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
- These problems admit nontrivial fast *bilinear algorithms*, which correspond to low-rank CP decompositions of the tensors

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 $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{m \times n \times k}$

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

Bilinear Algorithms

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

- 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 $\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$

$$\text{7 products}\left\{\begin{aligned}
\boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &= \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &= \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}\right.$$

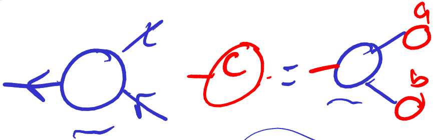
$$\begin{aligned}
\boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2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

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

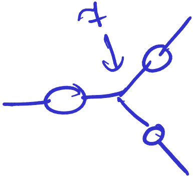
$$T(n) = \underset{\sim}{7}\,T(n/2) + O(n^2)$$

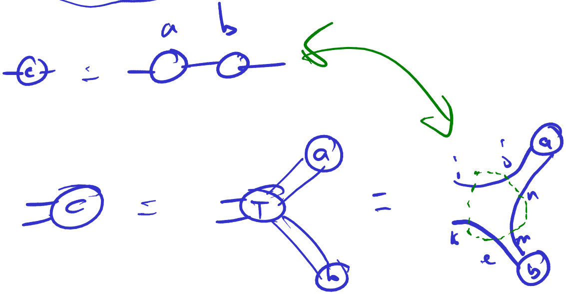
7 is rank of a $4\times4\times4$ 'mm' tensor

Bilinear problem



CP decomposition





$$t_{ijklmn} = \delta_{ij}\,\delta_{kl}\,\delta_{mn}$$

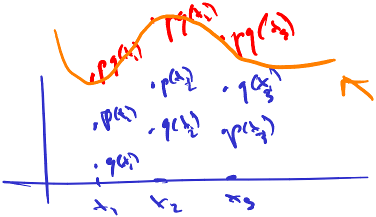
$$\delta_{ii} = 1 \qquad \delta_{ij} = 0 \quad i \neq j$$

Fast Bilinear Algorithms for Convolution

- Linear convolution corresponds to polynomial multiplication

dim n

degree $n-1$



degree $2n-1$ interpolant

poly is unique

- The *Toom-Cook* convolution algorithm computes the coefficients of $p \cdot q$ by computing $(p \cdot q)(x_i)$ for $i \in \{1, \ldots, n+k-1\}$ and interpolates

$2n-1$

$$V^{-1}_{\deg 2n-1}(x)\left(\left(V_{\deg=n-1}(x)\, p\right) \circ \left(V_{\deg=n-1}(x)\, q\right)\right)$$

interp.

$p(x_1) \ldots p(x_n)$

Toom-Cook Convolution and the Fourier Transform

- Vandermonde matrices are ill-conditioned with real nodes, but can be perfectly conditioned with complex nodes

- The *fast Fourier transform (FFT)* can be used to perform products with the DFT matrix in $O(n \log n)$ time

unitary

$u = D_{(n)} v$ $\qquad n_1 \times n_2 \qquad n = n_1 n_2$

$$u_i = \sum_j \omega_n^{ij} v_j$$

$$u_{i_1 + n_1 i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_n^{(i_1 + n_1 i_2)(j_2 + n_2 j_1)} v_{j_1 + n_1 j_2}$$

$$u_{i_1+n_1 i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2} \omega_n^{i_1 j_1} \cdot \underbrace{\omega_n^{(n_1) i_2 j_2}}_{\omega_{n_2}} \cdot \underbrace{\omega_n^{(n_2) i_1 j_1}}_{\omega_{n_1}} \cancel{\underbrace{\omega_n^{(n_1 n_2) i_2 j_1}}_{\omega_n^n = 1}} v_{j_1+n_1 j_2}$$

$$u_{i_1+n_1 i_2} = \sum_{j_1} \sum_{j_2} \omega_n^{i_1 j_1} \, \omega_{n_2}^{i_2 j_2} \, \omega_{n_1}^{i_1 j_1} \, v_{j_1+n_1 j_2}$$

$$= \sum_{j_1} \omega_{n_1}^{i_1 j_1} \Big(\underbrace{\omega_n^{i_1 j_1}}_{(D_{(n)}} \odot \Big(\underbrace{\sum_{j_2} \omega_{n_2}^{i_2 j_2}}_{(D_{(n_2)}} \underbrace{v_{j_1+n_1 j_2}}_{V))} \Big) \Big)$$

$$\left(D_{(n)} \odot \left(D_{(n_2)} \, V \right) \right) D_{(n_1)}$$

radix-2 DFT

$n_1 = 2 \Rightarrow T(n) = 2T(n/2) + O(n) = O(n \log n)$

$n_1 = n_2 = \sqrt{n} \Rightarrow T(n) = 2\sqrt{n}\, T(\sqrt{n}) + O(n) = O(n \log n)$

$n\, 2^{\log \log n} = n \log n$

Cyclic Convolution via DFT

- For linear convolution $D^{(n+k-1)}$ is used, for cyclic convolution $D^{(n)}$ suffices

- The DFT also arises in the eigendecomposition of a circulant matrix

$$C(a) = \begin{bmatrix} a_1 & & & a_3 & a_2 \\ a_2 & \ddots & & & a_3 \\ a_3 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & a_3 & a_2 & a_n \end{bmatrix} = D_{(n)}^{-1}(D_{(n)}a)D_{(n)}$$

Winograd's Algorithm for Convolution

- The DFT/FFT requires complex arithmetic, motivating alternatives such as the more general Winograd family of algorithms

p, q as input $n-1$ degree polynomials $M = m_1 \cdots m_n$

$v = pq \bmod M = pq$ $\deg(M) > \deg(v)$

$r_i := v = pq \bmod m_i$ for $i \in \{1, \dots, n\}$

$m_1 \dots m_n$ are coprime

Chinese remainder theorem

~~recovery of v~~ (pq) from

remainder $r_1 \dots r_n$

$\cdot\, M_i = M/m_i = m_1 \cdots m_{i-1} m_{i+1} \cdots m_n$

$\mapsto n_i,\ N_i = M/n_i$

$p = 1 + 2x$

$q = 2 + 3x + x^2$

$m_1 = x$

$m_2 = x^2$

$p \cdot q \bmod m_1 = 2$

$ m_2 = 2 + 7x$

$2 + 7x + 11x^2, \dots$

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm can be written as a bilinear algorithm by defining appropriate linear transformations

$\underline{p \bmod m_i}$ for $i \in 1 \ldots n$

$q \ldots$

$r_i = pq \bmod m_i$

$X_{\langle m,d\rangle} \in \mathbb{C}^{\deg(m) \times (d+1)}$

Toeplitz matrix

$\hat{p} = T_{\langle m\rangle} s + \hat{q}$

$q = X_{\langle m,d\rangle} \underline{p}$

$p \in \mathbb{C}^{d+1}$ $\quad \hat{q} = \hat{p} \bmod \hat{m} \Rightarrow \exists \hat{s}$ s.t. $\hat{p} = \hat{s}\hat{m} + \underbrace{\hat{q}}_{\text{remainder}}$

Algebraic Formulation of Winograd's Algorithm for Convolution

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

$\rho \leftarrow X_{\langle m,d\rangle}\, p$

$[A, B, C]$

bilinear alg for conv with vectors of dim $\deg(m)$

$\hat{p}\hat{q} \bmod \hat{m}$

$\left[(\hat{p} \bmod \hat{m})\ (\hat{q} \bmod \hat{m})\right] \bmod \hat{m}$

$X_{\langle m,d\rangle}\, C^{-1}\left(A\left(X_{\langle m,d\rangle}\hat{p}\right) \circ B\left(X_{\langle m,d\rangle}\hat{q}\right)\right)$

$\hat{m}_1 \ldots \hat{m}_n$

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm effectively merges smaller bilinear algorithms for linear convolution

$\underbrace{[A_1, B_1, C_1]}_{deg(m_1)} \quad \ldots \quad \underbrace{[A_n, B_n, C_n]}_{deg(m_n)}$

$\underbrace{[\tilde{A}, \tilde{B}, \tilde{C}]}_{deg(m)}$

$$\tilde{A} = \begin{bmatrix} A_1 X_{\langle m_1, deg(m_1)\rangle} \\ \vdots \\ A_n X_{\langle m_n, deg(m_n)\rangle} \end{bmatrix}$$

$\tilde{B} = \ldots \quad \tilde{C} = \begin{bmatrix} X_{\langle m_1, deg(m_1)\rangle} C_1 & \cdots \end{bmatrix}$

$$p(x) \bmod x - x_i = p(x_i)$$

Algebraic Formulation of Winograd's Algorithm for Convolution

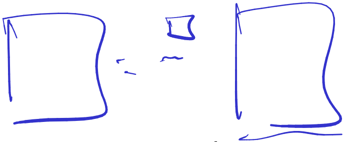
- A missing piece of the above formulation is how to realize Bézout's identity to compute $N^{(i)}$ and $e^{(i)}$

Nested Bilinear Algorithms for Convolution

- 2D convolution is equivalent to nested 1D convolution

$$T^{(1D)}$$

$$X = U * V$$



$$T^{(2D)} = T^{(1D)} \otimes T^{(1D)}$$

- 1D convolution can be reduced to 2D convolution with some work

- 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

nth order sym. tensors

$n!$ permutations

savings in memory

cost?

maybe

- Bilinear algorithms for symmetric tensor contractions exist with lower rank than their nonsymmetric counterparts

bilinear problem of mat. vec.

$T \in \mathbb{R}^{n^2 \times n \times n}$

sym. mat. vec

$T \in \mathbb{R}^{n(n+1)/2 \times n \times n}$

Symmetric Matrix Vector Product

- Consider computing $\boldsymbol{c} = \boldsymbol{A}\boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{A}^T$

$$c_i = \sum_{j=1}^{n} a_{ij} b_j$$

$a_{ij} = a_{ji}$, however, $a_{ij} b_j \neq a_{ji} b_i$

$a_{ij}(b_i + b_j)$

$$c_i = \sum_{j=1}^{n} a_{ij}(b_i + b_j) - \left(\sum_{j=1}^{n} a_{ij}\right) b_i$$

$$\binom{n+1}{2} = \frac{n(n+1)}{2}$$

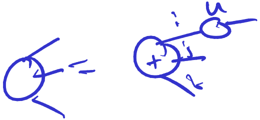
$$c_i = \sum_{j<i}^{n} a_{ij}(b_i + b_j) + \left(a_{ii} - \sum_{j\neq i}^{n} a_{ij}\right) b_i \approx n^2/2$$

$\binom{n}{2} = \frac{n(n-1)}{2}$ unique

n unique products

Partially-Symmetric Tensor Times Matrix (TTM)

- Can use symmetric mat-vec algorithm to accelerate TTM with partially symmetric tensor from $2n^4$ operations to $(3/2)n^4 + O(n^3)$



$t_{ijk} = t_{jik}$

$t_{ijk}(\underbrace{u_{i\ell} + u_{j\ell}}_{\bar{u}_{ij\ell}})$ $O(n^3)$

$w^{(k)}_{i\ell} = \sum_j t^{(k)}_{ij} u_{j\ell}$

$W^{(k)} = T^{(k)} U$

$O(n^3)$ to compute

$a_{ij} = \sum_k a_{ijk}$ for each $T^{(k)}$

$2 \cdot n \cdot \frac{n(n+1)}{2} \cdot n = n^4 + O(n^3)$

$2n^4$

Computing Symmetric Matrices

- Output symmetry can also be used to reduced cost, for example when computing a symmetrized outer product $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^T + \boldsymbol{b}\boldsymbol{a}^T$
- To symmetrize product of two symmetric matrices, can compute anticommutator, $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}$

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 $v+t$)
- Such tensor contractions can be done using $n^{s+t+v}/(s+t+v)! + O(n^{s+t+v-1})$ products