1차 함수를 컴퓨터가 구한다

경사하강법 풀이

동서대학교

최용균

1차 방정식에서 기울기를 구할 때

X data = 1, 2, 3, 4, 5

Y data = 2, 4, 6, 8, 10

$$f(x) = x * w + b \Rightarrow Y = x_1 * w_1 + b_1$$

우리는 X data(편의상 x로 표현) 와 Y data로 아래의 식을 유추 할 수 있다.

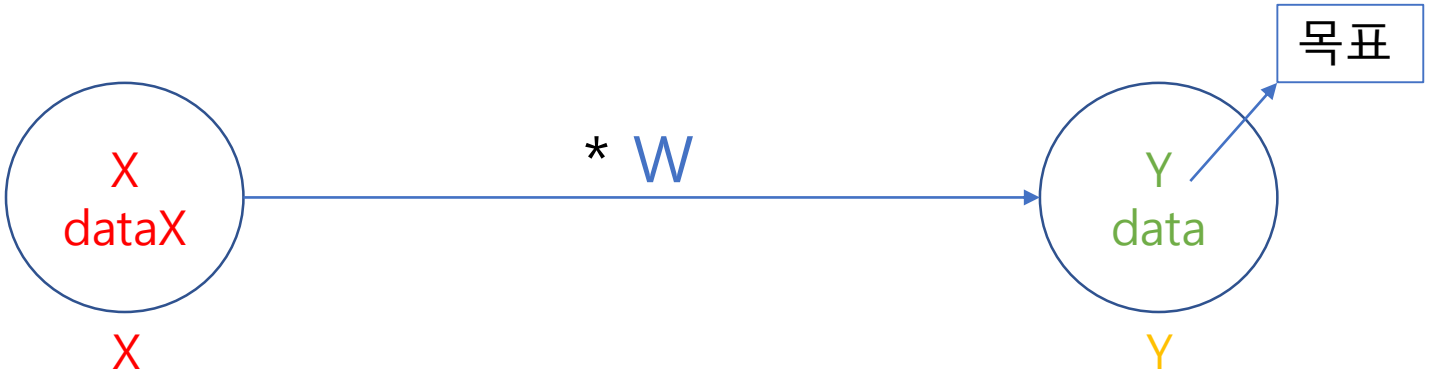
$$Y = x * 2 + 0$$

하지만 컴퓨터는 어떻게 답을 구할까?

$$f(x) = x * w + b$$

$$f(x) = Y$$

초기의 W와 b의 값이 2와 0이 아닐 경우
Y 와 Y data값은 달라요.



$$Y = x_1 * w_1 + b_1$$

처음의 W와 b의 값은 랜덤으로 주어진 상태일 때
Y data값이 나오기 위한 올바른 Y값을 계산하기 위해
W와 b를 업데이트 해주어야 할 필요가 있습니다.

W와 b의 값을 Y data가 나올 수 있도록!

W와 b의 값이 Y data가 나올 수 있도록 업데이트 한다는
생각으로 식을 구성하면 됩니다. 아래의 식 처럼 말이죠.

$$w_{new1} = w_1 + \Delta w_1$$

$$b_{new1} = b_1 + \Delta b_1$$

Δ ~와 관계가 있다는 의미입니다
W1과 관계가 있는 ,
b1과 관계가 있는

물론 이전의 값을 가지고 구해야만 관계가 있겠죠.

그렇다면 어떻게 W와 b값을 구할까

Y data의 값과 Y값의 차이가 0에 가까울 수록 W와 b의 값은 Y의 값이 Y data의 값에 가깝게 나오겠죠.

$$w_{new1} = w_1 + \Delta w_1$$

$$b_{new1} = b_1 + \Delta b_1$$

$$\Delta w_1, \Delta b_1$$

하지만 한번만으로는, 그리고 하나의 값만 가지고 학습 하는 것 보다 여러 번,여러 개의 값을 가지고 학습할 수록 좀더 원하는 목표의 W와 b의 값이 만들어 집니다.

학습할 때의 오차

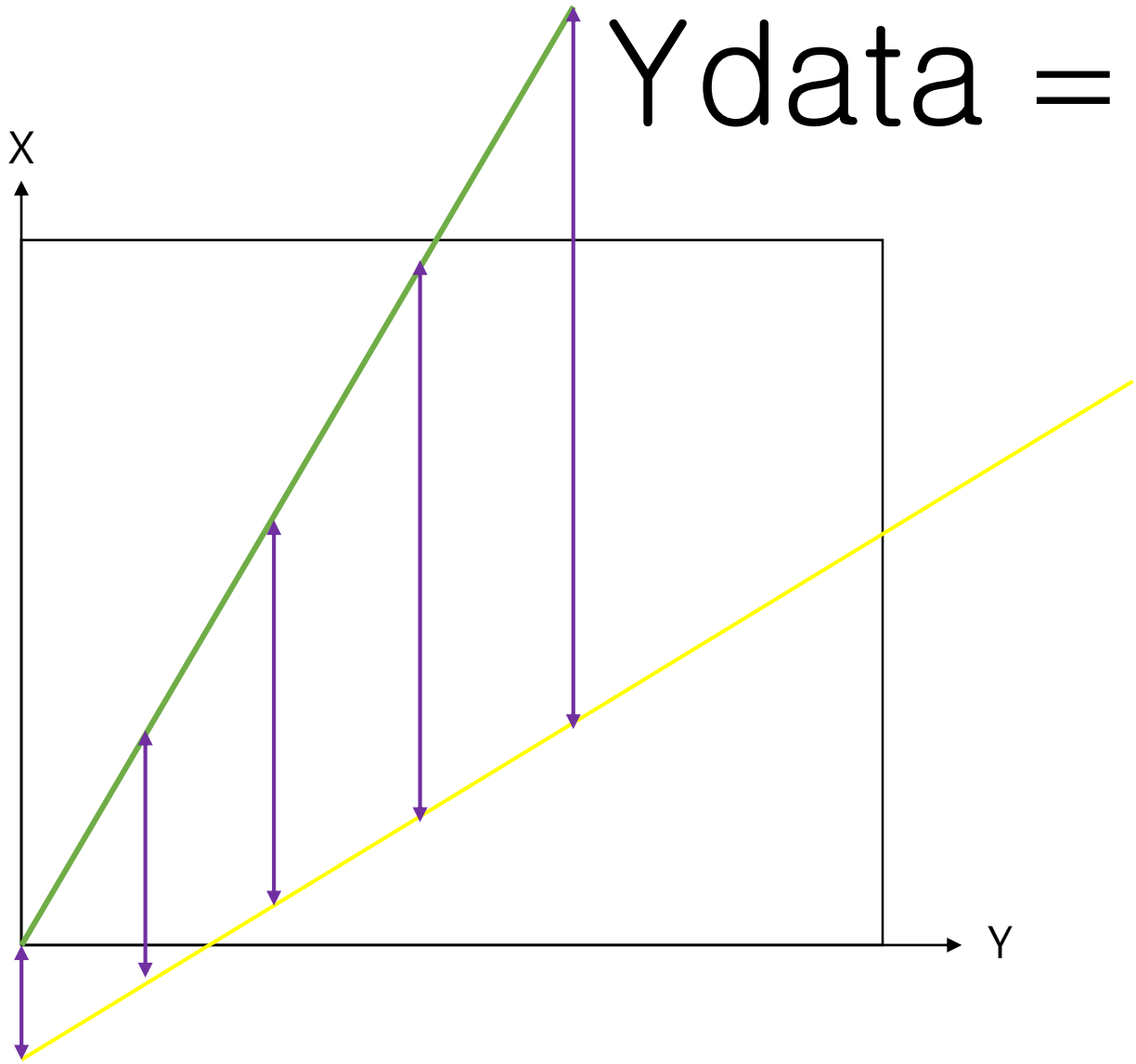
우리가 원하는 Y data의 값과
계산에 의해 나오는 Y의 값의 차이가 바로 오차입니다.

$$Loss = Y - Ydata$$

하지만 이것을 그래프위에서 보게 되면 그리고 우리의 1차 방정식은 X data와 Y data이기 때문에 2차원 그래프의 형태의 오차를 획득하게 됩니다.

오차 그래프

$$Ydata = x * 2 + 0$$



$$Y = x_1 * w_1 + b_1$$

우리가 원하는 목표함수는 초록색 선이고 현재의 학습해야 될 함수가 노란색 선입니다.

그리고 이 사이에 있는 보라색 선들이 바로 오차가 되는 거죠.

그렇다면 이 차이 만큼을 업데이트 시켜준다면 결국 우리가 원하는 값을 얻을 수 있습니다.

Y와 Y data간의 오차는?

Y와 Y data간의 오차를 구할 때는
차이가 너무 크거나 작을 경우를 대비해서 각 차이들의 평균을 구하여 적용하는게 효율적입니다.

해서 각 차이들의 평균을 분산이라고 말하는데 그 식은

$$Loss = \frac{1}{n}\sum (Y - Ydata)^2$$ 와 같습니다.

여기서 제곱이 들어가는 이유는 오차 크기를 구해야 하기때문에 –를 지우기 위함 입니다.

오차를 풀이하면

$$Loss = \frac{1}{n}\sum(Y - Ydata)^2$$

이 식을 분해하게 되면

$$Loss = \frac{1}{n}((Y_0 - Ydata_0)^2 + (Y_1 - Ydata_1)^2 + (Y_2 - Ydata_2)^2$$

$$\cdots + (Y_{n-1} - Ydata_{n-1})^2 + (Y_n - Ydata_n)^2)$$

가 됩니다.

그렇다면 Loss로 업데이트를?

우리가 구하기 위한 새로운 W와 b를 구하는 식은

$$w_{new1} = w_1 + \Delta w_1$$

$$b_{new1} = b_1 + \Delta b_1$$

인데 여기에 Loss의 값을 이용해서 업데이트를 시킨다면
Y의 값이 우리가 원하는 Y data에 가까워질 것 입니다.

어떻게 Loss를 이용해서 업데이트를 할까

Loss의 값을 그대로 사용하기 보다는 좀더 안전하게 사용하기 위해서 Loss값의 순간 변화량을 이용하는 것이 좋습니다.

해서 우리는 이 순간 변화량을 구하기 위해 미분을 쓰죠.
그렇습니다. 바로 경사하강법(Gradient Descent)입니다.

$$Loss = \frac{1}{n}\sum (Y - Ydata)^2$$

$$\frac{\Delta Loss}{\Delta w}, \frac{\Delta Loss}{\Delta b}$$

각각의 W에 대한 순간 변화량과
b에 대한 순간 변화량을 말이죠.

원하는 값에 대해 각각 미분한다

자 이제 미분을 해봅시다. $$Loss = \frac{1}{n}\sum(Y - Ydata)^2$$

$\frac{\Delta Loss}{\Delta w}$
$\frac{\Delta Loss}{\Delta b}$ 에 대한 미분은

$$\frac{\Delta Loss}{\Delta w} = \frac{1}{n}\sum 2*(x)((x*w+b)-Ydata)$$

$$\frac{\Delta Loss}{\Delta b} = \frac{1}{n}\sum 2*(1)((x*w+b)-Ydata)$$

이 되는데... 여기서 Y=x*w+b가 되게 되고 제곱인 부분이 앞으로 내려와
2가 곱해지고 제곱은 -1이 되어 사라집니다.
그리고 각각 '(x*w+b)-Ydata'에 대해서 W와 b를 나누면
2와 ((x*w+b)-Ydata)사이에 있는 'x'와 '1'이 남게 되어 곱해지게 됩니다.

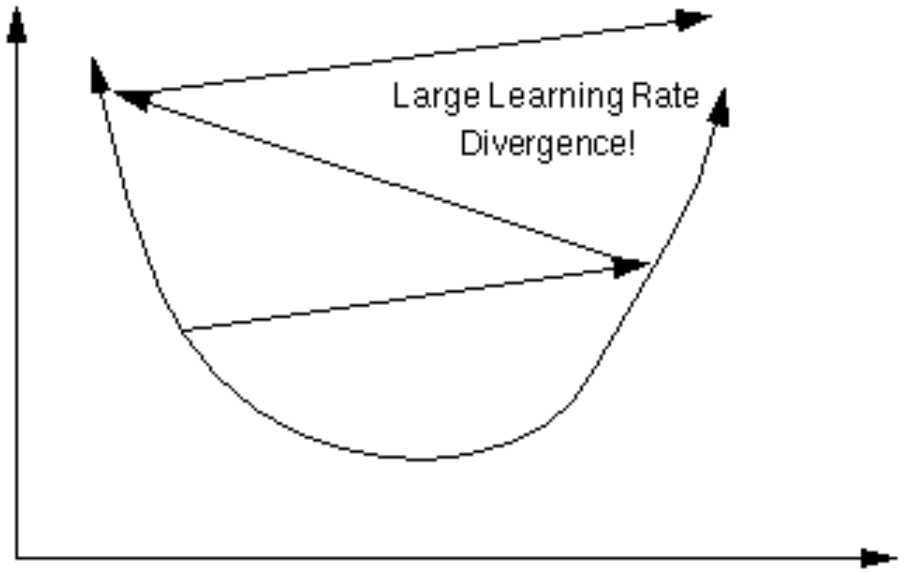
이제 적용해 봅시다

$$w_{new1} = w_1 - \frac{\Delta Loss}{\Delta w} = w_1 - \frac{1}{n}\sum 2 * (x)((x * w + b) - Ydata)$$

$$b_{new1} = b_1 - \frac{\Delta Loss}{\Delta w} = b_1 - \frac{1}{n}\sum 2 * (1)((x * w + b) - Ydata)$$

근데 여기서 하나 안전장치를 달아야 될 것 같아요.

오른쪽 그림처럼 업데이트의 값이 커버리면 안되기 때문에 한번 더 식을 고칠 필요가 있죠.



진짜 이제 적용해 봅시다

$$w_{new1} = w_1 - \frac{\Delta Loss}{\Delta w} = w_1 - \partial * \frac{1}{n}\sum(x) * ((x * w + b) - Ydata)$$

$$b_{new1} = b_1 - \frac{\Delta Loss}{\Delta w} = b_1 - \partial * \frac{1}{n}\sum(1) * ((x * w + b) - Ydata)$$

∂이 바로 학습데이터를 업데이트 할 때 그 크기를 결정 해주는 값이 됩니다. Learning Rate라고 합니다.

근데 이전 페이지의 식에서 2가 사라졌어요.
그것은 이미 Learning Rate에 흡수되어 일부러 적을 필요가 없어졌습니다.

Python tensorflow 를 이용한 구현

import tensorflow as tf (텐서플로 선언)
x_data = [1,2,3,4,5,6] (X data 선언)
y_data = [2,4,6,8,10,12] (Y data 선언)
W = tf.Variable(tf.random_uniform([1],-1.0,1.0)) (랜덤 W값 선언)
b = tf.Variable(tf.random_uniform([1],-1.0,1.0)) (랜덤 b값 선언)
Y = W * x_data + b (Y 식 선언)
loss = tf.reduce_mean(tf.square(Y-y_data)) (Loss값 , 분산 계산)
a = tf.Variable(0.01) (Learning Rate 선언)

Python tensorflow 를 이용한 구현

```
optimizer = tf.train.GradientDescentOptimizer(a)
 (경사하강법 사용, 미분)
Train = optimizer.minimize(loss)  (경사하강법을 이용해 loss값을 최소화로 목표함)

init = tf.global_variables_initializer()  (기본 텐서 선언)
sess = tf.Session() (기본 텐서 선언)
sess.run(init) (기본 텐서 선언)

for setp in range(400):      (400번 반복)
    sess.run(train)          (실제 실행부분)
    if setp % 20 ==0:        (20번마다)
        print(setp, sess.run(loss), sess.run(W), sess.run(b))  (값 출력)
```

실행 코드(python width tensorflow)

```
import tensorflow as tf
x_data = [1,2,3,4,5,6]
y_data = [2,4,6,8,10,12]

W = tf.Variable(tf.random_uniform([1],-1.0,1.0))
b = tf.Variable(tf.random_uniform([1],-1.0,1.0))

Y = W * x_data + b

loss = tf.reduce_mean(tf.square(Y-y_data))

a = tf.Variable(0.01)
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
for setp in range(400):
    sess.run(train)
    if setp % 20 ==0:
        print(setp, sess.run(loss), sess.run(W), sess.run(b))
```

실행결과: 횟수 : 오차: W : b

```
0 51.6884 [-0.03814193] [ 0.84283167]
20 0.260048 [ 1.72768188] [ 1.16229248]
40 0.224658 [ 1.74761534] [ 1.08050704]
60 0.194093 [ 1.76541197] [ 1.00431764]
80 0.167686 [ 1.78195345] [ 0.93350047]
100 0.144871 [ 1.79732847] [ 0.86767685]
120 0.125161 [ 1.81161952] [ 0.80649447]
140 0.108133 [ 1.82490277] [ 0.74962616]
160 0.0934208 [ 1.8372494] [ 0.69676787]
180 0.0807106 [ 1.84872532] [ 0.64763683]
200 0.0697296 [ 1.85939217] [ 0.60197002]
220 0.0602425 [ 1.8693068] [ 0.55952334]
240 0.0520463 [ 1.8785224] [ 0.52006978]
260 0.0449652 [ 1.88708818] [ 0.48339811]
280 0.0388476 [ 1.89504981] [ 0.44931236]
300 0.0335622 [ 1.9024502] [ 0.41763005]
320 0.0289959 [ 1.9093287] [ 0.38818178]
340 0.0250509 [ 1.91572225] [ 0.36080998]
360 0.0216427 [ 1.92166483] [ 0.33536825]
380 0.0186981 [ 1.9271884] [ 0.31172055]
```

```
import tensorflow as tf
x_data = [1,2,3,4,5,6]
y_data = [2,4,6,8,10,12]
W = tf.Variable(tf.random_uniform([1],-1.0,1.0))
b = tf.Variable(tf.random_uniform([1],-1.0,1.0))
Y = W * x_data + b
loss = tf.reduce_mean(tf.square(Y-y_data))
a = tf.Variable(0.01)

optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for setp in range(400):
    sess.run(train)
    if setp % 20 ==0:
        print(setp, sess.run(loss), sess.run(W), sess.run(b))
```