CS 598 EVS: Tensor Computations

Bilinear Algorithms

Edgar Solomonik

University of Illinois, Urbana-Champaign

Bilinear Problems

- A number of basic numerical problems can be thought of as bilinear functions associated with particular order 3 tensors
  - *matrix multiplication*
  - *discrete convolution*
  - *symmetric tensor contractions*
- These problems admit nontrivial fast *bilinear algorithms*, which correspond to low-rank CP decompositions of the tensors
  - *Strassen's $O(n^{\log_2(7)})$ algorithm for matrix multiplication as well as all other subcubic matrix multiplication*
  - *The discrete Fourier transform (DFT), Toom-Cook, and Winograd algorithms for convolution are also examples of bilinear algorithms*
- *We will review fast bilinear algorithms for all of these approaches, using 0-based indexing when discussing convolution*

Bilinear Problems

- A bilinear problem for any inputs $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^k$ computes $\boldsymbol{c} \in \mathbb{R}^m$ as defined by a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{m \times n \times k}$

$$c_i = \sum_{j,k} t_{ijk} a_j b_k \quad \Leftrightarrow \quad \boldsymbol{c} = \boldsymbol{f}^{(\boldsymbol{\mathcal{T}})}(\boldsymbol{a}, \boldsymbol{b})$$

- Variants of discrete convolutions (linear convolution, correlation, cyclic convolution) provide simple examples of $\boldsymbol{\mathcal{T}}$
  - *Linear convolution*

$$t_{ijk} = \begin{cases} 1 : k + i - j = 0 \\ 0 : \textit{otherwise} \end{cases} \quad \Rightarrow \quad c_i = \sum_{j,k} t_{ijk} a_j b_k = \sum_{j=\max(0,i-n+1)}^{\min(i,n-1)} a_j b_{i-j}$$

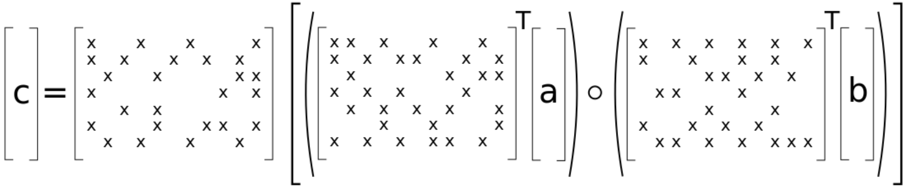
  - *Correlation obtained by transposing the first and last mode of the linear convolution tensor*
  - *Cyclic convolution has* $t_{ijk} = 1$ *if and only if* $k + i - j = 0 \pmod{n}$

Bilinear Algorithms

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

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

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



Bilinear Algorithms as Tensor Factorizations

- A bilinear algorithm corresponds to a CP tensor decomposition

$$
\begin{aligned}
c_i &= \sum_{r=1}^{R} f_{ir}^{(C)} \left(\sum_j f_{jr}^{(A)} a_j \right) \left(\sum_k f_{kr}^{(B)} b_k \right) \\
&= \sum_j \sum_k \left(\sum_{r=1}^{R} f_{ir}^{(C)} f_{jr}^{(A)} f_{kr}^{(B)} \right) a_j b_k \\
&= \sum_j \sum_k t_{ijk} a_j b_k \quad \textit{where} \quad t_{ijk} = \sum_{r=1}^{R} f_{ir}^{(C)} f_{jr}^{(A)} f_{kr}^{(B)}
\end{aligned}
$$

- For multiplication of $n \times n$ matrices, we can define a *matrix multiplication tensor* and consider algorithms with various bilinear rank
  - $\boldsymbol{T}$ *is* $n^2 \times n^2 \times n^2$
  - *Classical algorithm has rank* $R = n^3$
  - *Strassen's algorithm has rank* $R \approx n^{\log_2(7)}$

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,

$$T(n) = 7T(n/2) + O(n^2) = O(7^{\log_2 n}) = O(n^{\log_2 7})$$

For recent developments in algorithms for fast matrix multiplication, see "Flip Graphs for Matrix Multiplication", Kauers and Moosbauer (2023).

Fast Bilinear Algorithms for Convolution

- Linear convolution corresponds to polynomial multiplication
  - *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be coefficients of degree $n-1$ polynomial p and degree $k-1$ polynomial q then*

$$(p \cdot q)(x) = \sum_{i=0}^{n+k-1} c_i x^i \quad \textit{where} \quad c_i = \sum_{j=\max(0,i-n+1)}^{\min(i,n-1)} a_j b_{i-j}$$

  - *This view motivates algorithms based on polynomial interpolation*
- The *Toom-Cook* convolution algorithm computes the coefficients of $p \cdot q$ by computing $(p \cdot q)(x_i)$ for $i \in \{1, \dots, n+k-1\}$ and interpolates
  - *Let $\boldsymbol{V}_r$ be a $(n+k-1)$-by-r Vandermonde matrix based on the nodes $\boldsymbol{x}$, so that $\boldsymbol{V}_n\boldsymbol{a} = [p(x_1), \cdots, p(x_{n+k-1})]^T$, etc.*
  - *Then to evaluate p and q at $\boldsymbol{x}$ and interpolate, we compute*

$$\boldsymbol{c} = \boldsymbol{V}_{n+k-1}^{-1}((\boldsymbol{V}_n\boldsymbol{a}) \odot (\boldsymbol{V}_k\boldsymbol{b}))$$

    which is a bilinear algorithm

Toom-Cook Convolution and the Fourier Transform

- Vandermonde matrices are ill-conditioned with real nodes, but can be perfectly conditioned with complex nodes
  - *The condition number of a Vandermonde matrix with real nodes is exponential in its dimension*
  - *Choosing the nodes $\boldsymbol{x}$ to be the complex roots of unity gives the discrete Fourier transform (DFT) matrix $\boldsymbol{D}^{(n)}$, $d_{jk}^{(n)} = \omega_n^{jk}$ where $\omega_n = e^{2i\pi/n}$*
  - *Modulo normalization DFT matrix is orthogonal and symmetric (not Hermitian)*
- The *fast Fourier transform (FFT)* can be used to perform products with the DFT matrix in $O(n \log n)$ time *Taking $\tilde{\boldsymbol{D}}^{(n)}$ to be the $n_1 \times n_2$ (for $n = n_1 n_2$) leading minor of $\boldsymbol{D}_n$ we can compute $\boldsymbol{y} = \boldsymbol{D}^{(n)}\boldsymbol{x}$ via the split-radix-n_1 FFT,*

$$y_k = \sum_{i=0}^{n-1} x_i \omega_n^{ik} = \sum_{i=0}^{n/2-1} x_{2i}\omega_{n/2}^{ik} + \omega_n^k \sum_{i=0}^{n/2-1} x_{2i+1}\omega_{n/2}^{ik}$$

$$y_{(kn_1+t)} = \sum_{s=0}^{n_1-1} \omega_{n_1}^{st}\left[\omega_n^{sk} \sum_{i=0}^{n_2-1} x_{(in_1+s)}\omega_{n_2}^{ik}\right] \Leftrightarrow \boldsymbol{Y} = ([\tilde{\boldsymbol{D}}^{(n)} \odot (\boldsymbol{D}^{(n_2)}\boldsymbol{A})]\boldsymbol{D}^{(n_1)})^T$$

Cyclic Convolution via DFT

- For linear convolution $\boldsymbol{D}^{(n+k-1)}$ is used, for cyclic convolution $\boldsymbol{D}^{(n)}$ suffices
  - *Expanding the bilinear algorithm,* $\boldsymbol{y} = {\boldsymbol{D}^{(n)}}^{-1}((\boldsymbol{D}^{(n)}\boldsymbol{f}) \odot (\boldsymbol{D}^{(n)}\boldsymbol{g}))$*, we obtain*

$$y_k = \frac{1}{n}\sum_{i=0}^{n-1}\omega_{(n)}^{-ki}\Big(\sum_{j=0}^{n-1}\omega_{(n)}^{ij}f_j\Big)\Big(\sum_{t=0}^{n-1}\omega_{(n)}^{it}g_t\Big) = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{t=0}^{n-1}\omega_{(n)}^{(j+t-k)i}f_j g_t$$

  - *It suffices to observe that for any fixed* $u = j + t - k \neq 0$ *or* $\neq n$*, the outer summation yields a zero result, since the geometric sum simplifies to*

$$\sum_{i=0}^{n-1}\omega_{(n)}^{ui} = (1 - (\omega_{(n)}^{u})^n)/(1 - \omega_{(n)}^{u}) = 0$$

- The DFT also arises in the eigendecomposition of a circulant matrix
  - *The cyclic convolution is defined by the matrix-vector product* $\boldsymbol{y} = \boldsymbol{C}_{\langle \boldsymbol{a} \rangle}\boldsymbol{b}$ *where*

$$\boldsymbol{C}_{\langle \boldsymbol{a} \rangle} = \begin{bmatrix} a_0 & \cdots & a_1 \\ \vdots & \ddots & \vdots \\ a_{n-1} & \cdots & a_0 \end{bmatrix}$$

  - *The eigenvalue decomposition of this matrix is* $\boldsymbol{C}_{\langle \boldsymbol{a} \rangle} = {\boldsymbol{D}^{(n)}}^{-1}\operatorname{diag}(\boldsymbol{D}^{(n)}\boldsymbol{a})\boldsymbol{D}^{(n)}$

Symmetric Tensor Contractions

- Bilinear algorithms can also be used to accelerate tensor contractions for tensors with symmetry
  - *Recall a symmetric tensor is defined by e.g.,* $t_{ijk} = t_{ikj} = t_{kij} = t_{jki} = t_{jik} = t_{kji}$
  - *Tensors can also have skew-symmetry (also known as antisymmetry, permutations have* $+/-$ *signs), partial symmetry (only some modes are permutable), or group symmetry (blocks are zero if indices satisfy modular equation)*
  - *The simplest example of a symmetric tensor contraction is*

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \quad \text{where } \boldsymbol{A} = \boldsymbol{A}^T$$

    it is not obvious how to leverage symmetry to reduce cost of this contraction
- Bilinear algorithms for symmetric tensor contractions exist with lower rank than their nonsymmetric counterparts
  - *Symmetric matrix-vector product can be done with* $n(n+1)/2$ *multiplications*
  - *Cost of contractions of partially symmetric tensors reduced via this technique*

Symmetric Matrix Vector Product

- Consider computing $\boldsymbol{c} = \boldsymbol{A}\boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{A}^T$
  - *Typically requires* n^2 *multiplications since* $a_{ij}b_j \neq a_{ji}b_i$ *and* $n^2 - n$ *additions*
  - *Instead can compute*

$$v_i = \sum_{j=1}^{i-1} u_{ij} + \sum_{j=i+1}^{n} u_{ji} \quad \text{where} \quad u_{ij} = a_{ij}(b_i + b_j)$$

    using $n(n-1)/2$ *multiplications (since we only need* u_{ij} *for* $i > j$*) and about* $3n^2/2$ *additions, then*

$$c_i = (2a_{ii} - \sum_{j=1}^{n} a_{ij})b_i + v_i$$

    using n *more multiplications and* n^2 *additions*
  - *Beneficial when multiplying elements of* $\boldsymbol{A}$ *and* $\boldsymbol{b}$ *costs more than addition*
  - *This technique yields a bilinear algorithm with rank* $n(n+1)/2$

Partially-Symmetric Tensor Times Matrix (TTM)

- Can use symmetric mat-vec algorithm to accelerate TTM with partially symmetric tensor from $2n^4$ operations to $(3/2)n^4 + O(n^3)$
  - *Given* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n\times n\times n}$ *with symmetry* $a_{ijk} = a_{jik}$ *and* $\boldsymbol{B} \in \mathbb{R}^{n\times n}$*, we compute*
    $$c_{ikl} = \sum_j a_{ijk} b_{jl}$$
  - *We can think of this as a set of symmetric matrix-vector products*
    $$\boldsymbol{c}^{(k,l)} = \boldsymbol{A}^{(k)} \boldsymbol{b}^{(l)}$$
    and apply the fast bilinear algorithm
    $$v_{ikl} = \sum_{j=1}^{i-1} u_{ijkl} + \sum_{j=i+1}^{n} u_{ijkl} \quad \text{where} \quad u_{ijkl} = a_{ijk}(b_{il} + b_{jl})$$
    $$c_{ikl} = (2a_{iik} - \sum_{j=1}^{n} a_{ijk}) b_{il} + v_{ikl}$$
    using about $n^4/2$ *multiplications and* $n^4 + O(n^3)$ *additions (need only* n^3 *distinct sums of elements of* $\boldsymbol{B}$*) to compute* $\boldsymbol{\mathcal{V}}$*, then* $O(n^3)$ *operations to get* $\boldsymbol{\mathcal{C}}$ *from* $\boldsymbol{\mathcal{V}}$

Computing Symmetric Matrices

- Output symmetry can also be used to reduced cost, for example when computing a symmetrized outer product $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^T + \boldsymbol{b}\boldsymbol{a}^T$
  - *$\boldsymbol{C} = \boldsymbol{C}^T$ so suffices to compute c_{ij} for $i \geqslant j$, $c_{ij} = a_i b_j + a_j b_i$*
  - *To reduce number of products by a factor of 2, can instead compute*

$$c_{ij} = (a_i + a_j)(b_i + b_j) - v_i - v_j \quad \textit{where} \quad v_i = a_i b_i$$

- To symmetrize product of two symmetric matrices, can compute anticommutator, $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}$
  - *Each matrix can be represented with $n(n+1)/2$ elements, but products all n^3 products $a_{ik}b_{kj}$ are distinct (so typically cost is $2n^3$)*
  - *Cost can be reduced to $n^3/6 + O(n^2)$ products by amortizing terms in*

$$\begin{aligned} c_{ij} = &\sum_k (a_{ij} + a_{ik} + a_{jk})(b_{ij} + b_{ik} + b_{jk}) - n a_{ij} b_{ij} \\ &- \Big(\sum_k a_{ik} + a_{jk}\Big) b_{ij} - a_{ij}\Big(\sum_k b_{ik} + b_{jk}\Big) - \sum_k a_{ik} b_{ik} - \sum_k a_{jk} b_{jk} \end{aligned}$$

General Symmetric Tensor Contractions

- We can now consider the cost of a symmetrized contraction over v indices of symmetric tensors $\boldsymbol{\mathcal{A}}$ (of order $s+v$) and $\boldsymbol{\mathcal{B}}$ (of order $v+t$)

$$c_{i'_1\dots i'_s,j'_1\dots j'_t} = \sum_{\{i_1\dots i_s,j_1\dots j_t\}\in\Pi(i'_1\dots i'_s,j'_1\dots j'_t)} \sum_{k_1\dots k_v} a_{i_1\dots i_s,k_1\dots k_v} b_{k_1\dots k_v,j_1\dots j_t}$$

  where Π *gives all distinct partitions of the* $s+t$ *indices into two subsets of size* s *and* t*, e.g.,*

$$\Pi(i_1, j_1 j_2) = \{\{i_1, j_1 j_2\}, \{j_1, i_1 j_2\}, \{j_2, i_1 j_1\}\}$$

- Such tensor contractions can be done using $n^{s+t+v}/(s+t+v)! + O(n^{s+t+v-1})$ products
  - *General algorithm looks similar to anticommutator matrix product*
  - *After multiplying subsets of operands, unneeded terms are all computable with* $O(n^{s+t+v-1})$ *products*
  - *These approaches correspond to bilinear algorithms of this rank*

Summary of Bilinear Algorithms

We reviewed bilinear algorithms for 3 problems, which may all be viewed as special cases of tensor contractions

- *fast matrix multiplication algorithms such as Strassen's, reduce the asymptotic scaling of tensor contractions, as these are isomorphic to mat.-mul.*
- *fast convolution algorithms such as Toom-Cook and DFT/FFT, reduce even more significantly the asymptotic cost of tensor contractions with tensors that have Toeplitz/Hankel/circulant structure, as these are equivalent to convolutions*
- *symmetry-preserving tensor contractions algorithms reduce cost of tensor contractions by a factor that increases factorially with tensor order, if the tensors involved are symmetric*

Summary of Nested Bilinear Algorithms

For the tensor $\mathcal{T}^{(n)}$ defining any of the 3 problems for input size n, $\mathcal{T}^{(n)} \otimes \mathcal{T}^{(n)}$ defines a problem for larger inputs

- *in each case, we may obtain a bilinear algorithm of rank* R^2 *for* $\mathcal{T}^{(n)} \otimes \mathcal{T}^{(n)}$ *from bilinear algorithms of rank* R *for* $\mathcal{T}^{(n)}$ *via Kronecker products of the factors*
- *for matrix multiplication with dimension* n, $\mathcal{T}^{(n)} \otimes \mathcal{T}^{(n)}$ *defines the tensor for multiplication of matrices with dimension* n^2
- *for convolution of vectors with dimension* n, $\mathcal{T}^{(n)} \otimes \mathcal{T}^{(n)}$ *defines a 2D convolution (to which a 1D convolution of size equal to or within a constant of* n^2 *can be reduced)*
- *for symmetric tensor contractions,* $\mathcal{T}^{(n)} \otimes \mathcal{T}^{(n)}$ *defines the problem of contracting two partially symmetric tensors (with two groups of symmetric modes)*