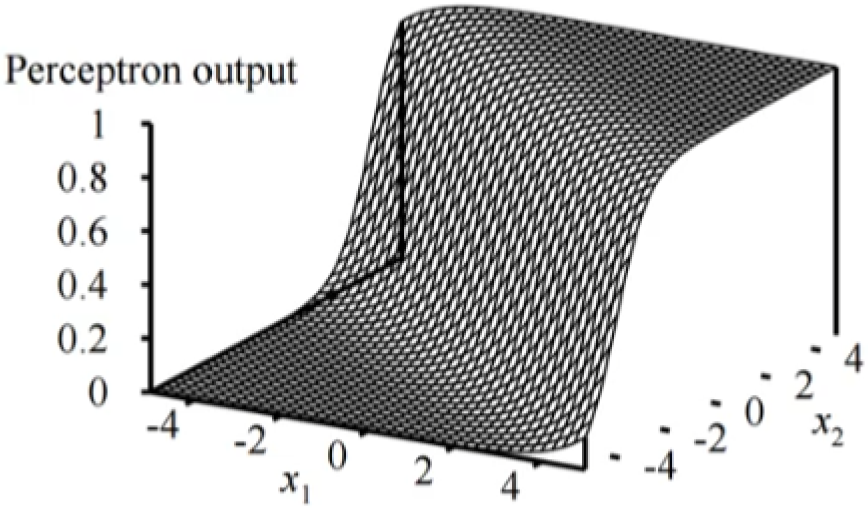
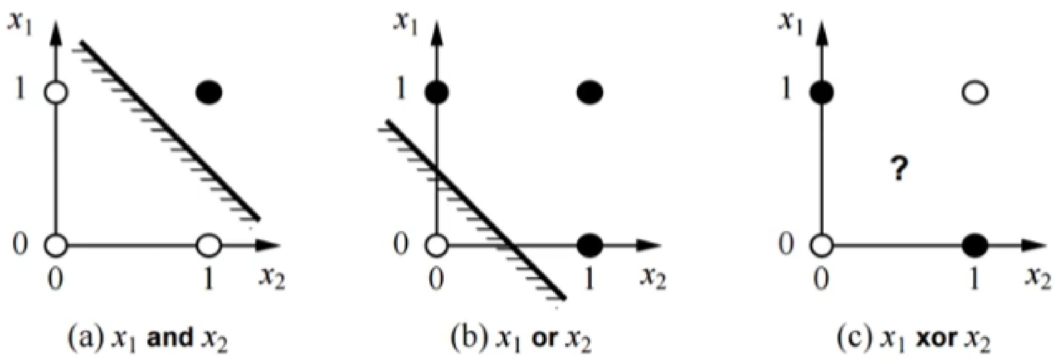
4 如何训练模型

4.1 模型架构

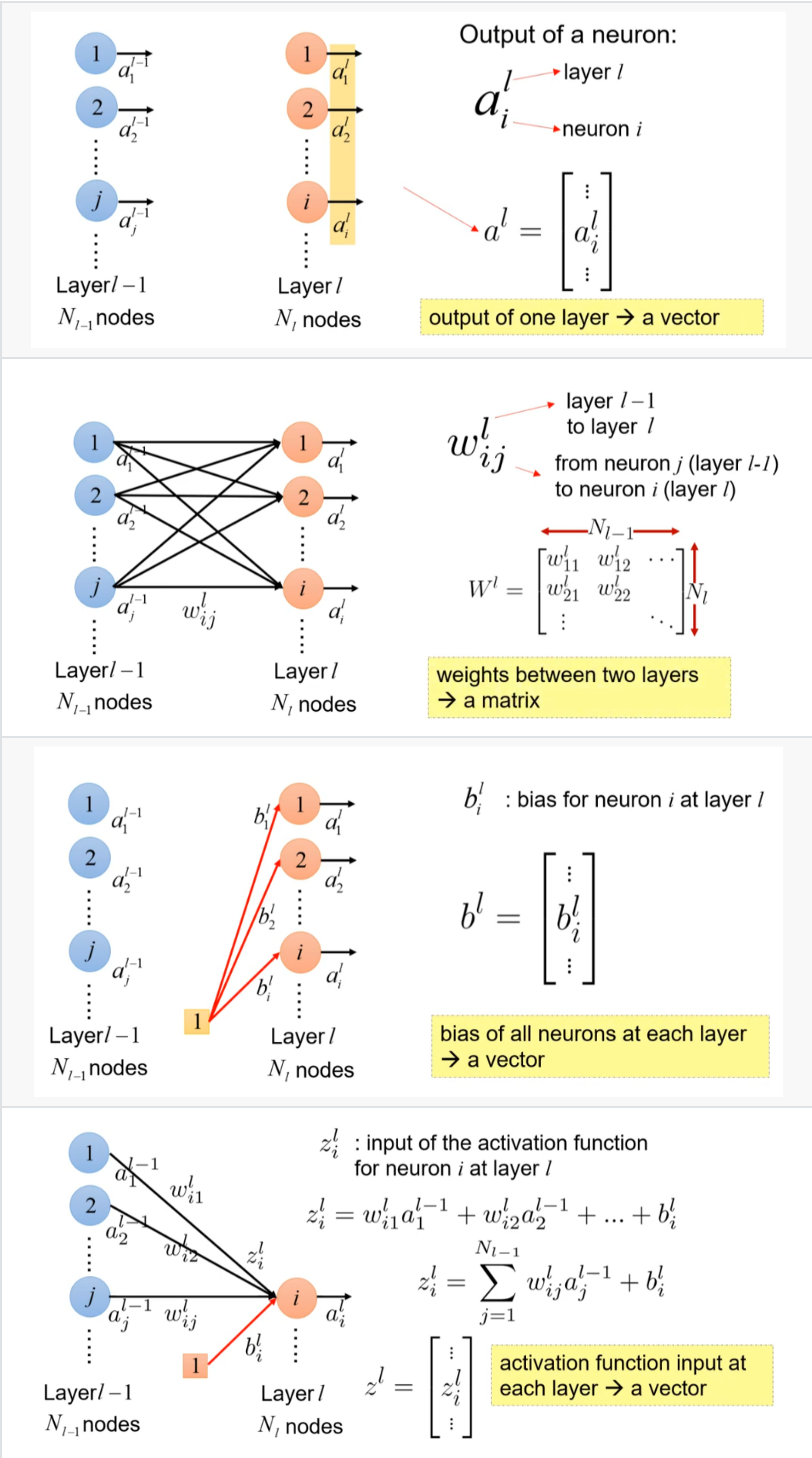
1. **A Single Layer of Neurons(Perceptron)**
    - 输出单元的都是分离的（没有共享的权重）；
    - 调整权重来改变位置、方向和陡峭程度；



2. **Limitation of Perceptron：例如下图中的第三种情况无法表示。**



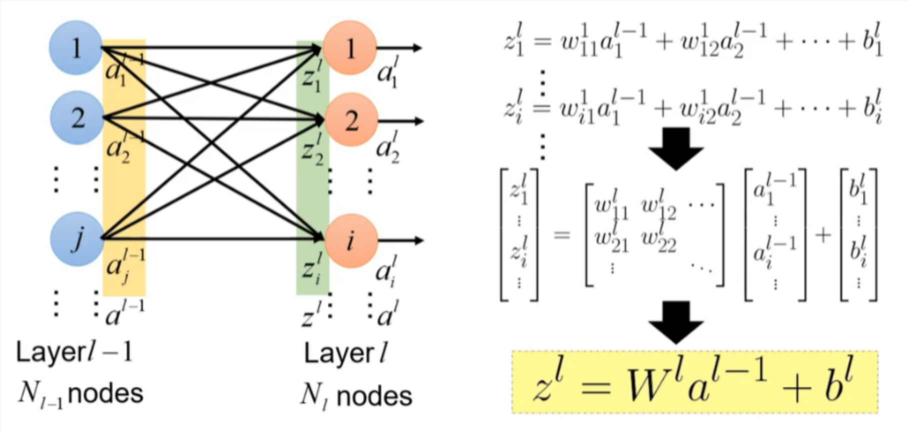
3. **Neural Network Model(Multi-Layer Perceptron)**
    - 符号定义：

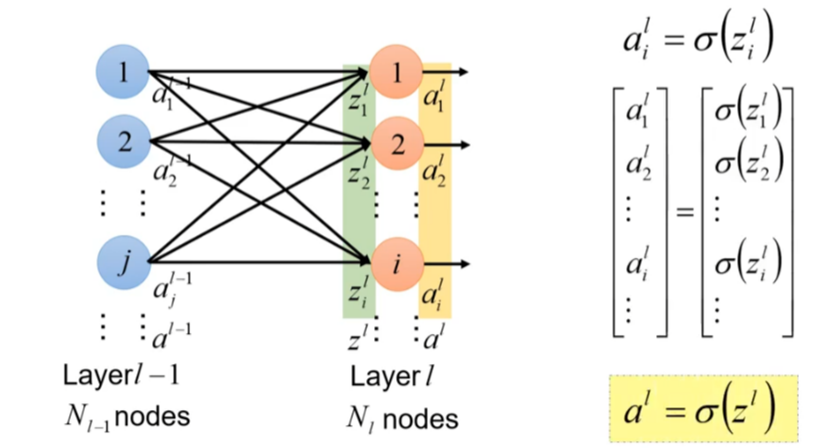

Output of a neuron:
layer l
neuron i
output of one layer → a vector
Layer l −1
N_{l-1} nodes
Layer l
N_l nodes
layer l −1
to layer l
from neuron j (layer l-1)
to neuron i (layer l)
weights between two layers
→ a matrix
: bias for neuron i at layer l
bias of all neurons at each layer
→ a vector
: input of the activation function
for neuron i at layer l
$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + ... + b_i^l$
$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$
activation function input at
each layer → a vector

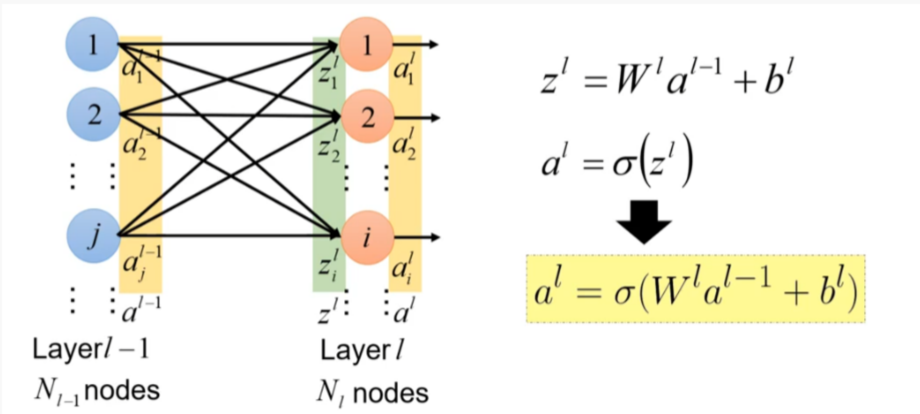
- 总览：

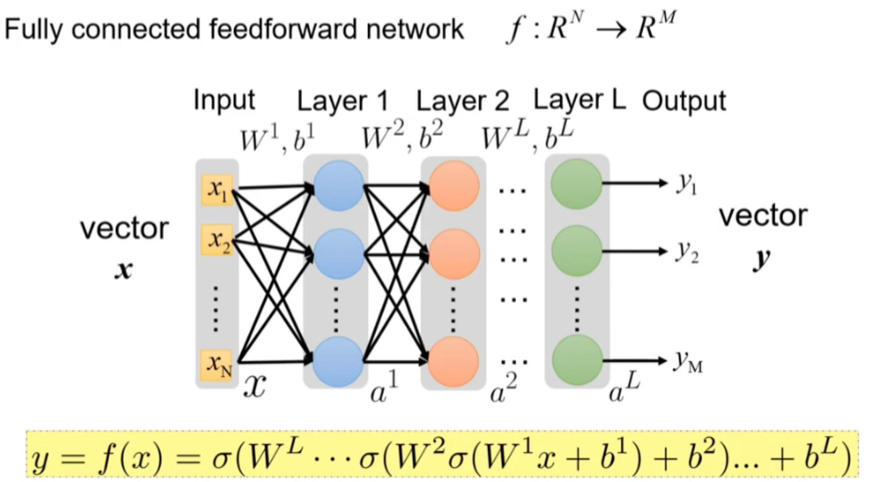
a_i^l : output of a neuron	w_{ij}^l : a weight
a^l : output vector of a layer	W^l : a weight matrix
z_i^l : input of activation function	b_i^l : a bias
z^l : input vector of activation function for a layer	b^l : a bias vector

- 符号之间的关系

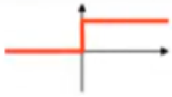
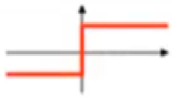
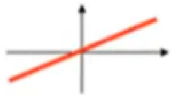
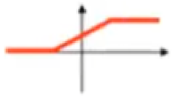
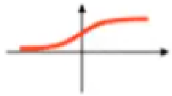
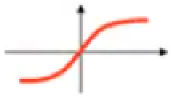

Layer $l-1$
N_{l-1} nodes
Layer l
N_l nodes
$z_1^l = w_{11}^1 a_1^{l-1} + w_{12}^1 a_2^{l-1} + \cdots + b_1^l$
$z_i^l = w_{i1}^1 a_1^{l-1} + w_{i2}^1 a_2^{l-1} + \cdots + b_i^l$
$z^l = W^l a^{l-1} + b^l$

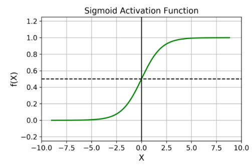
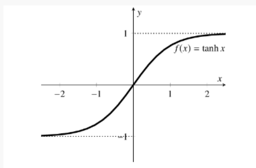
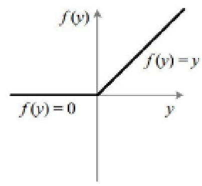
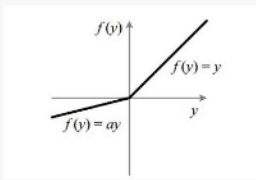
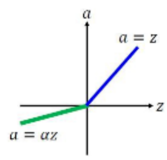

Layer $l-1$
N_{l-1} nodes
Layer l
N_l nodes
$a_i^l = \sigma(z_i^l)$
$a^l = \sigma(z^l)$


Layer $l-1$
N_{l-1} nodes
Layer l
N_l nodes
$z^l = W^l a^{l-1} + b^l$
$a^l = \sigma(z^l)$
$a^l = \sigma(W^l a^{l-1} + b^l)$


Fully connected feedforward network $f : R^N \to R^M$
Input
Layer 1
Layer 2
Layer L
Output
W^1, b^1
W^2, b^2
W^L, b^L
vector
$\boldsymbol{x}$
vector
$\boldsymbol{y}$
$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$

- 激活函数

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z)=\begin{cases}0, & z<0,\\ 0.5, & z=0,\\ 1, & z>0,\end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z)=\begin{cases}-1, & z<0,\\ 0, & z=0,\\ 1, & z>0,\end{cases}$	Perceptron variant	
Linear	$\phi(z)=z$	Adaline, linear regression	
Piece-wise linear	$\phi(z)=\begin{cases}1, & z\geq\frac{1}{2},\\ z+\frac{1}{2}, & -\frac{1}{2}<z<\frac{1}{2},\\ 0, & z\leq-\frac{1}{2},\end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)=\frac{1}{1+e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	Multi-layer NN	

函数名	函数原理（公式）	函数图像	常见应用
Sigmoid函数	$\sigma(x)=\frac{1}{1+e^{-x}}$		Logistic回归、简单的神经网络
Tanh函数	$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$		CNN、深度神经网络
ReLU	$\mathrm{ReLU}(x)=\max(0,x)$		CNN、深度神经网络
Leaky ReLU	$\text{Leaky ReLU}(x)=\begin{cases}x & \text{if } x\geq 0\\ \alpha x & \text{if } x<0\end{cases}$		CNN、深度神经网络
Parametric ReLU (PReLU)	$\mathrm{PReLU}(x)=\begin{cases}x & \text{if } x\geq 0\\ \alpha x & \text{if } x<0\end{cases}$		CNN、深度神经网

4.2 损失函数设计

1. **Function = Model Parameters**
   - 选择一个好的函数等同于选择一组模型参数；

2. **Model Parameter Measurement**
    - loss/cost/error function C(θ)

      好的模型参数意味着取min（C(θ)）
    - objective/reward function O(θ)

      好的模型参数意味着取max（O(θ)）
3. **常用的损失函数**

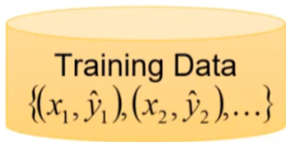
Square loss	$C(\theta)=(1-\hat{y}f(x;\theta))^2$
Hinge loss	$C(\theta)=\max(0,1-\hat{y}f(x;\theta))$
Logistic loss	$C(\theta)=-\hat{y}\log(f(x;\theta))$
Cross entropy loss	$C(\theta)=-\sum\hat{y}\log(f(x;\theta))$

Others: large margin, etc.

4.3 优化

1. **Gradient Descent**（梯度下降）
    - 在每次迭代中使用整个训练数据集来计算梯度，也即要看完所有的数据后再进行更新；
    - 当数据量很大时，如何有效的计算梯度下降：使用backpropagation；
    - **收敛速度与精度**：收敛速度**较慢**，但提供更准确的梯度估计，有助于找到全局最小值。
2. **Stochastic Gradient Descent**（SGD，随机梯度下降）
    - 在每次迭代中随机选择单个训练样本或样本的子集来计算梯度，随机抽取一个样本来进行更新，并且保证每个样本抽取到的概率是同等的；
    - **收敛速度与精度**：收敛速度**较快**，但梯度估计可能不够准确，有时可能导致在局部最小值附近波动。
    - epoch定义：所有的数据都被随机抽取一次称为一个epoch，类似于上述的梯度下降。

When running SGD, the model starts θ^0



pick x_1 $\quad \theta^1=\theta^0-\eta\nabla C_1(\theta^0)$

pick x_2 $\quad \theta^2=\theta^1-\eta\nabla C_2(\theta^1)$

⋮

pick x_k $\quad \theta^k=\theta^{k-1}-\eta\nabla C_k(\theta^{k-1})$

⋮

pick x_K $\quad \theta^K=\theta^{K-1}-\eta\nabla C_K(\theta^{K-1})$ → one epoch

see all training samples once

3. **Mini-Batch SGD**（小批量随机梯度下降）

	Batch Gradient Descent	Stochastic Gradient Descent	Mini-Batch SGD
参数量（每次迭代）	使用所有的样本	使用一个样本	使用b个样本（1<b<样本总量）
推导公式	$\theta^{i+1}=\theta^{i}-\eta\frac{1}{K}\sum_{k}\nabla C_k(\theta^{i})$	$\theta^{i+1}=\theta^{i}-\eta\nabla C_k(\theta^{i})$	$\theta^{i+1}=\theta^{i}-\eta\frac{1}{B}\sum_{x_k\in b}\nabla C_k(\theta^{i})$
训练速度	3rd	2nd	1th

tip：神经网络不能保证获得全局最小值。

4. **Practical Tips**
   - 从不同的起始点开始，再选择相对优秀的起始点；
   - 学习率不能过大或过小；
   - for mini-batch training：
     - 在每次epoch开始前，要重新打乱训练样本；
     - 每次使用一个固定数量的训练样本；
     - 调整学习率；
   - 预留一部分的训练集进行测试，因为训练集是已知结果的，可以很快反映出表现如何；
   - 调整训练过程：
     - 函数选择不够好；
     - 避免过拟合，解决方法：增加训练集数据等。

5 效率地计算大量参数（backpropagation）

5.1 Forward vs. Back Propagation

1. Forward Propagation

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 $C(\theta)$

1. Back Propagation

back-propagation

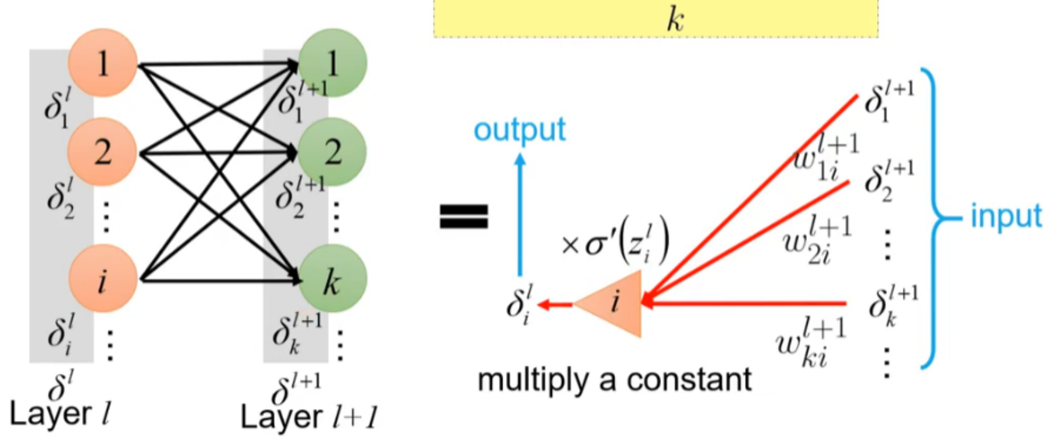
- allows the information from the cost to then flow backwards through the network, in order to compute the **gradient**

5.2 Back Propagation 推导

1. 推导示意图：

Rethink the propagation

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

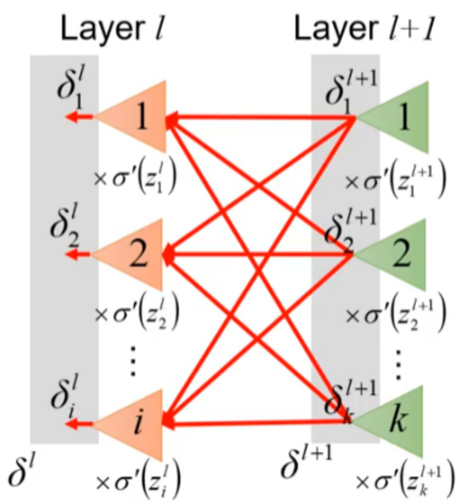


$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

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

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

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



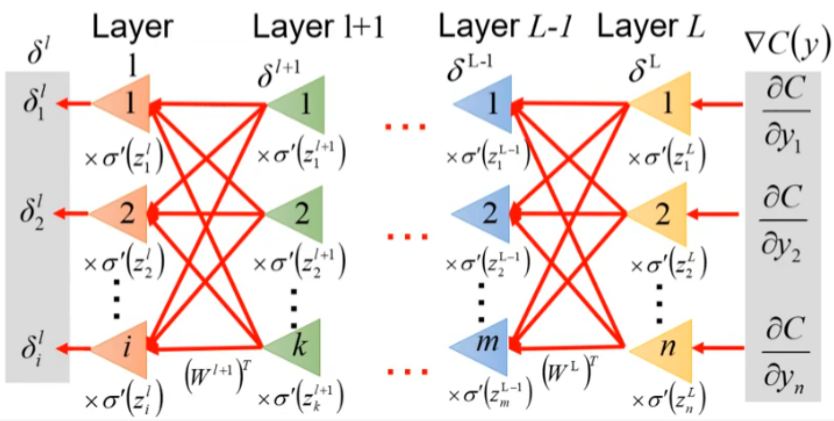
28 $\partial C(\theta)/\partial z_i^l = \delta_i^l$

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

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

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

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



2. BP算法最重要的两个步骤分别是Forward pass和Backward pass。BP算法的目的是求损失函数对权重/偏置参数的导数。

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

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

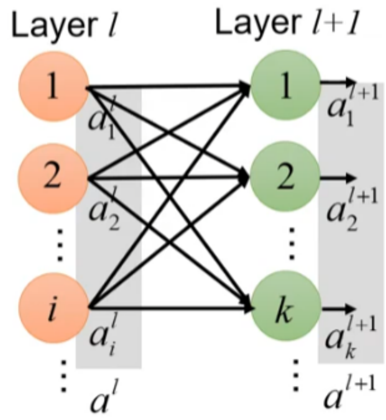
Forward Pass

$$z^1 = W^1 x + b^1 \qquad a^1 = \sigma(z^1)$$

$$\vdots$$

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

$$\vdots$$



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

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

Backward Pass

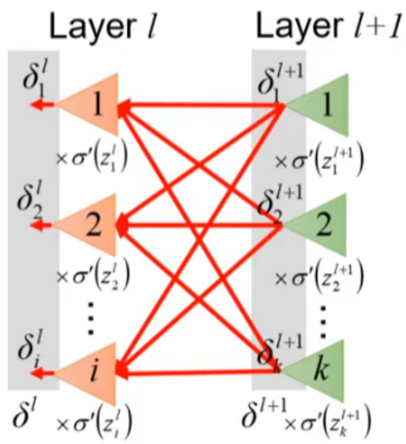
$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

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

$$\vdots$$

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

$$\vdots$$

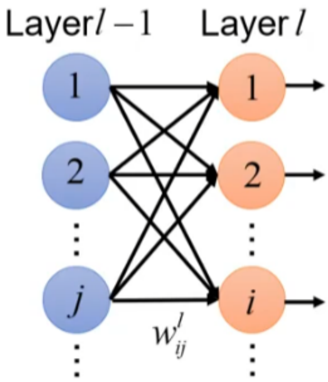


Concluding Remarks

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

$$\boxed{\delta_i^l}$$

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



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

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