



인공지능(AI)

MACHINE LEARNING, DEEP LEARNING

하성호

머신러닝

인공지능, 머신러닝, 딥러닝

인공지능

컴퓨터가 인간의 행동을 흉내내는 모든 기술

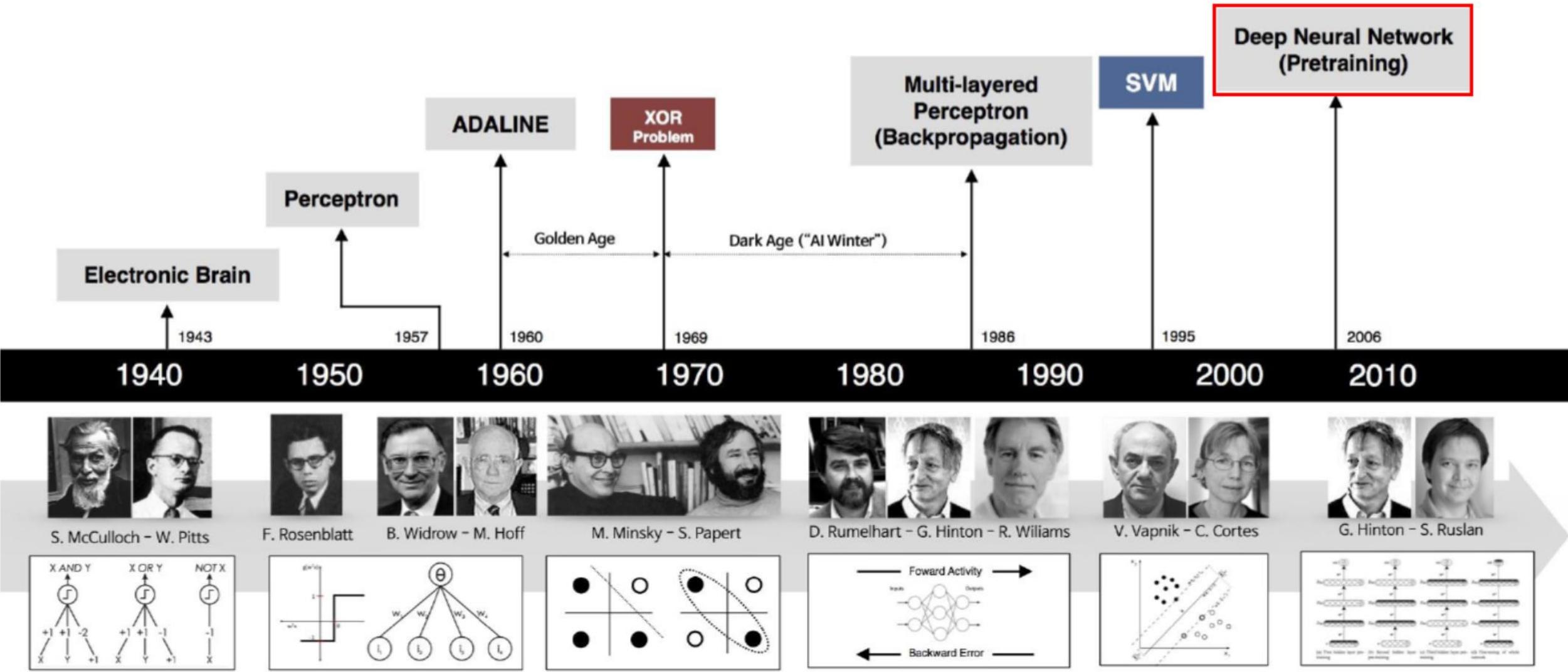
머신러닝

명시적으로 프로그래밍되지 않고 학습할 수 있는 기능

딥러닝

신경네트워크를 사용하여 데이터에서 패턴추출

인공지능, 머신러닝의 역사



인공지능, 머신러닝의 역사

- ▶ 2009 : ImageNet + AlexNet
- ▶ 2014 : GANs
- ▶ 2016-17 : AlphaGo, AlphaZero
- ▶ 2017-2020: AlphaFold

인공지능 쇼크

- ▶ DeepBlue - 2승 3무 1패 (1996)
- ▶ Watson 퀴즈쇼 – 1등 (2011)





인공지능 쇼크

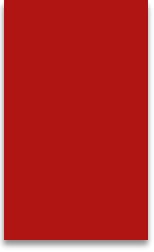
알파고(2016), 알파제로(2017)

머신러닝이란?

- ▶ 데이터로부터 학습하도록 컴퓨터를 프로그래밍하는 과학
- ▶ [일반적]
명시적인 프로그램 없이 컴퓨터가 학습하는 능력을 갖추게 하는 연구 분야
 - 아서 사무엘 (1959)
- ▶ [공학적]
어떤 작업 T에 대한 컴퓨터 프로그램의 성능을 P로 측정했을 때 경험 E로 인해 성능이 향상됐다면, 이 컴퓨터 프로그램은 작업 T와 성능 측정 P에 대한 경험 E로 학습한 것이다.
 - 톰 미첼 (1997)

우리 주위의 머신러닝 사례

- ▶ 광학 문자 판독기(Optical Character Recognition)
- ▶ 스팸필터(Spam filter)
- ▶ 음성 검색



“ 기계가 배운다는 것의 의미?

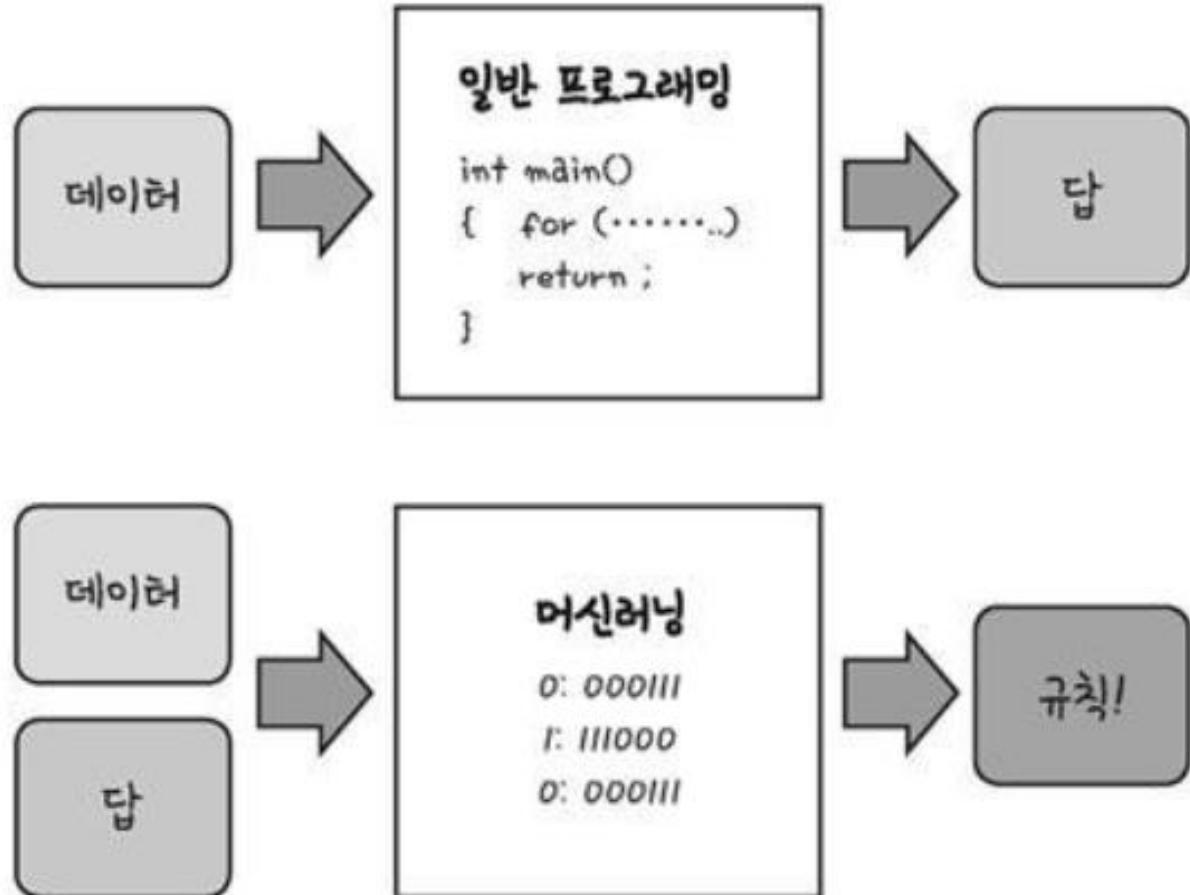
”

내 컴퓨터가 실제로 무언가를 배울 수 있을까?

일반 프로그래밍과 머신러닝 차이점

▶ 규칙을 찾는 방법

- ▶ 기존의 프로그래밍 방식은 데이터를 입력해서 답을 구하는데 초점이 맞춰져 있음
- ▶ 머신러닝은 데이터 안에서 규칙을 발견하고 그 규칙을 새로운 데이터에 적용해서 새로운 결과를 도출하는데 초점이 맞춰져 있음



머신러닝의 학습방법

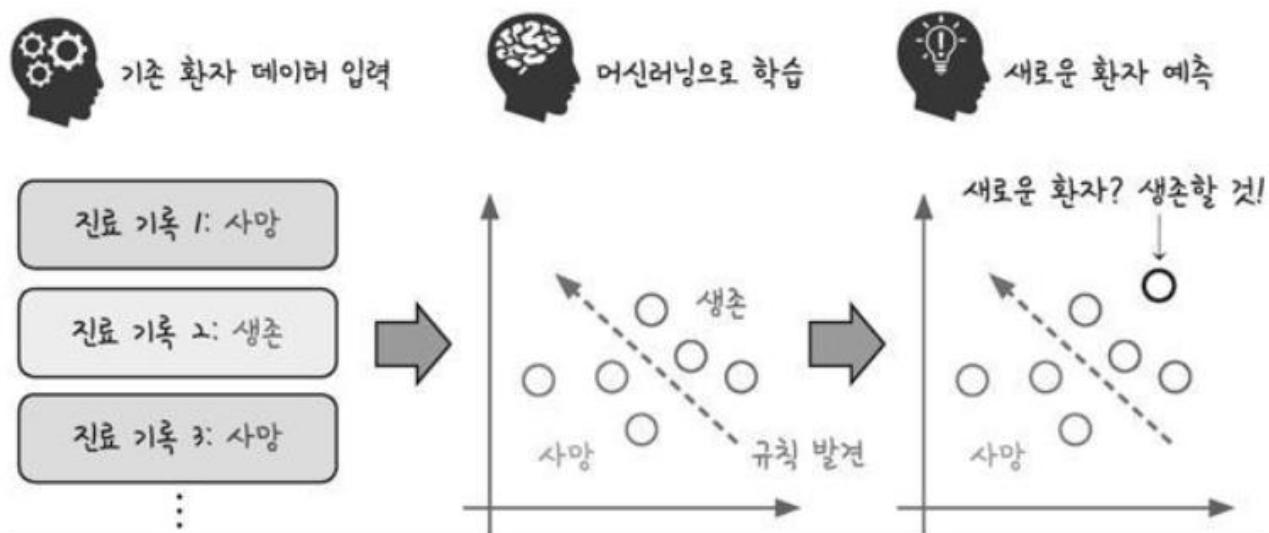
- ▶ 지도학습(Supervised Learning)
 - ▶ 입력 데이터와 정답 데이터가 학습 데이터로 함께 제공되는 방법
- ▶ 비지도학습(Unsupervised Learning)
 - ▶ 입력 데이터만 있고 정답 데이터는 없는 상태에서 학습하고 출력 데이터를 얻는 방법
 - ▶ 주어진 데이터를 분석해서 자동으로 군집을 묶어주는 클러스터링이 있음
- ▶ 강화학습(reinforcement Learning)
 - ▶ 측정값(관측 값)을 입력 받고 행동 방침을 출력하는 방법
 - ▶ 출력 단계에서는 정답인지 알 수 없으나 나중에 결과에 대한 보상(reward)로 정답 여부를 확인함

현 과정에서 다루는 것

- ▶ 지도학습(Supervised Learning)
 - ▶ 회귀(Regression)
 - ▶ 예) 하루 예산 매출액 , 출력이 연속값(Continuous Value)
 - ▶ 분류(Classification)
 - ▶ 예) 동물의 사진분류, 출력이 이산값(Discrete Value)

머신러닝 활용사례

- ▶ 대상
 - ▶ 중환자를 수술하는 의사
- ▶ 하고 싶은 일
 - ▶ 수술하기 전 수술 후 환자의 생존률을 예측할 수 없을까?
- ▶ 방법
 - ▶ 그동안 집도한 수술 환자의 전 상태와 수술 후 생존률을 정리한 데이터를 머신러닝 알고리즘에 넣는 것
 - ▶ 머신러닝은 환자들의 패턴분석
 - ▶ 패턴이 분석되는 과정을 학습(training)



우리가 배우려는 것

- ▶ 학습(training)과 예측(predict)의 구체적인 과정
 - ▶ 예측 성공률 == 데이터들 사이에서 현실과 가깝거나 동일한 경계선을 긋는 것

처음 해보는 머신러닝 실습

- ▶ 폐암수술 환자의 생존율 예측하기 실습

폐암환자 데이터 분석

▶ 데이터 특성

- ▶ 파일이름 : ThoracicSurgery.csv
- ▶ 폴란드 브로츠와프 의과대학 2013년 공개
- ▶ 수술 전 진단데이터와 수술 후 생존 결과
- ▶ 총 470개라인, 각 라인은 18개 항목
- ▶ 17개 정보, 속성(Attribute)
 - ▶ 종양의 유형, 폐활량, 호흡곤란여부, 고통 정도, 기침, 흡연, 천식 여부 등
- ▶ 마지막 R컬럼, 클래스(class)라 명명
 - ▶ 생존여부

▶ 딥러닝 구동법

- ▶ 속성만 뽑아서 데이터셋 만들기
- ▶ “클래스”를 담는 데이터셋 따로 만들기 (정답데이터)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	293	1	3.8	2.8	0	0	0	0	0	0	12	0	0	0	1	0	62	0
2	1	2	2.88	2.16	1	0	0	0	1	1	14	0	0	0	1	0	60	0
3	8	2	3.19	2.5	1	0	0	0	1	0	11	0	0	1	1	0	66	1
4	14	2	3.98	3.06	2	0	0	0	1	1	14	0	0	0	1	0	80	1
5	17	2	2.21	1.88	0	0	1	0	0	0	12	0	0	0	1	0	56	0
6	18	2	2.96	1.67	0	0	0	0	0	0	12	0	0	0	1	0	61	0
7	35	2	2.76	2.2	1	0	0	0	1	0	11	0	0	0	0	0	76	0
8	42	2	3.24	2.52	1	0	0	0	1	0	12	0	0	0	1	0	63	1
9	65	2	3.15	2.76	1	0	1	0	1	0	12	0	0	0	1	0	59	0
10	111	2	4.48	4.2	0	0	0	0	0	0	12	0	0	0	1	0	55	0
11	121	2	3.84	2.56	1	0	0	0	1	0	11	0	0	0	0	0	59	0
12	123	2	2.8	2.12	1	0	0	1	1	0	13	0	0	0	1	0	80	0
13	130	2	5.6	4.64	1	0	0	0	1	0	11	0	0	0	1	0	45	0
14	132	2	2.12	1.72	1	0	0	0	0	0	12	0	0	0	1	0	74	0
15	133	2	2.5	71.1	0	0	0	1	0	0	13	0	0	0	1	0	64	1

[실습코드] 폐암환자 생존율 예측

```
[1]: # 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense
```

```
[2]: # 필요한 라이브러리를 불러옵니다.  
import numpy as np  
import tensorflow as tf
```

```
[3]: # 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.  
np.random.seed(3)  
tf.random.set_seed(3)
```

```
[4]: # 준비된 수술 환자 데이터를 불러들입니다.  
Data_set = np.loadtxt("./dataset/ThoracicSurgery.csv", delimiter=',')  
Data_set
```

```
[4]: array([[293. , 1. , 3.8 , ... , 0. , 62. , 0. ],  
       [ 1. , 2. , 2.88, ... , 0. , 60. , 0. ],  
       [ 8. , 2. , 3.19, ... , 0. , 66. , 1. ],  
       ... ,  
       [406. , 6. , 5.36, ... , 0. , 62. , 0. ],  
       [ 25. , 8. , 4.32, ... , 0. , 58. , 1. ],  
       [447. , 8. , 5.2 , ... , 0. , 49. , 0. ]])
```

[실습코드] 폐암환자 생존율 예측

```
[5]: # 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다.  
X = Data_set[:,0:17]  
Y = Data_set[:,17]
```

속성(정보)은 종양의 유형, 폐활량, 호흡곤란 여부,

기침, 흡연, 천식여부 등의 17가지 환자 상태이구요.

마지막 18번째는 수술 후 생존 결과로 1이면 생존,

0이면 사망입니다.

```
[30]: X[0]
```

```
[30]: array([293. , 1. , 3.8, 2.8, 0. , 0. , 0. , 0. , 0. ,  
          0. , 12. , 0. , 0. , 0. , 1. , 0. , 62. ])
```

```
[31]: Y[0]
```

```
[31]: 0.0
```

[실습코드] 폐암환자 생존율 예측

```
[8]: # 딥러닝 구조를 결정합니다(모델을 설정하고 실행하는 부분입니다).
```

```
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
[9]: # 딥러닝을 실행합니다.
```

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=100, batch_size=10)
```

```
Train on 470 samples
```

```
Epoch 1/100
```

```
470/470 [=====] - 0s 643us/sample - loss: 0.1496 - accuracy: 0.8383
```

```
Epoch 2/100
```

```
470/470 [=====] - 0s 111us/sample - loss: 0.1445 - accuracy: 0.8447
```

```
Epoch 3/100
```

```
470/470 [=====] - 0s 104us/sample - loss: 0.1450 - accuracy: 0.8511
```

```
Epoch 4/100
```

```
470/470 [=====] - 0s 104us/sample - loss: 0.1442 - accuracy: 0.8511
```

```
Epoch 5/100
```

```
470/470 [=====] - 0s 96us/sample - loss: 0.1449 - accuracy: 0.8511
```

```
Epoch 6/100
```

```
470/470 [=====] - 0s 96us/sample - loss: 0.1440 - accuracy: 0.8511
```

[실습코드] 폐암환자 생존율 예측

```
470/470 [=====] - 0s 94us/sample - loss: 0.1199 - accuracy: 0.8596
Epoch 99/100
470/470 [=====] - 0s 95us/sample - loss: 0.1254 - accuracy: 0.8574
Epoch 100/100
470/470 [=====] - 0s 98us/sample - loss: 0.1254 - accuracy: 0.8489
[9]: <tensorflow.python.keras.callbacks.History at 0x1e32887df88>
```

```
[13]: #결과 확인
print('정확도 %.4f' % model.evaluate(X, Y)[1])
```

```
470/470 [=====] - 0s 34us/sample - loss: 0.1213 - accuracy: 0.8574
정확도 0.8574
```

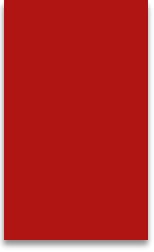
모델을 통한 검증

```
[49]: new_patient_data = np.array([X[0]])
new_patient_data
```

```
[49]: array([[293. , 1. , 3.8, 2.8, 0. , 0. , 0. , 0. , 0. ,
          0. , 12. , 0. , 0. , 0. , 1. , 0. , 62. ]])
```

```
[50]: predict = model.predict(new_patient_data)
predict
```

```
[50]: array([[0.01336104]], dtype=float32)
```



선형회귀 개요

선형회귀

- ▶ 선형회귀(Linear Regression)
 - ▶ 독립변수 x 를 사용해 종속변수 y 의 움직임을 예측하고 설명하는 일
 - ▶ 예) 공부시간에 따른 성적의 비례관계
- ▶ 통계적 관점의 목표
 - ▶ 가장 헐륭한 예측선 그리기
- ▶ 머신러닝적 관점의 목표
 - ▶ 컴퓨터 스스로 기준을 찾는 것

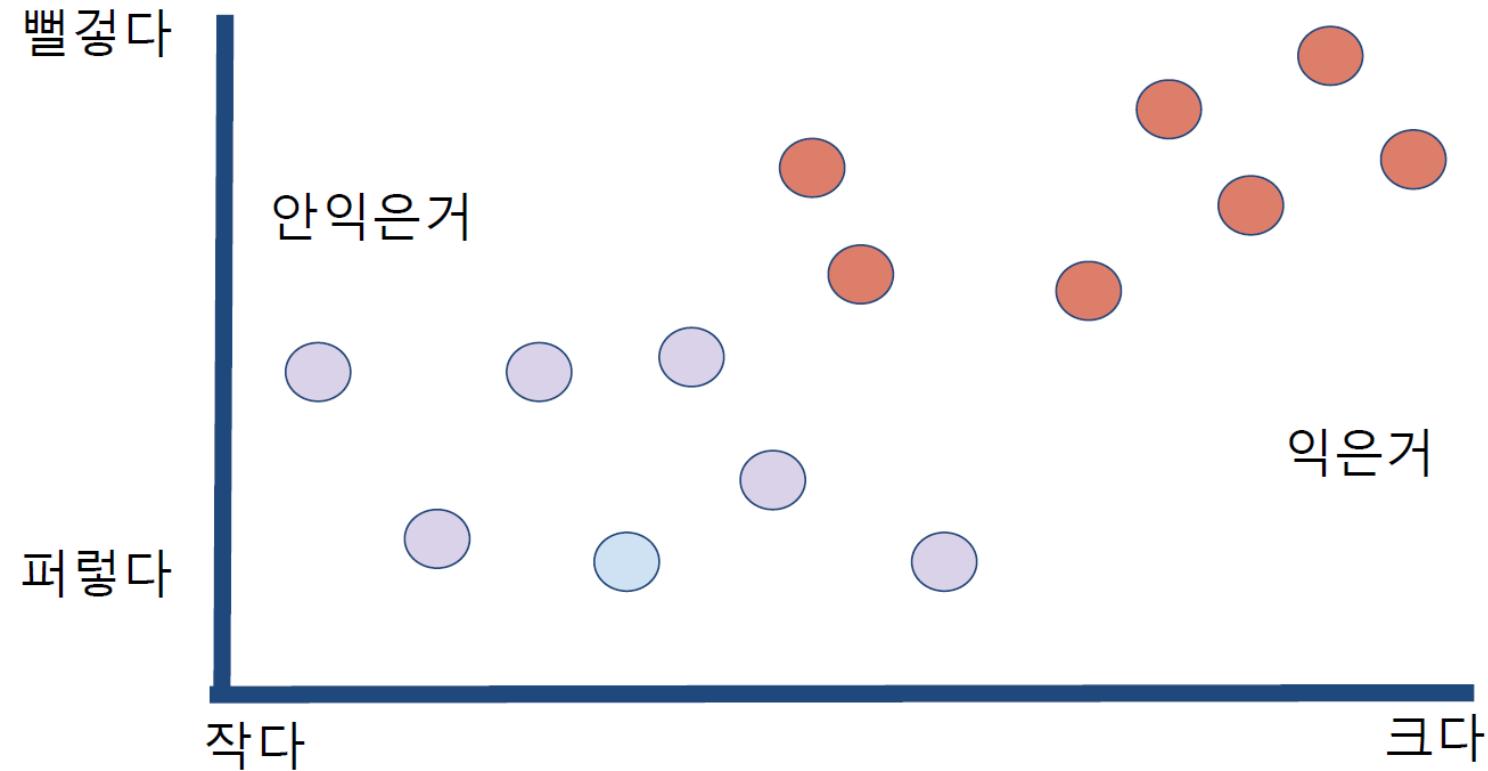
사과 익은 것 분류하기 - 사람

- ▶ 2개의 상자가 있다
 - ▶ 익은 것
 - ▶ 안 익은 것
- ▶ 사람이 분류되지 않은 사과를 사람이 분류해야 한다
- ▶ 2개 상자안의 사과를 보고 감을 잡고 분류한다.

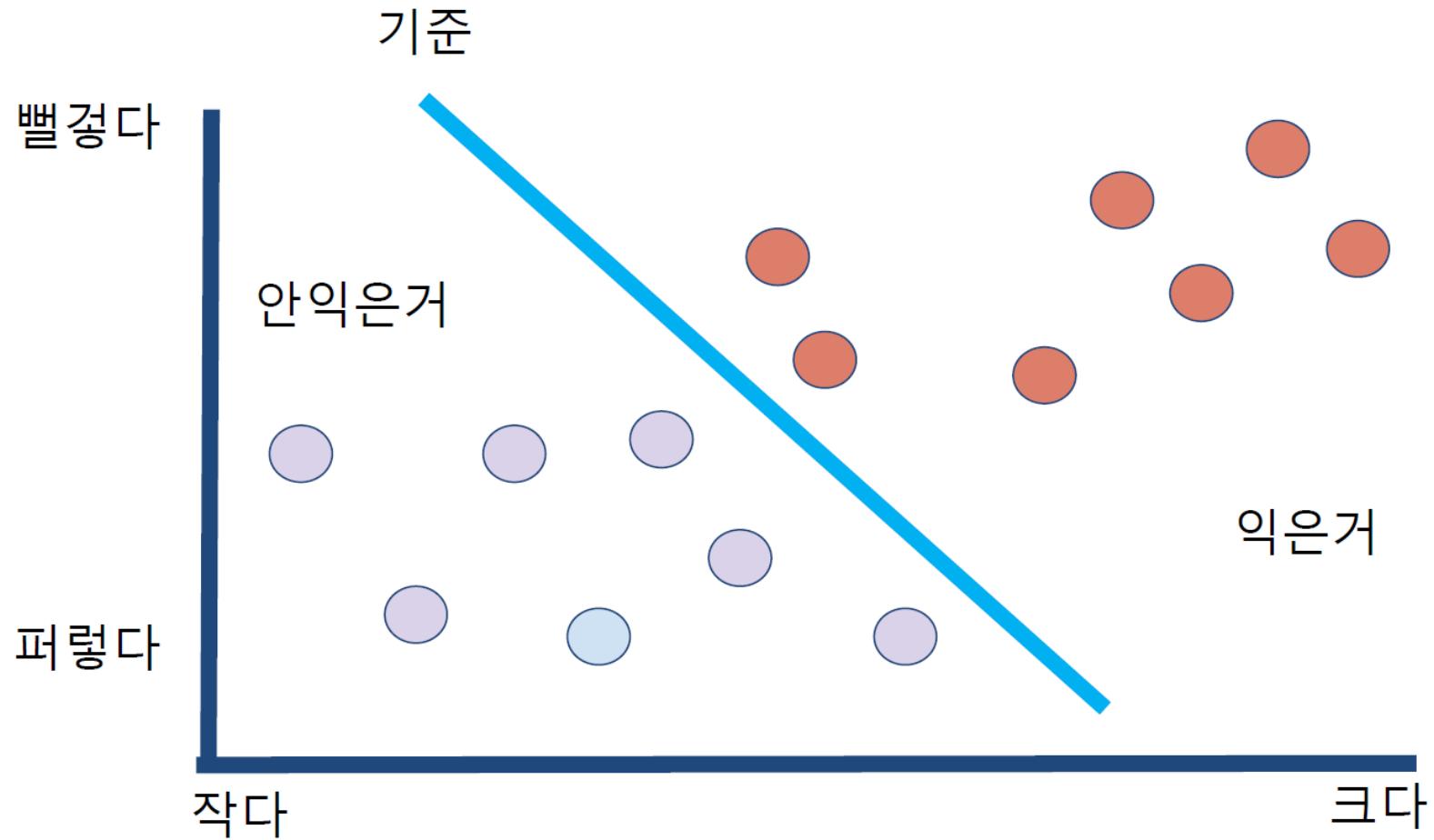
사과 익은 것 분류하기 - 로봇

- ▶ 로봇을 위한 프로그램을 작성해야 한다
 - ▶ '○○○하면 우측 상자에 그렇지 않으면 좌측 상자에'
- ▶ 분류할 때 사용한 기준을 하드 코딩해야 한다.

기준을 찾자 – 크기와 색깔을 고려



기준을 찾자 - 가르자

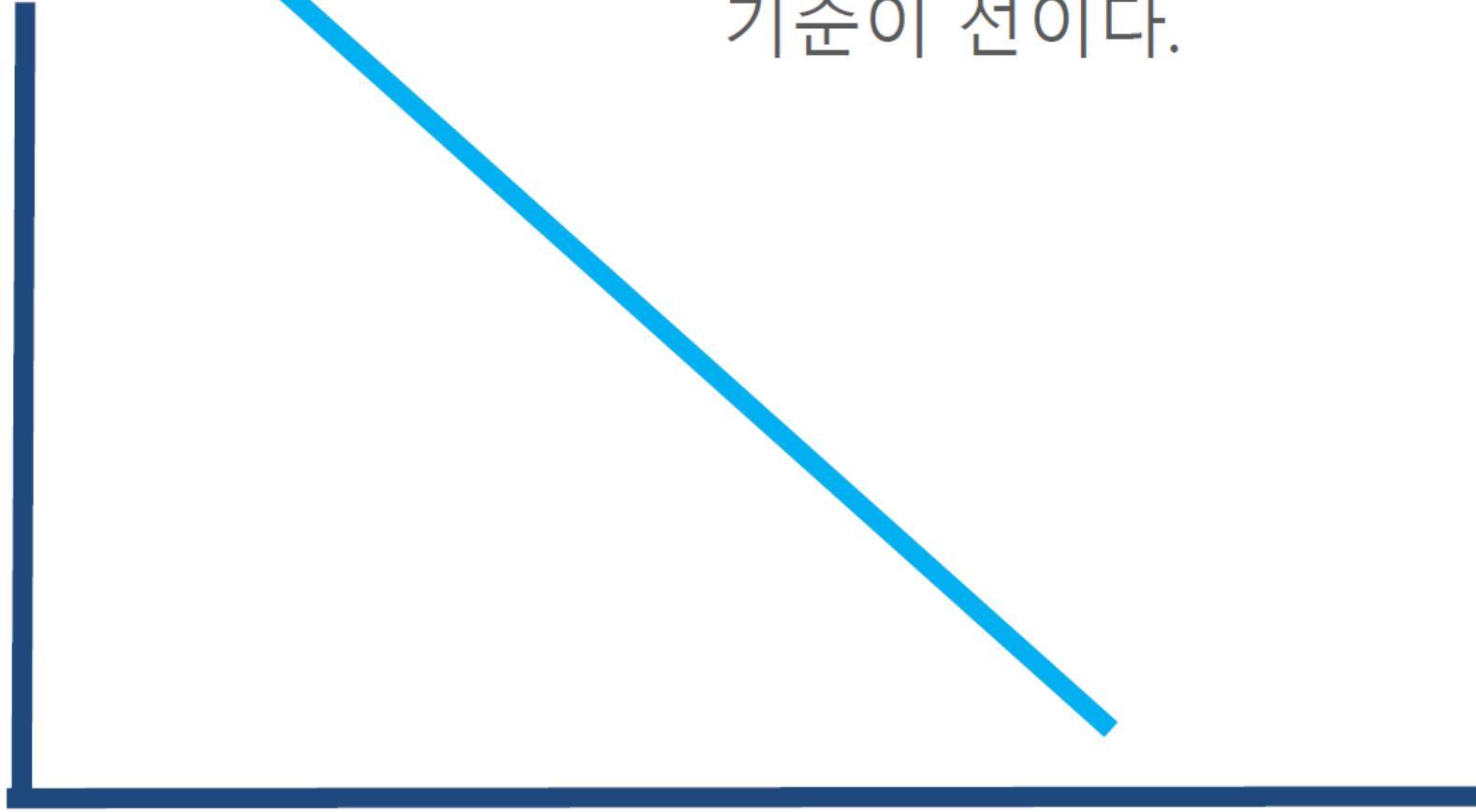


기준이 선이다



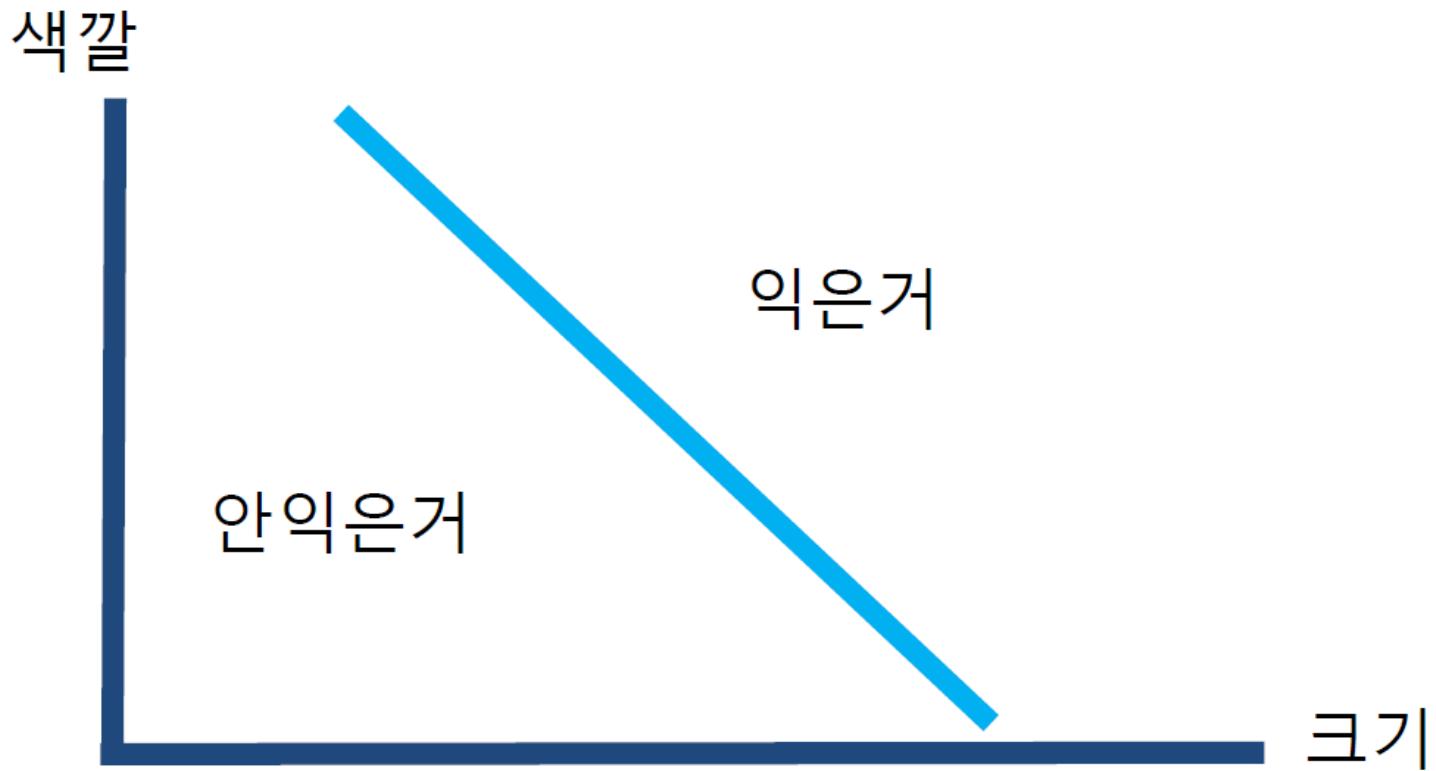
기준

기준이 선이다.



기준의 의미

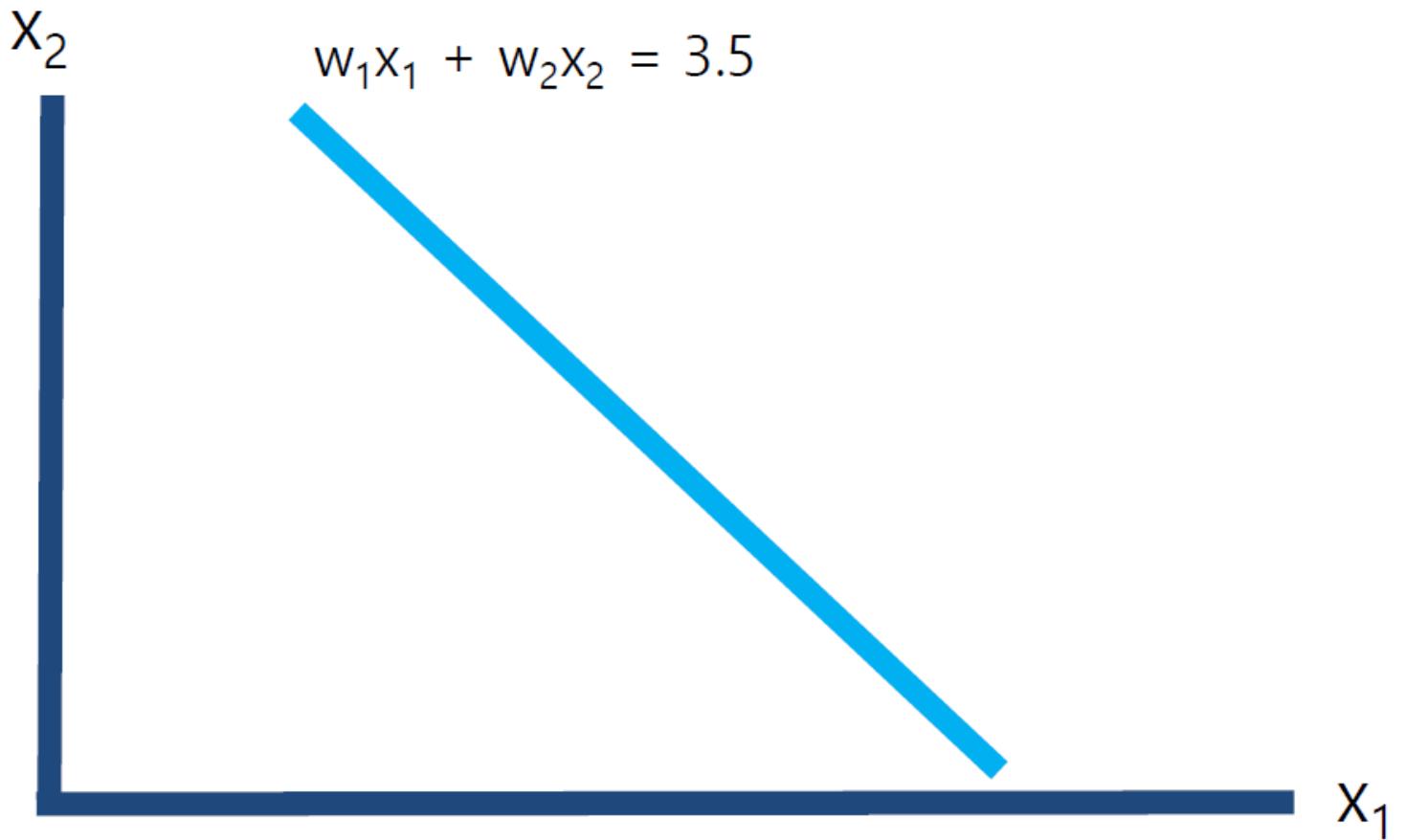
- ▶ 크기하고 색깔하고 같이 고려해서 어느정도 이상이면 익은거



선형회귀

- ▶ 선은 다음 식으로 표현

$$w_1x_1 + w_2x_2 = b$$



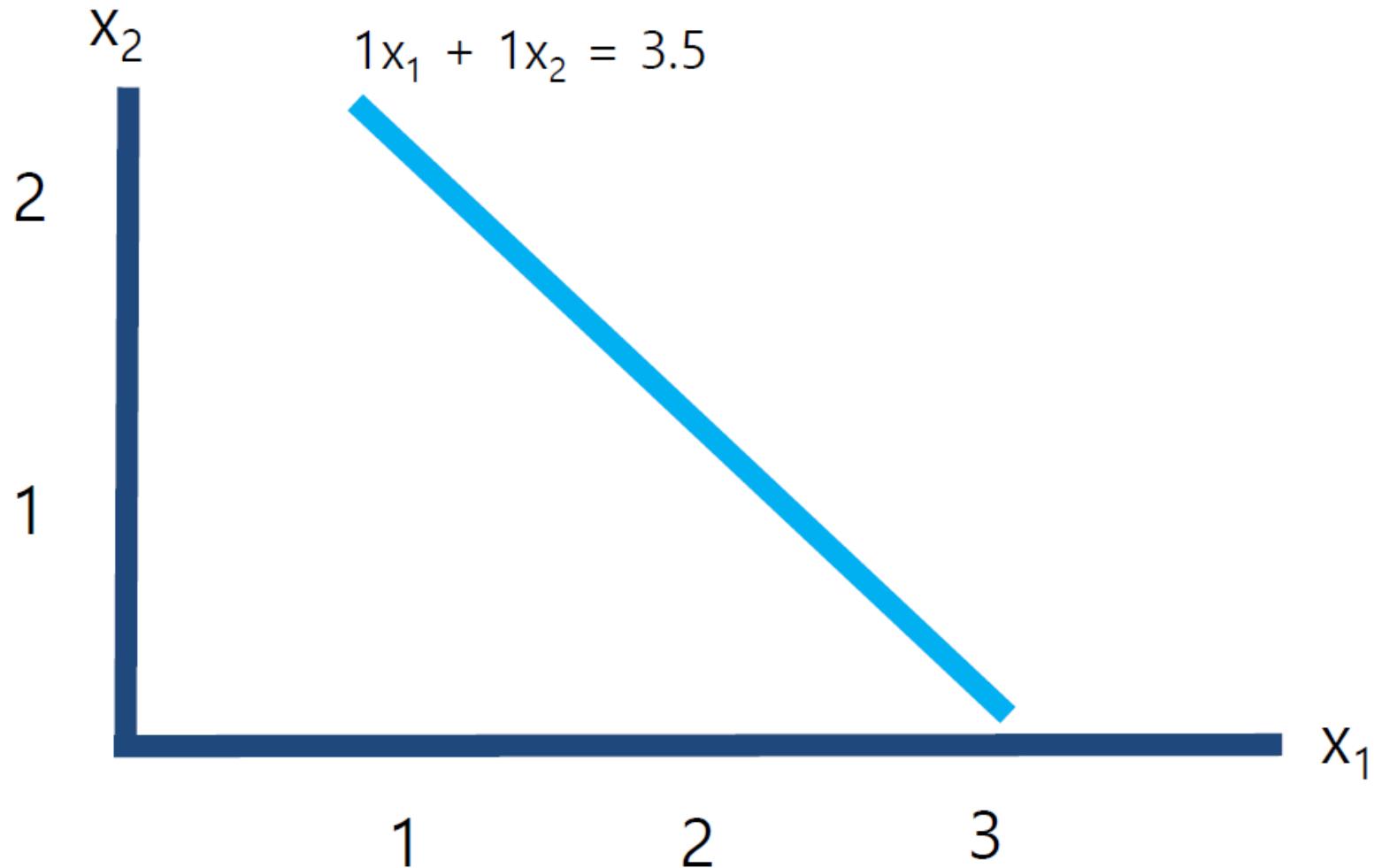
선

▶ $1x_1 + 1x_2 = 3.5$

크기 정도(x_1)하고

색깔 정도(x_2)를 합쳐서

3.5 정도인 것 찾기



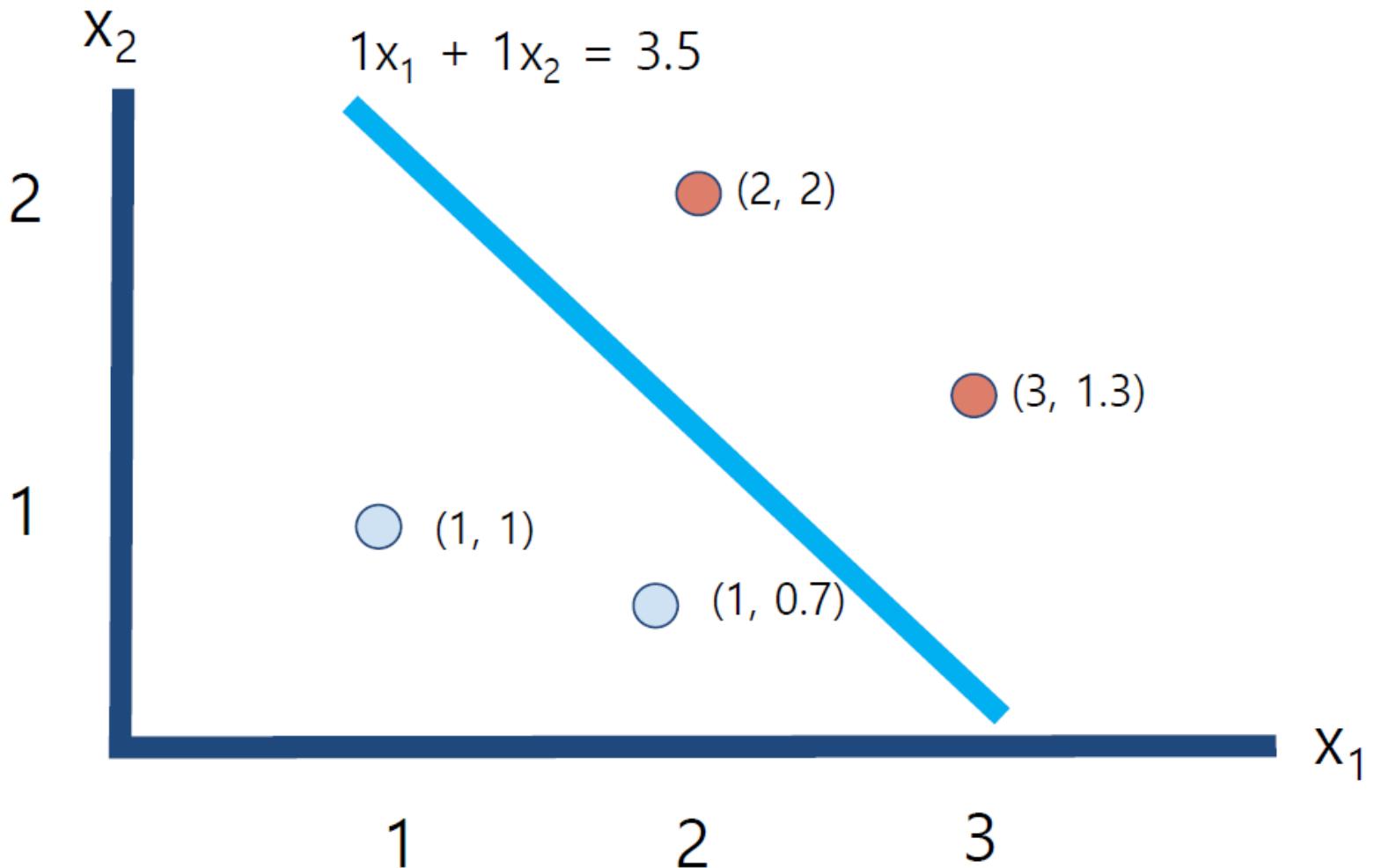
선과 점

▶ $1x_1 + 1x_2 = 3.5$

크기 정도(x_1)하고

색깔 정도(x_2)를 합쳐서

3.5 정도인 것 찾기



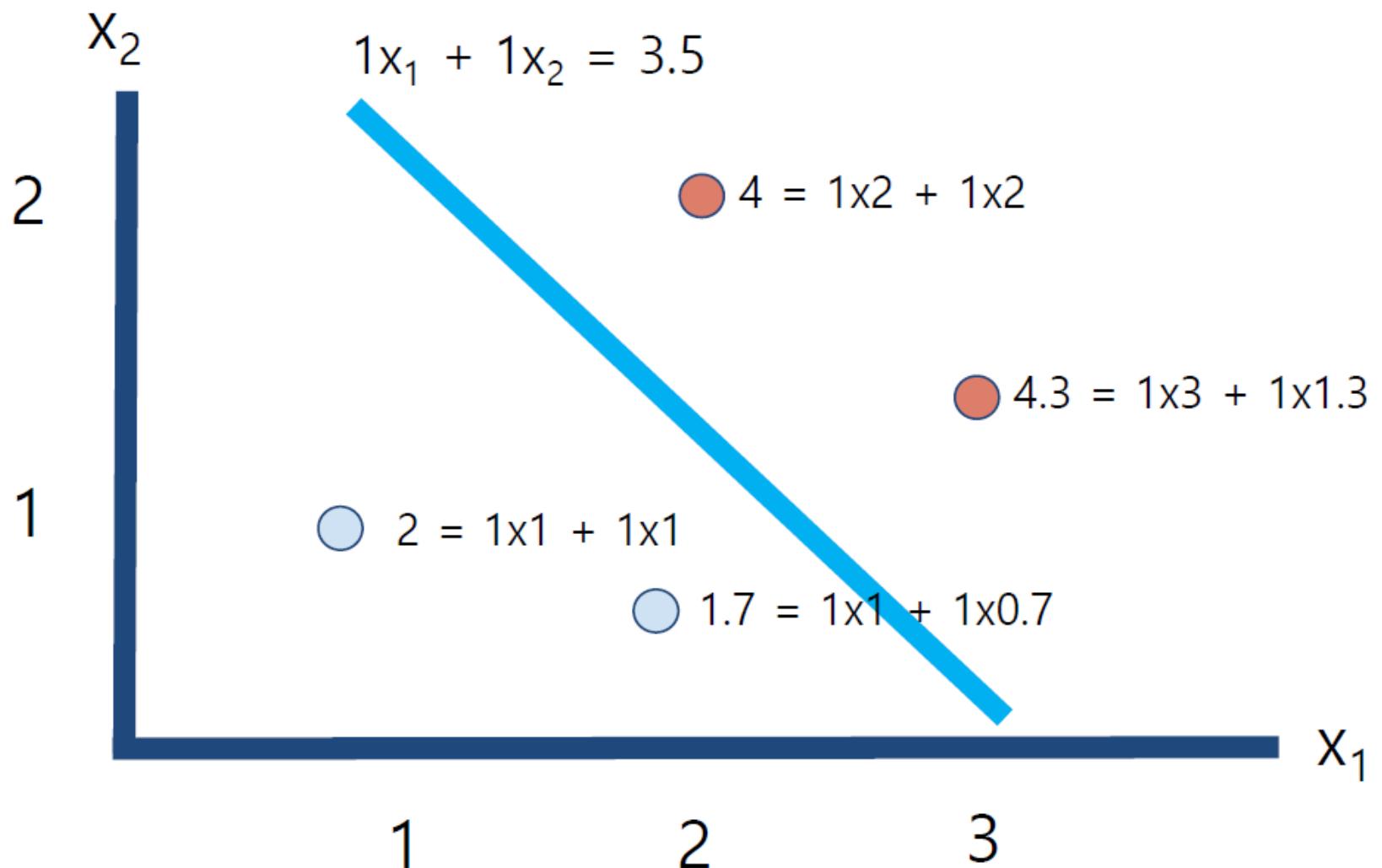
점의 값

▶ $1x_1 + 1x_2 = 3.5$

크기 정도(x_1)하고

색깔 정도(x_2)를 합쳐서

3.5 정도인 것 찾기



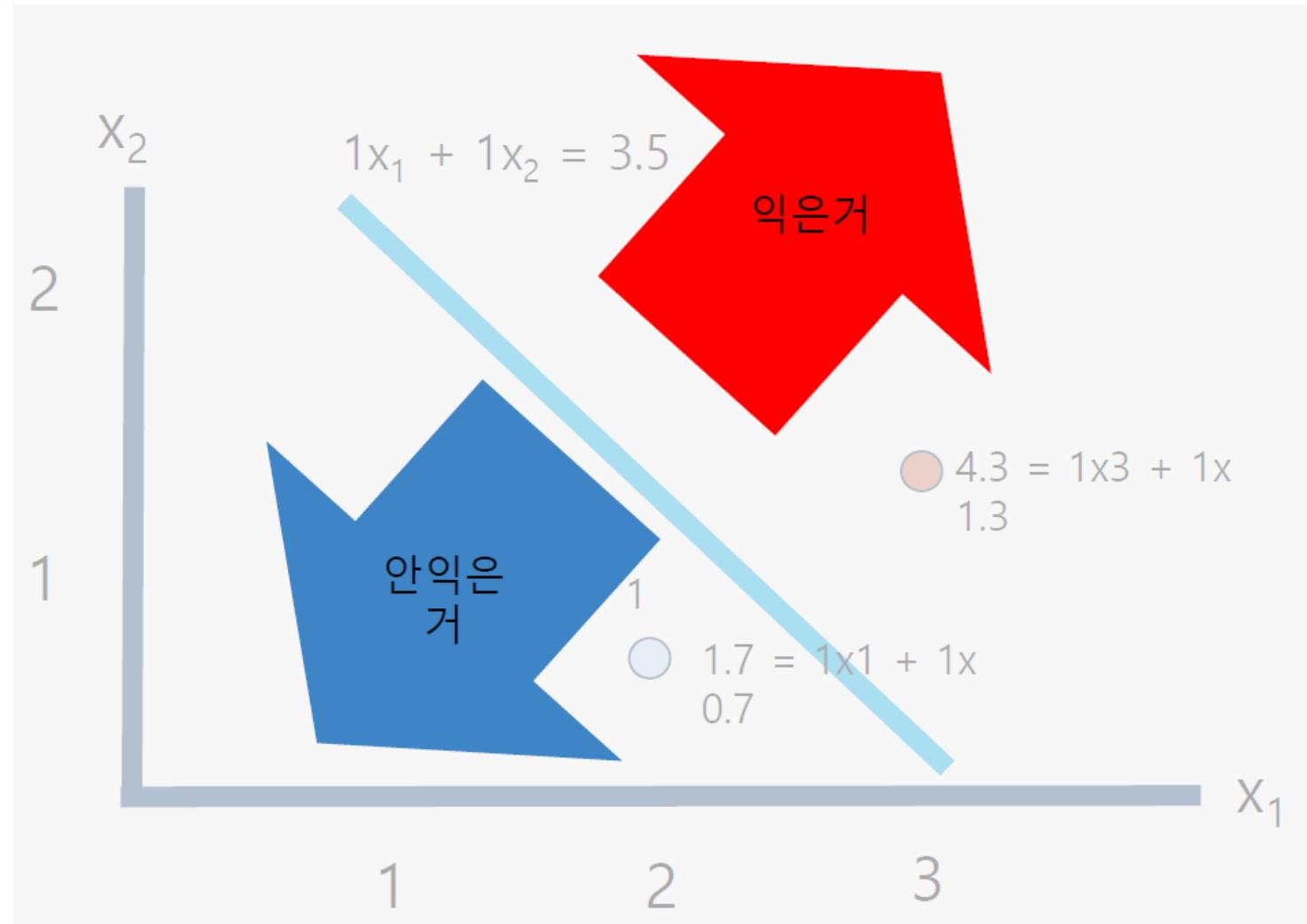
선으로 구분된 영역

▶ $1x_1 + 1x_2 = 3.5$

크기 정도(x_1)하고

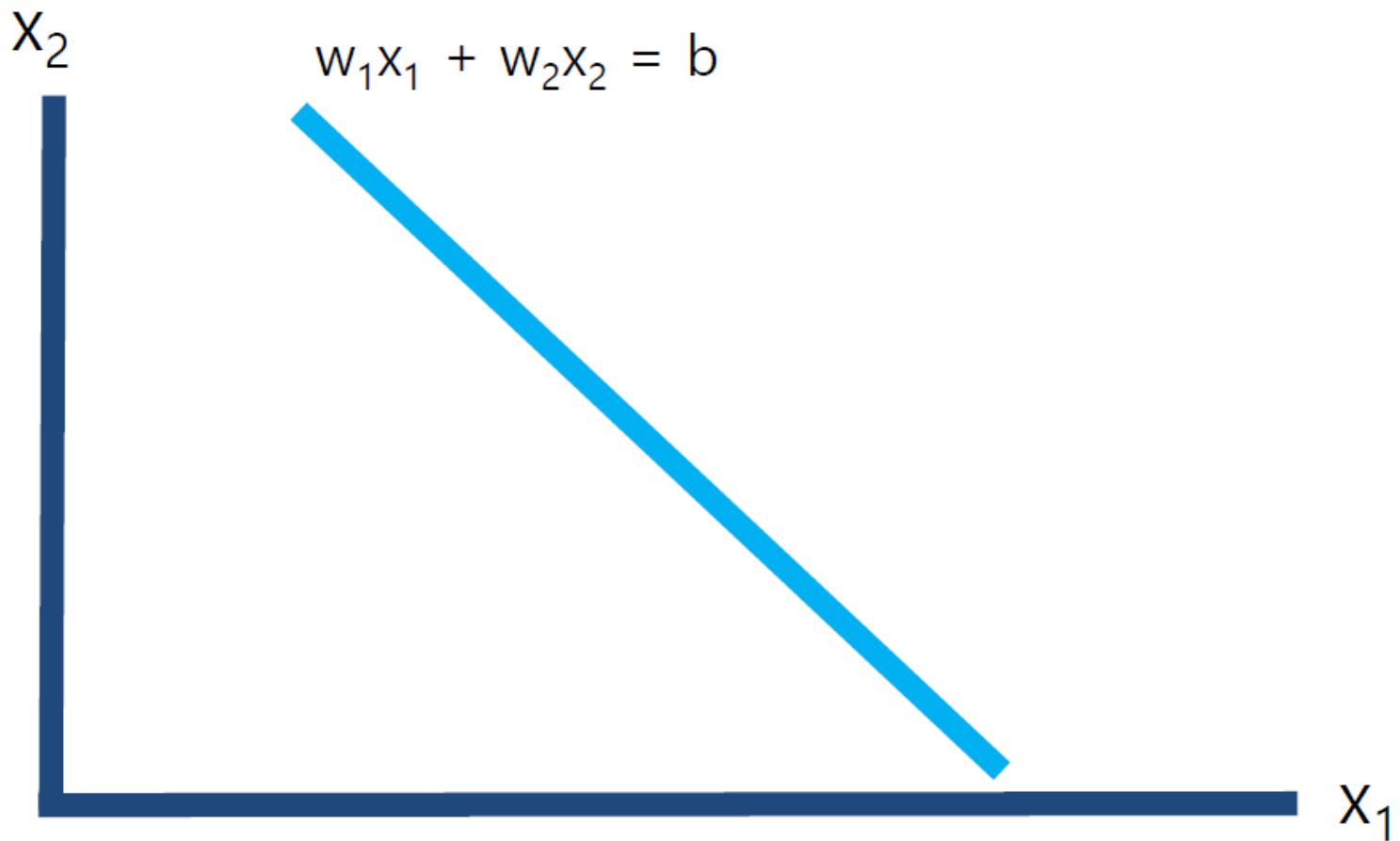
색깔 정도(x_2)를 합쳐서

3.5 정도인 것 찾기



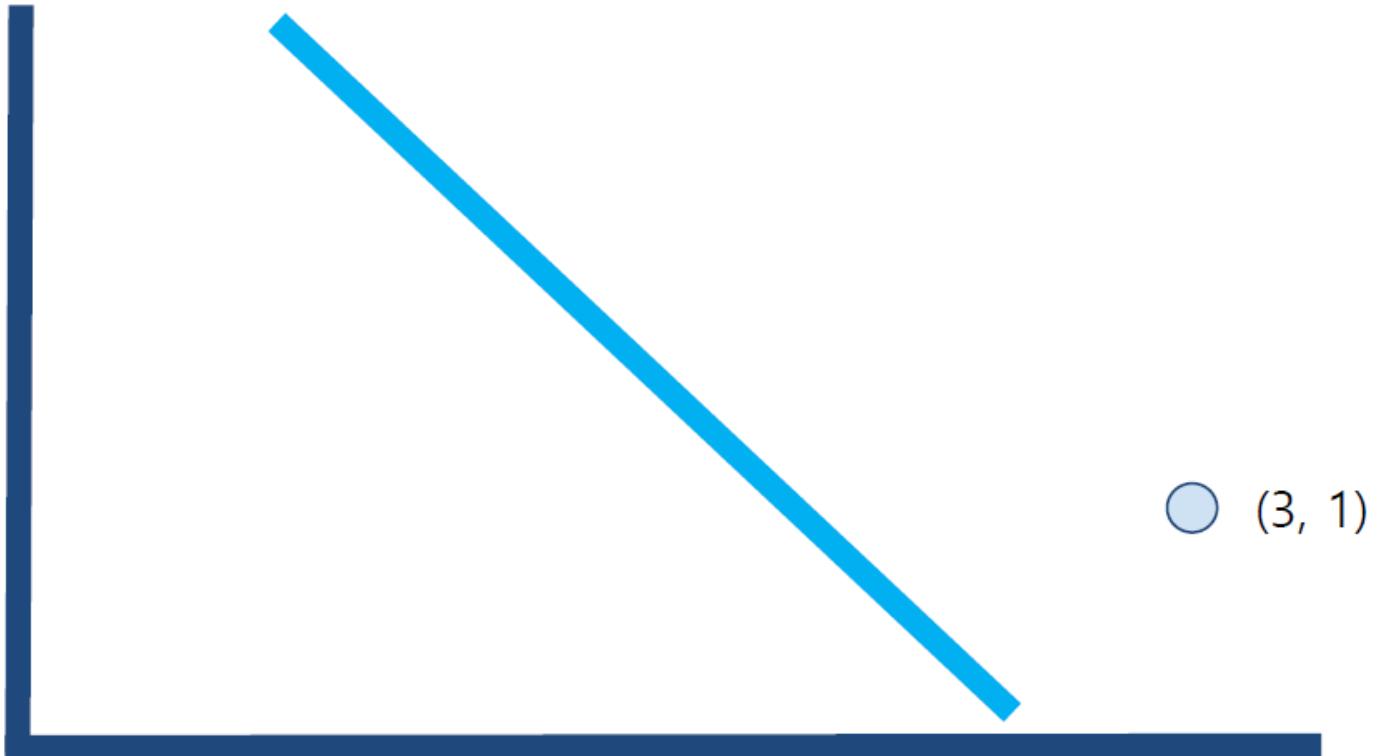
기준을 찾는다는 것

- ▶ 선을 표현하는 w_1, w_2, b 를 찾는 것
 - ▶ $w_1x_1 + w_2x_2 = b$



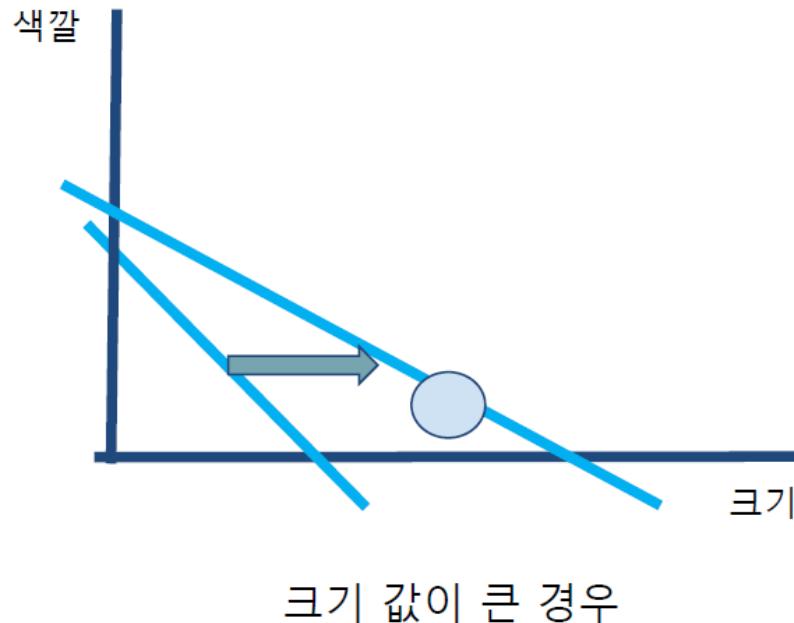
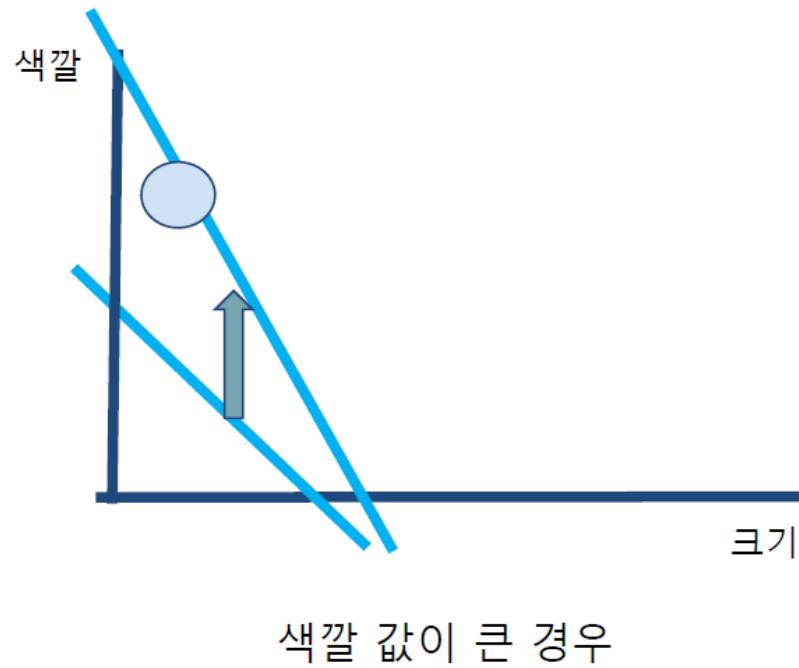
선을 찾는 방법

- ▶ 기본 아이디어 : 최종 오차는 입력 값 크기에 기인한다
- ▶ (3, 1)의 안 익은 사과의 경우 크기(3)이 색깔(1) 보다 결과에 더 영향을 끼쳤다.
- ▶ 색깔의 비중보다 크기의 비중을 더 크게 조정하자



입력 값에 비례하여 비중을 조정

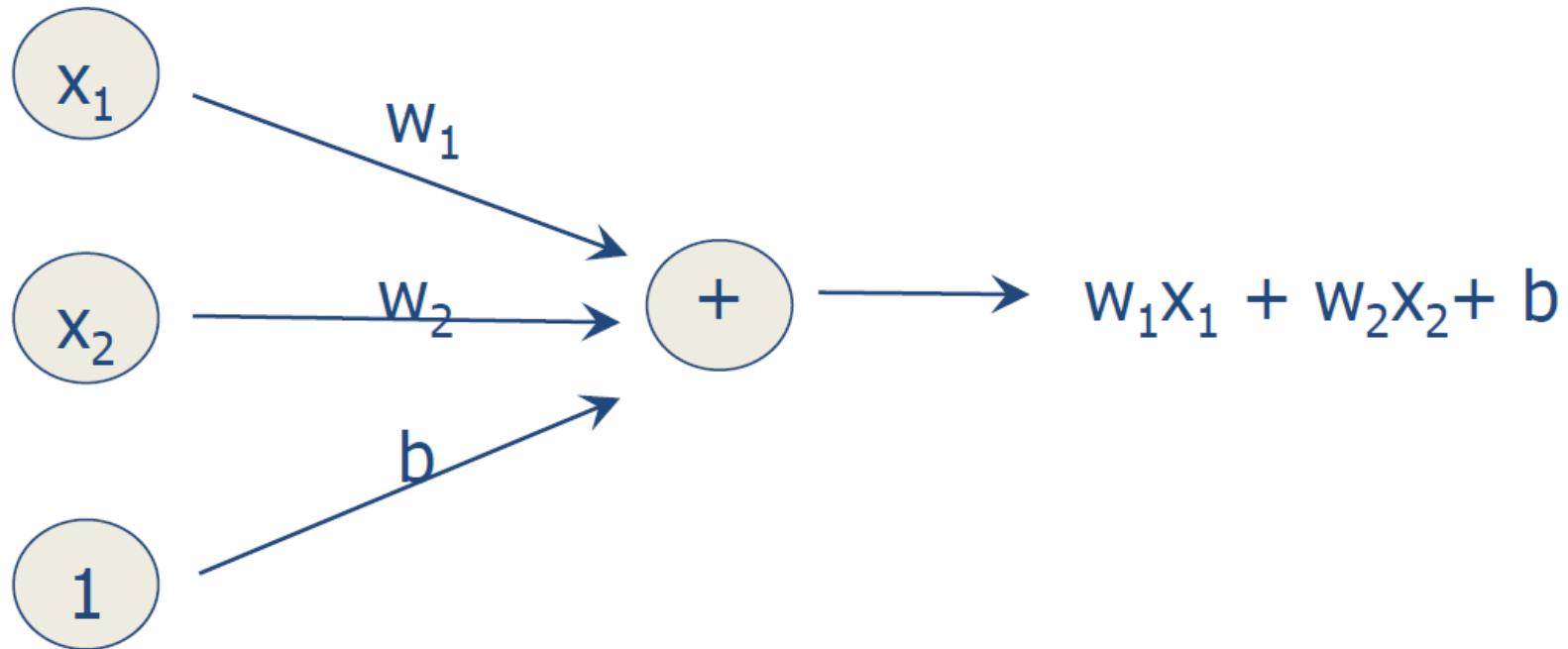
- ▶ 값이 큰 항목의 가중치를 더 크게 조정하자.



선을 학습하는 방법

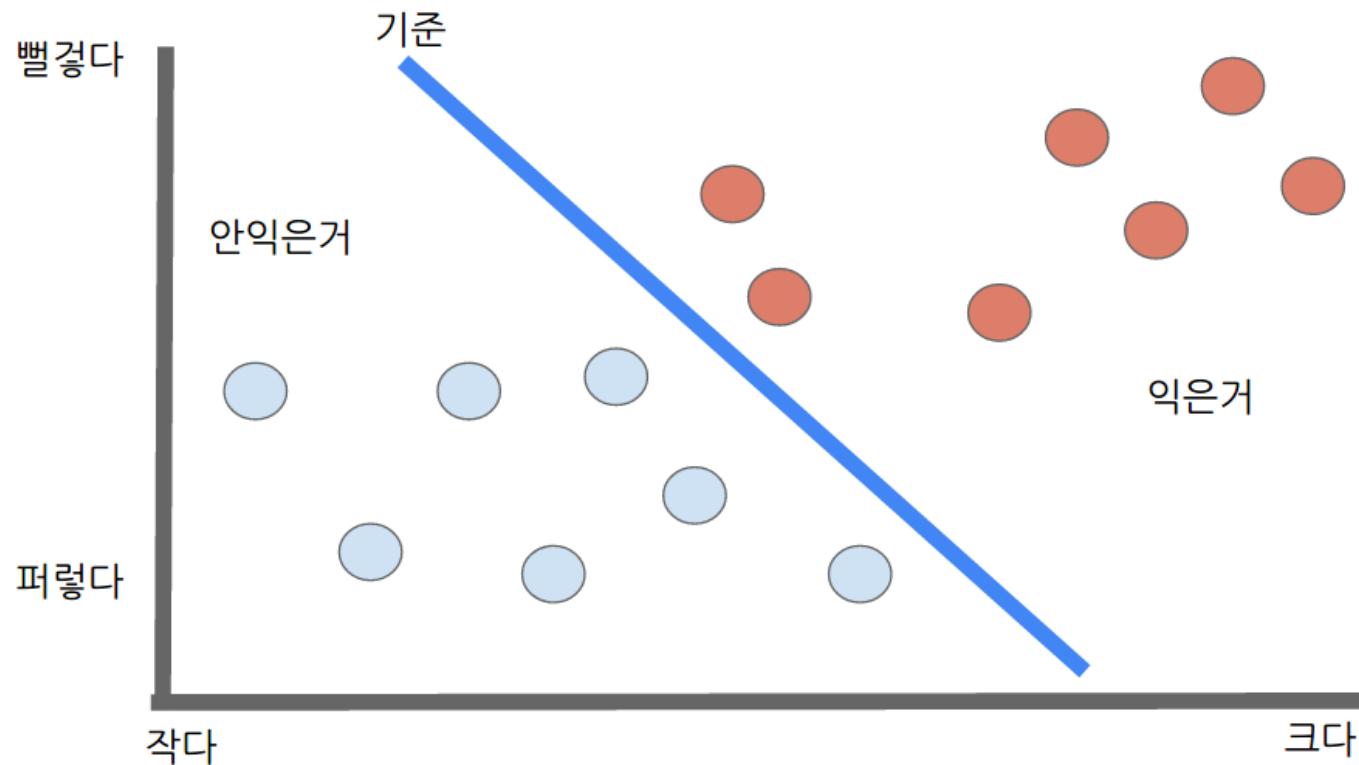
- ▶ 각 비중치를 오차와 각 입력값의 곱에 비례해서 비중치를 수정한다.
- ▶ 가중치의 변경 크기 = 입력값 * 오차 * 학습율

계산식의 다른 표현 - 그래프



선을 찾을 수 있다

- ▶ 실제로 간단한 로직으로 선을 찾을 수 있다
- ▶ 학습할 수 있다



머신러닝 용어

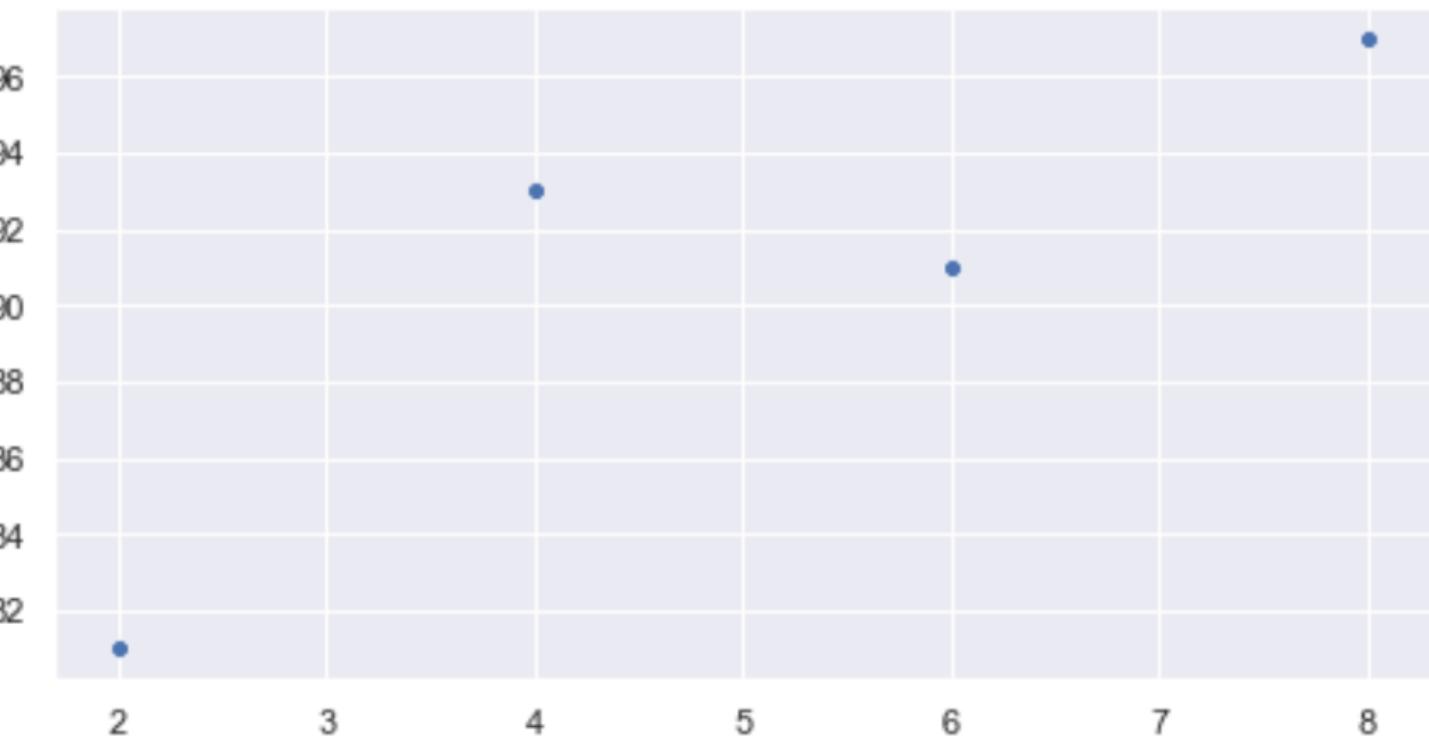
- ▶ 크기에다가 색깔을 추가하니 더 잘된다.
 - ▶ 크기, 색깔을 특질(feature)라 한다
- ▶ 각 사과마다 2개 값의 특질이 있다. 크기 3, 색깔 2
 - ▶ 특질(feature)을 벡터 혹은 입력 벡터라 한다.
- ▶ 사과 예의 경우 2개 특질(크기, 색깔)을 사용했다.
 - ▶ 입력 벡터가 2차원
- ▶ 학습을 한다는 건 선을 구성하는 3개의 값을 찾는 것이다.
- ▶ 선으로 양분한다.
 - ▶ 선형 분리(linear separating)
- ▶ 각 입력에 곱해지는 w_1, w_2 들을 가중치(weight)라 칭한다

선형회귀 실습

- ▶ 공부한 시간에 대한 점수 예측하기

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

- ▶ $X = \{2, 4, 6, 8\}$
 $Y = \{81, 93, 91, 97\}$



직선의 기울기



- ▶ 단일 선형회귀 기울기

- ▶ 일차함수

- ▶ $y = ax + b$

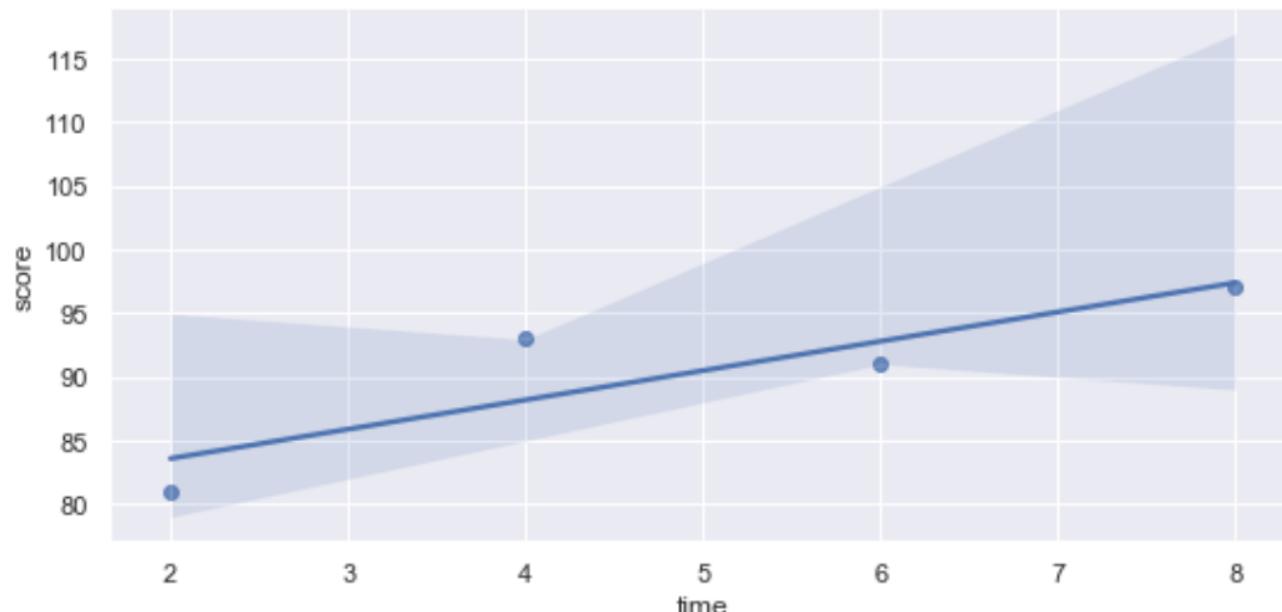
- ▶ x 독립변수(time)

- ▶ y 종속변수(score)

```
[17]: df = pd.DataFrame(np.array([[2,81],[4,93],[6,91],[8,97]]), columns=['time','score'])  
df
```

```
[17]:   time  score  
0      2     81  
1      4     93  
2      6     91  
3      8     97
```

```
[12]: sns.lmplot(x='time', y='score', data=df, height=4, aspect=2)  
plt.show()
```



일차함수 그래프

- 좌표로 변화하고 보니 수식인 일차함수로 그림을 표현할 수 있음

- $y = ax + b$

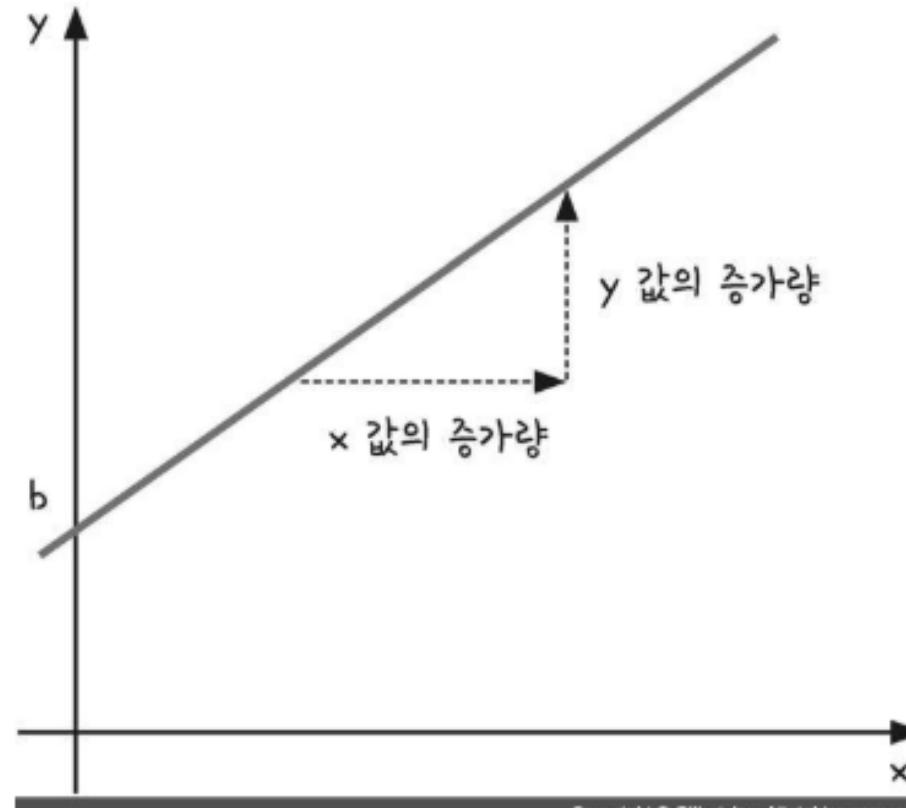
- y = 성적, 종속변수

- a = 기울기

- x = 공부시간, 독립변수

- b = 절편

- 선형회귀는 정확한 직선을 찾아가는 과정



첫번째 과제 – 정확한 직선을 찾아라

- ▶ 기울기 a 와 정확한 y 절편의 값 b 를 찾아라!!!!
- ▶ 공학적 방법 또는 지금까지 연구가 있는가???
 - ▶ 있다 -> 최소 제곱법

최소 제곱법(method of least squares)

- 회귀분석의 표준방식
- 실험이나 관찰을 통해 얻은 데이터를 분석하여 미지의 상수를 구할 때 사용하는 공식

$$a = \frac{\sum_{i=1}^n (x - \text{mean}(x))(y - \text{mean}(y))}{\sum_{i=1}^n (x - \text{mean}(x))^2}$$

$$a = \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균})^2 \text{의 합}}$$

최소 제곱법 적용

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

- ▶ 공부한 시간(x)의 평균 : $(2 + 4 + 6 + 8) / 4 = 5$
- ▶ 성적(y) 평균 : $(81 + 93 + 91 + 97) / 4 = 90.5$

$$a = \frac{(2-5)(81-90.5)+(4-5)(93-90.5)+(6-5)(91-90.5)+(8-5)(97-90.5)}{(2-5)^2+(4-5)^2+(6-5)^2+(8-5)^2}$$
$$= \frac{46}{20}$$

$$= 2.3$$

- ▶ 기울기 $a == 2.3$

$$a = \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균})^2 \text{의 합}}$$

최소 제곱법 적용

- y절편인 b를 구하는 공식

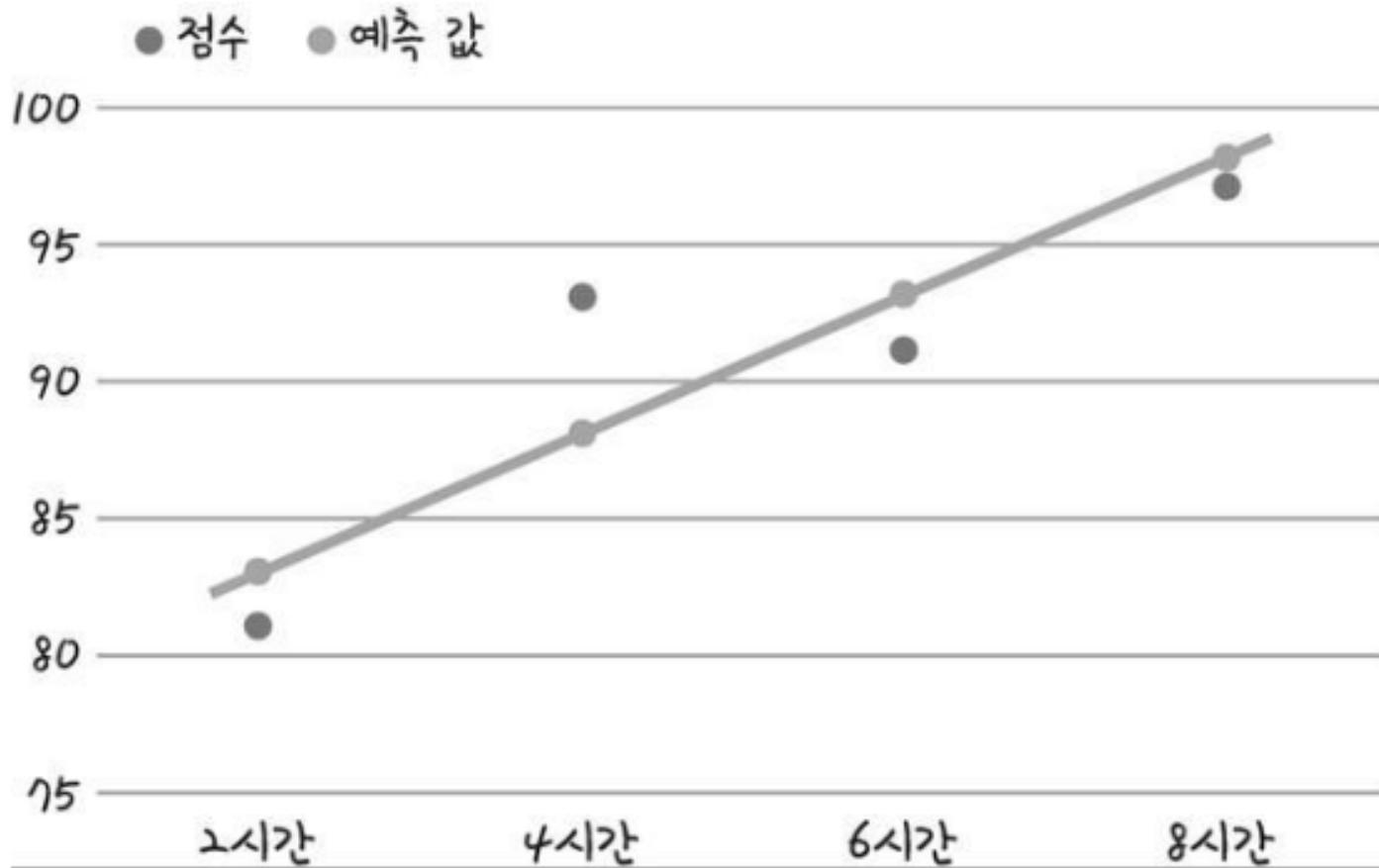
- b = mean(y) – (mean(x) * a)
- b = y의 평균 – (x의 평균 * 기울기 a)
- b = 90.5 – (2.3 * 5)
- b = 79

$$a = \frac{\sum_{i=1}^n (x - \text{mean}(x))(y - \text{mean}(y))}{\sum_{i=1}^n (x - \text{mean}(x))^2}$$

- y = 2.3x + 79

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	83.6	88.2	92.8	97.4

예측값 표현



[실습] 코딩으로 확인하는 최소 제곱법

```
[1]: import numpy as np
```

```
[2]: #x값과 y값  
x = [2, 4, 6, 8]  
y = [81, 93, 91, 97]
```

```
[3]: #x와 y의 평균값  
mx = np.mean(x)  
my = np.mean(y)  
print(f'x 시간의 평균값 : {mx}')  
print(f'y 성적의 평균값 : {my}')
```

x 시간의 평균값 : 5.0

y 성적의 평균값 : 90.5

```
[12]: #최소제곱법을 구현하기 위한 분모 만들기  
divisor = sum([(i - mx)**2 for i in x])  
divisor
```

```
[12]: 20.0
```

$$a \equiv \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균})^2 \text{의 합}}$$

[10]: #최소제곱법을 구현하기 위한 문자 만들기

```
def top(x, mx, y, my):
    d = 0
    for i in range(len(x)):
        d += (x[i] - mx) * (y[i] - my)
    return d
```

```
dividend = top(x, mx, y, my)
dividend
```

[10]: 46.0

[6]:
print('분모 : {divisor}')
print('분자 : {dividend}')

분모 : 20.0

분자 : 46.0

[7]: #기울기와 y절편 구하기
a = dividend / divisor
b = my - (mx*a)

[b 절편 구하는 공식]

$$a = \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균})^2 \text{의 합}}$$

[11]:

```
print(f'x 시간의 평균값 : {mx}')
print(f'y 성적의 평균값 : {my}')
print(f'분모 : {divisor}')
print(f'분자 : {dividend}')
print(f'기울기 a = {a}')
print(f'y 절편 b = {b}')
```

x 시간의 평균값 : 5.0

y 성적의 평균값 : 90.5

분모 : 20.0

분자 : 46.0

기울기 a = 2.3

y 절편 b = 79.0

$$b = y \text{의 평균} - (x \text{의 평균} * \text{기울기 } a)$$

앞으로 해야 하는 일

1. 기계가 스스로 기울기(가중치) a 값 **2.3** 을 찾는 일
2. 기계가 스스로 절편(바이어스) b 값 **79** 를 찾는 일

▶ 기계가 스스로 값을 찾는 마법 같은 일이 실제 가능할까???

어떻게 해야 기계가 스스로 값을 찾을 수 있을까?

- ▶ 선을 그려가면서 정답에 가깝게 찾아가는 알고리즘이 필요
- ▶ 어떤 기준으로 정답에 가깝게 찾아가야 하나??

- ▶ 오차평가 알고리즘 필요
 - ▶ 머신러닝 용어로는 손실함수(Loss Function)라 합니다.
 - ▶ 앞으로 다루게 되며 손실함수도 여러 종류가 있습니다.
 - ▶ 지금은 가장 기본적인 오차평가 방식만 다루겠습니다.

임의의 값을 주고 최적의 값과 비교

- ▶ 목표 최적의 값 : $y = 2.3x + 79$

- ▶ 임의의 값 대입 : $y = 3x + 76$

- ▶ 딥러닝에서는 랜덤값이 주어짐

- ▶ 오차 방정식

- ▶ 오차 = 실제값 - 예측값

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	83.6	88.2	92.8	97.4
오차	-2.6	4.8	-1.8	-0.4

$$\text{오차의 합} = \sum_{i=1}^n (p_i - y_i)^2$$

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	82	88	94	100
오차	1	-5	3	3

오차 구하기 공식

- 오차 제곱의 합

$$\text{오차의 합} = \sum_{i=1}^n (p_i - y_i)^2$$

- 평균 제곱 오차 (MSE)

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2$$

- 평균 제곱근 오차 (RMSE)

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2}$$

오차 값 정리

▶ $y = 3x + 76$

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	82	88	94	100
오차	1	-5	3	3

오차의 합 = $1 + (-5) + 3 + 3 = 2$ 합쳐서 0이 될 수 있다

평균 제곱 오차 = $1^2 + (-5)^2 + 3^2 + 3^2 = (1 + 25 + 9 + 9) / 4 = 44/4 = 11$

평균 제곱근 오차 = $\sqrt{1^2 + (-5)^2 + 3^2 + 3^2} = \sqrt{11} = 3.31662479036$

[실습] 코딩으로 확인하는 RMSE

▶ 평균 제곱근오차 실습

```
[1]: # 코딩으로 확인하는 평균 제곱근 오차  
import numpy as np
```

```
[2]: ab = [3, 76]  
ab
```

```
[2]: [3, 76]
```

```
[3]: data = [[2,81], [4,93], [6,91], [8,97]]  
data
```

```
[3]: [[2, 81], [4, 93], [6, 91], [8, 97]]
```

```
[4]: x = [i[0] for i in data]  
x
```

```
[4]: [2, 4, 6, 8]
```

```
[5]: y = [i[1] for i in data]  
y
```

```
[5]: [81, 93, 91, 97]
```

[실습] 코딩으로 확인하는 RMSE

```
[6]: def predict(x):
    return ab[0]*x + ab[1]
```

```
[7]: def rmse(p, a):
    return np.sqrt(((p-a) ** 2).mean())
```

```
[8]: def rmse_val(predict_result, y):
    return rmse(np.array(predict_result), np.array(y))
```

```
[9]: #예측값이 들어갈 빈 리스트
predict_result = []
```

```
[10]: #모든 x값을 한 번씩 대입하여 predict_result 리스트를 완성한다
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print(f"공부한 시간={x[i]}, 실제 점수={y[i]}, 예측 점수={predict(x[i])}")
```

공부한 시간=2, 실제 점수=81, 예측 점수=82

공부한 시간=4, 실제 점수=93, 예측 점수=88

공부한 시간=6, 실제 점수=91, 예측 점수=94

공부한 시간=8, 실제 점수=97, 예측 점수=100

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2}$$

```
[11]: #최종 RMSE 출력
print(f"rmse 최종값 : {str(rmse_val(predict_result,y))}")
```

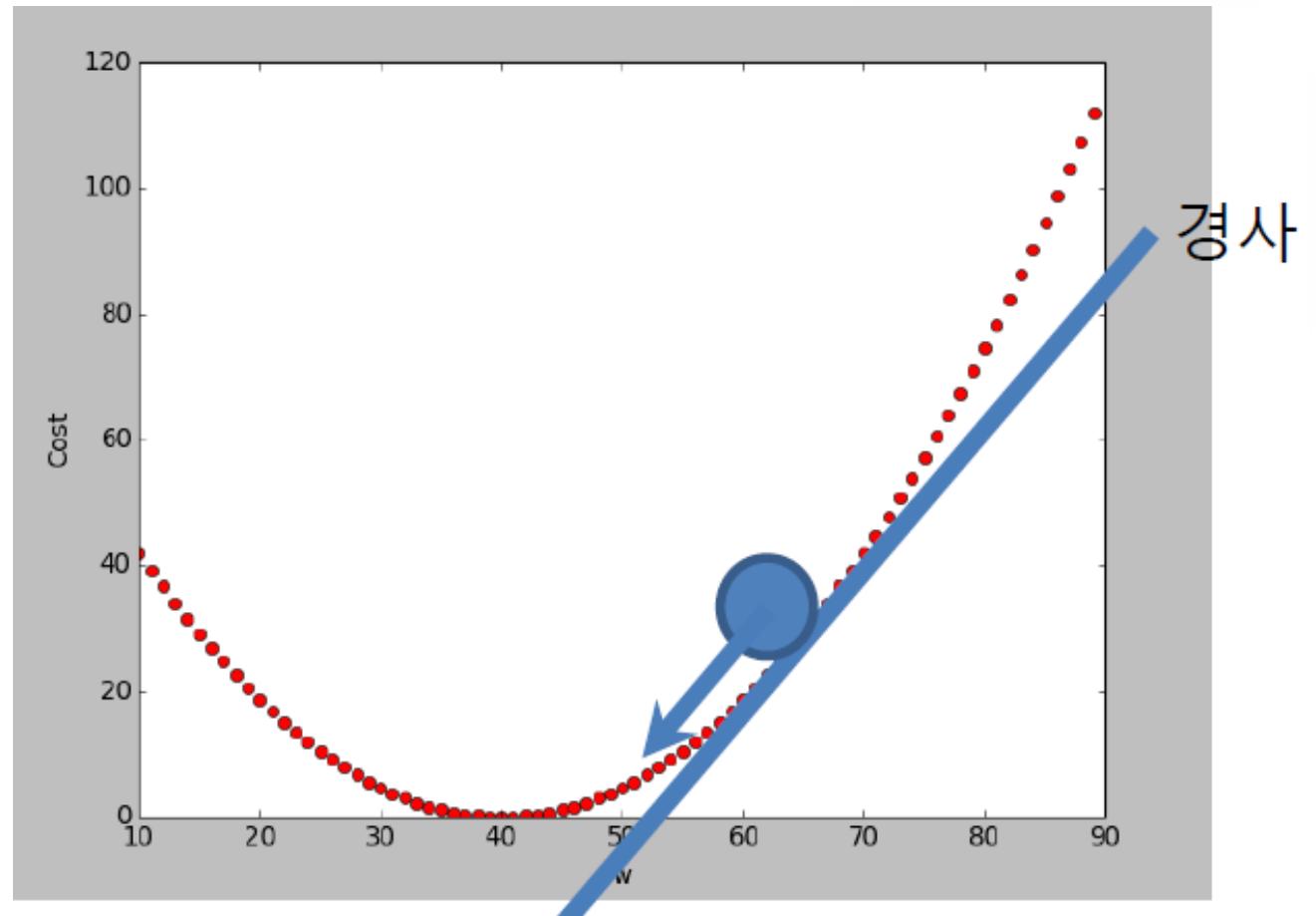
rmse 최종값 : 3.3166247903554

최적화의 기준

- ▶ 현재 코드의 결론으로 RMSE 값 3.31을 구했습니다.
- ▶ 3.31 보다 더 작은 RMSE을 가질 수 있는 a (기울기), b (절편)이 있다면 더욱 최적화된 예측선이 될 것입니다.
- ▶ 그럼 우리는 더 작은 RMSE를 가지는 a , b 를 위해 조금씩 값을 변경해보면 어떨까요?

경사 하강법(Gradient Decent)

- ▶ 오차 평면에 공을 올려 놓았을 때 공이 굴러가는 방향 (오차가 적어지는 방향)으로 가기 위해서 w 와 b 의 수정값 w_{gred} 와 b_{gred} 를 리턴
- ▶ 오차를 최소화 하기 위한 w 의 수정
 $w = w_{\text{gred}}$ 대입
 $b = b_{\text{gred}}$ 대입

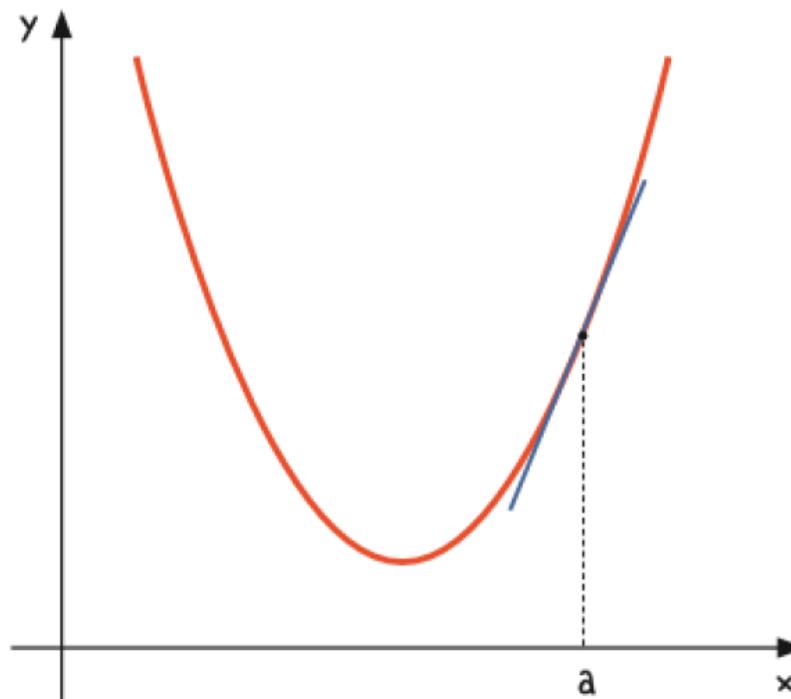


순간 변화율

- ▶ a 가 변화량이 0에 가까울 만큼 아주 미세하게 변화한다면,
- ▶ y 값의 변화 역시 아주 미세해서 0에 가까울 것
- ▶ 변화가 있긴 하지만, 그 움직임이 너무 미세하면,
- ▶ “어느 쪽으로 움직이려고 시도했다”는 정도의 느낌만 있을 뿐
- ▶ 이 느낌을 수학적으로 이름 붙인 것이 **순간변화율**

순간 변화율

- ▶ 순간변화율은 '어느 쪽' 이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선을 그릴 수가 있음
- ▶ 이 선이 바로 이 점에서의 '기울기'라고 불리는 접선



a 에서의 순간 변화율은 곧 기울기다!

미분

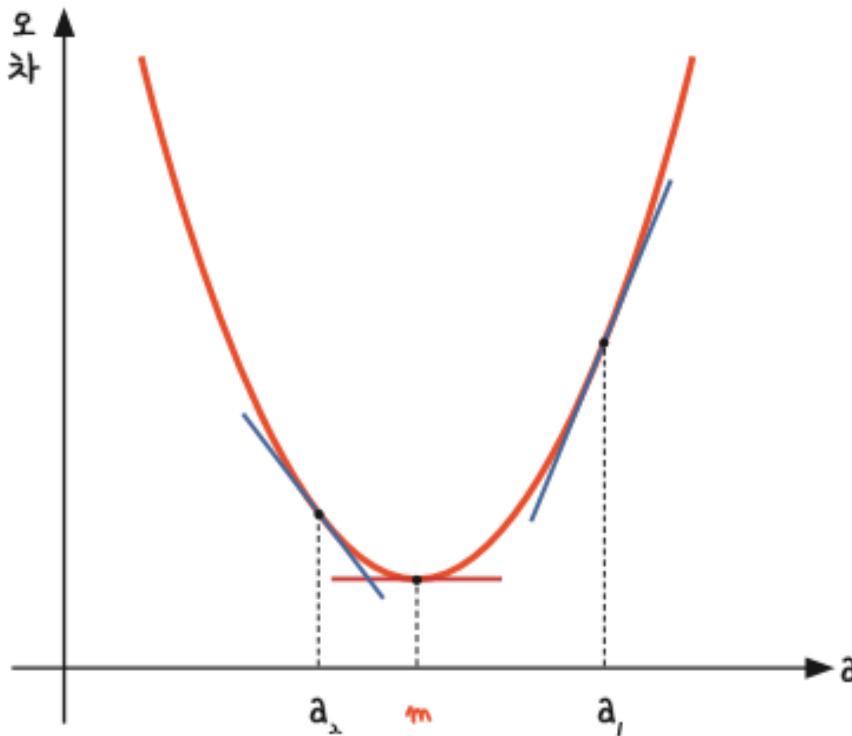
- ▶ 미분 x 값이 아주 미세하게 움직일 때의 y 변화량을 구한 뒤, 이를 x의 변화량으로 나누는 과정
- ▶ 한 점에서의 순간 기울기
- ▶ “함수 $f(x)$ 를 미분하라” 는 $\frac{d}{dx}f(x)$ 표기함.

$$\frac{d}{dx}f(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

① 함수 $f(x)$ 를 x 로 미분하라는 것은
② x 의 변화량이 0에 가까울 만큼 작을 때
③ y 변화량의 차이를
④ x 변화량으로 나눈 값(=순간 변화율)을 구하라는 뜻!

미분

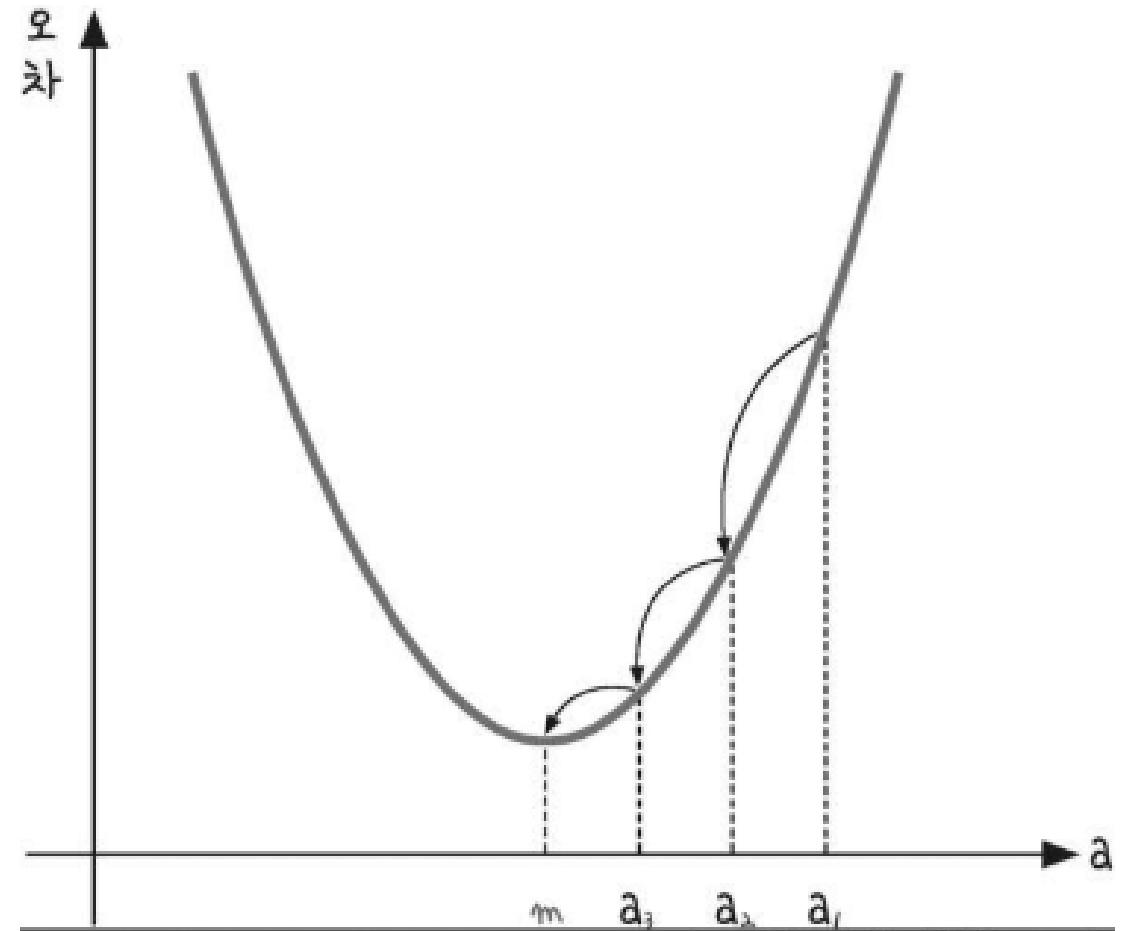
- ▶ $y = x^2$ 그래프에서 x 에 다음과 같이 a_1, a_2 그리고 m 을 대입하여 그 자리에서 미분하면 각점에서의 순간 기울기가 그려집니다



순간 기울기가 0인 점이 곧 우리가 찾는 최소값입니다

경사 하강법

- ▶ 기울기 a 를 크게 잡거나 작게 잡아도 **오차가 커짐을 확인**
- ▶ 기울기 a 와 오차 사이에는 **상관관계**가 있다
- ▶ 오차가 가장 작은 점은???
 - ▶ 평형이 되는 m
- ▶ **오차를 비교하여 가장 작은 방향으로 이동시키는 방법**



[실습] 임의의 값에서 최적 값 변환

- ▶ 경사 하강법을 통한 자동 최적 값 찾기

- ▶ 임의의 값 : $y = 3x + 76$

- ▶ 목표 최적의 값 : $y = 2.3x + 79$

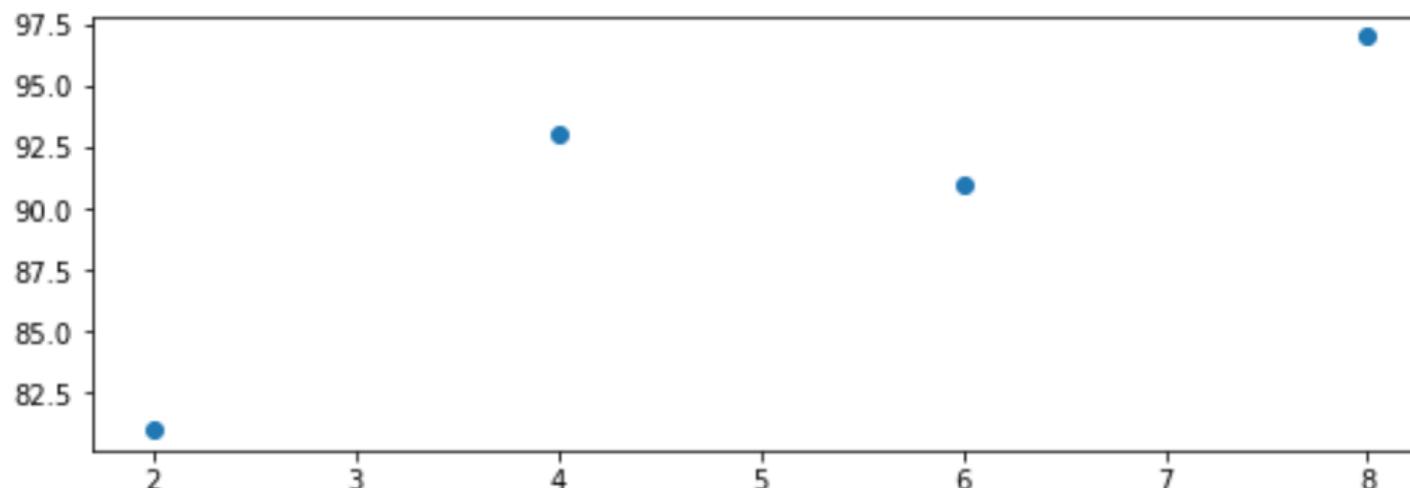
- ▶ 경사 하강법 공식

[실습] 임의의 값에서 최적 값 변환

```
[1]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt
```

```
[2]: #공부시간 X와 성적 Y의 리스트를 만듭니다.  
data = ([2,81], [4,93], [6,91], [8,97]) #공부시간, 실제성적  
x = [2, 4, 6, 8]  
y = [i[1] for i in data] #81, 93, 91, 97
```

```
[9]: #그래프로 그리기  
plt.figure(figsize=(9,3)) #넓이, 높이  
plt.scatter(x,y) #산점도  
plt.show()
```



[실습] 임의의 값에서 최적 값 변환

- ```
[4]: #리스트로 되어있는 x와 y의 넘파이 배열로 바꾸어 줍니다.
x_data = np.array(x) #기본적으로 파이썬은 리스트 --> 객체 , np.array --> 배열, 같은타입, 메모리, 속도
y_data = np.array(y)
```
- ```
[5]: #기울기 a와 절편 b의 값을 초기화 하자  
a = 3  
b = 76
```
- ```
[6]: #학습률
#한번 학습할 때 얼마큼 이동할 것인가?
lr = 0.01
```
- ```
[7]: epoch = 8000
```

[실습] 임의의 값에서 최적 값 변환

[8]: #경사 하강법을 시작합니다.

```
for i in range(epoch):
    y_hat = a * x_data + b #y를 구하는 식을 세웁니다. , y_hat은 예측값 hypothesis(가설)
    error = y_data - y_hat #오차 error = 실제값 - 예측값
    a_diff = -(1/len(x_data)) * sum(x_data * (error)) # 오차함수를 a로 미분한 값, a는 가중치, 기울기
    b_diff = -(1/len(x_data)) * sum(error) #오차함수를 b로 미분한 값
    a = a - (lr * a_diff) # 학습률을 곱해 기존의 a값을 업데이트합니다.
    b = b - (lr * b_diff) # 학습률을 곱해 기존의 b값을 업데이트합니다.
    if i % 100 == 0 : #100번 반복할 때마다 현재의 a값, b값을 출력합니다.
        print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))
```

epoch=0, 기울기=2.9400, 절편=75.9950

epoch=100, 기울기=2.7313, 절편=76.4262

epoch=200, 기울기=2.6667, 절편=76.8117

epoch=300, 기울기=2.6118, 절편=77.1395

epoch=400, 기울기=2.5651, 절편=77.4182

epoch=500, 기울기=2.5254, 절편=77.6551

[실습] 임의의 값에서 최적 값 변환

```
epoch=0, 기울기=2.9400, 절편=75.9950
epoch=100, 기울기=2.7313, 절편=76.4262
epoch=200, 기울기=2.6667, 절편=76.8117
epoch=300, 기울기=2.6118, 절편=77.1395
epoch=400, 기울기=2.5651, 절편=77.4182
epoch=500, 기울기=2.5254, 절편=77.6551
epoch=600, 기울기=2.4916, 절편=77.8566
epoch=700, 기울기=2.4629, 절편=78.0278
epoch=800, 기울기=2.4385, 절편=78.1735
epoch=900, 기울기=2.4178, 절편=78.2973
epoch=1000, 기울기=2.4001, 절편=78.4025
epoch=1100, 기울기=2.3851, 절편=78.4920
epoch=1200, 기울기=2.3724, 절편=78.5681
epoch=1300, 기울기=2.3615, 절편=78.6328
epoch=1400, 기울기=2.3523, 절편=78.6878
epoch=1500, 기울기=2.3445, 절편=78.7346
epoch=1600, 기울기=2.3378, 절편=78.7743
epoch=1700, 기울기=2.3322, 절편=78.8081
epoch=1800, 기울기=2.3273, 절편=78.8369
epoch=1900, 기울기=2.3232, 절편=78.8613
epoch=2000, 기울기=2.3198, 절편=78.8821
epoch=2100, 기울기=2.3168, 절편=78.8997
epoch=2200, 기울기=2.3143, 절편=78.9148
epoch=2300, 기울기=2.3121, 절편=78.9275
epoch=2400, 기울기=2.3103, 절편=78.9384
epoch=2500, 기울기=2.3088, 절편=78.9476
epoch=2600, 기울기=2.3075, 절편=78.9555
epoch=2700, 기울기=2.3063, 절편=78.9621
epoch=2800, 기울기=2.3054, 절편=78.9678
epoch=2900, 기울기=2.3046, 절편=78.9726
```

```
epoch=5000, 기울기=2.3002, 절편=78.9991
epoch=5100, 기울기=2.3001, 절편=78.9992
epoch=5200, 기울기=2.3001, 절편=78.9993
epoch=5300, 기울기=2.3001, 절편=78.9994
epoch=5400, 기울기=2.3001, 절편=78.9995
epoch=5500, 기울기=2.3001, 절편=78.9996
epoch=5600, 기울기=2.3001, 절편=78.9997
epoch=5700, 기울기=2.3000, 절편=78.9997
epoch=5800, 기울기=2.3000, 절편=78.9998
epoch=5900, 기울기=2.3000, 절편=78.9998
epoch=6000, 기울기=2.3000, 절편=78.9998
epoch=6100, 기울기=2.3000, 절편=78.9998
epoch=6200, 기울기=2.3000, 절편=78.9999
epoch=6300, 기울기=2.3000, 절편=78.9999
epoch=6400, 기울기=2.3000, 절편=78.9999
epoch=6500, 기울기=2.3000, 절편=78.9999
epoch=6600, 기울기=2.3000, 절편=78.9999
epoch=6700, 기울기=2.3000, 절편=78.9999
epoch=6800, 기울기=2.3000, 절편=79.0000
epoch=6900, 기울기=2.3000, 절편=79.0000
epoch=7000, 기울기=2.3000, 절편=79.0000
epoch=7100, 기울기=2.3000, 절편=79.0000
epoch=7200, 기울기=2.3000, 절편=79.0000
epoch=7300, 기울기=2.3000, 절편=79.0000
epoch=7400, 기울기=2.3000, 절편=79.0000
epoch=7500, 기울기=2.3000, 절편=79.0000
epoch=7600, 기울기=2.3000, 절편=79.0000
epoch=7700, 기울기=2.3000, 절편=79.0000
epoch=7800, 기울기=2.3000, 절편=79.0000
epoch=7900, 기울기=2.3000, 절편=79.0000
```

다중 선형 회귀

- ▶ 독립변수 $x_1, x_2, x_3 \dots x_n$ 으로 이루어진 선형 회귀
- ▶ 프레임워크 없이 파이썬만 사용하여 구현 하는 실습
- ▶ 시간, 공부 상관관계 속에 과외 실습내용을 삽입 해봄

공부한 시간	2시간	4시간	6시간	8시간
과외수업 횟수	0회	4회	2회	3회
성적	81점	93점	91점	97점

[실습]과외수업이 추가된 성적예측

공부한 시간	2시간	4시간	6시간	8시간
과외수업 횟수	0회	4회	2회	3회
성적	81점	93점	91점	97점

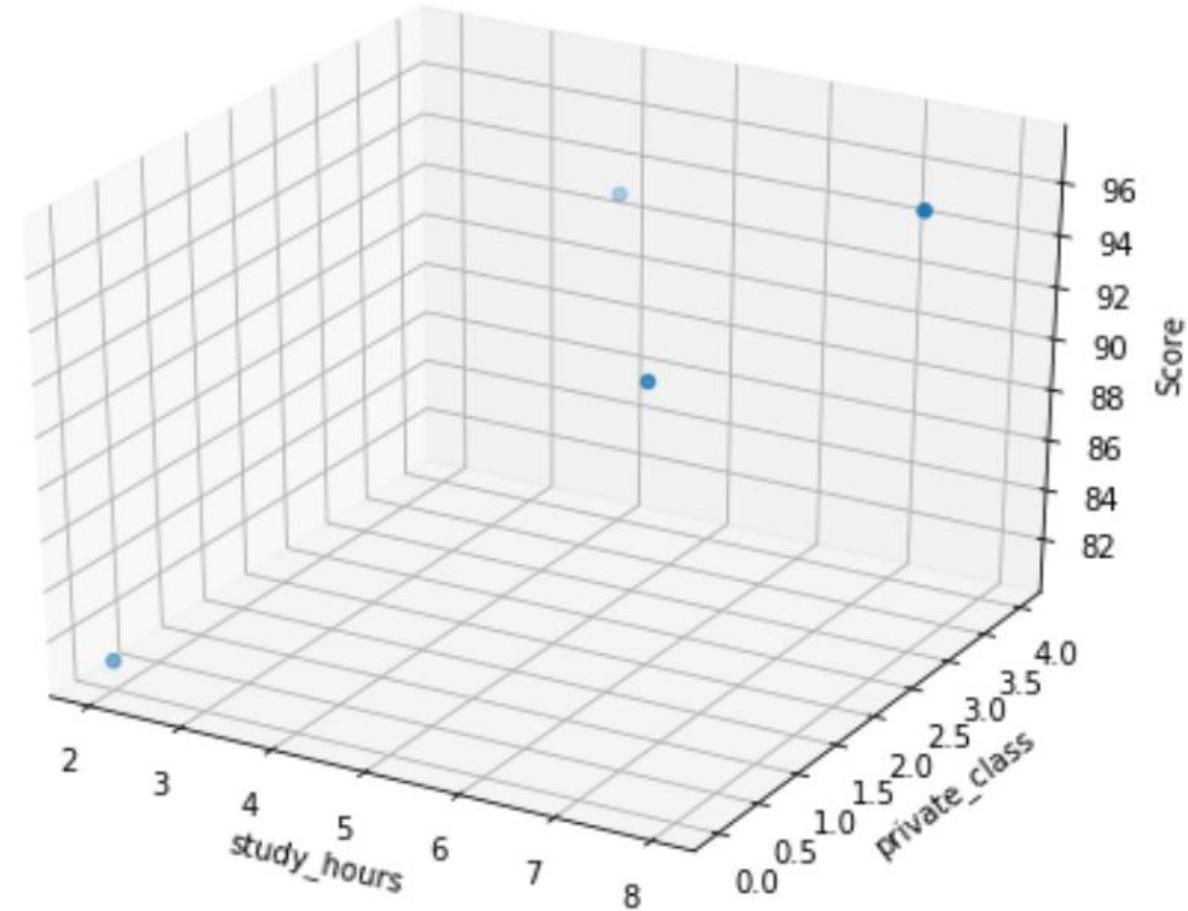
```
[1]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from mpl_toolkits import mplot3d
```

```
[2]: #공부시간 X와 성적 Y의 리스트를 만듭니다.  
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]  
x1 = [i[0] for i in data]  
x2 = [i[1] for i in data]  
y = [i[2] for i in data]
```

[실습]과외수업이 추가된 성적예측

[12]: #그래프로 확인해 봅니다.

```
plt.figure(figsize=(8,6))
ax = plt.axes(projection='3d')
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.dist = 10
#ax.plot(x1,x2,y) #3차원직선
ax.scatter(x1, x2, y)
plt.show()
```



[실습] 과외수업이 추가된 성적예측

```
[14]: #리스트로 되어 있는 x와 y값을 넘파이 배열로 바꾸어 줍니다.  
#인덱스를 주어 하나씩 불러와 계산이 가능해 지도록 하기 위함입니다.  
x1_data = np.array(x1)  
x2_data = np.array(x2)  
y_data = np.array(y)  
  
# 기울기 a와 절편 b의 값을 초기화 합니다.  
a1 = 0  
a2 = 0  
b = 0  
  
#학습률을 정합니다.  
lr = 0.05  
  
#몇 번 반복될지를 설정합니다. (0부터 세므로 원하는 반복 횟수에 +1을 해 주어야 합니다.)  
epochs = 2001
```

[실습] 과외수업이 추가된 성적예측

[15]: #경사 하강법을 시작합니다.

```
for i in range(epochs): # epoch 수 만큼 반복
    y_pred = a1 * x1_data + a2 * x2_data + b #y를 구하는 식을 세웁니다
    error = y_data - y_pred #오차를 구하는 식입니다.
    a1_diff = -(1/len(x1_data)) * sum(x1_data * (error)) # 오차함수를 a1로 미분한 값입니다.
    a2_diff = -(1/len(x2_data)) * sum(x2_data * (error)) # 오차함수를 a2로 미분한 값입니다.
    b_new = -(1/len(x1_data)) * sum(y_data - y_pred) # 오차함수를 b로 미분한 값입니다.
    a1 = a1 - lr * a1_diff # 학습률을 곱해 기존의 a1값을 업데이트합니다.
    a2 = a2 - lr * a2_diff # 학습률을 곱해 기존의 a2값을 업데이트합니다.
    b = b - lr * b_new # 학습률을 곱해 기존의 b값을 업데이트합니다.
    if i % 100 == 0: # 100번 반복될 때마다 현재의 a1, a2, b값을 출력합니다.
        print("epoch=% .f, 기울기1=% .04f, 기울기2=% .04f, 절편=% .04f" % (i, a1, a2, b))
```

epoch=0, 기울기1=23.2000, 기울기2=10.5625, 절편=4.5250
epoch=100, 기울기1=6.4348, 기울기2=3.9893, 절편=43.9757
epoch=200, 기울기1=3.7255, 기울기2=3.0541, 절편=62.5766
epoch=300, 기울기1=2.5037, 기울기2=2.6323, 절편=70.9656
epoch=400, 기울기1=1.9527, 기울기2=2.4420, 절편=74.7491
epoch=500, 기울기1=1.7042, 기울기2=2.3562, 절편=76.4554
epoch=600, 기울기1=1.5921, 기울기2=2.3175, 절편=77.2250
epoch=700, 기울기1=1.5415, 기울기2=2.3001, 절편=77.5720
...
...

epoch=700, 기울기1=1.5415, 기울기2=2.3001, 절편=77.5720
epoch=800, 기울기1=1.5187, 기울기2=2.2922, 절편=77.7286
epoch=900, 기울기1=1.5084, 기울기2=2.2886, 절편=77.7992
epoch=1000, 기울기1=1.5038, 기울기2=2.2870, 절편=77.8310
epoch=1100, 기울기1=1.5017, 기울기2=2.2863, 절편=77.8453
epoch=1200, 기울기1=1.5008, 기울기2=2.2860, 절편=77.8518
epoch=1300, 기울기1=1.5003, 기울기2=2.2858, 절편=77.8547
epoch=1400, 기울기1=1.5002, 기울기2=2.2858, 절편=77.8561
epoch=1500, 기울기1=1.5001, 기울기2=2.2857, 절편=77.8567
epoch=1600, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8569
epoch=1700, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8570
epoch=1800, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8571
epoch=1900, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8571
epoch=2000, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8571

[TensorFlow] 선형회귀 실습

[1]: # 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers # 케라스의 옵티마이저를 임포트
from keras.callbacks import EarlyStopping
```

Using TensorFlow backend.

[2]: # 필요한 라이브러리를 불러옵니다.

```
import numpy as np
import tensorflow as tf
```

[3]: # 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.

```
np.random.seed(3)
tf.random.set_seed(3)
```

[4]: # 준비된 수술 환자 데이터를 불러들입니다.

```
Data_set = np.loadtxt("./dataset/class.csv", delimiter=",")
Data_set
```

[4]: array([[2., 81.],
 [4., 93.],
 [6., 91.],
 [8., 97.]])

[TensorFlow] 선형회귀 실습

```
[5]: X = [i[0] for i in Data_set]
      Y = [i[1] for i in Data_set]
      print(X)
      print(Y)

      [2.0, 4.0, 6.0, 8.0]
      [81.0, 93.0, 91.0, 97.0]

[6]: model = Sequential()
      model.add(Dense(1, input_dim=1, activation='linear'))

[7]: # 학습을 실행합니다.
      sgd = optimizers.SGD(lr=0.01)
      model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['mse'])
      early_stopping = EarlyStopping()
      model.fit(X, Y, epochs=800, batch_size=1, shuffle=False, callbacks=[early_stopping])

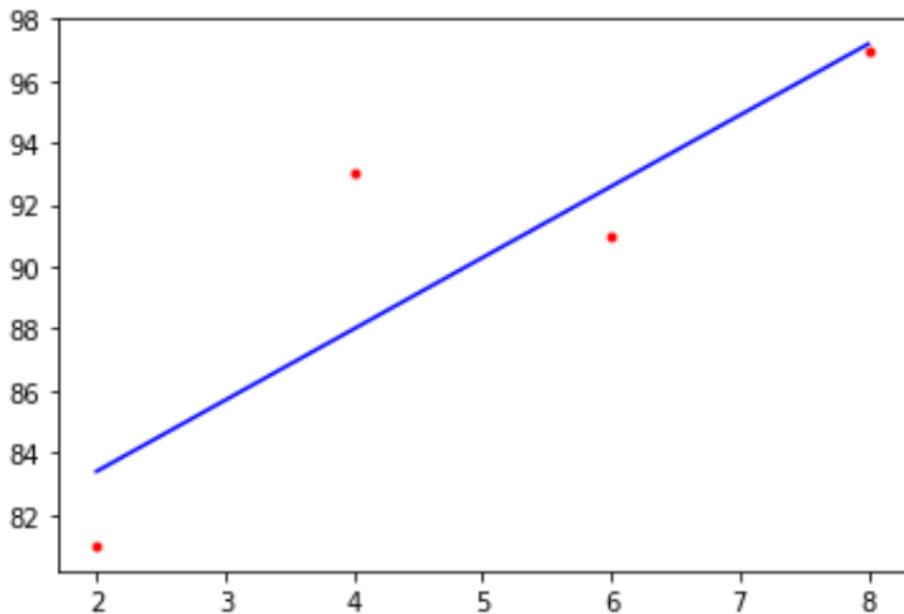
Train on 4 samples
Epoch 1/800
4/4 [=====] - 0s 45ms/sample - loss: 3810.1227 - mse: 3810.1226
Epoch 2/800
4/4 [=====] - 0s 2ms/sample - loss: 1253.6365 - mse: 1253.6365
Epoch 3/800
4/4 [=====] - 0s 3ms/sample - loss: 1269.7119 - mse: 1269.7120
```

[TensorFlow] 선형회귀 실습

```
Epoch 799/800  
4/4 [=====] - 0s 3ms/sample - loss: 12.3318 - mse: 12.3318  
Epoch 800/800  
4/4 [=====] - 0s 3ms/sample - loss: 12.3318 - mse: 12.3318
```

```
[7]: <tensorflow.python.keras.callbacks.History at 0x2b12ff77bc8>
```

```
[8]: %matplotlib inline  
import matplotlib.pyplot as plt  
plt.plot(X, model.predict(X), 'blue', X, Y, 'r.')  
plt.show()
```



[TensorFlow] 선형회귀 실습

```
[9]: #Gradient decent로 계산한 w와 b 출력  
model.layers[0].get_weights()
```

```
[9]: [array([[2.3000648]]], dtype=float32), array([78.79379], dtype=float32)]
```

```
[10]: w = model.layers[0].get_weights()[0][0]  
w
```

```
[10]: array([2.3000648], dtype=float32)
```

```
[11]: b = model.layers[0].get_weights()[1][0]  
b
```

```
[11]: 78.79379
```

```
[12]: #x가 2시간을 때와 4시간을 때 성적 예측값 구하기  
print(f'x = 2 일 때 {w*2 + b}')  
print(f'x = 4 일 때 {w*4 + b}')
```

```
x = 2 일 때 [83.39392]  
x = 4 일 때 [87.99405]
```

```
[13]: pred = model.predict([4])  
pred
```

```
[13]: array([[87.99405]], dtype=float32)
```

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	83.6	88.2	92.8	97.4
오차	-2.6	4.8	-1.8	-0.4

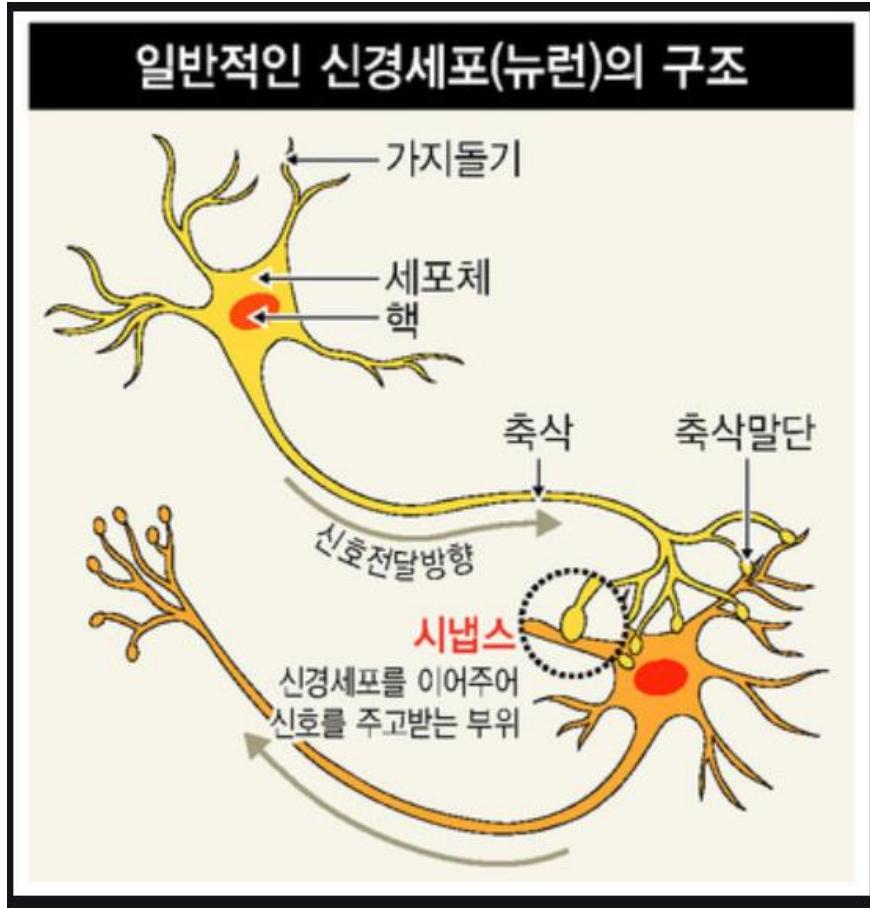
인공 신경망

신경세포

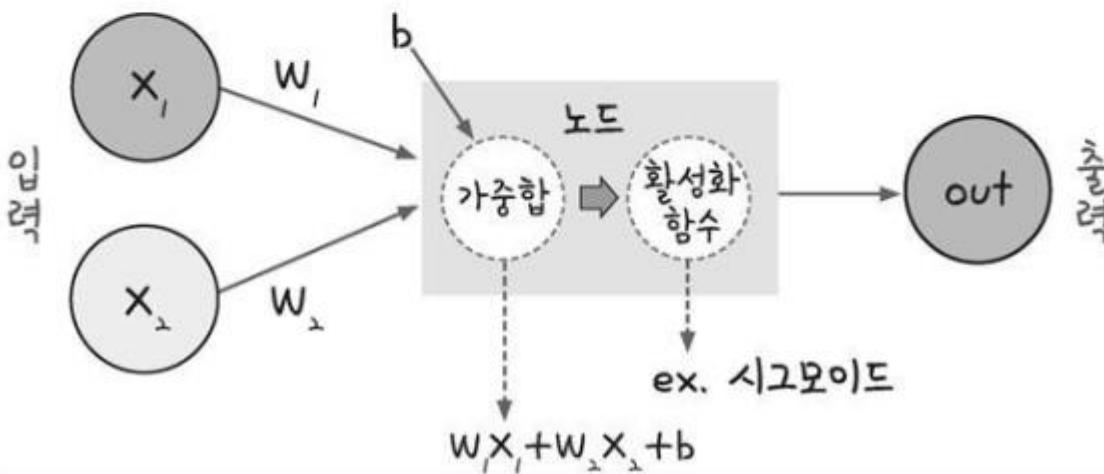
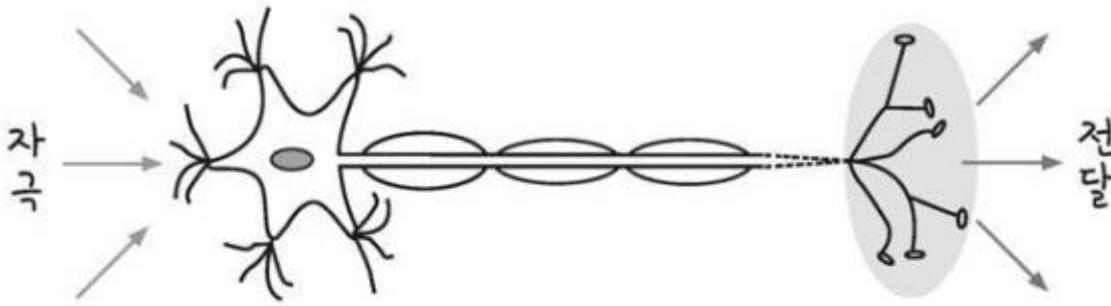
- ▶ 여러 개의 수상돌기에서 자극이 합해져서

일반적인 신경세포(뉴런)의 구조

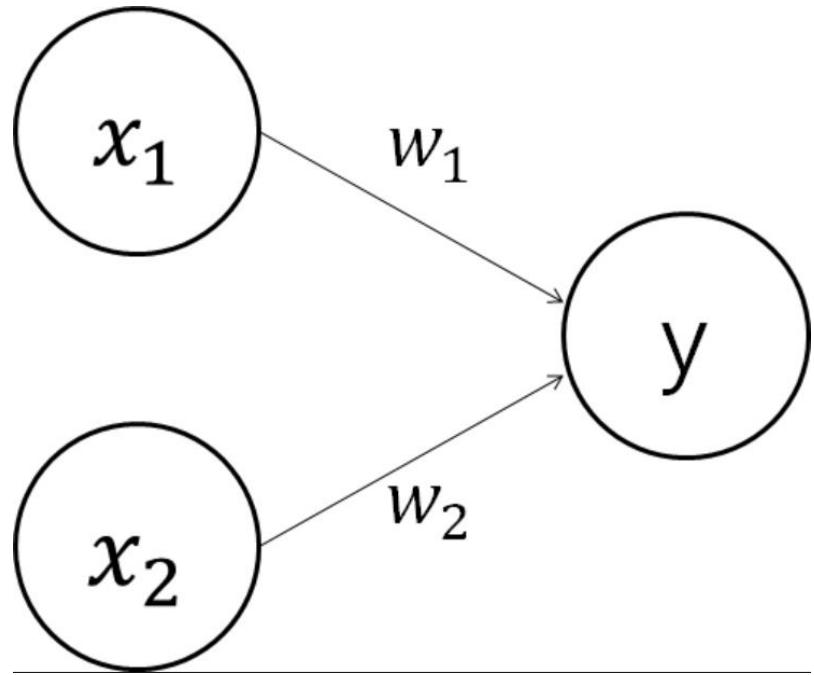
- ▶ 그 값이 특정 값 이상일 경우
- ▶ 축삭돌기로 자극을 전달한다



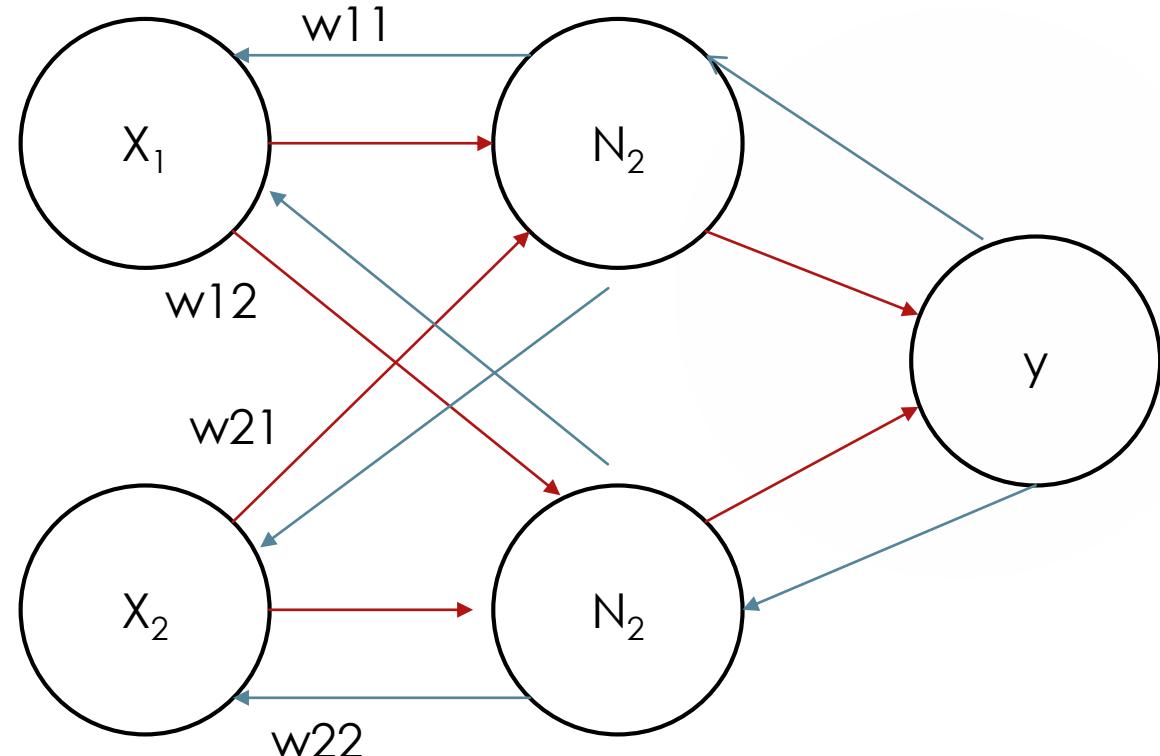
인공 신경망



퍼셉트론

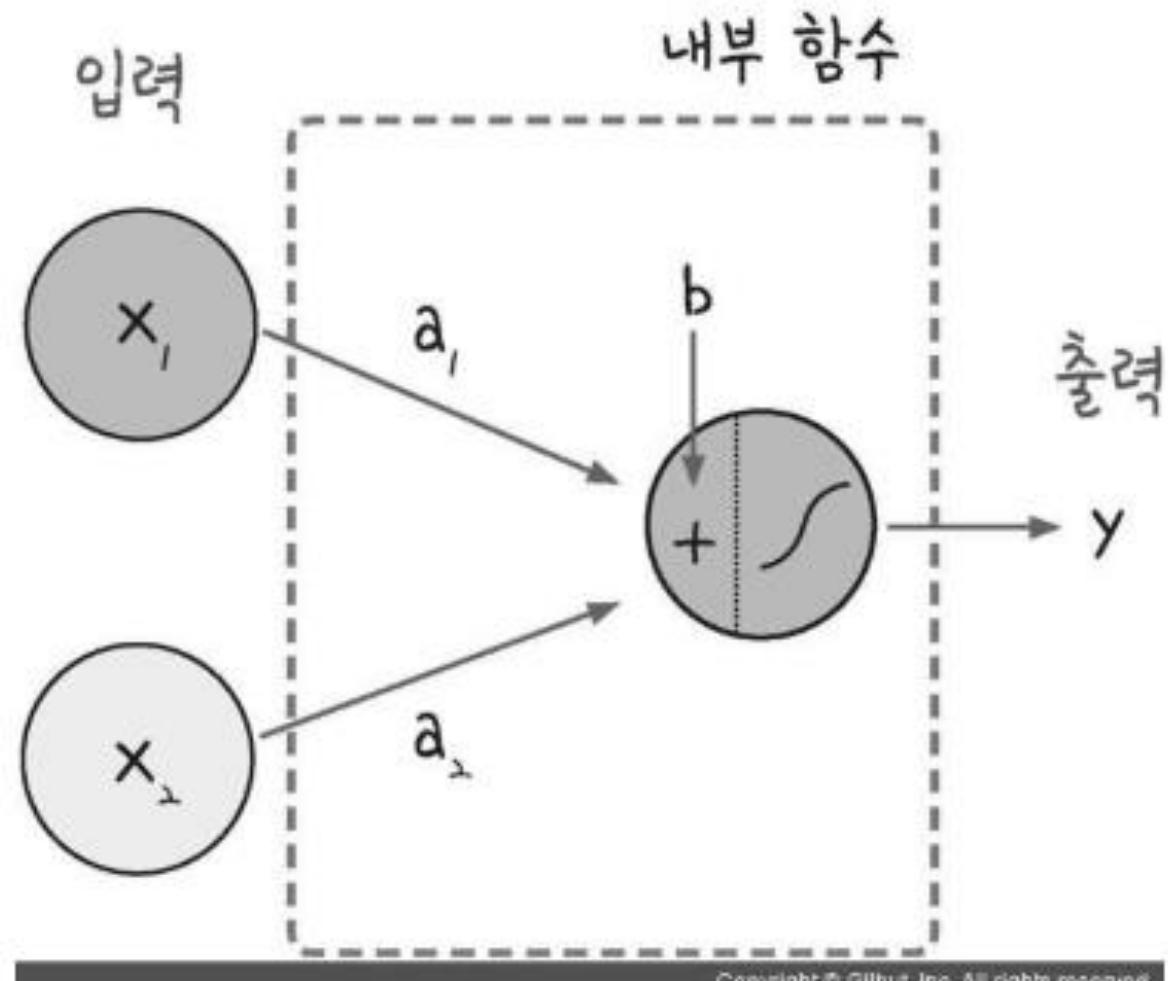


단일(층) 퍼셉트론



다층 퍼셉트론

퍼셉트론 방식의 표현



퍼셉트론(Perceptron)

- ▶ 인공 신경망의 다른 이름
- ▶ 다수의 신호를 입력으로 받아 하나의 신호를 출력한다
- ▶ 퍼셉트론은 신호 1과 0을 가질 수 있다
- ▶ 주로 분류에 사용
- ▶ 활성화함수는 Sigmoid, Relu 등이 있음

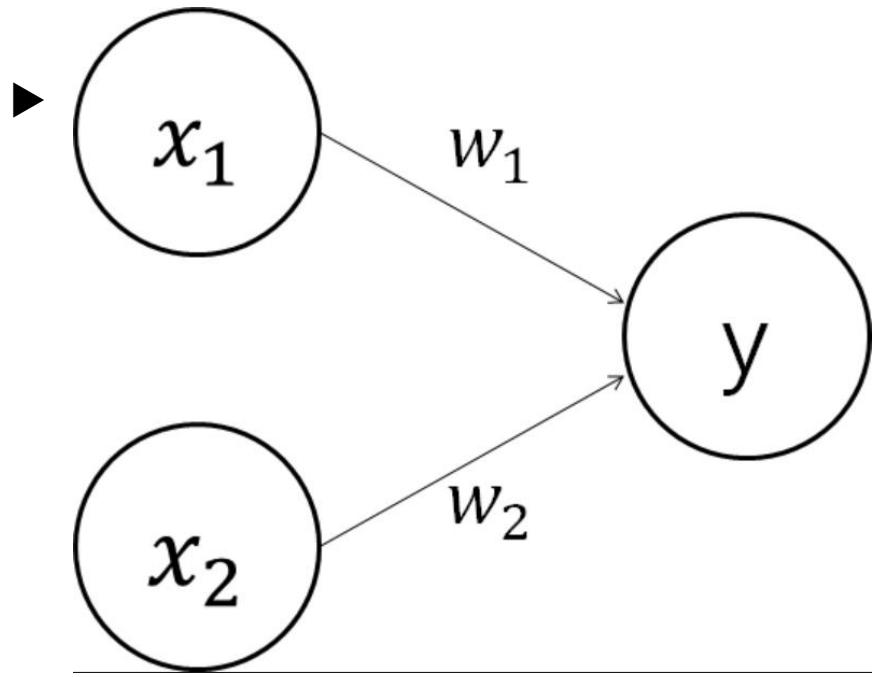
가중치, 가중합, 바이어스, 활성화 함수

- ▶ $y = ax + b$ (a 는 기울기, b 는 y 절편)
- ▶ $y = wx + b$ (w 는 가중치, b 는 바이어스)
 - ❖ 용어
- ▶ **가중합**(weighted sum) : 입력값(x)과 가중치(w)를 모두 곱을 모두 더한 결과에 바이어스를 더한 값
- ▶ **활성화 함수**(activation function) : 가중합의 결과를 놓고 0과 1을 출력해서 다음으로 보내는데 이 0과 1을 판단하는 함수를 활성화 함수



인공 신경망을
이용한 And 연산

수식으로 나타낸 퍼셉트론



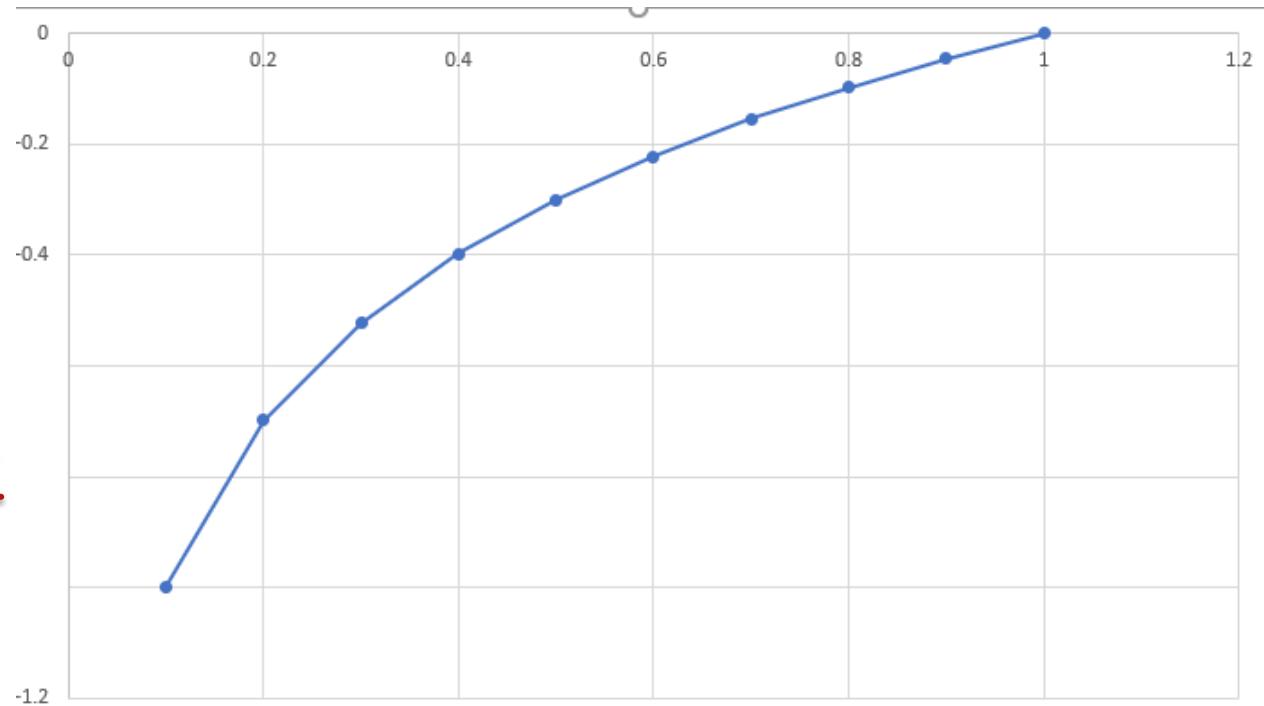
$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

입력이 두개인 퍼셉트론

인공 신경망의 Cost(Loss) 함수

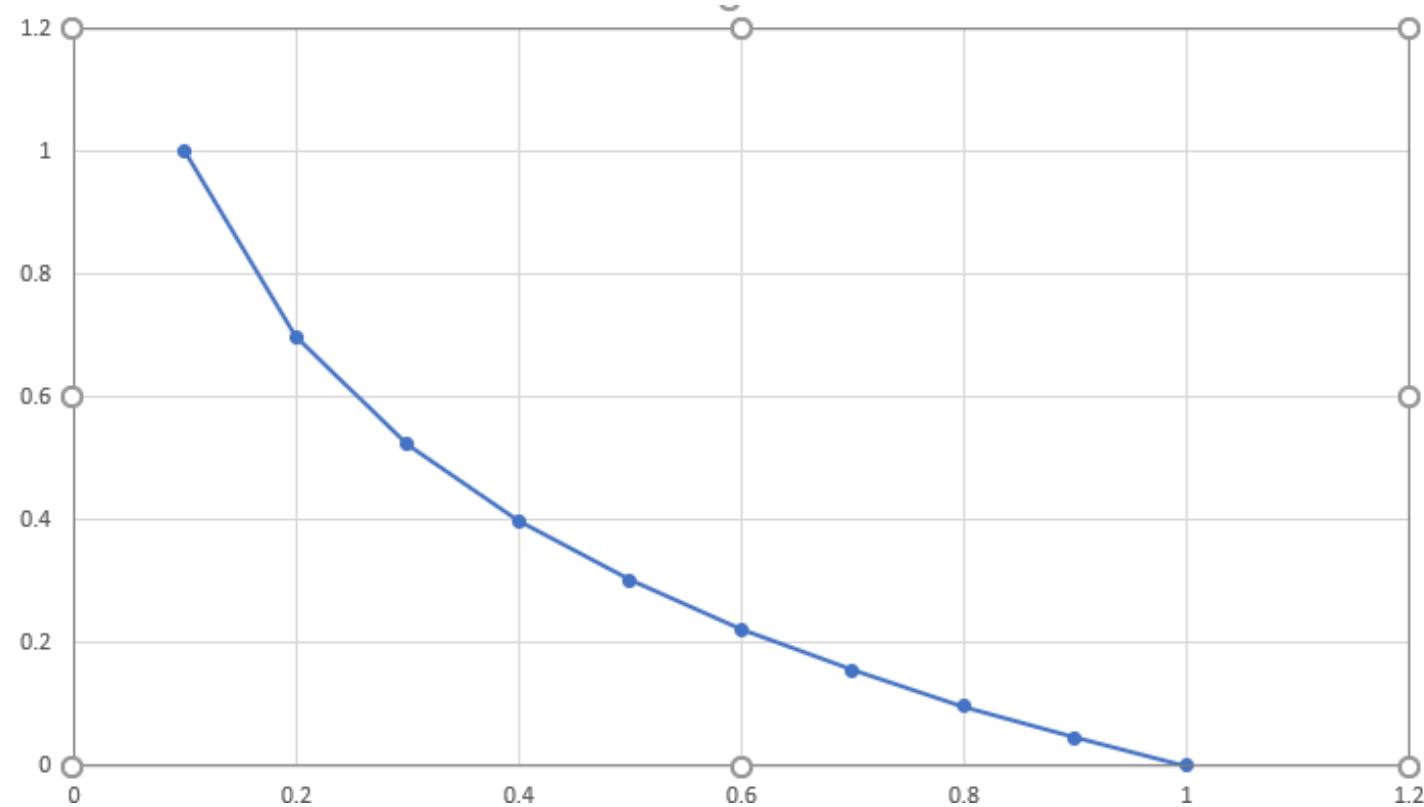
- ▶ 현재 모델이 이상적 모델과 비교했을 때 차이가 얼마나 나는지 정하기 위한 함수
- ▶ 결과값(output)을 이용하여 정의 -> 결과값을 줄이는 방향
- ▶ log를 이용해서 Cost 함수를 구현
- ▶ $\log(0) = \text{음의무한대}$
- ▶ $\log(1) = 0$

- MSE(mean square error) : $\text{sum}(y - f(x))^2$
- 분류 오차(classification error) : $-\text{ylog}(H(x)) - (1-y)\log(1-H(x))$
- 교차 엔트로피(cross entropy) : $-\text{sum}(y \log y)$



인공 신경망의 Cost(Loss) 함수

- ▶ 0~1 사이 log에 -를 곱한 Cost 함수를 이용해서 Gradient Decent를 구현



0~1 사이의 로그 값에
-를 곱한 값

인공 신경망의 Cost(Loss) 함수

- ▶ $H(x)$ 는 예측한 값
- ▶ y 는 실제 값
- ▶ 실제 값이 y 가 1일 때 예측값 $H(X) = 1$ 일 때는 왼쪽 그래프에서 오차는 0이 된다.
(cost=0)
- ▶ 예측값 $H(X) = 0$ 일 때는 왼쪽 그래프에서 오차 무한대(∞)가 된다.
(cost=무한대)
- ▶ 실제값 y 가 0일 때 예측값 $H(X) = 0$ 일 때는 오른쪽 그래프에서 오차는 0이 된다.
(cost=0)
- ▶ 예측값 $H(X) = 1$ 일 때는 오른쪽 그래프에서 오차는 무한대가 된다.
(cost=무한대)

인공신경망의 Cost 함수

- ▶ y 가 0일 때와 1일 때는 if 문을 사용하지 않고 하나의 수식으로 연결하면 다음과 같다

Cost function

$$cost(W) = \frac{1}{m} \sum c(H(x), y)$$

$$C(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

$$C(H(x), y) = -y \log(H(x)) - (1 - y) \log(1 - H(x))$$

시그모이드 함수

- ▶ 선형 회귀 결과를 0과 1로 변환해서 리턴하는 함수
- ▶ 선형회귀 식 $ax+b$ 를 0과 1로 리턴하는 시그모이드 함수 수식

$$y = \frac{1}{1 + e^{(ax+b)}}$$

- ▶ 여기서 e는 자연 상수라고 불리는 무리수로 값은 2.71828...
- ▶ 파이 (π)처럼 수학에서 중요하게 사용되는 상수로 고정된 값이므로 우리가 따로 구해야 하는 값은 아님
- ▶ 우리가 구해야 하는 값은 결국 $Wx+b$ 에서 W 와 b

Sigmoid – 자연상수 조회(e)

```
[1]: from math import e
```

```
[2]: e
```

```
[2]: 2.718281828459045
```

```
[3]: e**-1
```

```
[3]: 0.36787944117144233
```

```
[4]: e**0
```

```
[4]: 1.0
```

Sigmoid

```
[7]: 1/(1 + e**-(-5))
```

```
[7]: 0.006692850924284857
```

```
[8]: 1/(1 + e**-(-2.5))
```

```
[8]: 0.07585818002124356
```

```
[9]: 1/(1 + e**-(0))
```

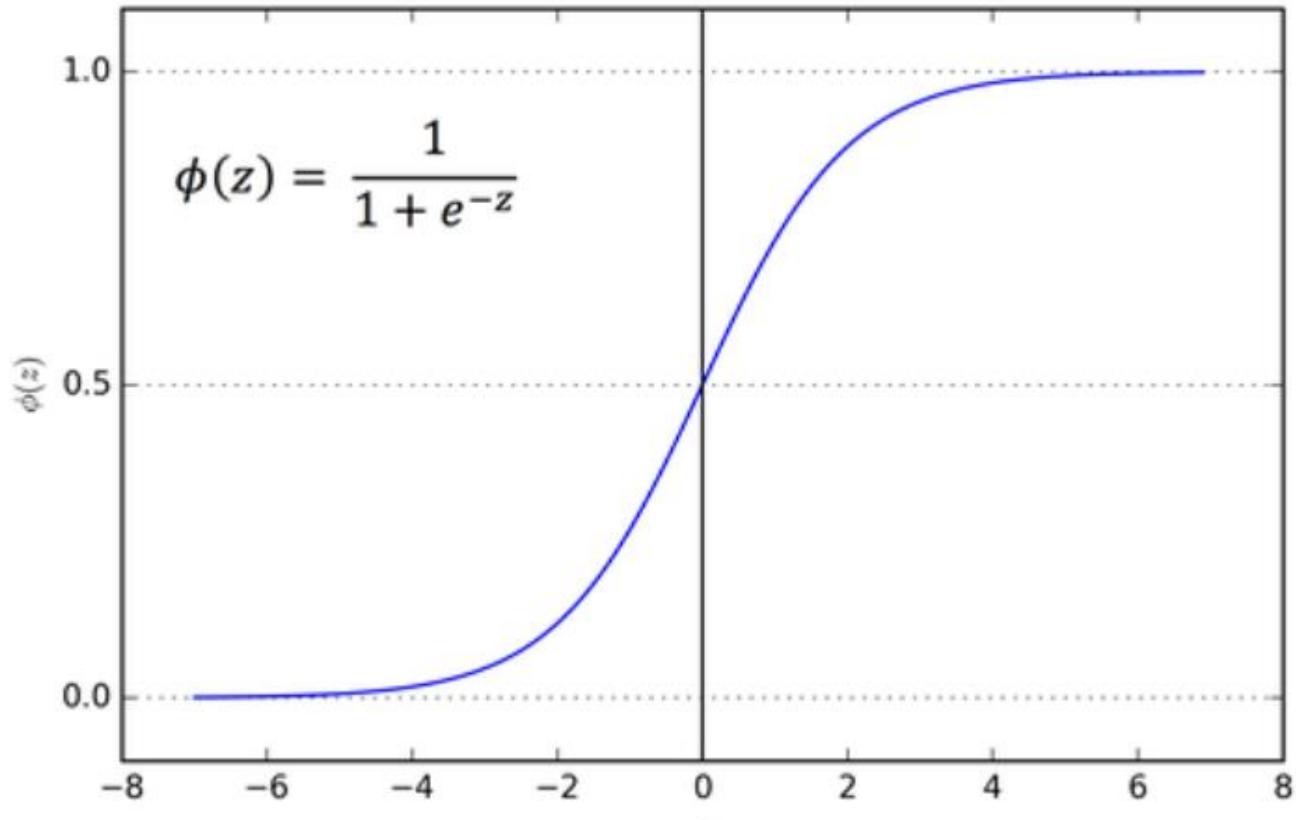
```
[9]: 0.5
```

```
[10]: 1/(1 + e**-(-2.5))
```

```
[10]: 0.07585818002124356
```

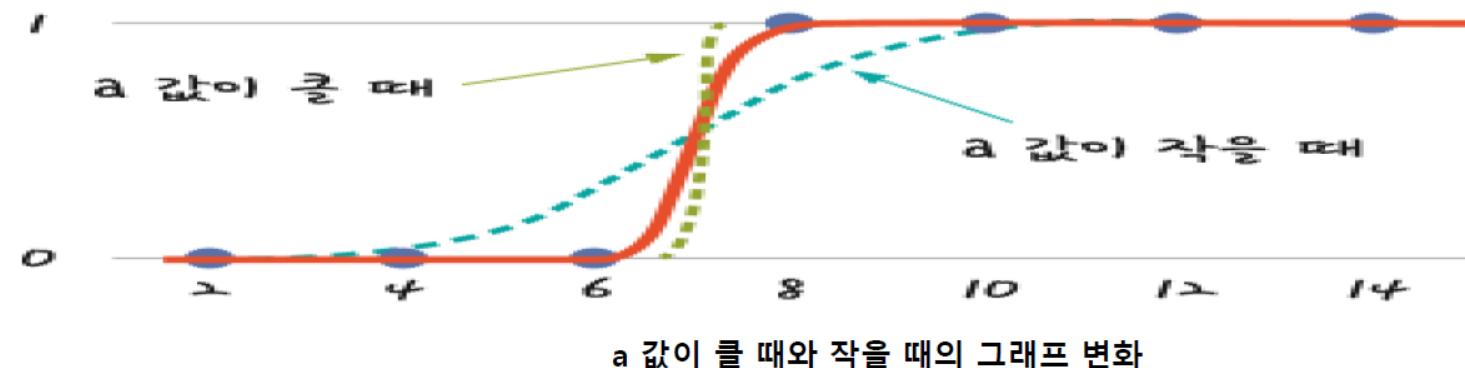
```
[11]: 1/(1 + e**-(5))
```

```
[11]: 0.9933071490757153
```



Sigmoid 함수

- ▶ 선형회귀에서 우리가 구해야 하는 것이 w 와 b 였듯 여기서도 마찬가지
- ▶ 앞서 구한 직선의 방정식과는 다르게 여기에서 w 와 b 는 어떤 의미를 지니고 있을까?
- ▶ 먼저 w 는 그래프의 경사도를 결정
- ▶ w 값이 커지면 경사가 커지고 a 값이 작아지면 경사가 작아짐



인공 신경망을 이용한 AND 연산

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

[실습] AND 연산

```
[1]: import numpy as np
```

```
[3]: # 가중치와 바이어스  
w11 = np.array([-2, -2])  
w12 = np.array([2, 2])  
w2 = np.array([1, 1])  
b1 = 3  
b2 = -1  
b3 = -1
```

```
[4]: # 퍼셉트론  
def MLP(x, w, b):  
    y = np.sum(w * x) + b  
    if y <= 0:  
        return 0  
    else:  
        return 1
```

```
[7]: # AND 계기  
def AND(x1,x2):  
    return MLP(np.array([x1, x2]), w2, b3)
```

[실습] AND 연산

```
[17]: # x1, x2 값을 번갈아 대입해 가며 최종값 출력  
print("### AND ###")  
for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:  
    y = AND(x[0], x[1])  
    print("입력 값: " + str(x) + " 출력 값: " + str(y))
```

```
### AND ###  
입력 값: (0, 0) 출력 값: 0  
입력 값: (1, 0) 출력 값: 0  
입력 값: (0, 1) 출력 값: 0  
입력 값: (1, 1) 출력 값: 1
```

퍼셉트론 문제해결



퍼셉트론 과제

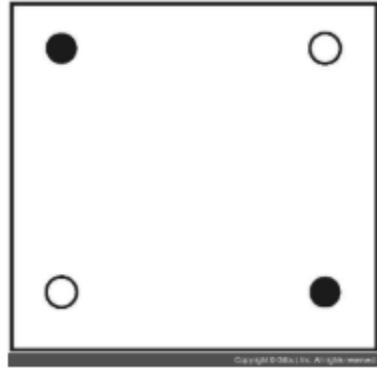


그림1. 검은점 두개와 흰점 두개

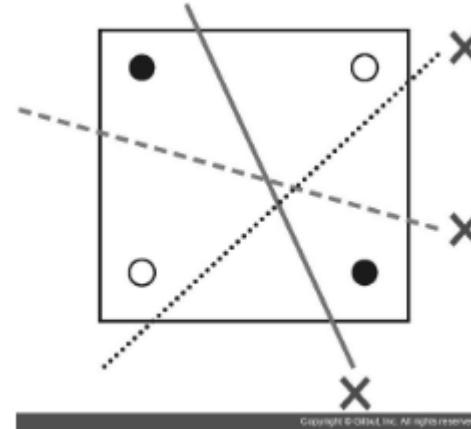


그림2. 선으로는 같은 색끼리 나눌수 없다

AND, OR, XOR 진리표

AND 진리표

x_1	x_2	결과값
0	0	0
0	1	0
1	0	0
1	1	1

OR 진리표

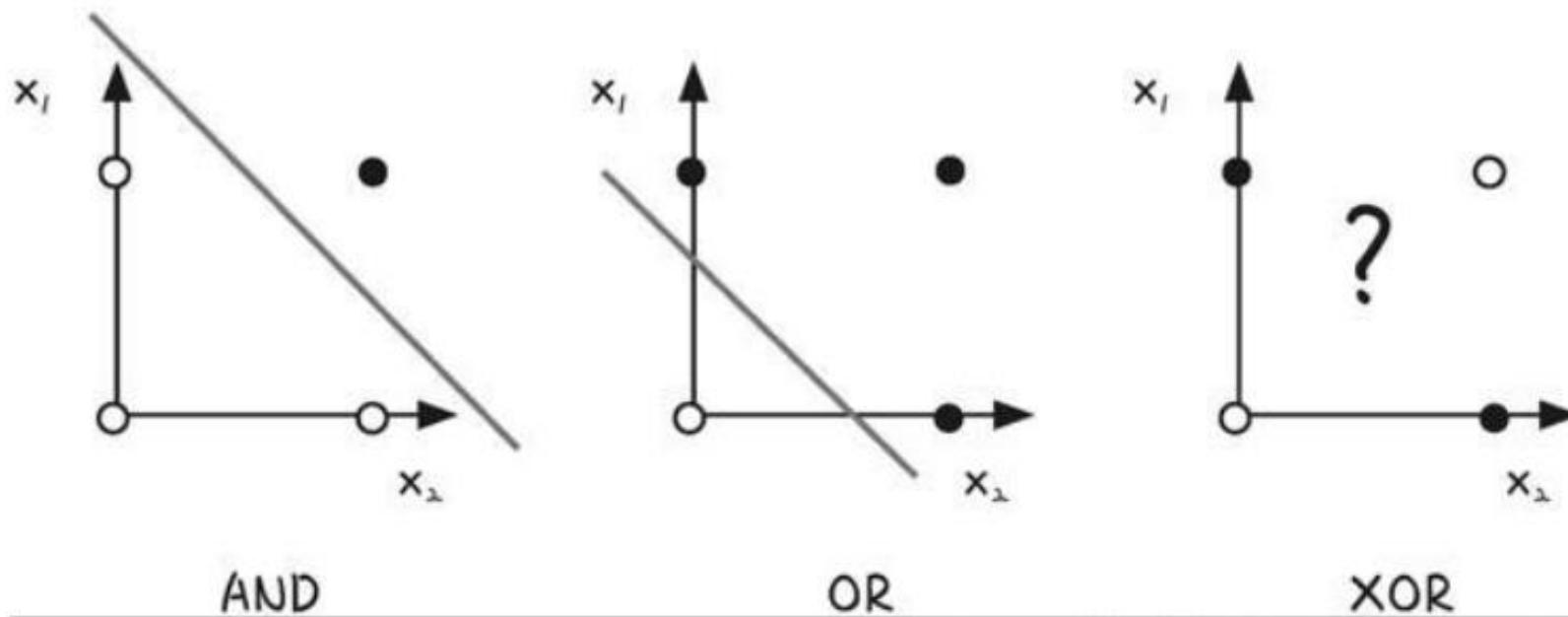
x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	1

XOR 진리표

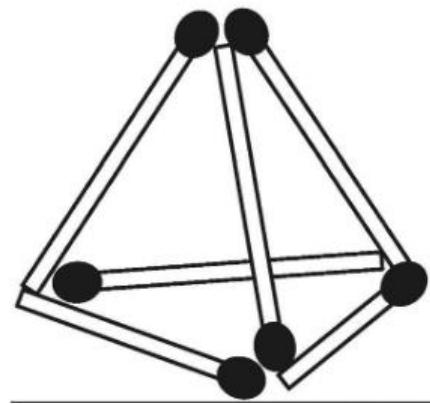
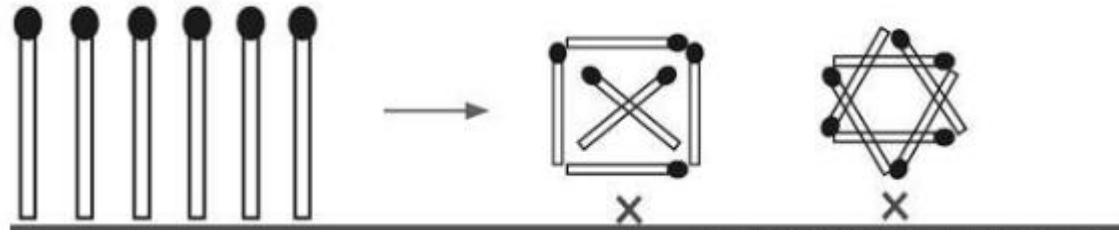
x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	0

2차원 평면에서의 XOR 불가능 문제

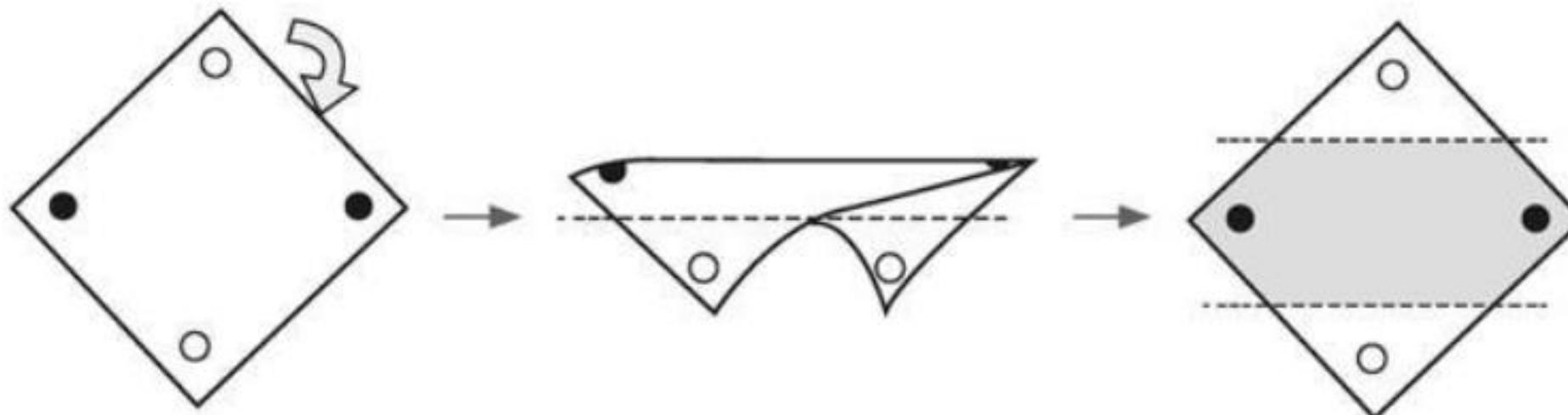
- ▶ 마빈민스키(Marvin Minsky) 1969 발표논문 퍼셉트론즈(Perceptrons)



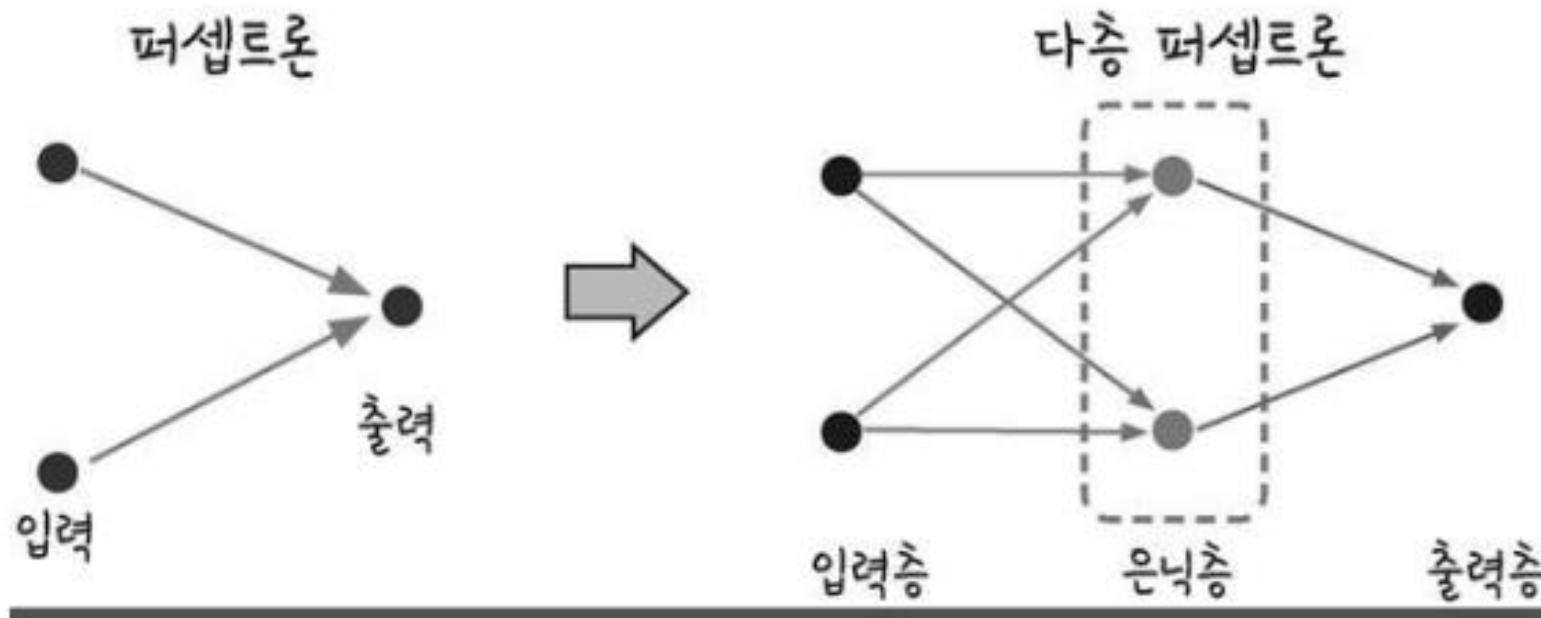
다층 퍼셉트론



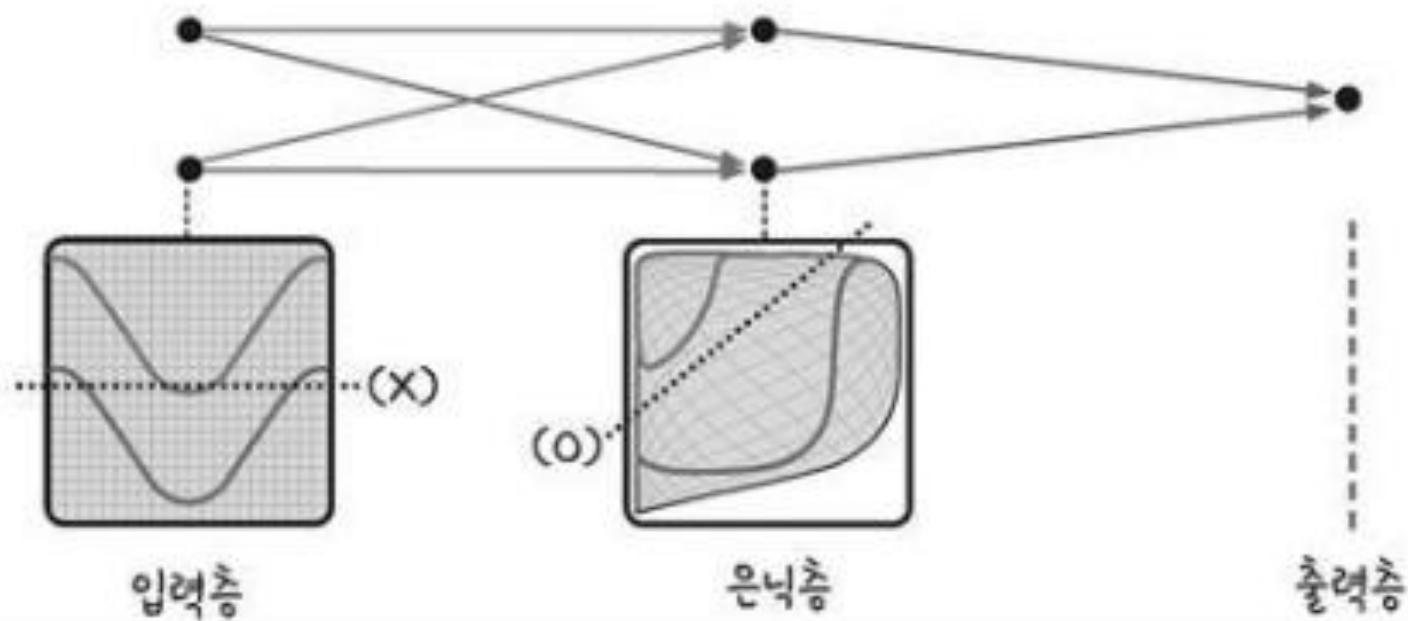
다층 퍼셉트론 개념도



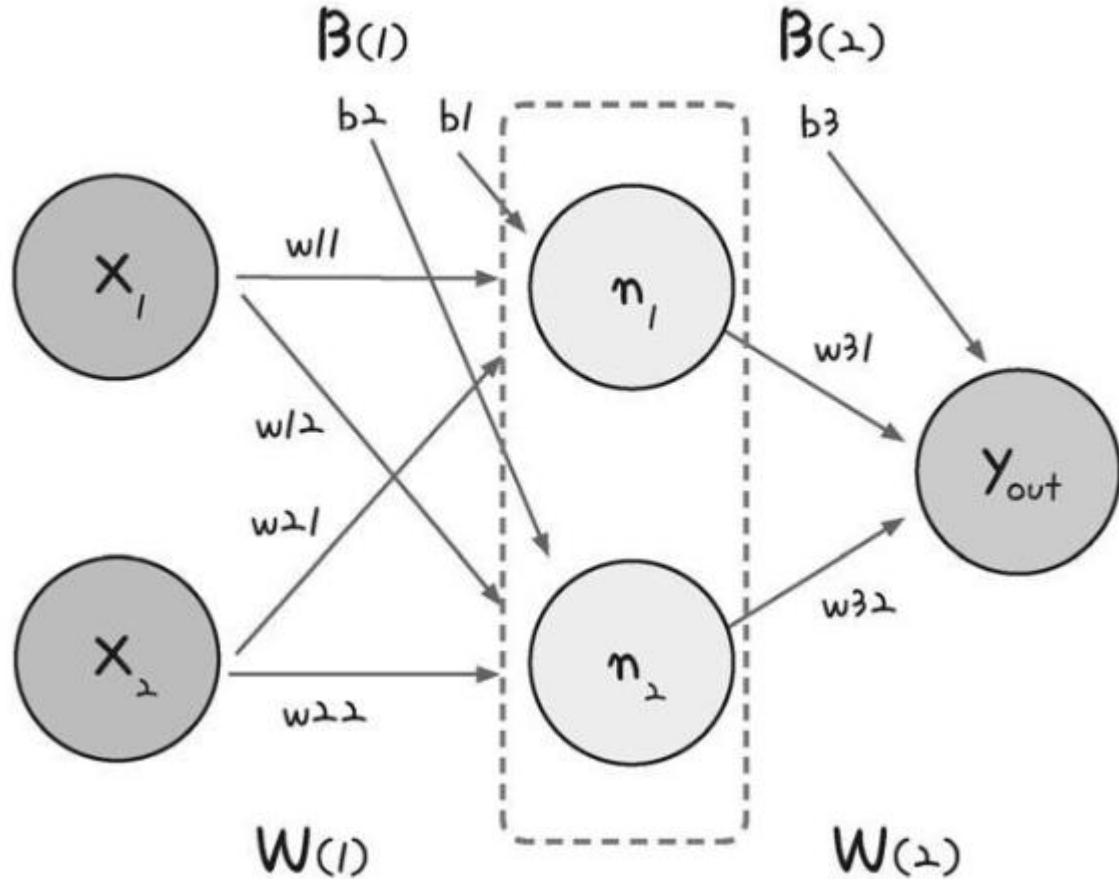
다층 퍼셉트론 개념도



다층 퍼셉트론 개념도

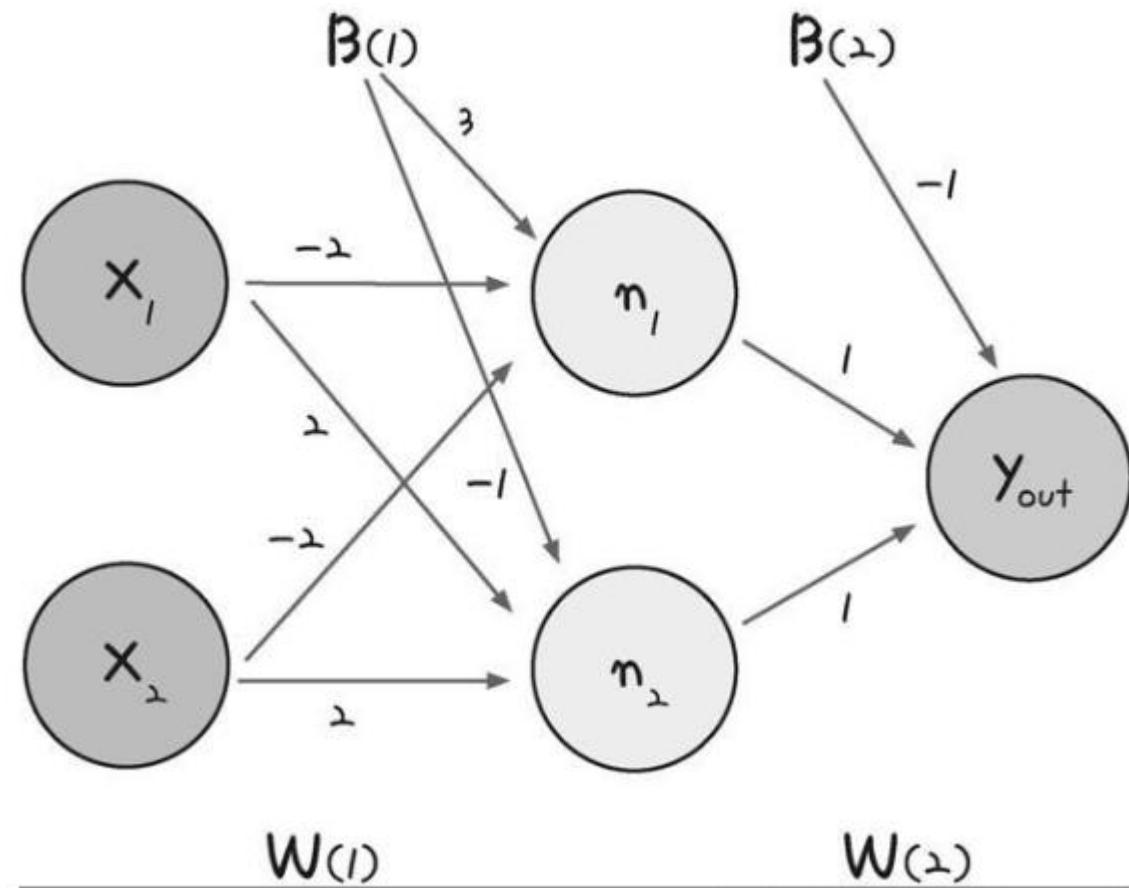


퍼셉트론 설계



XOR 문제의 해결

$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$
$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$



XOR 문제의 해결

```
# NAND 게이트
def NAND(x1, x2):
    return MLP(np.array([x1, x2]), w11, b1)

def MLP(x, w, b):
    y = np.sum(w * x) + b
    if y <= 0:
        return 0
    else:
        return 1

# OR 게이트
def OR(x1, x2):
    return MLP(np.array([x1, x2]), w12, b2)

# AND 게이트
def AND(x1, x2):
    return MLP(np.array([x1, x2]), w2, b3)

# XOR 게이트
def XOR(x1, x2):
    return AND(NAND(x1, x2), OR(x1, x2))
```

[실습] XOR

```
[1]: import numpy as np
```

```
[3]: # 가중치와 바이어스  
w11 = np.array([-2, -2])  
w12 = np.array([2, 2])  
w2 = np.array([1, 1])  
b1 = 3  
b2 = -1  
b3 = -1
```

```
[4]: # 퍼셉트론  
def MLP(x, w, b):  
    y = np.sum(w * x) + b  
    if y <= 0:  
        return 0  
    else:  
        return 1
```

[실습] XOR

[5]: # NAND 계산

```
def NAND(x1,x2):
    return MLP(np.array([x1, x2]), w11, b1)
```

[6]: # OR 계산

```
def OR(x1,x2):
    return MLP(np.array([x1, x2]), w12, b2)
```

[8]: # XOR 계산

```
def XOR(x1,x2):
    return AND(NAND(x1, x2),OR(x1,x2))
```

[실습] XOR

```
print("### XOR ###")
for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:
    y = XOR(x[0], x[1])
    print("입력 값: " + str(x) + " 출력 값: " + str(y))
```

```
### XOR ###
입력 값: (0, 0) 출력 값: 0
입력 값: (1, 0) 출력 값: 1
입력 값: (0, 1) 출력 값: 1
입력 값: (1, 1) 출력 값: 0
```



