ADD#1，

2023/03/01 24:00

- 矩陣相乘的前世今生
  - 請寫一篇關於矩陣相乘的報告
  - 中文寫，大概是一千字。
  - 請轉存為 **add1.pdf**
  - 請 commit 到 github

矩陣相乘的前世今生

數學上，矩陣乘法為以下的操作：
有三個矩陣：

$$\mathbf{A}_{mn} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$ $\mathbf{A}_{mn}$為m個 row（列）、n個 column（行）的矩陣

$$\mathbf{B}_{np} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{np} & \cdots & b_{np} \end{pmatrix}$$ $\mathbf{B}_{np}$為n個 row（列）、p個 column（行）的矩陣

$$\mathbf{C}_{mp} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$ $\mathbf{C}_{mp}$為m個 row（列）、p個 column（行）的矩陣

當兩個矩陣A、B滿足前者A的行數（column）等於後者B的列數（row），則可相乘得到矩陣C，記作：

$$\mathbf{AB} = \mathbf{C}$$

即：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

其中矩陣C的列數（row）為矩陣A的列數、矩陣C的行數（column）為矩陣B的行數。其運算本質為將兩個矩陣的元素依照下述方式相乘：

$$c_{ij} = a_{ik} \cdot b_{kj}$$

以下挑選幾個例子說明：

$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + \cdots + a_{1n} \cdot b_{n1}$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

$$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + \cdots + a_{2n} \cdot b_{n1}$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

$$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + \cdots + a_{1n} \cdot b_{n2}$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

矩陣相乘，一直是許多中學生的噩夢，因為耗時費力，以下將仔細估算到底有多費勁？

任何一個矩陣C的元素c_{ik}，都是n次乘法與$n-1$次加法，故總共要進行$m \cdot p$次這樣的運算，總計乘法nmp次、加法$(n-1)mp$次，由於乘法相對加法要慢很多（消耗必較多資源），故可用乘法作為評估矩陣相乘方法的指標。

以下為方便討論，以$n \times n$矩陣（n階方陣）為討論標的。

傳統方法 naive：$\mathcal{O}(n^3)$

如前面所演示的傳統做法，需要進行n^3次乘法，也就是說以前高中最常計算的3×3矩陣其實包含了$3^3 = 27$次的乘法（以及$2 \times 3^2 = 18$的加法），難怪很容易出錯！

1969, Strassen's algorithm：$\mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.807})$

矩陣的乘法其實有很多重複的部分，以最簡單的 2 階方陣為例：

$$\text{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{bmatrix}$$

其中包含$2^3 = 8$次元素之間的乘法。在電腦科學中，使用晶片來做計算有乘法嚴重慢

於加法的問題，兩者計算量差異甚大，故只要能夠把乘法以加法來取代，就可以做到計算上的加速！如同我們在做四則運算的時候，只要把原本要分開相乘的項進行適當的合成（加法）或拆解（減法），就可以「大幅度」的提升計算速度，舉例來說：

$$23 \times 79 + 77 \times 79 = (23 + 77) \times 79 = 100 \times 79 = 7900$$

相同的精神就應用在 Strassen 演算法上：用 18 個矩陣加法取代 1 個矩陣乘法，以下是 Strassen 演算法的數學模式：

由於矩陣乘法不存在上述簡單乘法中「直接」可以觀察到的共用項，但這不代表矩陣乘法不能這麼做（只是不容易觀察到而已！）Strassen 的一個偉大貢獻就是把這個隱藏的關係找出來，在他給的演算法中，需要先計算七個中間項（暫存變數）：

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$m_2 = (a_{21} + a_{22})b_{11}$$
$$m_3 = a_{11}(b_{11} - b_{22})$$
$$m_4 = a_{22}(b_{21} + b_{11})$$
$$m_5 = (a_{11} + a_{12})b_{22}$$
$$m_6 = (a_{21} - a_{11})(b_{11} + b_{22})$$
$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

然後，經過一系列加減就可以得到所求答案：

$$c_{11} = m_1 + m_4 - m_5 + m_7$$
$$c_{12} = m_3 + m_5$$
$$c_{21} = m_2 + m_4$$
$$c_{22} = m_1 - m_2 + m_3 + m_6$$

1971, Schönhage–Strassen algorithm：$\boldsymbol{O}\left(\boldsymbol{n}^{2.522}\right)$

Schönhage–Strassen 演算法以整數乘法的 FFT（快速傅立葉轉換）為基礎，進行一系列的數論轉換，數學理論對於一般人來說複雜難懂，但其成果是得以加速計算處理超大數的矩陣乘法，且提示後進這條路上竟然還有得以改進的空間。

結語

在 Strassen (1969)演算法第一次帶來勁爆的衝擊之後，數學界陸續有新的演算法被提出來，每一次革新的背後，都是高深的數學理論，雖然不一定能帶來全面性的實質加速（例如 Fürer 演算法雖然理論上優於 Schönhage–Strassen 演算法，但只有在處理極大的數時，才有作用，故實務上 Fürer 演算法並未使用），但旦特定領域卻可能有他實用的地方。

矩陣乘法不只是算法上的加速，還有精確度問題，這源自於浮點數的計算誤差，若演

算法在實作（編寫成可以執行的程式碼）時沒有妥善設計，可能會得到很糟的結果。

除了上述兩點（算法加速、精確度），實務上矩陣乘法很常遇到稀疏矩陣，在這種情況下， 還有另外針對性的演算法可以大幅簡化計算。