CS 598 EVS: Tensor Computations

Bilinear Algorithms

Edgar Solomonik

University of Illinois at 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*

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

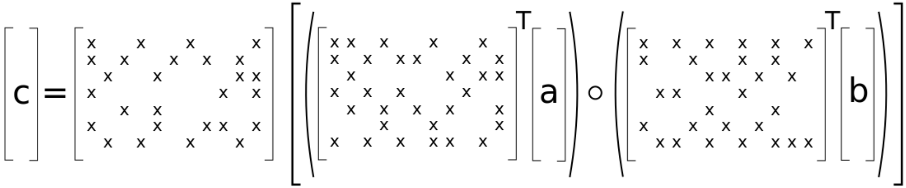
  - *Correlation obtained by transposing the first and last mode of the linear convolution tensor*
  - *Cyclic convolution has $t_{ijk} = 1$ if and only if $k + i - j = 0 \pmod{n}$*

Bilinear Algorithms

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

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

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

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

- 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,

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

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,* $\boldsymbol{y} = \boldsymbol{D}^{(n)^{-1}}\big((\boldsymbol{D}^{(n)}\boldsymbol{f}) \odot (\boldsymbol{D}^{(n)}\boldsymbol{g})\big)$, *we obtain*

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

  - *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* $\boldsymbol{C}_{\langle\boldsymbol{a}\rangle} = \boldsymbol{D}^{(n)^{-1}}\operatorname{diag}(\boldsymbol{D}^{(n)}\boldsymbol{a})\boldsymbol{D}^{(n)}$

Winograd's Algorithm for Convolution

- The DFT/FFT requires complex arithmetic, motivating alternatives such as the more general Winograd family of algorithms
  - *In Winograd's convolution algorithm, the remainder of the product $v = pq$ is computed using k distinct polynomial divisors, $m^{(i)}$, whose product is the polynomial M with deg$(M) >$ deg(v)*
  - *The k polynomial divisors, $m^{(1)}, m^{(2)}, \cdots, m^{(k)}$ must be coprime*
  - *From the k remainders, $u^{(i)} = pq \mod m^{(i)}$ the remainder $v = pq$ mod M is recovered via the Chinese remainder theorem*
  - *The theorem leverages Bézout's identity, which states that there exist polynomials $n^{(i)}$ and $N^{(i)}$ such that, for $M^{(i)} = M/m^{(i)}$,*

    $$M^{(i)}N^{(i)} + m^{(i)}n^{(i)} = 1$$

    which allow us to construct v

    $$v = \left(\sum_{i=1}^{k} u^{(i)}M^{(i)}N^{(i)}\right) \mod M$$

  - *Toom-Cook algorithms are special cases of Winograd's convolution algorithm, where the polynomial divisors are $m^{(i)}(x) = x - \chi_i$, where χ_i are nodes*

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm can be written as a bilinear algorithm by defining appropriate linear transformations
  - *Linear convolution corresponds to a product with a Toeplitz matrix,* $\boldsymbol{c} = \boldsymbol{T}_{\langle \boldsymbol{a},k\rangle}\boldsymbol{b}$ *where* $\boldsymbol{T}_{\langle \boldsymbol{a},k\rangle} \in \mathbb{R}^{n+k-1\times k}$ *is*

$$\boldsymbol{T}_{\langle \boldsymbol{a},k\rangle} = \begin{bmatrix} a_0 & & \\ \vdots & \ddots & \\ a_{n-1} & & a_0 \\ & \ddots & \vdots \\ & & a_{n-1} \end{bmatrix}$$

  - *Let* $\boldsymbol{X}_{\langle m,d\rangle} \in \mathbb{C}^{\textit{deg}(m)\times(d+1)}$ *be a matrix that computes the coefficients of* $\rho = p \pmod{m}$ *when multiplied by coefficients of degree* d *polynomial* p*,* $\boldsymbol{\rho} = \boldsymbol{X}_{\langle m,d\rangle}\boldsymbol{p}$

$$\boldsymbol{X}_{\langle m,d\rangle} = \begin{bmatrix}\boldsymbol{I} & -\boldsymbol{L}\boldsymbol{U}^{-1}\end{bmatrix}$$

    where $\boldsymbol{I}$ *is an identity matrix of dimension deg*(m)*,* $\boldsymbol{L}$ *contains the top deg*(m) *rows of* $\boldsymbol{T}_{\langle \boldsymbol{m},d-\textit{deg}(m)+1\rangle}$*, and* $\boldsymbol{U}$ *contains the bottom* $d+1$ *rows of* $\boldsymbol{T}_{\langle \boldsymbol{m},d-\textit{deg}(m)+1\rangle}$

Algebraic Formulation of Winograd's Algorithm for Convolution

- Given an operator $\boldsymbol{X}_{\langle m,d\rangle} \in \mathbb{C}^{\mathsf{deg}(m)\times(d+1)}$ to compute coefficients of $\rho = p \pmod{m}$, we can efficiently compute

$$pq \bmod m = (p \bmod m)(q \bmod m) \bmod m,$$
$$\boldsymbol{X}_{\langle m,\mathit{deg}(p)+\mathit{deg}(q)-1\rangle}(\boldsymbol{p} * \boldsymbol{q}) = \boldsymbol{X}_{\langle m,2\mathit{deg}(m)-1\rangle}\big((\boldsymbol{X}_{\langle m,\mathit{deg}(p)\rangle}\boldsymbol{p}) * (\boldsymbol{X}_{\langle m,\mathit{deg}(q)\rangle}\boldsymbol{q})\big)$$

  - *Further, given a bilinear algorithm $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ to compute linear convolution of two m-dimensional vectors, we can obtain a bilinear algorithm $(\boldsymbol{X}^T_{\langle m,\mathit{deg}(p)\rangle}\boldsymbol{A}, \boldsymbol{X}^T_{\langle m,\mathit{deg}(q)\rangle}\boldsymbol{B}, \boldsymbol{X}_{\langle m,2\mathit{deg}(m)-1\rangle}\boldsymbol{C})$ to compute $\rho = pq \bmod m$, since*

$$\boldsymbol{\rho} = \boldsymbol{X}_{\langle m,2\mathit{deg}(m)-1\rangle}\boldsymbol{C}\big((\boldsymbol{A}^T\boldsymbol{X}_{\langle m,\mathit{deg}(p)\rangle}\boldsymbol{p}) \odot (\boldsymbol{B}^T\boldsymbol{X}_{\langle m,\mathit{deg}(q)\rangle}\boldsymbol{q})\big).$$

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm effectively merges smaller bilinear algorithms for linear convolution
  - *Given* $M = \prod_{i=1}^{k} m^{(i)}$ *where deg*$(M) = n + r - 1$ *and* $m^{(1)}, \cdots, m^{(k)}$ *are coprime, as well as* $(\boldsymbol{A}^{(i)}, \boldsymbol{B}^{(i)}, \boldsymbol{C}^{(i)})$ *for* $i \in \{1, \ldots, k\}$, *where* $(\boldsymbol{A}^{(i)}, \boldsymbol{B}^{(i)}, \boldsymbol{C}^{(i)})$ *is a bilinear algorithm for linear convolution of vectors of dimension deg*$(m^{(i)})$
  - *Winograd's convolution algorithm yields a bilinear algorithm* $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ *for computing linear convolution with vectors of dimension* r *and* n, *where*

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{X}^T_{\langle m^{(1)}, r-1\rangle} \boldsymbol{A}^{(1)} & \cdots & \boldsymbol{X}^T_{\langle m^{(k)}, r-1\rangle} \boldsymbol{A}^{(k)} \end{bmatrix},$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{X}^T_{\langle m^{(1)}, n-1\rangle} \boldsymbol{B}^{(1)} & \cdots & \boldsymbol{X}^T_{\langle m^{(k)}, n-1\rangle} \boldsymbol{B}^{(k)} \end{bmatrix}, \textit{and}$$

$$\boldsymbol{C} = \begin{bmatrix} \tilde{\boldsymbol{C}}^{(1)} & \cdots & \tilde{\boldsymbol{C}}^{(k)} \end{bmatrix}$$

where $\tilde{\boldsymbol{C}}^{(i)} = \boldsymbol{X}_{\langle M, \mathit{deg}(M) + \mathit{deg}(m^{(i)}) - 2\rangle} \boldsymbol{T}_{\langle \boldsymbol{e}^{(i)}, \mathit{deg}(m^{(i)})\rangle} \boldsymbol{X}_{\langle m^{(i)}, 2\mathit{deg}(m^{(i)}) - 1\rangle} \boldsymbol{C}^{(i)}$ *and* $\boldsymbol{e}^{(i)}$ *are coefficients of polynomial* $e^{(i)} = M^{(i)} N^{(i)} \bmod M$.

Algebraic Formulation of Winograd's Algorithm for Convolution

- A missing piece of the above formulation is how to realize Bézout's identity to compute $N^{(i)}$ and $e^{(i)}$
  - $e^{(i)} = M^{(i)} N^{(i)} \bmod M$ *so it suffices to compute* $n^{(i)}$ *and* $N^{(i)}$ *then apply previously mentioned linear transformations*
  - *The extended Euclidian algorithm can be used for this task, or one can solve a linear system*
  - *The coefficients of polynomials* $\hat{N}$ *and* $\hat{n}$ *satisfying* $\hat{M}\hat{N} + \hat{m}\hat{n} = 1$ *for coprime* $\hat{M}$ *and* $\hat{m}$ *are*

$$\begin{bmatrix} \hat{\boldsymbol{N}} \\ \hat{\boldsymbol{n}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_{\langle \hat{\boldsymbol{M}}, \mathit{deg}(\hat{m})-1\rangle} & \boldsymbol{T}_{\langle \hat{\boldsymbol{m}}, \mathit{deg}(\hat{M})-1\rangle} \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Nested Bilinear Algorithms for Convolution

- 2D convolution is equivalent to nested 1D convolution
  - *Given* $\boldsymbol{F} \in \mathbb{R}^{r\times r}$ *and* $\boldsymbol{G} \in \mathbb{R}^{n\times n}$*, the 2D linear convolution* $\boldsymbol{Y} = \boldsymbol{F} * \boldsymbol{G}$ *with* $\boldsymbol{Y} \in \mathbb{R}^{(n+r-1)\times(n+r-1)}$ *gives*

$$y_{ab} = \sum_{i=\max(0,a-n+1)}^{\min(a,r-1)} \sum_{j=\max(0,b-n+1)}^{\min(b,r-1)} f_{ij} g_{a-i,b-j}$$

  - *2D bilinear problem is defined by tensor* $\mathcal{T}^{(2D)} = \mathcal{T} \otimes \mathcal{T}$ *where* $\otimes$ *is the natural generalization of Kronecker product to tensors*
- 1D convolution can be reduced to 2D convolution with some work
  - *For linear convolution, with vectors of dimension* $n = st$ *can reduce to* $s \times t$ *2D convolution to obtain rank* $(2s-1)(2t-1)$ *bilinear algorithm via overlap-add technique, which computes partial sums of the result of the 2D convolution*
  - *For cyclic convolution, Agarwal-Cooley algorithm uses the Chinese remainder theorem for integers to decouple dimension* $n = st$ *convolution to* $s \times t$ *2D cyclic convolution via permutations*
- For more details on the above derivations and a broader survey of convolution algorithms, see `https://arxiv.org/abs/1910.13367`

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* $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$ *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 $i \geq j$, $c_{ij} = a_i b_j + a_j b_i$*
  - *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*

Hankel Matrix Vector Product

- A Hankel matrix is a reflection of a Toeplitz matrix (which can similarly be used to compute convolution),

$$\boldsymbol{H} = \begin{bmatrix} h_1 & h_2 & h_3 & \cdots \\ h_2 & h_3 & & \\ h_3 & & \ddots & \\ \vdots & & & \end{bmatrix}$$

  - *Hankel matrices are symmetric across blocks and Hankel within blocks*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 & \boldsymbol{H}_2 \\ \boldsymbol{H}_2 & \boldsymbol{H}_1 \end{bmatrix}$$

    but not vice versa, i.e., some elements in $\boldsymbol{H}_1$ must be the same as in $\boldsymbol{H}_2$

- Can compute Hankel mat-vec by applying symmetric mat-vec recursively
  - *For $n = 2$ fast bilinear algorithm requires $n(n+1)/2 = 3$ products*
  - *Additions of Hankel matrices and vector entries can be done in $O(n)$ time, so*

$$T(n) = 3T(n/2) + O(n) = O(n^{\log_2(3)})$$

  - *This complexity is as good as the recursive application of Toom-Cook, but $O(n \log n)$ can be achieved via FFT or other techniques*

Group Symmetry

- Tensors arising in physical simulations often have group structure that reflects conservation laws
  - *Abelian group symmetries can be mapped to cyclic group, which can be used to define a block-sparse form of the tensors (here represented using extra modes)*
  - *A particularly common/important contraction with cyclic group symmetry is*

$$w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}.$$

where for some group size G, we have symmetries, e.g.,

$$\begin{aligned} w_{aA,bB,iI,jJ} &\neq 0 \text{ if } A + B - I - J \equiv 0 \pmod G, \\ u_{aA,bB,kK,lL} &\neq 0 \text{ if } A + B + K + L \equiv 0 \pmod G, \\ v_{kK,lL,iI,jJ} &\neq 0 \text{ if } K + L - I - J \equiv 0 \pmod G. \end{aligned}$$

  - *We can write each of these tensors using a reduced form and an irrep map,*

$$w_{aA,bB,iI,jJ} = r^{(W)}_{aA,bB,iI,j} m_{ABIJ}$$

where $m_{ABIJ} = 1$ if $A + B - I - J \equiv 0 \pmod G$ and $m_{ABIJ} = 0$ otherwise

Fast Algorithms for Contraction with Group Symmetry

- The irrep map tensor describes the cyclic group and has a lot of structure
  - *From definition,* $m_{ABIJ} = 1$ *if* $A + B - I - J \equiv 0 \pmod{G}$ *and* $m_{ABIJ} = 0$ *otherwise, we see there are* $O(G^3)$ *nonzeros*
  - *No matter the order of* $\boldsymbol{\mathcal{M}}$ *or the ordering of indices, it will have a tensor train decomposition of rank* G*, in particular,*

$$m_{ABIJ} = m^{(1)}_{ABQ} m^{(2)}_{QIJ} \quad \textit{where} \quad m^{(1)}_{ABQ} = 1 \quad \textit{iff} \quad A + B \equiv Q \pmod{G}, \quad \textit{etc.,}$$

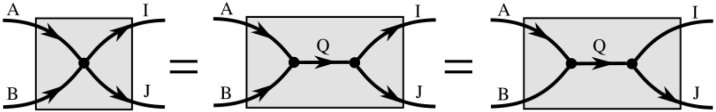
  - *We can then also define a reduced form relative to an auxiliary index in such a decomposition, e.g.,*

$$w_{aA,bB,iI,jJ} = \sum_{Q} r^{(W)}_{aA,bB,iQ,j} m^{(1)}_{ABQ} m^{(2)}_{QIJ}$$

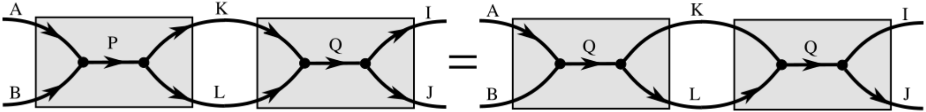
  - *By defining consistent reduced forms using auxiliary indices, we can efficiently contraction group symmetric tensors via set of dense symmetric contractions*

Fast Algorithms for Contraction with Group Symmetry

- We can represent the special reduced form via a tensor diagram

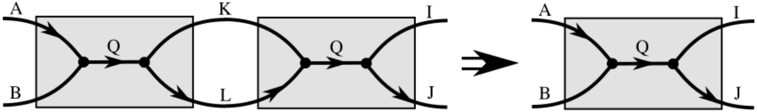


- Given a consistent choice of intermediate index in the two operand tensors, they can be unified



Fast Algorithms for Contraction with Group Symmetry

- After the indices are matched, we can contract efficiently



$$r^{(W)}_{aA,b,i,jJ,Q} = \sum_{k,l,L} r^{(U)}_{aA,b,k,lL,Q} r^{(V)}_{k,lL,i,jJ,Q}$$

- Naive contraction of original tensors had cost $O(n^6)$
  - *New algorithm has cost* $O(n^6/G^2)$
  - *Factor of* G^2 *improvement attainable for also for other (higher order) contractions*

Bilinear Algorithm for Contraction with Group Symmetry

- A group symmetric contraction is a bilinear algorithm
  - *Can view the contraction in a nested fashion as*

$$\mathcal{K}^{(W)}_{ABIJ} = \sum_{KL} \mathcal{K}^{(U)}_{ABKL} \cdot \mathcal{K}^{(V)}_{KLIJ}$$

where $\mathcal{F} = \mathcal{K}^{(U)}_{ABKL} \in \mathbb{R}^{n\times n\times n\times n}$ *and* $\mathcal{G} = \mathcal{K}^{(V)}_{ABKL} \in \mathbb{R}^{n\times n\times n\times n}$*, so that* $\mathcal{H} = \mathcal{F} \cdot \mathcal{G}$ *gives*

$$h_{abij} = \sum_{kl} f_{abkl} g_{klij} = \sum_{kl} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}$$

  - *Therefore, it suffices to find a bilinear algorithm for the first contraction of* $G \times G \times G \times G$ *tensors*

$$k^{(W)}_{ABIJ} = \sum_{KL} k^{(U)}_{ABKL} \cdot k^{(V)}_{KLIJ}$$

with the same sparsity as irrep maps $\mathcal{M}^{(U)}$*,* $\mathcal{M}^{(V)}$*, and* $\mathcal{M}^{(W)}$
  - *An algorithm for the full contraction can then be constructed by nesting with a standard blockwise contraction*

Bilinear Algorithm for Contraction with Group Symmetry

- The group symmetric contraction algorithm leverages a CP decomposition
  - *The tensor defining the bilinear problem group symmetric contraction can be written as*

$$T_{ABIJ,\hat{A}\hat{B}KL,\hat{K}\hat{L}\hat{I}\hat{J}} = \delta_{A+B,I+J}\delta_{A+B,-K-L} \prod_{X\in\{A,B,I,J,K,L\}} \delta(X,\hat{X})$$

  - *Using the CP decomposition obtained via the identity* $\delta_{A+B,I+J}\delta_{A+B,-K-L} = \sum_{Q=1}^{G}\delta_{AB,Q}\delta_{IJ,Q}\delta_{-K-L,Q}$, *we can define a bilinear algorithm of rank* G^4 *that acts as follows*

$$k^{(W)}_{ABIJ} = \sum_{\hat{A}\hat{J}LQ} \delta_{A,\hat{A}}\delta_{J,\hat{J}}\delta_{AB,Q}\delta_{IJ,Q} \left(\sum_{\hat{B}K}\delta_{-K-L,Q}k^{(U)}_{\hat{A}\hat{B}KL}\right) \left(\sum_{\hat{I}\hat{K}}\delta_{\hat{I}\hat{J},Q}k^{(V)}_{\hat{I}\hat{J}\hat{K}L}\right)$$