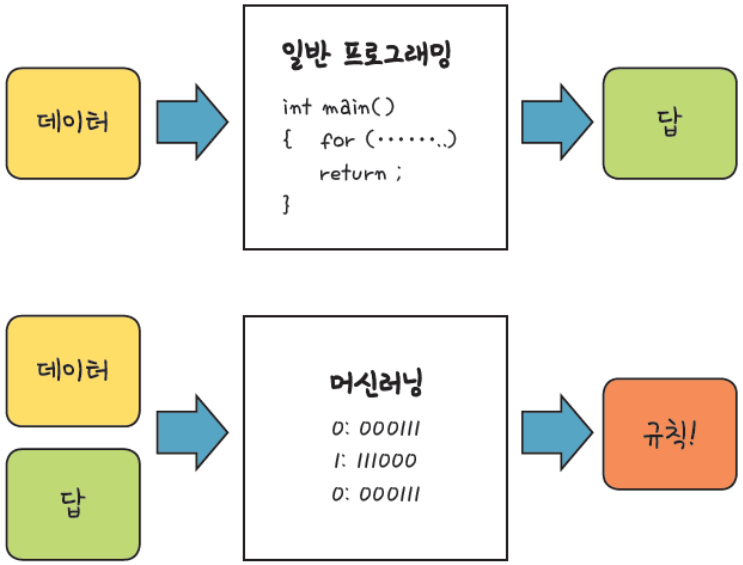
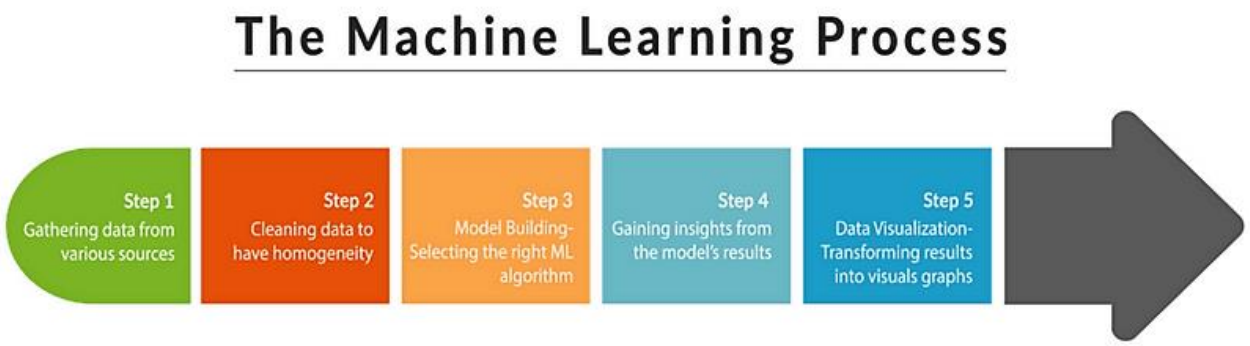
# 머신러닝&딥러닝을 위한 기초 수학
(모두의 딥러닝)

# 1장. 딥러닝의 이해

# 머신러닝

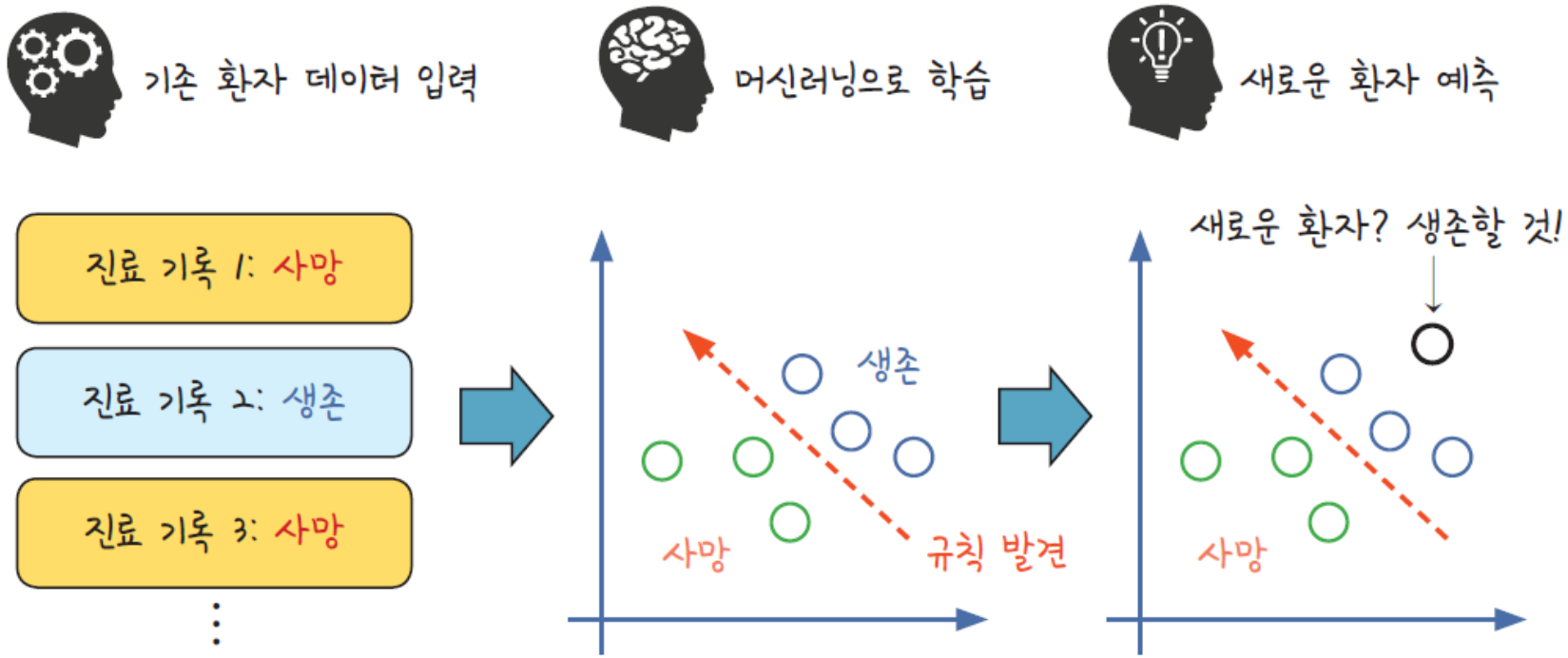
- 머신러닝 : 데이터를 이용해 미지의 일을 예측하기 위해 만들어진 기법



# 머신러닝

- 학습(training) : 데이터가 입력되고 패턴이 분석되는 과정




- 랜덤포레스트, 서포트 벡터 머신, 딥러닝 등 여러가지 머신러닝 기법들 존재

# 실습 : 폐암 수술 환자의 생존율 예측하기

속성(attribute), 특성(feature) X :
수술 환자 기록 17개 변수
(종양 유형, 폐활량, 호흡곤란 여부 등)

클래스 Y:
생존/사망

470행

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 293 | 1 | 3.8 | 2.8 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 62 | 0 |
| 1 | 2 | 2.88 | 2.16 | 1 | 0 | 0 | 0 | 1 | 1 | 14 | 0 | 0 | 0 | 1 | 0 | 60 | 0 |
| 8 | 2 | 3.19 | 2.5 | 1 | 0 | 0 | 0 | 1 | 0 | 11 | 0 | 0 | 1 | 1 | 0 | 66 | 1 |
| 14 | 2 | 3.98 | 3.06 | 2 | 0 | 0 | 0 | 1 | 1 | 14 | 0 | 0 | 0 | 1 | 0 | 80 | 1 |
| 17 | 2 | 2.21 | 1.88 | 0 | 0 | 1 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 56 | 0 |
| 18 | 2 | 2.96 | 1.67 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 61 | 0 |
| 35 | 2 | 2.76 | 2.2 | 1 | 0 | 0 | 0 | 1 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 76 | 0 |
| 42 | 2 | 3.24 | 2.52 | 1 | 0 | 0 | 0 | 1 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 63 | 1 |
| 65 | 2 | 3.15 | 2.76 | 1 | 0 | 1 | 0 | 1 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 59 | 0 |
| 111 | 2 | 4.48 | 4.2 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 55 | 0 |
| 121 | 2 | 3.84 | 2.56 | 1 | 0 | 0 | 0 | 1 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 59 | 0 |
| 123 | 2 | 2.8 | 2.12 | 1 | 0 | 0 | 1 | 1 | 0 | 13 | 0 | 0 | 0 | 1 | 0 | 80 | 0 |
| 130 | 2 | 5.6 | 4.64 | 1 | 0 | 0 | 0 | 1 | 0 | 11 | 0 | 0 | 0 | 1 | 0 | 45 | 0 |
| 132 | 2 | 2.12 | 1.72 | 1 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 1 | 0 | 74 | 0 |

- 1번째 항목부터 17번째 항목까지를 속성(attribute)이나 특성(feature) 이라고 함
- 정답에 해당하는 18번째 항목을 '클래스(class)'라고 함
- 딥러닝을 구동시키려면 '속성'만을 뽑아 데이터셋을 만들고, '클래스'를 담는 데이터셋을 또 따로 만들어 줘야 함

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

# 실습 : 폐암 수술 환자의 생존율 예측하기

In [4]:
```
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 필요한 라이브러리를 불러옵니다.
import numpy as np
import tensorflow as tf

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

# 준비된 수술 환자 데이터를 불러들입니다.
Data_set = np.loadtxt("../dataset/ThoracicSurgery.csv", delimiter=",")

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

# 딥러닝 구조를 결정합니다(모델을 설정하고 실행하는 부분입니다).
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

```
47/47 [==============================] - 0s 427us/step - loss: 0.4149 - accuracy: 0.8447
Epoch 98/100
47/47 [==============================] - 0s 423us/step - loss: 0.3857 - accuracy: 0.8596
Epoch 99/100
47/47 [==============================] - 0s 425us/step - loss: 0.4051 - accuracy: 0.8532
Epoch 100/100
47/47 [==============================] - 0s 438us/step - loss: 0.3832 - accuracy: 0.8468
```

Out[4]: <tensorflow.python.keras.callbacks.History at 0x7fa874158810>

# 실습 : 폐암 수술 환자의 생존율 예측하기

- loss는 예측이 실패할 확률, accuracy는 예측이 성공할 확률임
- 예측 성공률은 데이터를 분석해 데이터를 확장하거나, 딥러닝 구조를 적절하게 바꾸는 등의 노력으로 더 향상될 수 있음
- 그뿐만 아니라 학습에 사용되지 않은 데이터를 따로 모아 테스트를 해 보면서 이 예측 성공률이 정말로 가능한지를 확인하는 과정까지 거치게 됨
- 이러한 '최적화 과정'을 진행하려면 딥러닝의 구동 원리를 이해해야 함

# 파이썬과 라이브러리

- 파이썬
  - 초보자부터 전문가까지 사용자 폭이 넓은 프로그래밍 언어임
  - 다양한 플랫폼에서 사용할 수 있음
  - 특히 라이브러리가 풍부하여 연구 기관 및 산업계에서 두루 사용되고 있음

- 라이브러리
  - 특정 기능을 담은 작은 프로그램(모듈, module)을 말함
  - 라이브러리를 불러올 때 사용하는 명령어가 import임

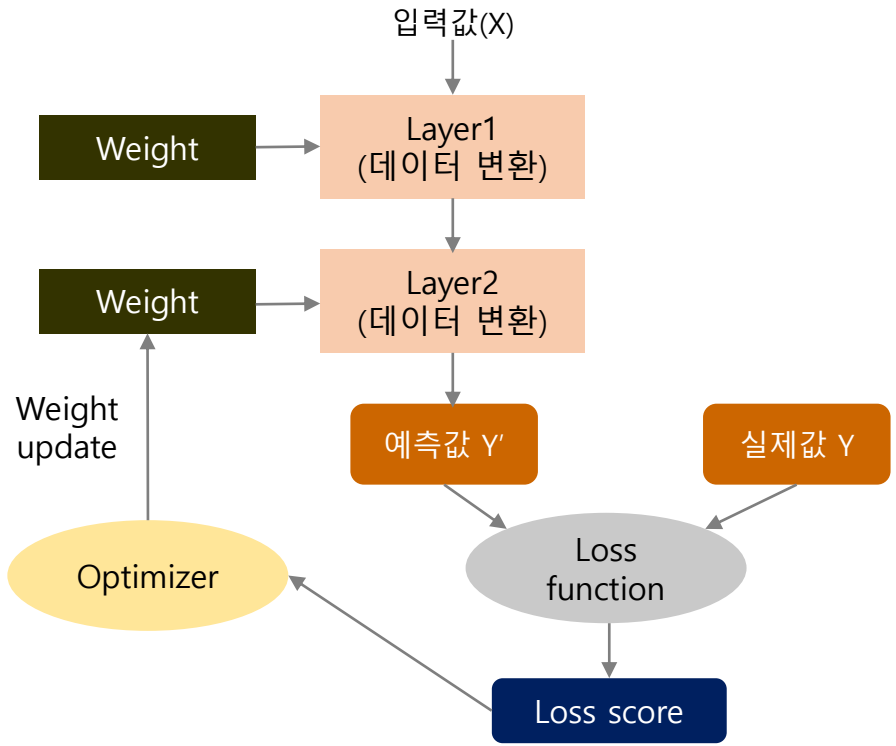
```
import numpy as np
```

→넘파이(numpy)라는 라이브러리를 np라는 이름으로 불러오라는 뜻

  - 넘파이는 수치 계산을 위해 만들어진 라이브러리로 데이터 분석에 많이 사용됨

# 딥러닝 학습 프로세스

입력값이 네트워크 층을 거치면 예측값이 나오고, 이를 실제값과 비교해서 Loss score를 계산한 후에 Optimizer를 통해 Weight를 업데이트 한다.

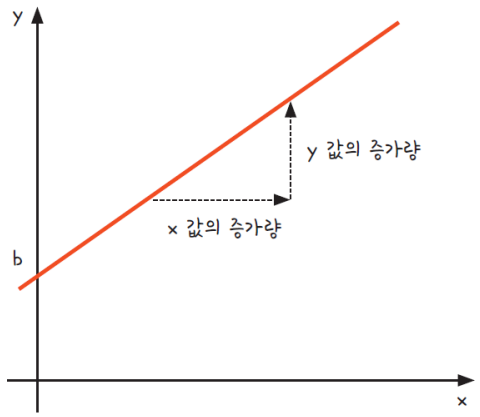


# 2장. 딥러닝 기초 수학

# 목차

# 일차 함수, 기울기와 y 절편

- 함수 : 두 집합 사이의 관계를 설명하는 수학 개념
- 변수 x와 y가 있을 때, x가 변하면 이에 따라 y는 어떤 규칙으로 변하는지를 나타냄
- 보통 함수를 나타낼 때는 function의 f와 변수 x를 사용해 $y=f(x)$ 라고 표시함
- 일차 함수 : x가 y에 관한 일차식으로 표현된 경우를 말함 $y = ax + b(a \neq 0)$
- x가 일차인 형태이며 x가 일차로 남으려면 a는 0이 아니어야 함
- 일차 함수식 $y=ax+b$ 에서 a는 기울기, b는 절편이라고 함

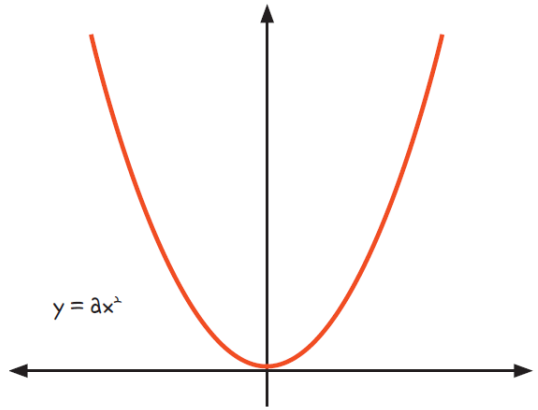



- 딥러닝의 수학 원리를 배울 때 초반부터 이 식이 등장함
- x가 주어지고 원하는 y값이 있을 때 적절한 a 와 b 를 찾는 것,

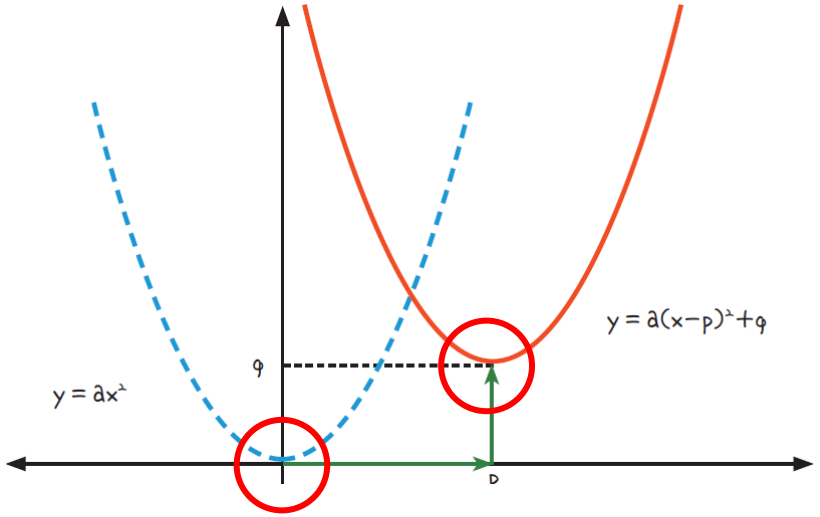
  이것이 바로 딥러닝을 설명하는 가장 간단한 표현임

# 이차 함수와 최솟값

- 이차 함수란 y가 x에 관한 이차식으로 표현되는 경우를 말함
- a > 0이면 아래로 볼록한 그래프가 됨




- 포물선의 맨 아래에 위치한 지점이 최솟값이 되는데, 딥러닝을 실행할 때 이 최솟값을 찾아내는 과정이 매우 중요함



# 미분, 순간 변화율과 기울기

- 딥러닝을 이해하는 데 가장 중요한 수학 원리는 미분이라고 할 수 있음
- 너무 미세해서 실제로 움직이는 게 아니라 방향만 드러내는 정도의 순간적인 변화만 있을 것임
- 이 순간의 변화를 놓고 순간 변화율이라는 이름을 붙임
- 순간 변화율은 어느 쪽을 향하는 방향성을 지니고 있으므로, 이 방향을 따라 직선을 길게 그려주면 그래프와 맞닿는 접선이 그려지는데, 이 선이 바로 이 점에서의 기울기가 됨

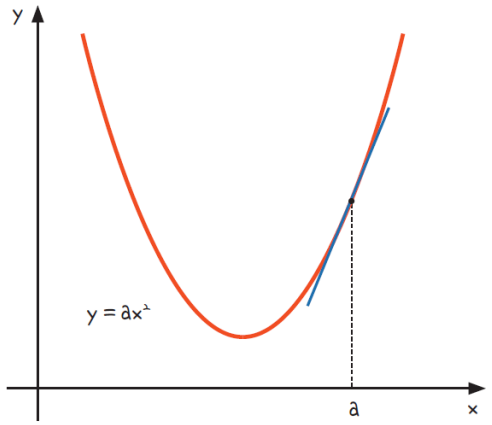
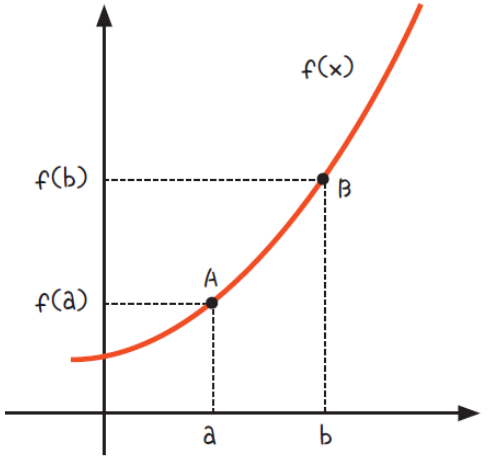



그림 2-4 a에서의 순간 변화율은 곧 기울기다!

- 미분을 한다는 것은 쉽게 말해 이 '순간 변화율'을 구한다는 것임
- 미분 계수 : 어느 순간에 어떤 변화가 일어나고 있는지를 숫자로 나타낸 것
- 미분 계수는 곧 그래프에서의 기울기를 의미함



# 미분, 순간 변화율과 기울기

- 여기서 Δ(델타)는 변화량을 나타내는 기호임

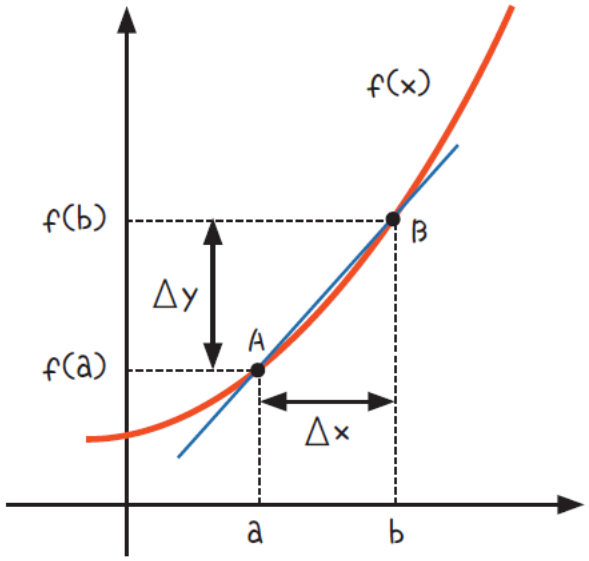



그림 2-6 A, B를 지나는 직선은
이 두 점 간의 기울기, 곧 평균 변화율을 의미

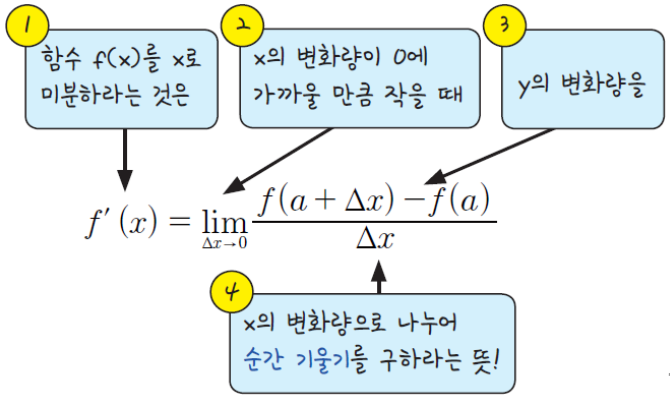
- 이 그래프에서 x값의 증가량은 $b-a$ 이고, y값의 증가량은 $f(b)-f(a)$

$$\text{직선 AB의 기울기} = \frac{y\text{값의 증가량}}{x\text{값의 증가량}} = \frac{f(b)-f(a)}{b-a} = \frac{f(a+\Delta x)-f(a)}{\Delta x}$$

- 이때 직선 AB의 기울기를 A와 B 사이의 '평균 변화율'이라고 함
- 미분을 배우고 있는 우리에게 필요한 것은 순간 변화율
- 순간 변화율: x 증가량 △x이 0에 가까울 만큼 아주 작을 때의 순간적인 기울기

$$\lim_{\Delta x \to 0} \frac{f(a+\Delta x)-f(a)}{\Delta x}$$

- $\lim_{\Delta x \to 0}$ : "x의 증가량이 0에 가까울 만큼 작을 때"라는 뜻
- 기울기는 $\frac{y\text{값의 증가량}}{x\text{값의 증가량}}$ 이므로 순간 기울기는 $\lim_{\Delta x \to 0} \frac{y\text{값의 증가량}}{x\text{값의 증가량}}$ 으로 표현되며,
  이것은 $\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{dy}{dx}$ 라고도 쓸 수 있음
- "함수 f(x)를 미분하라"는 것을 f'(x)또는 $\frac{d}{dx}f(x)$ 로 표기함



# 미분의 성질

- 다음은 딥러닝을 공부하는 과정 중에 자주 만나게 되는 중요한 4가지 미분의 성질

1 | $f(x)=a$ 에서 가 상수일 때 미분 값은 0임

2 | $f(x)=x$ 일 때의 미분 값은 1임

3 | $f(x)=ax$ 에서 a가 상수이면 미분 값은 a임

4 | $f(x)=x^a$ 에서 a가 자연수이면 미분 값은 $ax^{a-1}$ 이 됨

# 편미분

- 미분과 더불어 딥러닝을 공부할 때 가장 자주 접하게 되는 또 다른 수학 개념은 바로 편미분임
- 미분과 편미분 모두 '미분하라'는 의미에서는 다를 바가 없음
- 여러 가지 변수가 식 안에 있을 때, 모든 변수를 미분하는 것이 아니라 우리가 원하는 변수만 미분함
- 그 외에는 모두 상수로 취급

$$f(x, y) = x^2 + yx + a \text{ (a는 상수)}$$

- 변수가 f와 y중 어떤 변수로 미분해야 하는지를 정해야 하므로 편미분을 사용하는 것
- 이 식처럼 여러 변수 중에서 x에 관해서만 미분하고 싶다면, 함수 f를 "x에 관해 편미분하라"고 함 : $\frac{\partial f}{\partial x}$
- 함수 $f(x, y) = x^2 + yx + a$ 에 관해 편미분 하는 과정은 미분의 성질 4번에 따라 $x^2$항은 2x가 됨
- x에 관해 미분하면 다른 모든 항은 상수로 취급하므로 y는 상수가 됨
- 미분의 성질 3번에 따라 xy는 y가 됨
- 마지막 항 a는 미분의 성질 1번에 따라 0이 됨

$$f(x,y) = x^2 + xy + a \text{ 일 때,}$$

$$\frac{\partial f}{\partial x} = 2x + y$$

# 지수와 지수 함수

- 지수란 다음과 같은 형태를 말함

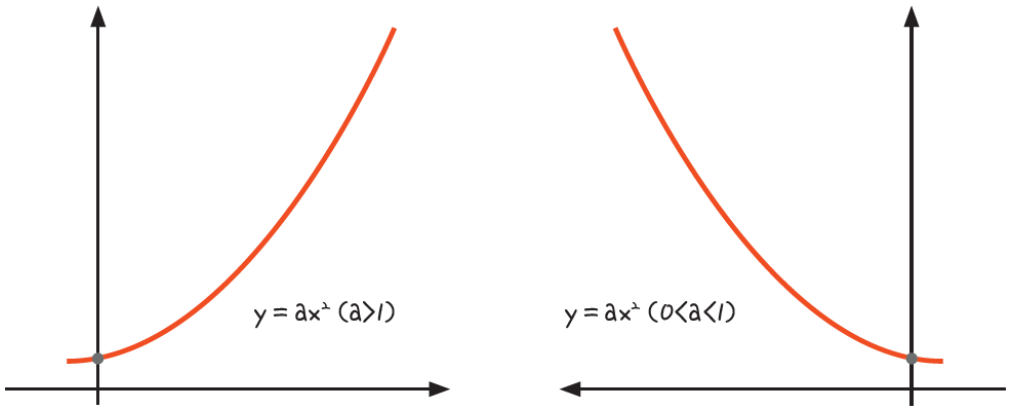
$$a^{\square}$$

- 여기서 a를 '밑'이라 하고 □를 '지수'라고 부름
- a를 □만큼 반복해서 곱한다는 뜻
- 지수 함수 : 변수 x가 지수 자리에 있는 경우를 말함

$$y = a^x (a \neq 1,\ a > 0)$$

- 지수 함수에서는 밑 a의 값이 무엇인지가 중요함
- 이 값이 1이면 함수가 아님
- 0보다 작으면 허수를 포함하게 되므로 안 됨
- 밑의 값은 a > 1이거나 0 < a < 1

그림 2-7 a>1일 때와 0<a<1일때의 지수 함수



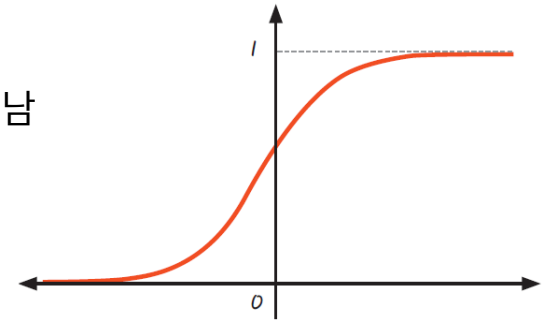
# 시그모이드 함수

- 시그모이드 함수 : 지수 함수에서 밑의 값이 자연 상수 e인 함수를 말함
- 자연 상수 e : '자연 로그의 밑', '오일러의 수' 등 여러 이름으로 불림

  파이 π처럼 수학에서 중요하게 사용되는 무리수(그 값은 대략 2.718281828...)
- 자연 상수 e가 지수 함수에 포함되어 분모에 들어가면 시그모이드 함수가 됨

$$f(x) = \frac{1}{1 + e^{-x}}$$

- 시그모이드 함수를 그래프로 그려보면 그림 2-8과 같이 S자 형태로 나타남

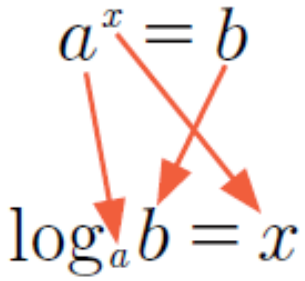


# 로그와 로그 함수

- 로그를 이해하려면 먼저 지수부터 이해해야 함

$$a^x = b$$

- 로그 : x를 구하기 위해 사용하는 방법
- 영어로 logarithm이라고 하는데 앞 세 글자 log를 씀
- 지수 식에서 a와 b의 위치를 다음과 같이 바꾸어 써주면 됨




- 로그 함수와 지수 함수는 서로 역함수의 관계
- 역함수는 x와 y를 서로 바꾸어 가지는 함수임
- 지수 함수 $y = a^x (a \neq 1,\ a > 0)$ 는 로그의 정의를 따라 $x = \log_a y$ 로 바꿀 수 있음
- 역함수를 만들기 위해 x와 y를 서로 바꾸어 주면 됨
- 역함수의 그래프는 $y = x$에 대하여 대칭인 선으로 나타남

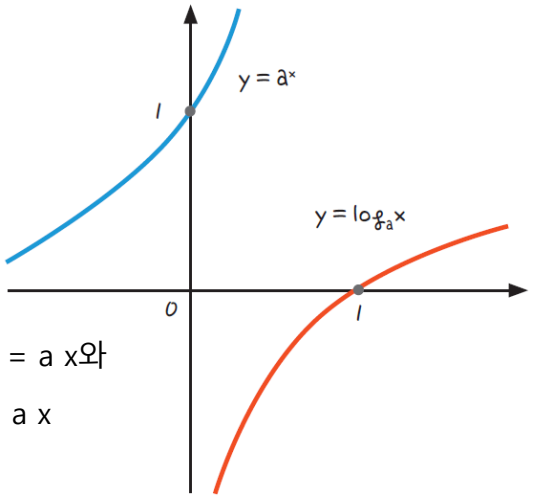



그림 2-9 지수 함수 y = a x와 로그 함수 y = log a x

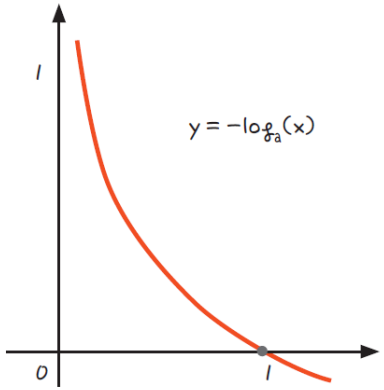
# 로그와 로그 함수

- 우리는 x가 1에 가까워지거나, 0에 가까워질수록 오차가 커지는 그래프가 필요함

**그림 2-10**

$y = -\log_a(x)$

1 | $x$축에 대하여 대칭 이동




2 | $x$축과 $y$축에 대하여 대칭 이동

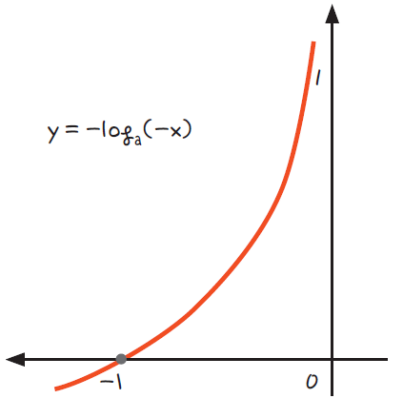



**그림 2-11**

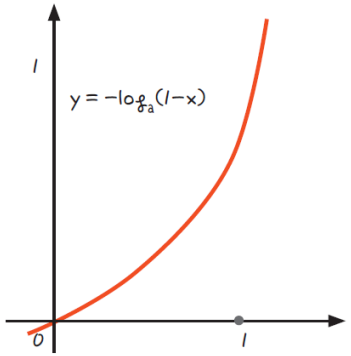
$y = -\log_a(-x)$

**그림 2-13**

$y = -\log_a x$

$y = -\log_a(1-x)$

3 | 2번 그래프를 $x$축 오른쪽 방향으로 1만큼 평행 이동




4 | 1번과 3번을 함께 나타낸 그래프

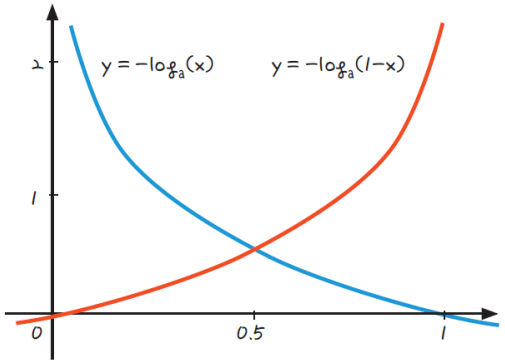



**그림 2-12**

$y = -\log_a(1-x)$

# 3장. 예측선 긋기: 선형회귀

# 목차

# 딥러닝의 기본적인 계산 원리

- 딥러닝의 가장 말단에서 이루어지는 기본적인 계산 원리 2가지

  1. 선형 회귀
  2. 로지스틱 회귀

# 선형회귀

**전제 : 'x값이 변함에 따라 y값도 변한다'**

- 독립 변수 : 독립적으로 변할 수 있는 x값
- 종속 변수 : 독립 변수에 따라 종속적으로 변하는 값
- 선형 회귀 : 독립 변수  x를 사용해 종속 변수 y의 움직임을 예측하고 설명하는 작업
- 단순 선형 회귀(simple linear regression) : 하나의 x값 만으로도 y값 설명 가능
- 다중 선형 회귀(multiple linear regression) : y값 설명에 x값이 여러 개 필요할 때

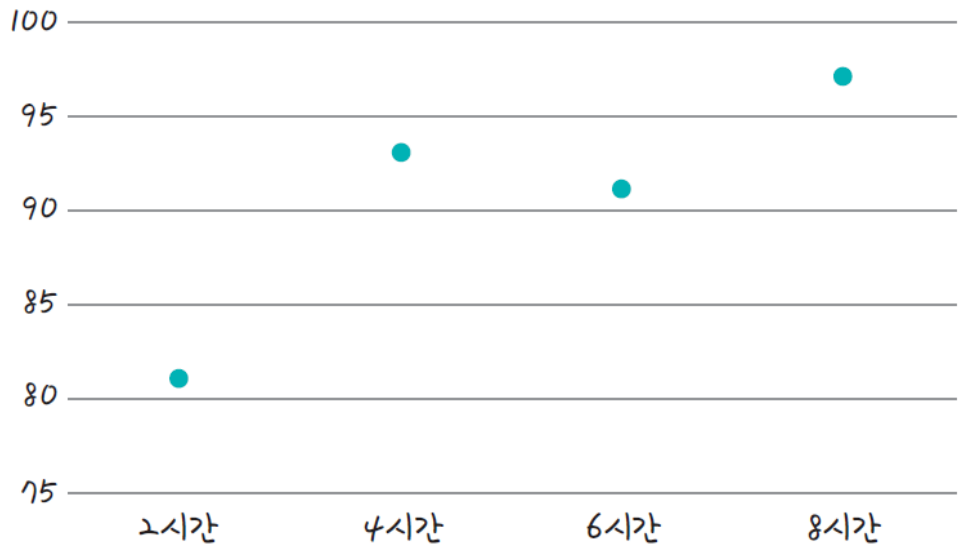
# 가장 훌륭한 예측선이란?

- 단순 선형 회귀 예시 : 공부한 시간 x, 성적 y

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

- 집합 X와 집합 Y를 다음과 같이 표현 가능

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



# 선형회귀

- 이 점들의 특징을 가장 잘 나타내는 선을 그리는 것이 선형 회귀
- 선은 직선이므로 일차 함수 그래프
- x값은 독립 변수이고 y값은 종속 변수(x값에 따라 y값은 달라짐)
- 다만, 정확하게 계산하려면 상수  a와  b의 값을 알아야 함
- 이 직선을 훌륭하게 그으려면 직선의 기울기 a값과 y절편 b값을 정확히 예측해 내야 함

# 최소 제곱법(일차 함수만 적용)

- 최소 제곱법(method of least squares) : 일차 함수의 기울기 a와 y절편 b를 구할 수 있음

  (다항 함수인 경우에는 경사하강법을 이용)

- 지금 가진 정보가 x값(입력 값, '공부한 시간')과 y값(출력 값, '성적')일 때, 기울기 a는

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

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

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

$$= \frac{46}{20}$$

$$= 2.3$$

# 최소 제곱법(일차 함수만 적용)

- 다음은 y절편인 b를 구하는 공식

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

- 즉, y의 평균에서 x의 평균과 기울기의 곱을 빼면 b의 값이 나온다는 의미

$$b = 90.5 - (2.3 \times 5)$$
$$= 79$$

- 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성됨

$$y = 2.3x + 79$$

# 최소 제곱법(일차 함수만 적용)

$$y = 2.3x + 79$$

- 예측 값 : 어떤 x를 대입했을 때 나오는 y값

| 공부한 시간 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| 성적 | 81 | 93 | 91 | 97 |
| **예측 값** | **83.6** | **88.2** | **92.8** | **97.4** |

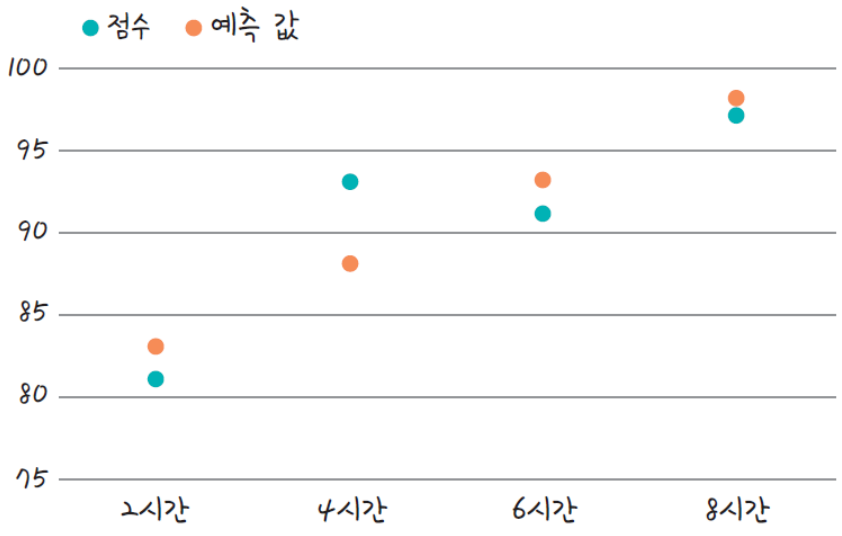



그림 3-2 공부한 시간, 성적, 예측 값을 좌표로 표현

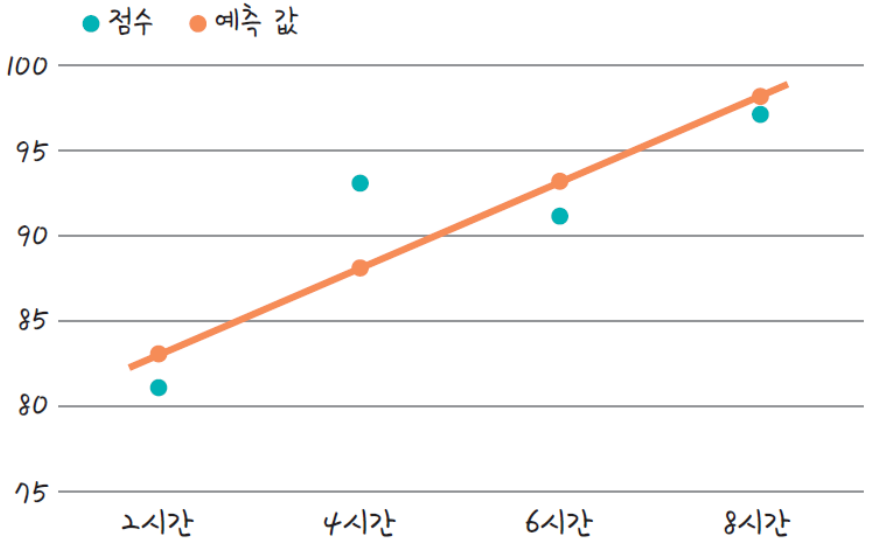



그림 3-3 오차가 최저가 되는 직선의 완성

# 코딩으로 확인하는 최소 제곱법(일차 함수만 적용)

예제 소스 : deeplearning_class/01_Least_Square_Method.ipynb

```
# -*- coding: utf-8 -*-
import numpy as np

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

# x와 y의 평균값
mx = np.mean(x)
my = np.mean(y)
print("x의 평균값:", mx)
print("y의 평균값:", my)

# 기울기 공식의 분모
divisor = sum([(mx - i)**2 for i in x])

# 기울기 공식의 분자
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)

print("분모:", divisor)
print("분자:", dividend)

# 기울기와 y 절편 구하기
a = dividend / divisor
b = my - (mx*a)

# 출력으로 확인
print("기울기 a =", a)
print("y 절편 b =", b)
```

데이터 값들을 리스트 x, y로 정의

TIP

파이썬 리스트를 만들려면 다음과 같이 리스트 이름을 정한 후 대괄호([ ])로 감싼 요소들을 쉼표(,)로 구분해 대입하면 됩니다.

리스트 이름 = [요소 1, 요소 2, 요소 3, ...]

mx, my
: x, y 각각 평균값 구하기

5.0
90.5

divisor(분모)
: x의 각 원소와 x의 평균값들의 차를 제곱해서 더한다.

다음과 같이 분자 값을 구하는 함수(top함수)를 정의하여

dividend 변수에 분자의 값을 저장

20.0
46.0

```
x의 평균값: 5.0
y의 평균값: 90.5
분모: 20.0
분자: 46.0
기울기 a = 2.3
y 절편 b = 79.0
```

**y = 2.3x + 79.0**

# 평균 제곱 오차(mse)

- 평균 제곱 오차(mean square error, MSE) : 딥러닝에서 오차를 평가하는 방법 중 가장 많이 사용됨

- 최소 제곱법과 다르게 여러 변수가 있을 때도 사용 가능

- 여러 개의 입력 값을 계산할 때 임의의 선을 그리고 난 후,
  이 선이 얼마나 잘 그려졌는지를 평가하여 조금씩 수정해 가는 방법

- 가설을 하나 세운 뒤, 이 값이 주어진 요건을 충족하는지 판단하여 조금씩 변화를 주고,
  이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법

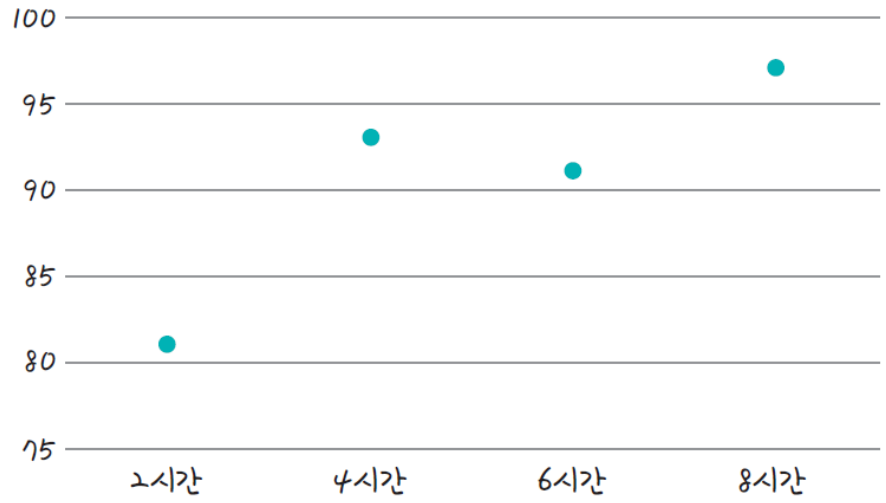
**“일단 선을 그어보고, 조금씩 수정해가기”**

# Loss function(오차 함수) 종류

| 계열 | 오차함수 종류 | 의미 | 사용처 | 특징 |
|---|---|---|---|---|
| 평균제곱 계열<br>(Mean squared) | MSE<br>(Mean squared error) | 평균제곱 오차 | 회귀 문제 | 속도가 느리다는 단점 |
| | MAE | 평균절대 오차 | | |
| | MAPE | 평균절대백분율 오차 | | |
| | MSLE | 평균제곱로그 오차 | | |
| 교차엔트로피 계열<br>(Cross-entropy) | categorical_<br>crossentropy | 범주형<br>교차 엔트로피 | 다중 클래스 분류 | 장점 :<br>출력 값에 로그를 취해서,<br>오차가 커지면 수렴 속도 증가,<br>오차가 작아지면 수렴 속도 감소 |
| | binary_<br>crossentropy | 이항<br>교차 엔트로피 | 이진 클래스 분류 | |

# 평균 제곱 오차(mse)

**그림 3-4** 공부한 시간과 성적의 관계도




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

- 임의의 값을 대입한 뒤 오차를 구하고 이 오차를 최소화하는 방식을 사용해서 a와 b의 값을 구해 보자
- 대강 선을 그어보기 위해서 기울기 a와 y절편 b를 임의의 수 3과 76이라고 가정해 보자

# 평균 제곱 오차(mse)

그림 3-5 임의의 직선 그려보기

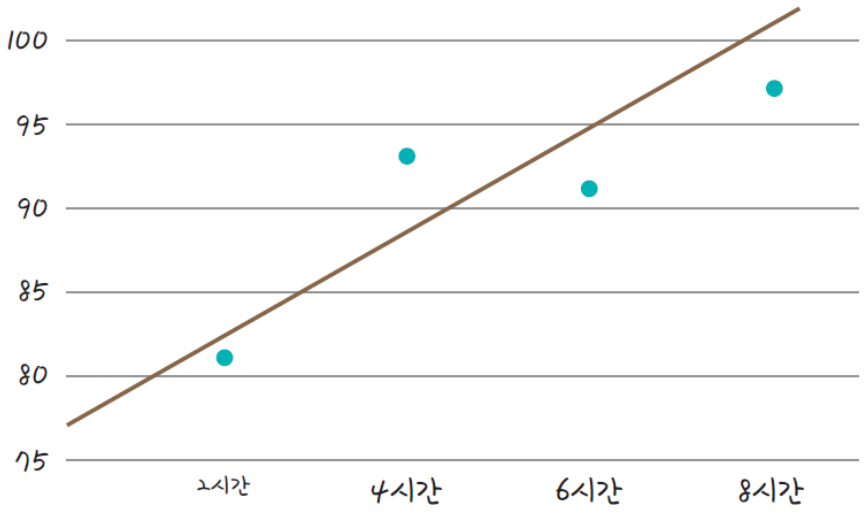
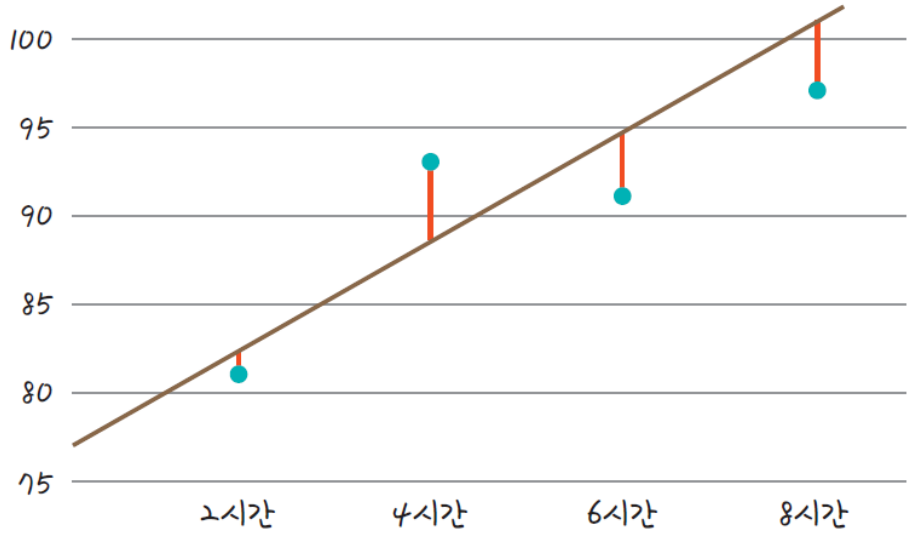



그림 3-6 임의의 직선과 실제 값 사이의 거리




- 임의의 직선이 어느 정도의 오차가 있는지를 확인하려면 각 점과 그래프 사이의 거리를 재면 됨

# 평균 제곱 오차(mse)

그림 3-7 기울기를 너무 크게 잡았을 때의 오차

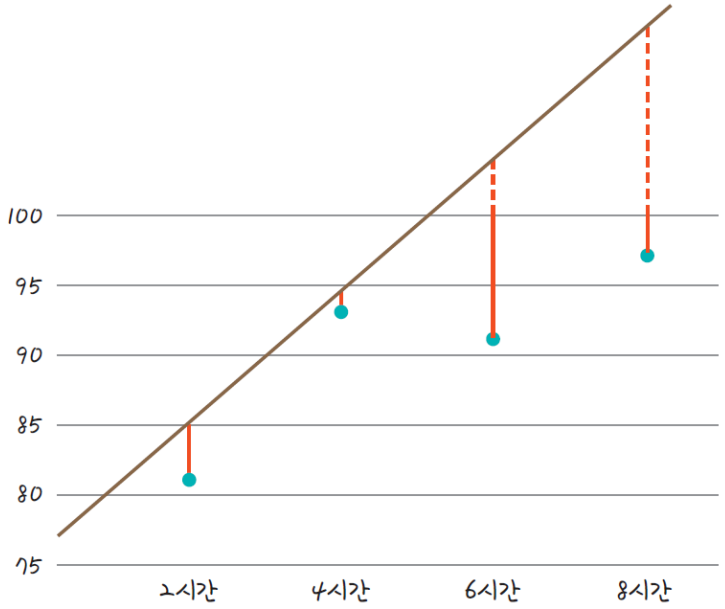
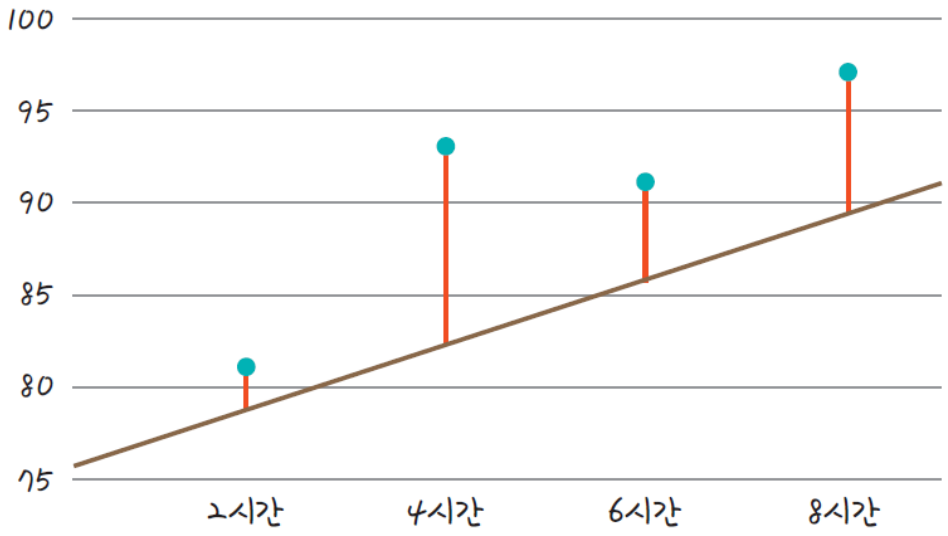



그림 3-8 기울기를 너무 작게 잡았을 때의 오차



# 평균 제곱 오차(mse)

- 그래프의 기울기가 잘못 되었을수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
- 만약 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있는 것을 알 수 있음
- 거리는 입력 데이터에 나와 있는 y의 '실제 값'과 x를 오차 = 예측 값 – 실제 값 의 식에 대입해서 나오는 '예측 값'과의 차이를 통해 구할 수 있음

| 공부한 시간(x) | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| 성적(실제 값, y) | 81 | 93 | 91 | 97 |
| 예측 값 | 82 | 88 | 94 | 100 |
| 오차 | 1 | –5 | 3 | 3 |

- 이렇게 해서 구한 오차를 모두 더하면 1 + (-5) + 3 + 3 = 2가 됨
- 이 값은 오차가 실제로 얼마나 큰지를 가늠하기에는 적합하지 않음
- 오차에 양수와 음수가 섞여 있어서 오차를 단순히 더해 버리면 합이 0이 될 수도 있기 때문임
- 부호를 없애야 정확한 오차를 구할 수 있음

  -> 오차의 합을 구할 때는 각 오차의 값을 제곱해 줌 $$\text{오차의 합} = \sum_{i}^{n} (\hat{y}_i - y_i)^2$$

# 평균 제곱 오차(mse)

- 오차의 합을 구할 때는 각 오차의 값을 제곱해 줌

$$\text{오차의 합} = \sum_{i}^{n} (\hat{y}_i - y_i)^2$$

- i는 x가 나오는 순서, n은 x원소의 총 개수(n=4개)를 의미함
- $\hat{y}_i$는 $X_i$에 대응하는 '실제 값'
- $y_i$는 $X_i$ 가 대입되었을 때 직선의 방정식( $p = 3x + 76$ )이 만드는 '예측 값'
- 이 식으로 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44
- 평균 제곱 오차(Mean Squared Error, MSE) :

  오차의 합에 이어 각 x 값의 평균 오차를 이용함

  위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있음

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n} \sum (\hat{y}_i - y_i)^2$$

4

- 이 식에 의해, 앞서 그은 임의의 직선은 44/4=11의 평균제곱오차(mse)를 갖는 직선

# 평균제곱오차(mse)와 선형 회귀

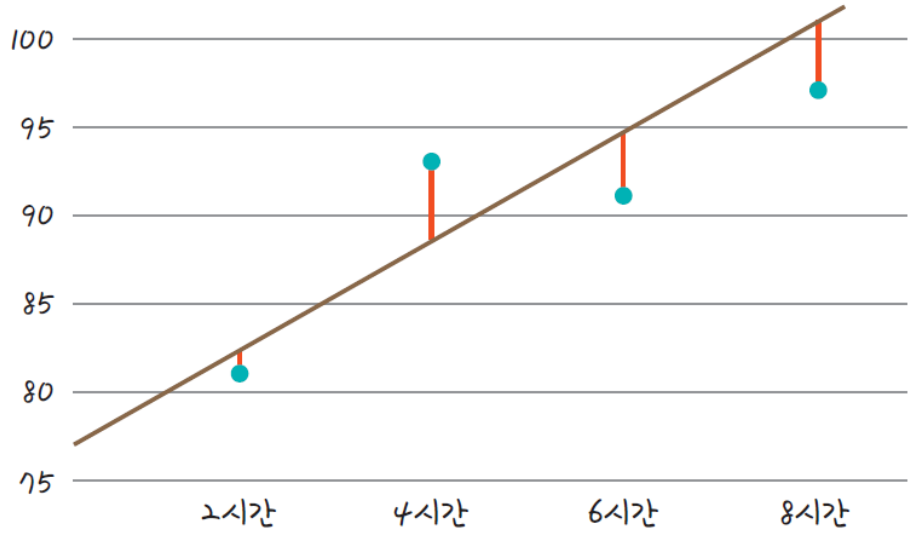
- 선형 회귀

  : 임의의 직선을 그어 이에 대한 평균 제곱 오차(mse)를 구하고,

  이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업

$$y = ax + b$$

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$



# 코딩으로 확인하는 평균 제곱 오차

예제 소스 : deeplearning_class/02_Mean_Squared_Error.ipynb

```
#-*- coding: utf-8 -*-

import numpy as np

#가상의 기울기 a와 y 절편 b
fake_a_b=[3,76]

# x 값과 y값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]

# y=ax + b에 a,b 값 대입하여 결과를 출력하는 함수
def predict(x):
    return fake_a_b[0]*x + fake_a_b[1]

# MSE 함수
def mse(y, y_hat):
    return ((y - y_hat) ** 2).mean()

# MSE 함수를 각 y값에 대입하여 최종 값을 구하는 함수
def mse_val(y, predict_result):
    return mse(np.array(y), np.array(predict_result))

# 예측값이 들어갈 빈 리스트
predict_result = []

# 모든 x값을 한 번씩 대입하여 predict_result 리스트완성.
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict(x[i])))

# 최종 MSE 출력
print("MSE 최종값: " + str(mse_val(predict_result,y)))
```

```
MSE 최종값: 11.0
```

임의로 정한 기울기 a와 y절편 b의 값이 각각 3, 76

공부한 시간 x 리스트와 성적 y 리스트를 만든다.

일차방정식 y=ax+b를 구현하는 함수 : predict

평균제곱오차(mse)를 계산해주는 함수 : mse

- 처음 가정한 a=3, b=76은 오차가 약 11.0 이라는 것 확인

- 앞으로 할 일
  - 이 오차를 줄이는 새로운 선 긋기
  - a와 b의 값을 적절히 조절하면서 오차의 변화를 살펴보고, 그 오차가 최소화되는 a, b 구하기

-> 경사 하강법

# 4장. 오차 수정하기: 경사 하강법

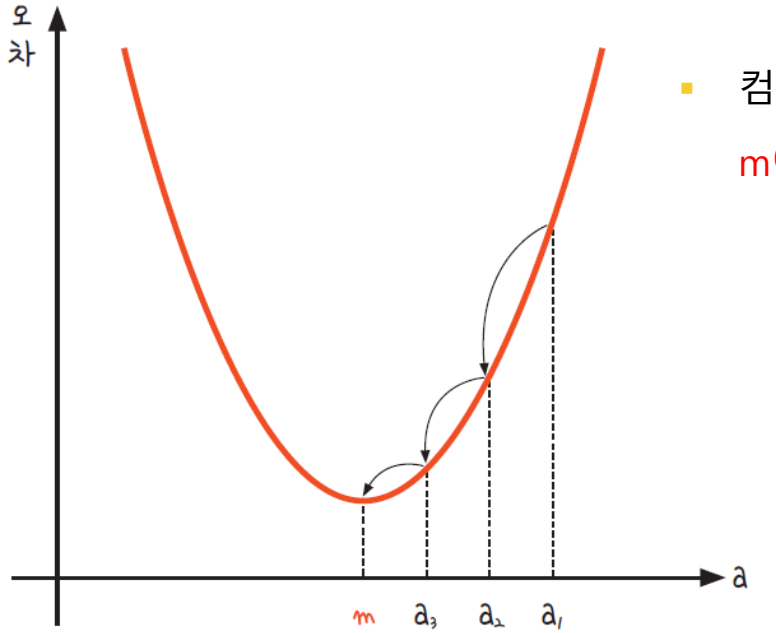
# 4장 오차 수정하기: 경사 하강법

# 경사 하강법의 개요

- a를 무한대로 키우거나 a를 무한대로 작게 할 때 오차도 무한대로 커지는 이러한 관계는 이차 함수 그래프로 표현할 수 있음

$$y = ax + b \qquad \text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$

오차가 가장 적은 점 m이 어디일까?




- 컴퓨터를 이용해 m의 값을 구하려면 임의의 한 점 $a_1$ 을 찍고, 이 점을 m에 가까운 쪽으로 점점 이동시키는 과정 ( $a_1$ -> $a_2$ -> $a_3$ )이 필요함

그림 4-1 기울기 a와 오차와의 관계: 적절한 기울기를 찾았을 때 오차가 최소화된다.

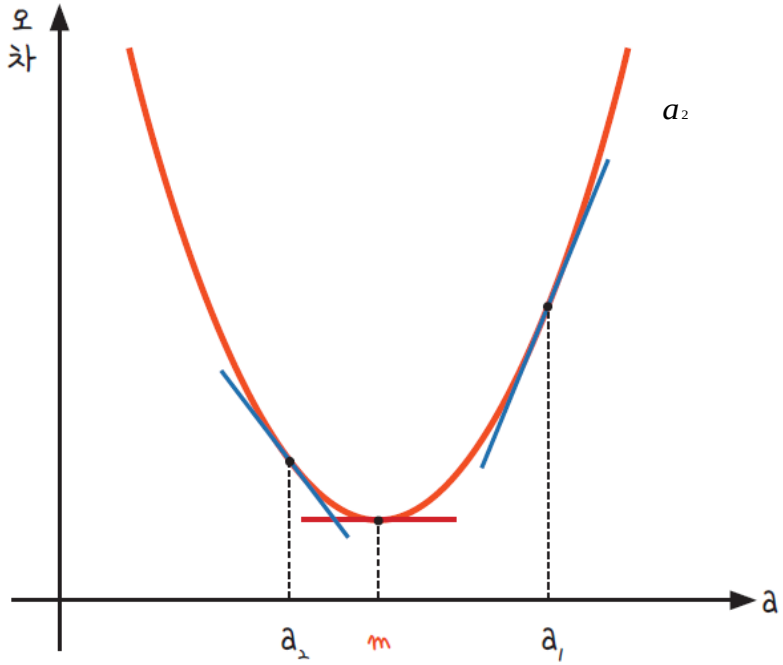
# 경사 하강법의 개요

- 경사 하강법(gradient descent)

  : 미분 기울기를 이용해서 그래프에서 오차를 비교하여 가장 작은 방향으로 이동시키는 방법

- $y=x^2$ 그래프에서 x에 다음과 같이 $a_1$ ,$a_2$ 그리고 m을 대입하여 그 자리에서 미분하면 그림 4-2처럼 각 점에서의 순간 기울기가 그려짐

그림 4-2 순간 기울기가 0인 점이 곧 우리가 찾는 오차가 최소값이 되는 점 m이다.



# 경사 하강법의 개요

- 여기서 눈여겨 봐야 할 것은 우리가 찾는 최솟값 m에서의 순간 기울기임
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x축과 평행한 선이 됨. 즉, 기울기가 0

  결국, 우리가 할 일은 '미분 값이 0인 지점'을 찾는 것이 됨

  **1 |** $a_1$에서 미분을 구한다.

  **2 |** 구해진 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킨 $a_2$에서 미분을 구한다.(그림 4-3 참조)

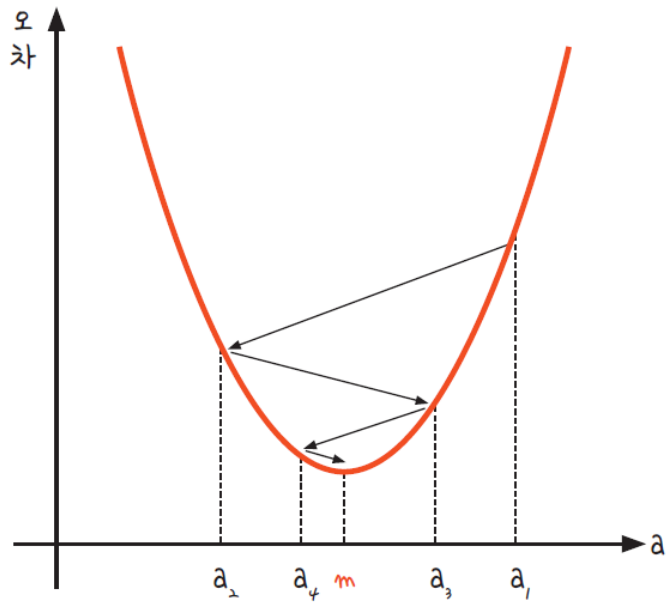
  **3 |** 위에서 구한 미분 값이 0이 아니면 위 과정을 반복한다.

  **4 |** 기울기가 0인 한 점 m으로 수렴한다.

- **경사 하강법**

  : 이렇게 반복적으로 기울기 a를 변화시켜서 m의 값을 찾아내는 방법

그림 4-3 최솟점 m을 찾아가는 과정

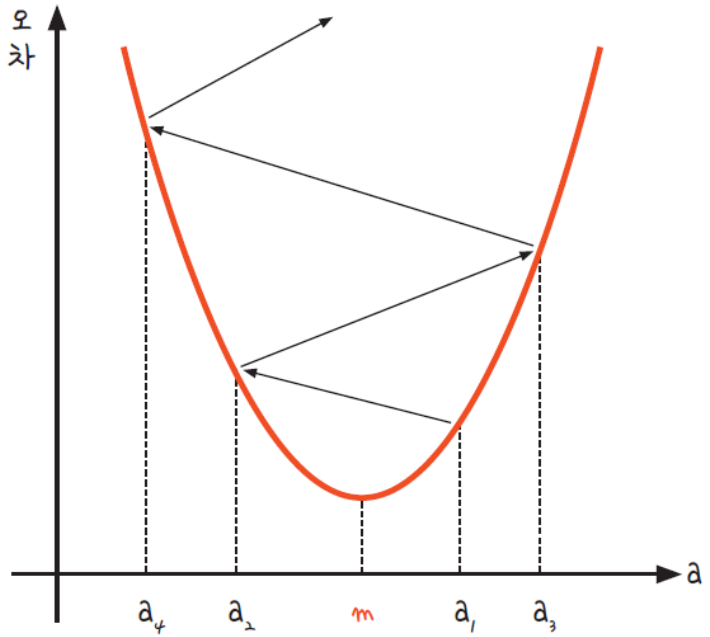


# 학습률(learning rate)

- 오차의 최소값이 되는 점 m을 찾아가는 과정에서 학습률 개념을 알 수 있다.

- 기울기의 부호를 바꿔 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면,
  a값이 한 점으로 모이지 않고 그림 4-4처럼 위로 치솟아 버림

그림 4-4 학습률을 너무 크게 잡으면 한 점으로 수렴하지 않고 발산한다.



# 학습률과 경사하강법

- 학습률(learning rate)

  : 어느 만큼 이동시킬지를 신중히 결정해야 하는데, 이때 이동 거리를 정해주는 것

- 딥러닝에서 학습률의 값을 적절히 바꾸면서 최적의 학습률을 찾는 것은 중요한 최적화(optimization) 과정 중 하나이다.

- **경사 하강법**

  : 오차의 변화에 따라 이차 함수 그래프를 만들고 적절한 학습률을 설정해 미분 값이 0인 지점 찾기

  - 최적의 b값을 구할 때 역시 경사 하강법을 사용한다.
  - b값이 너무 크면 오차도 함께 커지고, 너무 작아도 오차가 커진다.

# 코딩으로 확인하는 경사 하강법

- 최솟값을 구하기 위해서는 이차 함수에서 미분을 해야 함
- 그 이차 함수는 평균 제곱 오차를 통해 나온다는 것임
- 평균 제곱 오차의 식을 다시 옮겨 보면 다음과 같음

$$\frac{1}{n}\sum(\hat{y}_i - y_i)^2$$

- 여기서 $\hat{y}_i$은 $x_i$를 집어 넣었을 때의 값이므로 $y_i = ax_i + b$ 를 대입하면 다음과 같이 바뀜

$$\frac{1}{n}\sum((ax_i + b) - y_i)^2$$

- 이 값을 미분할 때 우리가 궁금한 것은 a 와 b라는 것에 주의해야 함. 식 전체를 미분하는 것이 아니라 필요한 값을 중심으로 미분해야 하기 때문(a와 b로 각각 *편미분)

$$a\text{로 편미분 한 결과} = \frac{2}{n}\sum(ax_i + b - y_i)x_i$$

$$b\text{로 편미분 한 결과} = \frac{2}{n}\sum(ax_i + b - y_i)$$

*편미분: 2장 딥러닝을 위한 기초 수학

# 코딩으로 확인하는 경사 하강법

TIP

### a로 편미분한 결과 유도 과정

$$\begin{aligned}\frac{a}{\partial a}MSE(a,b) &= \frac{1}{n}\sum[(ax_i+b-y_i)^2]' \\ &= \frac{2}{n}(ax_i+b-y_i)[(ax_i+b-y_i)]' \\ &= \frac{2}{n}\sum(ax_i+b-y_i)x_i\end{aligned}$$

### b로 편미분한 결과 유도 과정

$$\begin{aligned}\frac{a}{\partial a}MSE(a,b) &= \frac{1}{n}\sum[(ax_i+b-y_i)^2]' \\ &= \frac{2}{n}(ax_i+b-y_i)[(ax_i+b-y_i)]' \\ &= \frac{2}{n}\sum(ax_i+b-y_i)\end{aligned}$$

# 코딩으로 확인하는 경사 하강법

```
y_pred = a * x_data + b   # 오차 함수인 y = ax + b를 정의한 부분
error = y_data - y_pred   # 실제값 - 예측값, 즉 오차를 구하는 식


# 평균 제곱 오차를 a로 미분한 결과
a_diff = -(2 / len(x_data)) * sum(x_data * (error))
# 평균 제곱 오차를 b로 미분한 결과
b_diff = -(2 / len(x_data)) * sum(y_data - y_pred)
```

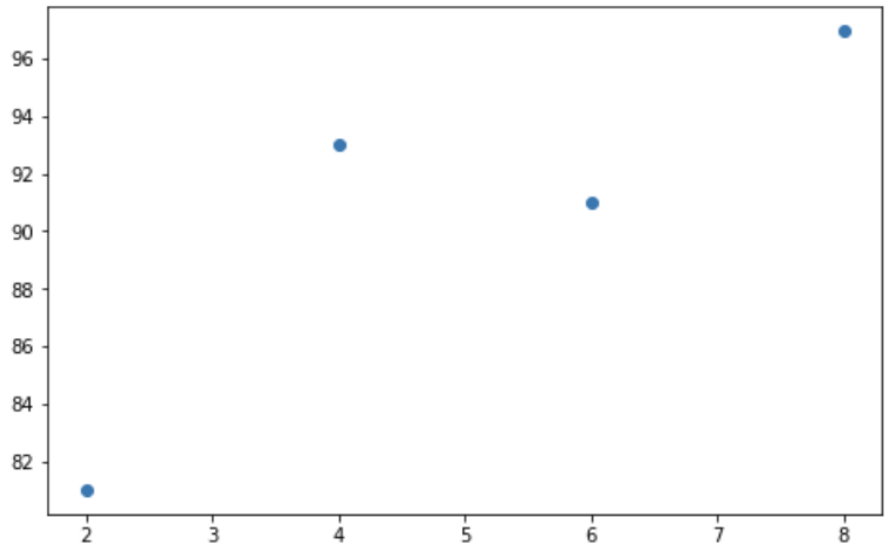
- 여기에 학습률을 곱해 기존의 a값과 b값을 업데이트해 줌

```
a = a - lr * a_diff   # 미분 결과에 학습률을 곱한 후 기존의 a값을 업데이트
b = b - lr * b_diff   # 미분 결과에 학습률을 곱한 후 기존의 b값을 업데이트
```

# 코딩으로 확인하는 경사 하강법

예제 소스: deeplearning_class/03_Linear_Regression.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

#그래프로 나타내 봅니다.
plt.figure(figsize=(8,5))
plt.scatter(x, y)
plt.show()
```



# 코딩으로 확인하는 경사 하강법

예제 소스: deeplearning_class/03_Linear_Regression.ipynb

```
#리스트로 되어 있는 x와 y값을 넘파이 배열로 바꾸어 줍니다.(인덱스를 주어 하나씩 불러와 계산이 가능해 지도록 하기 위함입니다.)
x_data = np.array(x)
y_data = np.array(y)

# 기울기 a와 절편 b의 값을 초기화 합니다.
a = 0
b = 0
```

```
#학습률을 정합니다.
lr = 0.03
```

학습률 = 0.03

TIP 여기서 에포크(epoch)는 입력 값에 대해 몇 번이나 반복하여 실험했는지를 나타냅니다. 우리가 설정한 실험을 반복하고 100번마다 결과를 내놓습니다.

```
#몇 번 반복될지를 설정합니다.
epochs = 2001
```

오차 수정(경사하강법) 횟수

경사하강법 실행

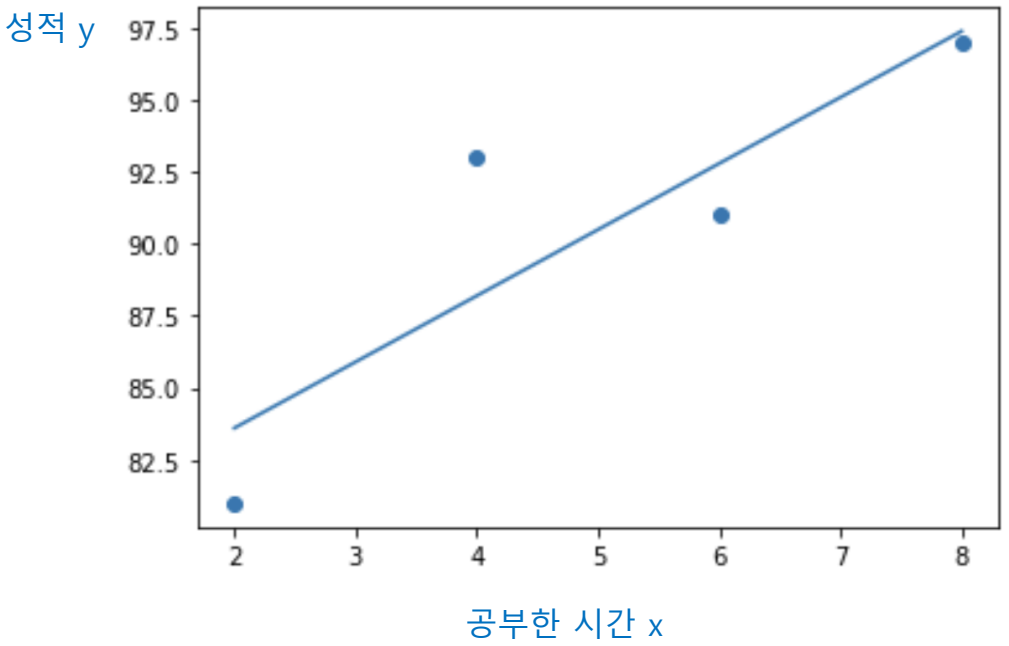
```
#경사 하강법을 시작합니다.
for i in range(epochs): # epoch 수 만큼 반복
    y_hat = a * x_data + b  #y를 구하는 식을 세웁니다
    error = y_data - y_hat  #오차를 구하는 식입니다.
    a_diff = -(2/len(x_data)) * sum(x_data * (error)) # 오차함수를 a로 미분한 값입니다.
    b_diff = -(2/len(x_data)) * sum(error)  # 오차함수를 b로 미분한 값입니다.
    a = a - lr * a_diff  # 학습률을 곱해 기존의 a값을 업데이트합니다.
    b = b - lr * b_diff  # 학습률을 곱해 기존의 b값을 업데이트합니다.
    if i % 100 == 0:    # 100번 반복될 때마다 현재의 a값, b값을 출력합니다.
        print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))
```

```
epoch=0, 기울기=27.8400, 절편=5.4300
epoch=100, 기울기=7.0739, 절편=50.5117
epoch=200, 기울기=4.0960, 절편=68.2822
epoch=300, 기울기=2.9757, 절편=74.9678
(중략)
epoch=1800, 기울기=2.3000, 절편=79.0000
epoch=1900, 기울기=2.3000, 절편=79.0000
epoch=2000, 기울기=2.3000, 절편=79.0000
```

경사하강법 실행 결과(학습률=0.03, epoch=2001)
기울기 a=2.3, y절편 b=79로 수렴하였다.

# 코딩으로 확인하는 경사 하강법

```
# 앞서 구한 기울기와 절편을 이용해 그래프를 그려 봅니다.
y_pred = a * x_data + b
plt.scatter(x, y)
plt.plot([min(x_data), max(x_data)], [min(y_pred), max(y_pred)])
plt.show()
```




$$y = ax + b$$

$$y = 2.3x + 79$$

# 다중 선형 회귀

# 다중 선형 회귀란

- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며, 정보를 추가해 새로운 예측값을 구하려면 변수의 개수를 늘려 다중 선형 회귀를 만들어 주어야 함

기존의 독립변수 '공부한 시간'(x1) 외에 '과외 수업 횟수'(x2) 변수를 추가해서 종속변수 '성적'(y) 예측

| 공부한 시간($x_1$) | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| **과외 수업 횟수($x_2$)** | 0 | 4 | 2 | 3 |
| 성적(y) | 81 | 93 | 91 | 97 |

- 그럼 지금부터 두 개의 독립 변수 $x_1$과 $x_2$가 생긴 것임
- 이를 사용해 종속 변수 y를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴

$$y = a_1 x_1 + a_2 x_2 + b$$

- 두 기울기 $a_1$, $a_2$를 구하는 방법 : 경사하강법

# 코딩으로 확인하는 다중 선형 회귀

예제 소스: deeplearning_class/04_Multi-Linear-Regression.ipynb

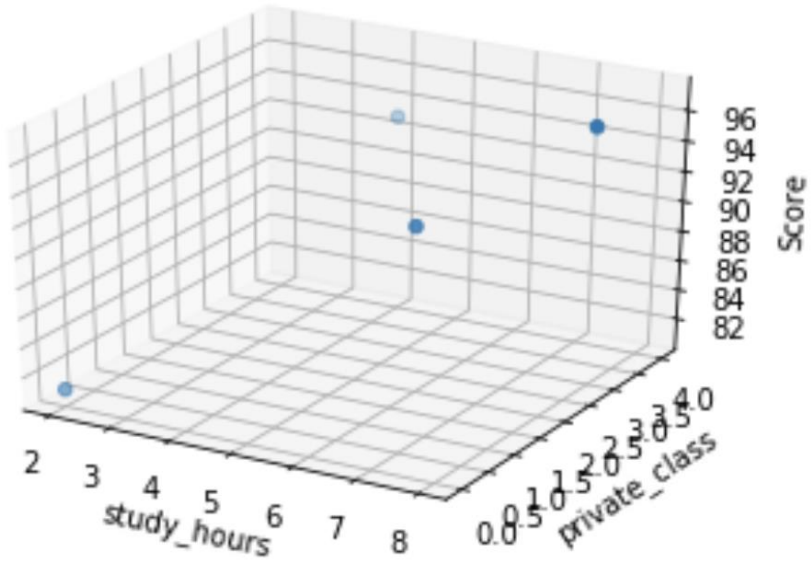
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#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]

#그래프로 확인해 봅니다.
ax = plt.axes(projection='3d')
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.dist = 11
ax.scatter(x1, x2, y)
plt.show()
```

이번에는 독립변수 x의 값이 두 개이므로 다음과 같이

data 리스트를 만들고,

x1과 x2라는 두 개의 독립 변수 리스트를 만들어 줌

그림 4-6 축이 하나 더 늘어 3D로 배치된 모습



# 코딩으로 확인하는 다중 선형 회귀

- 이제 x가 두 개가 되었으므로 $x_1$과 $x_2$두 가지의 변수를 지정함
- 각각의 값에 기울기 a값이 다르므로 기울기도 $a_1$과 $a_2$이렇게 두 가지를 만듦
- 경사 하강법을 적용하고 학습률을 곱해 기존의 값을 업데이트 함

# 코딩으로 확인하는 다중 선형 회귀

예제 소스: deeplearning_class/04_Multi-Linear-Regression.ipynb

```
#리스트로 되어 있는 x와 y값을 넘파이 배열로 바꾸어 줍니다.(인덱스를 주어 하나씩 불러와 계산이 가능해 지도록 하기 위함)
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 = 0.02
```

학습률 = 0.02

```
#몇 번 반복될지를 설정합니다.(0부터 세므로 원하는 반복 횟수에 +1을 해 주어야 합니다.)
epochs = 2001
```

오차 수정(경사하강법) 횟수

```
#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 = -(2/len(x1_data)) * sum(x1_data * (error)) # 오차함수를 a1로 미분한 값입니다.
    a2_diff = -(2/len(x2_data)) * sum(x2_data * (error)) # 오차함수를 a2로 미분한 값입니다.
    b_new = -(2/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))
```

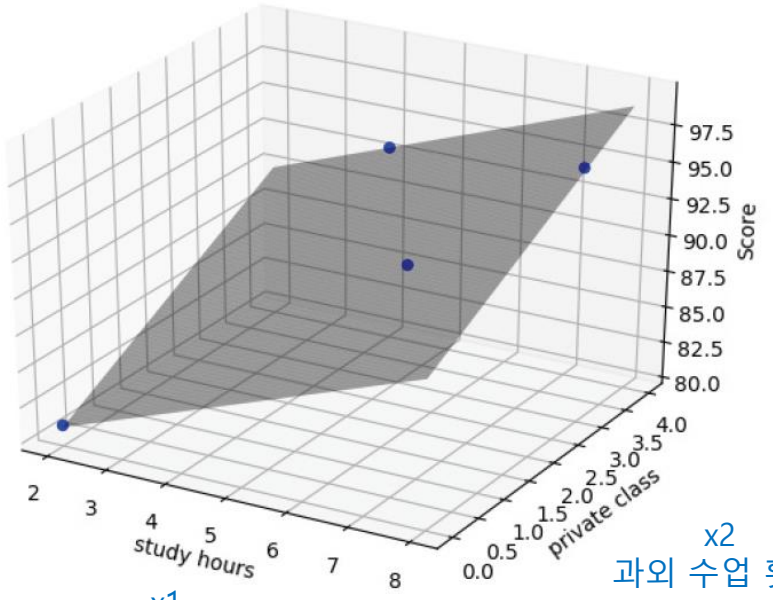
경사하강법 실행

```
epoch=0, 기울기1=18.5600, 기울기2=8.4500, 절편=3.6200
epoch=100, 기울기1=7.2994, 기울기2=4.2867, 절편=38.0427
epoch=200, 기울기1=4.5683, 기울기2=3.3451, 절편=56.7901
(중략)
epoch=1800, 기울기1=1.5001, 기울기2=2.2858, 절편=77.8563
epoch=1900, 기울기1=1.5001, 기울기2=2.2857, 절편=77.8567
epoch=2000, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8569
```

경사하강법 실행 결과(학습률=0.02, epoch=2001)
기울기$a_1$=1.5, 기울기$a_2$=2.2857, y절편 b=77.8569로 수렴하였다.

# 코딩으로 확인하는 다중 선형 회귀

- 다중 선형 회귀 문제에서의 기울기 $a_1$, $a_2$와 절편 b의 값을 찾아 확인할 수 있음




y
성적

x2
과외 수업 횟수

x1
공부한 시간

그림 4-7 다중 선형 회귀의 그래프: 차원이 하나 더 늘어난 모습

- 1차원 예측 직선이 3차원 '예측 평면'으로 바뀜
- 과외 수업 횟수(private class)라는 새로운 변수가 추가됨
- 1차원 직선에서만 움직이던 예측 결과가 더 넓은 평면 범위 안에서 움직이게 됨
- 이로 인해 좀 더 정밀한 예측을 할 수 있게 됨

# 5장. 참 거짓 판단 장치: 로지스틱 회귀

# 5장 참 거짓 판단 장치: 로지스틱 회귀

# 참 거짓 판단 장치: 로지스틱 회귀

- 전달받은 정보를 놓고 참과 거짓 중에 하나를 판단해 다음 단계로 넘기는 장치들이 딥러닝 내부에서 쉬지 않고 작동함
- 딥러닝을 수행한다는 것은 겉으로 드러나지 않는 '미니 판단 장치'들을 이용해서 복잡한 연산을 해낸 끝에 최적의 예측 값을 내놓는 작업
- 참인지 거짓인지를 구분하는 로지스틱 회귀의 원리를 이용해 '참, 거짓 미니 판단 장치'를 만들어 주어진 입력 값의 특징을 추출(학습, train)
- 이를 저장해서 '모델(model)'을 만듦
- 누군가 비슷한 질문을 하면 지금까지 만들어 놓은 이 모델을 꺼내어 답을 함(예측, prediction)
- 이것이 바로 딥러닝의 동작 원리

# 로지스틱 회귀의 정의

- 직선으로 해결하기에는 적절하지 않은 경우도 있음
- 점수가 아니라 오직 합격 또는 불합격만 발표되는 시험이 있다고 하자

| 공부한 시간 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
|---|---|---|---|---|---|---|---|
| 합격 여부 | 불합격 | 불합격 | 불합격 | 합격 | 합격 | 합격 | 합격 |

표 5-1 공부한 시간에 따른 합격 여부

- 합격을 1, 불합격을 0이라 하고, 이를 좌표 평면에 표현하면 다음과 같음

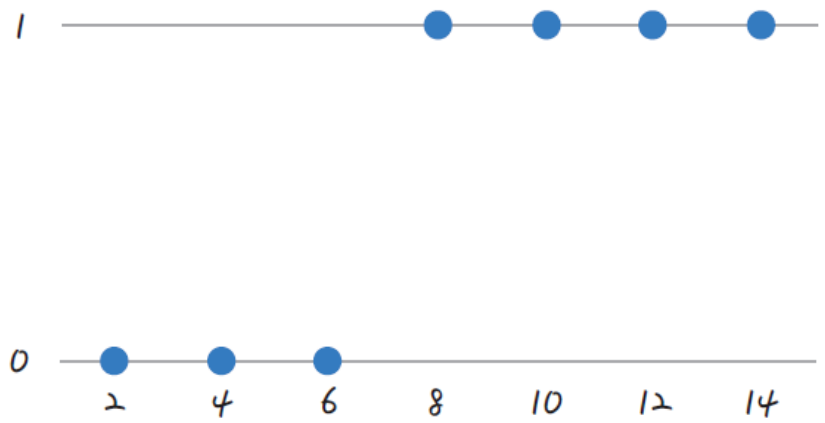



그림 5-1 합격과 불합격만 있을때의 좌표 표현

# 로지스틱 회귀의 정의

- 이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기가 어려움
- 점들의 특성을 정확하게 담아내려면 직선이 아니라 다음과 같이 S자 형태여야 함

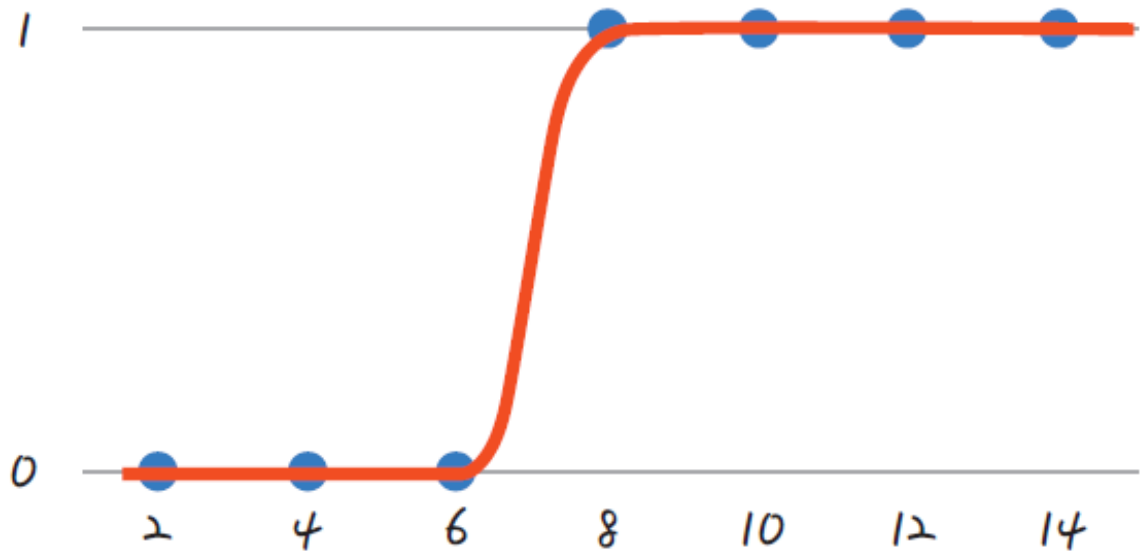



그림 5-2 각 점의 특성을 담은 선을 그었을 때

- **로지스틱 회귀** :

  선형 회귀와 마찬가지로 적절한 선을 그려가는 과정

- 다만 직선이 아니라, 참(1)과 거짓(0) 사이를 구분하는 S자 형태의 선을 그어 주는작업

# 시그모이드 함수

- 시그모이드 함수(sigmoid function) : S자 형태로 그래프가 그려지는 함수

  -> 로지스틱 회귀를 위해서 시그모이드 함수가 필요

- 시그모이드 함수를 이용해 로지스틱 회귀를 풀어나가는 공식은 다음과 같음

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

- a는 그래프의 경사도를 결정함

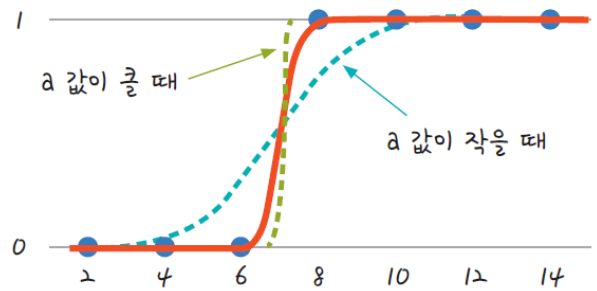



그림 5-3 a 값이 클 때와 작을 때의 그래프 변화

a값이 커지면 그래프의 경사가 커지고
a값이 작아지면 그래프의 경사가 작아진다.

- b는 그래프의 좌우 이동을 의미함

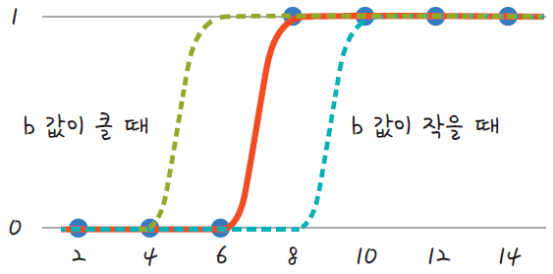



그림 5-4 b 값이 클 때와 작을 때의 그래프 변화

b값이 커지면 그래프가 왼쪽으로 이동하고
b값이 작아지면 그래프가 오른쪽으로 이동한다.

# 시그모이드 함수

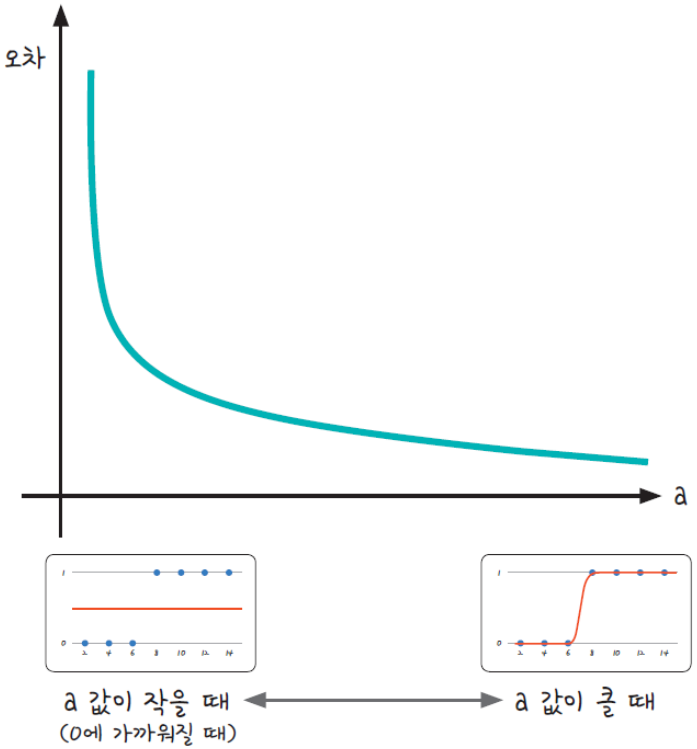



**그림 5-5** a와 오차와의 관계:

a가 작아질수록 오차는 무한대로 커지지만,

a가 커진다고 해서 오차가 없어지지는 않는다.

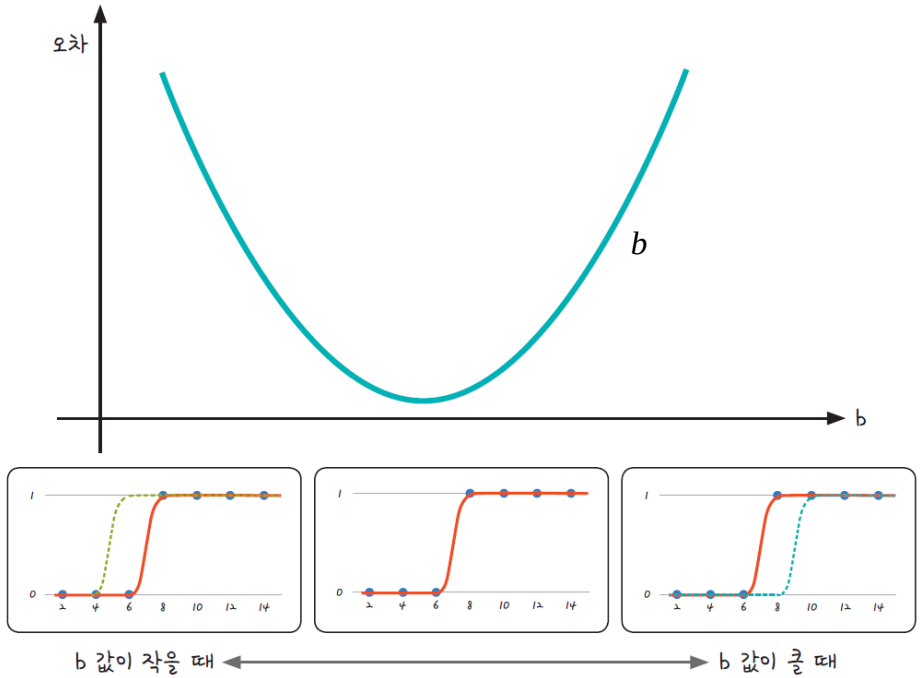



**그림 5-6** b와 오차와의 관계:

b 값이 너무 작아지거나 커지면 오차도 이에 따라 커진다.

# 오차 공식

- 시그모이드 함수에서 a, b 값을 구하는 방법 역시 경사하강법
- 경사 하강법은 먼저 오차를 구한 다음 오차가 작은 쪽으로 이동시키는 방법이므로 여기서도

오차(예측 값과 실제 값의 차이)를 구하는 공식이 필요함

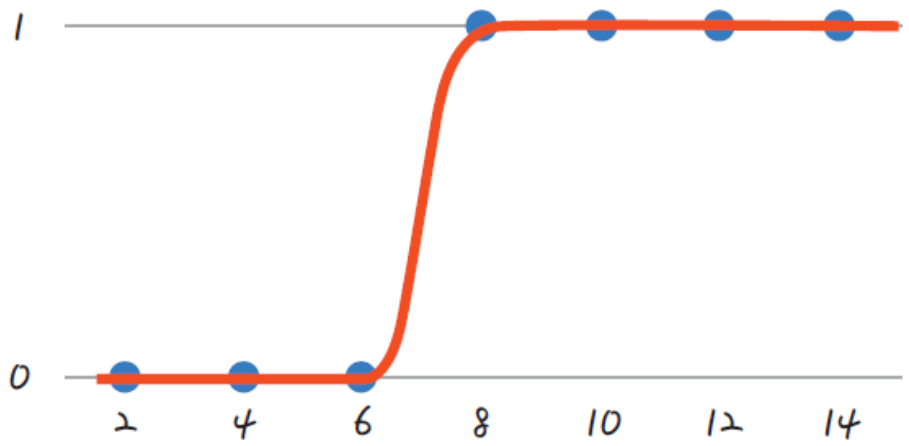
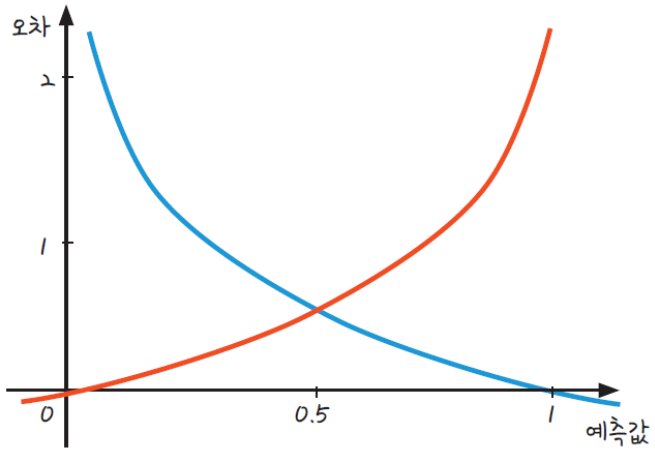



그림 5-7 시그모이드 함수 그래프

# 시그모이드 오차 공식을 위한 로그함수

- 시그모이드 함수의 특징은 y값이 0과 1 사이라는 것임
- 실제 값이 1일 때 예측 값이 0에 가까워지면 오차가 커짐
- 반대로, 실제 값이 0일 때 예측 값이 1에 가까워지는 경우에도 오차는 커짐
- 이를 공식으로 만들 수 있게 해 주는 함수가 바로 로그 함수임




- 파란색 선은 실제 값이 1일 때 사용할 수 있는 그래프

  - 예측 값이 1일 때 오차는 0, 예측 값이 0에 가까울수록 오차 커짐

- 빨간색 선은 반대로 실제 값이 0일 때 사용할 수 있는 함수

  - 예측 값이 0일때 오차는 0, 예측 값이 1에 가까울수록 오차 커짐

$$-\{\underbrace{y\_data \log h}_{A} + \underbrace{(1-y\_data)\log(1-h)}_{B}\}$$

- 실제 값(y_data)이 1이면 B 부분이 없어지고, 실제 값(y_data)이 0이면 A부분이 없어짐
- 실제 값에 따라 빨간색 그래프와 파란색 그래프를 각각 사용할 수 있게 됨

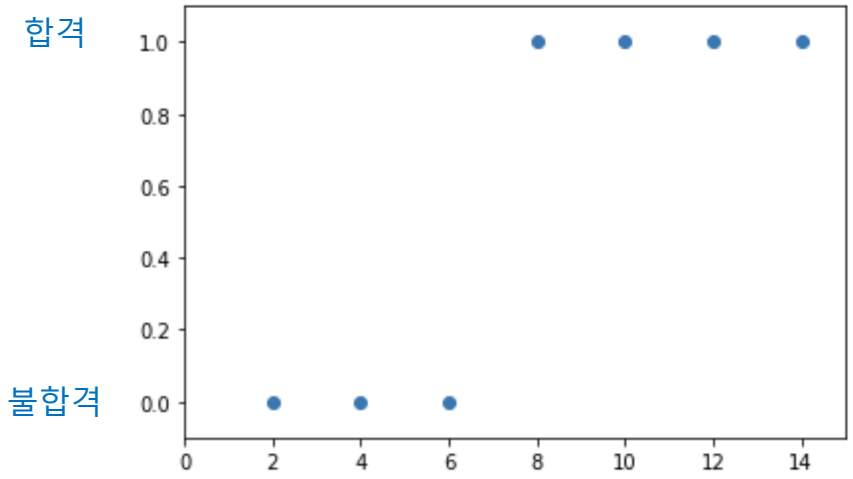
# 코딩으로 확인하는 로지스틱 회귀

예제 소스: deeplearnig_class/05_Logistic_regression.ipynb

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

#공부시간 X와 성적 Y의 리스트를 만듭니다.
data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14, 1]]

x_data = [i[0] for i in data]
y_data = [i[1] for i in data]

#그래프로 나타내 봅니다.
plt.scatter(x_data, y_data)
plt.xlim(0, 15)
plt.ylim(-.1, 1.1)
```

| 공부한 시간 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
|---|---|---|---|---|---|---|---|
| 합격 여부 | 불합격 | 불합격 | 불합격 | 합격 | 합격 | 합격 | 합격 |

Out[1]: (-0.1, 1.1)



# 코딩으로 확인하는 로지스틱 회귀

```
# 기울기 a와 절편 b의 값을 초기화 합니다.
a = 0
b = 0
```

기울기 a와 y절편 b값 초기화

```
#학습률을 정합니다.
lr = 0.05
```

학습률=0.05

```
#시그모이드 함수를 정의합니다.
def sigmoid(x):
    return 1 / (1 + np.e ** (-x))
```

시그모이드 함수 정의

```
#경사 하강법을 실행합니다.
for i in range(2001):
    for x_data, y_data in data:
        a_diff = x_data*(sigmoid(a*x_data + b) - y_data)
        b_diff = sigmoid(a*x_data + b) - y_data
        a = a - lr * a_diff
        b = b - lr * b_diff
        if i % 1000 == 0:    # 1000번 반복될 때마다 각 x_data값에 대한 현재의 a값, b값을 출력합니다.
            print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))
```

경사하강법 실행
(epoch=2001)

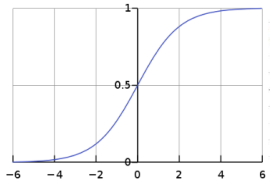
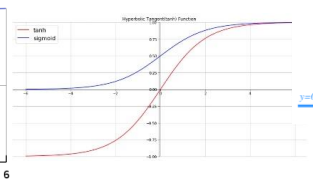
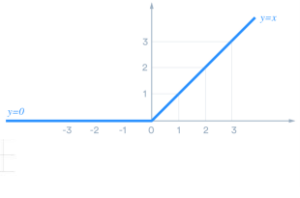
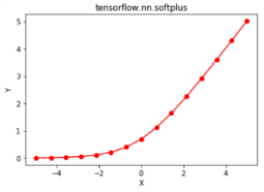
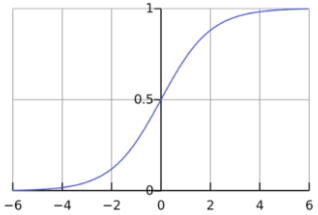
```
epoch=0, 기울기=-0.0500, 절편=-0.0250
epoch=0, 기울기=-0.1388, 절편=-0.0472
epoch=0, 기울기=-0.2268, 절편=-0.0619
(중략)
epoch=2000, 기울기=1.9068, 절편=-12.9513
epoch=2000, 기울기=1.9068, 절편=-12.9513
epoch=2000, 기울기=1.9068, 절편=-12.9513
```

경사하강법 실행 결과(학습률=0.05, epoch=2001)
기울기 a=1.9068, y절편 b=12.9513로 수렴하였다.

# 코딩으로 확인하는 로지스틱 회귀

- 시그모이드 형태의 함수가 잘 만들어지도록 a와 b의 값이 수렴된 것을 알 수 있음
- 만약 여기에 입력 값이 추가되어 세 개 이상의 입력 값을 다룬다면(다중 분류 문제) 시그모이드 함수가 아니라 소프트맥스(softmax) 라는 함수를 써야 함(12장 실습에서 다룰 예정)

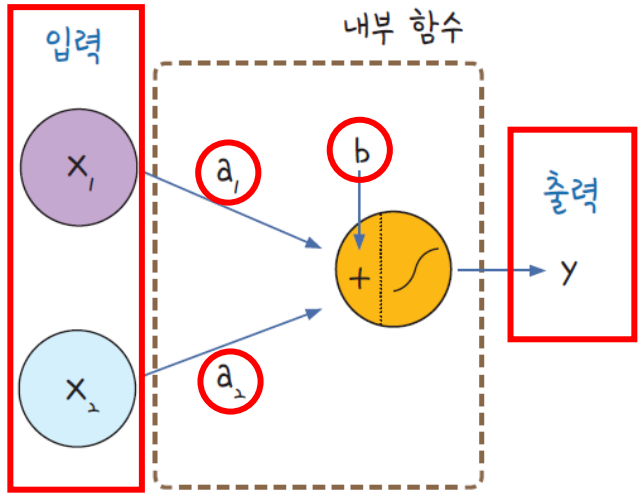
Sigmoid에서 시작된 활성화 함수는 ReLU를 비롯해 다양한 종류가 있다.

| 활성화 함수 | sigmoid | hyperbolic tangent | relu | softplus | softmax |
|---|---|---|---|---|---|
| |  |  |  |  |  |
| | $S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$ | $tanh(x) = \frac{1-e^{-x}}{1+e^{-x}}$ | $f(x) = max(0, x)$ | $f(x) = ln(1+e^x)$ | $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ |
| 설명 | 0 또는 1의 값이 출력됨 | 위 아래로 늘리기 | 0보다 작을 땐 0<br>0보다 클 땐 X | 0을 만드는 기준을 완화시키기 | 0과 1 사이 값이 여러 개 출력됨 |
| 출력값 | 0 or 1 | -1 ~ 1 | O or X | O or X | 0 ~ 1 |
| 사용처 | 이진 분류 출력층 | | Vanishing 해결<br>은닉층에 사용. | | 다중 분류 출력층 |

# 로지스틱 회귀에서 퍼셉트론으로

- 입력 값을 통해 출력 값을 구하는 함수 y는 다음과 같이 표현할 수 있음

$$y = a_1 x_1 + a_2 x_2 + b$$




- 입력 값 : 우리가 가진 값인 $x_1$, $x_2$
- 출력 값 : 계산으로 얻는 값 y

  -> 출력 값 y를 구하려면 가중치(weight) $a_1$값, $a_2$값

  그리고 편향(bias) b값이 필요함

**"x1과 x2가 입력되고, 각각 가중치 a1, a2를 만난다.**
**여기에 b값을 더한 후 시그모이드 함수를 거쳐 1 또는 0의 출력값 y를 출력한다."**

- 프랑크 로젠블라트가 퍼셉트론(1957년)이라는 이름을 붙임
- 이 퍼셉트론은 이 후 인공신경망, 오차 역전파 등의 발전을 거쳐 지금의 딥러닝으로 발전됨