CS 598 EVS: Tensor Computations

Basics of Tensor Computations

Edgar Solomonik

University of Illinois at Urbana-Champaign

Tensors

A *tensor* is a collection of elements

A few examples of tensors are

Reshaping Tensors

Its often helpful to use alternative views of the same collection of elements

Matrices and Tensors as Operators and Multilinear Forms

- What is a matrix?

- What is a tensor?

Tensor Transposition

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

Tensor Symmetry

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

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

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

Tensor Sparsity

We say a tensor $\mathcal{T}$ is *diagonal* if for some v, If most of the tensor entries are

zeros, the tensor is *sparse*

Tensor Products and Kronecker Products

Tensor products can be defined with respect to maps $f : V_f \to W_f$ and $g : V_g \to W_g$

$A \in R^{m_A \times n_A}$ $B \in R^{m_B \times n_B}$

A

B

$A \times B = T \in R^{m_A \times n_A \times m_B \times n_B}$

$a_{ij} b_{kl} = t_{ijkl}$

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

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

$A \otimes B = C$

$b_{ij} a_{kl} = c_{i+km_1, j+lm_2}$

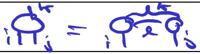
$$A \otimes B = \begin{bmatrix} \underline{a_{11}}\,\underline{B} & a_{12}B & \cdots \\ a_{21}B & & \\ \vdots & & \end{bmatrix}$$

$$T \in \mathbb{R}^{\alpha_i \times \alpha_j \cdots}$$

$$t_{ijk\underline{l}} \rightarrow T[\,\underline{l} + (k-1)\alpha_l + (j-1)\cdot\alpha_l\alpha_k \cdots]$$

Tensor Contractions

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)

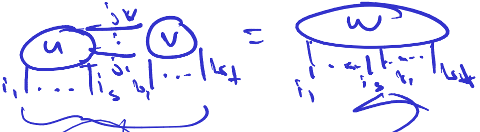
tensor contraction	*einsum*	*diagram*
inner product	$c = a_i b_i$	
outer product	$c_{ij} = a_i b_j$	
pointwise product	$c_i = a_i b_i$	
Hadamard product	$c_{ij} = a_{ij} b_{ij}$	
matrix multiplication	$c_{ij} = a_{ik} b_{kj}$	
batched mat.-mul.	$c_{ijk} = a_{ilk} b_{ljk}$	
tensor times matrix	$c_{ijk} = a_{ijl} b_{lk}$	

$C = A * B$

The terms 'contraction' and 'einsum' are also often used when more than two operands are involved

General Tensor Contractions

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



$$W_{i_1 \dots i_s k_1 \dots k_t} = \sum_{j_1 \dots j_v} u_{i_1 \dots i_s j_1 \dots j_v} v_{j_1 \dots j_v k_1 \dots k_t}$$

Unfolding the tensors reduces the tensor contraction to matrix multiplication

$$\bar{W}_{IK} = \sum_J \bar{u}_{IJ} \bar{v}_{JK} \qquad \bar{W} = \bar{u}\bar{v}$$

Properties of Einsums

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

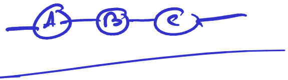
$$AB = BA$$

ABC

$$c_{ij} = \sum_k a_{ik} b_{kj} = \sum_k b_{kj} a_{ik}$$

$$d_{ij} = \sum_l a_{il} \left(\sum_k b_{lk} c_{kj} \right) = \sum_{lk} a_{il} b_{lk} c_{kj}$$

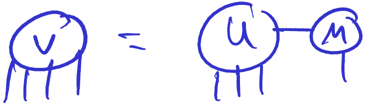
A contraction can be succinctly described by a *tensor diagram*




```
for l
  for k
    for i
      for j
        d_ij = d_ij + a_il b_lk c_kj
```

Matrix-style Notation for Tensor Contractions

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

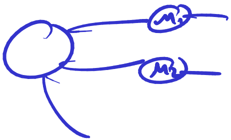


$$V_{(j)} = M U_{(j)} \in \mathbb{R}^{n_j \times (n_1, \ldots, n_N)}$$

$$\mathcal{V} = \mathcal{U} \times_k M \implies v_{i_1 \ldots i_N} = \sum_j u_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_N} m_{i_k j}$$

The *Khatri-Rao product* of two matrices $\boldsymbol{U} \in \mathbb{R}^{m \times k}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times k}$ products $\boldsymbol{W} \in \mathbb{R}^{mn \times k}$ so that $w_{[ij]k} = u_{ik} v_{jk}$

$$\begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix} \odot \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix} = \begin{bmatrix} u_1 \otimes v_1 & u_2 \otimes v_2 & \cdots & u_k \otimes v_k \end{bmatrix}$$


M1
M2


T
x1
M1
x2
M2

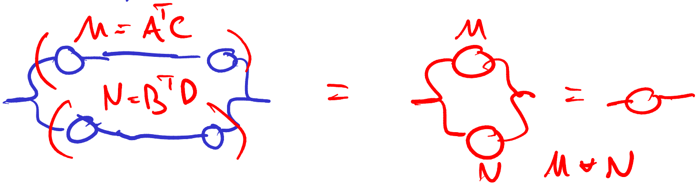
Identities with Kronecker and Khatri-Rao Products

- Matrix multiplication is distributive over the Kronecker product

$$\underbrace{(A\otimes B)(C\otimes D)}_{O(n^6)} = \underbrace{\underbrace{(AC)\otimes(BD)}}_{O(n^4)} \qquad O(n^3)$$

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

$$(A\odot B)^T(C\odot D) = (A^TC) * (B^TD)$$



$$\sum_{i,j} a_{ik} b_{jk} c_{il} d_{jl} = w_{kl}$$

$$\left(\sum_i a_{ik} c_{il}\right) \cdot \left(\sum_j b_{jk} d_{jl}\right) = (A^T C) * (B^T D)$$

Multilinear Tensor Operations

Given an order d tensor $\mathcal{T}$, define multilinear function $\boldsymbol{x}^{(1)} = \boldsymbol{f}^{(\mathcal{T})}(\boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(d)})$

$$x^{(1)} = f^{(T)}(x^{(2)}, \ldots, x^{(d)})$$

$$f^{(T)} \in \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{(d-1)\text{-times}} \to \mathbb{R}^n$$

$$x^{(1)}_{i_1} = \underbrace{\sum_{i_2 \ldots i_d}} T_{i_1 \ldots i_d} \, x^{(2)}_{i_2} \cdots x^{(d)}_{i_d}$$

Batched Multilinear Operations

The multilinear map $f^{(\mathcal{T})}$ is frequently used in tensor computations

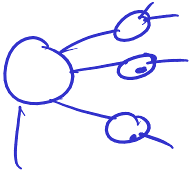
MTTKRP – matricized tensor times Khatri-Rao product

$$A^{(n)} = T_{(1)}\left(A^{(1)} \odot \cdots \odot A^{(n-1)} \odot A^{(n+1)} \odot \cdots \odot A^{(N)}\right)$$

$$a_{ir} = T_{ijk}\, b_{jr}\, c_{kr}$$

$$A = T_{(1)}\left(B \odot C\right)$$

$$\begin{bmatrix} a^{(1)} & \cdots & a^{(R)} \end{bmatrix} = \begin{bmatrix} f_{\mathcal{T}}(b^{(1)}, c^{(1)}) & \cdots & f_{\mathcal{T}}(b^{(R)}, c^{(R)}) \end{bmatrix}$$



TTMc

tensor-times-matrix chain

$$Z = T \times_1 A^{(1)} \times_2 A^{(2)} \cdots \times_{d-1} A^{(d-1)} \qquad A^{(1)} = \begin{bmatrix} a^{(1,1)} & \cdots & a^{(1,\ell)} \end{bmatrix}^T$$

$$Z_{i_1 \cdots i_d} = \sum_{j_1 \cdots j_d} T_{j_1 \cdots j_d}\, a^{(1)}_{i_1 j_1} \cdots a^{(d-1)}_{i_{d-1} j_{d-1}}$$

$$Z^{(i_1 \cdots i_{d-1})}_{i_d} =$$

$$Z^{(i_1 \cdots i_{d-1})} = f_T\left(a^{(1,i_1)} \cdots\right)$$

Tensor Norm and Conditioning of Multilinear Functions

We can define elementwise and operator norms for a tensor $\mathcal{T}$

Conditioning of Multilinear Functions

Evaluation of the multilinear map is typically ill-posed for worst case inputs

Well-conditioned Tensors

For equidimensional tensors (all modes of same size), some small ideally conditioned tensors exist

Ill-conditioned Tensors

For $n \notin \{2, 4, 8\}$ given any $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times n\times n}$, $\inf_{\boldsymbol{x},\boldsymbol{y}\in\mathbb{S}^{n-1}} \|\boldsymbol{f}^{(\boldsymbol{\mathcal{T}})}(\boldsymbol{x},\boldsymbol{y})\|_2 = 0$

CP Decomposition

- The *canonical polyadic or CANDECOMP/PARAFAC (CP) decomposition* expresses an order d tensor in terms of d factor matrices

CP Decomposition Basics

- The CP decomposition is useful in a variety of contexts

- Basic properties and methods

Tucker Decomposition

- The *Tucker decomposition* expresses an order d tensor via a smaller order d core tensor and d factor matrices

Tucker Decomposition Basics

- The Tucker decomposition is used in many of the same contexts as CP

- Basic properties and methods

Tensor Train Decomposition

- The *tensor train decomposition* expresses an order d tensor as a chain of products of order 2 or order 3 tensors

Tensor Train Decomposition Basics

- Tensor train has applications in quantum simulation and in numerical PDEs

- Basic properties and methods

Summary of Tensor Decomposition Basics

We can compare the aforementioned decomposition for an order d tensor with all dimensions equal to n and all decomposition ranks equal to R

decomposition	CP	Tucker	tensor train
size			
uniqueness			
orthogonalizability			
exact decomposition			
approximation			
typical method			

Bilinear Algorithms

A bilinear algorithm (V. Pan, 1984) $\Lambda = (\boldsymbol{F}^{(A)}, \boldsymbol{F}^{(B)}, \boldsymbol{F}^{(C)})$ computes

$$\boldsymbol{c} = \boldsymbol{F}^{(C)}[(\boldsymbol{F}^{(A)T}\boldsymbol{a}) \odot (\boldsymbol{F}^{(B)T}\boldsymbol{b})],$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are inputs and $\odot$ is the Hadamard (pointwise) product.

Bilinear Algorithms as Tensor Factorizations

- A bilinear algorithm corresponds to a CP tensor decomposition
- For multiplication of $n \times n$ matrices, we can define a *matrix multiplication tensor* and consider algorithms with various bilinear rank

Strassen's Algorithm

Strassen's algorithm $\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$

$$\begin{aligned}
\boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &= \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &= \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}$$

By performing the nested calls recursively, Strassen's algorithm achieves cost,