Prepared for submission to JCAP

Strassen's Matrix Multiplication

Mohammed Rizin [1][1]

[1]Umemployed, Navi Mumbai, Maharashtra

E-mail: mrizin2013@gmail.com

Abstract. Usual Matrix Multiplication takes $O(n^3)$. To avoid such a long computation, We adopt Strassen's Matrix Multiplication Algorithm

ArXiv ePrint: 3423.2342

1 Matrix Multiplication

We have been learning Matrix Multiplication since High School. Its a great tool. Matrix Multiplication can be said as dot product of row of one matrix with the columns of other matrix. The most commonly known algorithm for Matrix Multiplication is using three for loop. Here's the algorithm:

Algorithm 1.1 Simple Matrix Multiplication

```
procedure MatMul($A, B, n$)
    for $i \leftarrow 0$ to $n$ do
        for $j \leftarrow 0$ to $n$ do
            $C[i, j] \leftarrow 0$
            for $k \leftarrow 0$ to $n$ do
                $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
            end for
        end for
    end for
end procedure
```

The statement

$C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$

takes $O(n^3)$ because its inside 3 for loop. That the highest term. So this Algorithm take

$$\boxed{O(n^3)}$$

[1]He's Amazing

1.1 Matrix Multiplication with Divide and Conquer.

Divide and Conquer is the method of dividing a large problem into smaller same problem for which we know the answer and then combining it.

How do we use Divide And Conquer for Matrix muliplication First of we need to know whats the small problem: We know the Matrix multiplication of 2x2 Matrix. i.e. Suppose we have a 2x2 matrix:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$C_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$C_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$

$$C_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$C_{12} = a_{21} * b_{12} + a_{22} * b_{22}$$

Now we know the solution to small problem. Now we need to decide, How to divide a big problem.

$$\boldsymbol{A} = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right]$$

$$\boldsymbol{B} = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

If we consider the elements in this partitioned matrix as elements of new matrices:

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

We get a two by two matrix, for which we know the solution. We will divide the big matrix into small matrix by doing this operations. This only works with even order square matrix. But if we pad the matrix by 1 unit on both sides of odd order square matrix will give us even order square matrix.
Here's the python code for Matrix Multiplication with Divide and Conquer.

```
def MatMul(A, B, n):
    if n <= 2:
        return np.array([
            [A[0, 0] * B[0, 0] + A[0, 1]* B[1, 0],
             A[0, 0] * B[0, 1] + A[0, 1]* B[1, 1]],

            [A[1, 0] * B[0, 0] + A[1, 1]* B[1, 0],
             A[1, 0] * B[0, 1] + A[1, 1]* B[1, 1]]
        ])
    else:
        mid = n//2
        if n % 2 == 1:
            null = np.zeros((n + 1, n + 1))
            null[:n, :n] = A
            A = np.copy(null)
            null[:n, :n] = B
            B = np.copy(null)
            mid = (n + 1) // 2
        top_left = MatMul(A[:mid, :mid], B[:mid, :mid], mid) + MatMul(A[:mid, mid:], B[mi
        top_right = MatMul(A[:mid, :mid], B[:mid, mid:], mid) + MatMul(A[:mid, mid:], B[m

        bottom_left = MatMul(A[mid:, :mid], B[:mid, :mid], mid) + MatMul(A[mid:, mid:], B
        bottom_right = MatMul(A[mid:, :mid], B[:mid, mid:], mid) + MatMul(A[mid:, mid:],

        result = np.vstack(
            [np.hstack([top_left, top_right]),
            np.hstack([bottom_left, bottom_right])]
        )
        if result.shape[0] > n:
            return result[:n, :n]
        return result
```

The Recurrence Relation for this algorithm is :

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{8 \cdot T\left(\frac{n}{2}\right) + n^2} & \boldsymbol{n > 2} \\ \boldsymbol{1} & \boldsymbol{n <= 2} \end{cases} \tag{1.1}$$

So this algorithm of Complexity $\boldsymbol{O(n^3)}$. Since divide and Conquer Algorithm is recursive, it internally uses a stack which take extra space. So iterative algorithm is preferred (Both take $\boldsymbol{O(n^3)}$).
Matrix multiplication is troublesome process and it is active research topic and many of them had brought their own optimization for calculating MatMul. One of them is Strassen's Matrix Multiplication Method developed by Strassen.

2 Strassen's Matrix Multiplication

Major part of the matrix multiplication is multiplying. So if we can reduce the number of multiplication, we can make MatMul faster. Currently we have 8 multiplication operation. Strassen proposed some equation that has only 7 multiplication. Hence, reducing the time complexity a little less the $O(n^3)$. Here's the equations;

$$
\begin{aligned}
P &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
Q &= (A_{21} + A_{22}) \cdot B_{11} \\
R &= A_{11} \cdot (B_{12} - B_{22}) \\
S &= A_{22} \cdot (B_{21} - B_{11}) \\
T &= (A_{11} + A_{12}) \cdot B_{22} \\
U &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
V &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P + S - T + V \\
C_{12} &= R + T \\
C_{21} &= Q + S \\
C_{22} &= P + R - Q + U
\end{aligned}
$$

```
def MatMul(A, B, n):
    if n <= 2:
        return np.array([
            [A[0, 0] * B[0, 0] + A[0, 1]* B[1, 0],
             A[0, 0] * B[0, 1] + A[0, 1]* B[1, 1]],

            [A[1, 0] * B[0, 0] + A[1, 1]* B[1, 0],
             A[1, 0] * B[0, 1] + A[1, 1]* B[1, 1]]
        ])
    else:
        mid = n//2
        if n % 2 == 1:
            null = np.zeros((n + 1, n + 1))
            null[:n, :n] = A
            A = np.copy(null)
            null[:n, :n] = B
            B = np.copy(null)
            mid = (n + 1) // 2
        top_left = MatMul(A[:mid, :mid], B[:mid, :mid], mid) + MatMul(A[:mid, mid:], B[mi
        top_right = MatMul(A[:mid, :mid], B[:mid, mid:], mid) + MatMul(A[:mid, mid:], B[m

        bottom_left = MatMul(A[mid:, :mid], B[:mid, :mid], mid) + MatMul(A[mid:, mid:], B
        bottom_right = MatMul(A[mid:, :mid], B[:mid, mid:], mid) + MatMul(A[mid:, mid:],

        result = np.vstack(
            [np.hstack([top_left, top_right]),
            np.hstack([bottom_left, bottom_right])]
        )
        if result.shape[0] > n:
            return result[:n, :n]
        return result
```

Now this seems a lot right. No, this only takes 7 multiplication process and couple of additions rather than 8 multiplication process. Since addition is less time taking than multiplication. This should be better.
The Recurrence Relation is:

$$\boldsymbol{T(n) = \begin{cases} 7 \cdot T\left(\frac{n}{2}\right) + n^2 & n > 2 \\ 1 & n <= 2 \end{cases}} \tag{2.1}$$

So this algorithm of Complexity $\boldsymbol{O(n^{\log_2 7})}$. Since divide and Conquer Algorithm is recursive, it internally uses a stack which take extra space. So iterative algorithm is preferred (Both take $\boldsymbol{O(n^{2.81})}$).