Source:

- **Lectures on 'Machine Learning', Prof. Andrew Ng, Stanford University**

20170534

Introduction to Machine Learning

Lecture #10 Weight update in ANN (Learning)

국립
한밭대학교
HANBAT NATIONAL UNIVERSITY

HANBAT NATIONAL UNIVERSITY ELECTRONIC ENGINEERING

Instructor: Jeon, Seung-Bae

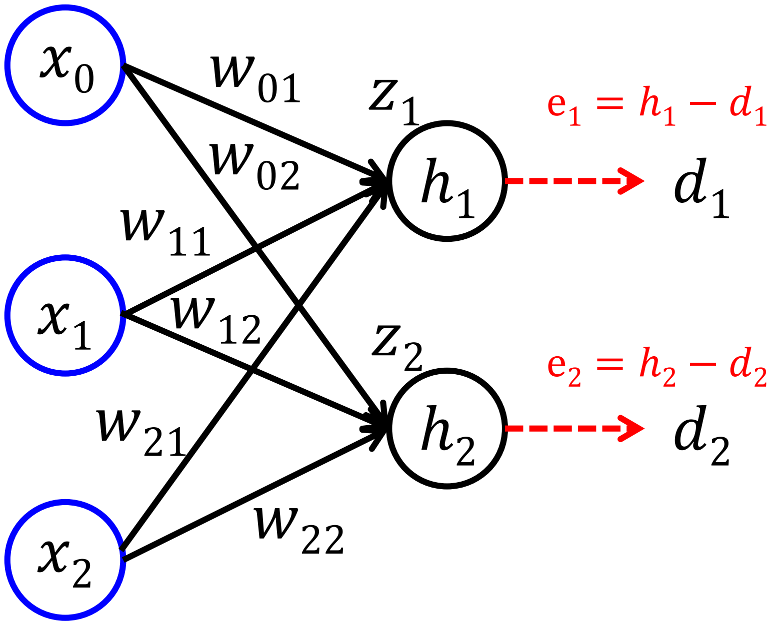
Assistant Professor
Hanbat National University
Department of Electronic Engineering

Contents

Contents

Weight update (training) in the single layer ANN - 1

Neural networks also use gradient descent for parameter (weight) update.



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}}$$

What is the cost function J?

Let's deal with the most simple case.

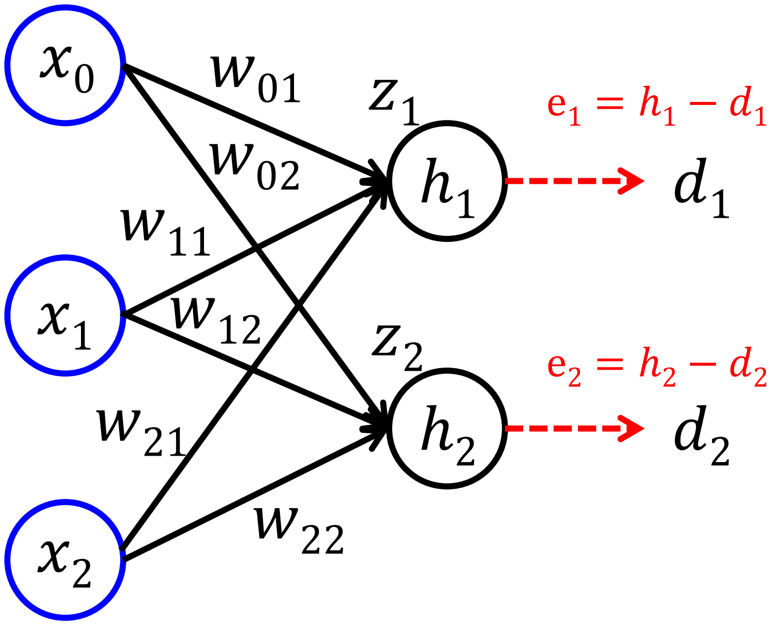
1) activation function = sigmoid

2) cost function = sum of squared errors

$$J = \sum_{n=1}^{k} {e_n}^2 = \sum_{n=1}^{k} \frac{1}{2}(h_n - dn)^2$$

Weight update (training) in the single layer ANN - 2

Neural networks also use gradient descent for parameter (weight) update.



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}} \quad , \ \alpha = \text{learning rate}$$

$$J = \sum_{n=1}^{k} \frac{1}{2} {e_n}^2 = \sum_{n=1}^{k} \frac{1}{2} (h_n - d_n)^2$$

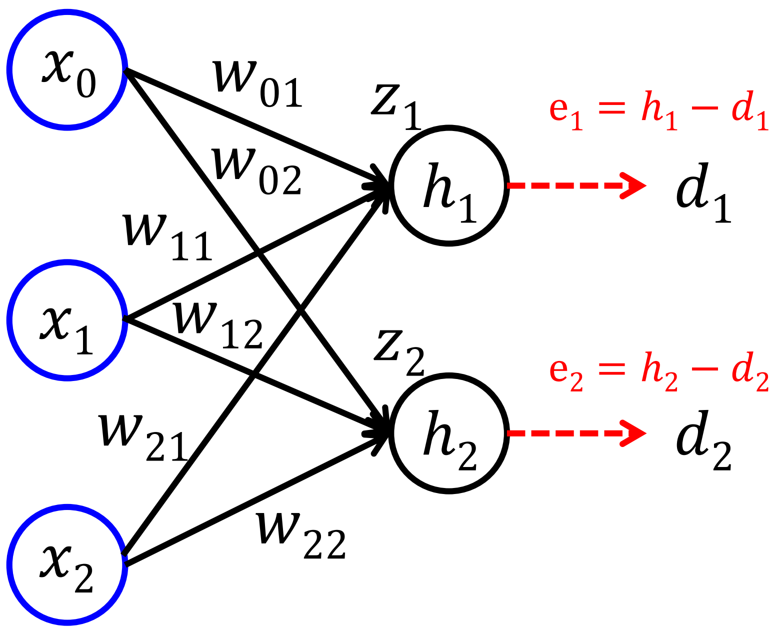
$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k} \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} \rightarrow \text{(step by step!)}$$

① ② ③

① $\frac{\partial J}{\partial h_k} = \frac{\partial \sum_{n=1}^{k} \frac{1}{2} (h_n - d_n)^2}{\partial h_k} = \frac{\partial \frac{1}{2} (h_k - d_k)^2}{\partial h_k} = (h_k - d_k)$

Weight update (training) in the single layer ANN - 3

Neural networks also use gradient descent for parameter (weight) update.



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}} \quad , \quad \alpha = \text{learning rate}$$

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k} \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} \rightarrow \text{(step by step!)}$$

② $$\frac{\partial h_k}{\partial z_k} = \frac{\partial g(z_k)}{\partial z_k} = \frac{\partial}{\partial z_k}\left(\frac{1}{1+e^{-z_k}}\right) = \frac{\partial}{\partial z}(1 + e^{-z})^{-1}$$

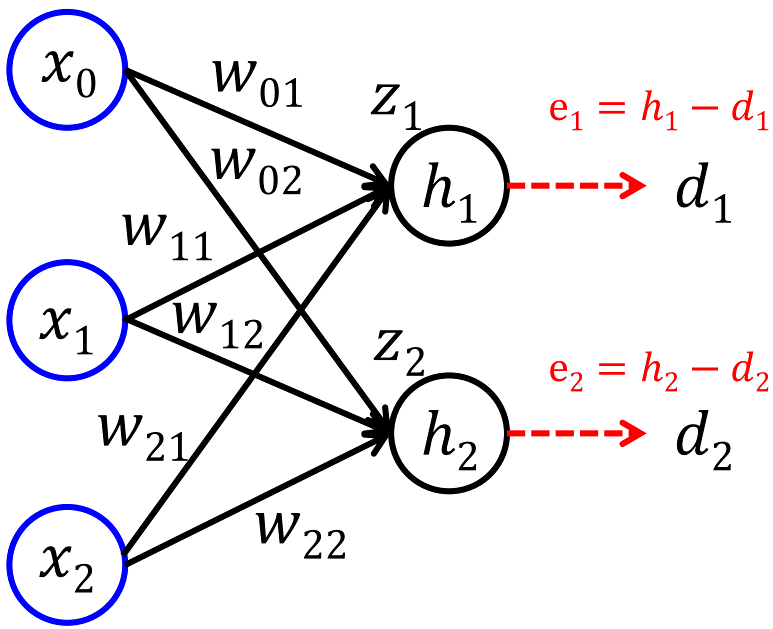
$$= -(1 + e^{-z})^{-2} \frac{\partial}{\partial z}(1 + e^{-z}) = -(1 + e^{-z})^{-2} \frac{\partial}{\partial z}(e^{-z})$$

$$= (1 + e^{-z})^{-2} (e^{-z}) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= h_k(1 - h_k)$$

Weight update (training) in the single layer ANN - 4

- **Neural networks also use gradient descent for parameter (weight) update.**



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}} \text{ , } \alpha = \text{learning rate}$$

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k} \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} \rightarrow \text{(step by step!)}$$

③ $$\frac{\partial z_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \left(\sum_{i=0}^{n} x_i \cdot w_{ik} \right)$$

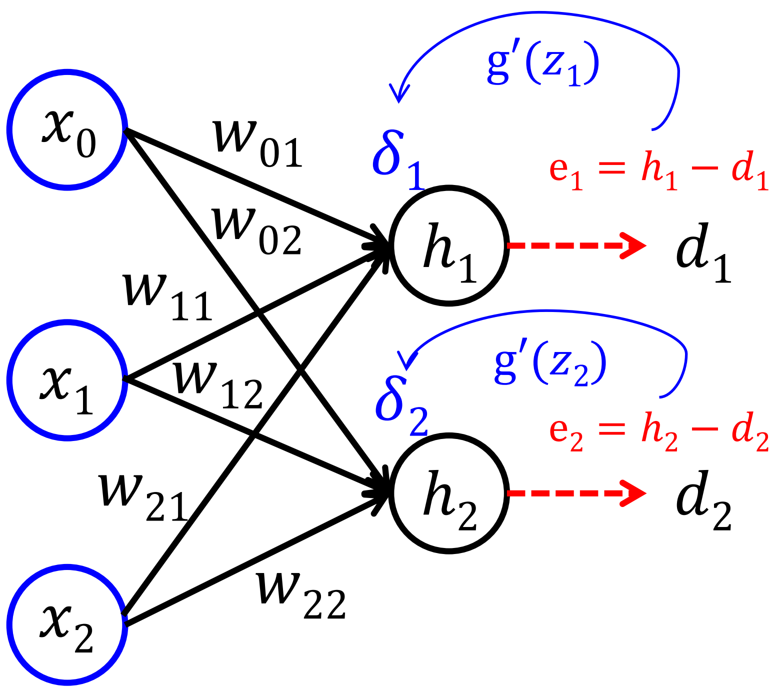
$$= \frac{\partial}{\partial w_{jk}} (x_j \cdot w_{jk}) = x_j$$

$$\therefore \frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k} \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} = (h_k - d_k)\, h_k(1 - h_k)\, x_j$$

$$= e_k\, g(z_k)(1 - g(z_k))\, x_j$$

Weight update (training) summary

Neural networks also use gradient descent for parameter (weight) update.



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}}, \quad \alpha = \text{learning rate}$$

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k}\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = (h_k - d_k)\, h_k(1 - h_k)\, x_j$$

$$= e_k\, g(z_k)(1 - g(z_k))\, x_j$$

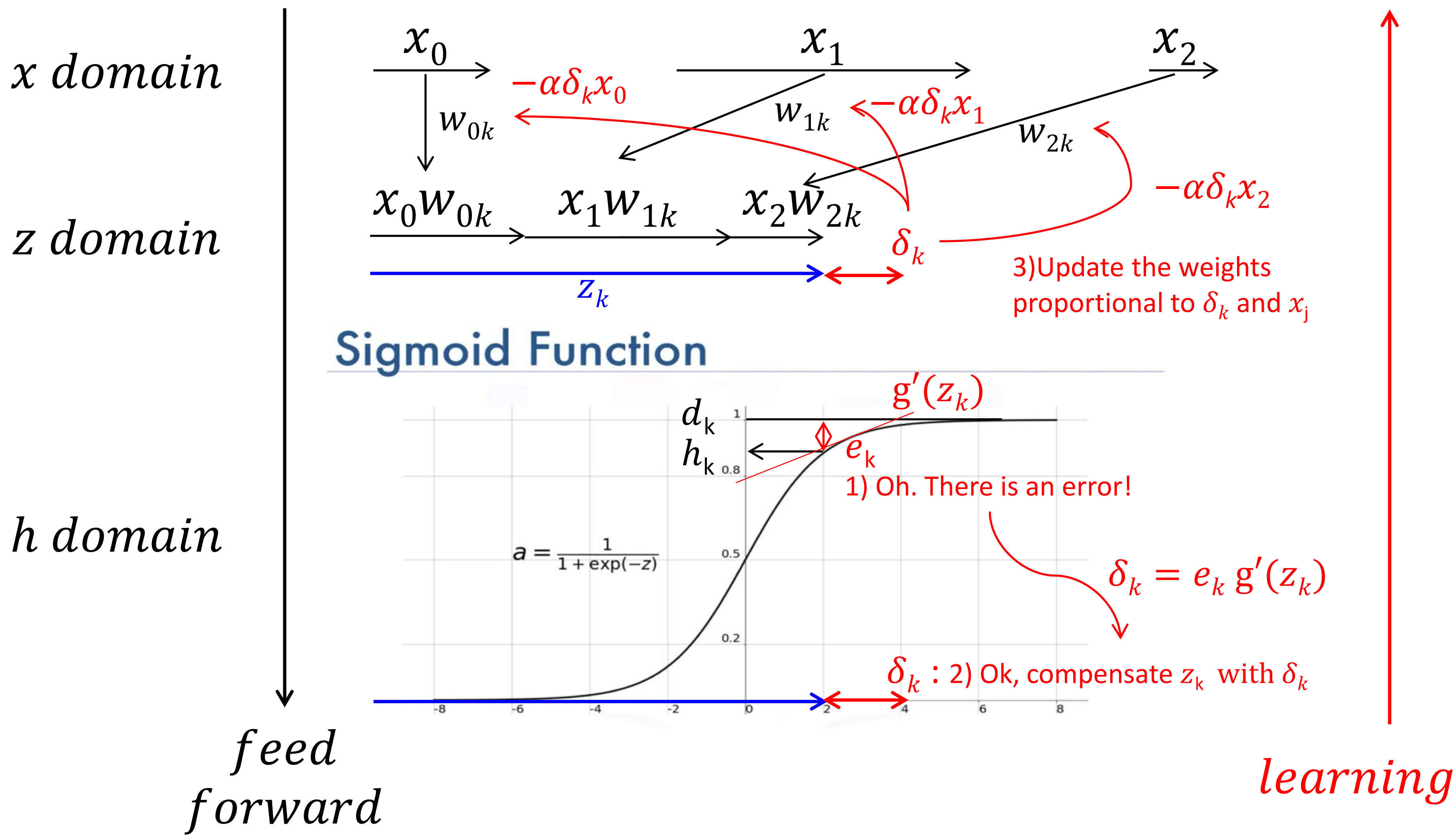
$$= e_k\, g'(z_k)\, x_j = \delta_k\, x_j$$

$$\rightarrow w_{jk} = w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}} = w_{jk} - \alpha \delta_k\, x_j$$

(this is called as the delta rule.)

Qualitative understanding

- **It's easier to understand if you think about the domain qualitatively.**

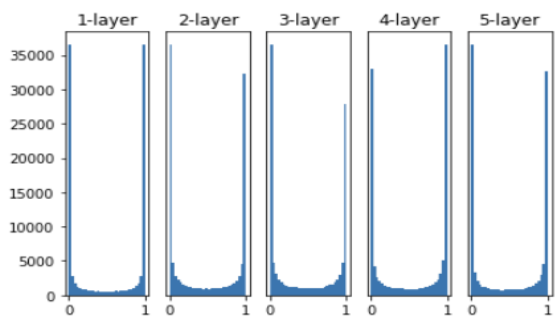


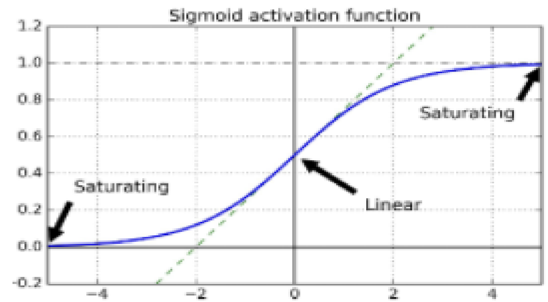
Process of the learning in the single layer ANN

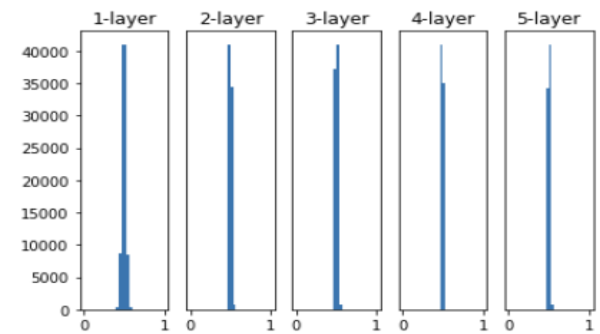
1) Initialize the weights value properly.
2) By entering the feature values of the training data into the neural network and feeding forward, obtain the output value h_k (this process is also called inference)
3) Calculate the error comparing the h_k to d_k
4) Calculate the delta based on the error
5) Calculate the amount of weight update.
6) Update all the weights
7) Repeat the above 2-6 for the entire training data (1 epoch)
8) Repeat the above epoch until the error becomes sufficiently small.

weight initialization techniques

It is not good if the variance is too large or too small.







Xavier initialization

- Xavier Normal Initialization

$$W \sim N(0, Var(W))$$

$$sigma = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

- Xavier Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \; +\sqrt{\frac{6}{n_{in} + n_{out}}})$$

```
w1 = np.random.normal(0,np.sqrt(2/(input_layer+hidden_layer)),size=(input_layer,hidden_layer))
w2 = np.random.normal(0,np.sqrt(2/(hidden_layer+output_layer)),size=(hidden_layer,output_layer))
```

He initialization (for ReLU)

- He Normal Initialization

$$W \sim N(0, Var(W))$$

$$sigma = \sqrt{\frac{2}{n_{in}}}$$

(n_{in} : 이전 layer(input)의 노드 수)

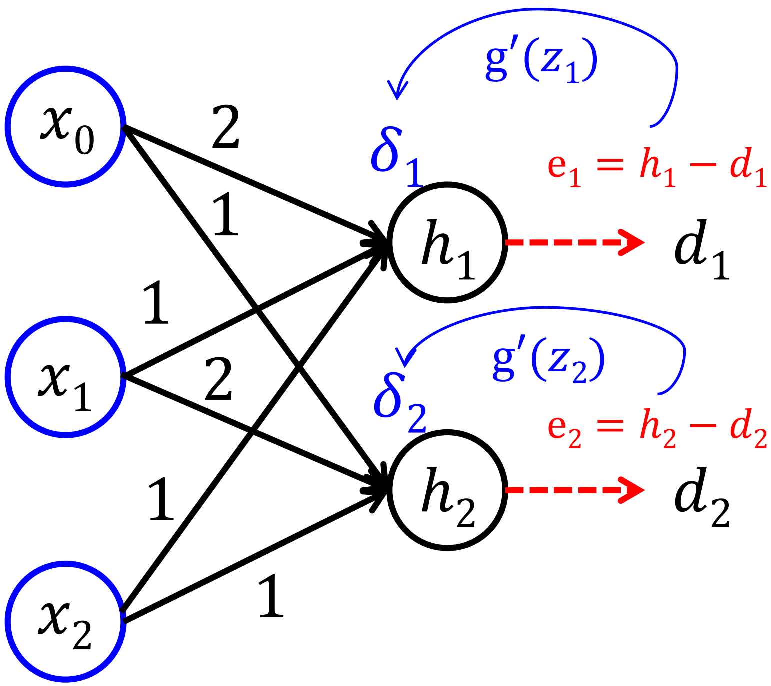
- He Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, \; +\sqrt{\frac{6}{n_{in}}})$$

(n_{in} : 이전 layer(input)의 노드 수)

Exercise

$x_0 = 1,\ x_1 = 1,\ x_2 = 0$ & d_1=1, d_2 = 0. Initial value of the weights are written in the figure.



x_0	x_1	x_2	d_1	d_2
1	1	0	1	0

w_{01}	w_{02}	w_{11}	w_{12}	w_{21}	w_{22}
2	1	1	2	1	1

z_1	z_2	h_1	h_2	e_1	e_2
3	3	0.952574	0.952574	-0.04743	0.952574

δ_1	δ_2	w_{01} update	w_{11} update	w_{21} update
-0.00214	0.043034	0.00214	0.00214	0

$-\alpha\delta_k x_j$

w_{02} update	w_{12} update	w_{22} update
-0.043034	-0.043034	0

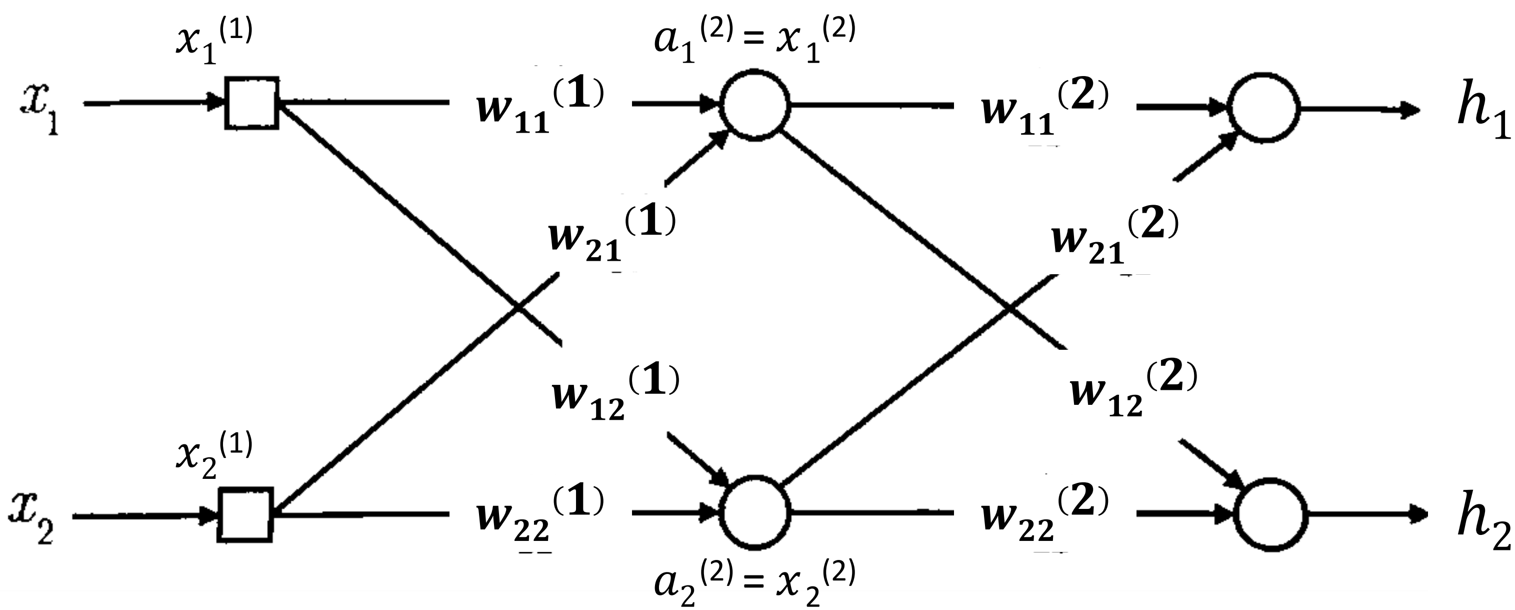
new w_{01}	new w_{02}	new w_{11}	new w_{12}	new w_{21}	new w_{22}
2.002143	0.956966	1.002143	1.956966	1	1

If there are multiple training examples, perform the above operation for each example.

Contents

Weight update in multi-layer ANN

Feedfoward (inference) in multi-layer ANN



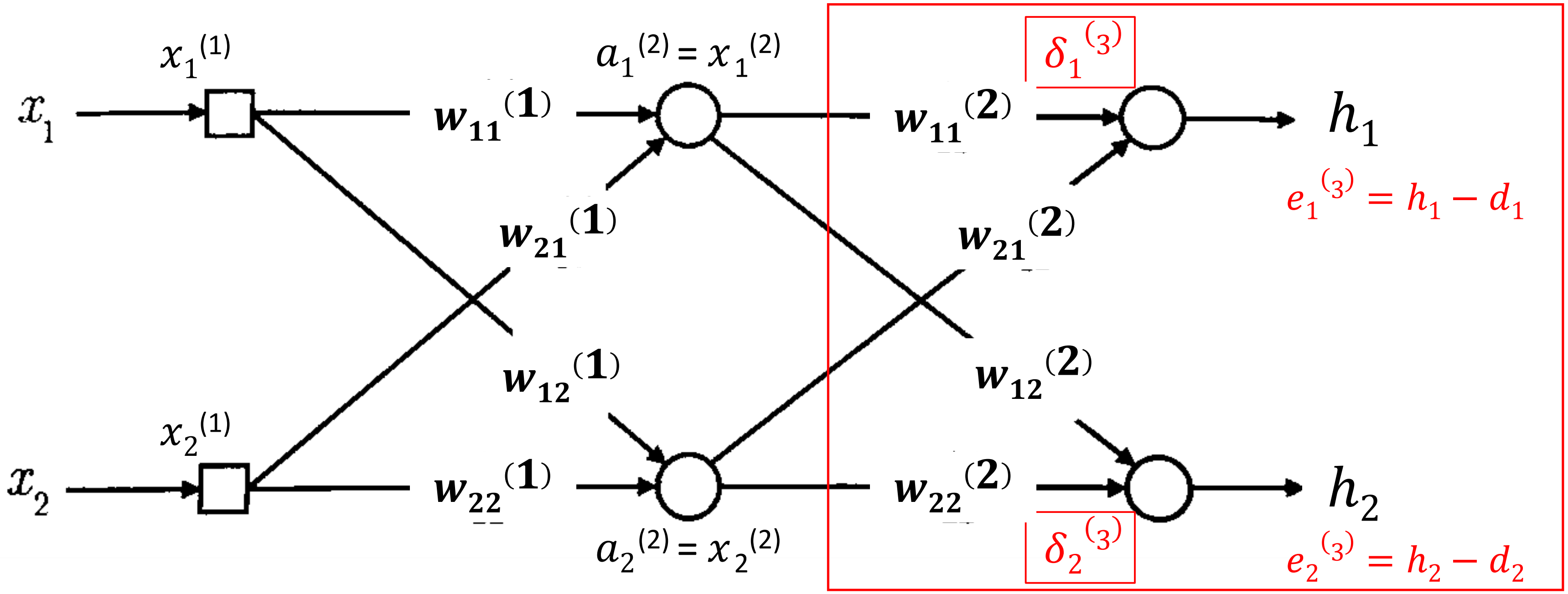
$$x = [x_1^{(1)}\ x_2^{(1)}] \quad w^{(1)} = \begin{bmatrix} w_{11}^{(1)} w_{12}^{(1)} \\ w_{21}^{(1)} w_{22}^{(1)} \end{bmatrix} \qquad x^{(2)} = [x_1^{(2)}\ x_2^{(2)}] \quad w^{(2)} = \begin{bmatrix} w_{11}^{(2)} w_{12}^{(2)} \\ w_{21}^{(2)} w_{22}^{(2)} \end{bmatrix}$$

$$z^{(2)} = x^{(1)} \cdot w^{(1)} = [z_1^{(2)}\ z_2^{(2)}] \qquad z^{(3)} = x^{(2)} \cdot w^{(2)} = [z_1^{(3)}\ z_2^{(3)}]$$

$$x^{(2)} = [g(z_1^{(2)})\ g(z_2^{(2)})] \qquad x^{(3)} = h = g(z^{(3)}) = [g(z_1^{(3)})\ g(z_2^{(3)})]$$

Weight update in multi-layer ANN

- **Apply the delta rule for each layer step by step (from the output layer).**



$$\delta_1^{(3)} = e_1^{(3)} g'(z_1^{(3)})$$

$$\delta_2^{(3)} = e_2^{(3)} g'(z_2^{(3)})$$

$h = [h_1 \;\; h_2]$ $\quad e^{(3)} = [e_1^{(3)} \;\; e_2^{(3)}]$ $\quad z^{(3)} = x^{(2)} \cdot w^{(2)} = [z_1^{(3)} \; z_2^{(3)}]$

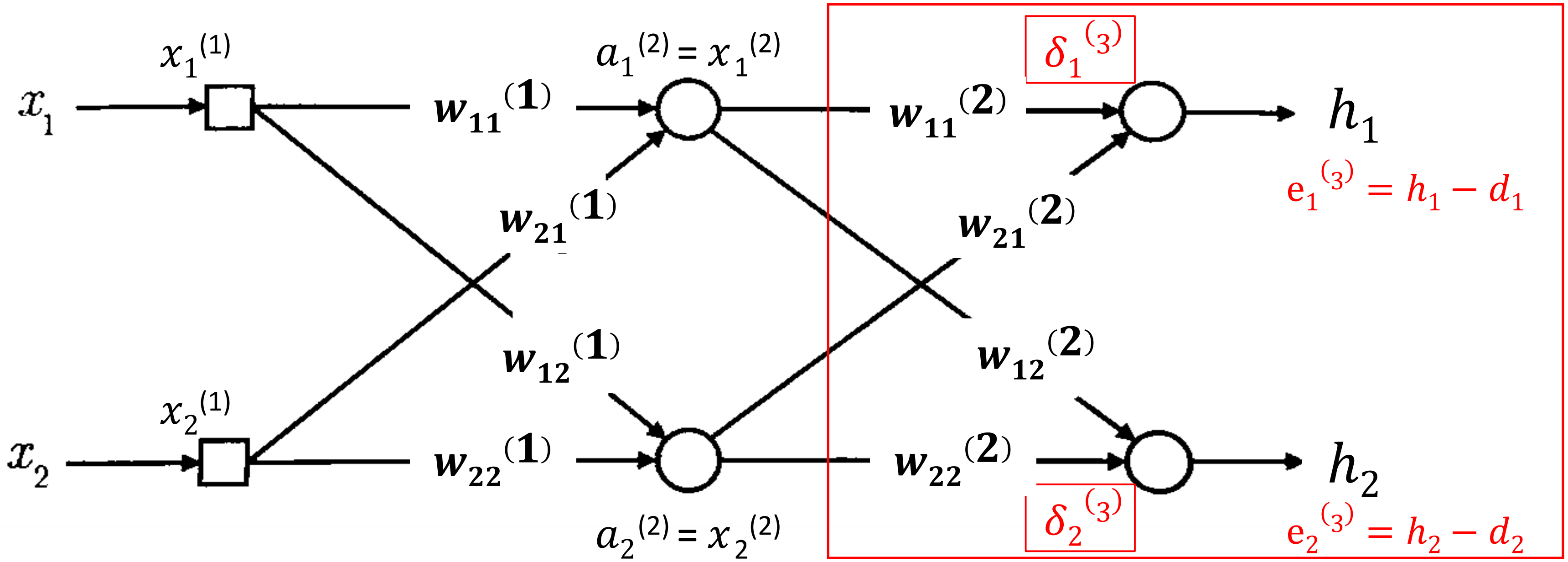
$d = [d_1 \;\; d_2]$ $\quad e^{(3)} = h - d$ $\quad g'(z^{(3)}) = [g'(z_1^{(3)}) \;\; g'(z_2^{(3)})]$

$$\delta^{(3)} = e^{(3)} * g'(z^{(3)}) = [e_1^{(3)} g'(z_1^{(3)}) \quad e_2^{(3)} g'(z_2^{(3)})]$$

※ * is not the inner product!

Weight update in multi-layer ANN

Apply the delta rule for each layer step by step (from the output layer).



$$\delta_1^{(3)} = e_1^{(3)} g'(z_1^{(3)}) \rightarrow w_{11}^{(2)} = w_{11}^{(2)} - \alpha\delta_1^{(3)} x_1^{(2)},\ w_{21}^{(2)} = w_{21}^{(2)} - \alpha\delta_1^{(3)} x_2^{(2)}$$

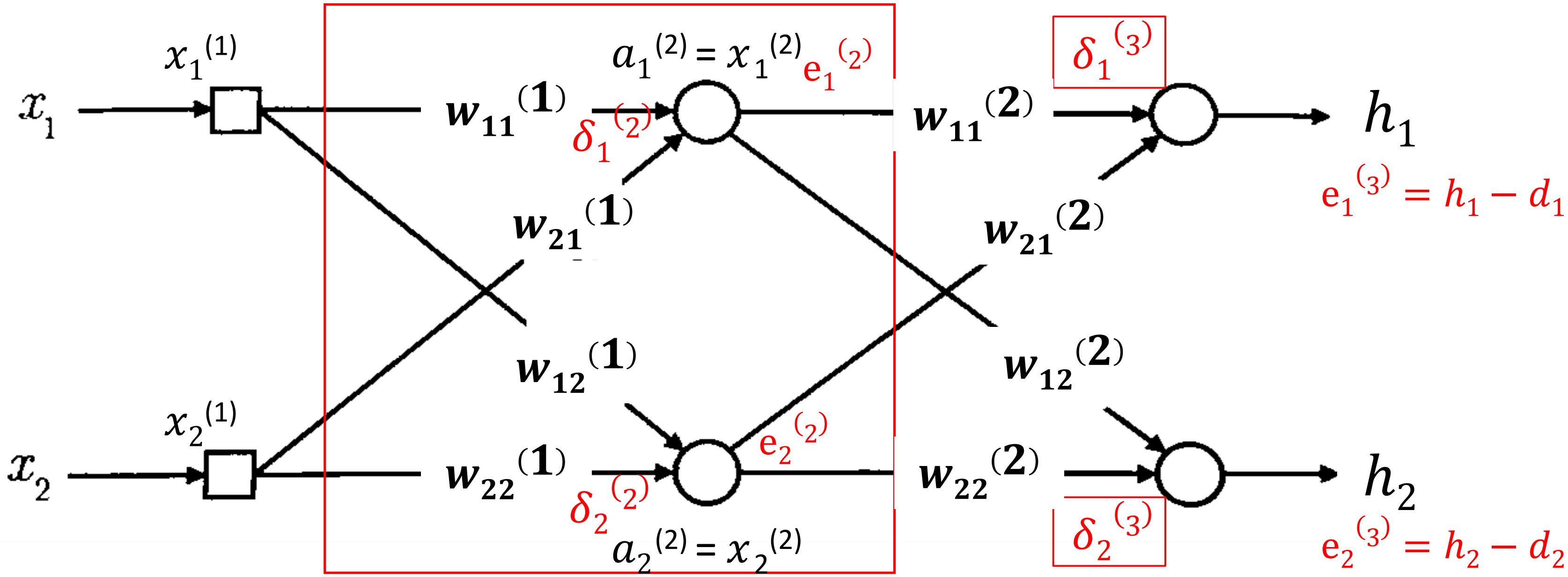
$$\delta_2^{(3)} = e_2^{(3)} g'(z_2^{(3)}) \rightarrow w_{12}^{(2)} = w_{12}^{(2)} - \alpha\delta_2^{(3)} x_1^{(2)},\ w_{22}^{(2)} = w_{22}^{(2)} - \alpha\delta_2^{(3)} x_2^{(2)}$$

$$\Delta w^{(2)} = \begin{bmatrix} -\alpha\delta_1^{(3)} x_1^{(2)} & -\alpha\delta_2^{(3)} x_1^{(2)} \\ -\alpha\delta_1^{(3)} x_2^{(2)} & -\alpha\delta_2^{(3)} x_2^{(2)} \end{bmatrix} = -\alpha \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} [\delta_1^{(3)} \delta_2^{(3)}] = -\alpha (x^{(2)})^T \cdot \delta^{(3)}$$

$$w^{(2)} = w^{(2)} + \Delta w^{(2)}$$

Weight update in multi-layer ANN

- **Apply the delta rule for each layer step by step (from the output layer).**



How can we get the $e^{(2)}$ and $\delta^{(2)}$?? → Here, we can use the "back propagation".

$$e_1^{(2)} = \delta_1^{(3)} w_{11}^{(2)} + \delta_2^{(3)} w_{12}^{(2)} \rightarrow \delta_1^{(2)} = e_1^{(2)} g'(z_1^{(2)})$$

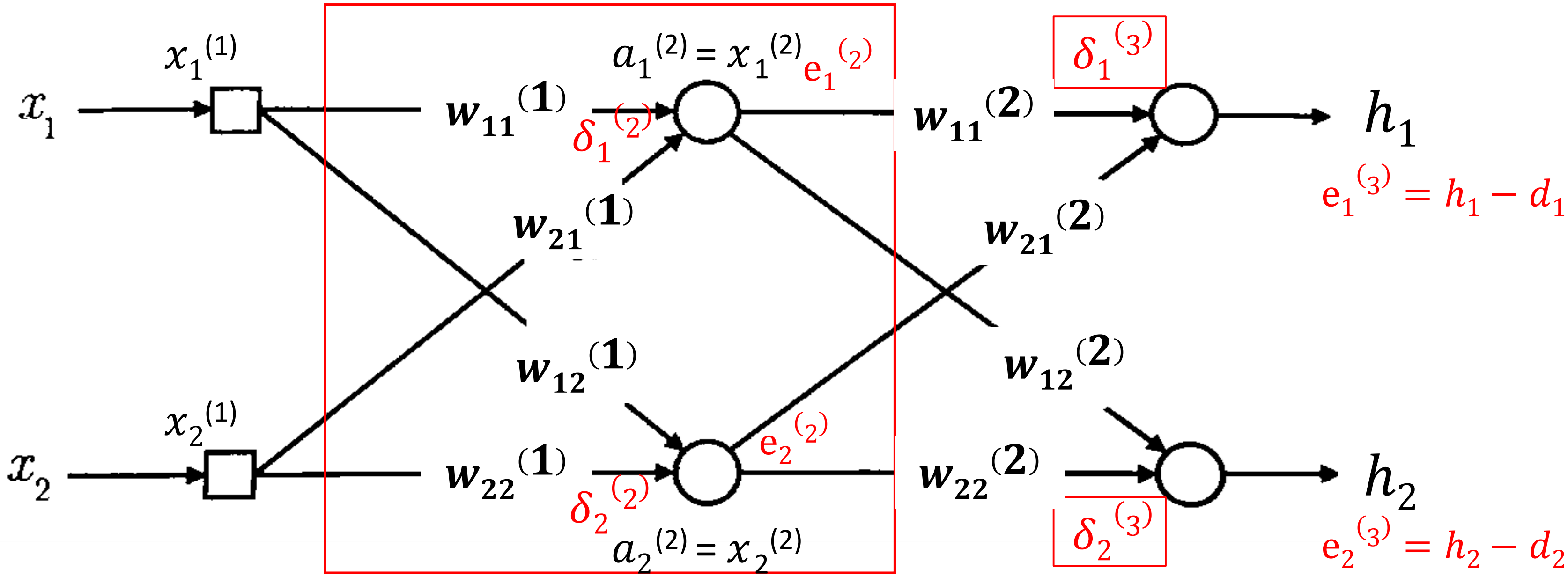
$$e_2^{(2)} = \delta_1^{(3)} w_{21}^{(2)} + \delta_2^{(3)} w_{22}^{(2)} \rightarrow \delta_2^{(2)} = e_2^{(2)} g'(z_2^{(2)})$$

$$\begin{aligned} e^{(2)} &= [e_1^{(2)} \; e_2^{(2)}] \\ &= [\delta_1^{(3)} \; \delta_2^{(3)}] \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} \end{bmatrix} \\ &= \delta^{(3)} \cdot (w^{(2)})^T \end{aligned}$$

$$\delta^{(2)} = e^{(2)} * g'(z^{(2)})$$

Weight update in multi-layer ANN

Apply the delta rule for each layer step by step (from the output layer).



$$\delta_1^{(2)} = e_1^{(2)} g'(z_1^{(2)}) \rightarrow w_{11}^{(1)} = w_{11}^{(1)} - \alpha\delta_1^{(2)} x_1^{(1)},\ w_{21}^{(1)} = w_{21}^{(1)} - \alpha\delta_1^{(2)} x_2^{(1)}$$

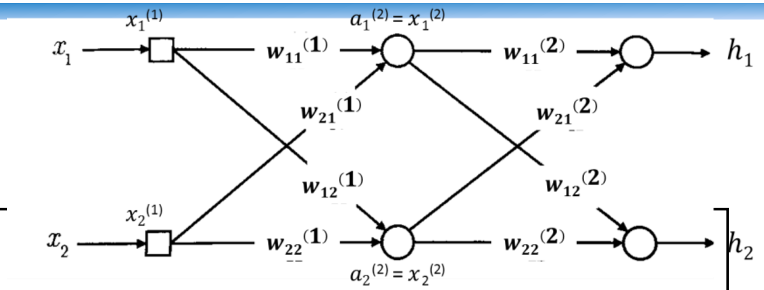
$$\delta_2^{(2)} = e_2^{(2)} g'(z_2^{(2)}) \rightarrow w_{12}^{(1)} = w_{12}^{(1)} - \alpha\delta_2^{(2)} x_1^{(1)},\ w_{22}^{(1)} = w_{22}^{(1)} - \alpha\delta_2^{(2)} x_2^{(1)}$$

$$\Delta w^{(1)} = \begin{bmatrix} -\alpha\delta_1^{(2)} x_1^{(1)} & -\alpha\delta_2^{(2)} x_1^{(1)} \\ -\alpha\delta_1^{(2)} x_2^{(1)} & -\alpha\delta_2^{(2)} x_2^{(1)} \end{bmatrix} = -\alpha \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} [\delta_1^{(2)} \delta_2^{(2)}] = -\alpha (x^{(1)})^T \cdot \delta^{(2)}$$

$$w^{(1)} = w^{(1)} + \Delta w^{(1)}$$

Process of the weight update in multi-layer ANN

Summary



1) Initialize the weight values properly

2) By entering the feature values of the training data into the neural network and feeding forward, obtain the output value h (this process is also called inference)

$$z^{(2)} = x^{(1)} \cdot w^{(1)} \qquad x^{(2)} = g(z^{(2)})$$
$$z^{(3)} = x^{(2)} \cdot w^{(2)} \qquad x^{(3)} = g(z^{(3)}) = h$$

3) Calculate the error at the output layer

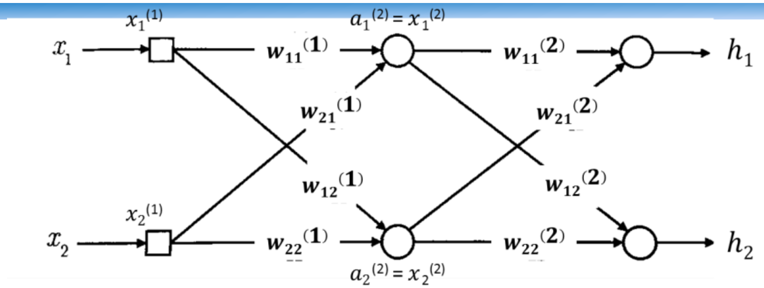
$$e^{(3)} = h - d$$

4) Conduct back propagation to calculate delta at the hidden layer.

$$e^{(3)} = h - d\ ,\ \delta^{(3)} = e^{(3)} * g'(z^{(3)})$$
$$e^{(2)} = \delta^{(3)} \cdot (w^{(2)})^T\ ,\ \delta^{(2)} = e^{(2)} * g'(z^{(2)})$$

Process of the weight update in multi-layer ANN

Summary



5) Calculate the amount of weight update based on the delta values.

$$\Delta w^{(2)} = -\alpha (x^{(2)})^T \cdot \delta^{(3)}$$
$$\Delta w^{(1)} = -\alpha (x^{(1)})^T \cdot \delta^{(2)}$$

6) Update the weights.

$$w^{(2)} = w^{(2)} + \Delta w^{(2)}$$
$$w^{(1)} = w^{(1)} + \Delta w^{(1)}$$

1) 7) Repeat the above 2-6 for the entire training data (1 epoch)

1) 8) Repeat the above epoch until the error becomes sufficiently small.

Even if the number of hidden layers increases, the overall concept is the same.

Different cost function (cross entropy)

- Squared error have been used as the cost function until the last slide.
- We can also use the cross entropy function (used in the logistic regression) as the cost function.
- **In general, the cross entropy shows better performance because it is more sensitive to the error.**

$$J = \sum_{i=1}^{n} [-d_i \log(h_i(x)) - (1-d_i)\log(1-h_i(x))]$$

Different cost function (cross entropy)

- **It doesn't change the process much.**
- **Only the method for obtaining delta at the output layer changes.**

1) Initialize the weight values properly

2) By entering the feature values of the training data into the neural network and feeding forward, obtain the output value h (this process is also called inference)

$$z^{(1)} = x^{(1)} \cdot w^{(1)} \qquad x^{(2)} = g(z^{(1)})$$
$$z^{(2)} = x^{(2)} \cdot w^{(2)} \qquad x^{(3)} = g(z^{(2)}) = h$$

3) Calculate the error at the output layer

$$e^{(3)} = h - d$$

4) Conduct back propagation to calculate delta at the hidden layer.

$$e^{(3)} = h - d \,,\ \delta^{(3)} = e^{(3)}$$
$$e^{(2)} = \delta^{(3)} \cdot (w^{(2)})^T \,,\ \delta^{(2)} = e^{(2)} * g'(z^{(2)})$$

Gradient descent in logistic regression (revisited)

$$\frac{\partial\, \text{Cost}\,(h_\theta(x^i)\,,y)}{\partial \theta_j} = \frac{\partial\, \text{Cost}\,(h_\theta(x^i)\,,y^i)}{\partial h_\theta(x^i)} \frac{\partial h_\theta(x^i)}{\partial z} \frac{\partial z}{\partial \theta_j}$$

$$z = \theta_0 + \theta_1 x_1{}^i + \theta_2 x_2{}^i + \cdots + \theta_n x_n{}^i \qquad \frac{\partial z}{\partial \theta_j} = x_j{}^i$$

$$\frac{\partial\, \text{Cost}\,(h_\theta(x^i)\,,y^i)}{\partial \theta_j} = \underbrace{\frac{\partial\, \text{Cost}\,(h_\theta(x^i)\,,y^i)}{\partial h_\theta(x^i)} \frac{\partial h_\theta(x^i)}{\partial z}}_{\delta} \frac{\partial z}{\partial \theta_j}$$

$$= \underbrace{-[y^i \frac{1}{h_\theta(x^i)} - (1-y^i)\frac{1}{1-h_\theta(x^i)}] * h_\theta(x^i)\,(1\text{-}h_\theta(x^i))}_{\delta} \; * x_j{}^i$$

$$= \underbrace{-[y^i(1-h_\theta(x^i)) - (1-y^i)h_\theta(x^i)]}_{\delta} * x_j{}^i$$

$$= \underbrace{(h_\theta(x^i) - y^i)}_{\delta\, =\, e}\, x_j{}^i$$

Contents

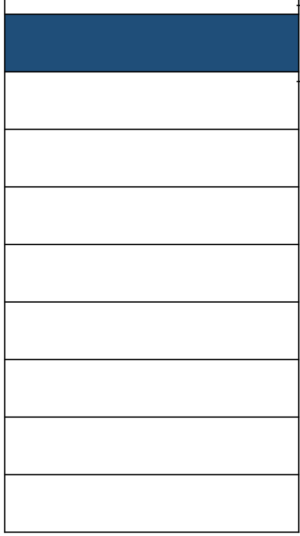
1. Unstability

- **The method of updating the weight for each example is called a stoichiometric gradient descent (SGD)**

  → **It can be unstable.**
  → **Batch or mini-batch**

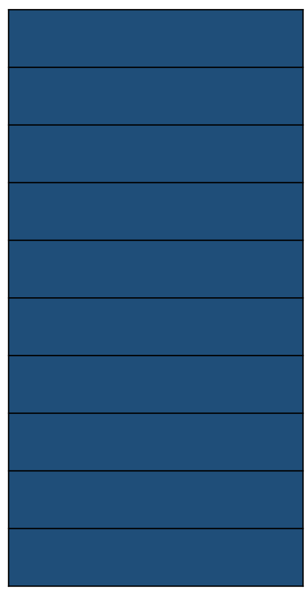
Stochastic

training



Training data

Batch



Training data

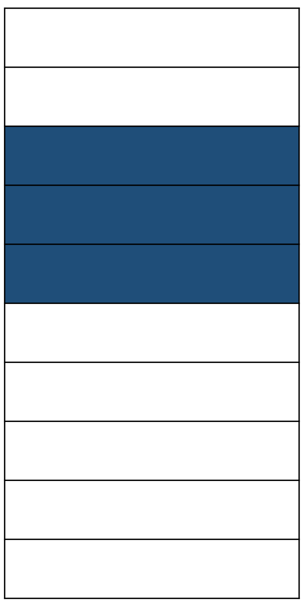
Calculate Δw_i
With each example but the update is conducted once with average.

$$\Delta w^{(1)} = \frac{1}{N} \sum \Delta w_i^{(1)}$$

Too slow

→ Mini-batch

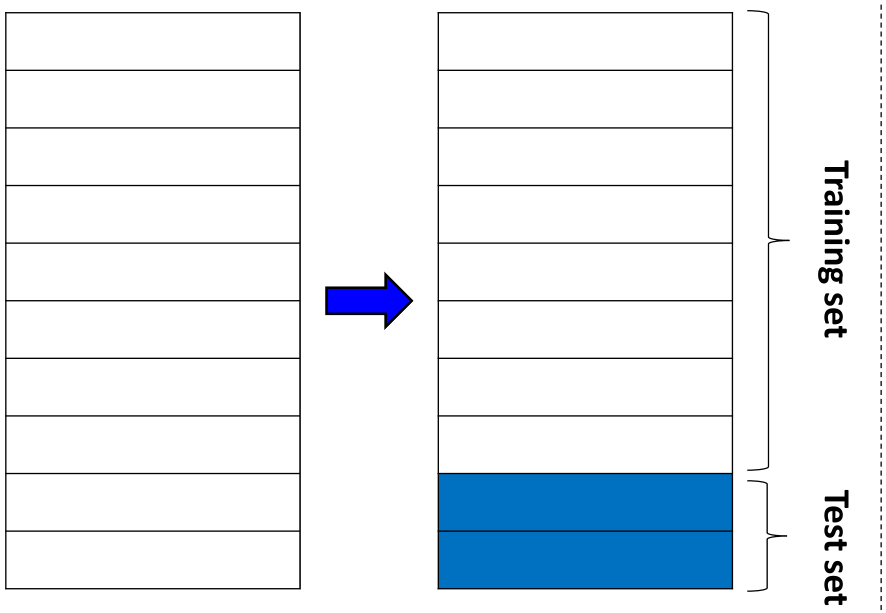
Mini-batch



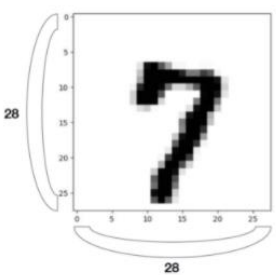
Training data

2. over-fitting

1. Utilize the separated test set to avoid over-fitting



1. **All data are divided into training set and test set.**
   → It usually has a ratio of 8:2.
2. **Conduct learning with the training set.**
3. **Validate the performance of the neural network with the training set**

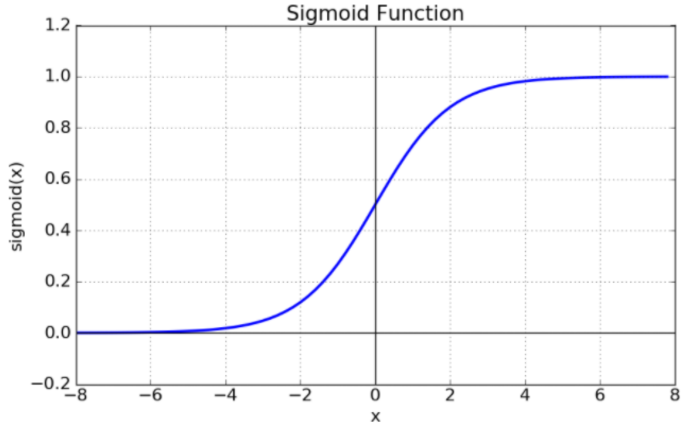


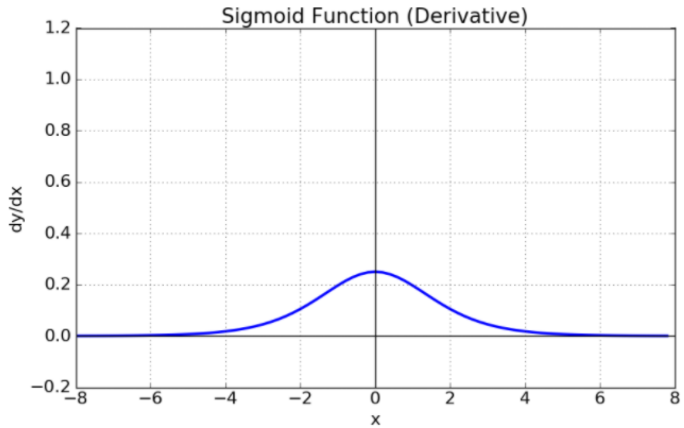

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes; 60,000 samples)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
T10k-images-idx3-ubyte.gz :  test set images (1648877 bytes; 10,000 samples)
T10k-labels-idx1-ubyte.gz :  test set labels (4542 bytes)
```

3. Poor learning ability

In this lecture, I explained the basic concept with familiar sigmoid.
But there are a few problems, and it is not a popular activation function in recent years.

Problem 1: Center value is not zero.





Sigmoid always outputs only positive values.

If the number of hidden layers is high, the variance continues to increase as it goes to the output layer, and the activation function output in the output layer is almost zero or converges to 1

→ The differential value may converge to zero.

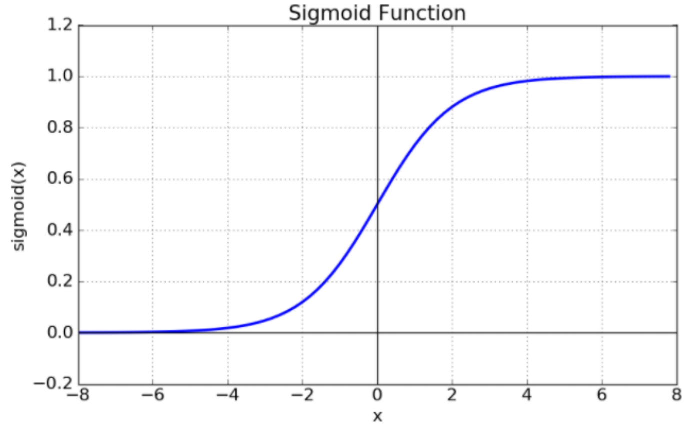
→ Unable to update weights

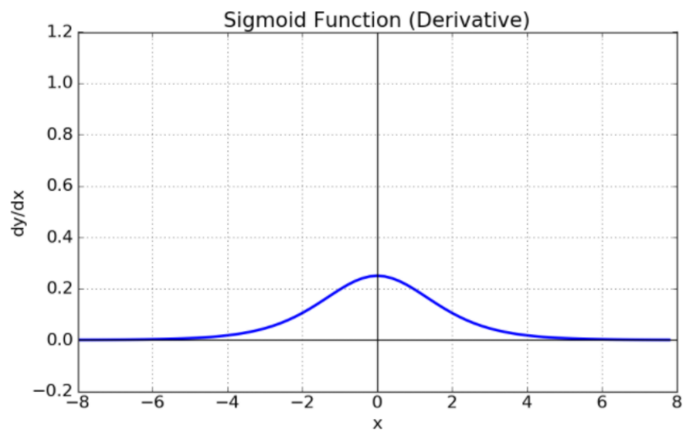
+ Additionally, the calculation is inefficient due to the exp.

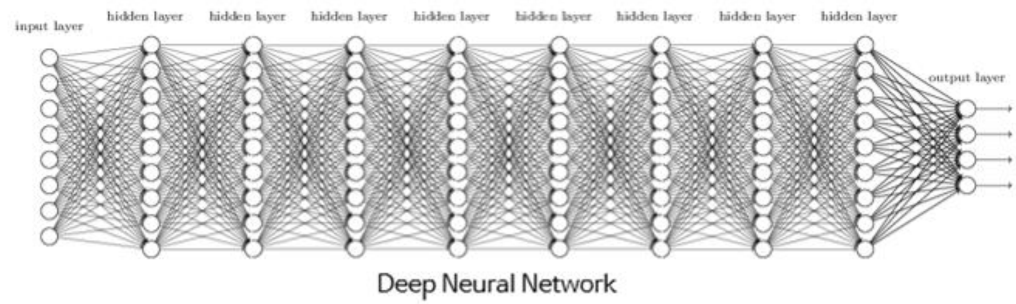
3. Poor learning ability

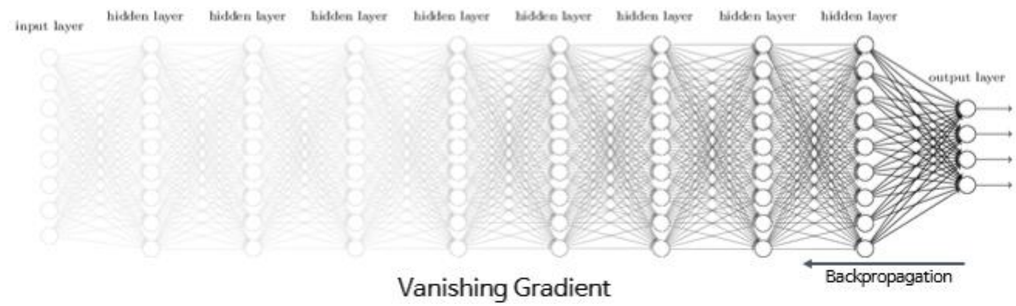
In this lecture, I explained the basic concept with familiar sigmoid.
But there are a few problems, and it is not a popular activation function in recent years.

Problem 2: Gradient vanishing









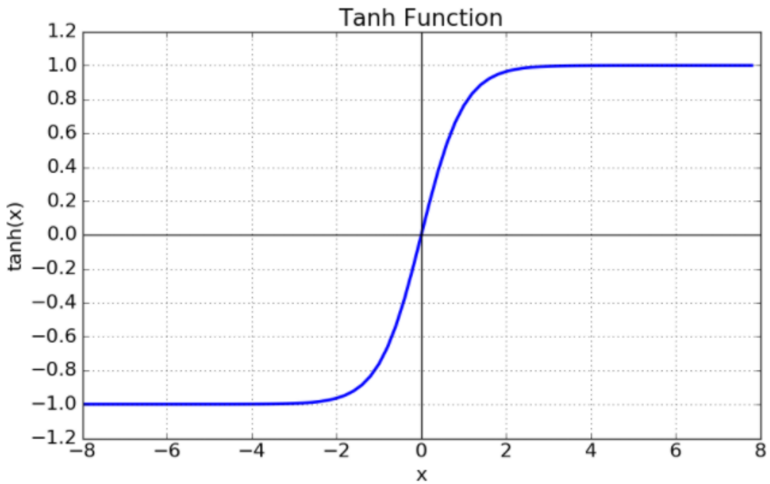
With high input → differential~ 0 → Training X

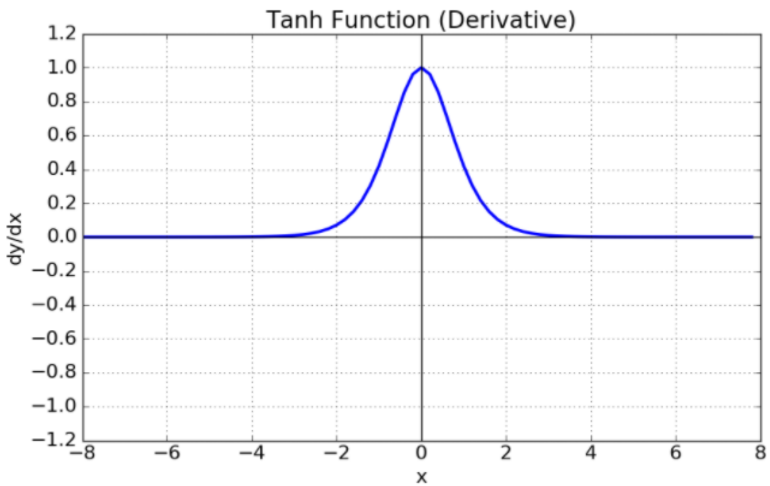
$\delta^{(k)} = e^{(k)} * g'(z^{(k)})$, $g'(z) = g(z)(1 - g(z)) < 1$

→ Delta value can be very small in the front layers.
→ No update in the front layers.

3. Alternatives

1. Hyperbolic Tangent





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

$$tanh'(x) = 1 - tanh^2(x)$$

1) **Center value : 0 ,**

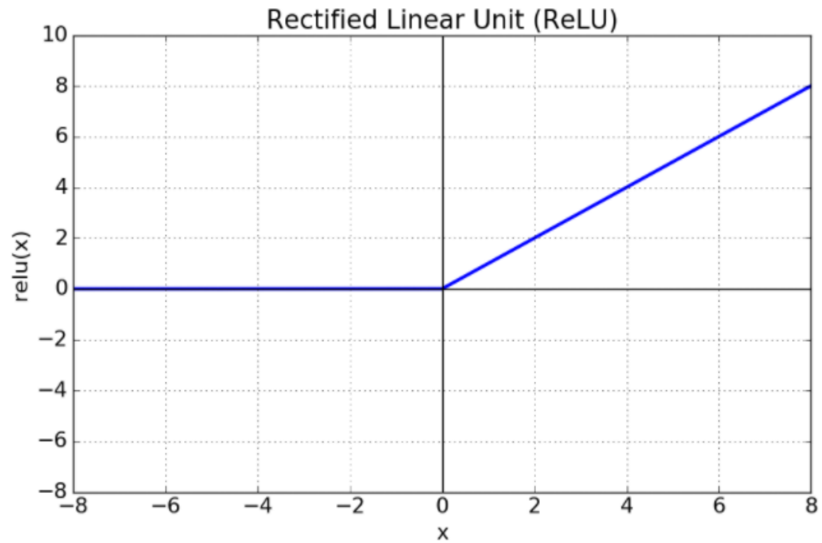
2) **It has large differential value → high endurance against to the gradient vanishing compared to the sigmoid function**

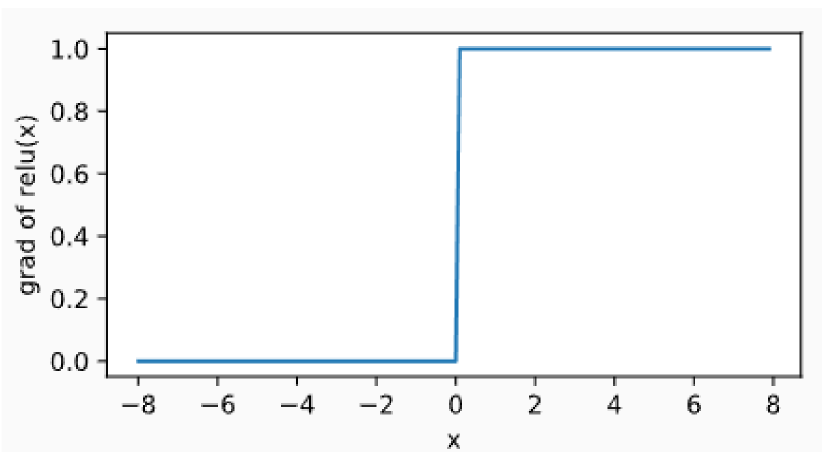
But, the gradient vanishing still occur

And it also use Exp

3. Alternatives

2. ReLU (Rectified Linear Unit)





$$g(x) = \begin{cases} x, & if\ x > 0 \\ 0, & if\ x \le 0 \end{cases}$$

$$g'(x) = \begin{cases} 1, & if\ x > 0 \\ 0, & if\ x \le 0 \end{cases}$$

1) Much faster update

2) Calculation in quite simple

3) For values with x<0, there is a disadvantage that neurons can die because the slope is zero.