Contents

Chapter 1

Introduction

A spatiotemporal dataset contains both information about time and space. It may come from any events that describe phenomena that exist at certain combinations of time and space. Therefore, spatiotemporal data analysis has great real-life implications, such as identifying traffic hot spots and determining healthcare distributions [5]. The main goal of spatiotemporal data analysis is to extract underlying patterns in time and space from the data. However, the innate characteristics of spatiotemporal data, including spatial and temporal correlations and its significant size, pose challenges for its analysis.

Tensor, a multidimensional array structure, can serve as a container for spatiotemporal data of high dimensions. Tensor decompositions can be used to extract latent patterns in time and space from the data. The canonical polyadic (CP) decomposition is one widely used tensor decomposition model. CP decomposition expresses a tensor as a linear combination of rank-one tensors. One advantage of tensor rank decomposition over matrix decomposition is the essential uniqueness of the decomposition for a high-order tensor under mild conditions [1]. For a high-order tensor, the combination of rank-one tensors that sums up to it is unique up to column scaling and permutations under mild conditions.

Fitting a CP model for a tensor can be viewed as a least squares problem. Through iterative process, the optimization algorithm finds the com-

bination of decomposed components that gives the best approximation of the original tensor. However, a classical CP model only provides a general framework for tensor decomposition. For tensors carrying spatiotemporal data, we would like to impose some constraints, such as non-negativity or smoothness on decomposed components, to make the results more interpretable. In this study, we choose to add smoothness constraints on the CP model when exploring spatial and temporal patterns from Uber pick up data. The motivation is that for a certain location, we do not expect the number of pickups to change drastically within 1 hour. And for a certain time, we do not expect the number of pickups changes drastically within two neighboring locations.

In addition, fitting a CP model usually starts with random guesses for factor matrices. For a tensor of significant size, the optimization process (e.g nonlinear conjugate gradient descent) takes a large number of iterations to find the optimal solution. There have been studies that propose using high-order singular value decomposition (HOSVD) as initialization for optimization process to decrease the number of iterations. We find out that HOSVD can capture the sparsity patterns (the locations of nonzero entries) in factor matrices of the final CP model efficiently. For potential future work, we may be able to further accelerate the optimization process by only updating the nonzero locations predicted by HOSVD.

1.1 Tensors

A tensor can be considered as a generalization of the matrix to higher dimensions. It is often considered as a multidimensional array. The order or mode of a tensor (denoted by N) refers to the number of dimensions of the tensor $\mathcal{X}$. A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a N-way tensor. A matrix can be viewed as a 2-way tensor.

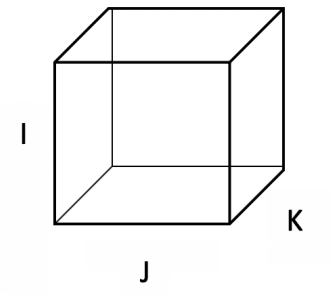

Figure 1.1: A 3-way tensor: $\mathcal{X}$ of size I × J × K

1.1.1 Tensor Inner Product

The inner product between two tensors of the same size $I_1 \times I_2 \times \cdots \times I_N$ is defined to be the sum the the products of corresponding entries:

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \ldots i_N} y_{i_1 i_2 \ldots i_N}$$

1.1.2 Tensor Norm

The norm of a tensor is analogous to the Frobenius norm of matrix. It is the square root of the sum of the squares of all entries. It can also be expressed

as the square root of the inner product of $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with itself:

$$\| \mathcal{X} \| = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle} = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \ldots i_N}^2}$$

1.1.3 Rank-One Tensors

An N-way tensor is called rank-one if it can be expressed as the outer product of N vectors.

A rank-one 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ can be expressed as the outer product of three vectors $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^J$, and $\boldsymbol{c} \in \mathbb{R}^k$:

$$\mathcal{X} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c}$$

Each element of $\mathcal{X}$, x_{ijk}, can be written as the product of corresponding elements in three vectors:

$$x_{ijk} = a_i b_j c_k$$

where $1 \leq \mathrm{i} \leq I$, $1 \leq \mathrm{j} \leq J$, $1 \leq \mathrm{k} \leq K$.

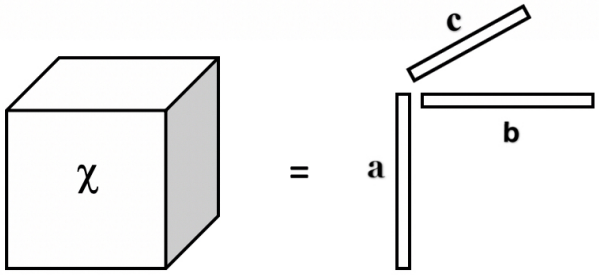

Figure 1.2: A rank-one 3-way tensor as the outer product of 3 vectors

1.2 Notations

We will keep the terminology consistent with terminology used in previous publications in the area of tensor decomposition. The tensor notation here is similar to that proposed by Kiers [3] and Kolda [1].

Symbol	Definition
$\bigotimes$	tensor-matrix product
$*$	elementwise product
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri Rao product
$\boldsymbol{\mathcal{X}}$	tensor
$\boldsymbol{A}$	matrix
$\boldsymbol{A}^{(k)}$	k-th mode factor matrix
$\boldsymbol{\mathcal{X}}_k$	mode-k matricization of $\boldsymbol{\mathcal{X}}$
$\boldsymbol{a}$	vector
a	scalar

1.2.1 Matricization

Matricization or unfolding, refers to the process of reordering a multiway array into a matrix. A general matricization can be found in Kolda [4]. The mode-n unfolding of a $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, denoted by $\mathcal{X}_{(n)}$. It maps the element of tensor at the location $(i_1, i_2, \ldots, i_N)$ to the matrix element at the location $(i_n$,j) [1].

A slice of a tensor is a two-dimensional section of it [1]. Frontal slices of a tensor is shown in Figure 1.3.

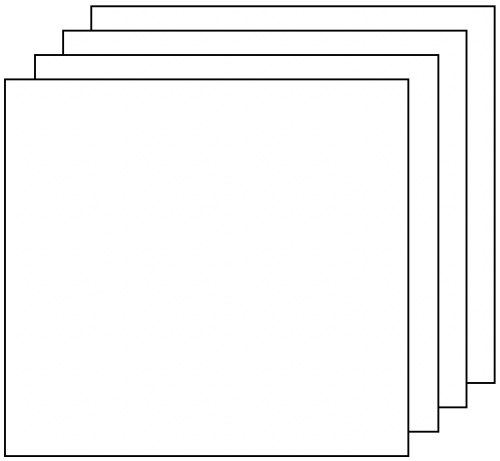

Figure 1.3: Frontal slices

We can use this notation to illustrate a tensor unfolding. For example, we have a tensor $\mathcal{X}$ of size $3 \times 4 \times 2$ and its two frontal slices are:

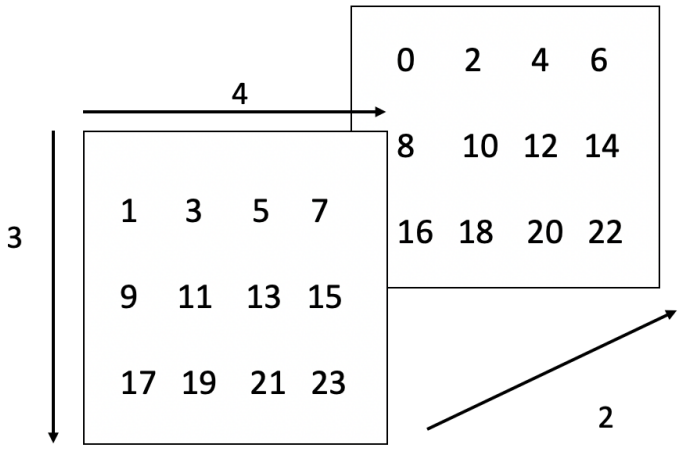

Figure 1.4: Frontal slices of the tensor $\mathcal{X}$ of size $3 \times 4 \times 2$

Then the mode-0 unfolding, denoted by $\mathcal{X}_0$ is:

$$\mathcal{X}_0 = \begin{pmatrix} 1 & 3 & 5 & 7 & 0 & 2 & 4 & 6 \\ 9 & 11 & 13 & 15 & 8 & 10 & 12 & 14 \\ 17 & 19 & 21 & 23 & 16 & 18 & 20 & 22 \end{pmatrix}$$

The mode-1 unfolding, denoted by $\mathcal{X}_1$, is:

$$\mathcal{X}_1 = \begin{pmatrix} 1 & 9 & 17 & 0 & 8 & 16 \\ 3 & 11 & 19 & 2 & 10 & 18 \\ 5 & 13 & 21 & 4 & 12 & 20 \\ 7 & 15 & 23 & 6 & 14 & 22 \end{pmatrix}$$

The mode-2 unfolding, denoted by $\mathcal{X}_2$, is:

$$\mathcal{X}_2 = \begin{pmatrix} 1 & 9 & 17 & 3 & 11 & 19 & 5 & 13 & 21 & 7 & 15 & 23 \\ 0 & 8 & 16 & 2 & 10 & 18 & 4 & 12 & 20 & 6 & 14 & 22 \end{pmatrix}$$

1.2.2 The n-mode Product

The n-mode product refers to multiplying a tensor by a matrix in mode n. The n-mode product of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and a matrix $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$, denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}$, is of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ and can be expressed elementwise as:

$$(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A})_{i_1 \ldots i_{n-1} i_J i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_N} x_{i_1 i_2 \ldots i_n} a_{j i_n}$$

1.2.3 Kronecker Product

The Kronecker product between matrix $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{K \times L}$, denoted by $\boldsymbol{A} \otimes \boldsymbol{B}$ is a matrix $\in \mathbb{R}^{(IK) \times (JL)}$ and is defined by

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots & a_{1J}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \cdots & a_{2J}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\boldsymbol{B} & a_{I2}\boldsymbol{B} & \cdots & a_{IJ}\boldsymbol{B} \end{bmatrix}$$

1.2.4 Khatri-Rao Product

The Khatri-Rao product between $\boldsymbol{A} \in \mathbb{R}^{I \times K}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times K}$, denoted by $\boldsymbol{A} \odot \boldsymbol{B}$, is a matrix $\in \mathbb{R}^{(IJ) \times K}$ and is defined by:

$$\boldsymbol{A} \odot \boldsymbol{B} = \begin{bmatrix} \boldsymbol{a}_1 \otimes \boldsymbol{b}_1 & \boldsymbol{a}_2 \otimes \boldsymbol{b}_2 & \cdots \boldsymbol{a}_K \otimes \boldsymbol{b}_K \end{bmatrix}$$

Chapter 2

CP Decomposition

The canonical polyadic (CP) decomposition is one of the most widely used tensor decomposition algorithms. Its development history can be found in Kolda and Bader [1]. CP expresses a N-way tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \cdots \times I_N$ as a sum of R rank-one tensors.

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \cdots \circ \boldsymbol{a}_r^{(N)} \qquad where \quad \lambda_r \in \mathbb{R}, \boldsymbol{a}_r^{(1)} \in \mathbb{R}^{I_1}, \ldots, \boldsymbol{a}_r^{(N)} \in \mathbb{R}^{I_N}, R \in \mathbb{Z}^+.$$

For a three-mode tensor $\boldsymbol{\mathcal{X}}$, it can be expressed as

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \boldsymbol{a}_r^{(3)}$$

A factor matrix corresponding to one mode of a tensor is defined as the combination of vectors in the rank-one components. For a 3-way tensor, there are three factor matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}$ corresponding to three modes of $\boldsymbol{\mathcal{X}}$, where $\boldsymbol{A}.^{(k)} = [\boldsymbol{a}_1^{(k)} \quad \boldsymbol{a}_2^{(k)} \quad \ldots \quad \boldsymbol{a}_R^{(k)}]$, for k = 1, 2, 3.

We can write the CP model in terms of factor matrices as

$$\boldsymbol{\mathcal{X}} = [\![\lambda; \boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}]\!] \qquad \boldsymbol{\lambda} \in \mathbb{R}^R.$$

For simplification, we often consider multipliers λ_r's are absorbed into the factors and write the CP model as

$$\mathcal{X} = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \boldsymbol{a}_r^{(3)} = [\![\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}]\!]$$

2.1 Tensor Rank

Tensor rank is defined as the smallest number of rank-one tensors that sum up to $\mathcal{X}$. Determining tensor rank can be challenging. As pointed out by Kolda and Bader [1], there is no algorithm that directly determines the rank of any given tensor. In practice, we determine the rank of a tensor by generating CP models with different choices of R. We choose the best R based on the fit and the core consistency of the CP model. In general, as the rank increases, the fit of the CP model increases. However, if we choose an R that is greater than the actual rank, which gives rise to the problem of overfactoring [2].

2.2 Fitting a CP Model

Since determining the rank of a given N-way tensor $\mathcal{Z}$ of size $I_1 \times I_2 \times \cdots \times I_N$ is difficult, we fit a N-way tensor with CP model with a predetermined rank R as:

$$\mathcal{Z} \approx \sum_{r=1}^{R} \boldsymbol{a_r}^{(1)} \circ \boldsymbol{a_r}^{(2)} \cdots \circ \boldsymbol{a_r}^{(\boldsymbol{N})} = [\![\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)}]\!]$$

In this equation, the factor matrix $\boldsymbol{A}^{(k)}$ is of size $I_k \times R$, for k = 1, 2, ..., N.

For a predetermined R, we want to find the combination of factor matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$ that gives the best approximation of the tensor $\mathcal{Z}$. It automatically becomes a least squares problem. The objective function we want to minimize can be expressed as a function of factor matrices:

$$f(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}) = \tfrac{1}{2} \parallel \mathcal{Z} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] \parallel^2$$

2.3 CP_OPT

Acar, Dunlavy and Kolda [2] presented a gradient-based optimization approach, CP_OPT, to solve the least squares problem. This algorithm views fitting a CP model as an optimization problem and explicitly calculates the gradient. It utilizes nonlinear conjugate gradient or L-BFGS as the optimization method.

2.3.1 Objective Function

The objective function can be broken down into three parts, which will help us easily derive the gradient. The complete derivation and proof and be found in Acar, Dunlavy and Kolda [2].

$$f(\mathbf{A}^{(1)},\mathbf{A}^{(2)},\ldots,\mathbf{A}^{N}) = \frac{1}{2}\parallel \mathcal{Z} \parallel^2 -\langle \mathcal{Z}, [\![\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)},\ldots,\boldsymbol{A}^{(N)}]\!]\rangle + \frac{1}{2}\parallel [\![\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)},\ldots,\boldsymbol{A}^{(N)}]\!] \parallel^2$$

2.3.2 CP Gradient

The partial derivatives of the objective function f with respective to factor matrices are given by

$$\frac{\partial f}{\partial \mathbf{A}^{(n)}} = -\mathbf{Z}_{(n)}\mathbf{A}^{(-n)} + \mathbf{A}^{(n)}\Gamma^{(n)},$$

where $\mathbf{A}^{(-n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)}$ and $\boldsymbol{\Gamma}^{(n)} = \gamma^{(1)} * \cdots * \gamma^{(n-1)} * \gamma^{(n+1)} * \cdots * \gamma^{(N)}$ with $\gamma^{(n)} = \left(\mathbf{A}^{(n)}\right)^T \mathbf{A}^{(n)}$.

Now the function f is expressed as a function of factor matrices and the partial derivative is calculated with respect to each column of a factor matrix. We can consider the objective function f as a function of a stacked

vector $\mathbf{x}$ that comprises factor matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{a}_1^{(1)} \\ \vdots \\ \mathbf{a}_R^{(1)} \\ \mathbf{a}_1^{(2)} \\ \vdots \\ \mathbf{a}_R^{(2)} \\ \vdots \\ \mathbf{a}_1^{(N)} \\ \vdots \\ \mathbf{a}_R^{(N)} \end{bmatrix}$$

We can also put the partial derivatives with respect to $\mathbf{A}^{(n)}$'s into a stacked vector.

For the following sections of this chapter, we will narrow our discussions to 3-way tensors. For a 3-way tensor $\mathcal{Z}$, the objective function of fitting a CP model is written as

$$f(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}) = \frac{1}{2} \parallel \mathcal{Z} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!] \parallel^2$$

We use the implementation of CP_OPT in Tensor Toolbox for Matlab developed by the Sandia National Laboratories as baseline.

2.4 CP_OPT with Smoothness Penalty

The general CP_OPT framework does not guarantee smoothness in the decomposed factor matrices. For spatiotemporal data analysis through CP decomposition, we expect the change in magnitudes between neighboring entries in the same column to be smooth. Therefore, we add a smoothness penalty term to the objective function.

2.4.1 Objective Function with Smoothness Penalty

For a N-way tensor $\mathcal{Z}$, we define a N-way tensor $\mathbf{L}$. Each factor matrix $\mathbf{L}^{(i)}$ is a square matrix of size $I_i \times I_i$, where I_i is the row dimension of $\mathbf{A}^{(i)}$. $\mathbf{L}^{(i)}$

has 1's along the diagonal and -1's along the subdiagonal.

$\mathbf{L}^{(i)} =$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & 1 \end{bmatrix}$$

The square of the Frobenius norm of $\mathbf{L}^{(k)}\mathbf{A}^{(k)}$ is equal to the sum of squares of the differences between neighboring cells in the same column for factor matrix $\mathbf{A}^{(k)}$, except for the first row.

$$\| \mathbf{L}^{(k)}\mathbf{A}^{(k)} \|_F^2 = \sum_{j=1}^{R} \sum_{r=2}^{I_k - 1} \left(a_{rj}^{(k)} - a_{(r+1)j}^{(k)} \right)^2$$

Then the objective function with smoothness constraints on output components becomes:

$$f(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}) = \frac{1}{2} \| \mathcal{Z} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!] \|^2 + \underbrace{\frac{\lambda}{2} \left(\|\mathbf{L}^{(1)}\mathbf{A}^{(1)}\|_F^2 + \|\mathbf{L}^{(2)}\mathbf{A}^{(2)}\|_F^2 + \|\mathbf{L}^{(3)}\mathbf{A}^{(3)}\|_F^2 \right)}_{f_p}$$

where λ is the strength of the penalty term.

2.4.2 Updated Gradient

The new objective function can be solve using CP_OPT by deriving a new gradient corresponds to the penalty term and adding this gradient to the existing gradient.

$$f_p = \frac{\lambda}{2} \left(\|\mathbf{L}^{(1)}\mathbf{A}^{(1)}\|_F^2 + \|\mathbf{L}^{(2)}\mathbf{A}^{(2)}\|_F^2 + \|\mathbf{L}^{(3)}\mathbf{A}^{(3)}\|_F^2 \right)$$

We take the partial derivative with respect to each factor matrix:

$$\frac{\partial f_p}{\partial \mathbf{A}^{(k)}} = \lambda \left(\mathbf{L}^{(k)} \right)^T \mathbf{L}^{(k)}\mathbf{A}^{(k)} \qquad k = 1, 2, 3$$

2.5 Optimization Method for CP_OPT

We have obtained the objective function f as a function of stacked vector $\mathbf{x}$ that comprises factor matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}$ and its corresponding gradient in the same form. The current implementation of CP_OPT in the Matlab Tensor Toolbox utilizes either first-order optimization method (nonlinear conjugate gradient descent) or second-order optimization method (L-BFGS) to find the optimal solution $\mathbf{x}$ and then transforms the solution into corresponding factor matrices.

Chapter 3

Experiment Outline

3.1 Data Description

We evaluate the modified algorithm on six-month Uber pick up data in New York City. The dataset covers statistics from April 2014 through August 2014, which has 1,426,993 pickups. The original dataset records the number of pickups in a certain location at a certain time. The location is expressed in latitude and longitude, whose values are rounded to three decimal places. The time is expressed in the date and the hour of the day. Tensor values (the counts) are integers.

3.2 Set up

3.2.1 Dataset Resize

The original data contains date, hour, latitude and longitude information for each of 1,426,993 pickups. For this study, we conduct experiments on 3-way tensor and choose to only keep hour, latitude and longitude information for each pickup. Hour has 24 different values that represent 24 hours in a day. Latitude has 1140 different values and longitude has 1717 different values. The gap between two different values is 0.001 for both latitude and longitude. To shrink the size of the tensor while keeping all the information, we reset the gap to be 0.01 and group some values together .

The tensor we operate on is of size $24 \times 176 \times 237$. The resulting tensor is sparse with only 4% of elements are nonzero.

3.2.2 Determine Rank: Fit and Core Consistency Diagnostic

When fitting a CP model for a tensor $\boldsymbol{\mathcal{X}}$, we choose a predetermined R to find the combination of N factor matrices that gives the best approximation to the original tensor:

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a_r}^{(1)} \circ \boldsymbol{a_r}^{(2)} \cdots \circ \boldsymbol{a_r}^{(N)} = [\![\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)}]\!]$$

We hope that the chosen R is close to the exact rank of the original tensor. Choosing a rank much smaller than the actual rank leads to poor fit. Choosing a rank greater than the actual rank leads to the problem of overfactoring. However, as mentioned in Chapter 1, there is no good algorithm that directly gives the exact rank of a given tensor.

CP decomposition is a special case of Tucker Decomposition where the core tensor is superdiagonal. Given the best CP model approximation to the original tensor, we can calculate the corresponding core tensor through the factor matrices. If we select an R greater than the actual rank for a CP model, the CORCONDIA, which measures how superdiagonal the calculated core tensor is, is large. Thus, as the R increases, we have a higher fit but the core consistency drops.

For this study, we first select a range of R that is likely to contain the exact rank of the tensor. Then we calculate the model fit and the core consistency score obtained from the Core Consistency Diagnostic (CORCONDIA) associated with different choices of R in this range. We narrow down the range of R by selecting R's that give better balances between the fit and the core consistency. For 3-way tensor,

$$Fit = (1 - \frac{\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{K}} \|^2}{\| \boldsymbol{\mathcal{X}} \|^2}) \times 100$$

where $\boldsymbol{\mathcal{K}} = [\![\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}]\!]$.

The Core Consistency Diagnostic (CORCONDIA) is calculated as

$$CORCONDIA = (1 - \frac{\| \mathcal{I} - \mathcal{G} \|^2}{\| \mathcal{I} \|^2}) \times 100$$

for a 3-way tensor $\mathcal{X}$, $\mathcal{G} = \mathcal{X} \times_1 (A^{(1)})^+ \times_2 (A^{(2)})^+ \times_3 (A^{(3)})^+$, where $(A^{(1)})^+, (A^{(2)})^+, (A^{(3)})^+$ are the Moore-Penrose inverses of factor matrices $A^{(1)}, A^{(2)}, A^{(3)}$ obtained from the CP model [7]. If we use R for the CP model, then $\mathcal{I}$ is a super-diagonal tensor of size $R \times R \times R$ with 1's along the superdiagonal.

For the Uber pick up tensor, we find out that the fit is 0.88 when R equals 2 and exceeds 0.95 when R increases to 5. We first set the range of R to be from 2 to 8. As shown in table 3.1, we generate a CP_OPT model for R = 1, 2, ..., 8 and calculate the fit and calculate the core consistency score of the CP model obtained from CP_OPT using HOSVD as initialization.

R	Fit	CORCONDIA
2	88.263	100.000
3	92.632	86.945
4	94.825	-59.764
5	96.182	-32.279
6	97.208	-509.308
7	97.866	-504.906
8	98.444	-19457.603

Table 3.1: The model fit and the CORCONDIA at different choices of R

We observe that the core consistency score changes drastically when R increases from 5 to 6. Therefore, we choose 5 as the maximum choice of R to avoid overfactoring.

3.3 Select Penalty Strength for the Smoothness Penalty

We generate CP_OPT model with smoothness constraint for a given R. In this study, we calculate the the smoothness score using the value of the

penalty term $\| \mathbf{L}^{(1)}\mathbf{A}^{(1)}\|_F^2 + \|\mathbf{L}^{(2)}\mathbf{A}^{(2)}\|_F^2 + \|\mathbf{L}^{(3)}\mathbf{A}^{(3)}\|_F^2$ as it measures the sum of squares of the column-wise neighboring cells for all factor matrices. A smaller smoothness score indicates a better smoothness in the decomposed factor matrices. For simplification, instead of finding the best λ for each choice of R, we find the the best λ associated with the maximum choice of R (e.g R = 5) and fix this λ for other choices of R.

Figure 3.1: The Model Fit against the Smoothness Score at R = 5. Data label indicates the value of penalty strength λ

As shown in Figure 3.1, there is a trade-off between the model fit and the smoothness score. We try the penalty strength from 0 to 10,000. We observe that the CP_OPT algorithm with smoothness constraint will crash when λ is greater than 100,000. This is because for a very large λ, the penalty term dominates the objective function and fails the optimization process. As λ increases from 0 to 100,000, the smoothness score drops

from 9.4 to 8.3. A reasonable choice of λ is between 20,000 and 60,000. We fix the λ to be 50,000 for later experiments.

3.4 Visualization of Smoothness Improvement

We measure the differences in values between two neighboring cells in every column of every factor matrix. For classical CP_OPT model and CP_OPT model with smoothness constraints, respectively, we plot these differences for each column of each factor matrix with different choices of R. In Figure 3.2 to 3.5, the x-axis refers to the index and the y-axis refers to the absolute value of the difference between neighboring cells. The red dashed line corresponds to the classical CP_OPT model and the black line corresponds to the CP_OPT model with smoothness constraints. For same index, the smaller the average value of the difference between neighbouring cells, the better the smoothness. One thing to clarify is that the improvement of smoothness is not necessarily applicable in every column in every factor matrix. Model fit is still a more dominant term in the objective function compared to the penalty term.

For example, at R = 4, we look at 4 columns in factor matrix corresponds to Mode-2 of the tensor. We observe significant improvements in smoothness for the 2^{nd}, as shown in Figure 3.2, and the 3^{rd} column, as shown in Figure 3,3. There is no big change for the 1^{st} column, as shown in Figure 3.6, and the 4^{th} column, as shown in Figure 3.7.

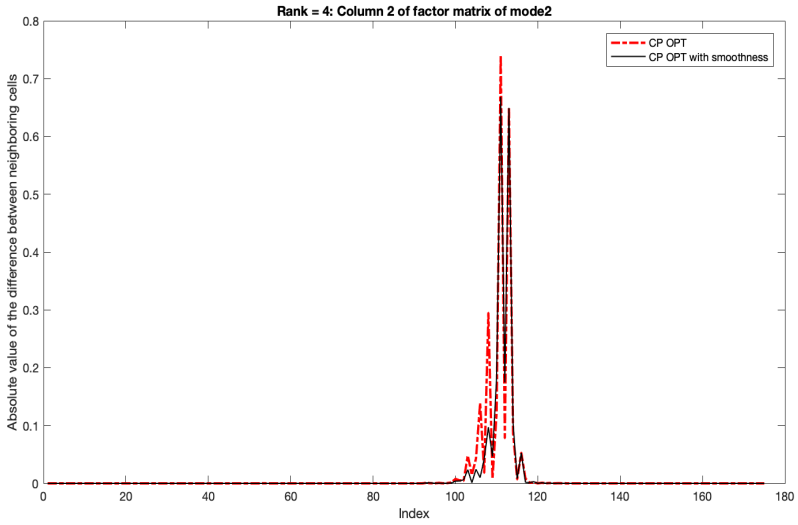

Figure 3.2: R = 4: 2nd column of factor matrix of mode 2 (latitude)

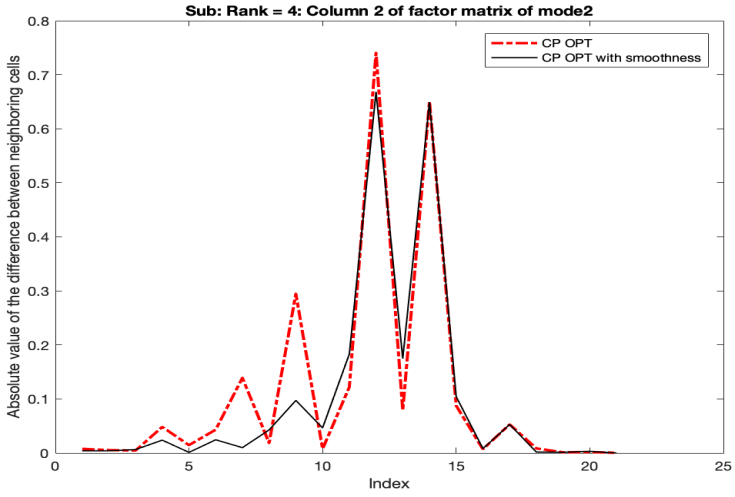

Figure 3.3: R = 4: Zoom in peak range of the 2nd column of factor matrix of mode 2 (latitude)

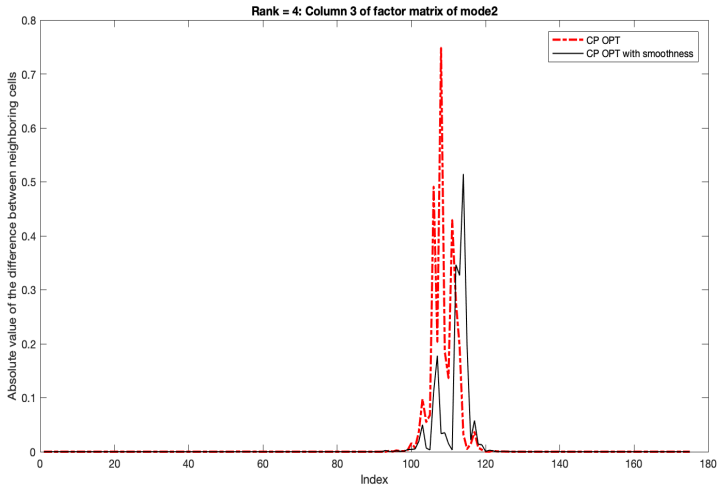

Figure 3.4: R = 4: 3rd column of factor matrix of mode 2 (latitude))

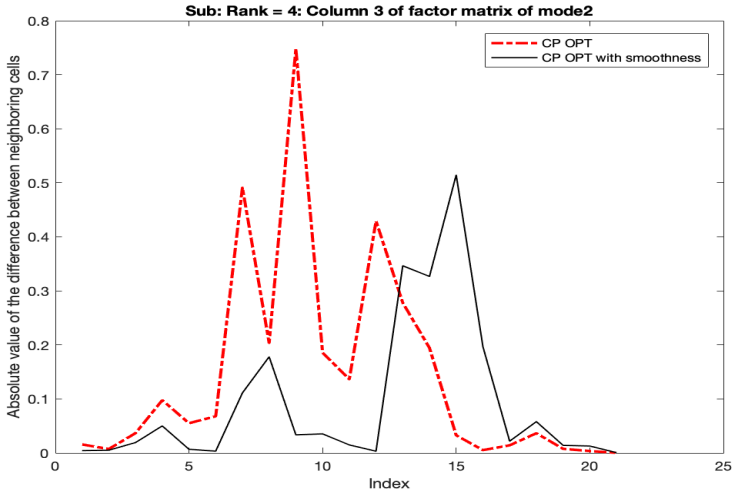

Figure 3.5: R = 4: Zoom in peak range of the 3rd column of factor matrix of mode 2 (latitude)

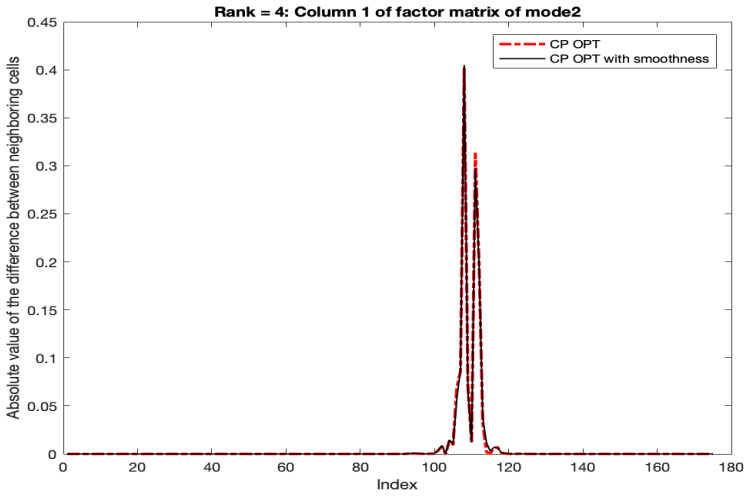

Figure 3.6: R = 4: 1st column of factor matrix of mode 2 (latitude)

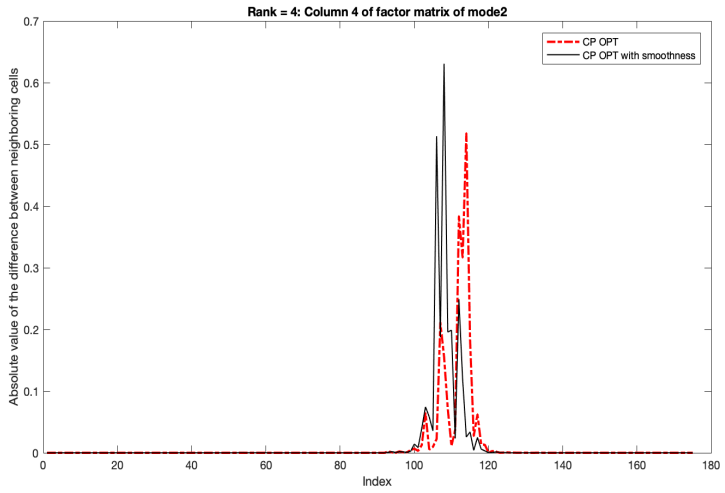

Figure 3.7: R = 4: 4th column of factor matrix of mode 2 (latitude)

3.5 Running Performance

We compare the running performance of the CP_OPT with smoothness constraints (λ = 50,000) to the classical CP_OPT under different choices of R and the results are shown in the following table. For both models, we use random initialization for the optimization process. For each R, we do 100 random initializations on both the CP_OPT and the CP_OPT with smoothness constrains. For each model with a different random initialization, we measure the model fit and the smoothness score. We measure the running performance based on the number of iterations required by the optimization method (L-BFGS). In the following table, **Classical** refers to the classical CP_OPT model, and **Smooth** refers to the CP_OPT model with smoothness constraints.

	Fit (%)		Smoothness		Average Iterations ± SD		Median Iterations	
R	Classical	Smooth	Classical	Smooth	Classical	Smooth	Classical	Smooth
2	88.259	87.743	1.974	1.945	151.6 ± 229.4	244.3 ± 47.7	95	243
3	92.632	91.907	3.328	3.328	359.5 ± 409.3	328.8 ± 64.3	234	309
4	94.822	93.761	5.970	5.493	1392.4 ± 1483.7	457.6 ± 79.8	739	447
5	96.103	94.797	9.234	8.286	1905.1 ± 1942.6	569.7 ± 100.8	1027	541.5

From the Table 3.5, we find out there is indeed an overall improvement in smoothness by adding the smoothness penalty, but with some compensation for the model fit. With random initialization for the optimization process, we observe that the standard deviation of the number of iterations can be huge.

We also observe that given random initializations, adding the smoothness penalty seems to be able to stabilize the number of iterations required by the optimization process and leads to smaller standard deviations.

Chapter 4

Exploring Sparsity

In the past, tensor decomposition problems focus on dense data, such as applications from Chemometrics, where most of the values of the tensor are nonzero. However, for spatiotemoporal data, the corresponding tensor is usually quite sparse. Existing tensor decomposition algorithms only provide a framework for general tensors, without capturing the innate structures of sparse tensors.

Furthermore, the optimization process of fitting a CP model usually starts with random initial guesses for factor matrices. However, for a sparse tensor, some of the factor matrices obtained from CP decomposition are usually quite sparse. Therefore, the optimization process that starts with random factor matrices that are fully dense can be slow. What's more, computing CP models with constraints, such as on smoothness or non-negativity, is much more expensive than the unconstrained CP models. Constraints generate a larger gradient and the optimization method (e.g L-BFGS) may take longer time to find the optimal CP model.

There have been studies that use high-order singular value decomposition (HOSVD) as initialization for optimization process to start at a location closer to the local minimum and thus decrease the number of iterations. We propose that using high-order Singular Value Decomposition (HOSVD) as initialization can help us learn the sparsity patterns in factor matrices obtained from CP Decomposition efficiently. We can accelerate the iterations

process further by truncating ”trivial” entries in HOSVD before putting it as the initial guess.

4.1 Tucker Decomposition and HOSVD

Tucker decomposition expresses a tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \cdots \times I_N$ into a core tensor $\mathcal{G}$ of size $R_1 \times R_2 \cdots \times R_N$ multiplied by a factor matrix corresponding each mode. CP decomposition is a special case of Tucker decomposition where the core tensor has a cubic structure and is superdiagonal.

We call $\mathcal{X}$ a rank-$(R_1 \times R_2 \cdots \times R_N)$ tensor. The n-rank is defined to be the dimension of vector space spanned by the fibers of n-th mode [1].

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{A}^{(1)} \times_2 \cdots \times_N \boldsymbol{A}^{(N)} = [\![\boldsymbol{\mathcal{G}}; \boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)}]\!]$$

Consider a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, Tucker decomposition of $\mathcal{X}$ is

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \boldsymbol{a_p} \circ \boldsymbol{b_q} \circ \boldsymbol{c_r} = [\![\boldsymbol{\mathcal{G}}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$$

where $\boldsymbol{A} \in \mathbb{R}^{I \times P}$, $\boldsymbol{B} \in \mathbb{R}^{J \times Q}$, and $\boldsymbol{C} \in \mathbb{R}^{K \times R}$.

4.2 HOSVD

High-order Singular Value Decomposition (HOSVD) is a special Tucker Decomposition. It generate one factor matrix whose columns are orthogonal to each other for each mode of the tensor.

One classical strategy to compute a HOSVD for a N-way tensor takes the following three steps [6]:
For $k = 1, 2, \ldots, N$

1. Construct the mode-k unfolding $\boldsymbol{\mathcal{X}}_k$
2. Compute the singular value decomposition $\boldsymbol{\mathcal{X}}_k = \boldsymbol{U}_k \Sigma_k \boldsymbol{V}_k^T$ and store the left singular vectors $\boldsymbol{U}_k$ as factor matrix $\boldsymbol{A}^k$
3. Compute the core tensor $\boldsymbol{\mathcal{G}}$ via multilinear multiplication $\boldsymbol{\mathcal{G}} = (\boldsymbol{U}_1^T, \boldsymbol{U}_2^T, \ldots, \boldsymbol{U}_N^T) \cdot \boldsymbol{\mathcal{X}}$

The other approach (interlaced HOSVD) interlaces the computation of factor matrices with core tensor and works as follows [6]:
Set $\boldsymbol{\mathcal{X}}^{(0)} = \boldsymbol{\mathcal{X}}_0$
For $k = 1, 2, \ldots, N$

1. Construct the mode-k unfolding $\boldsymbol{\mathcal{X}}_k^{(k-1)}$
2. Compute the singular value decomposition $\boldsymbol{\mathcal{X}}_k^{(k-1)} = \boldsymbol{U}_k \Sigma_k \boldsymbol{V}_k^T$ and store the left singular vectors $\boldsymbol{U}_k$ as factor matrix $\boldsymbol{A}^k$
3. Set $\boldsymbol{\mathcal{X}}_k^{(k)} = \boldsymbol{U}_k^T \boldsymbol{\mathcal{X}}_k^{(k-1)} = \Sigma_k \boldsymbol{V}_k^T$

Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}_N^{(N)}$. In theory, the interlaced HOSVD is more efficient than the classical HOSVD as it shrinks the size of the matrix whose SVD is to be find at each iteration.

A truncated HOSVD is computed by replacing the full singular value decomposition with a truncated SVD.

4.3 HOSVD as Initialization

The current implementation of CP_OPT supports using a classical version of HOSVD, which calls the Matlab function `nvecs`, as initialization for the optimization method. It can largely accelerate the optimization process compared to a random initialization. Most importantly, the main cost of computing the HOSVD for a low-rank tensor comes from computing a few (e.g R = 5) dominant eigenvectors of sparse matrices in this application. The cost of computing HOSVD is trivial compared to the cost of the optimization process because both nonlinear conjugate gradient descent and L-BFGS are iterative processes.

We propose that the interlaced HOSVD can give a better performance than the classical HOSVD. We utilize the `HOSVD` function in the Matlab Tensor Toolbox, an implementation of interlaced HOSVD, and make some modifications. It originally uses the `eig` solver to find call eigenvectors for each k-mode unfolding. Since the tensor is a sparse tensor, we choose to use the function `eigs` instead to compute the R dominant eigenvectors of

$(\mathcal{A}_{(k)}^{k-1})^T \mathcal{A}_{(k)}^{k-1}$, insteading of calculating all the eigenvectors and then do the selection.

On both the classical CP_OPT model and the CP_OPT model with smoothness constraints, we compare the performance of using random, the classical HOSVD, and the interlaced HOSVD as initialization for the optimization process.

4.3.1 Comparisons between Different Initializations

random represents the random initialization; **nvecs** represents the classical HOSVD; **HOSVD** represents our implementation of the interlaced HOSVD. We use the median of the number of iterations to measure the performance.

	Number of Iterations (Median)				
R	random	nvecs	HOSVD	Fit	Smoothness
2	95	67	61	88.263	1.973
3	234	114	88	92.632	3.431
4	739	254	277	94.825	5.964
5	1027	332	289	96.182	9.309

Figure 4.1: Performances of Different Initilizations for CP_OPT

	Number of Iterations (Median)				
R	random	nvecs	HOSVD	Fit	Smoothness
2	243	197	197	87.743	1.945
3	309	266	234	91.907	3.328
4	447	364	379	93.868	5.609
5	541.5	508	502	95.028	8.649

Figure 4.2: Performances of Different Initailizations for CP_OPT with Smoothness Constraints

For both the classical CP_OPT model and the CP_OPT model with smoothness constraints, using HOSVD as initialization fixes the number of iterations for the optimization method and the number is much smaller than the one taken by a random initialization. We also observe that the interlaced HOSVD has a slightly better performance than the classical HOSVD. For this experiment, we operate on a relative small tensor and we expect a bigger difference in performances if we operate on a tensor of a bigger size.

4.4 Sparsity of the Experiment Tensor

Our experiment tensor containing Uber pickup data contains only 4% of nonzero elements. We explore 3 factor matrices generated by CP_OPT and we find out that the mode 2 factor matrix corresponding to the latitude and the mode 3 factor corresponding to the longitude contains many trivial entries. We calculate the percentage of entries with magnitudes below $1e^{-5}$ in each factor matrix and the result is shown in the following table.

R	Mode 1	Mode 2	Mode 3
2	0	72.73	55.06
3	0	70.08	55.70
4	0	69.32	55.49
5	0	68.75	56.79

For all choices of R, the mode 1 factor matrix that corresponds to the hour is evenly distributed in terms of the magnitudes of entries and can be considered as a fully dense matrix. For the mode 2 factor matrix and the mode 3 factor matrix, they can be considered as sparse if regarding those trivial entries as zeros.

4.5 Learning Sparsity Pattern through HOSVD

For a 3-way tensor $\mathcal{X}$, we fit an interlaced HOSVD model and set the size of the core tensor to be $R \times R \times R$. We ignore the core tensor and only look at the factor matrices obtained from HOSVD. We compare the factor matrices obtained from HOSVD and the factor matrices generated by CP_OPT model. They have similar sparsity patterns. That is, the locations of nonzero entries in the factor matrices obtained from HOSVD match the locations of nonzero entries in the factor matrices obtained from CP_OPT. As a result, truncated HOSVD can serve as a good starting point for optimization process of CP_OPT for sparse tensors.

4.5.1 Experiments: Sparsity Patterns

On the test tensor containing Uber pickup data, we generate one HOSVD model and one CP_OPT model for R = 1, 2, 3, 4, 5. We plot the magnitudes of entries in a certain column of a certain factor matrix for both the model generated by HOSVD and the model generated by CP_OPT. For example, for R = 3, we compare the sparsity patterns in all 3 columns factor matrices generated by CP_OPT and the ones in factor matrices generated by HOSVD. Mode 2 and mode 3 correspond to latitude and longitude, respectively and the differences in magnitudes between nontrivial and nontrivial entries are significant. As shown in Figure 4.3, Figure 4.4 and Figure 4.5, HOSVD accurately captures the locations of nontrivial entries.

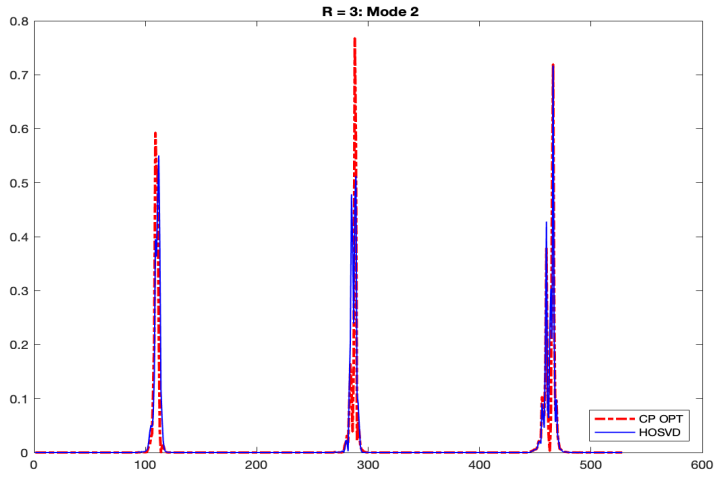

Figure 4.3: R = 3: Sparsity patterns comparison on mode 2 factor matrix between HOSVD and CP_OPT

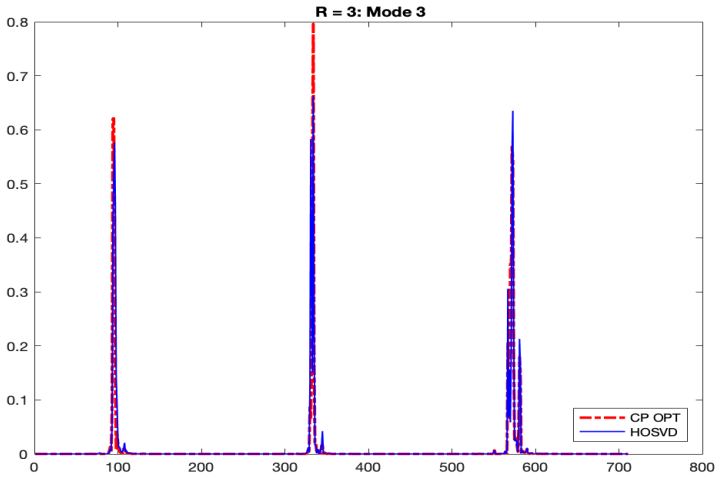

Figure 4.4: R = 3: Sparsity patterns comparison on mode 3 factor matrix between HOSVD and CP_OPT

As shown in Figure 4.5, We zoom in the peak area of the 3^{rd} column

of mode 3 factor matrix. The red dashed line corresponds to CP_OPT and the blue solid line corresponds to HOSVD. There are 3 peaks in the column corresponds to CP_OPT and HOSVD captures all three of them

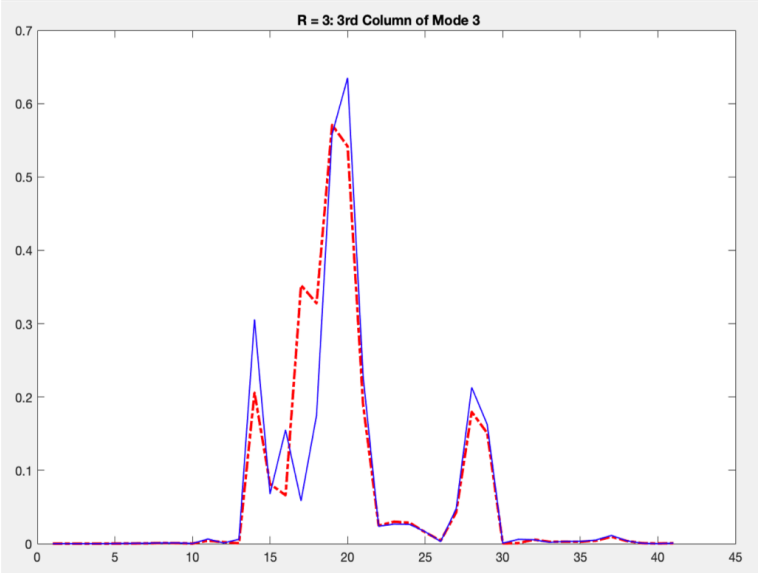

Figure 4.5: R = 3: Zoom in peak area of the 3rd column in mode 3 factor matrix

As shown in our experiments, HOSVD can predict the locations of non-trivial entries in factor matrices generated by CP_OPT.

4.6 HOSVD with Truncation

We propose that a better initial guess can be obtained by truncating small entries whose magnitudes are below some threshold (e.g $1e^{-5}$) in factor matrices obtained from HOSVD. We maintain that small entries that are irrelevant to the final analysis can slow down the convergence of optimization process. Our interpretation is that using HOSVD with truncation not

only makes the starting point for the optimization iterations closer to the local minimum, but also captures the sparsity patterns and incorporates a smoother structure.

4.6.1 Visualization

On our test tensor carrying Uber pickup data, we compare the sparsity patterns of factor matrices obtained by CP_OPT model to those obtained by the truncated HOSVD with a threshold of $1e^{-5}$. In CP_OPT model, the second and the third factor that correspond to the latitude and the longitude, respectively, are both sparse matrices. However, the first factor matrix corresponds to the hour of each pickup is a dense matrix. A straightforward interpretation is that there are pickups in every hour of the day. The threshold does not truncate the first factor matrix of HOSVD and therefore we only look at the second and the third factor matrix. We find that the truncated HOSVD captures the peak range of every column. The following two figures are the example of sparsity pattern comparisons between HOSVD and CP_OPT at R = 3. For both figures, the dashed blue line corresponds to CP_OPT and solid red line corresponds to HOSVD with truncation of $1e^{-5}$.

4.7 Experiments

HOSVD provides a starting point much closer to the local minimum compared to random initialization. For both the traditional HOSVD and the truncated HOSVD, the number of iterations for the classical CP_OPT model and for the CP_OPT model with smoothness constraints is fixed for a given R. For both the classical CP_OPT and CP_OPT with smoothness constraints, we use truncated interlaced HOSVD with different thresholds as initialization and measure the number of iterations required by the optimization process. We try the threshold from 0 (no truncation) to $1e^{-3}$.

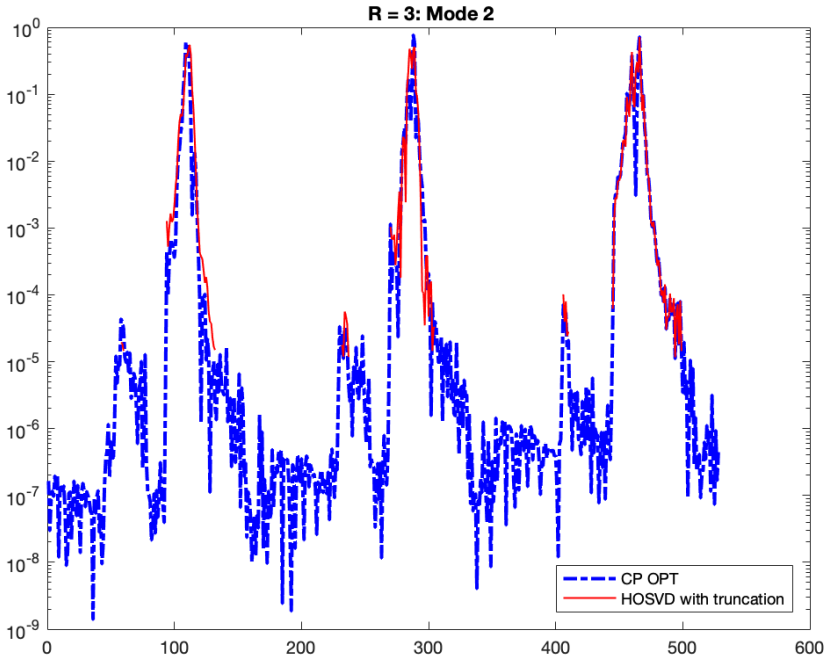

Figure 4.6: R = 3: Sparsity pattern comparisons in Mode 2 factor matrix

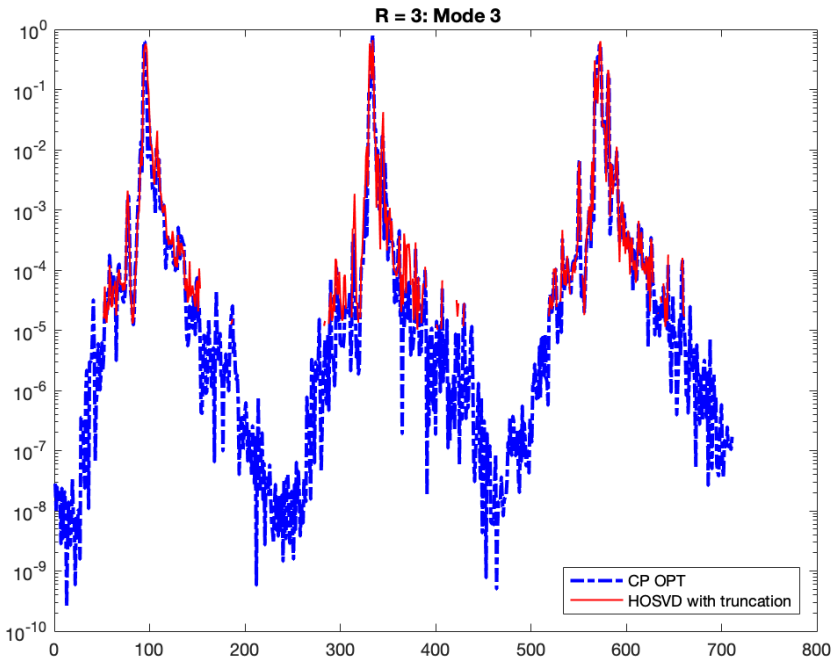

Figure 4.7: R = 3: Sparsity pattern comparisons in Mode 3 factor matrix

4.7.1 Truncated HOSVD as Initialization for CP_OPT

	Threshold					
Rank	0	1e-7	1e-6	1e-5	1e-4	1e-3
2	61	61	61	62	57	63
3	88	88	88	88	96	90
4	277	277	275	256	247	270
5	289	289	266	212	224	271

Figure 4.8: Number of iterations required by the CP_OPT with truncated HOSVD as initialization

We observe that if the threshold is too small (e.g $1e^{-7}$), truncation on HOSVD has no improvement in terms of the number of iterations required by the optimization process. As we gradually increase the threshold and get rid of more trivial entries in factor matrices generated by HOSVD as initialization, we observe an overall decrease in the number of iterations for different choices of R. As the threshold reaches $1e^{-3}$, the performance of truncated HOSVD drops.

4.7.2 Truncated HOSVD as Initialization for CP_OPT with Smoothness Constraints

	Threshold					
Rank	0	1e-7	1e-6	1e-5	1e-4	1e-3
2	165	220	182	218	216	221
3	234	237	237	238	234	252
4	379	380	351	312	327	386
5	502	424	434	467	482	499

Figure 4.9: Number of iterations required by the CP_OPT with smoothness constraints with truncated HOSVD as initialization

For CP_OPT with smoothness constraints, the truncated HOSVD can decrease the number of iterations required by the optimization process for a certain range of choices of thresholds. For example, compared to no truncation, removing entries with magnitudes below $1e^{-6}$ can significantly improve the number of iterations for R = 4 and R = 5.

Chapter 5

Conclusions

In this study, we customize the classical CP_OPT model for sparse tensors containing spatiotemoporal data mainly in 2 aspects:

1. Adding smoothness constrains on factor matrices by using Laplacian matrices;

2. Replacing the random initialization with the interlaced HOSVD to accelerate the optimization process; exploring the sparsity patterns in the decomposed components of CP_OPT using the interlaced HOSVD whose trivial entries are removed as initialization to further accelerate the optimization process by taking the advantage of sparsity

5.1 Smoothness Constraints

As introduced in Chapter 2, we construct the smoothness penalty term using Laplacian matrices to control the change in magnitudes between two neighboring entries in the same column. Introducing this penalty term indeed improves the overall smoothness in factor matrices. However, achieving a better smoothness will decrease the model fit to some extent. There is a trade-off between the model fit and the smoothness when choosing the penalty strength λ. In addition, the penalty term leads to a greater gradient and therefore L-BFGS takes a larger number of iterations to find the optimal in general.

5.2 Initialization Using HOSVD

Using the default random initialization for the optimization process leads to slow convergences and instability in the number of iterations required by the optimization method (e.g L-BFGS).

The current implementation supports using the classical HOSVD as initialization. We have proved that using the interlaced HOSVD as initialization has better performances compared to using the classical HOSVD. In addition, we find out HOSVD can predict the sparsity patterns in factor matrices generated by CP_OPT model. We observe the presence of a considerable amount of trivial entries (e.g entries with magnitudes below 1e−5) in factor matrices generated by HOSVD. Therefore, to avoid unnecessary computations for the optimization method (e.g L-BFGS), we truncate those entries with magnitudes below a self-determined threshold and use a truncated HOSVD as initialization for a CP_OPT model with or without smoothness constraints. Our experiments have shown that HOSVD with truncation can further accelerate the optimization process.

5.3 Future Work

Since the CP_OPT algorithm explicitly calculates the gradient of the objective function and expresses the objective function and the gradient as functions of a stacked vector, we can choose any optimization method that requires only these two pieces of information, not only the nonlinear conjugate gradient descent method and L-BFGS method, such as the fast proximal gradient method (FISTA).

We have shown that HOSVD can identify the locations of nontrivial entries in factor matrices generated by CP_OPT. For spatiotemporal data analysis, we actually care more about those nontrivial entries. In addition, moving trivial entries (e.g with magnitudes below $1e^{-5}$ in factor matrices generated by CP_OPT does not influence the model fit. Therefore, if the optimization method only updates the locations identified as the locations of nontrivial entries, we might be able to reduce the computational cost and accelerate the optimization process even more.

Bibliography

[1] T. Kolda, and B. Bader, *Tensor Decompositions and Applications*. Society for Industrial and Applied Mathematics, 2007, pp. 8–37.

[2] E. Acara, D.Dunlavyb, and T. Kolda, *A scalable optimization approach for fitting canonical tensor decompositions*. Journal of Chemometrics 25(2):67–86, February 2011.

[3] H. Kiers,*Towards a standardized notation and terminology in multiway analysis*. J. Chemometr., 14 (2000), pp. 105–122

[4] T. Kolda, and Tamara Gibson, *Multilinear operators for higher-order decompositions*. United States: N. p., 2006. Web. doi:10.2172/923081.

[5] A. Afshar,J. C. Ho,B.Dilkina,I. Perros,E.B.Khalil, L. Xiong, and V. Sunderam, *Cp-ortho: An orthogonal tensor factor- ization framework for spatio-temporal data*. In Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic In- formation Systems, SIGSPATIAL '17, pages 67:1–67:4, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5490-5. doi: 10.1145/3139958.3140047. URL http://doi.acm.org/10.1145/3139958.3140047.

[6] Higher-order singular value decomposition. (n.d.). In Wikipedia. Retrieved February, 2019, from https://en.wikipedia.org/wiki/Higher-order_singular_value_decomposition.

[7] R. Bro, H. Kiers,*A new efficient method for determining the number of components in PARAFAC models*, J. Chemometrics 2003; 17: 274±286.

[8] M. Bahadori, Q. Yu, and Y. Liu, *Fast Multivariate Spatio-temporal Analysis via Low Rank Tensor Learning*, Neural Information Processing Systems Conference 2014.