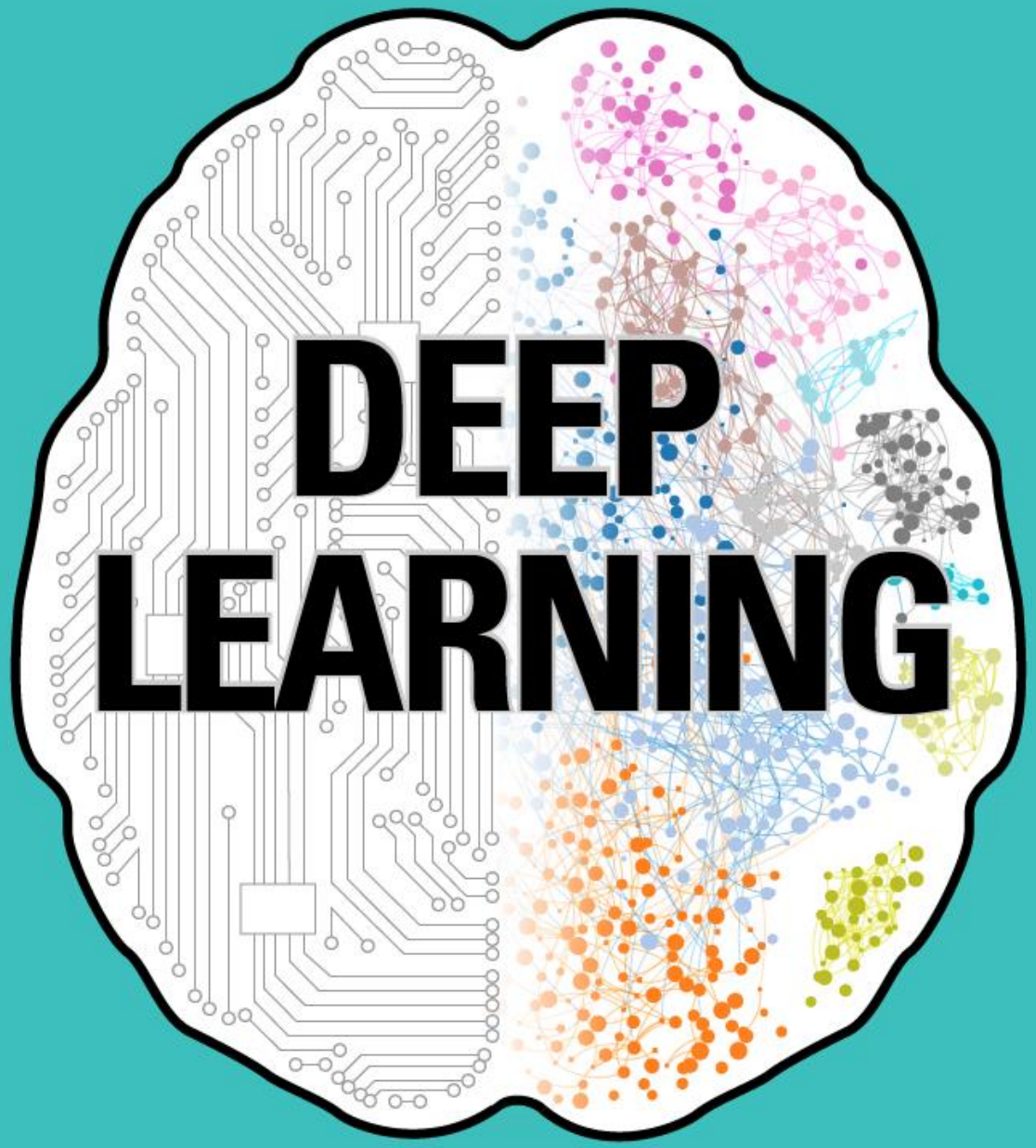
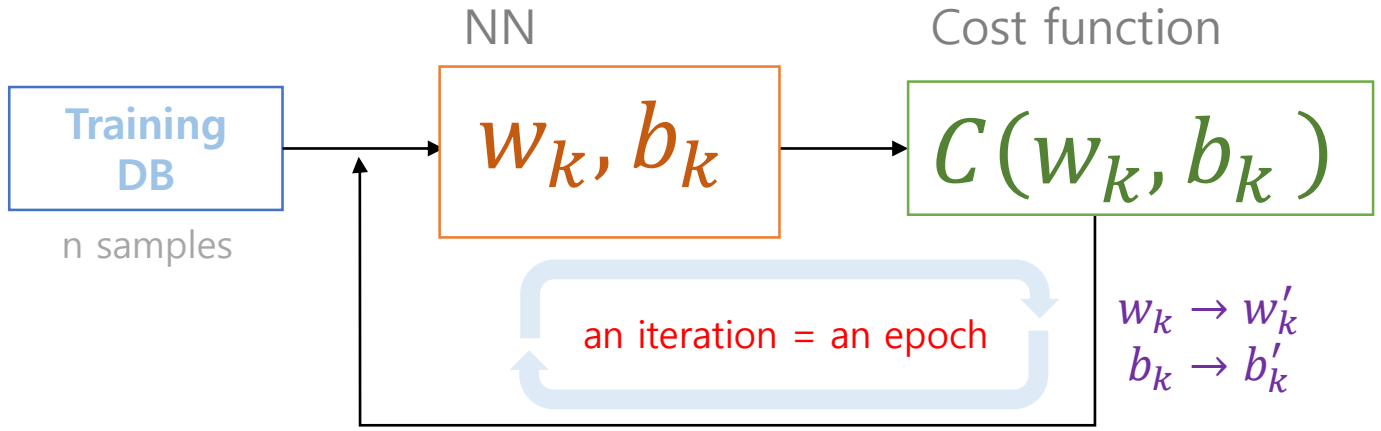


Neural Networks & Deep Learning

http://neuralnetworksanddeeplearning.com/chap2.html

Computing the gradient

Remember



At each layer

Stochastic Gradient Descent

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{X_j}$$

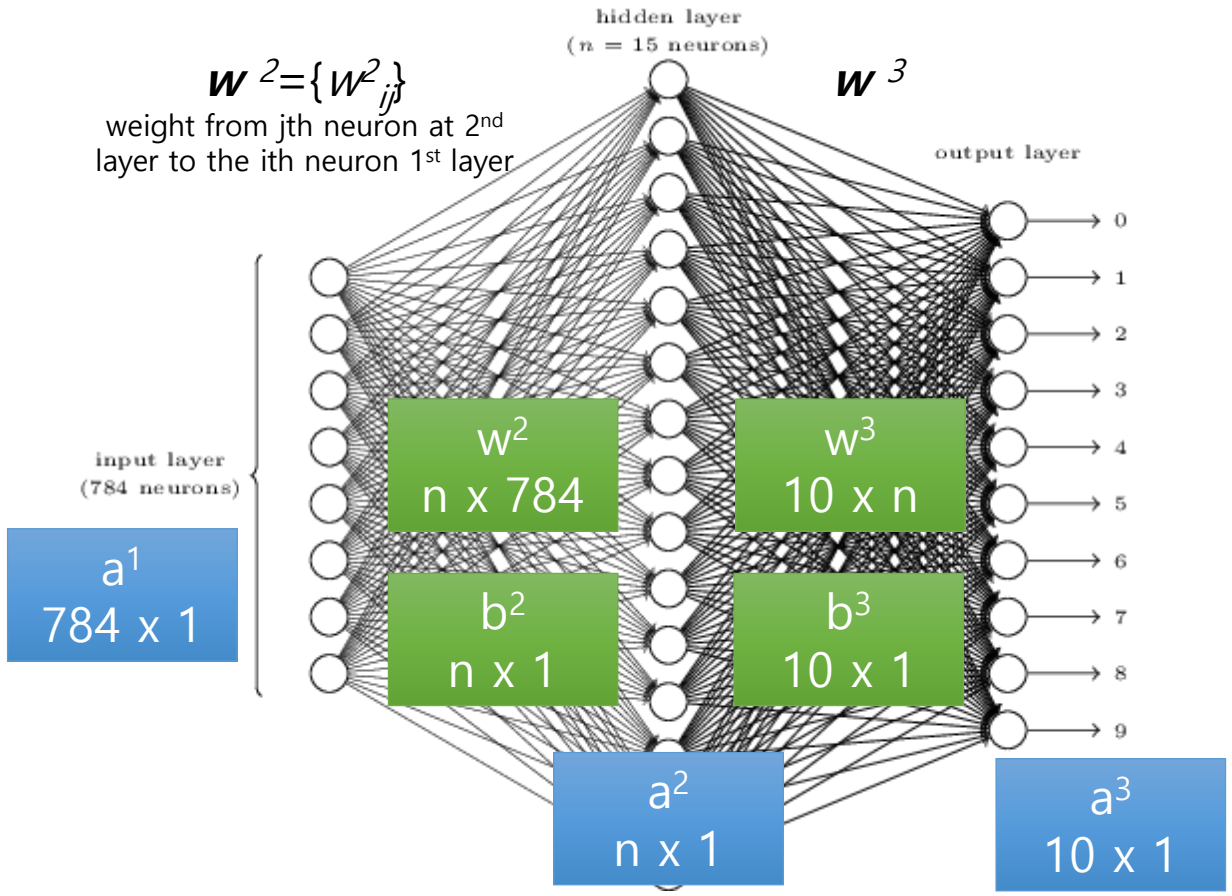
m<n samples

[Problem]
find the partial derivative *∂C/∂w* or *∂C/∂b* of the cost function *C* with respect to any *w* and *b* in the network

Backpropagation

- **Is a fast algorithm for computing gradients!!!**
- **Gives detailed insights how changing the weights and biases changes the overall behavior of the network!!!**

Warm up : a fast matrix-based approach for the output of NN



$w^2 = \{w^2_{jj}\}$
weight from jth neuron at 2nd layer to the ith neuron 1st layer

w^3

w^2 n x 784

w^3 10 x n

b^2 n x 1

b^3 10 x 1

a^1 784 x 1

a^2 n x 1

a^3 10 x 1

Two layers

$a^2 = \sigma(w^2 \cdot a^1 + b^2)$ $\quad$ $a^3 = \sigma(w^3 \cdot a^2 + b^3)$

Matrix form : $a^l = \sigma(w^l a^{l-1} + b^l)$

$z^l \equiv w^l a^{l-1} + b^l$

Element-wise : $a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$

$z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$

- Input to activation function : z
- Output from activation function : a

The two assumptions we need about the cost function

Assumption 1. Total cost of NN over training samples is the sum of cost for each training example

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

$$C = \frac{1}{n} \sum_x C_x \quad \text{where} \quad C_x = \frac{1}{2} \|y - a^L\|^2$$

[Assumption]
Total cost of NN is the average cost for individual training examples

The assumption gives $\partial C/\partial w = \sum \partial C_x/\partial w$, $\partial C/\partial b = \sum \partial C_x/\partial b$

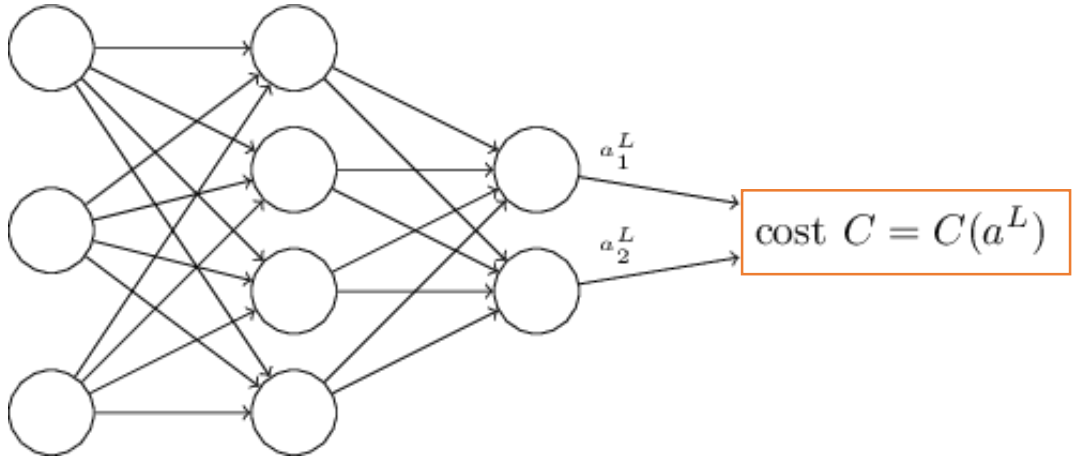
Then, we can focus not on $\partial C/\partial w$ (a whole cost of NN) but on $\partial C_x/\partial w$ (a cost of a single training sample)

Assumption 2. Cost for each training example is a function of a^L

$$C_x = \frac{1}{2} \|y - a^L\|^2$$

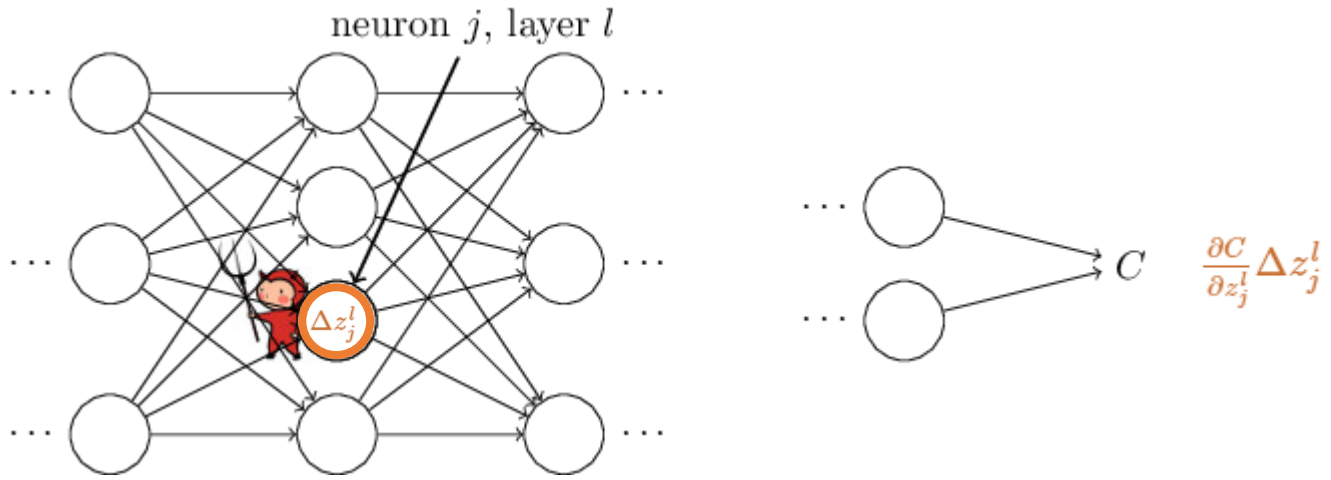
$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2$$

Let's ignore index 'x' because we are focusing cost for a training sample.
For a fixed 'x', desired output 'y' is also fixed.



The four fundamental equations behind backpropagation

Toy example



1. The demon adds a little change Δz_j^l
2. This change causes overall cost C is changed by an amount $\partial C/\partial z_j^l \cdot \Delta z_j^l$
3. Now, this demon is a good demon, and tries to find a Δz_j^l which makes the cost smaller.

$$minimize \ \frac{\partial C}{\partial z_j^l} \cdot \Delta z_j^l$$

System param. $\partial C/\partial z_j^l$		Control param. Δz_j^l
amount	*sign*	*Sign*
Big	Positive	Negative
	Negative	Positive
Close to zero (near to global optimal)		No way to improve

Let's define error of neuron j in layer l by

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

< Error Signal >

The four fundamental equations behind backpropagation

Plan of attack

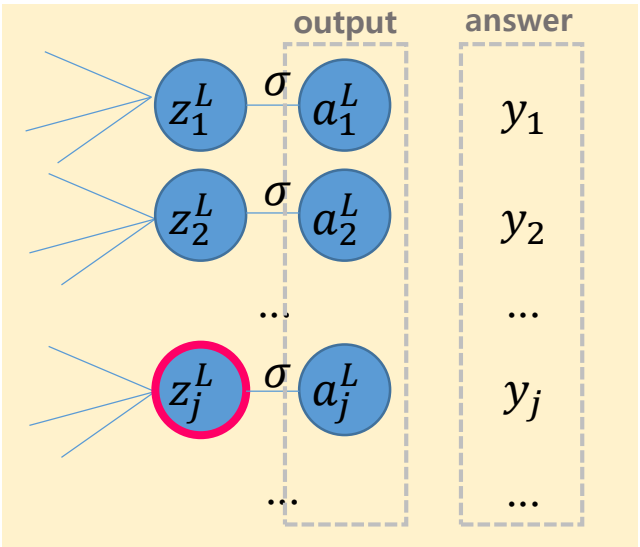
Cost	Error Signal	Error Gradient
C	$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$	$\partial C/\partial w_{jk}^l$ and $\partial C/\partial b_j^l$

Equation 1 for the error in the output layer, δ^L

$$\frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

Chain rule

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$



Quadratic Cost Function : $C = \frac{1}{2}\sum_k (y_k^L - a_k^L)^2$

$$\frac{\partial C}{\partial z_j^L} = \frac{\partial}{\partial z_j^L}\left(\frac{1}{2}\sum_k (y_k - a_k^L)^2\right) = \frac{\partial}{\partial z_j^L}\left(\frac{1}{2}\sum_k \left(y_k - \sigma(z_k^L)\right)^2\right) = \frac{\partial}{\partial z_j^L}\left(\frac{1}{2}\left(y_j - \sigma(z_j^L)\right)^2\right)$$

Only the j[th] term is valid

$$\frac{\partial C}{\partial z_j^L} = -\left(y_j - \sigma(z_j^L)\right)\sigma'(z_j^L) = (a_j^L - y_j)\sigma'(z_j^L)$$

The four fundamental equations behind backpropagation

Equation 1 for the error in the output layer, δ^L

Matrix form : General cost function

$$\delta^L = [\delta_1^L, \delta_2^L, \cdots, \delta_j^L, \cdots]^T = \left[\frac{\partial C}{\partial a_1^L} \cdot \sigma'(z_1^L), \frac{\partial C}{\partial a_2^L} \cdot \sigma'(z_2^L), \cdots \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L), \cdots\right]^T$$

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

Hadamard Product

- denoted by '$\odot$'
- element-wise product of two vectors
- also called Schur product

$$\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1*3\\2*4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

Matrix form : Quadratic cost function

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = (a_j^L - y_j)\sigma'(z_j^L)$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

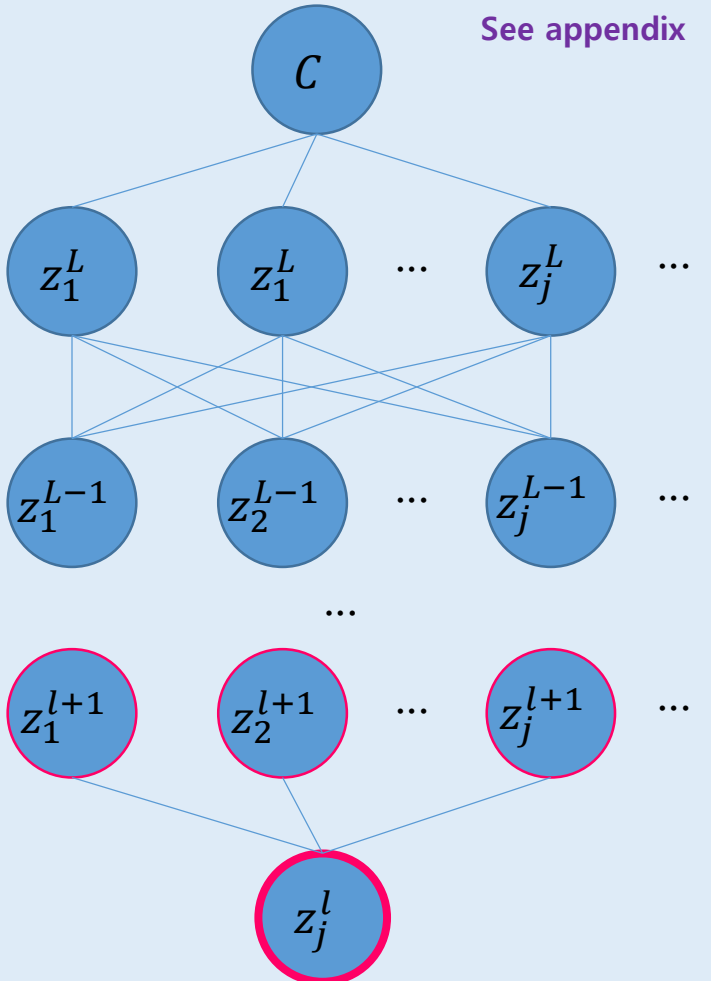
The four fundamental equations behind backpropagation

Equation 2

for the error in the lth layer in terms of the error in the next layer (l+1), $\delta^l \leftrightarrow \delta^{l+1}$

Chain Rule with Dependency Graph

See appendix



$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \underbrace{(factors\ from\ all\ paths\ from\ C\ to\ z_1^{l+1})}_{\partial C/\partial z_1^{l+1}} \cdot \frac{\partial z_1^{l+1}}{\partial z_j^l} + \underbrace{(factors\ from\ all\ paths\ from\ C\ to\ z_2^{l+1})}_{\partial C/\partial z_2^{l+1}} \cdot \frac{\partial z_2^{l+1}}{\partial z_j^l} + \cdots$$

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l)$$

$$z_k^{l+1} = \sum w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

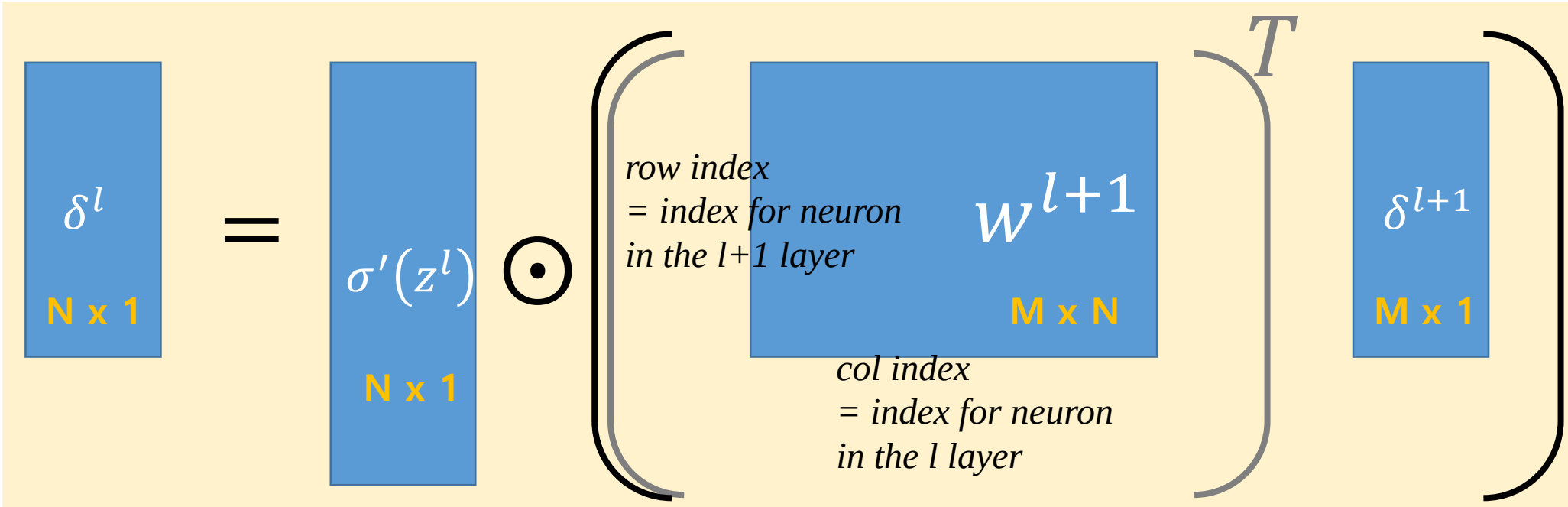
The four fundamental equations behind backpropagation

Equation 2

for the error in the lth layer in terms of the error in the next layer (l+1), $\delta^l \leftrightarrow \delta^{l+1}$

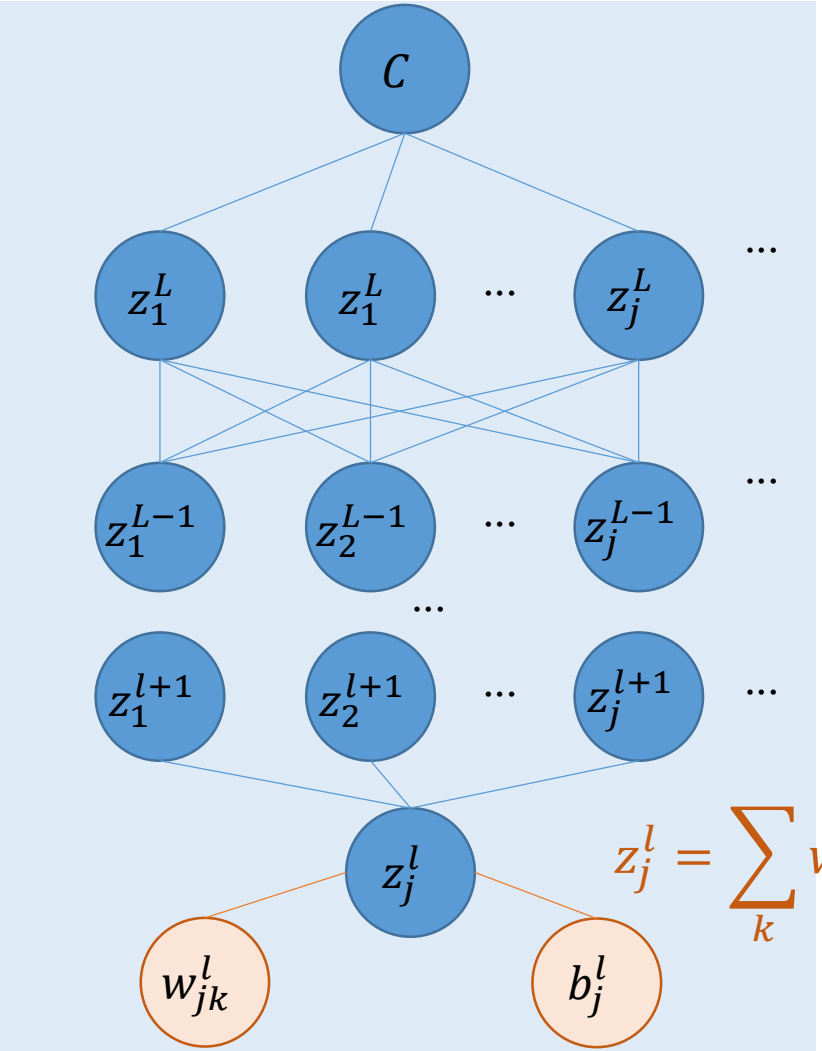
$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) = \sigma'(z_j^l) \sum_k \delta_k^{l+1} w_{kj}^{l+1}$$

$$\delta^l = \sigma'(z^l) \odot \left((w^{l+1})^T \delta^{l+1} \right)$$ Matrix Form



The four fundamental equations behind backpropagation

Equation 3,4 for the rate of change of the cost with respect to any bias in the network
for the rate of change of the cost with respect to any weight in the network



$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} = \delta_j^l a_k^{l-1}$$

$$\nabla_{b^l} C = \delta^l$$ Matrix Form

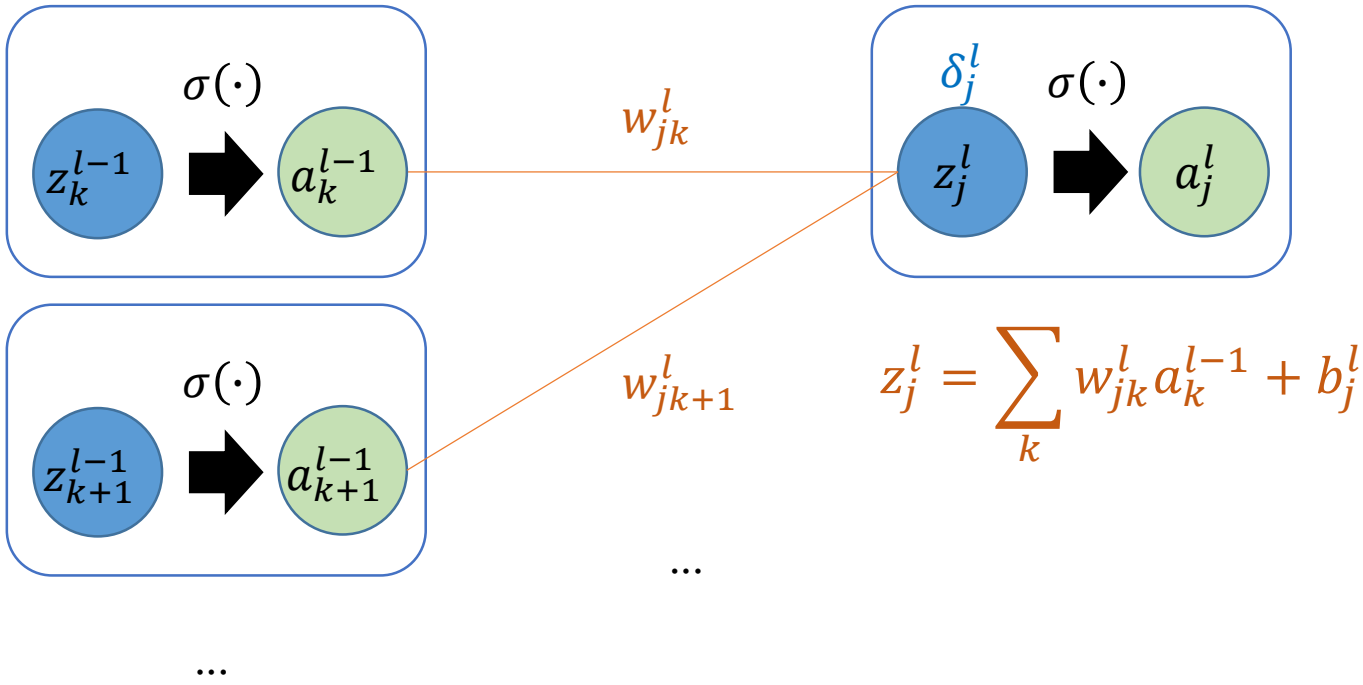
$$\nabla_{W^l} C = \delta^l \left(a^{l-1}\right)^T$$ Matrix Form

The four fundamental equations behind backpropagation

Equation 4

for the rate of change of the cost with respect to any weight in the network

$$\frac{\partial C}{\partial w_{jk}^{l}} = \delta_j^l a_k^{l-1}$$



$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

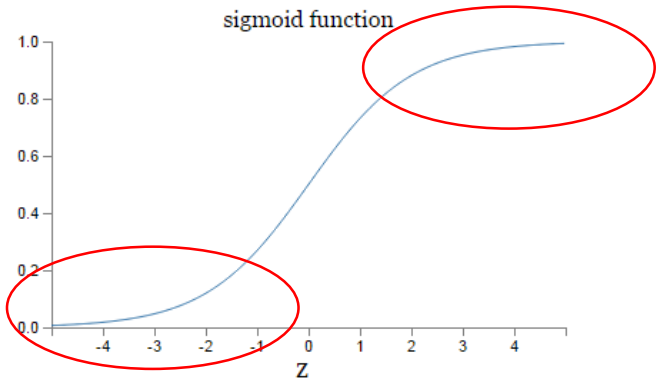
[Insight 1] Low-activation neurons

Any change in the weight on a low-activation neuron (small 'a') causes small change in 'C' the cost of NN. Vice versa, the weight learns slowly. (gradient of 'C' does not affect the weight)

[Insight 2] Saturated neurons

At the output layer, the weight on neurons with output of almost 0 or 1 learn slowly.
Similar remarks hold on the biases of output neurons.

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



The four fundamental equations behind backpropagation

Summary

1. Error at the output layer

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$ Remember 'C' is function of 'a'

2. Error relationship between two adjacent layers

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

3. Gradient of C in terms of bias

$$\nabla_{b^l} C = \delta^l$$

4. Gradient of C in terms of weight

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T$$

Error in any layer can be obtained by back-propagated manners

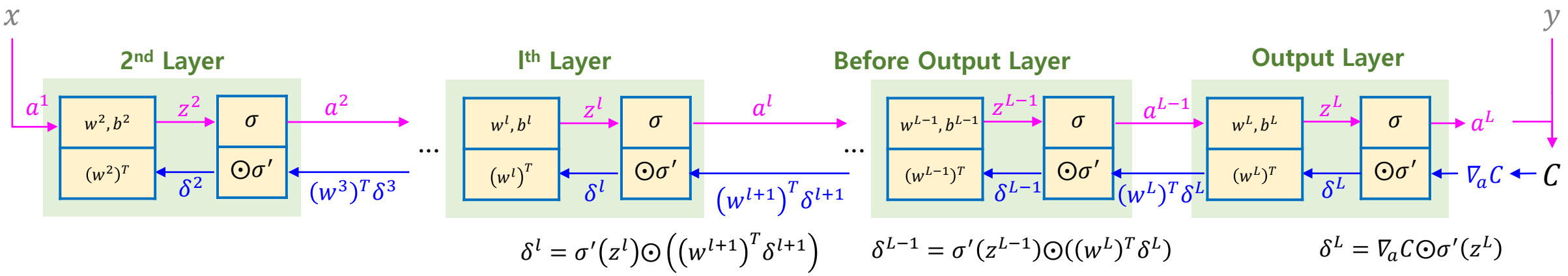
Gradient of C in terms of bias and weight for any layer can be obtained

The four fundamental equations behind backpropagation

Summary

1. Error at the output layer	2. Error relationship between two adjacent layers	3. Gradient of C in terms of bias	4. Gradient of C in terms of weight
$\delta^L = \nabla_a C \odot \sigma'(z^L)$	$\delta^l = \sigma'(z^l) \odot \left((w^{l+1})^T \delta^{l+1}\right)$	$\nabla_{b^l} C = \delta^l$	$\nabla_{w^l} C = \delta^l (a^{l-1})^T$

For a training sample (x, y)



The backpropagation algorithm

For a training sample

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

The backpropagation algorithm

For all training samples

1. **Input a set of training examples**

2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:

   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.

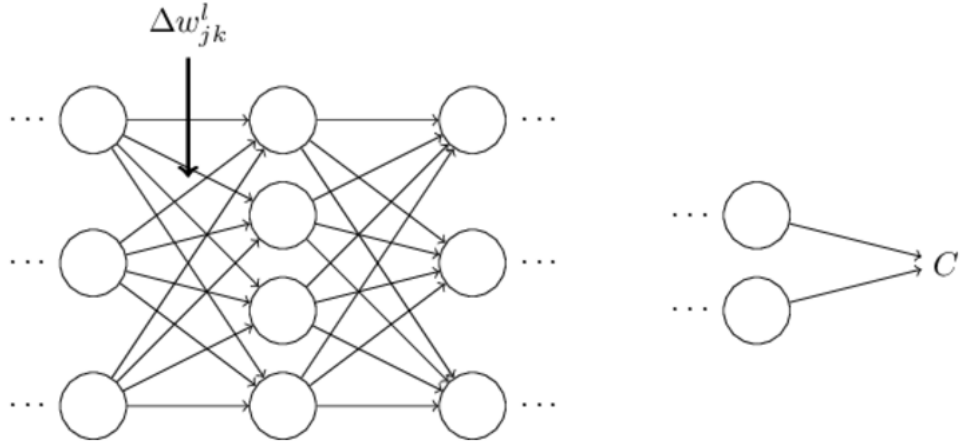
   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.

   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

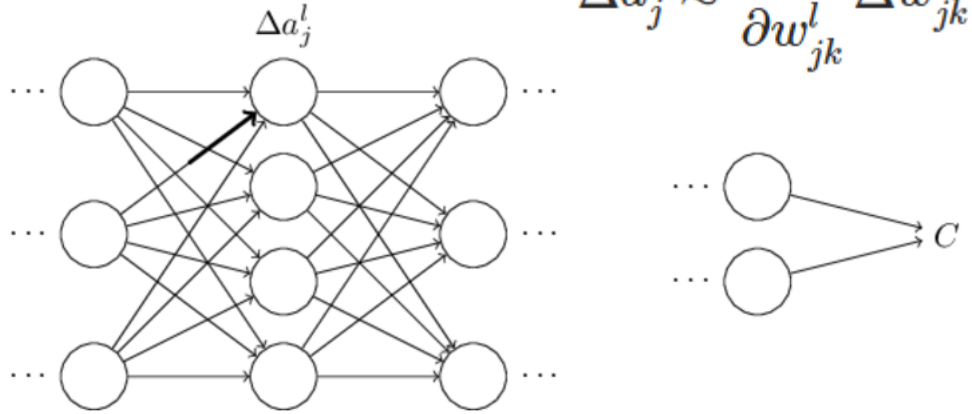
Backpropagation : the big picture

T = t0

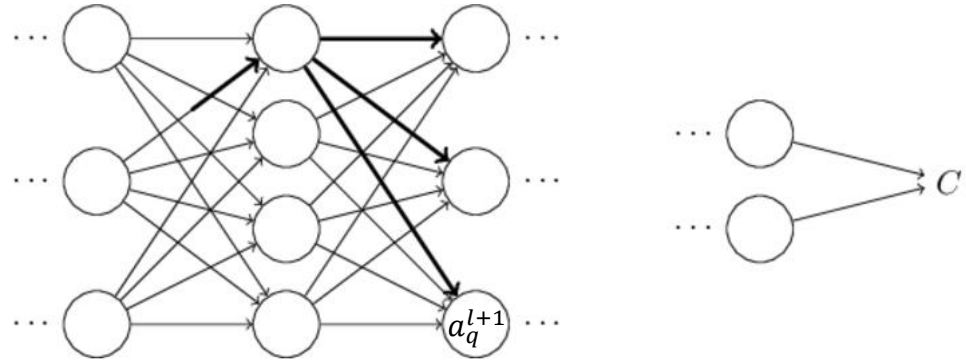


T = t1

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

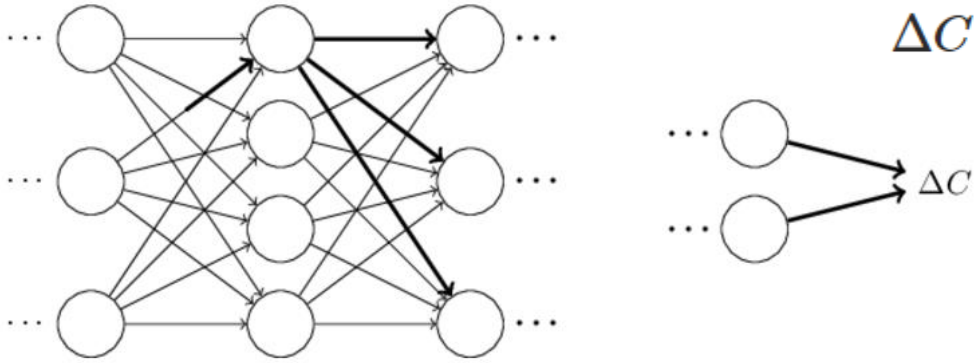


T = t2



T = t3

$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk}$$



$$\Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \Delta a^l_j \qquad \Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

For a path,

$$\Delta C \approx \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

Backpropagation : the big picture

For a path,

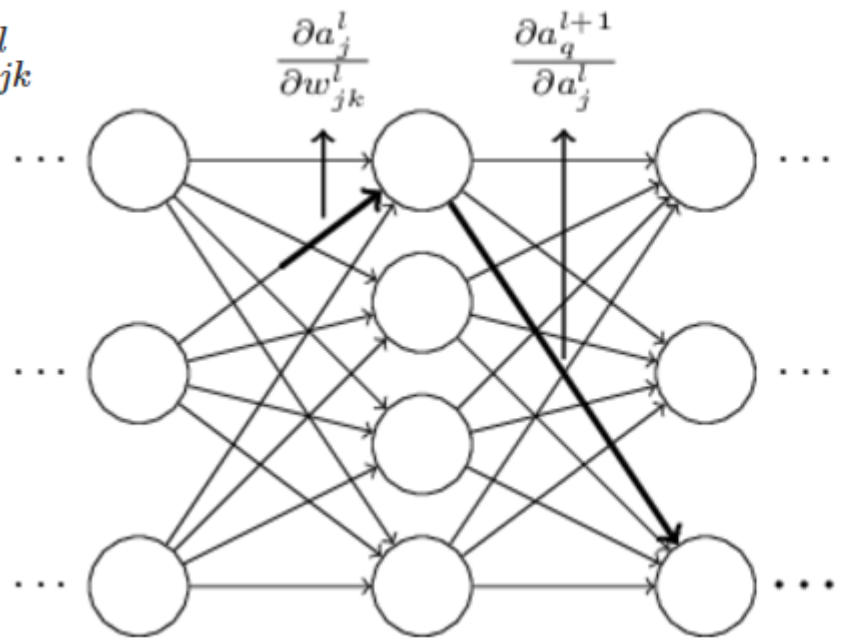
$$\Delta C \approx \frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$

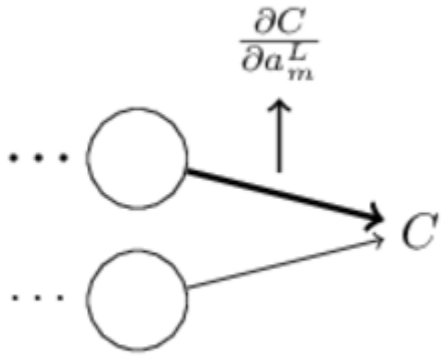
For all paths,

$$\Delta C \approx \sum_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$





Implementing our network to classify digits

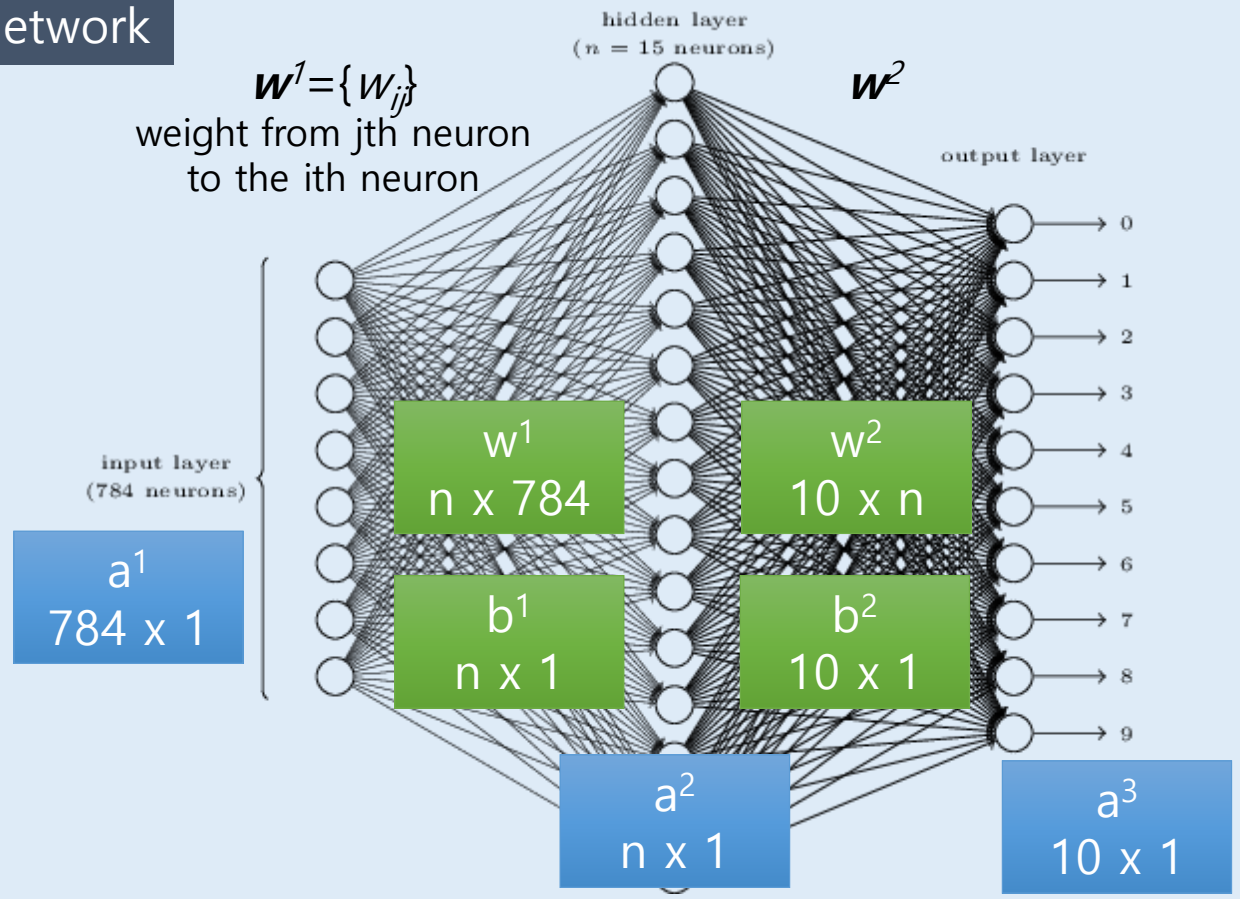
MNIST dataset

Modified subset of **NIST**(National Institute of Standards and Technology)
-MNIST are acquired from 250 people (employees + high school students)
-the images are grayscale and 28x28 pixels.

Our network

$w^1=\{w_{ij}\}$
weight from jth neuron to the ith neuron

w^2



a^1 784 x 1

w^1 n x 784

b^1 n x 1

w^2 10 x n

b^2 10 x 1

a^2 n x 1

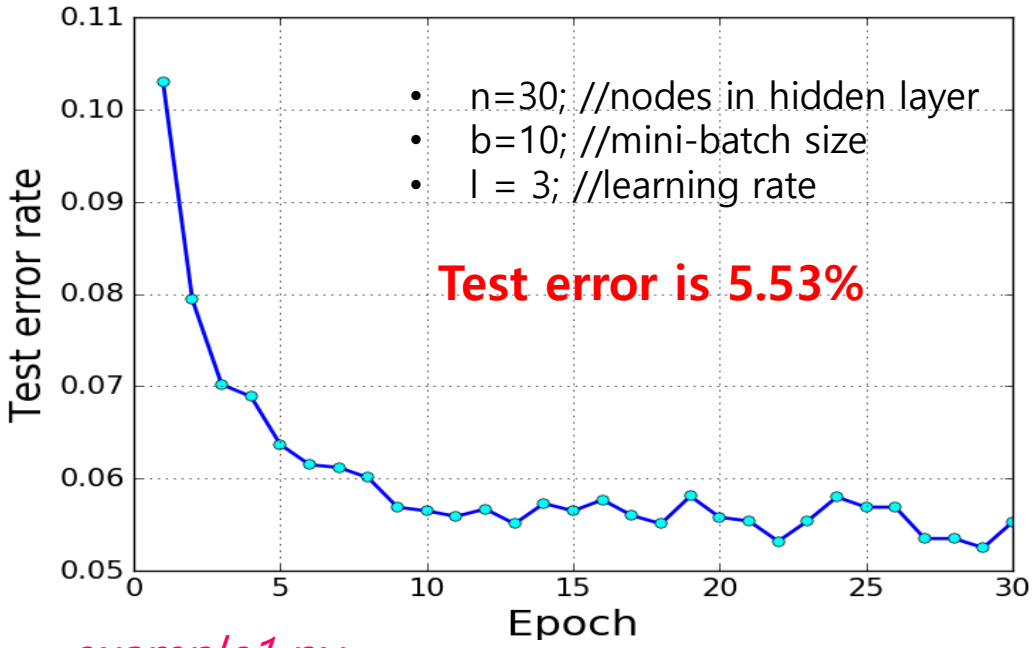
a^3 10 x 1

Two layers

$a^2 = \sigma(w^1 \cdot a^1 + b^1)$ $a^3 = \sigma(w^2 \cdot a^2 + b^2)$

Simulation Results

For all training samples



example1.py

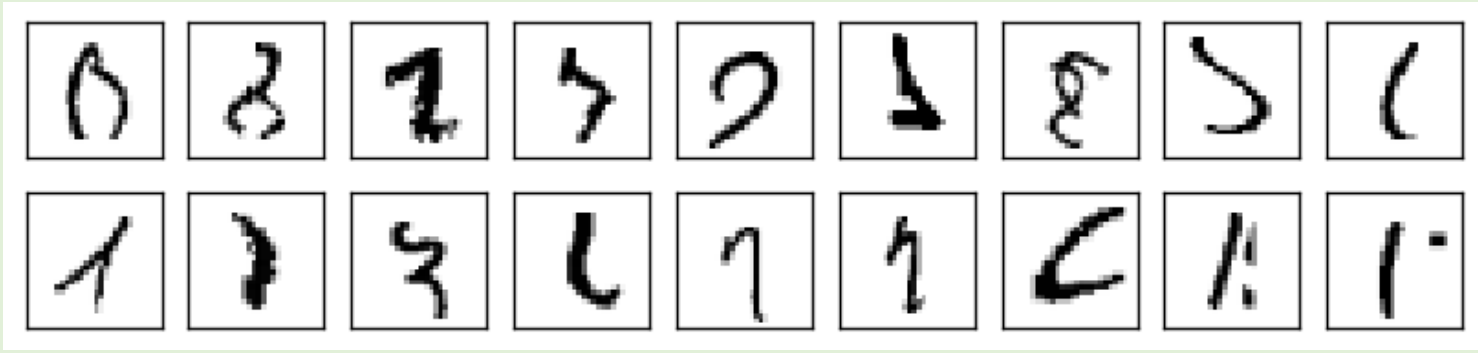
[Benchmark]

1. Random guess : Test error of **90%**
2. Handcraft feature (a simple algorithm with comparing average darkness) : Test error of **77.75%**
3. World Record...

World Record : Test error is **0.23%** → 9977 / 10000 !!!

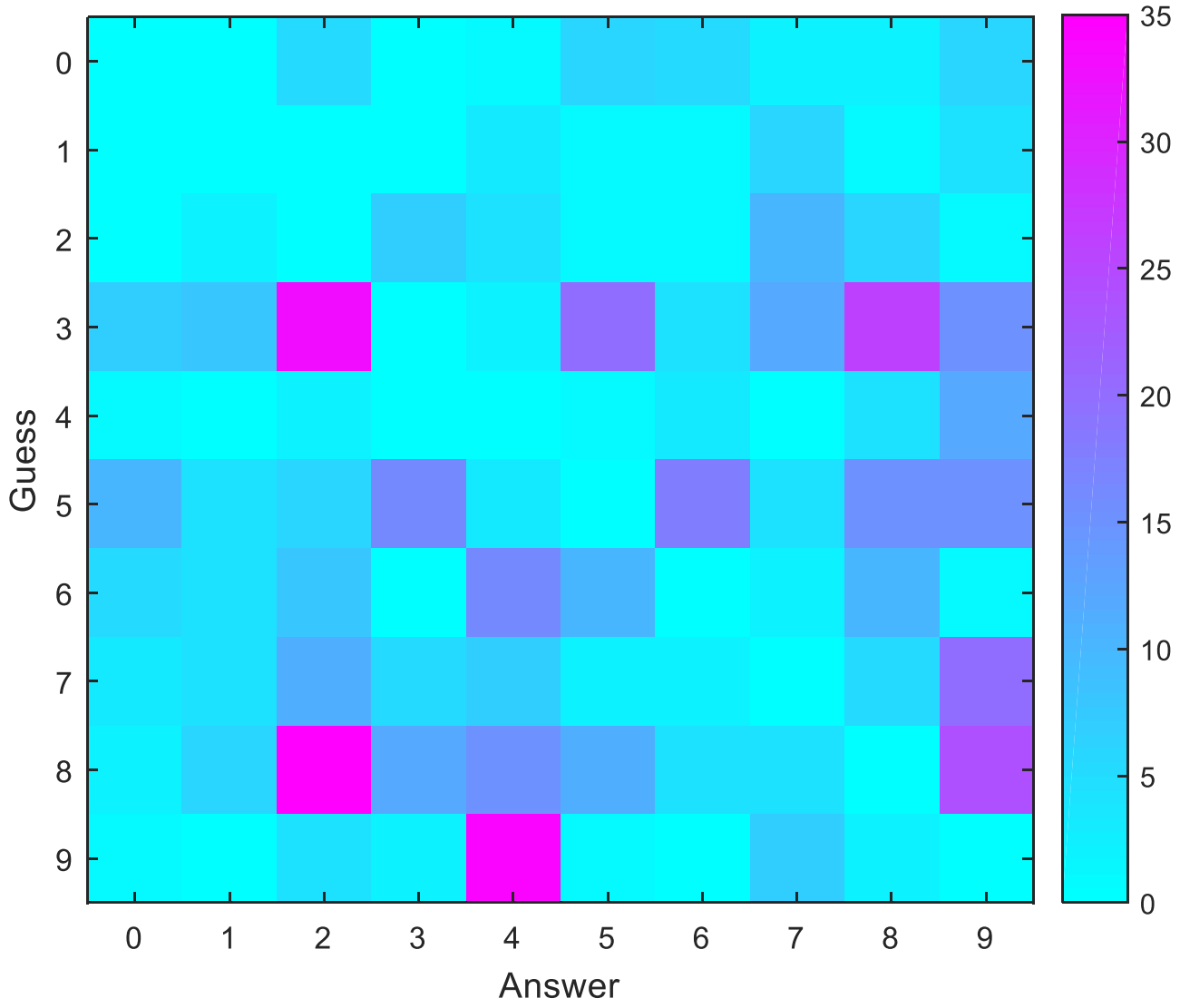
http://yann.lecun.com/exdb/mnist/

Guess what???
There are in test DB!



Simulation Results

Confusion Matrix

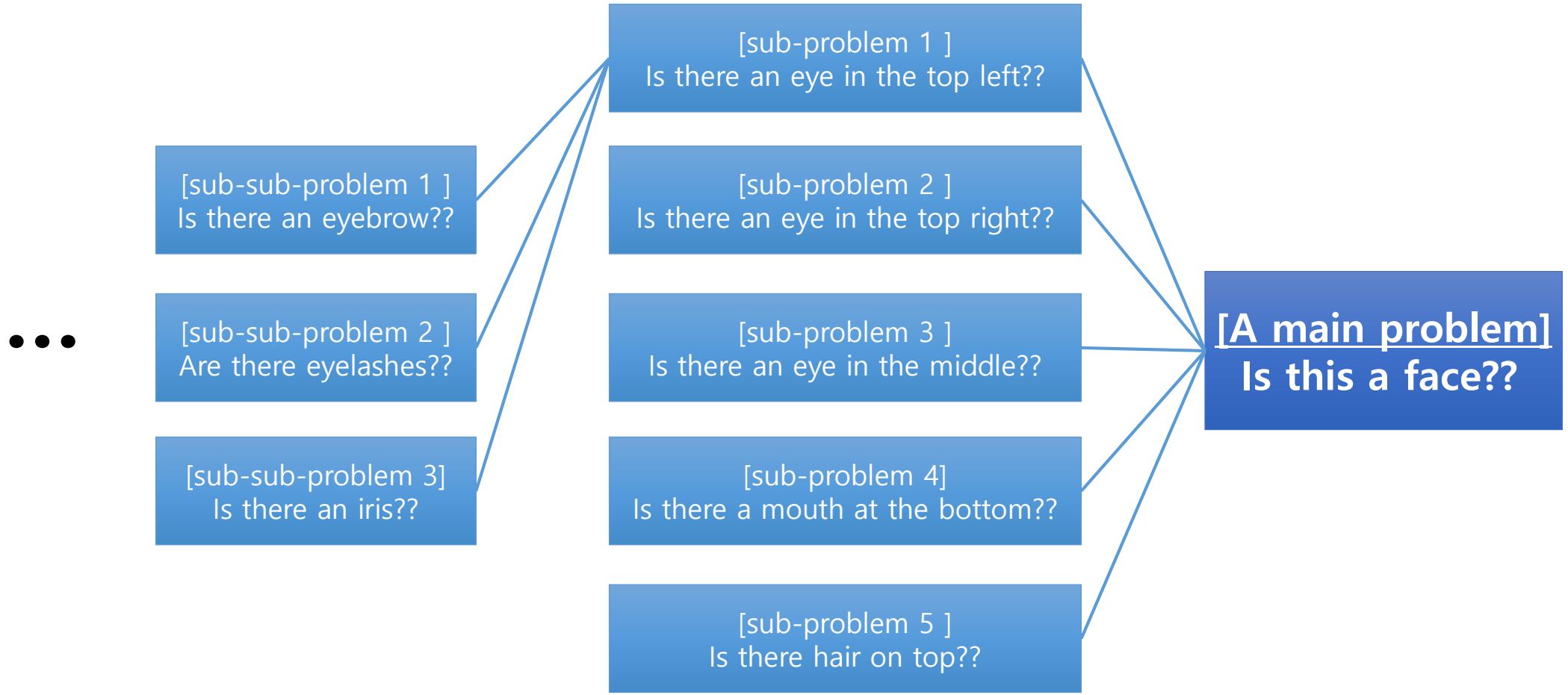


Guess	Answer
3	2

Guess	Answer
9	4

Toward deep learning

Let's move on another problem of detecting face.



- Each hidden layer breaks down a problem into sub-problems.
- Deep-NN breaks down a very complicated question into very simple questions answerable at the level of single pixels.

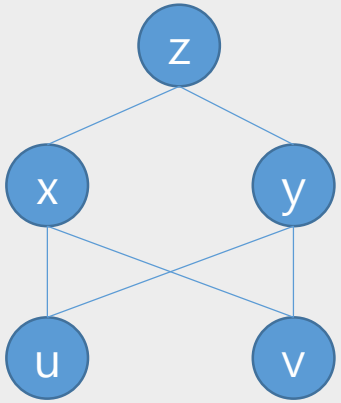
via Dependency Graph

Question

$$z = x^2 + y^2$$
$$x = u^2 - v^2$$
$$y = u \cdot v$$

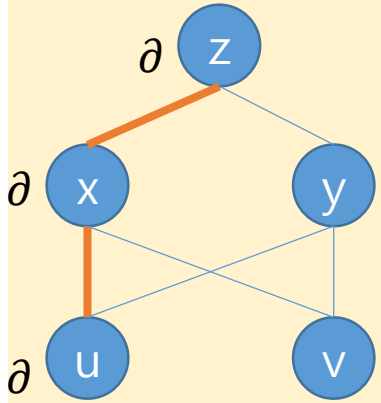
$$\frac{\partial z}{\partial u} = ???$$

Dependency Graph



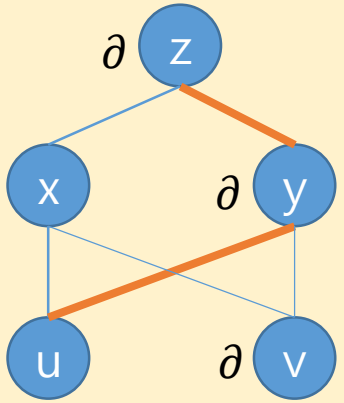
Partial derivative is sum of all possible paths

Path 1



$$\frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial u}$$

Path 2



$$\frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial u}$$

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial u}$$