Efficient Computation of Tucker Decomposition of Correlation-Based Tensors

Bill Xu

Advisor: Grey Ballard
in collaboration with Robert Lyday, and Paul Laurienti

May 13, 2020

Abstract

The Tucker decomposition is a generalization of the matrix singular value decomposition to tensors, which are multidimensional arrays. We seek to apply the decomposition as a dimension reduction technique in order to analyze large functional magnetic resonance imaging (f-MRI) datasets of human brains. Neuroscientists are particularly interested in correlation among different areas in the brain, but computing and storing pairwise correlations between all pairs of brain areas can be infeasible, especially when the data set includes multiple participants and multiple trials. The current practice is to downsample the data in order to reduce the number of brain areas in the data, but this process loses information. We show that the dimension reduction via Tucker decomposition can be computed without explicitly computing and storing all correlations, making data analysis with the original granularity feasible and efficient. We demonstrate the advantage of using the full granularity to answer scientific questions about the data, including classifying participants across multiple trials.

1 Introduction

1.1 Background

Correlation between different areas of brains has been topic of interest in neuroscience for a long time. One of the methods in brain imaging is Functional Magnetic Resonance Imaging (f-MRI), which measures brain activity by detecting changes of blood flow. While scanning brains using f-MRI, the unit of measurement for voxels is on the scale of cubic nanometers. For example, a typical available granularity is 4nm $\times$ 4nm $\times$ 4nm, which we will refer to as Voxel-based data. Because of the fine granularity, the data requires huge space of memory to store and time to process. Alternatively, neuroscientists have manually defined 268 anatomically specific regions, each of which combines different sets of voxels to save memory at a cost of reducing granularity. We call this coarser-grained representation Region-based data. Our research addresses two related problems: (1) Can we improve efficiency and reduce memory usage when processing f-MRI data? (2) What are the advantages of using Voxel-based over Region-based data in answering neuroscience questions?

1.2 Preliminaries

1.2.1 Singular Vector Decomposition

The Singular Value Decomposition (SVD) is a technique for linear dimensionality reduction to project high dimensional data to a lower dimensional space. In our research, SVD decomposes a flattened Neural Activity Tensor into matrix multiplication of left singular vectors, singular values, and right singular vectors. In this way we can map high dimensional f-MRI data into a low dimensional matrix of left singular vectors.

1.2.2 Machine Learning Models

We have implemented multiple machine learning models to experiment on scientific questions, including Random Forest, Gradient Boosting Decision Trees and K-Nearest Neighbors and Generalized Linear Regression. A random forest [1] is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Gradient Boosting [2] builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. Both of them exploit ensemble to improve predictive accuracy and generalizability. Gradient Boosting Decision Trees builds on weak learners (shallow trees) while Random Forest aims at fully growing decision trees. Therefore GBDT tends to reduce bias while Random Forest reduces variance.

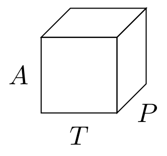
2 Data

The data we use comes from f-MRI (Functional Magnetic Resonance Imaging) brain image scans. Such scans of brain take places at certain intervals (or timesteps) for each participant. Therefore we can store the data in a 3 dimensional tensor called Activity Tensor $\boldsymbol{\mathcal{A}}$. Beyond that, we need several transformation based on $\boldsymbol{\mathcal{A}}$. We define a Static Correlation Tensor as $\boldsymbol{\mathcal{S}}$, Window Activity Tensor $\mathcal{W}$, Dynamic Correlation Tensor $\mathcal{C}$. Besides P is number of participants, A is number of brain regions (areas), T is number of time points, W is width of window, $N = T - W + 1$ is number of windows

2.1 Activity Tensor

In either region based or voxel based methods, the activity tensor, $\boldsymbol{\mathcal{A}}$ of the brain data is a three way tensor of Area (A), Time(T) and Participant(P). The entry a_{ijk} is a record of either regional or voxel brain data of area i, at time j for participant j.

$\boldsymbol{\mathcal{A}} =$



For the purpose of computing correlation, we define $\hat{\boldsymbol{\mathcal{A}}}$ from $\boldsymbol{\mathcal{A}}$ by shifting each time series towards center and normalizing.

$$\hat{\boldsymbol{\mathcal{A}}}(a,:,p) = \frac{\boldsymbol{\mathcal{A}}(a,:,p) - \overline{\boldsymbol{\mathcal{A}}(a,:,p)}}{\|\boldsymbol{\mathcal{A}}(a,:,p) - \overline{\boldsymbol{\mathcal{A}}(a,:,p)}\|_2}$$

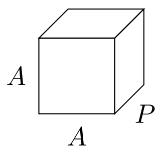
2.2 Static Analysis Correlation Tensor

The static analysis correlation tensor $\mathcal{S}$ is a three way tensor (A by A by P). $\mathcal{S}_{ijk}$ is the statistical correlation between two areas fibers A_{ik} and A_{jk} over T for participant k. The statistical correlation of two vectors x and y is defined by

$$\operatorname{corr}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} = \hat{\mathbf{x}}^T \hat{\mathbf{y}}$$

where $\hat{\mathbf{x}} = \frac{\mathbf{x} - \overline{x}\mathbf{1}}{\|\mathbf{x} - \overline{x}\mathbf{1}\|_2}$. Thus,

$$\boldsymbol{\mathcal{S}} =$$



where $\boldsymbol{\mathcal{S}}(:,:,p) = \hat{\boldsymbol{\mathcal{A}}}(:,:,p)\hat{\boldsymbol{\mathcal{A}}}(:,:,p)^T$.

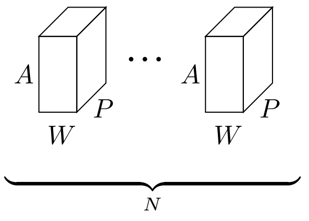
2.3 Dynamic Analysis Correlation Tensor

To calculate correlation between areas of a given participant at a specific time n, one should not use area fibers over full time, $\boldsymbol{\mathcal{A}}(a,:,p)$, to compute correlation. Instead, we propose sliding windows at time n with size W to compute correlation.

2.3.1 Windowed Activity Tensor

The windowed activity tensor $\boldsymbol{\mathcal{W}}$ is a four way tensor of Area (A), Time (W), Participant (P), and Window (N) . Entry $\boldsymbol{\mathcal{W}}_{ijkn}$ is a record of either regional or voxel brain data of area i, at window time j for participant j and window index n.

$$\boldsymbol{\mathcal{W}} =$$



Similar to its static counterpart, we define $\hat{\boldsymbol{\mathcal{X}}}$ $\hat{\boldsymbol{\mathcal{W}}}$ from $\boldsymbol{\mathcal{W}}$ by shifting and normalizing towards center.

$$\hat{\boldsymbol{\mathcal{W}}}(a,:,p,n) = \frac{\boldsymbol{\mathcal{W}}(a,:,p,n) - \overline{\boldsymbol{\mathcal{W}}(a,:,p,n)}}{\|\boldsymbol{\mathcal{W}}(a,:,p,n) - \overline{\boldsymbol{\mathcal{W}}(a,:,p,n)}\|_2}$$

2.3.2 Correlation Tensor

The dynamic correlation tensor $\boldsymbol{\mathcal{C}}$ is a four way tensor (A by A by P by N). The entry $\boldsymbol{\mathcal{C}}_{ijpn}$ is the statistical correlation between two areas i and j for participant k at time n.

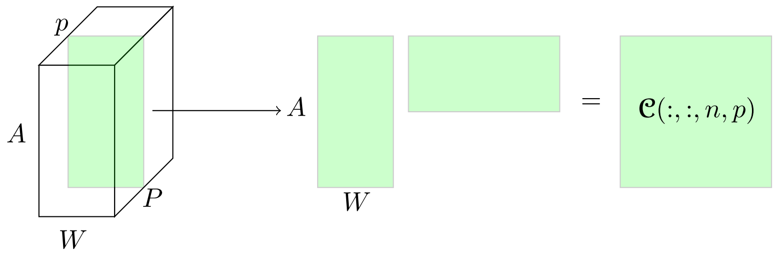
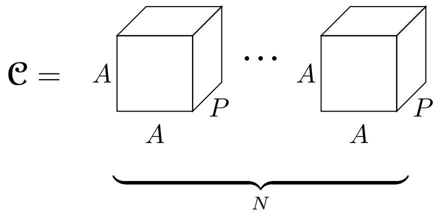


Figure 1: Subtensor corresponding to correlation of participant p and window n



where $\mathcal{C}(:,:,p,n) = \hat{\mathcal{W}}(:,:,p,n)\hat{\mathcal{W}}(:,:,p,n)^T$, and $\mathcal{C}(i,j,p,n)$ is the inner product of two vectors $\hat{\mathcal{W}}(i,:,p,n)$ and $\hat{\mathcal{W}}(j,:,p,n)$

3 Algorithms

3.1 Static Analysis Correlation Tensor

In case where we view tensor $\mathcal{S}$ as correlation of areas for participants over **all** time points of $\mathcal{A}$, the algorithm is presented as Algorithm 1. Notice that certain time series vectors must be centered and normalized before used to calculate correlation matrix.

Algorithm 1 Static Analysis Correlation Tensor

```
Require: $\mathcal{A}$ is original 3-way data tensor with dimension $A \times T \times P$
1: for p = 1:P do
2:     for a = 1:A do
3:         $\hat{\mathcal{A}}(a,:,p) = \mathcal{A}(a,:,p) - \overline{\mathcal{A}(a,:,p)}$          ▷ center time series vector
4:         $\hat{\mathcal{A}}(a,:,p) = \hat{\mathcal{A}}(a,:,p)/\|\hat{\mathcal{A}}(a,:,p)\|_2$          ▷ normalize time series vector
5:     end for
6:     $\mathcal{S}(:,:,p) = \hat{\mathcal{A}}(:,:,p)\hat{\mathcal{A}}(:,:,p)^T$          ▷ calculate correlation matrix
7: end for
```

3.2 Dynamic Analysis Correlation Tensor

Our primary goal is to compute the left singular vectors of $\mathbf{C}_{(4)}$, the participant-mode unfolding of dynamic correlation tensor $\mathcal{C}$. Thus, an intuitive algorithm is based on our experience in the previous section: explicitly compute windowed activity tensor $\mathcal{W}$, then calculate the corresponding correlation tensor $\mathcal{C}$, and finally form the left singular vectors of $\mathbf{C}_{(4)}$. This approach is presented as Algorithm 2.

Algorithm 2 Left Singular Vector of Dynamic Analysis Correlation Tenso r(explicit)

```
Require: $\boldsymbol{\mathcal{A}}$ is original 3-way data tensor with dimension $A \times T \times P$
Require: $W$ is the window size in windowed activity tensor
 1: N = T - W + 1                                        ▷ number of windows
 2: for p = 1:P do
 3:     for n = 1:N do
 4:         $\mathbf{W} = \boldsymbol{\mathcal{A}}(:, n : n + W - 1, p)$
 5:         for a = 1:A do
 6:             $\hat{\mathbf{W}}(a,:) = \mathbf{W}(a,:) - \overline{\mathbf{W}(a,:)}$          ▷ center (windowed) time series vector
 7:             $\hat{\mathbf{W}}(a,:) = \hat{\mathbf{W}}(a,:)/\|\hat{\mathbf{W}}(a,:)\|_2$        ▷ normalize (windowed) time series vector
 8:         end for
 9:         $\boldsymbol{\mathfrak{C}}(:,:,n,p) = \hat{\mathbf{W}}\hat{\mathbf{W}}^{\mathsf{T}}$          ▷ calculate 2D slice of dynamic correlation tensor
10:     end for
11: end for
12: $\mathbf{G} = \mathbf{C}_{(4)}\mathbf{C}_{(4)}^{\mathsf{T}}$          ▷ Compute Gram matrix of participant mode unfolding
13: $[\mathbf{U}, \mathbf{\Lambda}] = \mathrm{EVD}(\mathbf{G})$          ▷ $\mathbf{U}$ := eigenvectors; $\mathbf{\Lambda}$ := eigenvalues
14: Sort $\mathbf{U}, \mathbf{\Lambda}$ by descending order of $\mathbf{\Lambda}$
15: $\mathbf{\Sigma} = \sqrt{\mathbf{\Lambda}}$          ▷ singular values of $\mathbf{C}_{(4)}$, $\mathbf{U}$ are left singular vectors
```

There are two steps involved in this algorithm: (1) Computing correlation tensor $\boldsymbol{\mathfrak{C}}$, which costs A^2WPN flops because for each of N windows, and for each of P participants, there is a need to compute a matrix multiplication of shape $A \times W$ and its transpose, and such multiplication costs A^2W flops (since the result is symmetric). (2) Computing the SVD via and EVD of the Gram matrix to recover left singular vectors. Since $\mathbf{C}_{(4)}$ is a matrix of size $P \times (A \times A \times N)$, computing the Gram matrix costs A^2P^2N flops.

The space complexity is attributed to (1) Activity tensor $\boldsymbol{\mathcal{A}}$ which takes ATP unit of space. (2) Correlation tensor $\boldsymbol{\mathfrak{C}}$ which takes A^2NP unit of space.

Alternatively, we propose an approach that implicitly calculates the Gram matrix $\mathbf{G}$, resulting in both computational and spatial efficiency particularly when A is large. The implicit algorithm is based on following derivation:

$$\begin{aligned}
(\mathbf{C}_{(4)}\mathbf{C}_{(4)}^T)_{ij} &=< \mathbf{M}_i, \mathbf{M}_j > \\
&= \sum_n^N < C_n^i, C_n^j > \\
&= \sum_n^N < A_n^i {A_n^i}^T, A_n^j {A_n^j}^T > \\
&= \sum_n^N < {A_n^i}^T A_n^i, {A_n^j}^T A_n^j > \\
&= \sum_n^N \|{A_n^i}^T A_n^j\|_F^2
\end{aligned}$$

where $C_n^i = \boldsymbol{\mathfrak{C}}(:,:,n,i)$ and $A_n^i = \boldsymbol{\mathcal{A}}(:, n : n + W - 1, i)$.

As a result, it is possible to compute $\mathbf{C}_{(4)}\mathbf{C}_{(4)}^T$ without explicitly forming $\mathbf{C}_{(4)}$, but through the element-wise sum of squares of the matrix multiplication of activity window matrix for participants i and j. We present the implicit approach as Algorithm 3.

Algorithm 3 Left Singular Vector of Dynamic Analysis Correlation Tensor(implicit)

```
Require: $\boldsymbol{\mathcal{A}}$ is original 3-way data tensor with dimension $A \times T \times P$
Require: $W$ is the window size in windowed activity tensor
 1: N = T - W + 1                                        ▷ number of windows
 2: $\mathbf{G} = \mathbf{0}$                                          ▷ Initialize $\mathbf{G}$ to be $P \times P$
 3: for n = 1:N do
 4:     for i = 1:P do
 5:         $\mathbf{A_I} = \boldsymbol{\mathcal{A}}(:, n : n + W - 1, i)$
 6:         for a = 1:A do
 7:             $\hat{\mathbf{A_I}}(a,:) = \mathbf{A_I}(a,:) - \overline{\mathbf{A_I}(a,:)}$     ▷ center (windowed) time series vector
 8:             $\hat{\mathbf{A_I}}(a,:) = \hat{\mathbf{A_I}}(a,:)/\|\hat{\mathbf{A_I}}(a,:)\|_2$   ▷ normalize (windowed) time series vector
 9:         end for
10:         for j = 1:i do
11:             $\mathbf{A_J} = \boldsymbol{\mathcal{A}}(:, n : n + W - 1, j)$
12:             for a = 1:A do
13:                 $\hat{\mathbf{A_J}}(a,:) = \mathbf{A_J}(a,:) - \overline{\mathbf{A_J}(a,:)}$     ▷ center (windowed) time series vector
14:                 $\hat{\mathbf{A_J}}(a,:) = \hat{\mathbf{A_J}}(a,:)/\|\hat{\mathbf{A_J}}(a,:)\|_2$   ▷ normalize (windowed) time series vector
15:             end for
16:             $\mathbf{G}(i, j)$ += $\|\hat{\mathbf{A_I}}^T \hat{\mathbf{A_J}}\|_F^2$                         ▷ core update
17:         end for
18:     end for
19: end for
20: $\mathbf{G} = \mathbf{G} + \mathtt{tril}(\mathbf{G}, -1)^\mathsf{T}$          ▷ Copy lower triangle of $\mathbf{G}$ into upper triangle
21: $[\mathbf{U}, \mathbf{\Lambda}] = \mathrm{EVD}(\mathbf{G})$               ▷ $\mathbf{U}$ := eigenvectors; $\mathbf{\Lambda}$ := eigenvalues
22: Sort $\mathbf{U}, \mathbf{\Lambda}$ by descending order of $\mathbf{\Lambda}$
23: $\mathbf{\Sigma} = \sqrt{\mathbf{\Lambda}}$            ▷ singular values of $\mathbf{C}_{(4)}$, $\mathbf{U}$ are left singular vectors
```

The computational cost for the implicit algorithm is AW^2NP^2 since for each pair of P participants, and for each of N windows, there's a matrix multiplication of $\hat{\mathbf{A_I}}^T \hat{\mathbf{A_J}}$ that costs $2AW^2$.

The spatial cost for such implicit algorithm is always smaller than the explicit one – only activity tensor $\boldsymbol{\mathcal{A}}$ needs to be stored, thus a spatial complexity of ATP.

3.3 Efficiency Comparison

By the analysis in the previous section, we may compare the efficiency of two methods. The speedup for both time and space complexity is dependent on variable values A, W, P, N and T.

For our experimental data, we have a brain dataset involving 61 participants, 147 time points and window size of 60. Thus, the number of windows $N = T - W + 1 = 88$. If regional based sampling is adopted, we have 268 units of area, while voxel based sampling contains around 20,000 units of area.

With Region-based data, the speedup of implicit method over explicit method, in terms of time, is $0.8\times$ which actually decreases in speed. However, the speedup increases to $61\times$ under voxel based level.

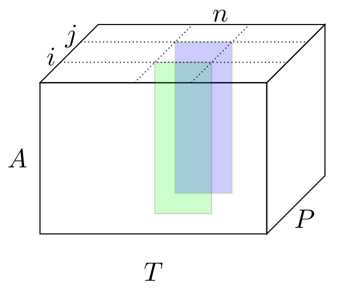


Figure 2: Visualization of inner loop of Algorithm 3 ($\hat{\mathbf{A}}_{\mathbf{I}}$ and $\hat{\mathbf{A}}_{\mathbf{J}}$)

	Time Complexity	Space Complexity
Explicit Method	$A^2WPN+ A^2P^2N$	$A^2NP + ATP$
Implicit Method	AW^2NP^2	ATP
Speedup	$\frac{A(W+P)}{W^2P}$	$\frac{AN}{T}+1$

Table 1: Time and space complexity compared between explicit and implicit methods.

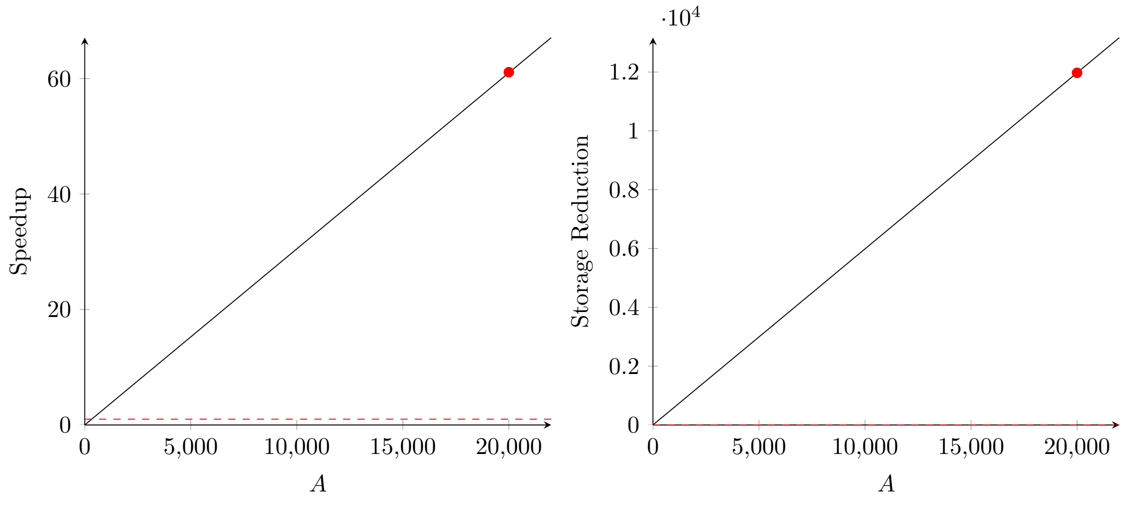


Figure 3: Speedup in time and storage reduction corresponding to number of areas. For the experimental data, implicit method costs less time than explicit method as $A > 327$ and always requires less memory than explicit method. The red line corresponds to value 1 to indicate when implicit method outperforms explicit method.

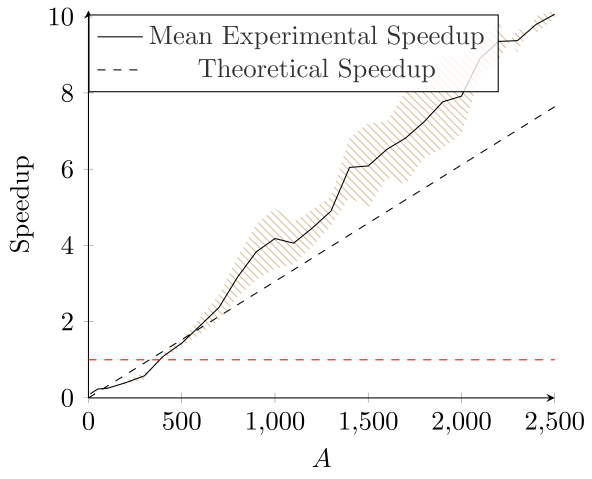


Figure 4: Speedup experiment on synthetic data, the speedup is roughly linear and exceeding 1 at $300 < A < 400$, a result corresponding with the theoretical analysis. The red line corresponds to value 1 to indicate when implicit method outperforms explicit method. The shadowed area is the range of 10 experimental speedups.

In terms of spatial complexity, the implicit method is 160.44 times more efficient for Region-based data, and 11,972.79 times more efficient for Voxel-based data. The implicit algorithm is always better in terms of spatial efficiency.

3.4 Experiment

Using the same values of T, W, and P as the experimental data in the previous section, we generate synthetic data with varying values of A and record the corresponding experimental speedup. We plot the maximum, minimum and mean values of speedups out of 10 trials below. We also tested the performance of both methods on our machine by evaluating GFLOPS (gigaflops per second). We see that implicit method becomes faster when A exceeds approximately 350, agreeing with the theoretical analysis. For larger A, we see that the experimental speedup exceeds the theoretical one (based solely on flop counts). This deviation is attributed to the difference in memory footprints; because the memory footprint of the implicit method is smaller, it enjoys much better cache utilization than the explicit method. At $A = 2{,}500$, the memory footprint of the explicit algorithm is 24.58 GB while the footprint of the implicit algorithm is only 16.82 MB. Because of the memory footprint of the explicit method, we could not compute empirical speedups for larger A. However, we have run implicit algorithm on the same synthetic data where A increases to 20000, and it only takes 18.26 seconds on a single thread of 2.6 GHz Intel Core i7.

Similarly, the implicit algorithm on our real voxel based data, where $A = 18225, W = 60, T = 156, P = 183$ takes 496.2MB of memory footprint and 298.8 seconds to run. It is almost impossible to store and process such data with the explicit algorithm on a normal computer.

We present the raw GFLOPS in Figure 5. We see that both methods perform at roughly the same efficiency for small values of A, so the speedup attained by the implicit method maps well to the theoretical comparison of the flop count. However, for larger A, we see a deviation in the performance of explicit and implicit methods, which helps explain why the empirical speedup is larger.

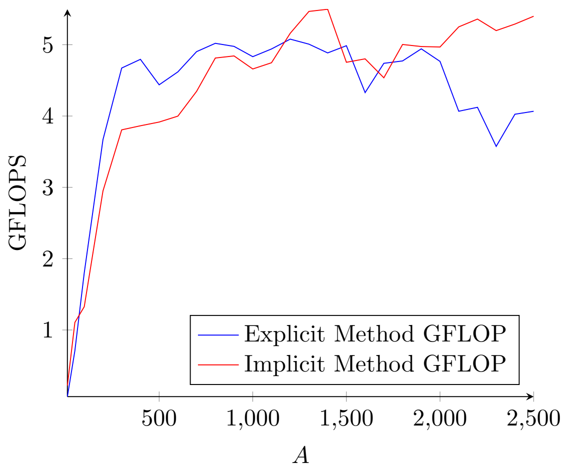


Figure 5: GFLOPS comparison of the same experiment on synthetic data

4 Voxel Based vs. Regional Based

x Given an efficient way of performing dimensionality reduction for Voxel-based data, our next question is, is it worth replacing Region-based analysis by Voxel-based analysis? We start off comparing the singular values and vectors resulted from the experiment above.

4.1 Singular values and vectors

Regional based activity tensor generates singular values that converge much more quickly to 0, while singular values from voxel based data are generally higher in quantity and decrease much more smoothly.

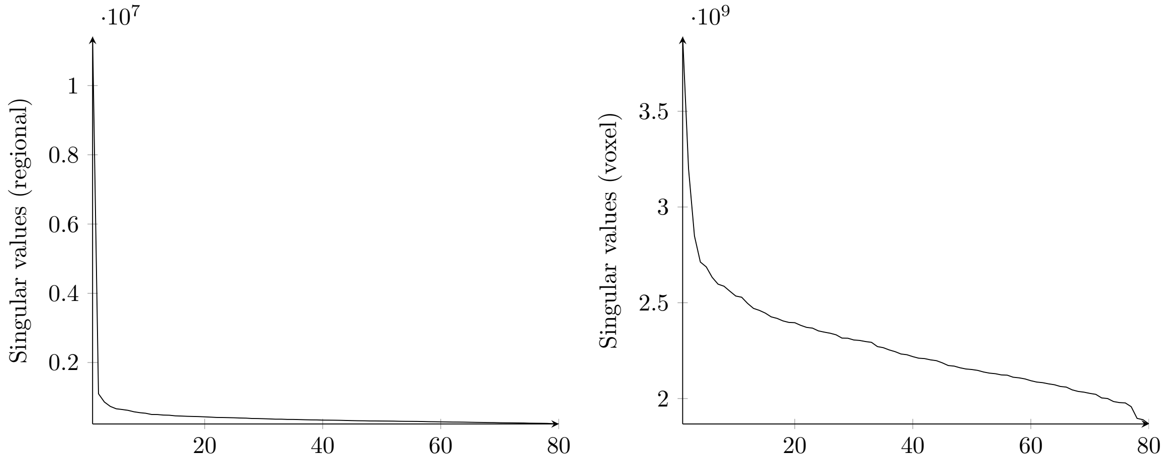
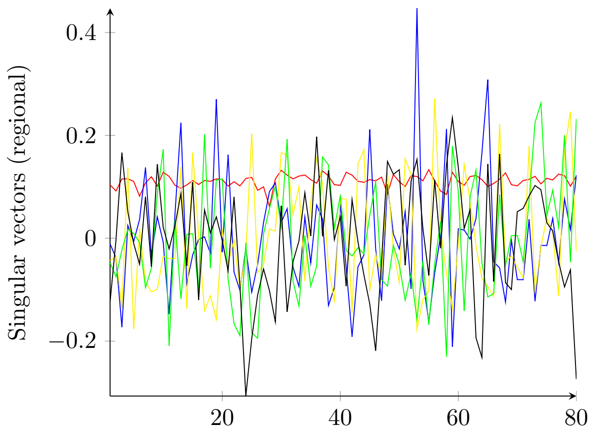


Figure 6: Singular values computed from two techniques.

The singular vectors from regional based data seems like random numbers around 0 without special patterns, while those from voxel based data contains several extreme values far from 0. Since x-axis corresponds to participants P, these spikes indicates that voxel based data contains some outstanding tendency for different participants.



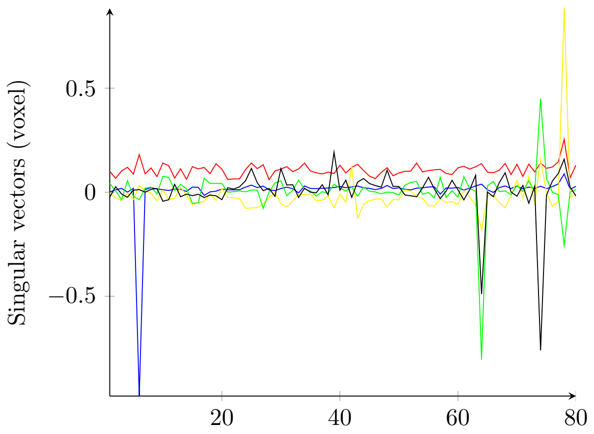


Figure 7: Singular vectors corresponding to the 5 biggest singular values computed from two techniques.

	Trained on Regonal Data	Trained on Voxel Data
Dummy Classifer	.016	.016
Random Forest	.007	.289
GBDT	0.497	.836
KNN	.011	.032

Table 2: Individual classification accuracy trained on regional and voxel data using Random Forest, Gradient Boosting Decision Trees and K-Nearest Neighbors classifiers.

	Trained on Regonal Data	Trained on Voxel Data
Dummy Classifer	.639	.639
Random Forest	.622	.743
GBDT	0.852	.857
KNN	.546	.595

Table 3: Farm/NonFarm Workers classification accuracy trained on regional and voxel data using Random Forest, Gradient Boosting Decision Trees and K-Nearest Neighbors classifiers.

4.2 Individual Classification

One of the scientific questions we try to answer is classification over participants P. The data we use for this task contains 61 participants each testing 3 tasks, thus $P = 183$. We label 61 individual participants and processed the data using our implicit algorithm to get a matrix of left singular vectors with dimension 183×183. To get fair accuracy, we use stratified 3-fold cross-validation to divide the data into 3 parts, each with 61 distinguished labels of participants. Then we iteratively train machine learning models based on two of three parts and use the left-out part of data to valid model accuracy. Our final accuracy is calculated as the average of the three validation scores.

We observe higher classification accuracy for Voxel-based data in machine learning models. The most successful model is GBDT (Gradient Boosting Decision Tree), which achieves 83.6% classification accuracy on voxel data while only 49.7% on regional data. It is a sheer win for voxel data to classify individual participants.

We also implemented other models such as Supporting Vector Machines [3] and Generalized Linear Regression. However they failed to converge due to the limited number of samples within classes.

4.3 Farm/Nonfarm Worker Classification

Using the same data as the previous section, we divide the 61 participants into 22 farm workers and 39 non-farm workers. Then we label the total 183 cases into two classes before a stratified 3 fold cross validation. However, the result of this task fails to prove the advantage of voxel based data over regional data. The classification accuracy for both form of data are close enough to each other.

4.4 Clustering

Besides supervised classification, we also tried to cluster one of the three tasks in an unsupervised way. After preprocessing with implicit algorithm, we have a 61×61 matrix of singular vectors. KMeans clustering on these singular vectors can separate them into two groups. Then we compare the result with Farm/Nonfarm worker labels to compute target similarity between them. It turns out that voxel based data seems hard for

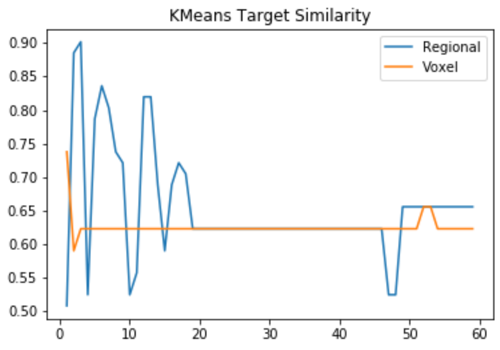


Figure 8: Farm/Non-farm worker classification accuracy

Kmeans to find any useful pattern. On the other side, regional data reaches its maximum as we use two leading left singular vectors, but fails to work as the number of singular vectors increases.

5 Conclusion

So far we have developed an efficient algorithm in performing SVD on correlation activity tensors. The speedup and storage reduction are significant both theoretically and practically. Using an efficient algorithm, we have performed several experiments and found that using voxel based data answers individual classification problem more accurately than region-based data. More experiments must be made in order to gain comprehensive insight of the advantage of voxel data, and we believe our implicit algorithm lays a basis for the potential research in the future.

References

[1] Breiman. “random forest”. *Machine Learning*, 2001.

[2] Jerome H Friedman. “greedy function approximation: a gradient boosting machine”. *Annals of statistics*, page 1189–1232, 2001.

[3] John C. Platt. “probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. 1999.