# AI 프로그래밍

- 3주차

인하 공전 컴퓨터 정보과

**민 정혜**

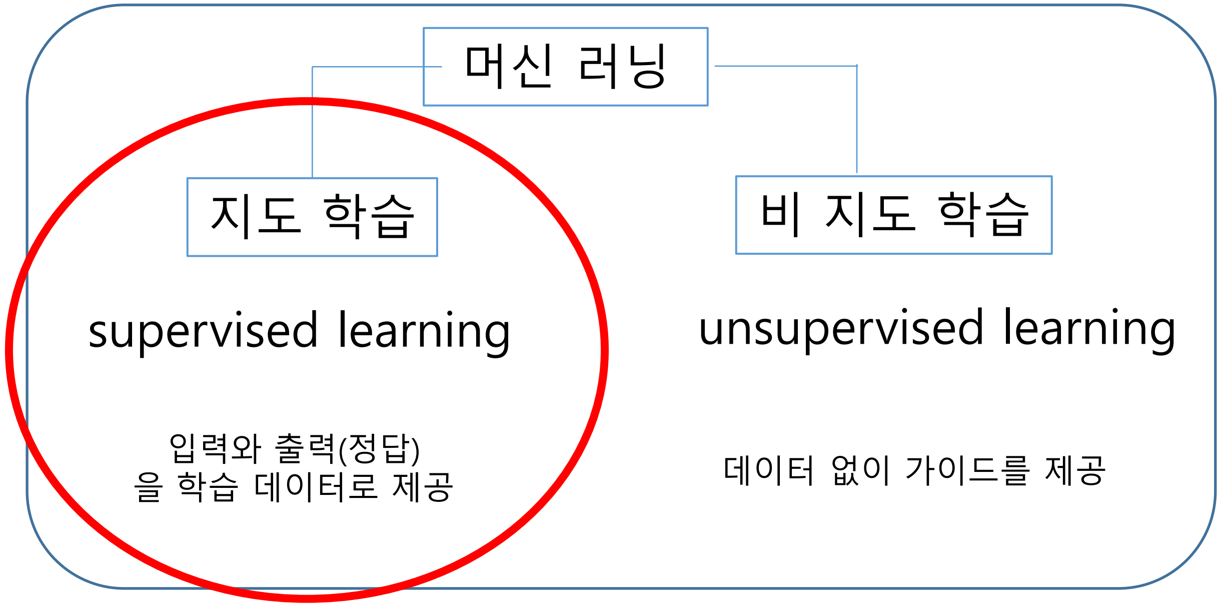
# 오늘 수업 순서



- 지난 주 내용 복습
- 선형 회귀 실습 (linear regression)
- 경사하강법

# 머신 러닝

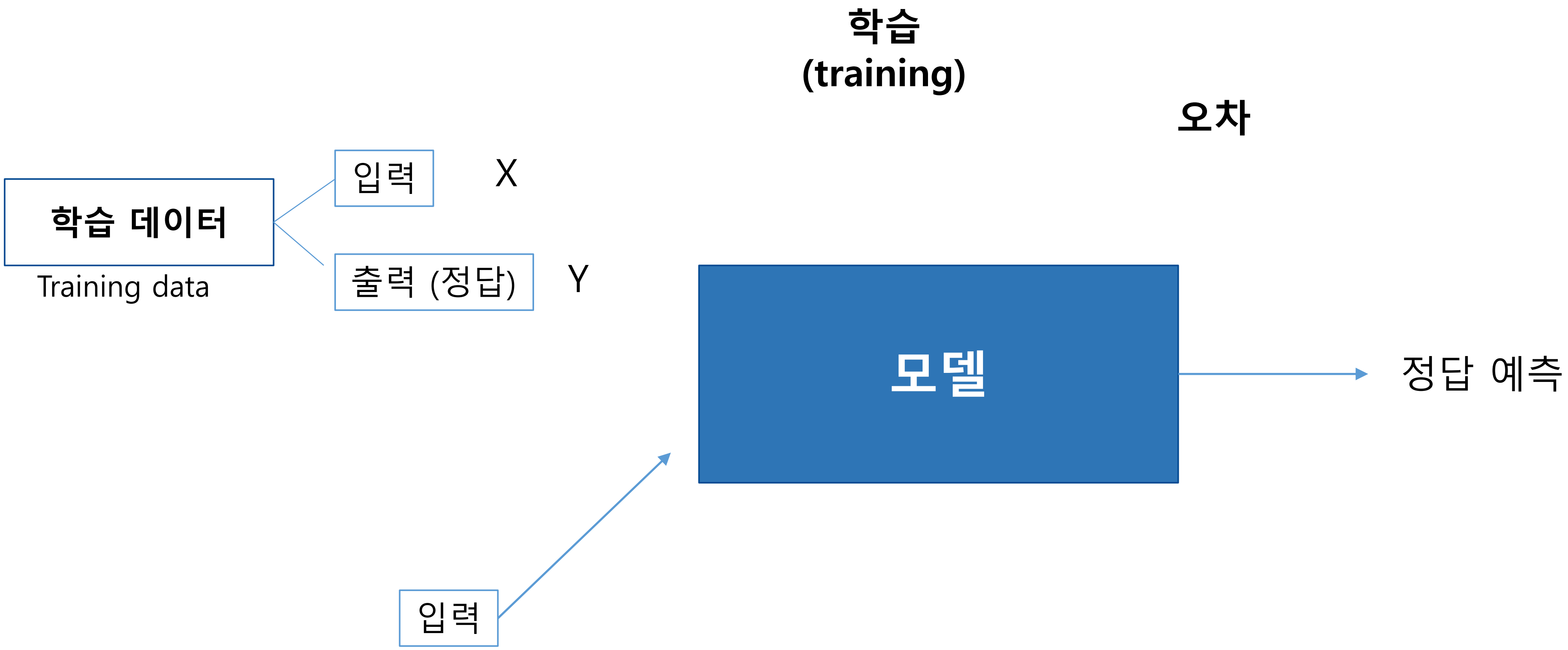



머신 러닝
지도 학습
비 지도 학습
supervised learning
unsupervised learning
입력와 출력(정답)
을 학습 데이터로 제공
데이터 없이 가이드를 제공


머신러닝
지도 학습
비지도 학습
회귀
• Linear
• polynomial
클러스터링
• SVD
• PAC
• K-means
판단트리
랜덤 포리스트
분류
• KNN
• Trees
• logistic Regression
• Naive-Bayes
• SVM

# 머신 러닝





# 머신 러닝 - 예제



**학습**
**(training)**

**학습 데이터**
Training data

입력 X

출력 (정답) Y

| 키(단위: cm) | 몸무게(단위: kg) |
|---|---|
| 174 | 71 |
| 152 | 55 |
| 138 | 46 |
| 128 | 38 |
| 186 | 88 |

**오차= 정답과 예측값의 차이**

**모델**

정답 예측

몸무게는?

입력

키가 165

# 회귀 (regression)



- **데이터를 이용하여 결과를 예측 하는 함수를 도출 y=f(x)**
- **선형 회귀**
  - 입력 값 x가 1차원일때
  - 직선의 방정식
  - Y=ax+b

⇒Y =wx+b,     w: weight, b: bias

| 키(단위: cm) | 몸무게(단위: kg) |
|---|---|
| 174 | 71 |
| 152 | 55 |
| 138 | 46 |
| 128 | 38 |
| 186 | 88 |

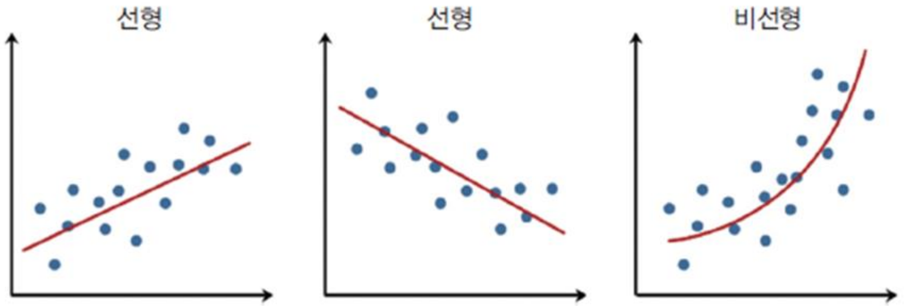



그림 4-2 회귀의 종류

참고자료 : 딥러닝 express

| 공부한 시간 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| 성적 | 81 | 93 | 91 | 97 |
| **예측 값** | 83.6 | 88.2 | 92.8 | 97.4 |

# 선형 회귀 예제



- 공부한 시간(x)과 성적 (y)
- Y=wx+b 로 예측
- 오차를 최소화 하는 w와 b 찾기

| 공부한 시간 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| 성적 | 81 | 93 | 91 | 97 |
| 예측 값 | 83.6 | 88.2 | 92.8 | 97.4 |

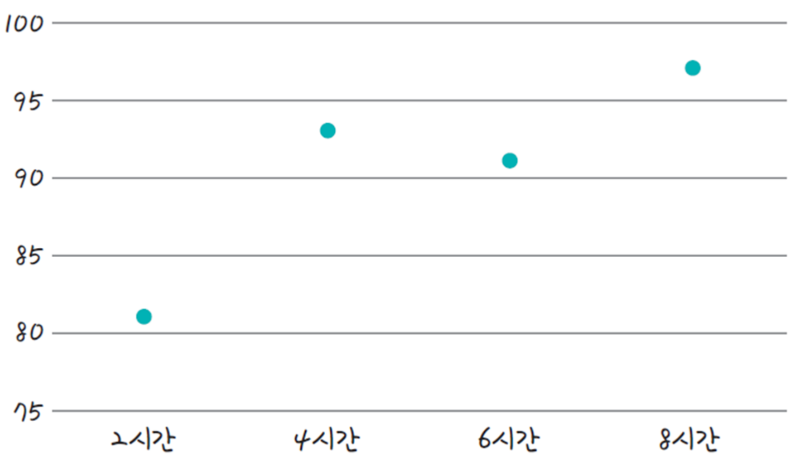



그림 3-4 공부한 시간과 성적의 관계도

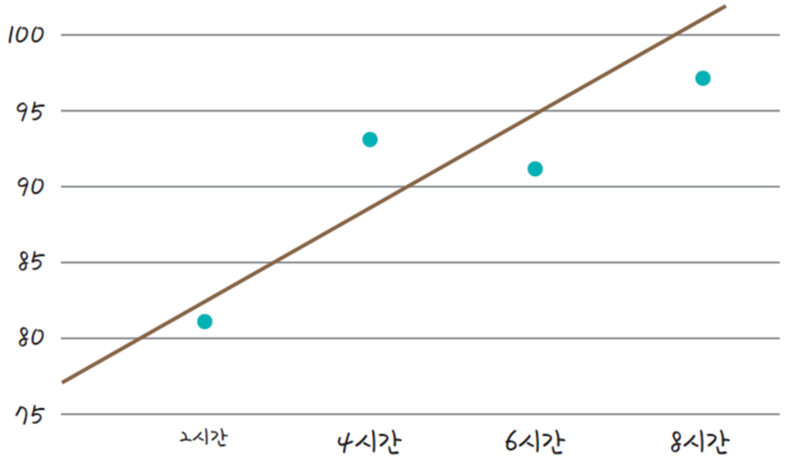



그림 3-5 임의의 직선 그려보기

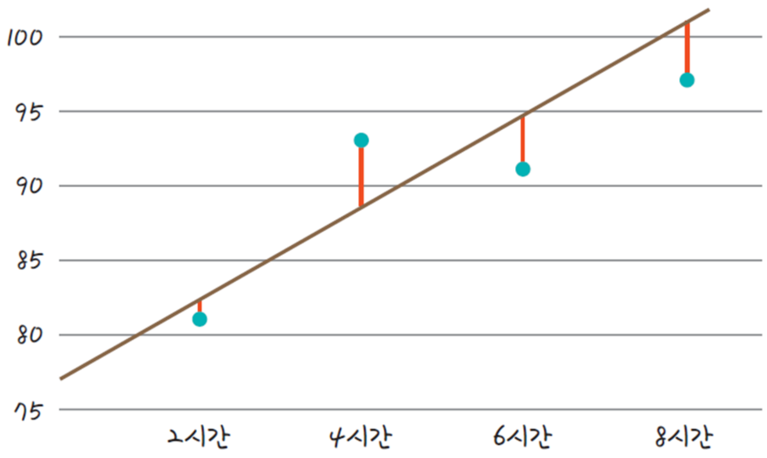



그림 3-6 임의의 직선과 실제 값 사이의 거리

오차

# 선형 회귀 예제



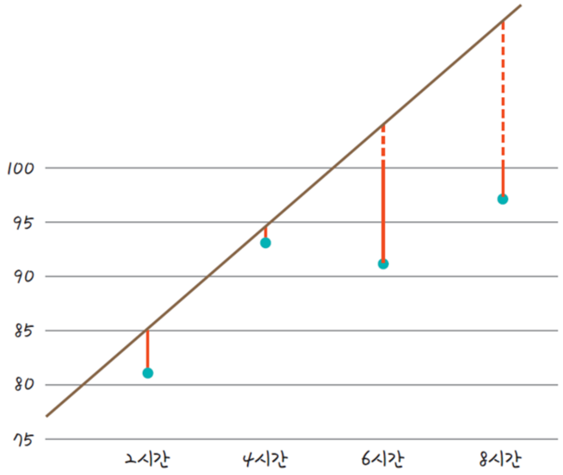



그림 3-7 기울기를 너무 크게 잡았을 때의 오차

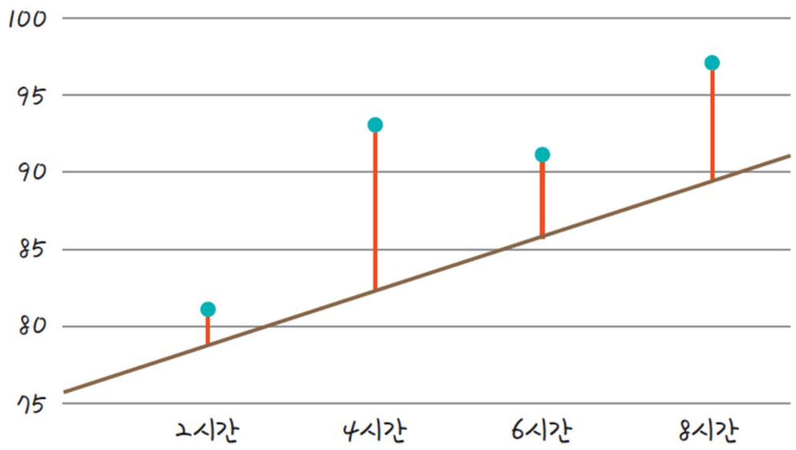



그림 3-8 기울기를 너무 작게 잡았을 때의 오차

# 선형 회귀 예제 실습



| x | Y |
|---|---|
| 0 | 3 |
| 1 | 3.5 |
| 2 | 5.5 |
| | |

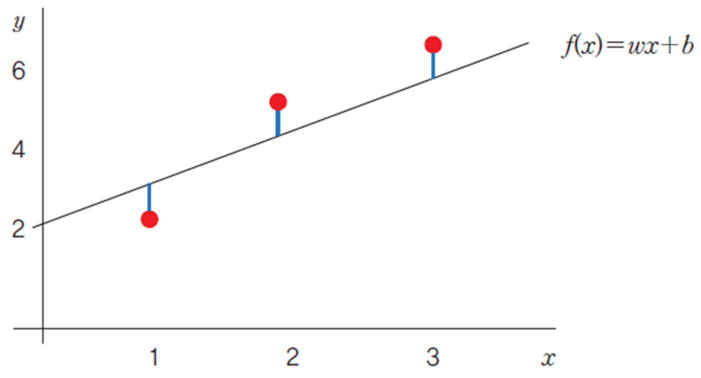



그림 4–5 데이터와 직선 간의 거리

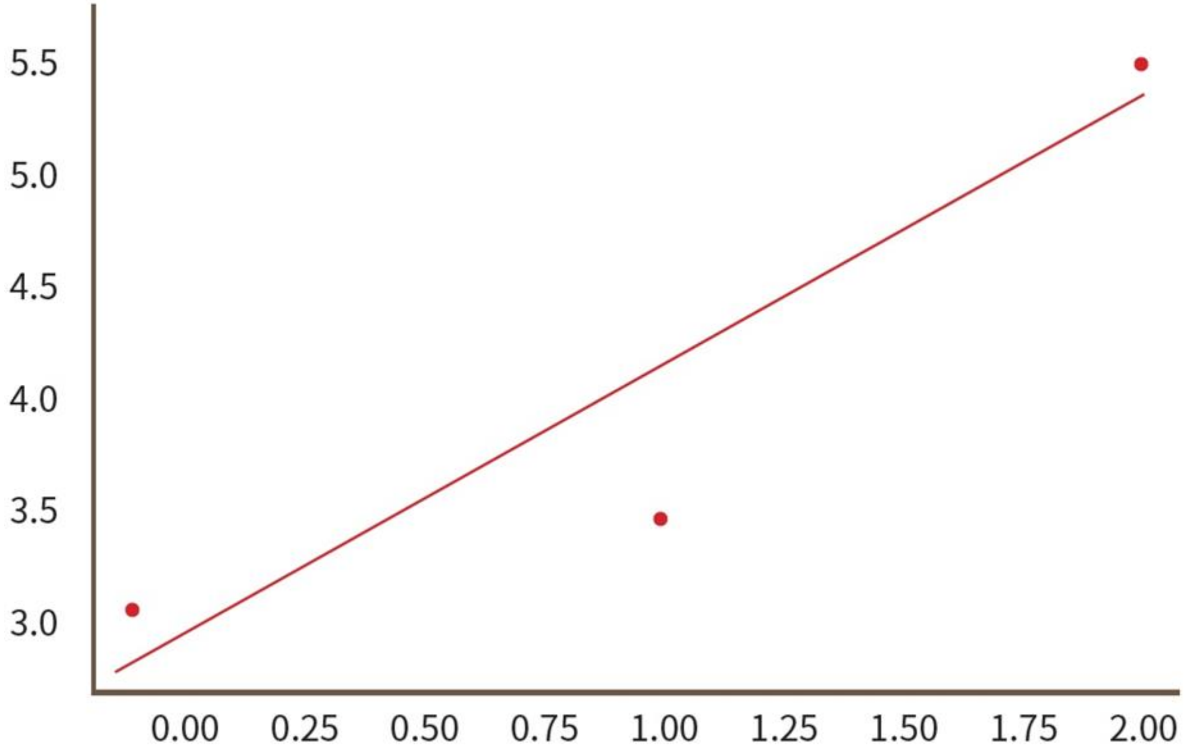
```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

w = 0       # 기울기
b = 0       # 절편

lrate = 0.01  # 학습률
epochs = 1000  # 반복 횟수

n = float(len(X)) # 입력 데이터의 개수

# 경사 하강법
for i in range(epochs):
    y_pred = w*X + b                          # 선형 회귀 예측값
    dw = (2/n) * sum(X * (y_pred-y))    # 넘파이 배열간의 산술 계산은 요소별로 적용
    db = (2/n) * sum(y_pred-y)          # sum()은 모든 요소들의 합을 계산하는 내장 함수
    w = w - lrate * dw                  # 기울기 수정
    b = b - lrate * db                  # 절편 수정

# 기울기와 절편을 출력한다.
print (w, b)

# 예측값을 만든다.
y_pred = w*X + b

# 입력 데이터를 그래프 상에 찍는다.
plt.scatter(X, y)

# 예측값은 선그래프로 그린다.
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
plt.show()
```

참고자료 : 딥러닝 express

# 선형 회귀 예제



| x | Y |
|---|---|
| 0 | 3 |
| 1 | 3.5 |
| 2 | 5.5 |
| | |




Y=1.252*x+2.745

X=3, 6.5

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


def mse(y,y_hat):
    return ((y-y_hat)**2).mean()

def mse_val(y,predict_result):
  return mse(np.array(y),np.array(predict_result))



X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

w = 0        # 기울기
b = 0        # 절편

lrate = 0.01  # 학습률
epochs = 500  # 반복 횟수

n = float(len(X)) # 입력 데이터의 개수
```

```
# 경사 하강법
for i in range(epochs):
    y_pred = w*X + b         # 선형 회귀 예측값
    dw = (2/n) * sum(X * (y_pred-y))  # 넘파이 배열간의 산술 계산은 요소별로 적용
    db = (2/n) * sum(y_pred-y)     # sum()은 모든 요소들의 합을 계산하는 내장 함수
    w = w - lrate * dw       # 기울기 수정
    b = b - lrate * db       # 절편 수정
    if (i%50==0):
      print('iteration %3d: loss  %4.2f w %3.2f b %3.2f '%(i,mse(y,y_pred),w,b))


# 기울기와 절편을 출력한다.
print ('########## final w,b',w, b)

# 예측값을 만든다.
y_pred = w*X + b


# 입력 데이터를 그래프 상에 찍는다.
plt.scatter(X, y)

# 예측값은 선그래프로 그린다.
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
plt.show()
```

첫번째 loss 값과 w,b 값을 저에게 채팅으로 보내 주세요.

```
for i in range(epochs):
    y_pred = w*X + b          # 선형 회귀 예측값
    dw = (2/n) * sum(X * (y_pred-y))  # 넘파이 배열간
의 산술 계산은 요소별로 적용
    db = (2/n) * sum(y_pred-y)     # sum()은 모든 요소
들의 합을 계산하는 내장 함수
    w = w - lrate * dw       # 기울기 수정
    b = b - lrate * db       # 절편 수정
    if (i%50==0):
      print('iteration %3d: loss  %4.2f w %3.2f b %3.2f
'%(i,mse(y,y_pred),w,b))
```

```
iteration 0: loss 17.17 w 0.10 b 0.08
iteration 50: loss 0.61 w 1.73 b 1.71
iteration 100: loss 0.33 w 1.73 b 2.05
iteration 150: loss 0.24 w 1.63 b 2.23
iteration 200: loss 0.19 w 1.53 b 2.36
iteration 250: loss 0.16 w 1.47 b 2.45
iteration 300: loss 0.15 w 1.41 b 2.52
iteration 350: loss 0.14 w 1.37 b 2.58
iteration 400: loss 0.13 w 1.34 b 2.62
iteration 450: loss 0.13 w 1.32 b 2.65
########## final w,b 1.3033228991130752
2.6760184293088694


1.25 2.74
```

# 선형 회귀 예제 실습



| x | Y |
|---|---|
| 0 | 3 |
| 1 | 3.5 |
| 2 | 5.5 |
| | |

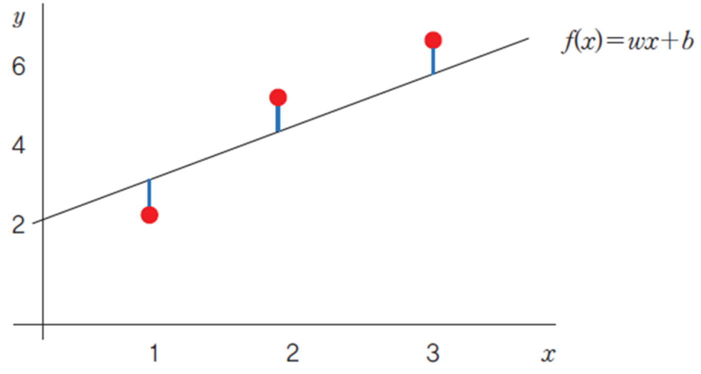


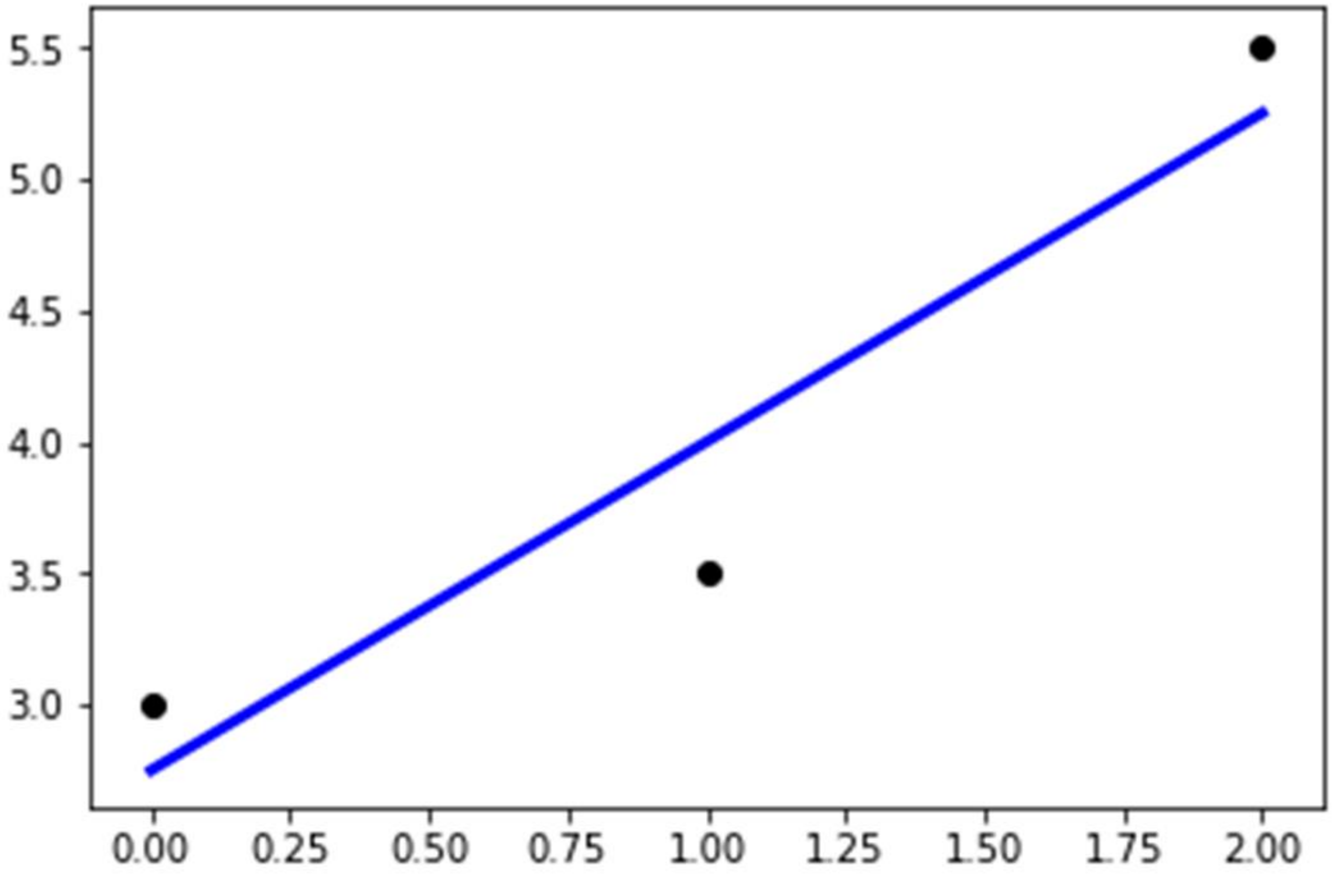

그림 4-5 데이터와 직선 간의 거리

```
import matplotlib.pylab as plt
from sklearn import linear_model

# 선형 회귀 모델을 생성한다.
reg = linear_model.LinearRegression()

# 데이터는 파이썬의 리스트로 만들어도 되고 아니면 넘파이의 배열로 만들어도 됨
X = [[0], [1], [2]]                    # 반드시 2차원으로 만들어야 함
y = [3, 3.5, 5.5]                      # y = x + 3

# 학습을 시킨다.
reg.fit(X, y)

print(reg.coef_)                       # 직선의 기울기
print(reg.intercept_)          # 직선의 y-절편
print(reg.score(X, y))

print(reg.predict([[5]]))

# 학습 데이터와 y 값을 산포도로 그린다.
plt.scatter(X, y, color='black')

# 학습 데이터를 입력으로 하여 예측값을 계산한다.
y_pred = reg.predict(X)

# 학습 데이터와 예측값으로 선그래프로 그린다.
# 계산된 기울기와 y 절편을 가지는 직선이 그려진다.
plt.plot(X, y_pred, color='blue', linewidth=3)
plt.show()
```

참고자료 : 딥러닝 express

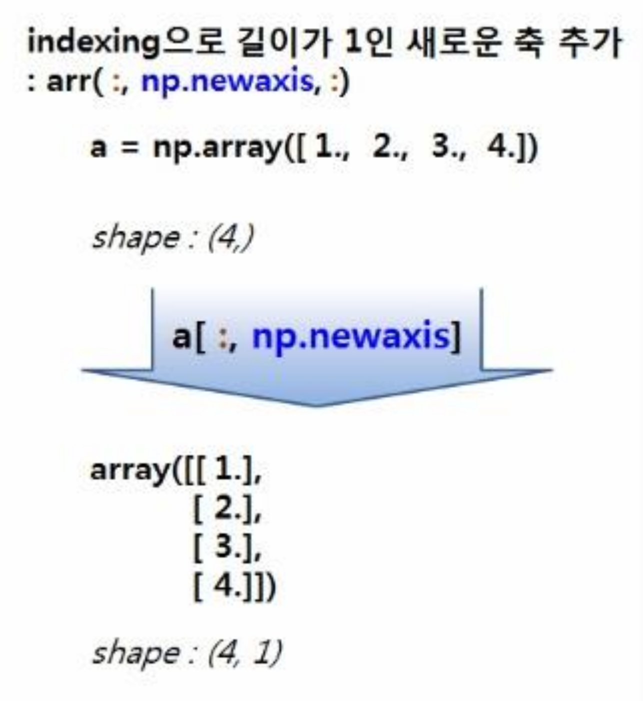
# 선형 회귀 예제





# Numpy



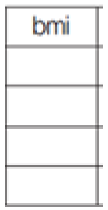



```python
import numpy as np

arr = np.array([1, 2, 3, 4])
arr1=arr
arr2=arr[np.newaxis]
arr3=arr[:, np.newaxis]
print(arr1,arr1.shape)
print(arr2,arr2.shape)
print(arr3,arr3.shape)
```

```
arr1 [1 2 3 4] (4,)
arr2 [[1 2 3 4]] (1, 4)
arr3 [[1] [2] [3] [4]] (4, 1)
```

# 선형 회귀 예제 실습 – 당뇨병 예제



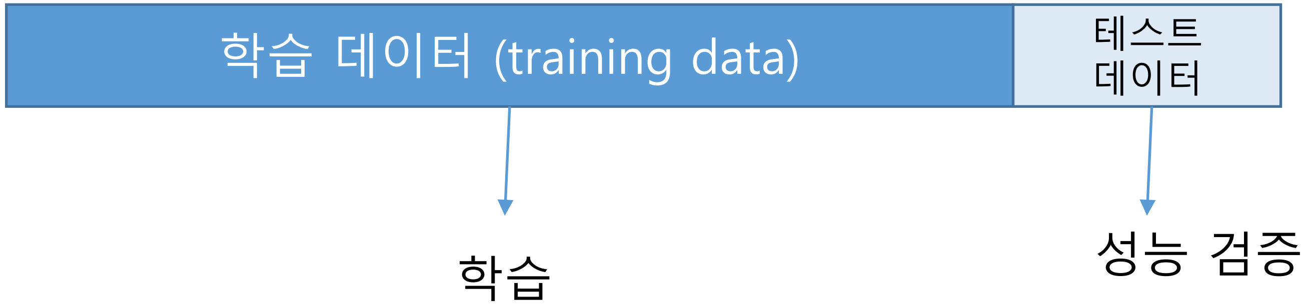

Bmi와 혈당간의 관계 예측

```
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

print('diabetes_X',diabetes_X.shape )


# 하나의 특징(BMI)만 추려내서 2차원 배열로 만든다. BMI 특징의 인덱스가 2이다.
diabetes_X_new0 = diabetes_X[:, 2]
print('diabetes_X_new0',diabetes_X_new0.shape )
diabetes_X_new = diabetes_X_new0[:, np.newaxis]
print('diabetes_X_new',diabetes_X_new.shape )
```

```
diabetes_X (442, 10)
diabetes_X_new0 (442,)
diabetes_X_new (442, 1)
```

# 선형 회귀 예제 실습






442개

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diabetes_X_new, diabetes_y, test_size=0.1, random_state=0)
print('X_train',X_train.shape )
print('X_test',X_test.shape )
print('Y_train',y_train.shape )
print('Y_test',y_test.shape )
```

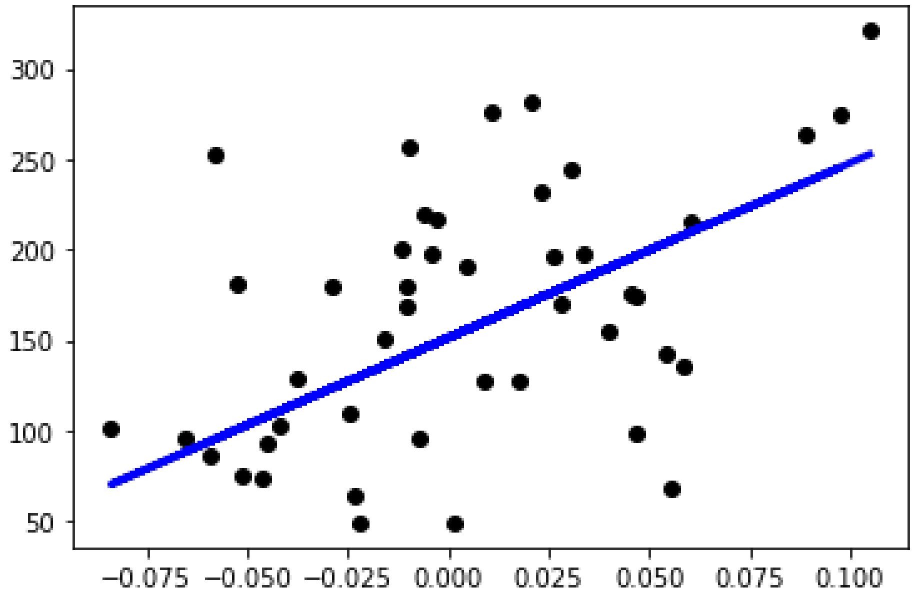
```
X_train (397, 1)
X_test (45, 1)
Y_train (397,)
Y_test (45,)
```

# 선형 회귀 예제 실습 – 당뇨병 예제



```
import matplotlib.pylab as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model

# 당뇨병 데이터 세트를 적재한다.
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

print('diabetes_X',diabetes_X.shape )
# 하나의 특징(BMI)만 추려내서 2차원 배열로 만든다. BMI 특징의 인덱스가 2이다.
diabetes_X_new0 = diabetes_X[:,  2]
print('diabetes_X_new0',diabetes_X_new0.shape )
diabetes_X_new = diabetes_X[:, np.newaxis, 2]
print('diabetes_X_new',diabetes_X_new.shape )

# 학습 데이터와 테스트 데이터를 분리한다.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diabetes_X_new, diabetes_y, test_size=0.1,
random_state=0)
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)

regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# 테스트 데이터로 예측해보자.
y_pred = regr.predict(X_test)
print(regr.predict([[0.01]])) # bmi가 0.01일때 혈당 예측값

# 실제 데이터와 예측 데이터를 비교해보자.
# plt.plot(y_test, y_pred, '.')

plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.show()
```

자료실의 diabetes_exe.ipynb
Bmi가 0.025일때의 혈당의 예측 값을 채팅으로 보내주세요



# 선형 회귀 예제 실습

다음은 집의 면적 당 가격을 정리한 표이다.

| | | | | | |
|---|---|---|---|---|---|
| 면적 | 5 | 7 | 12 | 13 | 19 |
| 가격 | 12 | 19 | 28 | 37 | 48 |

자료실의 regression1.ipnb 프로그램을 이용하여 (학습률은 0.001로 변경하시오)

1) 선형회귀로 분석 하여 직선의 방정식 y=wx+b를 구하여라
2) 면적이 10일때의 가격을 예측해보자.

print( wx +b)

(wx10 + b)

w,b, 면적이 10일때의 가격의 예측 값을 채팅으로 보내주세요.

# 선형 회귀 예제 실습

다음은 CPU 속도와 프로그래밍 수행 시간을 정리한 표이다.

| | | | | | |
|---|---|---|---|---|---|
| CPU | 30 | 50 | 80 | 90 | 120 |
| 수행 시간 | 70 | 140 | 145 | 170 | 260 |

자료실의 regression2.ipnb 프로그램을 이용하여
1) 선형회귀로 분석 하여 직선의 방정식 y=wx+b를 구하여라
1) CPU 속도가 100일때의 수행 시간을 예측 하시오.

w,b, CPU 속도가 100일때의 수행시간 예측 값을 예측 값을 채팅으로 보내주세요.

# 오차 계산하기 (1)

| | | | | |
|---|---|---|---|---|
| 공부한 시간(x) | 2 | 4 | 6 | 8 |
| 성적(실제 값, y) | 81 | 93 | 91 | 97 |
| 예측 값 | 82 | 88 | 94 | 100 |
| 오차 | 1 | −5 | 3 | 3 |

- 이렇게 해서 구한 오차를 모두 더하면 1 + (-5) + 3 + 3 = 2가 됨
- 이 값은 오차가 실제로 얼마나 큰지를 가늠하기에는 적합하지 않음
- 오차에 양수와 음수가 섞여 있어서 오차를 단순히 더해 버리면 합이 0이 될 수도 있기 때문임
- 부호를 없애야 정확한 오차를 구할 수 있음

$$\text{오차의 합} = \sum_{i}^{n} (\hat{y}_i - y_i)^2$$

- 이 식으로 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44임

오차=loss=손실 함수

# 오차 계산하기 (2)

- 평균 제곱 오차(Mean Squared Error, MSE) :

  오차의 합에 이어 각 x 값의 평균 오차를 이용함

  위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있음

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum(\hat{y}_i - y_i)^2$$

- 선형 회귀란 :

  임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고, 이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업임

# 오차 수정하기 : 경사 하강법

- Y=ax
- 오차가 가장 작은 지점은?

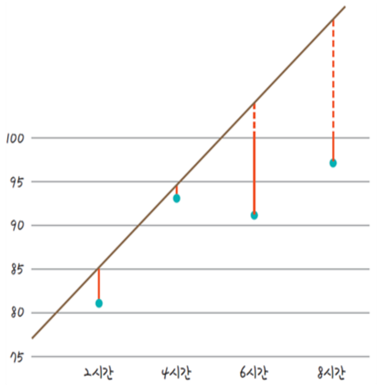


그림 3-7 기울기를 너무 크게 잡았을 때의 오차

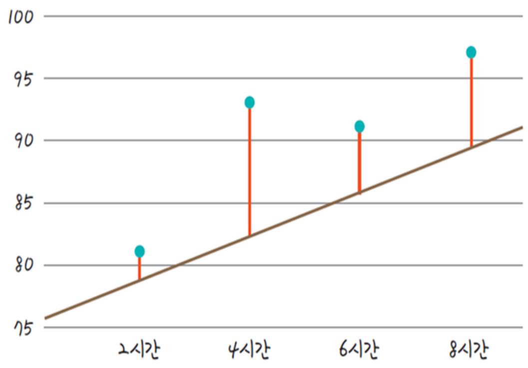
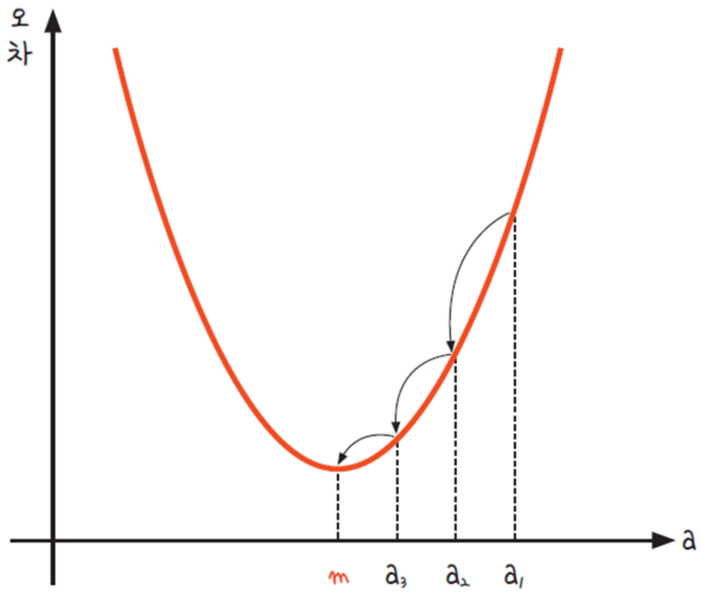


그림 3-8 기울기를 너무 작게 잡았을 때의 오차



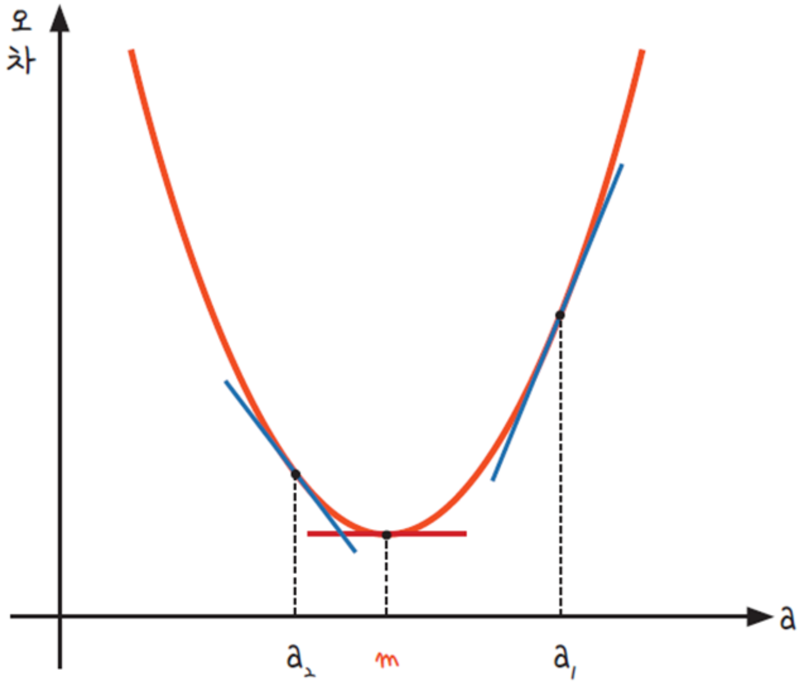

- 컴퓨터를 이용해 $m$의 값을 구하려면 임의의 한 점($a_1$)을 찍고 이 점을 $m$ 에 가까운 쪽으로 점점 이동($a_1 \rightarrow a_2 \rightarrow a_3$)시키는 과정이 필요함
- 경사 하강법(gradient descent) :

  그래프에서 오차를 비교하여 가장 작은 방향으로 이동시키는 방법이 있는데 바로 미분 기울기를 이용

# 경사 하강법

- $y = x^2$ 그래프에서 $x$에 다음과 같이 $a_1$, $a_2$그리고 $m$을 대입하여 그 자리에서 미분하면 그림 4-2처럼 각 점에서의 순간 기울기가 그려짐

- 여기서 눈여겨 봐야 할 것은 우리가 찾는 최솟값 m에서의 순간 기울기임
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x축과 평행한 선이 됨
- 즉, 기울기가 0임
- 우리가 할 일은 '미분 값이 0인 지점'을 찾는 것이 됨

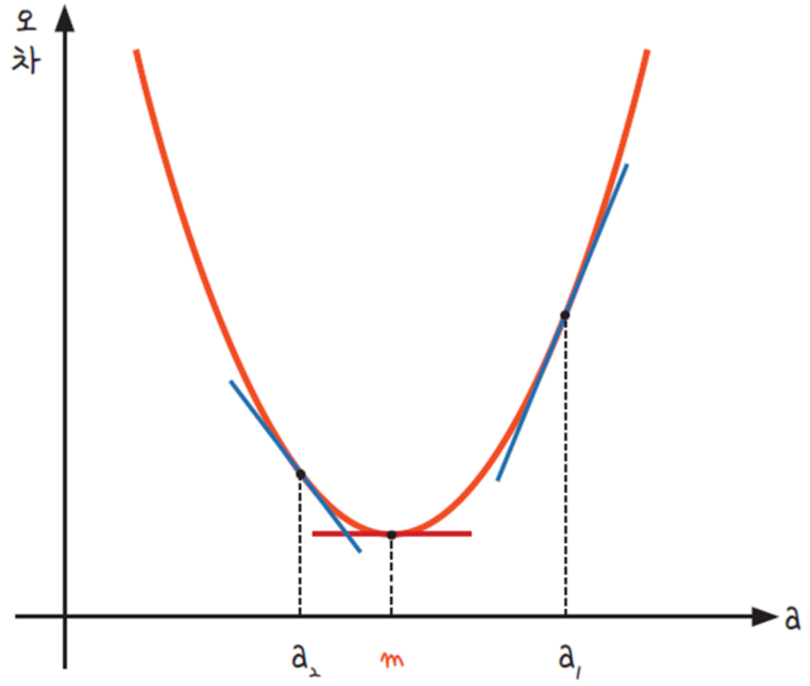


# 경사 하강법

1 | $a_1$ 에서 미분을 구함

2 | 구해진 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킨 $a_2$ 에서 미분을 구함(그림 4-3 참조).

3 | 위에서 구한 미분 값이 0이 아니면 위 과정을 반복함



# 경사 하강법

- 최솟값을 구하기 위해서는 이차 함수에서 미분을 해야 함
- 그 이차 함수는 평균 제곱 오차를 통해 나온다는 것임
- 평균 제곱 오차의 식을 다시 옮겨 보면 다음과 같음

$$\frac{1}{n}\sum\left(\hat{y}_i - y_i\right)^2$$

- 여기서 $\hat{y}_i$은 $x_i$ 를 집어 넣었을 때의 값이므로 $y_i = ax_i + b$ 를 대입하면

  다음과 같이 바뀜

$$\frac{1}{n}\sum\left(\left(ax_i + b\right) - y_i\right)^2$$

# 경사 하강법

TIP

a로 편미분한 결과 유도 과정

$$
\begin{aligned}
\frac{a}{\partial a} MSE(a,b) &= \frac{1}{n}\sum[(ax_i+b-y_i)^2]' \\
&= \frac{2}{n}(ax_i+b-y_i)[(ax_i+b-y_i)]' \\
&= \boxed{\frac{2}{n}\sum(ax_i+b-y_i)x_i}
\end{aligned}
$$

b로 편미분한 결과 유도 과정

$$
\begin{aligned}
\frac{a}{\partial a} MSE(a,b) &= \frac{1}{n}\sum[(ax_i+b-y_i)^2]' \\
&= \frac{2}{n}(ax_i+b-y_i)[(ax_i+b-y_i)]' \\
&\boxed{= \frac{2}{n}\sum(ax_i+b-y_i)}
\end{aligned}
$$

# 경사 하강법

```
y_pred = a * x_data + b   # 오차 함수인 y = ax + b를 정의한 부분
error = y_data - y_pred   # 실제값 - 예측값, 즉 오차를 구하는 식


# 평균 제곱 오차를 a로 미분한 결과
a_diff = -(2 / len(x_data)) * sum(x_data * (error))
# 평균 제곱 오차를 b로 미분한 결과
b_diff = -(2 / len(x_data)) * sum(y_data - y_pred)
```

```
a = a - lr * a_diff   # 미분 결과에 학습률을 곱한 후 기존의 a값을 업데이트
b = b - lr * b_diff   # 미분 결과에 학습률을 곱한 후 기존의 b값을 업데이트
```

# 선형 회귀 예제 실습



| x | Y |
|---|---|
| 0 | 3 |
| 1 | 3.5 |
| 2 | 5.5 |
| | |

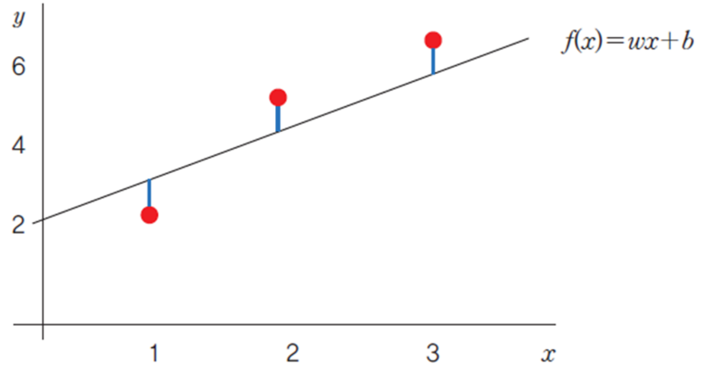



그림 4-5 데이터와 직선 간의 거리

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

X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

w = 0       # 기울기
b = 0       # 절편

lrate = 0.01  # 학습률
epochs = 1000  # 반복 횟수

n = float(len(X)) # 입력 데이터의 개수

# 경사 하강법
for i in range(epochs):
    y_pred = w*X + b                          # 선형 회귀 예측값
    dw = (2/n) * sum(X * (y_pred-y))          # 넘파이 배열간의 산술 계산은 요소별로 적용
    db = (2/n) * sum(y_pred-y)                # sum()은 모든 요소들의 합을 계산하는 내장 함수
    w = w - lrate * dw                        # 기울기 수정
    b = b - lrate * db                        # 절편 수정

# 기울기와 절편을 출력한다.
print (w, b)

# 예측값을 만든다.
y_pred = w*X + b

# 입력 데이터를 그래프 상에 찍는다.
plt.scatter(X, y)

# 예측값은 선그래프로 그린다.
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
plt.show()
```

참고자료 : 딥러닝 express

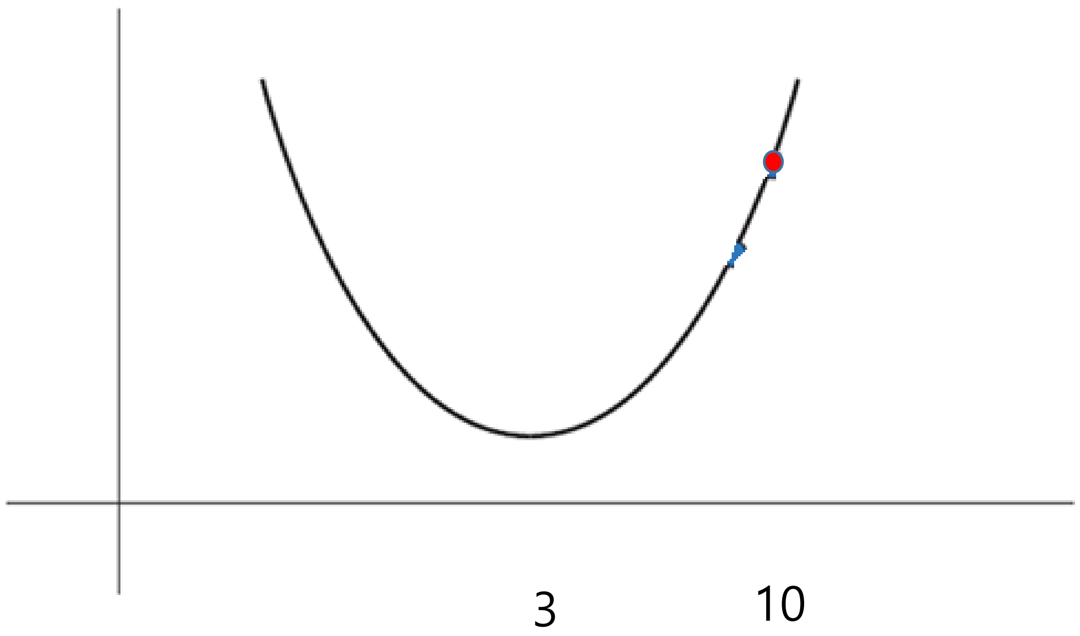
# 선형 회귀 예제 실습



```
iteration 0: loss 17.17 w 0.10 b 0.08
iteration 50: loss 0.61 w 1.73 b 1.71
iteration 100: loss 0.33 w 1.73 b 2.05
iteration 150: loss 0.24 w 1.63 b 2.23
iteration 200: loss 0.19 w 1.53 b 2.36
iteration 250: loss 0.16 w 1.47 b 2.45
iteration 300: loss 0.15 w 1.41 b 2.52
iteration 350: loss 0.14 w 1.37 b 2.58
iteration 400: loss 0.13 w 1.34 b 2.62
iteration 450: loss 0.13 w 1.32 b 2.65
########## final w,b 1.3033228991130752
2.6760184293088694


1.25 2.74
```

# 경사 하강법 실습

- **손실 함수** $y = (x-3)^2 + 10$
- **그래디언트: y' = 2x-6**




참고자료 : 딥러닝 express

# 경사 하강법 실습

- 손실 함수 $y = (x - 3)^2 + 10$
- 그래디언트: y' = 2x-6
- 학습률 : 0.2

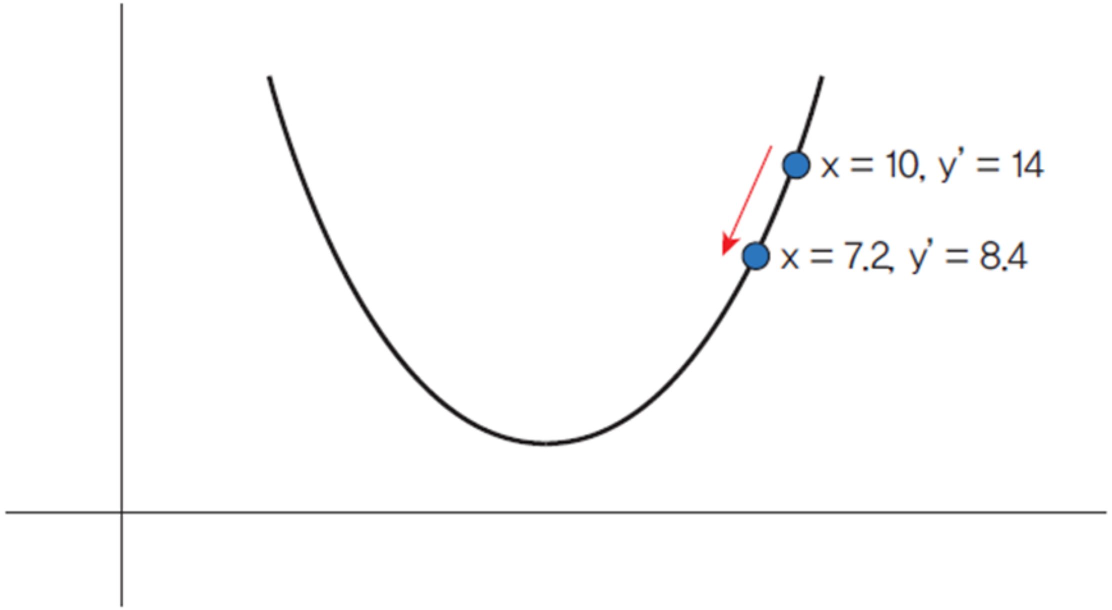


그림 6–12 그래디언트의 계산

- X=10, y'=14, 0.2*14=2.8,

Gradient의 반대 방향 => -2.8

10-2.8=7.2

- X=7.2, Y =8.4, 0.2*8.4=

# 경사 하강법 실습

grad2_exe.ipynb

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

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

list1 = []
list2 = []

# 그래디언트 강하법
for i in range(max_iterations):
    x = x - learning_rate * gradient(x)
    list1.append(x)
    list2.append(loss_func(x))
    print("X=", x, "loss", loss_func(x))

print("최소값 = ", x)

x1 = np.linspace(0.0, 10.0)
y1 = loss_func(x1)
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x1,y1)  # Plot some data on the axes.
ax.plot(list1,list2, '*')  # Plot some data on the axes.
```

3번째 x값과 loss값을 채팅으로 보내주세요

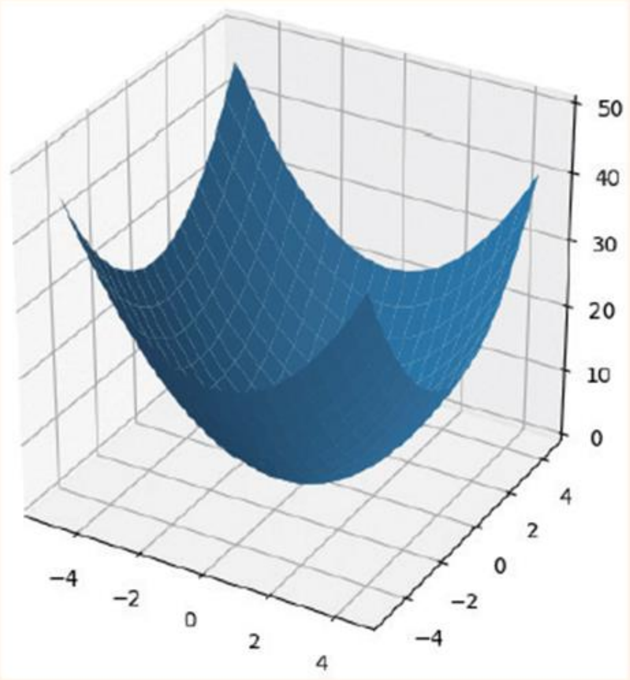
참고자료 : 딥러닝 express

# 경사 하강법 실습

f(x,y)=$x^2 + y^2$

grad3.ipynb 실습

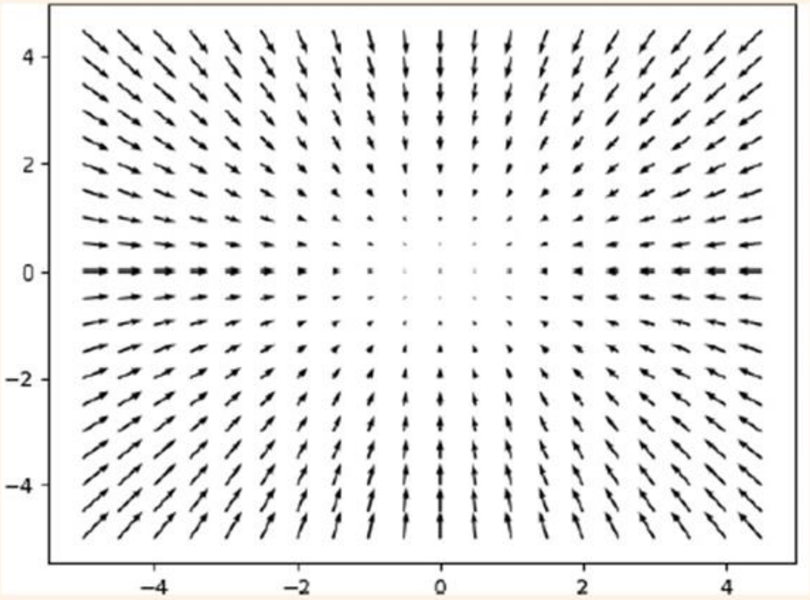
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-5, 5, 0.5)
y = np.arange(-5, 5, 0.5)
X, Y = np.meshgrid(x, y) # 참고 박스
Z = X**2 +Y**2          # 넘파이 연산

fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')

# 3차원 그래프를 그린다.
ax.plot_surface(X, Y, Z)
plt.show()
```




참고자료 :  딥러닝 express

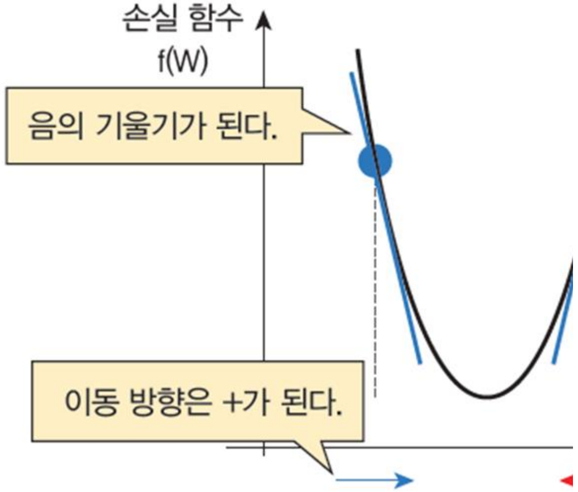
# 경사 하강법 실습

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

x = np.arange(-5,5,0.5)
y = np.arange(-5,5,0.5)
X, Y = np.meshgrid(x,y)
U = -2*X
V = -2*Y

plt.figure()
Q = plt.quiver(X, Y, U, V, units='width')
plt.show()
```




참고자료 : 딥러닝 express

# 경서 하강법 실습



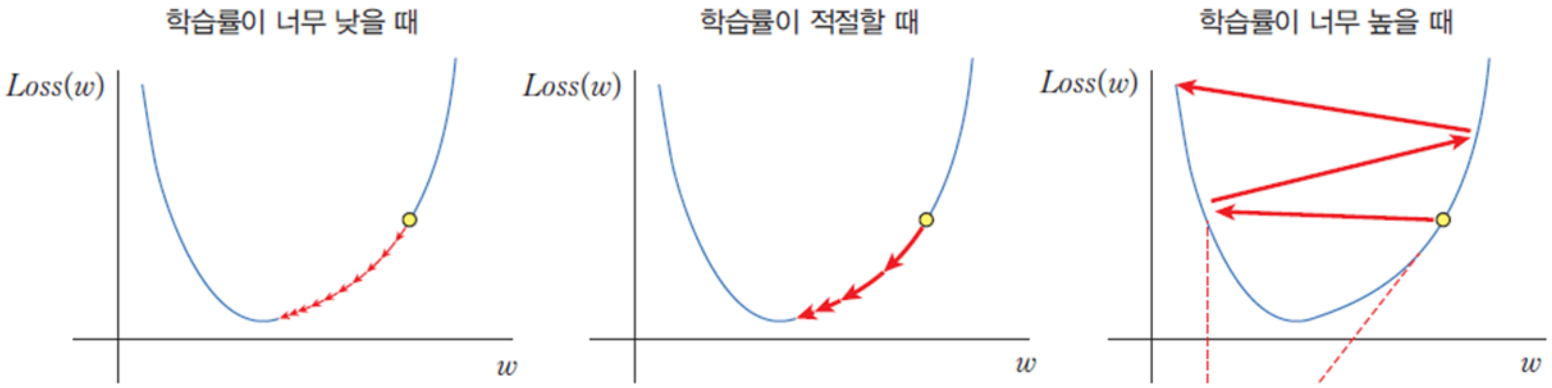




그림 4-9 학습률


이것은 마치 산에서 내려오는 것과 유사합니다. 현재 위치에서 산의 기울기를 계산하여서 기울기의 반대 방향으로 이동하면 산에서 내려오게 됩니다.

참고자료 : 딥러닝 express

# 경사하강법

- 그림 4-3처럼 기울기가 0인 한 점( m )으로 수렴함

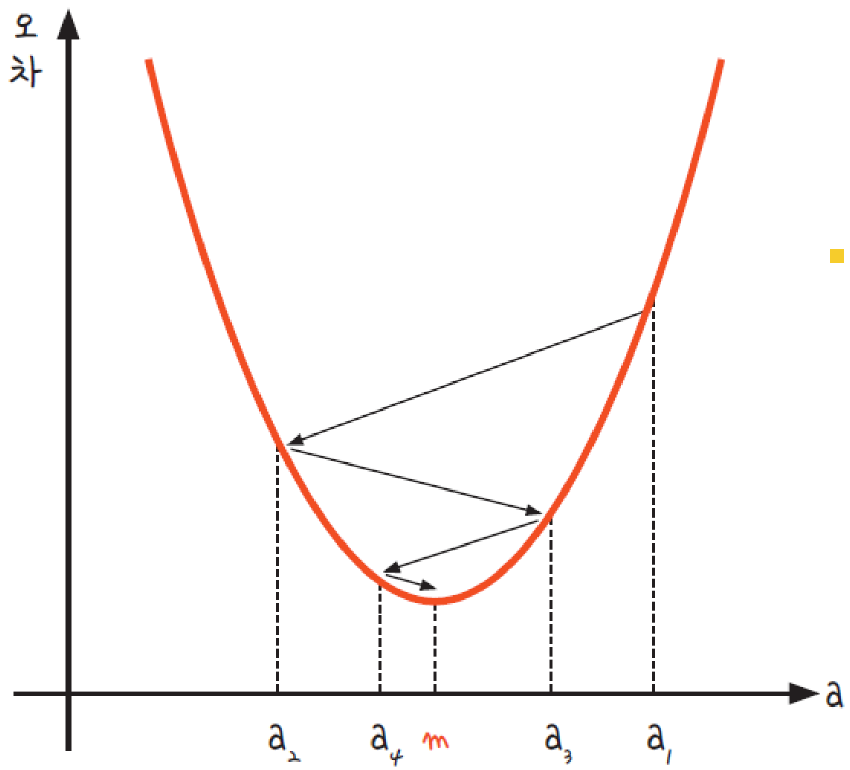



그림 4-3 최솟점 m을 찾아가는 과정

- 경사 하강법 :

  이렇게 반복적으로 기울기 a를 변화시켜서 m의 값을 찾아내는 방법을 말함

# 경서 하강법 실습

| 공부한 시간 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| 성적 | 81 | 93 | 91 | 97 |
| 예측 값 | 83.6 | 88.2 | 92.8 | 97.4 |

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

# 공부 시간 X와 성적 Y의 리스트를 만들기
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]
# 그래프로 나타내기
plt.figure(figsize=(8,5))
plt.scatter(x, y)
plt.show()

# 리스트로 되어 있는 x와 y 값을 넘파이 배열로 바꾸기(인덱스를 주어 하나씩 불러와 계산이 가능하게 하기 위함)
x_data = np.array(x)
y_data = np.array(y)

# 기울기 a와 절편 b의 값 초기화
a = 0
b = 0
```

```
# 학습률 정하기
lr = 0.05
```

```
# 몇 번 반복될지 설정(0부터 세므로 원하는 반복 횟수에 +1)
epochs = 2001

# 경사 하강법 시작
for i in range(epochs):          # 에포크 수만큼 반복
    y_pred = a * x_data + b      # y를 구하는 식 세우기
    error = y_data - y_pred      # 오차를 구하는 식
    # 오차 함수를 a로 미분한 값
    a_diff = -(1/len(x_data)) * sum(x_data * (error))
    # 오차 함수를 b로 미분한 값
    b_diff = -(1/len(x_data)) * sum(y_data - y_pred)
```

# 경서 하강법 실습

```
    a = a - lr * a_diff   # 학습률을 곱해 기존의 a값 업데이트
    b = b - lr * b_diff   # 학습률을 곱해 기존의 b값 업데이트

    if i % 100 == 0:        # 100번 반복될 때마다 현재의 a값, b값 출력
        print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))

# 앞서 구한 기울기와 절편을 이용해 그래프를 다시 그리기
y_pred = a * x_data + b
plt.scatter(x, y)
plt.plot([min(x_data), max(x_data)], [min(y_pred), max(y_pred)])
plt.show()
```

실행 결과

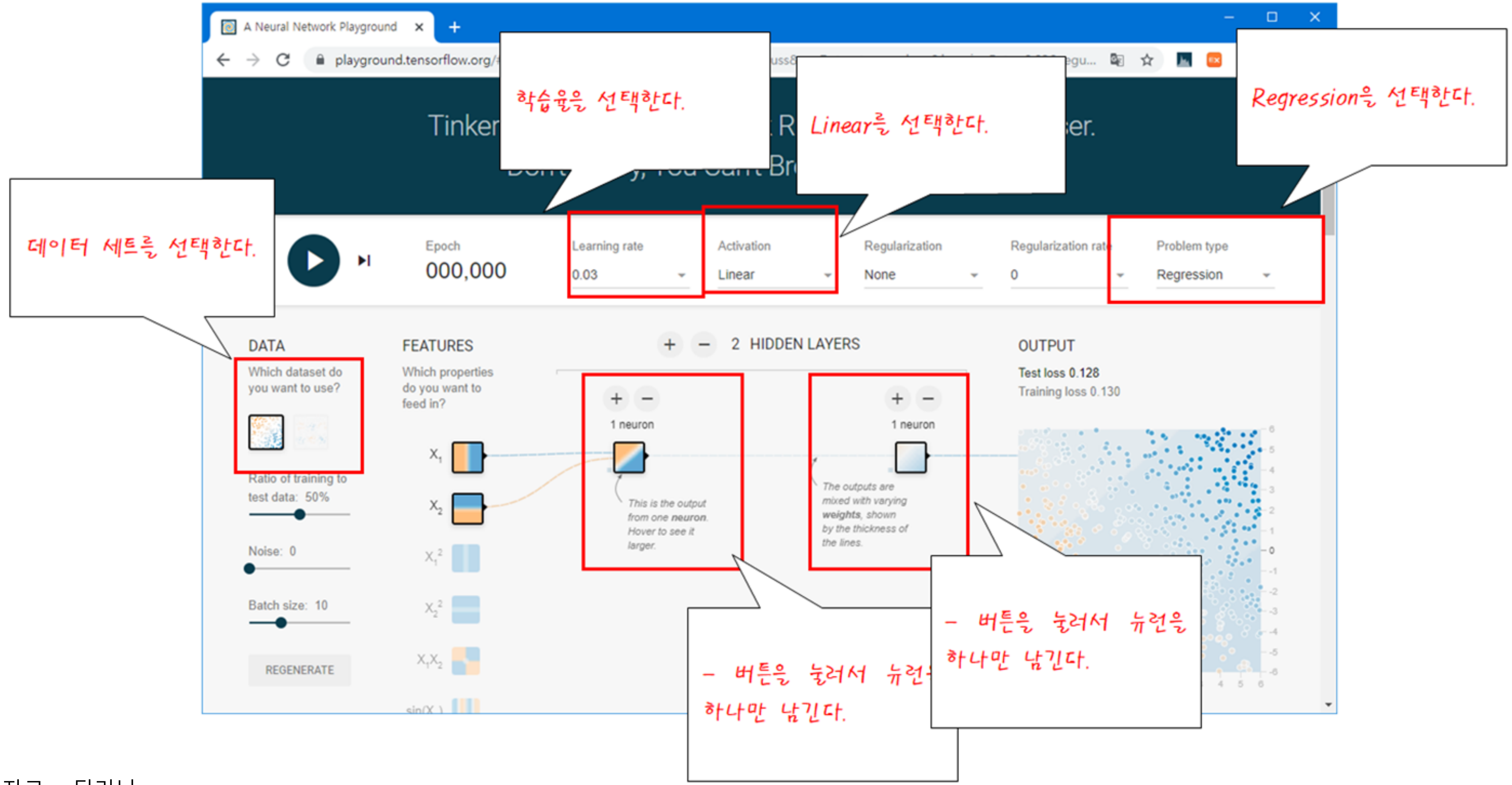
```
epoch=0, 기울기=23.2000, 절편=4.5250
epoch=100, 기울기=7.9316, 절편=45.3932
epoch=200, 기울기=4.7953, 절편=64.109
epoch=300, 기울기=3.4056, 절편=72.4022
(중략)
epoch=1800, 기울기=2.3000, 절편=79.0000
epoch=1900, 기울기=2.3000, 절편=79.0000
epoch=2000, 기울기=2.3000, 절편=79.0000
```

# 경사 하강법 실습

- colab_03_Linear_Regression.ipynb 실행
- 학습률 lr=0.0003 일때의 첫번째와 두번째 loss => 채팅으로 보내기
- 학습률 lr=0.0009 일때의 첫번째와 두번째 loss => 채팅으로 보내기

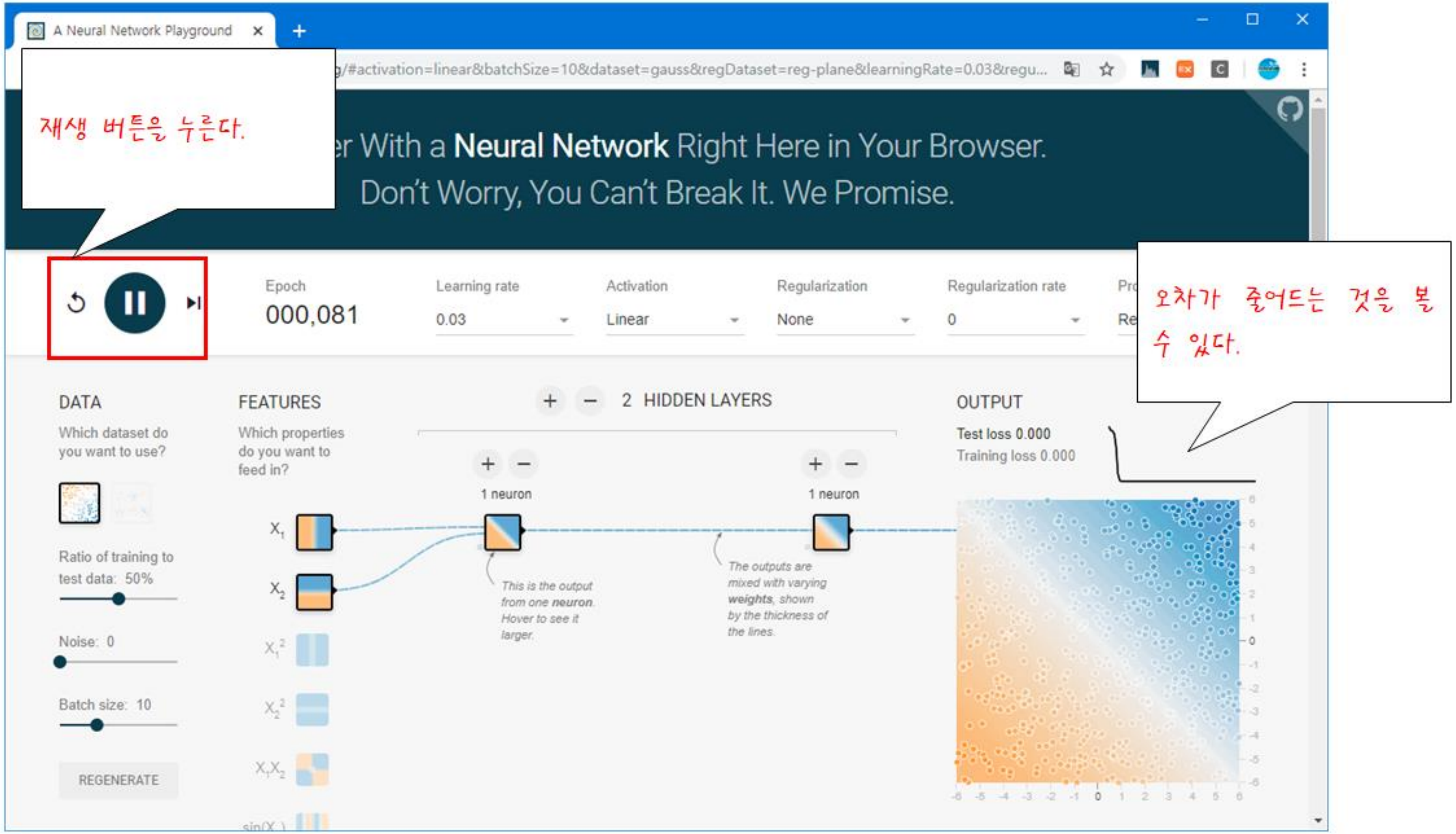
# 학습률 실습

- 구글의 텐서 플로우 플레이그라운드는 아주 유용한 사이트 (https://playground.tensorflow.org)이다.




참고자료 : 딥러닝 express

# 학습률 실습




참고자료 : 딥러닝 express

# 수업 내용 요약

- **손실 함수 (오차=loss)는 경사 하강법으로 감소 시킴**
- **경사 하강법**
  - 손실 함수(오차)를 미분한 값(기울기)의 반대 방향으로 진행 하여 최소값을 찾아감
  - 학습률에 비례 하여 진행
  - 학습률에 따라 최소값에 도달하는 시간과 정확도가 정해짐

matplotlib → 그래프 그리기

# 수고 하셨습니다

jhmin@inhatc.ac.kr