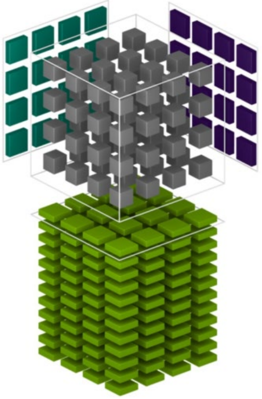
Day 3

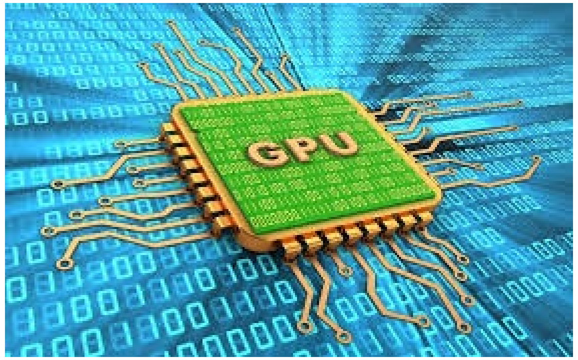
Mathematical Foundation – I

Linear Algebra & Tensor Concept

Why Linear Algebra?

- A good understanding of linear algebra is essential for understanding and working with many machine learning algorithms, especially deep learning algorithms
- Many machine learning algorithms require vectorized inputs (and produce vectorized outputs) and uses vectorization for parallelization of computation to achieve massive speed-up of training/inference of machine learning algorithms (especially on a GPU)





An example of linear equation

- **Rewrite the following linear equation in Matrix Format.**

2x + 4y = 22

3x + y = 13

$$\Rightarrow \begin{pmatrix} 2 & 4 \\ 3 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 22 \\ 13 \end{pmatrix}$$

Linear Algebra

- **Matrices and Vectors**
  - Definitions and terminology
  - Addition & Subtraction
  - Scalar multiplication
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix properties

Matrices and Vectors – Definitions and Terminology

Scalar

24

Object with a single value

Vector

$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$

row

or column $\begin{bmatrix} 2 \\ -8 \\ 7 \end{bmatrix}$

n x 1 matrix

$\mathbb{R}^3$

Usually denoted using small bold letters e.g. $\boldsymbol{x}$

Matrix

$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$

row(s) × column(s)

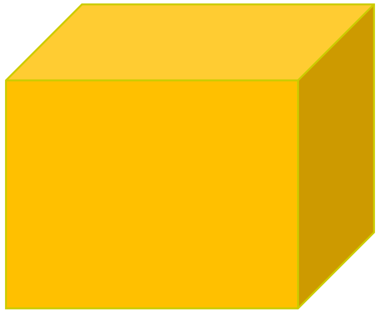
Rectangular array of numbers

$\mathbb{R}^{2 \times 3}$

Usually denoted using uppercase bold e.g. $\boldsymbol{A}$

Tensor

- **Tensor is a generalization of matrices to an arbitrary number of dimensions (or axis)**
- **Tensor is normally denoted as capital non-italicized letter, e.g. A.**



3-D tensor

Real world examples of data tensor

- **Vector data – 2D tensors of shape** `(sample, features)`
- **Timeseries data or sequence data – 3D tensors of shape** `(sample, timesteps, features)`
- **Images – 4D tensors of shape** `(samples, height, width, channels)`
- **Video – 5D tensors of shape** `(samples, frames, height, width, channels)`

Matrices and Vectors – Definitions and Terminology

Vector: An n x 1 matrix.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$y_i = i^{th}$ element

4 dimensional vector

1-indexed vs 0-indexed:

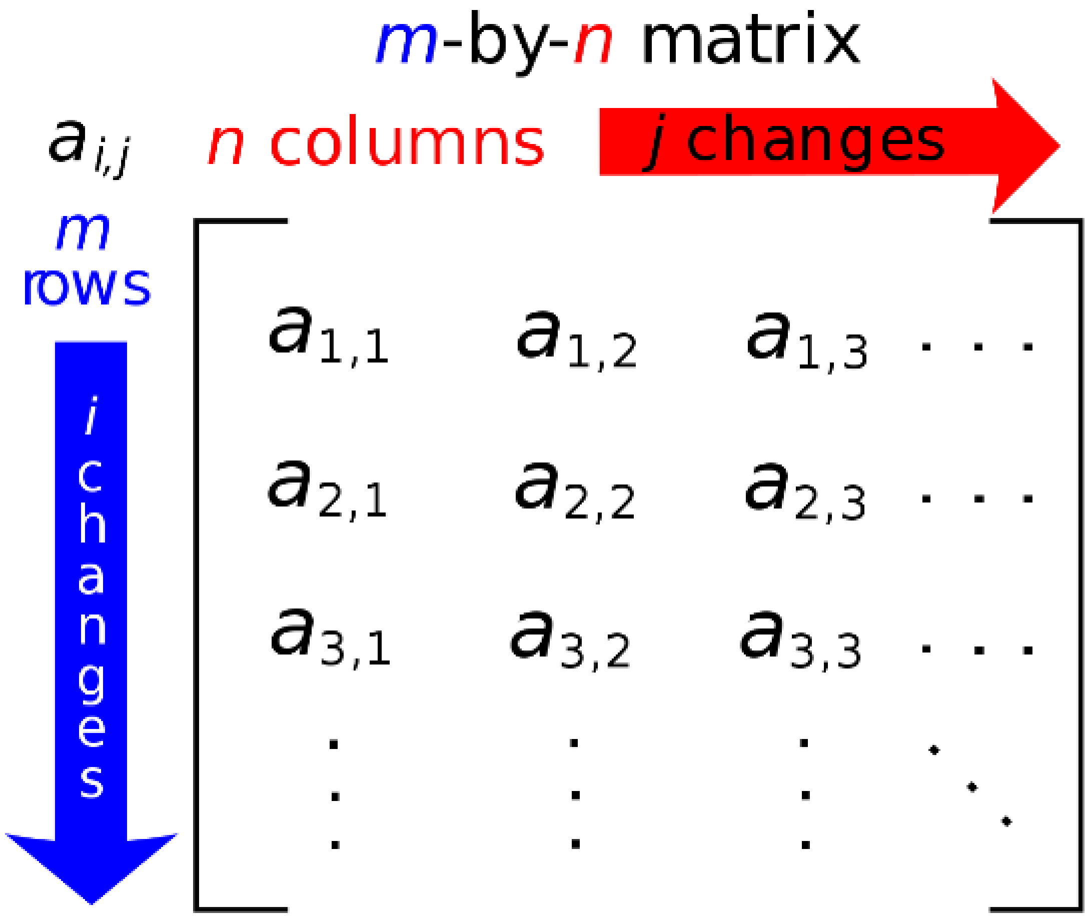
$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$y_1 = 460$ $\qquad$ $y_1 = 232$

$y_3 = 315$ $\qquad$ $y_3 = 178$

Matrices and Vectors – Definitions and Terminology

Matrix elements



$$\begin{bmatrix} 9 & 13 & 5 & 2 \\ 1 & 11 & 7 & 6 \\ 3 & 7 & 4 & 1 \\ 6 & 0 & 7 & 10 \end{bmatrix}$$

$M_{3,4} = 1$
$M_{2,2} = 11$

Matrix Addition & Subtraction

$$\underset{2 \times 2}{\overset{\text{Matrix 1}}{\begin{pmatrix} 10 & 0 \\ -4 & 5 \end{pmatrix}}} + \underset{2 \times 2}{\overset{\text{Matrix 2}}{\begin{pmatrix} -6 & 3 \\ 1 & -7 \end{pmatrix}}} = \underset{2 \times 2}{\overset{\text{Matrix 1 + 2}}{\begin{pmatrix} 4 & 3 \\ -3 & -2 \end{pmatrix}}}$$

$$\begin{bmatrix} 1 & 2 \\ -3 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 5 & -1 \end{bmatrix} = \begin{bmatrix} 1+4 & 2+3 \\ -3+5 & 4+(-1) \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 2 & 3 \end{bmatrix}$$

addition

We cannot add matrices of different dimensions.

$$\begin{bmatrix} 2 & 4 & 3 \\ 6 & 8 & 1 \end{bmatrix} - \begin{bmatrix} 4 & 6 & 3 \\ 5 & 2 & 7 \end{bmatrix} = \begin{bmatrix} 2-4 & 4-6 & 3-3 \\ 6-5 & 8-2 & 1-7 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 0 \\ 1 & 6 & -6 \end{bmatrix}$$

subtraction

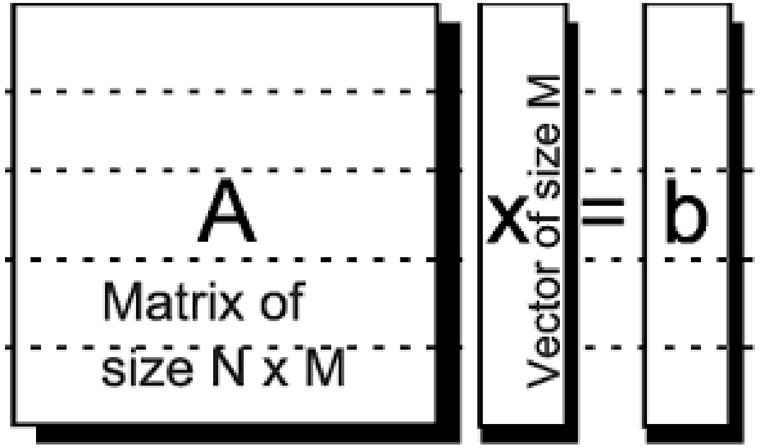
Matrix Scalar Multiplication

$$2 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 2.1 & 2.2 & 2.3 \\ 2.4 & 2.5 & 2.6 \\ 2.7 & 2.8 & 2.9 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

Result is matrix of same dimensions

Matrix-vector multiplication



Result will be an N-dimensional vector

$$\overset{\mathbf{A}}{\begin{bmatrix} 2 & 1 & 2 \\ 3 & 2 & 3 \\ 4 & 1 & 1 \end{bmatrix}} \mathbf{x} \overset{\mathbf{v}}{\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}} = \begin{bmatrix} 2*1 + 1*0 + 2*1 \\ 3*1 + 2*0 + 3*1 \\ 4*1 + 1*0 + 1*1 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix}$$

Matrix-Matrix multiplication

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

In order for this product to be defined, matrix ***A*** must have the same number of columns as ***B*** has
rows. If A is of shape (m x n) and B is of shape (n x p), then **C** is of shape (m x p).

Matrix properties

- **Commutive**
  - **Scalars are commutive**

    3 x 5 is the same as 5 x 3

  - **Matrices are not**

For $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 5 & 4 \\ -5 & 1 \end{bmatrix}$

$$\mathbf{AB} = \begin{bmatrix} 5 & 4 \\ 5 & 9 \end{bmatrix}$$

$$\mathbf{BA} = \begin{bmatrix} 13 & 4 \\ -3 & 1 \end{bmatrix}$$

Matrix properties

- **Associative**
  - **Scalars are associative**

    3 x 5 x 2 the order this is computed in doesn’t matter

    3 x 5 = 15 x 2 = 30

    5 x 2 = 10   3 x 10 = 30

  - **Matrices are associative**

Matrix properties

- **Identity matrix**
  - **1 is identity scalar**

    1 x z = z (true for any value of z), 1 is "identity"

  - **Identity Matrices**

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2 x 2   3 x 3

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

AI = IA = A   I identity matrix

Matrix properties

- **Inverse**
  - **Scalars**

$$3 \quad \times \quad 3^{-1} \quad = \quad 1$$

number x inverse = identity

  - **Only square matrices can have inverses** (but not all do, those that don't are known as singular)

$A \times A^{-1} = A^{-1} \times A = I$

$$\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \times \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Matrix properties

- **Transpose**

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

A $\qquad$ A^T

A motivating example

- **A lot of machine learning algorithm requires the computation of weighted sum of the input features (e.g. linear regression, or a linear layer in deep learning network)**

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$\hat{y}$ is the predicted value.

n is the number of features.

x_i is the i[th] feature value.

θ_j is the j[th] model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

A motivating example

- **We can compute the value in a traditional *for-loop* way:**

```
y_hat = 0

for i in range (1..n):
   y_hat += theta[i] * x[i]

y_hat += b
```

A motivating example

- **Or we express the equation in a more concise way using vectorized form** $\hat{y} = \boldsymbol{\theta}^T \cdot \mathbf{x}$

θ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.

θ^T is the transpose of θ (a row vector instead of a column vector).

$\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.

$\theta^T \cdot \mathbf{x}$ is the dot product of θ^T and $\mathbf{x}$.

A motivating example

- **we can compute the value using faster (parallelized) matrix dot product operation:**

```
y_hat = np.dot(np.transpose(thetha), x)
```

- **The speed-up is especially important in deep learning network typically consist of millions of weights θ (image we have write a for loop that loops millions of times !!)**

Let’s Practice using Numpy