Fast Matrix Multiplication and Inversion
Notes for Math 242, Linear Algebra, Lehigh University fall 2008

We describe a method which is theoretically faster for computing the product of two square matrices than a direct computation using the definition of matrix multiplication. We will also show, somewhat surprisingly, that one can also compute the inverse of a matrix with a number of computations that is not markedly different from the time for matrix multiplication (what this means will be made clear). These methods are of interest theoretically but the overhead involved usually does not make them useful in applications. Although some methods like these have been used efficiently for extremely large matrices.

If A is $m \times n$ and B is $n \times p$ then the product has size $m \times p$ (i.e., it has mp entries) and each entry is given by the formula for matrix multiplication

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Determining this sum requires n multiplications and $n-1$ additions. Thus overall we get the product AB with mnp multiplications and $m(n-1)p$ additions. Note also that adding two $n \times m$ matrices requires nm additions, one for each of the nm position.

In particular, multiplying two $n \times n$ matrices requires at least n^3 operations using the 'usual' method from the definition.

It seems like it should not be possible to do better than this, but V. Strassen in 1967 described a recursive procedure which does. (There are now even faster algorithms, which we not discuss here.)

The key is to note that multiplying two 2×2 matrices ordinarily requires 8 multiplications. By finding a way to do this with only 7 multiplications and applying this recursively to blocks we can get a faster (at least in theory) algorithm. For 2×2 matrices this procedure requires 18 additions as compared to 4 for usual approach but because of the way the cost of addition appears in the recursion this additional cost does not create a problem. If

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

define

$$\begin{array}{ll} p_1 = (a_{11}+a_{22})(b_{11}+b_{22}) & p_5 = (a_{11}+a_{12})b_{22} \\ p_2 = (a_{21}+a_{22})b_{11} & p_6 = (a_{21}-a_{11})(b_{11}+b_{12}) \\ p_3 = a_{11}(b_{12}-b_{22}) & p_7 = (a_{12}-a_{22})(b_{21}+b_{22}) \\ p_4 = a_{22}(b_{21}-b_{11}) & \end{array}$$

Then

$$AB = \begin{pmatrix} p_1+p_4-p_5+p_7 & p_3+p_5 \\ p_2+p_4 & p_1+p_3-p_2+p_6 \end{pmatrix}$$

We have now found AB with 7 multiplications and 18 additions instead of the usual 8 multiplications and 4 additions.

Let us begin by assuming that we wish to multiply two $n \times n$ matrices with $n = 2^k$, a power of 2. Partition the matrices into 4 blocks of size $2^{k-1} \times 2^{k-1}$. So in the formula above the a_{ij} and b_{ij} are now $2^{k-1} \times 2^{k-1}$ matrices. The matrix multiplications in the formula can be carried out recursively, applying the same procedure to the smaller matrices.

Let $T(2^k)$ denote the number of operations (both multiplication and addition) required to multiply using this procedure. Then the procedure uses 7 block multiplications and 7 block additions of $2^{k-1} \times 2^{k-1}$ matrices. Each of the 7 multiplications requires $T(2^{k-1})$ operations. Each of the 18 additions requires $2^{k-1}2^{k-1} = 4^{k-1}$ operations (all of which happen to be additions). This gives the following recursive formula

$$T(2^k) = 7T(2^{k-1}) + 18(4^{k-1})$$

with the initial condition that $T(2^1) = 12$ obtained by using the usual procedure for multiplying two 2×2 matrices. (Note, for the procedure we use the 'new' multiplication for blocks but when we reduce down to the 2×2 case we revert to usual matrix multiplication.)

By repeated substitution one can determine that
$T(2^k) = (36/7)7^k - 6(4^k) = 36(7^{k-1}) - 6(4^k)$. Once we discover this, it is straightforward to prove it is correct using induction. The basis is $12 = T(2^1) = 36(7^0) - 6(4^1) = 36 - 24$. Now, for $k \geq 2$ assume that $T(2^{k-1}) = 36(7^{k-2}) - 6(4^{k-1})$ (the formula holds for $k-1$) and use this to show that $T(2^k) = 36(7^{k-1}) - 6(4^k)$ (the formula holds for k). Using the recursion formula we have

$$\begin{aligned} T(2^k) &= 7T(2^{k-1}) + 18(4^{k-1}) \\ &= 7[36(7^{k-2}) - 6(4^{k-1})] + 18(4^{k-1}) \\ &= 36(7^{k-1}) - (-42 + 18)(4^{k-1}) \\ &= 36(7^{k-1}) - 6(4^k) \end{aligned}$$

and by induction the formula is proved.

In order to translate this into terms of n, we use a basic fact from logarithms, $a^{\log_c b} = b^{\log_c a}$. With $n = 2^k$, that is, $k = \log_2 n$ we get

$$T(n) = \frac{36}{7}7^k - 6(4^k) \leq \frac{36}{7}7^{\log_2 n} = \frac{36}{7}n^{\log_2 7}.$$

With $\log_2 7$ approximately 2.807, the running time is bounded by a multiple of $n^{2.81}$ which for large enough n is less than the n^3 needed for ordinary matrix multiplication. In practice the extra implementation costs do not make this a practical algorithm except for some very large instances.

We have not said what to do if the matrix sizes are not a power of 2. Simply add enough rows and columns of zeroes at the end of the matrices. This at most doubles the size and thus the running time is at most $(36/7)(2n)^{\log_2 7}$ which is still bounded by a multiple of $n^{2.81}$.

Inversion and matrix multiplication

In order to talk more easily about running times of algorithms we need to introduce the big oh notation.

Given two real non-negative functions $f(n)$ and $g(n)$ we say that $f(n)$ is $O(g(n))$ (read as f is oh of g) if there exist non-negative constants C, M such that for $n \geq M$ we have $f(n) \leq Cg(n)$. Roughly speaking, for large enough n f is bounded by a multiple of g.

For algorithms if $f(n)$ and $g(n)$, as function of the input size n are such that $f(n)$ is $O(g(n))$ then in some sense then running time for f is not much worse than that for g. It is longer by at most a constant multiple independent of the input size.

As noted before, we can compute the inverse (if it exists) of matrix A by solving $A\boldsymbol{x} = \boldsymbol{e_i}$ for $i = 1, 2, \dots n$. This can be done in $O(n^3)$ steps using Gaussian elimination. Our aim here is to show that we can compute inverses 'as quickly' as we can multiply two matrices (in the sense of big oh) and (less surprisingly) conversely.

If we can invert an $n \times n$ matrix in time $I(n)$ then we can multiply two $n \times n$ matrices in time $O(I(n))$.
If we can multiply two $n \times$n matrices in time $M(n)$ then we can invert an $n \times n$ matrix in time $O(M(n))$.

Assume that we can invert an $n \times n$ matrix in $I(n)$ steps. We will show that we can multiply two $n \times n$ matrices in $O(I(n))$ steps. Informally what we are showing is that if we have a subroutine to invert matrices we can use it to multiply two matrices (so without doing ordinary matrix multiplication). Imagine being stuck in a reality TV show for which you must multiply two matrices but the only 'tool' you have is something to invert a matrix.

Let A and B be two $n \times n$ matrices for which we wish to find the product AB. Let D be the following block matrix (with 0 indicating $n \times n$ matrices of 0's):

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}.$$

A straightforward block multiplication shows that

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}.$$

Thus, finding D^{-1} allows us to determine AB without explicitly carrying out a matrix multiplication (i.e., using inversion instead).

Now, assume that we can multiply two $n \times n$ matrices with an algorithm having running time $M(n)$. We will show that we can find the inverse of an $n \times n$ matrix with an algorithm having running time $O(M(n))$. For what follows to work we must assume that for for some constant C, we have $Cn^3 \geq M(n) \geq n^2$. Since it takes n^2 steps just to write down the answer the lower bound is a reasonable assumption. As we will later see a direct $O(n^3)$ algorithm for matrix inversion (using Gaussian elimination) the upper bound covers values of interest.

We will assume that n is a power of 2. If it was not, let 2^k be the smallest power of 2 greater than n and let $h = 2^k - n$. Note that $h \leq n$. It is easy to check that

$$\begin{pmatrix} A & 0 \\ 0 & I_h \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_h \end{pmatrix}.$$

If the running time is $O(M(n))$ for n a power of 2, it is at most $C_1M(n)$ for $n \geq M$ (for some C_1, M). Then for other $n \geq M$ we have running time at most $C_1M(n+h) \leq C_1M(2n) \leq 8C_1M(n)$ and thus we still have $O(M(n))$ (we have just changed the constant in the big Oh definition). Here we have assumed that the running time is non-decreasing in n and used $M(2n) \leq 2^3 M(n)$ from the assumption that $M(n) \leq n^3$.

We also make several other assumptions that are less obvious. A square matrix B is symmetric if it is equal to its transpose, $B^T = B$. A symmetric matrix B is positive definite if $\boldsymbol{x}^T B \boldsymbol{x} > 0$ for all nonzero vectors $\boldsymbol{x}$. Observe that the product in the previous sentence is a number. It can be shown that this is equivalent to B having a Cholesky factorization $B = MM^T$ where M is lower triangular with positive entries on the diagonal.

We assume that the matrix A that we wish to invert is symmetric and positive definite. To see that we can make this assumption we use the fact that $A^T A$ is symmetric and positive definite. The first statement is an easy exercise and the second follows since $\boldsymbol{x}^{\boldsymbol{T}}(A^T A)\boldsymbol{x} = (A\boldsymbol{x}) \cdot (A\boldsymbol{x})$ This is the square of the length of the vector $(A\boldsymbol{x})$ and if A is invertible then $(A\boldsymbol{x}) = \mathbf{0}$ only if $\boldsymbol{x} = \mathbf{0}$ as there is a unique solution when the inverse exists. So this is positive for all non-zero $\boldsymbol{x}$ and thus $A^T A$ is positive definite. Finally observe that if A^{-1} exists we have $A^{-1} = (A^T A)^{-1} A^T$ (check this by right multiplying by A). So using two more multiplications we can determine A^{-1} by first determining $(A^T A)^{-1}$ which is symmetric and positive definite if A is invertible.

Now partition A as follows, with B, C, D all of size $n/2 \times n/2$:

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

(Here we have used the fact that A is symmetric.) Let $S = D - CB^{-1}C^T$ (called the Schur complement). Then by a straightforward (but lengthy) block multiplication we can check that

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^T S^{-1} C B^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}.$$

One can also check that the assumption that A is symmetric and positive definite imply that B and S are also symmetric and positive definite (and thus also, as we will see later, that B^{-1} and S^{-1} exist). So we recursively find B^{-1} and S^{-1} (using matrix multiplication). Then, again using matrix multiplication compute CB^{-1}, then $CB^{-1}C^T$ then $S^{-1}CB^{-1}$ then $(CB^{-1})^T(S^{-1}CB^{-1}) = B^{-1}C^T S^{-1} C B^{-1}$ (since we have $B^{-1}C^T = (CB^{-1})^T$ from the fact that B^{-1} is symmetric, which follows from B symmetric). Also, since B^{-1} and S^{-1} are symmetric $(S^{-1}CB^{-1})^T = B^{-1}CTS^{-1}$. Thus we can determine A^{-1} using 2 inversions of $n/2 \times n/2$ matrices and 4 multiplications of $n/2 \times n/2$ matrices and $O(n^2)$ operations for transposes and an addition. Hence the running time satisfies

$$I(n) \leq 2I(n/2) + 4M(n/2) + O(n^2).$$

If we inductively assume that $I(n/2)$ is $O(M(n/2))$ then we get (with some computations which we will not show) that $I(n)$ is $O(M(n))$.