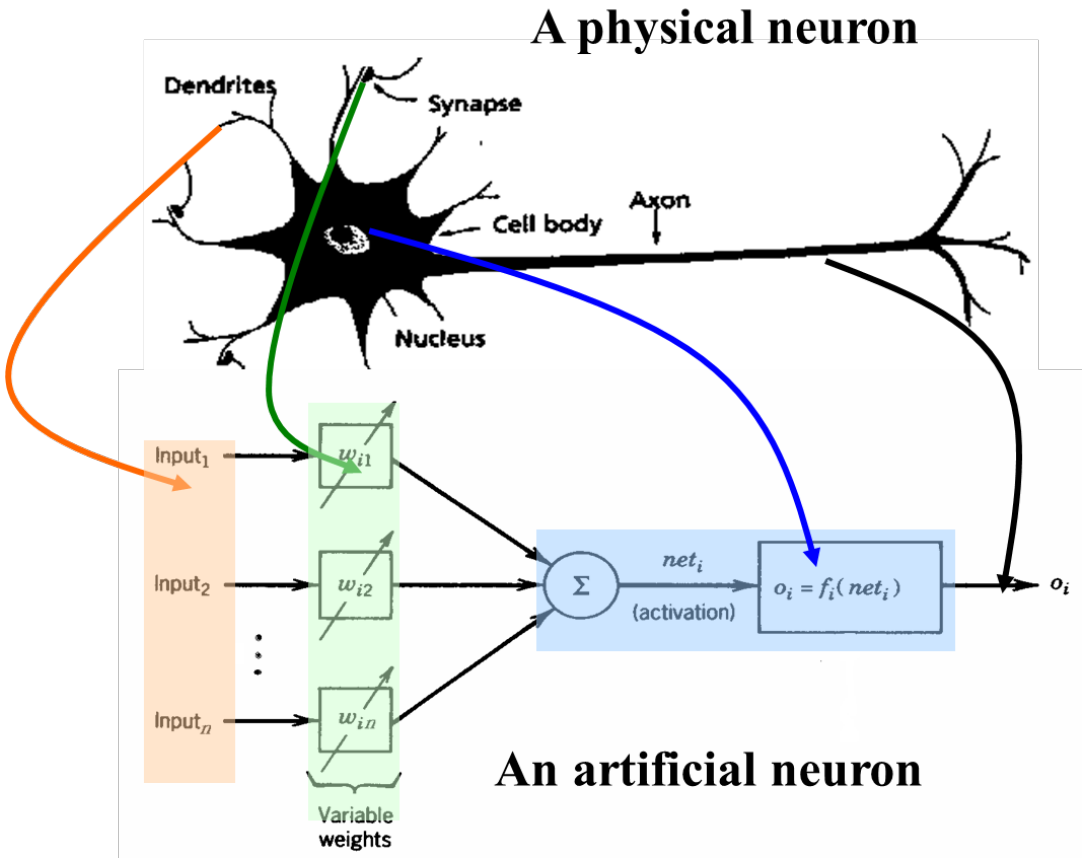
Multilayer Perceptron

Sangheum Hwang

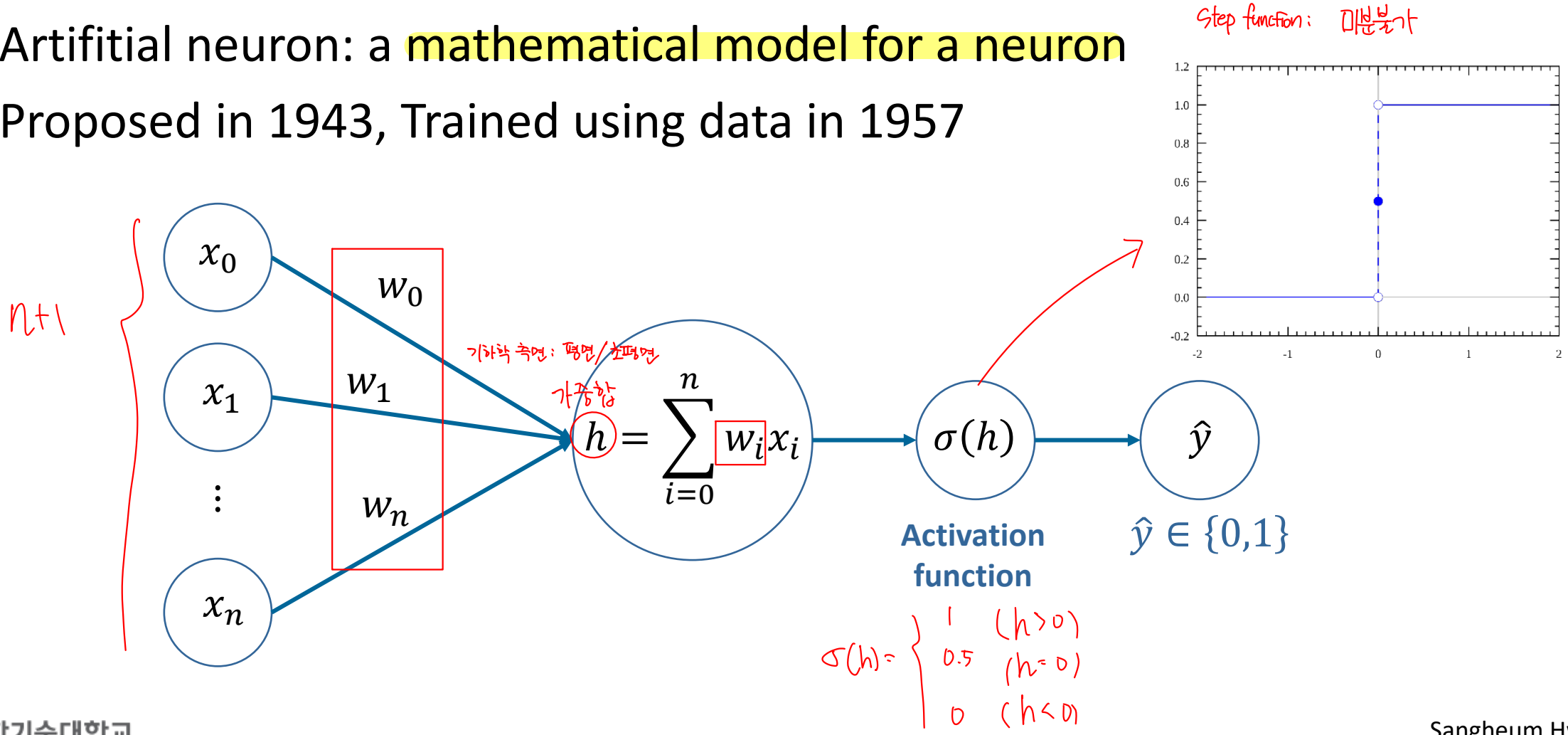
Biological Neuron

- How to represent a physical neuron as a mathematical model?

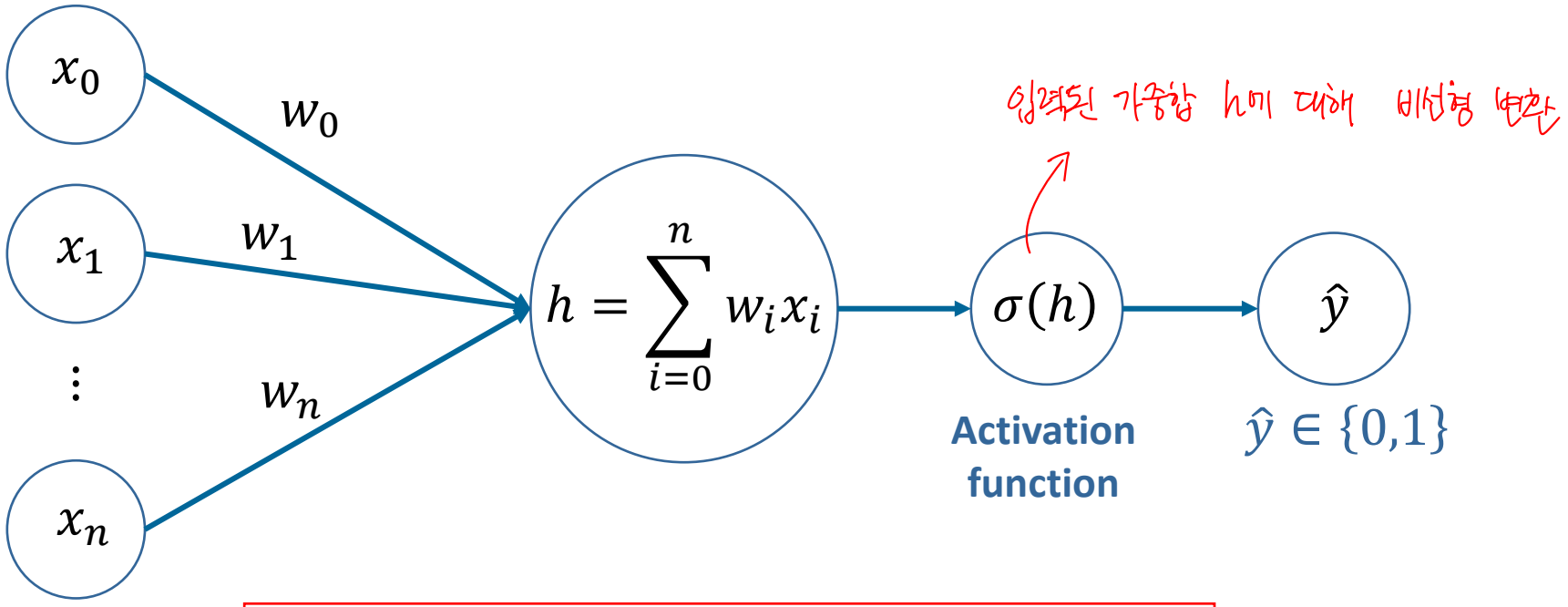


Perceptron

- Artifitial neuron: a mathematical model for a neuron
- Proposed in 1943, Trained using data in 1957



Perceptron

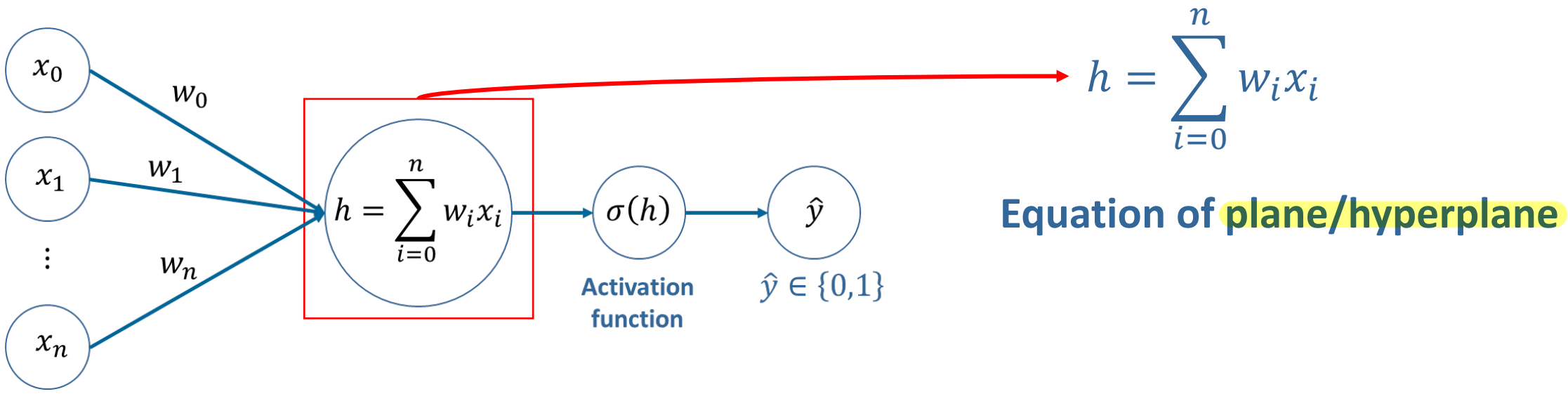


$$\hat{y} = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron

선형 결정 경계 ⇒ data는 하나의 직선, 평면으로 나눠 두 개의 클래스로 분류

- It describes a linear decision boundary with respect to inputs.

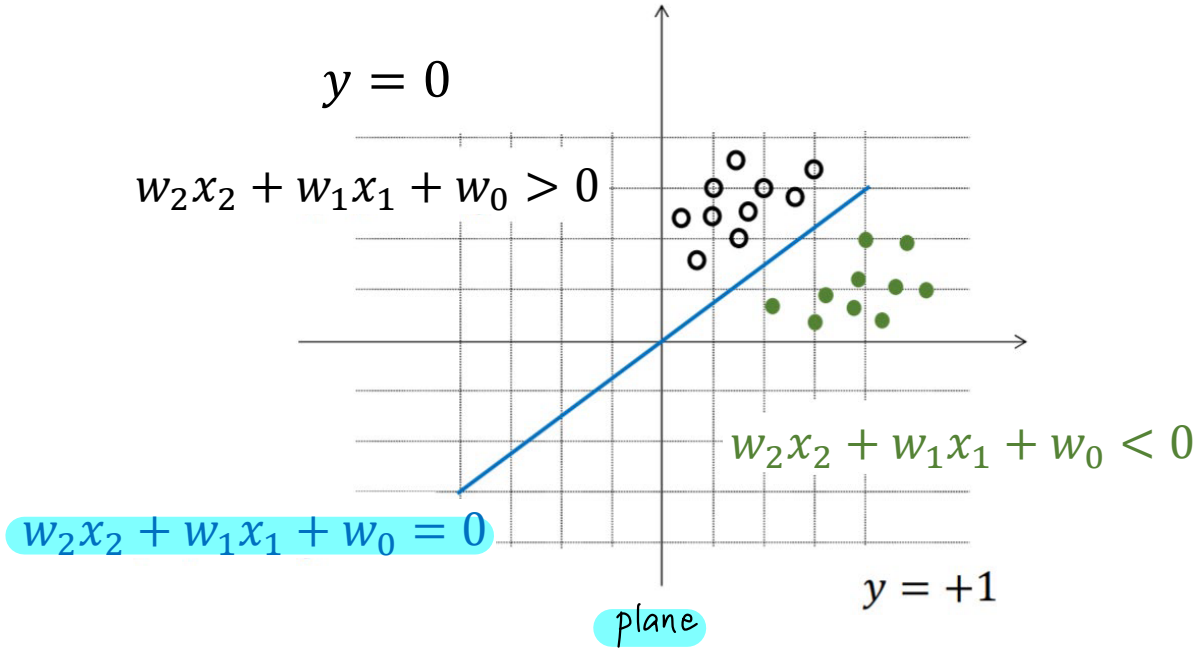


Perceptron

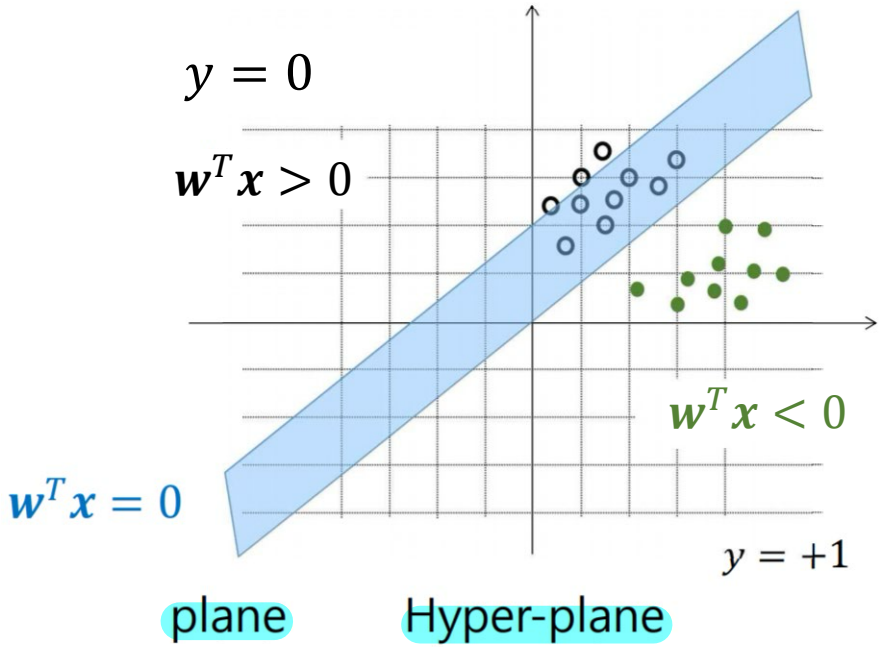
$$h = \sum_{i=0}^{n} w_i x_i$$

Equation of plane/hyperplane

N차원 초평면에서 N-1 초평면이 선형결정경계.



[Two input variables]



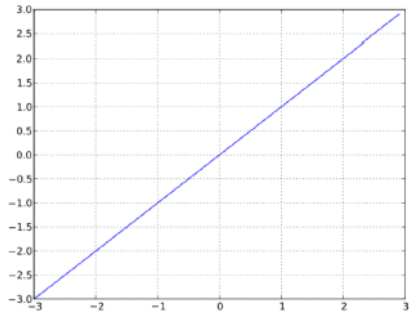
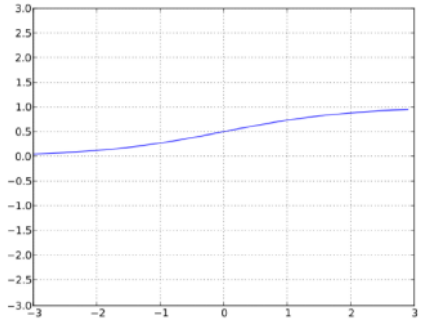
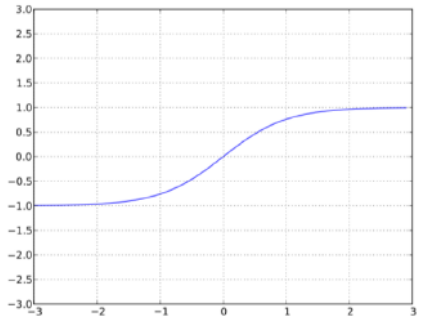
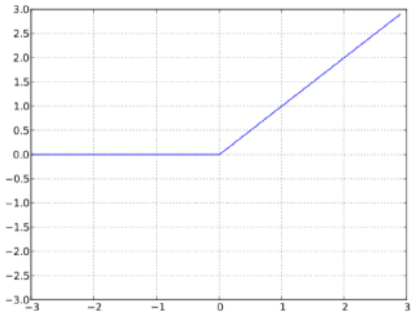
[Multiple input variables]

신경망에서 가중치 곱하고 더하기 : linear transformation.
⇒ activation 함수가 linear면 다층 레이어도 결국 하나의 선형 함수로 표현!

Activation Functions

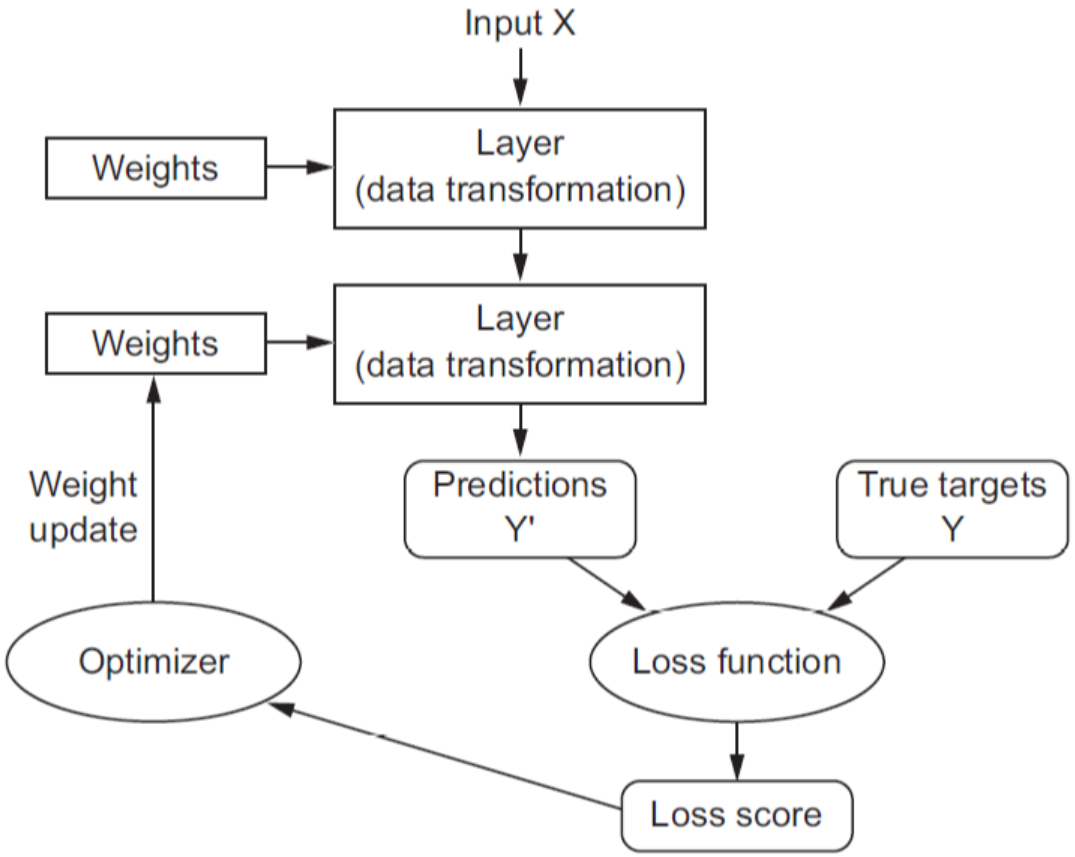
why necessary? { 모델 복잡도 ↑ / 비선형 문제해결. / multi-layer 가능.

- **Nonlinear transformation of a weighted sum of inputs**

Linear	Sigmoid	Tanh	ReLU
$g(a) = a$	$g(a) = \frac{1}{1 + \exp(-a)}$	$g(a) = \tanh(a) = \frac{\exp(2a) - 1}{\exp(2a) + 1}$	$g(a) = \max(0, a)$
			
• Linear transform • No input squashing	• Output → (0,1) • Always positive • Bounded • Strictly increasing	• Output → (-1,1) • Can be positive/negative • Bounded • Strictly increasing	• Always non-negative • Unbounded • Strictly increasing • Sparse activities

Training of Perceptron

- Recall that we have to define a loss function to evaluate how well a model fits given data.



binary classifier

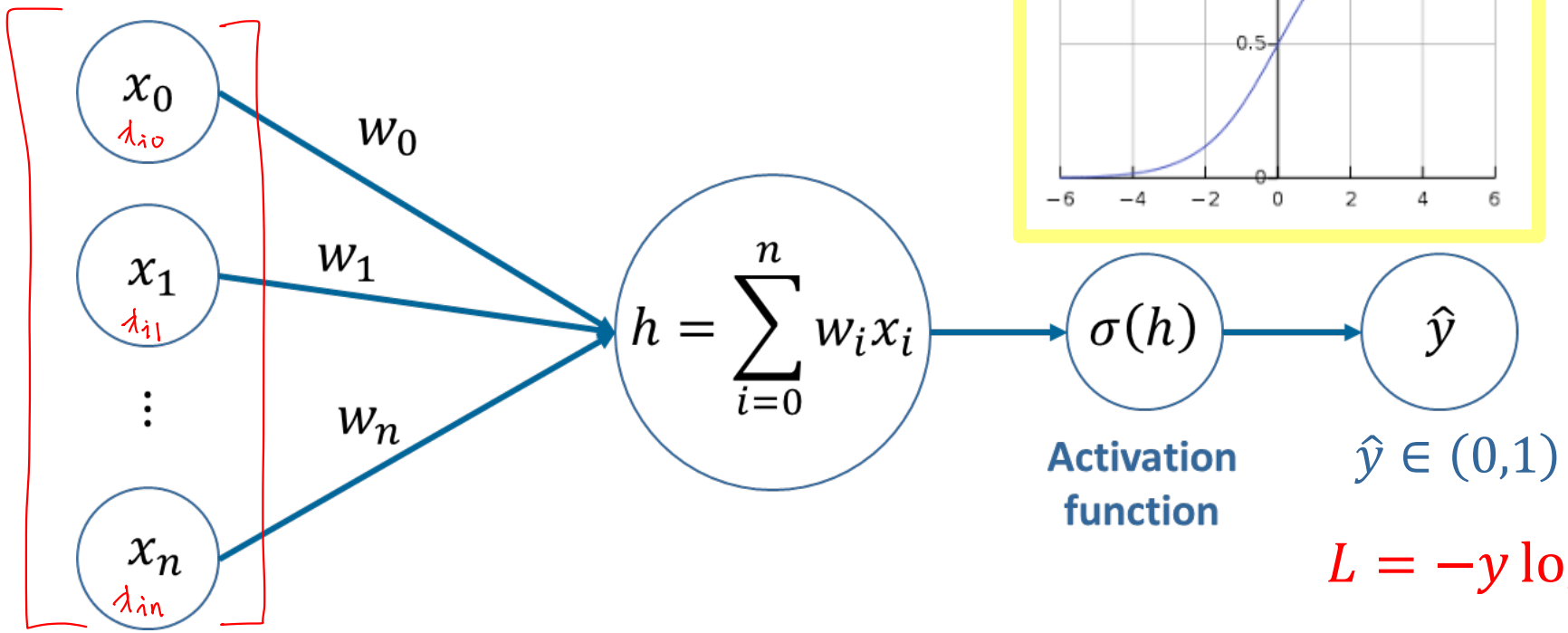
Training of Perceptron

- Consider a sigmoid function for an activation function.

$N \begin{cases} X_1, Y_1 \\ X_2, Y_2 \\ \vdots \\ X_N, Y_N \end{cases}$ $Y_i \in \{0, 1\}$

$X_i = \begin{bmatrix} x_{i0} \\ x_{i1} \\ \vdots \\ x_{in} \end{bmatrix}$

X_i $Y_i = 1$



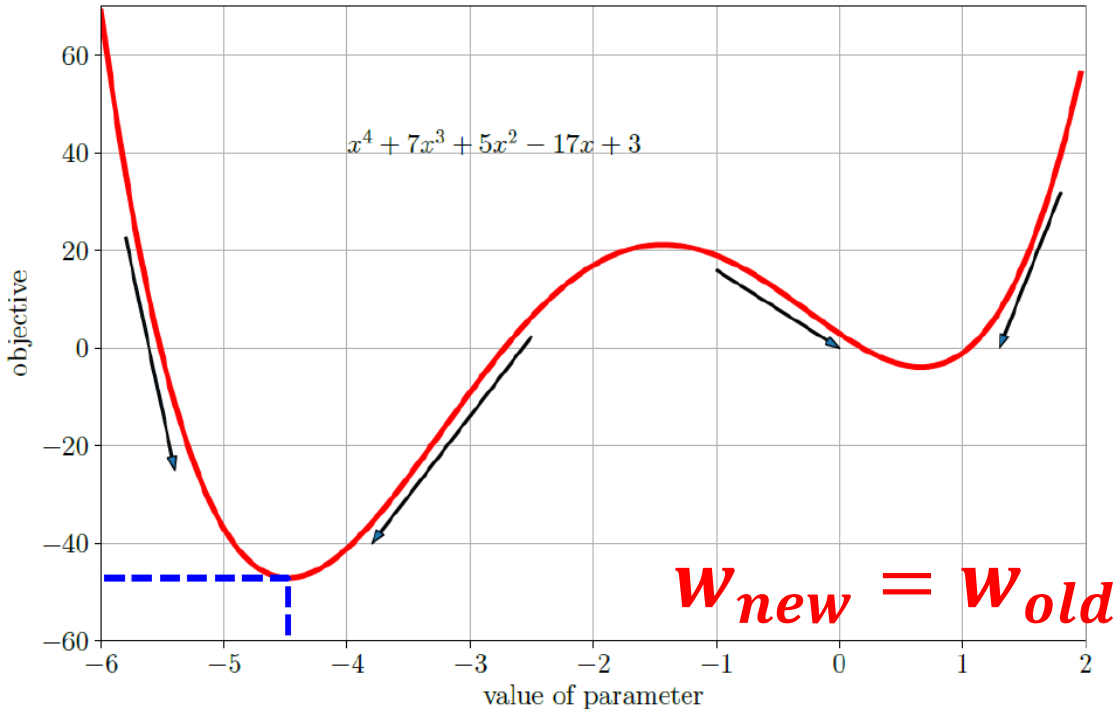
if $y = 0$, $L = -\log(1-\hat{y})$
if $y = 1$, $L = -\log(\hat{y})$

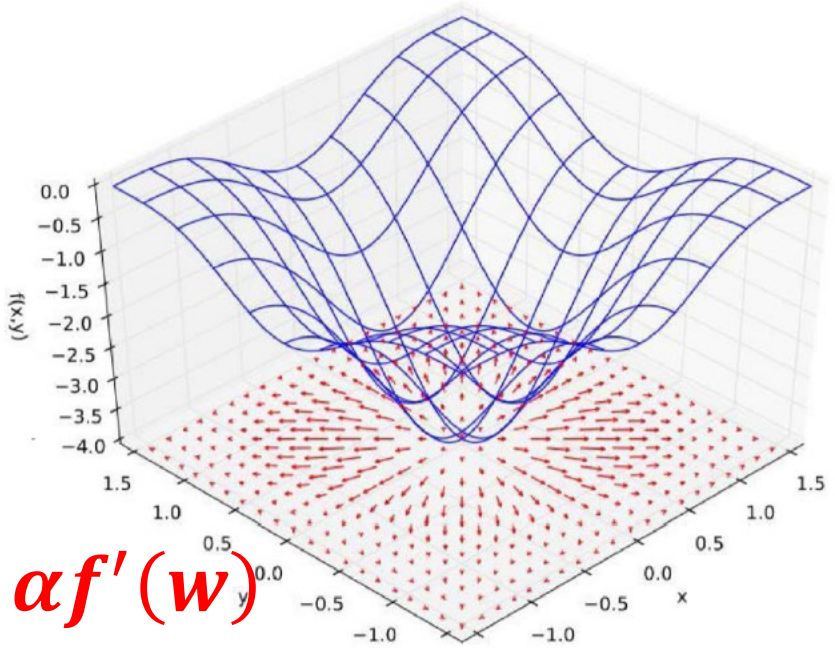
Negative log-likelihood

$$L = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

Training of Perceptron

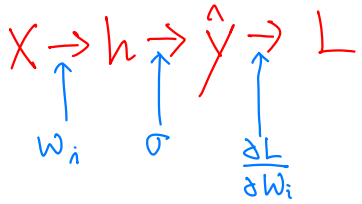
- How to train? → minimize loss function
- How to minimize? → gradient descent method

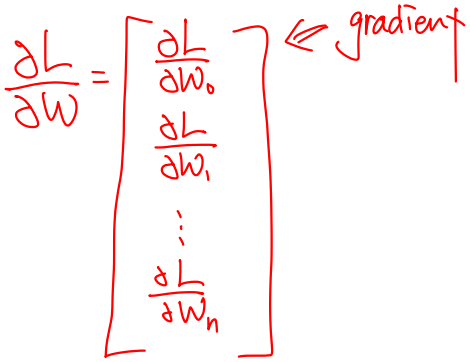




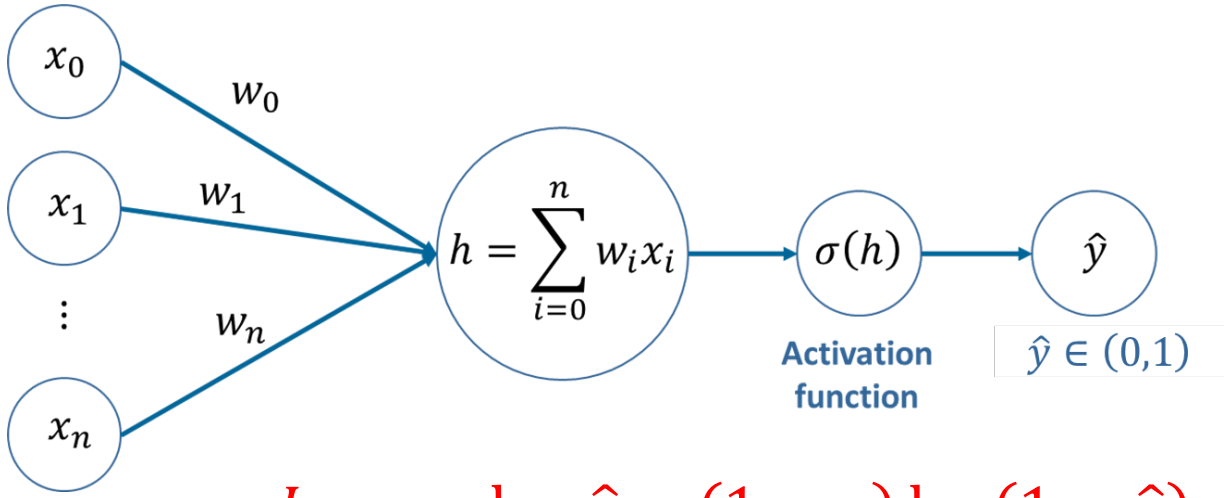
$$w_{new} = w_{old} - \alpha f'(w)$$

Training of Perceptron





- What do we need to perform the gradient descent method?
  - All you need is just the gradient!



$$L = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \frac{\partial h}{\partial w_i} = (\hat{y} - y) x_i$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

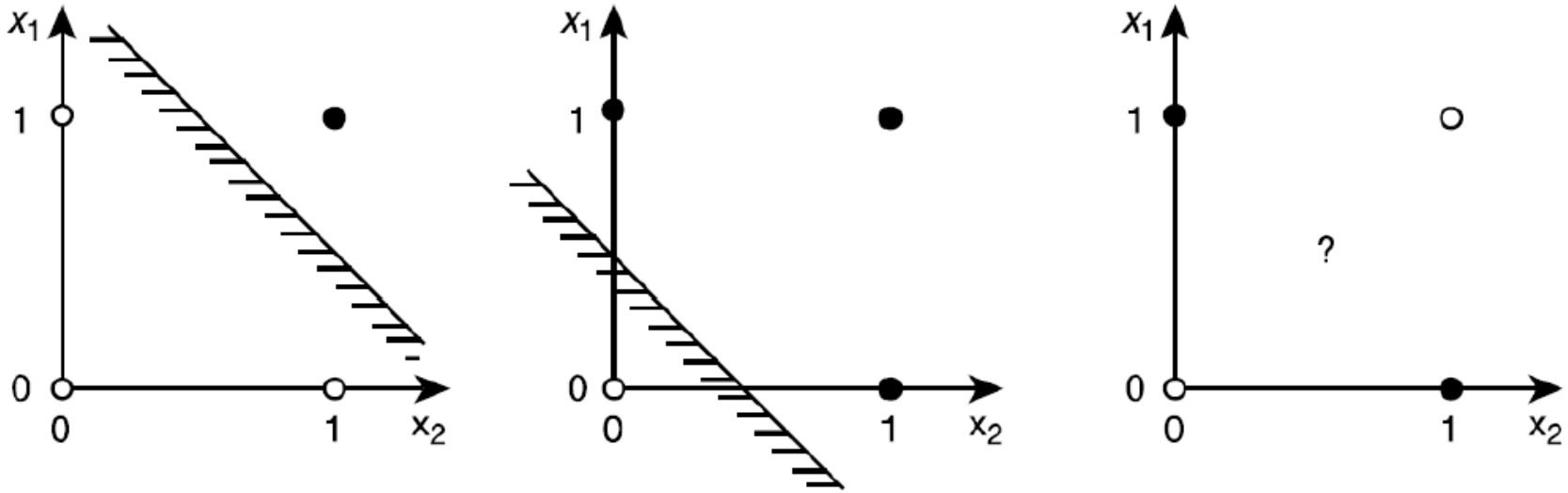
$$\frac{\partial \hat{y}}{\partial h} = \frac{1}{1+\exp(-h)} \cdot \frac{\exp(-h)}{1+\exp(-h)} = \hat{y}(1-\hat{y})$$

$$\frac{\partial h}{\partial w_i} = x_i$$

$$w_{new} = w_{old} - \alpha(\hat{y} - y) x_i$$

based on linear model

Limitation of Perceptron



(a) x_1 and x_2 (b) x_1 or x_2 (c) x_1 xor x_2

- AND, OR, NOT gates can be implemented by perceptron, but not XOR gate.

AND

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

OR

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

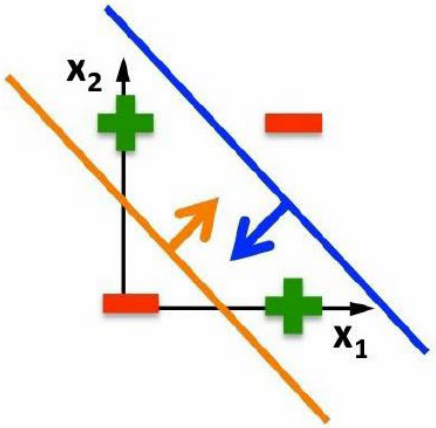
XOR

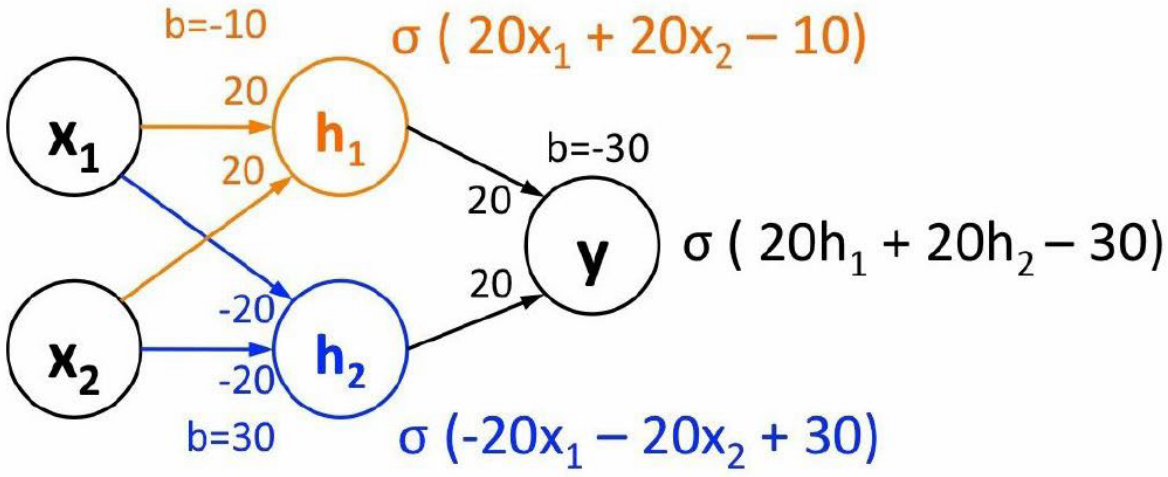
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Limitation of Perceptron

- But, we can solve XOR problem if we use two perceptrons together!

Linear classifiers
cannot solve this





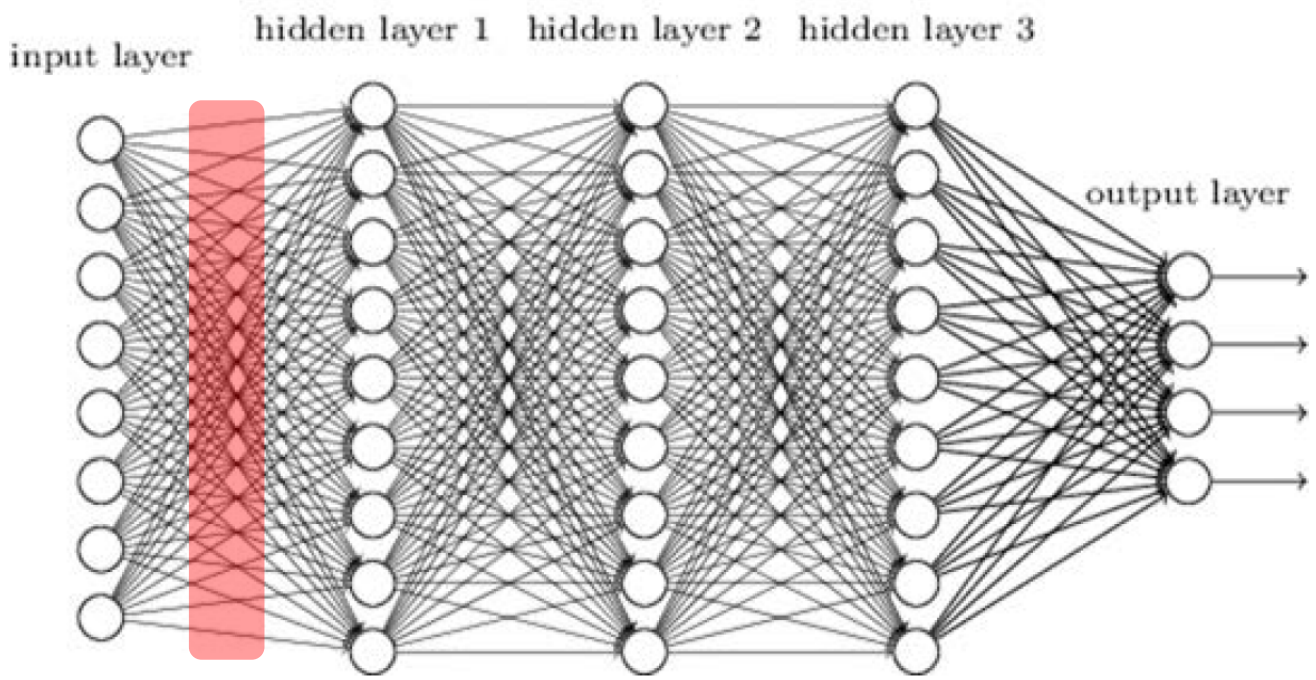
$\sigma(20*0 + 20*0 - 10) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$
$\sigma(20*0 + 20*1 - 10) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$

$\sigma(-20*0 - 20*0 + 30) \approx 1$
$\sigma(-20*1 - 20*1 + 30) \approx 0$
$\sigma(-20*0 - 20*1 + 30) \approx 1$
$\sigma(-20*1 - 20*0 + 30) \approx 1$

$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*1 - 30) \approx 1$

Multi-layer Perceptron

- Multi-layered structure with many perceptrons
- Connections between layers, no connections within layers



fully connected

Multi-layer Perceptron

- It can model an arbitrarily nonlinear function if its capacity is sufficient.

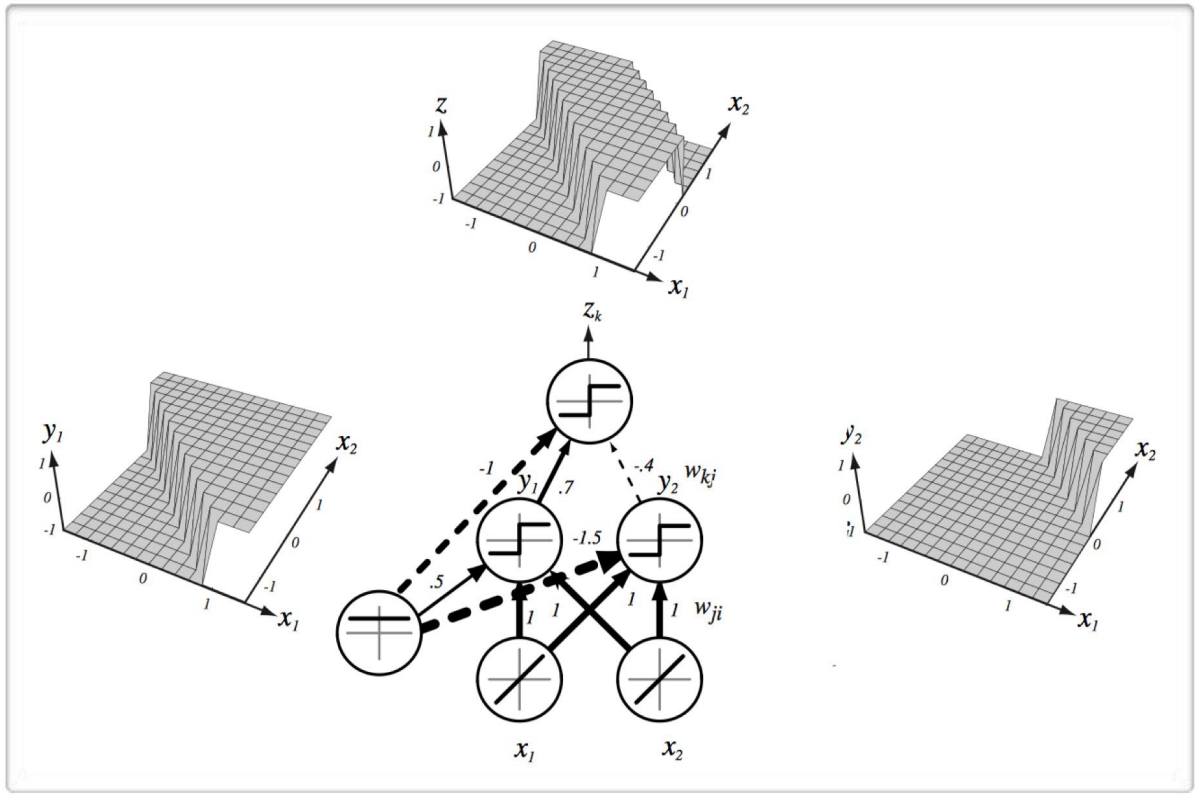


Image credit: Pascal Vincent

Multi-layer Perceptron

- It can model an arbitrarily nonlinear function if its capacity is sufficient.

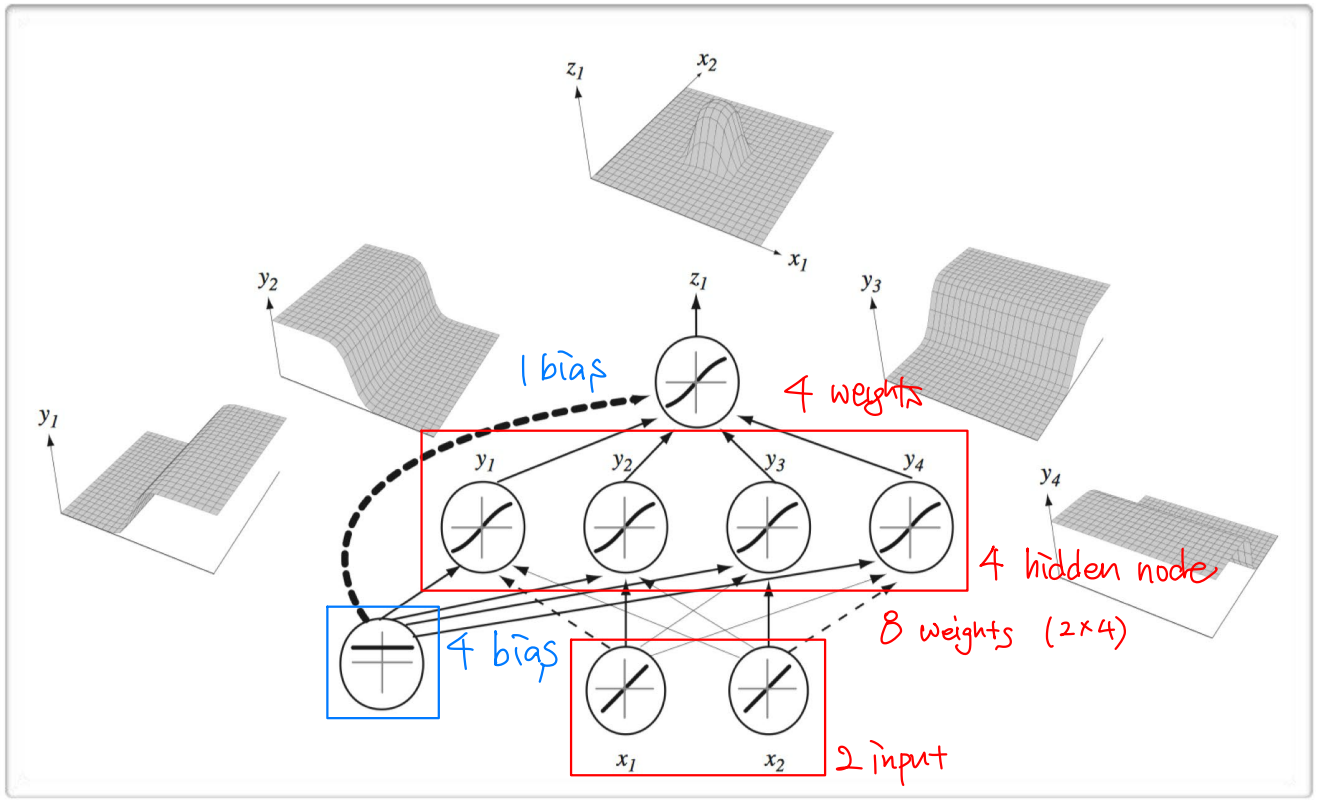


Image credit: Pascal Vincent

Loss functions

- Typical loss functions
  - Regression problem = squared loss $L = \frac{1}{2}(y - \hat{y})^2$
  - k-class classification problem = cross-entropy loss

$$L = -\sum_{i=1}^{k} y_i \log(\hat{y}_i) \quad \text{where} \quad y_i = \begin{cases} 1 & \text{if} \quad \text{label} = i \\ 0 & \text{otherwise} \end{cases}$$

$y_i = \{ [1, 0, \cdots, 0], [0, 1, \cdots, 0], \vdots, [0, 0, \cdots, 1] \}$

one-hot encoding

$y_i = [1, 0, 0, \cdots, 0]$ $\hat{y}_i = Prob(\text{label} = i)$

predicted probability

$L = -\log(\hat{y}_i)$

L 감소하는 방향으로!

Training of Perceptron

- Typical loss functions
  - How to convert the outputs to the probabilities → Softmax function

$$\hat{y}_i = \frac{\exp\{h_i\}}{\sum_{i=1}^{k} \exp\{h_i\}}$$

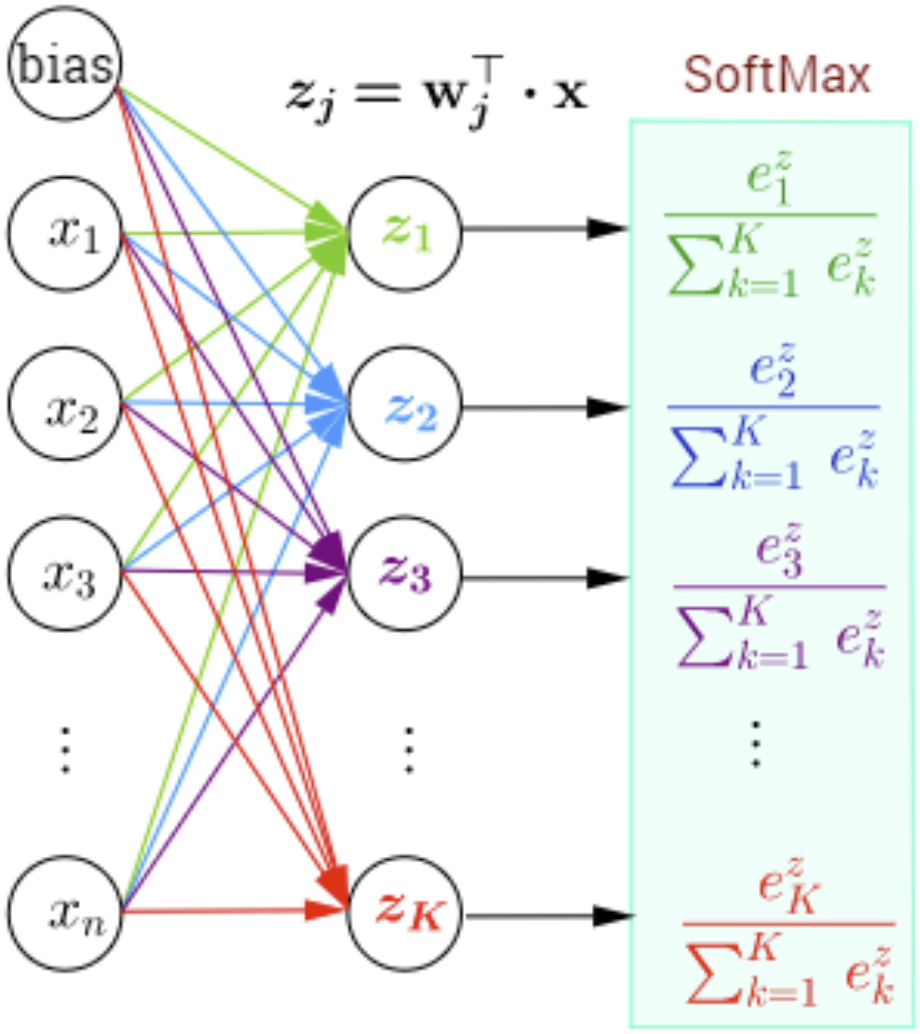
continuous

$\hat{y}_i$'s are strictly positive.

$$\sum_{i=1}^{k} \hat{y}_i = 1$$

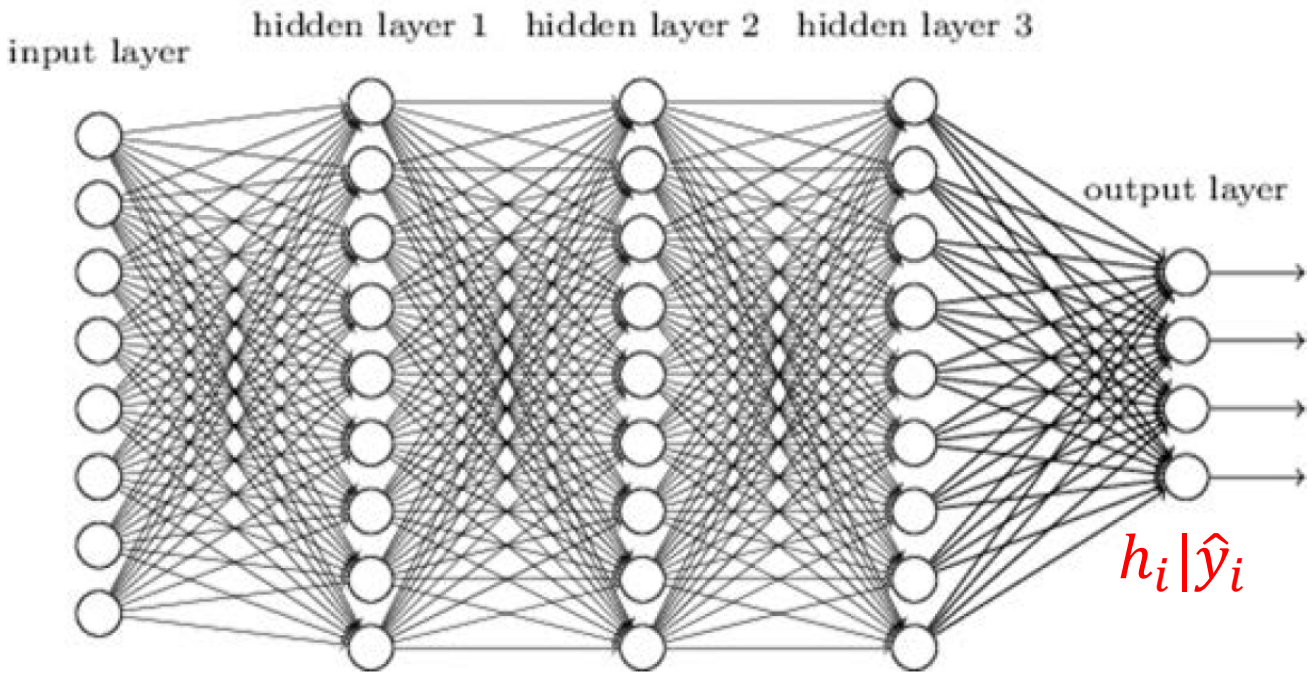
Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Supervised Learning with Multi-layer Perceptron

- Regression: output real values → squared loss
- Classification: output probabilities via softmax function → cross-entropy loss



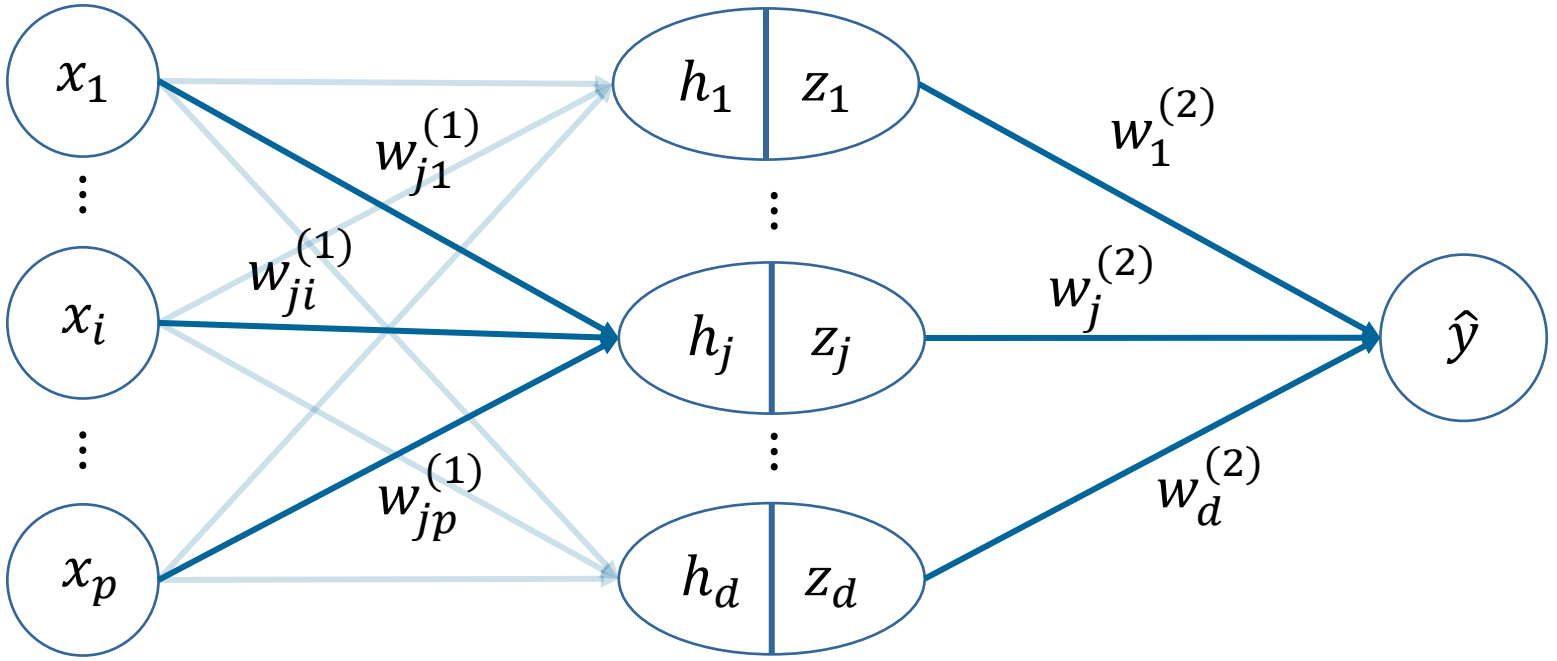
For regression, $\hat{y}_i = h_i$

For classification, $\hat{y}_i = \frac{\exp\{h_i\}}{\sum_{i=1}^{k} \exp\{h_i\}}$

Softmax

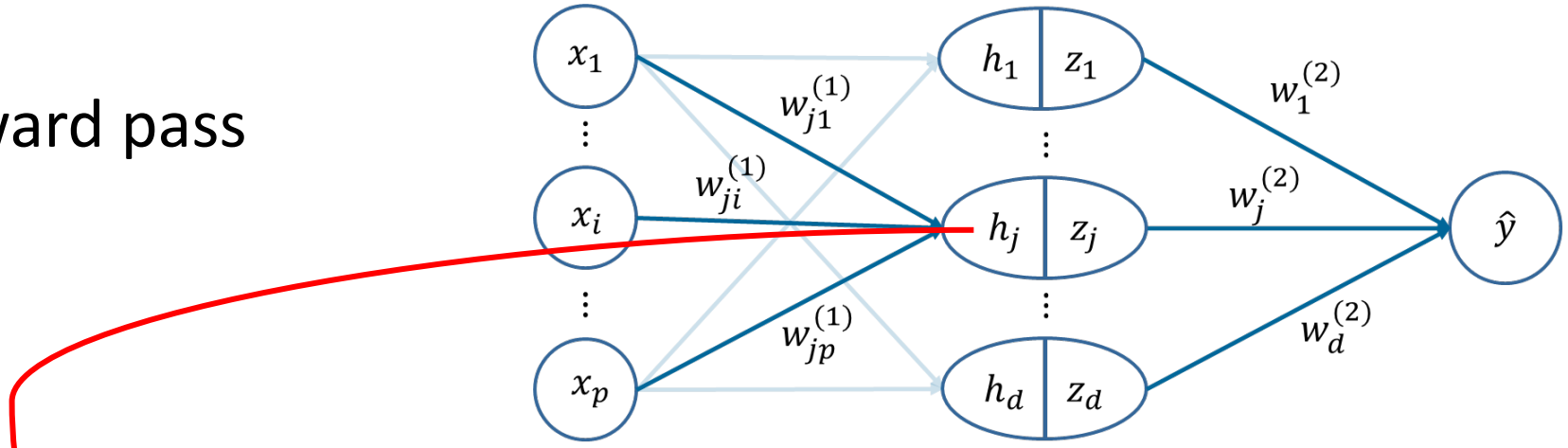
Training of Multi-layer Perceptron

- Consider an MLP for regression and a sigmoid activation function for hidden nodes.



Training of Multi-layer Perceptron

- Forward pass

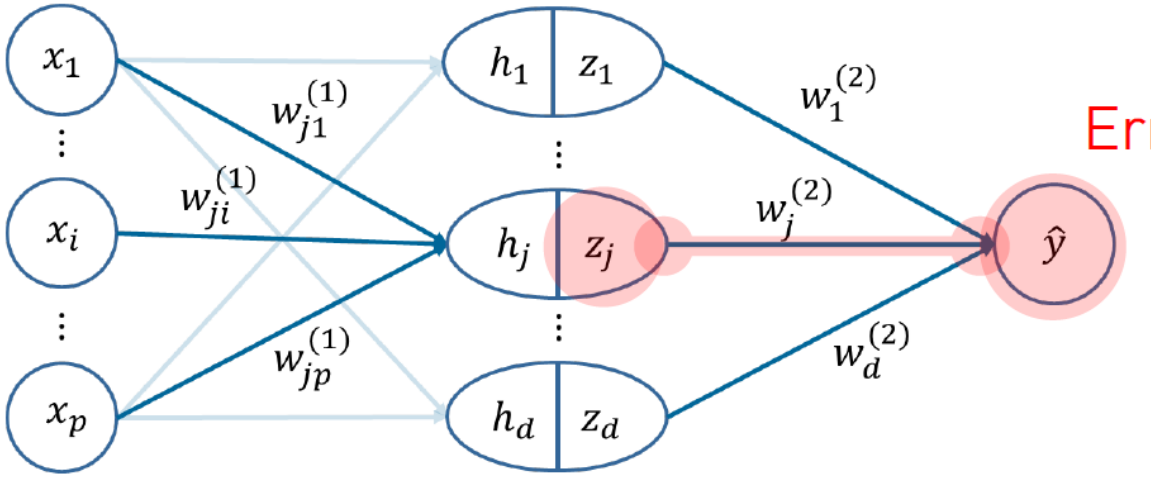


$$h_j = \sum_{i=1}^{p} w_{ji}^{(1)} x_i \quad \Rightarrow \quad z_j = \sigma(h_j) = \frac{1}{1 + \exp(-h_j)}$$

$$\Rightarrow \quad \hat{y} = \sum_{j=1}^{d} w_j^{(2)} z_j = \sum_{j=1}^{d} w_j^{(2)} \sigma\left(\sum_{i=1}^{p} w_{ji}^{(1)} x_i \right)$$

Training of Multi-layer Perceptron

- Given $(\boldsymbol{x}, y)$, compute the gradient. First, compute the derivative of L with respect to $w_j^{(2)}$.



Error $(\hat{y} - y)$

$$L = \frac{1}{2}(y - \hat{y})^2$$

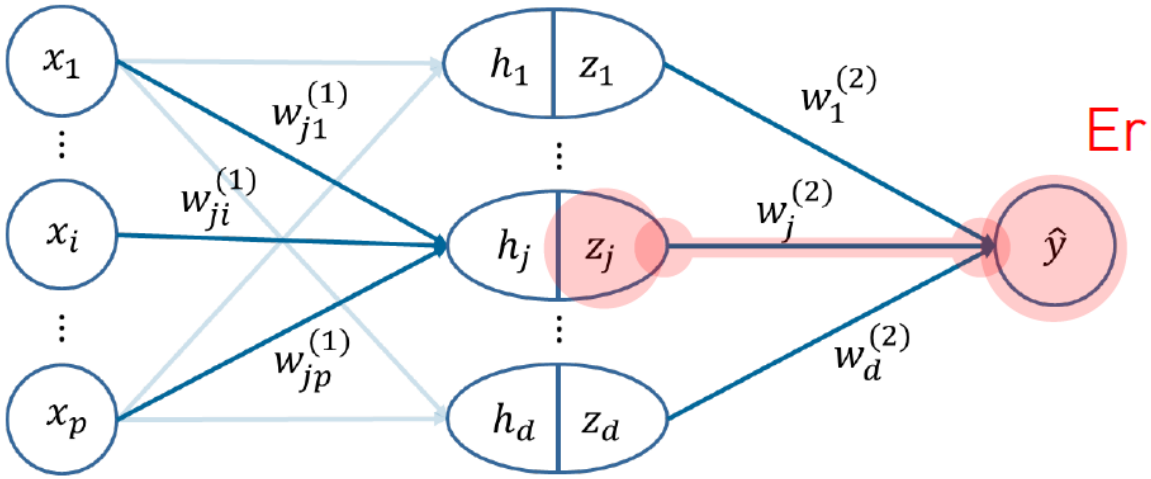
where

$$\hat{y} = \sum_{j=1}^{d} w_j^{(2)} z_j = \sum_{j=1}^{d} w_j^{(2)} \sigma\left(\sum_{i=1}^{p} w_{ji}^{(1)} x_i\right)$$

$$\frac{\partial L}{\partial w_j^{(2)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_j^{(2)}} = (\hat{y} - y) z_j$$

Training of Multi-layer Perceptron

- Then, compute the derivative of L with respect to $w_{ji}^{(1)}$.



Error $(\hat{y} - y)$

$$z = \frac{1}{1+e^{-h_j}}$$

$$\frac{d}{dh_j}\left(1+e^{-h_j}\right)^{-1} = -1 \cdot \left(1+e^{-h_j}\right)^{-2}\left(+e^{-h_j}\right)$$

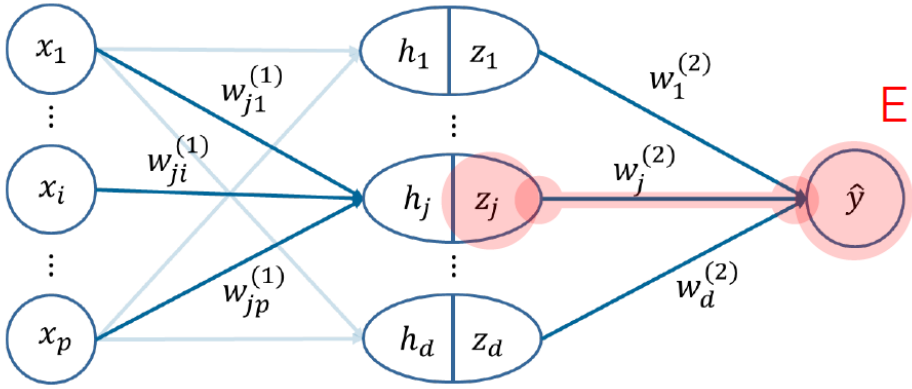
$$= \frac{1}{(1+e^{-h_j})} \cdot \frac{e^{-h_j}}{(1+e^{-h_j})}$$

$$(z)\ (1-z)$$

$$\frac{\partial L}{\partial w_{ji}^{(1)}} = \frac{\partial L}{\partial \hat{y}}\left(\frac{\partial \hat{y}}{\partial z_j}\right)\frac{\partial z_j}{\partial h_j}\cdot\frac{\partial h_j}{\partial w_{ji}^{(1)}} = (\hat{y}-y)\left(w_j^{(2)}\right)z_j(1-z_j)x_i$$

Training of Multi-layer Perceptron

- Backpropagating the error

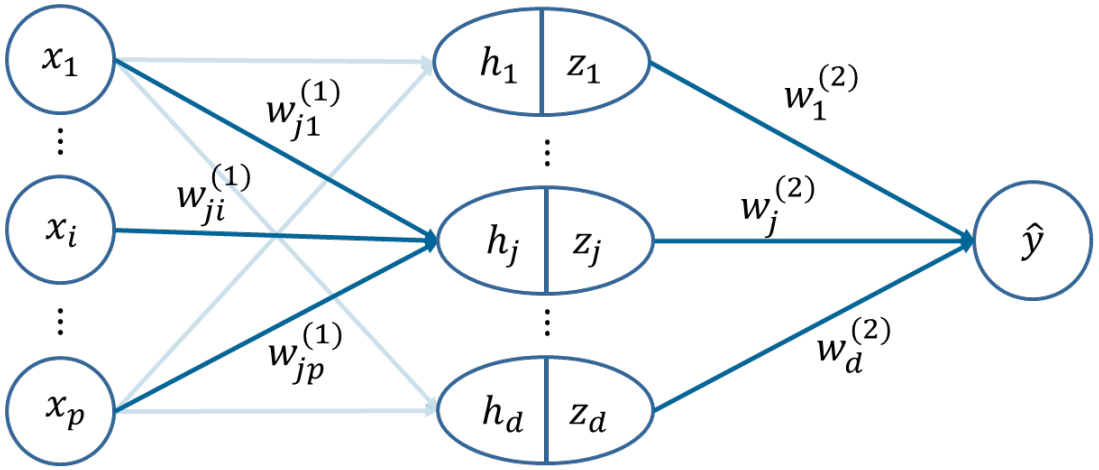


$$\frac{\partial L}{\partial w_j^{(2)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_j^{(2)}} = (\hat{y} - y) z_j$$

$$\frac{\partial L}{\partial w_{ji}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_j} \cdot \frac{\partial z_j}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{ji}^{(1)}} = (\hat{y} - y) w_j^{(2)} z_j (1 - z_j) x_i$$

Training of Multi-layer Perceptron

- Given $\{\boldsymbol{x}_k, y_k\}, k = 1, \ldots, n$, compute the gradient.



$$L = \frac{1}{n}\sum_{k=1}^{n} L_k$$

$$= \frac{1}{2n}\sum_{k=1}^{n} (y_k - \hat{y}_k)^2$$

$$\frac{\partial L}{\partial w_j^{(2)}} = \frac{1}{n}\sum_{k=1}^{n} \frac{\partial L_k}{\partial w_j^{(2)}} = \frac{1}{n}\sum_{k=1}^{n} (y_k - \hat{y}_k) z_{kj}$$

$$\frac{\partial L}{\partial w_{ji}^{(1)}} = \frac{1}{n}\sum_{k=1}^{n} \frac{\partial L_k}{\partial w_{ji}^{(1)}} = \frac{1}{n}\sum_{k=1}^{n} (y_k - \hat{y}_k) w_j^{(2)} z_{kj}(1 - z_{kj}) x_{ki}$$

Summary

- Multi-layer perceptron = a stack of multiple linear layers interleaved with non-linear activation functions
- Training of neural networks = computing gradients via the chain rule (backpropagation) and updating model parameters accordingly
- Reading assignments
  - Dive into deep learning: Chapter 5.1, 5.2, 5.3
  - Understanding deep learning: Chapter 4, 6.1, 6.2, 7.1-7.4