Artificial neural networks (ANN)

1. Overview
2. Formal neurons
3. Perceptron / Multi-layered NN
4. Error back propagation
5. Approximate functions

1 Overview

- A neural network is a network of interconnected elements. These elements were inspired from studies of biological nervous systems.

- The function of a neural network is to produce an output pattern when presented with an input pattern.

Biological neuron

axon endings of other neurons

endings

synapse

soma

axon

dendrite

2 Formal neuron

Inputs

Weights

Fan-out

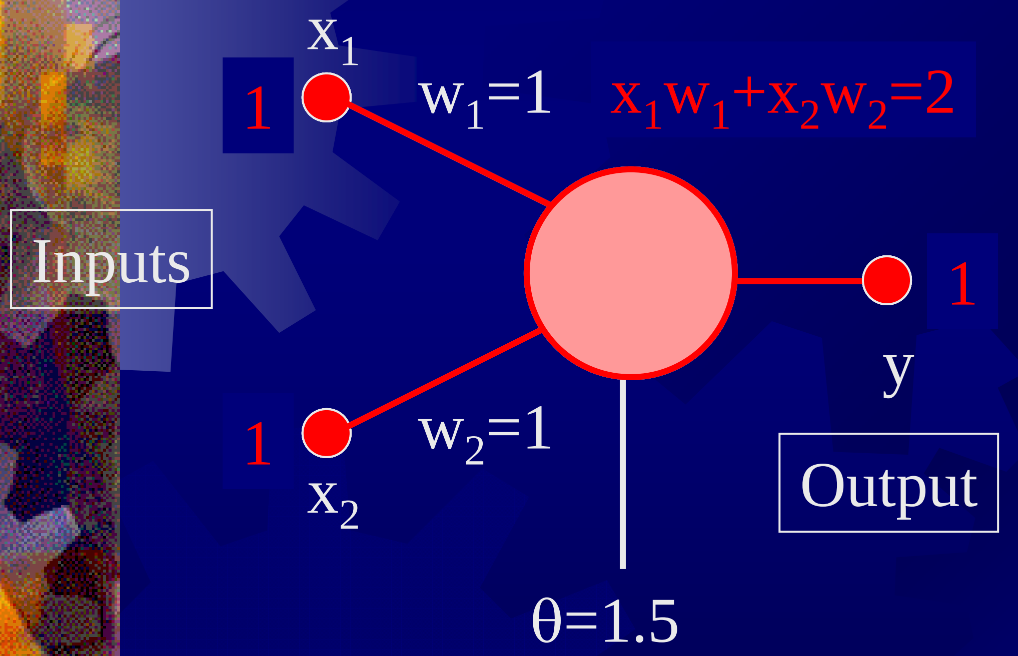
Threshold

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

$$f(x) = \begin{cases} 0 & (x<0) \\ 1 & (x \geqq 0) \end{cases}$$

Heaviside function

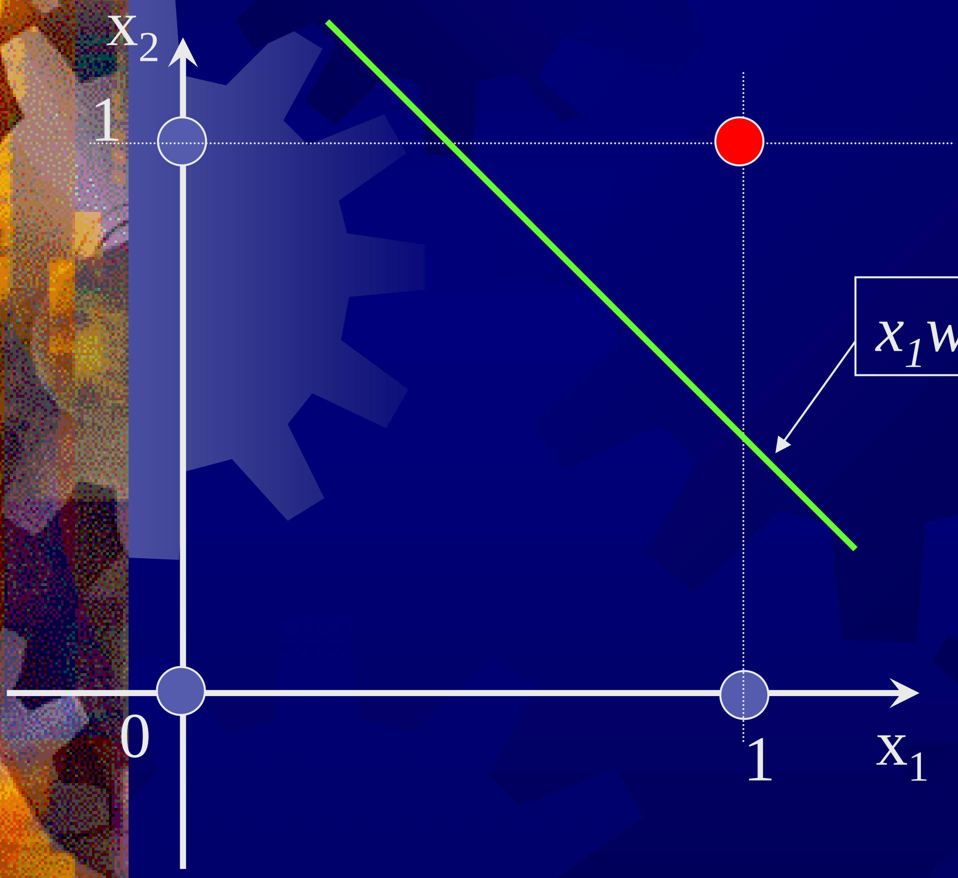
Calculation of a formal neuron



(0 0) → 0
(1 0) → 0
(0 1) → 0
(1 1) → 1

"AND" Logic

A formal neuron generates one hyper plane that divides input space



A hyper plane

$$\boxed{x_1 w_1 + x_2 w_2 = \theta\,(=1.5)}$$

(0 0) → 0
(1 0) → 0
(0 1) → 0
(1 1) → 1

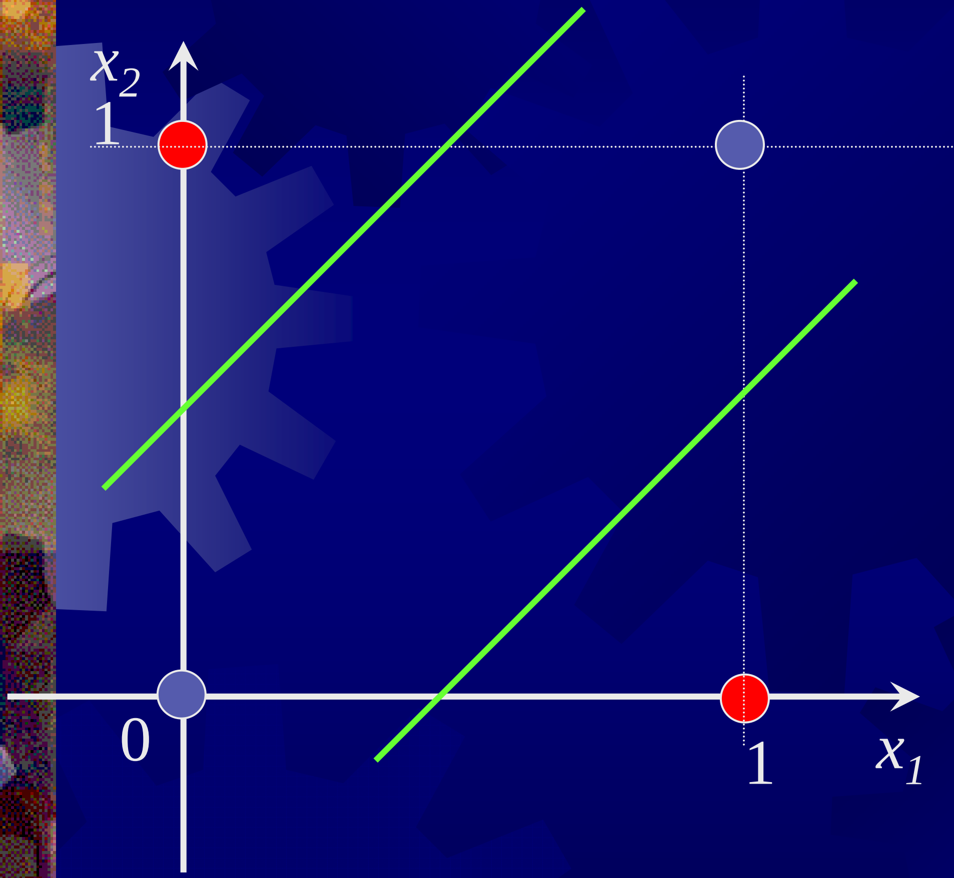
"AND" Logic

Other logics by a formal neuron

AND	OR	NOT
Inputs (x_1, x_2)	Inputs (x_1, x_2)	Input (x_1)
$w_1=1,\ w_2=1$	$w_1=1,\ w_2=1$	$w_1=-1$
$\theta=1.5$	$\theta=0.5$	$\theta=-0.5$

Only one hyper-plane a formal neuron can generate.

Functions that needs two or more hyper planes



Example: XOR
(0,0)→0
(1,0)→1
(0,1)→1
(1,1)→0

Combination or network of formal neurons is needed.

A combination for XOR

$(1, 0) \rightarrow 1$

$y_1=f(x_1-x_2-0.5)$

XOR

Inputs

OR

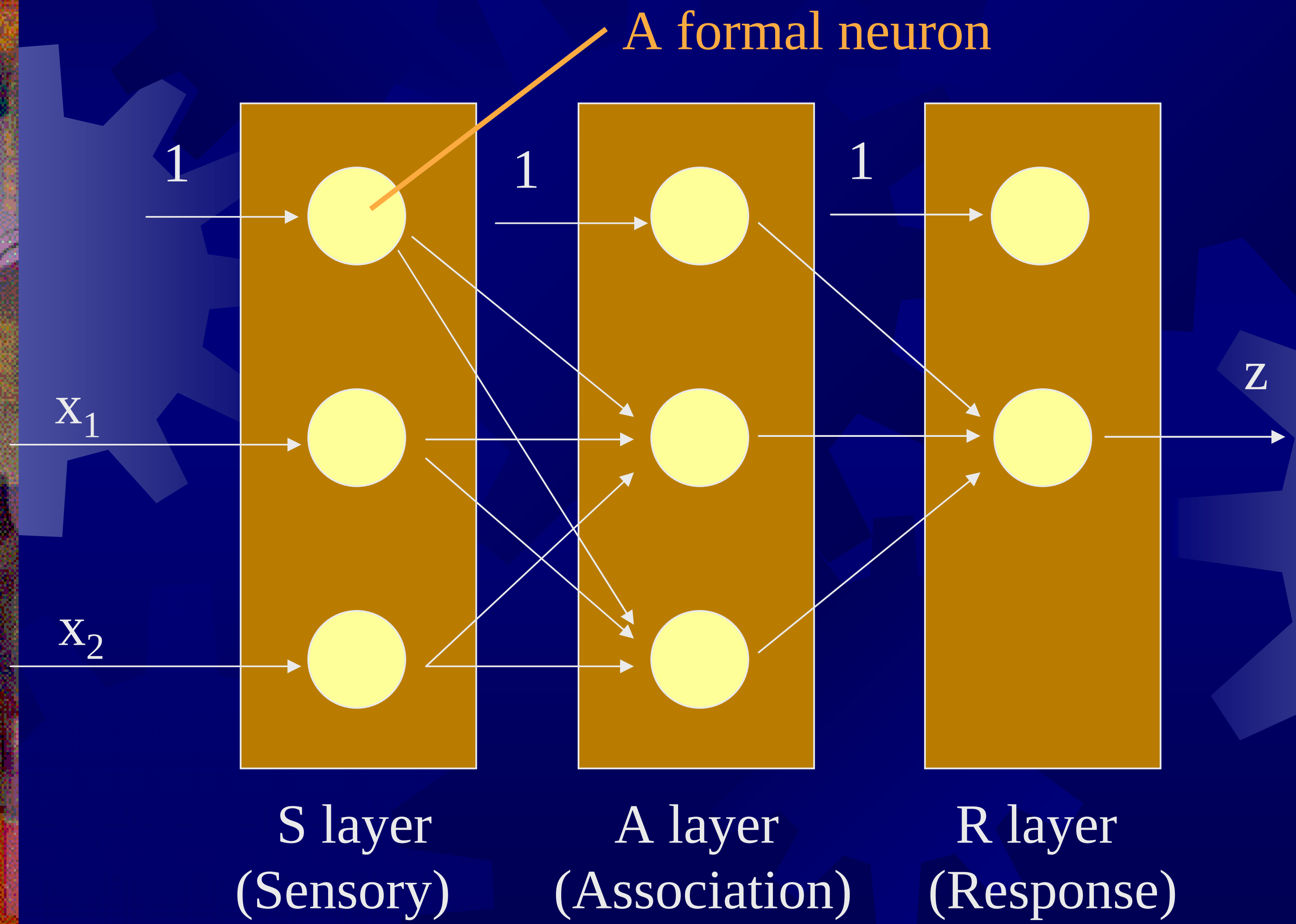
$(0, 1) \rightarrow 1$

$y_2=f(-x_1+x_2-0.5)$

3 Perceptron; Learning ANN

- “Perceptron” was proposed by Dr. F. Rosenblatt in 1958.
- It contains three layers of formal neurons called the Sensor, Association and Response.
- By changing the weights, a perceptron can learn correct outputs.

Perceptron: Multi-layered neural network

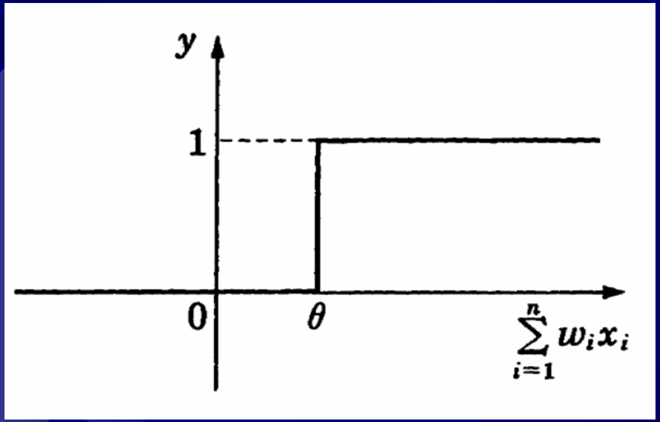


Extension of Perceptron; General Multi-layered NN

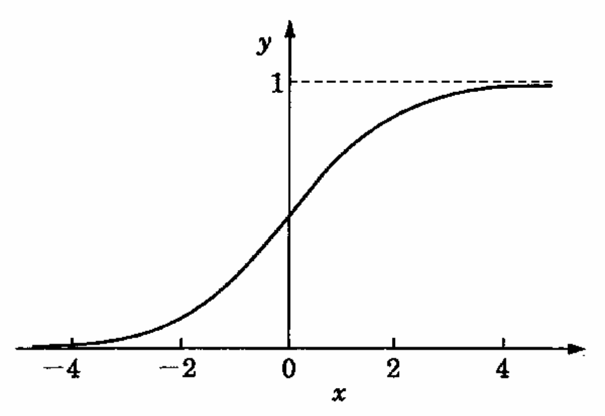
- The original perceptron has a week-point of learning. It does not guarantee to reach correct answer in all case.
- Prof. Amari and other researchers have proposed a method called “back propagation” by introducing sigmoid function instead of heaviside function.

Introducing sigmoid function

$$y=(1+e^{-x})^{-1}$$



Heaviside function



Sigmoid function

- In order to analyze neural networks, Heaviside function is not suitable because its derivative is not continuous.
- So, we introduce Sigmoid function that has continuous derivative.

Calculation in Multi-layered NN

Notation: Input for **i**th neuron in **l**th layer … x^l_i

Output of the neuron … z^l_i

Sensory layer = 0th layer, Response layer = Lth layer

Sensory Layer：output input signal directly

$$z^0_i = x_i$$

Other layers: calculate by Sigmoid function

$$z^l_i = f(x^l_i) = \frac{1}{1+e^{-x^l_i}}$$

Calculation of x^l_i

x^l_i depends on outputs of **(l-1)** th layer × weights

$$x^l_i = \sum_{j=0}^{n_{l-1}} w^l_{ij} \, z^{l-1}_j$$

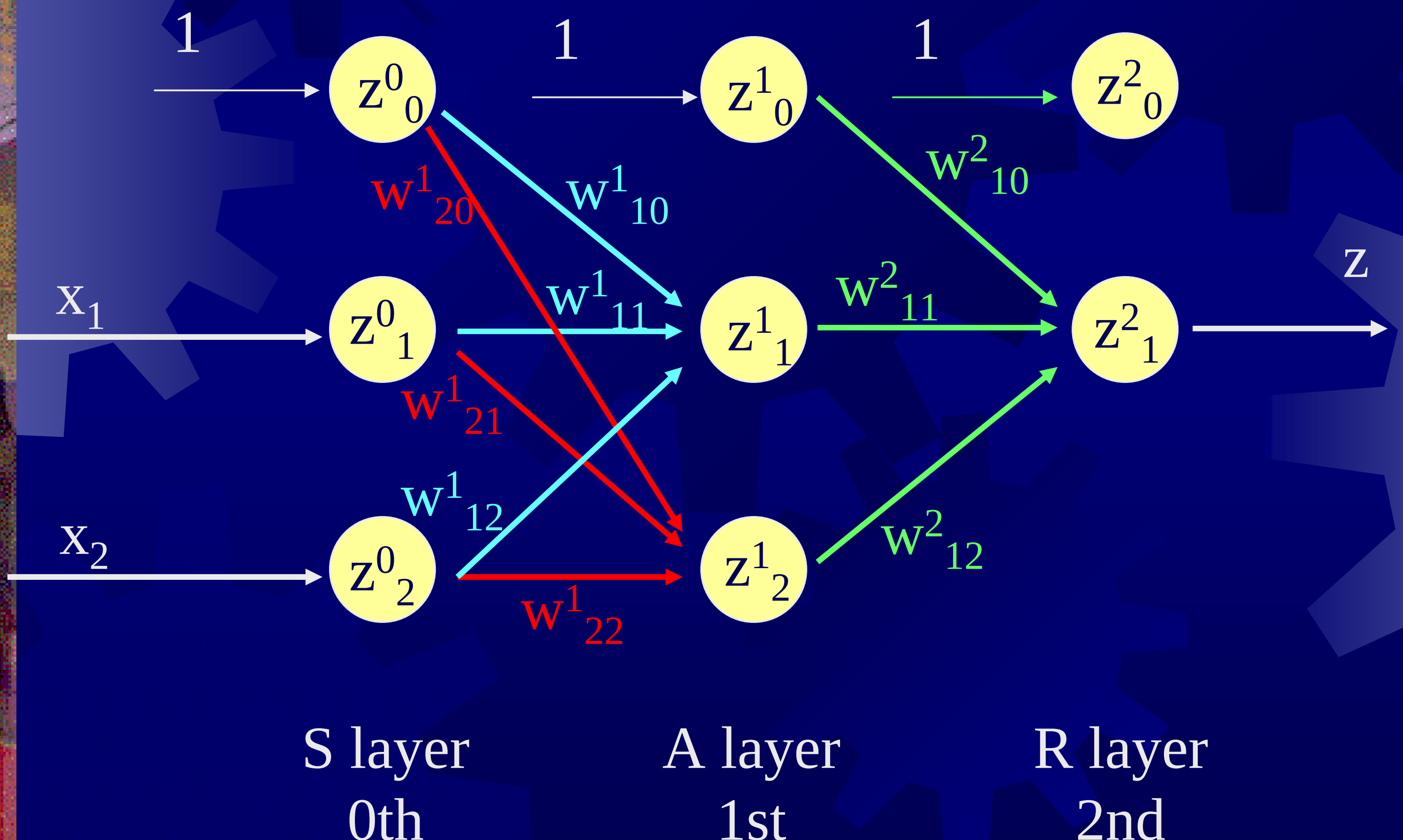
where

w^l_{ij} : weight from **j**th neuron in **l-1**th layer to **i**th neuron in **l**th layer.

n_{l-1}: Number of neurons in **l-1**th layer

Notation example

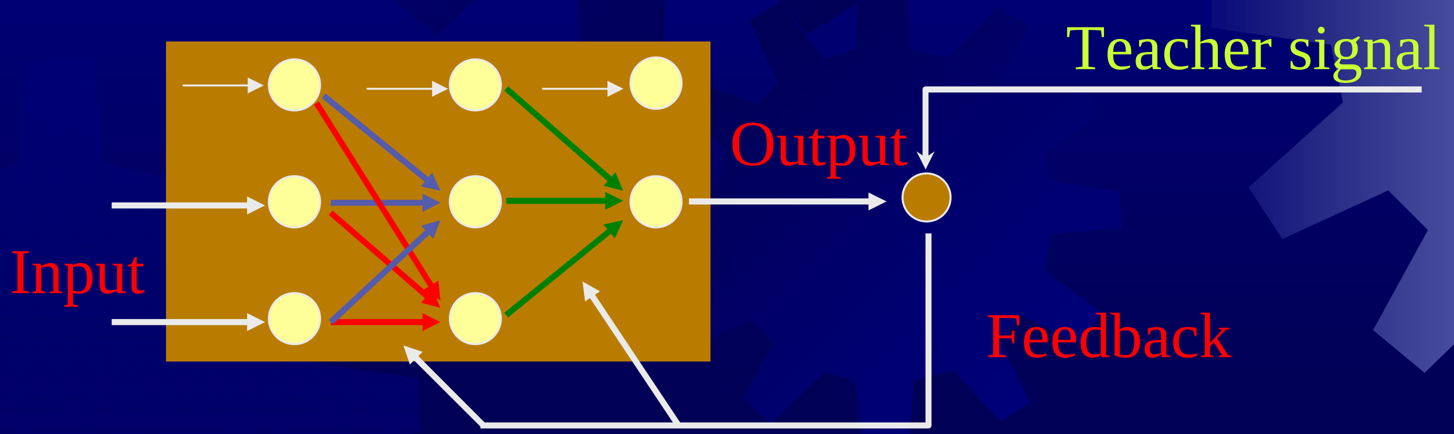
(2inputs 1output)



4 Error back propagation

Main idea

- Changes weights w^{*}_{**} according to output error.
- By comparing with teacher signals, feedback errors to the neural network.



Output error E

For single-output systems,

$$E = \frac{1}{2}(t - z)^2$$

where t is teacher signal

For multi-output systems,

$$E = \frac{1}{2}\sum_{j=1}^{n_L}(t_j - z^L{}_j)^2$$

Training: changing w^l_{ij} according to output error.

$$\underset{\text{(new)}}{w^l_{ij}} = \underset{\text{(current)}}{w^l_{ij}} + \Delta w^l_{ij}$$

$$\Delta w^l_{ij} = -\eta \frac{\partial E}{\partial w^l_{ij}}$$

η: Learning parameter $(0<\eta<1)$

Calculate Δw^{l}_{ij}

$$\Delta w^{l}_{ij} = -\eta \frac{\partial E}{\partial w^{l}_{ij}}$$

$$= -\eta \frac{\partial E}{\partial x^{l}_{i}} \frac{\partial x^{l}_{i}}{\partial w^{l}_{ij}}$$

Calculate Δw^{l}_{ij} (Cont.)

$$x^{l}_{i} = \sum_{j=0}^{n_{l-1}} w^{l}_{ij}\, z^{l-1}_{j}$$

$$= w^{l}_{i0} z^{l-1}_{0} + w^{l}_{i1} z^{l-1}_{1} + \ldots + w^{l}_{ij} z^{l-1}_{j} + \ldots$$

Then,

$$\Delta w^{l}_{ij} = - \eta \frac{\partial E}{\partial x^{l}_{i}} z^{l-1}_{j}$$

Calculate Δw^{l}_{ij} (Cont.)

Here, we define $\delta^{l}_{i} = -\dfrac{\partial E}{\partial x^{l}_{i}}$

$$\Delta w^{l}_{ij} = -\eta \frac{\partial E}{\partial x^{l}_{i}} z^{l-1}_{j}$$

$$= \eta\, \delta^{l}_{i}\, z^{l-1}_{j}$$

Therefore, we must calculate δ^{l}_{i}.

Calculate Δw^l_{ij} (Cont.)

At the **L**th layer (R layer),

$$E = \frac{1}{2}(\ t - z\ )^2 = \frac{1}{2}(\ t - f(x^L{}_1)\ )^2$$

The derivative of Sigmoid function f(x) becomes

$$\frac{d\,f(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = f(x)(1-f(x))$$

Therefore, at the **L**th layer,

$$\delta^L{}_1 = -\frac{\partial E}{\partial x^L{}_1} = (t\text{-}z)z(1\text{-}z)$$

(Notice : $f(x^L{}_i)=z$ because of R layer)

In the case of other layers,

$$\delta^l{}_j = z^l{}_j(1-z^l{}_j)\sum_{i=1}^{n_{l+1}} w^{l+1}{}_{ij}\delta^{l+1}{}_i$$

This result indicates δ of **l**th layer depends on δ in **l+1**th layer

Conclusion

- ANN = Network of formal neurons.
- Formal neurons with Heaviside function realizes logics.
- Back propagation method for neurons with Sigmoid function realizes learning ability and approximation of continuous functions.