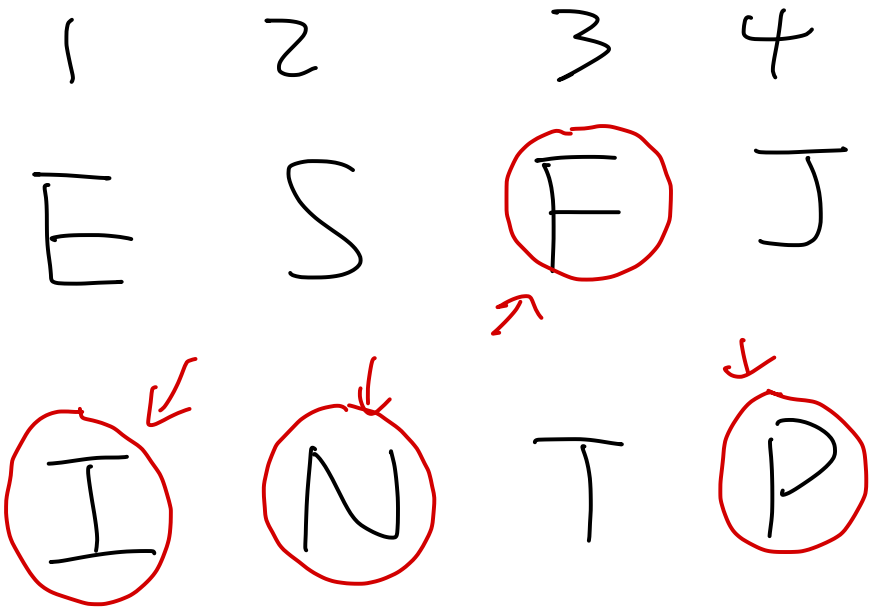
Inclass 22: Neural Network and Backpropagation

[SCS4049] Machine Learning and Data Science

Seongsik Park (s.park@dgu.edu)

School of AI Convergence & Department of Artificial Intelligence, Dongguk University



장유림,
이종우,
1만5천1백원.

Neural Networks and Perceptron

Biological neuron

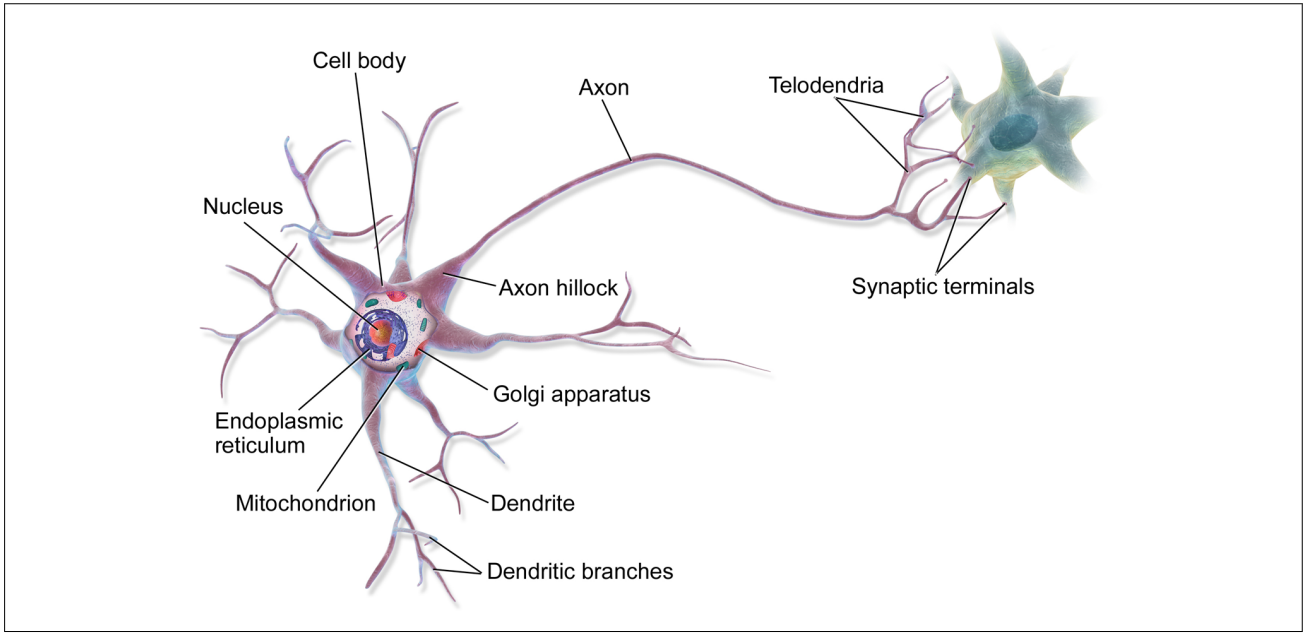


Figure 10-1. Biological neuron[3]

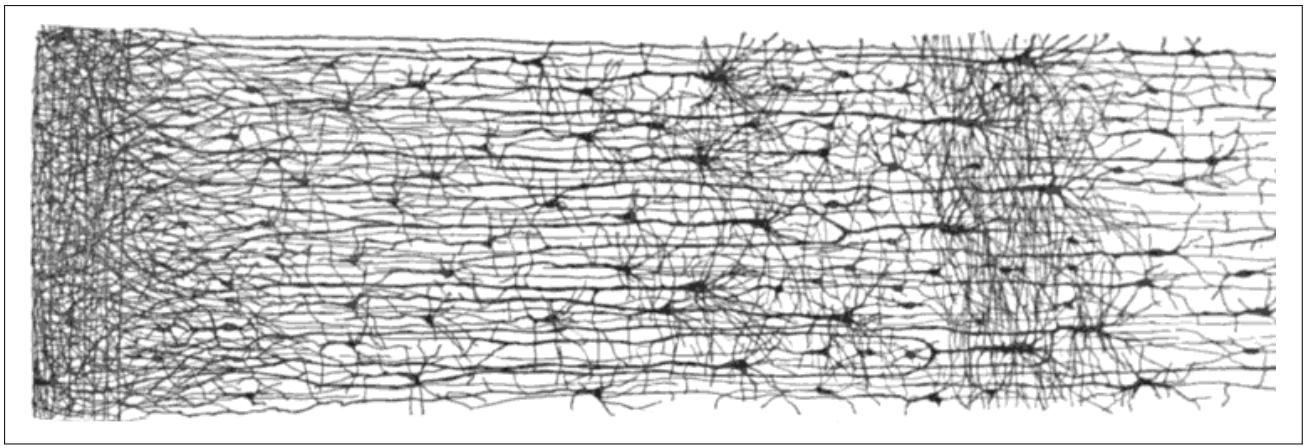


Figure 10-2. Multiple layers in a biological neural network (human cortex)[5]

Computing with the brain

An engineering perspective

- Compact
- Energy efficient (20 watts)
- 85 billion Glial cells (power, cooling, support)
- 86 billion Neurons (soma + wires)
- 69 billion Cerebellum neurons (soma + wires)
- 103 104 Connections (synapses) per neuron
- Volume = mostly wires

General computing machine?

- Slow for mathematical logic, arithmetic, etc
- Very fast for vision, speech, language, social interactions, etc
- Evolution: vision $\Longrightarrow$ language $\Longrightarrow$ logic

Artificial neural network

Birds inspired us to fly, burdock plants inspired velcro, and countless more inventions were inspired by nature. It seems only logical, then, to look at the brain's architecture for inspiration on how to build an intelligent machine. This is the key idea that sparked artificial neural networks (ANNs). However, although planes were inspired by birds, they don't have to flap their wings. Similarly, ANNs have gradually become quite different from their biological cousins. Some researchers even argue that we should drop the biological analogy altogether (e.g., by saying "units" rather than "neurons"), lest we restrict our creativity to biologically plausible systems.

Artificial neural network

ANNs are at the very core of Deep Learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex Machine Learning tasks, such as classifying billions of images (e.g., Google Images), powering speech recognition services (e.g., Apple's Siri), recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube), or learning to beat the world champion at the game of Go by playing millions of games against itself (DeepMind's Alpha-Zero).

Threshold logic unit

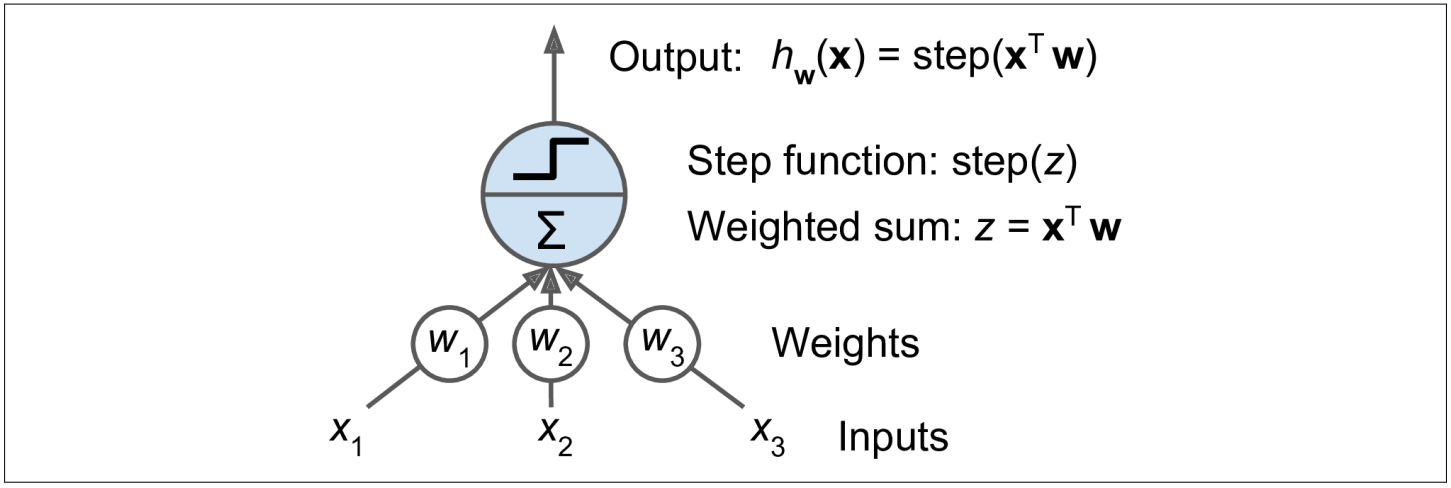


Figure 10-4. Threshold logic unit

Perceptron $\Longrightarrow$ binary classifier

$$\mathbf{z} = W_1 X_1 + W_2 X_2 + \cdots + X_m X_m = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_m \end{bmatrix}^T \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} = \mathbf{w}^T \mathbf{x} \qquad (1)$$

Threshold logic unit (TLU) or linear threshold unit (LTU)

$$h_{\mathbf{w}}(\mathbf{x}) = \text{step}(z) = \text{step}(\mathbf{w}^T \mathbf{x}) \qquad (2)$$

Threshold logic unit

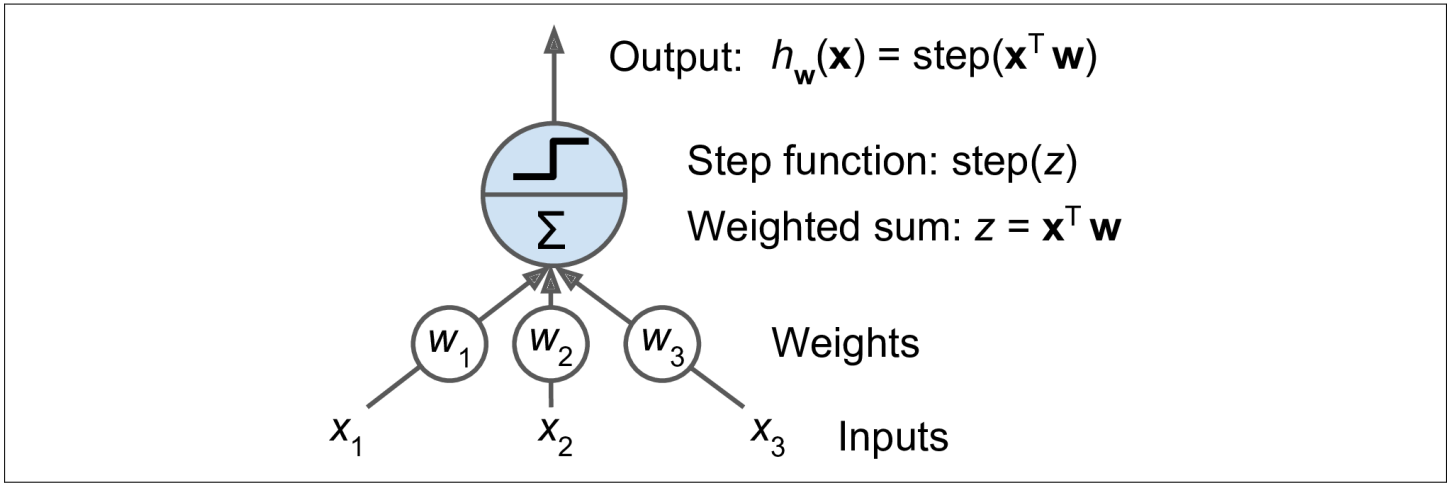


Figure 10-4. Threshold logic unit

Common step functions used in Perceptrons

$$\text{heavyside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{otherwise} \end{cases} \qquad \text{sign}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{otherwise} \end{cases} \tag{3}$$

Perceptron algorithm

Computing the outputs of a fully connected layer

$$h_{\mathbf{W},\mathbf{b}} = \phi(\mathbf{XW} + \mathbf{b}) \tag{4}$$

Perceptron learning rule (weight update)

$$w_{i,j} \leftarrow w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{5}$$

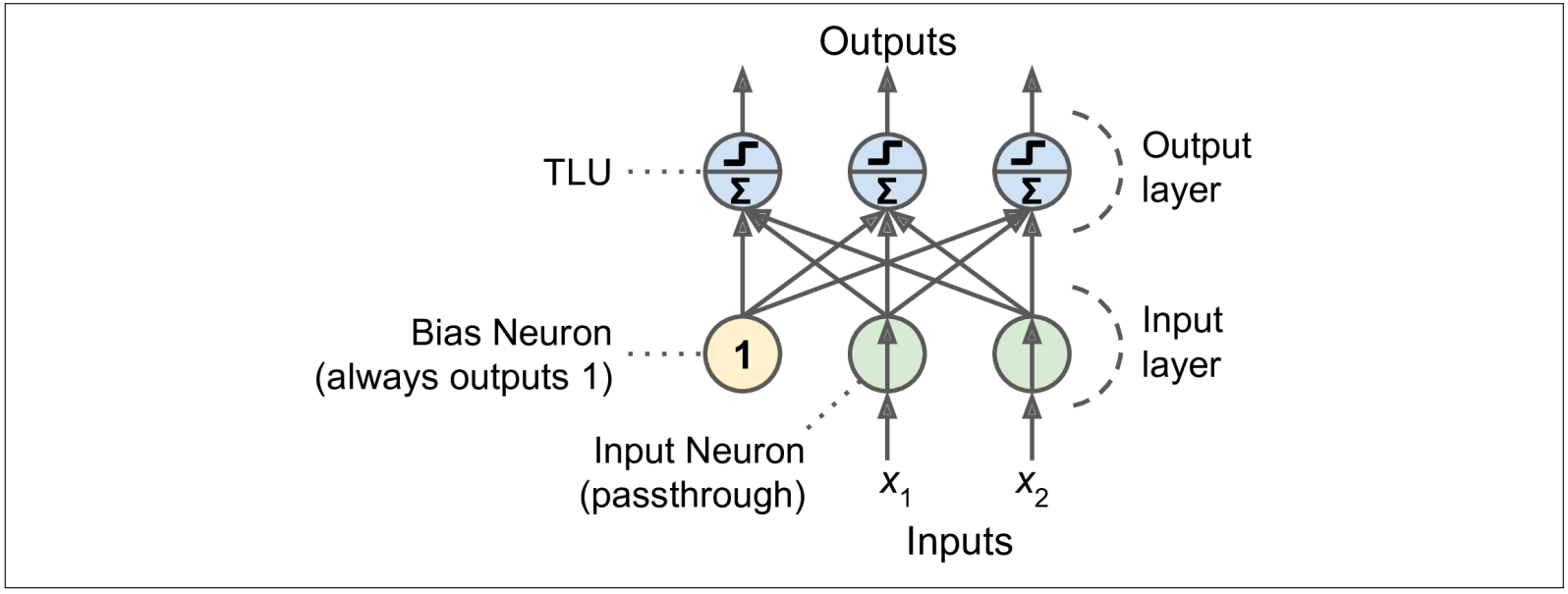


Figure 10-5. Perceptron diagram

Deep Feedforward Networks

Multi-layer perceptron algorithm

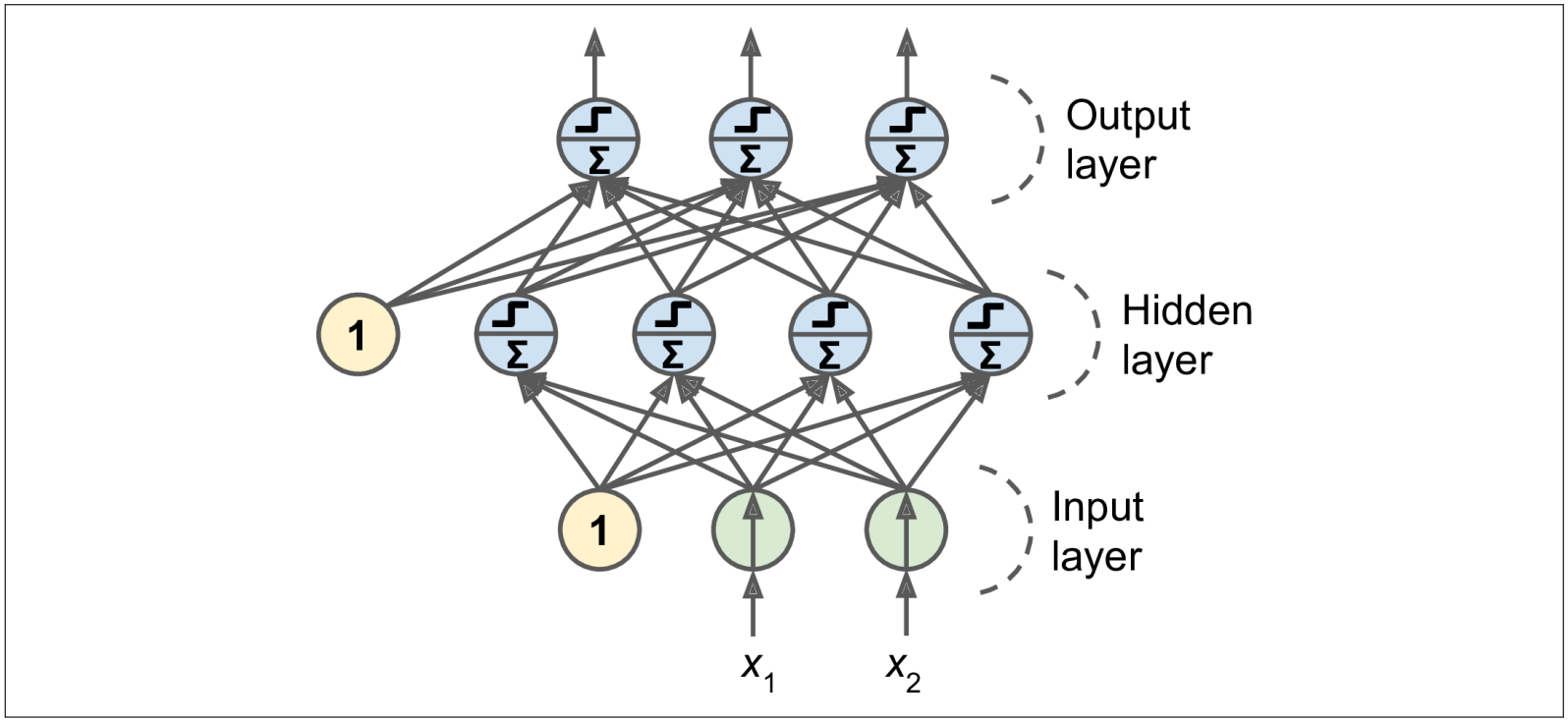


Figure 10-7. Multi-Layer Perceptron

Activation functions and their derivatives

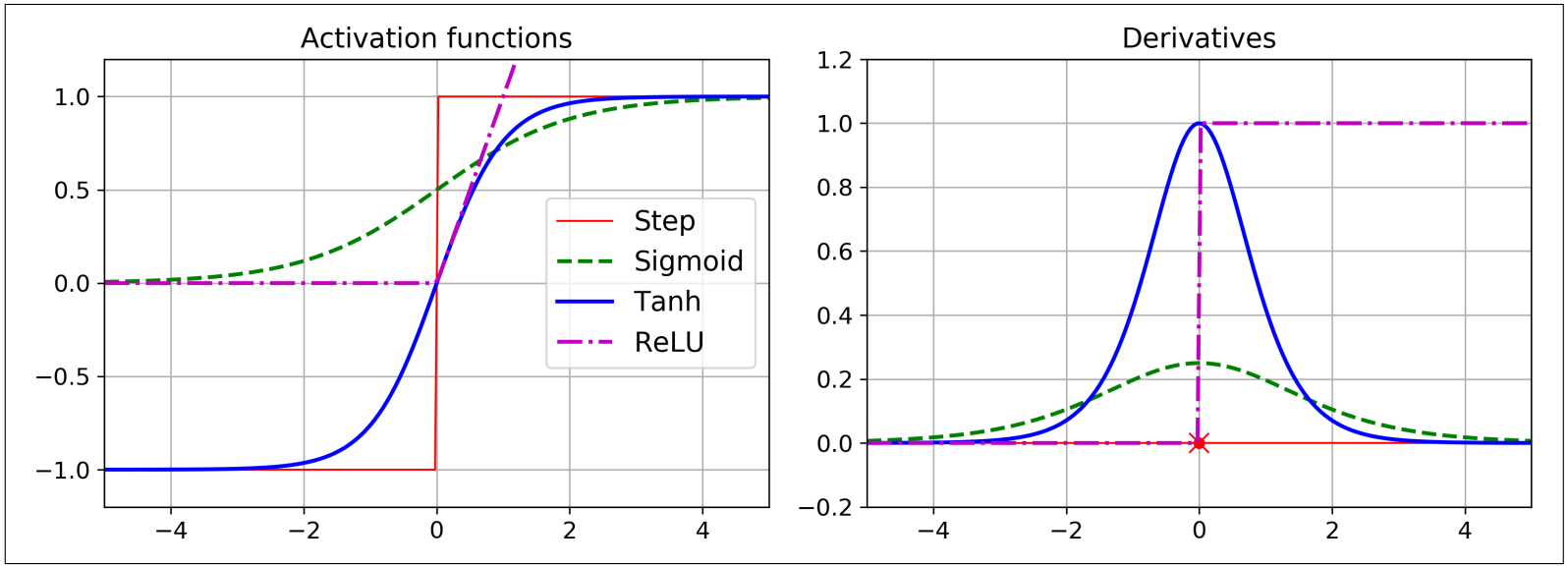


Figure 10-8. Activation functions and their derivatives

A modern MLP (including ReLU and softmax) for classification

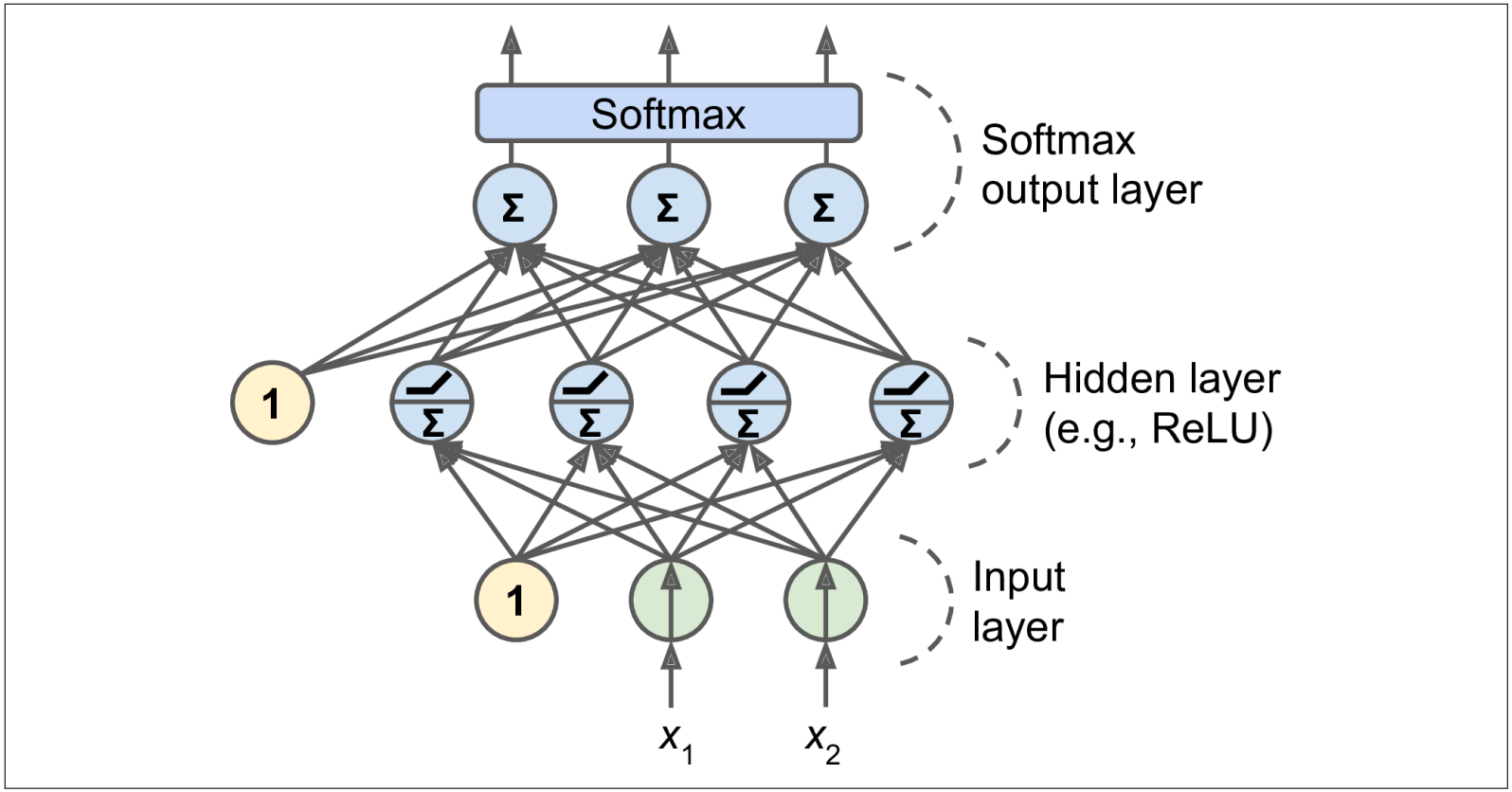


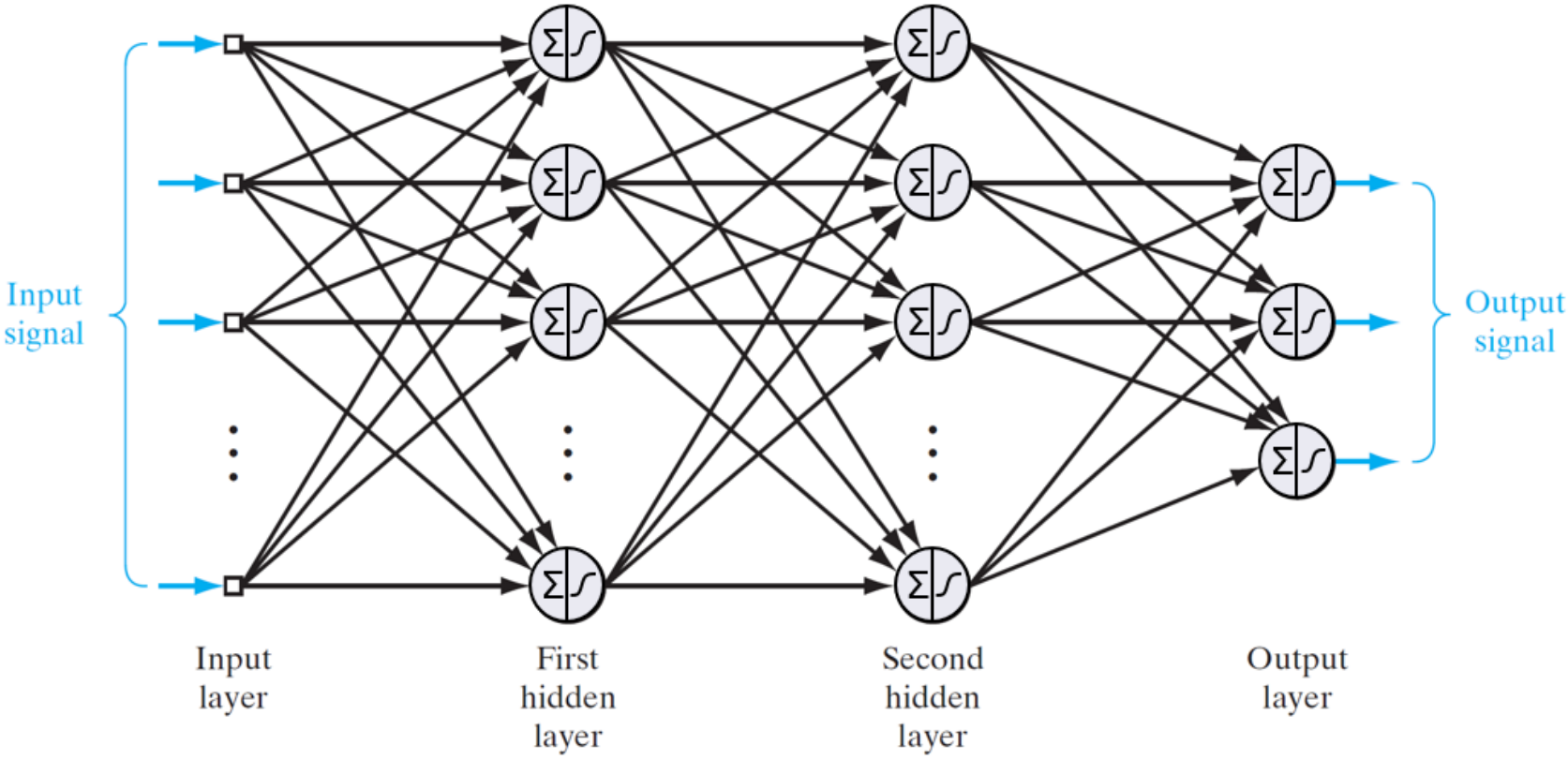
Figure 10-9. A modern MLP (including ReLU and softmax) for classification

Deep feedforward networks

Deep Feedforward Networks, Multi-Layer Perceptron, or Feedforward Neural Networks

- Fully Connected Multi-layer
- Feedforward $\Longrightarrow$ Feedback이 없음
  (Feedback 이 있는 신경망: Recurrent Neural Networks)
- Nonlinear Activation Functions
- Rumelhart (1986) 의 Backpropagation 알고리즘 이후 집중적 연구

Deep feedforward networks



Nonlinear activation functions

Logistic sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{7}$$

Hyperbolic tangent function

$$\varphi(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{8}$$

$$\varphi'(z) = 2\varphi(2z) - 1 \tag{9}$$

Rectified linear unit function

$$\mathrm{ReLU}(z) = \max(0, z) \tag{10}$$

$$\mathrm{ReLU}'(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{11}$$

Nonlinear activation functions

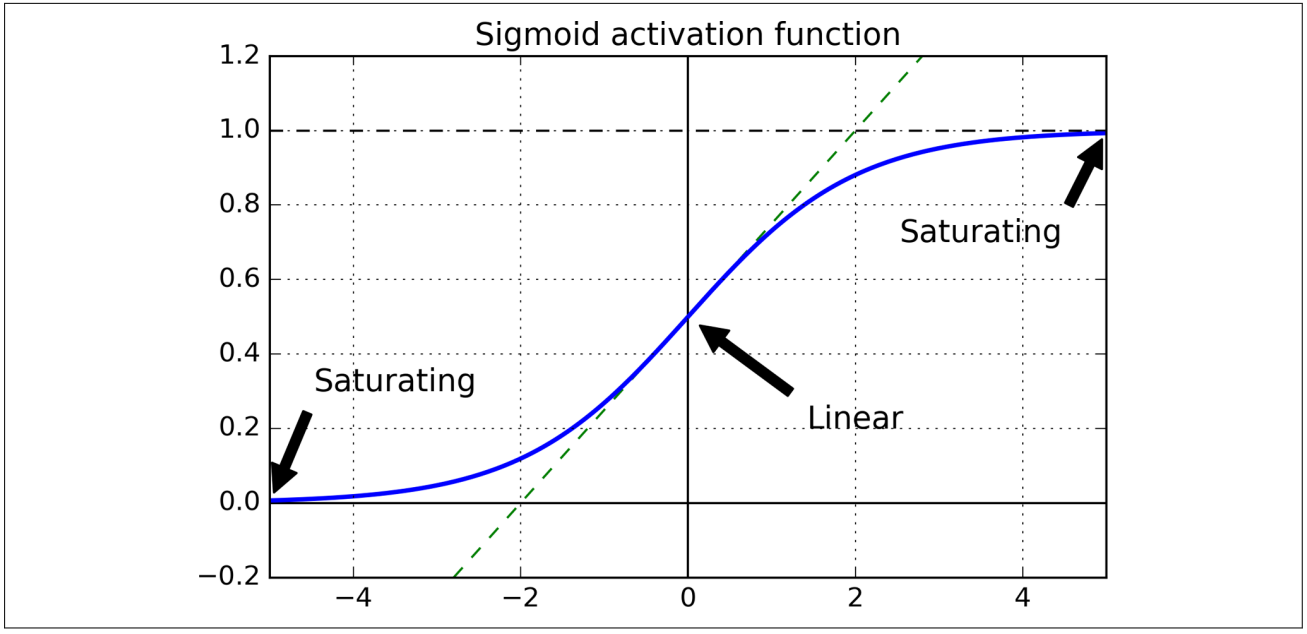


Figure 11-1. Logistic activation function saturation

Nonlinear activation functions

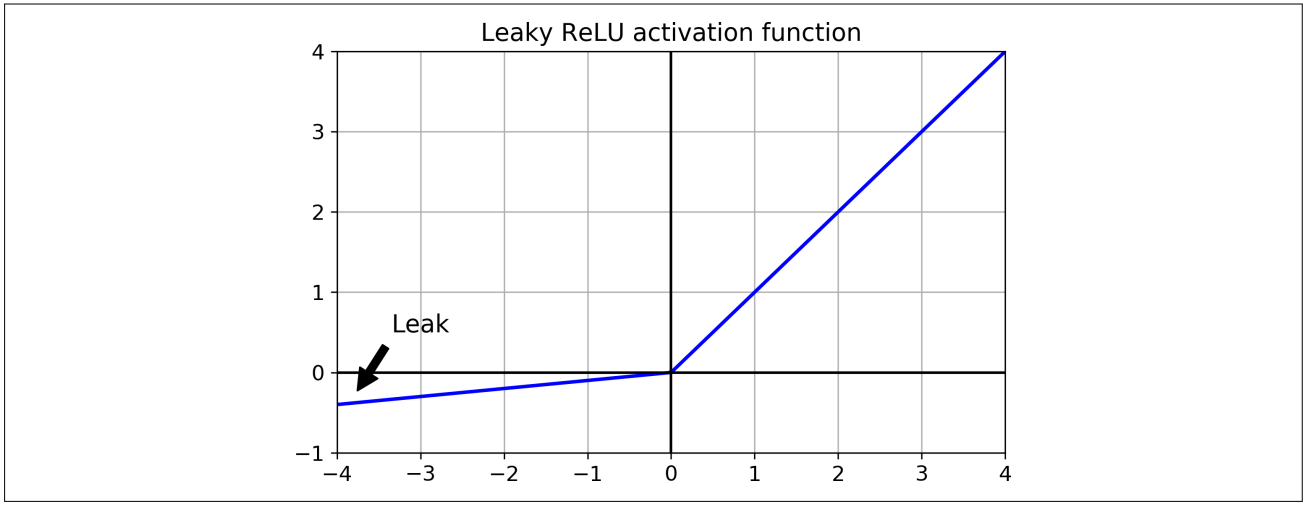
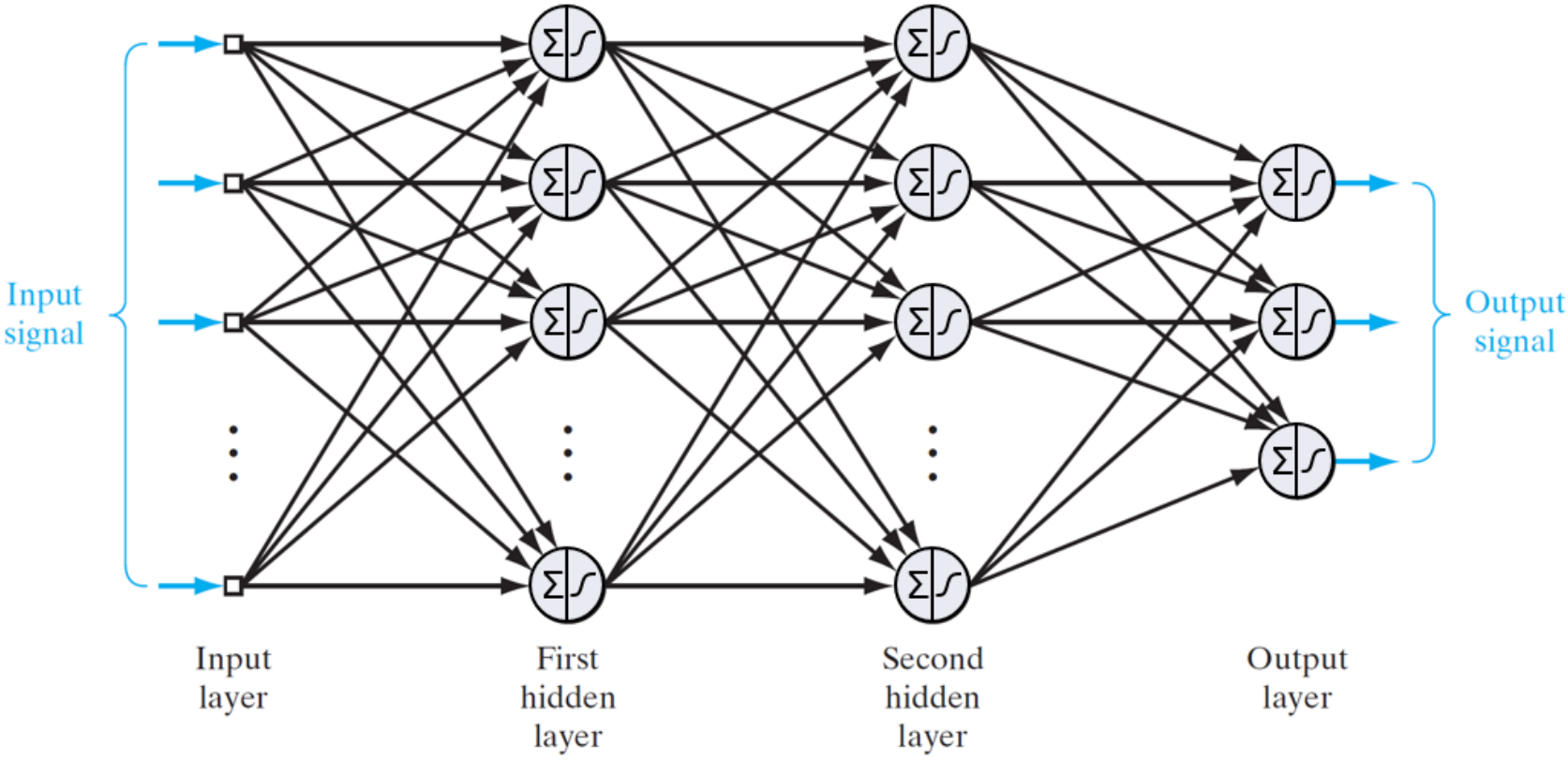


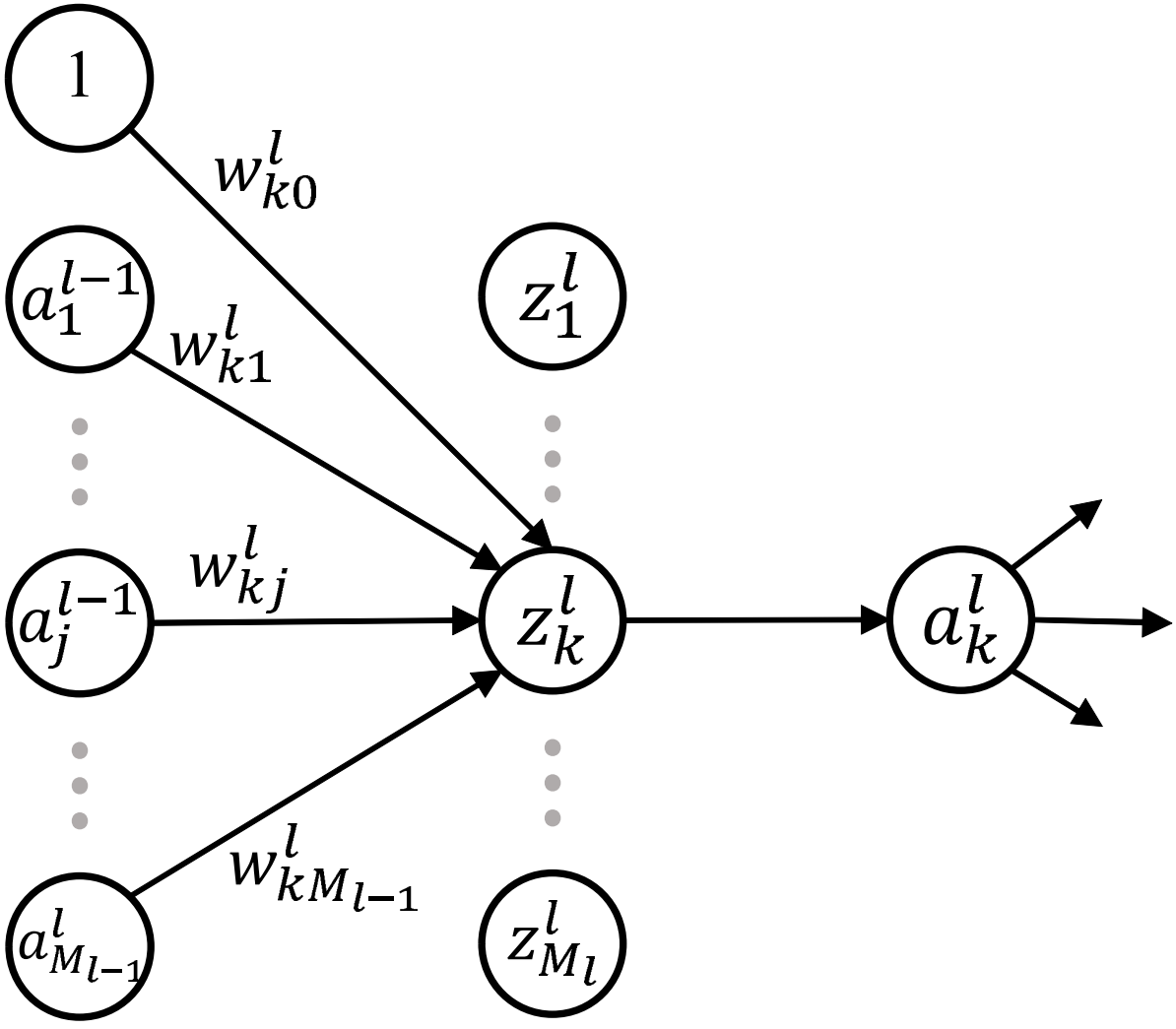
Figure 11-2. Leaky ReLU

Forward propagation

Deep feedforward networks



Feedforward propagation



Feedforward propagation

For l-th hidden layer

$$z_k^l = \sum_{j=0}^{M_{l-1}} w_{kj}^l a_j^{l-1} \qquad a_k^l = \sigma^l(z_k^l) \tag{12}$$

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} \qquad \mathbf{a}^l = \boldsymbol{\sigma}^l(\mathbf{z}^l) \tag{13}$$

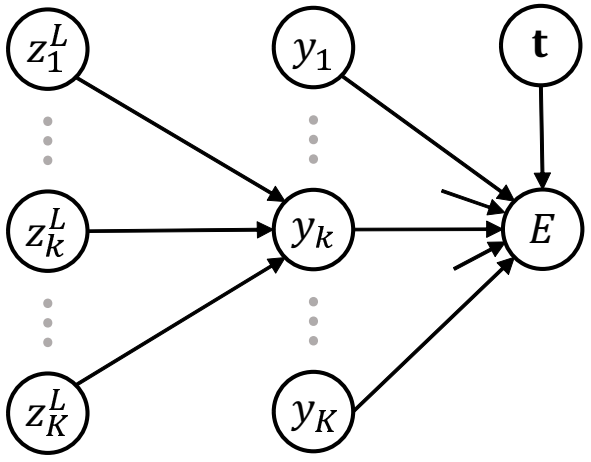
- z_k^l: layer l의 unit k로 들어오는 activation들의 weighted sum
- w_{kj}^l: layer $(l-1)$의 unit j에서 layer l의 unit k로 가는 가중치
- w_{k0}^l: layer l의 unit k에 대한 bias
- a_j^{l-1}: layer $(l-1)$의 unit j에서 나오는 activation 값
- a_0^l: layer l의 bias에 대응 $(=1)$
- a_k^l: layer l의 unit k의 activation 값
- σ^l: layer l의 activation function
- M_{l-1}: layer $(l-1)$의 hidden unit 수

Cost function

Cost function: cross-entropy

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{M_L} t_j \log y_j \tag{14}$$

$$y_k = \frac{\exp(z_k)}{\sum_{j=1}^{K}\exp(z_j)} \tag{15}$$



- t_j: $\mathbf{x}$의 class label (target value)
- K: 출력층의 unit 수 = class의 수, $M_L = K$

Stochastic gradient and backpropagation

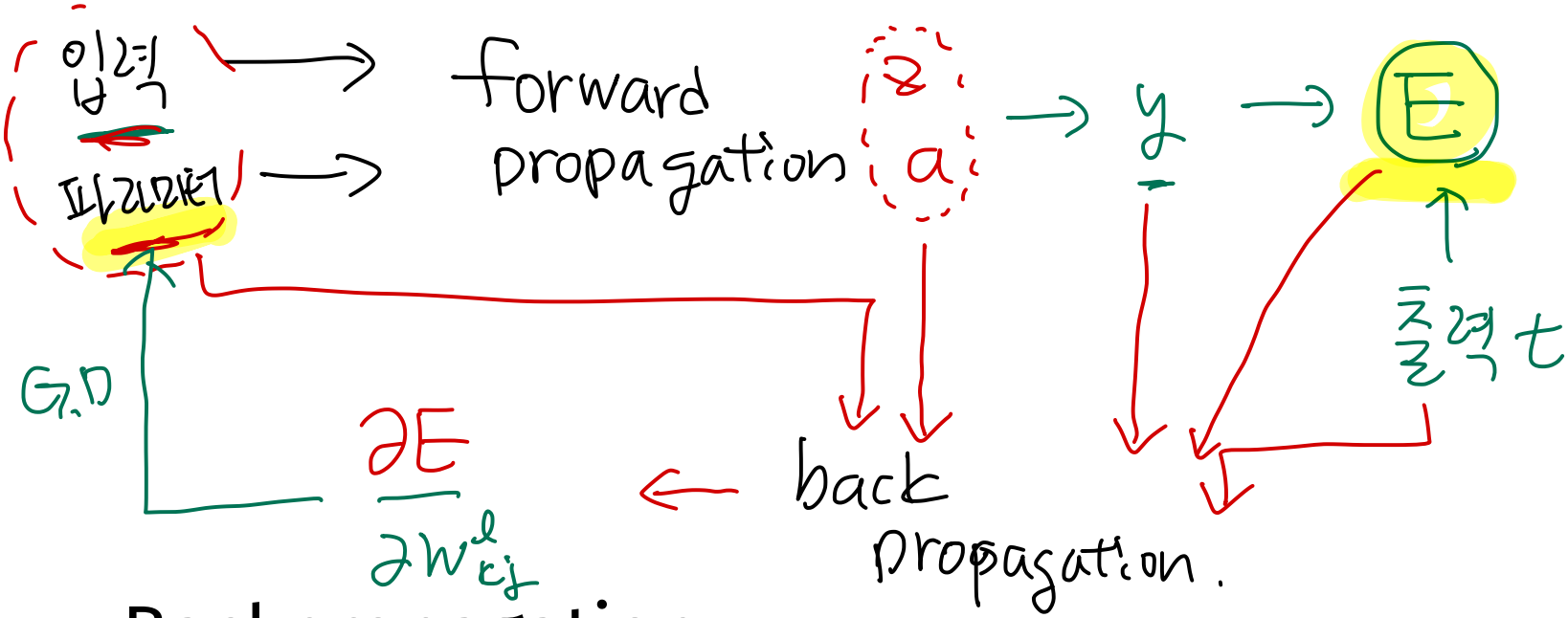
SGD weight update rule

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{16}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} E \tag{17}$$

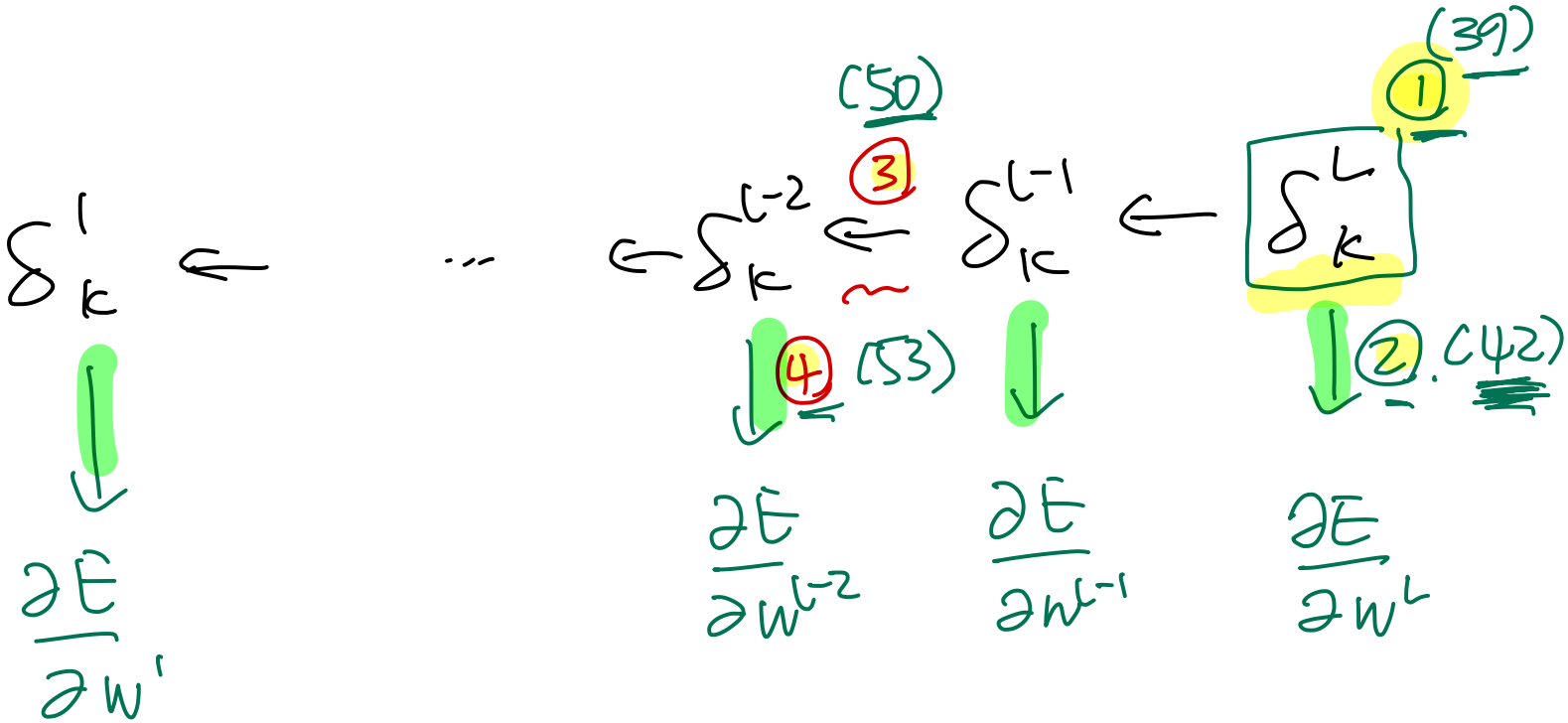
Credit assignment

- Weight 값들이 잘못 되었기 때문에 Cost 가 발생
  $\Longrightarrow$ 따라서 모든 weight에게 그 weight가 cost 발생에 기여한 만큼씩 gradient 를 배분해 주는 것
- 문제는 weight 들이 여러 층에 분산되어 있으며, 각 층은 비선형 함수이므로 배분 방법이 단순하지 않음 $\Longrightarrow$ backpropagation algorithm



Backpropagation

local gradient. $\frac{\partial E}{\partial z_k^l} = \delta_k^l$



$$w^l \leftarrow w^l - \eta \frac{\partial E}{\partial w^l}$$

Backpropagation

출력층부터 입력층까지 역순으로 모든 layer에서 weight w_{ij}^l을 업데이트

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{18}$$

Local gradient (or error of neuron)의 정의

$$\delta_j^l \triangleq \frac{\partial E}{\partial z_j^l} \tag{19}$$

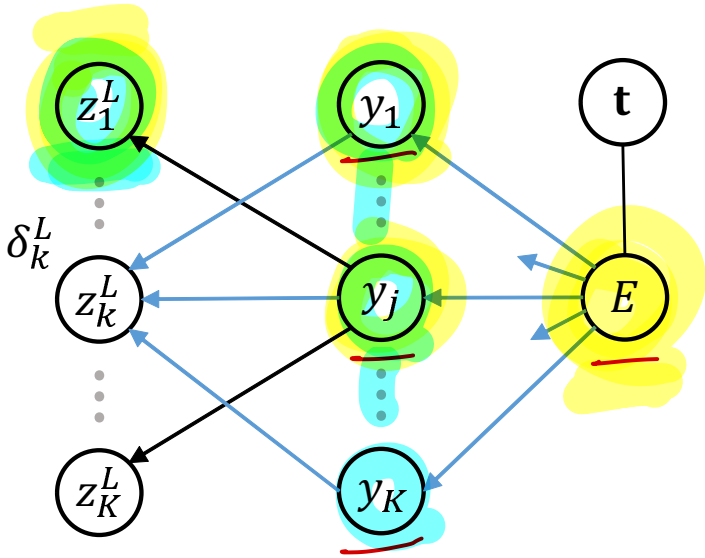
- Layer l의 unit k에서 error, 또는 하위층으로 분배할 gradient
- Activation a_j^l를 기준으로 생각할 수 있으나, nonlinear 함수를 포함하므로 다루기 어려움

Backpropagation: output layer 1/7

출력층에서의 local gradient δ_k^L 계산

$$\delta_k^L = \frac{\partial E}{\partial z_k^L} = \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_k^L} \tag{20}$$

- Softmax 함수는 $z_1^L, z_2^L, \ldots, z_K^L$에 의존
- 따라서 z_K^L의 변동은 $y_1, y_2, \ldots, y_K$에 모두 영향



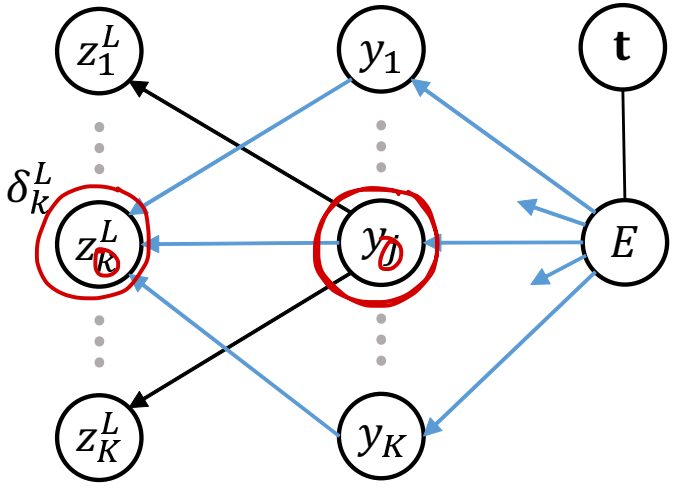
Backpropagation: output layer 2/7

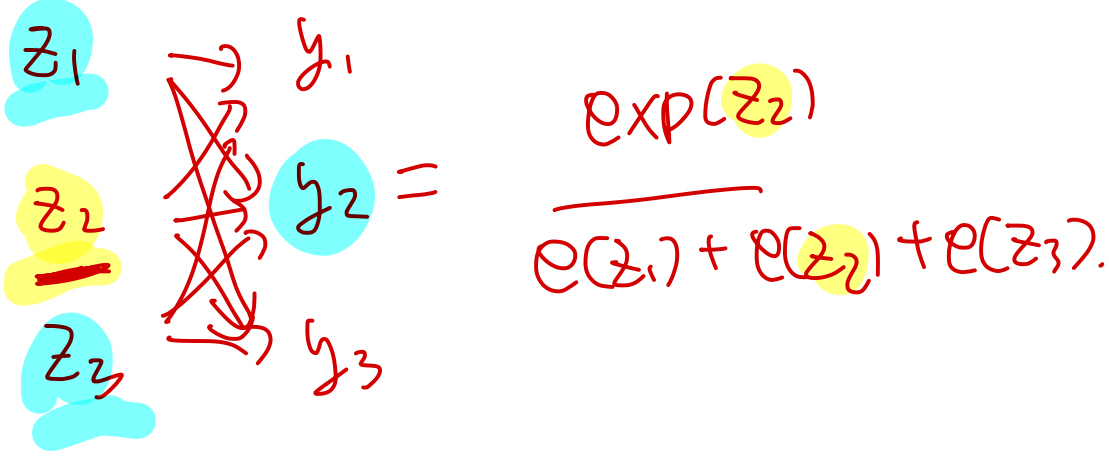
$\frac{\partial y_j}{\partial z_k^L}$ 의 계산: $k = j$인 경우

$$\frac{\partial y_k}{\partial z_k^L} = \frac{\partial}{\partial z_k^L} \frac{\exp(z_k)}{\sum_i \exp(z_i)} \tag{21}$$

$$= \frac{\exp(z_k)}{\sum_i \exp(z_i)} \left(1 - \frac{\exp(z_k)}{\sum_i \exp(z_i)}\right) \tag{22}$$

$$= y_k(1 - y_k) \tag{23}$$





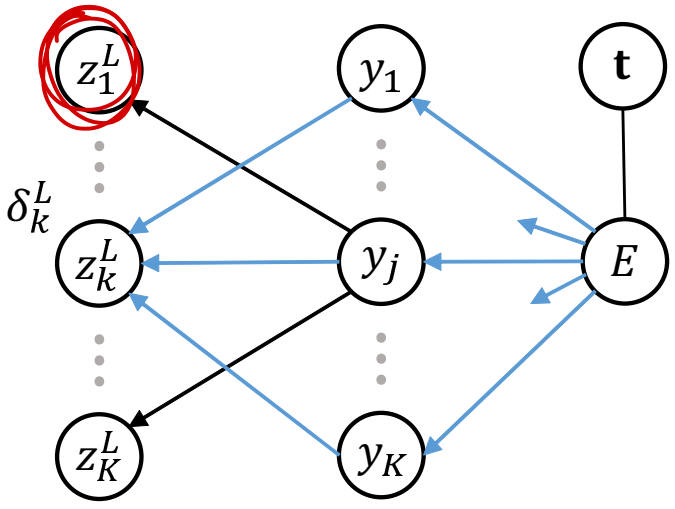
Backpropagation: output layer 3/7

$\frac{\partial y_j}{\partial z_k^L}$ 의 계산: $k \neq j$인 경우

$$\frac{\partial y_j}{\partial z_k^L} = \frac{\partial}{\partial z_k^L} \frac{\exp(z_j)}{\sum_i \exp(z_i)} \tag{24}$$

$$= -\frac{\exp(z_k)}{\sum_i \exp(z_i)} \frac{\exp(z_j)}{\sum_i \exp(z_i)} \tag{25}$$

$$= -y_k y_j \tag{26}$$



Backpropagation: output layer 4/7

From definition of local gradient

$$\delta_k^L = \frac{\partial E}{\partial z_k^L} = -\sum_j t_j \frac{\partial}{\partial z_k} \log y_j = -\sum_j t_j \frac{1}{y_j} \frac{\partial y_j}{\partial z_k} \tag{27}$$

$$= -\frac{t_k}{y_k} \frac{\partial y_k}{\partial z_k} - \sum_{j \neq k} \frac{t_j}{y_j} \frac{\partial y_j}{\partial z_k} \tag{28}$$

$$= -\frac{t_k}{y_k} y_k (1 - y_k) - \sum_{j \neq k} \frac{t_j}{y_j} (-y_k y_j) \tag{29}$$

$$= -t_k + t_k y_k + \sum_{j \neq k} t_j y_k \tag{30}$$

$$= -t_k + \sum_{j=1}^{K} t_j y_k \tag{31}$$

$$= -(t_k - y_k) \tag{32}$$

$y_k(1-y_k)$

$-y_j y_k$

$= -$ (정답값 − 예측값)

Backpropagation: output layer 5/7

In matrix form

$$\boldsymbol{\delta}^L = \frac{\partial E}{\partial \mathbf{z}^L} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{z}^L}\right)^T \left(\frac{\partial E}{\partial \mathbf{y}}\right) \tag{33}$$

$$= \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \cdots & \frac{\partial y_j}{\partial z_1} & \cdots & \frac{\partial y_K}{\partial z_1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial z_k} & \cdots & \frac{\partial y_j}{\partial z_k} & \cdots & \frac{\partial y_K}{\partial z_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial z_K} & \cdots & \frac{\partial y_j}{\partial z_K} & \cdots & \frac{\partial y_K}{\partial z_K} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \vdots \\ \frac{\partial E}{\partial y_j} \\ \vdots \\ \frac{\partial E}{\partial y_K} \end{bmatrix} \tag{34}$$

$$= \begin{bmatrix} \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_1} \\ \vdots \\ \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_k} \\ \vdots \\ \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_K} \end{bmatrix} = \begin{bmatrix} -(t_1 - y_1) \\ \vdots \\ -(t_k - y_k) \\ \vdots \\ -(t_K - y_K) \end{bmatrix} \tag{35}$$

Backpropagation: output layer 6/7

With vector form

$$\mathbf{y} = \boldsymbol{\varphi}(\mathbf{z}^L) \tag{36}$$

$$\boldsymbol{\delta}^L = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{z}^L}\right)^T \left(\frac{\partial E}{\partial \mathbf{y}}\right) \tag{37}$$

$$= \left(\boldsymbol{\varphi}'(\mathbf{z}^L)\right)^T \nabla_{\mathbf{y}} E \tag{38}$$

$$= -(\mathbf{t} - \mathbf{y}) \tag{39}$$

Backpropagation: output layer 7/7

$\frac{\partial E}{\partial w_{kj}^L}$ 의 계산

$$z_k^L = \sum_{j=0}^{M_{L-1}} w_{kj}^L a_j^{L-1} \tag{40}$$

$$\frac{\partial z_k^L}{\partial w_{kj}^L} = a_j^{L-1} \tag{41}$$

$$\frac{\partial E}{\partial w_{kj}^L} = \frac{\partial E}{\partial z_k^L}\frac{\partial z_k^L}{\partial w_{kj}^L} = -(t_k - y_k)a_j^{L-1} = \delta_k^L a_j^{L-1} \tag{42}$$

그러므로 weight update rule은

$$w_{kj}^L \leftarrow w_{kj}^L - \eta \frac{\partial E}{\partial w_{kj}^L} \tag{43}$$

$$= w_{kj}^L - \eta \delta_k^L a_j^{L-1} \tag{44}$$

$$= w_{kj}^L + \eta(t_k - y_k)a_j^{L-1} \tag{45}$$

(46)

Backpropagation: local gradient propagation

From δ_j^{l+1} to δ_j^l

$$\delta_k^l = \frac{\partial E}{\partial z_k^l} = \sum_j \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_k^l} = \sum_j \frac{\partial z_j^{l+1}}{\partial z_k^l} \delta_j^{l+1} \tag{47}$$

$$z_j^{l+1} = \sum_{i=0} w_{ji}^{l+1} a_i^l = \sum_{i=0} w_{ji}^{l+1} \sigma^l(z_i^l) \tag{48}$$

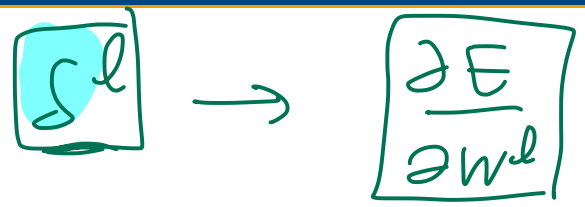
$$\frac{\partial z_j^{l+1}}{\partial z_k^l} = w_{jk}^{l+1} \sigma^{l'}(z_k^l) \tag{49}$$

그러므로

$$\delta_k^l = \sigma^{l'}(z_k^l) \sum_j w_{jk}^{l+1} \delta_{jk}^{l+1} \tag{50}$$

Backpropagation: weight in layer l

$\frac{\partial E}{\partial w_{kj}^l}$ 의 계산



$$z_k^l = \sum_{j=0}^{M_{l-1}} w_{kj}^l a_j^{l-1} \tag{51}$$

$$\frac{\partial z_k^l}{\partial w_{kj}^l} = a_j^{l-1} \tag{52}$$

$$\frac{\partial E}{\partial w_{kj}^l} = \frac{\partial E}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{kj}^l} = \delta_k^l a_j^{l-1} \tag{53}$$

그러므로 weight update rule은

$$w_{kj}^l \leftarrow w_{kj}^l - \eta \frac{\partial E}{\partial w_{kj}^l} \tag{54}$$

$$= w_{kj}^l - \eta \delta_k^l a_j^{l-1} \tag{55}$$

Backpropagation: summary

1. Forward propagation: 각 층 $l = 2, 3, \ldots, L$에 대해서 다음을 계산

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} \quad \mathbf{a}^l = \boldsymbol{\sigma}^l(\mathbf{z}^l) \qquad (56)$$

2. Output error $\boldsymbol{\delta}^L$: 출력층에서 비용함수를 이용해 local gradient 계산

$$\boldsymbol{\delta}^L = -(\mathbf{t} - \mathbf{y}) \qquad (57)$$

3. Backpropagate the error: 출력층부터 거꾸로 다음 식을 계산

$$\delta_k^l = \sigma^{l\prime}(z_k^l) \sum_j w_{jk}^{l+1} \delta_{jk}^{l+1} \qquad (58)$$

4. Output the gradient: 각 weight에 대해 gradient 계산

$$\frac{\partial E}{\partial w_{kj}^l} = \delta_k^l a_j^{l-1} \qquad (59)$$

5. Update weight:

$$w_{kj}^l \leftarrow w_{kj}^l - \eta \delta_k^l a_j^{l-1} \qquad (60)$$

Cov $C = C^T$

$U\Sigma V^T = V\Sigma^T U^T$

matrix, Symmetric $\Rightarrow$ $U\Sigma V^T$, $U = V$.

Backpropagation: summary