SMARCLE 2021 winter study Team 3

참 거짓 판단 장치

< 로지스틱 회귀 >

Logistic Regression

17 김찬영, 17 최태규, 18 장윤정, 20 김준수

SMARCLE

Contents

18 장윤정

로지스틱 회귀에 관한 이론

17 최태규

로지스틱 회귀 실습코드 설명

17 김찬영

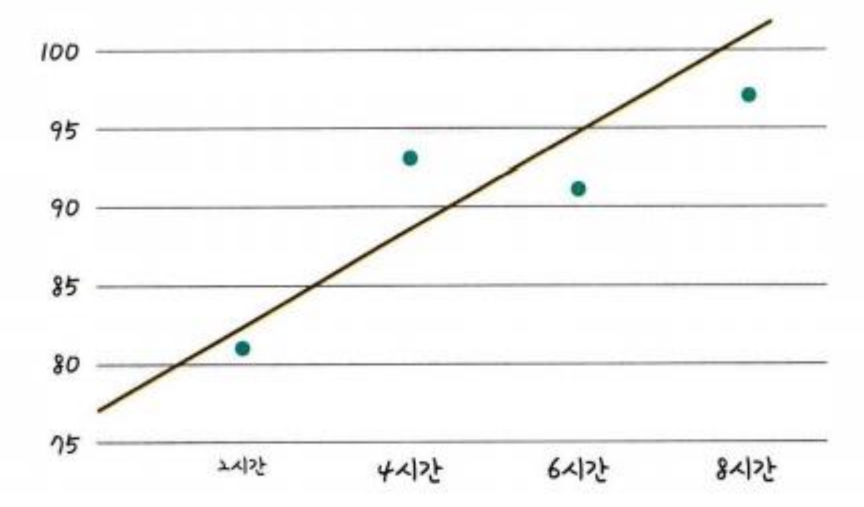
실습코드 개조 및 토론 & 로지스틱 회귀에서 퍼셉트론으로

SMARCLE

01 이론

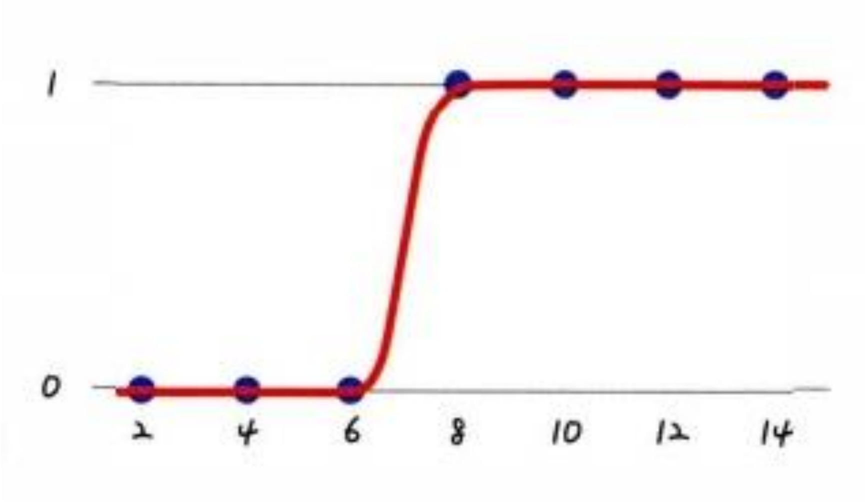
01 로지스틱 회귀

선형 회귀 (Linear Regression)



연속형 input ⇨ 연속형 output

로지스틱 회귀 (Logistic Regression)



연속형 input ⇨ 이산형 output

01 로지스틱 회귀

❑ 로지스틱 회귀의 사용

- 범주형 적 의사결정을 필요로하는 모델 ex) 제품이 불량인지 정상인지, 고객이 이탈고객인지 잔류고객인지 페이스북 피드를 보여줄지 숨길지, 메일이 스팸인지 햄인지 등

1. 이진변수 ex) 성공/실패, 사망/생존 ->binary classification
2. 멀티변수 -> Multi-Class Classification

How?

로지스틱 회귀 원리 이용 → 참거짓 판단장치 모델 생성 → 새로운 질문이 들어오면? → 모델의 범주 중 하나로 예측

01 로지스틱 회귀

1 (Cat)

0 (Non Cat)

0 (Non Cat)

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

Output은 0~1

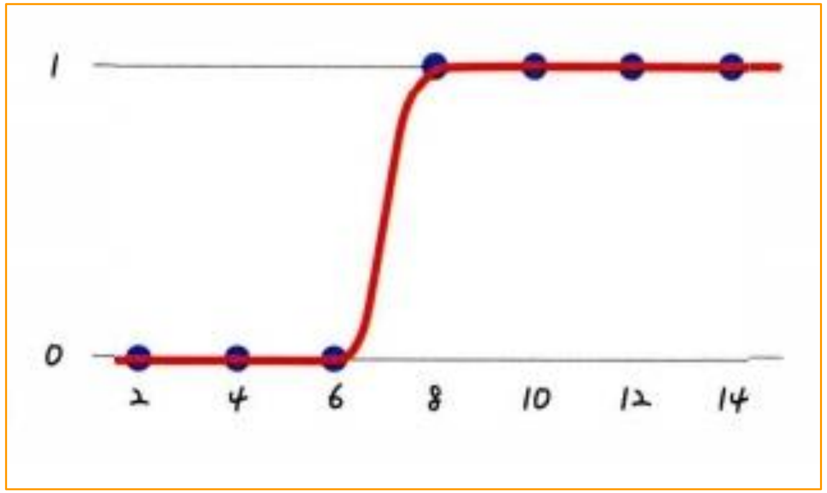
ex) 0.78 -> 고양이라고 판단

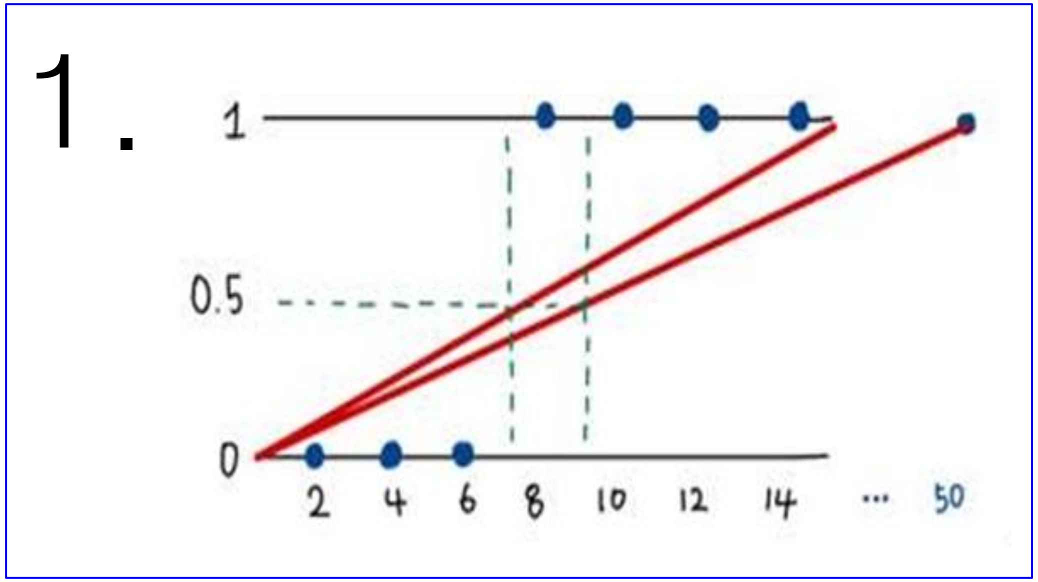
01

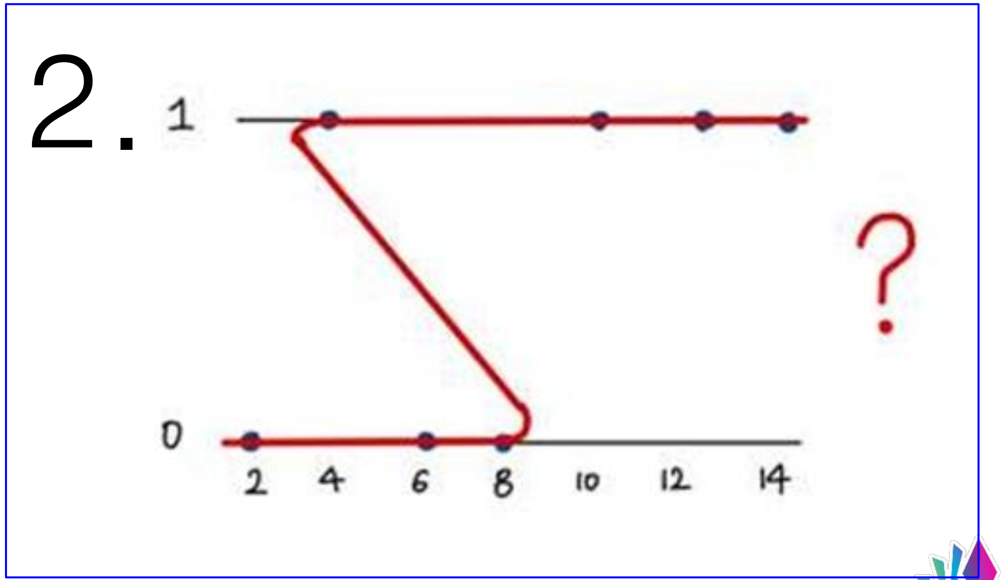
로지스틱 회귀

❑ 교재 <모두의 딥러닝> 속 예시에 대한 의문점

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



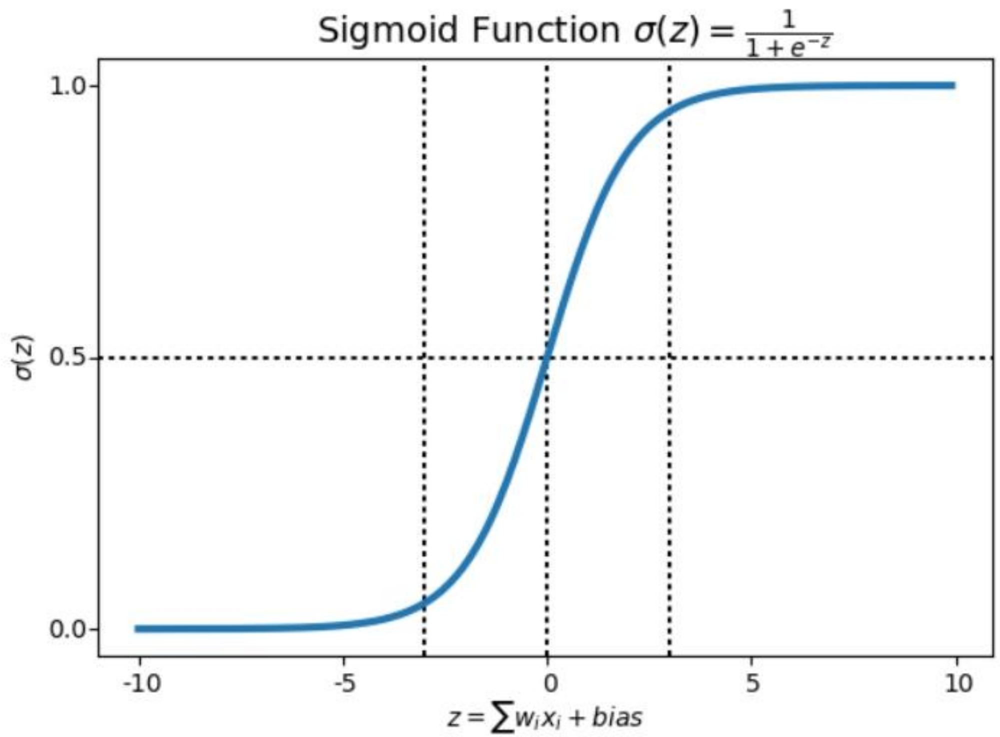




SMARCLE

02 시그모이드 함수

- Sigmoid function ≔ Logistic function
- input값에 대해 단조증가(or 단조감소) 하는 S자형 그래프
- Squashing function (Large input(-∞~∞) ⇨ Small output(0~1))
- 밑을 자연상수 e로 갖는 지수함수가 분모에 포함되는 함수



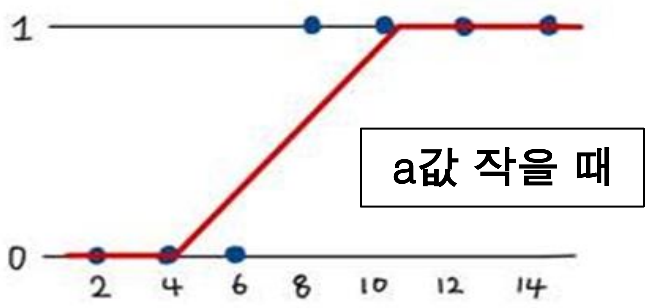
$$y = \frac{1}{1+e^{-(ax+b)}}$$

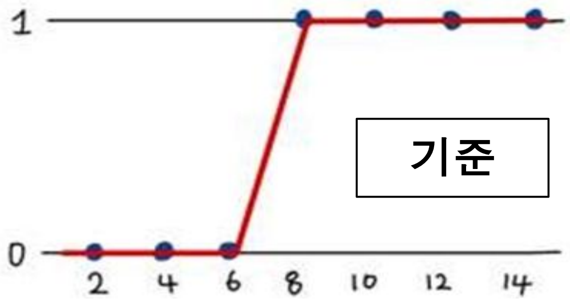
a와 b값에 따라 오차 변화

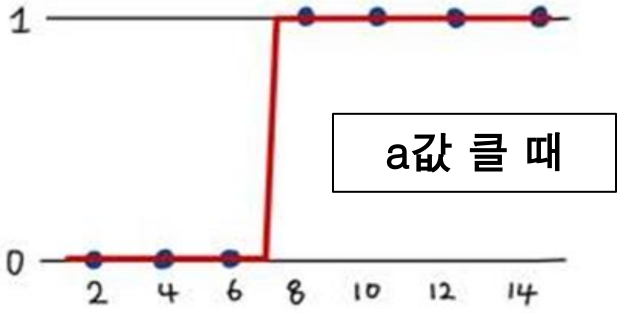
SMARCLE

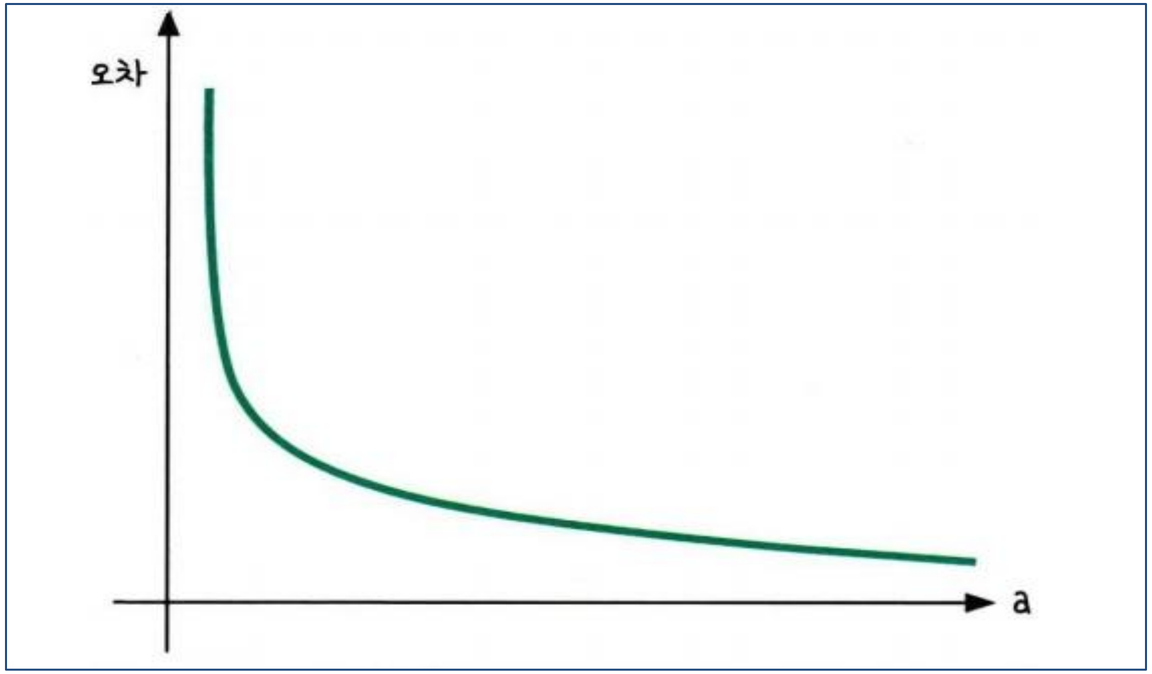
02

시그모이드 함수

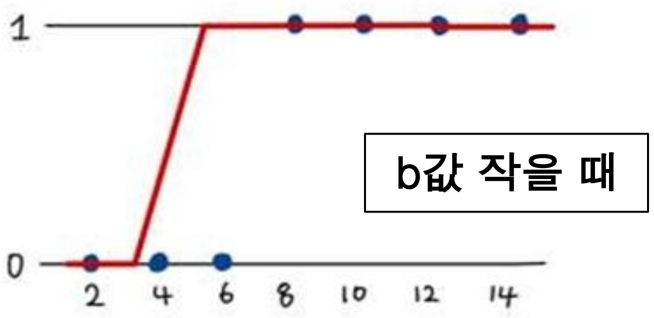

1
0
2 4 6 8 10 12 14
a값 작을 때

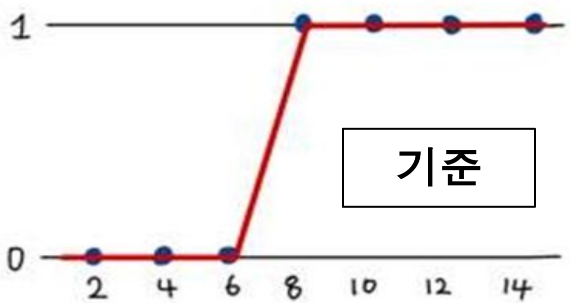

1
0
2 4 6 8 10 12 14
기준

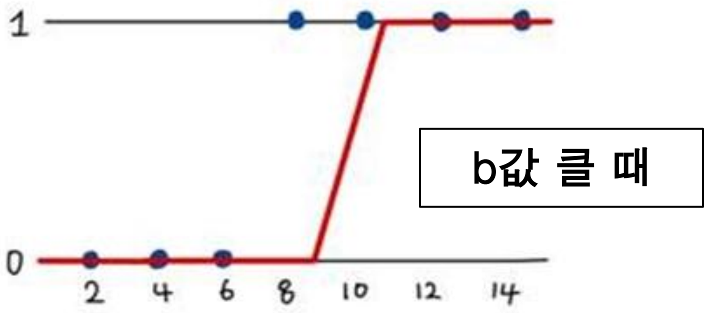

1
0
2 4 6 8 10 12 14
a값 클 때

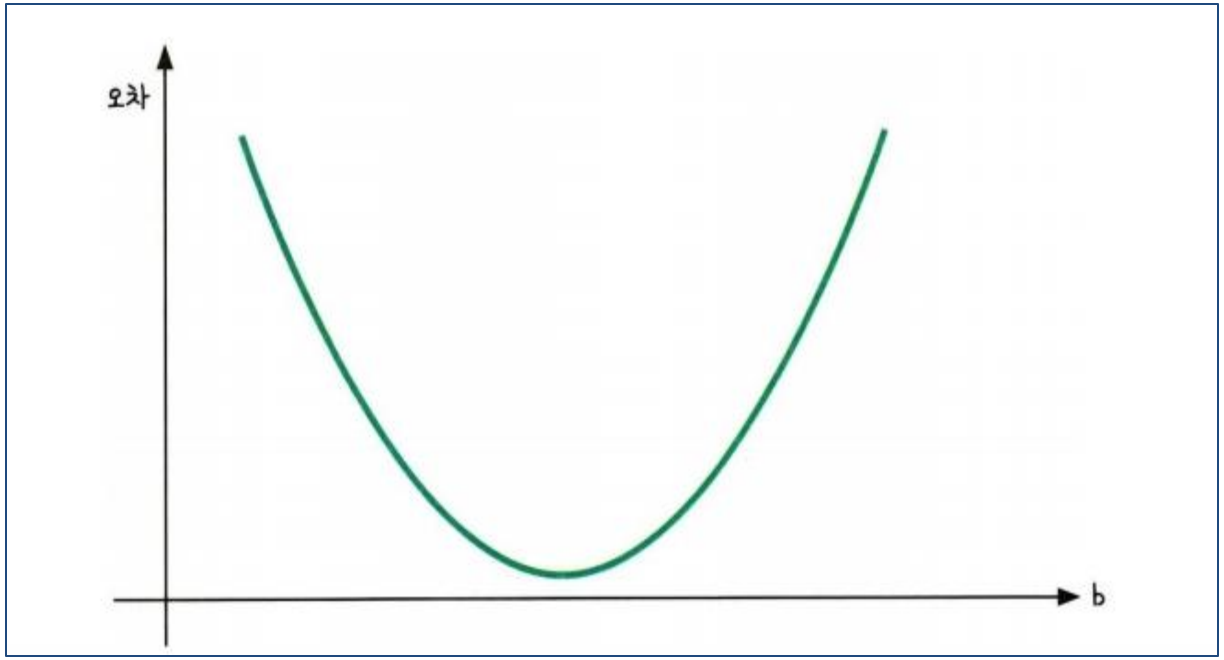

오차
a

02 시그모이드 함수


1
0
2 4 6 8 10 12 14
b값 작을 때


1
0
2 4 6 8 10 12 14
기준


1
0
2 4 6 8 10 12 14
b값 클 때


오차
b

03

오차 공식과 로그함수

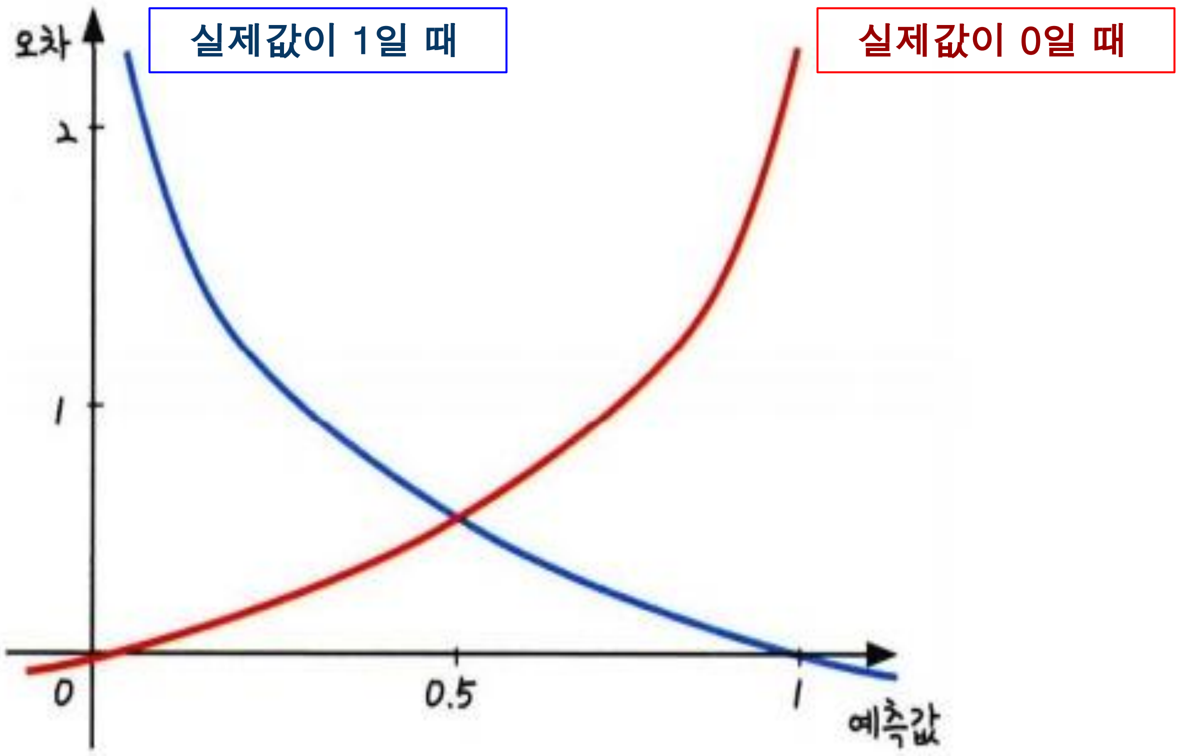
시그모이드 함수에서 a, b값 구하는 법?

경사 하강법!

오차 구하는 공식 필요!

정답에서 가까워질 수록 Cost function 값은 작고,
정답에서 멀어질 수록 Cost function 값은 크게 설계!

예측값과 실제 결과값의 차이를 나타내는 함수



SMARCLE

03 오차 공식과 로그함수

Cost function

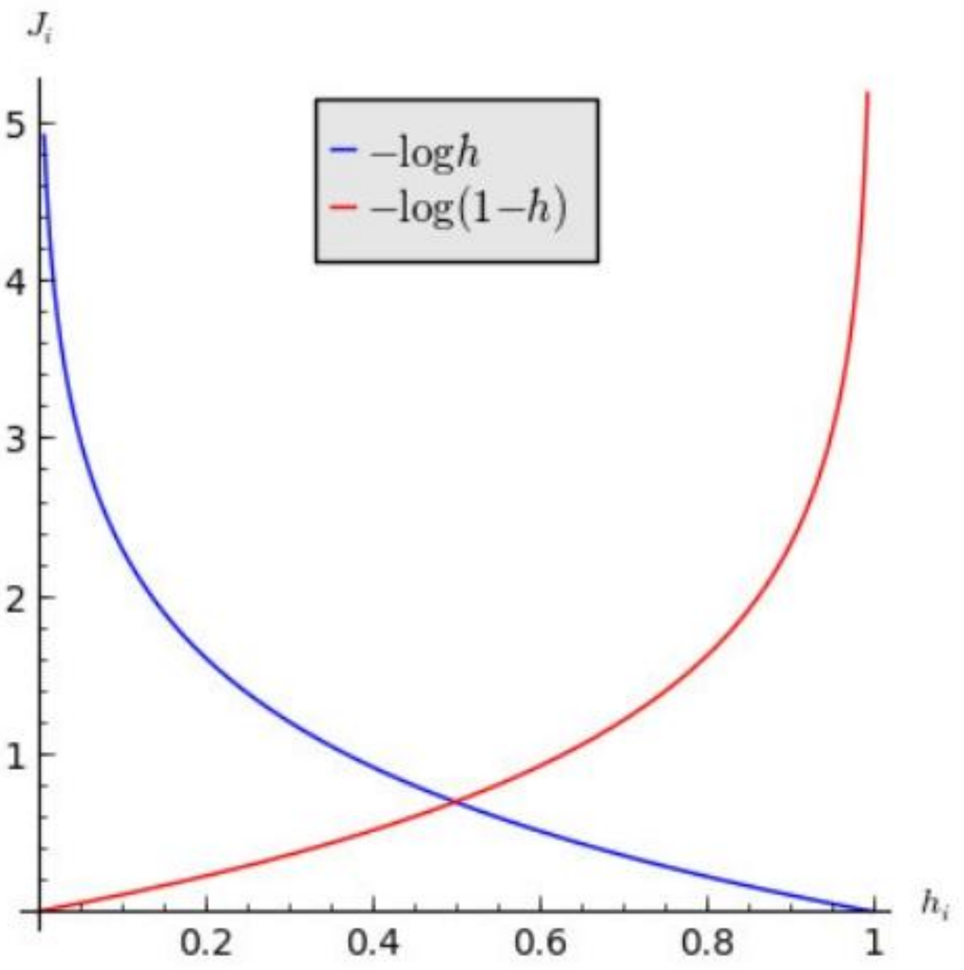
$$cost(W) = \frac{1}{m} \sum c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

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

Loss function

: input값에 대한 예측값과 실제값 사이의 오차를 계산하는 함수



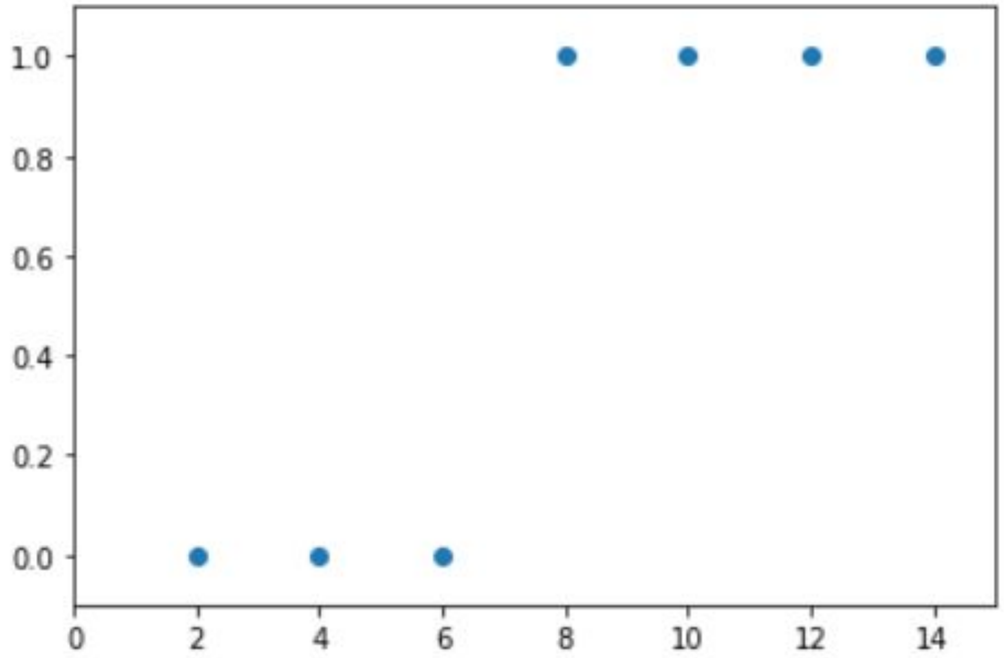
SMARCLE

출처 - https://copycode.tistory.com/162

02 실습코드 설명

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

```python
data =[[2,0],[4,0],[6,0],[8,1],[10,1],[12,1],[14,1]]

x_data = [i[0] for i in data] # 공부시간
y_data = [i[1] for i in data] # 합격여부




# 데이터 시각화
plt.scatter(x_data,y_data) # 점으로 나타내기
#plt.plot(x_data,y_data) # 선으로 나타내기
# x,y 범위 설정
plt.xlim(0,15)
plt.ylim(-0.1,1.1)
plt.show()

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

# a,b 설정
a=b=0

# 학습률
lr = 0.05
```



경사하강법

```
# 경사하강법
epoch=2001
for i in range(epoch):
  for x_data, y_data in data:
    # a,b 각각 편미분 값
    a_diff= x_data*(sigmoid(a*x_data+b)-y_data)
    b_diff = sigmoid(a*x_data + b) - y_data
    # a,b 최신화
    a= a- lr*a_diff
    b= b- lr*b_diff


    if i%1000 == 0:
      print("에포크%.4f " %(i))
      print("a=",a)
      print("b=",b)
```

SMARCLE

학습 결과

```
에포크0.0000
a= -0.05
b= -0.025
```

```
에포크1000.0000
a= 1.4119848217717417
b= -9.954745130962369
```

```
에포크2000.0000
a= 1.9068044592233457
b= -12.951253713260089
```

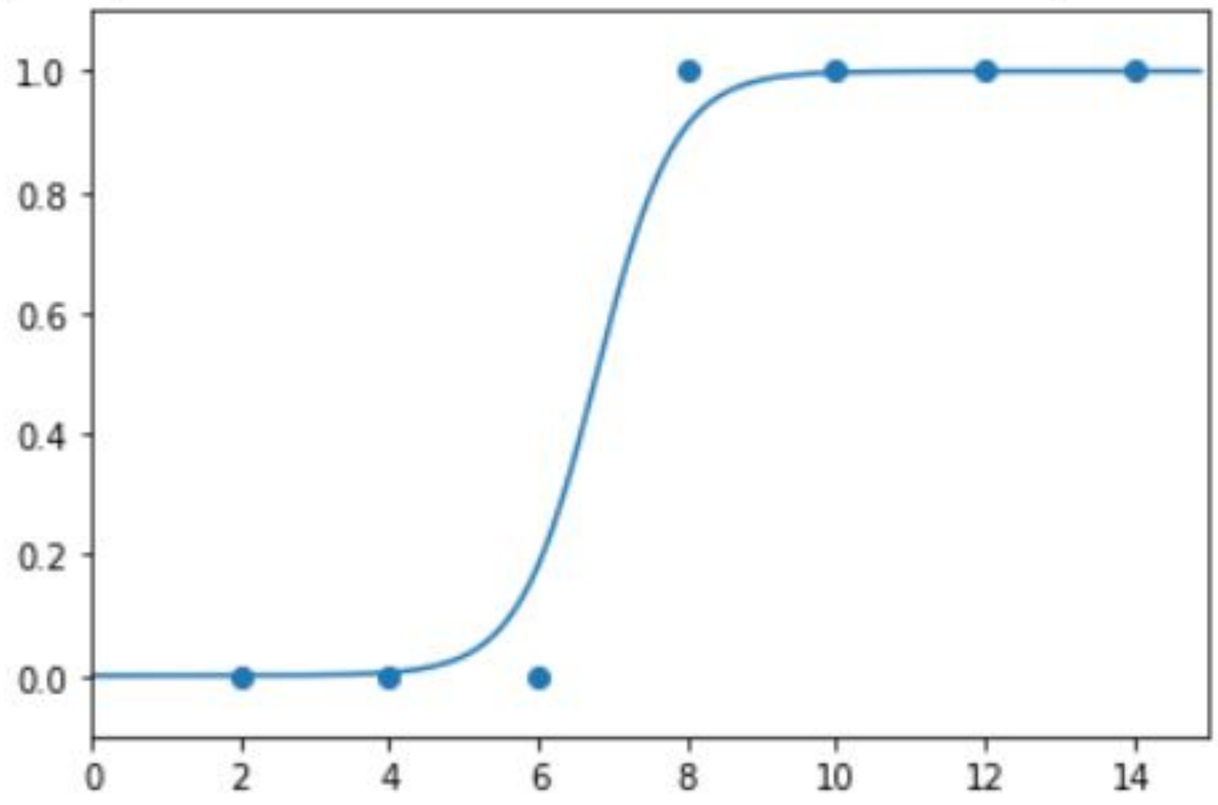
a 는 증가
b 는 감소

학습된 시그모이드 함수 시각화

```
# 학습시킨 로지스틱 회귀 함수 시각화

x_range= (np.arange(0,15,0.1)) # x 값의 범위 설정
plt.xlim(0,15)
plt.ylim(-0.1,1.1)

plt.plot(np.arange(0,15,0.1),np.array([sigmoid(a*x1+b) for x1 in x_range]))
```

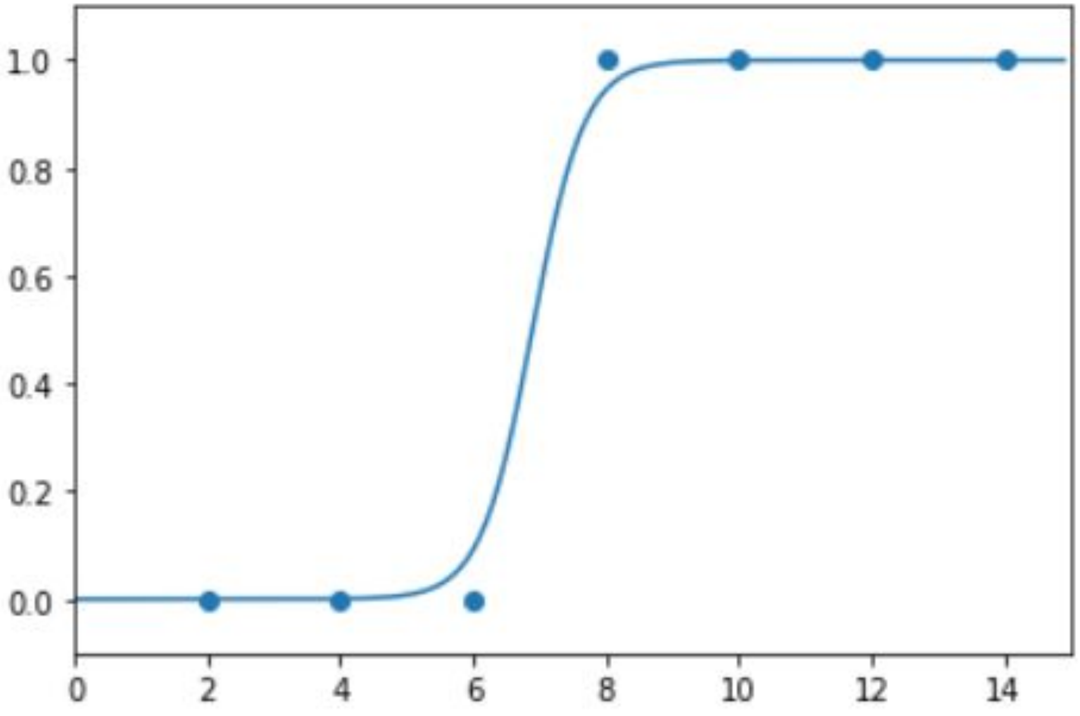
학습된 시그모이드 함수 시각화

```
에포크2000.0000
a= 1.9068044592233457
b= -12.951253713260089
```

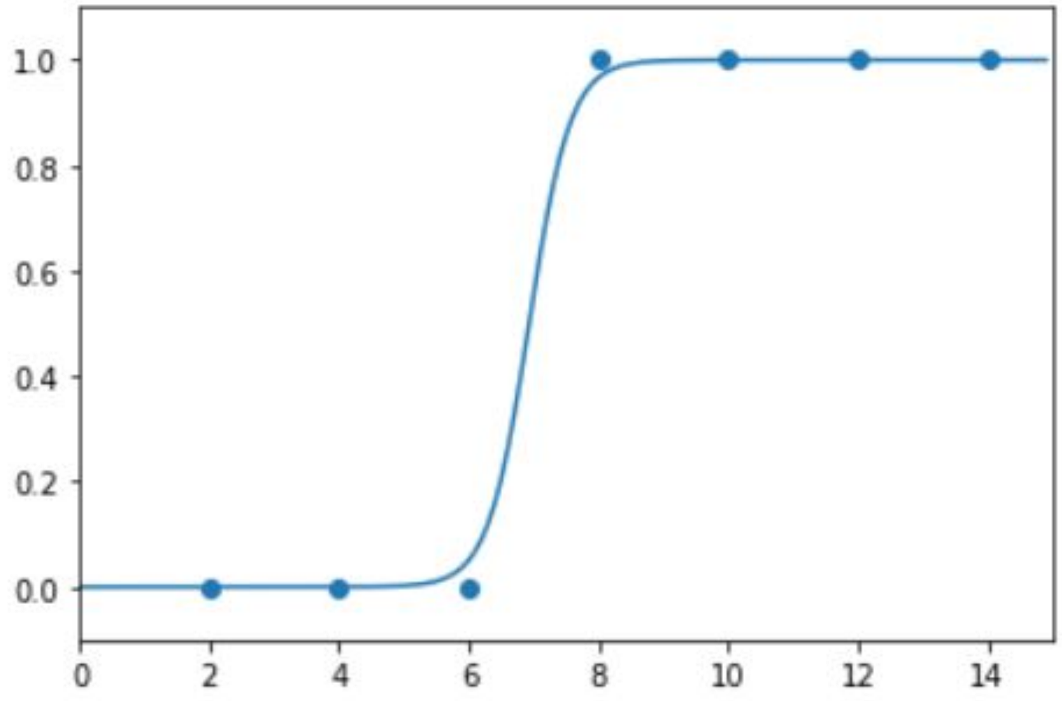


에포크(epoch) 변화시켰을 때

에포크5000.0000
a= 2.5706374134891763
b= -17.728398971271183



에포크10000.0000
a= 3.1509359239551777
b= -21.847191342449918



03

실습코드

개조 및 토론

01

Tensorflow 2.x

TensorFlow

구글이 2011년에 개발을 시작하여 2015년에 오픈 소스로 공개한 기계학습 라이브러리.

01 Tensorflow 2.x


```python
import numpy                    # 행렬 연산 등 외부 모듈 이용
import scipy.special            # 시그모이드 함수 등 외부 모듈 이용
import matplotlib.pyplot as plt # 이미지 출력 imshow 등 외부 모듈 이용

class MyNeuralNetwork:
    def __init__(self, inputNodesCnt, hiddenNodesCnt, outputNodesCnt, learningRate):
        self.inputNodesCnt  = inputNodesCnt     # 입력계층 노드 수
        self.hiddenNodesCnt = hiddenNodesCnt    # 은닉계층 노드 수
        self.outputNodesCnt = outputNodesCnt    # 출력계층 노드 수
        self.lr             = learningRate      # 학습률

        # numpy.random.normal 첫번째인자(0.0)를 중심, 두번째인자를 너비로하는 정규분포, 세번째인자를 크기로 생
        # 가중치(입력층 -> 은닉층): wih
        # 행렬 사이즈의 순서를 은닉, 입력 순으로 바꾼 이유는 아래 행렬 내적연산을 자연스럽게 하기 위해서이다.
        self.wih = numpy.random.normal(0.0, pow(self.hiddenNodesCnt, -0.5), (self.hiddenNodesCnt, self.i
        # 가중치(은닉층 -> 출력층): who
        self.who = numpy.random.normal(0.0, pow(self.outputNodesCnt, -0.5), (self.outputNodesCnt, self.h
        # 활성화함수 지정(여기서는 시그모이드함수 이용)
        self.activationFunc = lambda x: scipy.special.expit(x)

    # 입력데이터와 정답라벨을 입력받아 학습한다.(마지막에 가중치 업데이트함을 주목하자)
    def train(self, inputsList, targetsList):
        # 가로로된 리스트를 2차원으로 변경하고 세로로 세운다.
        inputs = numpy.array(inputsList, ndmin=2).T

        # 가로로된 리스트를 2차원으로 변경하고 세로로 세운다.
        # targets는 출력계층 노드 수와 같고, 정답지에는 0.99 그렇지 않은 경우 0.01로 저장됨
        # 정답인 1개만 아주 크고 나머진 아주 작은 one-hot encoding으로 구성되어 있다.
        targets = numpy.array(targetsList, ndmin=2).T

        # [은닉계층 노드 수 X 입력계층 노드 수] 행렬과 [입력계층 노드 수 X 1] 행렬의 내적을 구한 후 활성화 함수
        hiddenOutputs = self.activationFunc(numpy.dot(self.wih, inputs))

        # [출력계층 노드 수 X 은닉계층 노드 수] 행렬과 [은닉계층 노드 수 X 1] 행렬의 내적을 구한 후 활성화 함수
        # 출력계층에는 활성화 함수를 생략하기도 한다.
        hypothesis = self.activationFunc(numpy.dot(self.who, hiddenOutputs))

        # 정답과 가정사이에 발생한 오차1(=비용)
        cost = targets - hypothesis
        # (은닉계층 -> 출력계층) 가중치 업데이트
        self.who += self.lr * numpy.dot((cost * hypothesis * (1.0 - hypothesis)), numpy.transpose(hidden

        # 은닉계층 -> 출력계층의 전치와 오차1의 내적을 통해 오차2 계산
        hiddenErrors = numpy.dot(self.who.T, cost)
        # (입력계층 -> 은닉계층) 가중치 업데이트
        self.wih += self.lr * numpy.dot((hiddenErrors * hiddenOutputs * (1.0 - hiddenOutputs)), numpy.tr

    # 입력데이터를 신경망에 테스트하여 결과를 얻는다.
    def test(self, inputsList):
        # 가로로된 리스트를 2차원으로 변경하고 세로로 세운다.
        inputs = numpy.array(inputsList, ndmin=2).T

        # [은닉계층 노드 수 X 입력계층 노드 수] 행렬과 [입력계층 노드 수 X 1] 행렬의 내적을 구한 후 활성화 함수
        hiddenOutputs = self.activationFunc(numpy.dot(self.wih, inputs))

        # [출력계층 노드 수 X 은닉계층 노드 수] 행렬과 [은닉계층 노드 수 X 1] 행렬의 내적을 구한 후 활성화 함수
        return self.activationFunc(numpy.dot(self.who, hiddenOutputs))


# 학습용(traing)데이터와 테스트용(test)데이터를 분리하여 준비
trainingDataFileHandler = open("mnist_train.csv","r")
# 학습용데이터를 읽어서 리스트형태로 저장
trainingDataList = trainingDataFileHandler.readlines()
# 학습용데이터 파일 핸들러를 닫는다.
trainingDataFileHandler.close()


testDataFileHandler = open("mnist_test.csv","r")
# 테스트용데이터를 읽어서 리스트형태로 저장
testDataList = testDataFileHandler.readlines()
# 테스트용데이터 파일 핸들러를 닫는다.
testDataFileHandler.close()




inputNodesCnt  = 784  # 입력계층 노드 수
hiddenNodesCnt = 100  # 은닉계층 노드 수
outputNodesCnt = 10   # 출력계층 노드 수

learningRate = 0.1    # 학습률

net = MyNeuralNetwork(inputNodesCnt, hiddenNodesCnt, outputNodesCnt, learningRate)

# 현재 스텝을 기록한다.
step = [];
# 현재 스텝의 정확도를 기록한다.
stepPerf = [];

for epochs in range(1, 10, 1):
    for e in range(epochs):
        for record in trainingDataList:
            # 한줄의 데이터를 읽어 쉼표를 구분자로 하여 여러 셀에 기록한다.
            cells = record.split(',')

            # 색상값이 0~255 사이 숫자이다.
            # 잘 학습할 수 있도록 0은 피하고 큰수를 피하기 위해 0.01 ~ 1.00 사이로 조정한다.
            inputs = (numpy.asfarray(cells[1:])/255.0*0.99) + 0.01

            # 정답인 1개만 아주 크고 나머진 아주 작은 one-hot encoding으로 구성되어 있다.
            targets = numpy.zeros(outputNodesCnt) + 0.01
            targets[int(cells[0])] = 0.99

            # 학습시킨다.
            net.train(inputs, targets)

    resultCard = []

    for record in testDataList:
        # 한줄의 데이터를 읽어 쉼표를 구분자로 하여 여러 셀에 기록한다.
        cells = record.split(',')
        # 첫번째 항목은 라벨링된 정답지이다.
        target = int(cells[0])
```

MNIST NN using Numpy

```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_cro
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

MNIST NN using Tensorflow 2.x

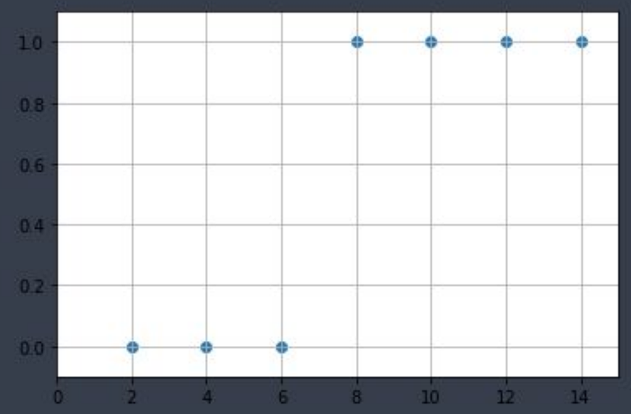
SMARCLE

In [39]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import random

data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14, 1]]

x1 = [i[0] for i in data]
y = [i[1] for i in data]

x_g=[i[0] for i in data]
y_g=[i[1] for i in data]
```

In [40]:

```
#그래프로 나타내 봅니다.
plt.scatter(x1, y)
plt.xlim(0, 15)
plt.ylim(-.1, 1.1)
plt.grid(True)
```



```
x1_data = np.array(x1)
y_data = np.array(y)

#a = tf.Variable(random.random())
#b = tf.Variable(random.random())

a = tf.Variable(0, dtype=tf.float32)
b = tf.Variable(0, dtype=tf.float32)
```

```
compute_loss():
hypothesis = tf.math.sigmoid(a*x1_data+b)
loss = -tf.math.reduce_mean(y_data * tf.math.log(hypothesis) + (1 - y_data) * tf.math.log(1-hypothesis))
return loss
```

SMARCLE

```python
optimizer = tf.optimizers.SGD(lr=0.05)
epoch = 15001

for i in range(epoch):
    optimizer.minimize(compute_loss, var_list=[a,b])
    if i%1000 == 0:
        print(i,'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())
```

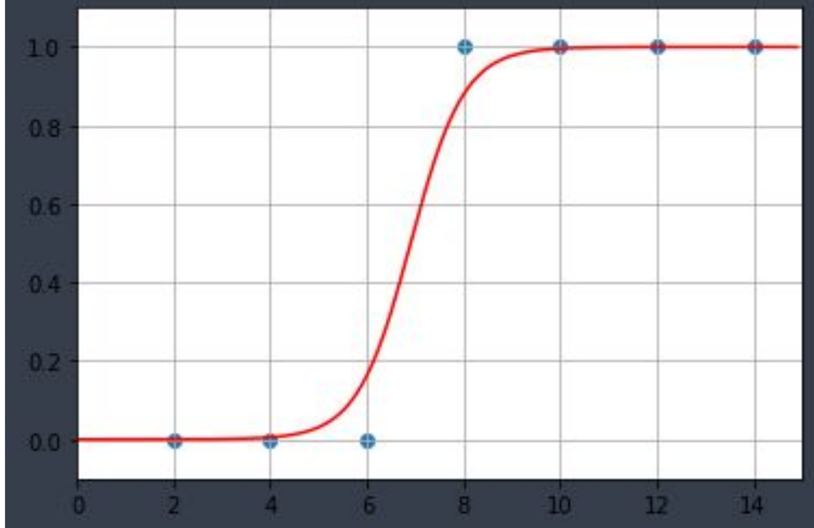
```
0 a: 0.11428572 b: 0.003571429 loss: 0.5546378
1000 a: 0.6151289 b: -3.9133995 loss: 0.18809369
2000 a: 0.831343 b: -5.517361 loss: 0.13457471
3000 a: 0.98071724 b: -6.602698 loss: 0.11040015
4000 a: 1.098763 b: -7.4521236 loss: 0.09564186
5000 a: 1.1981719 b: -8.163362 loss: 0.08530726
6000 a: 1.2850072 b: -8.782278 loss: 0.0774856
7000 a: 1.3626691 b: -9.334306 loss: 0.07126484
8000 a: 1.4332715 b: -9.835133 loss: 0.06614529
9000 a: 1.4982277 b: -10.295177 loss: 0.061825905
10000 a: 1.5585366 b: -10.721763 loss: 0.05811214
11000 a: 1.6149313 b: -11.120252 loss: 0.05487161
12000 a: 1.6679708 b: -11.494706 loss: 0.052010145
13000 a: 1.7180912 b: -11.848301 loss: 0.04945873
14000 a: 1.7656425 b: -12.183562 loss: 0.047164984
15000 a: 1.8109086 b: -12.50254 loss: 0.045088716
```

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

#경사 하강법을 실행합니다.
for i in range(epochs):
    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 % 100 == 0:    # 1000번 반복될 때마다 각 x_data값에 대한
        print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))
```

```python
def compute_loss():
    hypothesis = tf.math.sigmoid(a*x1_data+b)
    loss = -tf.math.reduce_mean(y_data * tf.math.log(hypothesis)
                                + (1 - y_data) * tf.math.log(1-hypothesis))
    return loss

optimizer = tf.optimizers.SGD(lr=0.05)
epoch = 15001

for i in range(epoch):
    optimizer.minimize(compute_loss, var_list=[a,b])
    if i%1000 == 0:
        print(i,'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())
```

```python
plt.scatter(x_g, y_g)
plt.xlim(0, 15)
plt.ylim(-.1, 1.1)
x_range = (np.arange(0, 15, 0.1)) #그래프로 나타낼 x값의 범위를 정합니다.
plt.plot(np.arange(0, 15, 0.1), np.array([tf.math.sigmoid(a*x + b) for x in x_range]),'-r')
plt.grid(True)
plt.show
print("기울기=%.04f, 절편=%.04f" % (a.numpy(), b.numpy()))
```

```
기울기=1.8109, 절편=-12.5025
```



```python
optimizer = tf.optimizers.SGD(lr=0.2)
epoch = 15001

for i in range(epoch):
    optimizer.minimize(compute_loss, var_list=[a,b])
    if i%1000 == 0:
        print(i,'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())
```

lr 0.05 -> 0.2

```
0 a: 0,4571429 b: 0,014285716 loss: 0,8730116
1000 a: 1,1001173 b: -7,4618363 loss: 0,09548956
2000 a: 1,4341743 b: -9,841531 loss: 0,06608279
3000 a: 1,6686838 b: -11,499739 loss: 0,051972844
4000 a: 1,8547254 b: -12,811153 loss: 0,043172337
5000 a: 2,0105572 b: -13,907752 loss: 0,037025653
6000 a: 2,1452446 b: -14,854552 loss: 0,032446183
7000 a: 2,2641075 b: -15,6895075 loss: 0,028885901
8000 a: nan b: nan loss: nan
9000 a: nan b: nan loss: nan
10000 a: nan b: nan loss: nan
11000 a: nan b: nan loss: nan
12000 a: nan b: nan loss: nan
13000 a: nan b: nan loss: nan
14000 a: nan b: nan loss: nan
15000 a: nan b: nan loss: nan
```

1) 수식(이상)과 구현(현실)은 다르다.

결론 부터 말하자면,
가급적이면 수식을 직접 구현하지 말고, [tf.losses, tf.contrib.losses, tf.nn] 등에 미리 구현된 함수를 사용해야 한다.

그 이유는, exp(x) 함수의 값이 지수적으로 증가하므로,
x가 어느 정도만 (e.g 800) 커져도 overflow를 일으키기 때문이다.

위에서 sigmoid(z) = (1 / (1 + exp(-z))) 이므로,
z가 -800 만되도 exp(-z) 가 overflow를 발생 시켜버리는 것이다.

```
compute_loss():
hypothesis = tf.math.sigmoid(a*x1_data+b)
loss = -tf.math.reduce_mean(y_data * tf.math.log(hypothesis) + (1 - y_data) * tf.math.log(1-hypothesis))
return loss
```

SMARCLE

2) SSE에 비해 너무 빠른 learning 으로 인한 oscillation 문제

cross entropy의 수렴속도는 SSE 에 비해 훨씬 빠르기 때문에,
learning rate을 줄여주어야 한다.

https://www.coursera.org/lecture/machine-learning/gradient-descent-in-practice-ii-learning-rate-3iawu

learning rate이 클 경우 oscillation등의 이유로 수렴하지 않는 문제가 발생할 수 있다.
oscillation 이 발생하는 지 알아보는 방법은, traning 시 loss 값을 출력해보는 것이다.
loss는 항상 줄어들어야 하는데, loss가 다시 증가할 경우 oscillation 을 의심해볼 수 있다.

따라서 cross entropy 를 쓸 경우 learning rate을 충분히 작게 해주어야 한다.

```
optimizer = tf.optimizers.SGD(lr=0.2)
epoch = 15001
```

```
optimizer = tf.optimizers.SGD(lr=0.05)
epoch = 15001
```

04

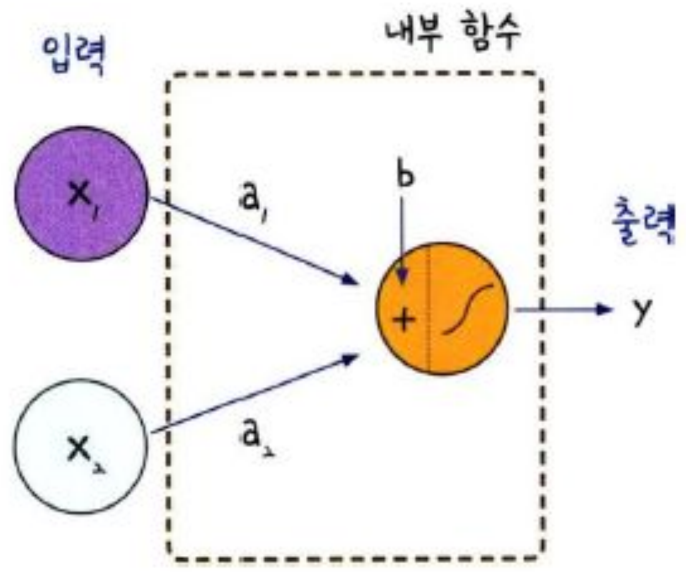
로지스틱 회귀에서

퍼셉트론으로

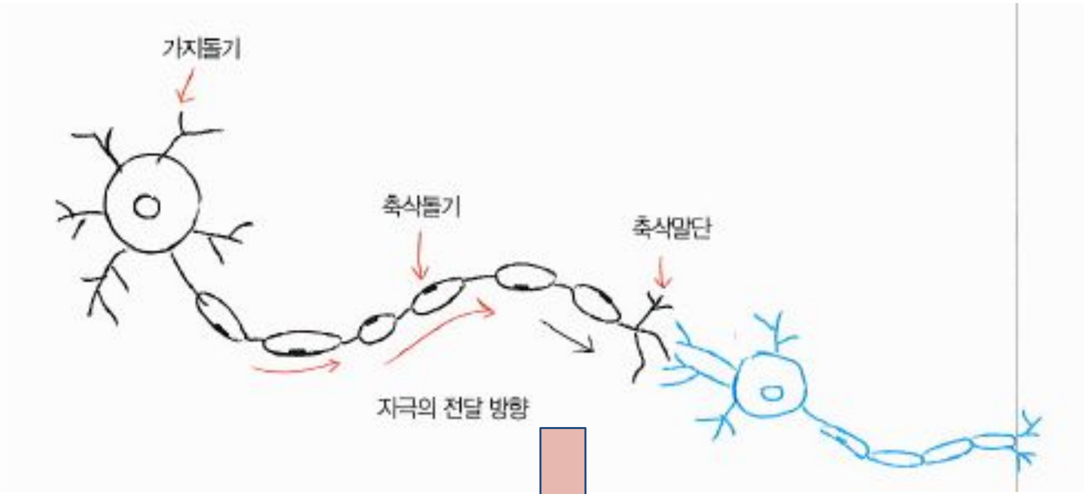
01 퍼셉트론 개요

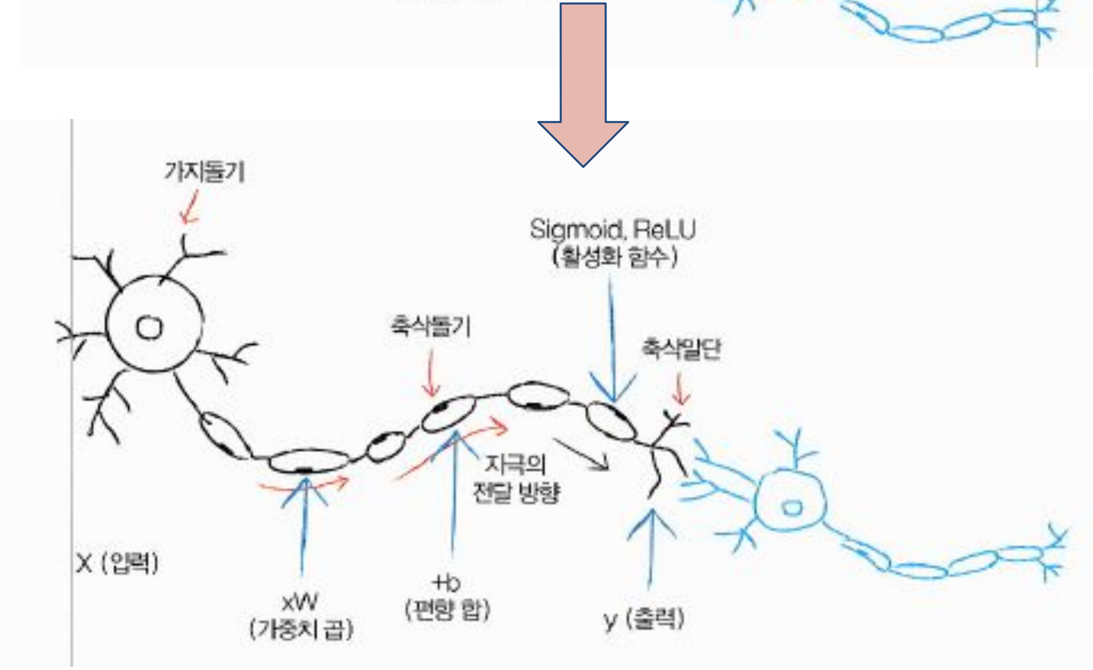
출력 값 입력 값

$$y = a_1x_1 + a_2x_2 + b$$



01 퍼셉트론 개요


가지돌기
축삭돌기
축삭말단
자극의 전달 방향

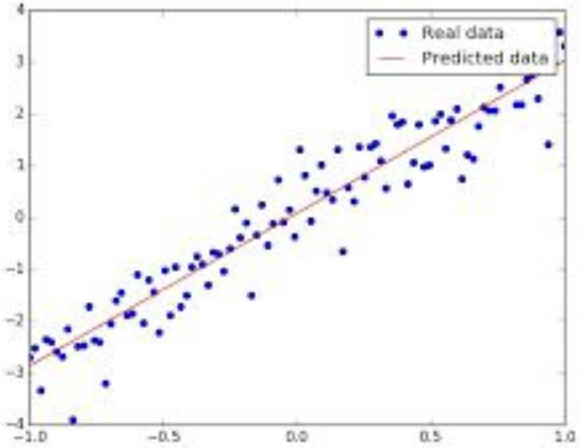

가지돌기
Sigmoid, ReLU
(활성화 함수)
축삭돌기
축삭말단
자극의
전달 방향
X (입력)
xW
(가중치 곱)
+b
(편향 합)
y (출력)

05 단원 정리

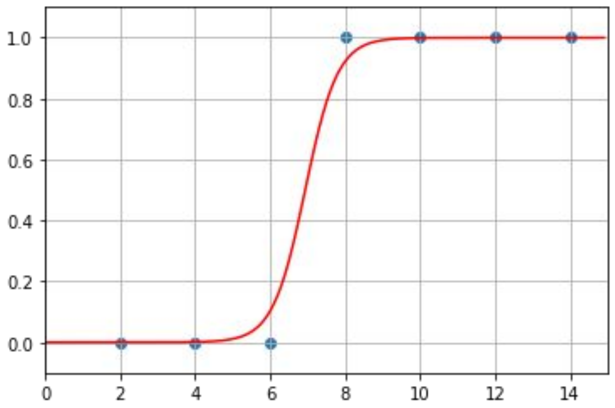
01 딥러닝의 동작 원리

딥러닝의 기본적인 두 가지 계산 원리

선형 회귀(예측선)
Linear Regression

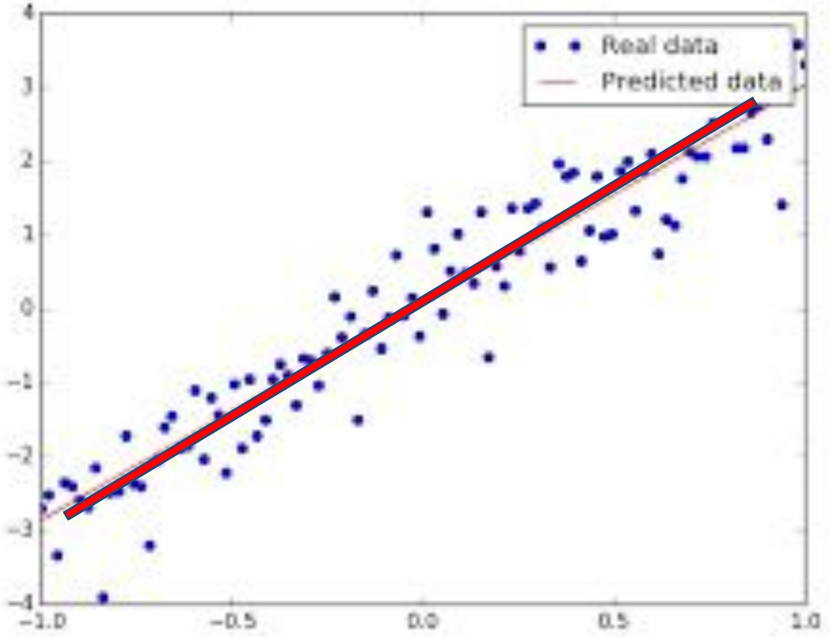


로지스틱 회귀(분류)
Logistic Regression



SMARCLE

02 가설함수 (Hypothesis)



빨간 선을 예측하고 싶어요!

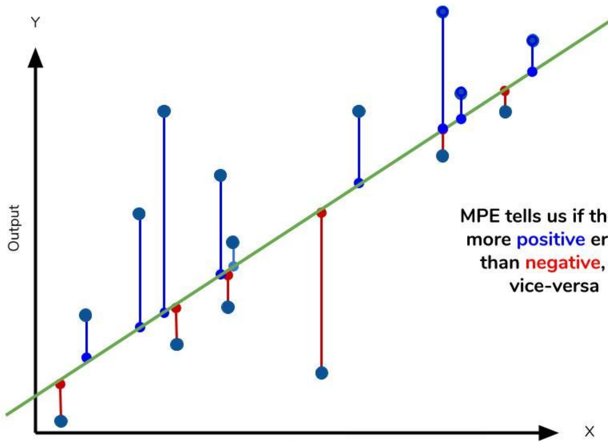
-> 가설함수

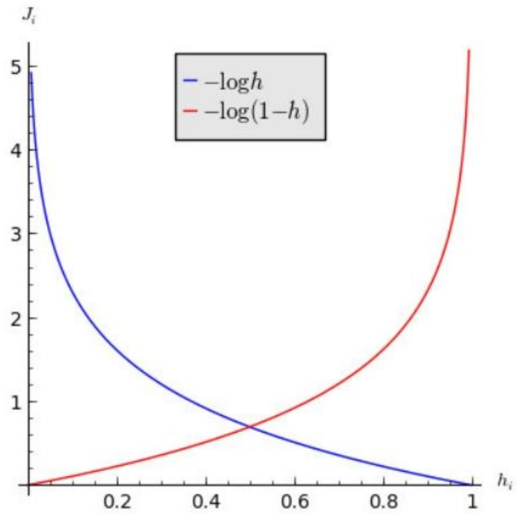
$$H(x) = ax + b$$

우리가 찾아내야 할 변수는 a, b

어떻게 a, b를 맞출 수 있을까?

03 Loss 구하기

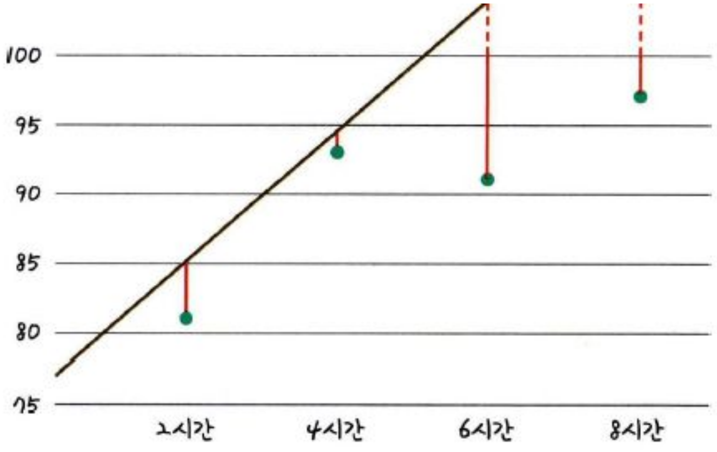


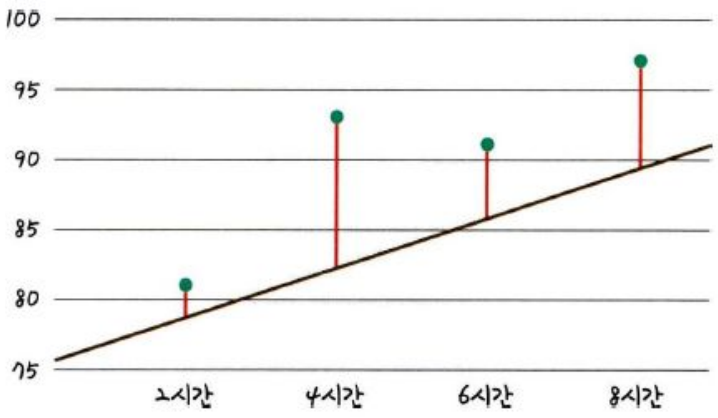


선형 회귀
MSE를 사용해 예측한 선과 점들의 오차를 계산한다.

로지스틱 회귀
두개의 log함수를 합친 Binary Cross Entropy를 사용해 오차를 계산한다.

03 Loss 구하기

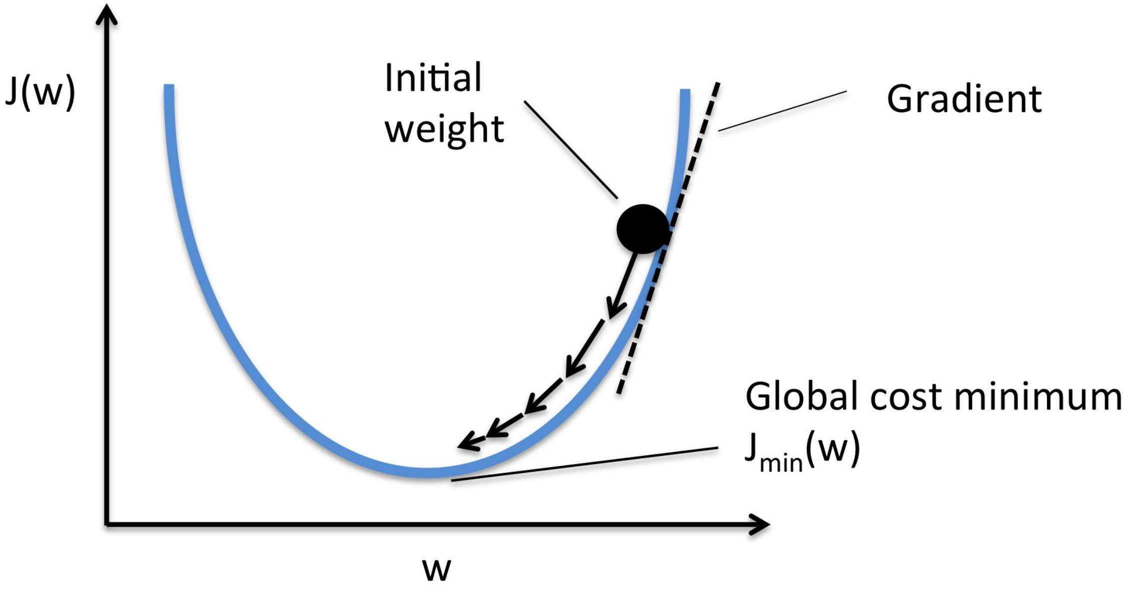




그런데 이렇게 선을 잘못 예측해 선과 점들 사이의 오차가 클 때

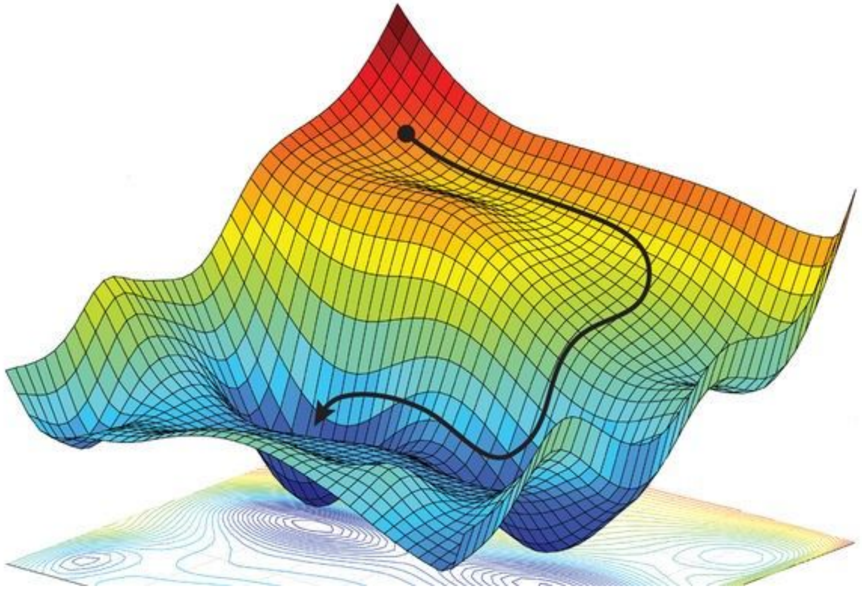
어떻게 해야 오차를 줄일 수 있을까?

04 Optimizer - 경사하강법(GD)



미분 기울기를 이용하는
경사하강법을 통해 오차를 가장 작은 방향으로 이동시킨다!

–>미분 값이 '0'인 지점 찾으면 정답!



SMARCLE

Thank you

QnA

SMARCLE