# ভেক্টর ও ভেক্টর স্পেস কী ছিল?

প্রথম দিনে আমরা শিখেছি ভেক্টর এবং ভেক্টর স্পেস।

- ভেক্টর হলো এমন একটি বস্তু যার মান (magnitude) ও দিক (direction) থাকে।
- উদাহরণ:
    - 2D তে: (3, 4)
    - 3D তে: (1, -2, 5)

একটি ভেক্টর স্পেস হলো এমন একটি জায়গা যেখানে:

- ভেক্টর যোগ করা যায়
- স্কেলার দিয়ে গুণ করা যায়
- কিছু নির্দিষ্ট নিয়ম মানা হয়

সহজভাবে বললে, এটা একটা গাণিতিক জগৎ যেখানে ভেক্টরগুলো থাকে।

# Linear Transformation কী?

এখন বলা হচ্ছে:

> A Linear Transformation is a function that maps vectors from one space to another.

মানে,

একটি **Linear Transformation** হলো এমন একটি ফাংশন যা:

- একটি ভেক্টর স্পেস থেকে
- আরেকটি ভেক্টর স্পেসে
- ভেক্টরকে ম্যাপ (রূপান্তর) করে।

### সহজ ভাষায়:

ধরো তুমি একটা ভেক্টর (2,1) নিলে।
এখন একটা নিয়ম প্রয়োগ করলে সেটা হয়ে গেল (4,2)।

এই "নিয়ম"টাই হলো **Linear Transformation**।

# কেন একে “Linear” বলা হয়?

কারণ এটা দুইটা গুরুত্বপূর্ণ নিয়ম মানে:

1⃣ যোগের ক্ষেত্রে:

$T(v + w) = T(v) + T(w)$

2️⃣ স্কেলার গুণের ক্ষেত্রে:

$T(c\ v) = c\ T(v)$

মানে, transformation করলে গাণিতিক সম্পর্ক নষ্ট হয় না।

# **Matrix** কীভাবে কাজ করে?

কনটেন্টে বলা হয়েছে:

> A Matrix is the numerical engine that performs this transformation.

মানে,
**Matrix** হলো সেই সংখ্যার মেশিন যা **Linear Transformation** সম্পাদন করে।

### উদাহরণ:

ধরো ম্যাট্রিক্স:

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

এটা যদি ভেক্টর (x, y) কে গুণ করি:

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

মানে ভেক্টর দ্বিগুণ হয়ে গেল।

এখানে ম্যাট্রিক্সটা space-কে "stretch" করছে।

# **Matrix** আসলে **space-**কে কী করে?

Matrix space-কে:

- Stretch করে (বড় করে)
- Compress করে (ছোট করে)
- Rotate করে (ঘুরিয়ে দেয়)
- Shear করে (বাঁকা করে দেয়)

- Reflect করে (উল্টে দেয়)

মানে পুরো জ্যামিতিক জায়গাটাকে পরিবর্তন করে দেয়।

## **Deep Neural Networks** এর সাথে সম্পর্ক কী?

এখানে সবচেয়ে গুরুত্বপূর্ণ অংশ:

> Understanding how matrices morph space is the fundamental prerequisite for understanding how Deep Neural Networks manipulate data.

Deep Neural Network (DNN) এ যা হয়:

- Input data → ভেক্টর আকারে থাকে
- প্রতিটি layer এ → Matrix multiplication হয়
- তারপর activation function প্রয়োগ হয়

মানে পুরো Neural Network আসলে:

Matrix × Vector + Activation

প্রতিটি layer একটা Linear Transformation করছে।

# 1 Linear Transformations

## 1.1 Functions in Higher Dimensions

### সাধারণ ক্যালকুলাসে কী হয়?

আমরা জানি:

f(x)

এখানে,

- ইনপুট = একটি সংখ্যা
- আউটপুট = আরেকটি সংখ্যা

উদাহরণ:

$f(x) = 2x$

যদি $x = 3$ হয় → আউটপুট = 6

এটা 1D → 1D ম্যাপিং।

## কিন্তু **Linear Algebra** তে কী হয়?

এখানে ইনপুট একটা সংখ্যা না, বরং একটা ভেক্টর।

$T(x)$

এখানে:

- ইনপুট = একটি ভেক্টর
- আউটপুট = আরেকটি ভেক্টর

## $R^n \rightarrow R^m$ মানে কী?

$T : R^n \rightarrow R^m$

এটা খুব গুরুত্বপূর্ণ।

- $R^n$ = n-ডাইমেনশনের ভেক্টর স্পেস
- $R^m$ = m-ডাইমেনশনের ভেক্টর স্পেস

### উদাহরণ:

- $R^2 \rightarrow R^2$
  (2D ভেক্টর থেকে 2D ভেক্টর)
- $R^3 \rightarrow R^2$
  (3D ভেক্টর থেকে 2D ভেক্টর)
- $R^{10} \rightarrow R^3$
  (10 feature থেকে 3 feature)

### সহজ উদাহরণ

ধরো:

$T(x,y) = (2x, 3y)$

এখানে:

- ইনপুট: (x,y)
- আউটপুট: (2x,3y)

মানে space কে stretch করা হচ্ছে।

# 1.2 The Rules of Linearity

সব transformation linear না।

Linear হতে হলে ২টা কঠোর নিয়ম মানতে হবে।

## Rule 1: Additivity

$T(u + v) = T(u) + T(v)$

মানে:

# প্রথমে যোগ করে **transform** করা

আলাদা আলাদা transform করে পরে যোগ করা

উদাহরণ**:**

ধরো:

$T(x,y) = (2x, 2y)$

$u = (1,1)$
$v = (2,0)$

তাহলে:

$u + v = (3,1)$

$T(u+v) = (6,2)$

এখন আলাদা করলে:

T(u) = (2,2)
T(v) = (4,0)

যোগ করলে:
(6,2)

✔ একই ফল → তাই additivity ঠিক আছে

## Rule 2: Homogeneity

T(cu) = cT(u)

মানে স্কেলার বাইরে আসতে পারবে।

উদাহরণ:

u = (1,2)
c = 3

cu = (3,6)

T(3,6) = (6,12)

অন্যদিকে:

T(1,2) = (2,4)
3T(1,2) = (6,12)

✔ একই ফল → homogeneity ঠিক আছে

# জ্যামিতিক অর্থ

এখন আসল গভীর বিষয়

## ১ **Origin (0)** স্থির থাকতে হবে

T(0) = 0

মানে (0,0) পয়েন্ট নড়তে পারবে না।

যদি origin সরে যায় → transformation linear না।

কারণ:

Linear transformation শুধু stretch, rotate, shear করতে পারে
কিন্তু shift করতে পারে না।

### উদাহরণ **(Linear না)**

T(x,y) = (x+1, y)

এখানে:

T(0,0) = (1,0)

Origin সরে গেছে ❌
তাই এটা linear না।

## ২ **Grid lines parallel** থাকতে হবে

এটা খুব সুন্দর জ্যামিতিক ধারণা।

ধরো 2D grid:

| | | |
- - - -
| | | |

Linear transformation করলে:

- লাইনগুলো বাঁকা হতে পারে
- stretch হতে পারে
- rotate হতে পারে

কিন্তু:

✔ সমান্তরাল থাকবে
✔ সমান দূরত্ব বজায় রাখবে

যদি grid বেঁকে অসমান হয়ে যায় → linear না।

# গভীরভাবে বোঝো

Linear transformation পারে:

- Stretch
- Compress
- Rotate
- Reflect
- Shear

কিন্তু পারে না:

- Shift (translate)
- Curve বানাতে

# **Deep Learning** এর সাথে সম্পর্ক

Neural Network এ প্রতিটি layer:

T(x) = Wx

এখানে W হলো matrix।

প্রতিটি layer একটা Linear Transformation।

তারপর activation function যোগ হয়

# 2 Matrices as Functions

## 2.1 The Transformation Matrix

মূল কথা:

প্রতিটি **Linear Transformation** কে একটি **Matrix** দিয়ে পুরোপুরি প্রকাশ করা যায়।

মানে —

> যদি transformation linear হয়, তাহলে সেটা সবসময় একটা matrix multiplication দিয়ে করা সম্ভব।

# **Basis Vector** দিয়ে পুরো **Space** বোঝা

2D space–এ দুইটা বিশেষ ভেক্টর আছে:

$e_1 = (1,0)$
$e_2 = (0,1)$

এগুলাকে বলে **basis vectors**।

এরা হলো পুরো space–এর building blocks।

যেকোনো ভেক্টর লেখা যায় এভাবে:

$x = x_1e_1 + x_2e_2$

মানে,

$(x_1, x_2)$

আসলে হলো:

$x_1(1,0) + x_2(0,1)$

## গুরুত্বপূর্ণ ধারণা:

যদি আমরা জানি:

- $T(e_1)$ কোথায় যায়
- $T(e_2)$ কোথায় যায়

তাহলে আমরা পুরো space কোথায় যাচ্ছে সেটা জেনে ফেলেছি।

কারণ সব ভেক্টরই $e_1$ আর $e_2$ দিয়ে তৈরি।

# **Transformation Matrix** তৈরি করা

ধরো:

$T(e_1) = (a, c)$
$T(e_2) = (b, d)$

তাহলে matrix হবে:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

⚡ লক্ষ্য করো:

- প্রথম column = $T(e_1)$
- দ্বিতীয় column = $T(e_2)$

এটাই transformation matrix।

# এখন যেকোনো ভেক্টর **transform** করা

ধরো:

$x = (x_1, x_2)$

তাহলে:

$$Ax = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

গুণ করলে:

$$= (ax_1 + bx_2 , \\ cx_1 + dx_2)$$

## গভীরভাবে বোঝো

এর মানে:

$$Ax = x_1T(e_1) + x_2T(e_2)$$

মানে matrix আসলে বলছে:

> "তুমি basis vector গুলোকে যেভাবে সরিয়েছো,
> আমি সেই অনুযায়ী পুরো space বানিয়ে দিচ্ছি।"

# জ্যামিতিক অর্থ

Matrix কী করছে?

- $e_1$ কে সরাচ্ছে
- $e_2$ কে সরাচ্ছে
- তারপর পুরো grid সেই অনুযায়ী deform করছে

এজন্য matrix মানেই space warping machine 🔥

# ছোট একটা বাস্তব উদাহরণ

ধরো:

$$T(e_1) = (2,0)$$
$$T(e_2) = (0,3)$$

তাহলে matrix:

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

এটা:

- x direction কে 2 গুণ করছে
- y direction কে 3 গুণ করছে

মানে rectangle stretch হয়ে যাবে।

# Neural Network–এর সাথে সংযোগ

এখানে বলা হয়েছে:

z = Wx + b

এটা একটি dense layer।

এখানে:

- x = input vector
- W = weight matrix
- b = bias
- z = output

## W কী করছে?

W হচ্ছে transformation matrix।

এটা input space–কে warp করছে।

মানে:

- Data কে stretch করছে
- rotate করছে
- shear করছে
- dimension বাড়াতে/কমাতে পারে

সব মিলিয়ে pattern আলাদা করা সহজ করছে।

## Bias (b) কী করছে?

আগে বলেছিলাম:

Linear transformation origin সরাতে পারে না।

কিন্তু Neural Network এ:

$z = Wx + b$

এই +b origin সরিয়ে দেয়।

মানে এটা pure linear না —
এটা আসলে **Affine Transformation**।

# Deep Insight

Neural Network layer =

1️⃣ Linear transformation (W)
2️⃣ Translation (b)
3️⃣ Non-linearity (activation function)

Repeated many times → complex pattern separation

# 3️⃣ Matrix Operations and Composition

# 3.1 Matrix Multiplication

## **Transformation** একটার পর একটা করলে কী হয়**?**

ধরো তুমি:

1️⃣ আগে Transformation **B** করলে
2️⃣ তারপর Transformation **A** করলে

তাহলে চূড়ান্ত ফল হবে একটি নতুন transformation **C**।

গাণিতিকভাবে:

$C = AB$

⚠️ খেয়াল করো:

আগে B, তারপর A
কিন্তু লেখা হয় AB
(ডানদিকেরটা আগে কাজ করে)

## সাইজ **(Dimension)** নিয়ম

যদি:

- A এর size = m × n
- B এর size = n × p

তাহলে:

- C = AB হবে m × p

কারণ মাঝের n মিলতে হবে।

## $C_{ij}$ কিভাবে বের হয়?

C এর i-তম row আর j-তম column বের করতে:

$C_{ij} = \Sigma (A_{ik} B_{kj})$

মানে:

A এর i-তম row
dot product
B এর j-তম column

ছোট উদাহরণ

ধরো:

A =

[1 2]
[3 4]

B =

[5 6]
[7 8]

তাহলে $C_{11}$ হবে:

(1×5 + 2×7) = 5 + 14 = 19

এইভাবেই পুরো matrix তৈরি হয়।

# জ্যামিতিক অর্থ

Matrix multiplication মানে:

> এক transformation এর উপর আরেক transformation বসানো।

যেমন:

- আগে rotate করলে
- তারপর stretch করলে

এটা একসাথে করলে আলাদা ফল আসবে।

## খুব গুরুত্বপূর্ণ নিয়ম

### **Matrix multiplication commutative** না

AB ≠ BA

মানে:

আগে rotate তারপর shear
≠
আগে shear তারপর rotate

ক্রম বদলালে ফল বদলে যায়।

বাস্তব উদাহরণ

ধরো:

- একটা ছবি 90° ঘুরালে
- তারপর ডানদিকে টানলে (shear)

ফল একরকম।

কিন্তু আগে shear করলে
তারপর rotate করলে
সম্পূর্ণ ভিন্ন ফল আসবে।

এটাই non-commutativity।

# 3.2 Identity and Inverses

## Identity Matrix (I)

Identity matrix হলো matrix জগতের সংখ্যা 1।

যেমন:

$I_3$ =
[1 0 0
 0 1 0
 0 0 1]

এটা দিয়ে গুণ করলে কিছুই বদলায় না।

Ix = x

মানে space অপরিবর্তিত থাকে।

জ্যামিতিকভাবে

Identity transformation =
কোনো পরিবর্তন না।

# Inverse Matrix ($A^{-1}$)

Inverse হলো এমন matrix যা আগের transformation পুরোপুরি undo করে।

$A^{-1}A = I$
$AA^{-1} = I$

## উদাহরণ

ধরো:

A = 90° clockwise rotation

তাহলে:

$A^{-1}$ = 90° counter-clockwise rotation

মানে একে অন্যকে বাতিল করে দেয়।

# Deep Geometric Insight

Matrix যদি:

- rotate করে
- stretch করে
- shear করে

তাহলে inverse matrix:

- উল্টো দিকে rotate করবে
- 1/scale দিয়ে shrink করবে
- উল্টো shear করবে

## সব **matrix** এর **inverse** নেই

যদি matrix space কে flatten করে ফেলে
(যেমন 2D → line বানিয়ে ফেলে)

তাহলে information হারিয়ে যায়
→ inverse সম্ভব না।

এমন matrix কে বলে singular matrix।

## Neural Network এর সাথে সম্পর্ক

Neural Network এ:

$z = Wx + b$

একাধিক layer থাকলে:

$z = W3\ W2\ W1\ x$

মানে:

Multiple transformations compose হয়ে যাচ্ছে।

এটাই matrix multiplication।

# 4 Systems of Linear Equations

## Matrix আকারে লেখা

একটা লিনিয়ার সমীকরণ সিস্টেমকে আমরা compact ভাবে লিখতে পারি:

$Ax = b$

এখানে:

- **A** = coefficient matrix (transformation)
- **x** = অজানা ভেক্টর (unknowns)
- **b** = target/output ভেক্টর

## উদাহরণ দিয়ে বোঝি

ধরো সমীকরণ:

2x + y = 5
x  - y = 1

এটা matrix আকারে:

[2 1] [x] = [5]
[1 -1] [y]   [1]

মানে:

Ax = b

# 4.1 Geometric Interpretation

এটাই সবচেয়ে গুরুত্বপূর্ণ অংশ

সমীকরণটা আসলে জিজ্ঞেস করছে:

> কোন ভেক্টর x কে transformation A এর মাধ্যমে পাঠালে
> ঠিক b তে গিয়ে পৌঁছাবে?

মানে:

A(x) = b

## জ্যামিতিকভাবে কী হচ্ছে?

Matrix A পুরো space কে warp করছে।

তুমি এমন একটা x খুঁজছো,
যাকে warp করলে ঠিক b পাওয়া যায়।

**2D** কল্পনা করো

- x হলো input arrow
- A হলো space warping machine
- b হলো target arrow

তাহলে প্রশ্ন:

কোন arrow দিলে,
machine সেটাকে ঠিক b বানাবে?

# যদি **A invertible** হয়

যদি matrix A:

- space flatten না করে
- dimension কমিয়ে না ফেলে
- information না হারায়

তাহলে এর inverse থাকবে।

তখন সমাধান খুব সহজ:

$A^{-1}Ax = A^{-1}b$

কারণ:

$A^{-1}A = I$

তাহলে:

$Ix = A^{-1}b$

মানে:

$x = A^{-1}b$

## জ্যামিতিক অর্থ

b–কে উল্টো transformation দিয়ে ফেরত পাঠালেই x পাওয়া যায়।

মানে:

Forward: A
Backward: $A^{-1}$

# যদি **A invertible** না হয়?

তখন তিনটা অবস্থা হতে পারে:

1. কোনো সমাধান নেই
2. অসীম সমাধান আছে
3. unique solution নেই

কারণ transformation space কে collapse করেছে।

যেমন:

2D → line বানিয়ে ফেললে
আসল 2D point recover করা যাবে না।

# **Machine Learning–**এ এর গুরুত্ব

ধরো Linear Regression:

$Xw = y$

এখানে:

- X = data matrix
- w = weights
- y = output

আমরা w খুঁজছি।

তাত্ত্বিকভাবে:

$w = (X^T X)^{-1} X^T y$

কিন্তু সমস্যা হলো:

Matrix inverse বের করা খুব ব্যয়বহুল।

# কেন **inverse compute** করা খারাপ ধারণা?

Inverse বের করতে সময় লাগে প্রায়:

$O(n^3)$

মানে dimension বড় হলে computation explode করে।

আরও বড় সমস্যা:

Numerical instability।

ছোট rounding error → বড় error হয়ে যেতে পারে।

# তাই বাস্তবে কী করা হয়?

Direct inverse না করে:

✔ Gradient Descent
✔ LU Decomposition
✔ QR Decomposition
✔ SVD

ব্যবহার করা হয়।

## **Gradient Descent** কী করছে আসলে?

এটা inverse বের করছে না।

বরং iterative ভাবে এমন x খুঁজছে
যাতে $Ax \approx b$ হয়।

# Deep Insight (খুব গুরুত্বপূর্ণ)

Linear algebra বলছে:

Solution = Inverse × Target

Machine Learning বলছে:

Direct inverse কোরো না।
Optimization দিয়ে solution খুঁজো।