Chapter 7
머신러닝 (Machine Learning) 개요; 05) ~ 09

집현전 Season 2
초급반 8조 주혜신, 장영찬, 차경묵

목차

01 로지스틱 회귀 (Logistic Regression)

머신러닝 (Machine Learning)

분류 (Classification)

- **이진 분류 (Binary Classification)**

  주어진 입력에 대해서
  둘 중 하나의 답을 정하는 문제

- **다중 클래스 분류 (Multi-Class Classification)**

  주어진 입력에 대해서
  두 개 이상의 정해진 선택지 중에서
  답을 정하는 문제

- **다중 레이블 분류 (Multi-Label Classification)**

회귀 (Regression)

분리된(비연속적인) 답이 결과가 아니라
연속된 값을 결과로 가짐
(예) 시계열 데이터를 이용한 주가 예측,
생산량 예측, 지수 예측 등

- **단순 선형 회귀**
- **다중 선형 회귀**
- **로지스틱 회귀**

01 로지스틱 회귀 (Logistic Regression)

- 둘 중 하나를 결정하는 이진 분류(binary classification)의 대표적인 알고리즘
  - 이름은 회귀이지만 분류 모델
  - 합격과 불합격, 스팸 메일과 정상 메일 등

01 로지스틱 회귀 (Logistic Regression)

- 합격/불합격 커트라인이 공개되지 않은 경우,
  학생들의 시험 성적에 따라서 합격, 불합격 여부를 판정하는 모델 만들기

데이터

시험점수 (x)	합/불 결과 (y)
45	불합격 (0)
50	불합격 (0)
55	불합격 (0)
60	합격 (1)
65	합격 (1)
70	합격 (1)

01 로지스틱 회귀 (Logistic Regression)

- 합격/불합격 커트라인이 공개되지 않은 경우,
  학생들의 시험 성적에 따라서 합격, 불합격 여부를 판정하는 모델 만들기

데이터

시험점수 (x)	합/불 결과 (y)
45	불합격 (0)
50	불합격 (0)
55	불합격 (0)
60	합격 (1)
65	합격 (1)
70	합격 (1)

실제값 y가 0 또는 1이라는 두 가지 값을 가짐
예측값도 0과 1 사이의 값을 가지도록 하는 것이 보편적

최종 예측값이 0.5 보다 작으면 0으로 예측한 것으로,
최종 예측값이 0.5 보다 크면 1로 예측한 것으로 판단

01 로지스틱 회귀 (Logistic Regression)

- 합격/불합격 커트라인이 공개되지 않은 경우,
  학생들의 시험 성적에 따라서 합격, 불합격 여부를 판정하는 모델 만들기

데이터

시험점수 (x)	합/불 결과 (y)
45	불합격 (0)
50	불합격 (0)
55	불합격 (0)
60	합격 (1)
65	합격 (1)
70	합격 (1)

실제값 y가 0 또는 1이라는 두 가지 값을 가짐
예측값도 0과 1 사이의 값을 가지도록 하는 것이 보편적

최종 예측값이 0.5 보다 작으면 0으로 예측한 것으로,
최종 예측값이 0.5 보다 크면 1로 예측한 것으로 판단

분류 문제를 풀 때 선형 회귀는 적합하지 않음!

01 로지스틱 회귀 (Logistic Regression)

- 합격/불합격 커트라인이 공개되지 않은 경우,
  학생들의 시험 성적에 따라서 합격, 불합격 여부를 판정하는 모델 만들기

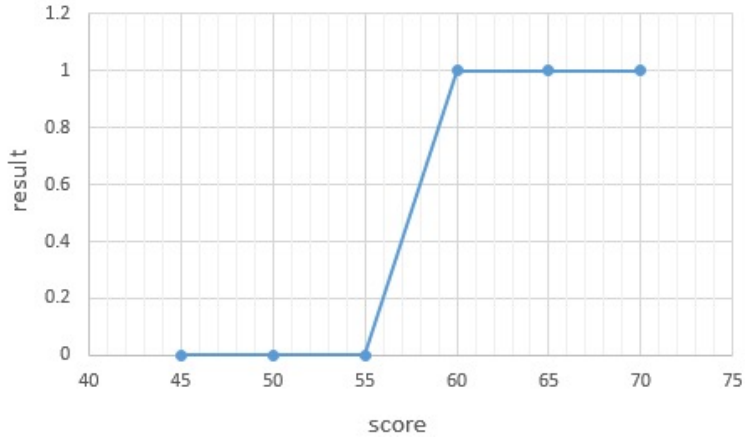
데이터

시험점수 (x)	합/불 결과 (y)
45	불합격 (0)
50	불합격 (0)
55	불합격 (0)
60	합격 (1)
65	합격 (1)
70	합격 (1)

실제값 y가 0 또는 1이라는 두 가지 값을 가짐
예측값도 0과 1 사이의 값을 가지도록 하는 것이 보편적

최종 예측값이 0.5 보다 작으면 0으로 예측한 것으로,
최종 예측값이 0.5 보다 크면 1로 예측한 것으로 판단

분류 문제를 풀 때 선형 회귀는 적합하지 않음!



01 로지스틱 회귀 (Logistic Regression)

0과 1 사이의 값을 가지면서, S자 형태로 그려지는 함수가 있을까?

01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

자연상수 e(2.718281…)

가중치(weight)

편향(bias)

01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

이때, 주어진 데이터에 가장 적합한 가중치(W)와 편향(bias)를 구해야 함!

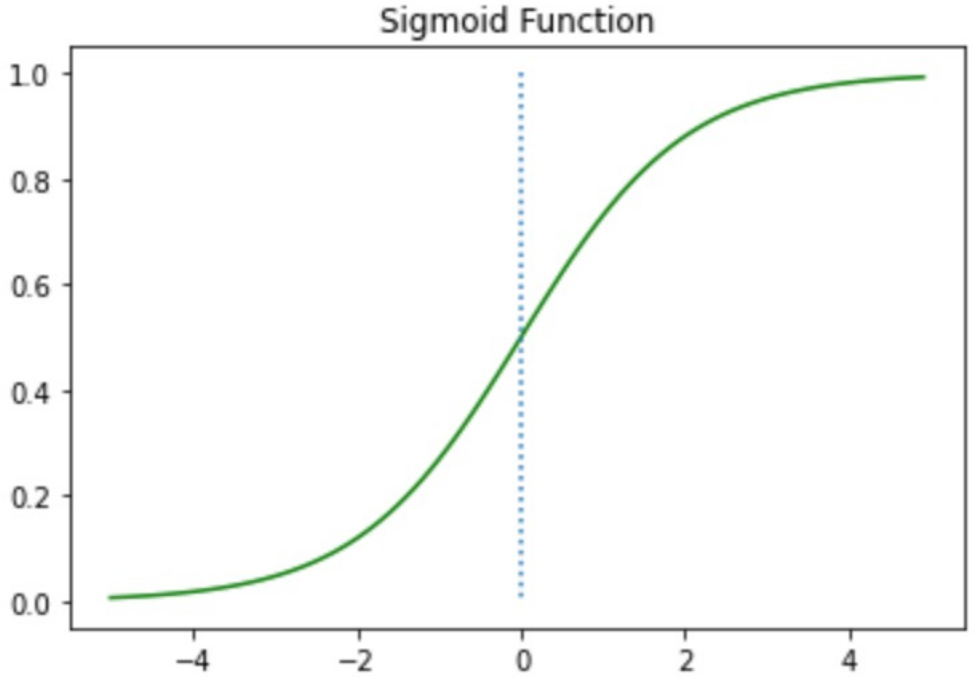
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

이때, 주어진 데이터에 가장 적합한 가중치(W)와 편향(bias)를 구해야 함!

```
[2]  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()
```



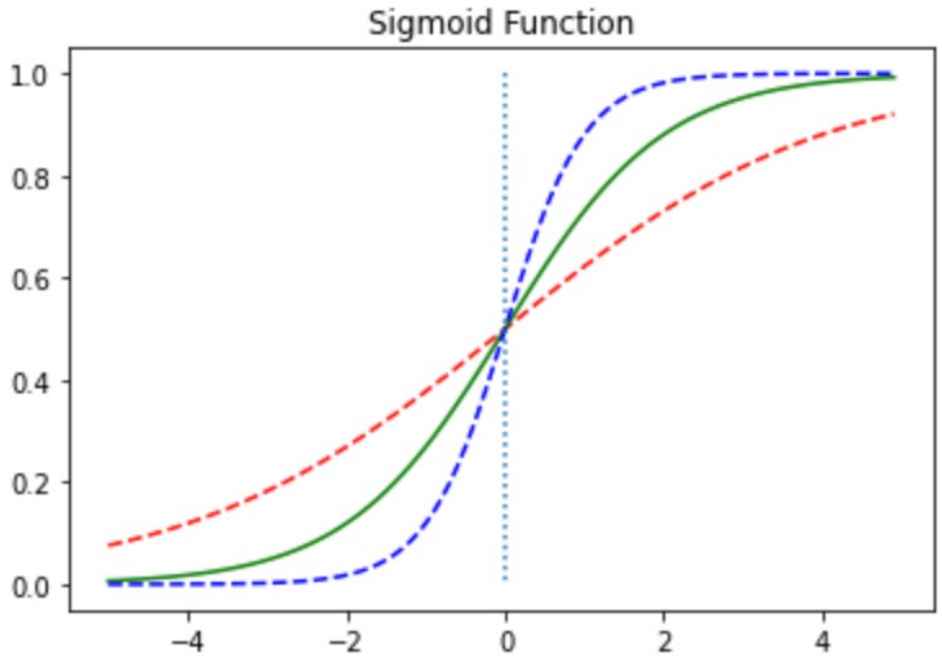
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

W : 그래프의 경사도를 결정

```
[3]  def sigmoid(x):
         return 1/(1+np.exp(-x))
     x = np.arange(-5.0, 5.0, 0.1)
     y1 = sigmoid(0.5*x)
     y2 = sigmoid(x)
     y3 = sigmoid(2*x)

     plt.plot(x, y1, 'r', linestyle='--') # W의 값이 0.5일때
     plt.plot(x, y2, 'g') # W의 값이 1일때
     plt.plot(x, y3, 'b', linestyle='--') # W의 값이 2일때
     plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
     plt.title('Sigmoid Function')
     plt.show()
```



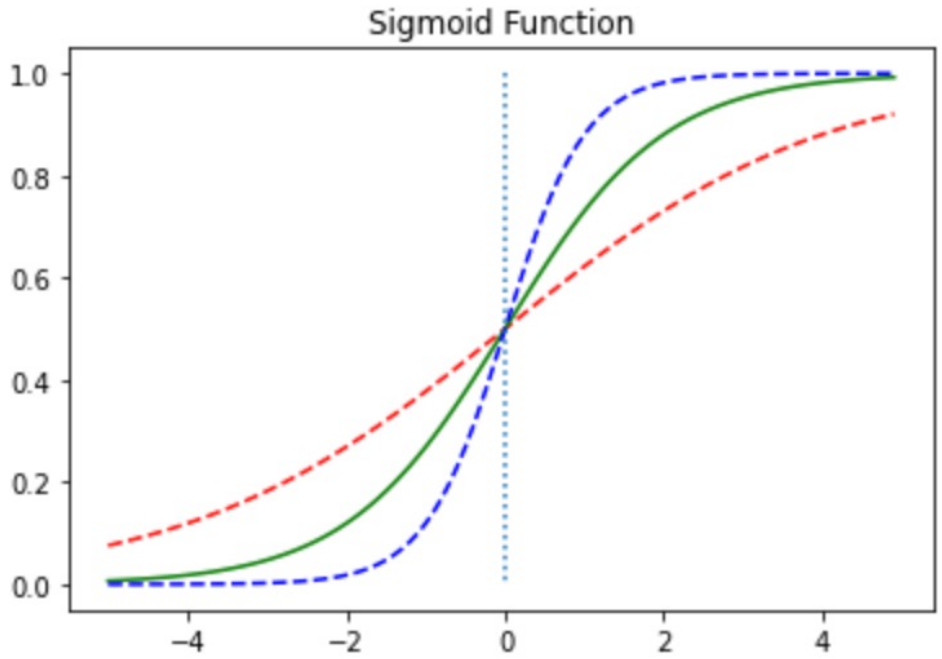
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

W : 그래프의 경사도를 결정

```
[3]  def sigmoid(x):
         return 1/(1+np.exp(-x))
     x = np.arange(-5.0, 5.0, 0.1)
     y1 = sigmoid(0.5*x)
     y2 = sigmoid(x)
     y3 = sigmoid(2*x)

     plt.plot(x, y1, 'r', linestyle='--') # W의 값이 0.5일때
     plt.plot(x, y2, 'g') # W의 값이 1일때
     plt.plot(x, y3, 'b', linestyle='--') # W의 값이 2일때
     plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
     plt.title('Sigmoid Function')
     plt.show()
```



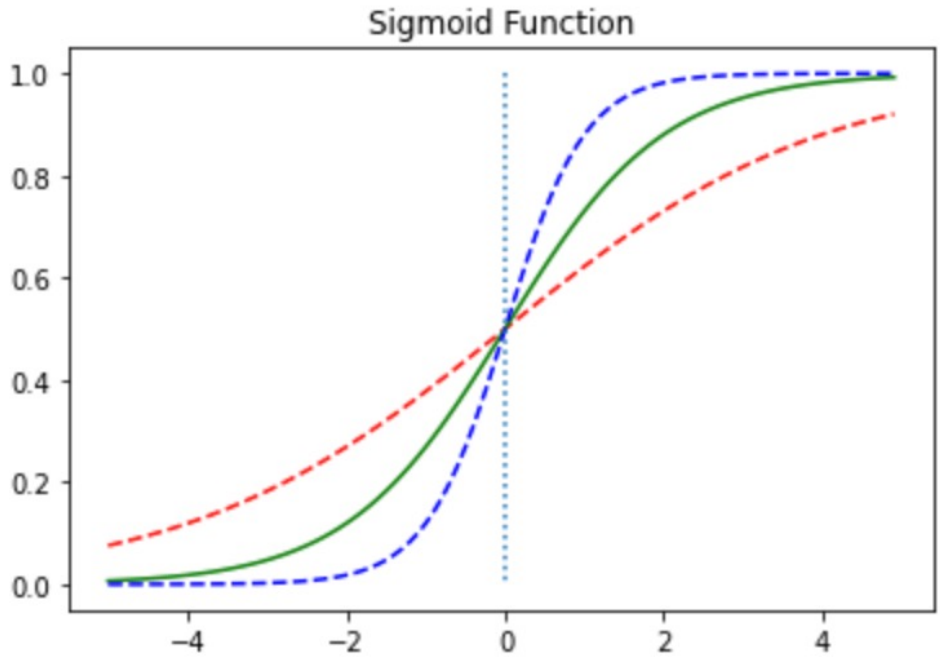
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

W : 그래프의 경사도를 결정

```
[3]  def sigmoid(x):
         return 1/(1+np.exp(-x))
     x = np.arange(-5.0, 5.0, 0.1)
     y1 = sigmoid(0.5*x)
     y2 = sigmoid(x)
     y3 = sigmoid(2*x)

     plt.plot(x, y1, 'r', linestyle='--') # W의 값이 0.5일때
     plt.plot(x, y2, 'g') # W의 값이 1일때
     plt.plot(x, y3, 'b', linestyle='--') # W의 값이 2일때
     plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
     plt.title('Sigmoid Function')
     plt.show()
```



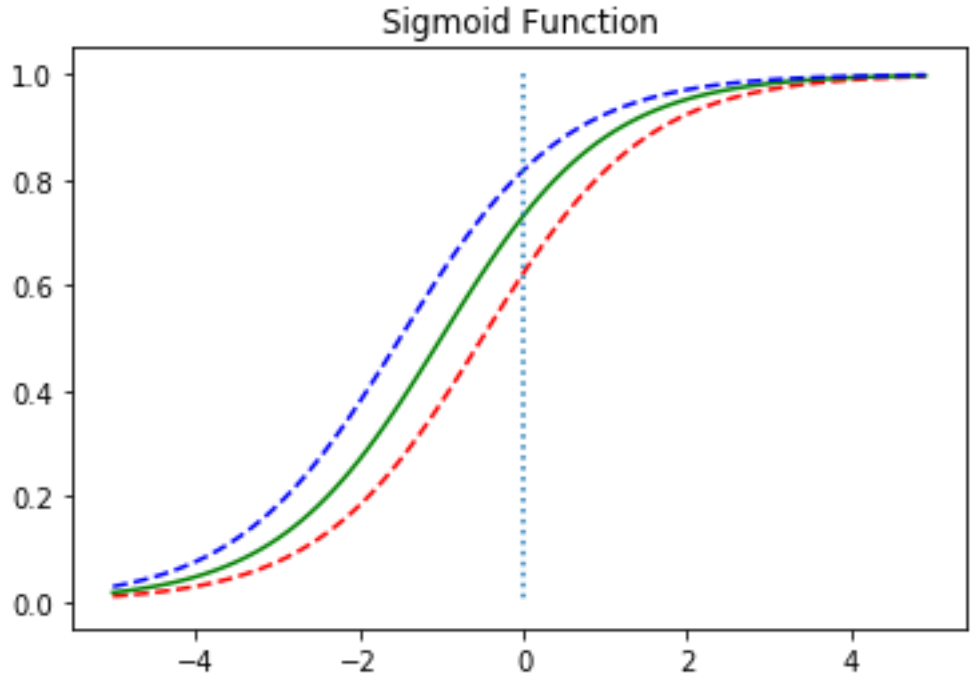
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

b : 그래프를 이동

```
[4]  def sigmoid(x):
         return 1/(1+np.exp(-x))
     x = np.arange(-5.0, 5.0, 0.1)
     y1 = sigmoid(x+0.5)
     y2 = sigmoid(x+1)
     y3 = sigmoid(x+1.5)

     plt.plot(x, y1, 'r', linestyle='--') # x + 0.5
     plt.plot(x, y2, 'g') # x + 1
     plt.plot(x, y3, 'b', linestyle='--') # x + 1.5
     plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
     plt.title('Sigmoid Function')
     plt.show()
```



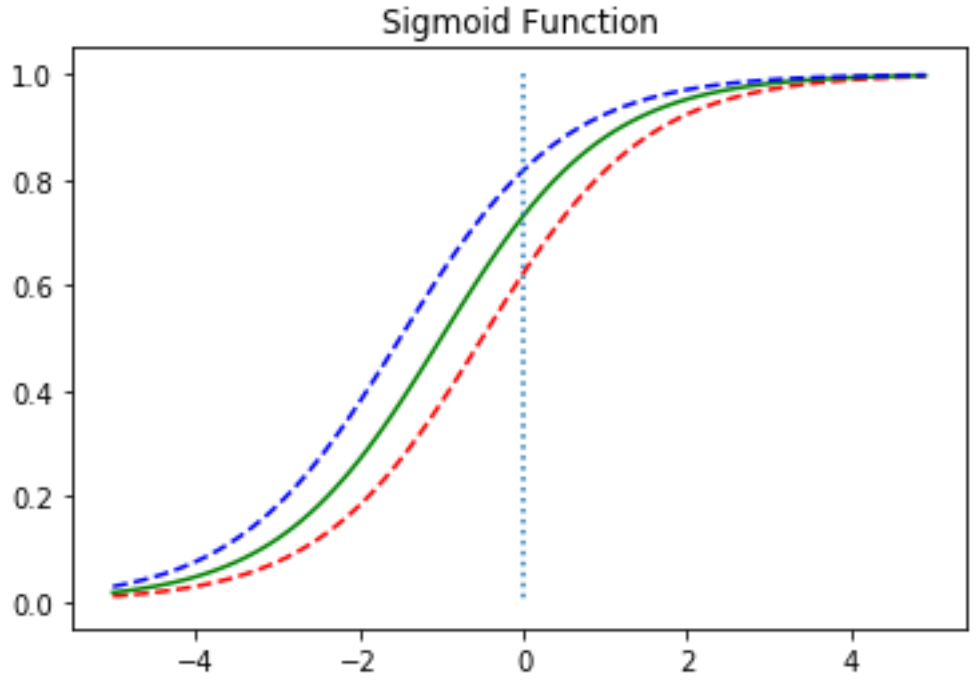
01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

b : 그래프를 이동

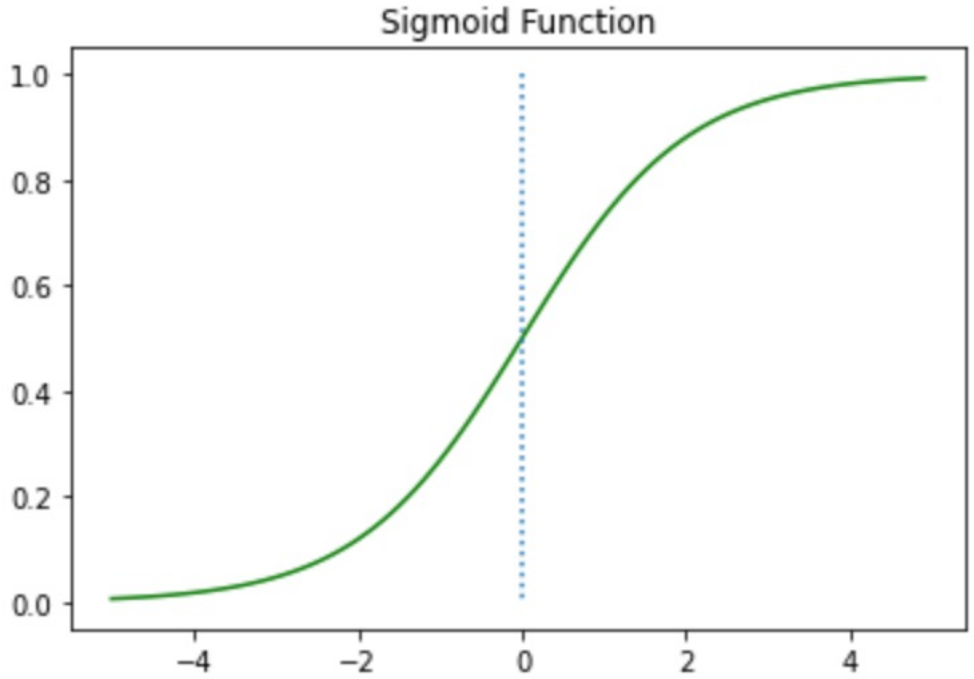
```
[4]  def sigmoid(x):
         return 1/(1+np.exp(-x))
     x = np.arange(-5.0, 5.0, 0.1)
     y1 = sigmoid(x+0.5)
     y2 = sigmoid(x+1)
     y3 = sigmoid(x+1.5)

     plt.plot(x, y1, 'r', linestyle='--') # x + 0.5
     plt.plot(x, y2, 'g') # x + 1
     plt.plot(x, y3, 'b', linestyle='--') # x + 1.5
     plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
     plt.title('Sigmoid Function')
     plt.show()
```



01 로지스틱 회귀 (Logistic Regression)

시그모이드 함수 (Sigmoid Function)

$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$



- 입력값이 커지면 1에 수렴, 입력값이 작아지면 0에 수렴함
- 0부터의 1까지의 값을 가짐
- 출력값이 0.5 이상이면 1(True), 0.5이하면 0(False)로 만들면 이진 분류 문제로 사용할 수 있음

01 로지스틱 회귀 (Logistic Regression)

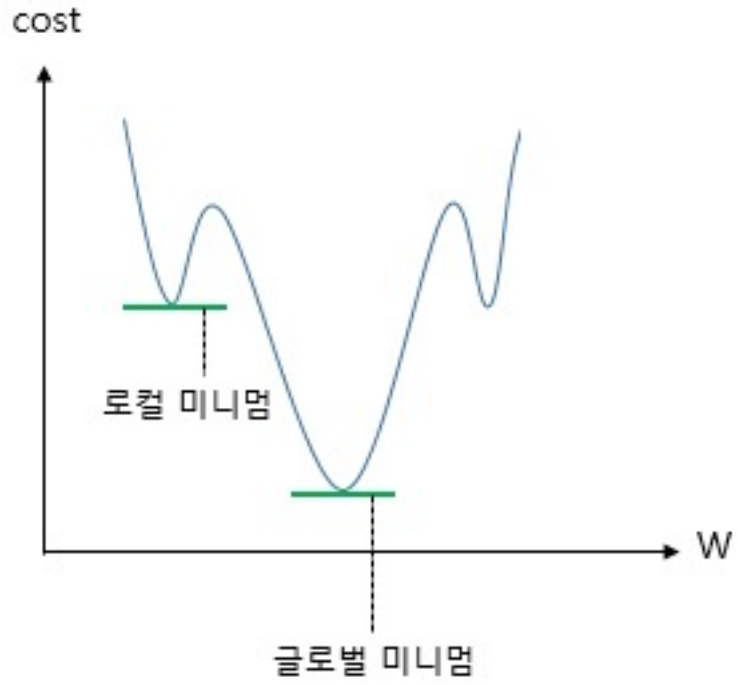
비용 함수 (Cost Function)

로지스틱 회귀는 비용 함수로 MSE(평균 제곱 오차)를 사용하지 않음

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

로지스틱 회귀는 비용 함수로 MSE(평균 제곱 오차)를 사용하지 않음



전체(global) 함수에 걸쳐 최소값인 글로벌 미니멈이 아닌,
특정 구역(local)에서의 최소값인 로컬 미니멈에 도달할 수 있음

⇒ Cost가 최소가 되는 가중치 W를 찾는다는
비용 함수의 목적에 맞지 않기 때문

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

따라서, 가중치 W를 최소로 만드는 새로운 비용 함수를 찾아야 함

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

따라서, 가중치 W를 최소로 만드는 새로운 비용 함수를 찾아야 함

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

샘플 데이터의 개수가 n개

실제값 yi와 예측값 H(xi)의 오차를 나타내는 함수 f

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

따라서, 가중치 W를 최소로 만드는 새로운 비용 함수를 찾아야 함

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

샘플 데이터의 개수가 n개

실제값 yi와 예측값 H(xi)의 오차를 나타내는 함수 f

⇒ 여기서 새로운 함수 f를 어떻게 정의하느냐에 따라서
가중치를 최소화하는 적절한 목적 함수가 완성됨

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

따라서, 가중치 W를 최소로 만드는 새로운 비용 함수를 찾아야 함

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

샘플 데이터의 개수가 n개

실제값 yi와 예측값 H(xi)의 오차를 나타내는 함수 f

⇒ 여기서 새로운 함수 f를 어떻게 정의하느냐에 따라서
가중치를 최소화하는 적절한 목적 함수가 완성됨

목적 함수는 전체 데이터에 대해서 어떤 함수 f의 값의 평균을 계산하고 있음
⇒ 적절한 가중치를 찾기 위해서는 결과적으로 실제값과 예측값에 대한 오차를 줄여야 함

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

따라서, 가중치 W를 최소로 만드는 새로운 비용 함수를 찾아야 함

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

샘플 데이터의 개수가 n개

실제값 yi와 예측값 H(xi)의 오차를 나타내는 함수 f

⇒ 여기서 이 f는 비용 함수(cost function)라고 하고,
식을 다시 쓰면?

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

시그모이드 함수는 0과 1 사이의 y값을 반환함

- 실제값이 1일 때 예측 값이 0에 가까워지면 오차가 커지고,
- 실제값이 0일 때 예측 값이 1에 가까워지면 오차가 커짐

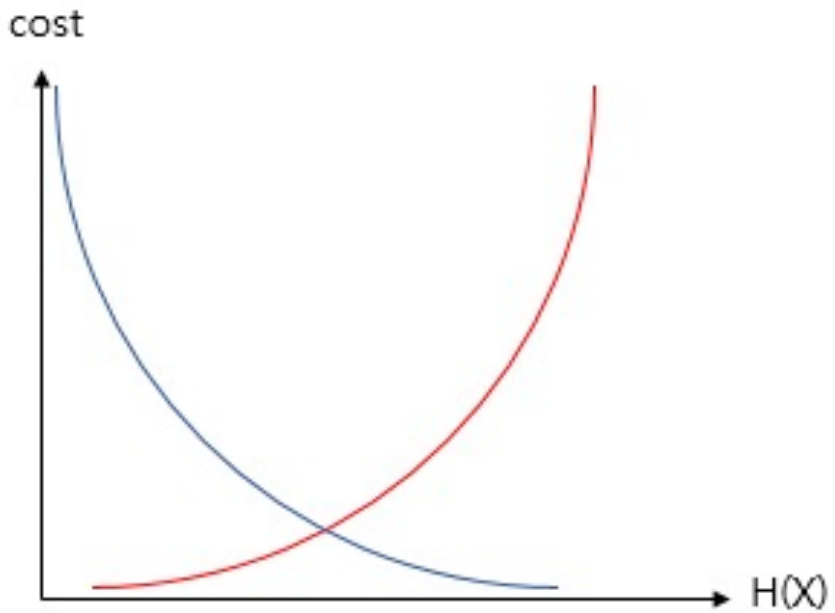
이러한 특성을 반영하는 로그 함수를 통해 표현할 수 있음

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

시그모이드 함수는 0과 1 사이의 y값을 반환함

- 실제값이 1일 때 예측 값이 0에 가까워지면 오차가 커지고,
- 실제값이 0일 때 예측 값이 1에 가까워지면 오차가 커짐

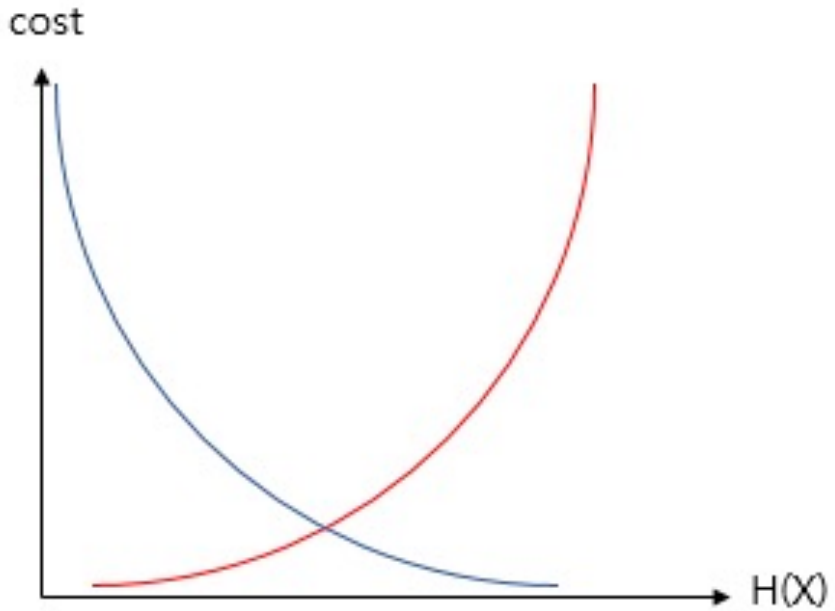
이러한 특성을 반영하는 로그 함수를 통해 표현할 수 있음

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



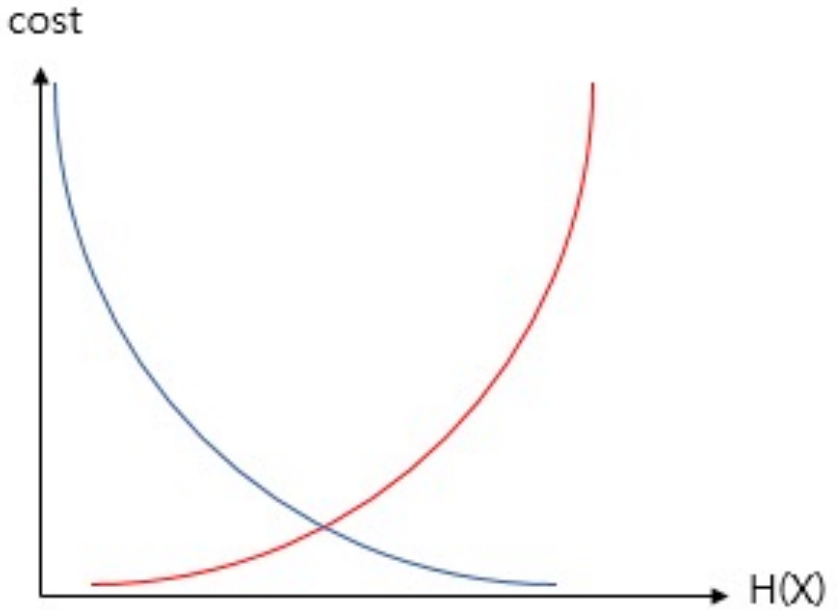
01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

  - 실제값이 1일 때, 예측값인 H(X)의 값이 1이면
    ⇒ 오차(cost)가 0
  - 실제값이 1일 때, 예측값인 H(X)의 값이 0으로 수렴하면
    ⇒ 오차(cost)는 무한대로 발산



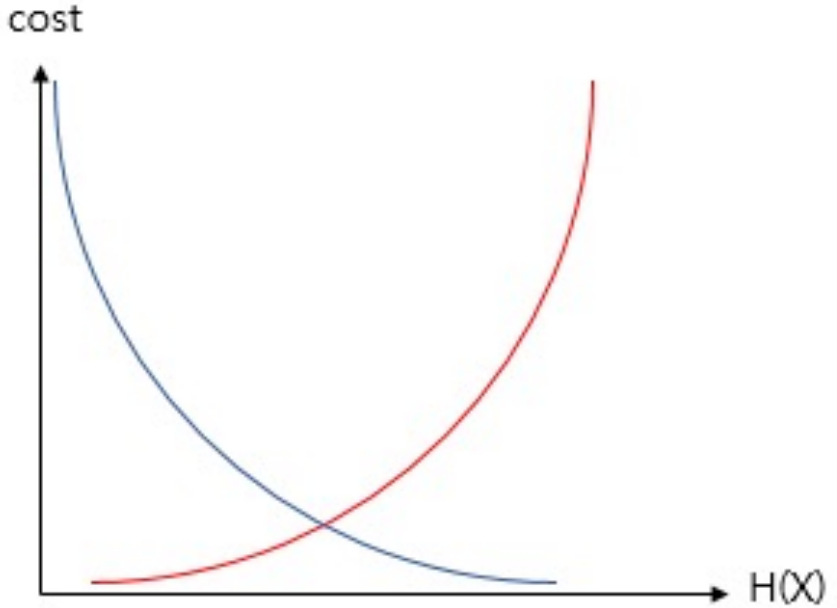
01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

  - 실제값이 1일 때, 예측값인 H(X)의 값이 1이면
    ⇒ 오차(cost)가 0
  - 실제값이 1일 때, 예측값인 H(X)의 값이 0으로 수렴하면
    ⇒ 오차(cost)는 무한대로 발산



- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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

  - 실제값이 0일 때, 예측값인 H(X)의 값이 0으로 수렴하면
    ⇒ 오차(cost)가 0
  - 실제값이 0일 때, 예측값인 H(X)의 값이 1이면
    ⇒ 오차(cost)는 무한대로 발산

01 로지스틱 회귀 (Logistic Regression)

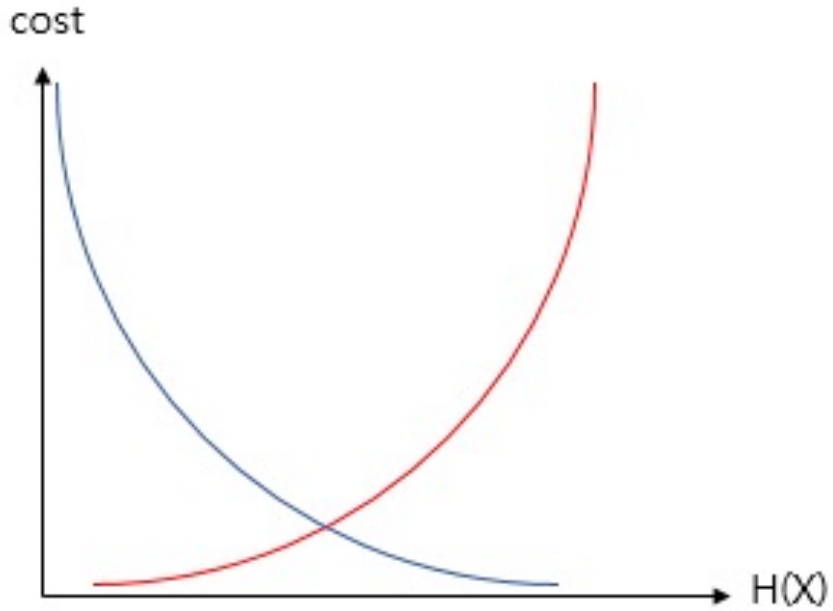
비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



하나의 식으로 표현하면?

$$\text{cost}\left(H(x), y\right) = -[ylogH(x) + (1-y)log(1-H(x))]$$

01 로지스틱 회귀 (Logistic Regression)

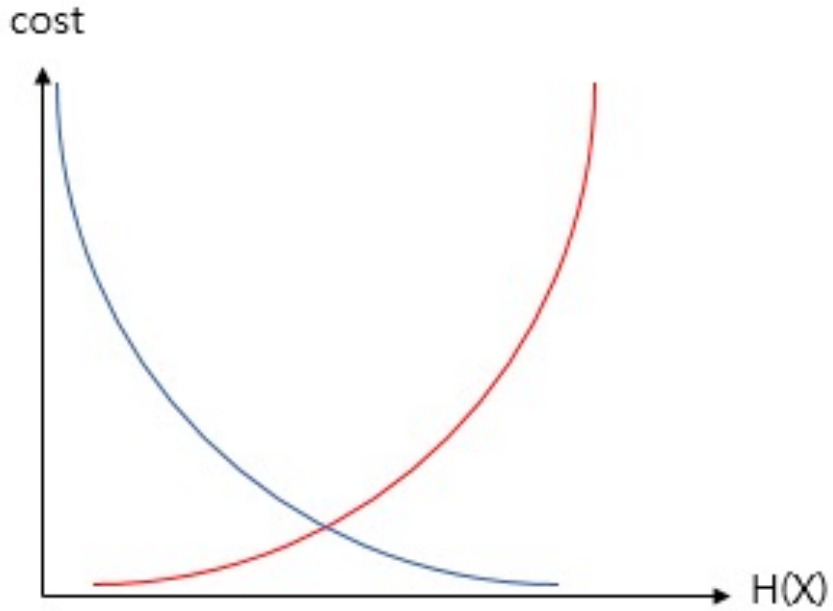
비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



하나의 식으로 표현하면?

$$\text{cost}\,(H(x), y) = -[ylogH(x) + (1-y)log(1-H(x))]$$

01 로지스틱 회귀 (Logistic Regression)

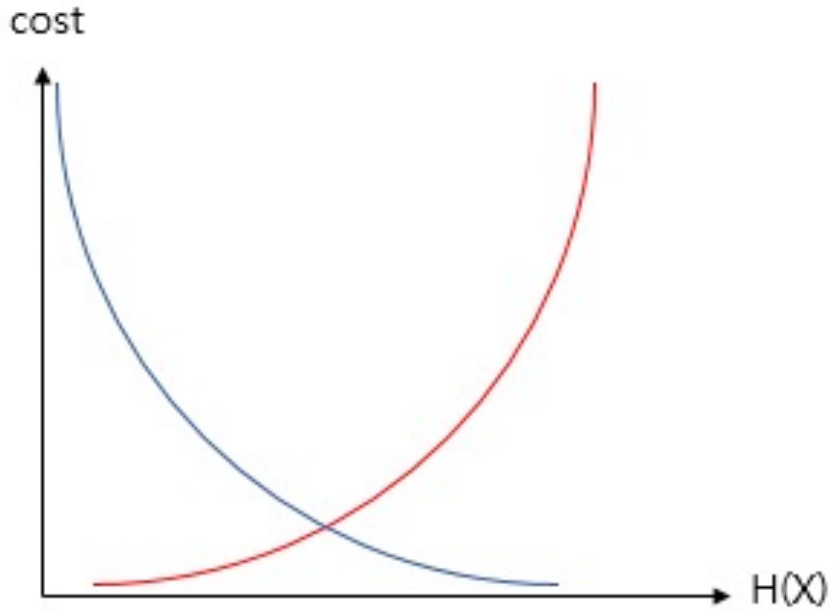
비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



하나의 식으로 표현하면?

$$\text{cost}\,(H(x), y) = -[y log H(x) + (1 - y) log(1 - H(x))]$$

01 로지스틱 회귀 (Logistic Regression)

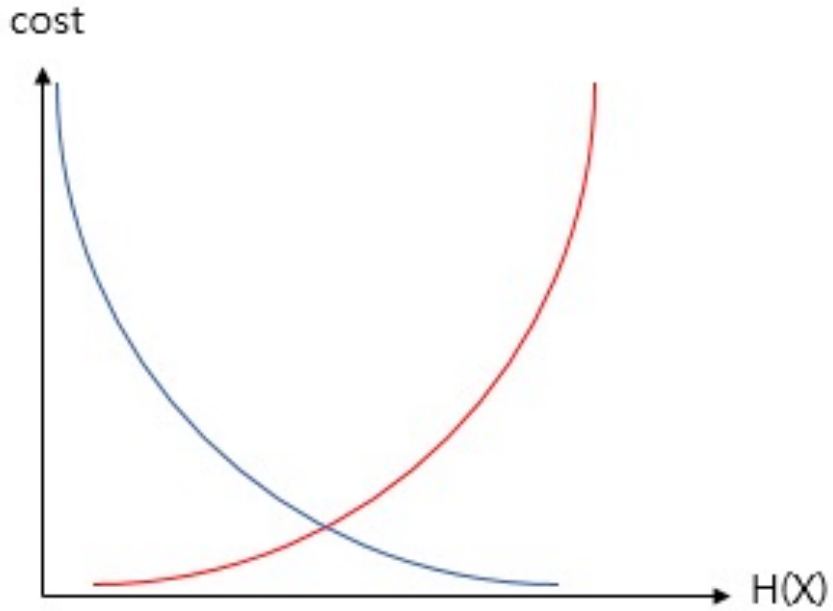
비용 함수 (Cost Function)

- y의 실제값이 1일 때, -logH(X) 그래프를 사용

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

- y의 실제값이 0일 때, –log(1–H(X)) 그래프를 사용

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



하나의 식으로 표현하면?

$$\text{cost}\,(H(x), y) = -[ylogH(x) + (1-y)log(1-H(x))]$$

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

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

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

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

크로스 엔트로피(Cross Entropy)함수

⇒ 결론적으로 로지스틱 회귀는 비용 함수로 크로스 엔트로피 함수를 사용하며,
가중치를 찾기 위해서 크로스 엔트로피 함수의 평균을 취한 함수를 사용함

01 로지스틱 회귀 (Logistic Regression)

비용 함수 (Cost Function)

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

크로스 엔트로피(Cross Entropy)함수

⇒ 결론적으로 로지스틱 회귀는 비용 함수로 크로스 엔트로피 함수를 사용하며,
가중치를 찾기 위해서 크로스 엔트로피 함수의 평균을 취한 함수를 사용함

02 로지스틱 회귀 실습

```
[5]  import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[6]  X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50]) # 임의의 숫자들의 나열
     y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10 이상인 경우 1을, 10 미만인 경우 0을 부여한 레이블

     model = Sequential()
     model.add(Dense(1, input_dim=1, activation='sigmoid')) # 1개의 실수인 X로부터 1개의 실수인 y를 예측

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

     model.fit(X,y, batch_size=1, epochs=200, shuffle=False) # 전체 훈련 횟수는 200회
```

02 로지스틱 회귀 실습

```
[5]  import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[6]  X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50]) # 임의의 숫자들의 나열
     y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10 이상인 경우 1을, 10 미만인 경우 0을 부여한 레이블

     model = Sequential()
     model.add(Dense(1, input_dim=1, activation='sigmoid')) # 1개의 실수인 X로부터 1개의 실수인 y를 예측

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

     model.fit(X,y, batch_size=1, epochs=200, shuffle=False) # 전체 훈련 횟수는 200회
```

02 로지스틱 회귀 실습

```
[5]  import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[6]  X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50]) # 임의의 숫자들의 나열
     y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10 이상인 경우 1을, 10 미만인 경우 0을 부여한 레이블

     model = Sequential()
     model.add(Dense(1, input_dim=1, activation='sigmoid')) # 1개의 실수인 X로부터 1개의 실수인 y를 예측

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

     model.fit(X,y, batch_size=1, epochs=200, shuffle=False) # 전체 훈련 횟수는 200회
```

02 로지스틱 회귀 실습

```
[5]  import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[6]  X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50]) # 임의의 숫자들의 나열
     y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10 이상인 경우 1을, 10 미만인 경우 0을 부여한 레이블

     model = Sequential()
     model.add(Dense(1, input_dim=1, activation='sigmoid')) # 1개의 실수인 X로부터 1개의 실수인 y를 예측

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

     model.fit(X,y, batch_size=1, epochs=200, shuffle=False) # 전체 훈련 횟수는 200회
```

02 로지스틱 회귀 실습

```
[5]  import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[6]  X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50]) # 임의의 숫자들의 나열
     y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) # 숫자 10 이상인 경우 1을, 10 미만인 경우 0을 부여한 레이블

     model = Sequential()
     model.add(Dense(1, input_dim=1, activation='sigmoid')) # 1개의 실수인 X로부터 1개의 실수인 y를 예측

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

     model.fit(X,y, batch_size=1, epochs=200, shuffle=False) # 전체 훈련 횟수는 200회
```

02 로지스틱 회귀 실습

```
Epoch 1/200
13/13 [==============================] - 0s 1ms/step - loss: 0.4373 - binary_accuracy: 0.9231
Epoch 2/200
13/13 [==============================] - 0s 1ms/step - loss: 0.4170 - binary_accuracy: 0.9231
Epoch 3/200
13/13 [==============================] - 0s 1ms/step - loss: 0.3968 - binary_accuracy: 0.9231
Epoch 4/200
13/13 [==============================] - 0s 1ms/step - loss: 0.3768 - binary_accuracy: 0.9231

Epoch 190/200
13/13 [==============================] - 0s 1ms/step - loss: 0.0902 - binary_accuracy: 0.9231
Epoch 191/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0900 - binary_accuracy: 0.9231
Epoch 192/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0898 - binary_accuracy: 1.0000
Epoch 193/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0896 - binary_accuracy: 1.0000

Epoch 197/200
13/13 [==============================] - 0s 1ms/step - loss: 0.0888 - binary_accuracy: 1.0000
Epoch 198/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0886 - binary_accuracy: 1.0000
Epoch 199/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0884 - binary_accuracy: 1.0000
Epoch 200/200
13/13 [==============================] - 0s 2ms/step - loss: 0.0882 - binary_accuracy: 1.0000
```

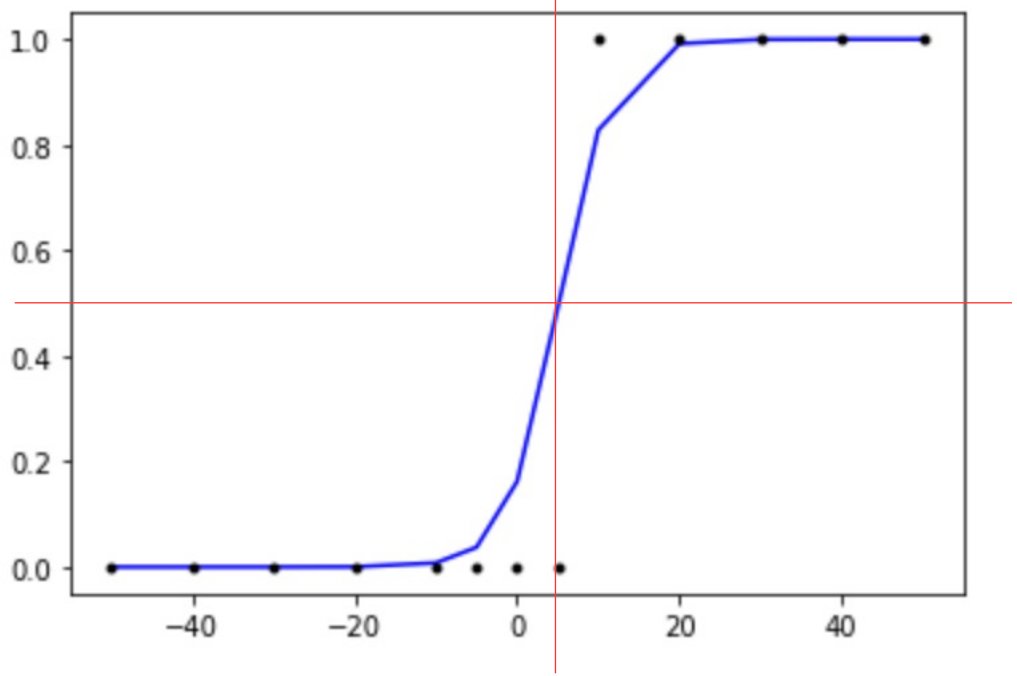
02 로지스틱 회귀 실습

실제값과 오차를 최소화하는 W와 b의 값을 가진
시그모이드 함수 그래프 플롯

적어도 X값이 5일 때는 y값이 0.5 보다 작고,
X값이 10일 때는 y값이 0.5를 넘을 것

```
[7]  plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

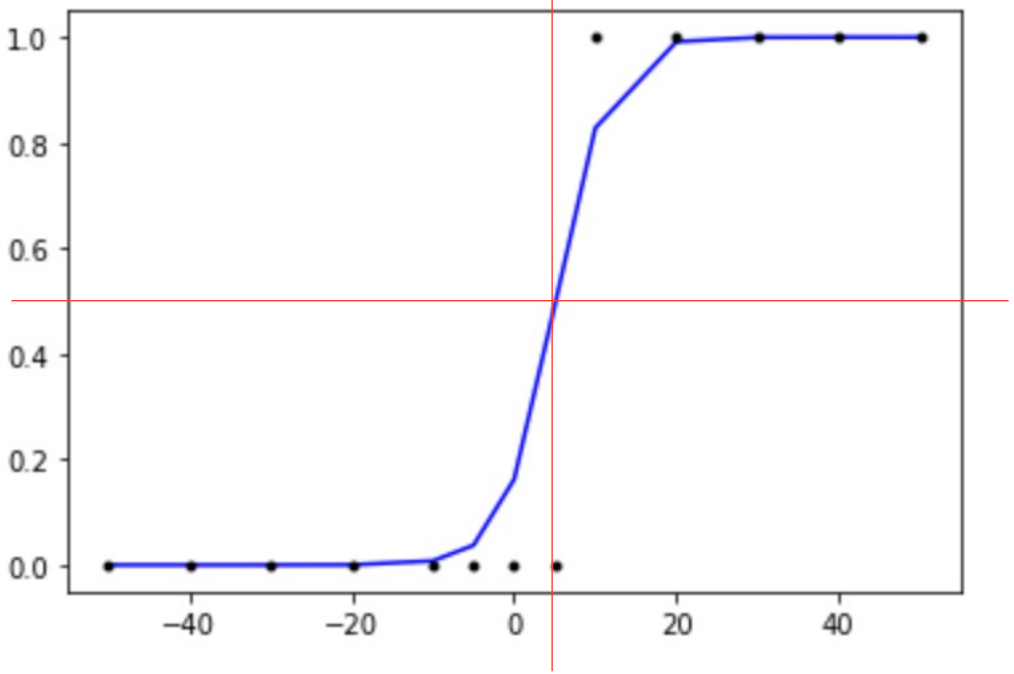
```
[<matplotlib.lines.Line2D at 0x7fd3fcb14190>,
 <matplotlib.lines.Line2D at 0x7fd3fcafbe10>]
```



02 로지스틱 회귀 실습

```
[7]  plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fd3fcb14190>,
 <matplotlib.lines.Line2D at 0x7fd3fcafbe10>]
```

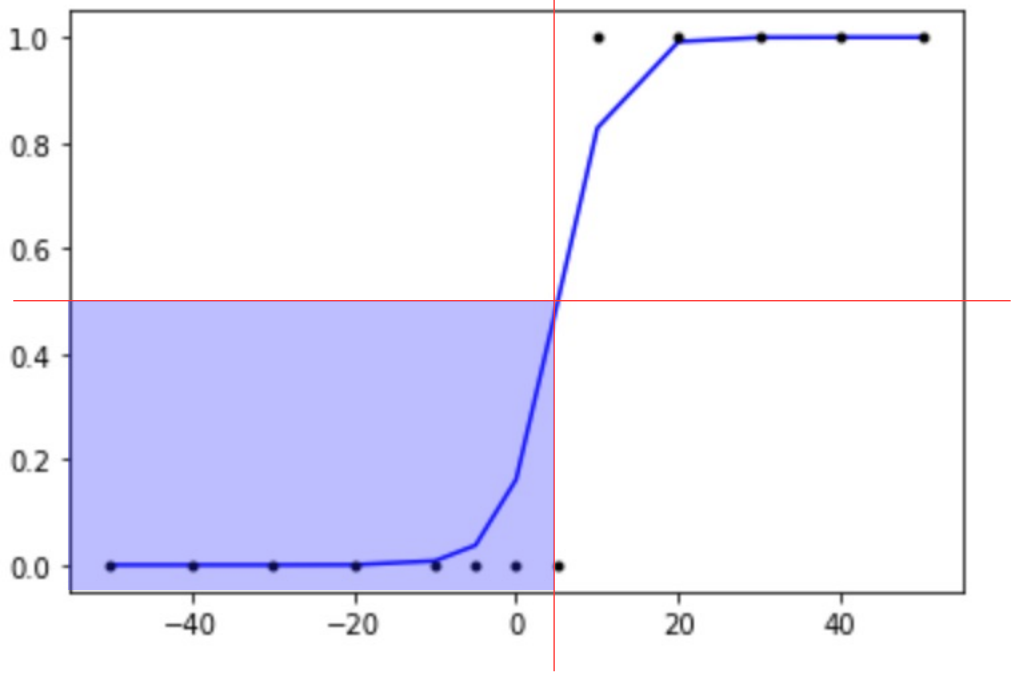



```
[8]  print(model.predict([1, 2, 3, 4, 4.5]))
     print(model.predict([11, 21, 31, 41, 500]))
```

02 로지스틱 회귀 실습

```
[7]  plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fd3fcb14190>,
 <matplotlib.lines.Line2D at 0x7fd3fcafbe10>]
```




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

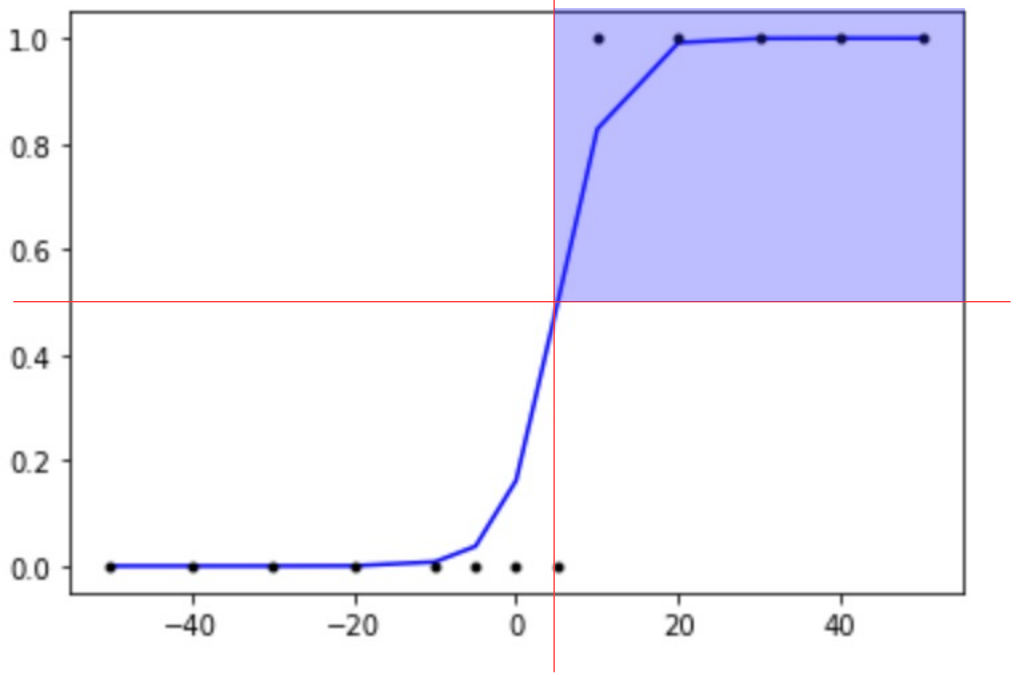
```
[[0.21056241]
 [0.26892874]
 [0.33657593]
 [0.41165772]
 [0.45106226]]
[[0.8691201 ]
 [0.99398786]
 [0.9997571 ]
 [0.9999903 ]
 [1.        ]]
```

X값이 5보다 작을 때는 0.5보다 작은 y값을 출력

02 로지스틱 회귀 실습

```
[7]  plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fd3fcb14190>,
 <matplotlib.lines.Line2D at 0x7fd3fcafbe10>]
```




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

```
[[0.21056241]
 [0.26892874]
 [0.33657593]
 [0.41165772]
 [0.45106226]]
[[0.8691201 ]
 [0.99398786]
 [0.9997571 ]
 [0.9999903 ]
 [1.        ]]
```

X값이 10보다 클 때는 0.5보다 큰 y값을 출력

03 다중 입력에 대한 실습

다중 선형 회귀란?

독립 변수가 2개 이상인 선형 회귀 (입력 벡터의 차원이 2이상)

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (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

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (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

가설 $H(X) = W_1x_1 + W_2x_2 + W_3x_3 + b$

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

	중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (y)
훈련	70	85	11	73
	71	89	18	82
	50	80	20	72
	99	20	10	57
	50	10	10	34
테스트	20	99	10	58
	40	50	20	56

가설 $H(X) = W_1x_1 + W_2x_2 + W_3x_3 + b$

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

	중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (y)
훈련	70	85	11	73
	71	89	18	82
	50	80	20	72
	99	20	10	57
	50	10	10	34
테스트	20	99	10	58
	40	50	20	56

```
[1] import numpy as np
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers
```

```
[2] # 입력 벡터의 차원은 3. 즉, input_dim=3
    X = np.array([[70,85,11], [71,89,18], [50,80,20], [99,20,10], [50,10,10]])

    # 출력 벡터의 차원은 1. 즉, output_dim=1
    y = np.array([73,82,72,57,34])

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

    # 학습률(learning rate, lr)은 0.00001
    sgd=optimizers.SGD(lr=0.00001)

    # 손실함수(loss function)은 평균제곱오차 mse를 사용함.
    model.compile(optimizer=sgd, loss='mse', metrics=['mse'])

    # 주어진 X와 y 데이터에 대해서 오차를 최소화하는 작업은 2,000번 시도함.
    model.fit(X, y, batch_size=1, epochs=2000, shuffle=False)
```

가설 $H(X) = W_1x_1 + W_2x_2 + W_3x_3 + b$

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

	중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (y)
훈련	70	85	11	73
	71	89	18	82
	50	80	20	72
	99	20	10	57
	50	10	10	34
테스트	20	99	10	58
	40	50	20	56

```
[1] import numpy as np
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers
```

```
[2] # 입력 벡터의 차원은 3. 즉, input_dim=3
    X = np.array([[70,85,11], [71,89,18], [50,80,20], [99,20,10], [50,10,10]])

    # 출력 벡터의 차원은 1. 즉, output_dim=1
    y = np.array([73,82,72,57,34])

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

    # 학습률(learning rate, lr)은 0.00001
    sgd=optimizers.SGD(lr=0.00001)

    # 손실함수(loss function)은 평균제곱오차 mse를 사용함.
    model.compile(optimizer=sgd, loss='mse', metrics=['mse'])

    # 주어진 X와 y 데이터에 대해서 오차를 최소화하는 작업은 2,000번 시도함.
    model.fit(X, y, batch_size=1, epochs=2000, shuffle=False)
```

가설 $H(X) = W_1x_1 + W_2x_2 + W_3x_3 + b$

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

```
[1]  import numpy as np
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers
```

```
[2]  # 입력 벡터의 차원은 3. 즉, input_dim=3
     X = np.array([[70,85,11], [71,89,18], [50,80,20], [99,20,10], [50,10,10]])

     # 출력 벡터의 차원은 1. 즉, output_dim=1
     y = np.array([73,82,72,57,34])

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

     # 학습률(learning rate, lr)은 0.00001
     sgd=optimizers.SGD(lr=0.00001)

     # 손실함수(loss function)은 평균제곱오차 mse를 사용함.
     model.compile(optimizer=sgd, loss='mse', metrics=['mse'])

     # 주어진 X와 y 데이터에 대해서 오차를 최소화하는 작업은 2,000번 시도함.
     model.fit(X, y, batch_size=1, epochs=2000, shuffle=False)
```

```
Epoch 1/2000
5/5 [==============================] - 0s 2ms/step - loss: 932.6215 - mse: 932.6215
Epoch 2/2000
5/5 [==============================] - 0s 2ms/step - loss: 625.9563 - mse: 625.9563
Epoch 3/2000
5/5 [==============================] - 0s 3ms/step - loss: 485.3116 - mse: 485.3116
Epoch 4/2000
5/5 [==============================] - 0s 2ms/step - loss: 391.9318 - mse: 391.9318
Epoch 5/2000
5/5 [==============================] - 0s 2ms/step - loss: 320.6948 - mse: 320.6948


Epoch 1997/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1998/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1999/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0257 - mse: 0.0257
Epoch 2000/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0257 - mse: 0.0257
<tensorflow.python.keras.callbacks.History at 0x7eff03094f90>
```

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

	중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (y)
훈련	70	85	11	73
	71	89	18	82
	50	80	20	72
	99	20	10	57
	50	10	10	34
테스트	20	99	10	58
	40	50	20	56

```
Epoch 1/2000
5/5 [==============================] - 0s 2ms/step - loss: 932.6215 - mse: 932.6215
Epoch 2/2000
5/5 [==============================] - 0s 2ms/step - loss: 625.9563 - mse: 625.9563
Epoch 3/2000
5/5 [==============================] - 0s 3ms/step - loss: 485.3116 - mse: 485.3116
Epoch 4/2000
5/5 [==============================] - 0s 2ms/step - loss: 391.9318 - mse: 391.9318
Epoch 5/2000
5/5 [==============================] - 0s 2ms/step - loss: 320.6948 - mse: 320.6948


Epoch 1997/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1998/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1999/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0257 - mse: 0.0257
Epoch 2000/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0257 - mse: 0.0257
<tensorflow.python.keras.callbacks.History at 0x7eff03094f90>
```

```
[3] print(model.predict(X))
```

```
[[73.07568]
 [81.97703]
 [71.97826]
 [57.1523 ]
 [33.70476]]
```

정확하지는 않지만,
어느정도 실제값에 근접한 예측

03 다중 입력에 대한 실습

다중 선형 회귀 – 예시 문제 풀이

중간고사 점수, 기말고사 점수, 추가점수를 이용하여 최종 점수 구하기

⇒ y를 결정하는데 있어 독립 변수가 3개 이상인 선형 회귀

데이터

	중간고사 점수 (x_1)	기말고사 점수 (x_2)	추가 점수 (x_3)	최종 점수 (y)
훈련	70	85	11	73
	71	89	18	82
	50	80	20	72
	99	20	10	57
	50	10	10	34
테스트	20	99	10	58
	40	50	20	56

```
Epoch 1/2000
5/5 [==============================] - 0s 2ms/step - loss: 932.6215 - mse: 932.6215
Epoch 2/2000
5/5 [==============================] - 0s 2ms/step - loss: 625.9563 - mse: 625.9563
Epoch 3/2000
5/5 [==============================] - 0s 3ms/step - loss: 485.3116 - mse: 485.3116
Epoch 4/2000
5/5 [==============================] - 0s 2ms/step - loss: 391.9318 - mse: 391.9318
Epoch 5/2000
5/5 [==============================] - 0s 2ms/step - loss: 320.6948 - mse: 320.6948


Epoch 1997/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1998/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0258 - mse: 0.0258
Epoch 1999/2000
5/5 [==============================] - 0s 4ms/step - loss: 0.0257 - mse: 0.0257
Epoch 2000/2000
5/5 [==============================] - 0s 3ms/step - loss: 0.0257 - mse: 0.0257
<tensorflow.python.keras.callbacks.History at 0x7eff03094f90>
```

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

```
[[58.00202 ]
 [55.831863]]
```

테스트 데이터셋으로 예측

03 다중 입력에 대한 실습

다중 로지스틱 회귀

y를 결정하는데 있어 독립변수 x가 2개 이상인 로지스틱 회귀

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

데이터

SepalLength Cm (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

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

데이터

SepalLength Cm (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

가설 $H(X) = sigmoid(W_1x_1 + W_2x_2 + b)$

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

데이터

SepalLength Cm (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

가설 $H(X) = sigmoid(W_1x_1 + W_2x_2 + b)$

```
[5] import numpy as np
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers

    # 입력 벡터의 차원은 2. 즉, input_dim=2.
    X = np.array([[0,0], [0,1], [1,0], [1,1]])

    # 출력 벡터의 차원은 1. 즉, output_dim=1.
    y = np.array([0, 1, 1, 1])

    model = Sequential()
    model.add(Dense(1, input_dim=2, activation='sigmoid'))

    # sgd는 경사 하강법을 의미.
    # 손실 함수(Loss function)는 binary_crossentropy(이진 크로스 엔트로피)를 사용함.
    model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])

    # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 800번 시도함.
    model.fit(X, y, batch_size=1, epochs=800, shuffle=False)
```

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

```
[5]  import numpy as np
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras import optimizers

     # 입력 벡터의 차원은 2. 즉, input_dim=2.
     X = np.array([[0,0], [0,1], [1,0], [1,1]])

     # 출력 벡터의 차원은 1. 즉, output_dim=1.
     y = np.array([0, 1, 1, 1])

     model = Sequential()
     model.add(Dense(1, input_dim=2, activation='sigmoid'))

     # sgd는 경사 하강법을 의미.
     # 손실 함수(Loss function)는 binary_crossentropy(이진 크로스 엔트로피)를 사용함.
     model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])

     # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 800번 시도함.
     model.fit(X, y, batch_size=1, epochs=800, shuffle=False)
```

```
Epoch 1/800
4/4 [==============================] - 0s 3ms/step - loss: 0.9834 - binary_accuracy: 0.5000
Epoch 2/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9695 - binary_accuracy: 0.2500
Epoch 3/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9560 - binary_accuracy: 0.2500
Epoch 4/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9429 - binary_accuracy: 0.2500
Epoch 5/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9301 - binary_accuracy: 0.2500

Epoch 796/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2266 - binary_accuracy: 1.0000
Epoch 797/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2264 - binary_accuracy: 1.0000
Epoch 798/800
4/4 [==============================] - 0s 6ms/step - loss: 0.2262 - binary_accuracy: 1.0000
Epoch 799/800
4/4 [==============================] - 0s 7ms/step - loss: 0.2261 - binary_accuracy: 1.0000
Epoch 800/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2259 - binary_accuracy: 1.0000
<tensorflow.python.keras.callbacks.History at 0x7efefc651d90>
```

03 다중 입력에 대한 실습

다중 로지스틱 회귀 – 예시 문제 풀이

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀있는 데이터를 사용하여, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측할 수 있는 모델을 만들기

⇒ y를 결정하는데 있어 독립 변수가 2개인 로지스틱 회귀

```
[5] import numpy as np
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers

    # 입력 벡터의 차원은 2. 즉, input_dim=2.
    X = np.array([[0,0], [0,1], [1,0], [1,1]])

    # 출력 벡터의 차원은 1. 즉, output_dim=1.
    y = np.array([0, 1, 1, 1])

    model = Sequential()
    model.add(Dense(1, input_dim=2, activation='sigmoid'))

    # sgd는 경사 하강법을 의미.
    # 손실 함수(Loss function)는 binary_crossentropy(이진 크로스 엔트로피)를 사용함.
    model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])

    # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 800번 시도함.
    model.fit(X, y, batch_size=1, epochs=800, shuffle=False)
```

```
Epoch 1/800
4/4 [==============================] - 0s 3ms/step - loss: 0.9834 - binary_accuracy: 0.5000
Epoch 2/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9695 - binary_accuracy: 0.2500
Epoch 3/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9560 - binary_accuracy: 0.2500
Epoch 4/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9429 - binary_accuracy: 0.2500
Epoch 5/800
4/4 [==============================] - 0s 4ms/step - loss: 0.9301 - binary_accuracy: 0.2500

Epoch 796/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2266 - binary_accuracy: 1.0000
Epoch 797/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2264 - binary_accuracy: 1.0000
Epoch 798/800
4/4 [==============================] - 0s 6ms/step - loss: 0.2262 - binary_accuracy: 1.0000
Epoch 799/800
4/4 [==============================] - 0s 7ms/step - loss: 0.2261 - binary_accuracy: 1.0000
Epoch 800/800
4/4 [==============================] - 0s 5ms/step - loss: 0.2259 - binary_accuracy: 1.0000
<tensorflow.python.keras.callbacks.History at 0x7efefc651d90>
```

```
[6] print(model.predict(X))
```

```
[[0.43147343]
 [0.87275296]
 [0.8356748 ]
 [0.9787049 ]]
```

⇒ 입력이 둘다 0, 0인 경우를 제외한 나머지 세 입력쌍(pair)에 대해서는 전부 값이 0.5를 넘음

04 벡터와 행렬 연산

벡터와 행렬 연산이 왜 필요한가?

- 인공 신경망 개념을 이해하기 위해서는
  각 변수들의 연산을 벡터와 행렬 연산으로 이해할 수 있어야 함
- 특히, 데이터와 변수의 개수로부터 행렬의 크기, 더 나아가 텐서의 크기를 산정할 수 있어야 함

04 벡터와 행렬 연산

벡터와 행렬과 텐서

벡터

- 크기와 방향을 가진 양
- 숫자가 나열된 형상
- 파이썬에서는 1차원 배열 또는 리스트로 표현함

행렬

- 행과 열을 가지는 2차원 형상을 가진 구조
- 가로줄을 행(row), 세로줄을 열(column)이라고 함
- 파이썬에서는 2차원 배열로 표현함

텐서

- 3차원부터는 텐서라고 부름
- 파이썬에서는 3차원 이상의 배열로 표현함

04 벡터와 행렬 연산

텐서 (Tensor)

Scalar (0차원 텐서)

- 하나의 실수값으로 이루어진 데이터

```
[3] d = np.array(5)
    print(d.ndim) # 차원수 출력
    print(d.shape) # 텐서의 크기 출력
```

```
0
()
```

SCALAR

11

04 벡터와 행렬 연산

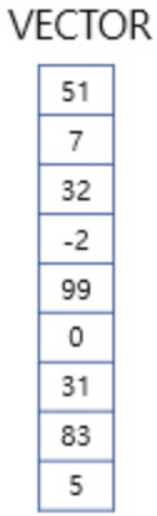
텐서 (Tensor)

Vector (1차원 텐서)

- 숫자를 특정 순서대로 배열한 것
- [!] 벡터의 차원과 텐서의 차원 개념을 구분해야 함
- 예제는 4차원 벡터이지만, 1차원 텐서
  - 벡터의 차원: 하나의 축에 차원들이 존재하는 것
  - 텐서의 차원: 축의 개수를 의미

```
[4]  d = np.array([1,2,3,4])
     print(d.ndim) # 차원수 출력
     print(d.shape) # 텐서의 크기 출력
```

```
1
(4,)
```



04 벡터와 행렬 연산

텐서 (Tensor)

Matrix (2차원 텐서)

- 행과 열이 존재하는 벡터의 배열
- 3개의 커다란 데이터가 있는데,
  그 각각의 커다란 데이터는 작은 데이터 4개로
  이루어졌다고 생각할 수 있음

```
[5] d = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
    print(d.ndim) # 차원수 출력
    print(d.shape) # 텐서의 크기 출력
```

```
2
(3, 4)
```

MATRIX

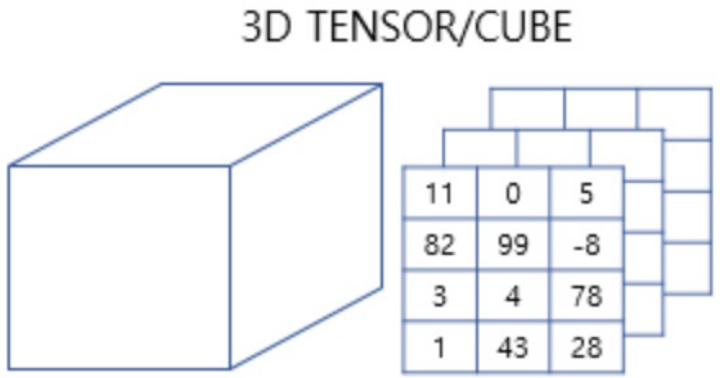
11	0	5
82	99	-8
3	4	78
1	43	28

04 벡터와 행렬 연산

텐서 (Tensor)

Tensor (3차원 텐서)

- 3차원 이상의 텐서로부터 본격적으로 텐서라고 부름
- 2개의 커다란 데이터가 있는데,
  그 각각은 3개의 더 작은 데이터로 구성되며,
  그 3개의 데이터는 또한 더 작은 5개의 데이터로 구성됨




```
[6]  d=np.array([
                 [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [10, 11, 12, 13, 14]],
                 [[15, 16, 17, 18, 19], [19, 20, 21, 22, 23], [23, 24, 25, 26, 27]]
                 ])
     print(d.ndim) # 차원수 출력
     print(d.shape) # 텐서의 크기 출력
```

```
3
(2, 3, 5)
```

04 벡터와 행렬 연산

텐서 (Tensor)

Tensor (3차원 텐서)

- 자연어 처리에서 특히 자주 보게 되는 것이 이 3D 텐서
- 3D 텐서는 시퀀스 데이터(sequence data)를 표현할 때 자주 사용되기 때문

- 여기서 시퀀스 데이터는 주로 단어의 시퀀스를 의미하며,
- 시퀀스는 주로 문장이나 문서, 뉴스 기사 등의 텍스트가 될 수 있음
  ⇒ 이 경우 3D 텐서는 (samples, timesteps, word_dim)이 됨

04 벡터와 행렬 연산

자연어처리에서 3차원 텐서의 개념이 어떻게 사용될까?

훈련 데이터

문서1 : I like NLP
문서2 : I like DL
문서3 : DL is AI

인코딩

단어	One-hot vector
I	[1 0 0 0 0 0]
like	[0 1 0 0 0 0]
NLP	[0 0 1 0 0 0]
DL	[0 0 0 1 0 0]
is	[0 0 0 0 1 0]
AI	[0 0 0 0 0 1]

04 벡터와 행렬 연산

자연어처리에서 3차원 텐서의 개념이 어떻게 사용될까?

훈련 데이터

문서1 : I like NLP
문서2 : I like DL
문서3 : DL is AI

인코딩

단어	One-hot vector
I	[1 0 0 0 0 0]
like	[0 1 0 0 0 0]
NLP	[0 0 1 0 0 0]
DL	[0 0 0 1 0 0]
is	[0 0 0 0 1 0]
AI	[0 0 0 0 0 1]

```
[[[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]],
[[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0]],
[[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]]]
```

04 벡터와 행렬 연산

자연어처리에서 3차원 텐서의 개념이 어떻게 사용될까?

훈련 데이터

문서1 : I like NLP
문서2 : I like DL
문서3 : DL is AI

인코딩

단어	One-hot vector
I	[1 0 0 0 0 0]
like	[0 1 0 0 0 0]
NLP	[0 0 1 0 0 0]
DL	[0 0 0 1 0 0]
is	[0 0 0 0 1 0]
AI	[0 0 0 0 0 1]

```
[[[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]],
[[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0]],
[[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]]]
```

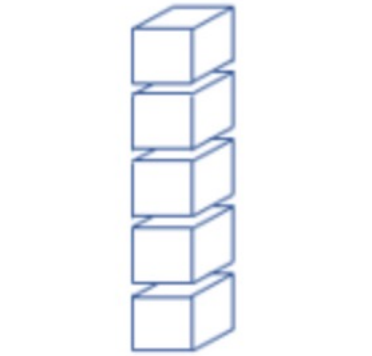
(3, 3, 6)의 크기를 가지는 3D 텐서

04 벡터와 행렬 연산

그 이상의 텐서

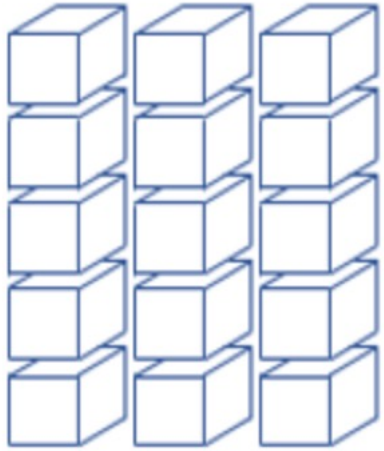
텐서는 배열로서 계속 확장될 수 있음

4차원 텐서



4D TENSOR
VECTOR OF CUBES

5차원 텐서



5D TENSOR
MATRIX OF CUBES

04 벡터와 행렬 연산

벡터와 행렬의 덧셈과 뺄셈

같은 크기의 두 개의 벡터나 행렬은 덧셈과 뺄셈을 할 수 있음 (같은 위치의 원소끼리 연산)
⇒ 요소별(element-wise) 연산

04 벡터와 행렬 연산

벡터의 덧셈과 뺄셈

$$a = \begin{bmatrix} 8 \\ 4 \\ 5 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$a + b = \begin{bmatrix} 8 \\ 4 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 6 \\ 8 \end{bmatrix}$$

$$a - b = \begin{bmatrix} 8 \\ 4 \\ 5 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2 \\ 2 \end{bmatrix}$$

- Numpy로 구현

```
[7]  a = np.array([8, 4, 5])
     b = np.array([1, 2, 3])
     print(a+b)
     print(a-b)
```

```
[9 6 8]
[7 2 2]
```

04 벡터와 행렬 연산

행렬의 덧셈과 뺄셈

$$a = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{bmatrix} \quad b = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$a + b = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{bmatrix} + \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 15 & 26 & 37 & 48 \\ 51 & 62 & 73 & 84 \end{bmatrix}$$

$$a - b = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{bmatrix} - \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 14 & 23 & 32 \\ 49 & 58 & 67 & 76 \end{bmatrix}$$

- Numpy로 구현

```
[8]  import numpy as np
     a = np.array([[10, 20, 30, 40], [50, 60, 70, 80]])
     b = np.array([[5, 6, 7, 8], [1, 2, 3, 4]])
     print(a+b)
     print(a-b)
```

```
[[15 26 37 48]
 [51 62 73 84]]
[[ 5 14 23 32]
 [49 58 67 76]]
```

04 벡터와 행렬 연산

벡터의 내적 (inner product)

- 벡터의 점곱(dot product)
- a·b

04 벡터와 행렬 연산

벡터의 내적 (inner product)

벡터의 내적이 성립하기 위해서는?

- 두 벡터의 차원이 같아야 하며,
- 두 벡터 중 앞의 벡터가 행벡터(가로 방향 벡터)이고 뒤의 벡터가 열벡터(세로 방향 벡터)여야 함

벡터의 내적의 결과값

- 스칼라(0차원 텐서)가 됨

04 벡터와 행렬 연산

벡터의 내적 (inner product)

벡터의 내적이 성립하기 위해서는?

- 두 벡터의 차원이 같아야 하며,
- 두 벡터 중 앞의 벡터가 행벡터(가로 방향 벡터)이고 뒤의 벡터가 열벡터(세로 방향 벡터)여야 함

벡터의 내적의 결과값

- 스칼라(0차원 텐서)가 됨

$$a \cdot b = [1\ 2\ 3] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1\times4 + 2\times5 + 3\times6 = 32(\text{스칼라})$$

```
[9]  a = np.array([1, 2, 3])
     b = np.array([4, 5, 6])
     print(np.dot(a,b))

     32
```

04 벡터와 행렬 연산

행렬의 곱셈

- 행렬의 곱셈을 이해하기 위해서는 벡터의 내적을 이해해야 함
- 왼쪽 행렬의 행벡터(가로 방향 벡터)와 오른쪽 행렬의 열벡터(세로 방향 벡터)의 내적(대응하는 원소들의 곱의 합)이 결과 행렬의 원소가 됨

04 벡터와 행렬 연산

다중 선형 회귀 행렬 연산으로 이해하기

독립 변수가 2개 이상일 때, 1개의 종속 변수를 예측하는 문제를 행렬의 연산으로 표현한다면?

04 벡터와 행렬 연산

다중 선형 회귀 행렬 연산으로 이해하기

독립 변수가 2개 이상일 때, 1개의 종속 변수를 예측하는 문제를 행렬의 연산으로 표현한다면?

독립변수 x가 n개인 다중 선형 회귀 수식

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n + b$$

04 벡터와 행렬 연산

독립변수 x가 n개인 다중 선형 회귀 수식

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n + b$$

⇒ 입력 벡터 [x1,...xn]와 가중치 벡터 [w1,...,wn]의 내적으로 표현할 수 있음

$$y = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \cdots \\ w_n \end{bmatrix} + b = x_1w_1 + x_2w_2 + x_3w_3 + \ldots + x_nw_n + b$$

⇒ 또는 가중치 벡터 [w1,...,wn]와 입력 벡터 [x1,...xn]의 내적으로 표현할 수도 있음

$$y = \begin{bmatrix} w_1 & w_2 & w_3 & \cdots & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} + b = x_1w_1 + x_2w_2 + x_3w_3 + \ldots + x_nw_n + b$$

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해본다고 합시다.

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해보기

데이터

집의 크기 (x_1)	방의 수 (x_2)	층의 수 (x_3)	얼마나 오래 (x_4)	집의 가격 (y)
1800	2	1	10	207
1200	4	2	20	176
1700	3	2	15	213
1500	5	1	10	234
1100	2	2	10	155

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해보기

데이터

집의 크기 (x_1)	방의 수 (x_2)	층의 수 (x_3)	얼마나 오래 (x_4)	집의 가격 (y)
1800	2	1	10	207
1200	4	2	20	176
1700	3	2	15	213
1500	5	1	10	234
1100	2	2	10	155

입력 행렬 X와 가중치 벡터 W의 곱으로 표현

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \\ x_{51} & x_{52} & x_{53} & x_{54} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 + x_{14}w_4 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 + x_{24}w_4 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 + x_{34}w_4 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 + x_{44}w_4 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 + x_{54}w_4 \end{bmatrix}$$

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해보기

데이터

집의 크기 (x_1)	방의 수 (x_2)	층의 수 (x_3)	얼마나 오래 (x_4)	집의 가격 (y)
1800	2	1	10	207
1200	4	2	20	176
1700	3	2	15	213
1500	5	1	10	234
1100	2	2	10	155

가설 $H(X) = XW + B$

$$\begin{bmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 + x_{14}w_4 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 + x_{24}w_4 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 + x_{34}w_4 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 + x_{44}w_4 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 + x_{54}w_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \\ b \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해보기

데이터

집의 크기 (x_1)	방의 수 (x_2)	층의 수 (x_3)	얼마나 오래 (x_4)	집의 가격 (y)
1800	2	1	10	207
1200	4	2	20	176
1700	3	2	15	213
1500	5	1	10	234
1100	2	2	10	155

가설 $H(X) = XW + B$

$$\begin{bmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 + x_{14}w_4 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 + x_{24}w_4 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 + x_{34}w_4 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 + x_{44}w_4 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 + x_{54}w_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \\ b \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

04 벡터와 행렬 연산

예시 문제:
집의 크기, 방의 수, 층의 수, 집이 얼마나 오래되었는지와 집의 가격이 기록된 부동산 데이터를 학습하여 새로운 집의 정보가 들어왔을 때, 집의 가격을 예측해보기

데이터

집의 크기 (x_1)	방의 수 (x_2)	층의 수 (x_3)	얼마나 오래 (x_4)	집의 가격 (y)
1800	2	1	10	207
1200	4	2	20	176
1700	3	2	15	213
1500	5	1	10	234
1100	2	2	10	155

또는, 가설 $H(X) = XW + B$

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} & x_{31} & x_{41} & x_{51} \\ x_{12} & x_{22} & x_{32} & x_{42} & x_{52} \\ x_{13} & x_{23} & x_{33} & x_{43} & x_{53} \\ x_{14} & x_{24} & x_{34} & x_{44} & x_{54} \end{bmatrix} + \begin{bmatrix} b & b & b & b & b \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix}$$

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

X: 독립 변수 x의 행렬 (= 입력 행렬(Input Matrix))
Y: 종속 변수 y의 행렬 (= 출력 행렬(Output Matrix))
W: 가중치 W의 행렬
B: 편향 b의 행렬

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

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

B행렬은 Y행렬의 크기에 영향 X
⇒ B행렬의 크기는 Y행렬의 크기와 같음

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

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

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

04 벡터와 행렬 연산

가중치(W)와 편향행렬(b)의 크기 결정
특성을 행렬의 열로 보는 경우를 가정하여, 행렬의 크기가 어떻게 결정되는지 알아보기

두 개의 행렬 J와 K의 곱은 다음 조건을 충족해야 함

- 1) 두 행렬의 곱 J x K에 대하여,
  행렬 J의 열의 개수 = 행렬 K의 행의 개수
- 2) 두 행렬의 곱 J x K의 결과로 나온 행렬 JK의 크기는
  J의 행의 크기와 K의 열의 크기를 가진다.

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

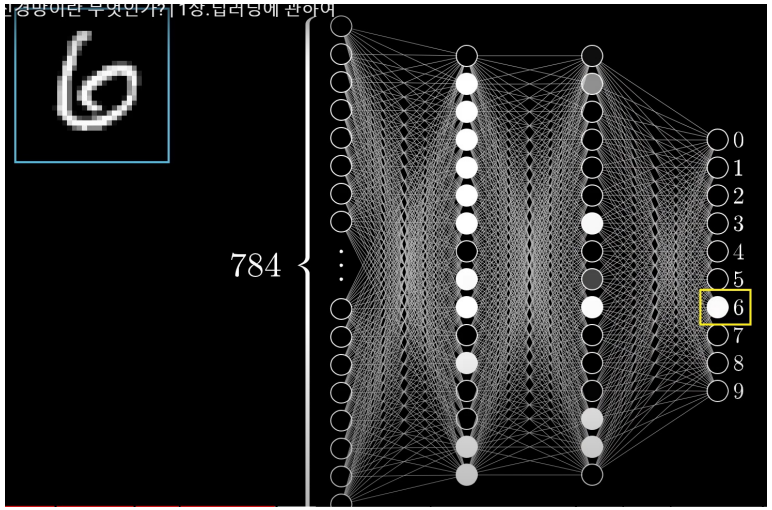
$$X_{m \times n} \times W_{n \times j} + B_{m \times j} = Y_{m \times j}$$

05 소프트맥스 회귀

다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

대표적으로 MNIST

28*28이미지 => 784차원 => 0~9의 선택지 중 하나를 고름



05 소프트맥스 회귀

다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

책의 예제에서는 **아이리스 품종 고르기**

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)	Species(y)
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
5.8	2.6	4.0	1.2	versicolor
6.7	3.0	5.2	2.3	virginica
5.6	2.8	4.9	2.0	virginica

05 소프트맥스 회귀

다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

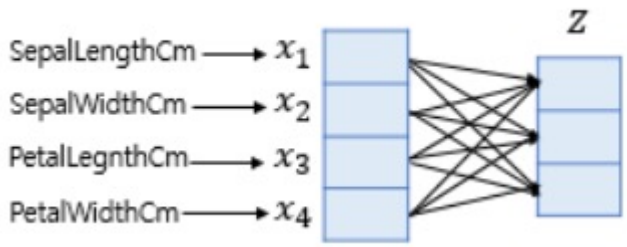
Linear함수에 시그모이드를 씌우면 0에서 1의 확률이 나옴
결과값에 각각 시그모이드를 사용하면 3개의 클래스가 각각의 확률이 나옴
결과값에 소프트맥스를 쓰면 3개의 클래스 합이 1이 됨

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

05 소프트맥스 회귀

다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

쉽게 이해하면, 각각의 결과 값을 합한 것을 분모로 만든 뒤 특정 클래스만 분자로 만듦



1 – 70
2 – 20
3 – 10

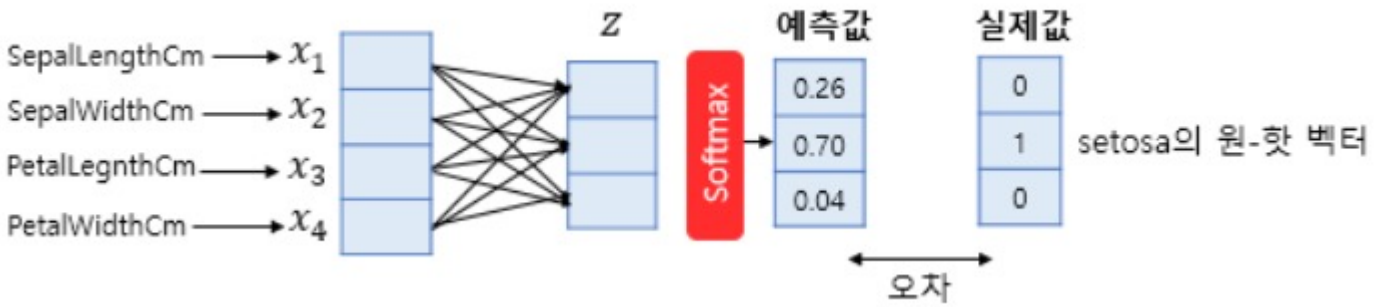
1이 나올 확률 = 70 / (70 + 20 + 10)
2이 나올 확률 = 20 / (70 + 20 + 10)
3이 나올 확률 = 10 / (70 + 20 + 10)

05 소프트맥스 회귀

다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

예측되는 확률이 나왔으면, 실제 정답과 차이를 구하기

원핫벡터를 쓰는 이유는 두번째 인덱스가 2라면
(0.26-0)^2 + (0.70-1)^2 + (0.04-1)^2 에서
(0.26-0)^2 + (0.70-1)^2 + (0.04-2)^2 로 변하게 되는데
해당 값이 변하면, 순서에 의미를 갖게 되는 문제가 있음
클래스의 표현 방법을 무작위로 가지게 됨



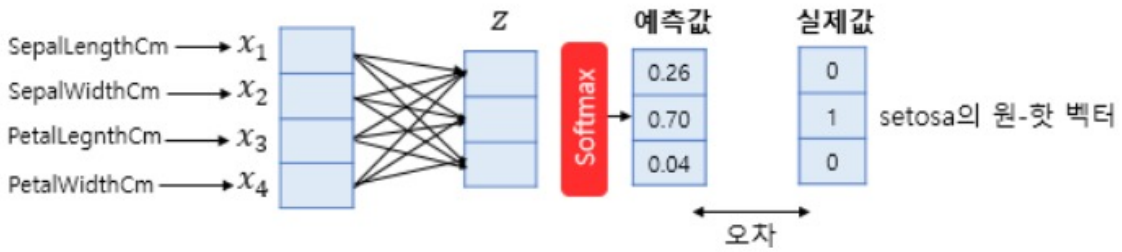
05 소프트맥스 회귀

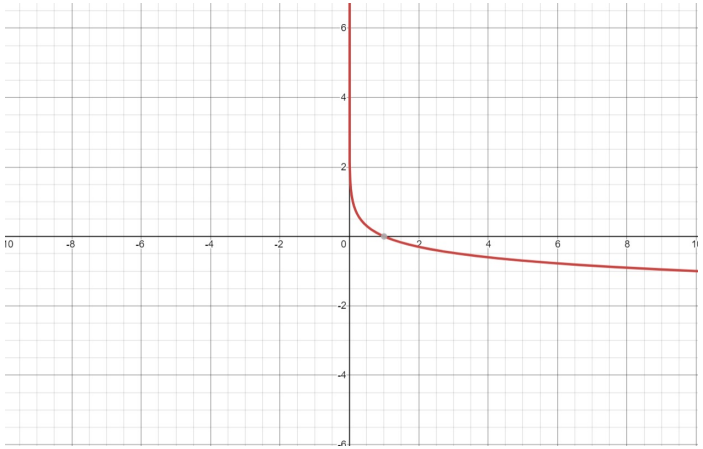
다중 클래스 분류
세 개 이상의 선택지 중 하나를 고르는 문제

비용함수로 크로스 엔트로피 함수를 사용
값이 1일 때 0의 값을 갖게 되고, 0일때 무한대의 값을 갖게 됨
⇒ 예측을 잘 하면 할 수록 비용함수가 작아짐

p(0.26)*0 + p(0.70)*1 + p(0.04)*0

오차를 극대화해서 아래와 같이 보면, 로그함수에 p(0.0001)이 매우 커지는 값을 갖게 되므로,
오차 값은 커지게 됨
p(0.0001)*1 + p(0.9997)*0 + p(0.0002)*0





05 소프트맥스 회귀

실습

1. 데이터 준비
1-1. 데이터 불러오기
1-2. 데이터 모델링
- train, valid_set으로 나눔
- 결과는 원핫벡터로 만듬

2. 딥러닝 모델 만들기

3. 학습하기

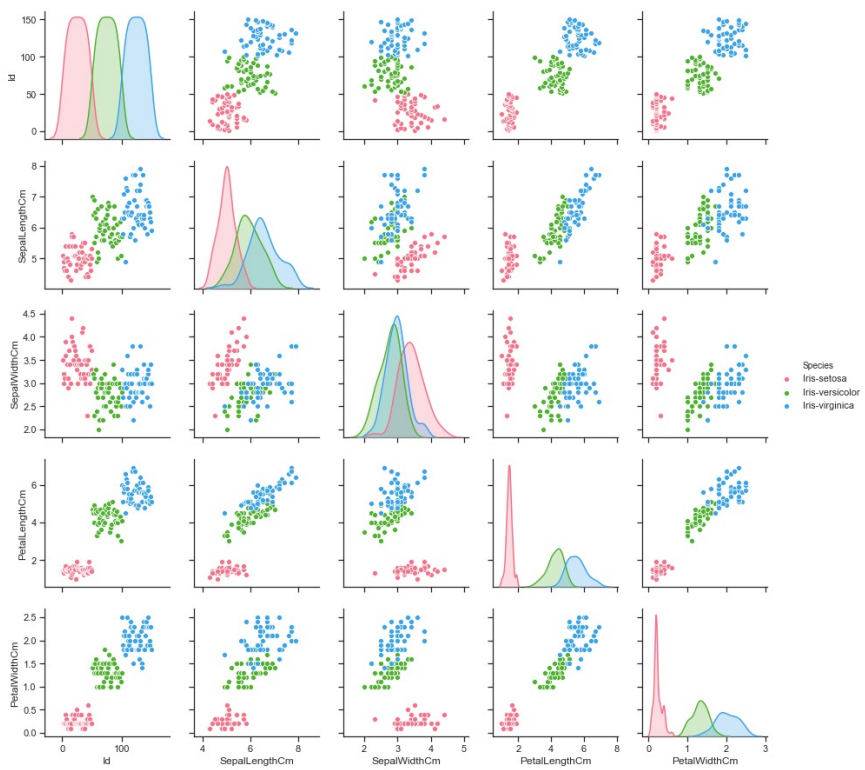
4. 테스트하기

05 소프트맥스 회귀

실습
1.1 데이터 불러오기

```
import pandas as pd
data = pd.read_csv('iris.csv',encoding='latin1')
print(data[:5])

import seaborn as sns
#del data['Id'] # 인덱스 열 삭제
sns.set(style="ticks", color_codes=True)
g = sns.pairplot(data, hue="Species", palette="husl")
```

	caseno	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa



05 소프트맥스 회귀

실습
1-2. 데이터 모델링

- **train, valid_set으로 나눔**
- **결과는 원핫벡터로 만듦**

```
(X_train, X_test, y_train, y_test) =
train_test_split(data_X, data_y, train_size=0.8,
random_state=1)

from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train) y_test =
to_categorical(y_test)

# 훈련 데이터와 테스트 데이터에 대해서 원-핫 인코딩
print(y_train[:5]) print(y_test[:5])
```

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]]
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

05 소프트맥스 회귀

실습
2. 딥러닝 모델 만들기
3. 학습하기

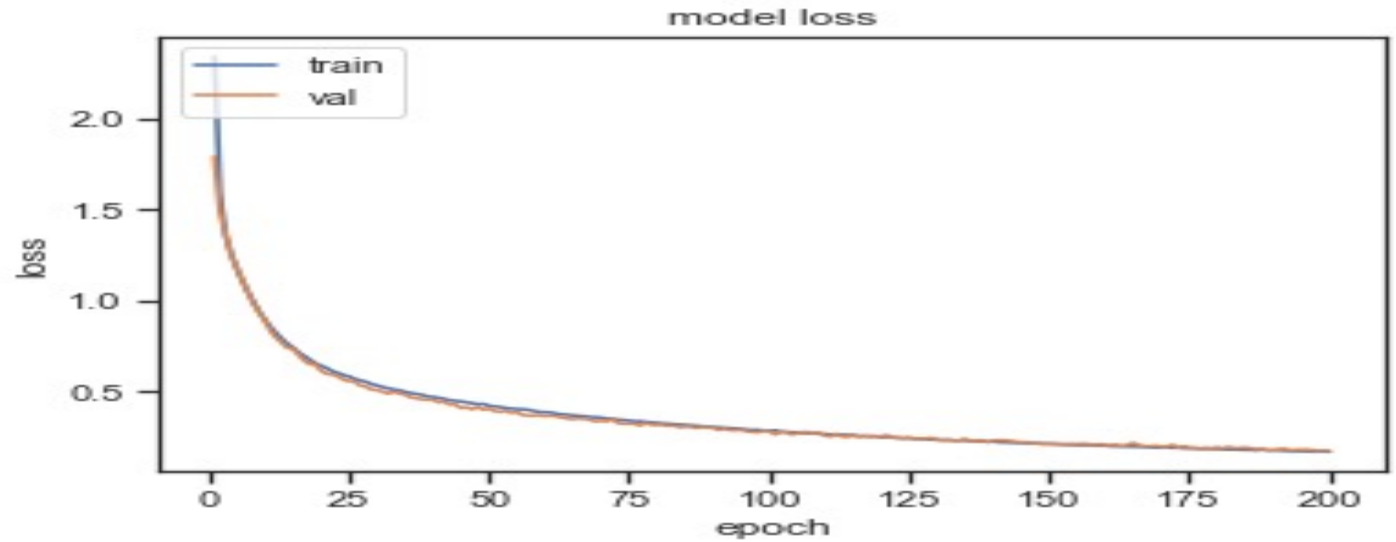
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
model=Sequential() model.add(Dense(3, input_dim=4, activation='softmax'))
sgd=optimizers.SGD(lr=0.01) model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['accuracy’])

history=model.fit(X_train,y_train, batch_size=1, epochs=200,
validation_data=(X_test, y_test)
```

```
Train on 120 samples, validate on 30 samples
Epoch 1/200
120/120 [==============================] - 0s 4ms/step - loss: 2.3404 - acc: 0.3083 - val_loss: 1.7910 - val_acc: 0.4333
... 중략 ...
Epoch 200/200
120/120 [==============================] - 0s 852us/step - loss: 0.1711 - acc: 0.9500 - val_loss: 0.1767 - val_acc: 1.0000
```

05 소프트맥스 회귀

실습
4. 테스트 하기

```
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()
```



감사합니다 ☺

집현전 Season 2
초급반 8조 주혜신, 장영찬, 차경묵