벡터 매트릭스 Numpy

명지대학교
MYONGJI UNIVERSITY

벡터와 스칼라

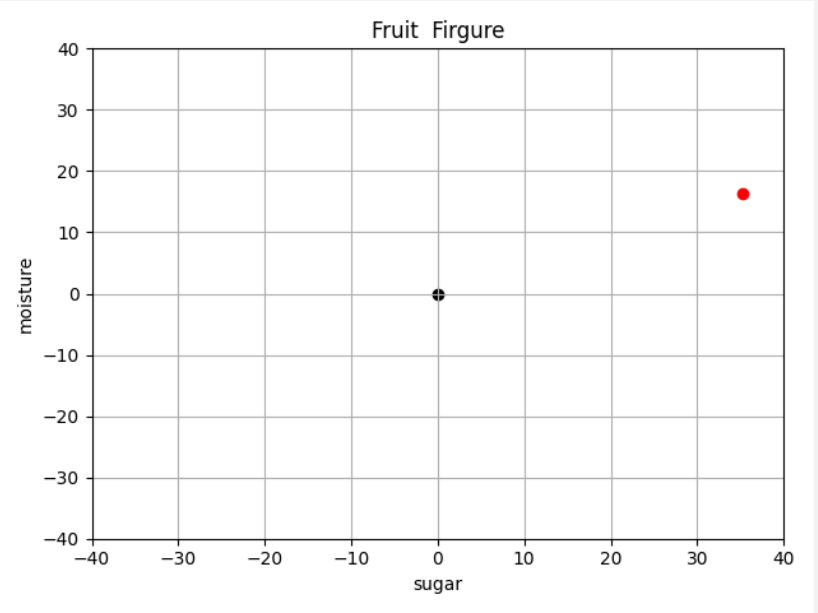
벡터

- 직선,벡터 : 선형대수학에서의 기본 조각
- 흔히들 벡터는 $\vec{a}$나 $\vec{x}$처럼 알파벳 위에 화살표를 표시
- 물리학시간에 벡터는 "크기와 방향이 있는 물리량"
- 과일이. $\begin{bmatrix} 35.2 \\ 16.3 \end{bmatrix}$ 과 같은 숫자의 리스트로 표현

```
# 딕셔너리로 표현
과일0 = {"당도":35.2, "수분함유량": 16.3}

#. 리스트로 표현
과일0 = [35.2, 16.3]
```

벡터위치

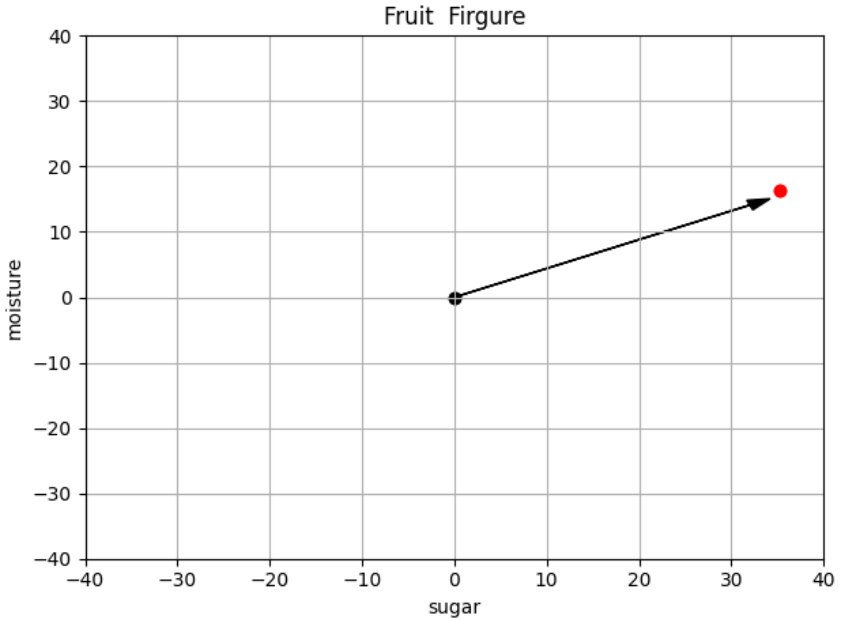
```
import matplotlib.pyplot as plt

#collapse-hide
plt.scatter(35.2, 16.3, c='r')
plt.scatter(0, 0, c='black')
plt.xlim(-40, 40)
plt.ylim(-40, 40)

plt.grid(True)
plt.xlabel('sugar')
plt.ylabel('moisture')
plt.title('Fruit   Firgure')

plt.scatter(35.2, 16.3, c='r')
plt.scatter(0, 0, c='black')
```



2차원 벡터

```
head_width = 1.6
width = 0.001
plt.arrow(0, 0, 35.2 - head_width * 2, 16.3 - head_width * 2 + 1,
          color = 'black',
          width=width,
          head_width=head_width)

plt.xlim(-40, 40)
plt.ylim(-40, 40)

plt.grid(True)
plt.xlabel('sugar')
plt.ylabel('moisture')
plt.title('Fruit  Firgure')
```

$\begin{bmatrix} 35.2 \\ 16.3 \end{bmatrix}$ 리스트는 공간상에서 화살표

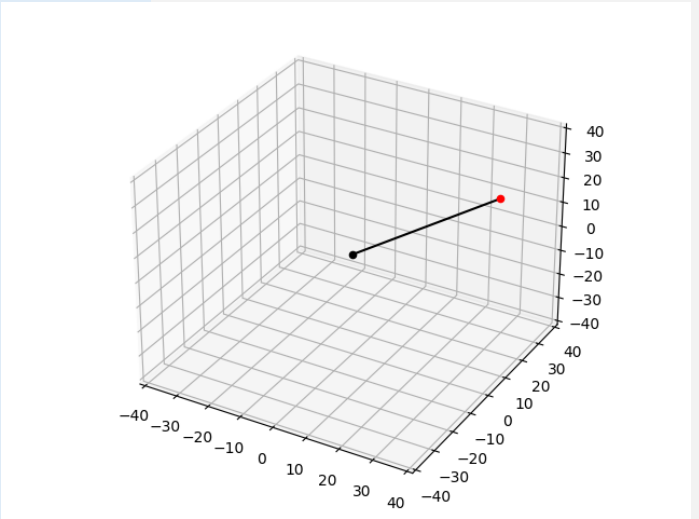


3차원 벡터

$$\begin{bmatrix} 35.2 \\ 16.3 \\ 25.1 \end{bmatrix}$$

3개의 열을 가지고 있다면 아래와 같이 나타낼 수 있음

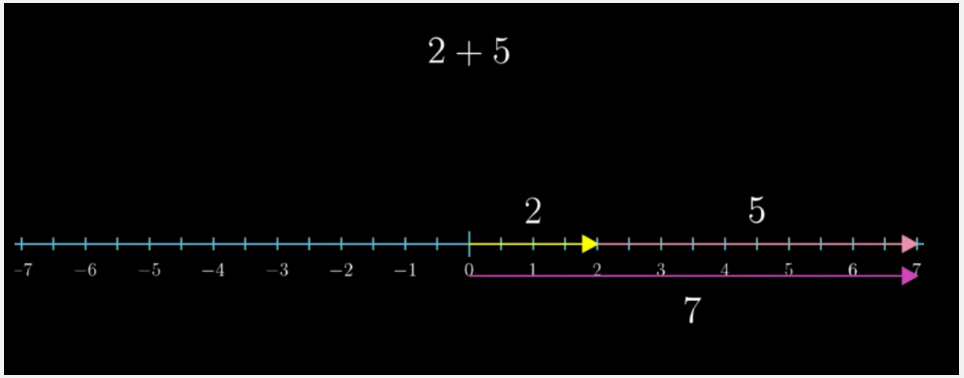
```
#collapse-hide
ax = plt.figure().gca(projection='3d')

ax.scatter(35.2, 16.3, 25.1, c='r')
ax.scatter(0, 0, 0, c='black')

head_width = 1.6
width = 0.001
ax.plot([0, 35.2], [0, 16.3], [0, 25.1], c='black')
ax.grid(True)
ax.set_xlim([-40, 40])
ax.set_ylim([-40, 40])
ax.set_zlim([-40, 40])
plt.show()
plt.close()
```



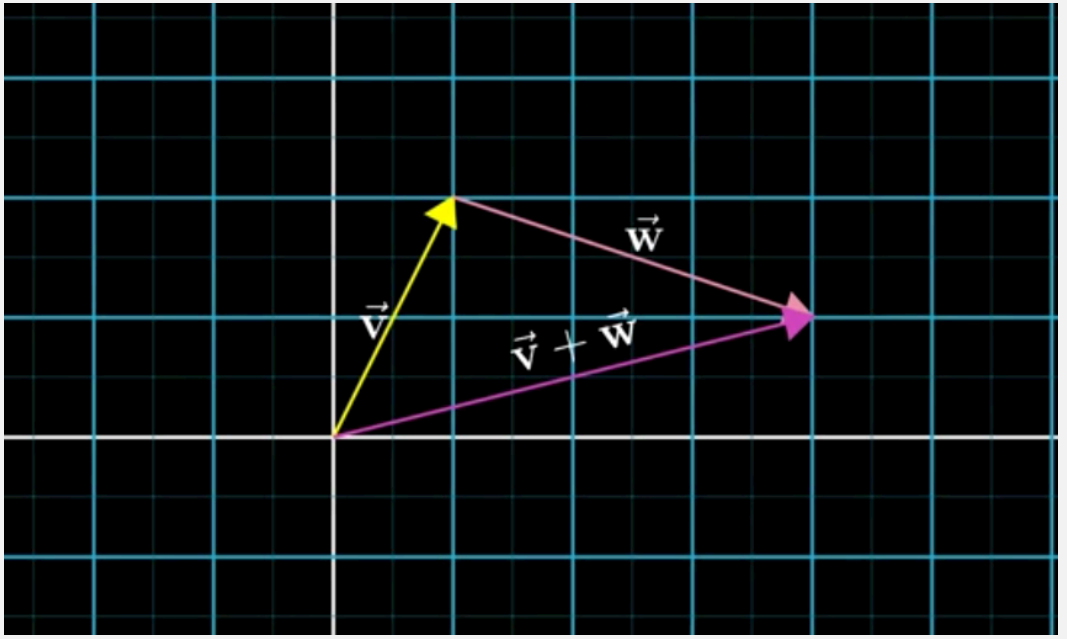
벡터의 두가지 기본연산 : 벡터합과 스칼라곱

벡터 합 : 두 벡터를 이어 붙이기

- 1차원 좌표 평면

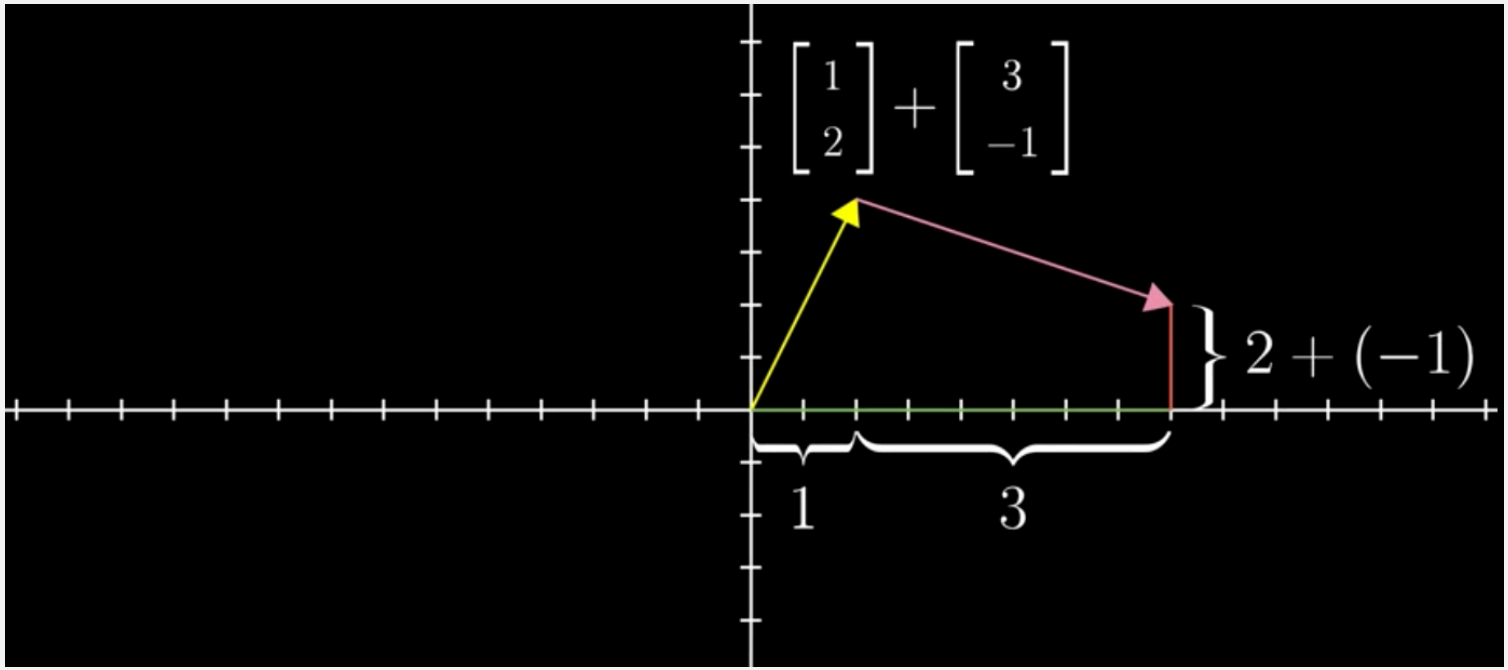


- 2차원 좌표 평면



벡터의 두가지 기본연산 : 벡터합과 스칼라곱

벡터 합 : 두 벡터를 이어 붙이기



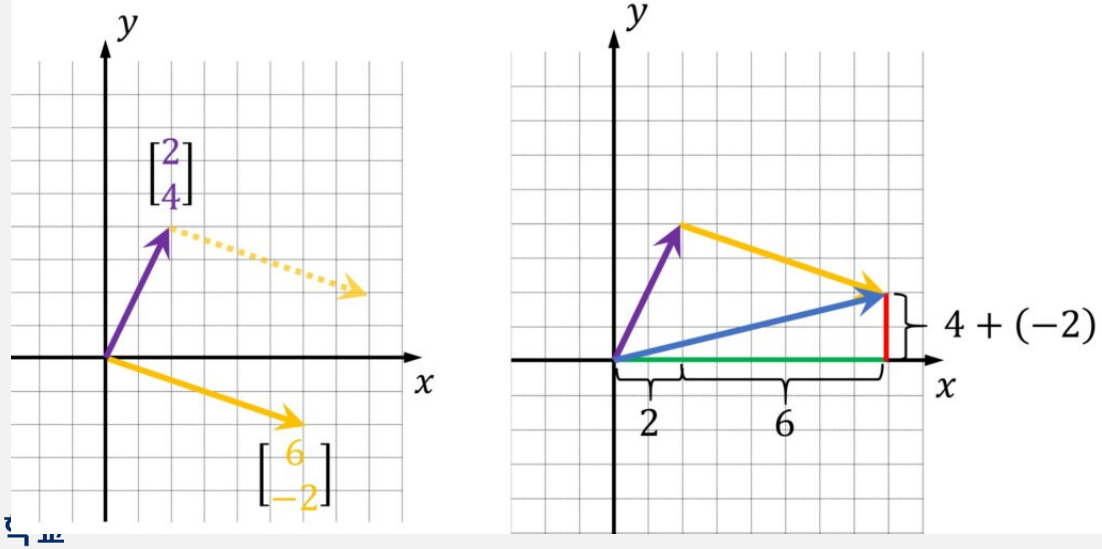
https://i.imgur.com/z6kNBro.png
https://i.imgur.com/JAhAn2b.png
https://i.imgur.com/DO5gFki.png

벡터의 두가지 기본연산 : 벡터합과 스칼라곱

- 즉, $\vec{v}$를 통해 이동했다가 그 위치에서 $\vec{w}$의 방향으로 그 크기만큼 이동한다면 결과적으로 두 벡터의 합 벡터인 $\vec{v}+\vec{w}$의 끝 지점에 위치

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix} + \begin{bmatrix} 6 \\ -2 \end{bmatrix} = \begin{bmatrix} 2+6 \\ 4+(-2) \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \end{bmatrix}$$



Numpy

Numpy vs python list

```
import numpy as np

npvec_v = np.array([1, 2])
npvec_w = np.array([3, -1])

npvec_addiation = npvec_v + npvec_w

npvec_addiation
```

```
pyvec_v = [1, 2]
pyvec_w = [3, -1]

pyvec_addition = pyvec_v + pyvec_w

pyvec_addition
```

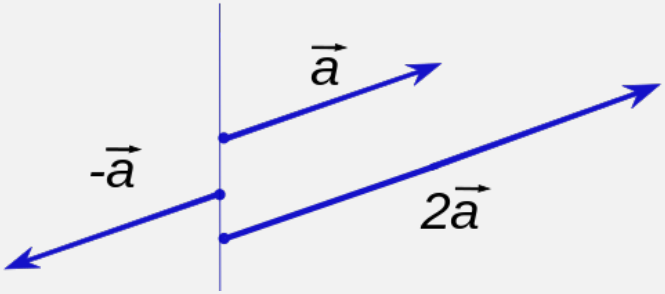
스칼라 곱 : 벡터를 상수배 만큼 늘리거나 줄이기

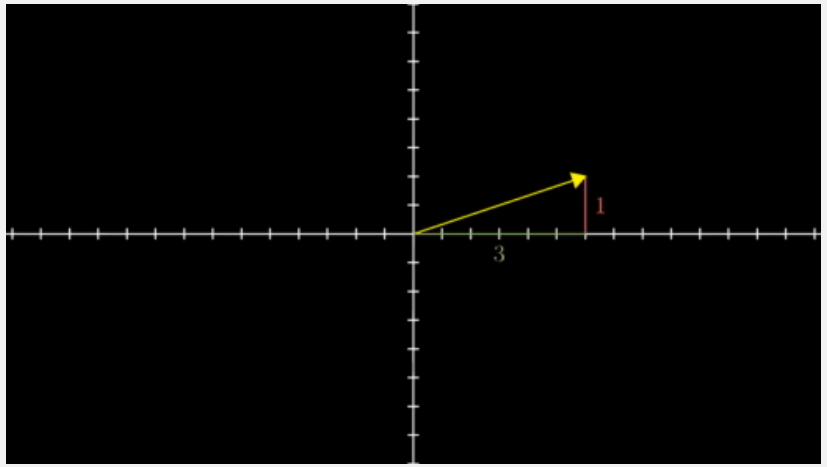
스칼라곱

- 벡터의 방향을 바꾸지 않고 그 크기만 상수배 만큼 늘리거나 줄이는 것

- 물론 음수를 곱하게 되면 방향이 뒤바뀌게 되지만 절대적으로 x와 y사이의 비율, 즉 기울기는 그대로 유지

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{*3=} \begin{bmatrix} -3 \\ -6 \end{bmatrix}$$





출처 : https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Scalar_multiplication_of_vectors2.svg/500px-Scalar_multiplication_of_vectors2.svg.png
출처 :https://github.com/gusdnd852/bigdata-lecture/blob/master/_notebooks/img/Day1/30.gif?raw=true

스칼라 곱 : 벡터를 상수배 만큼 늘리거나 줄이기

Numpy

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

npvec_multiplication = npvec_v * scalar

npvec_multiplication
```

배열의 곱

list

```
pyvec_v = [3, 1]
scalar = 2

pyvec_multiplication = pyvec_v * scalar

pyvec_multiplication
```

파이썬리스트의 곱셈 연산은 해당 리스트를 n배로 복제해 낸 원소를 가지고 있는 리스트를 새로 만드는 연산

내적

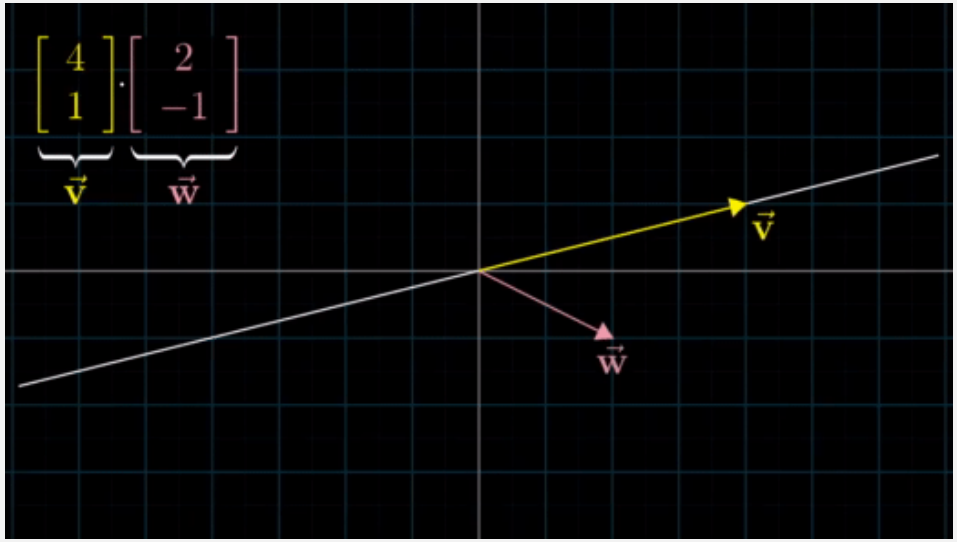
벡터와 벡터의 곱

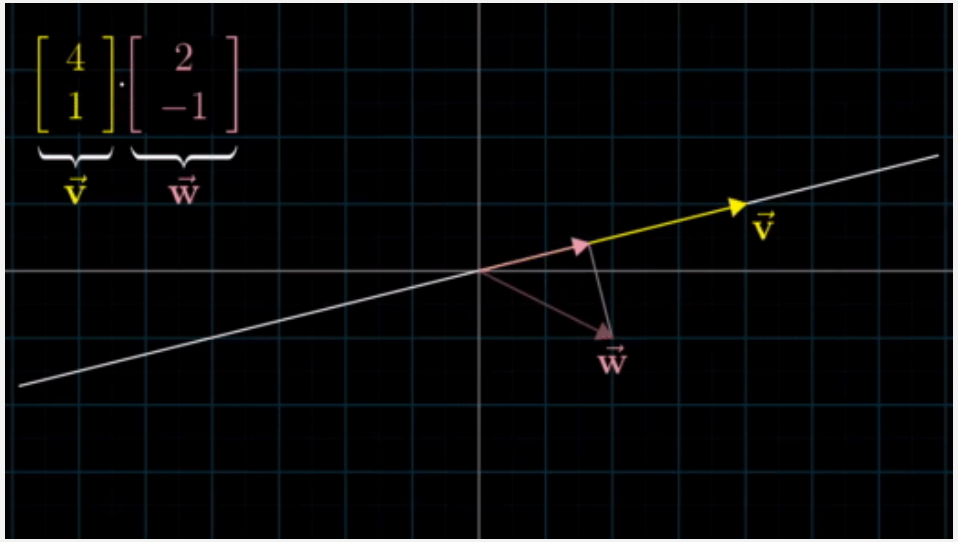
- 내적 : A * B 와 같이 내적할 때, B를 A가 존재하는 방향 '안'으로 넣은 상태에서 크기를 곱하기
- 외적: B와 A 모두가 존재하는 방향의 '밖'에서 크기를 곱하기

벡터-벡터 내적 (Vector-Vector Inner Product)

내적

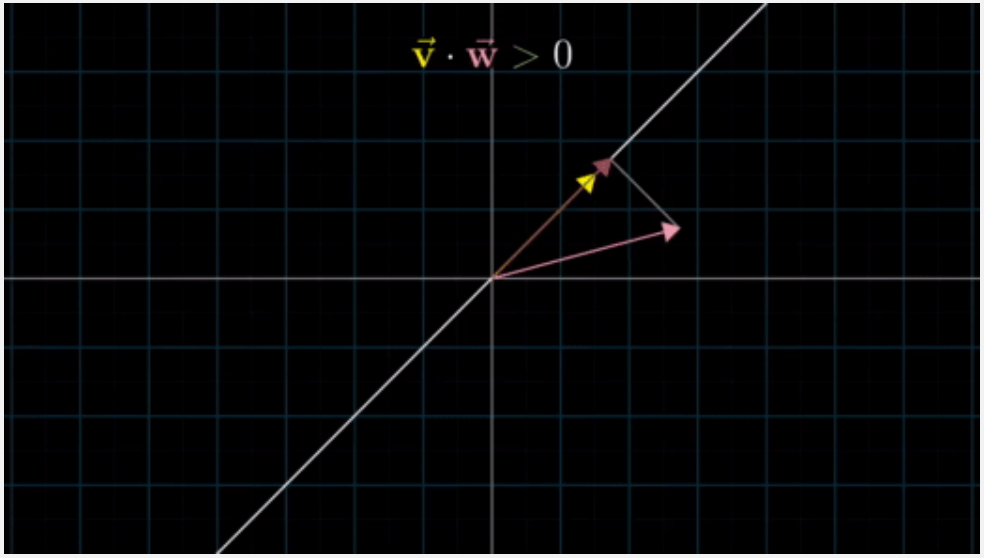
- 투영: 먼저 식($\vec{v} \cdot \vec{w}$) 에서 $\vec{w}$ 를 $\vec{v}$ 가 있는 방향 속으로 넣기
- $\vec{w}$ 가 있는 공간으로 투영된 $\vec{v}$와 $\vec{w}$ 의 크기를 서로 곱하기

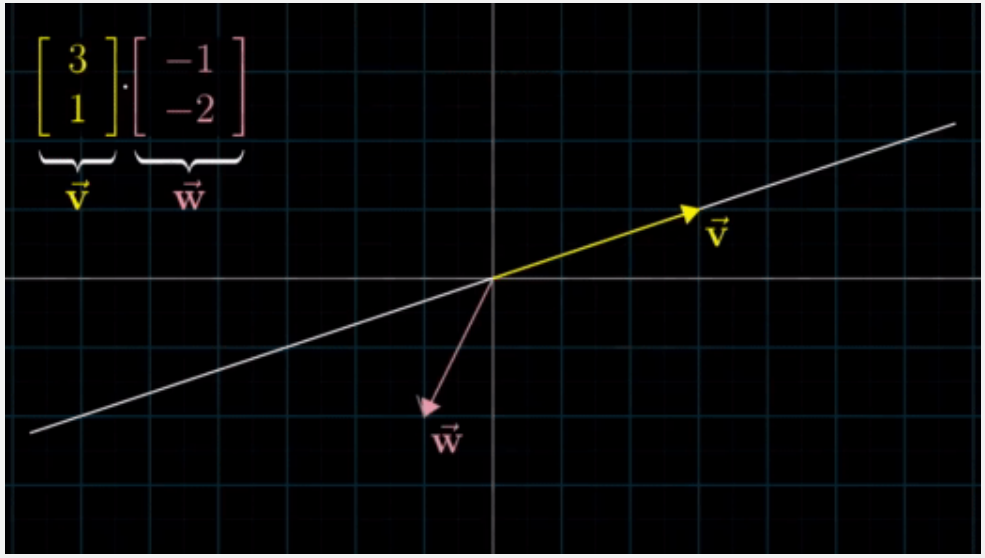




벡터-벡터 내적

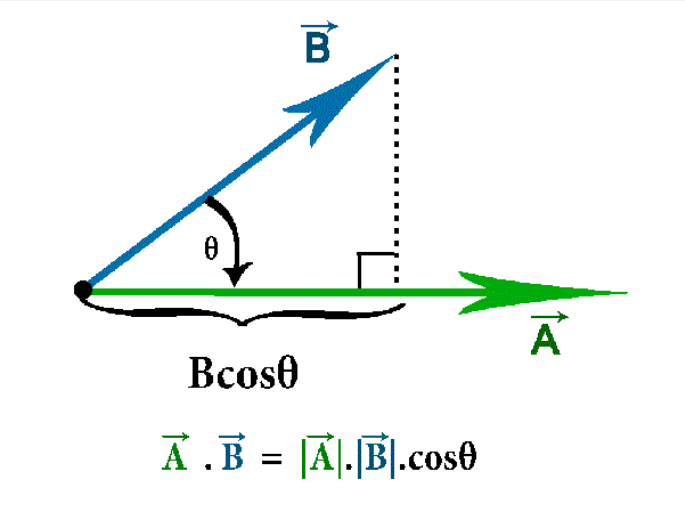
- 내적의 값 양수 : 만약 두 벡터의 방향이 같을 경우
- 내적의 값 0 : 두 벡터가 수직
- 내적의 값 음수: 약 두 벡터가 방향이 반대





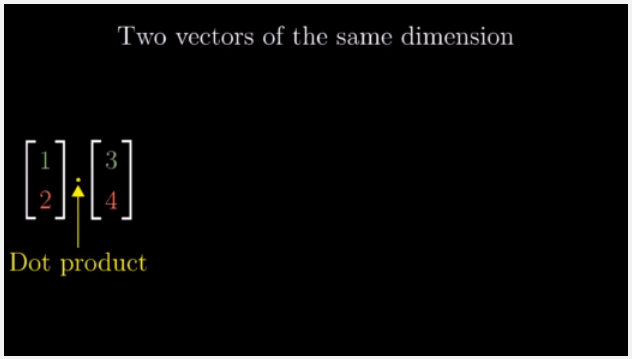
벡터-벡터 내적 계산

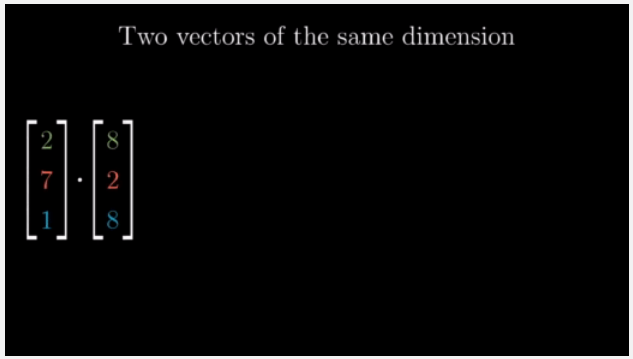
- 내적을 계산할 땐, 대부분 cos을 사용해서 계산
  - $\vec{v}$ 의 norm과 $\vec{w}$ 의 norm을 그냥 곱하는 것이 아니라, $\vec{v}$ 가 투영된 길이만큼 곱해하기.
  - 투영된 길이가 바로 $\vec{v}\cos\theta$

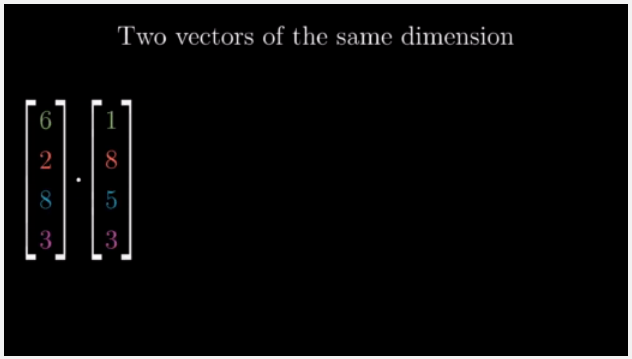


벡터-벡터 내적 계산

근데 매번 코사인 각도를 측정해서 계산하는 것은 너무 비효율적이고 복잡합니다. 따라서 우리는 주로 내적을 계산할 때 이렇게 계산하지 않고 아래처럼 계산








```
import numpy as np


vec_a = np.array([2, 1])
vec_b = np.array([3, 2])

vec_a @ vec_b
```

벡터 매트릭스 Numpy2

명지대학교
MYONGJI UNIVERSITY

행렬의 크기 모양 타입Shape

```
import numpy as np

vec_a = np.array([[1],
                  [-2]])

# 행 2개, 열 1개
vec_a.shape
vec_b = np.array([[1],
                  [-2],
                  [3]])

# 행 3개, 열 1개
vec_b.shape
matrix_a = np.array([[1, 1],
                     [-2, 0]])

# 행 2개, 열 2개
matrix_a.shape
matrix_a.ndim
matrix_a.dtype
```

행렬의 연산

행렬 합 (Matrix Addition)¶

- shape이 맞는 경우 행렬 합

```
matrix_a = np.array([[1, 1],
                     [-2, 0]])

matrix_b = np.array([[1, 1],
                     [2, 3]])

matrix_a + matrix_b
```

- shape이 맞지 않는 경우 행렬 합

```
matrix_a = np.array([[1, 1],
                     [-2, 0]])

matrix_b = np.array([[1, 1, 0],
                     [2, 3, 7]])

matrix_a + matrix_b
```

행렬의 연산

스칼라 곱 (Scalar Multiplication)

```
matrix_a = np.array([[1, 1],
                     [-2, 0]])

scalar = 2

matrix_a * scalar
```

원소곱 (Elementwise Multiplication)

```
matrix_a = np.array([[1, 1],
                     [-2, 0]])

matrix_b = np.array([[2, 100],
                     [2, 100]])

matrix_a * matrix_b
```

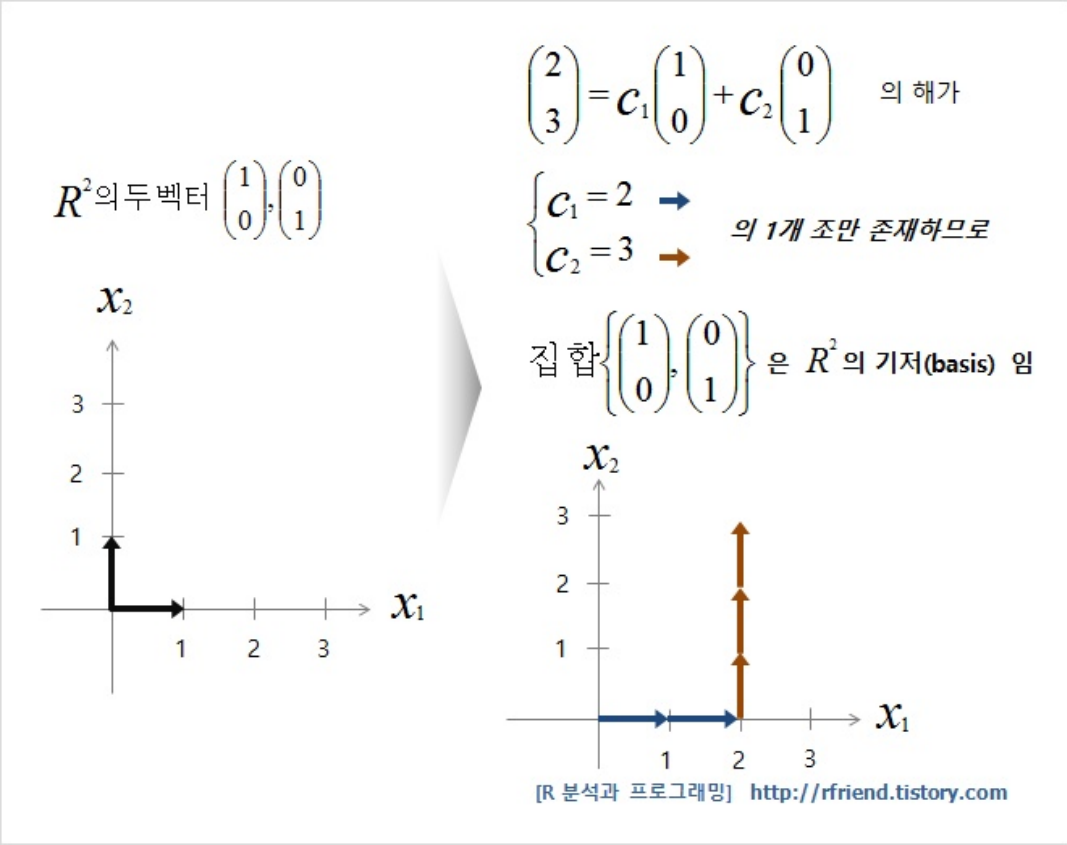
행렬의 연산

행렬-벡터 내적

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} @ \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

기저벡터

기저벡터

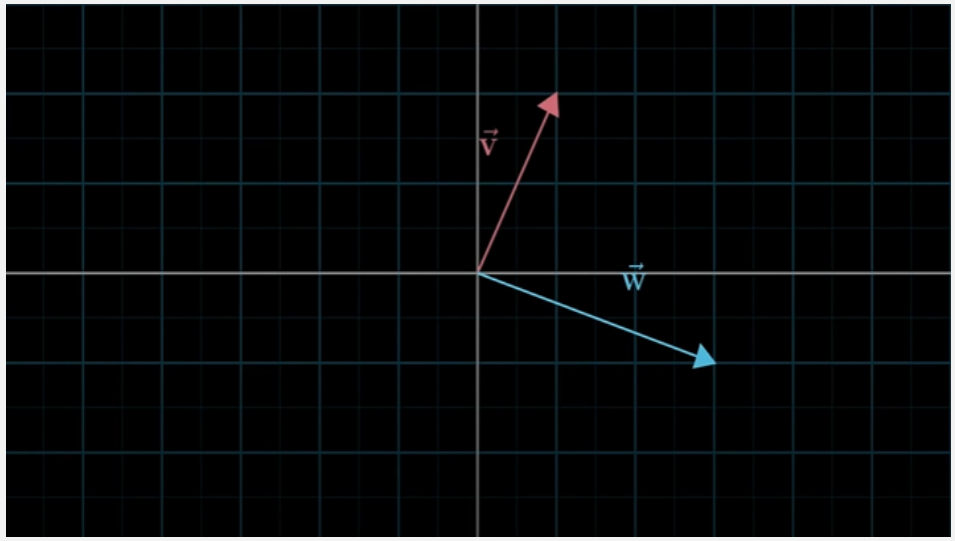


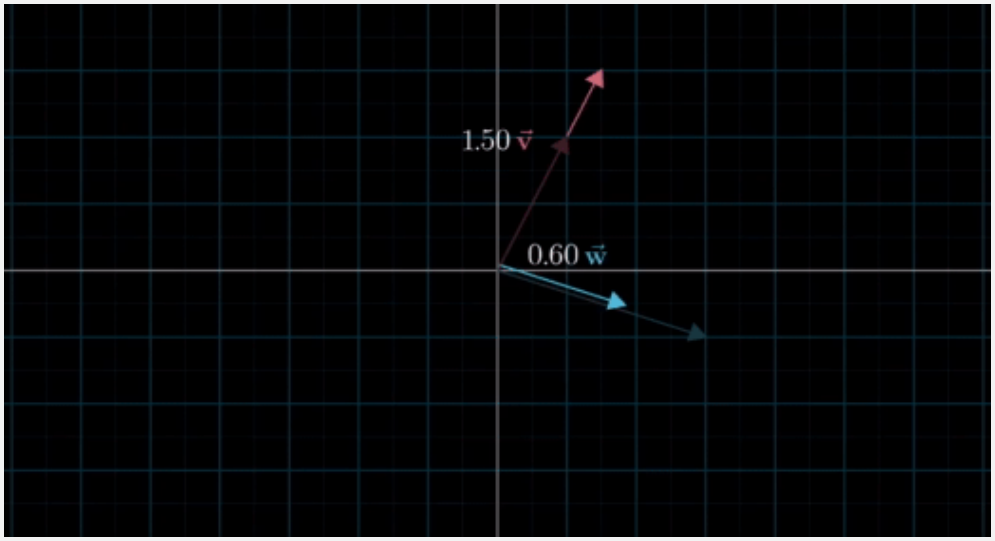
https://img1.daumcdn.net/thumb/R720x0.q80/?scode=mtistory2&fname=http%3A%2F%2Fcfile10.uf.tistory.com%2Fimage%2F265CDB4756A7868316AA5A

기저벡터

$$\vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \vec{w} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

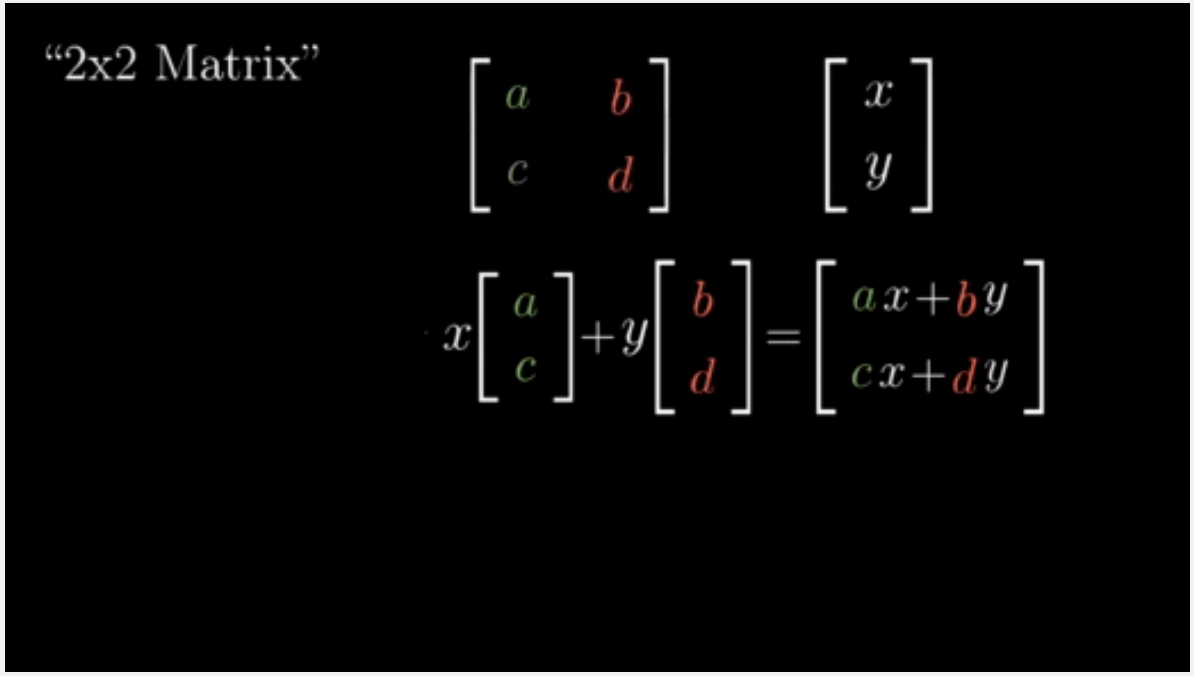
$v^{\rightarrow}$ 와 $w^{\rightarrow}$ 도 2차원 평면공간을 생성
n차원 공간에서 수많은 기적 벡터 존재
"정규 직교 기저벡터"





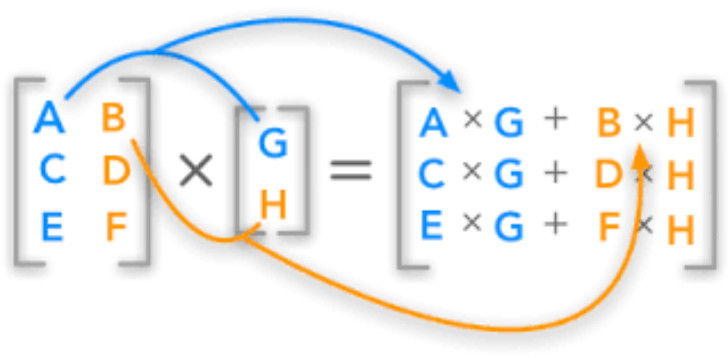
https://www.youtube.com/watch?v=k7RM-ot2NWY

행렬 벡터의 내적


"2x2 Matrix"
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
$\begin{bmatrix} x \\ y \end{bmatrix}$
$x\begin{bmatrix} a \\ c \end{bmatrix}+y\begin{bmatrix} b \\ d \end{bmatrix}=\begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$

Shape 규칙

- 벡터와 행렬 혹은 행렬과 행렬을 내적하려면 내적하려는 두 벡터/행렬의 Shape 이 맞아야




```
# Shape : (2, 1)
vec_x = np.array([[1],
                  [-2]])

# Shape : (3, 2)
mat_a = np.array([[1, 0],
                  [0, 1],
                  [0, 5]])

# Shape : (3, 2) @ (2, 1) = (3, 1)
mat_a @ vec_x
```

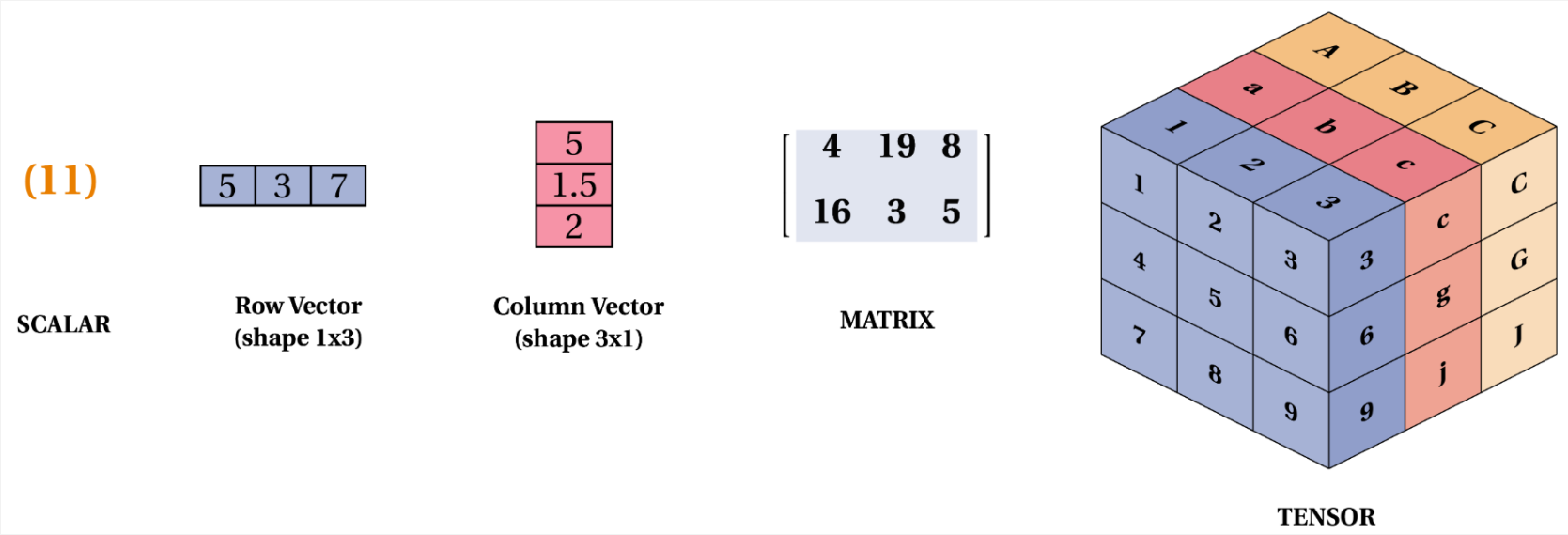
https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQVc8JvlKNThi23OHcm3-Lyq9Gc7-FHgch8Zw&usqp=CAU

벡터 매트릭스 Numpy3

명지대학교
MYONGJI UNIVERSITY

Tensor

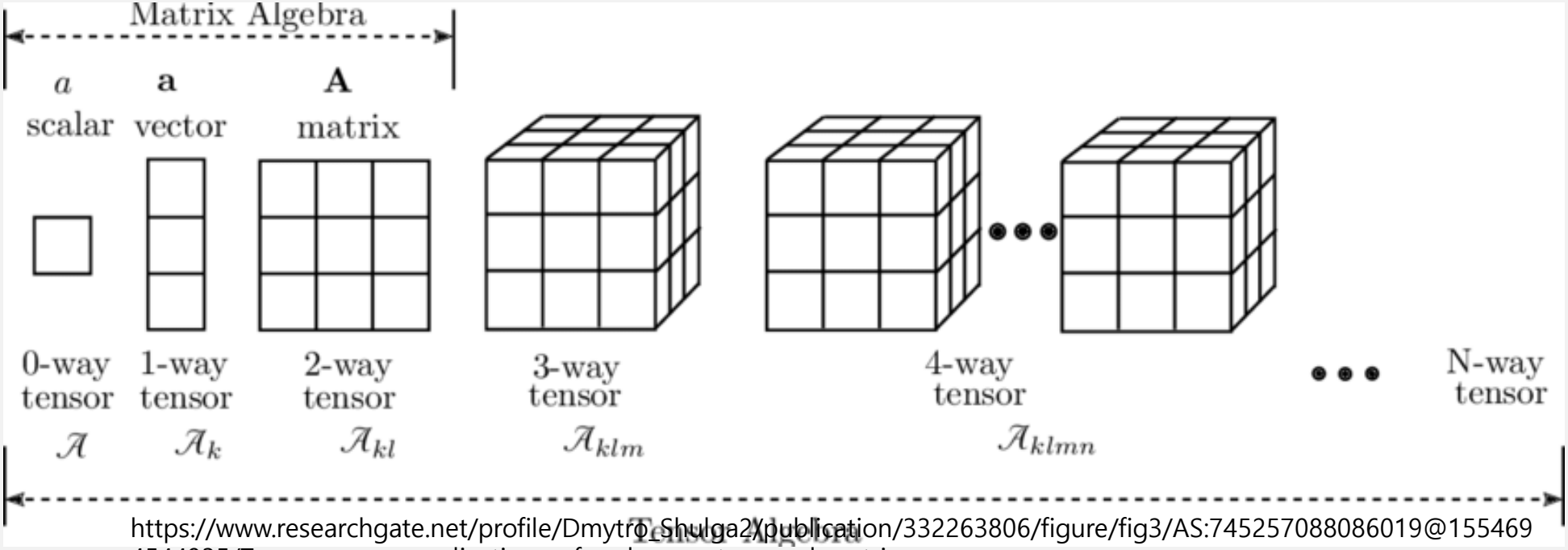
- 행렬은 기본적으로 벡터가 여러 개 쌓여서 만들어진 구조



https://i.pinimg.com/originals/80/c0/b1/80c0b157636aa8cb4b4e56180b8ab8e7.png

Tensor

- 랭크 :tensor의 차원수
- 데이터 자체가 몇개의 방향(차원)으로 존재하는지



https://www.researchgate.net/profile/Dmytro_Shulga2/publication/332263806/figure/fig3/AS:745257088086019@1554694544935/Tensors-as-generalizations-of-scalars-vectors-and-matrices.png

3차원 텐서 Shape 규칙

- (x, i, j) @ (x, j, k)을 연산할 때 x는 같거나 양쪽 중 한쪽이 1이여야 하며, 맨 끝 두개의 Shape은 (i, j) @ (j, k)에서 j는 동일해야하고, 결과 Shape은 (x, i, k)가 된다.
  - (4, 1, 2) @ (4, 2, 5) = (4, 1, 5)
  - (4, 1, 2) @ (3, 2, 5) = 가장 앞자리가 달라서 에러
  - (1, 1, 2) @ (3, 2, 5) = (3, 1, 5)

```
from numpy.random import rand
print((rand(4, 1, 2) @ rand(4, 2, 5)).shape)
print((rand(1, 1, 2) @ rand(3, 2, 5)).shape)
print((rand(4, 3, 1, 2) @ rand(4, 3, 2, 5)).shape)
print((rand(4, 5, 1, 2) @ rand(4, 1, 2, 5)).shape)
print((rand(4, 5, 1, 2) @ rand(1, 5, 2, 5)).shape)
```

Numpy : 전치 행렬

전치 행렬 T

```
import numpy as np

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

mat_a.T
```

Transpose() 연산

- 랭크가 3일 때 기본값은 (0, 1, 2)
- 0번째 1번째 2번째 Shape의 위치를 마음대로 변경

•Shape : (4, 5, 7) ← ([0]:4, [1]:5, [2]:7)
•transpose(0, 1, 2) : (4, 5, 7)
•transpose(0, 2, 1) : (4, 7, 5)
•transpose(1, 2, 0) : (5, 7, 4)
•transpose(1, 0, 2) : (5, 4, 7)

Numpy : squeeze

필요 없는 텐서를 없애기

- Shape : (4, 1, 2) → squeeze : (4, 2)
- Shape : (4, 1, 1, 1, 2) → squeeze : (4, 2)
- Shape : (1, 4, 1, 2) → squeeze : (4, 2)

```
1  from numpy.random import rand
2
3
4  print('origin : ', rand(4, 1, 2).shape, 'squeeze : ', rand(4, 1, 2).squeeze().shape)
5  print('origin : ', rand(4, 1, 1, 1, 2).shape, 'squeeze : ', rand(4, 1, 1, 1, 2).squeeze().shape)
6  print('origin : ', rand(1, 4, 1, 2).shape, 'squeeze : ', rand(1, 4, 1, 2).squeeze().shape)
```

```
origin :  (4, 1, 2) squeeze :  (4, 2)
origin :  (4, 1, 1, 1, 2) squeeze :  (4, 2)
origin :  (1, 4, 1, 2) squeeze :  (4, 2)
```

Numpy : expand_dims

squeeze와 반대로 원하는 Axis에 크기가 1인 Axis를 새로 만듦

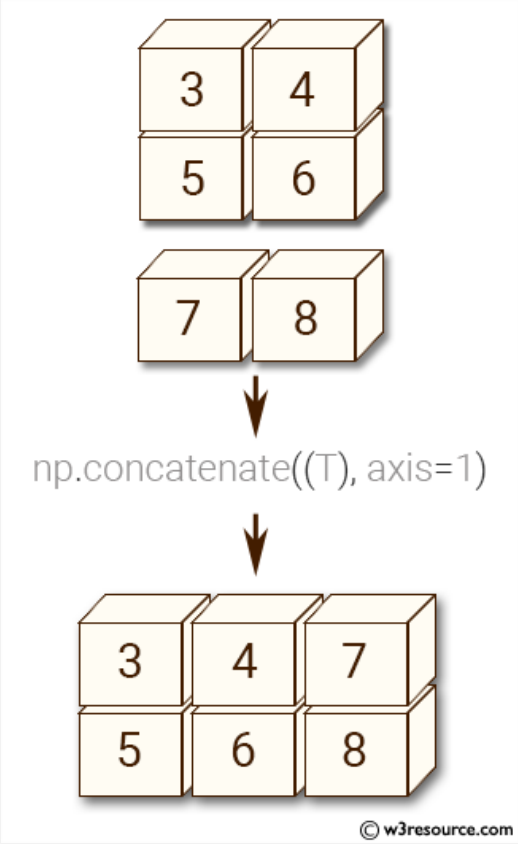
- Shape : (4, 2) → expand_dims(0) : (1, 4, 2)
- Shape : (4, 2) → expand_dims(1) : (4, 1, 2)
- Shape : (4, 2) → expand_dims(2) : (4, 2, 1)

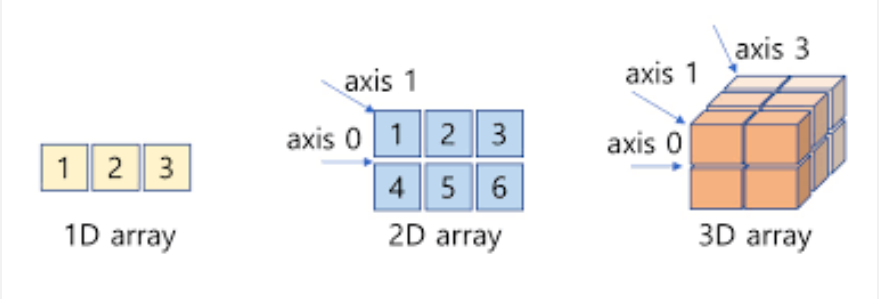
```
1  from numpy.random import rand
2
3
4  print('expand_dims(0) : ', np.expand_dims(rand(4, 2), axis=0).shape)
5  print('expand_dims(1) : ', np.expand_dims(rand(4, 2), axis=1).shape)
6  print('expand_dims(2) : ', np.expand_dims(rand(4, 2), axis=2).shape)
```

```
expand_dims(0) :  (1, 4, 2)
expand_dims(1) :  (4, 1, 2)
expand_dims(2) :  (4, 2, 1)
```

Numpy : Concatenate

- 두 텐서를 이어 붙여서 새로운 텐서





https://www.w3resource.com/w3r_images/numpy-manipulation-concatenate-function-image-2.png

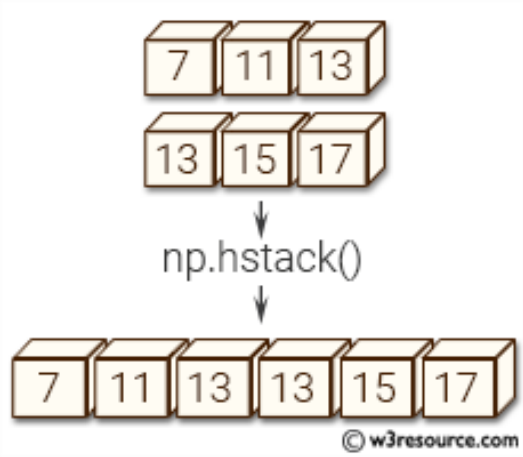
https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcSgdUDcxpqzY8aOf2kVfHCNasiMxcJBFXkidA&usqp=CAU

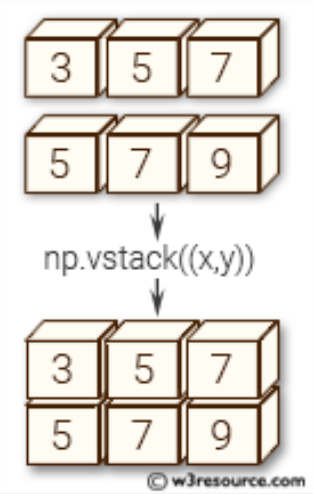
Numpy : Concatenate

```python
from numpy.random import rand

tensor_a = rand(4, 1, 2)
tensor_b = rand(4, 1, 2)

print(np.concatenate([tensor_a, tensor_b], axis=0).shape)
print(np.concatenate([tensor_a, tensor_b], axis=1).shape)
print(np.concatenate([tensor_a, tensor_b], axis=2).shape)
```

```
(8, 1, 2)
(4, 2, 2)
(4, 1, 4)
```

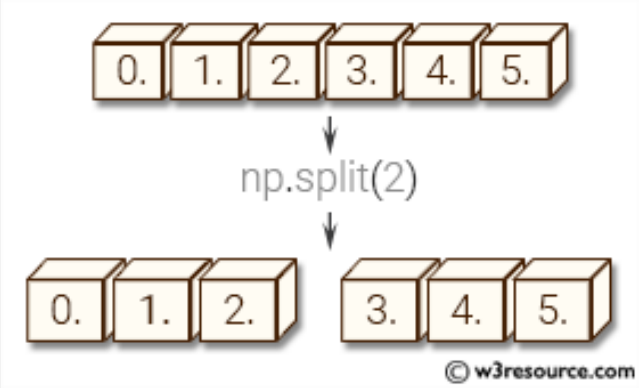
Numpy : Vstack hstack





https://www.w3resource.com/numpy/manipulation/

Numpy : Split




```
>>> import numpy as np
>>> a = np.arange(8.0)
>>> np.split(a, 2)
[array([ 0., 1., 2., 3.]), array([ 4., 5., 6., 7.])]
```