Introduction to Tensor Regression: An Example of Simple Least Square Regression

Zixiang Xu
zxu27@gmu.edu

Ben Seiyon Lee
slee287@gmu.edu

Motivation & Research Goal

Tensors, also known as multidimensional arrays, are generalizations of vectors and matrices to higher dimensions. In recent years, tensor data are fast emerging in a wide variety of scientific and business applications, including recommendation systems, speech or facial recognition, networks analysis, knowledge graphs and relational learning. Tensor data analysis is thus gaining increasing attention in statistics and machine learning communities.[1]

For the purpose of studying tensor, the article *Tensor in Modern Statistical Learning*[1] was published recently, where the tensor regression part of the paper took my interest. After googling and looking at the *Tensor Computation for Data Analysis*[2] book, I decided to take this simple LS tensor regression as my final project for STAT-676. And the main purpose of this project is to find out the performance of simple least square tensor regression under the basic settings.

Background Preparation

We start with several fundamental definitions related to tensors.

The *order* of a tensor, also referred to as the mode, is the dimension of the array. A first-order tensor is a vector, a second-order tensor is a matrix, and tensors of order three and higher are referred to as high-order tensors, see Figure 1.

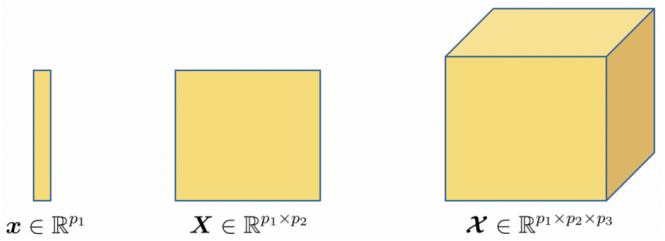

Figure 1: An example of first, second and third–order tensors

The *fiber* of a tensor is defined by fixing all indices but one. For example, given a third-order tensor $\mathcal{X} \in \mathbf{R}^{p_1 \times p_2 \times p_3}$, its mode-1, 2 and 3 fibers are denoted as $\mathcal{X}_{:jk}, \mathcal{X}_{i:k} and \mathcal{X}_{ij:}$, respectively; see Figure 2 for a graphic illustration.

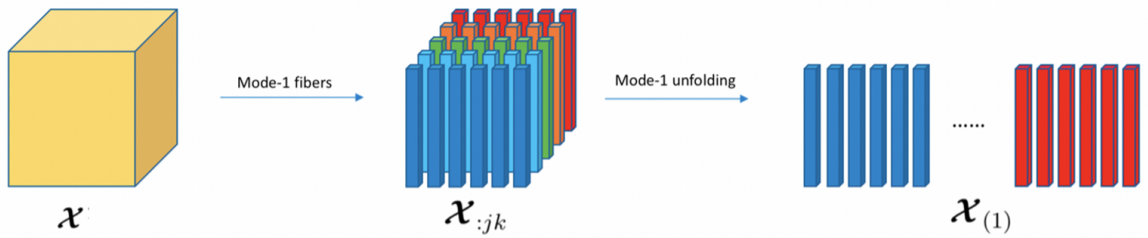

Figure 2: Tensor fibers and unfolding.

[1]The first introduction paragraph and the definitions introduced in the first part of next section are largely based on the previous article *Tensor in Modern Statistical Learning*[1]

Tensor *unfolding*, also known as tensor matricization, is a tensor operation that arranges tensor fibers into a matrix. Given a tensor $\mathcal{X} \in \mathbf{R}^{p_1 \times p_2 \times \cdots \times p_D}$, the mode-d unfolding, denotes as $\mathcal{X}_{(d)}$, arranges the mode-d fibers to be the columns of the resulting matrix. For example, the mold-1 unfolding of a third-order tensor $\mathcal{X} \in \mathbf{R}^{p_1 \times p_2 \times p_3}$, denoted by $\mathcal{X}_{(1)}$, results in the matrix $[\mathcal{X}_{:11}, \cdots, \mathcal{X}_{:p_2 1}, \cdots, \mathcal{X}_{:p_2 p_3}] \in \mathbf{R}^{p_1 \times (p_2 p_3)}$; again see Figure 2 for an illustration.

For a tensor $\mathcal{X} \in \mathbf{R}^{p_1 \times p_2 \times \cdots \times p_D}$ and a matrix $A \in \mathbf{R}^{J \times p_d}$, the *d-mode tensor matrix product*, denoted by $\times_d$, is defined as $\mathcal{X} \times_d A \in \mathbf{R}^{p_1 \times \cdots \times p_{d-1} \times J \times \cdots \times p_D}$. In this operation, each mode-d fiber of $\mathcal{X}$ is multiplied by the matrix A, and element-wisely, we have $(\mathcal{X} \times_d A)_{i_1, i_2, \cdots, i_D} = \sum_{j=1}^{p_d} \mathcal{X}_{i_1, \cdots, i_{d-1}, j, i_{d+1}, \cdots, i_D} A_{i_d j}$.

One interesting fact related to the d-mode tensor matrix product is that

$$\mathcal{Y} = \mathcal{X} \times_1 A^{(1)} \times_2 A^{(2)} \cdots \times_N A^{(N)} \iff Y_{(n)} = A^{(n)} X_{(n)} (A^{(N)} \otimes \ldots \otimes A^{(n+1)} \otimes A^{(n-1)} \otimes \ldots \otimes A^{(1)})^T, \tag{$*$}$$

where $\mathcal{X}, \mathcal{Y} \in \mathbf{R}^{p_1 \times p_2 \times \cdots \times p_N}$, $A^{(i)} \in \mathbf{R}^{p_i \times p_i}$, $X_{(n)}, Y_{(n)}$ are mode-n unfolding of tensor $\mathcal{X}, \mathcal{Y}$, and $\otimes$ is Kronecker product. (A simple proof is available *here*[3].) We will use this property (referred to as property ($*$) below) to deduce our algorithm.

Model & Algorithm

The simple tensor regression model is defined as:

$$\mathcal{Y} = \mathcal{X} \times_1 B^{(1)} \cdots \times_M B^{(M)} \times_{M+1} \mathbf{I}_N + \mathcal{E}, \tag{1}$$

where $\mathcal{X}, \mathcal{Y}, \mathcal{E} \in \mathbf{R}^{p_1 \times p_2 \times \cdots \times p_{M+1}}$ are tensors with the same structure. $\mathcal{X}$ is the explanatory variable, $\mathcal{Y}$ is the response variable, and $\mathcal{E}$ is the error term. Here $\mathcal{E}$ is assumed to have a distribution of high-dimension Gaussian with 0 correlations. $B^{(m)} \in \mathbf{R}^{p_m \times p_m}$ $(m = 1, \cdots, M)$ are matrices of parameters. Note that in this model, we are considering the last dimension of the tensors $(\mathcal{X}, \mathcal{Y})$ as a simple stack, i.e., the random tensors $(\mathcal{X}, \mathcal{Y})$ are a combination of p_{M+1} small tensors with dimension $p_1 \times p_2 \times \cdots \times p_M$. And that is why we are using $\mathbf{I}_N$ as our last matrix of parameters. Otherwise, without this assumption, there would be too many parameters and resulting in disaster estimation.

Our goal is to estimate $B^{(m)\prime}$s when $\mathcal{X}, \mathcal{Y}$ are known. That is, the problem would be solving

$$\underset{B^{(1)}, \cdots, B^{(M)}}{\operatorname{argmin}} L(B^{(1)}, \cdots, B^{(M)} | \mathcal{X}, \mathcal{Y}) = \underset{B^{(1)}, \cdots, B^{(M)}}{\operatorname{argmin}} ||\mathcal{Y} - \mathcal{X} \times_1 B^{(1)} \cdots \times_M B^{(M)} \times_{M+1} \mathbf{I}_N||_F^2.$$

An immediate algorithm would be computing

$$\underset{B^{(m)}}{\operatorname{argmin}} L(B^{(m)} | \mathcal{X}, \mathcal{Y}, B^{(1)}, \cdots, B^{(m-1)}, B^{(m+1)}, \cdots, B^{(M)}) = \underset{B^{(m)}}{\operatorname{argmin}} ||\mathcal{Y} - \mathcal{X} \times_1 B^{(1)} \cdots \times_M B^{(M)} \times_{M+1} \mathbf{I}_N||_F^2$$

repeatedly for m from 1 to M until the objective function L converges.

Using property $*$, the above optimization problem would be reduced to iteratively solving

$$\underset{B^{(m)}}{\operatorname{argmin}} ||Y_{(m)} - B^{(m)} \tilde{X}^m||_F^2,$$

where $\tilde{X}^m = X_{(m)} (B^{(N)} \otimes \ldots \otimes B^{(m+1)} \otimes B^{(m-1)} \otimes \ldots \otimes B^{(1)})^T$.

Overall, our algorithm is concluded below:

Algorithm A

Input: predictor $\mathcal{X}$, response $\mathcal{Y}$
Output: $\mathbf{B}^m, m = 1, \cdots, M$
Initialize $\mathbf{B}^m, m = 1, \cdots, M$
Iterate until convergence:
  For $m = 1, \cdots, M$:
    $\tilde{\mathbf{X}}^{(m)} = \mathbf{X}_{(m)}(\mathbf{B}^{(M)} \otimes \cdots \otimes \mathbf{B}^{(m+1)} \otimes \mathbf{B}^{(m-1)} \otimes \cdots \otimes \mathbf{B}^{(1)})^{\mathrm{T}}$
    $\mathbf{B}^{(m)} = \mathbf{Y}_{(m)}(\tilde{\mathbf{X}}^{(m)})^{\mathrm{T}}(\tilde{\mathbf{X}}^{(m)}(\tilde{\mathbf{X}}^{(m)})^{\mathrm{T}})^{-1}$
  End for

Simulation Study

To evaluate the performance of this simple tensor regression algorithm, we decide to generate two random samples $(\mathcal{X}, \mathcal{Y}, \mathcal{E})$ and $(\mathcal{X}_2, \mathcal{Y}_2, \mathcal{E}_2)$, one for estimation and the other one for prediction. See below for a simple diagram (3) with detailed explanations following. All the simulation steps listed are done in Rstudio with the package *rTensor*[4]. The code and other supportive documents are available on GitHub repository: *Project_Tensor_Regression*.

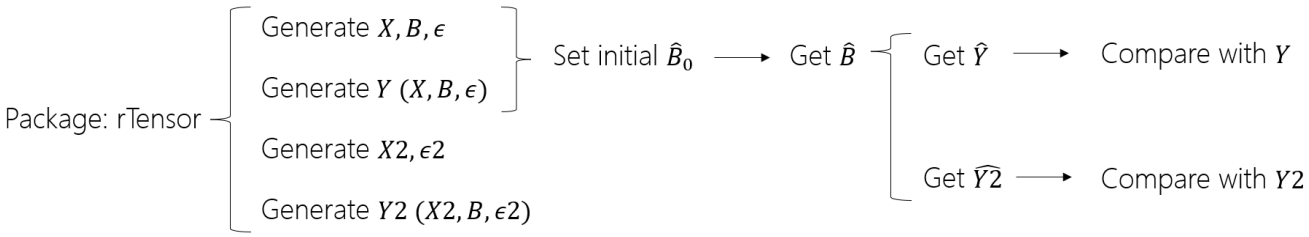

Figure 3: Simulation steps.

First, for a list of given $B^{(m)\prime}$s (randomly filled with independent normal samples), all $\mathcal{X}, \mathcal{X}_2, \mathcal{E}, \mathcal{E}_2$ are generated from a distribution of independent high-dimension Gaussian. Note that both $\mathcal{X}, \mathcal{X}_2$ are multiplied by a constant of 100 to make error terms $\mathcal{E}, \mathcal{E}_2$ relatively small. Otherwise, the prediction error could be huge.

As soon as we got $\mathcal{X}, \mathcal{X}_2, \mathcal{E}, \mathcal{E}_2$ and $B^{(m)\prime}$s, $\mathcal{Y}, \mathcal{Y}_2$ could be generated using model 1 with simple tensor summations as well as mode-d products.

After setting initial $\hat{B}_0^{(m)\prime}$s to identical matrices, we are now able to obtain our estimated $\hat{B}^{(m)\prime}$s by running the algorithm A mentioned above with data $(\mathcal{X}, \mathcal{Y})$. The stopping criteria are variable for the size of our datasets.

Plug back the estimated $\hat{B}^{(m)\prime}$s into the model, we can finally compute the prediction errors

$$\hat{\mathcal{Y}} = \mathcal{X} \times_1 \hat{B}^{(1)} \cdots \times_M \hat{B}^{(M)} \times_{M+1} \mathbf{I}_N, \text{ and } \hat{\mathcal{Y}}_2 = \mathcal{X}_2 \times_1 \hat{B}^{(1)} \cdots \times_M \hat{B}^{(M)} \times_{M+1} \mathbf{I}_N$$

We tried different settings of tensor structure (different dimensions for all variables) and the corresponding structure of parameter matrices. The detailed results of simulation are illustrated in the next section.

Results & Conclusion

In most cases, the iteration converges very quickly (e.g. figure 4). It usually only takes less than 20 iterations when we set the dimension of tensors to be, for instance, $(3 \times 4 \times 10)$, $(10 \times 10 \times 5 \times 30)$. or even $(2 \times 2 \times 2 \times 2 \times 2 \times 30)$. The algorithm converges fast possibly because the matrix inversion step won't be time-consuming since the matrices generated by independent variables do not have collinearity problem.

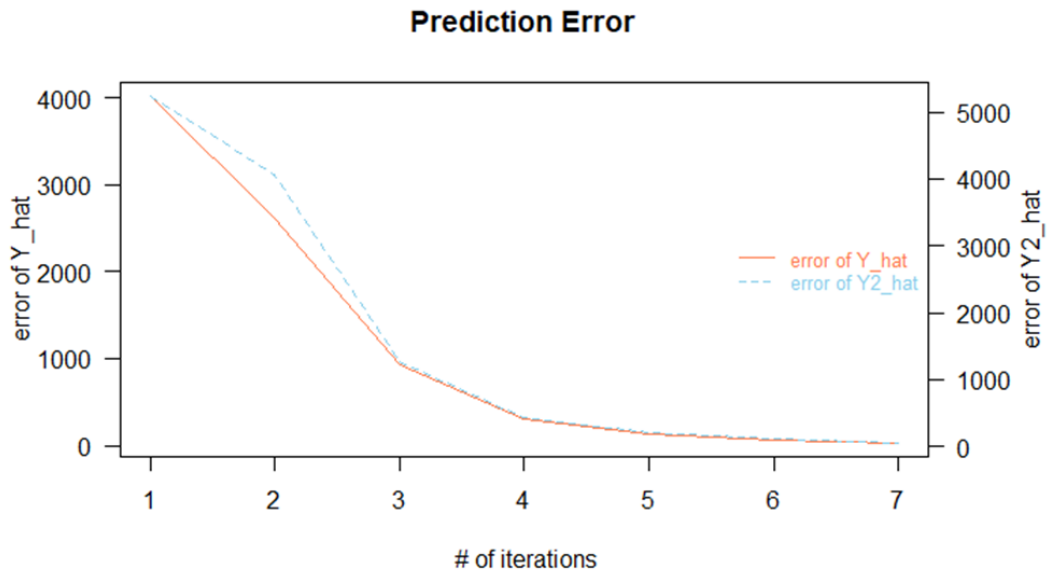

Figure 4: Simulation result. The orange line represents $||\mathcal{Y} - \hat{\mathcal{Y}}||_F$ while the blue dashed line is $||\mathcal{Y}_2 - \hat{\mathcal{Y}}_2||_F$. Both estimation group ($\mathcal{Y}$) and prediction group ($\mathcal{Y}_2$) converges in less than 10 iterations.

Notice that in the case of $(2 \times 2 \times 2 \times 2 \times 2 \times 30)$ setting, the actual number of parameters that we used is $2^2 + 2^2 + 2^2 + 2^2 + 2^2 = 20$. On the other hand, if we use a linear model instead, it would take $2 \times 2 \times 2 \times 2 \times 2 = 32$ parameters! In fact, our tensor regression model is simpler than naive linear model for high-dimensional data problems.

However, there do exist certain situations that the algorithm does not converge at all. The estimation fell into local minimum and failed to climb out. This would happen when we set, for example, dimensions of tensors to be $(3 \times 5 \times 10)$ with random seed 6 or $(2 \times 3 \times 4 \times 10)$ with random seed 6.

Overall, running on relatively small datasets, the simulation results show that our simple LS tensor regression model performed well in most of the cases in the sense of both estimation and prediction. Especially we observed that it can be efficient for high-dimensional data, which leads to possible application in real data analysis.

Future Discussion

As we mentioned above, evaluating the performance of the model in large datasets is a common concern. Besides, there remain a lot of interesting points regarding our model that could be explored

in the future. For example, in most cases, the data $\mathcal{X}$ and $\mathcal{Y}$ do not have the same structure. Therefore, we should consider changing the parameter matrices into non-square matrices. Also, we need to add a penalty term if the coefficients are assumed to have sparsity property. Moreover, the distribution of the error $\mathcal{E}$ could be non-normal or dependent on each other, where a transformation could be adopted.

References

[1] Will Wei Sun, Botao Hao, and Lexin Li. Tensors in modern statistical learning. *Wiley StatsRef: Statistics Reference Online*, pages 1–25, 2014.

[2] Zhen Long Ce Zhu Yipeng Liu, Jiani Liu. *Tensor Computation for Data Analysis*. Springer Cham.

[3] Lana Periša (https://math.stackexchange.com/users/452859/lana peri%c5%a1a). n mode product and kronecker product relation. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2312166 (version: 2017-06-06).

[4] James Li, Jacob Bien, and Martin T. Wells. rTensor: An R package for multidimensional array (tensor) unfolding, multiplication, and decomposition. *Journal of Statistical Software*, 87(10):1–31, 2018.

Appendix

0. Function preparation

```
create_B=function(d,random=FALSE){#a function to initialize B's
  l=vector('list',k)
  if(random){for(i in 1:k) l[[i]]=matrix(rnorm(dimension[i]^2),dimension[i])}
  else {for(i in 1:k) l[[i]]=diag(dimension[i])}
  return(l)
}
multiple_ttm=function(x,b){#a function to perform consecutive mode-d products
  if(length(b)>1) return(ttm(multiple_ttm(x,b[-length(b)]),b[[length(b)]],length(b)))
  return(ttm(x,b[[1]],1))
}
```

1. Generate Data

```
library(rTensor) #https://cran.r-project.org/web/packages/rTensor/index.html
set.seed(5)
dimension=c(3,4,10)
k=length(dimension)-1
e=rand_tensor(dimension)
X=100*rand_tensor(dimension)
B=create_B(dimension,random=TRUE) #generate random B's
Y=multiple_ttm(X,B)+e
X2=100*rand_tensor(dimension)
e2=rand_tensor(dimension)
Y2=multiple_ttm(X2,B)+e2
#X;B;e;Y # view generated sample data
```

2. Regression

```
N=500 #number of iterations
B_hat=create_B(dimension) #initialize B's as identical matrices
#B_hat=lapply(1:k, function(i) return(B[[i]]+matrix(rnorm(dimension[i]^2),dimension[i])/10))
#B;B_hat
B_hats=list()
Error_Y_hat=numeric(0) #store the prediction errors comparing to Y
Error_Y2_hat=numeric(0) #store the prediction errors comparing to Y2
for(i in 1:N){
  for(j in 1:k){ #each regression iteration
    X_j=k_unfold(X,j)@data
    temp=B_hat[-j]
    temp[[k]]=diag(dimension[k+1])
    X_tilde=X_j%*%t(kronecker_list(rev(temp)))
    Y_j=k_unfold(Y,j)@data
    B_hat[[j]]=Y_j%*%t(X_tilde)%*%solve(X_tilde%*%t(X_tilde))
    #print(B_hat)
  }
  B_hats[[i]]=B_hat #update after each small iteration
  Y_hat=multiple_ttm(X,B_hat)
```

```
  Y2_hat=multiple_ttm(X2,B_hat)
  Error_Y_hat=c(Error_Y_hat,fnorm(Y_hat-Y))
  Error_Y2_hat=c(Error_Y2_hat,fnorm(Y2_hat-Y2))
  #cat(i,fnorm(Y_hat-Y),fnorm(Y2_hat-Y2),'\n')
  if(tail(Error_Y_hat,1)<50) break
}
par(mar=c(5,4,4,4)+0.3) #draw the graph
plot(Error_Y_hat,type='l',col='coral',axes=FALSE,xlab='\ # of iterations',
     ylab='error of Y_hat',main='Prediction Error')
axis(side=2,at=pretty(range(Error_Y_hat)),las=1)
box()
par(new=TRUE)
plot(Error_Y2_hat,type='l',col='skyblue',axes=FALSE,xlab='',ylab='',lty=2)
axis(side=4,at=pretty(range(Error_Y2_hat)),las=1)
mtext("error of Y2_hat", side=4, line=3)
axis(1,pretty(range(1:length(Error_Y_hat))))
legend("right",legend=c("error of Y_hat","error of Y2_hat"),
  text.col=c("coral","skyblue"),lty=c(1,2),col=c("coral","skyblue"),cex=.8,bty="n")
```