머신런닝 개요

Lecture 6: Neural Network

College of Information and Electronic Engineering

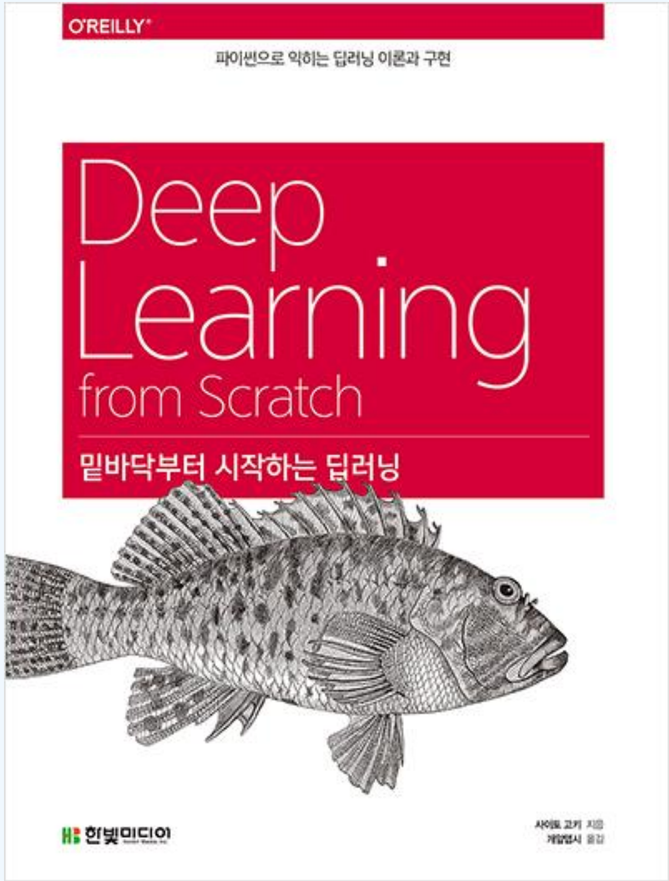
Kyung Hee Univeristy

Prof. Wonha Kim

(wonha@khu.ac.kr)

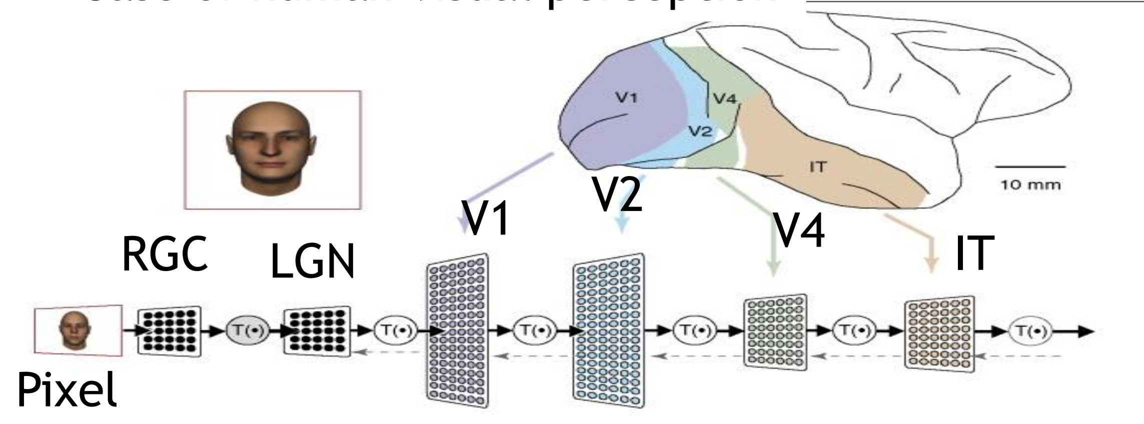
Contents

1. History of Neural Networks
2. Perceptions
3. Basics of Neural Network (NN)
   - Structure of NN
   - Graphical representation of NN
   - Matrix representation of NN
   - NN Computation : Forward Propagation
4. Neural Network Learning
   - Concept of NN Learning
   - Gradient Decent Optimization
   - Backward propagation
5. Issues of Neural Network
6. Verification



1. History of Neural Network (NN) : Human perception

- Case of human visual perception



- ➢ Photoreceptor: Intensity of light is encoded as frequency.
- ➢ RGC : Local contrast, orientation, color, movement
- ➢ LGN : Layout of scene, surface properties, global contours, etc.
- ➢ High level object recognition (V1, V2,V4,IT)
  - Associated with frontal, parietal, temporal lobe and inter-cortical
  - Visual information is encoded and highly compressed.
  - Capacity of human visual perception is limited.
  - Gesture and motion modify the visual information.
  - Short- and long-term memories easily retrieve visual information.

– James J. Dicardo *and et al*, *'How does the brain solve Visual Object Recognition?'* Neuron, vol. 73, pp.415~434, Feb. 2012

- Human brain as computing machine
  - Slow for mathematical logic, arithmetic, etc.
  - Very fast for vision, language, reasoning, etc.
  - Evolution: Light => Language=> Logic

⇩

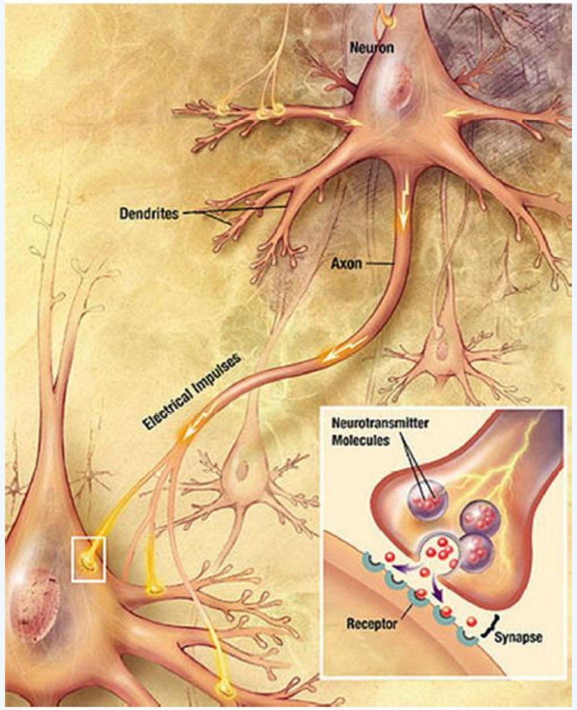
Perception is very fast. Computing is slow.

⇩

<u>**Neural Network**</u>

Exploiting hardware computation,
realize the human perception.

1. History of Neural Network (NN) : Neuron



- Neuron is an electrically excitable cell that receives, processes, and transmits information through electrical and chemical signal.
- The smallest unit of information processing in the brain
- Cell body performs simple calculations, Dendrite[déndrait] receives signals, Axon[ǽksan] transmits computation results.
- Humans have about 10^{11} neurons and a neuron connects with about 1000 other neurons. So there are about 10^{14} neuron connections.

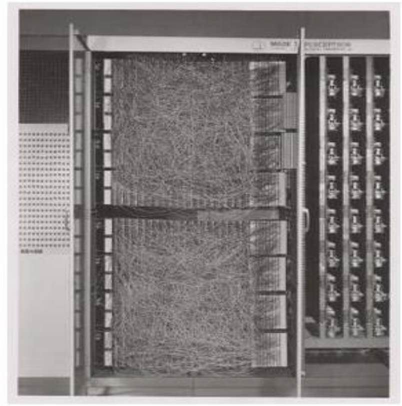
1. History of Neural Networks

- 1943년 : MCCulluch 과 Pitss 의 최초의 Neurocomputing
- 1949년 : Hebb가 학습 알고리즘 제안
- 1958년 : Mark I perceptron (F. Ronsenblatt)

  Widrow와 Hoff가 개선된 학습 알고리즘 (Adaline 과 Madaline) 발표

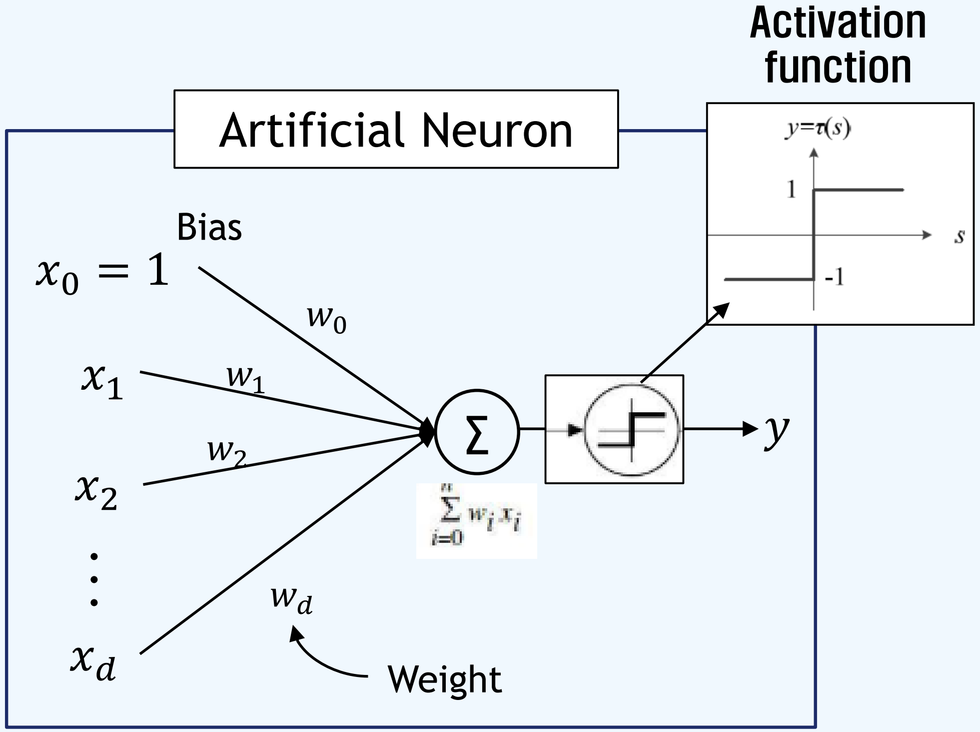
  Software was built on IBM 704 and then hardware was built.
- 1960년대 : Neuro Computing 과대 평가
- 1969년 : Mark Minsky와 Syemyour Papert 가 perceptron은 선형분류기에 불과하며 XOR의 문제조차 해결 못함을 검증 (in "Perceptrons")
- 1970년대 : 신경망 연구 퇴조
- 1986년 : Rumbelhart suggested multi-layer perceptron (『Parallel Distributed Processing』)

  Backpropagation algorithm이 정립되고 Neural Computing 연구 부활 .
- 1990년대 : Support Vector Machine (SVM)이 Neural Computing 보다 우수하다고 평가됨
- 2000년이후: Deep Learning 실현되어 신경망이 기계 학습의 주류 기술로 자리매김



Mark I Perceptron

- Artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network.
- Artificial neurons are elementary units in an artificial neural network.
- The Perceptron is an algorithm for learning a binary classifier.



- **Input vector : $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$**
- **Weight vector : $\mathbf{W} = (w_0, w_1, w_2, \cdots, w_d)^{\mathrm{T}}$**

Bias

Perceptron for learning

$$y = \begin{cases} 1 & (w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d > 0) \\ 0 & (w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d \le 0) \end{cases}$$

$$= \begin{cases} 1 & (\text{if } \mathbf{w} \cdot \mathbf{x} + w_0 > 0) \\ 0 & (\text{otherwise}) \end{cases} \Rightarrow \text{Binary classifier}$$

2. Perceptron (2/5) : Linear Classifier

- Matrix form with separated bias term

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^T,$$
$$\mathbf{w} = (w_1, w_2, \cdots, w_d)^T$$
$$s = \mathbf{w}^T\mathbf{x} + w_0,$$

- Matrix form including bias term

$$\mathbf{x} = (1, x_1, x_2, \cdots, x_d)^T$$
$$\mathbf{w} = (w_0, w_1, w_2, \cdots, w_d)^T$$
$$s = \mathbf{w}^T\mathbf{x},$$

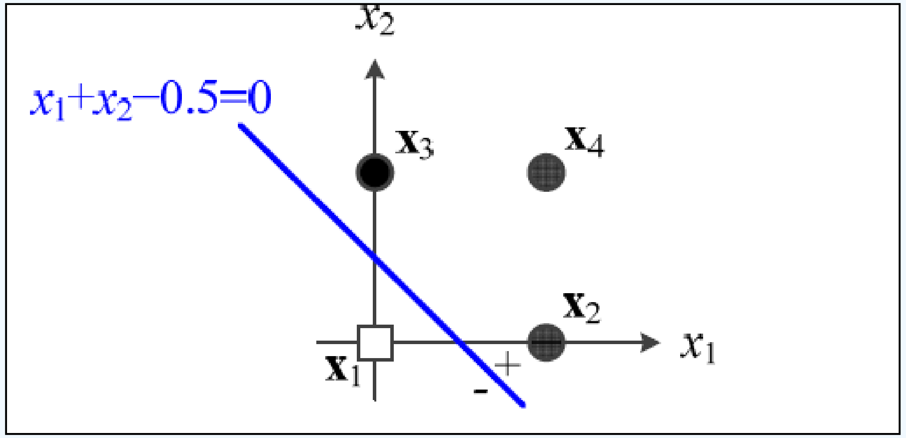
- Activation function (τ)

$$y = \tau(\mathbf{w}^T\mathbf{x})$$

- Geometric interpretation
  - Decision hyperplane (Classifier)

$$d(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_0 = 0$$

  - d=2 (2 dimension) : $d(\mathbf{x}) = d(x_1, x_2) = w_1x_1 + w_2x_2 + w_0 = 0$

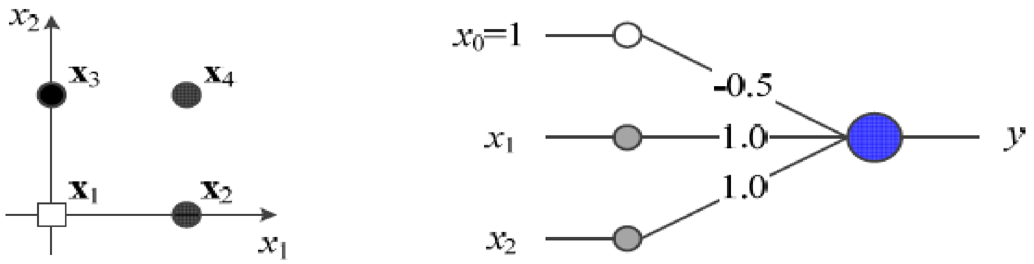


* d(x) : Classifier separating +1 and −1 spaces
* w_1 and w_2 determine direction, w_0 is the cut.
* 2-D: decision line. More than 2-D: decision plane

예제 3-1 **퍼셉트론의 동작**

2차원 특징 벡터로 표현되는 샘플을 4개 가진 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, $\mathbb{Y} = \{y_1, y_2, y_3, y_4\}$를 생각하자. [그림 3-4(a)]는 이 데이터를 보여준다.

$$\mathbf{x}_1 = \begin{pmatrix}0\\0\end{pmatrix},\ y_1 = -1,\ \ \mathbf{x}_2 = \begin{pmatrix}1\\0\end{pmatrix},\ y_2 = 1,\ \ \mathbf{x}_3 = \begin{pmatrix}0\\1\end{pmatrix},\ y_3 = 1,\ \ \mathbf{x}_4 = \begin{pmatrix}1\\1\end{pmatrix},\ y_4 = 1$$



(a) 훈련집합 (b) 퍼셉트론

그림 3-4 **OR 논리 게이트를 이용한 퍼셉트론의 동작 예시**

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

$$\begin{aligned}
&\mathbf{x}_1:\ s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, && \tau(-0.5) = -1\\
&\mathbf{x}_2:\ s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_3:\ s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_4:\ s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, && \tau(1.5) = 1
\end{aligned}$$

결국 [그림 3-4(b)]의 퍼셉트론은 샘플 4개를 모두 맞추었다. 이 퍼셉트론은 훈련집합을 100% 성능으로 분류한다고 말할 수 있다.

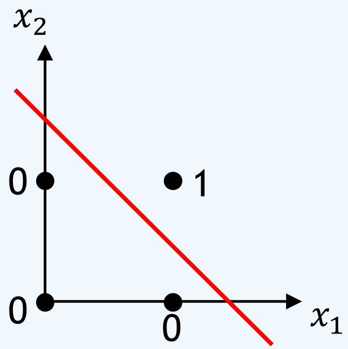
2. Perceptron (4/5) : Logical computation

- **Single layer perceptron : AND, NAND, OR**

$$y = \begin{cases} 1 & (b + w_1x_1 + w_2x_2 > 0) \\ 0 & (b + w_1x_1 + w_2x_2 \leq 0) \end{cases}$$

AND

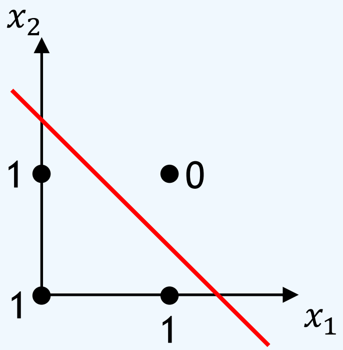
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



(w_1, w_2, b) = (0.5, 0.5, -0.7)

NAND

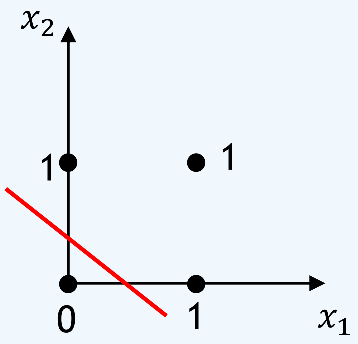
x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0



(w_1, w_2, b) = (-0.5, -0.5, 0.7)

OR

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

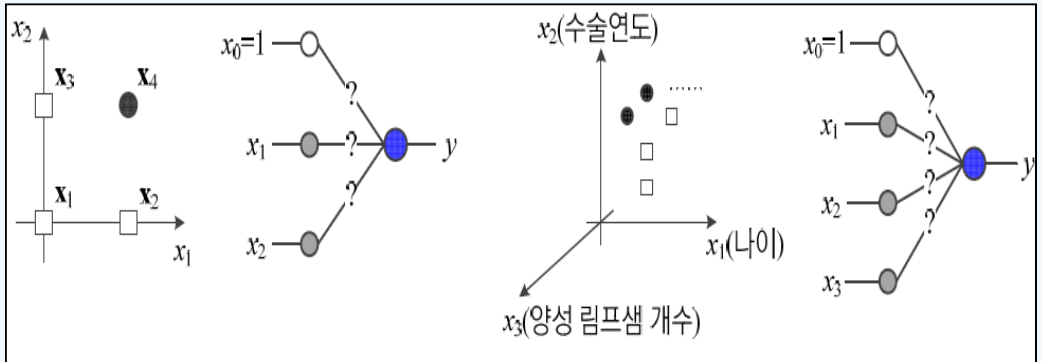


(w_1, w_2, b) = (1.0, 1.0, -0.5)

2. Perceptron (5/5) : Training

- How to determine w for the optimal classifier

  – Toy example



(a) AND 분류 문제 (b) Haberman survival 분류 문제

  – In real word, there are hundreds to tens of thousands of samples in d-dimensional space. (Ex, MNIST has 784 dimension and sixty thousand samples.)

- Parameter Training

Training Data : $\mathbf{x}_\mathrm{k} = [x_1^\mathrm{k}\ x_2^\mathrm{k}\ \cdots\ x_\mathrm{d}^\mathrm{k}]^\mathrm{T}$ (k=1,…, n), $\boldsymbol{Y} = [y_1\, y_2 \cdots\, y_n]^\mathrm{T}$

Perceptron
$\mathbf{w} = (w_0, w_1, w_2, \cdots, w_d)^\mathrm{T}$

Classifier output : $\mathrm{o_k} = \tau(\mathbf{W}^\mathrm{T} \cdot \mathbf{x}_\mathrm{k})$ → $\hat{\delta}(y_k, \mathrm{o_k})$

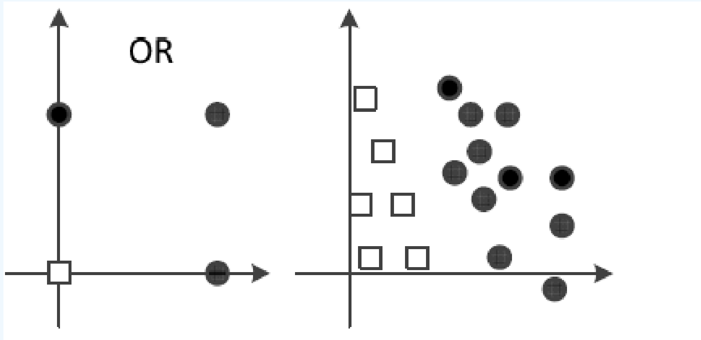
Estimate Data : $\mathbf{O} = [\hat{y}_1\, \hat{y}_2\, \cdots\, \hat{y}_n]^\mathrm{T}$ $\quad \hat{\delta}(y_k, \mathrm{o_k}) = \begin{cases} 1 & : y_k \neq \mathrm{o_k} \\ 0 & : y_k = \mathrm{o_k} \end{cases}$

Cost function : $\mathrm{J(W)} = \sum_{k=1}^{n} \hat{\delta}(y_k, \mathrm{o_k}) = \sum_{k=1}^{n} \hat{\delta}\left(y_k, \tau(\mathbf{W}^\mathrm{T} \cdot \mathbf{x}_\mathrm{k})\right)$

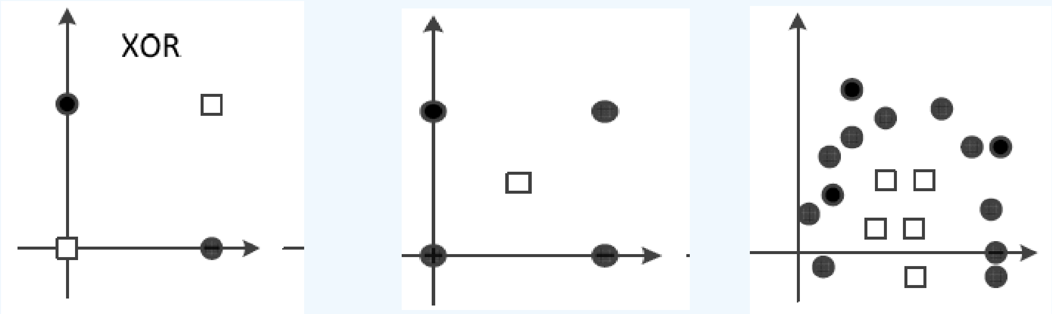
- J(W) >0
- If **W** is perfect, J(W) =0.
- More error samples, J(W) has larger value.

3. Multi-layer perceptron (1/5) : Background

- Linearly separable (OR)



- Not linearly separable (XOR)



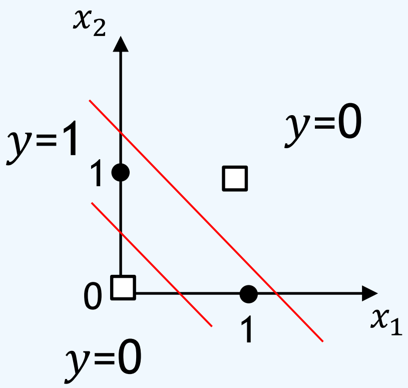
- 1969년 : Mark Minsky and Syemyour Papert 『Perceptrons』
  Pointed out the limitation of perceptron and suggest multi-layer structure. But not feasible at that time.
- 1974년 : Werbos suggested the error back-propagation
- 1986년 : Rumelhart, in his book "Parallel Distributed Processing", established mulit-layer perceptron theory
  to resurrect Neural network.

3. Multi-layer perceptron (2/5) : XOR

- Single perceptron is not possible.

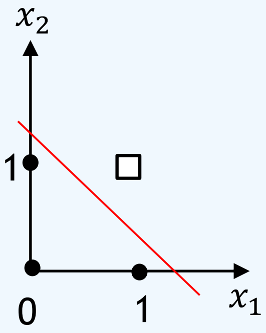
XOR

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

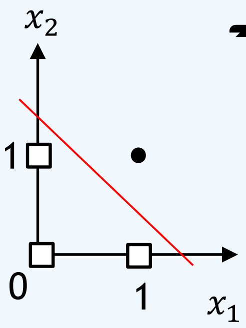


- Multi layer perceptions

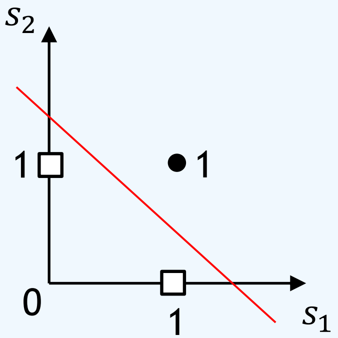
x_1	x_2	s_1
0	0	1
1	0	1
0	1	1
1	1	0

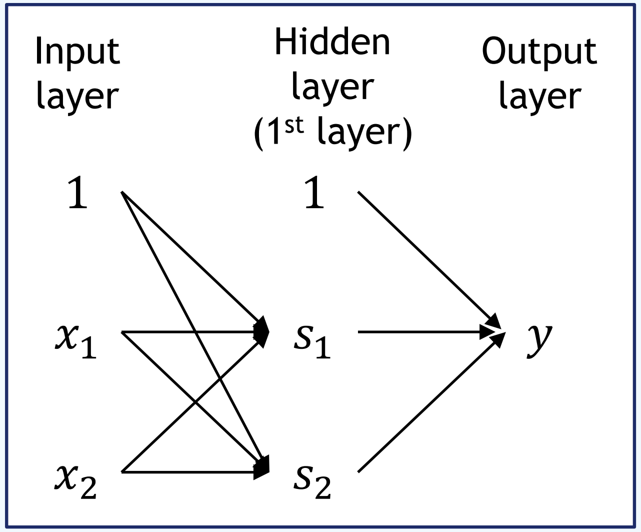


x_1	x_2	s_1
0	0	0
1	0	0
0	1	0
1	1	1



s_1	s_2	y
1	0	0
1	1	1
1	1	1
0	1	0

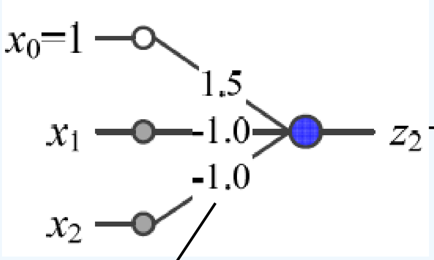


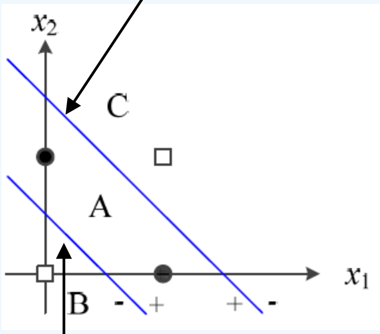


x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

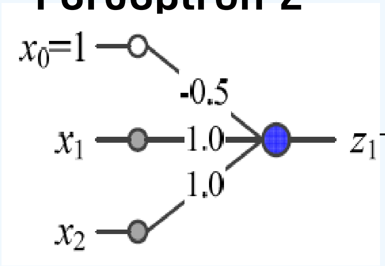
3. Multi-layer perceptron (3/5) : XOR

Perceptron 1

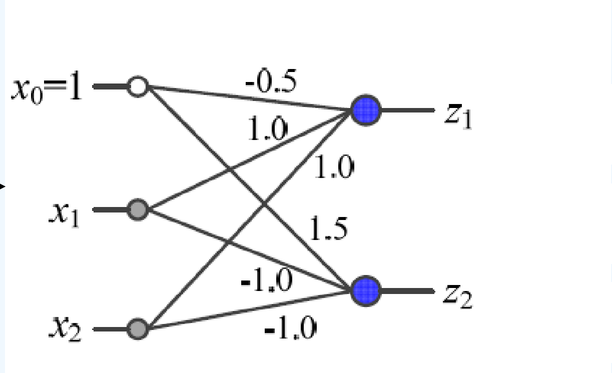


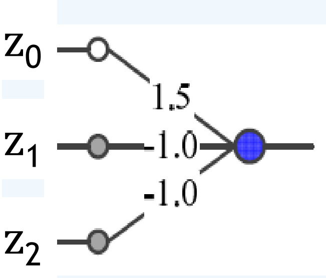


Perceptron 2

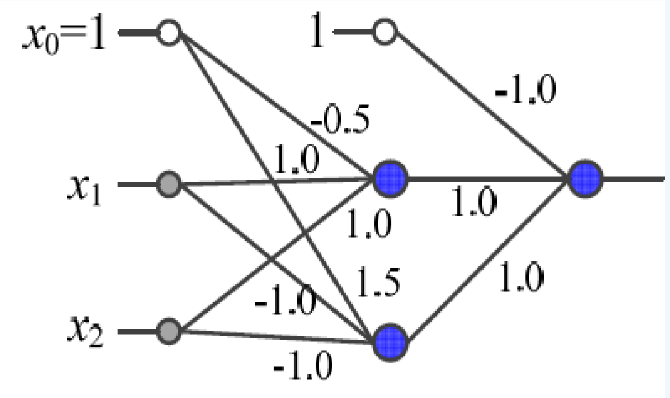


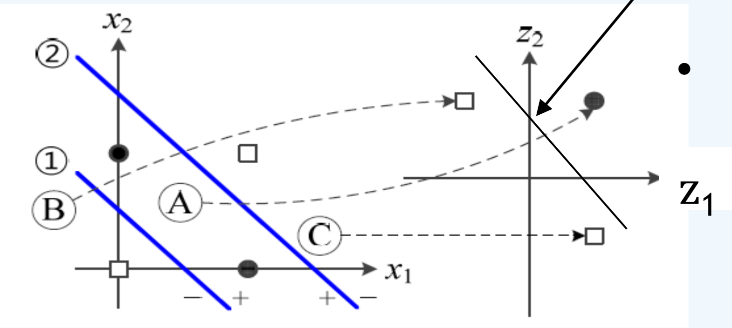
Combine two perceptrons in parallel.





Multi-layer perceptrons with 3 perceptrons.

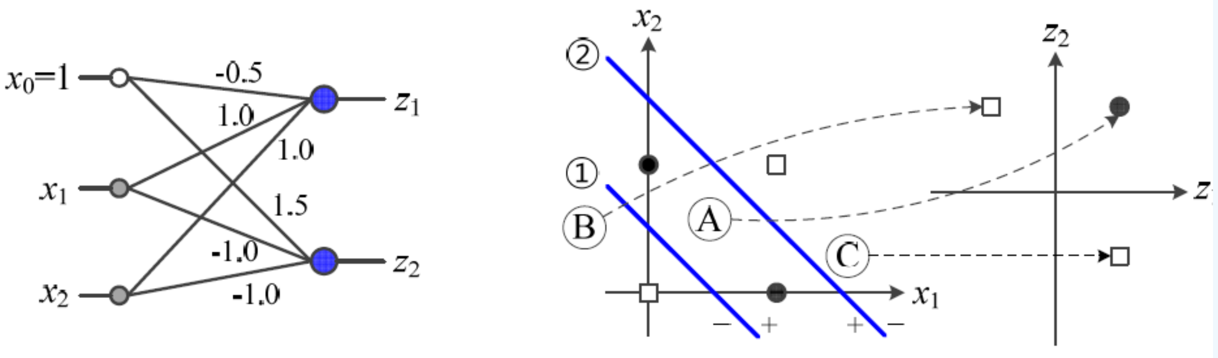




- Linear classifier at new feature space $\mathbf{z}$.

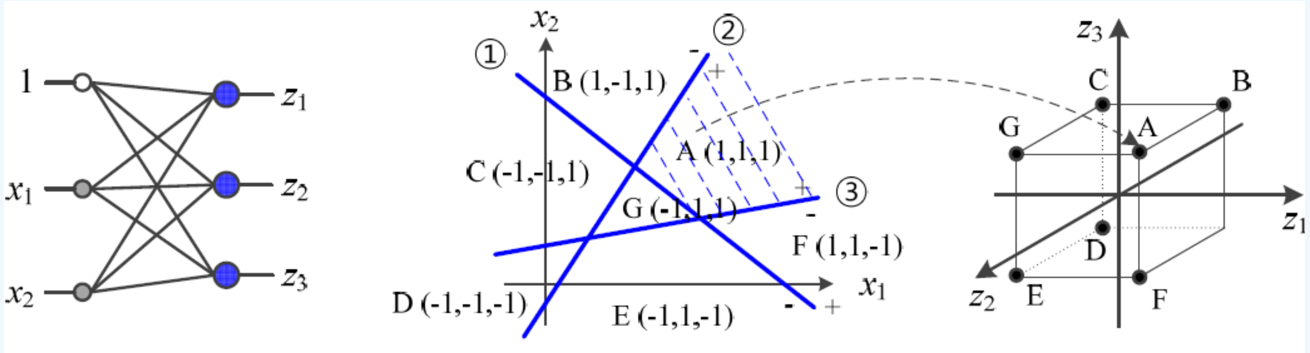
- Convert space $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$ to a new space $\mathbf{z} = (z_1, z_2)^{\mathrm{T}}$.

- Two perceptions separate 2-D space into two regions and map each region to a point at 2-D.



(a) 두 퍼셉트론을 병렬로 결합 (b) 원래 특징 공간 x를 새로운 특징 공간 z로 변환

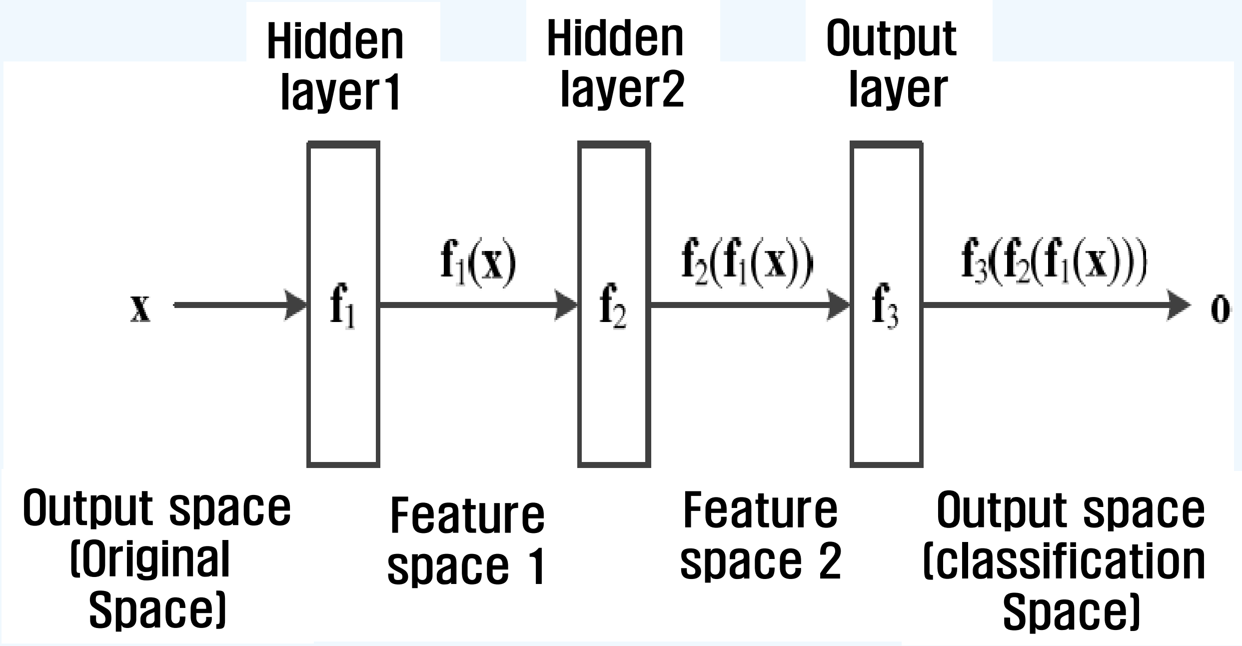
- Three perceptrons separate 2-D space into 7 regions and map each region to a point at 3-D.



(a) 퍼셉트론 3개를 결합 (b) 7개 부분공간으로 나눔 (c) 3차원 공간의 점으로 매핑

- *Comibnatin of P preceptrons*
  - separate $1 + \sum_{i=1}^{p} i$ spaces
  - convert to *p* dimensional space

- Hidden layers convert feature vectors into new feature spaces that are more efficient for classification.
- Modern machine learning names feature learning (Deep learning uses many layers.)

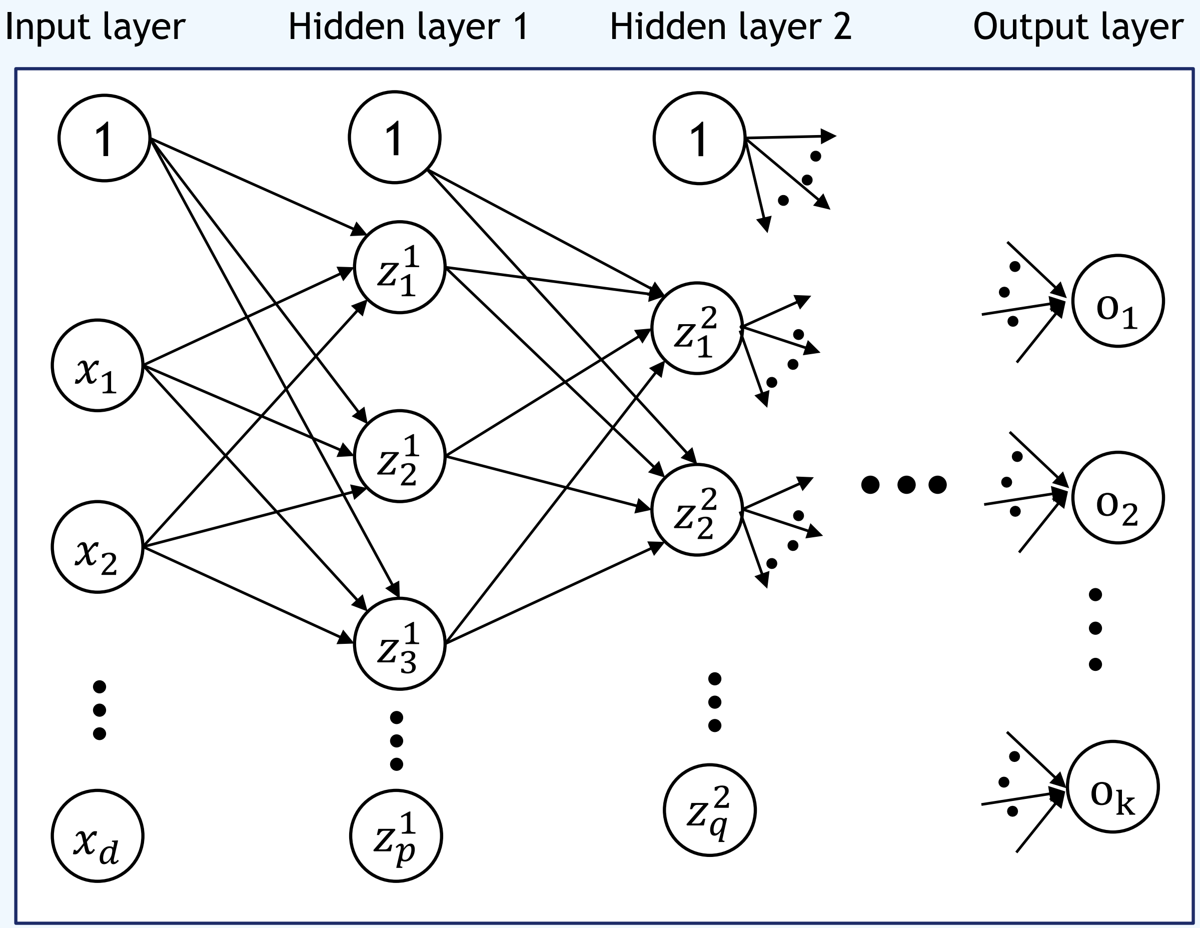


manifold

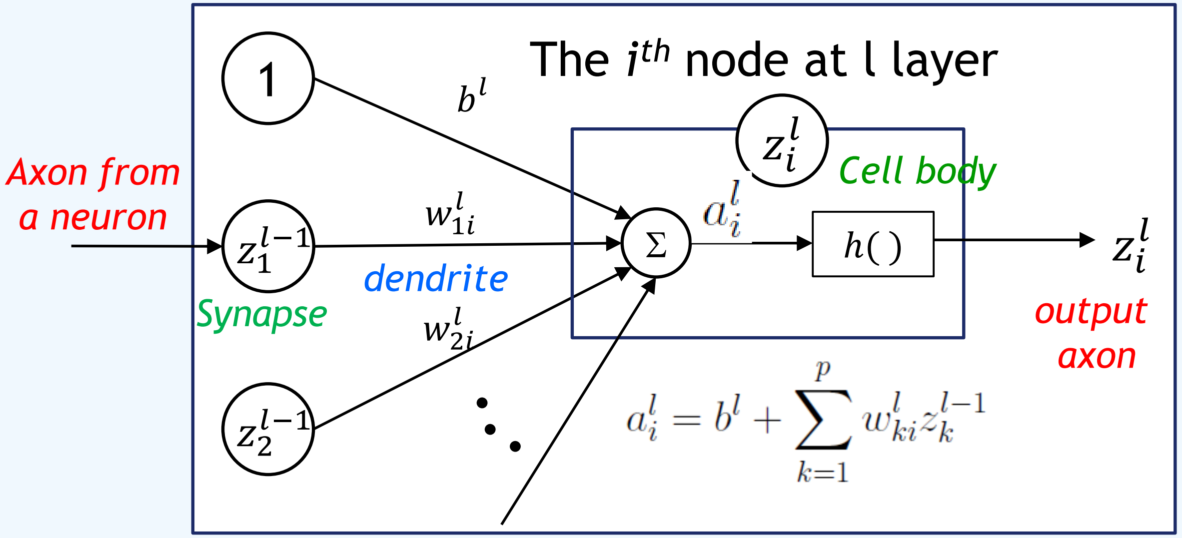
- L-layer perceptron : Deep Neural Network (DNN)

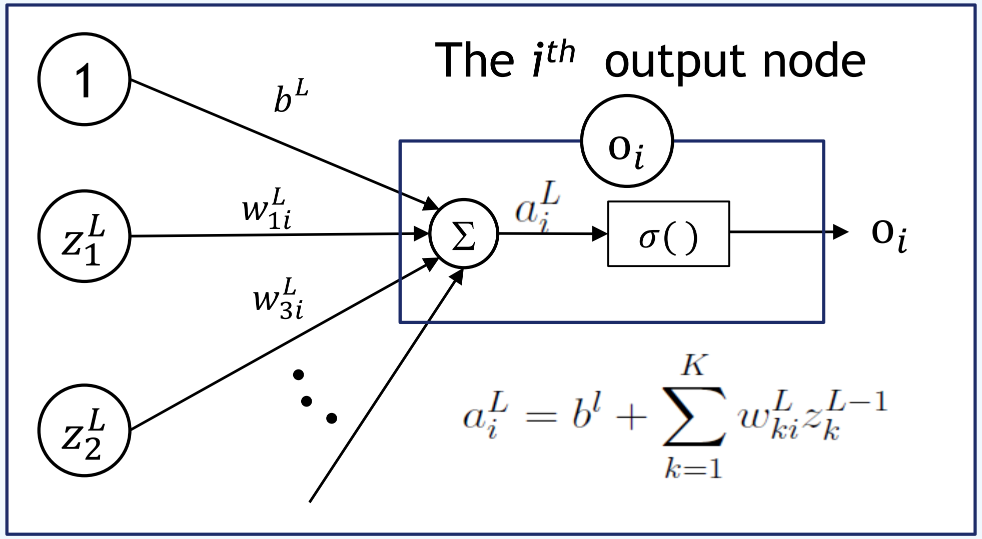
$$\mathbf{o} = \mathbf{f}_L\left(\cdots \mathbf{f}_2\left(\mathbf{f}_1(\mathbf{x})\right)\right), L \geq 4$$

4. MLP computing [1/9] : Graphical Representation

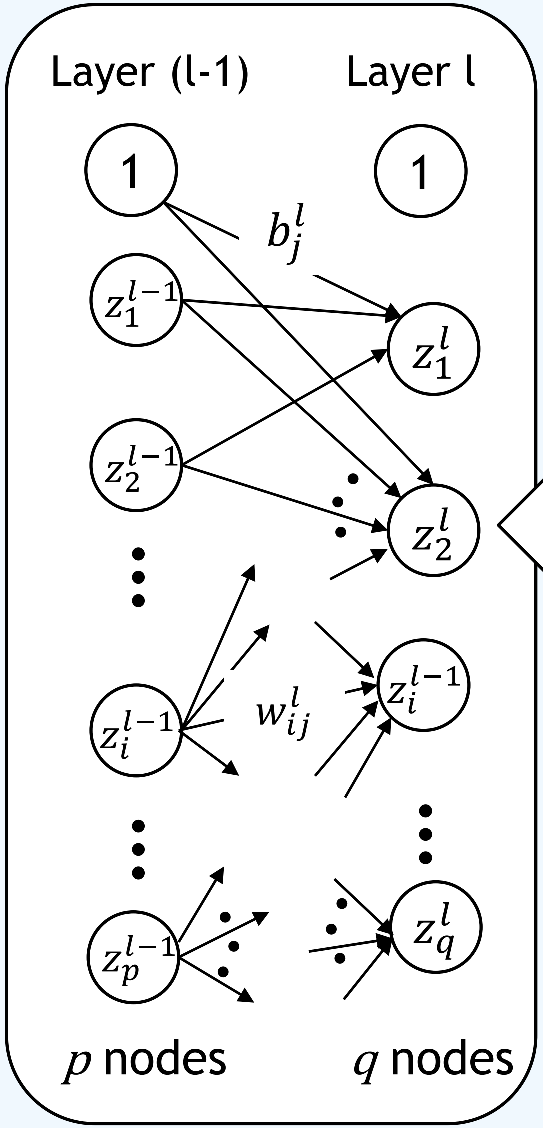


w_{ij}^{l} : weight from the i^{th} node at (l-1) layer to the j^{th} node at l layer.





L : Maximum layer depth



- b_j^l : biased value to the j^{th} node at the l-layer.
- Biased values to nodes at l-layer: $\mathbf{b}^l = [b_1^l \; b_2^l \; \ldots \; b_q^l]^T$
- w_{ij}^l : weight from the i^{th} node at (l-1) layer to the j$^{\text{th}}$ node at l-layer.
- Weights from nodes at (l-1)-layer to the j$^{\text{th}}$ node at l-layer :

$$\mathbf{w}_j^l = [w_{1j}^l \; w_{2j}^l \; \ldots \; w_{pj}^l]^T \quad (j = 1, \ldots q)$$

- Weights from (l-1)-layer to l-layer : $\mathbf{W}_{p\times q}^l = [\mathbf{w}_1^l, \mathbf{w}_2^l, \ldots, \mathbf{w}_q^l] = \begin{bmatrix} w_{11}^l & w_{12}^l & \ldots & w_{1q}^l \\ & \vdots & & \\ \cdots & w_{ij}^l & \cdots & \\ & \vdots & & \\ w_{p1}^l, & w_{p2}^l, & \ldots, & w_{pq}^l \end{bmatrix}$
- Node values at l-layer: $\mathbf{z}^l = [z_1^l \; z_2^l \; \ldots \; z_q^l]^T$
- Matrix representation from (l-1)-layer to l-layer:

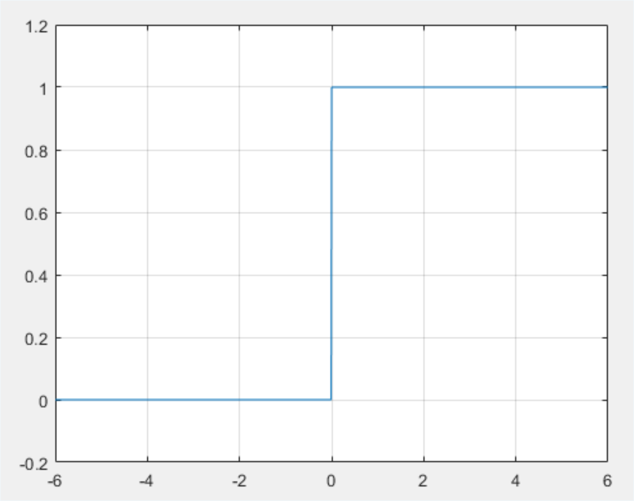
$$\mathbf{a}^l = \begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_q^l \end{bmatrix} = (\mathbf{W}_{p\times q}^l)^T \cdot \mathbf{z}^{l-1} + \mathbf{b}^l = \begin{bmatrix} w_{11}^l, & w_{21}^l, & \ldots, & w_{p1}^l \\ & \vdots & & \\ \cdots & w_{ij}^l & \cdots & \\ & \vdots & & \\ w_{1q}^l, & w_{2q}^l, & \ldots, & w_{pq}^l \end{bmatrix} \cdot \begin{bmatrix} z_1^{l-1} \\ z_2^{l-1} \\ \vdots \\ z_p^{l-1} \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_q^l \end{bmatrix} \Rightarrow \mathbf{z}^l = \begin{bmatrix} z_l^l \\ z_2^l \\ \vdots \\ z_q^l \end{bmatrix} (= h(\mathbf{a}^l)) = \begin{bmatrix} h(a_l^l) \\ h(a_2^l) \\ \vdots \\ h(a_q^l) \end{bmatrix}$$

- Input layer : $\mathbf{x}(= \mathbf{z}^{(0)}) = [\mathrm{x}_1 \, \mathrm{x}_2 \cdots \mathrm{x}_d]^T$
- Output layer : $\mathbf{o}(= \mathbf{z}^{(L+1)}) = [\mathrm{o}_1 \, \mathrm{o}_2 \cdots \mathrm{o}_c]^T$

4. MLP computing [3/9] : Activation functions for hidden layers

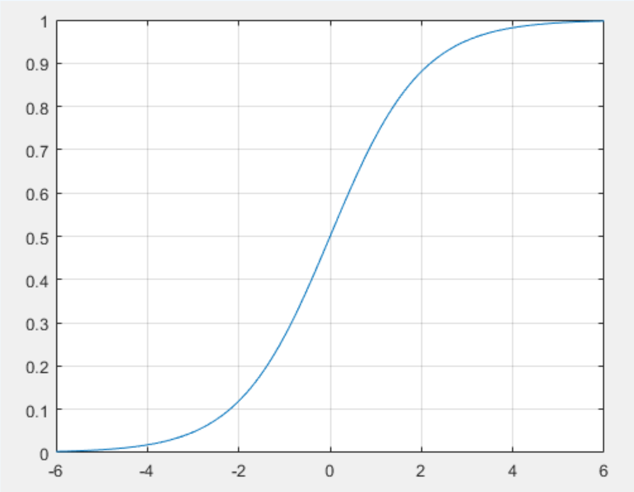
- Activation function is to convert a input signal of a node to an output signal.
- In Hidden layer, the activation function should be non-linear to learn any arbitrary complex data and be differentiable for back-propagation optimization strategy.

Step function



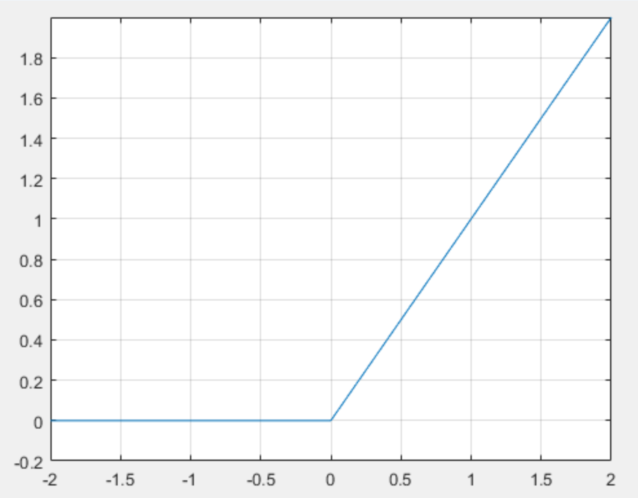
$$h(x) = \begin{cases} 1 \ (x > 0) \\ 0 \ (otherwise) \end{cases}$$

Sigmoid function

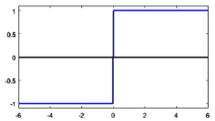
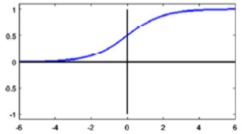
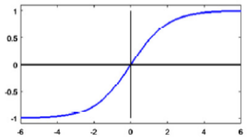
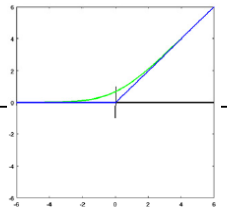


$$h(x) = \frac{1}{1 + e^{-x}}$$

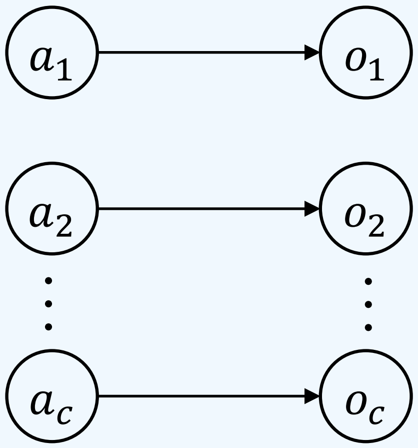
Rectified Linear Units (ReLU) function



$$h(x) = \begin{cases} x \ (x > 0) \\ 0 \ (x \leq 0) \end{cases} = \max(0, x)$$

Step	$\tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$		$\tau'(s) = \begin{cases} 0 & s \neq 0 \\ \text{N/A} & s = 0 \end{cases}$	-1 and 1
Logistic Sigmoid	$\tau(s) = \dfrac{1}{1 + e^{-as}}$		$\tau'(s) = a\tau(s)\big(1 - \tau(s)\big)$	(0,1)
Hyperbolic Tangent	$\tau(s) = \dfrac{2}{1 + e^{-as}} - 1$		$\tau'(s) = \dfrac{a}{2}(1 - \tau(s)^2)$	(-1,1)
Softplus	$\tau(s) = \log_e(1 + e^s)$		$\tau'(s) = \dfrac{1}{1 + e^{-s}}$	$(0, \infty)$
ReLU	$\tau(s) = \max(0, s)$		$\tau'(s) = \begin{cases} 0 & s < 0 \\ 1 & s > 0 \\ \text{N/A} & s = 0 \end{cases}$	$[0, \infty)$

• Identify function



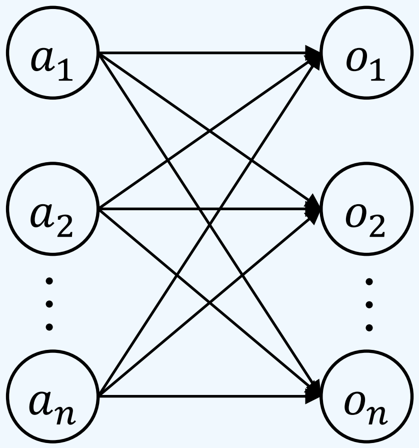
$a_1 = o_1$

$a_2 = o_2$

$a_c = o_c$

- Input = Output
- Key use of Identify function : Regression

• Softmax function



$$o_1 = \frac{\exp(a_1)}{\sum_{i=1}^{c} \exp(a_i)}$$

$$o_2 = \frac{\exp(a_2)}{\sum_{i=1}^{c} \exp(a_i)}$$

$$o_c = \frac{\exp(a_c)}{\sum_{i=1}^{c} \exp(a_i)}$$

- $o_k = \frac{\exp(a_k)}{\sum_{i=1}^{c} \exp(a_i)}$
- $o_1 + o_2 + \cdots + o_c = 1$

– Key use of Softmax function : Multiclass classification

– Key use of Sigmoid function : Binary class classification

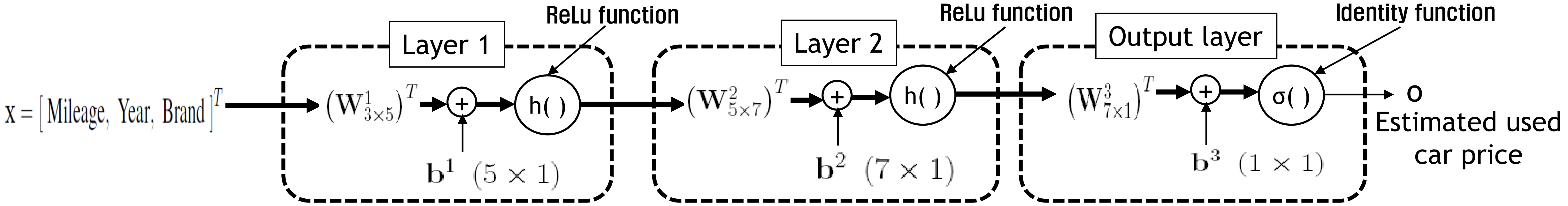
– Preventing overflow => Preventing too large values.

$$o_k = \frac{\exp(a_k)}{\sum_{i=1}^{c} \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^{c} \exp(a_i)} = \frac{\exp(a_k + \log_e C)}{\sum_{i=1}^{c} \exp(a_i + \log_e C)} = \frac{\exp(a_k + C')}{\sum_{i=1}^{c} \exp(a_i + C')}$$

where, usually, $C' = -\max(a_1, a_2, \cdots, a_c)$.

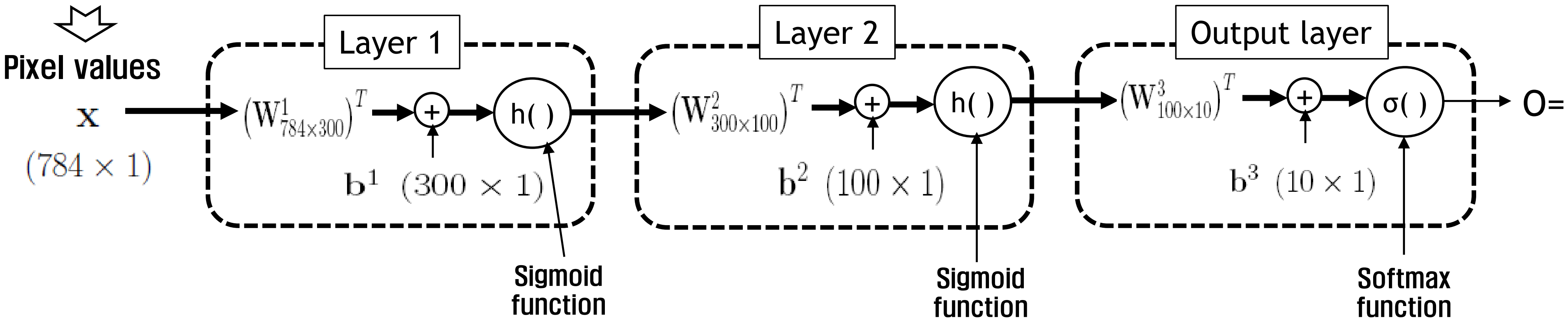
4. MLP computing [5/9] :Examples

- **Toy example : Used car price estimation**



- **MNIST Number Recognition**

* The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.



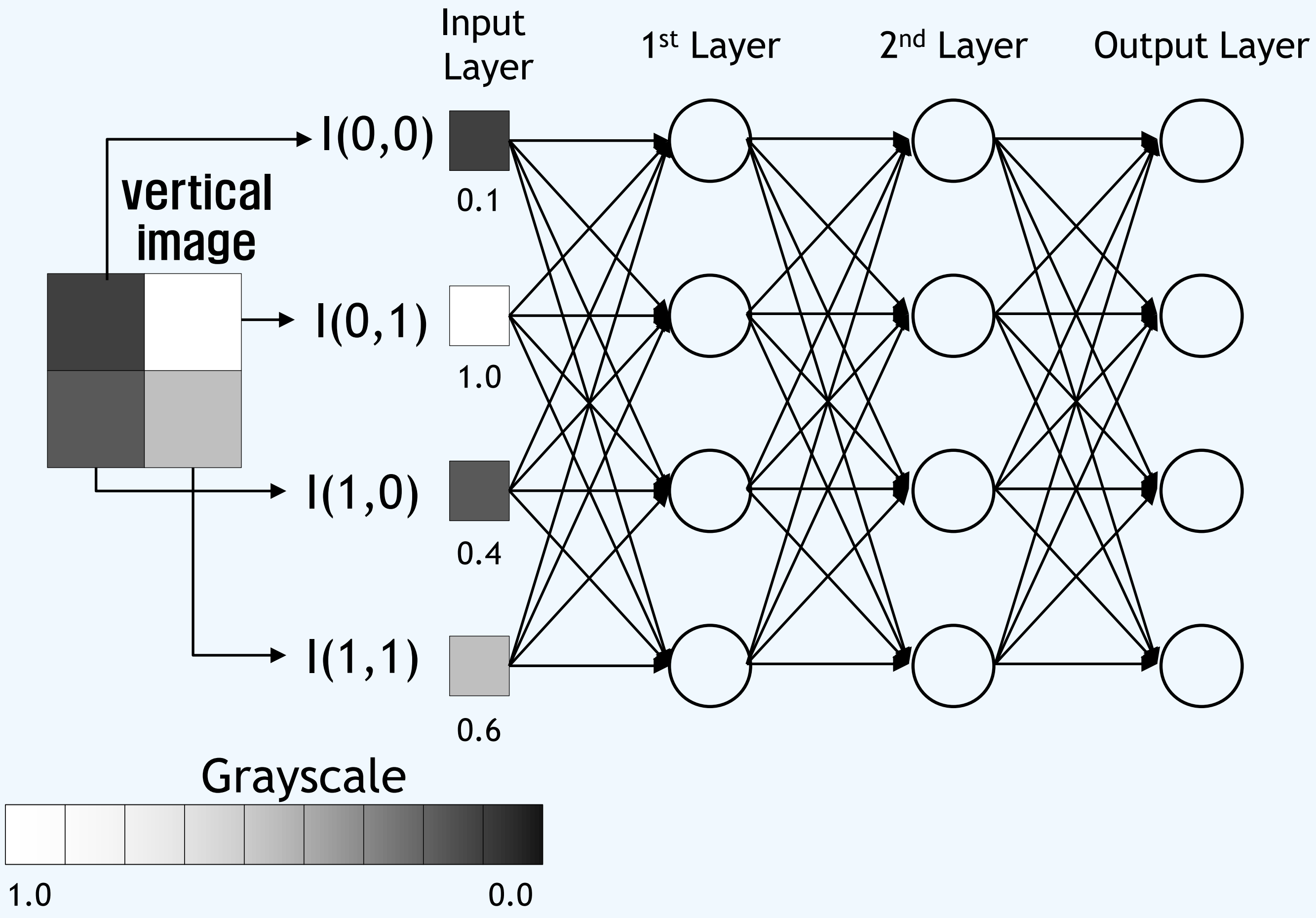
One hot encoding :
Only one is 1, all the others are 0.

O	One hot	Digit
0.01	0	0
0.02	0	1
0.71	1	2
0.05	0	3
0.07	0	4
0.03	0	5
0.02	0	6
0.07	0	7
0.01	0	8
0.01	0	9

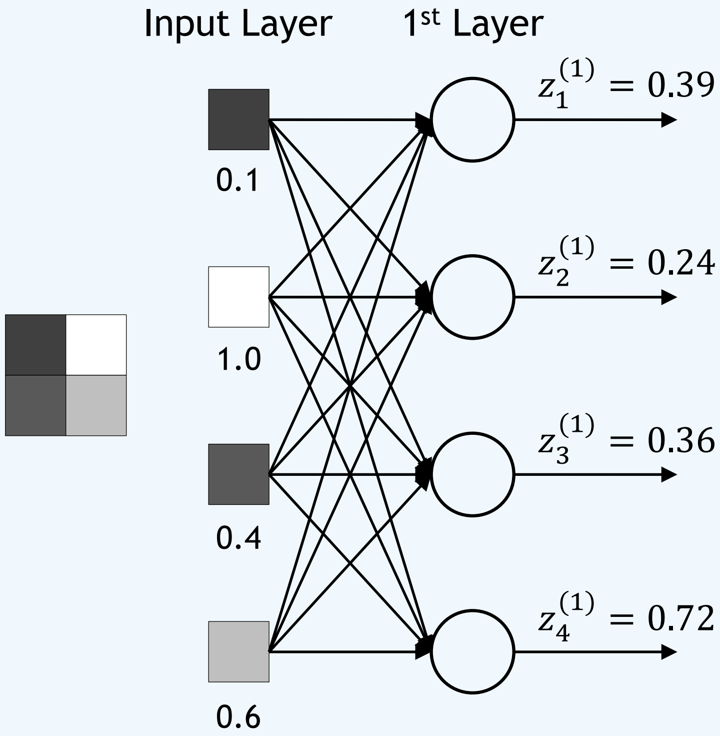
4. MLP computing [6/9] :Example-Detecting image direction (1/4)

- Detecting image direction (More sophisticated example)
  - 2x2 gray scale image, 4-directions
  - Label 4 directions with binary feature vector

Direction	Non	vertical	horizontal	diagonal
Image				
Label	[1,0,0,0]	[0,1,0,0]	[0,0,1,0]	[0,0,0,1]



- 1^{st} Layer

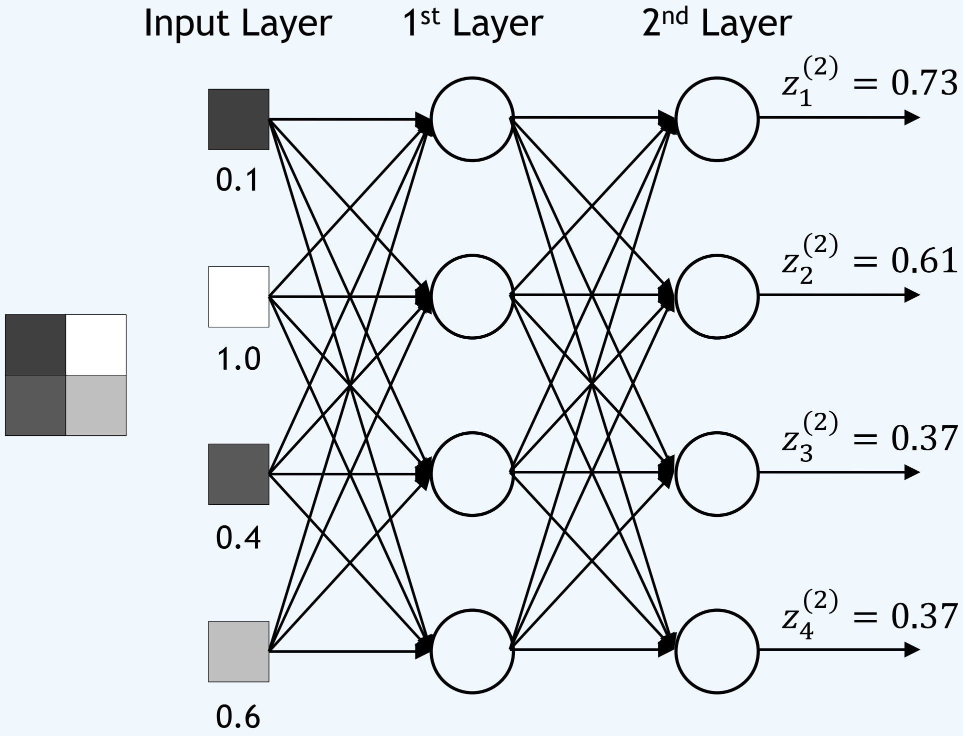


$$\underbrace{\begin{bmatrix} -0.82 & 0.71 & -0.55 & -0.08 \\ 0.19 & 0.93 & 0.07 & 0.28 \\ -0.52 & -0.02 & 0.52 & 0.83 \\ 0.68 & -0.56 & -0.30 & -0.68 \end{bmatrix}}_{W^1_{4x4}} \bullet \underbrace{\begin{bmatrix} 0.1 \\ 1.0 \\ 0.3 \\ 0.6 \end{bmatrix}}_{\mathbf{x}^1} + \underbrace{\begin{bmatrix} 0.02 \\ 0.01 \\ -0.01 \\ 0.04 \end{bmatrix}}_{\mathbf{b}^1} = \underbrace{\begin{bmatrix} 0.44 \\ 1.14 \\ 0.58 \\ -0.95 \end{bmatrix}}_{\mathbf{a}^1}$$

$$\underbrace{\begin{bmatrix} 0.44 \\ 1.14 \\ 0.58 \\ -0.95 \end{bmatrix}}_{\mathbf{a}^1} \Rightarrow \boxed{h(a)} \Rightarrow \underbrace{\begin{bmatrix} 0.39 \\ 0.24 \\ 0.36 \\ 0.72 \end{bmatrix}}_{\mathbf{z}^1}$$

$$h(a) = \frac{1}{1 + \exp(a)}$$

- 2nd Layer

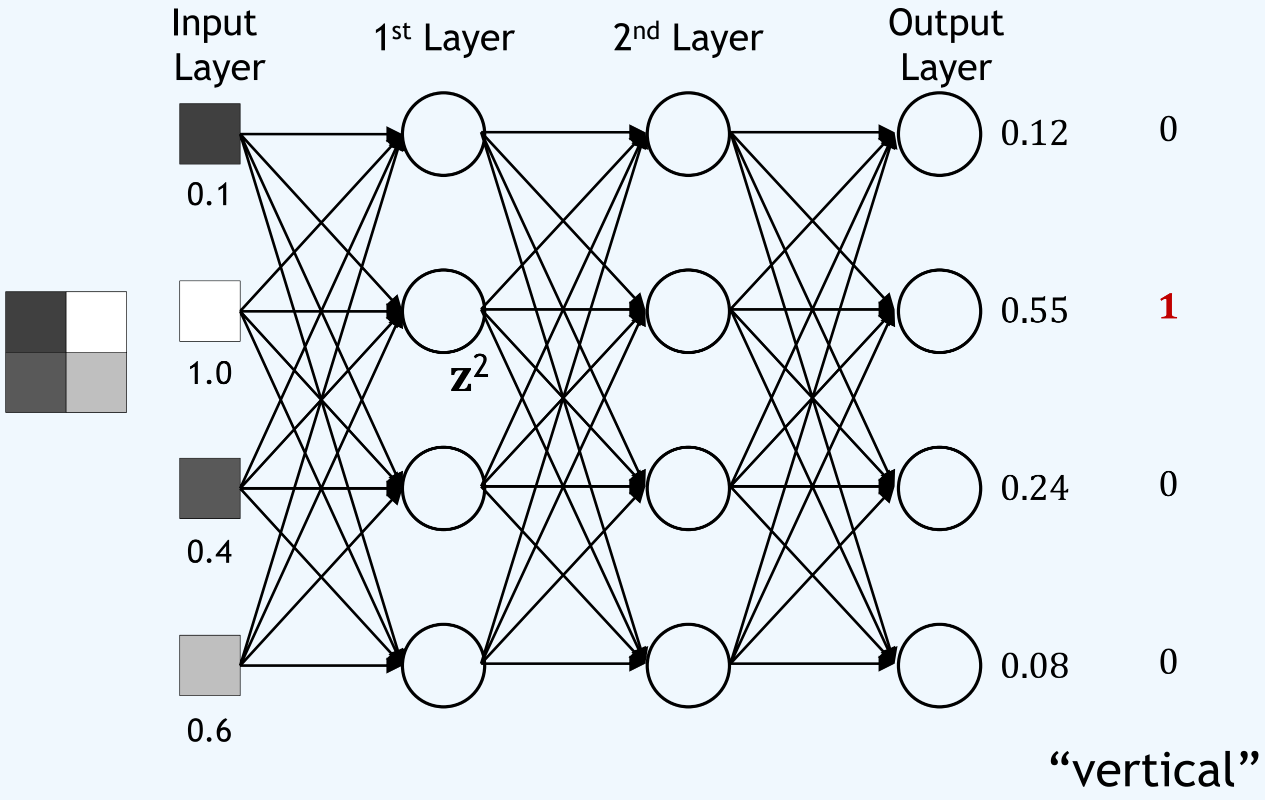


$$\overset{W^2_{4x4}}{\begin{bmatrix} -0.21 & -0.34 & -0.75 & -0.78 \\ -0.64 & 0.61 & -0.54 & -0.19 \\ 0.27 & 1.00 & -0.95 & 0.77 \\ 0.25 & 0.96 & 0.21 & 0.10 \end{bmatrix}} \times \overset{\mathbf{z}^1}{\begin{bmatrix} 0.39 \\ 0.24 \\ 0.36 \\ 0.72 \end{bmatrix}} + \overset{\mathbf{b}^2}{\begin{bmatrix} -0.01 \\ -0.03 \\ -0.01 \\ 0.05 \end{bmatrix}} = \overset{\mathbf{a}^2}{\begin{bmatrix} -1.01 \\ -0.46 \\ 0.55 \\ 0.52 \end{bmatrix}}$$

$$\overset{\mathbf{a}^2}{\begin{bmatrix} -1.01 \\ -0.46 \\ 0.55 \\ 0.52 \end{bmatrix}} \Rightarrow \boxed{h(a)} \Rightarrow \overset{\mathbf{z}^2}{\begin{bmatrix} 0.73 \\ 0.61 \\ 0.37 \\ 0.37 \end{bmatrix}}$$

$$\frac{1}{1 + \exp(a)}$$

- Output Layer



$$\underset{W^3_{4x4}}{\begin{bmatrix} -0.75 & 0.38 & 0.37 & -0.61 \\ -0.06 & 0.96 & 0.82 & 0.51 \\ 0.71 & -0.43 & 0.22 & -0.31 \\ -0.91 & -0.73 & 0.80 & -0.16 \end{bmatrix}} \times \underset{\mathbf{z}^3}{\begin{bmatrix} 0.73 \\ 0.61 \\ 0.37 \\ 0.37 \end{bmatrix}} + \underset{\mathbf{b}^3}{\begin{bmatrix} -0.03 \\ 0.03 \\ 0.01 \\ 0.02 \end{bmatrix}} = \underset{\mathbf{a}^3}{\begin{bmatrix} -0.44 \\ 1.07 \\ 0.24 \\ -0.86 \end{bmatrix}}$$

$$\underset{\mathbf{a}^3}{\begin{bmatrix} -0.44 \\ 1.07 \\ 0.24 \\ -0.86 \end{bmatrix}} \Rightarrow \boxed{\sigma(a)} \Rightarrow \underset{\mathbf{y}}{\begin{bmatrix} 0.12 \\ 0.55 \\ 0.24 \\ 0.08 \end{bmatrix}} \Rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

“vertical”

5. MLP Training [1/18] : Overview

- Training Data (number of features : d, number of output nodes : c, number of training data : n)

k^{th} feature vector : $\mathbf{x}_k = [x_1^k \ x_2^k \ \cdots \ x_d^k]^T$ (k=1,..., n)

Feature data matrix: $\mathbf{X} = [\mathbf{x}_1 \, \mathbf{x}_2 \ldots \mathbf{x}_n]^T = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^n \\ \vdots & & \ddots & \vdots \\ x_d^1 & x_d^2 & \cdots & x_d^n \end{bmatrix}$

k^{th} label (target) vector : $\mathbf{y}_k = [y_1^k \ y_2^k \ \cdots \ y_c^k]^T$ (k=1,..., n)

Label data matrix: $\mathbf{Y} = [\mathbf{y}_1 \, \mathbf{y}_2 \ldots \mathbf{y}_n]^T = \begin{bmatrix} y_1^1 & y_1^2 & \cdots & y_1^n \\ \vdots & & \ddots & \vdots \\ y_c^1 & y_c^2 & \cdots & y_c^n \end{bmatrix}$

X

L-layer Neural Network with **W**

Estimate Data : $\mathbf{O} = f(X|W)$ ($\mathbf{O} = [\mathbf{o}_1 \, \mathbf{o}_2 \ldots \mathbf{o}_n]^T$, $\mathbf{o}_k = [o_1^k \ y_2^k \ \cdots \ o_c^k]^T$)

Loss ← **Y**

Cost function : $\mathrm{J}(\mathbf{W}) = \mathbf{E}(\|\mathbf{Y} - \mathbf{O}\|)$

Loss : $\mathbf{E}(\|\mathbf{Y} - \mathbf{O}\|)$

- Mean Square Error
- Cross Entropy
- Others

- Neural Network Training

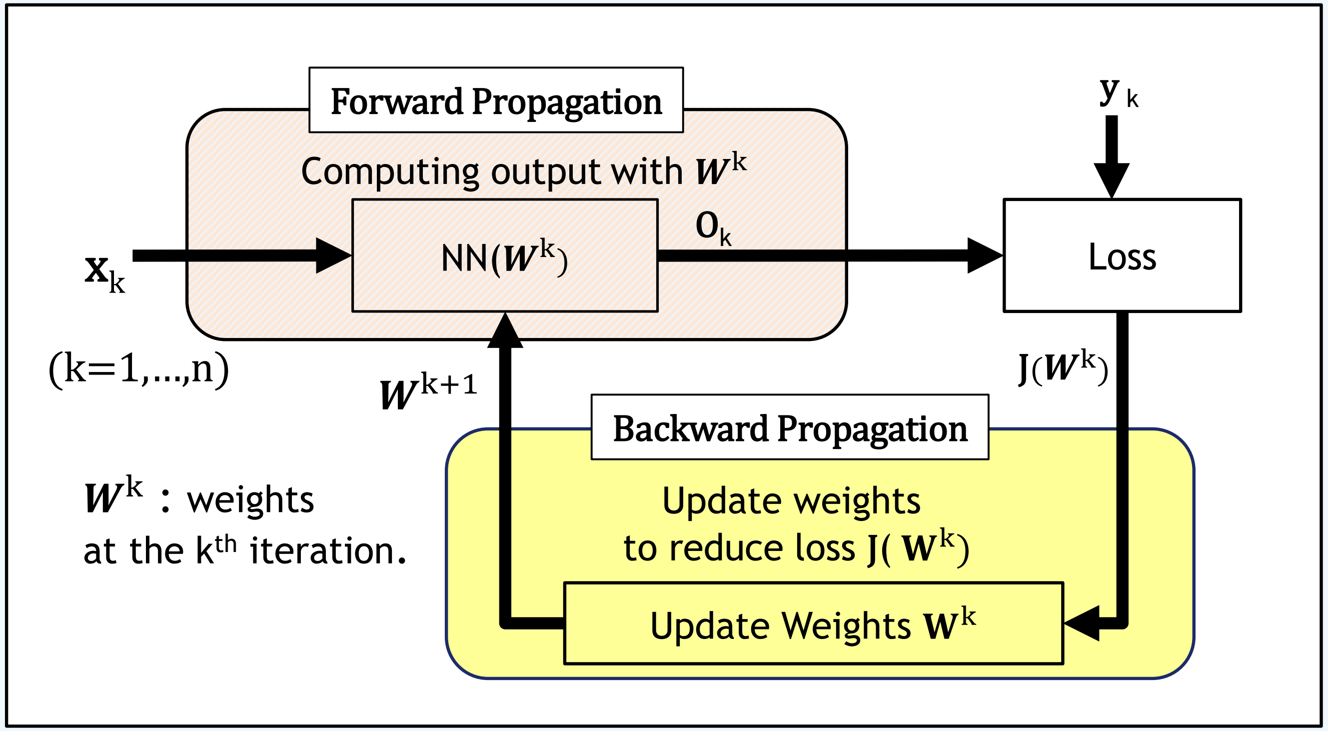
$\widehat{W} = \underset{W}{\operatorname{argmin}} \, \mathrm{J}(W)$

Gradient Descent

$\widehat{\mathbf{W}} = \mathbf{W} - \eta \cdot \nabla \mathbf{J}(W) = \mathbf{W} - \eta \cdot \frac{J(W)}{W}$

5. MLP Training [2/18] : Neural Net Training

- For a given NN structure, determine the optimal weights.
- Recursively update weights to reduce loss at every iteration.
- Terminate the iteration when the loss of training data reaches at a saturation value.



Typical Loss functions

- Mean Squared Error (MSE)

$$\mathbf{J}(W^k) = \sum_i^c (y_i^k - o_i^k)^2$$

(Useful for regression)

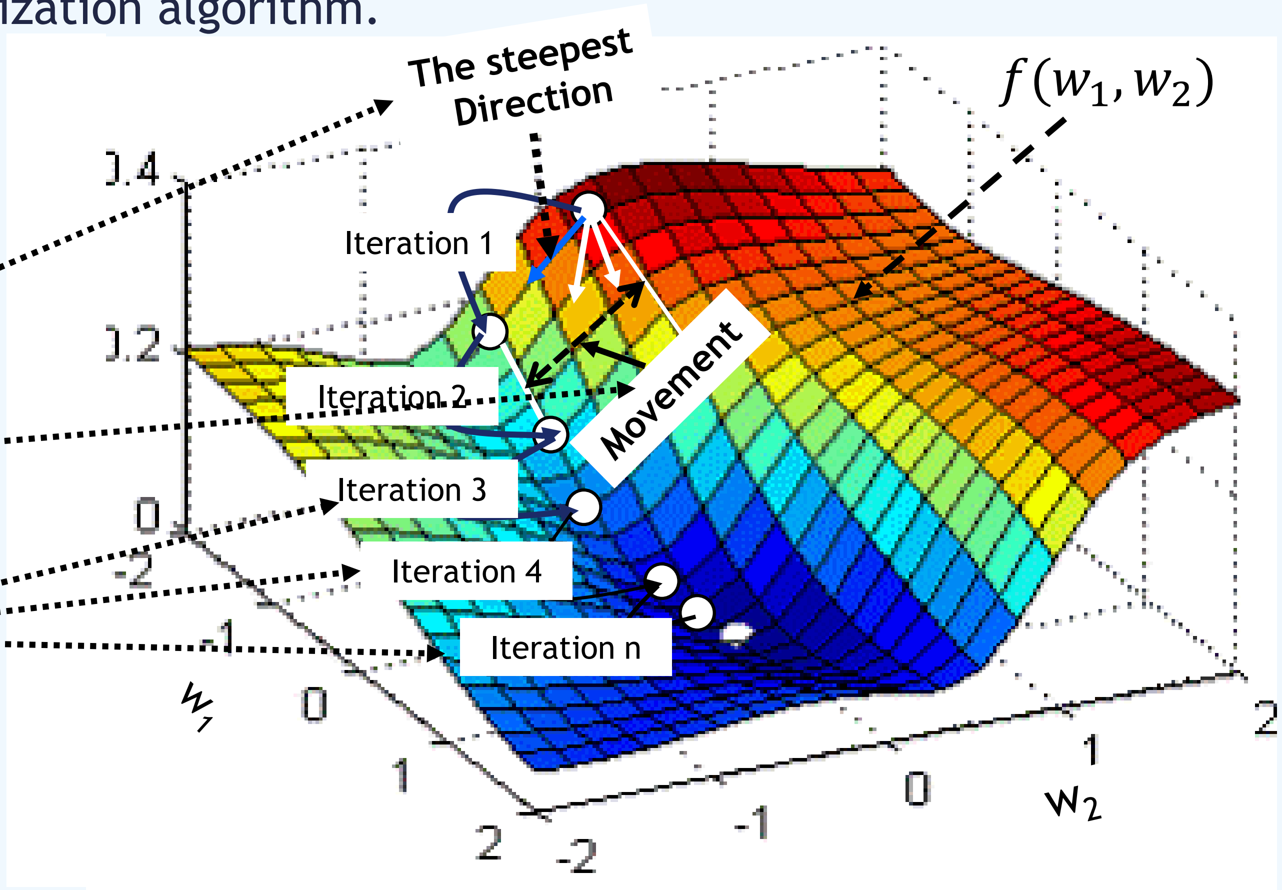
- Cross Entropy Error (CEE)

$$\mathbf{J}(W^k) = -\sum_i^c y_i^k \ln o_i^k$$

(Useful for classification)

- Loss functions have the minimum for convergence.
- Function of weights W (not X).

- Gradient Descent
- Goal : *Numerically* determine the minimum of a smooth function $f(w_1, w_2, \ldots, w_L)$.
- Gradient descent is a first-order iterative optimization algorithm.
- Procedure :
  - ➢ Decide the direction that diminishes the function at most. (Steepest direction)
  - ➢ Decide proper movement rate.
    Movement = rate * slope
  - ➢ Iterate until the function values saturate or reaches at the minimum.



5. MLP Training [4/18] : Gradient Descent (SDG) (2/3)

<u>*From Taylor series*</u>

- $L = \mathbf{J}(\mathbf{W}) = \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2)$
- $L + \Delta L = J(\mathrm{w}_1 + \Delta \mathrm{w}_1, \mathrm{w}_2 + \Delta \mathrm{w}_2) \approx \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2) + \frac{\partial \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2)}{\partial \mathrm{w}_1}\Delta \mathrm{w}_1 + \frac{\partial \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2)}{\partial \mathrm{w}_2}\Delta \mathrm{w}_2$
- $\Delta L \approx \frac{\partial J(\mathrm{w}_1,\mathrm{w}_2)}{\partial \mathrm{w}_1}\Delta \mathrm{w}_1 + \frac{\partial J(\mathrm{w}_1,\mathrm{w}_2)}{\partial \mathrm{w}_2}\Delta \mathrm{w}_2 = \left(\frac{\partial J(\mathrm{w}_1,\mathrm{w}_2)}{\partial \mathrm{w}_1}, \frac{\partial J(\mathrm{w}_1,\mathrm{w}_2)}{\partial \mathrm{w}_2}\right) \cdot (\Delta \mathrm{w}_1, \Delta \mathrm{w}_2) = \Delta \mathbf{J}(\mathbf{W}) \cdot \Delta \mathbf{W}$

To decrease ΔL maximally, $\Delta \mathbf{W} = -\eta \Delta \mathbf{J}(\mathbf{W})$ (Learning rate: $\eta > 0$)

Take steps proportional to the negative of the gradient of the function at the current point.

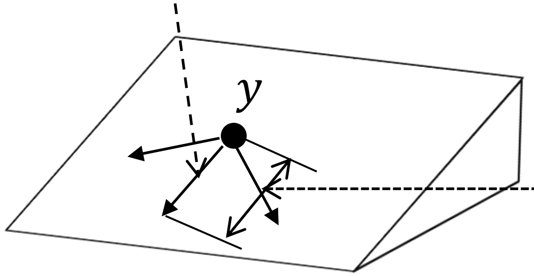
$$(\Delta \mathrm{w}_1, \Delta \mathrm{w}_2) = -\eta \left(\frac{\partial \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2)}{\partial \mathrm{w}_1}, \frac{\partial \mathbf{J}(\mathrm{w}_1, \mathrm{w}_2)}{\partial \mathrm{w}_2}\right)$$

The steepest direction:
Direction of $\Delta J(\mathbf{W})$

Movement:

$$|\Delta \mathbf{W}| = \eta |\Delta \mathbf{J}(\mathbf{W})|$$

= rate * The slope of the steepest direction



Weight Update

New Old

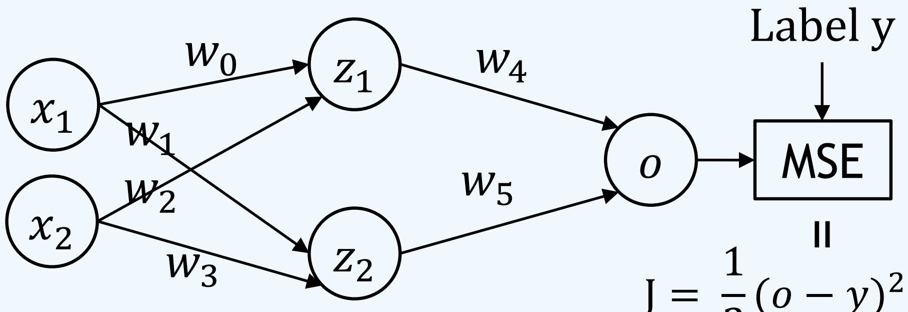
$$w_1^{n+1} = w_1^n + \Delta w_1 = w_1^n - \eta \frac{\partial \mathbf{J}(w_1, w_2, \dots, w_M)}{\partial \theta_1}$$

$$\vdots$$

$$w_M^{n+1} = w_M^n + \Delta w_M = w_M^n - \eta \frac{\partial \mathbf{J}(w_1, w_2, \dots, w_M)}{\partial w_M}$$

$\widehat{W} = \underset{W}{\mathrm{argmin}}\ \mathrm{J}(W)$ $\Leftrightarrow$ $\widehat{\mathbf{W}} = \mathbf{W} - \eta \cdot \nabla \mathbf{J}(W) = \mathbf{W} - \eta \cdot \frac{J(W)}{\partial W}$

- Very simple network



$$J = \frac{1}{2}(o-y)^2$$

$$J = (w_0 w_4 x_1 + w_2 w_4 x_2 + w_1 w_5 x_1 + w_3 w_5 x_2 - y)^2$$

$$\frac{\partial J}{\partial w_0} = \frac{\partial((w_0w_4x_1+w_2w_4x_2+w_1w_5x_1+w_3w_5x_2-y)^2)}{\partial w_0}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial((w_0w_4x_1+w_2w_4x_2+w_1w_5x_1+w_3w_5x_2-y)^2)}{\partial w_1}$$

$$\vdots$$

$$\frac{\partial J}{\partial w_5} = \frac{\partial((w_0w_4x_1+w_2w_4x_2+w_1w_5x_1+w_3w_5x_2-y)^2)}{\partial w_5}$$

Too complicated to be implemented in real systems.

- Using Chain rule : formula for computing the derivative of the composition of two or more functions.

$$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_0}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_1}$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_2}$$

$$\frac{\partial J}{\partial w_3} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3}$$

$$\frac{\partial J}{\partial w_4} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial w_4}$$

$$\frac{\partial J}{\partial w_5} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial w_5}$$

$$o = w_0 w_4 x_1 + w_2 w_4 x_2 + w_1 w_5 x_1 + w_3 w_5 x_2$$

$$z_1 = w_0 x_1 + w_2 x_2, \qquad z_1 = w_1 x_1 + w_3 x_2$$

$\frac{\partial J}{\partial o} = (o-y)$: Last layer

$$\frac{\partial J}{\partial z_1} = \frac{\partial(w_4 z_1 + w_5 z_2)}{\partial z_1} = w_4 + w_5 z_2$$

$$\frac{\partial J}{\partial z_2} = \frac{\partial(w_4 z_1 + w_5 z_2)}{\partial z_2} = w_4 z_1 + w_5$$

: 1-layer

$$\frac{\partial z_1}{\partial w_0} = \frac{\partial(w_0 x_1 + w_2 x_2)}{\partial w_0} = x_1 + w_2 x_2$$

$$\frac{\partial z_1}{\partial w_2} = \frac{\partial(w_0 x_1 + w_2 x_2)}{\partial w_2} = w_0 x_1 + x_2$$

$$\frac{\partial z_2}{\partial w_1} = \frac{\partial(w_1 x_1 + w_3 x_2)}{\partial w_1} = x_1 + w_3 x_2$$

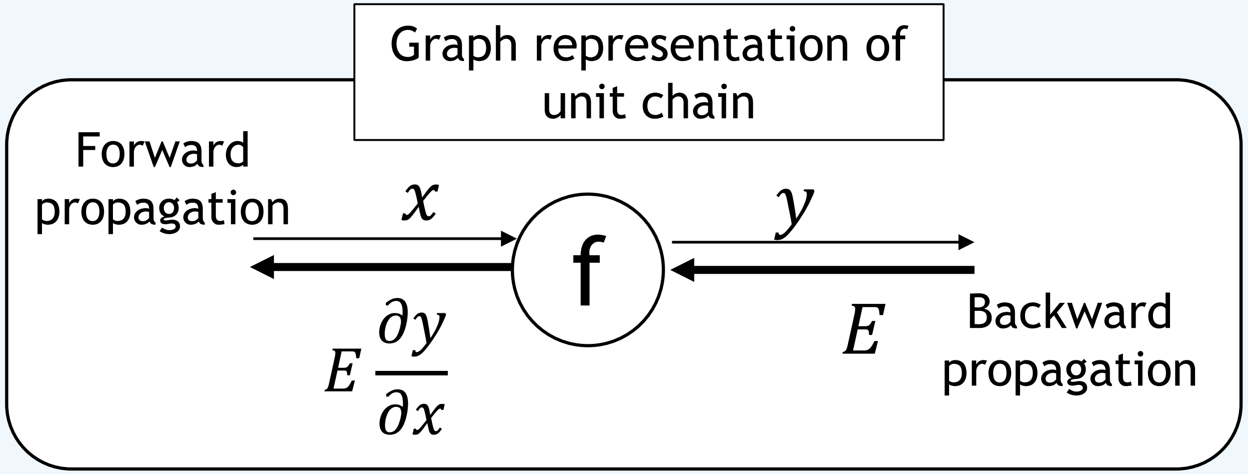
$$\frac{\partial z_2}{\partial w_3} = \frac{\partial(w_1 x_1 + w_3 x_2)}{\partial w_3} = w_1 x_1 + x_2$$

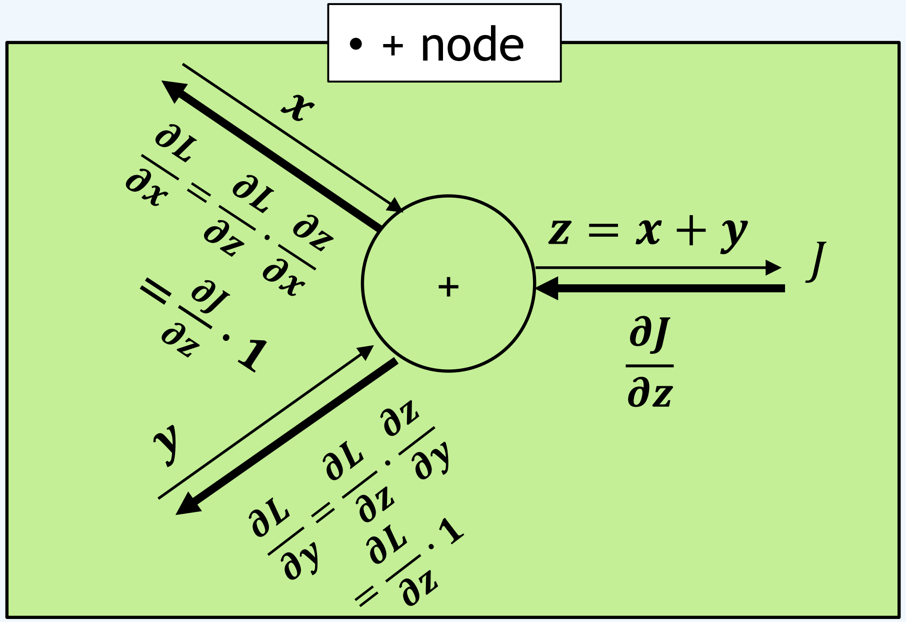
: Input layer

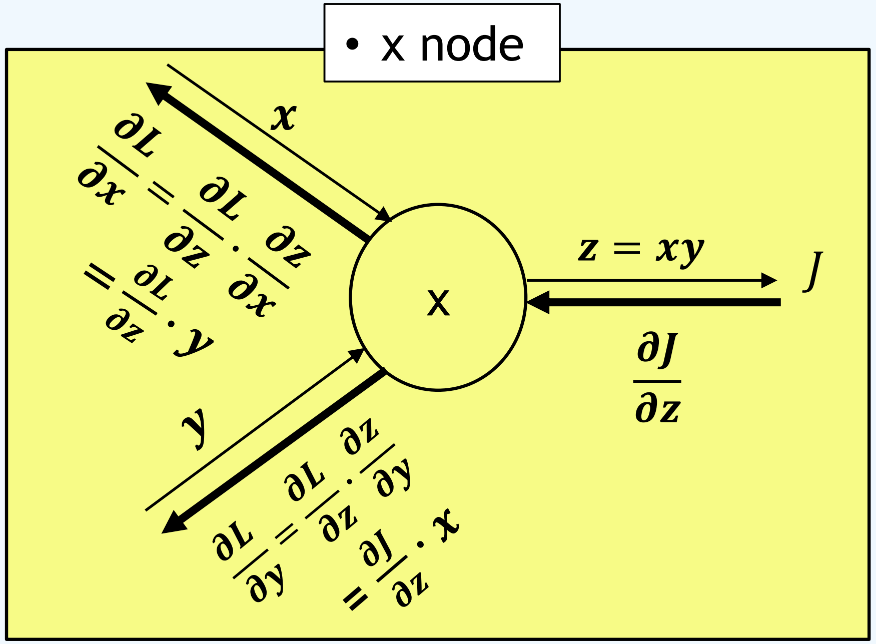
Computations at nodes is repeatedly used.
So, much more efficient

5. MLP Training [6/18] : Backward propagation

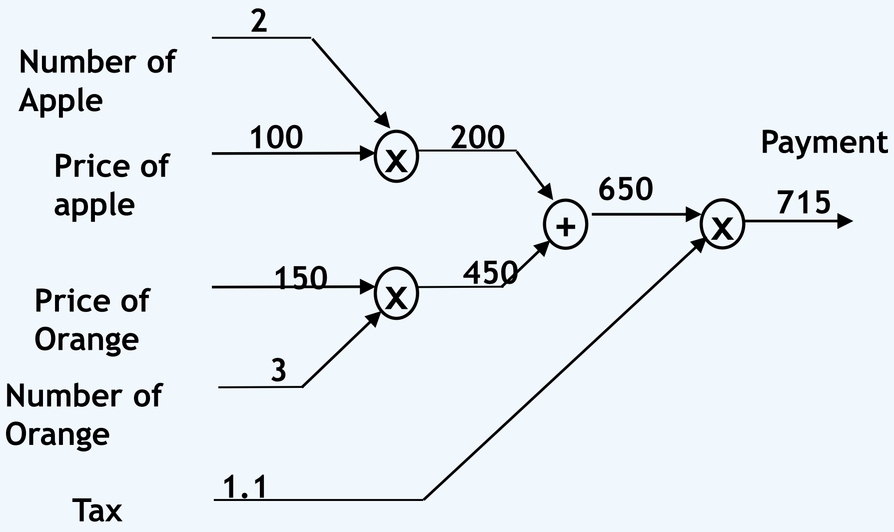
- Implementation for the chain rule in a network.



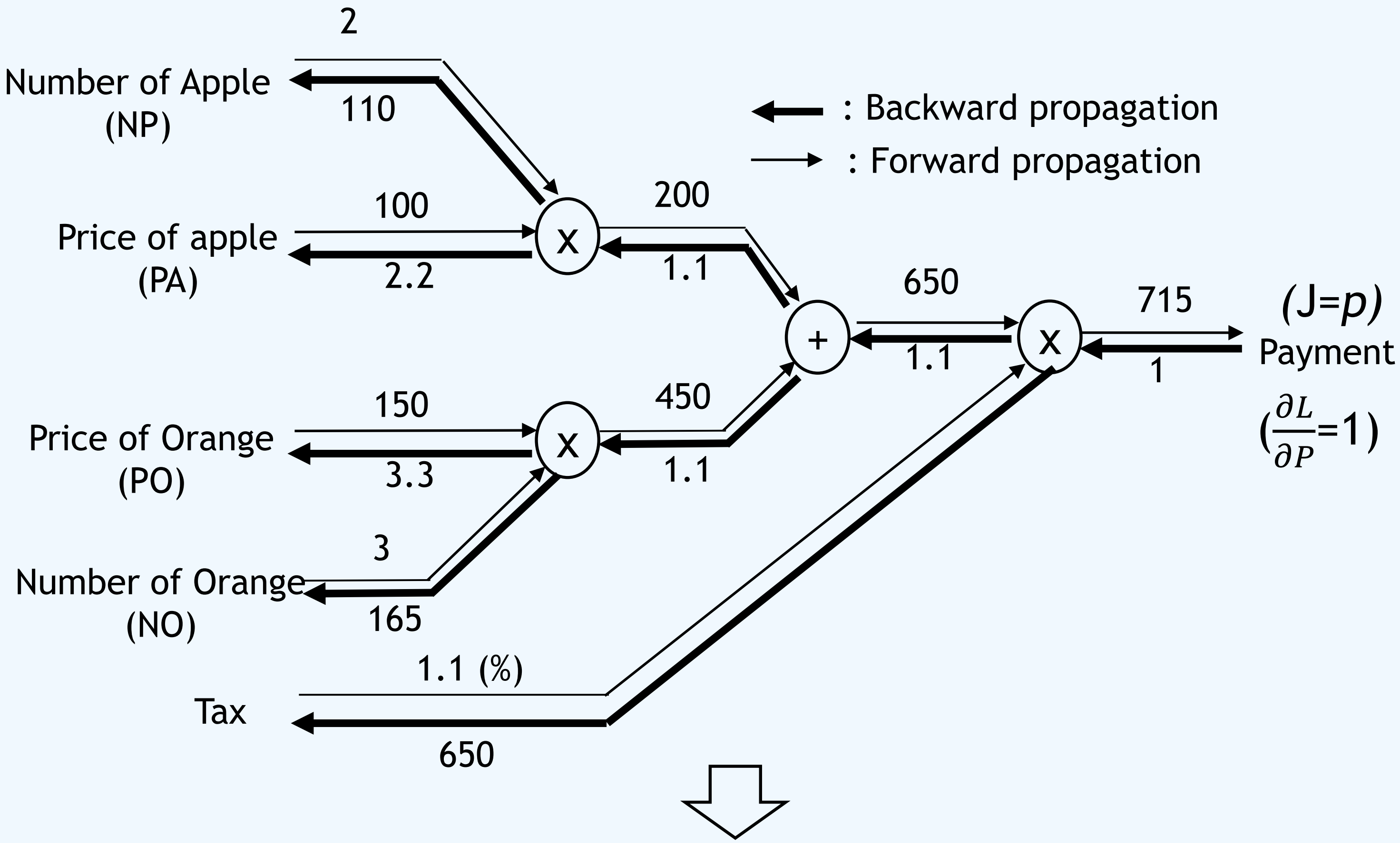




- Buy two apples of 100 won per one, three oranges of 150 won per one. Sales tax is 10% .

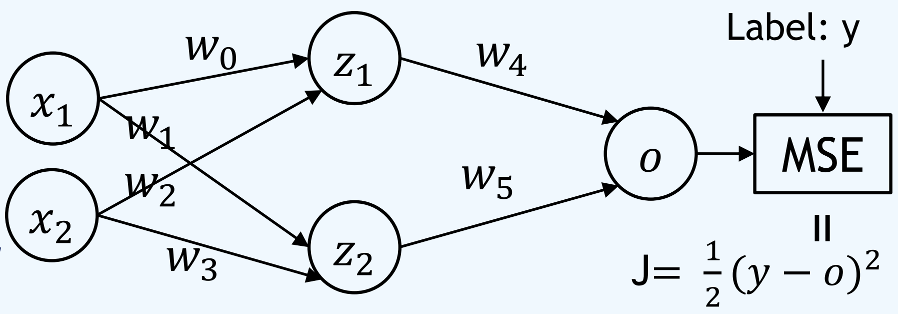


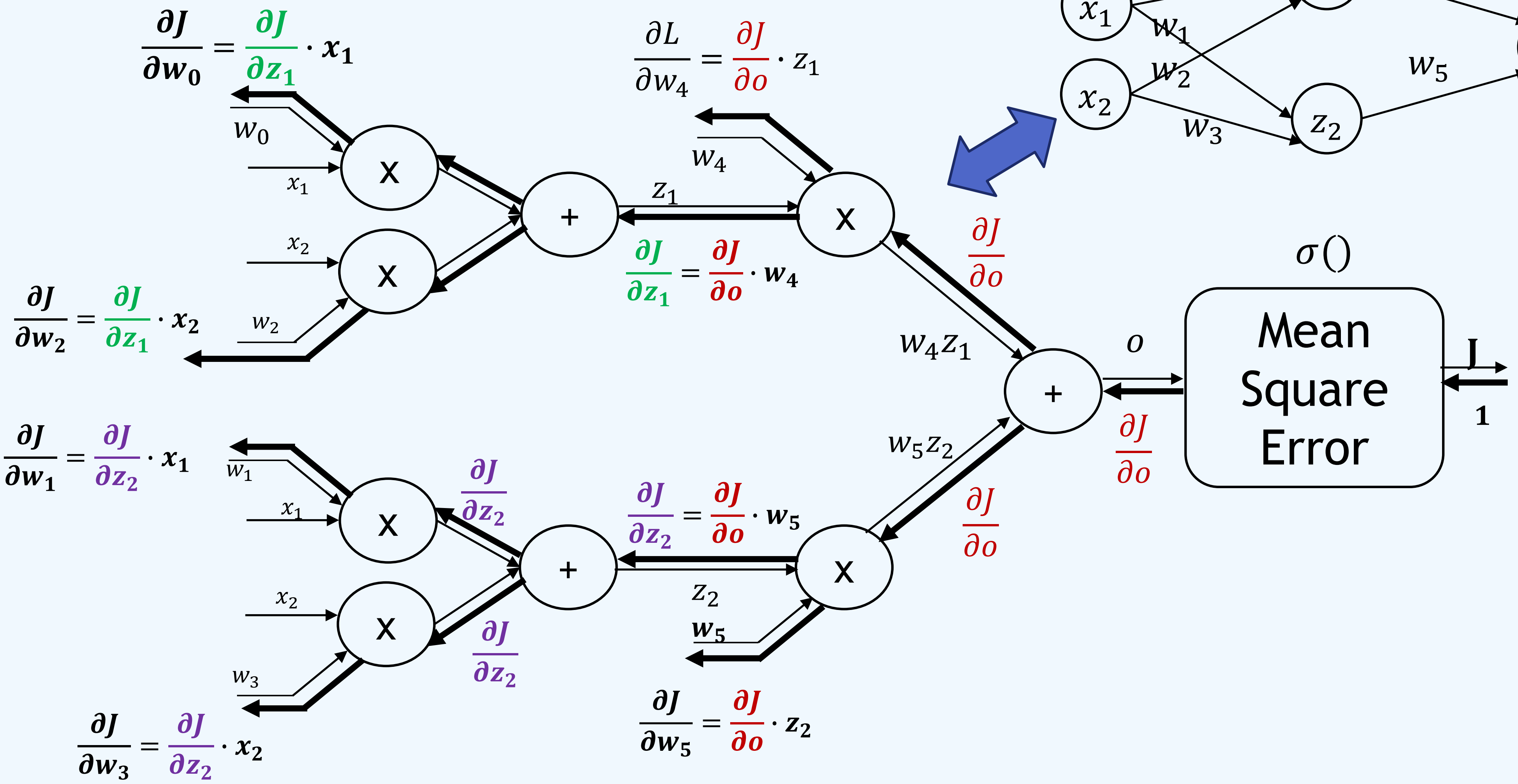
- Decide the price change with the changes of each item.

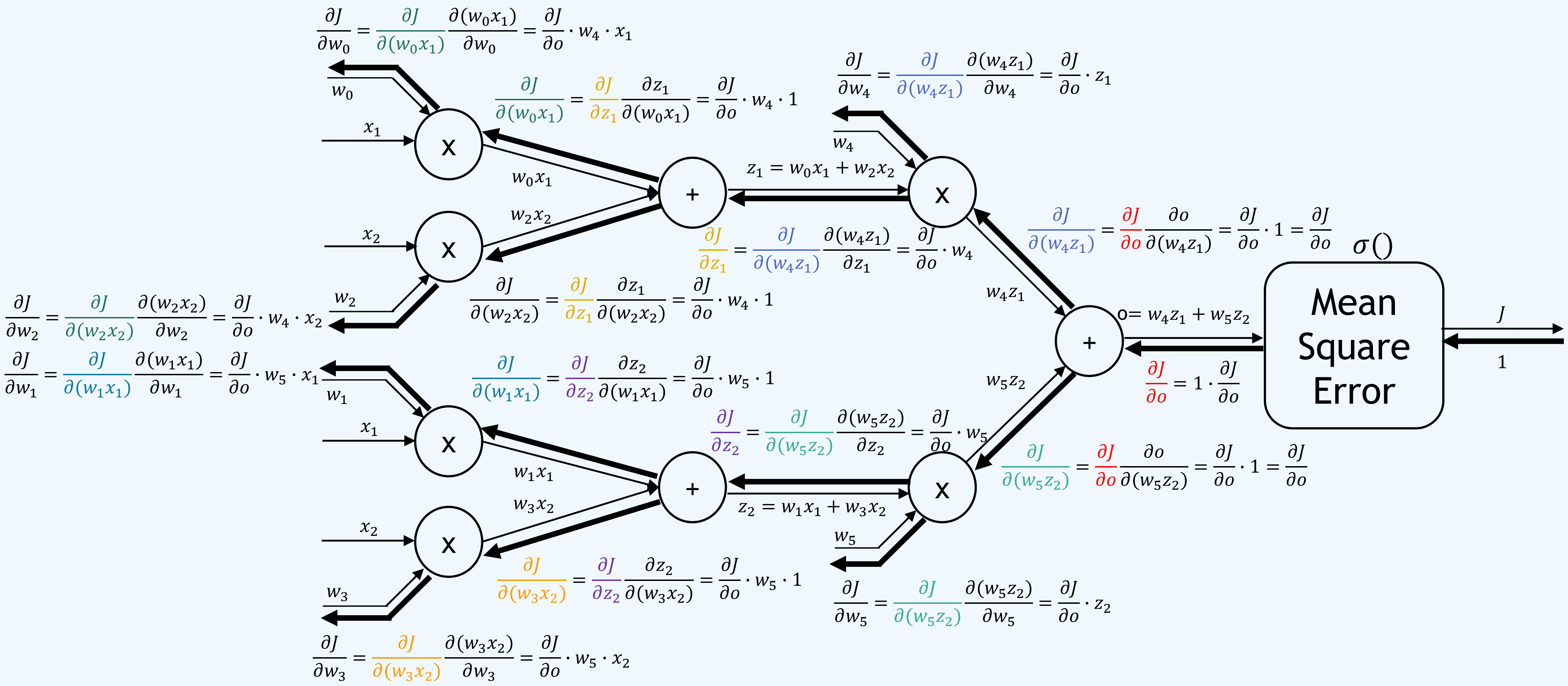


$$\Delta P = 110{*}\Delta NP + 2.2{*}\Delta PA + 3.3{*}\Delta PO + 165{*}\Delta NO + 6.5{*}\Delta TAX$$

- Gradient calculation with chain rule





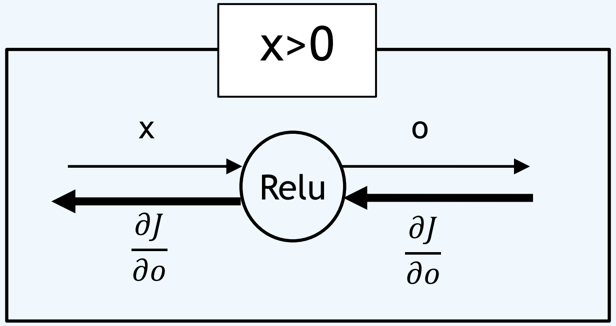


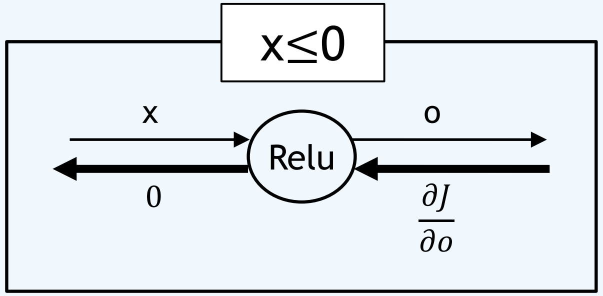
$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial (w_0x_1)} \frac{\partial (w_0x_1)}{\partial w_0} = \frac{\partial J}{\partial o} \cdot w_4 \cdot x_1$
$\frac{\partial J}{\partial (w_0x_1)} = \frac{\partial J}{\partial z_1} \frac{\partial z_1}{\partial (w_0x_1)} = \frac{\partial J}{\partial o} \cdot w_4 \cdot 1$
$\frac{\partial J}{\partial w_4} = \frac{\partial J}{\partial (w_4z_1)} \frac{\partial (w_4z_1)}{\partial w_4} = \frac{\partial J}{\partial o} \cdot z_1$
w_0
x_1
x
w_0x_1
w_2x_2
+
$z_1 = w_0x_1 + w_2x_2$
w_4
x
x_2
x
w_2
$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial (w_4z_1)} \frac{\partial (w_4z_1)}{\partial z_1} = \frac{\partial J}{\partial o} \cdot w_4$
$\frac{\partial J}{\partial (w_4z_1)} = \frac{\partial J}{\partial o} \frac{\partial o}{\partial (w_4z_1)} = \frac{\partial J}{\partial o} \cdot 1 = \frac{\partial J}{\partial o}$
$\sigma()$
$\frac{\partial J}{\partial (w_2x_2)} = \frac{\partial J}{\partial z_1} \frac{\partial z_1}{\partial (w_2x_2)} = \frac{\partial J}{\partial o} \cdot w_4 \cdot 1$
$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial (w_2x_2)} \frac{\partial (w_2x_2)}{\partial w_2} = \frac{\partial J}{\partial o} \cdot w_4 \cdot x_2$
w_4z_1
$o = w_4z_1 + w_5z_2$
+
Mean Square Error
J
1
$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial (w_1x_1)} \frac{\partial (w_1x_1)}{\partial w_1} = \frac{\partial J}{\partial o} \cdot w_5 \cdot x_1$
$\frac{\partial J}{\partial (w_1x_1)} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial (w_1x_1)} = \frac{\partial J}{\partial o} \cdot w_5 \cdot 1$
$\frac{\partial J}{\partial o} = 1 \cdot \frac{\partial J}{\partial o}$
w_5z_2
w_1
x_1
x
$\frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial (w_5z_2)} \frac{\partial (w_5z_2)}{\partial z_2} = \frac{\partial J}{\partial o} \cdot w_5$
$\frac{\partial J}{\partial (w_5z_2)} = \frac{\partial J}{\partial o} \frac{\partial o}{\partial (w_5z_2)} = \frac{\partial J}{\partial o} \cdot 1 = \frac{\partial J}{\partial o}$
w_1x_1
w_3x_2
+
x
$z_2 = w_1x_1 + w_3x_2$
x_2
x
w_5
w_3
$\frac{\partial J}{\partial (w_3x_2)} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial (w_3x_2)} = \frac{\partial J}{\partial o} \cdot w_5 \cdot 1$
$\frac{\partial J}{\partial w_5} = \frac{\partial J}{\partial (w_5z_2)} \frac{\partial (w_5z_2)}{\partial w_5} = \frac{\partial J}{\partial o} \cdot z_2$
$\frac{\partial J}{\partial w_3} = \frac{\partial J}{\partial (w_3x_2)} \frac{\partial (w_3x_2)}{\partial w_3} = \frac{\partial J}{\partial o} \cdot w_5 \cdot x_2$

 Backward propagation –Activation functions in hidden layer

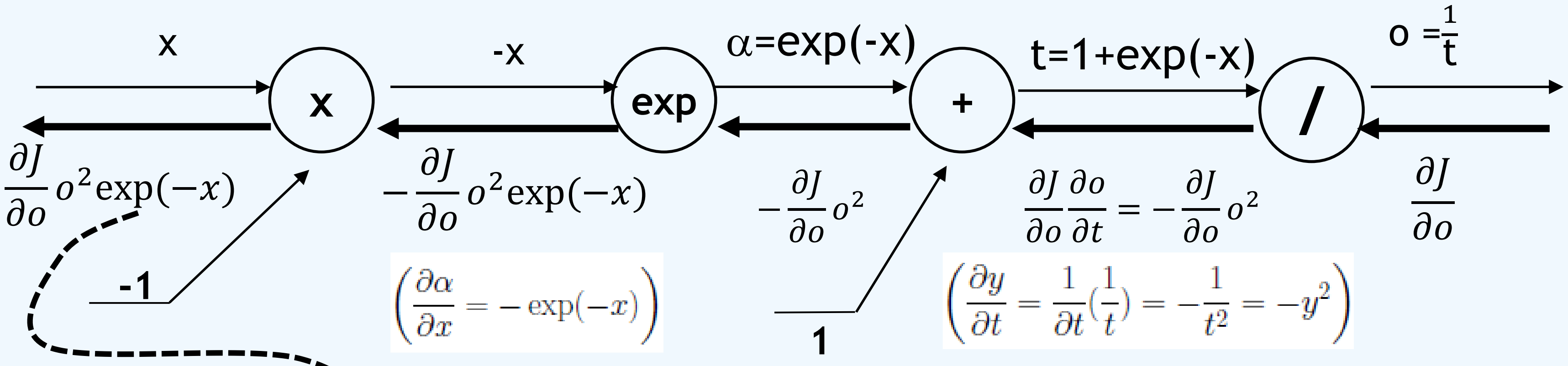
- Backward propagation of ReLU function

$$h(x) = \begin{cases} x \ (x > 0) \\ 0 \ (x \le 0) \end{cases} = \max(0, x)$$

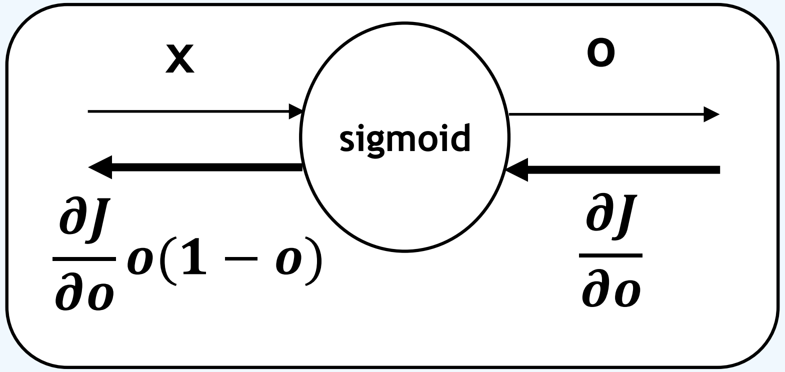




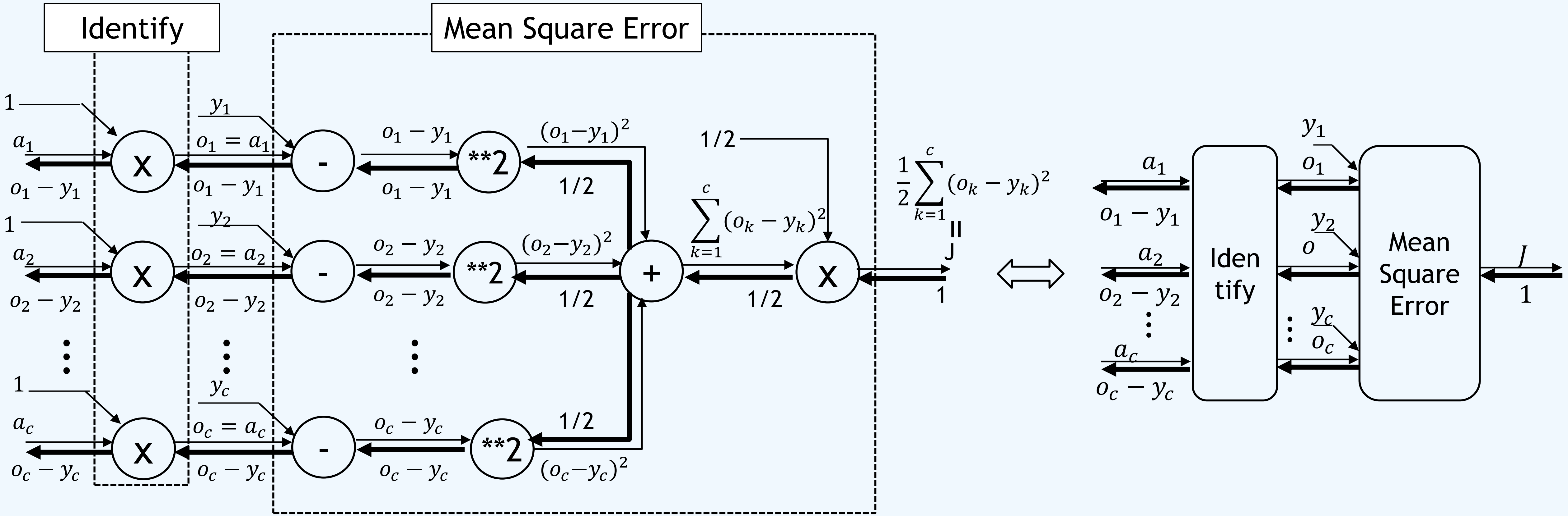
- Backward propagation of Sigmoid function $h(x) = \frac{1}{1 + e^{-x}}$



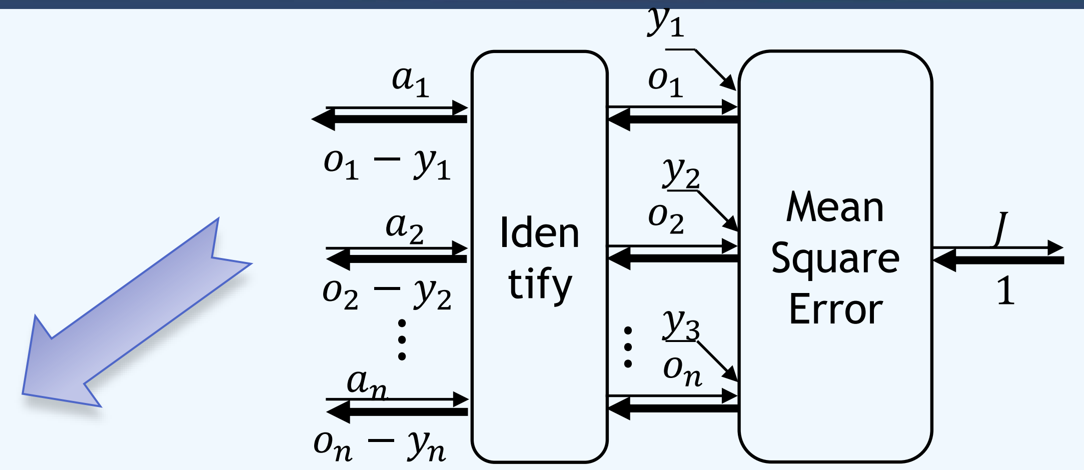
$$\begin{aligned} &\frac{\partial J}{\partial o} o^2 \exp(-x) \\ &= \frac{\partial J}{\partial o} \frac{1}{(1 + \exp(-\mathrm{x}))^2} \exp(-\mathrm{x}) \\ &= \frac{\partial J}{\partial o} \frac{1}{1 + \exp(-\mathrm{x})} \frac{\exp(-\mathrm{x})}{1 + \exp(-\mathrm{x})} \\ &= \frac{\partial J}{\partial o} o(1 - o) \end{aligned}$$

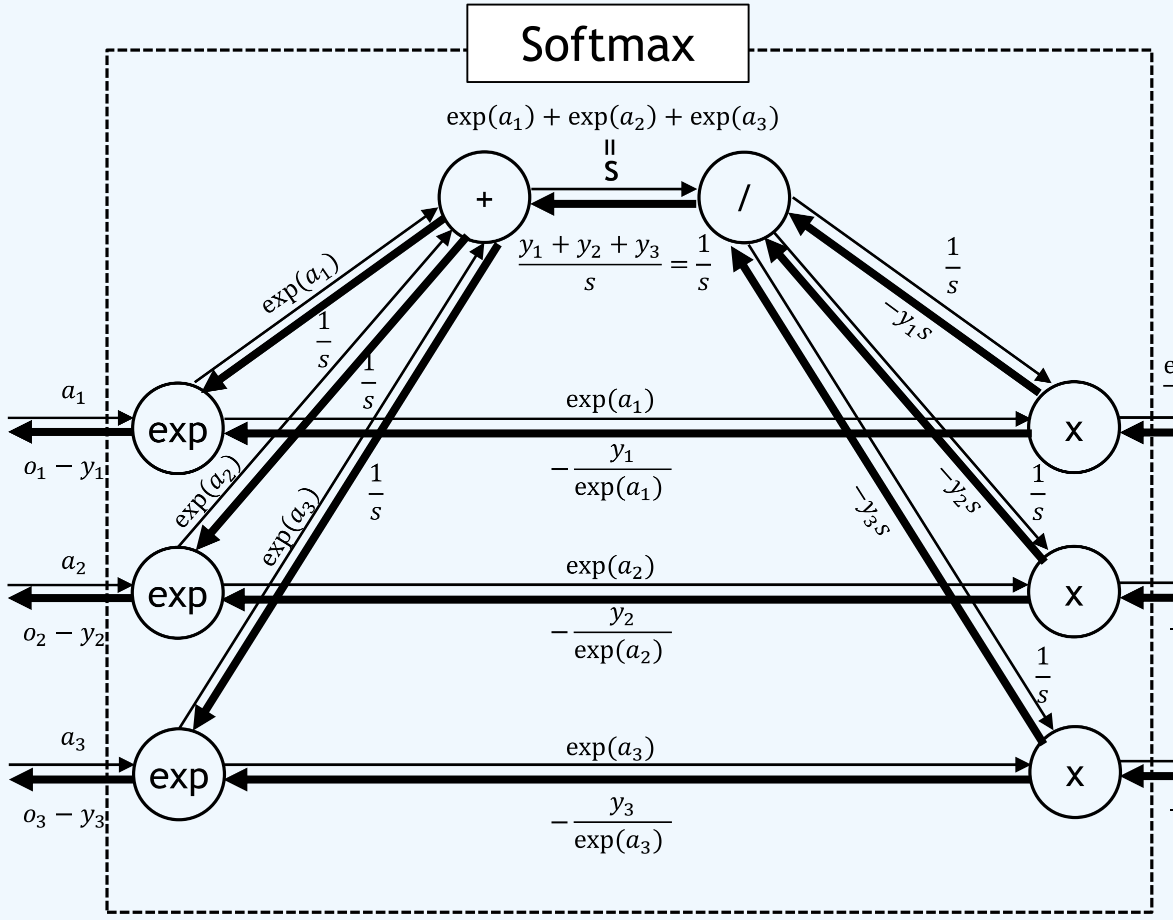


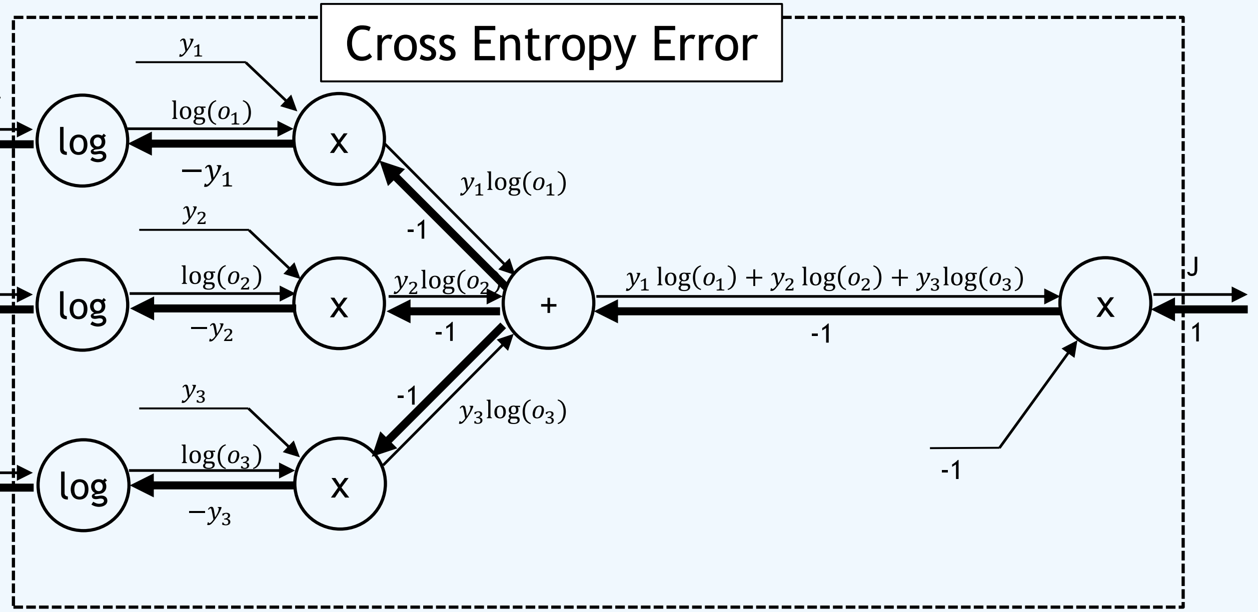
- Backward propagation of Identify function and Mean Square Error



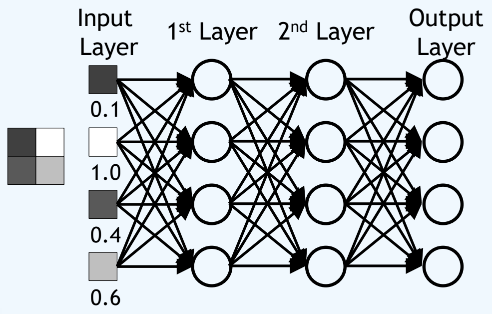
- **Backward propagation of Softmax and Cross Entropy Error**

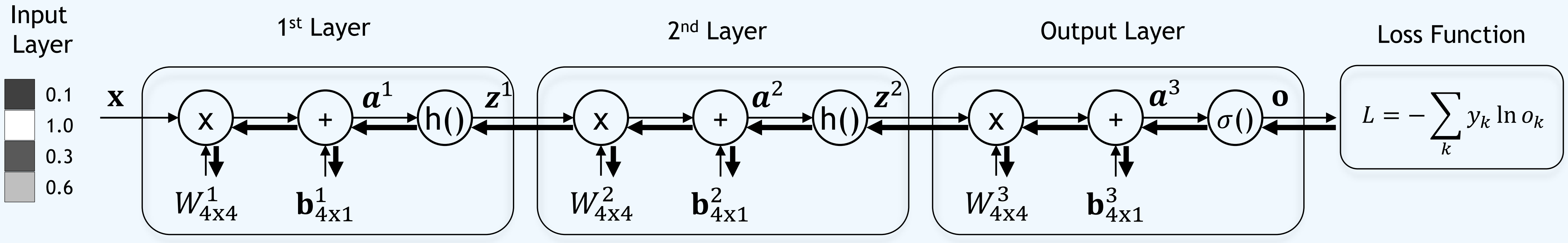






- Back propagation learning for image direction detection network
  - Loss function : Cross entropy error
  - Learning rate : η = 0.5
  - Optimizer : Gradient Descent





- Weight update :

$$W^k(i,j) = W^k(i,j) - \eta \times \frac{\partial L}{\partial a^k(i)} \times z^{k-1}(j)$$

$$b^k(i) = b^k(i) - \eta \times \frac{\partial L}{\partial a^k(i)}$$

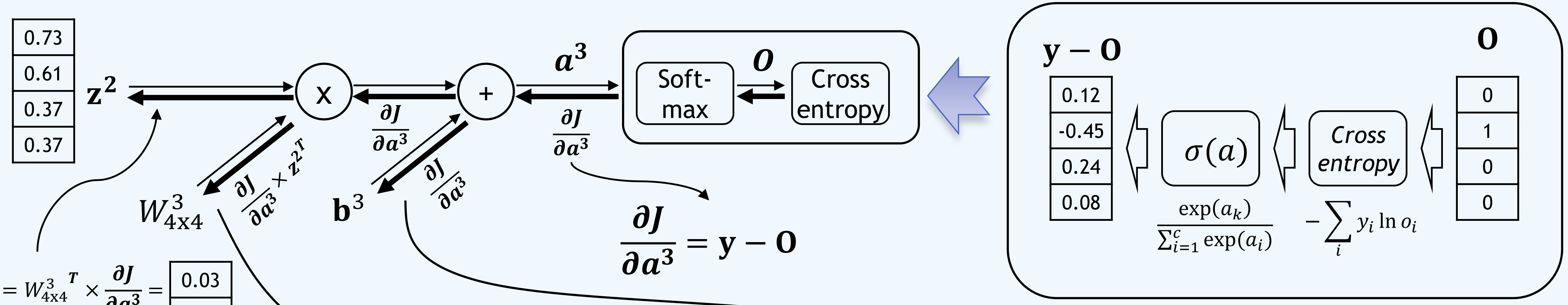
(From chain rule)

(From chain rule)

$(z^3 = y, z^0 = x)$

-> ak가 아니라 bk

- The Output Layer



$$\frac{\partial J}{\partial Z^2} = {W^3_{4\text{x}4}}^T \times \frac{\partial J}{\partial a^3} = \begin{bmatrix} 0.03 \\ -0.54 \\ -0.20 \\ -0.39 \end{bmatrix}$$

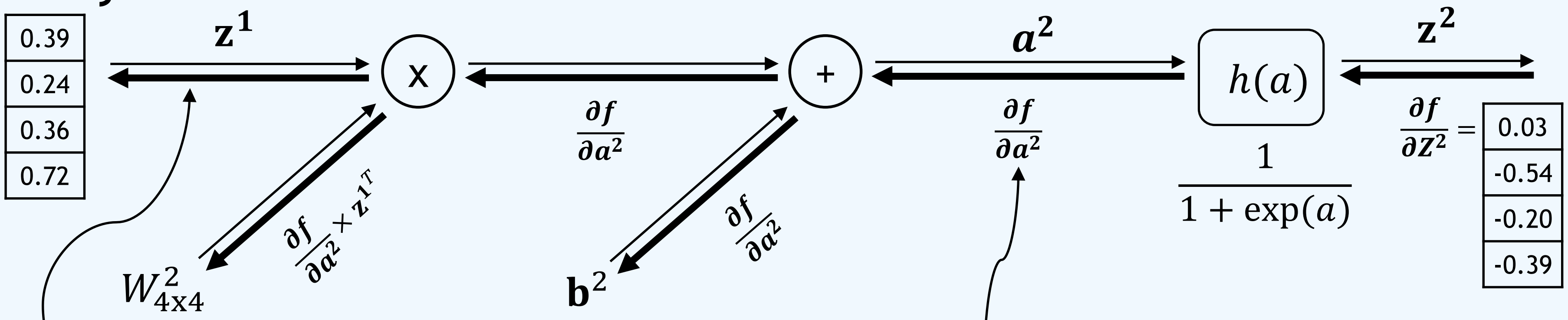
$$\frac{\partial J}{\partial W^3_{4x4}} = \frac{\partial J}{\partial a^3} \times z^{2^T}$$

$$\frac{\partial J}{\partial b^3} = \frac{\partial J}{\partial a^3}$$

$$\underset{W^3_{4\text{x}4}}{\begin{bmatrix} -0.80 & 0.34 & 0.35 & -0.63 \\ 0.10 & 1.10 & 0.90 & 0.59 \\ 0.62 & -0.50 & 0.18 & -0.35 \\ -0.94 & -0.75 & 0.79 & -0.17 \end{bmatrix}} = \underset{W^3_{4\text{x}4}}{\begin{bmatrix} -0.75 & 0.38 & 0.37 & -0.61 \\ -0.06 & 0.96 & 0.82 & 0.51 \\ 0.71 & -0.43 & 0.22 & -0.31 \\ -0.91 & -0.73 & 0.80 & -0.16 \end{bmatrix}} - \eta \times \underset{\mathbf{y}-\mathbf{o}}{\begin{bmatrix} 0.12 \\ -0.45 \\ 0.24 \\ 0.08 \end{bmatrix}} \times \underset{\mathbf{z}^{2T}}{\begin{bmatrix} 0.73 & 0.61 & 0.37 & 0.37 \end{bmatrix}}$$

$$\underset{\mathbf{b}^3}{\begin{bmatrix} -0.09 \\ -0.25 \\ -0.11 \\ -0.02 \end{bmatrix}} = \underset{\mathbf{b}^3}{\begin{bmatrix} 0.12 \\ -0.45 \\ 0.24 \\ 0.08 \end{bmatrix}} - \eta \times \underset{\mathbf{y}-\mathbf{o}}{\begin{bmatrix} 0.12 \\ -0.45 \\ 0.24 \\ 0.08 \end{bmatrix}}$$

- The 2nd Layer



$$\frac{\partial f}{\partial Z^1} = {W^2_{4x4}}^T \times \frac{\partial f}{\partial a^2} = \begin{bmatrix} 0.05 \\ -0.21 \\ 0.09 \\ -0.03 \end{bmatrix}$$

$$\frac{\partial f}{\partial a^2} = \frac{\partial f}{\partial Z^2} \circ \mathbf{z}^2 \circ (\mathbf{1} - \mathbf{z}^2) = \begin{bmatrix} 0.01 \\ -0.31 \\ -0.05 \\ -0.09 \end{bmatrix}$$

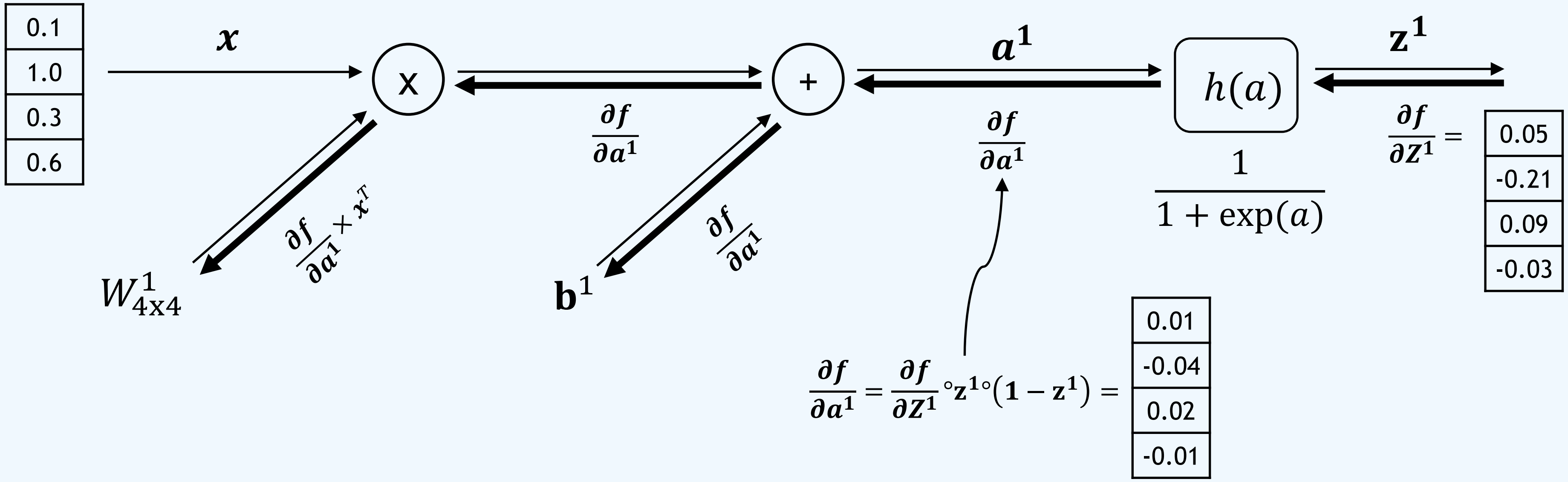
Hadamard product

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \circ \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} a \times e \\ b \times f \\ c \times g \\ d \times h \end{bmatrix}$$

$$\underset{W^2_{4x4}}{\begin{bmatrix} -0.21 & -0.34 & -0.75 & -0.78 \\ -0.61 & 0.63 & -0.52 & -0.14 \\ 0.28 & 1.01 & -0.94 & 0.79 \\ 0.27 & 0.97 & 0.23 & 0.13 \end{bmatrix}} = \underset{W^2_{4x4}}{\begin{bmatrix} -0.21 & -0.34 & -0.75 & -0.78 \\ -0.64 & -0.61 & -0.54 & -0.19 \\ 0.27 & 1.00 & -0.95 & 0.77 \\ 0.25 & 0.96 & 0.21 & 0.10 \end{bmatrix}} - \eta \times \underset{\frac{\partial f}{\partial a^2}}{\begin{bmatrix} 0.01 \\ -0.31 \\ -0.05 \\ -0.09 \end{bmatrix}} \times \underset{{\mathbf{z}^1}^T}{\begin{bmatrix} 0.39 & 0.24 & 0.36 & 0.72 \end{bmatrix}}$$

$$\underset{\mathbf{b}^2}{\begin{bmatrix} -0.01 \\ 0.03 \\ 0.01 \\ 0.10 \end{bmatrix}} = \underset{\mathbf{b}^2}{\begin{bmatrix} -0.01 \\ -0.03 \\ -0.01 \\ 0.05 \end{bmatrix}} - \eta \times \underset{\frac{\partial f}{\partial a^2}}{\begin{bmatrix} 0.01 \\ -0.31 \\ -0.05 \\ -0.09 \end{bmatrix}}$$

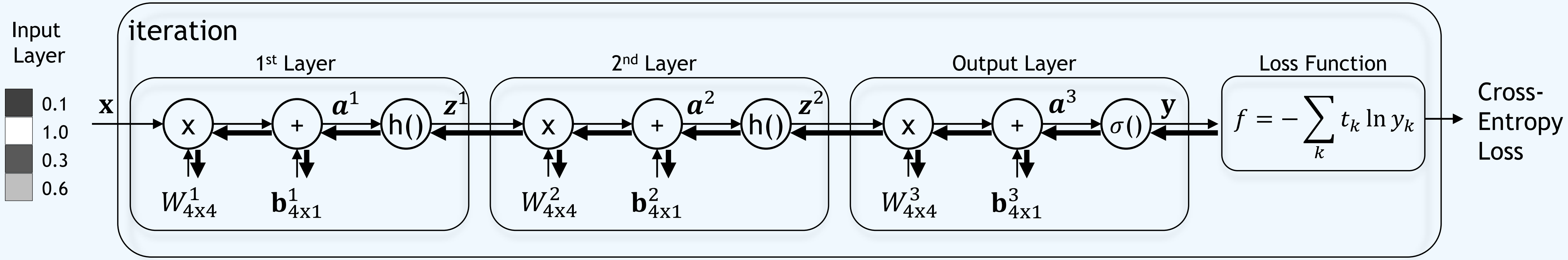
- The 1st Layer



$$\overset{W^1_{4\mathrm{x}4}}{\begin{bmatrix} -0.82 & 0.70 & -0.55 & -0.08 \\ 0.19 & 0.95 & 0.09 & 0.29 \\ -0.52 & -0.03 & 0.52 & 0.82 \\ 0.68 & -0.56 & -0.30 & -0.68 \end{bmatrix}} = \overset{W^1_{4\mathrm{x}4}}{\begin{bmatrix} -0.82 & 0.71 & -0.55 & -0.08 \\ 0.19 & 0.93 & 0.07 & 0.28 \\ -0.52 & -0.02 & 0.52 & 0.83 \\ 0.68 & -0.56 & -0.30 & -0.68 \end{bmatrix}} - \eta \times \overset{\frac{\partial f}{\partial \boldsymbol{a}^1}}{\begin{bmatrix} 0.01 \\ -0.04 \\ 0.02 \\ -0.01 \end{bmatrix}} \times \overset{\boldsymbol{x}^T}{\begin{bmatrix} 0.1 & 1.0 & 0.3 & 0.6 \end{bmatrix}}$$

$$\overset{\mathbf{b}^1}{\begin{bmatrix} 0.01 \\ 0.03 \\ -0.02 \\ 0.04 \end{bmatrix}} = \overset{\mathbf{b}^1}{\begin{bmatrix} 0.02 \\ 0.01 \\ -0.01 \\ 0.04 \end{bmatrix}} - \eta \times \overset{\frac{\partial f}{\partial \boldsymbol{a}^1}}{\begin{bmatrix} 0.01 \\ -0.04 \\ 0.02 \\ -0.01 \end{bmatrix}}$$

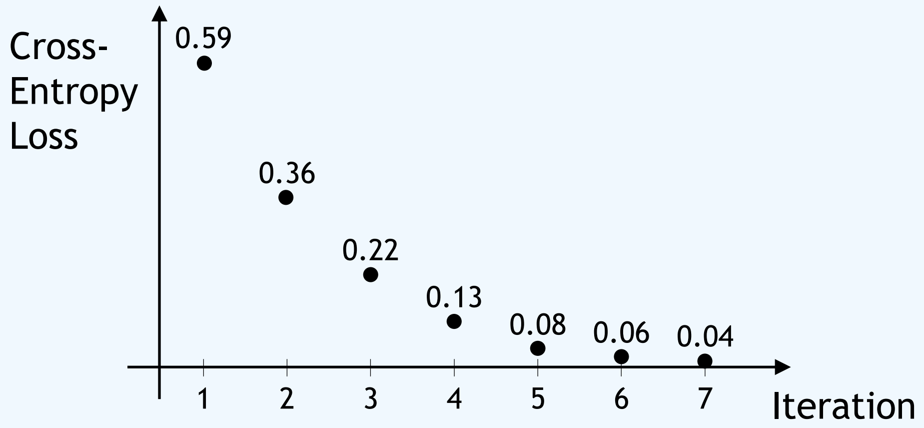
- Training : 7 iteration



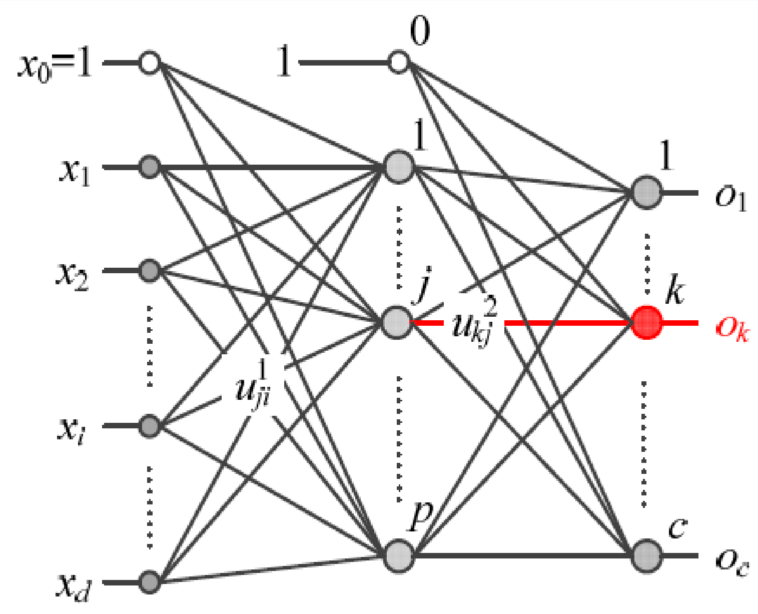
- Prediction(y)

Iteration	1	2	3	4	5	6	7	t
Predic-tion	0.12	0.09	0.06	0.04	0.03	0.02	0.01	0
	0.55	0.70	0.81	0.88	0.92	0.94	0.96	1
	0.24	0.15	0.09	0.05	0.03	0.02	0.01	0
	0.08	0.06	0.04	0.03	0.02	0.02	0.01	0

- Loss Decreasing



5. MLP Training [18/18] : Computation Load



- Num. of input nodes (features) : d
- Num. of internal nodes : p
- Num. of output nodes : c

- Forward Computation
  - Addition : dp+pc
  - Multiplication : dp+pc
- Backward Propagation (BP) Computation
  - Addition : c+2cp+dp
  - Multiplication : c+3cp+2dp
- BP needs only 1.5~2 times more computation than Forward.
- Learning computation : O((dp+pc)nq)
  where n : # of samples, # of iterations : q