AIoT Coding, Engineering and Entrepreneurial (AIoT CE^2) Skills Education for Gifted Students
– Advanced Python (NumPy)

CityU
Department of Electrical Engineering
香港城市大學
City University of Hong Kong

Table of Contents

The NumPy Library

- NumPy is a Python library for scientific computing
- It is pre-installed in Anaconda and Google Colab
- The NumPy library provides a lot of useful functions for numerical computations
- For example, to calculate the mean:

```python
import numpy as np
```

```python
x = [1,2,3]
print('The mean is:',np.mean(x))
```

```
The mean is: 2.0
```

```python
(1+2+3)/3
```

```
2.0
```

NumPy Array

- Large amount of data are processed in machine learning
- Data are represented in computer as n-d arrays, aka tensors
- NumPy provides a special data structure called NumPy array to represent tensors

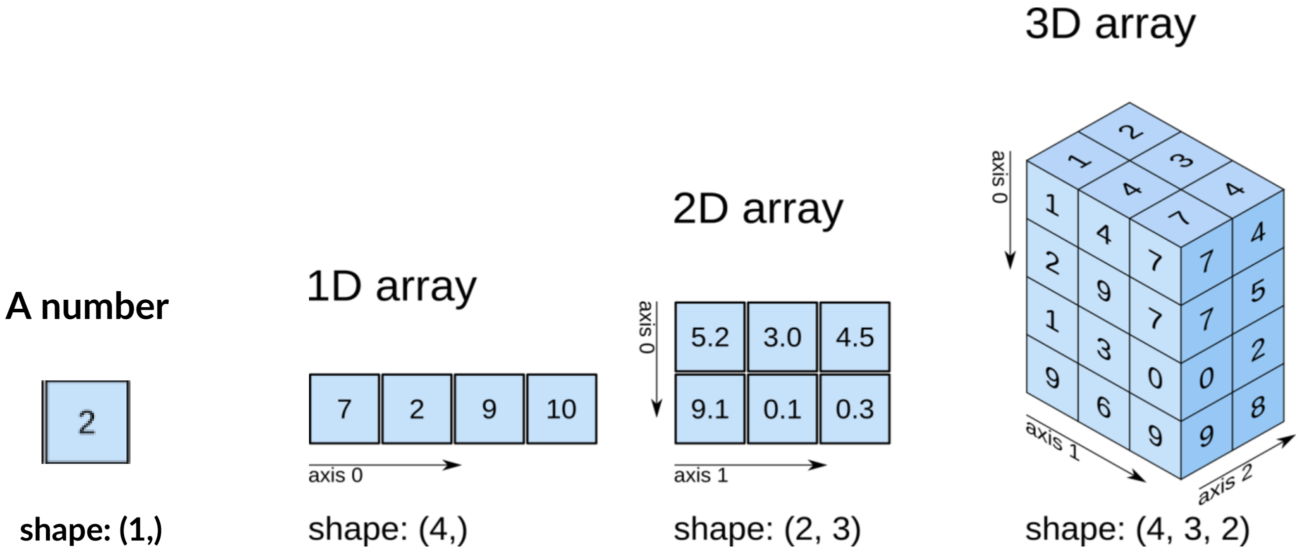
You see an image like this

Your computer sees an image like this

```
array([[1.        , 1.        , 1.        , ..., 0.99719451, 1.        ,
         1.        ],
       [1.        , 1.        , 1.        , ..., 0.9925698 , 1.        ,
         1.        ],
       [1.        , 1.        , 1.        , ..., 0.28915706, 1.        ,
         1.        ],
       ...,
       [1.        , 1.        , 1.        , ..., 1.        , 1.        ,
         1.        ],
       [1.        , 1.        , 1.        , ..., 1.        , 1.        ,
         1.        ],
       [1.        , 1.        , 1.        , ..., 1.        , 1.        ,
         1.        ]])
```

NumPy Array

- NumPy allows users to declare N dimensional (nd) arrays (Tensors)
- Using the function np.array( )



Scalar:
Rank 0

Vector:
Rank 1

Matrix:
Rank 2

Tensor:
Rank n

Array Creation

- There are different ways to create NumPy arrays:

1. np.array()
   - ➢ Create an array with from a list or tuple
   - ➢ Example:

```
1  np.array([1,2,3])
```

```
array([1, 2, 3])
```

2. np.zeros(*shape*), np.ones(*shape*)
   - ➢ Create an array full of zeros/ones with the specified shape
   - ➢ Example:

```
1  np.zeros((1, 4))
2
```

```
array([[0., 0., 0., 0.]])
```

```
1  np.ones((2, 2))
2
```

```
array([[1., 1.],
       [1., 1.]])
```

Array Creation

3. np.arange(*start, stop, step*)
   - Similar usage as range() function, but return an array instead
   - Example:

```
np.arange(11,26,2)
```

```
array([11, 13, 15, 17, 19, 21, 23, 25])
```

```
np.arange(0,0.5,0.1)
```

```
array([0. , 0.1, 0.2, 0.3, 0.4])
```

4. np.linspace(*start, stop, num*)
   - Create an array of *num* evenly spaced samples
   - Over the interval [start, stop]
   - Example:

```
np.linspace(0,0.5,5)
```

```
array([0.   , 0.125, 0.25 , 0.375, 0.5  ])
```

Array Creation

5. np.identity(*n*)
   - Create an identity array of size *n*
   - i.e. a square array with ones on the main diagonal
   - Example:

```
np.identity(4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Exercise 1

- NumPy array indexing and slicing work similar as list, try the codes below:

```python
a_list = [0,1,2,3,4,5]
b_list = a_list[1:4] #b is some elements of a
print('a before:',a_list)
print('b before:', b_list)
b_list[0] = 1000 #change an element of b
print('a after:', a_list)
print('b after:',b_list)
```

```python
a_array = np.array([0,1,2,3,4,5])
b_array = a_array[1:4] #b is some elements of a
print('a before:', a_array)
print('b before:', b_array)
b_array[0] = 1000 #change an element of b
print('a after:',a_array)
print('b after:',b_array)
```

- Does a_list change after changing an element of b_list? How about a_array?
- Can you propose a way where a_array won't be changed after changing b_array?

Array Shapes

- The shape of an array can be accessed by array.shape attribute

```python
x = np.array([[1,2],[3,4]])
print('Array x: \n', x)
print('Shape of x:', x.shape)


y = np.array([[1],[2],[3]])
print('Array y: \n', y)
print('Shape of y:', y.shape)
```

```
Array x: 
 [[1 2]
 [3 4]]
Shape of x: (2, 2)
Array y: 
 [[1]
 [2]
 [3]]
Shape of y: (3, 1)
```

Array Shapes

- NumPy arrays can be reshaped by array.reshape() method
- Equivalent to the previous slide:

```python
x = np.arange(1,5)
x = x.reshape(2,2)
print('Array x: \n', x)
print('Shape of x:', x.shape)


y = np.arange(1,4)
y = y.reshape(3,1)
print('Array y: \n', y)
print('Shape of y:', y.shape)
```

```
Array x:
 [[1 2]
 [3 4]]
Shape of x: (2, 2)
Array y:
 [[1]
 [2]
 [3]]
Shape of y: (3, 1)
```

Exercise 2

- Try the code below, what is the output?
- What does -1 in x.reshape(-1,2) mean?
- Can you try reshape(-1,2) on the arrays returned by np.arange(1,n) with different values of n, where n is an integer?
- What is the condition for n which no error is returned ?

```
x = np.arange(1,7)
x = x.reshape(-1,2)
print('Array x: \n', x)
print('Shape of x:', x.shape)

```

Array Arithmetic

- NumPy provides a linear algebra module for matrices and vectors computations:

1. np.transpose(a) or a.transpose() or a.T
   - Compute the transpose of the matrix **a**
   - Example:

```
a = np.arange(6).reshape(2,3)
a
```

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
np.transpose(a)
```

```
array([[0, 3],
       [1, 4],
       [2, 5]])
```

Array Arithmetic

2. np.dot(a, b)

   - Compute the dot product of the vectors **a** and **b**

$$\mathbf{a} \cdot \mathbf{b} = \sum_i a_i b_i$$

   - Example:

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \\ &= 1 \times 4 + 2 \times 5 + 3 \times 6 \\ &= 32 \end{aligned}$$

```
a = np.array([1,2,3])
b = np.array([4,5,6])

print('The dot prodcut of a and b is:', np.dot(a,b))
```

```
The dot prodcut of a and b is: 32
```

Array Arithmetic

3. np.linalg.norm(x, ord)

   - Compute the norm of the vector/matrix **x**
   - It computes the Frobenius norm if ord=None (default value)

$$||\mathbf{x}||_F = \sqrt{\sum_i \sum_j |x_{ij}|^2}$$

   - Example:

```
a = np.array([3,4])

np.linalg.norm(a)
```

```
5.0
```

```
a = np.array([[1,2],[3,4]])
np.linalg.norm(a)
```

```
5.477225575051661
```

Array Arithmetic

4. np.matmul(a, b)
   - Multiply matrix **a** and matrix **b**
   - Example:

```
a = np.identity(2)
b = np.array([[4, 1],
              [2, 2]])
```

```
np.matmul(a,b)
```

```
array([[4., 1.],
       [2., 2.]])
```

Array Arithmetic

5. np.linalg.inv(A)
   - Compute the inverse A^{-1} of matrix A, such that $AA^{-1} = \mathbb{1}$
   - Example:

```
a = np.array([[2, 2],
              [3, 4]])

inv_a = np.linalg.inv(a)
inv_a
```

```
array([[ 2. , -1. ],
       [-1.5,  1. ]])
```

```
np.matmul(a,inv_a)
```

```
array([[1., 0.],
       [0., 1.]])
```

Array Arithmetic

6. np.linalg.solve(a,b)

   ➢ Solve a linear matrix equation, or system of linear scalar equations

   ➢ Example, solving $\mathbf{Ax} = \begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$:

```python
A = np.array([[1,2],[3,5]])
b = np.array([1,2])

np.linalg.solve(A,b)
```

```
array([-1.,  1.])
```

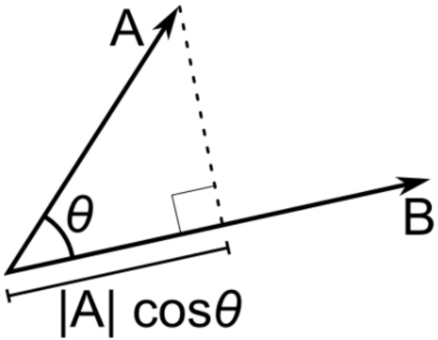
Solution:

$$\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Exercise 3

- Recall the geometric interpretation of the dot product:

$$\frac{\mathbf{A} \cdot \mathbf{B}}{|A||B|} = \cos\theta$$



- Verify that the angle between the vectors
  A = [0,1,2] and B=[3,0,0] is $\pi/2$ (90 degree)

- Hints:
  - Use np.arccos(x) to calculate $cos^{-1}x$
  - Use np.pi to get the value of π

Array Broadcasting

- Broadcasting refer to how NumPy treats arrays with different shapes during arithmetic operations
- If two arrays have the same shape, the operations will be performed elementwise
- If two arrays have different shapes, the smaller array will be broadcasted across the larger array

Array Broadcasting

- Consider the scalar multiplication, ie multiplying the whole array by 2:

$$2\mathbf{x} = 2 \cdot \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 & \cdots & 2x_n \end{bmatrix}$$

- Is it necessary to declare an array of numbers 2 with the same shape as x to perform the multiplication, like below?

```
a = np.repeat(2,5)
a
```

```
array([2, 2, 2, 2, 2])
```

```
x = np.arange(5)
x
```

```
array([0, 1, 2, 3, 4])
```

```
a*x
```

```
array([0, 2, 4, 6, 8])
```

Array Broadcasting

- Indeed, we can just multiply the array with an integer 2
- The integer 2 is broadcasted and multiplied with all elements of array x
- It yields the same result as the previous slide:

```python
x = np.arange(5)
x
```

```
array([0, 1, 2, 3, 4])
```

```python
a = 2
a*x
```

```
array([0, 2, 4, 6, 8])
```

Array Broadcasting

- Similar behavior when multiplying a 2x1 array with 2x4 array
- Array a is broadcasted along axis 1 of array x
- Don’t confuse it with matrix multiplication!

```
a = np.arange(2).reshape(2,1)
a
```

```
array([[0],
       [1]])
```

```
x = np.arange(8).reshape(2,4)
x
```

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
a*x
```

```
array([[0, 0, 0, 0],
       [4, 5, 6, 7]])
```

Broadcasting Rules

1. If the arrays do not have the same rank, prepend the shape of the lower rank array with 1 until both arrays have the same rank.

```
Image  (3d array): 256 x 256 x 3
Scale  (1d array):             3
Result (3d array): 256 x 256 x 3
```

→

```
Image  (3d array): 256 x 256 x 3
Scale  (1d array): 1   x 1   x 3
Result (3d array): 256 x 256 x 3
```

Examples are taken from documentation:
https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

Broadcasting Rules

2. The two arrays are said to be compatible in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.

Compatible in all dimensions

```
A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5
```

Incompatible in dimension 1

```
A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3
```

Broadcasting Rules

3. The arrays can be broadcast together if they are compatible in all dimensions.

Compatible in all dimensions

```
A = np.arange(15*3*5).reshape(15,3,5)
B = np.arange(15*5).reshape(15,1,5)
(A*B).shape
```

```
(15, 3, 5)
```

Incompatible in dimension 1

```
A = np.arange(2).reshape(2,1)
B = np.arange(8*4*3).reshape(8,4,3)
(A*B).shape
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-b38e2877dae0> in <module>
      1 A = np.arange(2).reshape(2,1)
      2 B = np.arange(8*4*3).reshape(8,4,3)
----> 3 (A*B).shape

ValueError: operands could not be broadcast together with shapes (2,1) (8,4,3)
```

Broadcasting Rules

4. After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

Maximum of dimensions of A or B

Broadcasting Rules

5. In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension

```
a = np.arange(2).reshape(2,1)
a
```

```
array([[0],
       [1]])
```

```
x = np.arange(8).reshape(2,4)
x
```

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
a*x
```

```
array([[0, 0, 0, 0],
       [4, 5, 6, 7]])
```

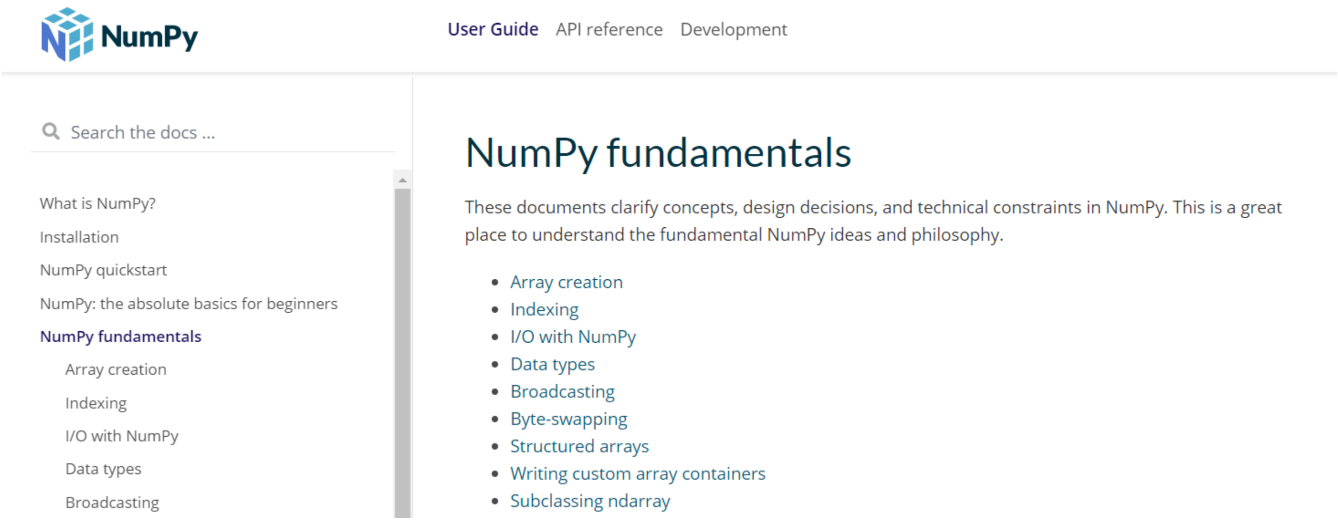
Exercise 4

- Optimization problems are common in machine learning, one of the algorithms to solve optimization problems is the Newton-Raphson algorithm.
- Refer to your Jupyter notebook, implement the Newton-Raphson algorithm for linear regression using NumPy.

More on the NumPy Library

- More on the NumPy library:

  https://docs.scipy.org/doc/numpy/user/basics.html



Next Lesson…

Advanced Python
Pandas
- Series and DataFrame
- Descriptive statistics in Pandas
- Group By: split-apply-combine
- Data visualization

專業 創新 胸懷全球
Professional·Creative
For The World

Department of
Electrical Engineering
香港城市大學
City University of Hong Kong