CORE
Control + Optimization Research Lab

Numpy

이동민

삼성전자 서울대 공동연구소

Jul 15, 2019

Outline

- Numpy
  - Numerical Python
  - Ndarray
  - Reshape & Indexing, Slicing
  - Creating Functions
  - Operation Functions

Numerical Python

- 코드로 방정식 표현하기

$$\begin{aligned} 2x_1 + 2x_2 + x_3 &= 9 \\ 2x_1 - x_2 + 2x_3 &= 6 \\ x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad \begin{bmatrix} 2 & 2 & 1 & 9 \\ 2 & -1 & 2 & 6 \\ 1 & -1 & 2 & 5 \end{bmatrix}$$

```
coefficient_matrix = [[2, 2, 1], [2, -1, 2], [1, -1, 2]]
constant_vector = [9, 6, 5]
```

- 하지만 위의 코드처럼 표현할 때 문제가 생김
  - 다양한 Matrix 계산을 어떻게 만들 것인가?
  - 굉장히 큰 Matrix에 대한 표현
  - 처리 속도 문제

- 따라서 적절한 패키지를 활용하는 것이 좋은 방법 → Numpy

Numerical Python

- Numpy란?
  - Numerical Python
  - Vector와 Matrix와 같은 Array 연산의 사실상 표준
  - 한글로 넘파이로 주로 통칭, 넘피/늄파이라고 부르기도 함

- Numpy 특징
  - 일반 List에 비해 빠르고(C언어로 짜여져 있음), 메모리 효율성이 높음
  - 반복문(for문이나 list compression) 없이 데이터 배열에 대한 처리를 지원함
  - 선형대수와 관련된 다양한 기능을 제공함
  - C, C++, 포트란 등의 언어와 통합 가능

Ndarray

- Numpy import

```
import numpy as np
```

- Array creation
  - Numpy는 np.array 함수를 통해 배열을 생성 → ndarray (n-dimension array)
  - Numpy는 하나의 데이터 type만 배열에 넣을 수 있음
  - List와 가장 큰 차이점 → Dynamic typing not supported
  - C의 array를 사용하여 배열을 생성함

```
array_creation = np.array([[1, 4, 5, "8"], [3, 5, 6, 7]], float)
print(array_creation)
print(type(array_creation))
print(array_creation.shape)
print(array_creation.dtype)
'''
[[1. 4. 5. 8.]
 [3. 5. 6. 7.]]
<class 'numpy.ndarray'>
(2, 4)
float64
'''
```

Ndarray

- Array shape
  - Vector

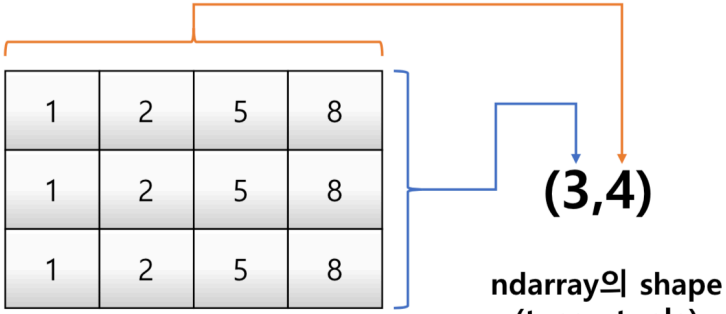
ndarray의 구성

(4,)

ndarray의 shape
(type : tuple)

```
vector = [1,2,3,4]
print(np.array(vector, float).shape)
'''
(4,)
'''
```

  - Matrix



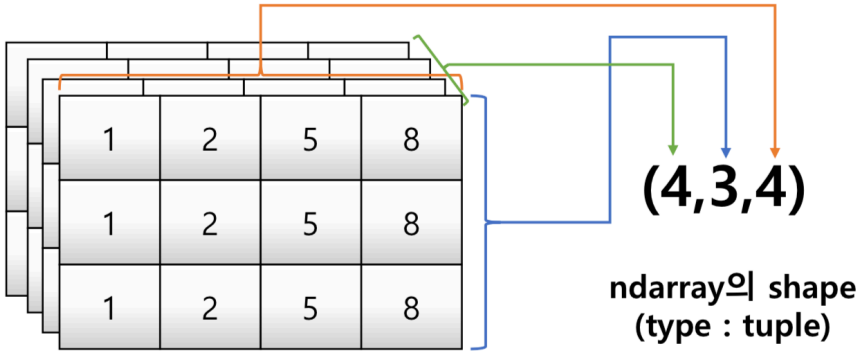
(3,4)

ndarray의 shape
(type : tuple)

```
matrix = [[1,2,5,8], [1,2,5,8], [1,2,5,8]]
print(np.array(matrix, float).shape)
'''
(3, 4)
'''
```

Ndarray

- Array shape
  - 3rd Order Tensor




```
tensor = [[[1,2,5,8], [1,2,5,8], [1,2,5,8]],
          [[1,2,5,8], [1,2,5,8], [1,2,5,8]],
          [[1,2,5,8], [1,2,5,8], [1,2,5,8]],
          [[1,2,5,8], [1,2,5,8], [1,2,5,8]]]
print(np.array(tensor, float).shape)
'''
(4, 3, 4)
'''
```

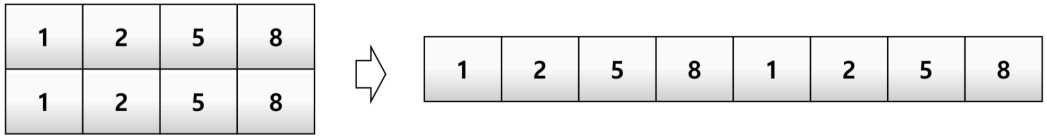
Ndarray

- Array type

```
array_dtype = np.array([[1, 2, 3], [4.5, 5, 6]], dtype=int)
print(array_dtype)
'''
[[1 2 3]
 [4 5 6]]
'''

array_dtype1 = np.array([[1, 2, 3], [4.5, 5, 6]])
print(array_dtype1)
'''
[[1.  2.  3. ]
 [4.5 5.  6. ]]
'''
```

Reshape & Indexing, Slicing

- Reshape



(2,4) (8,)

```
matrix = [[1,2,3,4], [1,2,5,8]]

reshape_array = np.array(matrix).reshape(8)
print(reshape_array)
'''
[1 2 3 4 1 2 5 8]
'''


reshape_array1 = np.reshape(matrix, [1, 8])
print(reshape_array1)
'''
[[1 2 3 4 1 2 5 8]]
'''
```

Reshape & Indexing, Slicing

- Indexing

```
array_indexing = np.array([[1, 2, 3], [4, 5, 6]])
print(array_indexing[0][0])
print(array_indexing[0,0])
'''
1
1
'''


array_indexing[0,0] = 10 # matrix 0,0에 10 할당
print(array_indexing)
'''
[[10  2  3]
 [ 4  5  6]]
'''
```

- Slicing

```
array_slicing = np.array([[1,2,5,8], [1,3,6,9], [1,4,7,10], [1,2,3,4]])
print(array_slicing[:2, :])
print(array_slicing[:, 1:3])
'''
[[1 2 5 8]
 [1 3 6 9]]
[[2 5]
 [3 6]
 [4 7]
 [2 3]]
'''
```

Creating Functions

- Zeros & Ones

```
zeros = np.zeros(shape=(10))
zeros1 = np.zeros((2, 5))
print(zeros)
print(zeros1)
'''
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
'''
```

```
ones = np.ones(10)
ones1 = np.ones((5, 2))
print(ones)
print(ones1)
'''
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
'''
```

- Something like

```
matrix = np.array([[1,3,5,7], [2,4,8,10]])
zeros = np.zeros_like(matrix)
print(zeros)
'''
[[0 0 0 0]
 [0 0 0 0]]
'''
```

Creating Functions

- Random Sampling
  - Uniform distribution
    - np.random.uniform(x,y,z)는 x이상부터 y미만인 z개를 uniform distribution random값으로 sampling

```
uniform_sampling = np.random.uniform(0, 1, (2,5))
print(uniform_sampling)
'''
[[0.26639967 0.62071191 0.67649403 0.45036727 0.01181673]
 [0.52986859 0.84493667 0.47515631 0.28743741 0.77223663]]
'''
```

  - Rand - Uniform distribution over $[0, 1)$

```
rand_sampling = np.random.rand(2,5)
print(rand_sampling)
'''
[[0.38000795 0.40664224 0.51519862 0.21076549 0.8430714 ]
 [0.67438271 0.16272181 0.1998186  0.9505862  0.76784629]]
'''
```

Creating Functions

- Random Sampling
  - Normal(Gaussian) distribution
    - np.random.normal(x,y,z)는 평균이 x이고 분산이 y인 z개를 normal distribution random값으로 sampling

```
normal_sampling = np.random.normal(0, 1, (2,5))
print(normal_sampling)
'''
[[-0.6013544   1.89544317  1.45838014 -0.48518027  1.11360633]
 [ 1.1276201  -0.99321183 -0.37415802 -2.25353429 -1.09036642]]
'''
```

  - Randn - Standard normal(gaussian) distribution of mean 0 and variance 1

```
randn_sampling = np.random.randn(2,5)
print(randn_sampling)
'''
[[ 0.92879391 -0.56569603 -0.30848259 -1.42508296 -0.92830748]
 [ 0.45703894 -0.24427407  0.45944466 -0.48407352 -0.3964146 ]]
'''
```

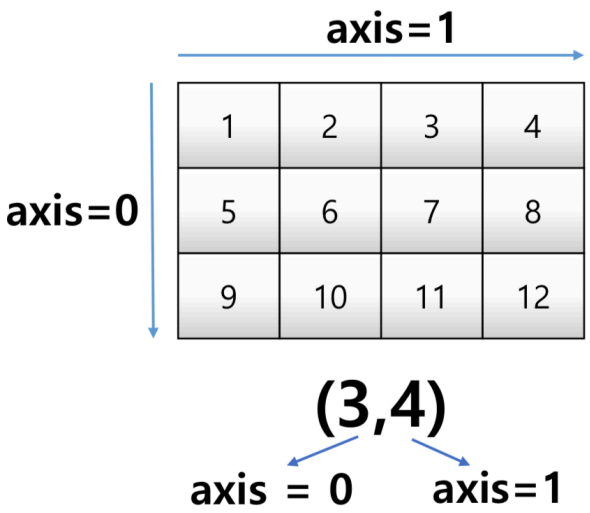
Operation Functions

- Sum

```
matrix = np.array([[1,3,5,7], [2,4,8,10]])
matrix_sum = matrix.sum()
print(matrix_sum)
'''
40
'''
```

Operation Functions

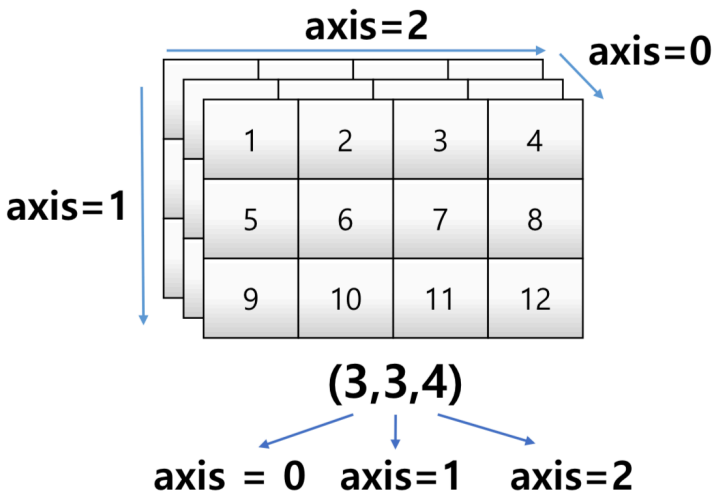
- Axis
  - Matrix




```
matrix = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

matrix_axis_0 = matrix.sum(axis=0)
print(matrix_axis_0)
'''
[15 18 21 24]
'''


matrix_axis_1 = matrix.sum(axis=1)
print(matrix_axis_1)
'''
[10 26 42]
'''


matrix_axis_2 = matrix.sum(axis=-1)
print(matrix_axis_2)
'''
[10 26 42]
'''
```

Operation Functions

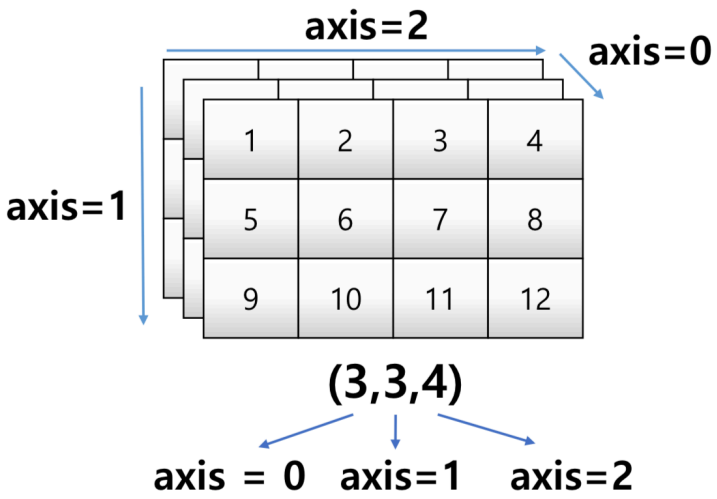
- Axis
  - Tensor




```
tensor = np.array([matrix, matrix, matrix])
print(third_order_tensor)
print(third_order_tensor.shape)
'''
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]]
(3, 3, 4)
'''
```

Operation Functions

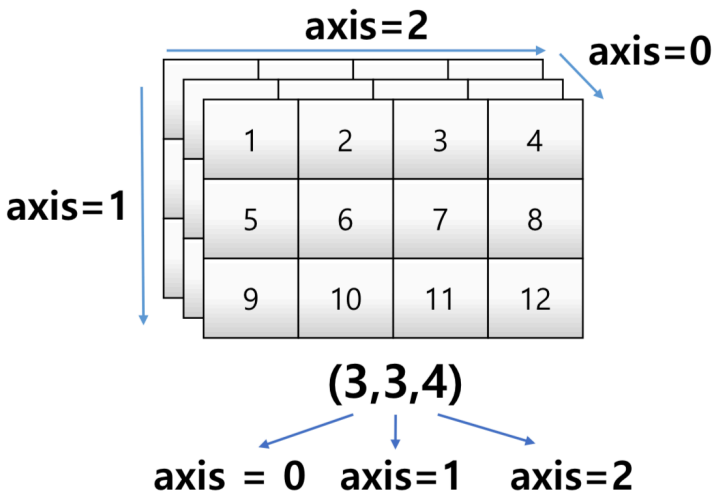
- Axis
  - Tensor




```
tensor_axis_0 = tensor.sum(axis=0)
print(tensor_axis_0)
'''
[[ 3  6  9 12]
 [15 18 21 24]
 [27 30 33 36]]
'''


tensor_axis_1 = tensor.sum(axis=1)
print(tensor_axis_1)
'''
[[15 18 21 24]
 [15 18 21 24]
 [15 18 21 24]]
'''
```

Operation Functions

- Axis
  - Tensor




```
tensor_axis_2 = tensor.sum(axis=2)
print(tensor_axis_2)
'''
[[10 26 42]
 [10 26 42]
 [10 26 42]]
'''

tensor_axis_3 = tensor.sum(axis=-1)
print(tensor_axis_3)
'''
[[10 26 42]
 [10 26 42]
 [10 26 42]]
'''
```

Operation Functions

- Mean & Std

```
matrix = np.array([[1,2,3,4], [5,6,7,8]])
mean = np.mean(matrix)
std = np.std(matrix)
print(mean)
print(std)
'''
4.5
2.29128784747792
'''
```

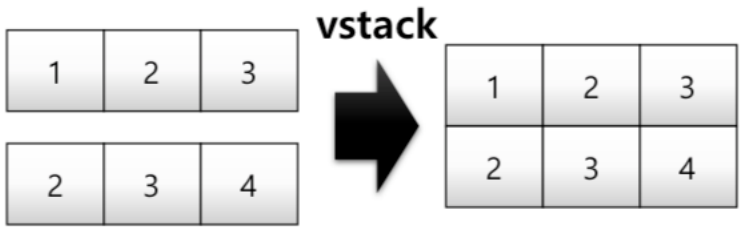
Operation Functions

- Mathematical functions
  - Exponential : exp(e^x), expm1($e^x - 1$), exp2(2^x), log($\log_e x$), log10($\log_{10} x$), log1p($\log_e x + 1$), log2($\log_2 x$), power(x^2), sqrt($\sqrt{x}$)
  - Trigonometric : sin, cos, tan, arcsin, arccos, arctan
  - Hyperbolic : sinh, cosh, tanh, arcsinh, arccosh, arctanh
  - 그 외에도 다양한 수학 연산자를 제공함 (np.something 호출)

```
matrix = np.array([[1,2,3,4], [5,6,7,8]])
exp = np.exp(matrix)
sqrt = np.sqrt(matrix)
print(exp)
print(sqrt)
'''
[[2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01]
 [1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03]]
[[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]]
'''
```

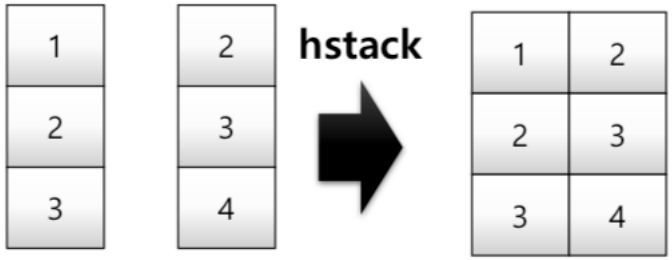
Operation Functions

- Vstack




```
a = np.array([1, 2, 3])
b = np.array([2, 3, 4])
vstack = np.vstack((a,b))
print(vstack)
'''
[[1 2 3]
 [2 3 4]]
'''
```

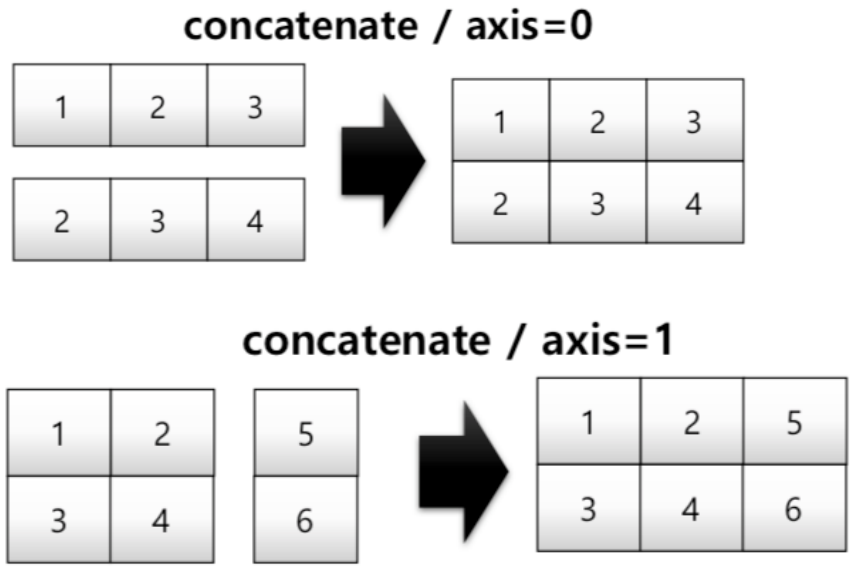
- Hstack




```
a = np.array([[1], [2], [3]])
b = np.array([[2], [3], [4]])
hstack = np.hstack((a,b))
print(hstack)
'''
[[1 2]
 [2 3]
 [3 4]]
'''
```

Operation Functions

- Concatenate



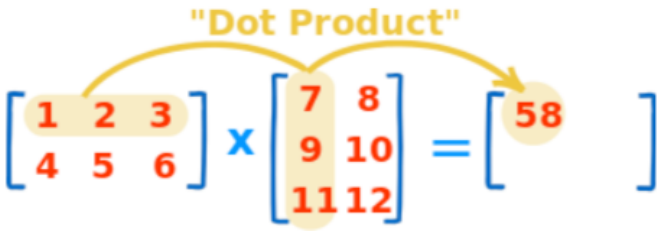

```
a = np.array([[1, 2, 3]])
b = np.array([[2, 3, 4]])

concat = np.concatenate((a,b)) # vstack
print(concat)
'''
[[1 2 3]
 [2 3 4]]
'''

concat_0 = np.concatenate((a,b), axis=0) # vstack
print(concat_0)
'''
[[1 2 3]
 [2 3 4]]
'''

concat_1 = np.concatenate((a,b), axis=1) # hstack
print(concat_1)
'''
[[1 2 3 2 3 4]]
'''

concat_none = np.concatenate((a,b), axis=None) # flatten
print(concat_none)
'''
[1 2 3 2 3 4]
'''
```

Operation Functions

- Element-wise operations

```
matrix = np.array([[1,2,3,4], [5,6,7,8]])
element_wise = matrix * matrix
print(element_wise)
'''
[[ 1  4  9 16]
 [25 36 49 64]]
'''
```

- Dot product




```
matrix = np.array([[1,2,3,4], [5,6,7,8]])

matrix_a = np.reshape(matrix, (4, 2))
print(matrix_a)
'''
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
'''

dot = np.dot(matrix, matrix_a)
print(dot)
'''
[[ 50  60]
 [114 140]]
'''
```

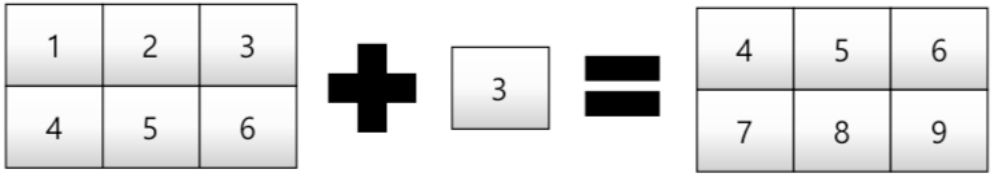
Operation Functions

- Transpose

```
matrix = np.array([[1,2,3,4], [5,6,7,8]])

transpose = np.transpose(matrix)
print(transpose)
'''
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
'''

transpose1 = matrix.transpose()
transpose2 = matrix.T
print(transpose1)
print(transpose2)
'''
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
'''
```

Operation Functions

- Broadcasting




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

print(matrix + scalar)        # add
'''
[[4 5 6]
 [7 8 9]]
'''

print(matrix - scalar)        # subtract
'''
[[-2 -1  0]
 [ 1  2  3]]
'''

print(matrix * scalar)        # multiply
'''
[[ 3  6  9]
 [12 15 18]]
'''

print(matrix / scalar)        # divide
'''
[[0.33333333 0.66666667 1.        ]
 [1.33333333 1.66666667 2.        ]]
'''
```

```
print(matrix // scalar)       # quotient
'''
[[0 0 1]
 [1 1 2]]
'''

print(matrix % scalar)        # remainder
'''
[[1 2 0]
 [1 2 0]]
'''

print(matrix ** scalar)       # power
'''
[[  1   8  27]
 [ 64 125 216]]
'''
```

Reference

- 머신러닝을 위한 Python 워밍업 (PDF Link)
- AI-python-connect (GitHub Link)

Thank you