2020531001 윤정인

인공지능

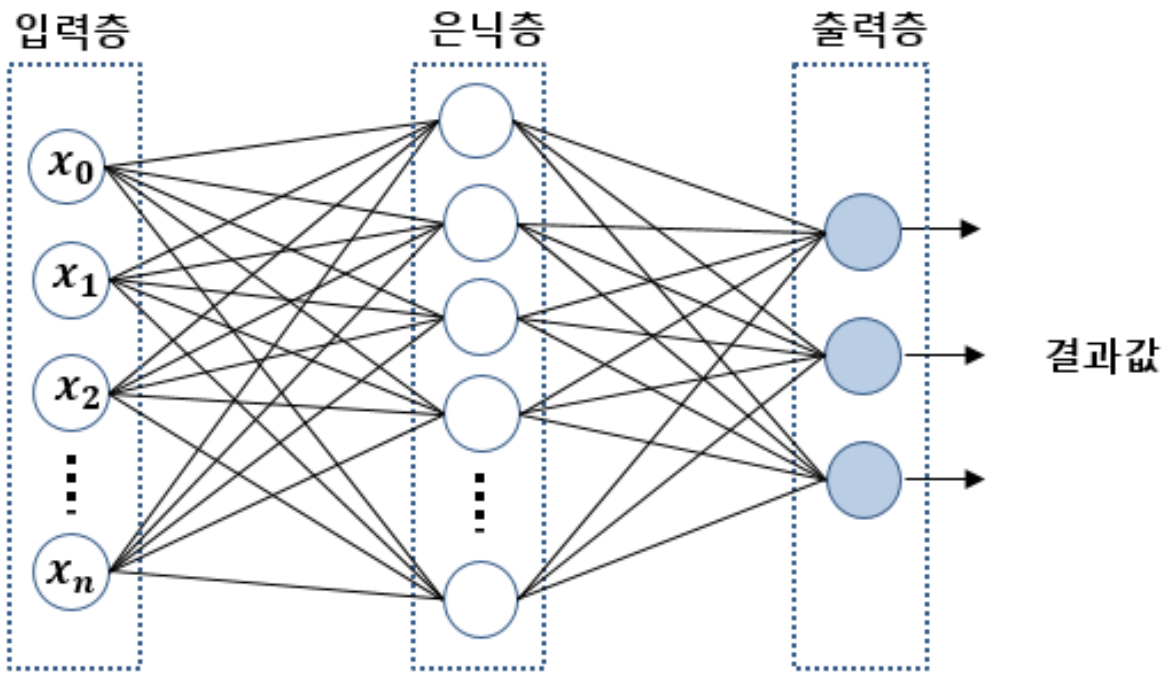
CHAP 13 – 신경망 2 (MLP)

Contents

01

다층 퍼셉트론

다층 퍼셉트론(MLP : Multilayer Perceptron)

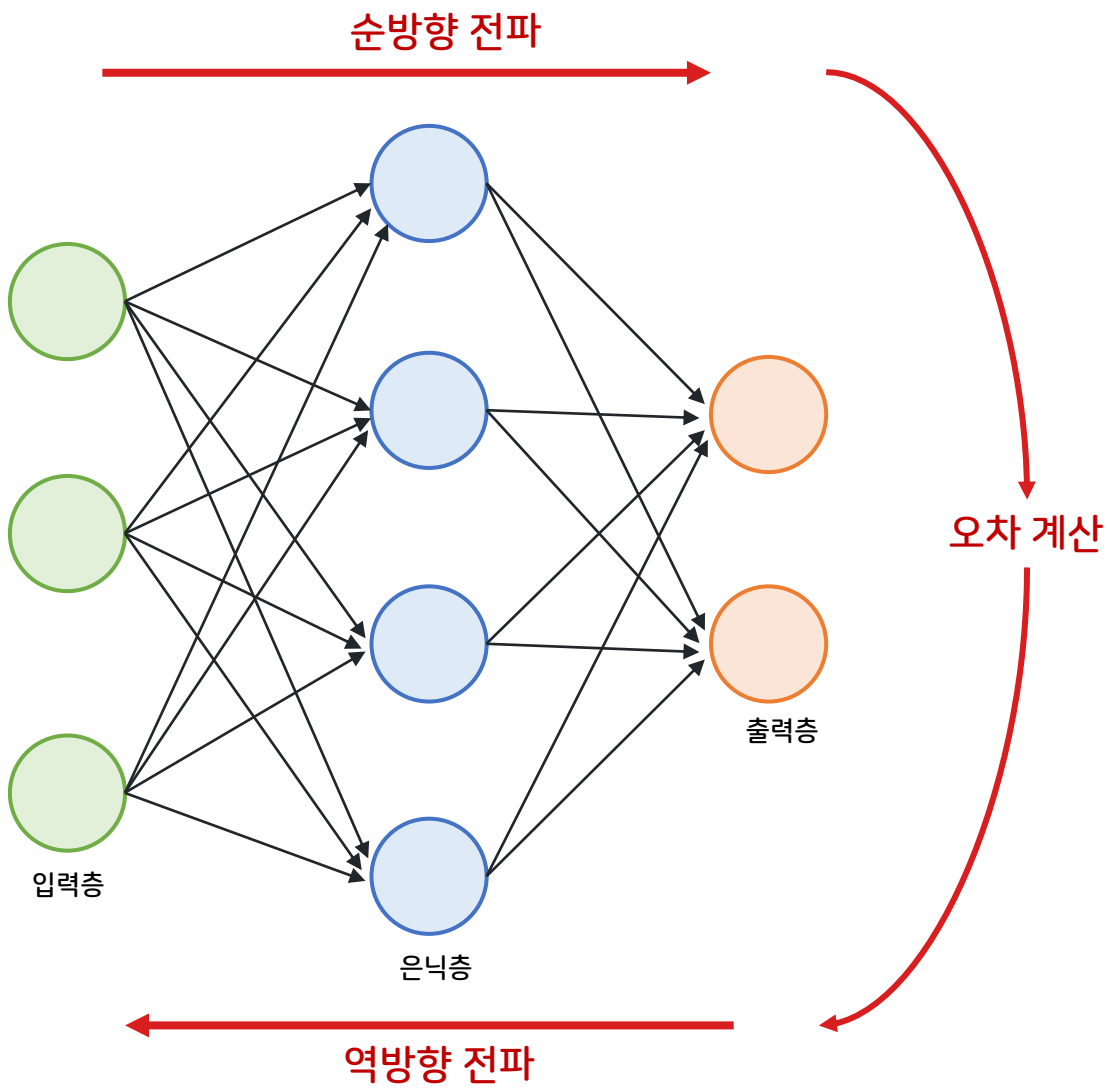


https://blog.naver.com/PostView.nhn?blogId=samsjang&logNo=221030487369&parentCategoryNo=&categoryNo=87&viewDate=&isShowPopularPosts=true&from=search

입력층과 출력층 사이에 은닉층(Hidden Layer)을 가지고 있는 신경망

단층 퍼셉트론으로 학습할 수 없었던 XOR 문제도 학습 가능

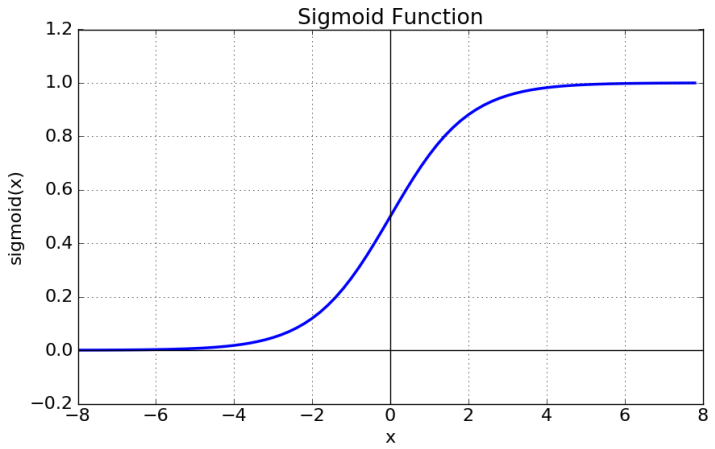
다층 퍼셉트론 학습 알고리즘 : 역전파 알고리즘



입력이 주어지면 순방향으로 계산하여 출력을 계산한 후, 실제 출력과 우리가 원하는 출력 간의 오차를 계산하여
이 오차를 역방향으로 전파하면서 오차를 줄이는 방향으로 가중치를 변경함

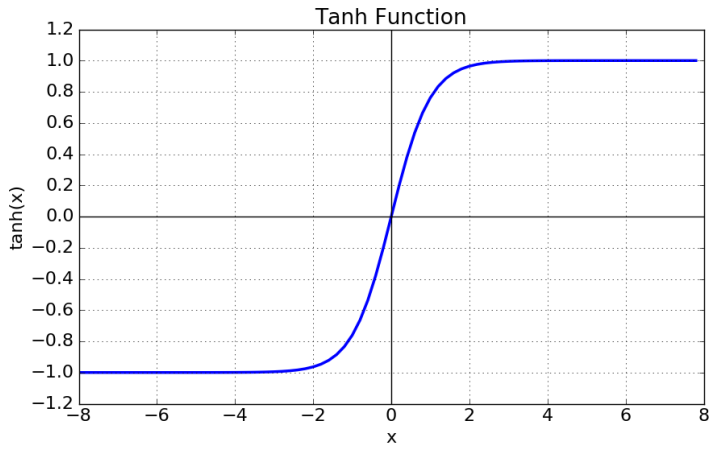
다층 퍼셉트론 활성화 함수 : 비선형 함수

Sigmoid 함수



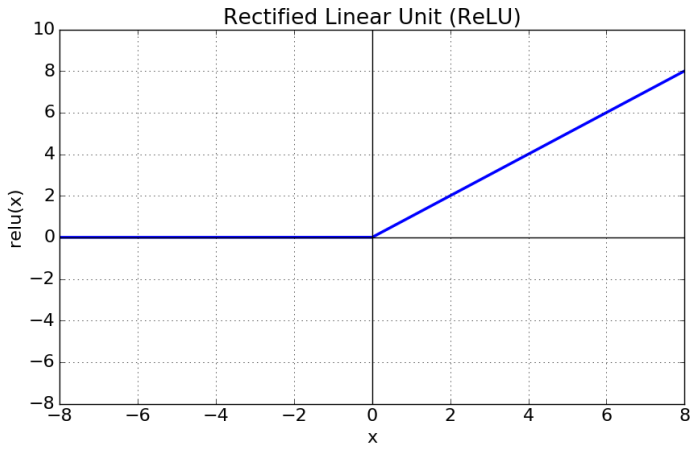
$$f(x) = \frac{1}{1 + e^{-x}}$$

TanH 함수



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

ReLU 함수



$$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

선형함수를 사용하면 은닉층을 아무리 많이 두어도 성능이 전혀 향상되지 않음

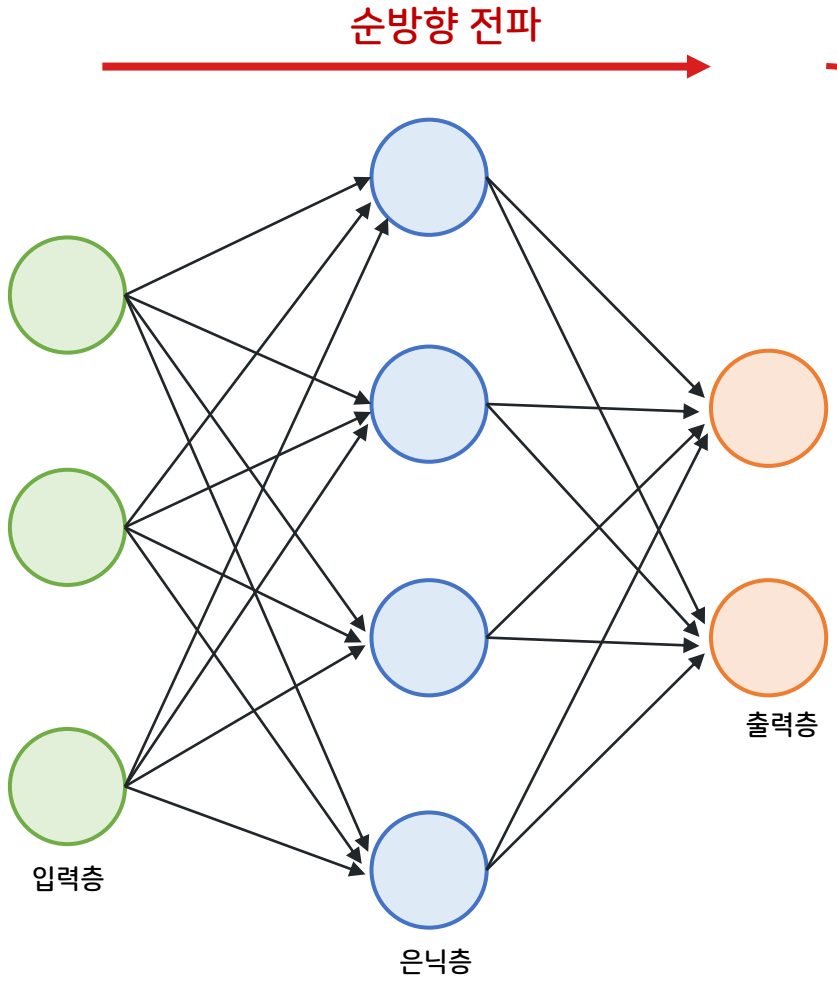
따라서 다층 퍼셉트론의 성능을 개선하기 위해서는 반드시 비선형 함수를 사용하여야 함

02

역전파 학습 알고리즘

역전파 학습 알고리즘 : 손실 함수란?

① 입력이 주어지면 순방향으로 계산하여 출력을 계산한 후

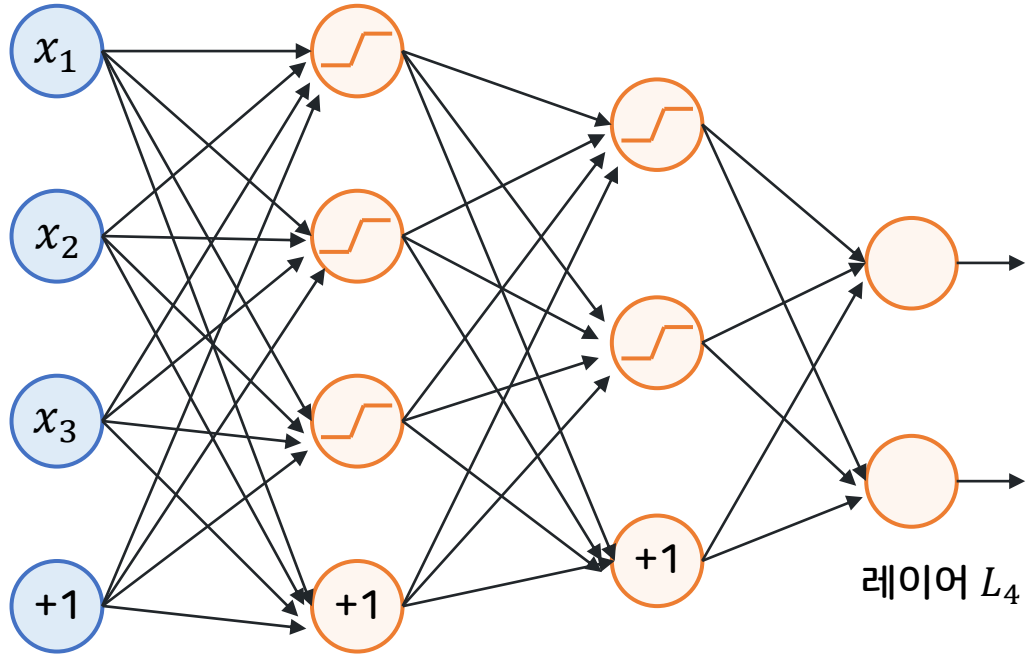


③ 오차를 역방향으로 전파하면서 오차를 줄이는 방향으로 가중치를 변경함

역전파 학습 알고리즘 : 손실 함수란?

입력 이미지

OR



고양이일 때만 1.0
그렇지 않으면 0.0

만약
입력 이미지가 고양이라면
올바른 출력은 (1,0)

강아지일 때만 1.0
그렇지 않으면 0.0

만약
입력 이미지가 강아지라면
올바른 출력은 (0,1)

1000장의 이미지를 가지고 학습을 수행 ▶ 고양이 사진 출력 (1,0) (정답) ▶ 이어서 강아지 사진 출력(1,0) (오답)

강아지 이미지에 대한 올바른 출력은 (0,1) (목표 출력값) 실제 출력은 (1,0) (실제 출력값) ▶ 오차는 (0-1)**2 + (1-0)**2 = 2.0

역전파 학습 알고리즘 : 손실 함수

$$E(w) = \frac{1}{2}\sum_{i}(t_i - o_i)^2$$

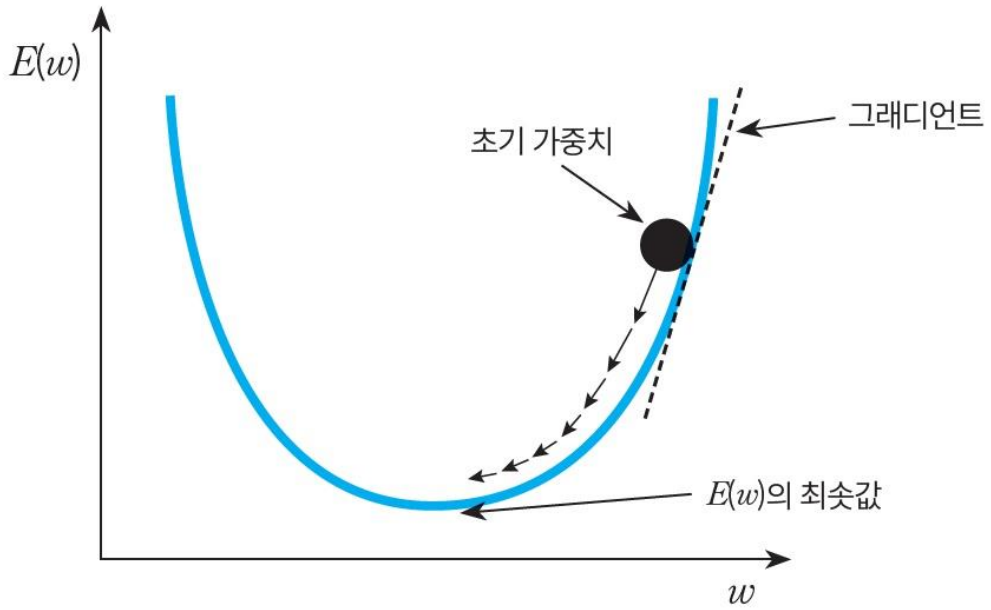
w: 가중치, i: 출력 노드의 번호, t : 훈련 샘플의 목표 출력값, o : 실제 출력값

역전파 알고리즘은 손실 함수 값을 줄이는 문제를 최적화 문제로 접근

$E(w)$를 최소로 만드는 가중치 w를 찾기

만약 손실 함수 값이 0이 되었다면 입력을 완벽하게 분류한 것

역전파 학습 알고리즘 : 경사하강법



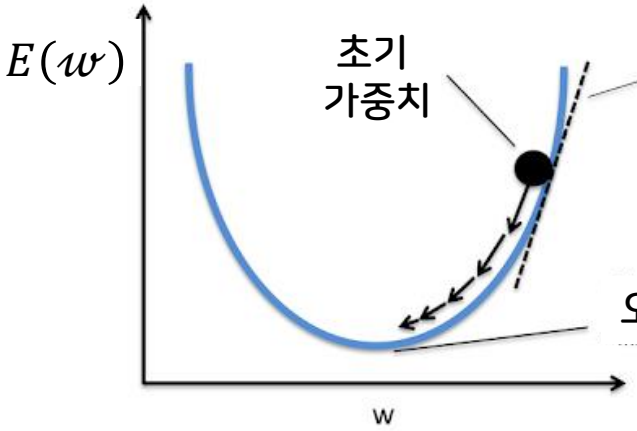
그래디언트는 접선의 기울기로 이해해도 됩니다. 접선의 기울기가 양수이면 반대로 w를 감소시킵니다.

그림 13-7 경사 하강법

역전파 학습 알고리즘 : 경사하강법

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

$E(w)$를 최소로 만드는 가중치 w를 찾기



그래디언트(접선의 기울기) $\frac{\partial E}{\partial w}$

$$w = w - \eta \frac{\partial E}{\partial w}$$

오차함수의 최솟값

경사하강법을 이용한 가중치 변경

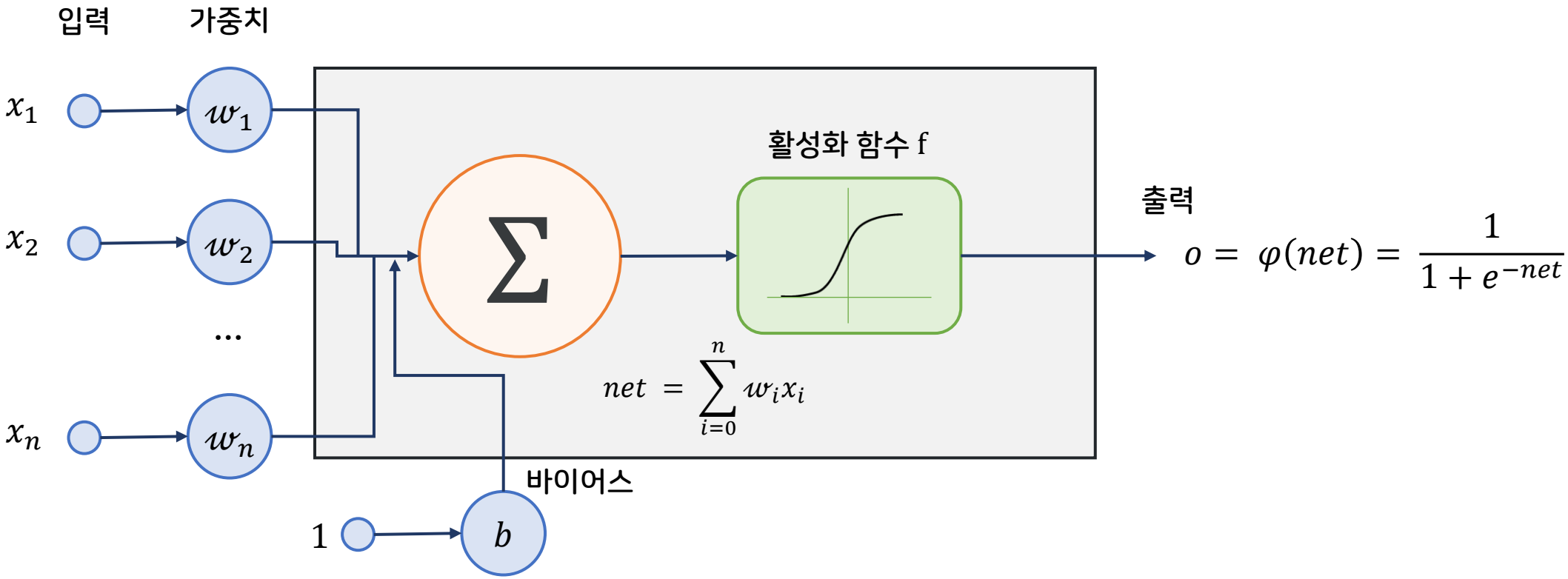
$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w}$$

$w(t)$: 현재의 가중치, η :학습률(0~1 사이 값), $w(t+1)$: 변경 후 가중치

03

역전파 알고리즘의 유도

역전파 알고리즘 상세 설명



활성화 함수

Sigmoid 함수

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

손실 함수

$$E = \frac{1}{2}\sum_{j=1}^{m}(t_j - o_j)^2$$

m : 출력 노드 개수, t: 목표 출력, o : 실제 출력

편미분 값

$$\frac{\partial E}{\partial w_{ij}}$$

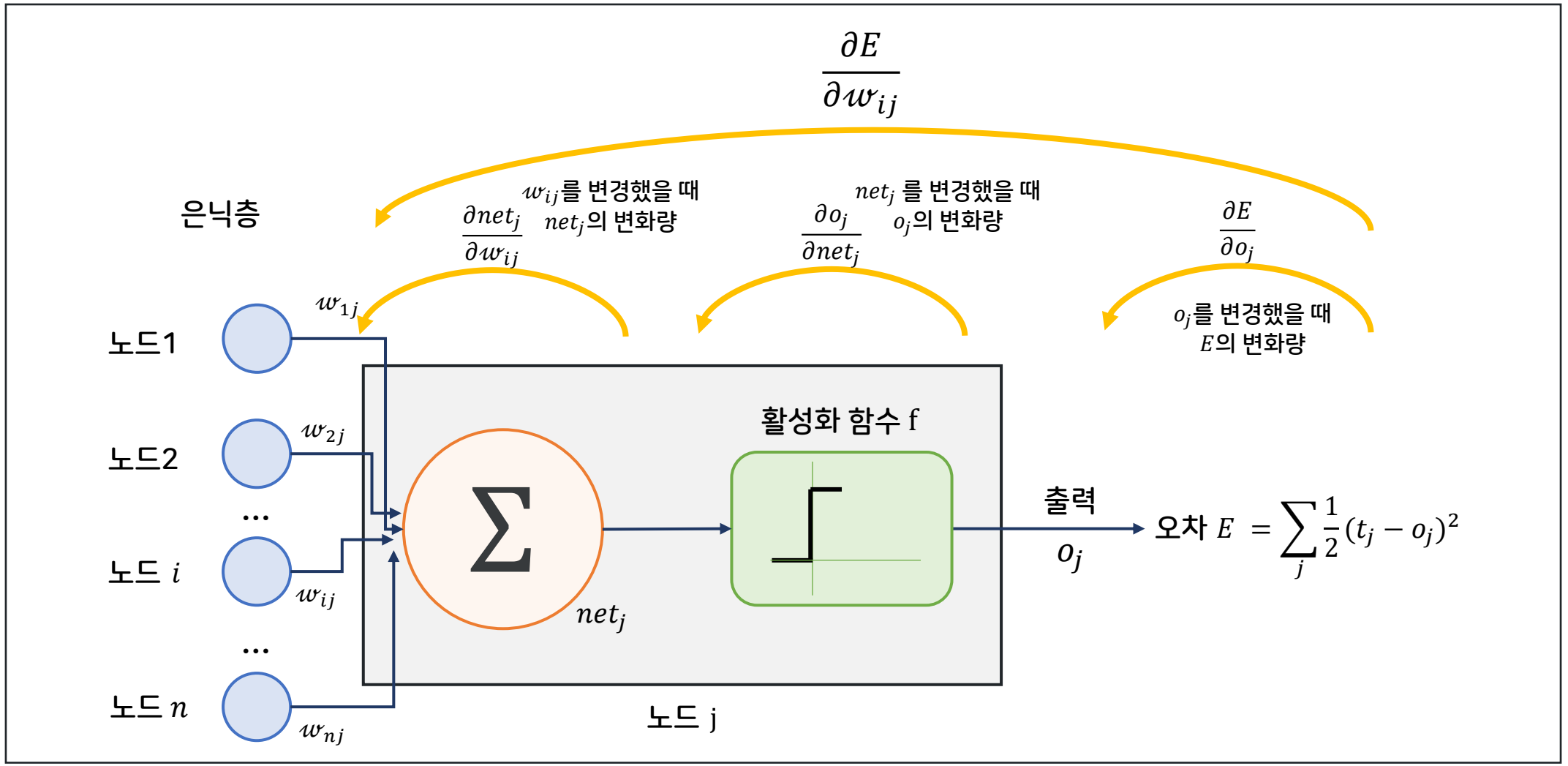
w_{ij} : 노드 i와 노드j를 연결하는 선의 가중치

w_{ij}가 변했을 때 손실함수 E가 얼마나 변하는가?

체인룰 사용

$$\frac{\partial E}{\partial w_{ij}} \xrightarrow{\text{체인룰}} \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

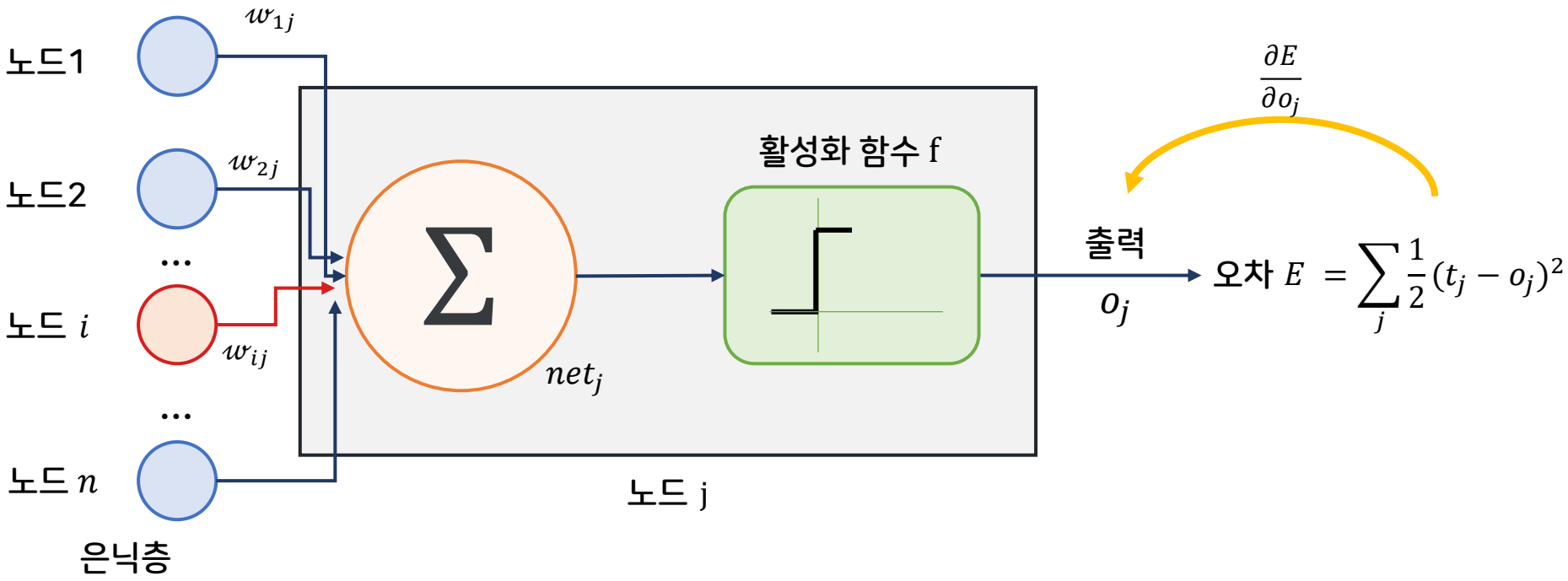
노드 j가 출력층에 있는 노드



출력 노드의 경우

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

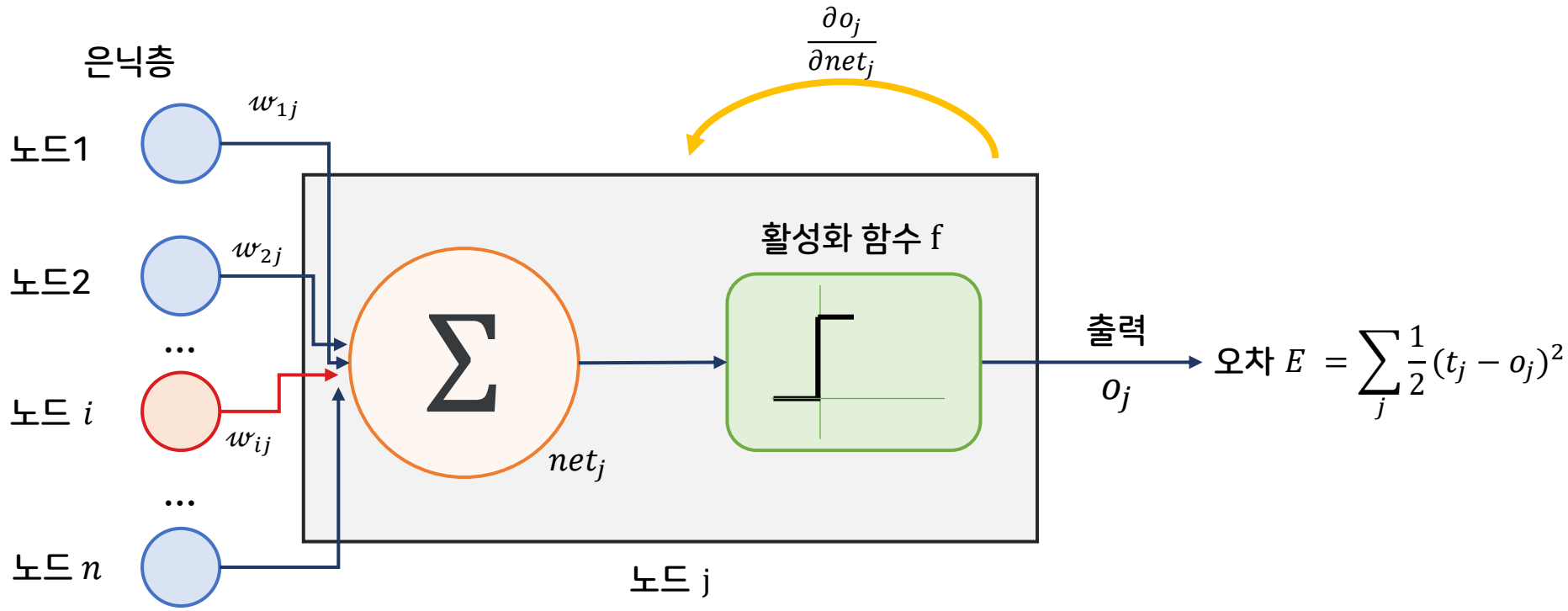
① 노드의 출력값 변화에 따른 오차의 변화율



j번째 노드가 출력 노드라면, $\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \sum_k \frac{1}{2}(t_j - o_j)^2 = o_j - t_j$

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

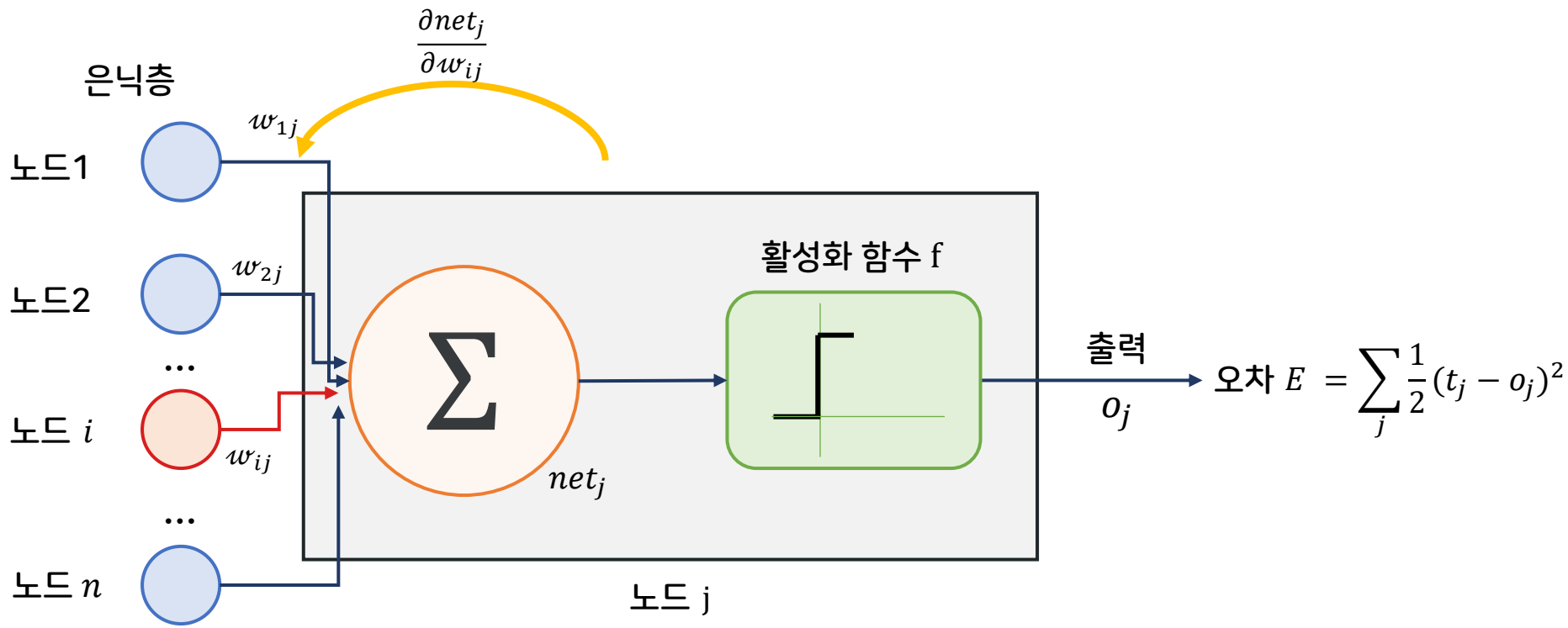
② 입력합의 변화에 따른 노드 j의 출력 변화율



$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \varphi(net_j)}{\partial net_j} = \varphi(net_j)\left(1 - \varphi(net_j)\right)$$

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

③ 가중치의 변화에 따른 net_j의 변화율



$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=0}^{n} w_{kj} O_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_j = o_i$$

노드 j가 출력층에 있는 노드

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = (o_j - t_j)\varphi(net_j)\left(1 - \varphi(net_j)\right)o_i$$

총정리

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

δ_j 는 신경망 계층에 따라 다음과 같이 구분하여서 계산

$$\delta_j = \begin{cases} \varphi'(net_j)(o_j - t_j) & \leftarrow j\text{가 출력 계층이면} \\ \left(\sum_k w_{jk}\delta_k\right)\varphi'(net_j) & \leftarrow j\text{가 은닉 계층이면} \end{cases}$$

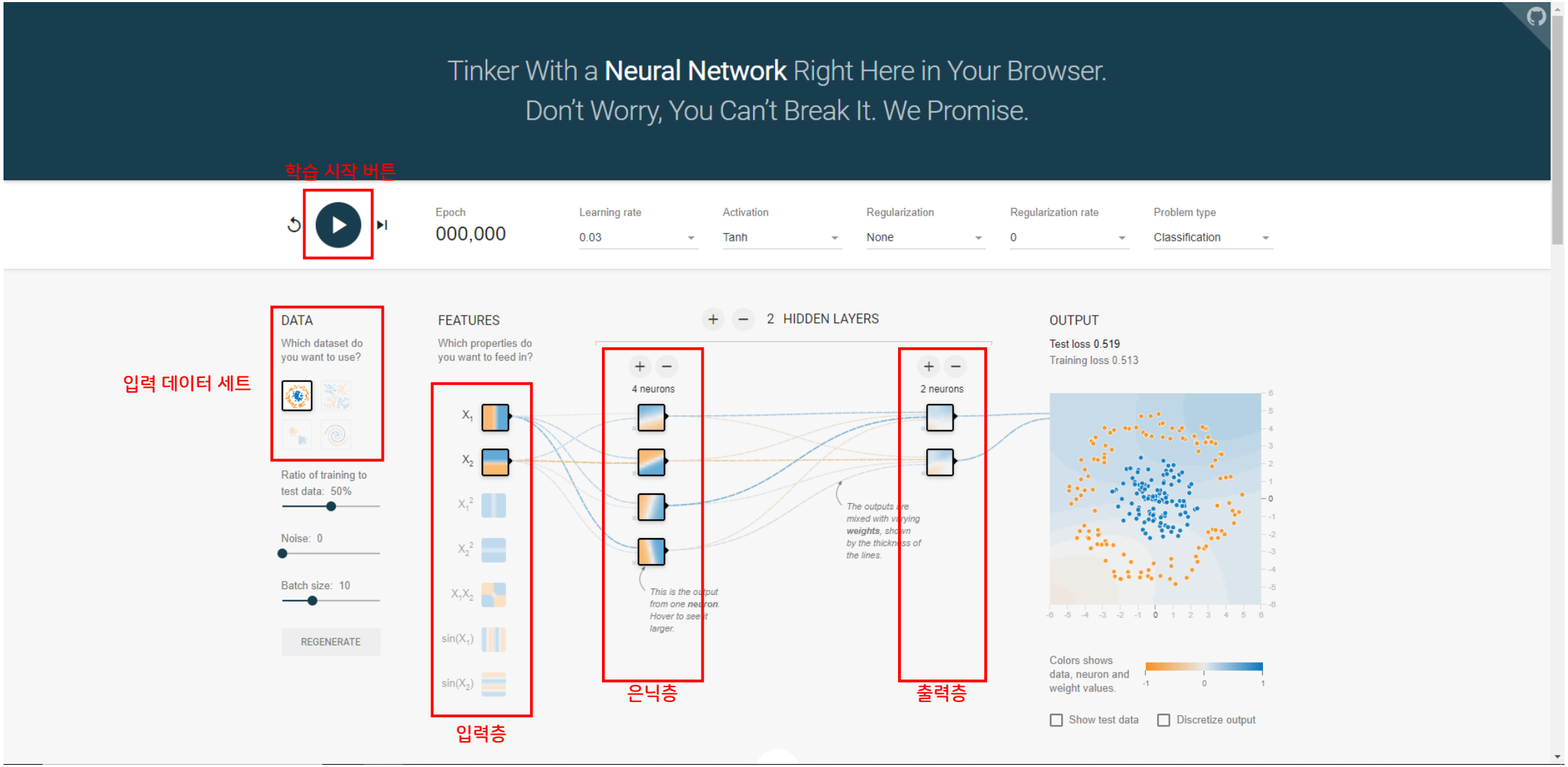
가중치의 변화량 Δw_{ij}은 다음식으로 계산

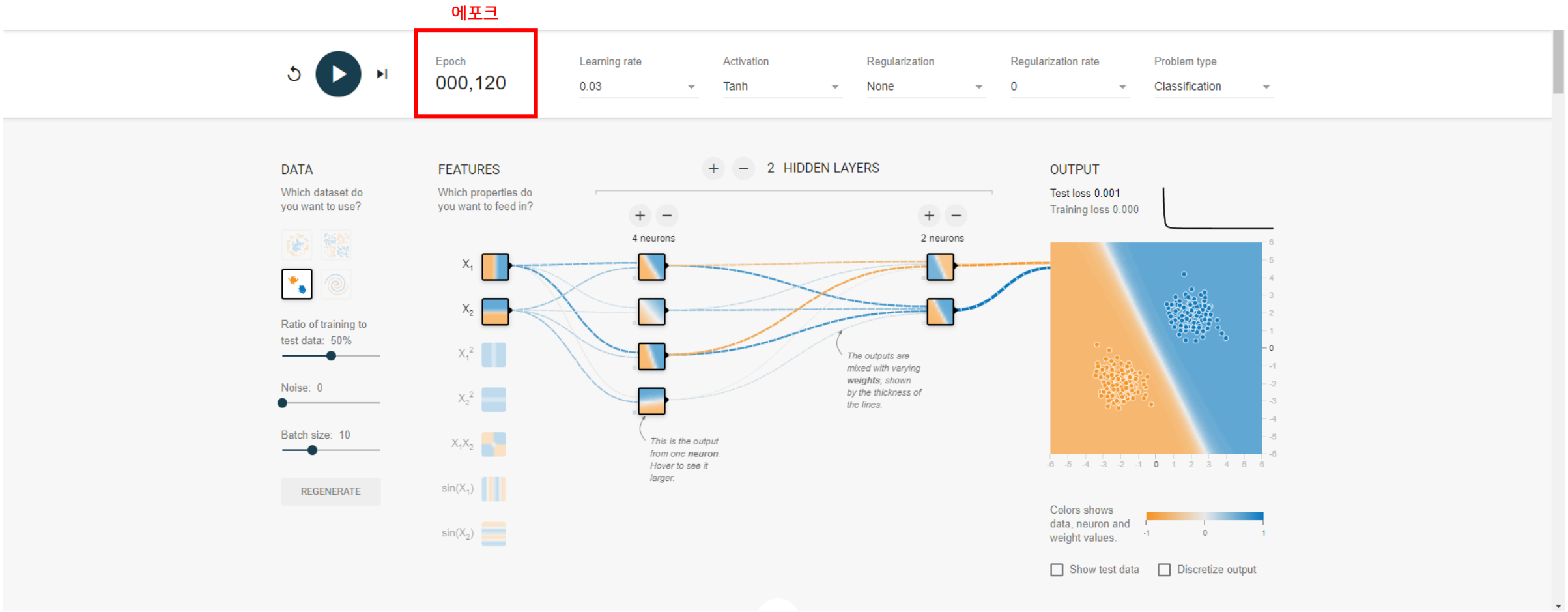
$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} - \eta o_i \delta_j$$

04

구글의 플레이그라운드 실습

https://playground.tensorflow.org

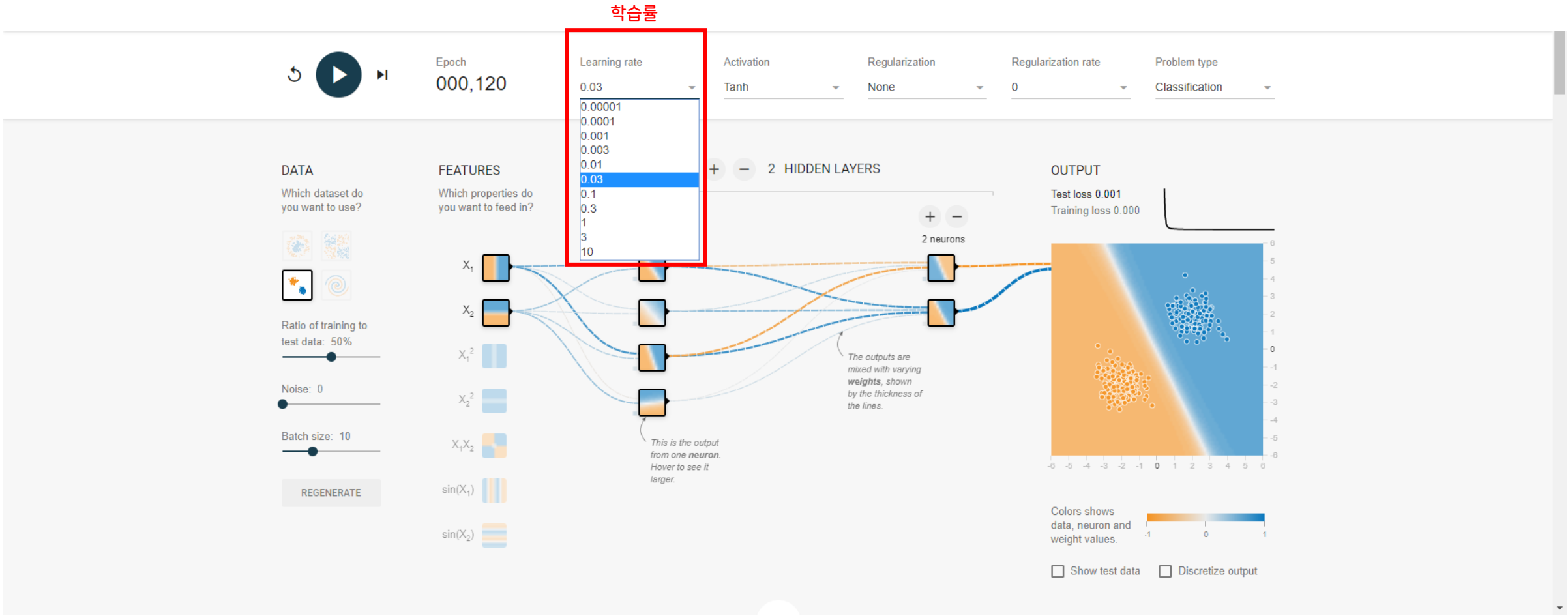




에포크

준비된 학습 예제를 한번 반복

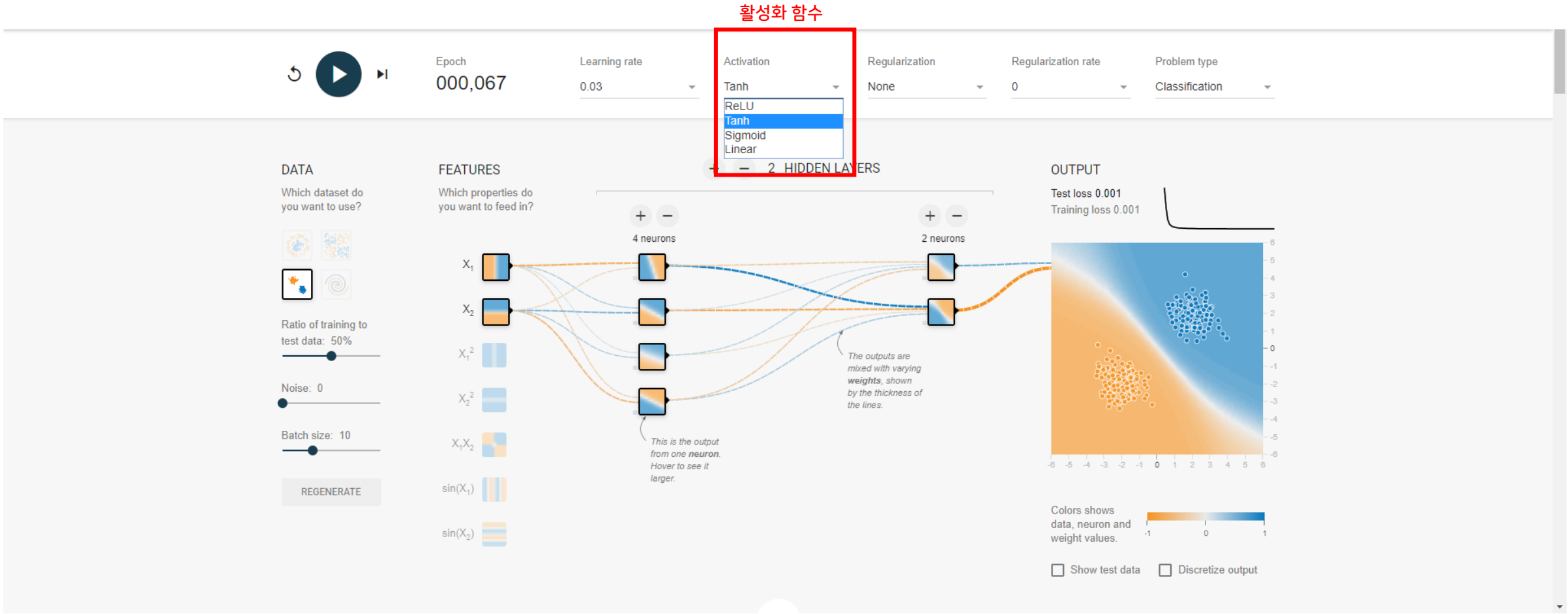
학습 예제 세트에 대해 학습을 한 번 실시할 때마다 에포크 수 증가



학습률

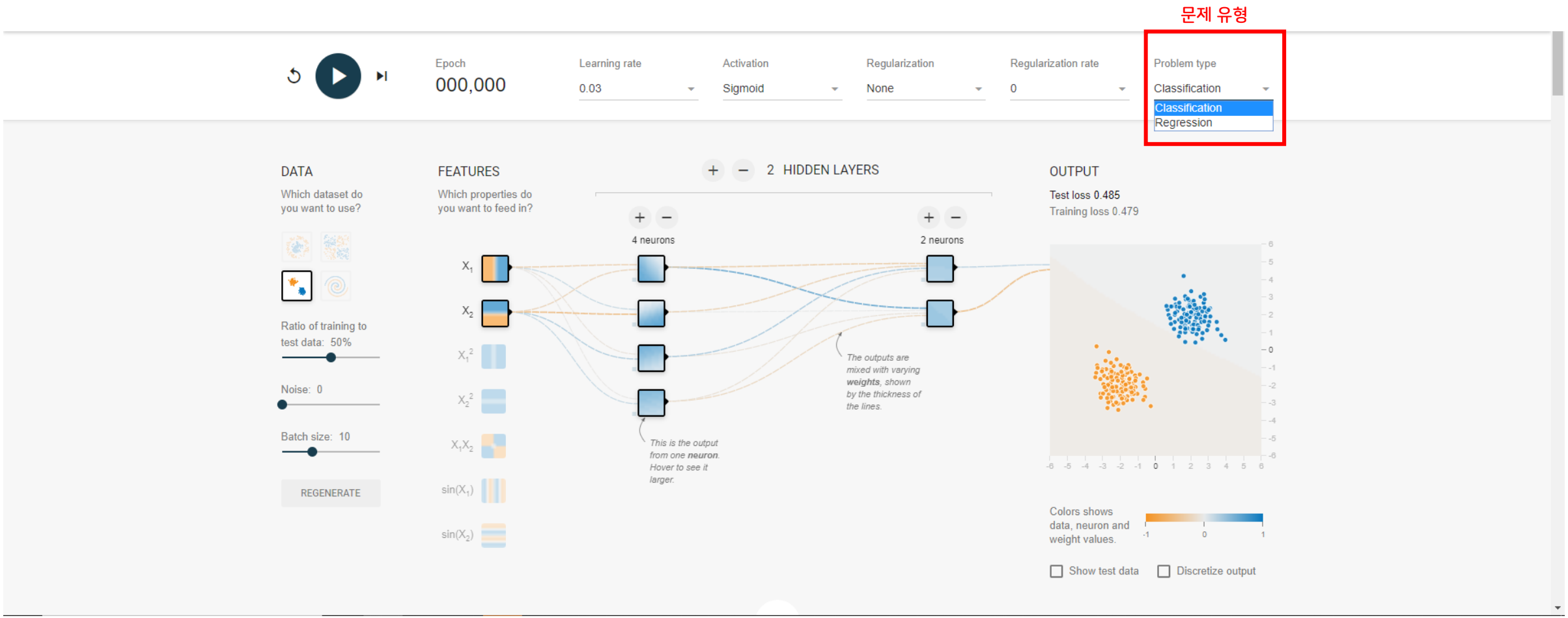
학습 속도 결정

숫자가 커질수록 학습속도 ↑



활성화 함수

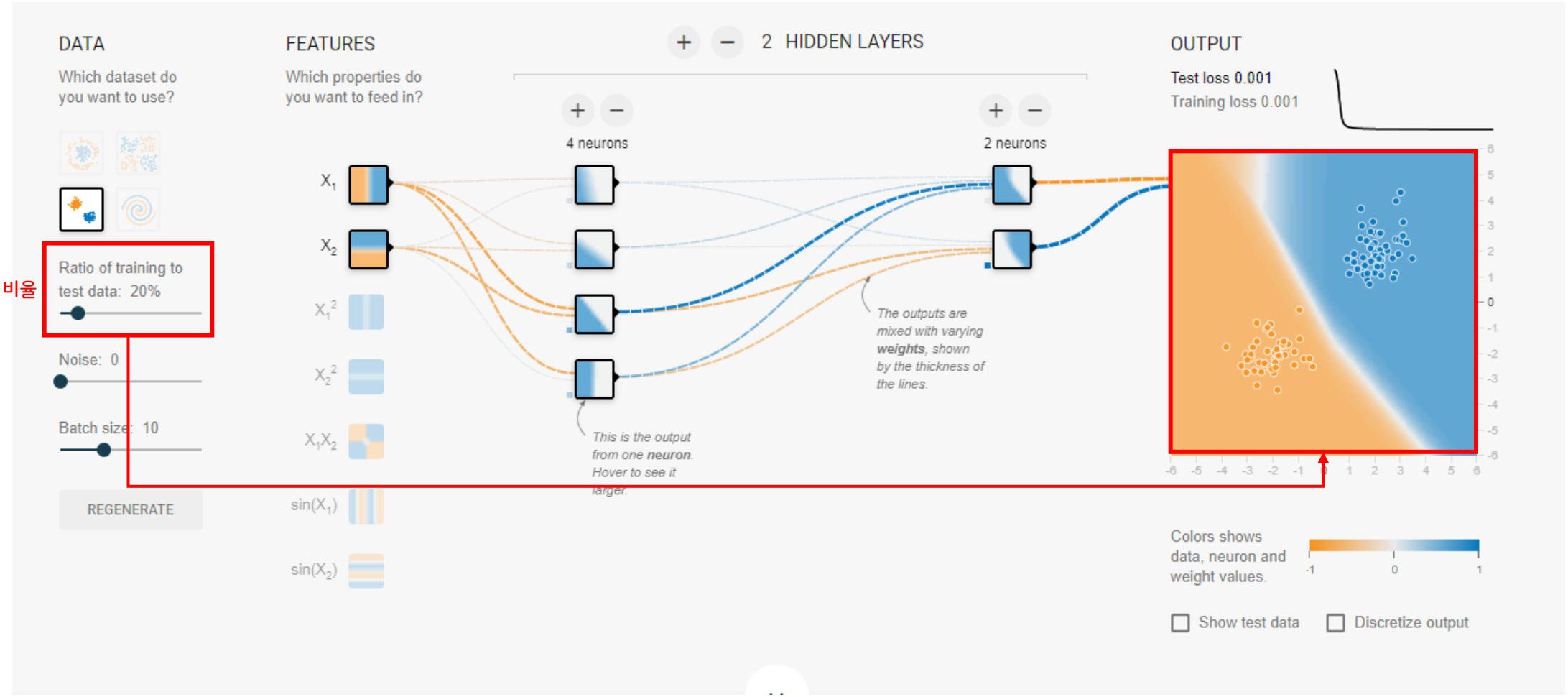
활성화 함수마다 학습되는 모양이 조금씩 상이함



문제 유형

Classification : 분류, Regression : 회귀

분류는 입력데이터 유형 4개, 회귀는 2개

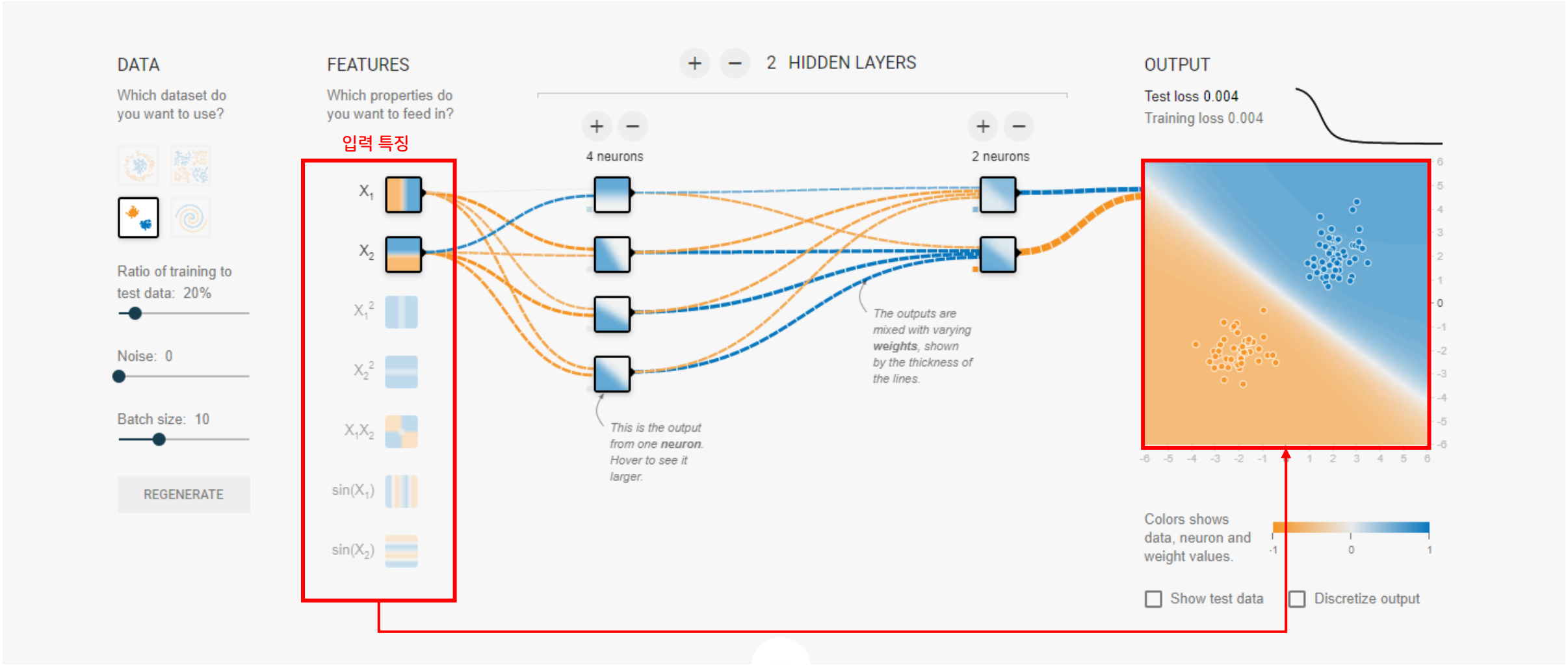


학습 데이터 비율

학습데이터 비율

전체 데이터 중 몇 %를 학습데이터로 사용할 것인지 결정

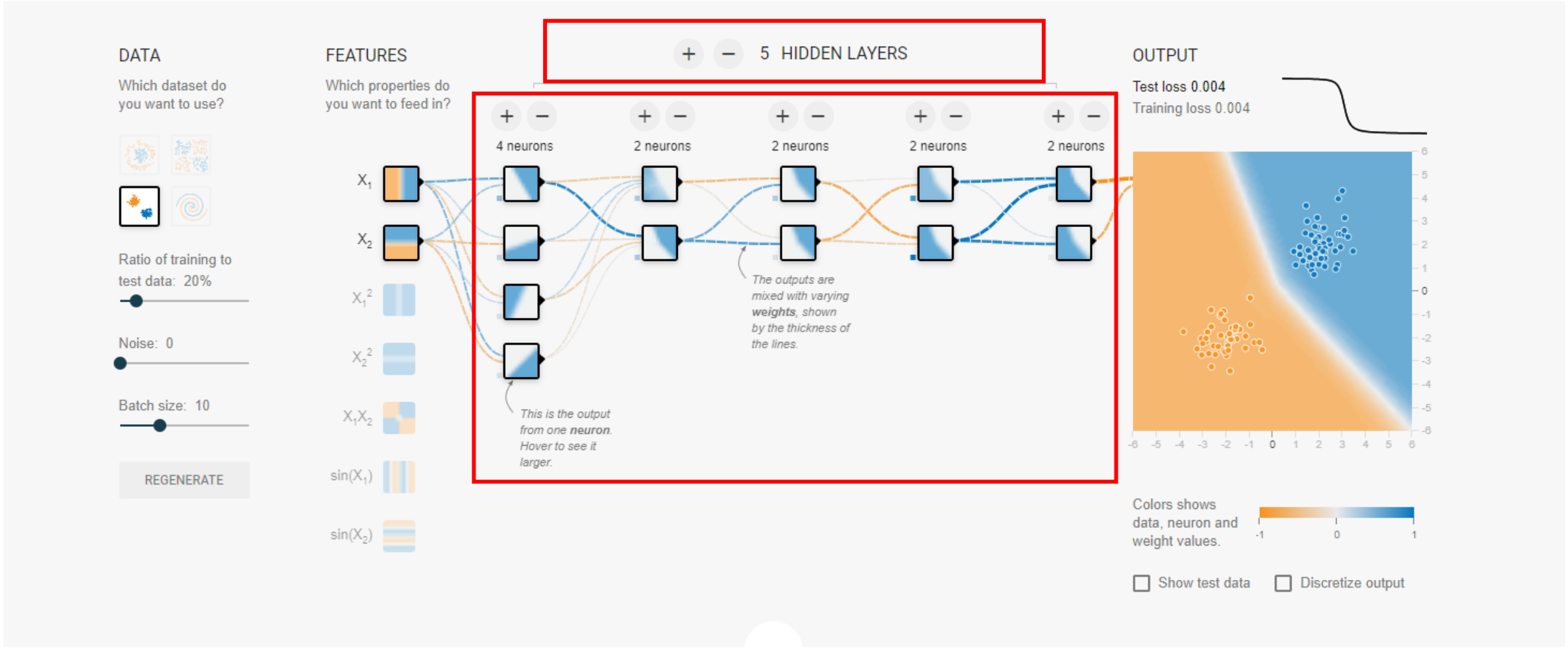
비율을 낮추면 점의 개수가 적어짐



입력 특징

x_1: 데이터 세트에 있는 점들의 x 좌표, x_2: 데이터 세트에 있는 점들의 y 좌표

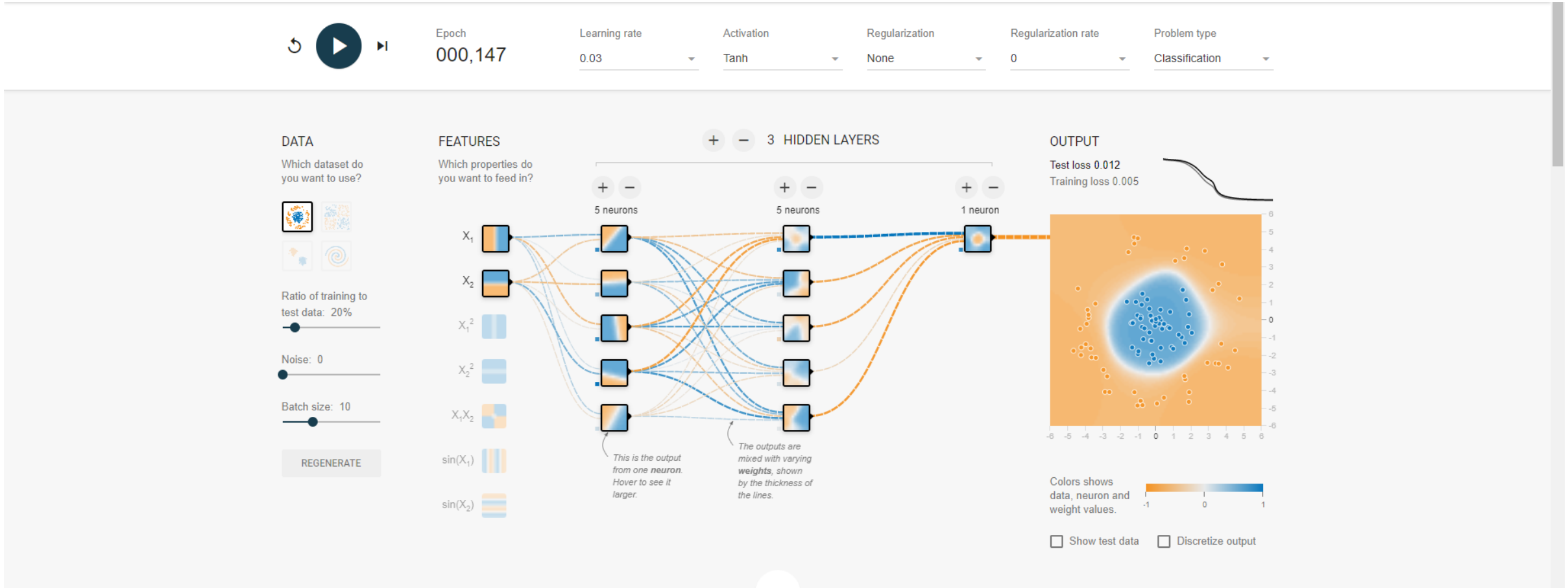
입력 특징을 달리하여 출력 모양을 다르게 할 수 있음



은닉층 추가

은닉층을 추가하면 훨씬 더 부드러운 판단 경계선을 가짐

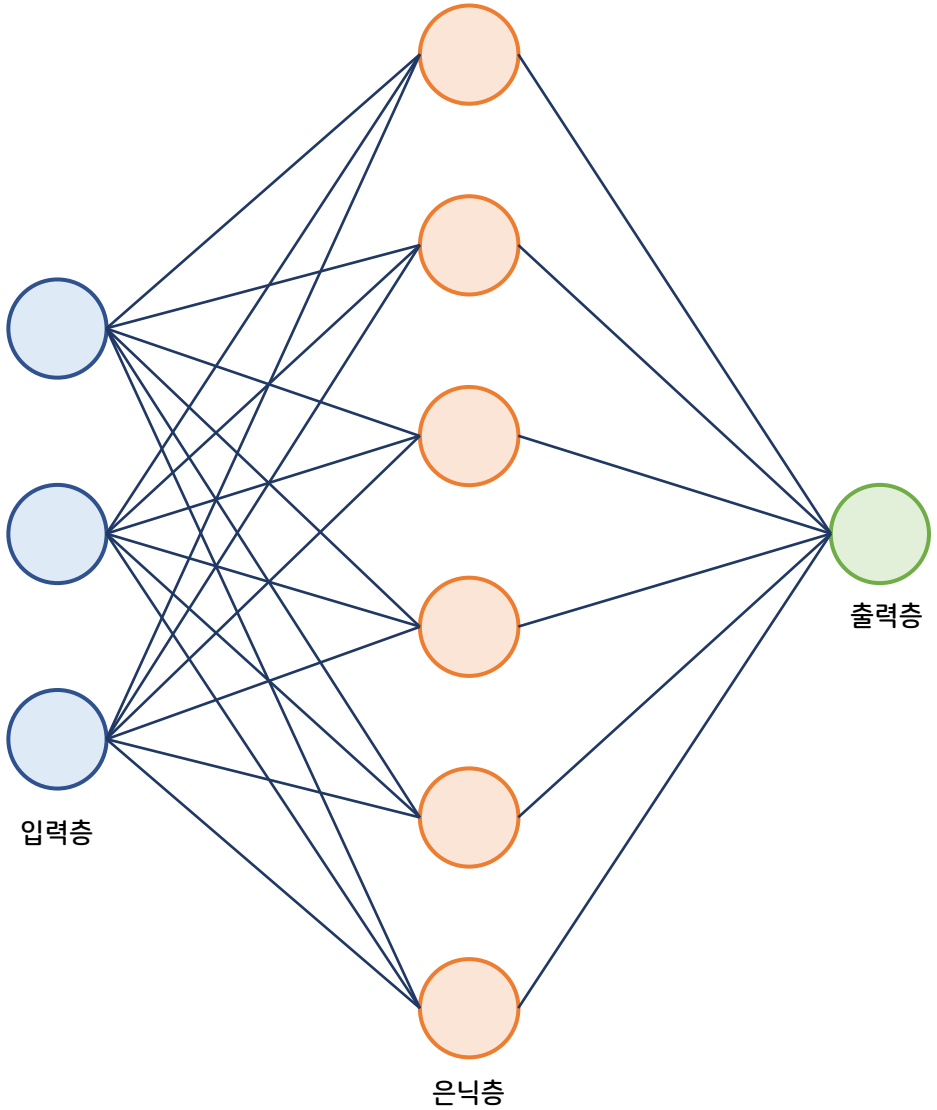
은닉층이 없으면 학습이 제대로 완료되지 않음


Epoch
000,147
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 20%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
3 HIDDEN LAYERS
5 neurons
5 neurons
1 neuron
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.012
Training loss 0.005
Colors shows data, neuron and weight values.
Show test data
Discretize output

05

넘파이를 이용하여 MLP 구현

Numpy를 이용한 MLP



행렬을 사용하여 구현

```python
import numpy as np

#시그모이드 함수
def actf(x):
  return 1/(+np.exp(-x))

#시그모이드 함수의 미분값
def actf_deriv(x):
  return x*(1-x)

#XOR연산을 위한 4행 * 2열의 입력 행렬
#마지막 열은 바이어스를 나타냄
X = np.array([[0,0,1], [0,1,1], [1,0,1], [1,1,1]])

#XOR 연산을 위한 4행 * 1열의 목표 행렬
y = np.array([[0],[1],[1],[0]])

np.random.seed(5)

inputs = 3     #입력층의 노드 개수
hiddens = 6    #은닉층의 노드 개수
outputs = 1    #출력층의 노드 개수

#가중치를 -1.0에서 1.0 사이의 난수로 초기화한다
weight0 = 2*np.random.random((inputs,hiddens))-1
weight1 = 2*np.random.random((hiddens,outputs))-1
```

```python
#반복
for i in range(10000):

  #순방향 계산
  layer0=X                        #입력을 layer0에 대입
  net1 = np.dot(layer0, weight0) #행렬의 곱을 계산
  layer1 = actf(net1)             #활성화 함수 적용
  layer1[:,-1]=1.0                #마지막 열은 바이어스를 나타냄. 1.0으로 만든다

  net2 = np.dot(layer1, weight1) #행렬의 곱을 계산
  layer2 = actf(net2)             #활성화 함수 적용

  #출력층에서의 오차를 계산
  layer2_error = layer2-y

  #출력층에서의 델타값을 계산
  layer2_delta = layer2_error*actf_deriv(layer2)

  #은닉층에서 오차 계산
  #여기서 T는 행렬의 전치를 의미
  #역방향으로 오차를 전파할 때는 반대방향이므로 행렬이 전치되어야 함
  layer1_error=np.dot(layer2_delta,weight1.T)

  #은닉층에서의 델타를 계산
  layer1_delta=layer1_error*actf_deriv(layer1)

  #은닉층->출력층을 연결하는 가중치를 수정
  weight1 += -0.2*np.dot(layer1.T, layer2_delta)

  #입력층->은닉층을 연결하는 가중치를 수정
  weight0 += -0.2*np.dot(layer0.T, layer1_delta)

print(layer2)
```

```
[[0.02449929]
 [0.98003059]
 [0.98066189]
 [0.02151195]]
```

06

구글의 텐서플로우

텐서플로우(TensorFlow)

딥러닝 프레임워크의 일종

텐서플로우 위에서 고수준 API를 제공하는Keras 라이브러리를 사용하여 구현

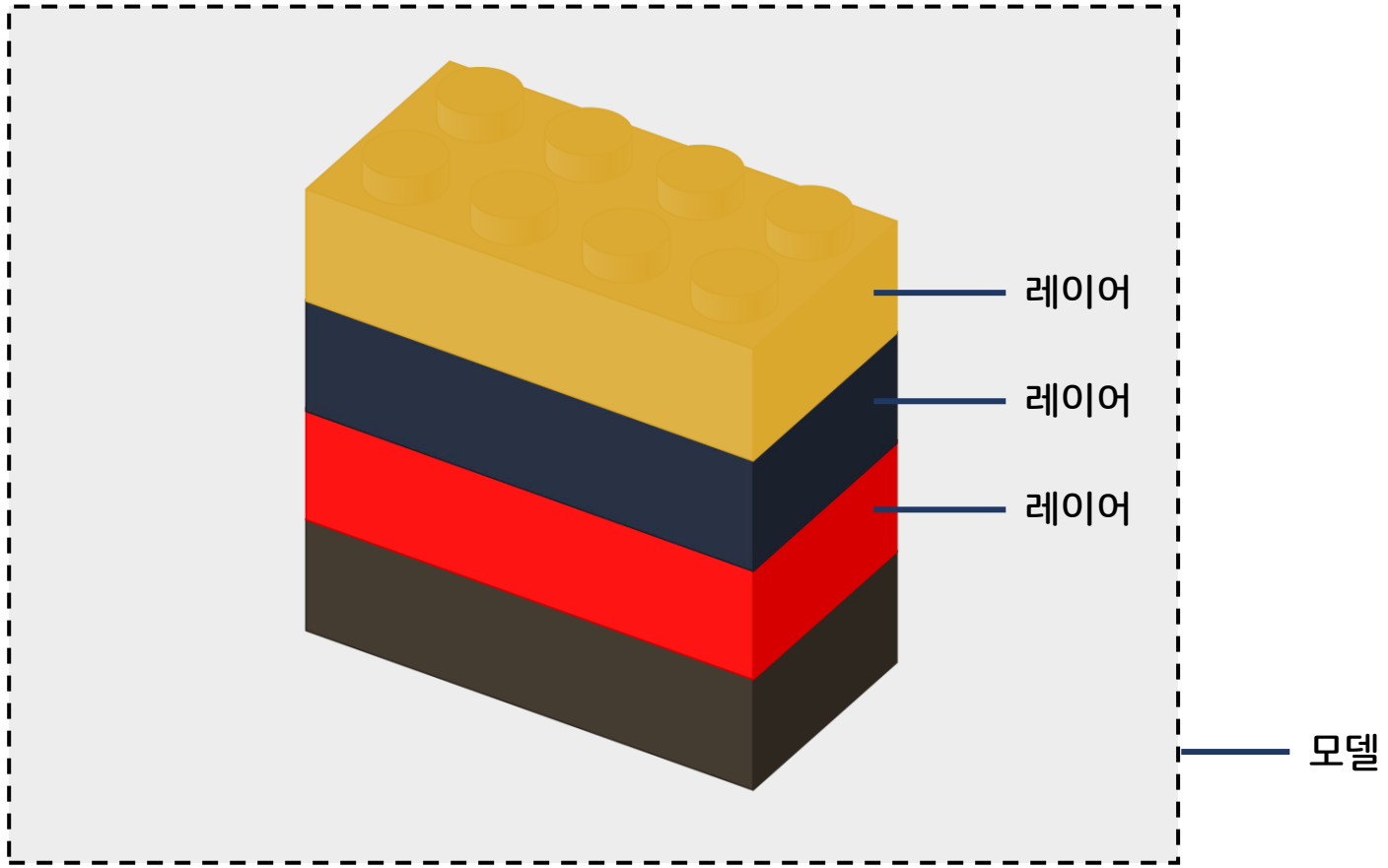
Tensor : 다차원 배열을 나타내는 용어로 스칼라, 벡터, 행렬, 텐서 지원

Flow : 데이터플로우를 의미

* Anaconda Navigator Version 5.2 에 TensorFlow, Keras 설치 필요

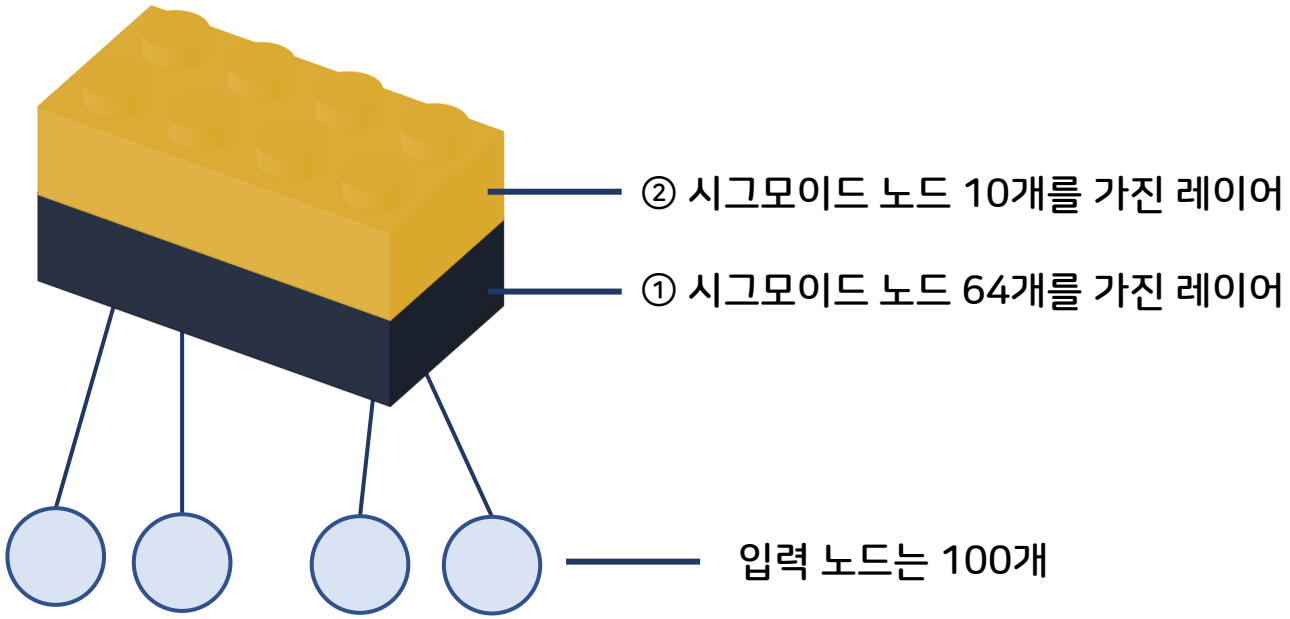
Keras 라이브러리

선형 스택 모델 생성




```
from tf.keras.models import Sequential

model = Sequential()
```

Keras 라이브러리



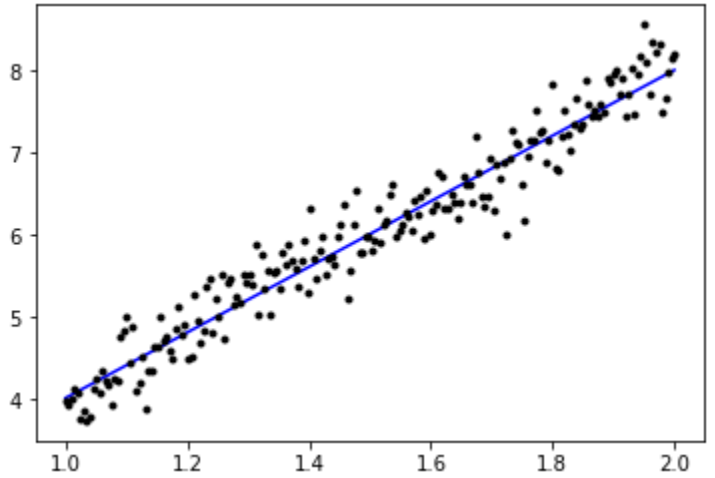

```
from tf.keras.models import Sequential

model = Sequential() #선형 스택 모델 생성

①model.add(Dense(units=64, activation='sigmoid', input_dim=100)) #모델에 레이어 쌓기, 활성화함수 sigmoid, 입력개수 100개
②model.add(Dense(units=10, activation='sigmoid')) #노드의 개수 10개인 완전 연결된 레이어 추가
```

Keras 라이브러리

```
from tf.keras.models import Sequential

model = Sequential() #선형 스택 모델 생성

model.add(Dense(units=64, activation='sigmoid', input_dim=100)) #모델에 레이어 쌓기, 활성화함수 sigmoid, 입력개수 100개
model.add(Dense(units=10, activation='sigmoid')) #노드의 개수 10개인 완전 연결된 레이어 추가

#모델이 완성되면 compile()함수를 이용하여 학습과정을 구성/ loss='mse'는 손실함수 평균제곱오차, SGD는 확률적 경사하강법
model.compile(loss='mse', optimizergd, metrics=['accuracy'])

#필요시 optimizer 세부 조정 가능 / 학습률: 0.01 / 모멘텀 : 0.9
#Model.compile(loss='mse', optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9))

#모델이 만들어지면 학습 진행. 학습은 fit() 함수호출
model.fit(X, y, epochs=5, batch_size=32)

#모델의 성능은 evaluate()함수로 평가
loss_and_metrics=model.evaluate(X, y, batch_size=128)

#새 데이터에 대한 예측은 predict()로 테스트 가능
classes = model.predict(new_X, batch_size=128)
```

Keras 예제 #1

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

#가상적인 데이터 생성
X = data =np.linspace(1,2,200)    # 시작값=1, 종료값=2, 개수=200
y = X*4 + np.random.randn(200)*0.3 #x를 4배로 하고 편차 0.3정도의 가우시안 잡음 추가

model = tf.keras.models.Sequential() #선형 스택 모델 생성
model.add(tf.keras.layers.Dense(1, input_dim = 1, activation= ' linear ' )) #선형함수 사용

model.compile(optimizer= ' sgd ' , loss= ' mse ' , metrics=[ ' mse ' ]) #확률적 경사하강법, 손실함수는 평균제곱오차, 에포크는 30
model.fit(X,y,batch_size=1, epochs=30) #학습 진행

predict = model.predict(data) #새 데이터 예측

plt.plot(data,predict,  ' b ' , data, y,  ' k.')  # 첫 번째 그래프는 파란색 마커로
plt.show() # 두 번째 그래프는 검정색 점으로
```



Keras 예제 #2

```
import tensorflow as tf
import numpy as np

X = np.array([[0,1],[0,1],[1,0],[1,1]]) #XOR연산을 위한 입력 행렬
y = np.array([[0],[1],[1],[0]]) #XOR연산을 위한 목표 행렬

model = tf.keras.models.Sequential() #선형 스택 모델 생성
model.add(tf.keras.layers.Dense(2, input_dim=2, activation='sigmoid')) #모델에 레이어 쌓기, 활성화함수는 sigmoid
model.add(tf.keras.layers.Dense(1, activation='sigmoid')) #모델에 레이어 쌓기, 활성화 함수는 sigmoid

sgd = tf.keras.optimizers.SGD(lr=0.1) #확률적 경사 하강법 학습률 0.1로 설정
model.compile(loss= ' mean_squared_error ' , optimizer=sgd) #학습 과정 구성, 손실함수는 평균 제곱 오차

model.fit(X,y, batch_size=1, epochs=1000) #학습진행 에포크는 1000
print(model.predict(X)) #실행 출력
```

```
4/4 [==============================] - 0s 748us/step - loss: 0.1487
Epoch 998/1000
4/4 [==============================] - 0s 748us/step - loss: 0.1485
Epoch 999/1000
4/4 [==============================] - 0s 998us/step - loss: 0.1484
Epoch 1000/1000
4/4 [==============================] - 0s 748us/step - loss: 0.1484
[[0.4557345 ]
 [0.4557345 ]
 [0.8708291 ]
 [0.21653537]]
```

THANK YOU