



# Perceptron: XOR 문제

머신러닝 / 딥러닝

임 경 태

Week	Chapter	Contents
1	1, 2장	강의 소개, 파이썬 복습
2	1, 3장	파이썬 복습, Numpy, Pandas
3	1, 4장	딥러닝을 위한 미분
4	5장	회귀
5	5장	분류
6	6장	Perceptron: XOR문제
7	7장	딥러닝
8	1~7장	중간고사
9	8장	MNIST 필기체 구현 (팀 프로젝트 공지)
10	9장	오차역전파
11	11장	합성곱 신경망(CNN)
12	12장	순환 신경망(RNN)
13	10장	자율주행 (Collision Avoidance, Transfer Learning)
14	11장	자율주행 (Load Following)
15	8~12장	기말고사 (or 프로젝트 발표)

---

# CONTENTS

---

- ① AND, NAND, OR GATE
- ② 퍼셉트론과 논리회로 구현
- ③ 지도학습과 논리회로 구현
- ④ 신경망



목적 : XOR게이트 구현 시 문제점과 딥러닝의 관계를 이해한다.



목표 : 딥러닝을 사용하는 이유를 이해한다.



내용 : XOR 게이트와 신경망

# CONTENTS

---

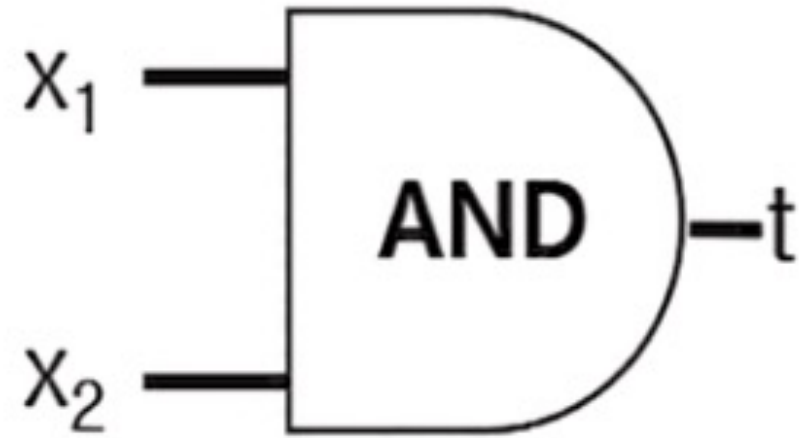
- ① **AND, NAND, OR GATE**
- ② 퍼셉트론과 논리회로 구현
- ③ 지도학습과 논리회로 구현
- ④ 신경망

### 📌 인공지능을 가능하다면 컴퓨터도 자동으로 제어할 수 있어야 하지 않을까?

- 논리 회로(영어: logic gate)는 불 대수를 물리적 장치에 구현한 것으로,  
하나 이상의 논리적 입력값에 대해 논리 연산을 수행 출력
- 논리회로의 연산을 자동으로 조합하면 생각하는 기계를 만들 수 있다고 생각함.  
따라서, 논리회로를 기계학습으로 예측 하는 것이 큰 화두였음.

**AND**

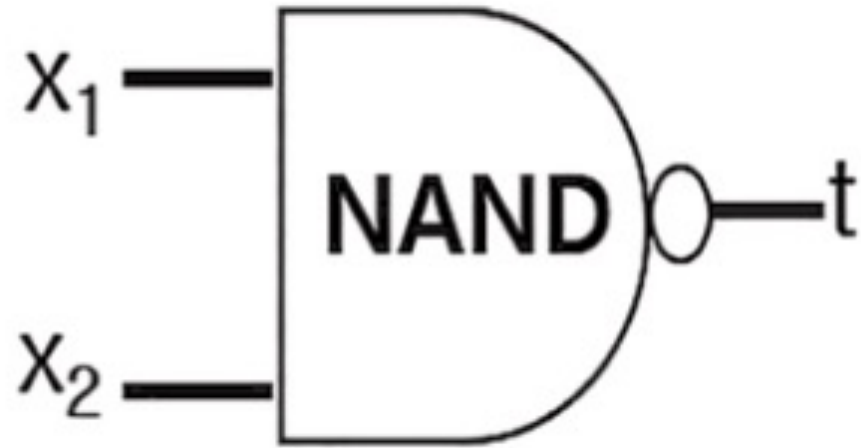
$X_1$	$X_2$	$t$
0	0	0
0	1	0
1	0	0
1	1	1



$$t = X_1 X_2$$

**NAND**

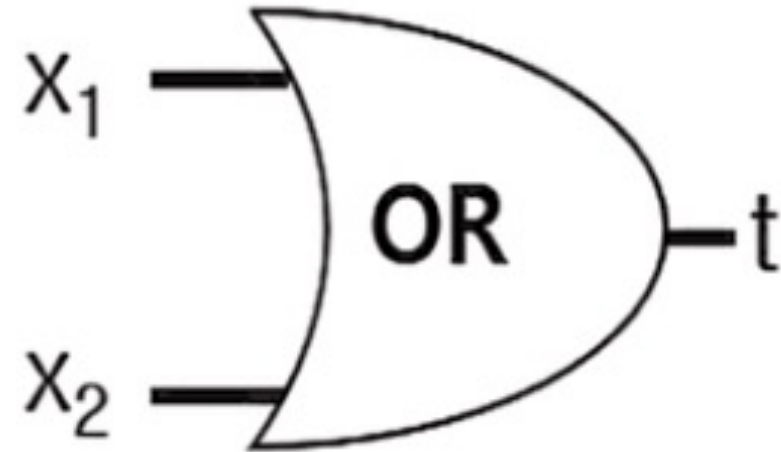
$X_1$	$X_2$	$t$
0	0	1
0	1	1
1	0	1
1	1	0



$$t = \overline{X_1 X_2}$$



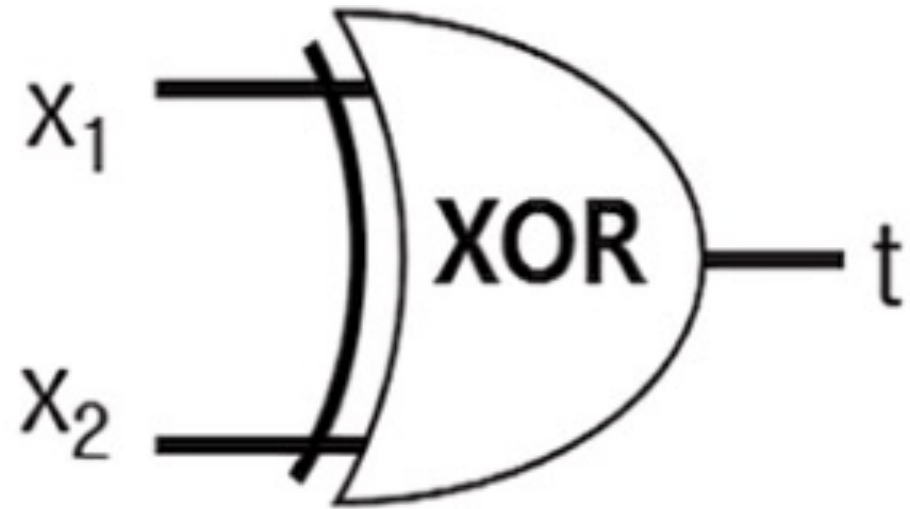
OR		
$X_1$	$X_2$	$t$
0	0	0
0	1	1
1	0	1
1	1	1



$$t = X_1 + X_2$$

## XOR

$X_1$	$X_2$	$t$
0	0	0
0	1	1
1	0	1
1	1	0



$$t = X_1 \oplus X_2$$

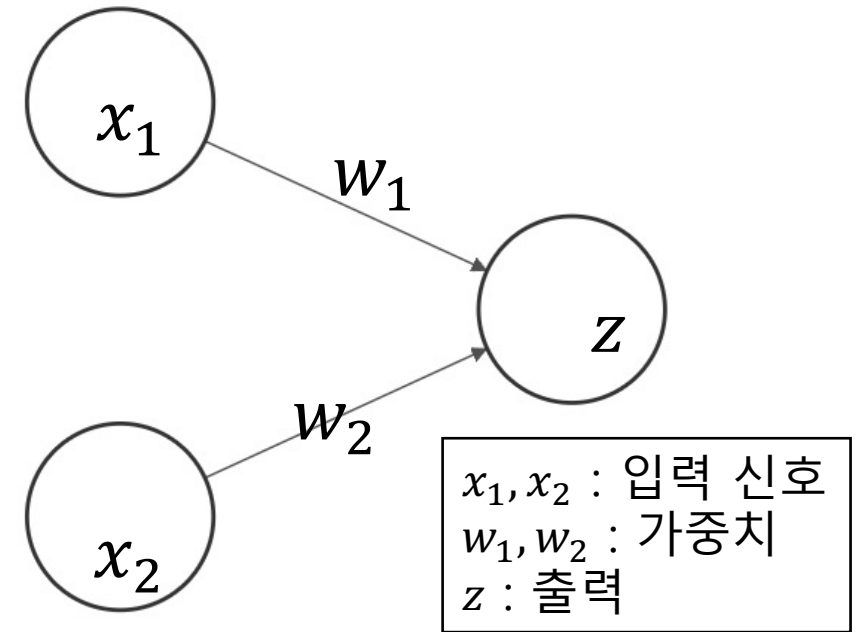
# CONTENTS

---

- ① AND, NAND, OR GATE
- ② 퍼셉트론과 논리회로 구현
- ③ 지도학습과 논리회로 구현
- ④ 신경망

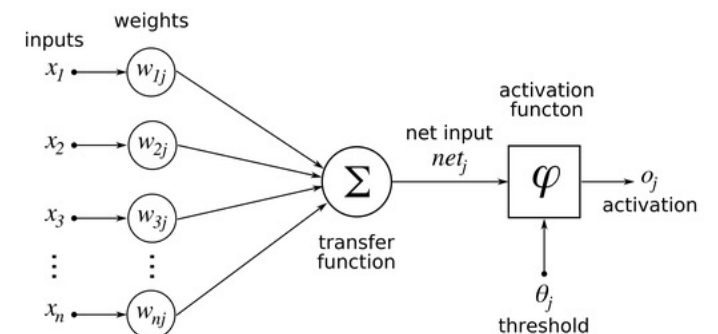
## Perceptron

- 신경망(딥러닝)의 기원이 되는 알고리즘
- 입력(신호)을 받아 한 개의 출력을 내보냄
- 여기에서 가중치  $w_1, w_2$ 는 상수로 직접 정함.



- 퍼셉트론은 뉴런에서 보내온 신호의 총 합( $w_1x_1 + w_2x_2$ )이 임계 값( $\theta$ )을 넘을 때 1을 출력, 넘지 못하면 0을 출력

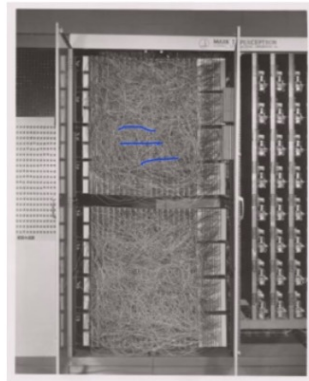
$$\rightarrow z = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$



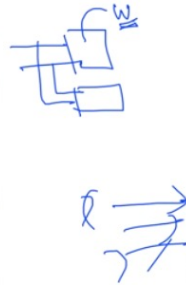
## Perceptron

- 퍼셉트론은 뉴런에서 보내온 신호의 총 합( $w_1x_1 + w_2x_2$ )이 임계 값( $\theta$ )을 넘을 때 1을 출력, 넘지 못하면 0을 출력

### Hardware implementations



Frank Rosenblatt, ~1957: Perceptron



Widrow and Hoff, ~1960: Adaline/Madaline

Hand-drawn blue schematic diagram showing a single node with multiple inputs, with a label 'y' indicating the output.

## Perceptron기반 논리회로 구현 (AND)

- Perceptron을 통해 AND게이트를 구현하기 위해서는 다음 표를 만족하도록 적당한 가중치와 임계값 ( $w_1, w_2, \theta$ )를 선택해야 한다. 이를 만족하는 순서쌍은 무수히 많다.

AND		
$X_1$	$X_2$	$t$
0	0	0
0	1	0
1	0	0
1	1	1

```
1 def AND(x_1, x_2):
2     w_1, w_2, theta = 0.5, 0.5, 0.8
3     temp = w_1*x_1 + w_2*x_2
4
5     if temp > theta:
6         return 1
7     else:
8         return 0
```

```
1 AND(0, 0) # 0 출력
2 AND(0, 1) # 0 출력
3 AND(1, 0) # 0 출력
4 AND(1, 1) # 1 출력
```

## Perceptron기반 논리회로 구현 (NAND)

- Perceptron을 통해 NAND게이트를 구현하기 위해서는 다음 표를 만족하도록 적당한 가중치와 임계값 ( $w_1, w_2, \theta$ )를 선택해야 한다. 이를 만족하는 순서쌍은 무수히 많다.

NAND

$X_1$	$X_2$	$t$
0	0	1
0	1	1
1	0	1
1	1	0

```
1 def NAND(x_1, x_2):
2     w_1, w_2, theta = -0.5, -0.5,
3     -0.8
4     temp = w_1*x_1 + w_2*x_2
5     if temp > theta:
6         return 1
7     else:
8         return 0
```

```
1 NAND(0, 0) # 1 출력
2 NAND(0, 1) # 1 출력
3 NAND(1, 0) # 1 출력
4 NAND(1, 1) # 0 출력
```

## Perceptron기반 논리회로 구현 (OR)

- Perceptron을 통해 OR게이트를 구현하기 위해서는 다음 표를 만족하도록 적당한 가중치와 임계값 ( $w_1, w_2, \theta$ )를 선택해야 한다. 이를 만족하는 순서쌍은 무수히 많다.

OR		
$X_1$	$X_2$	$t$
0	0	0
0	1	1
1	0	1
1	1	1

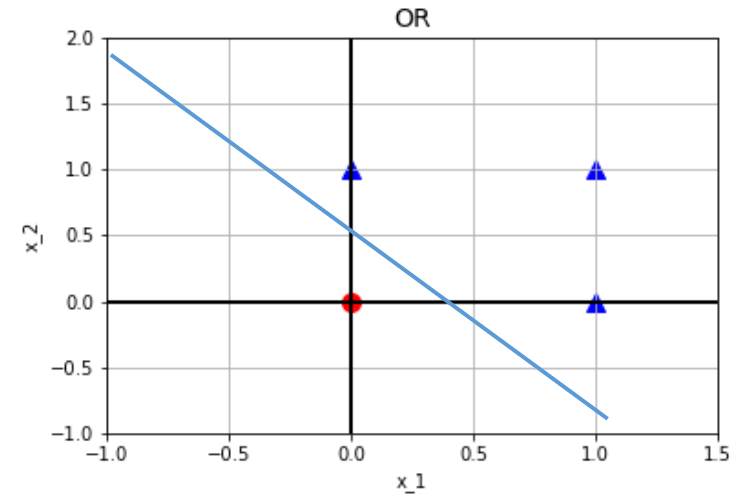
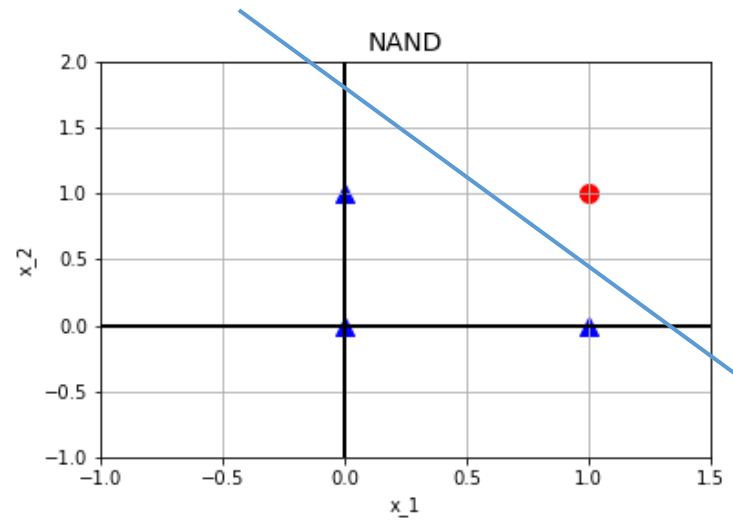
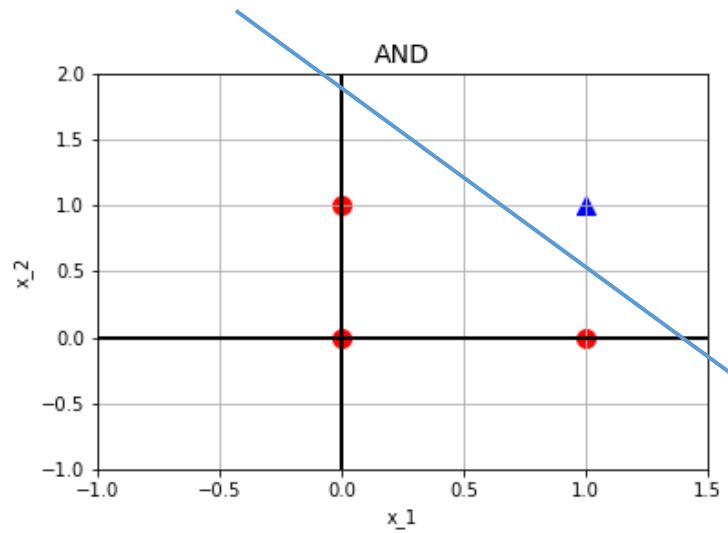
```
1 def OR(x_1, x_2):
2     w_1, w_2, theta = 0.5, 0.5, 0.2
3     temp = w_1*x_1 + w_2*x_2
4
5     if temp > theta:
6         return 1
7     else:
8         return 0
```

```
1 OR(0, 0) # 0출력
2 OR(0, 1) # 1출력
3 OR(1, 0) # 1출력
4 OR(1, 1) # 1출력
```



# Perceptron기반 논리회로 구현

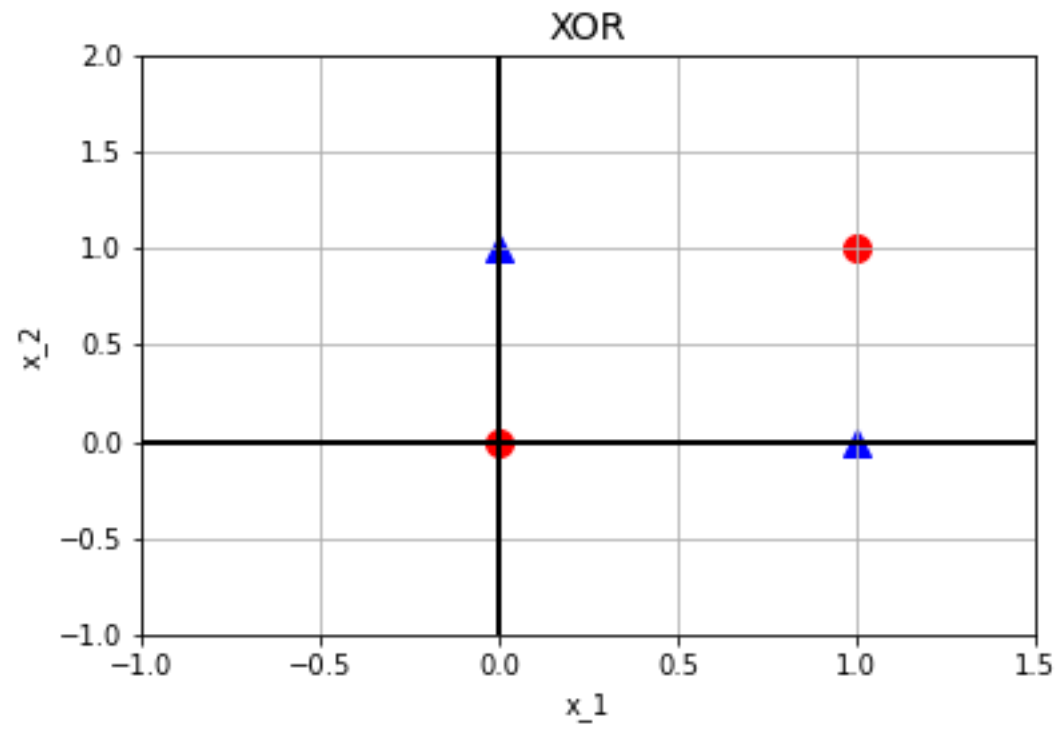
- 퍼셉트론의 식  $t = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$  을 살펴보면 퍼셉트론은 직선으로 두 영역을 나눕니다.



## Perceptron기반 논리회로 구현 (XOR)

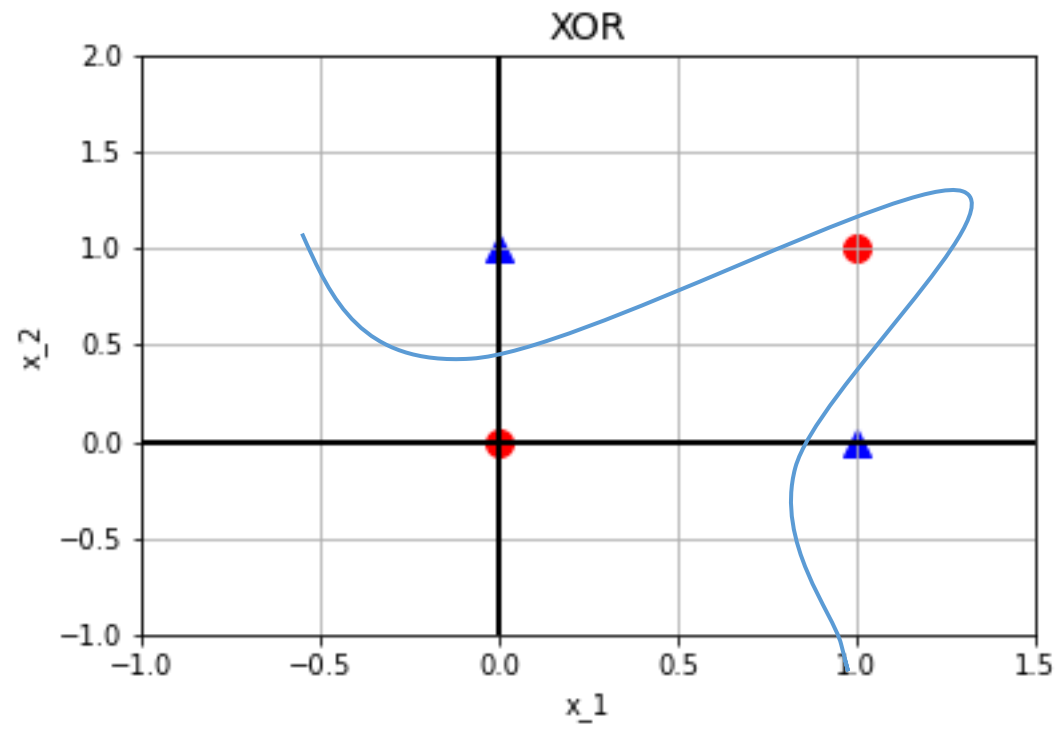
---

- XOR게이트의 그래프를 확인해보면 한 개의 직선으로 나눌 수 없음!



## Perceptron기반 논리회로 구현 (XOR)

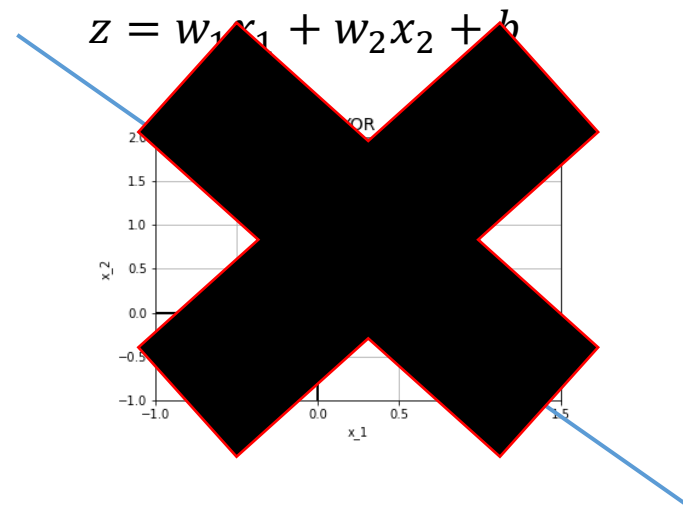
- XOR게이트의 그래프를 확인해보면 한 개의 직선으로 나눌 수 없음! 곡선이 필요  
→ Nonlinearity 추가 필요



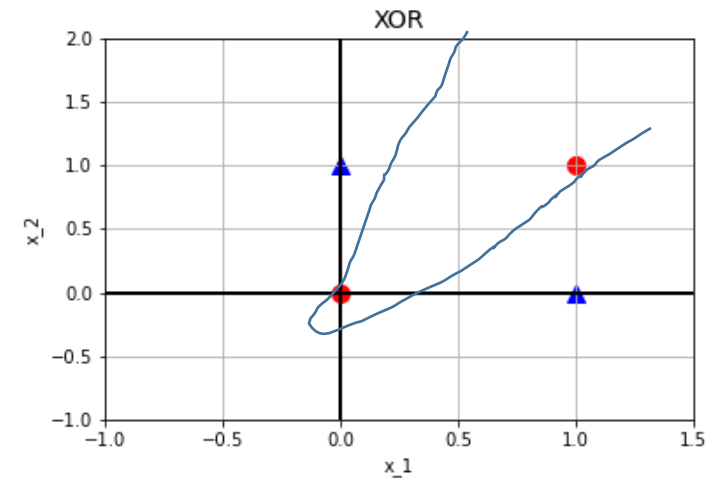
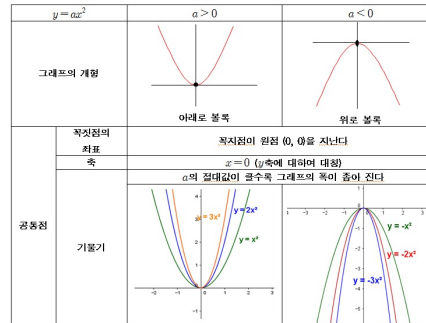
# Perceptron기반 논리회로 구현 (XOR)

XOR게이트의 그래프를 확인해보면 한 개의 직선으로 나눌 수 없음!

→ 어떻게 곡선으로 변환할 수 있을까? 2차 함수? 3차, 4차, 5차 함수도 될 텐데 어떤 것이 적절할까? (단 차원이 올라갈 수록 overfit될 수 있음)



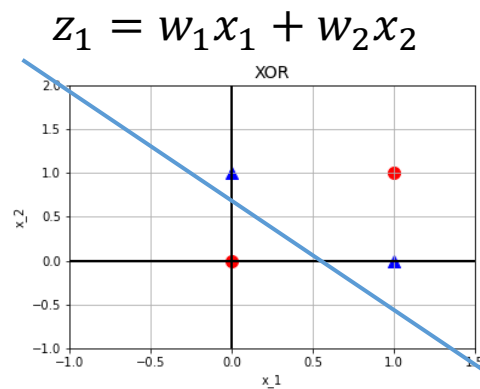
$$z = w_1x_1^2 + w_2x_2^2 + b$$



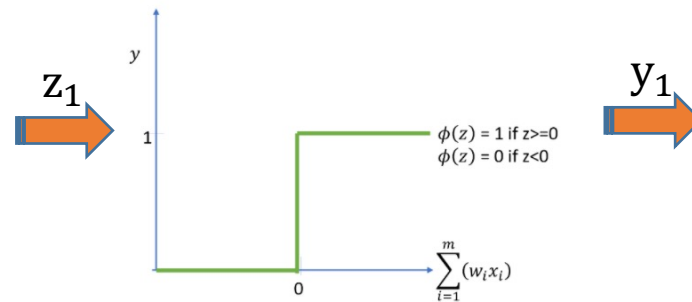
# Perceptron기반 논리회로 구현 (XOR)

XOR게이트의 그래프를 확인해보면 한 개의 직선으로 나눌 수 없음!

→ F(x)의 차원을 미리 정하지 말고 Perceptron을 중첩해보자



Perceptron 계단 함수



$$z_2 = w_2 y_1$$

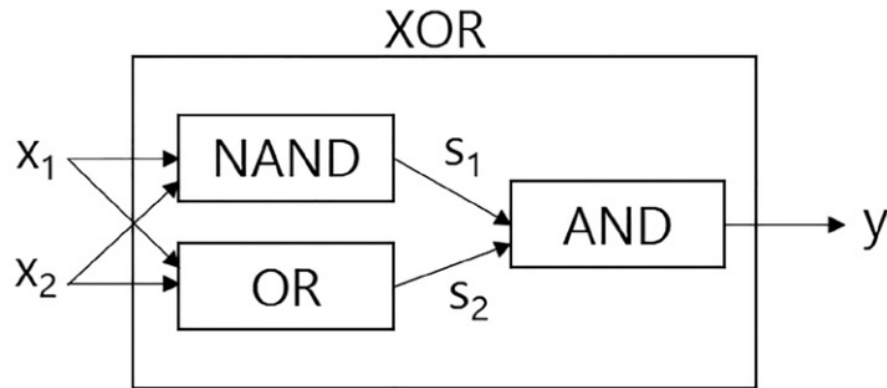
**z<sub>1</sub>**

**z<sub>2</sub>** → 리니어?  
 $z_2 = W_2(\text{act}(WX + b_1)) + b_2$

$$z_2 = W_2(\text{act}(WX + b_1)) + b_2$$

## Perceptron기반 논리회로 구현 (XOR)

- XOR게이트는 다음과 같이AND, NAND, OR게이트의 조합(Perceptron의 조합)으로 구성할 수 있다.



```
1 def XOR(x_1, x_2):  
2     s1 = NAND(x_1, x_2)  
3     s2 = OR(x_1, x_2)  
4     y = AND(s1, s2)  
5     return y
```

```
1 XOR(0, 0) # 0출력  
2 XOR(0, 1) # 1출력  
3 XOR(1, 0) # 1출력  
4 XOR(1, 1) # 0출력
```

$X_1$	$X_2$	NAND	OR	AND
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

### XOR 예제로 알 수 있는점

- 곡선으로 나누어야 하는 XOR게이트를 직선만 표현하는 퍼셉트론으로 나누기 위해 층(layer)을 쌓았다.
- 퍼셉트론의 층을 쌓았더니 **비선형 관계가 표현**이 되었다.
- 복잡한 관계가 **표현**이 되었다.
- 퍼셉트론 뭉치 = 신경망

# CONTENTS

---

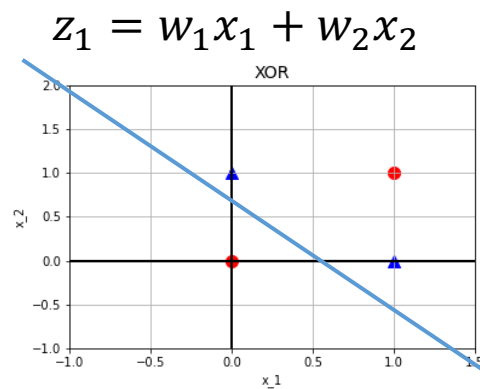
- ① AND, NAND, OR GATE
- ② 퍼셉트론과 논리회로 구현
- ③ 지도학습과 논리회로 구현
- ④ 신경망



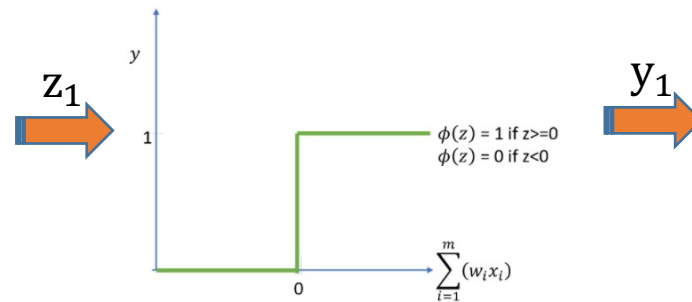
# 지도학습기반 논리회로 구현

퍼셉트론에서의 가중치( $w_1, w_2, \theta$ )를 직접 지정하지 않고 지도학습 알고리즘을 이용해 학습해서 최적값 (최적  $W$  값)을 구해보자.

→ 어라? act 함수가 미분 불가능함.



Perceptron 계단 함수



$$z_2 = w_2 y_1$$

**$z_1$**

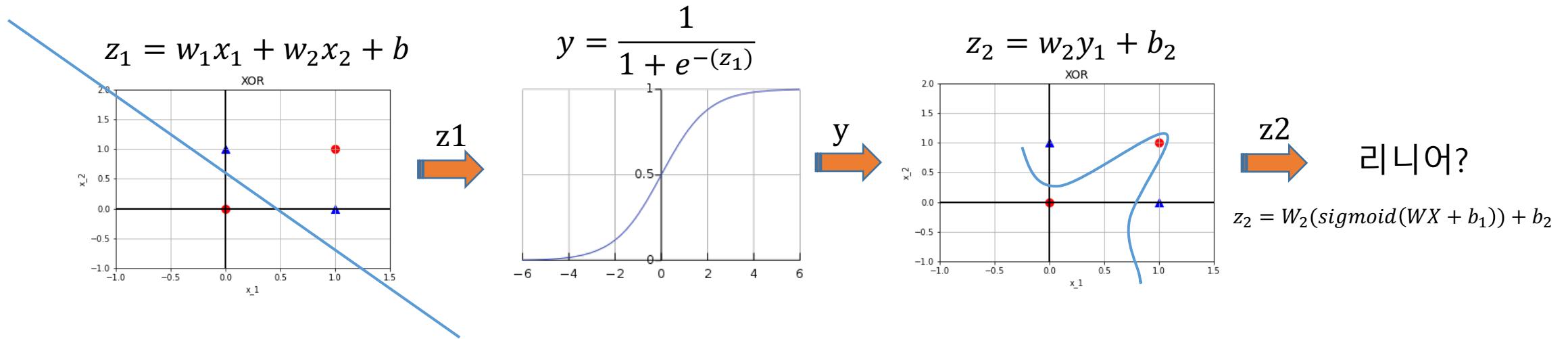
**$z_2$**  리니어?  
 $z_2 = W_2(act(WX + b_1)) + b_2$

$$z_2 = W_2(act(WX + b_1)) + b_2$$

# 지도학습기반 논리회로 구현

Perceptron의 act함수를 sigmoid로 바꾸면? == Logistic Regression

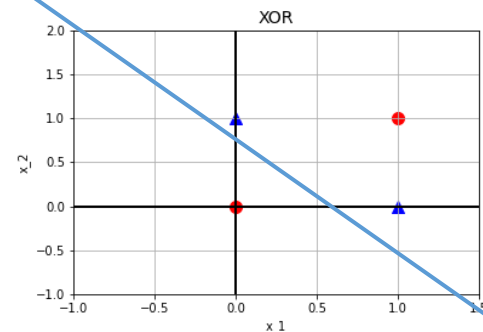
→ XOR의 출력값이 0과 1이기 때문에 Logistic regression 사용 가능



$$z_2 = W_2(\text{sigmoid}(WX + b_1)) + b_2$$

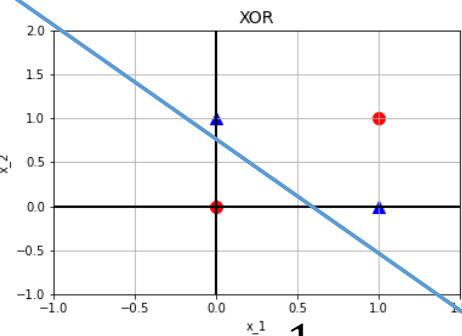
# 지도학습기반 논리회로 구현

$$z_1 = w_1x_1 + w_2x_2 + b$$



$z_1$

$$z_2 = w_1z_1 + b_2$$

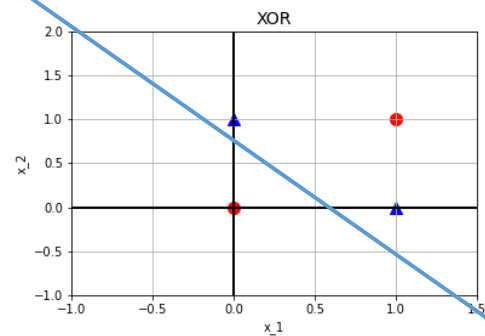


$z_2$

리니어?

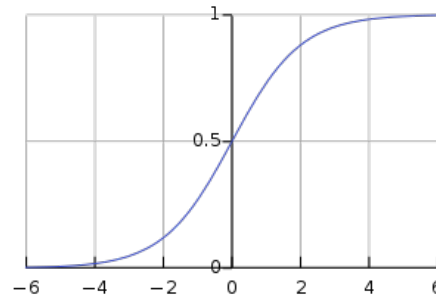
$$z_2 = W_2(WX + b_1) + b_2$$

$$z_1 = w_1x_1 + w_2x_2 + b$$



$z_1$

$$y = \frac{1}{1 + e^{-(z_1)}}$$

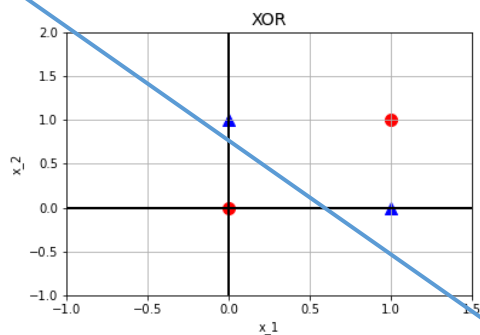


$y$

리니어?

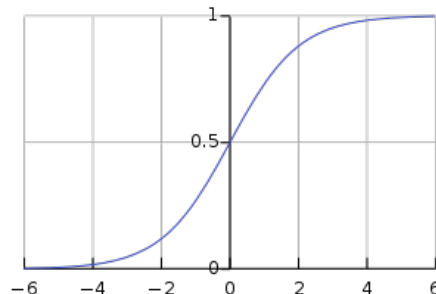
$$y_1 = \text{sigmoid}(WX + b_1)$$

$$z_1 = w_1x_1 + w_2x_2 + b$$



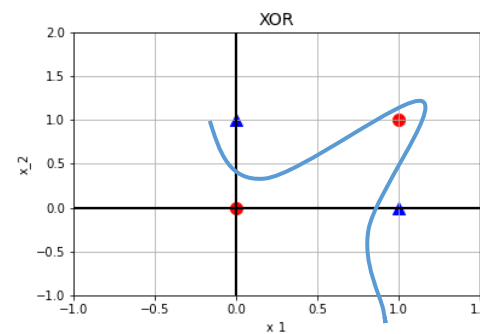
$z_1$

$$y = \frac{1}{1 + e^{-(z_1)}}$$



$y$

$$z_2 = w_1y_1 + w_2y_2 + b$$



$z_2$


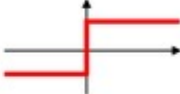


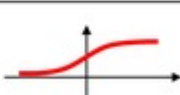

리니어?

$$z_2 = W_2(\text{sigmoid}(WX + b_1)) + b_2$$

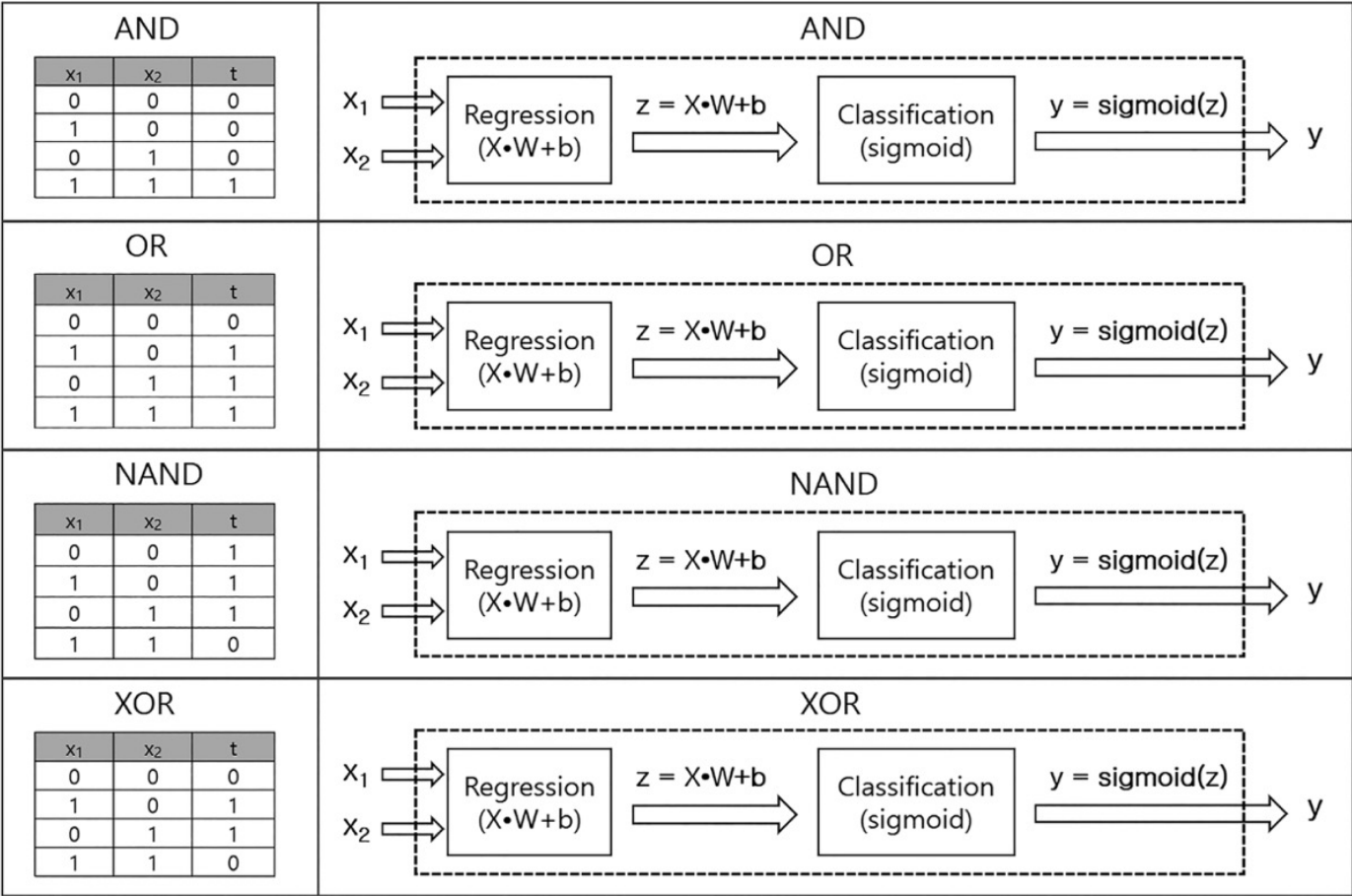
## 지도학습기반 논리회로 구현

그렇다면 act함수를 대체할 수 있는 함수는 어떤 것들이 있을까?

→ 곡선이 필요 → Nonlinearity 추가 필요 == Activation function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

📝 Logistic Regression을 이용한 논리회로 구현



# 지도학습기반 논리회로 구현



```
1 import numpy as np
2
3 # sigmoid 함수
4 def sigmoid(x):
5     return 1 / (1+np.exp(-x))
6
7 # 수치미분 함수
8 def numerical_derivative(f, x):
9     delta_x = 1e-4 # 0.0001
10    grad = np.zeros_like(x)
11
12    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
13
14    while not it.finished:
15        idx = it.multi_index
16        tmp_val = x[idx]
17        x[idx] = float(tmp_val) + delta_x
18        fx1 = f(x) # f(x+delta_x)
19
20        x[idx] = tmp_val - delta_x
21        fx2 = f(x) # f(x-delta_x)
22        grad[idx] = (fx1 - fx2) / (2*delta_x)
23
24        x[idx] = tmp_val
25        it.iternext()
26
27    return grad
```

## 지도학습기반 논리회로 구현

---

```
1 class LogicGate:
2
3     def __init__(self, gate_name, xdata, tdata): # xdata, tdata =>
4         numpy.array(...)
5
6         self.name = gate_name
7
8         # 입력 데이터, 정답 데이터 초기화
9         self.__xdata = xdata.reshape(4,2)
10        self.__tdata = tdata.reshape(4,1)
11
12        # 가중치 W, 바이어스 b 초기화
13        self.__W = np.random.rand(2,1) # weight, 2 X 1 matrix
14        self.__b = np.random.rand(1)
15
16        # 학습률 learning rate 초기화
17        self.__learning_rate = 1e-2
```

## 지도학습기반 논리회로 구현

```
1  # 손실함수
2  def __loss_func(self):
3
4      delta = 1e-7    # log 무한대 발산 방지
5
6      z = np.dot(self.__xdata, self.__W) + self.__b
7      y = sigmoid(z)
8
9      # cross-entropy
10     return -np.sum( self.__tdata*np.log(y + delta) + (1-
self.__tdata)*np.log((1 - y)+delta ) )
11
12     # 손실 값 계산
13     def error_val(self):
14
15         delta = 1e-7    # log 무한대 발산 방지
16
17         z = np.dot(self.__xdata, self.__W) + self.__b
18         y = sigmoid(z)
19
20         # cross-entropy
21         return -np.sum( self.__tdata*np.log(y + delta) + (1-
self.__tdata)*np.log((1 - y)+delta ) )
```

$$E(W, b) = -\sum_{i=1}^n \{t_i \log y_i + (1 - t_i) \log(1 - y_i)\}$$



## 지도학습기반 논리회로 구현

---

```
1  # 수치미분을 이용하여 손실함수가 최소가 될때 까지 학습하는 함수
2  def train(self):
3
4      f = lambda x : self.__loss_func()
5
6      print("Initial error value = ", self.error_val())
7
8      for step in range(8001):
9
10         self.__W -= self.__learning_rate * numerical_derivative(f, self.__W)
11
12         self.__b -= self.__learning_rate * numerical_derivative(f, self.__b)
13
14         if (step % 400 == 0):
15             print("step = ", step, "error value = ", self.error_val())
```

## 지도학습기반 논리회로 구현

---

```
1  # 미래 값 예측 함수
2  def predict(self, input_data):
3
4      z = np.dot(input_data, self.__W) + self.__b
5      y = sigmoid(z)
6
7      if y > 0.5:
8          result = 1  # True
9      else:
10         result = 0  # False
11
12     return y, result
```

# 지도학습기반 논리회로 구현

---

```
1 xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
2 tdata = np.array([0, 0, 0, 1])
3
4 AND_obj = LogicGate("AND_GATE", xdata, tdata)
5
6 AND_obj.train()
```

```
Initial error value = 4.4280539519443884
step = 0 error value = 4.3726227489326055
step = 400 error value = 1.5178494977079589
step = 800 error value = 1.1328073530602052
step = 1200 error value = 0.9126081130530589
step = 1600 error value = 0.7666648386148311
step = 2000 error value = 0.6615215795611431
step = 2400 error value = 0.581682022924658
step = 2800 error value = 0.5188158434291524
step = 3200 error value = 0.4679701275701782
step = 3600 error value = 0.4259825470677998
step = 4000 error value = 0.3907249768169546
step = 4400 error value = 0.3607063082440158
step = 4800 error value = 0.3348478165596296
step = 5200 error value = 0.31234850733000263
step = 5600 error value = 0.2926005494699164
step = 6000 error value = 0.2751341025075421
step = 6400 error value = 0.25958016184810406
step = 6800 error value = 0.2456448577435531
step = 7200 error value = 0.23309126313493536
step = 7600 error value = 0.22172625626544412
step = 8000 error value = 0.21139086521437228
```

## 지도학습기반 논리회로 구현

---

```
1 # AND Gate prediction
2 print(AND_obj.name, "\n")
3
4 test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
5
6 for input_data in test_data:
7     (sigmoid_val, logical_val) = AND_obj.predict(input_data)
8     print(input_data, " = ", logical_val, "\n")
```

AND\_GATE

[0 0] = 0


[0 1] = 0

[1 0] = 0

[1 1] = 1

# 지도학습기반 논리회로 구현

---



```
1 xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
2 tdata = np.array([0, 1, 1, 1])
3
4 OR_obj = LogicGate("OR_GATE", xdata, tdata)
5
6 OR_obj.train()
```

```
Initial error value = 1.9203099839204052
step = 0 error value = 1.9174840519283807
step = 400 error value = 1.1985036810532927
step = 800 error value = 0.851429490388919
step = 1200 error value = 0.6535332322124496
step = 1600 error value = 0.526959024376427
step = 2000 error value = 0.4395810609994197
step = 2400 error value = 0.3759416640080046
step = 2800 error value = 0.3277050577113909
step = 3200 error value = 0.28999176279497657
step = 3600 error value = 0.2597639868252872
step = 4000 error value = 0.23503699289965768
step = 4400 error value = 0.21446241567341195
step = 4800 error value = 0.19709407919824024
step = 5200 error value = 0.18224962949842524
step = 5600 error value = 0.16942530800107397
step = 6000 error value = 0.15824157365604338
step = 6400 error value = 0.14840733926592578
step = 6800 error value = 0.13969582026374797
step = 7200 error value = 0.13192784157214707
step = 7600 error value = 0.1249600587184293
step = 8000 error value = 0.11867649164239484
```

## 지도학습기반 논리회로 구현


---

```
1 # OR Gate prediction
2 print(OR_obj.name, "\n")
3
4 test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
5
6 for input_data in test_data:
7     (sigmoid_val, logical_val) = OR_obj.predict(input_data)
8     print(input_data, " = ", logical_val, "\n")
```

```
OR_GATE
[0 0] = 0
[0 1] = 1
[1 0] = 1
[1 1] = 1
```

# 지도학습기반 논리회로 구현

---



```
1 xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
2 tdata = np.array([1, 1, 1, 0])
3
4 NAND_obj = LogicGate("NAND_GATE", xdata, tdata)
5
6 NAND_obj.train()
```

```
Initial error value = 2.8216789234145
step = 0 error value = 2.8150609172326333
step = 400 error value = 1.6336201391128986
step = 800 error value = 1.191384159796652
step = 1200 error value = 0.9489022474034297
step = 1600 error value = 0.7917769657666989
step = 2000 error value = 0.6800882968216138
step = 2400 error value = 0.5960261551600591
step = 2800 error value = 0.5302517620516713
step = 3200 error value = 0.4773072985629022
step = 3600 error value = 0.4337510550105433
step = 4000 error value = 0.3972884006303736
step = 4400 error value = 0.3663232005208723
step = 4800 error value = 0.3397074647742464
step = 5200 error value = 0.31659285027117806
step = 5600 error value = 0.29633818912519777
step = 6000 error value = 0.27844958056608415
step = 6400 error value = 0.2625402838197325
step = 6800 error value = 0.24830311339577513
step = 7200 error value = 0.23549098370429117
step = 7600 error value = 0.22390291238361099
step = 8000 error value = 0.21337376752079162
```

## 지도학습기반 논리회로 구현

---

```
1 # NAND Gate prediction
2 print(NAND_obj.name, "\n")
3
4 test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
5
6 for input_data in test_data:
7     (sigmoid_val, logical_val) = NAND_obj.predict(input_data)
8     print(input_data, " = ", logical_val, "\n")
```

NAND\_GATE

[0 0] = 1

[0 1] = 1

[1 0] = 1

[1 1] = 0



# 지도학습기반 논리회로 구현

## XOR문제의 Logistic Regression을 이용한 Gate 구현

```
1 xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
2 tdata = np.array([0, 1, 1, 0])
3
4
5 XOR_obj = LogicGate("XOR_GATE", xdata, tdata)
6
7 # XOR Gate 를 보면, 손실함수 값이 2.7 근처에서 더 이상 감소하지 않는것
  을 볼수 있음
8 XOR_obj.train()
```

```
Initial error value = 3.6007423939178147
step = 0 error value = 3.58235051553463
step = 400 error value = 2.7899607773286785
step = 800 error value = 2.7775357154853664
step = 1200 error value = 2.7739919039628016
step = 1600 error value = 2.7729859234509022
step = 2000 error value = 2.7727007240321333
step = 2400 error value = 2.7726198917580827
step = 2800 error value = 2.7725969828988246
step = 3200 error value = 2.7725904901969907
step = 3600 error value = 2.772588650050717
step = 4000 error value = 2.7725881285168663
step = 4400 error value = 2.772587980703238
step = 4800 error value = 2.772587938809665
step = 5200 error value = 2.7725879269361093
step = 5600 error value = 2.7725879235708812
step = 6000 error value = 2.7725879226171015
step = 6400 error value = 2.77258792234678
step = 6800 error value = 2.7725879222701644
step = 7200 error value = 2.77258792224845
step = 7600 error value = 2.7725879222422956
step = 8000 error value = 2.7725879222405516
```

손실함수 값이 줄어들지 않는다!!

## 지도학습기반 논리회로 구현

---

XOR문제를 Logistic Regression으로 구현하고 예측해보니 오답 예측!!

```
1 # XOR Gate prediction => 예측이 되지 않음
2 print(XOR_obj.name, "\n")
3
4 test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
5
6 for input_data in test_data:
7     (sigmoid_val, logical_val) = XOR_obj.predict(input_data)
8     print(input_data, " = ", logical_val, "\n")
```

XOR\_GATE

[0 0] = 0

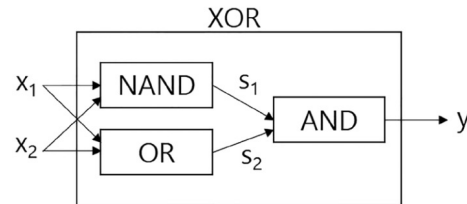
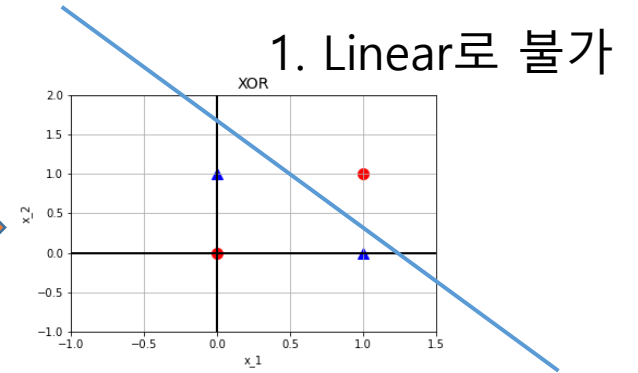
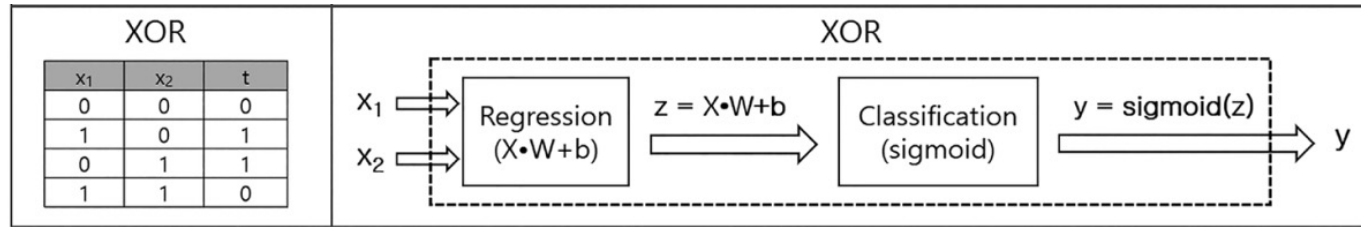
[0 1] = 0

[1 0] = 0

[1 1] = 1

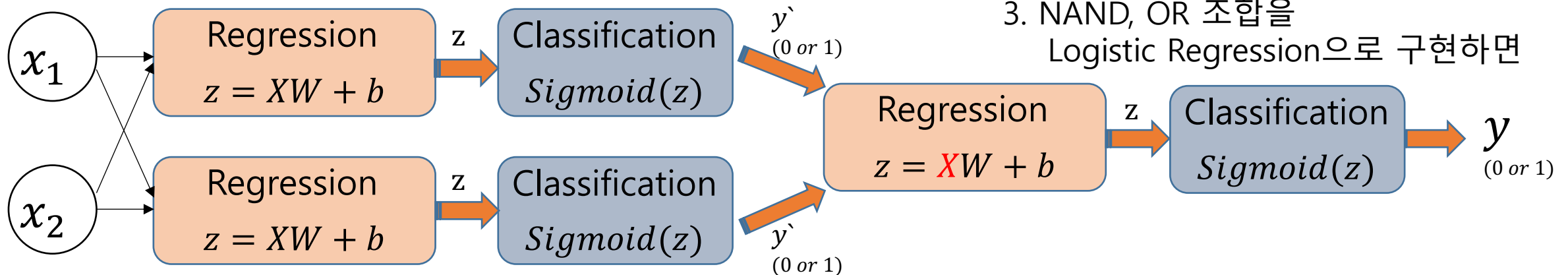
# 지도학습기반 논리회로 구현

- XOR게이트는 다음과 같이AND, NAND, OR게이트의 조합(Perceptron or Logistic Regression의 조합)으로 구성할 수 있다.



```
1 def XOR(x_1, x_2):
2     s1 = NAND(x_1, x_2)
3     s2 = OR(x_1, x_2)
4     y = AND(s1, s2)
5     return y
```

2. NAND, OR 조합 가능 (Nonlinear)



# 지도학습기반 논리회로 구현

```
1 # XOR 을 NAND + OR => AND 조합으로 계산함
2 input_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
3
4 s1 = []      # NAND 출력
5 s2 = []      # OR 출력
6
7 new_input_data = [] # AND 입력
8 final_output = []   # AND 출력
9
10 for index in range(len(input_data)):
11
12     s1 = NAND_obj.predict(input_data[index]) # NAND 출력
13     s2 = OR_obj.predict(input_data[index])   # OR 출력
14
15     new_input_data.append(s1[-1])           # AND 입력
16     new_input_data.append(s2[-1])           # AND 입력
17
18     (sigmoid_val, logical_val) =
19     AND_obj.predict(np.array(new_input_data))
20
21     final_output.append(logical_val)         # AND 출력, 즉 XOR 출력
22     new_input_data = []                     # AND 입력 초기화
23
24 for index in range(len(input_data)):
25     print(input_data[index], " = ", final_output[index], end='')
26     print("\n")
```

[0 0] = 0

[0 1] = 1

[1 0] = 1

[1 1] = 0

# CONTENTS

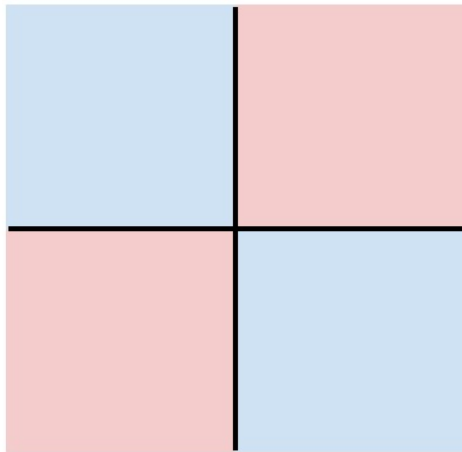
---

- ① AND, NAND, OR GATE
- ② 퍼셉트론과 게이트 구현
- ③ 지도학습과 게이트 구현
- ④ 신경망

## Hard cases for a linear classifier

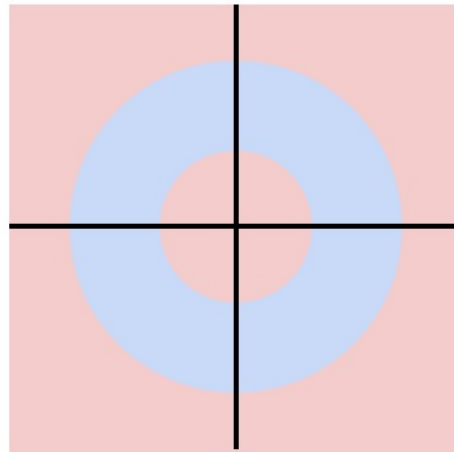
**Class 1:**  
number of pixels  $> 0$  odd

**Class 2:**  
number of pixels  $> 0$  even



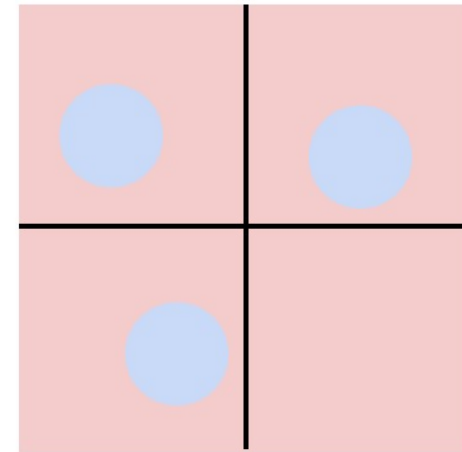
**Class 1:**  
 $1 \leq \text{L2 norm} \leq 2$

**Class 2:**  
Everything else



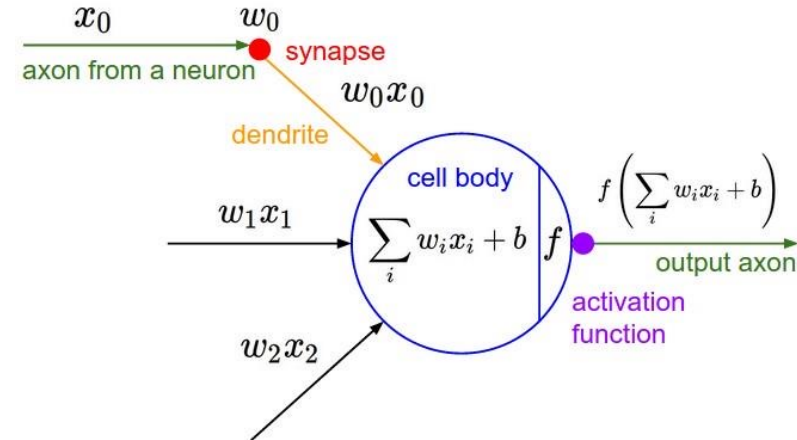
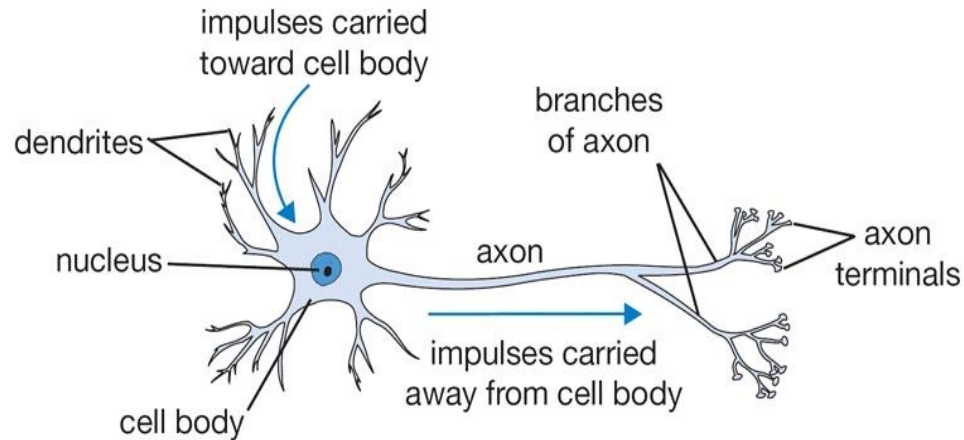
**Class 1:**  
Three modes

**Class 2:**  
Everything else



### ✎ XOR 예제로 알 수 있는점

- 퍼셉트론의 층을 쌓았더니 **비선형 관계가 표현**이 되었다.
- 복잡한 관계가 **표현**이 되었다.
- 퍼셉트론 뭉치 = 신경망



- 인간의 뇌: 여러 자극이 들어오고 일정 기준을 넘으면 이를 다른 뉴런에 전달하는 구조.
- 딥러닝: 어떤 인풋으로 들어오는 어떤 값들의 가중치와 bias의 합들이 어떤 값이 되면 활성화가 되고 아니면 활성화가 되지 않더라





감사합니다.