Efficient CP-ALS and Reconstruction From CP

Jed A. Duersch & Tamara G. Kolda
Sandia National Laboratories
Livermore, CA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.

This research was funded by LDRD project 199986.
Tensor Notation

$\mathbf{y}$ is a tensor.

$d$ is the order. Number of indices to locate an element. $\mathbf{y}_{(i_1,i_2,\ldots,i_d)}$

$n$ is the size of each mode. Generally mode $k$ has size $n_k$, but all set to $n$ for simplicity.

(It’s a hypercube!)
Canonical Polyadic Decomposition (CPD)

\[ \mathbf{Y} = \mathbf{a}_1 \mathbf{B}_1 \mathbf{C}_1 + \mathbf{a}_2 \mathbf{B}_2 \mathbf{C}_2 + \cdots + \mathbf{a}_r \mathbf{B}_r \mathbf{C}_r \]

CPD (aka CANDECOMP/PARAFAC) expresses \( \mathbf{Y} \) as a sum of rank-1 tensor products.

- \( d \) is the order.
- \( n \) is the size of each mode.
- \( r \) is the number of components.
As we discuss computations and memory requirements, keep this example in mind.

Order: $d = 4$
Mode-size: $n = 100$
Components: $r = 10$
Tensor Size: $n^d = 10^8$
Factor Matrices

We can arrange vector components as columns in factor matrices.

Ex: $10^8$

Ex: $10^3$ each

Ex: $d = 4 \ n = 100 \ r = 10$
**Problem: Reconstruct from CPD**

Sometimes you need to reconstruct the full tensor. This is `full()` in Tensor Toolbox for MATLAB.

Orange \[ O(n^d) \]

Ex: \( 10^8 \)

\[ y(i_1, i_2, \ldots, i_d) = \sum_{j=1}^{r} \lambda_j \prod_{k=1}^{d} A_{ikj}^{(k)} \]

Sum over each component

Optionally include scaling factors for each component

Product over factor matrices

Blue \( O(rn) \)

Ex: \( 10^3 \)

Compute: \( r \cdot n^d = 10^9 \)

fused multiply-accumulates (FMAs)

How can we compute this efficiently?

Ex: \( d = 4 \quad n = 100 \quad r = 10 \)
We can unfold $\mathbf{y}$ into a vector without memory movement.

**Natural descending** element order:

$\mathbf{y}_{(i_1,i_2,...,i_d)}$ is located at array offset

$$y[i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + ... + (i_d - 1)n^{d-1}].$$

**Vectorization:**

Take $\mathbf{y} = \text{vec}(\mathbf{Y})$ by flattening the multiindex

$$\hat{i} = i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + ... + (i_d - 1)n^{d-1}.$$

$$\mathbf{y}_{\hat{i}} = \mathbf{y}_{(i_1,i_2,...,i_d)}$$
Khatri-Rao Product (KRP)

Same as a tensor product (outer product) over matching columns

\[ K = A^{(2)} \bigotimes A^{(1)} \]

means

\[ K_{ij} = A_{i_2 j}^{(2)} A_{i_1 j}^{(1)} \quad \text{where } \hat{i} = i_1 + (i_2 - 1)n. \]
Vectorized Full is Expensive in Memory!

Using \( y = \text{vec}(\mathbf{y}) \)

Ex: \( 10^8 \)

\[ K = A^{(d)} \odot \ldots \odot A^{(2)} \odot A^{(1)} \]

Ex: \( 10^9 \) Ex: \( 10^3 \) each

we can rewrite this

\[ y_{(i_1,i_2,\ldots,i_d)} = \sum_{j=1}^{r} \lambda_j \prod_{k=1}^{d} A^{(k)}_{i_k,j} \]

as matrix-vector multiply.

\( y = K \lambda \)

10^9 FMAs

Ex: \( d = 4 \) n = 100 r = 10

\[ \text{Red} = O(rn^d) \]

Ex: \( 10^9 \)

Matrix-vector multiply (gemv)

1 FMA/move
Rather than unrolling all modes into the row index,

\[ \hat{i} = i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + \ldots + (i_d - 1)n^{d-1} \]

we can split modes between rows and columns.

Modes 1 to \( s \) go into rows and the rest in columns.

1:s \[ \hat{i}_1 = i_1 + (i_2 - 1)n + \ldots (i_s - 1)n^{s-1} \]

s+1:d \[ \hat{i}_2 = i_{s+1} + (i_{s+2} - 1)n + \ldots (i_d - 1)n^{d-s-1} \]

Still no memory movement

\[ Y = \text{mat}_{(1:s)}(Y) \]

\[ Y_{\hat{i}_1 \hat{i}_2} = Y(i_1,i_2,\ldots,i_d) \]
Key Idea: Matricized Reconstruction

Using \( Y = \text{mat}_{(1:s)}(Y) \) with

Left modes \( L = A^{(s)} \odot \cdots \odot A^{(2)} \odot A^{(1)} \)

Right modes \( R = A^{(d)} \odot \cdots \odot A^{(s+2)} \odot A^{(s+1)} \)

changes \( Y_{(i_1,i_2,\ldots,i_d)} = \sum_{j=1}^{r} \lambda_j \prod_{k=1}^{d} A_{i_k j}^{(k)} \)

into matrix multiply \( Y = L \text{ diag}(\lambda) R^T \)

Ex: \( d = 4 \ n = 100 \ r = 10 \)

Green = \( O(rn^{d/2}) \) Ex: \( 10^5 \)

Ex: \( 10^8 \)

Ex: \( 10^5 \) Ex: \( 10^3 \) each

Ex: \( 10^9 \)
Reduced Memory, Higher Computational Intensity

\[ y = K \lambda \]

Ex: \(10^8\)  
Ex: \(10^9\)

Construct:  
\(r n^d = 10^9\) floats

Compute:  
\(r n^d = 10^9\) FMAs

Output:  
\(n^d = 10^8\) floats

\[ Y = L \text{diag}(\lambda) R^T \]

Ex: \(10^8\)  
Ex: \(10^5\)  
Ex: \(10^5\)

Matrix-vector multiply (gemv)  
1 FMA/move

Matrix-matrix multiply (gemm)  
\(O(M^{3/2})\) FMAs/move

Construct:  
\(2 r n^{d/2} = 2 \times 10^5\) floats

Compute: same  
Output: same

Ex: \(d = 4\)  
\(n = 100\)  
\(r = 10\)
Choosing the optimal splitting

Fixed cost: Matrix-matrix multiply.

*Strong implementations of gemm() typically hit 90%+ of peak.*

*Memory movement is hidden.*

Controllable cost: Khatri-Rao product formation.

*Requires at least 1 pass over output memory:*

\[
 s^* = \arg\min_s \left( r \left[ \prod_{k=1}^{s} n_k \right] + r \left[ \prod_{k=s+1}^{d} n_k \right] \right)
\]

\[
 L \quad R
\]
Matricized full uses far less memory!

**Smaller n lets us test higher orders.**

Tests use: \( n = 10 \quad r = 10 \)

<table>
<thead>
<tr>
<th>Order ( d )</th>
<th>Tensor Elts</th>
<th>Vect Mem</th>
<th>Matr Mem</th>
<th>Vect Time</th>
<th>Matr Time</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>( 10^4 )</td>
<td>781 kB</td>
<td>15.6 kB</td>
<td>( 5.8 \times 10^{-4} ) s</td>
<td>( 4.8 \times 10^{-4} ) s</td>
<td>1.2</td>
</tr>
<tr>
<td>5</td>
<td>( 10^5 )</td>
<td>7.63 MB</td>
<td>85.9 kB</td>
<td>( 4.8 \times 10^{-3} ) s</td>
<td>( 7.7 \times 10^{-3} ) s</td>
<td>6.3</td>
</tr>
<tr>
<td>6</td>
<td>( 10^6 )</td>
<td>76.3 MB</td>
<td>156 kB</td>
<td>( 3.9 \times 10^{-2} ) s</td>
<td>( 2.7 \times 10^{-3} ) s</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>( 10^7 )</td>
<td>763 MB</td>
<td>859 kB</td>
<td>( 3.9 \times 10^{-1} ) s</td>
<td>( 2.0 \times 10^{-2} ) s</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>( 10^8 )</td>
<td>7.45 GB</td>
<td>1.53 MB</td>
<td>( 4.7 \times 10^{0} ) s</td>
<td>( 1.9 \times 10^{-1} ) s</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>( 10^9 )</td>
<td><strong>74.5 GB</strong></td>
<td><strong>8.39 MB</strong></td>
<td>FAILED</td>
<td>( 2.2 ) s</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Test were run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.*
Tests use $n = 10 \quad r = 10$

*Run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.*
Now we switch focus to extracting $\mathbf{y} = [\lambda; A^{(1)}, \ldots, A^{(d)}]$ from data tensor $\mathbf{x}$.

- repeat
  - for $k = 1 : d$
    - Fix factor matrices $A^{(1)}, \ldots, A^{(k-1)}, A^{(k+1)}, \ldots, A^{(d)}$.
    - Solve $A^{(k)}$.
  - end for
  - Compute fit.
- until fit change is below threshold

Gauss-Seidel iterative technique cycles through subproblems.
Optimal splitting allowed us to avoid forming a large KRP. Can we use the same technique here?

\[ Z^{(k)} = X^{(k)} \left[ A^{(d)} \odot \cdots \odot A^{(k+1)} \odot A^{(k-1)} \odot \cdots \odot A^{(1)} \right] \text{diag}(\lambda) \]

**Mode-k unfolding**

We just saw that even constructing this may be quite expensive!

- **Construct:**
  \[ r \, n^{d-1} = 10^7 \text{ floats} \]

- **Compute:**
  \[ r \, n^d = 10^9 \text{ FMAs} \]

Matricized tensor times Khatri-Rao product (MTTKRP)
Splitting the Khatri-Rao Product

If $k \leq s$, we can rewrite

$$Z^{(k)} = X^{(k)} \left[ A^{(d)} \odot \cdots \odot A^{(k+1)} \odot A^{(k-1)} \odot \cdots \odot A^{(1)} \right] \text{diag}(\lambda)$$

with an intermediate computation

$$X = \text{mat}_{1:s}(X) \quad W^{(R)} = X \left[ A^{(d)} \odot \cdots \odot A^{(s+1)} \right]$$

Finishing from this form is fast. Details omitted.

$\sim 1$ pass $= rn^d/2 = 10^5$ moves and FMAs

Ex: $d = 4$  $n = 100$  $r = 10$
Connection to Dimension Trees

Each node lists remaining (unreduced) modes.

Each leaf is an MTTKRP on a different mode.

Dimension trees save all intermediate results.

Bulk of speedup comes from keeping first reductions.

A. Phan, P. Tichavsky, A. Cichocki
“Fast Alternating LS Algorithms for High Order CANDECOMP/PARAFAC Tensor Factorizations,” 2013

K.. Rouse, G. Ballard, N. Knight
“Communication Lower Bounds for Matricized Tensor Times Khatri-Rao Product,” 2018

Minimal total construction from optimal splitting.
Memory: $2rn^{d/2}$
Compute: $2rn^d$

Follow-up ops are lower complexity (multi-Tensor-Times-Vector).
Compute: $\sim dln^{d/2}$
Optimal Split Reuse in Update Sequence

<table>
<thead>
<tr>
<th>Update:</th>
<th>Matrices used in efficient MTTKRP formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A^{(1)} \mapsto \hat{A}^{(1)} )</td>
<td>( A^{(2)} )</td>
</tr>
<tr>
<td>( A^{(2)} \mapsto \hat{A}^{(2)} )</td>
<td>( \hat{A}^{(1)} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \hat{A}^{(1)} )</td>
</tr>
<tr>
<td>( A^{(s)} \mapsto \hat{A}^{(s)} )</td>
<td>( \hat{A}^{(1)} )</td>
</tr>
<tr>
<td>( A^{(s+1)} \mapsto \hat{A}^{(s+1)} )</td>
<td>( \hat{A}^{(s+2)} )</td>
</tr>
<tr>
<td>( A^{(s+2)} \mapsto \hat{A}^{(s+2)} )</td>
<td>( \hat{A}^{(s+1)} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \hat{A}^{(s+1)} )</td>
</tr>
<tr>
<td>( A^{(d)} \mapsto \hat{A}^{(d)} )</td>
<td>( \hat{A}^{(s+1)} )</td>
</tr>
</tbody>
</table>

\[
W^{(R)} = X \begin{bmatrix}
    d \\
    k' = s+1
\end{bmatrix}
\]

\[
W^{(L)} = \begin{bmatrix}
    s \\
    k' = 1
\end{bmatrix}^T X
\]

Efficient convergence check uses \( W^{(L)} \) with \( \bigcirc_{k' = s+1:d} \hat{A}^{(k')} \) (Reuse next iteration!)

4/30/2018
J. A. Duersch at SIAM-ALA18
MTTKRP-Sequence Cost Comparison

Ex: \( d = 4 \ n = 100 \ r = 10 \)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>KRP Construction*</th>
<th>Ex (floats)</th>
<th>Post-KRP Computations*</th>
<th>Ex (FMAs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>( drn^{d-1} )</td>
<td>4.00 \times 10^7</td>
<td>( drn^d )</td>
<td>4.00 \times 10^9</td>
</tr>
<tr>
<td>OptSplit</td>
<td>( 2rn^{d/2} + drn^{d-1} )</td>
<td>2.04 \times 10^5</td>
<td>( 2rn^d + drn^{d/2} )</td>
<td>2.00 \times 10^9</td>
</tr>
<tr>
<td>DimTree</td>
<td>( 2rn^{d/2} + 4rn^{d/4} )</td>
<td>2.04 \times 10^5</td>
<td>( 2rn^d + 4rn^{d/2} )</td>
<td>2.00 \times 10^9</td>
</tr>
</tbody>
</table>

*Leading two terms. Assumes perfect splitting.*
Higher Order – Better Speed-up

Tests use $n = 10$  $r = 10$

*Run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.*
Splitting KRPs reduces memory and allows computational reuse.

• Optimal splitting minimizes construction cost for Khatri-Rao products.
• Matricized full() uses less memory which is faster.
  • Vectorized: $r n^d$  Matricized: $2rn^{d/2}$
• MTTKRP sequence (all modes) can be done forming two medium KRPs.
  • Standard: $drn^{d-1}$ floats  OptSplit: $2rn^{d/2}$ floats
• MTTKRP sequence can be computed using two matrix multiplies.
  • Standard: $drn^d$ FMAs  OptSplit: $2rn^d$ FMAs
• Follow-up ops are lower complexity - a simple implementation works well.
• Technique applies to both alternating (CP-ALS) and all-at-once (CP-OPT).

Jed A. Duersch - jaduers@sandia.gov