

## Chapter 1 벡터, 파트1: 벡터와 벡터의 기본 연산

- 벡터의 정의, 해석 방법,
- 파이썬으로 벡터 활용
- 벡터 대수학과 내적(dot product) 등 주요 연산
- 벡터 분해

### 1.1 NumPy로 벡터 생성 및 시각화하기

1	선형대수학에서 $[a]$ 는 수를 <b>순서대로 나열</b> 한 것을 의미한다. (벡터는 함수 등 다른 수학적 대상을 갖을 수 있으나 교재에서는 다루지 않는다)	a. 벡터(vector)
2	벡터는 다음의 두 가지 특징을 갖는다.  ① $[a]$ : 벡터가 가진 원소의 수 ② $[b]$ : 벡터가 열 방향인지 행 방향인지를 나타낸다.	a. 차원(dimensionality)  b. 방향(orientation)
3	차원은 $[a]$ 로 표기하며 $R$ 은 실수(Real number), $N$ 은 차원을 나타낸다. 참고로 $C$ 는 복소수(실수와 허수의 합으로 이루어진 수)이다.  <div style="text-align: center;"> <p>** example **</p> <math display="block">x = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 6 \end{bmatrix} \text{ 4차원 열 벡터, } x \in R^4</math> </div>	a. $R^N$
4	수학에서의 차원과 파이썬에서 차원은 다른 의미를 갖는다. 수학에서 차원은 $[a]$ 인 반면 파이썬에서 벡터와 행렬의 차원은 $[b]$ 이다.  예를 들어 NumPy는 원소의 수에 상관없이 <b>2차원 배열</b> 로 간주한다.	a. 원소의 수  b. 기하학적 차원
5	특정 방향이 없는 경우 파이썬에서 $[a]$ 로 설정된다. 출력은 행으로 출력되나 이는 $[b]$ 는 아니다.	a. 1차원 배열  b. 행 벡터

6	수학적 차원인 벡터의 원소 수는 파이썬에서는 [ a ] 혹은 [ b ]이라 한다.	a. 길이(length) b. 모양(shape)
7	벡터는 다음과 같이 표기한다.  ① 진한 로마자 : [ a ] ② 이탤릭체 : [ b ] or [ c ]	a. $\mathbf{v}$ b. $v$ c. $\vec{v}$
8	선형대수학에서 보통 벡터에 표시가 없다면 [ a ]을 가정한다.	a. 열 방향
9	행 벡터의 경우 [ a ]로 표기한다. $T$ 는 전치 연산(transpose operation)을 나타낸다.	a. $w^T$
10	파이썬 벡터는 여러 데이터 타입으로 나타낼 수 있다. <b>리스트 타입</b> 이 가장 간편하기는 하지만 선형대수학 연산은 리스트에 잘 동작하지 않기 때문에 [ a ]로 생성하는 것이 가장 좋다.	a. NumPy 배열
11	<p align="center"><b>** 벡터의 방향은 중요한가? **</b></p> <ul style="list-style-type: none"> <li>➤ 벡터는 세 개의 방향이 있다. ① <b>열 방향</b>, ② <b>행 방향</b>, ③ <b>방향이 없는 1차원 배열</b>.</li> <li>➤ 데이터를 <b>저장</b>할 때에는 벡터의 방향이 중요하지 않다.</li> <li>➤ 그러나 <b>연산</b>을 할 때에 벡터의 방향이 잘못되면 원하는 결과가 나오지 않는 경우가 발생할 수 있다.</li> </ul>	

### 1.1.1 벡터의 기하학적 해석

1	<p align="center">&lt; 벡터의 해석 &gt;</p> <p>① <b>대수학적 해석</b></p> <ul style="list-style-type: none"> <li>➤ 순서대로 [ a ]된 수 목록(ordered list of numbers).</li> <li>➤ 데이터 과학에 유용하다(시간별 판매 등).</li> </ul> <p>② <b>기하학적 해석</b></p> <ul style="list-style-type: none"> <li>➤ 특정 [ b ]와 [ c ]을 가진 직선.</li> <li>➤ 물리학과 공학에 유용하다.</li> </ul>	a. 나열 b. 길이(크기, magnitude) c. 방향(양의 x 축의 각도, angle)
---	--	---

2	<p>벡터의 두 점은 <b>꼬리(시작)</b>와 <b>머리(끝)</b>으로 불리며 일반적으로 머리에 <math>[a]</math>가 달려있다.</p>	a. 화살표
3	<p><b>벡터</b>는 직선의 형태로 <b>좌표</b>가 인코딩 것으로 착각할 수 있다. 그러나 벡터와 좌표는 다른 개념이다.</p> <p>단, 원점에서 시작하는 벡터는 좌표와 동일하며 이를 <math>[a]</math>이라 한다.</p>	a. 기준 위치 (standard position)
4	<div data-bbox="609 645 1040 1093" data-label="Figure"> </div> <ul style="list-style-type: none"> <li>✓ 위의 벡터는 모두 <b>동일한 벡터</b>이다.</li> <li>✓ 파란 원이 <b>머리(끝)</b>이고 노란 원이 <b>꼬리(시작)</b>이다.</li> <li>✓ 원점에서 시작하는 벡터는 <b>기준 위치(standard position)</b>로 좌표와 동일한 의미를 갖는다.</li> </ul>	

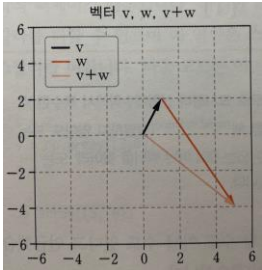
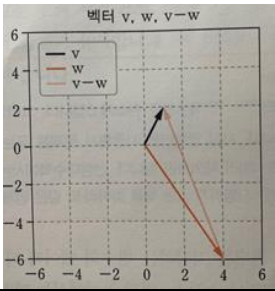
## 1.2 벡터 연산

벡터는 이야기의 주인공이다. 이러한 벡터(주인공)에게 부여되는 움직임을 연산(operation)이라 한다. 연산에는 간단한 덧셈부터 조금 어려운 특잇값 분해 등이 있다.

### 1.2.1 두 벡터의 덧셈

1	벡터의 덧셈과 뺄셈은 동일한 $[a]$ 을 갖는 벡터끼리만 가능하다( $R^N \pm R^N$ ).	a. 차원
2	벡터의 덧셈과 뺄셈은 서로 $[a]$ 되는 원소끼리 연산이 행해진다.  <pre>    ** example **     import numpy as np      v = np.array([4, 5, 6])     w = np.array([10, 20, 30])     vPlusw = v + w      print(vPlusw)  [14 25 36]</pre>	a. 대응
3	벡터의 방향이 다른 벡터의 덧셈과 뺄셈을 시행하는 경우 파이썬에서는 $[a]$ 이라는 연산을 시행한다.  <pre>    ** example **      v = np.array([[4, 5, 6]]) # 행벡터     w = np.array([[10, 20, 30]]).T # 열벡터 '.T'는 전치 연산.     v + w  array([[14, 15, 16],        [24, 25, 26],        [34, 35, 36]])</pre>	a. 브로드캐스팅 (broadcasting)
4	<p style="text-align: center;">&lt; 결론 &gt;</p> <p>두 벡터의 <b>차원</b>과 <b>방향</b>이 같을 때에만 벡터의 덧셈과 뺄셈이 가능하다.</p>	

## 1.2.2 벡터의 덧셈과 뺄셈의 기하학적 해석

1	<p style="text-align: center;"><b>&lt; 두 벡터를 기하학 적으로 더함 &gt;</b></p> <ul style="list-style-type: none"> <li>➤ 첫 번째 벡터의 [ a ] → 두 번째 벡터의 [ b ]로 이어지는 선</li> <li>➤ 모든 벡터를 더하면 첫 번째 [ a ] → 마지막 [ b ]로 이어지는 선이 된다.</li> </ul> <p style="text-align: center;">** example **</p> 	<p>a. 꼬리</p> <p>b. 머리</p>
2	<p style="text-align: center;"><b>&lt; 두 벡터를 기하학적으로 뺌 &gt;</b></p> <ul style="list-style-type: none"> <li>➤ 두 벡터의 꼬리를 [ a ]에 둔다(기준 좌표 이용)</li> <li>➤ 빼기의 결과는 '두 번째 벡터의 [ b ] → 첫 번째 벡터의 [ b ]'로 이어지는 선이 된다.</li> </ul> <p style="text-align: center;">** example **</p> 	<p>a. 같은 좌표</p> <p>b. 머리</p>
3	<p>벡터의 뺄셈은 다음의 이유로 기하학적으로 중요하다.</p> <div style="text-align: center;"> <p>벡터의 뺄셈</p> <p>↓</p> <p>직교벡터 분해의 기초</p> <p>↓</p> <p>선형 최소제곱법의 기초</p> <p>↓</p> <p>과학과 공학에서 활용되는 선형대수학의 가장 중요한 응용</p> </div>	

### 1.2.3 스칼라-벡터 곱셈

1	선형대수학에서 [ a ]란 벡터나 행렬에 포함된 숫자가 아닌 수 그 자체이다.	a. 스칼라(scalar)
2	스칼라는 일반적으로 [ a ] 그리고 [ b ]와 같은 그리스어 소문자로 나타낸다.	a. $\alpha$ b. $\lambda$
3	스칼라와 벡터의 곱은 다음처럼 매우 간단한다.  $\lambda = 4, \quad w = \begin{bmatrix} 9 \\ 4 \\ 1 \end{bmatrix}, \quad \lambda w = \begin{bmatrix} 4 \times 9 \\ 4 \times 4 \\ 4 \times 1 \end{bmatrix}$	
4	스칼라를 곱할 때에는 data type이 중요하다. 다음의 두 가지 경우는 파이썬 연산에서 자주 사용되므로 알아 두자.  ① list에 스칼라 연산을 하면 list의 요소를 [ a ]한다. ② NumPy 배열에 스칼라 연산을 하면 기존의 방식으로 스칼라를 [ b ]한다.  ✖ int와 float의 type 또한 맞추어 주어야 함을 주의하자  <div style="text-align: center;">** example **</div> <pre>import numpy as np  s = 2 a = [3, 4, 5] # 리스트 b = np.array(a) # np 배열  # type0  list인 경우와 array인 경우 스칼라 연산이 다르다. print(a*s) # 리스트의 경우 반복 print(b*s) # NumPy 배열의 경우 원래대로 스칼라 곱  [3, 4, 5, 3, 4, 5] [ 6  8 10]</pre>	a. 반복 b. 곱

## 1.2.4 스칼라-벡터 덧셈

기본적으로 **선형대수학**에서 스칼라와 벡터를 더하는 것은 **불가**하다. 그 이유는 스칼라와 벡터는 서로 다른 개념이기 때문이다.

그러나 **파이썬**에서는 벡터와 스칼라를 **더할 수 있다**. 더하는 방법은 스칼라-벡터 곱과 같이 벡터의 각 원소에 스칼라를 더한다.

**\*\* example \*\***

```
import numpy as np

s = 2
v = np.array([3, 6])
s + v

array([5, 8])
```

**\*\* 스칼라-벡터 곱셈의 기하학적 해석 \*\***

1	스칼라를 scalar라고 부르는 이유는 스칼라는 벡터의 [ a ]을 바꾸지 않고 벡터의 [ b ]만 조정하기 때문이다.  (scalar는 '방향의 구별은 없고 하나의 수치만으로 완전히 표시되는'의 의미가 있다.	a. 방향  b. 크기
2	스칼라-벡터 곱셈 결과는 스칼라가 ① 1 보다 크다, ② 0과 1 사이이다, ③ 0 이다, ④ 음수이다 에 따라 다르다.	
3	스칼라가 음수인 경우 벡터가 음의 방향으로 바뀌는 듯 보이는데 아래의 흐름을 따라가면 그렇지 않다는 것을 알 수 있다.  <b>&lt; 왜 음의 스칼라-벡터 곱은 방향이 바뀌는 것이 아닌가 &gt;</b>  벡터는 원점을 통과하여 <b>양방향의 무한대로 가는 무한히 긴 선</b> 이라는 해석이 존재한다.  → '회전된'벡터는 위의 의미에서 여전히 동일한 <b>무한한 선</b> 을 가리킨다.  → 음의 스칼라로 인하여 회전된 벡터 또한 여전히 <b>동일한 방향</b> 이다.  <b>※</b> 위의 해석은 <b>행렬 공간, 고유벡터, 그리고 특이벡터</b> 에서 중요하다.	

4	<p>벡터 덧셈과 스칼라-벡터 곱셈을 이용하면 벡터의 평균(vector average)을 구할 수 있다. N개의 벡터를 가정하면 N개의 벡터를 모두 더하고 스칼라 [ a ]을 곱해준다.</p> <p style="text-align: center;">[ b ]</p>	<p>a. <math>\frac{1}{N}</math></p> <p>b. <math>\frac{1}{N} * N</math> 개의 벡터 합</p>
---	--	---

### 1.2.5 전치

1	[ a ] 연산은 열 벡터를 행 벡터로 혹은 반대로 변환한다.	a. 전치(transpose)
2	<p>예를 들어 <math>i</math>행, <math>j</math>열을 갖는 행렬 <math>m</math>의 전치(transpose) 결과는 다음과 같다.</p> $m_{i,j}^T = [ a ]$	a. $m_{j,i}$
3	<p style="text-align: center;"><b>&lt; 중요 규칙 &gt;</b></p> <p>벡터에 두 번 전치(transpose) 연산을 하면 벡터는 원래 방향이 된다. 즉, 다음과 같다.</p> $v^{TT} = [ a ]$ <p>※ 위의 규칙은 데이터 과학과 머신러닝에서 중요한 증명의 핵심 근거가 된다. 예를 들면 <b>데이터 행렬 <math>A \times</math> 데이터 행렬 <math>A^T =</math> 대칭 공분산 행렬</b> 이 된다.</p>	a. $v$



## 1.2.6 파이썬에서 벡터 브로드캐스팅

브로드캐스팅 연산은 전통적인 선형대수학 교과서에는 존재하지 않으며 **현대 컴퓨터 기반 선형대수학**에만 존재하는 연산이다.

**\*\* example \*\***

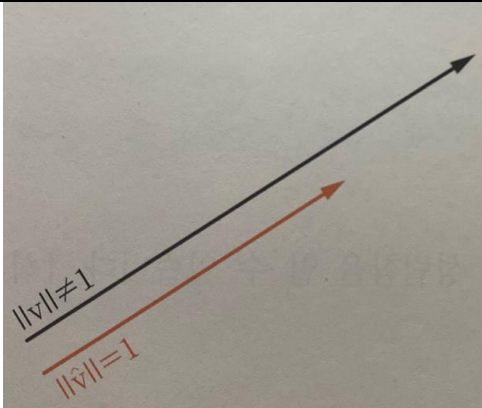
```
import numpy as np
v = np.array([[1, 2, 3]]).T # 열벡터
w = np.array([[10, 20]]) # 행벡터
v + w # 브로드캐스팅 덧셈

array([[11, 21],
       [12, 22],
       [13, 23]])
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 10 & 20 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} + \begin{bmatrix} 10 & 20 \\ 10 & 20 \\ 10 & 20 \end{bmatrix} = \begin{bmatrix} 11 & 21 \\ 12 & 22 \\ 13 & 23 \end{bmatrix}$$

## 1.3 벡터 크기와 단위벡터

1	벡터의 크기는 <b>기하학적 길이</b> 혹은 [ a ]이라고도 한다. 벡터의 크기를 구하는 방식은 일반적으로 [ b ] 거리 공식을 사용한다. 마지막으로 벡터의 크기는 [ c ]로 표기한다.	a. 노름(norm) b. 표준 유클리드(Euclidean) c. $  v  $						
2	벡터의 크기를 수식으로 표현하면 다음과 같다.  $  v   = [ a ]$	a. $\sqrt{\sum_{i=1}^n v_i^2}$						
3	앞서 언급하였듯 파이썬과 선형대수학 사이에는 용어의 불일치가 존재한다. 이를 정리하면 다음과 같다. <table><tr><th>파이썬</th><th>선형대수학</th></tr><tr><td>파이썬 함수 [ a ]</td><td>배열의 차원</td></tr><tr><td>파이썬 함수 [ b ]</td><td>기하학적 길이</td></tr></table>	파이썬	선형대수학	파이썬 함수 [ a ]	배열의 차원	파이썬 함수 [ b ]	기하학적 길이	a. len()  b. np.norm()
파이썬	선형대수학							
파이썬 함수 [ a ]	배열의 차원							
파이썬 함수 [ b ]	기하학적 길이							
4	벡터의 <b>기하학적 길이(노름, norm)</b> 가 1인 벡터를 [ a ]라 한다. [ a ]는 직교 행렬, 회전 행렬, 고유벡터, 특이벡터 등 응용에 사용된다. 단위벡터는 [ b ]로 정의된다.	a. 단위벡터(unit vector)  b. $  v   = 1$						

5	<p>비단위벡터의 경우 같은 방향의 단위벡터로 만들 수 있다. 방법은 비단위벡터에 <math>\frac{1}{\ a\ }</math>의 역수를 곱하며 <math>\hat{a}</math>로 표기한다.</p>	<p>a. 벡터 노름</p> <p>b. <math>\hat{a}</math></p>
6	<p>단위벡터 생성을 수식으로 표현하면 다음과 같다.</p> $\hat{a} = \frac{1}{\ a\ } a$ <p>※ 단, 모든 비단위벡터가 연관된 단위벡터를 갖는 것은 아님을 주의해야 한다.</p>	<p>a. <math>\frac{1}{\ v\ } v</math></p>
7	<div style="text-align: center;">  <p>** 비단위벡터의 단위벡터화 **</p> </div>	

## 1.4 벡터-내적

1	[ a ](점곱 & 스칼라곱)는 하나의 숫자로 <b>두 벡터 사이의 관계</b> 를 나타내며 선형대수학에서 가장 중요한 연산이다.	a. 내적(dot product)
2	내적의 표기는 [ a ], $a \cdot b$ , 그리고 $\langle a, b \rangle$ 등이 있으며 교재에서는 [ a ]를 사용한다.	a. $a^T b$
3	<p>내적을 계산하는 방법은 두 벡터의 대응되는 원소끼리 곱한 다음 모든 결과를 더한다.</p> $  \begin{aligned}  &[1, 2, 3, 4] \cdot [5, 6, 7, 8] \\  &= 1 \times 5 + 2 \times 6 + 3 \times 7 + 4 \times 8 \\  &= 5 + 12 + 21 + 32 \\  &= 70  \end{aligned}  $ <p>두 벡터 a와 b가 있을 때 a와 b의 내적을 식으로 표현하면 다음과 같다.</p> $\delta = [ a ]$ <p>※ 내적은 <b>동일한 차원</b>의 두 벡터 사이에서만 성립한다.</p>	a. $\sum_{i=1}^n a_i b_i$
4	<p>파이썬에서 벡터의 내적은 [ a ] 함수를 통해 쉽게 구할 수 있다.</p> <p>※ 단, [ a ]는 실제로 벡터-내적을 구현하지는 않으며 <b>첫 번째 입력이 행벡터</b>이고, <b>두 번째 입력이 열벡터</b>인 경우에만 결과를 출력한다.</p>	a. np.dot()
5	<p>벡터에 스칼라를 곱하면 내적도 그만큼 커지는 특성을 갖는다. 예를 들어 <math>a^T b = 70</math> 일 때 <math>10a^T b</math>는 [ a ]이 된다.</p>	a. 700
6	<p>내적은 두 벡터 사이의 [ a ] 또는 [ b ]의 척도로 해석할 수 있다.</p> <p>예를 들어 몸무게와 키는 상관관계가 있을 것이므로 내적 또한 클 것이다.</p>	<p>a. 유사성(similarity)</p> <p>b. 매핑(mapping)</p>

7	데이터에서 두 변수의 벡터의 내적을 구할 경우 [ a ]에 영향을 받을 수 있다.	a. 단위
8	두 변수의 벡터의 내적을 구할 때 단위의 영향을 받는다면 <b>정규화 계수</b> 로 단위의 차이를 제거할 수 있다.  이렇게 <b>정규화된 내적</b> 은 [ a ]라 불리며 데이터 과학에서 가장 중요한 분석이다.	a. 피어슨 상관계수 (Pearson correlation coefficient)

### 1.4.1 내적의 분배 법칙

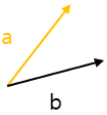
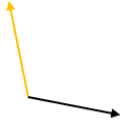



1	<p>내적의 분배 법칙은 다음과 같다.</p> $a^T(b + c) = [ a ]$ <pre> ** example ** import numpy as np  a = np.array([ 0, 1, 2 ]) b = np.array([ 3, 5, 8 ]) c = np.array([ 13, 21, 34 ])  # 내적 분배 법칙 res1 = np.dot( a, b+c ) res2 = np.dot( a,b ) + np.dot( a,c )  print(f'{res1}, {res2}') 110, 110 </pre>	a. $a^T b + a^T c$
---	--	--------------------

### 1.4.2 내적의 기하학적 해석

1	<p>다음은 벡터 내적의 <b>기하학적 정의</b>이다. 기하학적 정의의 식은 그 모양이 다르지만 원래 벡터 내적식과 수학적으로 동일하다.</p> <table><tr><th>내적 식</th><th>기하학적 내적 식</th></tr><tr><td><math display="block">\delta = \sum_{i=1}^n a_i b_i</math></td><td><math display="block">\alpha = [ a ]</math></td></tr></table>	내적 식	기하학적 내적 식	$\delta = \sum_{i=1}^n a_i b_i$	$\alpha = [ a ]$	a. $\cos(\theta_{v,w})\ v\ \ w\ $
내적 식	기하학적 내적 식					
$\delta = \sum_{i=1}^n a_i b_i$	$\alpha = [ a ]$					
2	<p>벡터의 크기는 양수이다. 그러나 기하학적 정의에서는 <b>cos이 -1과 1사이의 값</b>을 갖기에 <math>[ a ]</math>인 경우가 존재한다.</p> <p>이는 기하학적 정의에서 <math>[ b ]</math>가 두 벡터 사이의 <b>기하학적 정의</b>를 보인다.</p>	a. 음수 b. 내적의 부호				

3

두 벡터 사이의 각에 따라 내적 부호는 다섯 가지의 사례가 존재한다.

예각	둔각	직각	공선	공선
$\theta < 90^\circ$	$\theta > 90^\circ$	$\theta = 90^\circ$	$\theta = 0^\circ$	$\theta = 180^\circ$
				
$\cos(\theta) > 0$ $\alpha > 0$	$\cos(\theta) < 0$ $\alpha < 0$	$\cos(\theta) = 0$ $\alpha = 0$	$\cos(\theta) = 1$ $\alpha =  a  b $	$\cos(\theta) = -1$ $\alpha = - a  b $

4

< 암기하세요. 직교벡터의 내적은 0입니다 >

① 두 벡터가 [ a ]한다.

↕

② 두 벡터는 내적이 [ b ]이다.

↕

③ 두 벡터 사이의 각은 [ c ]이다.

a. 직교

b. 0

c.  $90^\circ$

a. 직교

b. 0

c.  $90^\circ$

## 1.5 그 외 벡터 곱셈

### 1.5.1 아다마르곱

1	<p>[ a ]은 두 벡터의 대응되는 각 원소를 곱한다. 곱의 결과는 두 벡터와 [ b ] 차원의 벡터이다.</p> <p>※ 동일한 차원의 벡터만 아다마르곱을 할 수 있다.</p>	<p>a. 아다마르곱 (Hardamard product)</p> <p>b. 같은</p>
2	<p>아다마르곱(Hardamard product)는 여러 [ a ]를 곱할 때 편리하다. 예를 들면 다음과 같다.</p> <p>소형 장치의 수 변수 × 장치당 가격 변수</p>	<p>a. 스칼라</p>

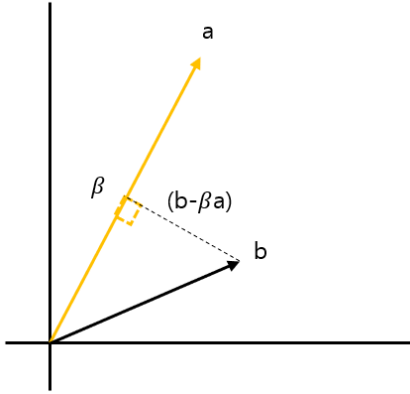
## 1.5.2 외적

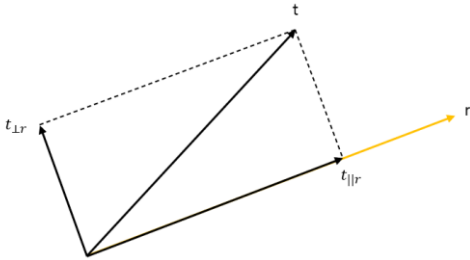
1	<p>외적은 열벡터와 행벡터를 이용해 [ a ]을 만든다.</p> <ul style="list-style-type: none"> <li>✓ 외적 행렬의 각 행 : [ b ] 스칼라에 대응되는 열벡터 원소의 곱</li> <li>✓ 외적 행렬의 각 열 : [ c ] 스칼라에 대응되는 행벡터 원소의 곱.</li> </ul>	<p>a. 행렬</p> <p>b. 행벡터</p> <p>c. 열벡터</p>												
2	<p style="text-align: center;"><b>&lt; 외적과 내적의 차이 &gt;</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th></th><th>외적</th><th>내적</th></tr> </thead> <tbody> <tr> <th>결과</th><td>행렬</td><td>스칼라</td></tr> <tr> <th>동일 차원 필요</th><td>X</td><td>O</td></tr> <tr> <th>표기</th><td><math>vw^T</math></td><td><math>v^T w</math></td></tr> </tbody> </table>			외적	내적	결과	행렬	스칼라	동일 차원 필요	X	O	표기	$vw^T$	$v^T w$
	외적	내적												
결과	행렬	스칼라												
동일 차원 필요	X	O												
표기	$vw^T$	$v^T w$												
3	<p>외적의 표기는 <math>vw^T</math>이고 내적의 표기는 <math>v^T w</math>로 표기가 다르다. 이는 외적은 [ a ]이고, 내적은 [ b ]임을 의미하며 교재 중반부에 이해하면 된다.</p> <p>※ 기본적으로 아무런 처리가 없다면 <b>열벡터임을 가정</b>한다(기억).</p>	<p>a. 열벡터 × 행벡터</p> <p>b. 행벡터 × 열벡터</p>												
4	<p style="text-align: center;"><b>&lt; 브로드캐스팅과 외적 &gt;</b></p> <ul style="list-style-type: none"> <li>✓ 브로드캐스팅 : 덧셈, 곱셈, 나눗셈과 같은 산술 연산을 벡터로 확장한 일반적인 [ a ].</li> <li>✓ 외적 : 두 벡터를 곱하는 특수한 [ b ].</li> </ul>	<p>a. 코딩 연산</p> <p>b. 수학적 기법</p>												
5	<p>NumPy를 이용해 각각 열과 행방향인 두 벡터를 [ a ] 또는 [ b ] 함수에 입력하여 외적을 계산할 수 있다.</p>	<p>a. np.outer()</p> <p>b. np.dot()</p>												

## 1.5.3 교차곱과 삼중곱

기하학과 물리학에서 사용되지만 기술 관련 응용 분야에서 자주 등장하지는 않기에 교재에서 다루지는 않는다.

## 1.6 직교벡터 분해

1	<p>벡터 또는 행렬을 [ a ]한다는 것은 여러 <b>간단한 조각</b>들로 나뉘어짐을 의미한다.</p> <p>example)  <math>42.01 \rightarrow 42 + 0.01</math>  <math>42 \rightarrow 2 \times 3 \times 7</math> (소인수 분해, Prime factorization)</p>	<p>a. 분해</p>
2	<p><b>&lt; 직교 투영법, Orthogonal projection &gt;</b></p>  <p>① <b>목표</b>  두 벡터 a와 b가 있음을 가정할 때 벡터 a에서 벡터 b의 머리와 최대한 <b>가까운 점</b>을 찾는다.</p> <p>② <b>최적화 문제로 변환</b>  투영 거리가 최소가 되도록 b를 a에 투영  → 벡터 a에 b를 투영한 점은 벡터 a의 크기를 줄인 <b>βa</b> (β는 스칼라)  → 스칼라 <b>β</b>를 찾는 것이 최종 목표</p> <p>③ <b>핵심은 벡터의 뽀셈</b>  벡터 b의 머리와 벡터 a가 <b>직각</b>으로 만나도록 그리면 벡터 b와 가장 가까운 벡터 a 위의 점 <b>βa</b>를 찾을 수 있다.  → 벡터의 뽀셈은 두 벡터의 머리 사이의 직선 = [ a ].  → 벡터 a와 벡터 b-βa는 직교.  → 벡터 a와 벡터 b-βa의 내적은 [ b ].</p>	<p>a. <math>(b-\beta a)</math></p> <p>b. 0</p> <p>c. <math>\frac{a^T b}{a^T a}</math></p> <p>d. 최소 거리</p>

	<p>위를 수식으로 풀어 쓰면 다음과 같다.</p> $  \begin{aligned}  a^T(b - \beta a) &= 0 \\  a^T b - \beta a^T a &= 0 \\  \beta a^T a &= a^T b \\  \beta &= [ c ]  \end{aligned}  $ <p>④ 결론</p> <p>직교 투영법(Orthogonal projection)은 점을 [ d ]로 선에 투영하는 공식. 이는 선형 모델을 푸는 데 잘 알려진 최소제곱식, 통계학, 머신러닝 등의 많은 응용분야의 기초가 된다.</p>	
3	<p>&lt; 직교 투영법 → 직교 벡터 분해 &gt;</p>  <p> <math>t</math> : 목표벡터,  <math>r</math> : 기준벡터,  <math>t_{  r}</math> : 기준벡터 <math>r</math>과 평행 성분,  <math>t_{\perp r}</math> : 기준벡터 <math>r</math>과 수직 성분         </p> <p>① 목적</p> <p>목표 벡터(<math>t</math>)를 두 개의 다른 벡터(<math>t_{  r}</math>와 <math>t_{\perp r}</math>)로 분해</p> <p>② 직교 벡터 성질</p> <ol style="list-style-type: none"> <li>두 벡터의 합은 목표벡터이다. [ a ]</li> <li>하나의 벡터는 기준벡터와 [ b ]이고, 다른 하나의 벡터는 기준벡터와 [ c ]이다.</li> </ol> <p>③ 직교 투영법(Orthogonal projection)을 통한 <math>t_{  r}</math> 계산.</p> <p>직교 투영법을 적용하면 다음과 같다.</p> $t_{  r} = [ d ]$ <p>※ 직교 투영법의 결과는 스칼라이지만 직교 분해의 결과는 크기를 조정된 벡터이다.</p>	<ol style="list-style-type: none"> <li><math>t_{  r} + t_{\perp r} = t</math></li> <li>직교(<math>t_{\perp r}</math>)</li> <li>평행(<math>t_{  r}</math>)</li> <li><math>r \frac{t^T r}{r^T r}</math></li> <li><math>t - r \frac{t^T r}{r^T r}</math></li> <li>0</li> </ol>



	<p>④ 직교 벡터의 성질을 통한 <math>t_{\perp r}</math> 계산.</p> <p>두 벡터의 합은 목표벡터이다.</p> <p><math>\rightarrow t_{  r} + t_{\perp r} = t</math></p> <p><math>\rightarrow t_{\perp r} = t - t_{  r}</math></p> <p><math>\rightarrow t_{\perp r} = [ e ]</math></p> <p>⑤ 증명 방법</p> <p>수직 성분과 기준벡터의 내적이 <math>[ f ]</math>임을 증명.</p> <p><math>\rightarrow t_{\perp r}^T r = [ f ]</math></p> <p><math>\rightarrow (t - r \frac{t^T r}{r^T r})^T r = [ f ]</math></p>	
5	<p>&lt; 요약 &gt;</p> <ul style="list-style-type: none"> <li>✓ 하나의 수학적 대상을 다른 수학적 대상으로 <b>분해</b>한다.</li> <li>✓ 분해의 세부적인 내용은 <b>제약 조건</b>(현재는 기준벡터와 직교 &amp; 평행)에 따라 달라진다.</li> <li>✓ 다른 제약 조건의 경우(<b>분석 목표가 다름</b>) 동일 벡터라도 다른 방식으로 분해한다.</li> </ul>	

## 1.7 정리

- ✓ 아무리 복잡한 계산이라도 선형대수학에서는 결국 간단한 연산으로 이루어져 있으며 대부분은 기하학적 직관으로 이해할 수 있다.
- ✓ 이번 장에서 배운 내용들은 책의 나머지 부분과 응용 선형대수학자로서 앞으로 많은 도움이 된다.

## \*\* 요점정리 \*\*

1	<p>&lt; 요점정리 &gt;</p> <ul style="list-style-type: none"> <li>✓ 벡터는 열 또는 행에 숫자를 [ a ]한 것이다. 벡터의 원소 수를 [ b ]이라고 하며, 벡터는 차원과 동일한 수의 [ c ]을 가진 기하학적 공간에서 <b>하나의 선</b>으로 나타낼 수 있다.</li> <li>✓ 덧셈, 뺄셈, 아다마르곱과 같은 벡터 산술 연산은 [ d ]별로 계산한다.</li> <li>✓ 내적은 차원이 같은 두 벡터 간의 [ e ]를 인코딩한 단일 숫자로, <b>원소별로 곱하고 합</b>해서 구한다.</li> <li>✓ 두 벡터가 직교하면 내적은 [ f ]이며 기하학적으로 벡터가 [ g ]으로 만나는 것을 의미한다.</li> <li>✓ 직교벡터 분해는 하나의 벡터를 기준벡터와 [ h ]하는 벡터, [ i ]한 벡터로 나누는 것이다. 분해 공식은 기하학적으로 도출될 수 있지만, 공식이 내포한 개념인 '[ j ]'이라는 문구를 기억해야 한다.</li> </ul>	<ul style="list-style-type: none"> <li>a. 나열</li> <li>b. 차원(dimensionality)</li> <li>c. 축</li> <li>d. 원소</li> <li>e. 관계</li> <li>f. 0</li> <li>g. 직각</li> <li>h. 직교</li> <li>i. 평행</li> <li>j. 크기에 대한 매핑</li> </ul>
---	---	---