원본 링크 : https://velog.io/@suminwooo/파이썬-Numpy-활용

Numpy 활용

(numpy에는 다양한 함수가 존재하지만, 기본적으로 알아두면 좋을 부분만 정리하였습니다. 중요하지만 복잡한 개념은 추가하지 않았습니다.)

Numpy 란?

- 넘파이는 Python에서 벡터, 행렬 등 수치 연산을 수행하는 선형대수 라이브러리입니다.
- 선형대수 관련 수치 연산을 지원하고 내부적으로는 C로 구현되어 있어 연산이 빠른 속도로 수행됩니다.

Numpy 사용 방법

- numpy 를 사용하기 위해서 다음과 같이 모듈을 import 합니다.
- 임포트를 할 때에는 numpy 라고 그대로 사용할 수 있지만, np 라는 축약된 이름을 관례적으로 많이 사용합니다.

```
import numpy as np # 설치시 pip install numpy
```

Numpy 함수 활용 및 예제

array 생성

- numpy의 주요 특징 중의 하나가 n차원 배열(ndarray) 객체이다.
- 수학식에서 행렬 연산과 비슷한 연산을 할 수 있다.

```
# array 생성
array1 = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(array1, type(array))
# [0 1 2 3 4 5 6 7 8 9] <class 'numpy.ndarray'>

array2 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print(array2)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]
```

크기 확인

- numpy에서는 shape를 통해 해당 array의 크기를 알 수 있다. (데이터 수& 몇 차원으로 존재하는지 등)

```
# 크기 확인
# array의 형태(크기)를 확인할 수 있다.
print(array1.shape)
# (10,)

print(array2.shape)
# (4, 3)
```

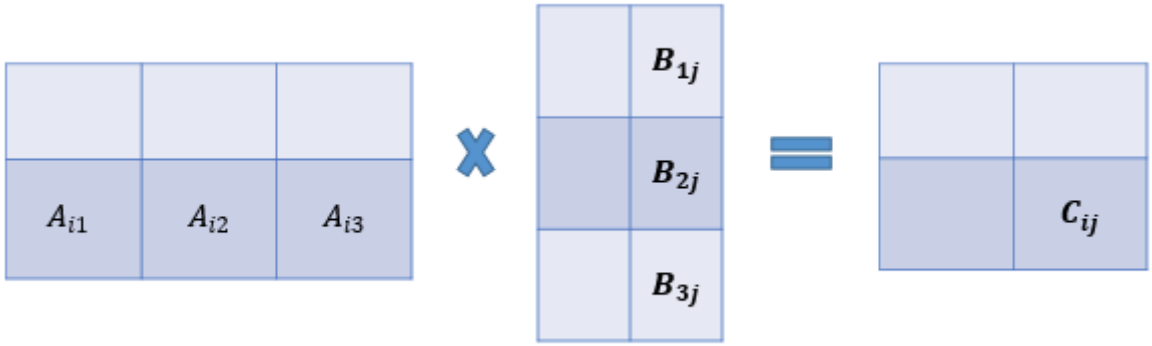
np.zeros(), np.ones(), np.arange() 함수

- numpy에서는 이전의 방법처럼 배열을 생성할 수 있지만, 0 또는 1로 이루어진 배열도 생성이 가능하다.

```
# np.zeros() 활용
np.zeros(10)
# array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

np.zeros((3,5))
# array([[0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.]])

# np.ones() 활용
np.ones(9)
# array([1., 1., 1., 1., 1., 1., 1., 1., 1.])

np.ones((2,10))
# array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
#        [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])


# np.arange() 활용
np.arange(10)
# array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange(3,10)
# array([3, 4, 5, 6, 7, 8, 9])
```

array 연산

- numpy에서 연산을 할때는 크기가 서로 동일한 array 끼리 연산이 진행된다. 이때 같은 위치에 있는 요소들 끼리 연산이 진행된다.
- 조심해야할 부분은 행렬의 곱처럼 진행되는 것(아래 이미지)이 아니라 각 요소별(동일한 위치)로 곱셈이 진행된다.




```
# 같은 array 계산
array3 = np.array([[1., 2., 3.], [4., 5., 6.]])
array3
# array([[1., 2., 3.],
#        [4., 5., 6.]])

array3 * array3
# array([[ 1.,  4.,  9.],
#        [16., 25., 36.]])

array3 - array3
# array([[0., 0., 0.],
#        [0., 0., 0.]])

1 / array3
# array([[1.        , 0.5       , 0.33333333],
#        [0.25      , 0.2       , 0.16666667]])

# 서로 다른 array 계산, 비교 가능
array4 = np.array([[0., 4., 1.], [7., 2., 12.]])
array4
# array([[ 0.,  4.,  1.],
#        [ 7.,  2., 12.]])

array3 + array4
# array([[ 1.,  6.,  4.],
#        [11.,  7., 18.]])

array3 < array4
# array([[False,  True, False],
#        [ True, False,  True]])
```

인덱싱, 슬라이싱

- numpy에서 사용되는 인덱싱은 리스트에서 사용하는 인덱싱과 동일하다.
- 리스트 인덱싱처럼 1번째로 시작하는 값의 인덱스는 0이다.
- 2차원의 경우 대괄호를 반복적으로 사용하여 접근할 수도 있고 쉼표를 이용하여 접근할 수도 있다.

```
# 1차원
array5 = np.arange(10)
array5
# array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

array5[5]
# 5

array5[5:8]
# array([5, 6, 7])

array5[5:8] = 10
array5
# array([ 0,  1,  2,  3,  4, 10, 10, 10,  8,  9])

# 2차원
array6 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array6[0]
# array([1, 2, 3])

array6[0][1]
# 2

array6[0, 1]
# 2
```

통계 함수

- 통계 함수를 통해 array의 합이나 평균등을 구할 수 있다.
- 또한 axis를 활용해 열 또는 행의 기준을 설정할 수 있다.

```
array8 = np.random.randn(7, 4)
# array([[-0.7005577 ,  0.05233897,  0.52168583],
#        [ 1.47013564, -0.26747237, -0.03893357]])

np.sum(array8)
# 1.0371967939458655

np.sum(array8, axis=1)
# array([-0.1265329,  1.1637297])

np.sum(array8, axis=0)
# array([ 0.76957794, -0.21513341,  0.48275226])

np.mean(array8)
# 0.17286613232431092

np.mean(array8, axis=0)
# array([ 0.38478897, -0.1075667 ,  0.24137613])
```

```
np.std(array8)
# 0.685654975739224

np.min(array8, axis=1)
# array([-0.7005577 , -0.26747237])

# 최소값 인덱스를 반환
np.argmin(array8)
# 0

# 최대값 인덱스를 반환
np.argmax(array8,axis=0)
# array([1, 0, 0])

# 누적합
np.cumsum(array8)
# array([-0.7005577 , -0.64821874, -0.1265329 ,  1.34360274,  1.07613037,
#         1.03719679])

# 누적곱
np.cumprod(array8)
# array([-7.00557700e-01, -3.66664650e-02, -1.91283752e-02, -2.81213063e-02,
#         7.52167255e-03, -2.92845581e-04])
```

다양한 함수

```
array7 = np.random.randn(5,3)
array7
# array([[ 0.71447189, -0.21040065,  0.4031654 ],
#        [ 1.23235593, -1.5653616 ,  1.01183471],
#        [ 0.04878855,  0.44906397, -0.58827852],
#        [ 1.07690984, -1.31534702,  0.5059711 ],
#        [ 0.76411248,  0.9710451 , -1.68417176]])

np.abs(array7)
# array([[0.71447189, 0.21040065, 0.4031654 ],
#        [1.23235593, 1.5653616 , 1.01183471],
#        [0.04878855, 0.44906397, 0.58827852],
#        [1.07690984, 1.31534702, 0.5059711 ],
#        [0.76411248, 0.9710451 , 1.68417176]])

np.sqrt(array7)
# array([[0.84526439,        nan, 0.63495307],
#        [1.11011528,        nan, 1.00589995],
#        [0.22088129, 0.67012235,        nan],
#        [1.03774266,        nan, 0.71131646],
#        [0.87413527, 0.98541621,        nan]])

np.square(array7)
# array([[5.10470083e-01, 4.42684345e-02, 1.62542340e-01],
#        [1.51870114e+00, 2.45035694e+00, 1.02380947e+00],
```

```
#        [2.38032218e-03, 2.01658445e-01, 3.46071619e-01],
#        [1.15973480e+00, 1.73013780e+00, 2.56006757e-01],
#        [5.83867879e-01, 9.42928584e-01, 2.83643451e+00]])

np.exp(array7)
# array([[2.04310741, 0.81025955, 1.4965544 ],
#        [3.42929922, 0.20901242, 2.75064301],
#        [1.0499983 , 1.56684488, 0.55528237],
#        [2.93559406, 0.26838117, 1.65859541],
#        [2.14708794, 2.64070281, 0.18559809]])

np.log(array7)
# array([[-0.33621162,         nan, -0.90840838],
#        [ 0.20892773,         nan,  0.01176522],
#        [-3.02025972, -0.80058994,         nan],
#        [ 0.07409568,         nan, -0.68127572],
#        [-0.26904028, -0.02938237,         nan]])

np.log10(array7)
# array([[-0.14601485,         nan, -0.39451675],
#        [ 0.09073616,         nan,  0.00510957],
#        [-1.31168213, -0.34769179,         nan],
#        [ 0.03217934,         nan, -0.29587429],
#        [-0.11684271, -0.0127606 ,         nan]])

np.log2(array7)
# array([[-0.48505084,         nan, -1.31055626],
#        [ 0.301419  ,         nan,  0.01697363],
#        [-4.35731371, -1.15500714,         nan],
#        [ 0.10689747,         nan, -0.9828731 ],
#        [-0.38814307, -0.04238979,         nan]])

np.sign(array7)
# array([[ 1., -1.,  1.],
#        [ 1., -1.,  1.],
#        [ 1.,  1., -1.],
#        [ 1., -1.,  1.],
#        [ 1.,  1., -1.]])

np.ceil(array7)
# array([[ 1., -0.,  1.],
#        [ 2., -1.,  2.],
#        [ 1.,  1., -0.],
#        [ 2., -1.,  1.],
#        [ 1.,  1., -1.]])

np.floor(array7)
# array([[ 0., -1.,  0.],
#        [ 1., -2.,  1.],
#        [ 0.,  0., -1.],
#        [ 1., -2.,  0.],
#        [ 0.,  0., -2.]])

np.isnan(array7)
```

```
# array([[False, False, False],
#        [False, False, False],
#        [False, False, False],
#        [False, False, False],
#        [False, False, False]])

np.isnan(np.log(array7))
# array([[False,  True, False],
#        [False,  True, False],
#        [False, False,  True],
#        [False,  True, False],
#        [False, False,  True]])

np.isinf(array7)
# array([[False, False, False],
#        [False, False, False],
#        [False, False, False],
#        [False, False, False],
#        [False, False, False]])

np.cos(array7)
# array([[ 0.75543937,  0.97794732,  0.91982372],
#        [ 0.33201636,  0.0054347 ,  0.53030614],
#        [ 0.99881007,  0.90085385,  0.83189721],
#        [ 0.4740515 ,  0.25268016,  0.87470424],
#        [ 0.72199671,  0.56443714, -0.1131327 ]])

np.tanh(array7)
# array([[ 0.61347338, -0.20734996,  0.38265414],
#        [ 0.84326133, -0.91628481,  0.7665198 ],
#        [ 0.04874987,  0.42112928, -0.52865637],
#        [ 0.79205034, -0.86562212,  0.46680014],
#        [ 0.64349291,  0.74916319, -0.93340082]])
```