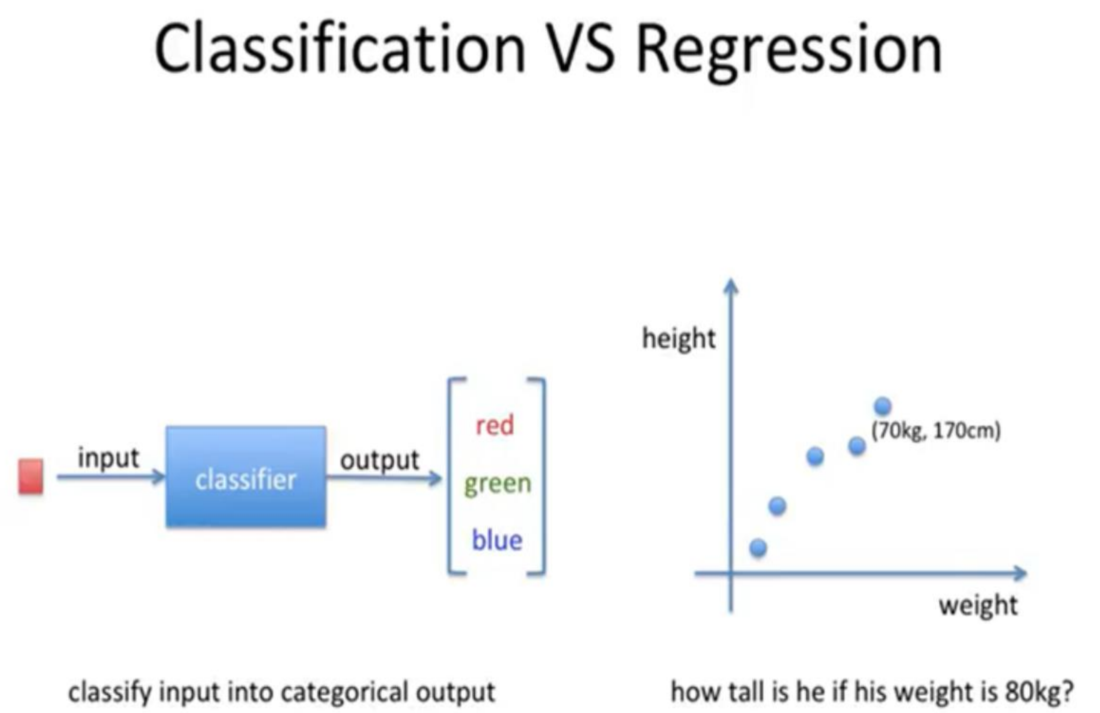
회귀와 분류 (regression and classification)

회귀(regresson)와 분류(classification)

- **회귀 모델**
  - 연속적인 값을 예측
    - **캘리포니아의 주택 가격이 얼마인가요?**
    - **사용자가 이 광고를 클릭할 확률이 얼마인가요?**
- **분류 모델**
  - 불연속적인 값을 예측
    - **주어진 이메일 메시지가 스팸인가요, 스팸이 아닌가요?**
    - **이 이미지가 강아지, 고양이 또는 햄스터의 이미지인가요?**



회귀의 어원

- **회귀 분석(regression analysis)**
  - 관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법
  - 회귀분석은 시간에 따라 변화하는 데이터나 어떤 영향, 가설적 실험, 인과 관계의 모델링 등의 통계적 예측에 이용
- **회귀(영어: regress 리그레스[*])의 원래 의미**
  - 옛날 상태로 돌아가는 것을 의미
  - 영국의 유전학자 프랜시스 골턴은 "평균으로의 회귀(regression to the mean)"
    - **부모의 키와 아이들의 키 사이의 연관 관계를 연구하면서 부모와 자녀의 키 사이에는 선형적인 관계가 있고 키가 커지거나 작아지는 것보다는 전체 키 평균으로 돌아가려는 경향이 있다는 가설을 세웠으며 이를 분석하는 방법을 "회귀분석"이라고 함**
    - **이러한 경험적 연구 이후, 칼 피어슨은 아버지와 아들의 키를 조사한 결과를 바탕으로 함수 관계를 도출하여 회귀분석 이론을 수학적으로 정립**

선형 회귀
(linear regression)

선형 회귀와 로지스틱 회귀

- **단순 선형 회귀 분석(Simple Linear Regression Analysis)**
  - 입력: 특징이 하나
  - 출력: 하나의 값
    - **키로 몸무게 추정**

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

- **다중 선형 회귀 분석(Multiple Linear Regression Analysis)**
  - 입력: 특징이 여러 개, 출력: 하나의 값
    - **역세권, 아파트 평수, 주소로 아파트값을 추정**

$$y = W_1 x_1 + W_2 x_2 + \ldots W_n x_n + b$$

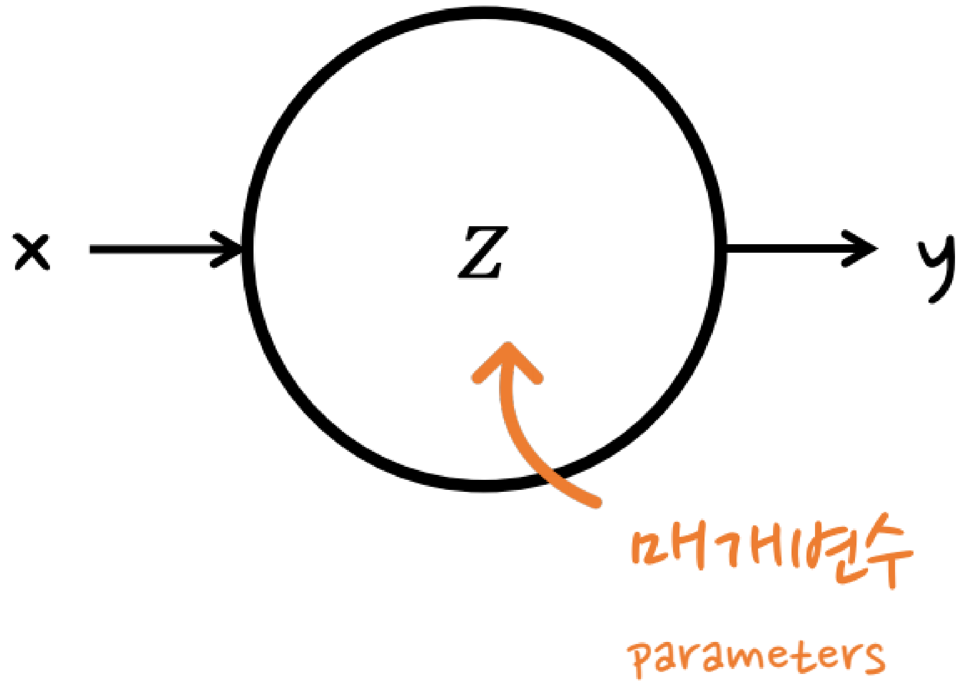
- **로지스틱 회귀(Logistic Regression)**
  - 이진 분류(Binary Classification)
  - 입력: 하나 또는 여러 개, 출력: 0 아니면 1
    - **타이타닉의 승객 정보로 죽음을 추정**

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

인공지능이란? W와 b 구하기

- **다음 식에서 가중치 W와 편향 b를 구하기**
  - W와 b를 매개변수 함

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



$$\theta = (w, b)$$

주요 용어 정리

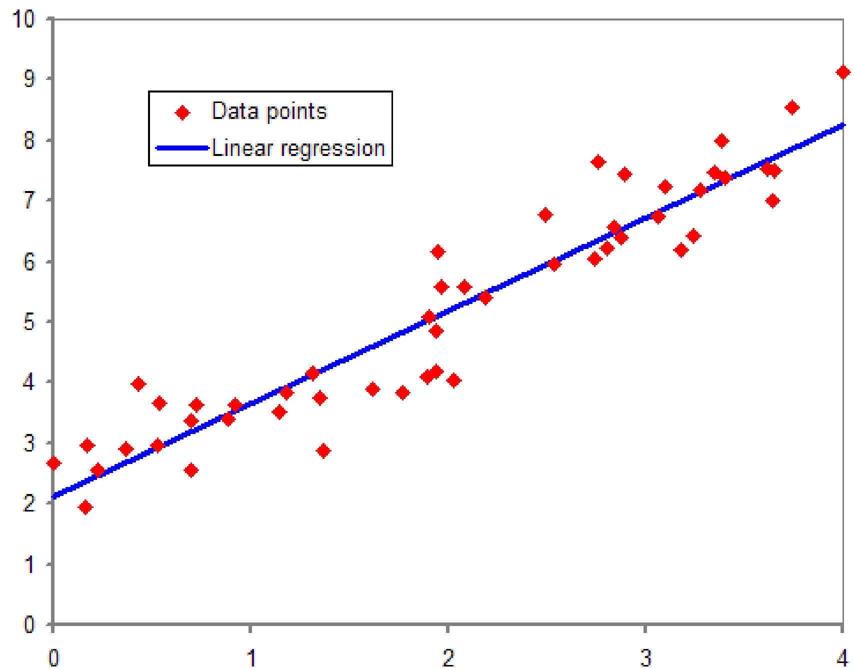
- **가설(Hypothesis)**
  - 가중치(weight)와 편향(bias)
  - 기울기와 절편
- **손실 함수(Loss Function)**
  - MSE(Mean Square Error 평균제곱오차)
  - Categorical crossentropy
  - Sparse Categorical crossentropy
- **경사 하강법(Gradient Descent)**
  - 내리막 경사 따라 가기
- **학습률(learning rate)**
  - 대표적인 하이퍼패러미터

<u>패러미터와 하이퍼패라미터</u>

학습에 의해 결정되는 값과
프로그래머가 결정하는 값

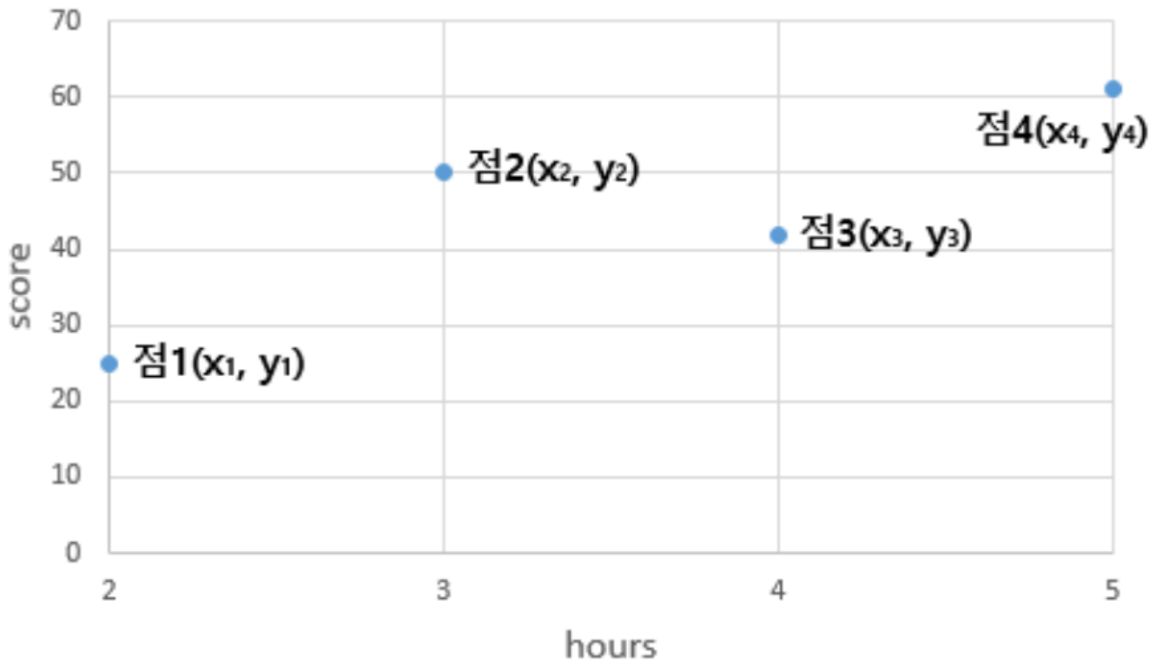
선형 회귀

- **Linear regression**
  - 데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법
    - **Y = aX + b**
      - **기울기 a와 절편인 b를 구하는 것**
  - 사례
    - **국어와 수학 성적**
    - **키와 몸무게**
    - **치킨과 맥주의 판매량**
    - **기저귀와 맥주의 판매량**
- **딥러닝 분야에서**
  - 선형 회귀
    - **Y = wX + b**
      - **가중치 w와 편향인 b를 구하는 것**



선형 회귀 문제 사례

- 공부 시간이 x라면, 점수는 y



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

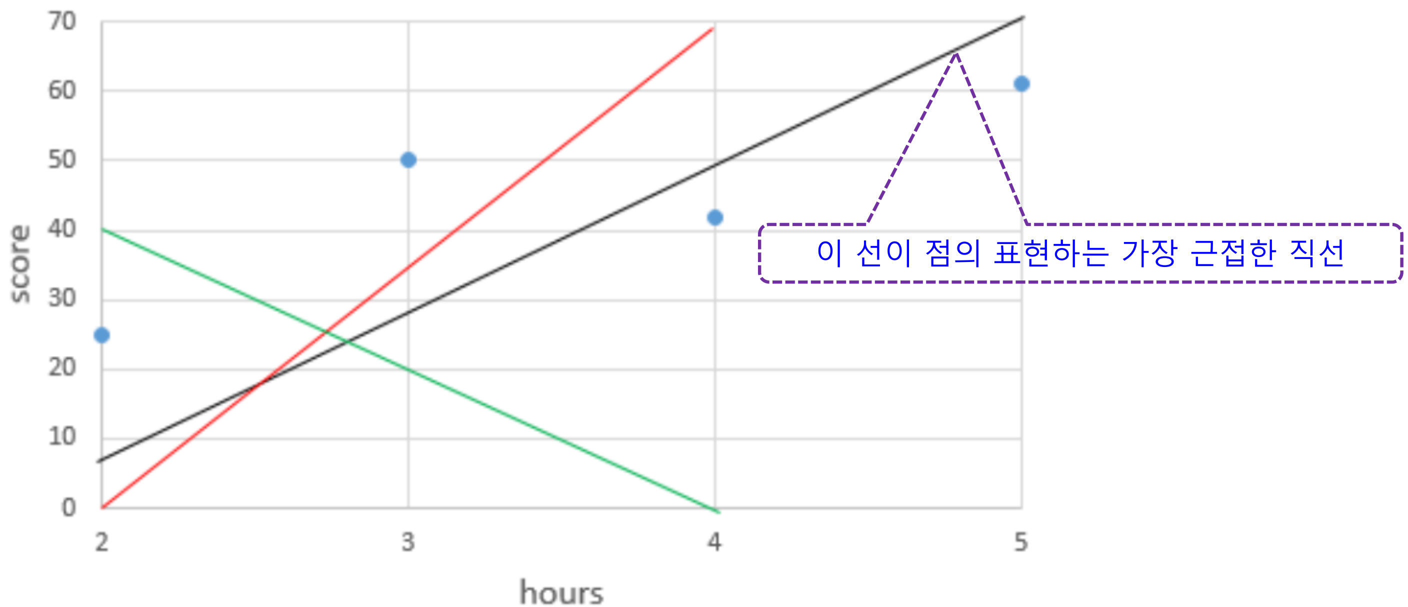
- 알려준 데이터로부터 x와 y의 관계를 유추
  - 학생이 6시간을 공부하였을 때의 성적
  - 그리고 7시간, 8시간을 공부하였을 때의 성적을 예측

가설

- **머신 러닝: y와 x간의 관계를 유추한 식을 가설(Hypothesis)**
  - H(x)에서 H는 Hypothesis를 의미

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

W:기울기, 가중치
b: 절편, 편향



- **선형 회귀에서 해야할 일은 결국 적절한 W와 b를 찾아내는 일**
  - 딥러닝 알고리즘이 하는 것이 바로 적절한 W와 b를 찾아내는 일

손실 함수(Loss function)

- **머신 러닝은 W와 b를 찾기 위해서**
  - 손실 함수를 정의
    - **실제 값과 가설로부터 얻은 예측 값의 오차를 계산하는 식**
  - 손실 함수 값을 최소화하는 최적의 W와 b를 찾아내려고 노력
- **손실 함수(Loss function)**
  - 목적 함수(Objective function), 비용 함수(Cost function)라고도 부름
  - 실제 값과 예측 값에 대한 오차에 대한 식
    - **예측 값의 오차를 줄이는 일에 최적화 된 식**

  - 평균 제곱 오차(Mean Squared Error, MSE) 등을 사용

$$\frac{1}{n}\sum_{i}^{n}\left[y_i - H(x_i)\right]^2$$

실제 값

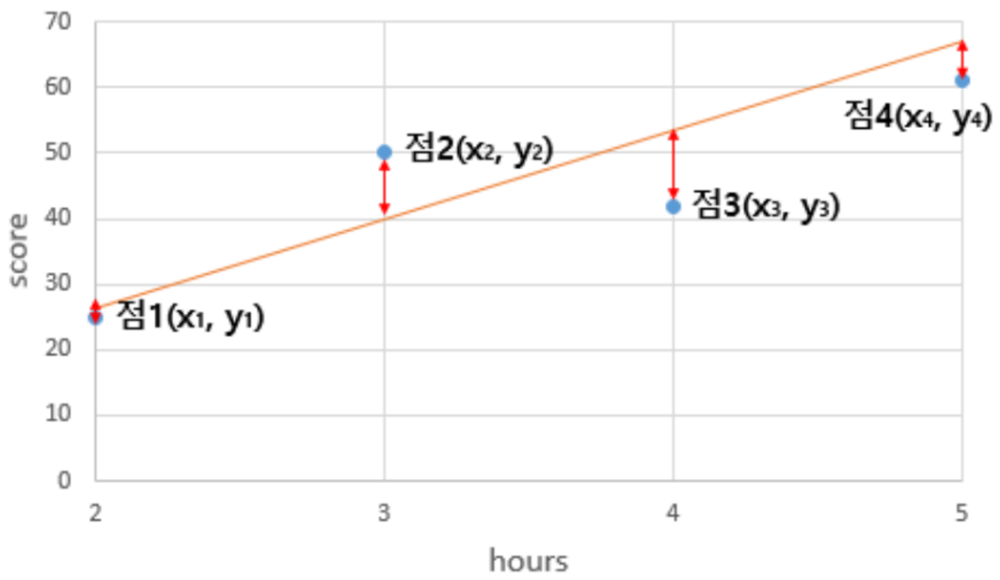
예측 값

손실 함수: MSE

- **W 와 b의 값을 찾아내기 위해 오차의 크기를 측정할 방법이 필요**
  - W: 13 b: 1로 예측한다면 y=13x+1 직선이 예측한 함수로 예측 값을 추정

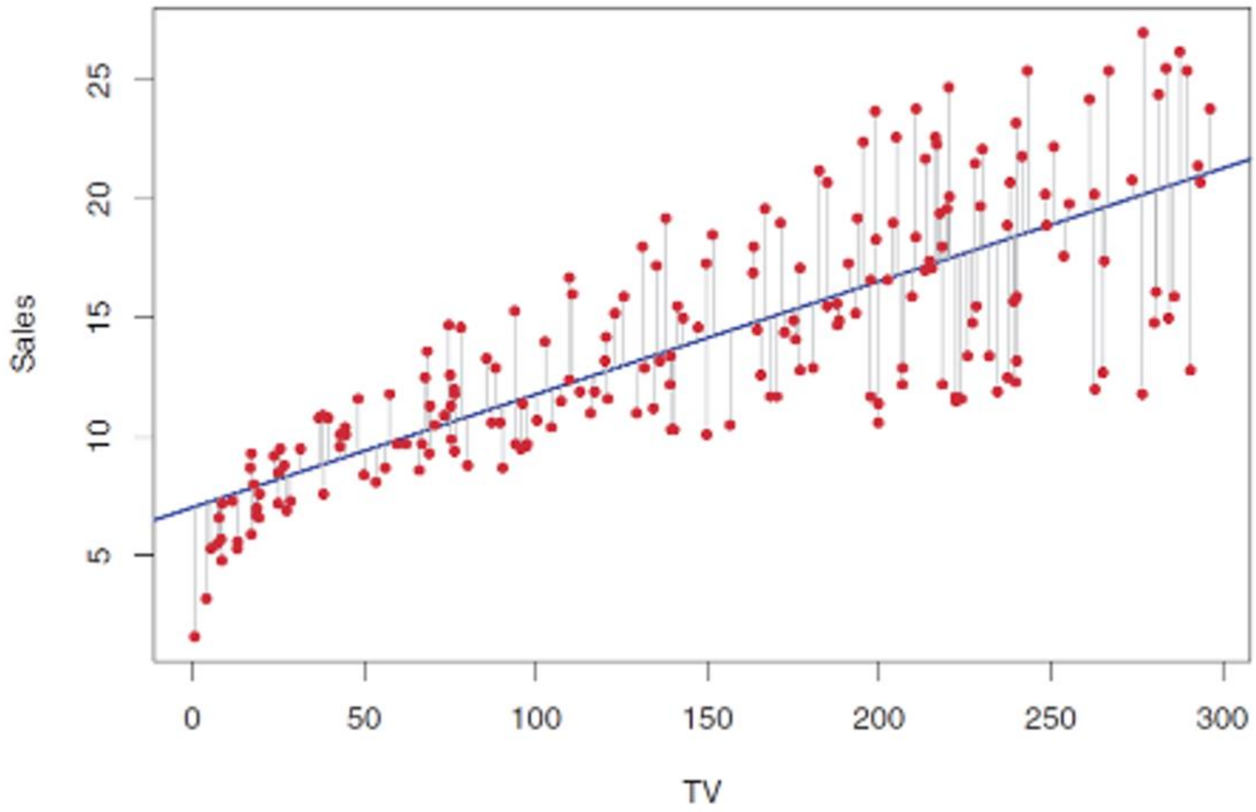
hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\frac{1}{n}\sum_{i}^{n}\left[y_i - H(x_i)\right]^2$$



손실 함수 MSE 이해

- **MSE**
  – 오차는 실제 데이터(빨간 점)와 예측 선(파란 선)의 차이의 제곱의 합



손실 함수를 W와 b의 함수로

- **평균 제곱 오차를 W와 b에 의한 비용 함수(Cost function)로 재정의**

$$cost(W,b) = \frac{1}{n}\sum_{i}^{n}\left[y_i - H(x_i)\right]^2$$

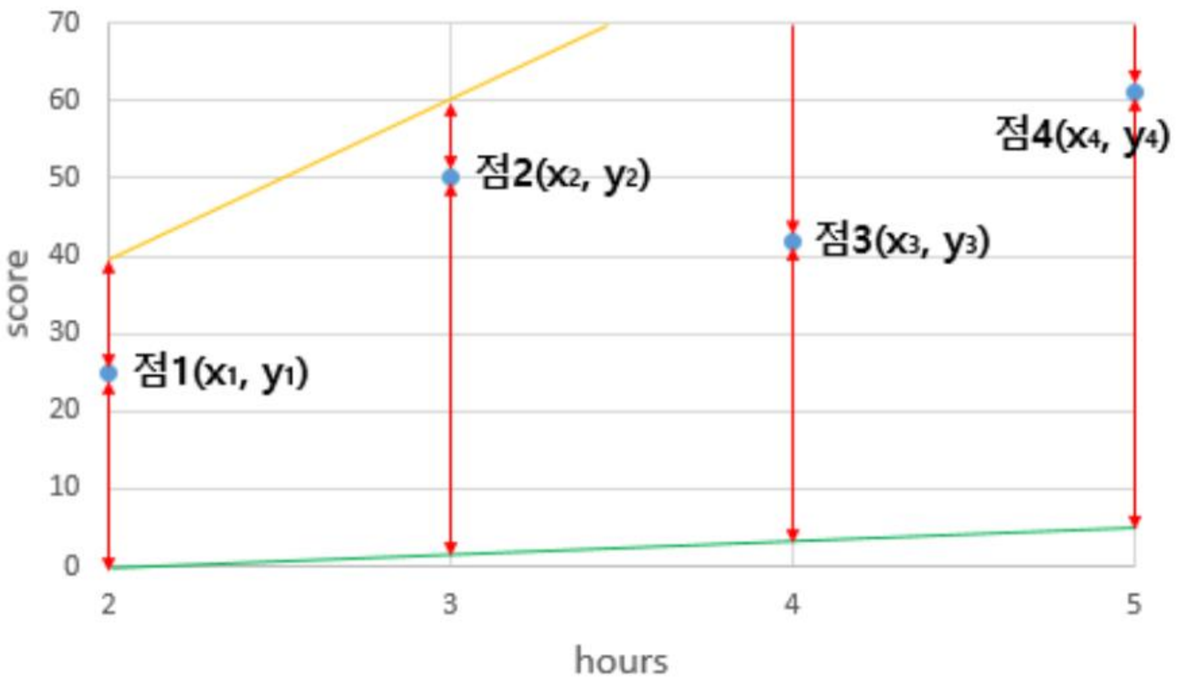
  - 모든 점 들과의 오차가 클수록 평균 제곱 오차는 커지며,
    - **오차가 작아질수록 평균 제곱 오차는 작아짐**

- **평균 제곱 오차**
  - cost(W, b)를 최소가 되게 만드는 W와 b를 구하면
    - **결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 됨**

$$W, b \rightarrow minimize\ cost(W,b)$$

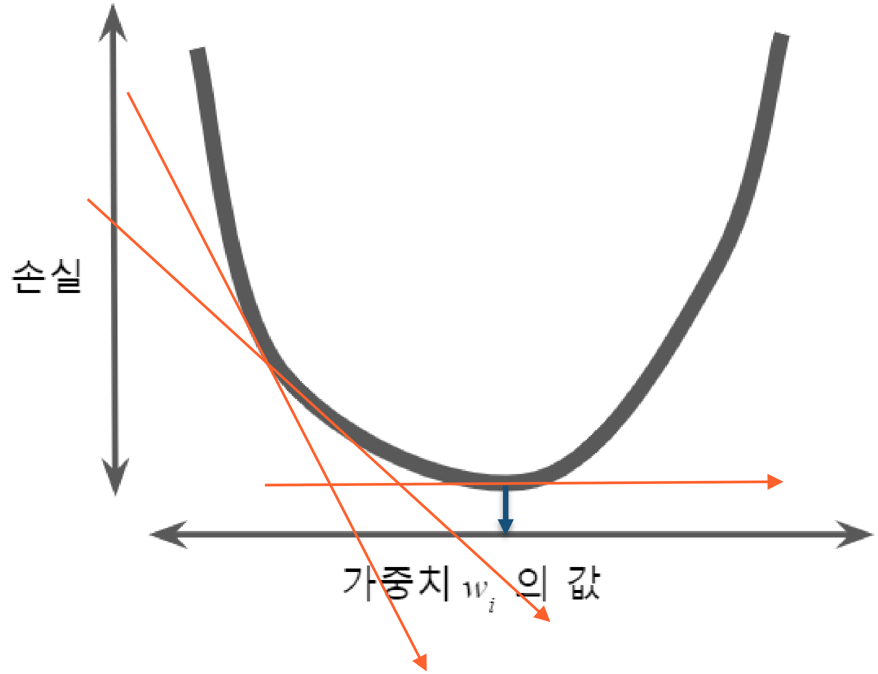
옵티마이저(Optimizer): 최적화 과정

- **머신 러닝에서 학습(training)**
  - 최적화 알고리즘(Optimizer algorithms)
  - 적절한 W와 b를 찾아내는 과정
    - **Gradient Descent(경사 하강법)**
      - **비용 함수(Cost Function)의 값을 최소로 하는 W와 b를 찾는 방법**
      - **경사 따라 내려 오기**



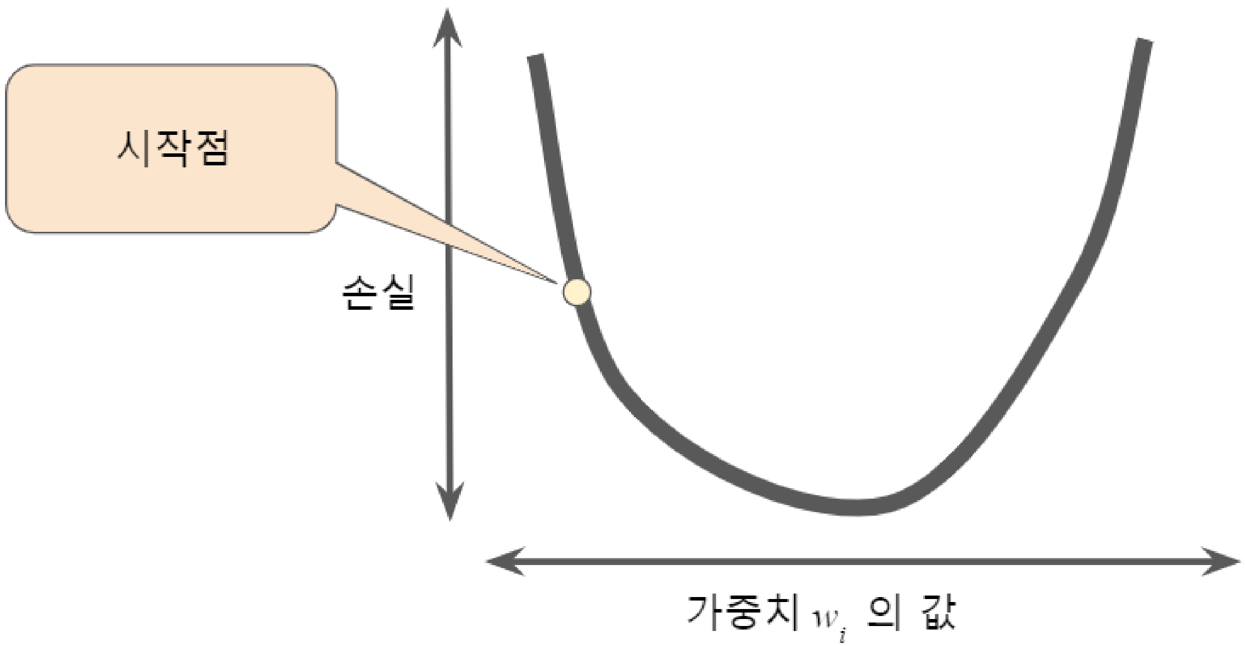
손실과 가중치

- **손실과 가중치 w_i을 대응한 그림**
  - 항상 볼록 함수 모양을 함
    - **도표가 다음과 같이 항상 그릇 모양으로 나타남**
- **볼록 문제에는 기울기가 정확하게 0인 지점인 최소값이 하나만 존재**
  - 이 최소값에서 손실 함수가 수렴
    - **결국 기울기를 구해야 함**



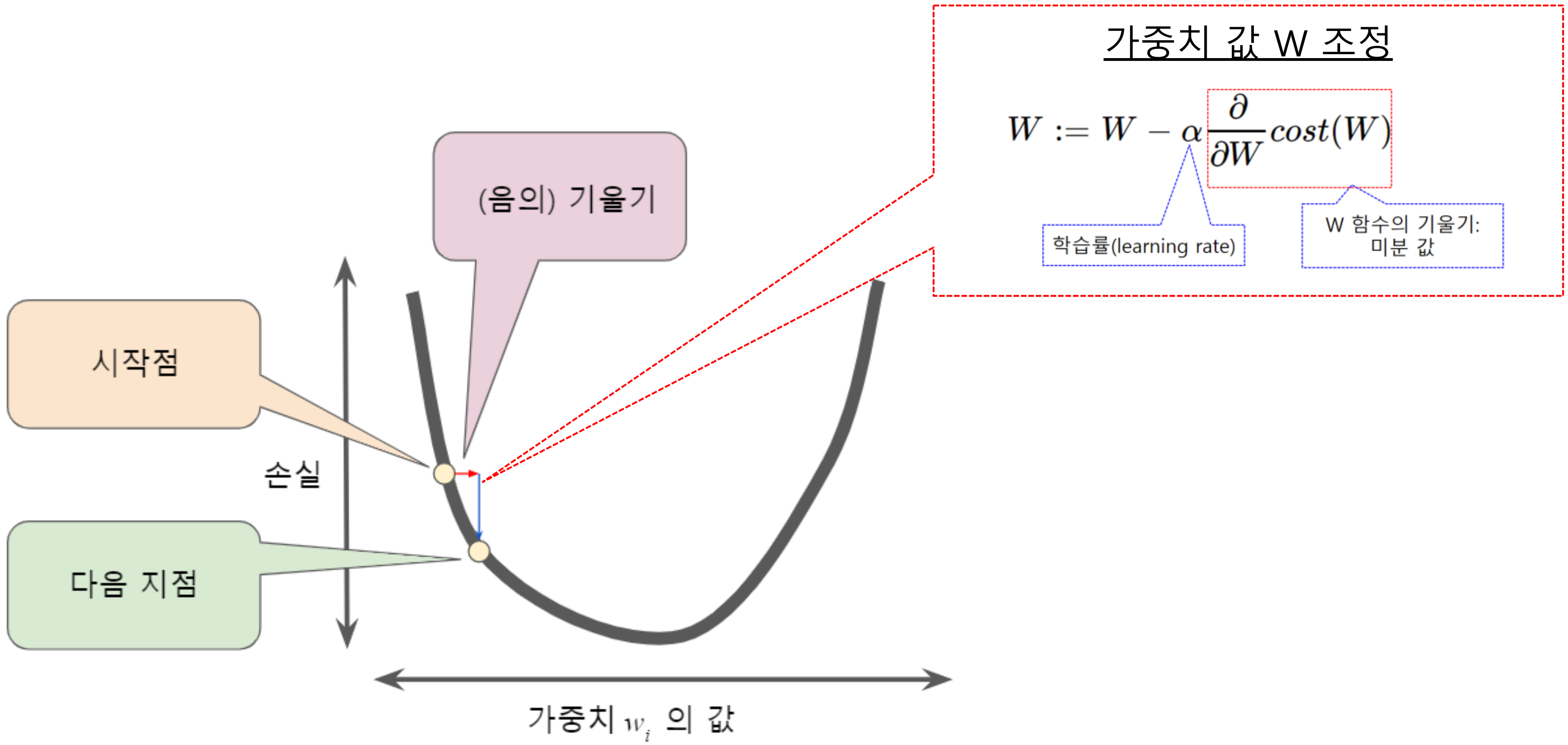
경사하강법

- **경사하강법의 첫 번째 단계**
  - 시작 값(시작점)을 선택
    - **시작점은 별로 중요하지 않음**
    - **따라서 많은 알고리즘에서는 0으로 설정하거나 임의의 값을 선택**
  - 시작점에서 손실 곡선의 기울기를 계산
    - **단일 가중치에 대한 손실의 기울기는 미분 값과 같음**



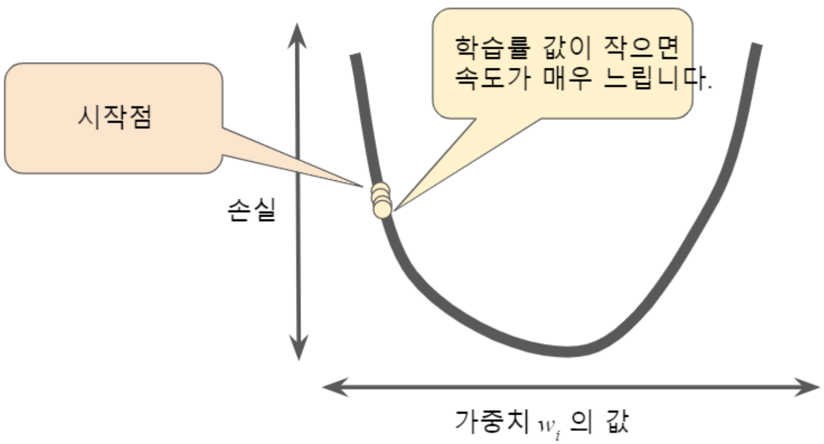
가중치의 조정

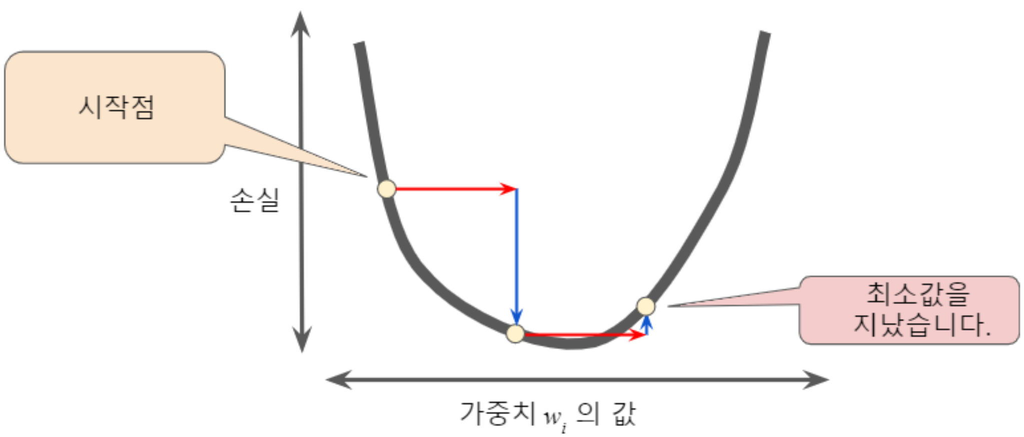
- **기울기가 0인 지점을 찾기 위해**
  - 기울기의 반대 방향으로 이동
    - **현재의 기울기가 음수이면**
      - **다음 가중치 값은 현재의 값보다 크게 조정**



학습률

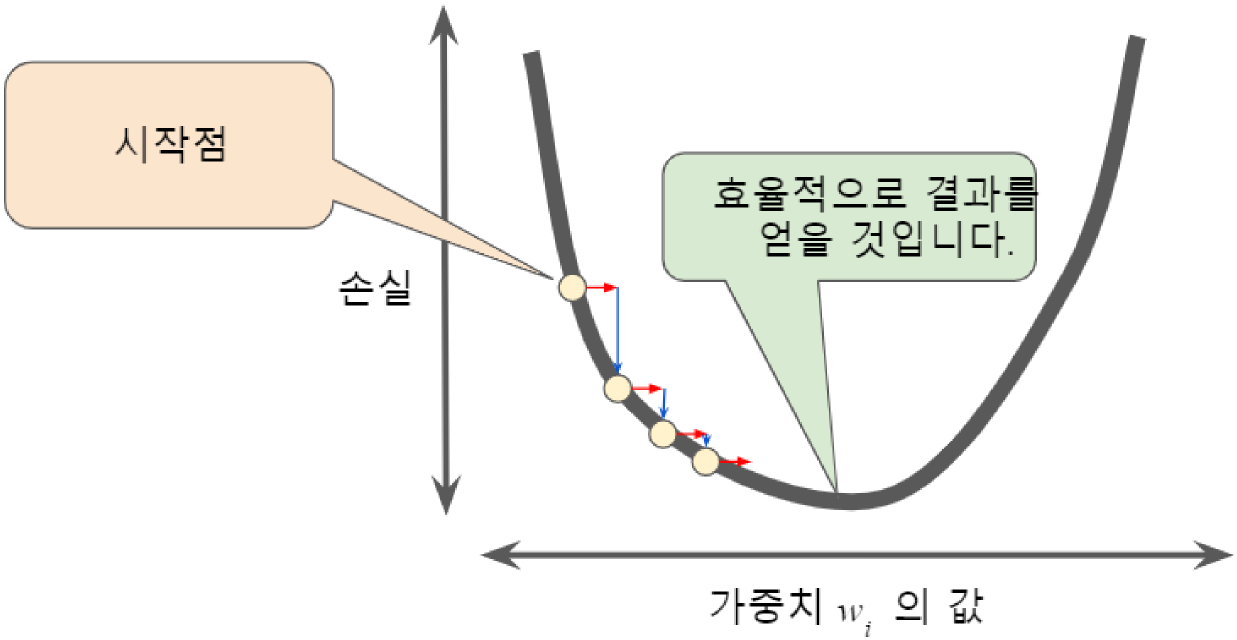
- **다음 가중치 값 결정 방법**
  - 기울기에 학습률(또는 보폭이라 불리는 스칼라)를 곱하여 다음 지점을 결정
    - **예를 들어 기울기가 -2.5이고 학습률이 0.01이면**
      - **w = w – (-2.5 x 0.01) = w + 0.025**
      - **경사하강법 알고리즘은 이전 지점으로부터 0.025 떨어진 지점을 다음 지점으로 결정**
- **학습률의 값**
  - 너무 작게 설정하면 학습 시간이 매우 오래 걸림
    - **반대로 학습률을 너무 크게 설정하면**
      - **다음 지점이 곡선의 최저점을 무질서하게 이탈할 우려가 있음**





적절한 학습률 설정

- **손실 함수의 기울기가 작다면 더 큰 학습률을 시도해 볼 수 있음**
  - 작은 기울기를 보완하고 더 큰 보폭을 만들어 냄

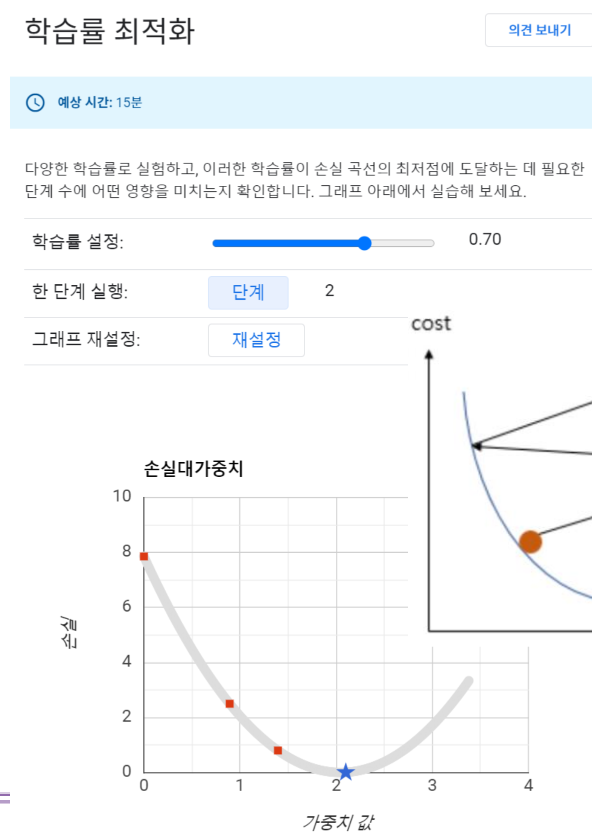


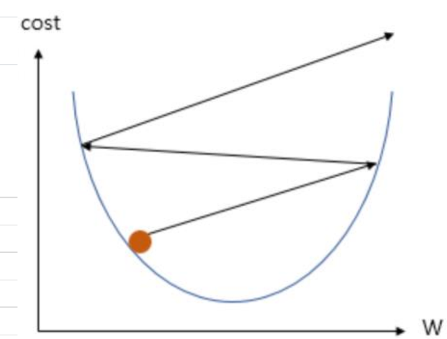
초매개변수와 학습률

- **초매개변수(hyperparameter)**
  - 딥러닝에서 우리가 설정하는 값
    - **모델 학습을 연속적으로 실행하는 중에 개발자 본인에 의해 조작되는 '손잡이'**
  - 예를 들어 학습률은 초매개변수 중 하나
    - **매개변수와 대비되는 개념**

학습률 실험

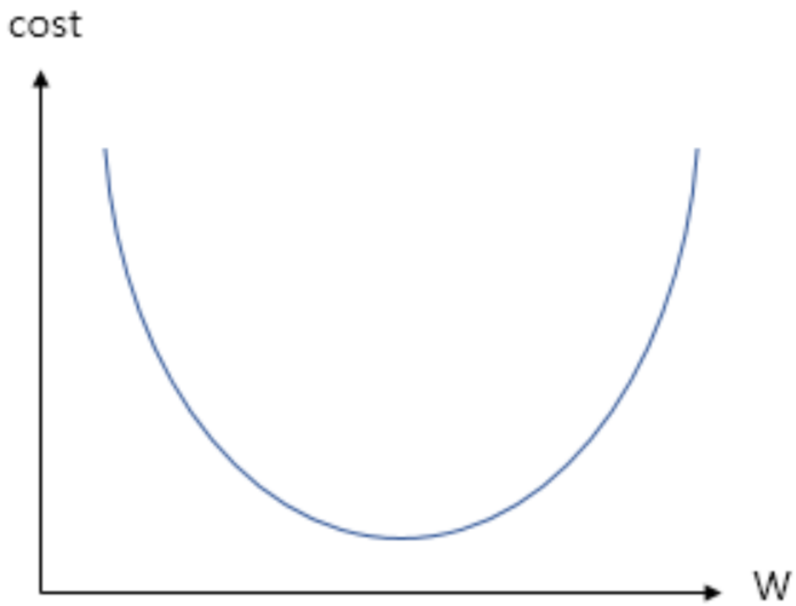
- **다양한 학습률로 실험**
  - 이러한 학습률이 손실 곡선의 최저점에 도달하는 데 필요한 단계 수에 어떤 영향을 미치는지 확인
  - https://developers.google.com/machine-learning/crash-course/fitter/graph?hl=ko
  - 머신러닝 단기집중과정 메뉴
    - **손실 줄이기 | 학습률 최적화**
- **학습률 α**
  - W의 값을 변경할 때
    - **얼마나 크게 변경할지를 결정**
  - 얼마나 큰 폭으로 이동할 지를 결정
    - **학습률 α의 값을 무작정 크게 하면**
      - **W의 값이 발산하는 상황**
    - **학습률 α가 지나치게 낮은 값을 가지면**
      - **학습 속도가 느려지므로 적당한 α의 값을 찾아내는 것도 중요**
- **0.001에서 0.1 정도 사용**

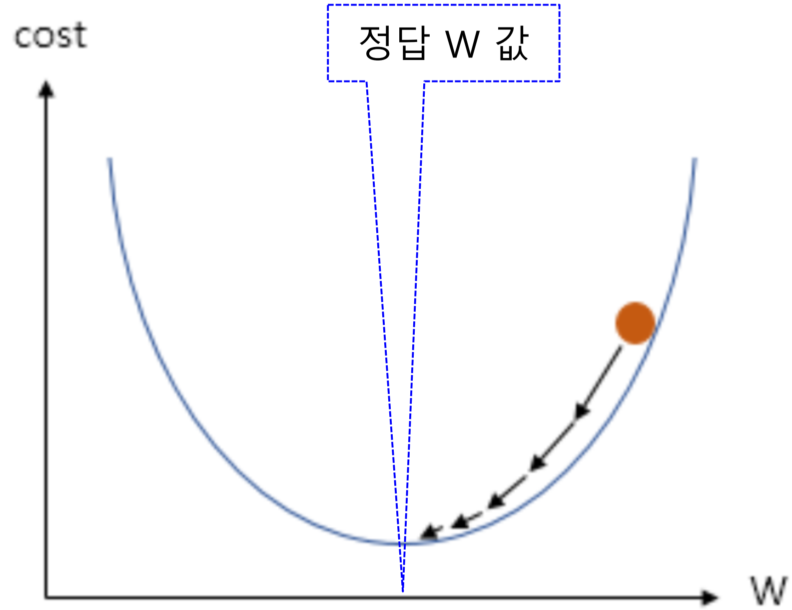




cost가 가장 최소값을 가지게 하는 W를 찾는 일

- **y = Wx라는 가설 H(x)**
  - 비용 함수의 값 cost(W)
    - **설명의 편의를 위해 편향 b가 없이 단순히 가중치 W만을 사용**





비용 함수와 최적의 W 구하기

- **비용 함수(Cost function)**

$$cost(W) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

- **Cost를 최소화하는 W를 구하기 위한 식**
  - 해당 식은 접선의 기울기가 0이 될 때까지 반복

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

학습률(learning rate)

W 함수의 기울기:
미분 값

  - 현재 W에서의 접선의 기울기와 α와 곱한 값을 현재 W에서 빼서 새로운 W의 값으로 다음 손실을 계산
  - 학습률(알파): 기울기가 최소인 다음 w로 가기 위한 비율

계산 과정의 의미

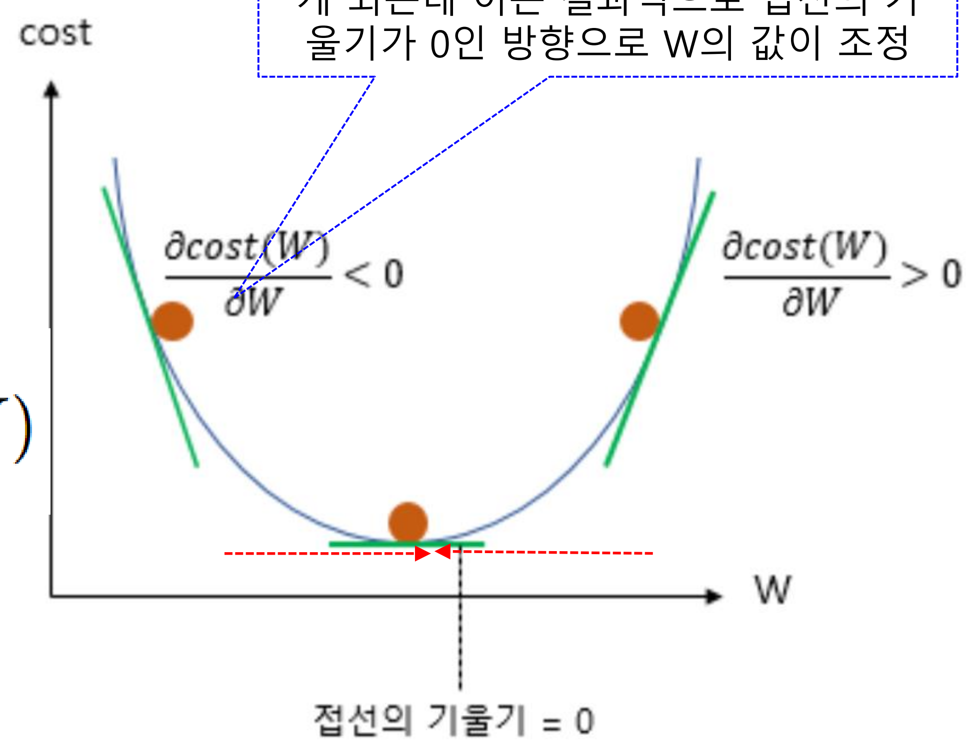
- **현재 W에서 현재 W에서의 접선의 기울기를 빼는 행위의 의미**
  - 접선의 기울기가 음수일 때

$$W := W - \alpha(\text{음수}) = W + \alpha(\text{양수})$$

  - 접선의 기울기가 양수일 때

$$W := W - \alpha(\text{양수})$$

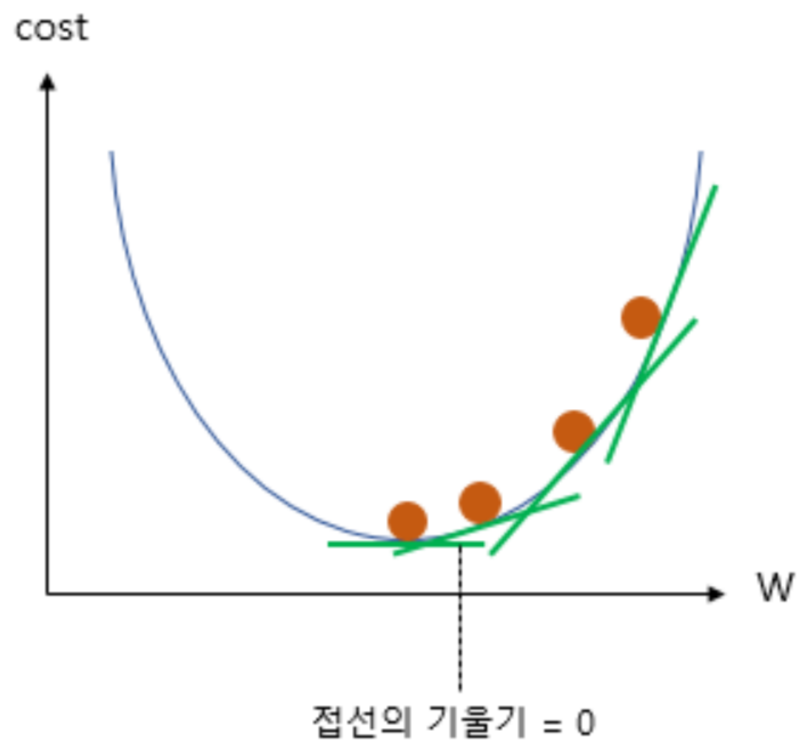
즉, 기울기가 음수면 W의 값이 증가하게 되는데 이는 결과적으로 접선의 기울기가 0인 방향으로 W의 값이 조정



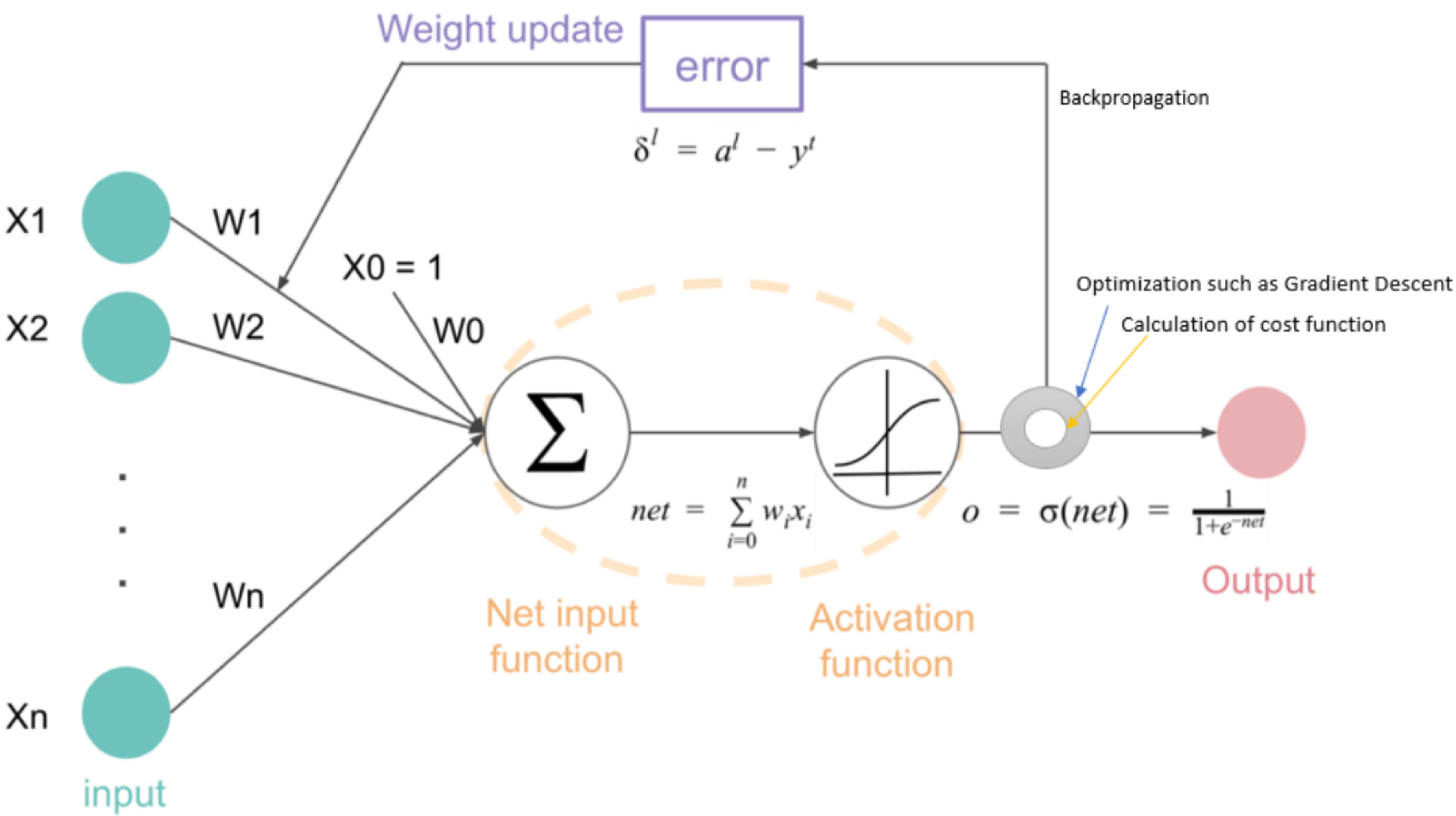
$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

경사 하강법 정리

- **경사 하강법(Gradient Descent)**
  - 내리막 경사 따라 가기
  - 접선의 기울기
    - **맨 아래의 볼록한 부분에서는 결국 접선의 기울기가 0**
  - cost가 최소화가 되는 지점은 접선의 기울기가 0이 되는 지점
    - **또한 미분값이 0이 되는 지점**
  - 경사 하강법의 아이디어
    - **비용 함수(Cost function)를 미분하여 현재 W에서의 접선의 기울기를 구하고**
    - **접선의 기울기가 낮은 방향으로 W의 값을 변경하고 다시 미분하고**
    - **이 과정을 접선의 기울기가 0인 곳을 향해 W의 값을 변경하는 작업을 반복하는 것**

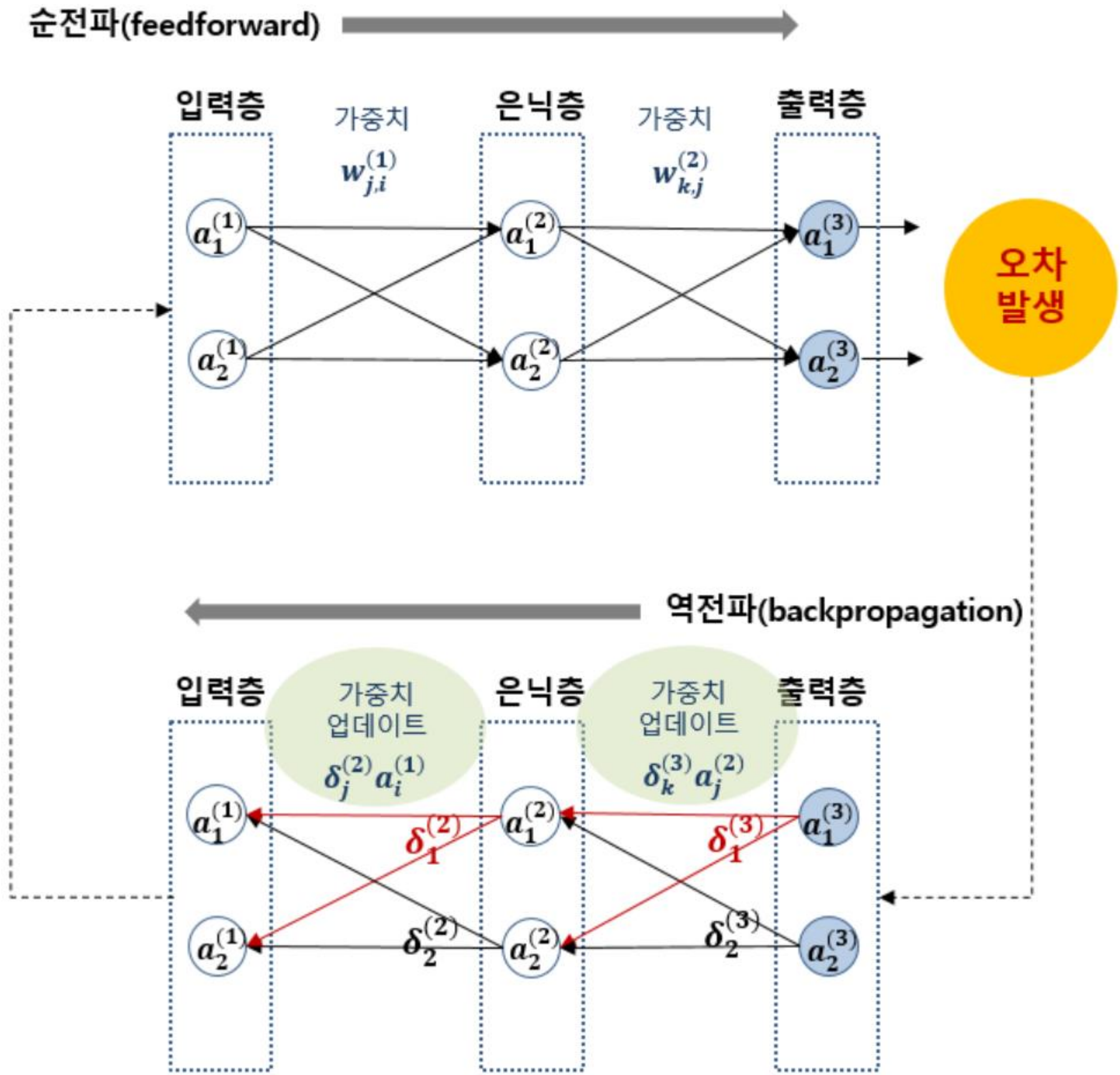


손실 함수를 최소로 하는 W와 b 구하는 과정



오차역전파

- **순전파와 역전파**
  - 순전파, 역전차, 가중치 수정, 순전파 ...를 계속 반복해 나가면서 오차 값이 적어지도록
- **1986년 제프리 힌튼이 적용**
  - 엄청난 처리 속도의 증가



https://blog.naver.com/samsjang/221033626685

선형 회귀
$y = 2x$ 예측

실습

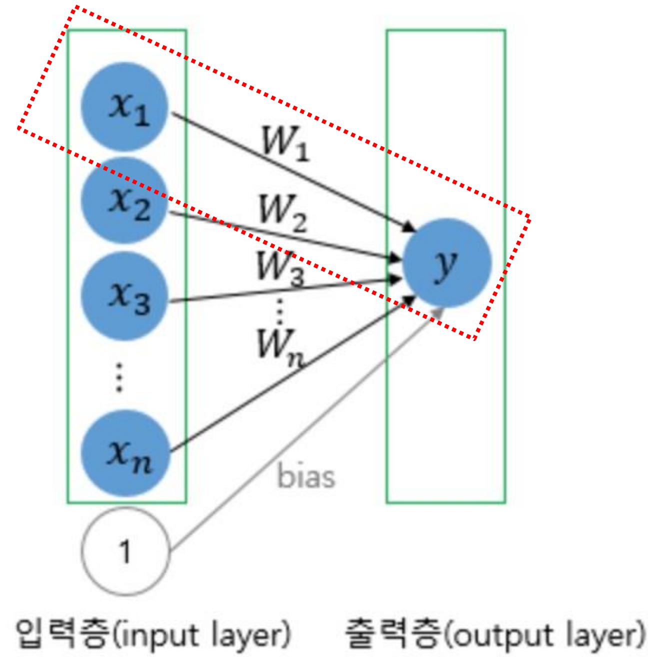
- **21-8-reg-basic.ipynb**

선형 회귀 문제

- y = 2x 에 해당하는 값을 예측
  - 훈련(학습) 데이터
    - x_train = [1, 2, 3, 4]
      y_train = [2, 4, 6, 8]
  - 테스트 데이터
    - x_test = [1.2, 2.3, 3.4, 4.5]
      y_test = [2.4, 4.6, 6.8, 9.0]
  - 예측, 다음 x에 대해 예측되는 y를 출력
    - [3.5, 5, 5.5, 6]

선형 회귀 케라스 구현(1)

- **하나의 Dense 층**
  - 입력은 1차원, 출력도 1차원
- **활성화 함수 linear**
  - 디폴트 값, 입력 뉴런과 가중치로 계산된 결과값이 그대로 출력으로




```
import tensorflow as tf

# ① 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]

# ② 모델 구성(생성)
model = tf.keras.models.Sequential([
    #                    출력, 입력=여러 개 원소의 일차원 배열, 그대로 출력
    tf.keras.layers.Dense(1, input_shape=(1, ), activation='linear')
    #Dense(1, input_dim=1)
])
```

선형 회귀 케라스 구현(2)

- **확률적 경사하강법(Stochastic Gradient Descent)**
  - optimizer='SGD'
    - **경사하강법의 계산량을 줄이기 위해 확률적 방법으로 경사하강법을 사용**
      - **전체를 계산하지 않고 확률적으로 일부 샘플로 계산**
- **mae**
  - 평균 절대 오차(MAE)
    - **모든 예측과 정답과의 오차 합의 평균**
    - **n = 오차의 갯수**
    - **∑ = 합을 나타내는 기호**

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

- **mse**
  - 오차 평균 제곱합(Mean Squared Error, MSE)
    - **모든 예측과 정답과의 오차 제곱 합의 평균**

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

```
# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력 정보를 지정
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse',
              metrics=['mae', 'mse'])
```

선형 회귀 모델 정보

```
# 모델을 표시(시각화)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

선형 회귀 모델 학습(훈련)

- **히스토리 객체**
  - 매 에포크 마다의 훈련 손실값 (loss)
  - 매 에포크 마다의 훈련 정확도 (accuracy)
  - 매 에포크 마다의 검증 손실값 (val_loss)
  - 매 에포크 마다의 검증 정확도 (val_acc)

```
# ④ 생성된 모델로 훈련 데이터 학습
# 훈련과정 정보를 history 객체에 저장
history = model.fit(x_train, y_train, epochs=500)
```

```
Epoch 374/500
1/1 [==============================] - 0s 1ms/step - loss: 4.2576e-04 - mae: 0.0172 - mse: 4.2576e-04
Epoch 375/500
1/1 [==============================] - 0s 1ms/step - loss: 4.2321e-04 - mae: 0.0171 - mse: 4.2321e-04
Epoch 376/500
1/1 [==============================] - 0s 2ms/step - loss: 4.2068e-04 - mae: 0.0171 - mse: 4.2068e-04
Epoch 377/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1817e-04 - mae: 0.0170 - mse: 4.1817e-04
Epoch 378/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1566e-04 - mae: 0.0170 - mse: 4.1566e-04
Epoch 379/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1318e-04 - mae: 0.0169 - mse: 4.1318e-04
```

선형 회귀 모델 성능 평가 및 예측

- ## 성능 평가

```
# ⑤ 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [2.4, 4.6, 6.8, 9.0]

print('손실', model.evaluate(x_test, y_test))
```

```
1/1 [==============================] - 0s 1ms/step - loss: 0.0012 - mae: 0.0313 - mse: 0.0012
손실: [0.0012317538494244218, 0.031307220458984375, 0.0012317538494244218]
```

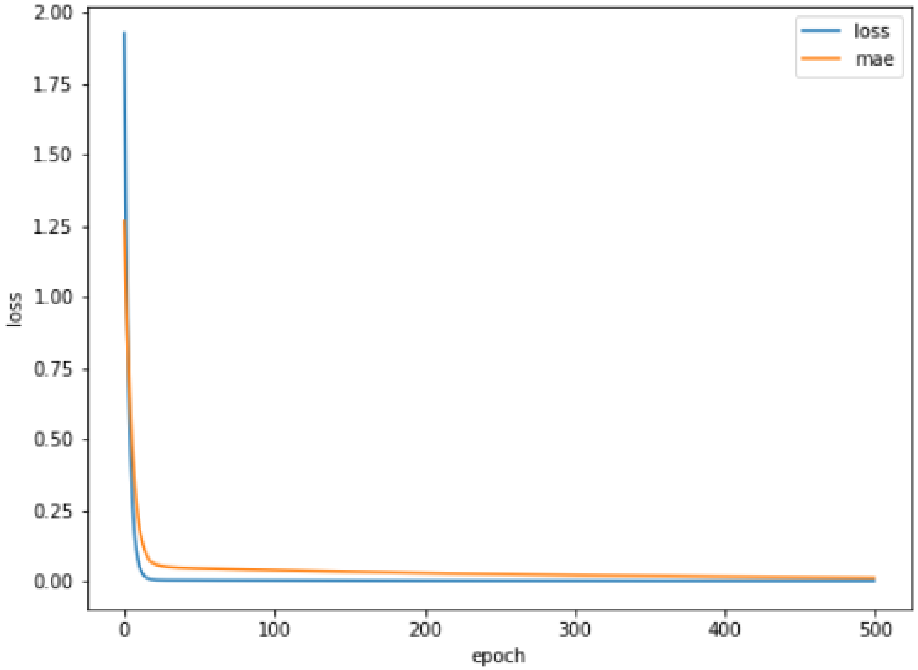
- ## 예측

```
# x = [3.5, 5, 5.5, 6]의 예측
print(model.predict([3.5, 5, 5.5, 6]))

pred = model.predict([3.5, 5, 5.5, 6])
# 예측 값만 1차원으로
print(pred.flatten())
print(pred.squeeze())
```

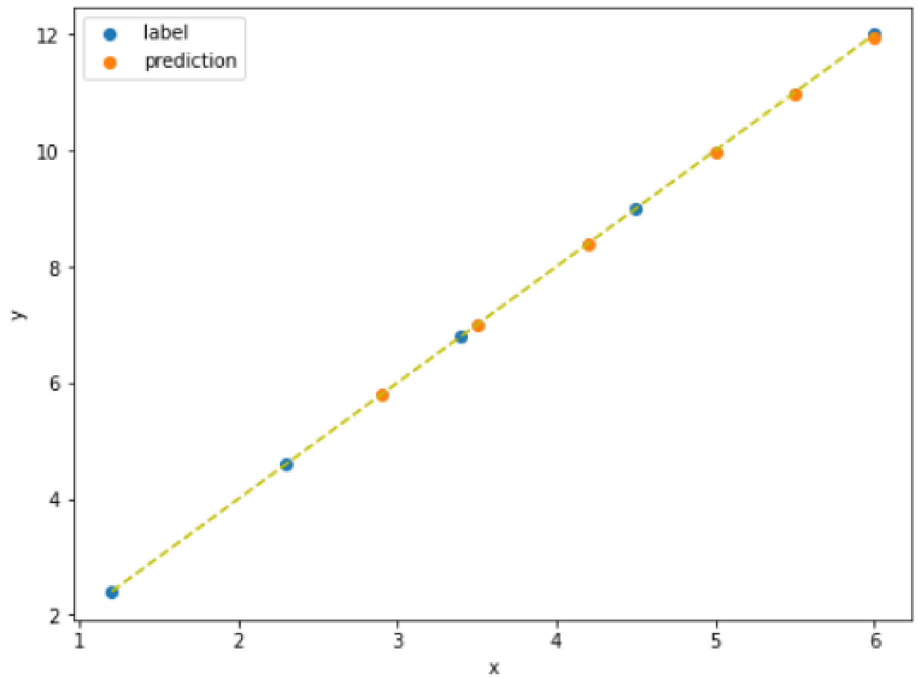
```
[[ 6.9934297]
 [ 9.975829 ]
 [10.969961 ]
 [11.964094 ]]
[ 6.9934297  9.975829  10.969961  11.964094 ]
[ 6.9934297  9.975829  10.969961  11.964094 ]
```

손실과 mae 시각화

```
import matplotlib.pylab as plt

# 그래프 그리기
fig = plt.figure(figsize=(8, 6))

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['mae'], label='mae')
#plt.plot(history.history['mse'], label='mse')

plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('loss')
```



예측 값 시각화




```
import matplotlib.pylab as plt

x_test = [1.2, 2.3, 3.4, 4.5, 6.0]
y_test = [2.4, 4.6, 6.8, 9.0, 12.0]

# 그래프 그리기
fig = plt.figure(figsize=(8, 6))

plt.scatter(x_test, y_test, label='label')
plt.plot(x_test, y_test, 'y--')

x = [2.9, 3.5, 4.2, 5, 5.5, 6]
pred = model.predict(x)
plt.scatter(x, pred.flatten(), label='prediction')

plt.legend(loc='best')
plt.xlabel('x')
plt.ylabel('y')
```

전 코드

- 입출력 층만 존재

```
# 버전 1.x만 가능
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# ① 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]

# ② 모델 구성(생성)
model = Sequential([
    Dense(1, input_shape=(1, ), activation='linear')
    #Dense(1, input_dim=1)
])

# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse',
              metrics=['mae', 'mse'])
# 모델을 표시(시각화)
model.summary()

# ④ 생성된 모델로 훈련 데이터 학습
model.fit(x_train, y_train, epochs=1000)

# ⑤ 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [2.4, 4.6, 6.8, 9.0]
print('정확도:', model.evaluate(x_test, y_test))

print(model.predict([3.5, 5, 5.5, 6]))
```

선형 회귀

$y = 2x + 1$ 예측

다음을 예측해 보세요

- x = [0, 1, 2, 3, 4]
- y = [1, 3, 5, ?, ?]

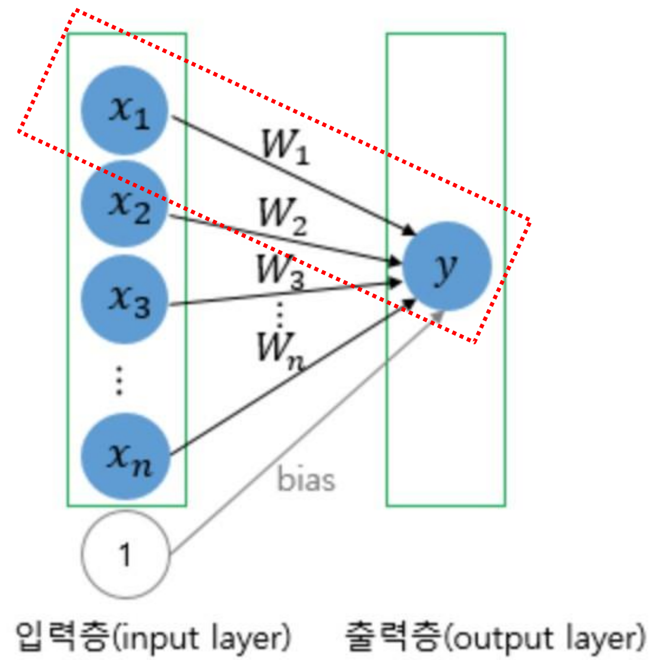
케라스로 예측

- **케라스와 numpy 사용**
- **학습에 3개 데이터**
  - x = [0, 1, 2, 3, 4]
    - **x[:3]**
  - y = [1, 3, 5, ?, ?]
    - **y[:3]**
- **예측**
  - 뒤 2개 데이터 사용
  - x = [0, 1, 2, 3, 4]
    - **x[3:]**
  - y = [1, 3, 5, ?, ?]
    - **y[3:]**

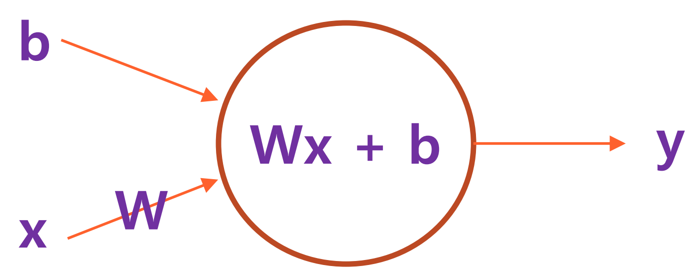
```
import tensorflow as tf
import numpy as np

#훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

#모델의 패라미터를 지정하고 모델 구조를 생성
#최적화 알고리즘: 확률적 경사 하강법(SGD: Stochastic Gradient Descent)
#손실 함수(loss function): 평균제곱오차(MSE: Mean Square Error)
model.compile('SGD', 'mse')

#생성된 모델로 훈련 자료로 입력(x[:3])과 출력(y[:3])을 사용하여 학습
#키워드 매개변수 epoch(에폭): 훈련반복횟수
#키워드 매개변수 verbose: 학습진행사항 표시
model.fit(x[:3], y[:3], epochs=1000, verbose=0)

#테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])

#학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```

가장 간단히 입력층과 출력층 구성

- **y[3:]의 2개 값을 맞추는 인공신경망**
  - 먼저 모델에서 W와 b를 구함
  - 완전연결계층
    - **fully connected or dense layer**
      - **입력 벡터에 가중치 벡터를 내적하고 편향값을 빼주는 연산**




```
import tensorflow as tf
import numpy as np

#훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

#모델의 패라미터를 지정한 후 학습
Model.compile('SGD', 'mse')
Model.fit(x[:3], y[:3], epochs=1000, verbose=0)

print('Targets(정답):', y[3:])
print('Predictions(예측):', model.predict(x[3:]).flatten())
```



케라스로 예측 순서

- ① 케라스 패키지 임포트
  - import tensorflow as tf
  - import numpy as np
- ② 데이터 지정
  - x = numpy.array([0, 1, 2, 3, 4])
  - y = numpy.array([1, 3, 5, 7, 9]) #y = x * 2 + 1
- ③ 인공신경망 모델 구성
  - model = tf.keras.models.Sequential()
  - model.add(tf.keras.layers.Dense(출력수, input_shape=(입력수,)))
- ④ 최적화 방법과 손실 함수 지정해 인공신경망 모델 생성
  - model.compile(' SGD ' , ' mse ')
- ⑤ 생성된 모델로 훈련 데이터 학습
  - model.fit(…)
- ⑥ 성능 평가
  - model.evaluate(…)
- ⑦ 테스트 데이터로 결과 예측
  - model.predict(…)

전 소스

```
import tensorflow as tf
import numpy as np

#훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

#모델의 패라미터를 지정하고 모델 구조를 생성
#최적화 알고리즘: 확률적 경사 하강법(SGD: Stochastic Gradient Descent)
#손실 함수(loss function): 평균제곱오차(MSE: Mean Square Error)
model.compile('SGD', 'mse')

#생성된 모델로 훈련 자료로 입력(x[:2])과 출력(y[:2])을 사용하여 학습
#키워드 매개변수 epoch(에폭): 훈련반복횟수
#키워드 매개변수 verbose: 학습진행사항 표시
model.fit(x[:3], y[:3], epochs=1000, verbose=0)

#테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])

#학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```

보스톤 주택 가격 예측

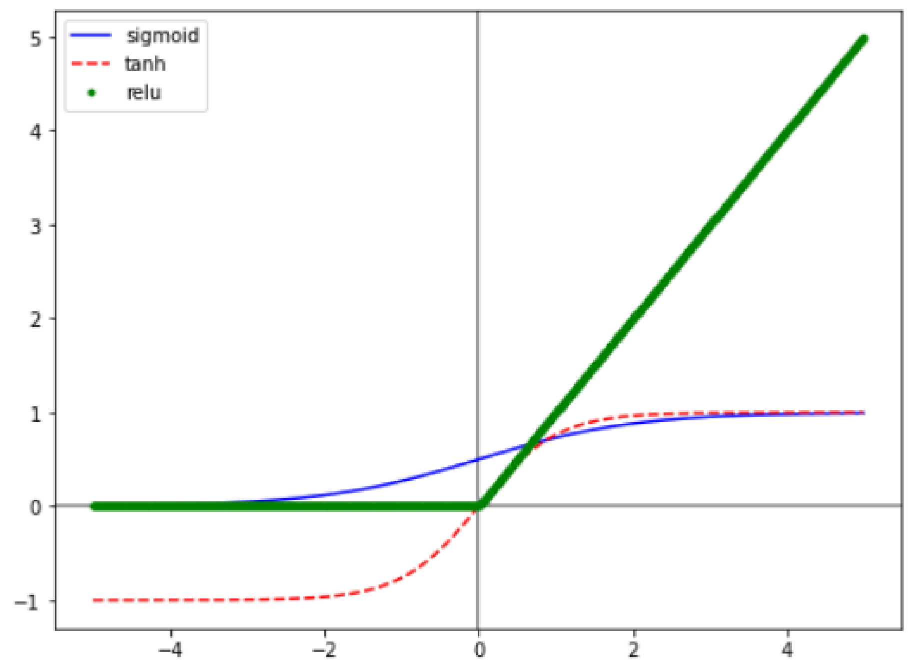
실습

- 21-9-boston_regression.ipynb

주요 활성화 함수

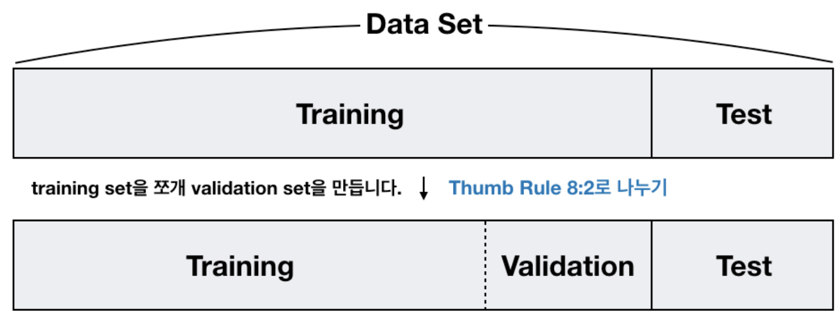
- ReLU
- Sigmoid
- Tanh

```
[22]
# 활성화 함수
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

x = np.arange(-5, 5, 0.01)
sigmoid_x = [sigmoid(z) for z in x]
tanh_x = [math.tanh(z) for z in x]
relu = [0 if z < 0 else z for z in x]

plt.figure(figsize=(8, 6))

plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.plot(x, relu, 'g.', label='relu')
plt.legend()
plt.show()
```



보스톤 주택 가격 예측

- **1978년 보스톤 지역 주택 가격 데이터 셋**
  - 506 개 타운의 주택 가격 중앙 값, 천 달러 단위
  - 범죄율, 방 수, 고속도로 까지 거리 등 13가지 특성, p93
  - 학습 데이터
    - **404개**
  - 테스트 데이터
    - **102개**




```
[17]
# 4.11 데이터 불러오기
from tensorflow.keras.datasets import boston_housing
(train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()

print(train_X.shape, test_X.shape)
print(train_X[0])
print(train_Y[0])
```

```
(404, 13) (102, 13)
[  1.23247   0.        8.14      0.        0.538     6.142    91.7
   3.9769    4.      307.       21.      396.9      18.72   ]
15.2
```

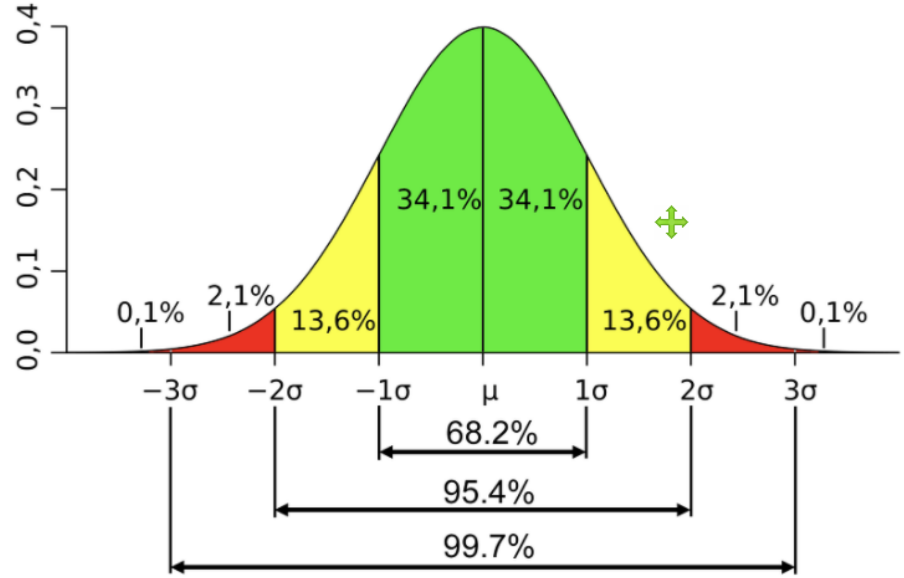
주택가격과 같은 연속형 변수를 예측: 회귀 분석

- **보스턴 주택가격 데이터는 행이 506, 열이 13개**
  - 마지막 변수 주택가격(중위값)을 나머지 13개 변수로 예측하는 문제
  - 특징(속성)
    - **CRIM: 자치시(town) 별 1인당 범죄율**
    - **ZN: 25,000 평방피트를 초과하는 거주지역의 비율**
    - **INDUS: 비소매상업지역이 점유하고 있는 토지의 비율**
    - **CHAS: 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)**
    - **NOX: 10ppm 당 농축 일산화질소**
    - **RM: 주택 1가구당 평균 방의 개수**
    - **AGE: 1940년 이전에 건축된 소유주택의 비율**
    - **DIS: 5개의 보스턴 직업센터까지의 접근성 지수**
    - **RAD: 방사형 도로까지의 접근성 지수**
    - **TAX: 10,000 달러 당 재산세율**
    - **PTRATIO: 자치시(town)별 학생/교사 비율**
    - **B: 1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.**
    - **LSTAT: 모집단의 하위계층의 비율(%)**
  - 정답
    - **MEDV: 주택가격(중앙값) (단위: $1,000)**

정규분포

- **평균=μ(뮤), 표준편차=σ(시그마)**

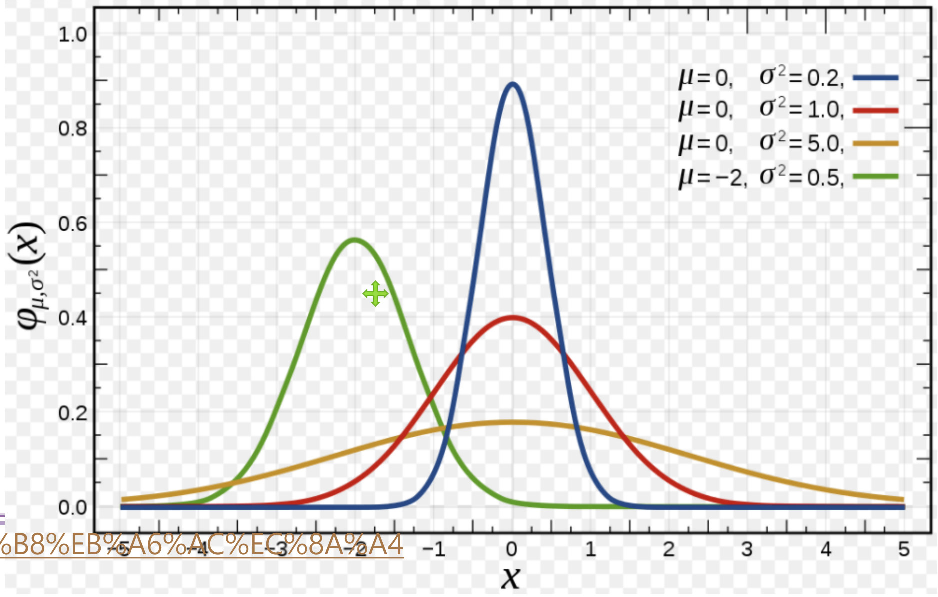
$$X \sim N(\mu, \sigma^2)$$



$$\boxed{X \sim N(\mu, \sigma^2) \rightarrow \quad Z = \frac{X - \mu}{\sigma} \sim N(0, 1)}$$

- **표준화 계산**
  - Z값 계산

$$Z = \frac{X - \mu}{\sigma}$$



https://datascienceschool.net/02%20mathematics/01.01%20%EA%B7%B8%EB%A6%AC%EC%8A%A4%20%EB%AC%B8%EC%9E%90.html

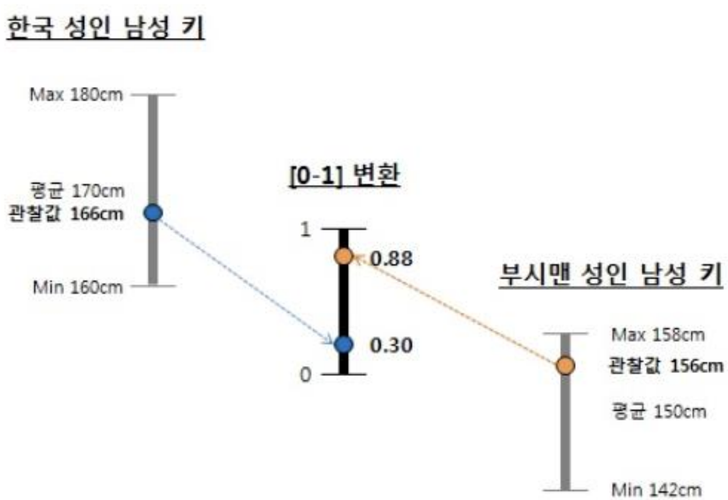
자료의 표준화

- 최소-최대 정규화

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

- Z점수 표준화

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \mathrm{Mean}(X)}{\mathrm{StdDev}(X)}$$



Standardizing A Variable

$$X = \{x_1, x_2, \dots, x_n\}$$

$$X'_i = \frac{X_i - \mathrm{mean}(X)}{\mathrm{sd}(X)}$$

Standardized variable is also called Z-score

자료의 표준화

- **표준화의 필요**
  - 특성의 단위가 다름
    - **비율, 0/1, 양수 등**
  - 표준화(standardization)가 학습 효율에 좋음
- **표준화 방법**
  - 학습 데이터:
    - **(train_X_i - 학습데이터평균) / 학습데이터 표준편차**
      - **정규 분포를 가정**
  - 테스트 데이터
    - **(test_X_i – 학습데이터평균) / 학습데이터 표준편차**
      - **테스트데이터가 정규 분포를 가정할 수 없으므로**

```
# 4.12 데이터 전처리(정규화)
x_mean = train_X.mean(axis=0)
x_std = train_X.std(axis=0)
train_X -= x_mean
train_X /= x_std
test_X -= x_mean
test_X /= x_std

y_mean = train_Y.mean(axis=0)
y_std = train_Y.std(axis=0)
train_Y -= y_mean
train_Y /= y_std
test_Y -= y_mean
test_Y /= y_std

print(train_X[0])
print(train_Y[0])
```

```
[-0.27224633 -0.48361547 -0.43576161 -0.25683275 -0.1652266 -0.1764426
0.81306188 0.1166983 -0.62624905 -0.59517003 1.14850044 0.44807713 0.8252202]
-0.7821526033779157
```

딥러닝 모델

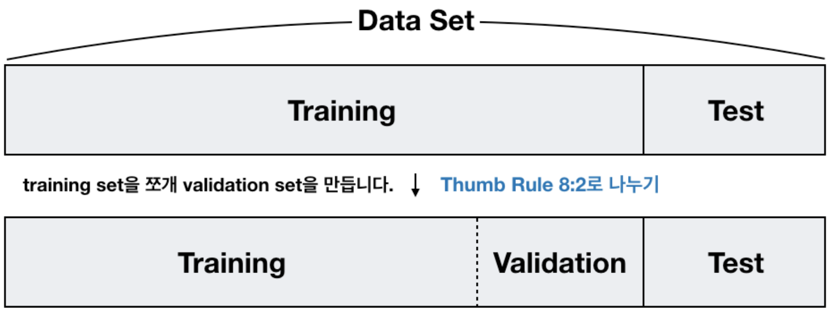
- **총 4개 층**
  - 출력 층은 회귀 모델, 주택 가격이므로 1
- **최적화**
  - 학습률
    - **lr=0.07**
  - 손실 함수
    - **mse**

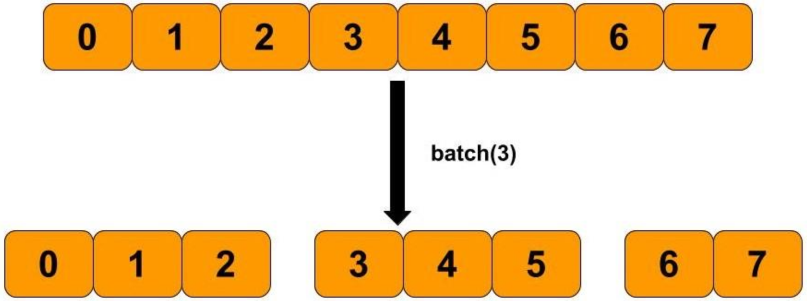
```
[20]
# 4.13 Boston Housing Dataset 회귀 모델 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 52)                728
_________________________________________________________________
dense_11 (Dense)             (None, 39)                2067
_________________________________________________________________
dense_12 (Dense)             (None, 26)                1040
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 27
=================================================================
Total params: 3,862
Trainable params: 3,862
Non-trainable params: 0
_________________________________________________________________
```

학습

- **배치 사이즈와 검증 데이터**
  - 훈련과 검증 분리, 훈련 데이터 404개 중 일부를 검증 데이터로 사용
    - **Validation_split:**
      - **검증 용 데이터의 비율**
    - **만일 .2면**
      - **훈련:검증 == 80%:20% 비중으로 준비**
  - Batch_size
    - **훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위 수**
  - train_size
    - **훈련 데이터 수**

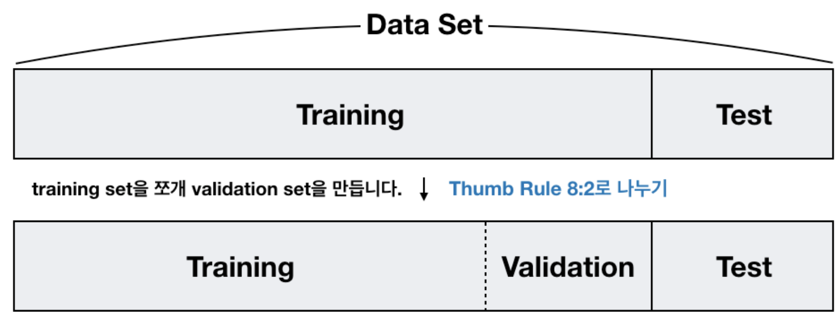





```
# 4.14 회귀 모델 학습
History = model.fit(train_X, train_Y, epochs=25,
                    batch_size=32,
                    validation_split=0.25)
```

학습 수 계산

- **순 학습 샘플 수 = 총 학습 샘플 수 – 검증 샘플 수 : 303**
  - 총 학습 셈플 수 404에서 validation_split=0.25이므로
    - **404 * ¼ = 101개가 검증 자료로**
    - **그러므로 순 학습용 자료: 303**
      - **404 – 101 = 303**



- **한 에폭에서 반복횟수**
  - 가중치와 편향 수정 횟수
    - **int(순 학습 샘플 수 / batch_size) + 1**
    - **int(303 / 32) + 1 = 10**

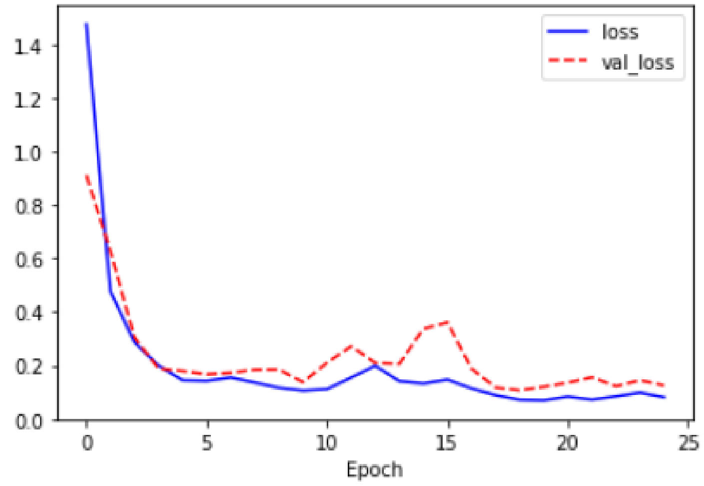
```
# 4.14 회귀 모델 학습
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25)
```

```
Epoch 1/25
10/10 [==============================] - 0s 7ms/step - loss: 0.0713 - val_loss: 0.1365
Epoch 2/25
10/10 [==============================] - 0s 3ms/step - loss: 0.0699 - val_loss: 0.1745
```

훈련과정 시각화와 평가

- **검증 데이터 손실**
  - 일반적으로 loss는 꾸준히 감소
  - val_loss
    - **일반적으로 loss 보다 높음**
    - **항상 감소하지도 않음**
- **평가 결과**
  - 손실 값
    - **작을수록 좋은 결과**
    - **검증 손실이 적을수록 테스트 평가의 손실도 적음**

```
[ ]
# 4.15 회귀 모델 학습 결과 시각화
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



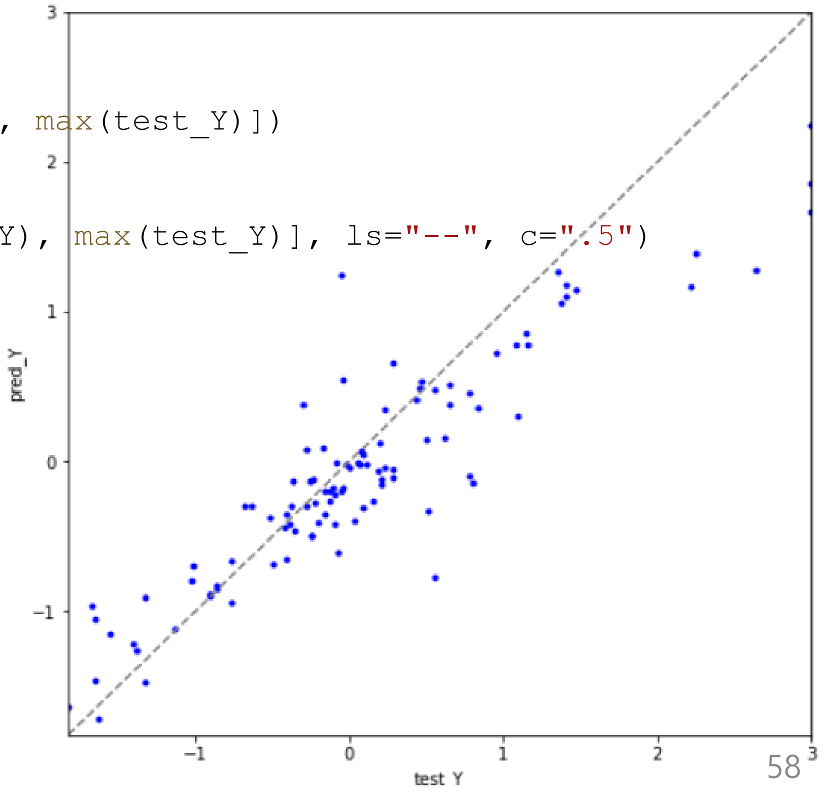

```
[60]
# 4.16 회귀 모델 평가
model.evaluate(test_X, test_Y)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.1999
0.19986297190189362
```

예측 시각화

- **테스트 데이터의 예측과 실제 주택 가격 비교**
  - 각 점들이 점선의 대각선에 있어야 좋은 예측

```python
# 4.17 실제 주택 가격과 예측 주택 가격 시각화
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

# y=x에 해당하는 대각선
plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".5")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```



자동으로 학습 중단

- **검증 손실(val_loss)이 적을수록 테스트 평가의 손실도 적음**
- **검증 데이터에 대한 성적이 좋도록 유도**
  - 과적합에 의해 검증 손실이 증가하면 학습을 중단되도록 지정
  - 함수 callbacks 사용

```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
          callbacks=[tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss')])
```

- **일찍 멈춤 기능**
  - tf.keras.callbacks.EarlyStopping
    - **monitor='val_loss'**
      - **지켜볼 기준 값이 검증 손실**
    - **patience=3**
      - **3회의 epochs를 실행하는 기준 값이 동안 최고 기록을 갱신하지 못하면(더 낮아지지 않으면) 학습을 멈춤**

EarlyStopping

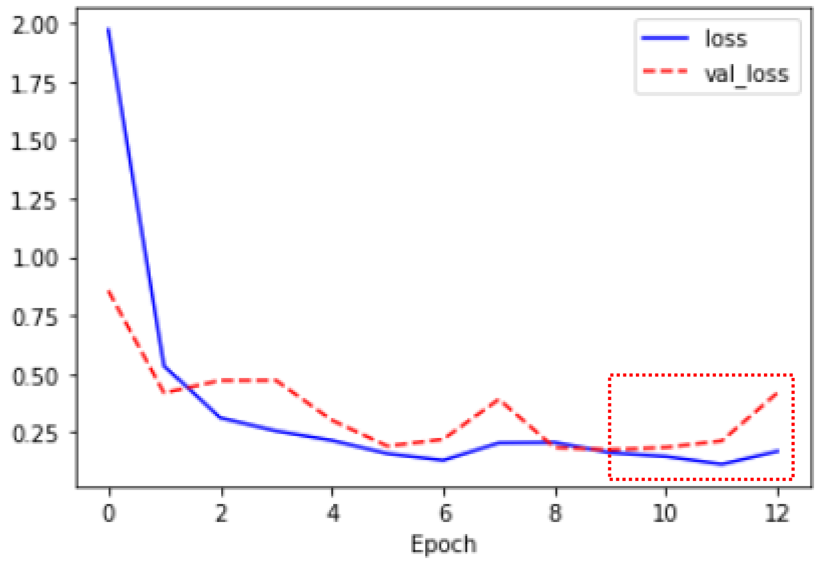
- **주어진 에폭인 25회 내에서**
  - 10 에폭의 기록인 .1748 이후
    - **13 에폭에서도 그 기록을 갱신하지 못했으므로 학습을 중단**
      - **patience = 3회까지 개선이 안되면 종료**

```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
          callbacks=[tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss')])
```

```
Epoch 9/25
10/10 [==============================] - 0s 4ms/step - loss: 0.2058 - val_loss: 0.1839
Epoch 10/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1620 - val_loss: 0.1748
Epoch 11/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1459 - val_loss: 0.1848
Epoch 12/25
10/10 [==============================] - 0s 4ms/step - loss: 0.1124 - val_loss: 0.2126
Epoch 13/25
10/10 [==============================] - 0s 4ms/step - loss: 0.1683 - val_loss: 0.4185
```

자동 중단 시각화

[73]
```
# 4.19 회귀 모델 학습 결과 시각화
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



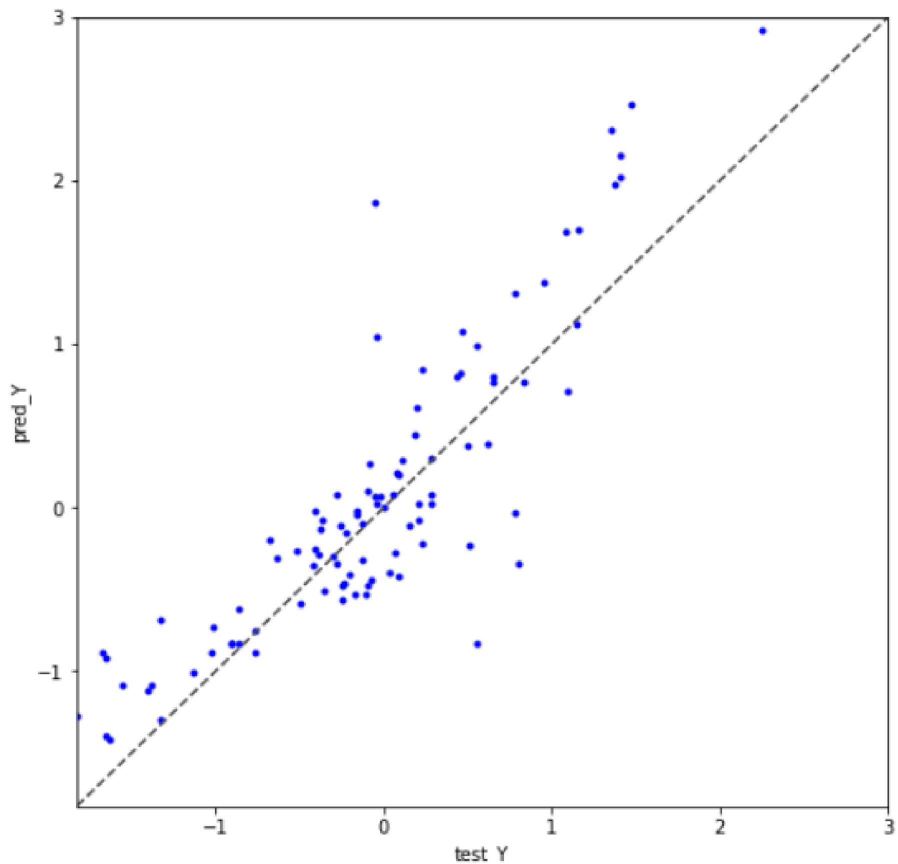
[74]
```
# 4.20 회귀 모델 평가
model.evaluate(test_X, test_Y)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.2536
0.2535804212093353
```

예측 시각화

[75]

```python
# 4.21 실제 주택 가격과 예측 주택 가격 시각화
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```



Dropout

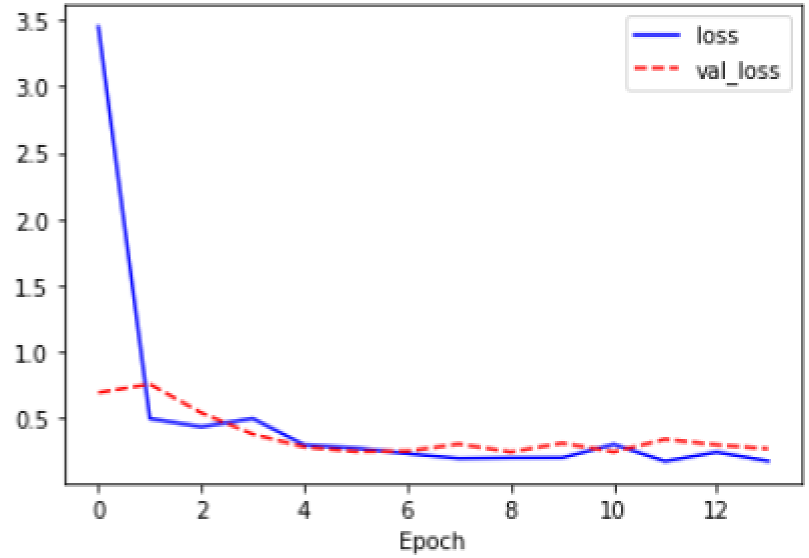
[77]

```
# 모델 재정의 및 학습, dropout 사용
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dropout(.1),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')

history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=5, monitor='val_loss')])
```

```
Epoch 1/25
10/10 [==============================] - 0s 10ms/step - loss: 3.4517 - val_loss: 0.6883
Epoch 2/25
10/10 [==============================] - 0s 4ms/step - loss: 0.4919 - val_loss: 0.7523
Epoch 3/25
10/10 [==============================] - 0s 5ms/step - loss: 0.4304 - val_loss: 0.5356
Epoch 4/25
10/10 [==============================] - 0s 4ms/step - loss: 0.4928 - val_loss: 0.3748
Epoch 5/25
10/10 [==============================] - 0s 4ms/step - loss: 0.2932 - val_loss: 0.2776
Epoch 6/25
10/10 [==============================] - 0s 6ms/step - loss: 0.2694 - val_loss: 0.2432
Epoch 7/25
10/10 [==============================] - 0s 4ms/step - loss: 0.2281 - val_loss: 0.2464
Epoch 8/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1911 - val_loss: 0.2999
Epoch 9/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1959 - val_loss: 0.2399
Epoch 10/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1966 - val_loss: 0.3071
Epoch 11/25
10/10 [==============================] - 0s 5ms/step - loss: 0.2973 - val_loss: 0.2406
Epoch 12/25
10/10 [==============================] - 0s 5ms/step - loss: 0.1697 - val_loss: 0.3360
Epoch 13/25
10/10 [==============================] - 0s 4ms/step - loss: 0.2392 - val_loss: 0.2938
Epoch 14/25
10/10 [==============================] - 0s 4ms/step - loss: 0.1710 - val_loss: 0.2645
```

학습 시각화

```
[78]
# 4.19 회귀 모델 학습 결과 시각화
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```




```
[79]
# 4.20 회귀 모델 평가
model.evaluate(test_X, test_Y)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.2792
0.27918723225593567
```

예측 시각화

[80]

```python
# 4.21 실제 주택 가격과 예측 주택 가격 시각화
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```