Understanding Deep Neural Networks

Chapter Three

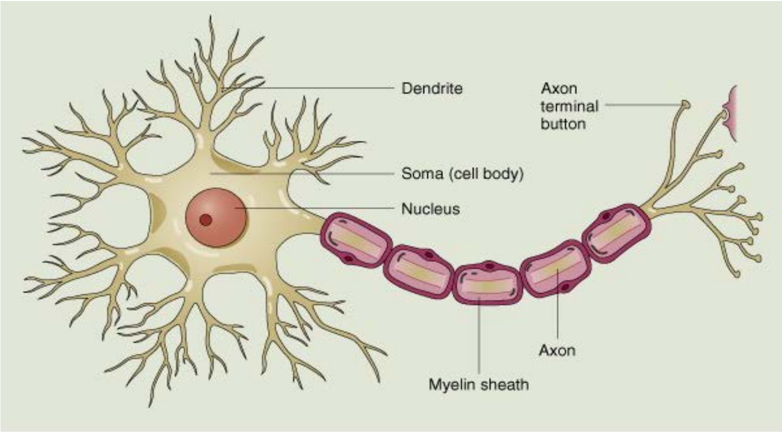
Backpropagation Algorithm

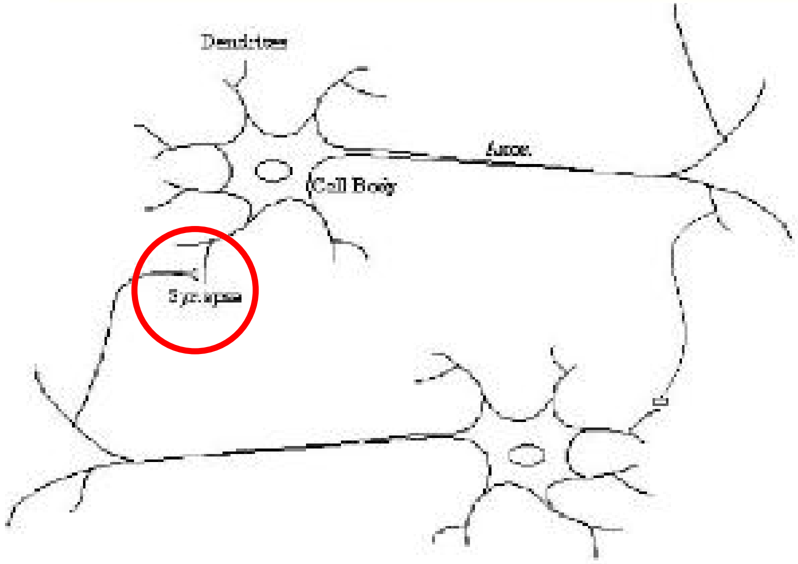
Zhang Yi, *IEEE Fellow*
Autumn 2019

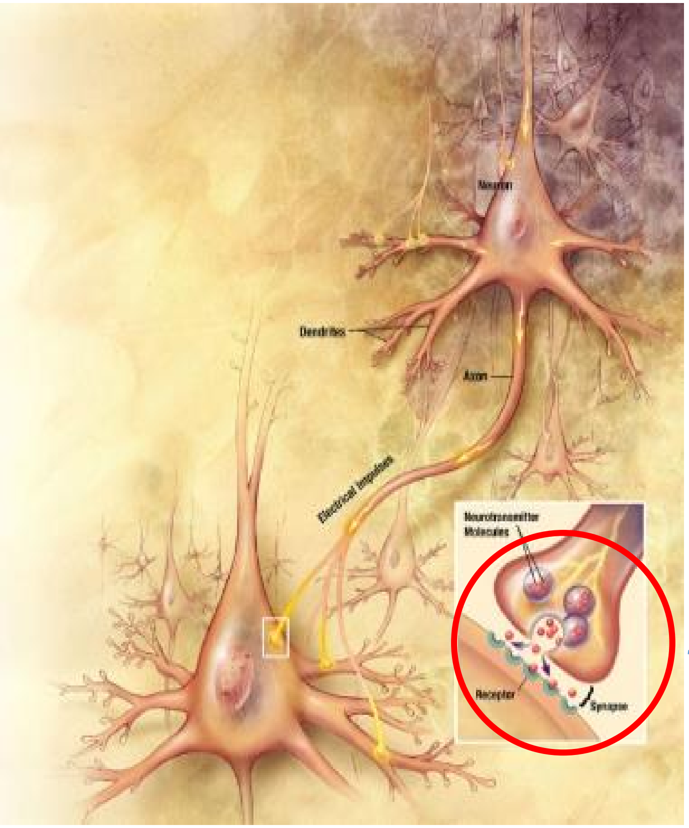
Outline

- 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

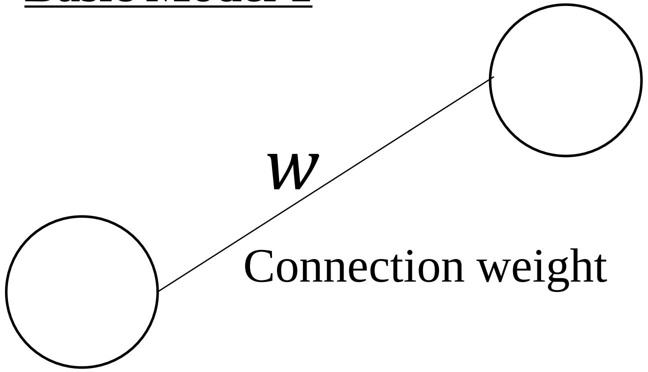
Computational Model of Neurons





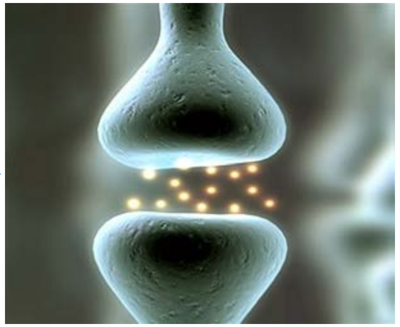


Basic Model 1

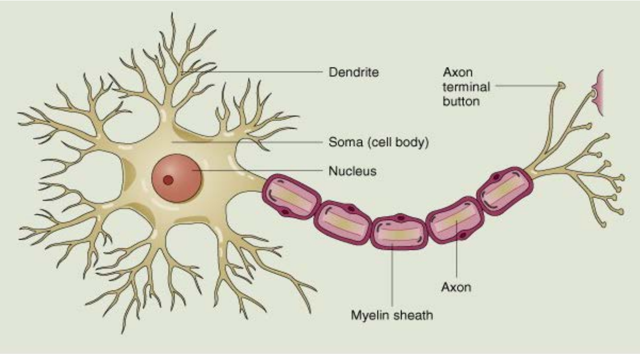


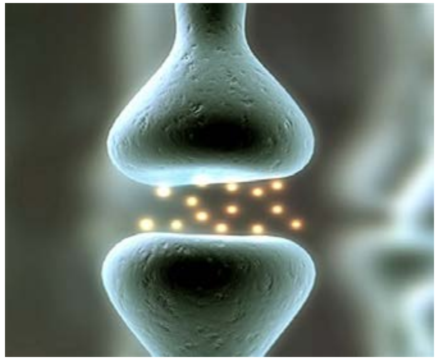
$w > 0, exciting\ connection$
$w = 0, no\ connection$
$w < 0, inhibition\ connection$

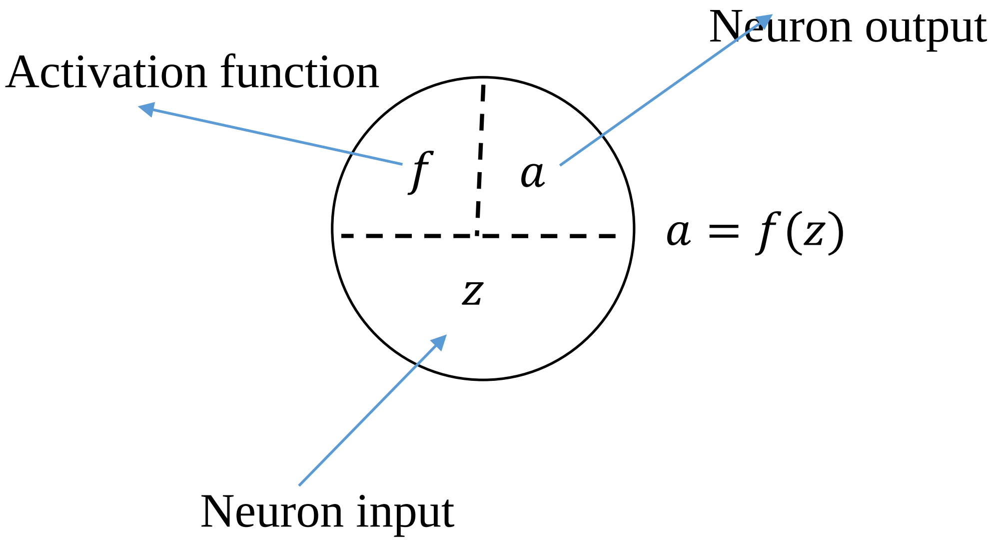
Synapse

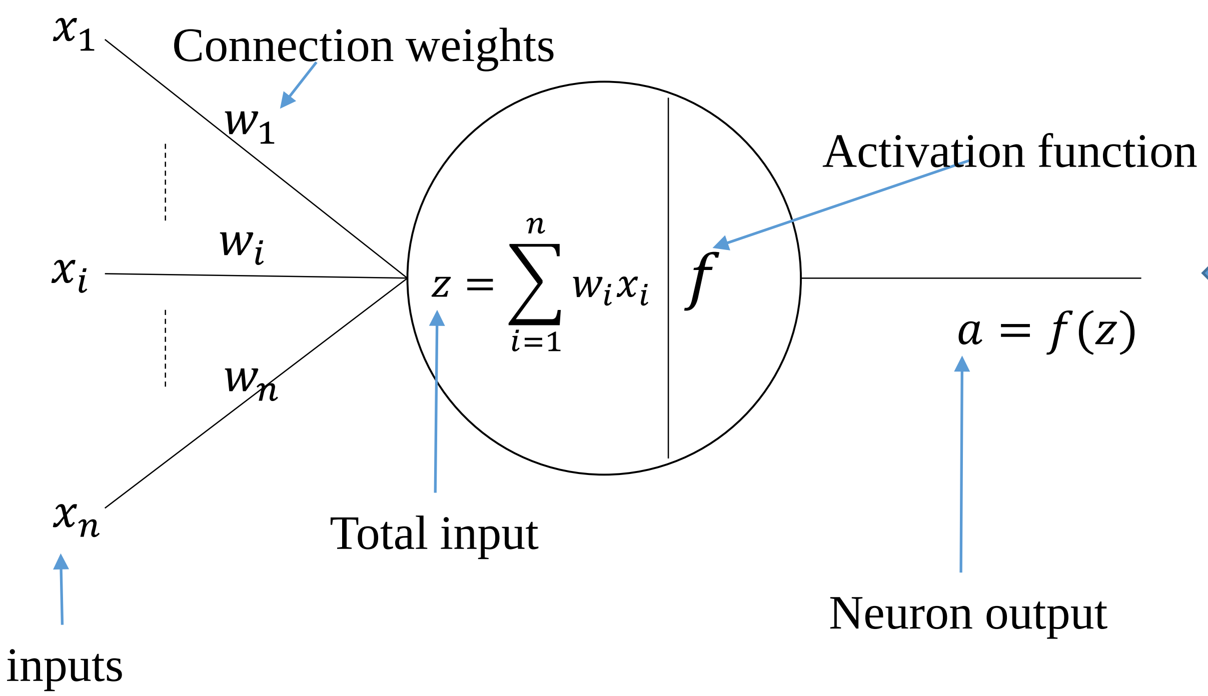


Computational Model of Neurons









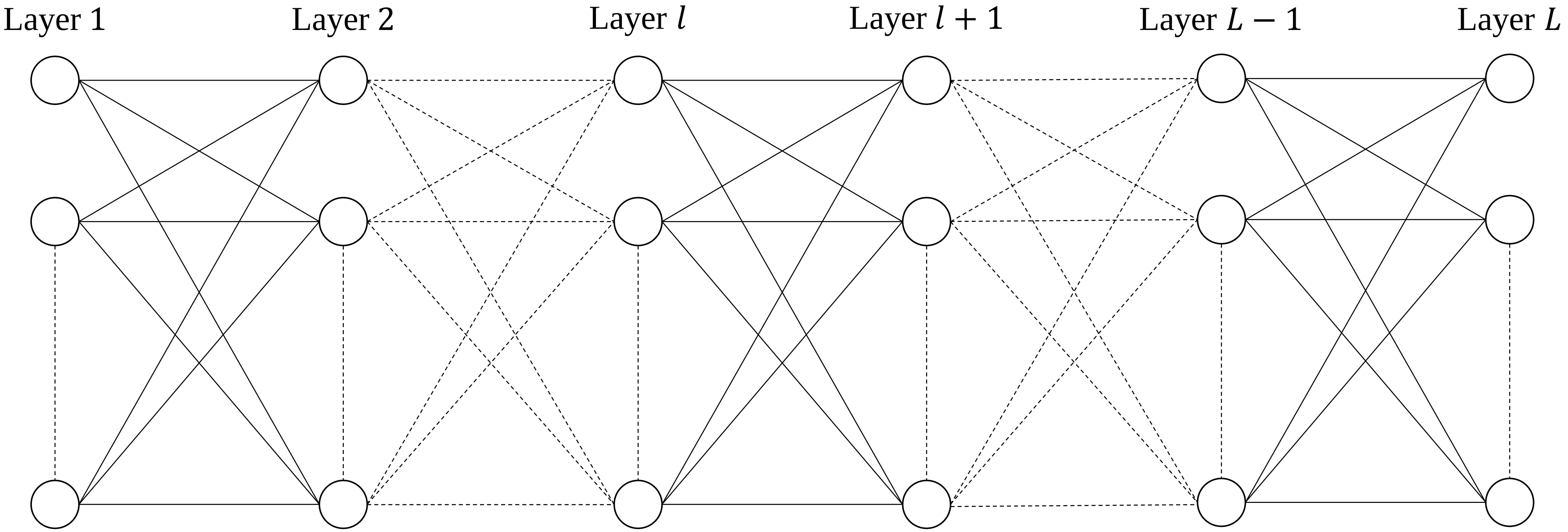
$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

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

<u>Basic Model 2</u>

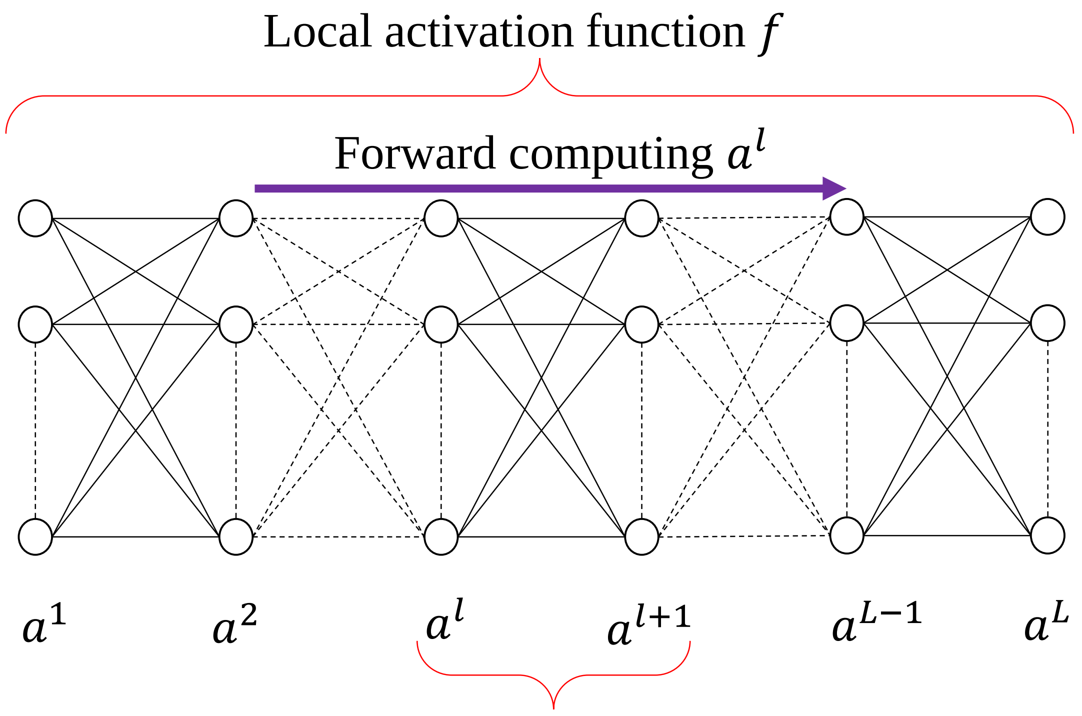
Computational Model of Neural Networks

Basic Model 3

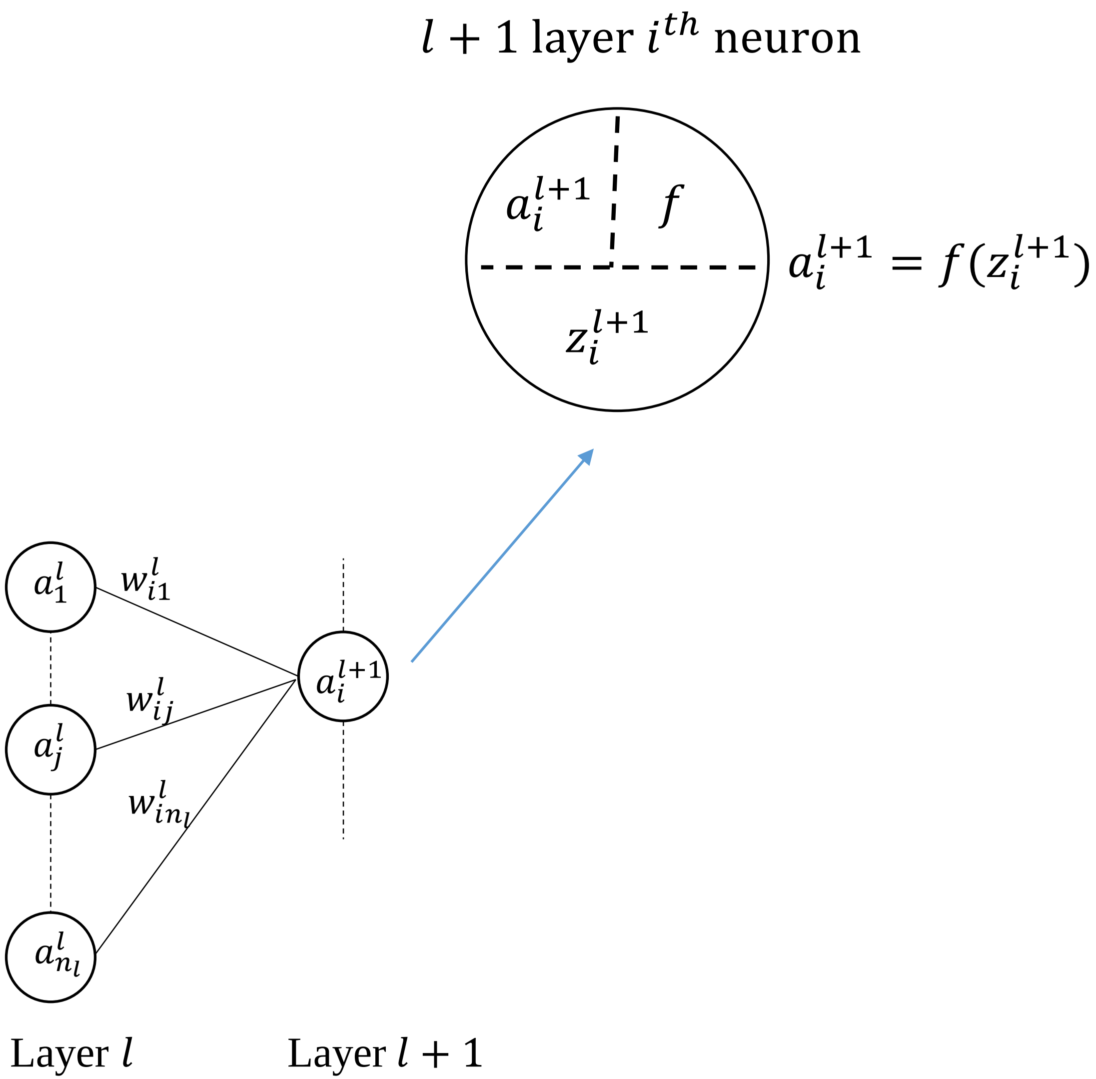


neurons	n_1		n_2		n_l		n_{l+1}		n_{L-1}		n_L
weights		W^1				W^l				W^{L-1}	
predictions	a^1		a^2		a^l		a^{l+1}		a^{L-1}		a^L

Forward Computing

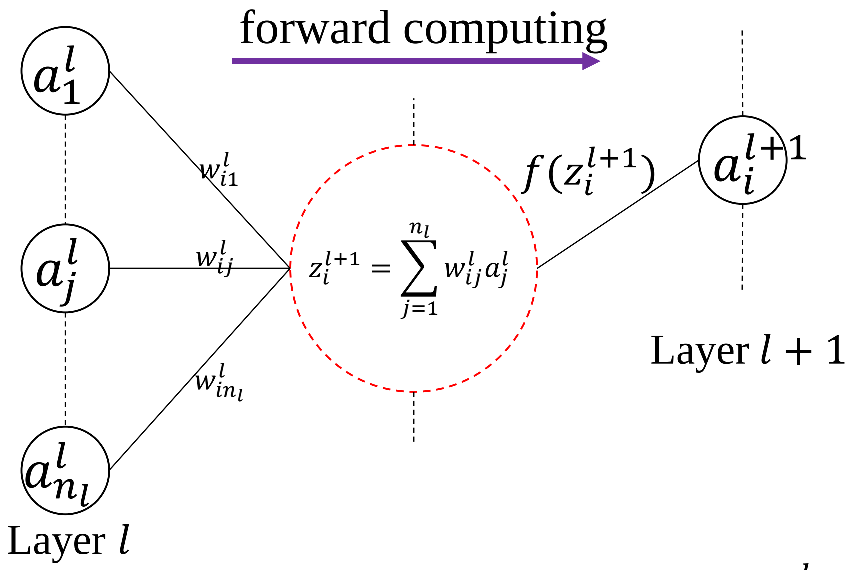


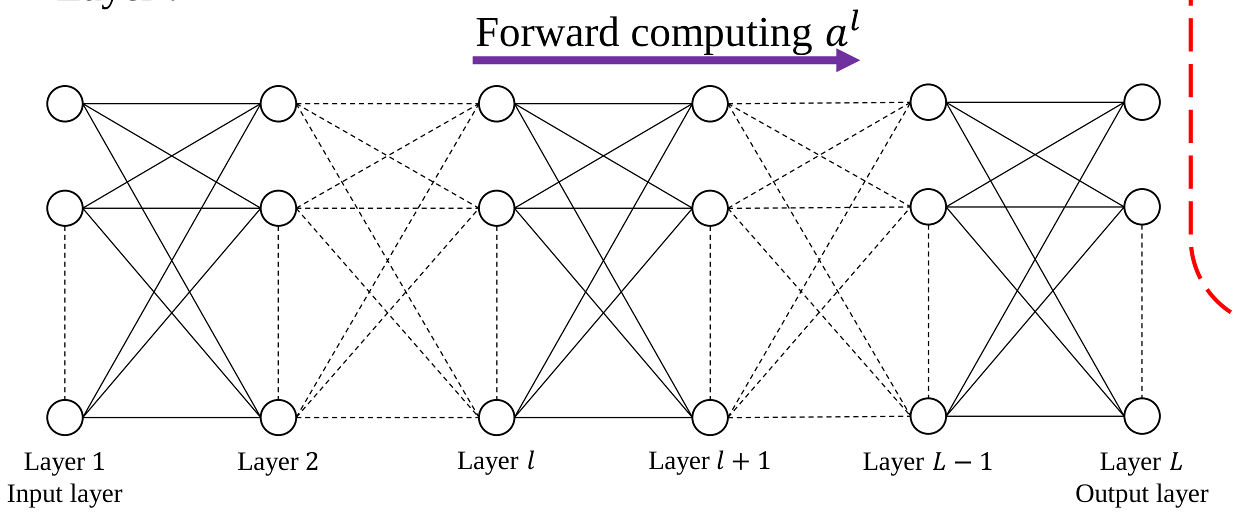
Computing unit



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

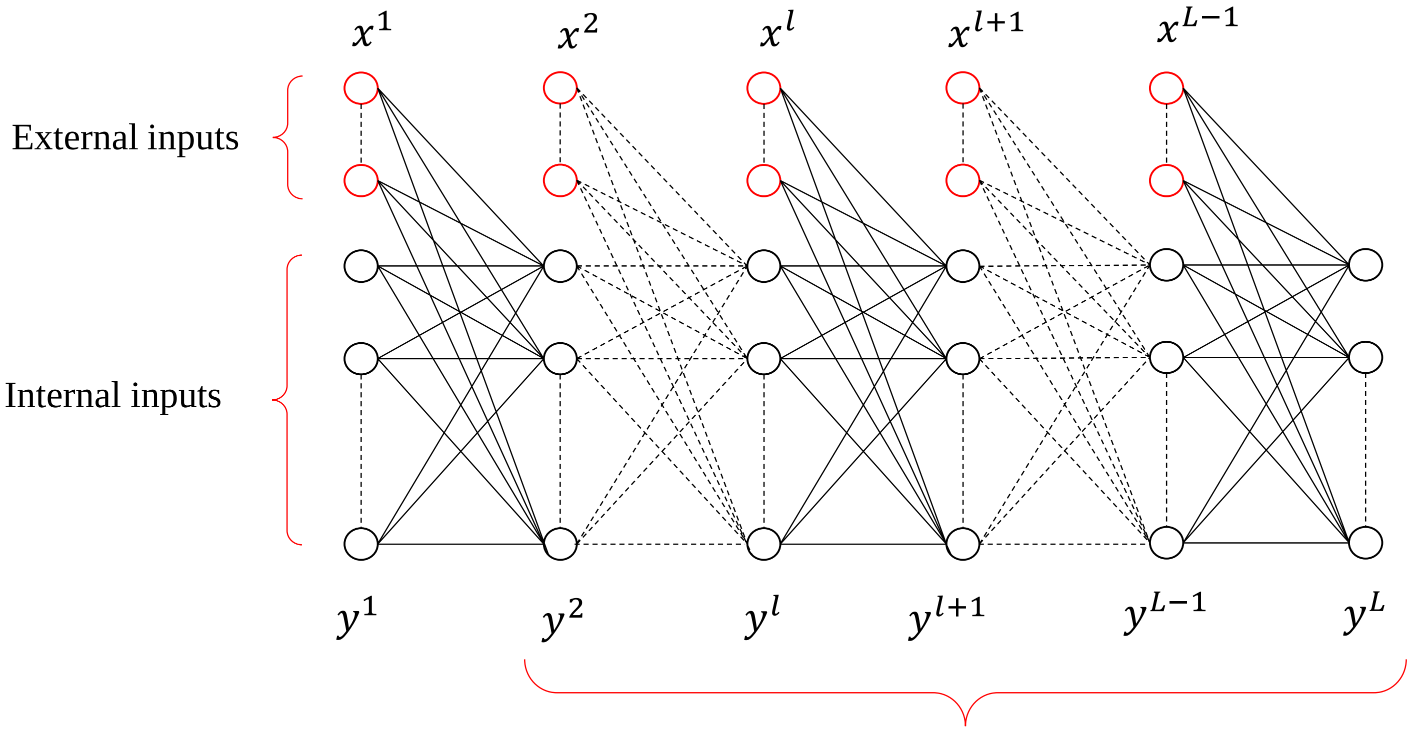
$$\begin{cases} a^{l+1} = f\left(z^{l+1}\right) \\ z^{l+1} = w^l a^l \end{cases}$$

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

External Inputs



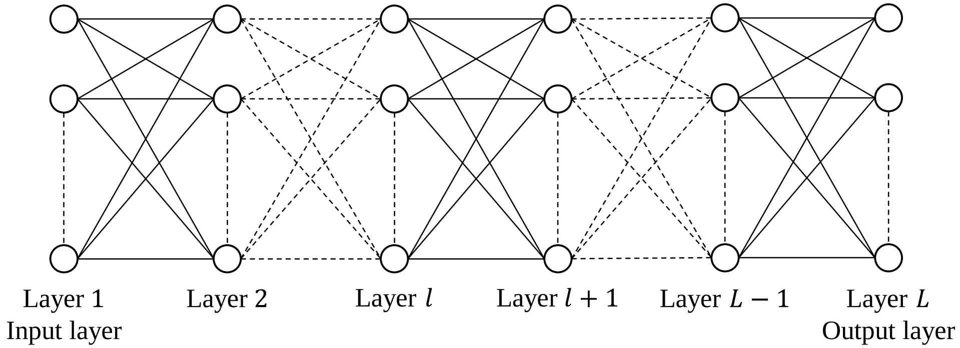
Internal representations / outputs / predictions

$$a^l = \begin{bmatrix} x^l \\ y^l \end{bmatrix}$$

$$z^{l+1} = W^l a^l$$

$$y^{l+1} = f\left(z^{l+1}\right)$$

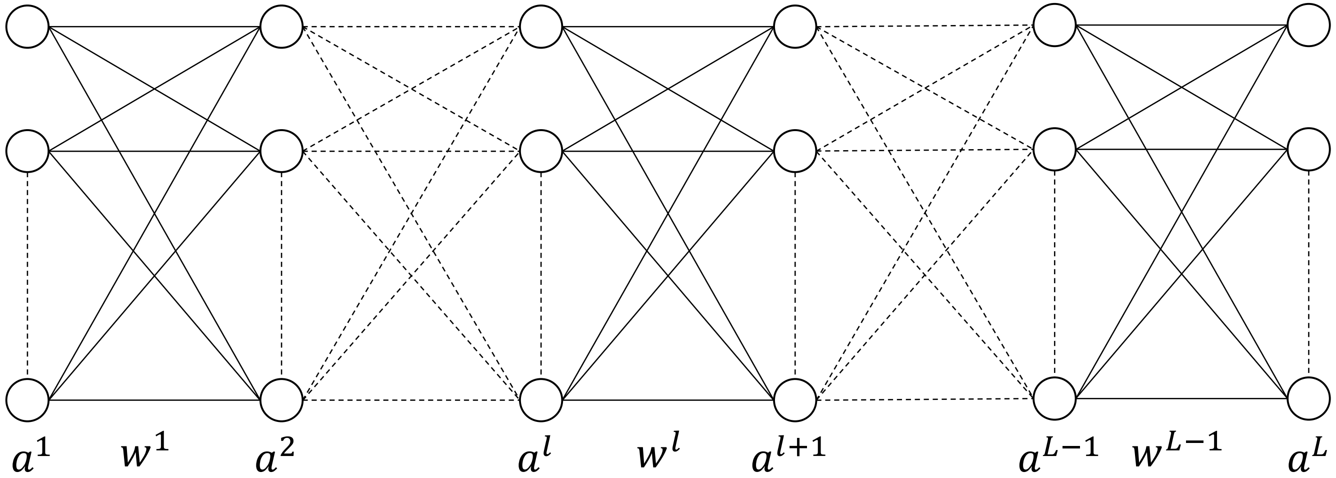
$$a^{l+1} = \begin{bmatrix} x^{l+1} \\ y^{l+1} \end{bmatrix}$$



External inputs:

If neurons in l layer are not connected to any neurons in previous layer, these neurons are called external inputs of $l+1$ layer. External inputs can exist in any layer except the last one.

Nonlinear Mapping / Dynamical Systems



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

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

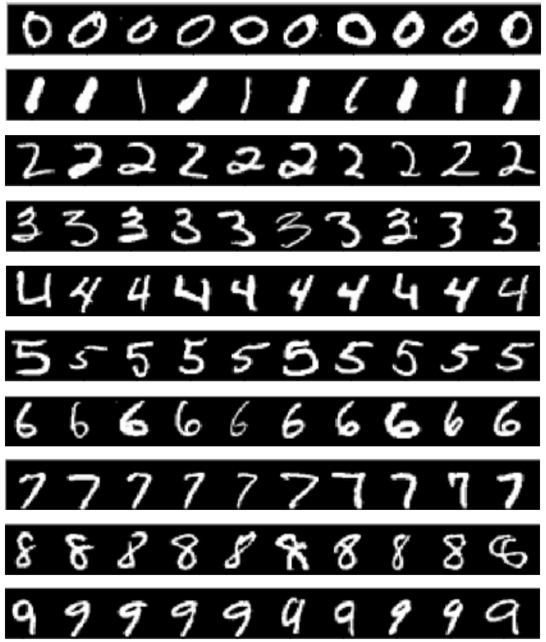
$$R^{n_1} \longrightarrow R^{n_L}$$

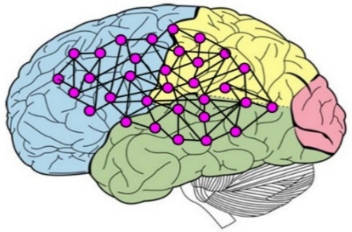
Nonlinear mapping

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

Discrete time dynamical system

Handwritten digits recognition





0
1
2
3
4
5
6
7
8
9

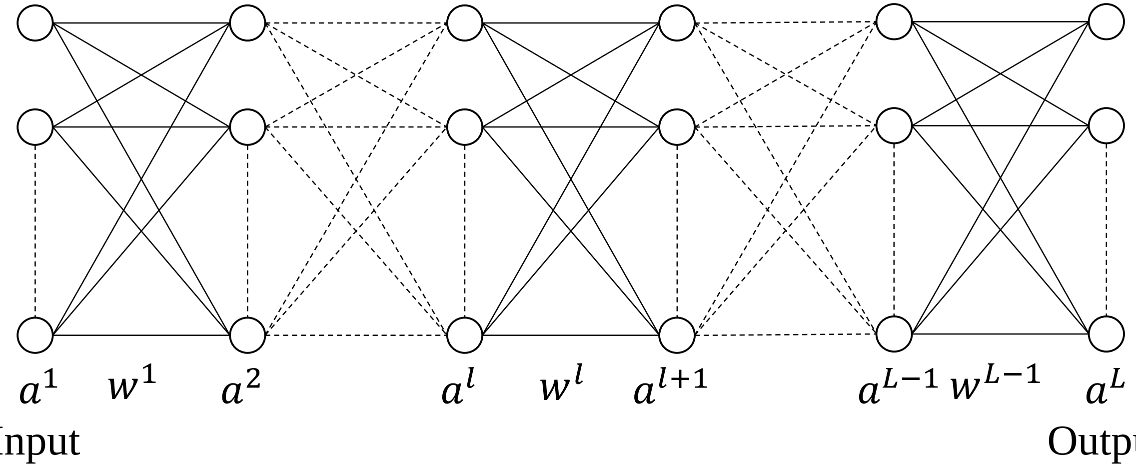
The human brain is so powerful so that any child can recognize the handwritten digits easily.

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?

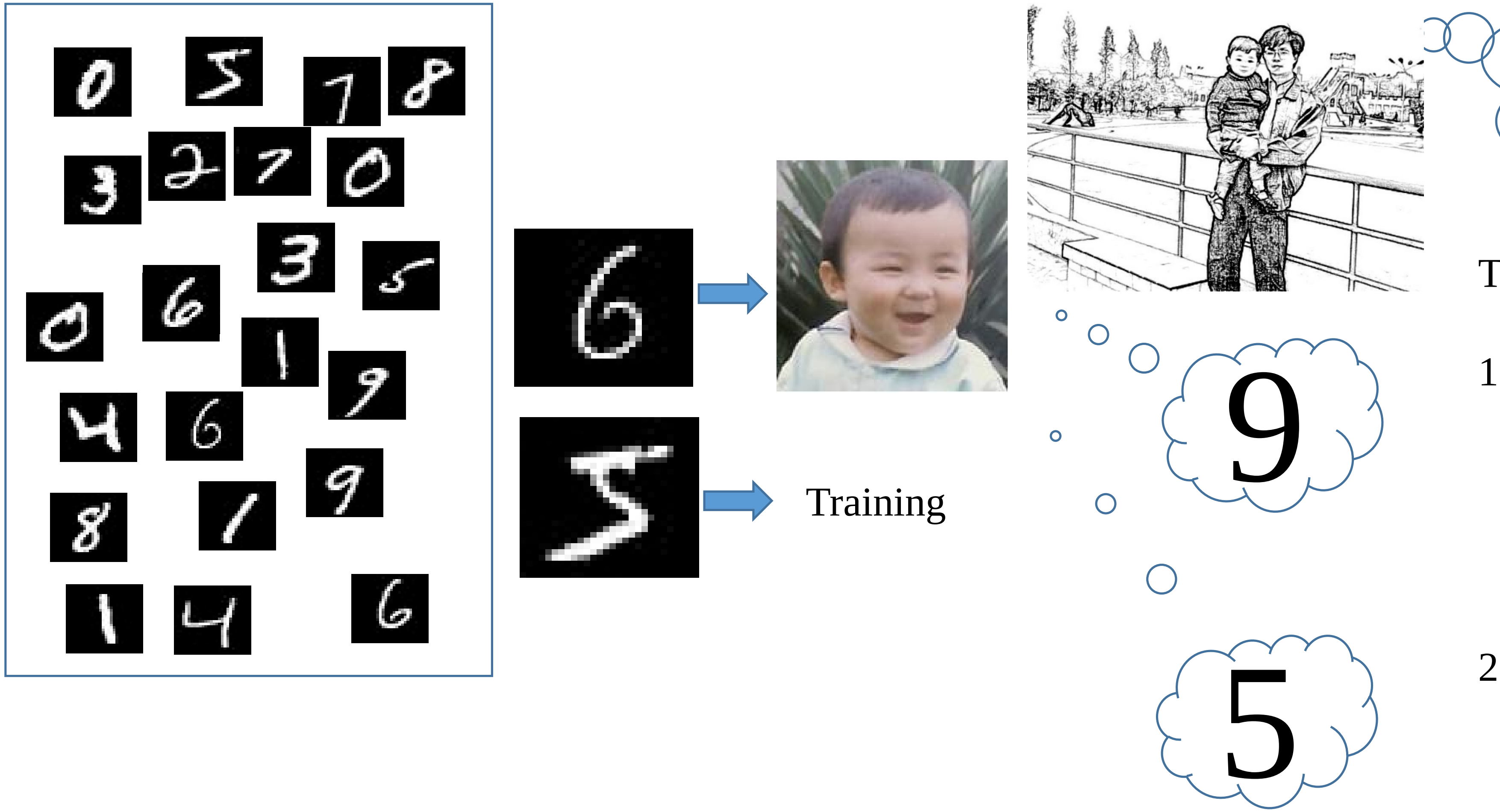
Artificial neural network



Outline

- 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

Network Performance: Cost Function



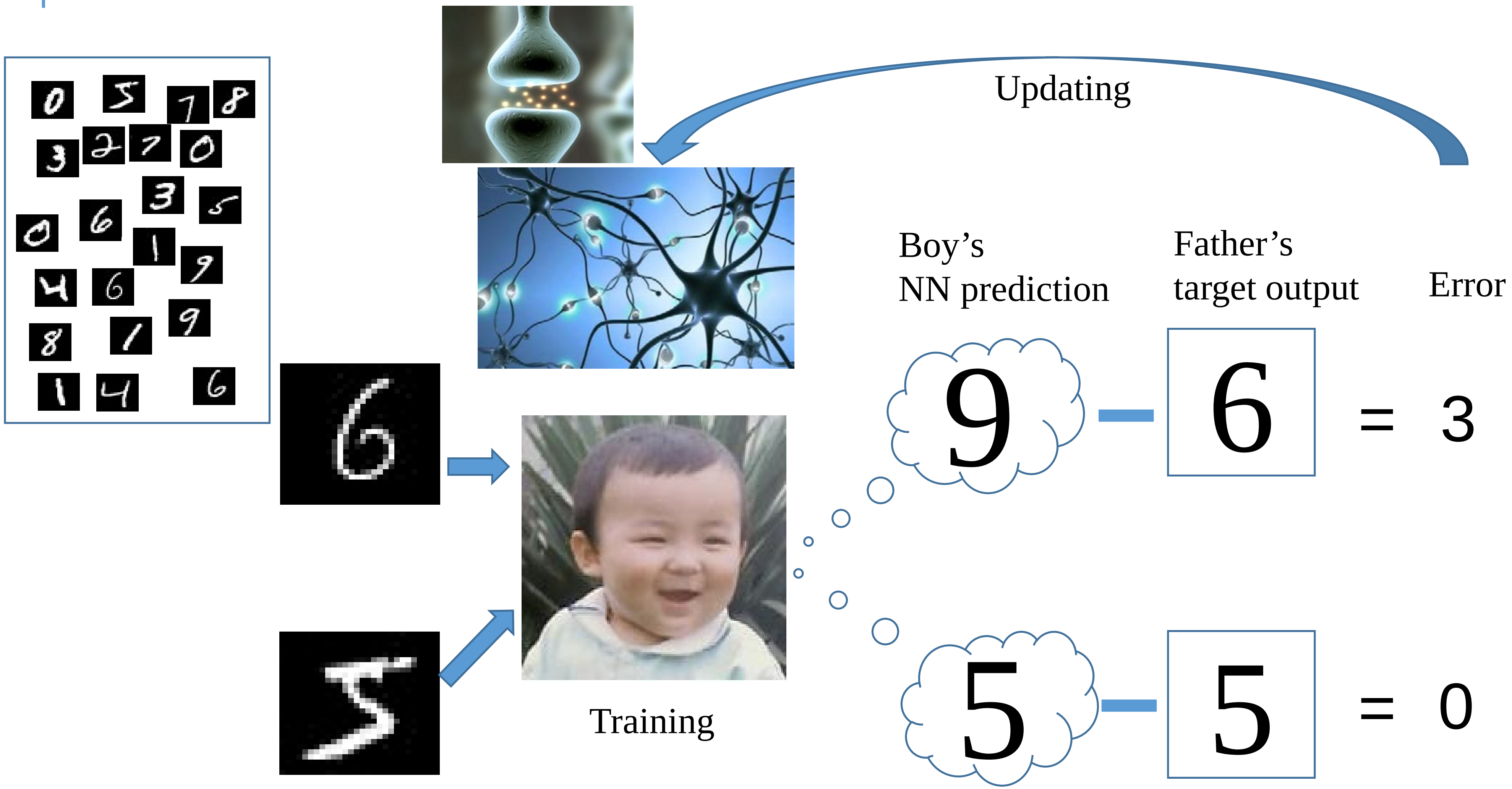
Good Performance!
The father knows the correct answer.
Supervised Learning

Two important factors:

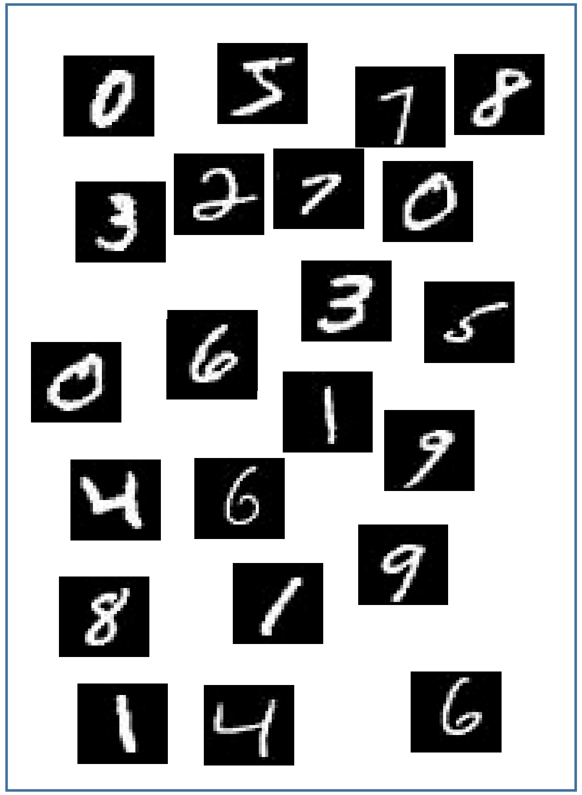
1. There must be a measure to measure the correctness between correct answer and the boy’s real output. ----- Performance function.

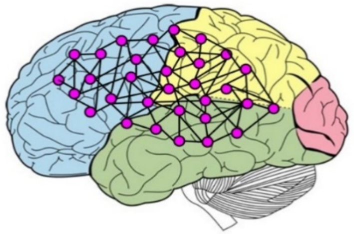
2. There must be a mechanism to change the knowledge system of the boy. ---- Learning algorithm.

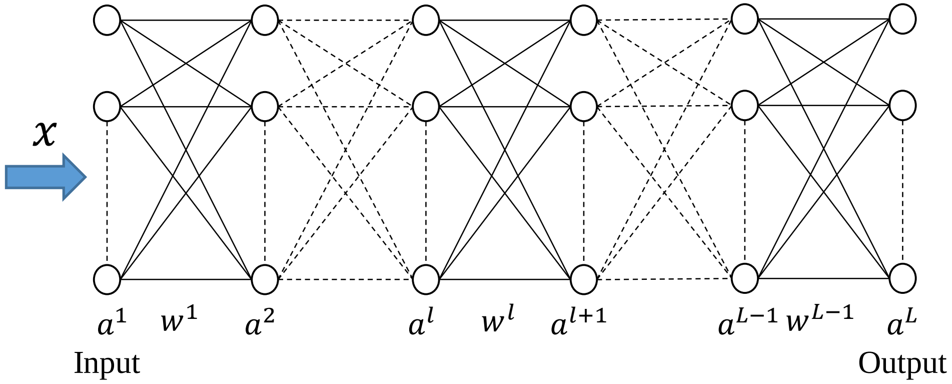
Network Performance: Cost Function



Network Performance: Cost Function







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.,

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

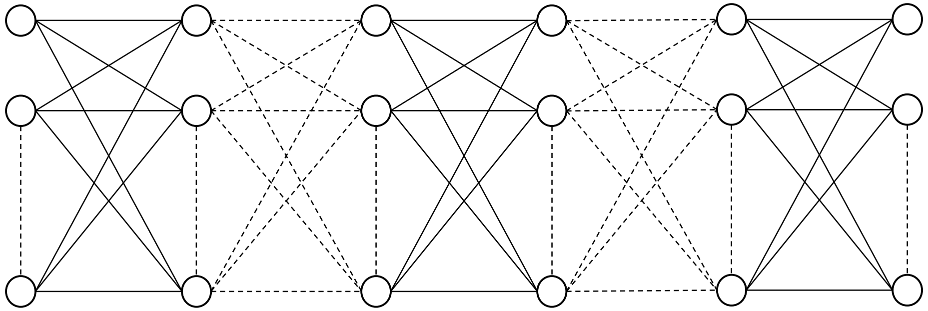
Supervised Learning

Training Data

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

A training sample (x, y^L)

Input x



Network prediction

Target

$$\underbrace{a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix} \qquad y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}}_{\text{Cost function}}$$

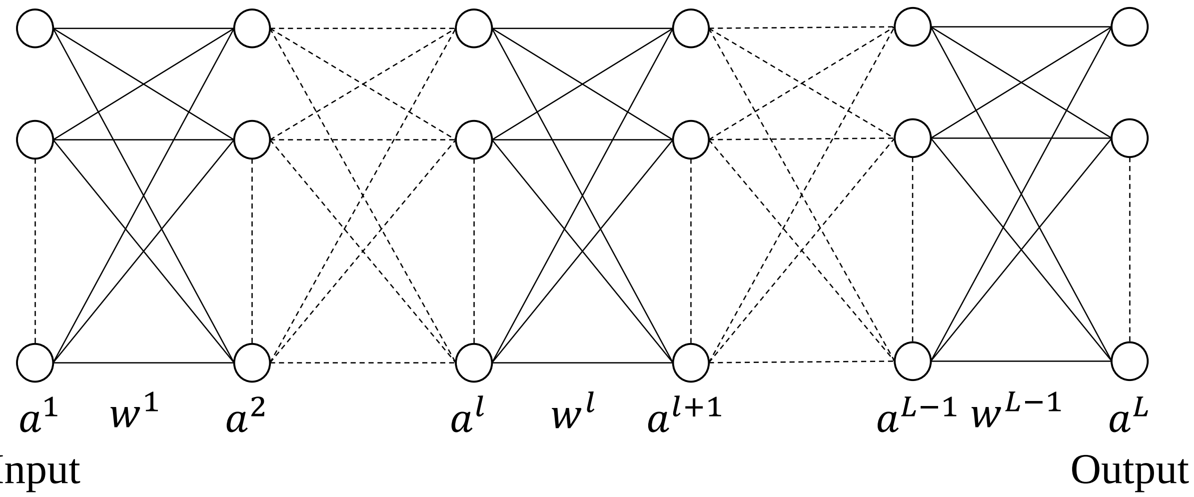
$$J(a^L, y^L) = J(w^1, \cdots, w^L)$$

updating the weights: Learning algorithm

Problem: How to construct a cost function?

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

Network Performance: Cost Function



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?

Target

Network prediction

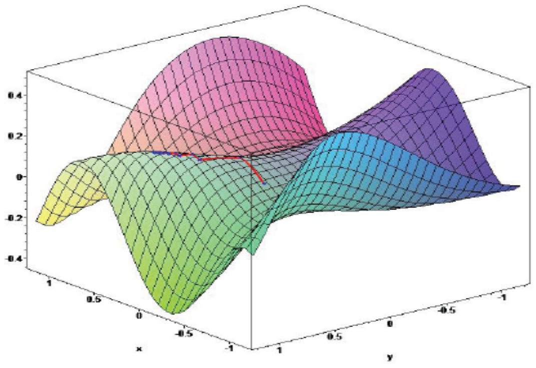
$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix} \qquad a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

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 $w^1, \cdots, w^L$.



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 $w^l (l = 1, \cdots, L)$ such that $w^l (l = 1, \cdots, L)$ is a minimum point of J.

Problem: How to find out the minimum points of J ?

Target

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

Network prediction

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

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 $w^1, \cdots, w^L$.

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

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

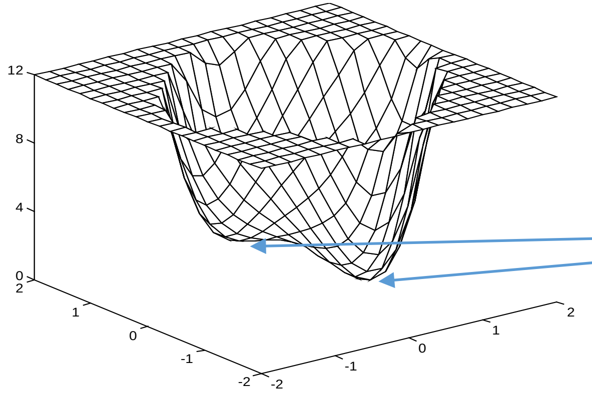
$$J(w), w \in R^n$$

w^* is a minimum point if

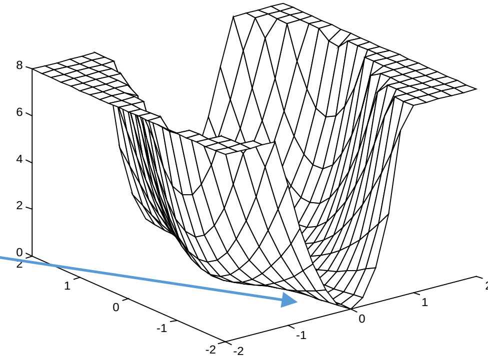
$$J(w^*) \leq J(w)$$

for any w that very close to w^*.

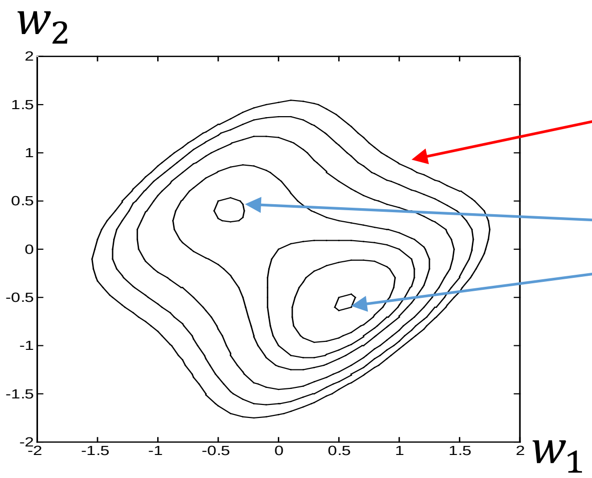
$$J(w_1, w_2) = (w_2 - w_1)^4 + 8w_1w_2 - w_1 + w_2 + 3$$



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

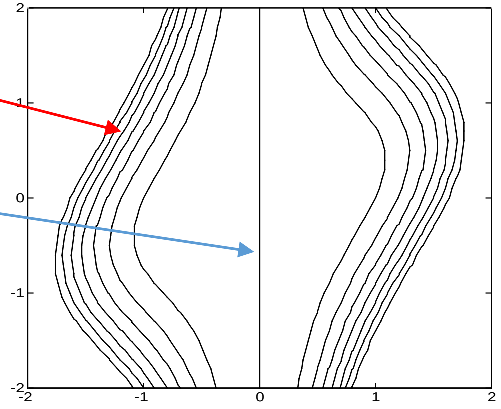


Minima



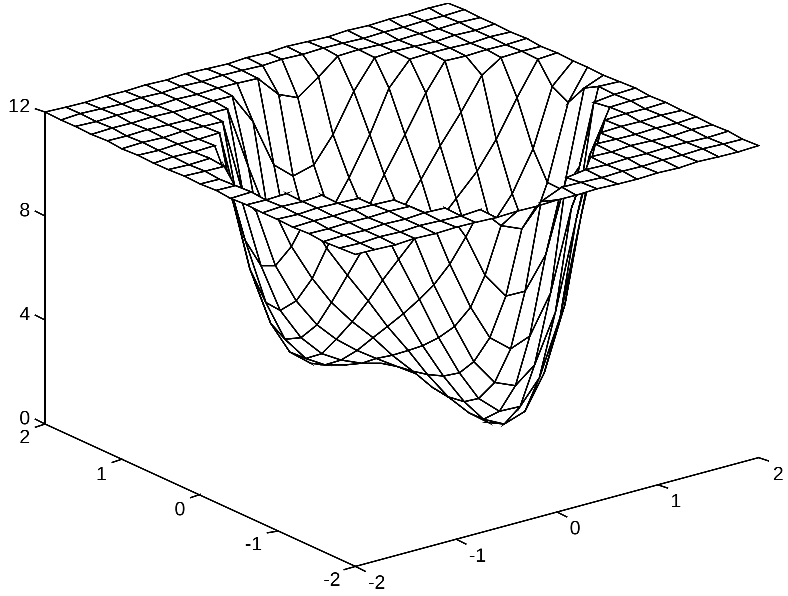
Contour

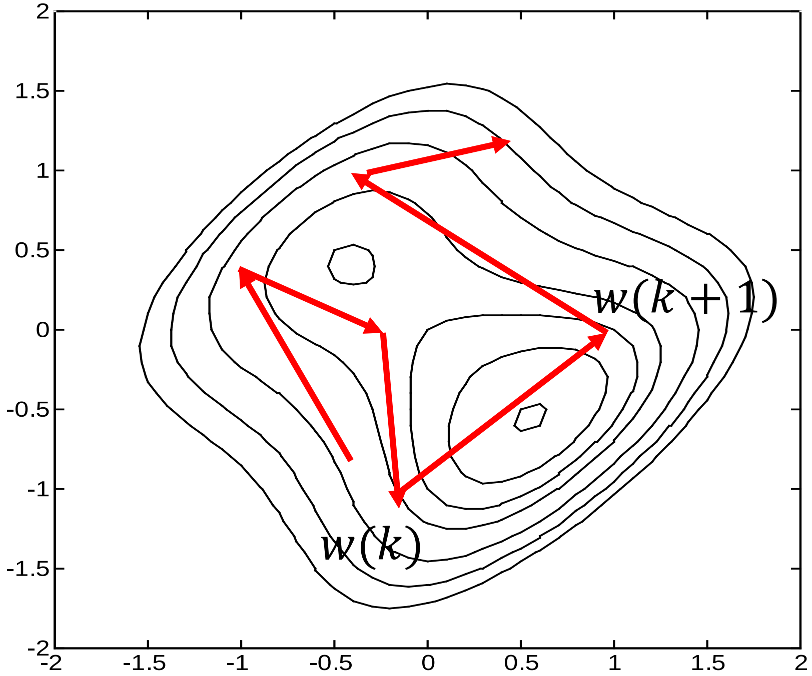
Minimum points



Problem:
How to find the minimum points?

Iteration Method

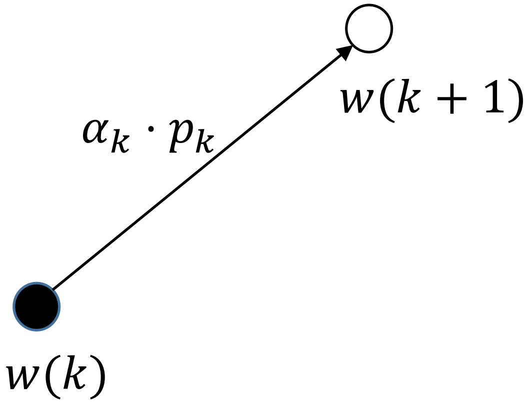




Finding a minimum point step by step

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

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

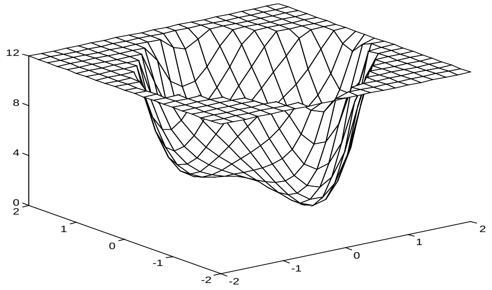


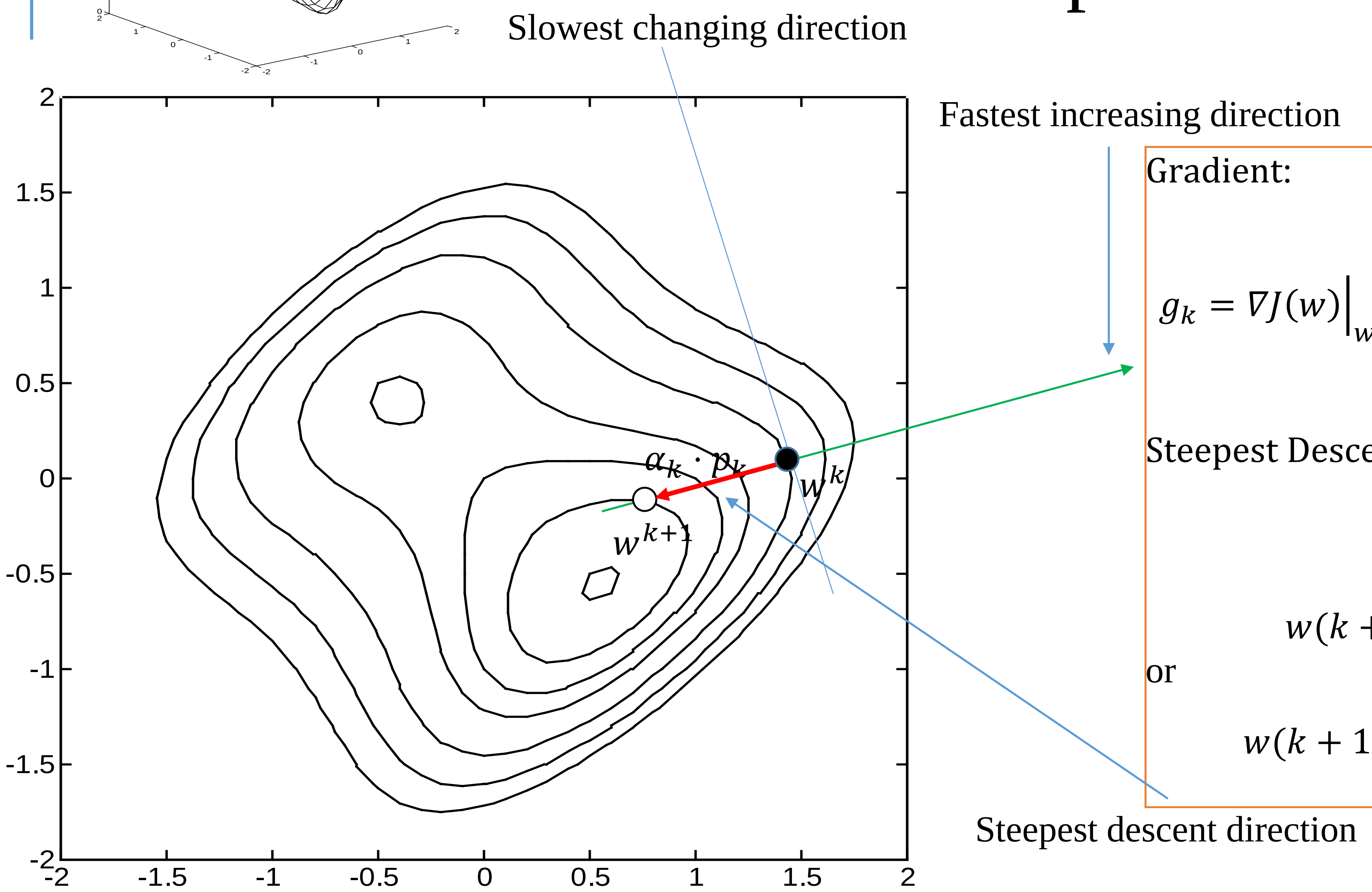
p_k is called searching direction

α_k is learning rate at step k.

Problem: How to get the searching direction p_k?

Steepest Descent Method





Gradient:

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

Steepest Descent Algorithm:

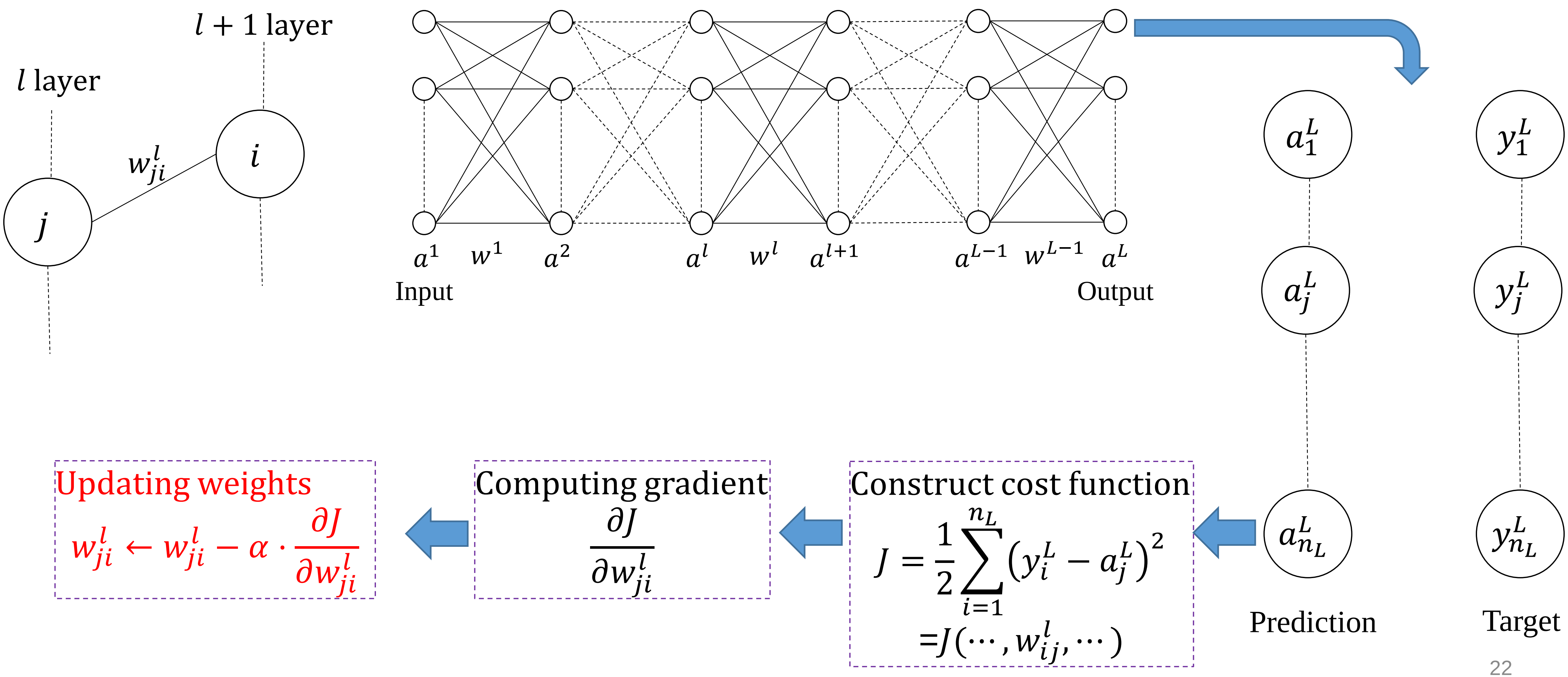
$$p_k = -g_k$$

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

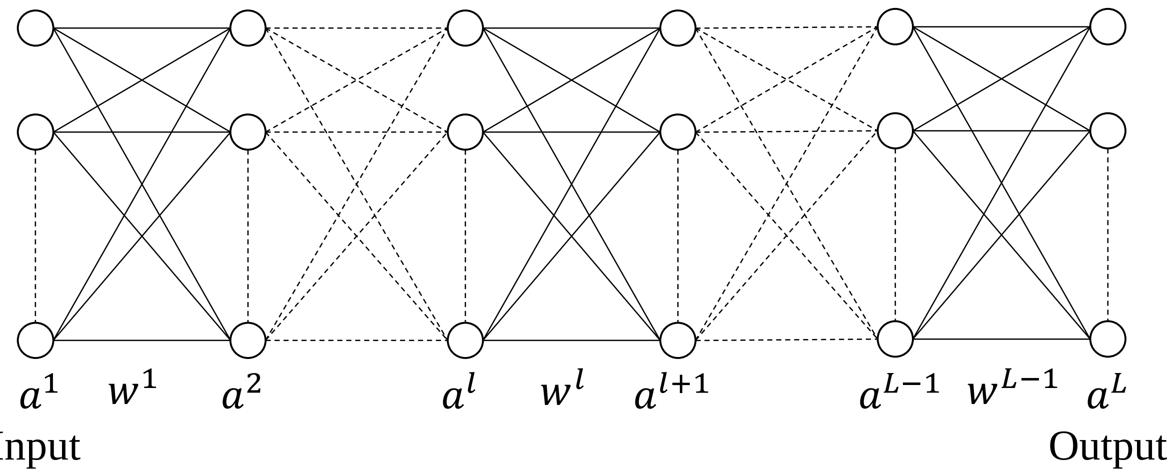
or

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

Steepest Descent Method



Steepest Descent Method

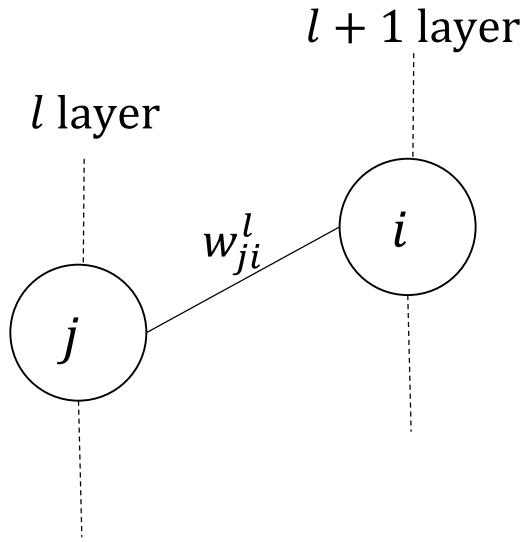


Target

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

prediction

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



Steepest Descent Method

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

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

1. Computing

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

2. Updating

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

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

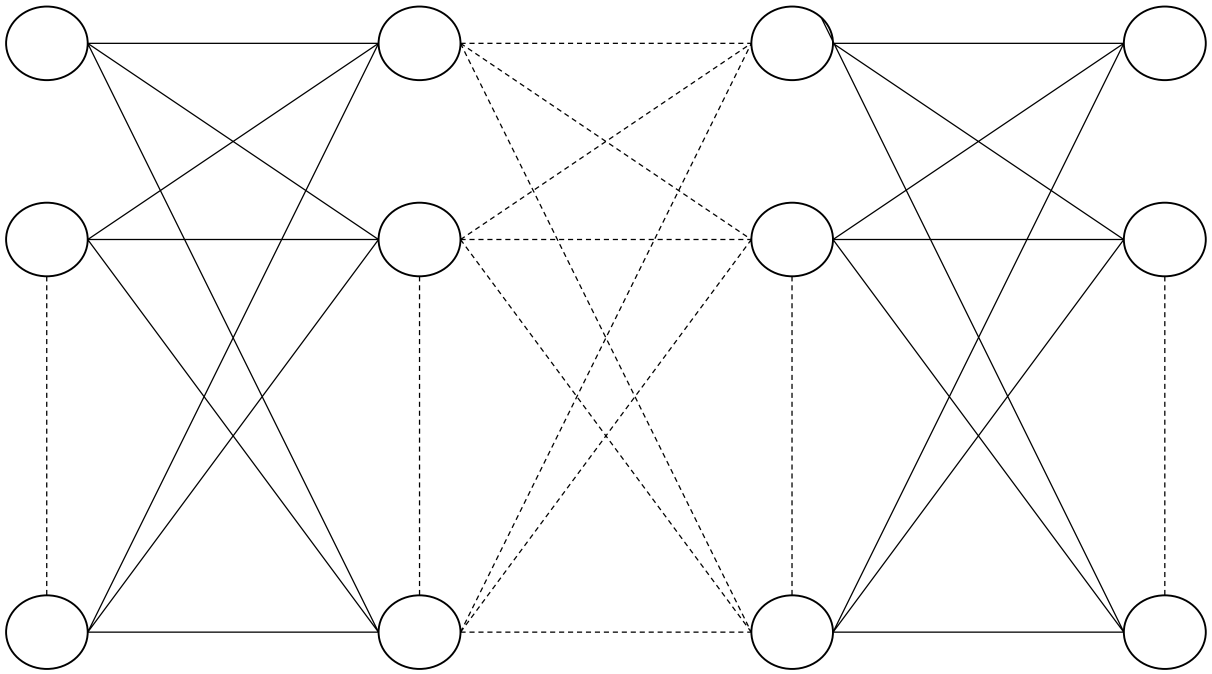
Answer:
Using the well-known BP method.

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

Forward computing a^l



Layer 1

Back propagation δ^l

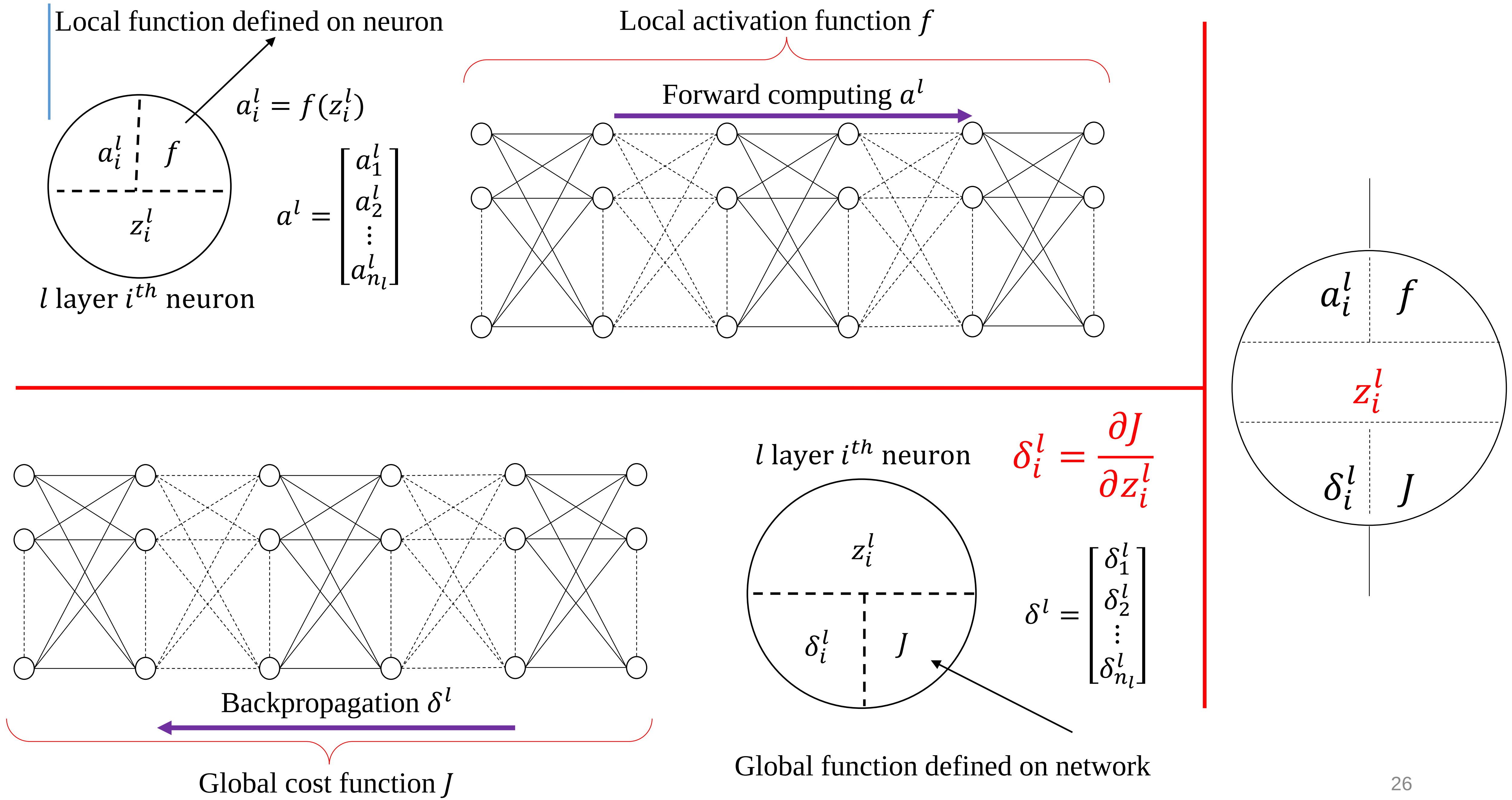
Layer L

Backpropagation is a efficient way to calculate

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

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

l layer

$$a_i^l = f(z_i^l)$$

define $\delta_i^l = \frac{\partial J}{\partial z_i^l}$

Problem 1:

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

$l+1$ layer

l layer

$$a_i^l = f(z_i^l)$$

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

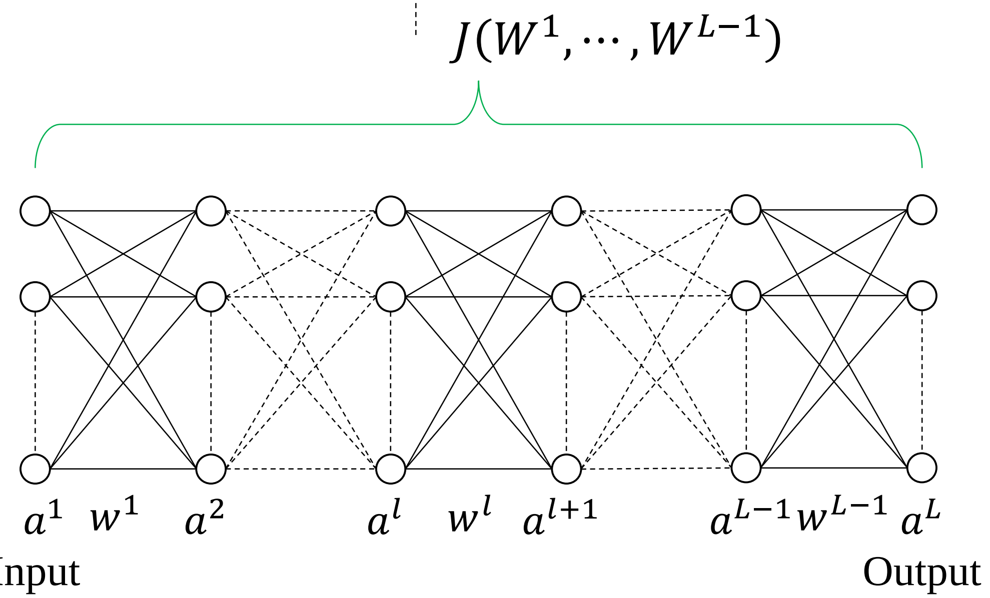
$$\frac{\partial J}{\partial w_{ji}^l}$$

$$w_{ji}^l$$

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

$$a_j^{l+1} = f(z_j^{l+1})$$

$$\delta_j^{l+1} = \frac{\partial J}{\partial z_j^{l+1}}$$

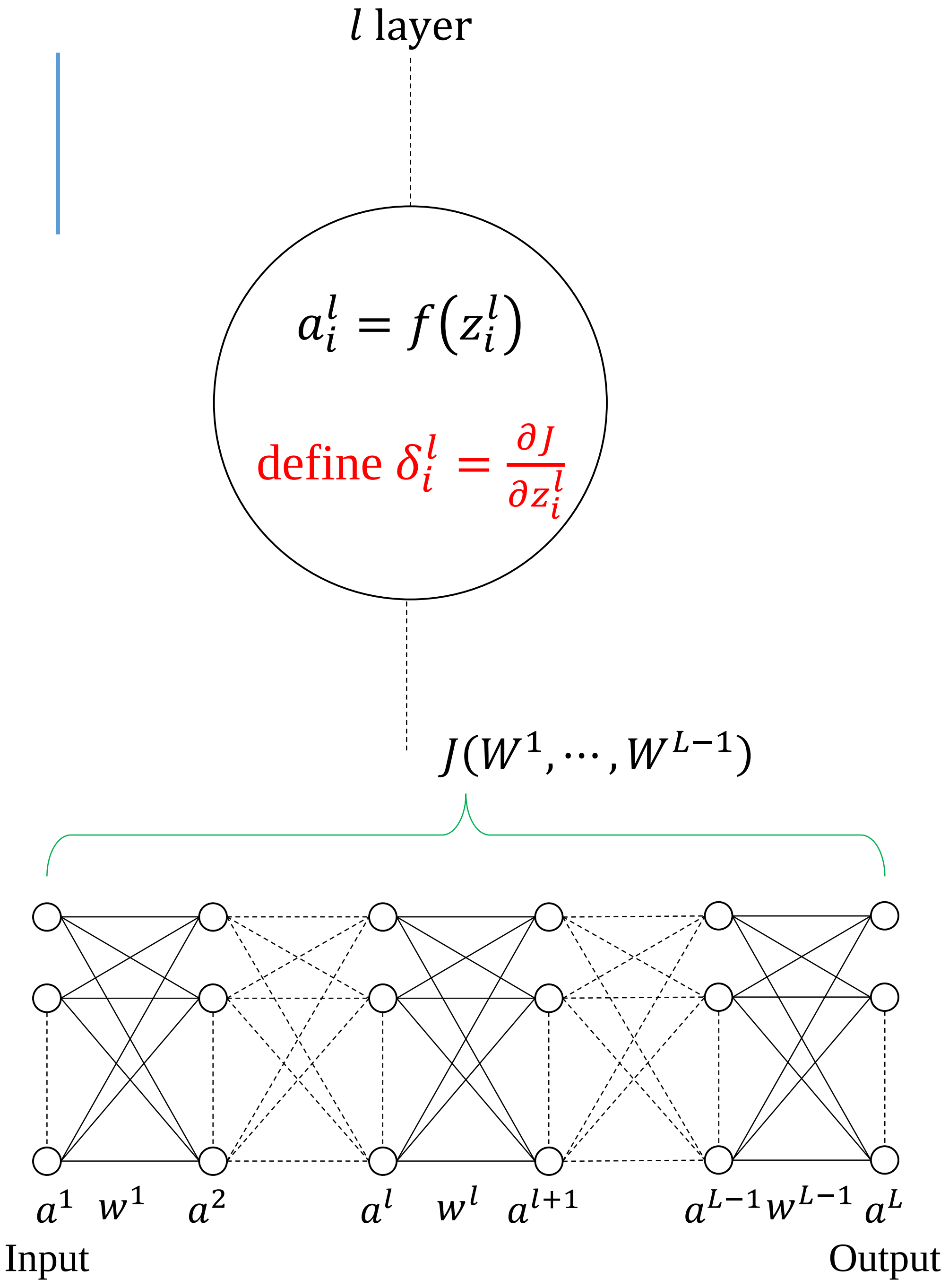


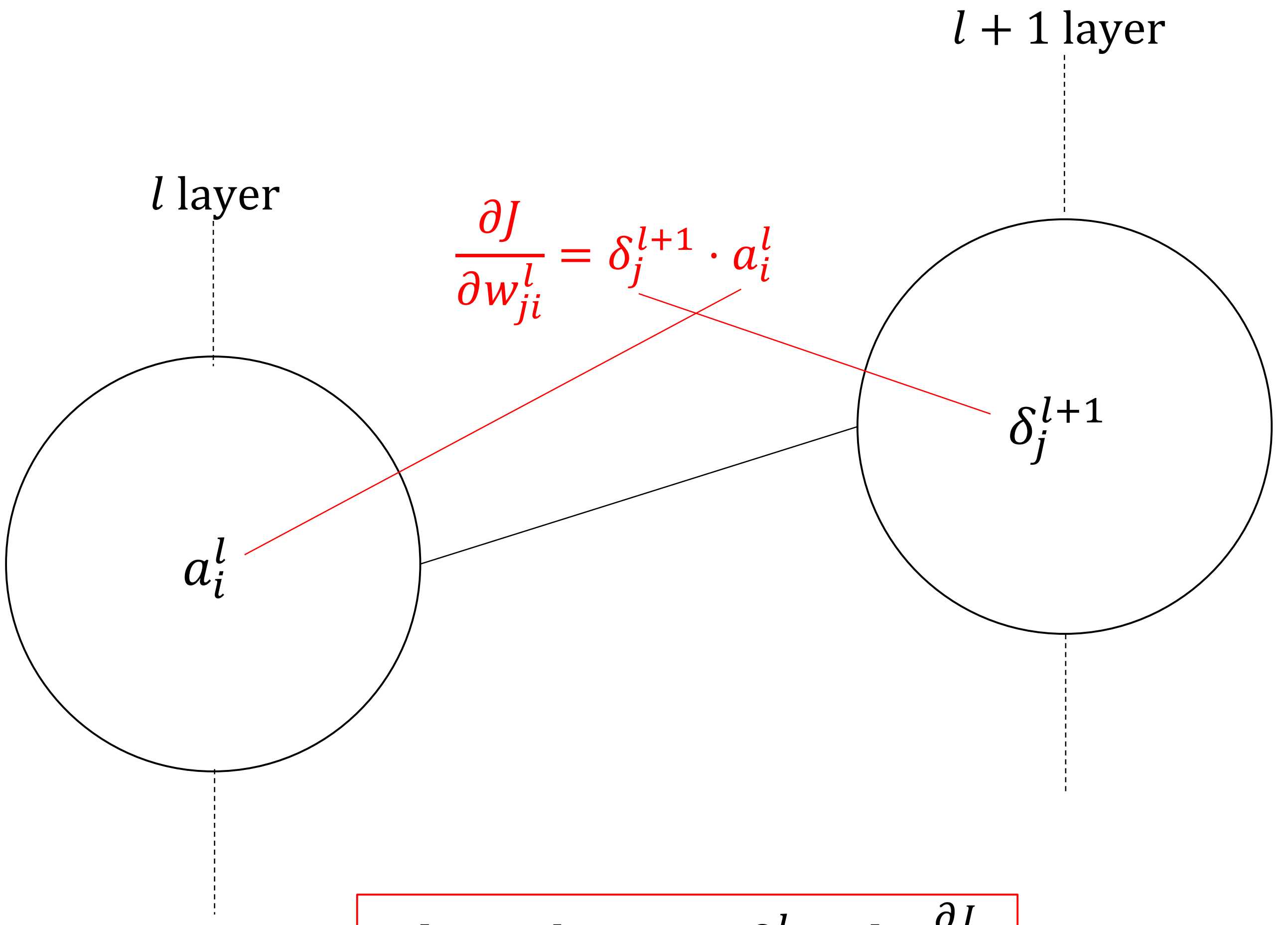
Relation between δ_i^l and $\frac{\partial J}{\partial w_{ji}^l}$

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

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 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_j^L ?

By definition

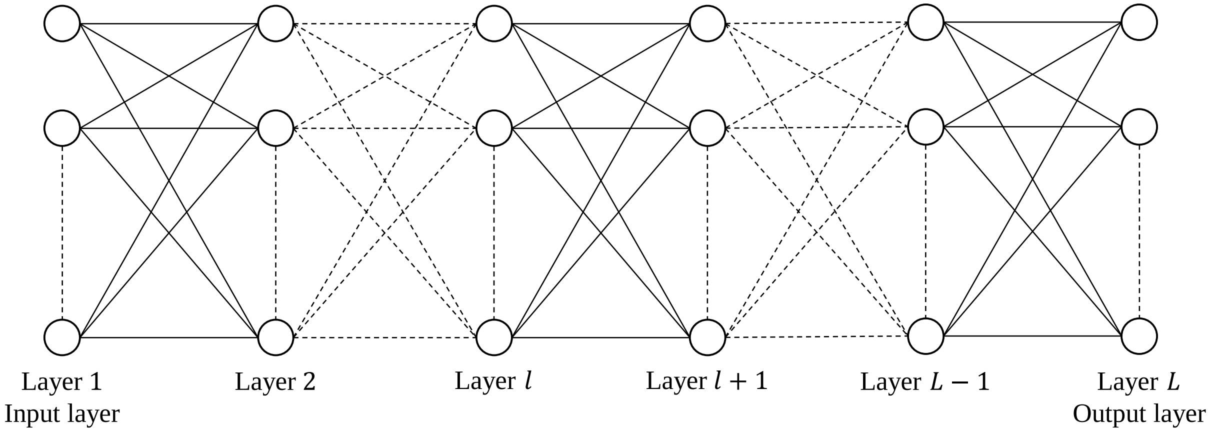
$$\delta_i^L = \frac{\partial J}{\partial z_i^L}$$

If

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

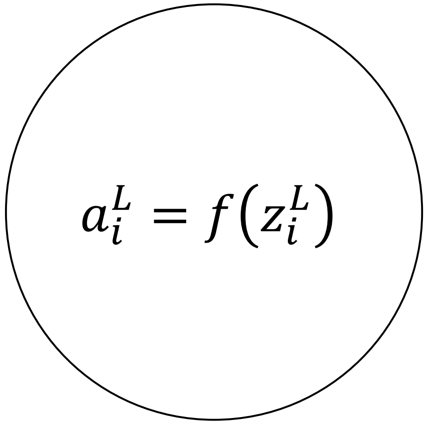
then,

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



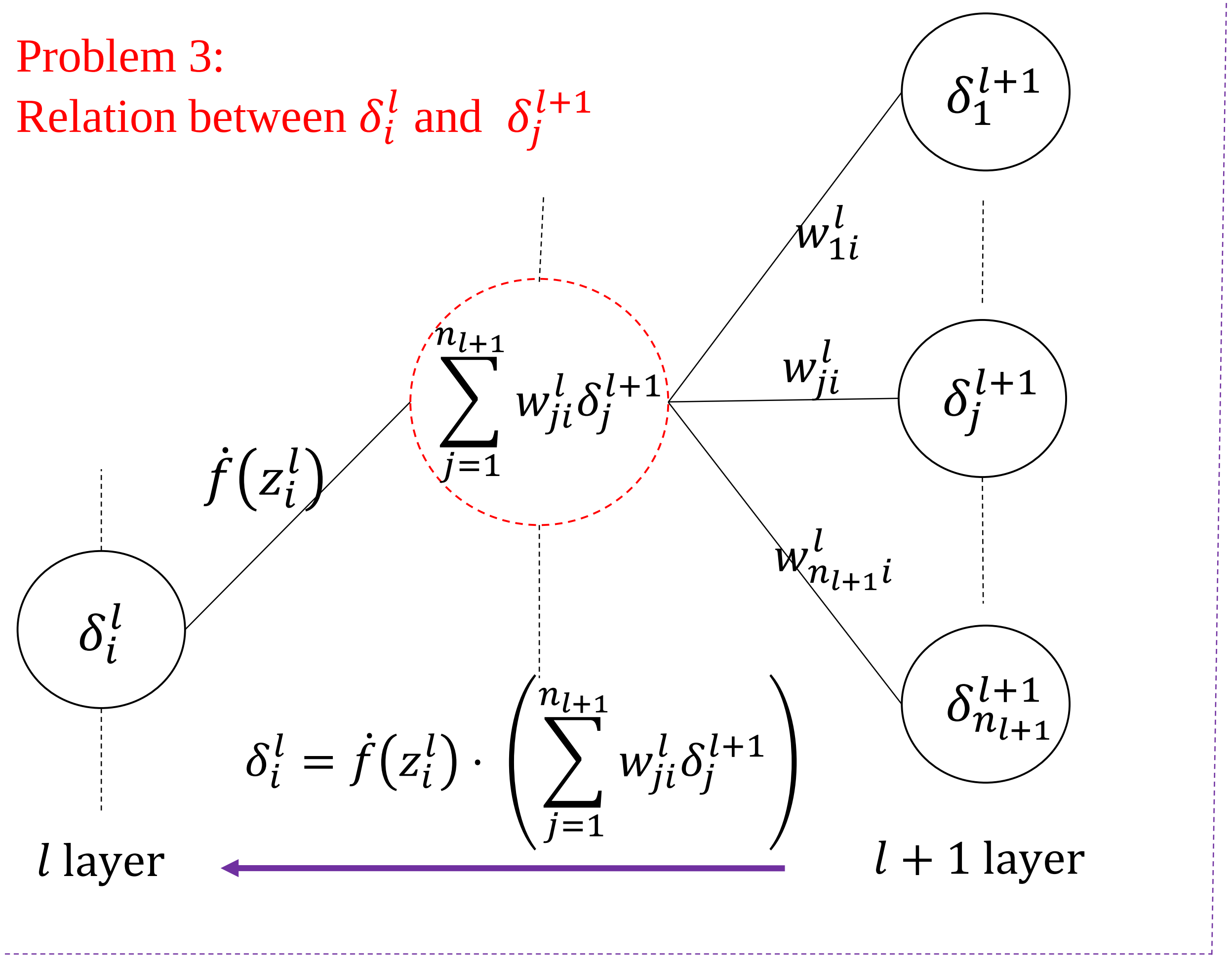
Backpropagation δ^l

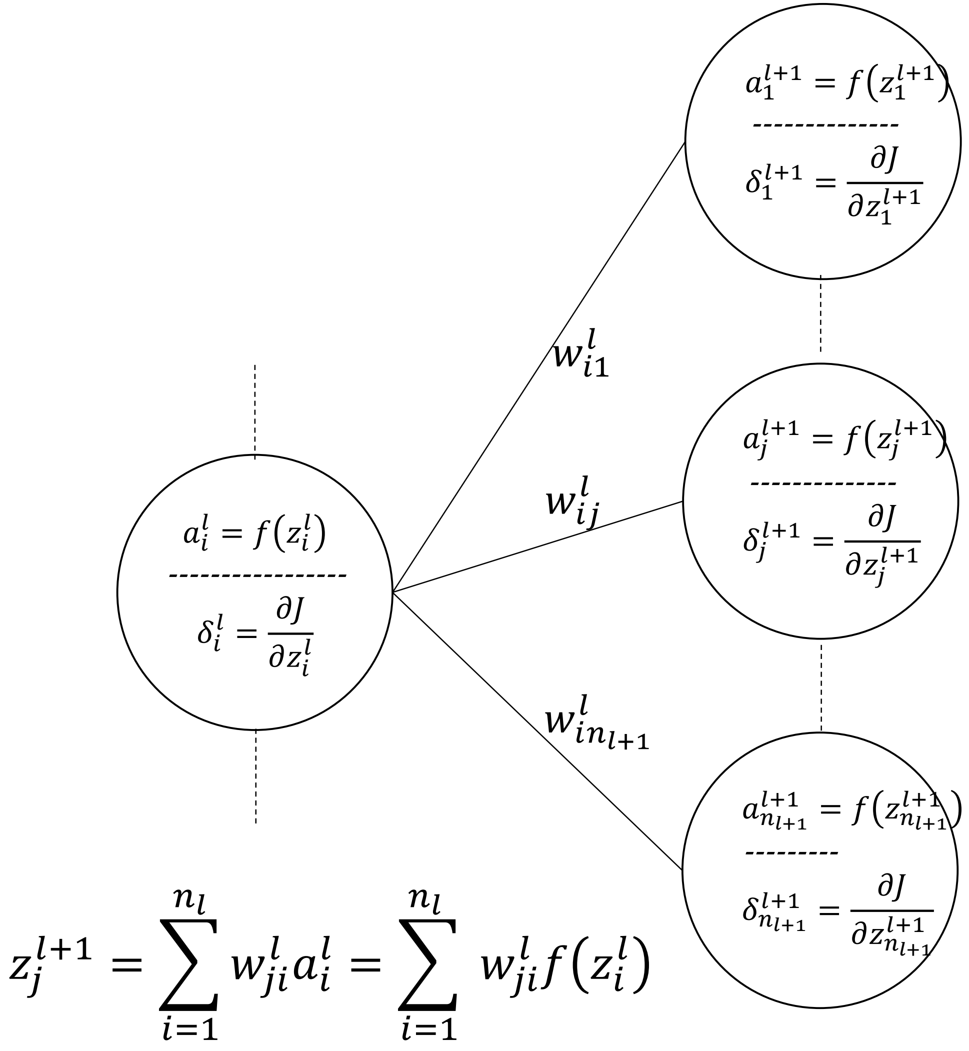
δ^{L}



L layer i^{th} neuron

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





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

Outline

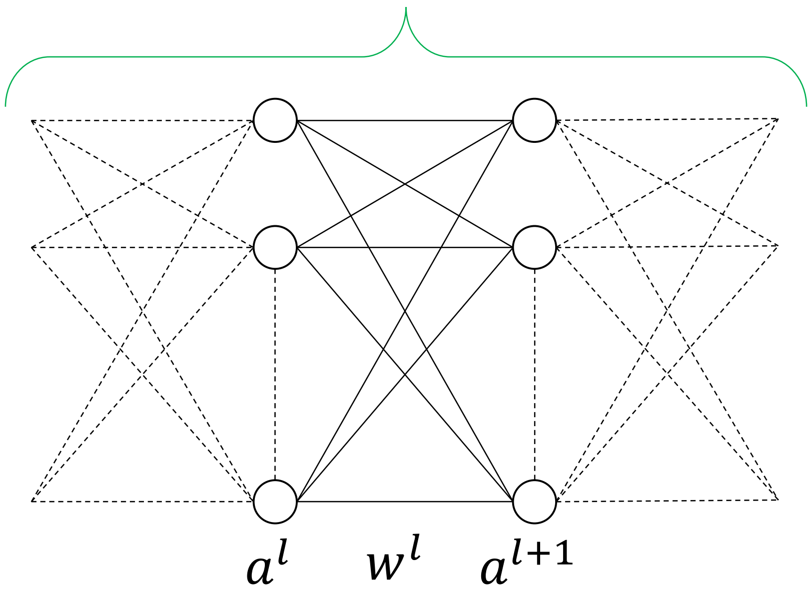
- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

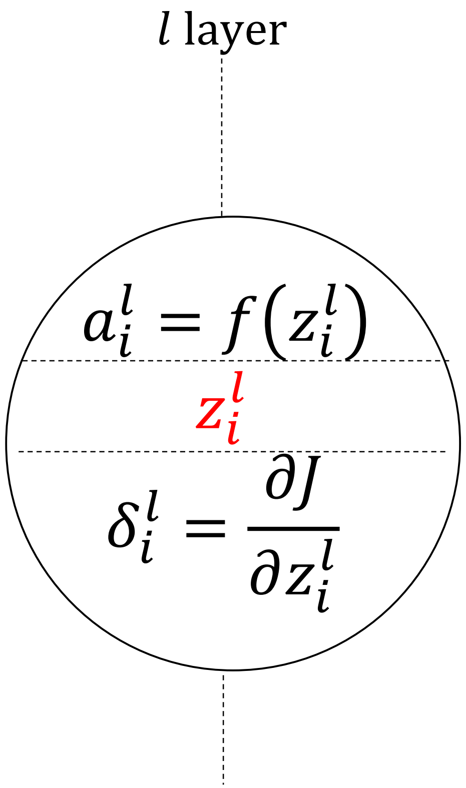
Three Pages to Understand BP: *The first page*

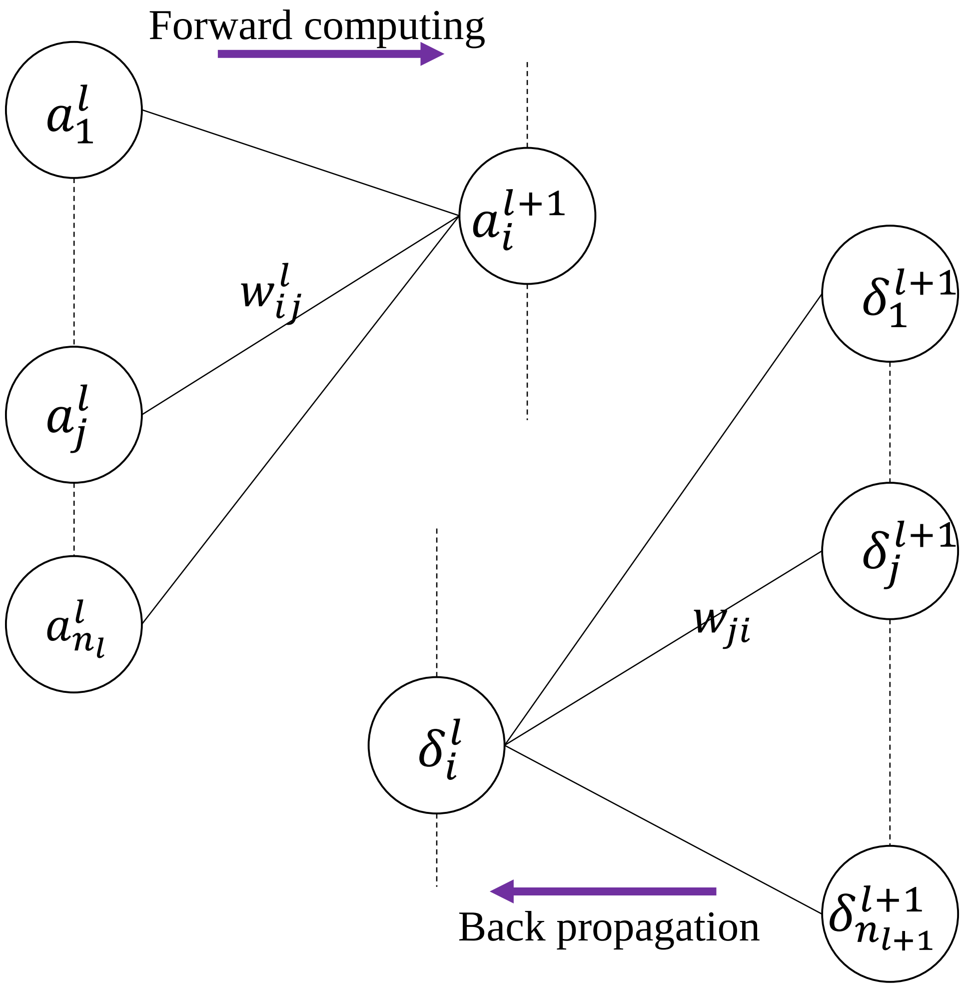
Cost function: $J(w^1, \cdots, w^L)$

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

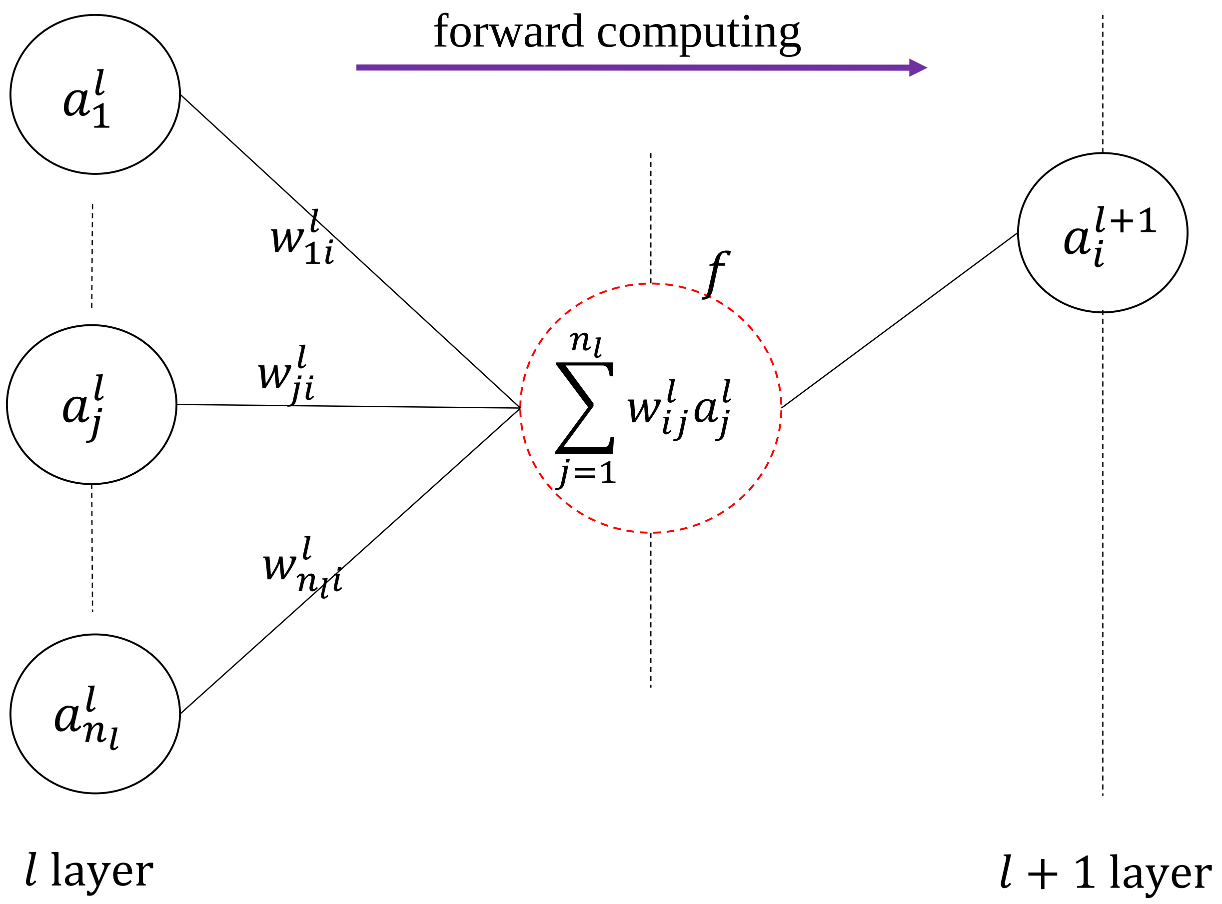
Relationship: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$







Three Pages to Understand BP: *The second page*



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

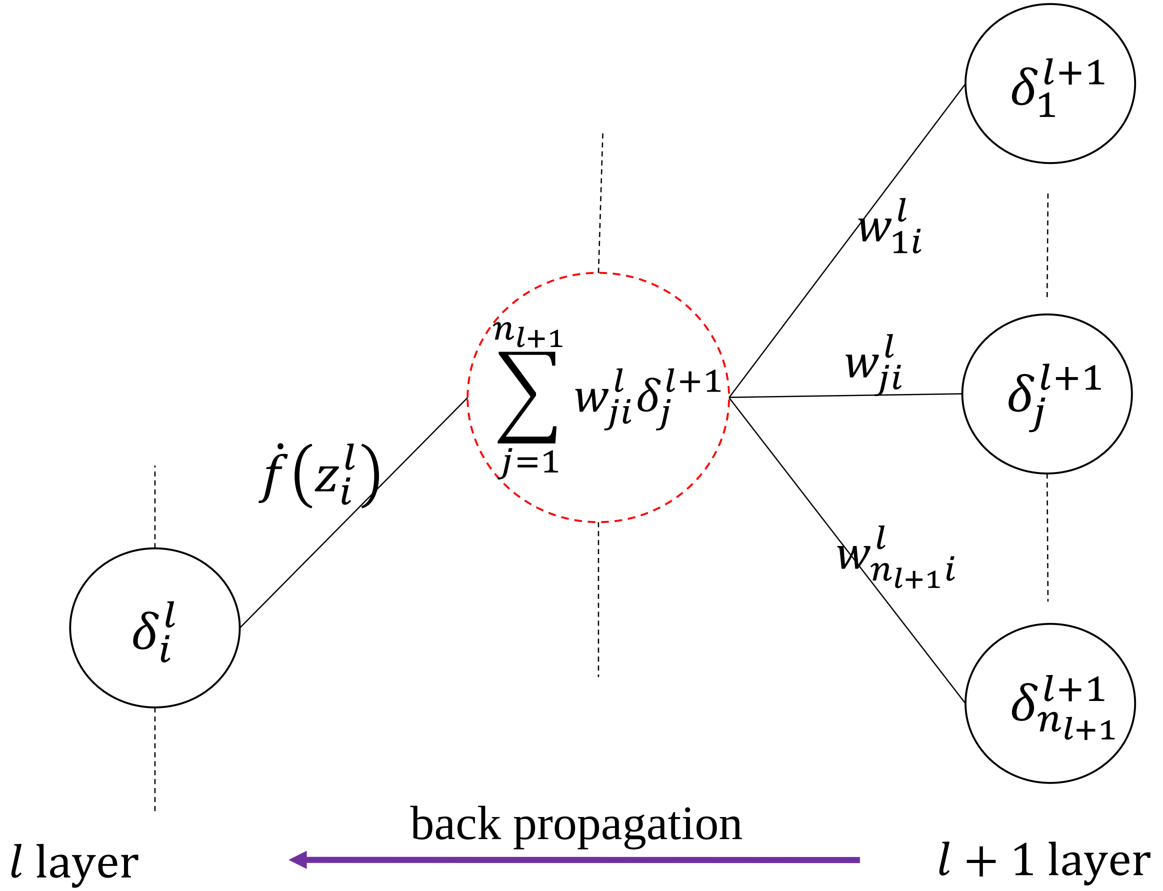
or --------------------

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

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

Three Pages to Understand BP: *The third page*

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



Outline

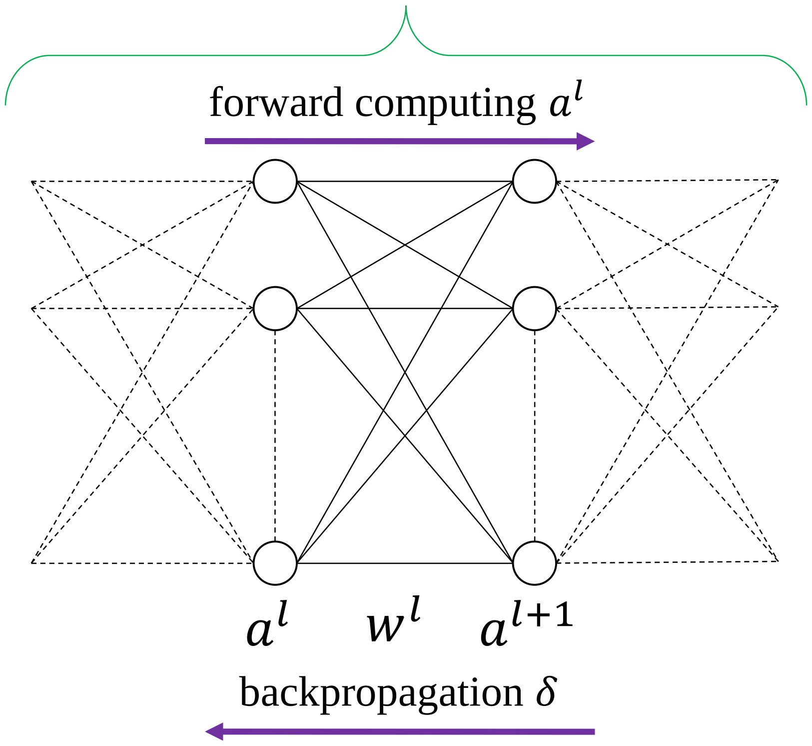
- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

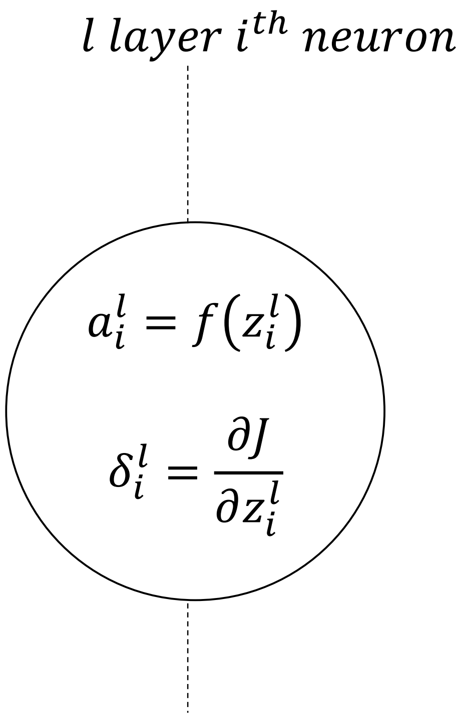
Only One Page to Understand BP

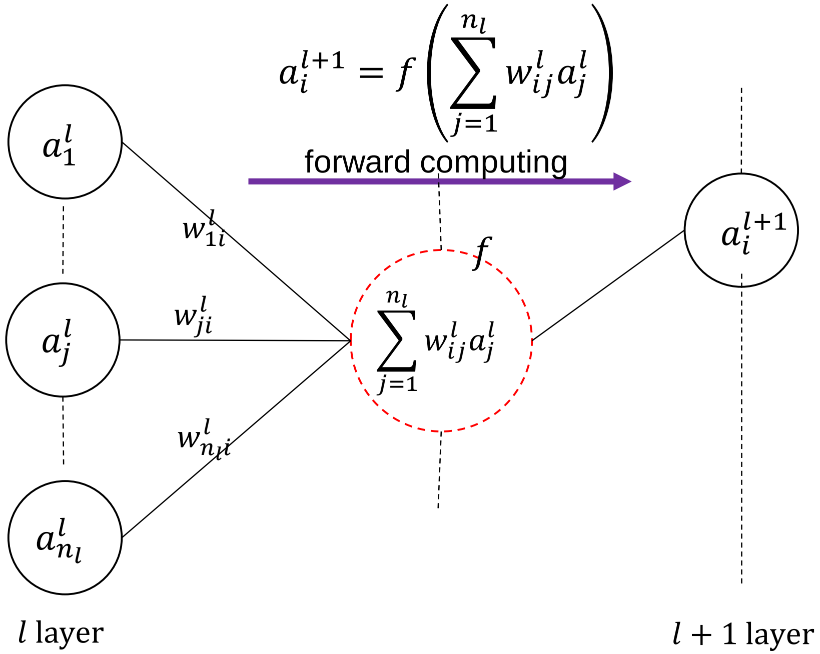
Cost function: $J(w^1, \cdots, w^L)$

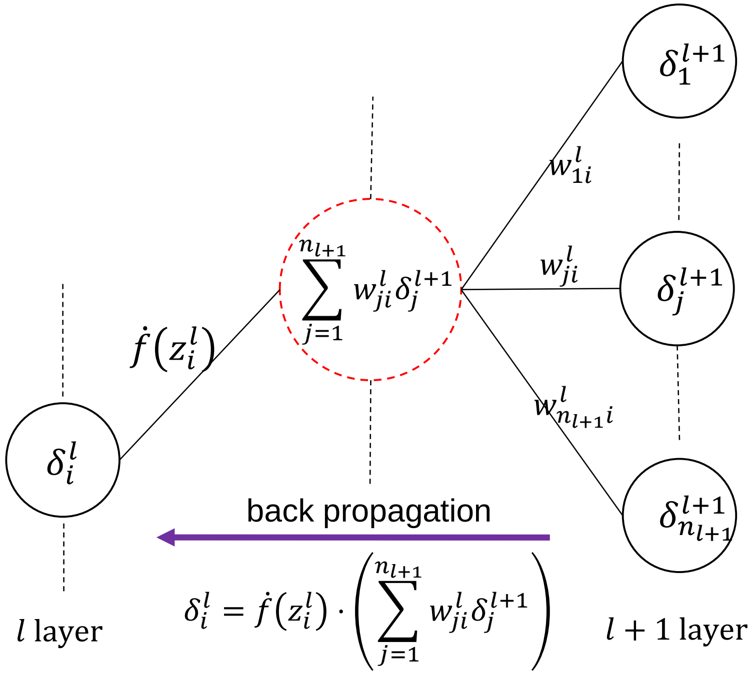
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \dfrac{\partial J}{\partial w_{ji}^l}$

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





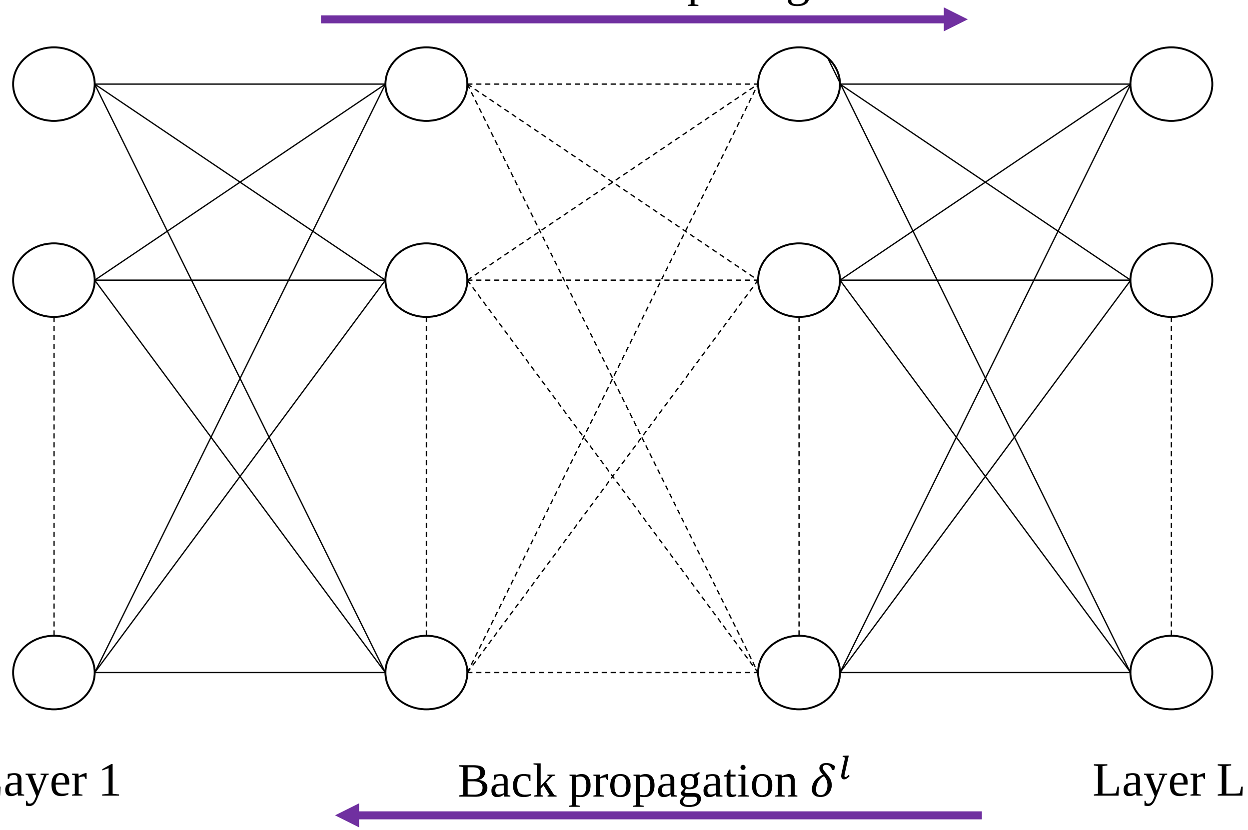


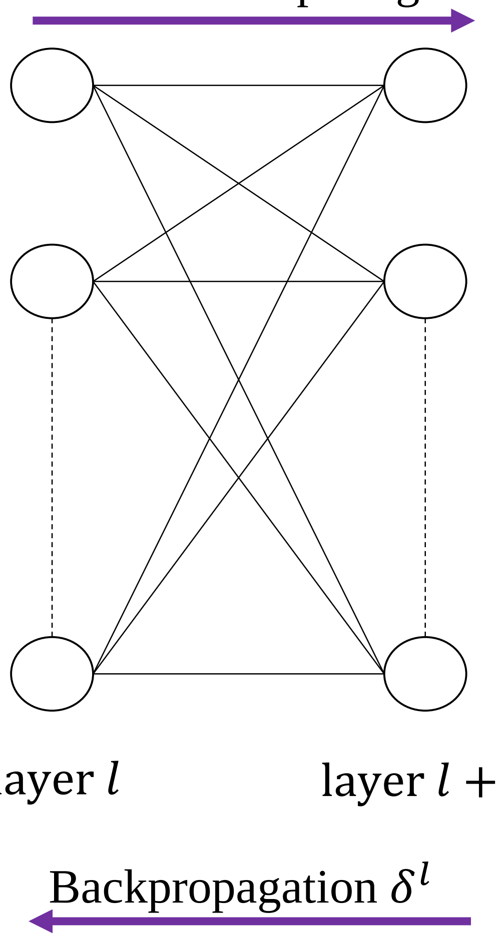


Outline

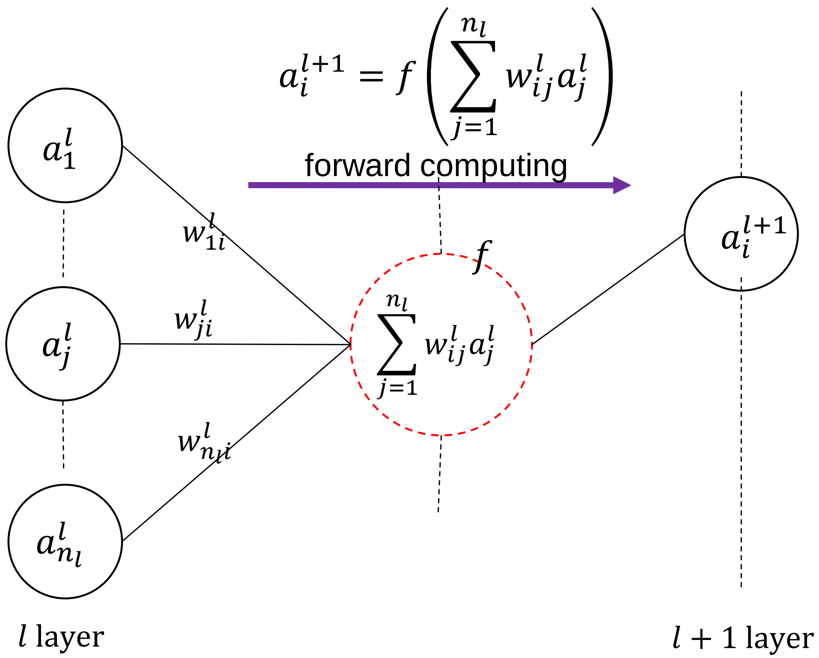
- Brief Review of Neural Networks Structure
- Network Performance: Cost Function
- Steepest Gradient Method
- Backpropagation
- Three Pages to Understand BP
- Only One Page to Understand BP
- The BP Algorithm
- Assignment

The BP Algorithm





The BP Algorithm



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

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

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

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

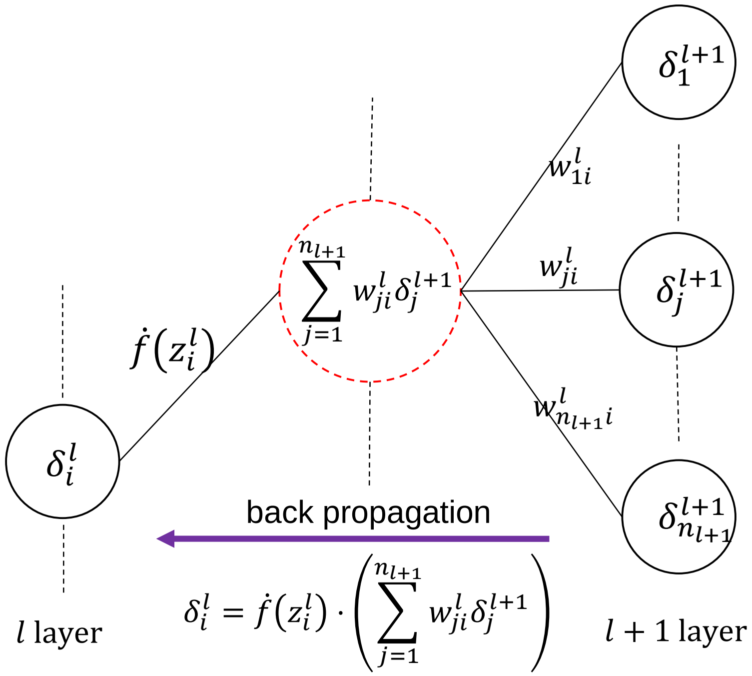
end

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

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

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

end

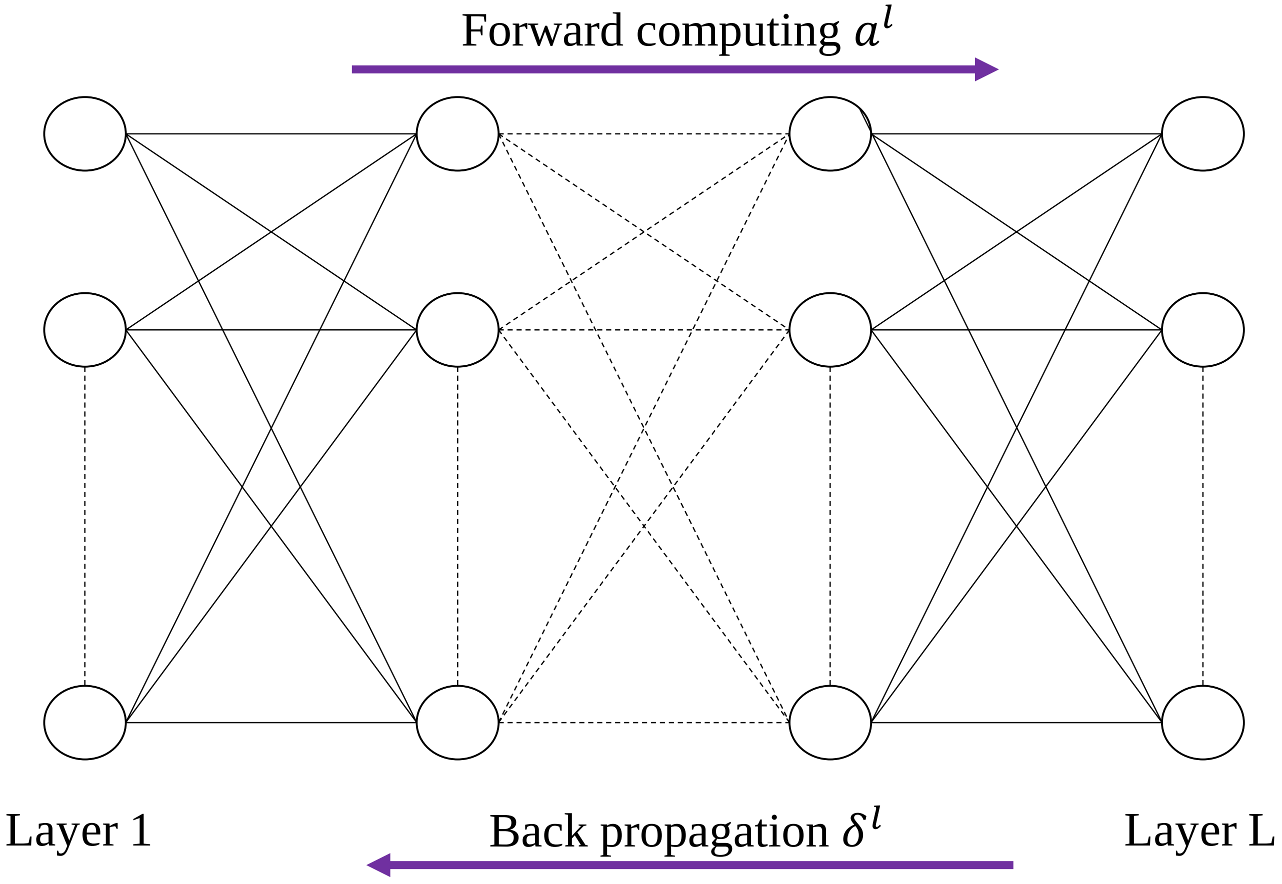


The BP Algorithm

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

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 $l = 1{:}L$

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

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 by MATLAB.

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

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 $l = 1: L$
$fc(W^l, a^l)$;
end

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

for $l = L - 1: 1$
$bc(\delta^{l+1})$;
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^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 $l = 1: L$
$fc(W^l, a^l)$;
end

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

for $l = L - 1: 1$
$bc(\delta^{l+1})$;
end

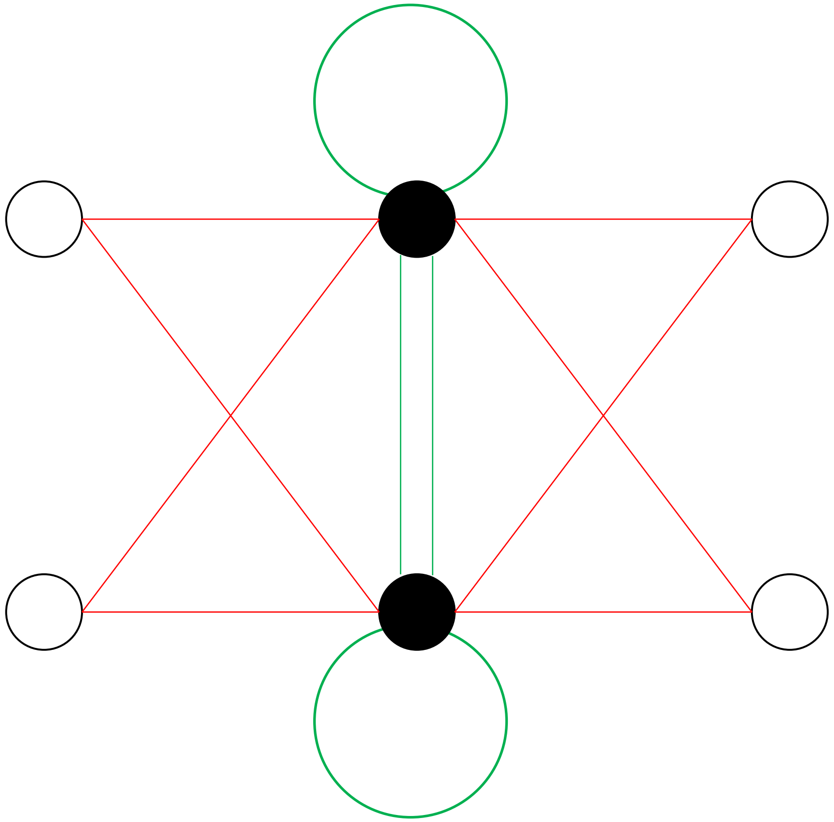
Step 4. Updating

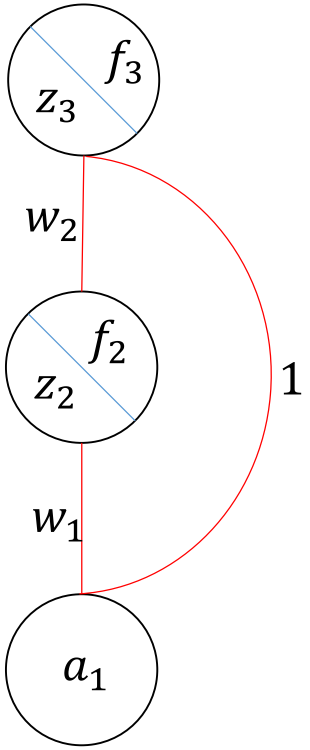
$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J(x, y^L)}{\partial w_{ji}^l}$$

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

Assignment

Assignment 2: Reform the following two networks to be in standard form, i.e., no any connection in any layer, no connection across any layer.





Thanks