Understanding Deep Neural Networks

Chapter Three

Backpropagation Algorithm

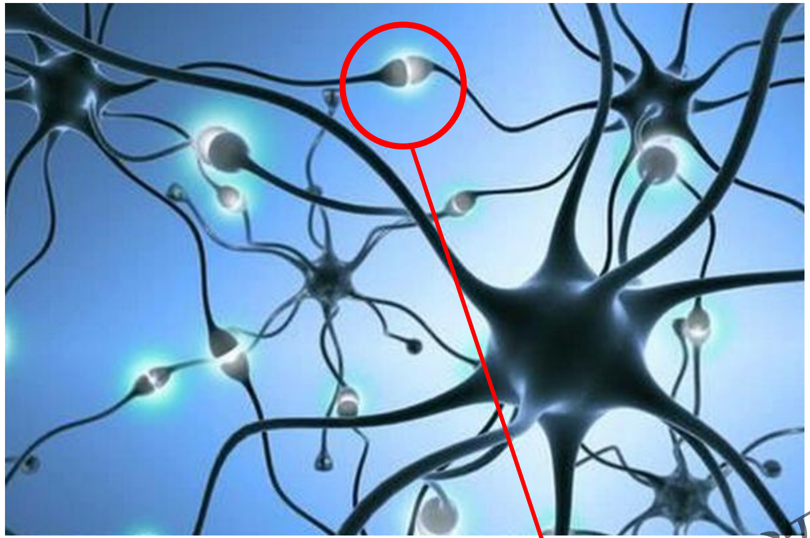
Zhang Yi, *IEEE Fellow*

Autumn 2020

Outline

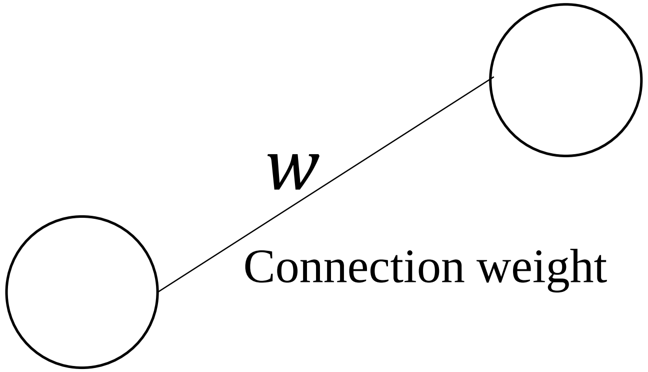
- Brief Review of Computational Model of Neural Networks
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

Computational Model of Synapse



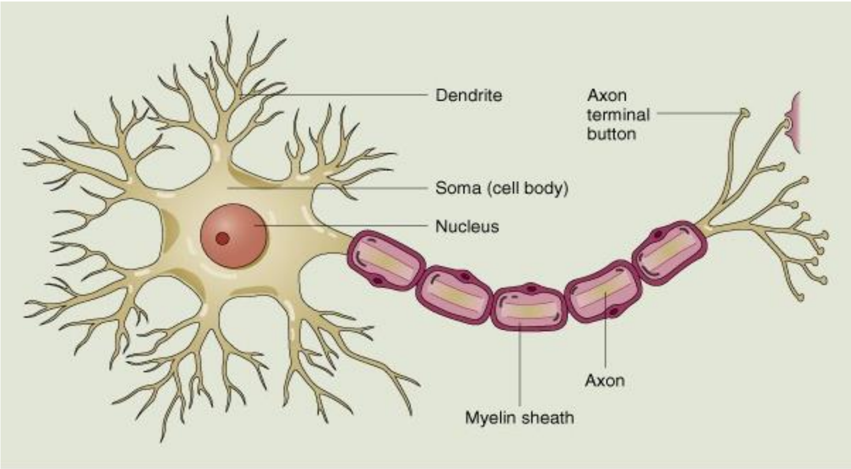
Synapse

$\approx$

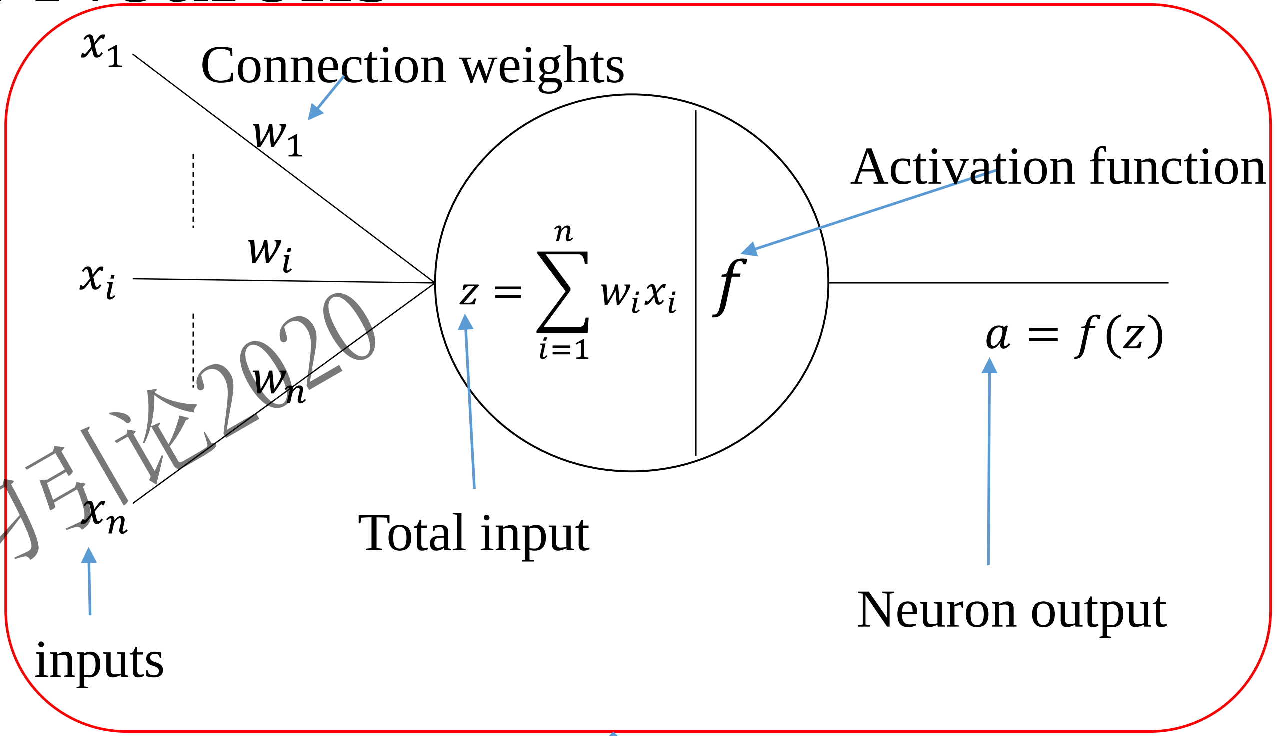


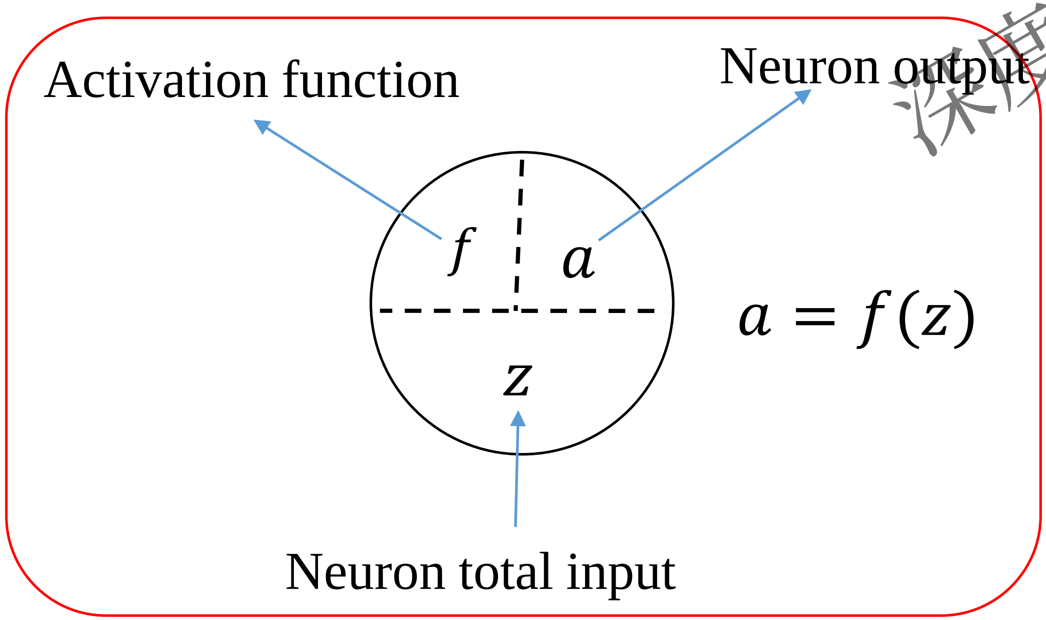
$w > 0$, exciting connection
$w = 0$, no connection
$w < 0$, inhibition connection

Computational Model of Neurons



$\approx$

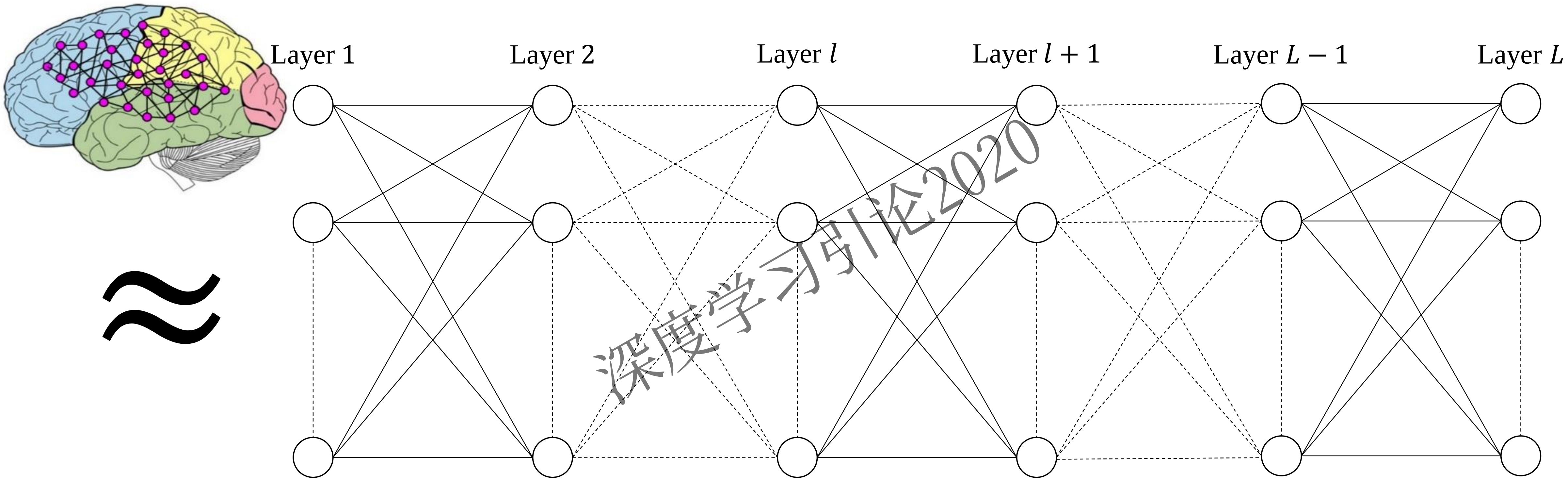




$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

$$z = \sum_{i=1}^{n} w_i x_i$$

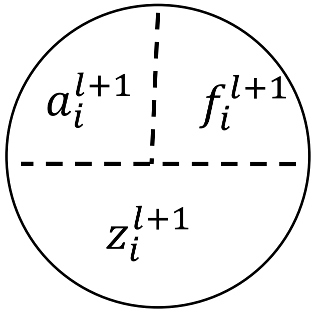
Computational Model of Neural Networks



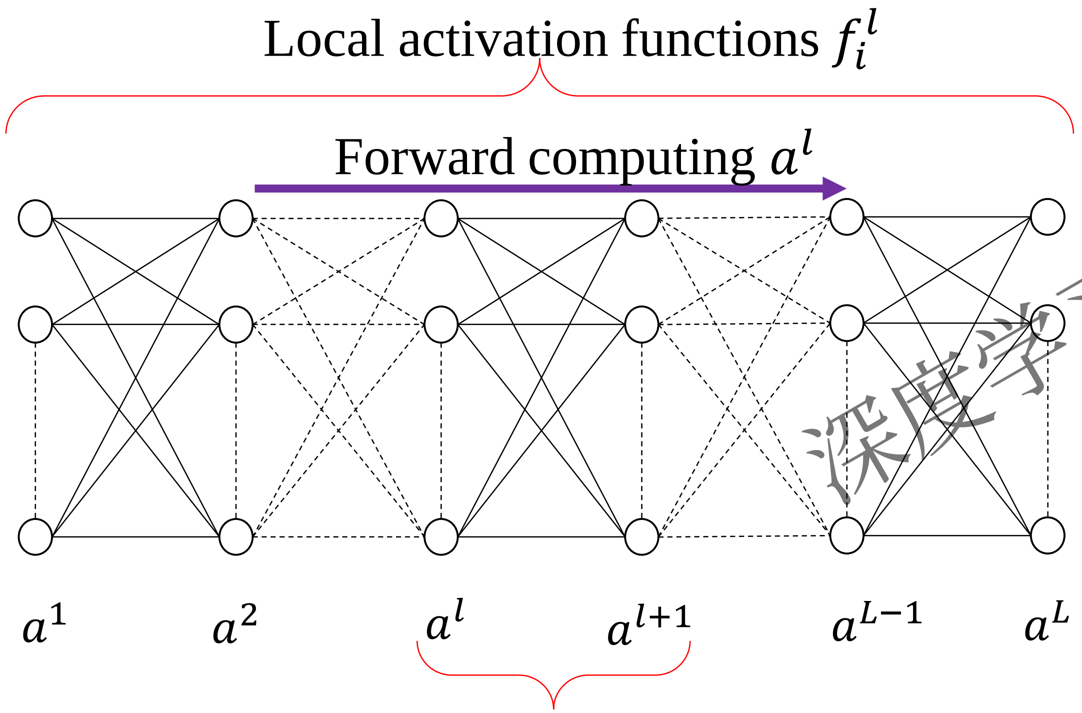
neurons	n_1		n_2		n_l		n_{l+1}		n_{L-1}		n_L
weights		W^1				W^l				W^{L-1}	
predictions	a^1		a^2		a^l		a^{l+1}		a^{L-1}		a^L

Forward Computing

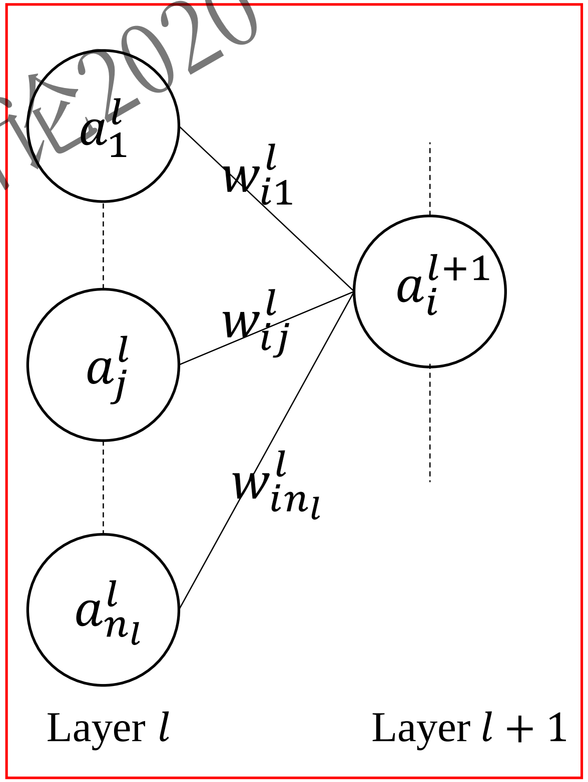
$l + 1$ layer i^{th} neuron



$$a_i^{l+1} = f_i^{l+1}(z_i^{l+1})$$

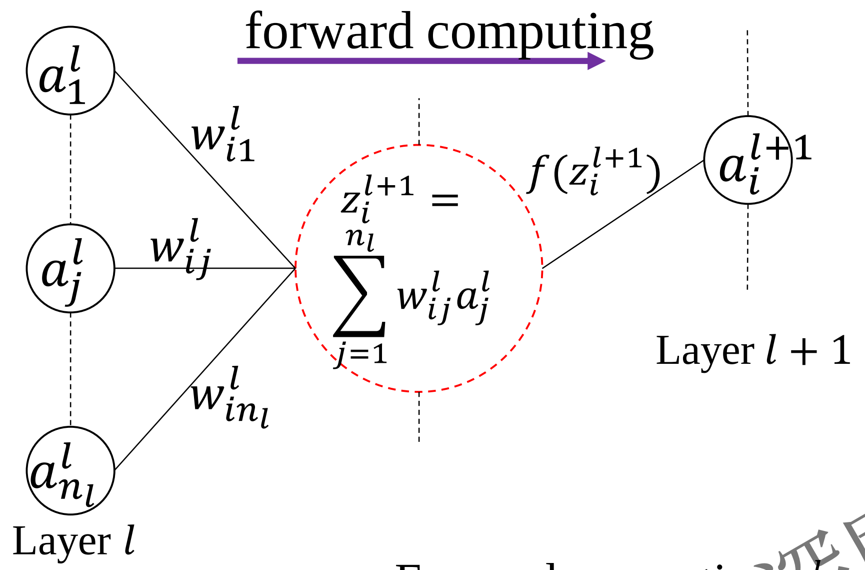


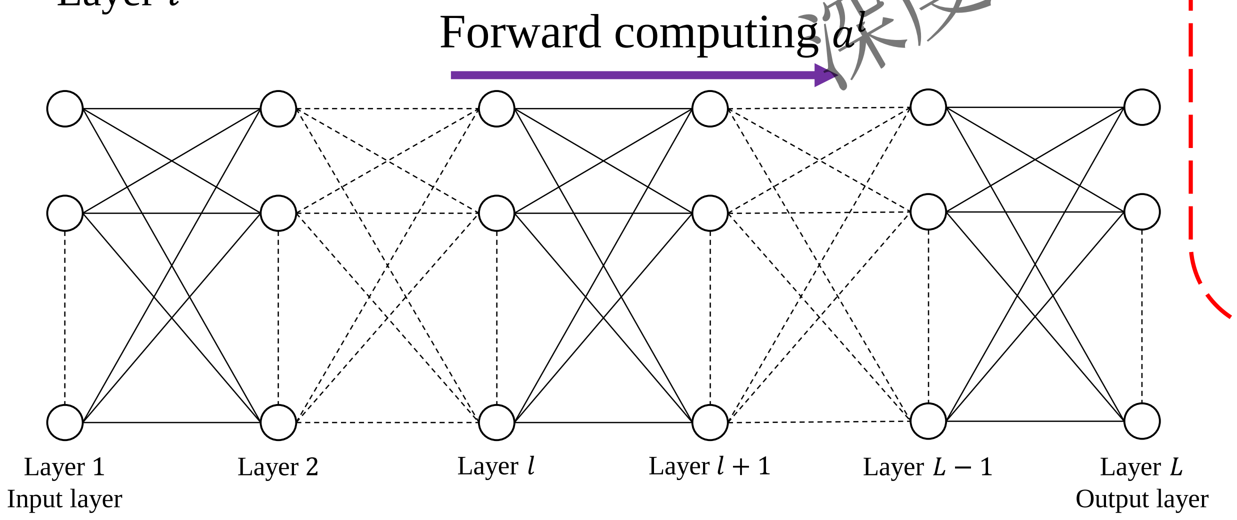
Computing unit



Computing unit

One page to understand forward computing





Component form

$$\begin{cases} a_i^{l+1} = f\left(z_i^{l+1}\right) \\ z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l \end{cases}$$

$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$

Vector form

$$\begin{cases} a^{l+1} = f\left(z^{l+1}\right) \\ z^{l+1} = w^l a^l \end{cases}$$

Algorithm:

$Input\ W^l, a^1$
$for\ l = 1{:}L$
$\quad a^{l+1} = fc\left(W^l, a^l\right)$
$return$

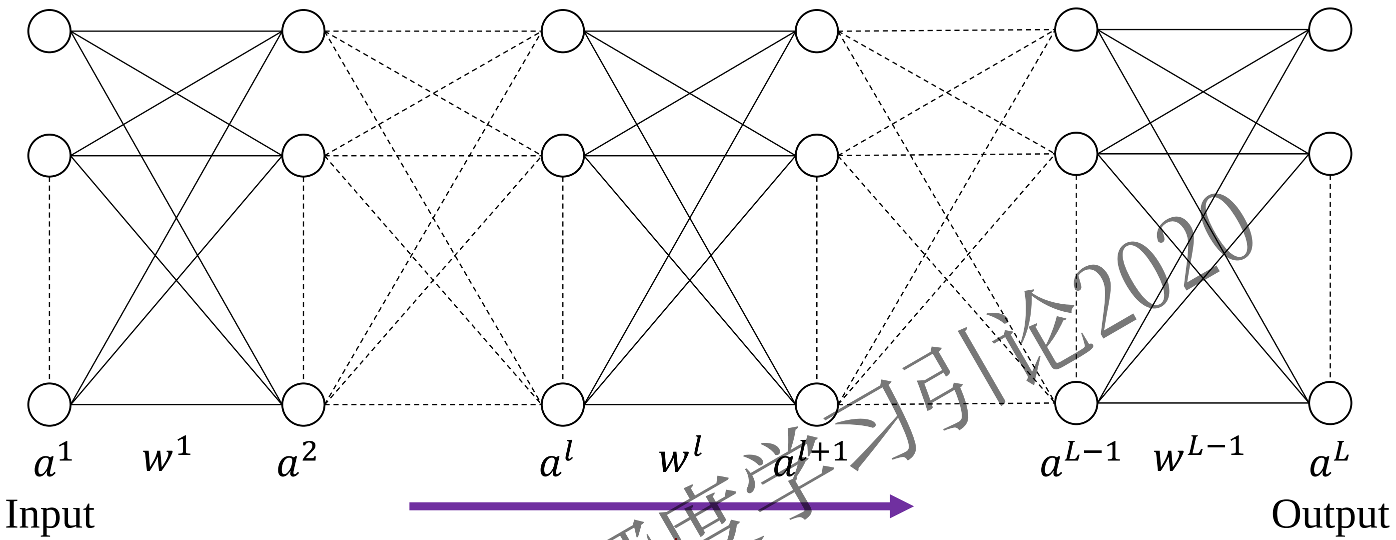
Function $fc\left(W^l, a^l\right)$
$for\ i = 1{:}n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f\left(z_i^{l+1}\right)$$

end

Nonlinear Mapping / Dynamical Systems



A neural network can be looked as a nonlinear mapping or a dynamical system.

$$a^L = f\left(W^{L-1} f\left(W^{L-2} f\left(W^{L-3} \cdots f\left(W^1 a^1\right)\right)\right)\right)$$

$$R^{n_1} \longrightarrow R^{n_L}$$

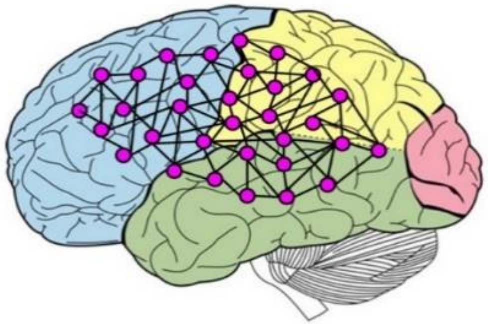
Nonlinear mapping

$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right) \xrightarrow{l \to t} a_i(t+1) = f\left(\sum_{j=1}^{n_t} w_{ij}(t) a_j(t)\right)$$

Discrete time dynamical system

Computational Model of Neural Networks

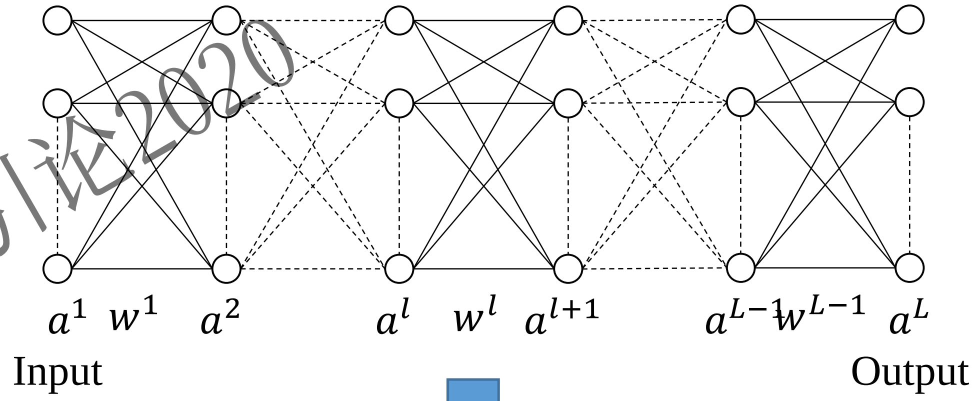
Biological neural network



Biological Intelligence

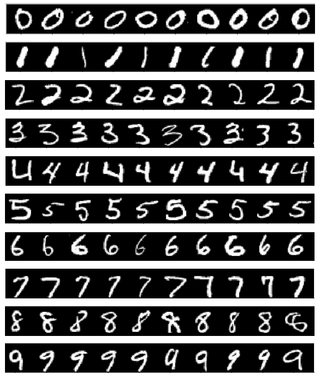
$\approx$

Artificial neural network

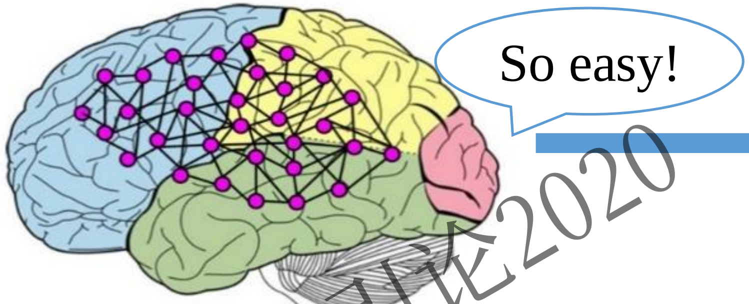


Artificial Intelligence

Handwritten digits recognition



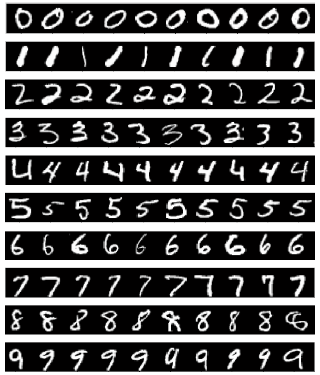
Biological neural network

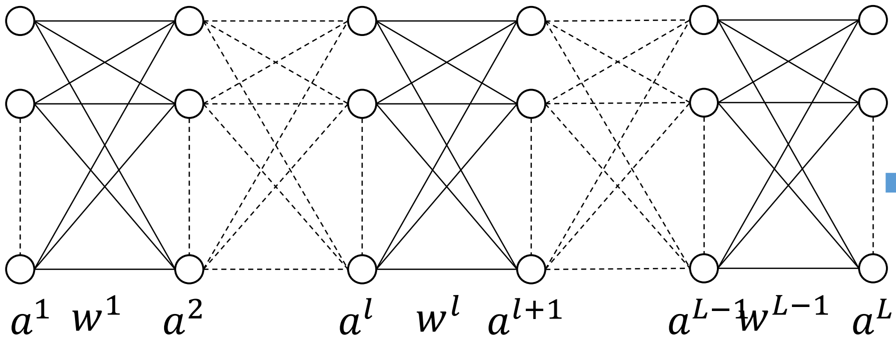


0
1
2
3
4
5
6
7
8
9

$\approx$

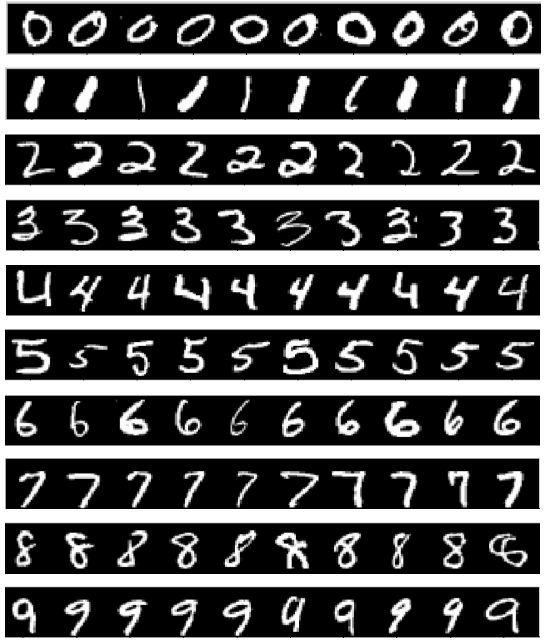
Artificial neural network

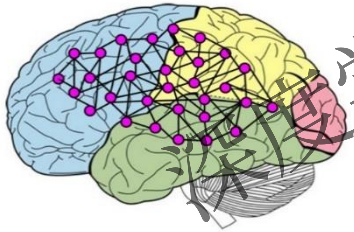




0
1
2
3
4
5
6
7
8
9

Handwritten digits recognition





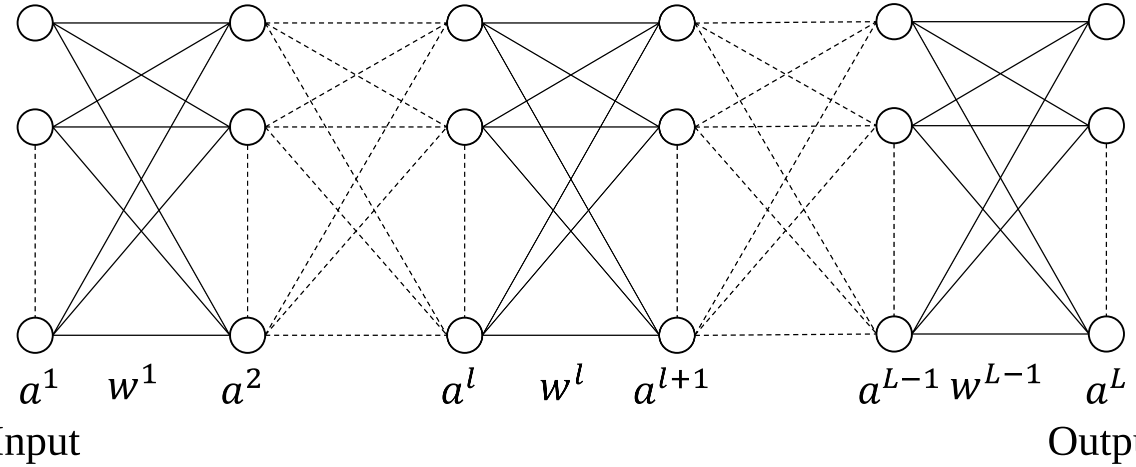
0
1
2
3
4
5
6
7
8
9

The human brain is so powerful so that any child can recognize the handwritten digits easily. Two important factors:

1. The brain has the structured ability.
2. Learning ability.

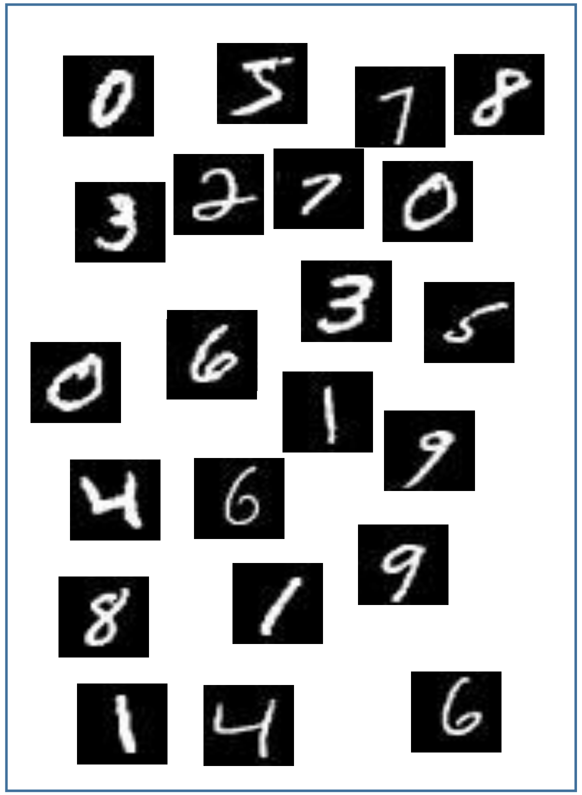
Next: How to develop learning methods to train the artificial network model?

Artificial neural network



Outline

Network Performance: Cost Function



Training Data

Training

Good Performance!

The father knows the correct answer.

Supervised Learning

9

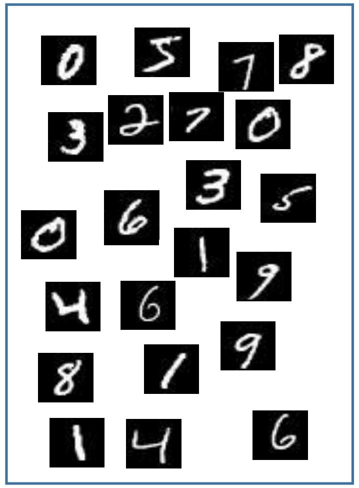
5

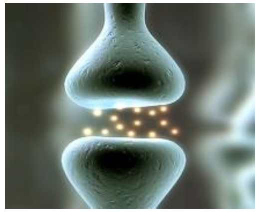
Two important factors:

1. There must be a measure to measure the correctness between correct answer and the boy's real output. ----- Performance function.

2. There must be a mechanism to change the knowledge system of the boy. ---- Learning algorithm.

Network Performance: Cost Function



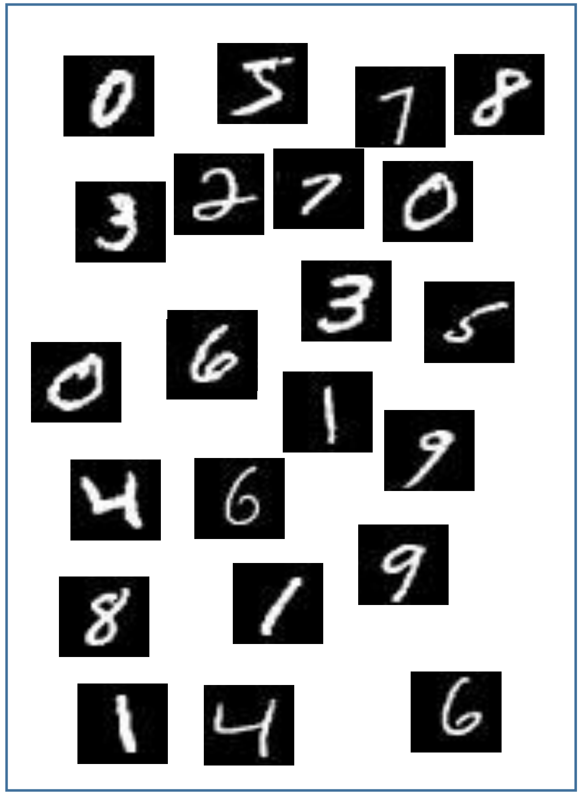


Updating

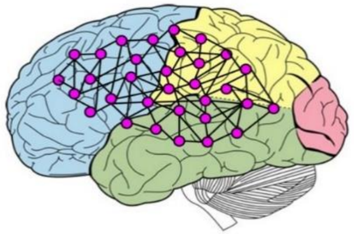
Training

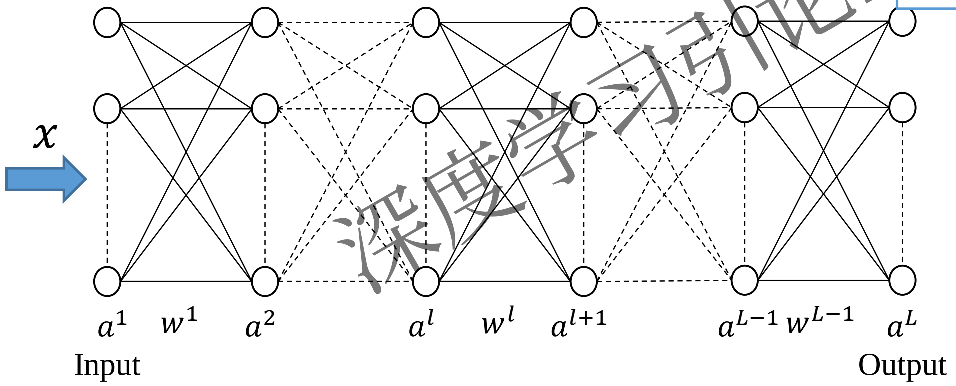
Boy's NN prediction		Father's target output		Error
9	−	6	=	3
5	−	5	=	0

Network Performance: Cost Function



Training Data Set





updating the weights: Learning algorithm

Network prediction

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

Target

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

$$J(a^L, y^L)$$

Performance function $J(a^L, y^L)$, or cost function, is used to describe the distance between a^L and y^L, $J(a^L, y^L)$ is indeed a function of $(w^1, \cdots, w^L)$, i.e.,

$$J = J(w^1, \cdots, w^L).$$

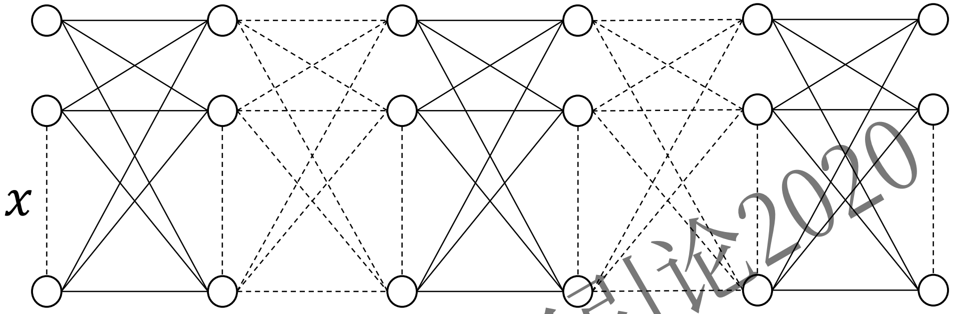
Supervised Learning

Training Data

$D = \{(x, y^L)\}$

A training sample (x, y^L)

Input x



Network prediction   Target

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix} \qquad y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

Cost function

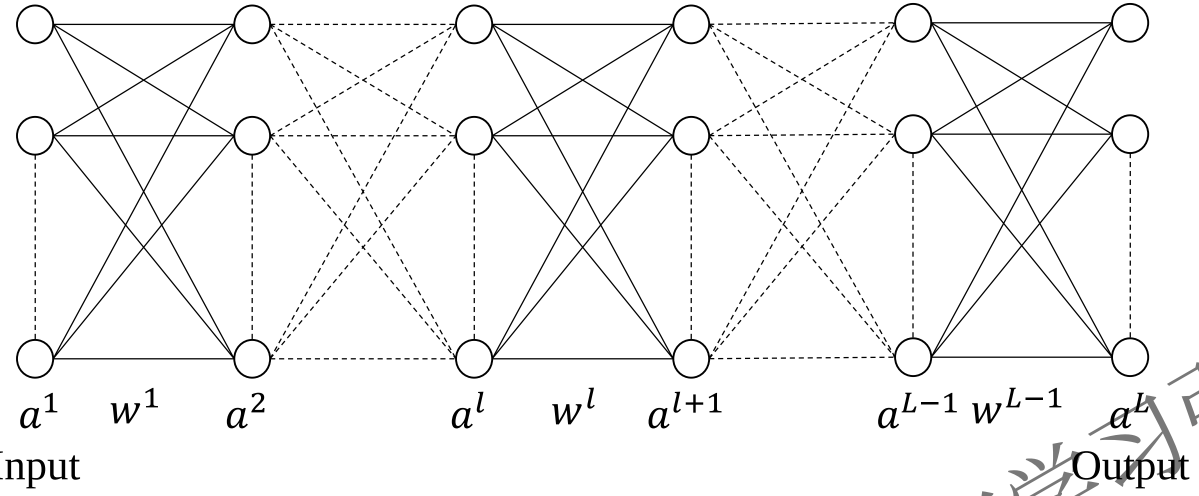
updating the weights: Learning algorithm

$$J(a^L, y^L) = J(w^1, \cdots, w^L)$$

In supervised learning, each training sample contains input and the associated target output.

Problem: How to construct a cost function?

Network Performance: Cost Function



Target

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

Network prediction

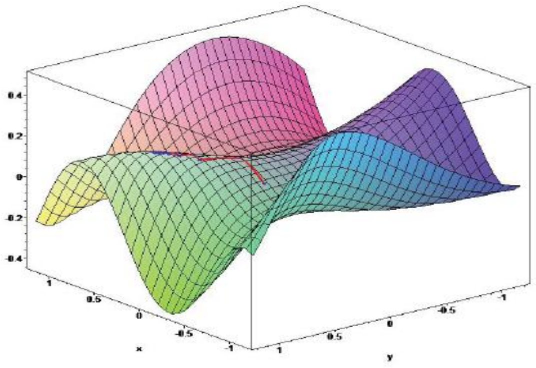
$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

There are many ways to construct cost functions.
A frequently used cost is as follows:

$$e_j = a_j^L - y_j^L$$

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2 = J(w^1, \cdots, w^L)$$

Clearly, J is a function of $w^1, \cdots, w^L$.



A cost function J describes the performance of the network. If the J is small, it implies that the network prediction a^L close to the target y^L, the network is called in good performance. Since J is a function with variables $(w^1, \cdots, w^L)$, good performance means to find suitable $(w^1, \cdots, w^L)$ such that J is small. The process of looking for suitable $(w^1, \cdots, w^L)$ is called network learning.

Problem: How to learn?

Network Performance: Cost Function

Learning is a process such that a^L is close to y^L, i.e., the cost function J reaches minimum. A cost function $J = J(w^1, \cdots, w^{L-1})$ is a function with variables $w^l (l = 1, \cdots, L)$, thus the network learning is to looking for some $w^l (l =$

Target

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

Network prediction

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

A frequently used cost function:

$$J = \frac{1}{2} \sum_{j=1}^{n_L} e_j^2 = J(w^1, \cdots, w^L)$$

J is a function of $w^1, \cdots, w^L$.

Learning = Looking for minimum points $w^l (l = 1, \cdots, L)$ of J

Outline

- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

Minimum Points

General Nonlinear function

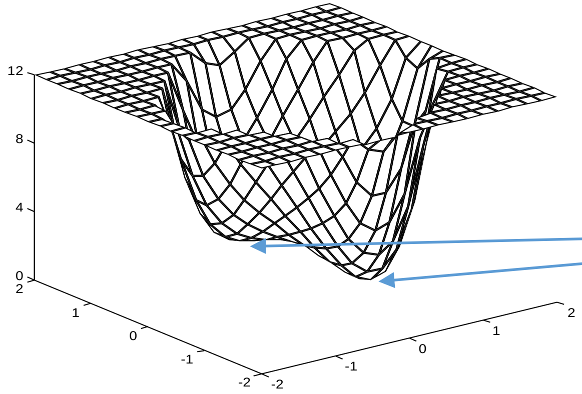
$$J(w), w \in R^n$$

w^* is a minimum point if

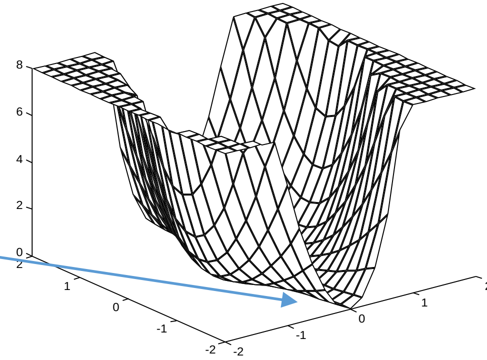
$$J(w^*) \leq J(w)$$

for any w that very close to w^*.

$J(w_1, w_2) = (w_2 - w_1)^4 + 8w_1w_2 - w_1 + w_2 + 3$

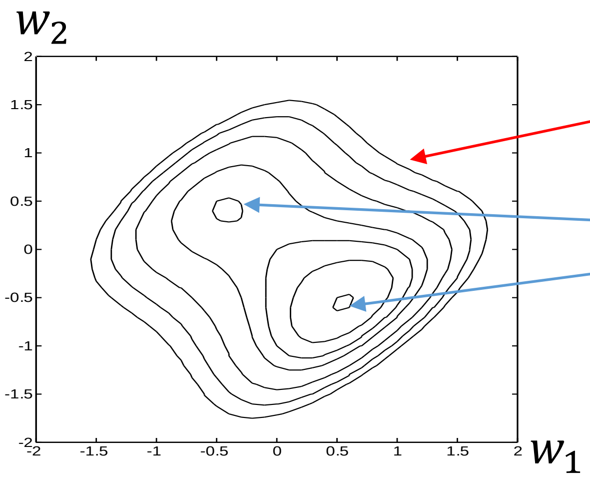


$J(w_1, w_2) = (w_1^2 - 1.5w_1w_2 + 2w_2^2)w_1^2$

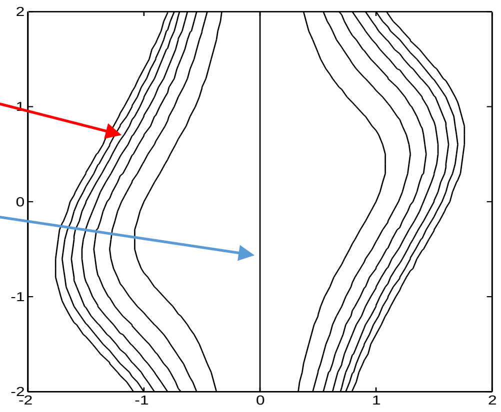


Minima

Contour

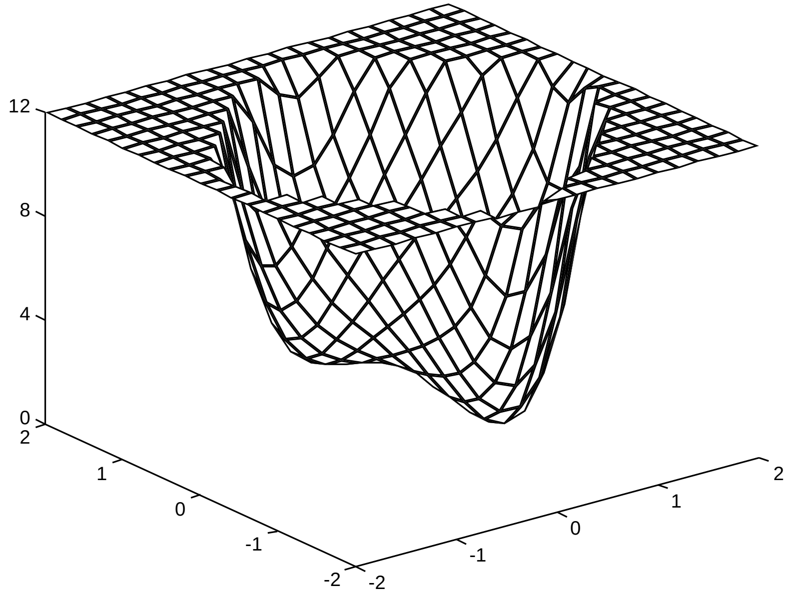


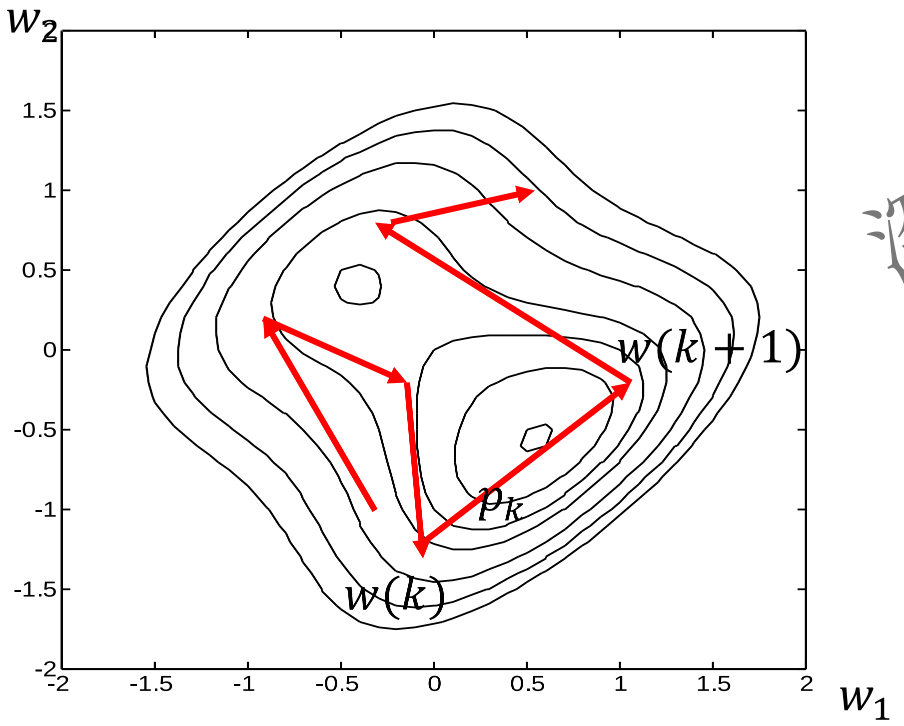
Minimum points



Problem:
How to find the minimum points?

Iteration Method

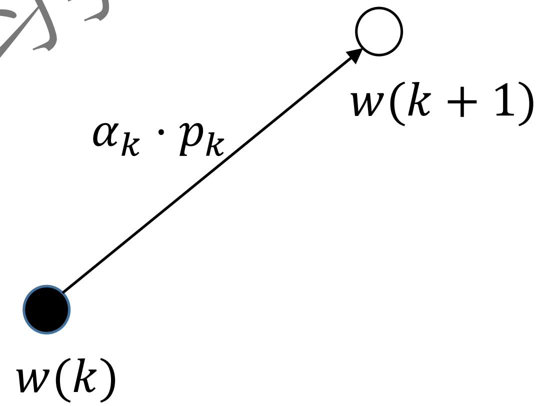




Finding a minimum point step by step

$$w(k+1) = w(k) \ + \alpha_k \cdot p_k$$

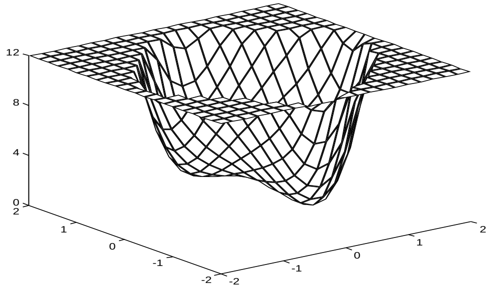
To begin the iteration, you must need a given starting point w_0.



p_k is called searching direction

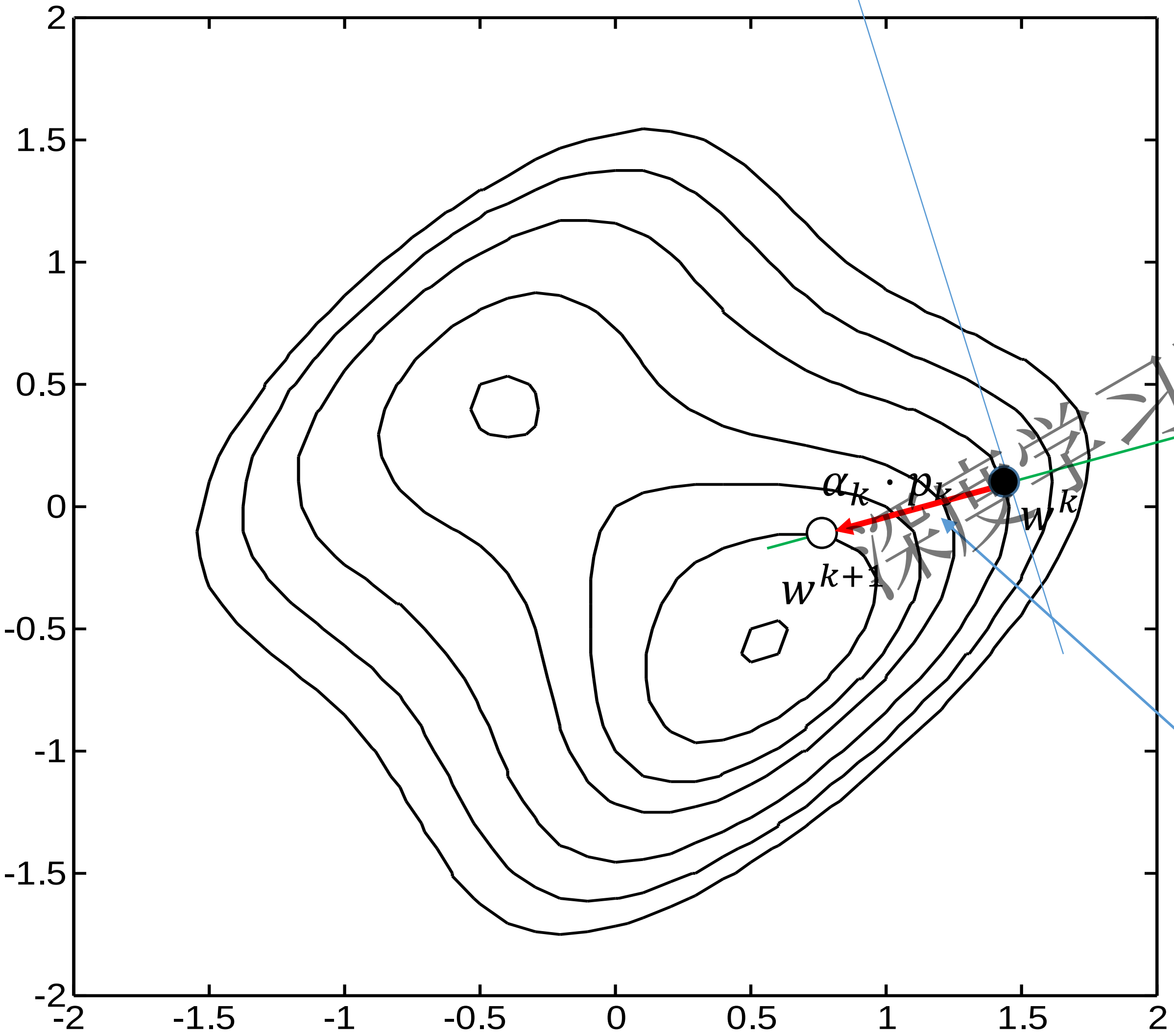
α_k is learning rate at step k.

Problem: How to get the searching direction p_k?



Steepest Descent Method

Slowest changing direction



Fastest increasing direction

Gradient:

$$g_k = \nabla J(w)\Big|_{w(k)} = \frac{\partial J}{\partial w}\Big|_{w(k)} = \begin{pmatrix} \dfrac{\partial J}{\partial w_1} \\ \vdots \\ \dfrac{\partial J}{\partial w_n} \end{pmatrix}\Bigg|_{w(k)}$$

Steepest Descent Algorithm:

$$p_k = -g_k$$

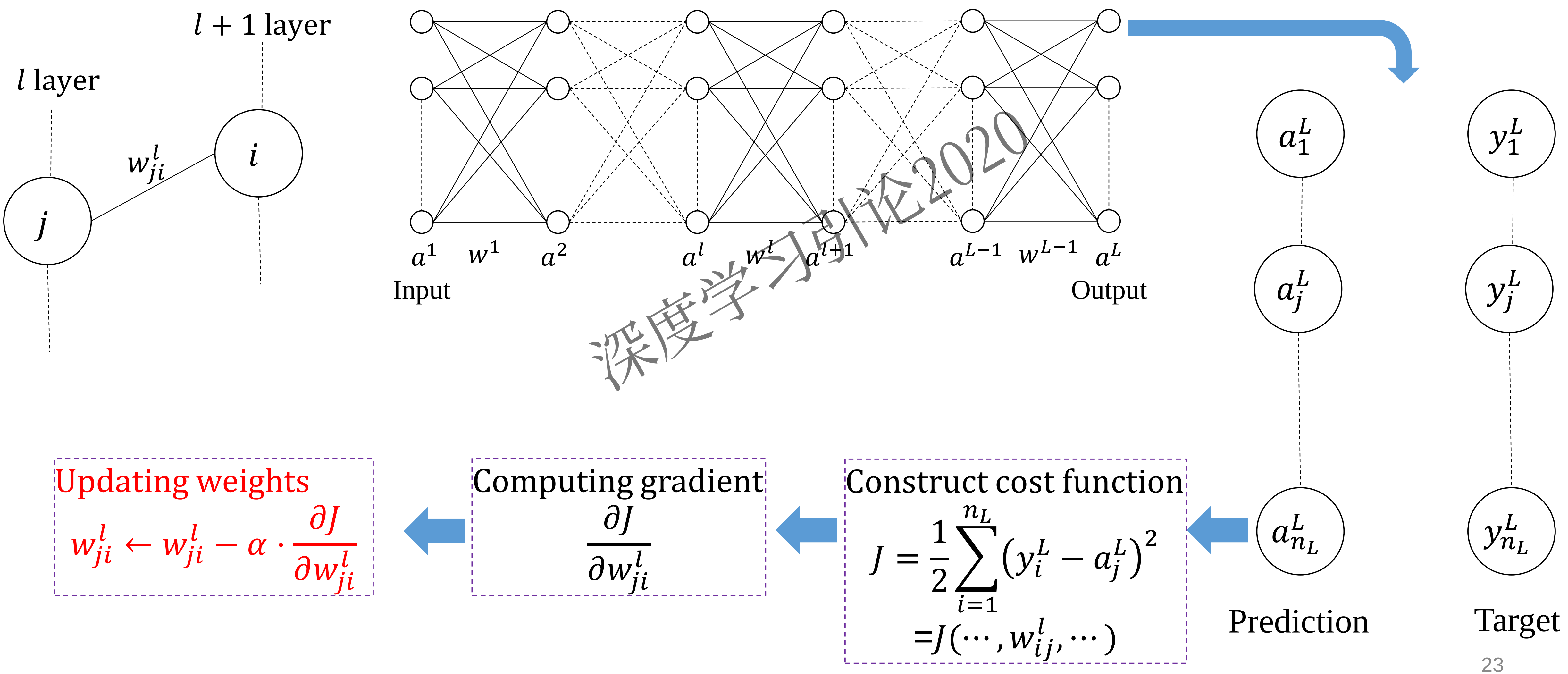
$$w(k+1) = w(k) - \alpha_k \cdot g_k$$

or

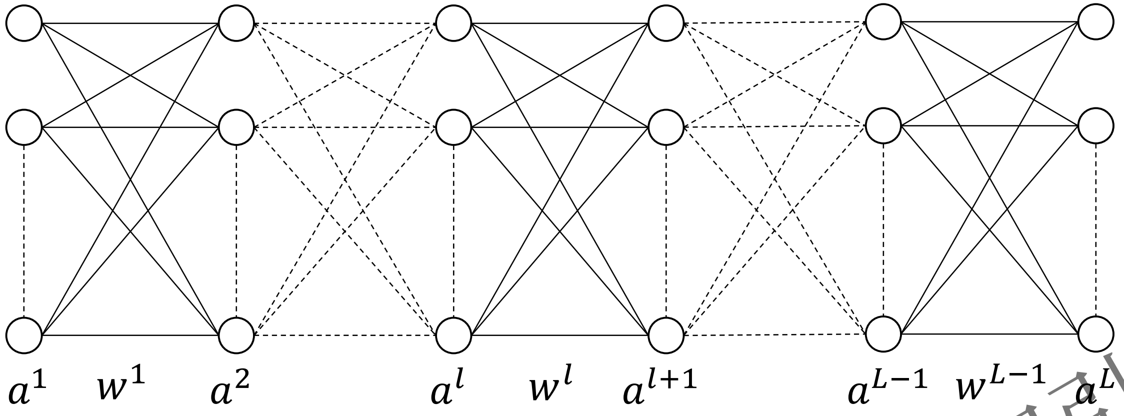
$$w(k+1) = w(k) - \alpha_k \cdot \frac{\partial J}{\partial w}\Big|_{w(k)}$$

Steepest descent direction

Steepest Descent Method



Steepest Descent Method

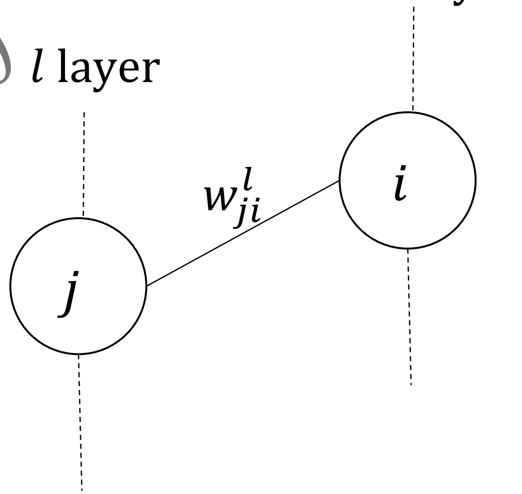


Target

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

Prediction

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$



Steepest Descent Method

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2 = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2 = J(\cdots, w_{ij}^l, \cdots)$$

1. Computing

$$\frac{\partial J}{\partial w_{ji}^l}$$

2. Updating

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$$

$$a^L = f(W^{L-1}a^{L-1}) = f\left(W^{L-1}f\left(W^{L-2}f\left(W^{L-3}\cdots f(W^1a^1)\right)\right)\right)$$

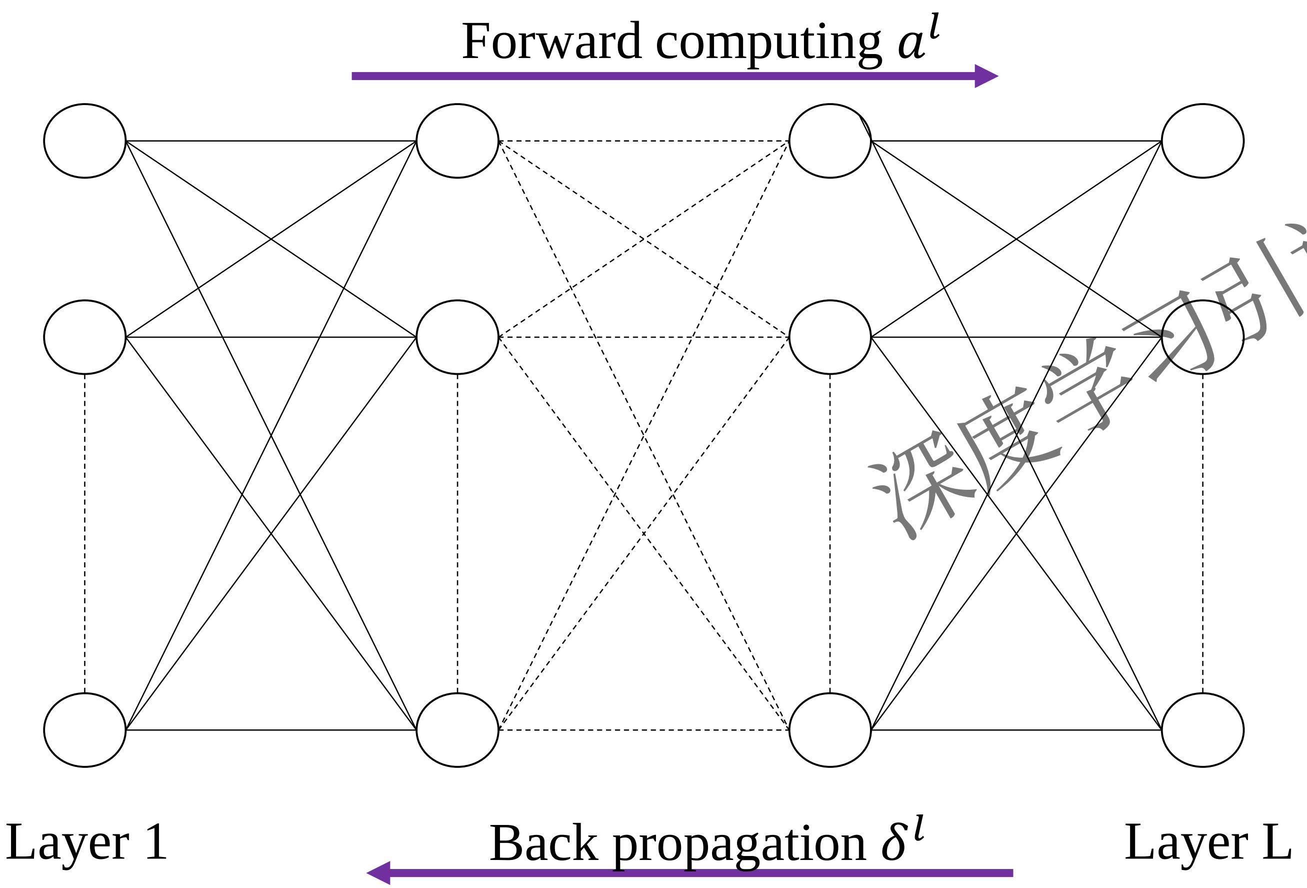
Problem: How to compute $\frac{\partial J}{\partial w_{ji}^l}$?

Answer:
Using the well-known BP method.

Outline

- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

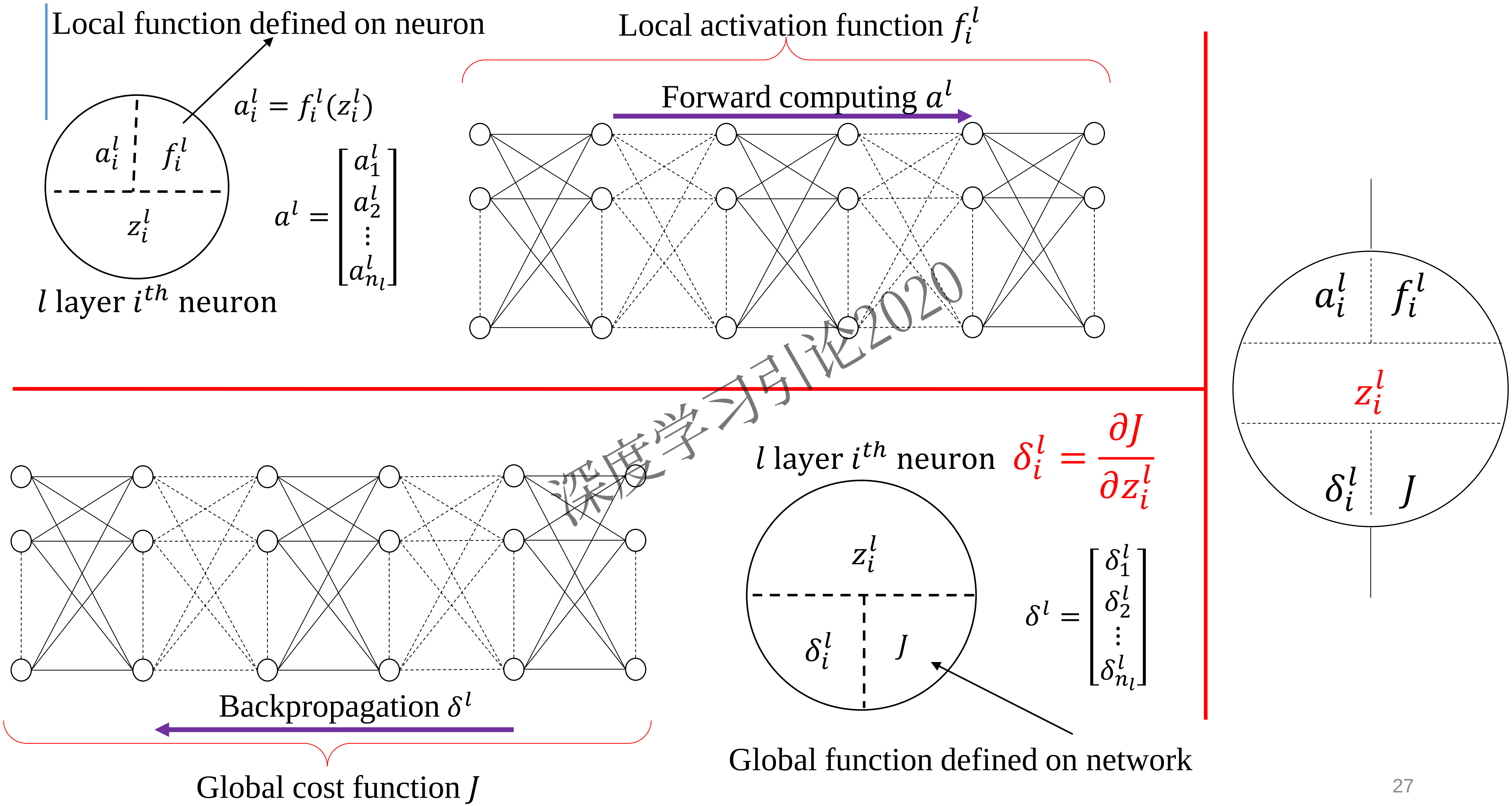
Backpropagation



Backpropagation is a efficient way to calculate $\dfrac{\partial J}{\partial w_{ji}^l}$

Cost function:

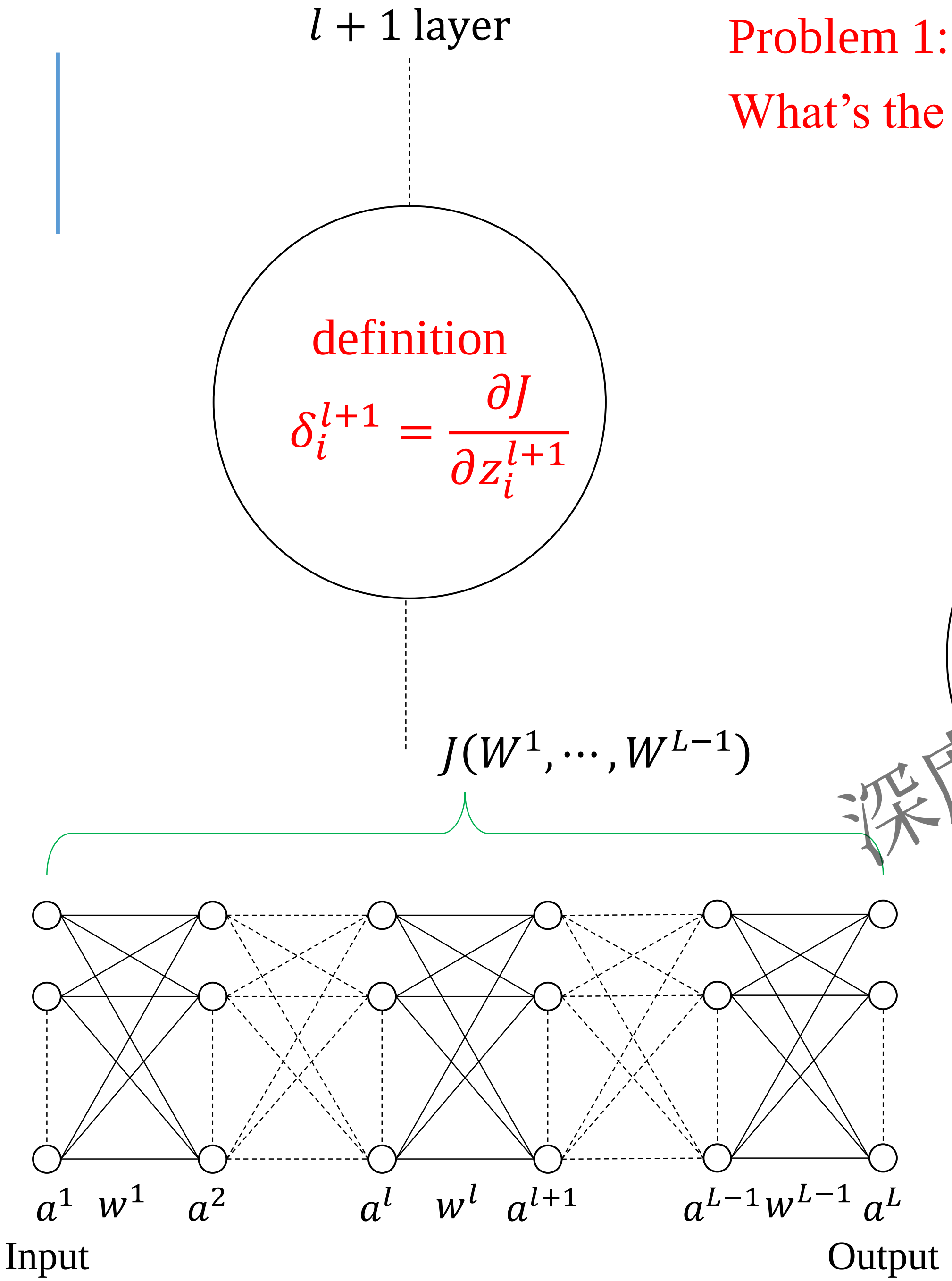
$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2 = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

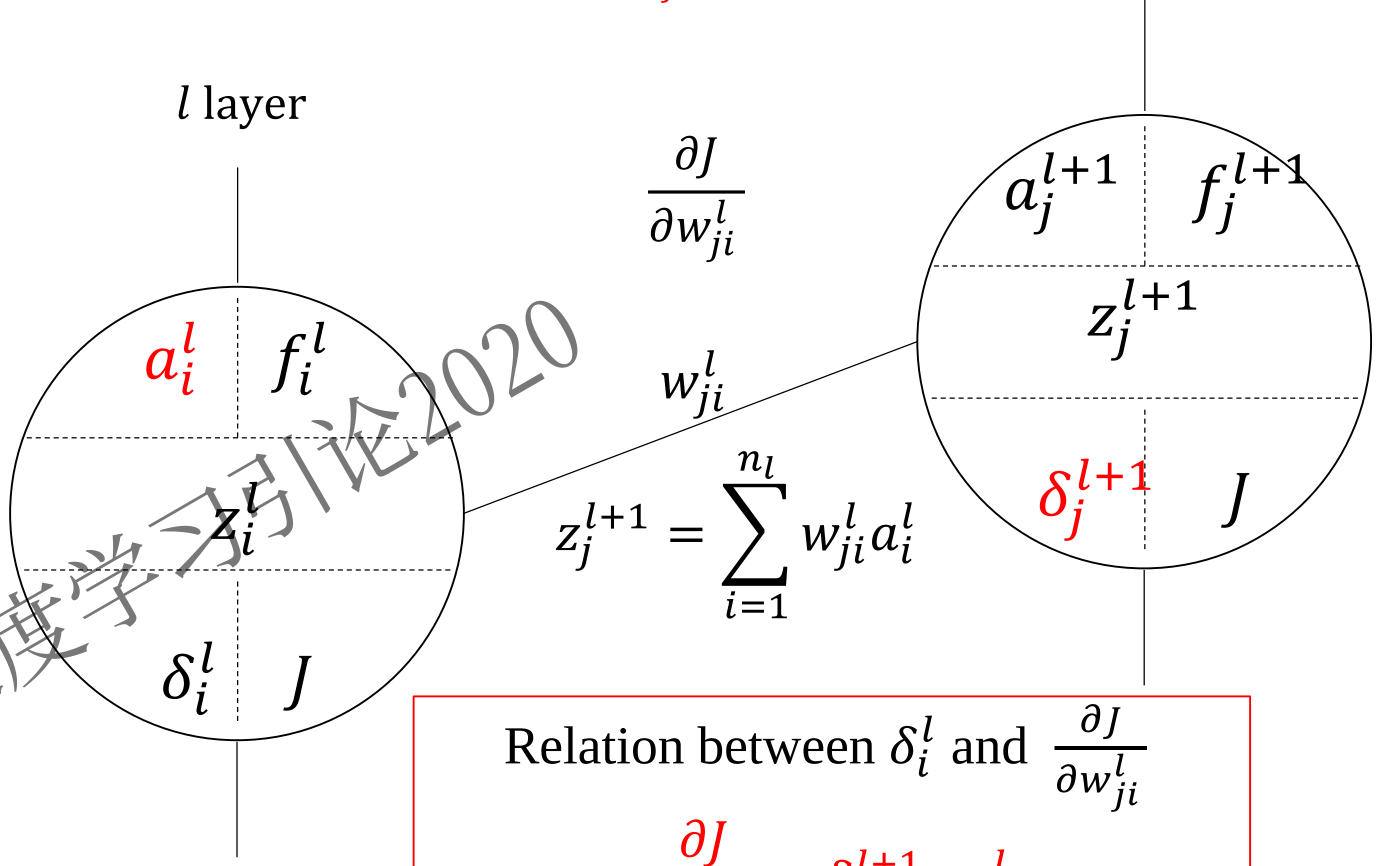


Local function defined on neuron
$a_i^l = f_i^l(z_i^l)$
a_i^l f_i^l
z_i^l
$a^l = \begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_{n_l}^l \end{bmatrix}$
l layer i^{th} neuron
Local activation function f_i^l
Forward computing a^l
a_i^l f_i^l
z_i^l
δ_i^l J
l layer i^{th} neuron $\delta_i^l = \frac{\partial J}{\partial z_i^l}$
z_i^l
δ_i^l J
$\delta^l = \begin{bmatrix} \delta_1^l \\ \delta_2^l \\ \vdots \\ \delta_{n_l}^l \end{bmatrix}$
Backpropagation δ^l
Global cost function J
Global function defined on network

Problem 1:

What's the relation between δ_i^{l+1} and $\frac{\partial J}{\partial w_{ji}^l}$?



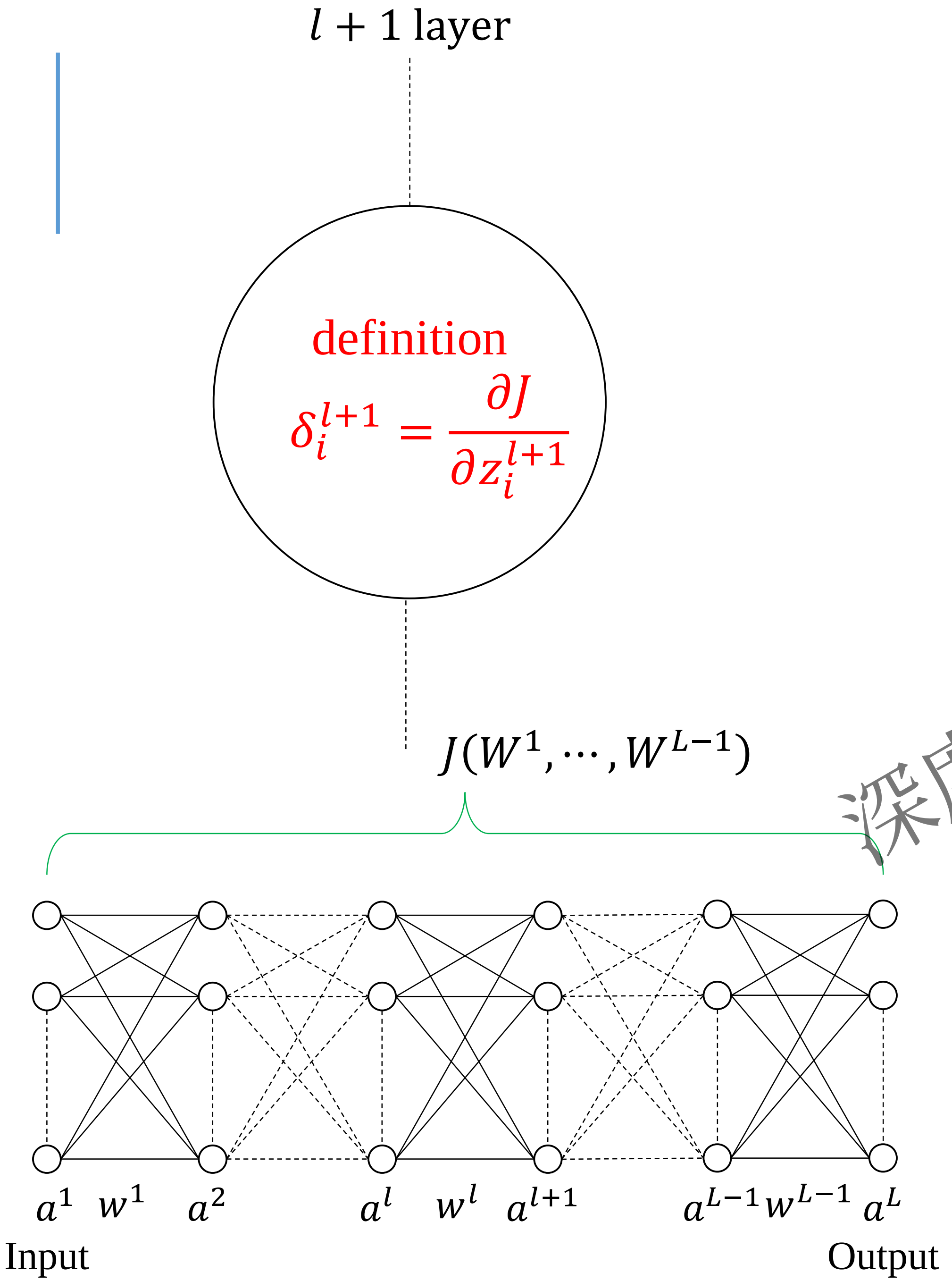


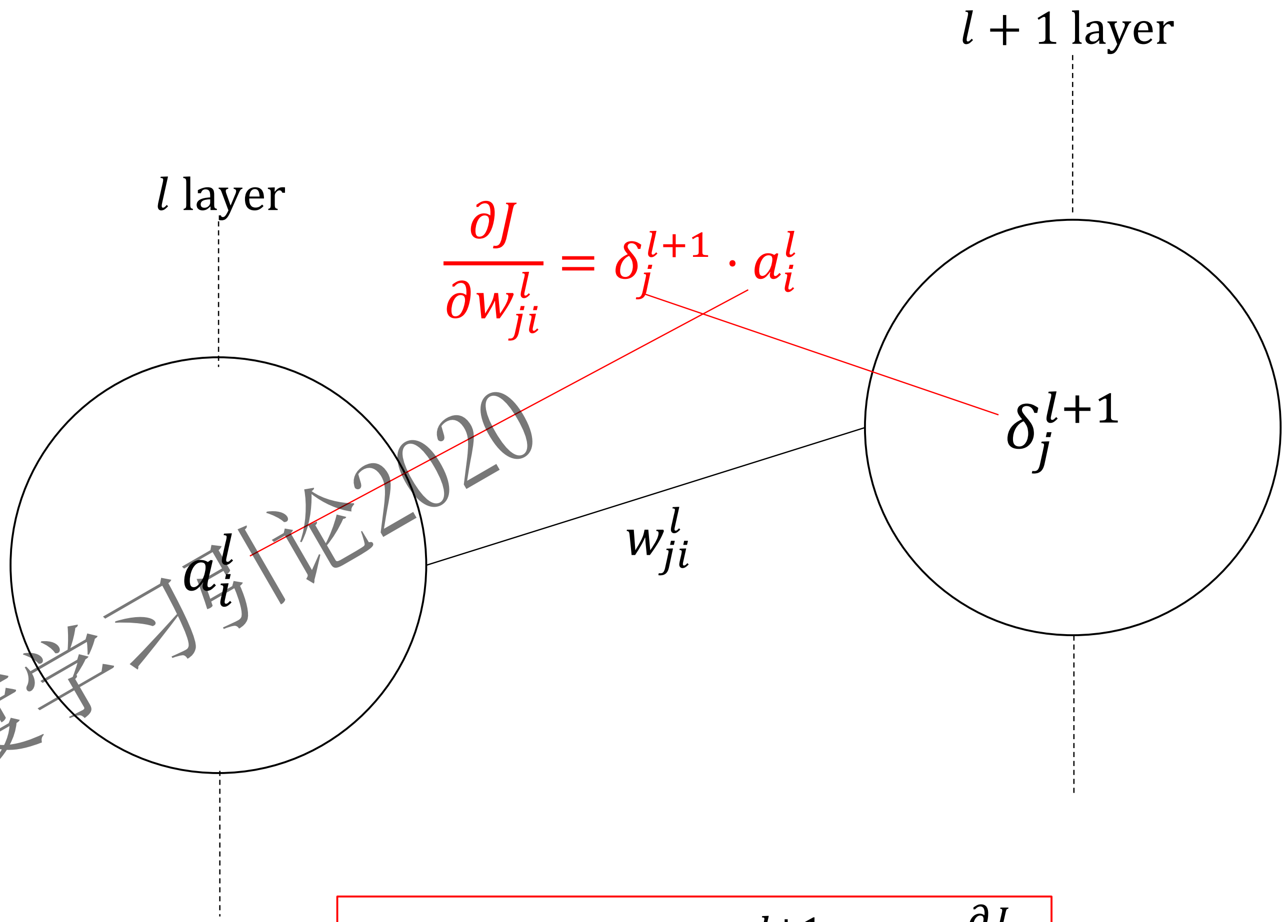
Relation between δ_i^l and $\frac{\partial J}{\partial w_{ji}^l}$

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

Why?

$$\frac{\partial J}{\partial w_{ji}^l} = \frac{\partial J}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$





Relation between δ_i^{l+1} and $\frac{\partial J}{\partial w_{ji}^l}$

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

Problem 2:

How to calculate the last layer's δ_i^L ?

By definition

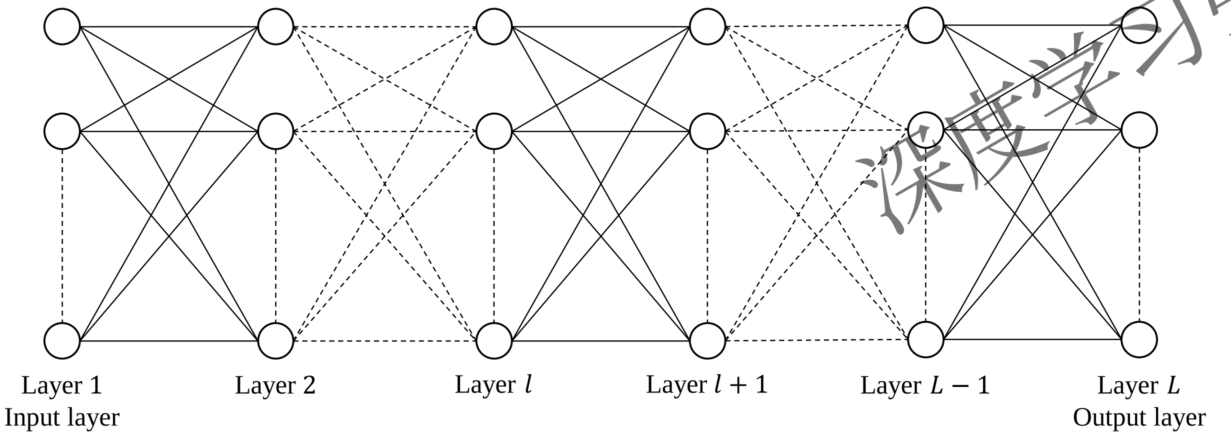
$$\delta_i^L = \frac{\partial J}{\partial z_i^L}$$

If

$$J = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

then,

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \left(a_i^L - y_i^L\right)\cdot\frac{\partial a_i^L}{\partial z_i^L} = \left(a_i^L - y_i^L\right)\cdot \dot{f}\left(z_i^L\right)$$



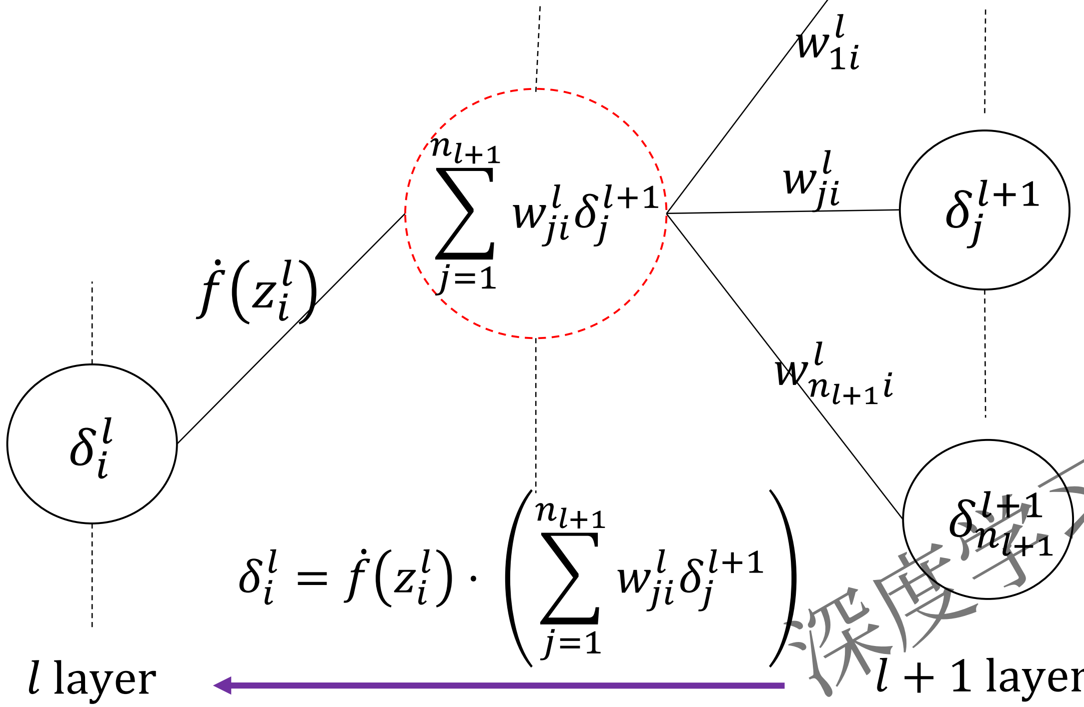
Backpropagation δ^l

δ^{L}

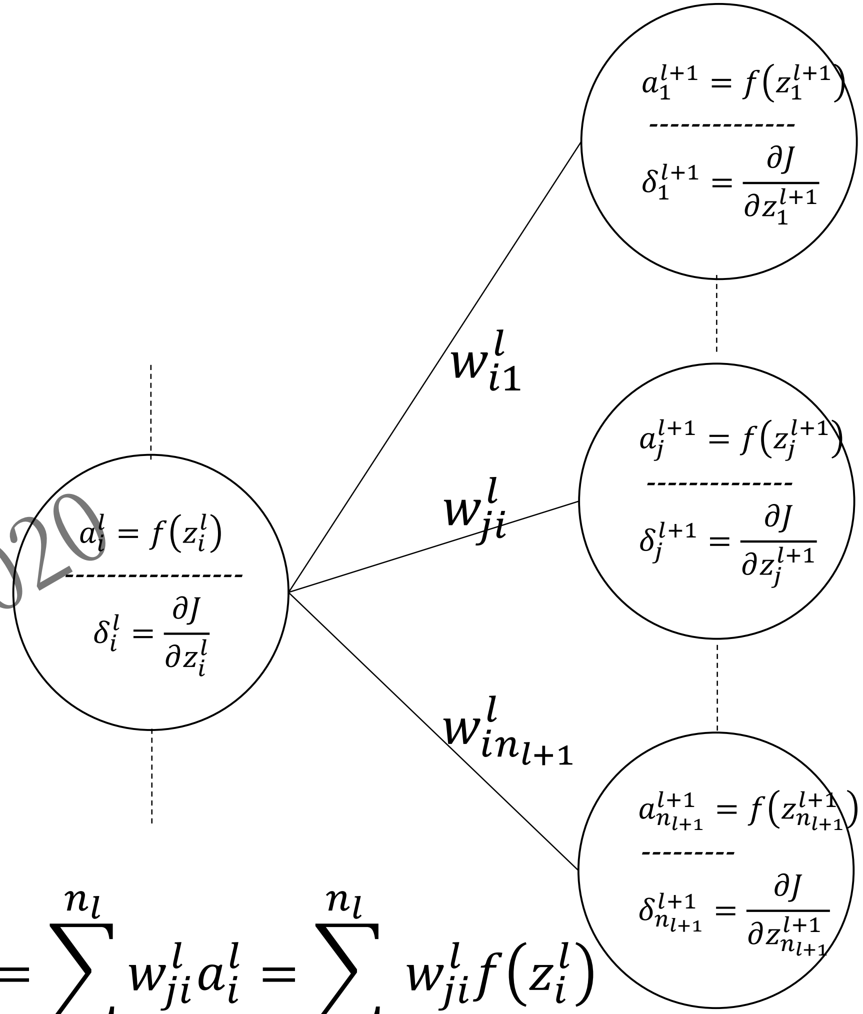
$$a_i^L = f\left(z_i^L\right)$$

L layer i^{th} neuron

Problem 3:
Relation between δ_i^l and δ_j^{l+1}



$$\delta_i^l = \dot{f}(z_i^l)\cdot\left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1}\right)$$



$$z_j^{l+1} = \sum_{i=1}^{n_l} w_{ji}^l a_i^l = \sum_{i=1}^{n_l} w_{ji}^l f(z_i^l)$$

$$\delta_i^l = \frac{\partial J}{\partial z_i^l} = \sum_{j=1}^{n_{l+1}} \frac{\partial J}{\partial z_j^{l+1}}\cdot\frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_{j=1}^{n_{l+1}} \delta_j^{l+1}\cdot\frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_{j=1}^{n_{l+1}} \delta_j^{l+1}\cdot w_{ji}^l\,\dot{f}(z_i^l) = \dot{f}(z_i^l)\cdot\left(\sum_{j=1}^{n_{l+1}} w_{ji}^l\cdot\delta_j^{l+1}\right)$$

Outline

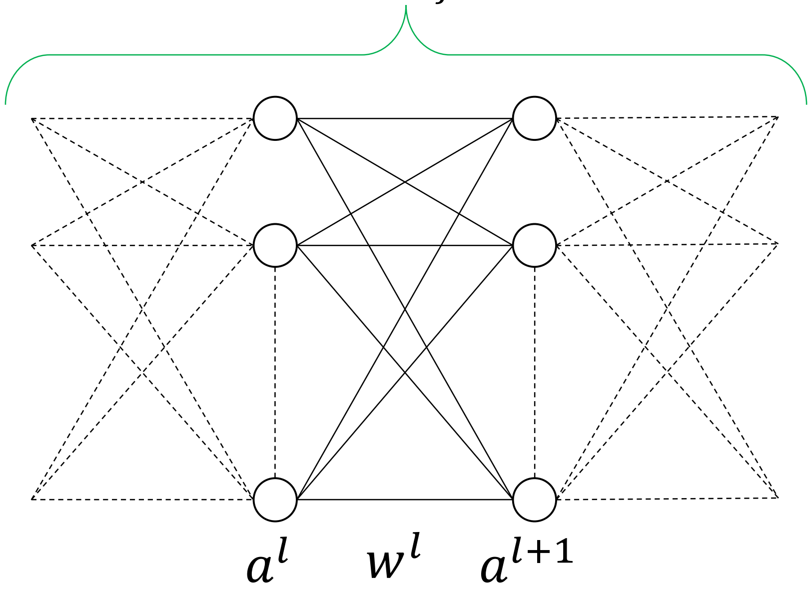
- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

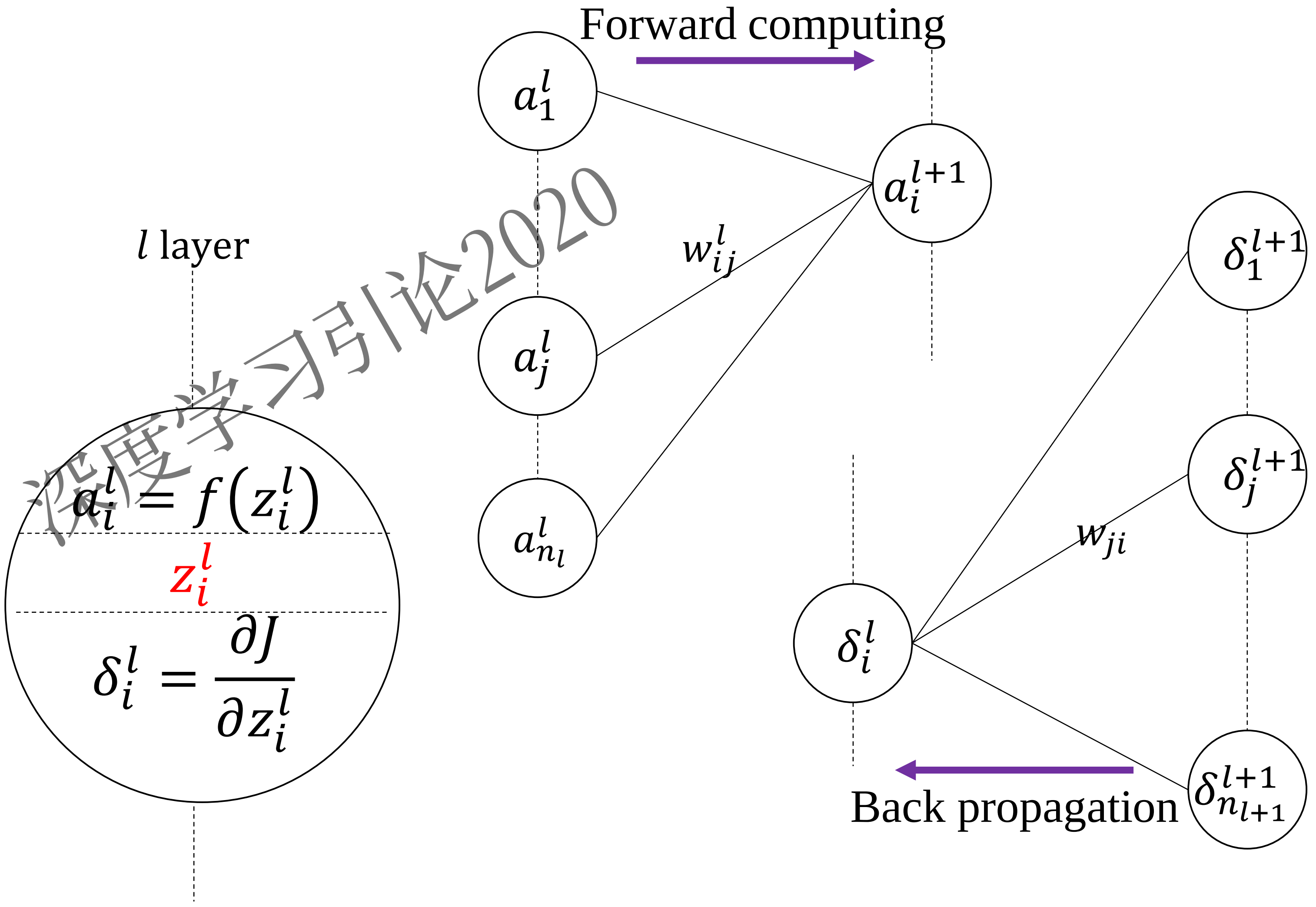
Three Pages to Understand BP: *The first page*

Cost function: $J(w^1, \cdots, w^L)$

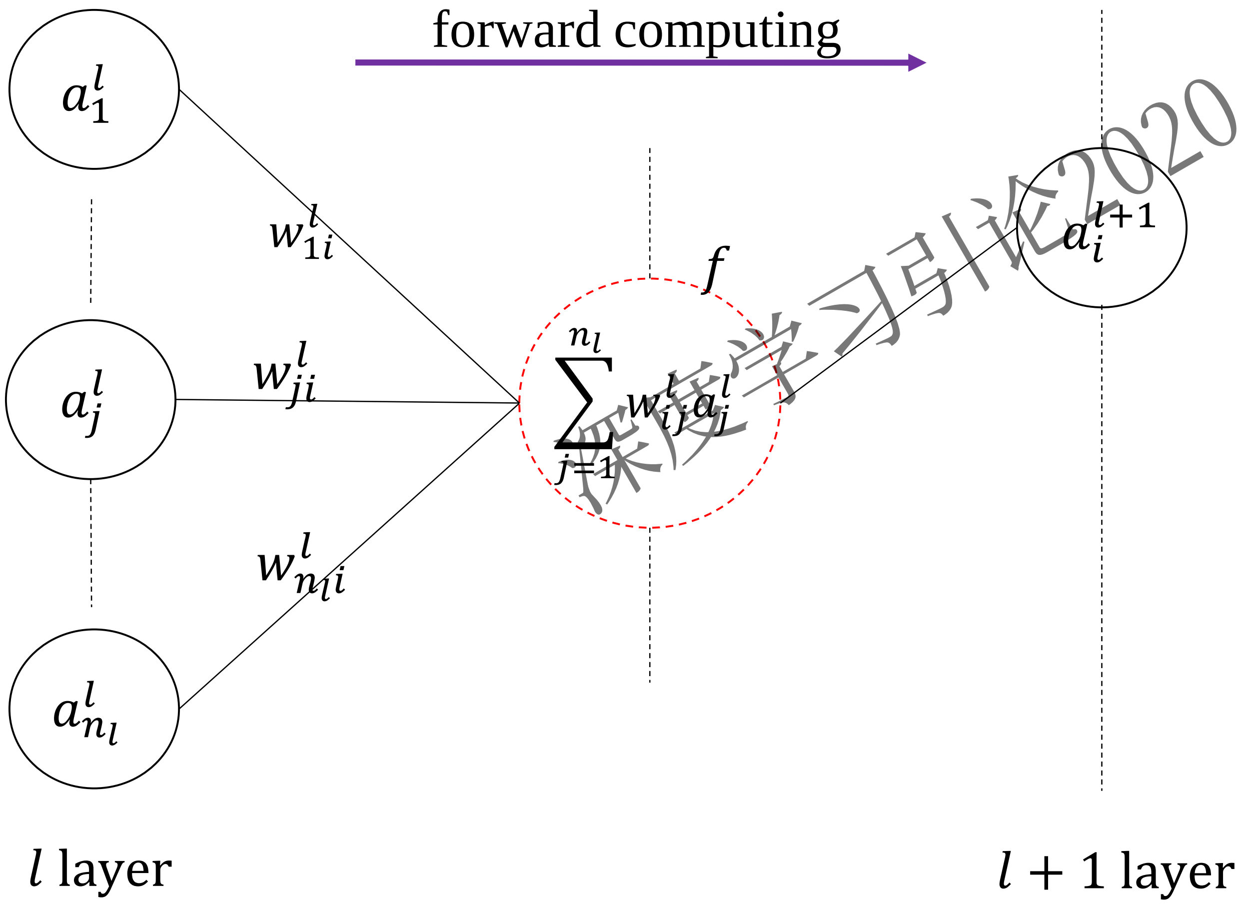
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Relationship: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$





Three Pages to Understand BP: *The second page*



$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$

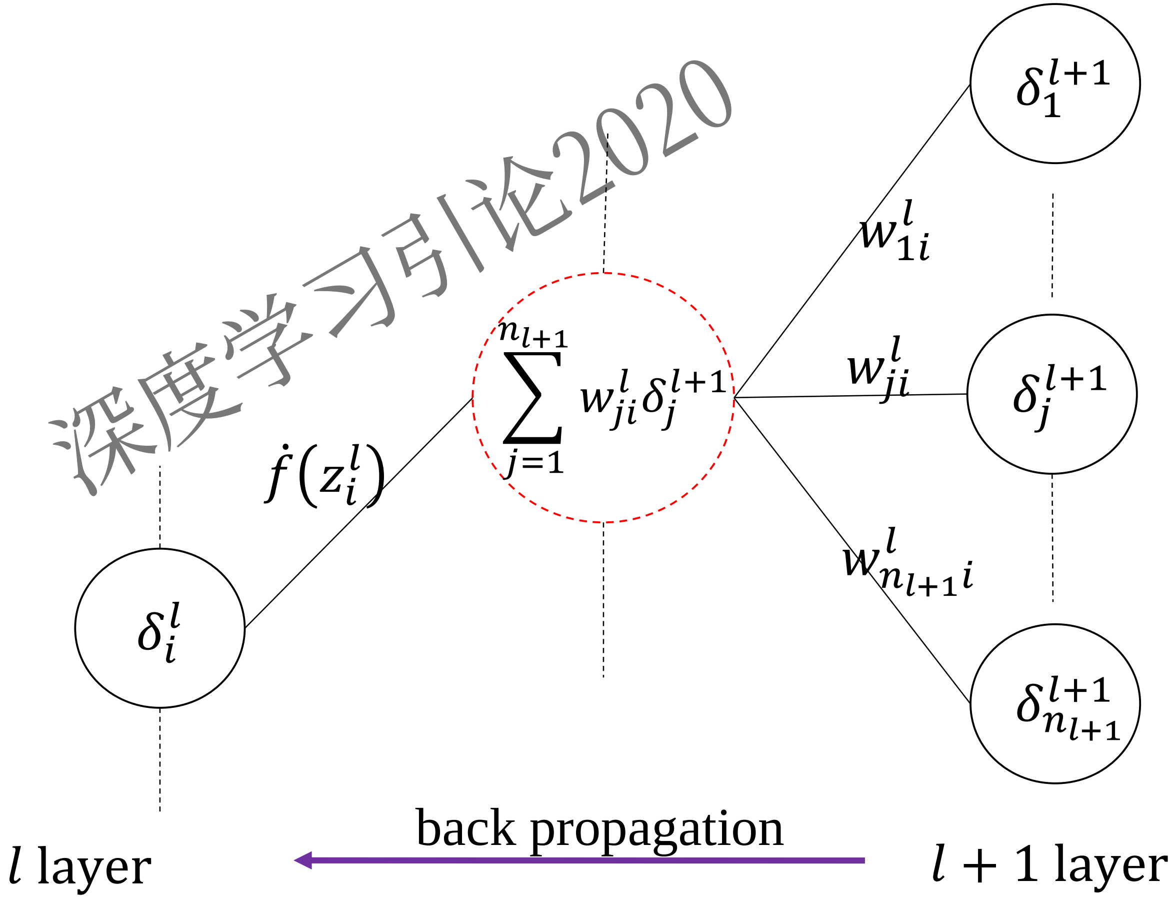
or --------------------

$$a_i^{l+1} = f\left(z_i^{l+1}\right)$$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

Three Pages to Understand BP: *The third page*

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$



Outline

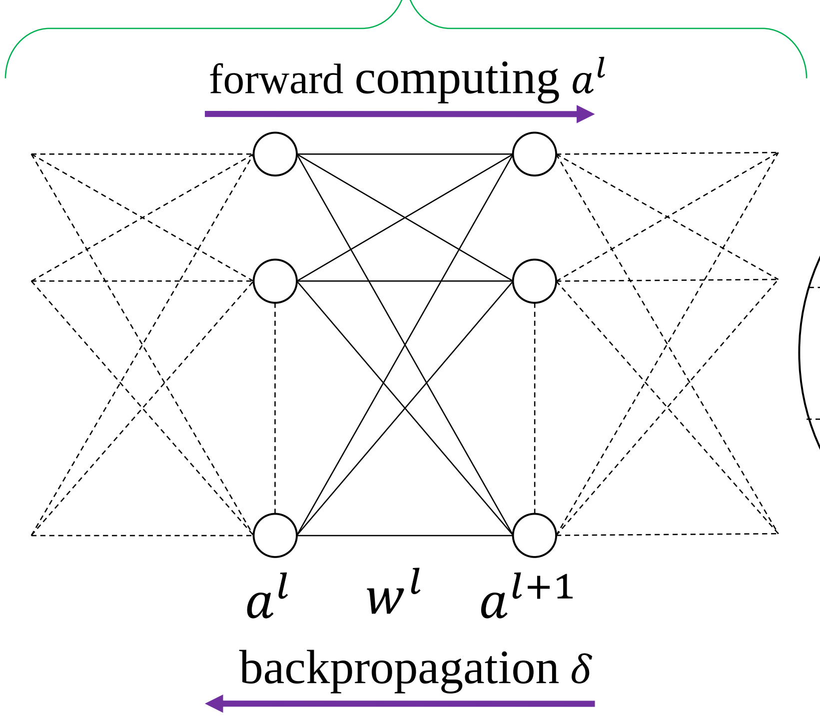
- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

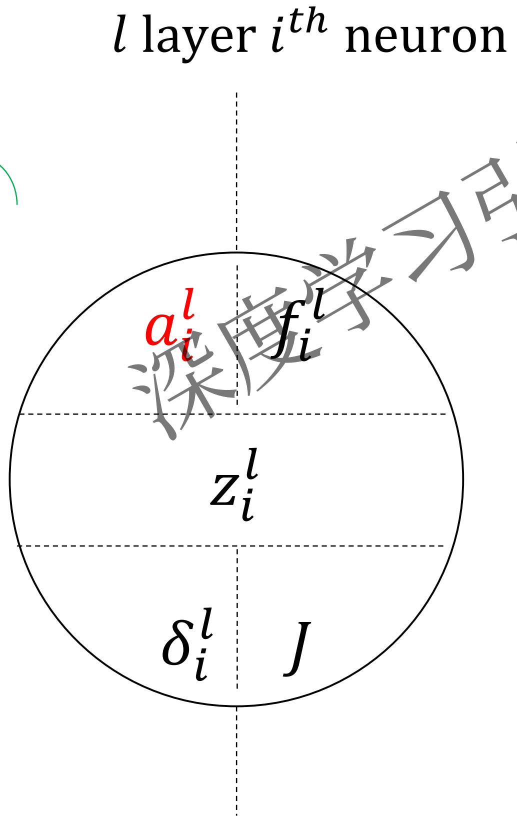
Only One Page to Understand BP

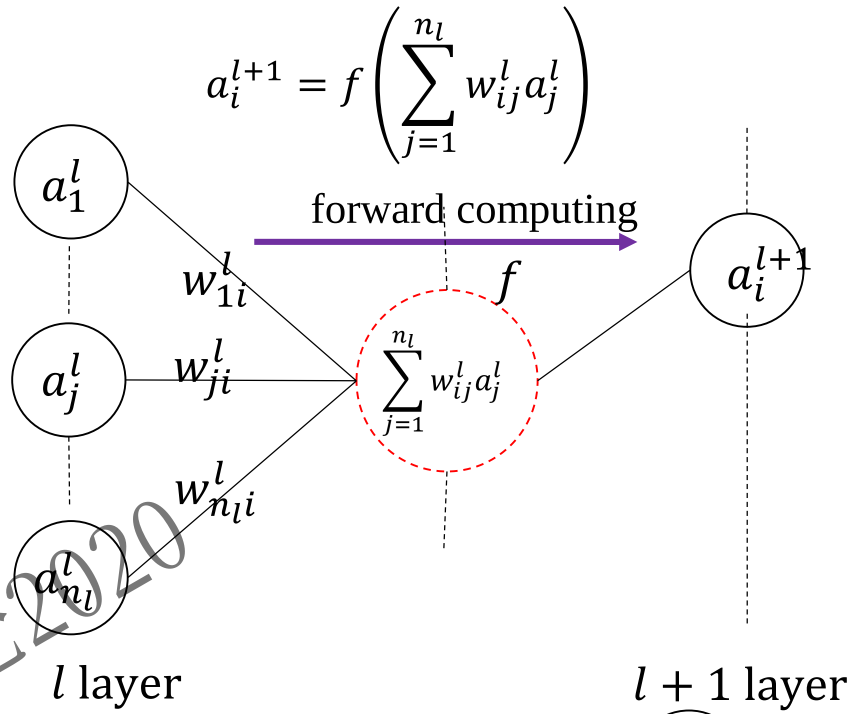
Cost function: $J(w^1, \cdots, w^L)$

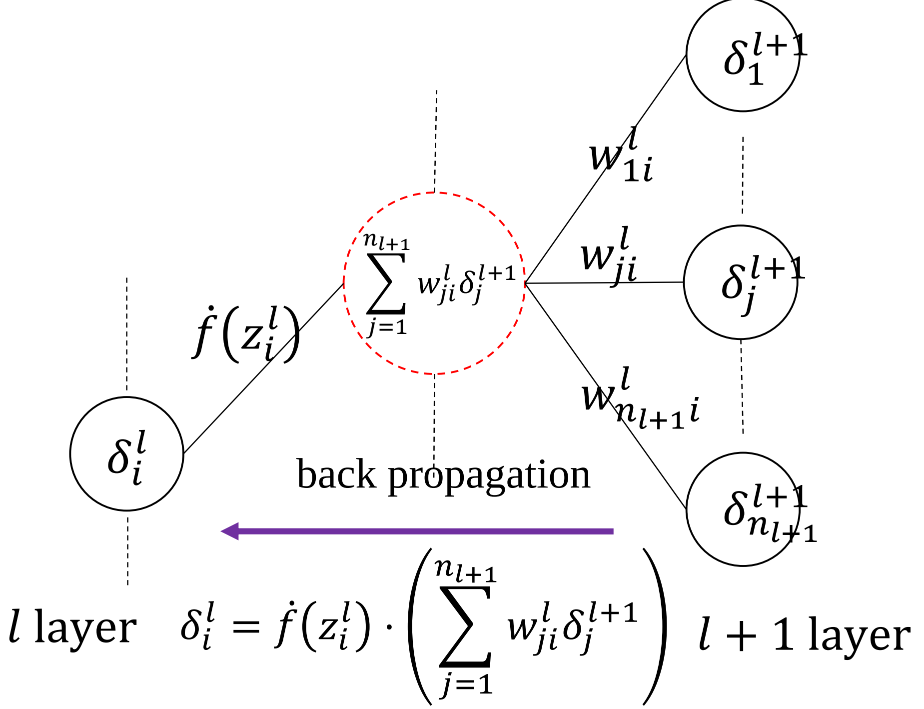
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Relationship: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$





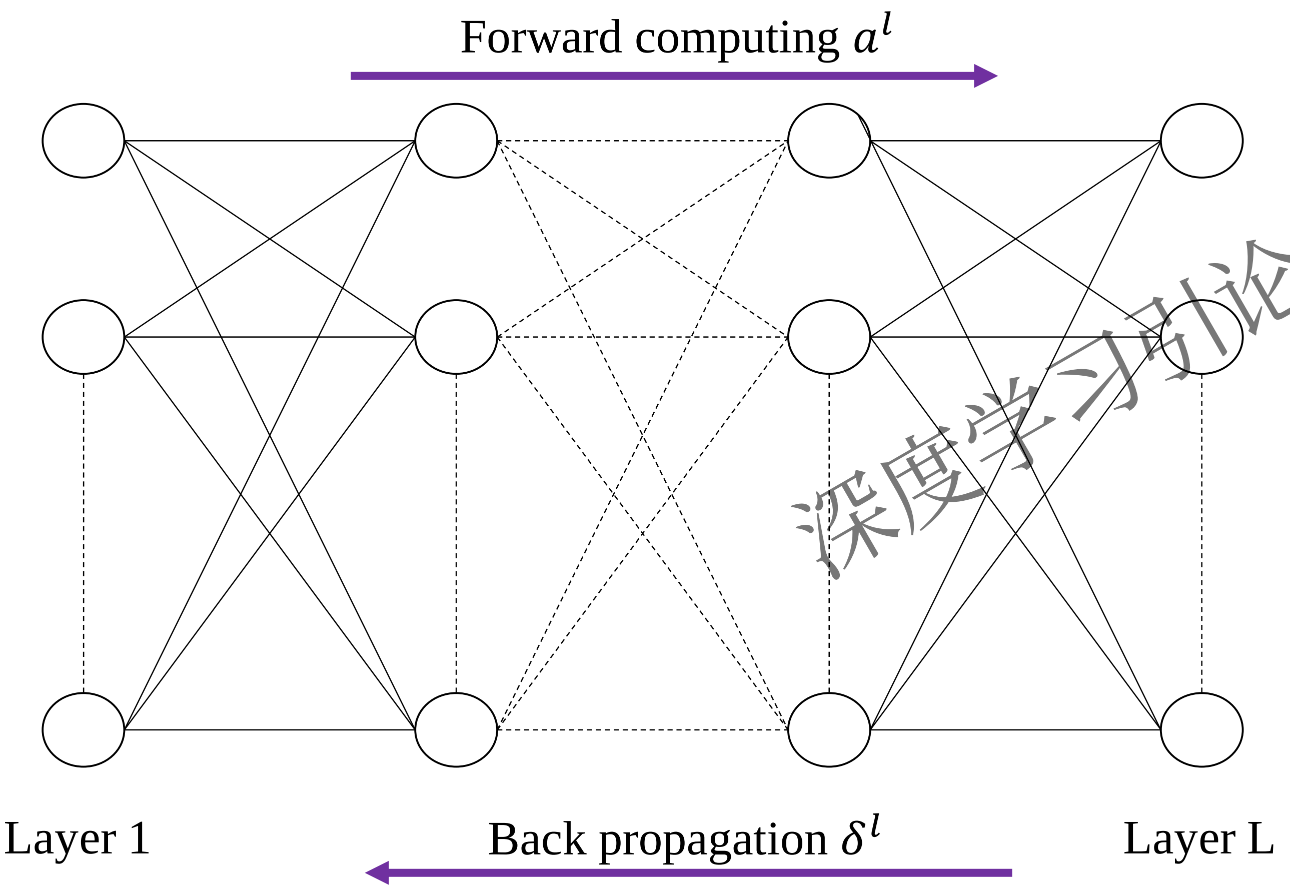


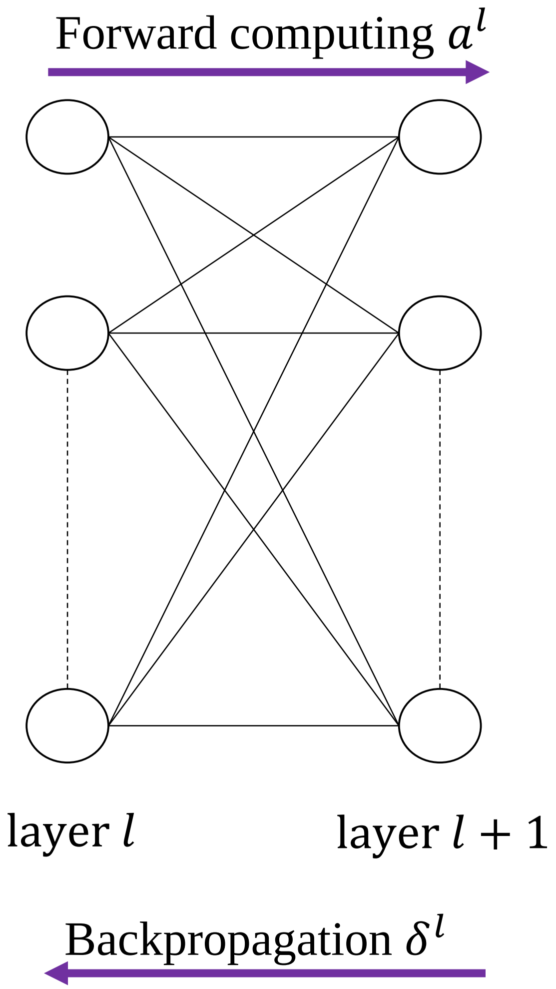


Outline

- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

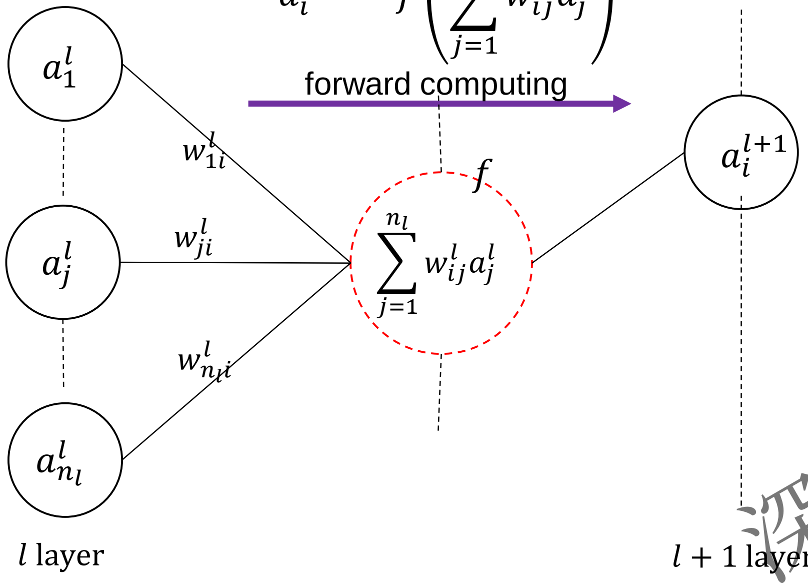
The BP Algorithm





The BP Algorithm

$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$



function $fc(W^l, a^l)$

$for\ i = 1{:}n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

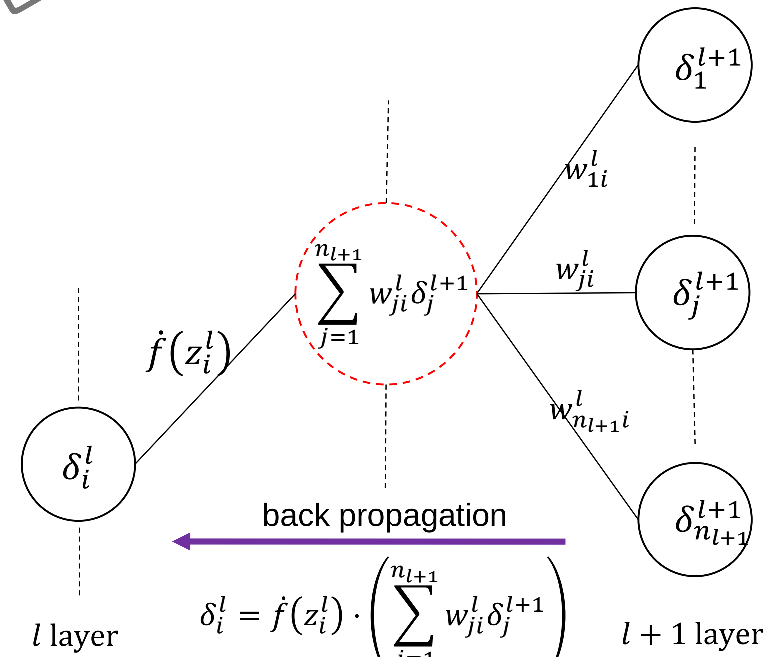
end

function $bc(W^l, \delta^{l+1})$

$for\ i = 1{:}n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1}\right)$$

end



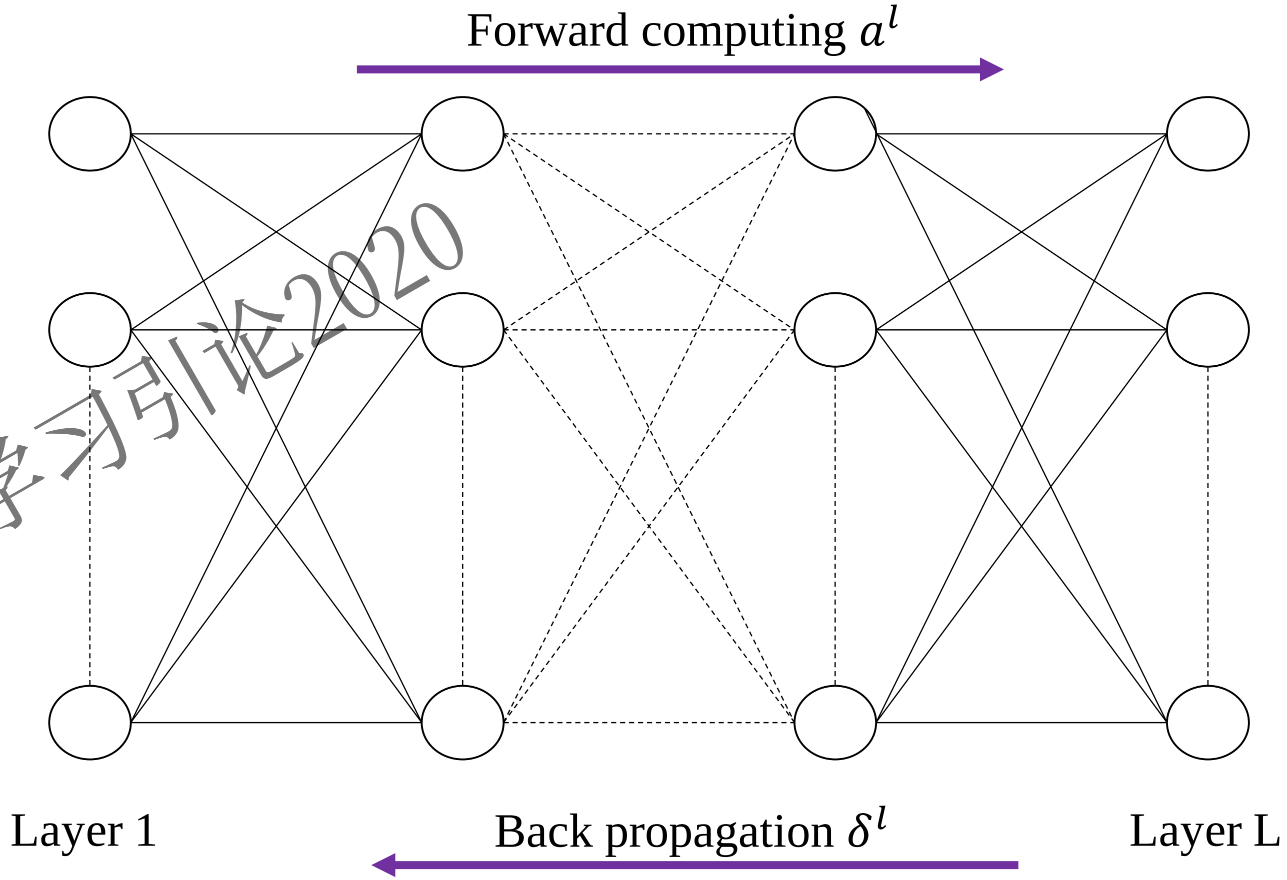
$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1}\right)$$

The BP Algorithm

Training Data

$D = \{(x, y^L) | m \text{ samples}\}$

x: input sample
y^L: target output



There are two ways to train the network.

1. Online training: For each sample $(x, y) \in D$, define a cost function, for example, as

$$J(x, y) = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

2. Batch training: Define cost function as

$$J = \frac{1}{m}\sum_{(x,y)\in D} J(x, y)$$

The BP Algorithm

Online BP Algorithm:

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initial each w_{ij}^l, and choose a learning rate α.

Step 3. Choose a sample $(x, y^L) \in D$, define $J(x, y^L)$, set $a^1 = x$

for $l = 1{:}L$

$fc(w^l, a^l)$;

end

$$\delta^L = \frac{\partial J(x, y^L)}{\partial z^L};$$

for $l = L - 1{:}1$

$bc(w^l, \delta^{l+1})$;

end

Step 4. Updating

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J(x, y)}{\partial w_{ji}^l}$$

Step 5. Return to Step 3 until each w^lconverge.

function $fc(w^l, a^l)$

$for\ i = 1{:}n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

function $bc(w^l, \delta^{l+1})$

$for\ i = 1{:}n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

The BP Algorithm

<u>*Batch BP Algorithm:*</u>

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initial each w_{ij}^l, and choose a learning rate α.

Step 3. For each sample $(x, y^L) \in D$, set $a^1 = x$

for $l = 1: L$

$fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 1$

$bc(w^l, \delta^{l+1})$;

end

$$\frac{\partial J}{\partial w_{ji}^l} \leftarrow \frac{\partial J}{\partial w_{ji}^l} + \delta_j^{l+1} \cdot a_i^l$$

Step 4. Updating

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$$

Step 5. Return to Step 3 until each w^lconverge.

function $fc(w^l, a^l)$

$for\ i = 1: n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

function $bc(w^l, \delta^{l+1})$

$for\ i = 1: n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

Outline

- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

Assignment

Assignment 1: Encoding the BP algorithms.

Function $fc\left(W^l, a^l\right)$

$for\ i = 1{:}\,n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f\left(z_i^{l+1}\right)$$

end

Function $bc\left(W^l, \delta^{l+1}\right)$

$for\ i = 1{:}\,n_l$

$$\delta_i^l = \dot{f}\left(z_i^l\right) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1}\right)$$

end

Batch BP Algorithm:

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initial each w_{ij}^l, and choose a learning rate α.

Step 3. For each sample $(x, y^L) \in D$, set $a^1 = x$

for $l = 1{:}\,L$

$fc\left(W^l, a^l\right)$;

end

$\delta^L = \dfrac{\partial J}{\partial z^L}$;

for $l = L - 1{:}\,1$

$bc\left(W^l, \delta^{l+1}\right)$;

end

$$\frac{\partial J}{\partial w_{ji}^l} \leftarrow \frac{\partial J}{\partial w_{ji}^l} + \delta_j^{l+1} \cdot a_i^l$$

Step 4. Updating

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$$

Step 5. Return to Step 3 until each w^l converge.

Online BP Algorithm:

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initial each w_{ij}^l, and choose a learning rate α.

Step 3. Choose a sample $(x, y^L) \in D$, define $J(x, y^L)$, set $a^1 = x$

for $l = 1{:}\,L$

$fc\left(W^l, a^l\right)$;

end

$\delta^L = \dfrac{\partial J(x, y^L)}{\partial z^L}$;

for $l = L - 1{:}\,1$

$bc\left(W^l, \delta^{l+1}\right)$;

end

Step 4. Updating

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J(x, y^L)}{\partial w_{ji}^l}$$

Step 5. Return to Step 3 until each w^l converge.

Thanks