CS 598 EVS: Tensor Computations
Tensor Computations

Edgar Solomonik

University of Illinois at Urbana-Champaign
Tensors
Tensors

A tensor is a collection of elements

- its dimensions define the size of the collection
- its order is the number of different dimensions
- specifying an index along each tensor mode defines an element of the tensor

A few examples of tensors are

- **Order 0 tensors are scalars**, e.g., \( s \in \mathbb{R} \)
- **Order 1 tensors are vectors**, e.g., \( v \in \mathbb{R}^n \)
- **Order 2 tensors are matrices**, e.g., \( A \in \mathbb{R}^{m \times n} \)
- **An order 3 tensor with dimensions** \( s_1 \times s_2 \times s_3 \) **is denoted as** \( T \in \mathbb{R}^{s_1 \times s_2 \times s_3} \) **with elements** \( t_{ijk} \) **for** \( i \in \{1, \ldots, s_1\} \), \( j \in \{1, \ldots, s_2\} \), \( k \in \{1, \ldots, s_3\} \)
Applications of Tensors

Tensors provide a mathematical formalism for multidimensional data

- tensors arise naturally from discretization of equations with multiple variables

- data that can be tabulated according to multiple parameters is representible by a tensor

- numerical simulations with regular grids represent the solution as tensors (typically order 3) and apply discretized operators, which are structured tensors (of order 6 if the grid is 3D)

- higher-order tensors may arise from higher-order (many-body) interactions, such as in quantum chemistry

- tensor decompositions provide general techniques for approximations and analysis of such tensors
Reshaping Tensors

It's often helpful to use alternative views of the same collection of elements

- **Folding** a tensor yields a higher-order tensor with the same elements
- **Unfolding** a tensor yields a lower-order tensor with the same elements
- In linear algebra, we have the unfolding $v = \text{vec}(A)$, which stacks the columns of $A \in \mathbb{R}^{m \times n}$ to produce $v \in \mathbb{R}^{mn}$
- For a tensor $T \in \mathbb{R}^{s_1 \times s_2 \times s_3}$, $v = \text{vec}(T)$ gives $v \in \mathbb{R}^{s_1 s_2 s_3}$ with
  \[
  v_{i + (j-1)s_1 + (k-1)s_1 s_2} = t_{ijk}
  \]
- A common set of unfoldings is given by matricizations of a tensor, e.g., for order 3,
  \[
  T_{(1)} \in \mathbb{R}^{s_1 \times s_2 s_3}, T_{(2)} \in \mathbb{R}^{s_2 \times s_1 s_3}, \text{ and } T_{(3)} \in \mathbb{R}^{s_3 \times s_1 s_2}
  \]
Tensor Transposition

For tensors of order $\geq 3$, there is more than one way to transpose modes

- A tensor transposition is defined by a permutation $p$ containing elements $\{1, \ldots, d\}$

  $$y_{i_{p_1}, \ldots, i_{p_d}} = x_{i_1, \ldots, i_d}$$

- In this notation, a transposition $A^T$ of matrix $A$ is defined by $p = [2, 1]$ so that

  $$b_{i_2i_1} = a_{i_1i_2}$$

- Tensor transposition is a convenient primitive for manipulating multidimensional arrays and mapping tensor computations to linear algebra

- When elementwise expressions are used in tensor algebra, indices are often carried through to avoid transpositions
Tensor Symmetry

We say a tensor is *symmetric* if $\forall j, k \in \{1, \ldots, d\}$

$$t_{i_1 \ldots i_j \ldots i_k \ldots i_d} = t_{i_1 \ldots i_k \ldots i_j \ldots i_d}$$

A tensor is *antisymmetric* (skew-symmetric) if $\forall j, k \in \{1, \ldots, d\}$

$$t_{i_1 \ldots i_j \ldots i_k \ldots i_d} = (-1) t_{i_1 \ldots i_k \ldots i_j \ldots i_d}$$

A tensor is *partially-symmetric* if such index interchanges are restricted to be within disjoint subsets of $\{1, \ldots, d\}$, e.g., if the subsets for $d = 4$ and $\{1, 2\}$ and $\{3, 4\}$, then

$$t_{ijkl} = t_{jikl} = t_{jilk} = t_{ijkl}$$
Tensor Sparsity

We say a tensor $\mathbf{T}$ is diagonal if for some $v$,

\[
t_{i_1,\ldots,i_d} = \begin{cases} v \\
0 & : \text{otherwise}
\end{cases}
\]

\[
= v_{i_1} \delta_{i_1i_2} \delta_{i_2i_3} \cdots \delta_{i_{d-1}i_d}
\]

- In the literature, such tensors are sometimes also referred to as ‘superdiagonal’
- Generalizes diagonal matrix
- A diagonal tensor is symmetric (and not antisymmetric)

If most of the tensor entries are zeros, the tensor is sparse

- Generalizes notion of sparse matrices
- Sparsity enables computational and memory savings
- We will consider data structures and algorithms for sparse tensor operations later in the course
Tensor Products and Kronecker Products

*Tensor products* can be defined with respect to maps \( f : V_f \rightarrow W_f \) and \( g : V_g \rightarrow W_g \)

\[
h = f \times g \quad \Rightarrow \quad g : (V_f \times V_g) \rightarrow (W_f \times W_g), \quad h(x, y) = f(x)g(y)
\]

Tensors can be used to represent multilinear maps and have a corresponding definition for a tensor product

\[
T = X \times Y \quad \Rightarrow \quad t_{i_1,...,i_m,j_1,...,j_n} = x_{i_1,...,i_m}y_{j_1,...,j_n}
\]

The *Kronecker product* between two matrices \( A \in \mathbb{R}^{m_1 \times m_2}, \ B \in \mathbb{R}^{n_1 \times n_2} \)

\[
C = A \otimes B \quad \Rightarrow \quad c_{i_2+(i_1-1)n_1,j_2+(j_1-1)n_2} = a_{i_1,j_1}b_{i_2,j_2}
\]

corresponds to transposing and unfolding the tensor product
General Tensor Contractions

Given tensor $\mathbf{U}$ of order $s + v$ and $\mathbf{V}$ of order $v + t$, a tensor contraction summing over $v$ modes can be written as

$$w_{i_1...i_s j_1...j_t} = \sum_{k_1...k_v} u_{i_1...i_s k_1...k_v} v_{k_1...k_v j_1...j_t}$$

- This form omits 'Hadamard indices', i.e., indices that appear in both inputs and the output (as with pointwise product, Hadamard product, and batched mat–mul.)
- Other contractions can be mapped to this form after transposition

Unfolding the tensors reduces the tensor contraction to matrix multiplication

- Combine (unfold) consecutive indices in appropriate groups of size $s$, $t$, or $v$
- If all tensor modes are of dimension $n$, obtain matrix–matrix product $C = AB$ where $C \in \mathbb{R}^{n^s \times n^t}$, $A \in \mathbb{R}^{n^s \times n^v}$, and $B \in \mathbb{R}^{n^v \times n^t}$
- Assuming classical matrix multiplication, contraction requires $n^{s+t+v}$ elementwise products and $n^{s+t+v} - n^{s+t}$ additions
Properties of Einsums

Given an elementwise expression containing a product of tensors, the operands commute

- For example $AB \neq BA$, but

\[
\sum_k a_{ik} b_{kj} = \sum_k b_{kj} a_{ik}
\]

- Similarly with multiple terms, we can bring summations out and reorder as needed, e.g., for $ABC$

\[
\sum_k a_{ik} \left( \sum_l b_{kl} c_{lj} \right) = \sum_{kl} c_{lj} b_{kl} a_{ik}
\]

A contraction can be succinctly described by a tensor diagram

- Indices in contractions are only meaningful in so far as they are matched up
- A tensor diagram is defined by a graph with a vertex for each tensor and an edge/leg for each index/mode
- Indices that are not-summed are drawn by pointing the legs/edges into whitespace
Matrix-style Notation for Tensor Contractions

The *tensor times matrix* contraction along the $m$th mode of $\mathbf{U}$ to produce $\mathbf{V}$ is expressed as follows

$$\mathbf{W} = \mathbf{U} \times_m \mathbf{V} \Rightarrow \mathbf{W}(m) = \mathbf{V} \mathbf{U}(m)$$

- $\mathbf{W}(m)$ and $\mathbf{U}(m)$ are unfoldings where the $m$th mode is mapped to be an index into rows of the matrix
- To perform multiple tensor times matrix products, can write, e.g.,

$$\mathbf{W} = \mathbf{U} \times_1 \mathbf{X} \times_2 \mathbf{Y} \times_3 \mathbf{Z} \Rightarrow w_{ijk} = \sum_{pqr} u_{pqr} x_{ip} y_{jq} z_{kr}$$

The *Khatri-Rao product* of two matrices $\mathbf{U} \in \mathbb{R}^{m \times k}$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$ produces $\mathbf{W} \in \mathbb{R}^{mn \times k}$ so that

$$\mathbf{W} = \begin{bmatrix} \mathbf{u}_1 \otimes \mathbf{v}_1 & \cdots & \mathbf{u}_k \otimes \mathbf{v}_k \end{bmatrix}$$

The Khatri-Rao product computes the einsum $\hat{w}_{ijk} = u_{ik} v_{jk}$ then unfolds $\mathbf{W}$ so that $w_{i+(j-1)n,k} = \hat{w}_{ijk}$
A **tensor contraction** multiplies elements of two tensors and computes partial sums to produce a third, in a fashion expressible by pairing up modes of different tensors, defining *einsum* (term stems from Einstein’s summation convention)

<table>
<thead>
<tr>
<th>tensor contraction</th>
<th>einsum</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner product</td>
<td>$w = \sum_i u_i v_i$</td>
<td></td>
</tr>
<tr>
<td>outer product</td>
<td>$w_{ij} = u_i v_{ij}$</td>
<td></td>
</tr>
<tr>
<td>pointwise product</td>
<td>$w_i = u_i v_i$</td>
<td></td>
</tr>
<tr>
<td>Hadamard product</td>
<td>$w_{ij} = u_{ij} v_{ij}$</td>
<td></td>
</tr>
<tr>
<td>matrix multiplication</td>
<td>$w_{ij} = \sum_k u_{ik} v_{kj}$</td>
<td></td>
</tr>
<tr>
<td>batched mat.-mul.</td>
<td>$w_{ijl} = \sum_k u_{ikl} v_{kjl}$</td>
<td></td>
</tr>
<tr>
<td>tensor times matrix</td>
<td>$w_{ilk} = \sum_j u_{ijk} v_{lj}$</td>
<td></td>
</tr>
</tbody>
</table>

The terms ‘contraction’ and ‘einsum’ are also often used when more than two operands are involved
Identities with Kronecker and Khatri-Rao Products

- Matrix multiplication is distributive over the Kronecker product

\[(A \otimes B)(C \otimes D) = AC \otimes BD\]

we can derive this from the einsum expression

\[
\sum_{kl} a_{ik} b_{jl} c_{kp} d_{lq} = \left( \sum_k a_{ik} c_{kp} \right) \left( \sum_l b_{jl} d_{lq} \right)
\]

- For the Khatri-Rao product a similar distributive identity is

\[(A \odot B)^T(C \odot D) = A^T C \ast B^T D\]

where \(\ast\) denotes that Hadamard product, which holds since

\[
\sum_{kl} a_{ki} b_{li} c_{kj} d_{lj} = \left( \sum_k a_{ki} c_{kj} \right) \left( \sum_l b_{li} d_{lj} \right)
\]
The canonical polyadic or CANDECOMP/PARAFAC (CP) decomposition expresses an order \( d \) tensor in terms of \( d \) factor matrices.

For a tensor \( \mathbf{T} \in \mathbb{R}^{n \times n \times n} \), the CP decomposition is defined by matrices \( \mathbf{U} \), \( \mathbf{V} \), and \( \mathbf{W} \) such that

\[
t_{ijk} = \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr}
\]

the columns of \( \mathbf{U} \), \( \mathbf{V} \), and \( \mathbf{W} \) are generally not orthonormal, but may be normalized, so that

\[
t_{ijk} = \sum_{r=1}^{R} \sigma_r u_{ir} v_{jr} w_{kr}
\]

where each \( \sigma_r \geq 0 \) and \( \|u_r\|_2 = \|v_r\|_2 = \|w_r\|_2 = 1 \).

For an order \( N \) tensor, the decomposition generalizes as follows,

\[
t_{i_1...i_d} = \sum_{r=1}^{R} \prod_{j=1}^{d} u_{i_j r}^{(j)}
\]

Its rank is generally bounded by \( R \leq n^{d-1} \).
CP Decomposition Basics

- The CP decomposition is useful in a variety of contexts
  - *If an exact decomposition with* $R \ll n^{d-1}$ *is expected to exist*
  - *If an approximate decomposition with* $R \ll n^{d-1}$ *is expected to exist*
  - *If the factor matrices from an approximate decomposition with* $R = O(1)$ *are expected to contain information about the tensor data*
  - *CP a widely used tool, appearing in many domains of science and data analysis*

- Basic properties and methods
  - *Uniqueness (modulo normalization) is dependent on rank*
  - *Finding the CP rank of a tensor or computing the CP decomposition is NP-hard (even with* $R = 1$*)*
  - *Typical rank of tensors (likely rank of a random tensor) is generally less than the maximal possible rank*
  - *CP approximation as a nonlinear least squares (NLS) problem and NLS methods can be applied in a black-box fashion, but structure of decomposition motivates alternating least-squares (ALS) optimization*
Alternating Least Squares Algorithm

- The standard approach for finding an approximate or exact CP decomposition of a tensor is the **alternating least squares (ALS) algorithm**
- Consider rank \( R \) decomposition of a tensor \( \mathbf{T} \in \mathbb{R}^{n \times n \times n} \) over \( \mathbb{R} \)
- A sweep takes as input \( [[\mathbf{U}^{(k)}, \mathbf{V}^{(k)}, \mathbf{W}^{(k)}]] \) solves 3 quadratic optimization problems to obtain \( [[\mathbf{U}^{(k+1)}, \mathbf{V}^{(k+1)}, \mathbf{W}^{(k+1)}]] \), updating each factor matrix in sequence, typically via the normal equations:

\[
(\mathbf{V}^{(k)T} \mathbf{V}^{(k)} \ast \mathbf{W}^{(k)T} \mathbf{W}^{(k)}) \mathbf{U}^{(k+1)} = \mathbf{T}_1(\mathbf{V}^{(k)T} \circ \mathbf{W}^{(k)})
\]

\[
(\mathbf{U}^{(k+1)T} \mathbf{U}^{(k+1)} \ast \mathbf{W}^{(k)T} \mathbf{W}^{(k)}) \mathbf{V}^{(k+1)} = \mathbf{T}_2(\mathbf{U}^{(k+1)T} \circ \mathbf{W}^{(k)})
\]

\[
(\mathbf{U}^{(k+1)T} \mathbf{U}^{(k+1)} \ast \mathbf{V}^{(k+1)T} \mathbf{V}^{(k+1)}) \mathbf{W}^{(k+1)} = \mathbf{T}_3(\mathbf{U}^{(k+1)T} \circ \mathbf{V}^{(k+1)})
\]

- Residual decreases monotonically, since the subproblems in each subset of \( nR \) variables are quadratic
- Forming the linear equations has cost \( O(dnR^2) \) while forming the right-hand-sides requires an MTTKRP with cost \( O(n^dR) \)
The **Tucker decomposition** expresses an order $d$ tensor via a smaller order $d$ core tensor and $d$ factor matrices.

- For a tensor $\mathbf{T} \in \mathbb{R}^{n \times n \times n}$, the Tucker decomposition is defined by core tensor $\mathbf{Z} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ and factor matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ with orthonormal columns, such that

$$
t_{ijk} = \sum_{p=1}^{R_1} \sum_{q=1}^{R_2} \sum_{r=1}^{R_3} z_{pqr} u_{ip} v_{jq} w_{kr}
$$

- For general tensor order, the Tucker decomposition is defined as

$$
t_{i_1 \ldots i_d} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} z_{r_1 \ldots r_d} \prod_{j=1}^{d} u_{i_j r_j}^{(j)}
$$

which can also be expressed as

$$
\mathbf{T} = \mathbf{Z} \times_1 \mathbf{U}^{(1)} \ldots \times_d \mathbf{U}^{(d)}
$$

- The Tucker ranks, $(R_1, R_2, R_3)$ are each bounded by the respective tensor dimensions, in this case, $R_1, R_2, R_3 \leq n$
- In relation to CP, Tucker is formed by taking all combinations of tensor products between columns of factor matrices, while CP takes only disjoint products
Tucker Decomposition Basics

- The Tucker decomposition is used in many of the same contexts as CP
  - If an exact decomposition with each $R_j < n$ is expected to exist
  - If an approximate decomposition with $R_j < n$ is expected to exist
  - If the factor matrices from an approximate decomposition with $R = O(1)$ are expected to contain information about the tensor data
  - Tucker is most often used for data compression and appears less often than CP in theoretical analysis

- Basic properties and methods
  - The Tucker decomposition is not unique (can pass transformations between core tensor and factor matrices, which also permit their orthogonalization)
  - Finding the best Tucker approximation is NP-hard (for $R = 1$, CP = Tucker)
  - If an exact decomposition exists, it can be computed by high-order SVD (HOSVD), which performs $d$ SVDs on unfoldings
    - HOSVD obtains a good approximation with cost $O(n^{d+1})$ (reducible to $O(n^d R)$ via randomized SVD or QR with column pivoting)
  - Accuracy can be improved by iterative nonlinear optimization methods, such as high-order orthogonal iteration (HOOI)
Tensor Train Decomposition

- The tensor train decomposition expresses an order $d$ tensor as a chain of products of order 2 or order 3 tensors
  
  - For an order 4 tensor, we can express the tensor train decomposition as

  \[
  t_{ijkl} = \sum_{p,q,r} u_{ip} v_{pq} w_{qr} z_{rl}
  \]

- More generally, the Tucker decomposition is defined as follows,

  \[
  t_{i_1 \ldots i_d} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{d-1}=1}^{R_{d-1}} u_{i_1 r_1}^{(1)} \left( \prod_{j=2}^{d-1} u_{r_{j-1} i_j r_j}^{(j)} \right) u_{r_{d-1} i_d}^{(d)}
  \]

- In physics literature, it is known as a matrix product state (MPS), as we can write it in the form,

  \[
  t_{i_1 \ldots i_d} = \langle u_{i_1}^{(1)}, U_{i_2}^{(2)} \cdots U_{i_{d-1}}^{(d-1)} u_{i_d}^{(d)} \rangle
  \]

- For an equidimensional tensor, the ranks are bounded as $R_j \leq \min(n^j, n^{d-j})$
Tensor Train Decomposition Basics

- Tensor train has applications in quantum simulation and in numerical PDEs
  - It is useful whenever the tensor is low-rank or approximately low-rank, i.e., $R_j R_{j+1} < n^{d-1}$ for all $j < d - 1$
  - MPS (tensor train) and extensions are widely used to approximate quantum systems with $\Theta(d)$ particles/spins
  - Often the MPS is optimized relative to an implicit operator (often of a similar form, referred to as the matrix product operator (MPO))
  - Operators and solutions to some standard numerical PDEs admit tensor-train approximations that yield exponential compression

- Basic properties and methods
  - The tensor train decomposition is not unique (can pass transformations, permitting orthogonalization into canonical forms)
  - Approximation with tensor train is NP hard (for $R = 1$, CP = Tucker = TT)
  - If an exact decomposition exists, it can be computed by tensor train SVD (TTSVD), which performs $d - 1$ SVDs
  - TTSVD can be done with the cost $O(n^{d+1})$ or $O(n^d R)$ with faster low-rank SVD
  - Iterative (alternating) optimization is generally used when optimizing tensor train relative to an implicit operator or to refine TTSVD
We can compare the aforementioned decomposition for an order $d$ tensor with all dimensions equal to $n$ and all decomposition ranks equal to $R$.

<table>
<thead>
<tr>
<th>decomposition</th>
<th>CP</th>
<th>Tucker</th>
<th>tensor train</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>$dnR$</td>
<td>$dnR + R^d$</td>
<td>$2nR + (d - 2)nR^2$</td>
</tr>
<tr>
<td>uniqueness</td>
<td>if $R \leq (3n - 2)/2$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>orthogonalizability</td>
<td>none</td>
<td>partial</td>
<td>partial</td>
</tr>
<tr>
<td>exact decomposition</td>
<td>NP hard</td>
<td>$O(n^{d+1})$</td>
<td>$O(n^{d+1})$</td>
</tr>
<tr>
<td>approximation</td>
<td>NP hard</td>
<td>NP hard</td>
<td>NP hard</td>
</tr>
<tr>
<td>typical method</td>
<td>ALS</td>
<td>HOSVD</td>
<td>TT-ALS (implicit)</td>
</tr>
</tbody>
</table>
Sparse Tensor Formats

- The overhead of transposition, and non-standard nature of the arising sparse matrix products, motivates sparse data structures for tensors that are suitable for tensor contractions of interest
  - Particularly important, especially for tensor decomposition, are MTTKRP (suffices to CP ALS) and TTMc (suffices for HOOI)
  - TTM is also prevalent, but is a less attractive primitive in the sparse case than MTTKRP and TTMc, as these yield dense, low-order outputs, while the output of TTM can be sparse and larger than the starting tensor
- The *compressed sparse fiber (CSF)* format provides an effective representation for sparse tensors
  - CSF can be visualized as a tree (diagram taken from original CSF paper, by Shaden Smith and George Karpis, IA^3, 2015)
Operations in Compressed Format

- CSF permits efficient execution of important sparse tensor kernels
  - Analogous to CSR format, which enables efficient implementation of the sparse matrix vector product
  - where row[i] stores a list of column indices and nonzeros in the i\textsuperscript{th} row of A

```python
for i in range(n):
    for (a_ij, j) in row[i]:
        y[i] += a_ij * x[j]
```

- In CSF format, a multilinear function evaluation $f^{(T)}(x, y) = T_1(x \odot y)$ can be implemented as

```python
for (i, T_i) in T_CSF:
    for (j, T_ij) in T_i:
        for (k, t_ijk) in T_ij:
            z[i] += t_ijk * x[j] * y[k]
```
MTTKRP and CSF pose additional implementation opportunities and challenges

- MTTKP $u_{ir} = \sum_{j,k} t_{ijk} v_{jr} w_{kr}$ can be implemented by adding a loop over $r$ to our code for $f(T)$, but would then require $3mr$ operations if $m$ is the number of nonzeros in $T$, can reduce to $2mr$ by amortization

```python
for (i,T_i) in T_CSF:
    for (j,T_ij) in T_i:
        for r in range(R):
            f_ij = 0
            for (k,t_ijk) in T_ij:
                f_ij += t_ijk * w[k,r]
                u[i,r] = f_ij * v[j,r]
```

- However, this amortization is harder (requires storage or iteration overheads) if the index $i$ is a leaf node in the CSF tree

- Similar challenges in achieving good reuse and obtaining good arithmetic intensity arise in implementation of other kernels, such as TTMc