머신 러닝·딥러닝에 필요한 수학 기초 with 파이썬

Matrix and Vector Calculation, Numpy

행렬과 벡터의 연산, Numpy

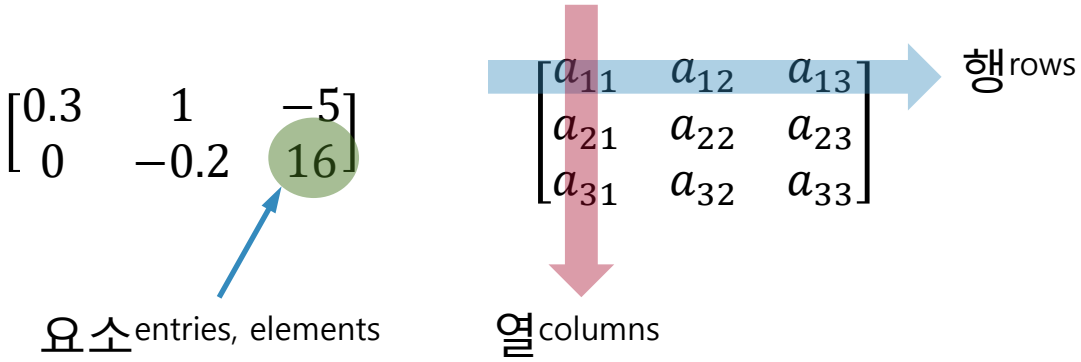
조준우
metamath@gmail.com

학습 개요

- 선형대수가 가장 중요?
  - 딥러닝: 한 벡터 공간을 다른 벡터 공간으로 매핑하기 위한 단순하고 연속된 기하학적 변환을 연결한 것(Chollet, François, Deep learning with python)
- 행렬과 벡터
  - 기본 정의
  - 연산
  - 행렬곱, 행렬-벡터곱의 해석
- 행렬을 이용한 데이터의 표현
  - 테이블형 데이터
  - 이미지
  - 문서
- 벡터화 코딩 실습

행렬Matrix

- 사각 괄호로 둘러 쌓인 숫자들의 배열(Rectangular array of numbers written between square brackets)



행렬Matrix

- 볼드 대문자 표시
- 행렬의 차원(크기) : 행 개수 × 열 개수

$$\mathbf{A} = [a_{jk}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- 보기의 경우 $m \times n$ 행렬
- 요소의 첫 번째 인덱스가 행 번호, 두 번째 인덱스가 열 번호

행렬Matrix

- 아래 행렬에서
- 행렬의 크기 2×3
- $a_{12} = 1$
- $a_{22} = -0.2$
- $a_{23} = 16$

$$\mathbf{A} = \begin{bmatrix} 0.3 & 1 & -5 \\ 0 & -0.2 & 16 \end{bmatrix}$$

행렬의 덧셈과 스칼라 곱셈

- 두 행렬 **A**, **B**의 크기가 같을 때 같은 위치의 요소의 덧셈

$$\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 10 & 12 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 5 & 1 \\ 3 & -2 \end{bmatrix} \qquad \mathbf{A} + \mathbf{B} = \begin{bmatrix} 8 & 2 \\ 13 & 10 \end{bmatrix}$$

- 크기가 다르면 덧셈 불가능

- 행렬에 대한 스칼라 곱은 각 요소에 스칼라를 곱하는 것

행렬 덧셈, 스칼라 곱 예

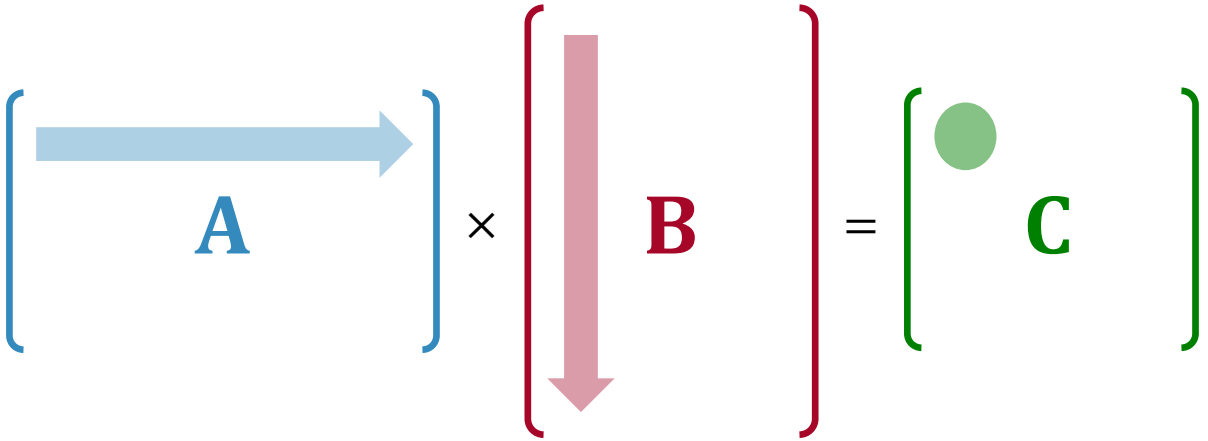
$$\begin{bmatrix} 8 & 2 \\ 16 & 10 \end{bmatrix} \times \frac{1}{2} + 3 \times \begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ 8 & 5 \end{bmatrix} + \begin{bmatrix} 3 & 3 \\ 9 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 4 \\ 17 & 17 \end{bmatrix}$$

```
A = np.asarray([8,2,16,10]).reshape(2,2)
B = np.asarray([1,1,3,4]).reshape(2,2)

print(A*(1/2) + 3*B)
```

```
[[ 7.  4.]
 [17. 17.]]
```

행렬-행렬의 곱셈

- 행과 열의 요소를 곱해서 더함.

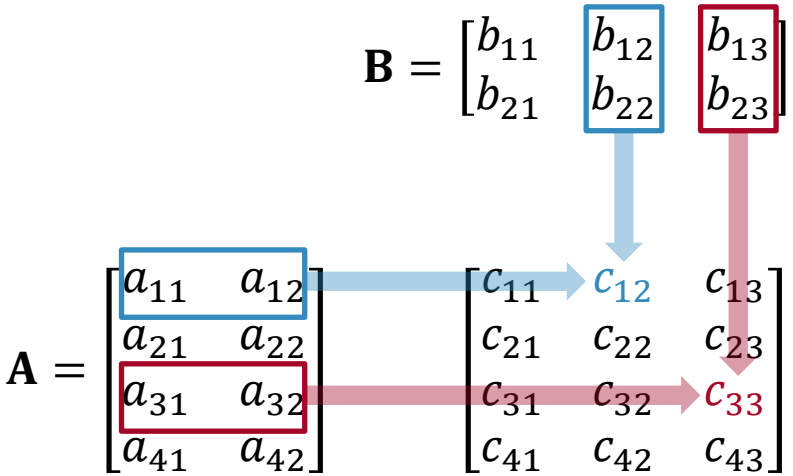


$m \times n$ matrix $\quad$ $n \times p$ matrix $\quad$ $m \times p$ matrix

$$= a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1n}b_{n1}$$

행렬-행렬의 곱셈

- 모든 행과 모든 열에 대해서 반복 계산




```
A = np.asarray([1, 2, 23, 43, 56,
32, 41, 55]).reshape(4,2)
B = np.asarray([11, 12, 13, 21, 22,
23]).reshape(2,3)
C = np.dot(A,B)
print(C)

[[  53   56   59]
 [1156 1222 1288]
 [1288 1376 1464]
 [1606 1702 1798]]
#반대로 곱할 수 있을까?
```

행렬의 전치Transpose

- 주어진 행렬의 행을 열로, 열을 행으로 이동
- $\mathbf{A}^{\mathrm{T}}$로 표시
- $(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$

$$A = \begin{bmatrix} 5 & -8 & 1 \\ 4 & 0 & 3 \end{bmatrix} \qquad A^{\mathrm{T}} = \begin{bmatrix} 5 & 4 \\ -8 & 0 \\ 1 & 3 \end{bmatrix}$$

```
A = np.asarray([5, -8, 1, 4, 0, 3]).reshape(2,3)
print(A.T)
```

```
[[ 5 4]
 [-8 0]
 [ 1 3]]
```

```
B = np.asarray([3, 1, 0, 1, 2, 8]).reshape(3,2)
print(np.dot(A,B).T)
```

```
[[17 18]
 [ 5 28]]
```

```
print(np.dot(B.T, A.T))
```

특별한 행렬

- 단위행렬unit, identity matrix
  - 대각 성분이 모두 1인 $n \times n$ 정사각 행렬, $\mathbf{I}_n$, $\mathbf{I}$
  - $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 대각행렬diagonal matrix
  - 대각 성분만 0이 아닌 성분을 가진 $n \times n$ 정사각 행렬

$$\mathbf{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 대칭행렬symmetric matrix
  - 정사각 행렬에 대해서 $\mathbf{S}^{\mathrm{T}} = \mathbf{S}$, 따라서 $s_{kj} = s_{jk}$인 행렬

$$\mathbf{S} = \begin{bmatrix} 10 & 20 & 100 \\ 20 & 10 & 120 \\ 100 & 120 & 20 \end{bmatrix}$$

역행렬Inverse

- $n \times n$ 행렬 $\mathbf{A}$의 역행렬은 $\mathbf{A}^{-1}$로 표시하며 다음을 만족

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

- 역행렬을 가지면 $\mathbf{A}$ 는 정칙행렬nonsingular matrix, 역행렬이 없으면 특이행렬 singular matrix

- 1차 연립 방정식의 해

$$\begin{aligned} 2x_1 + 1x_2 &= 3 \\ -6x_1 + 3x_2 &= -27 \end{aligned} \quad \rightarrow \quad \begin{bmatrix} 2 & 1 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -27 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 \\ -6 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 1 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -6 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ -27 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -6 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ -27 \end{bmatrix}$$

벡터Vector

- 행렬 관점에서 행이나 열을 하나만 가진 행렬
- 행렬의 크기가 $n \times 1$인 행렬
- 숫자 여러 개의 모임
- 행 벡터 또는 열 벡터라 하며 대부분 **별도의 언급이 없으면 벡터는 열 벡터**
- 볼드 소문자 표시
- 벡터의 차원 : 요소 개수, 보기의 경우 n차원 벡터

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

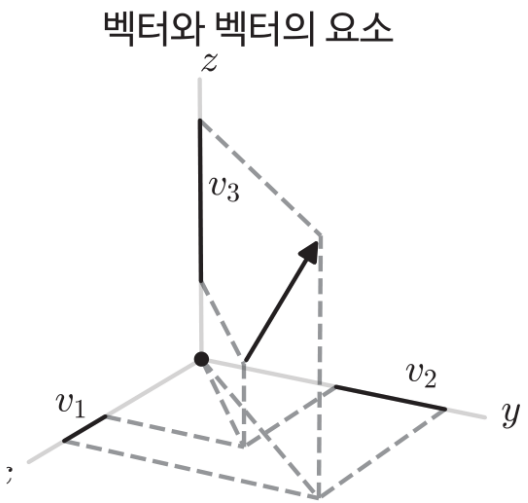
벡터Vector

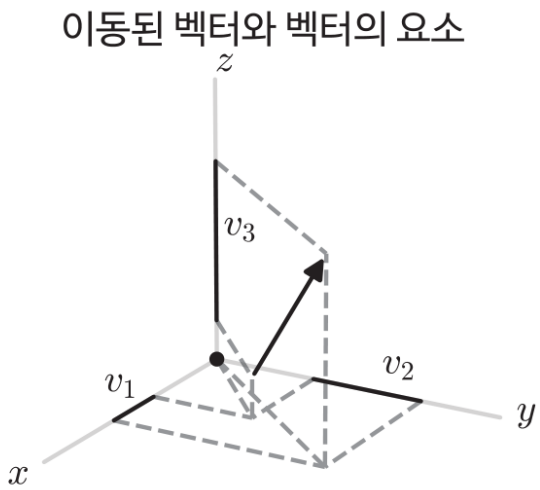
- 아래 벡터에서 벡터의 차원 4
- $v_1 = 23$
- $v_2 = 13$
- $v_4 = 100$

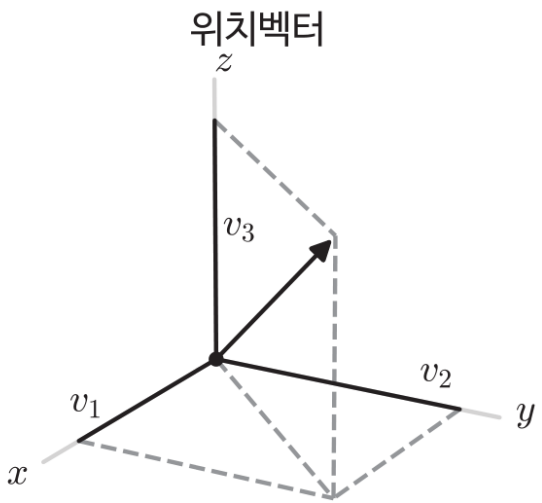
$$\mathbf{v} = \begin{bmatrix} 23 \\ 13 \\ 8 \\ 100 \end{bmatrix}$$

벡터Vector: 숫자 여러 개 모임?

- 3차원 공간의 화살표는 방향과 크기를 가짐
- 3차원 공간의 화살표: 숫자 세 개로 표현 가능

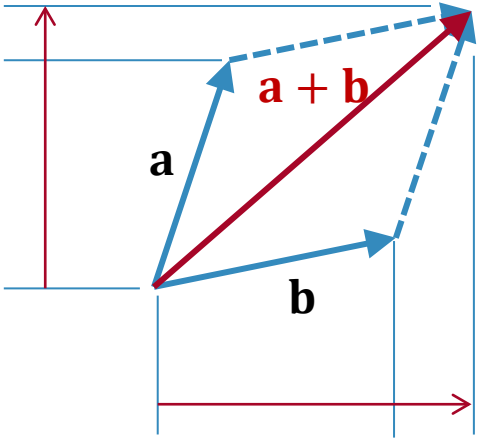


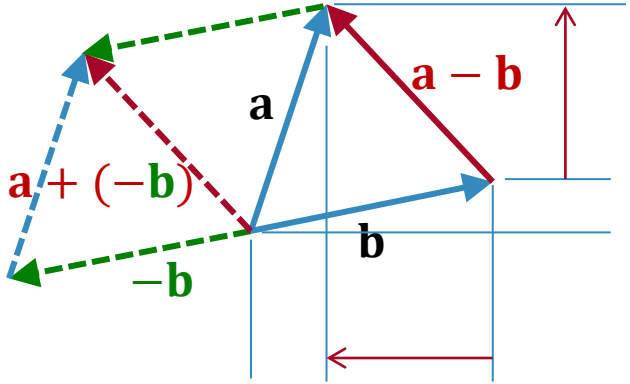




벡터의 덧셈, 뺄셈

- 벡터의 덧셈, 뺄셈은 요소끼리 덧셈, 뺄셈
- 덧셈 : 두 벡터가 이루는 평행 사변형으로 계산
- 뺄셈 : a-b = a+(-b)로 두고 덧셈과 동일하게 계산



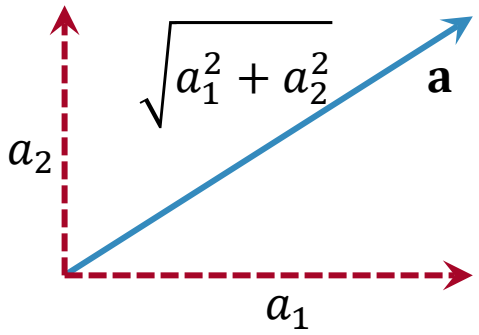


벡터의 크기(놈, 노름)Norm

- 2차원 벡터의 길이(크기) $|\mathbf{a}| = \sqrt{a_1^2 + a_2^2}$

- 노름 길이에 대한 일반적인 정의

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

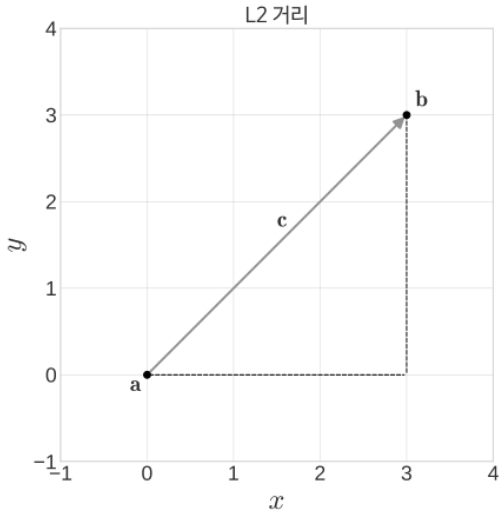


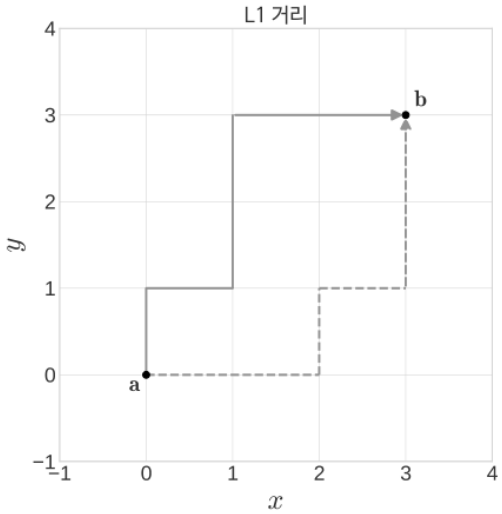
피타고라스 정리

벡터의 크기(놈, 노름)Norm

- L1 노름

$$\|\mathbf{x}\|_1 = \left(\sum_{i=1}^{n} |x_i|\right)$$



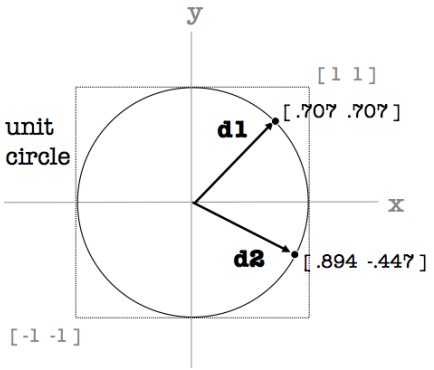


- L2 노름

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

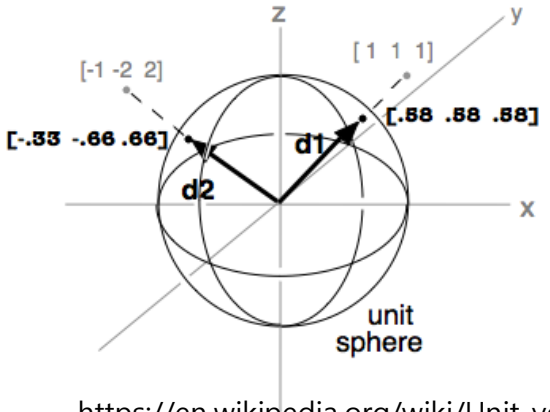
단위 벡터Unit vector

- 크기가 1인 벡터
- 벡터를 벡터의 크기로 나눔



$$\frac{(1\ \ 1)^{\mathrm{T}}}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}$$

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{|\mathbf{u}|}$$



$$\frac{(-1\ \ -2\ \ 2)^{\mathrm{T}}}{\sqrt{9}} = \begin{bmatrix} -\frac{1}{3} \\ -\frac{2}{3} \\ \frac{2}{3} \end{bmatrix}$$

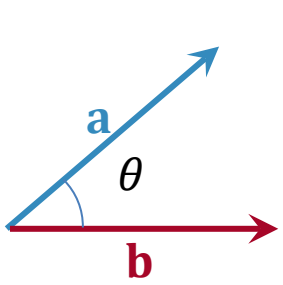
https://en.wikipedia.org/wiki/Unit_vector

벡터-벡터의 곱셈

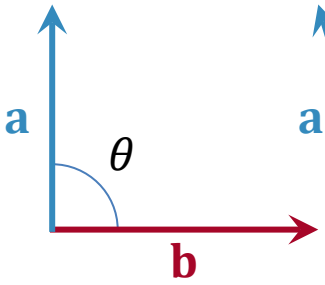
- 행과 열에 대해서 한번 계산
- 이런 연산을 벡터의 내적inner product 이라고도 함

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$\mathbf{a}^{\mathrm{T}} = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \rightarrow [c_1]$$

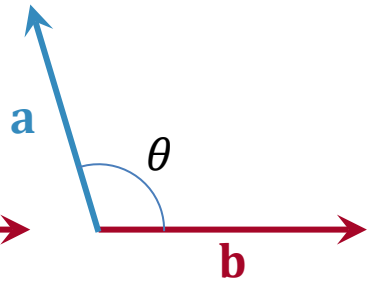


$\mathbf{a} \cdot \mathbf{b} > 0$



$\mathbf{a} \cdot \mathbf{b} = 0$

Orthogonality

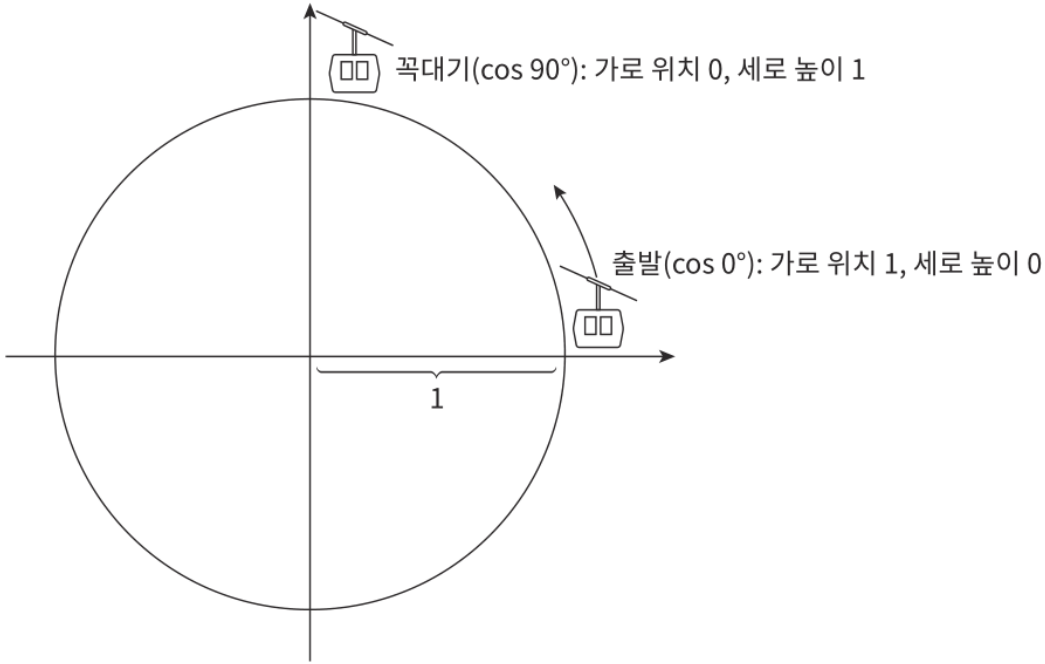


$\mathbf{a} \cdot \mathbf{b} < 0$

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} = \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$

cos: 가로함수, sin: 세로함수

- 대관람차의 위치를 가로, 세로로 나타낼 때
- 각도에 따른 가로 위치는 cos 함수, 세로 높이는 sin 함수



행렬을 이용한 데이터 표현: 테이블형데이터

- 신체 특징 데이터: 키, 몸무게, 혈압, ...
- 한 명 데이터를 벡터로 표시: $\mathbf{x} = (\text{키}, \text{몸무게}, \text{혈압}, \ldots)^T$
- 개별 데이터를 행 벡터로 저장
- 데이터 행렬 $\mathbf{X}$:(N,D), N: 데이터 개수, D: 데이터 차원

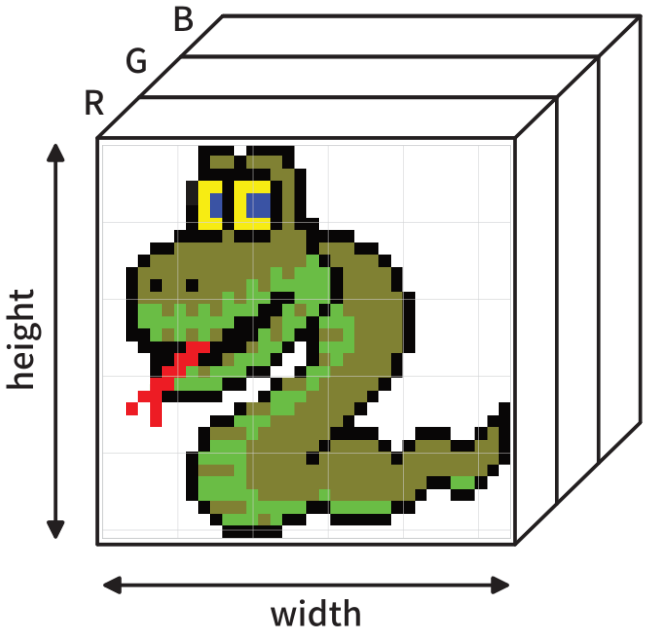
	키	몸무게	혈압	⋯
$\mathbf{x}_1$	170	64	90/100	...
$\mathbf{x}_2$	172	70	120/130	...
...	...	...	...	...
$\mathbf{x}_N$	182	82	100/120	...

개별 데이터

키 벡터

행렬을 이용한 데이터 표현: 이미지

- Channel first: (C, H, W), Channel last: (H, W, C)



행렬을 이용한 데이터 표현: 문서

```
docs = [
'This is the first document’,
'This is the second document',
'And the third one',
'Is this the first document']
```

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

```
V = ['<NULL>', 'and',
'document', 'first', 'is',
'one', 'second', 'the',
'third', 'this']
```

```
[
 [9] → [0,0,0,0,0,0,0,0,0,1]
 [4] → [0,0,0,0,1,0,0,0,0,0]
 [7] → [0,0,0,0,0,0,0,1,0,0]
 [3] → [0,0,0,1,0,0,0,0,0,0]
 [2] → [0,0,1,0,0,0,0,0,0,0]
]
```

인덱스Index

- 행렬, 벡터의 요소를 지정하기 위해 아래첨자로 사용하는 숫자
- 0부터 시작하는 방식과 1부터 시작하는 방식이 있음
- **수학** : 1부터 시작
- **컴퓨터 공학** : 0부터 시작
- 프로그램 언어

$$\mathbf{v} = \begin{bmatrix} 23 \\ 13 \\ 8 \\ 100 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

1부터 시작	0부터 시작
FORTRAN	Python
Julia	C/C++
MATLAB©	Java,
Octave	Javascript

Numpy

> Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB®, you might find this tutorial useful to get started with Numpy. – cs231n

ndarray

- numpy에서 제공하는 메인 객체인 배열
- 다차원 배열multidimensional array로써 1차원 배열(벡터), 2차원 배열(행렬), 3차원 배열(큐브), 4차원 배열(큐브가 여러 개 모인 것) 등등 계속 차원을 늘릴 수 있음
- 생성 : 행 우선 방식으로 [ ]로 적어주면 됨

```
import numpy as np

# 값을 지정하여 생성
A = np.array([[1,2],[3,4]])

# 랜덤으로 생성
E = np.random.rand(10)
F = np.random.rand(25).reshape(5,-1)
G = np.random.randint(1, 11, 9).reshape(3,3)
```

```
# 모든 요소가 0인 어레이
B = np.zeros(3,3)

# 모든 요소가 1인 어레이
C = np.ones(2,3)

# 대각 요소만 1인 어레이
D = np.eye(2)
```

Numpy 행렬-행렬의 곱셈

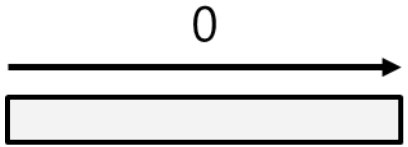
- Numpy array multiplication is not matrix multiplication

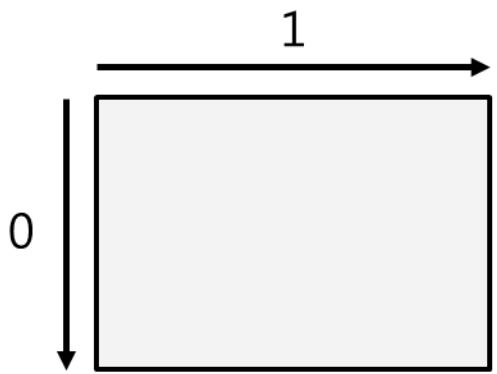
```
c = np.ones((3, 3))
C = np.matrix(c)

print("numpy array multiplication")
print(c*c)

print("numpy dot function")
print(c.dot(c))

print("numpy matrix multiplication")
print(C*C)
```

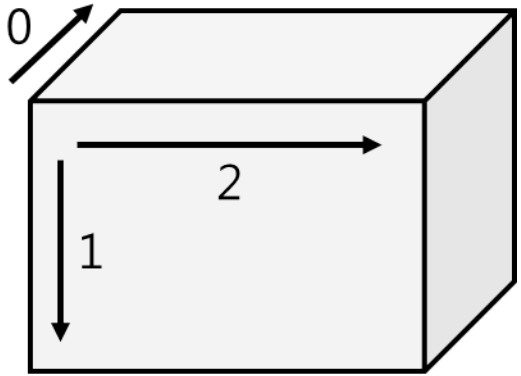
```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]]
```

axis

- 배열의 요소가 늘어선 방향
- 축에 번호를 붙여서 관리하는데 0부터 번호가 증가, 새롭게 생긴 axis가 0번







jupyter numpy.ipynb

기본 연산

- 기본 +, -, *, / 연산자를 그냥 쓰면 되고 요소별로 연산 됨

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

```
print(A+B)
[[ 6 10  7]
 [ 7  4  5]
 [ 9 11  5]]
```

```
print(A-B)
[[-4 -4  1]
 [ 3  2  3]
 [-1 -1 -1]]
```

```
print(A*B)
[[ 5 21 12]
 [10  3  4]
 [20 30  6]]
```

```
print(A/B)
[[0.2    0.4286 1.3333]
 [2.5    3.     4.    ]
 [0.8    0.8333 0.6667]]
```

인덱싱indexing

- 배열의 요소에 접근하기 위해서는 인덱스를 사용
- 인덱스는 0부터 시작
- 2축 이상에 대한 인덱싱
- 3행 3열 요소 C[2,2]

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 45 & 46 & 47 & 48 & 49 \end{bmatrix}$$

- 1행에서 3행까지~4열에서 5열까지 : 각 인덱스 자리에서 "시작인덱스:끝인덱스" 문법 적용 C[0:3, 3:5]

jupyter numpy.ipynb

어레이 인덱싱array indexing

- 인덱싱할 숫자를 임의의 어레이로 전달하는 방법
- 숫자를 지정할 자리에 숫자 대신 숫자 여러 개를 포함한 어레이를 전달하면 해당되는 요소 여러 개가 한번에 추출

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

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

# [6, 2]를 만들려면?
```

```
# 그냥 하나하나 지정해서 다 모음
np.array([A[1,2], A[0,1]])

# 첫번째 리스트에 행번호
# 두번째 리스트에 열번호
A[[1,0],[2,1]]
```

- 이 성질을 잘 이용하면 지정한 특정 요소에 대해서만 연산을 수행

어레이 인덱싱array indexing

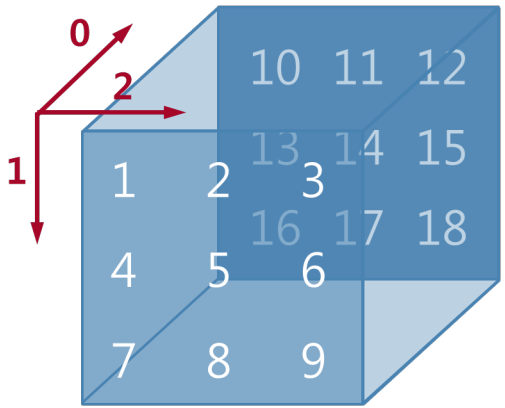
- Boolean 형 어레이를 전달

```
A = np.array([[1,2,3],[4,5,6],[7,8,9]])

A > 2
#결과
array([[False, False,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

```
print(A[A > 2])

#결과
array([[    1,     2,     3],
       [    4,     5,     6],
       [    7,     8,     9]])

array([[False, False,  True],
       [ True,  True,  True],
       [ True,  True,  True]])

array([3, 4, 5, 6, 7, 8, 9])
```

전치transpose

- `transpose(axis)` : 행렬의 전치와 같은 역할, 3개축 이상에서도 같은 논리로 동작




```
B = A[0,:]
print(B)
```

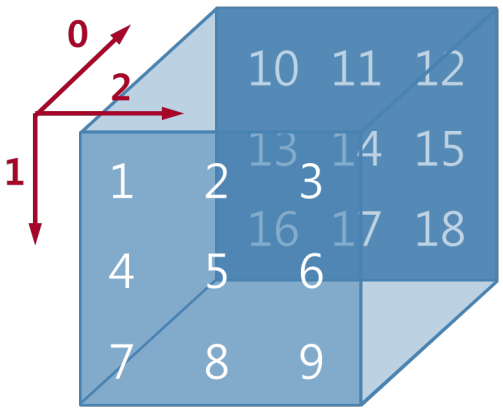
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
print(B.T)
```

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

전치transpose

- `transpose(axis)` : 행렬의 전치와 같은 역할, 3개축 이상에서도 같은 논리로 동작




```
print(A.transpose(2, 0, 1))

[[[ 1  4  7]
  [10 13 16]]

 [[ 2  5  8]
  [11 14 17]]

 [[ 3  6  9]
  [12 15 18]]]
```

브로드캐스팅broadcasting

- Numpy에서 shape가 다른 배열 간에도 산술 연산이 가능하게 하는 매커니즘

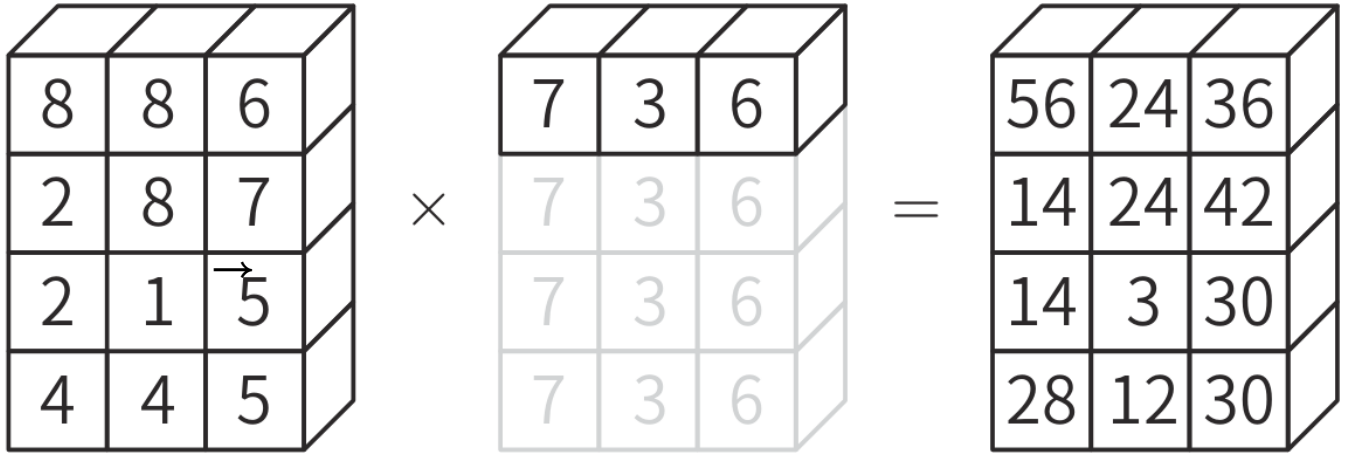
(3,)

(3,)　　(1,)

(3,)　　(3,)　　(3,)

jupyter numpy.ipynb

브로드캐스팅broadcasting



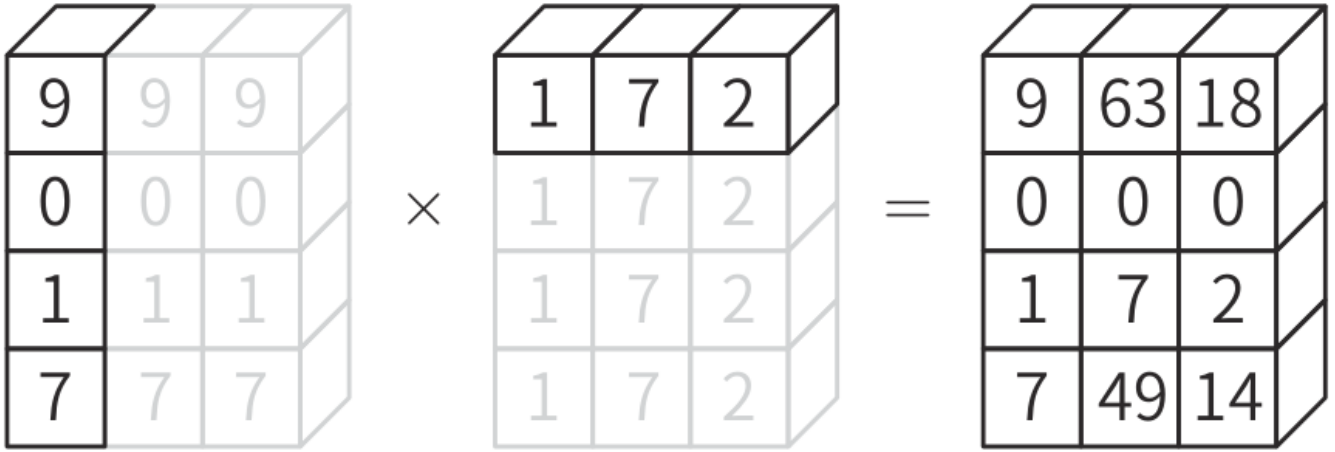
(4,3)
(4,3)
(4,3)

(3,)
(1,3)
(4,3)

(4,3)

jupyter numpy.ipynb

브로드캐스팅broadcasting



(4,1)
(4,1)
(4,3)

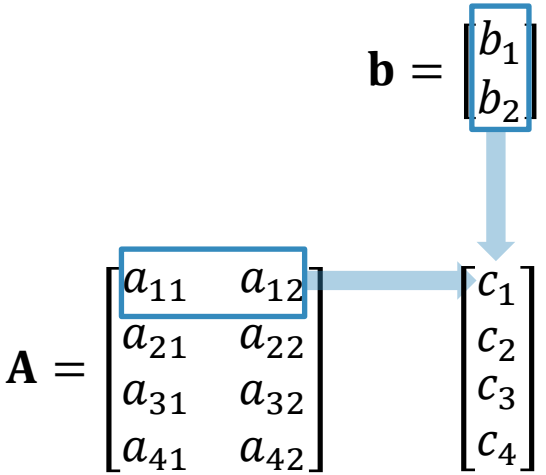
(3,)
(1,3)
(4,3)

(4,3)

jupyter numpy.ipynb

행렬-벡터의 곱셈

- 모든 행과 열에 대해서 반복 계산

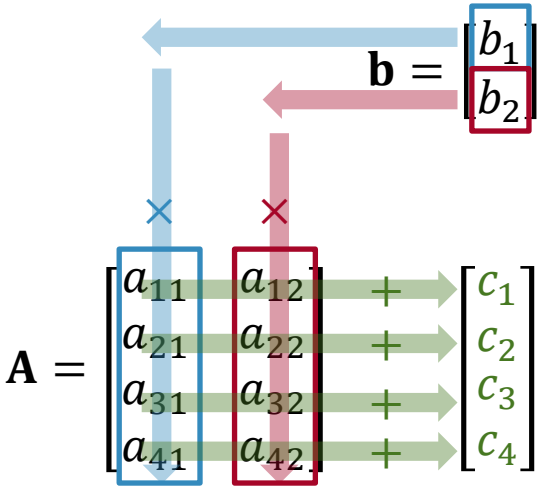



```
A = np.asarray([2, 2, 3, 8, 1, 1, 0,
1]).reshape(4,2)
b = np.asarray([3,5]).reshape(2,1)
C = np.dot(A,b)
print(C)

[[16]
 [49]
 [ 8]
 [ 5]]
#반대로 곱할 수 있을까?
```

행렬-벡터의 곱셈

- 모든 열의 선형결합linear combination
- 행렬과 행렬의 곱도 같은 작용의 반복

$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ matrix.ipynb




```
A = np.asarray([2, 2, 3, 8, 1, 1, 0,
1]).reshape(4,2)
b = np.asarray([3,5]).reshape(2,1)

A[:,0]*b[0] + A[:,1]*b[1]
(A*b.reshape(-1)).sum(axis=1)

array([16, 49, 8, 5])
```

행렬-행렬의 곱셈: 열조합

- 앞 행렬의 열을 조합하여 결과의 열을 계산
- 1열

$$\begin{bmatrix} 2 & 19 & 5 \\ 12 & 2 & 0 \\ 9 & 1 & 9 \\ 4 & 1 & 16 \end{bmatrix} \times \begin{bmatrix} 8 & 16 \\ 17 & 15 \\ 4 & 11 \end{bmatrix}$$

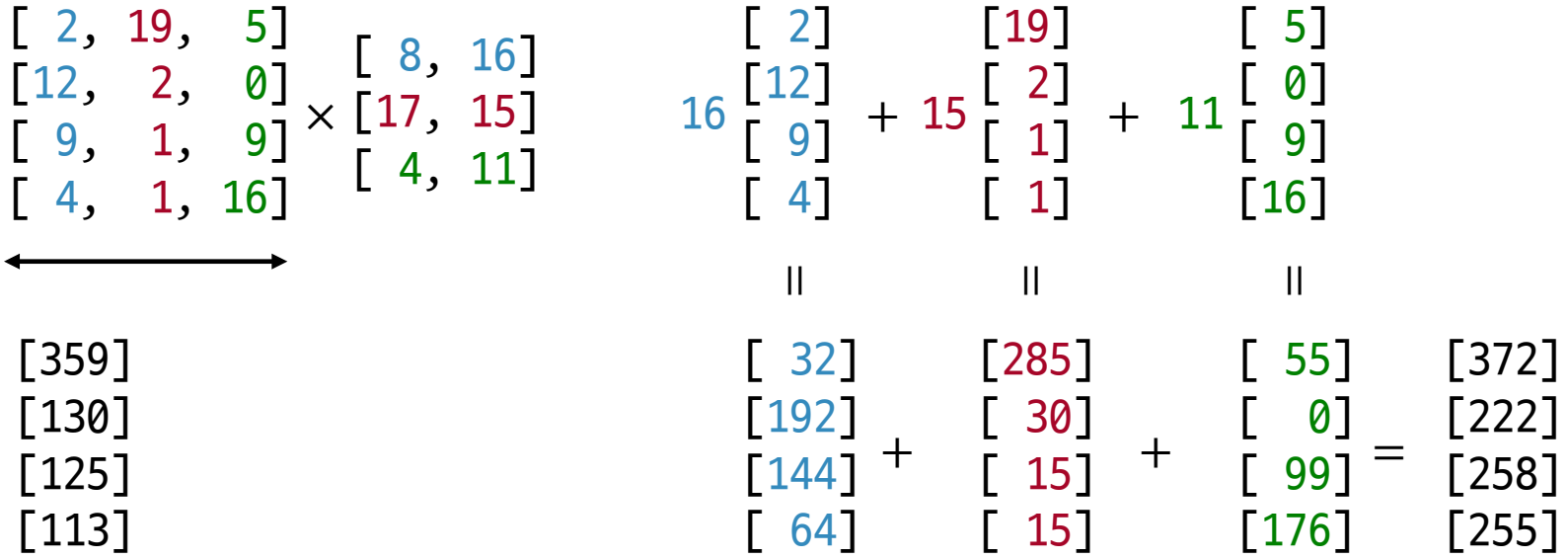
$$8\begin{bmatrix} 2 \\ 12 \\ 9 \\ 4 \end{bmatrix} + 17\begin{bmatrix} 19 \\ 2 \\ 1 \\ 1 \end{bmatrix} + 4\begin{bmatrix} 5 \\ 0 \\ 9 \\ 16 \end{bmatrix}$$

$$\| \qquad \| \qquad \|$$

$$\begin{bmatrix} 16 \\ 96 \\ 72 \\ 32 \end{bmatrix} + \begin{bmatrix} 323 \\ 34 \\ 17 \\ 17 \end{bmatrix} + \begin{bmatrix} 20 \\ 0 \\ 36 \\ 64 \end{bmatrix} = \begin{bmatrix} 359 \\ 130 \\ 125 \\ 113 \end{bmatrix}$$

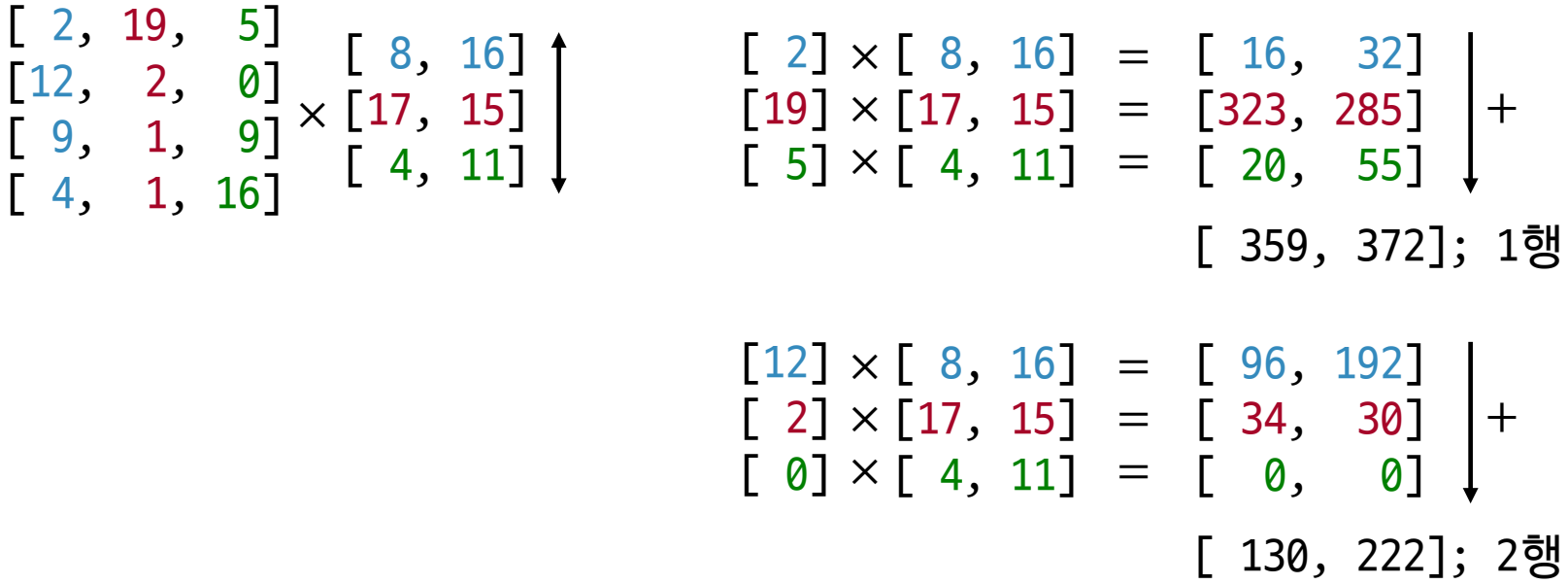
행렬-행렬의 곱셈: 열조합

- 2열



행렬-행렬의 곱셈: 행조합

- 뒤 형렬의 행을 조합하여 결과의 행을 계산



행렬 연산의 일반 성질

- 곱셈의 교환법칙이 성립하지 않는다.

$$\mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A}$$

$$(\mathbf{A}+\mathbf{B})+\mathbf{C}=\mathbf{A}+(\mathbf{B}+\mathbf{C})$$

$$\mathbf{A}+\mathbf{0}=\mathbf{A}$$

$$\mathbf{A}+(-\mathbf{A})=\mathbf{0}$$

$$c(\mathbf{A}+\mathbf{B})=c\mathbf{A}+c\mathbf{B}$$

$$(c+k)\mathbf{A}=c\mathbf{A}+k\mathbf{A}$$

$$(k\mathbf{A})\mathbf{B}=k(\mathbf{AB})$$

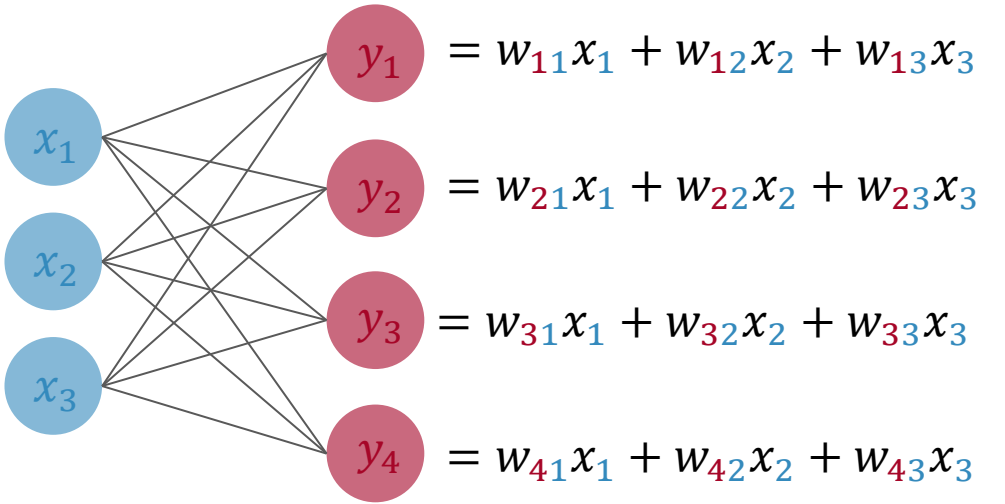
$$(\mathbf{AB})\mathbf{C}=\mathbf{A}(\mathbf{BC})$$

$$(\mathbf{A}+\mathbf{B})\mathbf{C}=\mathbf{AC}+\mathbf{BC}$$

$$\mathbf{C}(\mathbf{A}+\mathbf{B})=\mathbf{CA}+\mathbf{CB}$$

신경망에서 행렬-열우선

- 신경망의 각 레이어는 벡터
- 피드포워드 과정은 벡터에서 벡터로의 선형 변환 : 행렬곱 한번으로



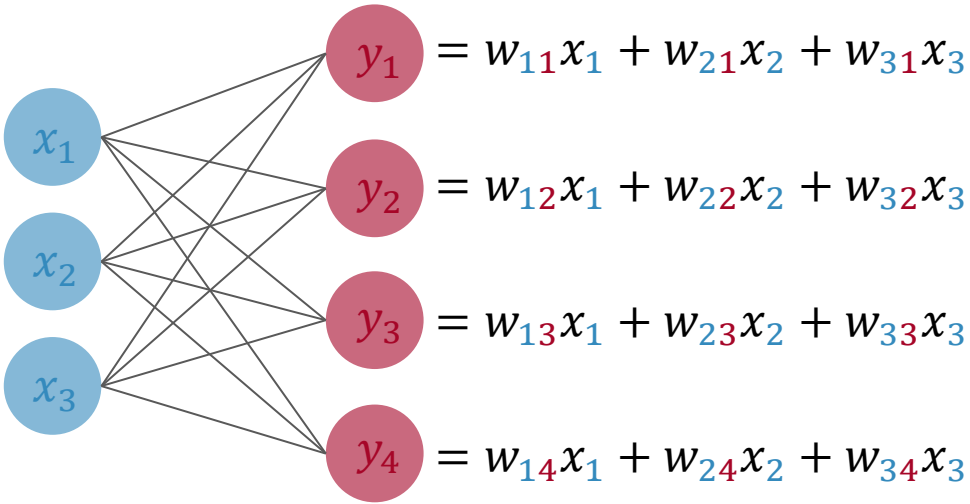
$$\begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}$$

$\mathbf{y}$	$\mathbf{W}$	$\mathbf{x}$
(4,1)	**(4,3)**	**(3,1)**

$f_1 : \mathbb{R}^3 \rightarrow \mathbb{R}^4$

신경망에서 행렬-행우선

- 신경망의 각 레이어는 벡터
- 피드포워드 과정은 벡터에서 벡터로의 선형 변환 : 행렬곱 한번으로



$$[y_1 \quad y_2 \quad y_3 \quad y_4] = [x_1 \quad x_2 \quad x_3] \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

$\mathbf{y}^T$	$\mathbf{x}^T$	$\mathbf{W}$
(1,4)	(1,3)	(3,4)

$f_1 : \mathbb{R}^3 \rightarrow \mathbb{R}^4$

행렬-벡터 연산

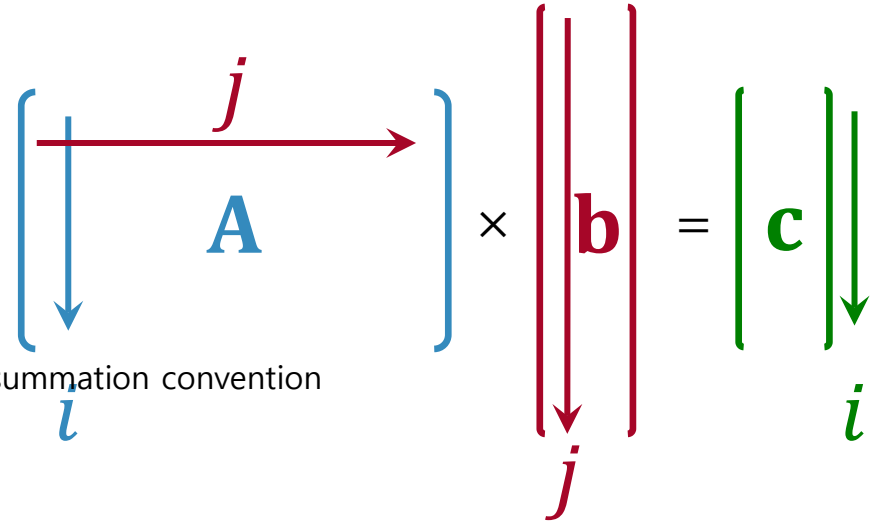
- **Python 구현 두 가지 방식**
  - For loop 방식 : 직관적이면서 구현이 쉬움
  - 벡터화vectorization 연산 : 덜 직관적이지만 효율적
- ipynb 파일로 몇 가지 예제를 실습

행렬 벡터 곱셈 표기

- 행렬과 벡터 곱셈 $\mathbf{c} = \mathbf{A}\mathbf{b}$

인덱스 표기

$$c_i = \sum_j A_{ij} b_j$$



아인슈타인 서메이션 컨벤션Einstein summation convention

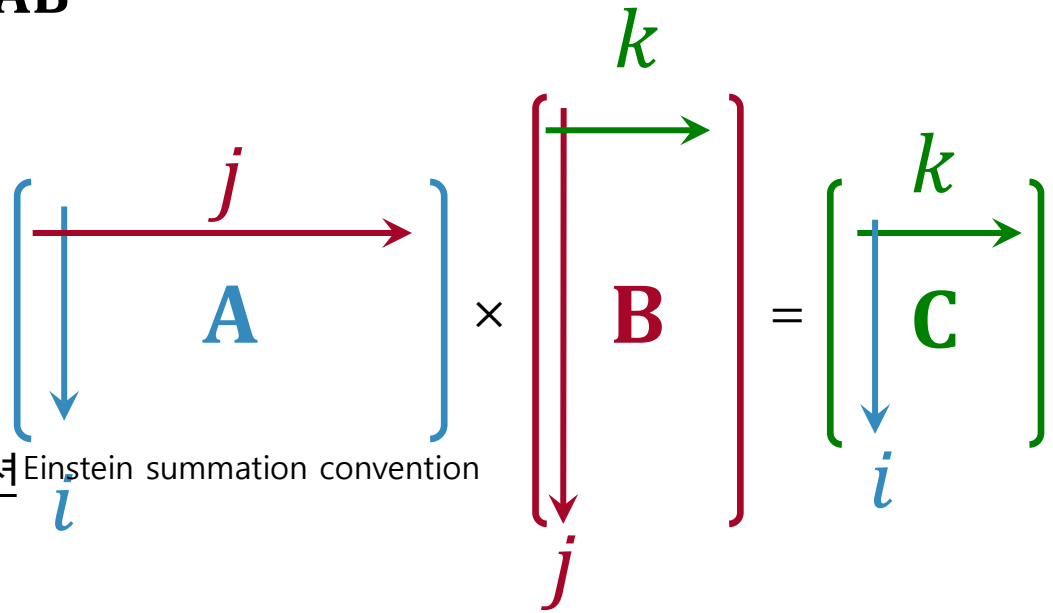
$$c_i = A_{ij} b_j$$

행렬 벡터 곱셈 표기

- 행렬과 벡터 곱셈 $\mathbf{C} = \mathbf{AB}$

인덱스 표기

$$C_{ik} = \sum_{j} A_{ij} B_{jk}$$



아인슈타인 서메이션 컨벤션 Einstein summation convention

$$C_{ik} = A_{ij} B_{jk}$$

행렬-벡터 연산 예(1)

- 주어진 식

$$y_i = \sum_{j=1}^{m} A_{ij} x_j$$

- For loop 방식

```
y = []
temp = 0
for i in range(A.shape[0]) :
    for j in range(A.shape[1]) :
        temp += A[i,j]*x[j]
    y.append(temp)
    temp = 0
```

vectorization

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

```
y = A.dot(x)
```

행렬-벡터 연산 예(1)

- 주어진 식

$$y_i = \sum_{j=1}^{m} A_{ji} x_j$$

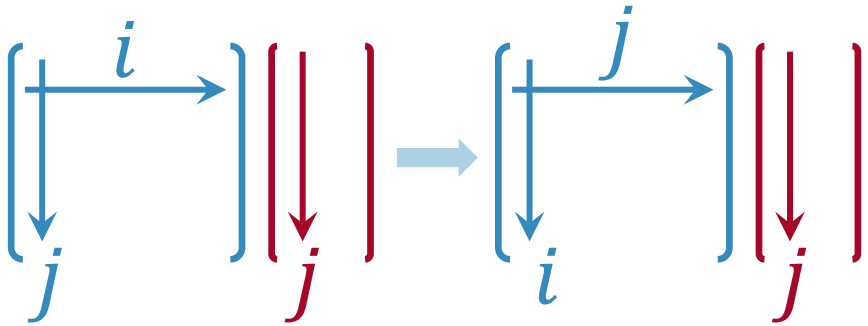
- For loop 방식

```
y = []
temp = 0
for i in range(A.shape[0]) :
    for j in range(A.shape[1]) :
        temp += A[j,i]*x[j]
    y.append(temp)
    temp = 0
```

vectorization

$$\mathbf{y} = \mathbf{A}^{\mathrm{T}}\mathbf{x}$$

```
y = A.T.dot(x)
```



행렬-벡터 연산 예(2)

- 주어진 식

$$\mathbf{S} = \sum_{i=1}^{m} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}$$

$$\mathbf{x}^{(1)} = [1 \quad 2 \quad 3]^{\mathrm{T}}$$
$$\mathbf{x}^{(2)} = [4 \quad 5 \quad 6]^{\mathrm{T}}$$
$$\mathbf{x}^{(3)} = [7 \quad 8 \quad 9]^{\mathrm{T}}$$
$$\mathbf{x}^{(4)} = [10 \quad 11 \quad 12]^{\mathrm{T}}$$
$$\mathbf{x}^{(5)} = [13 \quad 14 \quad 15]^{\mathrm{T}}$$

$\mathbf{x}^{(1)}\left(\mathbf{x}^{(1)}\right)^{\mathrm{T}}$ $\mathbf{x}^{(2)}\left(\mathbf{x}^{(2)}\right)^{\mathrm{T}}$ $\mathbf{x}^{(3)}\left(\mathbf{x}^{(4)}\right)^{\mathrm{T}}$

$$\mathbf{S} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix} + \begin{bmatrix} 16 & 20 & 24 \\ 20 & 25 & 30 \\ 24 & 30 & 36 \end{bmatrix} + \begin{bmatrix} 49 & 56 & 63 \\ 56 & 64 & 72 \\ 63 & 72 & 81 \end{bmatrix}$$

$$+ \begin{bmatrix} 100 & 110 & 120 \\ 110 & 121 & 132 \\ 120 & 132 & 144 \end{bmatrix} + \begin{bmatrix} 169 & 182 & 195 \\ 182 & 196 & 210 \\ 195 & 210 & 225 \end{bmatrix}$$

$\mathbf{x}^{(4)}\left(\mathbf{x}^{(4)}\right)^{\mathrm{T}}$ $\mathbf{x}^{(5)}\left(\mathbf{x}^{(5)}\right)^{\mathrm{T}}$

행렬-벡터 연산 for loop

- 주어진 식

$$\mathbf{S} = \sum_{i=1}^{m} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}$$

```
X = [x1, x2, x3, x4, x5]
m = len(X)
S = np.zeros((x1.shape[0], x1.shape[0]), dtype=int)

for i in range(m):
    x_i = X[i]
    S += np.dot(x_i, x_i.T)
```

행렬-벡터 연산 vectorization

- 주어진 식

$$\mathbf{S} = \sum_{i=1}^{m} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}$$

$$\mathbf{X}^{\mathrm{T}} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \mathbf{x}^{(3)} & \mathbf{x}^{(4)} & \mathbf{x}^{(5)} \\ \begin{bmatrix}1\\2\\3\end{bmatrix} & \begin{bmatrix}4\\5\\6\end{bmatrix} & \begin{bmatrix}7\\8\\9\end{bmatrix} & \begin{bmatrix}10\\11\\12\end{bmatrix} & \begin{bmatrix}13\\14\\15\end{bmatrix} \end{bmatrix}_{(3 \times 5)} \times \begin{bmatrix} \mathbf{x}^{(1)} = [1 \quad 2 \quad 3] \\ \mathbf{x}^{(2)} = [4 \quad 5 \quad 6] \\ \mathbf{x}^{(3)} = [7 \quad 8 \quad 9] \\ \mathbf{x}^{(4)} = [10 \quad 11 \quad 12] \\ \mathbf{x}^{(5)} = [13 \quad 14 \quad 15] \end{bmatrix}_{(5 \times 3)} = \mathbf{X}$$

```
X = np.array([x1, x2, x3, x4, x5]).reshape(5,3)      # row major
S = np.dot(X.T, X)
```