular vectors of $\mathbf{Y}_{(d)}$
  end for
until the convergence criterion is satisfied
$\mathcal{G} = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \times_2 \mathbf{A}^{(2)\mathsf{T}} \times_3 \cdots \times_D \mathbf{A}^{(D)\mathsf{T}}$

$$\mathcal{X} = \sum_{r=1}^{R} \mathcal{G}_r \times_1 \mathbf{A}_r^{(1)} \times_2 \mathbf{A}_r^{(2)} \times_3 \cdots \times_D \mathbf{A}_r^{(D)} + \varepsilon, \tag{2.55}$$

where $\mathcal{G}_r \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_D}$ and $\{\mathbf{A}_r^{(d)} \in \mathbb{R}^{I_d \times R_d}\}_{d=1}^{D}$, corresponding to rth sub-tensor component, are core tensor and factor matrices, respectively. An example of BCD for a third-order tensor is shown in Figure 2.11, where each of R sub-tensor components admits a Tucker decomposition. These Tucker decompositions corresponding to sub-tensor components have the same structures with respect to the multilinear ranks.

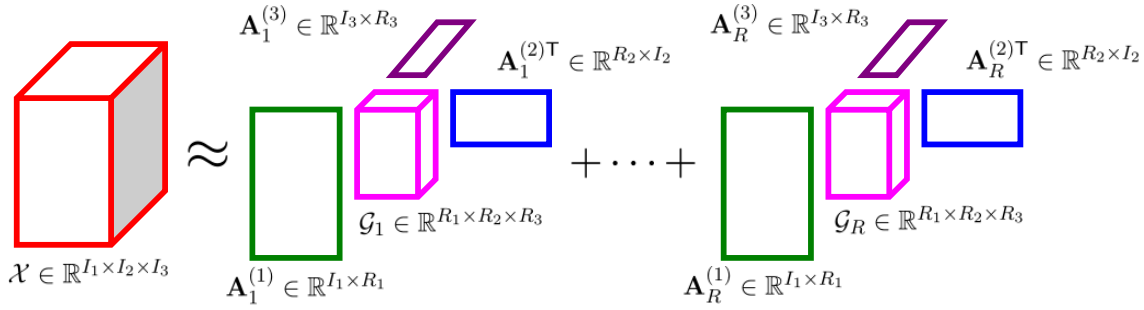


FIGURE 2.11 – Illustration of block component decomposition of a third-order tensor.

2.3 Scaling up Tensor Decompositions

In order to apply tensor decomposition to the big data applications, a number of approaches have been proposed recently to tackle the issues tensor decomposition with respect to scalability and efficiency. In general, these approaches attempt to find solutions to the scalability problem from the following four main aspects : (1) compression : rather than decomposing the full data, one decomposes a compressed representation of the original data ; (2) sparsity : by taking advantage of useful tools such as sparse matrix multiplication, the scalability can be

realized for big sparse tensor in many applications; (3) sampling: an approximate strategy to significantly reduce the computational cost while maintaining good accuracy, which is usually accomplished by drawing a few entries or subtensors from original tensor data; (4) parallel/distributed computing: the scalability can also be achieved by exploiting parallelization algorithms or distributed computation environments (e.g., Hadoop/MapReduce). (5) tensor networks (TNs) [Cichocki, 2014; Cichocki et al., 2016, 2017; Cichocki, 2018]: the use of tensor networks models is another promising way of scaling up tensor decompositions to big volume data. The concept of tensor networks is more generic which unifies tensor decompositions as its special cases (simple tensor networks or sub-networks). In contrast to conventional tensor decompositions, tensor networks decompose higher-order tensors into a set of sparsely interconnected small-scale low-order core tensors. In this manner, large-scale higher-order data can be approximately represented in highly compressed and distributed formats, resulting in both enhanced interpretation and computational advantage.

In what follows, we give a brief overview of some scalable algorithms for CP and Tucker decompositions. For more details, we refer readers to the paper [Papalexakis et al., 2016].

2.3.1 Scaling up CP Decomposition

The very early scalable algorithm through compression traces back to [Bro and Andersson, 1998]. The algorithm first computes Tucker decomposition of the original tensor to produce core tensor and factor matrices, which follows by a CP decomposition on the core tensor to obtain compressed factor matrices. The resulting compressed factors are then projected back to original subspaces using the Tucker factor matrices. It has been shown that the compressed factor matrices are capable of preserving the variation of the data. In another work based on compression strategy, Sidiropoulos et al. [2014] employs random projection matrices to create multiple compressed tensors that are decomposed in a parallel way, which follows by solving a least squares problem to find the true factors.

Then, some algorithms developed for the scalability of CP by exploiting sparse tensors and its related operations. In particular, [Kolda et al., 2005] and [Araujo et al., 2014] find multiple rank-one components using a higher-order power method and a deflation strategy. Combining the strategies of sparsity and distributed computing, a number of works [Kang et al., 2012; Choi and Vishwanathan, 2014; Ravindran et al., 2014; Smith et al., 2015] have been dedicated to scale up the CP-ALS algorithm by improving the efficiency of highly cost operation called matricized tensor times Khatri-Rao product (MTTKRP) [Papalexakis et al., 2016], which is also known as "intermediate data explosion" problem [Kang et al., 2012].

To avoid the "intermediate data explosion", these approaches first divide MTTKRP into smaller block-based operations such that multiplications can be carried out efficiently on sparse tensors. Meanwhile, block-based operations can be performed in parallel on a distributed architec-

ture such as MapReduce framework, significantly scaling up the decomposition. By unifying the ideas of sparsity, sampling and parallelization together, a few algorithms [Papalexakis et al., 2012; Erdos and Miettinen, 2013] sample subtensors from full tensor so as to reduce the dimensionality of the data and parallelize the decomposition. Basically, these approaches first select dense blocks within a tensor according to some randomization technique and then decompose these blocks separately. Finally, the partial factors of different blocks are merged to produce the resulting factors corresponding to the full tensor.

In addition to aforementioned ALS-based algorithms, several other types of algorithms have been proposed to scale up CP using parallelization computations. Among them, Beutel et al. [2014] presents a distributed MapReduce algorithm for CP by first dividing tensor into disjoint blocks and then applying stochastic gradient descent (SGD) to extract factors corresponding to these blocks. The flexible SGD framework allows one to adapt different types of loss functions or regularizations. Shin and Kang [2014] introduce another distributed MapReduce algorithm, based on coordinate descent (CD), to update individual coefficients of factors in column-wise manner, which is shown to be more beneficial in terms of memory gains.

2.3.2 Scaling up Tucker Decomposition

To tackle the "intermediate data explosion" problem in Tucker model, Kolda and Sun [2008] take advantage of efficient multiplication operations for sparse tensor to perform Tucker decomposition on very large-scale tensors. Tsourakakis [2010] applies sparsification of tensor for Tucker by randomly sampling nonzero elements of the tensor and scaling appropriately. As for the parallel computation, Jeon et al. [2015] scales both Tucker and CP by finding a decoupled implementation of the D-mode product, which can be easily adapted to the MapReduce framework.

2.4 Conclusion

In this chapter, we reviewed different aspects concerning high-order tensor including the basic concepts of tensor algebra and its decomposition. For the tensor basics, we introduced all the essential notations and definitions about tensor and their corresponding tensorial operations. As for the tensor decomposition formats, we mainly focused on the most widely used CP model and Tucker model, each of which links to several equivalent formulations that could be useful for different settings. We also discussed the properties associated with each model as well as their pros and cons. At last, we described how to scale up CP and Tucker decompositions to deal with large-scale tensorial data. All the content of this chapter lays a foundation for a better comprehension of tensor regression models and algorithms throughout this thesis.

The next chapter will give a brief overview of the most commonly used tensor regression methods that are developed based on the preliminaries of this chapter.

Chapitre 3

Tensor Regression Overview

Over the past few years, a variety of tensor regression approaches have been proposed to address the challenges arising from multivariate regression tasks. In this chapter, these tensor regression approaches are briefly reviewed from two primary perspectives: *linear tensor regression and nonlinear tensor regression modeling.*

3.1 Tensor Regression

Recall from the example introduced in Chapter 1 that a primary goal of MRI diagnosis task is to build the link between the brain images, usually being the form of higher-order tensors, and the clinical results. As we mentioned, such kind of task can naturally be reformulated as a tensor regression task with tensorial image as input and clinical outcome as output.

More formally, the regression task is to recover a function $f : \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D} \to \mathbb{R}$ from a collection of input-output data pairs $\{(\mathcal{X}_n, \mathcal{Y}_n)\}_{n=1}^{N}$ generated from the model

$$\mathcal{Y} = f(\mathcal{X}) + \varepsilon, \tag{3.1}$$

where the input $\mathcal{X}$ has a tensor structure and the output $\mathcal{Y}$ could be a tensor of any order. ε is also a tensor representing the error.

Depending on whether the function f is linear or not, the tensor regression approach can typically be categorized into linear tensor regression and nonlinear tensor regression. In the sections that follow, tensor regression methods are briefly reviewed from these two aspects.

3.1.1 Linear Tensor Regression

The standard multivariate linear regression model assumes $f(\mathcal{X})$ in (3.1) to be a linear function that often takes the following form via a vectorization operation of tensorial input $\mathcal{X}$ and tensorial coefficient $\mathcal{B}$

$$y = \langle vec(\mathcal{X}), vec(\mathcal{B}) \rangle + \varepsilon. \tag{3.2}$$

Modeling in terms of (3.2), the task can be readily solved by classical regression techniques. However, the gigantic dimensionality and complex multiway structure of the input $\mathcal{X}$ cause the main difficulty regarding this type of regression problem. In particular, naively vectorizing an ultrahigh dimensional data array $\mathcal{X}$ into a long vector $vec(\mathcal{X})$ will produce huge amount of parameters. For example, if we take a $3D$ MRI image of size $128 \times 128 \times 128$ as the input, the number of parameters resulting from vetorization operation in (3.2) will be as large as 2 millions. In this case, computation becomes intractable and theoretical guarantees are severely compromised when applying the classical regression approach. In addition, simply vectorizing a tensorial input destroys the spatial structure of the image tensor that contains a wealth of useful information.

In order to cut down huge parameters, one typical solution first carries out a dimension reduction step, often by principal components analysis (PCA) [Jolliffe, 2002], then fits the multivariate regression model with the reduced top principal components [Caffo et al., 2010]. However, since PCA is an unsupervised dimension reduction technique, the extracted principal components may have nothing to do with the output, the loss of information is inevitable in regression step.

CP Tensor Regression

Lately, Zhou et al. [2013] extended the generalized linear model (GLM) [Nelder and Baker, 1972] to treat higher-order tensor observation as input in linear regression model. Motivated by the low-rank idea using tensor decomposition tools, the authors combine GLM and CP decomposition to obtain the linear CP tensor regression model, which is free of the issues mentioned above. Specifically, Zhou and his colleagues assume a low-rank structure for tensor regression coefficient, and the linear systematic part of the model reads

$$\begin{aligned} y &= \langle \mathcal{X}, \mathcal{B} \rangle + b \\ &= \langle \mathcal{X}, \sum_{r=1}^{R} \mathbf{b}_r^{(1)} \circ \cdots \circ \mathbf{b}_r^{(D)} \rangle + b, \end{aligned} \tag{3.3}$$

where y is a scalar output and $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is a D-order tensor input. The coefficient tensor $\mathcal{B}$, measuring the effects of tensor input $\mathcal{X}$, is assumed to follow a rank-R CP decomposition $[\mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_D]$ with $\mathbf{B}_d = [\mathbf{b}_1^{(d)}, ..., \mathbf{b}_R^{(d)}] \in \mathbb{R}^{I_d \times R}$. b is the intercept constant.

Notice that by imposing CP format on the coefficient tensor $\mathcal{B}$, the dimensionality of parameters substantially decreases from $\mathcal{O}(I^D)$ to the scale of $\mathcal{O}(DIR)$, which is manageable and results in an efficient estimation. Although with such a massive dimensionality reduction, Zhou et al. [2013] demonstrate that this low-rank model could provide a reasonable recovery of even high rank signals.

Tucker Tensor Regression

The key difference between Tucker and CP formats suggests that Tucker is a more flexible model, as it allows to have the distinct number of basis vectors along each mode. Making use of this flexibility, Li et al. [2013] assumed that the coefficient tensor $\mathcal{B}$ admits a Tucker decomposition [Tucker, 1963; De Lathauwer et al., 2000a] instead of a CP decomposition. By approximating the coefficient tensor $\mathcal{B}$ with Tucker format

$$\mathcal{B} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_D=1}^{R_D} g_{r_1,\ldots,r_D} \mathbf{b}_{r_1}^{(1)} \circ \cdots \circ \mathbf{b}_{r_D}^{(D)}, \tag{3.4}$$

the linear part of Tucker tensor regression model becomes

$$\begin{aligned} y &= \langle \mathcal{X}, \mathcal{B} \rangle + b, \\ &= \langle \mathcal{X}, \sum_{r_1=1}^{R_1} \cdots \sum_{r_D=1}^{R_D} g_{r_1,\ldots,r_D} \mathbf{b}_{r_1}^{(1)} \circ \cdots \circ \mathbf{b}_{r_D}^{(D)} \rangle + b, \end{aligned} \tag{3.5}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$ is the core tensor with entries $\{g_{r_1,\ldots,r_D}\}_{r_1=1,\ldots,r_D=1}^{R_1,\ldots,R_D}$ capturing the interactions of factor matrices $\mathbf{B}_d \in \mathbb{R}^{I_d \times R_d}$ among different modes.

Now that since the authors adapt Tucker format to the coefficient tensor $\mathcal{B}$, then the total number of parameters for Tucker regression model turns out to be $\mathcal{O}(DIR + R^D)$, which is more than $\mathcal{O}(DIR)$ of CP regression but significantly less than that of $\mathcal{O}(I^D)$ if $R \ll I$.

The resulting Tucker regression model, as referred by (3.5), offers several benefits over CP regression model in the context of neuroimaging data analysis. First, Tucker produces a more parsimonious modeling, meaning a more compact model given limited sample size. For example, a $3D$ signal $\mathcal{B} \in \mathbb{R}^{16 \times 16 \times 16}$ admits a Tucker model with multilinear rank-$(R_1, R_2, R_3) = (2, 2, 5)$, and the number of parameters in this case is 131. On the contrary, $\mathcal{B}$ could only be modeled using a 5-component CP format, which yields 230 parameters. Second, Tucker model allows to choose different rank along each mode, which is especially useful when data is skewed in dimensions. Third, Tucker explicitly models the interaction between all factors, allowing for a finer grid search over a larger modeling space. All above benefits indicate the *flexibility* of Tucker regression model over CP regression model. Indeed, being a special case of Tucker format, the superdiagonal-style core tensor in CP significantly restricts the interactions of factors among different modes, leading to a model with inadequate predictability.

Both CP and Tucker tensor regression models are solved by alternating least square (ALS) [Kroonenberg and De Leeuw, 1980] algorithm that sequentially estimate one factor matrix at a time while keeping others fixed.

High-order Partial Least Squares Regression (HOPLS)

Zhao et al. [2013a] investigated a more general tensor regression model where the response is also a tensor consisting of multiple correlated outputs rather than a scalar output. The

circumstances of tensorial output emerge naturally in many real-world applications, such as estimation of $3D$ human pose positions from videos [Guo et al., 2012] and reconstruction of $3D$ limb trajectories from ECoG brain signals [Chao et al., 2010]. In this paper, Zhao et al. [2013a] generalize matrix PLS [Wold et al., 1984; Abdi, 2010] to high-order partial least squares (HOPLS) to handle tensor-output situation.

Generally speaking, HOPLS is a generalized multilinear tensor regression model whose objective is to predict an output tensor $\mathcal{Y}$ from an input tensor $\mathcal{X}$ through the extraction of a small number of common latent variables followed by a regression step against them [Zhao et al., 2013a]. The principle behind HOPLS is to sequentially conduct a set of joint block Tucker decompositions of $\mathcal{X}$ and $\mathcal{Y}$ with constraint that the extracted latent variables capture the maximum covariance between $\mathcal{X}$ and $\mathcal{Y}$.

Specifically, suppose we have a $(M+1)$th-order tensor $\mathcal{X} \in \mathbb{R}^{N \times I_1 \times \cdots \times I_M}$ and a $(L+1)$th-order tensor $\mathcal{Y} \in \mathbb{R}^{N \times J_1 \times \cdots \times J_L}$, which can be obtained by concatenating N pairs of observations $\{(\mathcal{X}_n, \mathcal{Y}_n)\}_{n=1}^N$ that couple in the first mode with equal number of samples. HOPLS decomposes $\mathcal{X}$ into a sum of *rank*-$(1, H_1, ..., H_M)$ Tucker blocks and $\mathcal{Y}$ into a sum of *rank*-$(1, K_1, ..., K_L)$ Tucker blocks, respectively. The model reads

$$\begin{aligned} \mathcal{X} &= \sum_{r=1}^{R} \mathcal{G}_r^{\mathcal{X}} \times_1 \mathbf{t}_r \times_2 \mathbf{P}_r^{(1)} \times \cdots \times_{M+1} \mathbf{P}_r^{(M)} + \epsilon_{\mathcal{X}} \\ \mathcal{Y} &= \sum_{r=1}^{R} \mathcal{G}_r^{\mathcal{Y}} \times_1 \mathbf{t}_r \times_2 \mathbf{Q}_r^{(1)} \times \cdots \times_{L+1} \mathbf{Q}_r^{(L)} + \epsilon_{\mathcal{Y}}, \end{aligned} \tag{3.6}$$

where R denotes the number of latent vectors and $\mathbf{t}_r \in \mathbb{R}^N$ corresponds to the rth latent column vector. $\mathcal{G}_r^{\mathcal{X}} \in \mathbb{R}^{1 \times H_1 \times \cdots \times H_M}$ and $\mathcal{G}_r^{\mathcal{Y}} \in \mathbb{R}^{1 \times K_1 \times \cdots \times K_L}$ represent the rth core tensor. $\{\mathbf{P}_r^{(m)}\}_{m=1}^M \in \mathbb{R}^{I_m \times H_m}$ and $\{\mathbf{Q}_r^{(l)}\}_{l=1}^L \in \mathbb{R}^{J_l \times K_l}$ are the respective rth projection matrices. Equivalently, the formula (3.5) can be rewritten in a more compact form as

$$\begin{aligned} \mathcal{X} &= \mathcal{G}^{\mathcal{X}} \times_1 \mathbf{T} \times_2 \mathbf{P}^{(1)} \times \cdots \times_{M+1} \mathbf{P}^{(M)} + \epsilon_{\mathcal{X}} \\ \mathcal{Y} &= \mathcal{G}^{\mathcal{Y}} \times_1 \mathbf{T} \times_2 \mathbf{Q}^{(1)} \times \cdots \times_{L+1} \mathbf{Q}^{(L)} + \epsilon_{\mathcal{Y}}, \end{aligned} \tag{3.7}$$

where $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R]$ is latent matrix obtained by concatenating R latent vectors. The loading matrices are $\mathbf{P}^{(m)} = [\mathbf{P}_1^{(1)}, ..., \mathbf{P}_R^{(M)}]$ and $\mathbf{Q}^{(l)} = [\mathbf{Q}_1^{(1)}, ..., \mathbf{Q}_R^{(L)}]$, while $\mathcal{G}^{\mathcal{X}} = diag(\mathcal{G}_1^{\mathcal{X}}, ..., \mathcal{G}_R^{\mathcal{X}}) \in \mathbb{R}^{R \times RH_1 \times \cdots \times RH_M}$ and $\mathcal{G}^{\mathcal{Y}} = diag(\mathcal{G}_1^{\mathcal{Y}}, ..., \mathcal{G}_R^{\mathcal{Y}}) \in \mathbb{R}^{R \times RK_1 \times \cdots \times RK_L}$ are core tensors with super block diagonal structure. The framework of HOPLS is illustrated in Figure 3.1, where both $\mathcal{X}$ and $\mathcal{Y}$ are factorized into a sum of R sub-tensor blocks, each of which admits a Tucker format.

Under the model (3.5), HOPLS aims to search for a set of latent vectors $\{\mathbf{t}_r\}_{r=1}^R$ and loading matrices $\{\{\mathbf{P}_r^{(m)}\}_{m=1}^M, \{\mathbf{Q}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ for R subtensor blocks, since these factors are essential for the final prediction. To this end, the authors provide an efficient algorithm to extract these factors in a sequential way following a deflation procedure. This procedure works as follows : the set of factors, $\{\mathbf{t}_1, \{\mathbf{P}_1^{(m)}\}_{m=1}^M, \{\mathbf{Q}_1^{(l)}\}_{l=1}^L\}$ associated with the first subtensor

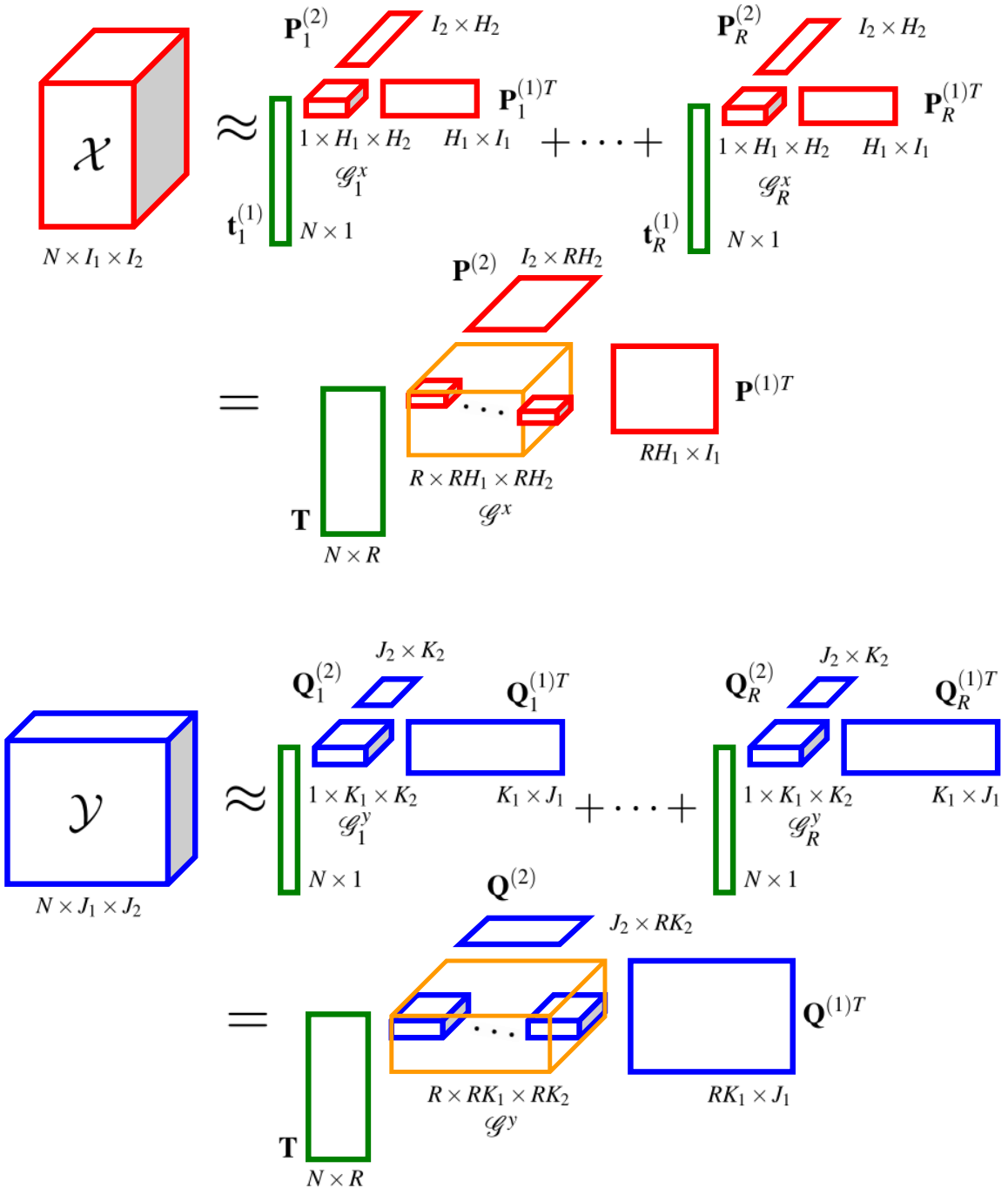


FIGURE 3.1 – An illustration of high-order partial least squares (HOPLS) for $M = 2$ and $L = 2$.

component, are extracted. Then, the explained variation via $\{\mathbf{t}_1, \{\mathbf{P}_1^{(m)}\}_{m=1}^M, \{\mathbf{Q}_1^{(l)}\}_{l=1}^L\}$ is subtracted from the variation of tensor pair yet to explain. Next, the same process is conducted for the second subtensor component. The procedure repeats until the extraction of the Rth subtensor component is accomplished.

Using the extraction of the factors corresponding to the first subtensor as demonstration, the first step begins with finding two sequences of loading matrices $\{\{\mathbf{P}_1^{(m)}\}_{m=1}^M, \{\mathbf{Q}_1^{(l)}\}_{l=1}^L\}$. The second step then estimates the latent vector $\mathbf{t}_1$.

In the first step, the extraction of loadings is accomplished by simultaneously minimizing the residuals $\|\epsilon_{\mathcal{X}}\|_{\mathsf{F}}$ and $\|\epsilon_{\mathcal{Y}}\|_{\mathsf{F}}$, which is equivalent to maximizing the core tensor $\|\mathcal{G}^{\mathcal{X}}\|_{\mathsf{F}}$ and $\|\mathcal{G}^{\mathcal{Y}}\|_{\mathsf{F}}$ jointly as below

$$\begin{aligned} & \max_{\mathcal{G}^{\mathcal{X}},\mathcal{G}^{\mathcal{Y}}} \|\mathcal{G}^{\mathcal{X}}\|_{\mathsf{F}}^2 \cdot \|\mathcal{G}^{\mathcal{Y}}\|_{\mathsf{F}}^2 \\ = & \max_{\mathcal{G}^{\mathcal{X}},\mathcal{G}^{\mathcal{Y}}} \|\langle \mathcal{G}^{\mathcal{X}}, \mathcal{G}^{\mathcal{Y}} \rangle_{\{1;1\}}\|_{\mathsf{F}}^2 \\ = & \max_{\{\mathbf{P}_1^{(m)},\mathbf{Q}_1^{(l)}\}} \|[\langle \mathcal{X}, \mathcal{Y} \rangle_{\{1;1\}}; \mathbf{P}_1^{(1)\mathsf{T}}, ..., \mathbf{P}_1^{(M)\mathsf{T}}, \mathbf{Q}_1^{(1)\mathsf{T}}, ..., \mathbf{Q}_1^{(L)\mathsf{T}}]\|_{\mathsf{F}}^2, \end{aligned} \tag{3.8}$$

here tensor contraction $\langle \mathcal{X}, \mathcal{Y} \rangle_{\{1;1\}}$ is defined as *1-mode cross-covariance tensor*

$$\mathcal{C} = COV_{\{1;1\}}(\mathcal{X}, \mathcal{Y}) \in \mathbb{R}^{I_1 \times \cdots \times I_M \times J_1 \times \cdots \times J_L}. \tag{3.9}$$

Finally, the optimization problem becomes

$$\begin{aligned} & \max_{\{\mathbf{P}_1^{(m)},\mathbf{Q}_1^{(l)}\}} \|[\mathcal{C}; \mathbf{P}_1^{(1)\mathsf{T}}, ..., \mathbf{P}_1^{(M)\mathsf{T}}, \mathbf{Q}_1^{(1)\mathsf{T}}, ..., \mathbf{Q}_1^{(L)\mathsf{T}}]\|_{\mathsf{F}}^2 \\ & \text{s.t. } \mathbf{P}_1^{(m)\mathsf{T}}\mathbf{P}_1^{(m)} = \mathbf{I}_{H_m} \ \forall m \ \text{ and } \ \mathbf{Q}_1^{(l)\mathsf{T}}\mathbf{Q}_1^{(l)} = \mathbf{I}_{K_l} \ \forall l, \end{aligned} \tag{3.10}$$

where $\mathbf{I}_{H_m}$ and $\mathbf{I}_{K_l}$ are identity matrices. Consequently, the loadings $\{\{\mathbf{P}_1^{(m)}\}_{m=1}^M, \{\mathbf{Q}_1^{(l)}\}_{l=1}^L\}$ can be estimated by Tucker decomposition of $\mathcal{C}$ using the higher order orthogonal iteration (HOOI) [De Lathauwer et al., 2000a], which is an efficient technique for computing the factor matrices of Tucker format, since it just computes an orthonormal basis of the dominant subspace.

In the second step, having known the loadings $\{\{\mathbf{P}_1^{(m)}\}_{m=1}^M, \{\mathbf{Q}_1^{(l)}\}_{l=1}^L\}$, the latent vector $\mathbf{t}_1$ can be determined by solving the following problem also via the HOOI algorithm

$$\min_{\mathbf{t}_1} \|\mathcal{X} - [\mathcal{G}^{\mathcal{X}}; \mathbf{t}_1, \mathbf{P}_1^{(1)}, ..., \mathbf{P}_1^{(M)}]\|_{\mathsf{F}}^2. \tag{3.11}$$

According to the deflation strategy described earlier, only one set of these parameters is extracted at a time until R sets are obtained or certain accuracy of approximating both $\mathcal{X}$ and $\mathcal{Y}$ is achieved.

Finally, the prediction can be made based on the core tensors $\{\mathcal{G}_r^{\mathcal{X}}, \mathcal{G}_r^{\mathcal{Y}}\}_{r=1}^R$ as well as the loading matrices $\{\{\mathbf{P}_r^{(m)}\}_{m=1}^M, \{\mathbf{Q}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ [Zhao et al., 2013a].

In contrast to unfolded PLS [Abdi, 2010], HOPLS has the advantage in terms of interpretation. The loadings $\{\{\mathbf{P}_r^{(m)}\}_{m=1}^M, \{\mathbf{Q}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$, e.g., like those extracted in the first step, can be easily interpreted as new subspace signatures of d-mode features, while unfolded PLS

has difficulty in interpreting the high dimensional loading matrices, because the unfolding operation of tensor destroys the multiway structure of the data.

N-way partial least squares (NPLS) [Bro, 1996] is the tensor regression model that jointly decomposes tensorial input and output into sum of rank-one tensors using CP decomposition, subject to that the found latent variables have the maximum pairwise covariance. As for the extraction of factors, NPLS algorithm is developed on the basis of CP-ALS algorithm [Carroll and Chang, 1970; Harshman, 1970]. Compared to HOPLS, NPLS is more robust to noise due to the sum of rank-one tensor structure, but it is limited regarding to good fitness. In contrast, the flexibility of HOPLS allows to balance the fitting and the significance of latent vectors in terms of model complexity, which implies a good generalization to new data. Furthermore, HOPLS offers a more efficient solution by boiling down to solving a singular value decomposition (SVD) problem at each mode, meanwhile NPLS relies on a much slower iterative ALS-based algorithm.

Extending NPLS to the online setting, the recursive N-way partial least squares (RNPLS) [Eliseyev and Aksenova, 2013] processes the tensor sequences by unifying the recursive calculation scheme of recursive partial least squares (RPLS) [Qin, 1998] with the multiway data representation of NPLS. Inheriting the drawbacks of NPLS, the RNPLS likewise suffers from the lack of adequate accuracy and the slow convergence rate because its solution is based on combining an NIPALS-style algorithm [Wold, 1975a] with CP decomposition. Thus, the speed is rather slow especially when a relatively larger number of latent vectors are required for sufficient accuracy, which significantly reduces the applicability of RNPLS in time-critical applications.

Tensor Regression for Multilinear Multitask Learning (MLMTL)

In recent years, several tensor regression methods have been proposed in the context of multilinear multitask learning problems (MLMTL) [Romera-Paredes et al., 2013; Wimalawarne et al., 2014; Signoretto et al., 2014]. In these methods the tasks are referred by multiple indices, indicating the complex correlations among all the tasks. For instance, in restaurant recommendation system, MLMTL can be converted as a regression problem whose objective is to predict the ratings of I_2 aspects (e.g., food quality, service quality, overall quality, etc) from I_3 customers on the basis of I_1 features (e.g., cuisine type, price, location, etc). Hence, there are in total $I_2 \times I_3$ tasks that are indexed by a pair indices *aspect* $\times$ *customer*.

To tackle such multimodal correlations between tasks, Romera-Paredes et al. [2013] came up with a multilinear regression model with a low-rank regularized regression coefficient tensor.

Concretely, Romera-Paredes and their colleagues formulate MLMTL as a set of T linear regression tasks, each of which is associated with a coefficient vector $\mathbf{w}_t \in \mathbb{R}^{I_1}$, $t \in [T]$, leading to a concatenated coefficient matrix $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_T] \in \mathbb{R}^{I_1 \times I_2 I_3}$. Then, the resulting coefficient

matrix $\mathbf{W}$ is naturally refolded into a coefficient tensor $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ where each task t can be identified by a multi-index $(i_2, i_3) \in (I_2 \times I_3)$. Formally, the coefficient tensor $\mathcal{W}$ can be estimated by solving the following optimization problem

$$\min \mathcal{F}(\mathcal{W}) + \lambda \mathcal{R}(\mathcal{W}), \tag{3.12}$$

where term $\mathcal{F}(\mathcal{W})$ representing the convex empirical loss is defined as

$$\mathcal{F}(\mathcal{W}) = \sum_{t=1}^{T} \sum_{n=1}^{N_t} \mathcal{L}(\langle \mathbf{x}_n^t, \mathbf{w}_t \rangle, y_n^t), \tag{3.13}$$

whereas term $\mathcal{R}(\mathcal{W})$ corresponds to low multilinear rank penalty that encourages the dominant common latent structure between tasks

$$\mathcal{R}(\mathcal{W}) = \frac{1}{D} \sum_{d=1}^{D} rank_d(\mathcal{W}). \tag{3.14}$$

Here, λ is the tuning hyperparameter and N_t denotes the number of training samples $\{(\mathbf{x}_n^t, y_n^t)\}$ available for the task t. Finding the exact solution for (3.12) is extremely difficult due to the highly non-convex multilinear rank function. Instead, Romera-Paredes et al. [2013] adapt a convex surrogate, using trace norms of the matricizations, to the regularization term $\mathcal{R}(\mathcal{W})$, called *tensor overlapped trace norm* [Gandy et al., 2011], and it is defined as

$$\|\mathcal{W}\|_{\mathsf{tr}} = \frac{1}{D} \sum_{d=1}^{D} \|\mathbf{W}_{(d)}\|_{\mathsf{tr}}, \tag{3.15}$$

and the optimization problem finally becomes

$$\min \mathcal{F}(\mathcal{W}) + \lambda \|\mathcal{W}\|_{\mathsf{tr}}. \tag{3.16}$$

To solve this problem, they resort to an alternating direction method of multiplier technique (ADMM) [Boyd et al., 2011].

Although obtaining a global solution, this approach has a major drawback regarding the storage complexity, since the coefficient tensor $\mathcal{W}$ can be sufficiently large. Furthermore, the storage requirement of $N + 1$ versions of coefficient tensor $\mathcal{W}$ in memory when using ADMM severely limits the applicability of this method to large-scale datasets.

Adopting the similar low-rank idea based on convex relaxation via trace norm, Signoretto et al. [2014] presented a general version of ADMM using Douglas-Rachford splitting method [Eckstein and Bertsekas, 1992]. Likewise, their method also suffers from the issue of scalability to large-scale problems. In order to address this issue, Romera-Paredes et al. [2013] also proposed a non-convex approach by imposing Tucker decomposition on the coefficient tensor $\mathcal{W}$ and employing ALS to estimate the resulting factors. Now the regularized optimization problem becomes

$$\min_{\mathbf{G}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(D)}} \mathcal{F}(\mathbf{G} \times_1 \mathbf{A}^{(1)} \cdots \times_D \mathbf{A}^{(D)}) + \lambda (\|\mathbf{G}\|_{\mathsf{F}}^2 + \frac{1}{D} \sum_{d=1}^{D} \|\mathbf{A}^{(d)}\|_{\mathsf{F}}^2), \tag{3.17}$$

where $\mathbf{G}$ is the core tensor and $\mathbf{A}^{(d)}$ is the factor matrix.

Exploiting the low-rank structure of $\mathcal{W}$, this strategy substantially reduces the space complexity by eliminating the need for storing the whole coefficient tensor $\mathcal{W}$. Though without guarantee of global optimality, Romera-Paredes et al. [2013] demonstrate the non-convex solution outperforms the convex counterpart in practice. The computational cost, however, is rather expensive since a gradient descent procedure needs to be executed when estimating each factor in the outer loop.

Following the research line of convex relaxation in MLMTL, Wimalawarne et al. [2014] extended a new tensor norm, named *tensor scaled latent trace norm*, to overcome the drawback of the overlapped trace norm (3.15) in situation where the coefficient tensor $\mathcal{W}$ usually possesses heterogeneous dimensions or heterogeneous multilinear ranks. The proposed new norm reads

$$\|\mathcal{W}\|_{\mathsf{scaled}} = \inf_{\mathcal{W}^{(1)}+\cdots+\mathcal{W}^{(D)}=\mathcal{W}} \frac{1}{D}\sum_{d=1}^{D}\frac{1}{\sqrt{I_d}}\|\mathbf{W}^{(d)}_{(d)}\|_{\mathsf{tr}}, \tag{3.18}$$

where $\mathcal{W}^{(d)}$ is defined as the sub-tensor component corresponding to the d mode. For the overlapped trace norm represented by (3.15), it is unclear which mode could actually be low rank. In contrast, by analyzing the error bounds incurred by different norms, Wimalawarne et al. [2014] consistently show the superiority of the scaled latent norm in recognizing the mode with the lowest rank relative to its dimension, thus resulting in a better regularization.

Overall, the existing methods for MLMTL based on the convex relaxation techniques are computationally very expensive due to the requirement of full SVD operations of large matrices in alternating direction methods, rendering them infeasible for large-scale applications.

Tensor Regression for Spatio-Temporal Data Analysis (GREEDY)

To deal with the scalability issues of [Romera-Paredes et al., 2013; Wimalawarne et al., 2014], Bahadori et al. [2014] proposed a low rank learning framework that can be applied to large climate forecasting task in the context of multivariate spatio-temporal data analysis [Cressie and Huang, 1999; Cressie and Wikle, 2015]. Their framework takes advantage of low-rank tensor modeling by representing spatio-temporal data in tensor form to exploit the shared intrinsic dependency structures among the variables, locations and time.

In spatio-temporal forecasting task [Bahadori et al., 2014], the objective is to predict the future tensor values utilizing the historical records, where spatio-temporal observations can naturally be represented as tensor $\mathcal{X} \in \mathbb{R}^{P\times T\times M}$ of T timestamps of M variables over P locations. Adopting vector auto regressive (VAR) models of K lag, Bahadori et al. [2014] describe the relationship between the previous observations and the K-step ahead predictions by the following matrix representation

$$\mathcal{X}_{:,t,m} = \mathcal{W}_{:,:,m}\mathbf{x}_{t,m} + \varepsilon_{:,t,m} \quad m = 1, ..., M \quad \text{and} \quad t = K+1, ..., T, \tag{3.19}$$

where the colon symbol : is used to index the entire rows or columns. $\mathcal{W} \in \mathbb{R}^{P \times KP \times M}$ is the coefficient tensor to be learned. $\mathbf{x}_{t,m} = [\mathcal{X}^T_{:,t-1,m}, ..., \mathcal{X}^T_{:,t-K,m}]^T$ is obtained by stacking K-lag previous observations before time T. $\varepsilon_{:,t,m}$ corresponds to the error terms. To capture the correlations in data, the low rank constraint imposed on $\mathcal{W}$ encourages the commonalities along three dimensions while reducing the model complexity, leading to the following optimization problem

$$\begin{aligned} &\min_{\mathcal{W}} \|\hat{\mathcal{X}} - \mathcal{X}\|_{\mathsf{F}}^2 + \mu \sum_{m=1}^{M} tr(\hat{\mathcal{X}}^T_{:,:,m} \mathbf{L} \hat{\mathcal{X}}_{:,:,m}) \\ &\text{s.t.} \quad rank(\mathcal{W}) \leq R \quad \hat{\mathcal{X}}_{:,:,m} = \mathcal{W}_{:,:,m} \mathbf{x}_{t,m}, \end{aligned} \tag{3.20}$$

where μ is found to balance the data fitting term and regularization term. The Laplacian regularizer $\mathbf{L}$ [Cvetković et al., 1980; Babić et al., 2002] accounts for the spatial proximity of data using geometrical information.

Recognizing (3.20) as a non-convex problem, Bahadori and his colleagues obtain the optimal solution using a greedy algorithm, namely GREEDY, that sequentially estimates rank-1 subspace based on orthogonal matching pursuit (OMP) [Pati et al., 1993]. Concretely, at each iteration, the GREEDY algorithm first attempts to search for the best rank-one estimation of the matricization of coefficient tensor $\mathcal{W}$ along a specific mode. This specific mode is selected whenever the largest amount of decrease in loss function is achieved. Next, the found rank-one estimation, represented by a set of factors, is refolded back into tensor and added to current estimation of coefficient tensor $\hat{\mathcal{W}}$. In this way, the coefficient tensor $\hat{\mathcal{W}}$ accumulates using above rank-one approximation process at each iteration. Finally, the algorithm is completed by an projection of $\hat{\mathcal{W}}$ onto the subspace spanned by the top singular vectors of its 1-mode matricization. The orthogonal projection operation aims at eliminating the redundant rank-one components to maintain the lowest rank complexity.

Such greedy algorithm has been shown to provide better performance than convex-relaxation-based approaches in terms of speed and accuracy. Bahadori et al. [2014] also provides a theoretical error bound that measures the suboptimality to the globally optimal solution at each iteration. However, this algorithm is specially designed for spatio-temporal data which requires the spatial and temporal modes are at least shared in common in terms of tensor structure between the input and output tensors. In other words, the output tensor has the same tensorial structure (e.g., the order, the dimensionality) as input tensor.

Accelerated Low-rank Tensor Online Learning (ALTO)

To face the challenges of heavy computational load and short response time that arise in the setting of online spatio-temporal stream analysis, Yu et al. [2015] leveraged the low-rank tensor learning via Tucker decomposition to develop a fast algorithm called accelerated low-rank tensor online learning (ALTO). Similar to GREEDY [Bahadori et al., 2014], with the goal

of estimating low-rank constrained coefficient tensor, the framework for the spatio-temporal can be formulated as

$$\begin{aligned} \min_{\mathcal{W}} \sum_{t,m} \|\mathcal{W}_{:,t,m}\mathcal{Z}_{:,t,m} - \mathcal{X}_{:,t,m}\|_{\mathsf{F}}^2 \\ \text{s.t.} \quad rank(\mathcal{W}) \leq R, \end{aligned} \tag{3.21}$$

where $\mathcal{Z} \in \mathbb{R}^{KP\times T\times M}$, $\mathcal{X} \in \mathbb{R}^{P\times T\times M}$ and $\mathcal{W} \in \mathbb{R}^{P\times KP\times M}$ are associated with input, output and coefficient tensor, respectively. Likewise, T corresponds to the timestamps, M denotes the number of variables, P is the number of locations and K is number of lags. Note that t increases at each timestamp.

In brief, ALTO algorithm, which solves the spatio-temporal stream task represented by (3.21), can be summarized into two stages. Stage 1 aims to update the coefficient tensor $\mathcal{W}$ in an unconstrained manner. In Stage 2, the updated solution is then projected to low-rank tensor space to satisfy the specified constraint $rank(\mathcal{W}) \leq R$.

In the setting of online tensor learning, the spatio-temporal observations keep growing in the temporal mode over time [Sun et al., 2008]. Therefore, the best strategy is to update coefficient tensor $\mathcal{W}$ in an incremental fashion. To this end, Stage 1 recursively updates the coefficient tensor $\mathcal{W}$ each time that a new mini-batch of size b arrives at timestamp T, according to the following α-rule

$$\mathbf{W}^{(k)} = (1-\alpha)\mathbf{W}^{(k-1)} + \alpha\mathbf{X}_{T+1:T+b}\,\mathbf{Z}_{T+1:T+b}^{\dagger}. \tag{3.22}$$

Here, the new $\mathcal{W}^{(k)}$ can be expressed as a convex combination of the old $\mathcal{W}^{(k-1)}$ and a closed-form solution for the current mini-batch data. To project the unconstrained solution to low-rank tensor space, Stage 2 takes the low-rank representation from last iteration as a starting point, that is

$$\mathcal{W}^{(k-1)} = \mathcal{G}^{(k-1)} \times_1 \mathbf{U}_1^{(k-1)} \times_2 \mathbf{U}_2^{(k-1)} \times_3 \mathbf{U}_3^{(k-1)}, \tag{3.23}$$

which follows by an augmentation and orthogonalization process of factor $\mathbf{U}_d^{(k-1)} \in \mathbb{R}^{I_d\times R}$ to obtain factor $\mathbf{V}_d^{(k-1)} \in \mathbb{R}^{I_d\times(R+K)}$, by introducing K extra random columns. Then, the new $\mathcal{W}^{(k)}$ is transformed by the perturbed space spanned by the columns of $\mathbf{V}_d^{(k-1)}$, yielding the augmented core $\tilde{\mathcal{G}}^{(k)} \in \mathbb{R}^{(R+K)\times(R+K)\times(R+K)}$ as

$$\tilde{\mathcal{G}}^{(k)} = \mathcal{W}^{(k)} \times_1 \mathbf{V}_1^{(k-1)} \times_2 \mathbf{V}_2^{(k-1)} \times_3 \mathbf{V}_3^{(k-1)}. \tag{3.24}$$

The augmented core tensor $\tilde{\mathcal{G}}^{(k)}$ is thereafter compressed into original size of $R\times R\times R$ using Tucker decomposition

$$\tilde{\mathcal{G}}^{(k)} \approx \mathcal{G}^{(k)} \times_1 \tilde{\mathbf{V}}_1^{(k)} \times_2 \tilde{\mathbf{V}}_2^{(k)} \times_3 \tilde{\mathbf{V}}_3^{(k)}, \tag{3.25}$$

and the new factor matrix $\mathbf{U}_d^{(k)}$ is also reached by

$$\mathbf{U}_d^{(k)} = \mathbf{V}_d^{(k-1)}\tilde{\mathbf{V}}^{(k)}. \tag{3.26}$$

Eventually, the projected low-rank coefficient tensor $\mathcal{W}^{(k)}$ can readily be written as

$$\mathcal{W}^{(k)} = \mathcal{G}^{(k)} \times_1 \mathbf{U}_1^{(k)} \times_2 \mathbf{U}_2^{(k)} \times_3 \mathbf{U}_3^{(k)}. \tag{3.27}$$

To sum up, ALTO obtains a closed-form solution for the update of the coefficient tensor and efficiently carries out the low-rank projection by making use of the projected factors from previous iteration, without suffering from the substantial cost of SVD on unfolding of full size tensor. In addition, the incorporation of the random variation into projection significantly overcomes the local optima issue yet with good approximation accuracy guarantees [Yu et al., 2015]. Nevertheless, ALTO, as its batch counterpart GREEDY [Bahadori et al., 2014], is limited to spatio-temporal data with a specified format between input and output.

Tensor Regression via Tensor Projected Gradient (TPG)

For accelerating the speed and avoiding the memory bottleneck, an efficient batch tensor regression algorithm called tensor projected gradient (TPG) was recently introduced by Yu and Liu [2016], building upon the tensor generalization of iterative hard thresholding [Blumensath and Davies, 2009] and a randomized sampling technique. Following the similar low-rank modeling framework as GREEDY [Bahadori et al., 2014], the authors consider the following optimization problem in the context of both multi-linear multi-task learning (MLMTL) and spatio-temporal forecasting

$$\begin{aligned} &\min_{\mathcal{W}} \mathcal{L}(\mathcal{W}; \mathcal{X}, \mathcal{Y}) \\ &\text{s.t.} \quad rank(\mathcal{W}) \leq R, \end{aligned} \tag{3.28}$$

where $\mathcal{L}$ refers to the loss function parameterized by tensor coefficient $\mathcal{W}$. The relationship between the input $\mathcal{X}$ and output $\mathcal{Y}$ linked by $\mathcal{W}$ is modeled using linear regression model as

$$\mathcal{Y} = \langle \mathcal{X}, \mathcal{W} \rangle + \varepsilon, \tag{3.29}$$

here $\mathcal{X} \in \mathbb{R}^{T \times I_1 \times I_3}$ and $\mathcal{Y} \in \mathbb{R}^{T \times I_2 \times I_3}$, and T in the first mode of both $\mathcal{X}$ and $\mathcal{Y}$ represents the sample size. ε corresponds to the Gaussian error term. Yu and Liu [2016] define the tensor inner product in (3.29) as the matrix multiplication on each slice, i.e.,

$$\langle \mathcal{X}, \mathcal{W} \rangle = \sum_{m=1}^{I_3} \mathcal{X}_{:,:,m} \mathcal{W}_{:,:,m}. \tag{3.30}$$

To find a solution for (3.28), Yu and Liu [2016] apply the subsampled projected gradient descent on the regression coefficient tensor $\mathcal{W}$. Basically, each iteration of TPG consists of a gradient step followed by a proximal point projection step [Rockafellar, 1976]. The gradient step attempts to compute the ordinary gradient of loss function $\mathcal{L}$ w.r.t to $\mathcal{W}$ in an unconstrained way

$$\nabla \mathcal{L}(\mathcal{W}; \mathcal{X}, \mathcal{Y}) = \langle \mathcal{X}^{\mathsf{T}}, \mathcal{Y} - \langle \mathcal{X}, \mathcal{W} \rangle \rangle. \tag{3.31}$$

Subsequently, the proximal point projection step $\mathcal{P}_R(\mathcal{W})$ carries out an tensor projection (ITP) procedure at each iteration k to locate a nearest low-rank solution. In particular, ITP efficiently conducts power iterations to sequentially find all the singular vectors in each projection matrix, onto which the regression coefficient $\mathcal{W}$ can be projected.

$$\mathcal{P}_R(\mathcal{W}^k) = \arg\min_{\mathcal{W}} \|\mathcal{W}^k - \mathcal{W}\|_{\mathsf{F}}^2 \tag{3.32}$$

$$\text{s.t.} \quad rank(\mathcal{W}) \leq R, \tag{3.33}$$

For the purpose of acceleration, a randomized subsampling technique called count sketch [Clarkson and Woodruff, 2013] is leveraged as a preprocessing step to reduce the large sample complexity as well as data noise.

From the theoretical point of view, under the restricted isometry property (RIP) assumption [Candes et al., 2006], TPG is shown to converge to an approximate solution in a constant number of iterations, depending merely on the signal to noise ratio [Yu and Liu, 2016]. Furthermore, estimation error of TPG is proved to be linear to the norm of the observation error. Although being locally optimal, the authors argue that the obtained solution is robust since different local optima are highly concentrated for Tucker model [Ishteva et al., 2011].

In summary, the main contribution of [Yu and Liu, 2016] lies in the projection step where they adapt power method for computing singular vectors in the tensor case. Specifically, they use power iteration to efficiently determine dominant singular vector for each mode by leveraging the property of Tucker model. In this way, full SVD can thus be avoided on large coefficient tensor. However, like GREEDY [Bahadori et al., 2014], TPG is not designed as a general tensor regression model, it is modeled under restriction with some special data structure between the input and output. As being a simple linear model described in (3.29), the correlation in the tensorial output side remains not well exploited.

Higher Order Low Rank Regression (HOLRR)

More recently, Rabusseau and Kadri [2016] extend low rank regression (LRR) [Mukherjee and Zhu, 2011] to higher order low rank regression (HOLRR) to handle the case of tensorial output with high multiway dependencies. HOLRR aims to discover a multilinear function that maps a vector input $\mathbf{x} \in \mathbb{R}^{I_0}$ to tensor output $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ from a set of samples generated from the model

$$\mathcal{Y} = \mathcal{W} \bar{\times}_1 \mathbf{x} + \varepsilon, \tag{3.34}$$

where $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_D}$ is the regression coefficient tensor and ε is the error. The authors achieve this goal by formulating the task as minimizing a least squares subject to low multilinear rank constraint on $\mathcal{W}$

$$\begin{aligned} &\min_{\mathcal{W}} \|\mathcal{W} \times_1 \widetilde{\mathbf{X}} - \widetilde{\mathcal{Y}}\|_{\mathsf{F}}^2 \\ &\text{s.t.} \quad rank(\mathcal{W}) \leq R, \end{aligned} \tag{3.35}$$

which is shown to be equivalent to solving the following problem under Tucker format

$$\begin{aligned} &\min_{\mathcal{G},\mathbf{U}_0,\{\mathbf{U}_d\}} \|\mathcal{W} \times_1 \widetilde{\mathbf{X}} - \widetilde{\mathcal{Y}}\|_{\mathsf{F}}^2 \\ &\text{s.t.} \quad \mathcal{W} = \mathcal{G} \times_1 \mathbf{U}_0 \times_2 \cdots \times_{D+1} \mathbf{U}_D \quad \text{and} \quad \mathbf{U}_d^{\mathsf{T}}\mathbf{U}_d = \mathbf{I} \quad \forall d, \end{aligned} \tag{3.36}$$

here $\mathbf{U}_d \in \mathbb{R}^{I_d \times R_d}$ is the factor matrix for mode d. $\widetilde{\mathbf{X}} \in \mathbb{R}^{N \times I_0}$ and $\widetilde{\mathcal{Y}} \in \mathbb{R}^{N \times I_1 \times \cdots \times I_D}$ are obtained by stacking N pairs of observations. The above problem (3.36) can be further derived as

$$\begin{aligned} &\min_{\mathbf{U}_0,\{\mathbf{U}_d\}} \|\widetilde{\mathcal{Y}} \times_1 \mathbf{P}_0 \times_2 \cdots \times_{D+1} \mathbf{P}_D - \widetilde{\mathcal{Y}}\|_{\mathsf{F}}^2 \\ &\text{s.t.} \quad \mathbf{U}_d^{\mathsf{T}}\mathbf{U}_d = \mathbf{I} \quad \forall d \qquad , \end{aligned} \tag{3.37}$$

where $\mathbf{P}_0 = \widetilde{\mathbf{X}}\mathbf{U}_0(\mathbf{U}_0^{\mathsf{T}}\widetilde{\mathbf{X}}^{\mathsf{T}}\widetilde{\mathbf{X}}\mathbf{U}_0)^{-1}\mathbf{U}_0^{\mathsf{T}}\widetilde{\mathbf{X}}^{\mathsf{T}}$ and $\mathbf{P}_d = \mathbf{U}_d\mathbf{U}_d^{\mathsf{T}}$ $(d \geq 1)$ correspond to the orthogonal projections onto the spaces spanned by columns of $\widetilde{\mathbf{X}}\mathbf{U}_0$ and $\mathbf{U}_d$ $(d \geq 1)$, respectively. Thus, the problem finally boils down to finding $D+1$ subspaces $\widetilde{\mathbf{X}}\mathbf{U}_0$ and $\{\mathbf{U}_d\}_{d=1}^D$ such that $\widetilde{\mathcal{Y}}$ has the proximal low-rank projection under them.

Rather than simultaneously estimating all the factor matrices, HOLRR obtains an approximate solution via solving $D+1$ independent eigenvectors problems. For nonlinear setting, A kernel extension of HOLRR, namely KHOLRR, is also introduced by exploiting the kernel trick. Theoretically, a generalization bound for the class of tensor-valued regression function is provided, indicating the error is dominated only by $\mathcal{O}(\sqrt{(R^{D+1} + (D+1)I)(D+1)\log(I)})$ instead of $\mathcal{O}(\sqrt{I^{D+1}})$ when imposing the low multilinear rank constraint [Rabusseau and Kadri, 2016].

To summarize, HOLRR is simple and enjoys the computational efficiency due to the closed-form solution without iterations. Different from previous models that all involve tensorial predictors, HOLRR ignores the correlations contained in tensorial predictor by feeding a vector-based input to the model. In practice, most of real-life tensor-based regression tasks involve tensorial predictors, thus the applicability of HOLRR might be limited.

Discussion

In Table 3.1, we summarize the existing linear tensor regression methods in terms of the application context, the input/output data type as well as the applications. Despite arising from different application domains and modeling settings, all these models share the low-rank assumption on the regression coefficient tensor. In other words, the coefficient tensor is assumed to following a low dimensional tensor factorization.

In general, these work can mainly be categorized into two classes of approaches. The first class, including [Romera-Paredes et al., 2013; Wimalawarne et al., 2014; Signoretto et al., 2014], relies on a convex relaxation of the low-rank non-convex problem using spectral regularization, i.e.,

the trace norm of matricizations. Though enjoying nice convex properties, this class suffers from slow convergence rate [Gandy et al., 2011]. On the other hand, the second class applies alternating least squares (ALS) to sequentially estimate one factor at a time while fixing the rest, such as [Zhou et al., 2013; Li et al., 2013; Bro, 1996; Eliseyev and Aksenova, 2013; Zhao et al., 2013a]. TPG [Yu and Liu, 2016] performs the gradient descent to directly optimize coefficient tensor but still resorts to an alternating strategy to estimate projecting factors during the projection step. The ALS based methods are practically effective but shown to exhibit unstable convergence properties and sub-optimal solutions [Yu and Liu, 2016].

Most of tensor regression studies are dedicated to the batch setting where the entire tensor sequence is demanded and processed. For many real-life tasks, however, the inputs and outputs are the extremely large or even infinite tensor sequences, especially in time-critical dynamic environments, where the new data keep coming fast over time. Although we can apply the batch methods to all the data each time a new pair arrives, they will quickly become computationally prohibitive or merely infeasible. Moreover, it is often impossible to store the whole data or require them to be available up front. Towards this end, a few sequential methods, like [Eliseyev and Aksenova, 2013; Yu et al., 2015], focus on such context and resolve the problem to some degree. Nevertheless, [Eliseyev and Aksenova, 2013] tends to be slow; [Yu et al., 2015] is restricted on the special data type and cannot be directly applied to generic tensor sequences. Thus, more studies need to be targeted on the memory efficient sequential methods.

TABLE 3.1 – Summarization of linear tensor regression models.

Model/Algorithm	Context	Input/Output	Application
CP reg. [Zhou et al., 2013]	bat.	tensor/scalar	medical imaging
Tucker reg. [Li et al., 2013]	bat.	tensor/scalar	medical imaging
NPLS [Bro, 1996]	bat.	tensor/tensor	chemometrics
RNPLS [Eliseyev and Aksenova, 2013]	seq.	tensor/tensor	neural signal processing
HOPLS [Zhao et al., 2013a]	bat.	tensor/tensor	neural signal processing
MLMTL [Romera-Paredes et al., 2013]	bat.	vector/vector	multi-task learning
GREEDY [Bahadori et al., 2014]	bat.	tensor/tensor	spatio-temporal analysis
ALTO [Yu et al., 2015]	seq.	tensor/tensor	spatio-temporal analysis
TPG [Yu and Liu, 2016]	bat.	tensor/tensor	spatio-temporal analysis
HOLRR [Rabusseau and Kadri, 2016]	bat.	vector/tensor	spatio-temporal analysis

3.1.2 Nonlinear Tensor Regression

The previous section primarily discusses the parametric tensor regression approaches relying on the linear (multilinear) modeling strategies. However, the linear model might suffer from the lack of predictive power due to the restriction of linearity. Such kind of model is not desirable in terms of predictability in situation where the nonlinear dependencies exist between tensor input and output. In this section, we take a look at some nonlinear tensor regression methods

in the literature that solve this issue to some extent.

Tensor Gaussian Process (Tensor GP)

Gaussian process (GP) [Rasmussen and Williams, 2005] is an important class of probabilistic models that specify a distribution over functions, which is a particular useful to capture the nonlinearity between input and output variables. GP is typically used in a nonlinear regression setting where a set of observations $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ is generated according to the model

$$y = f(\mathbf{x}) + \epsilon, \tag{3.38}$$

where $\mathbf{x}$ is the vector input, y is the scalar output and ϵ is zero-mean gaussian random noise with variance σ^2.

In the context of tensor regression, Zhao et al. [2013b] adapted GP to tensor-valued input space, leading to tensor GP. Concretely, they extend the standard GP regression model (3.38) to the following form

$$y = f(\mathcal{X}) + \epsilon. \tag{3.39}$$

Now the set of observations generated from the nonlinear function $f(\mathcal{X})$ of D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ in (3.39) turn out to be $\{(\mathcal{X}_n, y_n)\}_{n=1}^N$. Then, the latent function in (3.39) can be modeled by a GP

$$f(\mathcal{X}) \sim \mathcal{GP}(m(\mathcal{X}), k(\mathcal{X}, \mathcal{X}')|\theta), \tag{3.40}$$

where $m(\mathcal{X})$ is the mean function that set to be zero, $k(\mathcal{X}, \mathcal{X}')$ is the covariance function with tensorial arguments and θ is the associated hyperparameter vector.

The essential difference between classical GP and tensor GP lies in that the latter uses tensor kernel function $k(\mathcal{X}_m, \mathcal{X}_n)$ to replace regular kernel function $k(\mathbf{x}_m, \mathbf{x}_n)$ as entry of kernel matrix $\mathbf{K}$. Therefore, the key business of tensor GP is to design an appropriate kernel function with tensor-valued input, namely *tensor kernel*, which could effectively exploit the multiway structure of tensorial representations. Some straightforward standard kernels can readily be generalized to valid reproducing tensor kernels, including linear kernel

$$k(\mathcal{X}, \mathcal{Y}) = \langle \mathcal{X}, \mathcal{Y} \rangle = \langle vec(\mathcal{X}), vec(\mathcal{Y}) \rangle \tag{3.41}$$

and Gaussian RBF kernel

$$k(\mathcal{X}, \mathcal{Y}) = \exp(-\frac{1}{2\beta^2} \parallel \mathcal{X} - \mathcal{Y} \parallel_{\mathsf{F}}). \tag{3.42}$$

However, the above kernel function k naively measures a collection of entry-wise Euclidean distances between two tensors, which does not take into consideration the underlying multiway structural information in tensor, resulting in inadequate ability to capture the similarity between tensors. Taking gray-scale images (2nd-order tensor) and videos (3rd-order tensor) as an example, the application of these naive kernels leads to ignoring the relation between each

pixel and its neighbours for images, and the temporal structure will be additionally neglected for videos [Turaga et al., 2010, 2011].

To overcome the information loss when using naive kernels, Zhao et al. [2013b] present a probabilistic tensorial kernel based on generative models and information divergences. In general, the dissimilarity between two tensorial observations can be gauged by a product of a set of dissimilarity measures at each mode, i.e., in the form of *product kernel* [Lin, 2000; Signoretto et al., 2011]

$$k(\mathcal{X}, \mathcal{Y}) = k^1(\mathbf{X}_{(1)}, \mathbf{Y}_{(1)})k^2(\mathbf{X}_{(2)}, \mathbf{Y}_{(2)}) \cdots k^D(\mathbf{X}_{(D)}, \mathbf{Y}_{(D)}), \tag{3.43}$$

where k^d represents the valid factor kernels for mode d. In their work, the d-mode dissimilarity in k^d is characterized by the information divergence between two probabilistic distribution. Zhao et al. [2013b] assume the d-mode unfolding of tensor data $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ to be a matrix with rows representing I_d dimensional multivariate and columns representing $I_1 \cdots I_{d-1} I_{d+1} \cdots I_D$ samples. These I_d dimensional samples are generated from a d-mode parametric distribution $p(\mathbf{x}|\Omega_d)$, where Ω_d is the hyperparameter (for multivariate Gaussian $\Omega_d = \{\mu_d, \Sigma_d\}$) which can be estimated from the d-mode unfolding matrix . Hence, the d-mode dissimilarity measure can be obtained in terms of information divergence $\mathcal{D}(p\|q)$ between two distributions p and q

$$\mathcal{S}_d(\mathcal{X}\|\mathcal{Y}) = \mathcal{D}(p(\mathbf{x}|\Omega_d^{\mathcal{X}}) \parallel q(\mathbf{y}|\Omega_d^{\mathcal{Y}})). \tag{3.44}$$

Consequently, the resulting product tensor kernel, built up by multiplying D factor kernels associated with each mode, is given by

$$k(\mathcal{X}, \mathcal{Y}) = \alpha^2 \prod_{d=1}^{D} \exp(-\frac{1}{2\beta_d^2} \mathcal{S}_d(\mathcal{X} \parallel \mathcal{Y})), \tag{3.45}$$

where α is the magnitude parameter and $\{\beta_d\}_{d=1}^D$ are the parameters indicating length-scales. Note that the probabilistic kernel provides a way to model tensor from D viewpoints corresponding to D different low dimensional vector spaces, in this way multiway structure could be accounted for by the kernel function.

As for the prediction, tensor GP behaves exactly the same as standard GP except that one calculates the entries in kernel matrix $\mathbf{K}$ using tensor kernel function as referred by (3.45). Recall that the computational complexity of GP is $O(N^3)$, which is not desirable for tensor GP because the number of samples N needs to be quite large to achieve a good accuracy in many real-world applications.

Kernel-based Tensor Partial Least Squares (KTPLS)

Applying kernel-based concept to PLS regression, Zhao et al. [2013b] extended HOPLS to the nonlinear kernel-based tensor PLS (KTPLS) with the objective for predicting an output tensor from an input tensor with arbitrary orders. For this purpose, the concatenated N pairs

of observations $\{\mathcal{X} \in \mathbb{R}^{N \times I_1 \times \cdots \times I_M}, \mathcal{Y} \in \mathbb{R}^{N \times J_1 \times \cdots \times J_L}\}$ are mapped into infinite dimensional inner product feature space by ϕ and ψ

$$\begin{aligned} \phi &: \mathcal{X}_n \rightarrow \phi(\mathcal{X}_n) \in \mathbb{R}^{H_1 \times, \ldots, \times H_M} \\ \psi &: \mathcal{Y}_n \rightarrow \psi(\mathcal{Y}_n) \in \mathbb{R}^{K_1 \times, \ldots, \times K_L}, \end{aligned} \tag{3.46}$$

yielding $\phi(\mathcal{X})$ and $\psi(\mathcal{Y})$ that are denoted as Φ and Ψ respectively. In the light of HOPLS, KTPLS conducts the Tucker decompositions of Φ and Ψ simultaneously in the feature space by

$$\begin{aligned} \Phi &= \mathcal{G}_{\mathcal{X}} \times_1 \mathbf{T} \times_2 \mathbf{P}^{(1)} \times \cdots \times_{M+1} \mathbf{P}^{(M)} + \epsilon_{\mathcal{X}} \\ \Psi &= \mathcal{G}_{\mathcal{Y}} \times_1 \mathbf{U} \times_2 \mathbf{Q}^{(1)} \times \cdots \times_{L+1} \mathbf{Q}^{(L)} + \epsilon_{\mathcal{Y}} \\ \mathbf{U} &= \mathbf{T}\mathbf{D} + \mathbf{E}, \end{aligned} \tag{3.47}$$

where $\{\mathcal{G}_{\mathcal{X}}, \mathcal{G}_{\mathcal{Y}}\}$ are the core tensors while $\{\mathbf{P}^{(m)}, \mathbf{Q}^{(l)}\}$ refer to the loading matrices, all of which are in the feature space. $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_R]$, $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R]$ and $\mathbf{D}$ is a diagonal matrix denoting the inner relation between latent vectors $\mathbf{t}_r$ and $\mathbf{u}_r$. $\mathbf{E}$ is the residual for $\mathbf{U}$. By defining $\tilde{\mathcal{G}}_{\mathcal{X}}$ and $\tilde{\mathcal{G}}_{\mathcal{Y}}$ as

$$\begin{aligned} \tilde{\mathcal{G}}_{\mathcal{X}} &= \mathcal{G}_{\mathcal{X}} \times_2 \mathbf{P}^{(1)} \times \cdots \times_{D_1+1} \mathbf{P}^{(M)} \\ \tilde{\mathcal{G}}_{\mathcal{Y}} &= \mathcal{G}_{\mathcal{Y}} \times_2 \mathbf{Q}^{(1)} \times \cdots \times_{D_2+1} \mathbf{Q}^{(L)}, \end{aligned} \tag{3.48}$$

equations in (3.47) can be rewritten as

$$\begin{aligned} \tilde{\mathcal{G}}_{\mathcal{X}} &= \Phi \times_1 \mathbf{T}^{\mathsf{T}} \\ \tilde{\mathcal{G}}_{\mathcal{Y}} &= \Psi \times_1 \mathbf{U}^{\mathsf{T}}. \end{aligned} \tag{3.49}$$

On one hand, both $\tilde{\mathcal{G}}_{\mathcal{X}}$ and $\tilde{\mathcal{G}}_{\mathcal{Y}}$ lie in infinite dimensional feature space and thus they cannot be estimated explicitly. On the other hand, equations (3.49) can be in fact expressed as a linear combination of observations $\{\phi(\mathcal{X}_n)\}_{n=1}^N$ and $\{\phi(\mathcal{Y}_n)\}_{n=1}^N$. Therefore, one only needs to compute $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R]$, $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_R]$ instead of $\tilde{\mathcal{G}}_{\mathcal{X}}, \tilde{\mathcal{G}}_{\mathcal{Y}}$ with pairwise maximum covariance by sequentially solving the following optimization problem using a deflation procedure

$$\max_{\{\mathbf{w}_r^{(d_1)}, \mathbf{v}_r^{(d_2)}\}} [cov(\mathbf{t}_r, \mathbf{u}_r)]^2 \quad r = 1, ..., R, \tag{3.50}$$

where

$$\begin{aligned} \mathbf{t}_r &= \Phi \times_2 \mathbf{w}_r^{(1)\mathsf{T}} \times \cdots \times_{D_1+1} \mathbf{w}_r^{(D_1)\mathsf{T}} \\ \mathbf{u}_r &= \Phi \times_2 \mathbf{v}_r^{(1)\mathsf{T}} \times \cdots \times_{D_2+1} \mathbf{v}_r^{(D_2)\mathsf{T}}. \end{aligned} \tag{3.51}$$

Then the latent vectors can be obtained by equivalently solving the following eigenvalue problem [Rosipal and Trejo, 2002]

$$\begin{aligned} \Phi_{(1)}\Phi_{(1)}^{\mathsf{T}}\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_r &= \lambda \mathbf{t}_r \\ \mathbf{u}_r &= \Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_r. \end{aligned} \tag{3.52}$$

Actually, $\Phi_{(1)}\Phi_{(1)}^{\mathsf{T}}$ and $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}$ can be presented as a kernel matrix $\mathbf{K}_{\mathcal{X}}$ and $\mathbf{K}_{\mathcal{Y}}$, respectively. The preceding equations in (3.52) turn out to be

$$\begin{aligned} \mathbf{K}_{\mathcal{X}}\mathbf{K}_{\mathcal{Y}}\mathbf{t}_r &= \lambda \mathbf{t}_r \\ \mathbf{u}_r &= \mathbf{K}_{\mathcal{Y}}\mathbf{t}_r, \end{aligned} \tag{3.53}$$

which is where tensor kernel functions $(\mathbf{K}_{\mathcal{X}})_{mn} = k(\mathcal{X}_m, \mathcal{X}_n)$ and $(\mathbf{K}_{\mathcal{Y}})_{mn} = k(\mathcal{Y}_m, \mathcal{Y}_n)$ arise.

In these conditions, for a given new test point $\mathcal{X}_*$, the prediction y_* is as follows

$$y_*^{\mathsf{T}} = \mathbf{k}_*^{\mathsf{T}}\mathbf{U}(\mathbf{T}^{\mathsf{T}}\mathbf{K}_{\mathcal{X}}\mathbf{U})^{-1}\mathbf{T}^{\mathsf{T}}\mathbf{Y}_{(1)}, \tag{3.54}$$

where $(\mathbf{k}_*)_n = k(\mathcal{X}_*, \mathcal{X}_n)$. Note that here y_* is in a vector form and should be reformulated into a tensor form $\mathcal{Y}_*$. From this result, we should pay attention that by letting

$$\alpha = \mathbf{U}(\mathbf{T}^{\mathsf{T}}\mathbf{K}_{\mathcal{X}}\mathbf{U})^{-1}\mathbf{T}^{T}\mathbf{Y}_{(1)}, \tag{3.55}$$

the response $\mathcal{Y}_*$ can be predicted as a linear combination of N kernel functions

$$y_* = \sum_{n=1}^{N} \alpha_n k(\mathcal{X}_n, \mathcal{X}_*), \tag{3.56}$$

which coincides with the result from GP method [Zhao et al., 2013b].

Tensor Regression Networks

Nowadays, deep neural networks (DNN), particularly deep convolutional neural networks (CNN), have demonstrated remarkable performance improvement in a variety of fields. Several papers have been devoted to study the connection between DNN and tensor methods. However, seldom work has been done to investigate the deep tensor regression methods. Recently, Kossaifi et al. [2017] make their attempts to fist incorporate tensor regression as pluggable component into CNN.

The key idea behind [Kossaifi et al., 2017] is to reformulate the weight matrix in the fully connected (FC) layer as regression coefficient tensor which is assumed to be decomposed in a low-rank Tucker format. Figure 3.2 shows the tensor regression layer where the softmax outputs $\mathbf{y}$ result from the inner products between the activation tensor $\widetilde{\mathcal{X}}$ and the corresponding low-rank regression weights $\widetilde{\mathcal{W}}$.

By replacing FC layer with the proposed tensor regression layer (TRL), huge amount of parameters in FC layer can be significantly reduced, while the multiway structure information of activation tensor, generated from the core of CNN, can be preserved. In essence, TRL can be viewed as a nonlinear embedding of Tucker regression model [Li et al., 2013] into the architecture of CNN to further leverage its efficiency and representational expressivity. Nevertheless, the applicability of TRL somehow becomes diminished since it is specially designed for the CNN, typically with 3rd-order tensorial input.

Discussion

In this section, we reviewed the existing nonlinear regression models addressing the tensorial input and output. These approaches are summarized in Table 3.2. Generally, two modeling strategies can be adopted to capture the nonlinear dependencies between the tensorial input and output: kernel based and neural networks based methods.

Most of the kerneled based approaches, such as [Zhao et al., 2013b; Rabusseau and Kadri, 2016], are directly extended from their linear counterparts by exploiting the kernel trick. The key factor to the success of the kernel strategy is how to design appropriate tensorial function that is able to preserve multiway structure in tensor data while being computationally efficient. Currently, the most commonly used tensor kernel functions, e.g., chordal distance tensor kernel [Signoretto et al., 2011] or probabilistic tensor kernel [Zhao et al., 2013b], tend to demand high computational and storage load.

For anther promising research line, very few work have focused the intersection of tensor regression and deep learning except [Kossaifi et al., 2017]. Benefiting from the deep structure, one can explore a novel class of nonlinear tensor regression models that can represent the tensorial regression coefficient so as to have a better feature or latent components representation of data, which is expected to have a further enhanced predictability. The CNN has been shown empirically effective on 3rd-order tensor input and particularly useful in computer vision related tasks. Yet, how to extend advantageous deep architecture to more generic high order tensorial input and output data still remains a challenging problem.

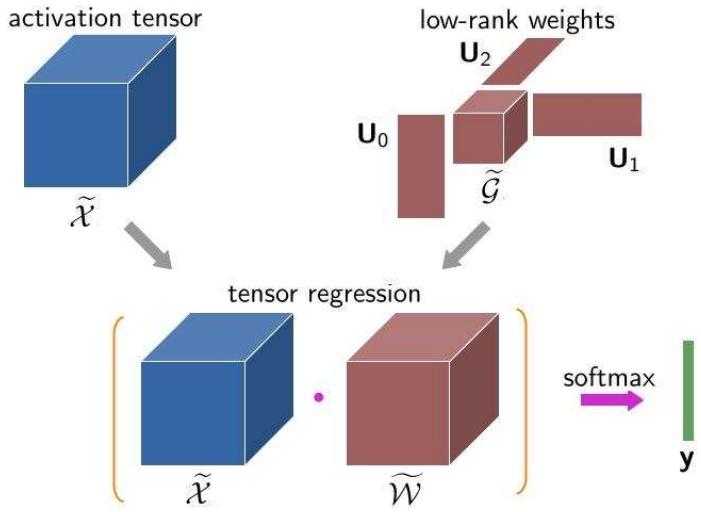


FIGURE 3.2 – An illustration of tensor regression layer (TRL).

TABLE 3.2 – Summarization of nonlinear tensor regression models.

Model/Algorithm	Cont.	Input/Output	Application
tensor GP [Zhao et al., 2013b]	bat.	tensor/scalar	neural signal processing
KTPLS [Zhao et al., 2013b]	bat.	tensor/tensor	neural signal processing
KHOLRR [Rabusseau and Kadri, 2016]	bat.	vector/tensor	spatio-temporal analysis
TRL [Kossaifi et al., 2017]	bat.	tensor/vector	computer vision

3.2 Our new contributions: the big picture

Before introducing our main contributions in details, it is helpful to first take a glance at them from the very top level. In Table 3.3, our new tensor regression models introduced in the remaining chapters are categorized into linear or nonlinear model, sequential or batch model, as well as model with scalar or tensor-variate output.

TABLE 3.3 – The overview of our new contributions.

Settings		Scalar Output	Tensor-variate Output
Linear	Batch	$\mathcal{H}$-Tucker regression	
	Sequential		IHOPLS ; RHOPLS
Nonlinear	Batch		KMTPLS
	Sequential	Tensor OLGP	

Under the multilinear assumption, our batch $\mathcal{H}$-Tucker regression model (Chapter 4) is able to handle the potentially very high-order tensorial input and scalar output. Our sequential IHOPLS (Chapter 6) is designed for the context of infinite time-dependent streams, while RHOPLS (Chapter 7) is a general sequential model especially useful for large-scale high-speed general tensor sequences. Both our IHOPLS and RHOPLS deal with tensor-variate input and tensor-variate output.

In the setting of nonlinear tensor regression, our sequential tensor OLGP (Chapter 5) with scalar output can be applied to large-scale tensor sequences in an online fashion. Finally, our batch KMTPLS (Chapter 8) addresses the nonlinearity between set of tensor-variate input and output blocks by incorporating the kernel machine into the model, which is used for merging tensor blocks from different sources to boost the predictability.

3.3 Conclusion

In this chapter, we briefly summarized the technical details of tensor regression and its recent advance, and categorized them in terms of linear modeling and nonlinear modeling. As for the linear models, the discussed approaches share the low-rank assumption that the model is parameterized by a high-order tensor which exists a low dimensional factorization. The

nonlinear models mainly take the advantage of kernel-based techniques to develop tensorial kernels and incorporate them into the well developed regression frameworks to accommodate high-order tensor input or output. We also gave the big picture of our new contributions to tensor regression modelings from the very top level. In the remaining chapters, we will introduce our contributions in details that address some of the issues of existing tensor regression approaches from several different perspectives.

In the next chapter, we will introduce our first contribution to address the tensor regression task with high-order tensorial input and scalar output.

Chapitre 4

Hierarchical Tucker Tensor Regression

In this chapter, a highly compact, flexible, and scalable hierarchical Tucker tensor regression framework [Hou and Chaib-draa, 2015] is presented based on the hierarchical Tucker decomposition (HTD, or $\mathcal{H}$-Tucker) [Hackbusch and Kühn, 2009; Grasedyck, 2010]. Specifically, the new regression model applies the $\mathcal{H}$-Tucker decomposition to approximate the coefficient tensor, leading to a sparse representation of fewer parameters. On one hand, this new approach shares the advantage of CP based model (presented in Chapter 2) in that its parameter complexity is free from exponential dependence on the order d. On the other hand, it preserves the flexibility like Tucker model by allowing distinct ranks according to a dimension tree structure. With this HTD framework, a scalable block relaxation algorithm is developed to conduct the parameter estimation. Finally, empirical studies on both synthetic simulation and real-life task demonstrate the effectiveness of this new approach.

4.1 Introduction

Recently, tensor-variate regression approaches have attracted increasing interest in medical imaging data analysis [Zhou et al., 2013; Li et al., 2013]. As we have already mentioned in previous chapters, these regression tasks involving higher-order tensors pose great challenges in two perspectives. On one hand, the ultrahigh dimensionality of tensor input like $3D$ or $4D$ image results in gigantic number of parameters. For instance, in brain image data analysis, the total number of parameters for a $3D$ magnetic resonance imaging (MRI) image of size $128 \times 128 \times 128$ amounts to 2 millions, while this number will end up with as high as 268 millions if we have a $4D$ functional magnetic resonance imaging (fMRI) image of size $128 \times 128 \times 128 \times 128$. Such huge number of parameters induced by these large-scale images causes severe issues in both theoretical aspect and practical aspect. As already noticed, the underlying complex multiway structural information contained in high-order tensors cannot be captured by simply turning them into a vector or matrix and applying two-way data analysis tools.

To tackle above issues, Zhou et al. [2013] assumed the tensor regression coefficient follows a low-

rank structure in terms of CP decomposition, resulting in the following CP tensor regression model

$$\begin{aligned} y &= \langle \mathcal{X}, \mathcal{B} \rangle + b \\ &= \langle \mathcal{X}, \sum_{r=1}^{R} \mathbf{b}_r^{(1)} \circ \cdots \circ \mathbf{b}_r^{(D)} \rangle + b, \end{aligned} \tag{4.1}$$

where the output $y \in \mathbb{R}$ is a scalar and the intercept $b \in \mathbb{R}$ is also a scalar. The input $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is a D-order tensor, so is the coefficient tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$. The factors $\mathbf{b}_r^{(d)} \in \mathbb{R}^{I_d}$ with $r = 1, ..., R$, $d = 1, ..., D$ are vectors. The number of parameters in (4.1) is substantially reduced from $\mathcal{O}(I^D)$ to $\mathcal{O}(DIR)$ with $I = \max\{I_d\}_{d=1}^{D}$, leading to an efficient estimation of factors.

However, the resulting model is simple but not flexible, due to the diagonal structure of core tensor in CP format (see Figure 2.8 and Section 2.2.1). The lack of flexibility might lead to an underfitting situation. To overcome this inflexibility of the CP based model, Li et al. [2013] employed Tucker format to tensorial input space instead. Tucker format is much more flexible since the model uses full core tensor rather than diagonal structured core tensor as CP.

By doing so, it is possible to handle the complex interactions between distinct factors across different modes. This also enables to model different modes using distinct ranks. Recall that the linear Tucker tensor regression model is given by

$$\begin{aligned} y &= \langle \mathcal{X}, \mathcal{B} \rangle + b \\ &= \langle \mathcal{X}, \sum_{r_1=1}^{R_1} \cdots \sum_{r_D=1}^{R_D} g_{r_1,...,r_D} \mathbf{b}_{r_1}^{(1)} \circ \cdots \circ \mathbf{b}_{r_D}^{(D)} \rangle + b, \end{aligned} \tag{4.2}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$ with entries $g_{r_1,...,r_D}$ denotes the core tensor. The factor $\mathbf{b}_{r_d}^{(d)} \in \mathbb{R}^{I_d}$ with $r_d = 1, ..., R_d$, $d = 1, ..., D$ is the column vector contained in factor matrix $\mathbf{B}_d \in \mathbb{R}^{I_d \times R_d}$. We denote $R = \max\{R_d\}_{d=1}^{D}$. This Tucker regression model significantly decreases to $\mathcal{O}(DIR + R^D)$ parameters while the original coefficient tensor has $\mathcal{O}(I^D)$ parameters.

Although the compression rate in (4.2) is tremendous, it is worth noting that for small orders, especially for the case of $D = 3$, Tucker model could easily replace the CP model. However, for larger order tensors, e.g., D is 4, 5, 6 and so on, Tucker based model is ineffective and not scalable since $\mathcal{O}(R^D)$ becomes the dominant term in $\mathcal{O}(DIR + R^D)$ and grows exponentially with the tensor order D, which is known as the curse of dimensionality [Oommen et al., 2008].

4.2 Hierarchical Tucker Decomposition (HTD)

Various other decompositions have been introduced to address the inadequacy of flexibility in CP model and avoid the exponential parameter growth with tensor order in Tucker model.

These new decompositions maintain the advantages of both CP model and Tucker model at the same time. One of them generalizes Tucker decomposition to hierarchical Tucker decomposition (HTD) [Hackbusch and Kühn, 2009; Grasedyck, 2010], which is also known as $\mathcal{H}$-Tucker decomposition. On one hand, $\mathcal{H}$-Tucker representation shares the advantage of CP model in that the total number of parameters as well as the storage complexity are free from exponential dependence on D. On the other hand, $\mathcal{H}$-Tucker preserves the flexibility of Tucker model and allows us to perform efficient linear algebra operations based on SVD of matrix unfoldings of a tensor.

The $\mathcal{H}$-Tucker is a novel structured format which efficiently represents a higher-order tensor by means of subspace approximation in a multi-level fashion [Hackbusch and Kühn, 2009; Grasedyck, 2010]. In general, the idea behind $\mathcal{H}$-Tucker is to recursively separate the modes of the tensor, which leads to a binary dimension tree $\mathcal{T}$ containing a subset of modes $t \subset \{1, 2, ..., D\}$ at each node of the tree. In this format, one can find the optimal subspace corresponding to each node t in $\mathcal{T}$.

A *binary dimension tree* $\mathcal{T}$ for a D-order tensor is a finite tree with root node, $root = \{1, ..., D\}$, and depth p such that each node $t \in \mathcal{T}$ is either

- a singleton mode $t = \{\mu\}$ corresponding to a leaf node, or
- the union of two disjoint successors $\mathcal{S}(t) = \{t_l, t_r\}$ such that $t = t_l \cup t_r$.

Here we denote $\mathcal{L}(\mathcal{T})$ as the set of leaf nodes, while $\mathcal{N}(\mathcal{T}) = \mathcal{T} \backslash \mathcal{L}(\mathcal{T})$ is denoted the set of inner nodes. The set $\mathcal{L}(\mathcal{T})$ contains all the leaves, each with a singleton mode. $\mathcal{N}(\mathcal{T})$ contains the interior nodes each with a subset of all modes. For $t \in \mathcal{N}(\mathcal{T})$, it is a disjoint union of left and right children $t = t_l \cup t_r$.

In this chapter, we employ the balanced binary tree in which the heights of left subtree and right subtree of each node differ by at most one. For example, a balanced binary dimension tree representing a 5-order tensor is illustrated in Figure 4.1. $\mathcal{L}(\mathcal{T})$ contains the leaves nodes {1},{2},{3},{4}, and {5}, each of which links with only one mode. $\mathcal{N}(\mathcal{T})$ includes all the inner nodes $\{1, 2\}$, $\{1, 2, 3\}$ and $\{4, 5\}$ as well as the root node $\{1, 2, 3, 4, 5\}$. Each inner node is related to a collection of modes, while the root node corresponds to all the modes. The purpose of the dimension tree is that all modes of tensor can be split and organized in a hierarchical fashion, such that the subspaces corresponding to these mode partitions can be efficiently represented in a nested way.

Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ be a D-order tensor. Tucker decomposition represents tensor $\mathcal{X}$ via the subspaces of D unfoldings $\{\mathbf{X}_{(d)}\}_{d=1,...,D}$, each of which corresponds to one specific mode d. Unlike Tucker, $\mathcal{H}$-Tucker uses a hierarchy of subspaces of the unfoldings $\{\mathbf{X}_{(t)}\}_{t \in \mathcal{T}}$ to represent $\mathcal{X}$, with each unfolding corresponding to one specific node t in dimension tree $\mathcal{T}$. One such node t might relate to a set of multiple modes. For each $t \in \mathcal{T}$, the $\mathbf{X}_{(t)}$ can be obtained by merging all the modes in subset t into the row indices and all the rest modes into the

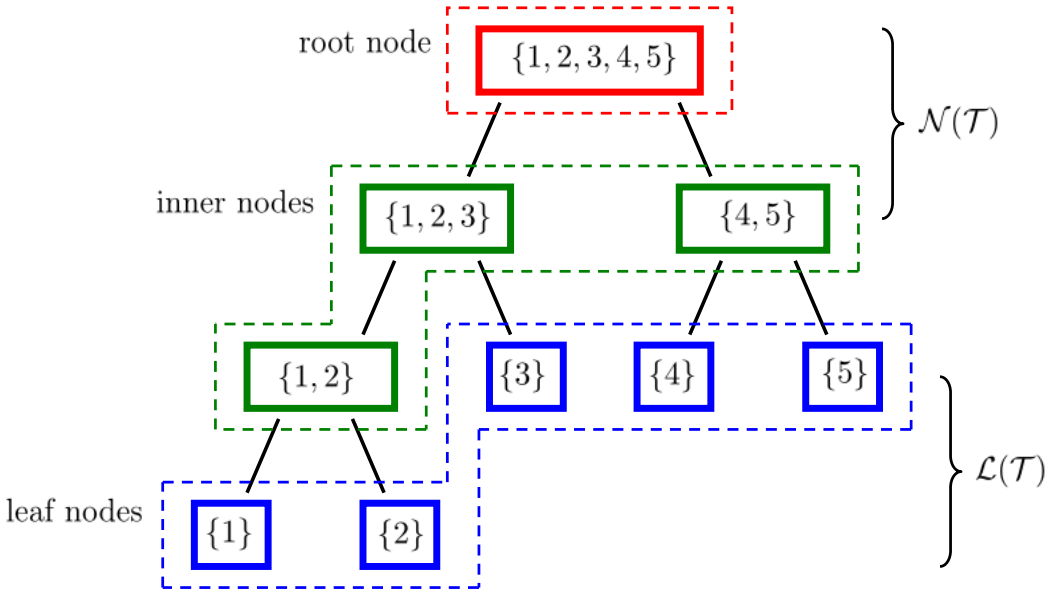


FIGURE 4.1 – Illustration of a balanced binary dimension tree $\mathcal{T}$ for a 5-order tensor.

column indices. For example, the unfolding $\mathbf{X}_{(\{1,2,3\})}$ of $\mathcal{X}$ associated with inner node $\{1,2,3\}$ in Figure 4.1 is obtained by reshaping $\mathcal{X}$ into a matrix, where the row index of an element of this matrix is determined by the indices from modes 1, 2, and 3, while the column index of that element is determined by the indices from modes 4 and 5 ; the unfolding $\mathbf{X}_{(\{1\})}$ of $\mathcal{X}$ associated with leaf node $\{1\}$ is essentially the standard unfolding of $\mathcal{X}$ along the mode 1 ; the unfolding of $\mathbf{X}_{(\{1,2,3,4,5\})}$ of the root node leads to nothing but a long vector, as a consequence of the standard vectorization operation on $\mathcal{X}$.

With $\mathcal{H}$-Tucker representation, we can then look for a set of basis factor matrices $\mathbf{U}^t$ for $t \in \mathcal{T}$, the columns of which span the column space of matrix $\mathbf{X}_{(t)}$, to represent the tensor $\mathcal{X}$. The dimension of $\mathbf{X}_{(t)}$, namely the rank $r_t = rank(\mathbf{X}_{(t)})$, is equal to the number of columns of $\mathbf{U}^t$. Analogous to the d-rank of Tucker, the collection of these ranks at all the nodes $(r_t)_{t\in\mathcal{T}}$ is defined as ht-rank of $\mathcal{X}$, i.e., the ht-rank in above example in Figure 4.1 can be denoted as $(1\text{-}r_{123}r_{45}\text{-}r_{12}r_3r_4r_5\text{-}r_1r_2)$. Note that here we define the rank r_{12345} corresponding to the root node t_{root} as 1, since basis factor matrix $\mathbf{U}^{t_{root}}$ associated with t_{root} incorporates all the modes into row indices, leaving $\mathbf{U}^{t_{root}}$ as a long vector. In contrast, the d-rank is defined as the column rank of d-mode matricization for Tucker.

Formally, let $\mathcal{T}$ be a dimension tree, the *hierarchical rank* or ht-rank $\underline{r}= (r_t)_{t\in\mathcal{T}}$ of a tensor $\mathcal{X}$ is defined as

$$r_t = rank(\mathbf{X}_{(t)}) \quad \forall t \in \mathcal{T}. \tag{4.3}$$

The set of all tensors of hierarchical rank at most $\underline{r}$, called $\mathcal{H}$-Tucker($\underline{r}$) tensors, are given by

$$\mathcal{H}\text{-Tucker}(\underline{r}) = \mathcal{H}\text{-Tucker}((r_t)_{t\in\mathcal{T}}) = \{\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_D} : rank(\mathbf{X}_{(t)}) = r_t, \ \ \forall t \in \mathcal{T}\}. \tag{4.4}$$

Let us now introduce the following nestedness property according to [Grasedyck, 2010].

Proposition 1 [Grasedyck, 2010]: Let $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_D}$, for $t \in \mathcal{N}(\mathcal{T})$, $t = t_l \cup t_r$ and $t_l \cap t_r = \emptyset$, then we have

$$span(\mathbf{X}_{(t)}) \subset span(\mathbf{X}_{(t_r)} \otimes \mathbf{X}_{(t_l)}), \tag{4.5}$$

where each non-leaf node has one left child node and one right child node whose modes are not overlapped, i.e., dimensions corresponding to t is the disjoint union of the dimensions corresponding to t_l and t_r. Equation (4.5) says that the column space of unfolding matrix is the subspace of the column space of tensor product between unfolding matrices of its children nodes.

This property suggests a relation of subspaces of matricizations between the parent and children, which enables us to establish the link between the corresponding basis factor matrices using a so-called *transfer matrix* $\mathbf{B}^t$ in

$$\mathbf{U}^t = (\mathbf{U}^{t_l} \otimes \mathbf{U}^{t_r})\mathbf{B}^t, \tag{4.6}$$

where $\mathbf{B}^t \in \mathbb{R}^{r_{t_l} r_{t_r} \times r_t}$ and r_{t_l}, r_{t_r}, r_t are the ht-rank at nodes t_l, t_l and t, respectively. Starting from the leaf singletons, the construction of $\mathcal{H}$-Tucker proceeds by applying equation (4.6) recursively until the root is reached. Note that r_t is fixed to 1 at the root node. As a result, this fact implies that the tensor $\mathcal{X}$ can be completely parameterized just by the transfer matrices $\{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}$ and basis factor matrices $\{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}$. Therefore, the overall complexity for storage is bounded by $\mathcal{O}(DIR + DR^3)$, where $I = \max\{I_i\}_{i=1}^D$ and $R = \max\{r_t\}_{t\in\mathcal{T}}$.

Example 2: Let's consider the $\mathcal{H}$-Tucker format depicted in Figure 4.2 for a 5-order tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_5}$ with dimension tree structure described in Figure 4.1.

In Figure 4.2, each subtree, highlighted using dotted rectangle, can be represented by a basis factor matrix $\mathbf{U}^t$ with t corresponding to the root of that subtree. For instance, the subtree located by the green(2) dotted rectangle can be represented via basis factor matrix $\mathbf{U}^{\{1,2,3\}}$ that summarizes the subspace corresponding to the inner node $\{1,2,3\}$, and this basis factor matrix $\mathbf{U}^{\{1,2,3\}}$ is in turn expressed using a transfer matrix $\mathbf{B}^{\{1,2,3\}}$ and two basis factor matrices $\mathbf{U}^{\{1,2\}}$, $\mathbf{U}^{\{3\}}$ associated with its children nodes $\{1,2\}$, $\{3\}$. Similarly, the subtree located by the blue(3) dotted rectangle can be represented via basis factor matrix $\mathbf{U}^{\{1,2\}}$ that describes the subspace corresponding to the inner node $\{1,2\}$. The factor matrix $\mathbf{U}^{\{1,2\}}$ can be expressed using a transfer matrix $\mathbf{B}^{\{1,2\}}$ and two basis factor matrices $\mathbf{U}^{\{1\}}$, $\mathbf{U}^{\{2\}}$ associated with its two children nodes $\{1\}$, $\{2\}$. At the leaf level the dimension tree $\mathcal{T}$, all subtrees turn out to be the leaf singletons and can be directly represented using a single basis factor matrix $\mathbf{U}^{\{i\}}$ for $i = 1,..,5$.

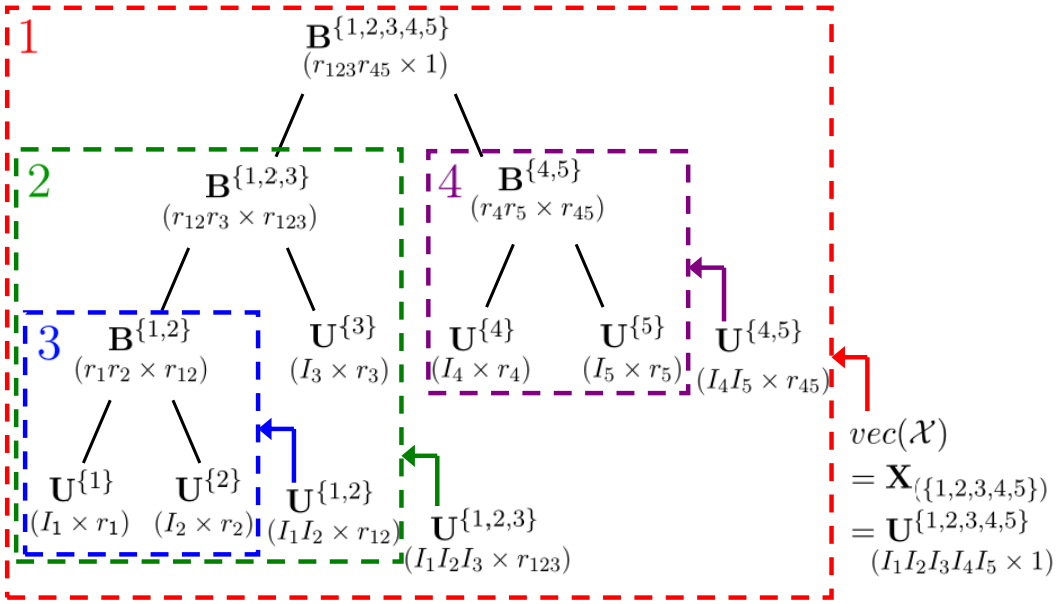


FIGURE 4.2 – Illustration of $\mathcal{H}$-Tucker format for a 5-order tensor.

To show how to represent the 5-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_5}$ in terms of nested subspaces, we start the procedure from bottom to top of the tree. The basis factor matrix $\mathbf{U}^{\{1,2\}} \in \mathbb{R}^{I_1 I_2 \times r_{12}}$ of inner node $\{1,2\}$ is obtained by tensor product of $\mathbf{U}^{\{1\}} \in \mathbb{R}^{I_1 \times r_1}$ and $\mathbf{U}^{\{2\}} \in \mathbb{R}^{I_2 \times r_2}$ and then transformed by the matrix $\mathbf{B}^{\{1,2\}} \in \mathbb{R}^{r_1 r_2 \times r_{12}}$ as

$$\mathbf{U}^{\{1,2\}} = (\mathbf{U}^{\{1\}} \otimes \mathbf{U}^{\{2\}})\mathbf{B}^{\{1,2\}}. \tag{4.7}$$

With $\mathbf{U}^{\{1,2\}}$ in hand, then the basis factor matrix $\mathbf{U}^{\{1,2,3\}} \in \mathbb{R}^{I_1 I_2 I_3 \times r_{123}}$ of inner node $\{1,2,3\}$ can be calculated by $\mathbf{U}^{\{1,2\}} \in \mathbb{R}^{I_1 I_2 \times r_{12}}$, $\mathbf{U}^{\{3\}} \in \mathbb{R}^{I_3 \times r_3}$ and $\mathbf{B}^{\{1,2,3\}} \in \mathbb{R}^{r_{12} r_3 \times r_{123}}$ as

$$\mathbf{U}^{\{1,2,3\}} = (\mathbf{U}^{\{1,2\}} \otimes \mathbf{U}^{\{3\}})\mathbf{B}^{\{1,2,3\}}. \tag{4.8}$$

Following the same pattern, we can reach the basis factor matrix $\mathbf{U}^{\{4,5\}} \in \mathbb{R}^{I_4 I_5 \times r_{45}}$ of inner node $\{4,5\}$ using $\mathbf{U}^{\{4\}} \in \mathbb{R}^{I_4 \times r_4}$ and $\mathbf{U}^{\{5\}} \in \mathbb{R}^{I_5 \times r_5}$ and $\mathbf{B}^{\{4,5\}} \in \mathbb{R}^{r_4 r_5 \times r_{45}}$ as

$$\mathbf{U}^{\{4,5\}} = (\mathbf{U}^{\{4\}} \otimes \mathbf{U}^{\{5\}})\mathbf{B}^{\{4,5\}}. \tag{4.9}$$

Finally, for the root node $\{1,2,3,4,5\}$, we assume the dimension of vector space spanned by the columns of $\mathbf{U}^{\{1,2,3,4,5\}}$ is one, that is, $r_{12345} = 1$. We also assume the vectorized tensor $vec(\mathcal{X})$ lies in this vector space. With $\mathbf{U}^{\{1,2,3\}}$, $\mathbf{U}^{\{4,5\}}$ and $\mathbf{B}^{\{1,2,3,4,5\}} \in \mathbb{R}^{r_{123} r_{45} \times r_{12345}}$, the

basis factor matrix $\mathbf{U}^{\{1,2,3,4,5\}} \in \mathbb{R}^{I_1 I_2 I_3 I_4 I_5 \times r_{12345}}$ of the root note can be achieved as

$$\begin{aligned} vec(\mathcal{X}) &= \mathbf{X}_{(\{1,2,3,4,5\})} \\ &= \mathbf{U}^{\{1,2,3,4,5\}} \\ &= (\mathbf{U}^{\{1,2,3\}} \otimes \mathbf{U}^{\{4,5\}})\mathbf{B}^{\{1,2,3,4,5\}} \\ &= ((\mathbf{U}^{\{1,2\}} \otimes \mathbf{U}^{\{3\}})\mathbf{B}^{\{1,2,3\}} \otimes (\mathbf{U}^{\{4\}} \otimes \mathbf{U}^{\{5\}})\mathbf{B}^{\{4,5\}})\mathbf{B}^{\{1,2,3,4,5\}} \\ &= ((\mathbf{U}^{\{1,2\}} \otimes \mathbf{U}^{\{3\}} \otimes \mathbf{U}^{\{4\}} \otimes \mathbf{U}^{\{5\}})(\mathbf{B}^{\{1,2,3\}} \otimes \mathbf{B}^{\{4,5\}})\mathbf{B}^{\{1,2,3,4,5\}} \\ &= ((\mathbf{U}^{\{1\}} \otimes \mathbf{U}^{\{2\}})\mathbf{B}^{\{1,2\}} \otimes \mathbf{U}^{\{3\}} \otimes \mathbf{U}^{\{4\}} \otimes \mathbf{U}^{\{5\}})(\mathbf{B}^{\{1,2,3\}} \otimes \mathbf{B}^{\{4,5\}})\mathbf{B}^{\{1,2,3,4,5\}} \\ &= (\mathbf{U}^{\{1\}} \otimes \mathbf{U}^{\{2\}} \otimes \mathbf{U}^{\{3\}} \otimes \mathbf{U}^{\{4\}} \otimes \mathbf{U}^{\{5\}})(\mathbf{B}^{\{1,2\}} \otimes \mathbf{I}^{\{3\}} \otimes \mathbf{I}^{\{4\}} \otimes \mathbf{I}^{\{5\}}) \\ &\quad (\mathbf{B}^{\{1,2,3\}} \otimes \mathbf{B}^{\{4,5\}})\mathbf{B}^{\{1,2,3,4,5\}}. \end{aligned} \tag{4.10}$$

From above equations, it is obvious that $\mathcal{X}$ depends only on basis factor matrices $\{\mathbf{U}^{\{i\}}\}_{i=1,\ldots,5}$ of leaf nodes, and the transferring matrices $\mathbf{B}^{\{1,2\}}$, $\mathbf{B}^{\{1,2,3\}}$, $\mathbf{B}^{\{4,5\}}$ and $\mathbf{B}^{\{1,2,3,4,5\}}$ of inner nodes. $\mathbf{I}^{\{3\}}$, $\mathbf{I}^{\{4\}}$ and $\mathbf{I}^{\{5\}}$ are the identity matrices.

4.3 $\mathcal{H}$-Tucker Tensor Regression Model

Our modeling strategy for tensor regression is based on the framework of generalized linear model (GLM) [Nelder and Baker, 1972; McCullagh and Nelder, 1989] (see Appendix A.4).

As previously stated, $\mathcal{H}$-Tucker is developed to keep all the merits of tensor subspace representation introduced via Tucker, while avoiding the exponential increase in the parameter size of the coefficient tensor. In $\mathcal{H}$-Tucker, the concept of tensor subspaces is exploited by choosing the subspace of tensor product of a pair of basis factors corresponding to children nodes. By recursively repeating this idea of subspace compression from the root down to the leaves, we obtain a hierarchical subspace representation with a massively reduced number of parameters.

With these benefits in hand, we naturally apply the $\mathcal{H}$-Tucker decomposition to the input space. That is, given a dimension tree $\mathcal{T}$ and ht-rank $\underline{r} = (r_t)_{t\in\mathcal{T}}$, the coefficient tensor $\mathcal{B}$ is assumed to follow a $\mathcal{H}$-Tucker($\underline{r}$) decomposition. The linear systematic part of generalized linear $\mathcal{H}$-Tucker tensor regression model [Hou and Chaib-draa, 2015] can be obtained by

$$\begin{aligned} g(\mu) &= \eta \\ &= \langle \mathcal{B}, \mathcal{X} \rangle + \beta^{\mathsf{T}}\mathbf{z} + b \\ &= \langle vec(\mathcal{B}), vec(\mathcal{X}) \rangle + \beta^{\mathsf{T}}\mathbf{z} + b. \end{aligned} \tag{4.11}$$

Similar to the linear part of classical GLM in (A.12) (see Appendix A.4), η is a linear combination of unknown parameters; μ is the expected response and $g(\cdot)$ is the link function [McCullagh and Nelder, 1989]. β denotes the regular vector coefficient and $\mathbf{z}$ corresponds to the regular vector predictor. The linear term $\beta^{\mathsf{T}}\mathbf{z}$ is incorporated to model the conventional features, e.g., in medical imaging data analysis, the gender, the age and the handiness of a subject are described by this term.

For the root node, we have

$$vec(\mathcal{B}) = \mathbf{U}^{root} \quad \text{with} \quad r_{root} = 1, \tag{4.12}$$

then applying nestedness property (4.6) to (4.12), we get

$$vec(\mathcal{B}) = (\mathbf{U}^{root_r} \otimes \mathbf{U}^{root_l})\mathbf{B}^{root}. \tag{4.13}$$

Substituting (4.13) into (4.11) leads to

$$\begin{aligned} g(\mu) &= \eta \\ &= \langle (\mathbf{U}^{root_r} \otimes \mathbf{U}^{root_l})\mathbf{B}^{root},\ vec(\mathcal{X}) \rangle + \beta^{\mathsf{T}}\mathbf{z} + b. \end{aligned} \tag{4.14}$$

For other inner nodes $t \in \mathcal{N}(\mathcal{T})$, we then recursively process the nestedness relation by replacing the $\mathbf{U}^t$ with its corresponding transfer matrix $\mathbf{B}^t$ and its children $\mathbf{U}^{t_l}$ and $\mathbf{U}^{t_r}$ from the top of tree to the bottom tree. This procedure stops when we reach all the leaf nodes $t \in \mathcal{L}(\mathcal{T})$.

By exploiting the mixed-product property of Kronecker product, the inner product part in (4.14) is equivalent to

$$\begin{aligned} &\langle vec(\mathcal{B}),\, vec(\mathcal{X}) \rangle = \\ &\qquad \langle (\bigotimes_{t \in \mathcal{L}(\mathcal{T})} \mathbf{U}^t)(\bigotimes_{t \in \mathcal{L}(\mathcal{T})^{L-1}} \mathbf{I}^t \otimes \bigotimes_{t \in \mathcal{N}(\mathcal{T})^{L-1}} \mathbf{B}^t) \cdots (\bigotimes_{t \in \mathcal{T}^l} \mathbf{B}^t) \cdots (\mathbf{B}^{root}),\, vec(\mathcal{X}) \rangle, \end{aligned} \tag{4.15}$$

where the level l of the tree $\mathcal{T}$ is

$$\mathcal{T}^l := \{ t \in \mathcal{T} : level(t) = l \} \quad \text{with} \quad 1 \le l \le L \tag{4.16}$$

which denotes the set of all nodes having a distance of exactly l to the root. Particularly, L refers to the leaf level. We also use $\mathcal{L}(\mathcal{T})^l$ ($\mathcal{N}(\mathcal{T})^l$) to represent the set of leaf nodes (inner nodes) in the level l. For a balanced binary tree, the leaf nodes may only appear in the highest or second highest levels, the number of leaf factor matrices to be estimated is D and the number of interior transfer matrices is equal to D-1. Notice that $\{\mathbf{I}^t \in \mathbb{R}^{r_t \times r_t}\}_{t \in \mathcal{L}(\mathcal{T})^{L-1}}$ are the identity matrices, which are used to take the place of singleton basis factor matrices $\{\mathbf{U}^t\}_{t \in \mathcal{L}(\mathcal{T})^{L-1}}$ which are used to be in the second highest level. By doing so, all the singleton basis factor matrices $\mathbf{U}^t$ are "pushed" to the highest level.

Finally, the resulting $\mathcal{H}$-Tucker regression model becomes

$$\begin{aligned} g(\mu) &= \eta \\ &= \langle vec(\mathcal{B}), vec(\mathcal{X}) \rangle + \beta^{\mathsf{T}}\mathbf{z} + b \\ &= \langle (\bigotimes_{t \in \mathcal{L}(\mathcal{T})} \mathbf{U}^t)(\bigotimes_{t \in \mathcal{L}(\mathcal{T})^{L-1}} \mathbf{I}^t \otimes \bigotimes_{t \in \mathcal{N}(\mathcal{T})^{L-1}} \mathbf{B}^t) \cdots (\bigotimes_{t \in \mathcal{T}^l} \mathbf{B}^t) \cdots (\mathbf{B}^{root}),\, vec(\mathcal{X}) \rangle + \beta^{\mathsf{T}}\mathbf{z} + b, \end{aligned} \tag{4.17}$$

Similar to the argument of the Tucker based model [Li et al., 2013], the number of free parameters in $\mathcal{H}$-Tucker model can be obtained as

$$\sum_{t\in\mathcal{L}(\mathcal{T})} I_t r_t + \sum_{t\in\mathcal{N}(\mathcal{T})} r_{t_r} r_{t_l} r_t - \sum_{t\in\mathcal{T}\setminus root} r_t^2, \tag{4.18}$$

where the last term $\sum_{t\in\mathcal{T}\setminus root} r_t^2$ is used for the propose of nonsingular transformation indeterminacy [Li et al., 2013].

We should stress the fact that the number of parameters in coefficient tensor in $\mathcal{H}$-Tucker regression model (4.17) is free from the exponential growth. Indeed, the total number of parameters in $\mathcal{H}$-Tucker regression model is linear in order D. For large order D, the number of parameters in $\mathcal{H}$-Tucker regression model is more than that of in the CP model but far less than that of in the Tucker model. Such reduction of parameters is essentially useful for neuroimaging data analysis, since only a limited number of images are available in most of datasets. Thus, $\mathcal{H}$-Tucker regression model could be more beneficial in terms of robust and efficient estimation and prediction.

Furthermore, like Tucker tensor regression model, $\mathcal{H}$-Tucker model allows us to have distinct rank r_t at each node of the dimension tree $\mathcal{T}$. This flexibility of $\mathcal{H}$-Tucker retains all the advantages of Tucker regression model as explained in Section 4.1.

In addition, $\mathcal{H}$-Tucker regression model is a SVD-based method and the binary tree structure enables to compute all low rank approximations of coefficient tensor by means of standard linear algebra tools.

4.4 Parameter Estimation

In this section, a highly scalable algorithm is presented using the maximum likelihood estimation (MLE) (see Appendix A.5) for the $\mathcal{H}$-Tucker regression model with linear systems part of GLM described by formula (4.17).

Given a collection of input-output data pairs $\mathcal{D} = \{(\mathcal{X}_n \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_D}, y_n \in \mathbb{R})\}_{n=1}^N$, e.g., a set of fMRI images $\mathcal{X}_n$ and the associated clinical outcomes y_n. The tensor coefficient $\mathcal{B} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_D}$ in (4.11) can be estimated by solving the following optimization problem

$$\begin{aligned} &\max_{\mathcal{B}} \ell(\mathcal{B};\mathcal{D}) \\ &\text{s.t.} \quad \mathcal{B} \in \mathcal{H}\text{-Tucker}((r_t)_{t\in\mathcal{T}}) \quad r_t \le R \quad \forall t \in \mathcal{T}, \end{aligned} \tag{4.19}$$

where ℓ is the concave log-likelihood function. We assume $\mathcal{B}$ follows a $\mathcal{H}$-Tucker format with a low-rank constraint. Based on the model (4.17), the optimization problem reflected by (4.19)

can be further written as

$$\begin{aligned} \max_{\mathbf{U}^t, \mathbf{B}^t, \beta, b} \; & \ell(\{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}, \{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}, \beta, b; \mathcal{D}) \\ \text{s.t.} \;\; & r_t \leq R \;\; \forall t \in \mathcal{T} \end{aligned} \quad , \tag{4.20}$$

In particular, it is well known that if we adopt GLM with a Gaussian response, the problem reflected by (4.20) is equivalent to the least square minimization problem given by

$$\begin{aligned} \min_{\{\mathbf{U}^t, \mathbf{B}^t, \beta, b\}} \{ \sum_{n=1}^{N} \| y_n - \langle (\bigotimes_{t\in\mathcal{L}(\mathcal{T})} \mathbf{U}^t)(\bigotimes_{t\in\mathcal{L}(\mathcal{T})^{L-1}} \mathbf{I}^t \otimes \bigotimes_{t\in\mathcal{N}(\mathcal{T})^{L-1}} \mathbf{B}^t) \\ \cdots (\bigotimes_{t\in\mathcal{T}^l} \mathbf{B}^t) \cdots (\mathbf{B}^{root}), \, vec(\mathcal{X}) \rangle - \beta^{\mathsf{T}} \mathbf{z} - b \|^2 \} \\ \text{s.t.} \;\; r_t \leq R \;\; \forall t \in \mathcal{T} \end{aligned} \quad . \tag{4.21}$$

It is worth noticing that in (4.20) or (4.21), the linear systematic part is only linear in each $\mathbf{U}^t$ and each $\mathbf{B}^t$ *separately*. Hence, we can then alternately update one basis factor (or transfer) matrix $\mathbf{U}^t$ (or $\mathbf{B}^t$) at a time while keeping the rest of matrices fixed. This is referred by the block relaxation algorithm (BRA) [De Leeuw, 1994; Lange, 2010], and it enables us to break the simultaneous estimation of all parameters into a sequence of low dimensional parameter optimizations using classical GLM [Nelder and Baker, 1972; McCullagh and Nelder, 1989]. Applying this strategy to our model, the estimation of factor matrices proceeds iteratively by sweeping all the nodes of dimension tree $\mathcal{T}$ from bottom to top in a sequential way.

In order to perform the classical GLM, one needs to isolate the component of interest from the rest part. Particularly, for the leaf basis factor matrix $\mathbf{U}^t$ in the leaf level L, the inner product in (4.15) can be rewritten as follows:

$$\langle \mathbf{U}^t \mathbf{L}^L (\bigotimes_{t'\in\mathcal{L}(\mathcal{T})\setminus t} \mathbf{U}^{t'})^{\mathsf{T}}, \mathbf{X}_{(t)} \rangle = \langle \mathbf{U}^t, \mathbf{X}_{(t)} (\bigotimes_{t'\in\mathcal{L}(\mathcal{T})\setminus t} \mathbf{U}^{t'}) (\mathbf{L}^L)^{\mathsf{T}} \rangle, \tag{4.22}$$

where

$$\mathbf{L}^L := (\bigotimes_{t'\in\mathcal{L}(\mathcal{T})^{L-1}} \mathbf{I}^{t'} \otimes \bigotimes_{t'\in\mathcal{N}(\mathcal{T})^{L-1}} \mathbf{B}^{t'}) \cdots (\bigotimes_{t'\in\mathcal{T}^l} \mathbf{B}^{t'}) \cdots (\mathbf{B}^{root}). \tag{4.23}$$

As for the interior node $t \in \mathcal{T}^l$ in intermediate level l, one can individually estimate $\mathbf{B}^t$ with only $r_{t_l} r_{t_r} r_t$ number of parameters by rewriting the inner product in (4.15) as follows:

$$\langle \mathbf{B}^t \mathbf{L}^l (\bigotimes_{t'\in\mathcal{T}^l\setminus t} \mathbf{B}^{t'})^{\mathsf{T}}, \mathbf{H}^l \rangle = \langle \mathbf{B}^t, \mathbf{H}^l (\bigotimes_{t'\in\mathcal{T}^l\setminus t} \mathbf{B}^{t'}) (\mathbf{L}^l)^{\mathsf{T}} \rangle, \tag{4.24}$$

here

$$\mathbf{H}^l := (\bigotimes_{t'\in\mathcal{T}^{l+1}} \mathbf{B}^{t'})^{\mathsf{T}} \cdots (\bigotimes_{t'\in\mathcal{L}(\mathcal{T})} \mathbf{U}^{t'})^{\mathsf{T}} vec(\mathcal{X}) \tag{4.25}$$

and

$$\mathbf{L}^l := (\bigotimes_{t'\in\mathcal{T}^{l-1}} \mathbf{B}^{t'}) (\bigotimes_{t'\in\mathcal{T}^{l-2}} \mathbf{B}^{t'}) \cdots (\mathbf{B}^{root}). \tag{4.26}$$

The isolation of $\mathbf{U}^t$ by (4.22) (or $\mathbf{B}^t$ by (4.24)) is in fact the sequential downsizing of the original data into a smaller and more manageable size.

We iteratively update $\mathbf{U}^t$ and (or) $\mathbf{B}^t$ for each node t from bottom to top and from left to right along each level of $\mathcal{T}$ until the log likelihood defined for classical GLM ceases to increase. The outline of the procedure is shown in Algorithm 4. Specifically, we first update the regular vector coefficient β using the previous factor and transfer matrices in line 3. Note that the intercept b is absorbed into β. Next, we estimate in lines 4-6 each basis factor matrix $\mathbf{U}^t$ from left to right in the leaf level by fixing the transfer matrices and other factor matrices of the tree $\mathcal{T}$. Subsequently, we update the transfer matrix $\mathbf{B}^t$ of each interior node again by keeping the factor matrices and other transfer matrices in the tree $\mathcal{T}$ fixed (lines 8-10).

To summarize, it has been proven that the block relaxation algorithm (BRA) monotonically increases the likelihood, and the estimation is guaranteed to converge whenever likelihood function is bounded [De Leeuw, 1994; Lange, 2010]. As an extension from Tucker regression model, our algorithm enjoys the same convergence properties as that of [Li et al., 2013], which is summarized as

Proposition 3: Suppose the following assumptions are satisfied for $\ell(\theta)$ with parameter $\theta := \{\beta, \{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}, \{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}\}$ in Algorithm 4

1. $\ell(\theta)$ is continuous, bounded above and with compact set, i.e., $\{\theta : \ell(\theta) \geq \ell(\theta)^{[0]}\}$;
2. $\ell(\theta)$ is strictly concave with respect to the update of each factor matrix;
3. $\ell(\theta)$ has a set of isolated stationary points [Li et al., 2013].

Then, we have

1. **global convergence** the sequence $\theta^{[m]}$ converge to a stationary point of $\ell(\theta)$;
2. **local convergence** the sequence $\theta^{[m]}$ are locally attracted to a strict local maximum $\theta^{[\infty]}$ if $\theta^{[0]}$ sufficiently close to $\theta^{[\infty]}$.

See Appendix B.1 for a sketch of the proof.

Following the similar arguments in [Li et al., 2013], Algorithm 4 reaches a stationary point where the likelihood stops increasing. In practice, however, we can always expect Algorithm 4 to converge to at least a local maximum. Therefore, one should run this algorithm multiple times in order to increase the chance of finding the global optimal estimate.

In tensor regression, regularization is essential when the number of parameters far exceeds the sample size, otherwise, the regression problem becomes ill-posed and the solution tends to be overfitting. Although our model is compact, we can further improve the learning performance by imposing sparsity penalty, i.e., l_1-norm, on the transfer matrix $\{\mathbf{B}_t\}_{t\in\mathcal{N}(\mathcal{T})}$. Specifically, we

instead maximize the penalized likelihood function ℓ when estimating the transfer matrix $\mathbf{B}_t$

$$\max_{\mathbf{B}_t} \ell(\alpha^{[m+1]}), \{\mathbf{U}_{t'}\}_{t'\in\mathcal{L}(\mathcal{T})}^{[m+1]}, \{\mathbf{B}_{t'}\}_{t'<t,t'\in\mathcal{N}(\mathcal{T})}^{[m+1]}, \mathbf{B}_t^{[m]}, \{\mathbf{B}_{t'}\}_{t'>t,t'\in\mathcal{N}(\mathcal{T})}^{[m]}) + \lambda \sum_{r_{t_l},r_{t_r},r_t} |b_{r_{t_l},r_{t_r},r_t}|, \tag{4.27}$$

where $b_{r_{t_l},r_{t_r},r_t}$ corresponds to each entry of $\mathbf{B}_t$, λ is the penalty tuning parameter. This form of $l1$-norm regularization causes many coefficients in $\mathbf{B}_t$ to be sufficiently small.

Algorithm 4 Block Relaxation Algorithm (BRA) for $\mathcal{H}$-Tucker Tensor Regression

Input: N input-output data pairs $\mathcal{D} = \{(\mathcal{X}_n \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_D}, y_n \in \mathbb{R})\}_{n=1}^N$
Output: β, $\{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}$, $\{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}$
Initialize $\beta^{[0]}$, $\{\mathbf{U}^{t[0]}\}_{t\in\mathcal{L}(\mathcal{T})}$, $\{\mathbf{B}^{t[0]}\}_{t\in\mathcal{N}(\mathcal{T})}$
repeat
  $\beta^{[m+1]} = \arg\max_\beta \ell(\beta^{[m]}, \{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}^{[m]}, \{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}^{[m]})$
  for each $t \in \mathcal{L}(\mathcal{T})$ **do**
    $\mathbf{U}^{t[m+1]} = \arg\max_{\mathbf{U}^t} \ell(\beta^{[m+1]}, \{\mathbf{U}^{t'}\}_{t'<t,t'\in\mathcal{L}(\mathcal{T})}^{[m+1]}, \mathbf{U}^{t[m]}, \{\mathbf{U}^{t'}\}_{t'>t,t'\in\mathcal{L}(\mathcal{T})}^{[m]}, \{\mathbf{B}^{t'}\}_{t'\in\mathcal{N}(\mathcal{T})}^{[m]})$
  end for
  for $l = L-1, ..., 1$ **do**
    for each $t \in \mathcal{N}(\mathcal{T})^l$ **do**
      $\mathbf{B}^{t[m+1]} = \arg\max_{\mathbf{B}^t} \ell(\beta^{[m+1]}, \{\mathbf{U}^{t'}\}_{t'\in\mathcal{L}(\mathcal{T})}^{[m+1]}, \{\mathbf{B}^{t'}\}_{t'<t,t'\in\mathcal{N}(\mathcal{T})}^{[m+1]}, \mathbf{B}^{t[m]}, \{\mathbf{B}^{t'}\}_{t'>t,t'\in\mathcal{N}(\mathcal{T})}^{[m]})$
    end for
  end for
until $\|\ell^{[m]} - \ell^{[m+1]}\| < \varepsilon$

4.5 Experimental Results

4.5.1 Simulations on Synthetic Data

In this simulation using synthetic data, we were interested in how the sample size influences the estimation accuracy of the proposed model. In this respect, we first generate the true signal tensor $\mathcal{B}$ using Tucker format, where the core tensor $\mathcal{G}$ and factor matrices $\mathbf{U}$ are randomly drawn from standard normal distributions. Here, we test two setups of the dimension of $\mathcal{B}$, which is fixed at 16 for the 4-order tensor and 9 for the 5-order tensor, respectively. The dimension of $\mathcal{G}$ is set as 4 for the 4-order tensor and 3 for the 5-order tensor. We set the true vector signal β as $\mathbf{1}$ of length 5 for both cases. Subsequently, the regular predictor vector $\mathbf{z} \in \mathbb{R}^5$ and tensor predictor $\mathcal{X} \in \mathbb{R}^{16\times16\times16\times16}$ (or $\mathcal{X} \in \mathbb{R}^{9\times9\times9\times9\times9}$ for the 5-order case) are all generated from independent standard normals. Finally, the output samples are produced by

$$y = \langle \mathcal{B}, \mathcal{X} \rangle + \beta^\mathsf{T}\mathbf{z} + \varepsilon, \tag{4.28}$$

where $\varepsilon \sim \mathcal{N}(0, 1)$ is an additive noise.

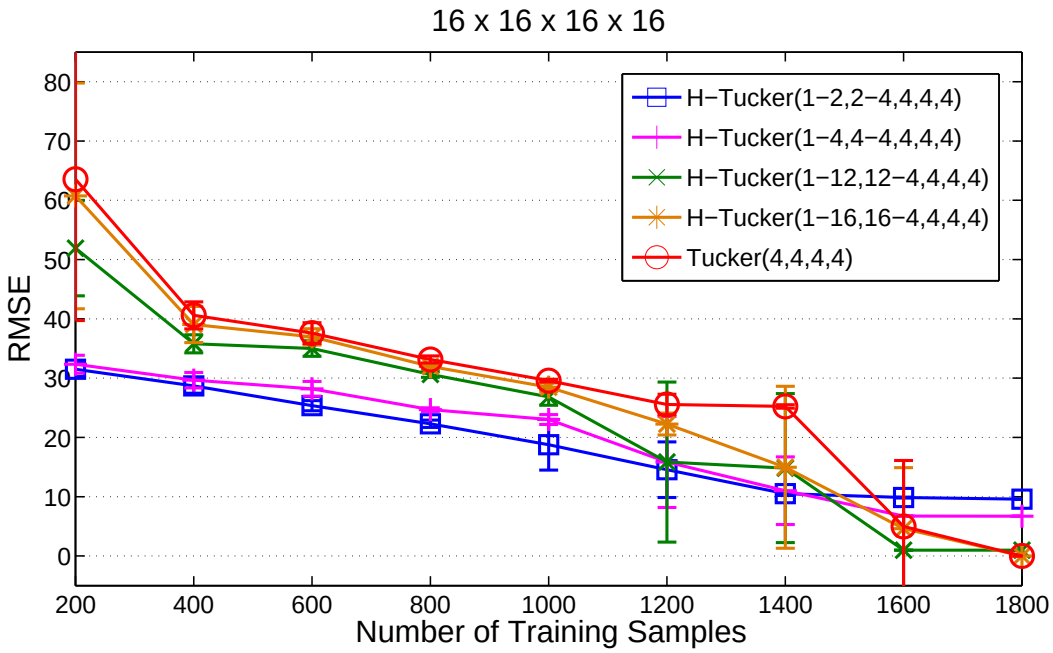


FIGURE 4.3 – Performance comparison vs. number of samples for case of 4-order tensor.

For comparison, we investigate the deviation between the learned $\hat{\mathcal{B}}$ and the true $\mathcal{B}$ for an increasing number of samples n in terms of root mean square error (RMSE). Note that the ht-ranks at leaf singletons are chosen to be the same as d-ranks of the corresponding modes. Moreover, the total number of free parameters of $\mathcal{H}$-Tucker model is less than or equal to Tucker's, e.g., in accord with d-ranks (4,4,4,4), ht-ranks can be (1-2,2-4,4,4,4) or (1-12,12-4,4,4,4) etc. We repeat the run for 50 times.

As shown in Figure 4.3 and Figure 4.4, it is straightforward to see the RMSE of $\mathcal{H}$-Tucker model of all settings progressively decreases as the sample size increases from 200 to 1800, implying the stability of the proposed method. When the sample size is relatively small, the $\mathcal{H}$-Tucker model of lower ht-rank obviously outperforms the Tucker model. For example, the Tucker model of rank (3,3,3,3,3) results in 333 free parameters. On the contrary, $\mathcal{H}$-Tucker model of ht-rank (1-2,2-2,3,3,3-3,3) has 130 free parameters. With only 800 training samples, the RMSE of learned $\mathcal{H}$-Tucker model is 10.3, which is evidently superior to that 21.9 of Tucker. This is the case especially for higher order data, since the parameter size of core tensor expands exponentially in d. This result indicates the $\mathcal{H}$-Tucker is more compact than Tucker, and scales better to higher order data.

For larger n, we can boost the learning performance by increasing the ht-rank adjusted to the sample size. If n is big enough, we can perfectly recover the true signal $\mathcal{B}$ with the same number of free parameters as Tucker's. In this case, the ht-rank we select is (1-9,9-9,3,3,3-3,3), resulting in the exact 333 free parameters. Though the true signal is generated from the Tucker

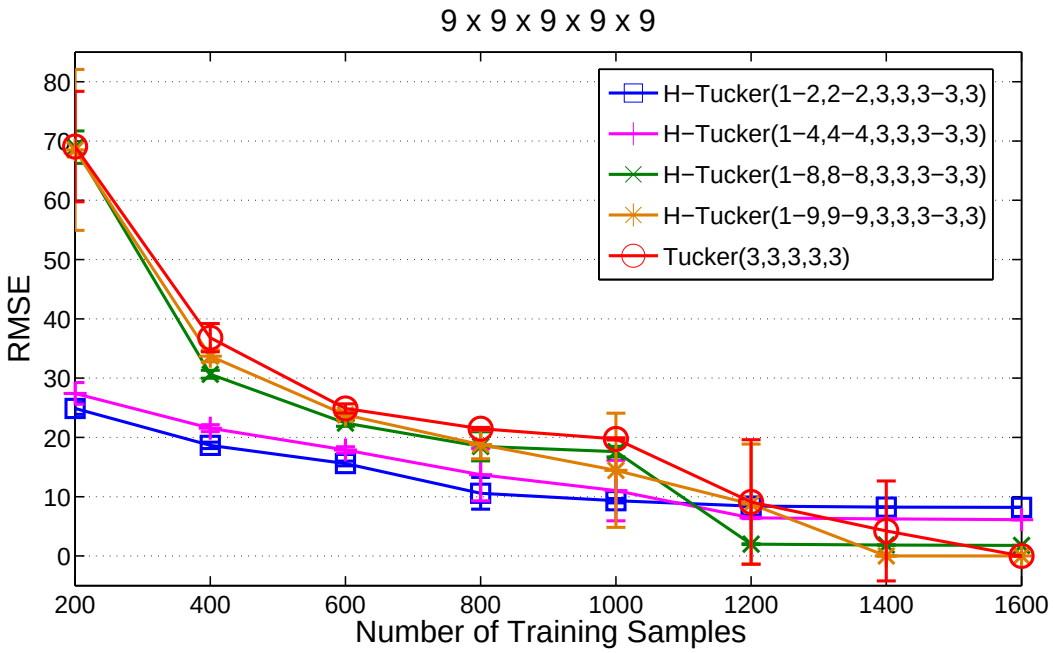


FIGURE 4.4 – Performance comparison vs. number of samples for case of 5-order tensor.

structure, we observe the result obtained by our method is better than Tucker model. When n is greater than 1400 for $\mathcal{H}$-Tucker (or 1600 for Tucker), the RMSE is almost zero. Comparing to Tucker model, the $\mathcal{H}$-Tucker is more flexible to efficiently represent the core tensor $\mathcal{G}$ by allowing distinct ranks in the intermediate levels of the tree structure. It can thereby achieve better estimation accuracy while requiring a small number of free parameters given limited number of samples.

4.5.2 Experiments on ADHD Brain Imaging Dataset

We also validated our model on the real-life attention deficit hyperactivity disorder (ADHD) brain imaging data. The ADHD data is freely available from the Neuro Bureau and is preprocessed according to the standard Burner pipeline, resulting in T1 images of size $121 \times 145 \times 121$ for each subject [ADHD, 2014]. For our experiment, we obtain 761 training subjects and 169 testing subjects after removing those with missing observations. We treat each MRI image represented by a 3-order tensor $\mathcal{X}$ as the image predictor, and the corresponding response y is the binary diagnosis outcome. Additionally, the regular vector predictor $\mathbf{z}$, namely individual's gender, age and handiness, is also included. To be consistent with the experiment settings of Tucker regression model, the whole dataset remains non-normalized in this experiment.

We compare our model with the classical Tucker regression model which is regarded as the state of the art and has been proved to be superior to the CP based modeling in [Li et al., 2013]. Toward this purpose, we downsize the original image into different sizes using the Daubechies

TABLE 4.1 – Performance comparison for the misclassification error of $\mathcal{H}$-Tucker regression and Tucker regression model on ADHD data.

Dimensions	Methods	Rank	Free Param.	Misclassi. Error	
				Reg	Non-Reg
$17 \times 20 \times 16$	$\mathcal{H}$-Tucker Regression	(1-4,3-2,2)	117	**0.296**	**0.302**
		(1-3,3-3,3)	159	**0.302**	**0.308**
		(1-3,4-3,3)	170	**0.296**	**0.302**
	Tucker Regression	(2,2,3)	117	0.308	0.331
		(3,3,3)	159	0.308	0.320
		(3,3,4)	177	0.302	0.320
$12 \times 14 \times 12$	$\mathcal{H}$-Tucker Regression	(1-4,3-2,2)	83	**0.284**	**0.308**
		(1-4,3-3,3)	114	**0.278**	**0.290**
		(1-3,4-3,3)	122	**0.284**	**0.302**
	Tucker Regression	(2,2,3)	83	0.290	0.314
		(3,3,3)	114	0.296	0.314
		(3,3,4)	128	0.290	0.314

D4 wavelet transformation [Daubechies, 1992]. The reduced dimensions we consider are $12 \times 14 \times 12$ and $17 \times 20 \times 16$. We also experiment with distinct ranks to see how the performance can be affected by the free parameter size. The performance is gauged by the misclassification error which is the percentage of misclassified labels in the test data. Note carefully that we follow the same guideline for choosing the ranks as previous section, and the sample size is approximately 4.5-9 times as large as number of free parameters. In the case of the regularization, we tune the penalty parameter λ based on the Bayesian information criterion (BIC) [Schwarz, 1978].

Table 4.1 shows the results of misclassification errors on the testing set with respect to different setups of dimensions and ranks. As expected, the $\mathcal{H}$-Tucker model outperforms the Tucker model in all cases with smaller or equal number of free parameters. We also observe that the non-regularized $\mathcal{H}$-Tucker model is already better or comparable to the regularized version of Tucker model. It is worth noticing that the ADHD-200 is a really hard task, the error rate around 0.28 achieved by the proposed method is fairly attractive especially when the original data is unbalanced.

4.6 Conclusion

In this chapter we have first motivated the $\mathcal{H}$-Tucker decomposition and then introduced a novel generalized linear tensor regression model based on it. The experiment results show that $\mathcal{H}$-Tucker is a highly compact, flexible and scalable model, which has better low-rank approximation using only a small number of parameters. This key advantage makes it the perfect model for the applications, such as MRI, fMRI analysis, where data is quite high dimensional but with quite restricted sample size.

In the next chapter, we will extend the tensor Gaussian process model, with tensorial input and scalar output, to the online local tensor Gaussian process model that can handle infinite time-dependent tensor streams.

Chapitre 5

Online Local Gaussian Process for Tensor Regression

In this chapter, a computationally-efficient online tensor GP framework [Hou et al., 2015] is proposed by introducing online local Gaussian process (OLGP) for tensor-variate regression. In this context, two efficient search strategies, namely input-based and input-output-based search, are presented to find the local GP experts so as to maintain the predictive accuracy. Finally, the effectiveness of tensor OLGP is demonstrated on a large-scale tensor regression task, i.e., limb motion reconstruction using brain signal [Chao et al., 2010].

5.1 Introduction

Linear tensor-variate regression approaches have been extensively studied over the past few years. This is attributed to their simplicity in modeling as well as the reasonable performance obtained on many high-order datasets. Being as multilinear models, however, they may fail to capture the nonlinear dependency structure that might exist between the input and output in many real-world regression tasks.

Recently, some nonlinear tensor regression approaches have been explored to deal with this nonlinearity. One of them called tensor-variate Gaussian process regression (tensor GP) proposed in [Zhao et al., 2014] is quite promising. This is because, instead of linear assumptions as is the case in [Zhou et al., 2013; Li et al., 2013; Hou and Chaib-draa, 2015], tensor GP is able to flexibly model the nonlinearity of the tensorial data by using the powerful Bayesian nonparametric Gaussian process (GP) [Rasmussen and Williams, 2005; Bishop, 2006; Wang, 2014]. However, the computation load caused by GP, $O(N^3)$, often makes tensor GP computationally prohibitive in practice, since the number of the training points N is often required to be quite large in practice (where the data points in high-order tensor space tend to be sparse due to the curse of dimensionality [Oommen et al., 2008]).

Compared to tensor GP in [Zhao et al., 2014], the new tensor regression model named tensor OLGP [Hou et al., 2015] introduced in this chapter takes advantage of online local Gaussian process (OLGP) [Nguyen-Tuong et al., 2009; Urtasun and Darrell, 2008] by assigning the data points to a number of the small-sized GP experts in an online fashion, thus significantly reducing the computation burden for large data sets.

5.2 Tensor GP Regression Review

In a standard tensor-variate regression task, we are given a training set $\mathcal{D} = \{(\mathcal{X}_n, y_n)\}_{n=1}^N$ where the scalar output $y_n \in \mathbb{R}$ is generated by a nonlinear function $f(\mathcal{X}_n)$ of the D-order tensor input $\mathcal{X}_n \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ with an additive Gaussian noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$

$$y_n = f(\mathcal{X}_n) + \epsilon_n. \tag{5.1}$$

For simplicity, we concatenate all the tensor inputs into a $(D+1)$-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times I_1 \times \cdots \times I_D}$ and put all the outputs into a vector $\mathbf{y} = [y_1, ..., y_N]^T$.

Recall that in Section 3.1.2, the tensor GP approach [Zhao et al., 2014] is based on the idea that the latent function f in (5.1) can be modeled by a GP, i.e.,

$$f(\mathcal{X}) \sim \mathcal{GP}(m(\mathcal{X}), k(\mathcal{X}, \mathcal{X}')|\theta), \tag{5.2}$$

where $m(\mathcal{X})$ is the mean function, $k(\mathcal{X}, \mathcal{X}')$ is the covariance function and θ is the associated hyperparameter vector. In this work, we follow a standard GP setting in [Rasmussen and Williams, 2005] where $m(\mathcal{X}) = 0$. For $k(\mathcal{X}, \mathcal{X}')$, we follow [Zhao et al., 2014] and adopt the following product probabilistic kernel:

$$k(\mathcal{X}, \mathcal{X}') = \alpha^2 \prod_{d=1}^{D} \exp\left(\frac{KL(p(\mathbf{x}|\Omega_d^{\mathcal{X}}) \parallel q(\mathbf{x}'|\Omega_d^{\mathcal{X}'}))}{-2\beta_d^2}\right), \tag{5.3}$$

where α is the the magnitude hyperparameter, and β_d denotes the d-mode length-scales hyperparameter. The distributions p and q in the Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951] are characterized by the hyperparameter Ω_d (i.e., for multivariate Gaussian $\Omega_d = \{\mu_d, \Sigma_d\}$ with μ_d being the mean vector and Σ_d being the covariance matrix) which can be estimated from the d-mode unfolding matrix $\mathbf{X}_{(d)}$ of tensor $\mathcal{X}$ by treating each $\mathbf{X}_{(d)}$ as a generative model with I_d number of variables and $I_1 \cdots I_{d-1} I_{d+1} \cdots I_D$ number of observations.

The goal of a tensor GP regression, as referred by (5.1), aims to infer the predictive distribution of the latent function value $f(\mathcal{X}_*) = f_*$ at a new test point $\mathcal{X}_*$ given the training data $\mathcal{D}$. According to the definition of GP, we can obtain that any finite number of latent function values at the tensorial inputs are Gaussian distributed [Rasmussen and Williams, 2005]. Consequentially, the joint distribution

$$p(f_*, \mathbf{y}|\mathcal{X}_*, \boldsymbol{\mathcal{X}}, \theta, \sigma^2) \tag{5.4}$$

is Gaussian. Moreover, based on the conditional property of Gaussian distribution, the predictive distribution

$$p(f_*|\mathcal{X}_*, \boldsymbol{\mathcal{X}}, \mathbf{y}, \theta, \sigma^2) = \mathcal{N}(m_*, \sigma_*^2) \tag{5.5}$$

is also Gaussian with

$$\begin{aligned} m_* &= \mathbf{k}_{\mathcal{X}}^{\mathsf{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \\ \sigma_*^2 &= k_* - \mathbf{k}_{\mathcal{X}}^{\mathsf{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_{\mathcal{X}}, \end{aligned} \tag{5.6}$$

where $k_* = k(\mathcal{X}_*, \mathcal{X}_*)$ and $\mathbf{k}_{\mathcal{X}} = k(\mathcal{X}_*, \boldsymbol{\mathcal{X}})$. However, as we mentioned, the computational complexity of GP, which is $O(N^3)$, often makes tensor GP computationally expensive because N is usually required to be quite large to achieve a reliable result for a tensor-variate regression task [Oommen et al., 2008].

In the following section, a computationally efficient tensor GP framework is proposed, where a fast data processing mechanism, inspired by online local Gaussian process (OLGP) as proposed in [Nguyen-Tuong et al., 2009; Urtasun and Darrell, 2008], is designed for tensor-variate regression (tensor OLGP).

5.3 Tensor OLGP Regression

In order to deal with large-scale tensor regression tasks, our tensor OLGP regression approach consists of the following two stages:

- **Stage 1 (GP Experts Construction)** : Using the covariance function of GP (5.3) as a similarity measurement to sequentially partition the training data points into a number of small-sized experts.
- **Stage 2 (Local Prediction)** : Finding a fixed-number of local GP experts to make predictions (for given test tensorial inputs) with a Gaussian mixture.

We now detail these two stages and then see the computational complexity behind our tensor OLGP.

5.3.1 GP Experts Construction

To achieve the computation efficiency, we propose to use the covariance function of GP referred by (5.3) (as a similarity measurement) to sequentially allocate the tensorial data into a collection of local experts. The whole mechanism to construct GP experts is shown in Algorithm 5. Specifically, when a new training data pair $\{\mathcal{X}_{new}, y_{new}\}$ arrives, we first calculate the similarity between $\mathcal{X}_{new}$ and the center of each local expert $\mathcal{C}_k$ using the probabilistic tensor kernel (line 3). Subsequently, we choose the closest expert t whose center has the highest similarity measure sim_t with $\mathcal{X}_{new}$ according to (5.3) (line 5). If sim_t is greater than a predefined threshold w_{gen}, then we insert this new data pair into that local expert t, and update the kernel matrix of expert t accordingly (line 7-12). Otherwise, we defined $\mathcal{X}_{new}$ as the center of

new expert $R+1$ (R is the total number of local experts), and thus initialize a new kernel matrix (line 14-16).

In this stage of GP expert construction, we process the large data set into a number of small-sized experts in an online fashion, thus naturally speed up the computation efficiency.

Algorithm 5 GP Experts Construction

1: **Input**: new tensor data pair $\{\mathcal{X}_{new}, y_{new}\}$, expert centers $\{\mathcal{C}_k\}_{k=1}^{R}$, threshold w_{gen}
2: **for** $k = 1$ to number of local experts R **do**
3: Compute the similarity to the kth expert using probabilistic tensor kernel function (5.3):
$w_k = k(\mathcal{X}_{new}, \mathcal{C}_k)$
4: **end for**
5: Choose the nearest local expert t: $sim_t = \max(w_k)$
6: **if** $sim_t > w_{gen}$ **then**
7: Insert $\{\mathcal{X}_{new}, y_{new}\}$ to the nearest local expert t:
8: $\boldsymbol{\mathcal{X}}_t = [\boldsymbol{\mathcal{X}}_t, \mathcal{X}_{new}]$, $\mathbf{y}_t = [\mathbf{y}_t, y_{new}]$
9: **if** maximum number of data points is reached **then**
10: delete another point by permutation
11: **end if**
12: Update the corresponding kernel matrix $\mathbf{K}_t$ by computing the kernel vector $\mathbf{k}_t(\mathcal{X}_{new}, \boldsymbol{\mathcal{X}}_t)$ for $\mathcal{X}_{new}$
13: **else**
14: Create a new expert:
15: $\mathcal{C}_{R+1} \doteq \mathcal{X}_{new}$
$\boldsymbol{\mathcal{X}}_{R+1} = [\mathcal{X}_{new}]$, $\mathbf{y}_{R+1} = [y_{new}]$
16: Initialize the new kernel matrix $\mathbf{K}_{R+1}$
17: **end if**

5.3.2 Local Prediction

Once partitioning the data sets into a number of small-sized experts, we then propose two local search strategies for predicting the new test point $\mathcal{X}_*$ in order to take into account the tradeoff between accuracy and efficiency: input-based searching strategy and input-output-based searching strategy.

A. Input-based Searching Strategy

This first strategy exploits M nearest local experts to make the prediction. These M local experts should have the highest similarities with $\mathcal{X}_*$ among all the local experts according to the kernel function defined in (5.3). Then the similarity measure $w_k = k(\mathcal{X}_*, \mathcal{C}_k)$ from the input space between the test point $\mathcal{X}_*$ and the expert center $\mathcal{C}_k$ can be used as the weight of local expert k. Hence, the resulting prediction $\hat{y}_*$ can be formulated as the weighted combination

from each local prediction

$$\bar{y}_k = \mathbf{k}_k(\mathcal{X}_*, \boldsymbol{\mathcal{X}}_k)^\mathsf{T}(\mathbf{K}_k + \sigma^2\mathbf{I})^{-1}\mathbf{y}_k \tag{5.7}$$

as follows:

$$\hat{y} = \frac{\sum_{k=1}^{M} w_k \bar{y}_k}{\sum_{k=1}^{M} w_k}. \tag{5.8}$$

B. Input-output-based Searching Strategy

In the regression task the latent function is a mapping between input and output, hence we propose to explore the experts which do not only consider the input space as in the previous strategy but also the output space.

This strategy works as follows: Given a test point $\mathcal{X}_*$, we start, in the first step, by finding its nearest local expert $\mathcal{C}_k \doteq \{\mathcal{X}_{\mathcal{C}_k}, y_{\mathcal{C}_k}\}$ from the input $\mathcal{X}$-space using tensor kernel function defined in (5.3), where the pair $\{\mathcal{X}_{\mathcal{C}_k}, y_{\mathcal{C}_k}\}$ is some data pair coming from local expert k.

With the nearest local expert $\mathcal{C}_k \doteq \{\mathcal{X}_{\mathcal{C}_k}, y_{\mathcal{C}_k}\}$ in hand, we aim to find all M candidate experts which are closest to $\mathcal{C}_k$ in output y-space in the second step. More specifically, we search for M local experts $\{\mathcal{C}_m \doteq \{\mathcal{X}_{\mathcal{C}_m}, y_{\mathcal{C}_m}\}\}_{m=1}^{M}$ that are being closest to $y_{\mathcal{C}_k}$ in y-space among all the local expert centers. In other words, this step intends to find M smallest Euclidian distances between $y_{\mathcal{C}_m}$ and $y_{\mathcal{C}_k}$. Then, from these already found $\{y_{\mathcal{C}_m}\}_{m=1}^{M}$, we can easily mark their corresponding local experts $\{\mathcal{C}_m\}_{m=1}^{M}$ as the candidates for prediction. Finally, we use the same weight described in the first strategy to combine the local predictions.

5.3.3 Computational Complexity

Table 5.1 shows the comparisons of overall computational complexity between classical tensor GP and our tensor OLGP. In both methods, we have to evaluate the tensor kernel function defined in (5.3) between any two points to build the kernel matrix $\mathbf{K}$. Such evaluation mainly depends on the estimation of hyperparameter Ω_d whose cost is dominated by the term $\mathcal{O}(I^{D+1})$, with $I = \max\{I_d\}_{d=1}^{D}$.

For standard tensor GP, the computational complexity of learning is $\mathcal{O}(N^3)$, plus the cost of establishing the kernel matrix $\mathcal{O}(N^2 I^{D+1})$. The complexity of prediction requires $\mathcal{O}(NI^{D+1})$ to compute the kernel vector $\mathbf{k}_{\mathcal{X}}(\mathcal{X}_*, \boldsymbol{\mathcal{X}})$, where N is the number of training samples.

TABLE 5.1 – Computational complexity of tensor GP and tensor OLGP using product probabilistic tensor kernel.

	Partitioning + Training	Prediction
tensor GP	$\mathcal{O}(N^2 I^{D+1} + N^3)$	$\mathcal{O}(NI^{D+1} + N^2)$
tensor OLGP	$\mathcal{O}(NRI^{D+1} + NSI^{D+1} + S^3)$	$\mathcal{O}(RI^{D+1} + M(SI^{D+1} + S^2))$

In contrast, the cost of tensor OLGP for learning includes finding the nearest local expert $\mathcal{O}(NRI^{D+1})$ and updating the kernel matrix of that local expert $\mathcal{O}(NSI^{D+1} + S^3)$, where R is the number of local experts and S is the maximum number of data points contained in each local expert. While the computational complexity of prediction arises from finding M nearest neighbours $\mathcal{O}(RI^{D+1})$ and making M local predictions $\mathcal{O}(M(SI^{D+1} + S^2))$. As we can see, the cost of our tensor OLGP for learning is only linear in N comparing to N^3 in the case of standard GP.

5.4 Experimental Results

In this section, we validate our tensor OLGP on a benchmark tensor regression application : the reconstruction of limb movements from monkey's brain signals using Neurotycho food tracking Electrocorticography (ECoG) dataset [Chao et al., 2010]. The input of tensor OLGP is the preprocessed ECoG signal, that is, a 3-order tensor (i.e., *time* $\times$ *frequency* $\times$ *channel*), and its output is the movement distance of the monkey's limb on different markers (shoulder, elbow or hand) along each axis (x, y or z). More specifically, the whole ECoG food tracking task dataset consists of 15 minutes long experiment with motion data sampled at 120Hz.

To illustrate the effectiveness of our approach, we first choose a subsegment of the whole dataset starting from the 2nd minute comprising 10 000 data pairs. We then conduct the experiments by randomly selecting a training set with size of 5000. The rest 5000 is used as the test set. As for the input, the wavelet transformed ECoG data are down-sampled to 5 channels and can thus be written as $\mathcal{X} \in \mathbb{R}^{5\times5\times5}$ for each sample. In our experiment, we choose the motion data corresponding to the shoulder marker along the x-coordinate.

To compare with tensor GP, we perform the evaluation of our tensor OLGP by showing the learning performance in terms of accuracy and efficiency. In particular, tensor x-OLGP and tensor xy-OLGP are the two variants of the tensor OLGP which correspond to the input-based searching strategy and input-output-based searching strategy, respectively. The root mean square error (RMSE) and negative log likelihood (NLL) are the most commonly used metrics for accuracy in Gaussian process. For both RMSE and NLL, the lower values imply

TABLE 5.2 – Performance comparison for the prediction of movement of shoulder marker along x-axis on ECoG data, with data size=10 000.

w_{gen}	Methods	RMSE	NLL	Running Time (s)	
				Training	Testing
0.5	tensor GP	$\mathbf{3.05 \pm 0.16}$	7.26 ± 0.57	1279.1 ± 9.2	2480.6 ± 16.7
	tensor x-OLGP	4.71 ± 0.15	$\mathbf{2.86 \pm 0.10}$	$\mathbf{321.0 \pm 3.9}$	503.5 ± 4.7
	tensor xy-OLGP	4.39 ± 0.18	4.53 ± 0.43	$\mathbf{321.0 \pm 3.9}$	$\mathbf{492.4 \pm 8.3}$
0.6	tensor GP	$\mathbf{3.05 \pm 0.16}$	7.26 ± 0.57	1279.1 ± 9.2	2480.6 ± 16.7
	tensor x-OLGP	4.56 ± 0.14	$\mathbf{2.66 \pm 0.07}$	$\mathbf{511.1 \pm 3.2}$	829.9 ± 6.4
	tensor xy-OLGP	3.82 ± 0.15	4.03 ± 0.41	$\mathbf{511.1 \pm 3.2}$	$\mathbf{822.0 \pm 6.8}$

more accurate results. Unlike RMSE that penalizes the inconsistency only, NLL penalizes both uncertainty and inconsistency [Deisenroth et al., 2009]. The hyperparameters contained in the probabilistic tensor kernel are set to the same values empirically for all the methods. We also manually tune the partitioning threshold parameter w_{gen} whose value balances the trade-off between accuracy and efficiency of the final performance. Here, w_{gen} is fixed to 0.5 and the R is set to 6. We repeated the experiment 10 times.

Table 5.2 shows the RMSE, NLL as well as the running time of all the approaches. As expected, our two tensor OLGP variants (tensor x-OLGP and tensor xy-OLGP) achieve the best results in terms of running time. The RMSE of our tensor OLGP is competitive to tensor GP and the NLL of our method outperforms tensor GP. Although tensor GP is slightly better in RMSE, the nonstationarity in the signal makes GP with high uncertainty, that is why tensor OLGP tries to capture the local structure to get lower uncertainty with a few nearby local experts. In particular, the training time of tensor OLGP, which is 321.0 seconds, is about 4 times as fast as that 1279.1 seconds of tensor GP.

We also observe that the tensor xy-OLGP is slightly better than tensor x-OLGP in RMSE but somehow worse than tensor x-OLGP in NLL. In another setting, we increase w_{gen} to 0.6 and obtain a similar result with an more obvious gap in RMSE between tensor x-OLGP and tensor xy-OLGP. This may be because the M most nearby local experts found by the second strategy tend to make a consistent prediction with the local expert closest to the test point $\mathcal{X}_*$. We should notice, however, that the found experts may be far away from the test point in the input space, leading to an uncertain prediction. This somehow explains why the input-based strategy is better in NLL.

TABLE 5.3 – Performance comparison for the prediction of movement of shoulder marker along x-axis on ECoG data, with data size=36 000 and $w_{gen} = 0.4$.

Methods	RMSE	NLL	Running Time (s)	
			Training	Testing
tensor GP	$\mathbf{3.40 \pm 0.19}$	10.15 ± 0.81	19141.9 ± 163.5	39152.4 ± 230.9
tensor x-OLGP	5.77 ± 0.19	$\mathbf{3.18 \pm 0.12}$	$\mathbf{2819.9 \pm 37.3}$	5135.2 ± 66.3
tensor xy-OLGP	5.62 ± 0.24	4.67 ± 0.48	$\mathbf{2819.9 \pm 37.3}$	$\mathbf{4503.0 \pm 48.1}$

From the Figure 5.1, it is straightforward to see that both RMSE and NLL of the test set decrease gradually when the proportion of data used in training increases from 1000 to 5000, implying the stability of the proposed methods. Comparing to the polynomial growth in $\mathcal{O}(N^3)$ of tensor GP, the learning time of tensor OLGP in Figure 5.2 only grows linearly in $\mathcal{O}(N)$.

We were also interested on how the performance can be affected by distinct number of local experts R. In this context, w_{gen} is set to 0.3, and R is listed from 2 to 12. These results are demonstrated in Figure 5.3. Observe that both RMSE and NLL reduce significantly before reaching their optimal values when R goes up. The result reflects that fact that a certain

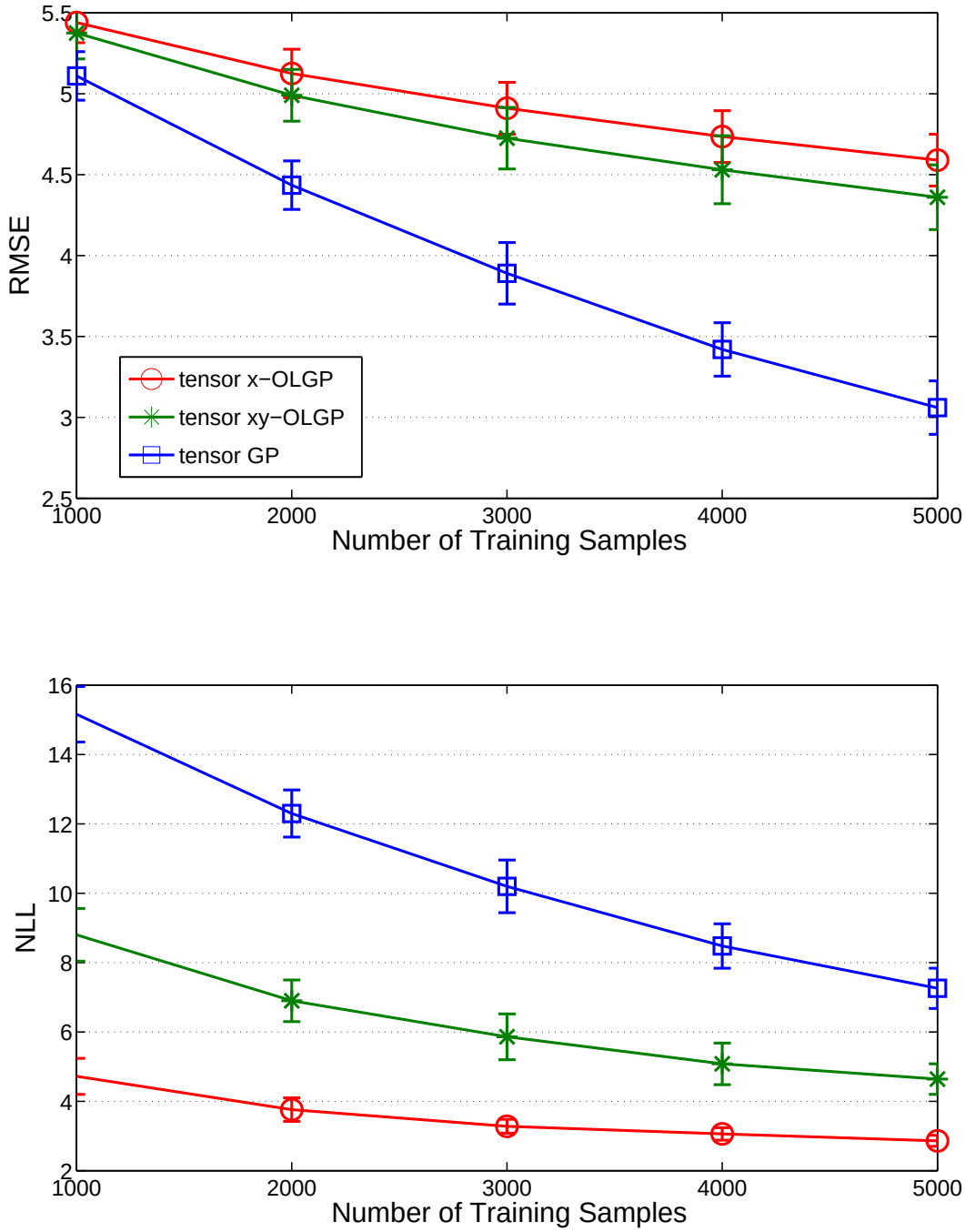


FIGURE 5.1 – RMSE (top) and NLL (bottom) vs. number of training samples, $w_{gen} = 0.5$, $R = 6$.

number of the most nearby local experts are required to guarantee a more accurate and reliable prediction. The further increase of the R brings no improvement in performance when the number of local experts becomes saturated.

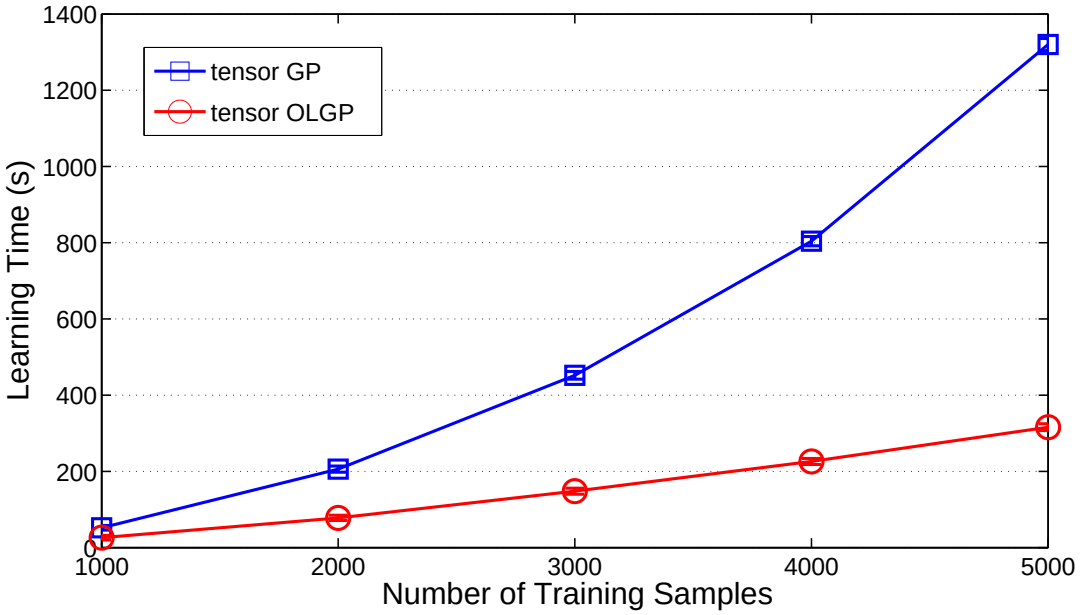


FIGURE 5.2 – Learning time vs. number of training samples, $w_{gen} = 0.5$, $R = 6$.

Finally, we show the performance of scalability to a very large dataset when comparing all the methods. Here, we use the first 5 minutes of ECoG data with total number of 36 000 data points, and randomly select 18 000 points as training set and use the rest as the testing set. As is shown in Table 5.3, the results confirm the great superiority of our method to tensor GP in terms of scalability. Note that we empirically set w_{gen} as 0.4 and R as 20, which makes our method relatively much faster than the case of 10 000 at only a small cost of accuracy loss.

5.5 Conclusion

In this chapter, a new tensor-variate local GP regression method have been introduced, which successfully adapts the local GP modeling to the tensor input space. By doing so, the new method is able to handle the applications of tensor steams in an online fashion. Furthermore, two different searching strategies have been explored to find the nearest neighbouring local experts for a reliable prediction. The experimental results have demonstrated the effectiveness and scalability of this method with very large-scaled data.

In the next chapter, we will focus on the sequential tensor regression method that also recursively address the infinite time-dependent tensor streams but with more general tensorial input and general tensorial output.

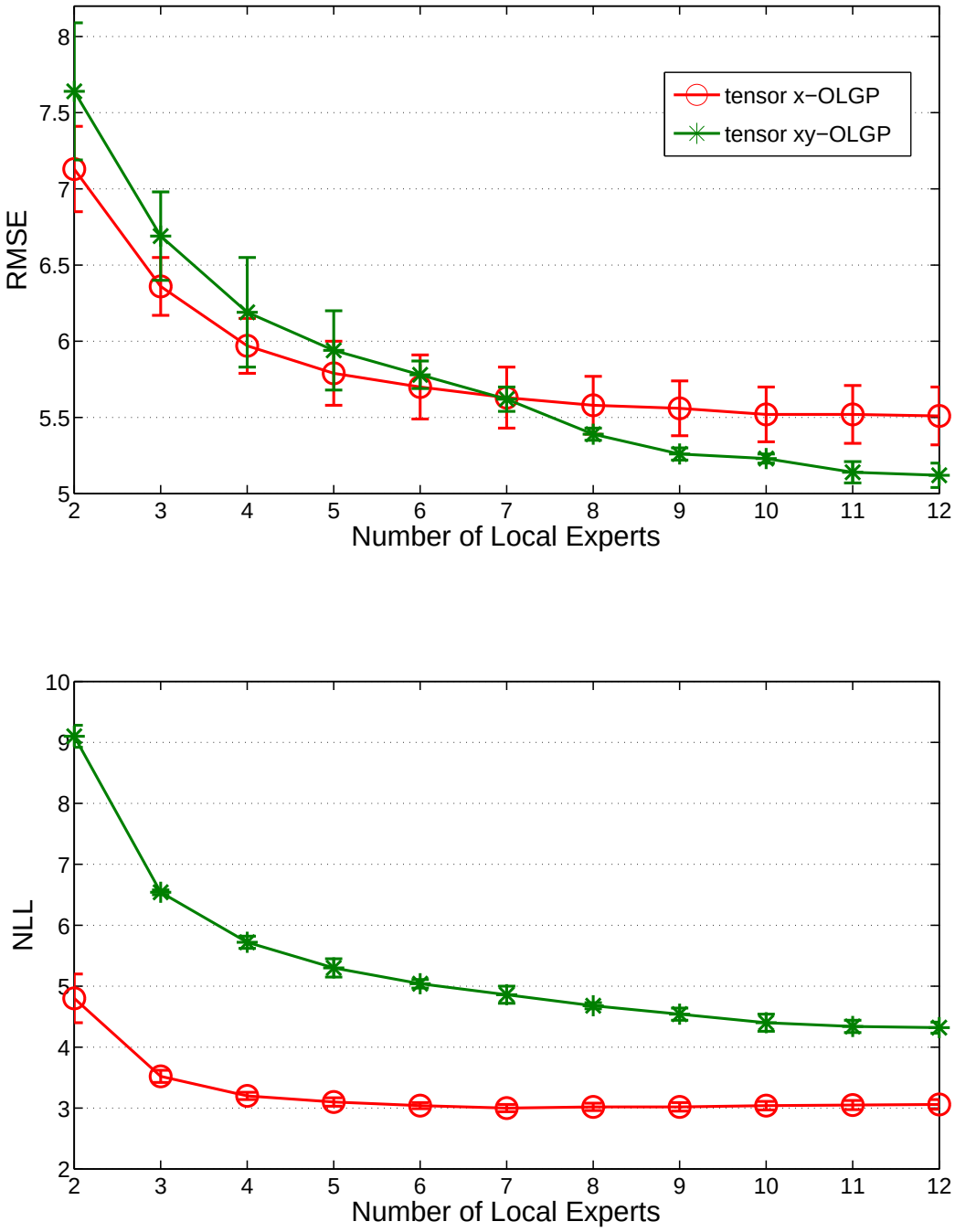


FIGURE 5.3 – RMSE (top) and NLL (bottom) vs. number of local experts.

Chapitre 6

Incremental Higher-order Partial Least Squares Regression (IHOPLS)

This chapter starts by presenting a brief review of higher-order partial least square (HOPLS) regression model. Then, a computationally efficient online tensor-variate regression algorithm, named incremental higher-order partial least square (IHOPLS) [Hou and Chaib-draa, 2016], is presented. This algorithm extends the standard HOPLS to the setting of infinite time-dependent tensor streams. Finally, the effectiveness and efficiency of the IHOPLS approach is demonstrated by experimental results of the real-world applications, i.e., reconstruction of $3D$ *motion trajectories from video and ECoG streams.*

6.1 Introduction

In previous chapters, we mainly focused on the tensor regression modelings with tensor-structured input and scalar output. Very little has been said about the learning models with tensor-structured output. Nevertheless, there are a large number of real-world regression tasks that involve tensorial output data containing abundant multiway structural information. Let us take the motion capture data from $3D$ human pose estimation as an example. Indeed, ignoring the correlations among positions of different parts of human body in this example might suffer a significant loss of accuracy when making prediction.

In order to leverage such multiway structural information, several approaches have been proposed to handle the output data with tensor structure. As mentioned in previous chapters, one of them in widespread use is N-way partial least squares (NPLS) [Bro, 1996] which carries out a joint CP decomposition of tensorial input and output into sum of rank-one tensors. To overcome the inferior fitness ability and the slow convergence rate of NPLS, the state-of-the-art higher-order partial least squares (HOPLS) regression model was introduced by Zhao et al. [2011, 2013a]. HOPLS is based on orthogonal block Tucker decomposition to project input

and output tensors onto a new latent space, so that the extracted factors can capture the covariance between input and output tensorial pair as much as possible. The most preferable property of HOPLS over NPLS lies in its power to provide superior predictability with optimal balance between fitness and model complexity [Zhao et al., 2013a].

For many real-world applications, however, the tensorial data that are encountered often take the form of extremely large or even infinite tensor sequences [Sun et al., 2008], especially in high-speed or time-critical dynamic environments, where the new tensor pairs keep coming fast over time. Although we can apply the batch method to all the data each time a new pair arrives, the HOPLS can nevertheless quickly become computationally prohibitive or merely infeasible. Moreover, it is often impossible to store the data entirely in the memory or require the whole dataset to be available up front. This hence sets the stage for the memory efficient sequential methods that can recursively handle the data.

Among these sequential methods, the recursive N-way partial least squares (RNPLS) [Eliseyev and Aksenova, 2013] processes the tensor sequences by unifying the recursive calculation scheme of recursive partial least squares (RPLS) [Qin, 1998] (see Appendix A.1) with the multiway data representation of NPLS. Inheriting the drawbacks of NPLS, the RNPLS likewise suffers from the lack of adequate accuracy and the slow convergence rate because its solution is based on combining an NIPALS-like algorithm [Wold, 1975a] with CP decomposition. Thus, the speed is rather slow especially when a relatively larger number of latent vectors are required for sufficient accuracy, which significantly reduces the applicability of RNPLS in time-critical applications.

Another recent work named accelerated low-rank tensor online learning (ALTO) [Yu et al., 2015] has been proposed for the speed of processing spatio-temporal tensor sequence using a random low-rank projection technique. However, ALTO is specially designed for spatio-temporal data structure and consequently demands the spatial and temporal modes are at least shared in common between the input and output tensors. Some other nice recursive calculation strategies [Sun et al., 2008; O'Hara, 2010] or fast decomposition method [Wang et al., 2015] have been proposed but just for the input-side tensor, and applied in the unsupervised learning settings only.

In this chapter, we extend HOPLS to the incremental higher-order partial least square (IHOPLS) [Hou and Chaib-draa, 2016] approach for the recursive calculation of infinite time-critical tensor streams. Compared to HOPLS, our IHOPLS approach aims to track the time-dependent changes and recursively updates the projection factors over time via an efficient incremental clustering strategy for the latent variables. In this way, the computational burden for very large-scale data can be greatly reduced while remaining in a low approximately constant level. Moreover, benefiting from the advantages of HOPLS over NPLS model, IHOPLS, as shown in Section 6.4, is much faster and exhibits a better predictive ability than RNPLS.

6.2 High-order Partial Least Squares Regression (HOPLS) Review

Recall that from Section 3.1.1, the principle behind HOPLS aims to sequentially conduct a set of joint block Tucker decompositions of $\mathcal{X}$ and $\mathcal{Y}$ with constraints that the extracted latent variables capture the maximum covariance between $\mathcal{X}$ and $\mathcal{Y}$. Specifically, suppose we have a $(M+1)$-order independent tensor $\mathcal{X} \in \mathbb{R}^{N \times I_1 \times \cdots \times I_M}$ and a $(L+1)$-order dependent tensor $\mathcal{Y} \in \mathbb{R}^{N \times J_1 \times \cdots \times J_L}$, which can be obtained by concatenating N pairs of observations $\{(\mathcal{X}^{(n)}, \mathcal{Y}^{(n)})\}_{n=1}^{N}$ that couple in the first mode with equal number of samples. Unlike NPLS, HOPLS decomposes $\mathcal{X}$ into a sum of *rank*-$(1, H_1, ..., H_M)$ Tucker blocks and $\mathcal{Y}$ into a sum of *rank*-$(1, K_1, ..., K_L)$ Tucker blocks, respectively. In this case, $\mathcal{X}$ and $\mathcal{Y}$ read

$$\begin{aligned} \mathcal{X} &= \sum_{r=1}^{R} \mathcal{G}_r^{\mathcal{X}} \times_1 \mathbf{t}_r \times_2 \mathbf{P}_r^{(1)} \times \cdots \times_{M+1} \mathbf{P}_r^{(M)} + \epsilon_{\mathcal{X}}, \\ \mathcal{Y} &= \sum_{r=1}^{R} \mathcal{G}_r^{\mathcal{Y}} \times_1 \mathbf{t}_r \times_2 \mathbf{Q}_r^{(1)} \times \cdots \times_{L+1} \mathbf{Q}_r^{(L)} + \epsilon_{\mathcal{Y}}, \end{aligned} \tag{6.1}$$

where R denotes the number of latent vectors and $\mathbf{t}_r \in \mathbb{R}^N$ corresponds to the rth latent column vector. $\mathcal{G}_r^{\mathcal{X}} \in \mathbb{R}^{1 \times H_1 \times \cdots \times H_M}$ and $\mathcal{G}_r^{\mathcal{Y}} \in \mathbb{R}^{1 \times K_1 \times \cdots \times K_L}$ represent the rth core tensors. $\{\mathbf{P}_r^{(m)}\}_{m=1}^{M} \in \mathbb{R}^{I_m \times H_m}$ and $\{\mathbf{Q}_r^{(l)}\}_{l=1}^{L} \in \mathbb{R}^{J_l \times K_l}$ are the respective rth projection matrices.

To complete HOPLS, [Zhao et al., 2013a] introduced an algorithm exploiting a deflation procedure to sequentially extract R latent vectors. Note that HOPLS uses a closed-form HOSVD solution [De Lathauwer et al., 2000a] on the cross-variance tensor to estimate the model parameters, which is more computationally efficient than an iterative NPLS.

6.3 Incremental Higher-order Partial Least Square Regression (IHOPLS)

The objective of our IHOPLS approach [Hou and Chaib-draa, 2016] is to deal with infinite time-evolving tensor streams and adapt HOPLS to online setting. For this purpose, we first perform a clustering of the incremental data pair in the projected latent subspace. Then, our model keeps track of the summary of previous data in terms of an internal data representation using the clustered latent variables and other model parameters. Because the latent variables are actually the compressed version of data in a low dimensional space, clustering on latent variables can be very efficient and alleviate the curse of dimensionality problem [Oommen et al., 2008]. By doing so, the standard HOPLS can be efficiently applied to the small-scale reconstructed tensors to update model parameters every time when a new tensor pair arrives, resulting in an much less and roughly constant processing time.

Algorithm 6 Incremental Higher-Order Partial Least Squares (IHOPLS)

Input: all the old model parameters $\tilde{\mathbf{T}} = [\tilde{\mathbf{t}}_1, ..., \tilde{\mathbf{t}}_R]$, $\{\{\tilde{\mathbf{P}}_r^{(m)}\}_{m=1}^M, \{\tilde{\mathbf{Q}}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ and $\{\tilde{\mathcal{G}}_r^{\mathcal{X}}, \tilde{\mathcal{G}}_r^{\mathcal{Y}}\}_{r=1}^R$, new tensor data pair $\{\mathcal{X}_{new} \in \mathbb{R}^{1\times I_1\times\cdots\times I_M}, \mathcal{Y}_{new} \in \mathbb{R}^{1\times J_1\times\cdots\times J_L}\}$

Output: all the new model parameters $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R]$, $\{\{\mathbf{P}_r^{(m)}\}_{m=1}^M, \{\mathbf{Q}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ and $\{\mathcal{G}_r^{\mathcal{X}}, \mathcal{G}_r^{\mathcal{Y}}\}_{r=1}^R$

Reconstruct the old internal tensor representation $\{\tilde{\mathcal{X}} \in \mathbb{R}^{N_{clus}\times I_1\times\cdots\times I_M}, \tilde{\mathcal{Y}} \in \mathbb{R}^{N_{clus}\times J_1\times\cdots\times J_L}\}$ from the old latent matrix $\tilde{\mathbf{T}}$, the loadings $\{\{\tilde{\mathbf{P}}_r^{(m)}\}_{m=1}^M, \{\tilde{\mathbf{Q}}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ and the core tensors $\{\tilde{\mathcal{G}}_r^{\mathcal{X}}, \tilde{\mathcal{G}}_r^{\mathcal{Y}}\}_{r=1}^R$ according to the equation (6.1):

$$\tilde{\mathcal{X}} = \sum_{r=1}^{R} \tilde{\mathcal{G}}_r^{\mathcal{X}} \times_1 \tilde{\mathbf{t}}_r \times_2 \tilde{\mathbf{P}}_r^{(1)} \times \cdots \times_{M+1} \tilde{\mathbf{P}}_r^{(M)}$$

$$\tilde{\mathcal{Y}} = \sum_{r=1}^{R} \tilde{\mathcal{G}}_r^{\mathcal{Y}} \times_1 \tilde{\mathbf{t}}_r \times_2 \tilde{\mathbf{Q}}_r^{(1)} \times \cdots \times_{L+1} \tilde{\mathbf{Q}}_r^{(L)}$$

Concatenate the $\{\mathcal{X}_{new}, \mathcal{Y}_{new}\}$ to $\{\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}\}$ in the first mode to generate tensor pair:

$$\mathcal{X} = \begin{bmatrix} \tilde{\mathcal{X}} \\ \mathcal{X}_{new} \end{bmatrix} \in \mathbb{R}^{N_{clus}+1\times I_1\times\cdots\times I_M} \quad \mathcal{Y} = \begin{bmatrix} \tilde{\mathcal{Y}} \\ \mathcal{Y}_{new} \end{bmatrix} \in \mathbb{R}^{N_{clus}+1\times J_1\times\cdots\times J_L}$$

Apply the standard HOPLS to $\{\mathcal{X}, \mathcal{Y}\}$ to find newly updated model parameters $\mathbf{T}$, $\{\{\mathbf{P}_r^{(m)}\}_{m=1}^M, \{\mathbf{Q}_r^{(l)}\}_{l=1}^L\}_{r=1}^R$ and $\{\mathcal{G}_r^{\mathcal{X}}, \mathcal{G}_r^{\mathcal{Y}}\}_{r=1}^R$

for $k = 1$ to number of clusters N_{clus} **do**

  Compute the similarity between the projected new data $\mathbf{T}(N_{clus}+1)$ from the $(N_{clus}+1)$th row vector of $\mathbf{T}$ and the projected cluster center in the kth row vector of $\mathbf{T}$:

$$w_k = < \frac{\mathbf{T}(N_{clus}+1)}{||\mathbf{T}(N_{clus}+1)||}, \frac{\mathbf{T}(k)}{||\mathbf{T}(k)||} >$$

end for

Choose the nearest cluster center v: $sim_v = \max(w_k)$

if $sim_v > w_{gen}$ **then**

  Update the number of data points in cluster v: $C_v = C_v + 1$

  Set the timestamp of cluster v: $TS_v = 1$

  Update the timestamps of other clusters v': $TS_{v'} = TS_{v'} + 1$

  Update the cluster center row vector:

$$\mathbf{T}(v) = \frac{(C_v - 1) * \mathbf{T}(v) + 1 * \mathbf{T}(N_{clus}+1)}{C_v}$$

  Remove the last row vector $\mathbf{T}(N_{clus}+1)$ from $\mathbf{T}$

else

  Set the number of data points in cluster $(N_{clus}+1)$: $C_{N_{clus}+1} = C_{N_{clus}+1} + 1$

  Set the timestamp of cluster $(N_{clus}+1)$: $TS_{N_{clus}+1} = 1$

  Update the timestamps of other clusters v': $TS_{v'} = TS_{v'} + 1$

  Update the number of row vector in $\mathbf{T}$: $N_{clus} = N_{clus} + 1$

  if maximum number of clusters N_{max} is reached **then**

    Remove the cluster center represented by the ith row vector with minimum ratio of C_i/TS_i, $i = 1, ..., N_{clus}$

  end if

end if

The whole mechanism named IHOPLS for incrementally updating all the model parameters is summarized in Algorithm 6. Similar to RNPLS, IHOPLS is a sequential algorithm that repeatedly receives and processes a new coming tensor pair over time. More specifically, for a new tensor pair $\{\mathcal{X}_{new} \in \mathbb{R}^{1\times I_1\times\cdots\times I_M}, \mathcal{Y}_{new} \in \mathbb{R}^{1\times J_1\times\cdots\times J_L}\}$, we first reconstruct the internal tensor representation $\{\tilde{\mathcal{X}} \in \mathbb{R}^{N_{clus}\times I_1\times\cdots\times I_M}, \tilde{\mathcal{Y}} \in \mathbb{R}^{N_{clus}\times J_1\times\cdots\times J_L}\}$ from the previous model parameters to summarize all the old data and approximate the current modeling state (line 3).

Note that the latent matrix $\tilde{\mathbf{T}}$ represents all clusters and each row vector in $\tilde{\mathbf{T}}$ stands for a projected cluster center or a projected data sample in latent subspace, while all the projection matrices and core tensors can be regarded as tensorial bases onto which the tensorial data projects. Thus, the number of row vectors N_{clus} is in fact the number of clusters. We then concatenate the new tensor pair $\{\mathcal{X}_{new}, \mathcal{Y}_{new}\}$ to $\{\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}\}$ along the first mode to generate tensor pair $\{\mathcal{X} \in \mathbb{R}^{N_{clus}+1\times I_1\times\cdots\times I_M}, \mathcal{Y} \in \mathbb{R}^{N_{clus}+1\times J_1\times\cdots\times J_L}\}$ with sample size $N_{clus}+1$ (line 4). Subsequently, we apply the standard HOPLS to $\{\mathcal{X}, \mathcal{Y}\}$ to extract new model parameters by incorporating the information contained in $\{\mathcal{X}_{new}, \mathcal{Y}_{new}\}$ into the model (line 5).

Having extracted all parameters, we next allocate the projected new data, represented by the extra row vector $\mathbf{T}(N_{clus}+1)$ in latent matrix $\mathbf{T}$, into a collection of clusters that are expressed by the cluster centers using row vectors from 1 to N_{clus}. To this end, we choose the nearest cluster v whose center $\mathbf{T}(v)$ has the highest similarity measure sim_v with row vector $\mathbf{T}(N_{clus}+1)$ (lines 6-9). If sim_v is greater than a predefined threshold w_{gen}, then we insert this new projected data into that cluster v, and update the cluster center $\mathbf{T}(v)$ accordingly (lines 10-15). Otherwise, we define $\mathbf{T}(N_{clus}+1)$ to be the new cluster center and increase the number of clusters by one (lines 17-20).

Meanwhile, we keep tracking both the number of data points and timestamps for each cluster i whose ratio C_i/TS_i indicates the relative importance of that cluster in participating in summarizing the current state of the model. This implies that the cluster with more data points and more recent timestamps is more important than those far in the past with less data points. If the number of clusters exceeds the maximum value, then the cluster with minimum ratio should be removed from $\mathbf{T}$ (lines 21-23).

For HOPLS, the overall computational cost is dominated by $\mathcal{O}(\max(NI^{M+L}, I^{M+L+1}))$ whose two terms correspond to establishing and decomposing the cross-covariance tensor, where I is the maximum index of all modes and is not large in practice. Evidently, as N increases over time, the first term will eventually surpass the second term owing to the unbounded data streams. Since the clusters number N_{clus} is bounded and small, the complexity of IHOPLS, on the other hand, is mainly affected by a constant $\mathcal{O}(\max(N_{clus}I^{M+L}, I^{M+L+1}))$.

6.4 Experimental Results

We compared our IHOPLS approach with RNPLS [Eliseyev and Aksenova, 2013] and with HOPLS [Zhao et al., 2013a] in an online fashion, that is, first making a prediction for the new input and then updating the model using the new ground truth. The performance was quantitatively evaluated in terms of the root mean squares of prediction (RMSEP) [Kim et al., 2005]

$$RMSEP = \sqrt{\frac{\|\mathcal{Y} - \hat{\mathcal{Y}}\|_{\mathsf{F}}^2}{I_1 I_2 \cdots I_D}} \tag{6.2}$$

and the Q index [Luo et al., 2015] which is defined as

$$Q = 1 - \frac{\|\mathcal{Y} - \hat{\mathcal{Y}}\|_{\mathsf{F}}}{\|\mathcal{Y}\|_{\mathsf{F}}} \tag{6.3}$$

where $\hat{\mathcal{Y}}$ is the prediction of D-order tensor $\mathcal{Y}$. I_d corresponds to the dimensionality of the mode d with $d = 1, ..., D$. The optimal hyperparameters of RNPLS were selected by cross-validation on the first $1/3$ data sequence, including the forgetting factor γ that is introduced for RPLS in Appendix A.1. In addition, the convergence criterion and the maximum iteration number for estimating the loadings were selected by cross-validation to be 10^{-5} and 40 that optimally balance between accuracy and efficiency. On the other hand, we empirically fixed number of latent vectors R and number of loading λ for both HOPLS and IHOPLS. We manually tuned the partitioning threshold parameter w_{gen} and the maximum cluster number N_{max} whose combination actually controls the number of clusters responsible for the recursive update. We repeatedly run the experiment 10 times.

6.4.1 Utrecht Multi-Person Motion Database

The first experiment was carried out on the Multi-Person Motion (UMPM) benchmark [Aa et al., 2011] that contains temporally synchronized video sequences and human motion capture data. In our experiment, the input was an intensity image sequence from the front camera with a downsized resolution of 32×24 pixels, taking form of a 3-order predictor tensor (i.e., *frames* $\times$ *width* $\times$ *height*). To test on large-sized data, we concatenated videos from four scenarios, including "triangle", "grab", "chair" and "table", into one big sequence with total 5250 frames at 25 fps (frame per second). The corresponding output containing $3D$ positions of 37 reflective markers can be represented as a 3rd-order tensor (i.e., *samples* $\times$ $3D$ *positions* $\times$ *markers*).

The averaged predictive performance and the learning time of the IHOPLS, HOPLS and RNPLS are summarized in Table 6.1. As we see, our approach significantly outperforms HOPLS and the best of RNPLS in prediction accuracy with respect to both high and low frame rate situation, implying the usefulness of IHOPLS for adjacent observations with relatively less overlapped features. In particular, the averaged improvement by IHOPLS over HOPLS and RNPLS at 5fps when using 4 latent vectors were significant 0.25, 0.08 in terms of Q, and

TABLE 6.1 – Performance comparison of IHOPLS, HOPLS and RNPLS for the averaged Q, RMSEP and total learning time on UMPM data.

Size = 1050 and Frequency = 5Hz				
Methods	Rank and Hyperparameter	Q	RMSEP	Total Time (s)
IHOPLS	$R = 4$, $\lambda = 2$, $w_{gen} = 0.2$, $N_{max} = 20$	**0.8240**	**149.6**	**505.3**
	$R = 8$, $\lambda = 4$, $w_{gen} = 0.2$, $N_{max} = 20$	**0.8274**	**145.2**	**902.7**
HOPLS	$R = 4$, $\lambda = 2$	0.5700	349.4	812.5
	$R = 8$, $\lambda = 4$	0.5864	332.4	1471.8
RNPLS	$F = 4$, $\gamma = 0.8$	0.7433	215.7	6901.3
Size = 5250 and Frequency = 25Hz				
Methods	Rank and Hyperparameter	Q	RMSEP	Total Time (s)
IHOPLS	$R = 4$, $\lambda = 2$, $w_{gen} = 0.2$, $N_{max} = 20$	**0.9645**	**29.5**	**2316.3**
	$R = 8$, $\lambda = 4$, $w_{gen} = 0.2$, $N_{max} = 20$	**0.9664**	**28.5**	**4775.2**
HOPLS	$R = 4$, $\lambda = 2$	0.5778	342.8	21145.3
	$R = 8$, $\lambda = 4$	0.6008	320.7	36396.6
RNPLS	$F = 4$, $\gamma = 0.6$	0.9337	50.4	19923.0

199.8, 66.1 in terms of RMSEP. This indicates the capability of IHOPLS in capturing local variation in dynamic streams. In the meantime, the averaged total learning time of IHOPLS, namely 505.3 seconds, is much faster than that 6901.3 seconds of RNPLS. Figure 6.1 and Figure 6.2 illustrate the CPU time and prediction error (RMSEP) versus elapsed frames for low frequency 5fps and high frequency 25fps, respectively.

Overall, the CPU cost for both IHOPLS and RNPLS exhibited a constant trend, while cost of offline HOPLS keeps increasing over time. In contrast to HOPLS and RNPLS, IHOPLS was highly computationally efficient no matter which frame frequency was used. Note carefully that, for both cases, the RMSEP of IHOPLS was relatively large at the beginning learning period but was quickly reduced to and remained in a low level as the test went on. This is the case except for a few bursts or bumps that correspond to the points when we concatenated different video sequences or significant changes of motions in the input streams, which is more obvious in the case of 5250 samples.

6.4.2 Neurotycho Electrocorticography Database

We also evaluated our IHOPLS approach on an application of reconstruction of limb movements from monkey's brain signals using Neurotycho food tracking Electrocorticography (ECoG) dataset [Chao et al., 2010]. Particularly, the wavelet transformed ECoG signal, a 4-order tensor $\mathcal{X} \in R^{N \times 10 \times 10 \times 32}$ (i.e., *samples* $\times$ *time* $\times$ *frequency* $\times$ *channels*), was employed as the input. The output was a 3rd-order tensor of $3D$ movement distances of the monkey's limb on different markers.

In Table 6.2, our approach shows a remarkable advantage regarding computational burden

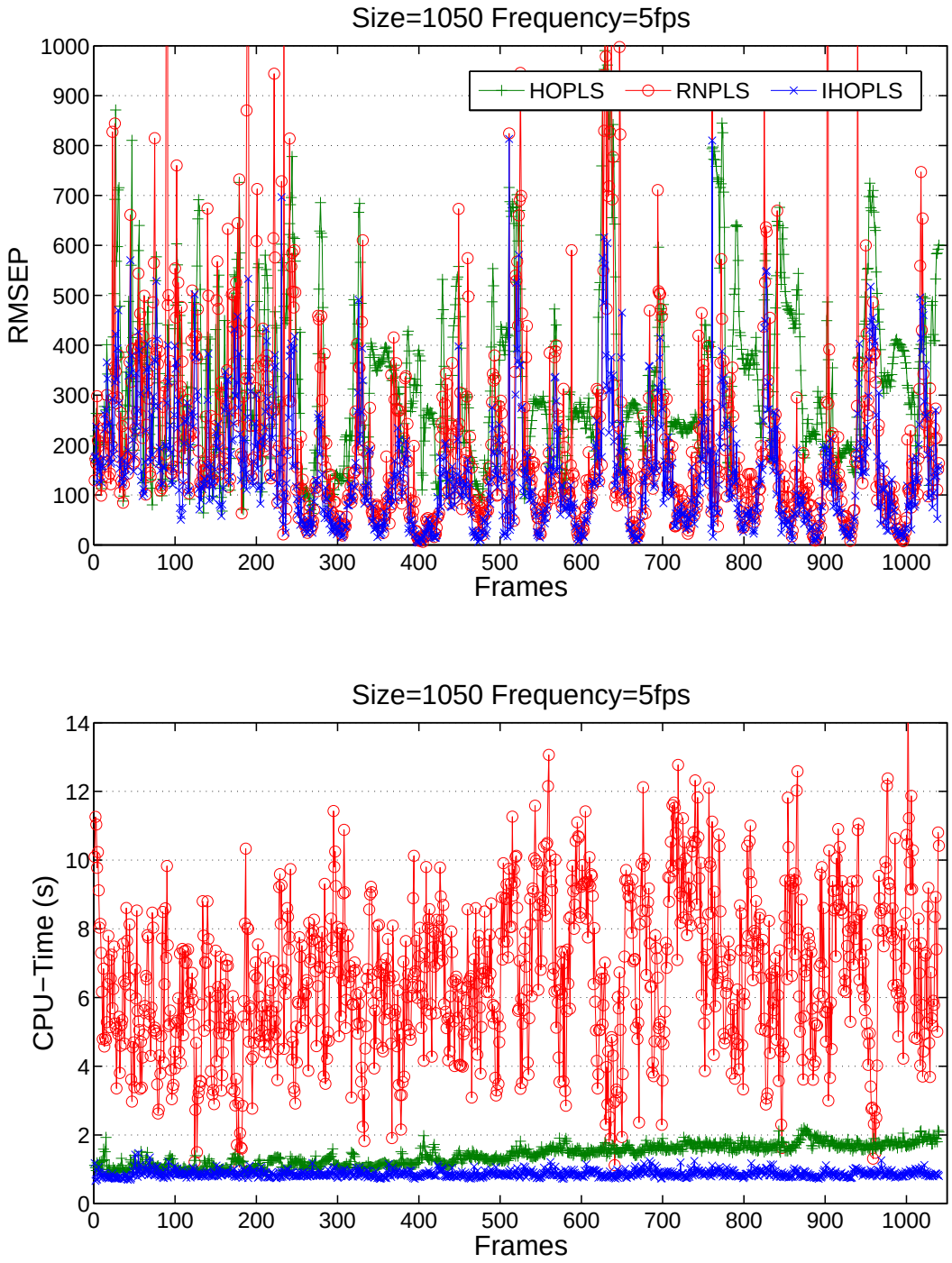


FIGURE 6.1 – Prediction error (top) and CPU cost (bottom) of three methods over time, $R = 8$ and $\lambda = 4$ for IHOPLS and HOPLS for sequence length of 1050 and frequency at 5fps on UMPM.

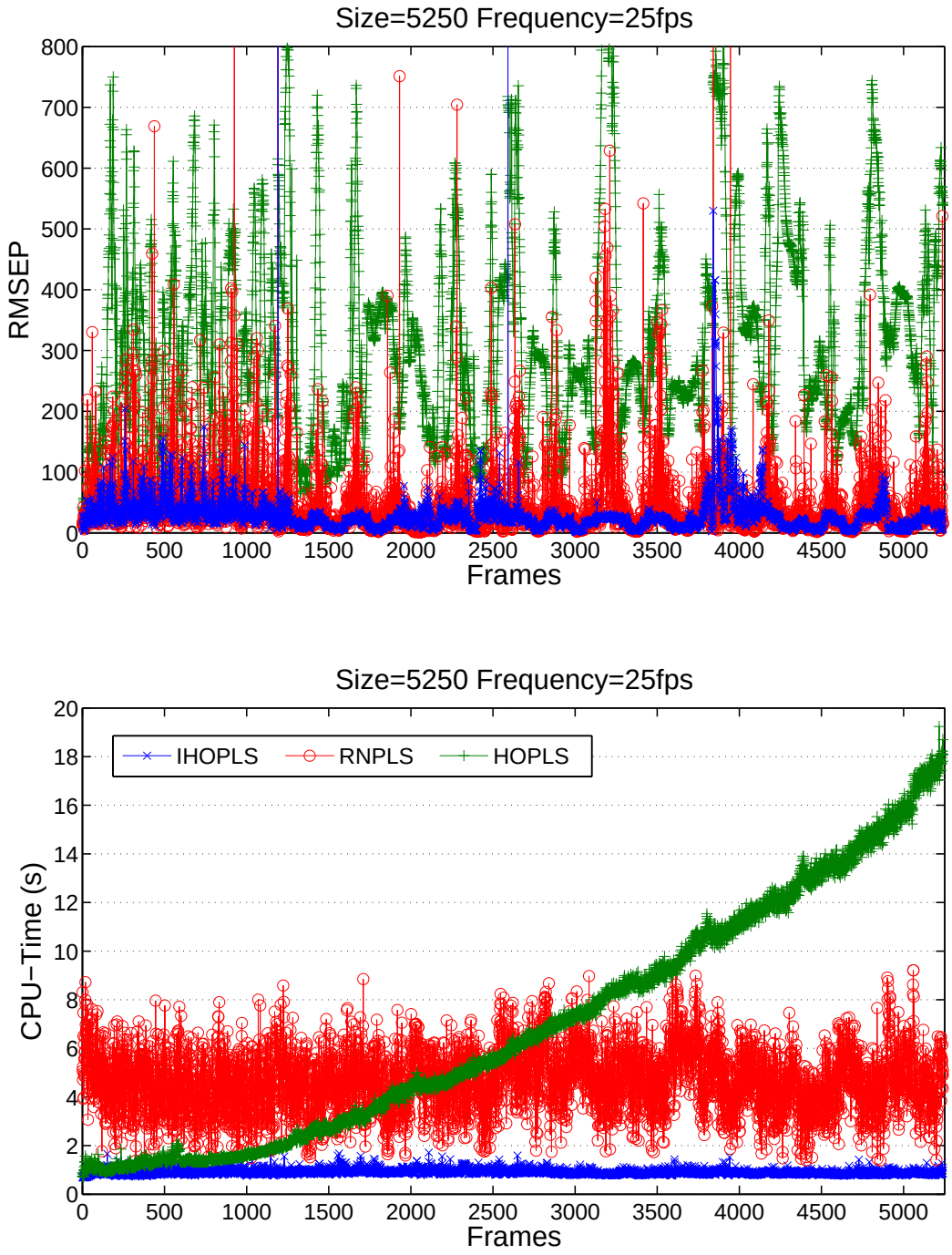


FIGURE 6.2 – Prediction error (top) and CPU cost (bottom) of three methods over time, $R = 8$ and $\lambda = 4$ for IHOPLS and HOPLS for sequence length of 5250 and frequency at 25fps on UMPM.

TABLE 6.2 – Performance comparison of IHOPLS, HOPLS and RNPLS for the averaged Q, RMSEP and total learning time on ECoG data.

Size = 900 and Frequency = 1Hz				
Methods	Rank and Hyperparameter	Q	RMSEP	Total Time (s)
IHOPLS	$R = 8$, $\lambda = 8$, $w_{gen} = 0.4$, $N_{max} = 100$	0.6962	34.4	**665.2**
HOPLS	$R = 8$, $\lambda = 8$	0.7003	33.7	1137.0
RNPLS	$F = 5$, $\gamma = 1$	**0.7036**	**33.2**	1647.8
Size = 18000 and Frequency = 20Hz				
Methods	Rank and Hyperparameter	Q	RMSEP	Total Time (s)
IHOPLS	$R = 8$, $\lambda = 8$, $w_{gen} = 0.4$, $N_{max} = 10$	**0.9204**	**9.0**	**17357.3**
HOPLS	$R = 8$, $\lambda = 8$	0.7214	30.3	586710.0
RNPLS	$F = 5$, $\gamma = 0.8$	0.9011	11.0	160113.4

over other methods. We also observed that IHOPLS achieved the best accuracy in the case of 20Hz while remaining competitive with HOPLS and RNPLS in the 1Hz case where the successive samples were selected with non-overlapped features.

Note that the accuracy was improved in the Figure 6.3 as we increased the maximum cluster number N_{max} from 10 to 400 for non-overlapped feature case, reflecting the fact that we need more clusters to characterize the variation of data. On the contrary, we may need fewer clusters for highly-overlapped samples so as to obtain a better predictive ability.

6.5 Discussion

We should mention that our IHOPLS method is specially designed for the online tensor stream setting where it is not feasible to search for the globally optimal model parameters since the new observations keep arriving. Rather than fixing the projection matrices and core tensors, we incrementally update them by assuming that the characteristics of the data might change over time. In practice, the number N_{clus} turns out to be very small such that the HOPLS can be executed efficiently owing to that the HOPLS is particularly suitable for small sample size [Zhao et al., 2013a]. IHOPLS only needs dozens or tens of samples to extract the initial model parameters. Moreover, it handles the time dependency by combining an appropriate maximum cluster number with an eliminating strategy based on an importance ratio.

6.6 Conclusion

This chapter presented our new online IHOPLS approach which efficiently adapts the standard HOPLS to the setting of infinite time-dependent tensor streams. Our approach allows recursively updating the regression coefficients by incrementally clustering the projected latent variables in latent space, leading to a constant computational load over time. We have

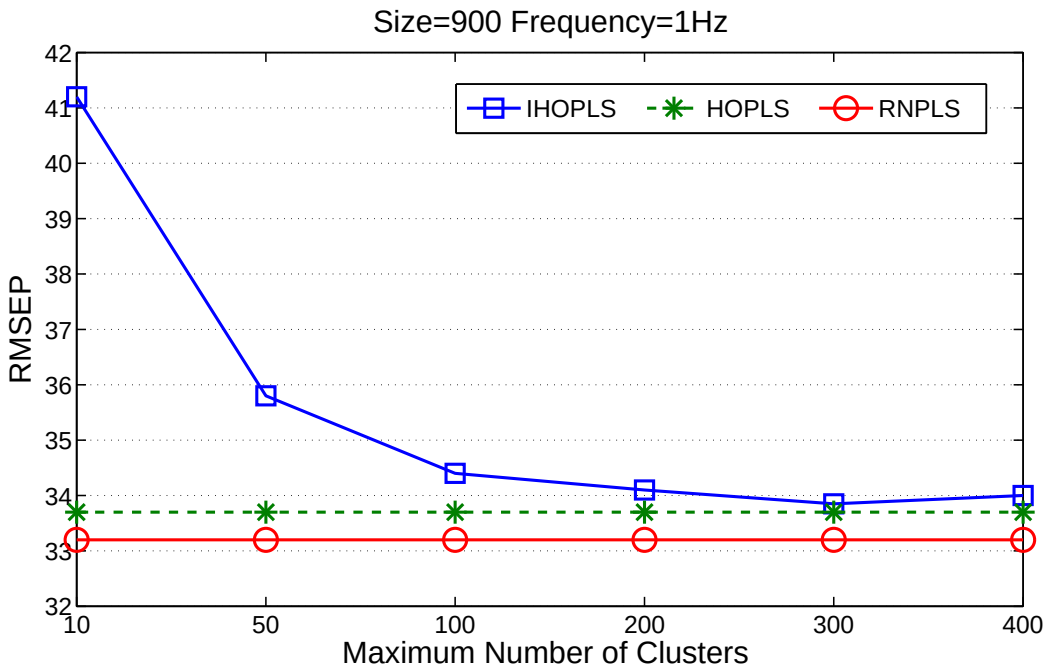


FIGURE 6.3 – Prediction errors versus N_{max} for $R = 8$ $\lambda = 8$ $w_{gen} = 0.4$ on ECoG data.

validated the effectiveness and scalability of our approach on two different real-life tensor regression applications with very large-scale data.

In the next chapter, we will address the tensor regression problem for general large tensor sequences using a more efficient sequential framework, which takes both 'sample' complexity and 'dimensionality' complexity into account.

Chapitre 7

Recursive Higher-order Partial Least Squares Regression (RHOPLS)

This chapter begins with the motivation and problem setting of the sequential processing of general tensor sequences for the regression problem. Then, for this setting, a fast tensor regression framework, called recursive higher-order partial least squares (RHOPLS) [Hou and Chaib-draa, 2017], is introduced. Finally, the significant speed-ups and high predictability of RHOPLS are verified on a variety of real-life applications.

7.1 Introduction

In the previous chapter, we proposed a simple recursive extension of HOPLS called incremental higher-order partial least squares (IHOPLS) to deal with infinite time-dependent tensor streams [Sun et al., 2008] by incrementally clustering the projected latent variables of data in latent space and summarizing the previous data. As a result, the computational load of regression on infinite tensor streams can be reduced to a roughly constant level over time.

In fact, IHOPLS is designed based on K-means clustering [Lloyd, 1982] of latent variables and consequently it leads to a significant deterioration of predictive performance when the dimensions of latent space required to be large to meet certain fitness accuracy. In addition, although successfully reducing the "sample" complexity of HOPLS, the computational load of IHOPLS, just like HOPLS, still remains a problem when data orders or d-ranks are large. This is due to the fact that it has to compute and decompose the huge cross-covariance tensor. Furthermore, IHOPLS suffers the same drawback as RNPLS in that it merges the factors at a large-scale raw data level and this can result in high computational and storage cost.

In this chapter, we introduce a fast tensor regression framework, called recursive higher-order partial least squares (RHOPLS) [Hou and Chaib-draa, 2017], for sequential blockwise processing of general tensor sequences. Our contributions are :

1. designing a recursive framework that efficiently updates the regression coefficients (factors) at a small-scale factor (feature) level instead of the large raw data (observation) level by integrating a low-rank modification strategy of the Tucker [O'Hara, 2010] into PLS ;
2. developing an efficient algorithm based on a series of computationally advantageous calculations yet with only a small number of factors for storage ;
3. applying RHOPLS to several challenging tasks such as estimation of human pose from videos, showing the great potentiality for fast real-time predictions of human pose positions.

In brief, our RHOPLS framework does not suffer from neither inferior predictability nor poor convergence rate of NPLS-based (CP-based) models. Our RHOPLS is also free from the computational issues related to the HOPLS-based models when the data order, "sample" complexity or "dimensionality" complexity is high. Finally, our RHOPLS exhibits highly competitive accuracy with the best batch methods but is much faster than other sequential methods.

7.2 Recursive Higher-order Partial Least Squares Regression (RHOPLS)

7.2.1 Problem Setting

Without the loss of generality, we consider an Nth-order input tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and an Mth-order output tensor $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$ with coupled observations, namely $I_1 = J_1$ and we process the data in terms of mini-batch of size b, say $\{\mathcal{X}(t) \in \mathbb{R}^{b \times I_2 \times \cdots \times I_N}, \mathcal{Y}(t) \in \mathbb{R}^{b \times J_2 \times \cdots \times J_M}\}$ for mini-batch t. I_0 stands for the number of initial observations. I denotes the maximum index of all modes, while R is used to define maximum *d-rank* of all modes which are supposed to be small in practice. Finally, F is the number of latent vectors, while L and K are used to denote the collection of *d-ranks* for the input and output loadings, respectively.

7.2.2 RHOPLS Framework

As mentioned in Section 1, sequential approaches to regression are capable of handling big data as well as online data. In this context, the key idea of RPLS [Qin, 1998] (see Appendix A.1) consists of directly adding the old matrix data to the new arriving matrix data to generate the joint matrix data, and then use an iterative procedure to extract the new set of factors from the joint matrix data. Notice that the old matrix data is represented by reformulating the previous set of factors into a fixed data size.

Extended from RPLS, RNPLS [Eliseyev and Aksenova, 2013] directly adds the old tensor data, refolded from the previous set of factors, to the new tensor data to form the joint tensor data, and then applies NPLS to joint tensor data to estimate the new set of factors.

Although recursively handling the new data, RNPLS has one major disadvantage in that it has to perform a slow iterative NIPALS-style procedure on the joint tensor data at the level of original dimensionality I. Furthermore, RNPLS represents the old tensorial data by refolding the factor matrices into tensor, such refolding operation on factors may cause undesired errors of approximating the old tensor data.

Different from RPLS and RNPLS, our RHOPLS framework proposes to recursively merge the new data into the old one in terms of factors (features) rather than considering the raw tensor data (observations). By doing so, RHOPLS directly additively updates the PLS-related parameters at much smaller scale using a closed-form solution, thus keeping track of the subspace patterns and the summary of model status.

Generally speaking, up to the point $t-1$, the set of old factors representing the RHOPLS model summarizes the total variation in all previous mini-batches, and hence approximates the current modeling state. Then, a newly arriving mini-batch t is decomposed into a set of incremental factors which contains the variation merely corresponding to the new data. Our RHOPLS framework carries out an update procedure in an "incremental approximation-expansion-compression-projection" fashion that effectively yet inexpensively "absorbs" the incremental factors into the old ones to produce a set of new factors at t. Continuing in the same spirit, we are able to efficiently process the new data in a mini-batch way over time.

Figure 7.1 illustrates the whole scheme that consists of the five principle steps as explained below. For the ease of demonstration, we choose $N=3$ order input tensor and $M=2$ order output tensor for visualization.

Step 0: Initial Approximation

As for the initial tensor pair $\{\mathcal{X}(0)\in\mathbb{R}^{I_0\times I_2\times\cdots\times I_N},\mathcal{Y}(0)\in\mathbb{R}^{I_0\times J_2\times\cdots\times J_M}\}$, we aim to extract a set of initial factors. Instead of using a deflation process of block tensor terms in HOPLS [Zhao et al., 2011, 2013a], we simply apply the standard Tucker to jointly decompose the initial data pair such that the latent components extracted from $\mathcal{X}(0)$ and $\mathcal{Y}(0)$ have the maximum pairwise covariance, which is of the form

$$\begin{aligned}\mathcal{X}(0)&\approx\mathcal{G}^x(0)\times_1\mathbf{T}(0)\times_2\mathbf{P}^{(2)}(0)\times_3\cdots\times_N\mathbf{P}^{(N)}(0),\\ \mathcal{Y}(0)&\approx\mathcal{G}^y(0)\times_1\mathbf{T}(0)\times_2\mathbf{Q}^{(2)}(0)\times_3\cdots\times_M\mathbf{Q}^{(M)}(0),\end{aligned}\tag{7.1}$$

where $\{\mathcal{G}^x(0)\in\mathbb{R}^{L_1\times L_2\times\cdots\times L_N},\mathcal{G}^y(0)\in\mathbb{R}^{K_1\times K_2\times\cdots\times K_M}\}$ serve as core tensors. $\mathbf{T}(0)\in\mathbb{R}^{I_0\times L_1}$ is the common latent matrix, while $\{\{\mathbf{P}^{(n)}(0)\}_{n=2}^N\in\mathbb{R}^{I_n\times L_n},\{\mathbf{Q}^{(m)}(0)\}_{m=2}^M\in\mathbb{R}^{J_m\times K_m}\}$ are the loadings corresponding to $\{\mathcal{X}(0),\mathcal{Y}(0)\}$. Clearly, we have $L_1=K_1$. To solve above factors, we first form the 1-mode cross-covariance tensor

$$\mathcal{C}(0)=\langle\mathcal{X}(0),\mathcal{Y}(0)\rangle_{\{1:1\}}\tag{7.2}$$

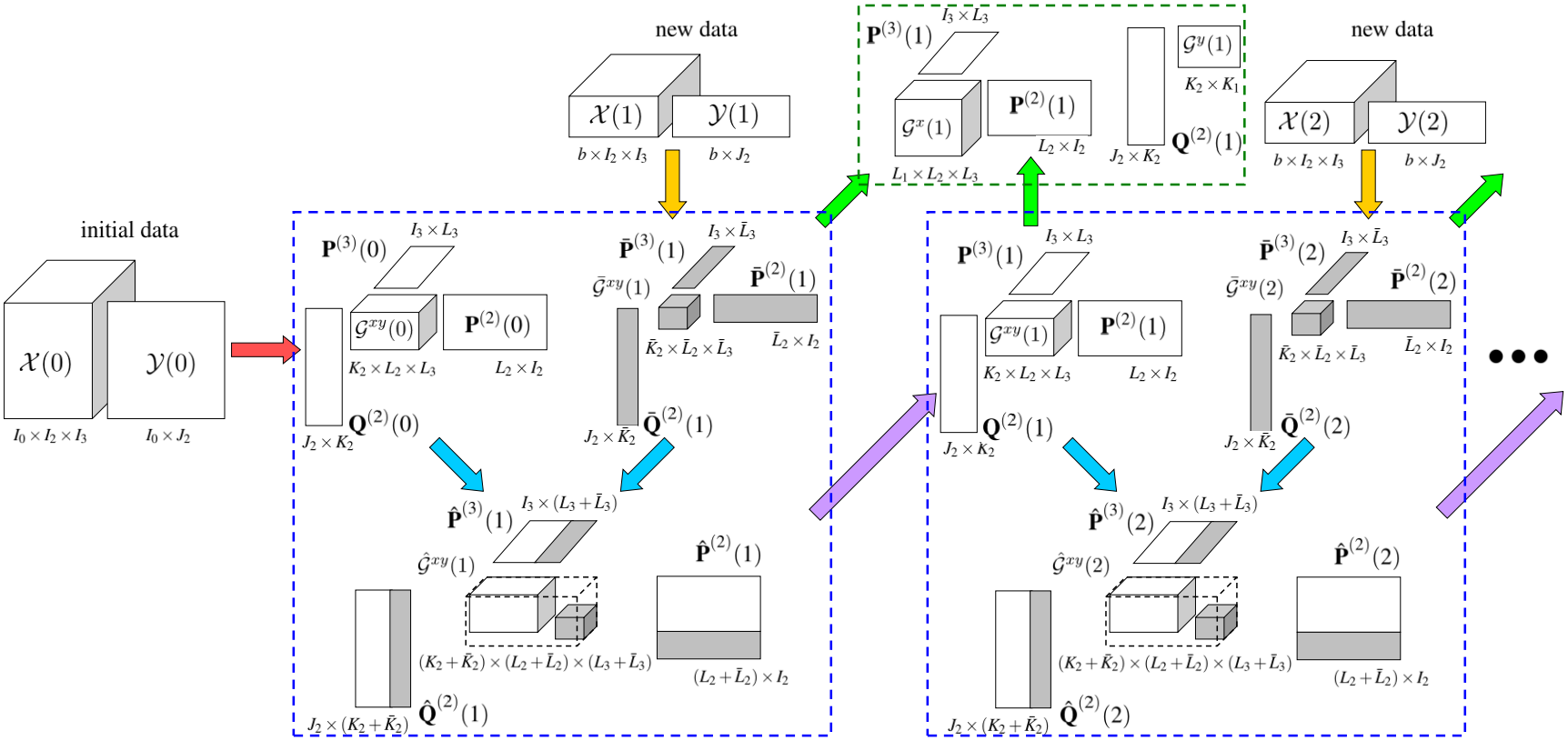


FIGURE 7.1 – The RHOPLS scheme. The framework generates a set of initial factors for the initial data (**Step 0** red arrow). At every iteration, the framework first generates a set of incremental factors for the new data (**Step 1** yellow arrow). Then, the information contained in new data, represented in terms of factors, is added to current model by an appending operation (**Step 2** blue arrow). Next, the augmented set of factors are truncated back into the ones with original sizes to yield new loadings (**Step 3** purple arrow). The new individual core tensors are produced using an internal tensor representation of model (in terms of factors) under the projection of the new loadings (**Step 4** green arrow).

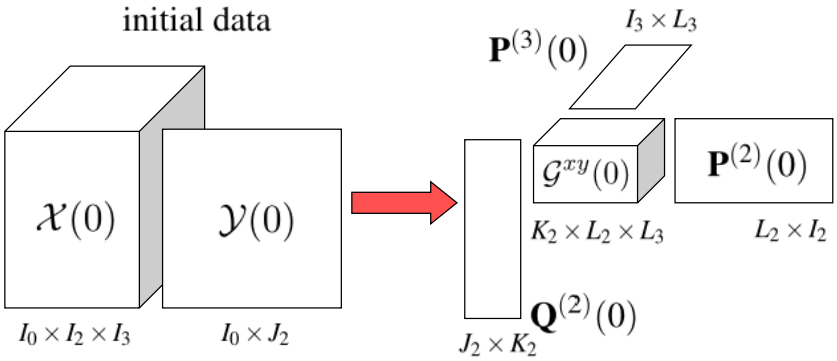


FIGURE 7.2 – The initial approximation step of RHOPLS framework for $t = 0$.

which follows by estimating all the loadings using a higher-order orthogonal iteration (HOOI) [De Lathauwer et al., 2000b] procedure as

$$\mathcal{C}(0) \approx \mathcal{G}^{xy}(0) \times_1 \mathbf{Q}^{(2)}(0) \times_2 \cdots \times_{M-1} \mathbf{Q}^{(M)}(0) \times_M \mathbf{P}^{(2)}(0) \times_{M+1} \cdots \times_{M+N-2} \mathbf{P}^{(N)}(0). \quad (7.3)$$

Here, $\mathcal{G}^{xy}(0) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N}$ is said to be 1-*mode cross-covariance core tensor*. Having obtained all the loadings, the core tensors $\{\mathcal{G}^x(0), \mathcal{G}^y(0)\}$ and common latent matrix $\mathbf{T}(0)$ can be computed according to (7.1) using higher-order singular vector decomposition (HOSVD) [De Lathauwer et al., 2000a]. From now on, these factors (common latent matrix, loadings, core tensors and the 1-mode cross-covariance core tensor), indicating the initial modeling state, are the parameters to be updated on new mini-batch. This process is depicted in Figure 7.2 in which the initial data $\mathcal{X}(0)$ and $\mathcal{Y}(0)$ are illustrated in terms of 3-order tensor and 2-order tensor (matrix), respectively.

Step 1: **Incremental Approximation**

Turning to mini-batch at the timestamp t, a new tensor pair $\{\mathcal{X}(t) \in \mathbb{R}^{b \times I_2 \times \cdots \times I_N}, \mathcal{Y}(t) \in \mathbb{R}^{b \times J_2 \times \cdots \times J_M}\}$ with small size b comes into the picture. To approximate the incremental data efficiently, we propose to collect the set of factors by first performing partial Tucker to $\mathcal{X}(t)$ and $\mathcal{Y}(t)$ separately, such that all the modes except the first mode are decomposed with d-ranks rank-$(\bar{L}_2, ..., \bar{L}_N)$ and rank-$(\bar{K}_2, ..., \bar{K}_M)$

$$\mathcal{X}(t) \approx \check{\mathcal{G}}^x(t) \times_2 \bar{\mathbf{P}}^{(2)}(t) \times_3 \cdots \times_N \bar{\mathbf{P}}^{(N)}(t), \quad (7.4)$$

$$\mathcal{Y}(t) \approx \check{\mathcal{G}}^y(t) \times_2 \bar{\mathbf{Q}}^{(2)}(t) \times_3 \cdots \times_M \bar{\mathbf{Q}}^{(M)}(t). \quad (7.5)$$

Thereafter, the 1-mode cross-covariance core tensor $\bar{\mathcal{G}}^{xy}(t)$ is immediately formed exploiting only the core tensors $\check{\mathcal{G}}^x(t)$ and $\check{\mathcal{G}}^y(t)$ as

$$\bar{\mathcal{G}}^{xy}(t) = \langle \check{\mathcal{G}}^x(t), \check{\mathcal{G}}^y(t) \rangle_{\{1:1\}}, \quad (7.6)$$

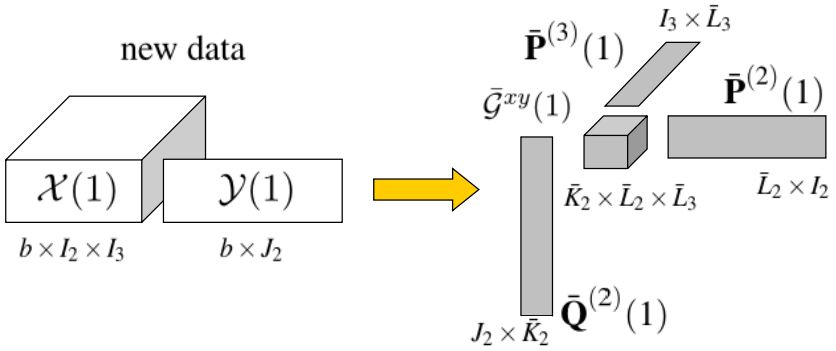


FIGURE 7.3 – The incremental approximation step of RHOPLS framework for $t = 1$.

where $\bar{\mathcal{G}}^{xy}(t) \in \mathbb{R}^{\bar{K}_2 \times \cdots \times \bar{K}_M \times \bar{L}_2 \times \cdots \times \bar{L}_N}$ can be calculated using only $\mathcal{O}(R^{M+N-2})$. In total, the cost for approximating the incremental data requires merely $\mathcal{O}(\max(R^{M+N-2}, RI^{M-1}, RI^{N-1}))$. Note that the HOPLS has to compute and decompose the 1-mode cross-covariance tensor of size $J_2 \times \cdots \times J_M \times I_2 \times \cdots \times I_N$ using entire data when estimating each one of R latent vectors [Zhao et al., 2013a], leading to costs as large as $\mathcal{O}(RI^{M+N-1})$. Thus, calculating and decomposing huge tensors R times yields substantial costs especially when data dimensionality, data order and d-ranks are large. In contrast to HOPLS, our Step 1 overcomes this drawback by first partially decomposing only the new mini-batch input and output individually via Tucker and then calculating the desired cross-covariance core tensor to gather the incremental factors. This step (see Figure 7.3 for $t = 1$) contributes to the speed-ups mainly from the perspective of significantly reducing the "dimensionality" complexity, since we extract eigenvectors on two much smaller-scale individual mini-batch tensors instead of on a massive cross-covariance tensor, and we also form the cross-covariance core tensor at the core tensor scale of R instead of at the original dimensionality I.

Step 2: **Expansion**

Inspired by the work of O'Hara [2010], which additively updates Tucker model using a low-rank subspace truncation strategy, we first append the set of incremental factors associated with the new mini-batch to the set of old factors that corresponds to the current model status, resulting in an augmented set of factors.

Different from the work of O'Hara [2010], we apply this strategy to the cross-covariance core tensor in a PLS framework involving both input and output sides rather than just input-side tensor as in O'Hara [2010]. Specifically, concatenating the variation captured by loading $\bar{\mathbf{P}}^{(n)}(t) \in \mathbb{R}^{I_n \times \bar{L}_n}$ to $\mathbf{P}^{(n)}(t-1) \in \mathbb{R}^{I_n \times L_n}$, we can get the augmented variation via loading

$$\hat{\mathbf{P}}^{(n)}(t) = [\mathbf{P}^{(n)}(t-1) \;\; \bar{\mathbf{P}}^{(n)}(t)] \in \mathbb{R}^{I_n \times (L_n + \bar{L}_n)} \quad \text{with} \quad n = 2, ..., N. \tag{7.7}$$

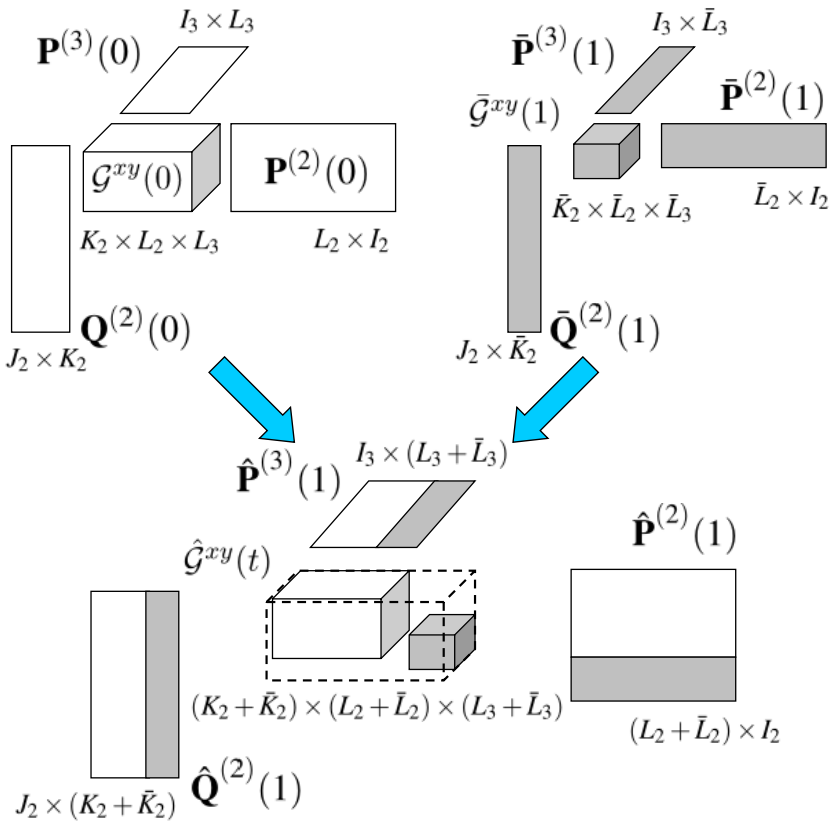


FIGURE 7.4 – The expansion step of RHOPLS framework for $t = 1$.

Likewise, we obtain the loading

$$\hat{\mathbf{Q}}^{(m)}(t) = [\mathbf{Q}^{(m)}(t-1) \;\; \bar{\mathbf{Q}}^{(m)}(t)] \in \mathbb{R}^{J_m \times (K_m + \bar{K}_m)} \quad \text{with} \quad m = 2, ..., M. \tag{7.8}$$

We can also get $\hat{\mathcal{G}}^{xy}(t) \in \mathbb{R}^{(K_2+\bar{K}_2)\times\cdots\times(K_M+\bar{K}_M)\times(L_2+\bar{L}_2)\times\cdots\times(L_N+\bar{L}_N)}$, named as augmented 1-mode cross-covariance core tensor, by appending $\bar{\mathcal{G}}^{xy}(t) \in \mathbb{R}^{\bar{K}_2\times\cdots\times\bar{K}_M\times\bar{L}_2\times\cdots\times\bar{L}_N}$ to $\mathcal{G}^{xy}(t-1) \in \mathbb{R}^{K_2\times\cdots\times K_M\times L_2\times\cdots\times L_N}$ in a super block-diagonal manner, leaving other new entries to be zeros. Figure 7.4 shows how the factors are concatenated together.

Step 3: **Compression**

Continuing on with the same strategy motivated by the work of O'Hara [2010], we next truncate the set of augmented factors back into the set of factors with the size of original d-ranks, which means finding the most dominant principal components in the subspaces of loadings out of the augmented set of loadings.

To this end, the augmented factors $\{\{\hat{\mathbf{P}}^{(n)}(t)\}_{n=2}^{N}, \{\hat{\mathbf{Q}}^{(m)}(t)\}_{m=2}^{M}\}$ and $\hat{\mathcal{G}}^{xy}(t)$ described in the previous Step 2 are truncated to produce new loadings $\{\{\mathbf{P}^{(n)}(t)\}_{n=2}^{N} \in \mathbb{R}^{I_n \times L_n}, \{\mathbf{Q}^{(m)}(t)\}_{m=2}^{M} \in$

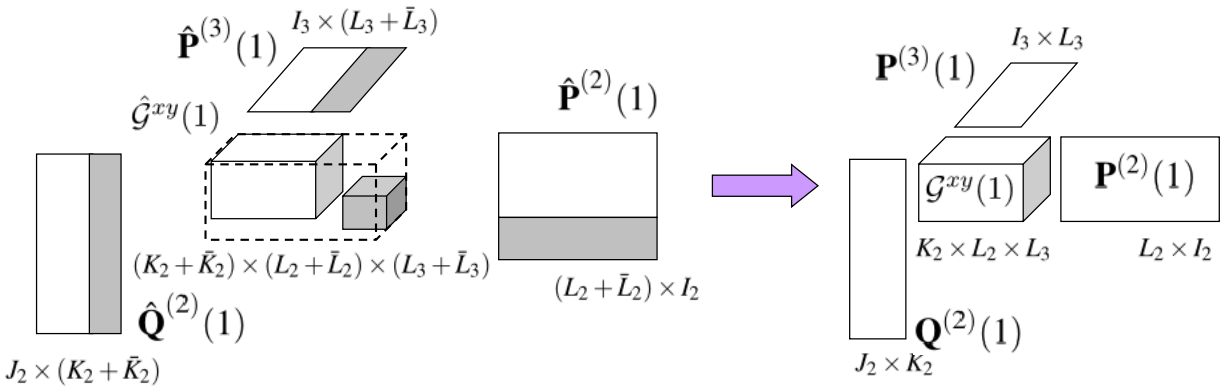


FIGURE 7.5 – The compression step of RHOPLS framework for $t = 1$.

$\mathbb{R}^{J_m \times K_m}\}$ as well as 1-mode cross-covariance core tensor $\mathcal{G}^{xy}(t) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N}$, hence keeping track of the change of patterns in each loading subspace. Achieving this leads to :

1. compute QR factorization on the augmented loadings :

$$\hat{\mathbf{P}}^{(n)}(t) = \mathbf{U}^{x(n)} \mathbf{V}^{x(n)}, \tag{7.9}$$

where $\mathbf{U}^{x(n)} \in \mathbb{R}^{I_n \times (L_n + \bar{L}_n)}$ and $\mathbf{V}^{x(n)} \in \mathbb{R}^{(L_n + \bar{L}_n) \times (L_n + \bar{L}_n)}$ for $n = 2, ..., N$;

$$\hat{\mathbf{Q}}^{(m)}(t) = \mathbf{U}^{y(m)} \mathbf{V}^{y(m)}, \tag{7.10}$$

$\mathbf{U}^{y(m)} \in \mathbb{R}^{J_m \times (K_m + \bar{K}_m)}$ and $\mathbf{V}^{y(m)} \in \mathbb{R}^{(K_m + \bar{K}_m) \times (K_m + \bar{K}_m)}$ for $m = 2, ..., M$.

2. transform cross-covariance core tensor $\hat{\mathcal{G}}^{xy}(t)$ to get

$$\tilde{\mathcal{G}}^{xy}(t) = \hat{\mathcal{G}}^{xy}(t) \times_1 \mathbf{V}^{y(2)} \times_2 ... \times_{M-1} \mathbf{V}^{y(M)} \times_M \mathbf{V}^{x(2)} \times_{M+1} ... \times_{M+N-2} \mathbf{V}^{x(N)}. \tag{7.11}$$

3. calculate the rank-$(L_2, ..., L_N, K_2, ..., K_M)$ orthogonal Tucker on the transformed core tensor $\tilde{\mathcal{G}}^{xy}(t)$ to get the resulting 1-mode cross-covariance core tensor $\mathcal{G}^{xy}(t)$ for mini-batch t

$$\tilde{\mathcal{G}}^{xy}(t) \approx \mathcal{G}^{xy}(t) \times_1 \mathbf{Z}^{y(2)} \times_2 ... \times_{M-1} \mathbf{Z}^{y(M)} \times_M \mathbf{Z}^{x(2)} \times_{M+1} ... \times_{M+N-2} \mathbf{Z}^{x(N)}, \tag{7.12}$$

where $\mathbf{Z}^{x(n)} \in \mathbb{R}^{(L_n + \bar{L}_n) \times L_n}$, $\mathbf{Z}^{y(m)} \in \mathbb{R}^{(K_m + \bar{K}_m) \times K_m}$ and $\mathcal{G}^{xy}(t) \in \mathbb{R}^{L_2 \times \cdots \times L_N \times K_2 \times \cdots \times K_M}$.

4. compute the loadings :

$$\mathbf{P}^{(n)}(t) = \mathbf{U}^{x(n)} \mathbf{Z}^{x(n)} \in \mathbb{R}^{I_n \times L_n}, \tag{7.13}$$

$$\mathbf{Q}^{(m)}(t) = \mathbf{U}^{y(m)} \mathbf{Z}^{y(m)} \in \mathbb{R}^{J_m \times K_m}. \tag{7.14}$$

For this Step 3, the computational cost is concentrated in the calculation of Tucker on tensor $\tilde{\mathcal{G}}^{xy}(t)$, that is, $\mathcal{O}(R^{N+M-1})$, which is not necessarily an issue since we operate the decomposition on a small scale R rather than a large dimensionality I. The illustration for compressing all the factors at $t = 1$ is shown in Figure 7.5.

In a word, Steps 2 and 3 together are applied to 1-mode cross-covariance core tensor for purpose of updating the loading factors, which contributes in part to the speed-ups from perspective of reducing the "sample" complexity to a low constant level.

Step 4: Projection

Now we update the individual core tensors of input and output from the internal representation of model under the projection of loadings obtained in the last step. These individual core tensors in conjunction with loadings are exploited to produce the final prediction. This procedure for $t = 1$ is shown in Figure 7.6.

More specifically, we begin with reconstruction of the old internal tensor representation $\mathcal{X}_{int}(t-$

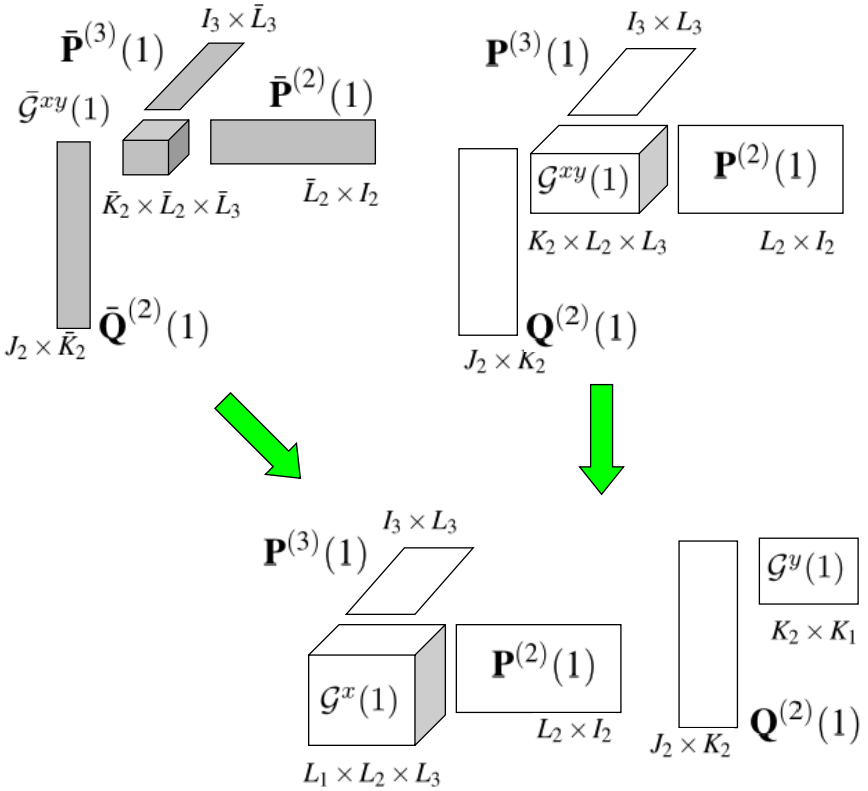


FIGURE 7.6 – The projection step of RHOPLS framework for $t = 1$.

$1) \in \mathbb{R}^{I_0 \times L_2 \times \cdots \times L_N}$ from the latent matrix $\mathbf{T}(t-1)$, the loadings $\{\mathbf{P}^{(n)}(t-1)\}_{n=2}^{N}$ and the core tensor $\mathcal{G}^x(t-1)$ under the projection of current loadings $\{\mathbf{P}^{(n)}(t)\}_{n=2}^{N}$ according to

$$\mathcal{X}_{int}(t-1) = \mathcal{G}^x(t-1) \times_1 \mathbf{T}(t-1) \times_2 \mathbf{P}^{(2)}(t)^{\mathsf{T}}\mathbf{P}^{(2)}(t-1) \times_3 \cdots \times_N \mathbf{P}^{(N)}(t)^{\mathsf{T}}\mathbf{P}^{(N)}(t-1). \tag{7.15}$$

We also reconstruct the incremental internal tensor representation $\bar{\mathcal{X}}_{int}(t) \in \mathbb{R}^{b \times L_2 \times \cdots \times L_N}$ from the core tensor $\check{\mathcal{G}}^x(t)$ and loadings $\{\bar{\mathbf{P}}^{(n)}(t)\}_{n=2}^{N}$ obtained in Step 1 by projecting onto the subspaces of the current loadings $\{\mathbf{P}^{(n)}(t)\}_{n=2}^{N}$, which becomes

$$\bar{\mathcal{X}}_{int}(t) = \check{\mathcal{G}}^x(t) \times_2 \mathbf{P}^{(2)}(t)^{\mathsf{T}}\bar{\mathbf{P}}^{(2)}(t) \times_3 \cdots \times_N \mathbf{P}^{(N)}(t)^{\mathsf{T}}\bar{\mathbf{P}}^{(N)}(t). \tag{7.16}$$

After concatenating $\bar{\mathcal{X}}_{int}(t)$ to $\mathcal{X}_{int}(t-1)$, we get the augmented internal representation $\mathcal{X}_{int}(t) \in \mathbb{R}^{(I_0+b) \times L_2 \times \cdots \times L_N}$. Then, $\mathcal{X}_{int}(t)$ is decomposed using Tucker-1 model [Cichocki et al., 2009] to get the common internal latent matrix $\mathbf{T}_{int}(t) \in \mathbb{R}^{(I_0+b) \times L_1}$ and the desired core tensor $\mathcal{G}^x(t) \in \mathbb{R}^{L_1 \times L_2 \times \cdots \times L_N}$ as

$$\mathcal{X}_{int}(t) \approx \mathcal{G}^x(t) \times_1 \mathbf{T}_{int}(t), \tag{7.17}$$

where $\mathbf{T}_{int}(t)$ is thereafter truncated to keep the very last I_0 rows, leading to the common $\mathbf{T}(t) \in \mathbb{R}^{I_0 \times L_1}$ for the purpose of internal representation reconstruction of the model in the subsequent mini-batch.

Similarly, we have $\mathcal{Y}_{int}(t) \in \mathbb{R}^{(I_0+b) \times K_2 \times \cdots \times K_M}$ and $\mathcal{G}^y(t) \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_M}$ for the output side. By employing our proposed projection strategy, the dominating cost of this step is substantially cut down from $\mathcal{O}(\max(I_0 I^N, I^{N+1}))$ to $\mathcal{O}(\max(I_0 R^N, R^{N+1}))$ with $R \ll I$, because we equivalently represent the model and calculate the new factors in terms of projected internal tensors that lie in the tensor space with small scale R but large original I. Step 4 is responsible for efficiently updating the individual core tensors, which also contributes in part to the total acceleration by reducing the "sample" complexity.

In summary, Step 0 is a preprocessing step that executes only once, while the other steps are conducted repeatedly for each new mini-batch. The whole framework of RHOPLS is illustrated again in Figure 7.7 and whole procedure of RHOPLS is outlined in Algorithm 7. With all the extracted loadings and individual core tensors in hand, the prediction can be made similar to the one in HOPLS [Zhao et al., 2011, 2013a]. It is important to note that RHOPLS demands a minimum space complexity in the sense that only a small number of factors, dominating by $\mathcal{O}(\max(R^{N+M-2}))$ of cross-covariance core tensor, are needed to be stored to represent the running model.

7.3 Experimental Results

In our experiments, the root mean squares of prediction (RMSEP) [Kim et al., 2005] as well as the Q index [Luo et al., 2015] are used to quantitatively gauge the predictive performance

Algorithm 7 Recursive Higher-order Partial Least Squares (RHOPLS)

Input: old parameters $\mathbf{T}(t-1)$, $\{\mathbf{P}^{(n)}(t-1)\}_{n=2}^{N}$, $\{\mathbf{Q}^{(m)}(t-1)\}_{m=2}^{M}$ and $\mathcal{G}^x(t-1), \mathcal{G}^y(t-1)$, new mini-batch pair $\{\mathcal{X}(t) \in \mathbb{R}^{b \times I_2 \times \cdots \times I_N}, \mathcal{Y}(t) \in \mathbb{R}^{b \times J_2 \times \cdots \times J_M}\}$

Output: new parameters $\mathbf{T}(t)$, $\{\mathbf{P}^{(n)}(t)\}_{n=2}^{N}$, $\{\mathbf{Q}^{(m)}(t)\}_{m=2}^{M}$ and $\mathcal{G}^x(t), \mathcal{G}^y(t)$

Center the new data pair $\{\mathcal{X}(t), \mathcal{Y}(t)\}$ accumulatively

/* **step 0**: **Initial Approximation (only executed once on the initial data at t=0)** */

$\mathcal{C}(0) \leftarrow \langle \mathcal{X}(0), \mathcal{Y}(0) \rangle_{\{1:1\}}$

Rank-$(K_2, ..., K_M, L_2, ..., L_N)$ orthogonal Tucker of $\mathcal{C}(0)$ by HOOI as (7.3)

Apply HOSVD to estimate $\mathbf{T}(0)$, $\mathcal{G}^x(0)$ and $\mathcal{G}^y(0)$

/* **Step 1**: **Incremental Approximation** */

Rank-$(\bar{L}_2, ..., \bar{L}_N)$ orthogonal Tucker decomposition of $\mathcal{X}(t)$ except the first mode by HOOI as (7.4)

Rank-$(\bar{K}_2, ..., \bar{K}_M)$ orthogonal Tucker decomposition of $\mathcal{Y}(t)$ except the first mode by HOOI as (7.5)

$\bar{\mathcal{G}}^{xy}(t) \leftarrow \langle \check{\mathcal{G}}^x(t), \check{\mathcal{G}}^x(t) \rangle_{\{1:1\}}$

/* **Step 2**: **Expansion** */

$\hat{\mathbf{P}}^{(n)}(t) \leftarrow [\mathbf{P}^{(n)}(t-1) \;\; \bar{\mathbf{P}}^{(n)}(t)]$ and $\hat{\mathbf{Q}}^{(m)}(t) \leftarrow [\mathbf{Q}^{(m)}(t-1) \;\; \bar{\mathbf{Q}}^{(m)}(t)]$

Append $\bar{\mathcal{G}}^{xy}(t)$ to $\mathcal{G}^{xy}(t-1)$ in a super block diagonal way to get the $\hat{\mathcal{G}}^{xy}(t)$, with other entries being zeros

/* **Step 3**: **Compression** */

QR factorization $\hat{\mathbf{P}}^{(n)}(t) \rightarrow \mathbf{U}^{x(n)}\mathbf{V}^{x(n)}$ and $\hat{\mathbf{Q}}^{(m)}(t) \rightarrow \mathbf{U}^{y(m)}\mathbf{V}^{y(m)}$

$\tilde{\mathcal{G}}^{xy}(t) \leftarrow \hat{\mathcal{G}}^{xy}(t) \times_1 \mathbf{V}^{y(2)} \times_2 ... \times_{M-1} \mathbf{V}^{y(M)} \times_M \mathbf{V}^{x(2)} \times_{M+1} ... \times_{M+N-2} \mathbf{V}^{x(N)}$

Rank-$(L_2, ..., L_N, K_2, ..., K_M)$ orthogonal Tucker decomposition of $\tilde{\mathcal{G}}^{xy}(t)$ by HOOI as (7.12)

$\mathbf{P}^{(n)}(t) \leftarrow \mathbf{U}^{x(n)}\mathbf{Z}^{x(n)}$ and $\mathbf{Q}^{(m)}(t) \leftarrow \mathbf{U}^{y(m)}\mathbf{Z}^{y(m)}$

/* **Step 4**: **Projection** */

$\mathcal{X}_{int}(t-1) \leftarrow \mathcal{G}^x(t-1) \times_1 \mathbf{T}(t-1) \times_2 \mathbf{P}^{(2)}(t)^{\mathsf{T}}\mathbf{P}^{(2)}(t-1) \times_3 \cdots \times_N \mathbf{P}^{(N)}(t)^{\mathsf{T}}\mathbf{P}^{(N)}(t-1)$

$\mathcal{Y}_{int}(t-1) \leftarrow \mathcal{G}^y(t-1) \times_1 \mathbf{T}(t-1) \times_2 \mathbf{Q}^{(2)}(t)^{\mathsf{T}}\mathbf{Q}^{(2)}(t-1) \times_3 \cdots \times_M \mathbf{Q}^{(M)}(t)^{\mathsf{T}}\mathbf{Q}^{(M)}(t-1)$

$\bar{\mathcal{X}}_{int}(t) \leftarrow \check{\mathcal{G}}^x(t) \times_2 \mathbf{P}^{(2)}(t)^{\mathsf{T}}\bar{\mathbf{P}}^{(2)}(t) \cdots \times_N \mathbf{P}^{(N)}(t)^{\mathsf{T}}\bar{\mathbf{P}}^{(N)}(t)$

$\bar{\mathcal{Y}}_{int}(t) \leftarrow \check{\mathcal{G}}^y(t) \times_2 \mathbf{Q}^{(2)}(t)^{\mathsf{T}}\bar{\mathbf{Q}}^{(2)}(t) \cdots \times_M \mathbf{Q}^{(M)}(t)^{\mathsf{T}}\bar{\mathbf{Q}}^{(M)}(t)$

Concatenate $\{\bar{\mathcal{X}}_{int}(t), \bar{\mathcal{Y}}_{int}(t)\}$ to $\{\mathcal{X}_{int}(t-1), \mathcal{Y}_{int}(t-1)\}$

$$\mathcal{X}_{int}(t) \leftarrow \begin{bmatrix} \mathcal{X}_{int}(t-1) \\ \bar{\mathcal{X}}_{int}(t) \end{bmatrix} \in \mathbb{R}^{(I_0+b) \times L_2 \times \cdots \times L_N}$$

$$\mathcal{Y}_{int}(t) \leftarrow \begin{bmatrix} \mathcal{Y}_{int}(t-1) \\ \bar{\mathcal{Y}}_{int}(t) \end{bmatrix} \in \mathbb{R}^{(I_0+b) \times K_2 \times \cdots \times K_M}$$

Rank-(L_1) orthogonal Tucker-1 decomposition of $\mathcal{X}(t)$ by HOSVD as (7.17)

$\mathcal{G}^y(t) \leftarrow \mathcal{Y}_{int}(t) \times_1 \mathbf{T}_{int}(t)^{\mathsf{T}}$

$\mathbf{T}(t) \leftarrow \mathbf{T}_{int}(t)(b+1 : I_0+b, :)$

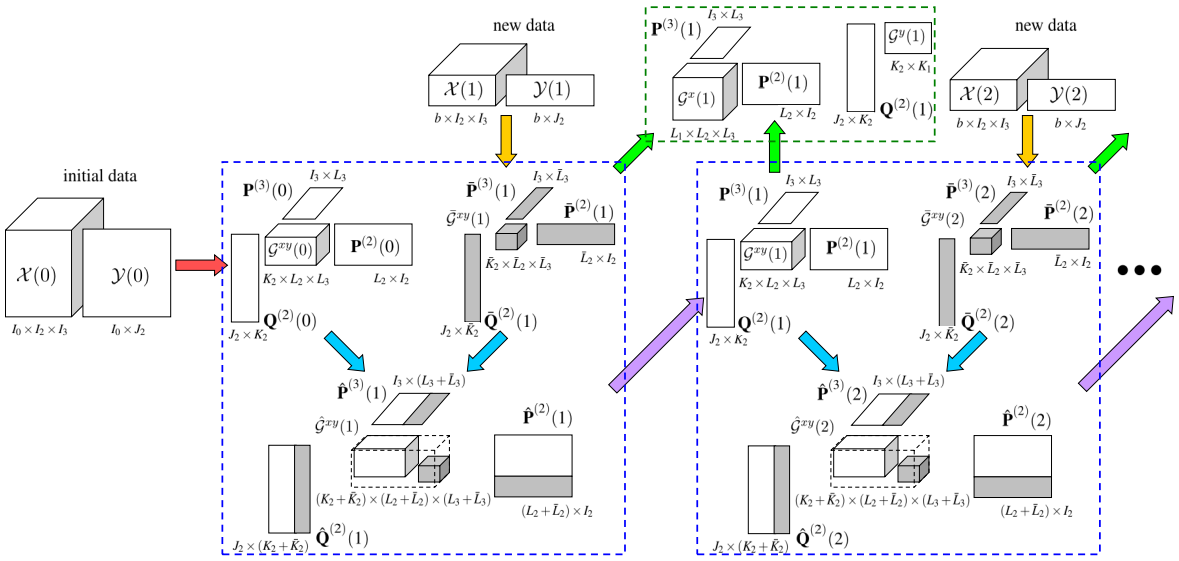


FIGURE 7.7 – The whole RHOPLS scheme.

of our approach. We recorded the CPU time for learning the new mini-batch for all recursive methods, and we also recorded CPU time for batch methods using the entire training set. We compared RHOPLS with NPLS [Bro, 1996], RNPLS [Eliseyev and Aksenova, 2013], HOPLS [Zhao et al., 2013a] and IHOPLS [Hou and Chaib-draa, 2016] on general tensorial sequences with no special structures assumed in contrast to spatio-temporal data.

To show the robustness, each sequence was randomly shuffled into 10 instances for evaluation, because our framework is designed for the general setting of sequence though it could be applied to the stream [Sun et al., 2008]. One half of the shuffled sequence served as training set while the remaining half was used for test. The optimal hyperparameters of all methods were determined by cross-validation. In addition, the convergence criterion and the maximum iteration number of RNPLS for estimating the loadings were set to 10^{-5} and 40 respectively, as a result of the optimal balance between accuracy and speed. For simplicity, we assumed the initial d-ranks are equal to the incremental d-ranks in RHOPLS.

7.3.1 Utrecht Multi-Person Motion Database (UMPM)

We first tested RHOPLS on the Utrecht Multi-Person Motion (UMPM) benchmark [Aa et al., 2011], which provides the simultaneous recordings of video sequences and $3D$ ground truth positions of human natural motions in daily life activities. For our test, the algorithm input was an intensity image sequence from the front camera with a downsized resolution of 24×32 pixels at 25fps, taking the form of a 3rd-order predictor tensor (i.e., *frames* $\times$ *width* $\times$ *height*). On the other hand, the corresponding output containing $3D$ positions of 37 reflective markers can be represented as a 3rd-order tensor (i.e., *samples* $\times$ $3D$ *positions* $\times$ *markers*).

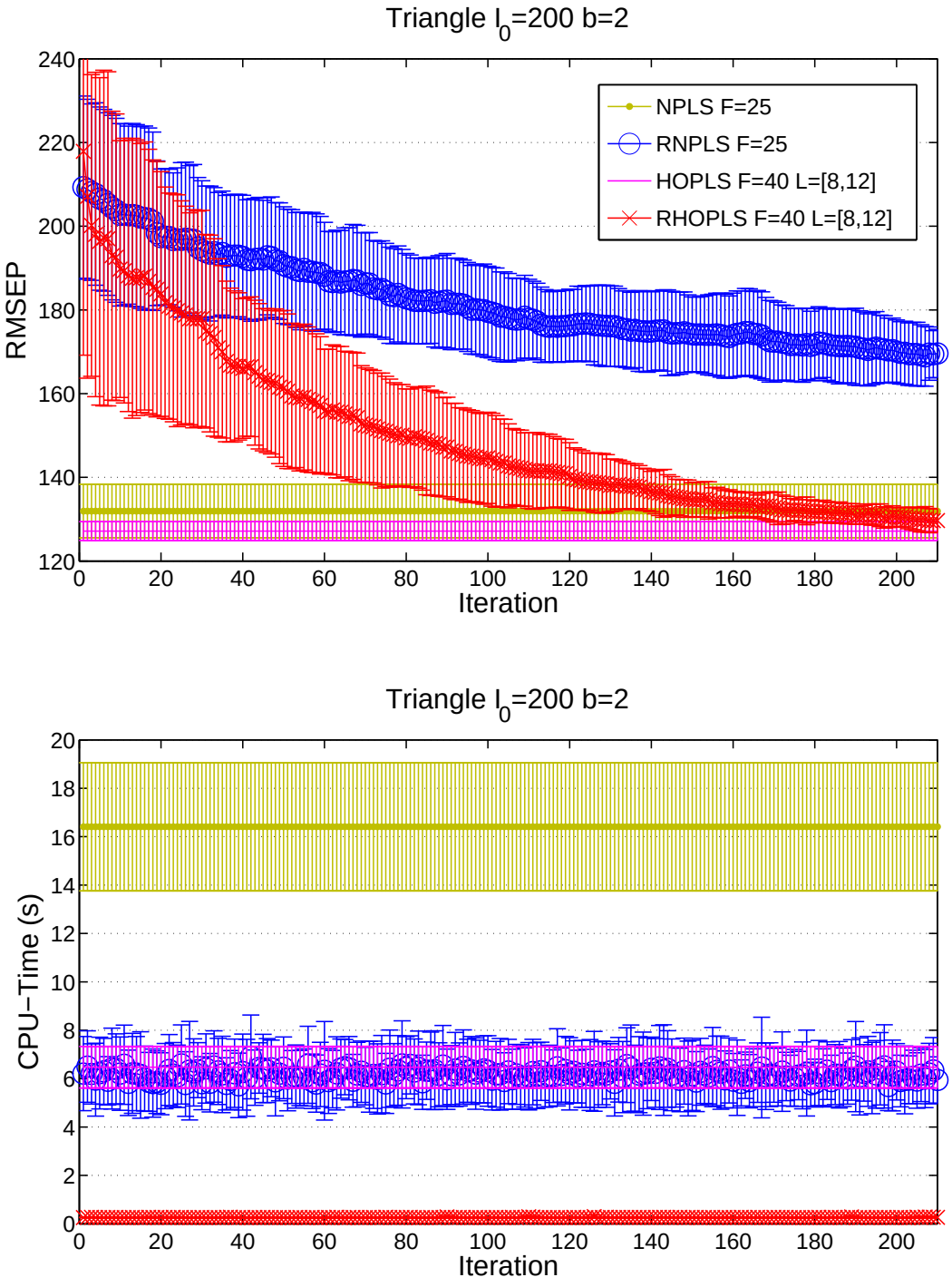


FIGURE 7.8 – For "triangle" scenario, learning error and learning time versus the iteration.

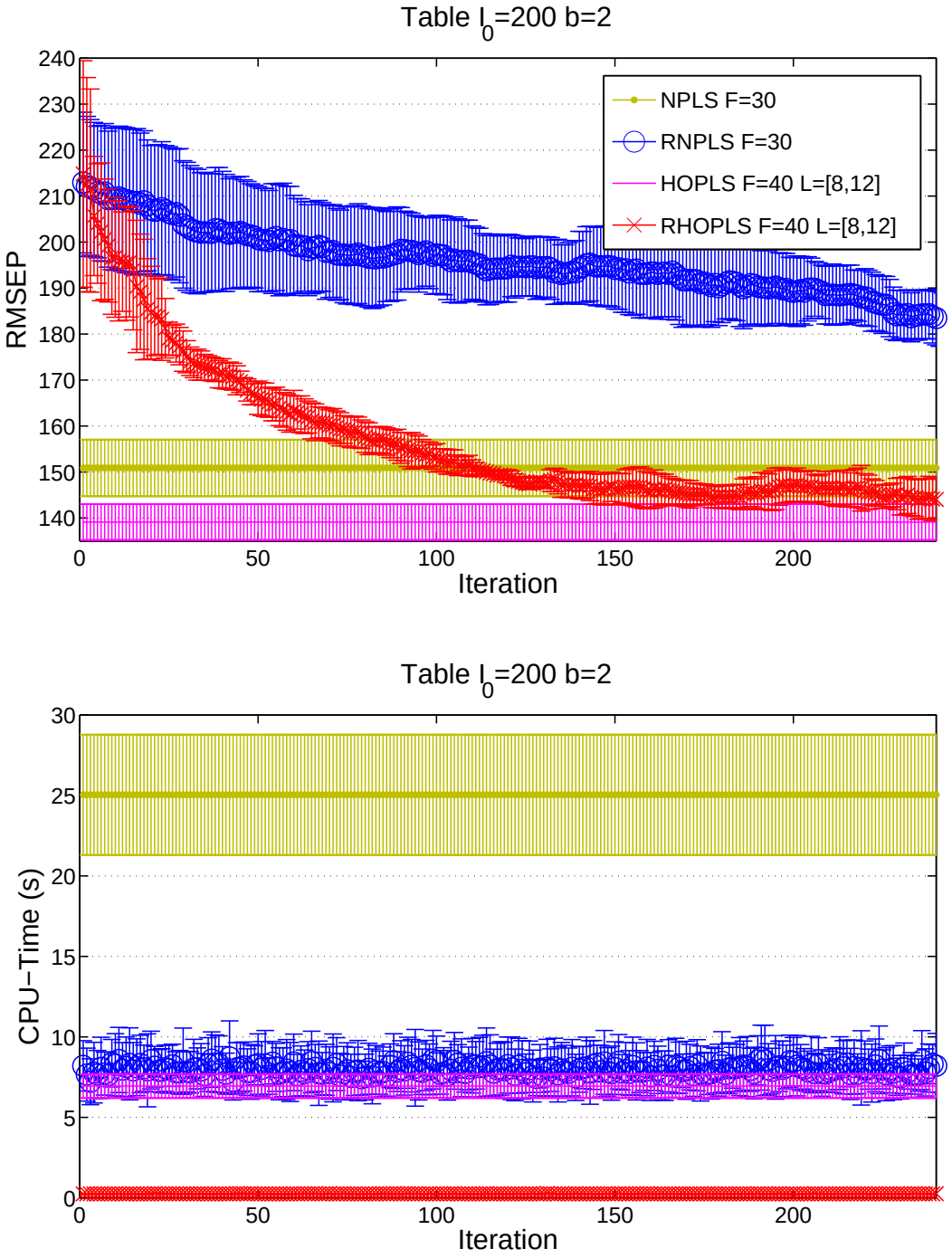


FIGURE 7.9 – For "table" scenario, learning error and learning time versus the iteration.

TABLE 7.1 – Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time with $I_0 = 200$ and $b = 2$ on UMPM data.

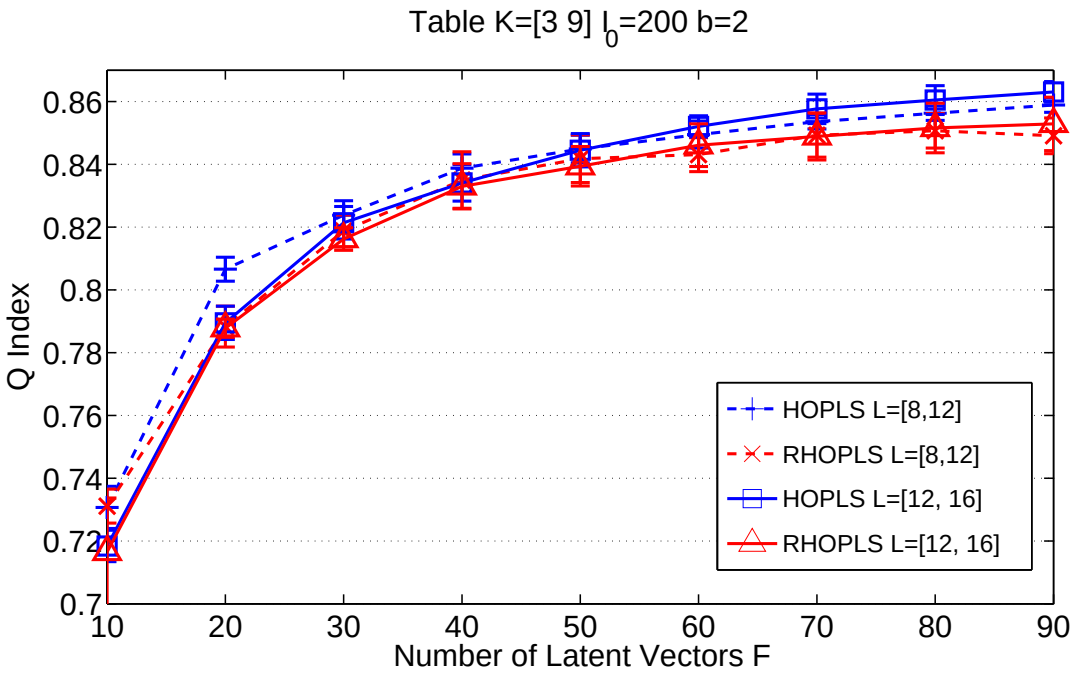
Scenario "triangle" Input Dimension= $1230 \times 24 \times 32$				
Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
NPLS	$F = 25$	.8431 (.0070)	132.0 (6.3)	16.0 (2.6)
RNPLS	$F = 25$, $\gamma = 1$	.8011 (.0092)	169.4 (8.7)	6.17 (1.25)
HOPLS	$F = 40, L = [8, 12], K = [3, 9]$	**.8480** (.0019)	**127.8** (2.2)	6.47 (0.34)
	$F = 40, L = [12, 16], K = [3, 9]$	**.8462** (.0016)	**129.0** (2.3)	5.94 (0.22)
IHOPLS	$F = 40, L = [12, 16], K = [3, 9]$	.7034 (.0104)	248.5 (8.8)	4.48 (0.45)
RHOPLS	$F = 40, L = [8, 12], K = [3, 9]$	**.8453** (.0035)	**129.6** (2.8)	**0.25** (0.01)
	$F = 40, L = [12, 16], K = [3, 9]$	**.8430** (.0039)	**132.3** (3.0)	**0.26** (0.01)
Scenario "table" Input Dimension= $1430 \times 24 \times 32$				
Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
NPLS	$F = 30$	.8240 (.0067)	151.2 (6.0)	25.4 (2.7)
RNPLS	$F = 30$, $\gamma = 1$	.7865 (.0083)	184.7 (7.1)	8.06 (1.47)
HOPLS	$F = 40, L = [8, 12], K = [3, 9]$	**.8388** (.0045)	**139.1** (3.8)	6.94 (0.38)
	$F = 40, L = [12, 16], K = [3, 9]$	**.8341** (.0058)	**142.7** (4.8)	6.04 (0.80)
IHOPLS	$F = 40, L = [12, 16], K = [3, 9]$	.7393 (.0069)	227.2 (8.1)	4.53 (0.42)
RHOPLS	$F = 40, L = [8, 12], K = [3, 9]$	**.8304** (.0058)	**145.3** (5.7)	**0.24** (0.01)
	$F = 40, L = [12, 16], K = [3, 9]$	**.8294** (.0072)	**146.8** (6.0)	**0.25** (0.01)

The averaged predictive performance as well as learning time are compared in Table 7.1. As we can see, RHOPLS achieves highly comparable accuracy with the batch HOPLS but is much faster. Spectacularly, the speed-ups of RHOPLS over RNPLS and NPLS for "table" scenario are more than 30 and 100 times, the acceleration rates are even higher with the larger numbers of latent vectors. Figure 7.8 and Figure 7.9 show that, for both "triangle" and "table" scenarios, the predictive error of RHOPLS keeps decreasing at a faster rate as the iteration goes on, while the CPU cost remains as a low constant trend over time.

In Figure 7.10, we can also observe that the prediction accuracy of RHOPLS stays very close to that of HOPLS as the number of latent vectors ranges from 10 to 90 for different loadings. However, the CPU time just slightly increases from 0.19s to 0.38s for RHOPLS with input loading [8, 12], which is in contrast to that of HOPLS from 1.79s to 19.22s, indicating the superior scalability of the RHOPLS with increasing number of latent vectors (d-ranks). The Q is given in Figure 7.11 with varying numbers of initial samples and mini-batch sizes. We may notice that only 50 initial samples, which is 8% of the whole training data, will suffice to guarantee a reasonably good result, i.e., nearly 0.78 for mini-batch of 2.

7.3.2 Neurotycho Electrocorticography Dataset (ECoG)

In this section, the tests were carried out on a benchmark tensor regression application which consists of decoding limb movements from monkey's brain signals using Neurotycho food tracking Electrocorticography (ECoG) dataset [Chao et al., 2010]. ECoG data contains a 15 minute-long recording and we downsampled motion data to different frequencies, producing



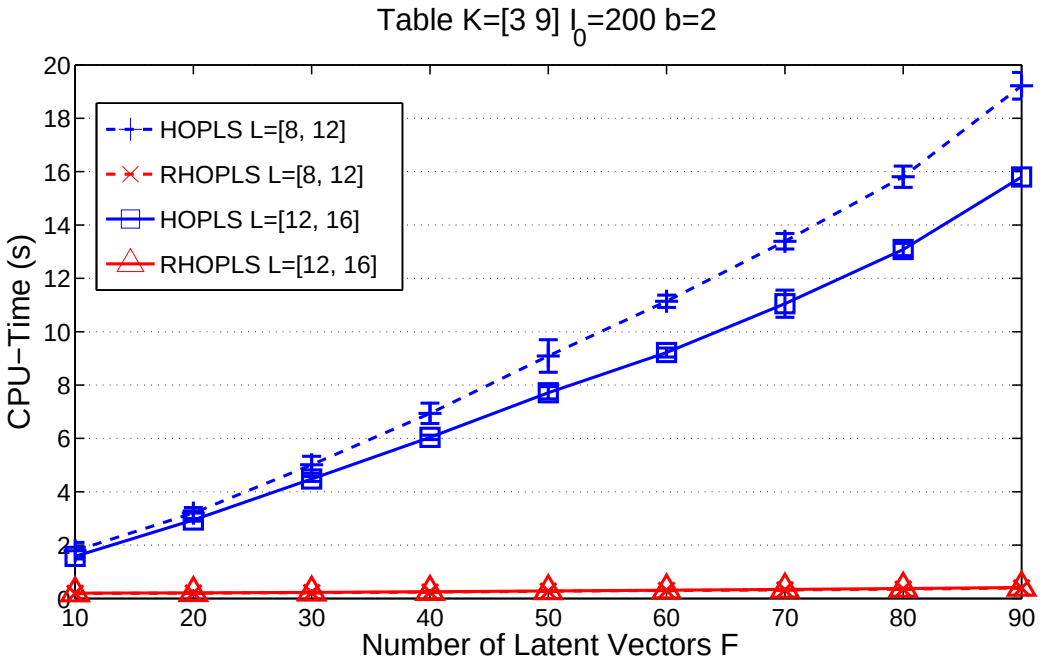


FIGURE 7.10 – The accuracy and learning time versus the number of latent vectors.

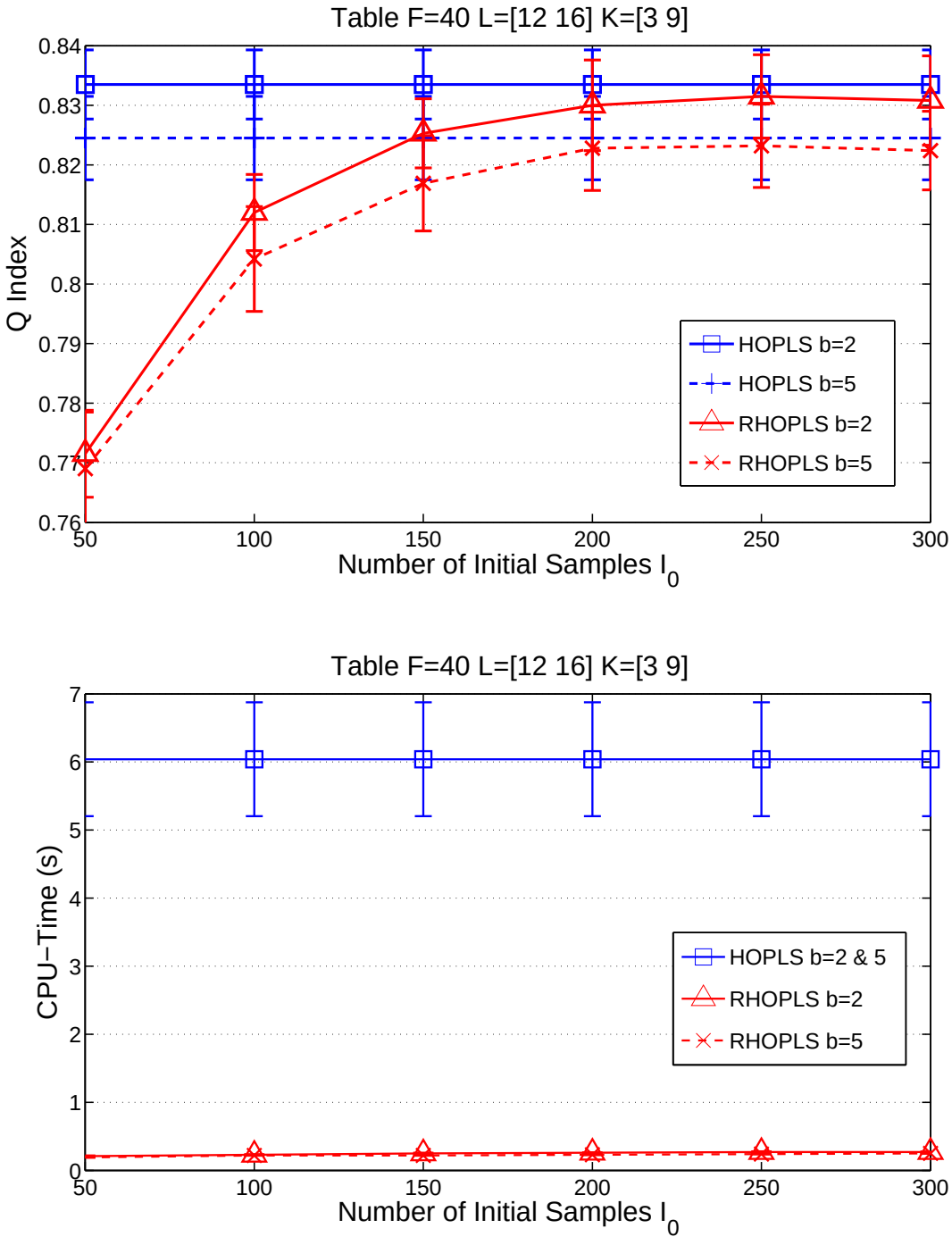


FIGURE 7.11 – The accuracy and learning time versus the number of initial samples.

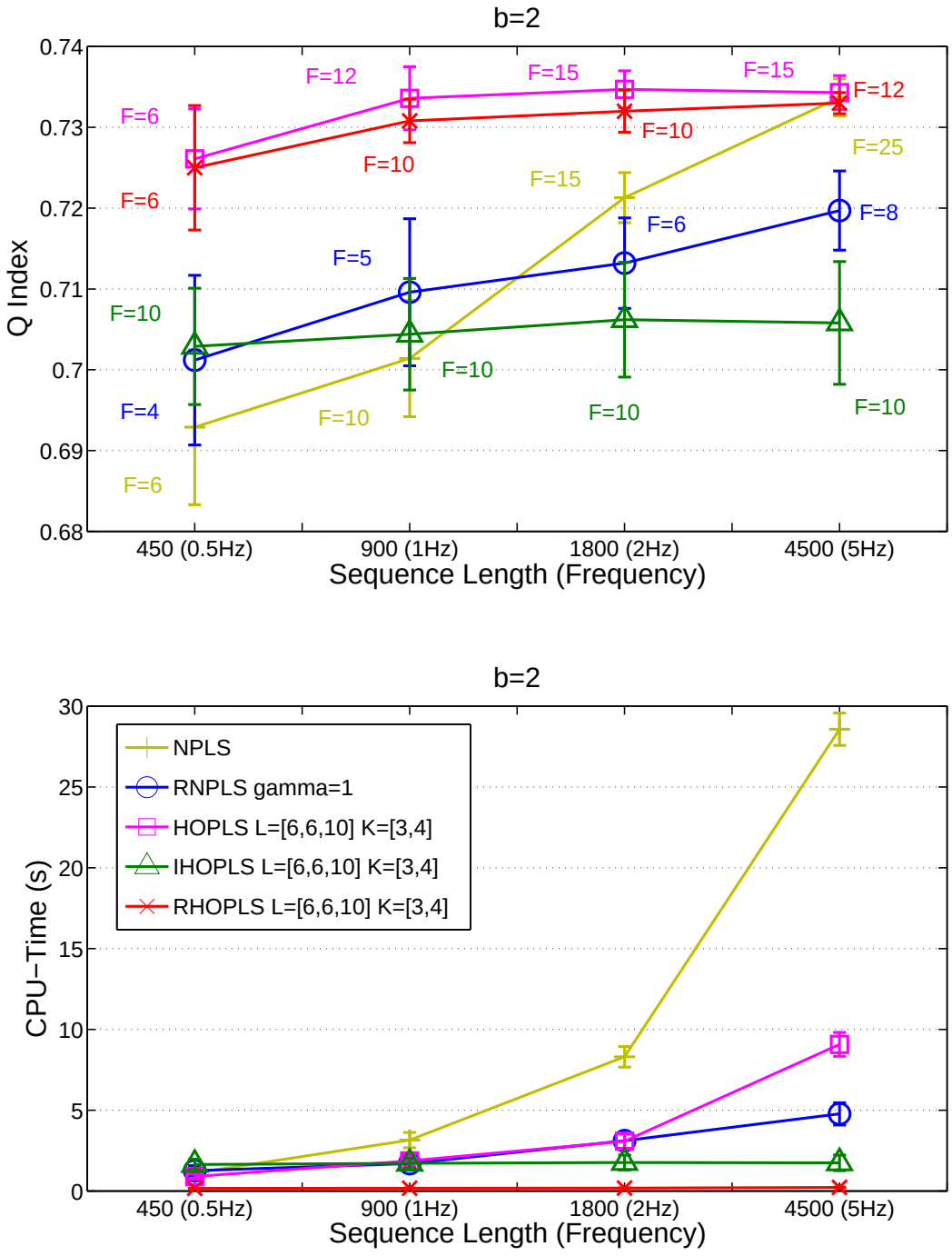


FIGURE 7.12 – Performance versus sequence length (frequency).

TABLE 7.2 – Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time with $I_0 = 20\%$ of the training set and $b = 2$ on ECoG data.

Input Dimension= $900 \times 10 \times 10 \times 16$ and Frequency=1Hz				
Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
NPLS	$F = 10$	.7014 (.0072)	33.5 (0.7)	3.15 (0.46)
RNPLS	$F = 5$, $\gamma = 1$	.7096 (.0071)	32.6 (0.6)	1.70 (0.35)
HOPLS	$F = 12$,$L = [6, 6, 10]$, $K = [3, 4]$	**.7336** (.0039)	**29.9** (0.4)	1.86 (0.34)
IHOPLS	$F = 10$,$L = [6, 6, 10]$, $K = [3, 4]$	.7044 (.0082)	33.1 (0.9)	1.64 (0.32)
RHOPLS	$F = 10$,$L = [6, 6, 10]$, $K = [3, 4]$	**.7308** (.0027)	**30.3** (0.3)	**0.16** (0.01)
Input Dimension= $4500 \times 10 \times 10 \times 16$ and Frequency=5Hz				
Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
NPLS	$F = 25$	.7338 (.0023)	29.9 (0.4)	28.6 (1.0)
RNPLS	$F = 8$, $\gamma = 1$	.7197 (.0049)	31.6 (0.5)	4.87 (0.69)
HOPLS	$F = 10$,$L = [6, 6, 10]$, $K = [3, 4]$	**.7343** (.0011)	**29.7** (0.2)	9.07 (0.74)
IHOPLS	$F = 10$,$L = [6, 6, 10]$, $K = [3, 4]$	.7058 (.0086)	33.0 (0.8)	1.74 (0.48)
RHOPLS	$F = 10$,$L = [6, 6, 10]$, $K = [3, 4]$	**.7330** (.0012)	**30.0** (0.2)	**0.20** (0.02)

various lengths of observations with different levels of overlapped features. As for the input, the wavelet transformed ECoG signal was represented as a 4-order tensor (i.e., *samples* $\times$ *time* $\times$ *frequency* $\times$ *channel*). A 3-order tensor of $3D$ movement distances of the monkey's limb on 4 markers was used as the output. The best results are obtained with various numbers of latent vectors adapting to the corresponding increasing lengths of sequences.

In Table 7.2, RHOPLS again performs consistently better than NPLS, RNPLS, IHOPLS for the 4-order's input situation. In Figure 7.12, RHOPLS maintains nearly the same predictability with HOPLS in the settings of both low frequency (difficult case) and high frequency (easy case). Meanwhile, RHOPLS remains as really low constant CPU time regardless of the length of sequence, and exhibits high speed-up rates over RNPLS, i.e., 24 times faster at 4500, and overall nearly 10 times faster than IHOPLS, while NPLS and HOPLS are almost useless in fast time-critical applications.

For visualization, an example of the observed and the predicted trajectories of monkey's hand at the frequency of 1Hz is given in Figure 7.13. Due to the clarity of demonstration, we just compare our RHOPLS with the best sequential method RNPLS, the best batch method HOPLS as well as the ground truth. As expected, RHOPLS archives the similar accuracy as that of HOPLS while it is much better than RHOPLS.

7.3.3 Multimodal Human Action Database (MHAD)

We also validated our approach on the Berkeley MHAD data [Ofli et al., 2013] that contains temporally synchronized video sequences and human motion capture data. We selected two most challenging actions that involve dynamics in both upper and lower extremities of

human body, namely "jumping in place" and "bending". All the recordings were taken from the C1 camera of the L1 cluster. Additionally, the total 5 short recordings per subject were concatenated into a long recording for each action at 22fps. In Table 7.3, we can see that RHOPLS achieves almost the best accuracies with much faster speed, and all other methods scale poorly with d-ranks, especially for CP-based methods, while RHOPLS is not sensitive to any particular subject or action and large d-ranks.

7.3.4 Comprehensive Climate Dataset (CCDS)

Finally, we verified our approach on the application of climate data analysis. The data was taken from Comprehensive Climate Dataset (CCDS) [Lozano et al., 2009] that consists of a collection of climate records of North America. Spanning from 1990 to 2001, this data provides monthly climate-related observations of 17 variables with 125 different locations. As it turns out, the time series data correspond to a 3rd-order tensor of size $156 \times 125 \times 17$. Table 7.4 illustrates an excellent trade-off between accuracy and CPU time of RHOPLS. For instance, it takes only 0.21s to achieve a quite reasonable RMSEP of 0.8681 in contrast to 0.8674 of GREEDY with more than 28s. In contrast, we are able to obtain the nearly best result of 0.8341 at only an little extra cost of CPU time if there is enough initial samples.

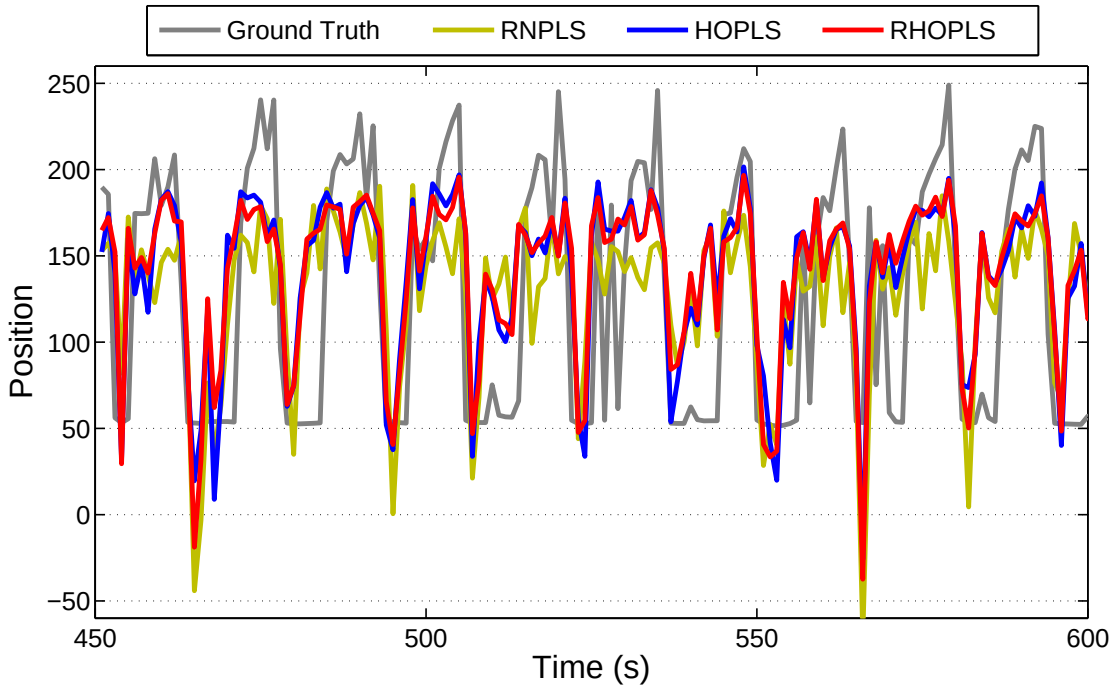


FIGURE 7.13 – An example of ground truth (150s time window) and the trajectories predicted by RHOPLS, HOPLS and RNPLS for Z-coordinate of the monkey's hand.

7.4 Discussion

The drastic accelerations of our RHOPLS are realized in two stages, the first, due to Step 1, focuses on the reduction of "dimensionality" complexity. On the basis of the first stage, the second stage (i.e., Steps 2,3 and 4) concentrates on making the low constant "sample" complexity possible. The overall speed-ups stem from directly updating the set of factors (regression coefficients) in lightweight manner at a small-scale factor (feature) level instead of the raw data level, such that the relatively expensive eigenvector-style calculations are able to execute a lot faster on the factor scale.

TABLE 7.3 – Performance comparison of NPLS, RNPLS, HOPLS, IHOPLS and RHOPLS for the averaged Q, RMSEP and learning time for $L = [12, 16]$, $K = [3, 10]$, $b = 2$ on MHAD data.

Jump (500-600)					
Subject	Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
S6 (F)	NPLS	$F = 20$	.7374 (.0035)	179.7 (2.5)	6.63 (.70)
	RNPLS	$F = 5$, $\gamma = 1$	.7271 (.0055)	186.8 (4.3)	3.01 (.47)
	HOPLS	$F = 20$	**.7396** (.0032)	**178.1** (2.4)	2.87 (.24)
	IHOPLS	$F = 20$, $N_{max} = 40$	.7138 (.0038)	194.0 (3.0)	2.34 (.16)
	RHOPLS	$F = 20$, $I_0 = 50$	.7303 (.0035)	184.5 (2.6)	**0.18** (.00)
		$F = 20$, $I_0 = 100$	**.7376** (.0032)	**179.5** (2.4)	**0.18** (.00)
		$F = 20$, $I_0 = 150$	**.7394** (.0033)	**178.2** (2.5)	**0.19** (.01)
S10 (M)	NPLS	$F = 15$	.7368 (.0044)	183.8 (2.3)	5.53 (.95)
	RNPLS	$F = 5$, $\gamma = 1$	.7255 (.0043)	191.6 (2.7)	2.56 (.45)
	HOPLS	$F = 20$	**.7406** (.0044)	**181.1** (2.4)	3.00 (.25)
	IHOPLS	$F = 20$, $N_{max} = 40$	.7086 (.0063)	200.4 (3.5)	2.44 (.13)
	RHOPLS	$F = 20$, $I_0 = 50$	.7314 (.0052)	188.6 (2.3)	**0.19** (.01)
		$F = 20$, $I_0 = 100$	**.7383** (.0040)	**182.6** (1.8)	**0.19** (.01)
		$F = 20$, $I_0 = 150$	**.7403** (.0042)	**181.2** (2.3)	**0.19** (.01)

Bend (1300-1500)					
Subject	Methods	Hyperparameter	Q	RMSEP	CPU Time (s)
S6 (F)	NPLS	$F = 40$	.7280 (.0023)	178.7 (1.2)	53.2 (3.4)
	RNPLS	$F = 6$, $\gamma = 1$	.6570 (.0438)	226.2 (9.8)	3.11 (.55)
	HOPLS	$F = 40$	**.7299** (.0025)	**177.5** (1.1)	6.97 (.11)
	IHOPLS	$F = 40$, $N_{max} = 40$	.6705 (.0063)	208.6 (4.3)	5.32 (.28)
	RHOPLS	$F = 40$, $I_0 = 100$	.7170 (.0025)	186.1 (1.6)	**0.23** (.01)
		$F = 40$, $I_0 = 200$	**.7254** (.0021)	**180.3** (0.8)	**0.24** (.01)
		$F = 40$, $I_0 = 300$	**.7278** (.0023)	**178.8** (0.9)	**0.26** (.01)
S10 (M)	NPLS	$F = 40$	.7469 (.0024)	168.3 (1.8)	50.1 (4.8)
	RNPLS	$F = 6$, $\gamma = 1$	.6878 (.0491)	208.0 (10.3)	2.68 (.87)
	HOPLS	$F = 40$	**.7491** (.0024)	**166.8** (1.8)	6.81 (.71)
	IHOPLS	$F = 40$, $N_{max} = 40$	.6956 (.0043)	193.6 (3.3)	5.14 (.21)
	RHOPLS	$F = 40$, $I_0 = 100$	.7338 (.0054)	176.9 (2.8)	**0.22** (.01)
		$F = 40$, $I_0 = 200$	**.7445** (.0027)	**169.8** (2.1)	**0.23** (.01)
		$F = 40$, $I_0 = 300$	**.7472** (.0027)	**168.1** (2.1)	**0.25** (.01)

TABLE 7.4 – Forecasting performance for $lag = 3$, trained with 50% of all the time series, $b = 1$ on CCDS data.

Methods	Rank and Hyperparameter	RMSEP	CPU Time (s)
GREEDY	ORTHO	0.8674	28.25
NPLS	$F = 10$	**0.8296**	18.13
RNPLS	$F = 5$ $\gamma = 0.9$	0.9295	7.11
HOPLS	$F = 8$ $L = [45, 14]$ $K = [15, 14]$	0.8367	10.05
IHOPLS	$F = 4$ $L = [15, 14]$ $K = [5, 14]$ $N_{max} = 20$	0.8700	2.77
	$F = 4$ $L = [45, 14]$ $K = [15, 14]$ $N_{max} = 20$	0.8449	5.20
RHOPLS	$F = 8$ $L = [15, 14]$ $K = [5, 14]$ $I_0 = 25$	0.8681	**0.21**
	$F = 8$ $L = [45, 14]$ $K = [15, 14]$ $I_0 = 15$	0.8699	0.85
	$F = 8$ $L = [45, 14]$ $K = [15, 14]$ $I_0 = 25$	0.8400	0.89
	$F = 8$ $L = [45, 14]$ $K = [15, 14]$ $I_0 = 35$	**0.8341**	0.92

Note that the incremental SVD-based decomposition tool in [O'Hara, 2010] only considers input-side data and cannot be directly applied to the tensor regression. We like to stress that the concepts of tensor decomposition tool and tensor regression are fundamentally different in terms of the targeted goal, algorithm and applications. Here, we proposed to integrate this tool as one part of our whole PLS-based framework to extract loadings. All steps in RHOPLS are essentially important and contribute to the overall speed-ups from different aspects and thus should not be treated separately.

7.5 Conclusion

In this chapter, we presented our general tensor regression framework RHOPLS. Compared it to other tensor regression approaches, we have in particular explained how it offers a series of computationally advantageous operations that effectively incorporate the incremental information from new mini-batch into the previous model approximation. We verified significant speed-ups and high predictability of RHOPLS on a variety of real-life applications. For future work, in connection with this chapter, our research will focus on how to adaptively select the rank hyperparameters each time a new mini-bath arrives.

Up to this point, all the tensor regression models in previous chapters involve only single input or output tensor. In the next chapter, we will address the task of integrating multiple data tensors.

Chapitre 8

Partial Least Squares Regression : A Kernel-based Multiblock Tensor Approach

In this chapter, a generalized nonlinear tensor regression framework, called kernel-based multiblock tensor PLS regression (KMTPLS) [Hou et al., 2016], is introduced. This chapter shows how KMTPLS integrates the information from multiple tensorial data sources and unifies the single and multiblock tensor regAression into one model using both common and discriminative features. Then, it shows how KMTPLS experimental results on real data benefits when applied to the real-world tensor regression task in computer vision.

8.1 Introduction

Several variants of tensor regression models have been previously investigated, all of which involve only a single source of tensorial data. Nevertheless, recent advancing technologies have been producing massive tensorial data streams from multiple sources or modalities that couple in a common domain, and thus can be analyzed jointly, e.g., the linked EEG and fMRI from neuroimaging data analysis.

Most of studies have focused on the single-block situation and the predictability is always restricted due to the limited amount of information contained in the single data source. If we simply apply the above methods and average the predictions among the blocks, then the obtained result will not outperform the best singleton case, since the common features of all data blocks are neglected. Such progress highlights a growing need for the development and application of tensor regression techniques by considering multiple predictor and response tensor blocks.

For this purpose, Smilde et al. [2000] proposed the multiway multiblock covariates regression

(MMCR) model, to quantitatively analyze a collection of predictor and response tensor blocks. Generally, MMCR handles all the predictor blocks in a hierarchical way so that a super latent factor matrix is defined based on all the individual latent factor matrices to make the final prediction. However, MMCR may suffer from low predictive ability due to the restriction of linearity, and thus is inadequate in terms of predictability in a situation where the nonlinear dependencies exist between the response and predictor tensor blocks. Moreover, Westerhuis et al. [1998] suggested that multilinear multiblock PLS like MMCR is equivalent to PLS model with all variables combined into a large $\mathcal{X}$-block in terms of predictive ability. Besides, the developed algorithm for solving MMCR is based on an alternating least square (ALS) style approach [Carroll and Chang, 1970], which is known to be a suboptimal procedure with slow convergence rate.

To address the limitations as those of MMCR, we present in this chapter a new generalized nonlinear tensor regression framework, namely kernel-based multiblock tensor partial least squares (KMTPLS) [Hou et al., 2016], that serves as an extension to KTPLS [Zhao et al., 2013b] by incorporating the kernel concept into the context of multiblock tensor regression. As for the implementation, our algorithm is also extended from the NIPALS-PLS algorithm [Wold et al., 1984; Rosipal and Trejo, 2002] (see Appendix A.2) to the high-order tensors. To this end, our contributions are as follows:

1. we introduce a generalized nonlinear framework that effectively fuses the information from multiple tensorial data sources and integrates single and multiblock tensor regression scenarios into one general model using both common and discriminative features;
2. we successfully address the nonlinear dependencies between multiple response and predictor tensor blocks by combining kernel machines with joint Tucker decomposition, which leads to a further enhanced predictive power;
3. we develop an efficient algorithm for KMTPLS based on sequentially extracting common and discriminative latent vectors, which can easily scale to a number of blocks.

To our knowledge, this is the first work that applies multiblock tensor regression approach to the multiview or multimodal human motion estimation problem in computer vision.

8.2 Kernel-based Multiblock Tensor PLS Regression (KMTPLS)

As already mentioned, the main objective of KMTPLS is to predict a set of dependent tensor blocks from a set of independent tensor blocks through the extraction of a small number of common and individual latent components followed by a regression step using them.

Without the loss of generality, we consider a (M_1+1)th-order independent tensor block $\mathcal{X}_1 \in \mathbb{R}^{N\times I_1\times\cdots\times I_{M_1}}$, a (M_2+1)th-order independent tensor block $\mathcal{X}_2 \in \mathbb{R}^{N\times J_1\times\cdots\times J_{M_2}}$ and a $(L+$

1)th-order dependent tensor block $\mathcal{Y} \in \mathbb{R}^{N \times K_1 \times \cdots \times K_L}$, which can be obtained by concatenating N pairs of observations $\{(\mathcal{X}_1^{(n)}, \mathcal{X}_2^{(n)}, \mathcal{Y}^{(n)})\}_{n=1}^{N}$ that couple in the first mode with the same sample size. Similar to KTPLS [Zhao et al., 2013b], the tensorial input and output data points $\mathcal{X}_1^{(n)}$, $\mathcal{X}_2^{(n)}$ and $\mathcal{Y}^{(n)}$ are mapped into the high-dimensional feature space $\mathcal{H}$ using a nonlinear transformation ϕ as follows:

$$\phi : \mathcal{X}^{(n)} \rightarrow \phi(\mathcal{X}^{(n)}) \in \mathbb{R}^{H_1 \times \cdots \times H_M}. \tag{8.1}$$

We thus have $\phi(\mathcal{X}_1)$, $\phi(\mathcal{X}_2)$ and $\phi(\mathcal{Y})$ for the corresponding blocks, which for simplicity can be denoted as Φ_1, Φ_2 and Ψ, respectively.

Unlike KTPLS, we now perform the Tucker decompositions of Φ_1, Φ_2 and Ψ jointly in $\mathcal{H}$ by taking both common and individual features into account

$$\begin{aligned}
\Phi_1 &= \mathcal{G}_{\mathcal{X}_1} \times_1 [\mathbf{T}_{com} | \mathbf{T}_{dis_1}] \times_2 \mathbf{P}_1^{(1)} \times \cdots \times_{M_1+1} \mathbf{P}_1^{(M_1)} + \epsilon_{\mathcal{X}_1}, \\
\Phi_2 &= \mathcal{G}_{\mathcal{X}_2} \times_1 [\mathbf{T}_{com} | \mathbf{T}_{dis_2}] \times_2 \mathbf{P}_2^{(1)} \times \cdots \times_{M_2+1} \mathbf{P}_2^{(M_2)} + \epsilon_{\mathcal{X}_2}, \\
\Psi &= \mathcal{G}_{\mathcal{Y}} \times_1 [\mathbf{U}_{com} | \mathbf{U}_{dis_1} | \mathbf{U}_{dis_2}] \times_2 \mathbf{Q}^{(1)} \times \cdots \times_{L+1} \mathbf{Q}^{(L)} + \epsilon_{\mathcal{Y}}, \\
\mathbf{U}_{dis_1} &= \mathbf{T}_{dis_1} \mathbf{D}_{dis_1} + \mathbf{E}_{dis_1}, \\
\mathbf{U}_{dis_2} &= \mathbf{T}_{dis_2} \mathbf{D}_{dis_2} + \mathbf{E}_{dis_2}, \\
\mathbf{U}_{com} &= \mathbf{T}_{com} \mathbf{D}_{com} + \mathbf{E}_{com},
\end{aligned} \tag{8.2}$$

where $\{\mathcal{G}_{\mathcal{X}_1}, \mathcal{G}_{\mathcal{X}_2}, \mathcal{G}_{\mathcal{Y}}\}$ stand for the core tensors and $\{\mathbf{P}_1^{(m_1)}, \mathbf{P}_2^{(m_2)}, \mathbf{Q}^{(l)}\}$ denote the corresponding loading factor matrices. $\{\mathbf{T}_{com}, \mathbf{U}_{com}\}$ are defined as the common latent factor matrices, while $\{\mathbf{T}_{dis_1}, \mathbf{U}_{dis_1}\}$ and $\{\mathbf{T}_{dis_2}, \mathbf{U}_{dis_2}\}$ correspond to the discriminative factor matrices. Note that each pair of latent factor matrices is connected by an inner relation, namely the diagonal matrices $\mathbf{D}_{com}$, $\mathbf{D}_{dis_1}$ and $\mathbf{D}_{dis_2}$, assuming that $\mathbf{U}$ is linearly approximated by $\mathbf{T}$. Finally, $\{\epsilon_{\mathcal{X}_1}, \epsilon_{\mathcal{X}_2}, \epsilon_{\mathcal{Y}}\}$ and $\{\mathbf{E}_{dis_1}, \mathbf{E}_{dis_2}, \mathbf{E}_{com}\}$ represent the residuals. As we can see, Φ_1 and Φ_2 are simultaneously correlated with Ψ by having the largest covariance between latent factors $\mathbf{T}_{com}$ and $\mathbf{U}_{com}$. Meanwhile, Φ_1 singly associates with Ψ by maximizing covariance between latent factors $\mathbf{T}_{dis_1}$ and $\mathbf{U}_{dis_1}$. Likewise, the individual connection between Φ_2 and Ψ is characterized via the maximum covariance between $\mathbf{T}_{dis_2}$ and $\mathbf{U}_{dis_2}$.

The framework of KMTPLS is shown in Figure 8.1. The latent factor matrix $[\mathbf{U}_{com} | \mathbf{U}_{dis_1} | \mathbf{U}_{dis_2}]$ of output block Ψ consists of individual latent factors $\mathbf{U}_{dis_1}$ and $\mathbf{U}_{dis_2}$ from each respective input block and common latent factor $\mathbf{U}_{com}$ from both input blocks. In essence, the common part $\mathbf{T}_{com}$ captures the variation that is present in all predictor blocks and hence describes what all blocks have in common. The discriminative parts $\mathbf{T}_{dis_1}$ and $\mathbf{T}_{dis_2}$ explain the variation only in their own individual tensor block. Rather than explicitly estimating all the high-dimensional core tensors and loading factors, we only need to compute all the latent factors described in (8.2).

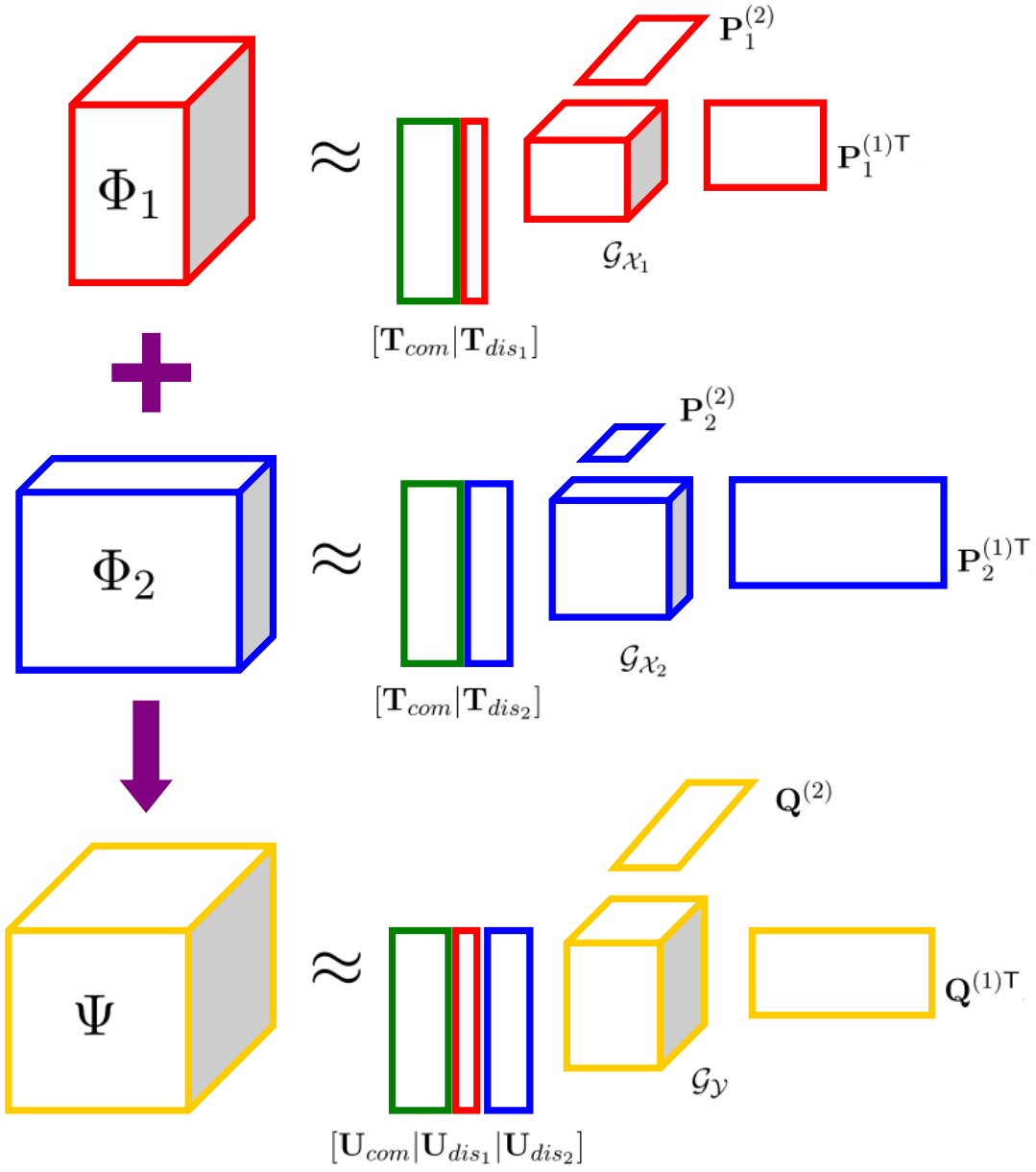


FIGURE 8.1 – Illustration of the framework of kernel-based multiblock tensor partial least squares (KMTPLS) regression.

In [Zhao et al., 2013b], the optimization objective related to the regression problem of single input tensor and single output tensor is given by

$$\begin{aligned} \max_{\{\mathbf{w}_r,\mathbf{v}_r\}} [cov(\mathbf{t}_r, \mathbf{u}_r)]^2 \quad \text{for} \quad r = 1, ..., R, \\ \text{s.t. } \mathbf{t}_r = \Phi_{(1)}\mathbf{w}_r, \ \mathbf{u}_r = \Psi_{(1)}\mathbf{v}_r, \end{aligned} \tag{8.3}$$

where $\Phi_{(1)}$ and $\Psi_{(1)}$ are the matricizations of single input Φ and output Ψ in the first mode, respectively. Here $\mathbf{w}_r$ and $\mathbf{v}_r$ serve as weight vectors. The goal is to maximize the covariance of latent vector $\mathbf{t}_r$ of input Φ and latent vector $\mathbf{u}_r$ of output Ψ for $r = 1, ..., R$.

In our 2-block input setting, for the common latent factor $\mathbf{T}_{com} = [t_1, ..., t_R]$ the optimization objective related to the regression problem extended from [Zhao et al., 2013b] turns out to be

$$\begin{aligned} \max_{\{\mathbf{w}_{1r},\mathbf{w}_{2r},\mathbf{v}_r\}} [cov_1(\mathbf{t}_r, \mathbf{u}_r)cov_2(\mathbf{t}_r, \mathbf{u}_r)]^2 \quad \text{for} \quad r = 1, ..., R, \\ \text{s.t. } \mathbf{t}_r = \Phi_{1(1)}\mathbf{w}_{1r} = \Phi_{2(1)}\mathbf{w}_{2r}, \ \mathbf{u}_r = \Psi_{(1)}\mathbf{v}_r, \end{aligned} \tag{8.4}$$

where we aim to optimize two covariance cov_1 and cov_2 at the same time, yielding the maximum product of pair-wise covariance of latent vectors. Here cov_1 involves input block Φ_1 and output block Ψ, while cov_2 relates to input block Φ_2 and output block Ψ. In the case of discriminative part $\mathbf{T}_{dis_1}$ ($\mathbf{T}_{dis_2}$), the objective is similar except that only the cov_1 (cov_2) should be taken into consideration.

The strategy for solving the above optimization problem (8.4) consists in sequentially extracting by deflation R pairs of the latent vectors $\{\mathbf{t}_r, \mathbf{u}_r\}$ to incorporate the information of Φ_1 and Φ_2 into the model simultaneously.

The KMTPLS procedure extending from the NIPALS-PLS algorithm [Wold et al., 1984; Rosipal and Trejo, 2002] is summarized in Algorithm 8, which consists of two major stages. In Stage 1 (lines 4-15), during each inner iteration the latent vector $\mathbf{t}_r$ is first updated by the information of Φ_1 at line 7 and then immediately followed by an update of the information of Φ_2 at line 9. Having extracted a new pair of $\{\mathbf{t}_r, \mathbf{u}_r\}$, the deflations with respect to $\Phi_{1(1)}\Phi_{1(1)}^\mathsf{T}$, $\Phi_{2(1)}\Phi_{2(1)}^\mathsf{T}$ and $\Psi_{(1)}\Psi_{(1)}^\mathsf{T}$ are executed from lines 12 to 14, removing the calculated variation from the corresponding blocks. The repeated extraction procedure stops when the desired R number of vector pairs are obtained (line 15). Stage 1 means the common contribution from Φ_1 and Φ_2 is collaboratively responsible for Ψ. Substituting these inner steps into each other, Stage 1 is in fact equivalent to solving the following eigenvalue problems

$$\begin{aligned} \Phi_{1(1)}\Phi_{1(1)}^\mathsf{T}\Psi_{(1)}\Psi_{(1)}^\mathsf{T}\Phi_{2(1)}\Phi_{2(1)}^\mathsf{T}\Psi_{(1)}\Psi_{(1)}^\mathsf{T}\mathbf{t}_r = \lambda\mathbf{t}_r, \\ \mathbf{u}_r = \Psi_{(1)}\Psi_{(1)}^\mathsf{T}\mathbf{t}_r. \end{aligned} \tag{8.5}$$

It is worth noting that $\Phi_{1(1)}\Phi_{1(1)}^\mathsf{T}$ and $\Phi_{2(1)}\Phi_{2(1)}^\mathsf{T}$, containing only the inner products between vectorized tensorial data points, can be substituted by kernel Gram matrices $\mathbf{K}_{\mathcal{X}_1}$ and $\mathbf{K}_{\mathcal{X}_2}$

respectively. Likewise, the $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}$ is also replaced with $\mathbf{K}_{\mathcal{Y}}$. Hence, the previous deflation step in Algorithm 8 for $\Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}}$ (line 12) finally becomes

$$\mathbf{K}_{\mathcal{X}_1} \leftarrow (\mathbf{I} - \mathbf{t}_r\mathbf{t}_r^T)\mathbf{K}_{\mathcal{X}_1}(\mathbf{I} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}), \tag{8.6}$$

and the previous formulations (8.5) can be rewritten as

$$\begin{aligned} \mathbf{K}_{\mathcal{X}_1}\mathbf{K}_{\mathcal{Y}}\mathbf{K}_{\mathcal{X}_2}\mathbf{K}_{\mathcal{Y}}\mathbf{t}_r &= \lambda\mathbf{t}_r, \\ \mathbf{u}_r &= \mathbf{K}_{\mathcal{Y}}\mathbf{t}_r. \end{aligned} \tag{8.7}$$

Thereafter, we continue to extract the discriminative latent pairs $\{\mathbf{T}_{dis_1}, \mathbf{U}_{dis_1}\}$ and $\{\mathbf{T}_{dis_2}, \mathbf{U}_{dis_2}\}$ (lines 16-30), which are used to explain the variation of each individual block, from the deflated Φ_1, Φ_2 and Ψ obtained in the first stage.

In Stage 2 of KMTPLS in Algorithm 8, we follow the similar extracting pattern as Stage 1 except that $\{\Phi_1, \Psi\}$ and $\{\Phi_2, \Psi\}$ are considered separately, which implies one separate effect on the final response Ψ is from Φ_1, and the other separate effect on the final response Ψ is from Φ_2.

As shown in Algorithm 8, the computational cost of extraction each pair of latent vectors (e.g., common latent vectors $\{\mathbf{t}_r, \mathbf{u}_r\}$ with $r = 1, ..., R_{com}$, or discriminative latent vectors $\{\mathbf{t}_{i\,r_i}, \mathbf{u}_{i\,r_i}\}$ with $r_i = 1, ..., R_{dis\,i}$ and $i = 1, , , T$) is $\mathcal{O}(N^2)$. Hence, the overall cost is proportional to the total number of desired latent components in T blocks, i.e., $R_{com} + R_{dis1} + \cdots + R_{disT}$, which means the algorithm can be scalable to relatively large number of T blocks. We should mention that the cost of establishing the kernel matrix, depending on the specially designed tensor kernel function, should also be taken into account.

For the single block case, the final prediction $\mathcal{Y}_{new}$ from single input test point $\mathcal{X}_{new}$ is given by Zhao et al. [2013b] as

$$\mathbf{y}_{new}^{\mathsf{T}} = \mathbf{k}_{\mathcal{X}_{new}}^{\mathsf{T}}\mathbf{U}(\mathbf{T}^{\mathsf{T}}\mathbf{K}_{\mathcal{X}}\mathbf{U})^{-1}\mathbf{T}^{\mathsf{T}}\mathbf{Y}_{(1)}, \tag{8.8}$$

where $(\mathbf{k}_{\mathcal{X}_{new}})_n = k(\mathcal{X}_{new}, \mathcal{X}^{(n)})$.

As for our 2-block input setting, with all the latent factors $\mathbf{T}_{com}$, $\mathbf{T}_{dis_1}$, $\mathbf{T}_{dis_2}$ and $\mathbf{U}_{com}$, $\mathbf{U}_{dis_1}$, $\mathbf{U}_{dis_2}$ in hand, the new response $\mathcal{Y}_{new}$ can be jointly predicted from the test point $\mathcal{X}_{1new}$ of the first input block and $\mathcal{X}_{2new}$ of the second input block by extending (8.8) to be

$$\mathbf{y}_{new}^{\mathsf{T}} = \alpha\mathbf{k}_{\mathcal{X}_{1new}}^{\mathsf{T}}\mathbf{U}_1(\mathbf{T}_1^{\mathsf{T}}\mathbf{K}_{\mathcal{X}_1}\mathbf{U}_1)^{-1}\mathbf{T}_{dis_1}^{\mathsf{T}}\mathbf{Y}_{(1)} + (1-\alpha)\mathbf{k}_{\mathcal{X}_{2new}}^{\mathsf{T}}\mathbf{U}_2(\mathbf{T}_2^{\mathsf{T}}\mathbf{K}_{\mathcal{X}_2}\mathbf{U}_2)^{-1}\mathbf{T}_{dis_2}^{\mathsf{T}}\mathbf{Y}_{(1)}, \tag{8.9}$$

where $(\mathbf{k}_{\mathcal{X}_{1new}})_n = k(\mathcal{X}_{1new}, \mathcal{X}_1^{(n)})$ and $(\mathbf{k}_{\mathcal{X}_{2new}})_n = k(\mathcal{X}_{2new}, \mathcal{X}_2^{(n)})$. $\mathbf{T}_1 = [\mathbf{T}_{com}|\mathbf{T}_{dis_1}]$, $\mathbf{T}_2 = [\mathbf{T}_{com}|\mathbf{T}_{dis_2}]$, $\mathbf{U}_1 = [\mathbf{U}_{com}|\mathbf{U}_{dis_1}]$ and $\mathbf{U}_2 = [\mathbf{U}_{com}|\mathbf{U}_{dis_2}]$. The hyperparameter α varying from 0 to 1 indicates the relative importance of each block, and thus can be heuristically determined according to their individual predictive ability. Notice that here $\mathbf{y}_{new}$ is in a vector form and should be reformulated into a tensor form $\mathcal{Y}_{new}$ by refolding.

Algorithm 8 Kernel-based Multiblock Tensor Partial Least Squares (KMTPLS)

Input: observations of Φ_1, Φ_2 and Ψ; number of desired latent vectors R, R_1 and R_2
Output: common matrices $\mathbf{T}_{com}$, $\mathbf{U}_{com}$ and discriminative matrices $\mathbf{T}_{dis_1}$, $\mathbf{U}_{dis_1}$, $\mathbf{T}_{dis_2}$, $\mathbf{U}_{dis_2}$
Initialize: randomly initialize $\mathbf{u}_r$, $\mathbf{u}_{1r_1}$ and $\mathbf{u}_{2r_2}$
/* **Stage 1**: **extract common latent factor $\mathbf{T}_{com}$, $\mathbf{U}_{com}$** */
repeat
  repeat
    $\mathbf{t}_r = \Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}}\mathbf{u}_r, \quad \mathbf{t}_r \leftarrow \mathbf{t}_r/\|\mathbf{t}_r\|$
    $\mathbf{u}_r = \Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_r, \quad \mathbf{u}_r \leftarrow \mathbf{u}_r/\|\mathbf{u}_r\|$
    $\mathbf{t}_r = \Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}}\mathbf{u}_r, \quad \mathbf{t}_r \leftarrow \mathbf{t}_r/\|\mathbf{t}_r\|$
    $\mathbf{u}_r = \Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_r, \quad \mathbf{u}_r \leftarrow \mathbf{u}_r/\|\mathbf{u}_r\|$
  until convergence
  deflate $\Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}}$ matrix:
  $\Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}} \leftarrow (\Phi_{1(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Phi_{1(1)})(\Phi_{1(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Phi_{1(1)})^{\mathsf{T}}$
  deflate $\Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}}$ matrix:
  $\Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}} \leftarrow (\Phi_{2(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Phi_{2(1)})(\Phi_{2(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Phi_{2(1)})^{\mathsf{T}}$
  deflate $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}$ matrix:
  $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}} \leftarrow (\Psi_{(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Psi_{(1)})(\Psi_{(1)} - \mathbf{t}_r\mathbf{t}_r^{\mathsf{T}}\Psi_{(1)})^{\mathsf{T}}$
until R pairs of latent vectors of $\{\mathbf{T}_{com}, \mathbf{U}_{com}\}$ or the rank condition is met
/* **Stage 2**: **extract discriminative latent factors $\mathbf{T}_{dis_1}$, $\mathbf{U}_{dis_1}$, $\mathbf{T}_{dis_2}$, $\mathbf{U}_{dis_2}$** */
repeat
  repeat
    $\mathbf{t}_{1r_1} = \Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}}\mathbf{u}_{1r_1}, \quad \mathbf{t}_{1r_1} \leftarrow \mathbf{t}_{1r_1}/\|\mathbf{t}_{1r_1}\|$
    $\mathbf{u}_{1r_1} = \Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_{1r_1}, \quad \mathbf{u}_{1r_1} \leftarrow \mathbf{u}_{1r_1}/\|\mathbf{u}_{1r_1}\|$
  until convergence
  deflate $\Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}}$ matrix:
  $\Phi_{1(1)}\Phi_{1(1)}^{\mathsf{T}} \leftarrow (\Phi_{1(1)} - \mathbf{t}_{1r_1}\mathbf{t}_{1r_1}^{\mathsf{T}}\Phi_{1(1)})(\Phi_{1(1)} - \mathbf{t}_{1r_1}\mathbf{t}_{1r_1}^{\mathsf{T}}\Phi_{1(1)})^{\mathsf{T}}$
  deflate $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}$ matrix:
  $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}} \leftarrow (\Psi_{(1)} - \mathbf{t}_{1r_1}\mathbf{t}_{1r_1}^{\mathsf{T}}\Psi_{(1)})(\Psi_{(1)} - \mathbf{t}_{1r_1}\mathbf{t}_{1r_1}^{\mathsf{T}}\Psi_{(1)})^{\mathsf{T}}$
  repeat
    $\mathbf{t}_{2r_2} = \Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}}\mathbf{u}_{2r_2}, \quad \mathbf{t}_{2r_2} \leftarrow \mathbf{t}_{2r_2}/\|\mathbf{t}_{2r_2}\|$
    $\mathbf{u}_{2r_2} = \Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}\mathbf{t}_{2r_2}, \quad \mathbf{u}_{2r_2} \leftarrow \mathbf{u}_{2r_2}/\|\mathbf{u}_{2r_2}\|$
  until convergence
  deflate $\Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}}$ matrix:
  $\Phi_{2(1)}\Phi_{2(1)}^{\mathsf{T}} \leftarrow (\Phi_{2(1)} - \mathbf{t}_{2r_2}\mathbf{t}_{2r_2}^{\mathsf{T}}\Phi_{2(1)})(\Phi_{2(1)} - \mathbf{t}_{2r_2}\mathbf{t}_{2r_2}^{\mathsf{T}}\Phi_{2(1)})^{\mathsf{T}}$
  deflate $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}}$ matrix:
  $\Psi_{(1)}\Psi_{(1)}^{\mathsf{T}} \leftarrow (\Psi_{(1)} - \mathbf{t}_{2r_2}\mathbf{t}_{2r_2}^{\mathsf{T}}\Psi_{(1)})(\Psi_{(1)} - \mathbf{t}_{2r_2}\mathbf{t}_{2r_2}^{\mathsf{T}}\Psi_{(1)})^{\mathsf{T}}$
until R_1 pairs of latent vectors of $\{\mathbf{T}_{dis_1}, \mathbf{U}_{dis_1}\}$ or the rank condition is met, and R_2 pairs of latent vectors of $\{\mathbf{T}_{dis_2}, \mathbf{U}_{dis_2}\}$ or the rank condition is met

TABLE 8.1 – Performance comparison of KMTPLS and MMCR for the optimal Rank, Q and RMSEP on UMPM data.

Scenario	Methods	Camera	Rank	Q	RMSEP
Grab	KMTPLS	F	(10)	0.7141	241.4
		L	(6)	0.7240	230.0
		R	(10)	0.7789	185.5
		F-L	(6,4,1)	**0.7690**	**192.7**
		F-R	(8,1,2)	**0.7950**	**172.8**
		L-R	(8,0,2)	**0.7995**	**169.7**
	MMCR	F-L	(3,10,2)	0.6237	347.7
		F-R	(3,10,8)	0.6111	351.2
		L-R	(3,10,1)	0.6497	292.7
Triangle	KMTPLS	F	(9)	0.7599	202.5
		L	(9)	0.7462	208.2
		R	(10)	0.7227	232.3
		F-L	(8,2,2)	**0.7801**	**181.3**
		F-R	(1,8,0)	**0.7629**	**200.2**
		L-R	(7,1,3)	**0.7691**	**190.6**
	MMCR	F-L	(4,9,1)	0.7168	241.3
		F-R	(4,10,1)	0.7195	239.9
		L-R	(5,8,2)	0.7324	224.5
Table	KMTPLS	F	(10)	0.6340	298.0
		L	(9)	0.5995	319.3
		R	(10)	0.5906	325.9
		F-L	(1,9,8)	**0.6910**	**247.4**
		F-R	(1,9,0)	0.6414	291.2
		L-R	(9,0,1)	**0.6847**	**250.7**
	MMCR	F-L	(3,9,9)	0.6713	267.2
		F-R	(5,10,1)	**0.6626**	**277.2**
		L-R	(4,7,2)	0.6761	260.9

8.3 Experimental Results

In all experiments, the MMCR and KMTPLS were used for comparison including the settings of single and multiple input blocks for KMTPLS model. For simplicity, the polynomial kernel function of second degree was employed. The predictive performance was quantitatively evaluated by means of the root mean squares of prediction (RMSEP) [Kim et al., 2005] and the Q index [Luo et al., 2015].

8.3.1 Utrecht Multi-Person Motion Database (UMPM)

We first applied our KMTPLS algorithm to a real-life tensor regression task, i.e., estimation of articulated $3D$ human pose positions from multiple video streams, to systematically show the advantages of our algorithm. The dataset was taken from the Utrecht Multi-Person Motion (UMPM) benchmark [Aa et al., 2011], which provides synchronized motion capture data and video sequences from multiple viewpoints.

More specifically, the video sequences of 30-60 seconds were captured using 4 Basler calibrated color cameras with a resolution of 644×484 pixels at 50 fps. In addition, to obtain the $3D$ ground truth information, a Vicon MoCap system was used to record the $3D$ positions of 37 reflective markers attached to each subject at a frame rate of 100 fps for describing the movements of head, shoulders, elbows, wrists, knees and ankles etc. The activities of natural motions of humans in daily life such as walking, jogging, balancing were captured within a region of 6m×6m. For our test, we employed the intensity image sequences, with a downsized resolution of 32×24 pixels, as inputs of our KMTPLS algorithm. Hence, each video sequence can be naturally represented as a 3-order predictor tensor (i.e., *frames* $\times$ *width* $\times$ *height*). As for the ground truth, the MoCap data was down-sampled at 50 fps, and thus can be expressed as a 3rd-order tensor (i.e., *samples* $\times$ *3D positions* $\times$ *markers*). For each scenario, we split the video sequence into two different partitions, i.e., a training set from the first $1/3$ part and a test set from the remaining $2/3$ part, respectively. The cross-validation applied on training set was performed to select all the desired tuning parameters.

In our experiments, we chose 3 principal cameras that form an equilateral triangle including the front camera (F), the left camera (L) and the right camera (R), and we mainly focused on the 2-block situation while the multiple input blocks cases were studied in the following section. To be fair, the maximal possible number of latent vectors that can be extracted from each individual block was set as 10 for both KMTPLS and MMCR.

Table 8.1 summarizes the prediction performance from KMTPLS and MMCR. Clearly, our method significantly outperformed MMCR in almost all cases in terms of predictive accuracy,

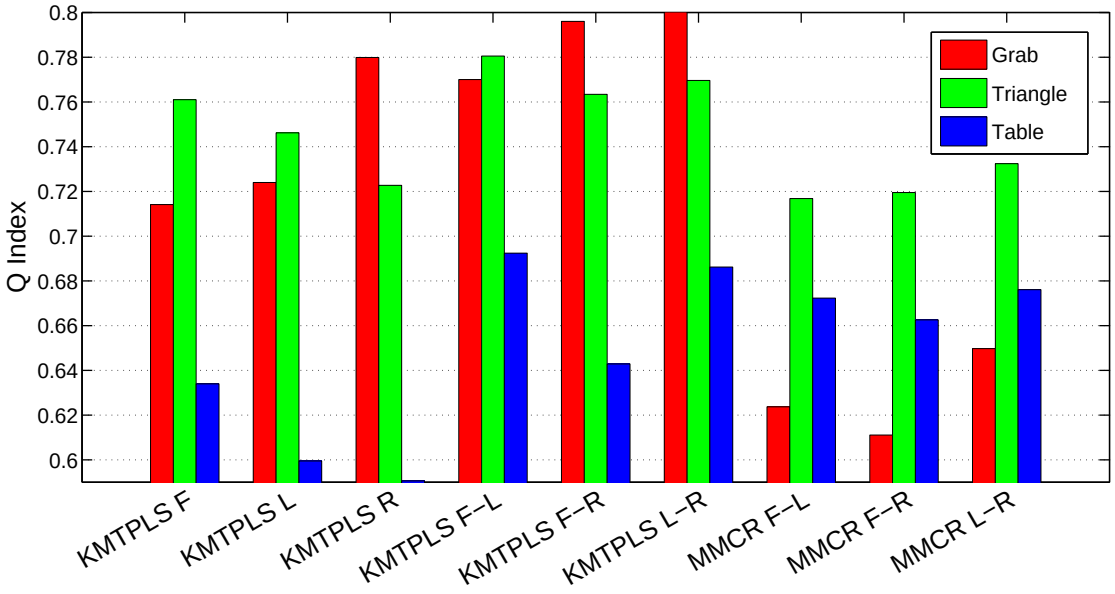


FIGURE 8.2 – Performance comparison of KMTPLS and MMCR for the Q versus the different combination of cameras on UMPM data.

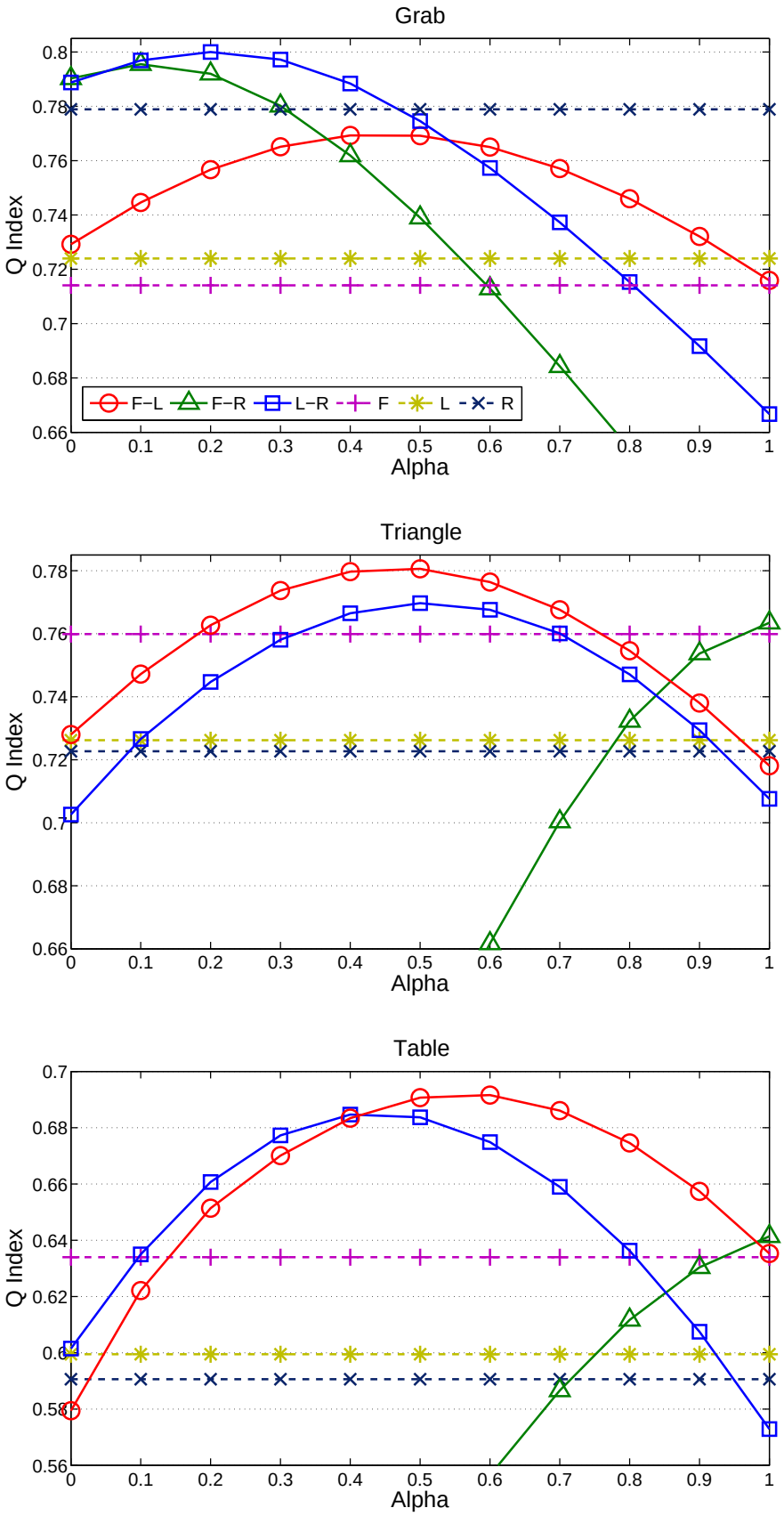


FIGURE 8.3 – Performance comparison of KMTPLS for the Q when using the optimal rank versus the relative importance α on UMPM data.

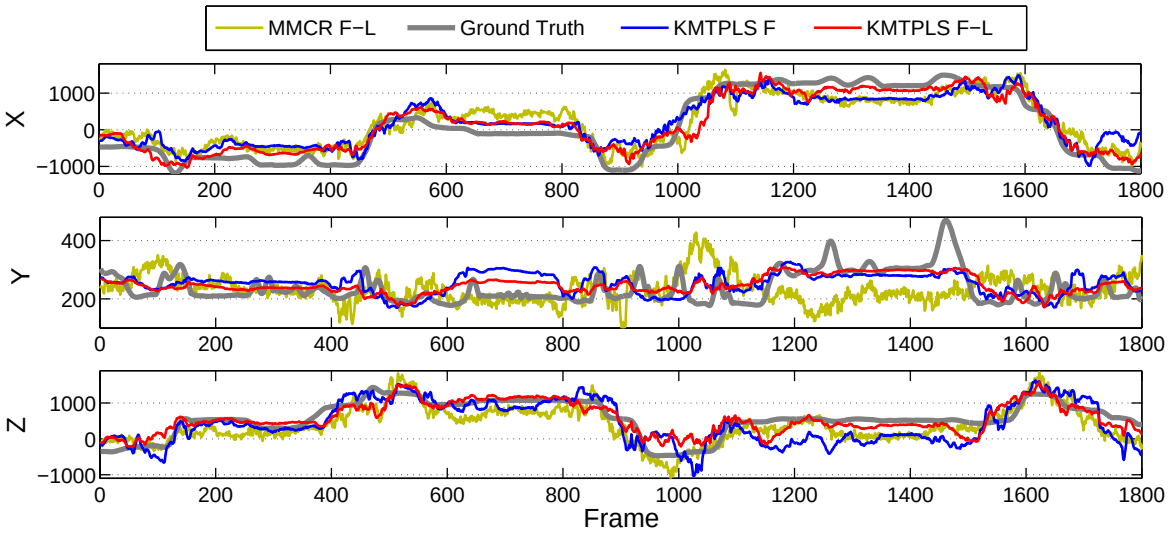


FIGURE 8.4 – Visualization of ground truth and the trajectories predicted by MMCR and KMTPLS in the "table" scenario.

especially in the "grab" scenario where the largest performance improvement were 30.09% with respect to Q when the front camera and the right camera were integrated. Meanwhile, as also shown in Figure 8.2, we can see that the considerably improved predictability were achieved by all 2-block situations in comparison to the corresponding single block cases. For example, in the "table" scene, the optimal Q obtained by the left and the right cameras were 0.5995 and 0.5906, while it was boosted to 0.6847, with the improvement of 14.21% and 15.93%, as we effectively fuse two cameras using KMTPLS.

To investigate the effects of α, which indicates the relative importance of each block as reflected in (8.9), on the predictive performance, Q is given in Figure 8.3 for varying α at the retained ranks. We may notice that the optimal α, which is the measurement of involvement in the final prediction, can somehow roughly reflect the relatively predictive power of each individual block. Specifically, the optimal α in the "grab" scene were 0.1 and 0.2 for the "F-R" and "L-R", indicating the model intended to attach more importance to the camera "R" with the best singleton predictive accuracy of 0.7789. For the visualization, Figure 8.4 further demonstrates the ground truth and the predicted $3D$ front angle trajectories for the test sequence using the front and left cameras. We can see that our KMTPLS "L-R" achieves much better accuracy than both KMTPLS "L" and MMCR "L-R" cases.

8.3.2 Berkeley Multimodal Human Action Database

In order to validate the superiority of KMTPLS in the context of several cameras from multiple views, a second experiment was carried out on the Berkeley Multimodal Human Action Database (MHAD) [Ofli et al., 2013] and it consists of temporally synchronized data simul-

taneously recorded by several different systems, including an optical motion capture system, four multi-view stereo camera clusters and two Microsoft Kinect cameras.

The MHAD comprises 11 actions performed by 12 subjects, with each subject performing 1 or 5 repetitions of each action per recording. We chose the 2 most challengeable actions with 5 repetitions that involve dynamics in both upper and lower extremities, namely "jumping in place" and "bending-hands up all the way down".

In this experiment, we were particularly interested in how the predictive performance is affected by the number of predictor blocks in the process of fusion. To this end, we selected 4 of C1 cameras from 4 clusters as well as 2 of C2 cameras from clusters L1 and L2, resulting in a total number of 6 cameras from all available 12 cameras. Note that each time we chose a subset of these 6 cameras as candidates to jointly make the prediction, we therefore have "6 *choose* T" (C_6^T) different combinations for the T-block situation. Similar to the previous experiment, both video sequence and MoCap data are represented as 3-order tensors. In the case of the ground truth, the $3D$ positions of 43 LED markers affixed to different parts of the body were captured within the space of 2m×2m. We took the first recording of each action as training set and the second recording as test set. The optimal number of common and discriminative latent vectors was selected by cross validation on the training set. The maximal number of latent vectors from each block for both models was fixed as 8. Finally, the total number of predictor tensor blocks T grew up to 4, and the importance parameter α for each block was simply fixed to be $1/T$.

The averaged prediction results as well as the learning time of the two methods are compared in Table 8.2. As was expected, KMTPLS exhibited much better performance than MMCR with respect to all the measurements. In particular, for the "bending" action, the averaged improvement by KMTPLS over MMCR when using 4 blocks were 7.53%, 8.28% and 4.28% in terms of Q, and 15.8%, 19.5% and 8.3% in terms of RMSEP for subjects "s6", "s10" and "s12", respectively. On the other hand, KMTPLS shows a consistently enhanced predictability as the number of blocks increases from 1 up to an optimal number, demonstrating the effectiveness of our fusion strategy. Specifically, comparing to 1-block KMTPLS, the improved accuracy of action "jumping" with respect to Q by 4-block KMTPLS accumulated to 2.40%, 1.71% and 2.92% for subjects "s6", "s10" and "s12".

Figure 8.5 and Figure 8.6 illustrate the best prediction results among the C_6^T combinations in each T-block case. We note that the predictive enhancement becomes smaller as more blocks were incorporated into the model, which was then followed by a slight decrease when the number of blocks exceeded an optimal threshold. For instance, in Figure 8.5, the optimal number of blocks for subject "s10" in "jumping" action is 3. This reflects the fact that adding too many blocks, on the other hand, may increase the chances of overfitting or bring noise into the model, leading to a degraded predictive ability.

TABLE 8.2 – Performance comparison of KMTPLS and MMCR for the averaged Q, RMSEP and learning time on MHAD data.

Action	Subject	Methods	Blocks	Q	RMSEP	Time (ms)
Jumping	S6	KMTPLS	1	0.6836 (0.0056)	209.3 (4.2)	**38** (19)
			2	**0.6955** (0.0043)	**200.7** (2.9)	70 (27)
			3	**0.6986** (0.0030)	**199.3** (1.9)	92 (58)
			4	**0.7000** (0.0025)	**198.5** (1.6)	139 (81)
		MMCR	2	0.6717 (0.0050)	218.3 (3.0)	1616 (999)
			3	0.6762 (0.0027)	215.4 (1.6)	2530 (1235)
			4	0.6790 (0.0019)	213.7 (1.1)	2954 (788)
	S10	KMTPLS	1	0.7141 (0.0057)	196.8 (4.3)	**45** (13)
			2	**0.7230** (0.0047)	**190.4** (3.5)	74 (43)
			3	**0.7253** (0.0038)	**188.3** (2.8)	134 (63)
			4	**0.7263** (0.0027)	**187.9** (2.0)	195 (85)
		MMCR	2	0.6930 (0.0047)	211.3 (3.4)	2248 (2175)
			3	0.6967 (0.0021)	208.7 (1.5)	2771 (2551)
			4	0.6989 (0.0010)	207.2 (0.8)	2985 (2408)
	S12	KMTPLS	1	0.7059 (0.0065)	198.9 (5.3)	**16** (6)
			2	**0.7199** (0.0032)	**189.2** (2.3)	31 (12)
			3	**0.7240** (0.0029)	**186.3** (2.2)	41 (13)
			4	**0.7265** (0.0016)	**184.3** (1.2)	69 (23)
		MMCR	2	0.6909 (0.0051)	208.1 (3.4)	1689 (983)
			3	0.6943 (0.0041)	205.4 (2.5)	2568 (1039)
			4	0.6959 (0.0036)	204.2 (2.1)	4280 (2136)
Bending	S6	KMTPLS	1	0.6991 (0.0059)	188.7 (4.4)	**71** (12)
			2	**0.7076** (0.0030)	**182.8** (2.3)	122 (15)
			3	**0.7112** (0.0021)	**179.9** (1.6)	173 (13)
			4	**0.7129** (0.0015)	**179.0** (1.2)	324 (44)
		MMCR	2	0.6524 (0.0179)	217.3 (11.1)	4510 (2201)
			3	0.6602 (0.0045)	212.8 (3.1)	8154 (3077)
			4	0.6630 (0.0029)	210.8 (1.8)	13116 (3974)
	S10	KMTPLS	1	0.7329 (0.0055)	168.9 (3.9)	**49** (22)
			2	**0.7449** (0.0029)	**160.2** (2.2)	110 (26)
			3	**0.7493** (0.0029)	**157.3** (1.3)	149 (21)
			4	**0.7516** (0.0015)	**155.6** (1.1)	199 (27)
		MMCR	2	0.7010 (0.0100)	188.9 (6.3)	8118 (3676)
			3	0.6945 (0.0164)	193.0 (10.5)	11575 (4390)
			4	0.6941 (0.0133)	193.4 (8.7)	15540 (4837)
	S12	KMTPLS	1	0.6755 (0.0060)	195.3 (4.3)	**69** (16)
			2	**0.6841** (0.0020)	**189.5** (1.4)	138 (19)
			3	**0.6859** (0.0015)	**188.8** (1.0)	226 (38)
			4	**0.6867** (0.0015)	**188.4** (1.0)	314 (45)
		MMCR	2	0.6541 (0.0032)	208.1 (1.7)	3256 (1318)
			3	0.6564 (0.0033)	206.8 (1.3)	7289 (2180)
			4	0.6585 (0.0020)	205.5 (1.0)	10649 (2925)

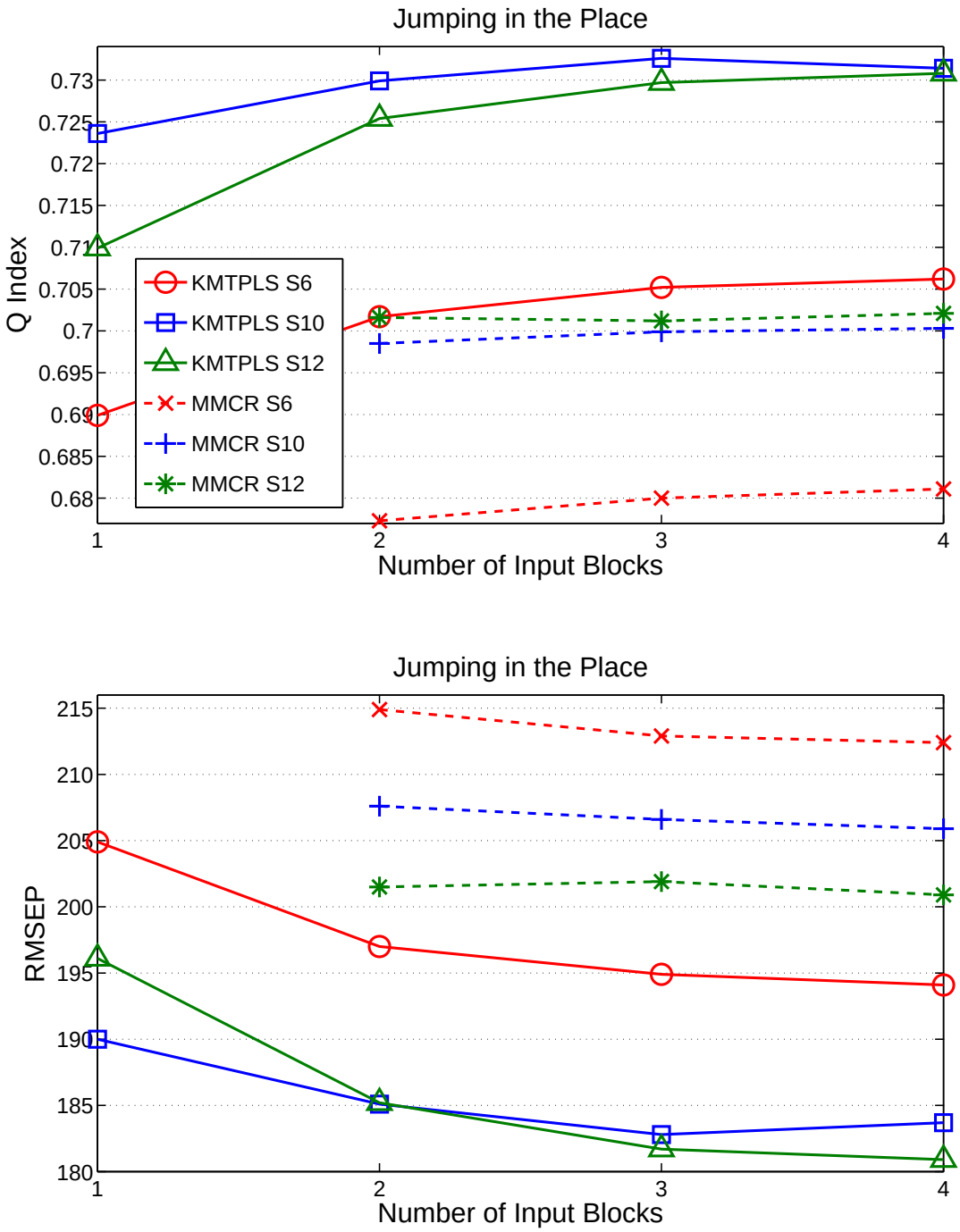


FIGURE 8.5 – Performance comparison of KMTPLS and MMCR for the best Q, RMSEP versus the number of input blocks on jumping action on MHAD data.

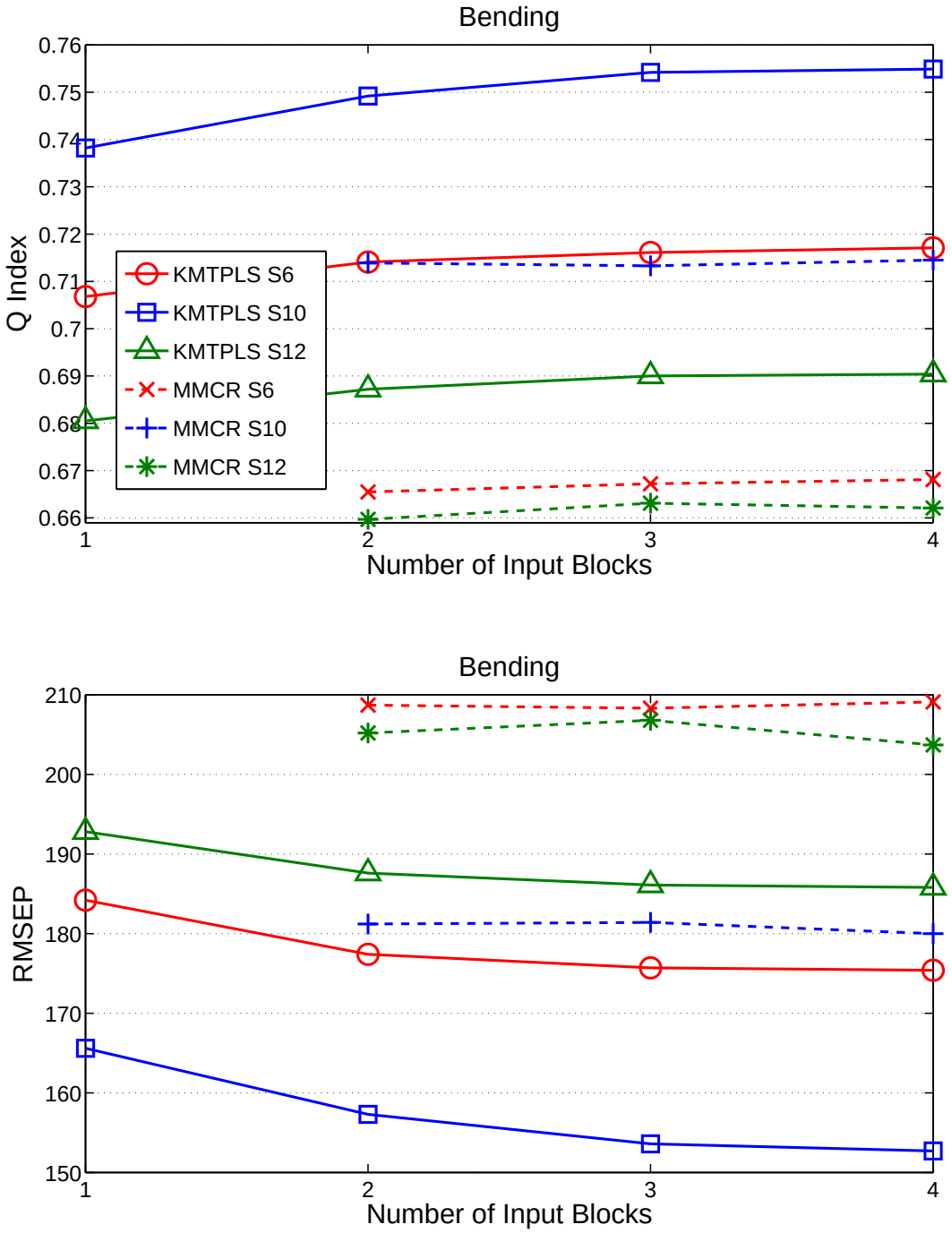


FIGURE 8.6 – Performance comparison of KMTPLS and MMCR for the best Q, RMSEP versus the number of input blocks on bending action on MHAD data.

8.4 Discussion

Some remarks about our KMTPLS approach can be noticed. This approach can actually be regarded as a generalized kernel-based nonlinear tensor regression framework that includes the KTPLS [Zhao et al., 2013b] as a special case by setting the number of latent components from other tensor blocks to be zero. Additionally, KMTPLS enables us to straightforwardly extend both model and algorithm to more general cases in which more than two predictor tensor blocks and one response tensor block are involved. Furthermore, based on common and discriminative features, KMTPLS is superior to MMCR with respect to the interpretability. Inheriting the advantage from KTPLS, KMTPLS is capable of flexibly adapting different types of specially defined tensor kernel functions, namely

$$\begin{aligned}(\mathbf{K}_{\mathcal{X}_1})_{nn'} &= k_{x_1}(\mathcal{X}_1^{(n)}, \mathcal{X}_1^{(n')}) \\ (\mathbf{K}_{\mathcal{X}_2})_{nn'} &= k_{x_2}(\mathcal{X}_2^{(n)}, \mathcal{X}_2^{(n')}) \\ (\mathbf{K}_{\mathcal{Y}})_{nn'} &= k_{y}(\mathcal{Y}^{(n)}, \mathcal{Y}^{(n')}),\end{aligned} \tag{8.10}$$

to different individual tensor blocks, leading to a better performance than that of combining all variables in one large $\mathcal{X}$-block. Note that our algorithm is extended from the NIPALS-PLS algorithm [Wold et al., 1984; Rosipal and Trejo, 2002], which has proved to be very robust for solving the eigenvalue related problems, providing a robust procedure for iteratively estimating latent components in our setting. Therefore, we can reliably extract a desired number of pairs of latent components up to the point when the rank of the corresponding unfolded tensor block is reached.

8.5 Conclusion

We have proposed a generalized nonlinear tensor regression framework KMTPLS that effectively fuses the information from multiple tensorial data sources and unifies the single and multiblock tensor regression into one model using both common and discriminative features. Comparing to multilinear model MMCR [Smilde et al., 2000], our approach can successfully deal with the nonlinear dependencies between multiple responses and predictor tensor blocks by combining kernel machines with joint Tucker decomposition, resulting in significantly enhanced predictive power.

The experimental results on both UMPM and MHAD databases have demonstrated the advantages of our method when applied to the real-world tensor regression task in computer vision. For future work, in connection with KMTPLS approach, we would like to mention that performance could be further improved at the cost of running time for designing and adapting more sophisticated tensorial kernel functions by exploiting more structural information of tensorial data than simply using polynomial or Gaussian kernel functions.

In the next chapter, we conclude the thesis and discuss some possible future work.

Chapitre 9

Conclusion

In this thesis, we have introduced several novel tensor-variate regression models and applied them to numerous real-world applications. Our proposed approaches provide promising solutions to different issues and problems confronted by the existing tensor regression models from different perspectives. These solutions have been validated on a number of challenging tensor-variate regression tasks with high accuracy and efficiency. In this last chapter, we sum up our contributions of the thesis as well as some suggestions for the future work.

9.1 Summary of the Contributions

Hierarchical Tucker Tensor Regression

In [Hou and Chaib-draa, 2015] and in Chapter 4, we first focused our attention on the multilinear regression model with tensorial input and scalar output, and we proposed a low-rank generalized linear tensor regression model by approximating the tensor regression coefficient with hierarchical Tucker representation.

Compared to the existing models [Zhou et al., 2013; Li et al., 2013], our proposed model preserves all the advantages in terms of simplicity and flexibility. The resulting model is highly compact as only $\mathcal{O}(DR^3 + DIR)$ parameters are needed for order D tensors of mode size I and rank R, which is free from the exponential growth in D. With this compression rate, our model can be found quite beneficial when applied to regression tasks with potentially very high order tensor input. (e.g., fifth-order face images with *pixels* $\times$ *illuminations* $\times$ *expressions* $\times$ *viewpoints* $\times$ *identities* [Vasilescu and Terzopoulos, 2002]). In addition, our model is particularly useful for application such as medical imaging data analysis (e.g., fMRI images) where only very limited number of samples are available, resulting in a better generalization performance. Moreover, as shown in Proposition 3 and Appendix B.1, our model is guaranteed to have good convergence properties as those of models in [Zhou et al., 2013; Li et al., 2013].

Tensor Online Local Gaussian Process Regression

In [Hou et al., 2015] and in Chapter 5, we then worked on the nonlinear nonparametric regression modeling with tensorial input and scalar output in the context of online setting. We extended tensor Gaussian process (tensor GP) [Zhao et al., 2014] to tensor online local Gaussian process (tensor OLGP) by constructing a number of small-sized GP experts in an sequential way, such that large-scale or even infinite tensor streams can be efficiently processed with a low constant computational load. Additionally, two effective searching strategies for finding local GP experts were presented to make accurate predictions.

In contrast to tensor GP, the cost of tensor OLGP for learning is reduced from $\mathcal{O}(N^2 I^{D+1}+N^3)$ to $\mathcal{O}(NRI^{D+1}+NSI^{D+1}+S^3)$, where N is the number of training samples, R is the number of local experts and S denotes the maximum number of data points contained in each local expert. While the computational complexity of prediction for tensor OLGP requires only $\mathcal{O}(RI^{D+1}+M(SI^{D+1}+S^2))$ as compared to $\mathcal{O}(NI^{D+1}+N^2)$ in tensor GP.

For the application, the effectiveness and scalability were validated on the regression task with large ECoG signal streams as the input. To our knowledge, we provide the first nonlinear tensor-variate regression algorithm that can be employed to the online setting.

Incremental Higher-order Partial Least Squares Regression

In [Hou and Chaib-draa, 2016] and in Chapter 6, we considered the multilinear tensor regression problems with both tensorial input and tensorial output in the context of time-critical dynamic environment. More specifically, we introduced incremental higher-order partial least squares (IHOPLS) algorithm to adapt the cutting-edge HOPLS [Zhao et al., 2013a] model to the setting of infinite time-dependent tensor streams.

Our IHOPLS is able to recursively update the projection matrices and core tensors over time by incrementally clustering the projected latent variables in latent space and summarizing the previous data in terms of a compact internal representation. As a result, in contrast to the batch HOPLS, IHOPLS greatly reduces the "sample" complexity to a low-cost constant level while maintaining high prediction accuracy. Furthermore, there is no need for IHOPLS to store the entire data as required by HOPLS. For the applications, we successfully applied IHOPLS to the reconstructions of $3D$ motion trajectories from video streams and also from ECoG streaming signals.

Recursive Higher-order Partial Least Squares Regression

Thereafter, we continued to work on multilinear tensor regression problems with both tensorial input and tensorial output for the setting of the large or infinite tensor sequences. To this end, we introduced in Chapter 7 a fast low-rank sequential tensor regression framework, called recursive higher-order partial least squares (RHOPLS) [Hou and Chaib-draa, 2017], to further

address the tensor-variate regression tasks with high "sample" complexity or "dimensionality" complexity. Especially, our proposed framework is designed to deal with the great challenges posed by the limited storage space and fast processing time allowed by dynamic environments when facing with very large-scale high-speed general tensor sequences.

Concretely, RHOPLS efficiently updates the regression coefficients at a small-scale factor (feature) level instead of the large raw data (observation) level by integrating a low-rank modification strategy of the Tucker [O'Hara, 2010] into PLS. Thus, our consecutive calculation scheme requires only a small set of factors to be stored. Compared to the existing approaches, our RHOPLS does not suffer from neither inferior predictability nor poor convergence rate of NPLS-based (CP-based) models. Our method is also free from the computational issues related to the HOPLS-based models when the data order, "sample" complexity or "dimensionality" complexity is high. Finally, our RHOPLS exhibits highly competitive accuracy with the best batch methods but is much faster than other sequential methods. For applications, we validated it on the challenging tasks of estimation of human pose from videos, showing the great potentiality for fast real-time predictions of human pose positions.

Kernel-based Multiblock Tensor Partial Least Squares Regression

At last, in [Hou et al., 2016] and in Chapter 8, we were interested in a more general tensor regression framework with the objective of predicting a set of dependent tensor blocks from a set of independent tensor blocks with a boosted predictive power. To this end, we proposed a kernel-based multiblock tensor partial least squares (KMTPLS) regression framework to achieve this goal through the extraction of a small number of common and discriminative latent components.

In this way, our framework effectively integrates the information from multiple tensorial data sources with different modalities, and unifies the single and multiblock tensor regression scenarios into one general model. Furthermore, comparing to multilinear model, KMTPLS successfully addresses the nonlinear dependencies between multiple inputs and outputs by combining kernel machines with joint Tucker decomposition, which leads to further enhanced predictability. Moreover, our algorithm for KMTPLS, based on sequentially extracting common and discriminative latent vectors, can easily scale to a number of input and output blocks.

To our knowledge, this is the first work that applies multiblock tensor regression approach to the multiview or multimodal human motion estimation problem in computer vision.

9.2 Possible Future Work

As we have already seen, most of existing tensor regression models are multilinear models that are based on the low-rank factorizations or decompositions, i.e., CP family and Tucker family, whose factors extracted from each mode are linear combinations of variables in that mode.

On one hand, multilinear tensor regression models have been extensively studied and seem to reach a performance bottleneck. On the other hand, it is well known that multilinear models have drawbacks in capturing the nonlinear structures in tensorial data. Therefore, nonlinear regression modeling approaches should be further explored in the context of tensors in order to overcome the limitations of multilinear models.

Recently, deep learning has emerged as a new promising area of machine learning research, which is a class of machine learning algorithms that are based on the learning of multiple levels of features or representations of the data with increasing level of abstraction [Hinton et al., 2006; Larochelle et al., 2009; Bengio et al., 2013; Havaei et al., 2016]. Higher level features are derived from lower level features to form a hierarchical representation using nonlinear transformations. In this way, deep learning frameworks have exhibited several favorable advantages over traditional learning methods, i.e., deep architectures can be representationally efficient ; deep hierarchical representations allow non-local generalization and provide good comprehensibility, etc. Deep learning has already been successfully applied to a variety of applications with significant improvement in performance, e.g., image classification [Simonyan and Zisserman, 2014; He et al., 2015], object detection [Girshick et al., 2014; Girshick, 2015; Ren et al., 2015] and so on.

For the future research, it seems promising to combine deep learning concepts with tensor regression modeling. By taking advantage of deep structure, one can explore a novel class of nonlinear tensor regression models with powerful predictability. One possible direction is to find a framework using multiple levels of factors to represent the tensorial regression coefficients so as to have a better feature or latent components representation of data, which is expected to have a significantly enhanced accuracy. On the basis of that, one can further investigate possible solutions to speed up learning and prediction process of deep tensor regression framework.

Annexe A

Mathematical Background

A.1 Partial Least Squares Regression (PLS)

Partial least squares (PLS) regression is an regression approach that is particularly useful in predicting a set of depend variables from a very large set of independent predictors through the extraction of a small number of latent variables.

PLS regression developed by [Wold et al., 1984] originated in econometrics, it then gained its popularity in computational chemistry. Most recently, PLS regression is being widely used in many applications especially in medical imaging or neuroimaging data analysis [Abdi, 2010; Krishnan et al., 2011].

The primary goal of PLS regression is to predict $\mathbf{Y} \in \mathbb{R}^{I \times M}$ from $\mathbf{X} \in \mathbb{R}^{I \times J}$ by exploiting their common structure called latent variables (or latent vectors). More specifically, PLS regression first looks for a set of latent variables that performs a simultaneous decomposition of $\mathbf{X}$ and $\mathbf{Y}$ that projects both $\mathbf{X}$ and $\mathbf{Y}$ onto a new subspace, meanwhile with constraint that these latent vectors explain as much as possible of the covariance of $\mathbf{X}$ and $\mathbf{Y}$. Then, PLS carries out a regression step where the factors extracted from decomposition of $\mathbf{X}$ and $\mathbf{Y}$ are used to predict a new response. The standard PLS regression takes the form of

$$\begin{aligned} \mathbf{X} &= \mathbf{T}\mathbf{P}^{\mathsf{T}} + \mathbf{E} = \sum_{r=1}^{R} \mathbf{t}_r \mathbf{p}_r^{\mathsf{T}} + \mathbf{E}, \\ \mathbf{Y} &= \mathbf{U}\mathbf{Q}^{\mathsf{T}} + \mathbf{F} = \sum_{r=1}^{R} \mathbf{u}_r \mathbf{q}_r^{\mathsf{T}} + \mathbf{F}, \end{aligned} \tag{A.1}$$

where $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R]$ and $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_R]$ are extracted orthogonal latent vectors from $\mathbf{X}$ and $\mathbf{Y}$ respectively with the constraint that $\mathbf{T}$ and $\mathbf{U}$ have the largest covariance. R represents the matrix rank. Here $\mathbf{T}$ and $\mathbf{U}$ are also referred as the score matrices, and $\mathbf{P} = [\mathbf{p}_1, ..., \mathbf{p}_R]$ and $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_R]$ are called as the loading matrices, $\mathbf{E}$ and $\mathbf{F}$ are the residuals of $\mathbf{X}$ and $\mathbf{Y}$. Figure A.1 is the illustration of PLS model corresponding to equations (A.1). Note that

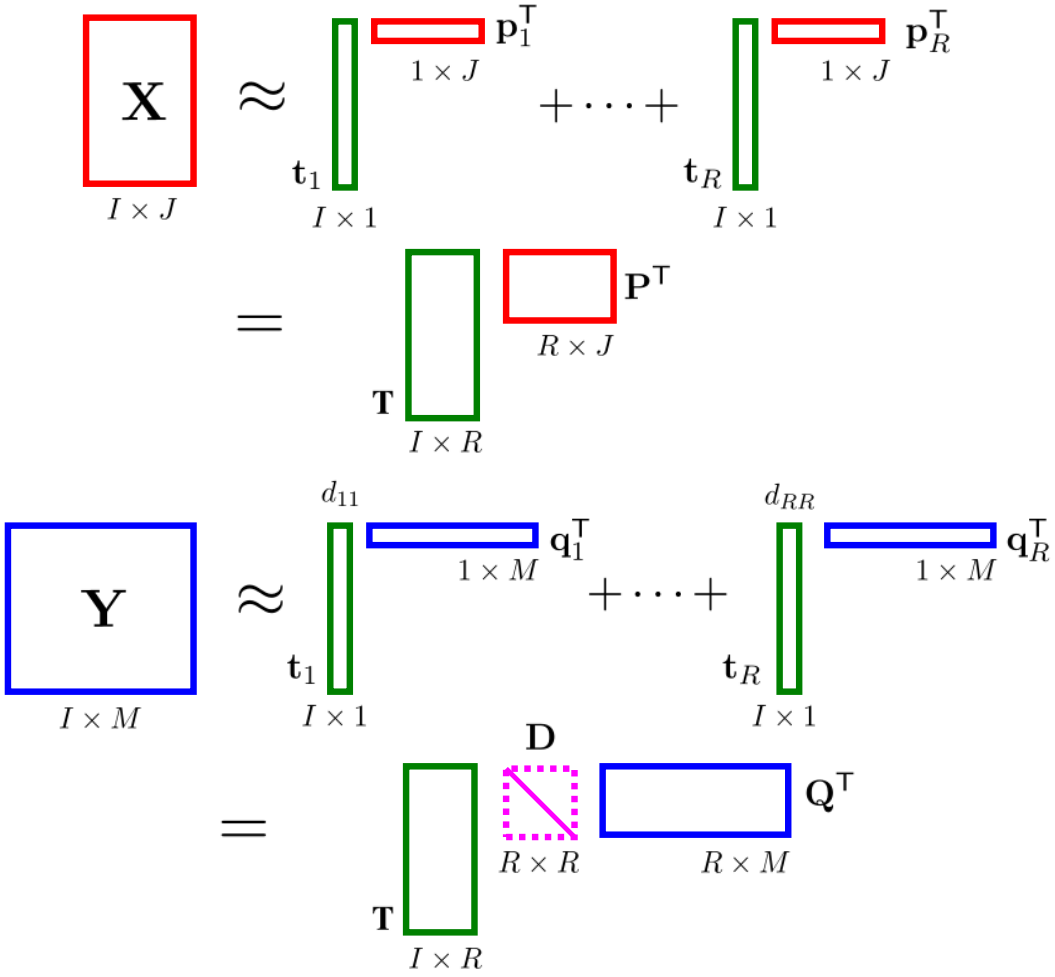


FIGURE A.1 – Illustration of the framework of partial least squares (PLS) regression.

$\mathbf{U}$ in (A.1) is depicted in expression of $\mathbf{u}_r = d_{rr}\mathbf{t}_r$, where the regression coefficients $\{d_{rr}\}_{r=1}^{R}$ are the elements of diagonal matrix $\mathbf{D}$. Both $\mathbf{X}$ and $\mathbf{Y}$ can be decomposed as sum of rank-one matrices.

To specify $\mathbf{T}$ and $\mathbf{U}$, we need to find two sets of weights $\mathbf{w}$ and $\mathbf{q}$ so as to create a linear combination of $\mathbf{X}$ as well as a linear combination of $\mathbf{Y}$ with the maximum covariance between them. Thus, we have $\mathbf{t} = \mathbf{X}\mathbf{w}$ and $\mathbf{u} = \mathbf{Y}\mathbf{q}$ and formulate the following optimization problem

$$\begin{aligned} &\max\nolimits_{\{\mathbf{w},\mathbf{q}\}}\{\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{Y}\mathbf{q}\} \\ &\text{s.t.} \quad \mathbf{w}^\mathsf{T}\mathbf{w} = 1 \quad \text{and} \quad \mathbf{q}^\mathsf{T}\mathbf{q} = 1. \end{aligned} \tag{A.2}$$

One is able to obtain the prediction for $\mathbf{Y}$ by substituting $\mathbf{U}$ with $\mathbf{T}$ and $\mathbf{D}$ as

$$\mathbf{Y} \approx \mathbf{T}\mathbf{D}\mathbf{Q}^T, \tag{A.3}$$

where $\mathbf{D} = diag(d_{11}, ..., d_{RR})$ is a diagonal matrix with weights with

$$d_{rr} = \frac{\mathbf{u}_r^T \mathbf{t}_r}{\mathbf{t}_r^T \mathbf{t}_r}. \tag{A.4}$$

The above equation demonstrates that one is interested in finding latent factor $\mathbf{T}$ both explain $\mathbf{X}$ and closely related $\mathbf{Y}$.

Wold [1975b] introduced a iterative procedure, namely nonlinear iterative partial least squares (NIPALS-PLS), to sequentially extract the latent vectors $\mathbf{t}$ and $\mathbf{u}$ from $\mathbf{X}$ and $\mathbf{Y}$ by means of a deflation strategy. In the deflation procedure, when the first pair of latent vectors $\mathbf{t}$ and $\mathbf{u}$ are found, their effects, in the form of rank-one matrices $\mathbf{t}\mathbf{p}^T$ and $\mathbf{u}\mathbf{q}^T$, are subtracted from both $\mathbf{X}$ and $\mathbf{Y}$, the procedure iterates repeatedly until $\mathbf{X}$ becomes the null matrix.

In order to handle sequential observations and time-dependent changes in the data, Qin [1998] extended the ordinary PLS regression to recursive partial least squares (RPLS) regression. More specifically, in addition to satisfy the conditions of (A.1), the following constraints are also held by construction

$$\begin{aligned} \mathbf{T}^\mathsf{T}\mathbf{E} &= 0, \\ \mathbf{T}^\mathsf{T}\mathbf{F} &= 0, \end{aligned} \tag{A.5}$$

meaning that the latent $\mathbf{T}$ should be orthogonal to residuals $\mathbf{E}$ and $\mathbf{F}$.

Furthermore, Qin [1998] shows that the residual $\mathbf{E}$ holds for the equation

$$\mathbf{E}^\mathsf{T}\mathbf{E} = 0. \tag{A.6}$$

if the rank R is sufficiently large, and this leads to

$$\begin{aligned} \mathbf{X}^\mathsf{T}\mathbf{X} &= \mathbf{P}\mathbf{P}^\mathsf{T}, \\ \mathbf{X}^\mathsf{T}\mathbf{Y} &= \mathbf{P}\mathbf{D}\mathbf{Q}^\mathsf{T}. \end{aligned} \tag{A.7}$$

(A.7) implies that $\mathbf{P}$ can be treated as summarization of $\mathbf{X}$. Likewise, $\mathbf{D}\mathbf{Q}^\mathsf{T}$ in (A.7) can be considered as summarization of $\mathbf{Y}$. In other words, the old data $\mathbf{X}$ and $\mathbf{Y}$ could be represented just by the factors $\mathbf{P}$ and $\mathbf{D}\mathbf{Q}^\mathsf{T}$ with fixed sizes. Therefore, at each timestamp, the new arriving pair of observations, i.e., $\{\mathbf{X}_{\mathsf{new}}, \mathbf{Y}_{\mathsf{new}}\}$, can be directly concatenated to a internal representation via $\mathbf{P}$ and $\mathbf{D}\mathbf{Q}^\mathsf{T}$, resulting in the following data representation with constant sizes

$$\left\{ \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_{\mathsf{new}} \end{bmatrix} \quad \begin{bmatrix} \mathbf{Y} \\ \mathbf{Y}_{\mathsf{new}} \end{bmatrix} \right\} = \left\{ \begin{bmatrix} \mathbf{P} \\ \mathbf{X}_{\mathsf{new}} \end{bmatrix} \quad \begin{bmatrix} \mathbf{D}\mathbf{Q}^\mathsf{T} \\ \mathbf{Y}_{\mathsf{new}} \end{bmatrix} \right\} \tag{A.8}$$

In this manner, the ordinary batch PLS can always be applied to (A.8) each time when new a pair of data block comes, resulting in a constant computation load overtime. This approach is called recursive PLS (RPLS).

Algorithm 9 Nonlinear Iterative Partial Least Squares (NIPALS)

1: **Input**: matrix $\mathbf{X}$
2: **Output**: latent vector $\mathbf{t}$ and weight vector $\mathbf{w}$
3: **Initialize**: randomly initialize $\mathbf{t}$
4: **repeat**
5: $\quad \mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{t}$
6: $\quad \mathbf{t} = \mathbf{X}\mathbf{w}$, $\mathbf{t} \leftarrow \mathbf{t}/\|\mathbf{t}\|$
7: **until** convergence
8: deflate matrix $\mathbf{X}$: $\mathbf{X} \leftarrow (\mathbf{X} - \mathbf{t}\mathbf{t}^\mathsf{T}\mathbf{X})$

Taking the time-varying process into account, the data representation of RPLS becomes

$$\left\{ \begin{bmatrix} \gamma\mathbf{P} \\ \mathbf{X}_{\mathsf{new}} \end{bmatrix} \quad \begin{bmatrix} \gamma\mathbf{D}\mathbf{Q}^\mathsf{T} \\ \mathbf{Y}_{\mathsf{new}} \end{bmatrix} \right\}, \tag{A.9}$$

where γ represents the forgetting factor usually ranging from 0 to 1.

A.2 Nonlinear Iterative Partial Least Squares PLS Regression (NIPALS-PLS)

The basic nonlinear iterative partial least squares (NIPALS) [Wold, 1966] is developed to solve the singular value decomposition problems and is closely related to the power method [Golub and Van Loan, 2012]. The NIPALS for one iteration is outlined in Algorithm 9. For the extraction of each pair of latent vector $\mathbf{t}$ and weight vector $\mathbf{w}$, the lines 5-6 are iteratively executed until convergence, which is followed by a deflation of $\mathbf{X}$ using $\mathbf{t}$ in line 8. The whole procedure can be performed repeatedly in order to extract multiple pairs of variables that are by construction orthogonal to each other.

Algorithm 10 Nonlinear Iterative Partial Least Squares for PLS Regression (NIPALS-PLS)

1: **Input**: input matrix $\mathbf{X}$, output matrix $\mathbf{Y}$
2: **Output**: latent vector $\mathbf{t}$ and $\mathbf{u}$, weight vector $\mathbf{w}$ and $\mathbf{q}$
3: **Initialize**: randomly initialize $\mathbf{u}$
4: **repeat**
5: $\quad \mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{u}$
6: $\quad \mathbf{t} = \mathbf{X}\mathbf{w}$, $\mathbf{t} \leftarrow \mathbf{t}/\|\mathbf{t}\|$
7: $\quad \mathbf{q} = \mathbf{Y}^\mathsf{T}\mathbf{t}$
8: $\quad \mathbf{u} = \mathbf{Y}\mathbf{q}$, $\mathbf{u} \leftarrow \mathbf{u}/\|\mathbf{u}\|$
9: **until** convergence
10: deflate matrix $\mathbf{X}$: $\mathbf{X} \leftarrow (\mathbf{X} - \mathbf{t}\mathbf{t}^\mathsf{T}\mathbf{X})$
11: deflate matrix $\mathbf{Y}$: $\mathbf{Y} \leftarrow (\mathbf{Y} - \mathbf{t}\mathbf{t}^\mathsf{T}\mathbf{Y})$

As have been shown in the previous section, in order to establish a linear relationship between the input variable $\mathbf{X} \in \mathbb{R}^{I \times J}$ and the output variable $\mathbf{Y} \in \mathbb{R}^{I \times M}$, PLS applies a linear combination of $\mathbf{X}$ and a linear combination of $\mathbf{Y}$ to create a collection of uncorrelated latent vectors $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_R] \in \mathbb{R}^{I \times R}$ and a collection of uncorrelated latent vectors $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_R] \in \mathbb{R}^{I \times R}$ respectively, such that $\mathbf{T}$ and $\mathbf{U}$ have the maximum covariance between them. The weight vector $\mathbf{w}_r$ of combination of input variable $\mathbf{X}$ and weight vector $\mathbf{q}_r$ of combination of output variable $\mathbf{Y}$ are proportional to the covariance when the corresponding latent vectors $\mathbf{t}_r$ and $\mathbf{u}_r$ are extracted (as reflected in (A.2)), with $r = 1, ..., R$. Having obtained all latent vectors $\mathbf{T}$ and $\mathbf{U}$, a least squares regression step can be carried out against $\mathbf{T}$ and $\mathbf{U}$.

To determine these latent vectors and weights, Wold et al. [1984] adapted the basic NIPALS algorithm to the PLS regression, leading to the NIPALS-PLS algorithm. The procedure of NIPALS-PLS for one iteration is summarized in Algorithm 10. Similar to NIPALS, NIPALS-PLS can be used to sequentially extract latent vectors and weights but from both input and output matrices.

A.3 Linear Algebra Basics

A.3.1 Eigenvalue Decomposition

Assume a nonzero square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, scalars $\lambda \in \mathbb{R}$ and nonzero vectors $\mathbf{u} \in \mathbb{R}^n$ that satisfy the following equation

$$\mathbf{A}\mathbf{u} = \lambda\mathbf{u}, \tag{A.10}$$

or in an equivalent form

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = 0, \tag{A.11}$$

then $\mathbf{u}$ is called *eigenvector* and λ is called *eigenvalue* associated with eigenvector $\mathbf{u}$.

Let $\{\lambda_i\}_{i=1}^n$ be a set of eigenvalues and let $\{\mathbf{u}_i\}_{i=1}^n$ be a set of corresponding eigenvectors. Typically, the eigenvectors can be rearranged into the columns of matrix $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_n]$, while the associated eigenvalues are stored in a diagonal matrix $\Lambda = diag(\lambda_1, ..., \lambda_n)$. Then, equation (A.10) becomes

$$\mathbf{A}\mathbf{U} = \mathbf{U}\Lambda. \tag{A.12}$$

Assume that the eigenvectors are linearly independent, then $\mathbf{U}^{-1}$ exists, and equation (A.12) can be rewritten as

$$\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^{-1}, \tag{A.13}$$

which is known as the *eigenvalue decomposition* (ED) of the matrix $\mathbf{A}$.

Furthermore, if $\mathbf{A} \in \mathbb{R}^{n \times n}$ is real symmetric matrix, then $\mathbf{A}$ can be factorized as

$$\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^{\mathsf{T}}, \tag{A.14}$$

where $\mathbf{U}$ is orthogonal, e.g., satisfies $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$. The diagonal Λ contains $\{\lambda_i\}_{i=1}^n$ with real numbers. The factorization (A.14) is call *symmetric eigenvalue decomposition* (SED) of the matrix $\mathbf{A}$. In an alternative form, SED also reads

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\mathsf{T}. \tag{A.15}$$

A.3.2 Singular Value Decomposition

Suppose a nonzero matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $rank(\mathbf{A}) = r$. Then there exists matrices $\mathbf{U} \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$ such that $\mathbf{U}$ and $\mathbf{V}$ are isometries, that is, $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{I}$, and $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_R)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\mathsf{T}. \tag{A.16}$$

The factorization in (A.16) is called *singular value decomposition* (SVD) of $\mathbf{A}$. The columns of $\mathbf{U}$ are orthonormal vectors called *left singular vectors*, the columns of $\mathbf{V}$ are orthonormal vectors referred to *right singular vectors* and entries $\{\sigma_i\}_{i=1}^r$ of Σ, which are uniquely determined, are called *singular values* of $\mathbf{A}$. Equivalently, SVD can also be formed as

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\mathsf{T}, \tag{A.17}$$

where $\mathbf{u}_i \in \mathbb{R}^m$ corresponds to the left singular vector and $\mathbf{v}_i \in \mathbb{R}^n$ corresponds to the right singular vector.

The SVD of a matrix $\mathbf{A}$ closely relates to the SED of nonnegative definite matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ in the following way

$$\mathbf{A}^\mathsf{T}\mathbf{A} = \mathbf{V}\Sigma^2\mathbf{V}^\mathsf{T} = [\mathbf{V}, \hat{\mathbf{V}}] \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} [\mathbf{V}, \hat{\mathbf{V}}]^\mathsf{T}, \tag{A.18}$$

where $[\mathbf{V}, \hat{\mathbf{V}}]$ is orthogonal. From equation (A.18), it is not hard to see that nonzero eigenvalues of $\mathbf{A}^\mathsf{T}\mathbf{A}$ are the square of singular values of $\mathbf{A}$, and the associated eigenvectors of $\mathbf{A}^\mathsf{T}\mathbf{A}$ are the right singular vectors of $\mathbf{A}$.

A.4 Generalized Linear Model (GLM)

The generalized linear model (GLM) was proposed in [Nelder and Baker, 1972; McCullagh and Nelder, 1989] whose response variable y is from an exponential family distribution including Normal, Gamma, Binomial and Poisson distributions and can be defined as

$$p(y|\theta, \phi) = \exp\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi)\} \tag{A.19}$$

where θ and ϕ are the natural and dispersion parameters, respectively.

In classical GLM, the relationship between the vector predictor $\mathbf{x} \in \mathbb{R}^I$ and the mean $\mu = \mathbf{E}(y|\mathbf{x})$ of response $y \in \mathbb{R}$ of (A.19) is established by a strictly increasing link function $g(\cdot)$ as

$$\begin{aligned} g(\mu) &= \eta \\ &= \beta^{\mathsf{T}}\mathbf{x} + b, \end{aligned} \tag{A.20}$$

or equivalently,

$$\mu = g^{-1}(\eta) \tag{A.21}$$

$$= g^{-1}(\beta^{\mathsf{T}}\mathbf{x} + b), \tag{A.22}$$

where η denotes the linear systematic part of GLM consisting of the coefficient vector $\beta \in \mathbb{R}^I$ and the intercept $b \in \mathbb{R}$. Most commonly used standard link functions $g(\cdot)$ are Identity (Normal), Inverse (Gamma), Logit (Binomial) and Log (Poisson).

A.5 Maximum Likelihood Estimation (MLE)

The maximum likelihood estimation (MLE) procedure is a most commonly used approach for parameter estimation. Specifically, let $p(x;\theta)$ be a parametric model, where $p(x)$ represents a probability density function of x that is parameterized by a parameter vector $\theta = (\theta_1, ..., \theta_m)$ with m dimensionality. Given n training samples $\{x_i\}_{i=1}^n$, the goal of MLE is to determine parameter θ such that the probability of obtaining these training samples is maximized. For this propose, the probability that training samples $\{x_i\}_{i=1}^n$ are generated under parameter θ can be viewed as a function of θ. Such function denoted by $\ell(\theta)$ is called *likelihood*. With an i.i.d assumption, $\ell(\theta)$ can be formulated as

$$\ell(\theta) = \prod_{i=1}^{n} p(x_i;\theta). \tag{A.23}$$

Therefore, the ML estimator $\hat{\theta}_{ML}$ can be found by maximizing the likelihood function $\ell(\theta)$

$$\hat{\theta}_{ML} = \arg\max_{\theta\in\Theta} \ell(\theta) = \arg\max_{\theta\in\Theta} \prod_{i=1}^{n} p(x_i;\theta) \tag{A.24}$$

Having known $\hat{\theta}_{ML}$, the density estimator $\hat{p}(x)$ can be obtained by

$$\hat{p}(x) = p(x;\hat{\theta}_{ML}). \tag{A.25}$$

In practice, it is usually more convenient and computationally easy to use log-likelihood $\log \ell(\theta)$ instead of likelihood $\ell(\theta)$. One can easily verify that the maximizer of $\log \ell(\theta)$ is same as the maximizer of $\ell(\theta)$, which is due to the fact that log is a monotone increasing function. Thus, the ML estimator $\hat{\theta}_{ML}$ of $\log \ell(\theta)$ function turns out to be

$$\hat{\theta}_{ML} = \arg\max_{\theta\in\Theta} \log \ell(\theta) = \arg\max_{\theta\in\Theta} \sum_{i=1}^{n} \log p(x_i;\theta). \tag{A.26}$$

Annexe B

Sketch of Proof

B.1 Sketch of Proof of Proposition 3

The proof of global convergence property can be directly derived from the standard theory for algorithms with monotone increasing objective functions. Since the block relaxation algorithm is a monotone increasing algorithm, the global convergence can be achieved under the following conditions [De Leeuw, 1994; Lange, 2004, 2010] :

1. ℓ has a compact set $\{\theta : \ell(\theta) \geq \ell(\theta)^{[0]}\}$;
2. ℓ has a set of isolated stationary points ;
3. the algorithmic mapping M is continuous ;
4. $\theta^{[m]}$ is a fixed point of algorithm if and only if $\theta^{[m]}$ is a stationary point of ℓ ;
5. $\theta^{[m]}$ is fixed point of algorithm if and only if $\ell(\theta^{[m+1]}) \geq \ell(\theta^{[m]})$ with equality.

The arguments of proving our block relaxation algorithm for $\mathcal{H}$-Tucker model satisfying above conditions are similar to those of [Li et al., 2013] except condition 4. To see condition 4 is satisfied, according to Fermat's theorem (theorem of stationary points), $\theta^{[m]} = \{\beta, \{\mathbf{U}^t\}_{t\in\mathcal{L}(\mathcal{T})}, \{\mathbf{B}^t\}_{t\in\mathcal{N}(\mathcal{T})}\}$ is a fixed point of the block relaxation algorithm when $D\ell(\beta) = 0$, $D\ell(\mathbf{B}^t) = 0$ for $t \in \mathcal{N}(\mathcal{T})$ and $D\ell(\mathbf{U}^t) = 0$ for $t \in \mathcal{L}(\mathcal{T})$. Therefore, $\theta^{[m]}$ is a fixed point of $\mathcal{H}$-Tucker block relaxation algorithm if and only if it is a stationary point of ℓ. As for the satisfaction of other conditions, see [Li et al., 2013] for more details.

The local convergence property is derived from Ostrowski theorem. The theorem states that the sequence $\theta^{[m+1]} = M(\theta^{[m]})$ is locally attracted to local maximum $\theta^{[\infty]}$ if the spectral radius of the differential of the algorithmic map is less than 1, i.e., $\rho(DM(\theta^{[\infty]})) < 1$. This condition is satisfied based on assumption 2 of proposition, see [Zhou et al., 2013; Li et al., 2013] for more details.

Bibliographie

Aa, N., Luo, X., Giezeman, G., Tan, R., and Veltkamp, R. (2011). Utrecht multi-person motion (UMPM) benchmark : A multi-person dataset with synchronized video and motion capture data for evaluation of articulated human motion and interaction. In *HICV Workshop in IEEE International Conference on Computer Vision (ICCV'11)*.

Abdi, H. (2010). Partial least squares regression and projection on latent structure regression (PLS regression). *Wiley Interdisciplinary Reviews : Computational Statistics*, 2(1) :97–106.

Acar, E., Çamtepe, S. A., Krishnamoorthy, M. S., and Yener, B. (2005). Modeling and multiway analysis of chatroom tensors. In *International Conference on Intelligence and Security Informatics (ISI'05)*, pages 256–268. Springer.

Acar, E. and Yener, B. (2009). Unsupervised multiway data analysis : A literature survey. *IEEE Transactions on Knowledge and Data Engineering*, 21(1) :6–20.

ADHD (2014). *The Burner Data of ADHD-200 Consortium.* ADHD-200 Consortium.

Andersen, A. H. and Rayens, W. S. (2004). Structure-seeking multilinear methods for the analysis of fMRI data. *NeuroImage*, 22(2) :728–739.

Andersen, C. M. and Bro, R. (2003). Practical aspects of PARAFAC modeling of fluorescence excitation-emission data. *Journal of Chemometrics*, 17(4) :200–215.

Araujo, M., Papadimitriou, S., Günnemann, S., Faloutsos, C., Basu, P., Swami, A., Papalexakis, E. E., and Koutra, D. (2014). Com2 : Fast automatic discovery of temporal (comet) communities. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'14)*, pages 271–283. Springer.

Babić, D., Klein, D., Lukovits, I., Nikolić, S., and Trinajstić, N. (2002). Resistance-distance matrix : A computational algorithm and its application. *International Journal of Quantum Chemistry*, 90(1) :166–176.

Bahadori, M. T., Yu, Q. R., and Liu, Y. (2014). Fast multivariate spatio-temporal analysis via low rank tensor learning. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 3491–3499.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning : A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8) :1798–1828.

Beutel, A., Talukdar, P. P., Kumar, A., Faloutsos, C., Papalexakis, E. E., and Xing, E. P. (2014). FlexiFaCT : Scalable flexible factorization of coupled tensors on hadoop. In *SIAM International Conference on Data Mining (SDM'14)*, pages 109–117. SIAM.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Blumensath, T. and Davies, M. E. (2009). Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3) :265–274.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1) :1–122.

Bro, R. (1996). Multiway calidration. multilinear PLS. *Journal of Chemometrics*, 10 :47–61.

Bro, R. (2006). Review on multiway analysis in chemistry 2000–2005. *Critical Reviews in Analytical Chemistry*, 36(3-4) :279–293.

Bro, R. and Andersson, C. A. (1998). Improving the speed of multiway algorithms Part II : Compression. *Chemometrics and Intelligent Laboratory Systems*, 42(1) :105–113.

Bro, R., Harshman, R. A., Sidiropoulos, N. D., and Lundy, M. E. (2009). Modeling multi-way data with linearly dependent loadings. *Journal of Chemometrics*, 23(7-8) :324–340.

Caffo, B. S., Crainiceanu, C. M., Verduzco, G., Joel, S., Mostofsky, S. H., Bassett, S. S., and Pekar, J. J. (2010). Two-stage decompositions for the analysis of functional connectivity for fmri with application to alzheimer's disease risk. *NeuroImage*, 51(3) :1140–1149.

Cameron, A. C. and Trivedi, P. K. (2013). *Regression Analysis of Count Data*, volume 53. Cambridge University Press.

Candes, E. J., Romberg, J. K., and Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8) :1207–1223.

Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart-Young decomposition. *Psychometrika*, 35(3) :283–319.

Chao, Z. C., Nagasaka, Y., and Fujii, N. (2010). Long-term asynchronous decoding of arm motion using electrocorticographic signals in monkeys. *Frontiers in Neuroengineering*, 3.

Chatterjee, S. and Hadi, A. S. (2015). *Regression Analysis by Example*. John Wiley & Sons.

Choi, J. H. and Vishwanathan, S. (2014). DFacTo : Distributed factorization of tensors. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 1296–1304.

Cichocki, A. (2013). Tensor decompositions : a new concept in brain data analysis ? *arXiv preprint arXiv :1305.0395*.

Cichocki, A. (2014). Tensor networks for big data analytics and large-scale optimization problems. *arXiv preprint arXiv :1407.3124*.

Cichocki, A. (2018). *Tensor Networks for Dimensionality Reduction, Big Data and Deep Learning*. Springer International Publishing.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. (2016). Tensor networks for dimensionality reduction and large-scale optimization : Part 1 low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4-5) :249–429.

Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., and Phan, H. A. (2015). Tensor decompositions for signal processing applications : from two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2) :145–163.

Cichocki, A., Phan, A.-H., Zhao, Q., Lee, N., Oseledets, I., Sugiyama, M., and Mandic, D. (2017). Tensor networks for dimensionality reduction and large-scale optimization : Part 2 applications and future perspectives. *Foundations and Trends in Machine Learning*, 9(6) :431–673.

Cichocki, A., Zdunek, R., Phan, A. H., and Amari, S.-i. (2009). *Nonnegative Matrix and Tensor Factorizations*: *Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley & Sons.

Clarkson, K. L. and Woodruff, D. P. (2013). Low rank approximation and regression in input sparsity time. In *ACM Symposium on Theory of Computing (STOC'13)*, pages 81–90. ACM.

Cong, F., Lin, Q.-H., Kuang, L.-D., Gong, X.-F., Astikainen, P., and Ristaniemi, T. (2015). Tensor decomposition of EEG signals : A brief review. *Journal of Neuroscience Methods*, 248 :59–69.

Cressie, N. and Huang, H.-C. (1999). Classes of nonseparable, spatio-temporal stationary covariance functions. *Journal of the American Statistical Association*, 94(448) :1330–1339.

Cressie, N. and Wikle, C. K. (2015). *Statistics for Spatio-temporal Data*. John Wiley & Sons.

Cvetković, D. M., Doob, M., and Sachs, H. (1980). *Spectra of Graphs*: *Theory and Application*, volume 87. Academic Press.

Daubechies, I. (1992). *Ten Lectures on Wavelets*, volume 61. SIAM.

De Lathauwer, L. (2008a). Decompositions of a higher-order tensor in block terms-Part I : Lemmas for partitioned matrices. *SIAM Journal on Matrix Analysis and Applications*, 30(3) :1022–1032.

De Lathauwer, L. (2008b). Decompositions of a higher-order tensor in block terms-Part II : Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3) :1033–1066.

De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000a). A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4) :1253–1278.

De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000b). On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4) :1324–1342.

De Lathauwer, L. and Nion, D. (2008). Decompositions of a higher-order tensor in block terms-Part III : Alternating least squares algorithms. *SIAM Journal on Matrix Analysis and Applications*, 30(3) :1067–1083.

De Leeuw, J. (1994). Block-relaxation algorithms in statistics. In *Information Systems and Data Analysis*, pages 308–324. Springer.

Deisenroth, M. P., Huber, M. F., and Hanebeck, U. D. (2009). Analytic moment-based gaussian process filtering. In *International Conference on Machine Learning (ICML'09)*.

Draper, N. R. and Smith, H. (2014). *Applied Regression Analysis*. John Wiley & Sons.

Eckstein, J. and Bertsekas, D. P. (1992). On the douglas rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3) :293–318.

Eliseyev, A. and Aksenova, T. (2013). Recursive N-way partial least squares for brain-computer interface. *PloS One*, 8(7) :e69962.

Erdos, D. and Miettinen, P. (2013). Walk'n'Merge : A scalable algorithm for boolean tensor factorization. In *IEEE International Conference on Data Mining (ICDM'13)*, pages 1037–1042. IEEE.

Gandy, S., Recht, B., and Yamada, I. (2011). Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse Problems*, 27(2) :025010.

Girshick, R. (2015). Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV'15)*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'14)*, pages 580–587.

Golub, G. H. and Van Loan, C. F. (2012). *Matrix Computations*, volume 3. JHU Press.

Grasedyck, L. (2010). Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4) :2029–2054.

Guo, W., Kotsia, I., and Patras, I. (2012). Tensor learning for regression. *IEEE Transactions on Image Processing*, 21(2) :816–827.

Hackbusch, W. and Kühn, S. (2009). A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5) :706–722.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure : Models and conditions for an explanatory multi-modal factor analysis. *UCLA working papers in phonetics*, 16 :1–84.

Harshman, R. A. (1972). PARAFAC2 : Mathematical and technical notes. *UCLA Working Papers in Phonetics*, 22 :30–47.

Harshman, R. A., Hong, S., and Lundy, M. E. (2003). Shifted factor analysis-Part I : Models and properties. *Journal of Chemometrics*, 17(7) :363–378.

Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.-M., and Larochelle, H. (2016). Brain tumor segmentation with deep neural networks. *Medical Image Analysis*.

Haykin, S. and Network, N. (2004). A comprehensive foundation. *Neural Networks*, 2(2004).

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *ArXiv Preprint ArXiv* : *1512.03385*.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7) :1527–1554.

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1) :164–189.

Hitchcock, F. L. (1928). Multiple invariants and generalized rank of a P-way matrix or tensor. *Journal of Mathematics and Physics*, 7(1) :39–79.

Hosmer, D. W. and Lemeshow, S. (2000). Introduction to the logistic regression model. *Applied Logistic Regression, Second Edition*, pages 1–30.

Hou, M. and Chaib-draa, B. (2015). Hierarchical tucker tensor regression : Application to brain imaging data analysis. In *IEEE International Conference on Image Processing (ICIP'15)*, pages 1344–1348. IEEE.

Hou, M. and Chaib-draa, B. (2016). Online incremental higher-order partial least squares regression for fast reconstruction of motion trajectories from tensor streams. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'16)*, pages 6205–6209. IEEE.

Hou, M. and Chaib-draa, B. (2017). Fast recursive low-rank tensor learning for regression. In *Internatiaonal Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 1851–1857.

Hou, M., Wang, Y., and Chaib-draa, B. (2015). Online local gaussian process for tensor-variate regression : Application to fast reconstruction of limb movements from brain signal. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*, pages 5490–5494. IEEE.

Hou, M., Zhao, Q., Chaib-draa, B., and Cichocki, A. (2016). Common and discriminative subspace kernel-based multiblock tensor partial least squares regression. In *Association for the Advancement of Artificial Intelligence Conference (AAAI'16)*.

Ishteva, M., Absil, P., Van Huffel, S., and De Lathauwer, L. (2011). Tucker compression and local optima. *Chemometrics and Intelligent Laboratory Systems*, 106(1) :57–64.

Jeon, I., Papalexakis, E. E., Kang, U., and Faloutsos, C. (2015). Haten2 : Billion-scale tensor decompositions. In *IEEE International Conference on Data Engineering (ICDE'15)*, pages 1047–1058. IEEE.

Jolliffe, I. (2002). *Principal Component Analysis*. Wiley Online Library.

Kang, U., Papalexakis, E., Harpale, A., and Faloutsos, C. (2012). Gigatensor : Scaling tensor analysis up by 100 times-algorithms and discoveries. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'12)*, pages 316–324. ACM.

Kiers, H. A. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3) :105–122.

Kim, H., Zhou, J. X., Morse III, H. C., and Park, H. (2005). A three-stage framework for gene expression data analysis by L1-norm support vector regression. *International Journal of Bioinformatics Research and Applications*, 1(1) :51–62.

Kolda, T. G. (2003). A counterexample to the possibility of an extension of the Eckart-Young low-rank approximation theorem for the orthogonal rank tensor. *SIAM Journal on Matrix Analysis and Applications*, 24(3) :762–767.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3).

Kolda, T. G., Bader, B. W., and Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE.

Kolda, T. G. and Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *IEEE International Conference on Data Mining (ICDM'08)*, pages 363–372. IEEE.

Kossaifi, J., Lipton, Z. C., Khanna, A., Furlanello, T., and Anandkumar, A. (2017). Tensor regression networks. *arXiv preprint arXiv :1707.08308*.

Krishnan, A., Williams, L. J., McIntosh, A. R., and Abdi, H. (2011). Partial least squares (PLS) methods for neuroimaging: A tutorial and review. *Neuroimage*, 56(2) :455–475.

Kroonenberg, P. M. and De Leeuw, J. (1980). Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1) :69–97.

Kruskal, J. B. (1989). Rank, decomposition, and uniqueness for 3-way and N-way arrays. In *Multiway Data Analysis*, pages 7–18. North-Holland Publishing Company.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22(1) :79–86.

Lange, K. (2004). *Optimization*. Springer Texts in Statistics. Springer-Verlag, New York.

Lange, K. (2010). *Numerical Analysis for Statisticians*. Springer Science & Business Media.

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10 :1–40.

Li, X., Zhou, H., and Li, L. (2013). Tucker tensor regression and neuroimaging analysis. *ArXiv Preprint ArXiv*: *1304.5637*.

Lin, Y. (2000). Tensor product space anova models. *Annals of Statistics*, pages 734–755.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2) :129–137.

Lozano, A. C., Li, H., Niculescu-Mizil, A., Liu, Y., Perlich, C., Hosking, J., and Abe, N. (2009). Spatial-temporal causal modeling for climate change attribution. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, pages 587–596. ACM.

Luo, L., Bao, S., and Gao, Z. (2015). Quality prediction based on HOPLS-CP for batch processes. *Chemometrics and Intelligent Laboratory Systems*, 143 :28–39.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, volume 37. CRC Press.

Miwakeichi, F., Martınez-Montes, E., Valdés-Sosa, P. A., Nishiyama, N., Mizuhara, H., and Yamaguchi, Y. (2004). Decomposing eeg data into space–time–frequency components using parallel factor analysis. *NeuroImage*, 22(3) :1035–1045.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (2015). *Introduction to Linear Regression Analysis*. John Wiley & Sons.

Mørup, M. and Schmidt, M. N. (2006). *Sparse non-negative tensor 2D deconvolution (SNTF2D) for multi channel time-frequency analysis*. Technical University of Denmark.

Mukherjee, A. and Zhu, J. (2011). Reduced rank ridge regression and its kernel extensions. *Statistical analysis and data mining*: *the ASA data science journal*, 4(6) :612–622.

Nelder, J. A. and Baker, J. (1972). *Generalized linear models*. Wiley Online Library.

Nguyen-Tuong, D., Peters, J. R., and Seeger, M. (2009). Local gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems (NIPS'09)*, pages 1193–1200.

Nion, D. and De Lathauwer, L. (2008). A block component model-based blind DS-CDMA receiver. *IEEE Transactions on Signal processing*, 56(11) :5567–5579.

Ofli, F., Chaudhry, R., Kurillo, G., Vidal, R., and Bajcsy, R. (2013). Berkeley MHAD : A comprehensive multimodal human action database. In *IEEE Winter Conference on Applications of Computer Vision (WACV'13)*, pages 53–60. IEEE.

O'Hara, M. J. (2010). On low-rank updates to the singular value and tucker decompositions. In *SIAM International Conference on Data Mining (SDM'10)*, pages 713–719. SIAM.

Oommen, T., Misra, D., Twarakavi, N. K., Prakash, A., Sahoo, B., and Bandopadhyay, S. (2008). An objective analysis of support vector machine based classification for remote sensing. *Mathematical Geosciences*, 40(4) :409–424.

Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2012). Parcube : Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD'12)*, pages 521–536. Springer.

Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2016). Tensors for data mining and data fusion : Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology*, 8(2) :1–44.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. (1993). Orthogonal matching pursuit : Recursive function approximation with applications to wavelet decomposition. In

Asilomar Conference on Signals, Systems and Computers (ASILOMAR'93), pages 40–44. IEEE.

Qin, S.-Z. J. (1998). Recursive PLS algorithms for adaptive data modeling. *Computers and Chemical Engineering*, 22(4) :503–514.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1) :81–106.

Rabusseau, G. and Kadri, H. (2016). Low-rank regression with tensor responses. In *Advances in Neural Information Processing Systems (NIPS'16)*, pages 1867–1875.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning*. MIT Press.

Ravindran, N., Sidiropoulos, N. D., Smith, S., and Karypis, G. (2014). Memory-efficient parallel computation of tensor and matrix products for big tensor decomposition. In *Asilomar Conference on Signals, Systems and Computers (ASILOMAR'14)*, pages 581–585. IEEE.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN : Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS'15)*, pages 91–99.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5) :877–898.

Romera-Paredes, B., Aung, H., Bianchi-Berthouze, N., and Pontil, M. (2013). Multilinear multitask learning. In *International Conference on Machine Learning (ICML'13)*, pages 1444–1452.

Rosipal, R. and Trejo, L. J. (2002). Kernel partial least squares regression in reproducing kernel hilbert space. *Journal of Machine Learning Research*, 2 :97–123.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6(2) :461–464.

Shin, K. and Kang, U. (2014). Distributed methods for high-dimensional and large-scale tensor factorization. In *IEEE International Conference on Data Mining (ICDM'14)*, pages 989–994. IEEE.

Sidiropoulos, N. D., Papalexakis, E. E., and Faloutsos, C. (2014). Parallel randomly compressed cubes : A scalable distributed architecture for big tensor decomposition. *IEEE Signal Processing Magazine*, 31(5) :57–70.

Signoretto, M., De Lathauwer, L., and Suykens, J. A. (2011). A kernel-based framework to tensorial data analysis. *Neural Networks*, 24(8) :861–874.

Signoretto, M., Dinh, Q. T., De Lathauwer, L., and Suykens, J. A. (2014). Learning with tensors : A framework based on convex optimization and spectral regularization. *Machine Learning*, 94(3) :303–351.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *ArXiv Preprint ArXiv*: *1409.1556*.

Smaragdis, P. (2004). Non-negative matrix factor deconvolution ; extraction of multiple sound sources from monophonic inputs. In *International Conference on Independent Component Analysis and Signal Separation (ICA'04)*, pages 494–499. Springer.

Smilde, A., Bro, R., and Geladi, P. (2005). *Multi-way Analysis*: *Applications in the Chemical Sciences*. John Wiley & Sons.

Smilde, A. K., Westerhuis, J. A., and Boque, R. (2000). Multiway multiblock component and covariates regression models. *Journal of Chemometrics*, 14(3) :301–331.

Smith, S., Ravindran, N., Sidiropoulos, N. D., and Karypis, G. (2015). SPLATT : Efficient and parallel sparse tensor-matrix multiplication. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'15)*, pages 61–70. IEEE.

Smola, A. and Vapnik, V. (1997). Support vector regression machines. In *Advances in Neural Information Processing Systems (NIPS'97)*, volume 9, pages 155–161.

Sun, J., Tao, D., Papadimitriou, S., Yu, P. S., and Faloutsos, C. (2008). Incremental tensor analysis : Theory and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD'08)*, 2(3) :11.

Tsourakakis, C. E. (2010). MACH : Fast randomized tensor decompositions. In *SIAM International Conference on Data Mining (SDM'10)*, pages 689–700. SIAM.

Tucker, L. R. (1963). Implications of factor analysis of three-way matrices for measurement of change. In *Problems in measuring change*, pages 122–137. University of Wisconsin Press.

Tucker, L. R. (1964). The extension of factor analysis to three-dimensional matrices. In *Contributions to mathematical psychology*, pages 110–127. Holt, Rinehart and Winston.

Turaga, P., Veeraraghavan, A., Srivastava, A., and Chellappa, R. (2010). Statistical analysis on manifolds and its applications to video analysis. In *Video Search and Mining*, pages 115–144. Springer.

Turaga, P., Veeraraghavan, A., Srivastava, A., and Chellappa, R. (2011). Statistical computations on grassmann and stiefel manifolds for image and video-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11) :2273–2286.

Urtasun, R. and Darrell, T. (2008). Sparse probabilistic regression for activity-independent human pose inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, pages 1–8. IEEE.

Vasilescu, M. and Terzopoulos, D. (2002). Multilinear analysis of image ensembles : Tensorfaces. In *European Conference on Computer Vision (ECCV'02)*, pages 447–460. Springer.

Wang, H. and Ahuja, N. (2008). A tensor approximation approach to dimensionality reduction. *International Journal of Computer Vision*, 76(3) :217–229.

Wang, Y., Tung, H.-Y., Smola, A. J., and Anandkumar, A. (2015). Fast and guaranteed tensor decomposition via sketching. In *Advances in Neural Information Processing Systems (NIPS'15)*, pages 991–999.

Wang, Y. L. (2014). *Interactions between Gaussian processes and Bayesian estimation.* PhD thesis, Université Laval.

Westerhuis, J. A., Kourti, T., and MacGregor, J. F. (1998). Analysis of multiblock and hierarchical PCA and PLS models. *Journal of Chemometrics*, 12(5) :301–321.

Wimalawarne, K., Sugiyama, M., and Tomioka, R. (2014). Multitask learning meets tensor factorization : Task imputation via convex optimization. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 2825–2833.

Wold, H. (1966). Estimation of principal components and related models by iterative least squares. *Multivariate Analysis*, 45(1) :391–420.

Wold, H. (1975a). Path models with latent variables : The NIPALS approach. In *Quantitative Sociology* : *International Perspectives on Mathematical and Statistical Model Building.* Academic Press.

Wold, H. (1975b). Soft modeling by latent variables : The nonlinear iterative partial least squares approach. *Perspectives in Probability and Statistics*, pages 520–540.

Wold, S., Ruhe, A., Wold, H., and Dunn, III, W. (1984). The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses. *SIAM Journal on Scientific and Statistical Computing*, 5(3) :735–743.

Yu, R., Cheng, D., and Liu, Y. (2015). Accelerated online low-rank tensor learning for multivariate spatio-temporal streams. In *International Conference on Machine Learning (ICML'15).*

Yu, R. and Liu, Y. (2016). Learning from multiway data : Simple and efficient tensor regression. In *International Conference on Machine Learning (ICML'16).*

Zhao, Q., Caiafa, C. F., Mandic, D. P., Chao, Z. C., Nagasaka, Y., Fujii, N., Zhang, L., and Cichocki, A. (2013a). Higher order partial least squares (HOPLS) : A generalized multilinear regression method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7) :1660–1673.

Zhao, Q., Caiafa, C. F., Mandic, D. P., Zhang, L., Ball, T., Schulze-Bonhage, A., and Cichocki, A. (2011). Multilinear subspace regression : An orthogonal tensor decomposition approach. In *Advances in Neural Information Processing Systems (NIPS'11)*, volume 2011, pages 1269–1277.

Zhao, Q., Zhou, G., Adali, T., Zhang, L., and Cichocki, A. (2013b). Kernelization of tensor-based models for multiway data analysis : Processing of multidimensional structured data. *IEEE Signal Processing Magazine*, 30(4) :137–148.

Zhao, Q., Zhou, G., Zhang, L., and Cichocki, A. (2014). Tensor-variate gaussian processes regression and its application to video surveillance. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*, pages 1265–1269. IEEE.

Zhou, H., Li, L., and Zhu, H. (2013). Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502) :540–552.