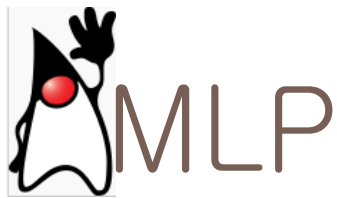


6장 MLP(다층 퍼셉트론)



- 퍼셉트론 : 하나의 뉴런만을 사용, 하나의 레이어로만 구성. XOR 처럼 선형분리 불가능한 문제를 해결 못함
-> 입력층과 출력층 사이에 은닉층(hidden layer)을 가지는 퍼셉트론으로 해결 : 다층 퍼셉트론(multilayer perceptron: MLP)

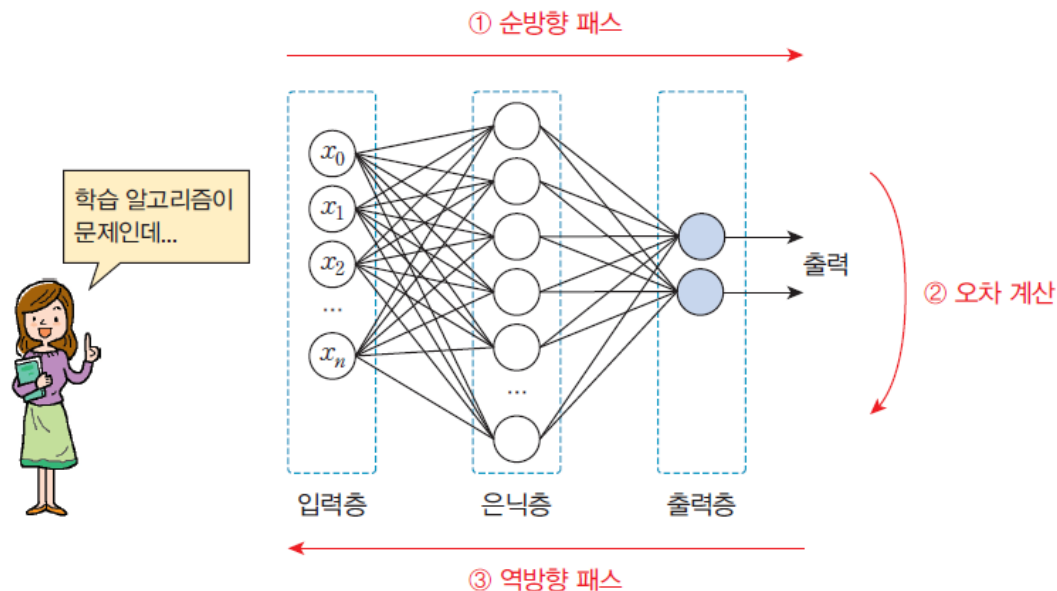


그림 6-1 MLP의 구조

- 활성화 함수(activation function) : 입력의 총합을 받아서 출력값을 계산하는 함수
 - 퍼셉트론 : 계단함수(step function) 사용
 - MLP : 다양한 활성화 함수(Sigmoid, TanH, ReLU)를 사용

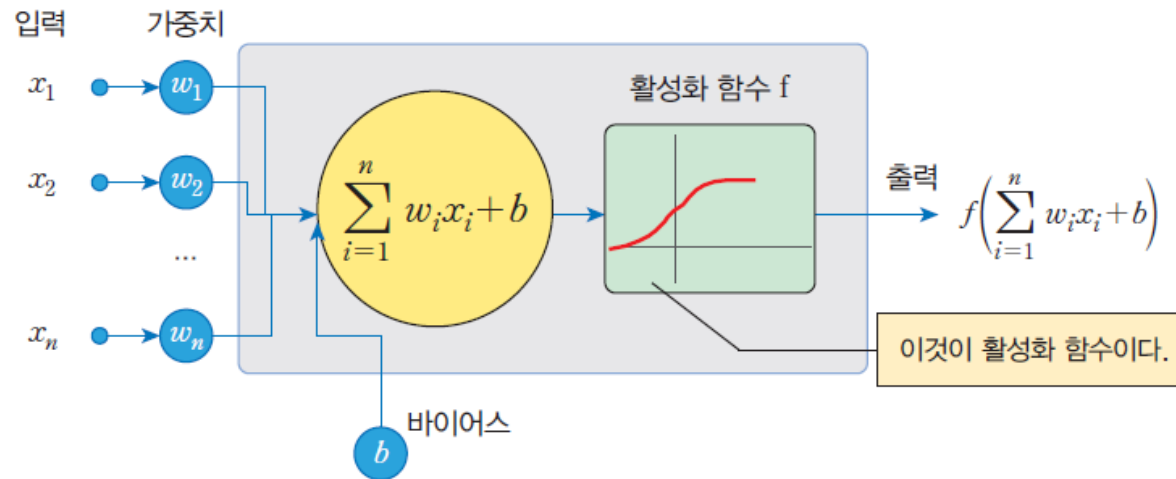


그림 6-2 활성화 함수



많이 사용되는 활성화 함수

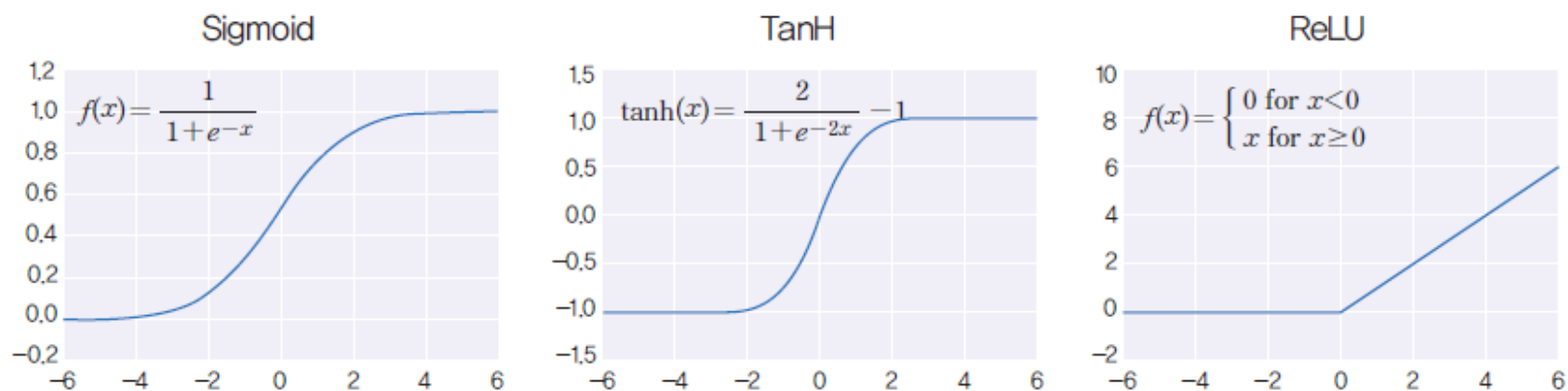


그림 6-3 많이 사용되는 활성화 함수



선형 레이어는 많아도 쓸모가 없다.

- MLP의 활성화 함수는 비선형 함수
- 은닉층을 아무리 많이 두어도 활성화 함수로 선형 함수를 사용하면 성능에 전혀 향상되지 않는다. 여러 개의 선형 함수를 결합해도 결국은 선형 함수 하나와 같이 때문이다.

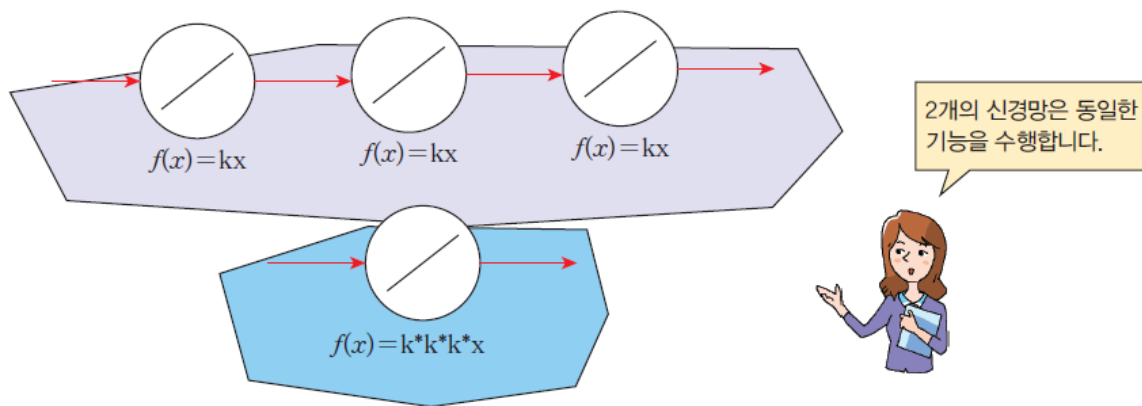


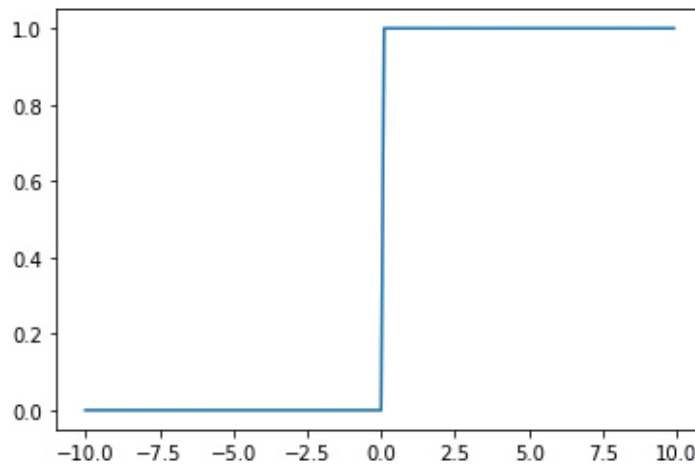
그림 6-4 선형 레이어는 아무리 많아도 하나의 레이어로 대체될 수 있다.



계단 함수(step function)

- 입력 신호의 총합이 0을 넘으면 1을 출력하고, 그렇지 않으면 0을 출력
- $x=0$ 에서 급격히 변화.

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

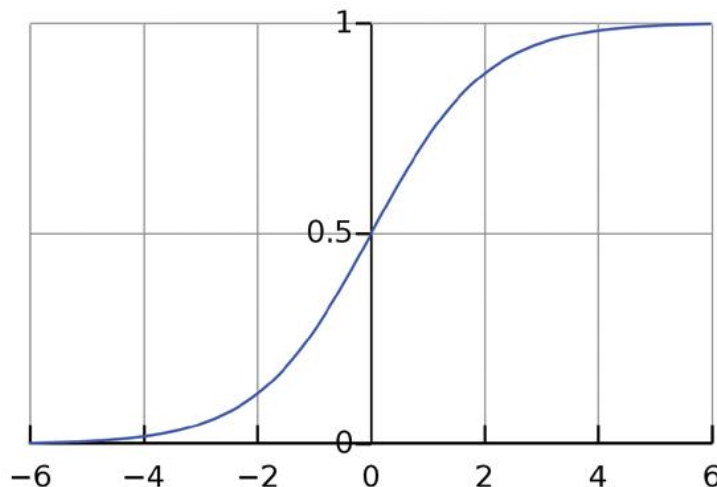




시그모이드 함수(sigmoid function)

- 1980년대부터 사용돼온 전통적인 활성화 함수
- S자 형태. 0.0에서 1.0까지의 연속적인 실수 출력
- 미끄럽게 변화하기 때문에 경사하강법에 필요한 미분이 가능

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

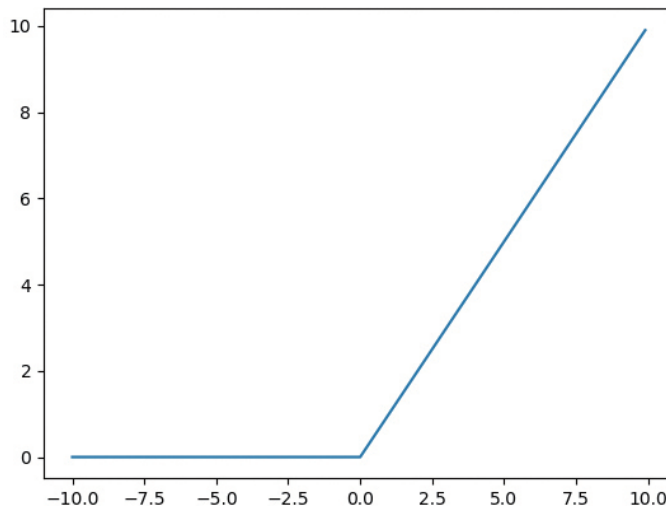




ReLU 함수 (Rectified Linear Unit function)

- 최근에 가장 인기 있는 활성화 함수
- 입력이 0을 넘으면 그대로 출력하고, 입력이 0보다 적으면 0을 출력
- 미분도 가능하고 그래디언트 감소가 일어나지 않아 많이 사용

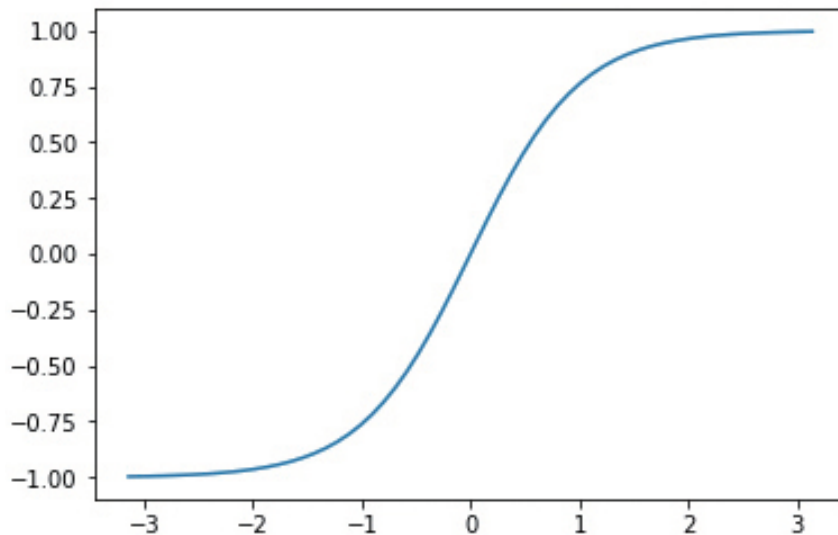
$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} = \text{np.maximum}(x, 0)$$



tanh 함수

- 넘파이에서 제공. 순환신경망(RNN)에서 많이 사용
- 시그모이드 함수와 비슷하지만 출력값이 -1에서 1까지.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$





Lab: 파이썬으로 활성화 함수 구현하기



MLP의 순방향 패스

- MLP = 순방향 패스 + 오차 계산 + 역방향 패스
- 순방향 패스 : 입력 신호가 입력층 유닛에 가해지고 이들 입력 신호가 은닉층을 통하여 출력층으로 전파되는 과정

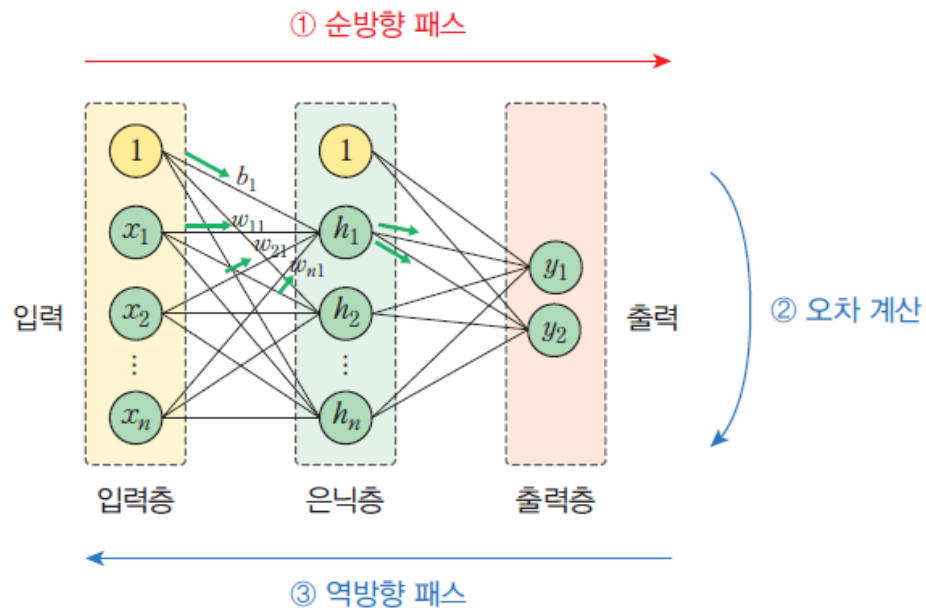
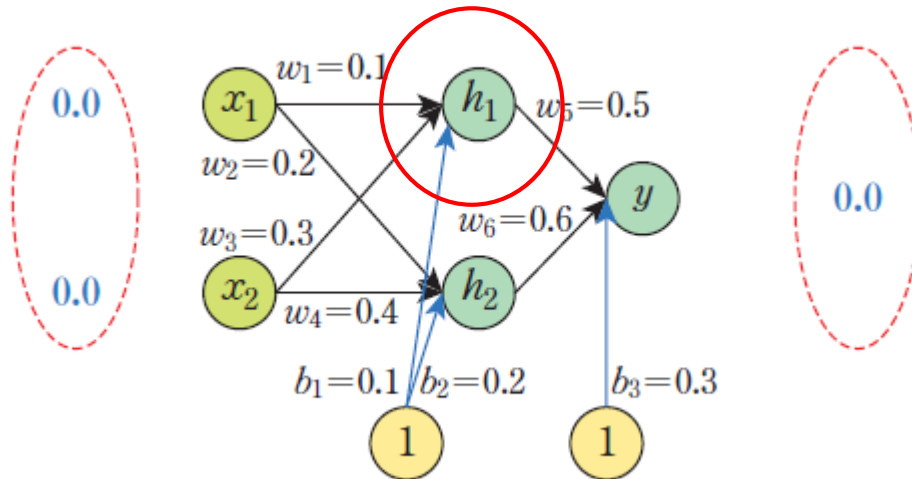


그림 6-5 순방향 패스



순으로 계산해보자.

- H_1 의 출력값 계산



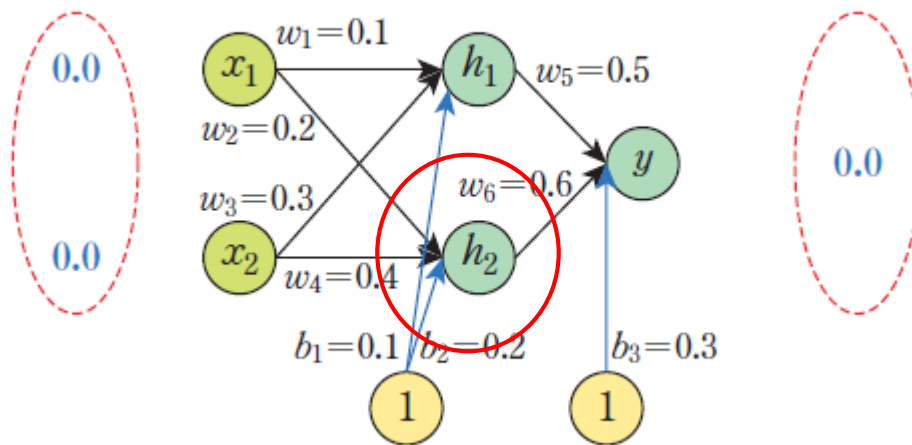
$$z_1 = w_1 * x_1 + w_3 * x_2 + b_1 = 0.1 * 0.0 + 0.3 * 0.0 + 0.1 = 0.1$$

$$a_1 = \frac{1}{1 + e^{-z_1}} = \frac{1}{1 + e^{-0.1}} = 0.524979$$



손으로 계산해보자.

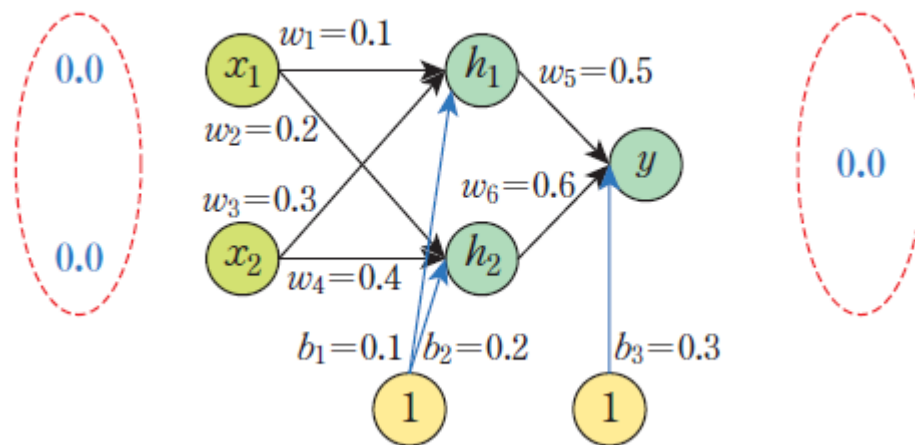
- H_2 의 출력값 계산



$$a_2 = 0.549834$$

손으로 계산해보자.

- y의 출력값 계산



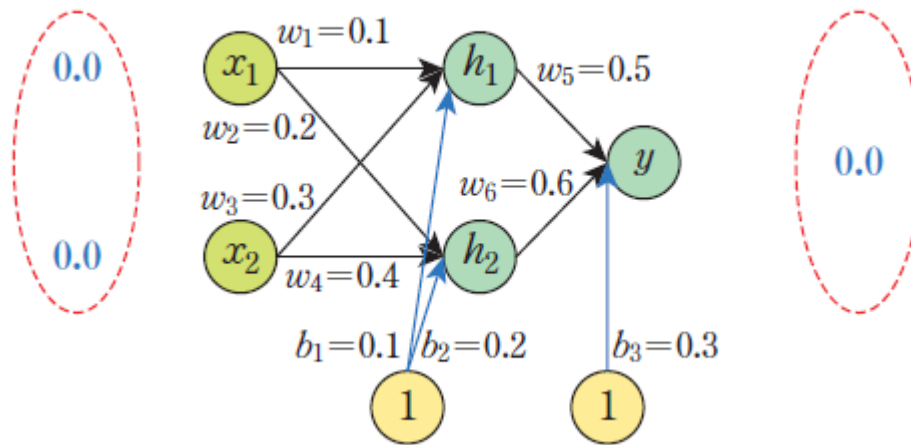
$$\begin{aligned} z_y &= w_5 * a_1 + w_6 * a_2 + b_3 \\ &= 0.5 * 0.524979 + 0.6 * 0.549834 + 0.3 = 0.892389 \end{aligned}$$

$$a_y = \frac{1}{1 + e^{-z_y}} = \frac{1}{1 + e^{-0.892389}} = 0.709383$$

정답은 0이지만 신경망의 출력은 0.71 정도이다.
오차가 상당함을 알 수 있다.



행렬로 표시해보자.



$$z_1 = w_1 * x_1 + w_3 * x_2 + b_1$$

$$z_2 = w_2 * x_1 + w_4 * x_2 + b_2$$

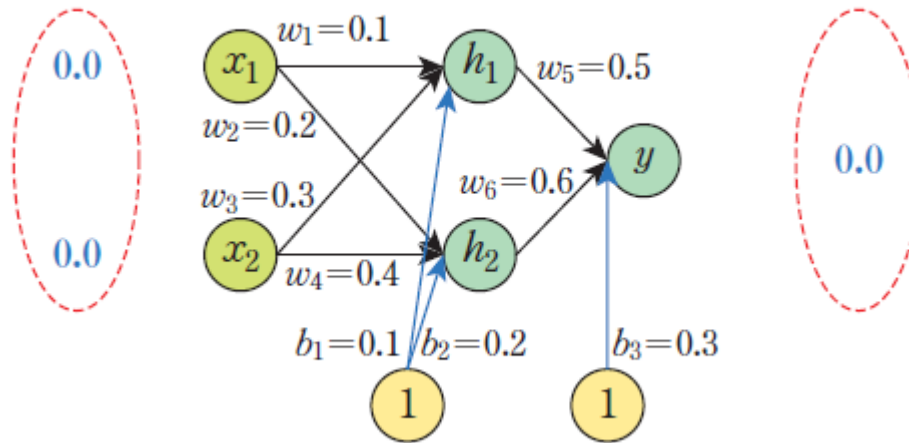


$$Z_1 = XW_1 + B_1$$

행렬로 표시할 수 있다.



행렬로 표시해보자.



$$X = [x_1 \ x_2], \quad B_1 = [b_1 \ b_2], \quad Z_1 = [z_1 \ z_2]$$

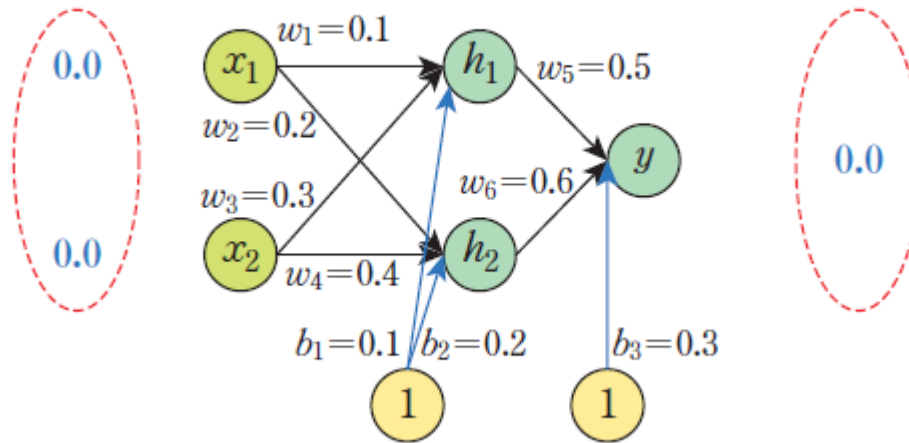
$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}$$



$$Z_1 = [z_1 \ z_2] = XW_1 + B_1 = [x_1 \ x_2] \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} + [b_1 \ b_2]$$



행렬로 표시해보자.



$$W_2 = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}$$

$$Z_2 = A_1 W_2 + B_2 = [a_1 \ a_2] \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} + [b_3]$$

$$A_2 = [y] = f(Z_2)$$



수학 복습 행렬의 곱셈

행렬의 곱셈은 다음과 같다. 첫 번째 행렬의 행과 두 번째 행렬의 열이 곱해져서 결과 행렬의 한 요소가 된다. 행과 열은 내적 계산과 동일하게 곱해진다.

$$\begin{array}{c} \text{내적} \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \\ \end{bmatrix} \\ \begin{array}{ccc} 2 \times 3 & 3 \times 2 & 2 \times 2 \end{array} \end{array}$$

행렬의 곱셈에서 항상 주의할 점은 첫 번째 행렬의 열의 개수와 두 번째 행렬의 행의 개수가 일치해야 한다는 점이다.

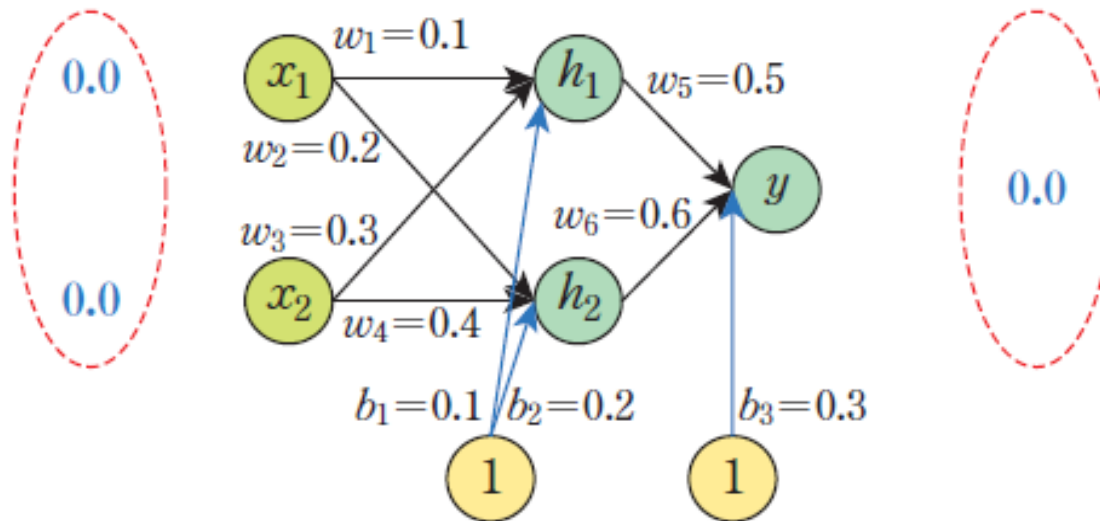
수학 복습 $WX+B$ 또는 $XW+B$?

입력 행렬과 가중치 행렬을 나타내는 방법은 책마다 조금씩 달라진다. 입력들이 행 벡터 형태로 행렬 X 에 쌓여 있다고 생각하면 $XW+B$ 와 같은 형태가 된다. 반대로 입력들이 열 벡터 형식으로 행렬 X 에 쌓여 있다고 생각하면 $WX+B$ 가 된다. 또 W 와 X 의 형태에 따라서 WX^T 로 표기해야 되는 경우도 있다. 본 책에서는 $XW+B$ 의 형태를 사용한다.

$$XW+B = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$
$$WX+B = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$



Lab: MLP 순방향 패스



손실 함수 계산

- 출력값이 올바르게 않으면 오차를 계산해서 오차를 줄이는 방향으로 가중치를 변경해야 한다.

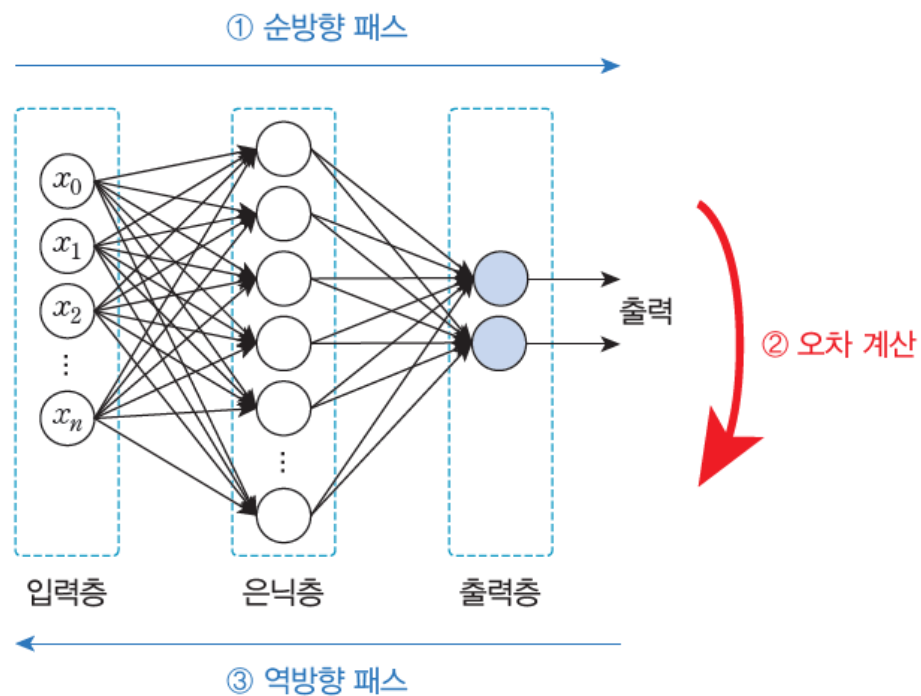
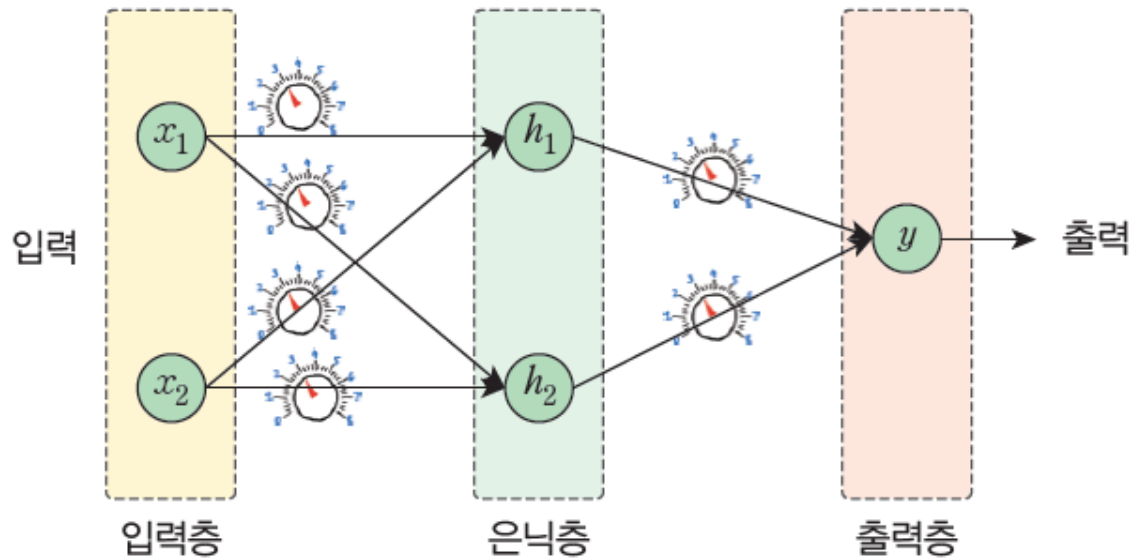


그림 6-6 오차 계산



가중치 == 다이어리

- 오차를 줄이는 방향으로 가중치를 조절한다.

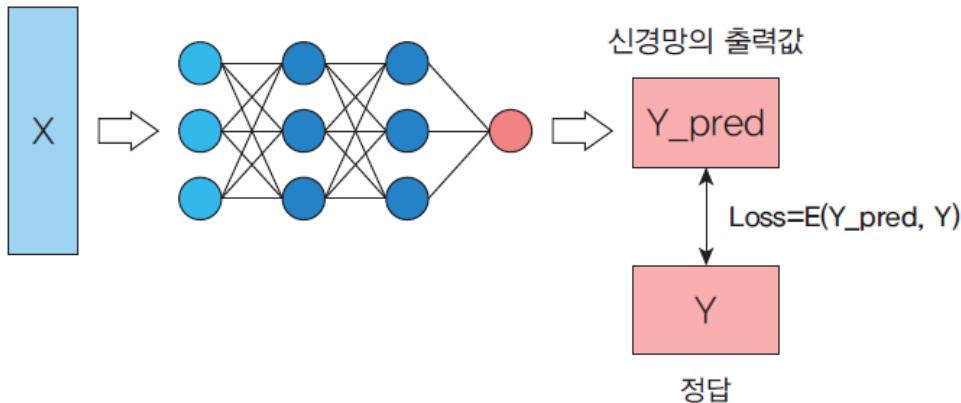




손실 함수(loss function)

- 신경망 학습의 성과를 나타내는 지표
- 정답과 출력값 사이의 오차
- 평균제곱오차(MSE: Mean Squared Error)가 많이 사용되었지만 최근에는 교차 엔트로피 함수가 인기가 있다.

입력 데이터



손실 함수는 출력과 정답
간의 차이로 정의됩니다.



그림 6-7 손실함수의 정의



평균 제곱 오차 (MSE)

- 예측값(y_i)과 정답(t_i) 간의 평균 제곱 오차

$$E(w) = \frac{1}{2} \sum_i (y_i - t_i)^2$$

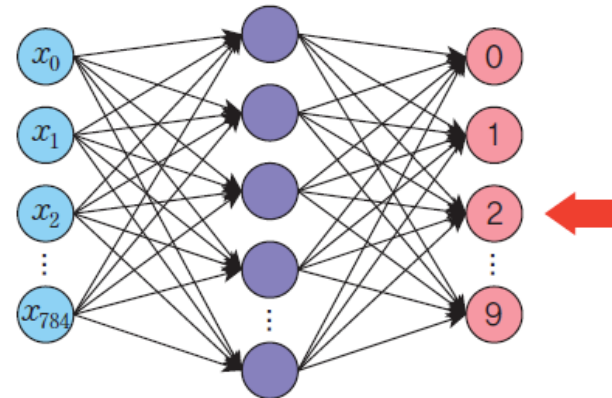


그림 6-8 MNIST 숫자 이미지를 분류하는 신경망

- 제곱하는 이유 ?
 - 오차를 양수화
 - 가중치를 조금만 변경하더라도 큰 오차변화를 확인할 수 있어 가중치의 미세조정에 효과적



평균 제곱 오차 (MSE)

```
>>> y = np.array([ 0.0, 0.0, 0.8, 0.1, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0 ])
>>> target = np.array([ 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ])

>>> def MSE(target, y):
    return 0.5 * np.sum((y-target)**2)

>>> MSE(target, y)
0.03
```

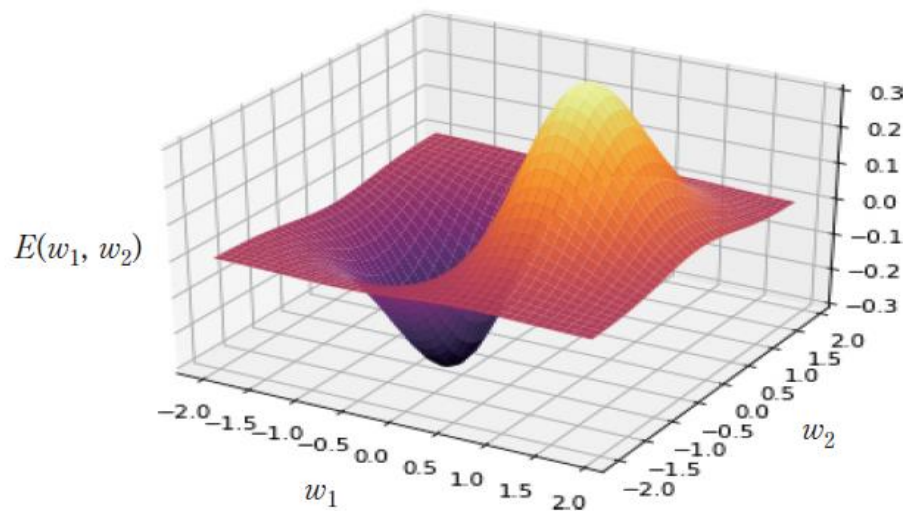
```
>>> y = np.array([ 0.9, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ])
>>> target = np.array([ 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ])

>>> def MSE(target, y):
    return 0.5 * np.sum((y-target)**2)

>>> MSE(target, y)
0.81
```


- 학습이란 손실함수를 최소로 만드는 가중치를 찾는 것

$$W^* = \operatorname{argmin}_W E(W)$$



옆의 평면에서 가장 낮은 값을 찾으면 됩니다.

그림 6-9 경사 하강법



경사하강법의 재소개

- 경사하강법 (gradient-descent method) : 함수의 1차 미분값(그라디언트)를 사용하는 반복적인 최적화 알고리즘
- 손실함수를 감소시키려면 미분값의 반대방향으로 가면 된다.

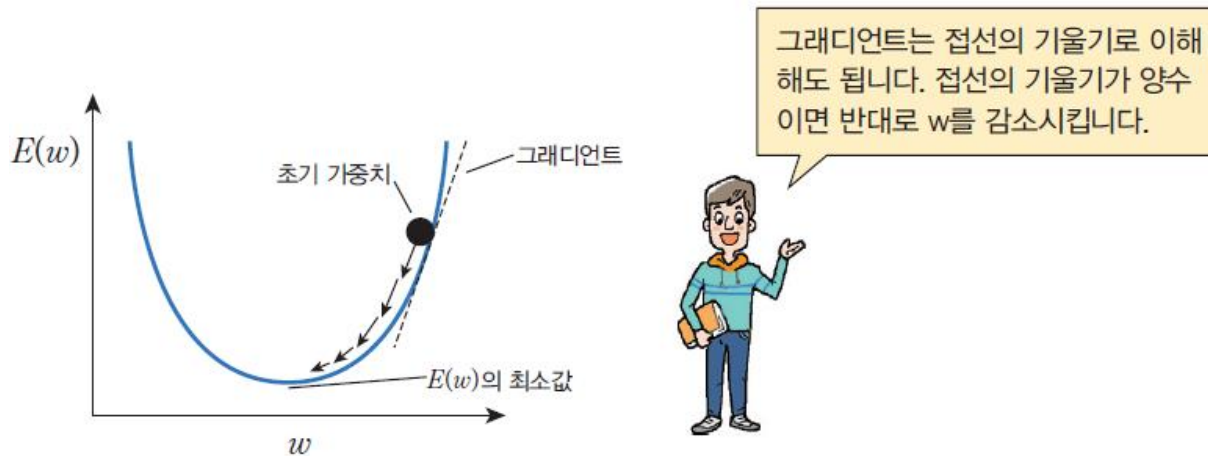


그림 6-11 경사 하강법

손실함수를 가중치로 미분한 값이 양수이면



가중치를 감소시킨다.

손실함수를 가중치로 미분한 값이 음수이면



가중치를 증가시킨다.



Lab: 경사하강법의 실습

- 손실 함수가 $y = (x - 3)^2 + 10$ 일 때
- 그래디언트 $y' = 2x - 6$
- $x = x - \text{학습률} * y'$: 그래디언트와 반대방향으로 가면서 x 를 구하기 위해 음수 곱셈

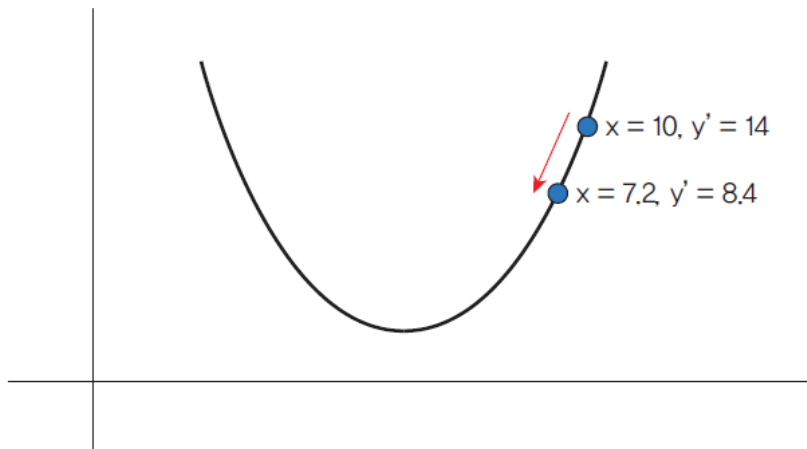


그림 6-12 그래디언트의 계산

$x = 10$ 일때

$$\text{손실함수} = (10 - 3)^2 + 10 = 59$$

$$y' = 2x - 6 = 2 * 10 - 6 = 14$$

$$\text{학습률}(0.2) * (-14) = -2.8$$

$$x = -2.8 + 10 = 7.2$$

$$\text{손실함수} = (7.2 - 3)^2 + 10 = 27.64$$

$$y' = 2x - 6 = 2 * 7.2 - 6 = 8.4$$

$$\text{학습률}(0.2) * (-8.4) = -1.68$$

$$x = -1.68 + 7.2 = 5.52$$

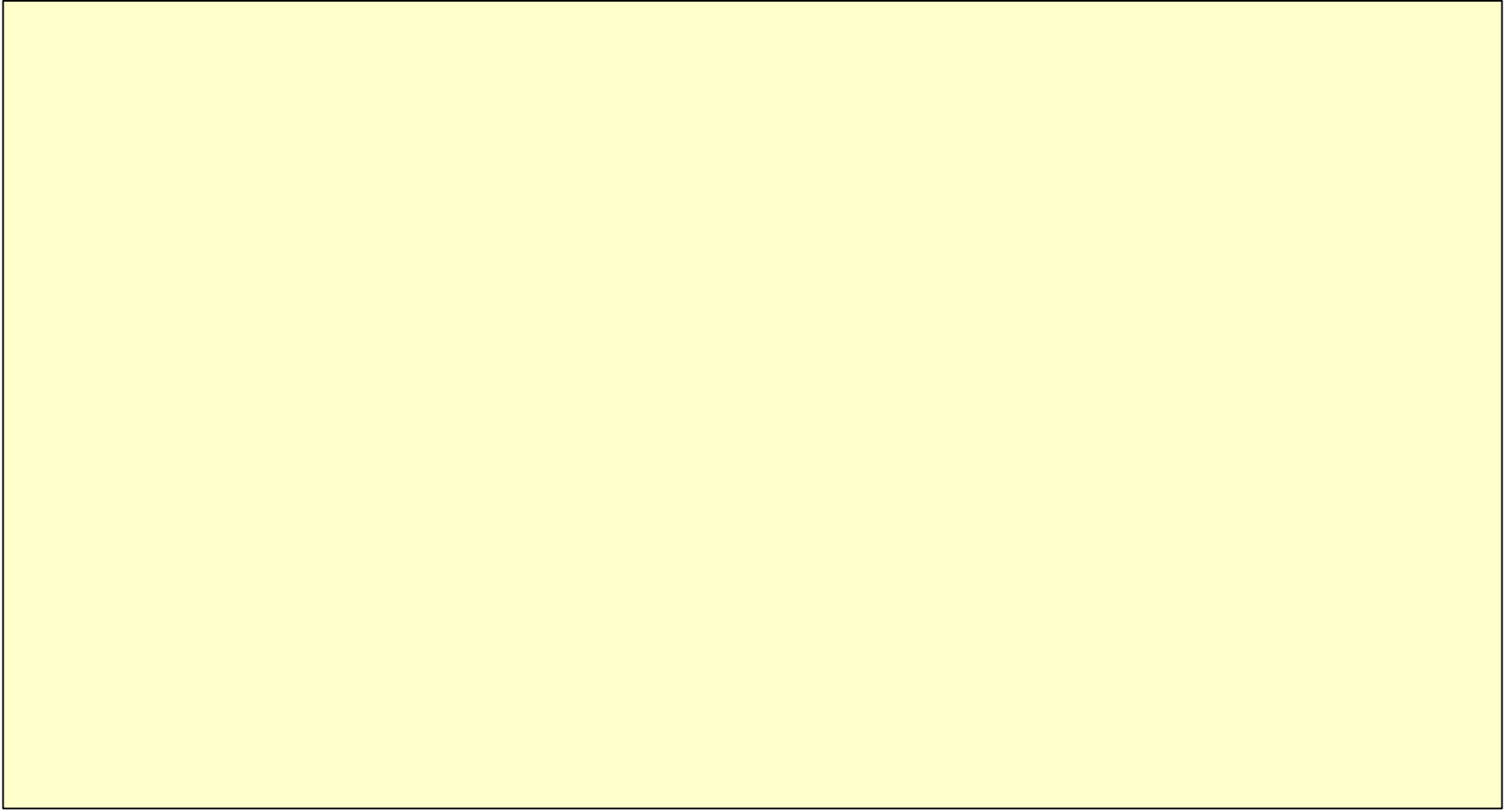
$$\text{손실함수} = (5.52 - 3)^2 + 10 = 16.35$$

...

...



경사 하강법 실습





Lab: 2차원 그래디언트 시각화

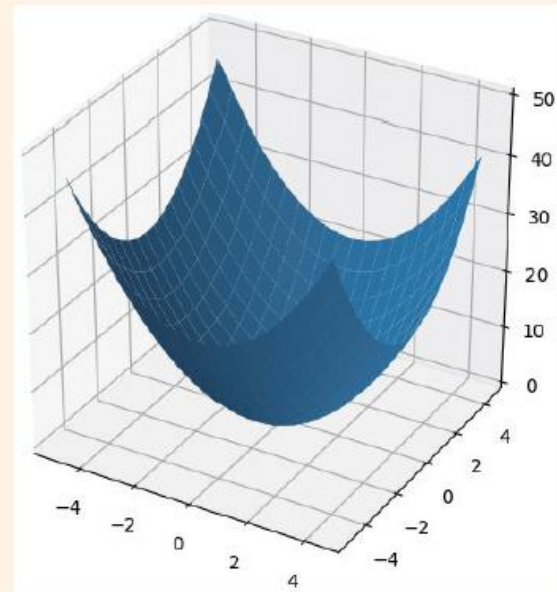
- 손실 함수가 $f(x,y) = x^2 + y^2$ 라고 가정하자. 실제로는 이와 같이 여러 개의 입력(변수)에 의해 구성된다. 비선형 구조이므로 편미분이 필요.
- 그래디언트를 계산하면 = $(2x, 2y)$

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.arange(-5, 5, 0.5)
y = np.arange(-5, 5, 0.5)
X, Y = np.meshgrid(x, y) # 참고 박스
Z = X**2 + Y**2           # 넘파이 연산
```

```
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')
```

```
# 3차원 그래프를 그린다.
ax.plot_surface(X, Y, Z)
plt.show()
```





Lab: 2차원 그래디언트 시각화

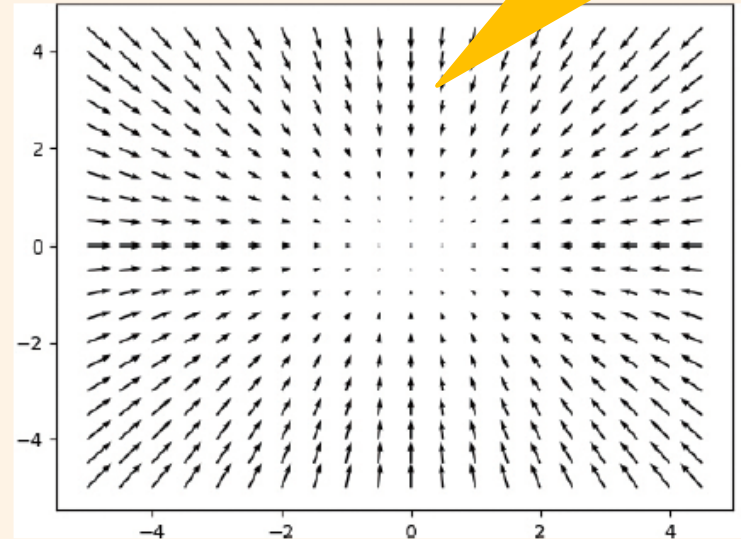
- 그래디언트를 계산하면 = $(2x, 2y)$. 편미분 필요

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

x = np.arange(-5,5,0.5)
y = np.arange(-5,5,0.5)
X, Y = np.meshgrid(x,y)
U = -2*X
V = -2*Y

plt.figure()
Q = plt.quiver(X, Y, U, V, units='width')
plt.show()
```

그래디언트의 음수



화살표가 최소값을 가리키고 있음을 알 수 있다.

- 어떤 위치에서든지 그래디언트의 역방향으로 가면 최저값에 도달할 수 있음을 알 수 있다.



역전파 학습 알고리즘

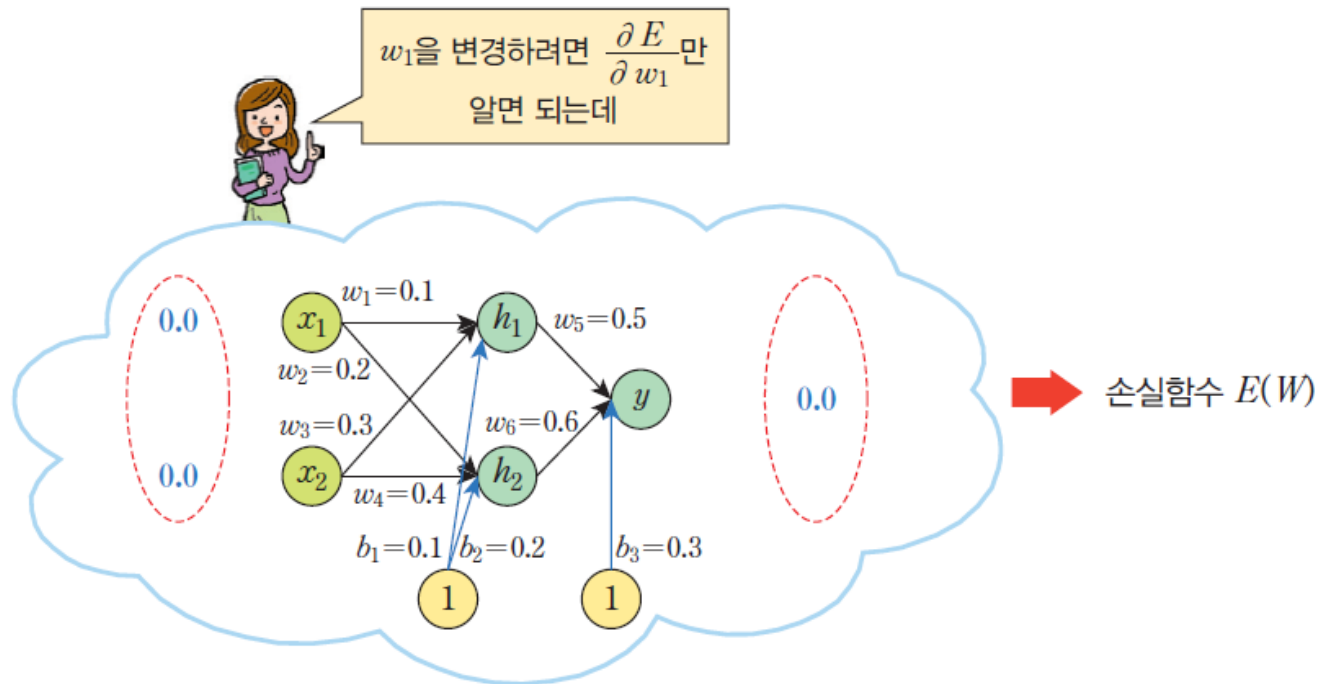
- 입력이 주어지면 순방향으로 계산하여 출력을 계산한 후에 실제 출력과 우리가 원하는 출력 간의 오차를 계산한다. 이 오차를 역방향으로 전파하면서 오차를 줄이는 방향으로 가중치를 변경한다.

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w}$$

- ① 가중치와 바이어스를 0부터 1 사이의 난수로 초기화한다.
- ② 수렴할 때까지 모든 가중치에 대하여 다음을 반복한다.
- ③ 손실함수 E 의 그래디언트 $\partial E / \partial w$ 을 계산한다.
- ④ $w(t+1) = w(t) - \eta \frac{\partial E}{\partial w}$

역전파 알고리즘

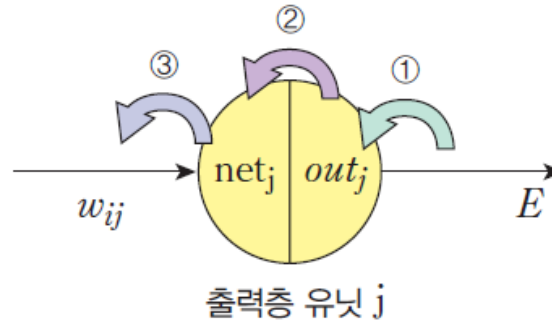
- 체인룰을 이용하여 유도가 가능하다.





역전파 알고리즘

$$\frac{\partial E}{\partial w_{ij}} = \underbrace{\frac{\partial E}{\partial out_j}}_{\textcircled{1}} \underbrace{\frac{\partial out_j}{\partial net_j}}_{\textcircled{2}} \underbrace{\frac{\partial net_j}{\partial w_{ij}}}_{\textcircled{3}}$$



$$\textcircled{1} \quad \frac{\partial E}{\partial out_j} = \frac{\partial}{\partial out_j} \sum \frac{1}{2} (target_k - out_k)^2 = out_j - target_j$$

유닛의 출력값 변환에 따른 오차의 변화율이다.

$$\textcircled{2} \quad \frac{\partial out_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(net_j)$$

입력값의 변화에 따른 유닛 j의 출력 변화율이다.
활성화 함수의 미분값이다.

$$\textcircled{3} \quad \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=0}^n w_{kj} out_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} out_i = out_i$$

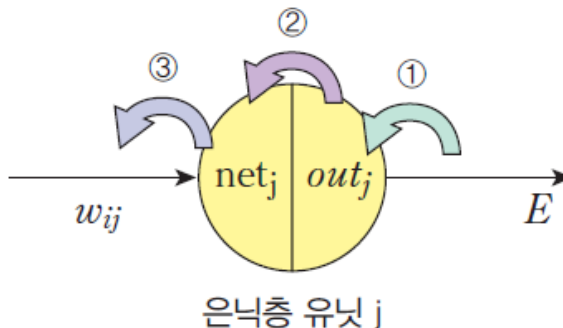
가중치의 변화에 따른 net_j의 변화율이라고 할 수 있다.

$$\therefore \frac{\partial E}{\partial w_{ij}} = \textcircled{1} \times \textcircled{2} \times \textcircled{3} = (out_j - target_j) \times f'(net_j) \times out_i$$



역전파 알고리즘

$$\frac{\partial E}{\partial w_{ij}} = \underbrace{\frac{\partial E}{\partial out_j}}_{\textcircled{1}} \underbrace{\frac{\partial out_j}{\partial net_j}}_{\textcircled{2}} \underbrace{\frac{\partial net_j}{\partial w_{ij}}}_{\textcircled{3}}$$

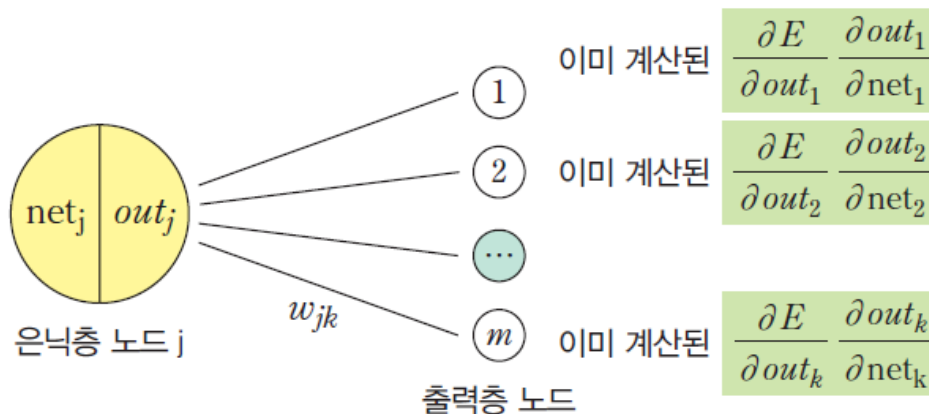


그런데 은닉층 유닛의 오차 E는 어떻게 계산하지?



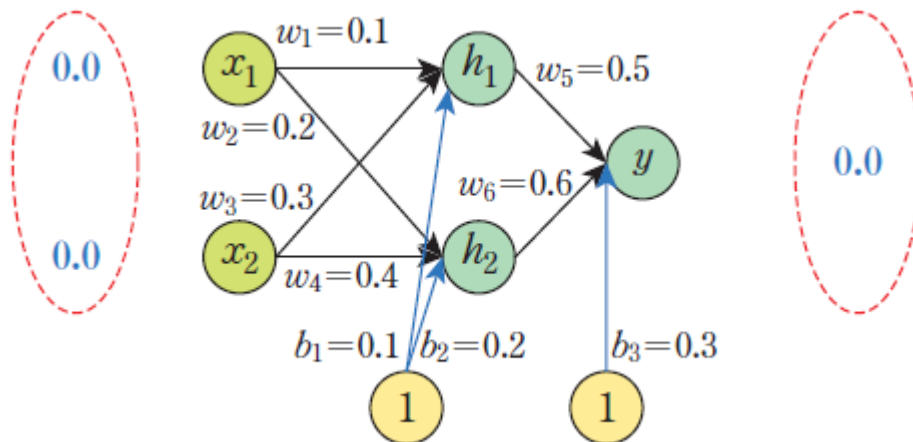
$$\frac{\partial E}{\partial out_j} = \sum_{k \in L} \left(\frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k} \frac{\partial net_k}{\partial out_j} \right) = \sum_{k \in L} \left(\frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k} w_{jk} \right)$$

이것도 체인룰이다!





여전파 알고리즘 순으로 계산해보자.



- 순방향 패스

$$\begin{aligned} \text{net}_y &= w_5 * \text{out}_{h1} + w_6 * \text{out}_{h2} + b_3 \\ &= 0.5 * 0.524979 + 0.6 * 0.549834 + 0.3 = 0.89239 \end{aligned}$$

$$\text{out}_y = \frac{1}{1 + e^{-\text{net}_y}} = \frac{1}{1 + e^{-0.89239}} = 0.709383$$

앞
표
표
참
조!

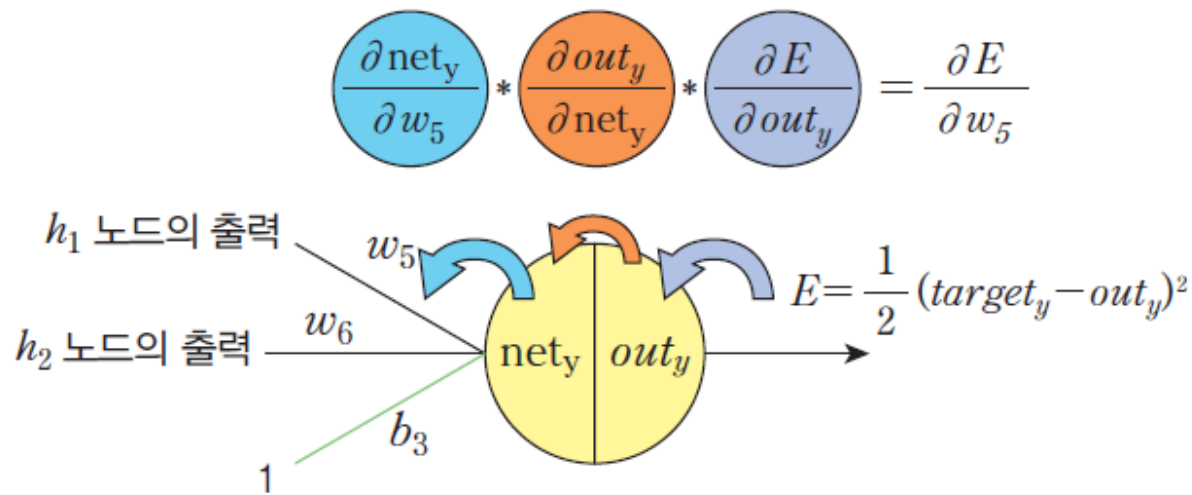


역전파 알고리즘 손으로 계산해보자.

- 총오차 계산

$$E = \frac{1}{2}(target_y - out_y)^2 = \frac{1}{2}(0.00 - 0.709383)^2 = 0.251612$$

- $\frac{\partial E}{\partial w_5}$ 만 계산해보자.





경사하강법 적용

- ① $\frac{\partial E}{\partial out_y} = 2 * \frac{1}{2}(target_y - out_y)^{2-1} * (-1) = (out_y - target_y)$
 $= (0.709383 - 0.00) = 0.709383$
- ② $\frac{\partial out_y}{\partial net_y} = f'(out_y) = out_y * (1 - out_y) = 0.709383 * (1 - 0.709383) = 0.206158$
- ③ $net_y = w_5 * out_{h1} + w_6 * out_{h2} + b_3 * 1$
 $\frac{\partial net_y}{\partial w_5} = 1 * out_{h1} + 0 + 0 = 0.524979$



$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_y} \frac{\partial out_y}{\partial net_y} \frac{\partial net_y}{\partial w_5}$$
$$= 0.709383 * 0.206158 * 0.524979 = 0.076775$$

$$w_5(t+1) = w_5(t) + \eta * \frac{\partial E}{\partial w_5} = 0.5 - 0.2 * 0.076775 = 0.484645$$



유니츠의 가중치와 바이어스

$$w_5(t+1) = w_5(t) + \eta * \frac{\partial E}{\partial w_5} = 0.5 - 0.2 * 0.076775 = 0.484645$$

$$w_6(t+1) = 0.583918$$

$$b_3(t+1) = 0.270750$$

가중치도 낮아지게 된다. . 현재 우리가 얻는 출력값은 0
이기 때문이다.

바이어스는 기존 값 보다 낮아지게 된다. 따라서 다음 번에
는 유니츠의 출력을 더 낮게 만들 거
이다. 현재 우리가 얻는 출력값은 0이기 때문이다.



입력층 -> 은닉층의 가중치와 바이어스

$$w_1(t+1) = w_1(t) + \eta * \frac{\partial E}{\partial w_1} = 0.10 - 0.2 * 0.0 = 0.10$$

$$w_2(t+1) = 0.2, \quad w_3(t+1) = 0.3, \quad w_4(t+1) = 0.4$$

입력값이 0이어서 가중치는 변경되지 않았다 (이제 은 퍼셉트론과 유사하다. 입력이 0이면 가중치를 아무리 바꿔도 무슨 소용인가?).

$$b_1(t+1) = 0.096352, \quad b_2(t+1) = 0.195656$$

이런 경우에는 바이어스가 큰 역할을 한다 (이래서 바이어스를 반드시 입력해야 한다) **바이어스는 기존 값 보다 낮아지게 된다.** 따라서 다음 번에는 은닉층의 출력을 더 낮게 만들 거다. 현재 우리가 얻는 출력값은 0이기 때문이다.

$$E = \frac{1}{2}(target - out_y)^2 = \frac{1}{2}(0.00 - 0.709383)^2 = 0.251612$$



경사하강법 1번 적용

$$E = \frac{1}{2}(target - out_y)^2 = \frac{1}{2}(0.00 - 0.699553)^2 = 0.244687$$



경사하강법 10000번 적용

$$E = \frac{1}{2}(target - out_y)^2 = \frac{1}{2}(0.00 - 0.005770)^2 = 0.000016$$

오차가 크게 줄었다.



넘파이를 이용한 MLP 구현



- 이런 복잡한 코딩을 해결하기 위해 TF, Keras가 등장(p.258)



구글의 플레이그라운드를 이용한 실험

- 어려운 수학없이 신경망의 아이디어를 파악하는데 도움이 되는 도구
- 자바 스크립트로 작성된 웹 애플리케이션으로 웹 브라우저에서 실행
- 사용자가 딥러닝 모델을 구성하고 여러 가지 매개 변수를 조정하면서 실험할 수 있는 기능을 제공



구글의 플레이그라운드

A Neural Network Playground

playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03&r...

학습 시작 버튼

Epoch: 000,000
Learning rate: 0.03
Activation: Sigmoid
Regularization: None
Regularization rate: 0
Problem type: Classification

DATA
Which dataset do you want to use?

입력 데이터 세트
Which properties do you want to feed in?

4 neurons
2 neurons

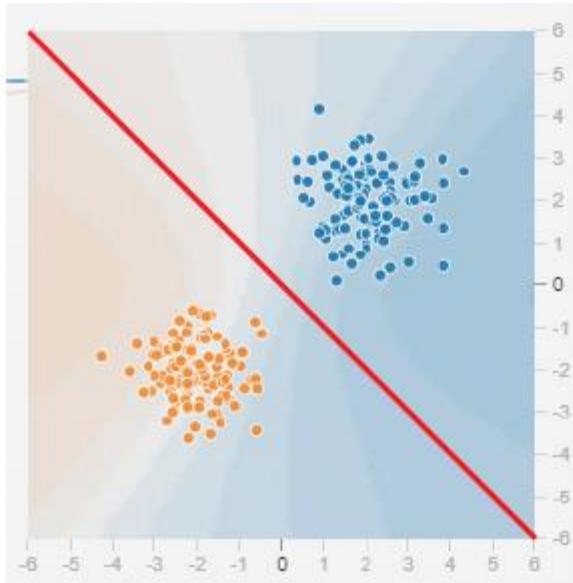
OUTPUT
Test loss 0.493
Training loss 0.497

입력층
은닉층
출력층

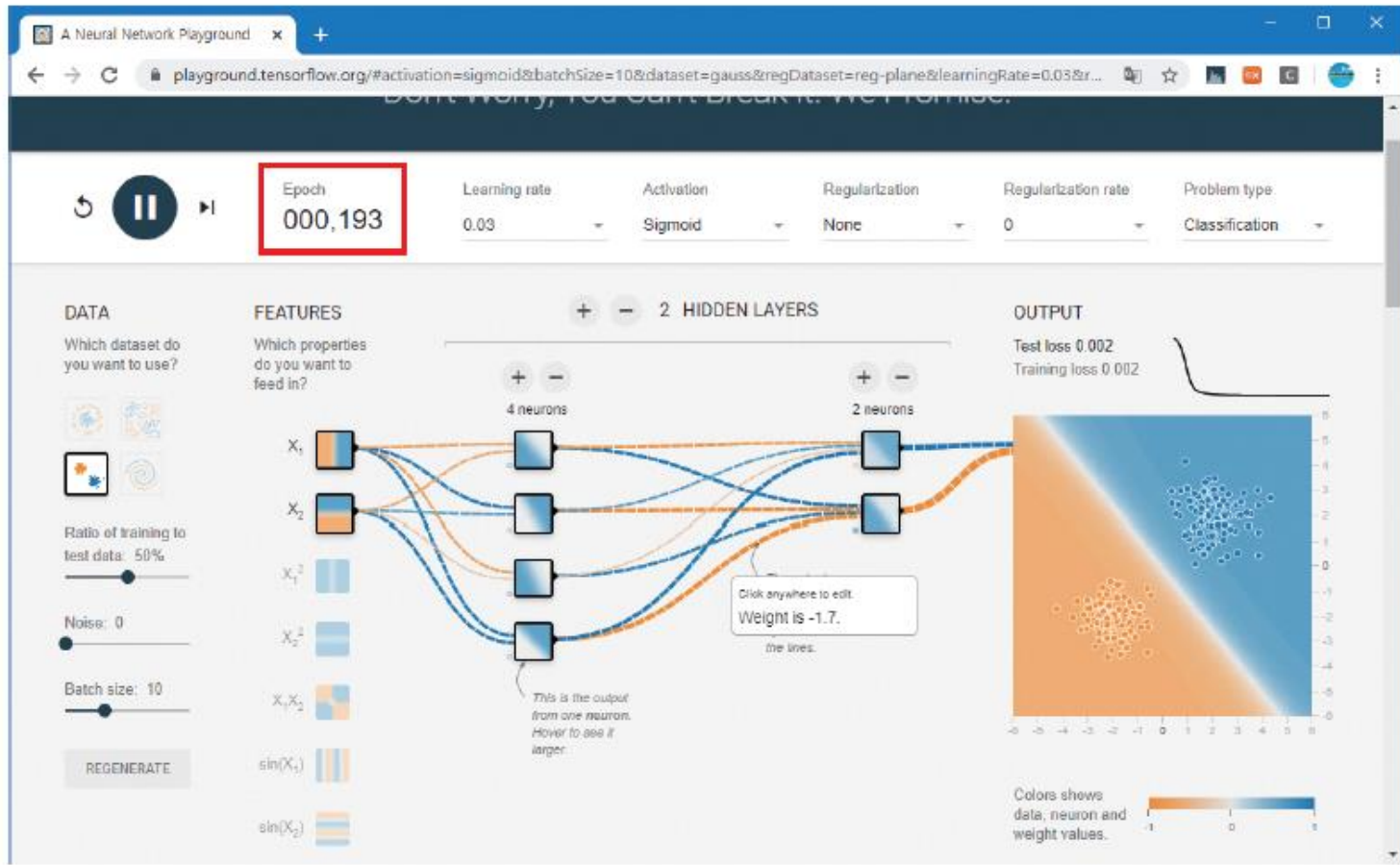
weight values.

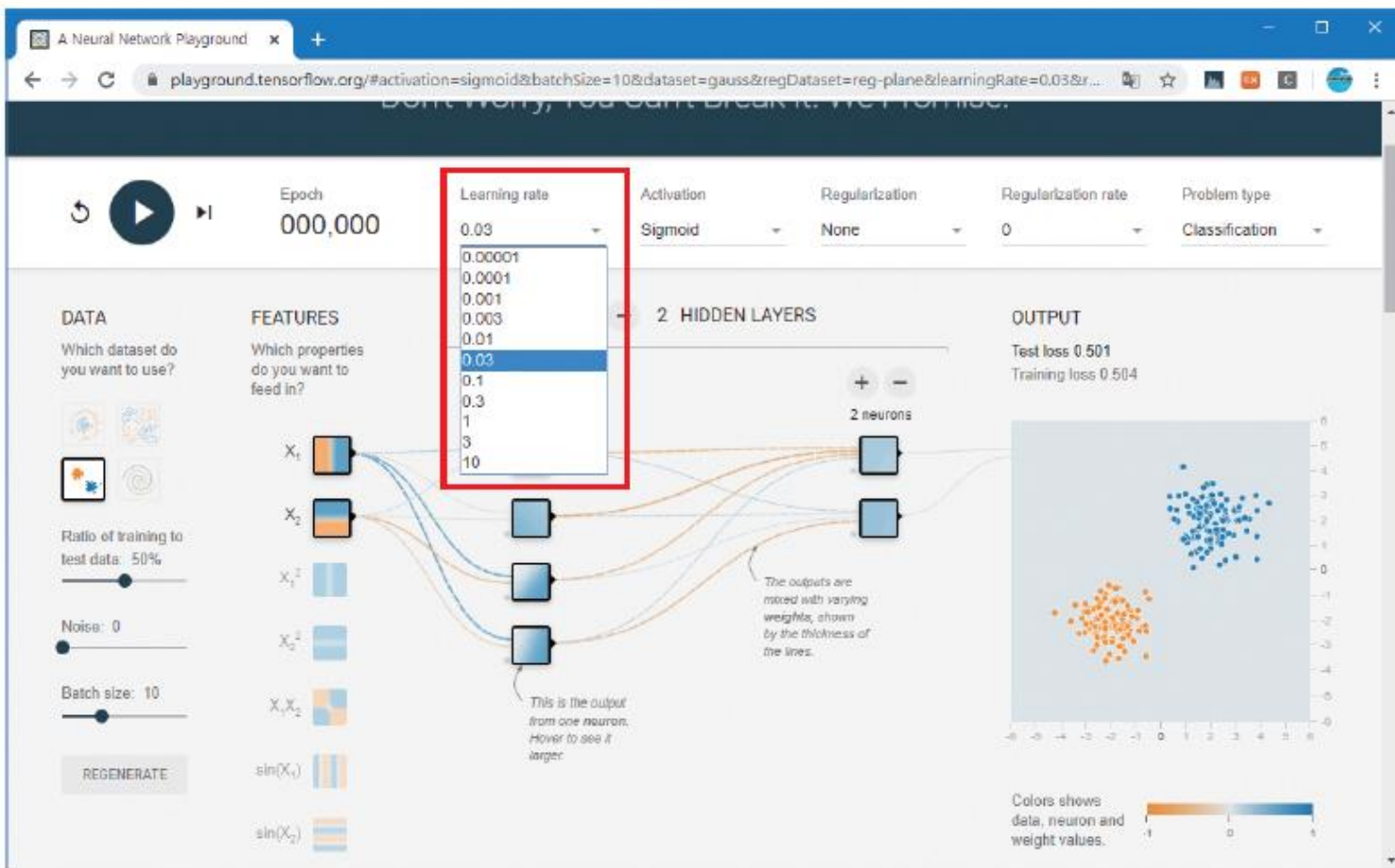


선형 분리 가능한 입력 데이터



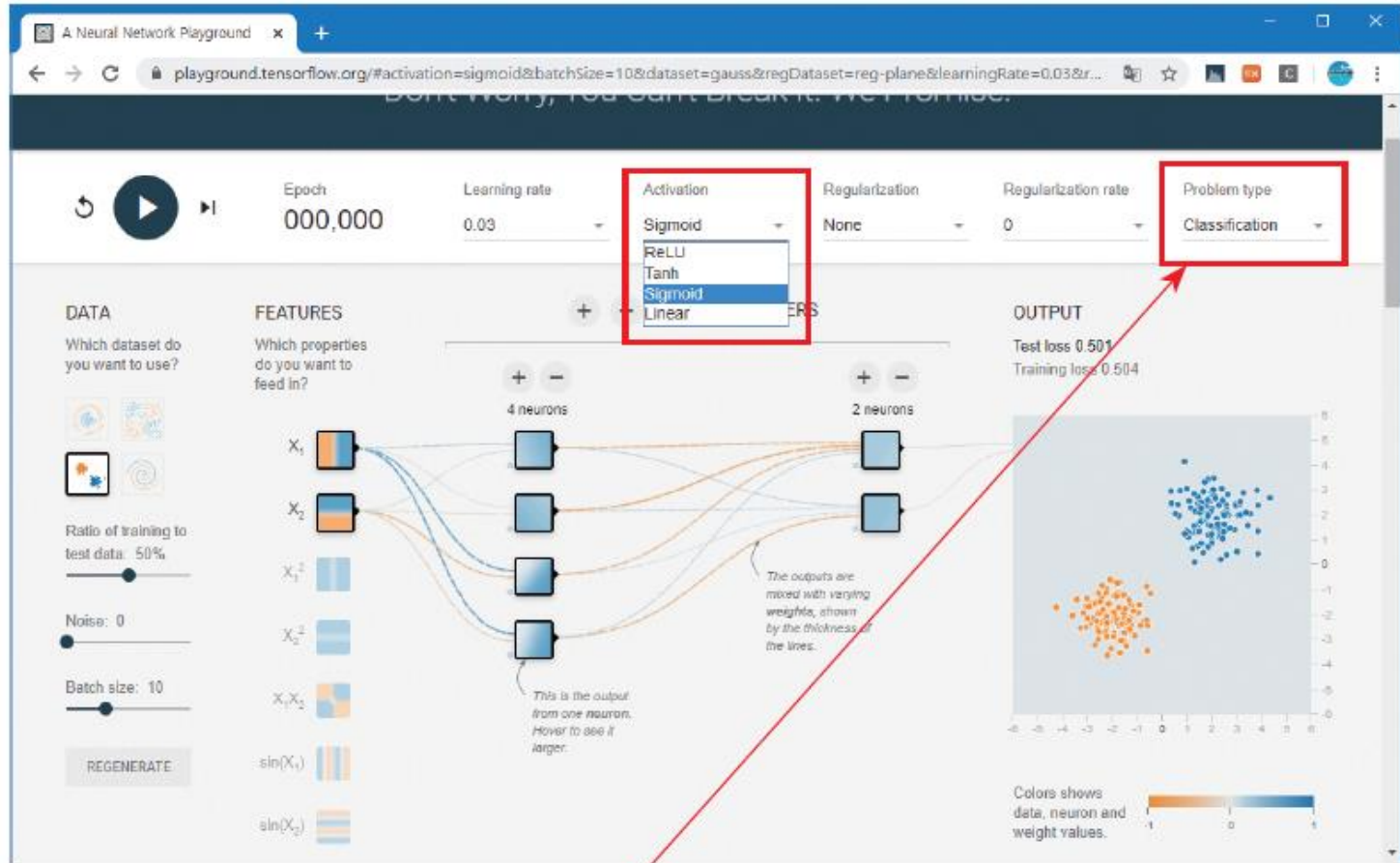
$$w_1x_1 + w_2x_2 + b = 0$$





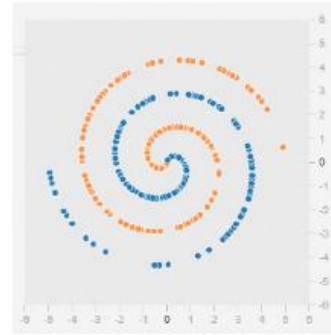
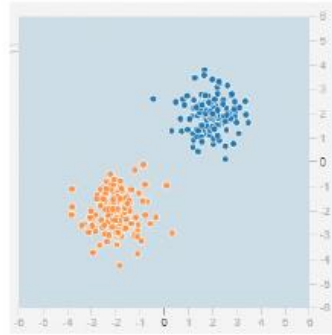
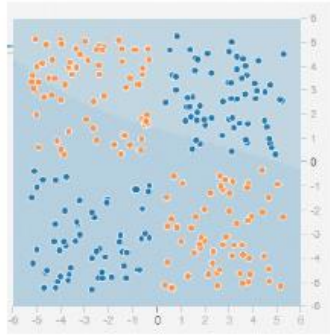
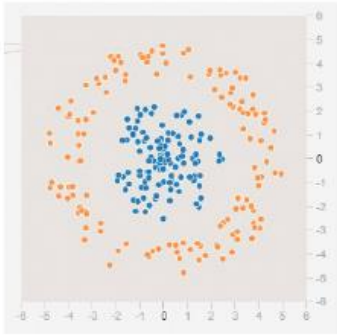


활성화 함수 선택

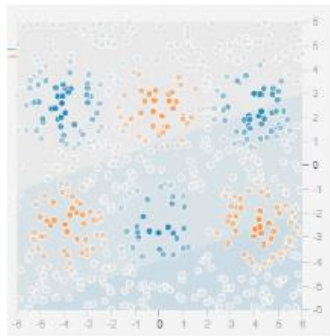
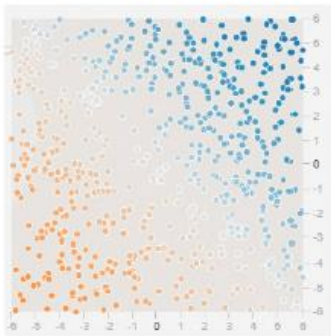


문제 유형

- 분류 문제



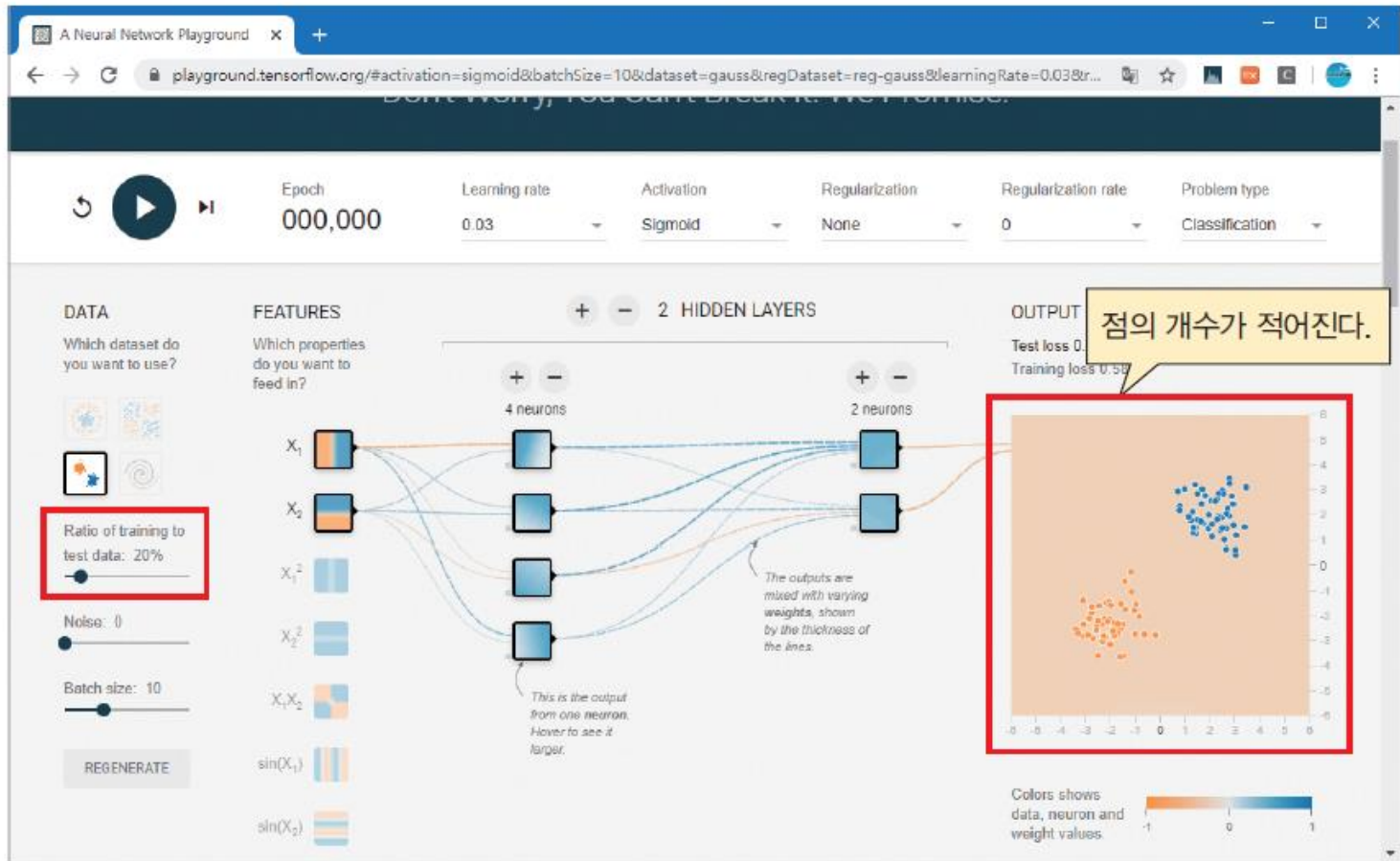
- 회귀 문제

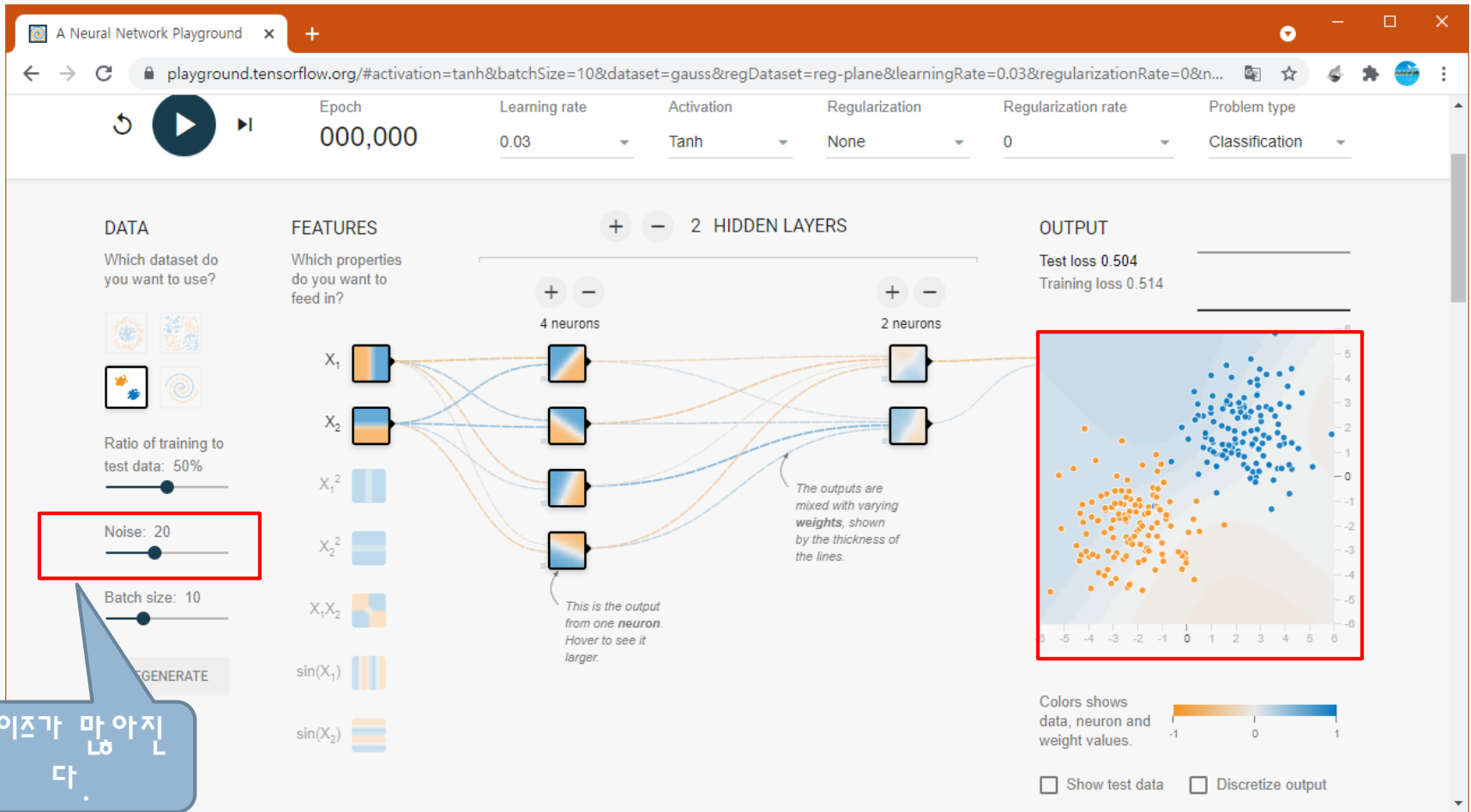




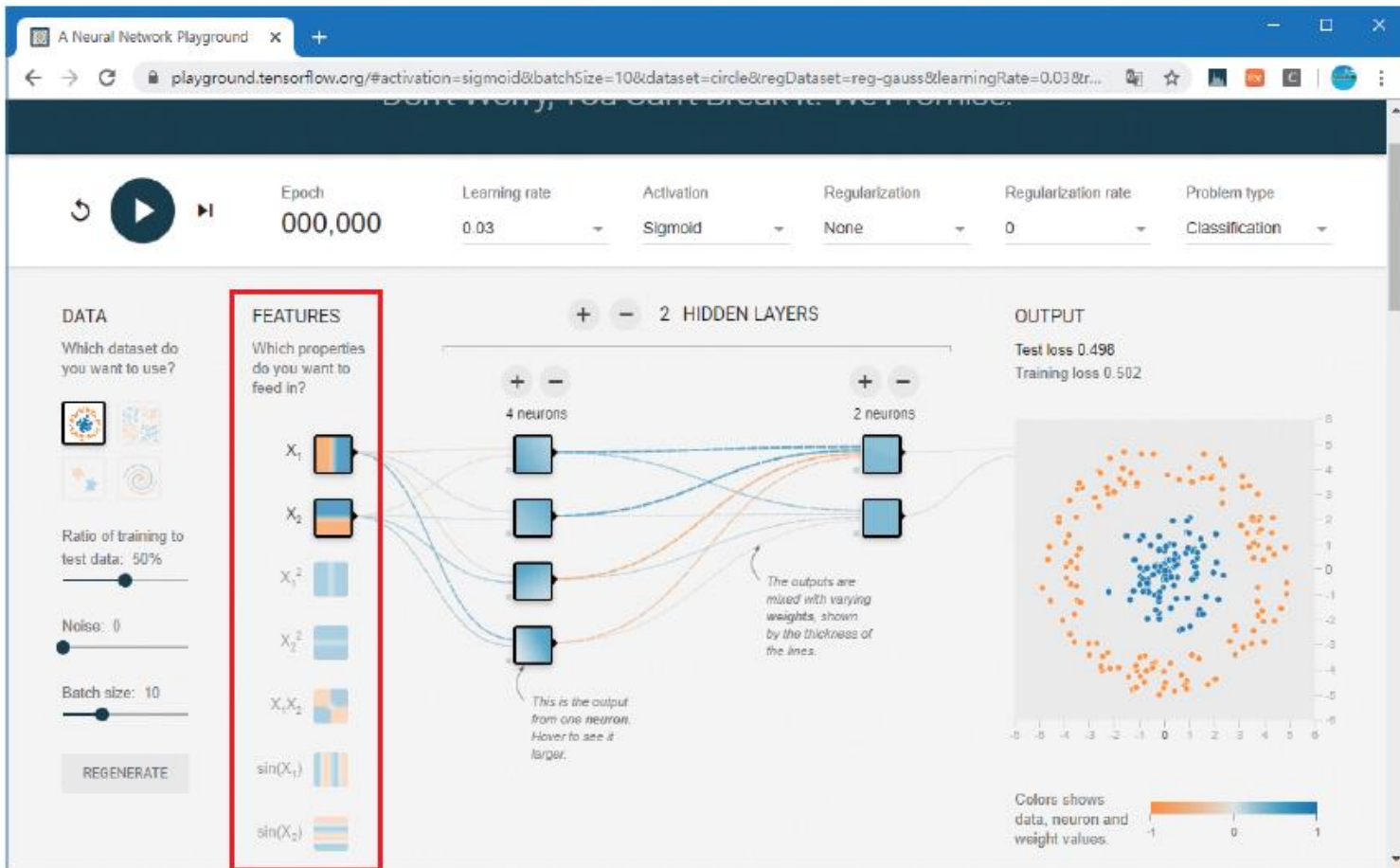
학습
기법

데이터와 테스트 데이터의 비율

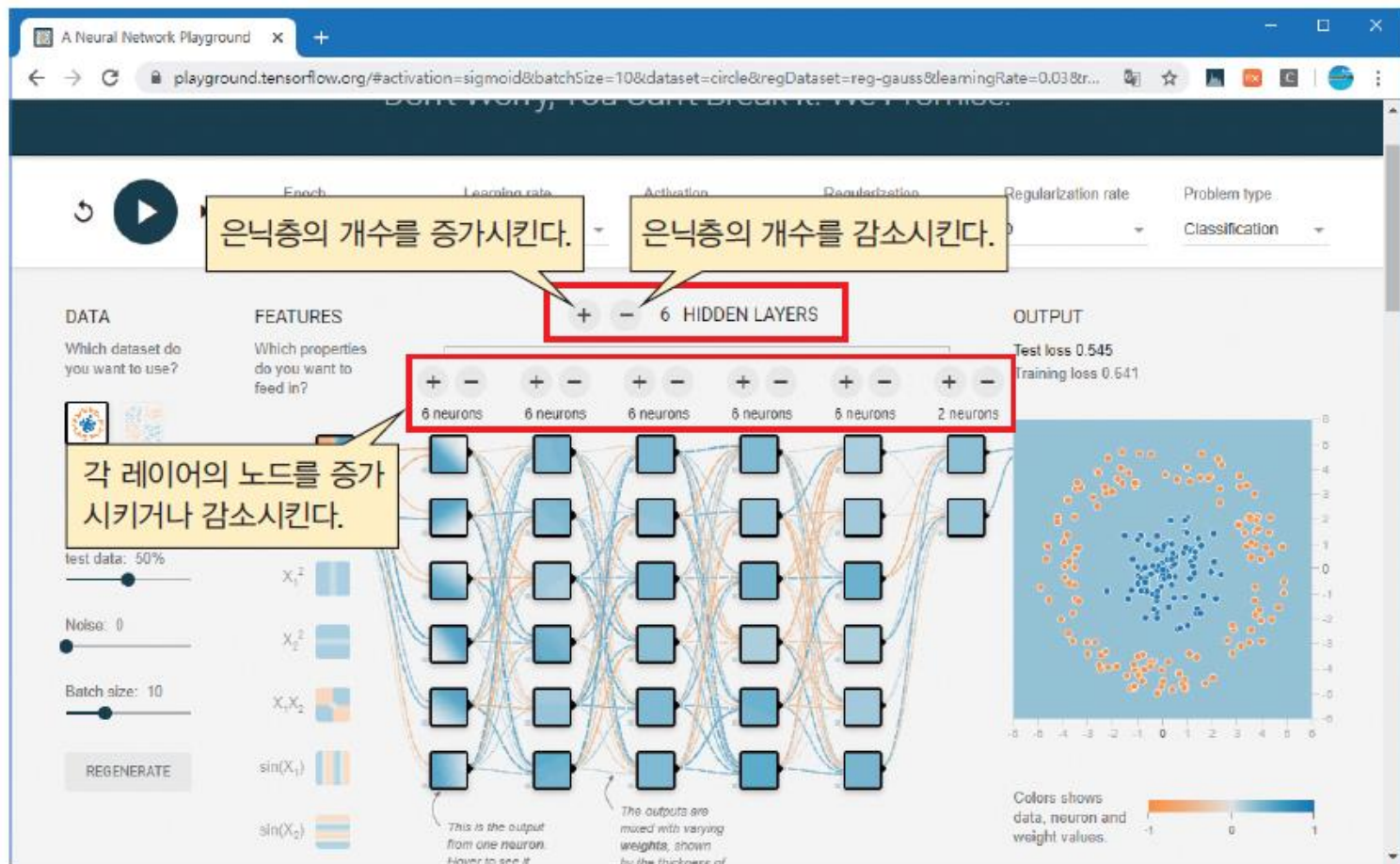




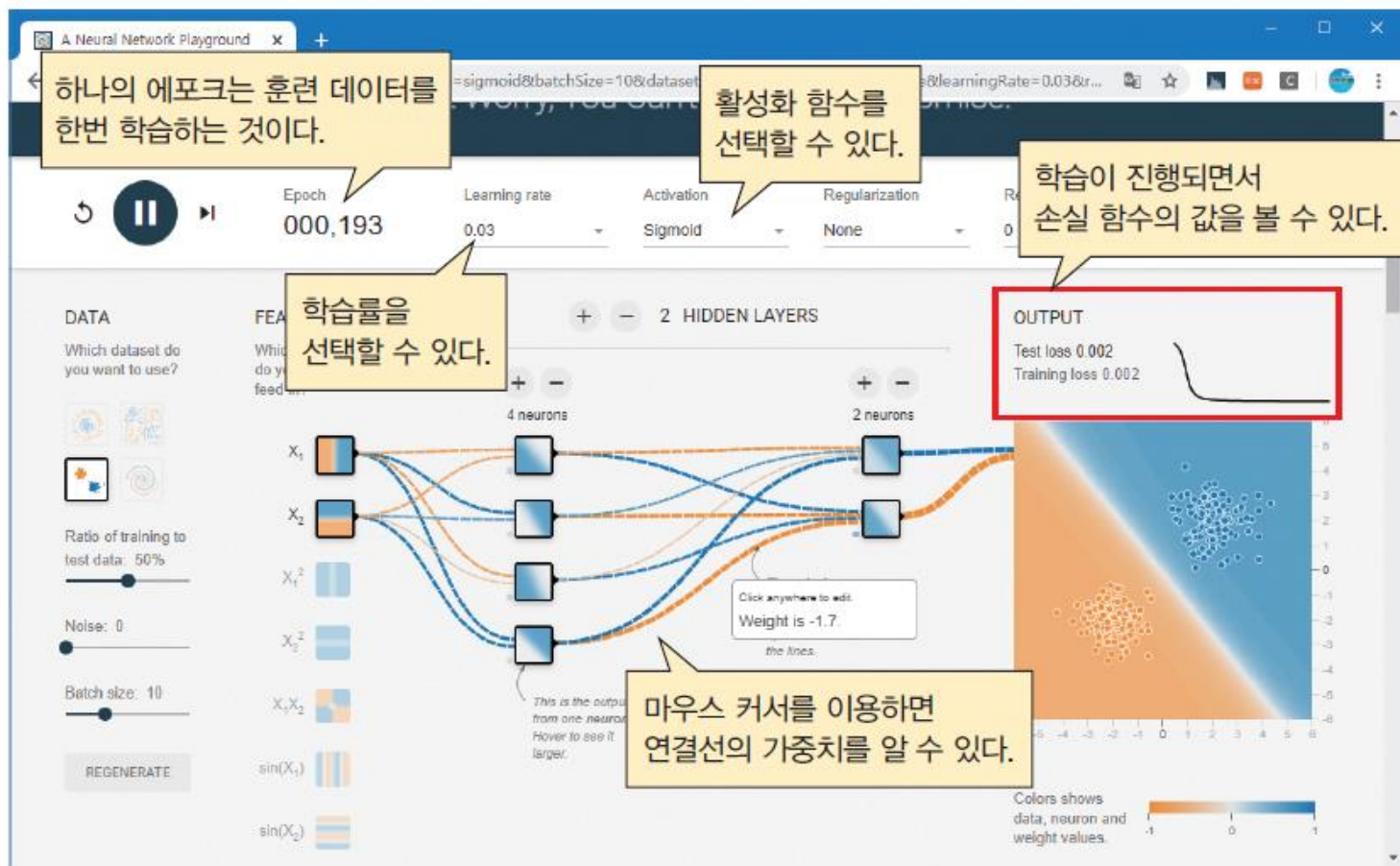
이력 특징 선택

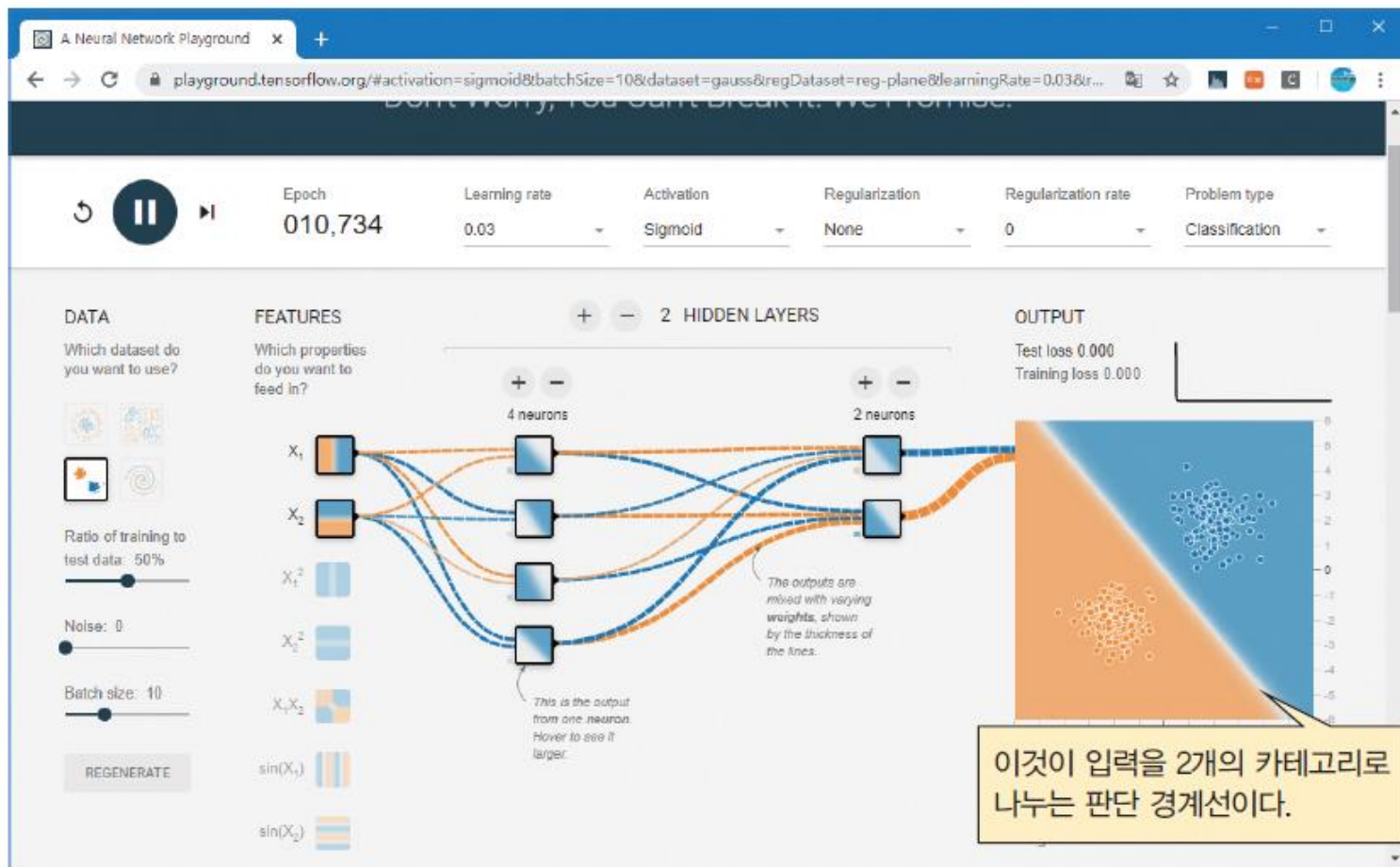


인공지능 추가하기



학습 시작







은닉층 없이 분류 시스

A Neural Network Playground

playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=xor®Dataset=reg-gauss&learningRate=0.03®...

Don't Worry, You Can't Break It. We Promise.

Epoch: 006,037
Learning rate: 0.03
Activation: Sigmoid
Regularization: None
Regularization rate: 0
Problem type: Classification

XOR와 유사한 데이터를 선택한다.

시그모이드 함수를 선택한다.

학습이 완료되지 않는다.

1 HIDDEN LAYER
2 neurons

OUTPUT
Test loss 0.277
Training loss 0.180

출력 유닛을 두 개만 남긴다.
은닉층은 없다.

Colors shows

은닉층을 추가한 시스 템

