AN INTRODUCTION TO FAST MATRIX MULTIPLICATION

JOSHUA KOO

Abstract. In this article, we introduce a couple $N \times N$ matrix multiplication algorithms that run in a faster time complexity than the commonly known time complexity of $O(N^3)$ as well as give a broad overview of a few other mathematical concepts fast matrix multiplication can be applied to.

1. Introduction

Matrices have always been both an important and useful tool in mathematics. Representing linear maps, they provide explicit computations in the field of linear algebra. They can also be applied to graph theory, geometry, ring theory, and many other areas of mathematics. In linear algebra, one of the first things students learn when understanding the fundamentals of matrices are matrix multiplication, known as the binary operation that takes two matrices and produces another matrix as an output, equivalent to the composition of linear maps. First described by the French mathematician Jacques Philippe Marie Binet in 1812, the common convention to multiply two matrices functions as the following: Given an $M \times N$ matrix A and an $N \times P$ matrix B,

$$A = \begin{pmatrix} A_{11} & \dots & A_{1N} \\ & \vdots & \\ A_{M1} & \dots & A_{MN} \end{pmatrix}, B = \begin{pmatrix} B_{11} & \dots & B_{1P} \\ & \vdots & \\ B_{N1} & \dots & B_{NP} \end{pmatrix},$$

then the product $C = AB$ is equivalent to the following:

$$C = \begin{pmatrix} A_{11}B_{11} + \dots + A_{1N}A_{N1} & \dots & A_{11}B_{1P} + \dots + A_{1N}B_{NP} \\ & \vdots & \\ A_{M1}B_{11} + \dots + A_{MN}A_{N1} & \dots & A_{M1}B_{1P} + \dots + A_{MN}B_{NP} \end{pmatrix}.$$

Namely,

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}.$$

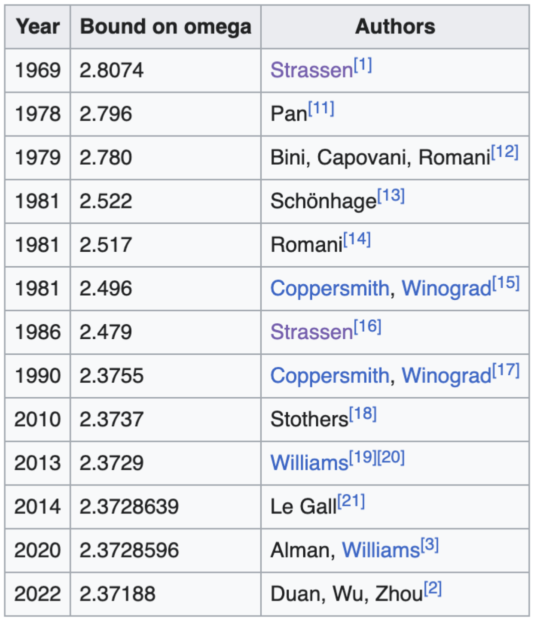
Note that in this paper, we will only observe the multiplication of $N \times N$ matrices, meaning $M = N$ and $P = N$ above. For convenience, let us now look at the pseudocode for multiplying two $N \times N$ matrices:

```
function Matrix_Multi(Matrix A, Matrix B) {
    Matrix C = C[N][N] #initialized to 0
    for (i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
            }
        }
    }
    return C
}
```

From the pseudocode, we can see that the time complexity of the matrix multiplication function is $O(N^3)$, by the 3 nested for loops. For those who do not know how time complexity is defined, here is a definition proposed by Michael Sipser:

"Let M be a deterministic Turing machine [an algorithm] that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine. Customarily we use n to represent the length of the input."

So why is this important? Faster time complexity can optimize programs and make computers more efficient, as well as influence real-world applications. Since introduced, it has always been believed that $O(N^3)$ is the fastest time complexity and that there's no other faster algorithm. However, this was proven false in 1969, when Volker Strassen, a German mathematician, published a faster algorithm proving that $O(N^3)$ is not optimal. Since then, mathematicians have continuously tried to find a faster algorithm. Proved by Duan, Wu, and Zhou, the fastest existing algorithm currently has a time complexity of $O(N^{2.37188})$. Here is an image of the history of time complexities for matrix multiplication:



Year	Bound on omega	Authors
1969	2.8074	Strassen[1]
1978	2.796	Pan[11]
1979	2.780	Bini, Capovani, Romani[12]
1981	2.522	Schönhage[13]
1981	2.517	Romani[14]
1981	2.496	Coppersmith, Winograd[15]
1986	2.479	Strassen[16]
1990	2.3755	Coppersmith, Winograd[17]
2010	2.3737	Stothers[18]
2013	2.3729	Williams[19][20]
2014	2.3728639	Le Gall[21]
2020	2.3728596	Alman, Williams[3]
2022	2.37188	Duan, Wu, Zhou[2]

2. Strassen Algorithm

Strassen's algorithm states that there is an algorithm faster than $O(N^3)$, namely $O(N^{\log_2 7})$.

Proof. Let $AB = C$. Strassen's algorithm begins by partitioning A, B, and C into equally sized matrices like the following:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

In the standard $O(N^3)$ algorithm, 8 multiplications would need to be required. However, Strassen defines new matrices to reduce the required number of multiplications to 7, as shown below:

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{21}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}$$

We can then represent C like the following:

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}.$$

$\square$

Note that the time complexity is $O(N^{\log_2 7})$ because we can recursively partition the $N \times N$ matrix into four blocks $\log_2 N$ times as each M_i has at most one degree of multiplication and are independent of each other. There are of course several limitations. Here is a statement by instructor Reza Zadeh from Stanford University:

"Our PRAM model [namely Strassen's algorithm] assumes zero communication costs between processors. The reason is because the PRAM model assumes a shared memory model, in which each processor has fast access to a single memory bank. Realistically, we never have efficient communication, since often times in the real world we have clusters of computers, each with its own private bank of memory. In these cases, divide and conquer is often impractical. It is true that when our data are split across multiple machines, having an algorithm operate on blocks of data at a time can be useful. However, as Strassen's algorithm continues to chop up matrices into smaller and smaller chunks, this places a large communication burden on distributed set ups because after the first iteration, it is likely that we will incur a shuffle cost as we are forced to send data between machines."

3. Pan

Although slower than Strassen's, Victor Pan, an American mathematician and computer scientist, and a few others used Strassen's idea to simultaneously compute two products of matrices. Instead of 2×2 matrices, let us partition matrices into 13×13 blocks.

Proof. Let A, B, C, A', B', C' be partitioned into 13×13 blocks. $C = AB$ and $C' = A'B'$ can then both simultaneously be calculated as the following. First, let

$$\begin{aligned} e_{ikj} &= (A_{ik} + A'_{kj})(B_{kj} + B'ji) \\ f_{kj} &= A'_{kj}B_{kj} \\ g_{ij} &= \left(\sum_{k=1}^{13} (A_{ik} + A'_{kj}) \right) B'_{ji} \\ h_{ki} &= A_{ik} \left(\sum_{j=1}^{13} (B_{kj} + B'_{ji}) \right). \end{aligned}$$

Then, we can find C and C' like the following:

$$\begin{aligned} C_{ij} &= \sum_{k=1}^{13} (e_{ikj} - f_{kj}) - g_{ij} \\ C'_{ki} &= \sum_{j=1}^{13} (e_{ikj} - f_{kj}) - h_{ki}. \end{aligned}$$

Note that calculating two products of 13×13 matrices by the standard algorithm with time complexity $O(N^3)$ is equivalent to $13^3 * 2 = 4394$ multiplications. However, with this algorithm, the number of multiplications necessary is only $13^2 + 3 * 13^2 = 2704$. Similar to Strassen's algorithm, by continuously dividing each matrix into submatrices of 13×13 blocks and recursively applying the formula above, we achieve a time complexity of $O\left(n^{\log_{13} \frac{2704}{2}}\right) = O\left(n^{2.811}\right)$. □

4. Approximation

Just like how sometimes probabilistic algorithms are faster, could approximating the product of matrices lead to a faster time complexity? The answer is yes. Let us first look at the following definition.

Definition 4.1 (Bilinear Operator)**.** An operator * is a bilinear operator if it satisfies the following conditions:

- $(A + A') * B = A * B + A' * B$
- $A * (B + B') = A * B + A * B'$
- $cA * B = A * cB = c(A * B)$

for all A, B, A', B', and some constant c.

Instead of 2×2 and 13×13 matrices, let us look at 3×3 matrices. Given 3×3 matrices A and B, we can approximate the product C using 21 multiplications. To do so, first, let

$$\begin{aligned}
e_{ij} &= \left(\epsilon^2 A_{i1} + A_{j3}\right)\left(B_{1j} + \epsilon B_{3i}\right) \\
f_{ij} &= \left(\epsilon^2 A_{i2} + A_{j3}\right)\left(B_{2j} - \epsilon B_{3i}\right) \\
g_j &= A_{j3}\left(B_{1j} + B_{2j}\right) \\
e_{ii} &= \left(\epsilon^2 A_{i1} + A_{i3}\right) B_{1i} \\
f_{ii} &= \left(\epsilon^2 A_{i2} + A_{i3}\right)\left(B_{2i} + \epsilon^2 B_{3i}\right).
\end{aligned}$$

Note that here, ϵ is an infinitesimal number. We then find that $C_{ij} = \frac{1}{\epsilon^2}(e_{ij} + f_{ij} - g_j) + \frac{1}{\epsilon}(e_{ij} - e_{ii})$. Like the previous two algorithms, we can recursively divide the matrix into 9 submatrices and apply the formulas above. Let * denote the approximated product using the method above. Then, * is a bilinear operator. Thus, for our original $O(N^3)$ algorithm we can replace each multiplication with *, and since $C_{ij} = \sum_{k=1}^{3}(A_{ik} * B_{kj}) \pmod{\epsilon}$, we have $C = AB = A * B \pmod{\epsilon}$, as desired.

5. Faster Algorithms and Schur's Complement

How have mathematicians found a faster algorithm from here? To understand how we would need to understand the Coppersmith-Winograd algorithm and the laser method. We would also have to understand how high tensor powers work. Note that faster multiplication methods can also be applied to other properties of matrices, such as finding their inverse. Similar to Strassen's, it has been found that

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1}. \end{pmatrix}$$

Like Strassen's, we can recursively find the inverse of the Schur's Complement, namely $D - CA^{-1}B$, and use fast matrix multiplication to successfully find the inverse.

References

[1] "Matrix Multiplication: Strassen's Algorithm." Stanford University, http://stanford.edu/rezab/dao/

[2] Sipser, Michael. Introduction to the Theory of Computation. 3rd ed. Boston, MA: Thomson Course Technology, 2013.

[3] Chan, Timothy. "Notes on Fast Matrix Multiplication" The Grainger College of Engineering, Illinois, 2020.

[4] "Fast Matrix Multiplication: Limitations of the Laser Method." University of Latvia, http://www.cs.toronto.edu/yuvalf/AmbFilLeG14.pdf.