Linear Algebra Notes

Abror Maksudov

Created: November 8, 2025

Last Updated: 2025-11-11 13:58:19+05:00

Contents

1 Vectors

1.1 Definition of a Scalar

A scalar is a **single number**, denoted as $a, b, c, \alpha, \beta, \gamma, \lambda, \mu$.

Space Complexity: $O(1)$ — *Time Complexity*: N/A.

1.2 Definition of a Vector

A vector is an **ordered list** of numbers, usually denoted as $\mathbf{v}$ or $\vec{v}$.

- **Column Vector (Default/Standard):**

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}^T$$

- **Row Vector:**

$$\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}^T$$

Space Complexity: $O(n)$ — *Time Complexity*: N/A.

1.3 Vector Operations

For vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$ and scalar $c \in \mathbb{R}$, the following rules hold:

1. **Scalar Multiplication:**

$$c\vec{v} = c \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}$$

2. **Vector Addition:**

$$\vec{u} + \vec{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

   Requirement: Both vectors must have the same dimension.

3. **Vector Subtraction:**

$$\vec{u} - \vec{v} = \vec{u} + (-1)\vec{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ \vdots \\ u_n - v_n \end{bmatrix}$$

   Requirement: Both vectors must have the same dimension.

Space Complexity: $O(n)$ — *Time Complexity*: $O(n)$.

1.4 Properties of Vector

Let $\vec{u}, \vec{v}, \vec{w} \in \mathbb{R}^n$ and $c, d \in \mathbb{R}$. Then:

(1)	$\vec{u} + \vec{v} = \vec{v} + \vec{u}$	(Commutativity)
(2)	$(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$	(Associativity)
(3)	$\vec{v} + \vec{0} = \vec{v}$	(Identity)
(4)	$\vec{v} + (-\vec{v}) = \vec{0}$	(Inverse)
(5)	$c(d\vec{v}) = (cd)\vec{v}$	(Associativity of scalars)
(6)	$1 \cdot \vec{v} = \vec{v}$	(Multiplicative identity)
(7)	$c(\vec{u} + \vec{v}) = c\vec{u} + c\vec{v}$	(Distributivity over vectors)
(8)	$(c + d)\vec{v} = c\vec{v} + d\vec{v}$	(Distributivity over scalars)

These 8 axioms define a linear space (or vector space) in $\mathbb{R}^n$.

1.5 Linear Combinations

A vector $\vec{w} \in \mathbb{R}^n$ is a **linear combination** of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_k \in \mathbb{R}^n$ if there exist scalars $c_1, c_2, \ldots, c_k \in \mathbb{R}$ such that:

$$\vec{w} = c_1\vec{v}_1 + c_2\vec{v}_2 + \cdots + c_k\vec{v}_k = \sum_{i=1}^{k} c_i\vec{v}_i = \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_k \\ | & | & & | \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \vec{w}$$

The scalars $c_1, c_2, \ldots, c_k$ are called **coefficients** or **weights** of the linear combination.

Space Complexity: $O(n)$ — *Time Complexity*: $O(nk)$.

1.6 Dot Product

The **dot product** of two vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$ is a scalar defined as:

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^{n} u_i v_i$$

Geometric View:

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos\theta$$

where θ is the angle between $\vec{u}$ and $\vec{v}$, and $\|\vec{u}\|$ is the **magnitude** (or **length** or **norm**) of $\vec{u}$.

Alternative notations:

$$\vec{u} \cdot \vec{v} = \langle \vec{u}, \vec{v} \rangle = \mathbf{u}^T \mathbf{v}$$

Special Cases:

(1) $\vec{u} \cdot \vec{v} = 0 \iff \theta = 90^\circ$ (Perpendicular)

(2) $\vec{u} \cdot \vec{v} > 0 \iff \theta < 90^\circ$ (Acute angle)

(3) $\vec{u} \cdot \vec{v} < 0 \iff \theta > 90^\circ$ (Obtuse angle)

(4) $\vec{u} \cdot \vec{u} = \|\vec{u}\|^2 \iff \theta = 0^\circ$ (Squared magnitude)

Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$.
Note: Highly optimized using SIMD (Single Instruction, Multiple Data) operations.

1.7 Properties of the Dot Product

Let $\vec{u}, \vec{v}, \vec{w} \in \mathbb{R}^n$ and $c \in \mathbb{R}$. Then:

(1) $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$ (Commutativity)

(2) $\vec{u} \cdot (\vec{v} + \vec{w}) = \vec{u} \cdot \vec{v} + \vec{u} \cdot \vec{w}$ (Distributivity)

(3) $(c\vec{u}) \cdot \vec{v} = c(\vec{u} \cdot \vec{v}) = \vec{u} \cdot (c\vec{v})$ (Scalar multiplication)

(4) $\vec{u} \cdot \vec{u} \geq 0$ (Positive definiteness)

(5) $\vec{u} \cdot \vec{u} = 0 \iff \vec{u} = \vec{0}$ (Definiteness)

(6) $\vec{0} \cdot \vec{v} = 0$ (Zero vector)

These properties define the dot product as an **inner product** on $\mathbb{R}^n$.

Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$.

1.8 Cross Product

The **cross product** of two vectors $\vec{u}, \vec{v} \in \mathbb{R}^3$ is a vector $\vec{u} \times \vec{v} \in \mathbb{R}^3$ that is **perpendicular** to both $\vec{u}$ and $\vec{v}$ as is defined as:

$$\vec{u} \times \vec{v} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

Note: Cross product is only defined in $\mathbb{R}^3$ (and $\mathbb{R}^7$, but rarely used).
Space Complexity: $O(1)$ — *Time Complexity*: $O(1)$

1.9 Properties of the Cross Product

Let $\vec{u}, \vec{v}, \vec{w} \in \mathbb{R}^3$, $c \in \mathbb{R}$ and θ be the angle between $\vec{u}$ and $\vec{v}$. Then:

(1)	$\vec{u} \times \vec{v} = -(\vec{v} \times \vec{u})$	(Anti-commutativity)
(2)	$\vec{u} \times (\vec{v} + \vec{w}) = \vec{u} \times \vec{v} + \vec{u} \times \vec{w}$	(Distributivity)
(3)	$(c\vec{u}) \times \vec{v} = c(\vec{u} \times \vec{v}) = \vec{u} \times (c\vec{v})$	(Scalar multiplication)
(4)	$\vec{u} \times \vec{u} = \vec{0}$	(Self cross product)
(5)	$\vec{u} \times \vec{0} = \vec{0} \times \vec{u} = \vec{0}$	(Zero vector)
(6)	$\vec{u} \times \vec{v} = \vec{0} \iff \vec{u} \parallel \vec{v}$	(Parallel vectors)
(7)	$\vec{u} \cdot (\vec{u} \times \vec{v}) = 0, \quad \vec{v} \cdot (\vec{u} \times \vec{v}) = 0$	(Orthogonality)
(8)	$\lVert\vec{u} \times \vec{v}\rVert = \lVert\vec{u}\rVert\lVert\vec{v}\rVert \sin\theta$	(Magnitude)

Warning: The cross product is **not associative** in general: $\vec{u} \times (\vec{v} \times \vec{w}) \neq (\vec{u} \times \vec{v}) \times \vec{w}$.

Space Complexity: $O(1)$ — *Time Complexity*: $O(1)$

1.10 Scalar Projection

Let $\vec{u}, \vec{v} \in \mathbb{R}^n$. Then, the **scalar (component) projection** of $\vec{u}$ onto $\vec{v}$ is the signed length of the projection:

$$\text{comp}_{\vec{v}}\vec{u} = \frac{\vec{u} \cdot \vec{v}}{\lVert\vec{v}\rVert}$$

Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$.

1.11 Vector Projection

Let $\vec{u}, \vec{v} \in \mathbb{R}^n$. Then, the **vector projection** of $\vec{u}$ onto $\vec{v}$ is a vector in the direction of $\vec{v}$:

$$\text{proj}_{\vec{v}}\vec{u} = \frac{\vec{u}\cdot\vec{v}}{\|\vec{v}\|^2}\vec{v} = \frac{\vec{u}\cdot\vec{v}}{\vec{v}\cdot\vec{v}}\vec{v} = (\text{comp}_{\vec{v}}\vec{u})\,\frac{\vec{v}}{\|\vec{v}\|}$$

Orthogonal Component:
The component of $\vec{u}$ orthogonal to $\vec{v}$ is:

$$\vec{u}_\perp = \vec{u} - \text{proj}_{\vec{v}}\vec{u}$$

Note that $\vec{u} = \text{proj}_{\vec{v}}\vec{u} + \vec{u}_\perp$ (orthogonal decomposition).

Space Complexity: $O(n)$ — *Time Complexity*: $O(n)$.

1.12 Angle Between Vectors

The **angle** θ between two non-zero vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$ is given by:

$$\cos\theta = \frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|} \iff \theta = \arccos\left(\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|}\right), \quad 0 \le \theta \le \pi$$

Special Cases:

(1) $\theta = 0 \iff \vec{u}$ and $\vec{v}$ point in the same direction $\quad (\cos\theta = 1)$

(2) $\theta = \dfrac{\pi}{2} \iff \vec{u} \perp \vec{v}$ $\quad (\cos\theta = 0)$

(3) $\theta = \pi \iff \vec{u}$ and $\vec{v}$ point in opposite directions $\quad (\cos\theta = -1)$

Note: The formula $\vec{u}\cdot\vec{v} = \|\vec{u}\|\|\vec{v}\|\cos\theta$ is the geometric definition of the dot product.
Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$

1.13 Cauchy-Schwarz Inequality

Let $\vec{u}, \vec{v} \in \mathbb{R}^n$. Then:

$$|\langle\vec{u},\vec{v}\rangle|^2 \le \langle\vec{u},\vec{u}\rangle \cdot \langle\vec{v},\vec{v}\rangle,$$

or,

$$|\langle\vec{u},\vec{v}\rangle|^2 \le \|\vec{u}\|^2\,\|\vec{v}\|^2 \iff |\langle\vec{u},\vec{v}\rangle| = \|\vec{u}\|\,\|\vec{v}\|\,|\cos\theta| \le \|\vec{u}\|\,\|\vec{v}\|\,,$$

or in component form,

$$\left(\sum_{i=1}^{n} u_i v_i\right)^2 \le \left(\sum_{i=1}^{n} u_i^2\right)\left(\sum_{i=1}^{n} v_i^2\right).$$

Equality holds if and only if $\vec{u}$ and $\vec{v}$ are linearly dependent.

Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$

1.14 Triangle Inequality

Let $\vec{u}, \vec{v} \in \mathbb{R}^n$. Then:

$$\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$$

Equality holds if and only if $\vec{u} = c\vec{v}$ with $c \geq 0$.
Reverse Triangle Inequality:

$$\big|\|\vec{u}\| - \|\vec{v}\|\big| \leq \|\vec{u} - \vec{v}\|$$

Space Complexity: $O(1)$ — *Time Complexity*: $O(n)$

1.15 Vectorization for Computational Efficiency

Vectorization is the process of replacing explicit loops with vector/matrix operations to achieve dramatic performance improvements by leveraging optimized, low-level linear algebra libraries and hardware parallelism.

Performance Example:

Operation	Complexity	Loop (Python)	Vectorized (NumPy)
Dot Product ($n = 10^6$)	$O(n)$	$\sim$ 500 ms	$\sim$ 1 ms
Matrix-Vector ($n \times n$)	$O(n^2)$	$\sim$ 2000 ms	$\sim$ 10 ms
Matrix-Matrix ($n \times n$)	$O(n^3)$	$\sim$ hours	$\sim$ seconds

Why Vectorization is Faster:

1. **Hardware (SIMD):** Modern CPUs use **S**ingle **I**nstruction, **M**ultiple **D**ata (SIMD) instructions. This allows a single CPU instruction to perform the same operation (e.g., multiplication) on multiple data elements (e.g., 8 or 16 numbers) at the same time. Vectorized code uses these instructions automatically; Python loops do not.

2. **Cache Efficiency:** Vectorized operations access memory in large, contiguous blocks. This is very "cache-friendly" and avoids the high cost of fetching data from main memory (RAM) for each small step of a loop.

3. **Optimized Libraries:** NumPy, PyTorch, and TensorFlow operations are thin wrappers around highly optimized libraries (like BLAS/LAPACK, MKL, cuBLAS) written in C or Fortran. These libraries have been fine-tuned for decades.

Space-Time Trade-off:

- Loops: $O(1)$ extra space, slow
- Vectorized: May need $O(n)$ temporary arrays, fast
- *Rule:* Memory is cheap, time is expensive

2 Matrices

2.1 Definition of a Matrix

A matrix is a **rectangular array** of numbers arranged in rows and columns, denoted as A, B, C or $\mathbf{A}, \mathbf{B}, \mathbf{C}$.

An $m \times n$ matrix A has m rows and n columns:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = [a_{ij}]_{m \times n}$$

where a_{ij} is the element in the i-th row and j-th column also denoted as A_{ij}.

Notation:

- $A \in \mathbb{R}^{m \times n}$ means A is an $m \times n$ matrix with real entries
- $(A)_{ij} = a_{ij}$ denotes the (i, j)-entry of matrix A
- $m \times n$ is the **dimension** or **size** of the matrix

Special Cases:

(1) $m = n \implies A$ is a **square matrix** $(n \times n)$

(2) $m = 1 \implies A$ is a **row vector** $(1 \times n)$

(3) $n = 1 \implies A$ is a **column vector** $(m \times 1)$

(4) $m = n = 1 \implies A$ is a **scalar** (1×1)

Interpretations: Unlike vectors, matrices have multiple common interpretations:

1. **As Data:** A way to store data in a grid.
2. **As a Collection of Vectors:**

$$\begin{bmatrix} | & | & & | \\ \vec{c}_1 & \vec{c}_2 & \cdots & \vec{c}_n \\ | & | & & | \end{bmatrix} \qquad \begin{bmatrix} - & \vec{r}_1 & - \\ - & \vec{r}_2 & - \\ & \vdots & \\ - & \vec{r}_m & - \end{bmatrix}$$

Set of column vectors — Set of row vectors

3. **As a Linear Transformation:** A function that "transforms" a vector $\vec{x}$ into a new vector $\vec{y}$ via matrix-vector multiplication ($\vec{y} = A\vec{x}$). The matrix can scale, rotate, or shear space.

Storage: Matrices are typically stored in row-major or column-major order in memory.
Space Complexity: $O(mn)$ — *Time Complexity*: $O(1)$

2.2 Types of Matrices

1. **Square Matrix:** $A \in \mathbb{R}^{n \times n}$ (number of rows = number of columns)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}_{3 \times 3}$$

2. **Zero Matrix:** All entries are zero, denoted O or $0_{m \times n}$

$$O = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{2 \times 3}$$

3. **Identity Matrix:** Square matrix with ones on the main diagonal and zeros elsewhere, denoted I or I_n

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Formally: $(I_n)_{ij} = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$ (Kronecker delta function)

4. **Diagonal Matrix:** Square matrix with non-zero entries only on the main diagonal

$$D = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} = \text{diag}(d_1, d_2, d_3)$$

5. **Upper Triangular Matrix:** All entries below the main diagonal are zero

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Formally: $u_{ij} = 0$ for $i > j$

6. **Lower Triangular Matrix:** All entries above the main diagonal are zero

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

Formally: $l_{ij} = 0$ for $i < j$

7. **Symmetric Matrix:** Square matrix where $A = A^T$ (equal to its transpose)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

Formally: $a_{ij} = a_{ji}$ for all i, j

8. **Skew-Symmetric Matrix:** Square matrix where $A = -A^T$

$$A = \begin{bmatrix} 0 & 2 & -3 \\ -2 & 0 & 5 \\ 3 & -5 & 0 \end{bmatrix}$$

Formally: $a_{ij} = -a_{ji}$ for all i, j (diagonal entries must be zero)

9. **Orthogonal Matrix:** Square matrix where $A^T A = AA^T = I$ (columns and rows are orthonormal)

10. **Row Echelon Form (REF):** Matrix where:

- All nonzero rows are above any rows of all zeros
- Leading entry (pivot) of each nonzero row is to the right of the leading entry of the row above it

11. **Reduced Row Echelon Form (RREF):** REF where:

- Each leading entry is 1
- Each leading 1 is the only nonzero entry in its column

Note: Special matrix types often have computational advantages (e.g., diagonal matrix multiplication is $O(n)$ instead of $O(n^3)$).

2.3 Matrix Operations

For matrices $A, B \in \mathbb{R}^{m \times n}$, $\vec{x} \in \mathbb{R}^n$ and scalar $c \in \mathbb{R}$, the following rules hold::

1. **Scalar Multiplication:**

$$(cA)_{ij} = c(a_{ij})$$

Space Complexity: $O(mn)$ — *Time Complexity*: $O(mn)$.

2. **Matrix Addition:**

$$(A + B)_{ij} = a_{ij} + b_{ij}$$

Requirement: Both matrices must have the same dimension.
Space Complexity: $O(mn)$ — *Time Complexity*: $O(mn)$.

3. **Matrix Subtraction:**

$$(A - B)_{ij} = a_{ij} - b_{ij}$$

Requirement: Both matrices must have the same dimension.
Space Complexity: $O(mn)$ — *Time Complexity*: $O(mn)$.

4. **Matrix Transpose:**

$$(A^T)_{ij} = a_{ji}$$

Space Complexity: $O(mn)$ — *Time Complexity*: $O(mn)$.

5. **Matrix-Vector Multiplication:**

$$\vec{y} = A\vec{x} \quad (A \in \mathbb{R}^{m \times n} : \vec{x} \in \mathbb{R}^n \mapsto \vec{y} \in \mathbb{R}^m)$$

Space Complexity: $O(m)$ — *Time Complexity*: $O(mn)$.

6. **Matrix-Matrix Multiplication:**

$$C = AB \quad (A \in \mathbb{R}^{m\times k}, B \in \mathbb{R}^{k\times n} \implies C \in \mathbb{R}^{m\times n})$$

Space Complexity: $O(mn)$ — *Time Complexity*: $O(mnk)$.
Note: For two $n \times n$ matrices, the time complexity is $O(n^3)$.

2.4 Matrix-Vector Multiplication

The product of a matrix $A \in \mathbb{R}^{m\times n}$ and a column vector $\vec{x} \in \mathbb{R}^{n\times 1}$ results in a column vector $\vec{y} \in \mathbb{R}^{m\times 1}$ and is defined as:

$$A\vec{x} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix} = \vec{y}.$$

Requirement: The number of columns in A must equal the dimension of $\vec{x}$.
This operation can be interpreted in two essential ways:

1. **The "Row Picture" (Dot Product Method):**
   Each entry of the output vector $\vec{y}$ is the **dot product** of the corresponding **row of** A with the vector $\vec{x}$.

$$\begin{bmatrix} - & \vec{r}_1 & - \\ - & \vec{r}_2 & - \\ & \vdots & \\ - & \vec{r}_m & - \end{bmatrix} \begin{bmatrix} | \\ \vec{x} \\ | \end{bmatrix} = \begin{bmatrix} \vec{r}_1 \cdot \vec{x} \\ \vec{r}_2 \cdot \vec{x} \\ \vdots \\ \vec{r}_m \cdot \vec{x} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

   This is the standard method used for manual computation.

2. **The "Column Picture" (Linear Combination Method):**
   The output vector $\vec{y}$ is a **linear combination** of the **columns of** A, where the **entries of** $\vec{x}$ **are the weights**.

$$\begin{bmatrix} | & | & & | \\ \vec{c}_1 & \vec{c}_2 & \cdots & \vec{c}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = x_1 \begin{bmatrix} | \\ \vec{c}_1 \\ | \end{bmatrix} + x_2 \begin{bmatrix} | \\ \vec{c}_2 \\ | \end{bmatrix} + \cdots + x_n \begin{bmatrix} | \\ \vec{c}_n \\ | \end{bmatrix}$$

   This interpretation is more conceptual and is fundamental to understanding linear systems, column space, and rank.

Space Complexity: $O(m)$ — *Time Complexity*: $O(mn)$

2.5 Matrix-Matrix Multiplication

For matrices $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$, the product $C = AB \in \mathbb{R}^{m\times n}$ is defined as:

$$C = AB = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{kn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

where each entry is computed as:

$$c_{ij} = \sum_{r=1}^{k} a_{ir} b_{rj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}$$

Requirement: The number of columns in A must equal the number of rows in B.
This operation can be interpreted in multiple ways:

1. **The "Entry-wise" View (Dot Product Method):**
   Each entry c_{ij} of the product matrix C is the **dot product** of the i-th **row of** A with the j-th **column of** B.

   $$c_{ij} = (\text{row } i \text{ of } A) \cdot (\text{column } j \text{ of } B) = \sum_{r=1}^{k} a_{ir} b_{rj}$$

   $$\begin{bmatrix} \cdots & \vec{r}_i & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ \vec{c}_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \\ \cdots & C_{ij} & \cdots \\ & \vdots & \ddots \end{bmatrix}$$

   This is the standard computational method.

2. **The "Column Picture":**
   Each **column of** AB is a **linear combination of the columns of** A, with weights taken from the corresponding **column of** B.

   $$\text{col}_j(C) = A \cdot (\text{col}_j(B))$$

   $$AB = A \begin{bmatrix} | & | & & | \\ \vec{b}_1 & \vec{b}_2 & \cdots & \vec{b}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ A\vec{b}_1 & A\vec{b}_2 & \cdots & A\vec{b}_n \\ | & | & & | \end{bmatrix}$$

   This view shows that every column of AB must be in the column space of A.

3. **The "Row Picture":**
   Symmetrically, each **row of** AB is a **linear combination of the rows of** B, with weights taken from the corresponding **row of** A.

   $$\text{row}_i(C) = (\text{row}_i(A)) \cdot B$$

   $$AB = \begin{bmatrix} - & \vec{a}_1 & - \\ - & \vec{a}_2 & - \\ & \vdots & \\ - & \vec{a}_m & - \end{bmatrix} B = \begin{bmatrix} - & \vec{a}_1 B & - \\ - & \vec{a}_2 B & - \\ & \vdots & \\ - & \vec{a}_m B & - \end{bmatrix}$$

   This view shows that every row of C must be in the row space of B.

Identity Property:

$$AI_n = A, \quad I_m A = A$$

where I is the identity matrix of appropriate size.

Note: Optimized algorithms like Strassen's can achieve $O(n^{2.807})$ for square matrices. Modern libraries use cache-optimized blocking and GPU parallelization for practical speedups.
Space Complexity: $O(mn)$ — *Time Complexity*: $O(mnk)$ (naive algorithm), $O(n^3)$ (for $n \times n$ matrices).