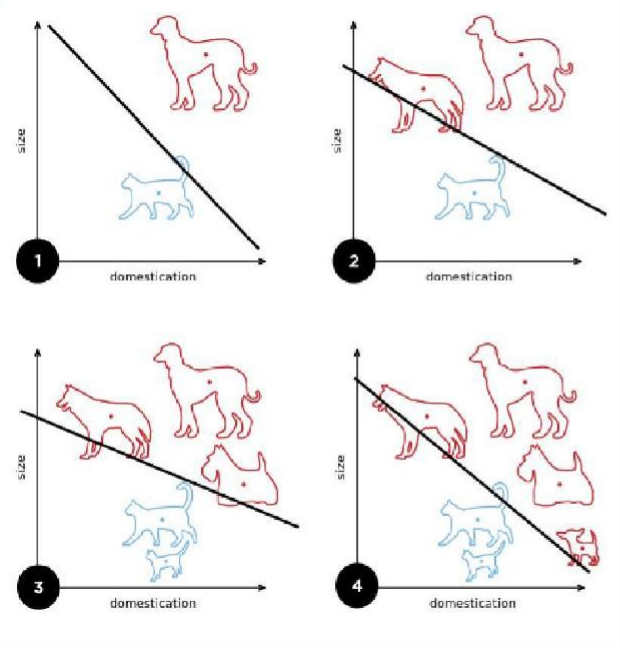
1. Logistic Regression

Recap:
Binary classification with Perceptron

- 이진분류(binary classification)



CASE 1) 시험 점수를 입력하면 합격/ 불합격
CASE 2) 환자 정보를 입력하면 수술 성공 여부
CASE 3) 이미지를 넣으면 개? 고양이?
CASE 4) ETC

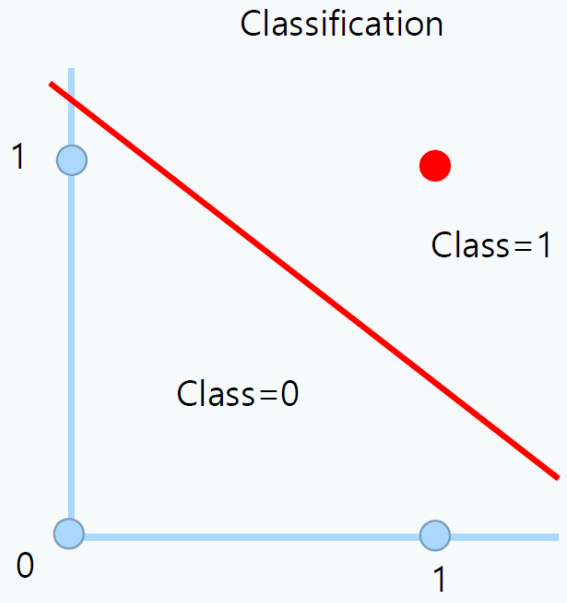
Binary classification

Recap:
Binary classification with Perceptron

- 이진분류(binary classification)
  - XOR 문제 (AND gate)

학교 수업 참여(x1)	학교 야자 참여 (x2)	합격
0	0	0
0	1	0
1	0	0
1	1	1

$$y = w_1 x_1 + w_2 x_2 + b$$



Recap:
Binary classification with Perceptron

- 이진분류(binary classification)
  - XOR 문제 (AND gate)

$$f(w_1x_1 + w_2x_2 + b) = y$$

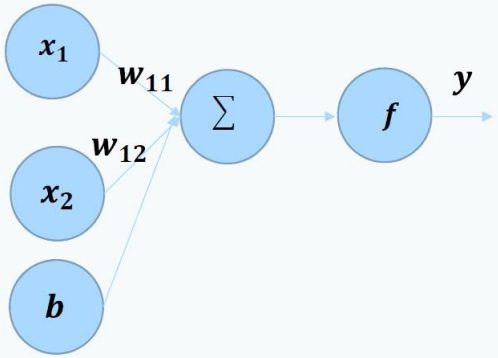
loss

$$f([\,x1 \quad x2]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = f([0 \quad 0]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = [y] \qquad ([y]-[0])^2$$

$$f([\,x1 \quad x2]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = f([0 \quad 1]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = [y] \qquad ([y]-[0])^2$$

$$f([\,x1 \quad x2]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = f([1 \quad 0]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = [y] \qquad ([y]-[0])^2$$

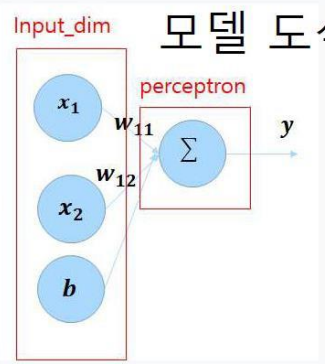
$$f([\,x1 \quad x2]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = f([1 \quad 1]\begin{bmatrix}w_{11}\\w_{12}\end{bmatrix} + \mathrm{b}) = [y] \qquad ([y]-[1])^2$$



Recap:
Binary classification with Perceptron

- 이진분류(binary classification)
  - XOR 문제 (AND gate)

```
## data 선언
x_data = [[0.,0.], [0.,1.], [1.,0.],[1.,1.]]
y_data = [[0.], [0.], [0.], [1.]]
test_data=[[0.8, 0.8]]

## tf.keras를 활용한 perceptron 모델 구현.
model = tf.keras.Sequential() ## 모델 선언
model.add(tf.keras.layers.Dense(1, input_dim=2)) # 선언된 모델에 add를통해 쌓아감.은닉층

# 모델 loss, 학습 방법 결정하기
optimizer=tf.keras.optimizers.SGD(lr=0.001) ### 경사 하강법으로 global min 에 찾아가는 최적화 방법 선언.
loss=tf.keras.losses.mse  ## 예측값 과 정답의 오차값 정의. mse는 mean squre error로 (예측값 - 정답)^2 를 의미
metrics=tf.keras.metrics.mean_squared_error ### 학습하면서 평가할 메트릭스 선언언

# 모델 컴파일하기
model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])

# 모델 동작하기
model.fit(x_data, y_data, epochs=1000, batch_size=4)

# 결과를 출력합니다.
print(" test data [0.8, 0.8] 예측 값 : ", model.predict(test_data))
```



모델 도식도

모델 summury

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

Batch_size=4

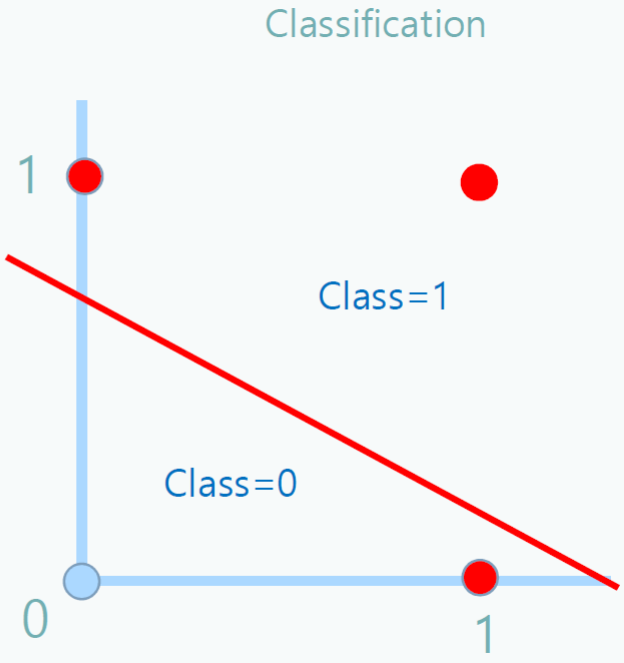
W11, w12, b

Recap:
Binary classification with Perceptron

- 이진분류(binary classification)
  - XOR 문제 (OR gate)

학교 수업 참여(x1)	학교 야자 참여 (x2)	합격
0	0	0
0	1	1
1	0	1
1	1	1

$$y = w_1 x_1 + w_2 x_2 + b$$



Recap:
Binary classification with Perceptron

- 이진분류(binary classification)
  - XOR 문제 (OR gate)

$$f(w_1 x_1 + w_2 x_2 + b) = y$$

$$f([\,x1 \quad x2]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = f([0 \quad 0]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = [y]$$

$$f([\,x1 \quad x2]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = f([0 \quad 1]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = [y]$$

$$f([\,x1 \quad x2]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = f([1 \quad 0]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = [y]$$

$$f([\,x1 \quad x2]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = f([1 \quad 1]\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \mathrm{b}) = [y]$$

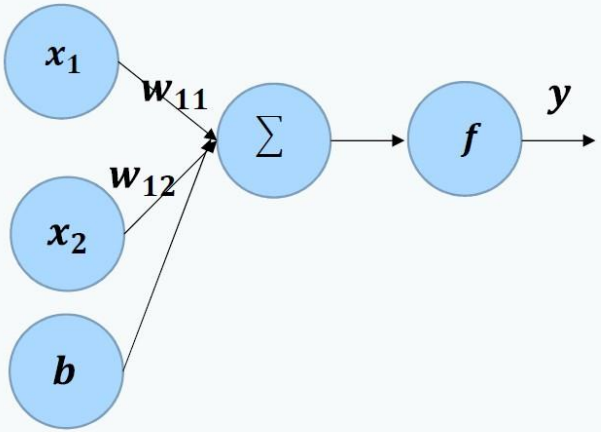
loss

$$([y]-[0])^2$$

$$([y]-[1])^2$$

$$([y]-[1])^2$$

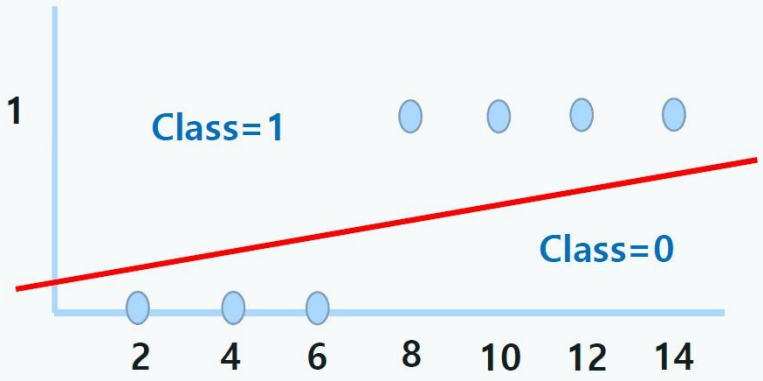
$$([y]-[1])^2$$



Logistic Regression

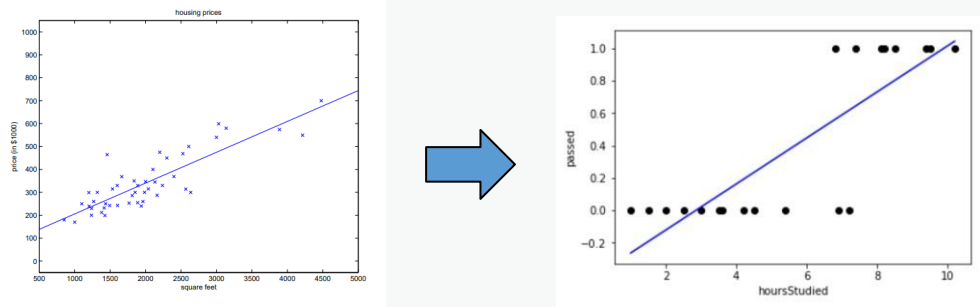
- Logistic regression with Perceptron

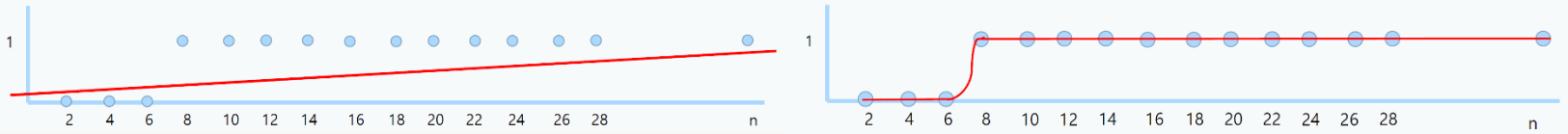
공부시간 (x)	합격 여부(y)
2	불합격 -> 0
4	불합격 -> 0
6	불합격 -> 0
8	합격 -> 1
10	합격 -> 1
12	합격 -> 1
14	합격 -> 1

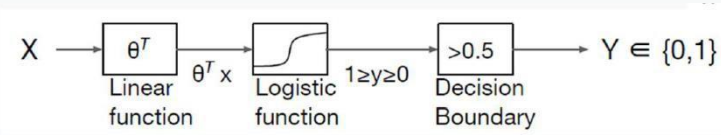


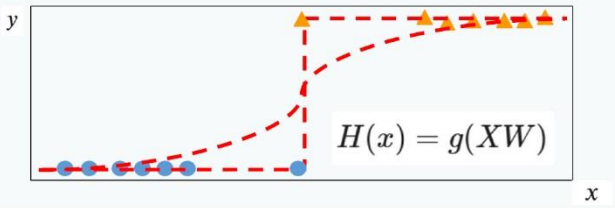
Logistic Regression

- Logistic regression with Perceptron
  - linear regression algorithm to try to predict y given x
  - To classification, changing the form for our hypotheses $h_\theta(x)$



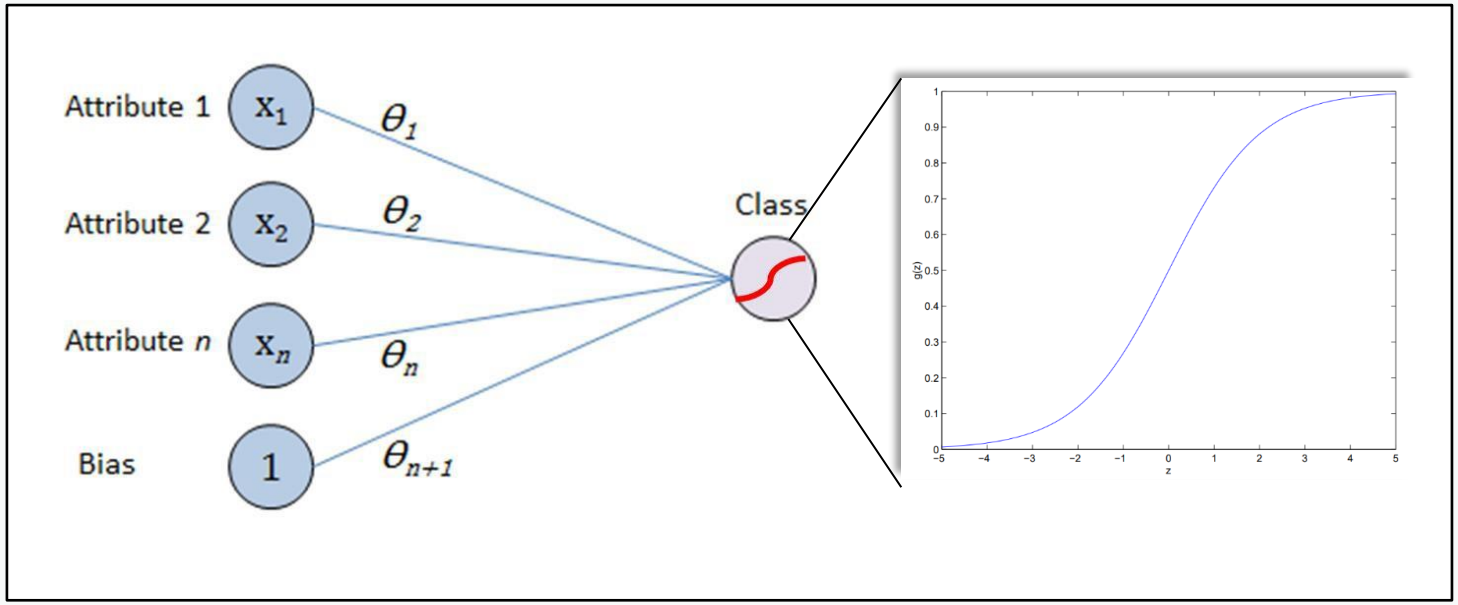






Logistic Regression

- Logistic function (sigmoid function)
  - $g(s) = \frac{1}{1+e^{-s}} = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$
  - $\text{sigmoid}(h_\theta(x)) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n + \theta_{n+1} b$



Logistic Regression

- Odds ratio
  - $h(x) = \sigma_{i=0}^{n} \theta_i x_i = \theta^T x$ (linear regression)
  - $P(Y = 1|X = x) = \theta^T x$ (classification problem)

$$\boldsymbol{odds} = \frac{\boldsymbol{P(A)}}{\boldsymbol{1 - P(A)}} \rightarrow applying\ odds\ to\ left\ hand\ side$$

- $\frac{P(Y=1|X=x)}{1-P(Y=1|X=x)} = \theta^T x \rightarrow \log_e \frac{P(Y=1|X=x)}{1-P(Y=1|X=x)} = \theta^T x$
- $\frac{P(Y=1|X=x)}{1-P(Y=1|X=x)} = \boldsymbol{e^s}, where\ s = \theta^T x$

$a^{\log_a b} = t$

$\log_a a^{\log_a b} = \log_a t$

$\log_a b \times \log_a a = \log_a t$

$\log_a b = \log_a t \quad (\because \log_a a = 1)$

$b = t$

$b = a^{\log_a b} \quad (\because t = a^{\log_a b})$

Logistic Regression

- Odds ratio
  - $\frac{P(Y=1|X=x)}{1-P(Y=1|X=x)} = e^s$
  - $P(Y=1|X=x) = e^s\left(1-P(Y=1|X=x)\right)$
    $= e^s - e^s P(Y=1|X=x)$
  - $P(Y=1|X=x)(1+e^s) = e^s$
  - $\boldsymbol{P(Y=1|X=x) = \frac{e^s}{(1+e^s)} \rightarrow \frac{e^s * e^{-s}}{(1+e^s)*e^{-s}}}$
  - $\boldsymbol{= \frac{1}{1+e^{-s}}}$

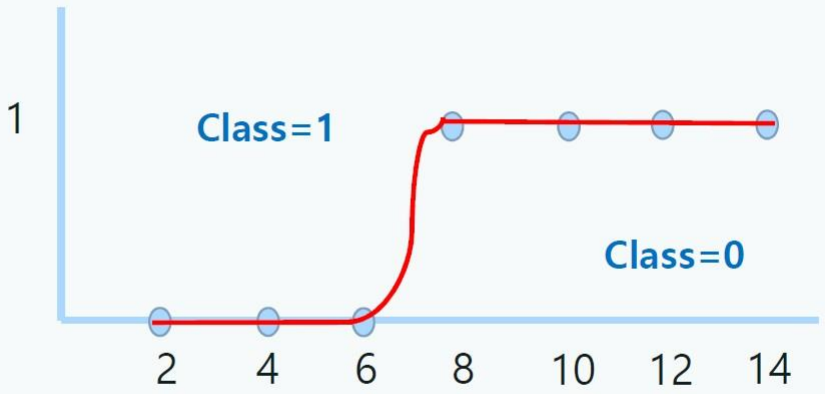
Logistic Regression

- Decision boundary of Binary Logistic Regression
  - $P(Y = 1|X = x) > P(Y = 0|X = x)$
  - ➔ $P(A) > 1 - P(A)$
  - ➔ $\frac{P(A)}{1-P(A)} > 1$ ➔ $\log \frac{P(A)}{1-P(A)} > 0$
  - ➔ $\boldsymbol{\theta^T x > 0}$
- Classification
  - $y = \frac{1}{1+e^{-\theta^T x}}, \begin{bmatrix} y \to 1, & if\ \theta^T x \to \infty \\ y \to \frac{1}{2}, & if\ \theta^T x \to 0 \\ y \to 0, & if\ \theta^T x \to -\infty \end{bmatrix}$

Practice 1:
Logistic Regression

- 로지스틱 회귀

공부시간 (x)	합격 여부(y)
2	불합격 -> 0
4	불합격 -> 0
6	불합격 -> 0
8	합격 -> 1
10	합격 -> 1
12	합격 -> 1
14	합격 -> 1



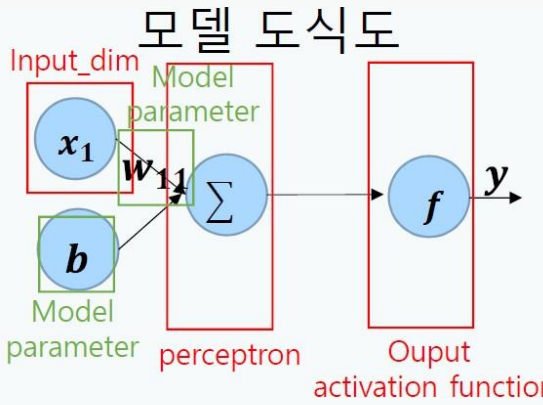
Practice 1:
Logistic Regression

- 로지스틱 회귀

```python
## data 선언
x_data = [[2.], [4.], [6.], [8.], [10.], [12.], [14.]]
y_data = [[0.], [0.], [0.], [1.], [1.], [1.], [1.]]
test_data= [[3.]]
test_data2= [[7.]]
test_data3= [[17.]]
test_data4= [[60.]]

## tf.keras를 활용한 perceptron 모델 구현.
model = tf.keras.Sequential() ## 모델 선언
model.add(tf.keras.layers.Dense(1, input_dim=1, activation='sigmoid')) # 선언된 모델에 add를 통해 쌓아감. 은닉층
model.summary()

# 모델 loss, 학습 방법 결정하기
optimizer=tf.keras.optimizers.SGD(learning_rate=0.01)   ### 경사 하강법으로 global min 에 찾아가는 최적화 방법 선언.
loss=tf.keras.losses.mean_squared_error   ## 예측값 과 정답의 오차값 정의. mse는 mean squre error로 (예측값 - 정답)^2
metrics=tf.keras.metrics.binary_accuracy  ### 학습하면서 평가할 메트릭스 선언언

# 모델 컴파일하기 - 모델 및 loss 등 구조화한 모델을 컴퓨터가 실제 작동할 수 있도록 변환
model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])

# 모델 동작하기
model.fit(x_data, y_data, epochs=1500, batch_size=2)

# 결과를 출력합니다.
print(" test data [3.] 예측 값 : ", model.predict(test_data))
print(" test data [7.] 예측 값 : ", model.predict(test_data2))
print(" test data [17.] 예측 값 : ", model.predict(test_data3))
print(" test data [60.] 예측 값 : ", model.predict(test_data4))
```

모델의 평가 방법

모델 도식도



모델 summury

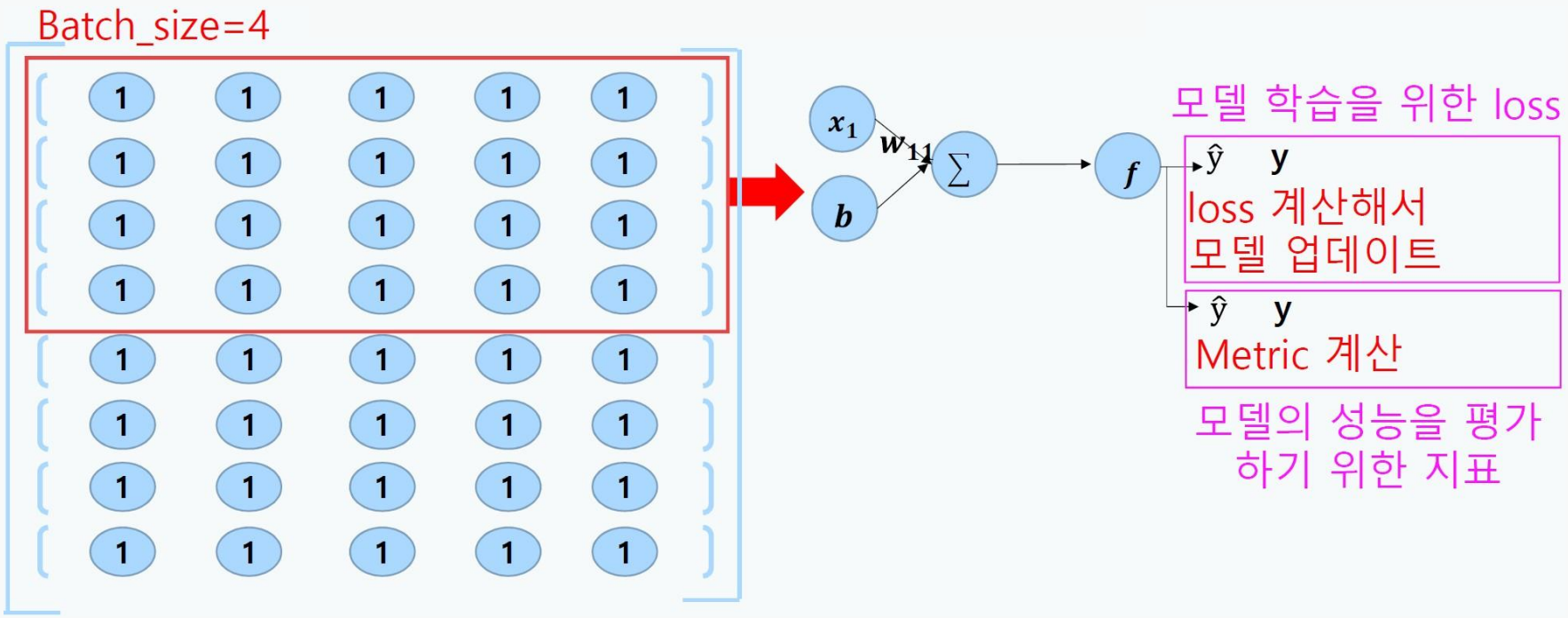
```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable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

Train on 7 samples
Epoch 1/1000
```

Batch size

W11, b

Practice 1:
Logistic Regression

- 로지스틱 회귀
  - 배치학습 및 학습과정 예시



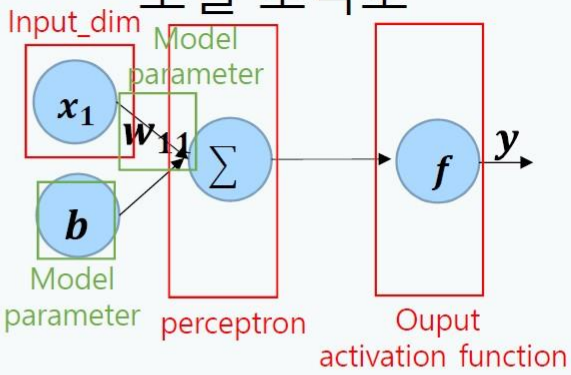
Practice 1:
Logistic Regression

- 로지스틱 회귀

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
Train on 7 samples
Epoch 1/1000
```

Batch_size

모델 도식도




```
x_data = [[2.], [4.], [6.], [8.], [10.], [12.], [14.]]
y_data = [[0.], [0.], [0.], [1.], [1.], [1.], [1.]]
```

```
x_data = [[2.], [4.], [6.], [8.], [10.], [12.], [14.]]
y_data = [[0.], [0.], [0.], [1.], [1.], [1.], [1.]]
```

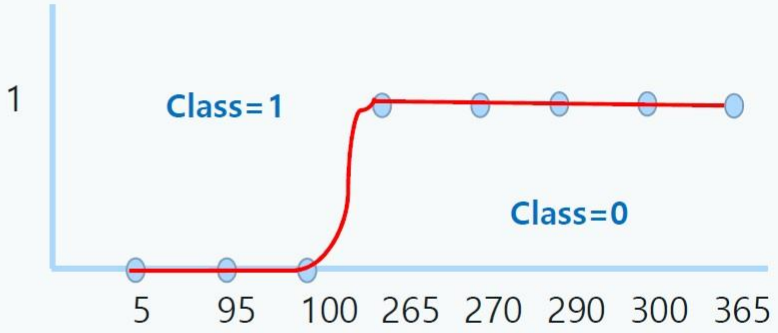
데이터 입력 차원 : (4, 1)

Perceptron (node) (3, 1) Sigmoid

Exercise 1:
Logistic Regression

- 로지스틱 회귀
  - 학습이 어려움

공부 일수	합격 여부(y)
5	불합격 -> 0
95	불합격 -> 0
100	불합격 -> 0
265	합격 -> 1
270	합격 -> 1
290	합격 -> 1
300	합격 -> 1
365	합격 -> 1



Loss Function:
Cross Entropy

- 분류를 위한 손실함수

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

시그모이드 함수의 특징을 보면 y값은 0~1사이 값
정답은 0 or 1이다.

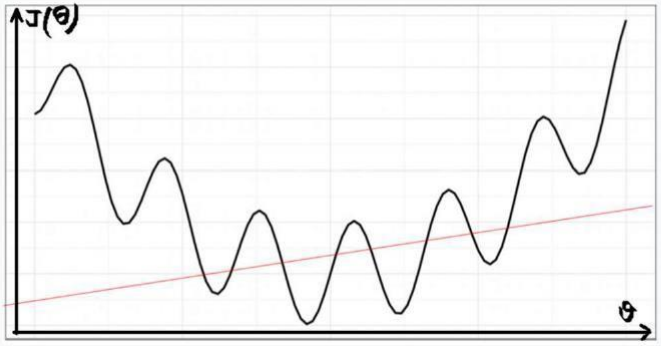
Case1) 실제 값이 1 일 때
예측 값이 0에 가까워지면 오차가 커져야 한다.

Case2) 실제 값이 0 일 때
예측 값이 1에 가까워져도 오차는 커져야 한다.

Loss Function: Cross Entropy

- 분류 문제에서 MSE의 문제
  - Non-convex loss function

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

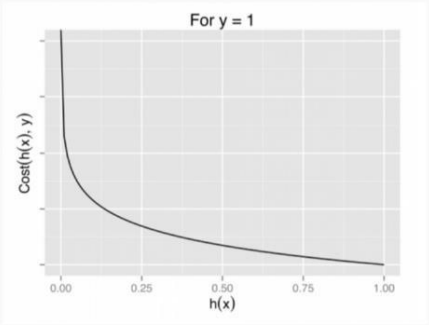


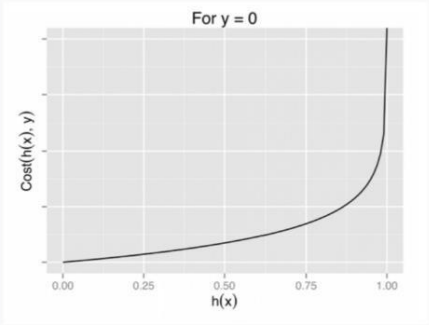
Non-convex function

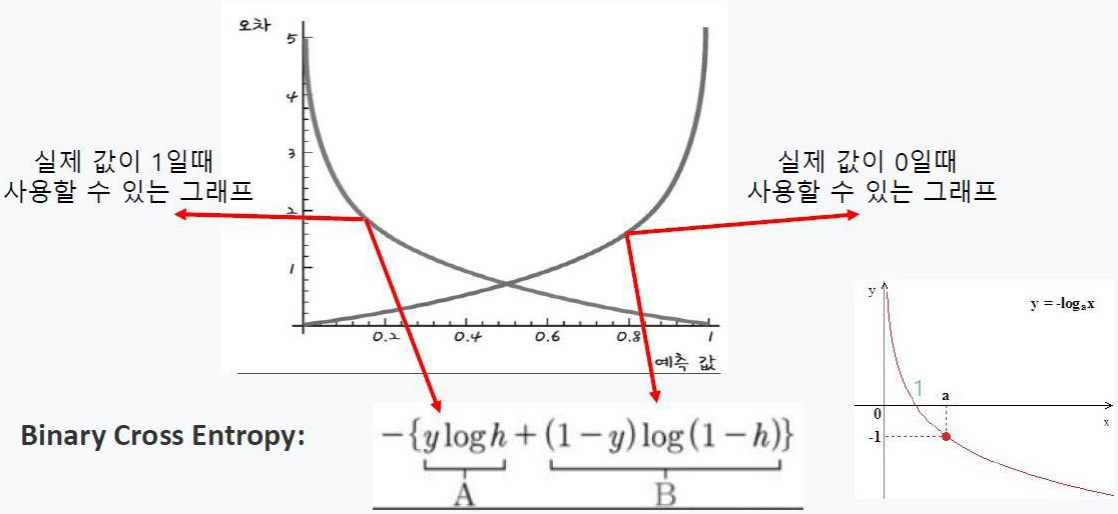
Loss Function

- 분류를 위한 손실함수: Cross Entropy

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

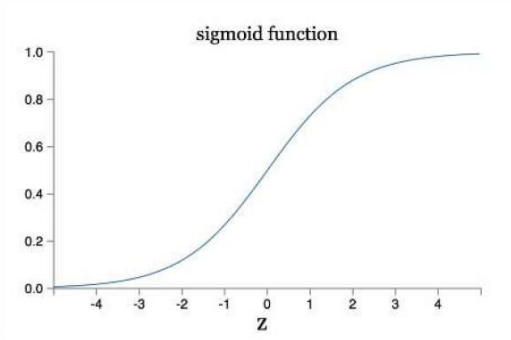






Loss Function: Cross Entropy

- Logistic Regression with CE loss
  - Mean Squared Error

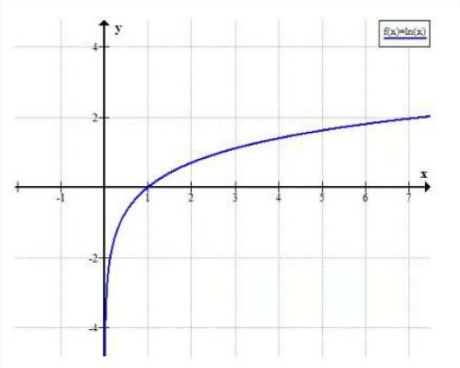


classification

$$\text{cost} = (\sigma(\hat{y}) - y)^2$$

미분하면 $2\sigma(\hat{y})(\sigma(\hat{y}))'$
즉, 미분하면 1과 0에 수렴하는
부분은 굉장히 작은 값으로 나타내어
진다. 그러므로 학습이 느려짐.

  - Cross Entropy loss



$$\text{cost} = (\sigma(\hat{y}) - y)^2$$

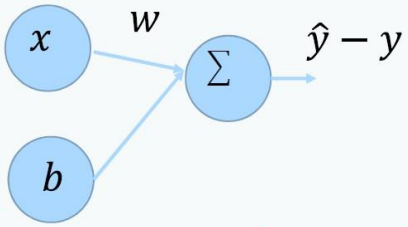
$$\text{cost} = -[y ln\sigma(\hat{y}) + (1-y)\ln(1-\sigma(\hat{y}))]$$

성질

1. 예측 값인 $\sigma(\hat{y})$은 0~1사이 값. 그러므로 항상 음의 값만 존재
2. $\sigma(\hat{y})$이 y에 가까운 값일 수록 0에 가까워 짐.

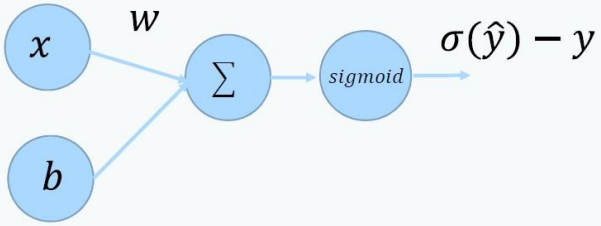
Loss Function: Cross Entropy

- Logistic Regression with CE loss



regression

cost = $(\hat{y} - y)^2$



Logistic regression(classification)

cost = $(\sigma(\hat{y}) - y)^2$

$$\text{cost} = -[y ln\sigma(\hat{y}) + (1-y)\ln(1-\sigma(\hat{y}))]$$

cost 미분 하면 $-[y\frac{1}{\sigma(\hat{y})} - (1-y)\frac{1}{1-\sigma(\hat{y})}]\sigma(\hat{y})'$ ← 대입

Sigmoid 함수$\sigma(\hat{y})$를 미분하면 $\sigma(\hat{y})' = \sigma(\hat{y})(1-\sigma(\hat{y}))$

cost의 미분값은 $-[y\frac{1}{\sigma(\hat{y})} - (1-y)\frac{1}{1-\sigma(\hat{y})}]\sigma(\hat{y})'$

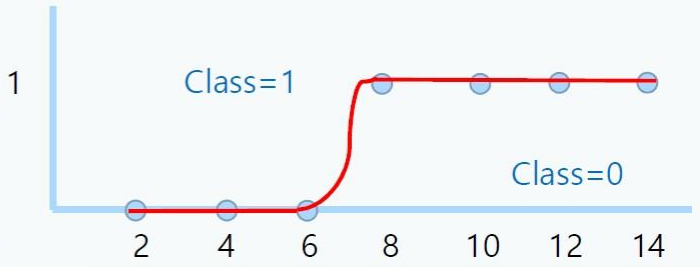
$= -[y - \sigma(\hat{y})]$

즉, 정답과 예측의 차이가 클수록 미분의 값은 커지므로 학습이 잘됨

Practice 2:
Cross Entropy

- Logistic Regression with CE loss
  - Activation, CE loss 적용

공부시간 (x)	합격 여부(y)
2	불합격 -> 0
4	불합격 -> 0
6	불합격 -> 0
8	합격 -> 1
10	합격 -> 1
12	합격 -> 1
14	합격 -> 1



practice : P_02_02_tensor_cross_entropy.py

Exercise 2:
Cross Entropy

- Logistic Regression with CE loss
  - Activation, CE loss 적용

공부 일수	합격 여부(y)
5	불합격 -> 0
30	불합격 -> 0
95	불합격 -> 0
100	불합격 -> 0
265	합격 -> 1
270	합격 -> 1
290	합격 -> 1
300	합격 -> 1
365	합격 -> 1

Perceptron Summary

- Linear Regression with MSE
- Logistic Regression with MSE or Cross Entropy

1. Linear Regression

$$\mathcal{L}_{mse} = \frac{1}{n}\sum_{i=1}^{n}(H(x_i) - y_i)^2$$

-> 문제점 : binary classification 에서 사용할 시 플리지 않을때가 존재함.

---

2. Logistic Linear Regression (Sigmoid만 적용)

$$\mathcal{L}_{mse} = \frac{1}{n}\sum_{i=1}^{n}(\sigma(H(x_i)) - y_i)^2$$

-> 해결점 : 출력할때 sigmoid 함수를 통해서 출력 0~1사이 값 이며 커브피팅 효과
-> 문제점 : 기존 mse loss를 사용하면 두가지 문제가 있음 1. local min에 빠질 확률이 크고,
2. 시그모이드 미분값으로 w 학습이 잘안되거나 느림

---

3. Logistic Regression(Classification)

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

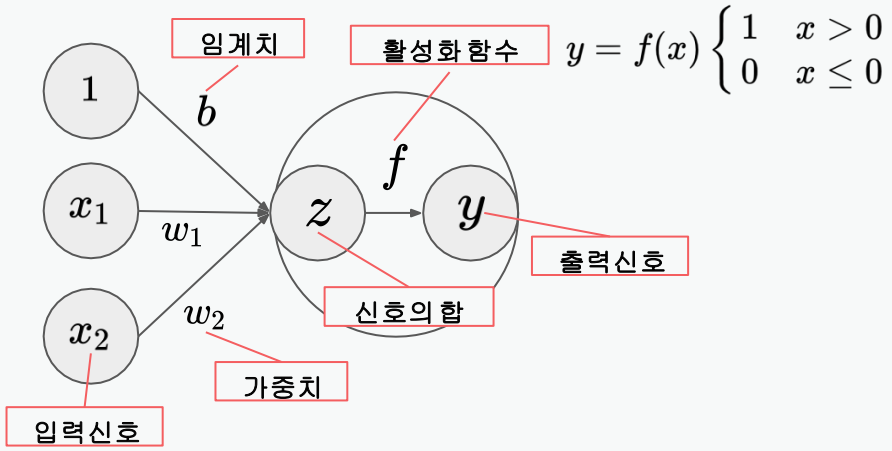
-> 해결점 : mse loss 함수 대신 binary_crossentropy loss

2. Feed-forward Neural Networks

Logic Gates

- Logic gates (논리회로)
  - 제일 간단한 퍼셉트론으로 해결 가능
  - 두개의 입력 신호값을 받고 하나의 신호로 출력
  - 입력/출력 신호값은 0 혹은 1

$$y = \begin{cases} 1 & w_1 x_1 + w_2 x_2 + b > 0 \\ 0 & w_1 x_1 + w_2 x_2 + b \leq 0 \end{cases}$$



Logic Gates

- AND 구현하기
  - 모든 입력신호가 1인 경우만 활성화 (출력신호=1)
  - 가중치와 임계치를 정해보자!
- Example)
  - w1 = 0.5, w2=0.5, b=-0.7
  - z = 0.5 x 0 + 0.5 x 0 - 0.7 = -0.7
  - y = f(z) = 0 (-0.7 ≤ 0 이기 때문)
- 가중치 = 매개변수
  - 매개변수의 조정을 통해서 다른 결과 값을 도출

$$y = \begin{cases} 1 & w_1 x_1 + w_2 x_2 + b > 0 \\ 0 & w_1 x_1 + w_2 x_2 + b \leq 0 \end{cases}$$

x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

AND **진리표**

Logic Gates

- NAND, OR 구현
  - NAND: 모든 입력 신호가 1인 경우만 비활성화(출력 신호=0)
  - OR: 모든 입력 신호가 0인 경우만 비활성화(출력 신호=0)

x1	x2	y
0	0	1
1	0	1
0	1	1
1	1	0

NAND **진리표**

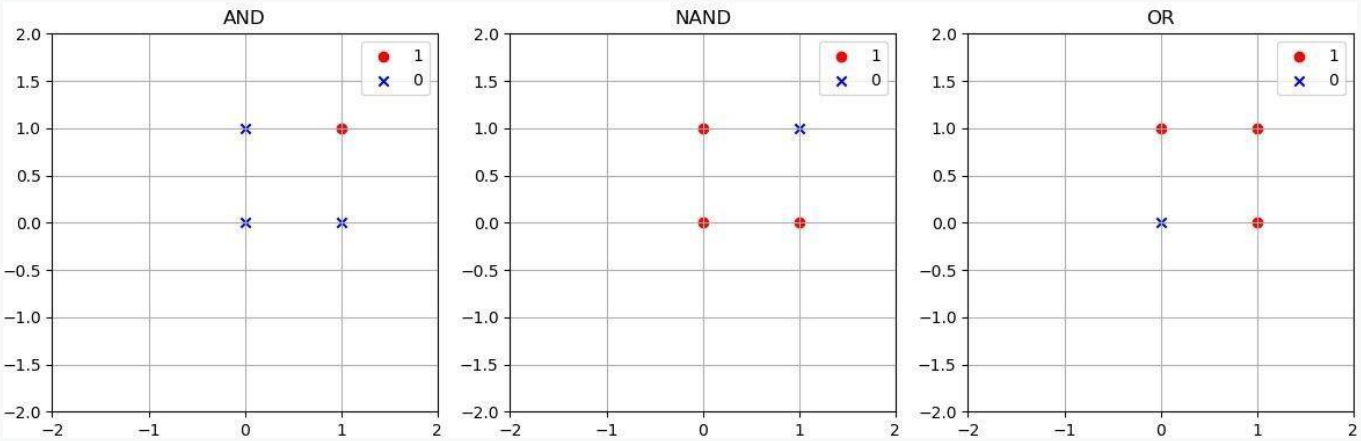
x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	1

OR **진리표**

Logic Gates

- 논리 회로의 기하적 의미
  - 평면위에 점을 두 가지 클래스(0 혹은 1)를 분류하는 것
  - 수식 변형:

$$w_1 x_1 + w_2 x_2 + b = 0 \quad \longrightarrow \quad x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

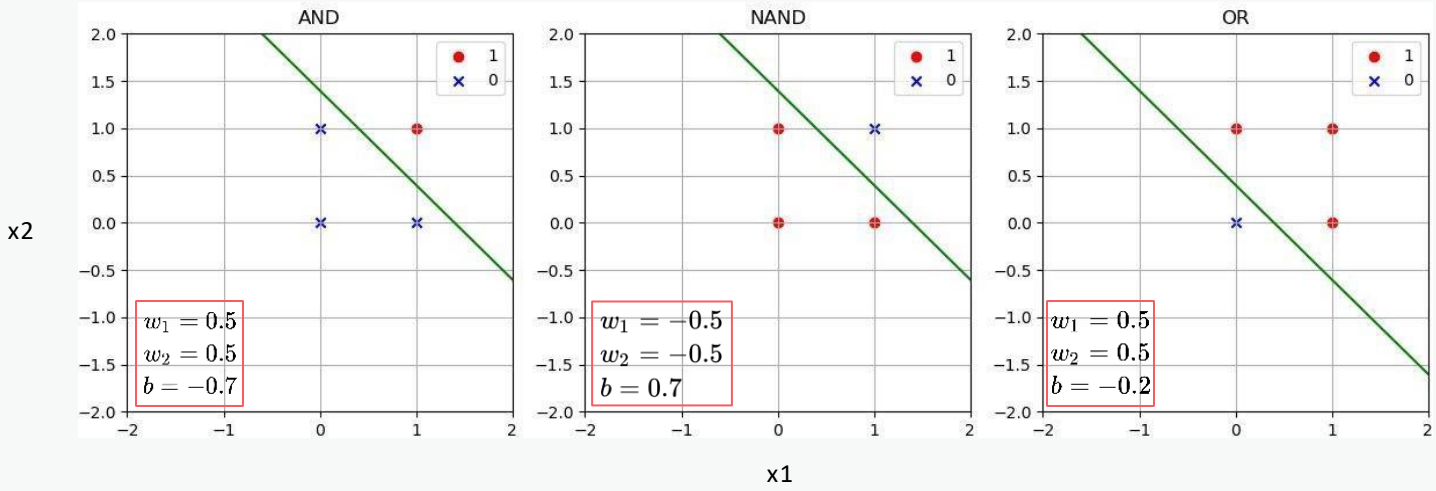


Logic Gates

- 논리 회로의 기하적 의미
  - 평면위에 점을 두 가지 클래스(0 혹은 1)를 분류하는 것
  - 신호의 합(z)은 평면위에 어떤 점을 의미

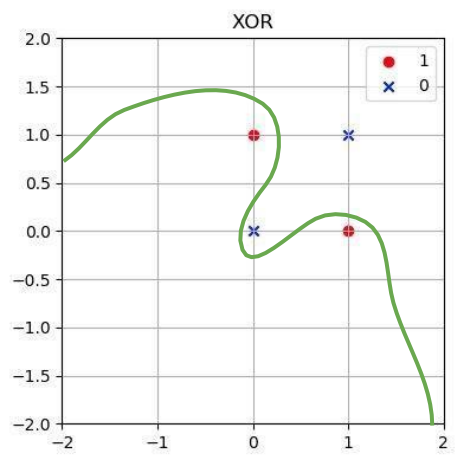
$$z = w_1 x_1 + w_2 x_2 + b$$

  - 하나의 선으로 두가지 클래스 분류가능! = **선형분류기 (linear separable function)**



XOR Problem

- XOR 문제
  - 모든 입력 신호가 0 혹은 1 경우만 비활성화(출력 신호=0)
  - 퍼셉트론으로 분류가 가능할까(하나의 직선으로 두 가지 클래스 분류가능)?
- Limitation of perceptron
  - XOR 문제 해결 불가능

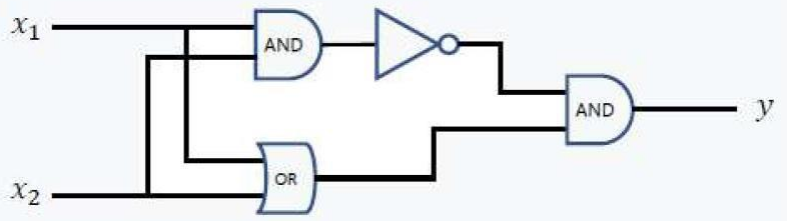


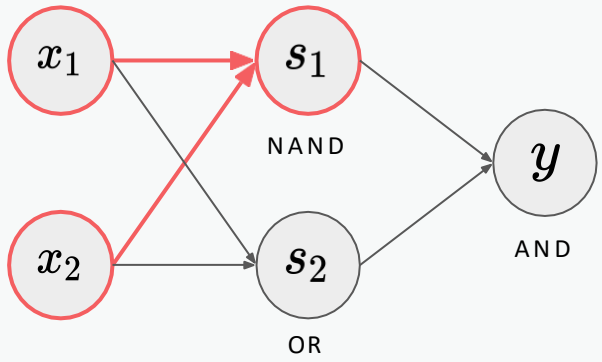
x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR **진리표**

XOR Problem

- XOR 문제 해결법
  - 여러 개의 퍼셉트론을 쌓아서 해결해보자



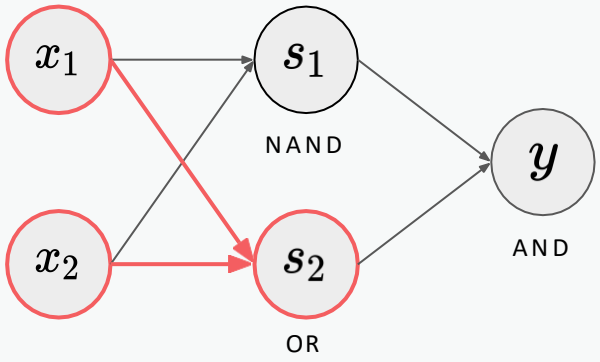


x1	x2	s1	s2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

XOR 진리표

XOR Problem

- XOR 문제 해결법
  - 여러 개의 퍼셉트론을 쌓아서 해결해보자

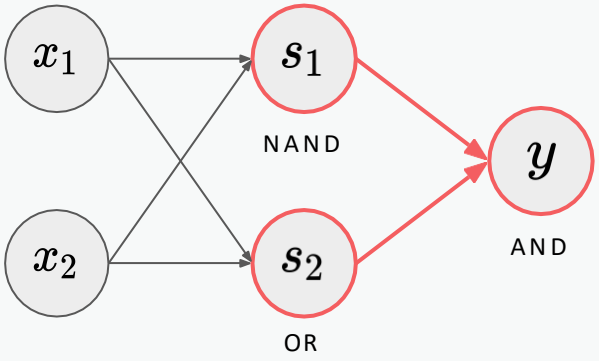


x1	x2	s1	s2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

XOR **진리표**

XOR Problem

- XOR 문제 해결법
  - 여러 개의 퍼셉트론을 쌓아서 해결해보자

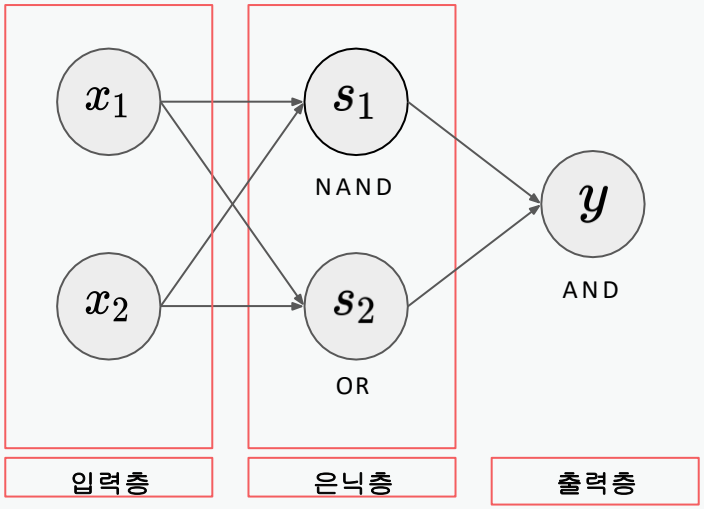


x1	x2	s1	s2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

XOR **진리표**

Multi-layer Perceptron

- 다층 퍼셉트론
  - 여러 개의 퍼셉트론을 쌓아서 **여러 함수의 조합**으로 분류를 수행

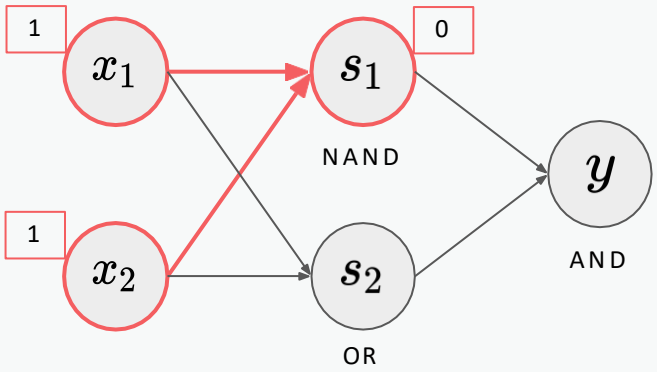


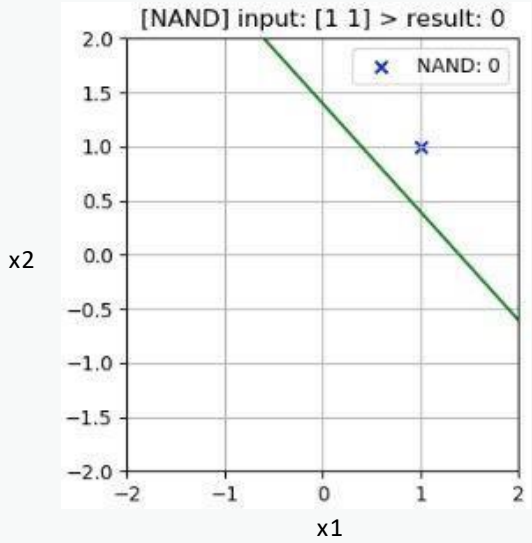
x1	x2	s1	s2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

XOR **진리표**

Multi-layer Perceptron

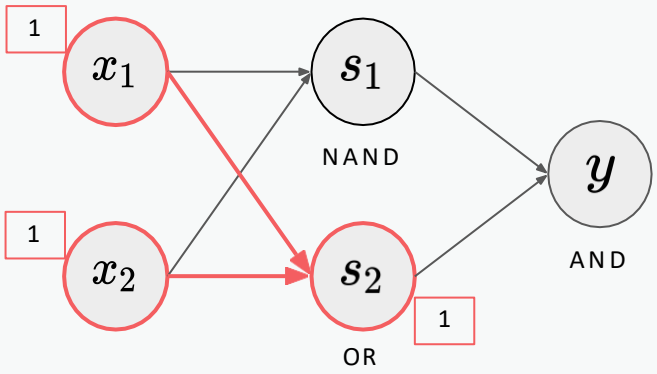
- “여러 함수의 조합” 어떤 의미?
  - ex: 입력 신호 = (1, 1), 목표 출력 = 0

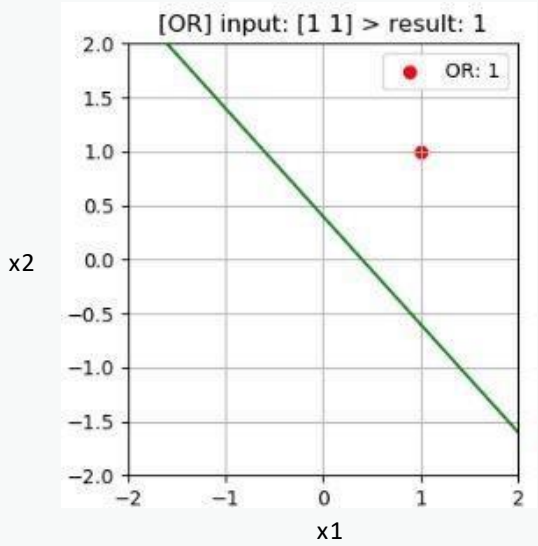




Multi-layer Perceptron

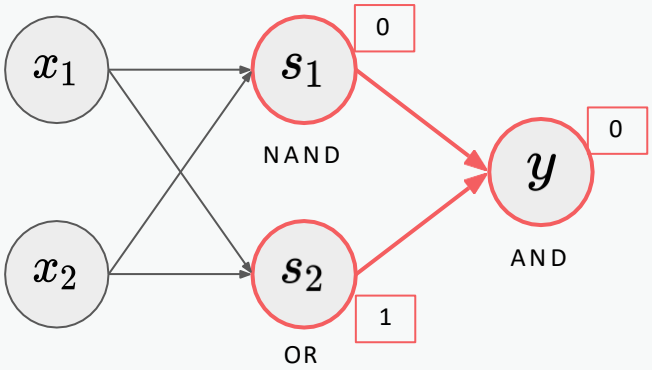
- “여러 함수의 조합” 어떤 의미?
  - ex: 입력 신호 = (1, 1), 목표 출력 = 0

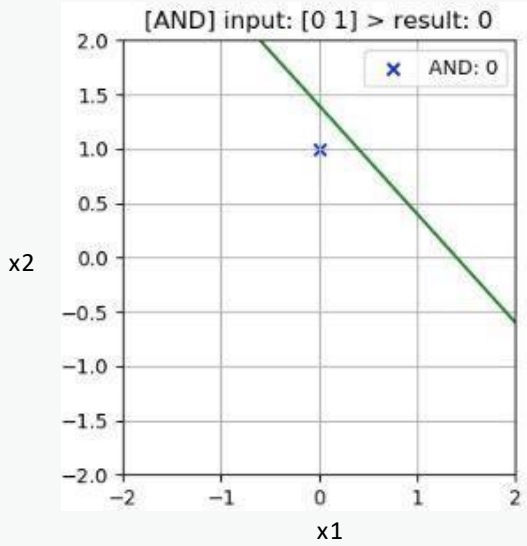




Multi-layer Perceptron

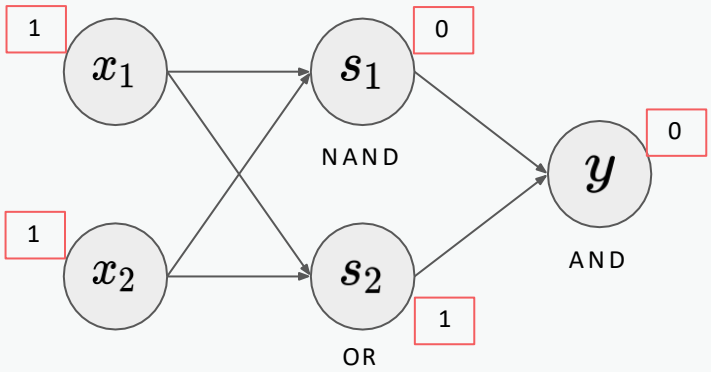
- “여러 함수의 조합” 어떤 의미?
  - ex: 입력 신호 = (1, 1), 목표 출력 = 0

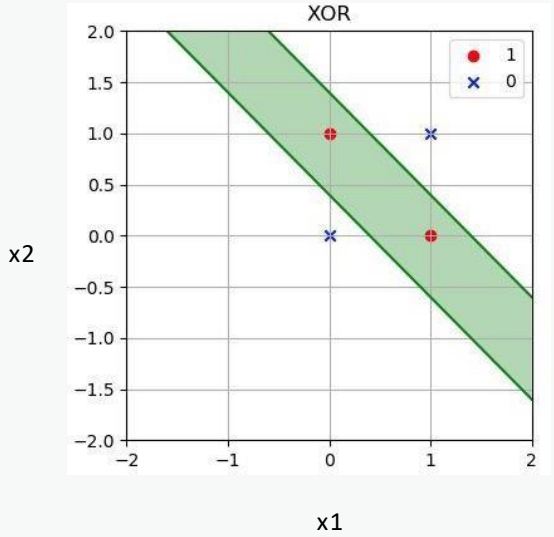




Multi-layer Perceptron

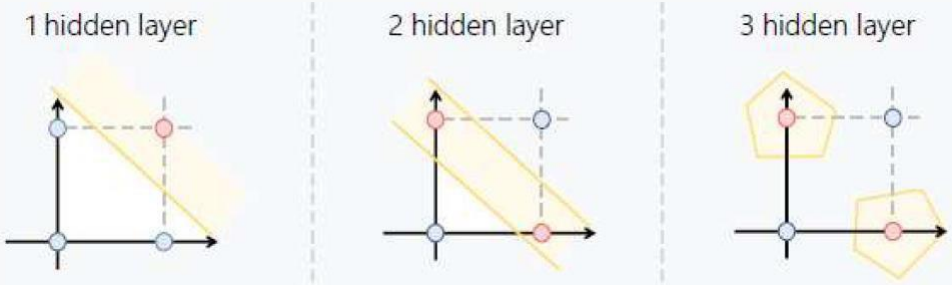
- “여러 함수의 조합” 어떤 의미?
  - 전체 과정: 3개의 함수(NAND, OR, AND)로 비선형적인 판별 경계를 구현
  - 은닉층의 역할: 입력 features를 새로운 features로 표현



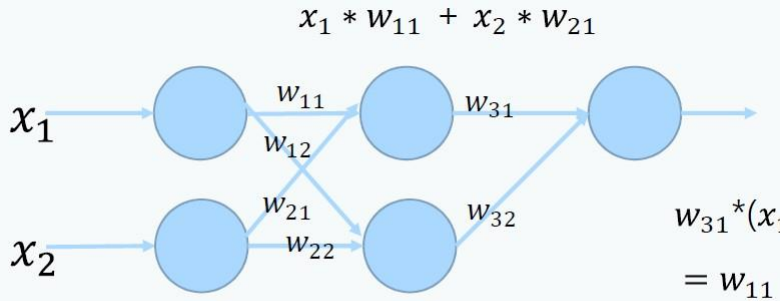


Multi-layer Perceptron

- Layer의 개수에 따라 결정할 수 있는 영역
  - 적층 구조의 신경망을 통해 비선형의 데이터를 선형 분리 가능



  - 신경망의 비선형성?



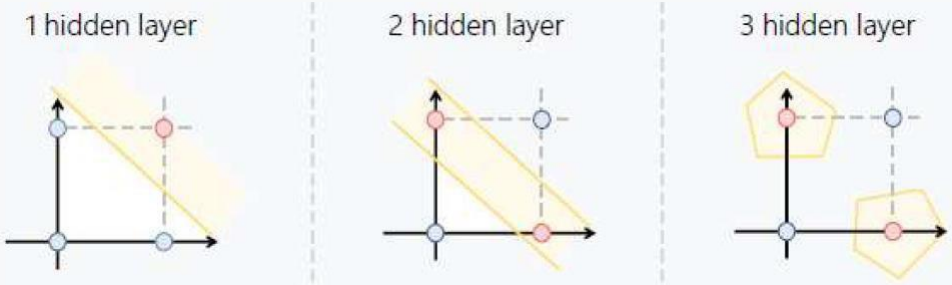
$$w_{31}{}^*(x_1 * w_{11} + x_2 * w_{21}) + w_{31}{}^*(x_1 * w_{21} + x_2 * w_{22})$$

$$= w_{11} * w_{21} * w_{31} * w_{31} * x_1 + w_{21} * w_{22} * w_{31} * w_{31} * x_1$$

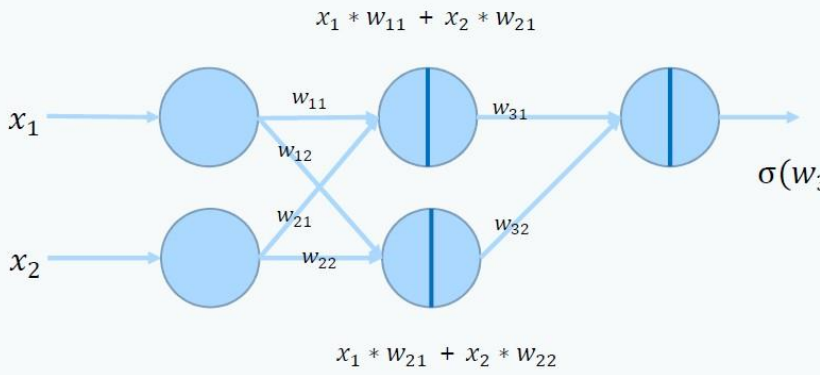
$$= W1 * x_1 + W2 * x_2$$

Multi-layer Perceptron

- Layer의 개수에 따라 결정할 수 있는 영역
  - 적층 구조의 신경망을 통해 비선형의 데이터를 선형 분리 가능



  - 신경망의 비선형성?



$$\sigma(w_{31}{}^{*}\sigma(x_1 * w_{11} + x_2 * w_{21})) + \sigma(w_{31}{}^{*}\sigma(x_1 * w_{21} + x_2 * w_{22}))$$

activation function:
sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

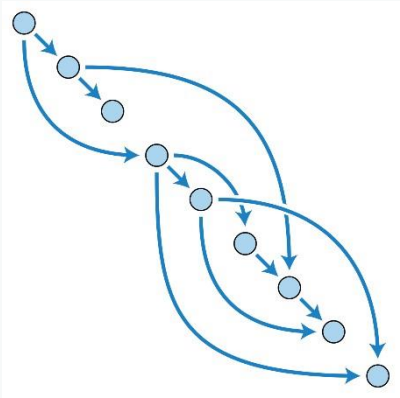
Feed-forward Neural Network

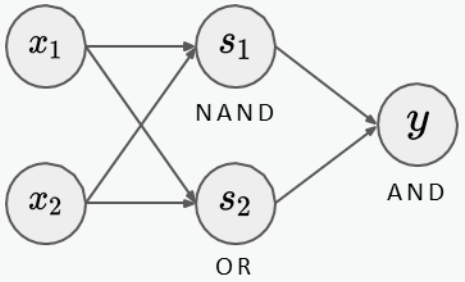
- Perceptron:
  - 여러 개의 신호를 받아서 하나의 신호로 출력하는 알고리즘 / 선형 분류기
- Multi-layer Perceptron:
  - “여러 개의 퍼셉트론(Perceptron)을 층으로 구성” 관점
  - 단순 선형 분류기인 퍼셉트론으로 해결 할 수 없던 XOR 문제를 해결
  - 퍼셉트론을 여러 층으로 쌓아서 입력 데이터를 새로운 층에서 다르게 표현함
  - Neural Network 의 모태
- Feed-forward Neural Network
  - “순방향(FeedForward) + 네트워크(Network)” 관점
  - 예시: 다층 퍼셉트론

Feed-forward Neural Network

- 순방향(FeedForward)
  - 입력층에서 출력층까지 앞으로만 전달
  - **유향 비순환 그래프**(DAG, Directed Acyclic Graph)

- 네트워크(Network)
  - 수많은 다른 함수들의 조합으로 표현 가능 $y = g(x) = f^{(2)}\left(f^{(1)}(x)\right)$
  - 예시: 다층 퍼셉트론 $y = g(x_1, x_2) = \text{AND}\left(\text{NAND}(x_1, x_2), \text{OR}(x_1, x_2)\right)$

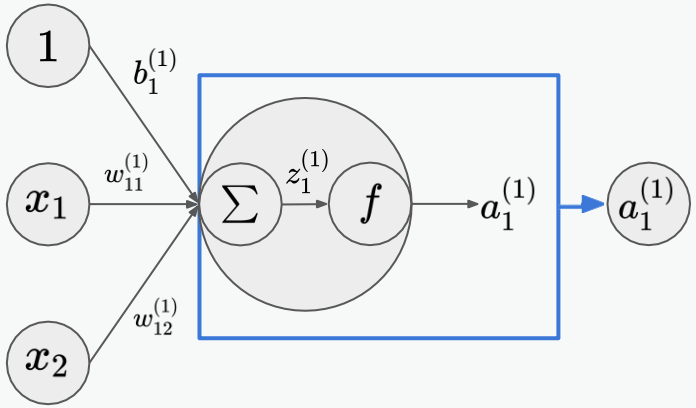




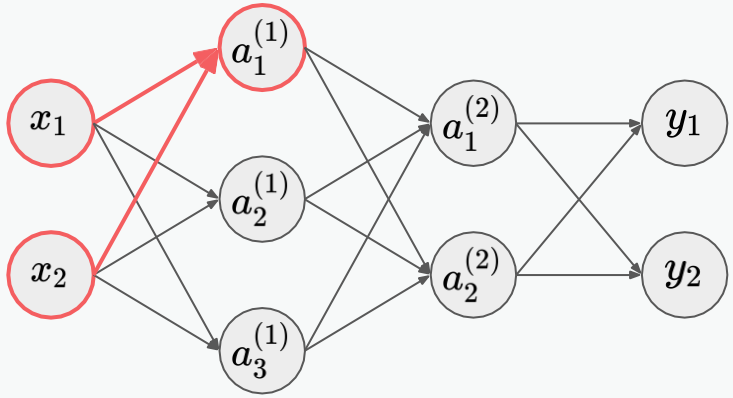
Feed-forward Neural Network

- 구성요소: 네트워크 표기법
  - 3층 네트워크(입력층은 세지 않는다)

$$y = g(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)$$



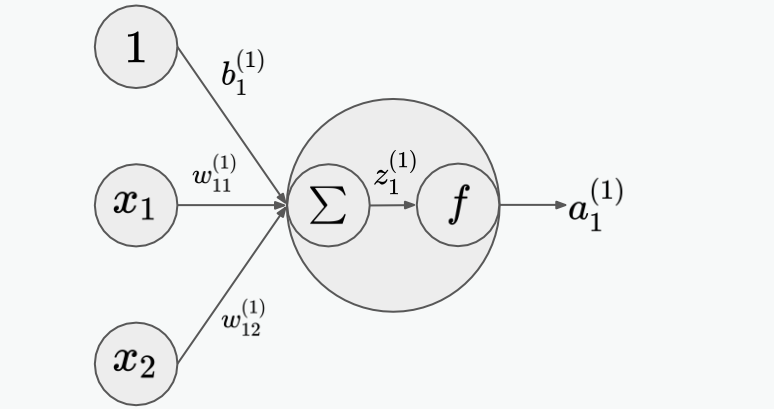
입력층에서은닉층1의 첫번째 뉴런으로 가는 퍼셉트론

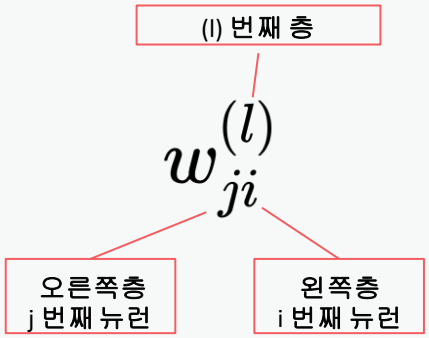


Input Layer | Hidden Layer 1 | Hidden Layer 2 | Output Layer

Feed-forward Neural Network

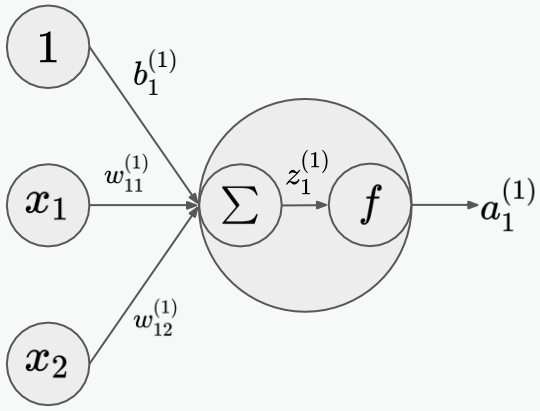
- 구성요소: 가중치 표기법





Feed-forward Neural Network

- FeedForward 과정(일부)
  - z: 가중합(weighted sum)
  - f: 활성함수(activation function)
    - 왜 활성함수(activation)를 사용하는가?



입력층에서은닉층1의 첫번째 뉴런으로
가는 퍼셉트론

가중합 단계

$$z_j^{(l)} = \sum_i (w_{ji}^{(l)} x_i + b_j^{(l)})$$

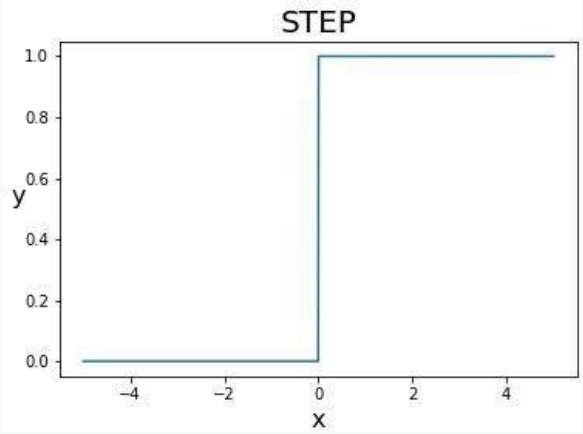
활성화 단계

$$a^{(l)} = f(z_j^{(l)})$$

Activation Function

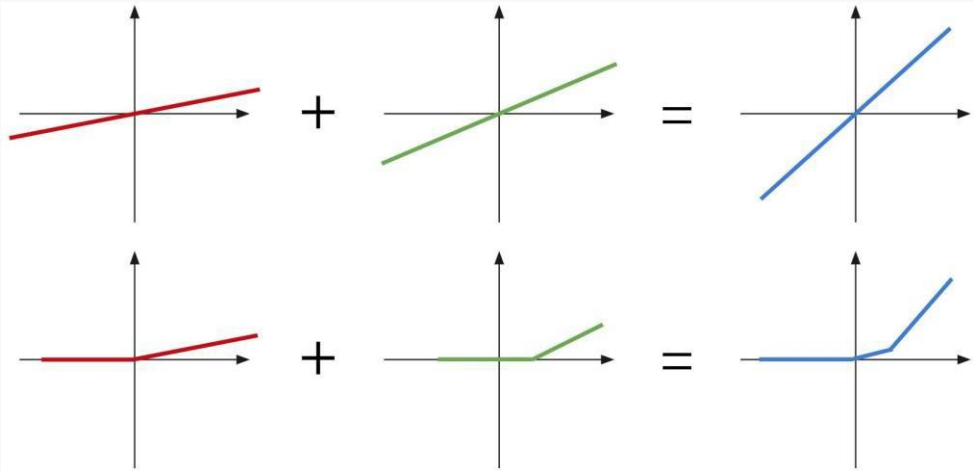
- 구성요소: 활성화 함수
  - 퍼셉트론에서 사용된 활성화 함수: 계단함수(Step Function)
  - 비선형함수

$$y = f(x) \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



Activation Function

- 왜 비선형함수를 사용할까?
  - 선형함수의 특징: 선형함수끼리의 결합 = 선형함수
  - 즉, 선형함수를 아무리 여러층 쌓아도 선형함수라는 뜻
  - 여러 비선형 함수(non-linear functions)의 결합이 더 다양한 표현을 만들 수 있다.
  - ➔ 더 복잡한 문제 해결 가능

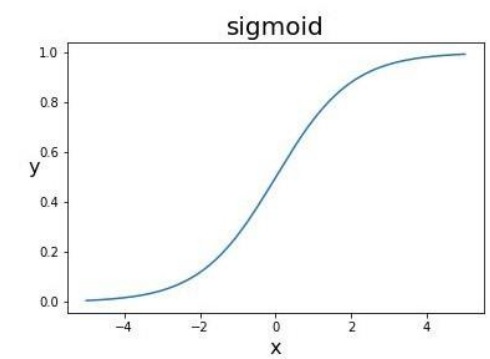


Activation Function

- 다양한 비선형 함수들: Sigmoid, ReLU, Tanh
  - Connectionist AI
  - Derivatives

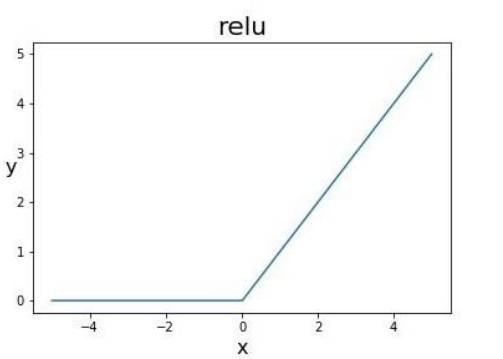
$$f(x) = \frac{1}{1+e^{-x}}$$

$$y \in [0, 1]$$



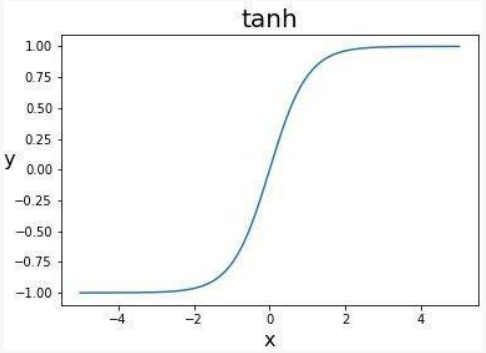
$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

$$y \in [0, x]$$



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$y \in [-1, 1]$$



Forward Propagation

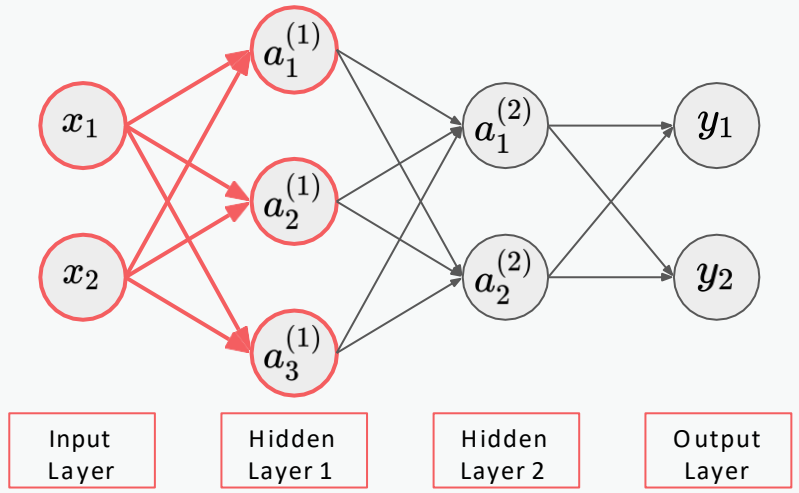
- 순방향 전파 (Forward propagation)
  - Input Layer > Hidden Layer 1

가중합 단계

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}$$
$$z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$
$$z_3^{(1)} = w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + b_3^{(1)}$$

활성화 단계

$$a_1^{(1)} = f(z_1^{(1)})$$
$$a_2^{(1)} = f(z_2^{(1)})$$
$$a_3^{(1)} = f(z_3^{(1)})$$



Forward Propagation

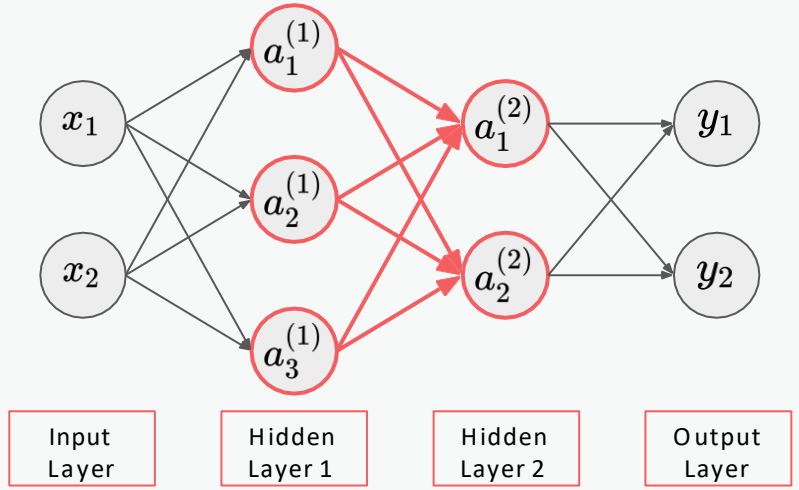
- 순방향 전파 (Forward propagation)
  - Hidden Layer 1 > Hidden Layer 2

가중합 단계

$$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)}$$
$$z_2^{(2)} = w_{21}^{(1)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)} + w_{23}^{(2)} a_3^{(1)} + b_2^{(2)}$$

활성화 단계

$$a_1^{(2)} = f(z_1^{(2)})$$
$$a_2^{(2)} = f(z_2^{(2)})$$



Forward Propagation

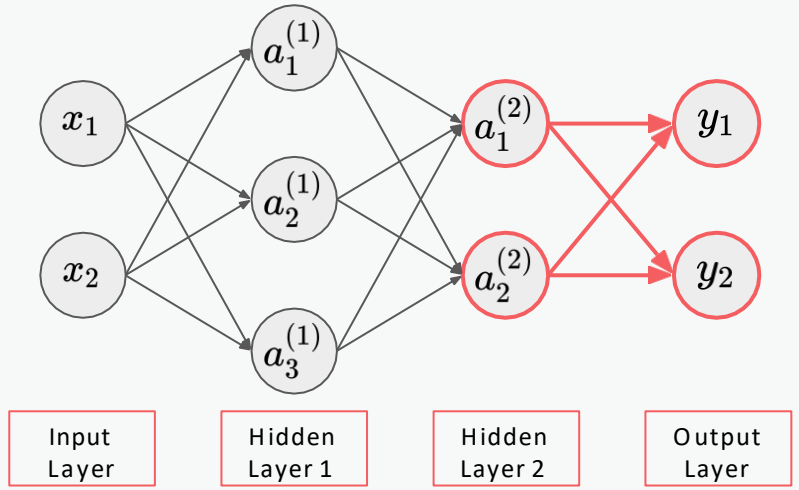
- 순방향 전파 (Forward propagation)
  - Hidden Layer 2 > Output Layer
  - 출력층의 활성화 함수는 특별함(차후에 손실함수와 연결)

가중합 단계

$$z_1^{(3)} = w_{11}^{(3)} a_1^{(2)} + w_{12}^{(3)} a_2^{(2)} + b_1^{(3)}$$
$$z_2^{(3)} = w_{21}^{(3)} a_1^{(2)} + w_{22}^{(3)} a_2^{(2)} + b_2^{(3)}$$

활성화 단계

$$y_1 = f(z_1^{(3)})$$
$$y_2 = f(z_2^{(3)})$$



Forward Propagation

- 순방향 전파 (Forward propagation)
  - 실제 연산시 뉴런 하나씩 계산하는 것이 아니라 행렬로 모든 것을 계산
  - Input Layer > Hidden Layer 1

가중합 단계

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}$$
$$z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$
$$z_3^{(1)} = w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + b_3^{(1)}$$

$$\begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^{(1)} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix}^{(1)} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}^{(1)}$$

$$z^{(1)} = x {W^{(1)}}^T + b^{(1)}$$

활성화 단계

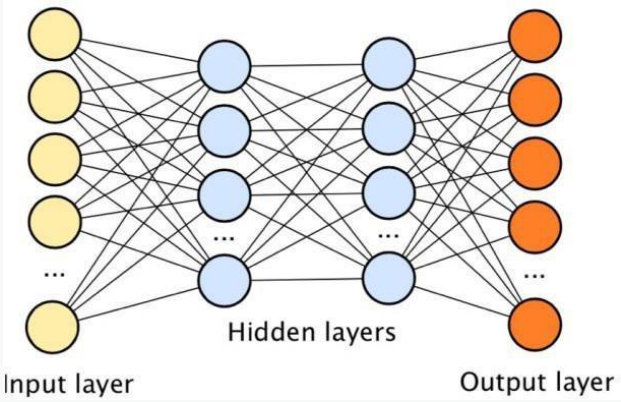
$$a_1^{(1)} = f(z_1^{(1)})$$
$$a_2^{(1)} = f(z_2^{(1)})$$
$$a_3^{(1)} = f(z_3^{(1)})$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}^{(1)} = f\left(\begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^{(1)}\right)$$

$$a^{(1)} = f(z^{(1)})$$

Neural Networks

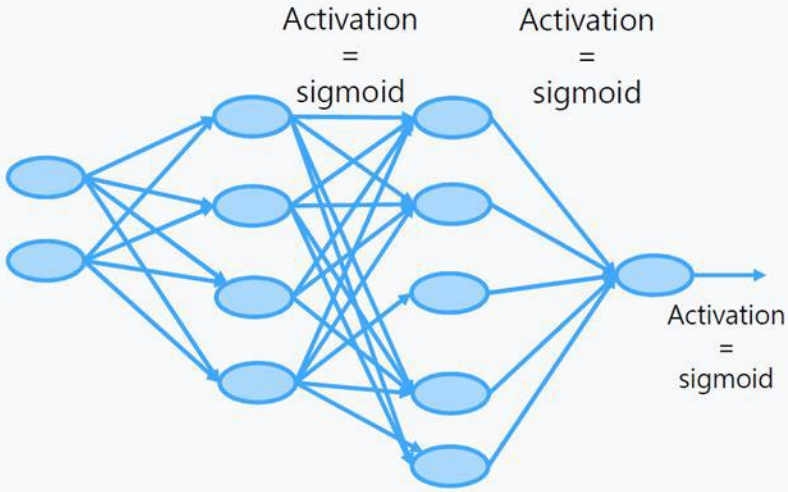
- 층을 깊게 쌓으면 심층 신경망(Deep Neural Network)
  - 비선형 함수를 사용하기 때문에 각 층에서 데이터를 다양하게/다른 관점으로 표현한다.
  - 더 복잡한 함수를 만들어 복잡한 문제를 해결할 수 있다.



Practice 3:
Multi-layer Perceptron

- Multi-layer Perceptron for XOR problem

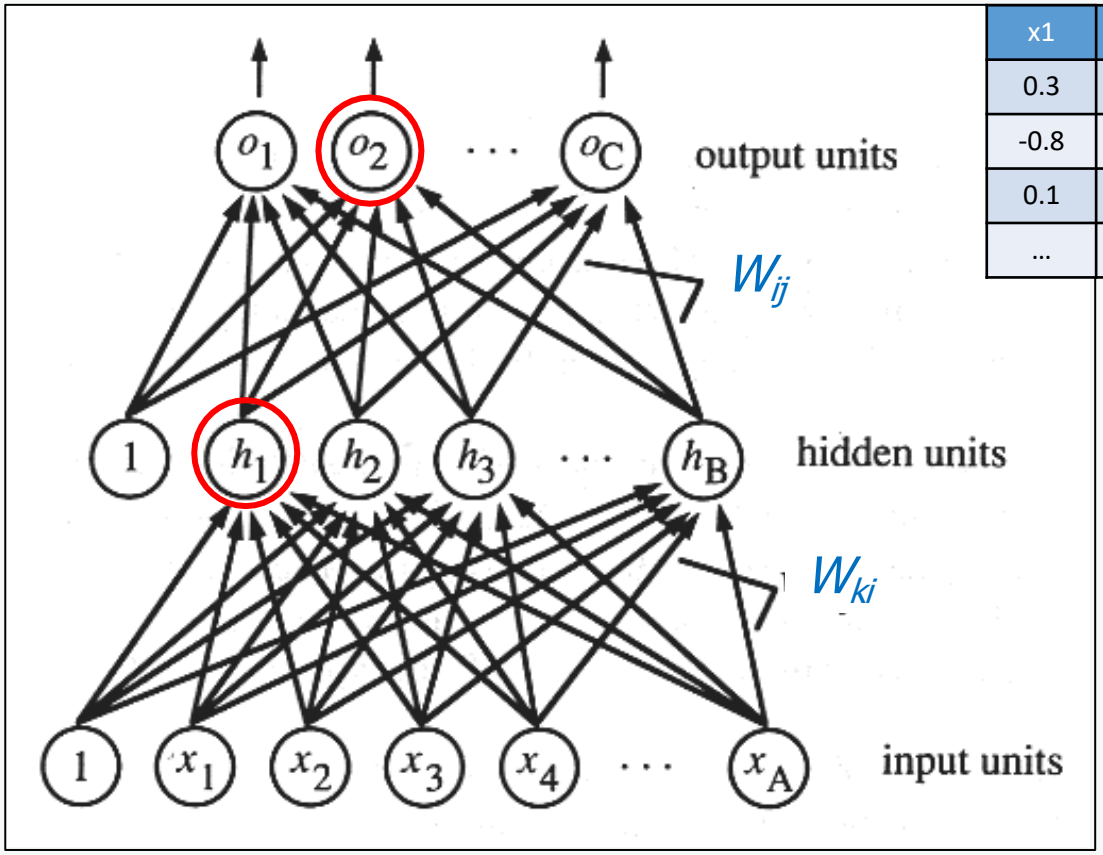
```
## data 선언
x_data = [[0.,0.], [0.,1.], [1.,0.],[1.,1.]]
y_data = [[0.], [1.], [1.], [0.]]
test_data=[[0.5, 0.5]]


## tf.keras를 활용한 perceptron 모델 구현.
model = tf.keras.Sequential() ## 모델 선언
model.add(tf.keras.layers.Dense(4, input_dim=2, activation='sigmoid')) # 선언된 모
model.add(tf.keras.layers.Dense(5, activation='sigmoid'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid')) # 선언된 모델에 add를통해
model.summary()

# 모델 loss, 학습 방법 결정하기
optimizer=tf.keras.optimizers.SGD(learning_rate=0.5) ### 경사 하강법으로 global mi
loss=tf.keras.losses.binary_crossentropy  ## 예측값 과 정답의 오차값 정의.
metrics=tf.keras.metrics.binary_accuracy ### 학습하면서 평가할 메트릭스 선언언

# 모델 컴파일하기
model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])

# 모델 동작하기
model.fit(x_data, y_data, epochs=2000, batch_size=4)
```



3. Backpropagation

Back Propagation

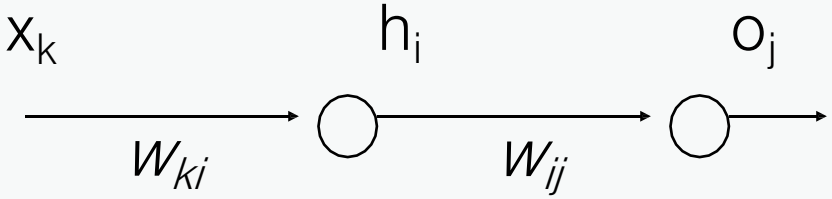
- Multi-layer neural networks



x1	x2	x3	d1	d2	d3
0.3	0.9	0.4	1	0	0
-0.8	-0.1	0.6	0	0	1
0.1	-0.5	-0.2	0	1	0
…	…	…	…	…	…

Back Propagation

- Forward propagation



  - Activation of hidden layer

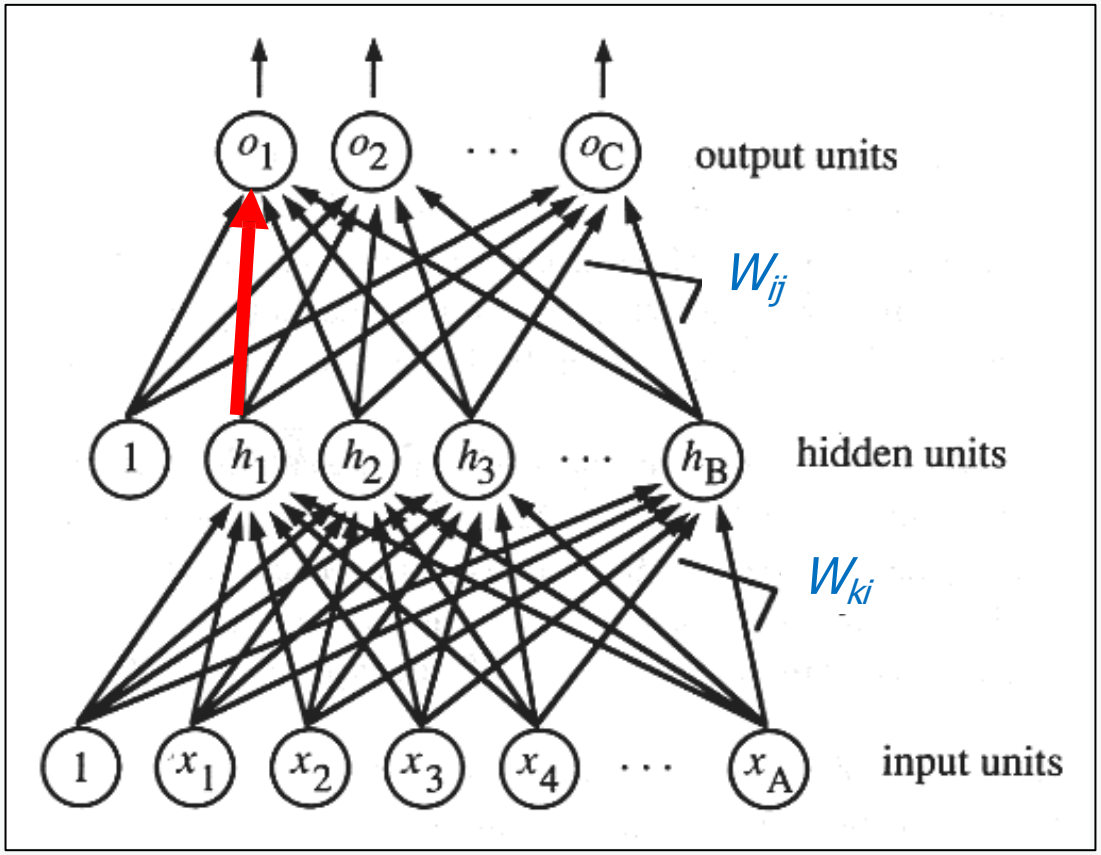
$$h_i = \frac{1}{(1+e^{-S_x})}, \quad S_x = \sum x_k \cdot w_{ki}$$

  - Activation of output layer

$$o_j = \frac{1}{(1+e^{-S_h})}, \quad S_h = \sum h_i \cdot w_{ij}$$

Back Propagation

$E = E_1 = \frac{1}{2}(d_1 - O_1)^2$



Back Propagation

- Learning
  - Let

$$E = \frac{1}{2}\sum_j (d_j - O_j)^2 \qquad E_j = \frac{1}{2}(d_j - O_j)^2$$

  - Adjusting weights to output layers

$$\Delta w_{ij} = -c \cdot \frac{\partial E}{\partial w_{ij}} = -c \cdot \frac{\partial E_j}{\partial w_{ij}}$$

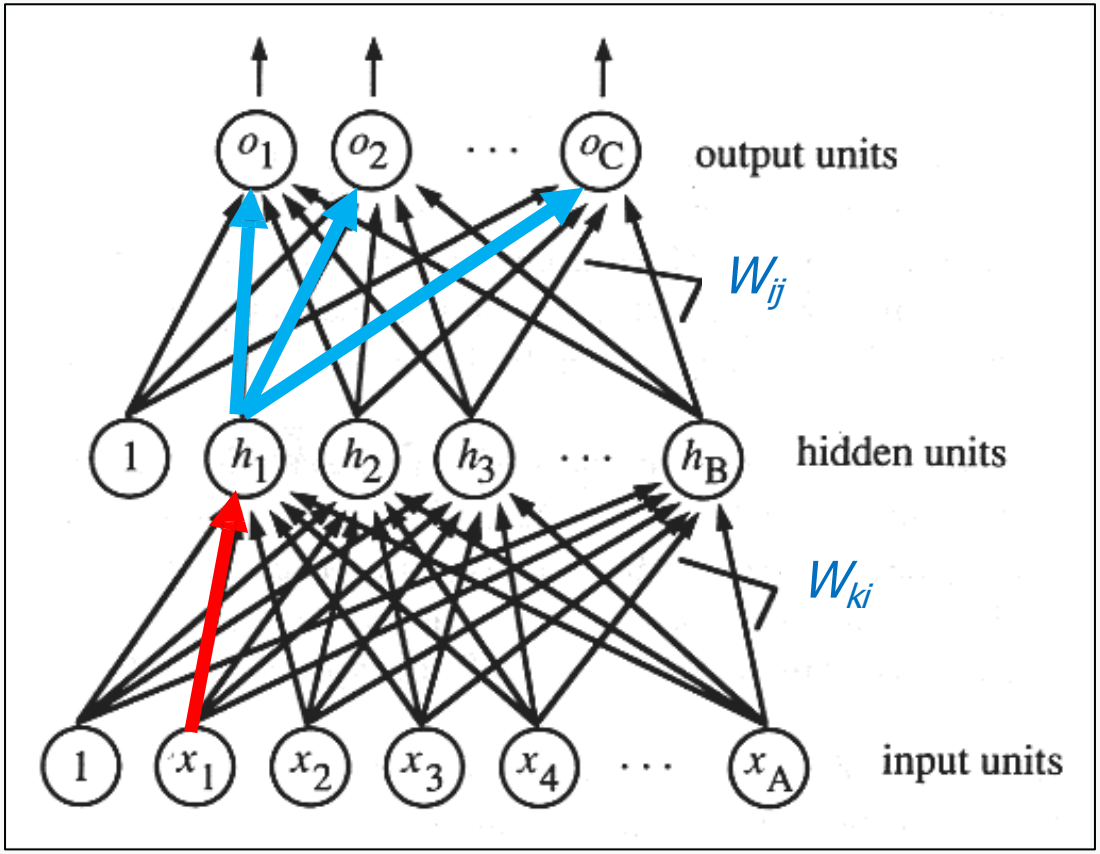
$$= -c \cdot \frac{\partial E_j}{\partial} \cdot \frac{\partial O}{}$$

$$\Delta w_{ij} = c \cdot (d_j - O_j) \cdot O_j (1 - O_j) \cdot h_i$$

Back Propagation

- Calculating Error

$$E = \sum E_j = \tfrac{1}{2}\sum (d_j - O_j)^2$$



Back Propagation

- Adjusting weights to hidden layers

$$\Delta w_{ki} = -c \cdot \frac{\partial E}{\partial w_{ki}}$$

$$= -c \cdot \sum_j \frac{\partial E_j}{\partial h_i} \cdot h_i (1 - h_i) \cdot x_k$$

$$= -c \cdot \sum_j \left(\frac{\partial E_j}{\partial} \cdot \frac{\partial O_j}{} \cdot \partial S_h \right.$$

$$\Delta w_{ki} = c \cdot \left(\sum_j ((d_j - O_j) \cdot O_j (1 - O_j) \cdot w_{ij}) \right) \cdot h_i (1 - h_i) \cdot x_k$$

Back Propagation

Define network

Initialize all weights

Until satisfied, Do

For each training example <X, d>

compute O = f(X)

For each output unit j

$\delta_j = (d_j - o_j)\, o_j\, (1 - o_j)$

For each hidden unit i

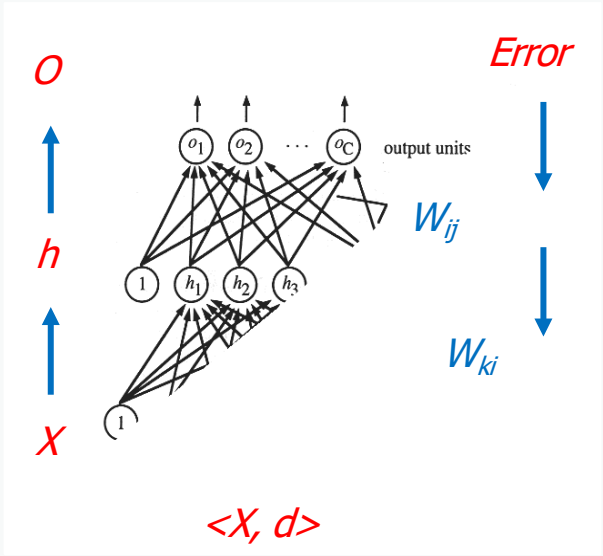
$\delta_i = (\Sigma\, w_{ij} \delta_j)\, h_i\, (1 - h_i)$

Update each network weight

$w_{ij} = w_{ij} + c\, \delta_j\, h_i$

$w_{ki} = w_{ki} + c\, \delta_i\, x_k$

x1	x2	x3	d1	d2	d3
0.3	0.9	0.4	1	0	0
-0.8	-0.1	0.6	0	0	1
0.1	-0.5	-0.2	0	1	0
...	...	...	...	...	...



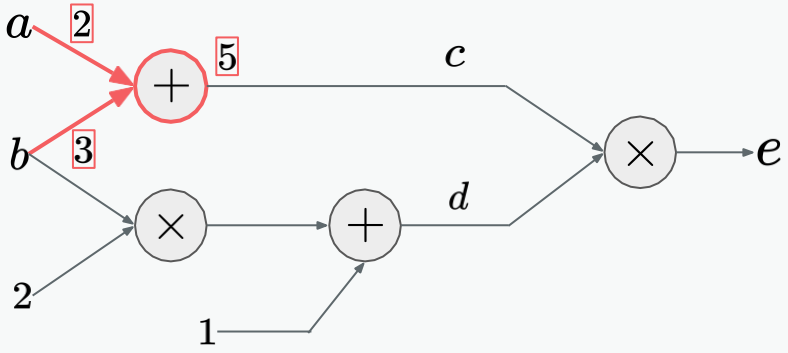
Back Propagation

- 오차역전파 (error back propagation)
  - 조금 더 효율적으로 기울기를 구할 방법은 무엇인가?
  - 한번에 계산하려고 하지말고 단계를 나눠서 계산하자!
  - 계산 그래프와 미분의 연쇄법칙을 이용하면 가능하다!

Computational Graph

- 계산 그래프
  - 계산과정을 그래프(graph)로 나타낸 것
  - 의미: 복잡한 계산 과정을 나눠서 “국소적 계산”으로 표현할 수 있다.
  - 모든 뉴럴 네트워크는 계산 그래프로 표현할 수 있다.

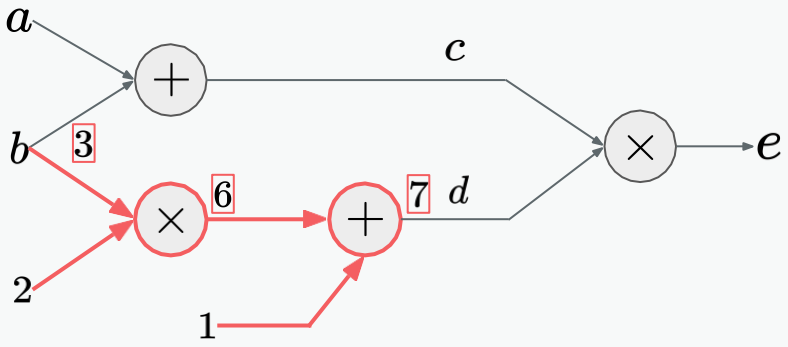
$$c(a,b) = a + b \quad \cdots (1)$$
$$d(b) = 2 \times b + 1 \quad \cdots (2)$$
$$e(c,d) = c \times d \quad \cdots (3)$$



Computational Graph

- 계산 그래프
  - 계산과정을 그래프(graph)로 나타낸 것
  - 의미: 복잡한 계산 과정을 나눠서 “국소적 계산”으로 표현할 수 있다.
  - 모든 뉴럴 네트워크는 계산 그래프로 표현할 수 있다.

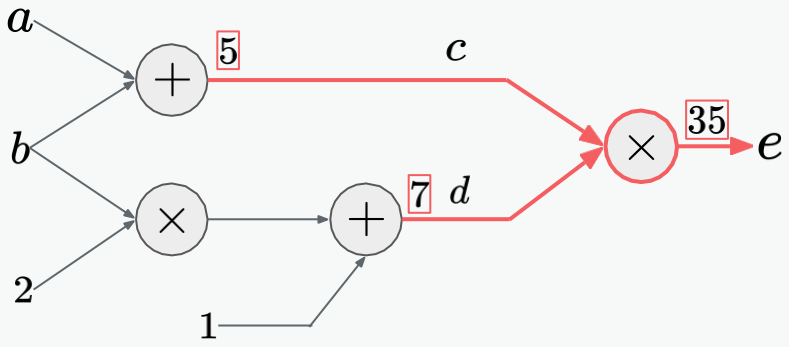
$$c(a,b) = a + b \quad \cdots (1)$$
$$d(b) = 2 \times b + 1 \quad \cdots (2)$$
$$e(c,d) = c \times d \quad \cdots (3)$$



Computational Graph

- 계산 그래프
  - 계산과정을 그래프(graph)로 나타낸 것
  - 의미: 복잡한 계산 과정을 나눠서 “국소적 계산”으로 표현할 수 있다.
  - 모든 뉴럴 네트워크는 계산 그래프로 표현할 수 있다.

$$c(a,b) = a + b \quad \cdots (1)$$
$$d(b) = 2 \times b + 1 \quad \cdots (2)$$
$$e(c,d) = c \times d \quad \cdots (3)$$



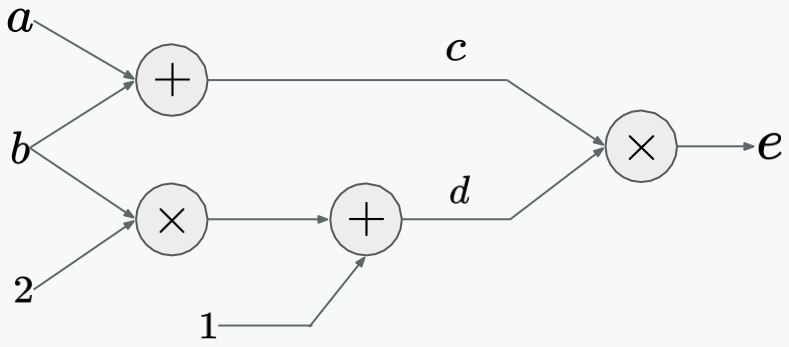
Computational Graph

- 계산 그래프에서 미분하기
  - 변수 a가 e에 얼만큼 영향을 미치는지 알고싶다!

  ➔ a가 조금 변했을 때 e가 얼마나 변화하는가?

  ➔ e를 a로 편미분!
  - 편미분: 다른 것은 변화하지 않고, a에 대한 변화량만 보고싶다!

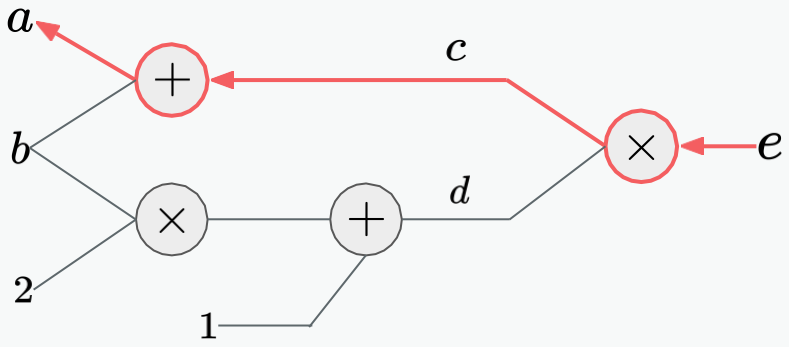
$$\frac{\partial e}{\partial a} = ?$$



Chain Rule

- 미분의 연쇄 법칙
  - 합성함수의 도함수에 관한 공식
  - 합성함수의 미분은 합성함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다.
  - a의 e에 대한 영향력 = c의 e에 대한 영향력 x a의 c에 대한 영향력

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a}$$

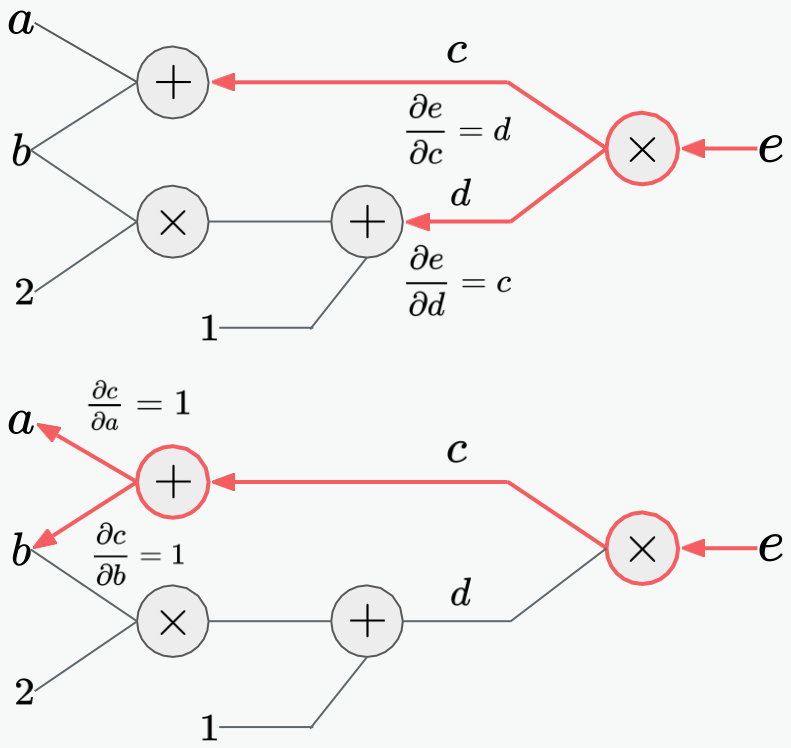


Chain Rule

- 미분의 연쇄 법칙

$$\frac{\partial e}{\partial a} = \boxed{\frac{\partial e}{\partial c}}\frac{\partial c}{\partial a} \qquad e(c,d) = c \times d \qquad \frac{\partial e}{\partial c} = d \quad \frac{\partial e}{\partial d} = c$$

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\boxed{\frac{\partial c}{\partial a}} \qquad c(a,b) = a + b \qquad \frac{\partial c}{\partial a} = 1 \quad \frac{\partial c}{\partial b} = 1$$



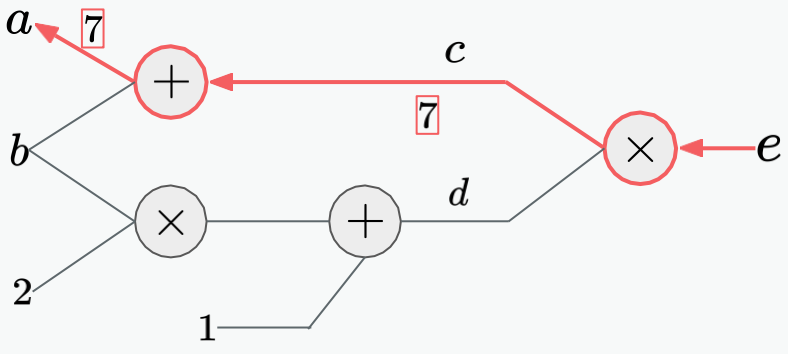
Chain Rule

- 미분의 연쇄 법칙
  - 합성함수의 도함수에 관한 공식
  - 합성함수의 미분은 합성함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다.
  - a의 e에 대한 영향력 = (c의 e에 대한 영향력) x (a의 c에 대한 영향력)

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a}$$
$$= d \times 1 = d$$

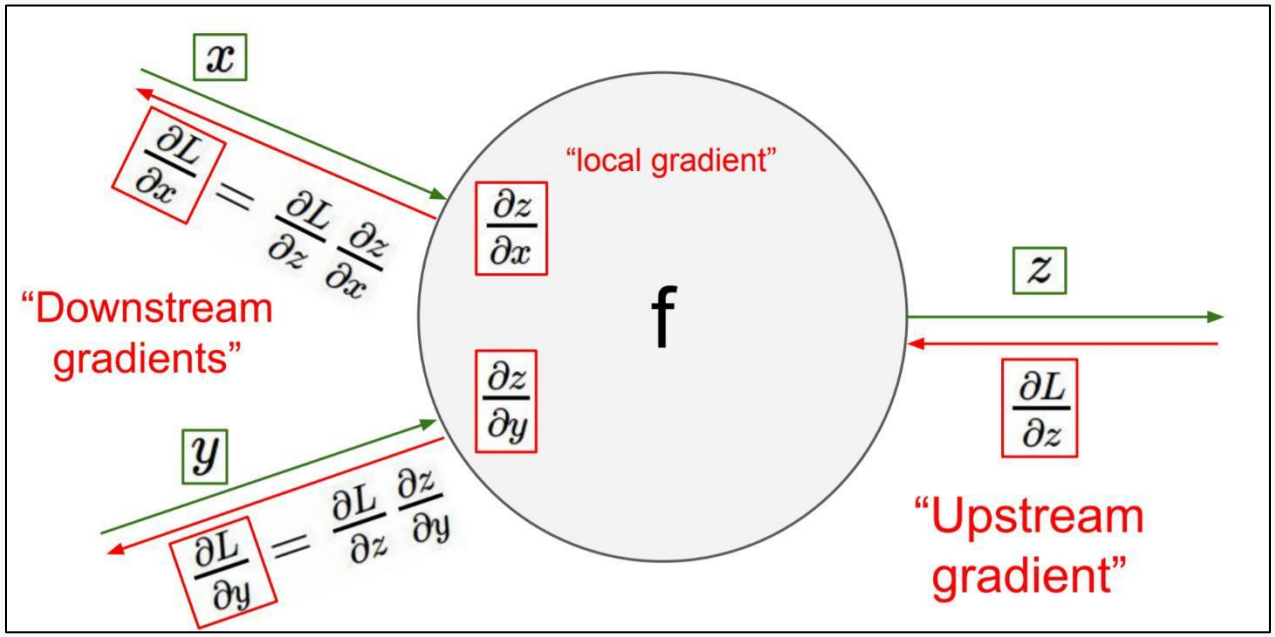
변수 a가 1 증가 시, e의 변화

$$e(a=2, b=3) = 35$$
$$e(a=3, b=3) = 42$$



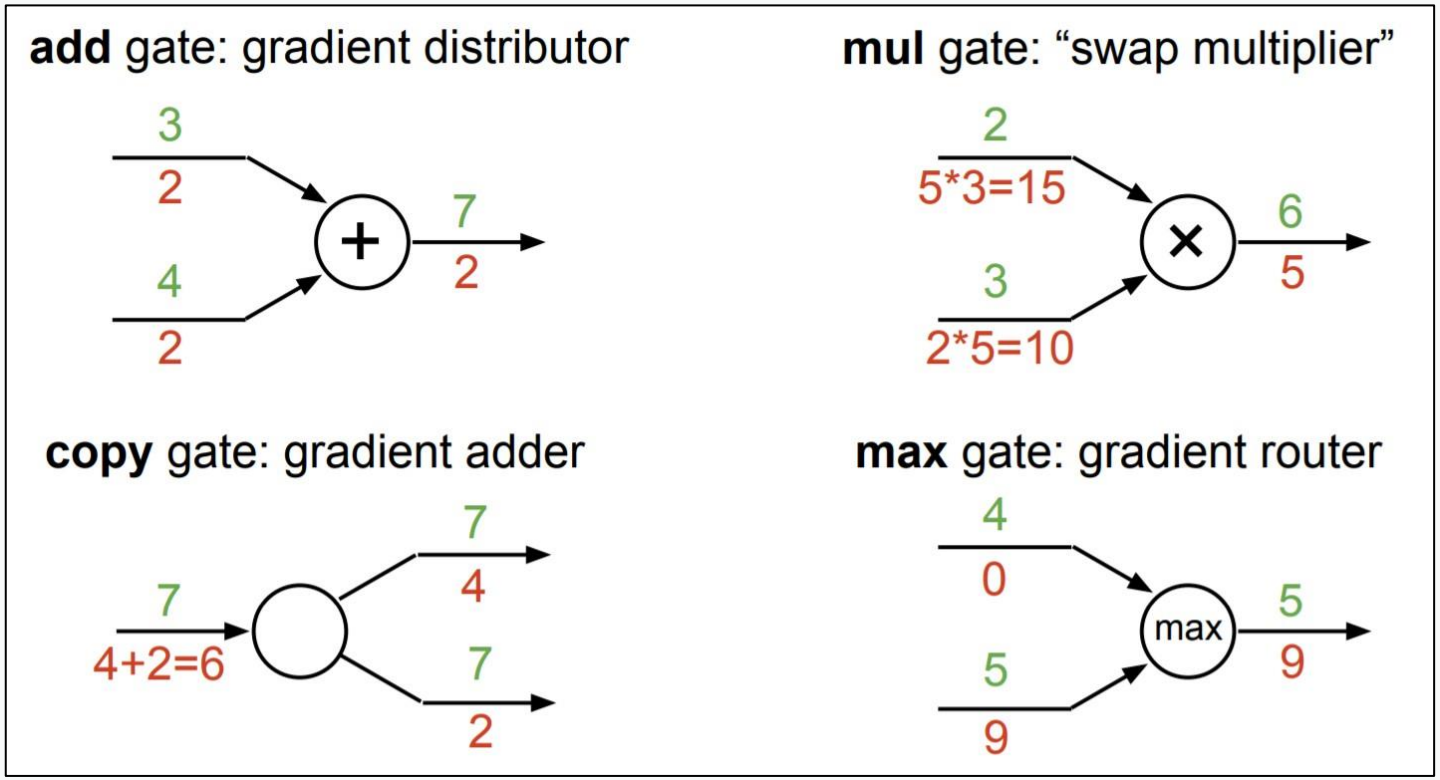
Chain Rule

- 미분의 연쇄 법칙 일반화
  - downstream gradient = upstream gradient * local gradient



Chain Rule

- gradient flow의 패턴
  - 각각의 operation을 class로 구성하고, forward 연산 시 이미 backward가 어떻게 이루어질지 결정할 수 있음



4. Neural Networks in Various Tasks

Practice 5:
Regression with Neural Networks

- House price prediction
  - Dataset: house_price_of_area.csv

	x1	x2	x3	x4	x5	y
1	32	84.87	10	24.98	121.54	37.9
2	19.5	306.59	9	24.98	121.53	42.2
3	13.3	561.98	5	24.98	121.54	47.3
	…	…	…	…	…	…
415	6.5	90.45	9	24.97	121.54	63.9

샘플수 : 415
속성 : 1 (house age), Length2(distance station), Length3(number of convenience stores) 등 5개
예측값 : house_price_of_area

Practice 5:
Regression with Neural Networks

- House price prediction
  - Dataset: house_price_of_area.csv

```python
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

## 데이터 읽어오기.
df = pd.read_csv("../dataset/house_price_of_unit_area.csv")
print(df.info())
print(df.head())

## data copy
dataset=df.copy()

## 데이터에서 label 데이터 추출
label_data=dataset.pop("house price of unit area")

# 모델의 설계
input_Layer = tf.keras.layers.Input(shape=(5,))
x = tf.keras.layers.Dense(50, activation='sigmoid')(input_Layer)
x= tf.keras.layers.Dense(100, activation='sigmoid')(x)
x= tf.keras.layers.Dense(300, activation='sigmoid')(x)
Out_Layer= tf.keras.layers.Dense(1, activation=None)(x)

model = tf.keras.Model(inputs=[input_Layer], outputs=[Out_Layer])
model.summary()
```

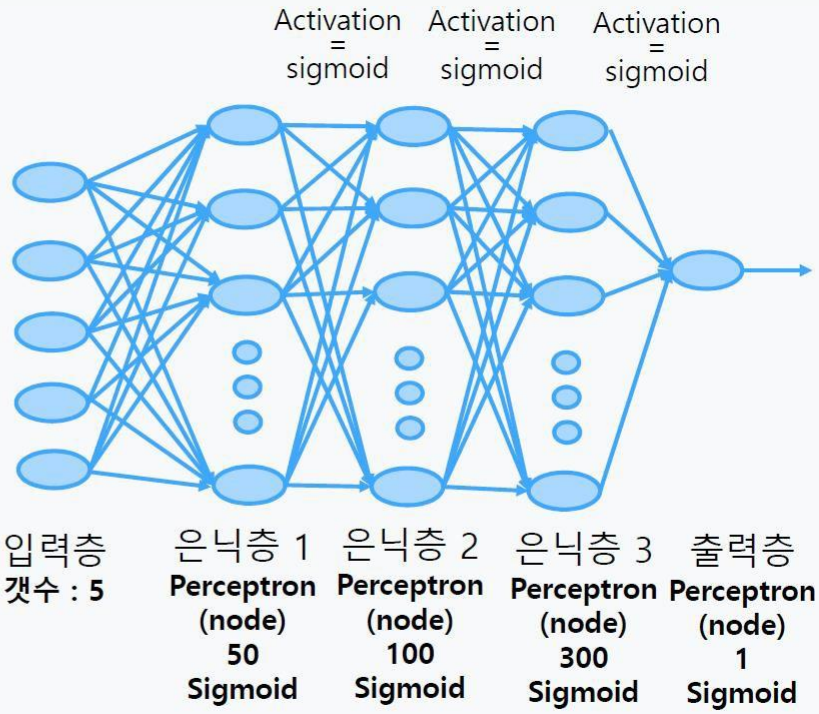
dataset	x1	x2	x3	x4	x5	y
1	32	84.87	10	24.98	121.54	37.9
2	19.5	306.59	9	24.98	121.53	42.2
3	13.3	561.98	5	24.98	121.54	47.3
	…	…	…	…	…	…
415	6.5	90.45	9	24.97	121.54	63.9

dataset label_data

Practice 5:
Regression with Neural Networks

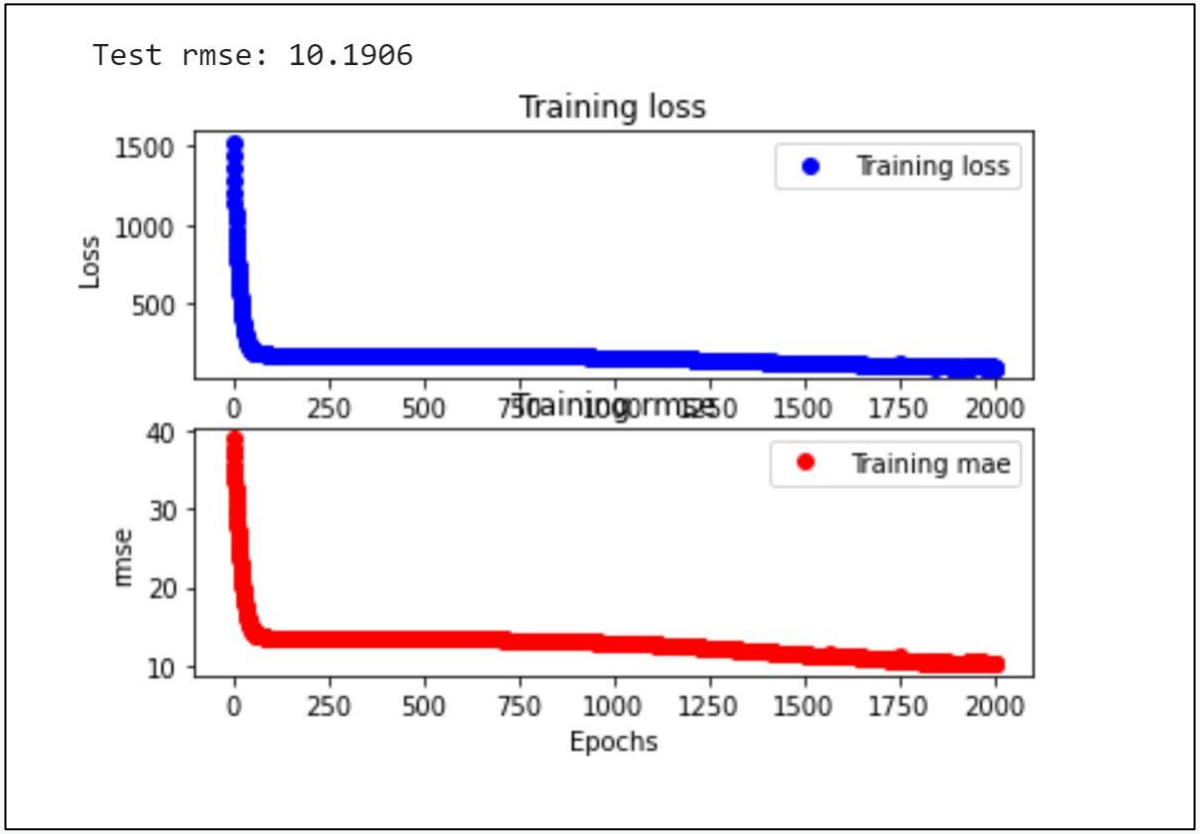
- House price prediction
  - Dataset: house_price_of_area.csv

```python
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

## 데이터 읽어오기.
df = pd.read_csv("../dataset/house_price_of_unit_area.csv")
print(df.info())
print(df.head())

## data copy
dataset=df.copy()

## 데이터에서 label 데이터 추출
label_data=dataset.pop("house price of unit area")

# 모델의 설계
input_Layer = tf.keras.layers.Input(shape=(5,))
x = tf.keras.layers.Dense(50, activation='sigmoid')(input_Layer)
x= tf.keras.layers.Dense(100, activation='sigmoid')(x)
x= tf.keras.layers.Dense(300, activation='sigmoid')(x)
Out_Layer= tf.keras.layers.Dense(1, activation=None)(x)

model = tf.keras.Model(inputs=[input_Layer], outputs=[Out_Layer])
model.summary()
```



Practice 5:
Regression with Neural Networks

- House price prediction
  - Dataset: house_price_of_area.csv



Exercise 3:
Regression with Neural Networks

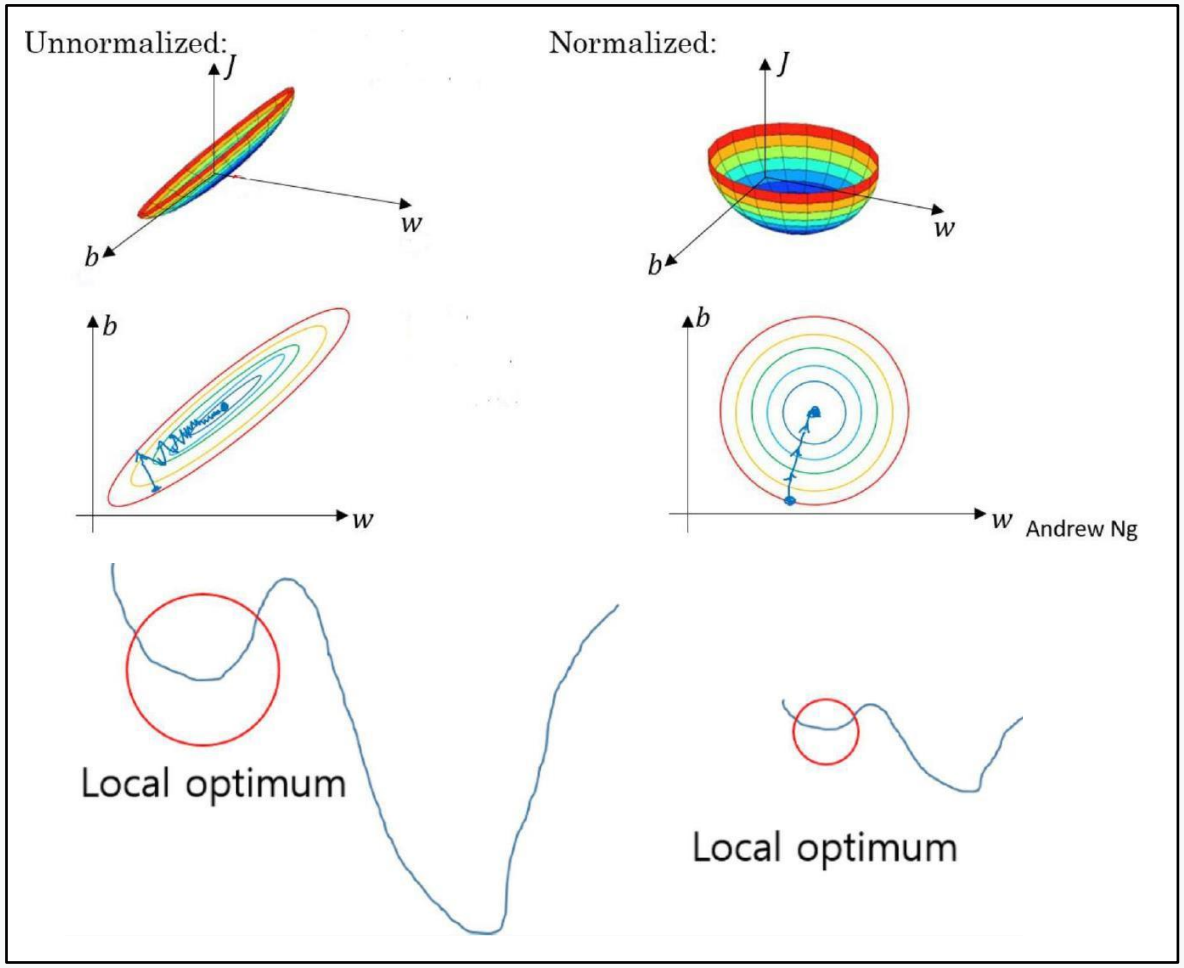
- House price prediction
  - BostonHousing.csv 데이터를 이용하여 Neural Network 를 학습시키고 집값을 예측하는 모델을 만들기

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
1	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
2	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
3	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
	...	...	...	...	...	...	...	...	...	...	...	...	...	...
506	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

샘플수 : 506
속성 : CRIM(범죄율), ZN(거주지역비율), INDUS(토지비율),CHAS(찰스강 경계위치),NOX(일산화질소값),RM(방의갯수) ... 등 총 13개
예측값 : MEDV(집값: 단위는 $1,000)

Practice 6:
NN with Normalization

- Normalization



Practice 6:
NN with Normalization

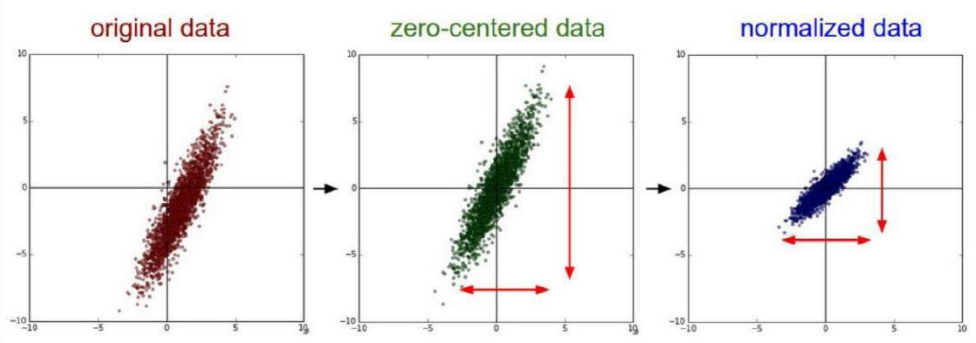
- Normalization

1. Min max : 일정범위(일반적으로 0~1) 사이로 scaling 함

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2. Standardization : mean 차감을 통해 zero-centered 화 해주고 std로 나누어 주어 데이터가 일정 범위안에 머무르게 함

$$x = \frac{x - x_{mean}}{x_{std}}$$



Practice 6:
NN with Normalization

- Normalization

```
## 데이터 읽어오기.## 데이터 읽어오기.
raw_df = pd.read_csv("../dataset/test.csv")
print(raw_df.info())
print(raw_df.head())
dataset=raw_df.copy()

## 데이터 분포도 확인하기.
sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
plt.show()

train_labels = dataset.pop("D")

## 데이터의 min , max, mean, std 값 구하기.
dataset_stats = dataset.describe()
dataset_stats = dataset_stats.transpose()

## data normalization
def min_max_norm(x):
  return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])

def standard_norm(x):
  return (x - dataset_stats['mean']) / dataset_stats['std']

min_max_norm_train_data = min_max_norm(dataset)
standard_norm_train_data = standard_norm(dataset)
```

Data 읽어오기

	A	B	C	D
1	1	10	100	2
2	2	20	200	2
3	3	30	300	2
	...	...	...	...
10	10	100	1000	2

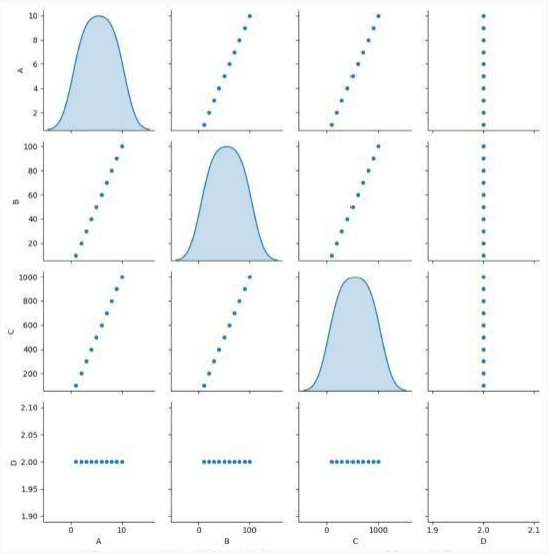
Practice 6:
NN with Normalization

- Normalization

```
5
6   ## 데이터 읽어오기.## 데이터 읽어오기.
7   raw_df = pd.read_csv("../dataset/test.csv")
8   print(raw_df.info())
9   print(raw_df.head())
10  dataset=raw_df.copy()
11
12  ## 데이터 분포도 확인하기.
13  sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
14  plt.show()
15
16  train_labels = dataset.pop("D")
17
18  ## 데이터의 min , max, mean, std 값 구하기.
19  dataset_stats = dataset.describe()
20  dataset_stats = dataset_stats.transpose()
21
22  ## data normalization
23  def min_max_norm(x):
24    return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])
25
26  def standard_norm(x):
27    return (x - dataset_stats['mean']) / dataset_stats['std']
28
29  min_max_norm_train_data = min_max_norm(dataset)
30  standard_norm_train_data = standard_norm(dataset)
31
```

Data 읽어오기

Data 분포도



Practice 6:
NN with Normalization

- Normalization

```
## 데이터 읽어오기.## 데이터 읽어오기.
raw_df = pd.read_csv("../dataset/test.csv")
print(raw_df.info())
print(raw_df.head())
dataset=raw_df.copy()

## 데이터 분포도 확인하기.
sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
plt.show()

train_labels = dataset.pop("D")

## 데이터의 min , max, mean, std 값 구하기.
dataset_stats = dataset.describe()
dataset_stats = dataset_stats.transpose()

## data normalization
def min_max_norm(x):
  return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])

def standard_norm(x):
  return (x - dataset_stats['mean']) / dataset_stats['std']

min_max_norm_train_data = min_max_norm(dataset)
standard_norm_train_data = standard_norm(dataset)
```

Data 읽어오기

Data 분포도

입력/정답분리

A	B	C	D
1	10	100	2
2	20	200	2
3	30	300	2
...	...	...	...
10	100	1000	2

Practice 6:
NN with Normalization

- Normalization

```
5
6   ## 데이터 읽어오기.## 데이터 읽어오기.
7   raw_df = pd.read_csv("../dataset/test.csv")
8   print(raw_df.info())
9   print(raw_df.head())
10  dataset=raw_df.copy()
11
12  ## 데이터 분포도 확인하기.
13  sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
14  plt.show()
15
16  train_labels = dataset.pop("D")
17
18  ## 데이터의 min , max, mean, std 값 구하기.
19  dataset_stats = dataset.describe()
20  dataset_stats = dataset_stats.transpose()
21
22  ## data normalization
23  def min_max_norm(x):
24    return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])
25
26  def standard_norm(x):
27    return (x - dataset_stats['mean']) / dataset_stats['std']
28
29  min_max_norm_train_data = min_max_norm(dataset)
30  standard_norm_train_data = standard_norm(dataset)
31
```

Data 읽어오기

Data 분포도

입력/정답분리

A	B	C	D
1	10	100	2
2	20	200	2
3	30	300	2
...	...	...	...
10	100	1000	2

Practice 6:
NN with Normalization

- Normalization

```
5
6   ## 데이터 읽어오기.## 데이터 읽어오기.
7   raw_df = pd.read_csv("../dataset/test.csv")
8   print(raw_df.info())
9   print(raw_df.head())
10  dataset=raw_df.copy()
11
12  ## 데이터 분포도 확인하기.
13  sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
14  plt.show()
15
16  train_labels = dataset.pop("D")
17
18  ## 데이터의 min , max, mean, std 값 구하기.
19  dataset_stats = dataset.describe()
20  dataset_stats = dataset_stats.transpose()
21
22  ## data normalization
23  def min_max_norm(x):
24    return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])
25
26  def standard_norm(x):
27    return (x - dataset_stats['mean']) / dataset_stats['std']
28
29  min_max_norm_train_data = min_max_norm(dataset)
30  standard_norm_train_data = standard_norm(dataset)
31
```

Data 읽어오기

Data 분포도

입력/정답분리

데이터 특성 값 구하기

	A	B	C	D
count	10.00000	10.000000	10.000000	10.0
mean	5.50000	55.000000	550.000000	2.0
std	3.02765	30.276504	302.765035	0.0
min	1.00000	10.000000	100.000000	2.0
25%	3.25000	32.500000	325.000000	2.0
50%	5.50000	55.000000	550.000000	2.0
75%	7.75000	77.500000	775.000000	2.0
max	10.00000	100.000000	1000.000000	2.0

Practice 6:
NN with Normalization

- Normalization

```
## 데이터 읽어오기.## 데이터 읽어오기.
raw_df = pd.read_csv("../dataset/test.csv")
print(raw_df.info())
print(raw_df.head())
dataset=raw_df.copy()

## 데이터 분포도 확인하기.
sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
plt.show()

train_labels = dataset.pop("D")

## 데이터의 min , max, mean, std 값 구하기.
dataset_stats = dataset.describe()
dataset_stats = dataset_stats.transpose()

## data normalization
def min_max_norm(x):
  return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])

def standard_norm(x):
  return (x - dataset_stats['mean']) / dataset_stats['std']

min_max_norm_train_data = min_max_norm(dataset)
standard_norm_train_data = standard_norm(dataset)
```

Data 읽어오기

Data 분포도

입력/정답분리

데이터 특성 값 구하기

데이터 특성 매트릭스 행/열 바꾸기

	A	B	C	D
count	10.00000	10.000000	10.000000	10.0
mean	5.50000	55.000000	550.000000	2.0
std	3.02765	30.276504	302.765035	0.0
min	1.00000	10.000000	100.000000	2.0
25%	3.25000	32.500000	325.000000	2.0
50%	5.50000	55.000000	550.000000	2.0
75%	7.75000	77.500000	775.000000	2.0
max	10.00000	100.000000	1000.000000	2.0

	count	mean	std	min	25%	50%	75%	max
A	10.0	5.5	3.027650	1.0	3.25	5.5	7.75	10.0
B	10.0	55.0	30.276504	10.0	32.50	55.0	77.50	100.0
C	10.0	550.0	302.765035	100.0	325.00	550.0	775.00	1000.0

Practice 6:
NN with Normalization

- Normalization

```
5
6   ## 데이터 읽어오기.## 데이터 읽어오기.
7   raw_df = pd.read_csv("../dataset/test.csv")
8   print(raw_df.info())
9   print(raw_df.head())
10  dataset=raw_df.copy()
11
12  ## 데이터 분포도 확인하기.
13  sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
14  plt.show()
15
16  train_labels = dataset.pop("D")
17
18  ## 데이터의 min , max, mean, std 값 구하기.
19  dataset_stats = dataset.describe()
20  dataset_stats = dataset_stats.transpose()
21
22  ## data normalization
23  def min_max_norm(x):
24    return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])
25
26  def standard_norm(x):
27    return (x - dataset_stats['mean']) / dataset_stats['std']
28
29  min_max_norm_train_data = min_max_norm(dataset)
30  standard_norm_train_data = standard_norm(dataset)
31
```

Data 읽어오기

Data 분포도

입력/정답분리

데이터 특성
값 구하기

데이터 특성
매트릭스 행/열 바꾸기

Min/max normalization , std normalization 함수선언

$$x = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad x = \frac{x - x_{mean}}{x_{std}}$$

Practice 6:
NN with Normalization

- Normalization

```
## 데이터 읽어오기.## 데이터 읽어오기.
raw_df = pd.read_csv("../dataset/test.csv")
print(raw_df.info())
print(raw_df.head())
dataset=raw_df.copy()

## 데이터 분포도 확인하기.
sns.pairplot(dataset[["A","B","C","D"]], diag_kind="kde")
plt.show()

train_labels = dataset.pop("D")

## 데이터의 min , max, mean, std 값 구하기.
dataset_stats = dataset.describe()
dataset_stats = dataset_stats.transpose()

## data normalization
def min_max_norm(x):
  return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])

def standard_norm(x):
  return (x - dataset_stats['mean']) / dataset_stats['std']

min_max_norm_train_data = min_max_norm(dataset)
standard_norm_train_data = standard_norm(dataset)
```

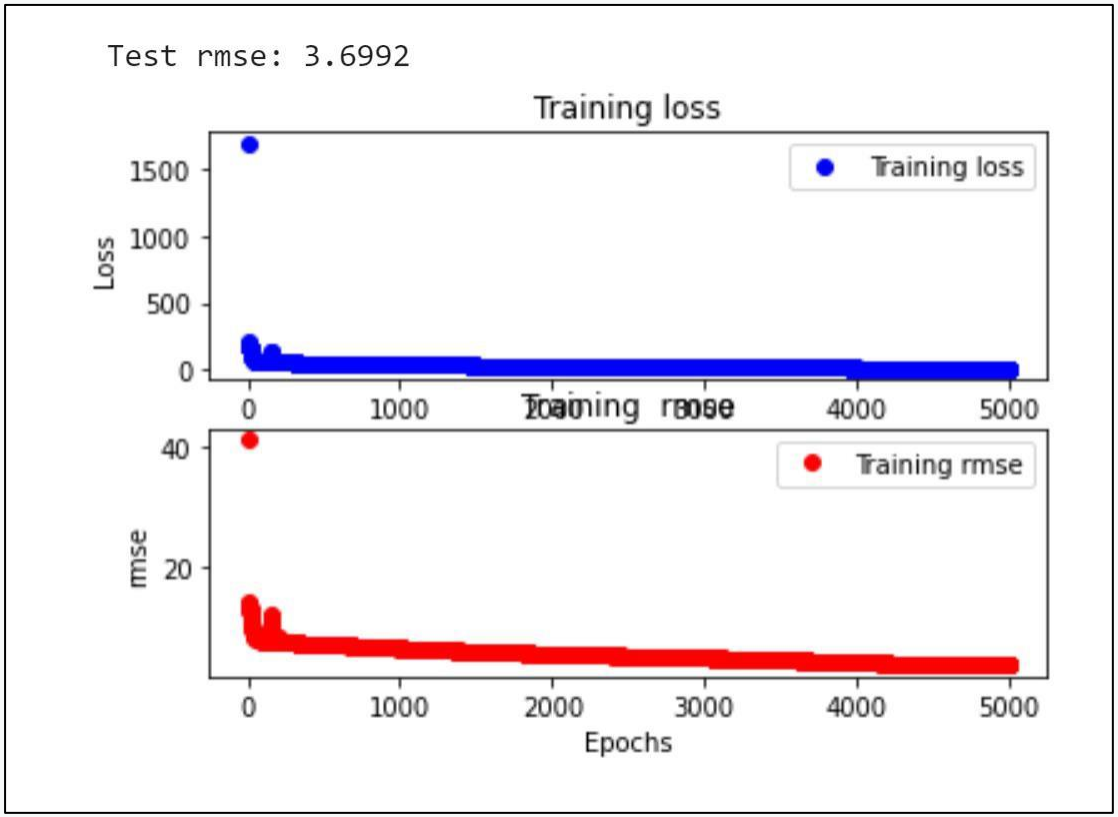
Min/max normalization , std normalization 적용

	A	B	C
0	0.000000	0.000000	0.000000
1	0.111111	0.111111	0.111111
2	0.222222	0.222222	0.222222
3	0.333333	0.333333	0.333333
4	0.444444	0.444444	0.444444
5	0.555556	0.555556	0.555556
6	0.666667	0.666667	0.666667
7	0.777778	0.777778	0.777778
8	0.888889	0.888889	0.888889
9	1.000000	1.000000	1.000000

	A	B	C
0	-1.486301	-1.486301	-1.486301
1	-1.156012	-1.156012	-1.156012
2	-0.825723	-0.825723	-0.825723
3	-0.495434	-0.495434	-0.495434
4	-0.165145	-0.165145	-0.165145
5	0.165145	0.165145	0.165145
6	0.495434	0.495434	0.495434
7	0.825723	0.825723	0.825723
8	1.156012	1.156012	1.156012
9	1.486301	1.486301	1.486301

Practice 7:
NN with Normalization

- House price prediction
  - Dataset: house_price_of_area.csv
  - normalization을 적용해보기



Exercise 4:
NN with Normalization

- House price prediction
  - BostonHousing.csv 데이터를 이용하여 Neural Network 를 학습시키고 집값을 예측하는 모델을 만들기
  - Normalization을 적용하기