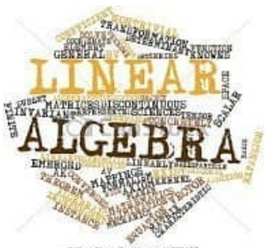
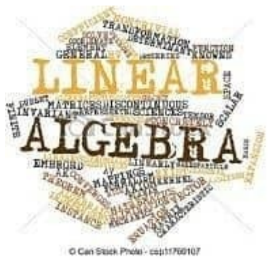
Why Linear Algebra?

- Linear algebra finds widespread application because it generally parallelizes extremely well. Further to that most linear algebra operations can be implemented without messaging passing which makes them amenable to MapReduce implementations.
- Machine Learning deals with the handling of enormous data sets. And an effective way to represent this data is in the form of 2D arrays or rectangular blocks in which each row represents a sample or a complete record and a column represents a feature or an attribute. It is natural to think of the array as a matrix and each column (attribute) as a vector.



Why Linear Algebra?

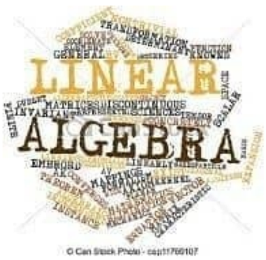
- If you want to get into the theory of it all, you need to know linear algebra. If you want to read white papers and consider cutting edge new algorithms and systems, you need to know a lot of math.
- The concepts of Linear Algebra are crucial for understanding the theory behind Machine Learning, especially for Deep Learning. They give you better intuition for how algorithms really work under the hood, which enables you to make better decisions. So if you really want to be a professional in this field, you cannot escape mastering some of its concepts.



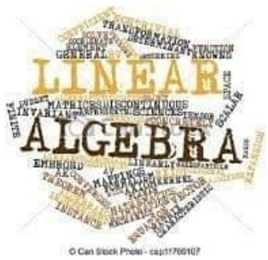
Syllabus

- Scalar
- Vector
- Matrix
- Tensor
- Matrix operations
- Dot product
- Angle between 2 vectors
- Unit vector

t.me/machinelearningfromscratch



- Projections
- Equation of a line, plane, hyperplane, circle, shpere, hypersphere, ellipse, ellipsoid, hyperellipsoid
- Distance between points
- PCA (pricipal component analysis)



- Linear Least Squares
- SVD (singular value decomposition)
- Eigen decomposition
- Eigen values & eigen vectors
- Matrix factorization
- Matrix Decomposition
- LU Decomposition

t.me/machinelearningfromscratch

- Matrix Vector Multiplication
- Matrix Matrix Multiplication
- Matrix Multiplication Properties
- Orthogonalization
- Orthonormalization
- QR Decomposition

Scalar

Scalar is a single number. Usually, we write scalars in italic and lowercase, such as “x”. Scalar could be any type of number, for example

1. number
2. Rational number
3. Irrational number

Code in numpy

```
import numpy as np
x = np.array(1.666666)
```

@learn.machinelearning

vector

Vector A vector is a tuple of one or more values called scalars(array of numbers.). Typically, we denote vectors with lowercase letter, and put elements of a vector as a column enclosed in square brackets

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Code in numpy

```
import numpy as np
x = np.array([1,2,3,4,5,6,7,8,9])
```

@learn.machinelearning

Matrix

Matrix is a 2-D array of numbers, which usually denoted with bold and uppercase variable names, such as:

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m,1} & X_{m,2} & \dots & X_{m,n} \end{bmatrix}$$

Code in numpy

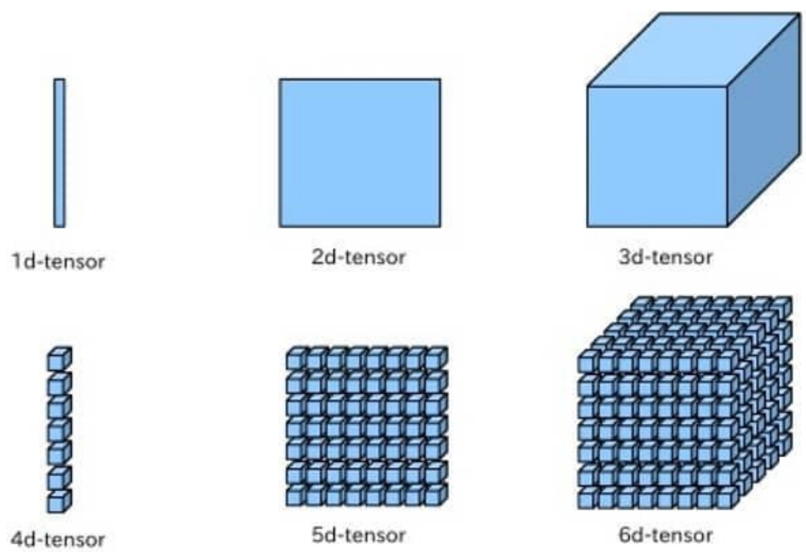
```
import numpy as np
x = np.array([[1,2,3,4],[5,6,7,8],
[9,10,11,12],[13,14,15,16]])
```

@learn.machinelearning

Tensor

Tensor is a k-D array of numbers. The concept of tensor is a bit tricky. You can consider tensor as a container of numbers, and the container could be in any dimension. For example, scalars, vectors, and matrices, are considered as the simplest tensors:

1. Scalar is a 0-dimensional tensor
2. Vector is a 1-dimensional tensor
3. Matrix is a 2-dimensional tensor



source medium.com

@learn.machinelearning

Tensor

commonly, we store time series data in 3-D tensor, image data in 4-D tensor, video data in 5-D tensor, etc.

Code in numpy

```
import numpy as np

x = np.array([[[1, 2, 3],[4, 5, 6],[7, 8, 9]],
 [[10, 20, 30],[40, 50, 60],[70, 80, 90]],
 [[100, 200, 300],[400, 500, 600],[700, 800, 900]]])
```

@learn.machinelearning

Matrix operations

Scalar+Matrix: add the scalar to each element in the matrix

$$\mathbf{C} = \mathbf{A} + b, \text{ where } C_{i,j} = A_{i,j} + b$$

$$\mathbf{C} = \mathbf{A} + b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + 3.14 = \begin{bmatrix} 4.14 & 5.14 \\ 6.14 & 7.14 \end{bmatrix}$$

@learn.machinelearning

Matrix operations

Vector + Matrix : add the vector to each row in the matrix

$$\mathbf{C} = \mathbf{A} + \boldsymbol{v}, \text{ where } C_{i,j} = A_{i,j} + v_j$$

$$\mathbf{C} = \mathbf{A} + \boldsymbol{v} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 11 & 13 & 15 \end{bmatrix}$$

@learn.machinelearning

Matrix operations

Matrix + Matrix: add their corresponding elements as long as the matrices have the same shape

$$\mathbf{C} = \mathbf{A} + \mathbf{B}, \text{ where } C_{i,j} = A_{i,j} + B_{i,j}$$

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

@learn.machinelearning

Matrix operations

Scalar・Matrix: each element of the matrix multiply the scalar

$$\mathbf{C} = \mathbf{A} \cdot b, \text{ where } C_{i,j} = A_{i,j} \cdot b$$

$$\mathbf{C} = \mathbf{A} \cdot b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 5 = \begin{bmatrix} 5 & 10 \\ 15 & 20 \end{bmatrix}$$

@learn.machinelearning

Matrix operations

Matrix • Vector: dot product of each row in the matrix and the vector

$$\mathbf{A} \cdot \boldsymbol{v} = \boldsymbol{b}, \text{ where } A_{i,:} \cdot \mathbf{v} = b_i$$

$$\mathbf{A} \cdot \boldsymbol{v} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 8 + 3 \cdot 9 \\ 4 \cdot 7 + 5 \cdot 8 + 6 \cdot 9 \end{bmatrix} = \begin{bmatrix} 50 \\ 122 \end{bmatrix}$$

Matrix operations

Matrix-Matrix Multiplicatio

This is a more complex operation than matrix addition because it does not simply involve multiplying the matrices element-wise. Instead a more complex procedure is utilised, for each element, involving an entire row of one matrix and an entire column of the other.

AB≠BA

$$\underset{\large A}{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}} \underset{\large B}{\begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix}} = \underset{\large A * B}{\begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix}}$$

@learn.machinelearning

DOT PRODUCT

The dot product of two vectors is a scalar. Dot product of vectors and matrices (matrix multiplication) is one of the most important operations in deep learning.

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

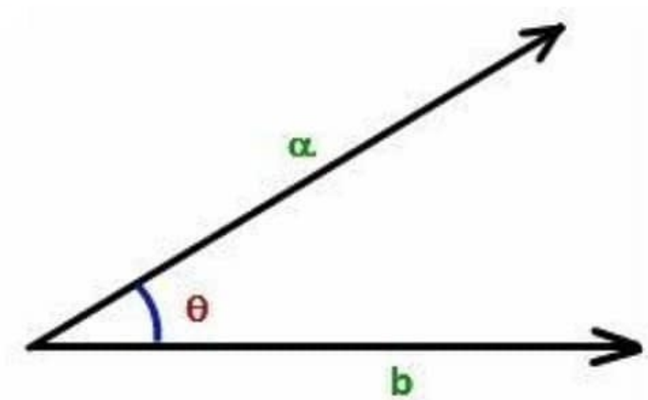
code

```
y = np.array([1,2,3])
x = np.array([2,3,4])
np.dot(y,x) = 20
```

@learn.machinelearning

Angle between Two Vectors

If the two vectors are assumed as $\vec{a}$ and $\vec{b}$ then the dot created is articulated as a.b. Let's suppose these two vectors are separated by angle θ. To know what's the angle measurement we solve with the below formula



$$a.b = |a||b| \cos\theta$$

@learn.machinelearning

Angle between Two Vectors

The angle between two vectors formula is given by

$$\theta = cos^{-1}\frac{a.b}{\mid a \mid\mid b \mid}$$

example

a = (2,2)   b = (0,3)

a.b=(2)(0)+(2)(3) = 6

|a| = √(2^2 + 2^2) = √(8) = 2√2.

|b| = √(0^2 + 3^2) = √(9) = 3.

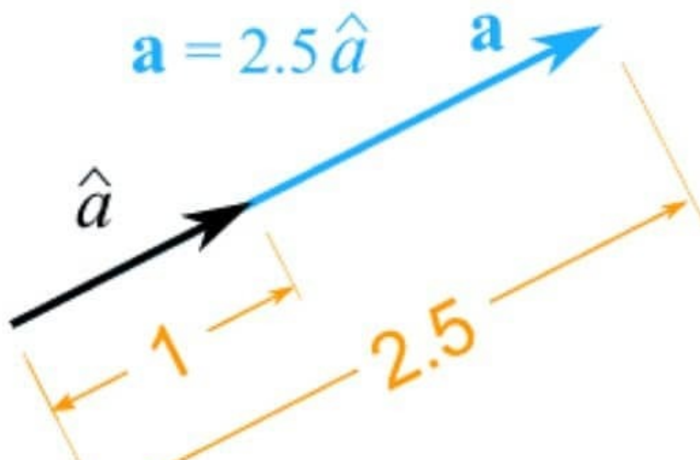
cosθ = 6 / (2√2 * 3) = 1 / √2 = √2 / 2.

This is true for θ = π/4 or 45º.

@learn.machinelearning

Unit vector

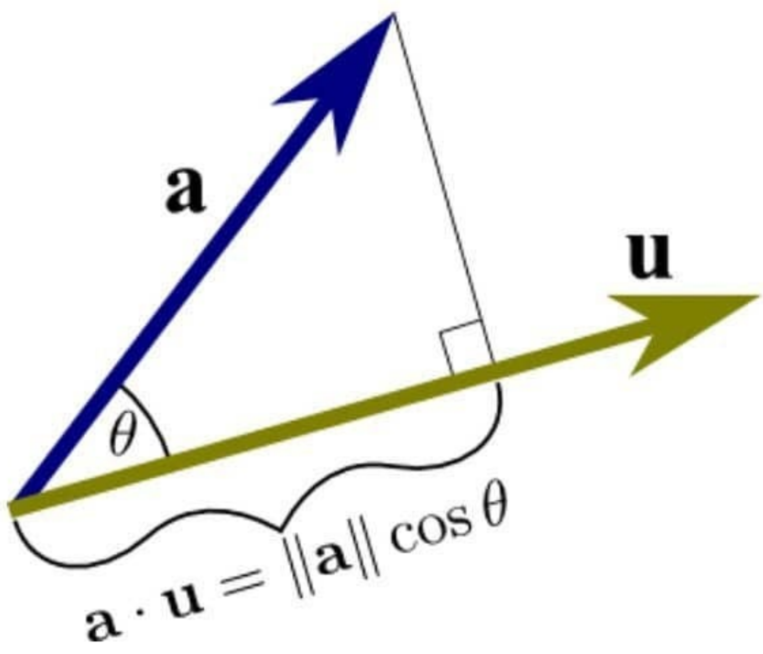
A vector has magnitude (how long it is) and direction. A Unit Vector has a magnitude of 1 and could be pointing in any direction. Such vectors are usually used to specify directions and therefore they do not have any dimension or unit like other vectors. The symbol is usually a lowercase letter with a "hat", such as: (Pronounced "a-hat")



@learn.machinelearning

Projections

One important use of dot products is in projections. The scalar projection of a onto u is the length of the segment AB shown in the figure below. The vector projection of a onto u is the vector with this length that begins at the origin point in the same direction (or opposite direction if the scalar projection is negative) as a.



@learn.machinelearning

Matrix Factorization

- Matrix decompositions are also called as Matrix Factorization it is a method that reduces a matrix into sub-matrices.
- It is same as writing factors to a numbers example we can write 15 as 5X3.
- It is easy to perform complex matrix operations on the decomposed matrix rather than on the original matrix itself.
- Two simple and widely used matrix decomposition methods are the LU matrix decomposition and the QR matrix decomposition

Applications

- Background Removal
- Topic Modeling
- Recommendations Systems
- Eigen Faces
- PCA (Principle component analysis
- Non-negative matrix factorization
- SVD (singular value decomposition)

If you know any matrix factorization application please mention them in the comments.

@learn.machinelearning

LU decomposition

- The LU decomposition is for square matrices and decomposes a matrix into L and U components.

$$S = L \,.\, U$$

- Where S is the square matrix that we wish to decompose, L is the lower triangle matrix and U is the upper triangle matrix.
- The factors L and U are triangular matrices. The factorization that comes from elimination is S = LU.
- LU decomposition is found using an iterative process and can fail for those matrices that cannot be decomposed.
- simplify the solving of systems of linear equations, such as finding the coefficients in linear regression, as well as in calculating the determinant and inverse of a matrix.

@learn.machinelearning

LU decomposition

- **Example (code)**

```
import  numpy as np
from scipy.linalg import lu
```

INPUT

```
A = array([[1, 2, 3], [4, 5, 6],
[7, 8, 9]])
print(A)
# LU decomposition
L, U = lu(A)
print(L)
print(U)
```

OUTPUT

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[[ 1.          0.          0.        ]
 [ 0.14285714  1.          0.        ]
 [ 0.57142857  0.5         1.        ]]

[[  7.00000000e+00   8.00000000e+00   9.00000000e+00]
 [  0.00000000e+00   8.57142857e-01   1.71428571e+00]
 [  0.00000000e+00   0.00000000e+00  -1.58603289e-16]]
```

@learn.machinelearning

QR decomposition

- The QR decomposition is for m x n matrices (not limited to square matrices) and decomposes a matrix into Q and R components.

$$S = Q \cdot R$$

- Where S is the matrix that we wish to decompose, Q a matrix with the size m x m, and R is an upper triangular matrix with the size m x n.
- QR decomposition is found using an iterative process and can fail for those matrices that cannot be decomposed.
- simplify the solving of systems of linear equations, such as finding the coefficients in linear regression, as well as in calculating the determinant and inverse of a matrix.

@learn.machinelearning

QR decomposition

Example (Code)

```
import numpy as np
from numpy.linalg import qr
```

INPUT

```
A = np.array([[1, 2], [3, 4], [5, 6]])
print(A)
# QR decomposition
Q, R = qr(A, 'complete')
print(Q)
print(R)
```

OUTPUT

```
[[1 2]
 [3 4]
 [5 6]]

[[-0.16903085  0.89708523  0.40824829]
 [-0.50709255  0.27602622 -0.81649658]
 [-0.84515425 -0.34503278  0.40824829]]

[[-5.91607978 -7.43735744]
 [ 0.          0.82807867]
 [ 0.          0.        ]]
```

@learn.machinelearning