Introduction to Tensor Methods for Computational Efficiency

James Albert Nguyen

November 2024

Abstract

Mathematicians originally invented tensors to be a generalization of a matrix in higher-dimensions to model multilinear relationships in theoretical physics, tensors are now structurally important in different fields of computer science. Advancements in computational efficiency in computer science are approached from two directions: one focuses on designing powerful hardware, while the other focuses on optimizing computing algorithms to minimize unnecessary computations. The increasing importance of complexity reduction in learning algorithms emphasizes current research endeavors in efficient tensor methods for computation and data representation. Developments in efficient tensor operations have been able to greatly reduce the computational complexity of basic computer functions to enhance the processing speeds of expensive model training. We also explore how techniques like decomposing tensors into lower-dimensional representations can be leveraged to further address computational and memory constraints. Current research hopes to combine these methods to optimize the performance of growing models in fields that rely heavily on high-dimensional operations.

1 Introduction

Lacking an organizational structure to describe multilinear relationships between several properties, mathematician Gregorio Ricci-Curbastro invented the tensor as a generalization of a matrix in higher dimensions. Originally, tensor calculus was most relevant in physics and engineering, famously integral to the development of Albert Einstein's theory of general relativity.

More recently, though, tensor calculus is the framework used to organize the impossibly complex relationships that define deep learning algorithms. The high-dimensional structure it provides continues to become more significant as we attempt to model increasingly complex relationships in data. Consequently, mathematicians are focused on improving the computational methods leveraged to process large amounts of data. The methods described in this paper detail how efficient methods for computing tensor operations are significant in reducing the computational complexity of large-scale operations. As discoveries in machine learning continue to unveil new capabilities, their computational complexity reveals the relevancy of methods such as efficient tensor multiplication and tensor decomposition. The work described in this exposition communicates the theoretical foundations that inspire efficient numerical methods.

2 Important Notation

2.1 A Brief Introduction to Tensors

Within the scope of this discussion, we refer to a tensor as an algebraic structure used to store values in higher dimensions. In lower dimensions, we know of similar structures such as scalars, vectors, and matrices to be 0th order, 1st order, and 2nd order tensors, respectively. We will borrow much of our notation from [Rabanser et al., 2017].

Definition 1 (Order-n Tensor)**.** Let $\mathcal{X}$ be a tensor of order-n be an n-dimensional tensor. We denote $\mathcal{X} \in \mathbb{R}^{I_1 \times,\ldots,\times I_n}$ where each I_i denotes the number of elements in the i^{th} dimension for $i \in \{1,\ldots,n\}$.

To maintain simplicity and relevancy, we will proceed to refer only to order-3 tensors. Such tensors have indexing properties called *sub-fields* that we refer to as *fibers* and *slices*.

Definition 2 (Fiber)**.** Fibers of a tensor are identified by fixing all but one index of the tensor. Let $i,j,k \in \mathbb{N}$ be integers such that $i \in \{1,\ldots,I_1\}$, $j \in \{1,\ldots,I_2\}$, and $k \in \{1,\ldots,I_3\}$ be the indices of an order-3 tensor $\mathcal{X}$. We denote the fibers of $\mathcal{X}$ as columns $x_{jk} = \mathcal{X}_{:jk}$, rows $x_{ik} = \mathcal{X}_{i:k}$, and tubes $x_{ij} = \mathcal{X}_{ij:}$.

Definition 3 (Slice)**.** Slices of a tensor are obtained by fixing all but two indices of the tensor. Let $i,j,k \in \mathbb{N}$ be integers such that $i \in \{1,\ldots,I_1\}$, $j \in \{1,\ldots,I_2\}$, and $k \in \{1,\ldots,I_3\}$ be the indices of an order-3 tensor $\mathcal{X}$. We denote the slices of $\mathcal{X}$ as frontals $X_k = \mathcal{X}_{::k}$, laterals $X_j = \mathcal{X}_{:j:}$, and horizontals $X_i = \mathcal{X}_{i::}$.

The operations that govern tensor-tensor interaction distinguish tensors from simply being higher-dimensional arrays of scalars. We define some foundational operations that will be relevant in later discussion.

Definition 4 (Tensor Outer Product)**.** Let $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_m}$ be tensors of order-n and order-m, respectively. Then denote $\mathcal{A} \otimes \mathcal{B}$ as the tensor outer product between $\mathcal{A}$ and $\mathcal{B}$. Let The resulting tensor $\mathcal{A} \otimes \mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times J_1 \times \cdots \times J_m}$ is an order-$(m+n)$ tensor with elements defined as

$$(\mathcal{A} \otimes \mathcal{B})_{i_1,\ldots,i_n,j_1,\ldots,j_n} = \mathcal{A}_{i_1,\ldots,i_n} \cdot \mathcal{B}_{j_1,\ldots,j_n}$$

for $i_k \in \{1,\ldots,I_k\}, j_l \in \{1,\ldots,J_l\}$ where $1 \leq k \leq n$ and $1 \leq l \leq m$.

From the definition above, one can construct from n vectors a tensor $\mathcal{X}$ of order-n. Let $x_1,\ldots,x_n$ be vectors. Then

$$\mathcal{X} = x_1 \otimes \cdots \otimes x_n.$$

3 Efficient Tensor Operations

One of the primary interests of computational efficiency is to reduce the number of total floating point operations required to come to a desired result. Such a motivation guides the field of numerical analysis. As a motivating example, we highlight the importance of Volker Strassen's 1969 discovery of a more efficient algorithm for computing the product of two square matrices of order-$m2^n$ for $m,n \in \mathbb{N}$ [Strassen, 1969]. In the following section, we evaluate the cost of two matrix multiplication algorithms with respect to time complexity. We describe time complexity with respect to Big-O notation.

Definition 5 (Big-O Notation)**.** Let f be a function. Then we measure its computational complexity by counting the number of floating point operations (FLOPS) required to compute the solution $f(n)$, where n is a scalar by which the complexity of f increases. Then we say that $f(n) = O(g(n))$ for some g if for some $M \in \mathbb{R} \implies f(n) \leq M|g(n)|, \forall n \in \mathbb{N}$.

3.1 Strassen's Algorithm

Strassen's algorithm for efficient matrix multiplication relies on sub-matrix multiplication to reduce the problem into several lower-dimensional problems.

Definition 6 (Sub-Matrix)**.** Let A, be a square matrix of order-$2n$, where $n \in \mathbb{N}$. Let each quadrant A_{ij} of A to be a square sub-matrix of A of order-n such that

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

Lemma 6.1 (Sub-matrix Multiplication)**.** Let A, B be square matrices of order-$2n$. Then the matrix product AB can be computed by sub-matrix multiplication as follows:

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

Beyond this simplification, Strassen realized that several of the matrix-multiplications used to compute 2×2 matrix multiplication were repeated in the procedure. He leveraged this realization to reduce the computational complexity by reusing these products between intermediate steps. Ultimately, his significant discovery can be described as performing 2×2 matrix multiplication in only 7 multiplications, a reduction from 8 multiplications required in the traditional algorithm.

Theorem 6.1 (Strassen's Algorithm)**.** *Let A, B be square matrices of order-$m2^{k+1}$ for some $m, k \in \mathbb{R}$ such that*

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

Denote $\alpha_{m,k}$ be Strassen's Algorithm for square multiplication between matrices of order-$m2^{k+1}$. Then let

$$\begin{aligned} I &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ II &= (A_{21} + A_{22})B_{11}, \\ III &= A_{11}(B_{12} - B_{22}), \\ IV &= A_{22}(-B_{11} + B_{21}), \\ V &= (A_{11} + A_{12})B_{22}, \\ VI &= (-A_{11} + A_{21})(B_{11} + B_{12}), \\ VII &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

such that the components can be used to compute

$$\begin{aligned} C_{11} &= I + IV - V + VII, \\ C_{21} &= II + IV, \\ C_{12} &= III + IV, \\ C_{22} &= I + III - II + VI \end{aligned}$$

where matrix multiplication uses $\alpha_{m,k}$ and matrix addition is performed element-wise as usual.

Proof. Consider the equivalent computations:

$$
\begin{aligned}
C_{11} &= I + IV - V + VII \\
&= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{22}(-B_{11} + B_{21}) - (A_{11} + A_{12})B_{22} + (A_{12} - A_{22})(B_{21} + B_{22}) \\
&= A_{11}B_{11} + A_{12}B_{21} \\
C_{21} &= II + IV \\
&= (A_{21} + A_{22})B_{11} + A_{22}(-B_{11} + B_{21}) \\
&= A_{21}B_{11} + A_{22}B_{21} \\
C_{12} &= III + IV \\
&= A_{11}(B_{12} - B_{22}) + A_{22}(-B_{11} + B_{21}) \\
&= A_{11}B_{12} + A_{12}B_{22} \\
C_{22} &= I + III - II + VI \\
&= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{11}(B_{12} - B_{22}) - (A_{21} + A_{22})B_{11} + (-A_{11} + A_{21})(B_{11} + B_{12}) \\
&= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

■

Although less straightforward than the traditional method for performing matrix multiplication, the following proof shows that Strassen's algorithm produces the result of $m2^k + 1$ matrix multiplication in less total floating point operations.

To solve for the time complexity of Strassen's algorithm, we identify it as a Divide and Conquer Recursive algorithm, meaning it is an algorithm that divides the problem into subproblems and solves them recursively. This allows us to use the Master Theorem to solve for time complexity.

Theorem 6.2 (Master Theorem)**.** *The time complexity of a Divide and Conquer algorithm can modeled by*

$$T(n) = aT(\frac{n}{b}) + O(f(n))$$

where a denotes the number of subproblems, b denotes the factor by which the problem is reduced, and $f(n)$ denotes the cost of the combining subproblems' solutions. Then the total time complexityd $T(n)$ follows the following guidelines:

$$
T(n) \simeq \begin{cases} O(n^{\log_b a}) & f(n) \text{ is polynomially smaller than } n^{\log_b a} \\ O(f(n)) \log n & f(n) \text{ is polynomially equal to } n^{\log_b a} \\ f(n) & f(n) \text{ is polynomially greater than } n^{\log_b a} \end{cases}
$$

Proof. We consider a recursive tree T, where the root node T_0 is the initial problem from which we obtain sub-problems. Then each layer T_i of the tree has a time complexity equal to $T_i(n) = a_i f(n/b^i)$, where a is the number of sub-problems per iteration, and b is the multiplicative factor by which the problem decreases. Then there are $\log_n(b)$ layers of the tree and the total time complexity of T can be described as the sum of the complexity at each layer T_i:

$$T(n) = [\sum_{i=0}^{log_n(b)} a^i f(\frac{n}{b^i})] + O(n^{log_b(a)}),$$

where the term $O(n^{\log_b(a)})$ is equal to the sum of the complexity cost across the leaves of T. From this statement, we let $\varepsilon > 0 \in \mathbb{R}$ and consider the following cases:

1. $f(n) = O(n^{log_b(a)-\varepsilon}) < O(n^{\log_b(a)})$,

2. $f(n) = O(n^{\log_b(a)})$,
3. $f(n) = O(n^{\log_b(a)+\varepsilon}) > O(n^{\log_b(a)})$.

Case (1):

$$T(n) = [\sum_{i=0}^{\log_n(b)} a^i f(\frac{n}{b^i})] + O(n^{\log_b(a)}) \leq [\sum_{i=0}^{\log_n(b)} a^i (\frac{n}{b^i})^{\log_b a-\varepsilon}] + O(n^{\log_b(a)})$$

$$\text{where } \sum_{i=0}^{\log_n(b)} a^i (\frac{n}{b^i})^{\log_b a-\varepsilon} = n^{\log_b a-\varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\epsilon} = n^{\log_b a-\varepsilon} \sum_{i=0}^{\log_b n} b^{i\varepsilon}$$

$$= n^{\log_b a-\varepsilon} \frac{b^{\varepsilon(\log_b n+1)} - 1}{b^\varepsilon - 1} = n^{\log_b a-\varepsilon} \frac{b^\varepsilon n^\varepsilon - 1}{b^\varepsilon - 1}$$

$$\leq n^{\log_b a-\varepsilon} \frac{b^\varepsilon n^\varepsilon - 1}{b^\varepsilon} = n^{\log_b a} \frac{b^\varepsilon n^\varepsilon - 1}{b^\varepsilon}$$

$$\leq M|n^{\log_b a}|, \text{ where } M \in \mathbb{R} \text{ is some constant.}$$

Then $f(n) = O(n^{\log_b(a)-\varepsilon}) < O(n^{\log_b(a))} \implies f(n) = O(n^{\log_b a})$. Since Case (1) is most relevant to the following claim, we leave Cases (2) and (3) as an exercise to the reader. For more details see [University, 2012] ■

This technique emphasizes the significance of reducing the number of multiplications required for 2×2 matrix multiplication. When we reimagine each of these multiplications as a subproblem of each recursive step of Strassen's algorithm, the Master Theorem reveals a reduction in overall time complexity, especially in high-dimensional matrix operations.

Claim 7 (Strassen's Algorithm: Computational Complexity)**.** Let $n = m2^{k+1}$ for some $m, k \in \mathbb{R}$ and let Strassen's Algorithm be used to compute AB for A, B are square matrices of order-n. Then Strassen's algorithm for order-n matrix multiplication has a complexity rate of $O(n^{\log_2 7})$.

Proof. Let Strassen's algorithm for matrix multiplication be denoted by $\alpha_{m,k}$, such that $\alpha_{m,k}$ is a Divide and Conquer Recursive algorithm with $a = 7$ subproblems, each reducing the problem by a factor of $b = 2$. Combining each subproblem involves only matrix multiplication, which grows in complexity proportionally to $f(n) = n^2$ since there are n^2 elements in each matrix. Thus, we model the time complexity $T(n)$ of $\alpha_{m,k}$ with the Master Theorem:

$$T(n) = 7T(n/2) + O(n^2)$$
$$\implies n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} > n^2$$

Then by the conditions of the Master Theorem, $T(n) \simeq O(n^{\log_2 7}) \approx O(n^{2.81})$.

■

These results are particularly interesting because of the property of sub-matrix multiplication that allows Strassen's algorithm to scale to higher dimensions. Considering this "slight" improvement, matrices of much higher dimension, which are commonly found in applications in physical and computer sciences, see drastic improvement in computational efficiency.

3.2 Lower Bound Complexity

Strassen's groundbreaking discovery in 1969 initiated a search for increasingly efficient algorithms. However, it was soon proven by Shmuel Winograd that order-2 square matrix multiplication requires at least 7 multiplications [Winograd, 1971]. While this discovery did nullify the possibility of performing a order-2 matrix

multiplication in 6 multiplications, it raised the question of whether or not the lower bound for matrices of order-2^n would also be 7^n multiplications, or equivalently, a time complexity of $O(n^{\log_2 7})$.

A string of computationally superior algorithms for matrices of different sizes soon followed Strassen's discovery; however, none of them would surpass Strassen's efficiency until 2022, when a team of researchers at Google Deepmind would leverage reinforcement learning to take advantage of an important tensor property.

4 Google Deepmind: AlphaTensor

A collaborative project named AlphaTensor made a significant breakthrough in 2022, using reinforcement learning to search for efficient matrix algorithms [Fawzi et al., 2022]. To describe Deepmind's procedure, we define other important tensor properties.

4.1 Tensor Properties

To motivate the following section, we first recall properties of matrix rank decomposition. Let the rank r of matrix $M \in \mathbb{R}^{m\times n}$, where r is the number of linearly independent rows or columns of M. Then M can be decomposed into the product of two matrices

$$M = AB^T, \text{ with } A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}.$$

Similarly, we expand upon this concept to tensor rank decomposition after some foundational definitions. We, again, borrow notation as described in [Rabanser et al., 2017].

Definition 8 (Rank-1 Tensor)**.** An order-n tensor can be described as rank-1 if it can be composed of the outer product of n vectors. Explicitly, a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots I_n}$ is of rank-1 if and only if $\exists\, v_1, \ldots, v_n$ such that $\mathcal{X} = v_1 \otimes \cdots \otimes v_n$

Definition 9 (Tensor Rank)**.** We say that the rank of a tensor $rank(\mathcal{X}) = R$ if a minimum of R rank-1 tensors are needed to sum to $\mathcal{X}$. That is,

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r v_r^{(1)} \otimes \cdots \otimes v_r^{(n)}, \text{where } \lambda_r \in \mathbb{R} \text{ are scalars and } v_r^{(1)} \cdots v_r^{(n)} \text{ are vectors.}$$

Relevantly, we note that since matrix multiplication $(A, B) \mapsto AB$ is linear in both arguments, we say that it is a bilinear function of A and B. This provides a very important basis for the following developments, which primarily depend on the Universal Property of Tensor Products. Note that the following definition refers to R-modules; however, we will simply refer to them as vector spaces for relevancy, as they are special cases of R-modules.

Definition 10 (Tensor Products)**.** Let U, V be vector spaces. Then the tensor product of U and V is itself a vector space denoted $U \otimes V$ with an associated bilinear map $\pi : U \times V \mapsto U \otimes V$ with $\dim(U \otimes V) = \dim(U) \cdot \dim(V)$. Furthermore, $u \otimes v \in U \otimes V$ are tensors where $u \in U, v \in V$.

Lemma 10.1. Let U and V be vector spaces. Then the tensor product and its bilinear map $\pi : U\times V \mapsto U\otimes V$ exist always.

Proof. Since U and V are vector spaces, they are R-modules. Let P be the R-module generate by the elements of $U \times V$. Suppose $u_1, \ldots, u_k \in U, v_1, \ldots, v_k \in V$. Then we can generate any element of P by a linear combination of the elements of U and V,

$$r_1(u_1, v_1) + \cdots + r_k(u_k, v_k), with r_1, \ldots, r_k \in R.$$

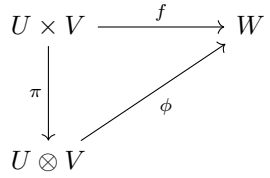

Figure 1: Commutative Diagram for Universal Property of Tensor Products

We want to define a submodule Q of P such that we define its quotient $T = P/Q$ as the tensor product of U and V. To achieve this, we give generators of G. From $u_1, u_2, u \in U, v_1, v_2, v \in V, r \in R$, we have

$$(u_1 + u_2, v) - (u_1, v) - (u_2, v) \tag{1}$$
$$r(u, v) - (ru, v) \tag{2}$$
$$(u, v_1 + v_2) - (u, v_1) - (u, v_2) \tag{3}$$
$$r(u, v) - (u, rv) \tag{4}$$

are generators of Q. Let $\pi : U \times V \mapsto U \otimes V$ be a map. Then to fulfill our definition, we want to verify that π is bilinear and universal.

To check that π is bilinear, we verify the additive and multiplicative properties of π. Having designed the generators of Q to satisfy these properties, it is easy to show that 1 and 3 encode the additive property in the first and second components, respectively, and 2 and 4 encode the multiplicative property in the first and second components, respectively. Thus, π is a bilinear map.

■

Theorem 10.1 (Universal Property of Tensor Products)**.** *Let U and V be vector spaces and U^* and V^* be the sets of linear functions (dual spaces) on U and V. Let $f : U \times V \mapsto W$ be any bilinear map. Define the tensor product of U and V to be a vector space denoted $U \otimes V$. Then:*

1. *There exists a unique bilinear map $\pi : U \times V \mapsto U \otimes V$,*
2. *For any bilinear map $f : U \times V \mapsto W$, there exists a unique homomorphism $\phi : U \otimes V \mapsto W$*
3. *The relationship satisfies $f(u, v) = \phi \circ \pi(u, v)$*

The relationship can be described by the following commutative diagram:

Proof. We continue with the notation from the proof of 4.1. To check that π also satisfies the universal property, we consider a bilinear map

$$f : U \times V \mapsto W,$$

where W is some R-module. The universal property of free-modules describes that the existence of f implies the existence of some R-module

$$\phi : P \mapsto W$$

that extends f. However, since f is bilinear, it is true that the generators of Q are in the kernel of ϕ. Recall that $T = P/Q$ is the quotient of P over Q; then the universal property of a quotient suggests that there exists a unique homomorphism $\psi : T \mapsto W$.

■

Recalling that matrix multiplication is a bilinear map, we establish that any matrix multiplication algorithm can be represented by a unique order-3 tensor $\mathcal{X} \in U \otimes V$, where this vector space is the tensor product of the vector spaces that contain the matrices of interest. We can update our commutative diagram:

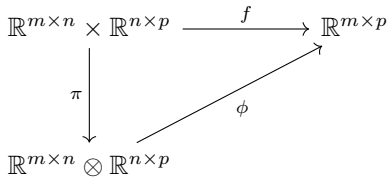

Figure 2: Bilinear Map from Matrix Multiplication Algorithm to Unique Tensor Rank Decomposition

Explicitly, let $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p} \implies AB \in \mathbb{R}^{m\times p}$ and let f be a bilinear map for matrix multiplication AB. Then there exists a tensor $\mathcal{T}_{m,n,p} \in \mathbb{R}^{m\times n} \otimes \mathbb{R}^{n\times p}$ such that

$$\phi \circ \pi(A, B) = \phi(\mathcal{T}_{m,n,p}) = f : \mathbb{R}^{m\times n} \times \mathbb{R}^{n\times p} \mapsto \mathbb{R}^{m\times p}.$$

If we denote $\mathcal{T}_n$ to be the tensor that is representative of order-n square matrix multiplication, we can describe its algorithm as its rank-decomposition into the sum of R many rank-1 tensor products. We will denote

$$\mathcal{T}_n = \sum_{r=1}^{R} a^{(r)} \otimes b^{(r)} \otimes c^{(r)}$$

is a tensor decomposition for an order-n matrix multiplication algorithm.

In 2022, the researchers at Google Deepmind released a breakthrough, describing a process used to search for such algorithms using deep reinforcement learning techniques. Although beyond the scope of this analysis, we discuss the mathematics that motivate the AlphaTensor project that led to the discovery of several new algorithms. Simply stated, the game subtracts the outer tensor product of a generated triplet $(a^{(t)}, b^{(t)}, c^{(t)})$ from an iteration tensor $\mathcal{S}_t$ with the goal of reaching the 0-tensor in as few iterations as possible. Once the iteration reaches the 0-vector, the sequence of triplets satisfies $\mathcal{T}_n = \sum_{t=1}^{R} a^{(t)} \otimes b^{(t)} \otimes c^{(t)}$, where R is equal to the number of iterations.

We motivate this idea by visualizing Volker Strassen's algorithm for order-2 square matrix multiplication as $\mathcal{X}_2$. Recall the notation from 3.1 and consider the following tensors:

$$\begin{aligned}
I &= (a_1 + a_4)(b_1 + b_4) \\
II &= (a_3 + a_4)b_1 \\
III &= a_1(b_2 - b_4) \\
IV &= a_4(b_3 - b_1) \\
V &= (a_1 + a_2)b_4 \\
VI &= (a_3 - a_1)(b_1 + b_2) \\
VII &= (a_2 - a_4)(b_3 + b_4) \\
C_{11} &= I + IV - V + VII \\
C_{12} &= I + III \\
C_{21} &= II + IV \\
C_{22} &= I - II + III + VI
\end{aligned}$$

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix},$$

$$B = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Figure 3: Tensor Factor Representation of Strassen's Order-2 Matrix Multiplication Algorithm

In 3, we see arrays A, B, C for which the columns of A and B are vectors (rank-1 tensors) that convey the first and second factors, respectively, of each of the 7 products (sub-problems) in Strassen's algorithm. Meanwhile, the rows of C convey the 4 sums of products. We denote the r^{th} column of A, B, C to be $a^{(r)}, b^{(r)}, c^{(r)}$ for $r \in \{1, \ldots, R\}$, where R is the rank of the tensor with form

$$\mathcal{T}_2 = \sum_{r=1}^{7} a^{(r)} \otimes b^{(r)} \otimes c^{(r)}.$$

It becomes immediately apparent that the rank of the tensor is equivalent to the number of total multiplications, and consequently the time complexity, required to compute the order-2 matrix product. In fact, it is true for all square matrices of order-n that an algorithm that can be represented by a tensor decomposition $\mathcal{T}_n$ of $rank(\mathcal{T}_n) = R$ can compute its associated matrix product with asymptotic complexity $O(n^{\log_n R})$. Reasoning for this claim can be found at [Landsberg, 2017]

Revisiting Winograd's lower bound the number of multiplications required to compute the order-2 square matrix product, it seems to hold that there is no way to achieve a complexity of 6 multiplications. However, a shocking result from AlphaTensor reveals a new domain for advancement in the field of efficient matrix multiplication. Alongside several other new algorithms, a tensor representation for an order-4 matrix multiplication algorithm was discovered to be of $rank(\mathcal{T}_4) = 47$. This algorithm, not only outperforms the previous method (which scaled Strassen's algorithm using submatrix multiplication), but it boasts an asymptotic complexity of $O(n^{\log_4(47)}) \approx O(n^{2.78}) < O(n^{\log_2(7)}) = O(n^{2.81})$. Therefore, while Winograd's proof holds, as expected, we celebrate a new domain for higher efficiency between the range of $O(n^{\log_2(6)})$ and $O(n^{\log_2(7)})$.

References

[Fawzi et al., 2022] Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, F. J. R., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., and Kohli, P. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

[Landsberg, 2017] Landsberg, J. M. (2017). *Geometry and Complexity Theory*, volume 169 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.

[Rabanser et al., 2017] Rabanser, S., Shchur, O., and Günnemann, S. (2017). Introduction to tensor decompositions and their applications in machine learning.

[Strassen, 1969] Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356.

[University, 2012] University, C. (2012). Master method proof. Accessed: 2024-11-19.

[Winograd, 1971] Winograd, S. (1971). On multiplication of 2 × 2 matrices. *Linear Algebra and its Applications*, 4(4):381–388.