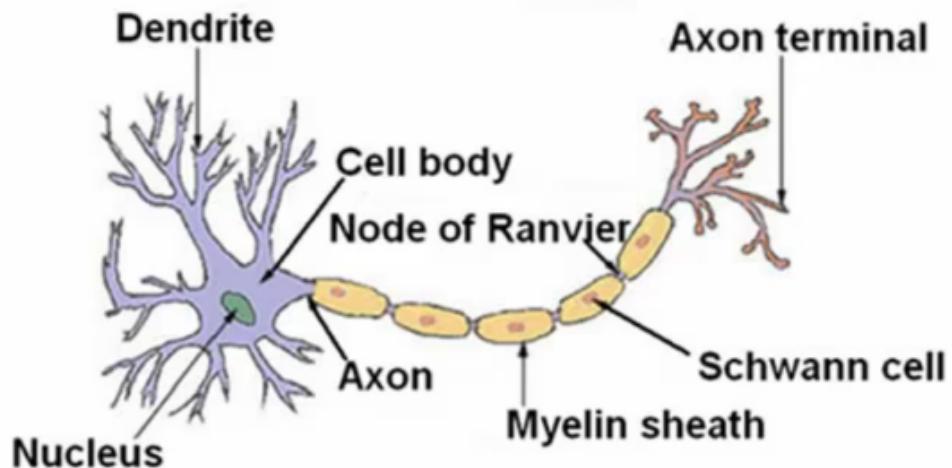
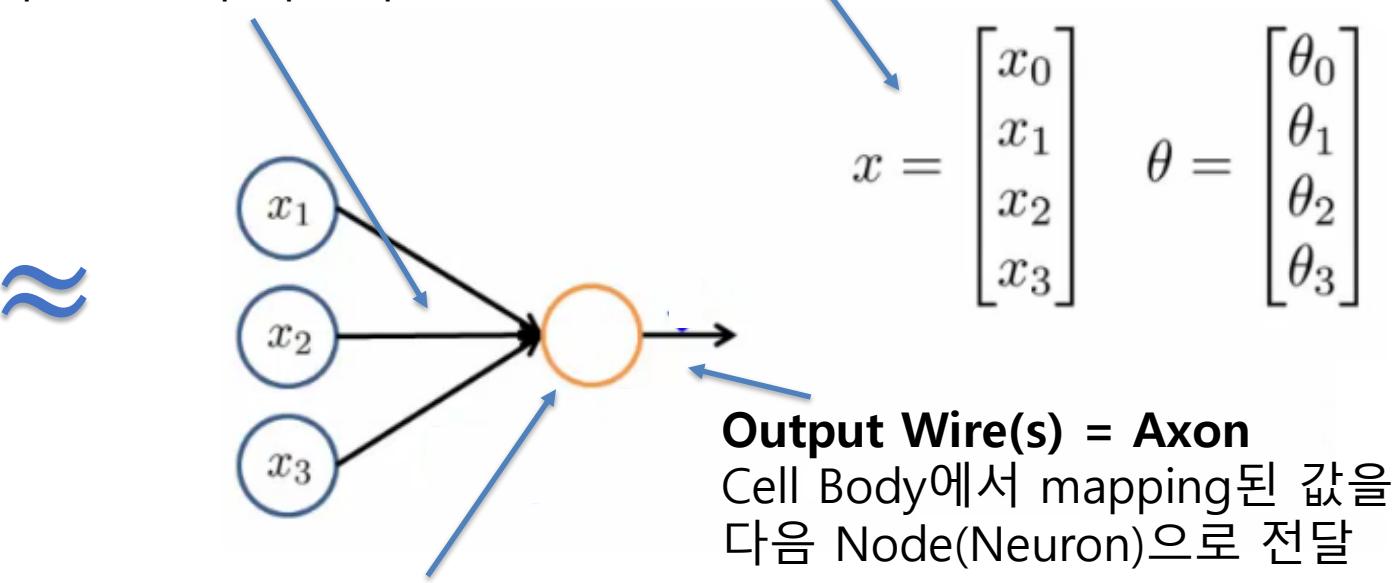


# Feedforward Neural Network

# Brain VS Artificial Neural Network



**Input Wire(s) = Dendrite**  
이전 Layer의 Node에서의 값을 전달해 Fan-in의 역할 수행



**Node = Cell Body**  
현 Layer의 Node는 Cell Body와 마찬가지로 간단한 연산을 수행  
Input을 Weight Matrix와 Activation Function을 통해 Compute

**Node Value (Input or Activation) = Stimulus**  
각 단계의 Node(Neuron)에서 compute된 value로 연쇄적인 network를 거쳐 최종적 정보로 전달

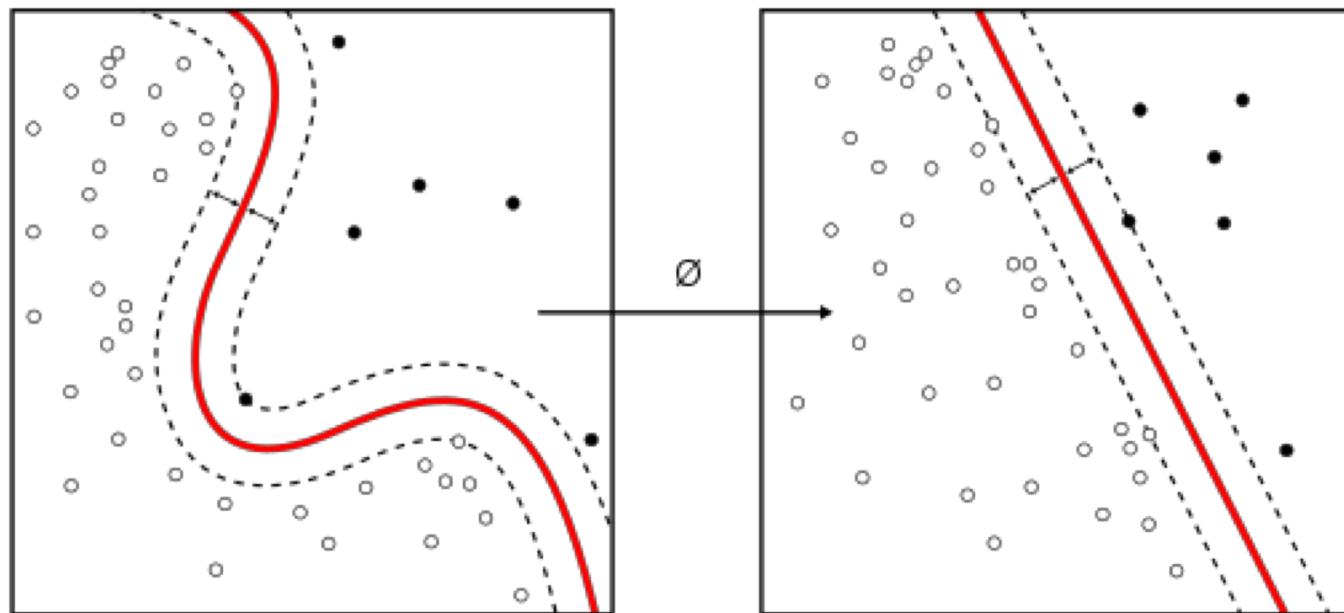
# Linear classification

- For learning linear classification
- Converge when data are linearly separable
- Activation function<sup>0</sup>| sigmoid<sup>0</sup>| 고 error term<sup>0</sup>| cross-entropy(logistics)<sup>0</sup>| Neural Network.

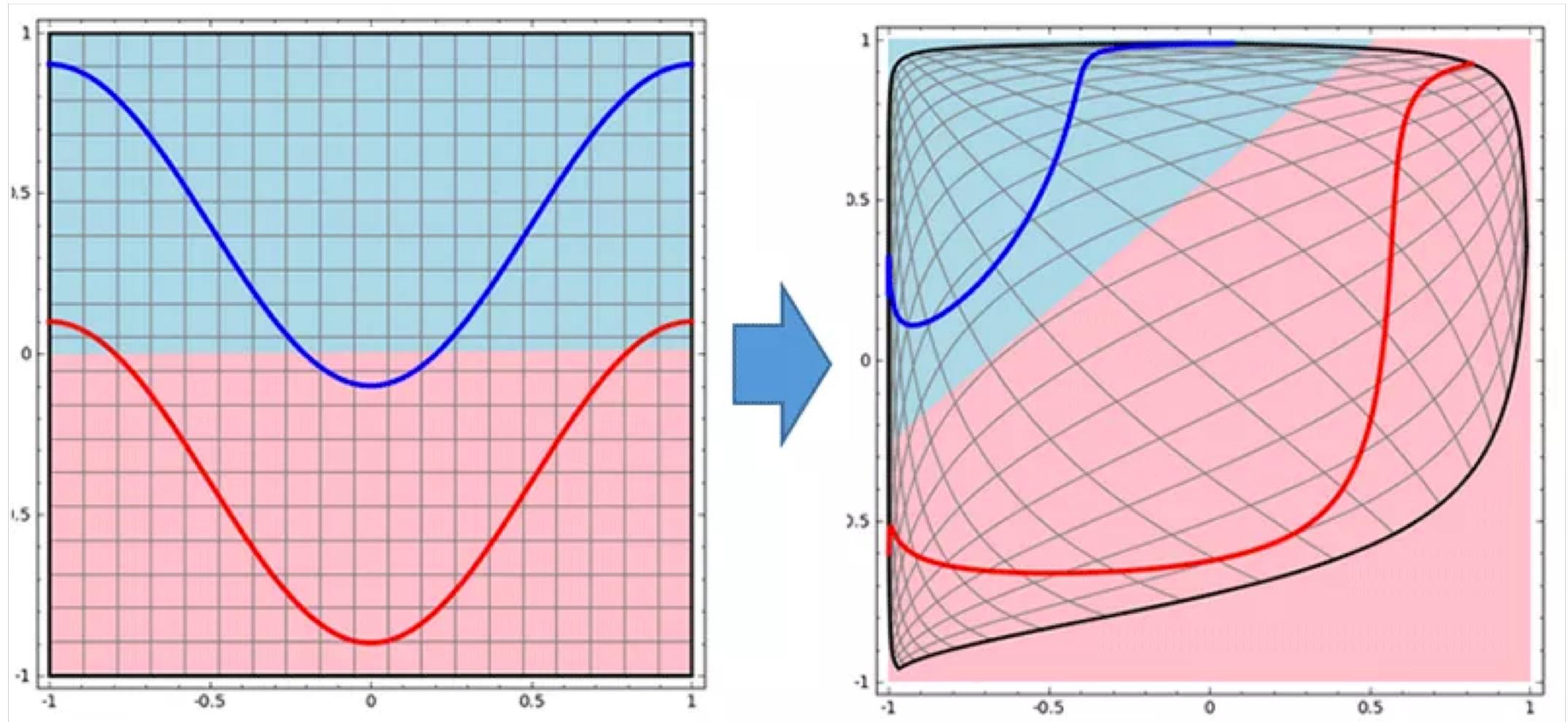
$$y = \text{sgn} (\mathbf{w}^\top \mathbf{x} + b)$$

# Nonlinearity

- Nonlinear transformation



# Nonlinearity



# Activation Function

## ◆ Activation Function:

- 이전 layer의 unit에서의 값과 weight set의 product를 다음 layer의 unit들로 mapping
- 숨겨진 non-linear feature를 반영

## ◆ Different Activation Functions

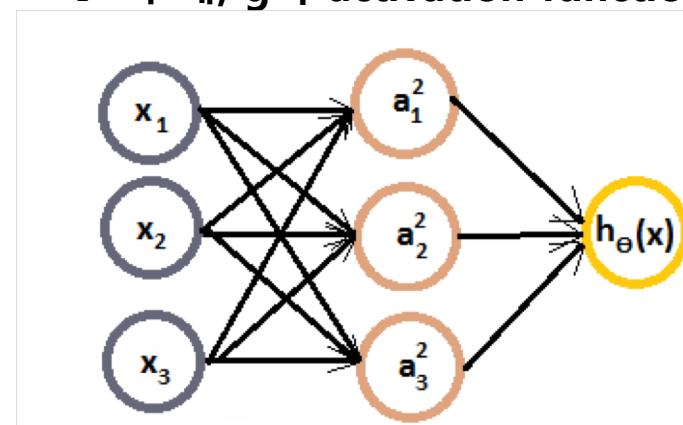
- Identity
- Sigmoid
- Hyperbolic Tangent
- Rectified Linear Unit (ReLU)

### [Example]

$$z_1^{(2)} = \Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3$$

$$a_1^{(2)} = g(z_1^{(2)})$$

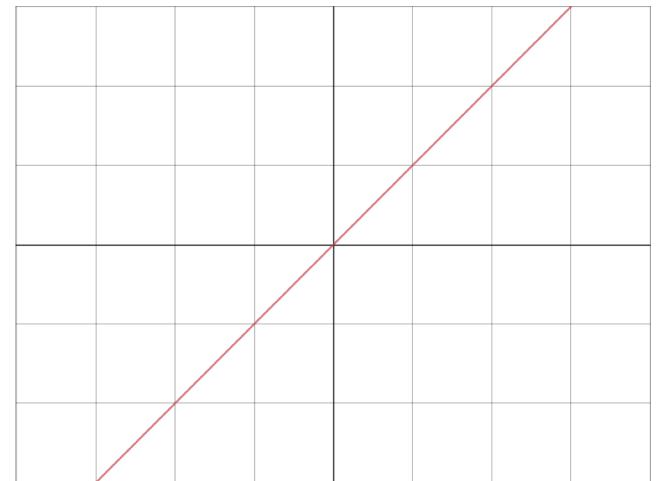
→ 이 때,  $g$ 가 activation function



# Identity / Sigmoid Function

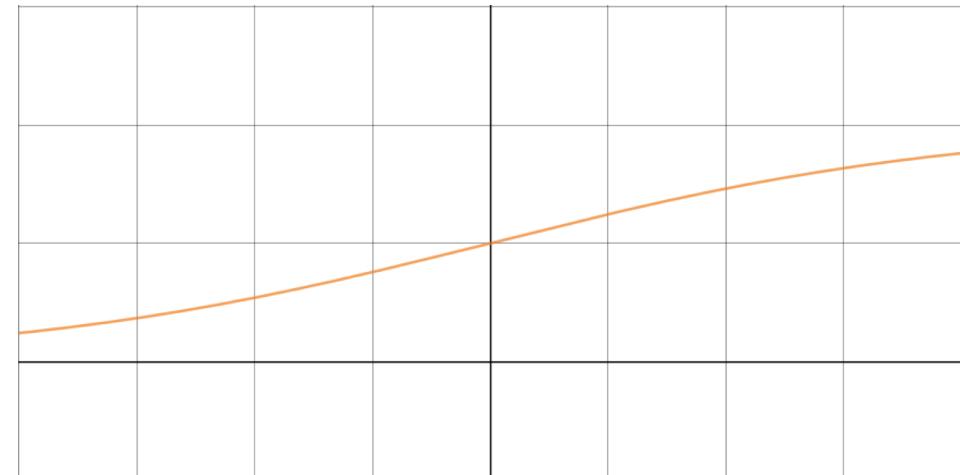
◆ Identity Function:  $f(x) = x$

- Derivative:  $f'(x) = 1$
- Range:  $(-\infty, \infty)$
- Product를 그대로 mapping



◆ Sigmoid Function:  $f(x) = \frac{1}{1 + e^{-x}}$

- Derivative:  $f'(x) = f(x)(1-f(x))$
- Range:  $(0, 1)$
- 값의 범위가 제한적이므로 Classification에서 용이

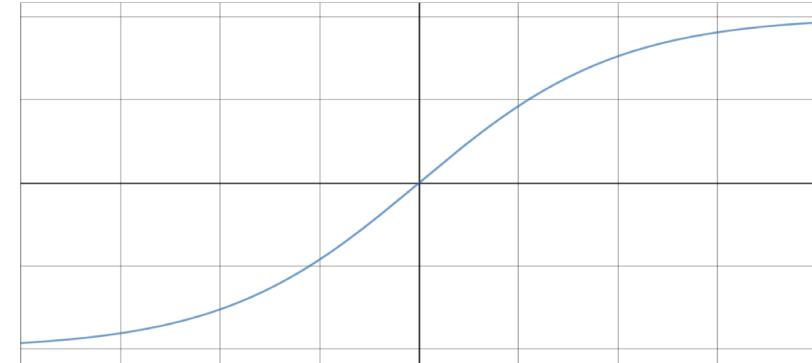


# Hyperbolic Tangent / ReLU Function

## ◆ Hyperbolic Tangent Function:

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

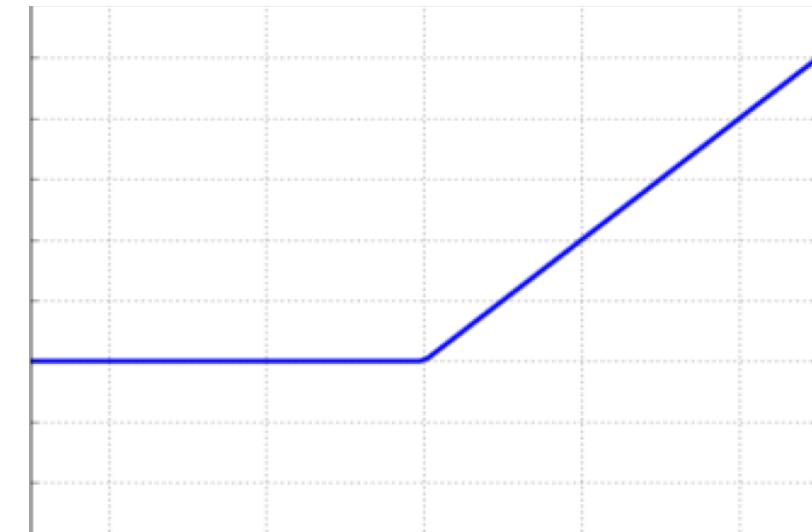
- Derivative:  $f'(x) = 1 - f(x)^2$
- Range: (-1, 1)
- Sigmoid Function과 유사한 특징을 나타냄.



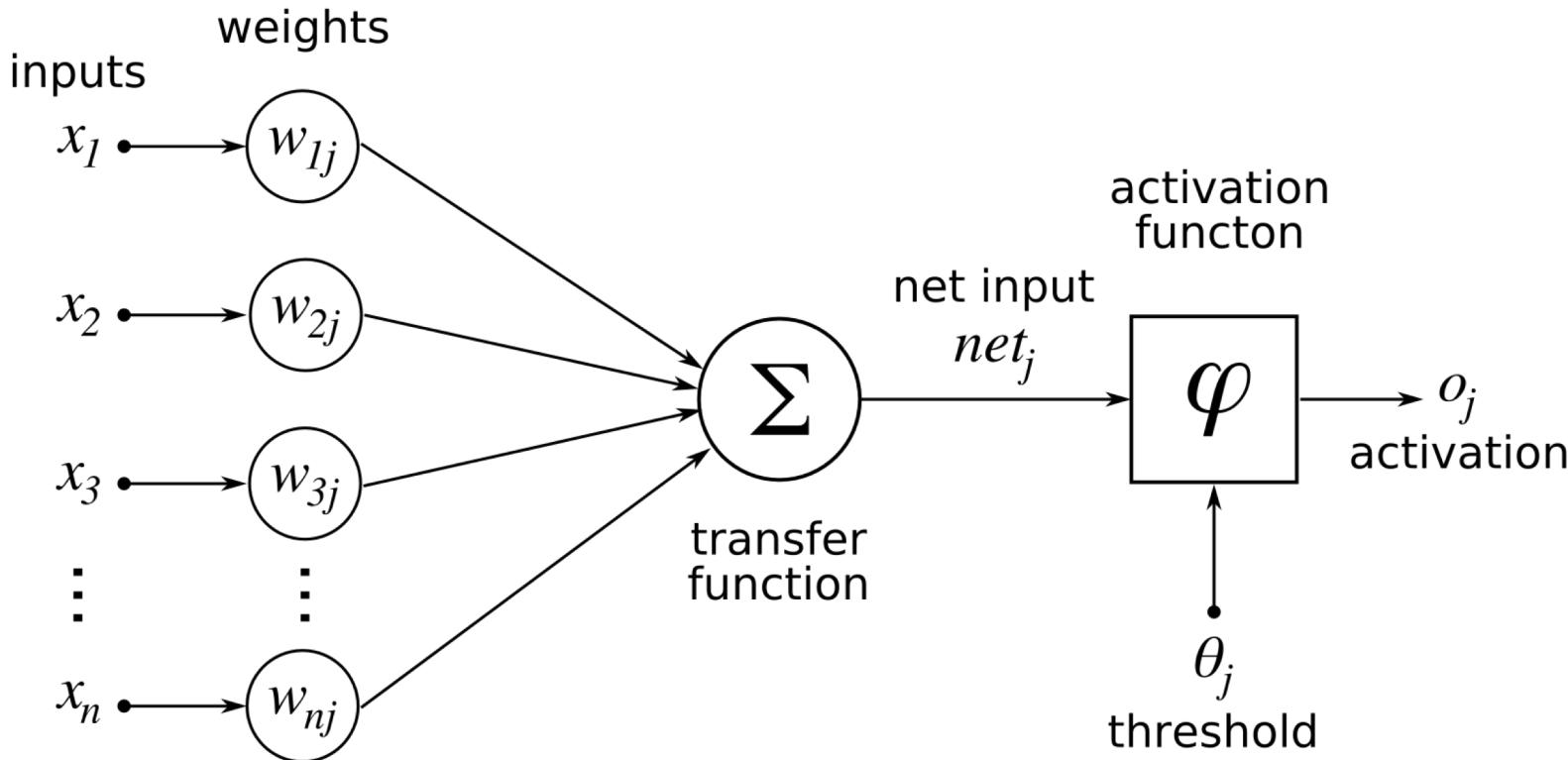
## ◆ ReLU Function:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

- Derivative:  $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
- Range:  $[0, \infty)$
- 기존 Function들의 미분값이 변수의 변화 등을 왜곡하는 문제 해결
- 주로 Computer Vision 분야에서만 사용



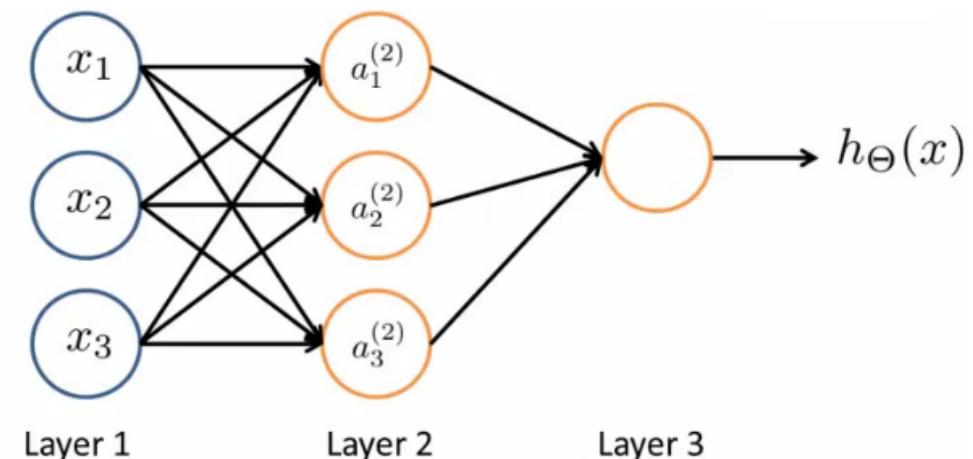
# Neural Network의 구조



- 보통의 neural network는 directed graph로 information propagation이 한 방향으로 고정됨.
- 인접한 layer끼리만 edge를 가짐.
- Network에 Input이 들어오면 weights이 곱해지고 activation function에 의해 각기 다른 노드들이 activate되어 다음 layer로 전달됨.

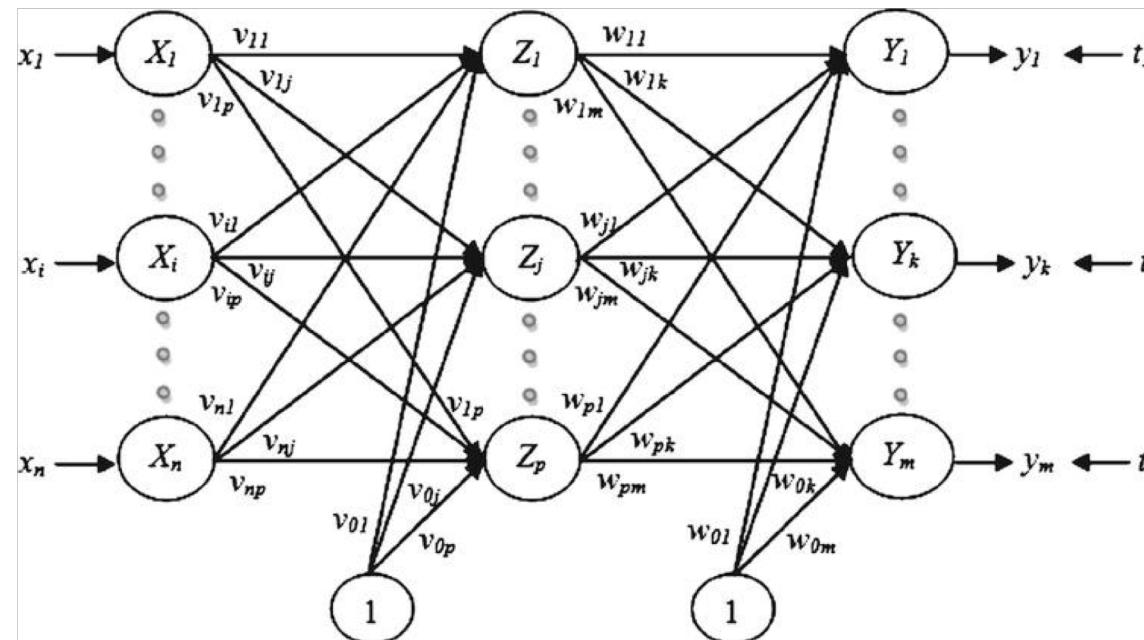
# Terminology

- Node: input을 받아 computation 과정을 거쳐 output 발생
- Computation = Regression hypothesis calculation
  - Input을 받아  $h_\theta(x)$ 를 output으로 내보내는 점이 동일
- 신경망의 각 층위: Layer
  - Input Layer – Hidden Layers – Output Layer
- 각 Layer에는 도식화에선 생략되는 **Bias Unit** 존재
  - Bias unit,  $x_0$ , 는 성분이 모두 1

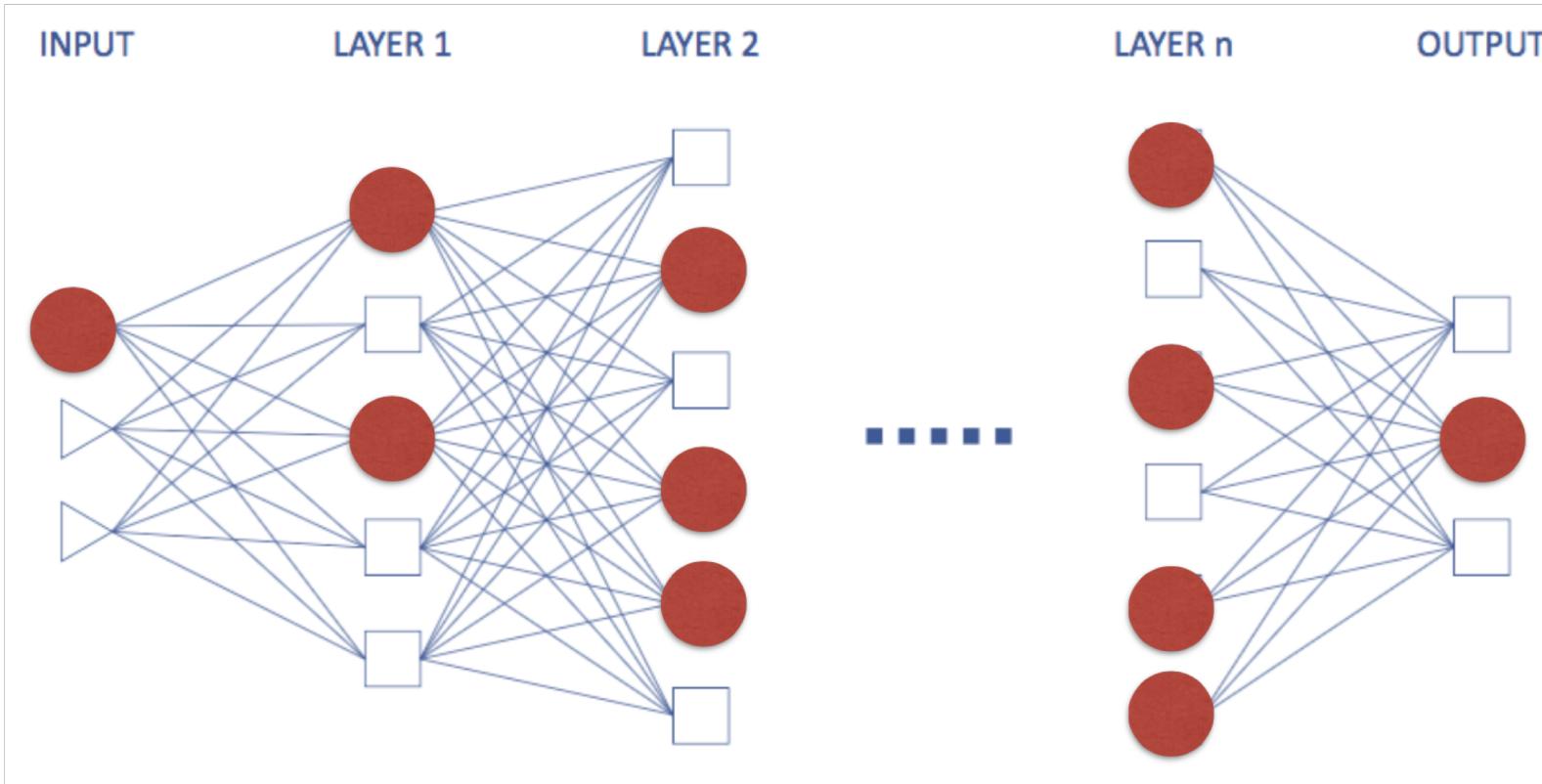


# Multilayer Neural Network

- 하나의 layer의 결과 값은 새 space에서 하나의 feature
- feature 하나로는 prediction 하기엔 표현력이 부족
- 표현력을 강화시키기 위해 여러 개의 perceptron 사용
- 표현력을 강화시키기 위해 feature에 대한 여러 번의 nonlinear transformation



# Inference via Neural Network



- 모든 parameter가 결정되었다고 가정하고 neural network가 결과를 inference하는 과정.
- 각 layer는 주어진 input에 대해 다음 layer의 activation을 결정.
- 맨 마지막 layer의 activation에 의해 output이 결정됨.
- 그림에서 빨간색 노드는 activate된 뉴런
- Classification의 경우 마지막 layer의 노드 개수는 분류하고자 하는 class의 개수와 동일

# Cost Function

- Cost Function은 설립한 hypothesis와 실제 측정값과의 차이
  - Learning의 기준이 되므로 ‘목적 함수’라는 표현 또한 사용
  - 모델의 goal은 cost function을 minimize하는 weights을 찾는 것.
- ◆ Regression:
- Least Square Method
- ◆ Classification:
- Cross-Entropy

# Least Square Method

- ◆ LSM: computes the sum of squared residual, the difference between prediction and label

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

- 위의 Cost Function을 최소화 함으로써 적절한 weight,  $\theta$ 를 구하는 방법
- Linear Regression Problem에서 주로 사용
- Gradient:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y) \\ &= (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left( \sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_{\theta}(x) - y) x_j\end{aligned}$$

# Cross-Entropy Method

## ◆ Cross-Entropy:

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

- 오차에 기하급수적으로 비례해 MSE보다 훨씬 더 민감하게 반응.
- 특히 Activation Function으로 Sigmoid Function 사용시, Sigmoid Function의 제한된 미분계수 범위로 인해 생기는 MSE의 문제를 해결.
- Gradient:

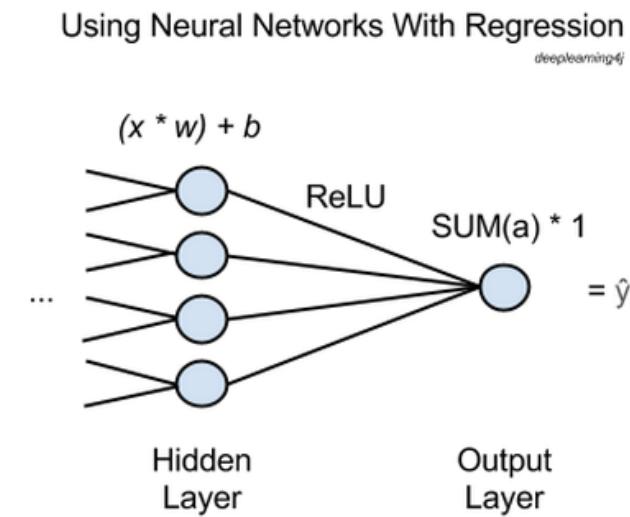
$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- MSE와 마찬가지로, Gradient가 prediction과 label의 차이에 비례

# Cost Function: Regression

◆ **Cost Function:**  $J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$

- How to use NN for regression?
- Regression을 수행하는 NN은 output이 1개
- The input is combined in various ways, multiplied by the weights, added to the bias.
- The sum of the products is fed to the activation function
- Output은 단순히 직전 node의 activation의 합에 1을 곱한 값
- 결과로 나온 NN의 예상값:  $\hat{y}$



# Cost Function: Classification

◆ **Cost Function:**  $J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$

- Classification problem은  $k$ 개의 class로 분류
- $h_\theta(x) \in \mathbb{R}^k$
- Sum up from 1 to  $k$  ( $k$ 는 # of output nodes)
- 위 등식의 generalization

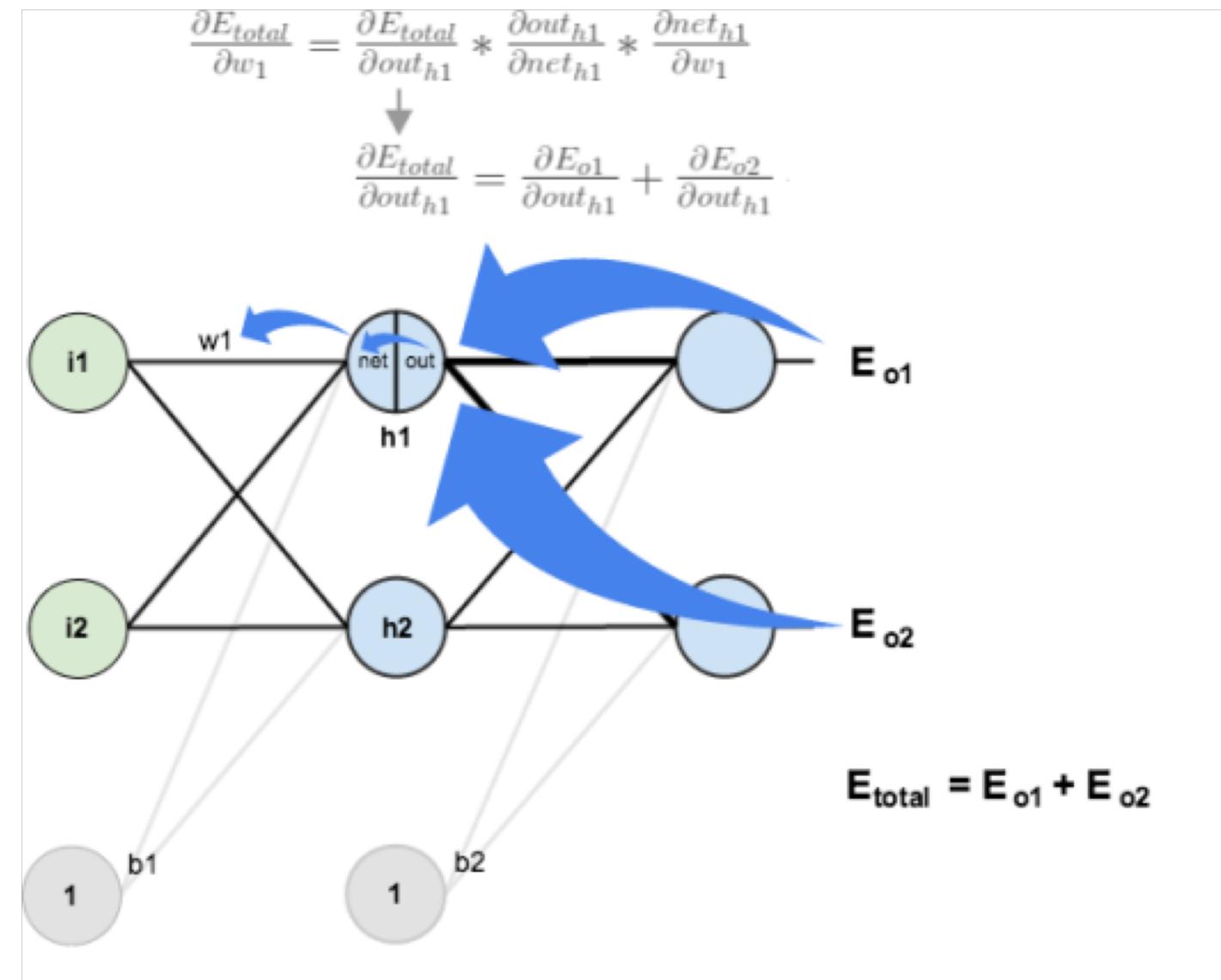
$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$
$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

# implementation

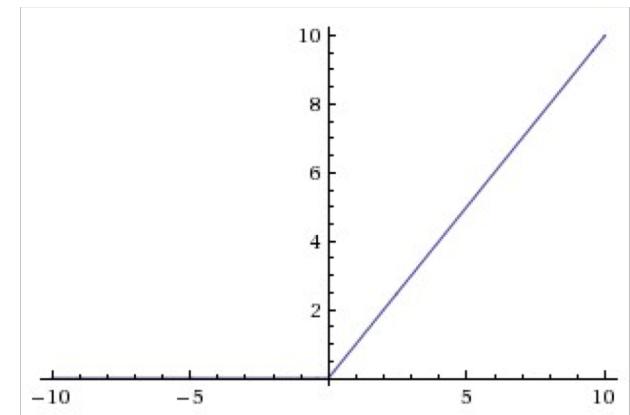
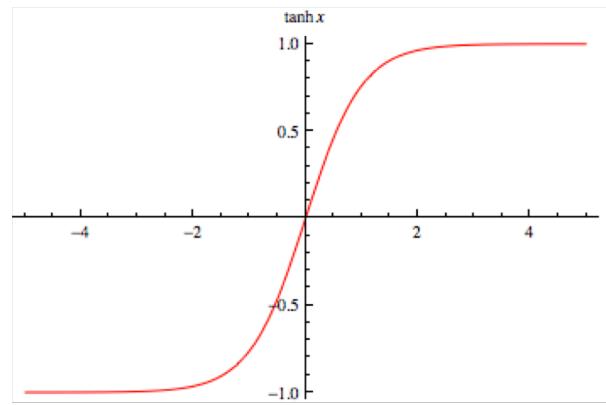
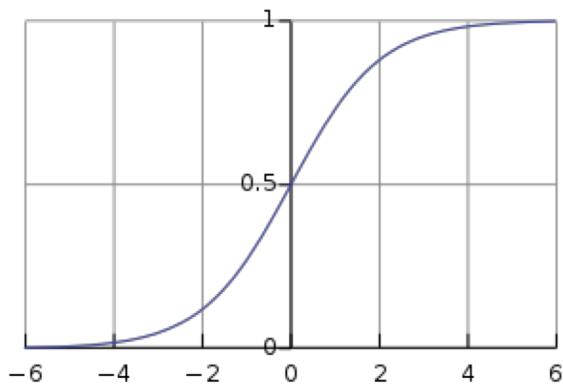
- Perceptron 하나에 필요한 연산들
  - Input  $x$ 의 요소들과  $w$ 의 element-wise 곱
  - Element-wise 곱 결과의 합
- 여러 노드를 사용하도록 구현해야 한다.
  - 모델의 표현력 증대
- 데이터를 한 번에 하나씩 학습하면 비효율적
  - 여러 개의 데이터를 한 step에 학습할 수 있어야 한다 (batch)
- `tf.matmul`을 사용하여 위의 모든 요구사항을 충족시킬 수 있다.

# Back propagation

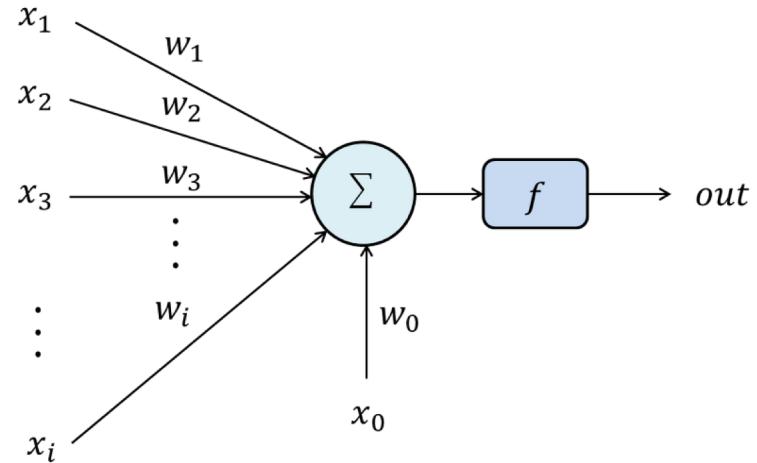
- tf.train.Optimizer 사용



# Activation function



# implementation



D: Feature dimension

B: Batch\_size

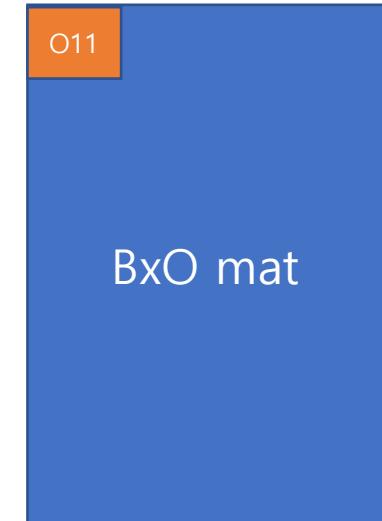
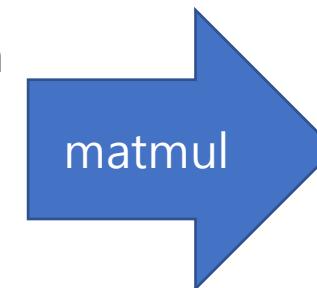


O: # of node

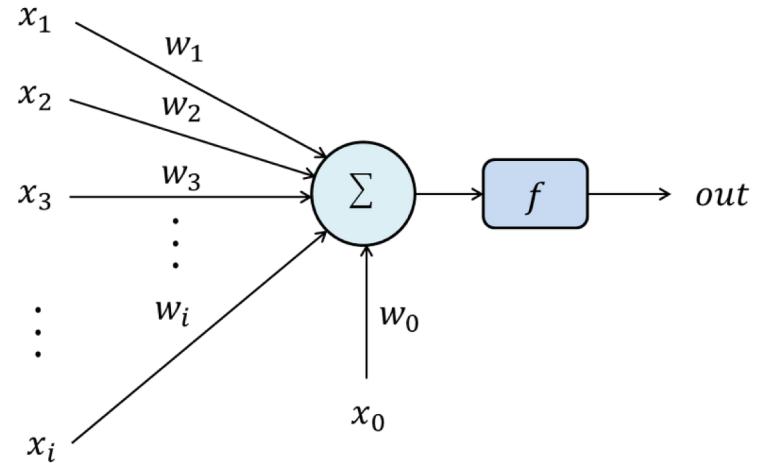


\*

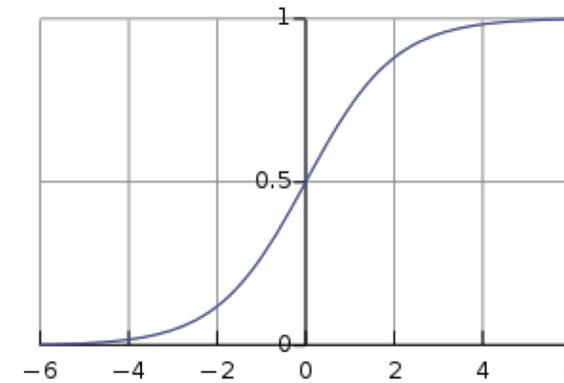
D: Feature dimension



# implementation



Nonlinear activation function 적용



# Feature 변환이 끝난 후

- 변형된 feature를 이용하여 분류 혹은 회귀문제를 푼다.
- 마지막 layer의 output 차원은 풀고 싶은 문제에 따라 달라짐
  - Classification: 마지막 layer의 node 수를 class 수에 맞춤
    - 결과 값을 확률로 나타내기 위해 softmax (multinomial logistic regression) 사용
    - Softmax 결과 값을 이용하여 cross entropy를 minimization
  - Regression: 예측하고 싶은 값의 개수에 따라 맞춤
    - 결과 값을 이용하여 MAE, RMSE 등을 한 후 minimization
- 마지막 layer에는 nonlinear activation function을 쓰지 않음

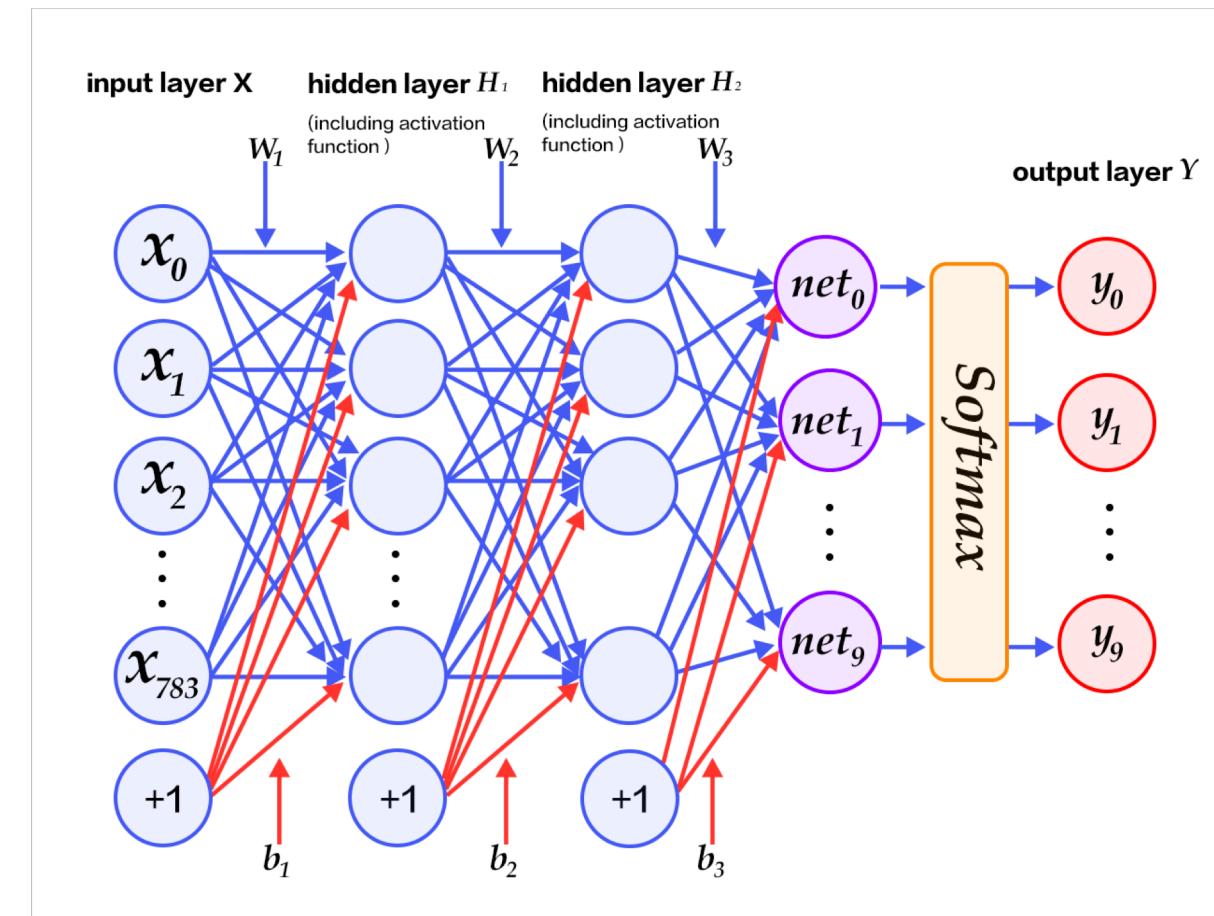
# MNIST를 분류하는 모델 구현하기

- 숫자 이미지를 실제 int형 정수로 인식하는 문제
- $P(0 | \text{image}), P(1 | \text{image}) \dots P(9 | \text{image})$
- MNIST input은 28x28 이미지이나 784차원 벡터로 flatten하여 MLP에 input로 넣음
- 자료의 수: 55,000개의 training image, 5,000개의 validation image, 10,000개의 test image.
- Tensorflow 안에 tutorial로 자료가 존재 (`tensorflow.examples.tutorials.mnist`): 바로 읽어 들일 수 있음.



# MNIST를 분류하는 모델 구현하기

- `tf.placeholder(dtype, shape, name)`
- `tf.reshape`
- `tf.matmul(a, b, ...)`
- `+`
- `tf.sigmoid(x, name)`
- `tf.nn.softmax(logits, ...)`
- `tf.argmax()`
- `tf.equal()`
- `tf.cast()`
- `tf.reduce_mean()`



# MNIST를 분류하는 모델 구현하기

- Cross entropy

$$CrossEntropy(prediction, y) = \frac{1}{n} \sum_{j=1}^n \sum_{i=1}^d \{-y_{ji} \log(prediction_{ji})\}$$

# 필요 함수들

```
tf.placeholder(dtype, shape)
tf.get_variable(name, initializer=None)
tf.constant(value, shape=None)
tf.random_normal(shape) # shape 크기를 갖고 요소가 normal distribution을 따르는 랜덤값
tf.truncated_normal(shape) # shape 크기를 갖고 요소가 normal distribution을 따르는 랜덤값
tf.zeros(shape) # shape 크기를 갖고 요소가 모두 0인 행렬
tf.matmul(a, b) # 행렬 곱. a*b
tf.reduce_mean(x, axis) # 행렬 x의 요소들 평균
tf.reduce_sum(x, axis) # 행렬 x의 요소들 합
tf.equal(a, b) # a==b 이면 true
tf.add_n(x) # 요소별 덧셈
tf.argmax(x, axis) # 행렬 x의 요소 중 최댓값의 인덱스
tf.cast(x, dtype) # 행렬 x의 요소들의 데이터 타입을 dtype으로 변환
tf.nn.l2_loss(x) # l2 loss를 계산하는 레이어
tf.nn.sigmoid(x, name=None) # sigmoid function을 적용하는 레이어
tf.nn.softmax(x) # softmax function을 적용하는 레이어
tf.trainable_variables() # train에 사용되는 변수들을 반환
```