

[2017 Winter]

# Tensorflow Tutorial

## 01. 기본 개념

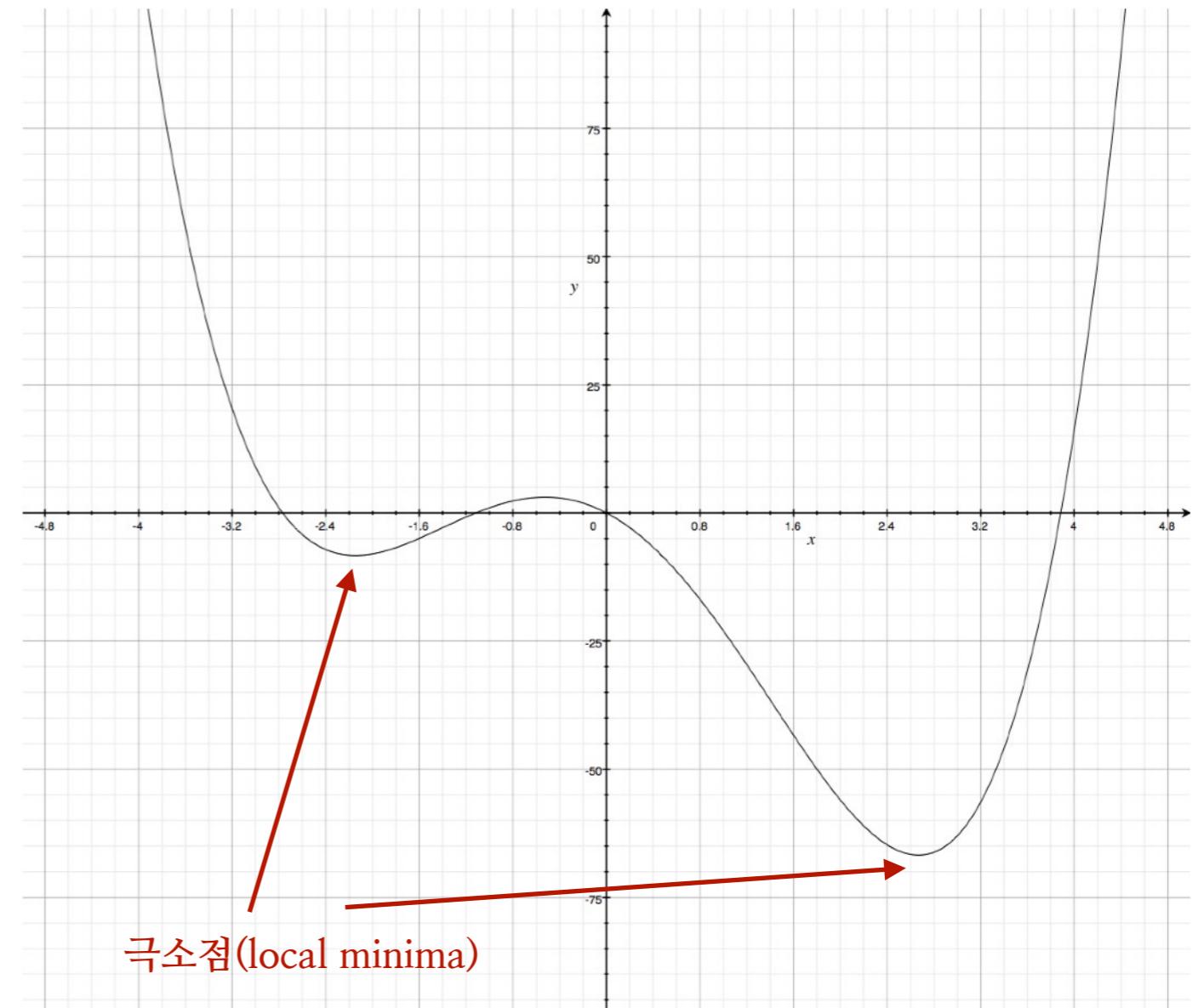
## 다음 함수의 최소값은?

$$f(x) = x^4 - 12x^2 - 12x$$

$$f'(x) = 4x^3 - 24x - 12 = 0$$

$x = ?$

미분계수가 0이 되는 x값을  
항상 대수적으로 구할수 있는 것은 아니다.



극소점(local minima)

# Gradient Descent Method

Let  $x_0 = 4$  (random choice)

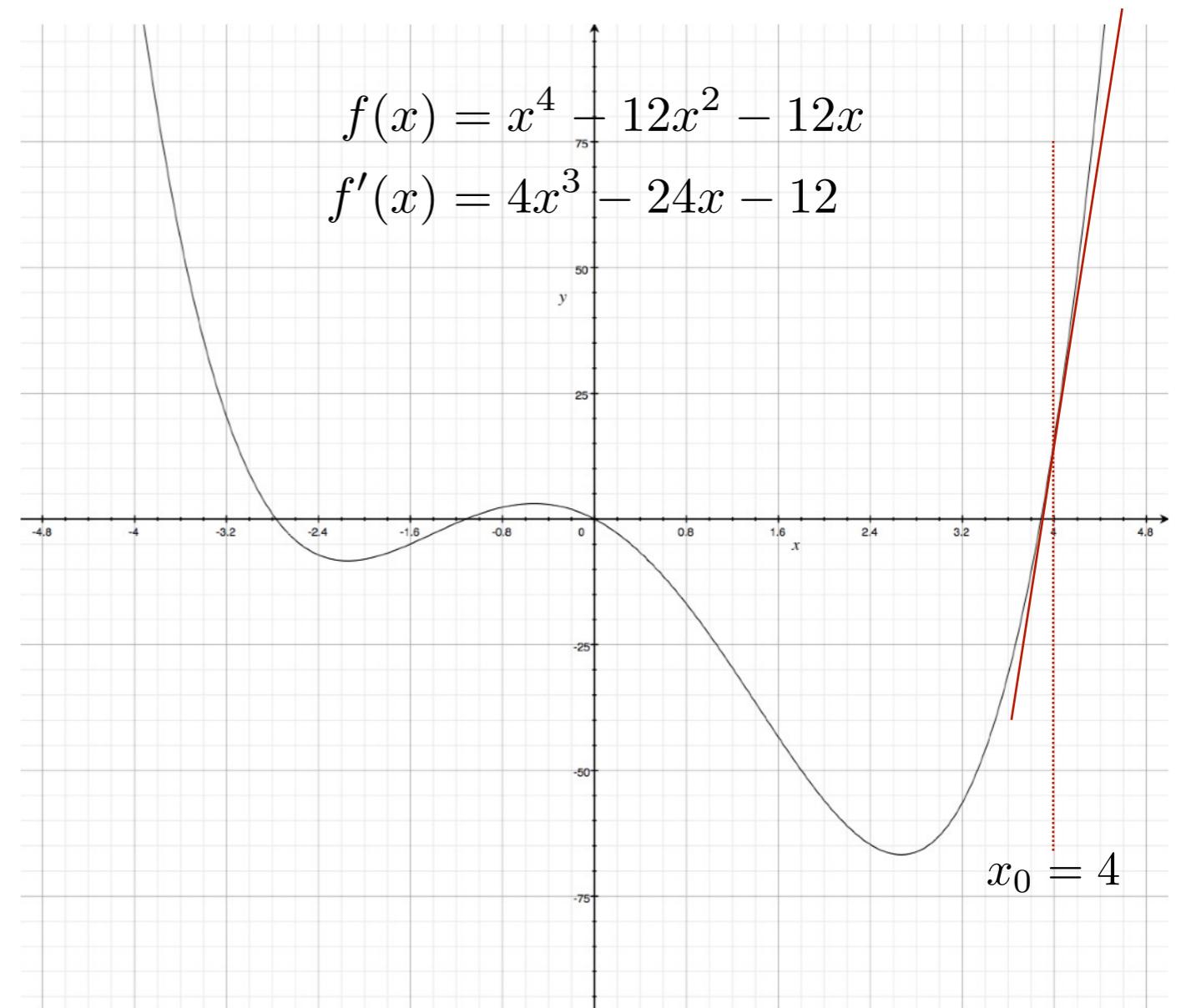
$$f'(4) = 4 \cdot 4^3 - 24 \cdot 4 - 12 = 148.0$$

함수  $f(x)$ 는  $x=4$ 에서 증가상태이다.  
따라서  $x=4$ 에서 왼쪽으로 이동하면  
함수값은 감소한다.  
따라서  $x=4$ 보다 왼쪽의 적절한 값을  
 $x_1$ 으로 선택한다.

$$x_1 = x_0 - \rho f'(x_0)$$



$\rho$ 는 적당한 상수(예를 들어 0.005), 기울기가 가파르면 많이 이동하고  
기울기가 완만하면(0에 가까우면) 극소점 근처일 가능성성이 있으므로 조금만 이동한다.



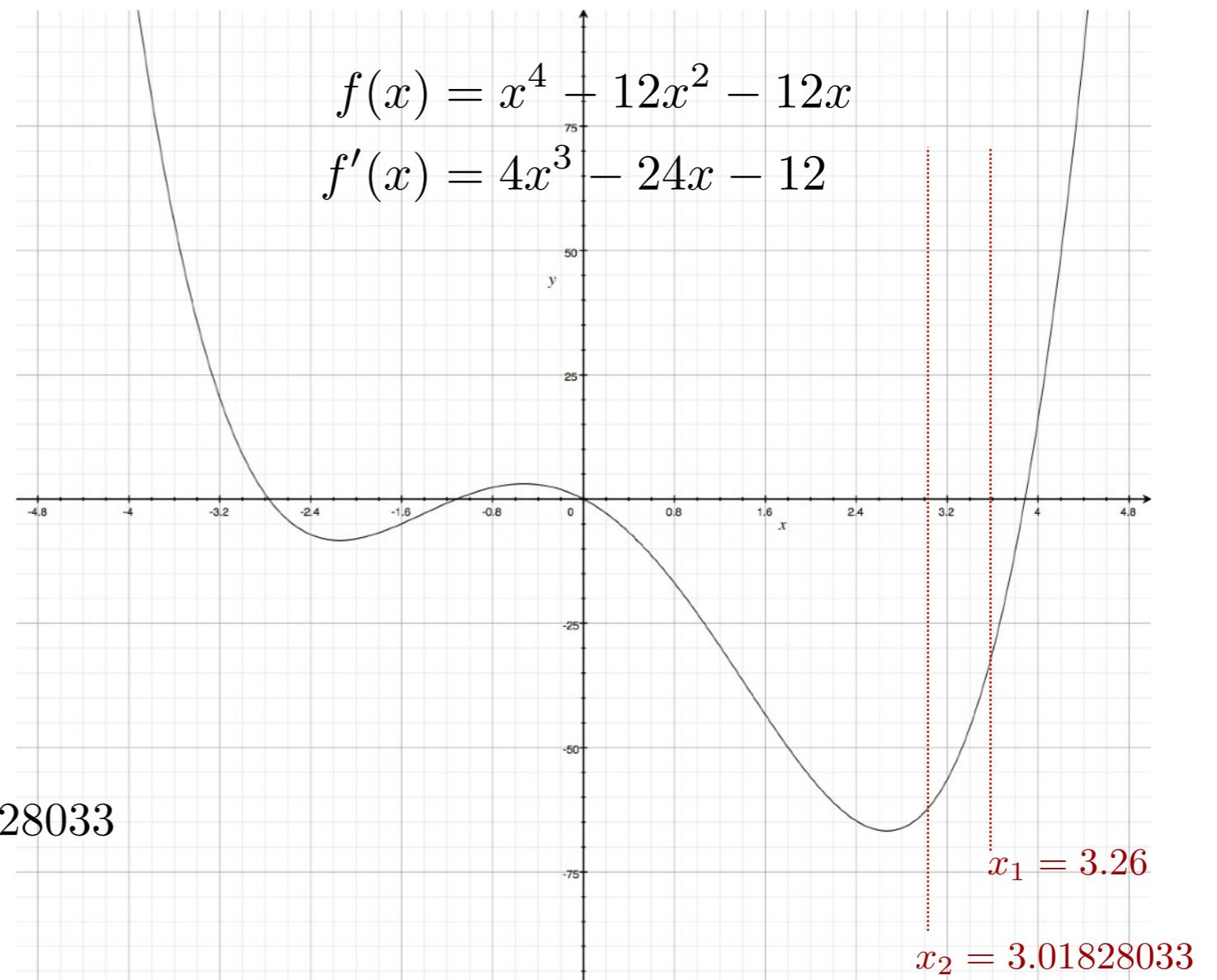
# Gradient Descent Method

$\rho$ 는 적절한 작은 상수

$$\begin{aligned}x_1 &= x_0 - \rho f'(x_0) \\&= 4.0 - 0.005 \cdot 148.0 = 3.26.\end{aligned}$$

$$f'(3.26) = 48.343904$$

$$\begin{aligned}x_2 &= x_1 - \rho f'(x_1) \\&= 3.26 - 0.005 \cdot 48.343904 = 3.01828033\end{aligned}$$



# Gradient Descent Method

```
from random import random
```

```
def f(x):
    return x**4 - 12.0*x**2 - 12.0*x

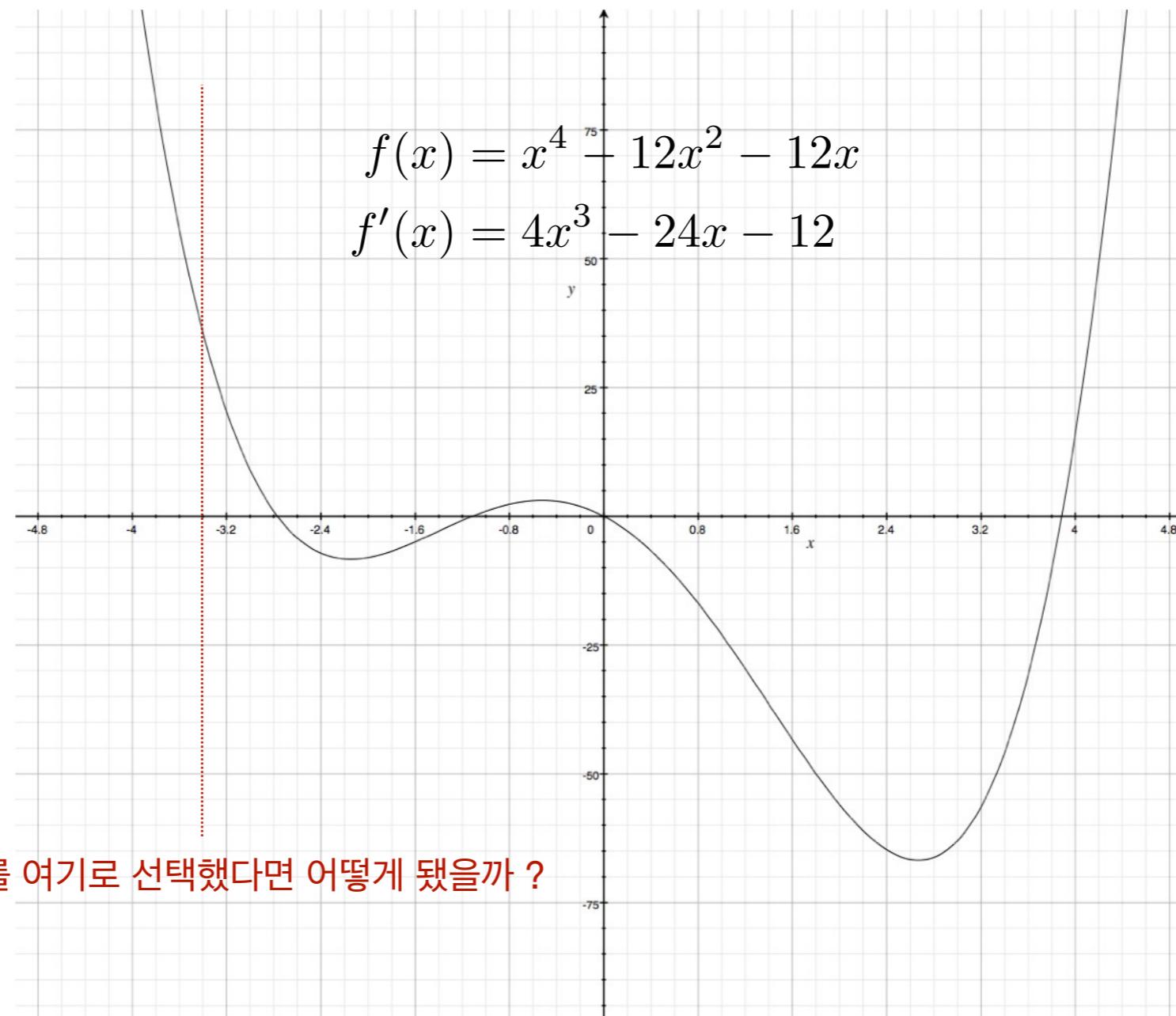
def df(x):
    return 4 * x**3 - 24 * x - 12

rho = 0.005
precision = 0.000000001
difference = 100
x = random()
# x = 4.0

while difference > precision:
    dr = df(x)
    prev_x = x
    x = x - rho * dr
    difference = abs(prev_x - x)
    print("x = {:.f}, df = {:.10.6f}, f(x) = {:.f}".format(x, dr, f(x)))
```

```
x = 0.507647, df = -21.362430, f(x) = -9.117818
x = 0.625948, df = -23.660235, f(x) = -12.059596
x = 0.756157, df = -26.041743, f(x) = -15.608237
x = 0.898249, df = -28.418364, f(x) = -19.810186
x = 1.051544, df = -30.658959, f(x) = -24.664778
x = 1.214474, df = -32.586094, f(x) = -30.097584
x = 1.384385, df = -33.982236, f(x) = -35.937832
x = 1.557447, df = -34.612424, f(x) = -41.913333
x = 1.728785, df = -34.267496, f(x) = -47.677470
x = 1.892903, df = -32.823581, f(x) = -52.873332
x = 2.044403, df = -30.299973, f(x) = -57.218908
x = 2.178836, df = -26.886671, f(x) = -60.576838
x = 2.293423, df = -22.917486, f(x) = -62.973170
x = 2.387376, df = -18.790455, f(x) = -64.558244
x = 2.461721, df = -14.869031, f(x) = -65.537041
...
...
```

# Gradient Descent Method



만약 처음에  $x_0$ 를 여기로 선택했다면 어떻게 됐을까 ?

**Gradient descent 알고리즘은 local minimum에 빠질 수 있다.**  
즉 항상 최소값을 찾는 것은 아니다.

- 다음과 같이 2개의 변수를 가지는 함수의 최소값은?

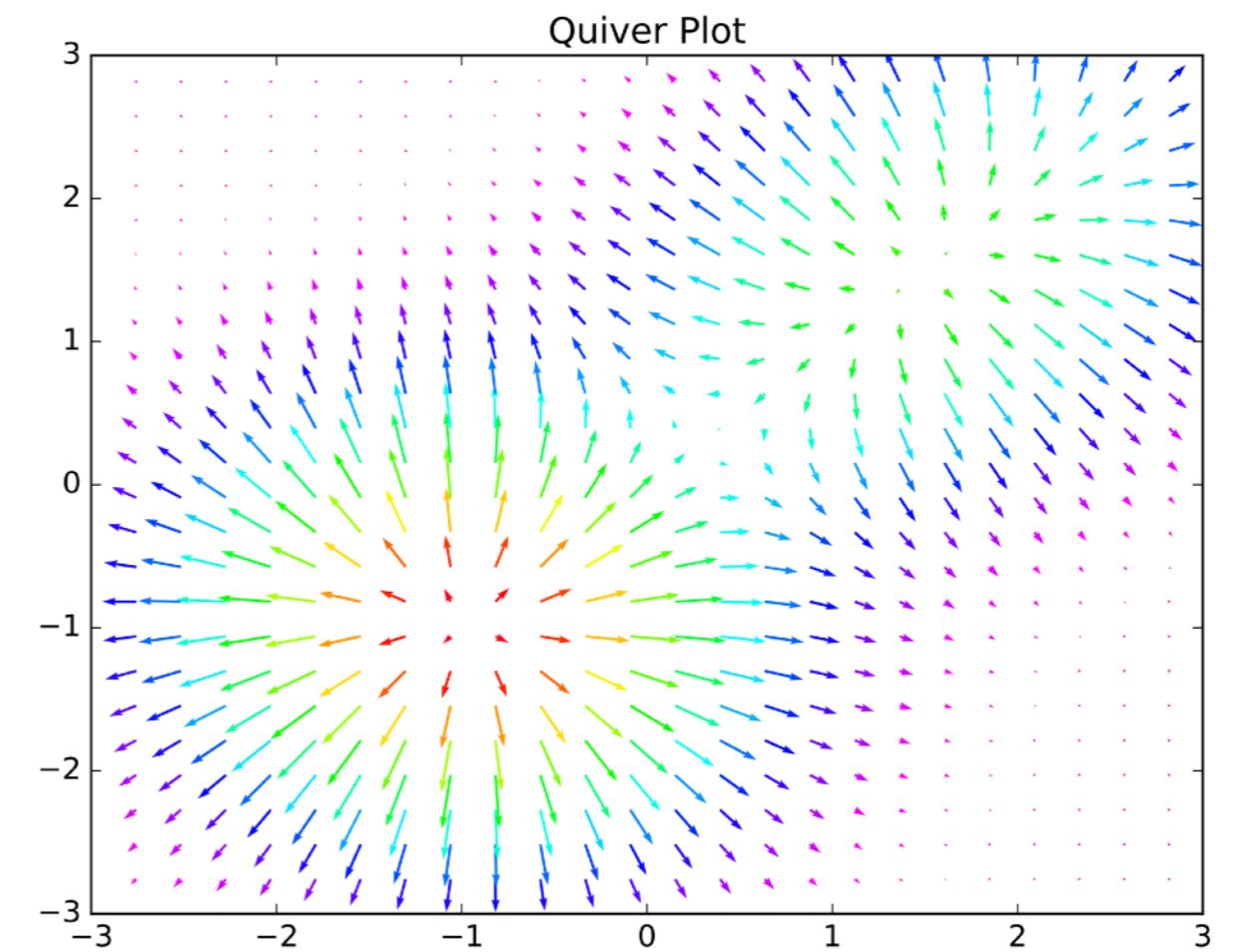
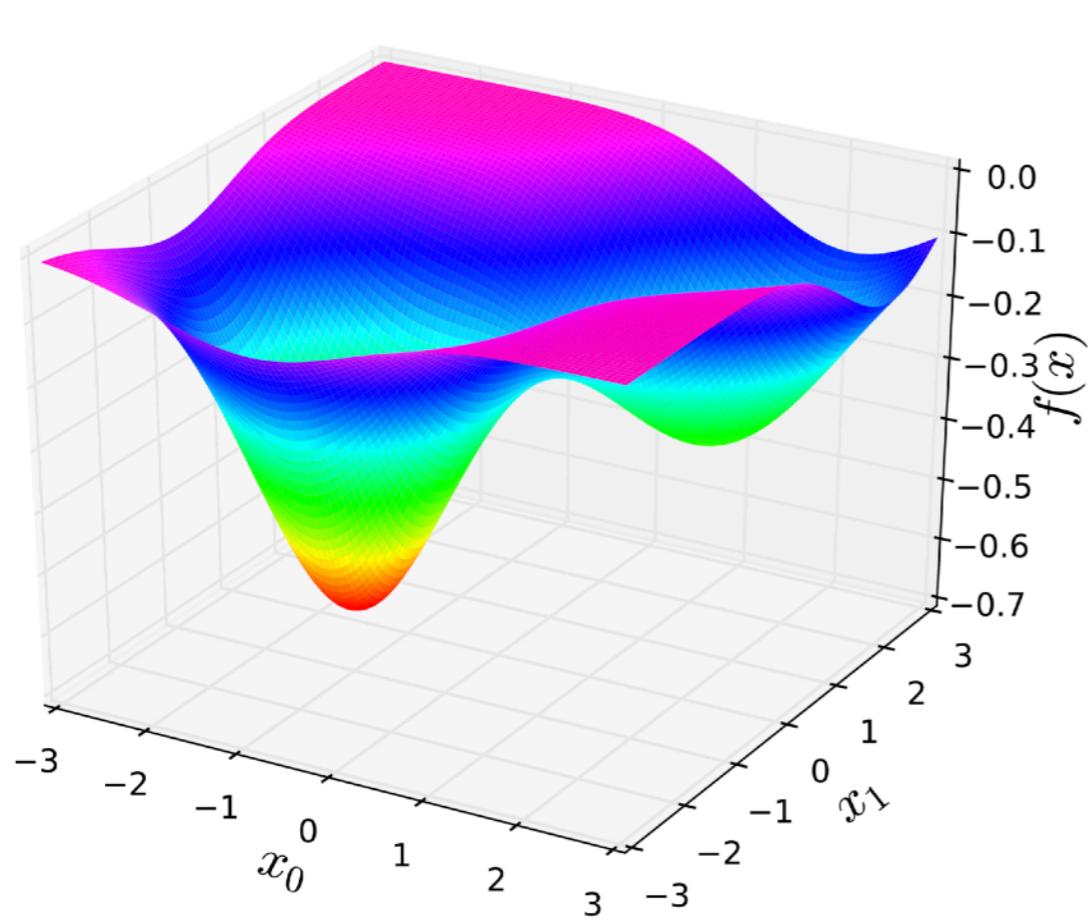
$$f(x, y) = 2x^2 + 2xy + y^2$$

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x, y) \\ \frac{\partial}{\partial y} f(x, y) \end{bmatrix} = \begin{bmatrix} 4x + 2y \\ 2x + 2y \end{bmatrix}$$

gradient

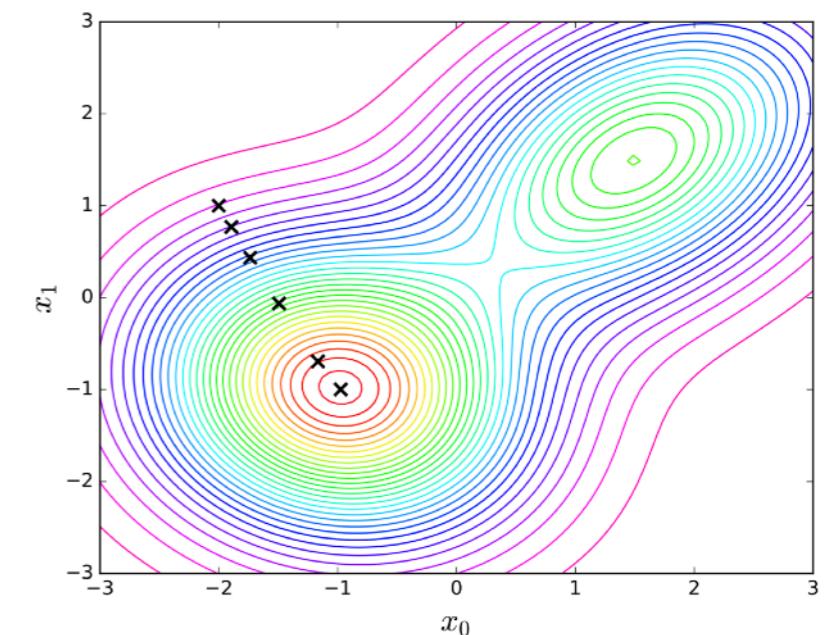
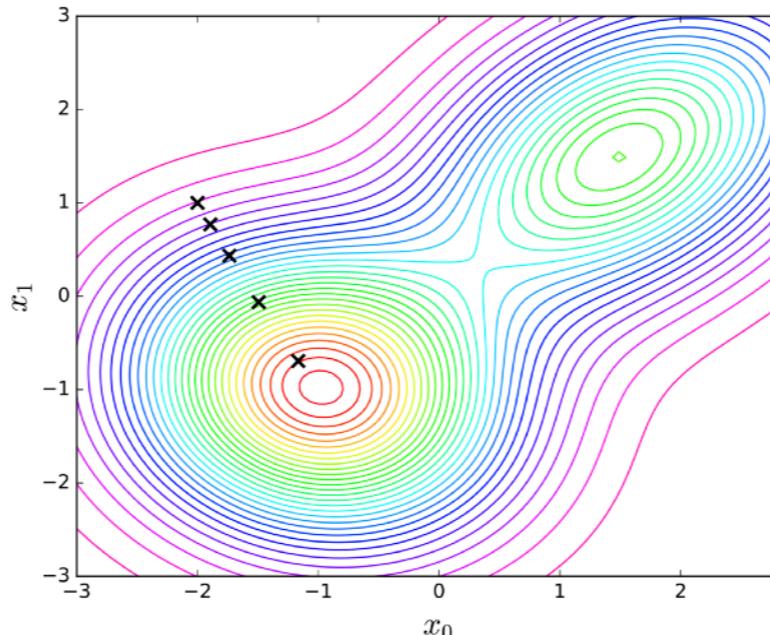
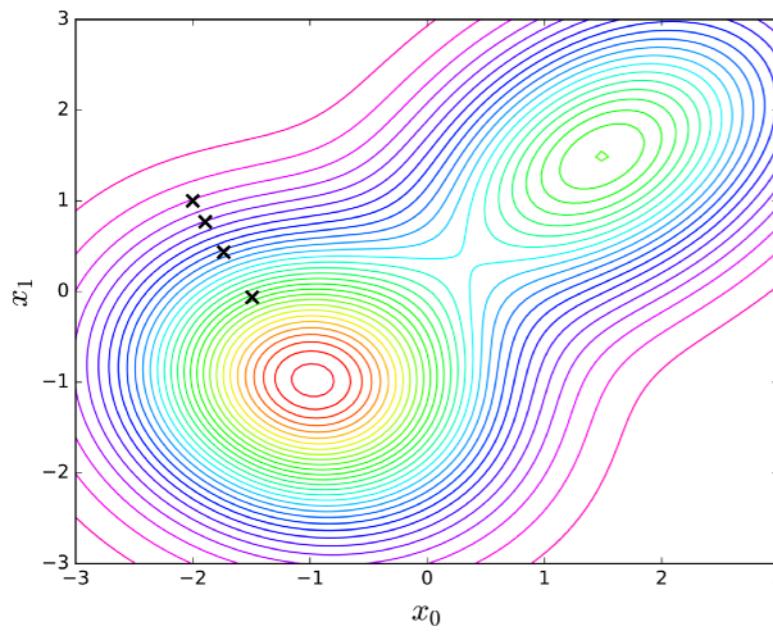
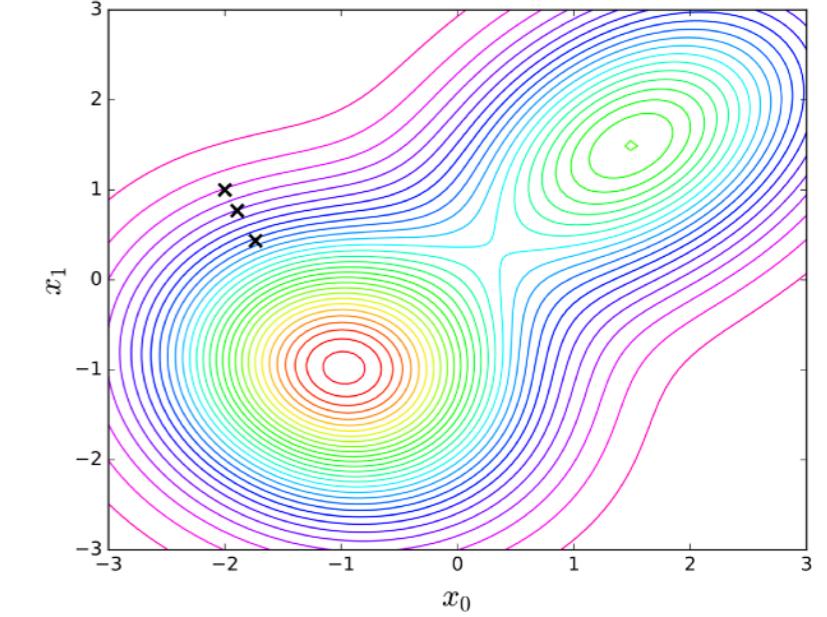
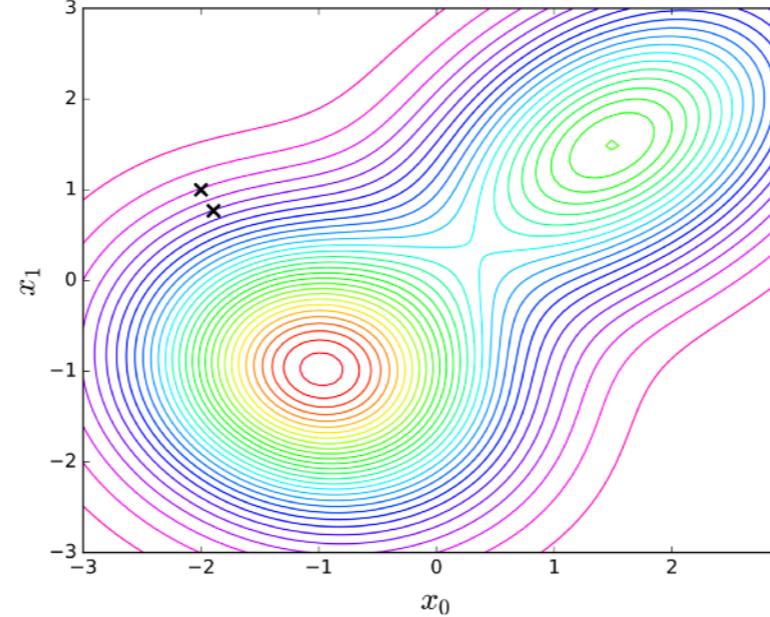
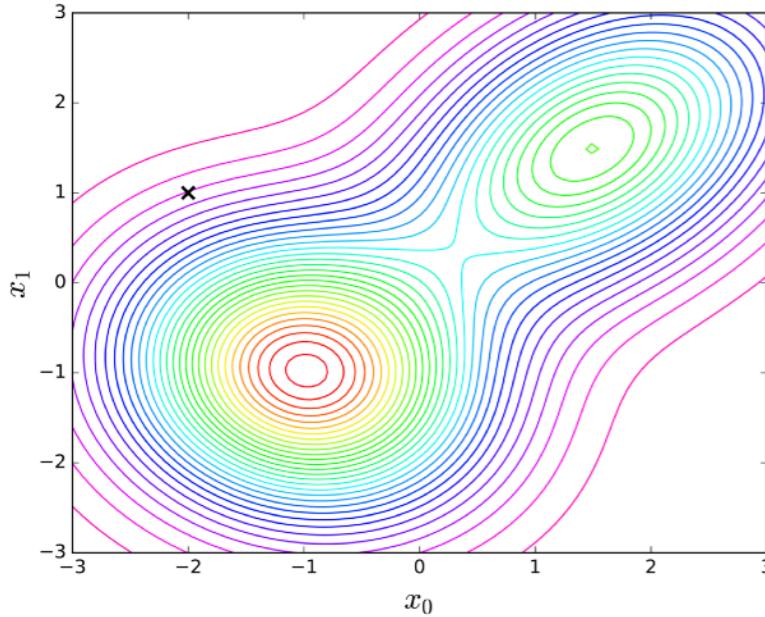
$\nabla f(x_0, y_0)$ 는  $(x_0, y_0)$ 에서 함수값이 가장 빠르게 증가하는 방향과 증가하는 정도를 나타내는 벡터이다.

# Gradient Descent Method



Gradient는 함수값이 가장 빠르게 증가하는 방향과  
증가하는 정도를 나타낸다.

# Gradient Descent Method



## Gradient Descent Method

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}})$$

Current value of  $\mathbf{x}$

Step size

Gradient

The diagram illustrates the gradient descent update rule. The equation  $\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}})$  is displayed. Three arrows point to the terms: one from "Current value of  $\mathbf{x}$ " to  $\mathbf{x}^{\text{old}}$ , one from "Step size" to  $\rho$ , and one from "Gradient" to  $\nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}})$ .

# Gradient Descent Method

```
import numpy as np

def f(x):
    return 2*x[0]**2 + 2*x[0]*x[1] + x[1]**2

def df(x):
    dx = 4*x[0] + 2*x[1]
    dy = 2*x[0] + 2*x[1]
    return np.array([dx, dy])

rho = 0.005
precision = 0.0000001
difference = 100
x = np.random.rand(2)

while difference > precision:
    dr = df(x)
    prev_x = x
    x = x - rho * dr
    difference = np.dot(x-prev_x, x-prev_x)
    print("x = {}, df = {}, f(x) = {:.f}"
          .format(np.array2string(x), np.array2string(dr), f(x)))
```

```
...
x = [-0.02640796  0.06073794], df = [ 0.01687681  0.06952397], f(x) = 0.001876
x = [-0.02648718  0.06039464], df = [ 0.01584404  0.06865996], f(x) = 0.001851
x = [-0.02656139  0.06005557], df = [ 0.01484056  0.06781492], f(x) = 0.001827
x = [-0.02663071  0.05972063], df = [ 0.0138656   0.06698837], f(x) = 0.001804
x = [-0.02669531  0.05938973], df = [ 0.0129184   0.06617983], f(x) = 0.001782
x = [-0.0267553   0.05906278], df = [ 0.01199824  0.06538885], f(x) = 0.001760
x = [-0.02681082  0.05873971], df = [ 0.01110438  0.06461498], f(x) = 0.001738
x = [-0.026862    0.05842042], df = [ 0.01023614  0.06385778], f(x) = 0.001718
x = [-0.02690896  0.05810484], df = [ 0.00939284  0.06311684], f(x) = 0.001697
x = [-0.02695183  0.05779288], df = [ 0.00857382  0.06239175], f(x) = 0.001678
```

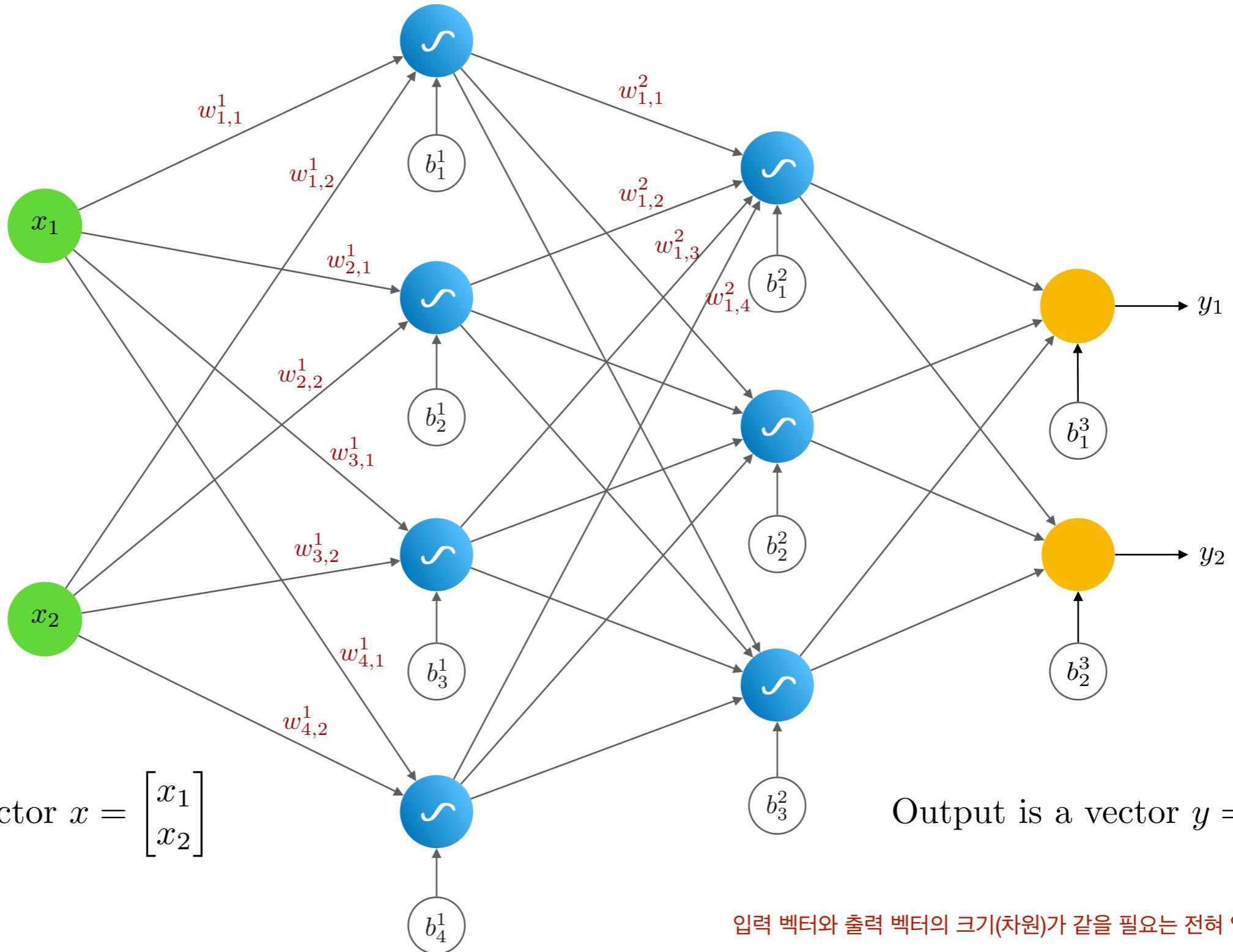
# What is Neural Network ?

Input Layer

Layer1

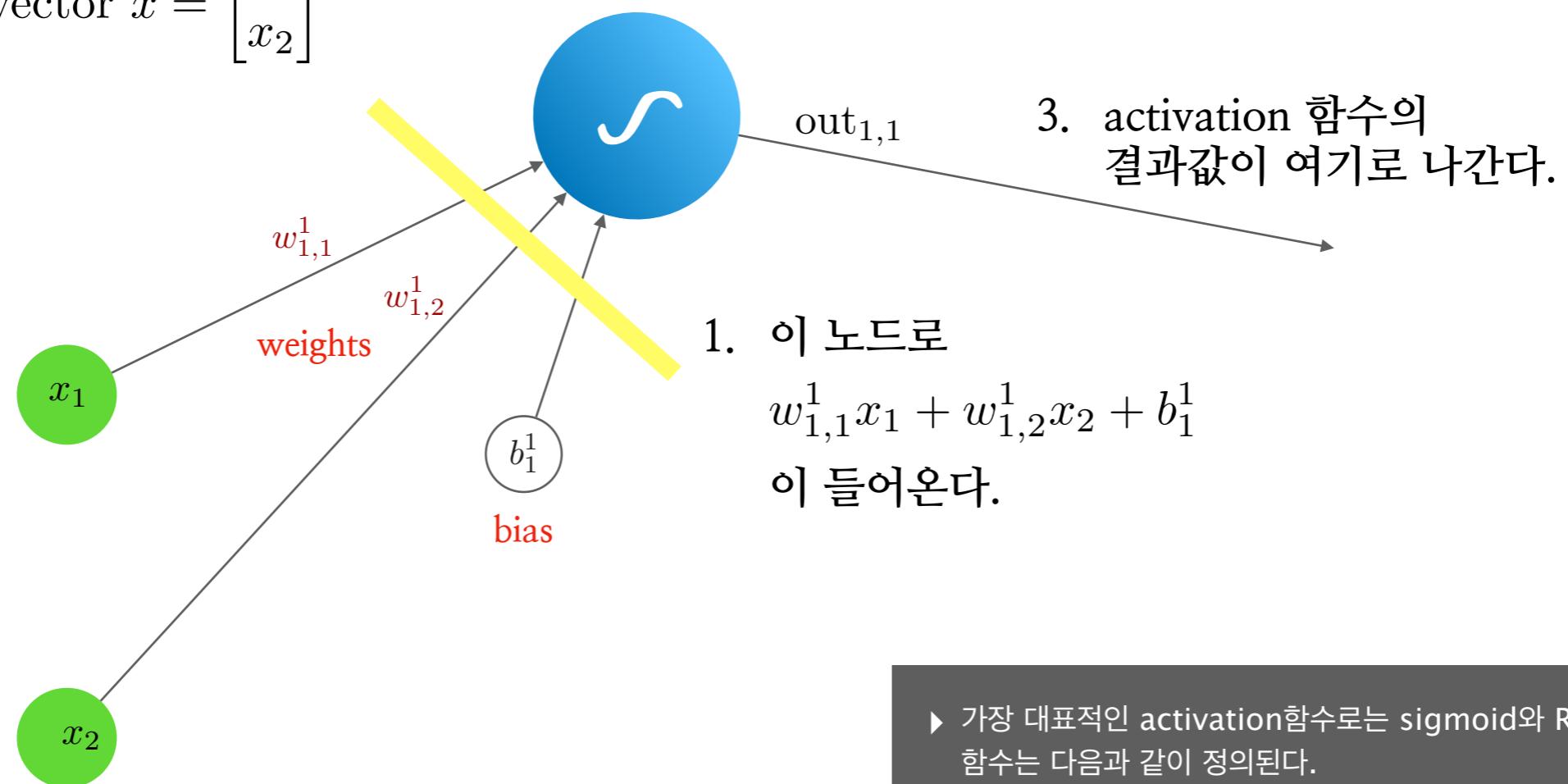
Layer2

Output Layer



# How does it work?

Input is a vector  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



▶ 가장 대표적인 activation 함수로는 sigmoid와 Relu가 있다. Sigmoid 함수는 다음과 같이 정의된다.

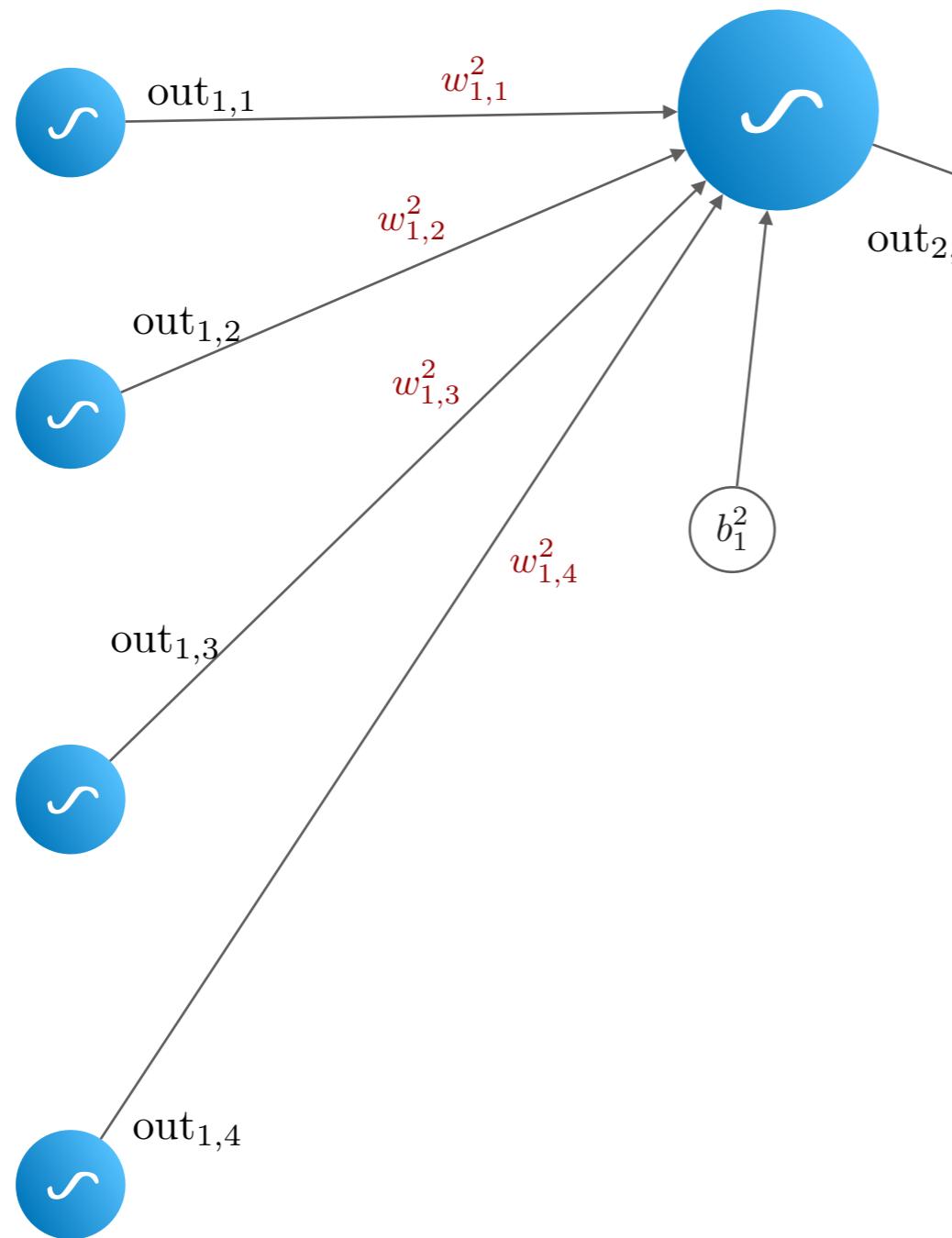
$$S(x) = \frac{1}{1 + e^{-x}}$$

▶ 따라서, 이 노드의 출력값은 다음과 같다.

$$\text{out}_{1,1} = \frac{1}{1 + e^{-(w_{1,1}^1 x_1 + w_{1,2}^1 x_2 + b_1^1)}}$$

# What is Neural Network ?

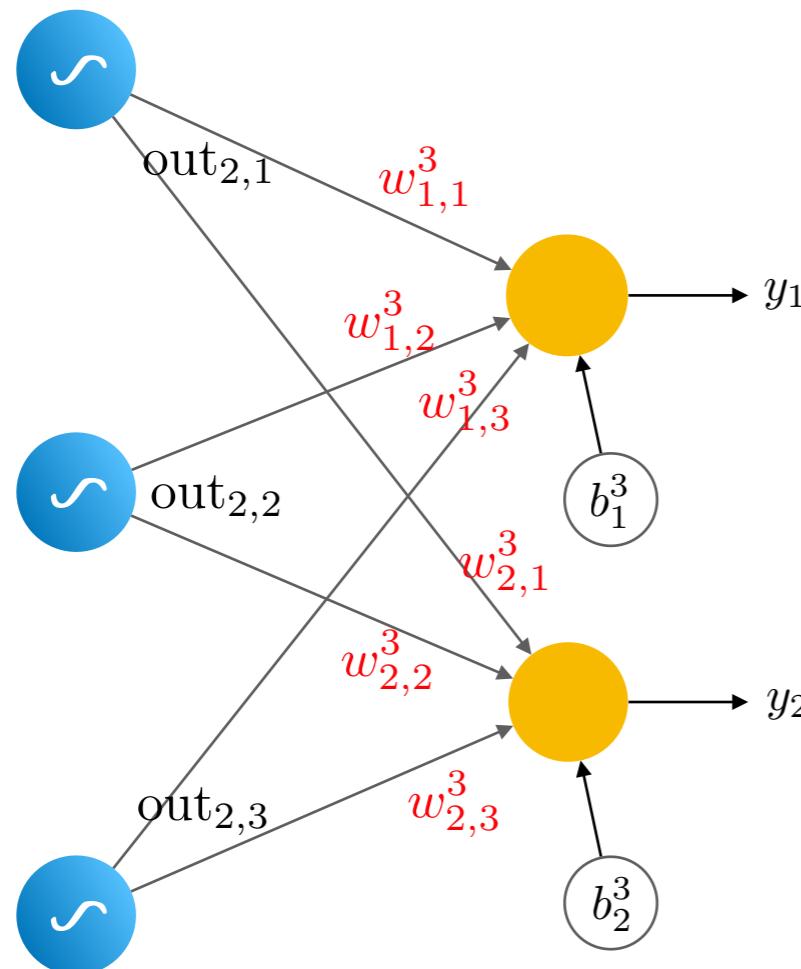
Layer1



Layer2

$$out_{2,1} = \frac{1}{1 + e^{-(w_{1,1}^2 out_{1,1} + w_{1,2}^2 out_{1,2} + w_{1,3}^2 out_{1,3} + w_{1,4}^2 out_{1,4} + b_1^2)}}$$

Layer2

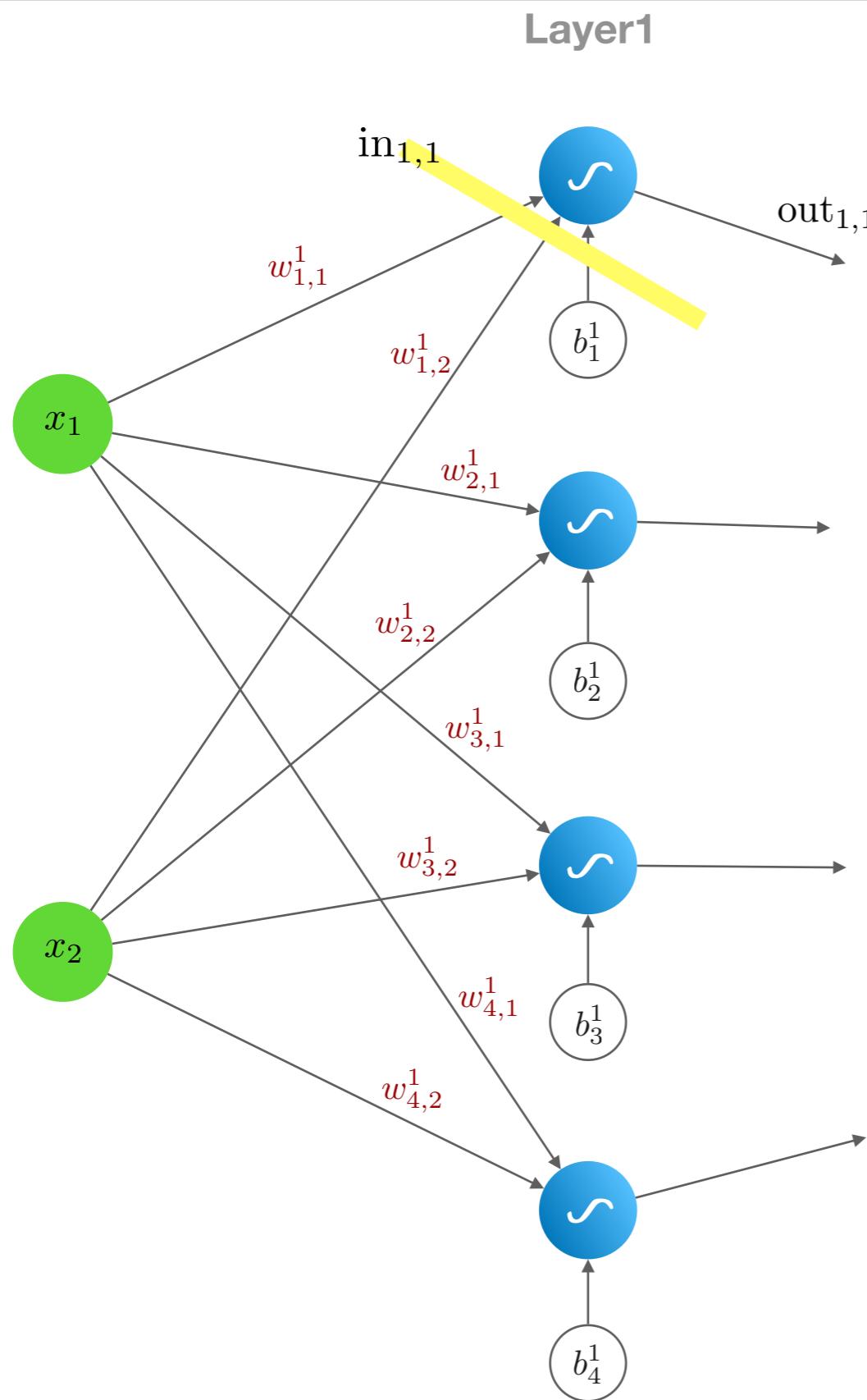


$$y_1 = w_{1,1}^3 \text{out}_{2,1} + w_{1,2}^3 \text{out}_{2,2} + w_{1,3}^3 \text{out}_{2,3} + b_1^3$$

$$y_2 = w_{2,1}^3 \text{out}_{2,1} + w_{2,2}^3 \text{out}_{2,2} + w_{2,3}^3 \text{out}_{2,3} + b_2^3$$

Output layer에 어떤 activation 함수를 사용할지는 문제의 성격에 따라 다르다.  
 (이 예에서는 linear function  $g(x) = x$ 를 사용한 것으로 하였다.)

# How does it work?



▶ Layer1의 각 노드에서 activation 함수를 적용하기 직전의 값을  $in_{1,1}, in_{1,2}, in_{1,3}, in_{1,4}$ 라고 부른다면 각각은 다음과 같다.

$$in_{1,1} = w_{1,1}^1 x_1 + w_{1,2}^1 x_2 + b_1^1$$

$$in_{1,2} = w_{2,1}^1 x_1 + w_{2,2}^1 x_2 + b_2^1$$

$$in_{1,3} = w_{3,1}^1 x_1 + w_{3,2}^1 x_2 + b_3^1$$

$$in_{1,4} = w_{4,1}^1 x_1 + w_{4,2}^1 x_2 + b_4^1$$

▶ 행렬  $W_1$ 과  $b_1$ 을 다음과 같이 정의한다.

$$W_1 = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 \\ w_{2,1}^1 & w_{2,2}^1 \\ w_{3,1}^1 & w_{3,2}^1 \\ w_{4,1}^1 & w_{4,2}^1 \end{bmatrix} \quad \text{“현재 층의 노드 개수} \times \text{이전 층의 노드 개수”}$$

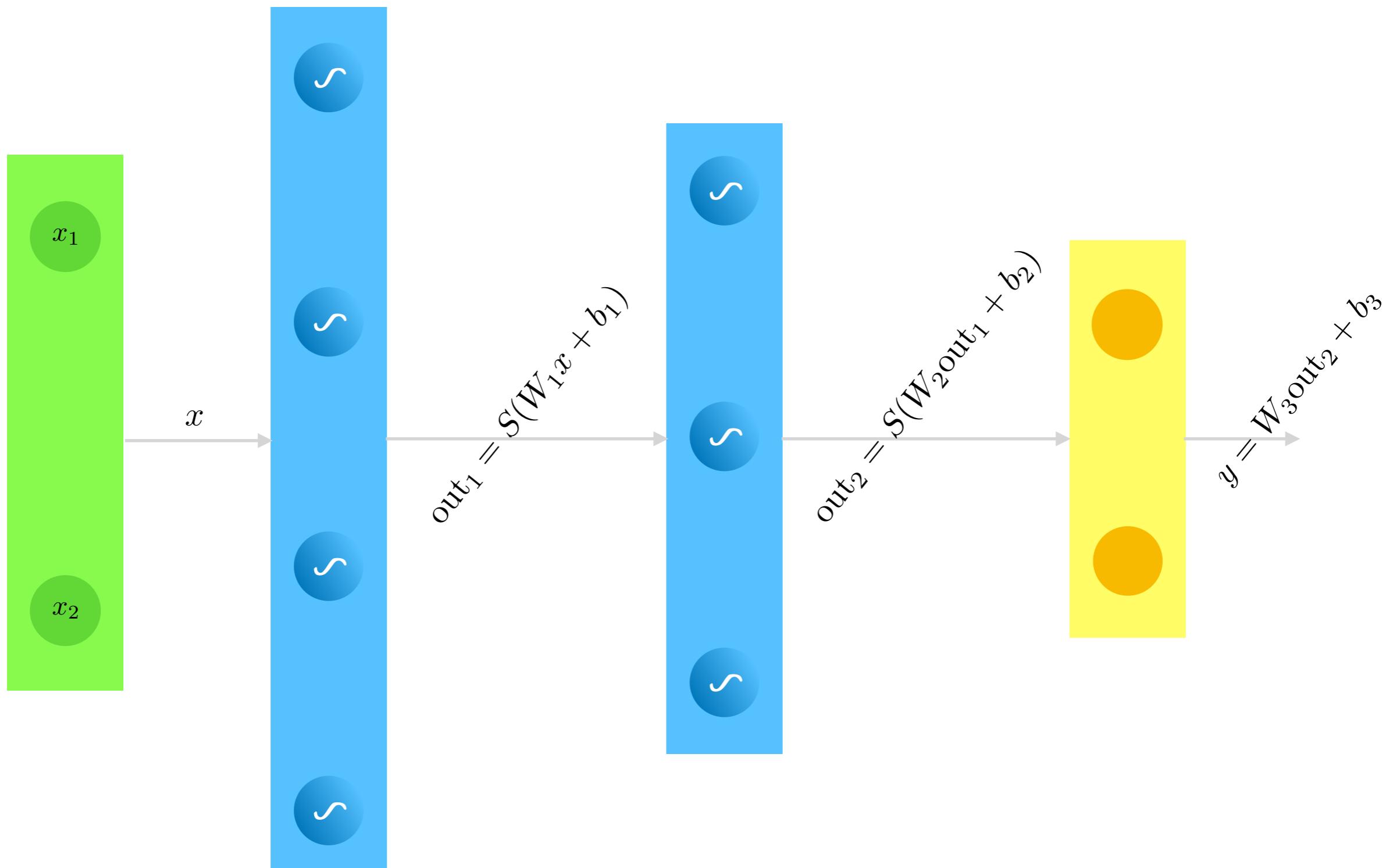
$$b_1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_4^1 \end{bmatrix}, \quad in_1 = \begin{bmatrix} in_{1,1} \\ in_{1,2} \\ in_{1,3} \\ in_{1,4} \end{bmatrix}$$

▶ 그러면 다음과 같은 행렬식으로 표현된다.

$$in_1 = W_1 x + b_1, \quad x = [x_1, x_2]^T$$

$$out_1 = S(in_1) \quad \text{← } S\text{는 activation 함수}$$

# How does it work?



Fully connected neural network은 잘 정리된 행렬식으로 표현 가능하다.

$$y = W_3 \text{out}_2 + b_3$$

$$\text{out}_2 = S(W_2 \text{out}_1 + b_2)$$

$$\text{out}_1 = S(W_1 x + b_2)$$

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

x가 벡터일 때

$$S([x_0, x_1]) = \left[ \frac{1}{1 + e^{-x_0}}, \frac{1}{1 + e^{-x_1}} \right]$$

$$y = W_3 \cdot \frac{1}{1 + e^{-(W_2 \cdot \frac{1}{1 + e^{-(W_1 x + b_2)}} + b_2)}} + b_3$$

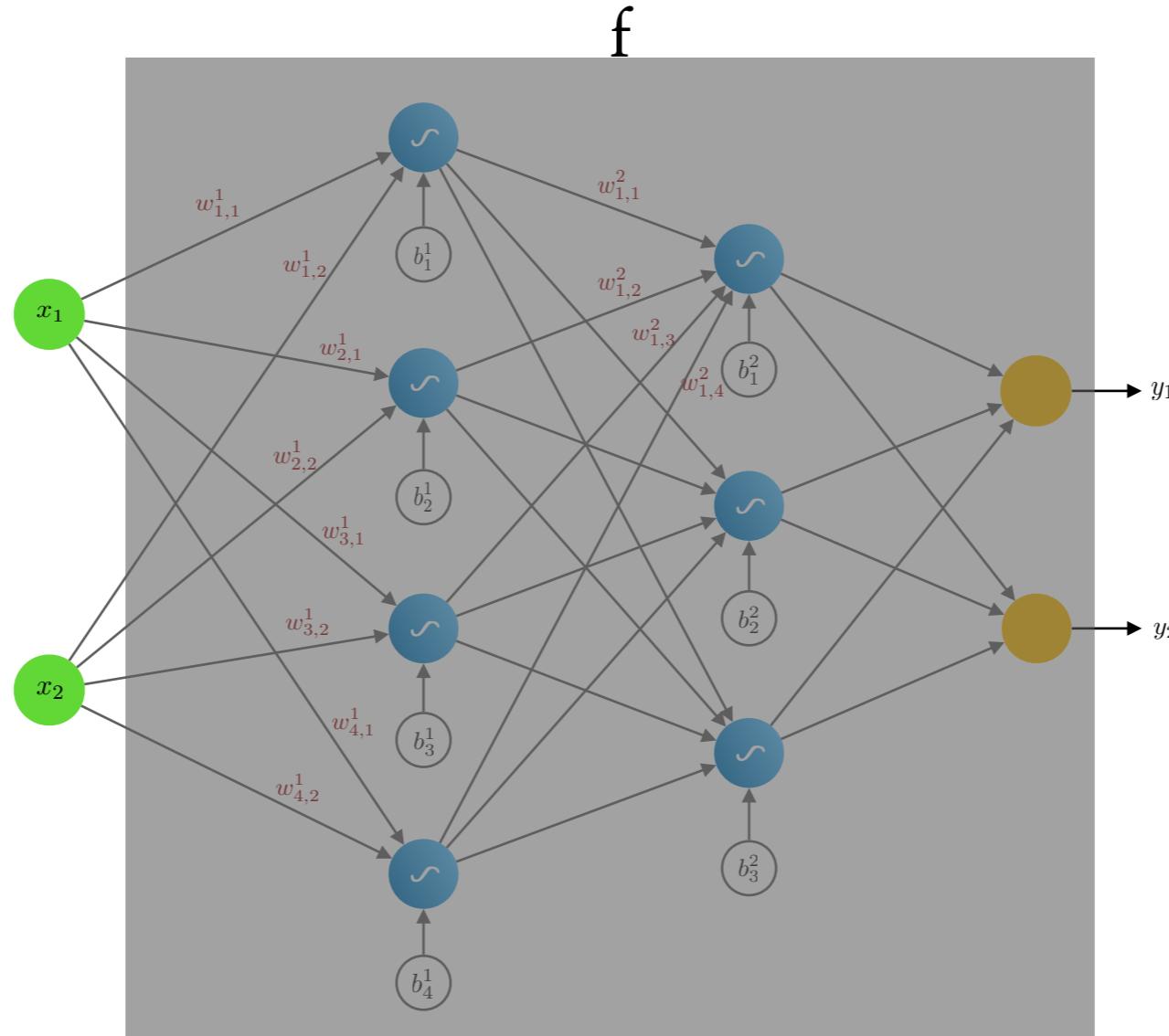
**“Neural network은 미분하기 쉬운 잘 정의된 함수일 뿐이다.”**

- 신경망은 기본적으로 지도학습(supervised learning)이다. 즉 정답을 알고 있는 학습 데이터를 사용하여 신경망을 학습시킨다.
- 학습(training) 데이터는 데이터(feature, 우리의 경우 동물 이미지)와 정답(label, 우리의 경우 무슨 동물인지)으로 구성

Training Data Set		
data	feature	label
X <sub>1</sub>	x <sub>1</sub> =2, x <sub>2</sub> =3	y <sub>1</sub> =1.0, y <sub>2</sub> =0.0
X <sub>2</sub>	x <sub>1</sub> =5, x <sub>2</sub> =8	y <sub>1</sub> =0.5, y <sub>2</sub> =0.0
X <sub>3</sub>	x <sub>1</sub> =1, x <sub>2</sub> =9	y <sub>1</sub> =0.0, y <sub>2</sub> =0.5
...		
X <sub>N</sub>	x <sub>1</sub> =3, x <sub>2</sub> =0	y <sub>1</sub> =0.0, y <sub>2</sub> =1.0

Test Data Set		
data	feature	label
X <sub>1</sub>	x <sub>1</sub> =5, x <sub>2</sub> =1	?
X <sub>2</sub>	x <sub>1</sub> =2, x <sub>2</sub> =8	?
X <sub>3</sub>	x <sub>1</sub> =2, x <sub>2</sub> =3	?



feature	label (desired output)	(actual) output	error
$X_1 = (2, 3)$	$\bar{Y}_1 = (1.0, 0.0)$	$Y_1 = f(X_1, W, b)$	$\bar{Y}_1 - Y_1$
$X_2 = (5, 8)$	$\bar{Y}_2 = (0.5, 0.0)$	$Y_2 = f(X_2, W, b)$	$\bar{Y}_2 - Y_2$
$X_3 = (1, 9)$	$\bar{Y}_3 = (0.0, 0.5)$	$Y_3 = f(X_3, W, b)$	$\bar{Y}_3 - Y_3$
...		...	
$X_N = (3, 0)$	$\bar{Y}_N = (0.0, 1.0)$	$Y_N = f(X_N, W, b)$	$\bar{Y}_N - Y_N$

여기서  $W$ 와  $b$ 는 각각 모든 weight들과 bias들의 합집합을 나타낸다.

feature	label (desired output)	(actual) output	error
$X_1 = (2, 3)$	$\bar{Y}_1 = (1.0, 0.0)$	$Y_1 = f(X_1, W, b)$	$\bar{Y}_1 - Y_1$
$X_2 = (5, 8)$	$\bar{Y}_2 = (0.5, 0.0)$	$Y_2 = f(X_2, W, b)$	$\bar{Y}_2 - Y_2$
$X_3 = (1, 9)$	$\bar{Y}_3 = (0.0, 0.5)$	$Y_3 = f(X_3, W, b)$	$\bar{Y}_3 - Y_3$
...		...	
$X_N = (3, 0)$	$\bar{Y}_N = (0.0, 1.0)$	$Y_N = f(X_N, W, b)$	$\bar{Y}_N - Y_N$

## >Total error (혹은 loss)

$$E_{X, \bar{Y}}(W, b) = \sum_{i=1}^N \|\bar{Y}_i - Y_i\|^2 \quad \leftarrow \begin{array}{l} \text{모든 학습 데이터에 대한 에러의 squared sum,} \\ \text{여기서 중요한 점은 이것이 “W와 b의 함수”라는 점} \\ (\text{왜냐하면 } X\text{들과 } \bar{Y}\text{들은 입력으로 주어진 상수값들이기 때문}) \end{array}$$

“Gradient descent 알고리즘을 이용하여  $E_{X, \bar{Y}}(W, b)$ 를 최소로하는  $W$ 와  $b$ 를 찾아라 !!!”

## >Error(loss) 함수와 그것의 gradient:

$$E_{X, \bar{Y}}(W, b) = \sum_{i=1}^N \|\bar{Y}_i - Y_i\|^2$$

$$\nabla E_{X, \bar{Y}}(W, b) = ?$$

**Gradient descent 알고리즘을 적용하기 위해서는  
error함수의 gradient를 구해야 함**

하지만 이걸 어떻게 구할지 걱정할 필요는 없음.  
똑똑한 사람들이 이미 구해 놓았음.  
그리고 별로 어렵지도 않음

W와 b의 값을 random하게 지정한다.  
각각을  $W_0$ 와  $b_0$ 라고 하자.

$Y_i, i = 1, \dots, N$ ,를 계산한다.

$\nabla E_{X, \bar{Y}}(W_0, b_0)$ 를 계산한다.

W와 b를 다음과 같이 갱신한다.

$$W_1, b_1 = W_0, b_0 - \rho \nabla E_{X, \bar{Y}}(W_0, b_0)$$

$Y_i, i = 1, \dots, N$ ,를 다시 계산한다.

$\nabla E_{X, \bar{Y}}(W_1, b_1)$ 를 계산한다.

W와 b를 다음과 같이 갱신한다.

$$W_2, b_2 = W_1, b_1 - \rho \nabla E_{X, \bar{Y}}(W_1, b_1)$$

...

# SGD: Stochastic Gradient Descent

- Loss는 모든 학습 데이터에 대한 오류의 합이다.

$$E_{X, \bar{Y}}(W, b) = \sum_{i=1}^N \|\bar{Y}_i - Y_i\|^2$$

- Training 데이터가 많을 경우 계산량이 많다.

- Training 데이터를 일정한 크기의 여러 집합으로 랜덤하게 분할하여 각각을 **mini-batch**라고 부른다.

- 각 round마다 서로 다른 mini-batch를 돌아가면서 사용한다.
- 모든 training 데이터가 한 번씩 사용되었을 때 한 **시대(epoch)**가 지났다고 말하고, 보통 학습은 여러 시대를 거쳐서 이루어진다.

W와 b의 값을 random하게 지정한다.  
각각을  $W_0$ 와  $b_0$ 라고 하자.

$Y_i, i = 1, \dots, N$ ,를 계산한다.

$\nabla E_{X, \bar{Y}}(W_0, b_0)$ 를 계산한다.

W와 b를 다음과 같이 갱신한다.

$$W_1, b_1 = W_0, b_0 - \rho \nabla E_{X, \bar{Y}}(W_0, b_0)$$

$Y_i, i = 1, \dots, N$ ,를 다시 계산한다.

$\nabla E_{X, \bar{Y}}(W_1, b_1)$ 를 계산한다.

W와 b를 다음과 같이 갱신한다.

$$W_2, b_2 = W_1, b_1 - \rho \nabla E_{X, \bar{Y}}(W_1, b_1)$$

첫 번째  
배치를 사용

두 번째  
배치를 사용

...