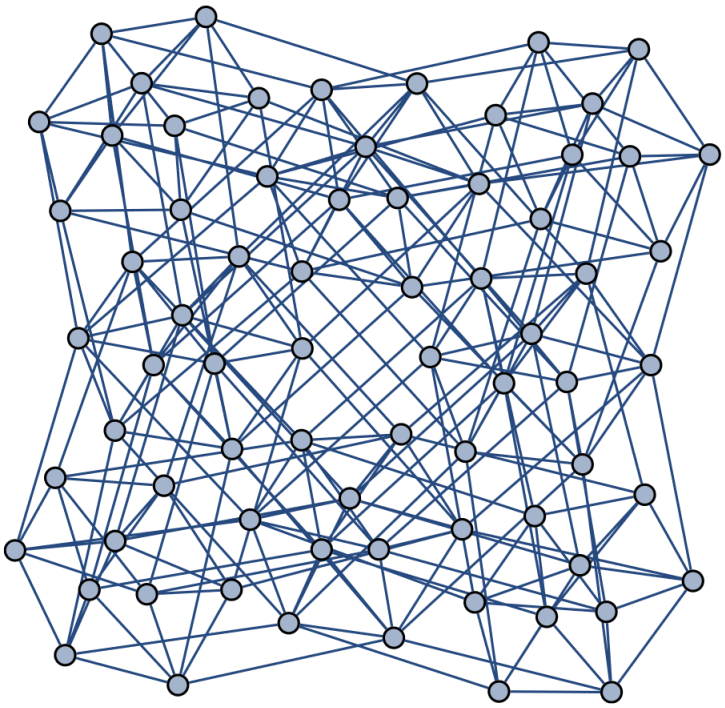
Universitat Autònoma de Barcelona

Facultad de Ciencias

Grado en Matemáticas

Trabajo final de grado

La multiplicación de matrices, del número mínimo de productos a la inteligencia artificial



Rocío Rodríguez López

Tutor: Roberto Rubio Núñez

2023-2024

Índice

Introducción

La multiplicación de matrices es una operación muy utilizada en el ámbito matemático y computacional. Se utiliza en una amplia gama de aplicaciones, desde la simulación de sistemas físicos hasta el procesamiento de imágenes, donde es utilizada para aplicar filtros y transformaciones que mejoran la calidad y la comprensión de los datos. Además, son la base de muchos algoritmos de entrenamiento y predicción en el ámbito de la inteligencia artificial y el aprendizaje automático.

La eficiencia en la multiplicación de matrices es importante, más aún cuando se consideran matrices grandes y operaciones repetitivas. Por lo tanto, conocer el número mínimo de operaciones necesarias para llevar a cabo esta tarea es fundamental para optimizar el algoritmo y mejorar el rendimiento de sistemas computacionales. En este contexto nos preguntamos, ¿cuál es el número mínimo de productos necesarios para realizar una multiplicación de matrices?

Este trabajo lo podemos dividir en tres partes. La primera parte, de carácter teórico, aborda la pregunta anterior en el caso de matrices 2×2. Aunque el algoritmo de Strassen se introdujo en 1969, demostrar que siete son el mínimo número de operaciones necesarias requiere un desarrollo teórico que realizó Winograd en 1970 y que presentamos en el primer capítulo de este trabajo. Además, analizamos cuando es más útil este algoritmo y las mejoras que ofrece en comparación con el algoritmo estándar de multiplicación de matrices.

En el estudio de matrices de distintos tamaños, resulta muy útil la notación de los llamados tensores, que son generalizaciones de matrices. En la segunda parte vemos cómo se han aplicado métodos de inteligencia artificial a este formalismo tensorial para descubrir nuevos algoritmos, que reducen el número de multiplicaciones de los algoritmos conocidos hasta los tamaños de matrices 5×5.

Por último, veremos como, pese a la potencia de la inteligencia artificial, métodos computacionales clásicos consiguieron mejorar algunos de los algoritmos descubiertos mediante inteligencia artificial. En cuestión de ocho días lograron reducir el número de multiplicaciones para el tamaño 5×5 y, en apenas dos meses, mejorar o igualar el número de multiplicaciones para el resto de tamaños.

1. Número mínimo de productos necesarios

El objetivo de este capítulo es demostrar que cualquier algoritmo que calcule la multiplicación entre dos matrices 2×2 requiere, como mínimo, siete multiplicaciones. Para poder demostrarlo, comenzaremos por definir el concepto de algoritmo.

Definición 1. *Sea un cuerpo $\mathcal{F}$. Un algoritmo de N pasos es una función*

$$\alpha : \{1, 2, ..., N\} \to \mathcal{F} \cup \{x_1, x_2, ..., x_n\} \cup (\{1, 2, ..., N\}^2 \times \{p_1, p_2, p_3, p_4\})$$

donde si $\alpha(i) = (j_1, j_2, p_k)$ entonces $j_1, j_2 < i$. Además, definimos las siguientes funciones, para $k \in \{1, 2, 3, 4\}$ y $\mathcal{F}(x_1, ..., x_n)$ el cuerpo de funciones,

$$p_k : \mathcal{F}(x_1, ..., x_n) \times \mathcal{F}(x_1, ..., x_n) \to \mathcal{F}(x_1, ..., x_n)$$

que satisfacen $p_1(u, v) = u + v$, $p_2(u, v) = u - v$, $p_3(u, v) = u \cdot v$ y $p_4(u, v) = u/v$ para $v \neq 0$. La evaluación de un algoritmo es una función $e_\alpha : \{1, 2, ..., N\} \to \mathcal{F}(x_1, ..., x_n)$ definida por [1]:

a) $e_\alpha(i) = \alpha(i)$ si $\alpha(i) \in \mathcal{F} \cup \{x_1, x_2, ..., x_n\}$.

b) $e_\alpha(i) = p_k(e_\alpha(j_1), e_\alpha(j_2))$ si $\alpha(i) = (j_1, j_2, p_k)$ y en el caso que $k = 4$, $e_\alpha(j_2) \neq 0$.

Restringiéndonos al caso de matrices 2×2 [5], un algoritmo puede calcular el producto $A \cdot B$, siendo A, B dos matrices con entradas $(a_{ij}), (b_{ij})$ respectivamente, si existen j_1, j_2, j_3, j_4 tal que

$$e_\alpha(j_1) = a_{11} \cdot b_{11} + a_{12} \cdot b_{21},$$

$$e_\alpha(j_2) = a_{11} \cdot b_{12} + a_{12} \cdot b_{22},$$

$$e_\alpha(j_3) = a_{21} \cdot b_{11} + a_{22} \cdot b_{21},$$

$$e_\alpha(j_4) = a_{21} \cdot b_{12} + a_{22} \cdot b_{22}.$$

Además, si tenemos que $e_\alpha(i) \in \mathbb{Q} \cup \{a_{11}, ..., a_{22}, b_{11}, ..., b_{22}\}$, lo llamamos *data step*, en cambio, si se obtiene de sumar, restar, multiplicar o dividir $e_\alpha(i_1)$ y $e_\alpha(i_2)$, para $i_1, i_2 < i$, se llama *computation step*, y los elementos $e_\alpha(i_1)$ y $e_\alpha(i_2)$ los denominamos argumentos.

1.1. Producto de matriz por vector

El siguiente teorema determina el mínimo número de multiplicaciones y divisiones necesarias para calcular ciertas funciones $\psi = \phi\mathbf{x} + \varphi$, para un cuerpo $\mathcal{F}$, donde $\phi \in \mathcal{M}_{t\times n}(\mathcal{F})$, $\mathbf{x}$ denota

un vector de indeterminadas $(x_1, ..., x_n)$ y $\varphi = (\varphi_1, ..., \varphi_t) \in \mathcal{F}^t$.

Trabajaremos bajo la hipótesis de que $\mathcal{F}$ contiene un subcuerpo $\mathcal{G}$ con infinitos elementos.

Teorema 2. *[3] Sea α un algoritmo que calcula ψ. Denotamos por $\phi_1, ..., \phi_n$ las columnas de ϕ.*
Si existen λ vectores de $\{\phi_1, ..., \phi_n\}$ que no satisfacen una combinación lineal en $\mathcal{G}^t$, entonces α tiene al menos λ multiplicaciones o divisiones.
Además, α tiene como mínimo λ pasos k tal que cada $\alpha(k)$ es de la forma (j_1, j_2, p_3) con $e_\alpha(j_1) \notin \mathcal{G}$ y $e_\alpha(j_2) \notin \mathcal{G}$ o bien es de la forma (j_1, j_2, p_4) para $e_\alpha(j_2) \notin \mathcal{G}$ (es decir, las multiplicaciones y divisiones por un elemento de $\mathcal{G}$ no se contabilizan).

Demostración. Para esta demostración asumimos que α es el algoritmo que calcula $\phi\mathbf{x}+\varphi$ con el mínimo número de multiplicaciones y divisiones. Observamos que, si los primeros q pasos de α no usan multiplicaciones ni divisiones que no estén en $\mathcal{G}$, entonces para toda $i = 1, 2, ..., q$ satisface

$$e_\alpha(i) = \sum_{j=1}^{n} g_{ij}x_j + f_j,$$

para $g_{ij} \in \mathcal{G}$ y $f_i \in \mathcal{F}$.
Demostraremos el teorema por inducción sobre λ. Si $\lambda = 1$ hay un vector de $\{\phi_1, ..., \phi_n\}$ que no cumple ninguna combinación lineal en $\mathcal{G}^t$, por tanto, existen i_0, j_0 de manera que $\phi_{i_0 j_0} \notin \mathcal{G}$. Si suponemos que no hay ninguna multiplicación/división que esté en $\mathcal{G}$ entonces por la existencia del algoritmo α y para alguna k se cumple:

$$e_\alpha(k) = \sum_{j=1}^{n} \phi_{i_0 j}x_j + \varphi_{i_0}.$$

Pero como α no tiene multiplicaciones ni divisiones que se contabilicen, entonces existen $g_j \in \mathcal{G}$ para $j = 1, ..., n$ y $f \in \mathcal{F}$, de manera que $e_\alpha(k) = \sum_{j=1}^{n} g_j x_j + f$. Por consiguiente, $g_{j_0} = \phi_{i_0 j_0}$ contradiciendo el hecho que $\phi_{i_0 j_0} \notin \mathcal{G}$, por tanto, si $\lambda = 1$, α tiene al menos una multiplicación o división.
Suponemos que el enunciado es cierto para $\lambda = N$ y vemos si se cumple para $\lambda = N + 1$. Suponemos que hay $N + 1$ vectores en $\{\phi_1, ..., \phi_n\}$ que no cumplen una combinación lineal en $\mathcal{G}^t$. Sea k el entero más pequeño donde las multiplicaciones y divisiones que se contabilizan ocurran en el paso k, entonces se cumple

$$e_\alpha(k) = (\sum_{i=1}^{n} g_i x_i + f) \cdot (\sum_{i=1}^{n} h_i x_i + f') \quad \text{o} \quad e_\alpha(k) = (\sum_{i=1}^{n} g_i x_i + f) : (\sum_{i=1}^{n} h_i x_i + f'),$$

para $g_i, h_i \in \mathcal{G}$ y $f, f' \in \mathcal{F}$. Además, uno de los h_i o uno de los g_i no es cero, de lo contrario $e_\alpha(k) \in \mathcal{F}$ y no se contabilizaría ninguna multiplicación. Suponemos que uno de los h_i es distinto de 0, sin perdida de generalidad, suponemos que $h_n \neq 0$. Como las multiplicaciones por h_n o h_n^{-1} no se contabilizan, consideramos $h_n = 1$.
Sea $g \in \mathcal{G}$ un elemento el cual satisface que, si sustituimos x_n por $g - f' - \sum_{i=1}^{n-1} h_i x_i$, el algoritmo resultante α' cumple que $e_{\alpha'}$ es una función total. Sabemos que g existe porque hay un número finito de sustituciones de x_n para que $e_{\alpha'}$ no sea total y $\mathcal{G}$ tiene infinitos elementos. Sustituyendo $g - f' - \sum_{i=1}^{n-1} h_i x_i$ por x_n obtenemos un algoritmo α' que calcula $\phi' x' + \varphi'$, donde $\phi' = \phi_j - h_i \phi_n$ para $j = 1, ..., n-1$, $\varphi' = \varphi + (g - f')\phi_n$ y $x' = (x_1, ..., x_n)$.
El número de multiplicaciones y divisiones que se consideran en α' es, como mínimo, una menos que en α, a causa de que en el paso k no hay ninguna multiplicación o división que se contabilice en α'; debido a que el algoritmo β, que calcula $g - f' - \sum_{i=1}^{n-1} h_i x_i$, no tiene multiplicaciones o divisiones que se cuente. No obstante, en el algoritmo α', existen al menos N vectores en $\{\phi_1, ..., \phi_n\}$ que no cumplen ninguna combinación lineal en $\mathcal{G}^t$ y por hipótesis de inducción, α' tiene al menos N multiplicaciones y divisiones. Por lo tanto, α tiene al menos $N + 1$ multiplicaciones y divisiones que son consideradas.

□

Corolario 3. *El mínimo número de multiplicaciones o divisiones que se requieren en un algoritmo para calcular $A \cdot b$, donde $A = (a_{ij})$ es una matriz $m \times n$ y $b = (b_j)$ es un n-vector, es $m \cdot n$.*

Demostración. Consideramos ϕ la matriz $m \times mn$, cuyos coeficientes son de la siguiente forma:

$$\phi_{ij} = \begin{cases} b_k & \text{si } j = (i-1)n + k,\ 1 \leq k \leq n \\ 0 & \text{de lo contrario} \end{cases}$$

Sea $\mathbf{a}$ el vector columna $(a_{11}, ..., a_{1n}, a_{21}, ..., a_{2n}, ..., a_{mn})$. Entonces se cumple que $Ab = \phi\mathbf{a}$. Como no hay ninguna combinación lineal entre las $m \cdot n$ columnas de ϕ, aplicando el Teorema 1, obtenemos que el algoritmo para multiplicar Ab tiene como mínimo $m \cdot n$ multiplicaciones o divisiones.

□

1.2. Lemas preliminares

Ahora nos centraremos en las matrices 2×2 y el cuerpo de los racionales, para encontrar el número mínimo de multiplicaciones necesarias para multiplicar dos matrices.

Lema 4. *Sea α un algoritmo para multiplicar dos matrices 2×2, y sean $p_1, p_2, ..., p_m$ los resultados de los computation steps obtenidos de realizar multiplicaciones. Por lo tanto, para cada paso k existen números racionales r_i, r_{ij} y r'_{ij}, de manera que*

$$e_\alpha(k) = r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i=1}^{2}\sum_{j=1}^{2} r_{ij} a_{ij} + \sum_{i=1}^{2}\sum_{j=1}^{2} r'_{ij} b_{ij}. \tag{1}$$

Demostración. Haremos una demostración por inducción sobre el número de pasos.
Para $k = 1$, por definición $e_\alpha(1) \in \mathbb{Q} \cup \{a_{11}, ..., a_{22}, b_{11}, ..., b_{22}\}$, por lo tanto, se puede escribir de la forma deseada. Como se cumple para el primer paso, suponemos que la igualdad se cumple para los pasos menores que k y demostramos que también se cumple para el paso k. Entonces, o bien $e_\alpha(k) \in \mathbb{Q} \cup \{a_{11}, ..., a_{22}, b_{11}, ..., b_{22}\}$ y se podría expresar como la ecuación 1, o bien $e_\alpha(k) = e_\alpha(j_1) \pm e_\alpha(j_2)$ para $j_1, j_2 < k$. En este caso, como hemos supuesto que la igualdad se cumplía para todos los pasos menores que k, tenemos que existen r_i, r_{ij}, r'_{ij} y s_i, s_{ij}, s'_{ij} números racionales que satisfacen

$$e_\alpha(j_1) = r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i=1}^{2}\sum_{j=1}^{2} r_{ij} a_{ij} + \sum_{i=1}^{2}\sum_{j=1}^{2} r'_{ij} b_{ij},$$
$$e_\alpha(j_2) = s_0 + \sum_{i=1}^{m} s_i p_i + \sum_{i=1}^{2}\sum_{j=1}^{2} s_{ij} a_{ij} + \sum_{i=1}^{2}\sum_{j=1}^{2} s'_{ij} b_{ij}.$$

En consecuencia,

$$e_\alpha(j) = (r_0 + s_0) + \sum_{i=1}^{m} (r_i + s_i) p_i + \sum_{i=1}^{2}\sum_{j=1}^{2} (r_{ij} + s_{ij}) a_{ij} + \sum_{i=1}^{2}\sum_{j=1}^{2} (r'_{ij} + s'_{ij}) b_{ij}$$

y obtenemos lo deseado.

□

Lema 5. *Sea α un algoritmo que calcula el producto de matrices 2×2 con m pasos de multiplicación. Entonces existe un algoritmo β que requiere m pasos, de manera que los argumentos de cada multiplicación son de la forma*

$$\sum_{j=1}^{2}\sum_{i=1}^{2} r_{ij} a_{ij} + \sum_{j=1}^{2}\sum_{i=1}^{2} r'_{ij} b_{ij}.$$

Demostración. Sean $u, v \in \mathbb{Q}[a_{11}, ..., b_{22}]$ dos polinomios. Definimos las funciones

$$L_i : \mathbb{Q}[a_{11}, ..., b_{22}] \to \mathbb{Q}[a_{11}, ..., b_{22}]$$

para $i = 0, 1, 2, 3$; entonces si $p \in \mathbb{Q}[a_{11}, ..., b_{22}]$ es un polinomio, $L_0(p)$ es el término independiente, $L_1(p)$ es la parte lineal, $L_2(p)$ es la parte cuadrática y $L_3(p) = p - L_0(p) - L_1(p) - L_2(p)$. Sea $e_\alpha(j) = e_\alpha(j_1) \cdot e_\alpha(j_2) = u \cdot v$, un paso del algoritmo, resultado de multiplicar dos anteriores. Entonces $e_\alpha(j) = u \cdot v = [L_0(u) + (u - L_0(u))] \cdot [L_0(v) + (v - L_0(v))] = L_0(u) \cdot L_0(v) + L_0(u) \cdot (v - L_0(v)) + L_0(v) \cdot (u - L_0(u)) + (u - L_0(u))(v - L_0(v))$.

Tenemos que $L_0(u) \cdot L_0(v)$ es un número racional, $L_0(u) \cdot (v - L_0(v))$ y $L_0(v) \cdot (u - L_0(u))$ son multiplicaciones por un número racional. Por lo tanto, podemos obtener un algoritmo β que tenga el mismo número de multiplicaciones que α y si $e_\beta(i) = w \cdot t$, para $w, t \in \mathbb{Q}[a_{11}, ..., b_{22}]$ dos polinomios, entonces $L_0(w) = L_0(t) = 0$, es decir, los polinomios w, t no tienen término constante. A su vez, como $L_2(w \cdot t) = L_2(w) \cdot L_0(t) + L_1(w) \cdot L_1(t) + L_0(w) \cdot L_2(t)$ y la parte constante es nula, se cumple que $L_2(w \cdot t) = L_1(w) \cdot L_1(t)$, es decir, la parte cuadrática del producto de polinomios $w \cdot t$ es el resultado del producto de la parte lineal de ambos polinomios. Si β es un algoritmo, por el Lema 1, para cada i y $j \in \{1, 2\}$ se cumple:

$$\sum_{k=1}^{2} a_{ik} b_{kj} = r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i=1}^{2} \sum_{j=1}^{2} r_{ij} a_{ij} + \sum_{i=1}^{2} \sum_{j=1}^{2} r'_{ij} b_{ij},$$

donde los elementos p_i son los resultados obtenidos de realizar multiplicaciones. Aplicamos L_2 a ambos lados, obteniendo la parte cuadrática de cada lado:

$$\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{i=1}^{m} r_i L_2(p_i).$$

Consideramos $p_i = q_i \cdot s_i$; donde q_i, s_i son dos polinomios. Entonces,

$$\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{i=1}^{m} r_i L_1(q_i) \cdot L_1(s_i).$$

Por lo tanto, podemos construir un algoritmo β que requiere m pasos y los argumentos de cada multiplicación son $L_1(q_i)$ y $L_1(s_i)$, que corresponden a la parte lineal de cada polinomio. Por el Lema 1, estos argumentos son de la forma

$$\sum_{j=1}^{2} \sum_{i=1}^{2} r_{ij} a_{ij} + \sum_{j=1}^{2} \sum_{i=1}^{2} r'_{ij} b_{ij}.$$

□

Lema 6. *Sean (v_1, v_2, v_3, v_4) y (u_1, u_2, u_3, u_4) dos vectores linealmente independientes de $\mathbb{Q}^4$. Entonces cualquier algoritmo que calcule*

$$f_1 = a_{11}(v_1 b_{11} + v_2 b_{12}) + a_{12}(v_1 b_{21} + v_2 b_{22}) + a_{21}(v_3 b_{11} + v_4 b_{12}) + a_{22}(v_3 b_{21} + v_4 b_{22})$$

$$f_2 = a_{11}(u_1b_{11} + u_2b_{12}) + a_{12}(u_1b_{21} + u_2b_{22}) + a_{21}(u_3b_{11} + u_4b_{12}) + a_{22}(u_3b_{21} + u_4b_{22})$$

requiere al menos 4 multiplicaciones.

Demostración. Haremos una demostración por reducción al absurdo. Suponemos que solo necesitamos 3 multiplicaciones para calcular f_1 y f_2.
Primero observamos que podemos escribir f_1, f_2 de la siguiente forma reordenando términos:

$$f_1 = b_{11}(v_1a_{11} + v_3a_{21}) + b_{12}(v_2a_{11} + v_4a_{21}) + b_{21}(v_1a_{12} + v_3a_{22}) + b_{22}(v_2a_{12} + v_4a_{22})$$

$$f_2 = b_{11}(u_1a_{11} + u_3a_{21}) + b_{12}(u_2a_{11} + u_4a_{21}) + b_{21}(u_1a_{12} + u_3a_{22}) + b_{22}(u_2a_{12} + u_4a_{22})$$

Por el Teorema 1, se deduce que los vectores

$$\begin{pmatrix} v_1b_{11} + v_2b_{12} \\ u_1b_{11} + u_2b_{12} \end{pmatrix}, \begin{pmatrix} v_1b_{21} + v_2b_{22} \\ u_1b_{21} + u_2b_{22} \end{pmatrix}, \begin{pmatrix} v_3b_{11} + v_4b_{12} \\ u_3b_{11} + u_4b_{12} \end{pmatrix}, \begin{pmatrix} v_3b_{21} + v_4b_{22} \\ u_3b_{21} + u_4b_{22} \end{pmatrix}$$

son linealmente dependientes. Por lo tanto, por el mismo razonamiento, los vectores

$$\begin{pmatrix} v_1a_{11} + v_3a_{21} \\ u_1a_{11} + u_3a_{21} \end{pmatrix}, \begin{pmatrix} v_2a_{11} + v_4a_{21} \\ u_2a_{11} + u_4a_{21} \end{pmatrix}, \begin{pmatrix} v_1a_{12} + v_3a_{22} \\ u_1a_{12} + u_3a_{22} \end{pmatrix}, \begin{pmatrix} v_2a_{12} + v_4a_{22} \\ u_2a_{12} + u_4a_{22} \end{pmatrix}$$

también son linealmente dependientes.
En consecuencia, existen $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{Q}$ de manera que se cumple

$$\begin{cases} \alpha_1 \begin{pmatrix} v_1 \\ u_1 \end{pmatrix} + \alpha_2 \begin{pmatrix} v_3 \\ u_3 \end{pmatrix} = 0 \\ \alpha_1 \begin{pmatrix} v_2 \\ u_2 \end{pmatrix} + \alpha_2 \begin{pmatrix} v_4 \\ u_4 \end{pmatrix} = 0 \end{cases} \qquad \begin{cases} \beta_1 \begin{pmatrix} v_1 \\ u_1 \end{pmatrix} + \beta_2 \begin{pmatrix} v_2 \\ u_2 \end{pmatrix} = 0 \\ \beta_1 \begin{pmatrix} v_3 \\ u_3 \end{pmatrix} + \beta_2 \begin{pmatrix} v_4 \\ u_4 \end{pmatrix} = 0 \end{cases}$$

Entonces, el espacio que contiene los vectores $\begin{pmatrix} v_1 \\ u_1 \end{pmatrix}, \begin{pmatrix} v_2 \\ u_2 \end{pmatrix}, \begin{pmatrix} v_3 \\ u_3 \end{pmatrix}$ y $\begin{pmatrix} v_4 \\ u_4 \end{pmatrix}$ es unidimensional.
Este hecho contradice la hipótesis de que los vectores (v_1, v_2, v_3, v_4) y (u_1, u_2, u_3, u_4) son linealmente independientes.

□

1.3. Demostración del número mínimo

Utilizando los lemas anteriores podemos demostrar cuantas multiplicaciones necesitamos como mínimo para realizar una multiplicación de dos matrices 2×2.

Teorema 7. *Todo algoritmo que calcule la multiplicación de dos matrices* 2×2 *requiere al menos* 7 *multiplicaciones.*

Demostración. Vamos a demostrarlo por reducción al absurdo. Suponemos que existe un algoritmo que requiere como máximo seis multiplicaciones.
Por el Lema 4 y Lema 5, existen seis productos $p_1, ..., p_6$ y veinticuatro números racionales r_{ij}^k con $i, j \in \{1, 2\}$, $k \in \{1, ..., 6\}$ tal que

$$\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{k=1}^{6} r_{ij}^k p_k \quad \text{para} \quad i, j \in \{1, 2\}.$$

Es decir, cada entrada de la matriz resultante es combinación lineal de los productos $p_1, ..., p_6$. Consideramos R la siguiente matriz 4×6:

$$R = \begin{pmatrix} r_{11}^1 & r_{11}^2 & \cdots & r_{11}^6 \\ r_{12}^1 & \cdots & \cdots & r_{12}^6 \\ r_{21}^1 & \cdots & \cdots & r_{21}^6 \\ r_{22}^1 & \cdots & \cdots & r_{22}^6 \end{pmatrix},$$

el vector $p = (p_1, ..., p_6)$ y el vector $c = (c_{11}, c_{12}, c_{21}, c_{22})$ que satisface $c_{ij} = \sum_{k=1}^{2} a_{ik} b_{kj}$; entonces se cumple que $R \cdot p = c$.
El rango de la matriz R tiene que ser 4, ya que ninguna combinación lineal de los c_{ij} se anula. Por tanto, existe una matriz 4×4, con coeficientes $D = (d_{ij})$ para $i, j \in \{1, 2, 3, 4\}$, que satisface

$$R' = D \cdot R = \begin{pmatrix} 1 & 0 & 0 & 0 & r_1 & s_1 \\ 0 & 1 & 0 & 0 & r_2 & s_2 \\ 0 & 0 & 1 & 0 & r_3 & s_3 \\ 0 & 0 & 0 & 1 & r_4 & s_4 \end{pmatrix}.$$

Sea $D \cdot R \cdot c = D \cdot c = t = (t_{11}, t_{12}, t_{21}, t_{22})$ donde $t_{ij} = d_{m1}c_{11} + d_{m2}c_{12} + d_{m3}c_{21} + d_{m4}c_{22}$ para $m \in \{1, 2, 3, 4\}$. En consecuencia, como $c_{ij} = \sum_{k=1}^{2} a_{ik} b_{kj}$, cada t_{ij} es de la siguiente forma: $t_{ij} = d_{m1}(a_{11}b_{11} + a_{12}b_{21}) + d_{m2}(a_{11}b_{12} + a_{12}b_{22}) + d_{m3}(a_{21}b_{11} + a_{22}b_{21}) + d_{m4}(a_{21}b_{12} + a_{22}b_{22}) = a_{11}(d_{m1}b_{11} + d_{m2}b_{12}) + a_{12}(d_{m1}b_{21} + d_{m2}b_{22}) + a_{21}(d_{m3}b_{11} + d_{m4}b_{12}) + a_{22}(d_{m3}b_{21} + d_{m4}b_{22})$. Cada t_{ij} se puede calcular usando 3 multiplicaciones; entonces $d_{m1}b_{11} + d_{m2}b_{12}$, $d_{m1}b_{21} + d_{m2}b_{22}$,

$d_{m3}b_{11} + d_{m4}b_{12}$ y $d_{m3}b_{21} + d_{m4}b_{22}$ son linealmente dependientes y se cumple

$$\begin{vmatrix} d_{m1}b_{11} + d_{m2}b_{12} & d_{m1}b_{21} + d_{m2}b_{22} \\ d_{m3}b_{11} + d_{m4}b_{12} & d_{m3}b_{21} + d_{m4}b_{22} \end{vmatrix} = 0;$$

$$\begin{vmatrix} d_{m1}b_{11} & d_{m2}b_{22} \\ d_{m3}b_{11} & d_{m4}b_{22} \end{vmatrix} + \begin{vmatrix} d_{m2}b_{12} & d_{m1}b_{21} \\ d_{m4}b_{12} & d_{m3}b_{21} \end{vmatrix} = b_{11}b_{22}\begin{vmatrix} d_{m1} & d_{m2} \\ d_{m3} & d_{m4} \end{vmatrix} + b_{12}b_{21}\begin{vmatrix} d_{m2} & d_{m1} \\ d_{m4} & d_{m3} \end{vmatrix}$$

$$= (b_{11}b_{22} + b_{12}b_{21})\begin{vmatrix} d_{m1} & d_{m2} \\ d_{m3} & d_{m4} \end{vmatrix} = det(B)\begin{vmatrix} d_{m1} & d_{m2} \\ d_{m3} & d_{m4} \end{vmatrix} = 0 \Rightarrow \begin{vmatrix} d_{m1} & d_{m2} \\ d_{m3} & d_{m4} \end{vmatrix} = 0$$

Por el Lema 3, sabemos que como máximo uno de los r_i y uno de los s_i es 0. Asumimos, sin perdida de generalidad, que $r_1 \neq 0$, $r_2 \neq 0$ y $r_3 \neq 0$. Como D es una matriz invertible se cumple que $rang\begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ d_{31} & d_{32} \end{pmatrix} = 2$ o $rang\begin{pmatrix} d_{13} & d_{14} \\ d_{23} & d_{24} \\ d_{33} & d_{34} \end{pmatrix} = 2$. Asumimos que (d_{11}, d_{12}) y (d_{21}, d_{22}) son linealmente independientes y además $(d_{31}, d_{32}) \neq 0$. Si fuera así, reemplazaríamos D por

$$D' = \begin{pmatrix} \frac{1}{s_1} & 0 & 0 & 0 \\ -\frac{s_2}{s_1} & 1 & 0 & 0 \\ -\frac{s_3}{s_1} & 0 & 1 & 0 \\ -\frac{s_4}{s_1} & 0 & 0 & 1 \end{pmatrix} \times D$$

obteniendo que R' es de la misma forma que $D' \cdot R$.

Como (d_{11}, d_{12}) y (d_{21}, d_{22}) son linealmente independientes y $(d_{31}, d_{32}) \neq 0$, se cumple que o bien (d_{11}, d_{12}) y (d_{31}, d_{32}) son linealmente independientes, o bien lo son (d_{21}, d_{22}) y (d_{31}, d_{32}). Suponemos, sin perdida de generalidad, que (d_{11}, d_{12}) y (d_{31}, d_{32}) son linealmente independientes, entonces $(d_{m3}, d_{m4}) = \lambda_m(d_{m1}, d_{m2})$ para $m = 1, 2, 3$; ya que (d_{m3}, d_{m4}) y (d_{m1}, d_{m2}) son linealmente dependientes.

Como cada expresión $t_2 - \frac{s_2}{s_1}t_1$ y $t_3 - \frac{s_2}{s_1}t_1$ se puede calcular usando 3 multiplicaciones, esto implica que

$$\begin{vmatrix} d_{21} - \frac{s_2}{s_1}d_{11} & d_{22} - \frac{s_2}{s_1}d_{12} \\ d_{23} - \frac{s_2}{s_1}d_{13} & d_{24} - \frac{s_2}{s_1}d_{14} \end{vmatrix} = 0 \tag{2}$$

$$\begin{vmatrix} d_{31} - \frac{s_3}{s_1}d_{11} & d_{32} - \frac{s_3}{s_1}d_{12} \\ d_{33} - \frac{s_3}{s_1}d_{13} & d_{34} - \frac{s_3}{s_1}d_{14} \end{vmatrix} = 0 \tag{3}$$

Usando las relaciones anteriores, obtenemos que la igualdad (2) se cumple si y solo si $\lambda_1 = \lambda_2$ y la (3) se cumple si y solo si $\lambda_1 = \lambda_3$. Sean α, β dos racionales que satisfacen $(d_{31}, d_{32}) = \alpha(d_{11}, d_{12}) + \beta(d_{21}, d_{22})$, y sabiendo que $\lambda_1 = \lambda_2 = \lambda_3$, tenemos que $(d_{33}, d_{34}) = \alpha(d_{13}, d_{14}) +$

$\beta(d_{23}, d_{24})$. Por las igualdades anteriores, se contradice que D sea invertible.
Por tanto, hemos encontrado una contradicción a la hipótesis que dos matrices 2×2 se pueden multiplicar usando como máximo seis multiplicaciones.

□

1.4. Algoritmos de multiplicación de matrices

A continuación, explicaremos algunos algoritmos para multiplicar matrices, considerando

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{21} & a_{22} & \cdots & a_{nn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{21} & b_{22} & \cdots & b_{nn} \end{pmatrix}$$

y C matrices $n \times n$, donde C es el resultado del producto $C = A \cdot B$.

1.4.1. Algoritmo estándar

El método usual para multiplicar matrices, el cual llamaremos estándar, sigue el siguiente procedimiento para el cálculo de cada coeficiente en el caso del producto de matrices $n \times n$:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \text{ para } i, j = 1, 2, ..., n$$

Veamos para el caso generar, en el que multiplicamos dos matrices $n \times n$, cuantas sumas y multiplicaciones son necesarias al efectuar el algoritmo. Es posible calcularlo teniendo en cuenta que, para obtener cada coeficiente se realizan n multiplicaciones y $n - 1$ sumas, para un total de n^2 coeficientes. Por lo tanto, obtenemos que al realizar el algoritmo se efectúan n^3 multiplicaciones y $n^2 \cdot (n - 1)$ sumas.

En el caso en que multiplicamos dos matrices 2×2, el algoritmo y la matriz resultante es de la siguiente forma:

$$C = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$

En esta situación, para realizar la multiplicación son necesarias 8 multiplicaciones y 4 sumas.

1.4.2. Algoritmo de Strassen

El algoritmo de Strassen proporciona otra forma de multiplicar matrices, la idea principal del algoritmo es reducir el número de multiplicaciones a costa de aumentar el número de sumas y crear variables auxiliares. El procedimiento para calcular el producto de matrices 2×2 es de la siguiente forma:

Empezamos creando las variables auxiliares m_i para $i = 1, 2, ..., 7$.

$m_1 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22})$

$m_2 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$

$m_3 = (a_{11} - a_{21}) \cdot (b_{11} + b_{12})$

$m_4 = (a_{11} + a_{12}) \cdot b_{22}$

$m_5 = a_{11} \cdot (b_{12} - b_{22})$

$m_6 = a_{22} \cdot (b_{21} - b_{11})$

$m_7 = (a_{21} + a_{22}) \cdot b_{11}$

El resultado de la multiplicación es:

$$C = \begin{pmatrix} m_1 + m_2 - m_4 + m_6 & m_4 + m_5 \\ m_6 + m_7 & m_2 - m_3 + m_5 - m_7 \end{pmatrix}$$

El método de Strassen se utiliza para matrices 2×2, aunque también se puede generalizar en matrices $2^k \times 2^k$ para alguna $k \in \mathbb{N}$. En este caso, se separan las matrices A, B en submatrices $\frac{2^k}{2} \times \frac{2^k}{2}$, es decir,

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

donde el primer coeficiente de cada submatriz A_{11}, A_{12}, A_{21} y A_{22} son a_{11}, $a_{1,2^k}$, $a_{2^{k-1}+1,1}$ y $a_{2^{k-1}+1,2^{k-1}+1}$ respectivamente. Luego se aplica el algoritmo de Strassen con estas submatrices como elementos, aplicando este método reiteradamente se obtiene el resultado.

Observamos que, con este método y para matrices 2×2, se necesitan 7 multiplicaciones y 18 sumas para realizar el producto de matrices. Comparado con el algoritmo estándar en el que necesitábamos 8 multiplicaciones y 4 sumas, se ahorra 1 multiplicación a costa de realizar 14 sumas más.

2. Tensores

El algoritmo de Strassen reduce la complejidad de la multiplicación a medida que aumenta el tamaño de las matrices. Este descubrimiento impulsó la búsqueda de algoritmos más eficientes para la multiplicación de matrices de diferentes tamaños. Como se ha visto en el capítulo anterior, podemos expresar la multiplicación de matrices mediante algoritmos; ahora, expresaremos este producto mediante tensores.

2.1. Definición de tensor

Introducimos el concepto de tensor como una generalización de las matrices.

Definición 8. *Un tensor T de dimensión n y orden $I_1 \times I_2 \times ... \times I_n$ para $I_i = 1, 2, ..., m_i$ con $m_i \geq 2$ para toda $i = 1, ..., n$ es un conjunto de $I_1 \cdot I_2 \cdot ... \cdot I_n$ elementos. Cada elemento del tensor lleva n subíndices $T = (t_{i_1 i_2 ... i_n})$; indicando la posición del elemento en el tensor.*
En el caso de un tensor de orden $n \times n \times ... \times n$ diremos que es de orden n.

Ejemplo 9. *En este trabajo, el caso más relevante será el de tensor tridimensional. Un tensor tridimensional T de orden $n \times m \times p$ es un conjunto de $n \cdot m \cdot p$ elementos ordenados en p matrices $n \times m$. Cada elemento del tensor lleva tres subíndices $T = (t_{ijk})$ para $i = 1, ..., n$, $j = 1, ..., m$ y $k = 1, ..., p$; el primero corresponde a la fila en la que se encuentra, el segundo indica la columna y el último la profundidad.*

Para poder visualizar este ejemplo representamos un tensor tridimensional de orden 3 que se asemeja a un cubo o una caja tridimensional dividida en pequeñas celdas. Cada celda corresponde a un elemento del tensor y su posición.

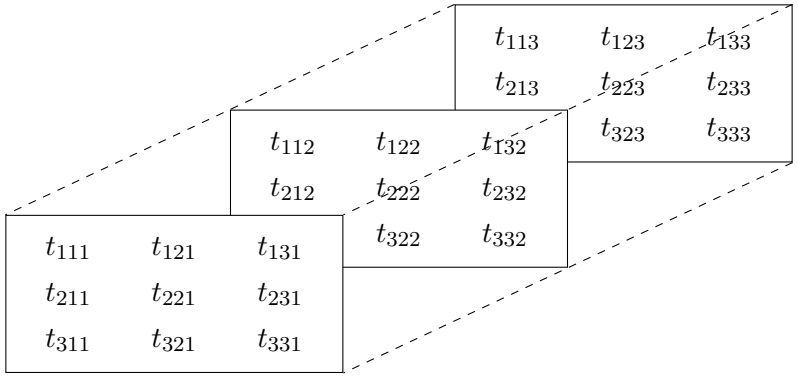


Figura 1: **Tensor tridimensional de orden 3**

2.2. Algoritmos como descomposición tensorial

Denotaremos la representación del proceso de multiplicación de matrices $n \times n$ como el tensor $\mathfrak{T}_n$. Este tensor es fijo, lo que significa que no varía según las matrices específicas que se estén

multiplicando. Sus elementos son valores binarios $\{0,1\}$ y su tamaño es de $n^2 \times n^2 \times n^2$.

En un contexto más general, cuando se multiplican matrices no cuadradas de dimensiones $n \times m$ y $m \times p$, denotamos el tensor por $\mathfrak{T}_{n,m,p}$. Basándonos en esta notación, el tensor $\mathfrak{T}_n$ se refiere al caso específico $\mathfrak{T}_{n,n,n}$. Definimos el producto tensorial (exterior) necesario para realizar una descomposición de un tensor.

Definición 10. *Sea $T = (t_{i_1 i_2 \ldots i_m})$ un tensor de dimensión m y orden $I_1 \times I_2 \times \ldots \times I_m$; y $S = (s_{j_1 j_2 \ldots j_n})$ un tensor de dimensión n y orden $J_1 \times J_2 \times \ldots \times J_n$.*
El producto tensorial (exterior) $T \otimes S$ es un tensor de dimensión $m+n$ y orden $I_1 \times I_2 \times \ldots \times I_m \times J_1 \times J_2 \times \ldots \times J_n$. Sus componentes satisfacen $(t \otimes s)_{i_1 i_2 \ldots i_m j_1 j_2 \ldots j_n} = t_{i_1 i_2 \ldots i_m} s_{j_1 j_2 \ldots j_n}$ donde i_k para $k = 1, \ldots, m$ son los índices que recorren las dimensiones de T y j_l para $l = 1, \ldots, n$ los índices que recorren las dimensiones de S.

Ejemplo 11. *Veamos un ejemplo con dos vectores $v = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}$ y $u = \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix}$. El producto tensorial $v \otimes u$ es un tensor de dimensión 2 y orden 3×3 donde $(v \otimes u)_{ij} = v_i u_j$ para $i, j = 1, 2, 3$:*

$$v \otimes u = \begin{pmatrix} v_1 u_1 & v_1 u_2 & v_1 u_3 \\ v_2 u_1 & v_2 u_2 & v_2 u_3 \\ v_3 u_1 & v_3 u_2 & v_3 u_3 \end{pmatrix}$$

Ejemplo 12. *Veamos ahora un ejemplo del producto tensorial (exterior) en el caso de tener una matriz y un vector. Sea $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ una matriz y $v = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}$ un vector.*
El producto tensorial $A \otimes v$ es un tensor tridimensional y de orden 3:

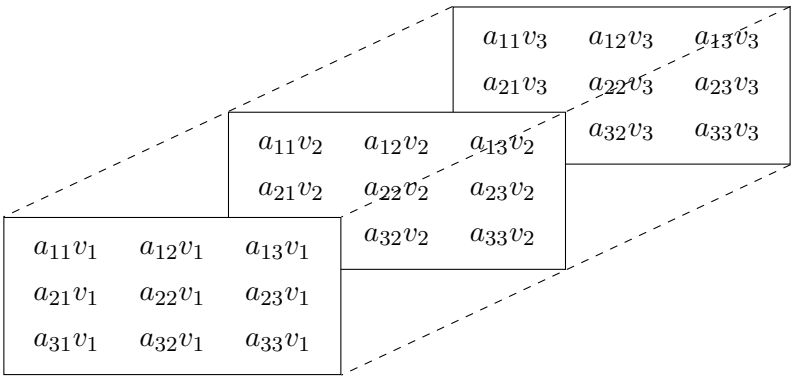


Figura 2: **Tensor tridimensional de orden 3**

La descomposición del tensor $\mathfrak{T}_n$ en R términos de rango uno se puede expresar como:

$$\mathfrak{T}_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)},$$

donde $\otimes$ representa el producto tensorial (exterior) y $u^{(r)}, v^{(r)}, w^{(r)}$ son vectores.

Definición 13. *El rango de un tensor tridimensional $\mathfrak{T}$ de orden n es el mínimo número de tensores de rango 1 en que podemos expresar $\mathfrak{T}$ como una combinación lineal de estos tensores.*

Por lo tanto, dada la descomposición anterior, decimos que si $\mathfrak{T}_n$ se puede descomponer en R términos de rango uno, entonces se cumple que $Rango(\mathfrak{T}_n) \leq R$. Además, R coincide con el número de multiplicaciones necesarias para realizar el producto de dos matrices.

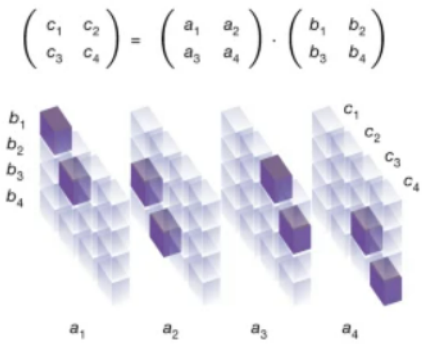


Figura 3: **Representación de la multiplicación de matrices.** [10]

La figura 3 muestra cómo la multiplicación de matrices 2×2 se representa mediante el tensor $\mathfrak{T}_2$. En este tensor, los valores establecidos son 1 en las posiciones coloreadas y 0 en las transparentes. Esta representación ilustra el algoritmo estándar de multiplicación, donde cada elemento c_i, correspondiente a las columnas de las cuatro submatrices, es el resultado de la suma de las multiplicaciones de los elementos a_i, b_i de cada columna coloreada, donde a_i corresponde a cada submatriz y b_i a cada fila de estas.

Por ejemplo, c_1 se obtiene de sumar a_1b_1 y a_2b_3, lo que se expresa como $c_1 = a_1b_1 + a_2b_3$.

La descomposición de este tensor $\mathfrak{T}_2$ es crucial para desarrollar nuevos algoritmos, como el algoritmo de Strassen.

Las matrices U, V y W, de la figura 4, se forman a partir de los vectores u, v, y w respectivamente, que surgen de la fórmula de descomposición de tensores. La matriz U consta de cuatro filas correspondientes a las entradas de la primera matriz a multiplicar, a_i, y siete columnas que representan los pasos previos, es decir, las variables auxiliares m_i. De manera análoga, se construye la matriz V con los elementos de la segunda matriz a multi-

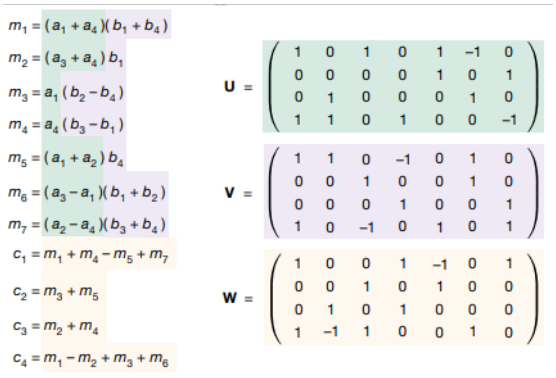


Figura 4: **Algoritmo de Strassen.** [10]

plicar. Por último, la matriz W tiene cuatro filas
que corresponden a las cuatro entradas de la ma-
triz solución del producto y las columnas que indican las variables auxiliares m_i, señalando cuáles hay que sumar para obtener el resultado.

En la figura 4 se pueden apreciar los vectores obtenidos a partir de la descomposición tensorial. Sin embargo, en la figura 5, se presentan los siete tensores de rango uno que, al sumarse, conforman el tensor inicial $\mathfrak{T}_2$.

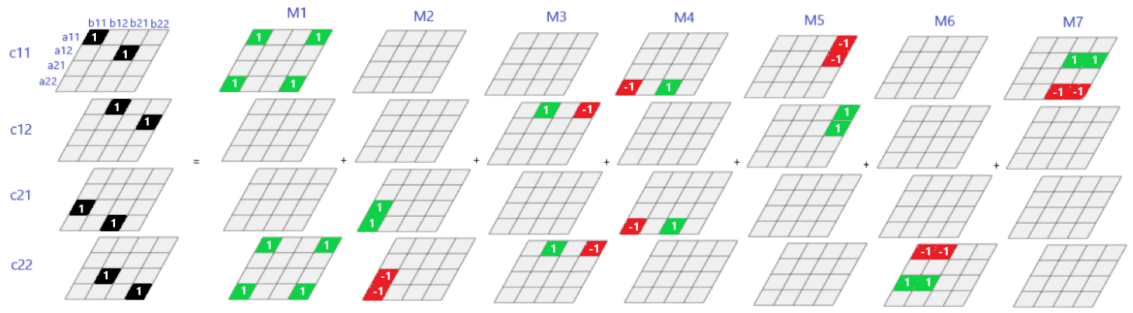


Figura 5: **Representación algoritmo de Strassen.**

La descomposición de un tensor en R términos de rango uno proporciona un algoritmo para calcular el producto de matrices $AB = C$, donde $A = (a_{ij})$ y $B = (b_{ij})$ pertenecen a $\mathcal{M}_{n\times n}$. Este algoritmo utiliza como parámetros un conjunto de vectores $\{u^{(r)}, v^{(r)}, w^{(r)}\}_{r=1}^{R}$, cada uno de longitud n^2, y un valor R que controla el número de multiplicaciones necesarias para realizar este producto. Por lo tanto, el siguiente algoritmo (1) proporciona la matriz resultante de multiplicar AB a partir del valor R y un conjunto de vectores especificados.

Algorithm 1

for $r = 1, ..., R$ **do**
$m_r \leftarrow (u_1^{(r)} a_1 + ... + u_{n^2}^{(r)} a_{n^2})(v_1^{(r)} b_1 + ... + v_{n^2}^{(r)} b_{n^2})$
end for
for $i = 1, ..., n^2$ **do**
$c_i \leftarrow w_i^{(1)} m_1 + ... + w_i^{(R)} m_R$
end for
return C

Como hemos visto previamente, un algoritmo de multiplicación puede ser expresado como un tensor, el cual a su vez se puede descomponer en una suma de tensores de rango uno.

Esta forma de expresar los tensores nos permite utilizar técnicas avanzadas de inteligencia artificial para descubrir nuevos algoritmos de multiplicación más eficiente. Esta optimización permite mejorar el rendimiento de cálculos computacionales.

3. Inteligencia artificial

El objetivo de este capítulo es, mediante el uso de tensores, encontrar algoritmos eficientes utilizando la inteligencia artificial. Para alcanzar este propósito, explicaremos qué es el aprendizaje por refuerzo y cómo se utiliza la búsqueda de árboles de Monte Carlo en esta tarea.

3.1. Aprendizaje por refuerzo

El aprendizaje por refuerzo se ubica dentro del aprendizaje automático, donde un agente aprende a través de la interacción y la experiencia con un entorno. Este enfoque se centra en aprender qué acciones tomar para maximizar una recompensa numérica.

Definimos los integrantes que el agente utiliza. Consideramos los estados en cada paso t, a tiempo discreto, $S_t \in \mathcal{S}$, donde $\mathcal{S}$ representa el conjunto de los estados posibles, y las acciones en cada paso $A_t \in \mathcal{A}(S_t)$, donde $\mathcal{A}(S_t)$ es el conjunto de acciones disponibles en el estado S_t. En el paso $t+1$ el agente recibe una recompensa numérica, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

Algunas características distintivas del aprendizaje por refuerzo [8] son su naturaleza de ciclo cerrado, donde las acciones del agente influyen en las futuras entradas. Además, el agente debe descubrir, mediante prueba y error, qué acciones generan la mayor recompensa. Es crucial destacar que estas acciones no solo impactan en la recompensa inmediata, sino también en situaciones futuras, lo que añade una complejidad adicional al proceso de aprendizaje.

Un desafío propio del aprendizaje por refuerzo, no tan común en otras áreas del aprendizaje automático, es el equilibrio entre exploración y explotación. Para obtener una gran recompensa, el agente debe elegir acciones que han sido efectivas en el pasado, lo que implica explotar el conocimiento adquirido. Sin embargo, para descubrir acciones aún más efectivas, el agente también debe explorar nuevas posibilidades. En resumen, el agente se enfrenta a un dilema constante entre elegir acciones conocidas para maximizar la recompensa actual y probar nuevas acciones para mejorar el rendimiento futuro. La clave radica en encontrar el equilibrio adecuado entre exploración y explotación para obtener resultados óptimos en la resolución de problemas de aprendizaje por refuerzo.

Procesos de decisiones de Markov

El aprendizaje por refuerzo se formaliza como un proceso de decisión de Markov. Un agente, que aprende y toma las decisiones, y un entorno, donde se encuentran las acciones y los

estados, interactúan en cada paso a tiempo discreto. Es decir, en cada paso t, el agente se encuentra en un estado $S_t \in \mathcal{S}$, selecciona una acción $A_t \in \mathcal{A}(S_t)$ y recibe una recompensa numérica, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

Además, el agente implementa una política, $\pi_t(a|s)$, que es la probabilidad de escoger la acción a si estamos en el estado s. El objetivo del agente será maximizar la recompensa que recibe a largo plazo, denotamos G_t la función del rendimiento a partir de un instante t, es decir, $G_t = R_{t+1}+R_{t+2}+...+R_T$, donde T es el tiempo del último paso dado. Otro concepto similar es el de descuento, donde el agente selecciona las acciones para maximizar la suma de recompensas de descuento que recibirá en el futuro, $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, donde γ es un parámetro, $0 \leq \gamma \leq 1$, llamado la tasa de descuento.

Para definir un proceso de decisión de Markov primero tenemos que definir la propiedad de Markov. Esta propiedad nos dice que conocido el presente, el conocimiento del pasado no importa para conocer el futuro. Es decir, para todas las $r \in R, s \in S$ y todos los posibles valores del pasado $S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$, se cumple

$$\mathbb{P}(R_{t+1} = r, S_{t+1} = s|S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t) = \mathbb{P}(R_{t+1} = r, S_{t+1} = s|S_t, A_t).$$

Una tarea de aprendizaje por refuerzo que satisface la propiedad de Markov se llama proceso de decisión de Markov. Además, si los espacios de estados y de acciones son finitos, los llamaremos proceso de decisión de Markov finito, que son los que utilizaremos.

Denotamos a partir de un estado y acción, s y a, la probabilidad del siguiente estado y su recompensa como:

$$p(s', r|s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a).$$

En consecuencia, podemos calcular la recompensa esperada para cada estado y acción

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r|s, a).$$

Otra característica de los algoritmos del aprendizaje por refuerzo es la estimación de funciones de valor, que estiman lo beneficioso que es para el agente encontrarse en un determinado estado. El valor, $v_\pi(s)$, de un estado s bajo una política π es el rendimiento esperado cuando

se comienza en el estado s y se utiliza π para escoger las siguientes acciones. Se define como

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

Análogamente, definimos el valor de tomar una acción, a, en un estado s bajo una política π como el rendimiento esperado:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a].$$

En resumen, utilizaremos los procesos de decisión de Markov en los casos que tengamos un problema de aprendizaje por refuerzo en el que cada paso solo dependa del estado y la información actual.

3.2. Monte Carlo en AlphaZero

La búsqueda de árboles de Monte Carlo se originó como un método de búsqueda heurística, donde se abordan juegos combinatorios empleando una estrategia de búsqueda en árbol basada en muestreo aleatorio en lugar de hacer una exploración completa. Este enfoque utiliza simulaciones aleatorias para explorar el espacio de búsqueda y decidir las mejores acciones en cada paso. Al centrarse en los nodos más relevantes con mayor frecuencia se obtienen estadísticas más precisas, mejorando así los resultados.

El algoritmo opera mediante la construcción de un árbol de búsqueda a partir del estado actual del juego o problema. Cada nodo del árbol representa un estado s del juego y está asociado con un valor, $v_\pi(s)$, que refleja la utilidad esperada de ese estado. El proceso está compuesto por cuatro etapas principales:

- **Selección:** Iniciando desde la raíz del árbol queremos explorar los nodos del árbol que nos lleven al mejor resultado, por lo tanto, buscamos una forma de dar valor a los nodos para seleccionar el más adecuado.
- **Expansión:** Una vez seleccionado un nodo (es decir, un estado del juego) buscamos expandir el árbol considerando todas las siguientes acciones posibles conocidas en ese momento.
- **Simulación:** Se simulan secuencias de jugadas desde los nodos recién expandidos hasta alcanzar un estado terminal o un límite de profundidad. Durante estas simulaciones, se emplea una política de juego aleatoria o heurística para determinar los movimientos.
- **Retropropagación:** Una vez alcanzado el estado terminal, los resultados obtenidos

deben propagarse hasta la raíz del árbol para almacenar la información y mejorar las estimaciones de cada estado.

Los métodos de Monte Carlo revelan y promedian los rendimientos para cada par estado-acción. Consideramos los métodos de Monte Carlo para aprender la función de valor, $v_\pi(s)$, de un estado s para una política determinada π. Una forma de estimarla es promediando los rendimientos R_t para $t > t_s$, siendo t_s el paso en el que se encuentra en el estado s, es decir, los observados después de las visitas a ese estado.

Ahora, implementamos la búsqueda de árboles de Monte Carlo con AlphaZero. Aunque el algoritmo de búsqueda de árboles de Monte Carlo es superior a la búsqueda por fuerza bruta, se enfrenta al problema de manejar una gran cantidad de estados válidos, lo que puede producir que el tiempo requerido sea poco factible.

Para abordar este problema, AlphaZero propone utilizar una red neuronal donde en cada estado devuelve una política, π, adecuada y un valor, $v_\pi(s)$, para escoger mejores acciones y mejorar el rendimiento en el juego. Inicialmente, AlphaZero inicia con una red neuronal con pesos aleatorios. Para perfeccionar su estrategia de juego, se recurre al aprendizaje por refuerzo. Este método implica que AlphaZero juegue en numerosas partidas y luego ajustar los pesos de la red neuronal con el objetivo de codificar qué jugadas son más efectivas y cuáles no lo son.

3.3. El aprendizaje por refuerzo para descubrir nuevos algoritmos

Volvemos ahora a las descomposiciones de los tensores para encontrar algoritmos eficientes descomponiendo el tensor $\mathfrak{T}_n$ con el menor rango R posible. Se plantea como un problema de aprendizaje por refuerzo con un solo agente, AlphaTensor, en un espacio de un juego para un jugador, TensorGame. AlphaTensor se basa en AlphaZero, que logró y aprendió a jugar a juegos como el ajedrez, el Go y el shogi.

El juego funciona siguiendo los siguientes pasos:

1. Comenzamos la partida en el estado inicial con el tensor conocido $\mathfrak{T}_n$, denotado como $S_0 = \mathfrak{T}_n$.

2. Durante el juego, en cada paso t, el jugador o agente selecciona un conjunto $\{u^{(t)}, v^{(t)}, w^{(t)}\}$. Se actualiza el estado S_t restando al tensor del estado anterior el tensor de rango uno formado por los vectores del conjunto seleccionado. Es decir, $S_{t+1} = S_t - u^{(t+1)} \otimes v^{(t+1)} \otimes w^{(t+1)}$.

3. El objetivo del juego es obtener el tensor cero utilizando el menor número de pasos posible. Por lo tanto, a lo largo del juego, encontramos una factorización del tensor inicial $S_0 = \sum_{t=1}^{R} u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$ que llega a $S_R = \mathbf{0}$.

Si prolongamos excesivamente el juego, nos enfrentamos a una descomposición con un valor de R considerablemente elevado, lo que resultará en un algoritmo poco eficaz. Para evitar esta posibilidad, se establece un límite máximo de pasos, denotado por R_{limit}.

En cada turno del juego se aplica una penalización de -1 para promover una duración más corta del juego, lo que ayuda a encontrar el camino más corto hacia el tensor $\mathbf{0}$. Además, si después de alcanzar el límite de pasos R_{limit}, el tensor resultante es diferente del tensor $\mathbf{0}$, el agente recibe una penalización adicional. Es común también restringir las entradas de los vectores $\{u^{(t)}, v^{(t)}, w^{(t)}\}$ a un conjunto discreto E especificado por el usuario, por ejemplo, $E = \{-1, 0, 1\}$.

Al igual que AlphaZero, AlphaTensor emplea una red neuronal profunda para dirigir un algoritmo de búsqueda de árboles de Monte Carlo. En cada estado, la red neuronal estima una política, que es una distribución de probabilidad sobre las acciones, y un valor, que es la estimación del rendimiento futuro del estado actual. Este valor deriva de las penalizaciones previas y proporciona una estimación de la distribución de los rendimientos a partir del estado actual S_t, lo que ofrece una idea sobre el rango del tensor S_t.

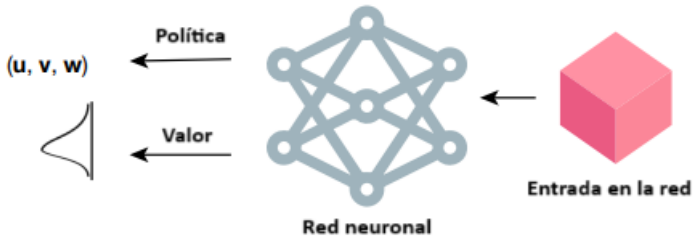


Figura 6: **Red neuronal.** [10]

AlphaTensor inicia el juego desde el tensor $\mathfrak{T}_n$ y, en cada paso, utiliza un árbol de Monte Carlo para elegir la siguiente acción. Los juegos terminados se emplean como información para la red, con el objetivo de mejorar sus parámetros.

Además, utiliza juegos sintéticos como información. Estos juegos se generan a partir de la descomposición de tensores. Aunque descomponer un tensor es un proceso complejo, la tarea inversa, es decir, construir un tensor a partir de factores de rango uno, es sencilla: solo hay que realizar el producto de estos factores y sumar el resultado. Por lo tanto, podemos generar, aleatoriamente, un conjunto de factores $\{u^{(t)}, v^{(t)}, w^{(t)}\}_{t=1}^{R}$ y calcular el tensor

$\mathfrak{D} = \sum_{t=1}^{R} u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$. Estos tensores y su descomposición serán los juegos sintéticos, que ayudaran a mejorar la complejidad de la red neuronal.

Otro aspecto que contribuye a mejorar el rendimiento del juego son los cambios de base. El tensor representado en la figura 3 es la representación de la multiplicación de matrices en la base canónica. Es posible reescribir este tensor en otra base, siendo equivalente al tensor inicial debido a que tienen el mismo rango. La descomposición obtenida al representar el tensor en otra base se convierte en un algoritmo de multiplicación (utilizando el Algoritmo 1) al cambiar a la base canónica. Al inicio de cada juego, se aplica a $\mathfrak{T}_n$ un cambio de base aleatorio y luego AlphaTensor juega sobre esta base. Este cambio proporciona diversidad en los juegos que el agente puede jugar y amplía el conjunto de posibilidades, ya que no es necesario que el agente encuentre una descomposición en todas las bases; es suficiente con que encuentre una de bajo rango en al menos una de ellas.

En resumen, comenzamos el juego con el tensor $\mathfrak{T}_n$ cambiándole la base. En cada paso se realiza un árbol de búsqueda de Monte Carlo para elegir la siguiente acción. AlphaTensor toma la acción y repite el proceso hasta obtener $S_R = \mathbf{0}$ o alcanzar el paso R_{limit}, obteniendo una jugada. Estas jugadas, junto con las jugadas sintéticas, actualizan y entrenan la red neuronal. Esta red actualizada la usará el método de Monte Carlo para generar nuevos juegos. Este proceso puede ilustrarse a través del siguiente diagrama.

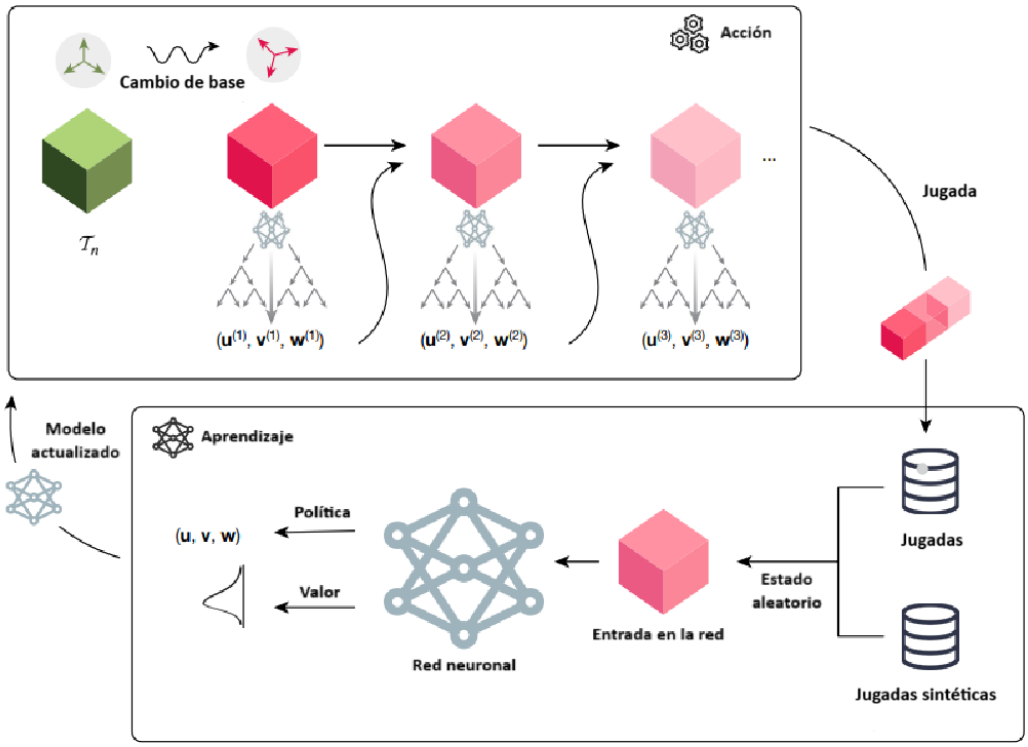


Figura 7: **Diagrama del juego.** [10]

3.4. Resultados del algoritmo

Los nuevos algoritmos de multiplicación se descubren utilizando los métodos de descomposición y el aprendizaje por refuerzo para matrices de tamaño $n \times m$ y $m \times p$ para $n, m, p \leq 5$.

Los algoritmos descubiertos por AlphaTensor mejoran los algoritmos conocidos para diferentes tamaños, ya que reducen el número de multiplicaciones requeridas. Es importante recordar que el número de multiplicaciones necesarias para llevar a cabo el producto de matrices es equivalente al rango del tensor asociado. La siguiente tabla presenta una comparación entre la complejidad de los algoritmos conocidos y los descubiertos por AlphaTensor.

Tamaño (n,m,p)	Mejor método conocido	Mejor rango conocido	AlphaTensor modular	AlphaTensor estándar
(2,2,2)	Strassen, 1969 [2]	7	7	7
(3,3,3)	Laderman, 1976 [6]	23	23	23
(4,4,4)	Strassen, 1969 [2]	49	47	49
(5,5,5)	(3,5,5)+(2,5,5)	98	96	98
(2,2,3)	(2,2,2)+(2,2,1)	11	11	11
(2,2,4)	(2,2,2)+(2,2,2)	14	14	14
(2,2,5)	(2,2,2)+(2,2,3)	18	18	18
(2,3,3)	Hopcroft y Kerr, 1971 [4]	15	15	15
(2,3,4)	Hopcroft y Kerr, 1971 [4]	20	20	20
(2,3,5)	Hopcroft y Kerr, 1971 [4]	25	25	25
(2,4,4)	Hopcroft y Kerr, 1971 [4]	26	26	26
(2,4,5)	Hopcroft y Kerr, 1971 [4]	33	33	33
(2,5,5)	Hopcroft y Kerr, 1971 [4]	40	40	40
(3,3,4)	Smirnov, 2013 [7]	29	29	29
(3,3,5)	Smirnov, 2013 [7]	36	36	36
(3,4,4)	Smirnov, 2013 [7]	38	38	38
(3,4,5)	Smirnov, 2013 [7]	48	47	47
(3,5,5)	Sedoglavic y Smirnov, 2021 [9]	58	58	58
(4,4,5)	(4,4,2)+(4,4,3)	64	63	63
(4,5,5)	(2,5,5) $\otimes$ (2,1,1)	80	76	76

Cuadro 1: Tabla comparativa entre algoritmos. [10]

En la primera columna se muestran las dimensiones de las matrices que se multiplicarán, expresadas en la forma (n, m, p), donde se refiere a dos matrices de tamaño $n \times m$ y $m \times p$ respectivamente. En la segunda y tercera columna se indican el mejor algoritmo conocido y su correspondiente rango. Finalmente, en la cuarta y quinta columna se presentan los rangos de los algoritmos obtenidos con AlphaTensor en aritmética modular ($\mathbb{Z}_2$) y aritmética estándar, respectivamente.

La tabla muestra que, para matrices cuadradas, representadas en las primeras cuatro filas, AlphaTensor mejora el rango conocido en dos casos cuando se utiliza únicamente aritmética modular. También se observan mejoras en el rango en el caso de matrices no cuadradas, en este caso con ambas aritméticas.

Es importante destacar que, en los casos en los que AlphaTensor no mejora el rango conocido, redescubre el algoritmo que proporciona el mejor rango conocido. En otras palabras, no se encuentra ningún algoritmo obtenido por AlphaTensor con un rango mayor que los algoritmos ya conocidos.

4. Más allá de la IA

Antes de la inteligencia artificial se usaban métodos computacionales clásicos utilizando tensores, combinando estos métodos con la inteligencia artificial han conseguido llegar mas lejos. Han conseguido mejorar el algoritmo encontrado por AlphaTensor para matrices de orden 5 en aritmética modular ($\mathbb{Z}_2$) de 96 a 95 multiplicaciones. La solución tiene la siguiente forma:

El algoritmo contiene unas matrices fijas $\alpha_{ij}^{(k)}, \beta_{ij}^{(k)}, \gamma_{ij}^{(k)}$ para $1 \leq i, j \leq 5$ y $1 \leq k \leq 95$ formadas por ceros y entre uno y diez unos, que se pueden encontrar explícitamente en [12, pp. 5-14]. Dadas las matrices $A = (a_{ij}), B = (b_{ij}) \in \mathcal{M}_{5\times5}(\mathbb{Z}_2)$ el algoritmo calcula las variables auxiliares m_k para $k = 1, ..., 95$ y la matriz $C = (c_{ij}) \in \mathcal{M}_{5\times5}(\mathbb{Z}_2)$ que es el producto $C = AB$. Entonces,

$$m_k = \left(\sum_{i=1}^{5}\sum_{j=1}^{5} \alpha_{ij}^{(k)} a_{ij}\right)\left(\sum_{i=1}^{5}\sum_{j=1}^{5} \beta_{ij}^{(k)} b_{ij}\right),$$

para $k = 1, ..., 95$ y

$$c_{ij} = \sum_{k=1}^{95} \gamma_{ij}^{(k)} m_k,$$

para $i, j = 1, ..., 5$.

Veamos cómo se encuentran estos métodos [13]: cada algoritmo corresponde a un esquema de multiplicación (Definición 15), cada esquema representa un vértice de un grafo cuyas aristas pueden ser un *flip*, que convierte un esquema en otro diferente con el mismo número de multiplicaciones o una reducción, que conecta un esquema con otro con un número menor de multiplicaciones.

Recordamos que habíamos definido el tensor de multiplicación de matrices (n, n, n) como

$$\mathfrak{T}_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)},$$

donde $u^{(r)}, v^{(r)}, w^{(r)}$ son vectores y $\otimes$ representa el producto tensorial (externo). Para simplificar las explicaciones siguientes, cambiaremos la notación utilizada, considerando matrices en lugar de vectores. Al fin y al cabo, una matriz puede verse como un vector dispuesto de tal manera que permite realizar multiplicaciones.

Definición 14. *Sea $n, m, p \in \mathbb{N}$ y K un cuerpo. El tensor de multiplicación de matrices*

(n, m, p) *se define por*

$$\mathfrak{T}_{n,m,p} = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{p} a_{i,j} \otimes b_{j,k} \otimes c_{k,i}$$

donde $(a_{i,j}) \in \mathcal{M}_{n\times m}(K)$, $(b_{i,j}) \in \mathcal{M}_{m\times p}(K)$ *y* $(c_{i,j}) \in \mathcal{M}_{p\times n}(K)$. *Además, estas matrices son de tal forma que tienen un* 1 *en la posición* (i, j) *y ceros en las demás posiciones. Este tensor corresponde al producto* $C^T = AB$.

Recordamos también que el rango de un tensor es el número más pequeño, R, tal que el tensor se puede expresar como una suma de R tensores de rango uno 13. Los tensores de rango uno son distintos de cero y se pueden representar como $A \otimes B \otimes C$ donde $A \in \mathcal{M}_{n\times m}(K)$, $B \in \mathcal{M}_{m\times p}(K)$ y $C \in \mathcal{M}_{p\times n}(K)$.

4.1. Esquemas de multiplicación

Definamos que es un esquema de multiplicación y cómo son los esquemas obtenidos por una reducción y por un *flip*.

Definición 15. *Un esquema de multiplicación de matrices* (n, m, p) *es un conjunto finito* $S = \{A^{(i)} \otimes B^{(i)} \otimes C^{(i)} | i \in \{1, ..., r\}\}$, *formado por tensores de rango uno, cuya suma es* $\mathfrak{T}_{n,m,p}$, *el tensor de multiplicación de matrices* (n, m, p).

Sabemos, además, que la suma de k tensores de rango uno no forma necesariamente un tensor de rango k. Un ejemplo de este caso sería la ecuación

$$\begin{aligned} &a_{1,1} \otimes b_{1,1} \otimes c_{1,1} + a_{1,1} \otimes b_{12} \otimes c_{12} + a_{1,1} \otimes b_{2,1} \otimes (c_{1,1} + c_{1,2}) \\ &= a_{1,1} \otimes (b_{1,1} + b_{2,1}) \otimes c_{1,1} + a_{1,1} \otimes (b_{1,2} + b_{2,1}) \otimes c_{1,2}. \end{aligned}$$

Podemos generalizar este concepto definiendo los esquemas de multiplicación de matrices reducibles.

Definición 16. *Sean* $n, m, p, r \in \mathbb{N}$, K *un cuerpo y* $S = \{A^{(i)} \otimes B^{(i)} \otimes C^{(i)} | i \in \{1, ..., r\}\}$ *un esquema de multiplicación de matrices* (n, m, p). *Entonces* S *es un esquema reducible si existe un conjunto no vacío* $I \subset \{1, ..., r\}$ *que satisface*

1. $dim_K(A^{(i)})_{i\in I} = 1$
2. $\{B^{(i)} | i \in I\}$ *es linealmente dependiente sobre* K

Proposición 17. *Sean* $n, m, p, r \in \mathbb{N}$, K *un cuerpo y* S *un esquema de multiplicación de matrices* (n, m, p) *reducible de rango* r. *Entonces existe un esquema de multiplicación de matrices* (n, m, p) *de rango* $r - 1$.

Demostración. Tenemos que S es un esquema reducible, por lo tanto, se cumplen las condiciones de la definición 16. Como $\{B^{(i)}|i \in I\}$ es linealmente dependiente, existe un $t \in I$ tal que $B^{(t)} = \sum_{i\in I\setminus\{t\}} \beta_i B^{(i)}$ para $\beta_i \in K$. Además, como $dim_K(A^{(i)})_{i\in I} = 1$ existen $\alpha_i \in K$ tal que $A^{(t)} = \alpha_i A^{(i)}$.
Entonces,

$$\begin{aligned}\sum_{i\in I} A^{(i)} \otimes B^{(i)} \otimes C^{(i)} &= \sum_{i\in I\setminus\{t\}} A^{(i)} \otimes B^{(i)} \otimes C^{(i)} + A^{(t)} \otimes B^{(t)} \otimes C^{(t)} \\ &= \sum_{i\in I\setminus\{t\}} A^{(i)} \otimes B^{(i)} \otimes C^{(i)} + \alpha_i A^{(i)} \otimes \sum_{i\in I\setminus\{t\}} \beta_i B^{(i)} \otimes C^{(t)} \\ &= \sum_{i\in I\setminus\{t\}} A^{(i)} \otimes B^{(i)} \otimes C^{(i)} + \sum_{i\in I\setminus\{t\}} A^{(i)} \otimes B^{(i)} \otimes \alpha_i\beta_i C^{(t)} \\ &= \sum_{i\in I\setminus\{t\}} A^{(i)} \otimes B^{(i)} \otimes (C^{(t)} + \alpha_i\beta_i C^{(t)}).\end{aligned}$$

Por lo tanto,

$$S' = \{A^{(i)} \otimes B^{(i)} \otimes C^{(i)}|i \in \{1, ..., r\}\setminus I\} \cup \{A^{(i)} \otimes B^{(i)} \otimes (C^{(i)} + \alpha_i\beta_i C^{(t)})|i \in I\setminus\{t\}\}$$

es un esquema de multiplicación de rango $r - 1$.

□

A partir de la demostración de la proposición 17 podemos definir la reducción de un esquema S.

Definición 18. *Dado un esquema $S = \{A^{(i)} \otimes B^{(i)} \otimes C^{(i)}|i \in \{1, ..., r\}\}$, decimos que S' es la reducción de S si podemos expresarlo de la siguiente manera:*

$$S' = \{A^{(i)} \otimes B^{(i)} \otimes C^{(i)}|i \in \{1, ..., r\}\setminus I\} \cup \{A^{(i)} \otimes B^{(i)} \otimes (C^{(i)} + \alpha_i\beta_i C^{(t)})|i \in I\setminus\{t\}\}.$$

A partir de un esquema de multiplicación podemos obtener otro esquema a través de diferentes transformaciones. Consideremos K un cuerpo y expresemos el tensor de rango uno mediante el producto $A \otimes B \otimes C$ donde $A \in \mathcal{M}_{n\times m}(K)$, $B \in \mathcal{M}_{m\times p}(K)$ y $C \in \mathcal{M}_{p\times n}(K)$, decimos que dos esquemas son equivalentes si se pueden conectar mediante una de las siguientes transformaciones [11, pp. 3-5]. La primera, basada en la igualdad $(AB)^T = B^T A^T$, intercambia los tensores de rango uno $A \otimes B \otimes C$ por $B^T \otimes A^T \otimes C^T$. La segunda reemplaza el tensor de rango uno $A \otimes B \otimes C$ por $B \otimes C \otimes A$. Por último, si consideramos una matriz invertible $U \in \mathcal{M}_{m\times m}(K)$, donde se satisface $AB = AUU^{-1}B$, entonces podemos reemplazar el tensor de rango uno $A\otimes B\otimes C$ por $AU \otimes U^{-1}B\otimes C$, obteniendo así un esquema de multiplicación. Estas transformaciones generan el grupo de simetría de esquemas de multiplicación (n, m, p) [11].

Dado que la reducibilidad se preserva mediante la simetría, para obtener un esquema reducible a partir de uno irreducible, se utiliza una transformación que, en general, no preserva la simetría: el *flip*. La idea consiste en restar una componente de un tensor de rango uno y añadirlo a otro, lo cual es posible cuando dos tensores de rango uno comparten un factor en común. Por ejemplo,

$$A \otimes B \otimes C + A \otimes B' \otimes C' = A \otimes (B + B') \otimes C + A \otimes B' \otimes (C' - C).$$

Como este tipo de transformaciones no afecta a todos los elementos del esquema podemos esperar que el esquema inicial y el esquema resultante no sean equivalentes.

Definición 19. *Sean $n, m, p, r \in \mathbb{N}$ y S, S' dos esquemas de multiplicación de rangos r. Decimos que S' es un flip de S si existen*

1. $T_1 = A \otimes B \otimes C \in S$,
2. $T_2 = A \otimes B' \otimes C' \in S$,
3. $T \in \{A \otimes B \otimes C', A \otimes B' \otimes C\}$,

de manera que $(S \backslash \{T_1, T_2\}) \cup \{T_1 + T, T_2 - T\} = S'$.
Esta definición se puede aplicar para cualquier permutación de A, B y C.

La unión de $(S \backslash \{T_1, T_2\}) \cup \{T_1 + T, T_2 - T\}$ es siempre disjunta, ya que S y S' son dos esquemas con el mismo rango. Además, estos cambios son reversibles porque si $S' = (S \backslash \{T_1, T_2\}) \cup \{T_1 + T, T_2 - T\}$ entonces $S = (S' \backslash \{T_1 + T, T_2 - T\}) \cup \{(T_2 - T) + T, (T_1 + T) - T\}$.

Esta definición nos indica los cambios que se requieren para reemplazar dos tensores de rango uno en un esquema formado por otros tensores de rango uno.

4.2. El grafo *flip*

Veamos como son los grafos formados por los esquemas de multiplicación y los tipos de aristas.

Definición 20. *Sean $n, m, p \in \mathbb{N}$ y Z el conjunto de todas las órbitas de los esquemas de multiplicación de matrices (n, m, p) bajo el grupo de simetría y definimos*

$$E_1 = \{(S, S') | S' \text{ es un flip de } S\}$$
$$E_2 = \{(S, S') | S' \text{ es una reducción de } S\}$$

Entonces,

1. *El grafo* $G = (Z, E_1 \cup E_2)$ *lo llamamos el grafo flip. Las aristas en* E_1 *las llamamos flips y las de* E_2 *reducciones.*

2. *Para un* $r \in \mathbb{N}$ *dado, el subgrafo de* G *formado por el conjunto de vértices* $\{S \in Z : rango(S) \leq r\}$ *lo llamamos el grafo flip de rango como máximo* r*.*

3. *Para un* $r \in \mathbb{N}$ *dado, el conjunto* $\{S \in Z : \ rango(S) = r\}$ *lo llamamos el nivel r-ésimo de* G*.*

Observamos que los *flips* conectan dos vértices del mismo nivel, siendo bidireccional. En cambio, las reducciones conectan un vértice con otro de un nivel inferior, es decir, con un esquema con un número menor de multiplicaciones, siendo aristas dirigidas. Además, existen los lazos, los cuales corresponden a *flips* cuyo esquema equivalente es él mismo.

El siguiente grafo muestra la componente conexa de la multiplicación de matrices de tamaño $(2, 2, 2)$ sobre $\mathbb{Z}_2$, mostrando los esquemas de multiplicación con un rango máximo de 8, y relaciona el algoritmo estándar con el de Strassen.

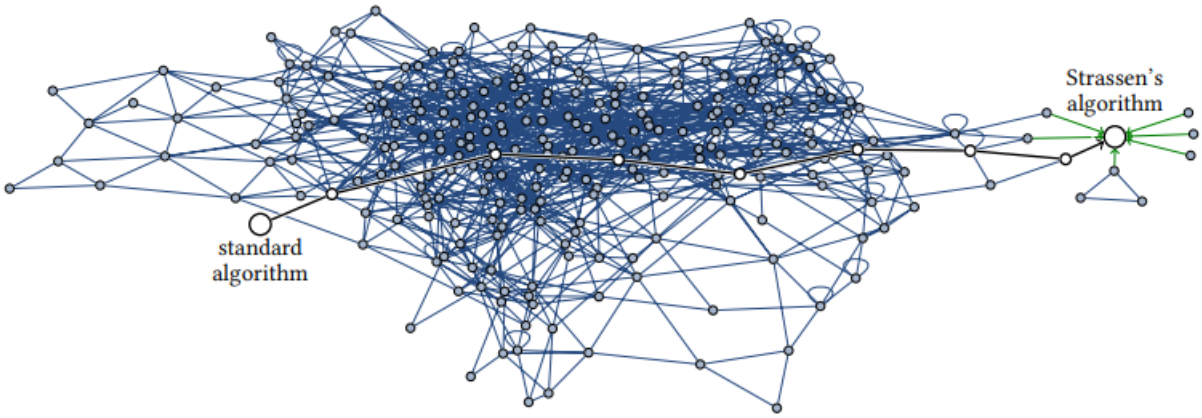


Figura 8: **Grafo de los esquemas de matrices** $(2, 2, 2)$**.** [13]

Observamos que la distancia entre el algoritmo estándar y el de Strassen es 8, indicada en la figura con una ruta marcada. Esta figura forma una componente conectada por 272 vértices y 1183 aristas, de las cuales solo 7 son reducciones y dentro de los *flips* están los lazos que son esquemas equivalentes a si mismos.

En el caso de la multiplicación de matrices 3×3 sobre el cuerpo $\mathbb{Z}_2$, el grafo es tan grande que no se puede determinar la componente completa del algoritmo estándar con un rango máximo de 27. El algoritmo estándar solo tiene un vecino, es decir, en distancia 1 solo conecta con 1 vértice. Mientras que en distancia 2 encontramos 600 vértices, en distancia 3 alrededor de 20000 y en distancia 4 casi 600000. Calcular los vértices en el caso de distancia 5 no es

factible; además, ninguna de las aristas en las distancias $1, 2, 3$ y 4 corresponde a una reducción.

Considerando el grafo formado por los esquemas de multiplicación $(3, 3, 3)$ de rango como máximo 23, los esquemas obtenidos mediante paseos aleatorios partiendo del algoritmo estándar pertenecen a 584 componentes diferentes. Una de estas componentes corresponde al siguiente grafo.

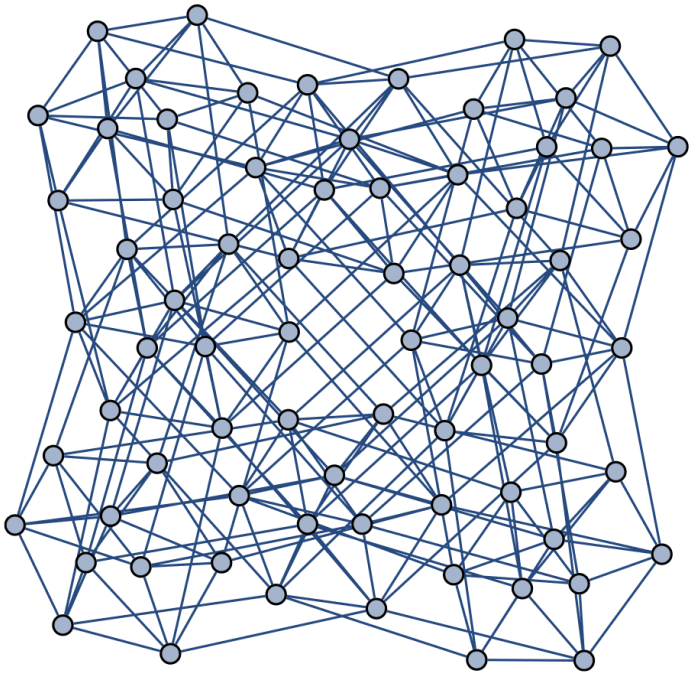


Figura 9: **Componente de los esquemas de matrices** $(3, 3, 3)$ [13]

Observamos que ninguna arista corresponde a una reducción, es decir, no se encuentra ningún esquema con un rango menor que 23, que coincide con el récord establecido por Laderman [6].

4.3. Resultados

Estos métodos han sido utilizados para matrices $n \times m$ y $m \times p$ donde $2 \leq n, m, p \leq 5$. Los resultados obtenidos son los siguientes, siendo K un cuerpo cualquiera.

Tamaño (n,m,p)	Mejor rango en K	Rango descubierto en K	Mejor rango en $\mathbb{Z}_2$	Rango descubierto en $\mathbb{Z}_2$
(2,2,2)	7	7	7	7
(3,3,3)	23	23	23	23
(4,4,4)	49	49	47	47
(5,5,5)	98	97	96	95
(2,2,3)	11	11	11	11
(2,2,4)	14	14	14	14
(2,2,5)	18	18	18	18
(2,3,3)	15	15	15	15
(2,3,4)	20	20	20	20
(2,3,5)	25	25	25	25
(2,4,4)	26	26	26	26
(2,4,5)	33	33	33	33
(2,5,5)	40	40	40	40
(3,3,4)	29	29	29	29
(3,3,5)	36	36	36	36
(3,4,4)	38	38	38	38
(3,4,5)	47	47	47	47
(3,5,5)	58	58	58	58
(4,4,5)	63	62	63	60
(4,5,5)	76	76	76	76

Cuadro 2: Tabla comparativa entre los rangos conocido y los encontrados. [13]

Jakob Moosbauer y Manuel Kauers han continuado trabajando en esta línea de investigación y meses más tarde publicaron unos resultados obtenidos al utilizar esta técnica en multiplicaciones de matrices de tamaños $(n, m, 6)$ donde $2 \leq n, m \leq 6$ [14]. Puede parecer un pequeño avance, pero cabe señalar que un mínimo incremento en el orden de las matrices supone un gran aumento en el tamaño del espacio de búsqueda. De hecho, para algunos tamaños, este método no logró alcanzar el mejor rango conocido, achacan este suceso en parte a las limitaciones del propio método y a los pocos recursos informáticos. Los resultados obtenidos en este caso son los siguientes:

Tamaño (n,m,p)	Rango del algoritmo estándar	Mejor rango conocido	Rango descubierto
(2,2,6)	24	21	21
(2,3,6)	36	30	30
(2,4,6)	48	39	39
(2,5,6)	60	48	48
(2,6,6)	72	57	56
(3,3,6)	54	40	42
(3,4,6)	72	56	56
(3,5,6)	90	70	71
(3,6,6)	108	80	85
(4,4,6)	96	73	74
(4,5,6)	120	93	93
(4,6,6)	144	105	116
(5,5,6)	150	116	116
(5,6,6)	180	137	144
(6,6,6)	216	160	164

Cuadro 3: Tabla comparativa entre los mejores rangos y los encontrados. [14]

A partir de la tabla, vemos destacados siete casos en los que el rango obtenido con esta técnica no mejora ni iguala el que es el mejor rango. Sin embargo, en el caso $(2, 6, 6)$ mejora de 57 a 56 lo cual supone la primera mejora desde los resultado de Hopcroft y Kerr en 1971 [4].

Bibliografía

[1] Shmuel Winograd. «On the number of multiplications required to compute certain functions». En: *Proceedings of the National Academy of Sciences* 58.5 (1967), págs. 1840-1842.

[2] Volker Strassen. «Gaussian elimination is not optimal». En: *Numerische mathematik* 13.4 (1969), págs. 354-356.

[3] Shmuel Winograd. «On the number of multiplications necessary to compute certain functions». En: *Communications on Pure and Applied Mathematics* 23.2 (1970), págs. 165-179.

[4] John E Hopcroft y Leslie R Kerr. «On minimizing the number of multiplications necessary for matrix multiplication». En: *SIAM Journal on Applied Mathematics* 20.1 (1971), págs. 30-36.

[5] Shmuel Winograd. «On multiplication of 2×2 matrices». En: *Linear algebra and its applications* (1971).

[6] J. D. Laderman. «A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications». En: *Bulletin of the american mathematical society* 82 (1976), págs. 126-128.

[7] Alexey V Smirnov. «The bilinear complexity and practical algorithms for matrix multiplication». En: *Computational Mathematics and Mathematical Physics* 53 (2013), págs. 1781-1795.

[8] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Alexandre Sedoglavic y Alexey V Smirnov. «The tensor rank of 5x5 matrices multiplication is bounded by 98 and its border rank by 89». En: *arXiv preprint arXiv:2101.12568* (2021).

[10] Alhussein Fawzi et al. «Discovering faster matrix multiplication algorithms with reinforcement learning». En: *Nature* 610.7930 (2022), págs. 47-53.

[11] Manuel Kauers y Jakob Moosbauer. «A normal form for matrix multiplication schemes». En: *International Conference on Algebraic Informatics*. Springer. 2022, págs. 149-160.

[12] Manuel Kauers y Jakob Moosbauer. «The FBHHRBNRSSSHK-Algorithm for Multiplication in $\mathbb{Z}_2^{5\times5}$ is still not the end of the story». En: *arXiv preprint arXiv:2210.04045* (2022).

[13] Manuel Kauers y Jakob Moosbauer. «Flip graphs for matrix multiplication». En: *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*. 2023, págs. 381-388.

[14] Manuel Kauers y Jakob Moosbauer. «Some New Non-Commutative Matrix Multiplication Algorithms of Size $(n, m, 6)$». En: *arXiv preprint arXiv:2306.00882* (2023).

Apéndice

Escribimos el algoritmo estándar y el algoritmo de Strassen utilizando la definición de algoritmo. Luego, realizamos un ejemplo de cada algoritmo.

A. Algoritmo estándar

Nos restringimos al caso de matrices 2×2. Sea α el algoritmo que calcula la multiplicación de matrices con el método estándar. Entonces el algoritmo explícito es el siguiente:

$$e_\alpha(1) = \alpha(1) = a_{11} \qquad e_\alpha(2) = \alpha(2) = a_{12} \qquad e_\alpha(3) = \alpha(3) = a_{21} \qquad e_\alpha(4) = \alpha(4) = a_{22}$$
$$e_\alpha(5) = \alpha(5) = b_{11} \qquad e_\alpha(6) = \alpha(6) = b_{12} \qquad e_\alpha(7) = \alpha(7) = b_{21} \qquad e_\alpha(8) = \alpha(8) = b_{22}$$
$$e_\alpha(9) = \alpha(9) = (1, 5, p_3) = p_3(e_\alpha(1), e_\alpha(5)) = e_\alpha(1) \cdot e_\alpha(5) = a_{11} \cdot b_{11}$$
$$e_\alpha(10) = \alpha(10) = (2, 7, p_3) = p_3(e_\alpha(2), e_\alpha(7)) = e_\alpha(2) \cdot e_\alpha(7) = a_{12} \cdot b_{21}$$
$$e_\alpha(11) = \alpha(11) = (1, 6, p_3) = p_3(e_\alpha(1), e_\alpha(6)) = e_\alpha(1) \cdot e_\alpha(6) = a_{11} \cdot b_{12}$$
$$e_\alpha(12) = \alpha(12) = (2, 8, p_3) = p_3(e_\alpha(2), e_\alpha(8)) = e_\alpha(2) \cdot e_\alpha(8) = a_{12} \cdot b_{22}$$
$$e_\alpha(13) = \alpha(13) = (3, 5, p_3) = p_3(e_\alpha(3), e_\alpha(5)) = e_\alpha(3) \cdot e_\alpha(5) = a_{21} \cdot b_{11}$$
$$e_\alpha(14) = \alpha(14) = (4, 7, p_3) = p_3(e_\alpha(4), e_\alpha(7)) = e_\alpha(4) \cdot e_\alpha(7) = a_{22} \cdot b_{21}$$
$$e_\alpha(15) = \alpha(15) = (3, 6, p_3) = p_3(e_\alpha(3), e_\alpha(6)) = e_\alpha(3) \cdot e_\alpha(6) = a_{21} \cdot b_{12}$$
$$e_\alpha(16) = \alpha(16) = (4, 8, p_3) = p_3(e_\alpha(4), e_\alpha(8)) = e_\alpha(4) \cdot e_\alpha(8) = a_{22} \cdot b_{22}$$
$$e_\alpha(17) = \alpha(17) = (9, 10, p_1) = p_1(e_\alpha(9), e_\alpha(10)) = e_\alpha(9) + e_\alpha(10) = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}$$
$$e_\alpha(18) = \alpha(18) = (11, 12, p_1) = p_1(e_\alpha(11), e_\alpha(12)) = e_\alpha(11) + e_\alpha(12) = a_{11} \cdot b_{12} + a_{12} \cdot b_{22}$$
$$e_\alpha(19) = \alpha(19) = (13, 14, p_1) = p_1(e_\alpha(13), e_\alpha(14)) = e_\alpha(13) + e_\alpha(14) = a_{21} \cdot b_{11} + a_{22} \cdot b_{21}$$
$$e_\alpha(20) = \alpha(20) = (15, 16, p_1) = p_1(e_\alpha(15), e_\alpha(16)) = e_\alpha(15) + e_\alpha(16) = a_{21} \cdot b_{12} + a_{22} \cdot b_{22}$$

Por tanto, obtenemos:

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} e_\alpha(17) & e_\alpha(18) \\ e_\alpha(19) & e_\alpha(20) \end{pmatrix} = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$

Ejemplo

Consideramos dos matrices 2×2 y calculamos el producto utilizando el algoritmo estándar. Sean $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ y $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, queremos obtener $C = A \cdot B$.

$e_\alpha(1) = 1$	$e_\alpha(2) = 2$	$e_\alpha(3) = 3$	$e_\alpha(4) = 4$
$e_\alpha(5) = 5$	$e_\alpha(6) = 6$	$e_\alpha(7) = 7$	$e_\alpha(8) = 8$
$e_\alpha(9) = 1 \cdot 5 = 5$	$e_\alpha(10) = 2 \cdot 7 = 14$	$e_\alpha(11) = 1 \cdot 6 = 6$	$e_\alpha(12) = 2 \cdot 8 = 16$
$e_\alpha(13) = 3 \cdot 5 = 15$	$e_\alpha(14) = 4 \cdot 7 = 28$	$e_\alpha(15) = 3 \cdot 6 = 18$	$e_\alpha(16) = 4 \cdot 8 = 32$
$e_\alpha(17) = 5 + 14 = 19$	$e_\alpha(18) = 6 + 16 = 22$	$e_\alpha(19) = 15 + 28 = 43$	$e_\alpha(20) = 18 + 32 = 50$

Así obtenemos el resultado:

$$C = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

B. Algoritmo de Strassen

Sea β el algoritmo que calcula la multiplicación de matrices 2×2 con el método de Strassen, entonces los pasos del algoritmo serían los siguientes:

$e_\beta(1) = \beta(1) = a_{11}$ $e_\beta(2) = \beta(2) = a_{12}$ $e_\beta(3) = \beta(3) = a_{21}$ $e_\beta(4) = \beta(4) = a_{22}$

$e_\beta(5) = \beta(5) = b_{11}$ $e_\beta(6) = \beta(6) = b_{12}$ $e_\beta(7) = \beta(7) = b_{21}$ $e_\beta(8) = \beta(8) = b_{22}$

$e_\beta(9) = \beta(9) = (2, 4, p_2) = p_2(e_\beta(2), e_\beta(4)) = e_\beta(2) - e_\beta(4) = a_{12} - a_{22}$

$e_\beta(10) = \beta(10) = (7, 8, p_1) = p_1(e_\beta(7), e_\beta(8)) = e_\beta(7) + e_\beta(8) = b_{21} + b_{22}$

$e_\beta(11) = \beta(11) = (1, 4, p_1) = p_1(e_\beta(1), e_\beta(4)) = e_\beta(1) + e_\beta(4) = a_{11} + a_{22}$

$e_\beta(12) = \beta(12) = (5, 8, p_1) = p_1(e_\beta(5), e_\beta(8)) = e_\beta(5) + e_\beta(8) = b_{11} + b_{22}$

$e_\beta(13) = \beta(13) = (1, 3, p_2) = p_2(e_\beta(1), e_\beta(3)) = e_\beta(1) - e_{beta}(3) = a_{11} - a_{21}$

$e_\beta(14) = \beta(14) = (5, 6, p_1) = p_1(e_\beta(5), e_\beta(6)) = e_\beta(5) + e_\beta(6) = b_{11} + b_{12}$

$e_\beta(15) = \beta(15) = (1, 2, p_1) = p_1(e_\beta(1), e_\beta(2)) = e_\beta(1) + e_\beta(2) = a_{11} + a_{12}$

$e_\beta(16) = \beta(16) = (6, 8, p_2) = p_2(e_\beta(6), e_\beta(8)) = e_\beta(6) - e_\beta(8) = b_{12} - b_{22}$

$e_\beta(17) = \beta(17) = (7, 5, p_2) = p_2(e_\beta(7), e_\beta(5)) = e_\beta(7) - e_\beta(5) = b_{21} - b_{11}$

$e_\beta(18) = \beta(18) = (3, 4, p_1) = p_1(e_\beta(3), e_\beta(4)) = e_\beta(3) + e_\beta(4) = a_{21} + a_{22}$

$e_\beta(19) = \beta(19) = (9, 10, p_3) = p_3(e_\beta(9), e_\beta(10)) = e_\beta(9) \cdot e_\beta(10) = (a_{12} - a_{22}) \cdot (b_{21} + b_{22})$

$e_\beta(20) = \beta(20) = (11, 12, p_3) = p_3(e_\beta(11), e_\beta(12)) = e_\beta(11) \cdot e_\beta(12) = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$

$e_\beta(21) = \beta(21) = (13, 14, p_3) = p_3(e_\beta(13), e_\beta(14)) = e_\beta(13) \cdot e_\beta(14) = (a_{11} - a_{21}) \cdot (b_{11} + b_{12})$

$e_\beta(22) = \beta = (15, 8, p_3) = p_3(e_\beta(15), e_\beta(8)) = e_\beta(15) \cdot e_\beta(8) = (a_{11} + a_{12}) \cdot b_{22}$

$e_\beta(23) = \beta(23) = (1, 16, p_3) = p_3(e_\beta(1), e_\beta(16)) = e_\beta(1) \cdot e_\beta(16) = a_{11} \cdot (b_{12} - b_{22})$

$e_\beta(24) = \beta(24) = (4, 17, p_3) = p_3(e_\beta(4), e_\beta(17)) = e_\beta(4) \cdot e_\beta(17) = a_{22} \cdot (b_{21} - b_{11})$

$e_\beta(25) = \beta(25) = (18, 5, p_3) = p_3(e_\beta(18), e_\beta(5)) = e_\beta(18) \cdot e_\beta(5) = (a_{21} + a_{22}) \cdot b_{11}$

$e_\beta(26) = \beta(26) = (19, 20, p_1) = p_1(e_\beta(19), e_\beta(20)) = e_\beta(19) + e_\beta(20)$

$e_\beta(27) = \beta(27) = (26, 22, p_2) = p_2(e_\beta(26), e_\beta(22)) = e_\beta(26) - e_\beta(22)$

$e_\beta(28) = \beta(28) = (27, 24, p_1) = p_1(e_\beta(27), e_\beta(24)) = e_\beta(27) + e_\beta(24)$

$e_\beta(29) = \beta(29) = (22, 23, p_1) = p_1(e_\beta(22), e_\beta(23)) = e_\beta(22) + e_\beta(23)$

$e_\beta(30) = \beta(30) = (24, 25, p_1) = p_1(e_\beta(24), e_\beta(25)) = e_\beta(24) + e_\beta(25)$

$e_\beta(31) = \beta(31) = (20, 21, p_2) = p_2(e_\beta(20), e_\beta(21)) = e_\beta(20) - e_\beta(21)$

$e_\beta(32) = \beta(32) = (31, 23, p_1) = p_1(e_\beta(31), e_\beta(23)) = e_\beta(31) + e_\beta(23)$

$e_\beta(33) = \beta(33) = (32, 25, p_2) = p_2(e_\beta(32), e_\beta(25)) = e_\beta(32) - e_\beta(25)$

El resultado de la multiplicación es:

$$C = \begin{pmatrix} e_\beta(28) & e_\beta(29) \\ e_\beta(30) & e_\beta(35) \end{pmatrix}$$

Ejemplo

Consideramos las dos mismas matrices 2×2 del ejemplo anterior y calculamos el producto usando el método de Strassen.

Sean $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ y $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, queremos obtener $C = A \cdot B$.

$e_\beta(1) = 1$	$e_\beta(2) = 2$	$e_\beta(3) = 3$
$e_\beta(4) = 4$	$e_\beta(5) = 5$	$e_\beta(6) = 6$
$e_\beta(7) = 7$	$e_\beta(8) = 8$	$e_\beta(9) = 2 - 4 = -2$
$e_\beta(10) = 7 + 8 = 15$	$e_\beta(11) = 1 + 4 = 5$	$e_\beta(12) = 5 + 8 = 13$
$e_\beta(13) = 1 - 3 = -2$	$e_\beta(14) = 5 + 6 = 11$	$e_\beta(15) = 1 + 2 = 3$
$e_\beta(16) = 6 - 8 = -2$	$e_\beta(17) = 7 - 5 = 2$	$e_\beta(18) = 3 + 4 = 7$
$e_\beta(19) = -2 \cdot 15 = -30$	$e_\beta(20) = 5 \cdot 13 = 65$	$e_\beta(21) = -2 \cdot 11 = -22$
$e_\beta(22) = 3 \cdot 8 = 24$	$e_\beta(23) = 1 \cdot (-2) = -2$	$e_\beta(24) = 4 \cdot 2 = 8$
$e_\beta(25) = 7 \cdot 5 = 35$	$e_\beta(26) = -30 + 65 = 35$	$e_\beta(27) = 35 - 24 = 11$
$e_\beta(28) = 11 + 8 = 19$	$e_\beta(29) = 24 + (-2) = 22$	$e_\beta(30) = 8 + 35 = 43$
$e_\beta(31) = 65 - (-22) = 87$	$e_\beta(32) = 87 + (-2) = 85$	$e_\beta(33) = 85 - 35 = 50$

Así obtenemos el resultado:

$$C = \begin{pmatrix} e_\beta(28) & e_\beta(29) \\ e_\beta(30) & e_\beta(35) \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$