

# **Pytorch로 배우는 Neural Network 입문**

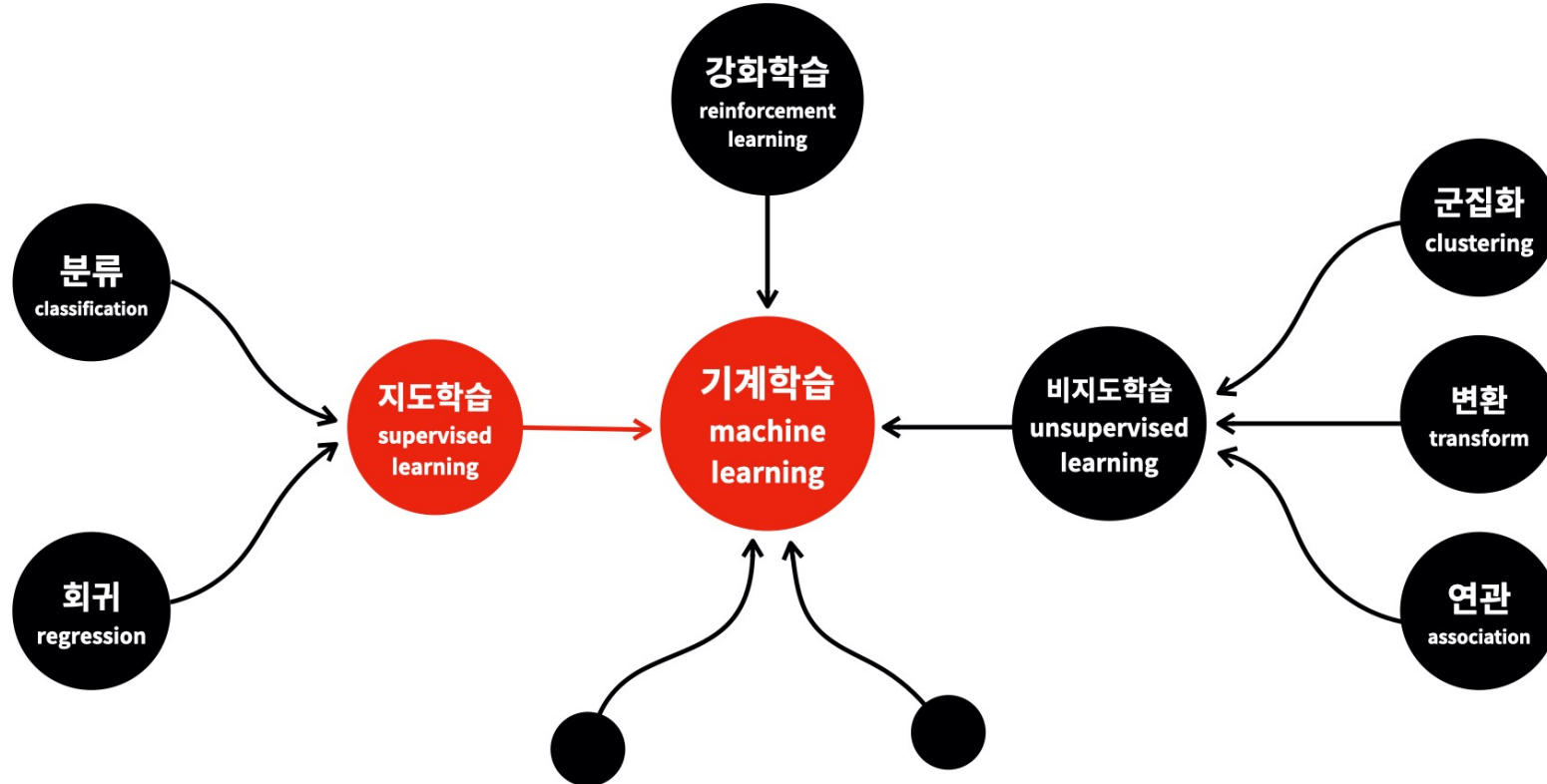
# 학습 순서

- ML : 기계가 데이터를 학습한다
- Pytorch Basic
- Linear Regression 선형 회귀
- Multi Class Classification 멀티 클래스 분류기
- Neural Network : Multi Layer Perceptron

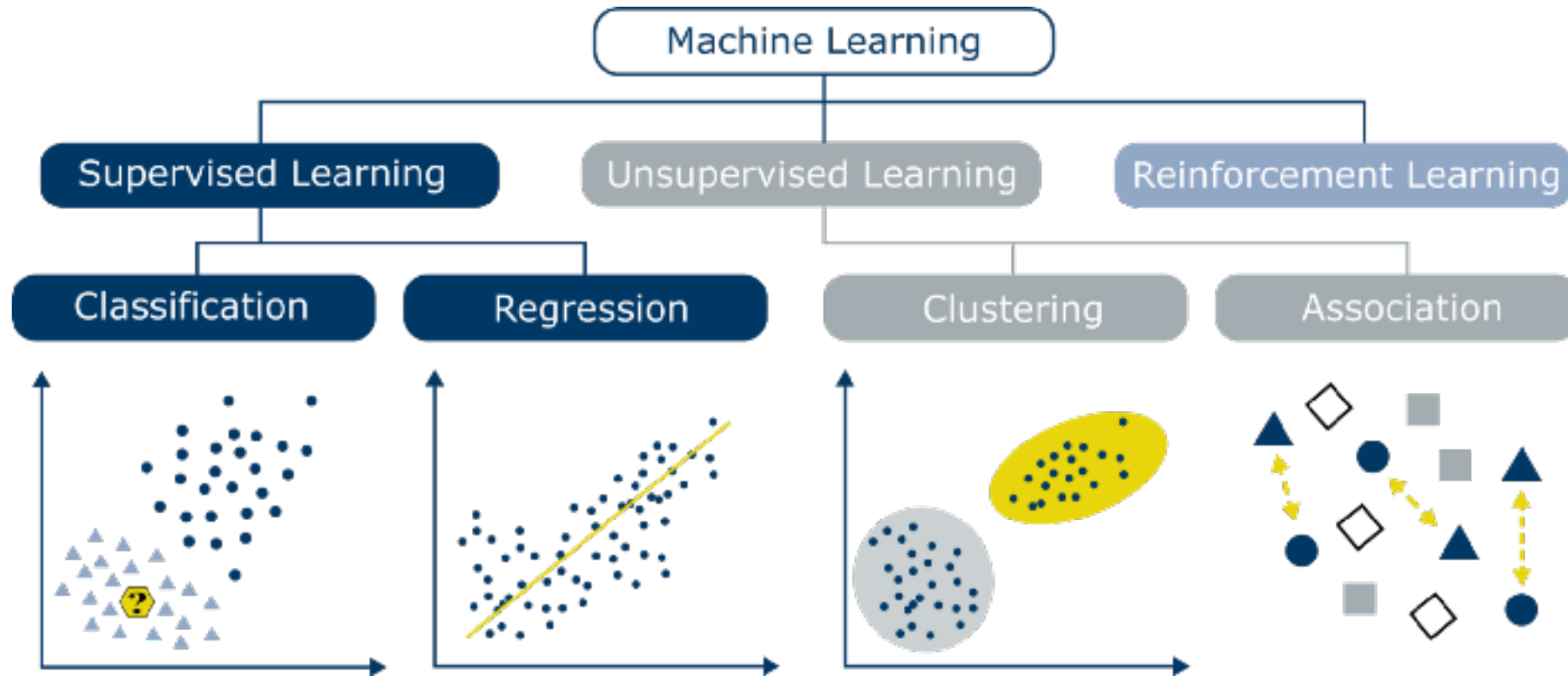
# **Machine Learning**

## **: 기계가 데이터를 학습한다**

# Machine Learning?



# Machine Learning?



# Machine Learning : 기계가 데이터를 학습한다

- 학습 : Training? 무엇을 학습?
  - $Y = \text{target}$  을 학습
  - Target은 Task에 따라 달라짐
  - Regression / Classification
- 대표적으로 Computer Vision / NLP / Speech Recognition 등등...
  - 인공지능을 대표하는 분야
  - 딥러닝을 주도적으로 이끄는 분야

# Pytorch Basic

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/pytorch\\_basic.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/pytorch_basic.ipynb)

# **선형 회귀 (Linear Regression)**



# 선형 회귀(Linear Regression)

- Why Linear Regression?
  - $Y = Wx + b$  를 알기 위해
  - Multi layer Perceptron 도 결국  $Y = Wx + b$ 에 기반
- 직관적인 Linear Regression으로 머신러닝 학습 과정 이해하기

# 선형 회귀(Linear Regression)

- 알아야 할 개념들
  - Hypothesis :  $Y = Wx + b \rightarrow W (?) / b (?)$
  - Cost Function
  - Gradient Descent

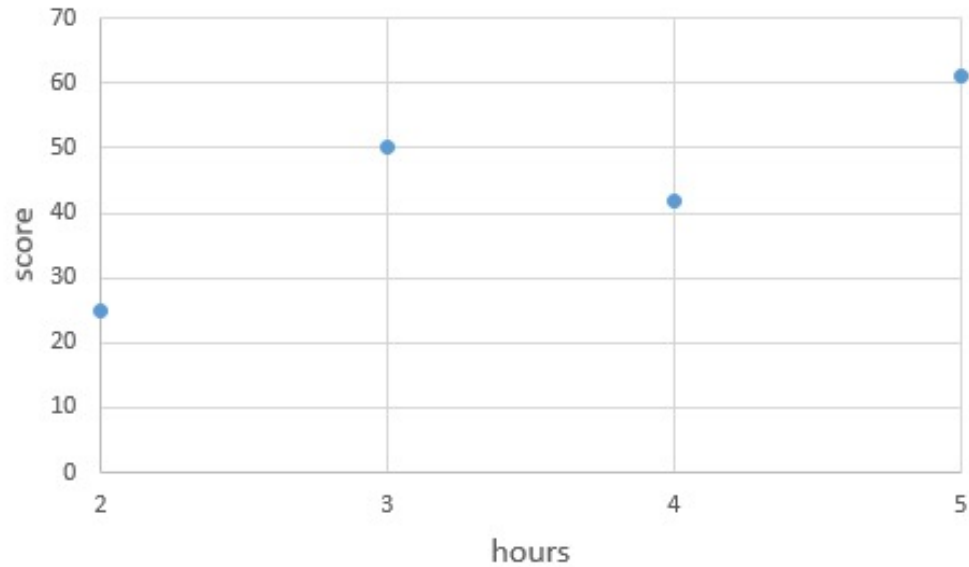
# 선형 회귀(Linear Regression)

## Linear Regression

- 시험 공부에 투자한 시간(x) 으로 성적(Y = Target)을 예측해보자.
- Regression :  $Y = Wx + b$ 
  - W : Weight (가중치)
  - b : bias (편향)
  - 선형 회귀란 학습 데이터와 가장 잘 맞는 하나의 직선을 찾는 일
- X : Feature / 독립변수
- Y : Target / 종속변수
- 머신 러닝에서 식을 세울 때 식을 가설(Hypothesis)이라 한다.

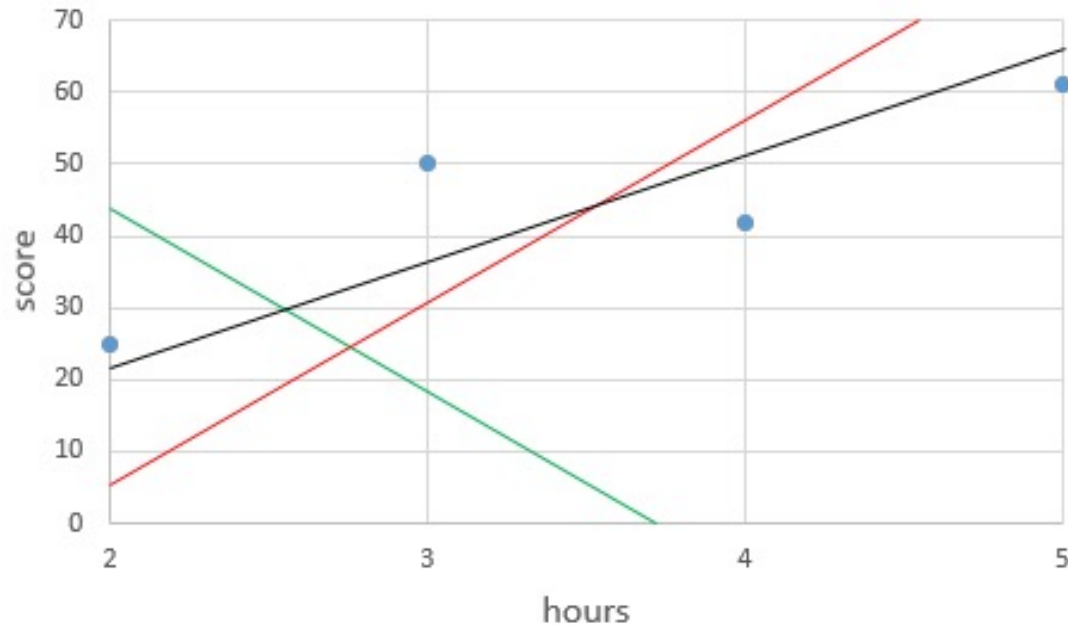
# 선형 회귀(Linear Regression)

- Cost Function
  - 비용 함수(cost function) = 손실 함수(loss function)  
= 오차 함수(error function) = 목적 함수(objective function)



# 선형 회귀(Linear Regression)

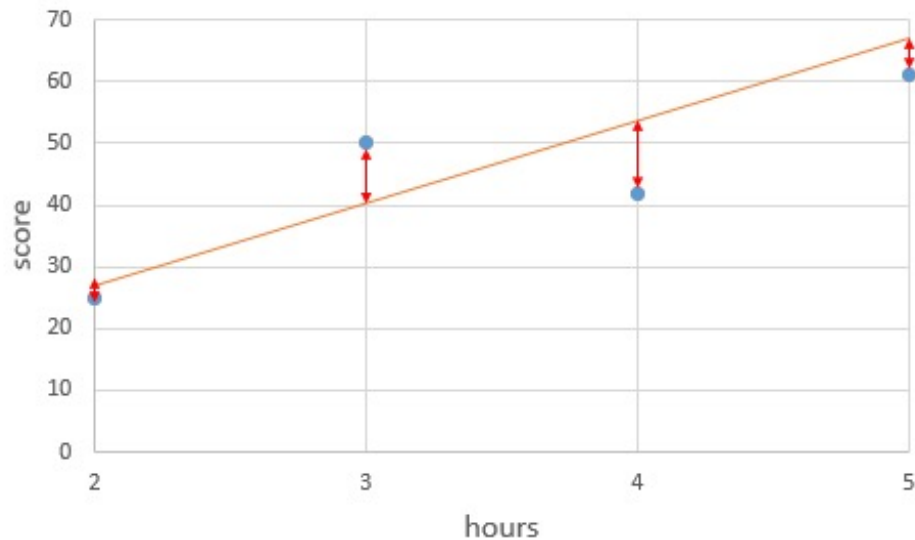
- Cost Function
  - 목표 : 4개의 점을 가장 잘 표현하는 직선을 그리는 일



# 선형 회귀(Linear Regression)

## Cost Function

- 오차(error) : 실제 값(4개의 점)과 직선의 예측 값(동일한 x값에서의 직선의 y값)에 대한 값의 차이를 빨간색 화살표 ↓로 표현한 것
- 주황색 식 :  $y = 13x + 1$



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

# 선형 회귀(Linear Regression)

## Cost Function

- 평균 제곱 오차(Mean Squared Error, MSE) : 오차의 제곱 합에 대한 평균
- Linear Regression에서 학습의 의미 :
  - Train 데이터를 기반으로 Cost(W, b) 를 최소화하는 W와 b를 찾는 것

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

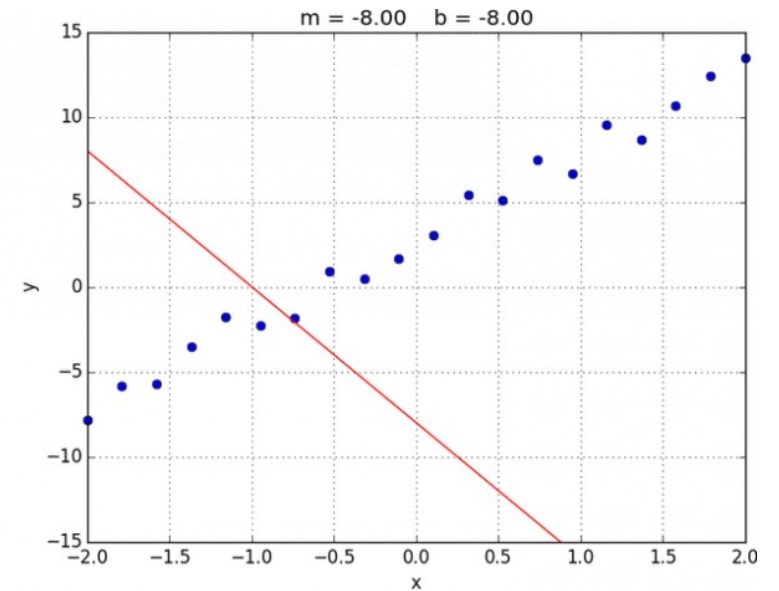
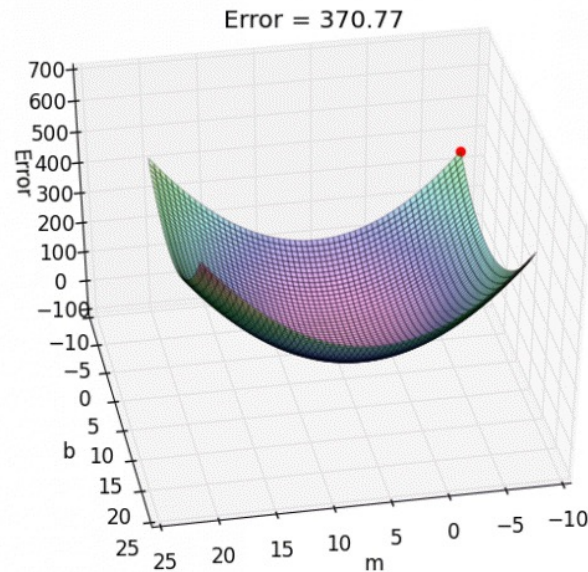
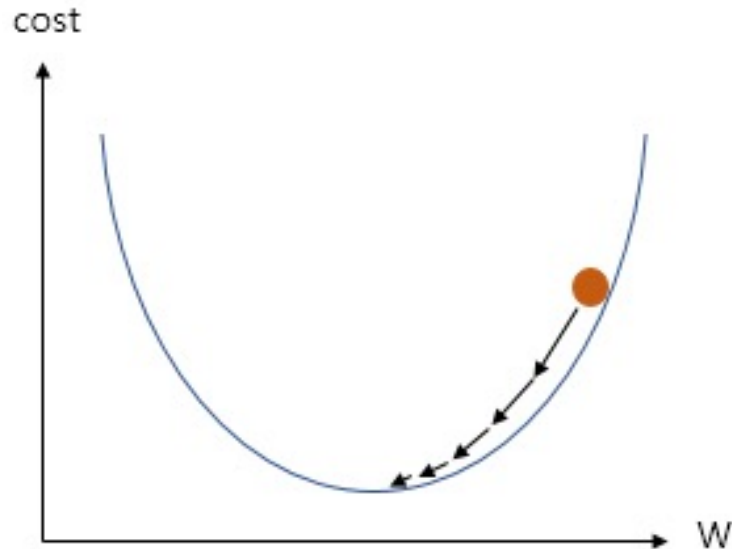
$$cost(W, b) = \frac{1}{n} \sum_{i=1}^n \left[ y^{(i)} - H(x^{(i)}) \right]^2$$

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

# 선형 회귀(Linear Regression)

## 경사 하강법(Gradient Descent)

- 최적화 이론(Optimization) / 비용 함수(Cost Function)의 값을 최소로 하는  $W$ 와  $b$ 를 찾는 방법
- Optimizer 알고리즘을 통해 적절한  $W$ 와  $b$ 를 찾아내는 과정을 학습(training)

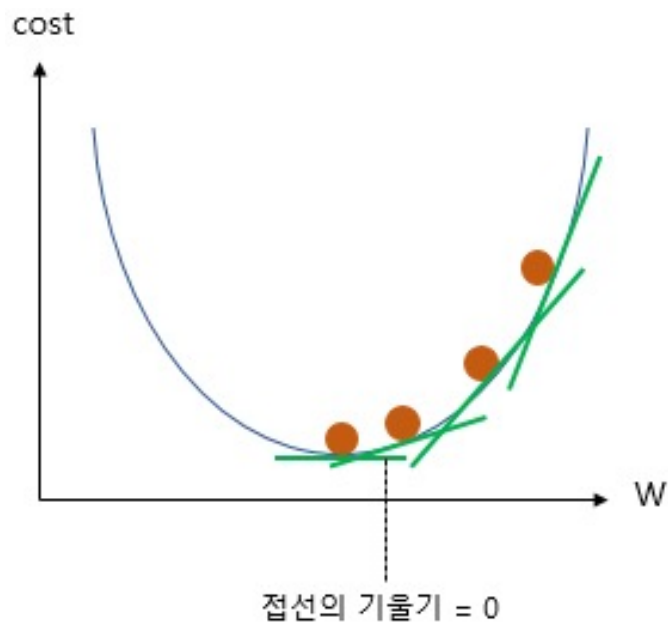




# 선형 회귀(Linear Regression)

## 경사 하강법(Gradient Descent)

- 임의의 초기값  $W$  값을 정한다
- 현재  $\text{Cost}(W, b)$ 에 대한  $W$ 의 기울기를 구한다.
- 맨 아래의 볼록한 부분에서는 결국 접선의 기울기 = 0
- 학습률(learning rate) :  $\alpha$  얼마나 학습할 지 결정

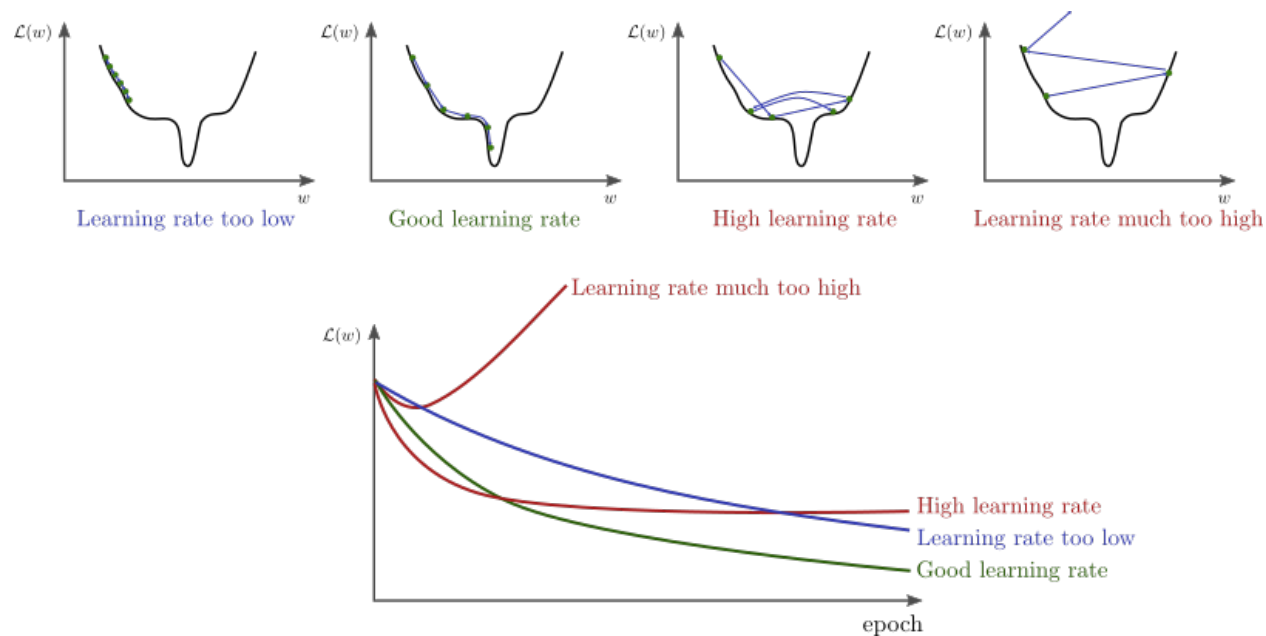


$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$

# 선형 회귀(Linear Regression)

## 경사 하강법(Gradient Descent)

- Learning Rate 설정
  - 하이퍼 파라미터 → 모델 선택
  - 정답은 없다. 적절한 값을 설정한다



# 선형 회귀(Linear Regression)

## Linear Regression 실습

[https://github.com/wooridle/mlp\\_lecture/blob/main/1.linear\\_regression.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/1.linear_regression.ipynb)

# 다중 선형 회귀(Multivariable)

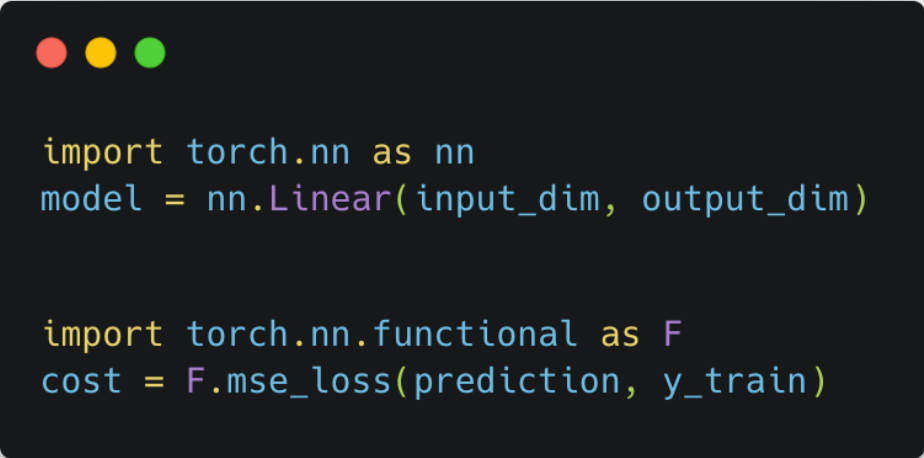
## Multivariable Linear Regression

- 독립변수  $X$  (Feature)가 여러 개인 선형 회귀
- 실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/2.multivariable\\_LR.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/2.multivariable_LR.ipynb)

# Pytorch nn.Module

nn.Module을 사용하여 단순 선형, 다중 선형 회귀 구하기

- Weight, bias를 직접 선언할 필요가 없다
- nn.Linear(input\_dim, output\_dim) : <https://pytorch.org/docs/1.9.1/generated/torch.nn.Linear.html>
- 실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/3.nn\\_Module.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/3.nn_Module.ipynb)



```
import torch.nn as nn
model = nn.Linear(input_dim, output_dim)

import torch.nn.functional as F
cost = F.mse_loss(prediction, y_train)
```

# 선형 모델 Class로 구현하기

선형회귀 모델을 Class형태로 구현

- Pytorch로 구현된 대부분 모델은 Class형태로 구현되어 있음
- 실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/%084.class\\_model.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/%084.class_model.ipynb)

# **이진 분류(Logistic Regression)**

# 이진 분류(Logistic Regression)

Logistic Regression?

- Linear Regression Hypothesis :  $H(x) = Wx + b$
- Logistic Regression Hypothesis :  $H(x) = \text{Sigmoid}(Wx + b)$
- LR 출력값을 Logistic 함수로 맵핑 한 것

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

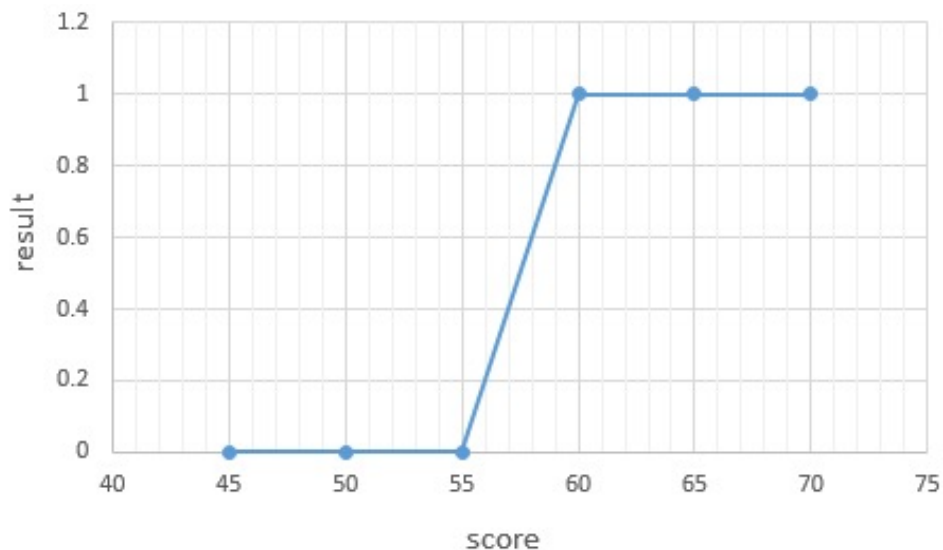


# 이진 분류(Logistic Regression)

## Logistic Regression?

- Y(target) : 목표 값(정답/레이블)이 다름
  - 선형 회귀 : 점수(score) 실수 (Regression)
  - Logistic 회귀 : 불/합 카테고리 (Classification)

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



# 이진 분류(Logistic Regression)

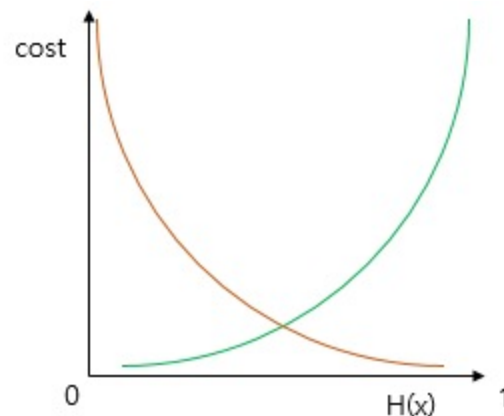
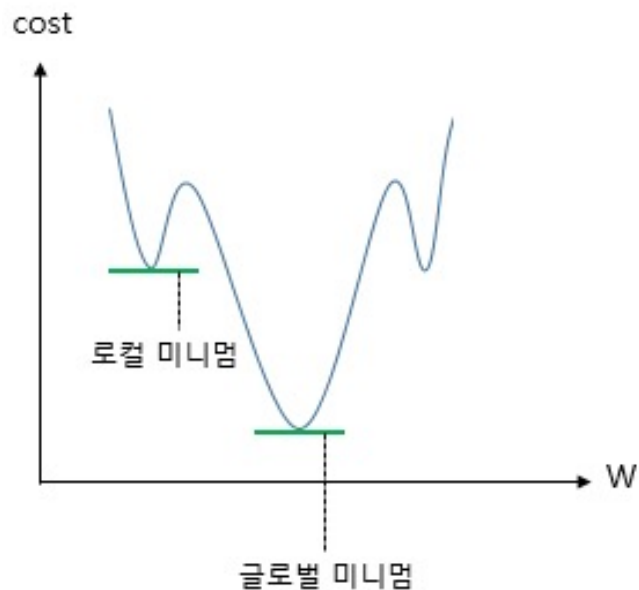
## Cost Function

- 선형 회귀에서 사용한 Mean Square Error를 사용하면 Local Minimal에 빠진다
  - 새로운 Cost Function이 필요
  - 설계 : 정답과 예측 값이 가까우면 오차를 작게, 멀어지면 크게

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

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

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



# 이진 분류(Logistic Regression)

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/5.logistic\\_regression.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/5.logistic_regression.ipynb)

# Multi Class Classification 멀티 클래스 분류기

# Multi Class Classification

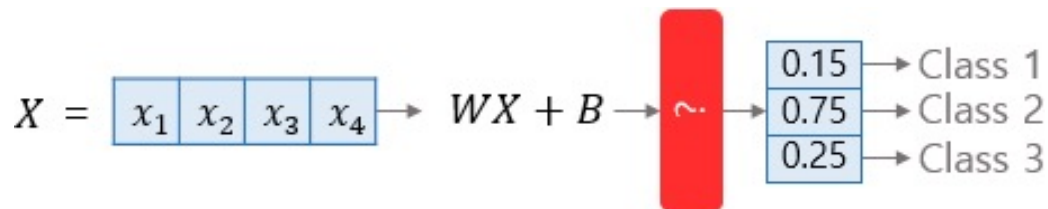
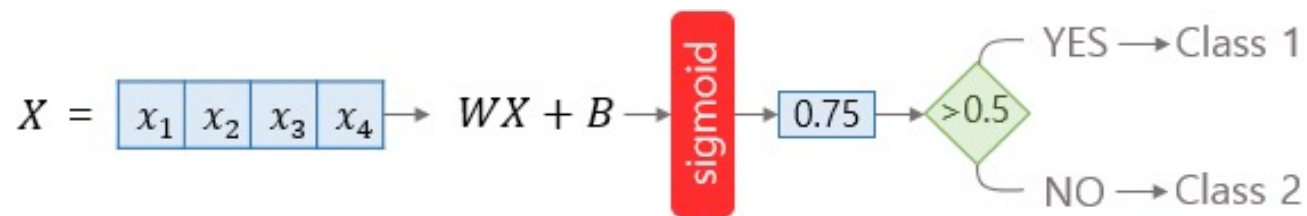
## Multi Class Classification

- 세 개 이상의 답 중 하나를 고르는 문제를 다중 클래스 분류

- Y(target) : 3개 이상 카테고리

- Binary Classification :  $H(X) = \text{sigmoid}(WX + B)$

- Multi Class Classification :  $H(X) = \text{softmax}(WX + B)$   $p_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$  for  $i = 1, 2, \dots, k$



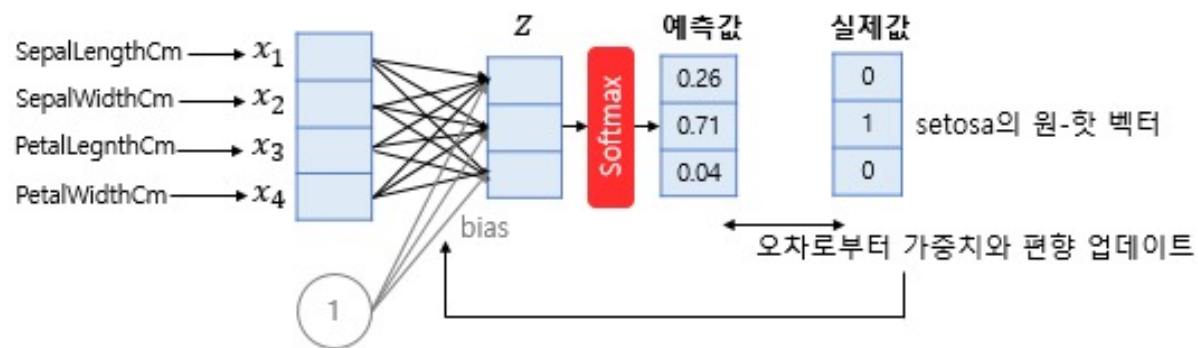
# Multi Class Classification

$$\text{softmax} \left( \begin{matrix} W \\ c \times f \end{matrix} \times \begin{matrix} X \\ f \times 1 \end{matrix} + \begin{matrix} B \\ c \times 1 \end{matrix} \right) = \begin{matrix} \text{예측값} \\ c \times 1 \end{matrix}$$

$$\hat{Y} = \text{softmax}(XW + B)$$

$$\begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \\ y_{41} & y_{42} & y_{43} \\ y_{51} & y_{52} & y_{53} \end{pmatrix} = \text{softmax} \left( \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \\ x_{51} & x_{52} & x_{53} & x_{54} \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} + \begin{pmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{pmatrix} \right)$$


## 학습 과정



# Multi Class Classification

## Cost Function

- Cross Entropy 함수

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


실제값 원-핫 벡터의 j번째 인덱스

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

N개 데이터에 대한 평균

- 이진 분류에서 Cross Entropy 함수

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

# Multi Class Classification

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/6\\_multi\\_class\\_classification.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/6_multi_class_classification.ipynb)

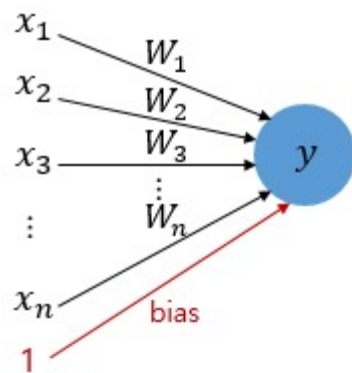
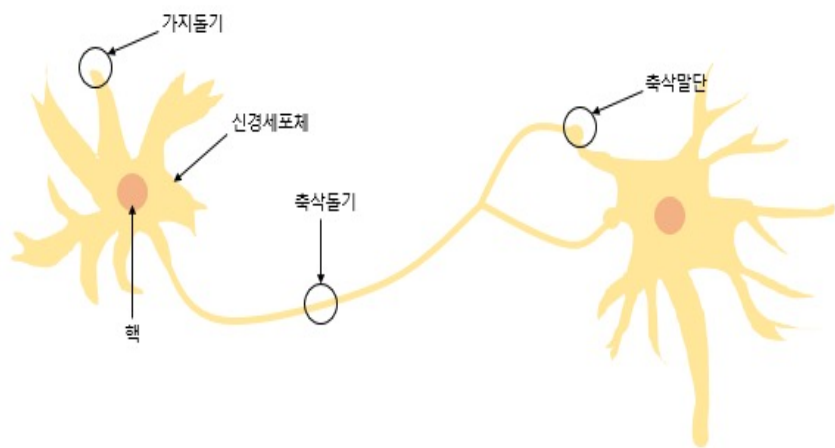


# **Neural Network : Multi Layer Perceptron**

# Multi Layer Perceptron

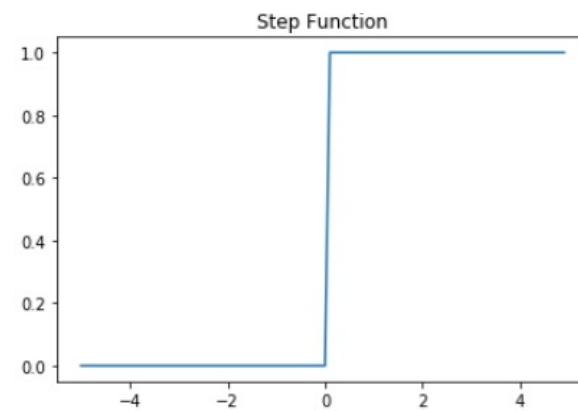
## 퍼셉트론(Perceptron)

- 1957년에 제안한 초기 형태의 인공 신경망으로 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
- 입력값과 가중치의 곱의 전체 합이 임계치(threshold)를 넘으면 종착지에 있는 인공 뉴런은 출력 신호로서 1을 출력



$$\text{if } \sum_i^n W_i x_i + b \geq 0 \rightarrow y = 1$$

$$\text{if } \sum_i^n W_i x_i + b < 0 \rightarrow y = 0$$



# Multi Layer Perceptron

## 단층 퍼셉트론(Single-Layer Perceptron)

- 단층 퍼셉트론은 값을 보내는 단계, 값을 받아서 출력하는 단계 2개로만 구성
- 단층 퍼셉트론을 이용하면 AND, NAND, OR 게이트를 쉽게 구현 가능

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

AND 게이트

```
def AND_gate(x1, x2):  
    w1=0.5  
    w2=0.5  
    b=-0.7  
    result = x1*w1 + x2*w2 + b  
    if result <= 0:  
        return 0  
    else:  
        return 1
```

AND\_gate(0, 0), AND\_gate(0, 1), AND\_gate(1, 0), AND\_gate(1, 1)

$x_1$	$x_2$	$y$
0	0	1
0	1	1
1	0	1
1	1	0

NAND 게이트

```
def NAND_gate(x1, x2):  
    w1=-0.5  
    w2=-0.5  
    b=0.7  
    result = x1*w1 + x2*w2 + b  
    if result <= 0:  
        return 0  
    else:  
        return 1
```

NAND\_gate(0, 0), NAND\_gate(0, 1), NAND\_gate(1, 0), NAND\_gate(1, 1)

# Multi Layer Perceptron

## 단층 퍼셉트론(Single-Layer Perceptron)

- 단층 퍼셉트론은 값을 보내는 단계, 값을 받아서 출력하는 단계 2개로만 구성
- 단층 퍼셉트론을 이용하면 AND, NAND, OR 게이트를 쉽게 구현 가능

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

OR 게이트

```
def OR_gate(x1, x2):
```

```
    w1=0.6
```

```
    w2=0.6
```

```
    b=-0.5
```

```
    result = x1*w1 + x2*w2 + b
```

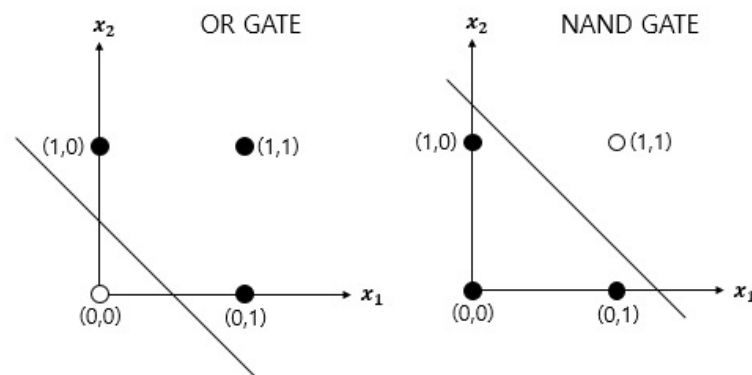
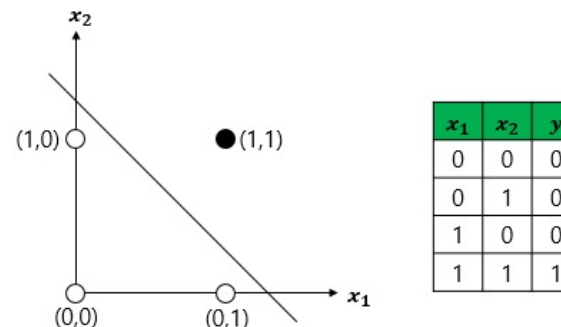
```
    if result <= 0:
```

```
        return 0
```

```
    else:
```

```
        return 1
```

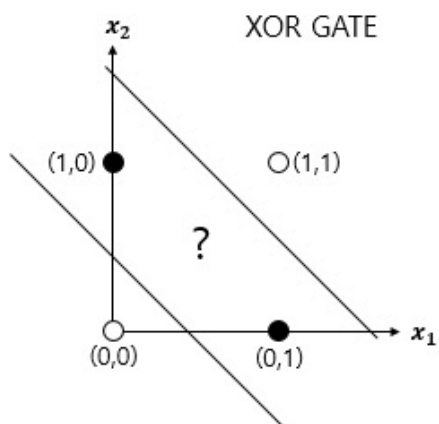
OR\_gate(0, 0), OR\_gate(0, 1), OR\_gate(1, 0), OR\_gate(1, 1)



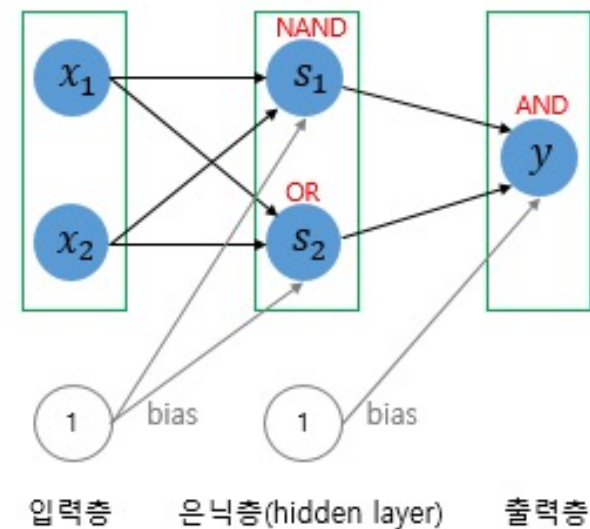
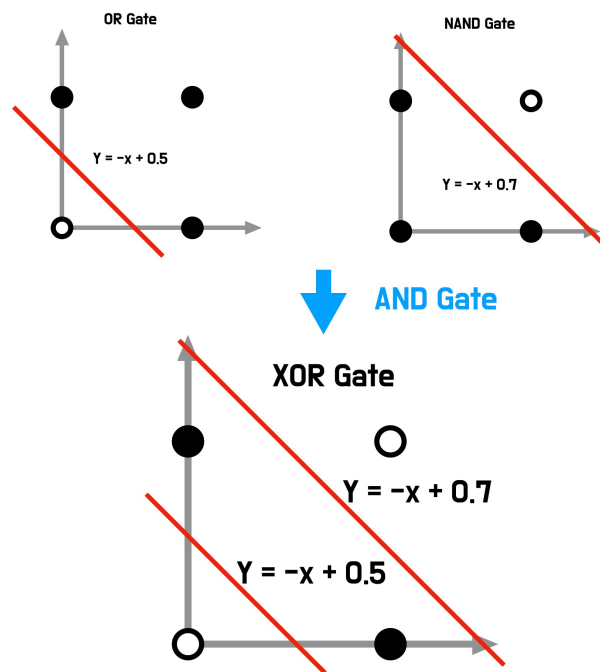
# Multi Layer Perceptron

## 단층 퍼셉트론(Single-Layer Perceptron)

- XOR 게이트는 단층 퍼셉트론으로 풀 수 없다. 단층 퍼셉트론의 한계
- 단층 퍼셉트론을 여러개 쌓아서 풀 수 있지 않을까?
  - Multi Layer Perceptron

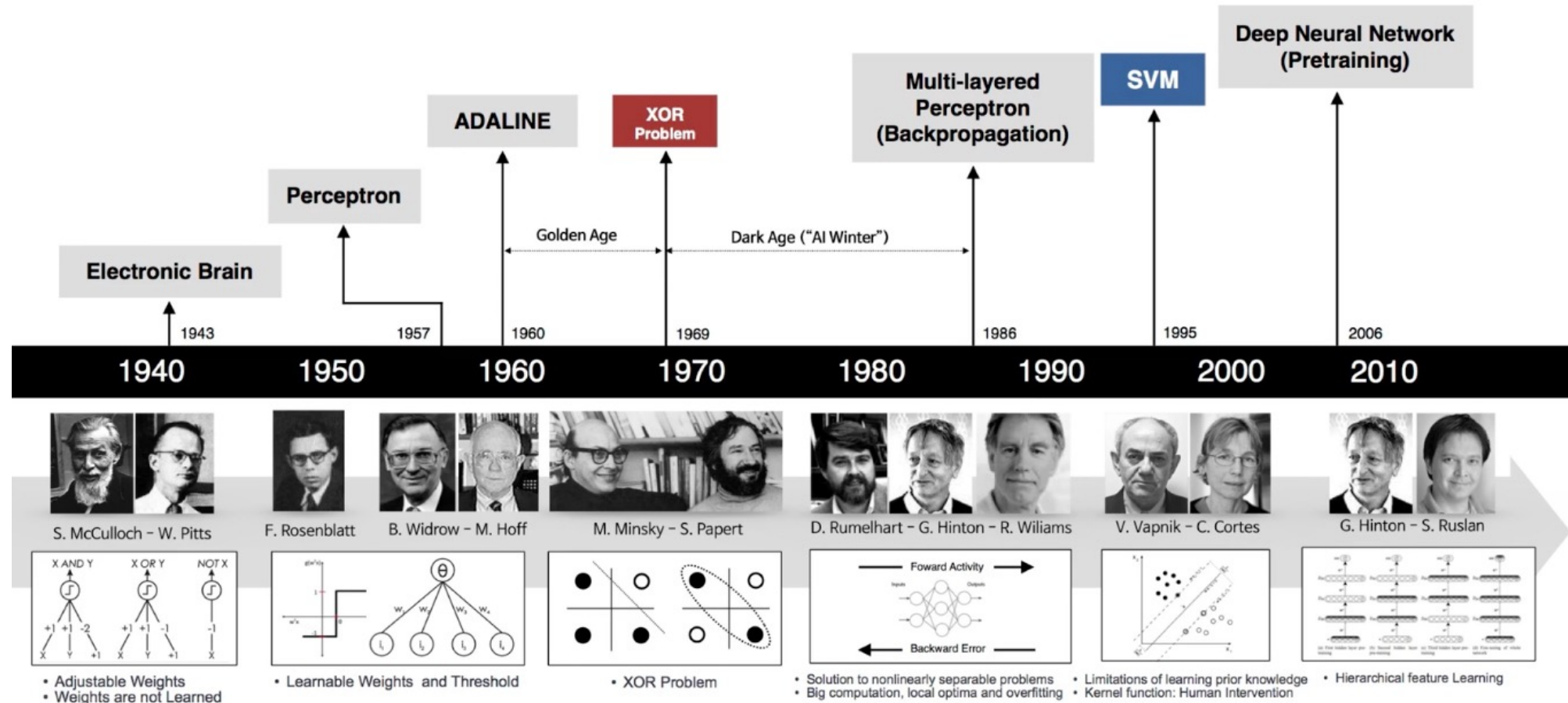


$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0



# Multi Layer Perceptron

## History of Neural Network



# Multi Layer Perceptron

단층 퍼셉트론(Single-Layer Perceptron)

XOR 문제 - 단층 퍼셉트론 구현

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/7.single\\_layer\\_perceptron\\_xor.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/7.single_layer_perceptron_xor.ipynb)

# Multi Layer Perceptron

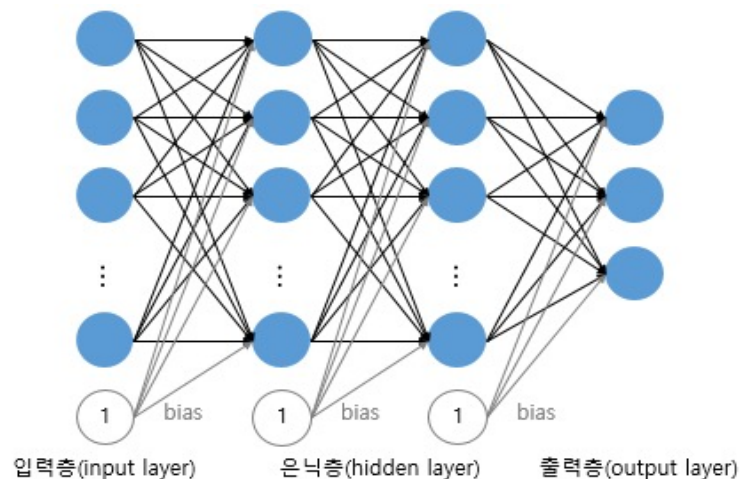
## 다층 퍼셉트론(Multi-Layer Perceptron)

- 은닉층(hidden layer)
- 비선형성(Nonlinearity)
  - 입력들 사이의 복잡한 상호 작용 추가
  - 비선형이 아니면, 여러 층을 쌓아도 결국 1개의 층으로 표현 가능

$$\begin{aligned} \mathbf{h} &= \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \\ \mathbf{o} &= \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2 \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{o}) \end{aligned}$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2 = \mathbf{W}_2 (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (\mathbf{W}_2 \mathbf{W}_1) \mathbf{x} + (\mathbf{W}_2 \mathbf{b}_1 + \mathbf{b}_2) = \mathbf{W} \mathbf{x} + \mathbf{b}$$

단층 신경망은 선형모델



$$\begin{aligned} \mathbf{h} &= \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ \mathbf{o} &= \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2 \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{o}) \end{aligned}$$

비선형함수  $\sigma$  맵핑

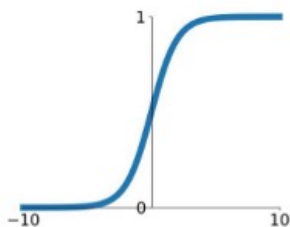


# Multi Layer Perceptron

## 활성화 함수(Activation Function)

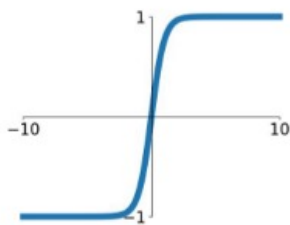
### Sigmoid

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



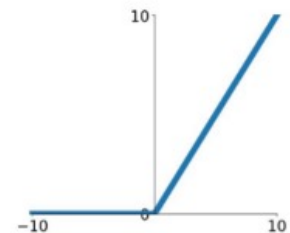
### tanh

$$\tanh(x)$$



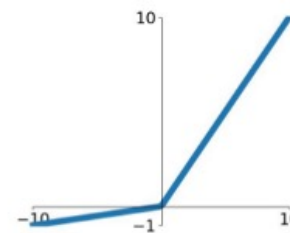
### ReLU

$$\max(0, x)$$



### Leaky ReLU

$$\max(0.1x, x)$$

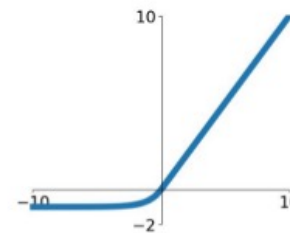


### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

### ELU

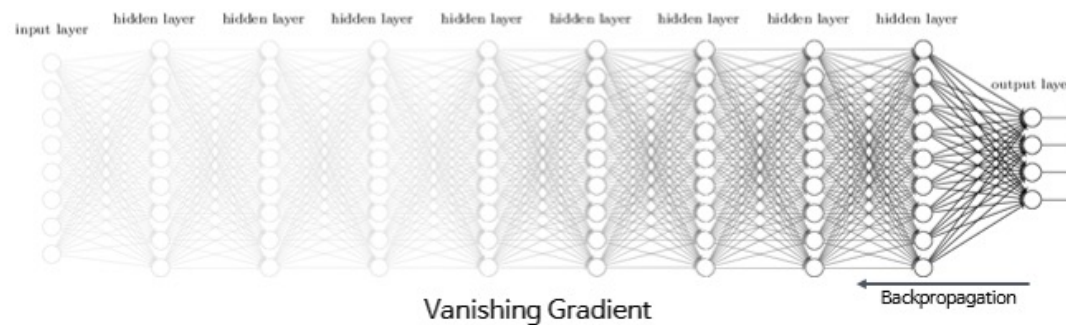
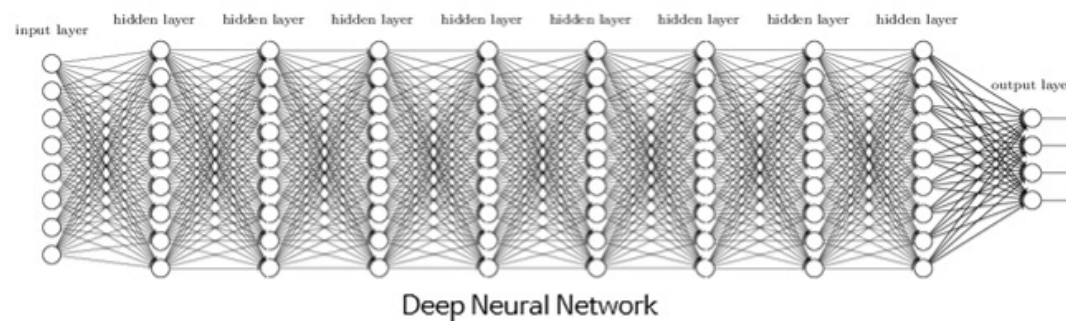
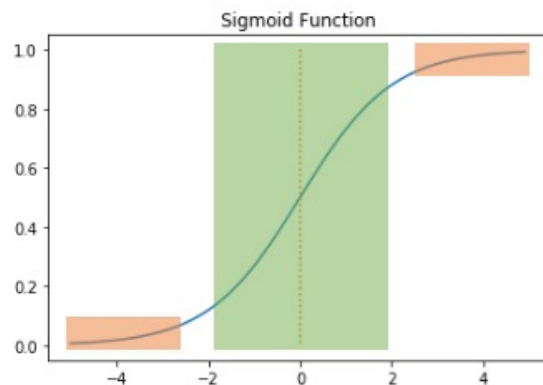
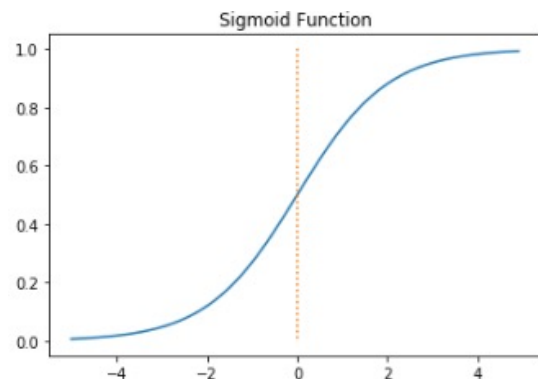
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Multi Layer Perceptron

## 시그모이드 함수(Sigmoid function)

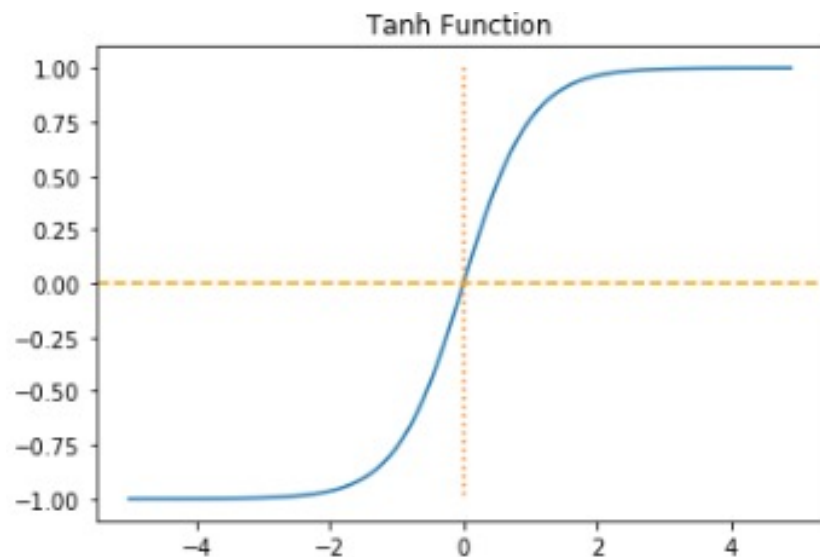
- 입력 값을 0과 1 사이의 값으로 변환
- 기울기 소실(Vanishing Gradient) 문제



# Multi Layer Perceptron

하이퍼볼릭탄젠트 함수(Hyperbolic tangent function)

- $\tanh$ 는 입력값을 -1과 1사이의 값으로 변환
- 0을 중심
- 시그모이드 함수보다 반환값의 변화폭이 큼

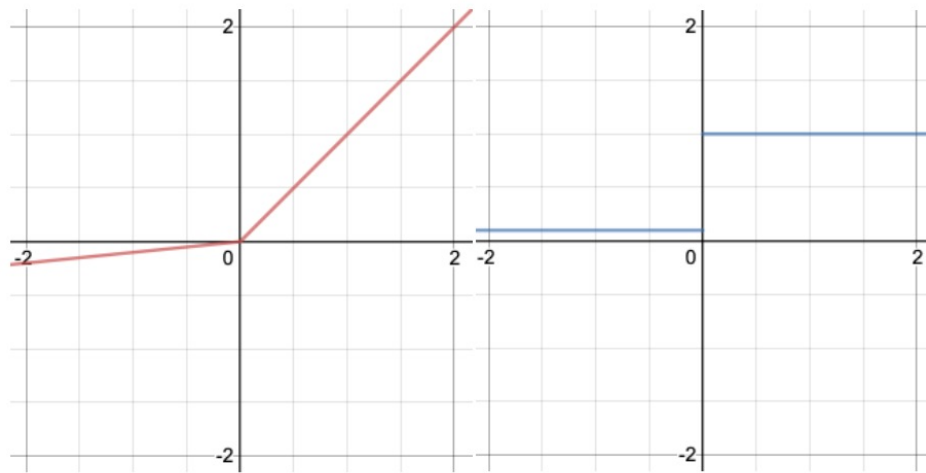


# Multi Layer Perceptron

## LeakyReLU

- 입력 값이 음수일 때 완만한 선형 함수
- 일반적으로 알파를 0.01로 설정

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

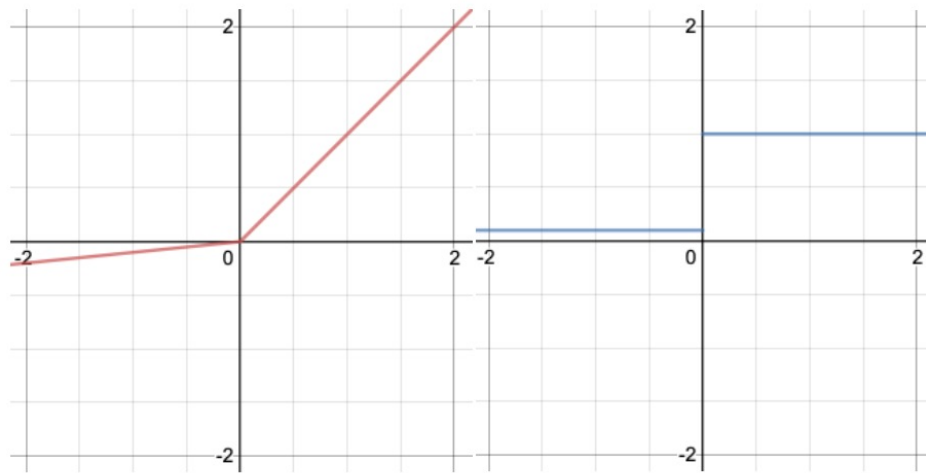


# Multi Layer Perceptron

## LeakyReLU

- 입력 값이 음수일 때 완만한 선형 함수
- 일반적으로 알파를 0.01로 설정

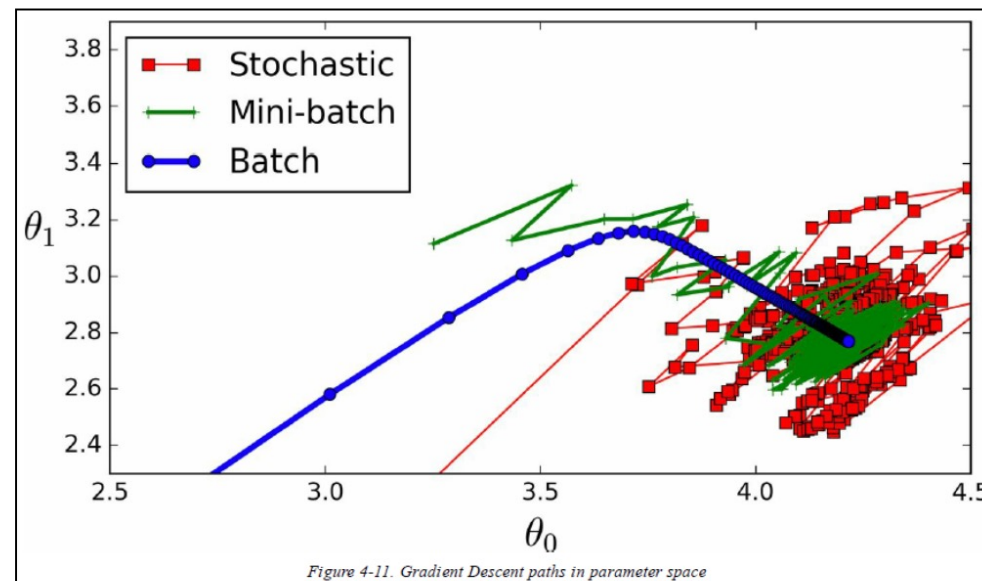
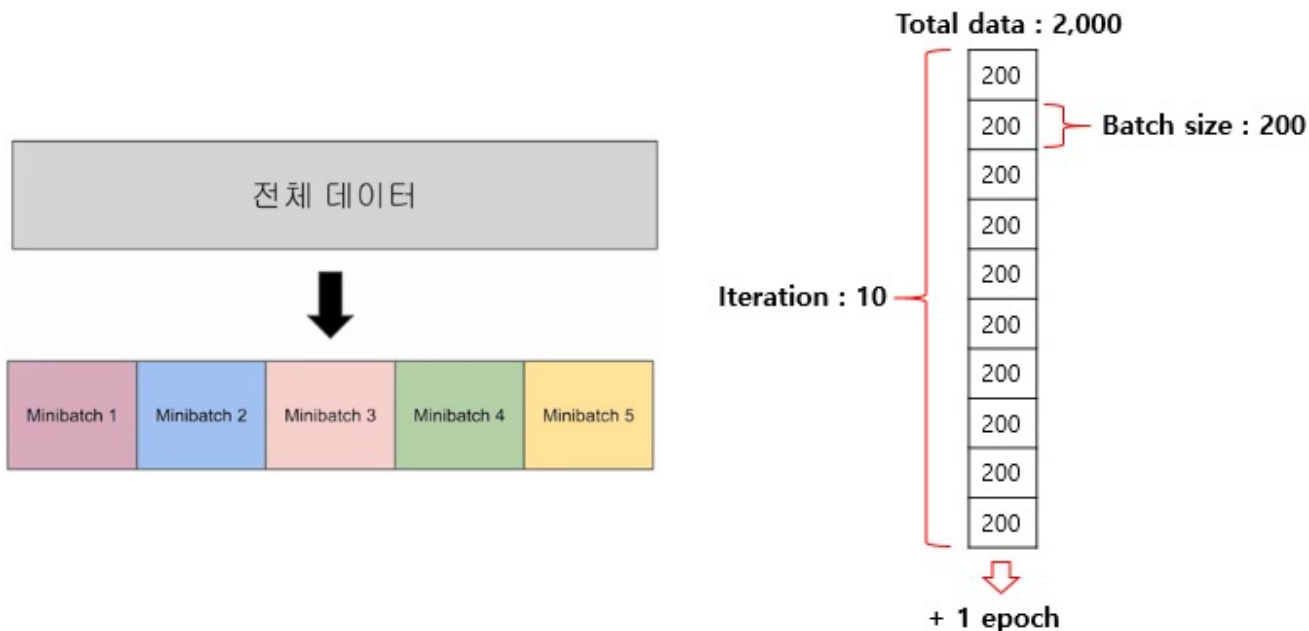
$$f(x) = \begin{cases} x & (x > 0) \\ \alpha x & (x \leq 0) \end{cases} \quad f(\alpha, x) = \begin{cases} 1 & (x > 0) \\ \alpha & (x \leq 0) \end{cases}$$



# Multi Layer Perceptron

## Batch / Mini-Batch / Stochastic Gradient Descent

- Stochastic Gradient Descent : 데이터 1개 씩 샘플링 후 학습
- Batch : 전체 데이터 셋을 한번에 학습 → 학습 관점에서선 좋지만, GPU 메모리에 올릴 수 없다.
- Mini-Batch : 일반적인 접근 법 batch size만큼 샘플링



# Multi Layer Perceptron

다층 퍼셉트론(Multi-Layer Perceptron)

XOR 문제 - 다층 퍼셉트론 구현

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/8.multi\\_layer\\_perceptron\\_xor.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/8.multi_layer_perceptron_xor.ipynb)

# Multi Layer Perceptron

다층 퍼셉트론(Multi-Layer Perceptron)

MNIST 분류하기

실습 : [https://github.com/wooridle/mlp\\_lecture/blob/main/9.mnist\\_mlp.ipynb](https://github.com/wooridle/mlp_lecture/blob/main/9.mnist_mlp.ipynb)



# Machine Learning Process

