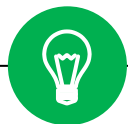


Applied Deep Learning

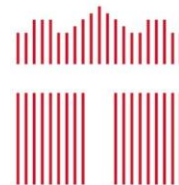


Backpropagation for Optimization



September 14th, 2023

<http://adl.miulab.tw>



National
Taiwan
University
國立臺灣大學

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

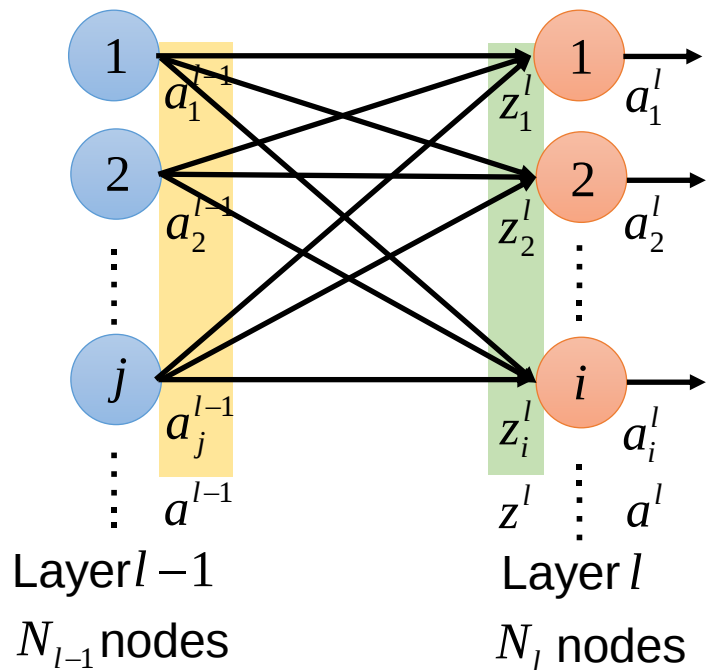
w_{ij}^l : a weight

W^l : a weight matrix

b_i^l : a bias

b^l : a bias vector

Layer Output Relation – from a to z



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

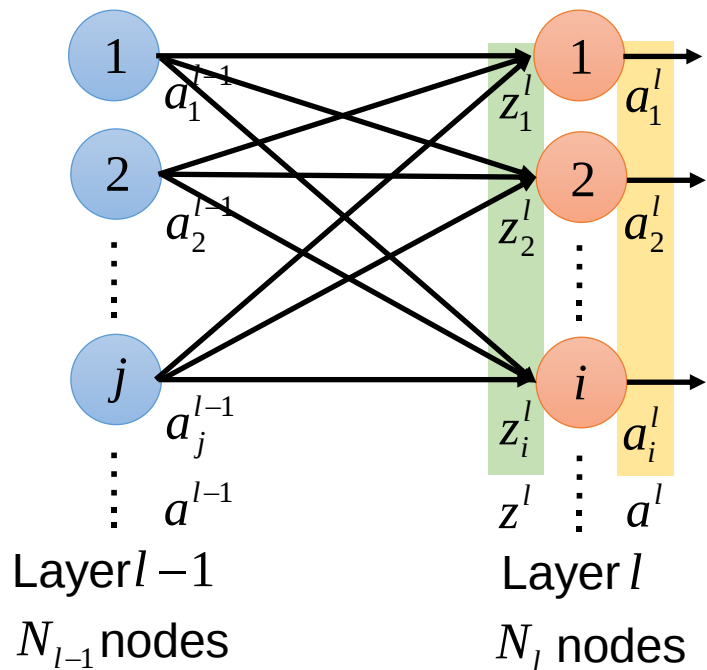
$$z_i^l = w_{i1}^1 a_1^{l-1} + w_{i2}^1 a_2^{l-1} + \dots + b_i^l$$

$$\vdots$$

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

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

Layer Output Relation – from z to a

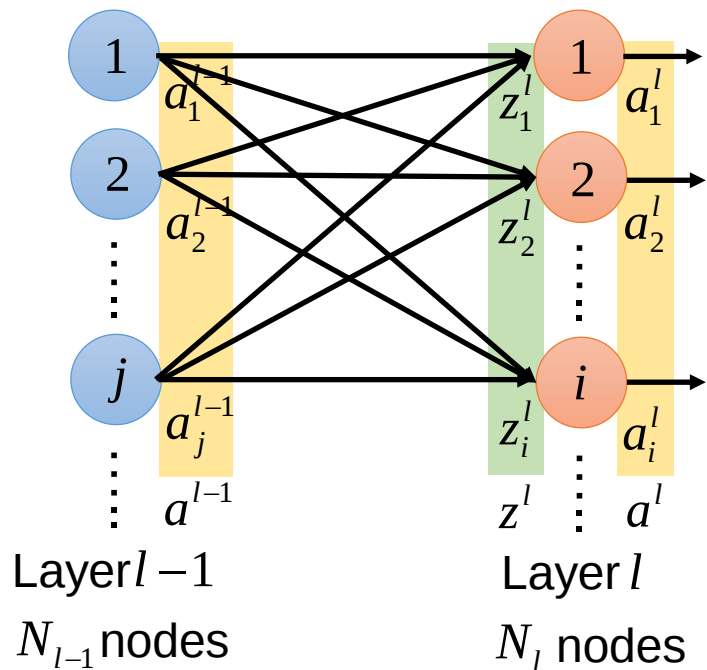


$$a_i^l = \sigma(z_i^l)$$

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

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

Layer Output Relation



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

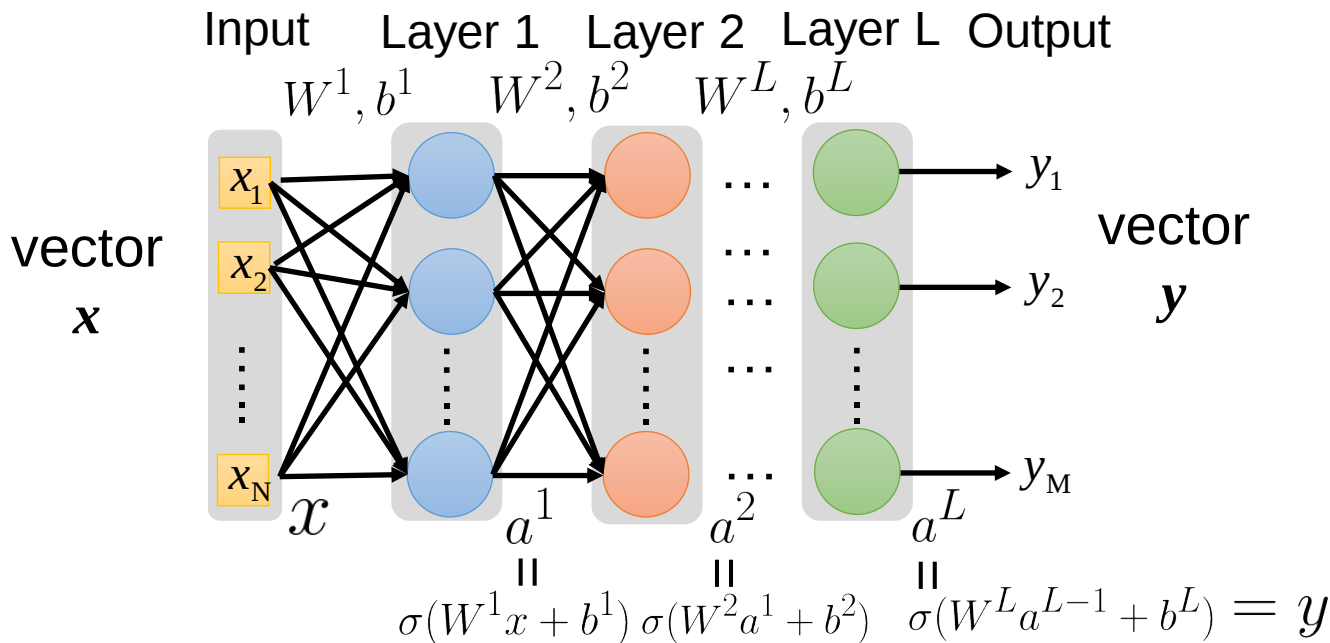
$$a^l = \sigma(z^l)$$



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

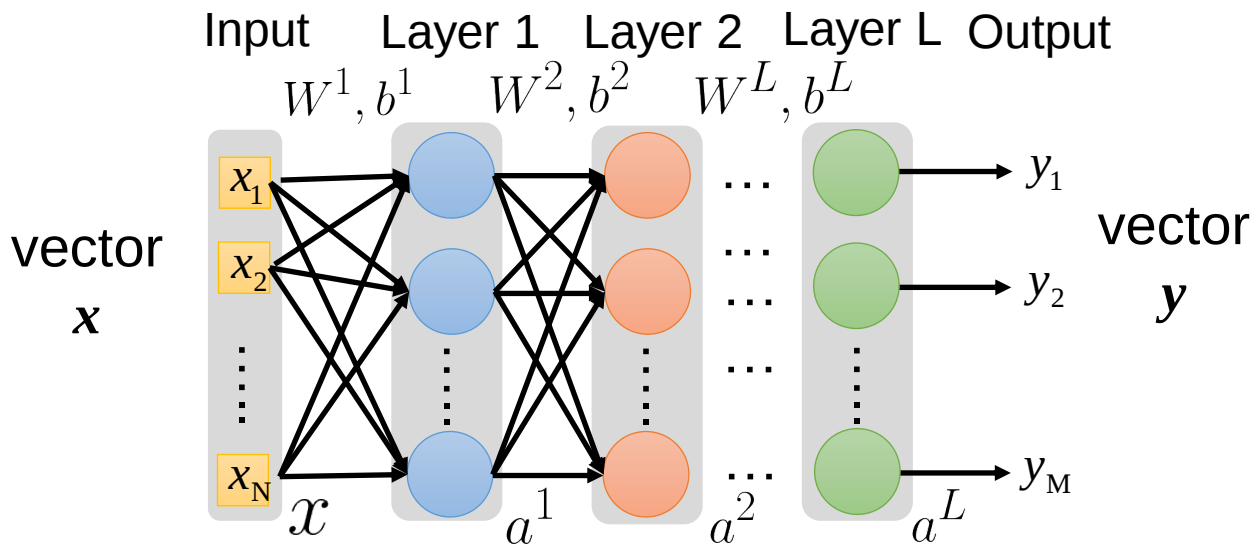
7 Neural Network Formulation

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



8 Neural Network Formulation

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



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

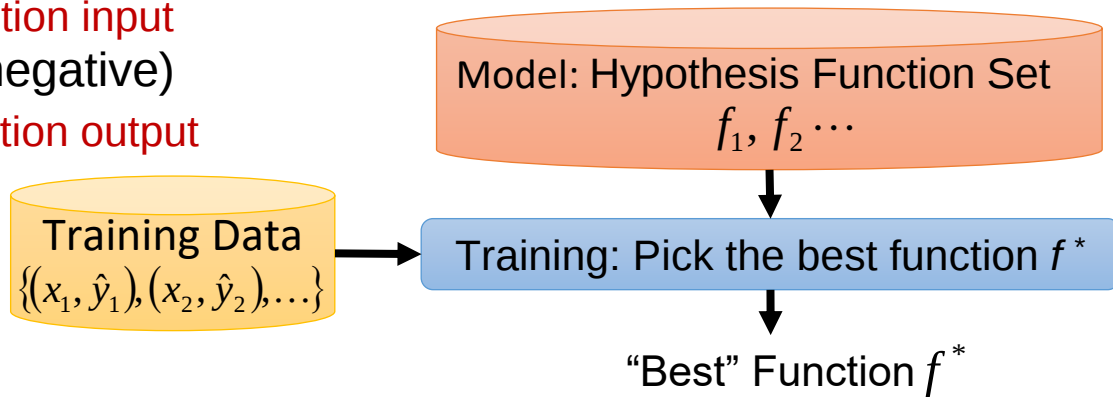
9 Loss Function for Training

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

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

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

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

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

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**.

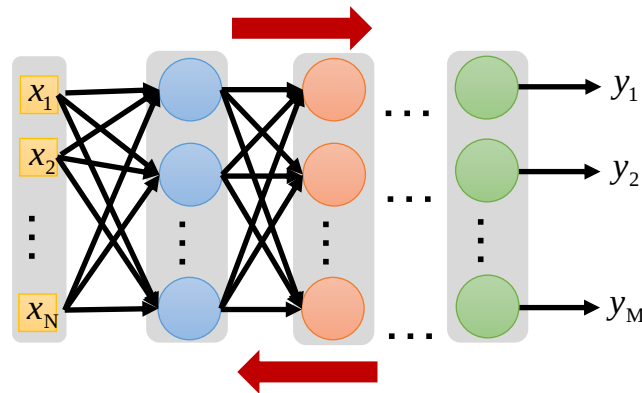
11

Backpropagation

如何有效率地計算大量參數呢？

12 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

$$\Delta w \rightarrow \Delta x \rightarrow \Delta y \rightarrow \Delta z$$

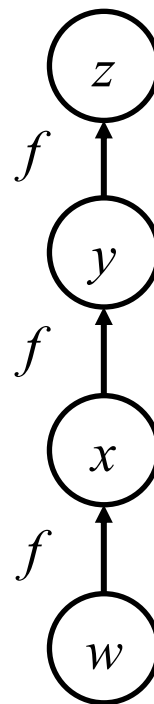
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w}$$

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

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

forward propagation for cost

back-propagation for gradient



Gradient Descent for Neural Network

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

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

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

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

Algorithm

Initialization: start at θ^0

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

{

 compute gradient at θ^i

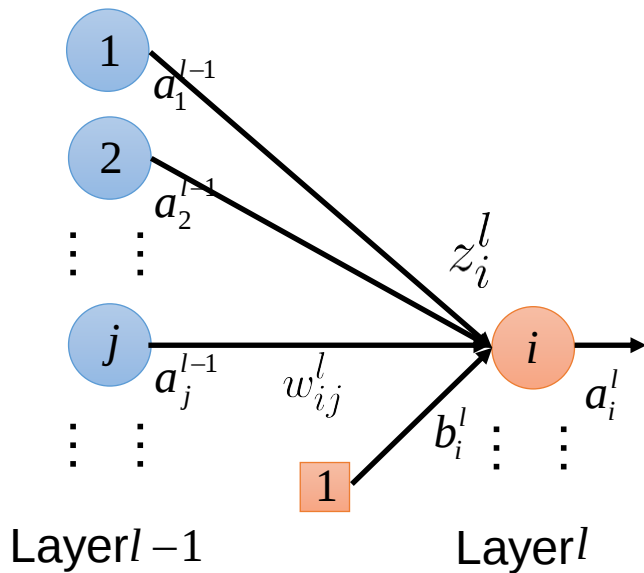
 update parameters

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

}

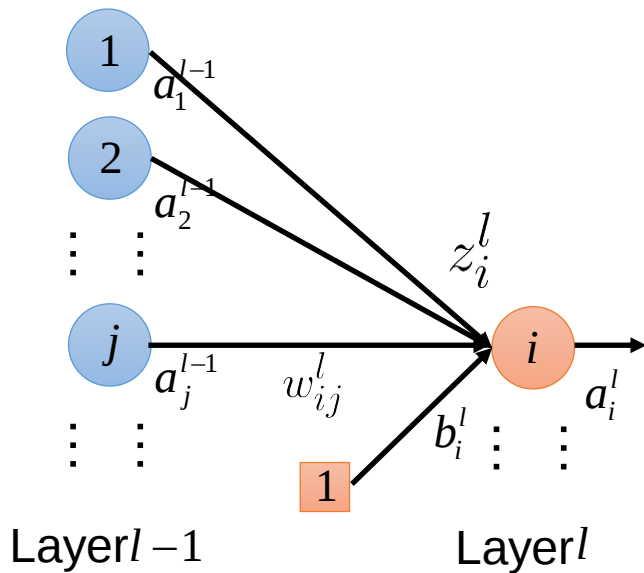
Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

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



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

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



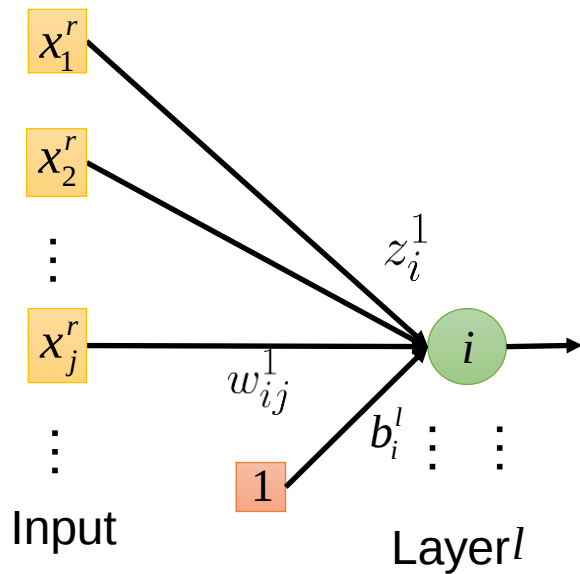
$$z^l = W^l a^{l-1} + b^l$$

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

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

17

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

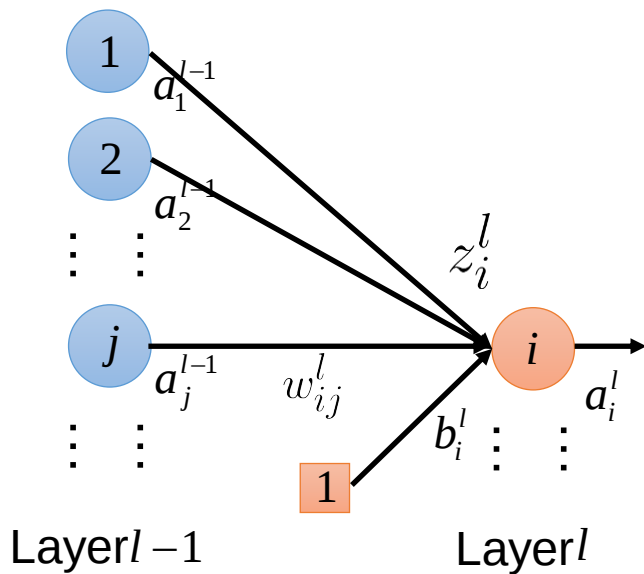


$$z^1 = W^1 x + b^1$$

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

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

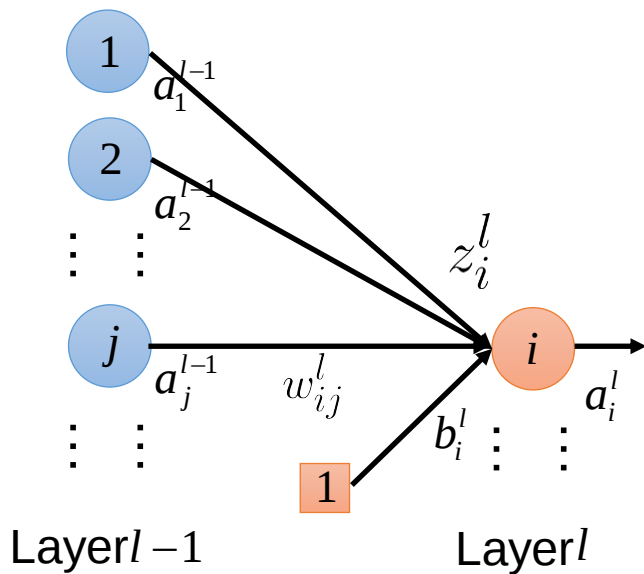
$$\partial C(\theta) / \partial w_{ij}^l$$



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

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

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

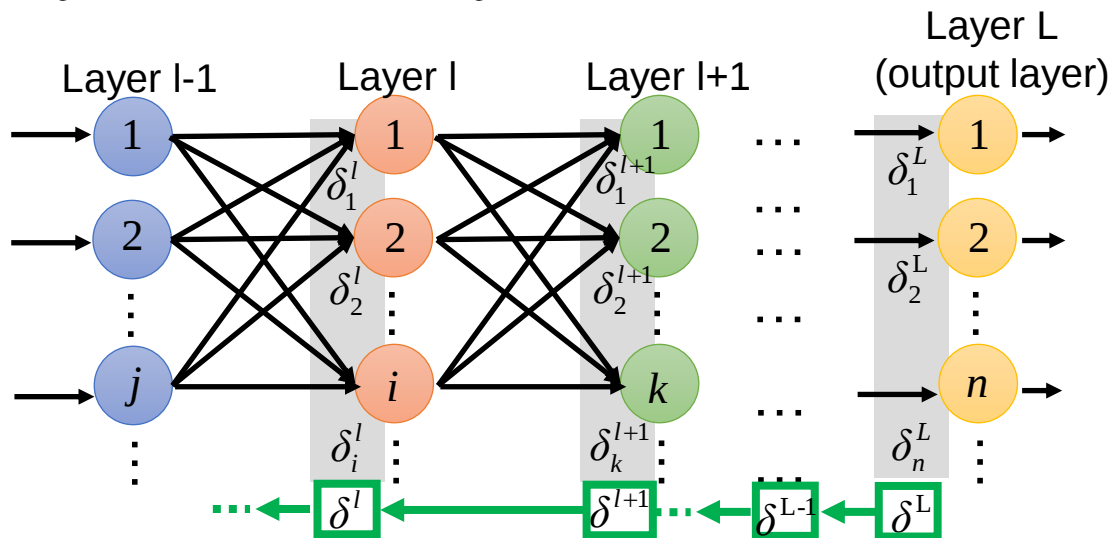


$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

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

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

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}

$$\begin{aligned} \delta_i^L &= \frac{\partial C}{\partial z_i^L} & \Delta z_i^L &\rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C \\ &= \boxed{\frac{\partial C}{\partial y_i}} \frac{\partial y_i}{\partial z_i^L} \end{aligned}$$

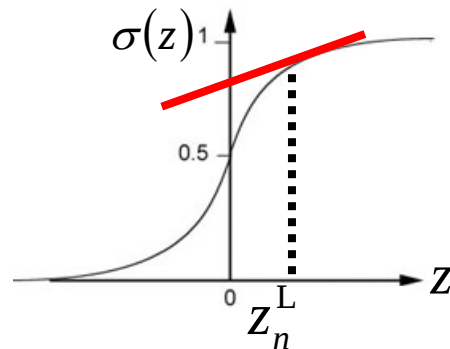
$\partial C / \partial y_i$ depends on the loss function

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

● Idea: from L to 1

① **Initialization: compute δ^L**

② Compute δ^l based on δ^{l+1}



$$\delta_i^L = \frac{\partial C}{\partial z_i^L} \quad \Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

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

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

$$\nabla C(y) = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_i} \\ \vdots \end{bmatrix}$$

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

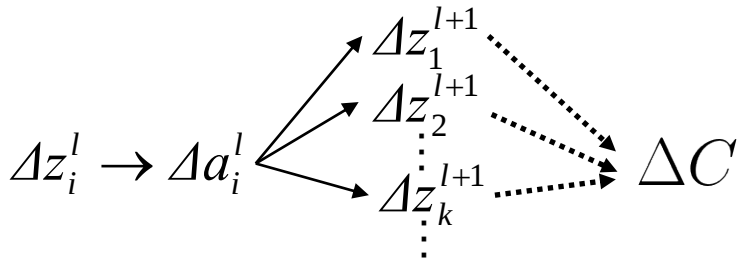
$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

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

● Idea: from L to 1

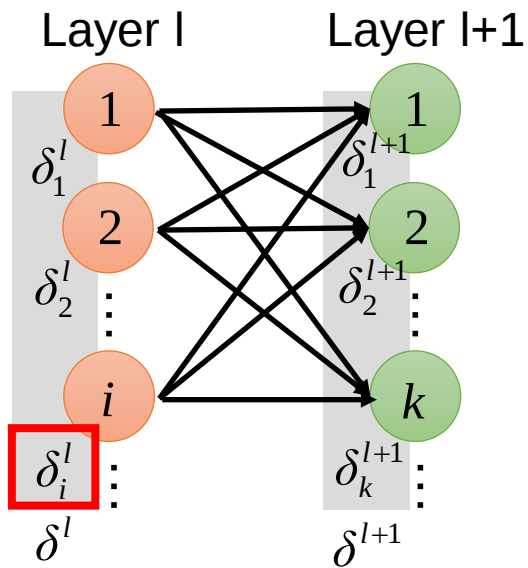
① Initialization: compute δ^L

② **Compute δ^l based on δ^{l+1}**



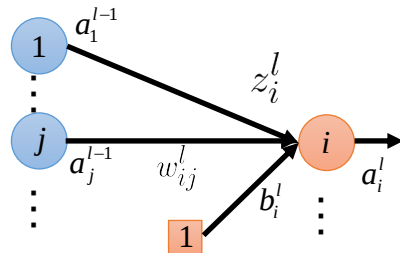
$$\begin{aligned} \delta_i^l &= \frac{\partial C}{\partial z_i^l} = \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \right) \\ &= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}}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

A red arrow points from the term δ_k^{l+1} in the second equation to the corresponding δ_k^{l+1} in the diagram on the right.



25

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

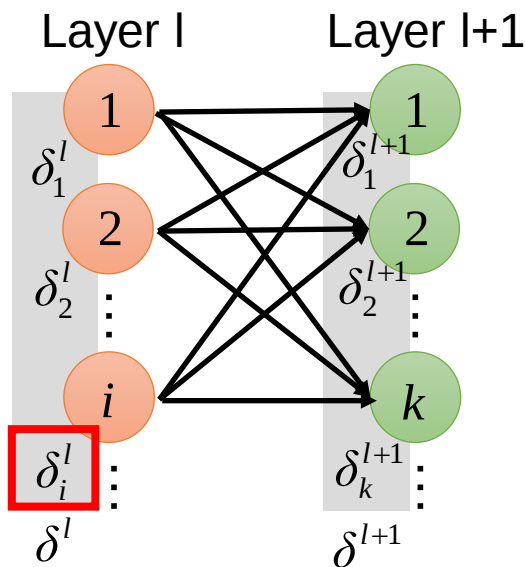


● Idea: from L to 1

① Initialization: compute δ^L

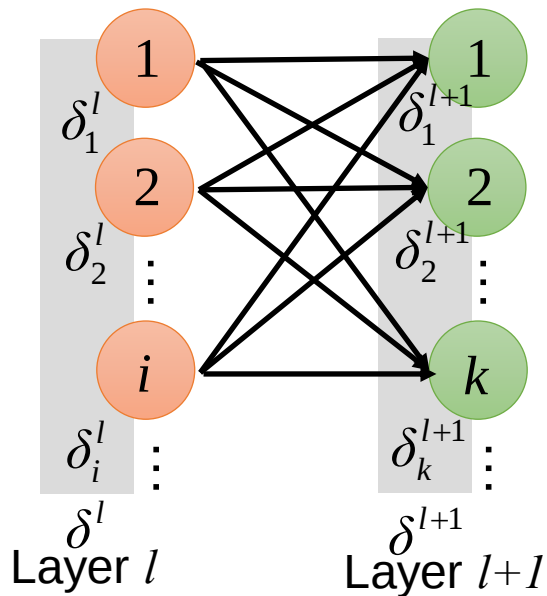
② Compute δ^l based on δ^{l+1}

$$\begin{aligned} \delta_i^l &= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1} \\ &= \sum_k w_{ki}^{l+1} a_i^l + b_k^{l+1} \\ &= \sigma'(z_i) \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1} \\ &= \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1} \end{aligned}$$

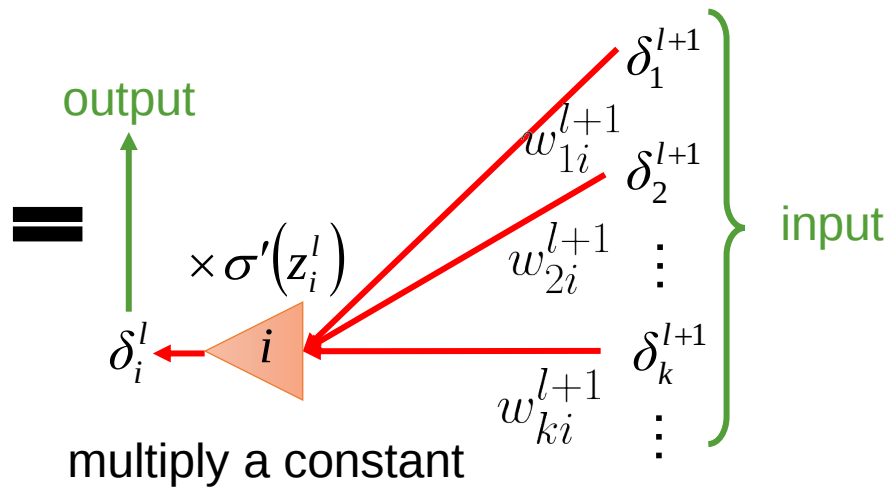


$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

- Rethink the propagation



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

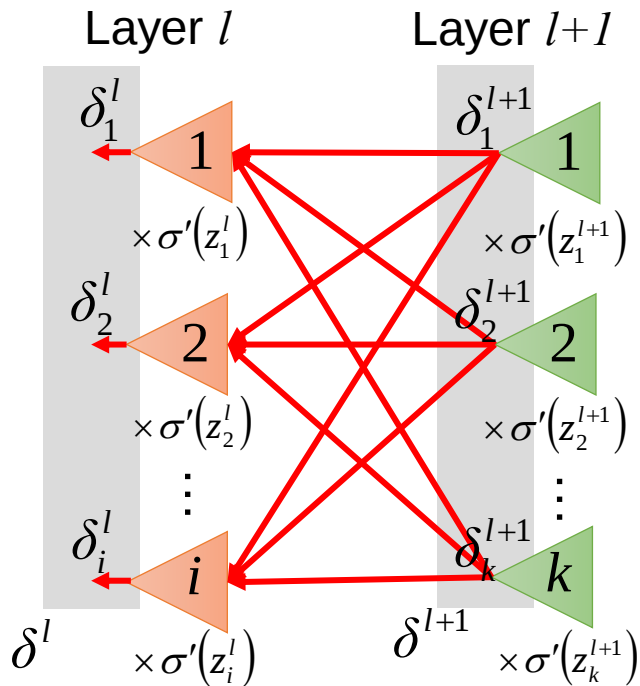


$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

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

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

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



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

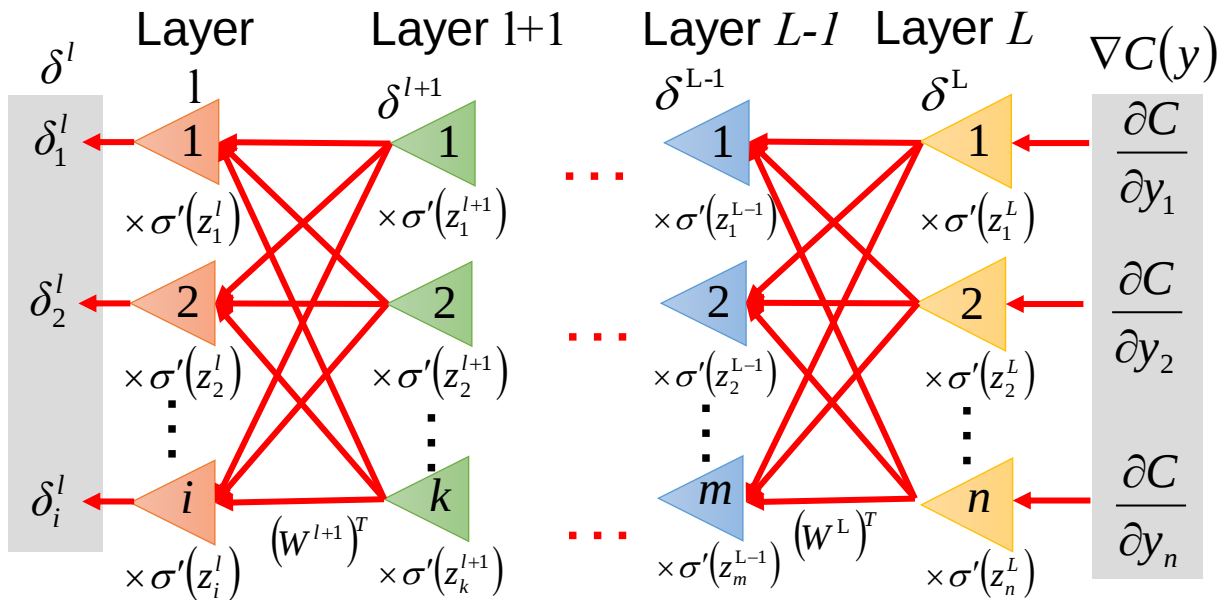
$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

● Idea: from L to 1

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

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

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



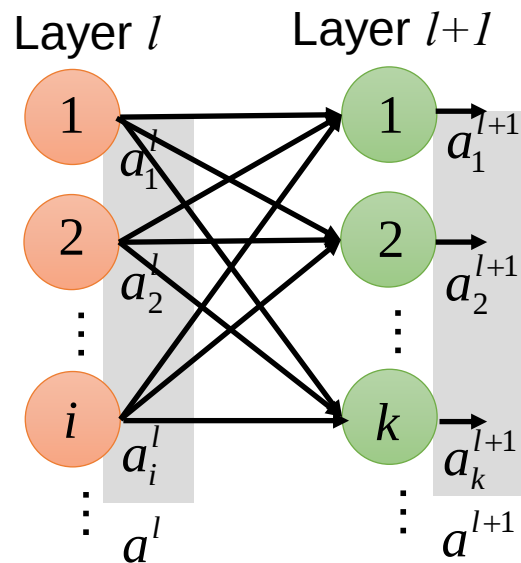
Backpropagation

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

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

Forward Pass

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



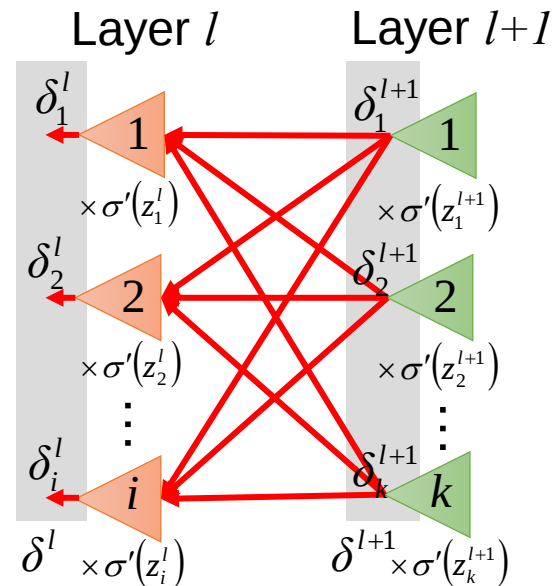
Backpropagation

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

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

Backward Pass

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



Gradient Descent for Optimization

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

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

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

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

Algorithm

Initialization: start at θ^0

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

{

 compute gradient at θ^i

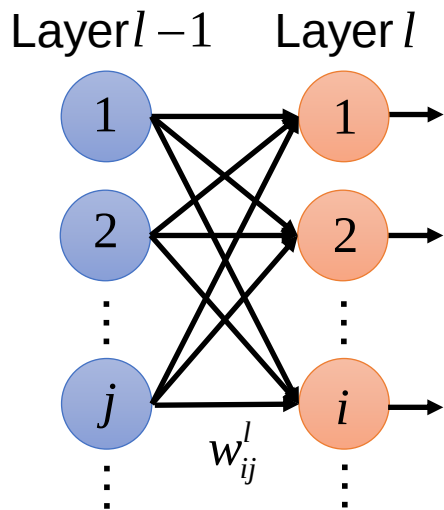
 update parameters

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

}

Concluding Remarks

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



$$\delta_i^l$$

Backward Pass

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

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

Forward Pass

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

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



Thanks!

Any questions ?

You can find the course information at

- <http://adl.miulab.tw>
- adl-ta@csie.ntu.edu.tw
- slido: #ADL2023
- YouTube: Vivian NTU MiuLab