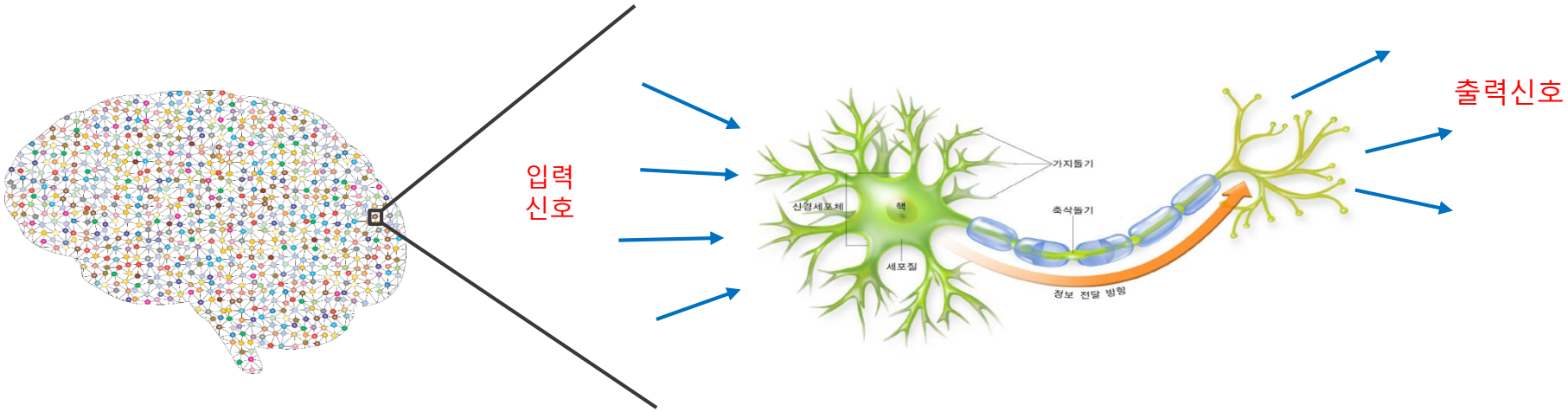
# 인공신경망

한국공학대학교
전자공학부
채승호 교수

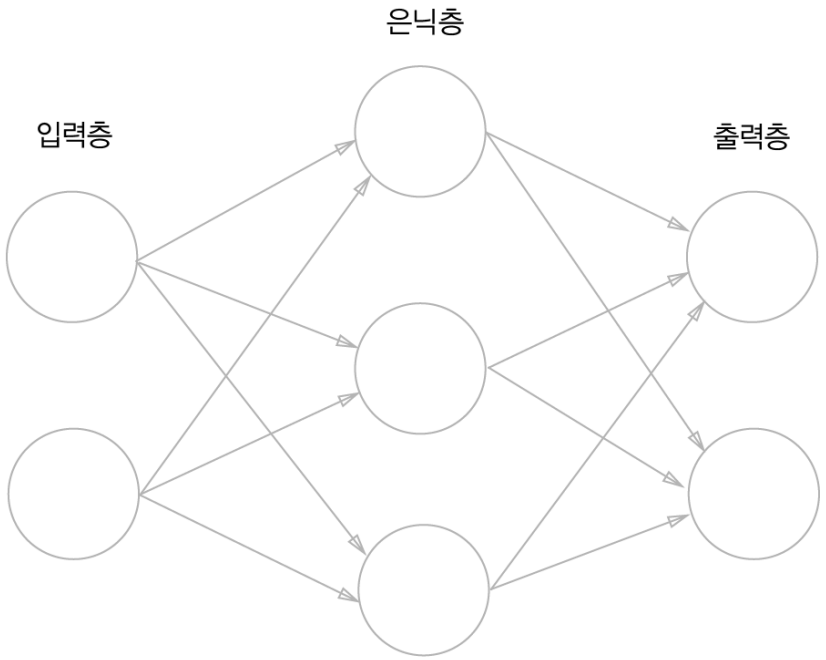
# 신경망(Neural Network)과 뉴런(Neuron)

- 뉴런(Neuron)
  - 천연 신경망 구성 기본 단위 세포
    - 신호 전달 절차
      - 축색종단은 다음 뉴런의 수상돌기와 인접 → 시냅스(synapse)
      - 축생종단에서는 전달된 신호의 강도에 따라 도파민 분비
        - » 신호 강도가 임계값을 넘으면 도파민 분비 ➔ 활성화
        - » 신호 강도가 입계값을 넘지 않으면 도파민 분비 X ➔ 비활성화
      - 분비된 도파민은 다음 뉴런의 수상 돌기에 전달됨



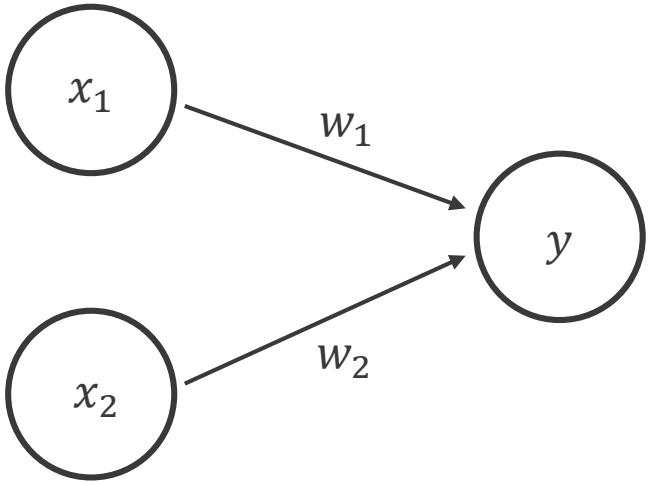
# 인공신경망 모델

- 인공신경망(Artificial Neural Network, ANN)
  - 생물 신경계통의 상호작용을 모방하여 구성한 네트워크 형태 인공지능 모델
  - 특징
    - 머신러닝 분야의 가장 성공적인 모델 중 하나
    - 인공 신경망
      - 유연한 구조
      - 다중 클래스 분류 가능
      - 속성공간의 다양한 분할 가능



# 퍼셉트론(Perceptron)

- 퍼셉트론
  - 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 제안한 초기 형태의 인공 신경망으로 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
  - 인공신경망을 구성하는 기본 단위
  - 실제 뇌를 구성하는 신경 세포 뉴런의 동작과  유사




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

$$= \begin{cases} 0, & w_0 + w_1 x_1 + w_2 x_2 \leq 0 \\ 1, & w_0 + w_1 x_1 + w_2 x_2 > 0 \end{cases}$$

$$-\theta = w_0$$

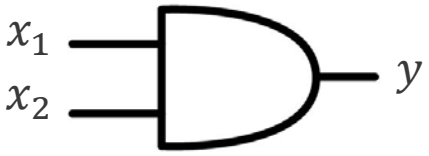
# 퍼셉트론

- Numpy를 이용한 퍼셉트론 class 구현

```python
import numpy as np

class perceptron:
    def __init__(self,w):
        self.w = w

    def output(self, x):
        tmp = np.dot(self.w, np.append(1,x))
        result = 1.0*(tmp>0)
        return result
```

# 퍼셉트론

- 퍼셉트론을 이용한 단순 논리회로 구현
  - AND 게이트

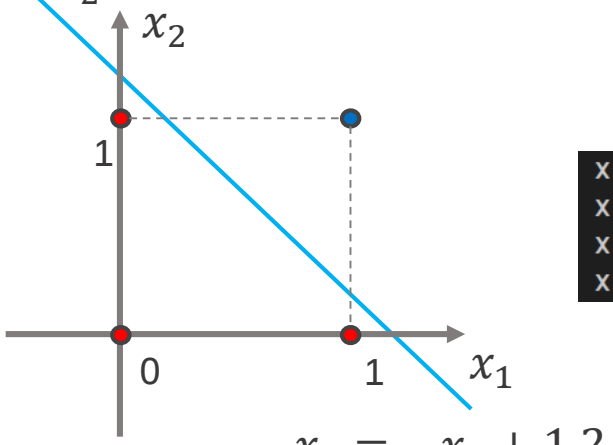



Ex) $w_0 = -1.2, w_1 = 1, w_2 = 1$

```
w = np.array([-1.2, 1, 1])
and_gate = perceptron(w)
x_list = [[0,0],[1,0],[0,1],[1,1]]
for x in x_list:
    print(f'x = {x} =====> {and_gate.output(x)}')
```

| Inputs | | Output |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$-1.2 + x_1 + x_2 = 0$

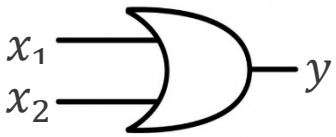



$x_2 = -x_1 + 1.2$

```
x = [0, 0] =====> 0.0
x = [1, 0] =====> 0.0
x = [0, 1] =====> 0.0
x = [1, 1] =====> 1.0
```

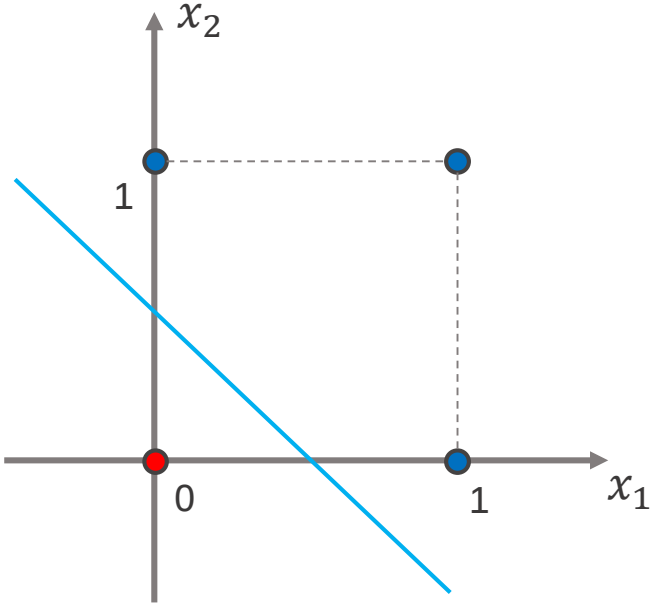
# 퍼셉트론

- 퍼셉트론을 이용한 단순 논리회로 구현
  - OR 게이트



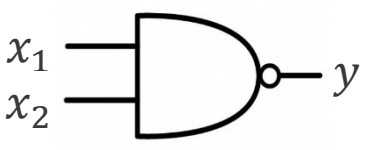

| Inputs | | Output |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$w_0 = ?, w_1 = ?, w_2 = ?$ 는 무엇이 되어야 할까?



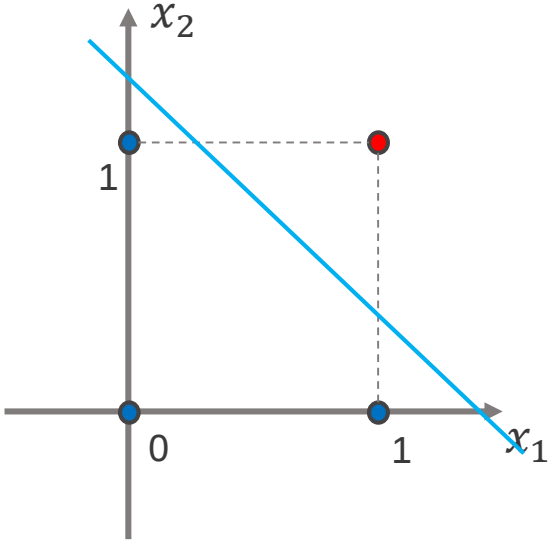
# 퍼셉트론

- 퍼셉트론을 이용한 단순 논리회로 구현
  - NAND 게이트



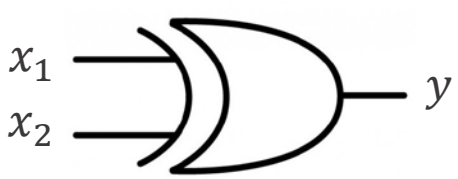

$w_0 = ?, w_1 = ?, w_2 = ?$ 는 무엇이 되어야 할까?

| Inputs | | Output |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



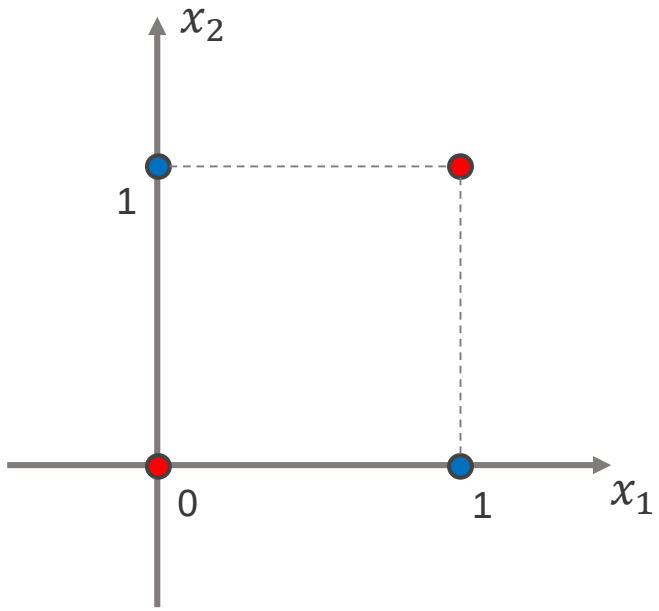
# 퍼셉트론의 한계

- XOR 게이트



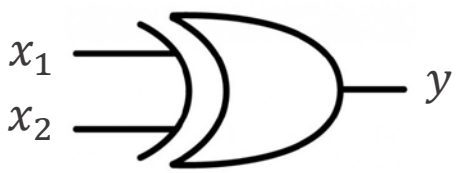

| Inputs | | Output |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

직선으로 분류 가능한가?

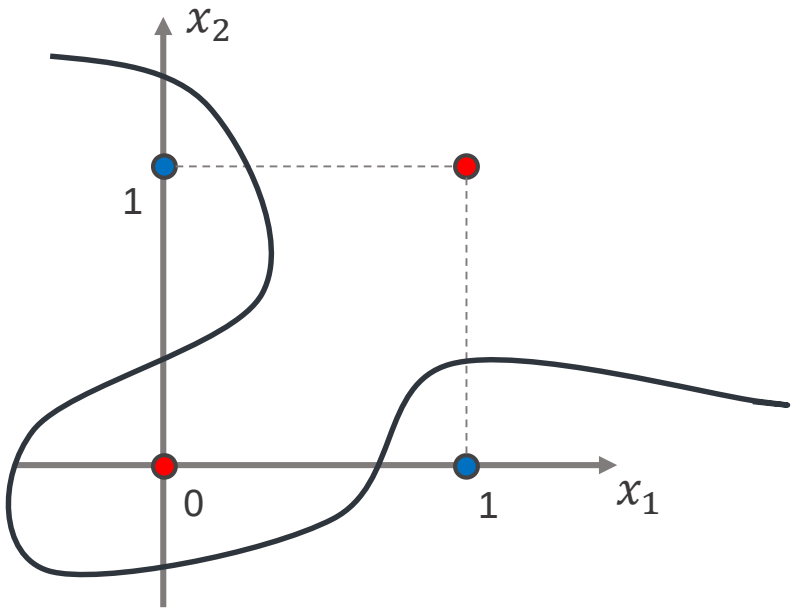


# 퍼셉트론의 한계

- XOR 게이트

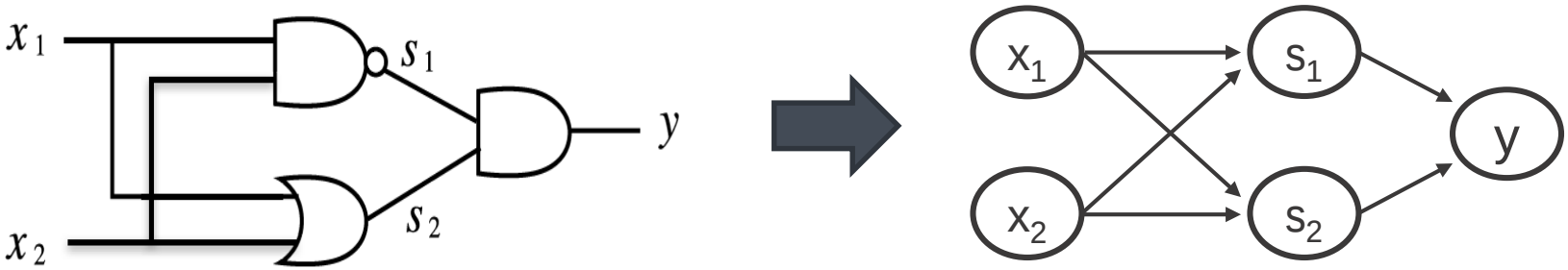



| Inputs | | Output |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# 퍼셉트론의 한계

- 단층 퍼셉트론
  - 선형 표현만 가능 → XOR와 같은 비선형 표현을 필요하는 문제 해결 불가
- 다층 퍼셉트론
  - 계층 추가를 통한 속성 공간 분할성능 향상
  - 2계층 구조만으로 거의 모든 형태의 함수 표현 가능

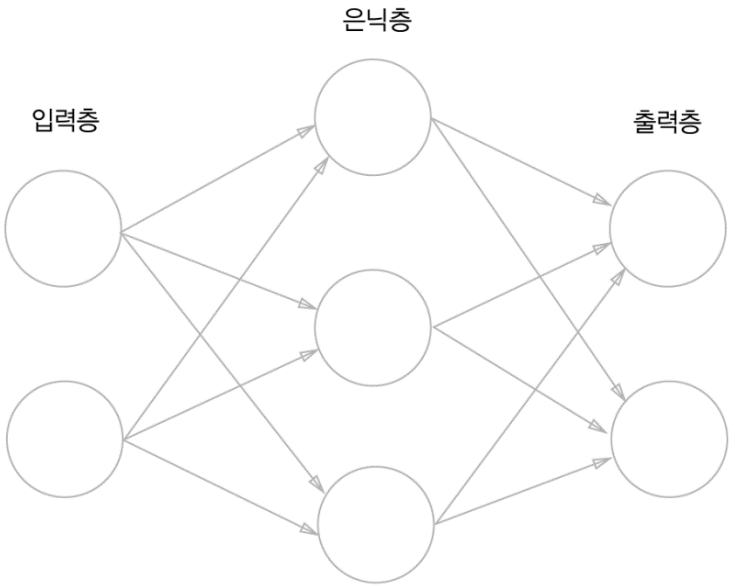


# 다계층 신경망(Multi-layer Neural Networks)

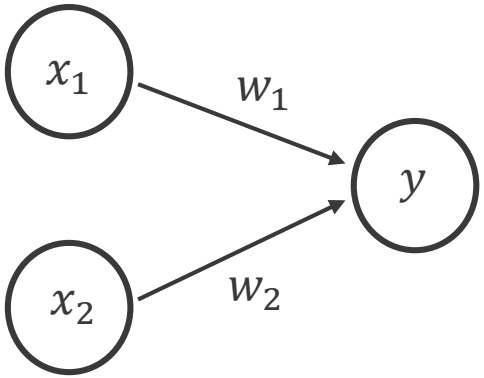
- 다계층신경망 구조
  - 입력층 (Input layer):
    - 신경망으로의 입력 값들이 위치

      $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i,\ z_j = h(a_j)$
  - 은닉층 (Hidden layer)
    - 입력층과 출력층 사이에 위치한 계층
    - $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j,\ y_k = \sigma(a_k)$
    - 은닉층의 수와 은닉층의 노드의 수는 설계자의 몫
      - 최적 은닉층 수와 은닉층 노드의 수는 열린 문제
      - 은닉층과 은닉층 노드의 수가 증가할수록 시스템은 유연해지지만, 연산 복잡도는 증가함
  - 출력층 (Output layer)
    - 신경망의 출력 값들이 위치

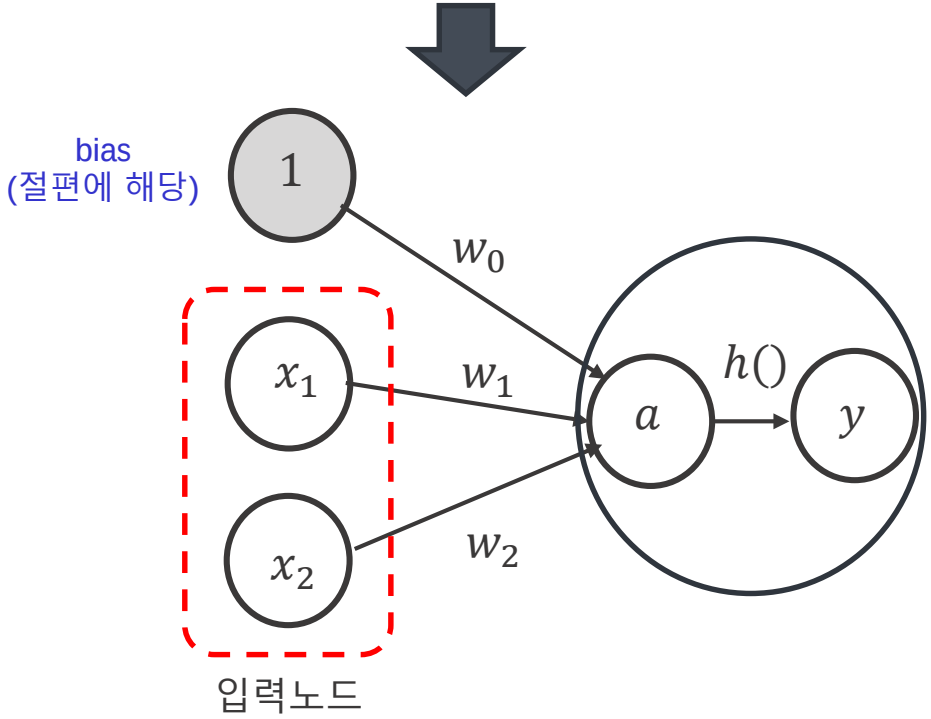
      $y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)}\, h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$



# 퍼셉트론의 다른 표현




$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 \leq \theta \\ 1, & w_1x_1 + w_2x_2 > \theta \end{cases}$$



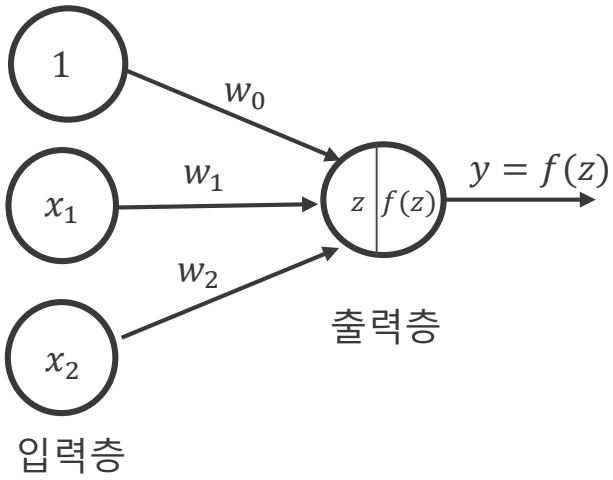

$$a = w_0 + w_1x_1 + w_2x_2$$

$$y = h(a)$$

활성화 함수(activation function)

Ex) $h(a) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$

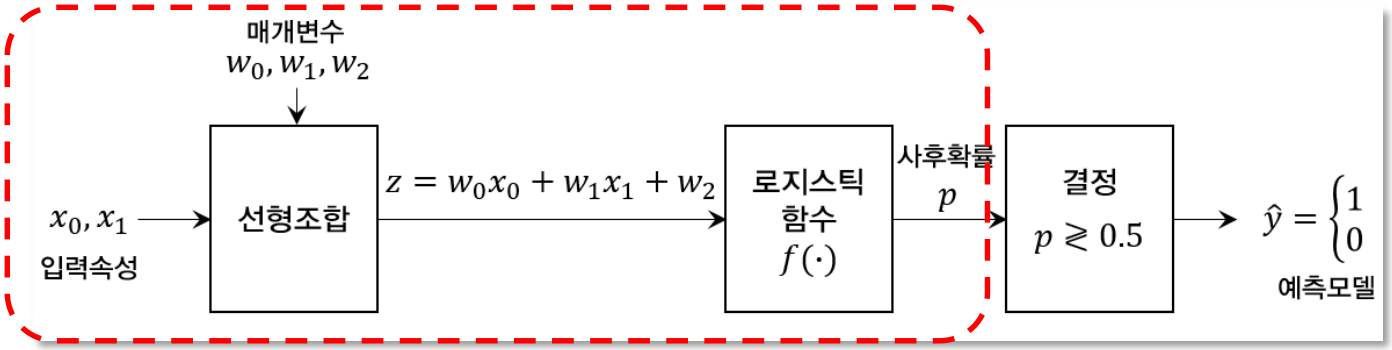
# 퍼셉트론의 다른 표현




$$z = w_0x_0 + w_1x_1 + w_2x_2 = \sum_{n=0}^{2} w_nx_n = \mathbf{w}^T\mathbf{x}$$

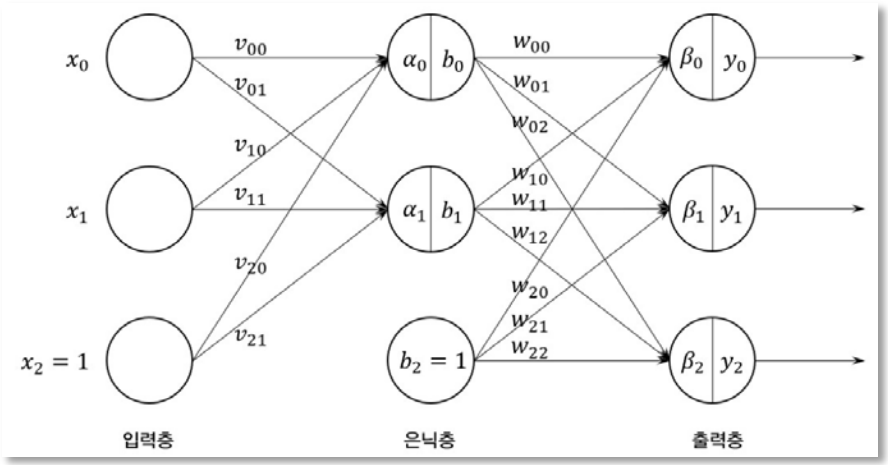
$$y = f(z) = f\left(\sum_{n=0}^{2} w_nx_n\right) = f(\mathbf{w}^T\mathbf{x})$$

로지스틱 함수를 활성 함수로 사용하는 퍼셉트론 = 로지스틱 회귀



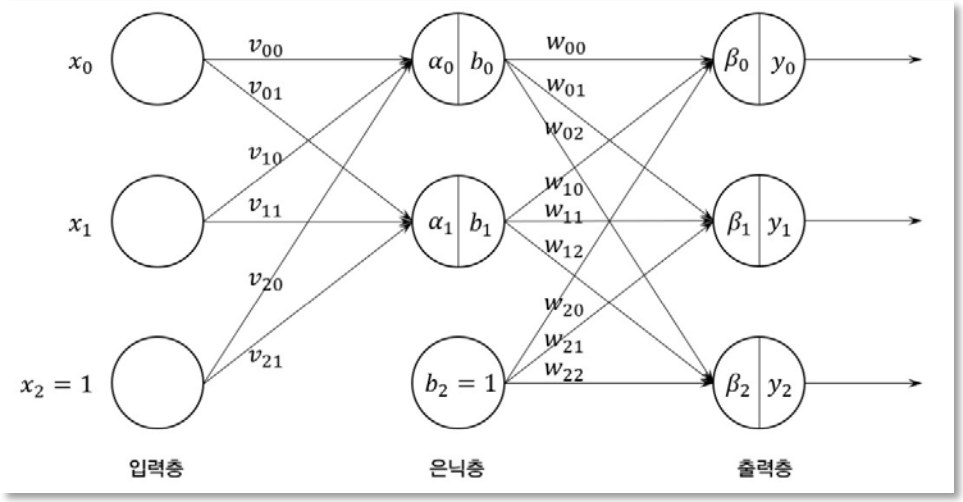
# 다계층 신경망

- 예시) 2계층 인공 신경망: 2입력 3클래스 분류 시스템




- 활성화 함수 (Activation function)
  - 입력노드와 더미노드를 제외한 모든 노드는 연산 및 활성화 기능 담당
  - 은닉층과 출력층이 각각 서로 다른 활성화 함수 가용 가능
- 출력층
  - 출력층 노드의 개수는 클래스의 개수와 동일
  - 각 노드의 출력 $y_0, y_1, y_2$는 해당 클래스에 속할 확률 또는 가능성을 나타냄
  - 확률을 나타내는 경우 $y_0 + y_1 + y_2 = 1$

# 다계층 신경망




은닉층 입출력 관계

$$\alpha_0 = v_{00}x_0 + v_{10}x_1 + v_{20}x_2$$
$$\alpha_1 = v_{01}x_0 + v_{11}x_1 + v_{21}x_2$$

$$b_0 = f(\alpha_0)$$
$$b_1 = f(\alpha_1)$$

출력층 입력

$$\beta_0 = w_{00}b_0 + w_{10}b_1 + w_{20}b_2$$
$$\beta_1 = w_{01}b_0 + w_{11}b_1 + w_{21}b_2$$
$$\beta_2 = w_{02}b_0 + w_{12}b_1 + w_{22}b_2$$

시스템 출력

$$\hat{y}_0 = f(\beta_0)$$
$$\hat{y}_1 = f(\beta_1)$$
$$\hat{y}_2 = f(\beta_2)$$

**시스템 학습 목표**

$\hat{y}_0 \approx y_0, \hat{y}_1 \approx y_1, \hat{y}_2 \approx y_2$를 만족하는 최적 매개변수를 찾음!

# 활성화 함수(Activation function)

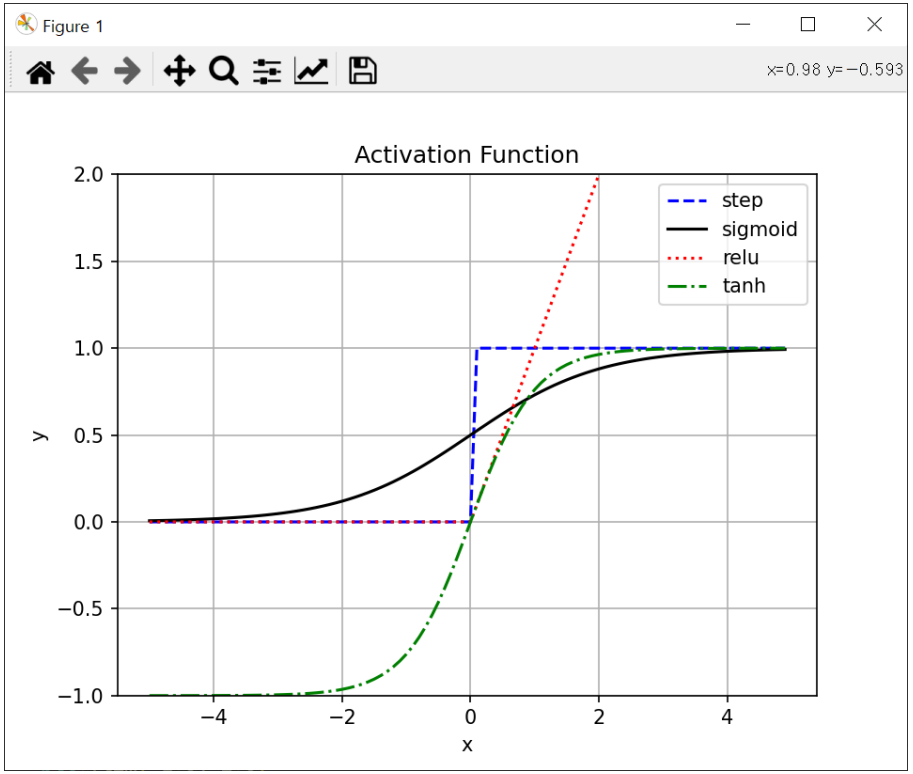
- 시그모이드 함수 : $h(x) = \frac{1}{1-e^{-x}}$
  - 이진 분류에서의 클래스 확률을 표현하는데 적합
  - 미분이 간단하여 수학적으로 사용하기 편리함 : $\frac{d}{dz}f(z) = f(z)(1 - f(z))$

- 계단 함수 : $h(a) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$

- tanh 함수 : $h(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

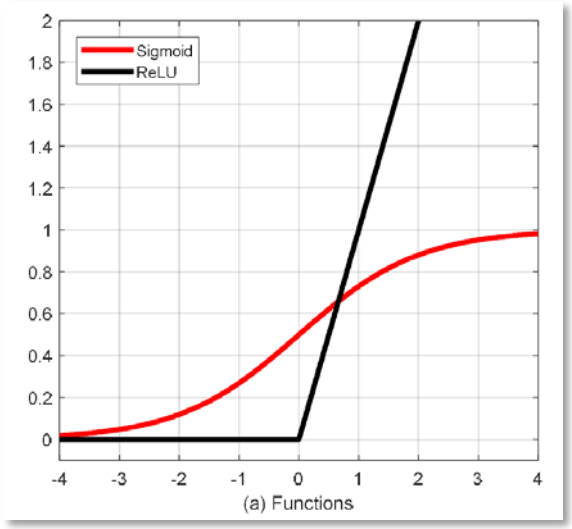
- Relu(Rectified Linear Unit) 함수: $h(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$
  - 입력이 0보다 작으면 출력은 0 → 비활성
  - 입력이 0보다 크면 입력값 그대로 출력됨 → 활성인 동시에 출력 크기로 활성 강도 표현

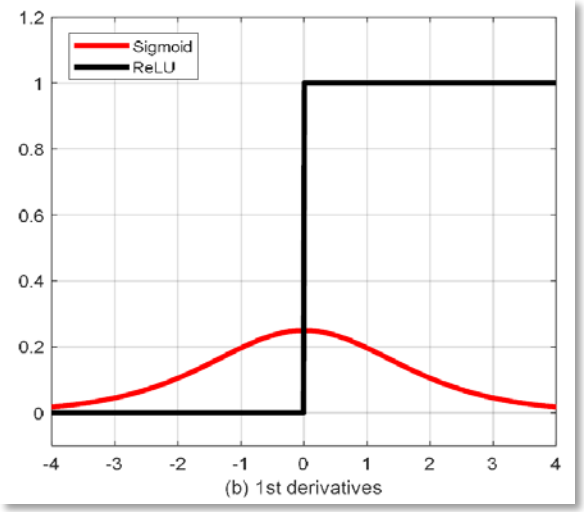
# 활성화 함수(Activation function)



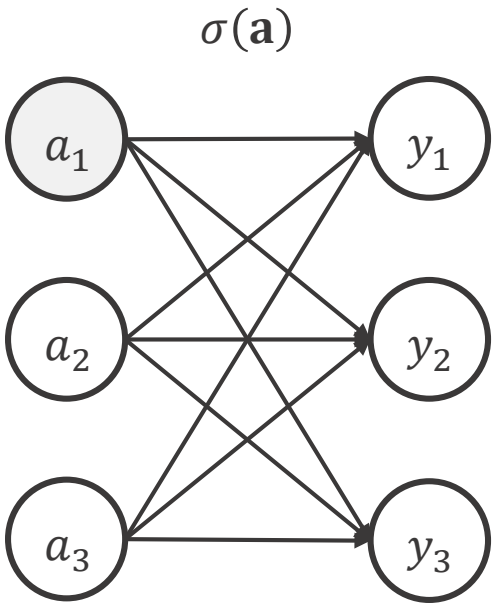
# 활성화 함수(Activation function)

- 시그모이드 계열 함수들의 단점
  - 입력값의 크기에 상관없이 출력 크기는 0과 1사이 값
  - 활성화 강도 표현의 어려움
    - 입력 신호 세기가 세더라도 출력 최대값은 항상 1
    - 인공 신경망에서 신호가 강한 뉴런에 대한 고려가 어려움
    - 어떤 뉴런에서 신호의 세기가 커지는 경우 출력이 항상 1이 됨
      - » 경사하강을 위한 경사가 0으로 수렴
      - » 그래디언트 소멸 문제
    - 인공 신경망의 규모가 커짐에 따라 시그모이드 함수의 문제가 누적
      - » 딥러닝의 한계로 작용
  - 한계를 극복하기 위한 새로운 활성함수 필요
    - 새로운 대안: ReLU 함수
    - 오늘날 인공 신경망의 표준 활성함수로 활용됨



# 출력층의 활성화 함수




- 출력층 설계
  - 문제에 따라 다른 활성화 함수를 활용
    - 회귀(Regression):
      - 입력 데이터에서 연속적인 수치를 예측하는 문제
        ex) 주가 예측, 날씨 예측
      - 일반적으로 항등(Identity)함수 사용
    - 분류(Classification):
      - 입력 데이터가 어떤 클래스(class)에 속하는지 분류하는 문제
        ex) 강아지-고양이 이미지 분류, 손글씨 인식 등
      - 분류하고 싶은 클래스 수로 설정하는 것이 일반적임
        - » ex) 고양이-강아지 사진 분류: 출력 노드 2개, 숫자 이미지 분류: 출력 노드 10개
      - 일반적으로 소프트맥스(softmax) 함수 사용

$$y_k = \sigma(\mathbf{a}) = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$$

      - 소프트맥스 함수의 결과는 각 클래스에 대한 확률로 해석될 수 있음

# 출력층의 활성화 함수

```python
import numpy as np

def softmax(x):
    y = np.exp(x)
    sum = np.sum(y)
    return y/sum

a = np.array([0.3, 2.9, 4.0])
print(softmax(a))
```

- 소프트맥스(Softmax) 함수
  - 시그모이드 함수의 확장된 함수
    - n개의 입력에 대한 소프트맥스 함수 정의

$$f(z_k) = \frac{e^{z_k}}{\sum_{i=0}^{n-1} e^{z_i}}, k = 0,1, \dots, n-1$$

    - 예를 들어 n=3인 경우,

$$f(z_0) = \frac{e^{z_0}}{e^{z_0}+e^{z_1}+e^{z_2}}, f(z_1) = \frac{e^{z_1}}{e^{z_0}+e^{z_1}+e^{z_2}}, f(z_2) = \frac{e^{z_2}}{e^{z_0}+e^{z_1}+e^{z_2}},$$

  - 모든 함수 값의 합이 항상 1이 됨
    - 다중 클래스 분류에서 각 클래스의 확률을 나타내는데 사용
    - 다중 클래스 분류 시스템의 출력층 활성함수로 많이 사용됨

# 출력층의 활성화 함수

- 소프트맥스 함수
  - 오버플로우(Overflow) 문제: 큰 숫자를 무한대로 취급

    Ex) $e^{1000}$은 inf으로 처리됨

```
import numpy as np

x = np.array([1010, 1000, 990])
y = np.exp(x)
sum = np.sum(y)
result = y/sum

print(f'y = {y}')
print(f'sum = {sum}')
print(f'result = {result}')
```

```
y = [inf inf inf]
sum = inf
result = [nan nan nan]
```

  - 해결방안

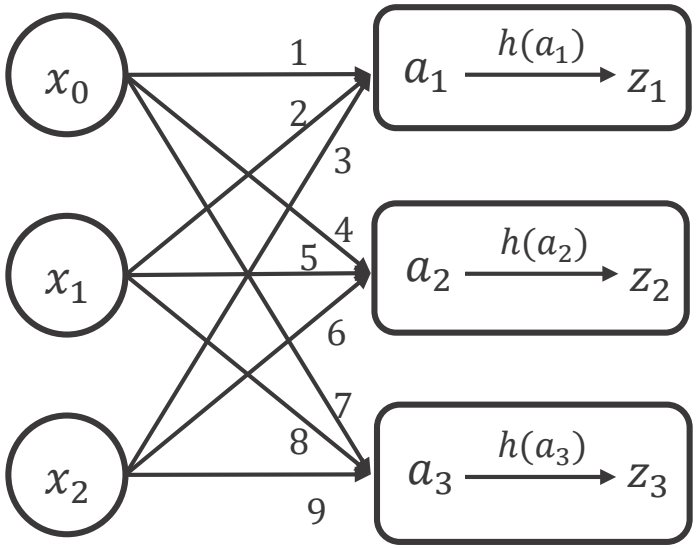
$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}} = \frac{Ce^{a_k}}{C\sum_{i=1}^{n} e^{a_i}}$$
$$= \frac{e^{a_k+\log C}}{\sum_{i=1}^{n} e^{a_i+\log C}}$$
$$= \frac{e^{a_k+C'}}{\sum_{i=1}^{n} e^{a_i+C'}}$$

```
import numpy as np

x = np.array([1010, 1000, 990])
c = np.max(x)
y = np.exp(x-c)
sum = np.sum(y)
result = y/sum

print(f'y = {y}')
print(f'sum = {sum}')
print(f'result = {result}')
```

```
y = [1.00000000e+00 4.53999298e-05 2.06115362e-09]
sum = 1.0000454019909162
result = [9.99954600e-01 4.53978686e-05 2.06106005e-09]
```

# 단층신경망(Single-Layer Neural Network)

- 배열연산을 활용한 효과적인 신경망 구현




```python
import numpy as np

def sigmoid(x):
    return 1/(1+np.exp(-x))

x = np.array([1,2,3])
W = np.array([[1,4,7],[2,5,8],[3,6,9]])
print(f'x = {x}')
print(f'W = {W}')

a = np.dot(x,W)
print(f'a = {a}')

z = sigmoid(a)
print(f'z = {z}')
```

- $a_j = \sum_{i=0}^{2} w_{ji} x_i$ for $j \in \{1, 2, 3\}$

  → $\mathbf{a} = \mathbf{xW}$

```
x = [1 2 3]
W = [[1 4 7]
 [2 5 8]
 [3 6 9]]
a = [14 32 50]
z = [0.99999917 1.         1.        ]
```

$$\mathbf{a} = [a_1,\ a_2,\ a_3], \mathbf{x} = [x_0 = 1, x_1, x_2],\ \mathbf{W} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

# 다계층신경망(Neural Network)

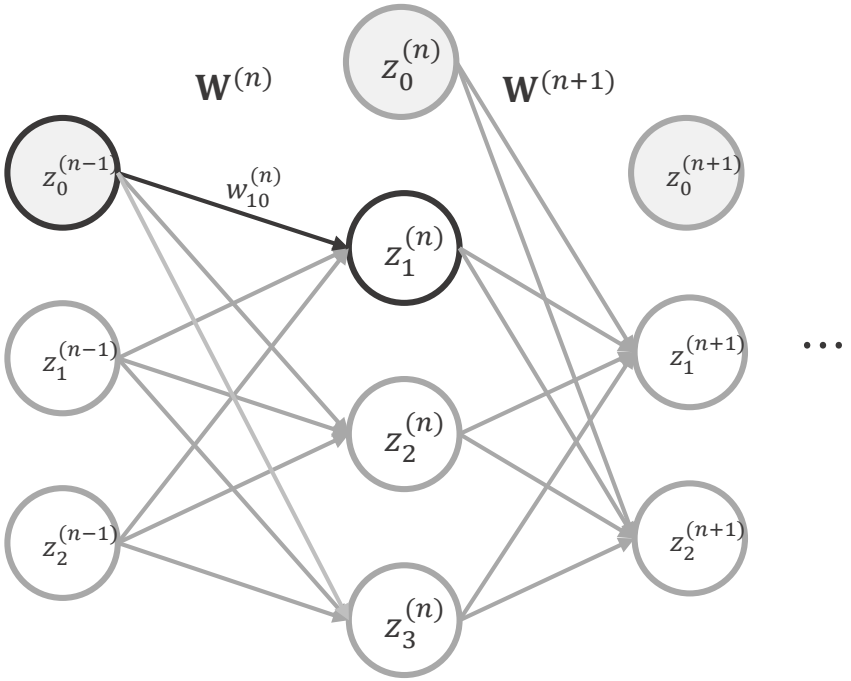
- $N$계층 신경망 구현하기

$$w_{ij}^{(n)}$$

계층 $n$ 가중치

앞 계층 노드 $j$

뒷 계층 노드 $i$




- 각 계층에서의 연산

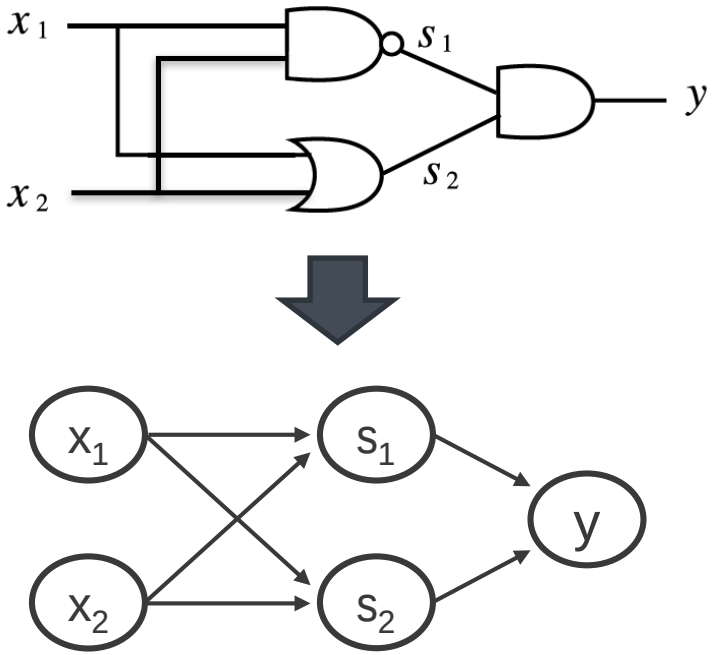
$$\mathbf{a}^{(n)} = \mathbf{z}^{(n-1)}\mathbf{W}^{(n)}, \quad \mathbf{z}^{(n)} = h\left(\mathbf{a}^{(n)}\right)$$

$$\mathbf{a}^{(n)} = \left[a_1^{(n)}, a_2^{(n)}, a_3^{(n)}\right], \ \mathbf{z}^{(n-1)} = \left[z_0^{(n-1)} = 1, z_1^{(n-1)}, z_2^{(n-1)}, z_3^{(n-1)}\right]$$

$$\mathbf{W}^{(n)} = \begin{bmatrix} w_{10}^{(n)} & w_{20}^{(n)} & w_{30}^{(n)} \\ w_{11}^{(n)} & w_{21}^{(n)} & w_{31}^{(n)} \\ w_{12}^{(n)} & w_{22}^{(n)} & w_{32}^{(n)} \end{bmatrix}$$

# 10주차 실습 #1

- 퍼셉트론을 활용하여 XOR 게이트를 구현하시오.
  - OR, NAND 게이트로 동작하기 위한 퍼셉트론의 가중치를 찾고 OR, NAND 게이트를 구현하시오.
  - AND, OR, NAND 퍼셉트론을 활용하여 XOR 게이트를 구현하시오.




OR

```
x = [0, 0] =====> 0.0
x = [1, 0] =====> 1.0
x = [0, 1] =====> 1.0
x = [1, 1] =====> 1.0
```

NAND

```
x = [0, 0] =====> 1.0
x = [1, 0] =====> 1.0
x = [0, 1] =====> 1.0
x = [1, 1] =====> 0.0
```

XOR

```
x = [0, 0] =====> 0.0
x = [1, 0] =====> 1.0
x = [0, 1] =====> 1.0
x = [1, 1] =====> 0.0
```

# 10주차 실습 #2

- 다음 4가지 경우에 대한 2계층 신경망을 설계하고 값을 찾으시오.
  - Weight Matrix

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix}, \quad \mathbf{W}^{(2)} = \begin{bmatrix} 0.1 & 0.2 \\ 0.1 & 0.4 \\ 0.2 & 0.5 \\ 0.3 & 0.6 \end{bmatrix}$$

    - Case 1) 은닉층: 시그모이드 함수 $h(x) = \frac{1}{1+e^{-x}}$ , 출력층 항등 함수 $\sigma(x) = x$
    - Case 2) 은닉층: 시그모이드 함수, 출력층 소프트맥스 함수
    - Case 3) 은닉층: Relu 함수, 출력층 항등 함수
    - Case 4) 은닉층: Relu 함수, 출력층 소프트맥스 함수
  - 다음 주어진 입력에 대해 위의 4가지 경우의 결과값을 확인하시오.
    - $\mathbf{x} = [1.0, 0.5] \Rightarrow \mathbf{y} = ?$

# 10주차 실습 #3

- “NN_data.csv” 데이터를 활용한 실습
  - 1) “NN_data.csv” 데이터를 3원 평면에 나타내시오. (다음 그림 참고)
  - 2) One-Hot Encoding 구현
    - 데이터에서의 y값(target) 으로부터 분류 할 Class가 몇 개인지 Check
    - 각 Class에 대해 One-Hot 표현으로 변환
    - “NN_data.csv”에 적용
  - 3) 2계층 신경망 구현
    - 데이터에서의 Input 속성 수 및 Output Class 수를 자동으로 Check하는 기능
    - Hidden layer의 Node 수를 자유롭게 설정하는 기능
    - Input 속성 수, Output Class 수, Hidden Node 수에 따라 Weight Matrix 생성 및 초기화
    - “NN_data.csv”에 적용: 랜덤 초기화 된 Weight Matrix로부터 $\hat{y}$ 값 도출

# 10주차 실습 #3

- Class1 : (x0,x1,x2)=(2,4,6)을 중심으로 노이즈가 추가된 데이터
- Class2 : (x0,x1,x2)=(4,6,2)을 중심으로 노이즈가 추가된 데이터
- Class3 : (x0,x1,x2)=(6,2,4)을 중심으로 노이즈가 추가된 데이터

- 각 Class별 300개씩 총 900개