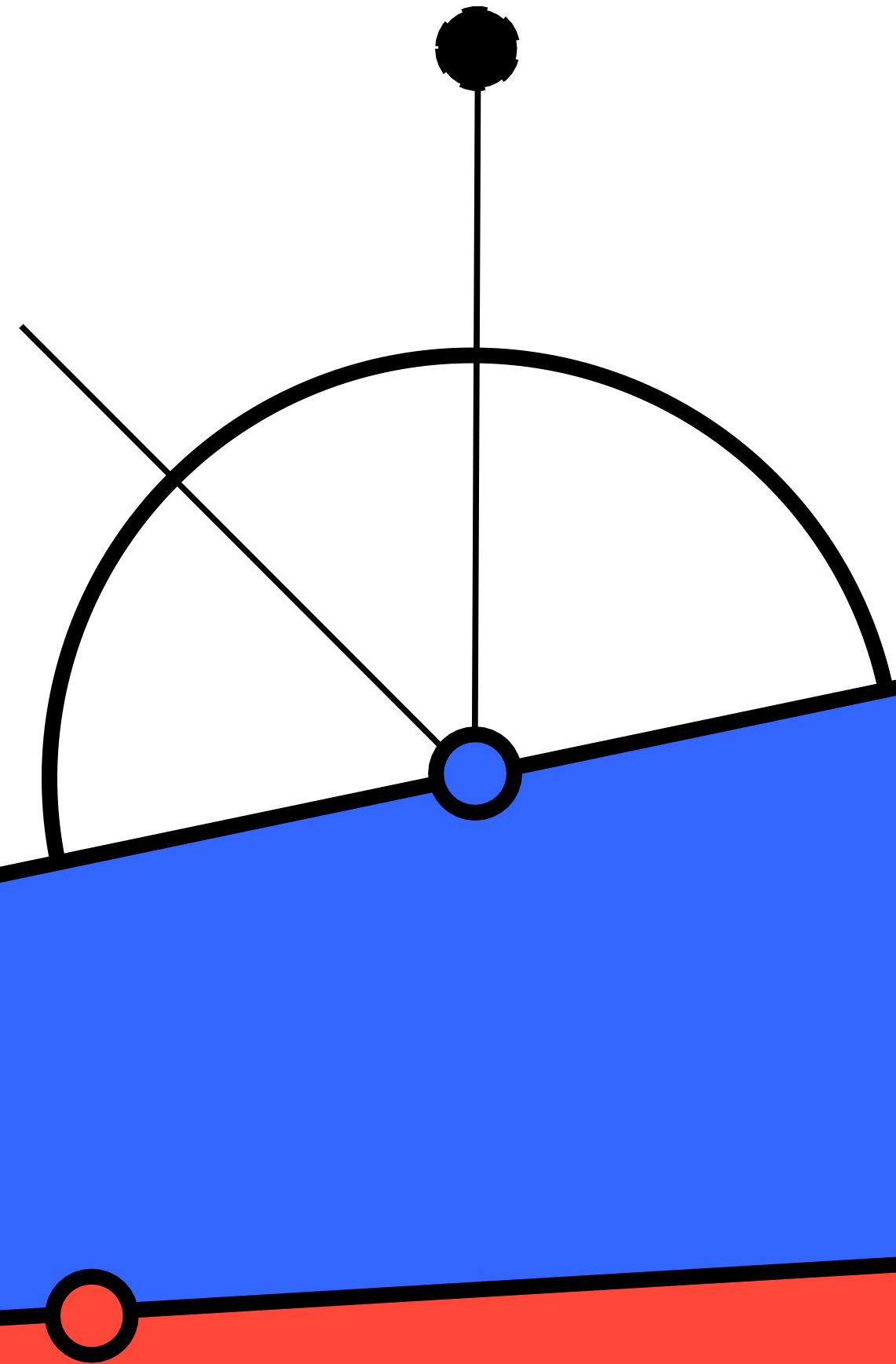


딥 러닝을 이용한 자연어 처리 입문

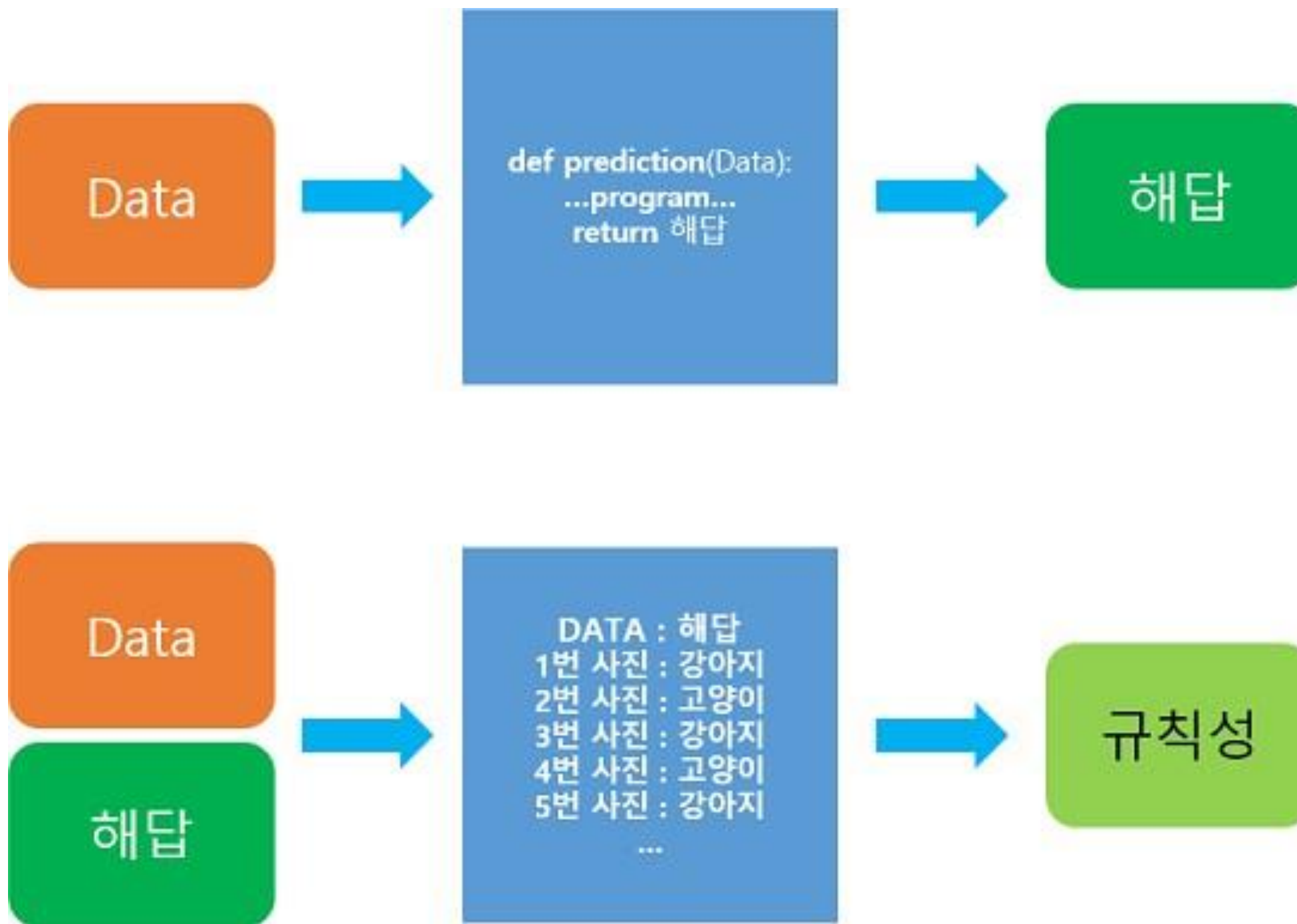
06. 머신 러닝 (Machine Learning) 개요



목차

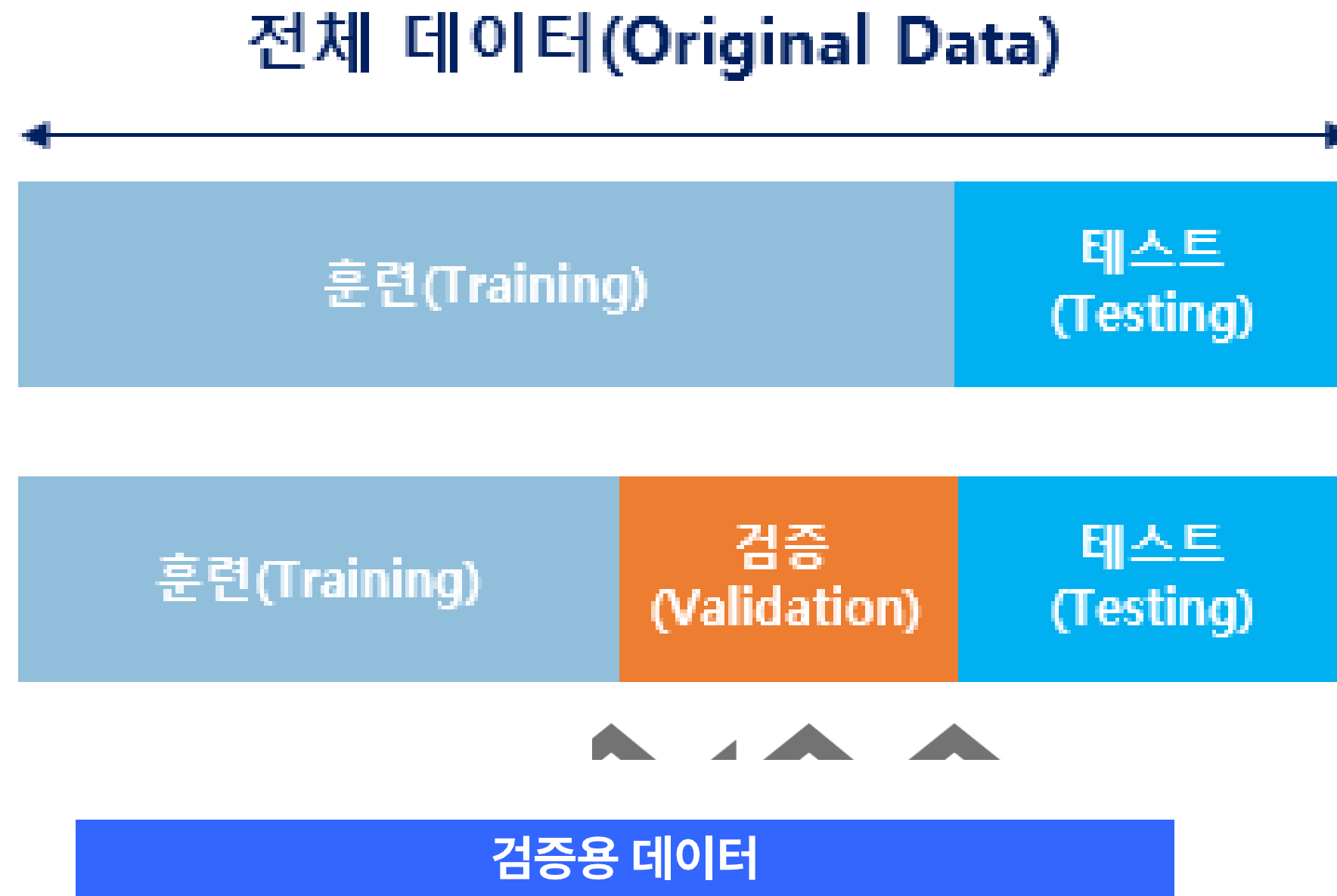
- 06-01 머신 러닝이란**
- 06-02 머신 러닝 훑어보기**
- 06-03 선형 회귀(Linear Regression)**
- 06-04 자동 미분과 선형 회귀 실습**
- 06-05 로지스틱 회귀(Logistic Regression)**
- 06-06 로지스틱 회귀 실습**
- 06-07 다중 입력에 대한 실습**
- 06-08 벡터와 행렬 연산**
- 06-09 소프트맥스 회귀(Softmax Regression)**
- 06-10 소프트맥스 회귀 실습**

06-01 머신 러닝이란?



06-02 머신 러닝 훑어보기

1. 머신 러닝 모델의 평가



모델이 훈련 데이터에 과적합(overfitting) 이 되고 있는지 판단

06-02 머신 러닝 훑어보기

2. 분류(Classification)와 회귀(Regression)

1) 이진 분류 문제(Binary Classification)

주어진 입력에 대해서 **두 개의 선택지** 중 하나의 답을 선택해야 하는 경우

ex. 종합 시험 성적표를 보고 최종적으로 합격, 불합격인지

메일을 보고나서 정상 메일, 스팸 메일인지

06-02 머신 러닝 훑어보기

2. 분류(Classification)와 회귀(Regression)

2) 다중 클래스 분류(Multi-class Classification)

주어진 입력에 대해서 **세 개 이상의 선택지** 중에서 답을 선택해야 하는 경우

ex. 과학, 영어, IT, 학습지, 만화라는 레이블이 붙어있는 5개의 책장

새 책이 입고되면 다섯 개의 책장 중에서

분야에 맞는 적절한 책장에 책을 넣어야

06-02 머신 러닝 훑어보기

2. 분류(Classification)와 회귀(Regression)

3) 회귀 문제(Regression)

연속적인 값의 범위 내에서 예측값이 나오는 경우
역과의 거리, 인구 밀도, 방의 개수 등을 입력하면 부동산 가격을 예측
부동산 가격을 7억 8,456만 3,450원으로 예측,
or 8억 1257만 300원으로 예측

06-02 머신 러닝 훑어보기

3. 지도 학습과 비지도 학습

1) 지도 학습(Supervised Learning)

레이블(Label)이라는 정답과 함께 학습하는 것

2) 비지도 학습(Unsupervised Learning)

데이터에 별도의 레이블이 없이 학습하는 것

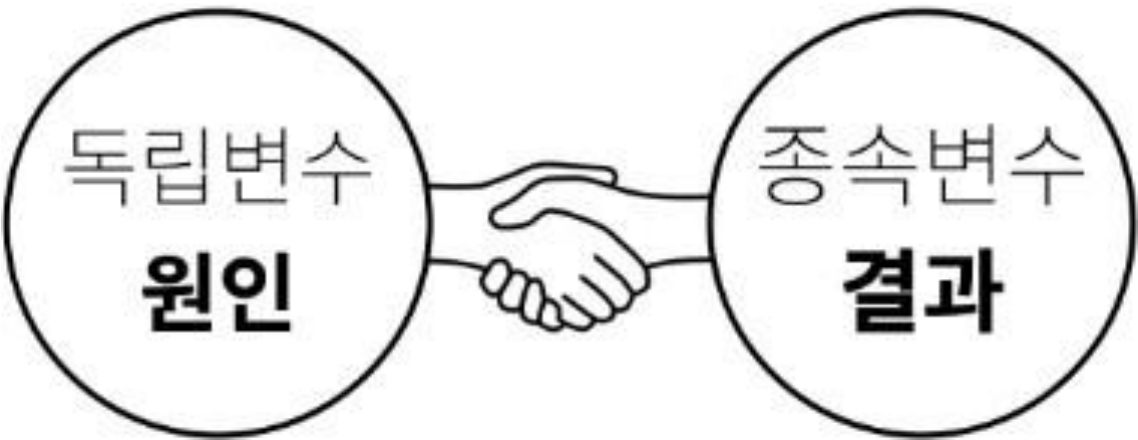
3) 자기지도 학습(Self-Supervised Learning, SSL)

레이블이 없는 데이터가 주어지면

모델이 학습을 위해서 스스로 데이터로부터 레이블을 만들어서 학습

06-02 머신 러닝 훑어보기

4. 샘플(Sample)과 특성(Feature)



Feature-1	Feature-2	Feature-3	Feature-4	...	Feature-n	
x_1	x_2	x_3	x_4	...	x_n	Sample-1
x_1	x_2	x_3	x_4	...	x_n	Sample-2
x_1	x_2	x_3	x_4	...	x_n	Sample-3
x_1	x_2	x_3	x_4	...	x_n	Sample-4
...
x_1	x_2	x_3	x_4	...	x_n	Sample-m

샘플(Sample) = 행
각각의 독립 변수 = 특성(Feature) = 열

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

$$\text{정확도}(Accuracy) = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

정확도(Accuracy)

맞춘 문제수를 전체 문제수로 나눈 값

맞춘 결과와 틀린 결과에 대한 세부적인 내용을 알려주지 못함



실제 클래스
(Actual Class)

Negative(0)

Positive(1)

예측 클래스
(Predicted Class)

Negative(0)

Positive(1)

TN
(True Negative)

FP
(False Positive)

FN
(False Negative)

TP
(True Positive)

혼동 행렬(Confusion Matrix)

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

		예측 클래스 (Predicted Class)	
		Negative(0)	Positive(1)
실제 클래스 (Actual Class)	Negative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

혼동 행렬(Confusion Matrix)

- True Positive(TP) : 실제 True인 정답을 True라고 예측 (정답)
- False Positive(FP) : 실제 False인 정답을 True라고 예측 (오답)
- False Negative(FN) : 실제 True인 정답을 False라고 예측 (오답)
- True Negative(TN) : 실제 False인 정답을 False라고 예측 (정답)

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

		예측 클래스 (Predicted Class)	
		Negative(0)	Positive(1)
실제 클래스 (Actual Class)	Negative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

혼동 행렬(Confusion Matrix)

1) 정밀도(Precision)

모델이 True라고 분류한 것 중에서 실제 True인 것의 비율

$$\text{정밀도} = \frac{TP}{TP + FP}$$

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

		예측 클래스 (Predicted Class)	
		Negative(0)	Positive(1)
실제 클래스 (Actual Class)	Negative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

혼동 행렬(Confusion Matrix)

2) 재현율(Recall)

실제 True인 것 중에서 모델이 True라고 예측한 것의 비율

$$\text{재현율} = \frac{TP}{TP + FN}$$

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

		예측 클래스 (Predicted Class)	
		Negative(0)	Positive(1)
실제 클래스 (Actual Class)	Negative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

혼동 행렬(Confusion Matrix)

3) 정확도(Accuracy)

전체 예측한 데이터 중에서 정답을 맞춘 것에 대한 비율

$$\text{정확도} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

06-02 머신 러닝 훑어보기

5. 혼동 행렬(Confusion Matrix)

3) 정확도(Accuracy)

전체 예측한 데이터 중에서 정답을 맞춘 것에 대한 비율

$$\text{정확도} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

Accuracy로 성능을 예측하는 것이 적절하지 않은 때

스팸 메일 분류기

메일 100개 중 스팸 메일은 5개

스팸 메일 분류기는 모두 정상 메일이라고 탐지

> 정확도는 95%

정작 스팸 메일은 하나도 못 찾아낸 셈

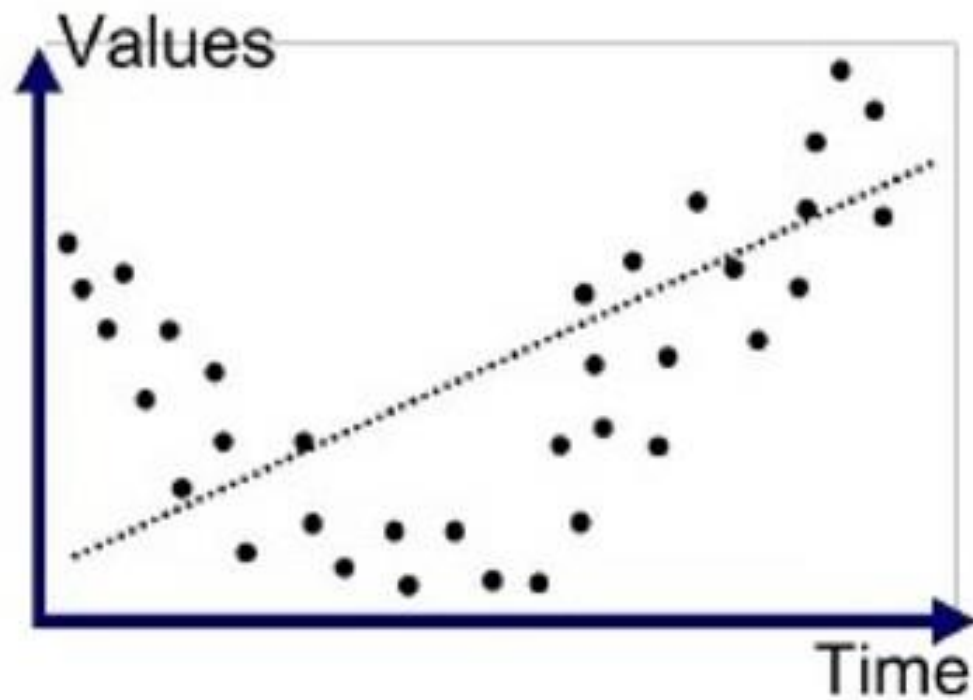
중요 데이터가 전체 데이터에서

너무 적은 비율을 차지한다면

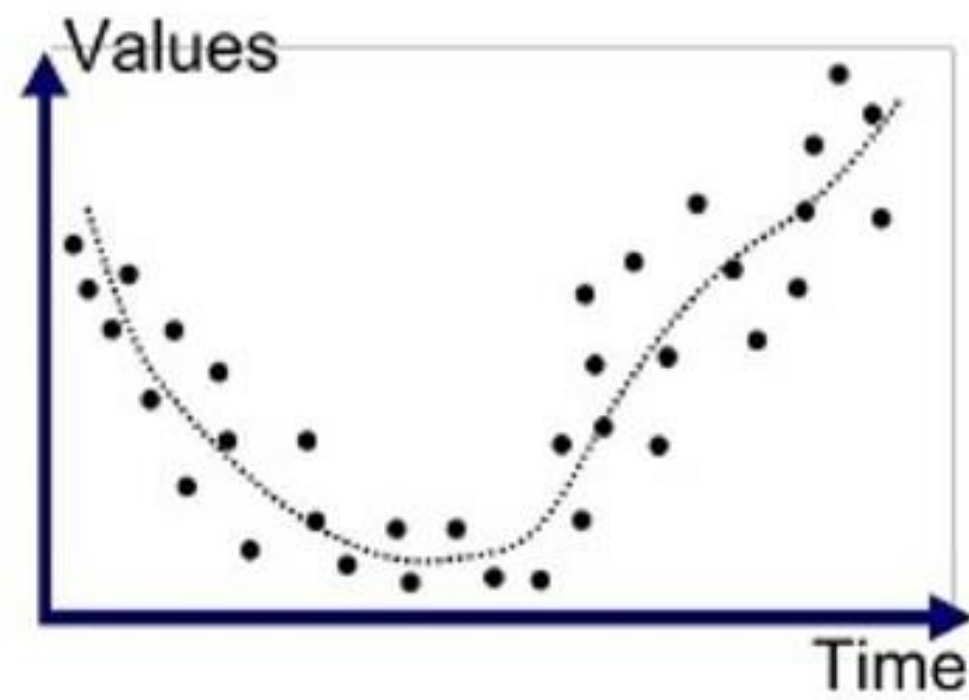
정확도는 좋은 측정 지표가 될 수 없음 !!

06-02 머신 러닝 훑어보기

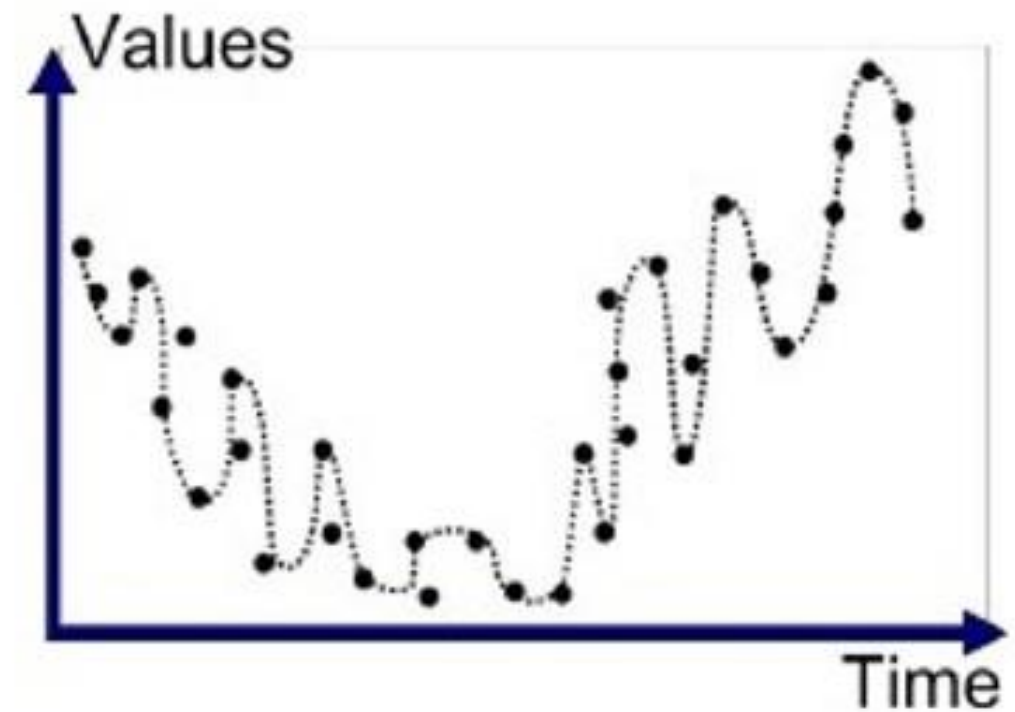
6. 과적합(Overfitting)과 과소 적합(Underfitting)



Underfitted



Good Fit/Robust



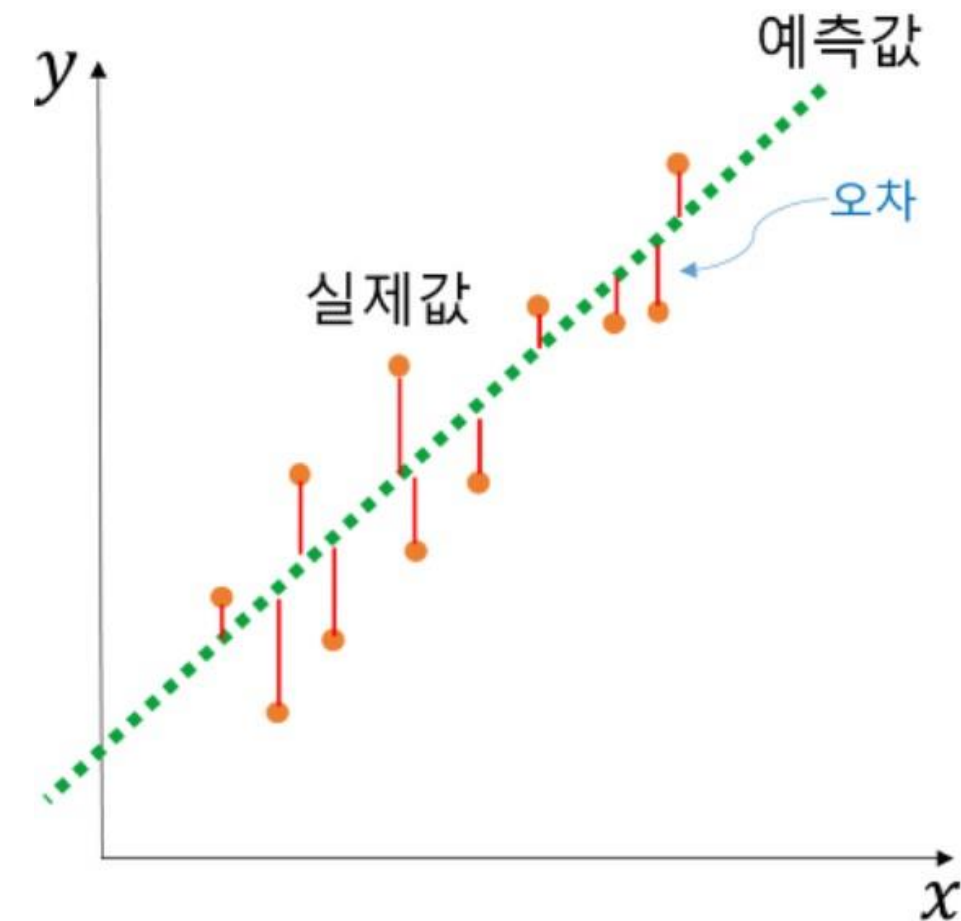
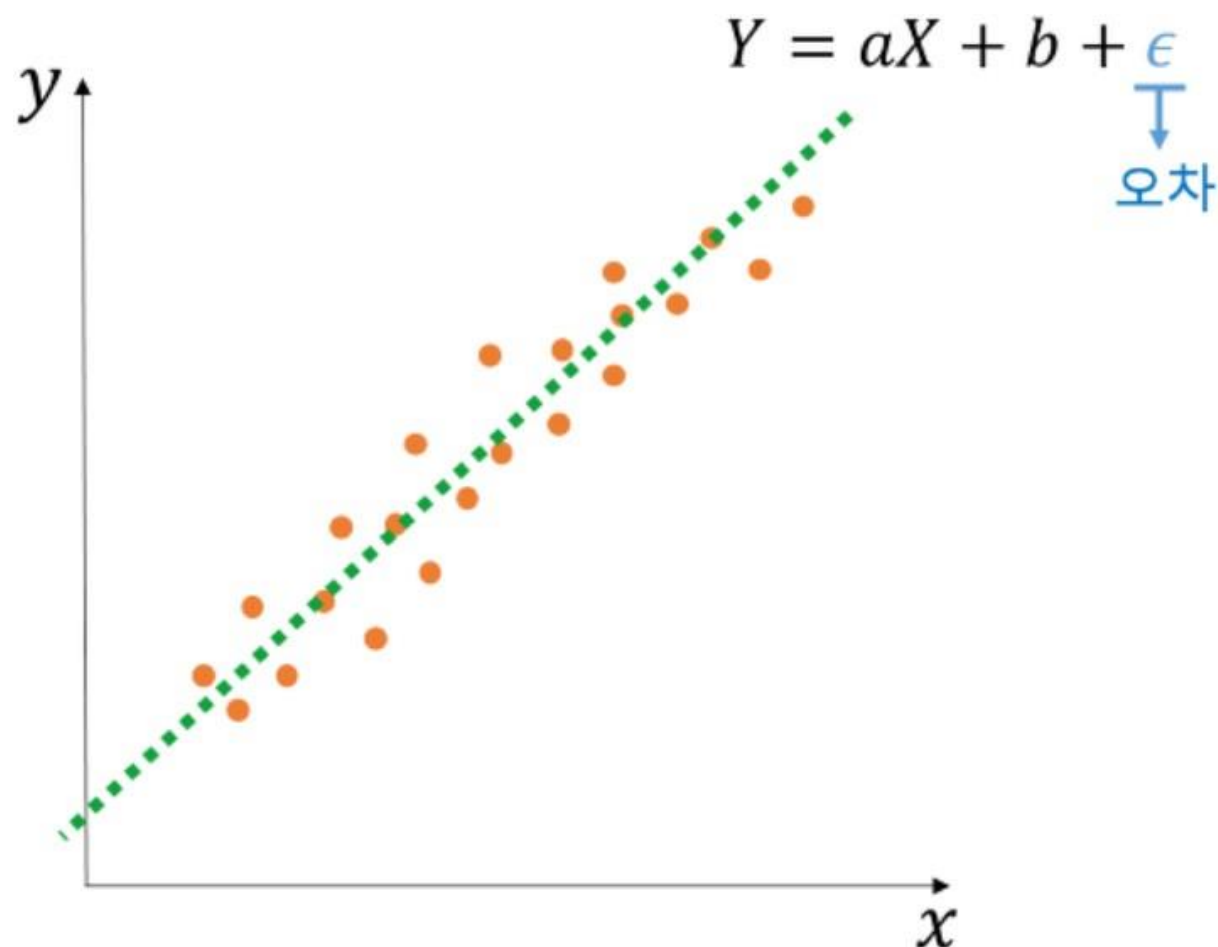
Overfitted

06-03 선형 회귀(Linear Regression)

1. 선형 회귀(Linear Regression)

주어진 데이터의 관계를 가장 잘 나타내는 선형 함수를 찾아내는 통계적인 방법
독립 변수와 종속 변수 간의 선형적인 관계를 모델링하고 예측

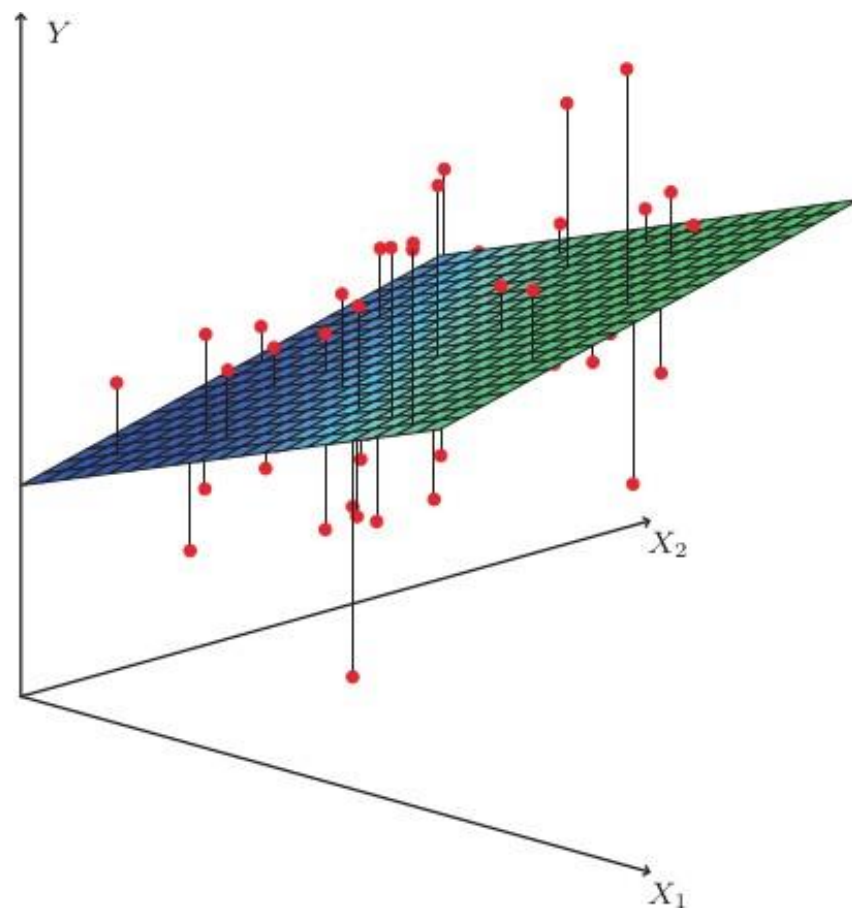
1) 단순 선형 회귀 분석



06-03 선형 회귀(Linear Regression)

1. 선형 회귀(Linear Regression)

2) 다중 선형 회귀 분석



$$y = w_1x_1 + w_2x_2 + \dots w_nx_n + b$$

ex. 집의 매매 가격

단순히 집의 평수가 크다고 결정되는 게 아니라

집의 층의 수, 방의 개수, 지하철 역과의 거리와도 영향

06-03 선형 회귀(Linear Regression)

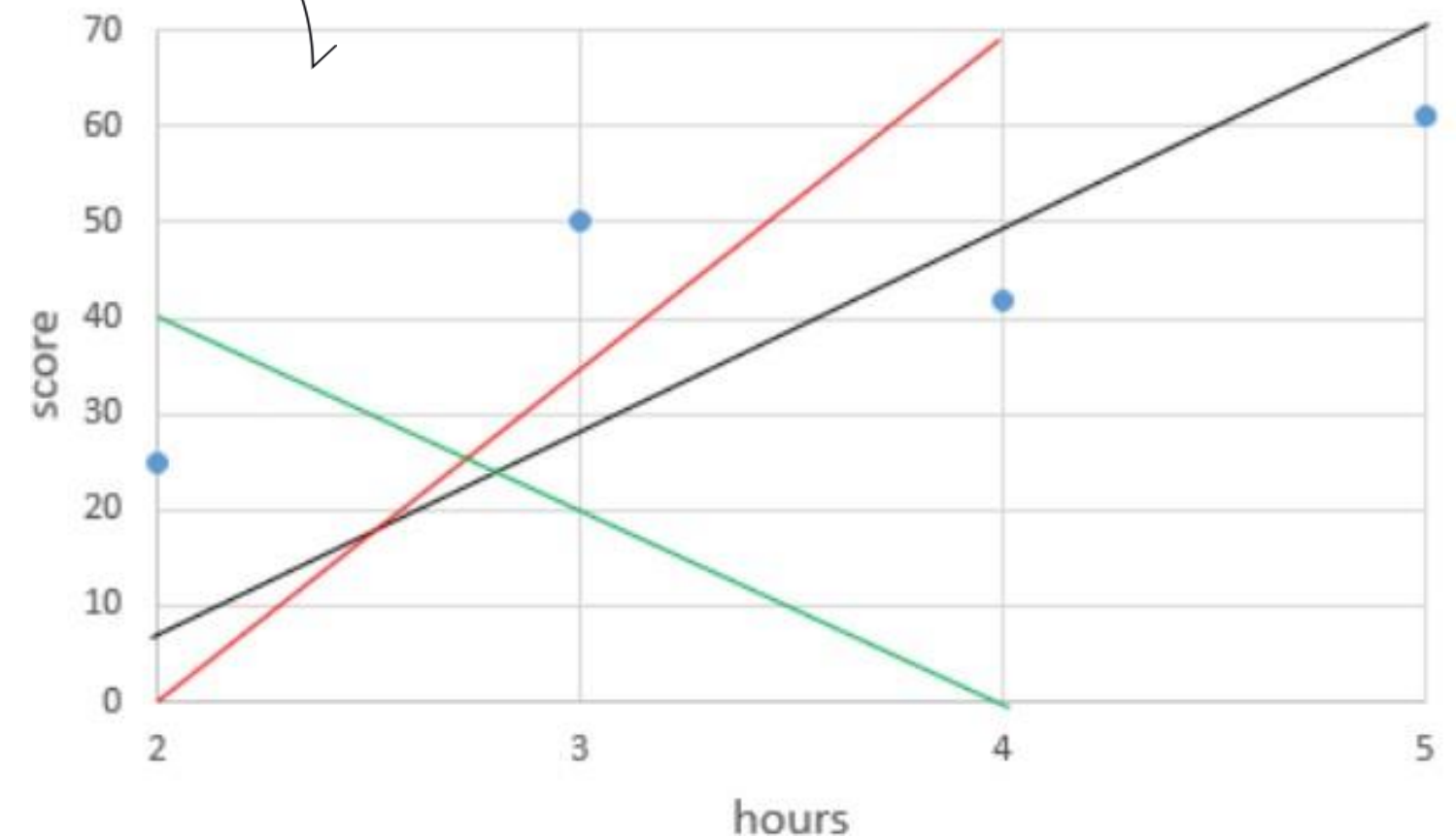
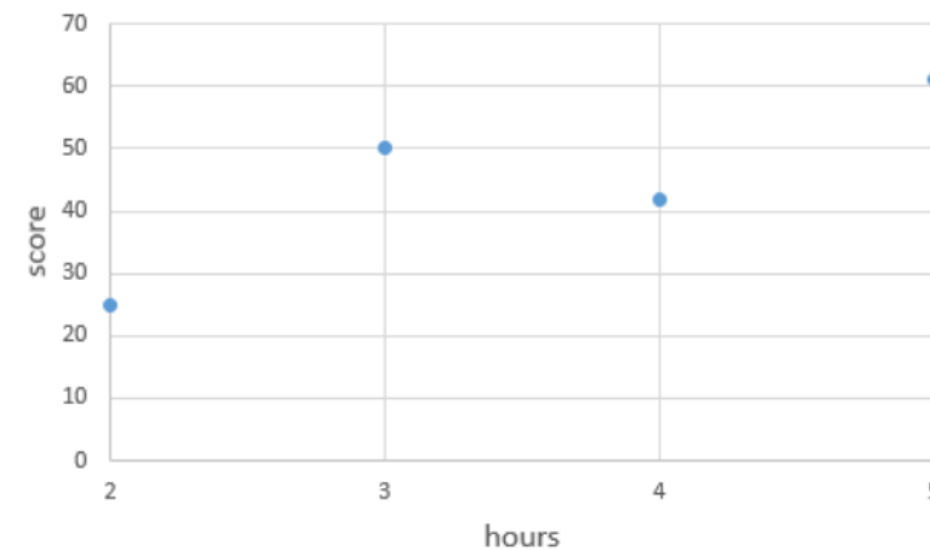
2. 가설(Hypothesis) 세우기

x, y 관계를 유추하기 위해서 수학적으로 식을 세움
> 이러한 식을 가설(Hypothesis)

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

hours(x)	score(y)
2	25
3	50
4	42
5	61

DATA



06-03 선형 회귀(Linear Regression)

3. 비용 함수(Cost function) : 평균 제곱 오차(MSE)

목적 함수(Objective function)

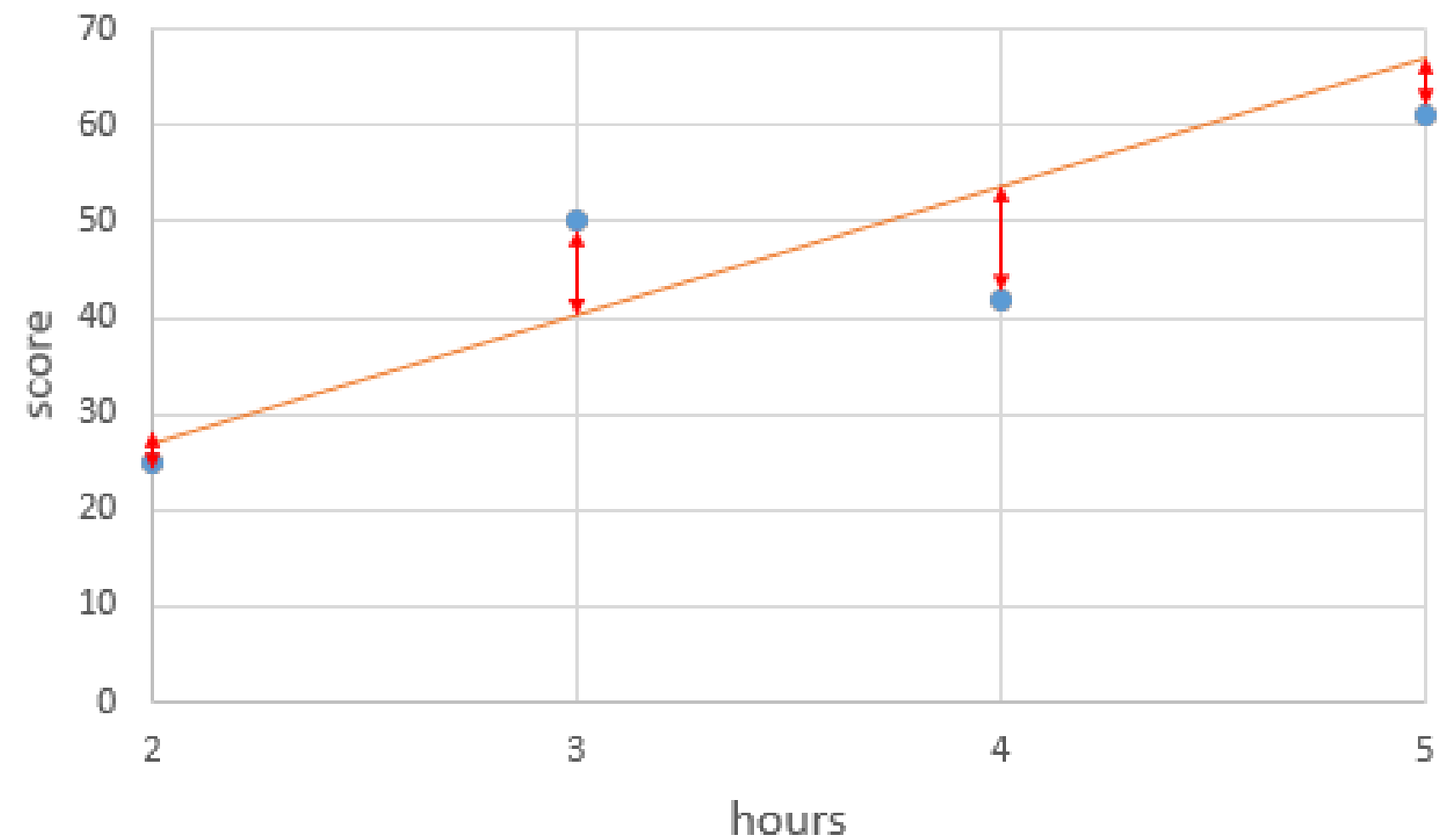
= 비용 함수(Cost function)

= 손실 함수(Loss function)

= 실제값과 예측값에 대한 오차에 대한 식

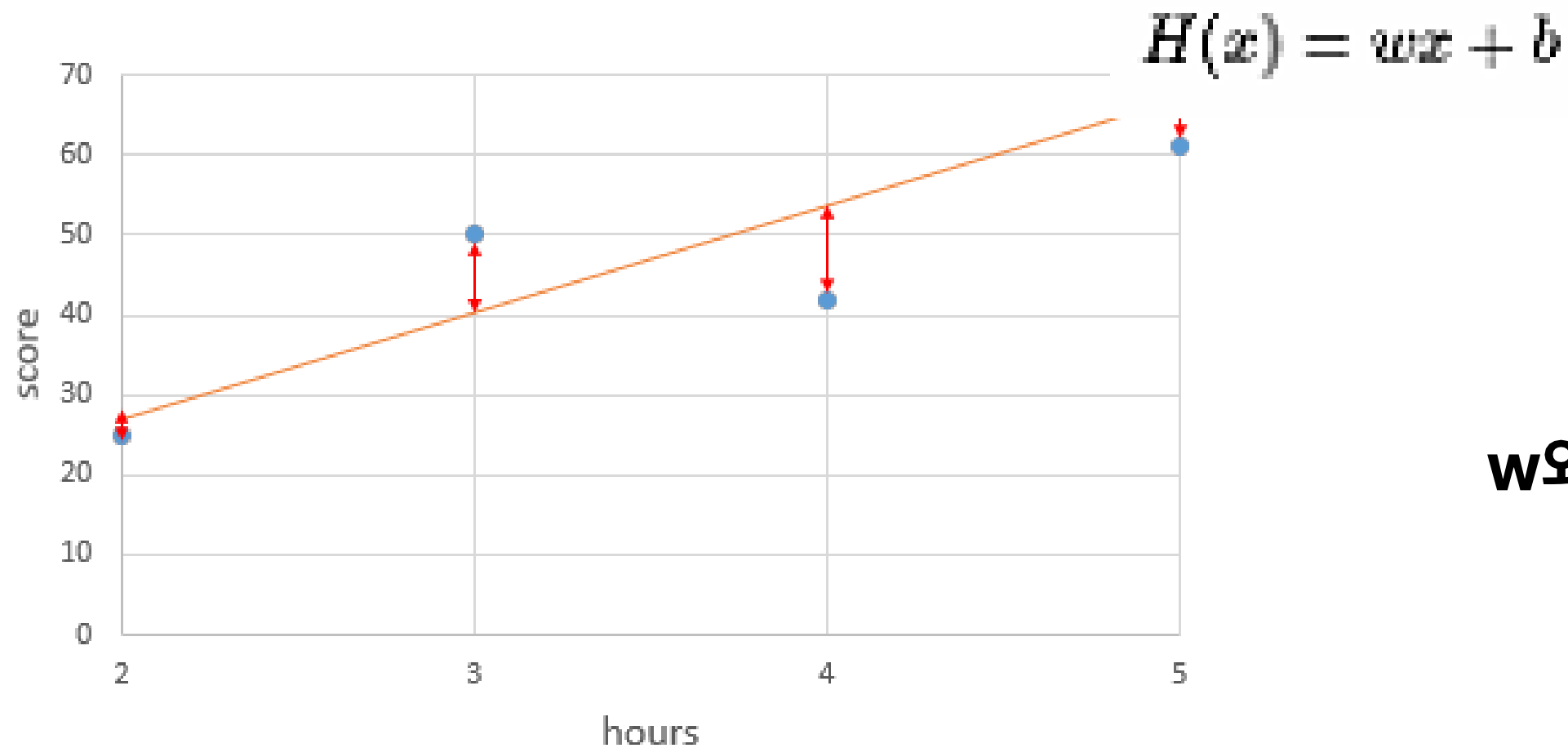
예측값의 오차를 줄이는 일에 최적화 된 식이어야 !

회귀 문제의 경우에는 주로 평균 제곱 오차
(Mean Squared Error, **MSE**)가 사용



06-03 선형 회귀(Linear Regression)

3. 비용 함수(Cost function) : 평균 제곱 오차(MSE)



오차 = 실제값과 $H(x)$ 의 차이

전체 오차의 크기를 최소값으로 만드는
 w 와 b 를 찾아내는 것이 정답인 직선을 찾아내는 것

전체 오차의 크기는 어떻게 구할까?

06-03 선형 회귀(Linear Regression)

3. 비용 함수(Cost function) : 평균 제곱 오차(MSE)


hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-9	-5

음수 오차도 있고, 양수 오차도 있으므로
모든 오차를 제곱하여 더하는 방법 사용

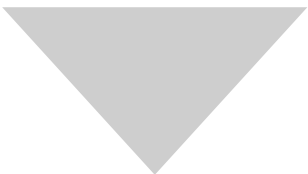
$$\sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = (-2)^2 + 10^2 + (-9)^2 + (-5)^2 = 210$$

06-03 선형 회귀(Linear Regression)

3. 비용 함수(Cost function) : 평균 제곱 오차(MSE)

$$\sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = (-2)^2 + 10^2 + (-9)^2 + (-5)^2 = 210$$



평균 제곱 오차의 값을 최소값으로 만드는
w와 b 를 찾아내는 것이 정답인 직선을 찾아내는 일

$$\frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = 210/4 = 52.5$$


평균 제곱 오차(Mean Squared Error, MSE)
:데이터의 개수인 n으로 나누면 > 오차의 제곱합에 대한 평균

모든 점들과의 오차가 클 수록 평균 제곱 오차는 커지며,
오차가 작아질 수록 평균 제곱 오차는 작아짐

비용함수

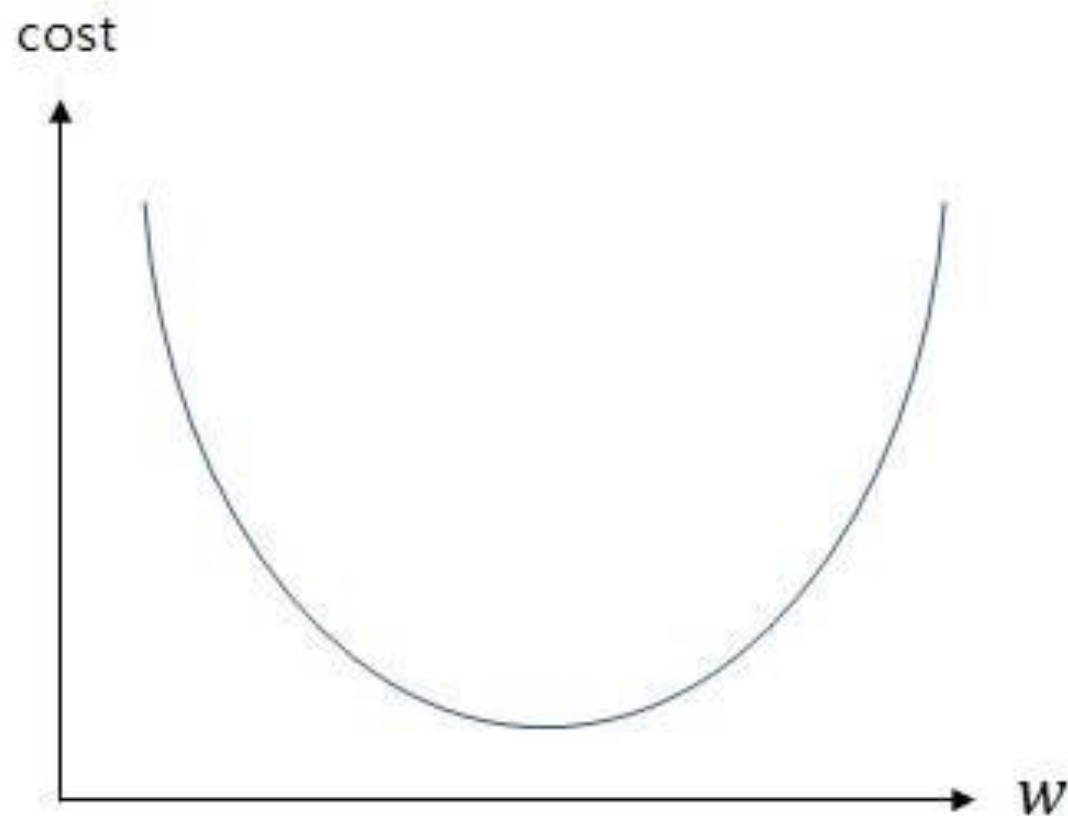
$$cost(w, b) = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$


$$w, b \rightarrow \text{minimize } cost(w, b)$$

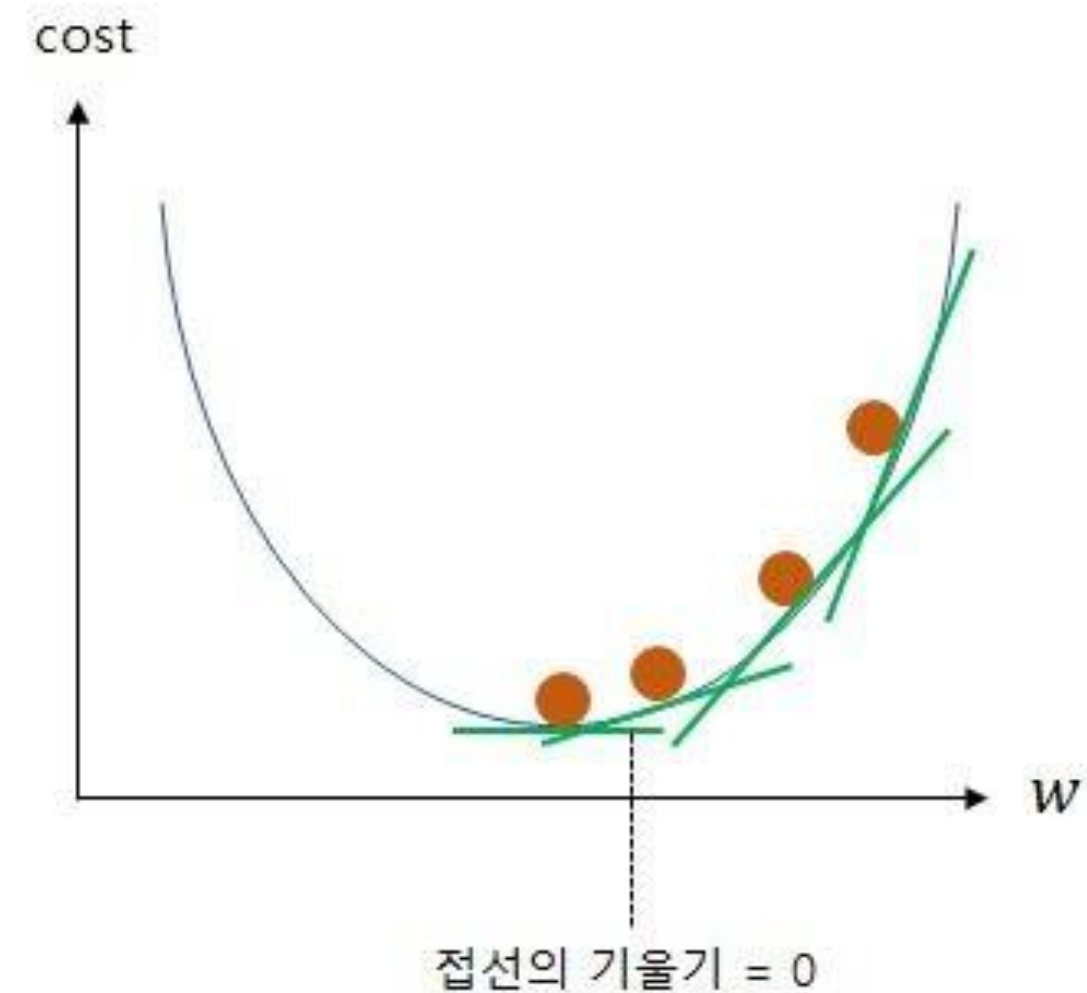
06-03 선형 회귀(Linear Regression)

4. 옵티마이저(Optimizer) : 경사하강법(Gradient Descent)

$y = wx$ 라는 가설 $H(x)$

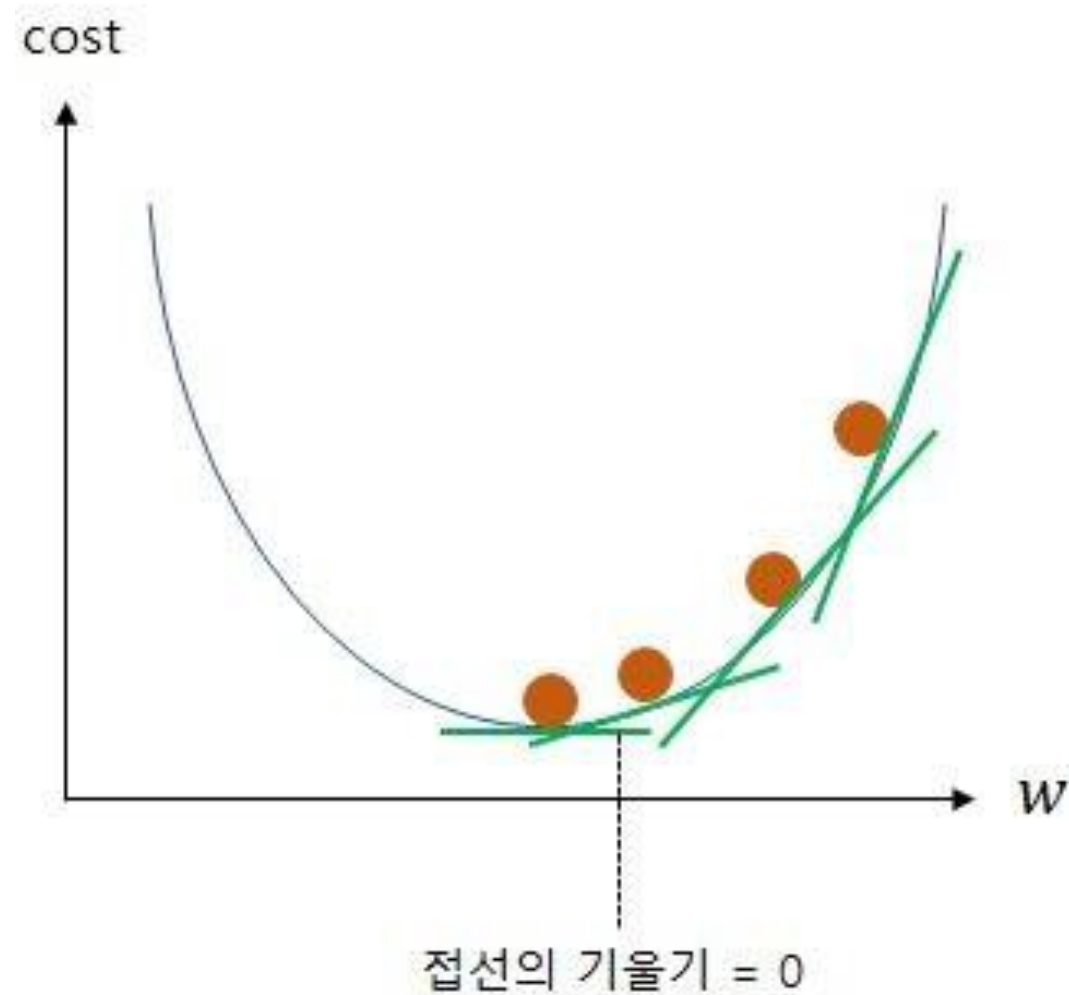


cost가 가장 최소값을 가지게 하는 w 를 찾는 일 (볼록한 부분의 맨 아래 부분)



06-03 선형 회귀(Linear Regression)

4. 옵티마이저(Optimizer) : 경사하강법(Gradient Descent)



cost가 최소화 되는 지점은 접선의 기울기가 0이 되는 지점
= 미분값이 0이 되는 지점

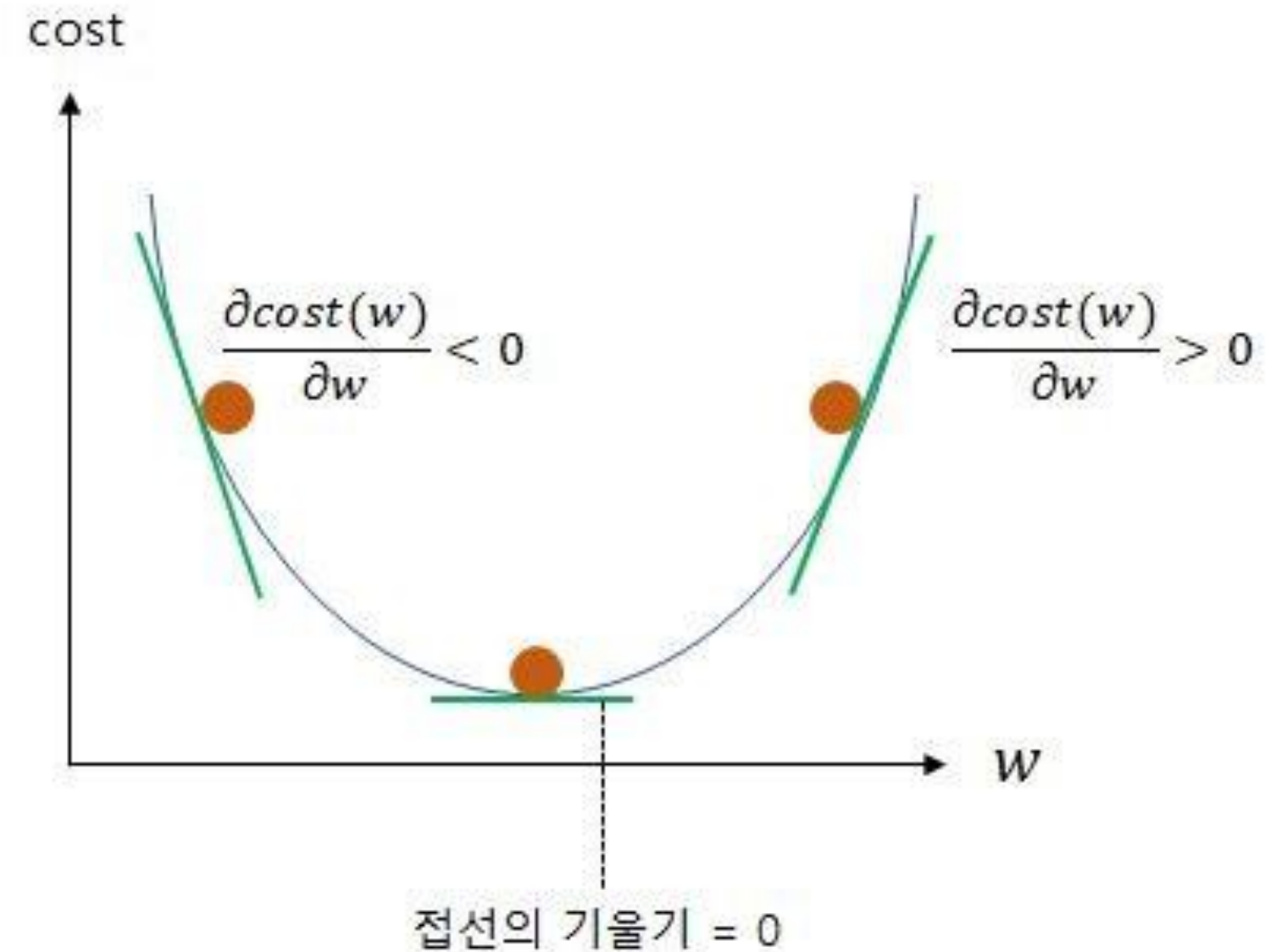
$$cost(w, b) = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

06-03 선형 회귀(Linear Regression)

4. 옵티마이저(Optimizer) : 경사하강법(Gradient Descent)

$$w := w - \alpha \frac{\partial}{\partial w} cost(w)$$

손실 함수를 최소화하는 방향으로 w 를 조정, 0이 될때까지 반복
 α = 학습률 = 접선의 기울기



접선의 기울기가 음수일 때, 0일때, 양수일 때

06-03 선형 회귀(Linear Regression)

4. 옵티마이저(Optimizer) : 경사하강법(Gradient Descent)

$$w := w - \alpha \frac{\partial}{\partial w} cost(w)$$

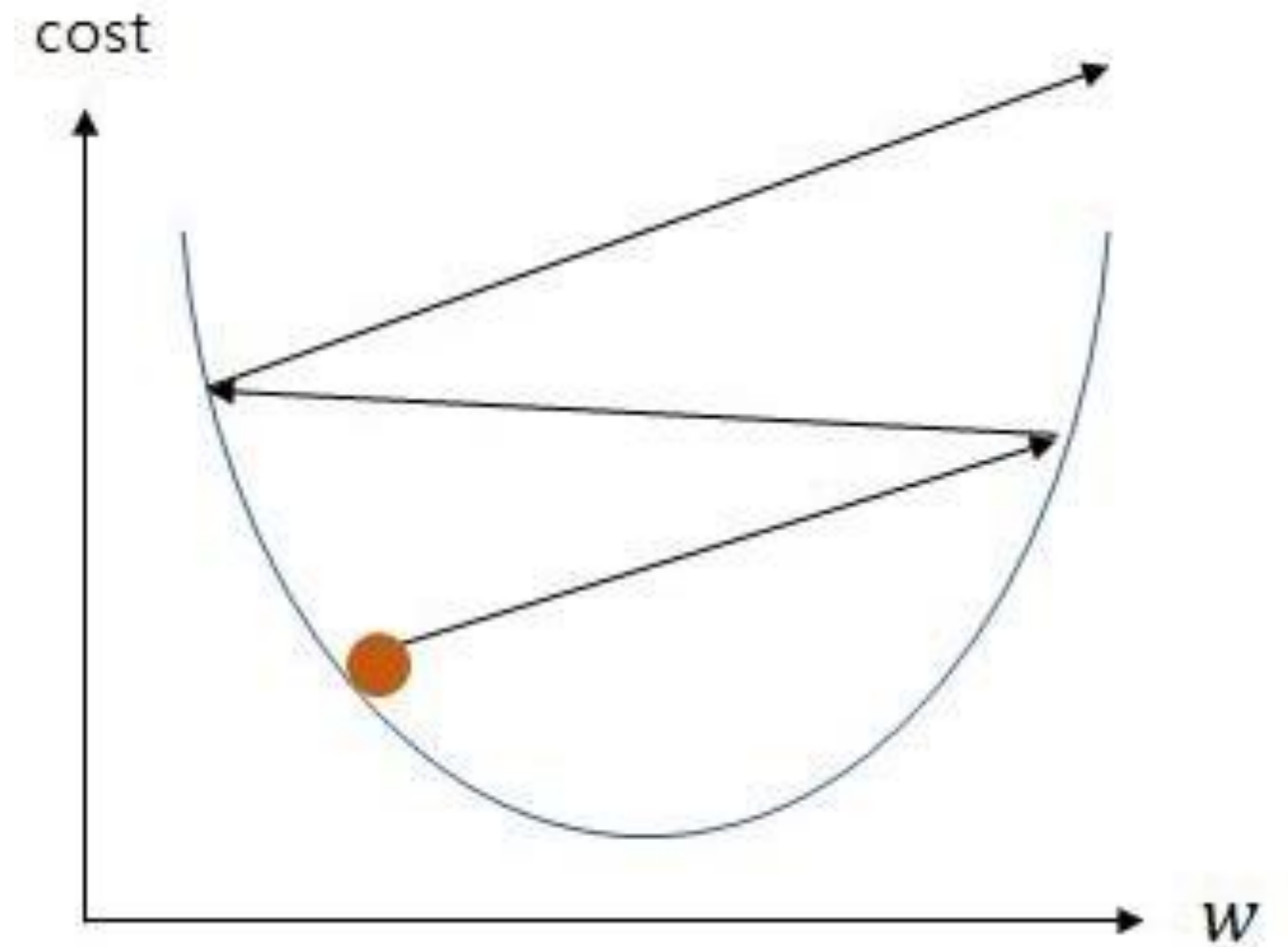
α = 학습률 = 접선의 기울기

학습률(learning rate)은

경사 하강법에서 얼마나 크게 파라미터를 업데이트할지를 결정하는 값

> 최소값을 찾기 위해 경사를 따라 내려가는데,

학습률은 한 번에 얼마나 멀리 내려갈지를 결정



학습률이 너무 크면 접선을 따라 이동하는 것이 아니라 발산
학습률은 문제에 따라 다르게 설정되며 실험을 통해 조정

06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

데이터 준비
입력 데이터(x)와 해당하는 실제값(y)를 준비

모델 구성
선형 회귀 모델의 가중치(w)와 편향(b)을 변수로 설정

가설 함수 정의
입력 변수(x)를 이용하여 예측값(y_{pred})을 계산하는 가설 함수

손실 함수 정의
예측값(y_{pred})과 실제값(y)의 차이를 계산하는 손실 함수

06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

옵티마이저 설정
모델의 파라미터를 업데이트하기 위한
옵티마이저를 설정

학습 과정
가설 함수를 이용하여 가설의 예측값(y_{pred})을 계산
손실 함수를 이용하여 예측값(y_{pred})과 실제값(y)의 차이를 계산
손실 함수 $cost$ 에 대한 파라미터(w, b)의 그래디언트(미분값)를 업데이트
옵티마이저를 사용하여 그래디언트를 이용해 파라미터(w, b)를 업데이트
위 과정을 반복

06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

```
# 학습될 가중치 변수를 선언
w = tf.Variable(4.0)
b = tf.Variable(1.0)
```

```
@tf.function
def hypothesis(x):
    return w*x + b
```

```
x_test = [3.5, 5, 5.5, 6]
print(hypothesis(x_test).numpy())
```

```
[15.  21.  23.  25.]
```

06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

```
@tf.function
def mse_loss(y_pred, y):
    # 두 개의 차이값을 제공을 해서 평균을 취한다.
    return tf.reduce_mean(tf.square(y_pred - y))
```

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적
```

```
optimizer = tf.optimizers.SGD(0.01)
```

06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

```
for i in range(301):
    with tf.GradientTape() as tape:
        # 현재 파라미터에 기반한 입력 x에 대한 예측값을 y_pred
        y_pred = hypothesis(x)

        # 평균 제곱 오차를 계산
        cost = mse_loss(y_pred, y)

        # 손실 함수에 대한 파라미터의 미분값 계산
        gradients = tape.gradient(cost, [w, b])

        # 파라미터 업데이트
        optimizer.apply_gradients(zip(gradients, [w, b]))

    if i % 10 == 0:
        print("epoch : {:3} | w의 값 : {:.5.4f} | b의 값 : {:.5.4} | cost : {:.5.6f}".format(i, w.numpy(), b.numpy(), cost))
```


06-04 자동 미분과 선형 회귀 실습

1. 자동 미분을 이용한 선형 회귀 구현

```
epoch : 0 | w의 값 : 8.2133 | b의 값 : 1.664 | cost : 1402.555542  
... 중략 ...  
epoch : 280 | w의 값 : 10.6221 | b의 값 : 1.191 | cost : 1.091434  
epoch : 290 | w의 값 : 10.6245 | b의 값 : 1.176 | cost : 1.088940  
epoch : 300 | w의 값 : 10.6269 | b의 값 : 1.161 | cost : 1.086645
```

```
x_test = [3.5, 5, 5.5, 6]  
print(hypothesis(x_test).numpy())
```

```
[38.35479  54.295143 59.608593 64.92204 ]
```

06-04 자동 미분과 선형 회귀 실습

2. 케라스로 구현하는 선형 회귀

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적

model = Sequential()

# 출력 y의 차원은 1. 입력 x의 차원(input_dim)은 1
# 선형 회귀이므로 activation은 'linear'
model.add(Dense(1, input_dim=1, activation='linear'))

# sgd는 경사 하강법을 의미. 학습률(learning rate, lr)은 0.01.
sgd = optimizers.SGD(lr=0.01)

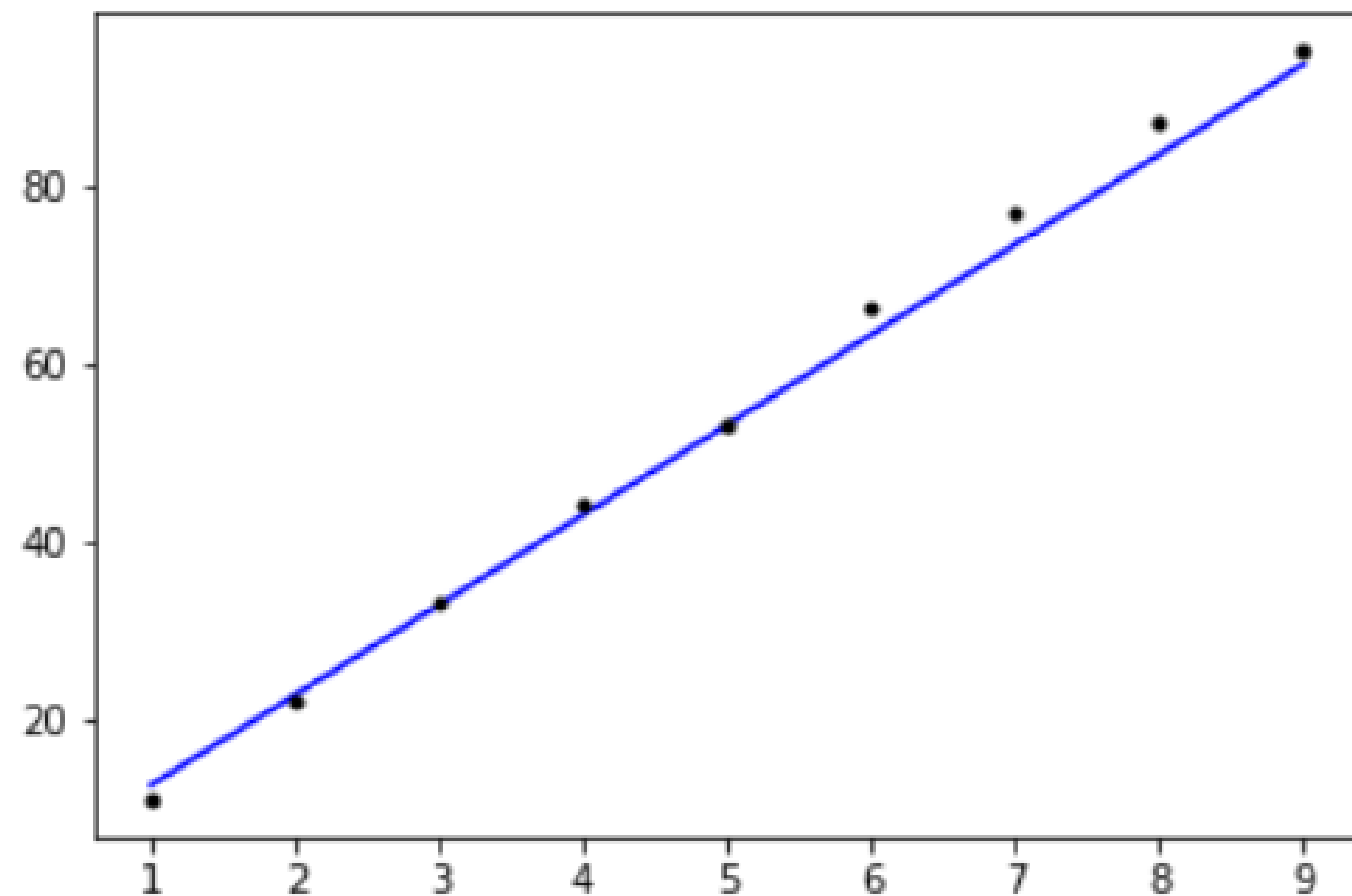
# 손실 함수(Loss function)은 평균제곱오차 mse를 사용합니다.
model.compile(optimizer=sgd, loss='mse', metrics=['mse'])

# 주어진 x와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.
model.fit(x, y, epochs=300)
```

06-04 자동 미분과 선형 회귀 실습

2. 케라스로 구현하는 선형 회귀

```
plt.plot(x, model.predict(x), 'b', x, y, 'k.')
```



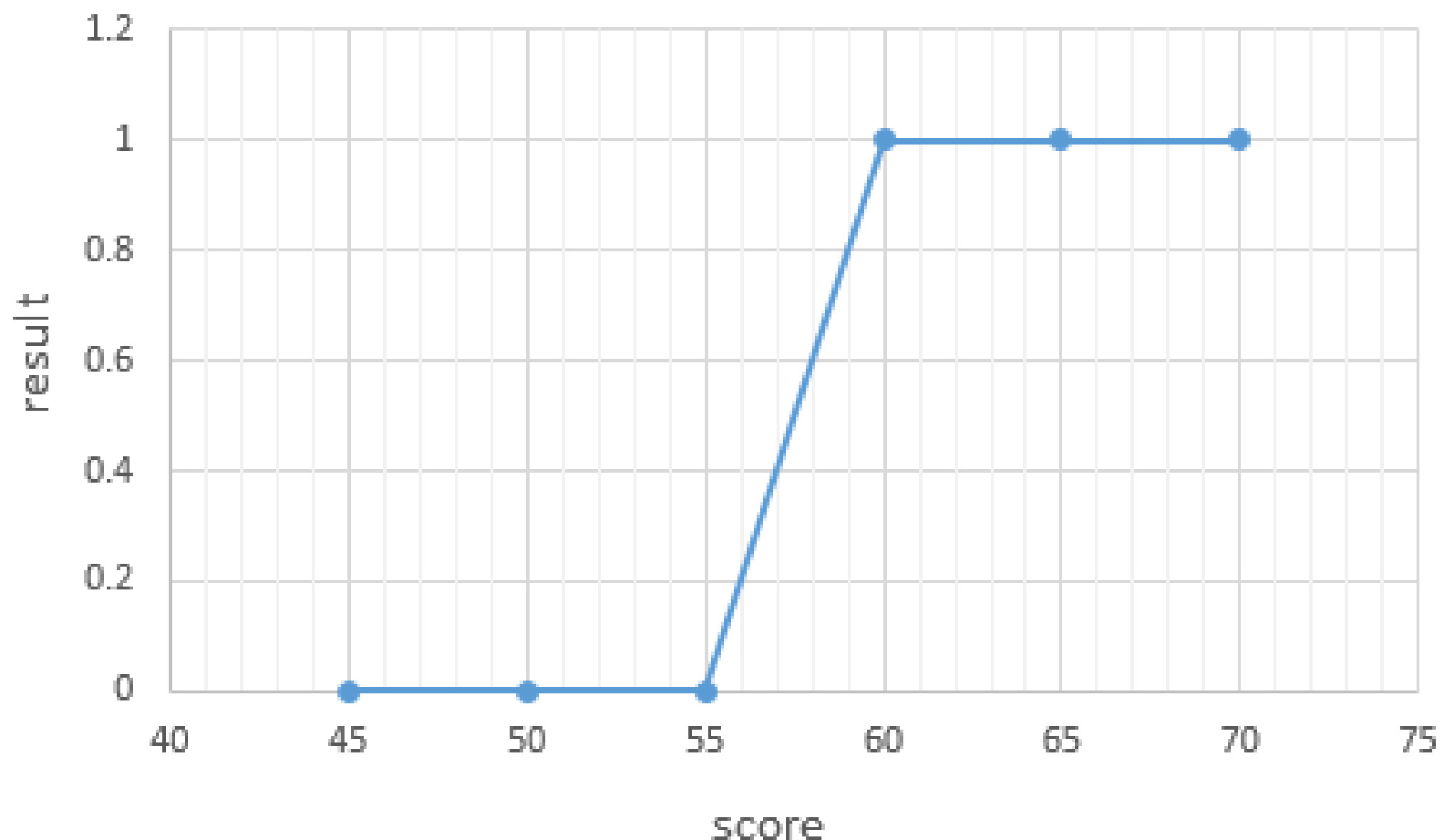
```
print(model.predict([9.5]))
```

```
[[98.556465]]
```

06-05 로지스틱 회귀 (Logistic Regression)

1. 이진 분류(Binary Classification)

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격

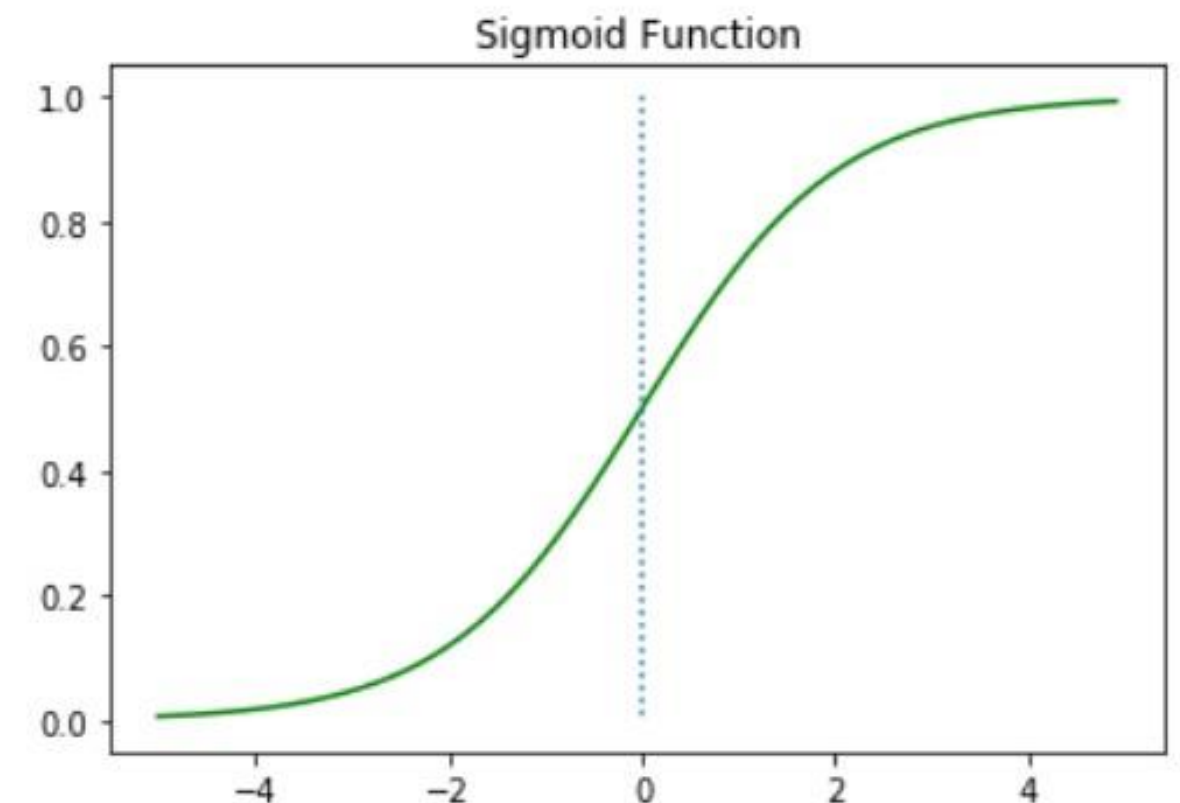


06-05 로지스틱 회귀 (Logistic Regression)

2. 시그모이드 함수(Sigmoid function): σ

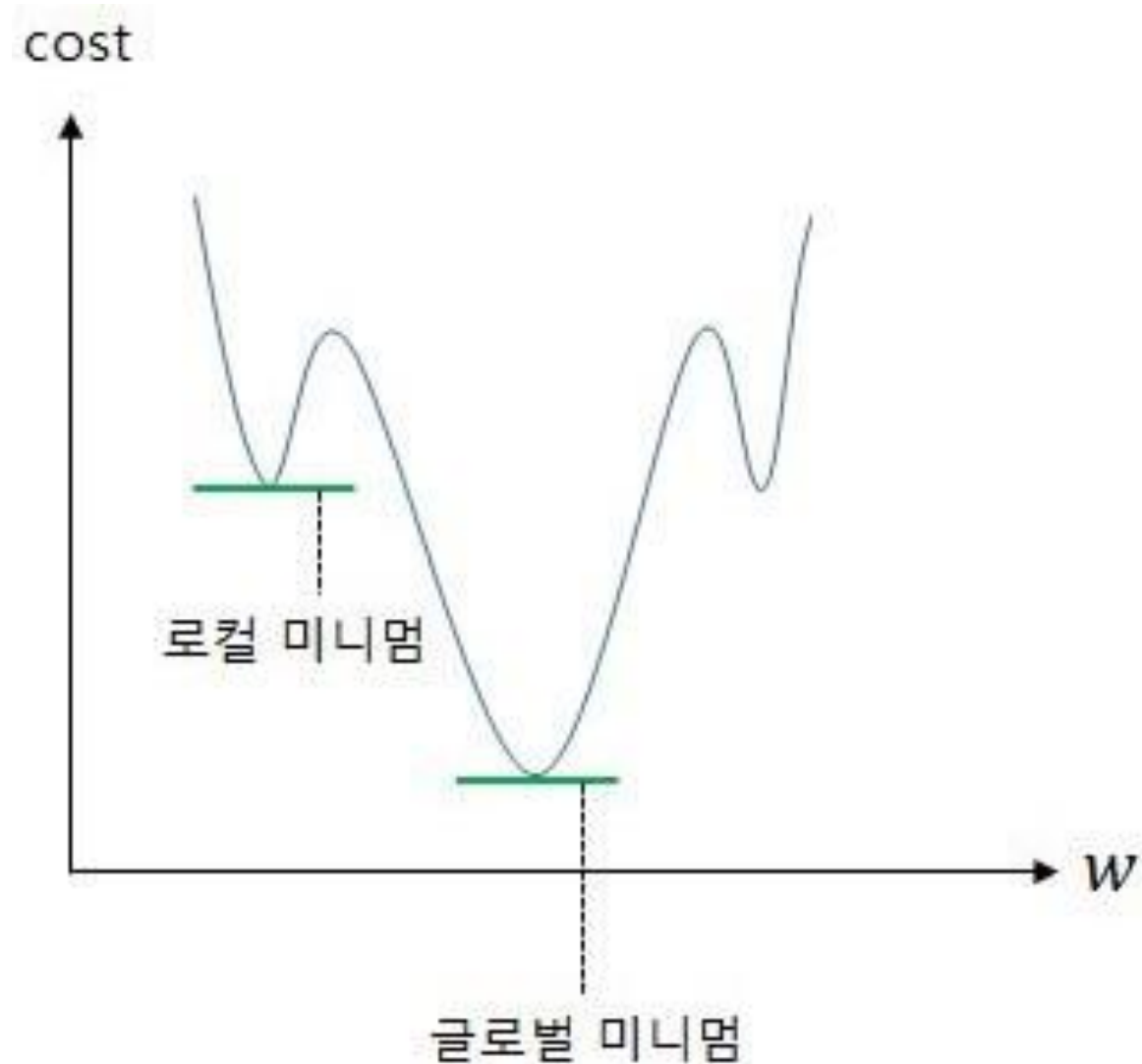
$$H(x) = \frac{1}{1 + e^{-(wx+b)}} = \text{sigmoid}(wx + b) = \sigma(wx + b)$$

```
def sigmoid(x):  
    return 1/(1+np.exp(-x))  
  
x = np.arange(-5.0, 5.0, 0.1)  
y = sigmoid(x)  
  
plt.plot(x, y, 'g')  
plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가  
plt.title('Sigmoid Function')  
plt.show()
```



06-05 로지스틱 회귀 (Logistic Regression)

3. 비용 함수(Cost function)



로지스틱 회귀에서 평균 제곱 오차를 비용 함수로 사용하면
> 경사 하강법을 통해 찾고자 하는 최소값이 아닌
특정 구역에서의 최소값인 로컬 미니멈에 빠질 가능성이 큼

이는 전체 함수에서의 최소값이 아닌
작은 구역에서의 최소값으로 갇혀있는 상황을 의미
잘못된 가중치가 선택

>> 로지스틱 회귀에서 평균 제곱 오차를 사용하는 것은
좋지 않은 선택

06-05 로지스틱 회귀 (Logistic Regression)

3. 비용 함수(Cost function)

로지스틱 회귀라는 문제에서
가중치를 **최소**로 만드는 적절한 **새로운 비용** 함수를 찾아야

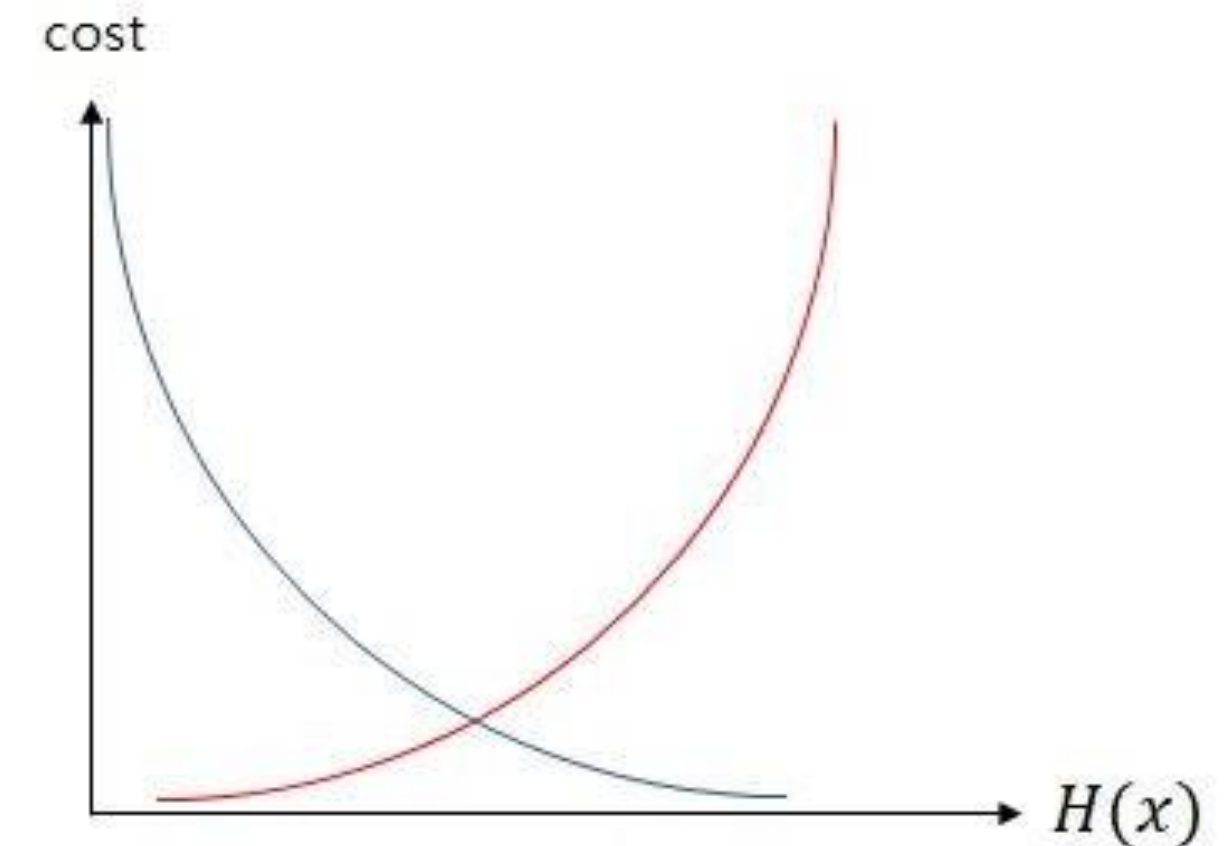
가중치를 최소화하는 함수를 목적 함수 J

$$J(w) = \frac{1}{n} \sum_{i=1}^n \text{cost} \left(H(x^{(i)}), y^{(i)} \right)$$

$$J(w) = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log H(x^{(i)}) + (1 - y^{(i)}) \log(1 - H(x^{(i)}))]$$

$$\text{if } y = 1 \rightarrow \text{cost} (H(x), y) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost} (H(x), y) = -\log(1 - H(x))$$



06-06 로지스틱 회귀 실습

1. 케라스로 구현하는 로지스틱 회귀

```
x = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50])
y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10부터 1

model = Sequential()
model.add(Dense(1, input_dim=1, activation='sigmoid'))

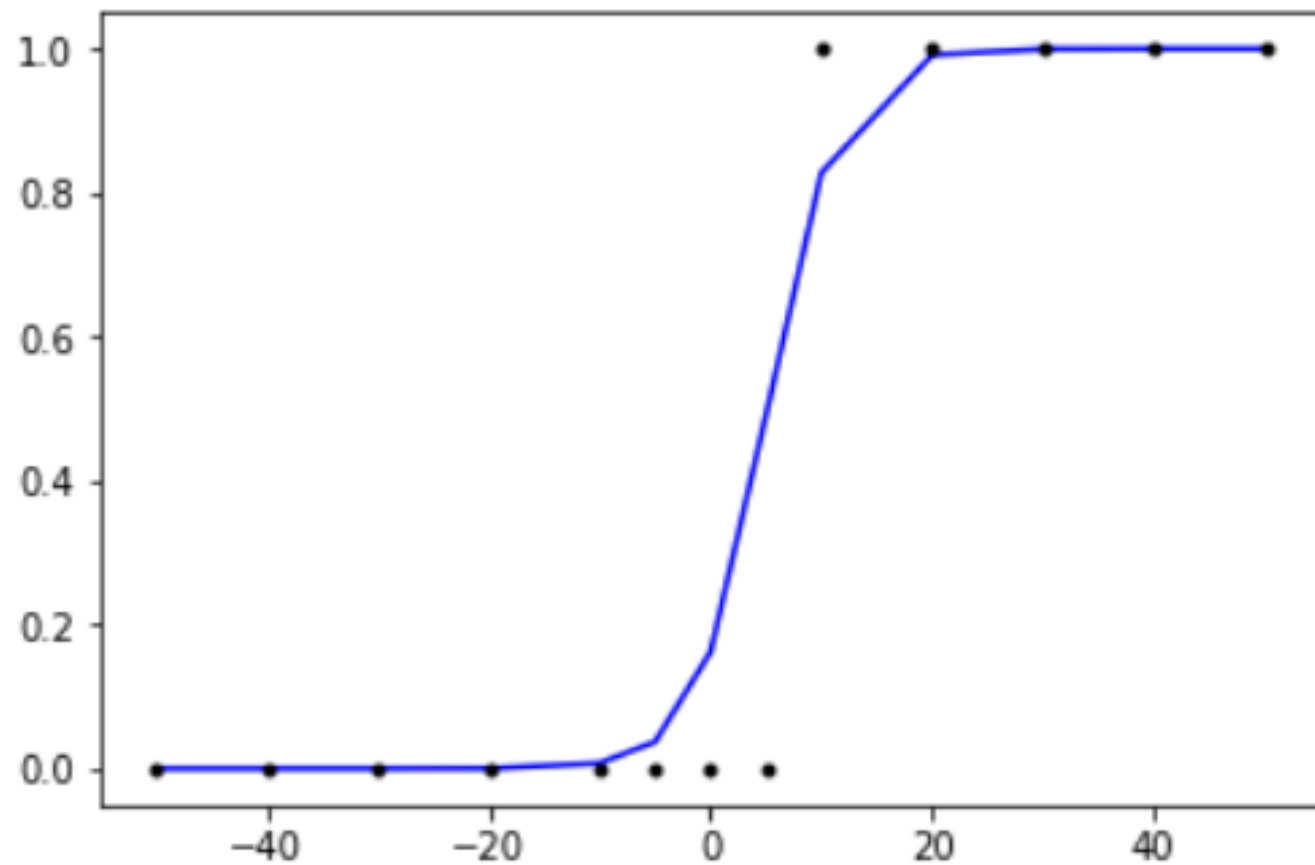
sgd = optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['binary_accuracy'])

model.fit(x, y, epochs=200)
```


06-06 로지스틱 회귀 실습

1. 케라스로 구현하는 로지스틱 회귀

```
plt.plot(x, model.predict(x), 'b', x,y, 'k.')
```



06-06 로지스틱 회귀 실습

1. 케라스로 구현하는 로지스틱 회귀

```
print(model.predict([1, 2, 3, 4, 4.5]))  
print(model.predict([11, 21, 31, 41, 500]))
```

```
[[0.21071826]  
 [0.26909265]  
 [0.33673897]  
 [0.41180944]  
 [0.45120454]]  
[[0.86910886]  
 [0.99398106]  
 [0.99975663]  
 [0.9999902 ]  
 [1.          ]]
```

06-07 다중 입력에 대한 실습

1. 다중 선형 회귀

Midterm(x_1)	Final(x_2)	Added point(x_3)	Score(\$1000)(y)
70	85	11	73
71	89	18	82
50	80	20	72
99	20	10	57
50	10	10	34
20	99	10	58
40	50	20	56

$$H(X) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

06-07 다중 입력에 대한 실습

1. 다중 선형 회귀

```
# 중간 고사, 기말 고사, 가산점 점수
X = np.array([[70,85,11], [71,89,18], [50,80,20], [99,20,10], [50,10,10]])
y = np.array([73, 82 ,72, 57, 34]) # 최종 성적

model = Sequential()
model.add(Dense(1, input_dim=3, activation='linear'))

sgd = optimizers.SGD(lr=0.0001)
model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
model.fit(X, y, epochs=2000)
```

06-07 다중 입력에 대한 실습

1. 다중 선형 회귀

```
print(model.predict(X))
```

```
[[73.15294 ]  
 [81.98001 ]  
 [71.93192 ]  
 [57.161617]  
 [33.669353]]
```

06-07 다중 입력에 대한 실습

1. 다중 선형 회귀

```
X_test = np.array([[20,99,10], [40,50,20]])  
print(model.predict(X_test))
```

```
[[58.08134 ]  
 [55.734634]]
```

06-07 다중 입력에 대한 실습

2. 다중 로지스틱 회귀

SepalLengthCm(x_1)	PetalLengthCm(x_2)	Species(y)
5.1	3.5	A
4.7	3.2	A
5.2	1.8	B
7	4.1	A
5.1	2.1	B

$$H(X) = \text{sigmoid}(w_1x_1 + w_2x_2 + b)$$

06-07 다중 입력에 대한 실습

2. 다중 로지스틱 회귀

```
X = np.array([[0, 0], [0, 1], [1, 0], [0, 2], [1, 1], [2, 0]])
y = np.array([0, 0, 0, 1, 1, 1])

model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])

model.fit(X, y, epochs=2000)
```


06-07 다중 입력에 대한 실습

2. 다중 로지스틱 회귀

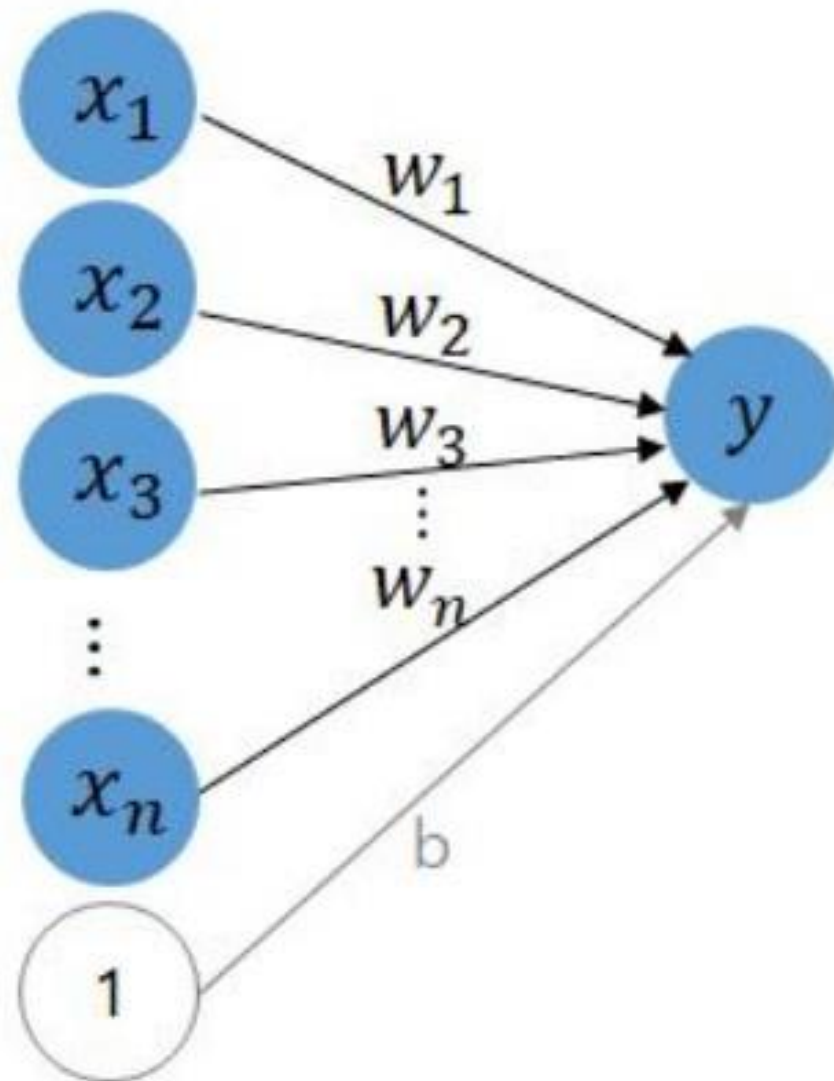
```
print(model.predict(X))
```

```
[[ 0.23379876]  
 [ 0.48773268]  
 [ 0.4808667 ]  
 [ 0.7481605 ]  
 [ 0.74294543]  
 [ 0.7376603 ]]
```

06-07 다중 입력에 대한 실습

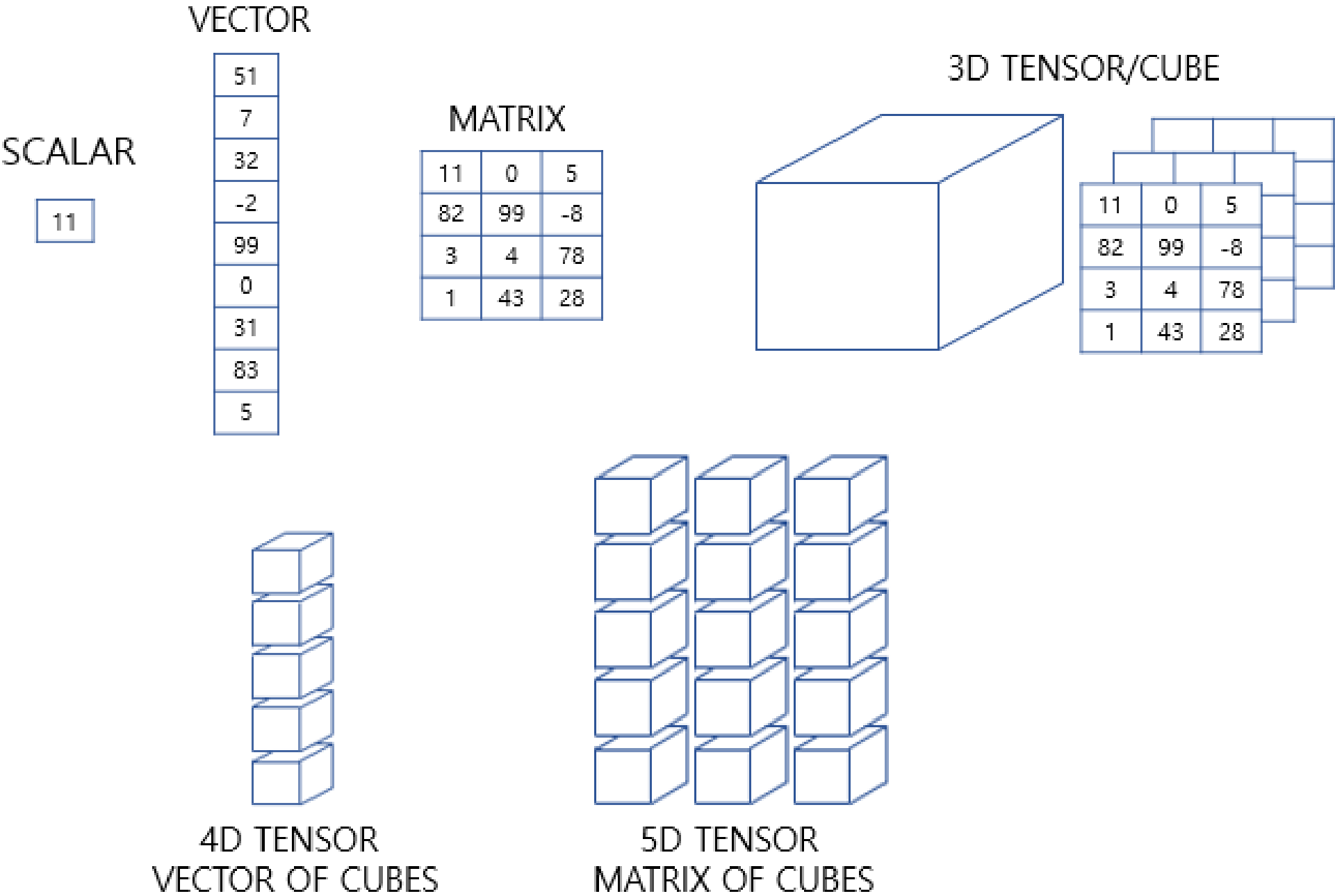
3. 인공 신경망 다이어그램

$$y = \text{sigmoid}(w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b) = \sigma(w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b)$$




06-08 벡터와 행렬 연산

1. 텐서(Tensor)




06-08 벡터와 행렬 연산


2. 가중치와 편향 행렬의 크기 결정

$$X_{m \times n} \times W_{? \times ?} + B_{m \times j} = Y_{m \times j}$$


같아야 한다

$$X_{m \times n} \times W_{n \times ?} + B_{m \times j} = Y_{m \times j}$$


같아야 한다

$$X_{m \times n} \times W_{n \times j} + B_{m \times j} = Y_{m \times j}$$


같아야 한다

06-09 소프트맥스 회귀 (Softmax Regression)

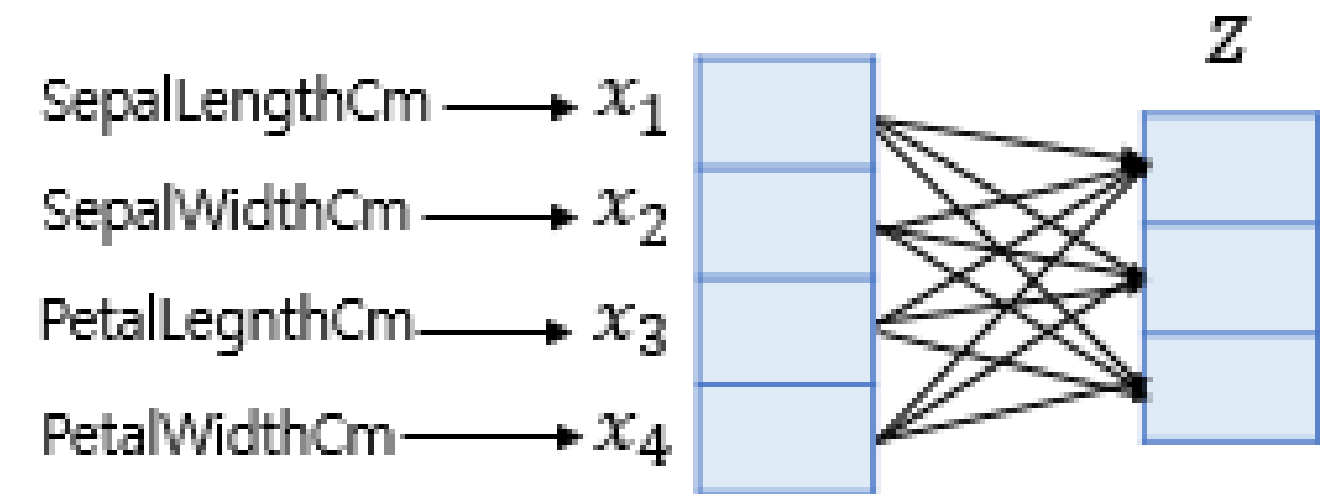
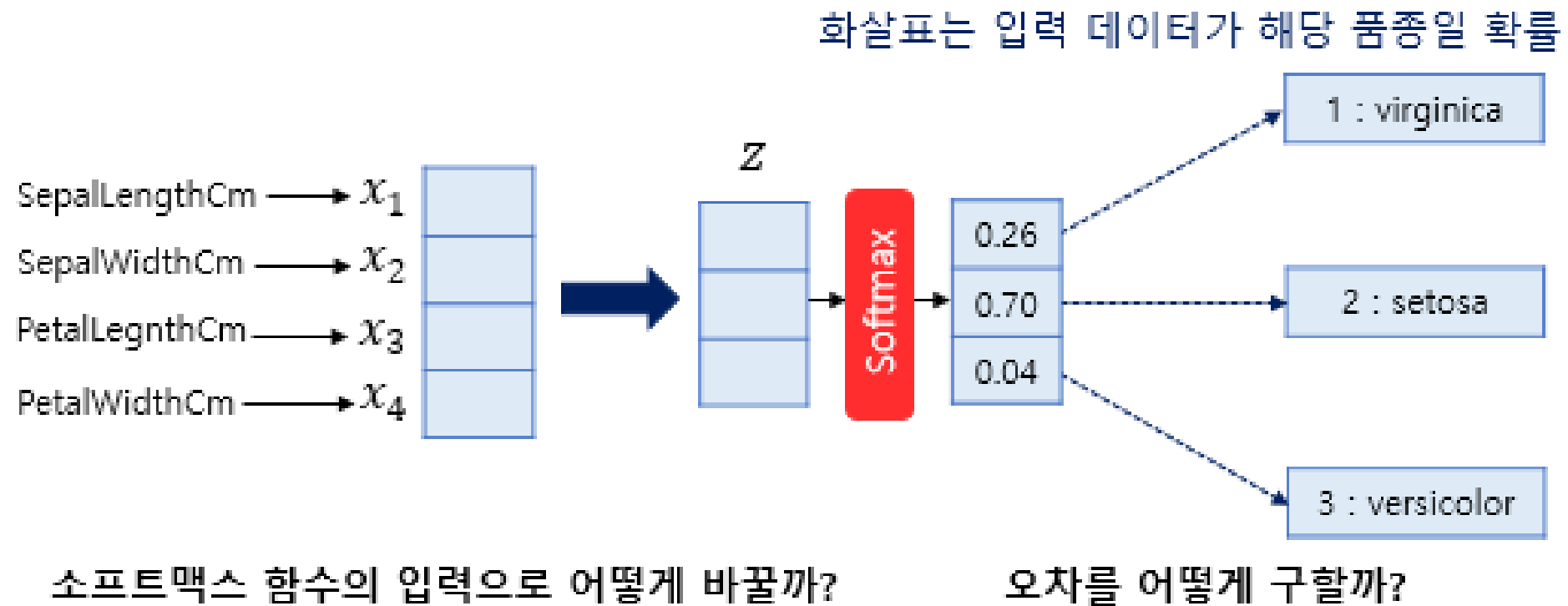
1. 소프트맥스 함수(Softmax function)

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \text{ for } i = 1, 2, \dots, k$$

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{j=1}^3 e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^3 e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^3 e^{z_j}} \right] = [p_1, p_2, p_3] = [p_{\text{virginica}}, p_{\text{setosa}}, p_{\text{versicolor}}]$$

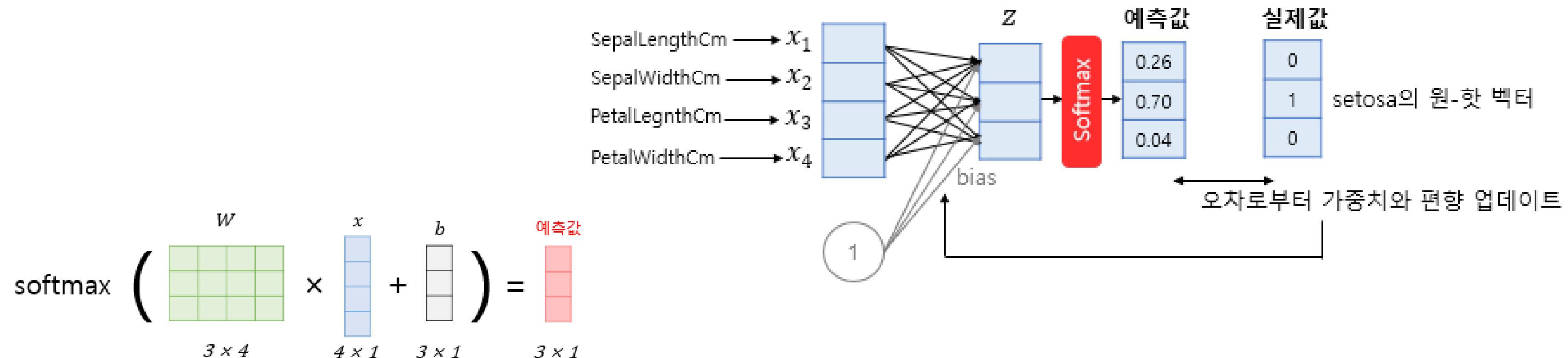
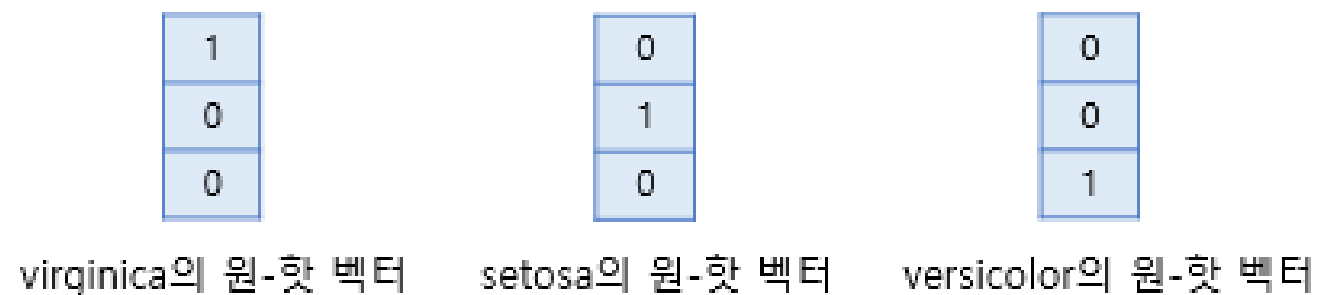
06-09 소프트맥스 회귀 (Softmax Regression)

1. 소프트맥스 함수(Softmax function)



06-09 소프트맥스 회귀 (Softmax Regression)

1. 소프트맥스 함수(Softmax function)



06-09 소프트맥스 회귀 (Softmax Regression)

2. 비용 함수(Cost function)

1) 크로스 엔트로피 함수

$$cost = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k y_j^{(i)} \log(p_j^{(i)})$$

2) 이진 분류에서의 크로스 엔트로피 함수

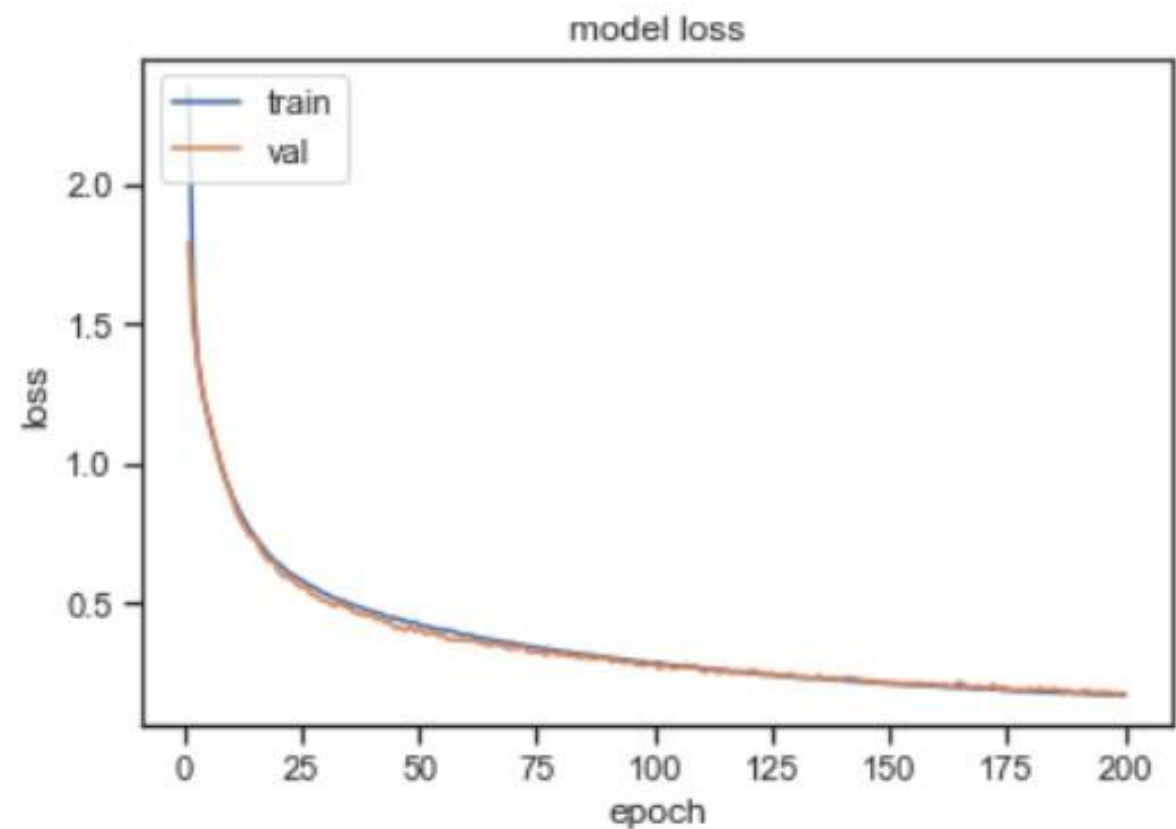
$$cost = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k y_j^{(i)} \log(p_j^{(i)}) = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)})]$$

06-10 소프트맥스 회귀 실습

```
model = Sequential()  
model.add(Dense(3, input_dim=4, activation='softmax'))  
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  
history = model.fit(X_train, y_train, epochs=200, batch_size=1, validation_data=(X_test, y_test))
```

06-10 소프트맥스 회귀 실습

```
epochs = range(1, len(history.history['accuracy']) + 1)
plt.plot(epochs, history.history['loss'])
plt.plot(epochs, history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



06-10 소프트맥스 회귀 실습

```
print("\n 테스트 정확도: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

```
테스트 정확도: 0.9667
```