1. 벡터가 뭔가요?

벡터는 숫자를 원소로 가지는 리스트 또는 배열이다.

세로로 나열되어 있는 배열은 **열 벡터**, 가로로 나열된 배열은 **행 벡터**라고 부른다. 이때 벡터의 원소 개수를 **벡터의 차원**이라고 부른다.

수식

$$\mathbf{x} = \begin{bmatrix} 1 \\ 7 \\ 2 \end{bmatrix} \qquad \mathbf{x}^{\top} = [1, 7, 2]$$

코드

```
x = [1, 7, 2]
```

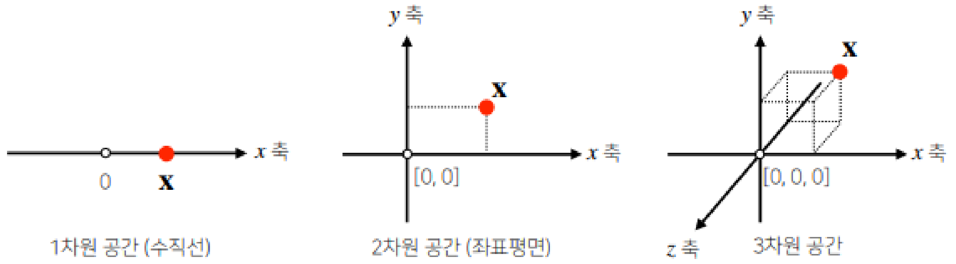
```
x = np.array([1, 7, 2])
```

`np` 는 `numpy` 를 말합니다

보통 코드로 표현할 때는 numpy를 많이 사용하며 위처럼 보통 행 벡터를 선언한다.

벡터는 공간에서 **한 점**을 의미하며 원점으로부터 상대적 **위치**를 표현한다.

또한 벡터에 **숫자를 곱해주면 길이만 변한다.** 벡터에 숫자를 곱하는 것을 **스칼라 곱**이라고 부르며 1보다 크면 길이가 늘어나고 작으면 줄어든다. 0보다 작은 경우 방향이 바꾼다.

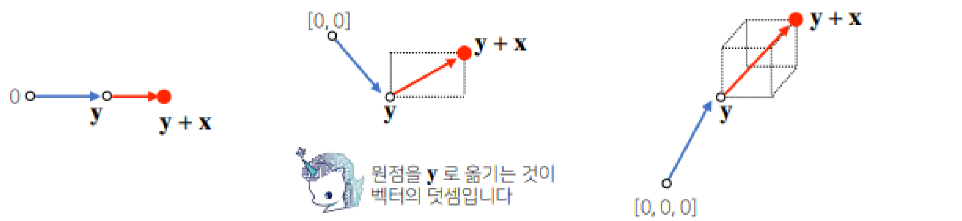


2. 벡터의 연산

벡터끼리 같은 모양을 가지면 **덧셈, 뺄셈**을 할 수 있다. 각 원소를 위치에 맞게 일대일 대응해 계산하면 된다.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} \qquad \mathbf{x} \pm \mathbf{y} = \begin{bmatrix} x_1 \pm y_1 \\ x_2 \pm y_2 \\ \vdots \\ x_d \pm y_d \end{bmatrix}$$

두 벡터의 덧셈은 다른 벡터로부터 **상대적 위치이동**을 표현한다.



벡터끼리 같은 모양을 가지면 **성분곱(Hadamard product)**을 계산할 수 있다. 각 원소를 위치에 맞게 대응시켜 곱해주면 된다.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} \quad \mathbf{x} \odot \mathbf{y} = \begin{bmatrix} x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_d y_d \end{bmatrix}$$

덧셈, 뺄셈, 성분곱 연산은 넘파이에서 숫자끼리의 연산과 동일한 수식으로 계산할 수 있다.

```
[1]  import numpy as np

[2]  x = np.array([1, 7, 2])
     y = np.array([5, 2, 1])

[3]  x + y

     array([6, 9, 3])

[4]  x - y

     array([-4,  5,  1])

     x * y

     array([ 5, 14,  2])
```

3. 벡터의 노름 구해보기

벡터의 **노름(norm)**은 주어진 벡터와 원점의 거리를 의미한다. 사실 노름은 여러 종류가 있는데 보통 L1-노름, L2-노름으로 나눈다.

L1-노름은 각 성분의 변화량의 절대값을 모두 더하고, L2-노름은 피타고라스 정리를 이용해 유클리드 거리를 계산한다.

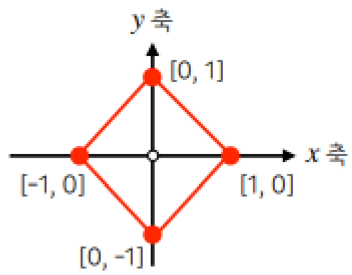
```python
def l1_norm(x):
    x_norm = np.abs(x)
    x_norm = np.sum(x_norm)
    return x_norm

def l2_norm(x):
    x_norm = x*x
    x_norm = np.sum(x_norm)
    x_norm = np.sqrt(x_norm)
    return x_norm
```

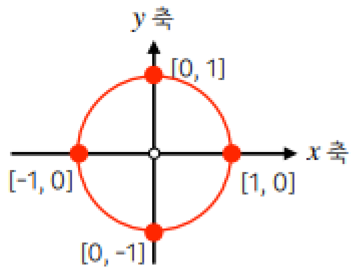
$$\|\mathbf{x}\|_1 = \sum_{i=1}^{d} |x_i|$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{d} |x_i|^2}$$

이때 노름의 종류에 따라 기하학적 성질이 달라진다. L1-노름과 L2-노름을 적용시켜 원을 그리면 아래와 같다. 두 원이 다른 이유는 **거리의 개념이 달라지기 때문**이다. 머신러닝에서는 각 성질들이 필요할 때가 있으므로 둘 다 사용한다.



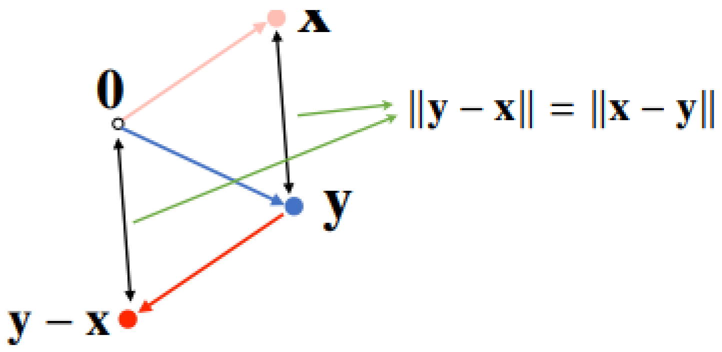
L_1-노름 상의 원: $\{\mathbf{x} : \|\mathbf{x}\|_1 = 1\}$
예: Robust 학습, Lasso 회귀



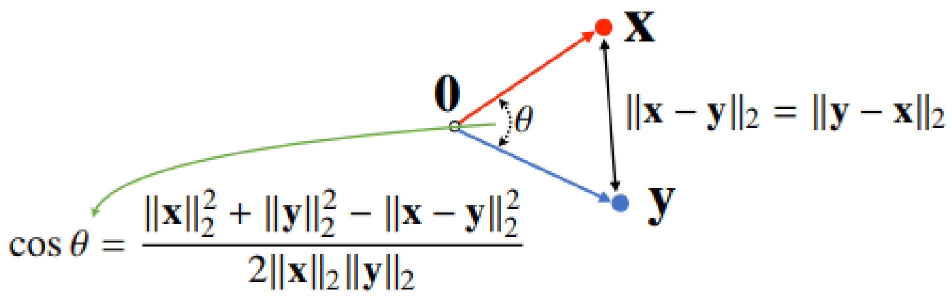
L_2-노름 상의 원: $\{\mathbf{x} : \|\mathbf{x}\|_2 = 1\}$
예: Laplace 근사, Ridge 회귀

4. 두 벡터 사이의 거리

L1, L2-노름을 이용해 **두 벡터 사이의 거리**를 계산할 수 있는데 이때 벡터의 뺄셈을 사용한다.



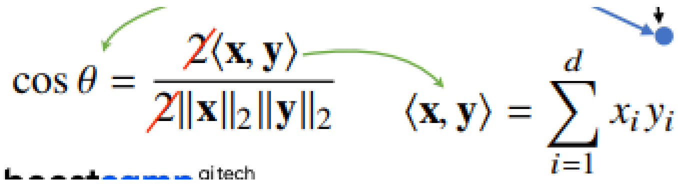
두 벡터 사이의 거리를 이용해 **각도도 계산**할 수 있다. 주의할 점은 **L2-노름만 가능하다**는 점이다. **제2 코사인 법칙**을 이용해 두 벡터 사이의 각도를 계산할 수 있다.



참고) 코사인 1법칙, 2법칙

$$a = b\cos C + c\cos B \qquad a^2 = b^2 + c^2 - 2bc\cos A$$
$$b = c\cos A + a\cos C \qquad b^2 = c^2 + a^2 - 2ca\cos B$$
$$c = a\cos B + b\cos A, \qquad c^2 = a^2 + b^2 - 2ab\cos C$$

위 수식에서 분자를 풀어주면 2||x*y||가 된다. 즉, **내적**을 사용하면 분자를 쉽게 계산할 수 있다.

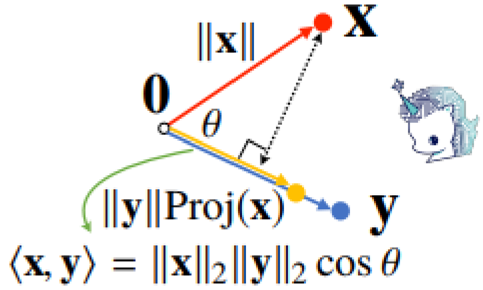


내적은 np.inner()를 이용해 계산한다.

```python
def angle(x, y):
    v = np.inner(x, y) / (l2_norm(x) * l2_norm(y))
    theta = np.arccos(v)
    return theta
```

5. 내적

내적은 정사영(orthogonal projection)된 벡터의 길이와 관련있다. Proj(x)의 길이는 ||x||*cos이 된다. 이때 내적은 **정사영의 길이(Proj(x)의 길이)를 벡터 y의 길이 ||y||만큼 조정한 값**이다. 내적을 이용해 **두 벡터의 유사도를 측정**하는데 사용할 수 있다.



<내적 추가 자료>

$\vec{a}, \vec{b}$ 가 이루는 각 θ일 때,

$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}|\cos\theta$ ($\vec{a}, \vec{b}$ 의 내적 (inner product))

특징

1. 내적의 결과는 스칼라 값이다.

2. <a, a> = |a||a|cos0 = |a|^2이다.

3. a혹은 b가 0 벡터이면 <a, b> = 0이다.

4. 교환법칙, 분배법칙이 성립한다.