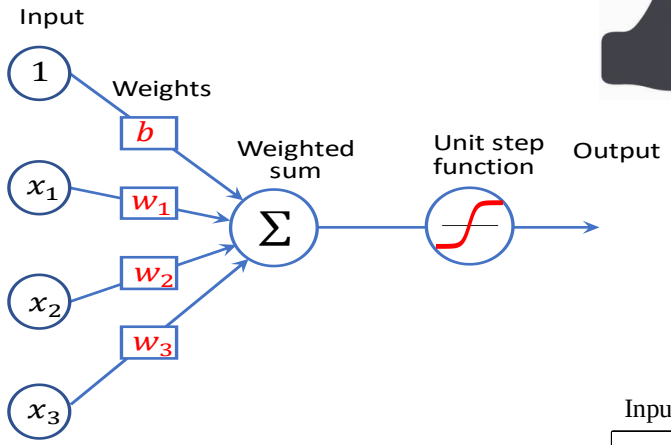
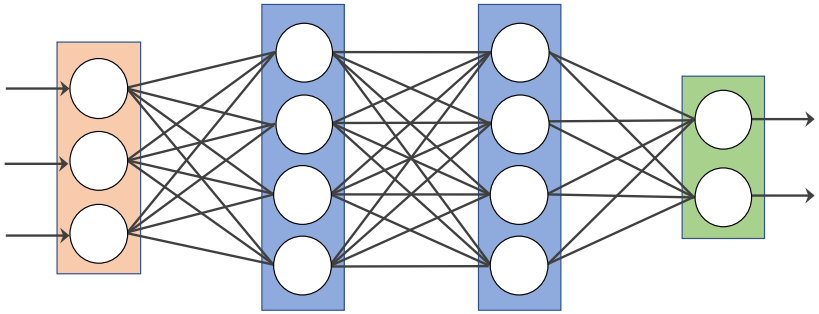
In this Course

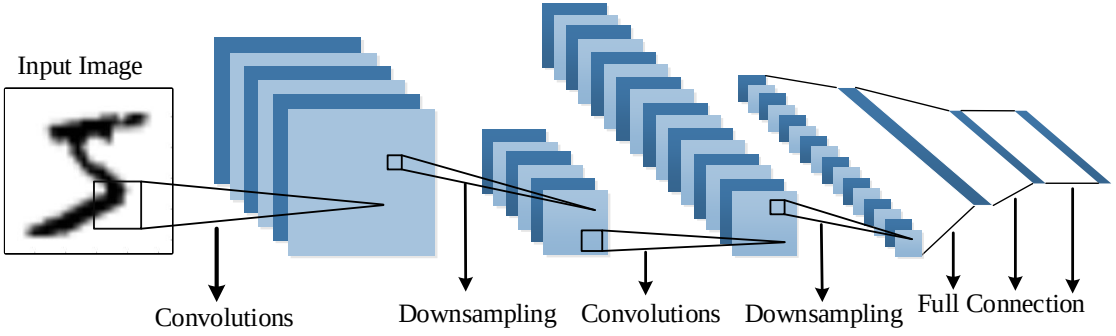
1. DL basics, linear regression, logistic regression etc.



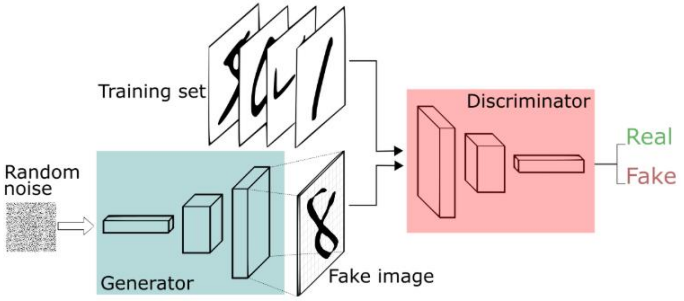
2. Multilayer neural networks, backpropagation



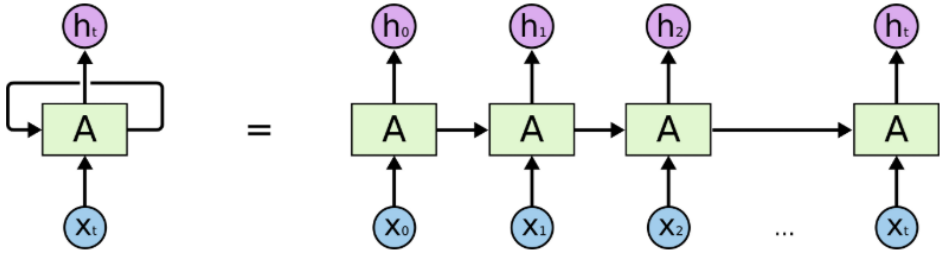
3. Convolutional Neural Networks and Applications



4. Generative Adversarial Networks



5. Recurrent networks and applications

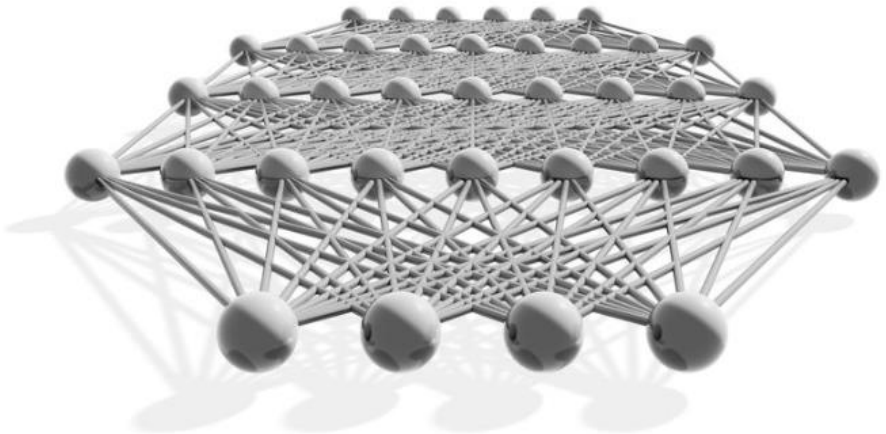


Last Lecture

- Tips for Gradient Descent
- Logistic Regression

Lecture 3

- Neural Networks
- Multilayer Neural Networks
- Backpropagation

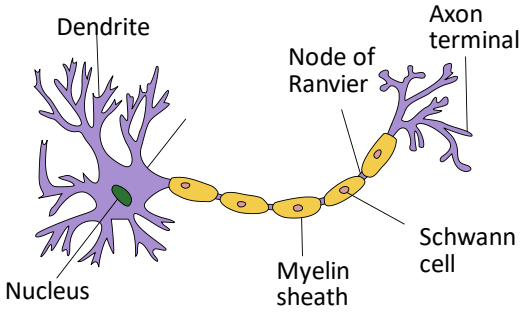


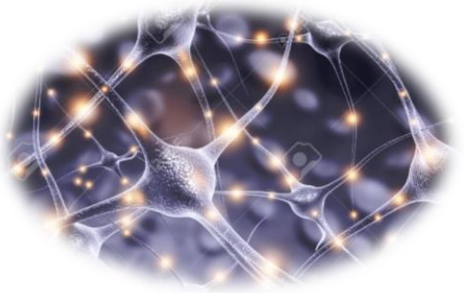
Many slides adapted from Hungyi Lee

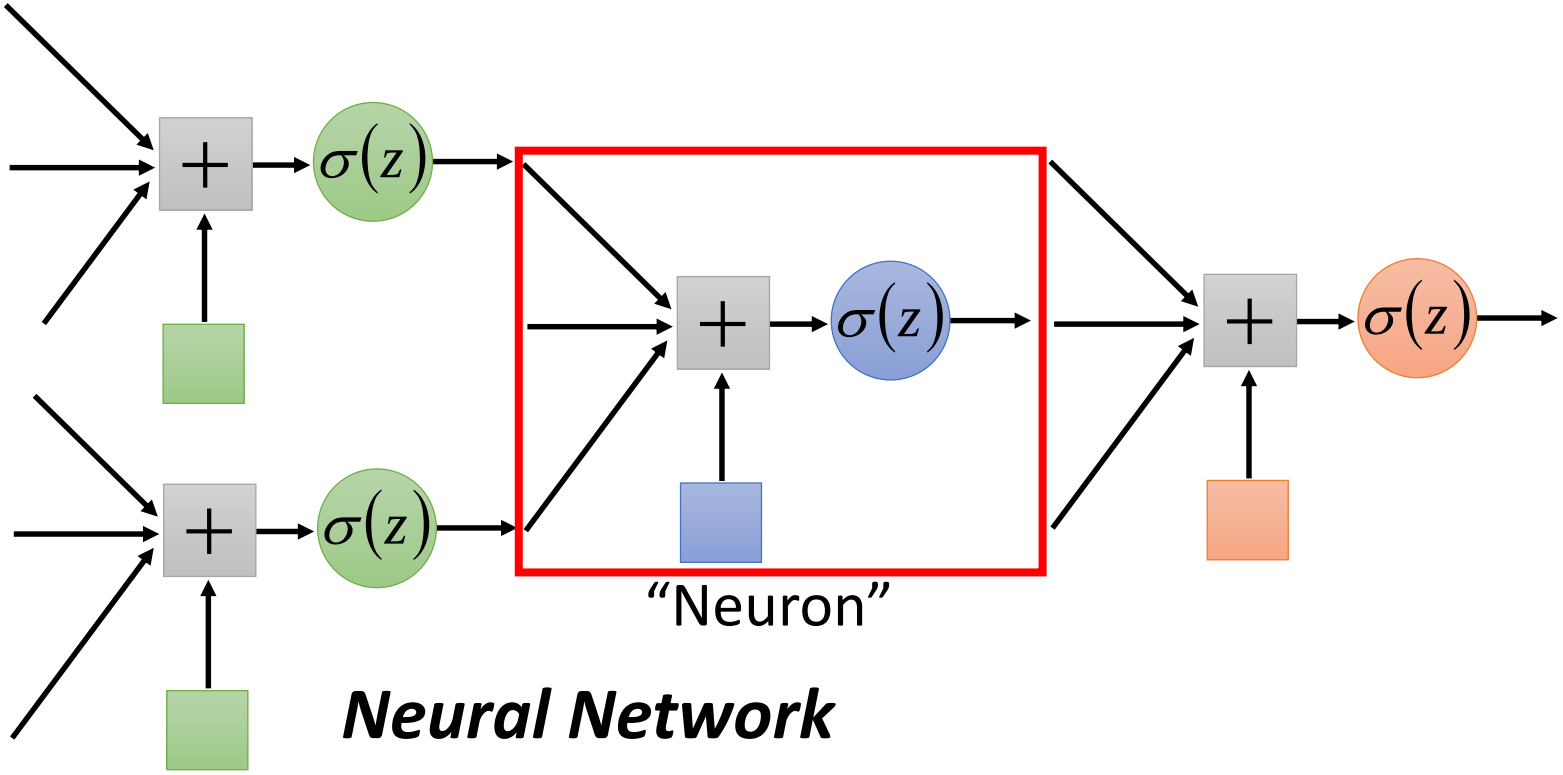
Three Steps for Deep Learning

Deep Learning is so simple. Don't be afraid……

Neural Network





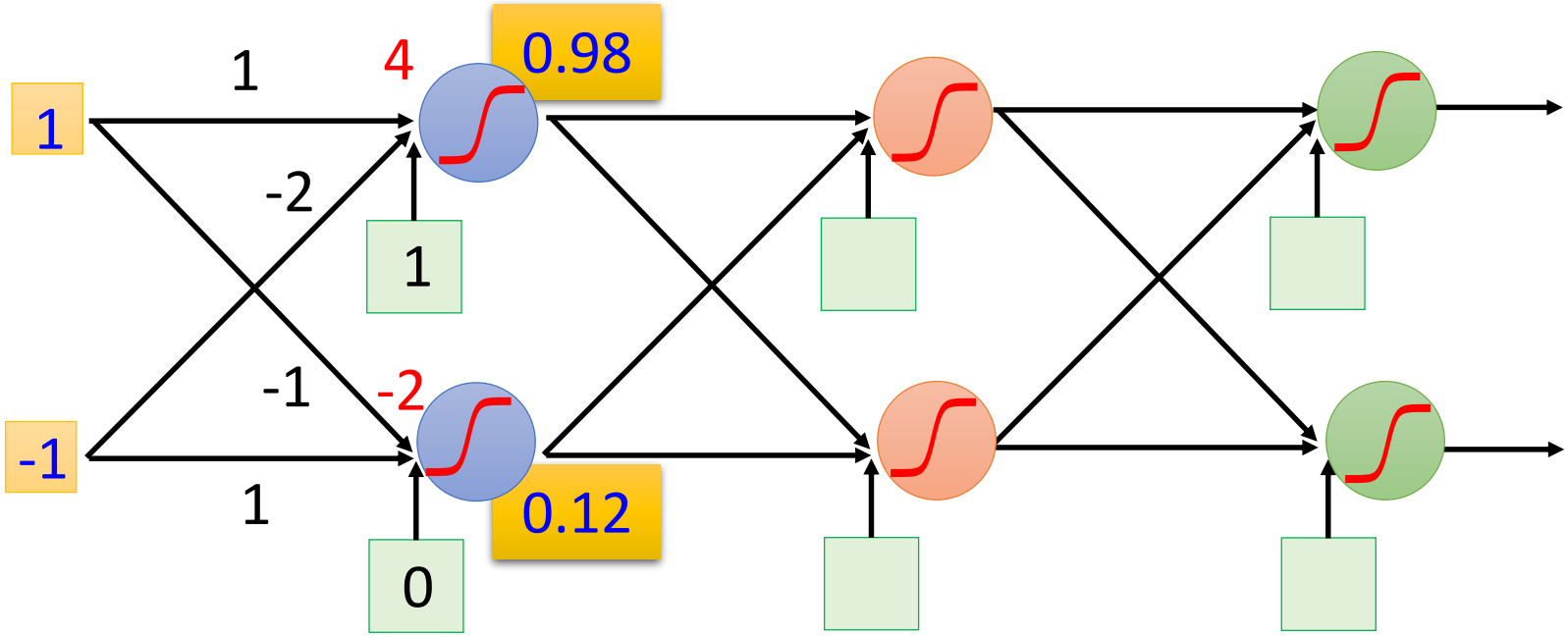


Neural Network

Different connection leads to different network structures

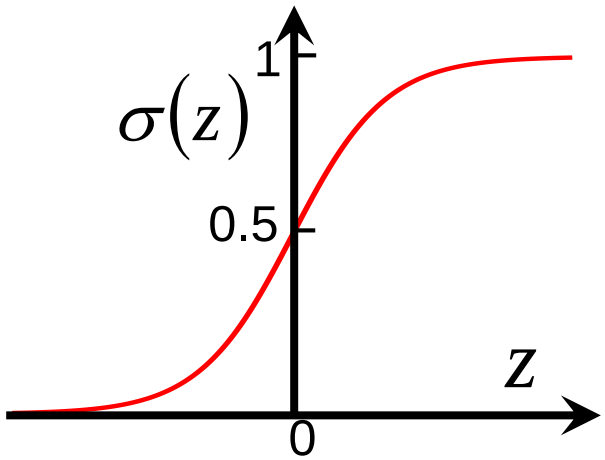
Network parameter θ : all the weights and biases in all the "neurons"

Fully Connect Feedforward Network

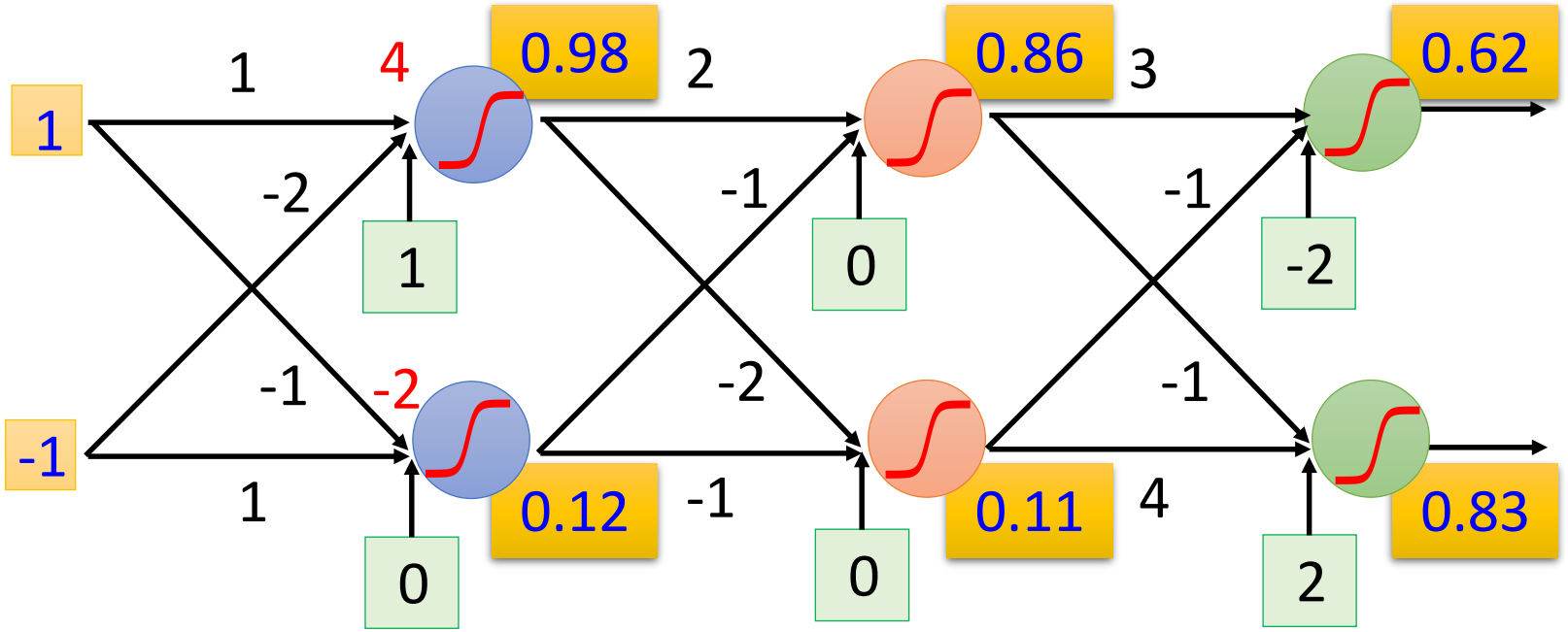


Sigmoid Function

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

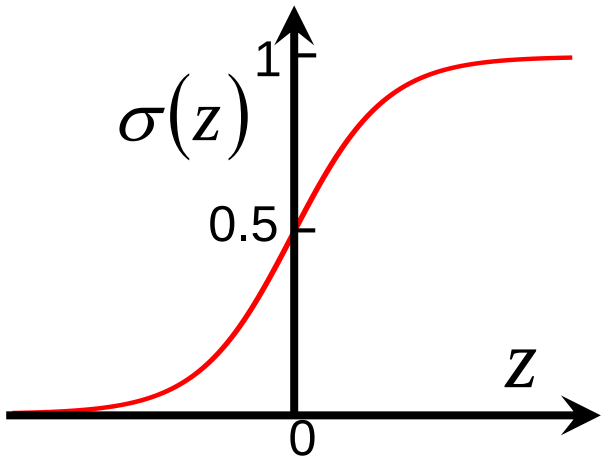


Fully Connect Feedforward Network

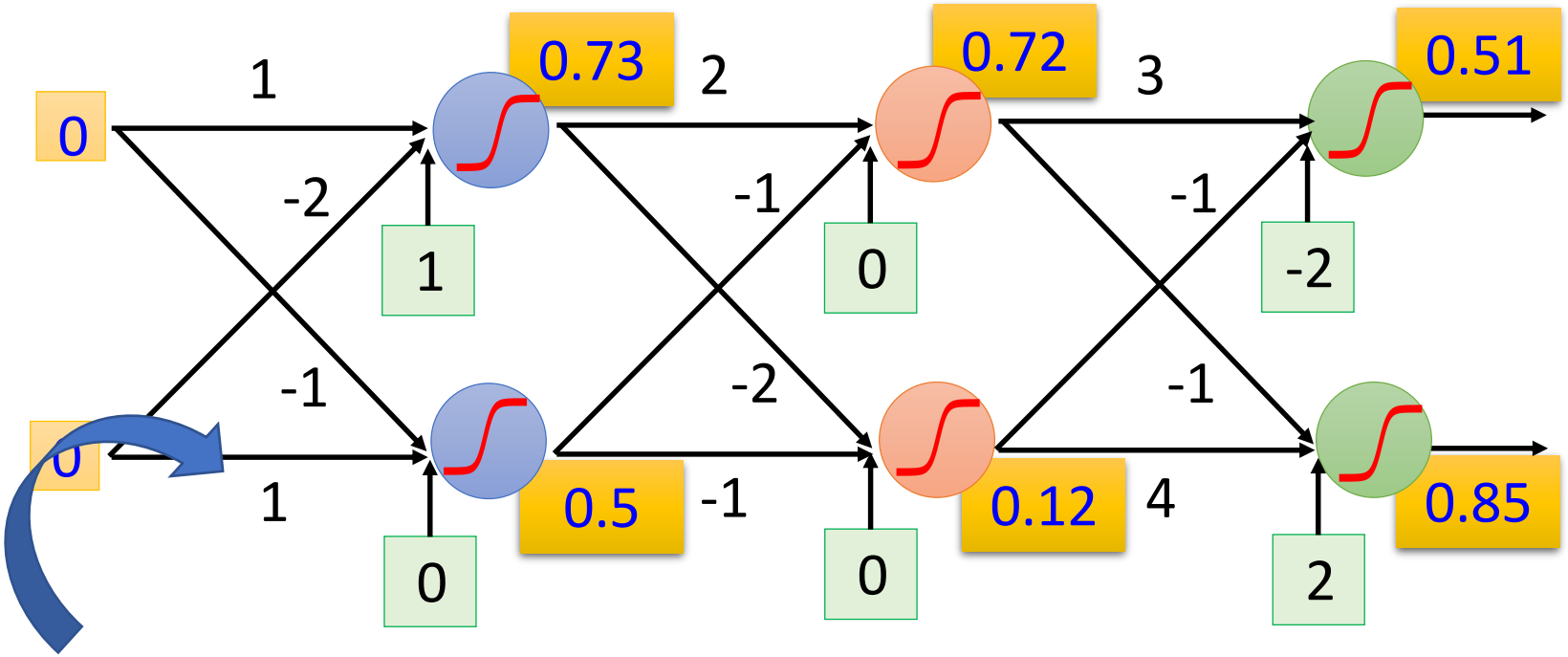


Sigmoid Function

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



Fully Connect Feedforward Network



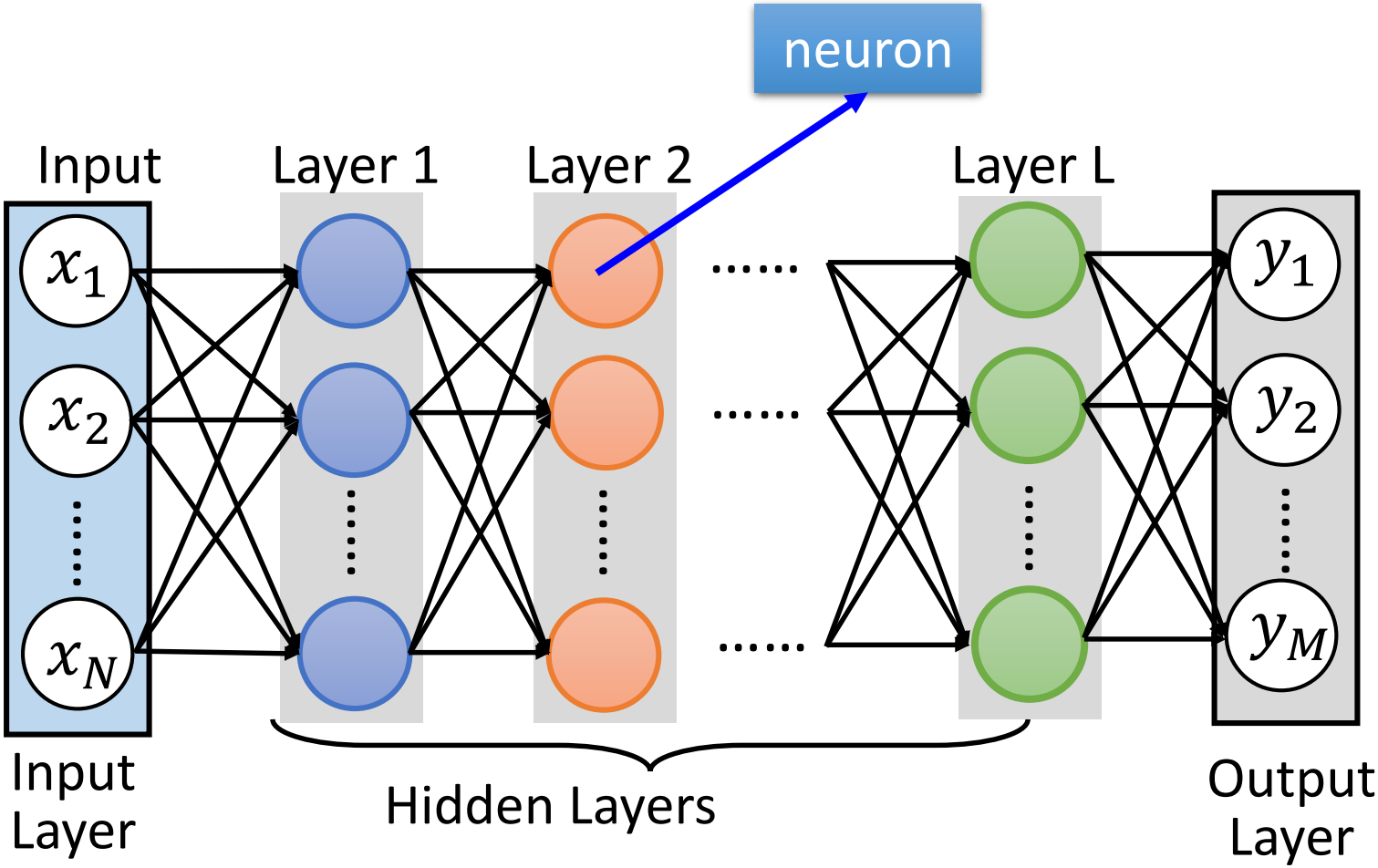
This is a function.

Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

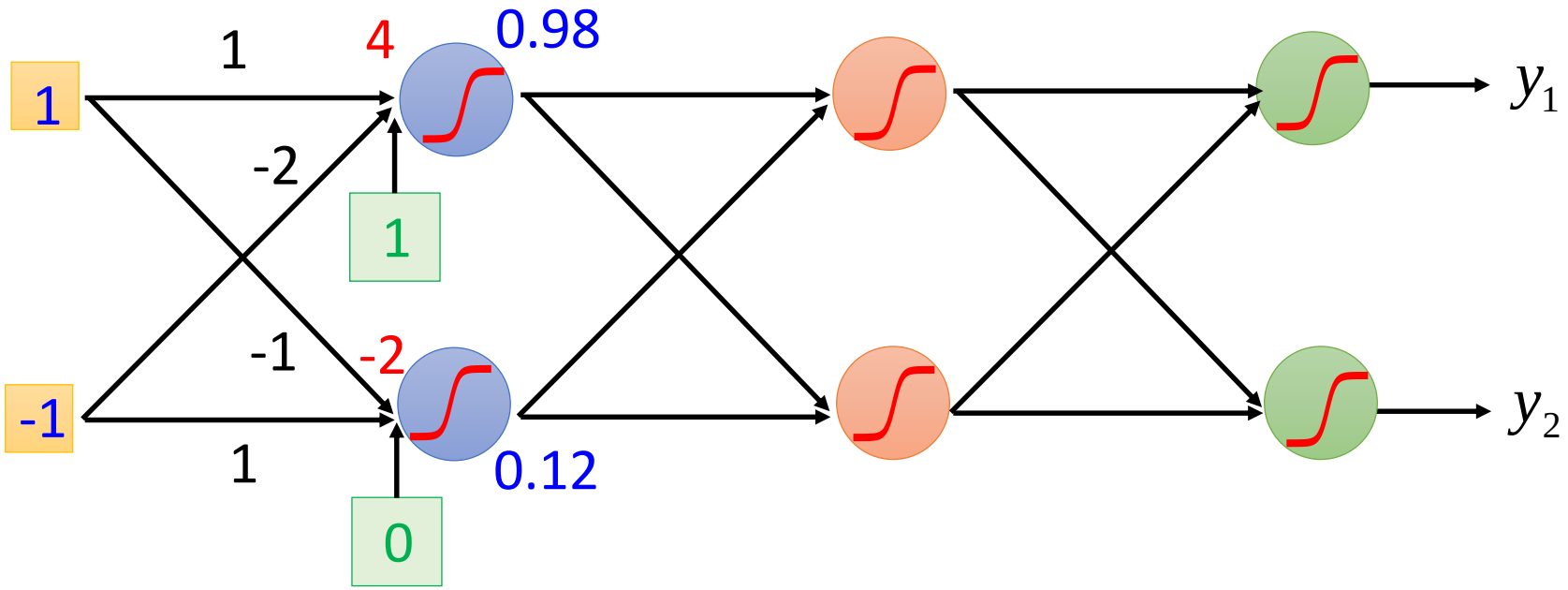
Given network structure, define ***a function set***

Fully Connect Feedforward Network



- Each hidden layer can have a different number of neurons.

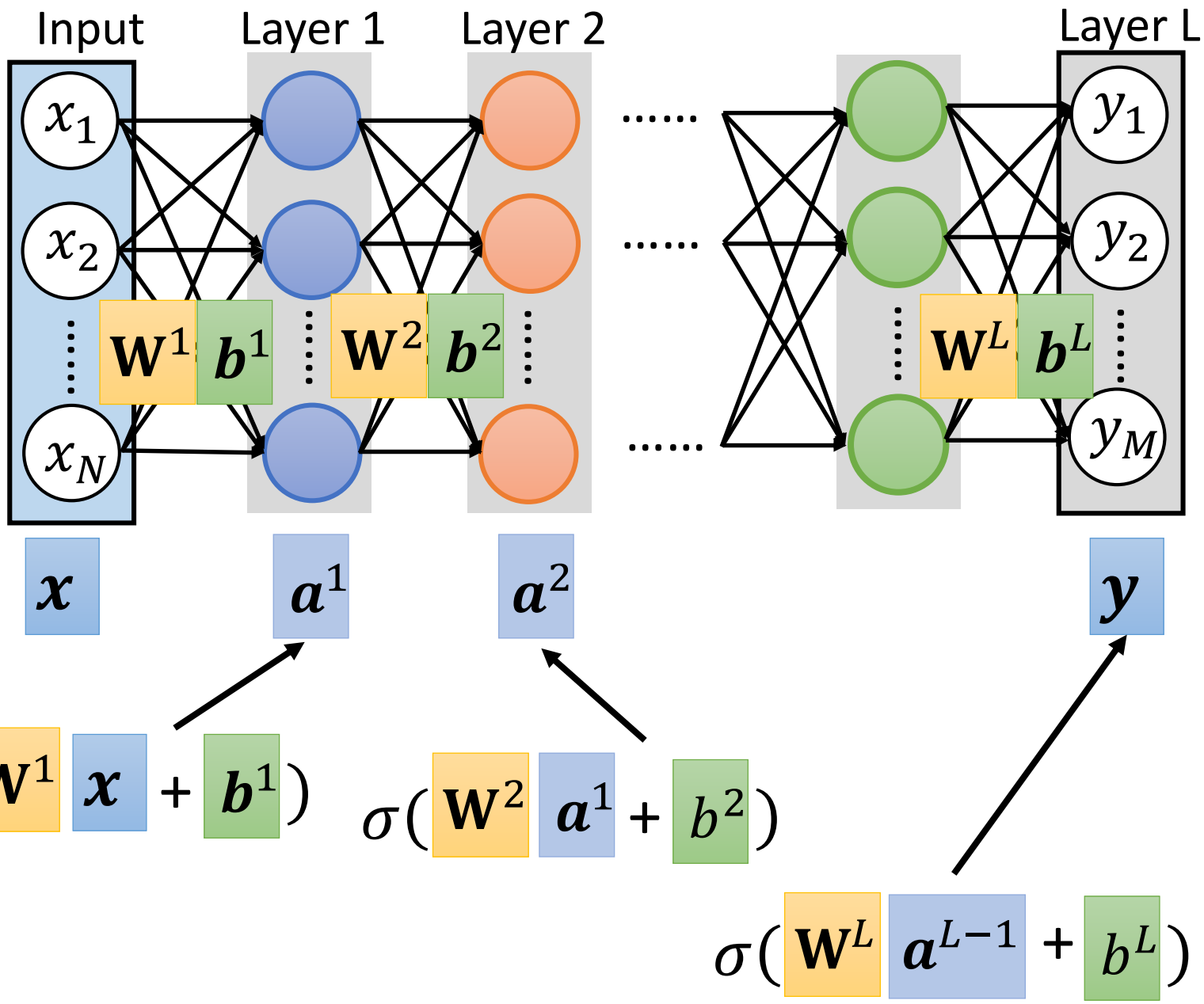
Matrix Operation



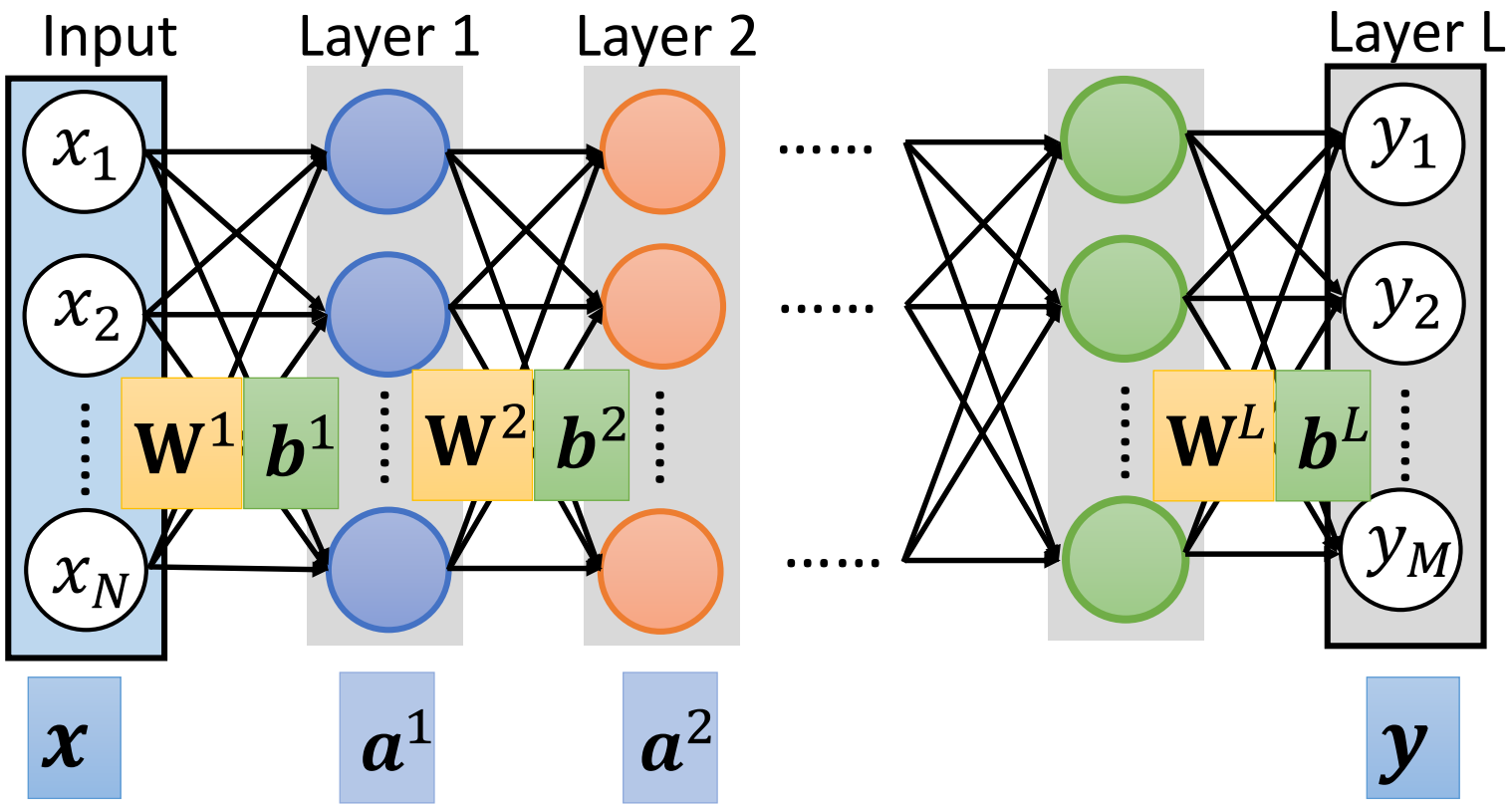
$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

$$\begin{bmatrix}4\\ -2\end{bmatrix}$$

Neural Network



Neural Network

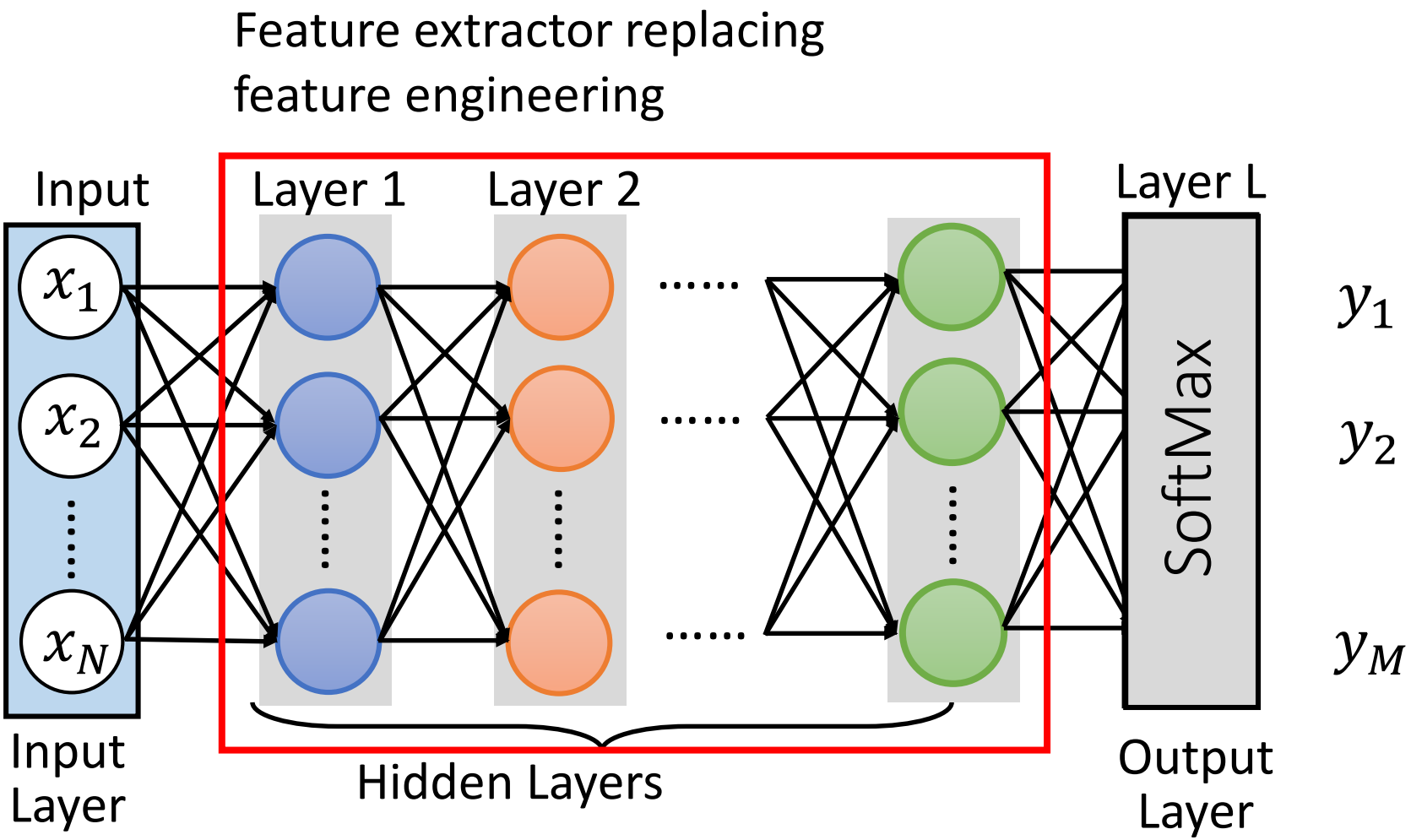


$y = f(x)$

Using parallel computing techniques to speed up matrix operation

$$\sigma(\mathbf{W}^L \cdots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 x + b^1) + b^2) \ldots + b^L)$$

Output Layer as MultiClass Classifier

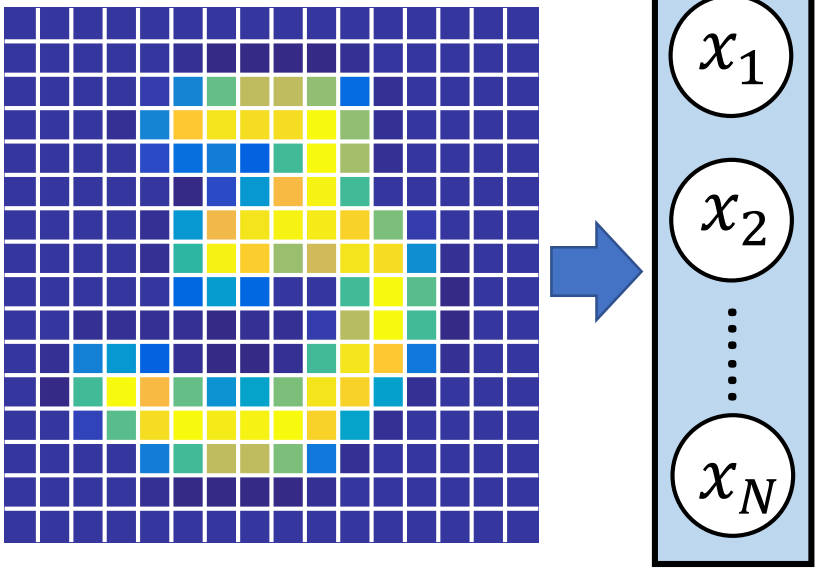


= Multiclass Classifier

Example Application

Handwriting Digit Recognition

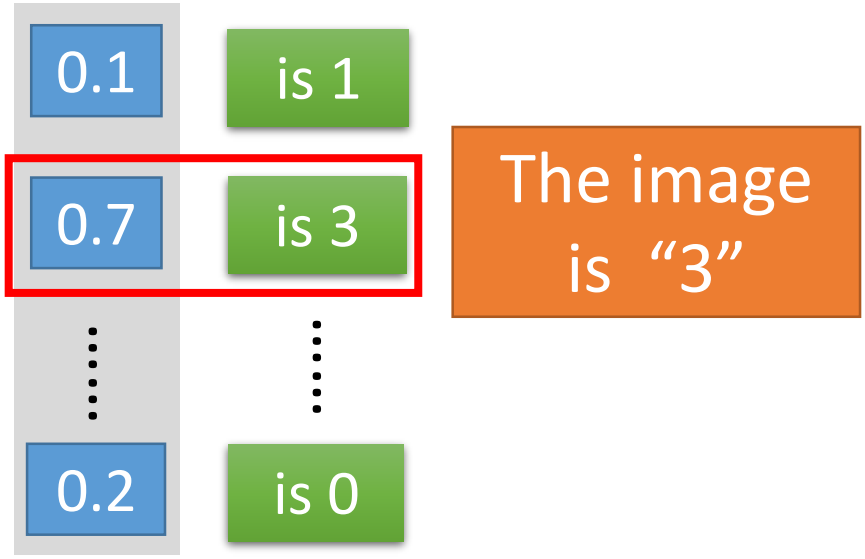
Input (Image Data)



16 x 16 = 256

N=256

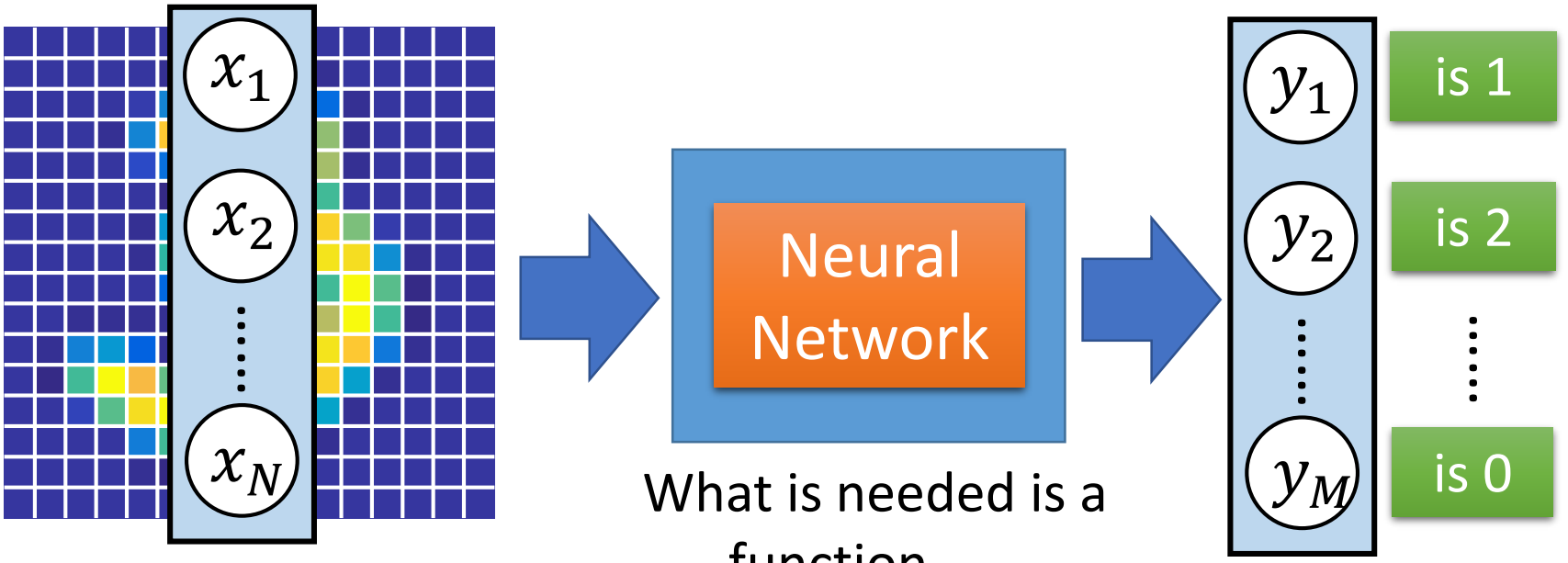
Output (Dim Fixed)



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition

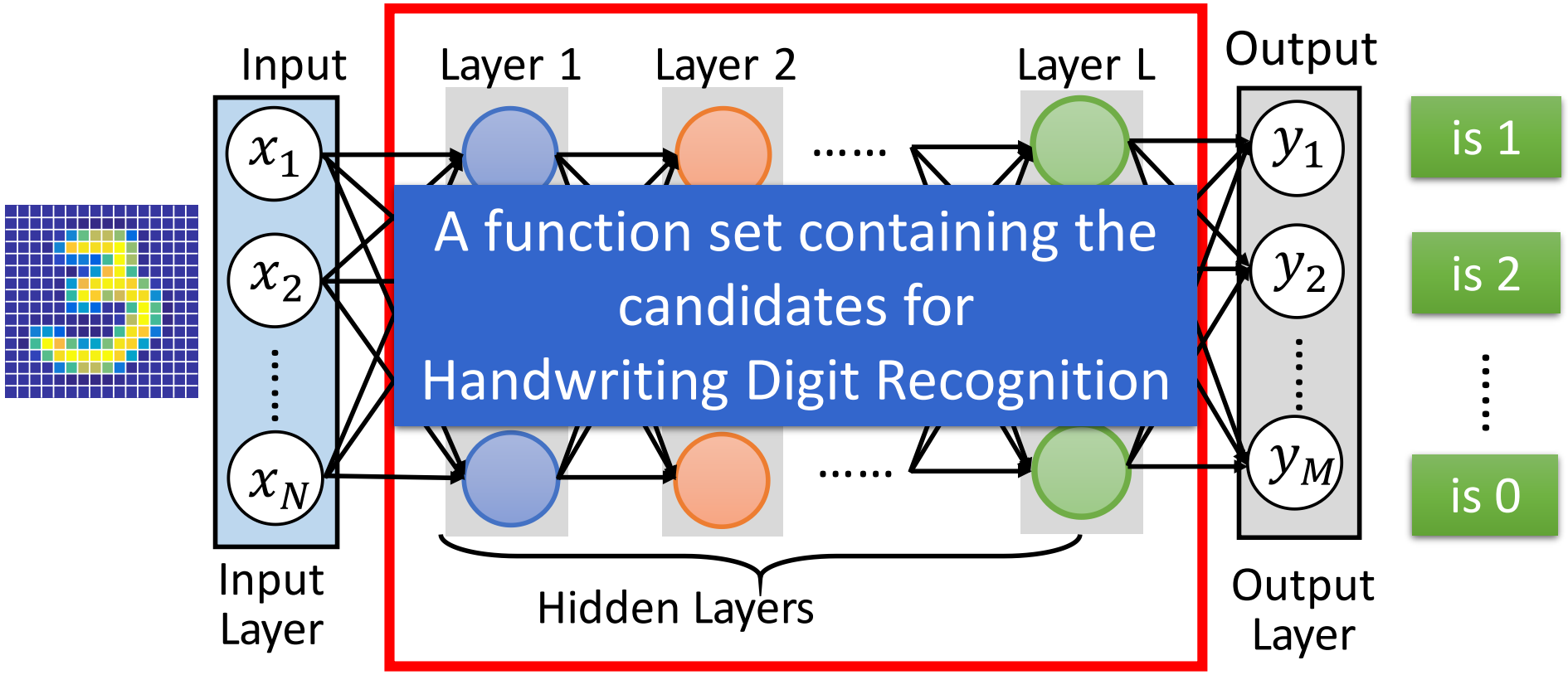


Input: Vector of 256 dimension
N=256

What is needed is a function ……

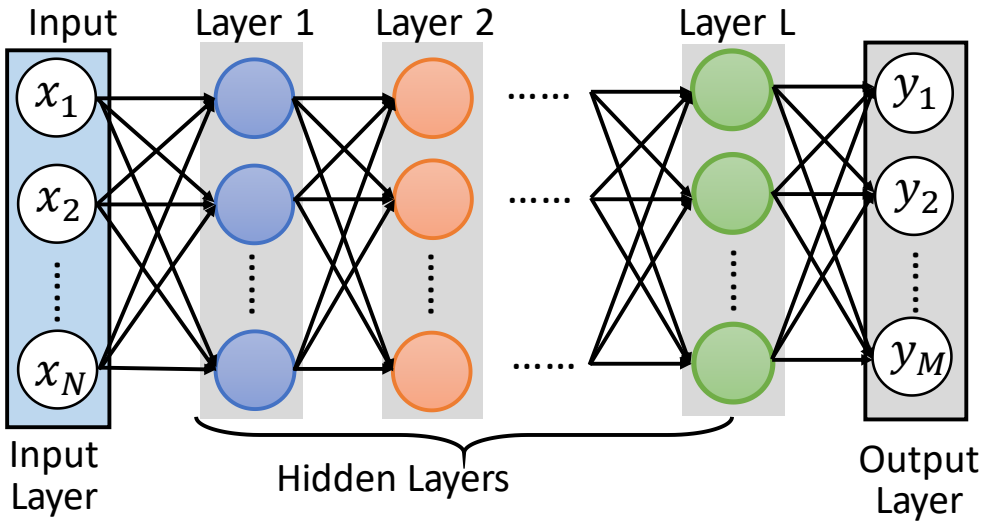
Output: Vector of 10 dimension
M=10 classes

Example Application



You need to decide the network structure to let a good function in your function set.

FAQ



- Q1: How many layers? How many neurons for each layer?

Trial and Error + Intuition

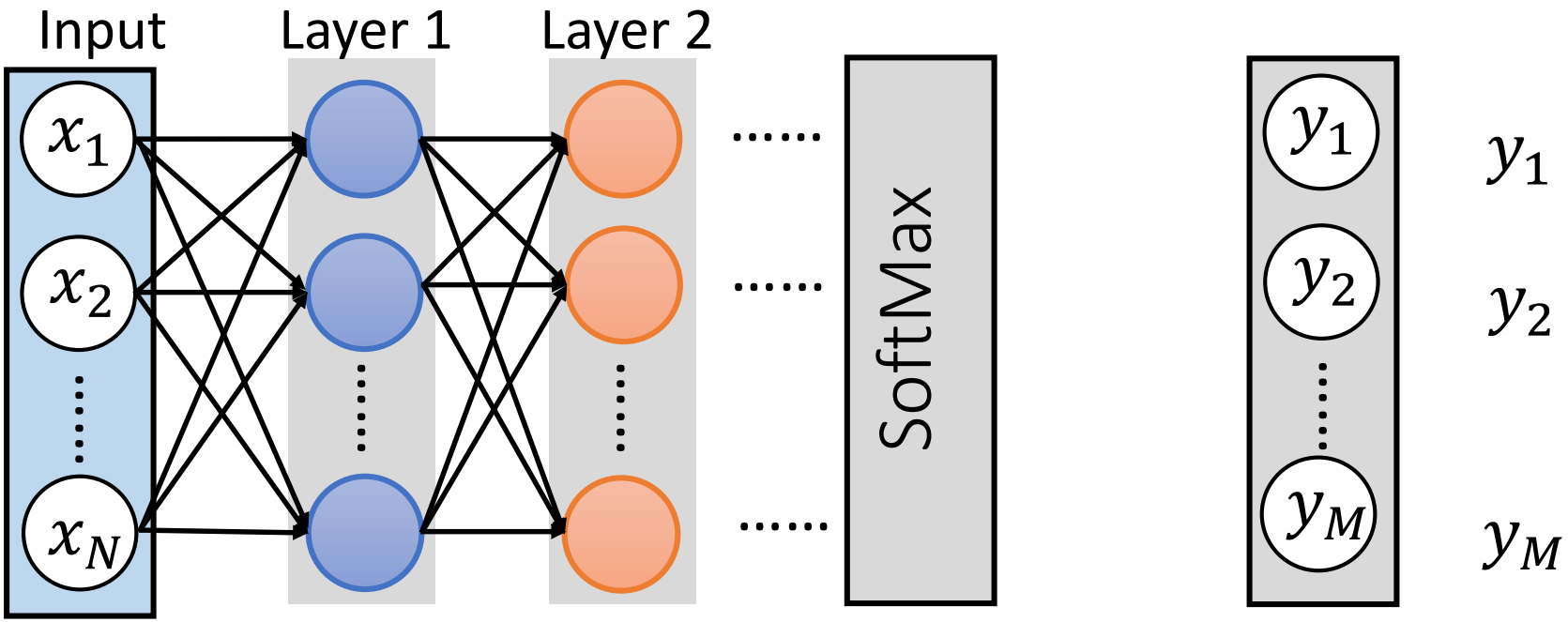
- Q2: Can the structure be automatically determined?
  - *e.g.* automatic Network Architecture Search (NAS)
- Q3: Can we design the network structure?

Convolutional Neural Network (CNN)

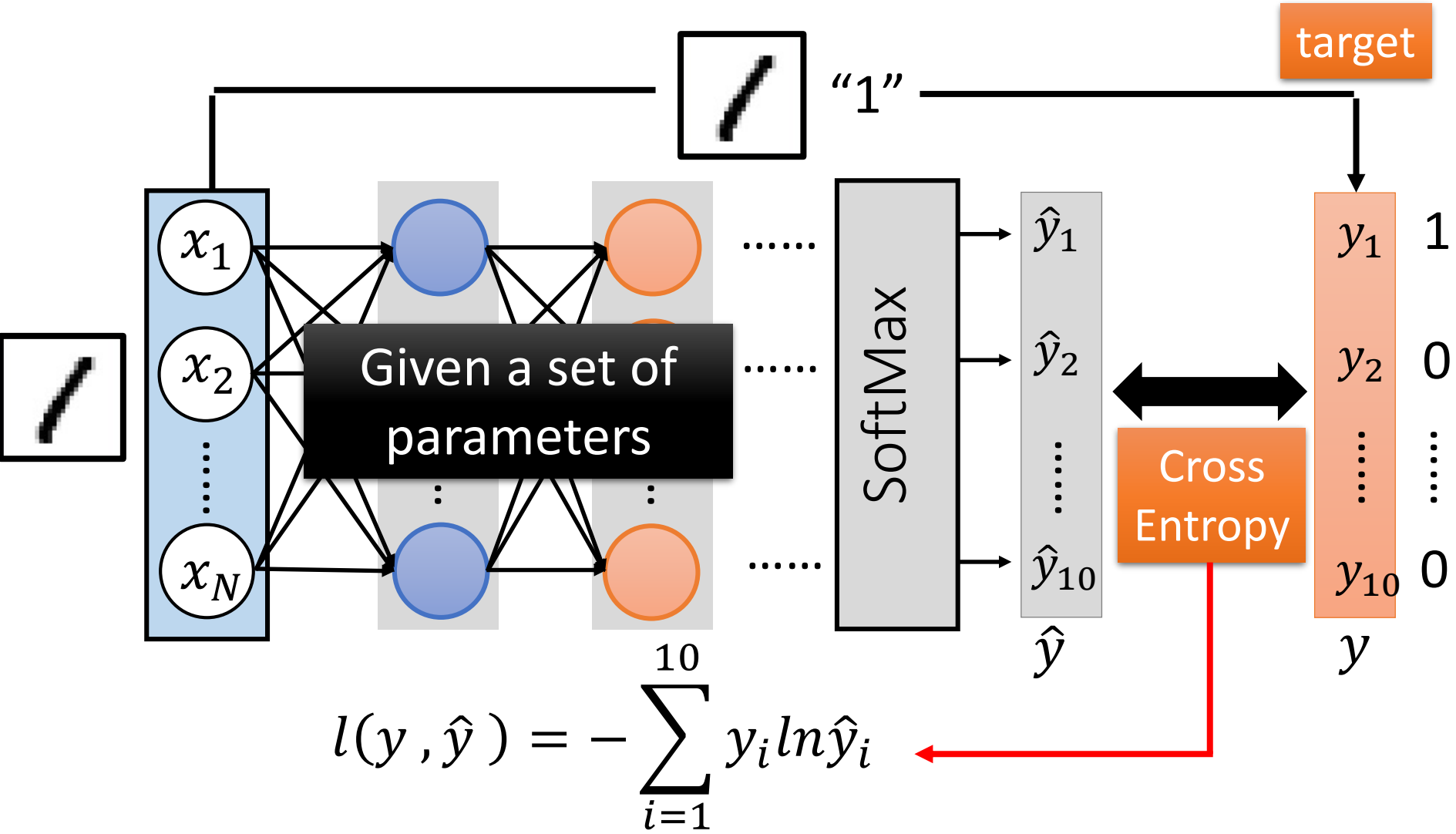
Three Steps for Deep Learning

Deep Learning is so simple. Don't be afraid……

Output Layer as MultiClass Classifier

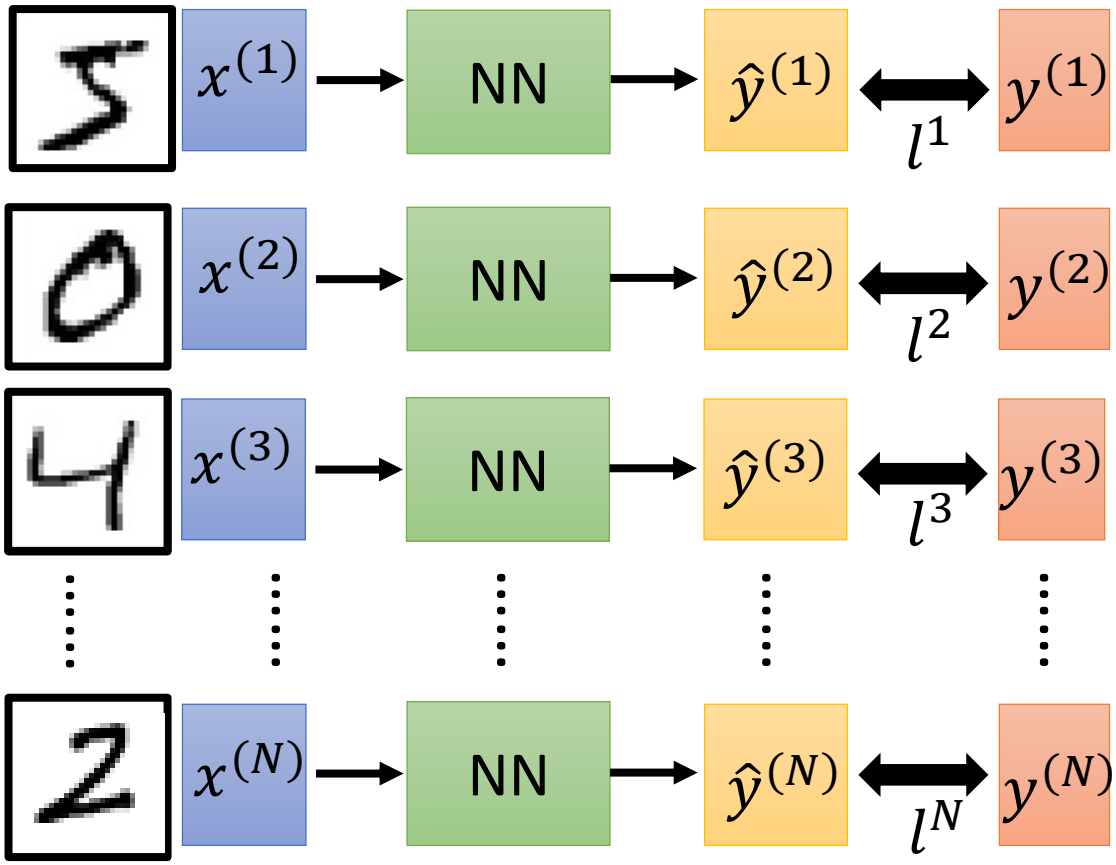


Loss for an Example



Total Loss

For all training data …



Total Loss:

$$L = \sum_{n=1}^{N} l^n$$

Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters*** θ^* that minimize total loss L

Three Steps for Deep Learning

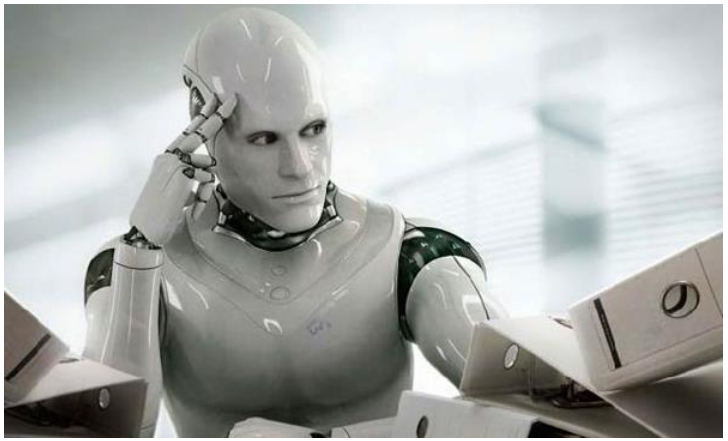
Deep Learning is so simple. Don’t be afraid……

Gradient Descent

This is the “learning” of machines in deep learning ……

➡ Even alpha go using this approach.

People image ……

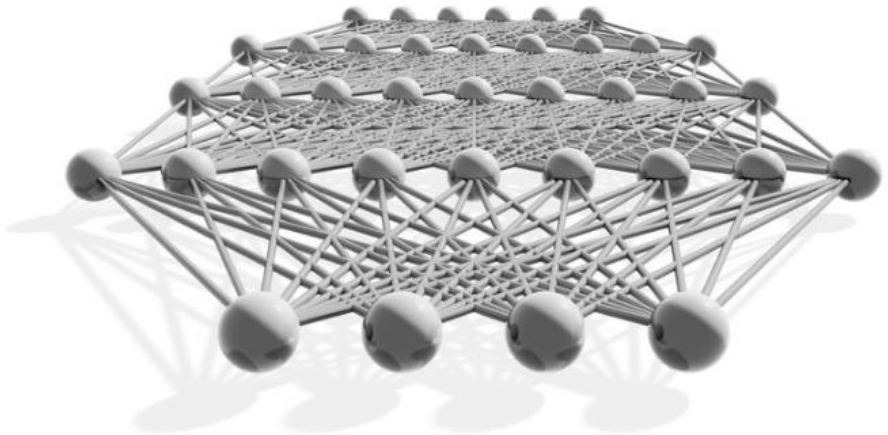


Actually …..

I hope you are not too disappointed.

Lecture 3

- Neural Networks
- Multilayer Neural Networks
- Backpropagation



Backpropagation for Fully Connect Feedforward Network

Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Starting Parameters $\theta^{(0)} \longrightarrow \theta^{(1)} \longrightarrow \theta^{(2)} \longrightarrow \cdots\cdots$

$$\nabla C(\theta) = \begin{bmatrix} \partial C(\theta)/\partial w_1 \\ \partial C(\theta)/\partial w_2 \\ \vdots \\ \partial C(\theta)/\partial b_1 \\ \partial C(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

$Compute\ \nabla C(\theta^{(0)})$ $\quad \theta^{(1)} = \theta^{(0)} - \eta \nabla C(\theta^{(0)})$

$Compute\ \nabla C(\theta^{(1)})$ $\quad \theta^{(2)} = \theta^{(1)} - \eta \nabla C(\theta^{(1)})$

Millions of parameters ……

To compute the gradients efficiently, we use ***backpropagation***.

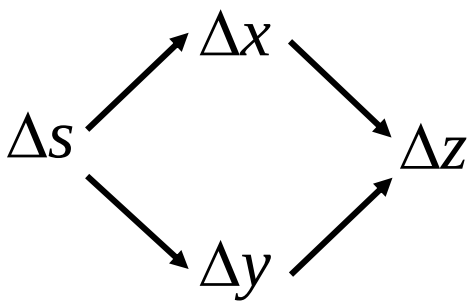
Chain Rule

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

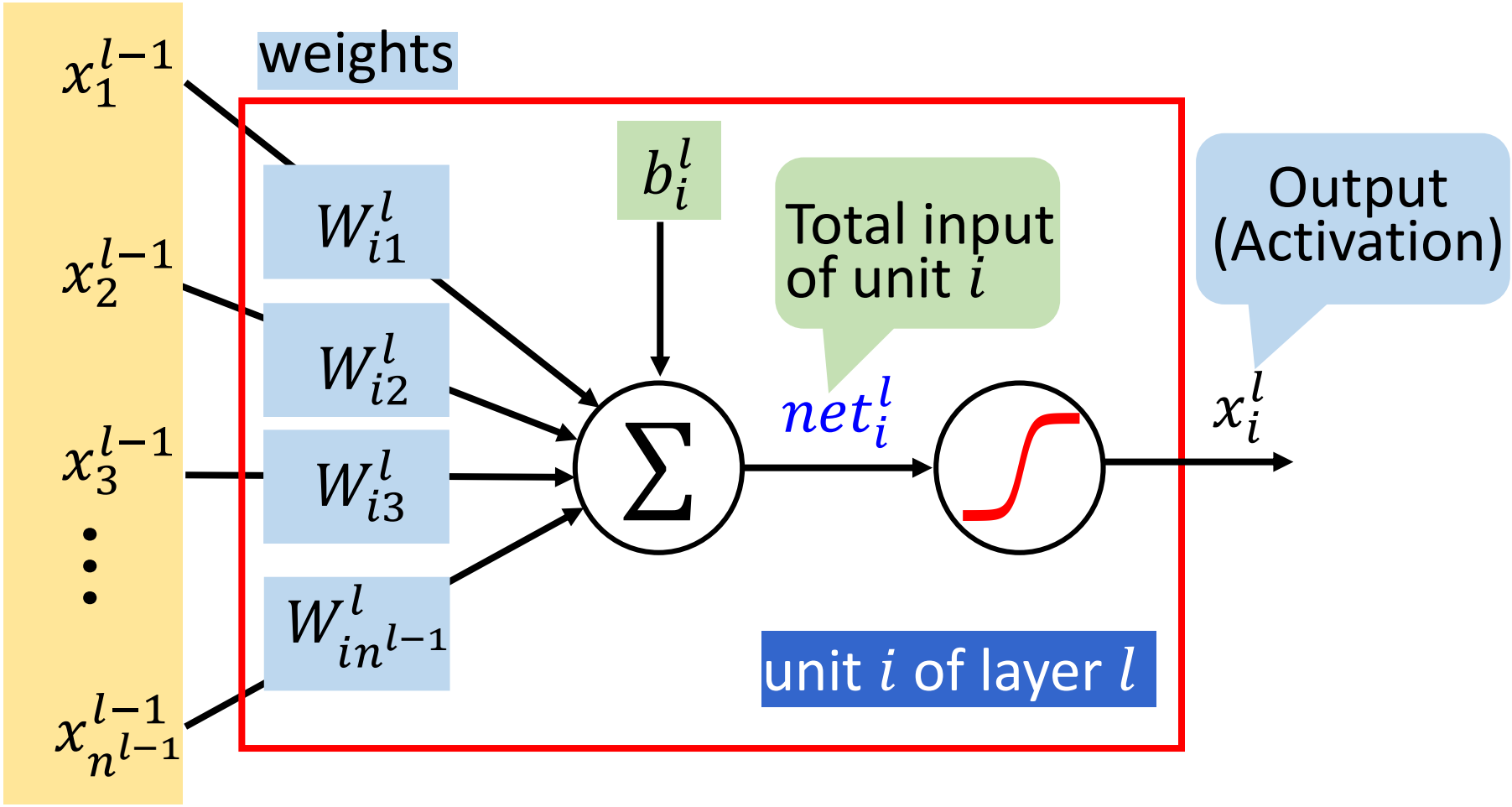
Case 2

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



$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

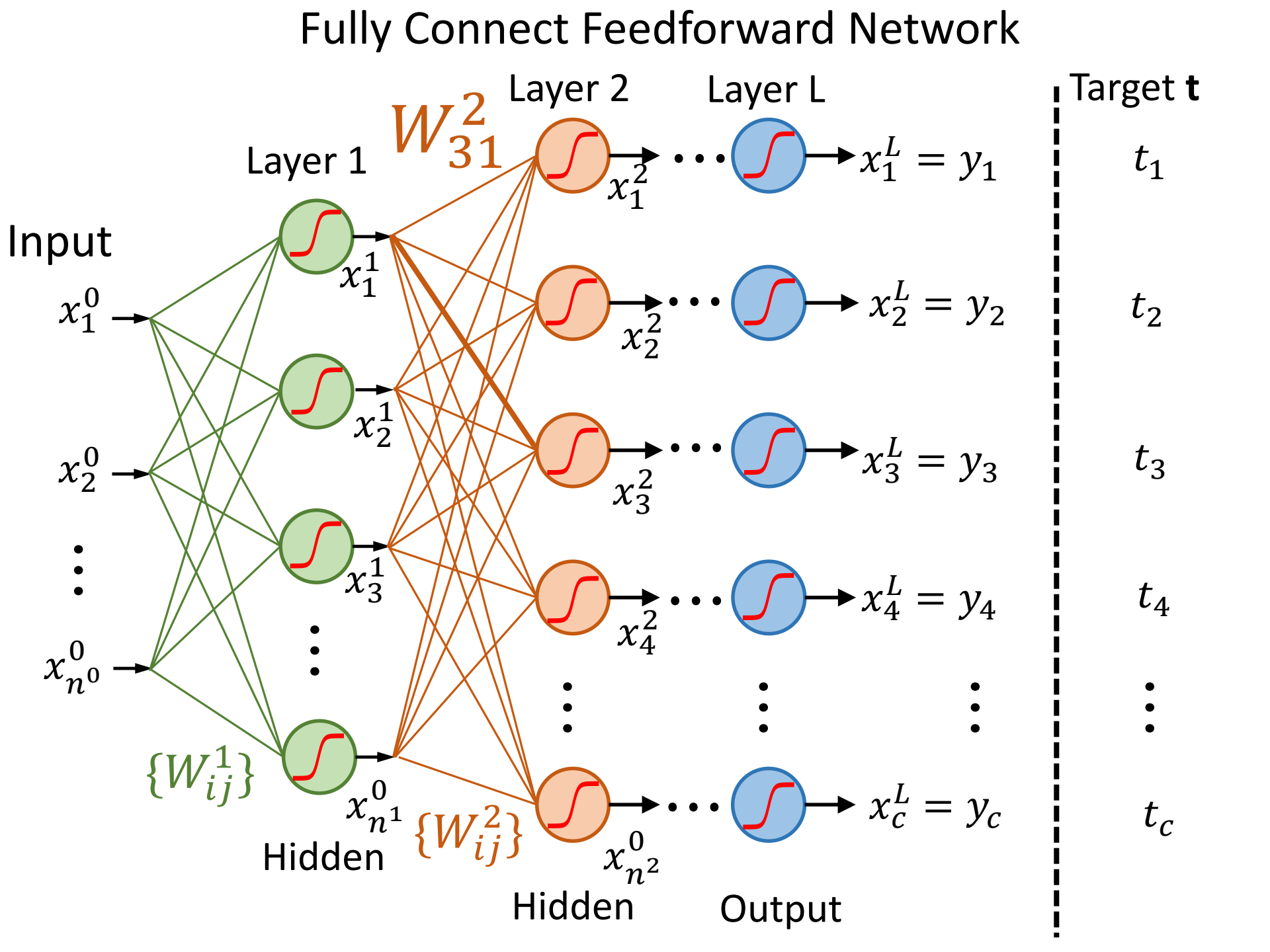
Recall: An Artificial Neuron



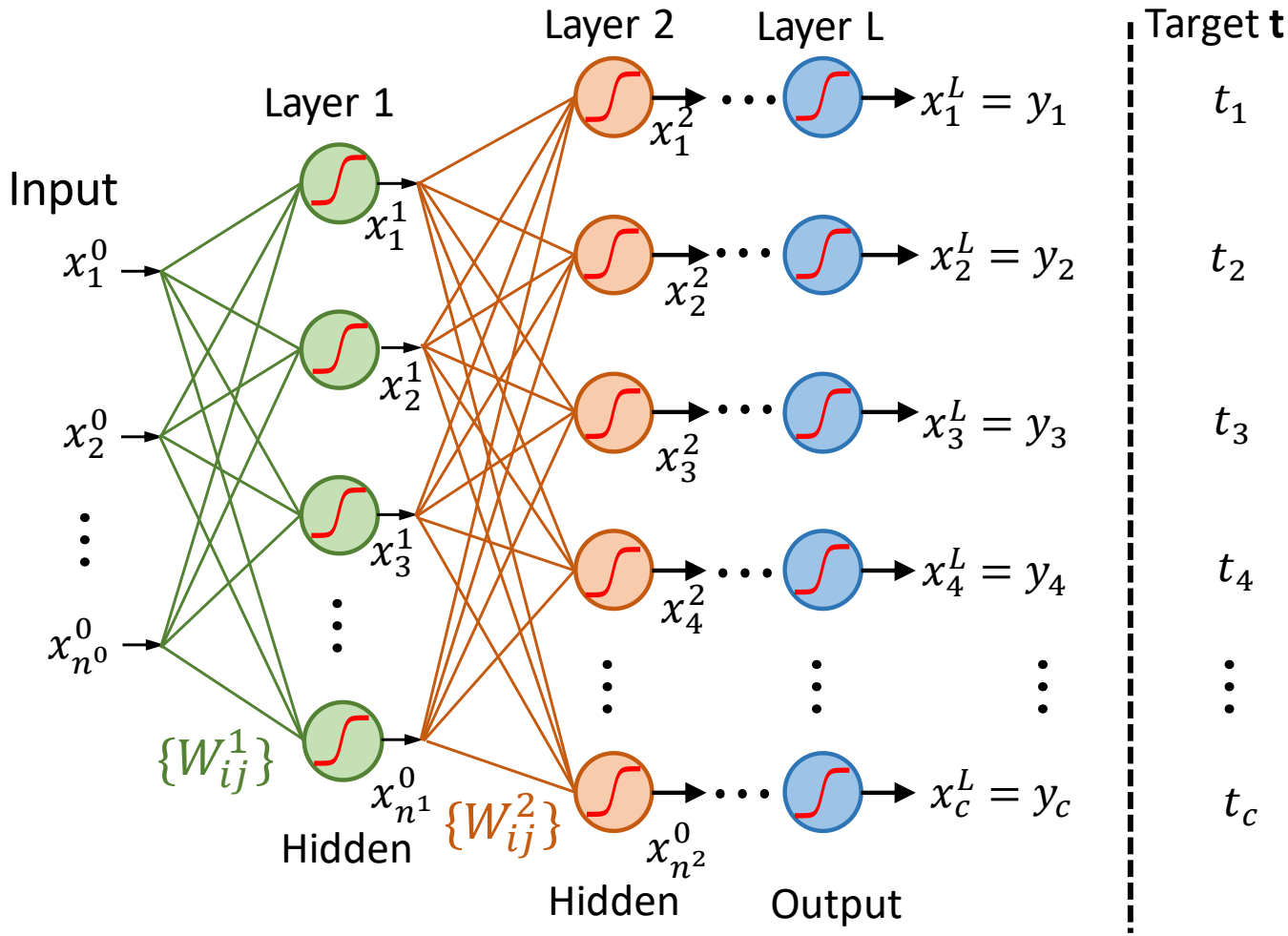
n^{l-1} activations from previous layer as input

Total input of unit i : $net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l \, x_j^{l-1} + b_i^l$

Output of unit i : $x_i^l = \sigma\left(net_i^l\right)$


Fully Connect Feedforward Network
Input
Layer 1
Layer 2
Layer L
Target **t**
W_{31}^2
x_1^0
x_2^0
$x_{n^0}^0$
x_1^1
x_2^1
x_3^1
$x_{n^1}^0$
$\{W_{ij}^1\}$
Hidden
$\{W_{ij}^2\}$
x_1^2
x_2^2
x_3^2
x_4^2
$x_{n^2}^0$
Hidden
$x_1^L = y_1$
$x_2^L = y_2$
$x_3^L = y_3$
$x_4^L = y_4$
$x_c^L = y_c$
Output
t_1
t_2
t_3
t_4
t_c

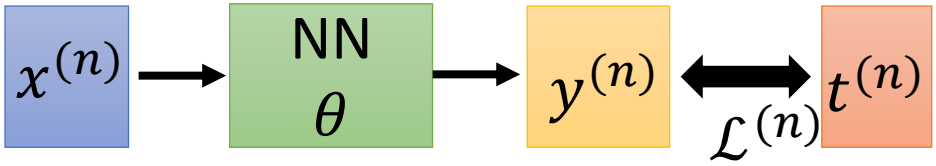
Fully Connect Feedforward Network



Total input of unit i : $net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l \, x_j^{l-1} + b_i^l$

Output of unit i : $x_i^l = \sigma\left(net_i^l\right)$

Backpropagation

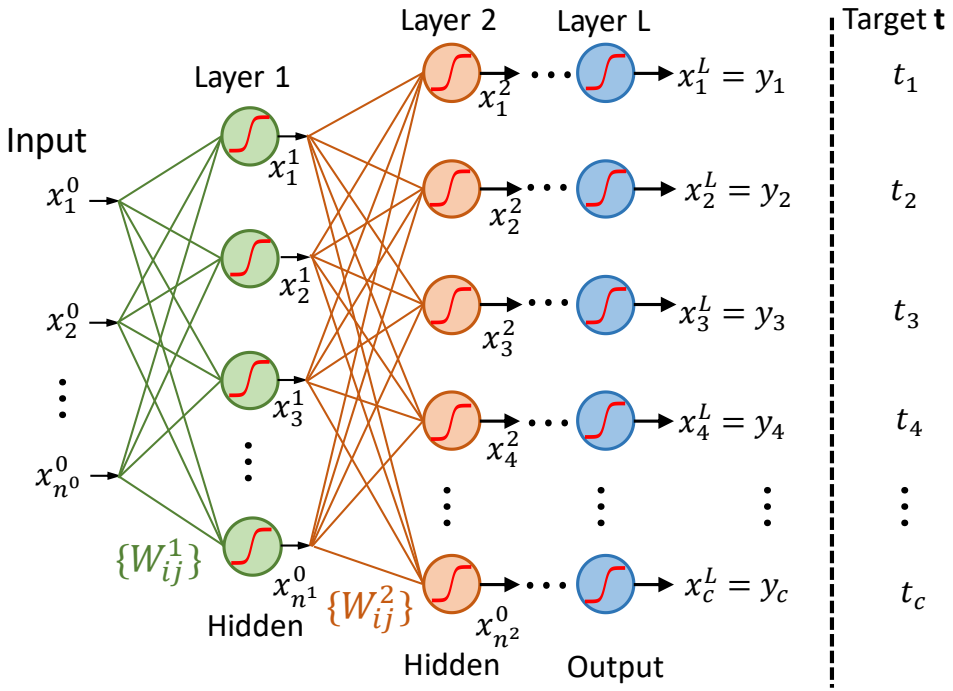


$$\mathcal{C}(\theta) = \sum_{n=1}^{N} \mathcal{L}^{(n)}(\theta) \quad \Rightarrow \quad \frac{\partial \mathcal{C}(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}^{(n)}(\theta)}{\partial w}$$

- Choose squared error as training error measurement and sigmoid as activation function at the output layer

$$\mathcal{L}(\theta) = \frac{1}{2} \sum_{k=1}^{c} (t_k - y_k)^2$$

Loss for one training sample



Gradient Descent

Network parameters $\theta = \{W_{ij}^l, b_i^l\} \begin{cases} l = 1, \dots, L \\ i = 1, \dots, n^l \\ j = 1, \dots, n^{l-1} \end{cases}$

For example: n^{l-1}=1000, n^l=1000

Then each layer has 10^6 parameters

Tens of Millions of parameters ……

Need to compute partial derivative with respect to each parameter:

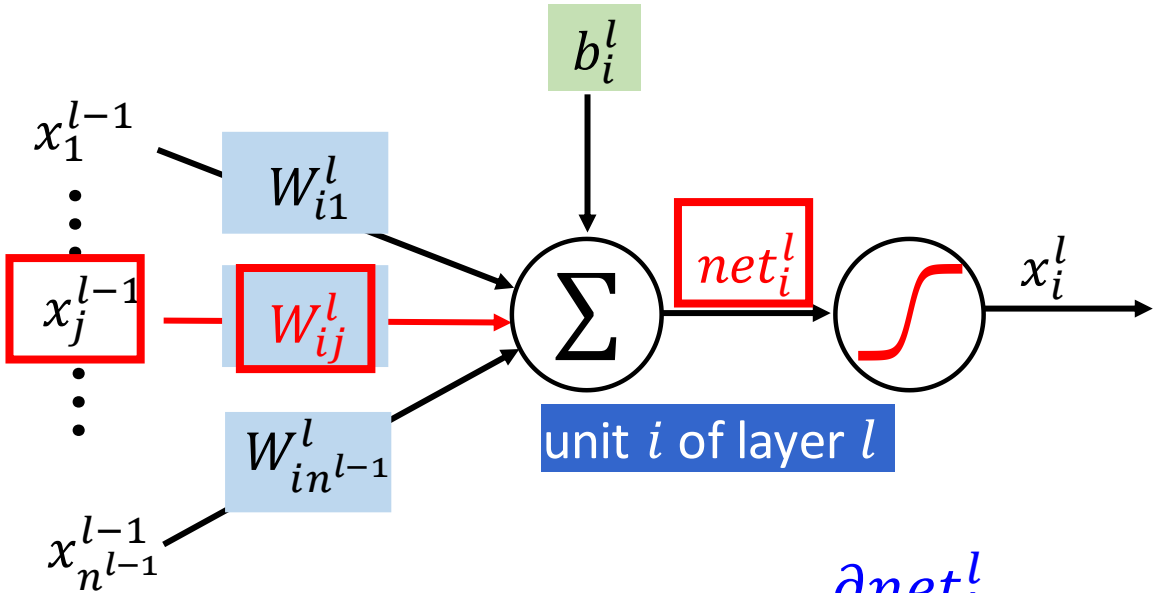
$$W_{ij}^l \leftarrow W_{ij}^l - \eta \frac{\partial \mathcal{L}(\theta)}{\partial W_{ij}^l}$$

$$b_i^l \leftarrow b_i^l - \eta \frac{\partial \mathcal{L}(\theta)}{\partial b_i^l}$$

Compute $\frac{\partial \mathcal{L}}{\partial W_{ij}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_i^l}$

$$net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l \, x_j^{l-1} + b_i^l$$

- Identify the relation between W_{ij}^l and net_i^l, because W_{ij}^l can only affect the network output through net_i^l



- Apply chain rule:

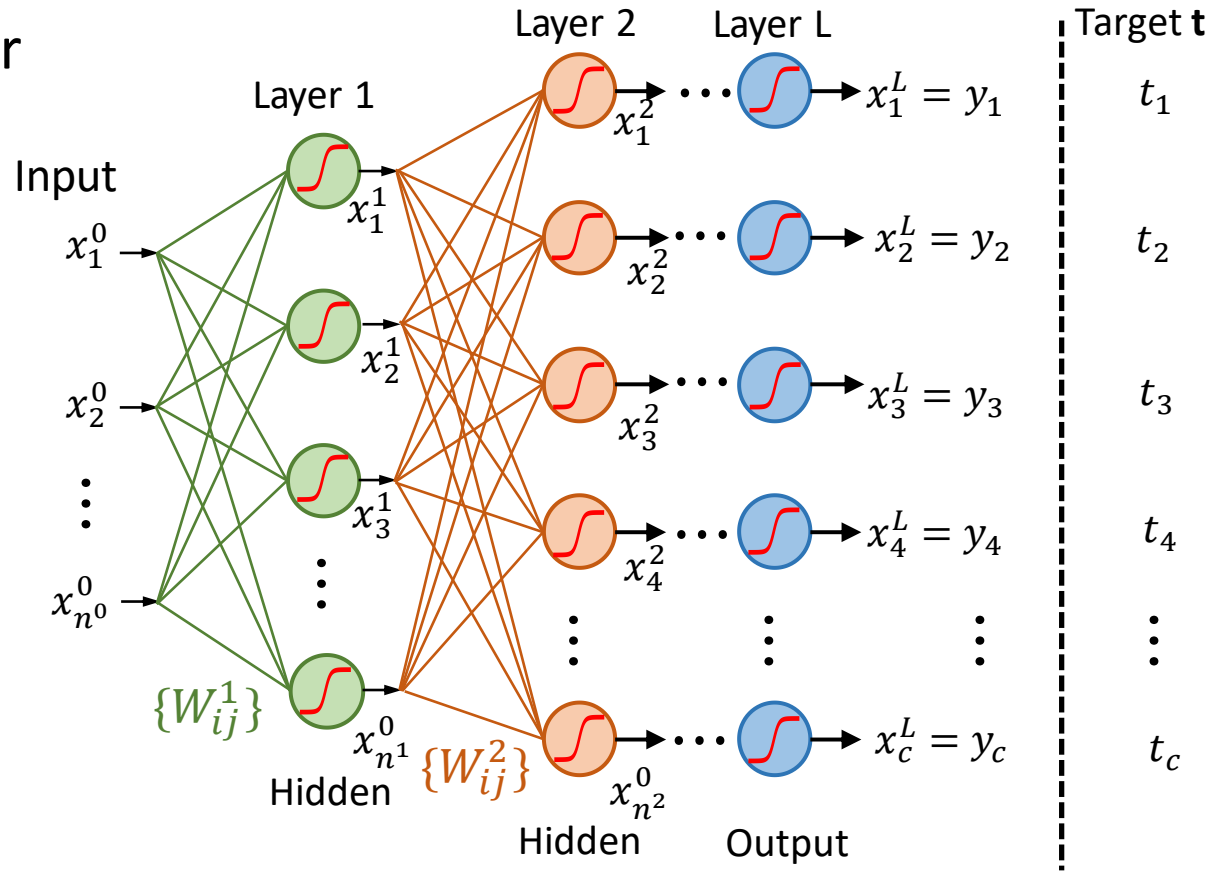
where: $\frac{\partial net_i^l}{\partial W_{ij}^l} = x_j^{l-1}$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = \frac{\partial net_i^l}{\partial W_{ij}^l} \frac{\partial \mathcal{L}}{\partial net_i^l} = x_j^{l-1} \frac{\partial \mathcal{L}}{\partial net_i^l} ?$$

Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$ $\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = x_j^{l-1} \frac{\partial \mathcal{L}}{\partial net_i^l}$

- Identify the relation between net_i^l and those neurons connected to it in the immediate downstream (next layer)

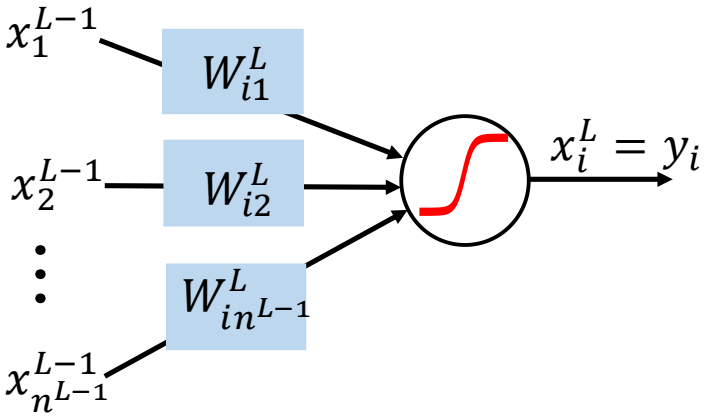
- Two cases:
  $l = L$: the output layer
  $l < L$: hidden layers

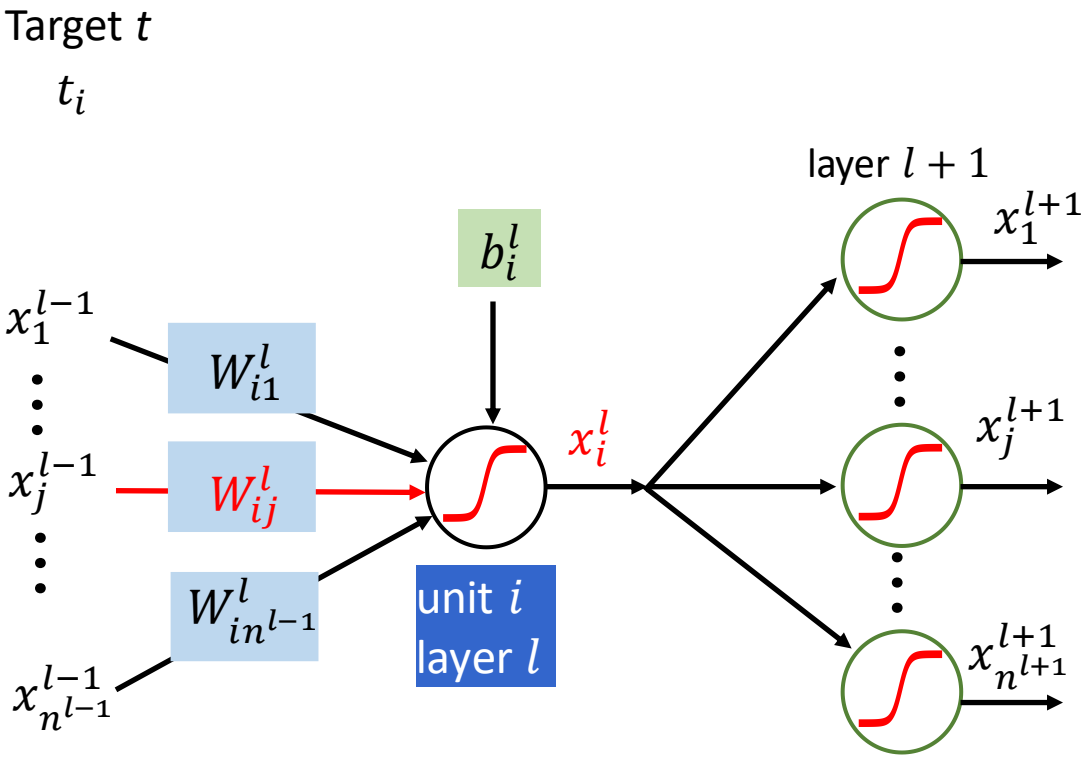


Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = x_j^{l-1} \frac{\partial \mathcal{L}}{\partial net_i^l}$$

- Identify the relation between net_i^l and those neurons connected to it in the immediate downstream (next layer)
- Case 1
  $l = L$: the output units
- Case 2:
  $l < L$: hidden units



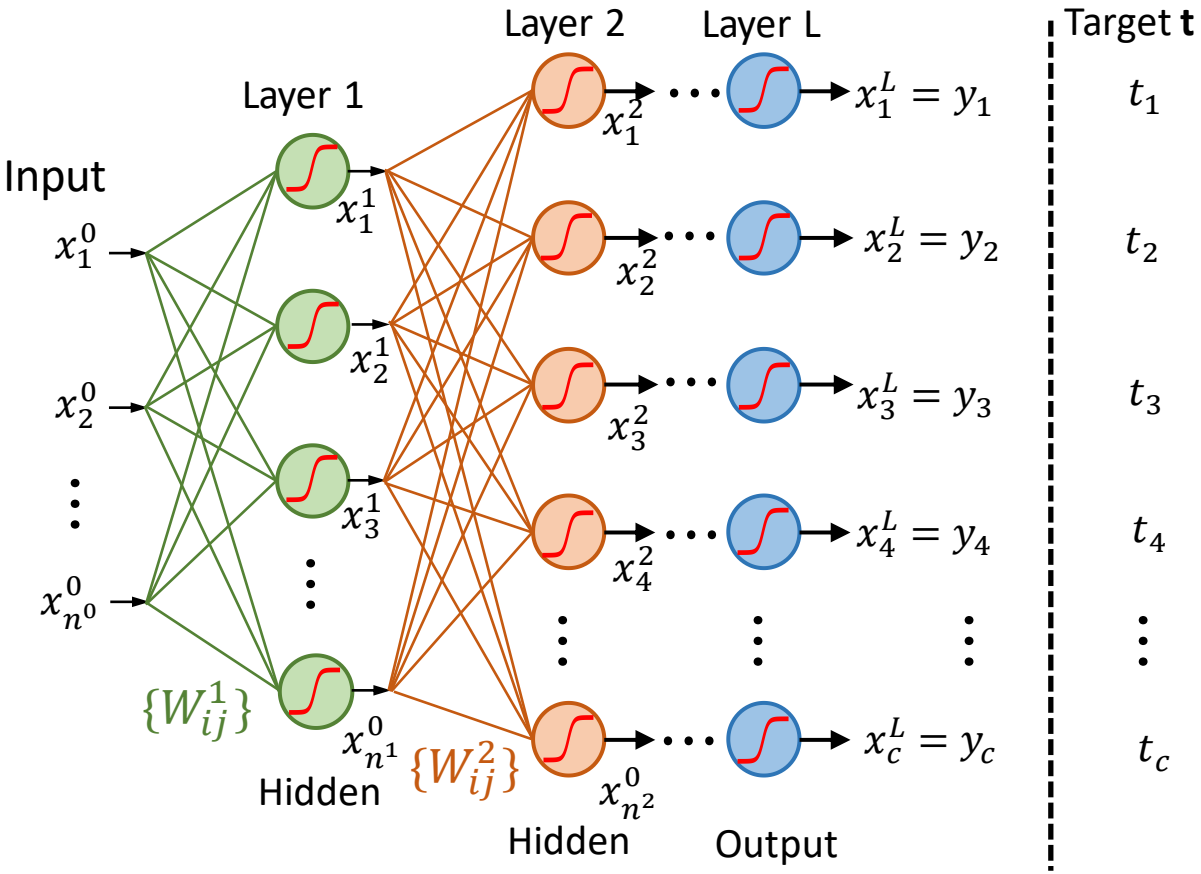


Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$

- Sensitivity of unit i at layer l

Describe how the overall error changes with the unit's net activation:

$$\delta_i^l = -\frac{\partial \mathcal{L}}{\partial net_i^l}$$



Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$

$$\mathcal{L}(\theta) = \frac{1}{2}\sum_{k=1}^{c}(t_k - y_k)^2$$

- Case 1
  $l = L$: the output units

$$x_i^L = \sigma\left(net_i^L\right)$$

$$x_i^L = y_i$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^L} = x_j^{L-1}\frac{\partial \mathcal{L}}{\partial net_i^L}$$

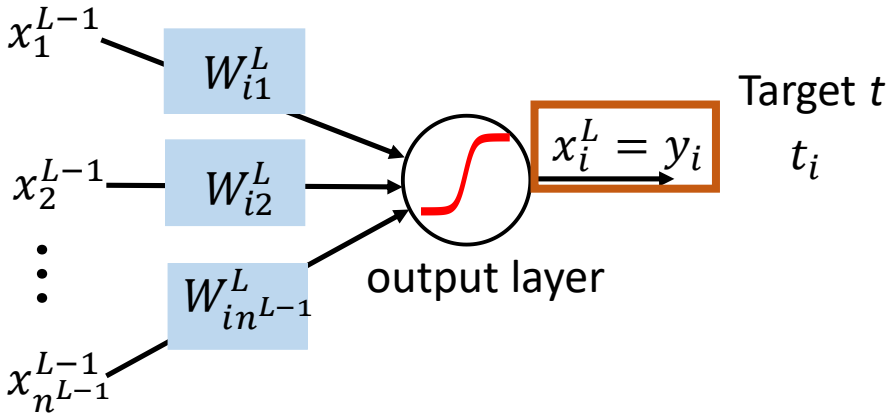
- Chain Rule

$$\frac{\partial \mathcal{L}}{\partial net_i^L} = \frac{\partial x_i^L}{\partial net_i^L}\frac{\partial \mathcal{L}}{\partial x_i^L} = \sigma'\left(net_i^L\right)\frac{\partial \mathcal{L}}{\partial y_i} = -(t_i - y_i)\sigma'\left(net_i^L\right)$$

Suppose:

$$\delta_i^L = (t_i - y_i)\sigma'\left(net_i^L\right)$$

Then: $\frac{\partial \mathcal{L}}{\partial W_{ij}^L} = -\delta_i^L x_j^{L-1}$



Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = x_j^{l-1} \frac{\partial \mathcal{L}}{\partial net_i^l}$$

$$x_i^l = \sigma\left(net_i^l\right)$$

- Case 2
  $l < L$: hidden units

- Identify the relation between net_i^l and those neurons connected to it in the immediate downstream (next layer)

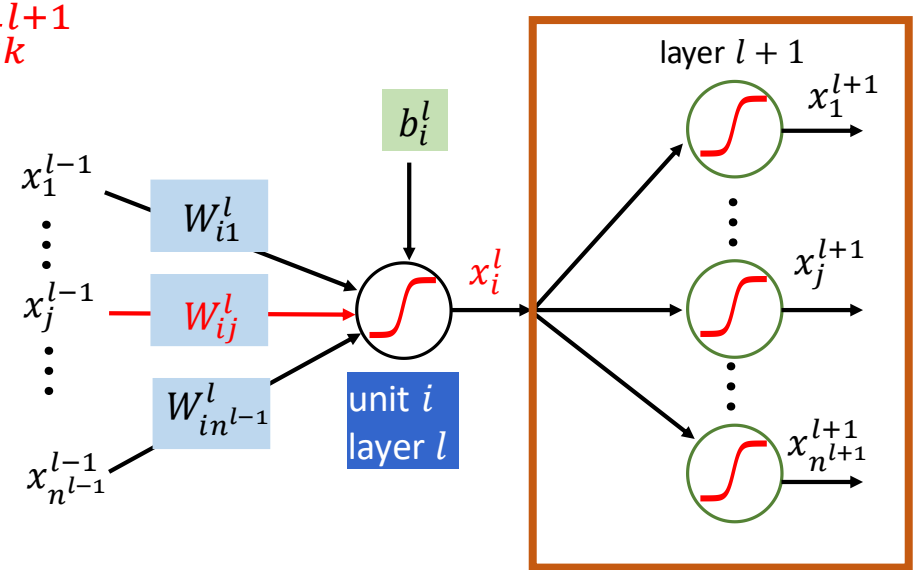
$$\delta_i^l = -\frac{\partial \mathcal{L}}{\partial net_i^l}$$

- Chain Rule

$$\frac{\partial \mathcal{L}}{\partial net_i^l} = \sum_{k \in downstream(net_i^l)} \frac{\partial net_k^{l+1}}{\partial net_i^l} \frac{\partial \mathcal{L}}{\partial net_k^{l+1}}$$

$$= \sum_{k=1}^{n^{l+1}} \frac{\partial net_k^{l+1}}{\partial net_i^l} \frac{\partial \mathcal{L}}{\partial net_k^{l+1}}$$

$$\delta_i^l = \sum_{k=1}^{n^{l+1}} \frac{\partial net_k^{l+1}}{\partial net_i^l} \left(\delta_k^{l+1}\right)$$



Compute $\frac{\partial \mathcal{L}}{\partial net_i^l}$

- Case 2
  $l < L$: hidden units

$$\delta_i^l = \sum_{k=1}^{n^{l+1}} \frac{\partial net_k^{l+1}}{\partial net_i^l} \delta_k^{l+1}$$

$$= \sum_{k=1}^{n^{l+1}} \frac{\partial net_k^{l+1}}{\partial x_i^l} \frac{\partial x_i^l}{\partial net_i^l} \delta_k^{l+1}$$

$$= \sum_{k=1}^{n^{l+1}} W_{ki}^{l+1} \sigma'(net_i^l) \delta_k^{l+1}$$

$$= \sigma'(net_i^l) \sum_{k=1}^{n^{l+1}} W_{ki}^{l+1} \delta_k^{l+1}$$

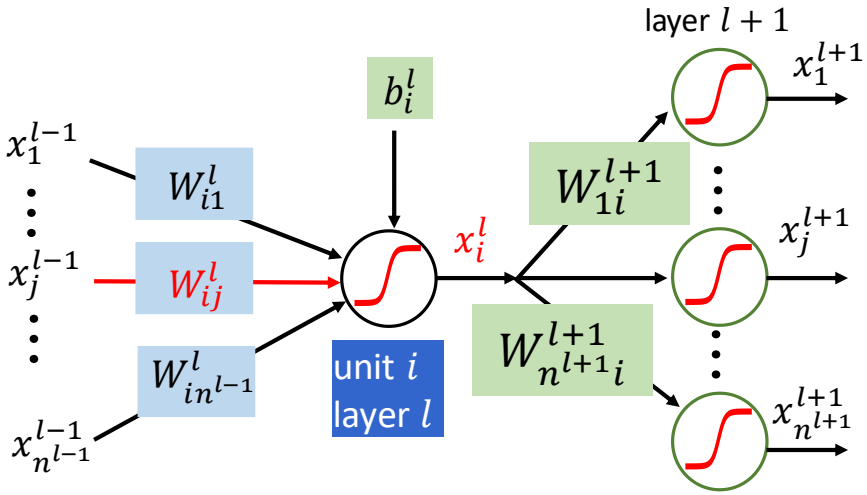
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = -x_j^{l-1} \delta_i^l$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = x_j^{l-1} \frac{\partial \mathcal{L}}{\partial net_i^l}$$

$$x_i^l = \sigma(net_i^l)$$

σ is the activation function

$$net_k^{l+1} = \sum_{i=1}^{n^l} W_{ki}^{l+1} x_i^l + b_k^{l+1}$$



Summary: Error Propagation

$$\delta_i^l = -\frac{\partial \mathcal{L}}{\partial net_i^l}$$

- Case 1

  $l = L$: the output units

σ is the activation function

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^L} = -\delta_i^L x_j^{L-1}$$

$$\delta_i^L = \sigma'\left(net_i^L\right)(t_i - y_i)$$

error of output layer

- Case 2

  $l < L$: hidden units

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = -\delta_i^l x_j^{l-1}$$

$$\delta_i^l = \sigma'\left(net_i^l\right) \sum_{k=1}^{n^{l+1}} W_{ki}^{l+1} \delta_k^{l+1}$$

error backpropagated

δ_i^l : an "error term" that measures how much that unit is "responsible" for any errors in the output.

Case 1 and case 2 can be unified.

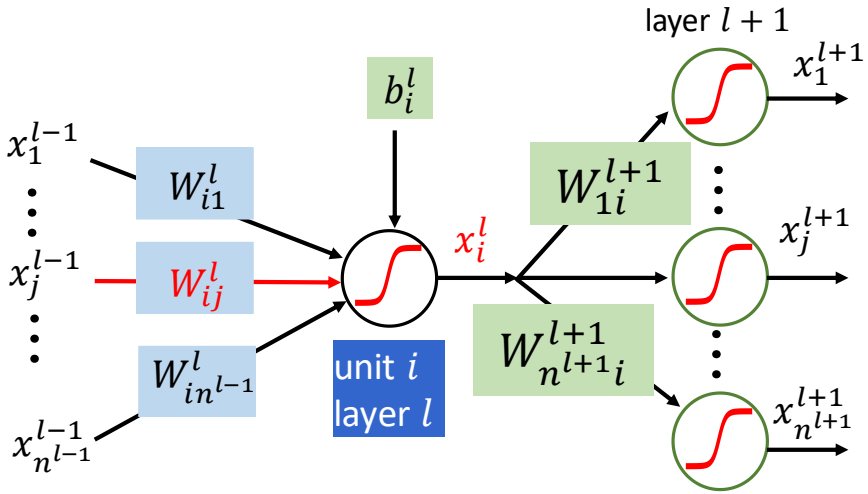
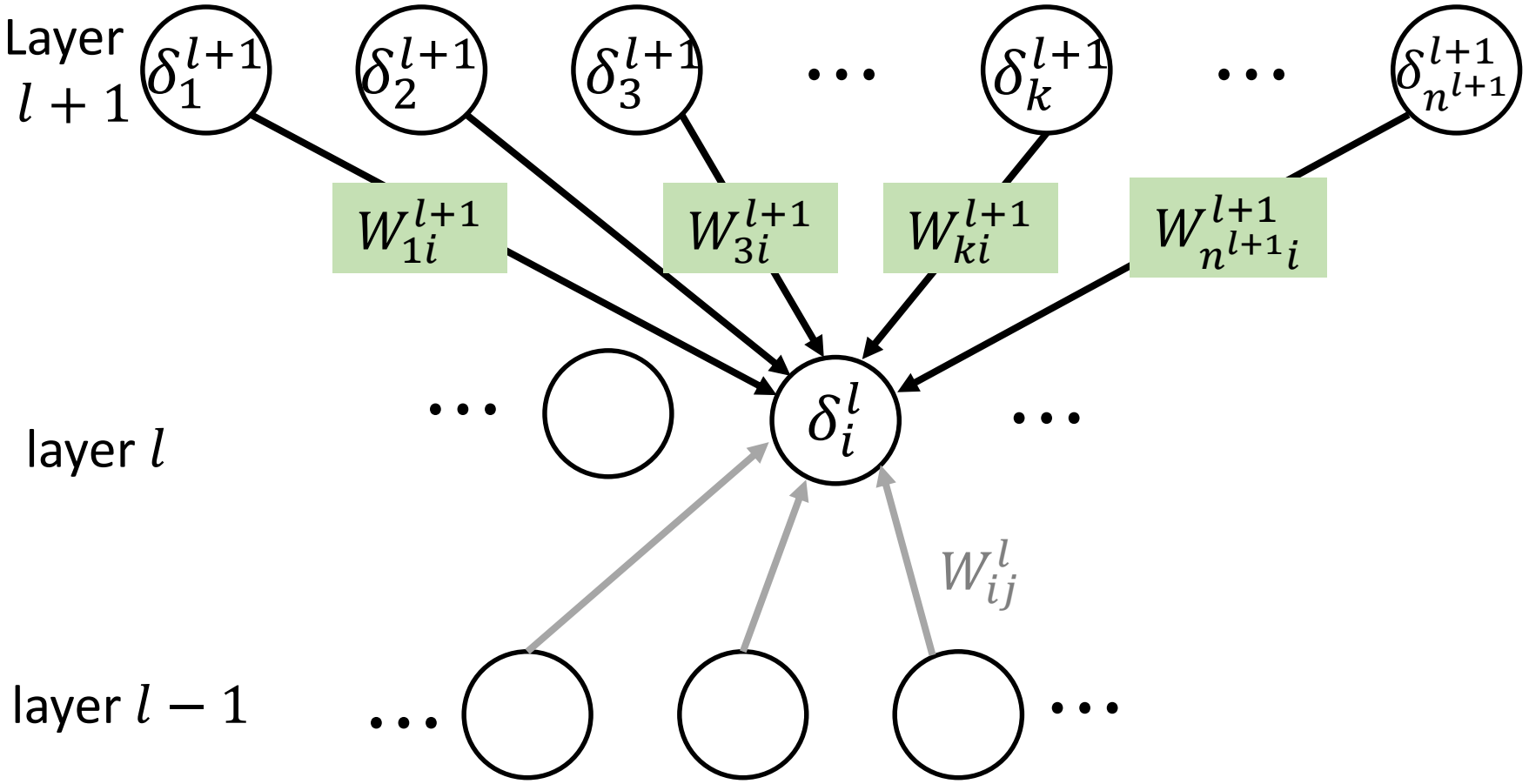


Illustration of Error Propagation

$$\delta_i^l = \sigma'\left(net_i^l\right) \sum_{k=1}^{n^{l+1}} W_{ki}^{l+1} \delta_k^{l+1}$$



- The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units.
- The output unit sensitivities are thus propagated “back” to the hidden units.

BP: Summary

$$net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l x_j^{l-1} + b_i^l$$

$$x_i^l = \sigma(net_i^l)$$

- $l = L$: the output units

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^L} = -\delta_i^L x_j^{L-1}$$

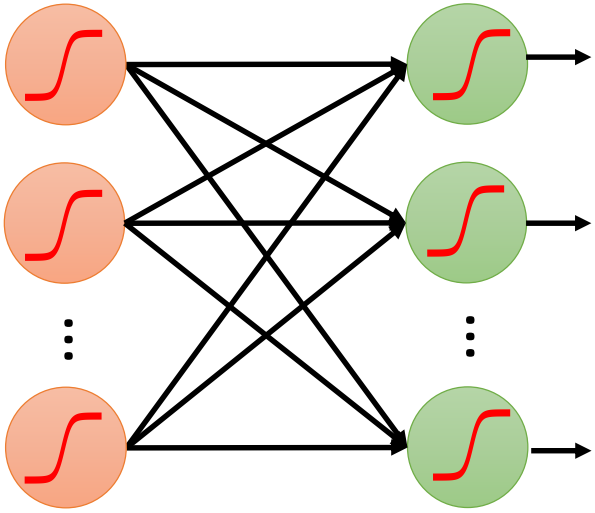
$$\delta_i^L = (t_i - y_i)\sigma'(net_i^L)$$

- $0 < l < L$: hidden units

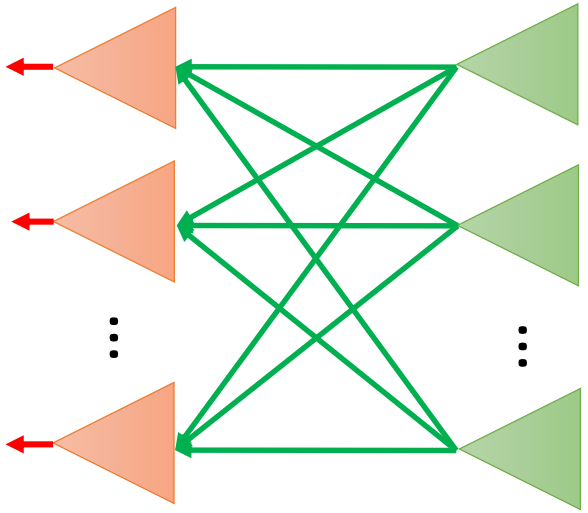
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = -\delta_i^l x_j^{l-1}$$

$$\delta_i^l = \sigma'(net_i^l) \sum_{k=1}^{n^{l+1}} W_{ki}^{l+1} \delta_k^{l+1}$$

Forward Pass



Backward Pass



$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = -\delta_i^l x_j^{l-1}$$

BP: Summary

$$net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l x_j^{l-1} + b_i^l$$

$$x_i^l = \sigma\left(net_i^l\right)$$

- $l = L$: the output units

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^L} = -\delta_i^L x_j^{L-1}$$

$$\delta_i^L = (t_i - y_i)\sigma'\left(net_i^L\right)$$

- $0 < l < L$: hidden units

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = -\delta_i^l x_j^{l-1}$$

$$\delta_i^l = \sigma'\left(net_i^l\right)\sum_{k=1}^{n^{l+1}} W_{ki}^{l+1}\delta_k^{l+1}$$

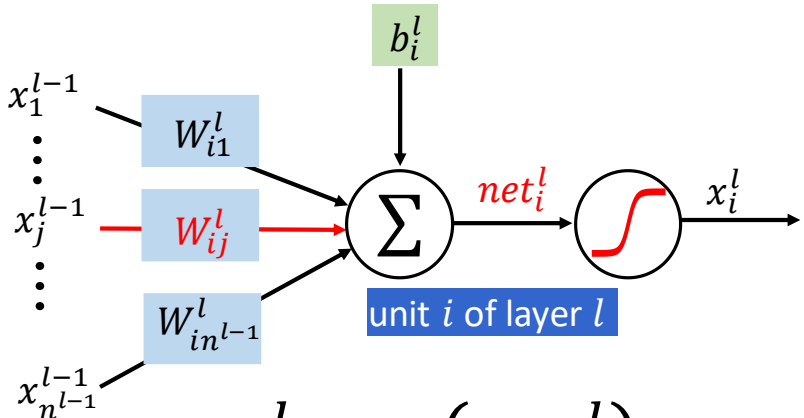
- Implementation note

Need to compute: $\sigma'\left(net_i^L\right)$

Suppose sigmoid activation function:

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

$$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right)$$

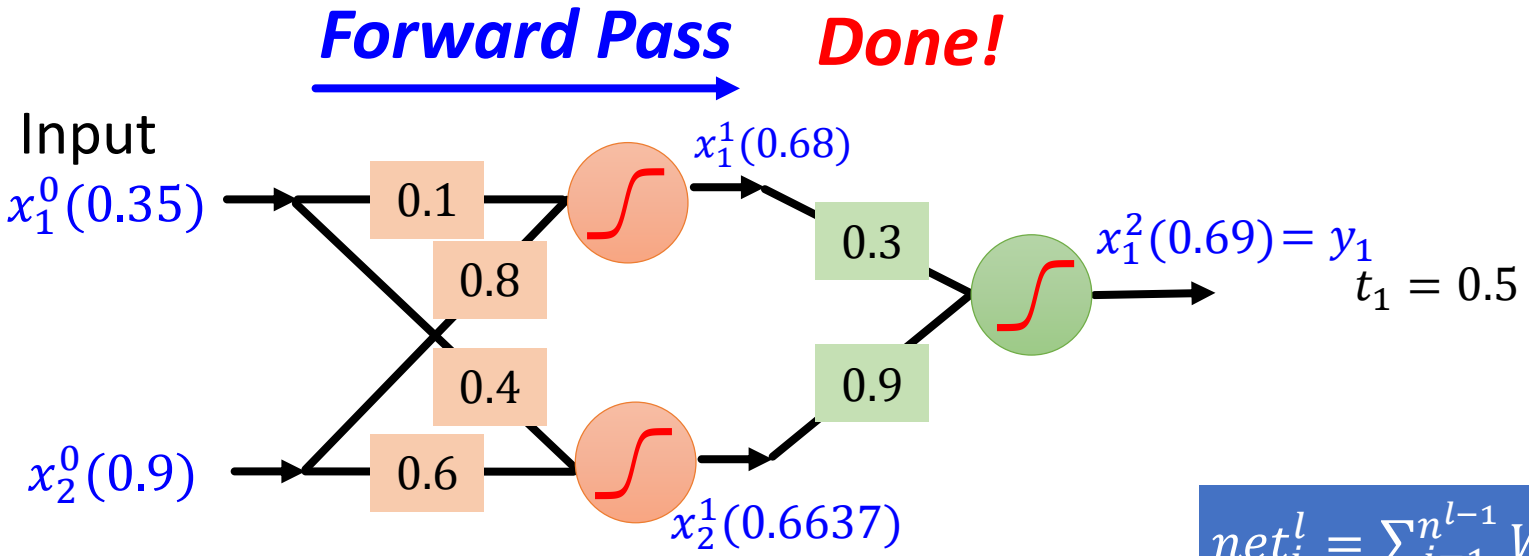


$$x_i^l = \sigma\left(net_i^l\right)$$

Activations $\{x_i^l\}$ have already been stored away from the forward pass through the network.

$$\sigma'\left(net_i^l\right) = \sigma\left(net_i^l\right)\left(1 - \sigma\left(net_i^l\right)\right) = x_i^l(1 - x_i^l)$$

An Example for Backpropagation



$W_{11}^1 = 0.1$ $W_{21}^1 = 0.4$

$W_{12}^1 = 0.8$ $W_{22}^1 = 0.6$

$$net_i^l = \sum_{j=1}^{n^{l-1}} W_{ij}^l \, x_j^{l-1}$$

$$x_i^l = \sigma(net_i^l)$$

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

$$x_1^1 = \frac{1}{1 + exp(-(0.1 * 0.35 + 0.9 * 0.8))} = 0.68$$

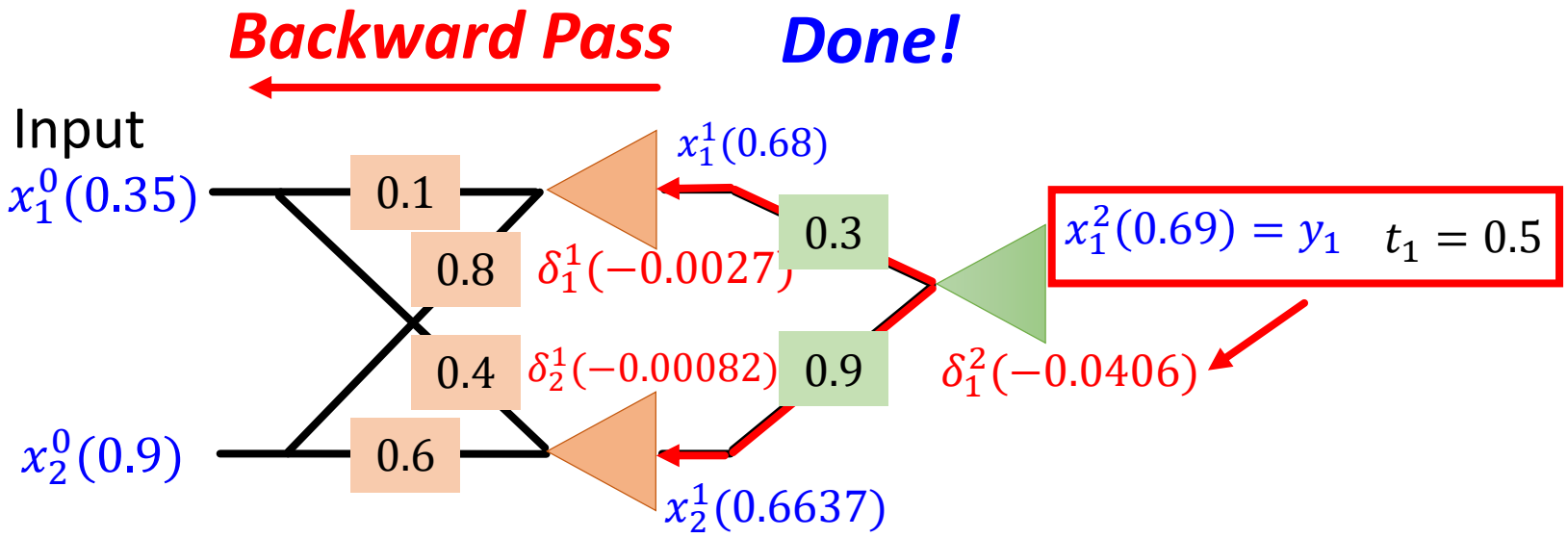
$x_2^1 = 0.6637$

$W_{11}^2 = 0.3$

$W_{12}^2 = 0.9$

$x_1^2 = 0.69 = y_1$

An Example for Backpropagation



L =2: the output units: $\partial\mathcal{L}/\partial W_{ij}^L = -\delta_i^L x_j^{L-1}$

$\delta_i^L = (t_1 - y_1)\sigma'(net_i^L)$

$\delta_1^2 = (t_1 - y_1)\sigma'(net_1^2) = (t_1 - y_1)x_1^2(1 - x_1^2)$

$= (0.5 - 0.69) * 0.69 * (1 - 0.69) = -0.0406$

$\sigma'(net_i^l) = x_i^l(1 - x_i^l)$

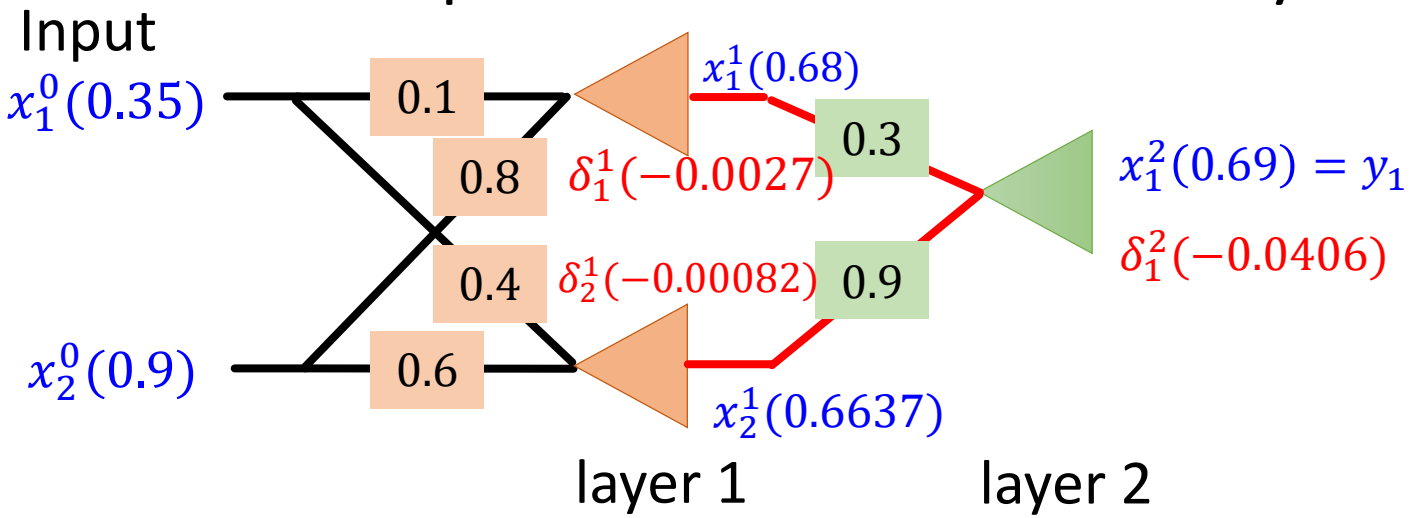
$0 < l < 2$: hidden units: $\partial\mathcal{L}/\partial W_{ij}^l = -\delta_i^l x_j^{l-1}$

$\delta_i^l = \sigma'(net_i^l)\sum_{k=1}^{n^{l+1}} W_{ki}^{l+1}\delta_k^{l+1}$

$\delta_1^1 = x_1^1(1 - x_1^1) * W_{11}^2 * \delta_1^2 = 0.68 * (1 - 0.68) * 0.3 * (-0.0406) = -0.0027$

$\delta_2^1 = 0.6637 * (1 - 0.6637) * 0.9 * (-0.0406) = -0.0082$

An Example for BP: Summary



$1 \le l \le 2$: hidden units: $\partial \mathcal{L}/\partial W_{ij}^l = -\delta_i^l x_j^{l-1}$

$W_{11}^1 = 0.1$ $W_{21}^1 = 0.4$ $W_{11}^2 = 0.3$

$W_{12}^1 = 0.8$ $W_{22}^1 = 0.6$ $W_{12}^2 = 0.9$

$$\frac{\partial \mathcal{L}}{\partial W_{11}^2} =$$

$$\frac{\partial \mathcal{L}}{\partial W_{12}^2} =$$

$$\frac{\partial \mathcal{L}}{\partial W_{11}^1} =$$

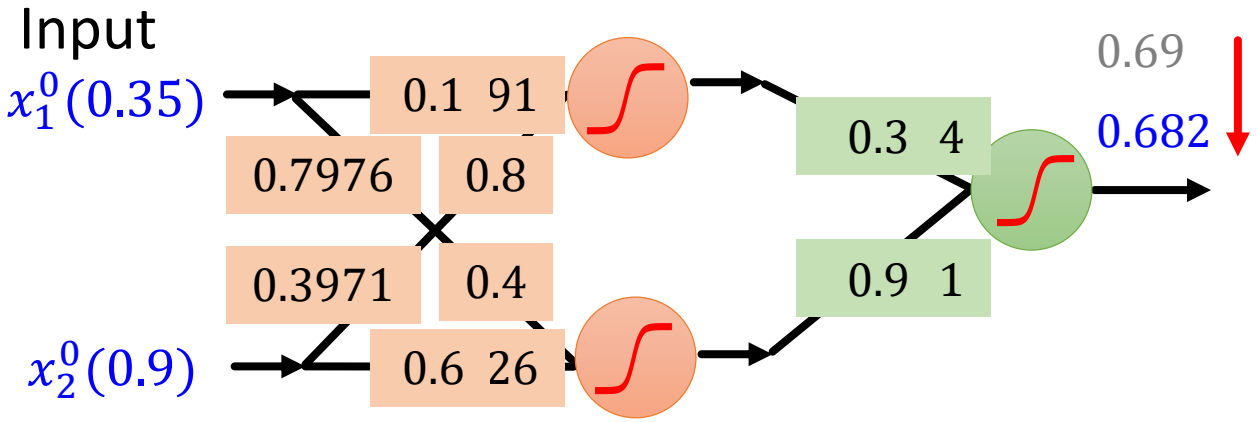
$$\frac{\partial \mathcal{L}}{\partial W_{12}^1} =$$

$$\frac{\partial \mathcal{L}}{\partial W_{21}^1} =$$

$$\frac{\partial \mathcal{L}}{\partial W_{22}^1} =$$

An Example for Backpropagation

Weights Update



Suppose: learning rate $\eta = 1$

$$W_{11}^1 = 0.1 - 0.000945 = 0.0991$$

$$W_{12}^1 = 0.8 - 0.00243 = 0.7976$$

$$W_{21}^1 = 0.4 - 0.00287 = 0.3971$$

$$W_{22}^1 = 0.6 - 0.00738 = 0.5926$$

$$W_{11}^2 = 0.3 - 0.0276 = 0.2724$$

$$W_{12}^2 = 0.9 - 0.0269 = 0.8731$$

Just Repeat!

$$\frac{\partial \mathcal{L}}{\partial W_{11}^1} = 0.000945$$

$$\frac{\partial \mathcal{L}}{\partial W_{12}^1} = 0.00243$$

$$\frac{\partial \mathcal{L}}{\partial W_{21}^1} = 0.00287$$

$$\frac{\partial \mathcal{L}}{\partial W_{22}^1} = 0.00738$$

$$\frac{\partial \mathcal{L}}{\partial W_{11}^2} = 0.0276$$

$$\frac{\partial \mathcal{L}}{\partial W_{12}^2} = 0.0269$$

Next Lecture

Convolutional Neural Network