4월14일 수업 코드

epsilon = 0.0000001

def perceptron(x1, x2):

w1, w2, b = 1.0, 1.0, -1.5

sum = x1\*w1+x2\*w2+b

if sum > epsilon : # 부동소수점 오차를 방지하기 위하여

return 1

else :

return 0

print(perceptron(0, 0))

print(perceptron(1, 0))

print(perceptron(0, 1))

print(perceptron(1, 1))

0

0

0

1

import numpy as np

epsilon = 0.0000001

def perceptron(x1, x2):

X = np.array([x1, x2])

W = np.array([1.0, 1.0])

B = -1.5

sum = np.dot(W, X)+B

if sum > epsilon :

return 1

else :

return 0

print(perceptron(0, 0))

print(perceptron(1, 0))

print(perceptron(0, 1))

print(perceptron(1, 1))

0

0

0

1

import numpy as np

epsilon = 0.0000001 # 부동소수점 오차 방지

def step\_func(t): # 퍼셉트론의 활성화 함수

if t > epsilon: return 1

else: return 0

X = np.array([ # 훈련 데이터 세트

[0, 0, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[0, 1, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[1, 0, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[1, 1, 1] # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

])

y = np.array([0, 0, 0, 1]) # 정답을 저장하는 넘파이 행렬

W = np.zeros(len(X[0]))

def perceptron\_fit(X, Y, epochs=10): # 퍼셉트론 학습 알고리즘 구현

global W

eta = 0.2 # 학습률

for t in range(epochs):

print("epoch=", t, "======================")

for i in range(len(X)):

predict = step\_func(np.dot(X[i], W))

error = Y[i] - predict # 오차 계산

W += eta \* error \* X[i] # 가중치 업데이트

print("현재 처리 입력=",X[i],"정답=",Y[i],"출력=",predict,"변경된 가중치=", W)

print("================================")

def perceptron\_predict(X, Y): # 예측

global W

for x in X:

print(x[0], x[1], "->", step\_func(np.dot(x, W)))

perceptron\_fit(X, y, 6)

perceptron\_predict(X, y)

epoch= 0 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 0 변경된 가중치= [0. 0. 0.] 현재 처리 입력= [0 1 1] 정답= 0 출력= 0 변경된 가중치= [0. 0. 0.] 현재 처리 입력= [1 0 1] 정답= 0 출력= 0 변경된 가중치= [0. 0. 0.] 현재 처리 입력= [1 1 1] 정답= 1 출력= 0 변경된 가중치= [0.2 0.2 0.2] ================================ epoch= 1 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 1 변경된 가중치= [0.2 0.2 0. ] 현재 처리 입력= [0 1 1] 정답= 0 출력= 1 변경된 가중치= [ 0.2 0. -0.2] 현재 처리 입력= [1 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.2 0. -0.2] 현재 처리 입력= [1 1 1] 정답= 1 출력= 0 변경된 가중치= [0.4 0.2 0. ] ================================ epoch= 2 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 0 변경된 가중치= [0.4 0.2 0. ] 현재 처리 입력= [0 1 1] 정답= 0 출력= 1 변경된 가중치= [ 0.4 0. -0.2] 현재 처리 입력= [1 0 1] 정답= 0 출력= 1 변경된 가중치= [ 0.2 0. -0.4] 현재 처리 입력= [1 1 1] 정답= 1 출력= 0 변경된 가중치= [ 0.4 0.2 -0.2] ================================ epoch= 3 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.2] 현재 처리 입력= [0 1 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.2] 현재 처리 입력= [1 0 1] 정답= 0 출력= 1 변경된 가중치= [ 0.2 0.2 -0.4] 현재 처리 입력= [1 1 1] 정답= 1 출력= 0 변경된 가중치= [ 0.4 0.4 -0.2] ================================ epoch= 4 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.4 -0.2] 현재 처리 입력= [0 1 1] 정답= 0 출력= 1 변경된 가중치= [ 0.4 0.2 -0.4] 현재 처리 입력= [1 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.4] 현재 처리 입력= [1 1 1] 정답= 1 출력= 1 변경된 가중치= [ 0.4 0.2 -0.4] ================================ epoch= 5 ====================== 현재 처리 입력= [0 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.4] 현재 처리 입력= [0 1 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.4] 현재 처리 입력= [1 0 1] 정답= 0 출력= 0 변경된 가중치= [ 0.4 0.2 -0.4] 현재 처리 입력= [1 1 1] 정답= 1 출력= 1 변경된 가중치= [ 0.4 0.2 -0.4] ================================ 0 0 -> 0 0 1 -> 0 1 0 -> 0 1 1 -> 1

연습문제 5-b

from sklearn.linear\_model import Perceptron

# 샘플과 레이블이다.

X = [[0,0],[0,1],[1,0],[1,1]]

y = [0, 0, 0, 1]

# 퍼셉트론을 생성한다. tol는 종료 조건이다. random\_state는 난수의 시드이다.

clf = Perceptron(tol=1e-3, random\_state=0)

# 학습을 수행한다.

clf.fit(X, y)

# 테스트를 수행한다.

print(clf.predict(X))

[0 0 0 1]

from sklearn.linear\_model import Perceptron

# 샘플과 레이블이다.

X = [[160,55],[163,43],[165,48],[170,80],[175,76],[180,70]]

y = [0, 0, 0, 1,1,1]

# 퍼셉트론을 생성한다. tol는 종료 조건이다. random\_state는 난수의 시드이다.

clf = Perceptron(tol=1e-3, random\_state=0)

# 학습을 수행한다.

clf.fit(X, y)

# 테스트를 수행한다.

print(clf.predict(X))

print(clf.coef\_)

print(clf.intercept\_)

[0 0 0 1 1 1]

[[-59. 160.]]

[-1.]

epsilon = 0.0000001 # 부동소수점 오차 방지

def step\_func(t): # 퍼셉트론의 활성화 함수

if t > epsilon: return 1

else: return 0

X = np.array([ # 훈련 데이터 세트

[0, 1, -1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[1, 0, -1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[1, 2, -1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[2, 1, -1] # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

])

y = np.array([0, 0, 1, 1]) # 정답을 저장하는 넘파이 행렬

W = np.zeros(len(X[0]))

def perceptron\_fit(X, Y, epochs=10): # 퍼셉트론 학습 알고리즘 구현

global W

eta = 1.0 # 학습률

for t in range(epochs):

print("epoch=", t, "======================")

for i in range(len(X)):

predict = step\_func(np.dot(X[i], W))

error = Y[i] - predict # 오차 계산

W += eta \* error \* X[i] # 가중치 업데이트

print("현재 처리 입력=",X[i],"정답=",Y[i],"출력=",predict,"변경된 가중치=", W)

print("================================")

def perceptron\_predict(X, Y): # 예측

global W

for x in X:

print(x[0], x[1], "->", step\_func(np.dot(x, W)))

perceptron\_fit(X, y, 6)

perceptron\_predict(X, y)

epoch= 0 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 0 변경된 가중치= [0. 0. 0.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 0 변경된 가중치= [0. 0. 0.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 0 변경된 가중치= [ 1. 2. -1.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 1 변경된 가중치= [ 1. 2. -1.] ================================ epoch= 1 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 1 변경된 가중치= [1. 1. 0.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 1 변경된 가중치= [0. 1. 1.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 1 변경된 가중치= [0. 1. 1.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 0 변경된 가중치= [2. 2. 0.] ================================ epoch= 2 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 1 변경된 가중치= [2. 1. 1.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 1 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] ================================ epoch= 3 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] ================================ epoch= 4 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] ================================ epoch= 5 ====================== 현재 처리 입력= [ 0 1 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 0 -1] 정답= 0 출력= 0 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 1 2 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] 현재 처리 입력= [ 2 1 -1] 정답= 1 출력= 1 변경된 가중치= [1. 1. 2.] ================================ 0 1 -> 0 1 0 -> 0 1 2 -> 1 2 1 -> 1

epsilon = 0.0000001 # 부동소수점 오차 방지

def step\_func(t): # 퍼셉트론의 활성화 함수

if t > epsilon: return 1

else: return 0

X = np.array([ # 훈련 데이터 세트

[0.1, 0.7, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[0.3, 0.6, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[0.4, 0.1, 1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[0.6,0.3,1], # 맨 끝의 1은 바이어스를 위한 입력 신호 1이다.

[0.1,1.0,1],

[0.3,1.0,1],

[0.7,0.7,1],

[0.9,0.5,1]

])

y = np.array([0, 0,0,0 ,1, 1,1,1]) # 정답을 저장하는 넘파이 행렬

W = np.zeros(len(X[0]))

def perceptron\_fit(X, Y, epochs=10): # 퍼셉트론 학습 알고리즘 구현

global W

eta = 0.2 # 학습률

for t in range(epochs):

print("epoch=", t, "======================")

for i in range(len(X)):

predict = step\_func(np.dot(X[i], W))

error = Y[i] - predict # 오차 계산

W += eta \* error \* X[i] # 가중치 업데이트

print("현재 처리 입력=",X[i],"정답=",Y[i],"출력=",predict,"변경된 가중치=", W)

print("================================")

def perceptron\_predict(X, Y): # 예측

global W

for x in X:

print(x[0], x[1], "->", step\_func(np.dot(x, W)))

perceptron\_fit(X, y, 10)

perceptron\_predict(X, y)

epoch= 5 ====================== 현재 처리 입력= [0.1 0.7 1. ] 정답= 0 출력= 1 변경된 가중치= [-0.06 0.18 -0.2 ] 현재 처리 입력= [0.3 0.6 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.18 -0.2 ] 현재 처리 입력= [0.4 0.1 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.18 -0.2 ] 현재 처리 입력= [0.6 0.3 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.18 -0.2 ] 현재 처리 입력= [0.1 1. 1. ] 정답= 1 출력= 0 변경된 가중치= [-0.04 0.38 0. ] 현재 처리 입력= [0.3 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [-0.04 0.38 0. ] 현재 처리 입력= [0.7 0.7 1. ] 정답= 1 출력= 1 변경된 가중치= [-0.04 0.38 0. ] 현재 처리 입력= [0.9 0.5 1. ] 정답= 1 출력= 1 변경된 가중치= [-0.04 0.38 0. ] ================================ epoch= 6 ====================== 현재 처리 입력= [0.1 0.7 1. ] 정답= 0 출력= 1 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.3 0.6 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.4 0.1 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.6 0.3 1. ] 정답= 0 출력= 0 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.1 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.3 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [-0.06 0.24 -0.2 ] 현재 처리 입력= [0.7 0.7 1. ] 정답= 1 출력= 0 변경된 가중치= [0.08 0.38 0. ] 현재 처리 입력= [0.9 0.5 1. ] 정답= 1 출력= 1 변경된 가중치= [0.08 0.38 0. ] ================================ epoch= 7 ====================== 현재 처리 입력= [0.1 0.7 1. ] 정답= 0 출력= 1 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.3 0.6 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.4 0.1 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.6 0.3 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.1 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.3 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.7 0.7 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.06 0.24 -0.2 ] 현재 처리 입력= [0.9 0.5 1. ] 정답= 1 출력= 0 변경된 가중치= [0.24 0.34 0. ] ================================ epoch= 8 ====================== 현재 처리 입력= [0.1 0.7 1. ] 정답= 0 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.3 0.6 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.4 0.1 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.6 0.3 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.1 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.3 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.7 0.7 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.9 0.5 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] ================================ epoch= 9 ====================== 현재 처리 입력= [0.1 0.7 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.3 0.6 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.4 0.1 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.6 0.3 1. ] 정답= 0 출력= 0 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.1 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.3 1. 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.7 0.7 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] 현재 처리 입력= [0.9 0.5 1. ] 정답= 1 출력= 1 변경된 가중치= [ 0.22 0.2 -0.2 ] ================================ 0.1 0.7 -> 0 0.3 0.6 -> 0 0.4 0.1 -> 0 0.6 0.3 -> 0 0.1 1.0 -> 1 0.3 1.0 -> 1 0.7 0.7 -> 1 0.9 0.5 -> 1

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.linear\_model import Perceptron

iris = load\_iris()

X=iris.data[:,(0,1)]

y=(iris.target == 0).astype(int)

percep = Perceptron(random\_state=32)

percep.fit(X,y)

percep.score(X,y)

0.8533333333333334

def step(x):

result =x >0.000001

return result.astype(int)

import numpy as np

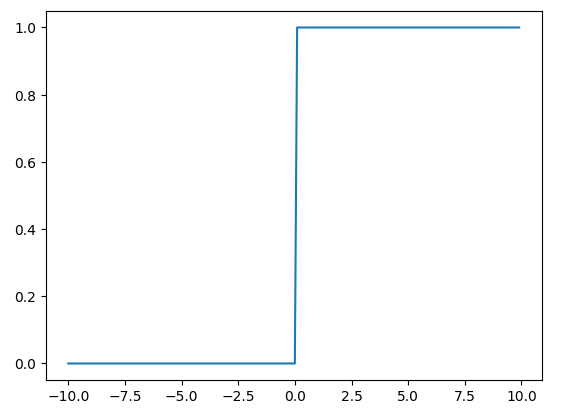
import matplotlib.pyplot as plt

x=np.arange(-10.0,10.0,0.1)

y=step(x)

plt.plot(x,y)

plt.show()



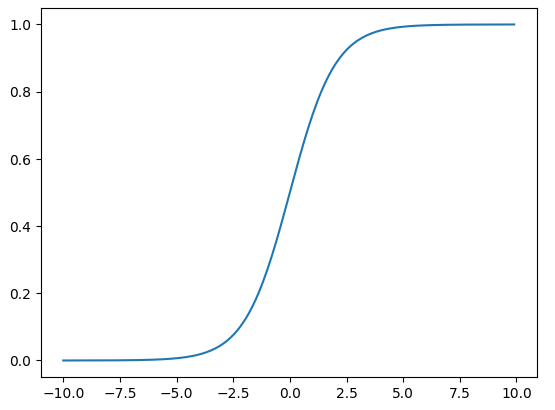
def sigmoid(x):

return 1.0/(1.0+np.exp(-x))

y=sigmoid(x)

plt.plot(x,y)

plt.show()



def tahn(x):

result = x>0.000001

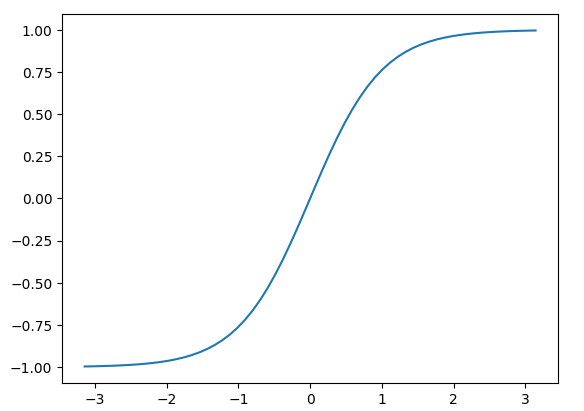
return 2.0/(1.0+np.exp(-2\*x))-1

x=np.linspace(-np.pi,np.pi,60)

y=tahn(x)

plt.plot(x,y)

plt.show()



def relu(x):

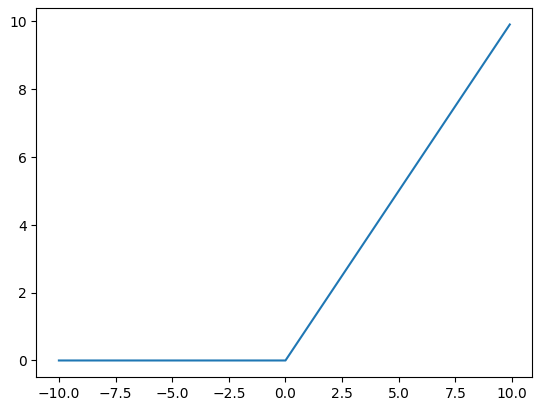
return np.maximum(x,0)

x=np.arange(-10.0,10.0,0.1)

y=relu(x)

plt.plot(x,y)

plt.show()



import numpy as np

# 시그모이드 함수

def actf(x):

return 1/(1+np.exp(-x))

# 시그모이드 함수의 미분치

def actf\_deriv(x):

return x\*(1-x)

# 입력유닛의 개수, 은닉유닛의 개수, 출력유닛의 개수

inputs, hiddens, outputs = 2, 2, 1

learning\_rate=0.2

# 훈련 샘플과 정답

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

T = np.array([[0], [1], [1], [0]])

W1 = np.array([[0.10, 0.20],

[0.30, 0.40]])

W2 = np.array([[0.50], [0.60]])

B1 = np.array([0.1, 0.2])

B2 = np.array([0.3])

# 순방향 전파 계산

def predict(x):

layer0 = x # 입력을 layer0에 대입한다.

Z1 = np.dot(layer0, W1)+B1 # 행렬의 곱을 계산한다.

layer1 = actf(Z1) # 활성화 함수를 적용한다.

Z2 = np.dot(layer1, W2)+B2 # 행렬의 곱을 계산한다.

layer2 = actf(Z2) # 활성화 함수를 적용한다.

return layer0, layer1, layer2

def test():

for x, y in zip(X, T):

x = np.reshape(x, (1, -1)) # x를 2차원 행렬로 만든다.입력은 2차원이어야 한다.

layer0, layer1, layer2 = predict(x)

print(x, y, layer2)

test()

[[0 0]] [0] [[0.70938314]]

[[0 1]] [1] [[0.72844306]]

[[1 0]] [1] [[0.71791234]]

[[1 1]] [0] [[0.73598705]]

x = 10

learning\_rate = 0.2

precision = 0.00001

max\_iterations = 100

# 손실함수를 람다식으로 정의한다.

loss\_func = lambda x: (x-3)\*\*2 + 10

# 그래디언트를 람다식으로 정의한다. 손실함수의 1차 미분값이다.

gradient = lambda x: 2\*x-6

# 그래디언트 강하법

for i in range(max\_iterations):

x = x - learning\_rate \* gradient(x)

print("손실함수값(", x, ")=", loss\_func(x))

print("최소값 = ", x)

손실함수값( 7.199999999999999 )= 27.639999999999993  
손실함수값( 5.52 )= 16.350399999999997  
손실함수값( 4.512 )= 12.286143999999998  
손실함수값( 3.9071999999999996 )= 10.82301184  
손실함수값( 3.54432 )= 10.2962842624  
손실함수값( 3.3265919999999998 )= 10.106662334464  
손실함수값( 3.1959551999999998 )= 10.03839844040704  
손실함수값( 3.11757312 )= 10.013823438546535  
손실함수값( 3.070543872 )= 10.004976437876753  
손실함수값( 3.0423263232 )= 10.001791517635631  
손실함수값( 3.02539579392 )= 10.000644946348826  
손실함수값( 3.015237476352 )= 10.000232180685577  
손실함수값( 3.0091424858112 )= 10.000083585046807  
손실함수값( 3.00548549148672 )= 10.000030090616852  
손실함수값( 3.003291294892032 )= 10.000010832622067  
손실함수값( 3.0019747769352194 )= 10.000003899743945  
손실함수값( 3.0011848661611316 )= 10.00000140390782  
손실함수값( 3.000710919696679 )= 10.000000505406815  
손실함수값( 3.0004265518180073 )= 10.000000181946453  
손실함수값( 3.0002559310908046 )= 10.000000065500723  
손실함수값( 3.0001535586544827 )= 10.00000002358026  
손실함수값( 3.0000921351926895 )= 10.000000008488893  
손실함수값( 3.0000552811156136 )= 10.000000003056002  
손실함수값( 3.000033168669368 )= 10.00000000110016  
손실함수값( 3.000019901201621 )= 10.000000000396058  
손실함수값( 3.0000119407209724 )= 10.000000000142581  
손실함수값( 3.0000071644325836 )= 10.00000000005133  
손실함수값( 3.00000429865955 )= 10.000000000018478  
손실함수값( 3.00000257919573 )= 10.000000000006652  
손실함수값( 3.0000015475174377 )= 10.000000000002395  
손실함수값( 3.0000009285104627 )= 10.000000000000862  
손실함수값( 3.0000005571062776 )= 10.00000000000031  
손실함수값( 3.0000003342637664 )= 10.000000000000112  
손실함수값( 3.0000002005582598 )= 10.00000000000004  
손실함수값( 3.0000001203349558 )= 10.000000000000014  
손실함수값( 3.0000000722009736 )= 10.000000000000005  
손실함수값( 3.000000043320584 )= 10.000000000000002  
손실함수값( 3.0000000259923505 )= 10.0  
손실함수값( 3.0000000155954103 )= 10.0  
손실함수값( 3.000000009357246 )= 10.0  
손실함수값( 3.0000000056143477 )= 10.0  
손실함수값( 3.0000000033686085 )= 10.0  
손실함수값( 3.000000002021165 )= 10.0  
손실함수값( 3.000000001212699 )= 10.0  
손실함수값( 3.0000000007276193 )= 10.0  
손실함수값( 3.0000000004365717 )= 10.0  
손실함수값( 3.000000000261943 )= 10.0  
손실함수값( 3.000000000157166 )= 10.0  
손실함수값( 3.0000000000942997 )= 10.0  
손실함수값( 3.0000000000565796 )= 10.0  
손실함수값( 3.000000000033948 )= 10.0  
손실함수값( 3.0000000000203686 )= 10.0  
손실함수값( 3.0000000000122213 )= 10.0  
손실함수값( 3.000000000007333 )= 10.0  
손실함수값( 3.0000000000043996 )= 10.0  
손실함수값( 3.0000000000026397 )= 10.0  
손실함수값( 3.0000000000015836 )= 10.0  
손실함수값( 3.0000000000009504 )= 10.0  
손실함수값( 3.00000000000057 )= 10.0  
손실함수값( 3.000000000000342 )= 10.0  
손실함수값( 3.000000000000205 )= 10.0  
손실함수값( 3.000000000000123 )= 10.0  
손실함수값( 3.0000000000000737 )= 10.0  
손실함수값( 3.0000000000000444 )= 10.0  
손실함수값( 3.0000000000000266 )= 10.0  
손실함수값( 3.000000000000016 )= 10.0  
손실함수값( 3.0000000000000098 )= 10.0  
손실함수값( 3.0000000000000058 )= 10.0  
손실함수값( 3.0000000000000036 )= 10.0  
손실함수값( 3.000000000000002 )= 10.0  
손실함수값( 3.0000000000000013 )= 10.0  
손실함수값( 3.000000000000001 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
손실함수값( 3.0000000000000004 )= 10.0  
최소값 = 3.0000000000000004