

Matrix multiplication

We will be concerned with multiplying two $n \times n$ matrices.

How long does it take?

Classical method.

Let $X = AB$, where $A = (a_{ij})$, $B = (b_{ij})$ and $X = (x_{ij})$. Then

$$x_{ij} = \sum_{k=1}^n a_{ik}b_{kj}.$$

This takes n^3 multiplications and $n^2(n - 1)$ additions. Total time $\Theta(n^3)$.

2×2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

That is, 8 multiplications and 4 additions.

Matrix multiplication – Strassen's method

Define

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

Then

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}.$$

That is, 7 multiplications and 18 additions.

Strassen's method (continued)

To multiply two $n \times n$ matrices A and B , partition them into blocks of size $n/2 \times n/2$:

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right) \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array} \right)$$

Then use Strassen's method to multiply them. This takes 7 **matrix** multiplications and 18 **matrix** additions. Since matrix multiplications are much more expensive than matrix additions, saving one multiplication more than compensates for the extra additions.

Furthermore, these 7 matrix multiplications of size $n/2$ can be done by the same method. The total time is given by the recurrence

$$T(n) = 7T(n/2) + \Theta(n^2).$$

Solution: $T(n) = \Theta(n^{\lg 7}) \approx \Theta(n^{2.81})$.

Strassen's method (continued)

- Strassen's method beats the classical method in practice for n larger than about 40–100.
- Theoretically faster methods exist, such as $O(n^{2.376})$ but they are not believed to be practical.
- The same method can be used to speed up finding the inverse of a matrix, solving linear equations, least-squares problems, linear programming, and lots of other things.
- Nobody knows what is the smallest β such that matrix multiplication can be done in time $O(n^\beta)$. Maybe β can be arbitrarily close to 2, but nobody can prove it.

RSA Public Key cryptosystem

A. Bob establishes his public key.

1. Bob selects two large prime numbers p, q and computes $n = pq$.
2. Bob finds e, d such that $ed \equiv 1 \pmod{(p-1)(q-1)}$.
3. Bob publishes the **public key** $P = (e, n)$, and destroys p, q .
4. Bob keeps secret the **private key** $S = (d, n)$.

B. Alice sends a message M (an integer mod n) to Bob that only Bob can read.

1. Alice computes $C = M^e \pmod n$ and sends C to Bob.
2. Bob computes $M' = C^d \pmod n$.

The point: $M' = M$ (ie. Bob has recovered the original message)

RSA Public Key cryptosystem (continued)

Fermat's theorem: If p is prime, then $a^{p-1} \equiv 1 \pmod{p}$ for all $a \neq 0$.

Chinese remainder theorem: If p and q are different primes, then for all x, y ,
 $x \equiv y \pmod{p}$ and $x \equiv y \pmod{q} \Leftrightarrow x \equiv y \pmod{pq}$.

Why the RSA method works.

1. Alice sends $C = M^e \pmod{n}$ and Bob computes $M' = C^d \pmod{n}$.
2. Therefore $M' \equiv M^{ed} \pmod{n}$.
3. Since $ed \equiv 1 \pmod{(p-1)(q-1)}$, we have $ed = 1 + k(p-1)(q-1)$ for some k .
4. Therefore, modulo p , $M' \equiv M^{1+k(p-1)(q-1)} \equiv M \times M^{k(p-1)(q-1)} \equiv M \times (M^{k(p-1)})^{q-1} \equiv M \times 1^{q-1} \equiv M$.
5. Similarly, $M' \equiv M \pmod{q}$.
6. Therefore, $M' \equiv M \pmod{pq}$. Recall that $n = pq$.

RSA Public Key cryptosystem (continued)

Example.

1. Bob chooses $p = 8765417$ and $q = 41637511$, and computes $n = pq = 364970146757087$.
2. Bob finds $e = 19$ and $d = 211298476836619$ satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$ (can be done using the Euclidean algorithm).
3. Bob **publishes** $(e, n) = (19, 364970146757087)$ and **keeps secret** $(d, n) = (211298476836619, 364970146757087)$.

Now Alice wants to send message $M = 258818273883381$.

1. Alice computes $C = 258818273883381^e \pmod n = 96553210852936$ and sends C to Bob.
2. Bob computes $96553210852936^d \pmod n = 258818273883381 = M$.

RSA Public Key cryptosystem (continued)

- To crack the encryption, one can try to extract the e -th root of C (nobody knows how), or to recover p, q by factoring n . If n is hundreds of bits long, factoring is too hard by current methods.
- The algorithm is symmetrical in d and e . If Bob publishes the number $C = (\text{"Bob"}_in_ASCII)^d \bmod n$, then anyone can check that Bob must have produced it, since $C^e \bmod n$ is "Bob" in ASCII. This is a **digital signature**.
- In practice RSA is too slow for encrypting large quantities of data. Instead, it is used to securely exchange secret keys for a faster encryption algorithm.
- If quantum computers ever become reality, they might make RSA obsolete by making integer factorisation easy.