Matrix Operations

October 18, 2014

1 Introduction

In this section, we look at some of the basic operations when looking at different matrix operations. In specific, we are going to look at the following operations:

- The multiplications
- The inverse

2 Matrix Multiplications

In this section, we look at the cost of making a matrix multiplication.In specific, the Strassen's algorithm which was the first algorithm to prove that $O\left(n^3\right)$ is not the best complexity for matrix multiplications. This upper bound was believe correct for the matrix multiplication because the nature of the definition.

Definition 1. Given A, B matrices with dimensions $n \times n$, the multiplication is defined as:

$$\begin{aligned} C &= AB \\ c_{ij} &= \sum_{k=1}^{n} a_{ik} b_{kj} \end{aligned}$$

Thus the final algorithm is

Algorithm 1 Matrix multiplication

```
Square-Matrix-Multiply(A,B)
n = A.rows
let C be a new matrix of nxn
for i = 1 to n
        for j = 1 to n
                C[i,j] = 0
                for k = 1 to n
                        C[i,j] = C[i,j]+A[i,j]*B[i,j]
return C
```

2.1 Strassen's Algorithm

The Strassen's algorithm is a divide and conquer algorithm which split the three matrices involved in the matrix algorithm in the following way:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} \tag{1}$$

Thus, we have then the following:

$$r = a \times e + b \times g, \ s = a \times f + b \times h$$

$$t = c \times e + d \times g, \ u = c \times f + d \times h$$

This has the following recursion and complexity, $T(n) = 8T\left(\frac{n}{2}\right) + \Theta\left(n^2\right)$ and $T(n) = \Theta\left(n^3\right)$ respectively.

Basically the Strassen's algorithms has the following steps:

Algorithm 2 Strassen's Algorithm

---

1. Divide the input matrices A and B into $\frac{n}{2} \times \frac{n}{2}$ sub matrices
2. Using $\Theta\left(n^2\right)$ scalar additions and subtractions, compute 14 matrices $A_1, B_1, ..., A_7, B_7$ each of which is $\frac{n}{2} \times \frac{n}{2}$.
3. Recursively compute the seven matrices products $P_i = A_i B_i$ for $i = 1, 2, 3, ..., 7$.
4. Compute the desired matrix

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

by adding and or subtracting various combinations of the P_i matrices, using only $\Theta\left(n^2\right)$ scalar additions and subtractions

---

At the slides you can see an attempt of how the algorithm could have been designed.

In any case, Strassen showed that the upper bound of $O\left(n^3\right)$ is not the last bound. It is more, it has been shown recently in 2012 that the possible bound is at $O\left(n^2\right)$.

3 Solving systems of linear equations

In many areas of engineering and mathematics (Numerical analysis, differential equations, etc) there is a need to develop a solution for systems of equations:

$$
\begin{aligned}
a_{11}x_1 + ... + a_{1n}x_n &= b_1 \\
a_{21}x_1 + ... + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + ... + a_{nn}x_n &= b_n
\end{aligned}
$$

For this, we can rewrite the systems of equations into a matrix-vector equation:

$$
\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{2}
$$

This can be solved by using the inverse matrix operation by simply looking at the following equation:

$$
Ax = b \Longrightarrow x = A^{-1}b \tag{3}
$$

Clearly, we are looking at the cases when the matrix A is not singular.

Definition 2. A square matrix that is not invertible is called singular or degenerate. A square matrix is singular if and only if its determinant is 0.

Example 3. We can have systems of differential equations of first order:

$$
\begin{aligned}
a_{11}x_1 + ... + a_{1n}x_n &= \frac{dx_1}{dt} \\
a_{21}x_1 + ... + a_{2n}x_n &= \frac{dx_2}{dt} \\
&\vdots \\
a_{n1}x_1 + ... + a_{nn}x_n &= \frac{dx_n}{dt}
\end{aligned}
$$

We can solve this system if we have initial conditions for the differentials.

The problem with the previous methods is the inherent instability and high complexity of simply calculating A^{-1}. Thus,we require something more stable and faster.

4 LUP Decomposition

The idea behind LUP decomposition is to find three $n \times n$ matrices L, U, and P such that

$$PA = LU \tag{4}$$

Each is called

- L is a unit lower-triangular matrix.
- U is an upper-triangular matrix.
- P is a permutation matrix.

Example 4. A lower-triangular matrix look like

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Example 5. A upper looks like

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

The final example is about the permutation matrix:

Example 6. For order three, we have something like:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Thus, we can solve $Ax = b$ once we found the decomposition by using the following substitutions:

$$\begin{aligned} PAx &= Pb \\ LUx &= Pb \end{aligned}$$

Now by making $y = Ux$, we have that

$$Ly = Pb \tag{5}$$

Which is a lower triangular system. Thus, we only need to solve the system the solve the system $y = Ux$. This is called forward substitution. After that we can solve the upper-triangular system:

$$Ux = y \tag{6}$$

By the back-substitution method. All this is true because P is invertible, which allows to have the following equality

$$A = P^{-1}LU \tag{7}$$

Or in other words:

$$\begin{aligned} Ax &= P^{-1}LUx \\ &= P^{-1}Ly \\ &= P^{-1}Pb \\ &= b \end{aligned}$$

4.1 Fordward and Backward Substitution

In order to solve the lower triangular system in $\Theta\left(n^2\right)$, we use an algorithm called fordward substitution. It depends on the compact representation of the permutation P by using an array $\pi\left[1...n\right]$. Thus, each P_{ij} is defined as follows

$$P_{ij} = \begin{cases} 1 & \text{if } j = \pi\left[i\right] \\ 0 & \text{if } j \neq \pi\left[i\right] \end{cases} \tag{8}$$

Thus, PA has $a_{\pi[i],j}$ in row i and column j, and Pb has $b_{\pi[i]}$ as its ith element. This allows to have the following representation:

$$\begin{array}{ccc} y_1 & = & b_{\pi[1]} \\ l_{21}y_1 + y_2 & = & b_{\pi[2]} \\ l_{21}y_1 + l_{32}y_2 + y_3 & = & b_{\pi[3]} \\ & \vdots & \\ l_{n1}y_1 + l_{n2}y_2 + l_{n3}y_3 + \cdots + y_n & = & b_{\pi[n]} \end{array}$$

Then, we have the following solution for each y_i:

$$y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij}y_j \tag{9}$$

In a similar way, we have that for the upper triangular system can be rewritten as:

$$\begin{array}{ccc} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n & = & y_1 \\ u_{22}x_2 + \cdots + u_{2n}x_n & = & y_2 \\ & \vdots & \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n & = & y_{n-1} \\ u_{nn}x_n & = & y_n \end{array}$$

Thus,

$$x_i = \frac{\left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)}{u_{ii}} \tag{10}$$

Then, we have the following algorithm:

Algorithm 3 LUP-Solve

LUP-SOLVE(L, U, π, b)

$n = L.rows$
let x be a new vector of length n
for $i = 1$ **to** n
    $y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij} y_j$
for $i = n$ **downto** 1
    $x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j\right) / u_{ii}$
return x

5 Computing the LU decomposition

5.1 Case $P = I_n$

In this case, $A = LU$, a LU decomposition of A (Assuming that the matrix is non-singular). To obtain this decomposition we use the Gaussian elimination process:

- If $n = 1$, then we are done because $L = I_1$ and $U = A$.
- If $n > 1$ then:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{22} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix} \tag{11}$$

This can be decomposed further:

$$A = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - \frac{vw^T}{a_{11}} \end{pmatrix} \tag{12}$$

where the $(n-1) \times (n-1)$ matrix $A' - \frac{vw^T}{a_{11}}$ is the Schur complement of A with respect to a_{11} (Called pivots), which is not singular because A is not singular. Now recursively decompose it into:

$$A' - \frac{vw^T}{a_{11}} = L'U' \tag{13}$$

Thus:

$$A = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix} = LU \tag{14}$$

Then, we have that

Algorithm 4 LU-Decomposition

```
LU-DECOMPOSITION(A)
n = A.rows
let L and U be new n × n matrices
initialize U with 0s below the diagonal
initialize L with 1s on the diagonal and 0s above the diagonal
for k = 1 to n
    u_kk = a_kk
    for i = k + 1 to n
        l_ik = a_ik / u_kk      // l_ik holds v_i
        u_ki = a_ki             // u_ki holds w_i^T
    for i = k + 1 to n
        for j = k + 1 to n
            a_ij = a_ij − l_ik u_kj
return L and U
```

5.2 General Case

In this case, we use the following idea:

- Move the largest absolute value element a_{k1} to the position (1,1) in the matrix by using a permutation matrix Q to increase stability and avoid division by zero.

This allows to have the following without a division by zero:

$$QA = \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - \frac{vw^T}{a_{11}} \end{pmatrix} \tag{15}$$

And again

$$P' \left(A' - \frac{vw^T}{a_{11}} \right) = L'U' \tag{16}$$

Then the permutation P can be defined as

$$P = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} Q \tag{17}$$

The code is at the slides.

6 Inverting Matrices

The LUP decomposition allows to compute the inverse by simply looking at the following computation.

- Given $AX = I_n$, we can decompose the the LUP of A, then solve the following system $Ax_i = e_i$ for all $i = 1, ..., n$ using the LUP as follows
  - $PAx_i = Pe_i \Longrightarrow LUx_i = Pe_i$
  - Use Fordward to solve $L(Ux_i) = Ly_i = Pe_i$
  - Use Backward to solve $Ux_i = y_i$

7 Matrix Multiplication and Inversion Complexities

Theorem. *Multiplication no harder than inversion*

Note

- If $M(n)$ denotes the time for the multiplication of two matrices of $n \times n$.
- If $I(n)$ denotes the time of inverting a non-singular matrix of $n \times n$.

Proof. Let A and B be $n \times n$ matrices whose product is C. Define the following matrix

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

with inverse

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

It is possible to construct D in $\Theta\left(n^2\right)$ time which is $O(I(n))$ $\left(I(n) = \Omega\left(n^2\right)\right)$. Thus, inversion can be done in $O\left(I\left(3n\right)\right) = O\left(I\left(n\right)\right)$ by regularity condition on $I\left(n\right)$. Then $M\left(n\right) = O\left(I\left(n\right)\right)$. □

Theorem. *Inversion is no harder than multiplication.*

Proof. We let this to you. □