# FTDS // Matrix


Hacktiv8 DS Curriculum Team

Phase 0 Learning Materials

Hacktiv8 DS
Curriculum
Team


Objectives 03

Definition, Notation, and Types 04

Addition/Substraction 07

Multiplication/Division 09

Matrix Operations 16

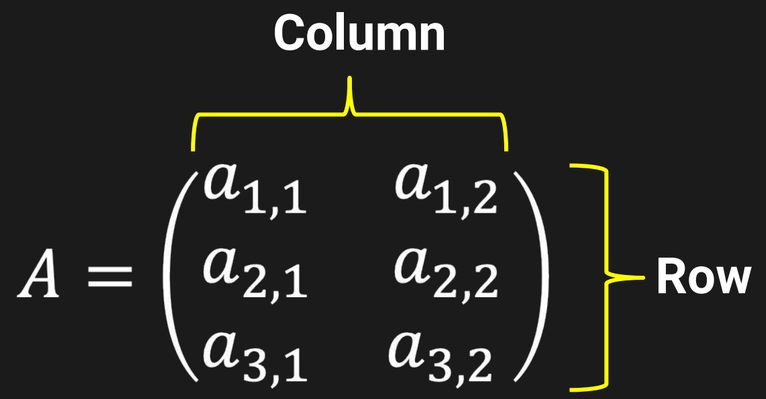
Tensor 19


# Contents

- Basic understanding of matrix
- Able to perform operations of matrix
- Introduce to tensor
- Able to implement the matrix concepts to Python

**Not like vector that is a one-dimensional array, otherwise, matrix is a two-dimensional array that consists of row and column.**

**Suppose that we have 3-row, 2-column matrix:**




**We can express a matrix as an array:**

$$A = ((a_{1,1}, a_{1,2}), (a_{2,1}, a_{2,2}), (a_{3,1}, a_{3,2}))$$

# Define a Matrix on Code

```
# create a matrix
import numpy as np
# define matrix
v = np.array([ [1, 2, 3], [1, 0, 3] ])
print(v)
```

***Output:*** [ [1 2 3],
[1 0 3] ]

# Types of Matrix

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

**Square Matrix**
***X(m,m)***

**X=array( [4, 9, 2],**
**[3, 5, 7],**
**[8, 1, 6] )**

$$\begin{bmatrix} 4 & 9 & 2 \\ 0 & 5 & 7 \\ 0 & 0 & 6 \end{bmatrix}$$

**Upper Triangle Matrix**

**np.triu(X)**

$$\begin{bmatrix} 4 & 0 & 0 \\ 3 & 5 & 0 \\ 8 & 1 & 6 \end{bmatrix}$$

**Lower Triangle Matrix**

**np.tril(X)**

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

**Diagonal Matrix**

**np.diag(X)**

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

**Identity Matrix**

**np.eye(shape)**

**Matrix can be added or substracted with other matrix. Addition/substraction will be performed on each element. The size of matrices shoud be the same. Supposed we have 3x2 matrices.**

$$C = A \pm B$$

$$C = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} \pm \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} \pm b_{1,1} & a_{1,2} \pm b_{1,2} \\ a_{2,1} \pm b_{2,1} & a_{2,2} \pm b_{2,2} \\ a_{3,1} \pm b_{3,1} & a_{3,2} \pm b_{3,2} \end{pmatrix}$$

# Addition/ Substraction on Code

```
# Addition
import numpy as np
# define first matrix
A = np.array([ [1, 2, 3],
               [4, 5, 6] ])
# define second matrix
B = np.array([ [1, 2, 3],
               [4, 5, 6] ])
# add matrices
C=A+B
print(C)
```

***Output:***
```
[ [ 2  4  6 ]
  [ 8 10 12 ] ]
```

```
# Substraction
import numpy as np
# define first matrix
A = np.array([ [1, 2, 3],
               [4, 5, 6] ])
# define second matrix
B = np.array([ [1, 2, 3],
               [4, 5, 6] ])
# substract matrices
C=A-B
print(C)
```

***Output:***
```
[ [ 0 0 0 ]
  [ 0 0 0 ] ]
```

There are three ways to do matrix multiplication, which are matrix-scalar, hadamard product, and matrix-matrix.

$$B = kA$$

To perform matrix multiplication by a scalar, just multiply each element with the scalar. Let we have a 3x2 matrix.

$$B = k\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} = \begin{pmatrix} ka_{1,1} & ka_{1,2} \\ ka_{2,1} & ka_{2,2} \\ ka_{3,1} & ka_{3,2} \end{pmatrix}$$

**Basically, Hadamard product is not really different to the vector - vector multiplication. The operation is performed element-wise. The multiplication symbol will be notated by ○ (circle).**

$$C = A \circ B$$

$$C = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} \circ \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} \times b_{1,1} & a_{1,2} \times b_{1,2} \\ a_{2,1} \times b_{2,1} & a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{3,1} & a_{3,2} \times b_{3,2} \end{pmatrix}$$

**Matrix-Matrix multiplication is different to the other multiplications and quite complicated. The operation should follow the rule that The number of columns in the first matrix must equal the number of rows in the second matrix.**

$$C(m,n) = A(m,k) \cdot B(k,n)$$

$$C = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} \times b_{1,1} + a_{1,2} \times b_{2,1} & a_{1,1} \times b_{1,2} + a_{1,2} \times b_{2,2} \\ a_{2,1} \times b_{1,1} + a_{2,2} \times b_{2,1} & a_{2,1} \times b_{1,2} + a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} & a_{3,1} \times b_{1,2} + a_{3,2} \times b_{2,2} \end{pmatrix}$$

# Multiplication on Code //1

```
# Matrix-Scalar
from numpy import array
# define vector
A = array([ [1, 2, 3], [4, 5, 6] ])
k = 2
 # multiplying vectors
C = k * A
print(C)
```

***Output:*** [ [2, 4, 6],
[8, 10, 12] ]

```
# Hadamard Product
from numpy import array
# define vector
A = array([ [1, 2, 3], [4, 5, 6] ])
B = array([ [1, 2, 1], [0, 2, 3] ])
# multiplying vector
c = a * 3
print(c)
```

***Output:*** [ [1, 4, 3],
[0, 10, 18] ]

# Multiplication on Code //2

```
# matrix dot product
from numpy import array
# define first matrix
A = array([ [1, 2],
        [3, 4],
        [5, 6]])
# define second matrix
B = array([ [1, 2],
        [3, 4]])

# multiply matrices
C = A.dot(B)
print(C)
# multiply matrices with @ operator
D=A@B
print(D)
```

***Output:***

```
[[ 7 10]
 [15 22]
 [23 34]]

[[ 7 10]
 [15 22]
 [23 34]]
```

**A matrix division is performed the same as vector division and Hadamard product.**

$$C = \frac{A}{B}$$

$$C = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} / \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{pmatrix} = \begin{pmatrix} a_{1,1}/b_{1,1} & a_{1,2}/b_{1,2} \\ a_{2,1}/b_{2,1} & a_{2,2}/b_{2,2} \\ a_{3,1}/b_{3,1} & a_{3,2}/b_{3,2} \end{pmatrix}$$

# Division on Code

```
# Matrix-Matrix Division
import numpy as np
# define vector
A = np.array([ [1, 2, 3], [4, 5, 6] ])
B = np.array([ [1, 2, 1], [1, 2, 3] ])
# multiplying vector
C = A/B
print(C)
```

***Output:*** [ [1, 1, 3],
[4, 2.5, 2] ]

# Transpose

**A defined matrix can be transposed, which creates a new matrix with the number of columns and rows flipped. This is denoted by the superscript *T* next to the matrix $A^T$ .**

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

$$A^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

```
# transpose matrix
from numpy import array
# define matrix
A = array([ [1, 2],
            [3, 4],
            [5, 6]])
# calculate transpose
C = A.T
print(C)
```

***Output:*** [ [1, 3, 5],
[2, 4, 6] ]

# Inverse

**Matrix inversion is a process that finds another matrix that when multiplied with the matrix, results in an identity matrix. The operation of inverting a matrix is indicated by a −1 superscript next to the matrix; for example, $A^{-1}$. The result of the operation is referred to as the inverse of the original matrix; for example, B is the inverse of A.**

**The matrix inversion operation is not computed directly, but rather the inverted matrix is discovered through a numerical operation, where a suite of efficient methods may be used, often involving forms of matrix decomposition.**

```
# invert matrix
from numpy import array
from numpy.linalg import inv
# define matrix
A = array([ [1.0, 2.0],
[3.0, 4.0]])
# invert matrix
B = inv(A)
print(B)
```

***Output***: [[ 1. 0. ]
[ 8.88e-16 1. ]]

# Determinant

**The determinant describes the relative geometry of the vectors that make up the rows of the matrix. More specifically, the determinant of a matrix A tells you the volume of a box with sides given by rows of A.**

**It is denoted by the det(A) notation or |A|, where A is the matrix on which we are calculating the determinant. The matrix should be a square matrix**

```
# matrix determinant
from numpy import array
from numpy.linalg import det
# define matrix
A = array([ [1, 2, 3], [4, 5, 6],
[7, 8, 9]])
# calculate determinant
B = det(A)
print(B)
```

***Output:*** [ [1, 3, 5],
[2, 4, 6] ]

**A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array. A vector is a one-dimensional or first order tensor and a matrix is a two-dimensional or second order tensor.**

**Suppose we have a 3 x 3 x 3 three-dimensional tensor:**

$$T = \left( \begin{pmatrix} t_{1,1,1} & t_{1,2,1} & t_{1,2,1} \\ t_{2,1,1} & t_{2,2,1} & t_{2,3,1} \\ t_{3,1,1} & t_{3,2,1} & t_{3,3,1} \end{pmatrix}, \begin{pmatrix} t_{1,1,2} & t_{1,2,2} & t_{1,2,2} \\ t_{2,1,2} & t_{2,2,2} & t_{2,3,2} \\ t_{3,1,2} & t_{3,2,2} & t_{3,3,2} \end{pmatrix}, \begin{pmatrix} t_{1,1,3} & t_{1,2,3} & t_{1,2,3} \\ t_{2,1,3} & t_{2,2,3} & t_{2,3,3} \\ t_{3,1,3} & t_{3,2,3} & t_{3,3,3} \end{pmatrix} \right)$$

# Define a Tensor on Code

```
# create tensor
from numpy import array
T = array([ [[1,2,3], [4,5,6], [7,8,9]],
        [[11,12,13], [14,15,16], [17,18,19]],
[[21,22,23], [24,25,26], [27,28,29]] ])
print(T.shape)
print(T)
```

***Output:***

```
(3, 3, 3)

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

 [[11 12 13]
  [14 15 16]
  [17 18 19]]

 [[21 22 23]
  [24 25 26]
  [27 28 29]] ]
```