Applied Deep Learning

Backpropagation for Optimization

March 10th, 2020 **http://adl.miulab.tw**

國立臺灣大學 National Taiwan University

2

Parameter Optimization

最佳化參數

Notation Summary

a_i^l : output of a neuron

a^l : output vector of a layer

z_i^l : input of activation function

z^l : input vector of activation function for a layer

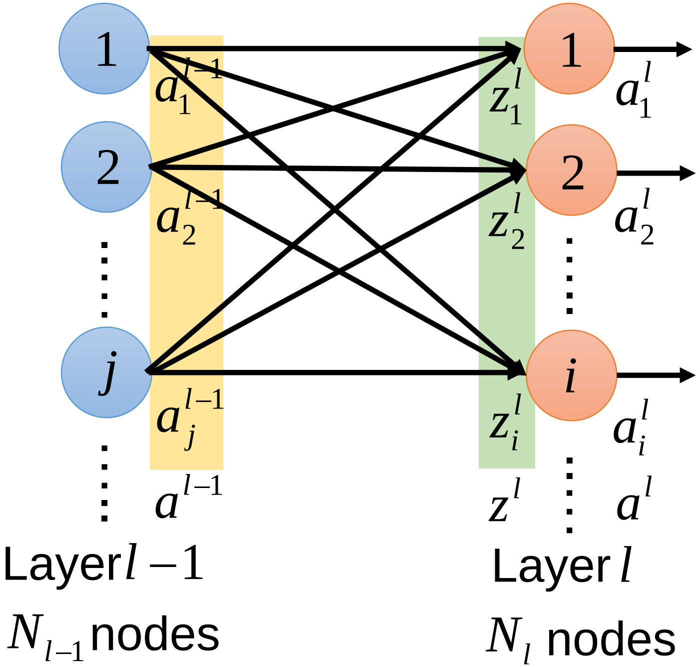
w_{ij}^l : a weight

W^l : a weight matrix

b_i^l : a bias

b^l : a bias vector

Layer Output Relation – from a to z



$$z_1^l = w_{11}^1 a_1^{l-1} + w_{12}^1 a_2^{l-1} + \cdots + b_1^l$$

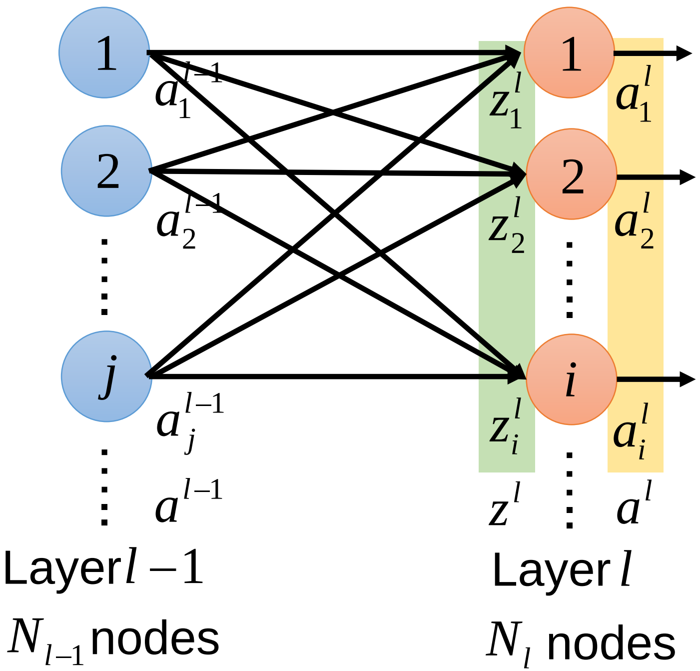
$$z_i^l = w_{i1}^1 a_1^{l-1} + w_{i2}^1 a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

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

Layer Output Relation – from *z* to *a*

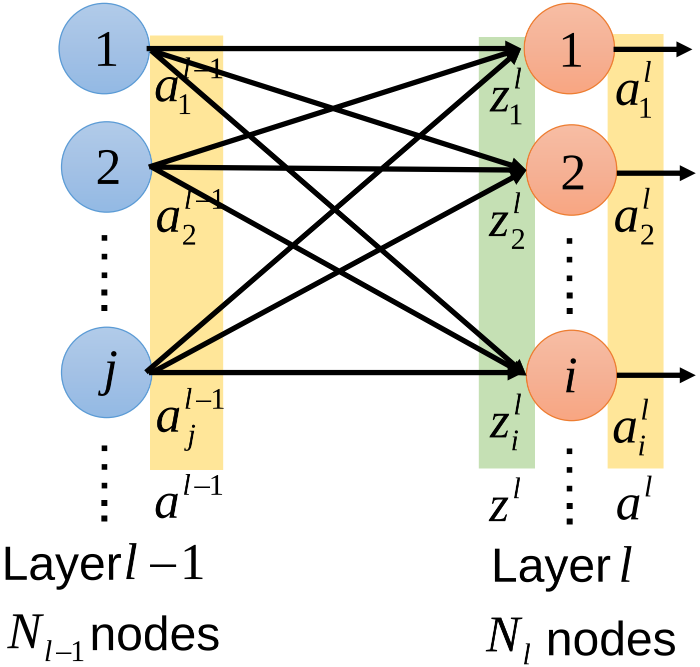


$$a_i^l = \sigma\left(z_i^l\right)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma\left(z_1^l\right) \\ \sigma\left(z_2^l\right) \\ \vdots \\ \sigma\left(z_i^l\right) \\ \vdots \end{bmatrix}$$

$$a^l = \sigma\left(z^l\right)$$

6 Layer Output Relation



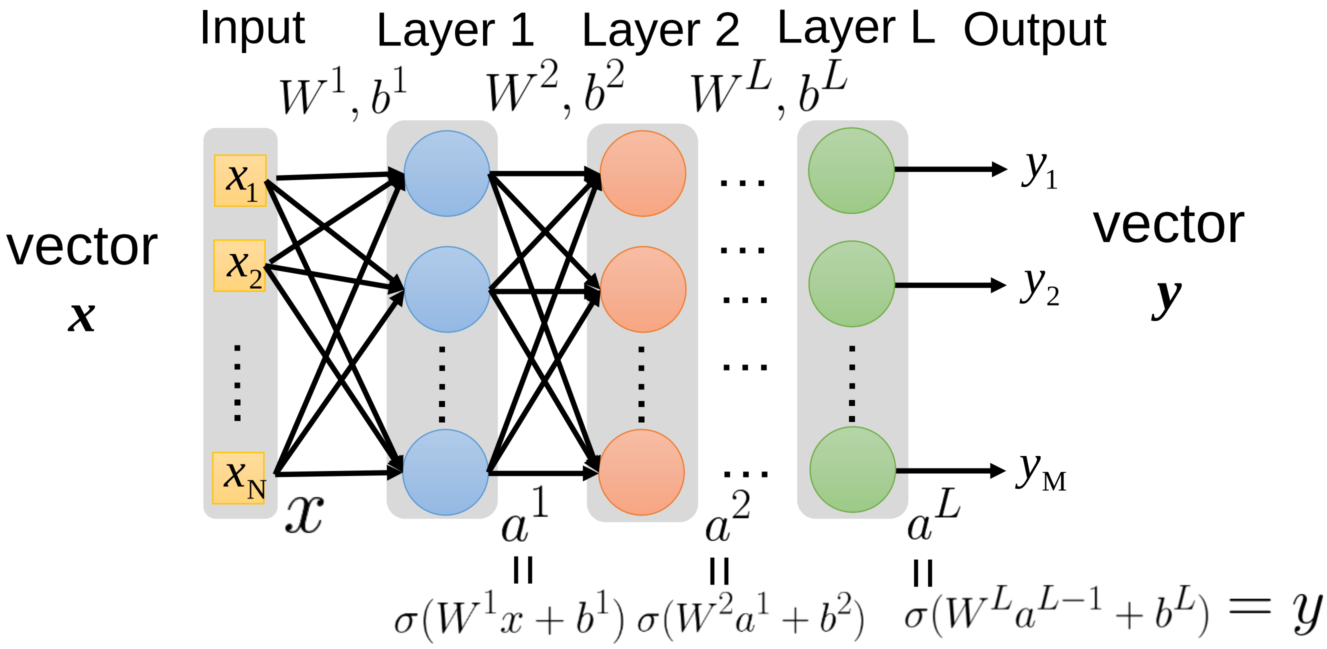
$$z^l = W^l a^{l-1} + b^l$$

$$a^l = \sigma(z^l)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

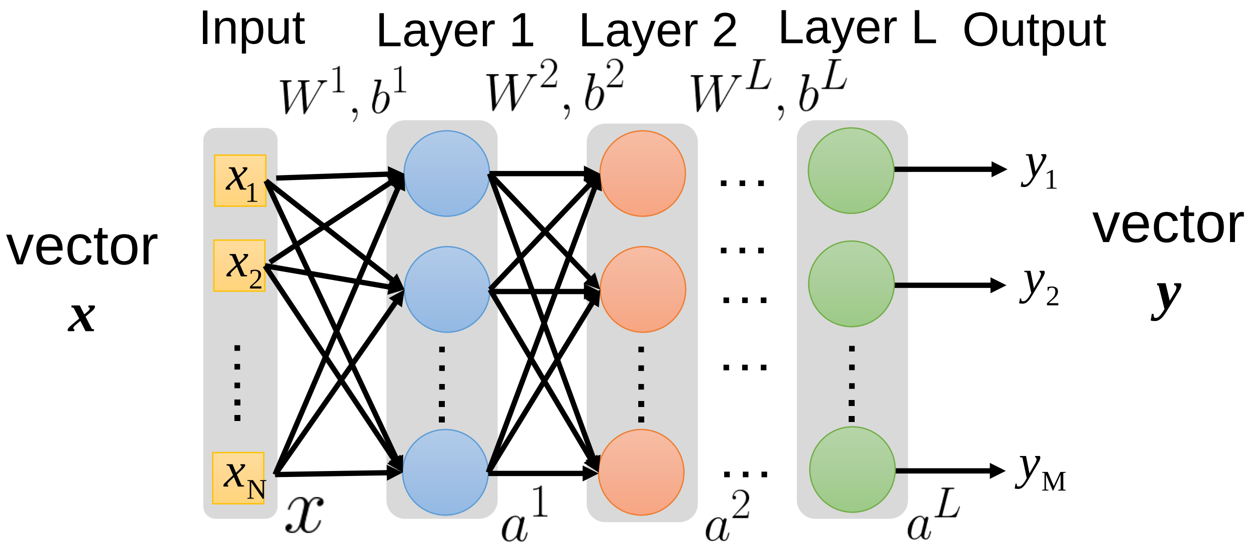
Neural Network Formulation

- Fully connected feedforward network $f : R^N \rightarrow R^M$



Neural Network Formulation

- Fully connected feedforward network $f : R^N \to R^M$



$$y = f(x) = \sigma(W^L \cdots \sigma(W^2\sigma(W^1x + b^1) + b^2) \ldots + b^L)$$

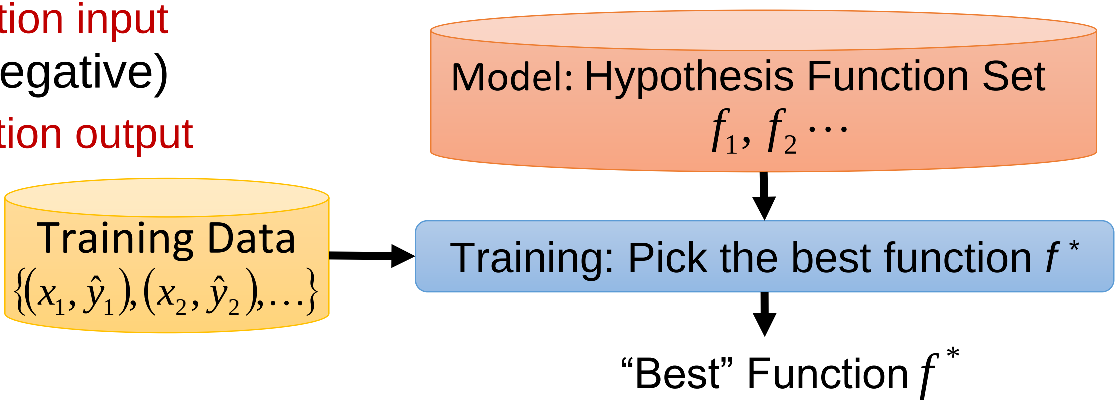
Loss Function for Training

x : “It claims too much.”

function input

$\hat{y}$: - (negative)

function output



A “Good” function: $f(x;\theta) \sim \hat{y} \Rightarrow \|\hat{y} - f(x;\theta)\| \approx 0$

Define an example loss function: $C(\theta) = \sum_k \|\hat{y}_k - f(x_k;\theta)\|$

sum over the error of all training samples

Gradient Descent for Neural Network

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

$$\theta = \left\{W^1, b^1, W^2, b^2, \cdots W^L, b^L\right\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$

}

Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

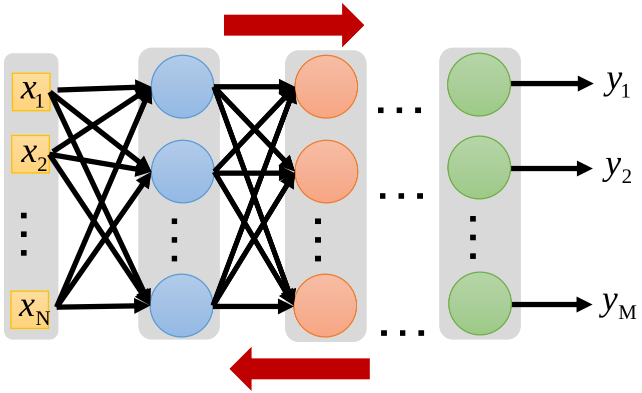
11

Backpropagation

如何有效率地計算大量參數呢？

Forward v.s. Back Propagation

- In a feedforward neural network
  - forward propagation
    - from input x to output y information flows forward through the network
    - during training, forward propagation can continue onward until it produces a scalar cost $C(\theta)$
  - back-propagation
    - allows the information from the cost to then flow backwards through the network, in order to compute the **gradient**
    - can be applied to any function



Chain Rule

$$\Delta w \to \Delta x \to \Delta y \to \Delta z$$

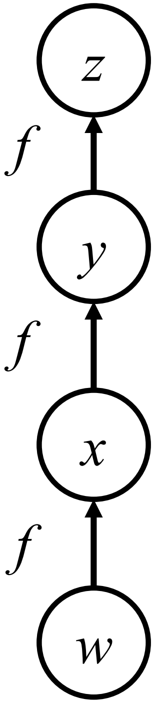
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= f'(y)f'(x)f'(w)$$

forward propagation for cost

$$= f'(f(f(w)))f'(f(w))f'(w)$$

back-propagation for gradient



Gradient Descent for Neural Network

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

$$\theta = \left\{ W^1, b^1, W^2, b^2, \cdots W^L, b^L \right\}$$

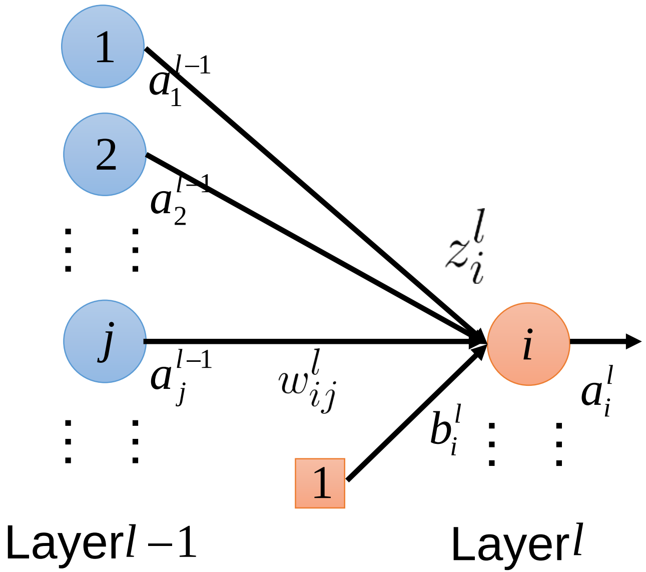
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm
Initialization: start at θ^0
while($\theta^{(i+1)} \neq \theta^i$)
{
compute gradient at θ^i
update parameters
$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$
}

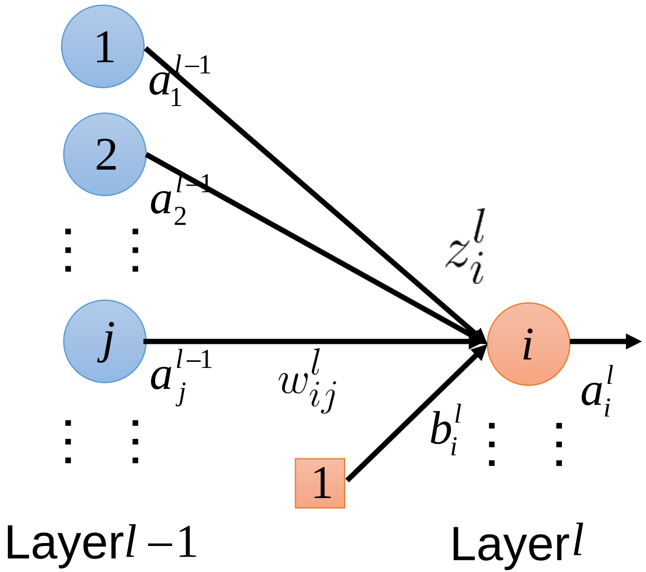
Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

$\partial C(\theta) / \partial w_{ij}^{l}$



$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

$\partial z_i^l / \partial w_{ij}^l \ (l > 1)$

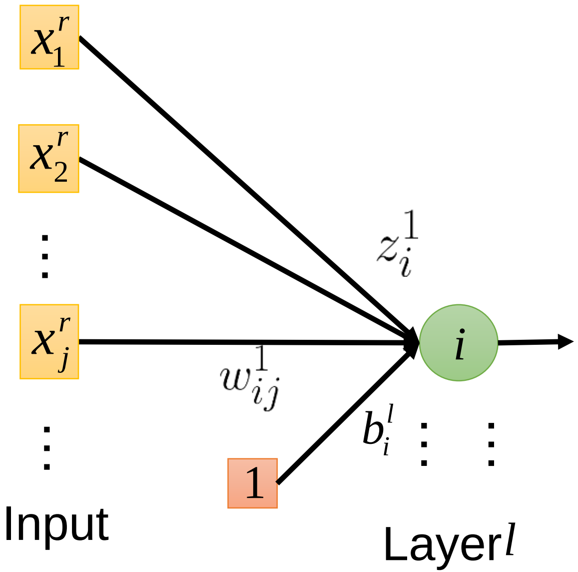


$$z^l = W^l a^{l-1} + b^l$$

$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

$\partial z_i^l / \partial w_{ij}^l \; (l = 1)$

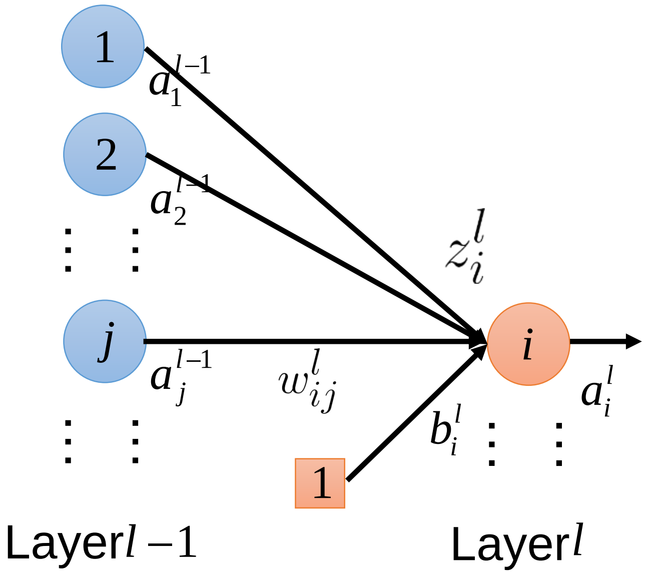


$$z^1 = W^1 x + b^1$$

$$z_i^1 = \sum_j w_{ij}^1 x_j + b_i^1$$

$$\frac{\partial z_i^1}{\partial w_{ij}^1} = x_j$$

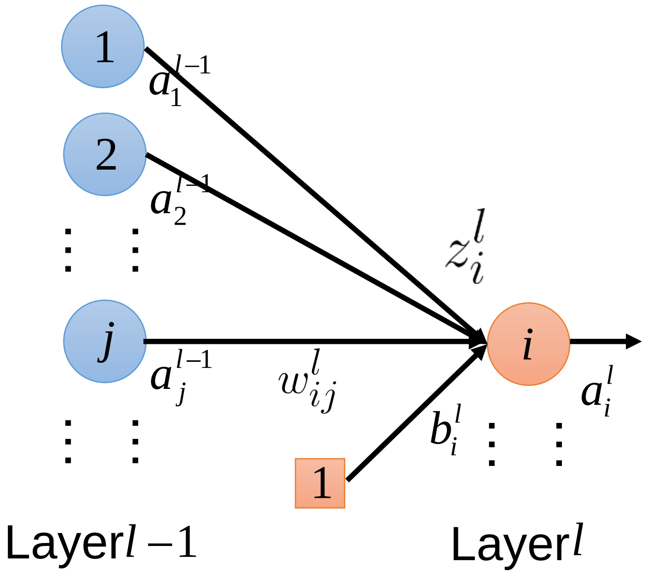
$\partial C(\theta) / \partial w_{ij}^{l}$



$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

$$\frac{\partial z_i^{l}}{\partial w_{ij}^{l}} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$

$\partial C(\theta) / \partial w_{ij}^l$

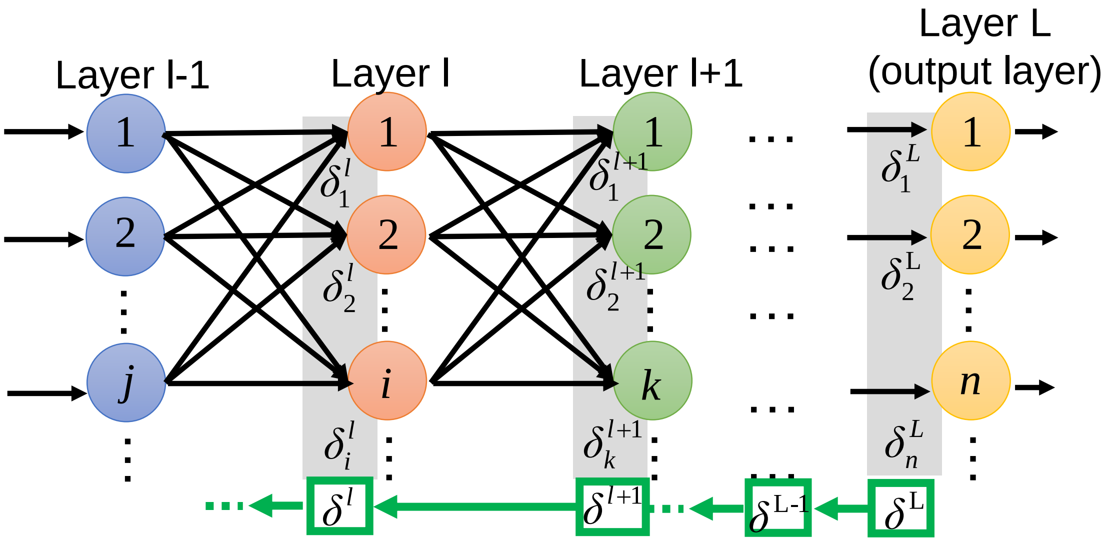


$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$\partial C(\theta)/\partial z_i^l$

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l : the propagated gradient corresponding to the l-th layer



Idea: computing δ^l layer by layer (from δ^L to δ^1) is more efficient

$\partial C(\theta)/\partial z_i^l = \delta_i^l$

- Idea: from L to 1
  - ① Initialization: compute δ^L
  - ② Compute δ^l based on δ^{l+1}

$\partial C(\theta)/\partial z_i^l = \delta_i^l$

- Idea: from L to 1
  - ① **Initialization: compute δ^L**
  - ② Compute δ^l based on δ^{l+1}

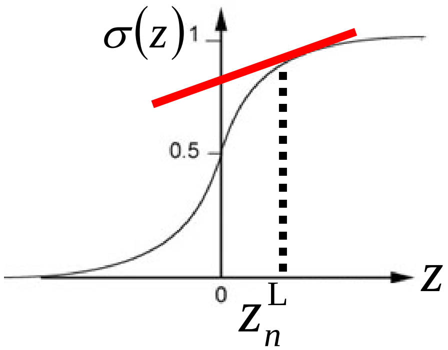
$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial y_i}\frac{\partial y_i}{\partial z_i^L}$$

$$\Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

$\partial C/\partial y_i$ depends on the loss function

$\partial C(\theta)/\partial z_i^l = \delta_i^l$

- Idea: from L to 1
  - ① **Initialization: compute δ^L**
  - ② Compute δ^l based on δ^{l+1}



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

$$\Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

$$= \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^L} \qquad y_i = a_i^L = \sigma(z_i^L)$$

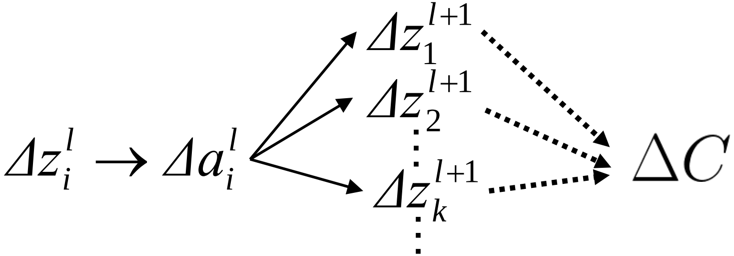
$$= \frac{\partial C}{\partial y_i} \sigma'(z_i^L)$$

$$\sigma'(z^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_i^L) \\ \vdots \end{bmatrix} \qquad \nabla C(y) = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_i} \\ \vdots \end{bmatrix}$$

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

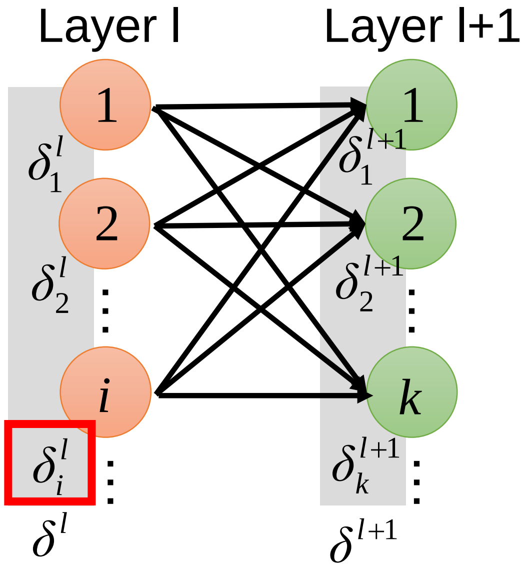
$\partial C(\theta)/\partial z_i^l = \delta_i^l$

- Idea: from L to 1
  - ① Initialization: compute δ^L
  - ② **Compute δ^l based on δ^{l+1}**



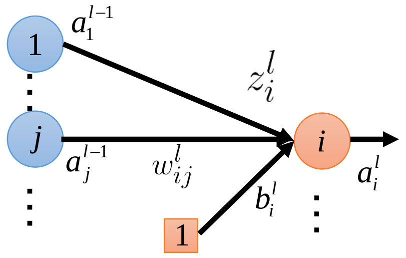
$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l}\right)$$

$$= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l}\right) \delta_i^{l+1}$$



25

$\partial C(\theta)/\partial z_i^l = \delta_i^l$



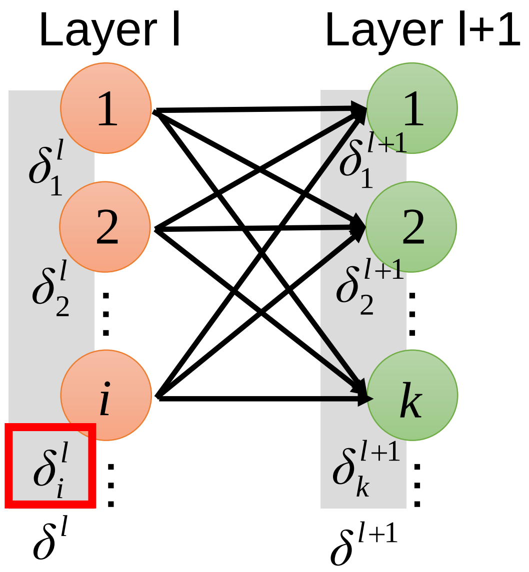
- Idea: from L to 1
  - ① Initialization: compute δ^L
  - ② **Compute δ^l based on δ^{l+1}**

$$\delta_i^l = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

$$= \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

$$= \sigma'(z_i) \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

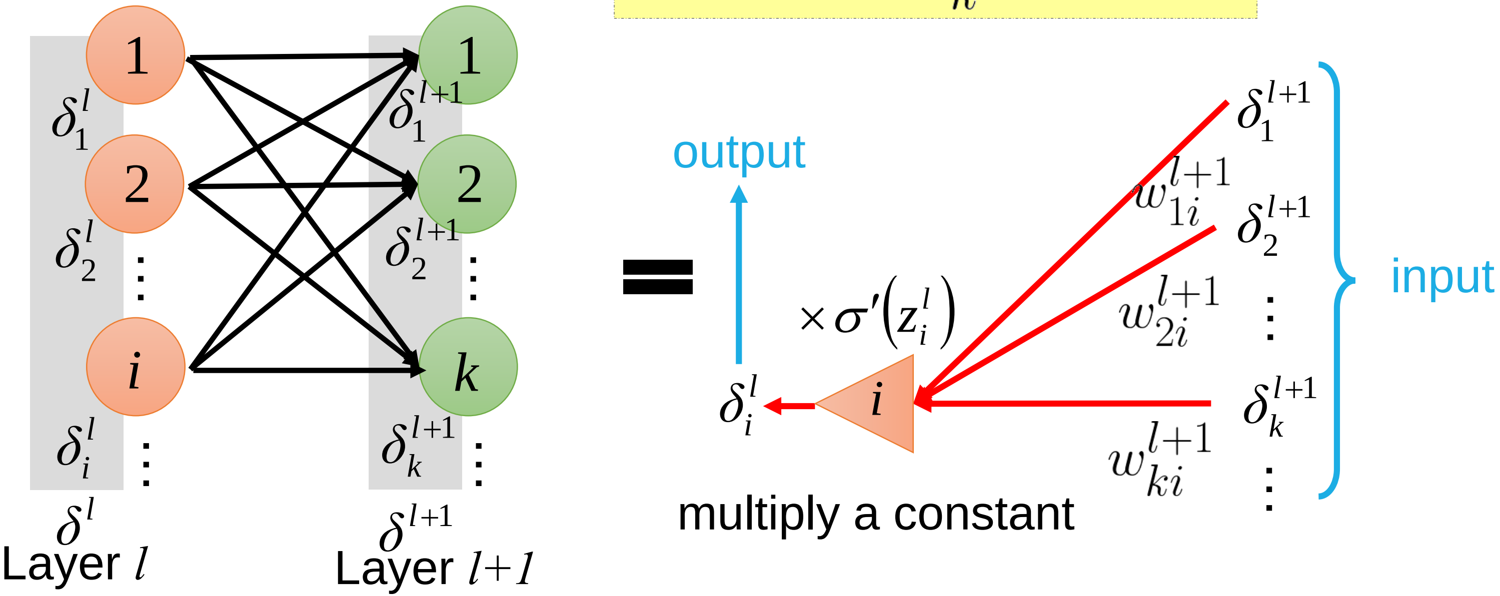
$$= \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$



$\partial C(\theta)/\partial z_i^l = \delta_i^l$

- Rethink the propagation

$$\delta_i^l = \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

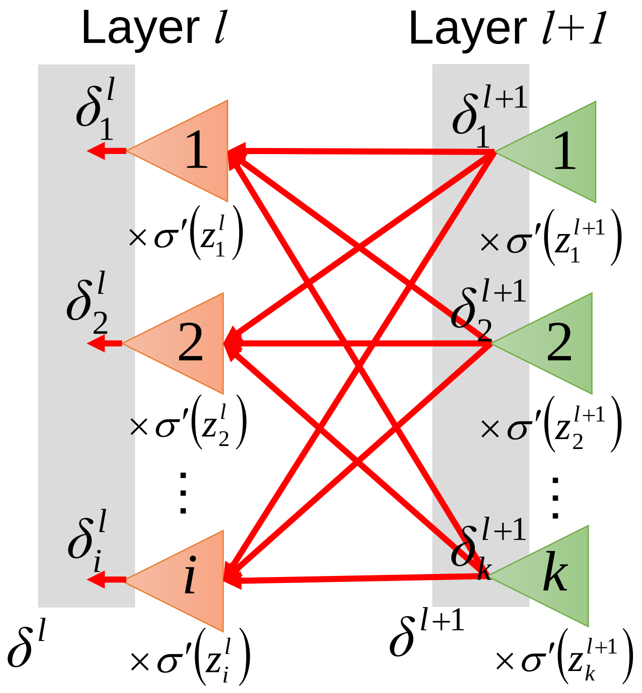


$\partial C(\theta)/\partial z_i^l = \delta_i^l$

$$\delta_i^l = \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

$$\sigma'(z^l) = \begin{bmatrix} \sigma'(z_1^l) \\ \sigma'(z_2^l) \\ \vdots \\ \sigma'(z_i^l) \\ \vdots \end{bmatrix}$$

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$

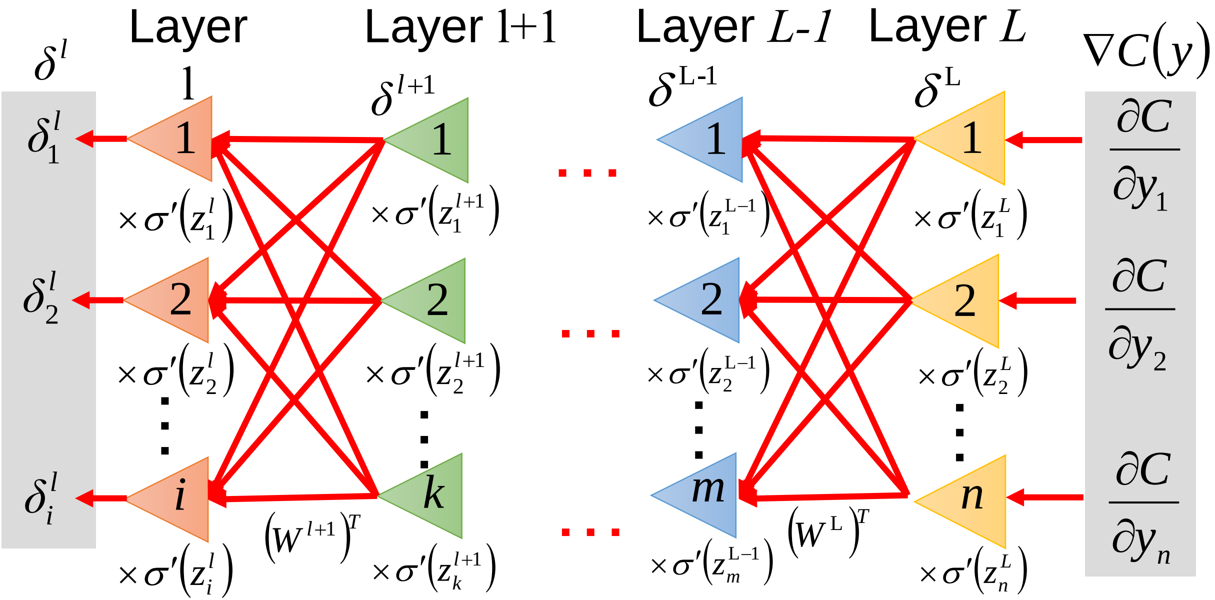


$$\partial C(\theta)/\partial z_i^l = \delta_i^l \qquad \frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l}$$

- Idea: from L to 1
  - ① Initialization: compute δ^L
  - ② Compute δ^{l-1} based on δ^l

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$



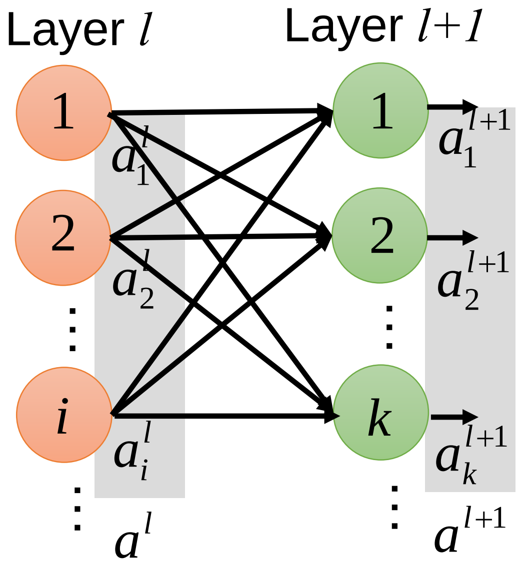
Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$

Forward Pass

$$z^1 = W^1 x + b^1 \qquad a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l \quad a^l = \sigma(z^l)$$
$$\vdots$$



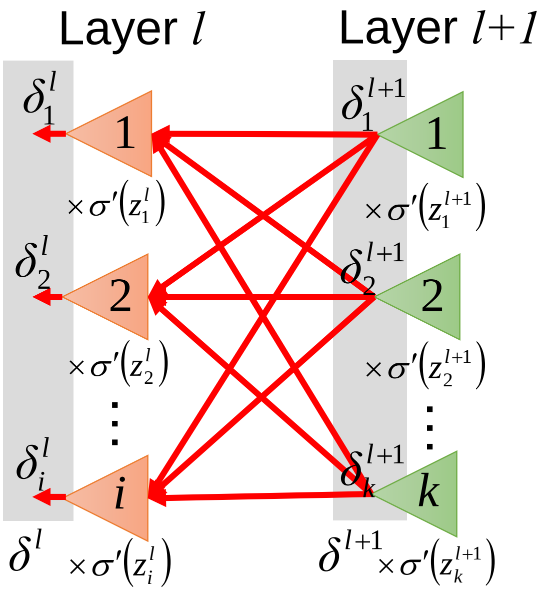
Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$



Gradient Descent for Optimization

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

$$\theta = \left\{W^1, b^1, W^2, b^2, \cdots W^L, b^L\right\}$$

$$W^l = \begin{bmatrix} w^l_{11} & w^l_{12} & \cdots \\ w^l_{21} & w^l_{22} & \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b^l_i \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w^l_{ij}} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b^l_i} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$

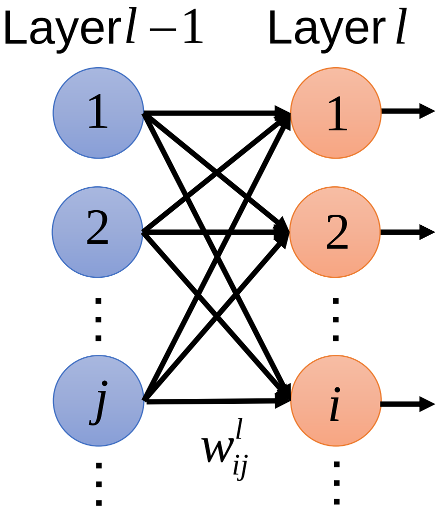
}

Concluding Remarks

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l

$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$



Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Compute the gradient based on two pre-computed terms from backward and forward passes