Feed forward Neural Networks

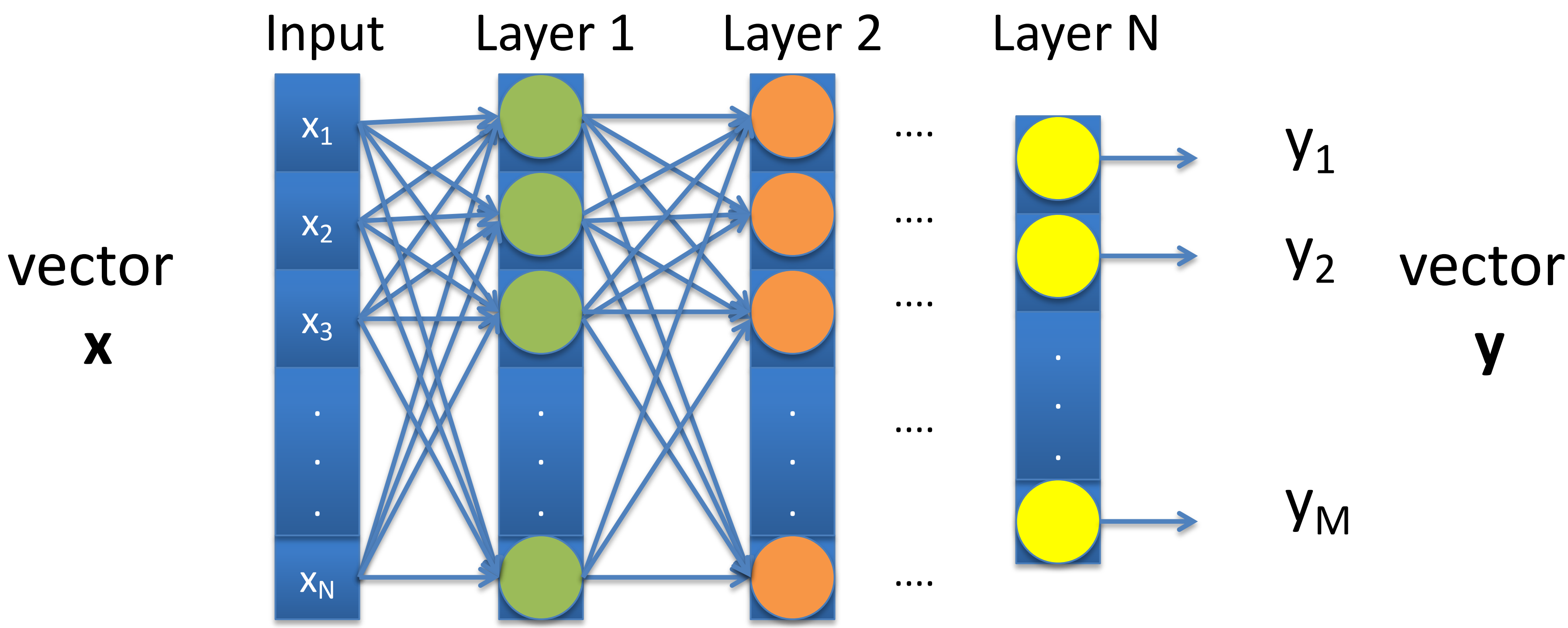
Thang Vu

27.11.2025

Overview

- Review
  - Backpropagation
  - Cross-entropy loss function
- Activation function
  - Sigmoid
  - Tanh
  - ReLU
- Setting a Network
- Learning: practical usages

Computation of the final output



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

Empirical risk minimization

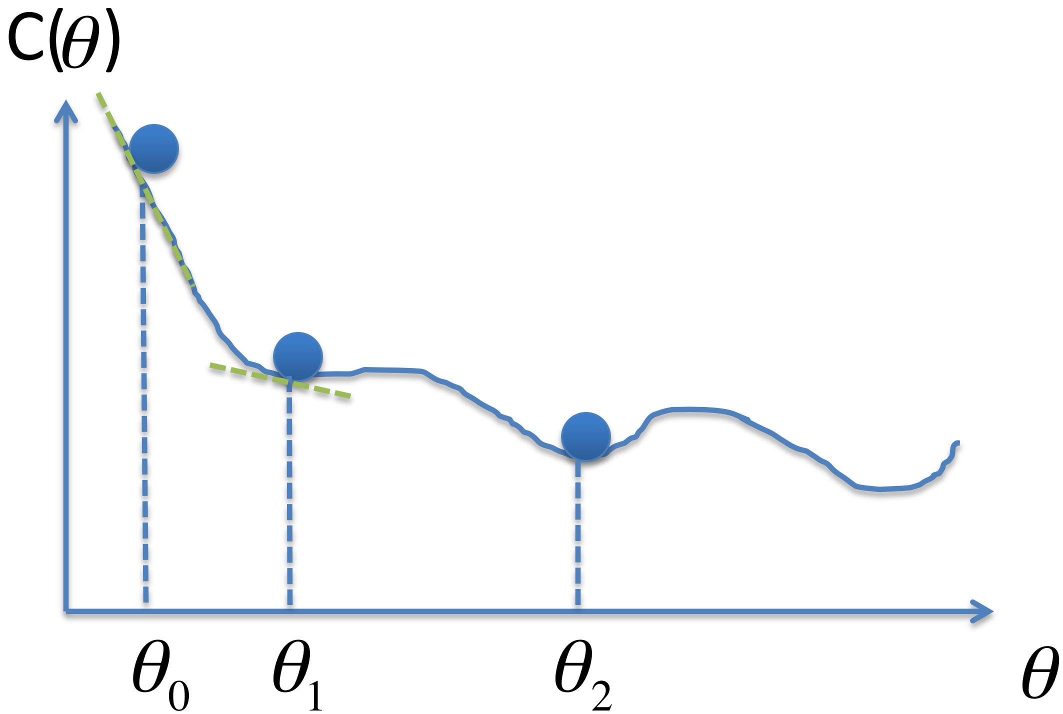
- Given a finite set of training data
- Empirical risk = average loss on this training data

$$C(\theta) = \frac{1}{|D|} \sum_{(x,y)} c(f(x), y)$$

$$= \frac{1}{|D|} \sum_{(x,y)} c(\theta)$$

Gradient descent

- First consider that θ has only one variable



- Randomly start at θ_0
- Compute $dC(\theta_0)/d\theta$

$$\theta_1 \leftarrow \theta_0 - \eta dC(\theta_0)/d\theta$$

- Compute $dC(\theta_1)/d\theta$

$$\theta_2 \leftarrow \theta_1 - \eta dC(\theta_1)/d\theta$$

- ……

Gradient descent for Neural Network

- We will do the same thing as presented before
- Starting parameters θ_0

$$\longrightarrow \theta_1 \longrightarrow \theta_2 \longrightarrow \theta_3 \; \ldots\ldots\ldots\ldots$$

- However,

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

- i.e. millions of parameters ☹
- On the other hand, a lot of training data too!!

Backpropagation

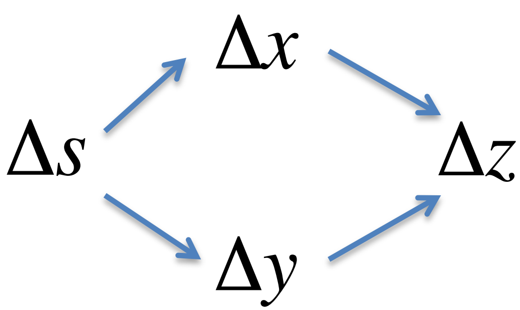
Based on the chain rules:

- Case 1: $y = g(x)$

  $z = h(y)$ $\qquad \dfrac{dz}{dx} = \dfrac{dz}{dy}\dfrac{dy}{dx}$

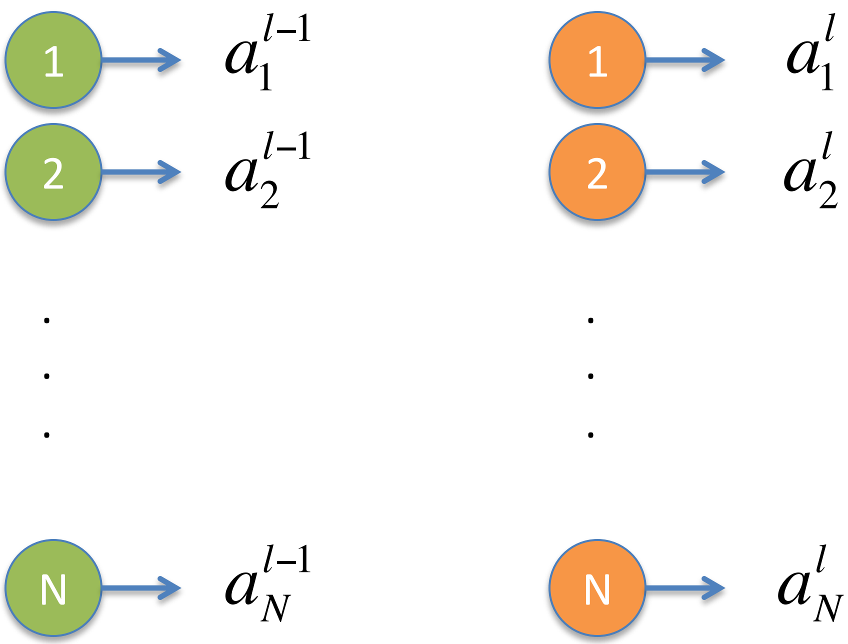
  $\Delta x \rightarrow \Delta y \rightarrow \Delta z$

- Case 2: $x = g(s) \quad y = h(s) \quad z = k(x, y)$



$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

Notation



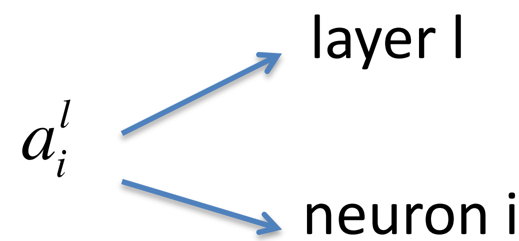
Layer l-1

N_{l-1} nodes

Layer l

N_l nodes

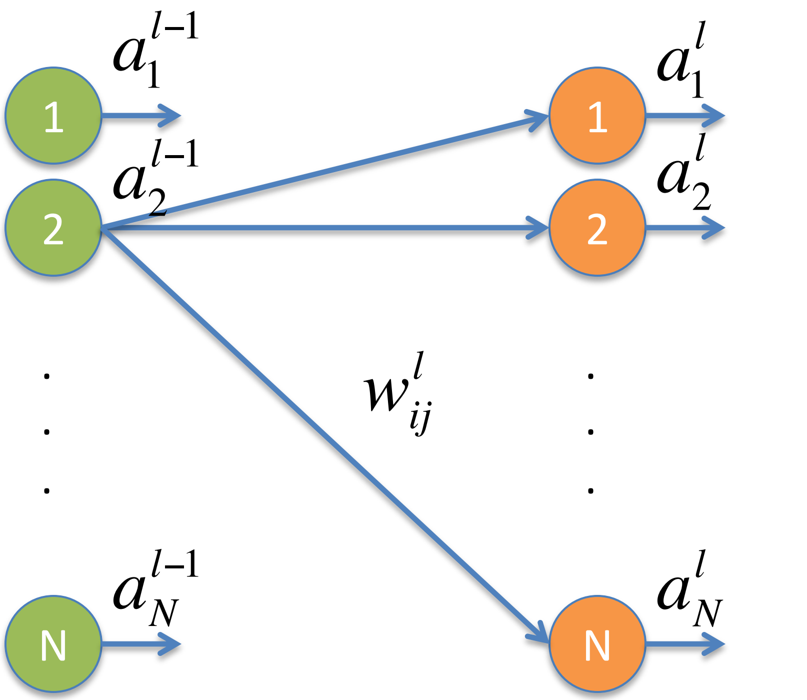
- Output of a neuron:



- Output of one layer:

a^l is a vector

Notation



Layer l-1

N_{l-1} nodes

Layer l

N_l nodes

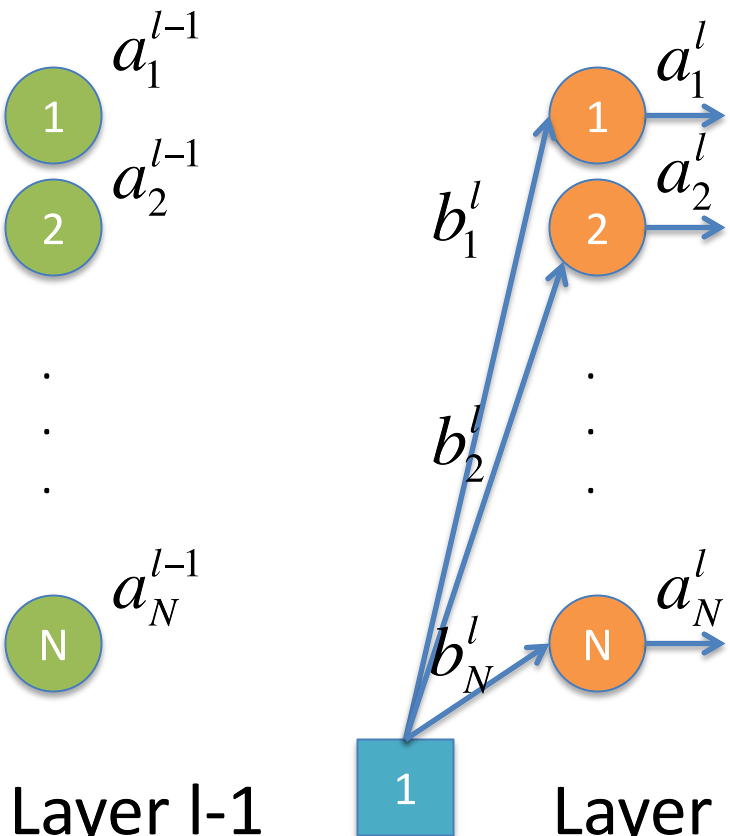
- Weights:

w_{ij}^l → layer l – 1 to layer l

w_{ij}^l → from neuron j (layer l -1) to neuron i (layer l)

$$W^l = \underbrace{\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & & \end{bmatrix}}_{N_{l-1} \text{ columns},\; N_l \text{ rows}}$$

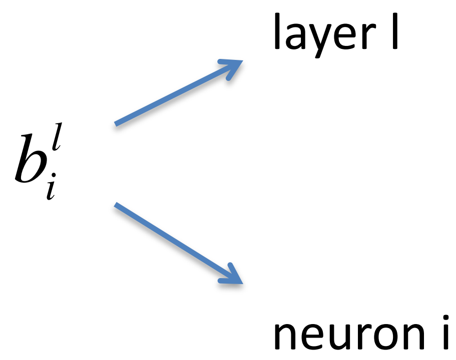
Notation



Layer l-1
N_{l-1} nodes

Layer l
N_l nodes

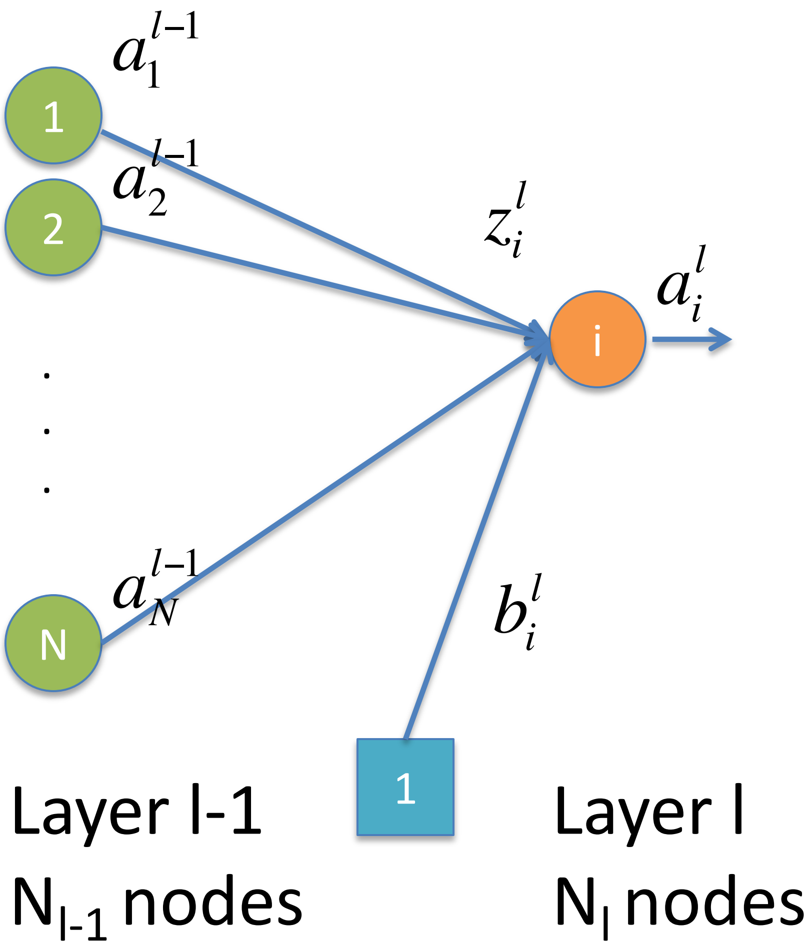
- Biases:



$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \end{bmatrix}$$

Bias for all the neurons in layer l

Notation



z_i^l : input of the activation function for neuron i at layer l

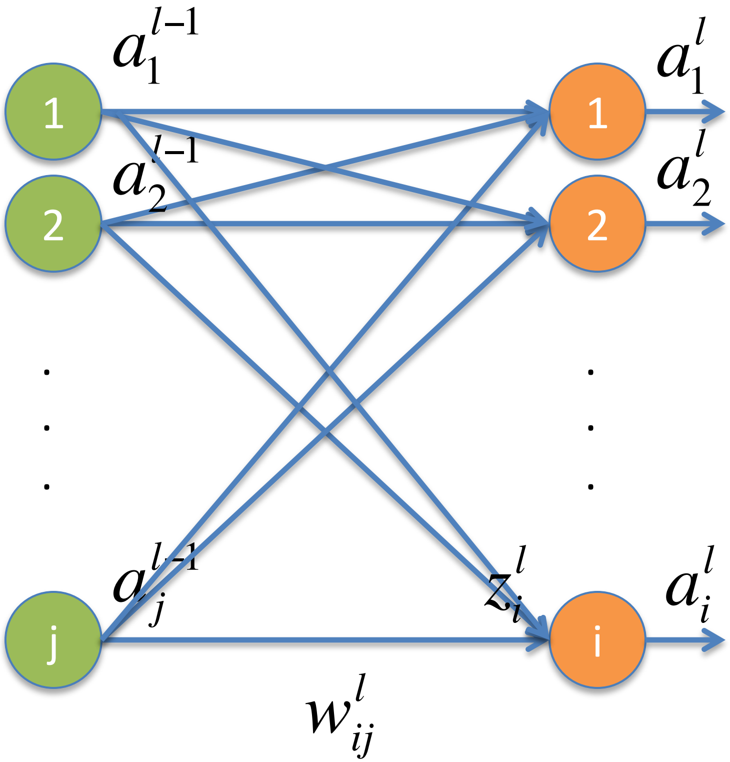
z^l : input of the activation function of all the neurons in layer l

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

or in another form

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$

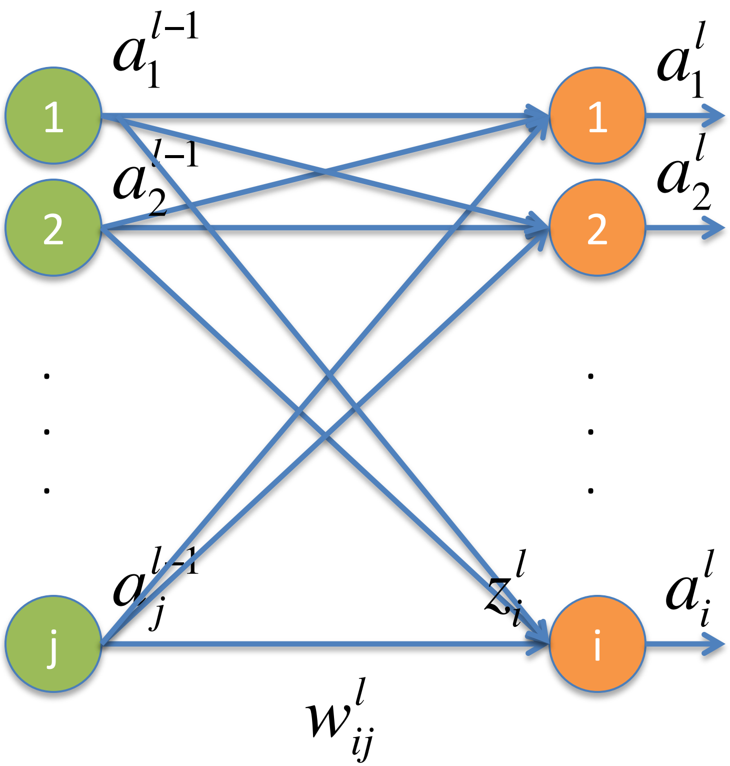
Compute $\frac{\partial C}{\partial w_{ij}^l}$



$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \ldots \rightarrow \Delta C$$

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

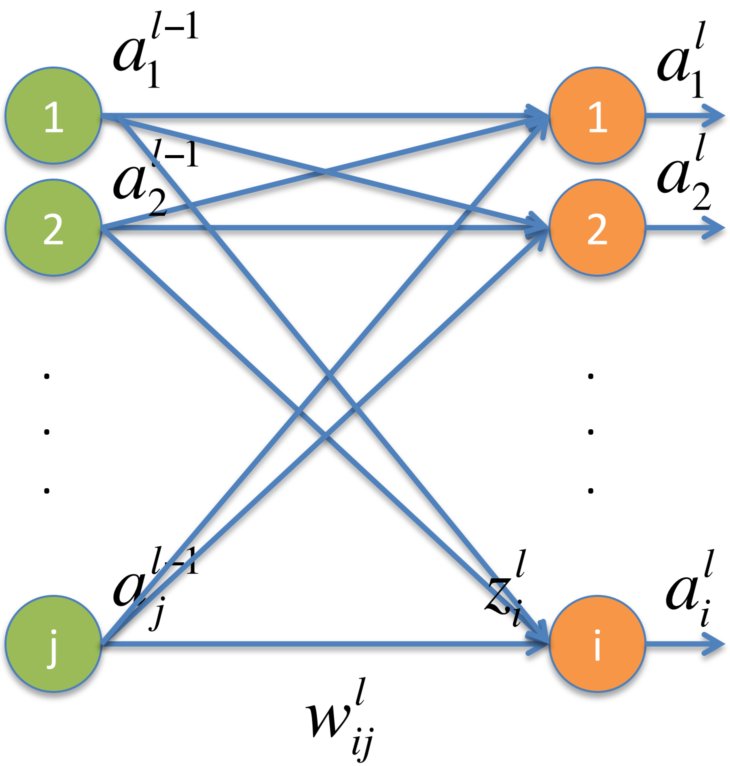
Compute $\frac{\partial C}{\partial w_{ij}^l}$



$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \ldots \rightarrow \Delta C$$

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

Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - First term



- If l > 1:

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

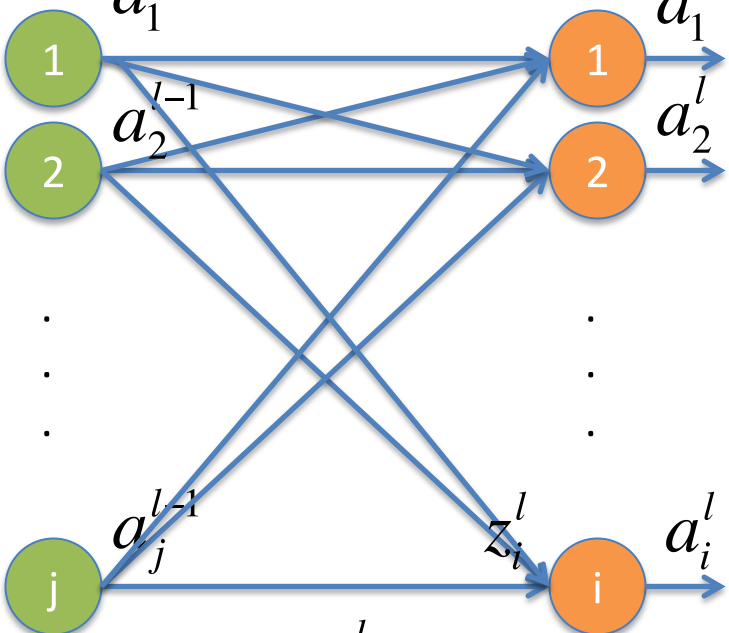
$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

- If l = 1:

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

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

Compute $\frac{\partial C}{\partial w_{ij}^l}$

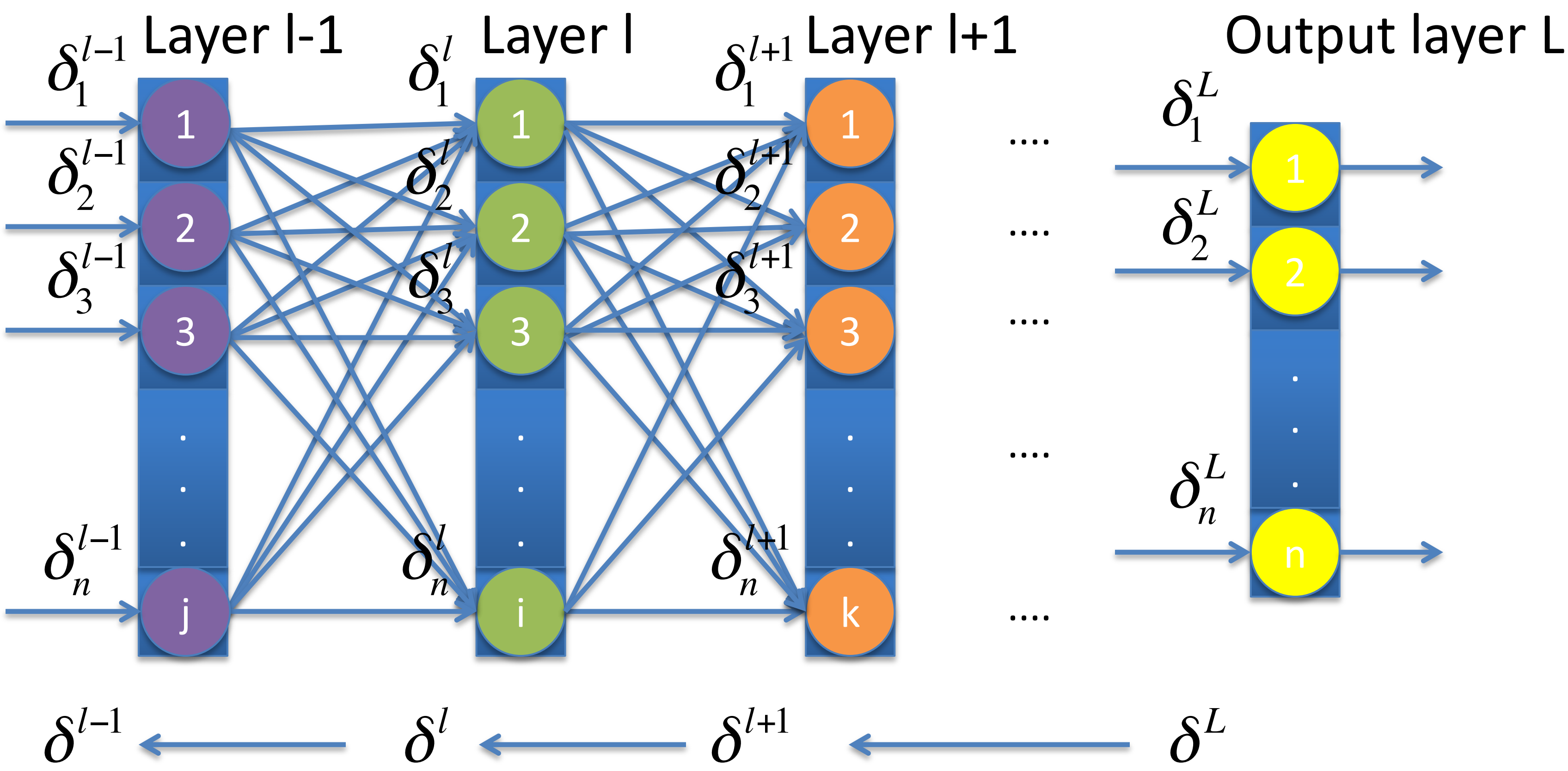


$$a_1^{l-1} \quad a_2^{l-1} \quad a_j^{l-1} \quad w_{ij}^l \quad z_i^l \quad a_1^l \quad a_2^l \quad a_i^l$$

$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \ldots \rightarrow \Delta C$$

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

Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - Second term



Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - Second term

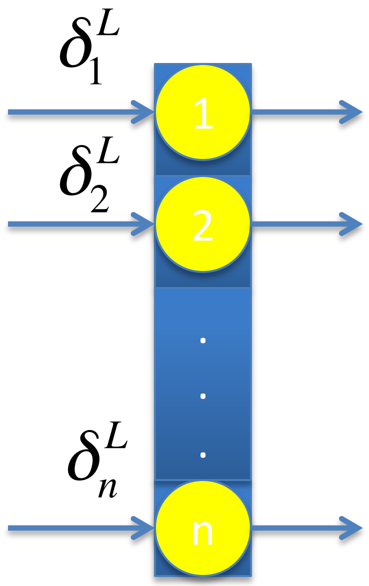
$$\Delta z_n^L \rightarrow \Delta a_n^L = \Delta y_n \rightarrow \Delta C$$

$$\delta_n^L = \frac{\partial C}{\partial z_n^L} = \frac{\partial a_n^L}{\partial z_n^L} \frac{\partial C}{\partial a_n^L}$$

$\sigma'(z_n^L)$

Depending on
The loss function

Output layer L

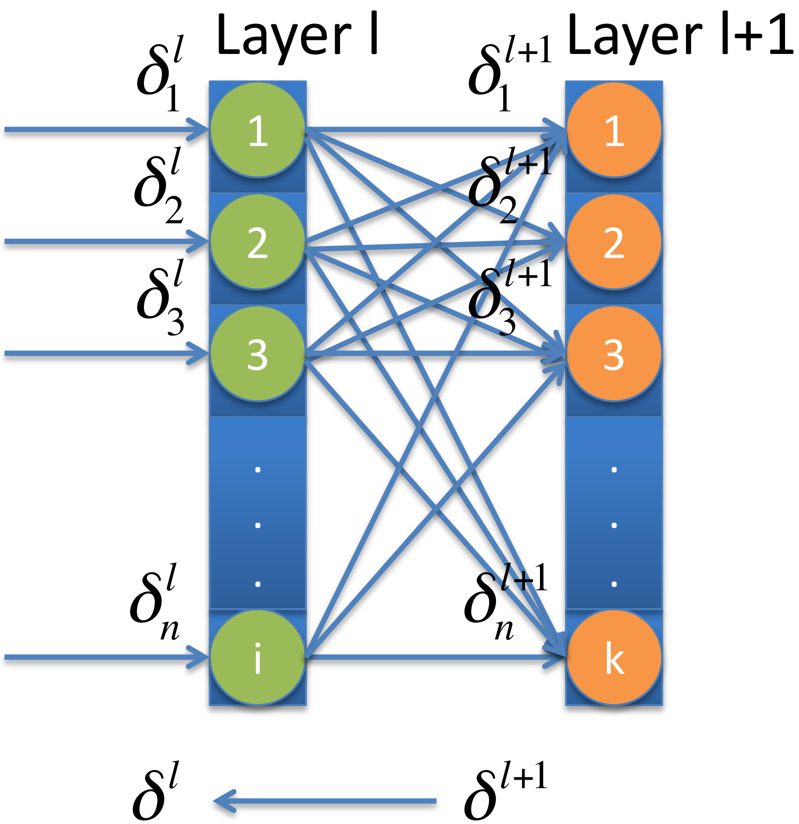


δ^L

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

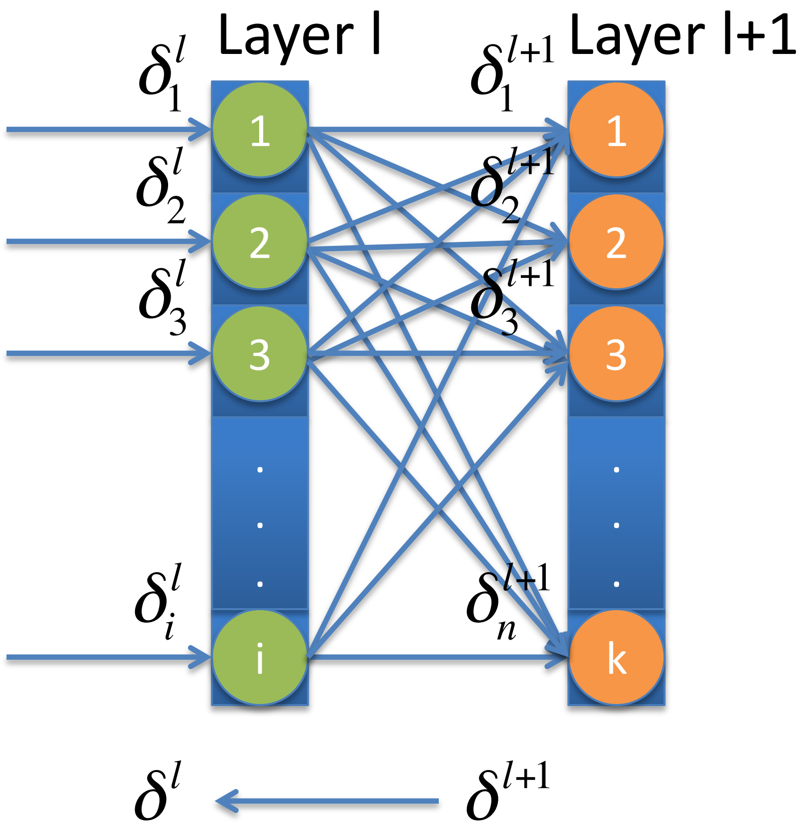
Elementwise multiplication

Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - Second term



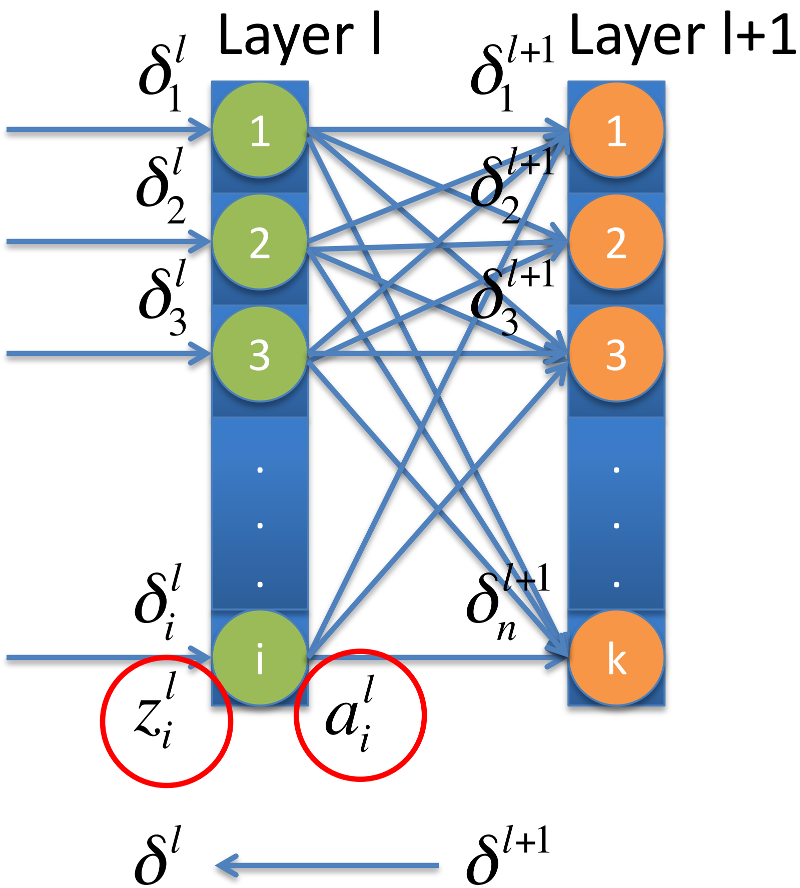
$$\delta^l = \sigma'(z^l)(W^{l+1})^T \delta^{l+1}$$

Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - Second term



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

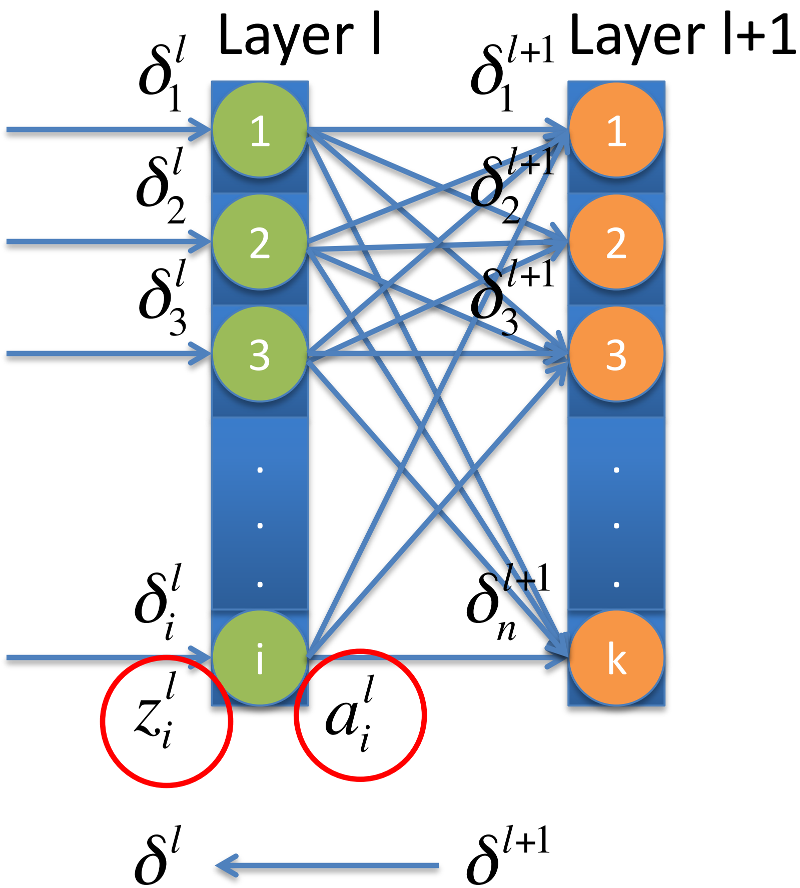
Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



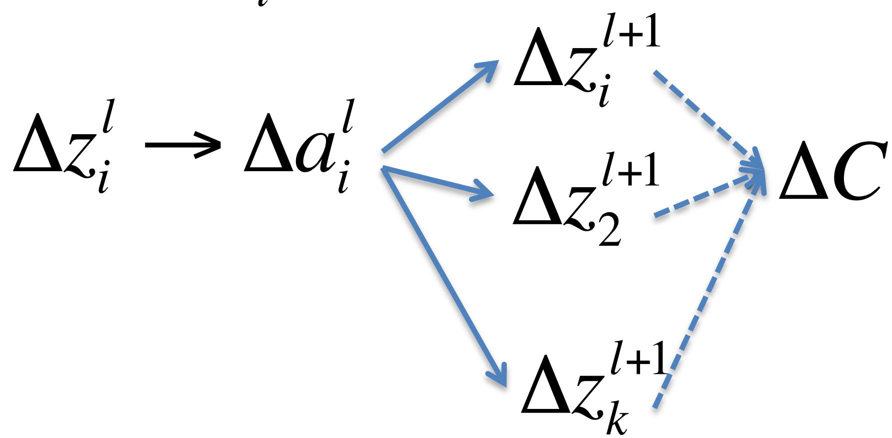
$$\delta_i^l = \frac{\partial C}{\partial z_i^l}$$

$$\Delta z_i^l \rightarrow \Delta a_i^l$$

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



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



Chain Rules

- Case 1:

$$y = g(x)$$

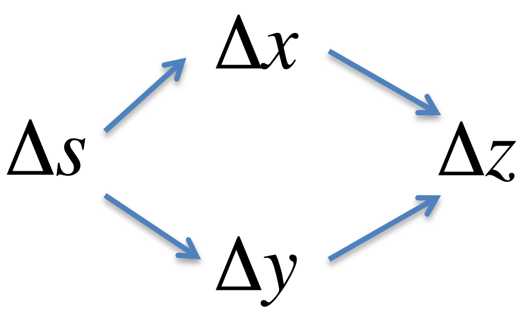
$$z = h(y)$$

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

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

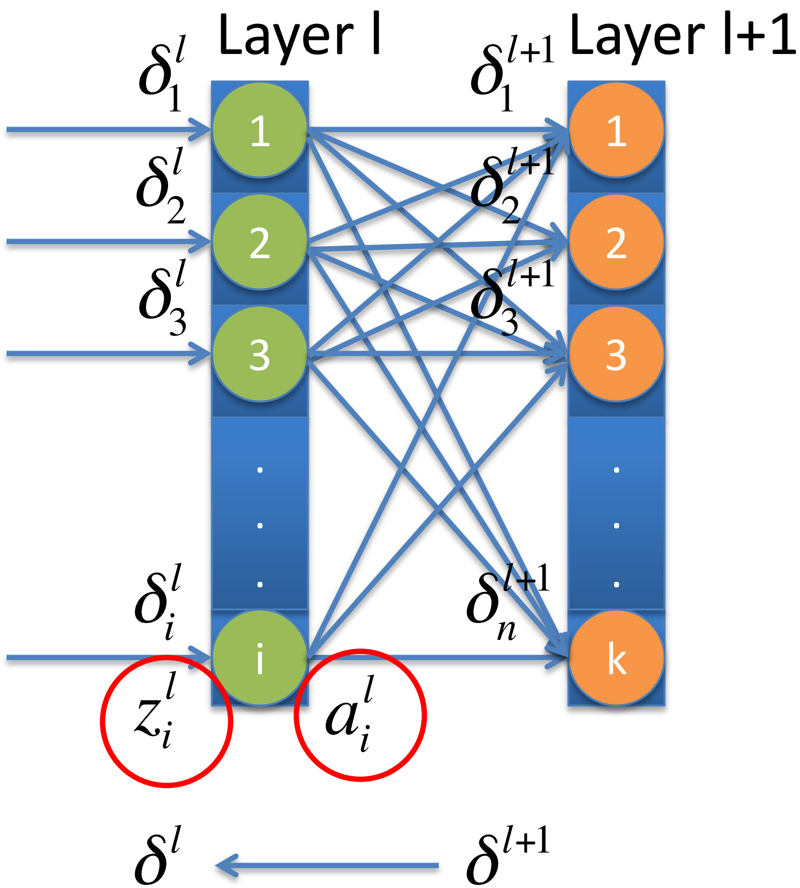
- Case 2:

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

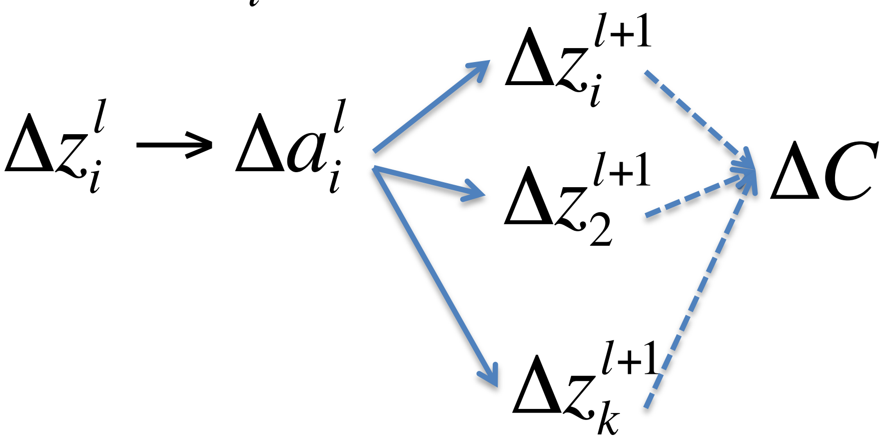


$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term

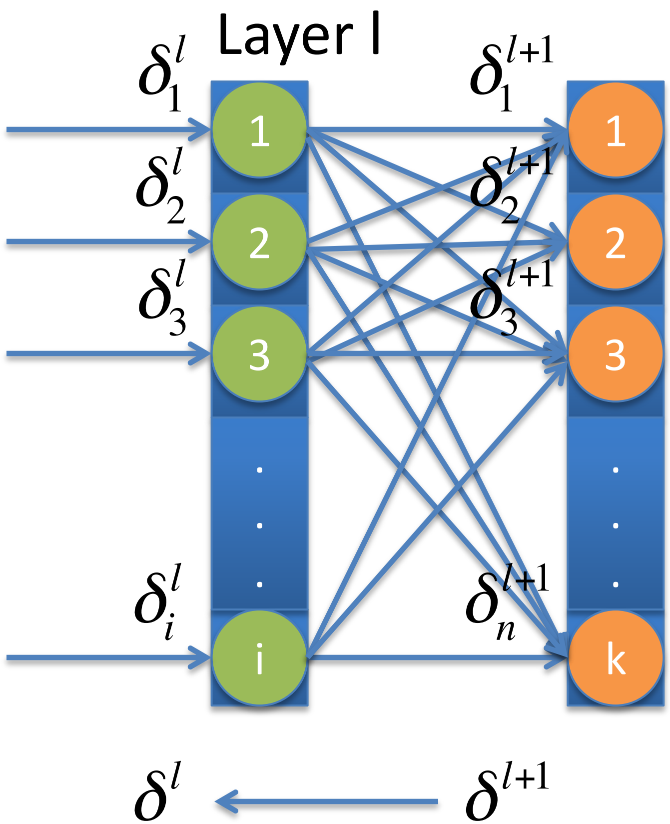


$$\delta_i^l = \frac{\partial C}{\partial z_i^l}$$



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

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



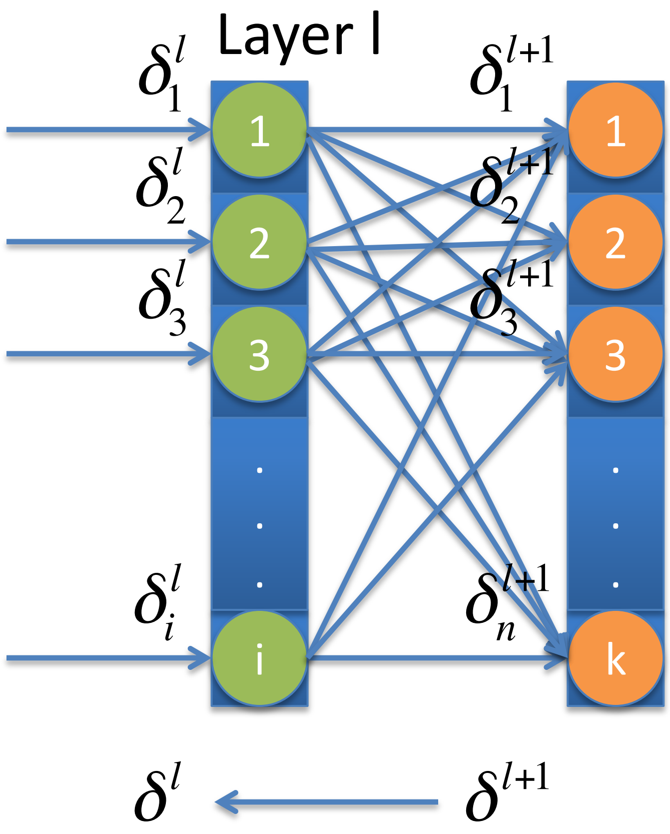
$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial C}{\partial z_k^{l+1}}$$

$$\sigma'(z_i^l) \qquad \delta_k^{l+1}$$

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

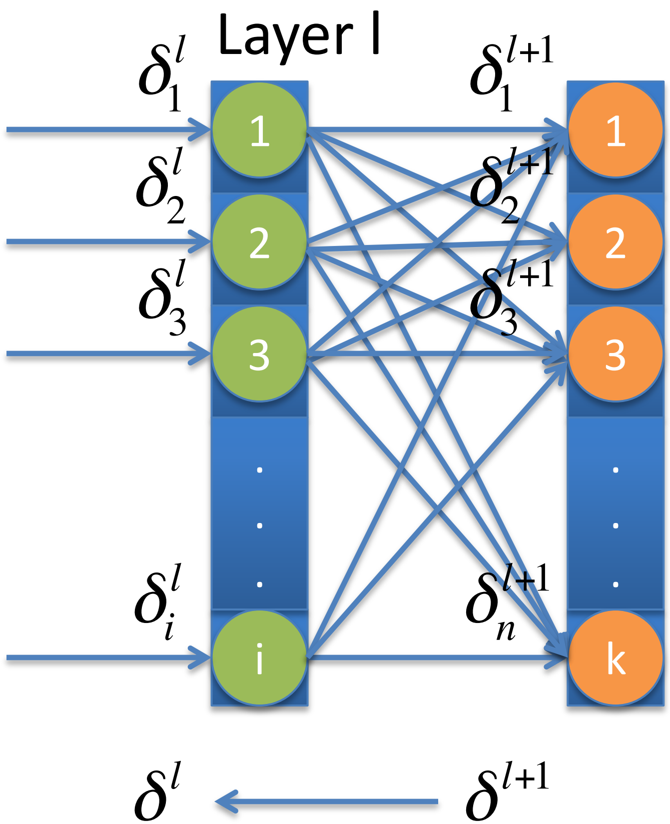
$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



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

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



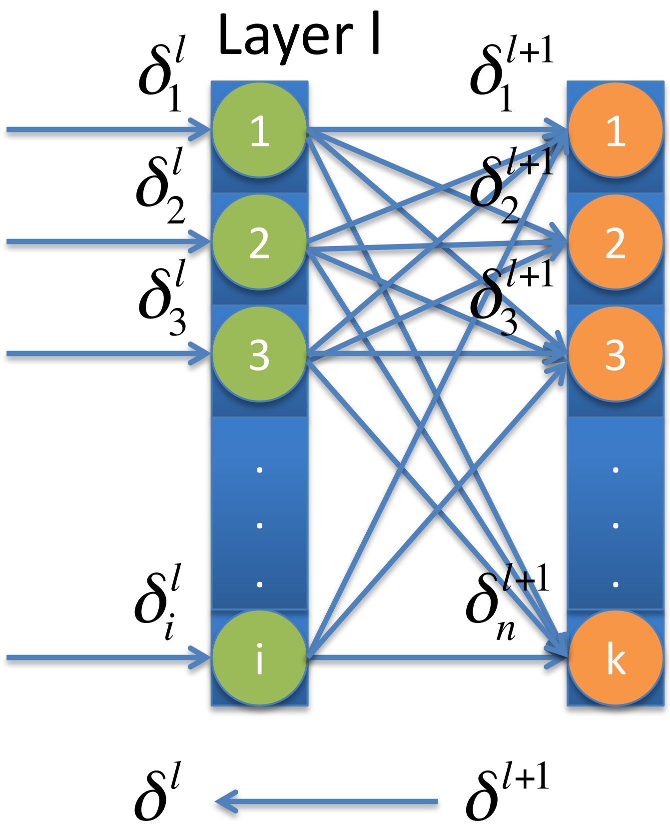
$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

$$\delta_1^l = \sigma'(z_1^l) \sum_k w_{k1}^{l+1} \delta_k^{l+1}$$

$$\delta_2^l = \sigma'(z_2^l) \sum_k w_{k2}^{l+1} \delta_k^{l+1}$$

$$\delta_3^l = \sigma'(z_3^l) \sum_k w_{k3}^{l+1} \delta_k^{l+1}$$

Compute $\frac{\partial C}{\partial w_{ij}^{l}}$ - Second term



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

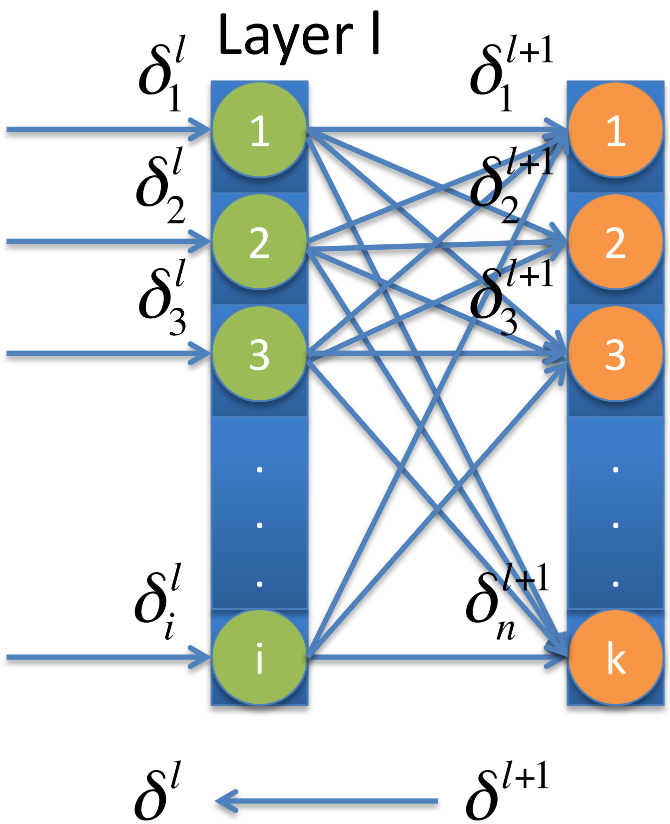
$$\delta_1^l = \sigma'(z_1^l)\sum_k w_{k1}^{l+1}\delta_k^{l+1}$$

$$\delta_2^l = \sigma'(z_2^l)\sum_k w_{k2}^{l+1}\delta_k^{l+1}$$

$$\delta_3^l = \sigma'(z_3^l)\sum_k w_{k3}^{l+1}\delta_k^{l+1}$$

$$W^{l+1} = \underbrace{\begin{bmatrix} w_{11}^{l} & w_{12}^{l} & \cdots \\ w_{21}^{l} & w_{22}^{l} & \cdots \\ \vdots & & \end{bmatrix}}_{N_l} \; \} N_{l+1}$$

Compute $\frac{\partial C}{\partial w^l_{ij}}$ - Second term



$$\delta^l_i = \sigma'(z^l_i)\sum_k w^{l+1}_{ki}\delta^{l+1}_k$$

$$\delta^l_1 = \sigma'(z^l_1)\sum_k w^{l+1}_{k1}\delta^{l+1}_k$$

$$\delta^l_2 = \sigma'(z^l_2)\sum_k w^{l+1}_{k2}\delta^{l+1}_k$$

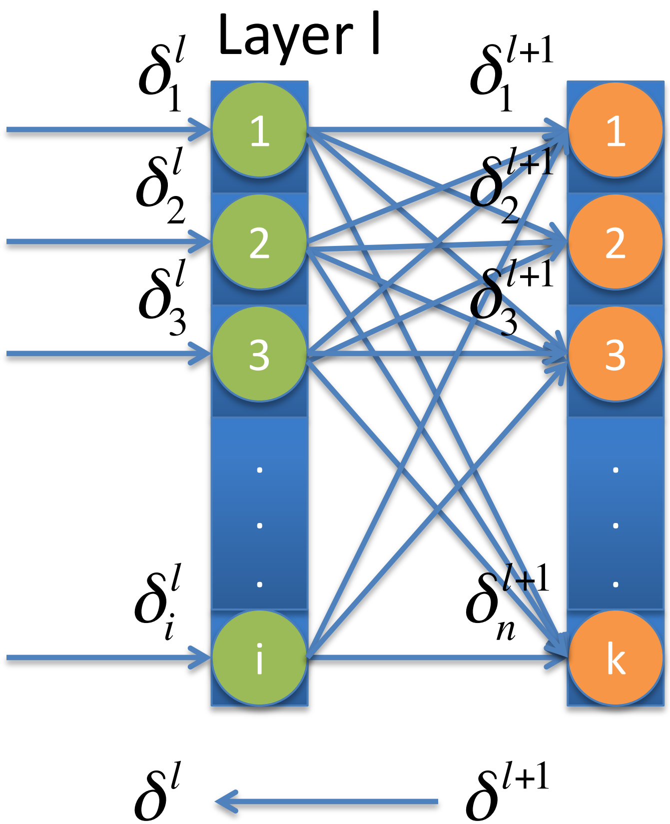
$$\delta^l_3 = \sigma'(z^l_3)\sum_k w^{l+1}_{k3}\delta^{l+1}_k$$

N_l

$$W^{l+1} = \begin{bmatrix} w^l_{11} & w^l_{12} & \cdots \\ w^l_{21} & w^l_{22} & \cdots \\ \vdots & & \end{bmatrix}$$

N_{l+1}

Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



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

$$\delta_1^l = \sigma'(z_1^l)\sum_k w_{k1}^{l+1}\delta_k^{l+1}$$

$$\delta_2^l = \sigma'(z_2^l)\sum_k w_{k2}^{l+1}\delta_k^{l+1}$$

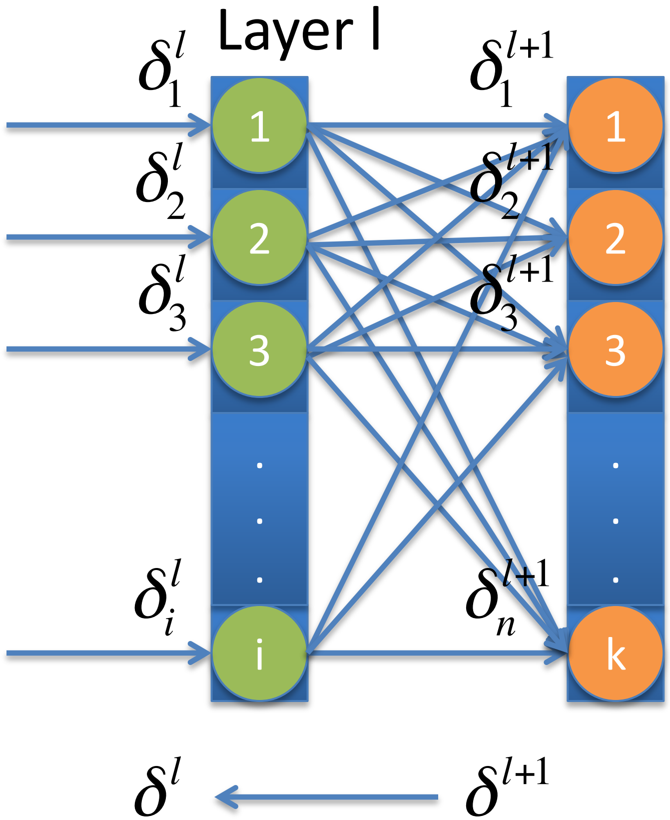
$$\delta_3^l = \sigma'(z_3^l)\sum_k w_{k3}^{l+1}\delta_k^{l+1}$$

N_l

$$W^{l+1^T} = \begin{bmatrix} w_{11}^l & w_{21}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & & \end{bmatrix}$$

N_{l+1}

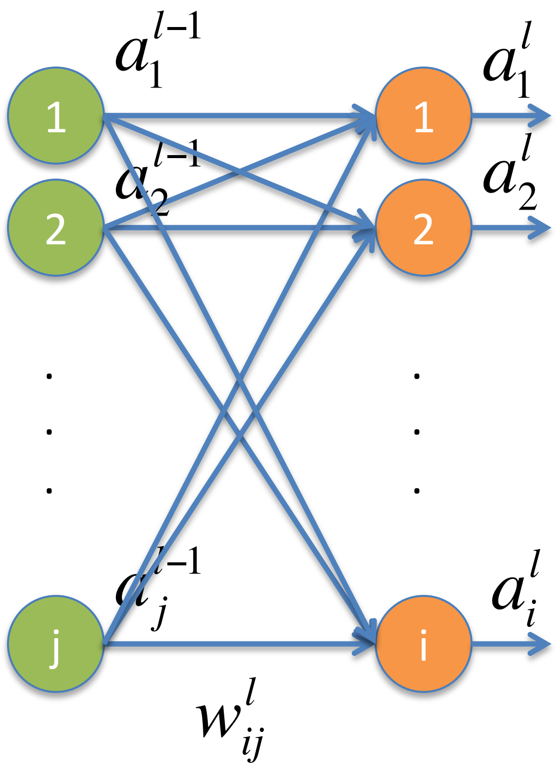
Compute $\frac{\partial C}{\partial w_{ij}^l}$ - Second term



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

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

Compute $\frac{\partial C}{\partial w_{ij}^l}$



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

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

Forward pass

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

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

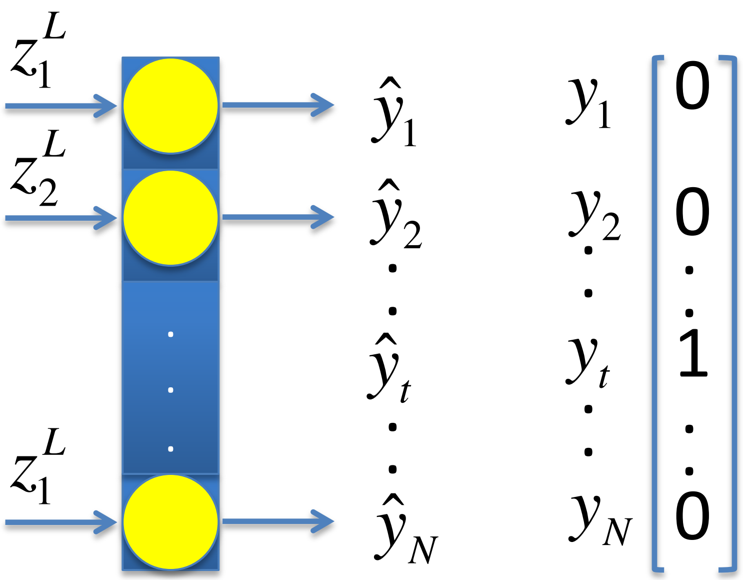
Backward pass

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

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

Loss function

Ouput layer L



- Sofmax function:

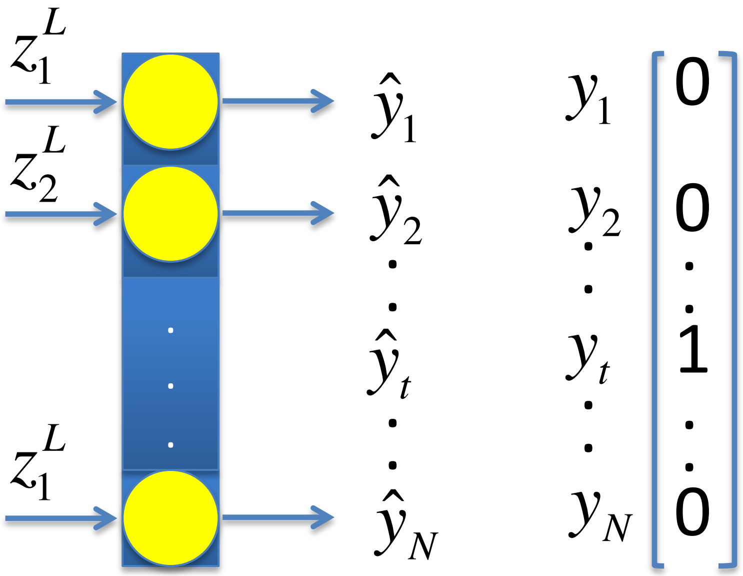
$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

- Cross Entropy (CE):

$$c(\theta) = C_x(\theta) = -\log y_t$$

Cross Entropy Loss function

Ouput layer L



$$C_x(\theta) = -\log y_t \qquad \delta_i^L = \frac{\delta C_x}{\delta z_i^L}$$

- Case 1: $i = t$

$$\delta_i^L = \frac{\delta C_x}{\delta z_t^L} = \frac{\delta C_x}{\delta y_t}\frac{\delta y_t}{\delta z_t^L}$$

$$= -\frac{1}{\hat{y}_t}\hat{y}_t(1-\hat{y}_t) = \hat{y}_t - 1$$

- Case 2: $i \neq t$

$$\delta_i^L = \frac{\delta C_x}{\delta z_i^L} = -\frac{1}{\hat{y}_t}(-\hat{y}_t\hat{y}_i) = \hat{y}_i$$

Live Voting

Overview

- Review
  - Backpropagation
  - Cross-entropy loss function
- Activation function
  - Sigmoid
  - Tanh
  - ReLU
- Setting a Network
- Learning: practical usages

Activation function

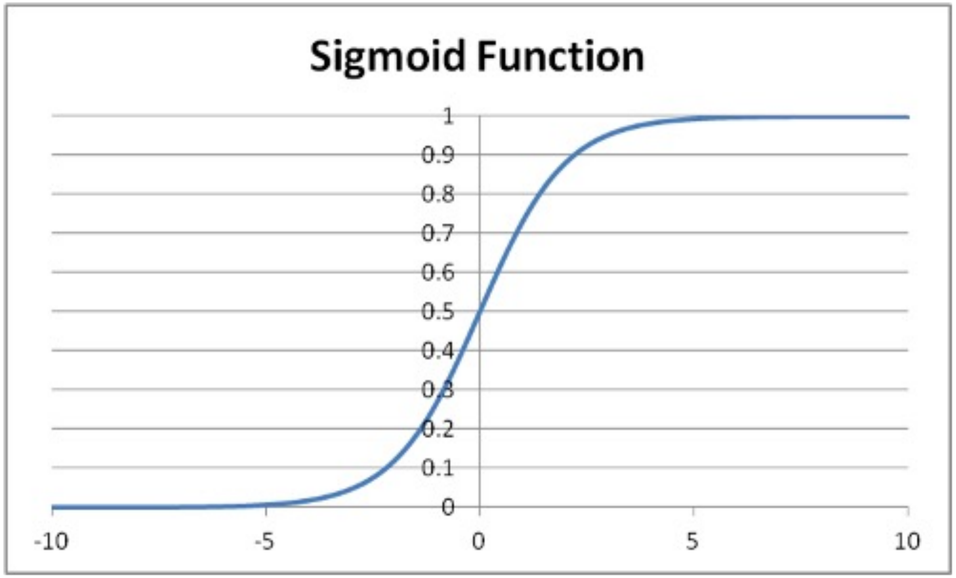
- For the hidden layers:
  - Sigmoid
  - Tanh
  - ReLu and its variation
- For the output layer
  - Softmax

Sigmoid function

- Frequently used activation function
- It is still used as baseline system

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = ?$$



Sigmoid function

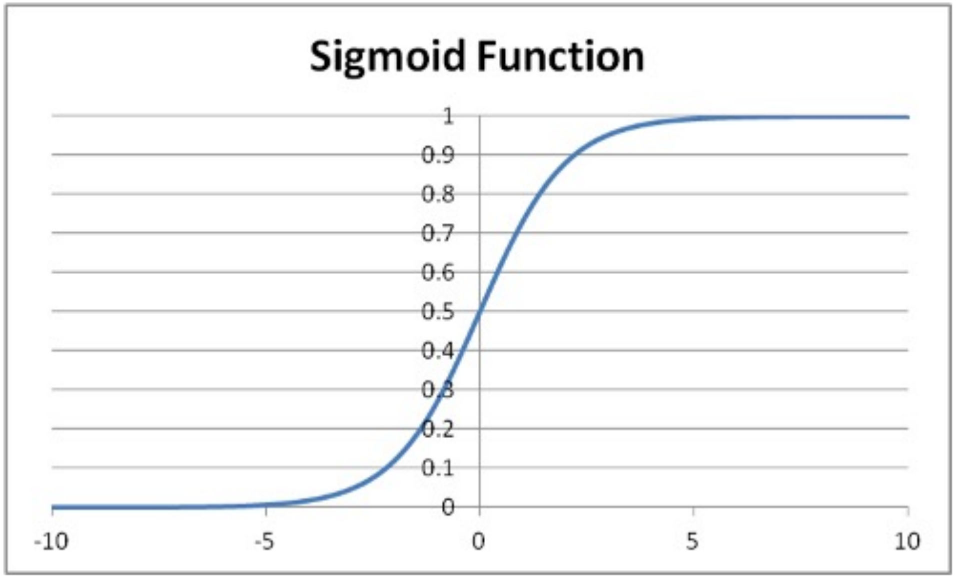
- Frequently used activation function
- It is still used as baseline system

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = -1.(1+e^{-z})^{-2}(-e^{-z})$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}$$

$$= \frac{1}{1+e^{-z}}.(1-\frac{1}{1+e^{-z}}.) = \sigma(z)(1-\sigma(z))$$

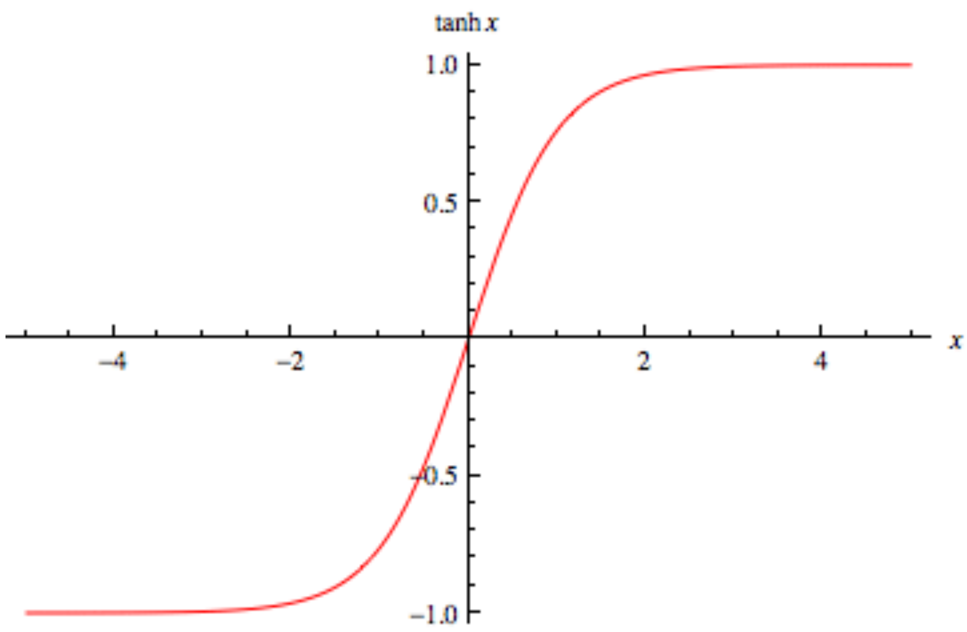


Tanh function

- In NLP applications, tanh function is more often used than sigmoid function

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\tanh'(z) = ?$$

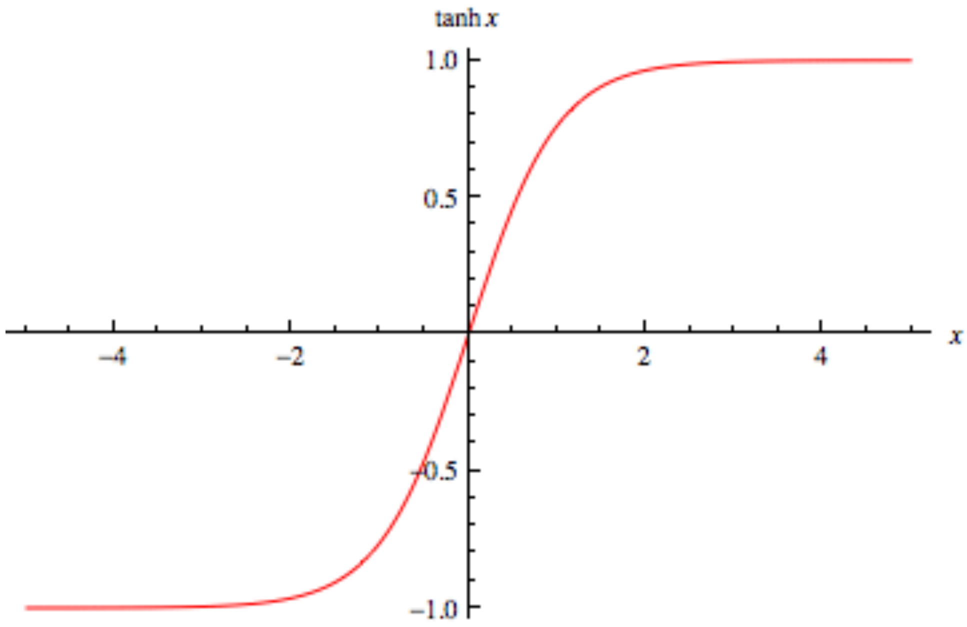


Tanh function

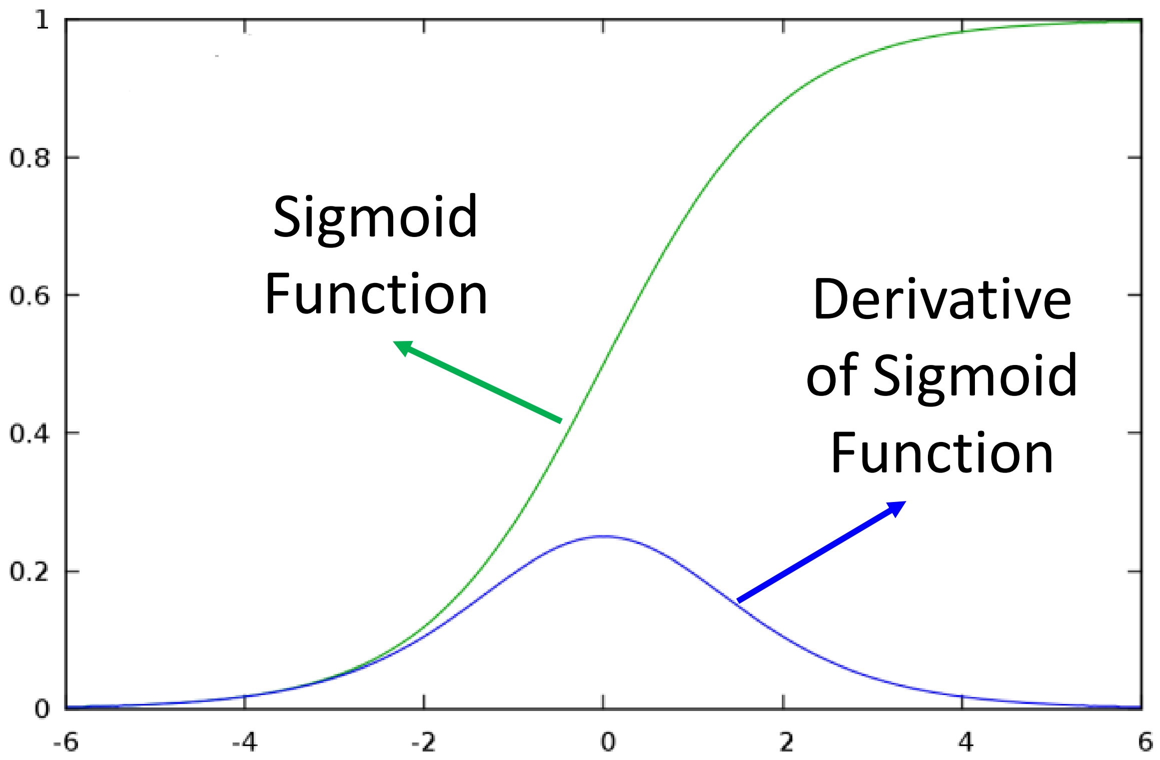
- In NLP applications, tanh function is more often used than sigmoid function

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\tanh'(z) = ... = 1 - \tanh^2(z)$$

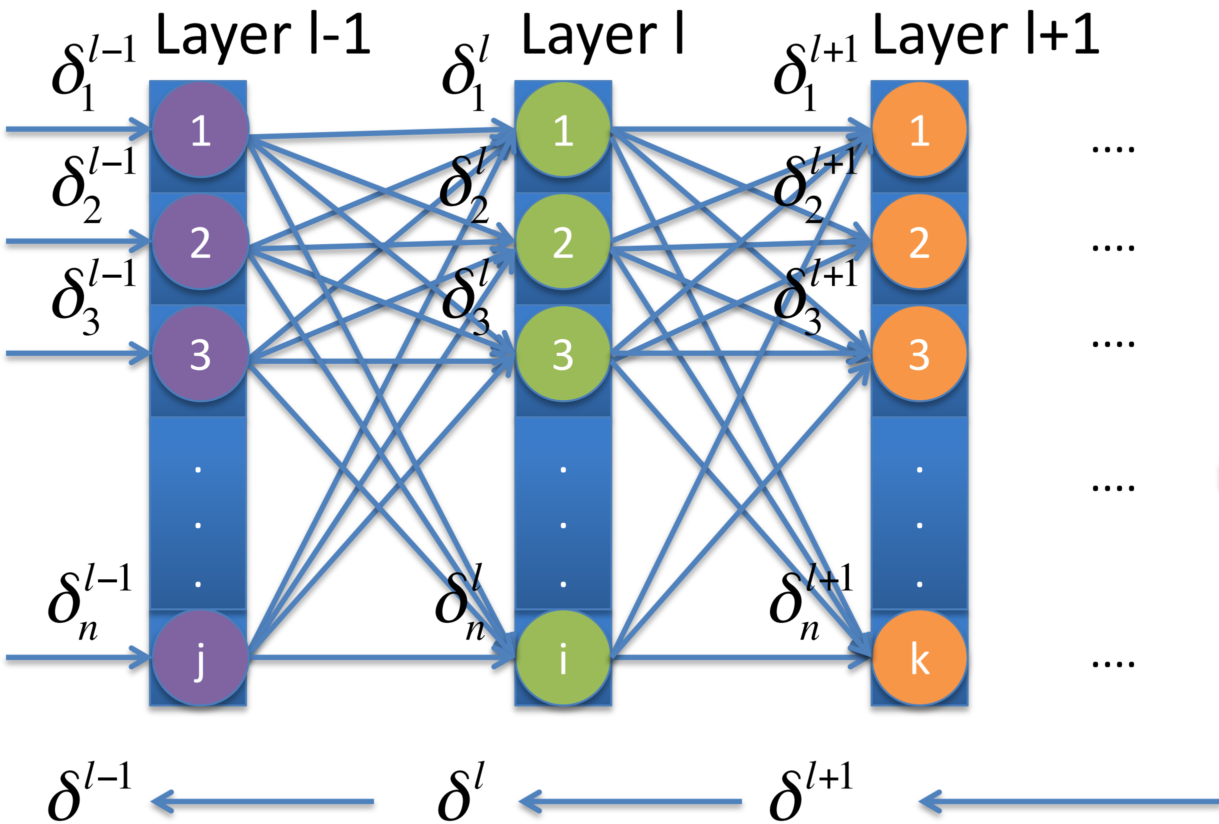


Problem of Sigmoid



Vanishing Gradient Problem

- Backward pass

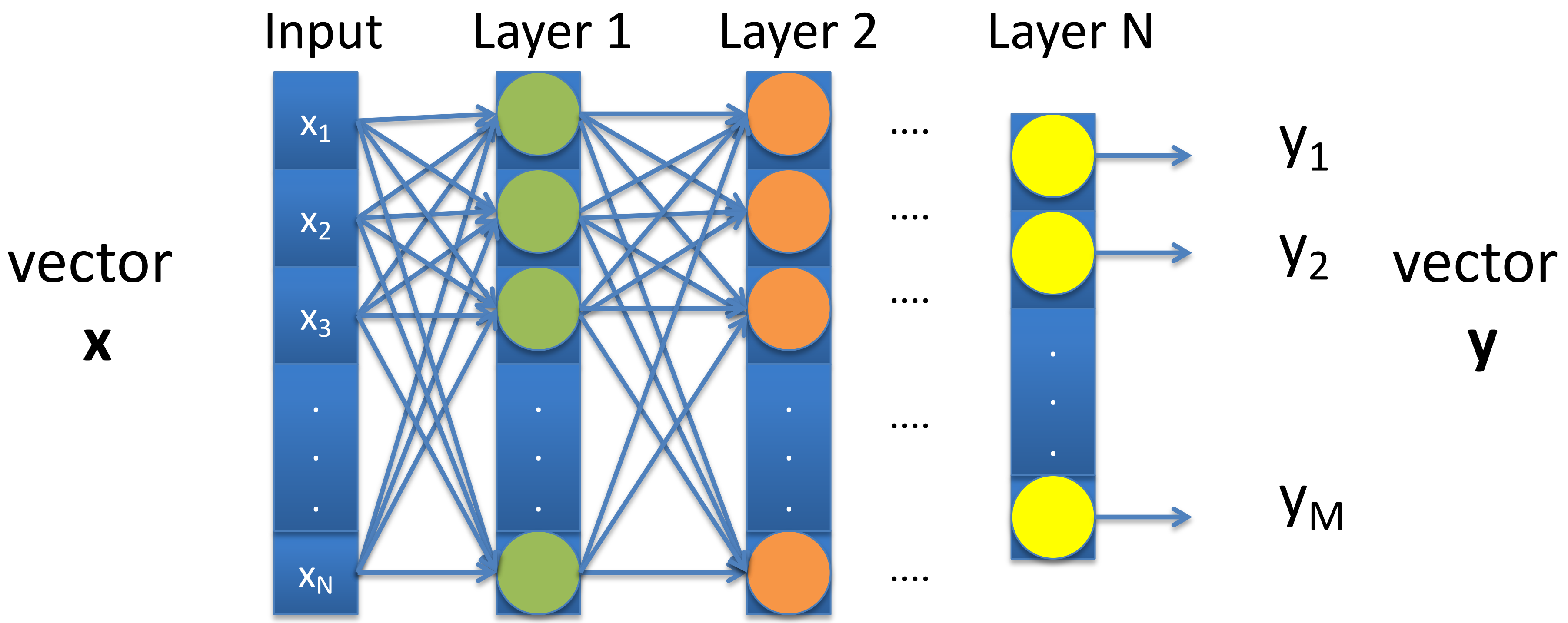


$$\delta^L = \sigma'(z^L)\nabla C(y)$$

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

→ Gradient is getting smaller

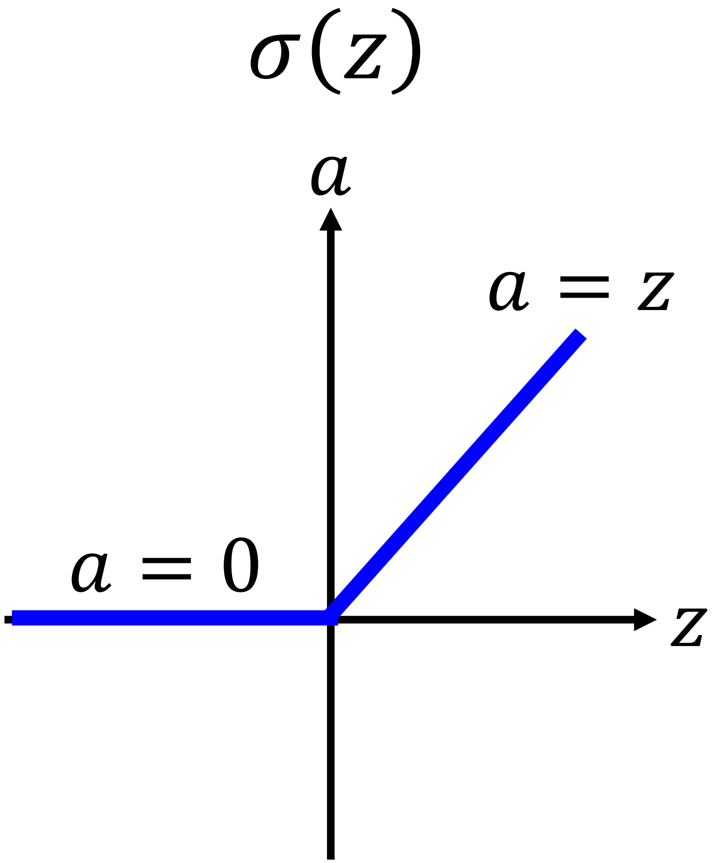
Vanishing Gradient Problem

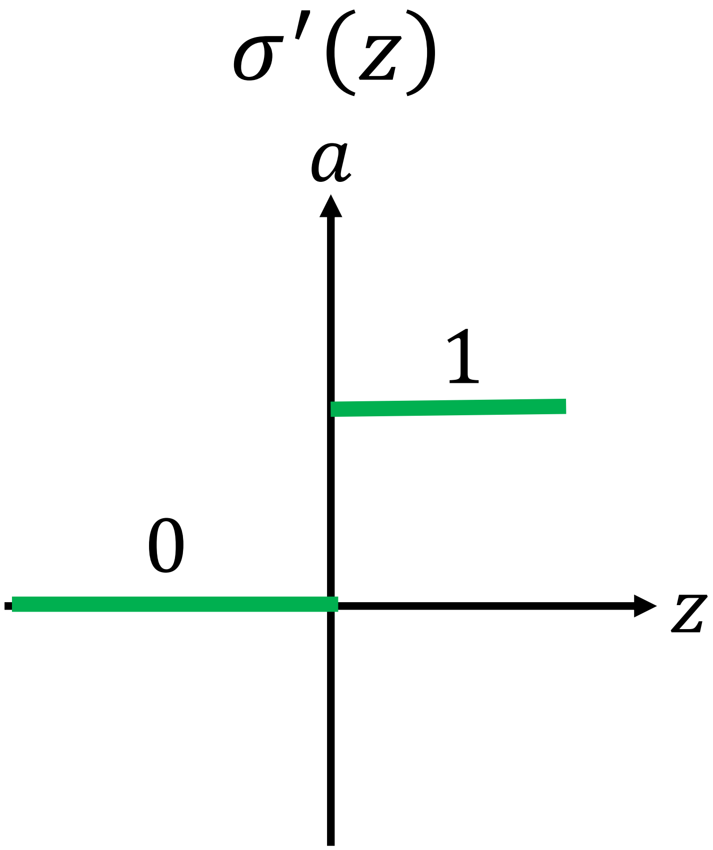


Learn very slowly
Still random

Learn faster
Already converge

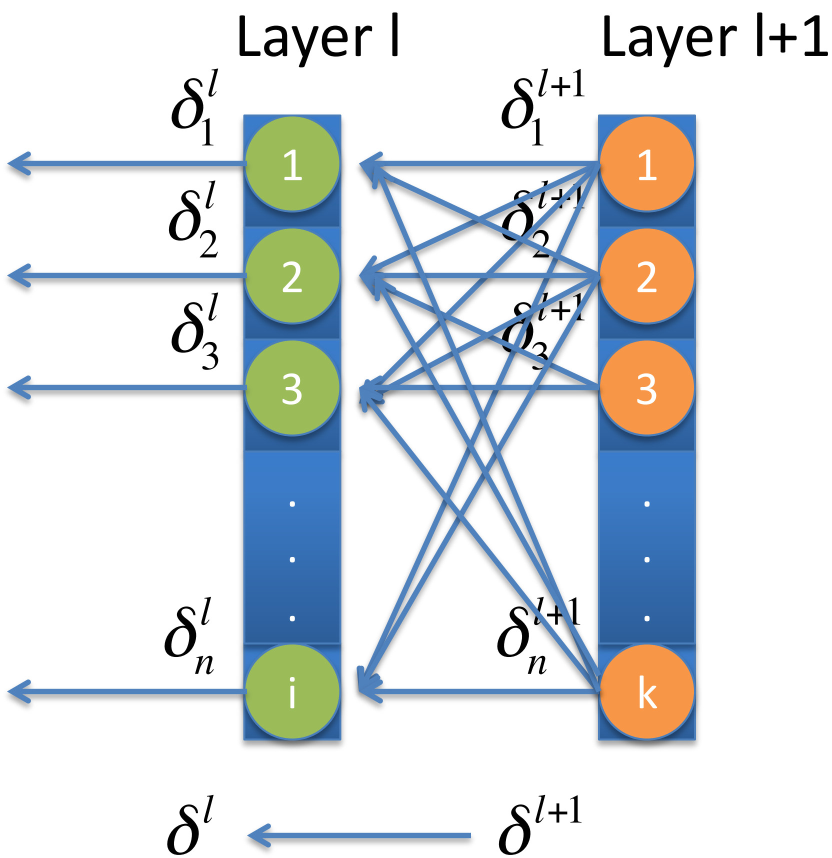
Recitifier Linear Unit (ReLU)



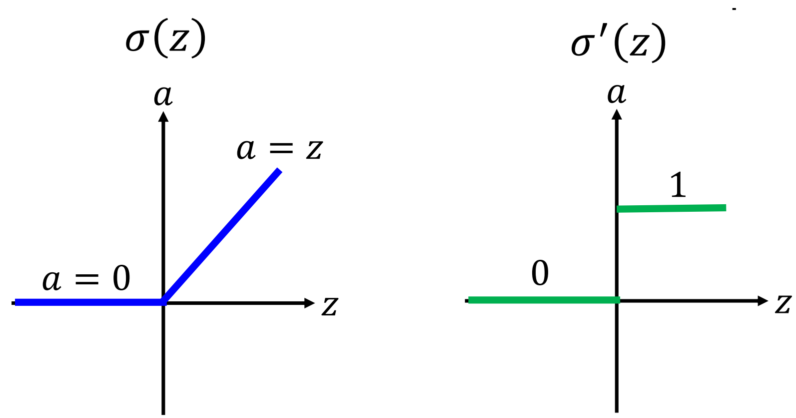


ReLU

- Backward Pass

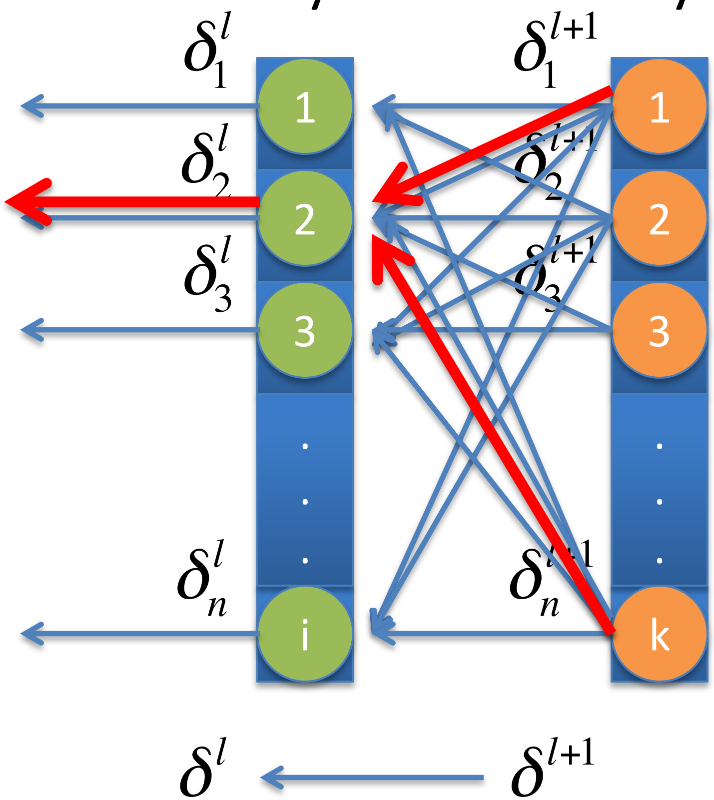


$$\delta^{l} = \sigma'(z^{l})(W^{l+1})^{T}\delta^{l+1}$$



ReLU

- Backward Pass

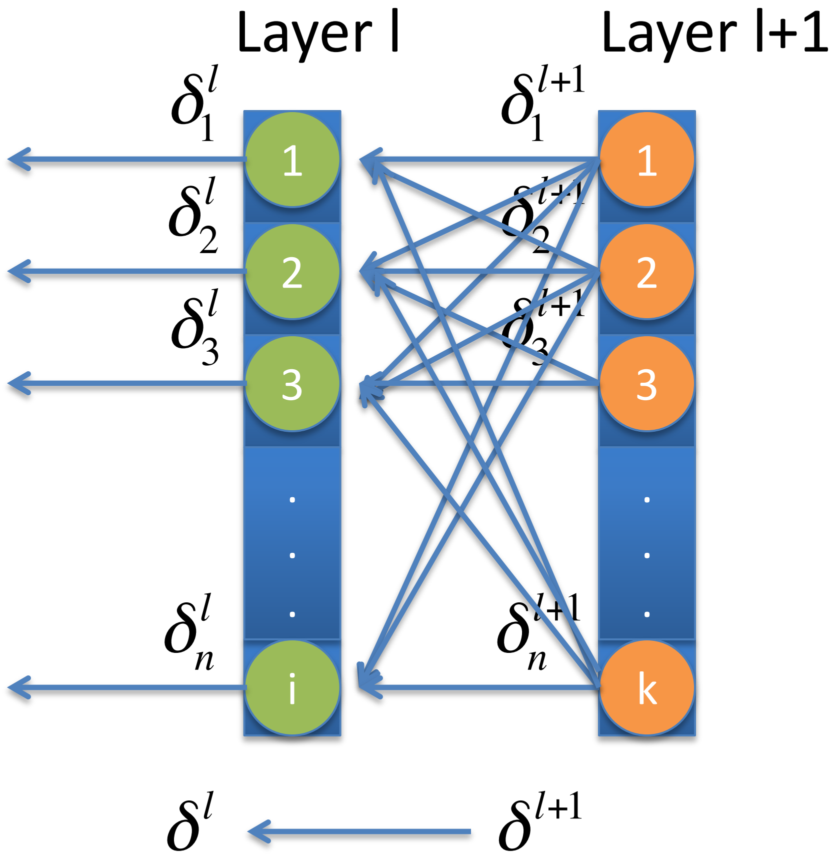


$$\delta^l = \sigma'(z^l)(W^{l+1})^T \delta^{l+1}$$

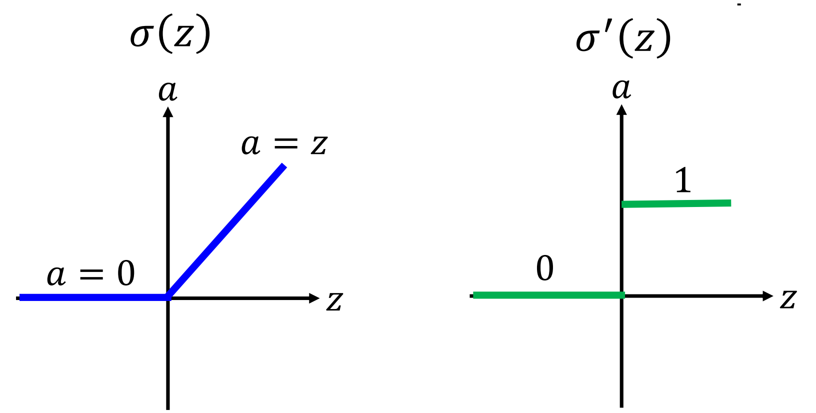
- All the weights connected to not-activated neurons have zero gradient
- Only activated neurons impact the training

ReLU

- Backward Pass



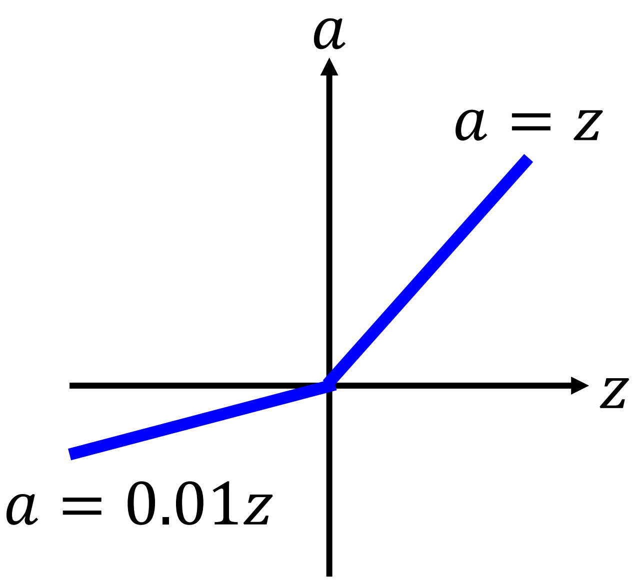
$$\delta^l = \sigma'(z^l)(W^{l+1})^T \delta^{l+1}$$



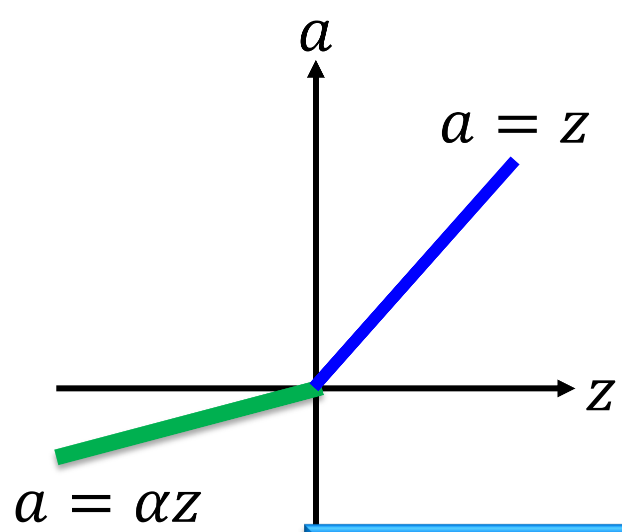
The Dying ReLU Problem

ReLU - Variants

Leaky ReLU



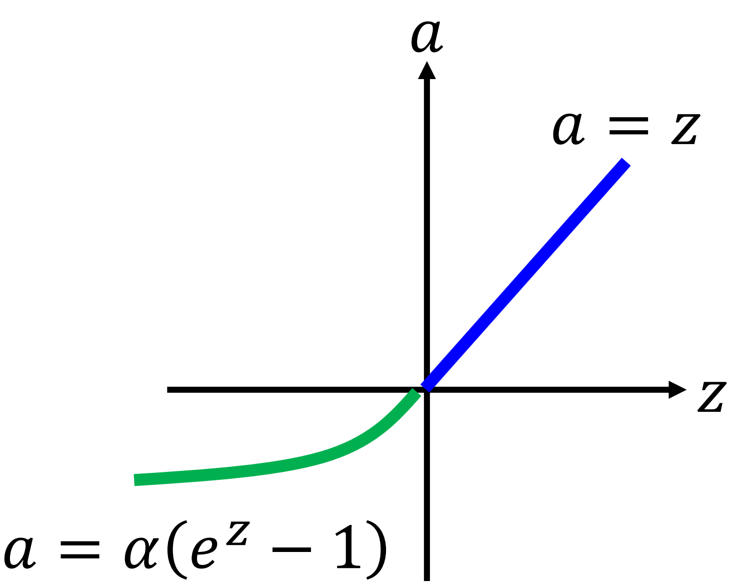
Parametric ReLU



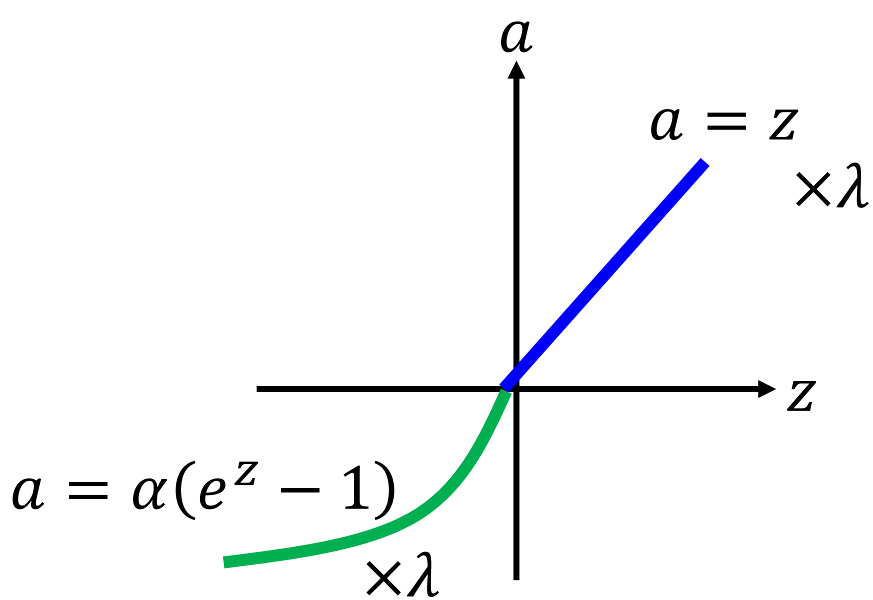
α also learned by gradient descent

ReLU - Variants

Exponential Linear Unit (ELU)



Scaled ELU (SELU)



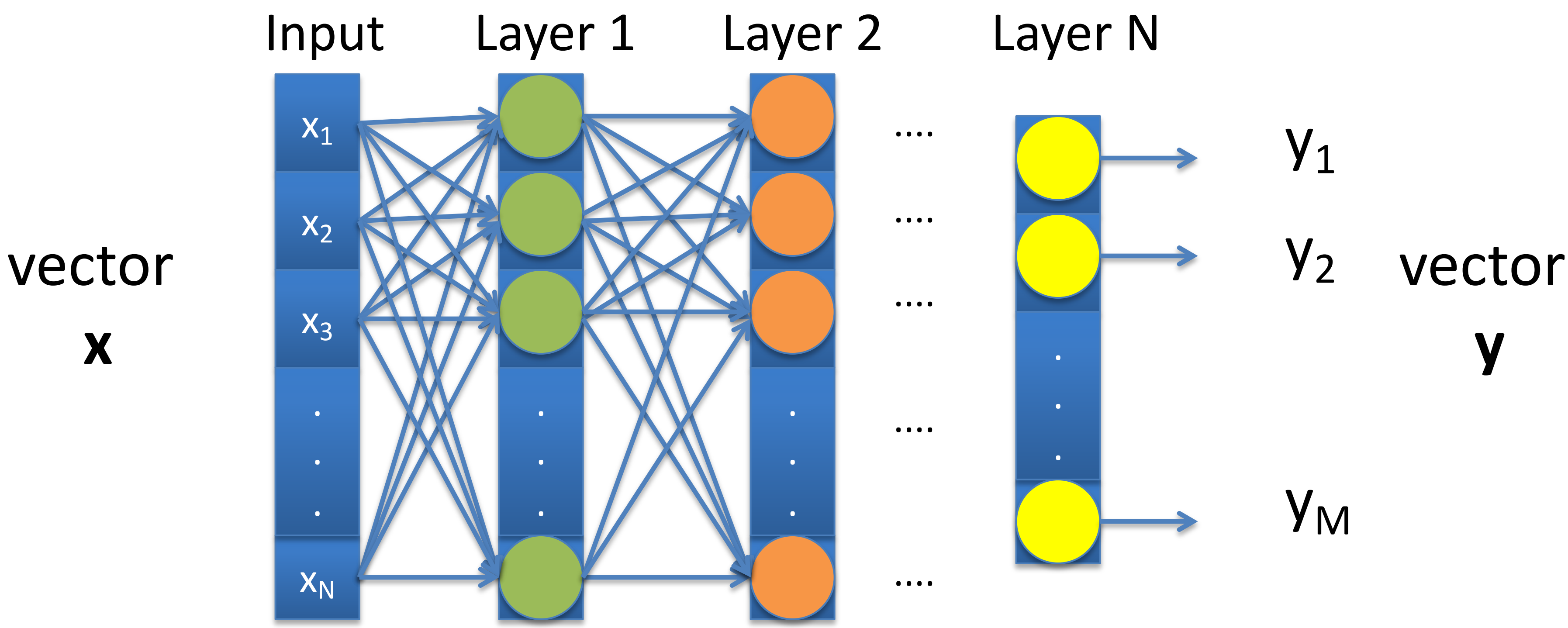
$$\alpha = 1.6732632423543772848170429916717$$
$$\lambda = 1.0507009873554804934193349852946$$

Live Voting

Overview

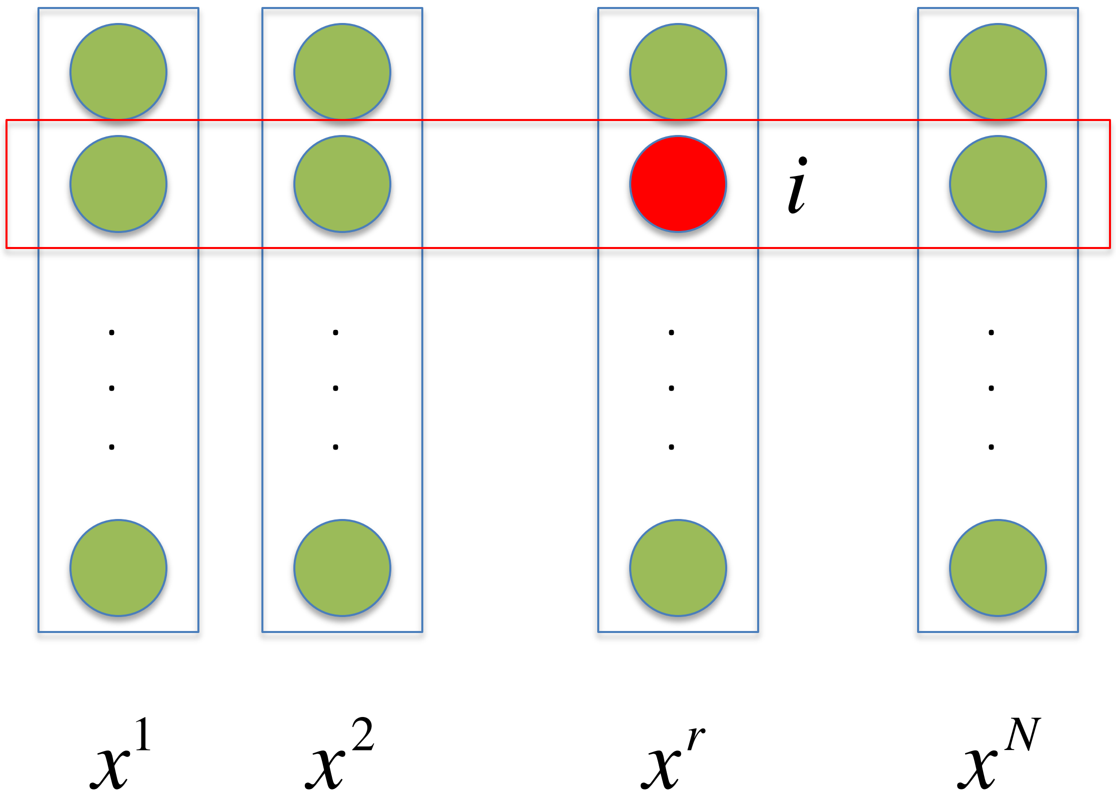
- Review
  - Backpropagation
  - Cross-entropy loss function
- Activation function
  - Sigmoid
  - Tanh
  - ReLU
- Setting a Network
- Learning: practical usages

Computation of the final output



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

The Input



- For each of the dimension compute the mean m_i σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

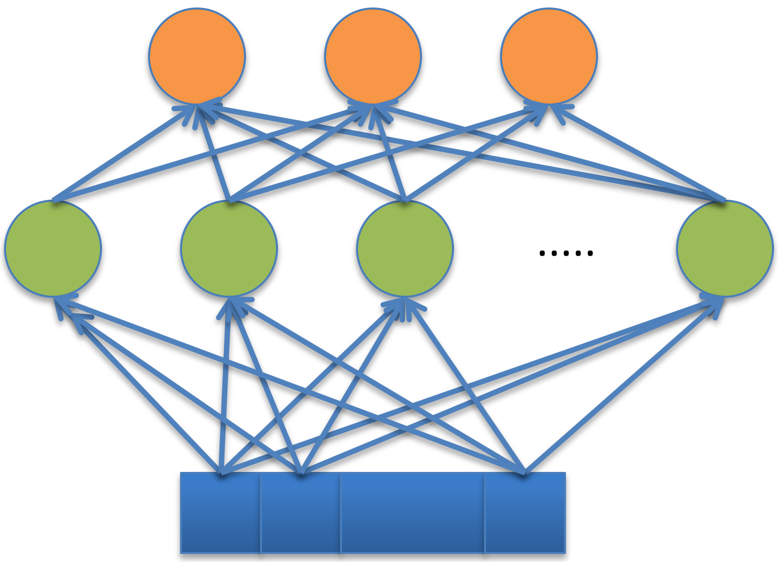
- Mean = 0
- Variance = 1

Setting up the architecture

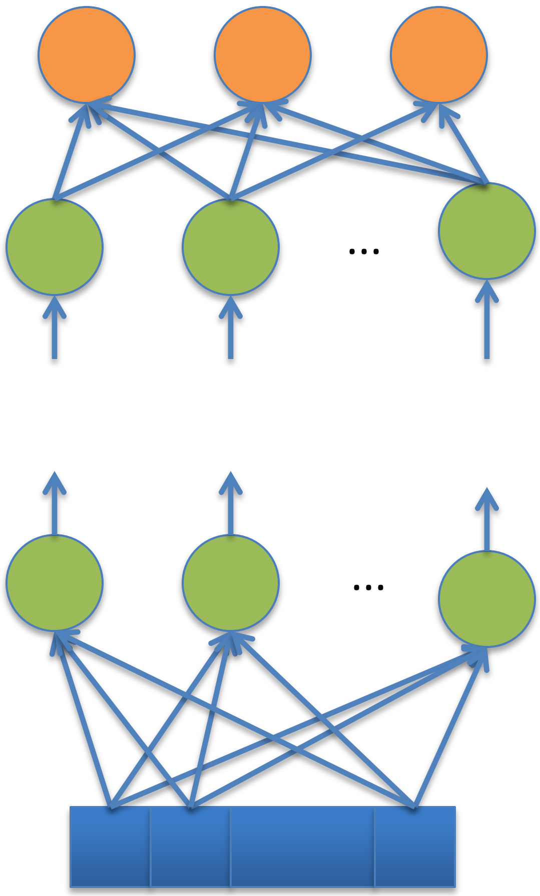
You need to answer the following questions:

- How many layers?
  - Shallow vs. deep
- How many nodes for each hidden layer?
  - For the output layer, #nodes = #target classes
- Which activation function?
  - Sigmoid vs. Tanh vs. ...

Fat + short vs. Thin + tall ??



Shallow



Deep

Fat + short vs. Thin + tall: ASR

- Word error rate (WER)

L x N	WER (%)
1 x 2k	24.2
2 x 2k	20.4
3 x 2k	18.4
4 x 2k	17.8
5 x 2k	17.2
7x2k	17.1

1 x N	WER (%)
1 x 3772	22.5
1 x 4,634	22.6
1 x 16k	22.1

Frank Seide, Gang Li and Dong Yu. Conversational Speech Transcriptions Using Context-Dependent Deep Neural Networks. In Proc. of Interspeech, 2011

Overview

- Review
  - Backpropagation
  - Cross-entropy loss function
- Activation function
  - Sigmoid
  - Tanh
  - ReLU
- Setting a Network
- Learning: practical usages

Learning: Practical Usages

- Parameter Initialization
- Learning Rate
- Stochastic Gradient Descent and Mini-batch
- Learning Recipe

Parameter Initialization

- For gradient descent, we need to intialize the parameters θ_0
- It has strong impact on the model quality
  - Different parameter initialization $\longrightarrow$ different results
- Some suggestions
  - Do not set the parameters equally
  - Set the parameters randomly, e.g. uniform (-r,r) distribution

$$r = \sqrt{6 / (fan\_in + fan\_out)}$$

  - Train several models with different parameter initializations and combine them

Stochastic Gradient Descent

- Gradient Descent:

$$\theta_i \leftarrow \theta_{i-1} - \eta \nabla C(\theta_{i-1})$$

  - In which

$$\nabla C(\theta_{i-1}) = \frac{1}{R} \sum_r \nabla C^r(\theta)$$

- Stochastic Gradient Descent:
  - Pick an example x^r

$$\theta_i \leftarrow \theta_{i-1} - \eta \nabla C^r(\theta_{i-1})$$

Stochastic Gradient Descent

- *What is an epoch?*
- Training data: $(x_1, y_1), (x_2, y_2), \ldots, (x_R, y_R)$
- When using the stochastic gradient descent:
  - Starting with θ_0
  - Pick (x_1, y_1) $\quad \theta_1 \leftarrow \theta_0 - \eta \nabla C^1(\theta_0)$

    (x_2, y_2) $\quad \theta_2 \leftarrow \theta_1 - \eta \nabla C^2(\theta_1)$

    (x_R, y_R) $\quad \theta_R \leftarrow \theta_{R-1} - \eta \nabla C^R(\theta_{R-1})$

    (x_1, y_1) $\quad \theta_{R+1} \leftarrow \theta_R - \eta \nabla C^{R+1}(\theta_R)$

Seen all the training data

One epoch

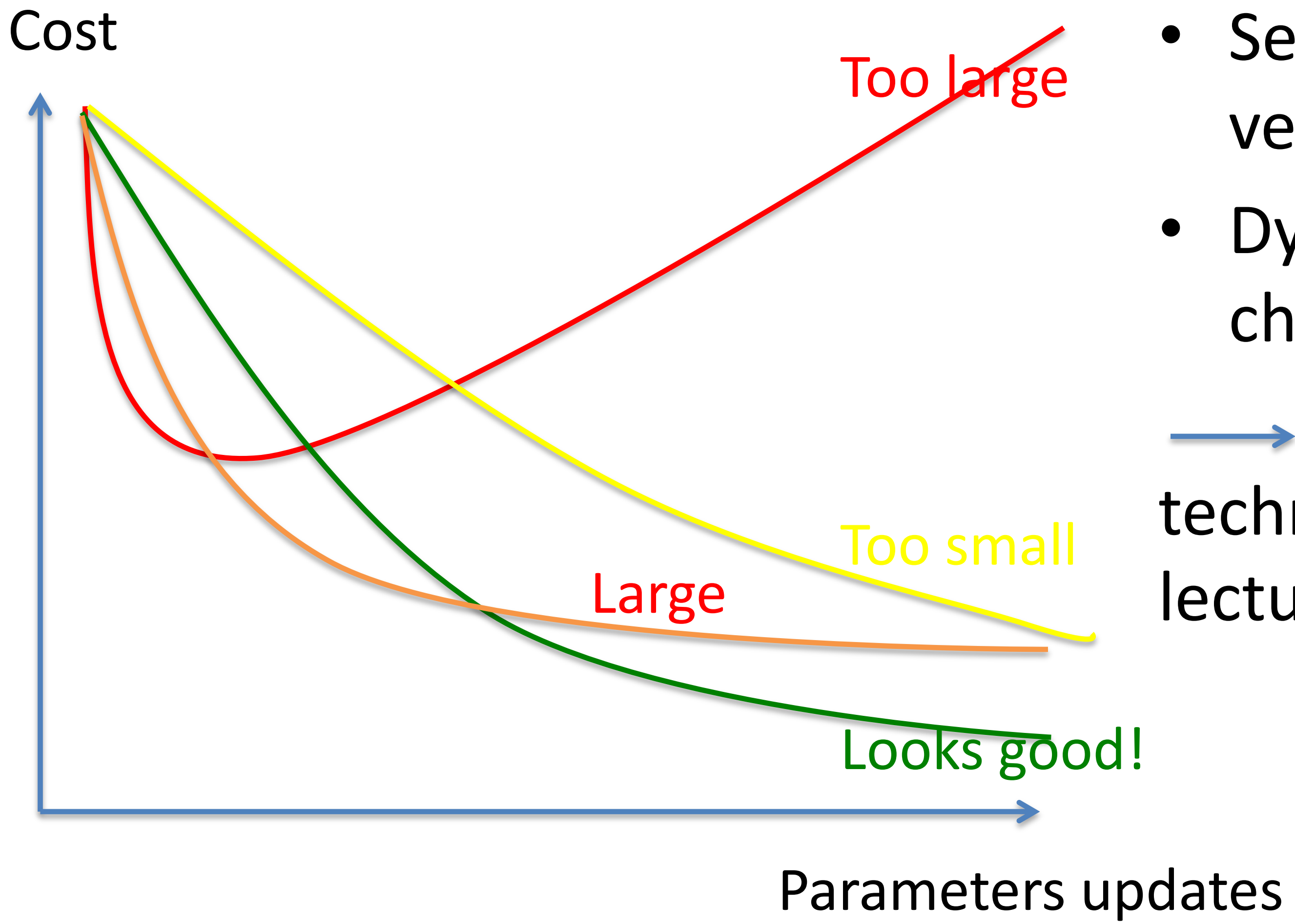
Stochastic Gradient Descent

- Mini-batch Gradient Descent:
  - Pick B examples as a batch b
  - B is the batch size

$$\theta_i \leftarrow \theta_{i-1} - \eta \frac{1}{B} \sum_{x_r \in b} \nabla C^r(\theta_{i-1})$$

- Mini-batch Gradient Descent is faster than Stochastic Gradient Descent
  - Less updates
  - Better parallelization
- <u>Important:</u> Shuffle the data after each epoch

Learning rate



- Set the learning rate very carefully
- Dynamically changed

→ More advanced techniques in the next lectures

Learning Rate

- Popular & Simple idea: Reduce the learning rate by some factor every few epochs
  - At the beginning, larger learning rate
  - After several epochs, reduce the learning rate

$$\eta^t = \eta / (t+1)$$

When to reduce the learning rate?

How much should we reduce the learning rate?

Learning Recipe

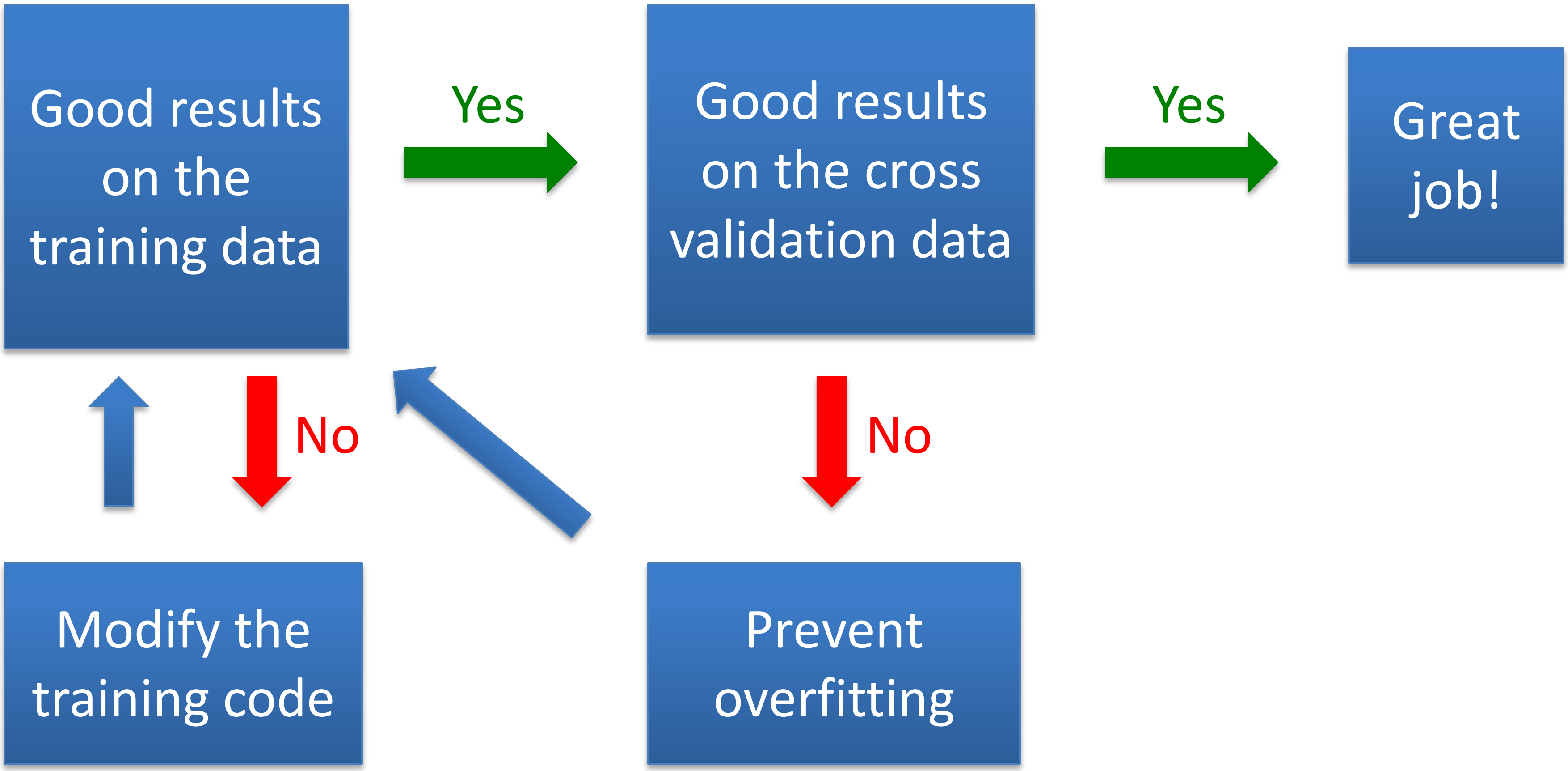
- Split the data in three parts

Training Data	Cross Validation Data	Evaluation Data
Used for training and monitoring the performance	Used for monitoring the performance and tuning parameters	Don't touch it till the deadline

Learning Recipe

- Monitor the cost function after each update
- Evaluate your model not only on the cross validation but also on the training data
  - The model should work well at first on the training data
- The learning rate can be adjusted based on the performance of the cross validation set
  - Decrease the learning rate if no improvement can be observed on the cross validation set

Learning Recipe



Thanks for listening!