Lecture 19 Introduction to Matrix Multiplication and Bilinear Problems
Scribe: Ron Estrin, Brad Nelson, Nolan Skochdopole, Yinzhan Xu **Date:** November 16, 2021

1 Introduction

Starting from this lecture, we start a series of several lectures around the topic of matrix multiplication. In this lecture, we will first describe Strassen's algorithm, which multiplies two $n \times n$ matrices in $O(n^{2.81})$ field operations. Then we will introduce bilinear problems, which will be used in the next several lectures.

Some of these notes were adapted from Markus Bläser's notes [3].

Let $\mathcal{K}$ be a field. Given a matrix $X \in \mathcal{K}^{m \times n}$ and a matrix $Y \in \mathcal{K}^{n \times l}$, their product Z is an $m \times l$ matrix such that $Z_{i,j} = \sum_k X_{i,k} Y_{k,j}$.

Recall the definition of straight line program (SLP)/arithmetic circuit model, where each operation is an arithmetic operation $+, -, *, /$ in the field that takes two already computed values as inputs, or λ_+ that adds a scalar to a computed value, or λ_* that multiplies a computed value with a scalar.

Let F be a rational function on variables $x_1, \ldots, x_n$, we define its cost $C(F)$ to be the minimum number of operations in a SLP for F. In Ostrowski's cost model, the cost of $*$ and $/$ are 1, and the cost of all other operations are 0. Let $C^{*/}$ be the minimum Ostrowski cost of a straight line program computing F. We will see that for matrix multiplication, considering the Ostrowski cost is sufficient.

2 Strassen's Algorithm

Strassen's algorithm [2] considers matrices partitioned into blocks, and multiplication is computed block-wise.

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \quad Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \quad \Longrightarrow z = \begin{bmatrix} x_{11}y_{11} + x_{12}y_{21} & x_{11}y_{12} + x_{12}y_{22} \\ x_{21}y_{11} + x_{22}y_{21} & x_{21}y_{12} + x_{22}y_{22} \end{bmatrix}$$

Naively there are 8 matrix multiplications on matrices size $\frac{n}{2} \times \frac{n}{2}$. Strassen showed that we can use only 7 at the cost of some extra additions/subtractions:

Theorem 2.1 (Strassen)**.** $C^{*/}(2 \times 2$ *matrx multiplication*$) \leq 7$.

Algorithm 1: Strassen(A, B)

$P_1 \leftarrow (x_{11} + x_{22})(y_{11} + y_{22})$;
$P_2 \leftarrow (x_{21} + x_{22})y_{11}$;
$P_3 \leftarrow x_{11}(y_{12} - y_{22})$;
$P_4 \leftarrow x_{22}(y_{21} - y_{11})$;
$P_5 \leftarrow (x_{11} + x_{12})y_{22}$;
$P_6 \leftarrow (x_{21} - x_{11})(y_{11} + y_{12})$;
$P_7 \leftarrow (x_{12} - x_{22})(y_{21} + y_{22})$;
$z_{11} \leftarrow P_1 + P_4 - P_5 + P_7$;
$z_{12} \leftarrow P_3 + P_5$;
$z_{21} \leftarrow P_2 + P_4$;
$z_{22} \leftarrow P_1 - P_2 + P_3 + P_6$;
return Z ;

Proof. In Algorithm 1, computing each of P_i costs one multiplication and no multiplications are used elsewhere. It is easy while tedious to check that $z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}$ are correct entry values for Z. □

Using the Strassen's algorithm for 2×2 matrix multiplication, we can actually obtain a faster matrix multiplication algorithm for general n.

Proposition 1. *$C^{*,/}(2 \times 2$ matrx multiplication$) \leq 7$ implies that $n \times n$ matrix multiplication has an $O(n^{\log_2 7})$ time algorithm (and also an SLP of that size).*

Proof. Given matrices X, Y of dimension $n \times n$, we want to compute XY. Without loss of generality, we can assume that n is a power of 2 (If n is not a power of 2, we can pad empty columns and rows to both matrix). We can split X, Y to four $n/2 \times n/2$ blocks, and apply Algorithm 1 by regarding each $n/2$ by $n/2$ block of X, Y as elements in the ring $\mathcal{K}^{n/2 \times n/2}$. Addition and Subtraction in this ring takes $O(n^2)$ time. If we let $T(n)$ to be the time to multiply two $n \times n$ matrices, then multiplication in $\mathcal{K}^{n/2 \times n/2}$ takes $T(n/2)$ time. Therefore, if we apply Algorithm 1 recursively, we get that

$$T(n) \leq 7 \cdot T(\frac{n}{2}) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7}) = O(n^{2.81}).$$

□

3 Quadratic and Bilinear Problems

Let $\mathcal{K}$ be a field. Let $x_1, \ldots, x_N$ be N variables. Let $F = \{f_1, \ldots, f_K\}$ be a set of rational functions over $x_1, \ldots, x_N$ over $\mathcal{K}$. We say F is a set of quadratic forms if $f_k = \sum_{i,j=1}^{N} t_{i,j,k} x_i x_j$ for all $1 \leq k \leq K$ for some coefficients $t_{i,j,k}$.

If we split the variables into two parts $x_1, \ldots, x_N, y_1, \ldots, y_M$, we say F is a set of bilinear forms if $f_k = \sum_{i=1}^{N} \sum_{j=1}^{M} t_{i,j,k} x_i y_j$ for all $1 \leq k \leq K$. Here, $t = \{t_{i,j,k}\}_{i \in [N], j \in [M], k \in [K]}$ is an order 3 tensor in $\mathcal{K}^{N \times M \times K}$.

3.1 Examples of bilinear problems

The first example is $N \times M \times K$ matrix multiplication. In matrix multiplication, given $\{x_{i,j}\}_{i,j}$ and $\{y_{j,k}\}_{j,k}$, we need to compute $z_{i,k} = \sum_j x_{i,j} y_{j,k}$. We can consider this as a bilinear problem. Let t be the matrix multiplication tensor for $N \times M$ by $M \times K$ matrix multiplications, then

$$t_{(i,j'),(j,k'),(i',k)} = \begin{cases} 1 & \text{if } i = i', j = j', k = k' \\ 0 & \text{otherwise.} \end{cases}$$

For symmetry, we actually define the matrix multiplication tensor $\langle N, M, K \rangle$ of $N \times M$ by $M \times K$ matrix multiplication as t above but the last coordinates are (k, i') instead of (i', k), effectively computing the transpose of z instead of z:

$$\langle N, M, K \rangle_{(i,j'),(j,k'),(k,i')} = \begin{cases} 1 & \text{if } i = i', j = j', k = k' \\ 0 & \text{otherwise.} \end{cases}$$

Another example is the polynomial multiplication tensor. Let $p(z) = \sum_{i=0}^{N} a_i z^i$, and $q(z) = \sum_{j=0}^{M} b_j z^j$ be two polynomials with coefficients in $\mathcal{K}$. Then their product is $(p \cdot q)(z) = \sum_{\ell=0}^{M+N} c_\ell z^\ell$, where

$$c_\ell = \sum_{\substack{i \\ j=\ell-i}} a_i b_j.$$

The tensor for polynomial multiplication is defined as

$$t_{i,j,\ell} = \begin{cases} 1 & \text{if } i + j = \ell \\ 0 & \text{otherwise.} \end{cases}$$

3.2 Trilinear Representation

If $F = \{f_1, \ldots, f_K\}$ is a set of bilinear forms over variables $x_1, \ldots, x_N, y_1, \ldots, y_M$, where for each $k \in [K]$, $f_k = \sum_{i=1}^{N} \sum_{j=1}^{M} t_{ijk} x_i y_j$, we can represent the bilinear problem of computing F via the following trilinear polynomial:

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{K} t_{ijk} x_i y_j z_k,$$

where we think of z_k as the output variables, and each f_k is just the coefficient in front of z_k.

Thus bilinear computational problems are in one to one correspondence with trilinear polynomials. The coefficients t_{ijk} define the tensor corresponding to the bilinear problem, and sometimes we can talk about the tensor being the trilinear polynomial itself.

For matrix multiplication we choose to use the z variables representing the transpose of the output, so that the $\langle M, N, L \rangle$ tensor is just

$$\sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{\ell=1}^{L} X_{ij} Y_{j\ell} Z_{\ell i}.$$

As we now also have z variables, we say that a bilinear problem is defined over three variable sets $x_\cdot, y_\cdot, z_\cdot$.

3.3 Pictorial Representation of Tensors for Bilinear Problems

Suppose the bilinear problem is over variables $\{x_1, \ldots, x_N, y_1, \ldots, y_M, z_1, \ldots, z_K\}$, and is $\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} t_{ijk} x_i y_j z_k$, then the pictorial representation can be viewed as an $M \times N$ matrix, where the (i, j)-th entry is $\sum_{\ell} t_{i,j,\ell} z_\ell$.

For instance, the pictorial representation of 2×2 matrix multiplication is

$y_{2,2}$	0	$z_{1,2}$	0	$z_{2,2}$
$y_{2,1}$	0	$z_{1,1}$	0	$z_{2,1}$
$y_{1,2}$	$z_{1,2}$	0	$z_{2,2}$	0
$y_{1,1}$	$z_{1,1}$	0	$z_{2,1}$	0
	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	$x_{2,2}$

Another example is the pictorial representation of degree 3 polynomial multiplication, which can be represented as

b_3	c_3	c_4	c_5	c_6
b_2	c_2	c_3	c_4	c_5
b_1	c_1	c_2	c_3	c_4
b_0	c_0	c_1	c_2	c_3
	a_0	a_1	a_2	a_3

As we can see, the values of the tensor are the same along the diagonals. In some sense, we can regard Fast Fourier Transformation as an algorithm that exploits this structure. In order to obtain fast matrix multiplication algorithms, we need to exploit the structure of the matrix multiplication tensor, which has a completely different structure.

3.4 Vermeidung von Divisionen

Strassen [1] showed that division is not necessary for computing any set of quadratic forms. His paper was called "Vermeidung von Divisionen", which means "avoiding divisions" in German. This result can be more formally described as the following theorem.

Theorem 3.1 ([1]). *Let $F = \{f_1, \ldots, f_k\}$ be a set of quadratic forms where $f_k = \sum_{i,j} t_{i,j,k} x_i x_j$. If the field has infinite size and $C^{*/}(F) = \ell$, then there exists a SLP of Ostrowski cost ℓ consisting of the following multiplications.*

$$P_\lambda = \left(\sum_i u_{\lambda,i} x_i\right)\left(\sum_j v_{\lambda,j} x_j\right) \quad \forall 1 \le \lambda \le \ell,$$

and every $f_k \in F$ is a linear combination of $\{P_\lambda\}_\lambda$.

Here is some intuition. Since we can write $1/(1-x)$ as a formal power series $\sum_{i\ge 0} x^i$, and because monomials of degree larger than the largest degree in each f_k would need to cancel, we can ignore them and only represent $1/x$ as a finite sum. Hence divisions can be avoided this way. Also, if an optimal SLP compute a product of three or more variables, since we are only computing quadratic forms, these products will have to eventually cancel as well, so we might as well avoid them. Thus, products of two linear combinations of x_i are enough. We won't formally prove this theorem in this class, but we will use it.

We can use $C^*(F)$ to denote the minimum number of $*$ operations in a SLP computing F with no divisions. Theorem 3.1 implies that when F is a set of quadratic forms, $C^*(F) = C^{*/}(F)$.

3.5 Rank of a Bilinear Problem

Let $F = \{f_k = \sum_{i,j} t_{i,j,k} x_i y_j\}$ be a set of bilinear forms. Consider the minimum number ℓ such that there exist ℓ products

$$P_\lambda = \left(\sum_{i=1}^{N} u_{\lambda,i} x_i\right)\left(\sum_{j=1}^{M} v_{\lambda,j} y_j\right) \quad \forall \lambda \in [\ell],$$

such that every $f_k \in F$ is a linear combination of $\{P_\lambda\}_\lambda$. We define the rank (a.k.a. bilinear complexity) $R(F)$ of F to be this value ℓ.

By definition of $R(F)$, $R(F) \le \ell$ if and only if there exists $u_{\lambda,i}, v_{\lambda,j}, w_{\lambda,k}$ such that for every k, $f_k = \sum_{\lambda=1}^{\ell} w_{\lambda,k}\left(\sum_{i=1}^{N} u_{\lambda,i} x_i\right)\left(\sum_{j=1}^{M} v_{\lambda,j} y_j\right)$. If we write f_k as $\sum_{i,j} t_{i,j,k} x_i y_j$, we obtain that $R(F) \le \ell$ if and only there exists $u_{\lambda,i}, v_{\lambda,j}, w_{\lambda,k}$ such that $t_{i,j,k} = \sum_{\lambda=1}^{\ell} u_{\lambda,i} v_{\lambda,j} w_{\lambda,k}$.

In trilinear notation we have that the rank of $\sum_{i,j,k} t_{i,j,k} x_i y_j z_k$ is the minimum ℓ such that for $\lambda = 1, \ldots, \ell$, there exist vectors $u_\lambda = (u_{\lambda,1}, \ldots, u_{\lambda N})$, $v_\lambda = (v_{\lambda,1}, \ldots, v_{\lambda M})$, $w_\lambda = (w_{\lambda,1}, \ldots, w_{\lambda K})$ so that

$$\sum_{i,j,k} t_{i,j,k} x_i y_j z_k = \sum_{\lambda=1}^{\ell} (\sum_{i=1}^{N} u_{\lambda i} x_i)(\sum_{j}^{M} v_{\lambda j} y_j)(\sum_{k}^{K} w_{\lambda k} z_k).$$

For three vectors $u \in \mathcal{K}^K$, $v \in \mathcal{K}^M$, $w \in \mathcal{K}^N$, we can define a tensor $u \otimes v \otimes w$ to be such that $(u \otimes v \otimes w)_{i,j,k} = u_i v_j w_k$ for $i \in [K], j \in [M], k \in [N]$. We call such a tensor a rank 1 tensor, or a *triad*.

Using this notation, we can rephrase the previous discussion as $R(F) \le \ell$ for F with $t \in \mathcal{K}^{K \times N \times M}$ if and only if for $\lambda = 1, \ldots, \ell$, there exist vectors $u_\lambda \in \mathcal{K}^K, v_\lambda \in \mathcal{K}^M, w_\lambda \in \mathcal{K}^N$ such that $t = \sum_{\lambda=1}^{\ell} u_\lambda \otimes v_\lambda \otimes w_\lambda$, i.e., t is a sum of ℓ rank 1 tensors.

Definition 3.1. *The rank of a tensor t, $R(t)$, is the minimum number of rank 1 tensors that sum to t.*

This is the clear extension of matrix rank to tensor rank.

It turns out that the rank of the tensor of a set of bilinear forms F is closely related to the Ostrowski cost of F.

Theorem 3.2. *Let $F = \{f_1, \ldots, f_k\}$ be a set of bilinear forms, then $C^{*/}(F) \le R(F) \le 2C^{*/}(F)$.*

Proof. By definition of $R(F)$, there exists $\ell = R(F)$ products, for $\lambda = 1, \ldots, \ell$:

$$P_\lambda = \left(\sum_{i=1}^{N} u_{\lambda,i} x_i\right)\left(\sum_{j=1}^{M} v_{\lambda,j} y_j\right),$$

such that every $f_k \in F$ is a linear combination of $\{P_\lambda\}_\lambda$. This is a SLP with Ostrowski cost ℓ. Thus, $C^{*/}(F) \le \ell = R(F)$.

Now let's show that $R(F) \le 2C^{*/}(F)$. Consider a SLP for F that has the optimum Ostrowski cost $C^{*/}(F) = \ell$. By Theorem 3.1, there exists $w_{\lambda,k}, u_{\lambda,i}, u'_{\lambda,i}, v_{\lambda,i}, v'_{\lambda,i}$ such that for every k,

$$f_k = \sum_{\lambda=1}^{\ell} w_{\lambda,k}\left(\sum_i u_{\lambda,i} x_i + \sum_j v_{\lambda,j} y_j\right)\left(\sum_i u'_{\lambda,i} x_i + \sum_j v'_{\lambda,j} y_j\right).$$

We can expand the above formula to write f_k as

$$\sum_{\lambda=1}^{\ell} w_{\lambda,k}\left(\sum_i u_{\lambda,i} x_i\right)\left(\sum_j v'_{\lambda,j} y_j\right) + \sum_{\lambda=1}^{\ell} w_{\lambda,k}\left(\sum_i u'_{\lambda,i} x_i\right)\left(\sum_j v_{\lambda,j} y_j\right) + (\text{linear combination of } x_i x_{i'}, y_j, y_{j'}).$$

Since f_k is bilinear, the linear combination of the products $x_i x_{i'}, y_j, y_{j'}$ must be 0. Therefore,

$$f_k = w_{\lambda,k}\left(\sum_i u_{\lambda,i} x_i\right)\left(\sum_j v'_{\lambda,j} y_j\right) + \sum_{\lambda=1}^{\ell} w_{\lambda,k}\left(\sum_i u'_{\lambda,i} x_i\right)\left(\sum_j v_{\lambda,j} y_j\right),$$

for every k, which means $R(k) \le 2\ell$ by definition of rank. □

3.6 Winograd's Example

Do non-bilinear products help in the computation of bilinear forms? Winograd's example shows that it's possible.

Consider a matrix multiplication of an $M \times 2$ matrix X and a $2 \times N$ matrix Y. This problem can be written as bilinear forms since $z_{i,j} = x_{i,1} y_{1,j} + x_{i,2} y_{2,j}$. We can compute Z using the following SLP:

1. For every $i \in [M], j \in [N]$, compute $p_{i,j} = (x_{i,1} + y_{2,j})(x_{i,2} + y_{1,j})$.
2. For every $i \in [M]$, compute $x_{i,1} x_{i,2}$.
3. For every $j \in [N]$, compute $y_{1,j} y_{2,j}$.
4. Set $z_{i,j} = p_{i,j} - x_{i,1} x_{i,2} - y_{1,j} y_{2,j}$.

We can see the SLP above uses only $MN + M + N$ products.

We can set $M = 2, N = n$, then the SLP above solves matrix multiplication of a 2×2 matrix and a $2 \times n$ matrix using only $3n + 2$ operations of type $*$. On the other hand, the best known bilinear algorithm for computing such matrix products applies Strassen's algorithm between the first matrix and each 2×2 block of the second matrix. Thus, it takes roughly $\frac{n}{2} \times 7 = 3.5n$ products.

References

[1] Volker Strassen, *Vermeidung von Divisionen*. J. Reine 1973. Angew. Math., 264:184–202.

[2] Volker Strassen, *Gaussian elimination is not optimal*. Numerische mathematik 13.4 (1969): 354-356.

[3] Markus Bläser, *Complexity of bilinear problems*. Lecture notes. Scribed by Fabian Bendun, 2009.