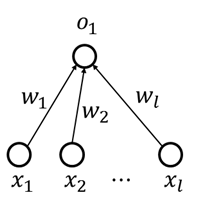
|  |
| --- |
| 인공지능(01) |
| Gradient descen학습과제 보고서 |
|  |

|  |
| --- |
| 컴퓨터공학과 2017154041 허민  2021.04.01 |

**1. 구현코드와 관련된 개념 요약 정리**

* **LMS(Least Mean Square) 최소자승법**



그림

최소 평균 제곱법은 위의 그림과 같은 단층 네트워크에서 오차 제곱 합의 평균(mean square errorm, MSE)을 최소화하는 가중치들을 찾는 방법이다. 흔히 모델의 파라미터를 구하기 위한 대표적인 방법 중 하나로서 모델과 데이터와의 resiual^2의 합 또는 평균을 최소화 하도록 파라미터를 결정하는 방법이다. 여기서 Resuial 이란 예측할려는 모델과 기존 데이터의 오차이다.



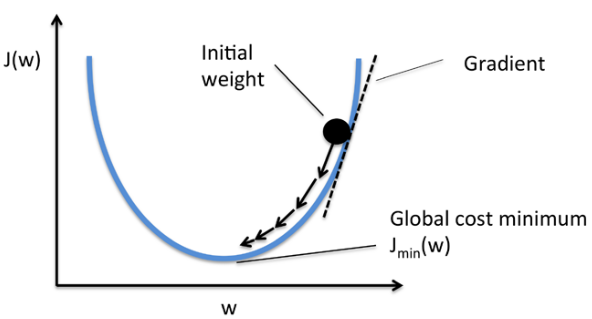
수식



수식

만약 f(x)가 선형함수 ax + b인 경우에는 수식2를 최소화 하도록 a와 b를 결정하는 것이 LMS 학습 법칙이다.

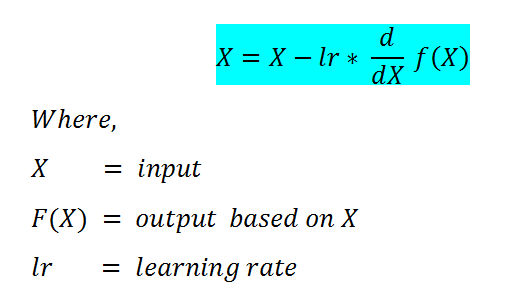
* **(Gradient Descent) 경사 강하법**



경사 강하법은 파라미터 w에 대한 error를 계산하는 함수(Loss Function)을 정의하고, 이 함수가 최소화 되는 방향을 찾아서 점점 나아가는 것이다.

수식

구체적으로는 우리는 MSE가 아닌 error function(수식 3)을 각 파라미터로 편미분하여, 함수인 각 파라미터에 대한 기울기를 구해 구해진 값에 learning Rate(학습률)을 곱한 후 차를 통해 각 파라미터를 갱신합니다. 이때 learning Rate를 0.1, 0.2로 정해 두 번의 실험을 통해서 학습률에 따른 학습의 차이에 대해 비교해보겠습니다.



수식

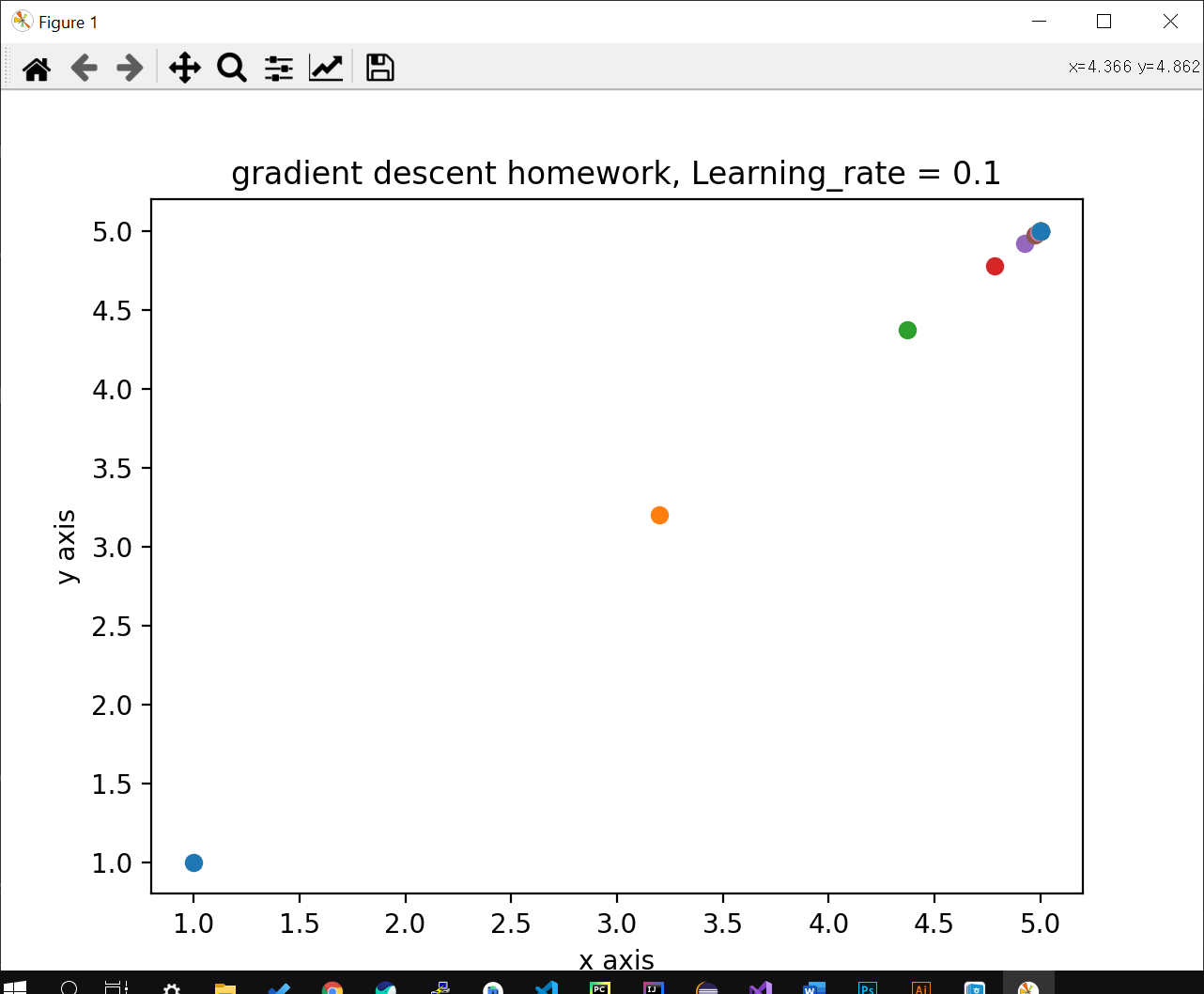
**2. 구현 코드(learning rate = 0.1)**

import numpy as np  
import matplotlib.pyplot as plt  
  
X = 1  
#초기 X 인풋값  
Y = 5  
#임의의 Y타겟값.  
#임의의 선형 수식은 Y = X로 구현  
plt.scatter(X,X)  
##Gradient Descent 구현  
#업데이트할 W : Learning Rate \* ((Y예측 - Y실제) \* X)/2  
#업데이트할 b : Learning Rate \* ((Y예측 - Y실제) \* 1)/2  
#초기 w값과 b값 지정  
W = 2  
b = 1  
  
#학습률은 0.1과 0.2로진행  
learning\_rate = 0.1  
learning\_rate2 = 0.2  
  
#에포크를 200번정도 돌려보면서 학습 시켜볼 예정이다.  
for epoch in range(200):  
 Y\_Pred = W \* X + b #예측값  
  
 error = np.abs(Y\_Pred - Y)/2  
 if Y\_Pred >= Y: #예측값이 타겟값보다 커질경우 학습을 종료한다.  
 break  
 #gradient descent 계산  
 w\_grad = learning\_rate \* ((Y\_Pred - Y)\*X)/2  
 b\_grad = learning\_rate \* ((Y\_Pred - Y))/2  
 #움직여야할 방향을 정한 것이다.  
  
 #W,b 값 갱신  
 W = W - w\_grad  
 b = b - b\_grad  
  
 # 10번째마다 학습현황을 파악  
 if epoch % 10 == 0:  
 Y\_Pred = W \* X + b  
 print("epoch------{0}".format(epoch))  
 print('y\_pred : {0} , y : {1}'.format(Y\_Pred,Y))  
 print('w\_grad : {0} , W : {1}'.format(w\_grad,W))  
 print('b\_grad : {0} , b : {1}'.format(b\_grad,b))  
 print('error : {0}'.format(error))  
 plt.scatter(Y\_Pred,Y\_Pred)  
  
plt.title('gradient descent homework, Learning\_rate = 0.1')  
plt.xlabel('x axis')  
plt.ylabel('y axis')  
plt.show()

**3. 구현 코드 설명**

초기 input값은 x, weight = 2, b = 1, learning Rate = 0.1로 지정하였다. 목표는 초기 input 값이 f(x) = x 의 선형 함수 위에서 임의의 타겟값인 y = 5 가 될 수 있도록 경사강하법을 통해 학습시키는 것이다. 200번의 학습 도중 예측값이 타겟값보다 커질경우 학습이 종료되게 설정하였고 200번 전체 과정의 변화율을 파악하는 것이 어려워 10번째마다 각 파라메터의 변화율을 볼 수 있도록 코드를 짰다. 그리고 예측값을 좌표평면 위에 찍어보면서 타겟값으로 이동이 되고 있는지 시각적으로 확인 해보았다.

**4. 구현 결과(learning Rate = 0.1)**



epoch------0

y\_pred : 3.2 , y : 5

w\_grad : -0.1 , W : 2.1

b\_grad : -0.1 , b : 1.1

error : 1.0

epoch------10

y\_pred : 4.37237880782 , y : 5

w\_grad : -0.03486784401 , W : 2.68618940391

b\_grad : -0.03486784401 , b : 1.68618940391

error : 0.34867844010000004

epoch------20

y\_pred : 4.781162021736975 , y : 5

w\_grad : -0.012157665459056943 , W : 2.8905810108684875

b\_grad : -0.012157665459056943 , b : 1.890581010868488

error : 0.12157665459056943

epoch------30

y\_pred : 4.923695915104611 , y : 5

w\_grad : -0.004239115827521634 , W : 2.961847957552305

b\_grad : -0.004239115827521634 , b : 1.9618479575523056

error : 0.042391158275216334

epoch------40

y\_pred : 4.973394410705418 , y : 5

w\_grad : -0.0014780882941434648 , W : 2.986697205352709

b\_grad : -0.0014780882941434648 , b : 1.9866972053527092

error : 0.014780882941434648

epoch------50

y\_pred : 4.990723204626824 , y : 5

w\_grad : -0.0005153775207320078 , W : 2.9953616023134115

b\_grad : -0.0005153775207320078 , b : 1.9953616023134124

error : 0.005153775207320077

epoch------60

y\_pred : 4.996765381460154 , y : 5

w\_grad : -0.00017970102999145256 , W : 2.998382690730077

b\_grad : -0.00017970102999145256 , b : 1.9983826907300772

error : 0.0017970102999145254

epoch------70

y\_pred : 4.998872158253208 , y : 5

w\_grad : -6.265787482178098e-05 , W : 2.999436079126604

b\_grad : -6.265787482178098e-05 , b : 1.9994360791266041

error : 0.0006265787482178098

epoch------80

y\_pred : 4.999606745899049 , y : 5

w\_grad : -2.1847450052847606e-05 , W : 2.9998033729495246

b\_grad : -2.1847450052847606e-05 , b : 1.9998033729495246

error : 0.00021847450052847606

epoch------90

y\_pred : 4.9998628807735175 , y : 5

w\_grad : -7.6177348045547434e-06 , W : 2.999931440386759

b\_grad : -7.6177348045547434e-06 , b : 1.9999314403867585

error : 7.617734804554743e-05

epoch------100

y\_pred : 4.999952189482003 , y : 5

w\_grad : -2.656139888745912e-06 , W : 2.999976094741002

b\_grad : -2.656139888745912e-06 , b : 1.9999760947410012

error : 2.656139888745912e-05

epoch------110

y\_pred : 4.999983329503165 , y : 5

w\_grad : -9.261387130976573e-07 , W : 2.9999916647515827

b\_grad : -9.261387130976573e-07 , b : 1.999991664751582

error : 9.261387130976573e-06

epoch------120

y\_pred : 4.999994187357168 , y : 5

w\_grad : -3.229246017699694e-07 , W : 2.9999970936785845

b\_grad : -3.229246017699694e-07 , b : 1.9999970936785838

error : 3.229246017699694e-06

epoch------130

y\_pred : 4.9999979732567645 , y : 5

w\_grad : -1.1259684642439539e-07 , W : 2.9999989866283823

b\_grad : -1.1259684642439539e-07 , b : 1.999998986628382

error : 1.1259684642439538e-06

epoch------140

y\_pred : 4.99999929331833 , y : 5

w\_grad : -3.9260092776771675e-08 , W : 2.999999646659165

b\_grad : -3.9260092776771675e-08 , b : 1.9999996466591652

error : 3.926009277677167e-07

epoch------150

y\_pred : 4.999999753595338 , y : 5

w\_grad : -1.3689147904116795e-08 , W : 2.999999876797669

b\_grad : -1.3689147904116795e-08 , b : 1.9999998767976688

error : 1.3689147904116794e-07

epoch------160

y\_pred : 4.9999999140840075 , y : 5

w\_grad : -4.773110706679518e-09 , W : 2.999999957042004

b\_grad : -4.773110706679518e-09 , b : 1.999999957042003

error : 4.7731107066795175e-08

epoch------170

y\_pred : 4.999999970042945 , y : 5

w\_grad : -1.6642808020606027e-09 , W : 2.9999999850214736

b\_grad : -1.6642808020606027e-09 , b : 1.9999999850214718

error : 1.6642808020606026e-08

epoch------180

y\_pred : 4.999999989554621 , y : 5

w\_grad : -5.802988312098023e-10 , W : 2.999999994777312

b\_grad : -5.802988312098023e-10 , b : 1.9999999947773095

error : 5.802988312098023e-09

epoch------190

y\_pred : 4.999999996357922 , y : 5

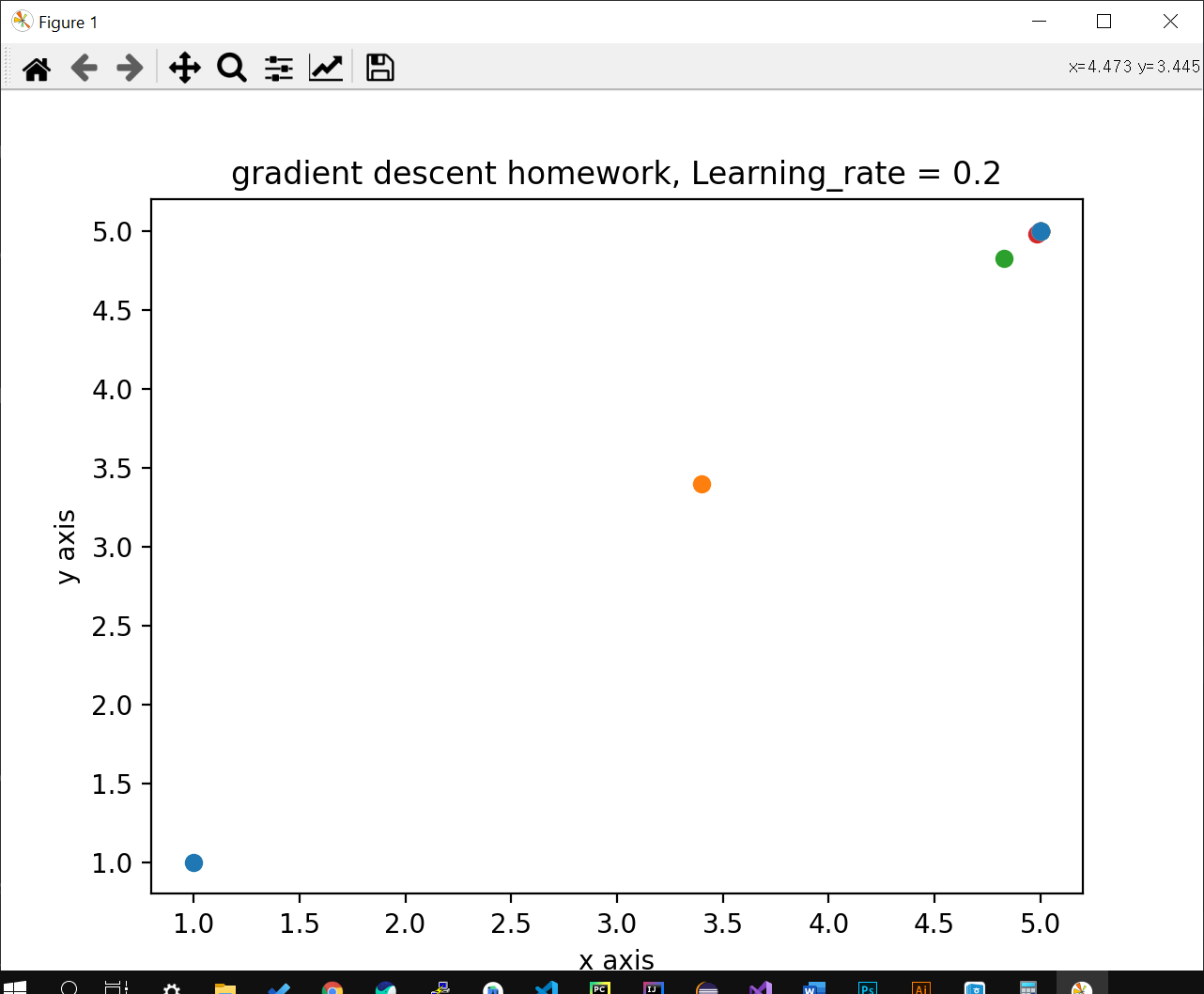
w\_grad : -2.0233765773980395e-10 , W : 2.9999999981789625

b\_grad : -2.0233765773980395e-10 , b : 1.9999999981789596

error : 2.0233765773980394e-09

200번의 학습결과 그래프를 보게 되면 학습이 잘 진행되어 에러값이 줄어들면서 초기값 (1,1) 에서 목표 값인 (5,5)로 잘 이동하고 있는 것을 볼 수 있다. 최종적인 예측값을 보게되면 : 4.999999996357922의 값을 얻어낸 것을 알 수 있다.

**4. 구현 결과(learning Rate = 0.2)**



epoch------0

y\_pred : 3.4000000000000004 , y : 5

w\_grad : -0.2 , W : 2.2

b\_grad : -0.2 , b : 1.2

error : 1.0

epoch------10

y\_pred : 4.82820130816 , y : 5

w\_grad : -0.021474836480000016 , W : 2.91410065408

b\_grad : -0.021474836480000016 , b : 1.9141006540799999

error : 0.10737418240000007

epoch------20

y\_pred : 4.981553255926291 , y : 5

w\_grad : -0.0023058430092136816 , W : 2.9907766279631454

b\_grad : -0.0023058430092136816 , b : 1.9907766279631451

error : 0.011529215046068408

epoch------30

y\_pred : 4.9980192959371434 , y : 5

w\_grad : -0.00024758800785704696 , W : 2.999009647968572

b\_grad : -0.00024758800785704696 , b : 1.9990096479685713

error : 0.0012379400392852347

epoch------40

y\_pred : 4.999787323520674 , y : 5

w\_grad : -2.658455991575437e-05 , W : 2.9998936617603373

b\_grad : -2.658455991575437e-05 , b : 1.9998936617603367

error : 0.00013292279957877184

epoch------50

y\_pred : 4.999977164036917 , y : 5

w\_grad : -2.854495385484768e-06 , W : 2.9999885820184584

b\_grad : -2.854495385484768e-06 , b : 1.999988582018458

error : 1.4272476927423838e-05

epoch------60

y\_pred : 4.999997548007135 , y : 5

w\_grad : -3.0649910822688756e-07 , W : 2.9999987740035676

b\_grad : -3.0649910822688756e-07 , b : 1.9999987740035672

error : 1.5324955411344376e-06

epoch------70

y\_pred : 4.99999973671927 , y : 5

w\_grad : -3.291009118555621e-08 , W : 2.999999868359635

b\_grad : -3.291009118555621e-08 , b : 1.9999998683596352

error : 1.6455045592778106e-07

epoch------80

y\_pred : 4.999999971730447 , y : 5

w\_grad : -3.533694137303201e-09 , W : 2.9999999858652235

b\_grad : -3.533694137303201e-09 , b : 1.9999999858652238

error : 1.7668470686516002e-08

epoch------90

y\_pred : 4.999999996964581 , y : 5

w\_grad : -3.794275116320023e-10 , W : 2.99999999848229

b\_grad : -3.794275116320023e-10 , b : 1.9999999984822905

error : 1.8971375581600114e-09

epoch------100

y\_pred : 4.999999999674075 , y : 5

w\_grad : -4.0740655293802776e-11 , W : 2.9999999998370375

b\_grad : -4.0740655293802776e-11 , b : 1.9999999998370372

error : 2.0370327646901387e-10

epoch------110

y\_pred : 4.999999999965004 , y : 5

w\_grad : -4.374545170549027e-12 , W : 2.999999999982502

b\_grad : -4.374545170549027e-12 , b : 1.9999999999825018

error : 2.1872725852745134e-11

epoch------120

y\_pred : 4.999999999996243 , y : 5

w\_grad : -4.696687483374263e-13 , W : 2.9999999999981215

b\_grad : -4.696687483374263e-13 , b : 1.999999999998121

error : 2.348343741687131e-12

epoch------130

y\_pred : 4.999999999999597 , y : 5

w\_grad : -5.044853423896712e-14 , W : 2.999999999999799

b\_grad : -5.044853423896712e-14 , b : 1.999999999999798

error : 2.5224267119483557e-13

epoch------140

y\_pred : 4.9999999999999565 , y : 5

w\_grad : -5.417888360170764e-15 , W : 2.9999999999999787

b\_grad : -5.417888360170764e-15 , b : 1.9999999999999778

error : 2.708944180085382e-14

epoch------150

y\_pred : 4.999999999999995 , y : 5

w\_grad : -5.329070518200752e-16 , W : 2.9999999999999982

b\_grad : -5.329070518200752e-16 , b : 1.999999999999997

error : 2.6645352591003757e-15

epoch------160

y\_pred : 4.999999999999999 , y : 5

w\_grad : -8.881784197001253e-17 , W : 2.9999999999999996

b\_grad : -8.881784197001253e-17 , b : 1.9999999999999993

error : 4.440892098500626e-16

epoch------170

y\_pred : 4.999999999999999 , y : 5

w\_grad : -8.881784197001253e-17 , W : 2.9999999999999996

b\_grad : -8.881784197001253e-17 , b : 1.9999999999999993

error : 4.440892098500626e-16

epoch------180

y\_pred : 4.999999999999999 , y : 5

w\_grad : -8.881784197001253e-17 , W : 2.9999999999999996

b\_grad : -8.881784197001253e-17 , b : 1.9999999999999993

error : 4.440892098500626e-16

epoch------190

y\_pred : 4.999999999999999 , y : 5

w\_grad : -8.881784197001253e-17 , W : 2.9999999999999996

b\_grad : -8.881784197001253e-17 , b : 1.9999999999999993

error : 4.440892098500626e-16

구현 결과 학습률이 0.1이 였을때의 최종 예측값은 4.999999996357922이였는데 비해 학습률이 0.2일 경우 최종 예측값은 4.999999999999999의 결과를 보여주고 있다. 또한 오차값도 학습률이 0.1 일떈 2.0233765773980394e-09 였지만 0.2일떈 4.440892098500626e-16의 아주 작은 수치가 되었음을 알 수 있었다.

|  |  |  |  |
| --- | --- | --- | --- |
| **학습률** | **학습 횟수(회)** | **학습 최종 에러값** | **학습 최종 예측값** |
| 0.1 | 200 | 2.0233765773980394e-09 | 4.999999996357922 |
| 0.2 | 200 | 4.440892098500626e-16 | 4.999999999999999 |

**5. 마무리하며**

그렇다면 왜 학습률에 따라 예측값과 에러값의 차이가 나는걸까? 경사 강하법에선 시작점에서 다음 점으로 이동할 때 학습률만큼 이동한 점을 다음점으로 정한다. 즉 학습률이란 현재 점에서 다음 점으로 얼만큼 이동할지, 모델이 얼마난 세세하게 학습을 할지를 의미한다.

학습률이 너무 작다면 손실이 최적인 가중치를 찾는데 오랜 시간이 걸릴 것이고, 학습률이 너무 크다면 최적점을 무질서하게 이탈할 수 도 있다. 따라서 손실을 줄이고 올바른 학습을 하기 위해선 적절한 학습률을 찾는데 노력해야한다.