Matrix Theory: Faster multiplication[*]

You are thrown into a prison, and told by the prison warden that you will be freed once you compute the product $M\vec{a}$, where M is an n-by-n matrix, and $\vec{a}$ is a vector in $\mathbb{R}^n$. What is the fastest way to gain freedom?

The obvious way is to compute the product using the formula

$$(M\vec{a})_i = \sum_k m_{i,k} a_k \qquad \text{for } i = 1, \ldots, n.$$

This requires n^2 multiplications and approximately the same number of additions. Since multiplication is harder than addition, the bulk of our work will consist of the n^2 multiplications. Can we do the computation using fewer than n^2 operations? No, because each entry of the matrix M is important, and there are n^2 of them.

What if the warden was more evil, and wanted you to compute $M\vec{a}_1, M\vec{a}_2, \ldots, M\vec{a}_n$, i.e., not just one, but n different products! What would you do? Of course, you could compute each of $M\vec{a}_1, \ldots, M\vec{a}_n$ is turn, for the total of $n \cdot n^2 = n^3$ multiplications. However, it turns out that one can do away with fewer multiplications.

Note that the problem is really to compute the product MA where A is the n-by-n matrix obtained by concatenating the columns $\vec{a}_1, \ldots, \vec{a}_n$. Let's pretend that n is even for simplicity[1], which allows us to divide each of the matrices M and A into four $n/2$-by-$n/2$ submatrices

$$M = \left(\begin{array}{c|c} M_{1,1} & M_{1,2} \\ \hline M_{2,1} & M_{2,2} \end{array}\right) \qquad A = \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right).$$

The key insight is that to multiply M and A, one can pretend that M and A are 2-by-2 matrices with entries from $M_{n/2,n/2}$ and use the usual rules of matrix multiplication (can you see why?). In other words,

$$MA = \left(\begin{array}{c|c} M_{1,1}A_{1,1} + M_{1,2}A_{2,1} & M_{1,1}A_{1,2} + M_{1,2}A_{2,2} \\ \hline M_{2,1}A_{1,1} + M_{2,2}A_{2,1} & M_{2,1}A_{1,2} + M_{2,2}A_{2,2} \end{array}\right).$$

This way our task reduces to 8 multiplications of $n/2$-by-$n/2$ matrices, and some additions. As we know from the above, the multiplication of $n/2$-by-$n/2$ matrices can be done using $(n/2)^3$ multiplications, and so the total number of multiplications using this method is $8(n/2)^3 = n^3$. So, this method appears to be no better than the naive method.

However, it is possible to multiply two 2-by-2 matrices using only 7 multiplications! Instead of writings the cumbersome formula with 7 multiplications, we shall first look at a

[*]These notes are available from the course webpage, and directly from `http://www.borisbukh.org/MatrixTheory13/notes_fast_multiplication.pdf`

[1]Of course, the prison warden is evil, and so he will choose n to be odd. Can you see what to do then?

simpler, but related problem: integer multiplication. Imagine that instead of multiplying matrices, the warden asked you to multiply to n-digit integers, A and B. The obvious multiplication method would require multiplying each digit of A by each digit of B, and then adding the results; that would take n^2 multiplications. A faster way splits A and B into two $n/2$-digit long numbers

$$A = \overline{A_1 A_0} \qquad \text{and} \quad B = \overline{B_1 B_0},$$

where the line over A_1A_0 and B_1B_0 signifies concatenation, i.e.,

$$A = A_1 10^{n/2} + A_0 \quad \text{and} \quad B = B_1 10^{n/2} + B_0.$$

The product of A and B is then

$$AB = A_1B_1 10^n + (A_0B_1 + A_1B_0)10^{n/2} + A_0B_0.$$

We have reduced multiplication of two n-digit numbers to 4 multiplications of $n/2$-digit numbers. Since we can multiply $n/2$-digit numbers using $(n/2)^2$ ordinary multiplications, this gives us a method to multiply the n-digit numbers using $4(n/2)^2 = n^2$ multiplications, which is no better than what we already could do. However, we can compute AB using only three multiplications. Namely, we compute

$$\begin{aligned} &A_0B_0, \\ &A_1B_1, \\ &(A_0 + A_1)(B_0 + B_1). \end{aligned}$$

To obtain $A_0B_1 + A_1B_0$ we just do a few extra additions,

$$A_0B_1 + A_1B_0 = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1.$$

How many multiplication do we do with this method? Computation of A_0B_0 and A_1B_1 each takes $(n/2)^2$ multiplications, whereas $(A_0+A_1)(B_0+B_1)$ might take $(n/2+1)^2$ multiplication since $A_0 + A_1$ and $B_0 + B_1$ might have as many as $n/2 + 1$ digits. That adds up to $2(n/2)^2 + (n/2+1)^2 = \frac{3}{4}n^2 + n + 1$ multiplications! This method is thus faster than then the usual method for large n. For even greater speed-up, one can use this method to compute products of $n/2$-digit numbers instead of using the slower algorithm.

So, what about the evil warden and product of matrices MA? Here are the 7 products that we compute:

$$\begin{aligned} P_1 &= (M_{1,1} + M_{2,2})(A_{1,1} + A_{2,2}) \\ P_2 &= (M_{2,1} + M_{2,2})A_{1,1} \\ P_3 &= M_{1,1}(A_{1,2} - A_{2,2}) \\ P_4 &= M_{2,2}(A_{2,1} - A_{1,1}) \\ P_5 &= (M_{1,1} + M_{1,2})A_{2,2} \\ P_6 &= (M_{2,1} + M_{1,1})(A_{1,1} + A_{1,2}) \\ P_7 &= (M_{1,2} - M_{2,2})(A_{2,1} + A_{2,2}). \end{aligned}$$

Then

$$MA = \left(\begin{array}{c|c} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ \hline P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{array}\right).$$

Historical remarks. The 3-multiplication method for the integer multiplication is due to Anatolii Karatsuba. It was invented in 1960. While it might seem like an eternity ago to most of us, this means that for the first few thousand years humans used an inferior way to multiply integers! The 7-multiplication method is due to Volker Strassen from 1969.

The idea of Karatsuba has been extended, and there is now an algorithm to multiply two n-digit numbers that takes almost $n \log n$ operations for very large values of n. The algorithm is based on the linear transformation called *Fourier transform*, whose description is beyond the scope of this course. For matrices, the ideas based on non-commutative Fourier transform lead to the fastest known algorithms (for large n), but it is still not known if it is possible to multiply two n-by-n matrices in as few as n^{2+c} operations for any desired value of $c > 0$.