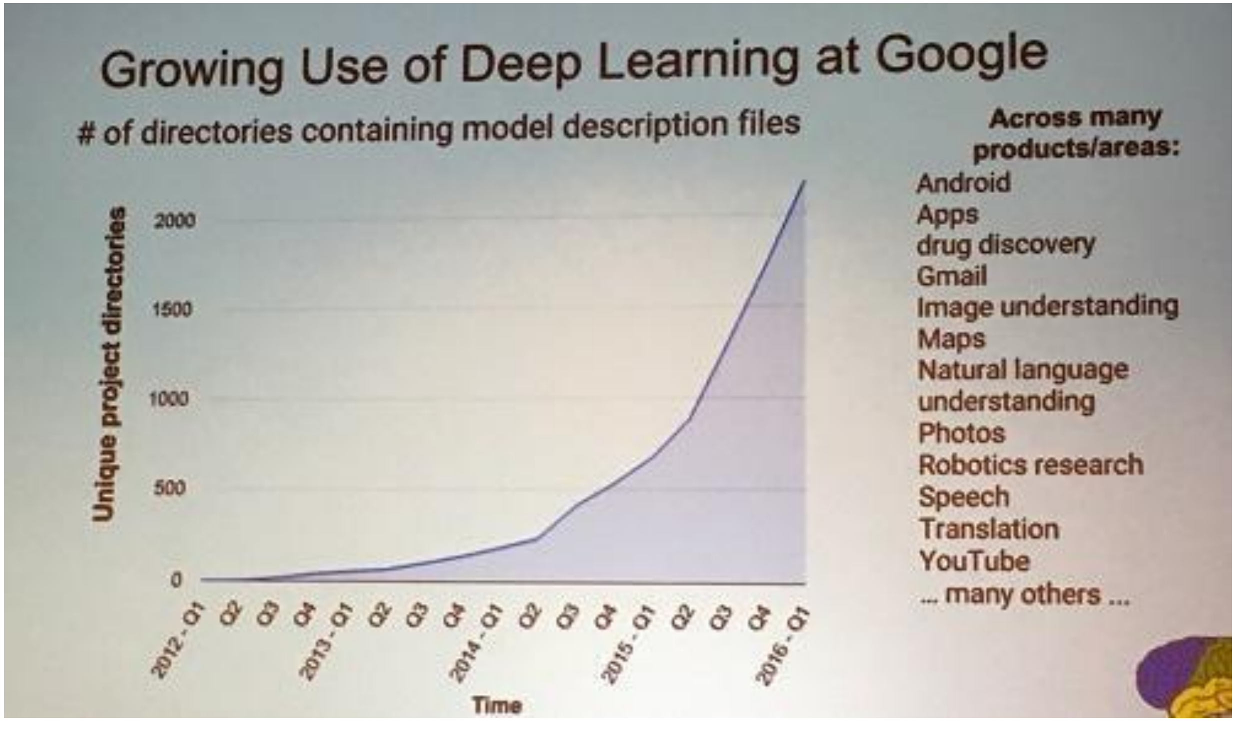
Deep Learning

Hung-yi Lee
李宏毅

Deep learning attracts lots of attention.

- I believe you have seen lots of exciting results before.



Deep learning trends at Google. Source: SIGMOD 2016/Jeff Dean

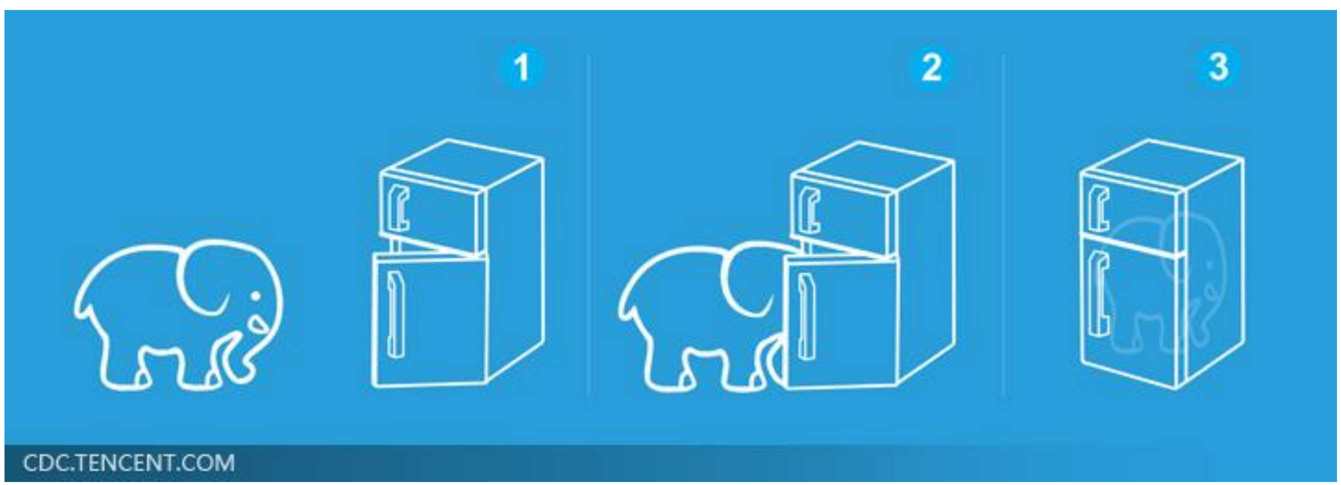
Ups and downs of Deep Learning

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: RBM initialization
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition
- 2015.2: Image recognition surpassing human-level performance
- 2016.3: Alpha GO beats Lee Sedol
- 2016.10: Speech recognition system as good as humans

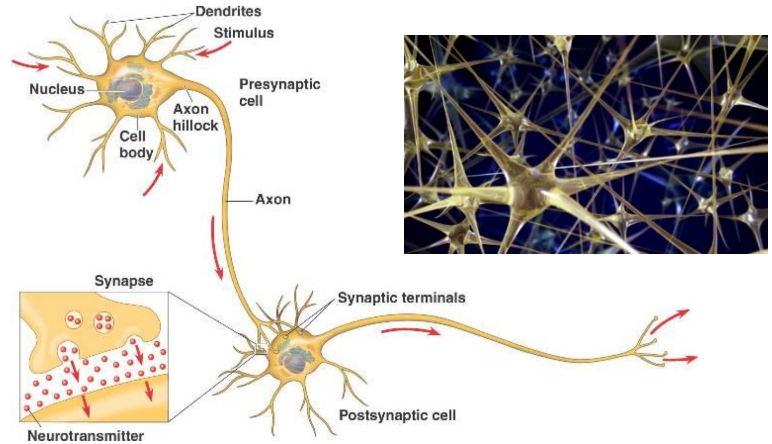
Three Steps for Deep Learning

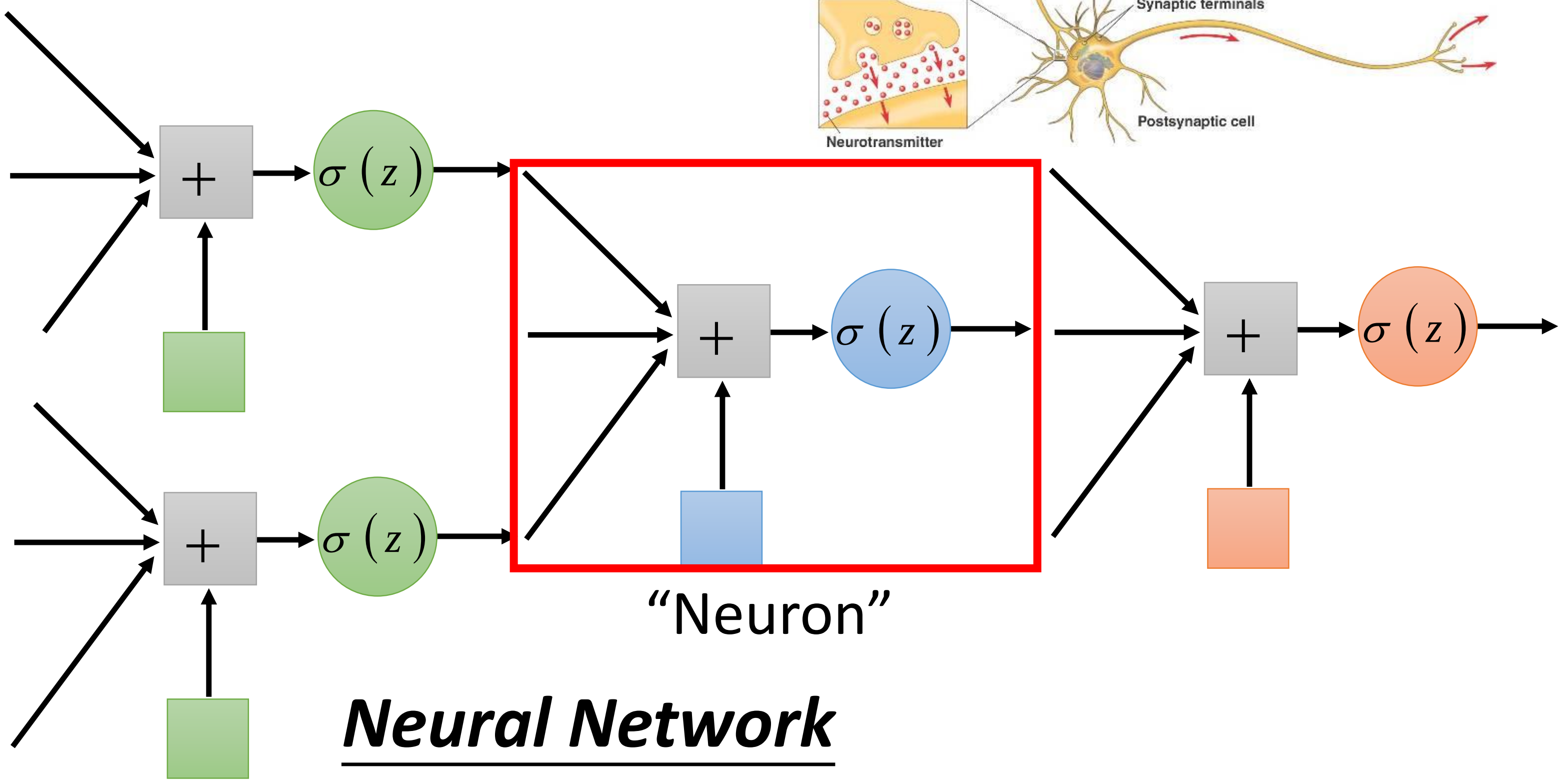
Step 1: Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



Neural Network



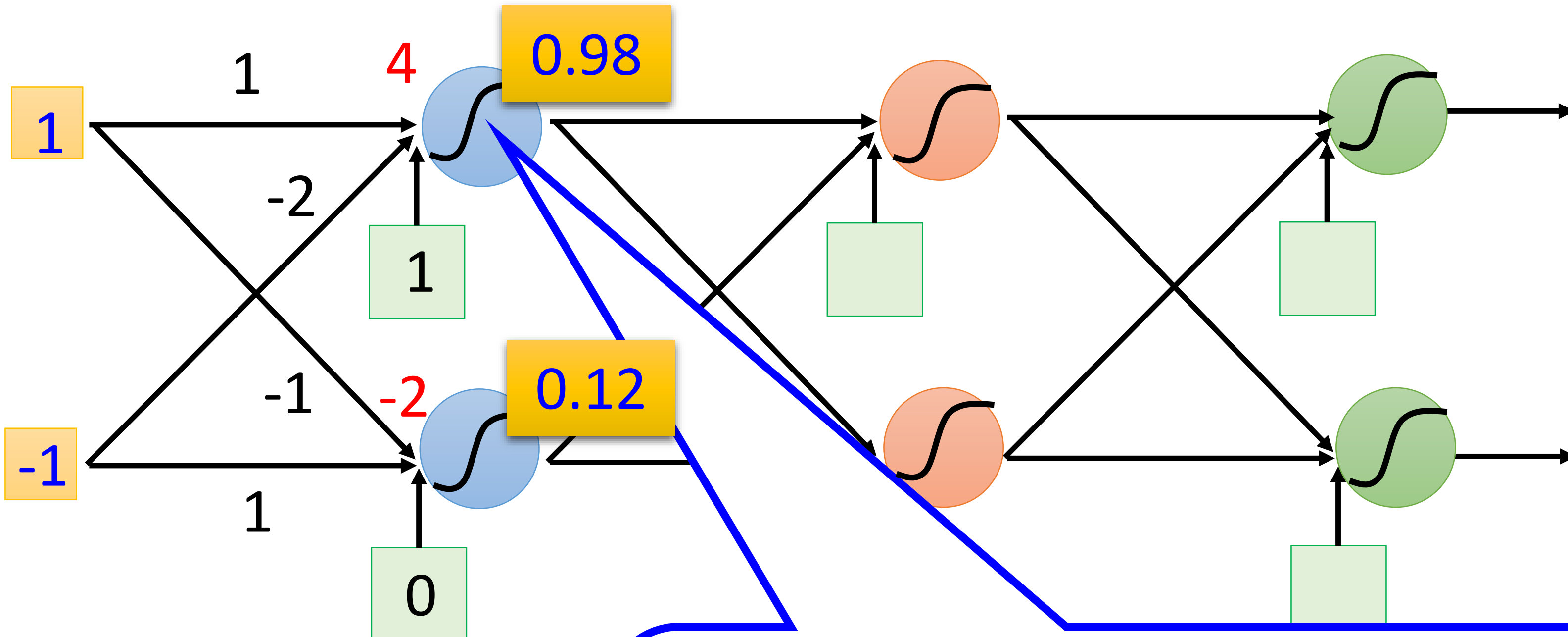


Neural Network

Different connection leads to different network structures

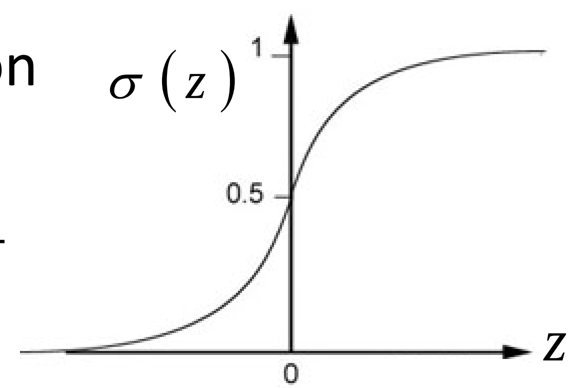
Network parameter θ: all the weights and biases in the “neurons”

Fully Connect Feedforward Network

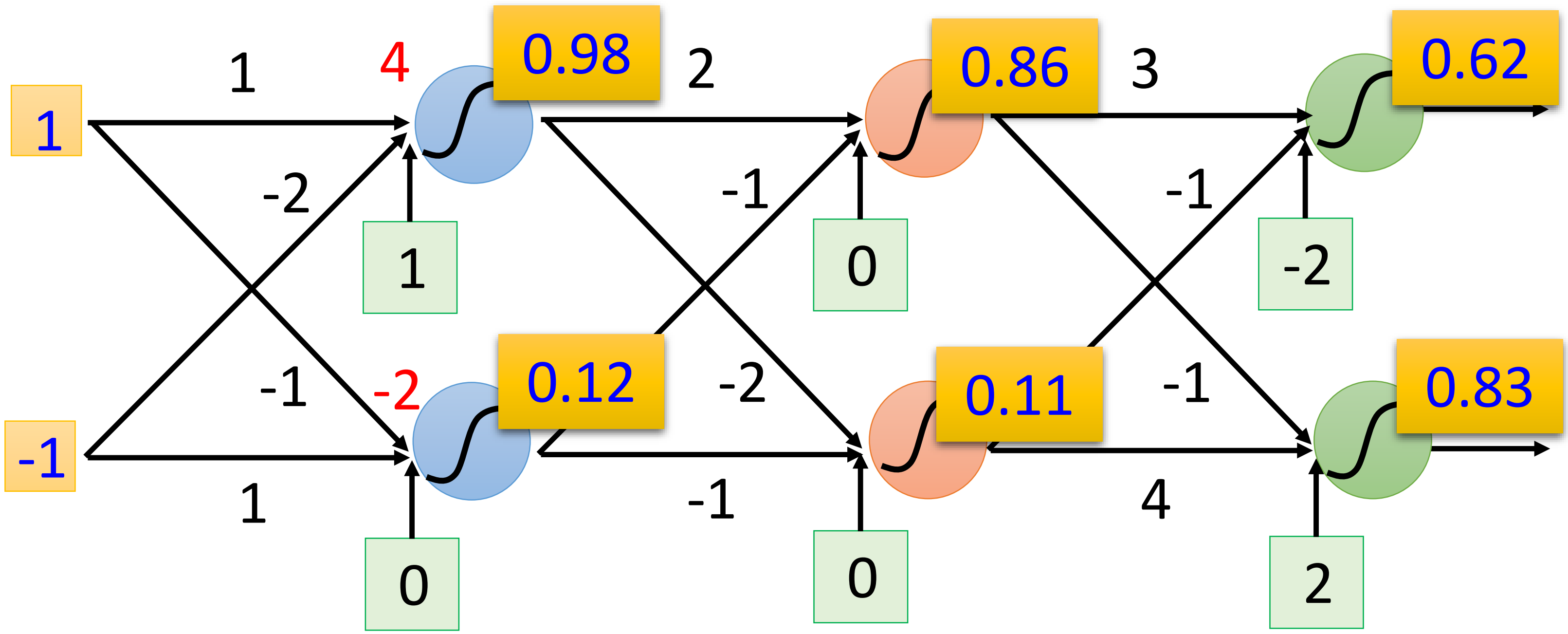


Sigmoid Function

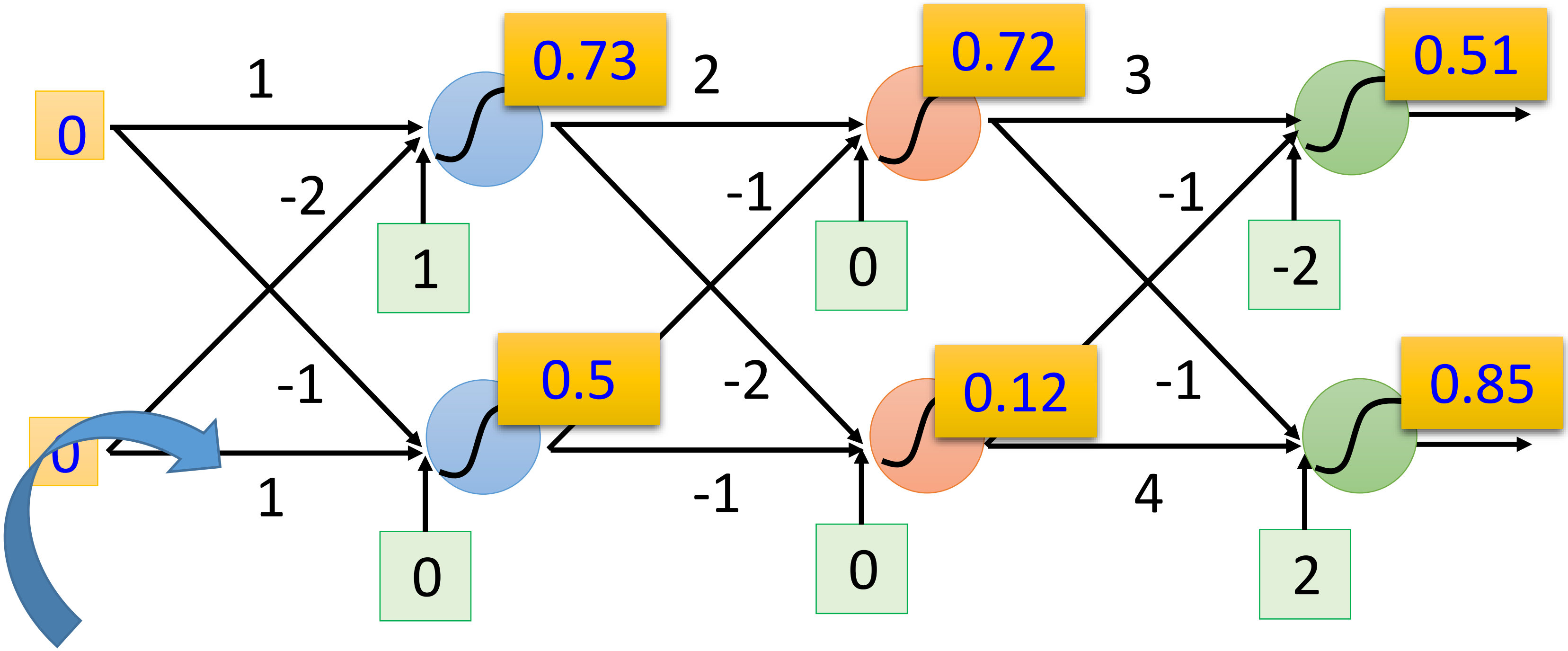
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Fully Connect Feedforward Network



Fully Connect Feedforward Network



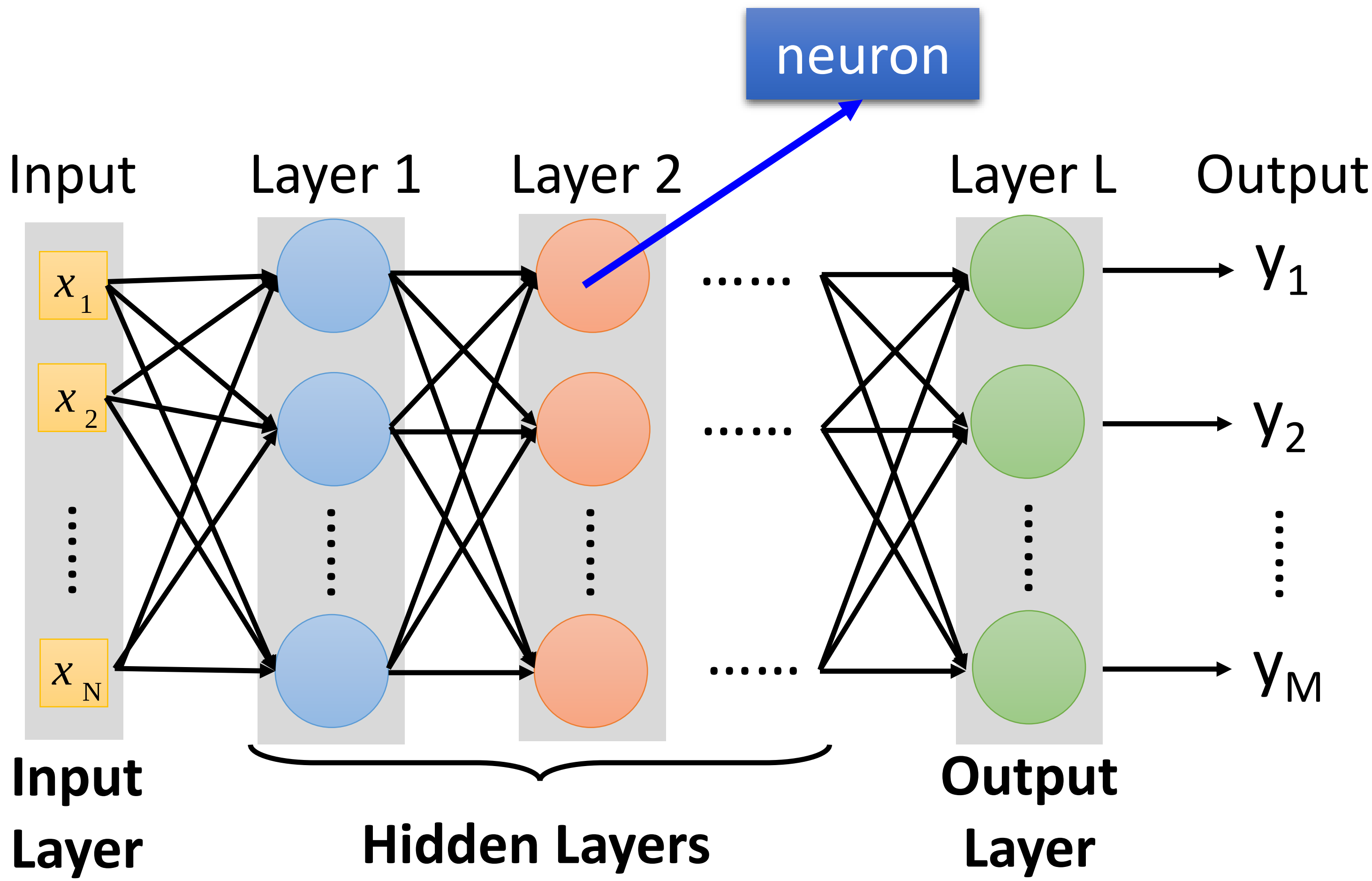
This is a function.

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

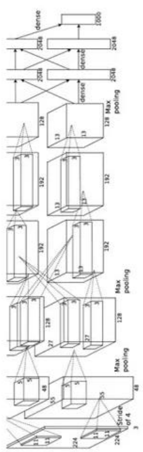


Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

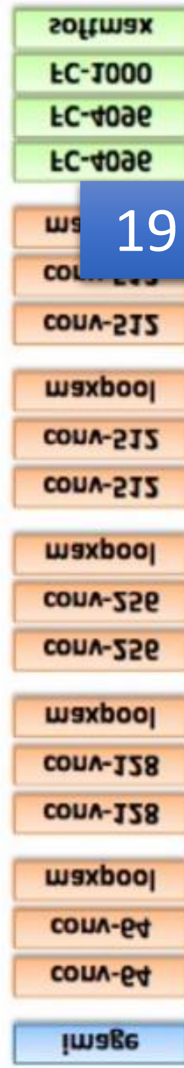
16.4%



AlexNet (2012)

19 layers

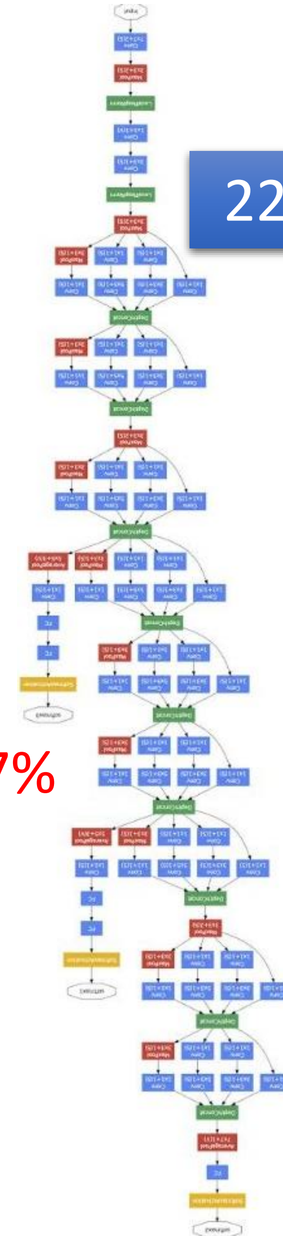
7.3%



VGG (2014)

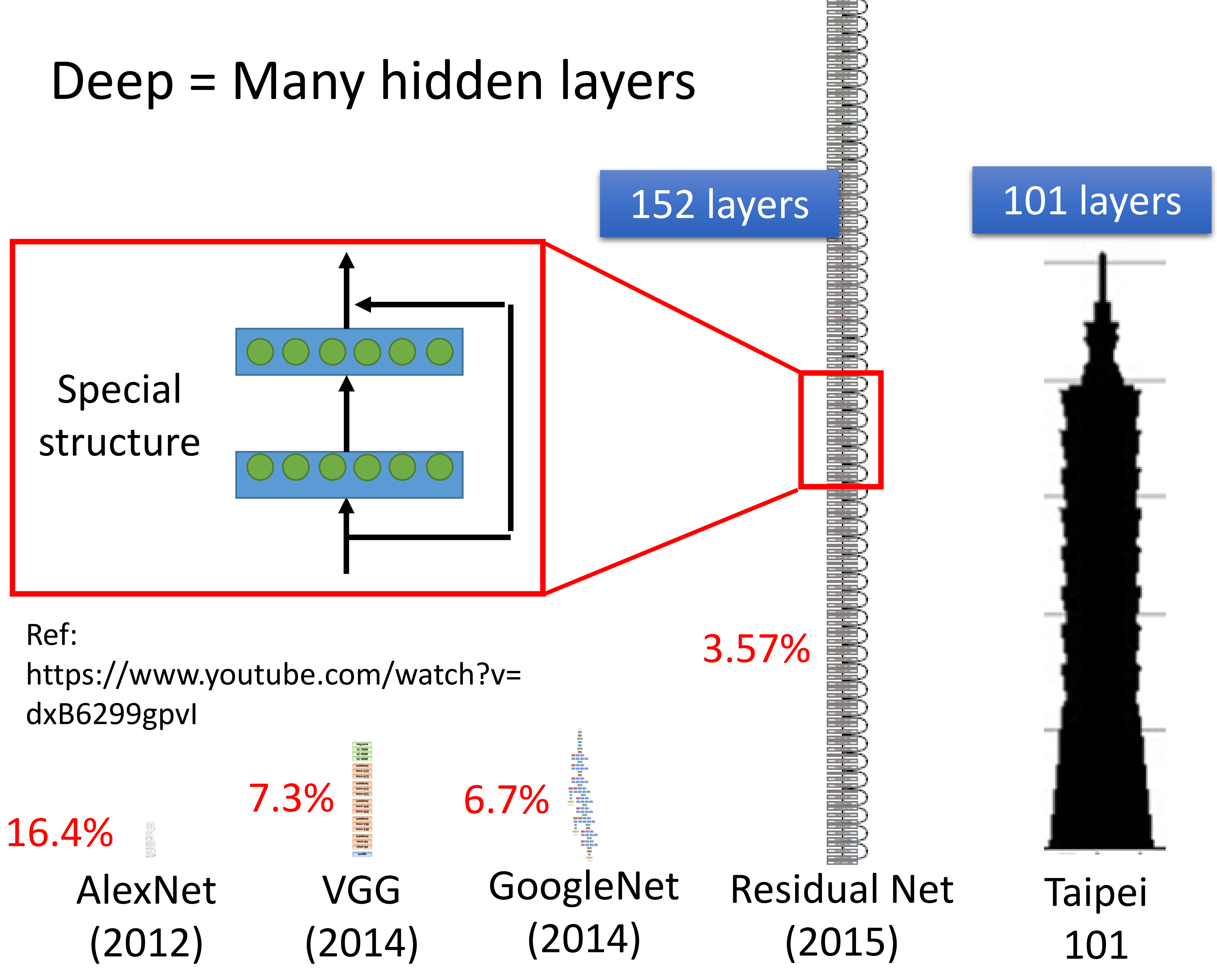
22 layers

6.7%



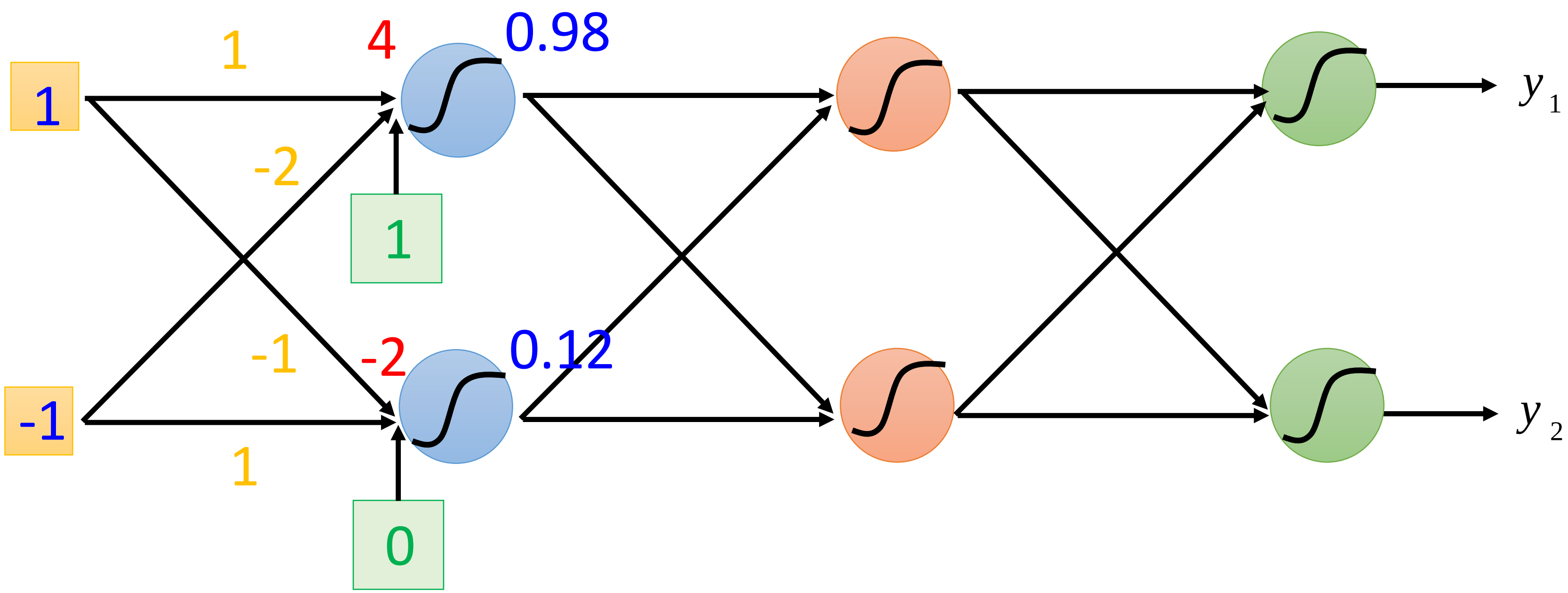
GoogleNet (2014)

Deep = Many hidden layers



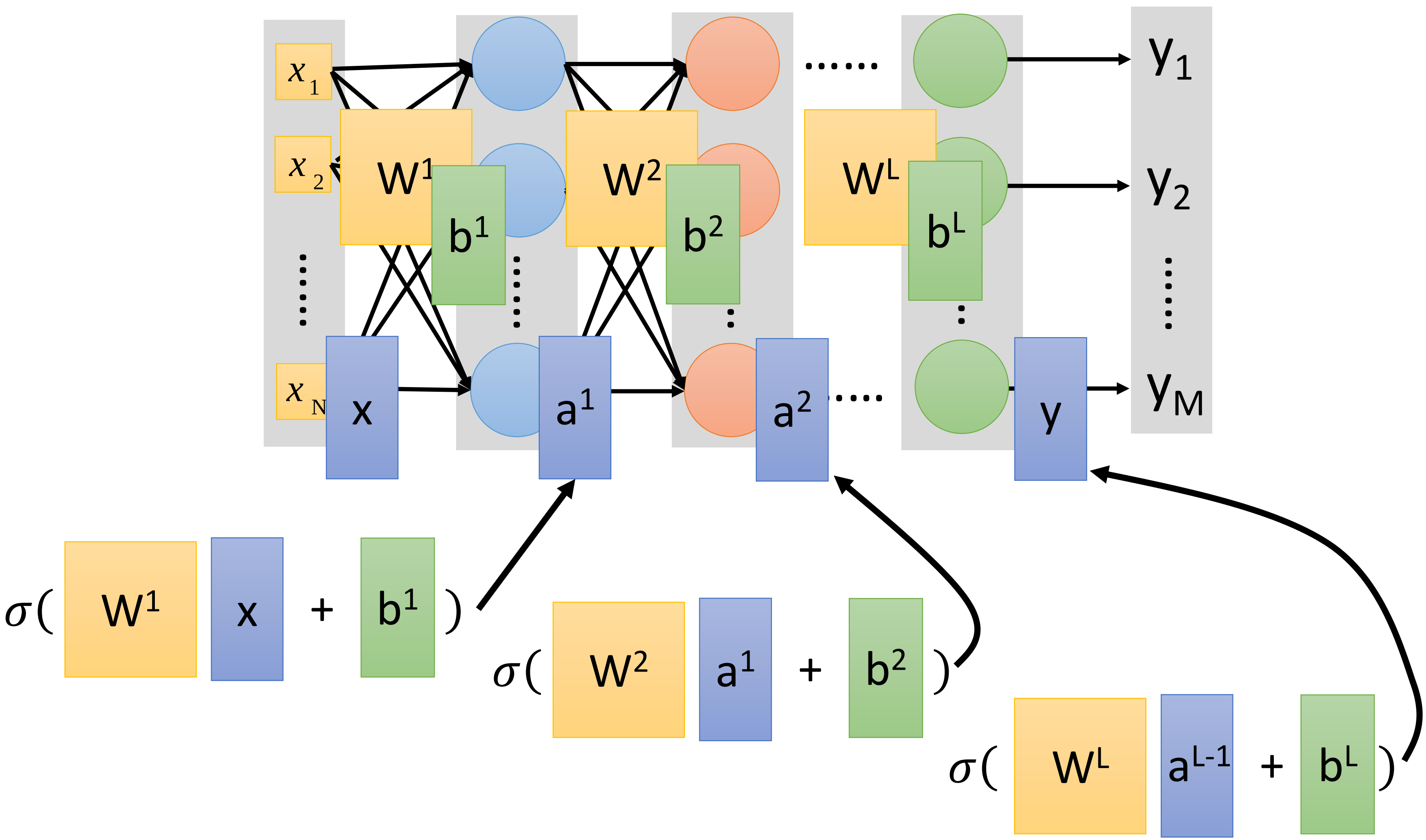
Ref: https://www.youtube.com/watch?v=dxB6299gpvI

Matrix Operation

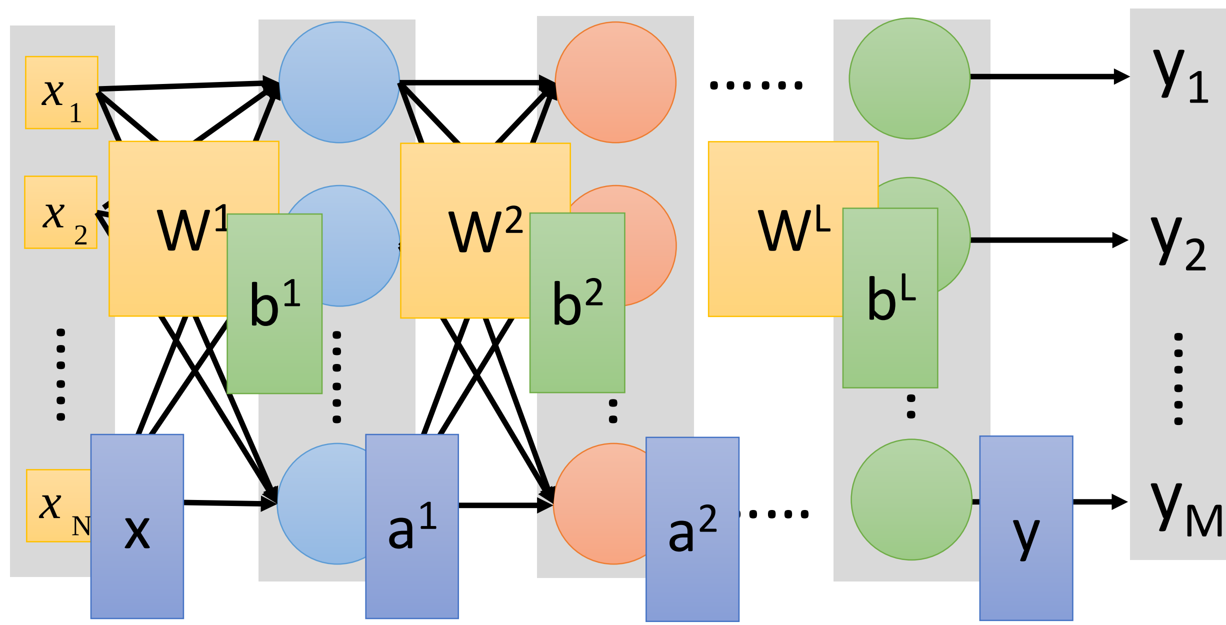


$$\sigma\Big(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\Big) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network

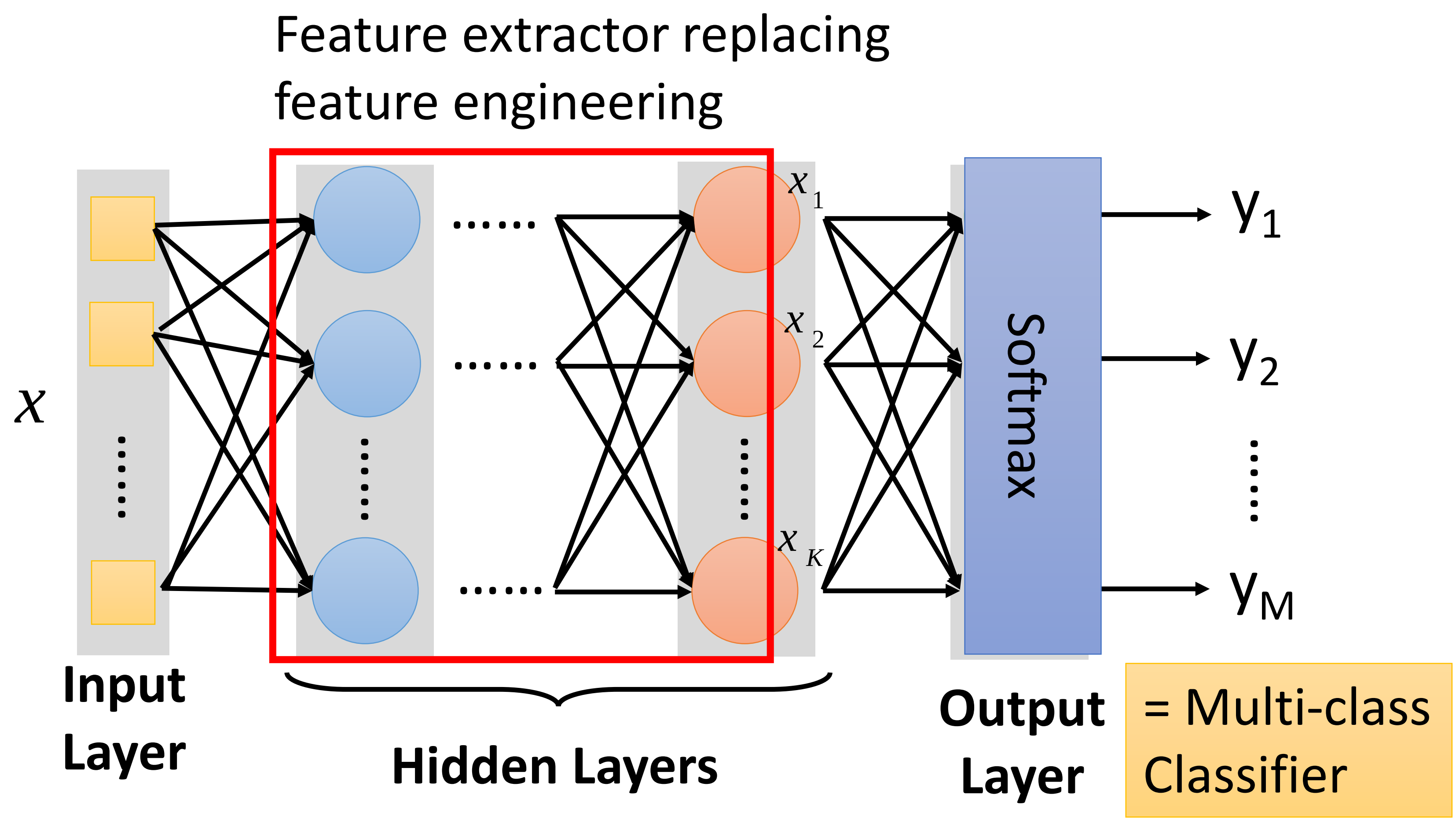


$$y = f(x)$$

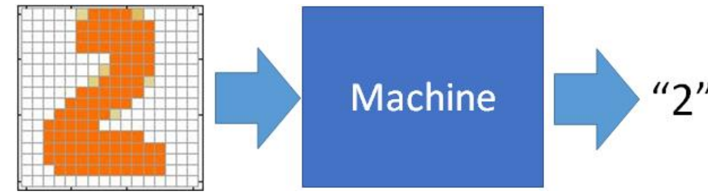
Using parallel computing techniques to speed up matrix operation

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

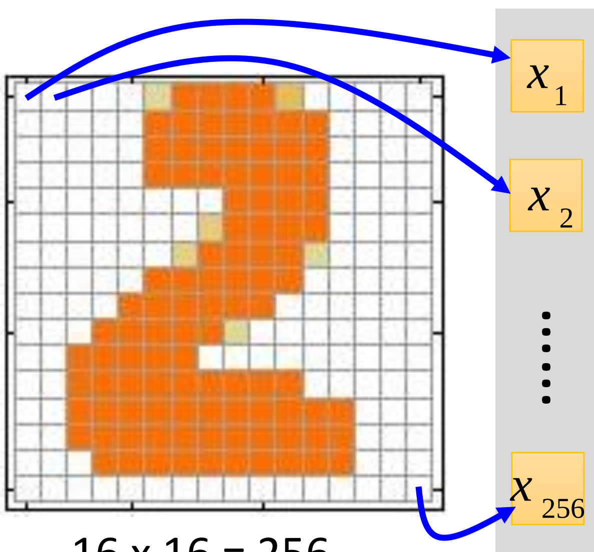
Output Layer as Multi-Class Classifier



Example Application



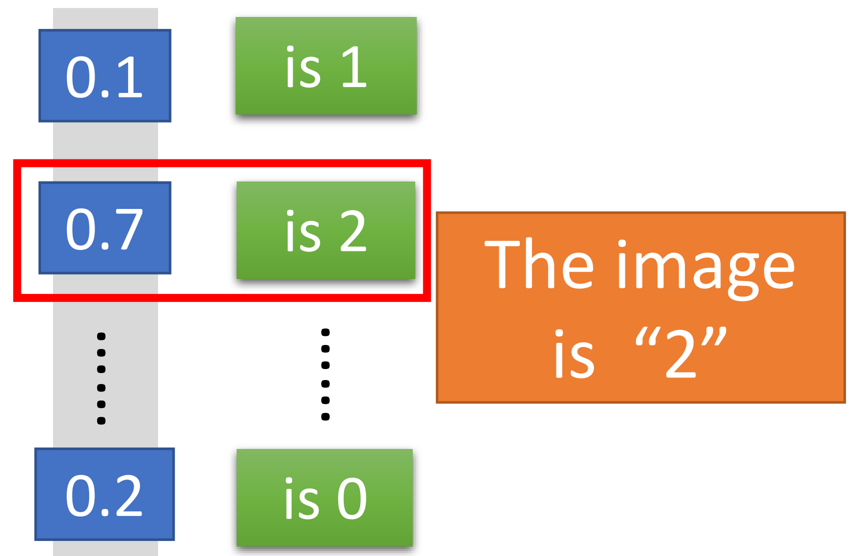
Input



16 x 16 = 256

Ink → 1
No ink → 0

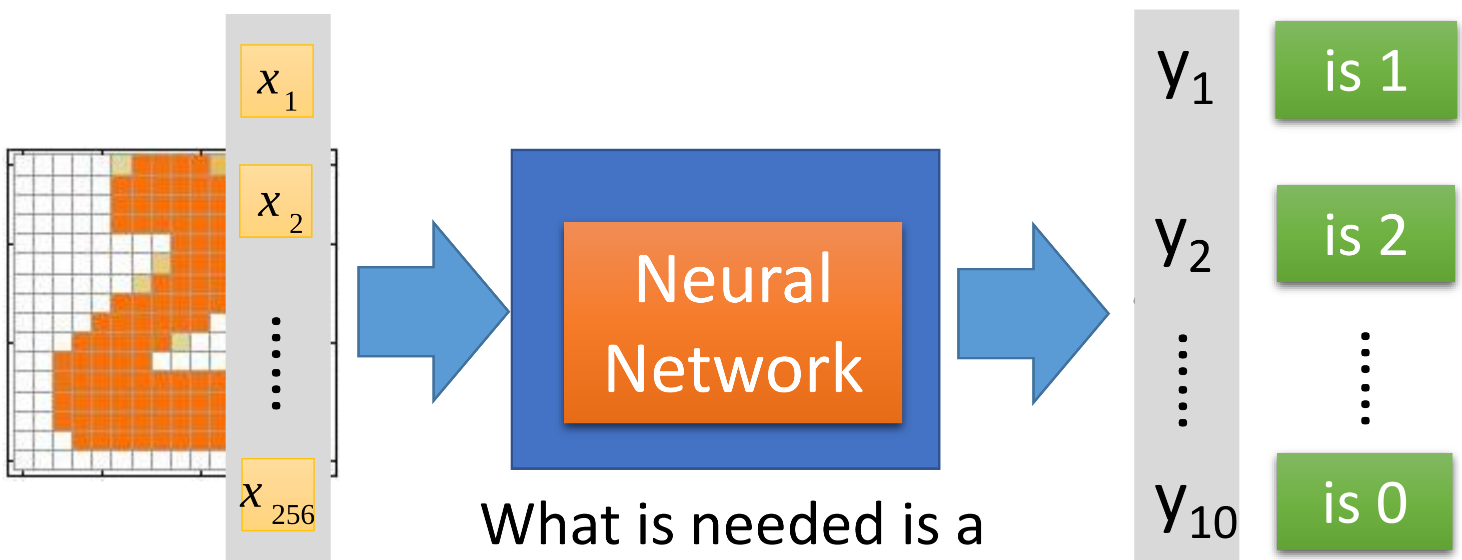
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition

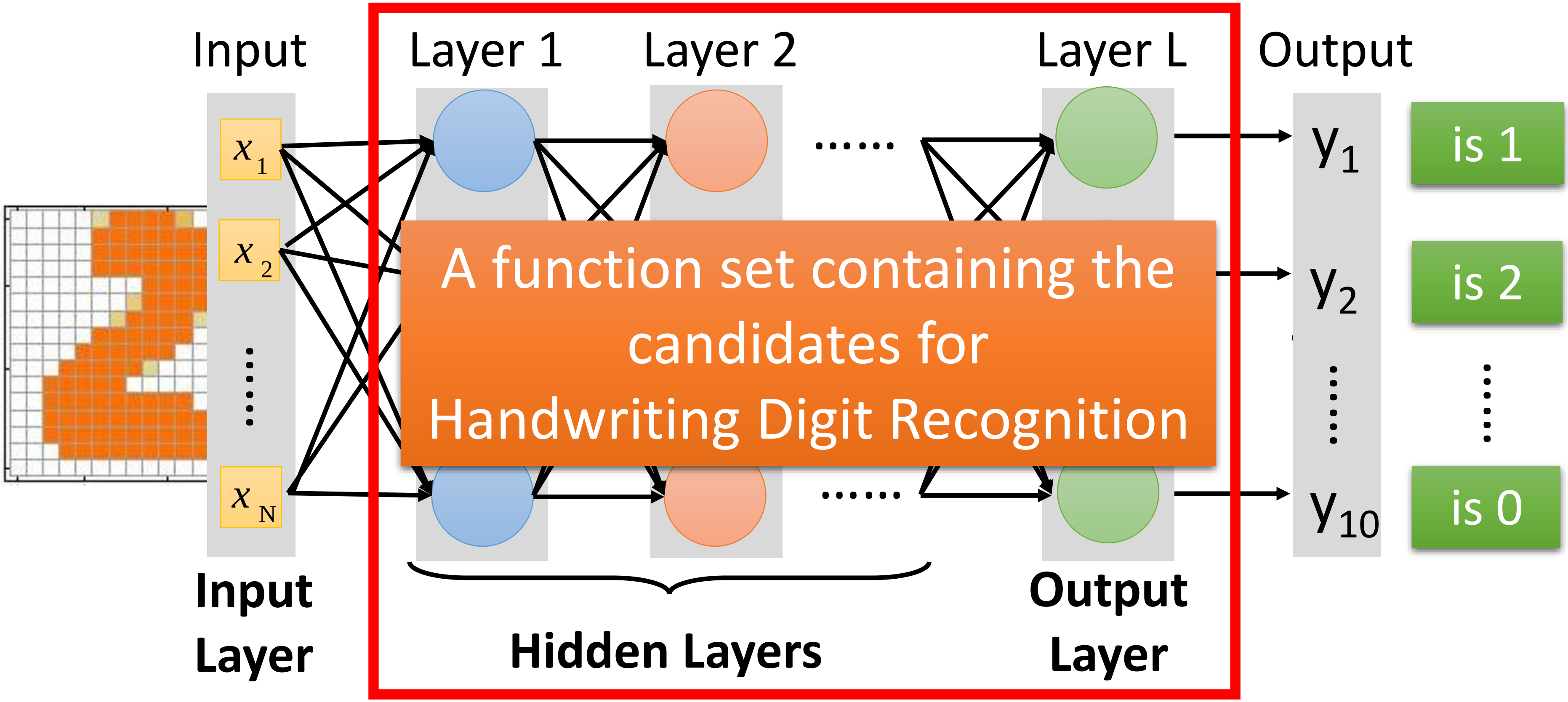


What is needed is a function ……

Input:
256-dim vector

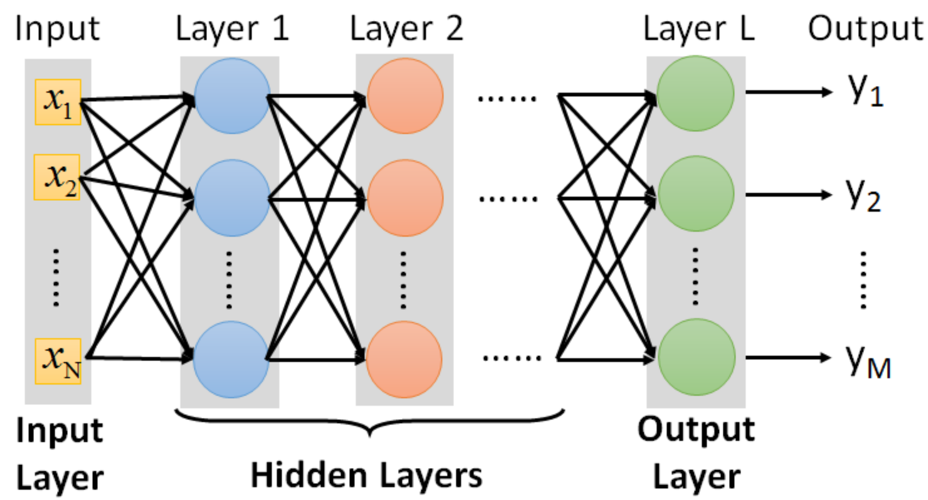
output:
10-dim vector

Example Application



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

FAQ



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

Trial and Error + Intuition

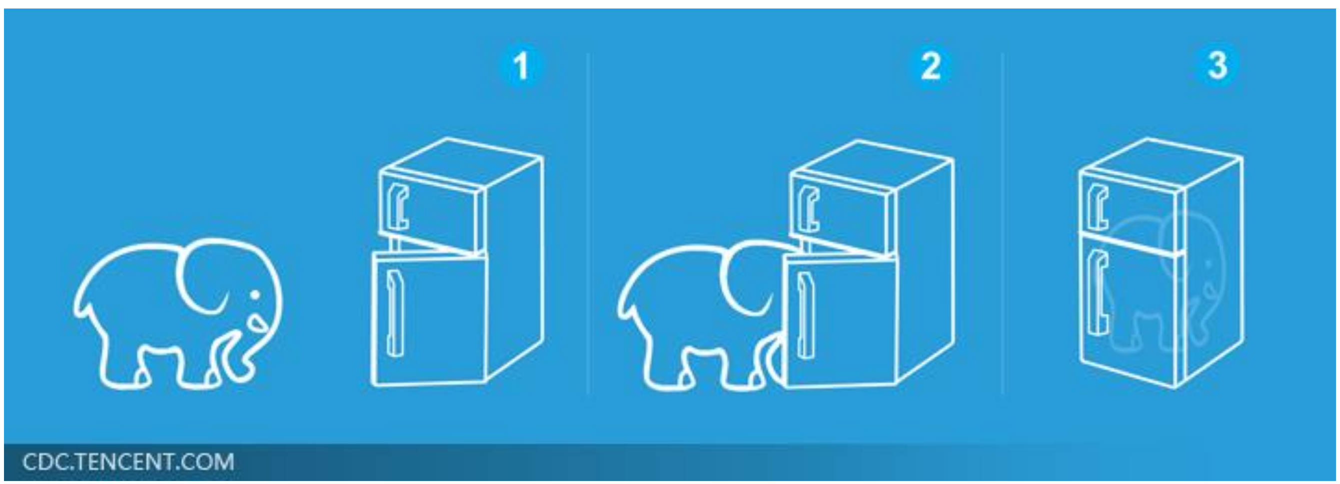
- Q: Can the structure be automatically determined?
  - E.g. Evolutionary Artificial Neural Networks
- Q: Can we design the network structure?

Convolutional Neural Network (CNN)

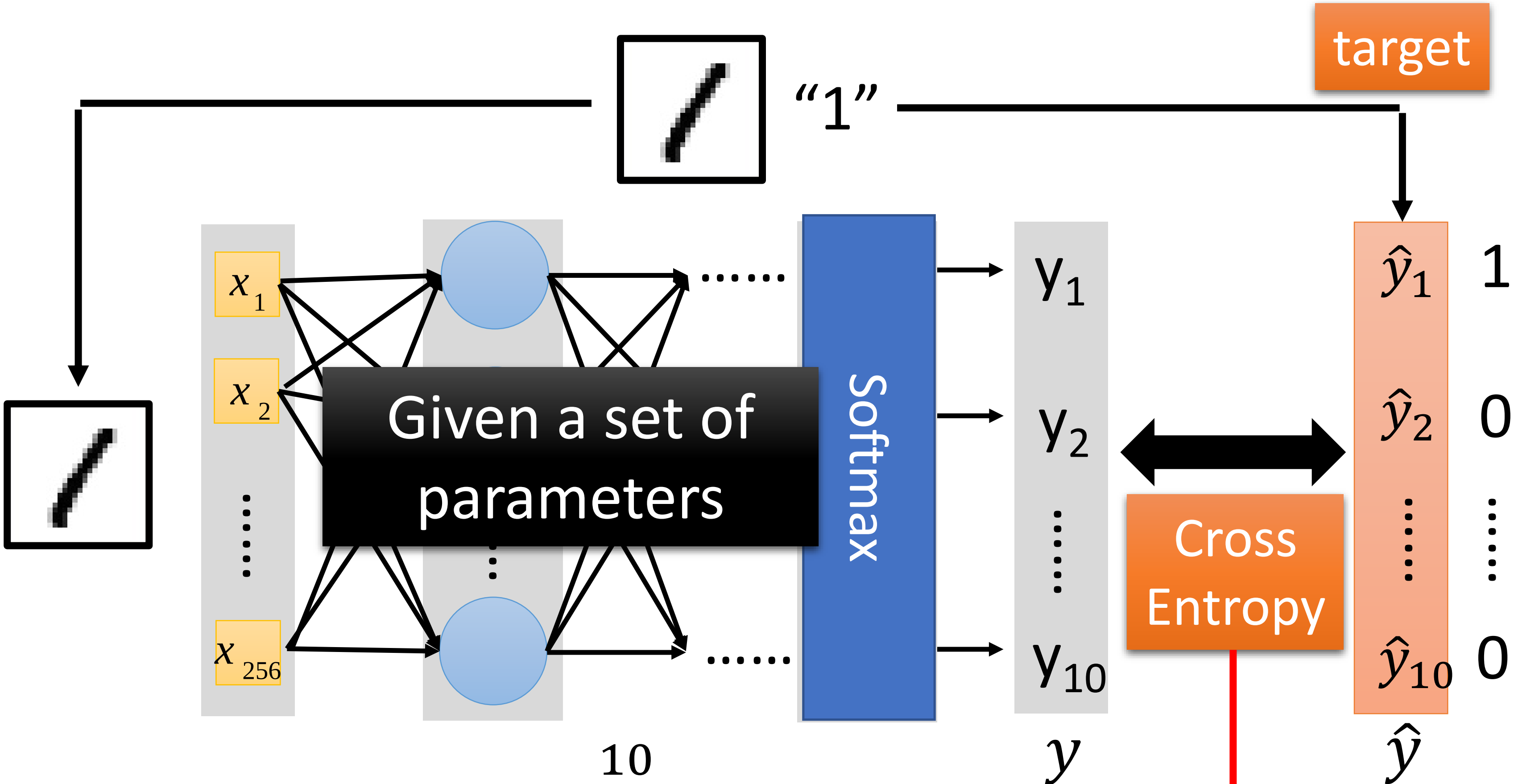
Three Steps for Deep Learning

Step 1: Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



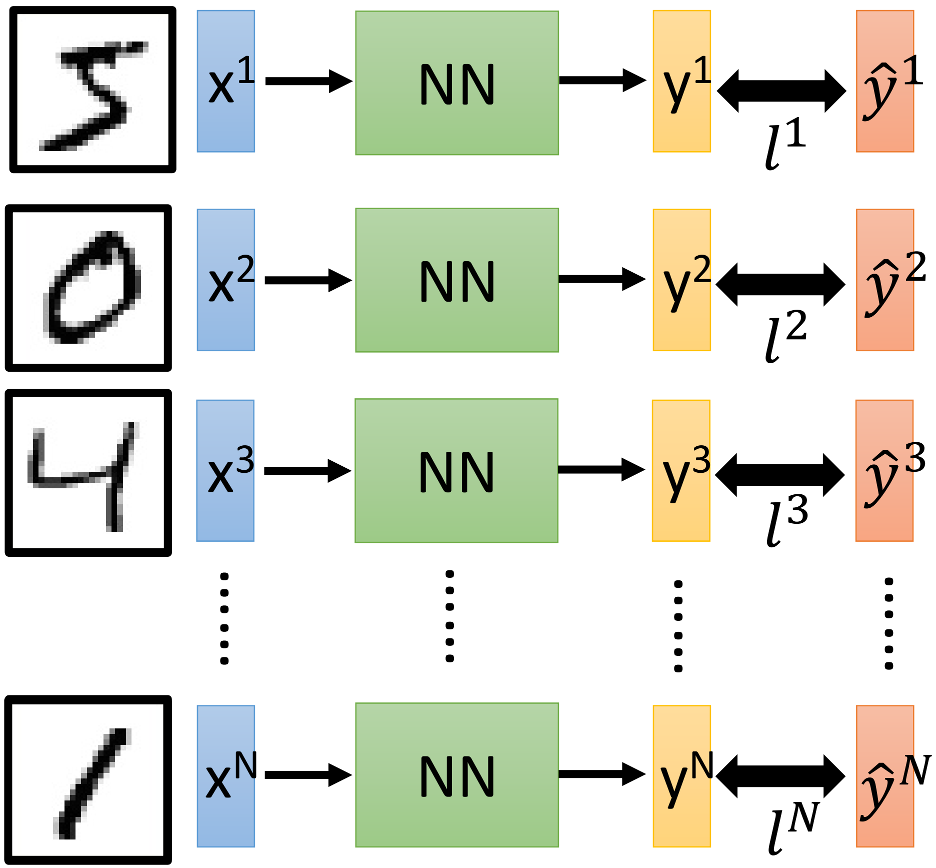
Loss for an Example



$$l(y, \hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

Total Loss

For all training data …



Total Loss:

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

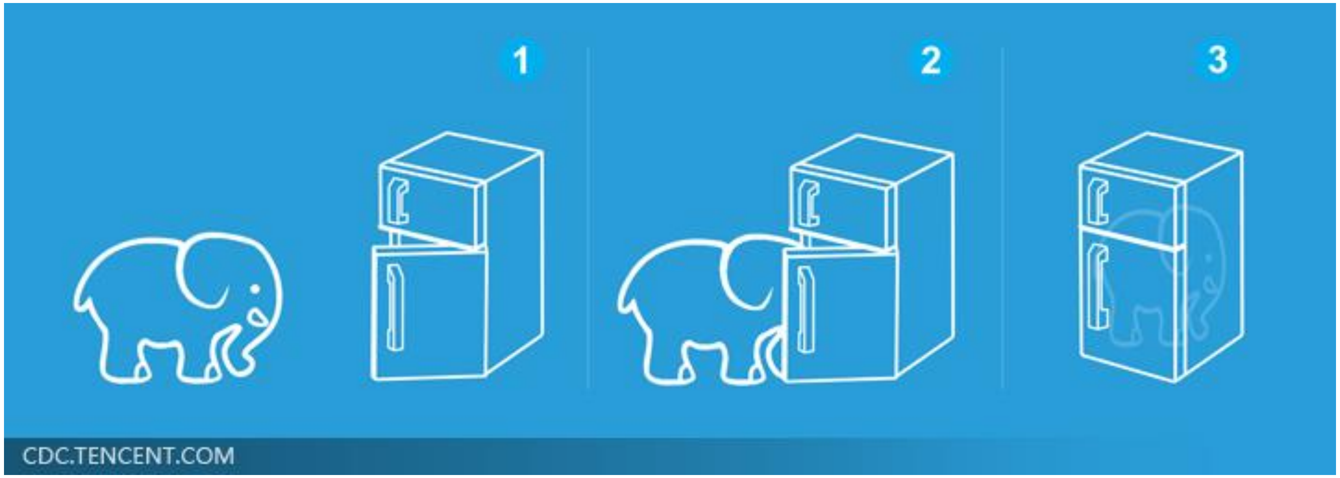
Find ***a function in function set*** that minimizes total loss L

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

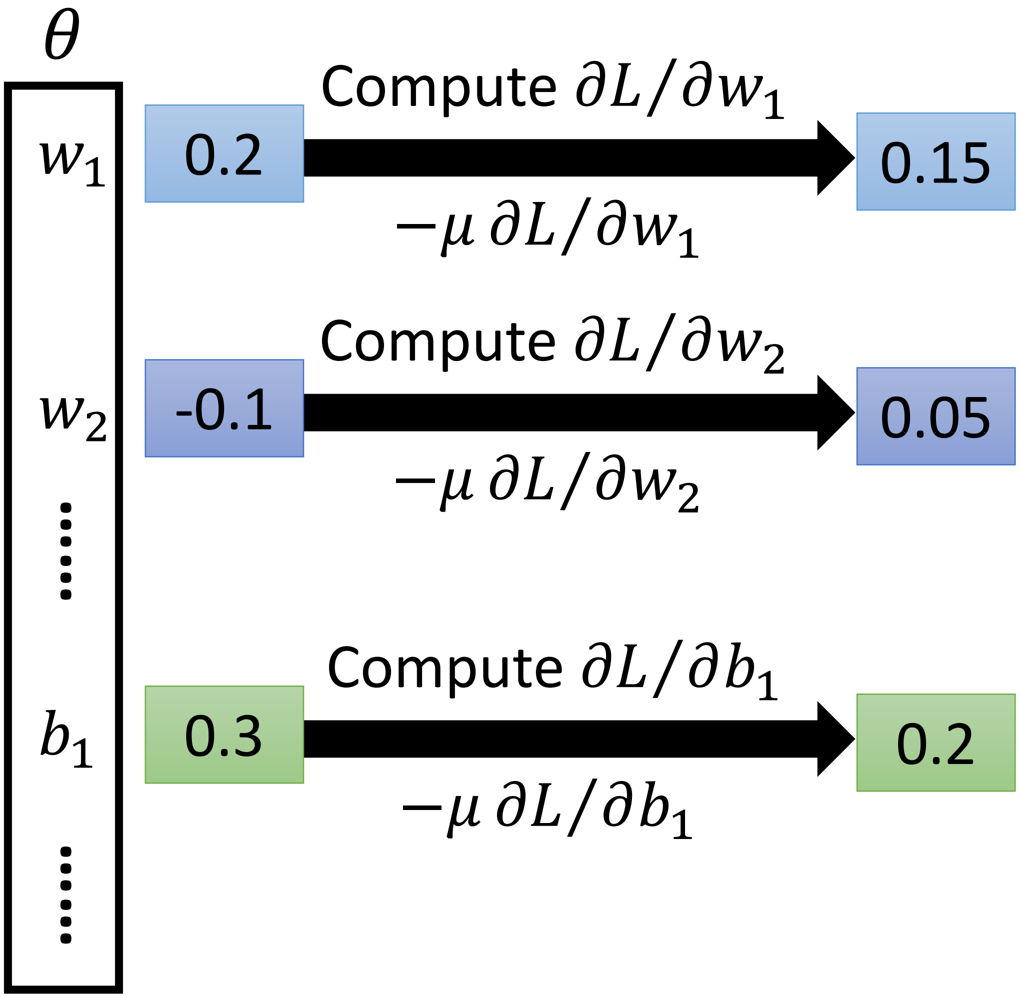
Three Steps for Deep Learning

Step 1: Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



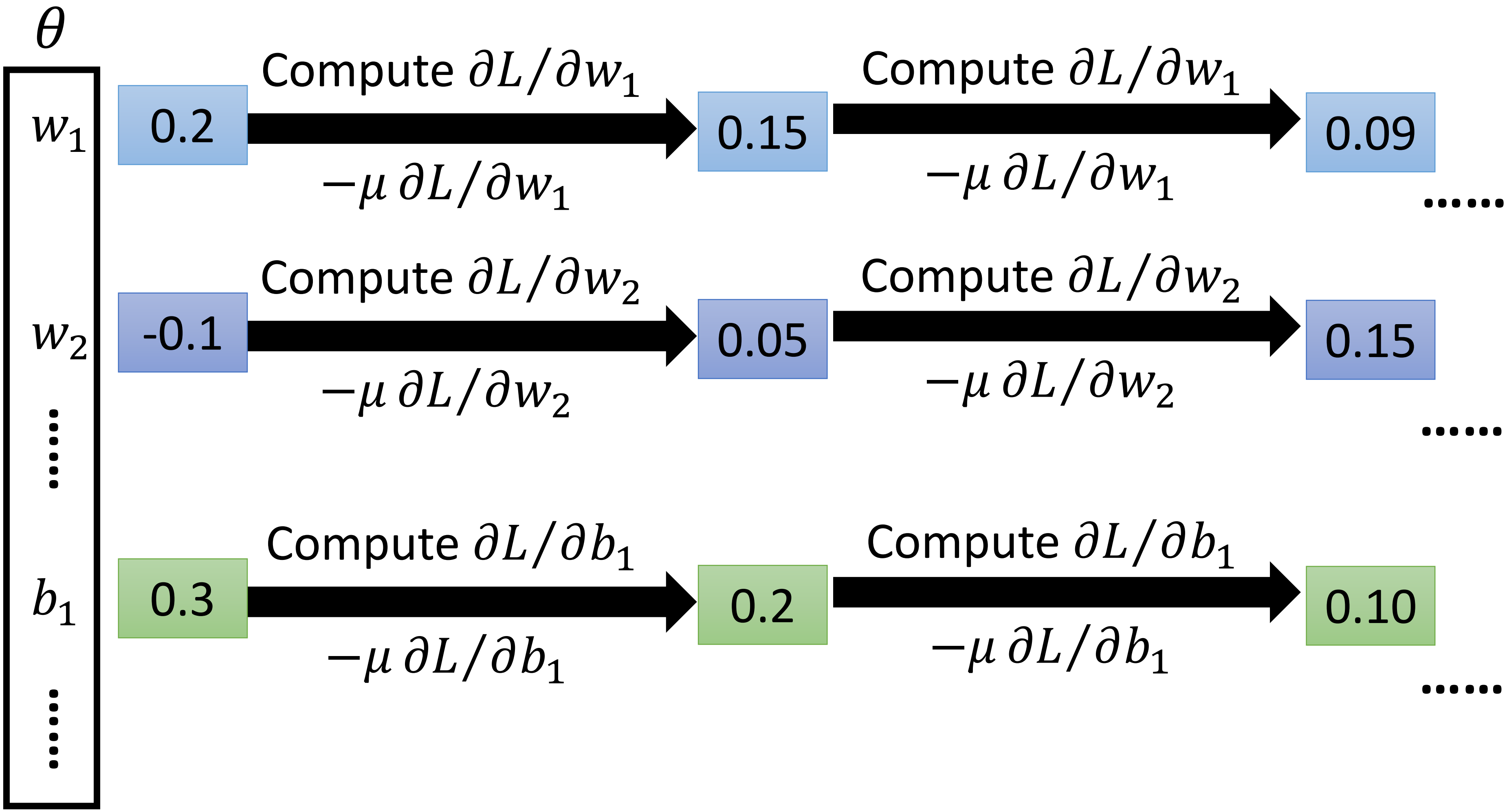
Gradient Descent



$$\nabla L = \begin{bmatrix} \dfrac{\partial L}{\partial w_1} \\ \dfrac{\partial L}{\partial w_2} \\ \vdots \\ \dfrac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

gradient

Gradient Descent

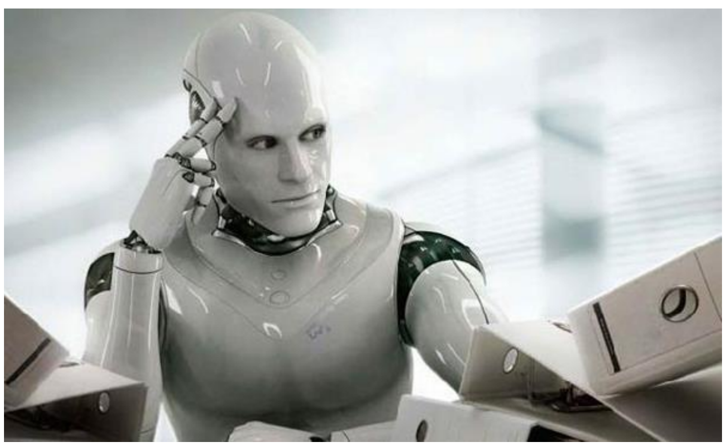


Gradient Descent

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 :p

Backpropagation

- Backpropagation: an efficient way to compute $\partial L / \partial w$ in neural network

TensorFlow

torch

theano

Caffe

Microsoft CNTK

Chainer

Deep Learning library produced by Amazon
DSSTNE

mxnet

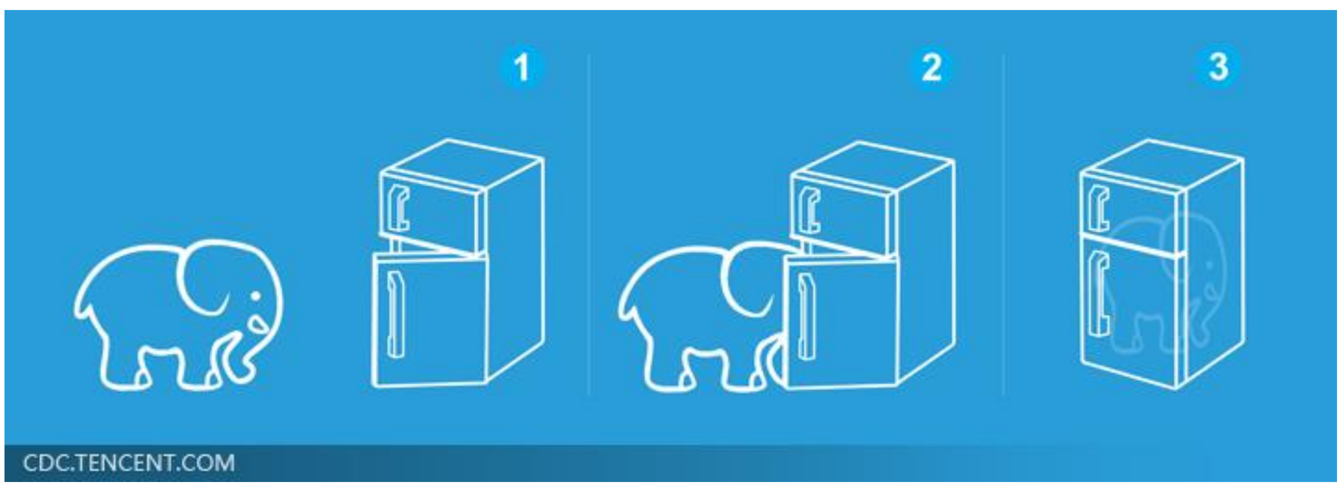
libdnn
台大周伯威
同學開發

Ref:
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html

Three Steps for Deep Learning

Step 1: Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



Gradient Descent

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

$$\text{Starting Parameters} \quad \theta^0 \longrightarrow \theta^1 \longrightarrow \theta^2 \longrightarrow \ldots\ldots$$

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

Compute $\nabla L(\theta^0)$ $\quad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$

Compute $\nabla L(\theta^1)$ $\quad \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

Millions of parameters ……

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

Chain Rule

Case 1

$$y = g(x) \qquad z = h(y)$$

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

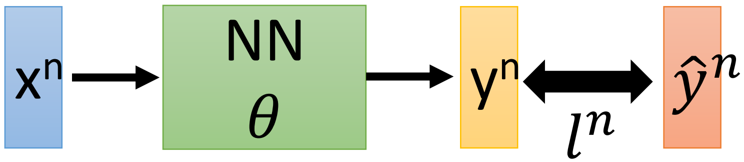
Case 2

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

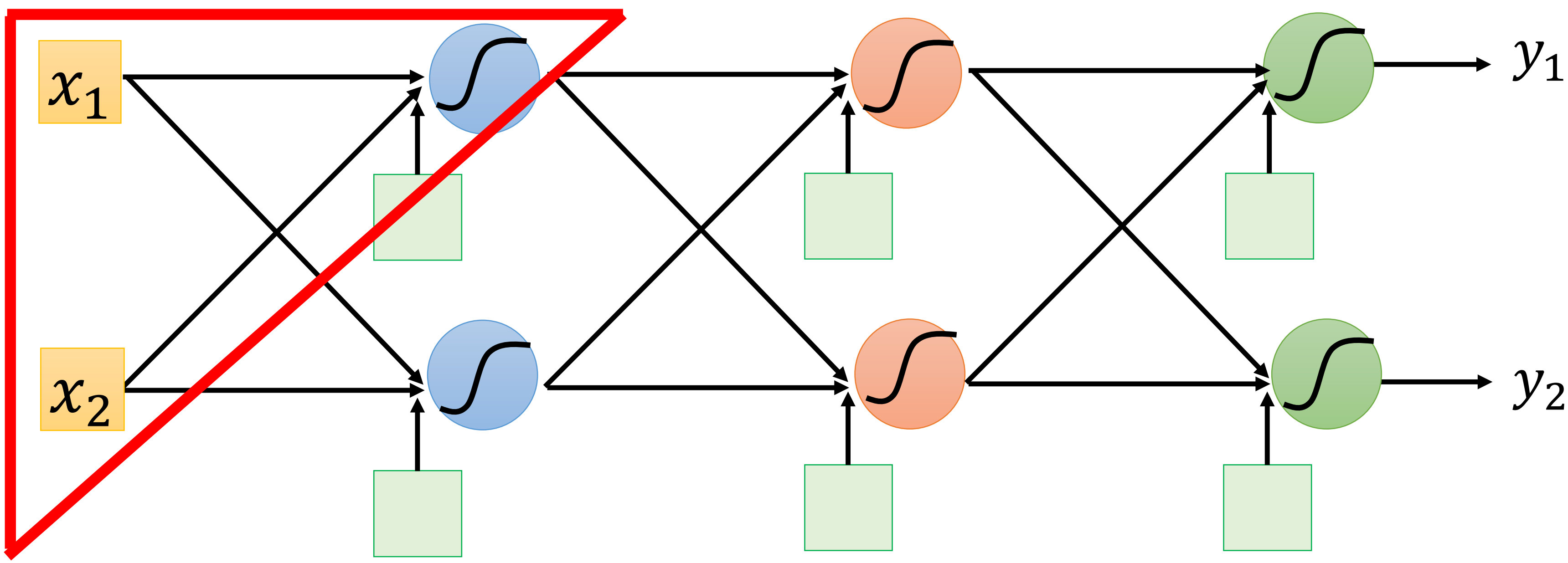
$$\begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd}$$

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

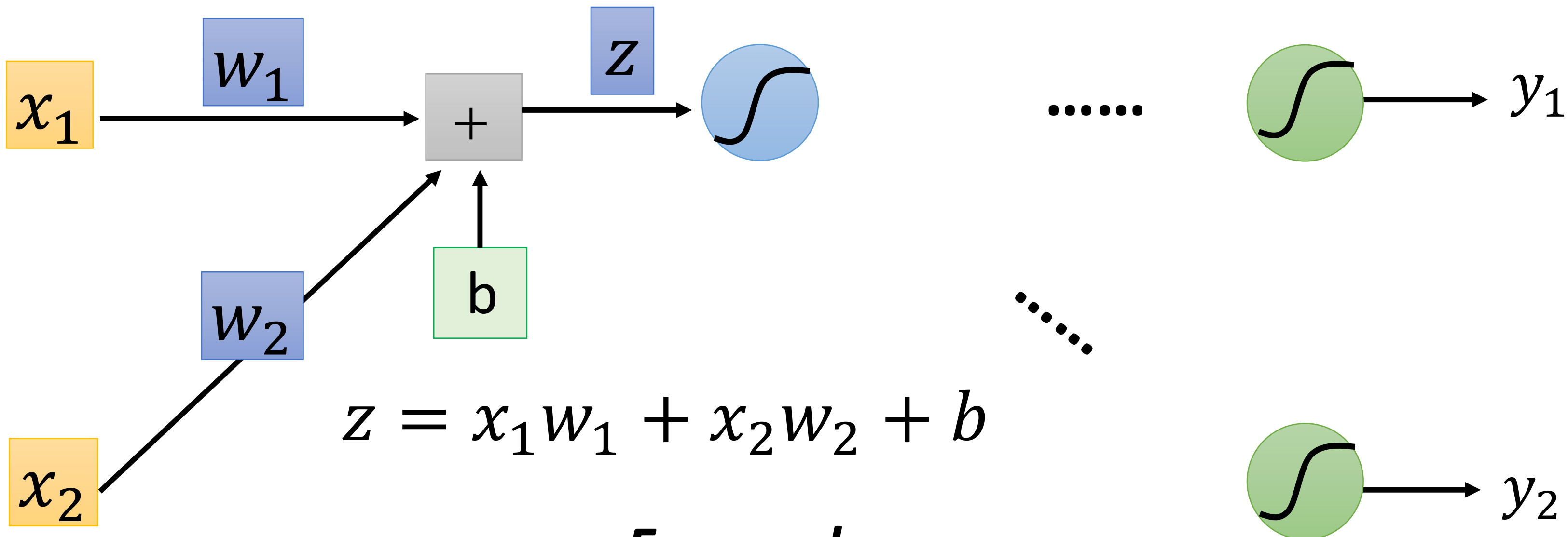
Backpropagation



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



Backpropagation



$$z = x_1 w_1 + x_2 w_2 + b$$

$$\frac{\partial l}{\partial w} = ? \quad \frac{\partial z}{\partial w}\frac{\partial l}{\partial z}$$

(Chain rule)

Forward pass:

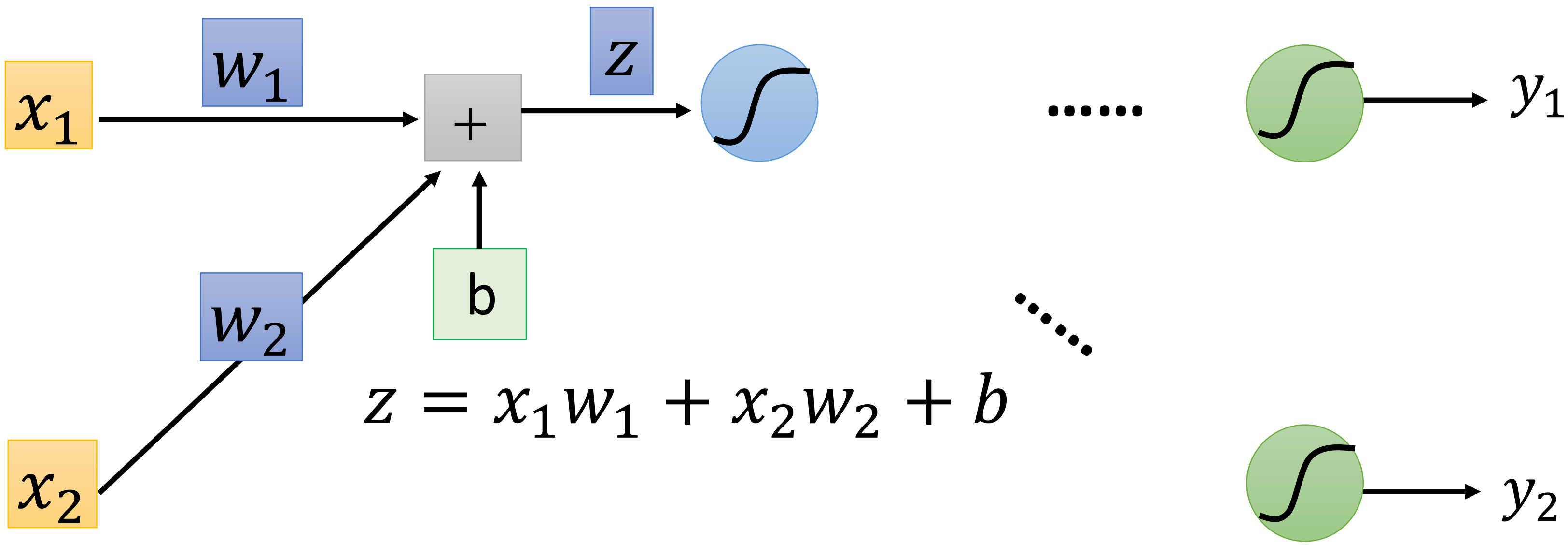
Compute $\partial z / \partial w$ for all parameters

Backward pass:

Compute $\partial l / \partial z$ for all activation function inputs z

Backpropagation – Forward pass

Compute $\partial z / \partial w$ for all parameters



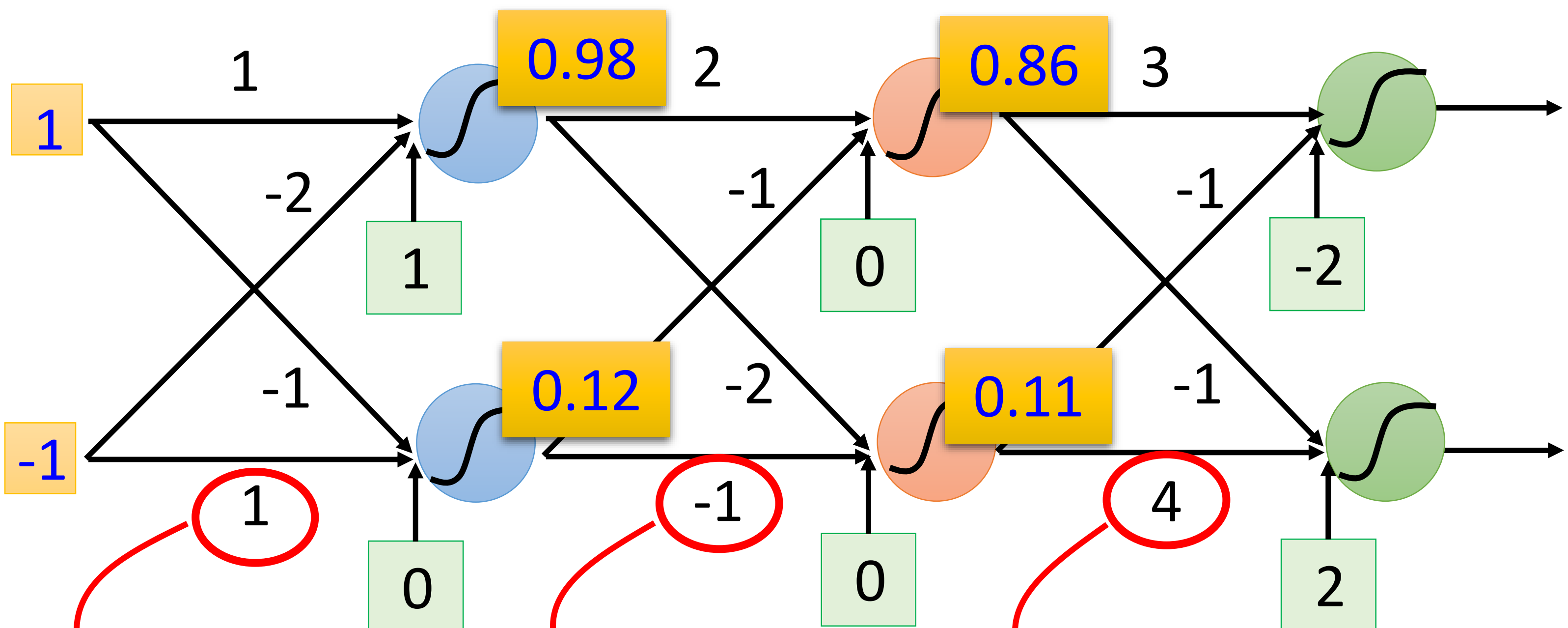
$$z = x_1 w_1 + x_2 w_2 + b$$

$$\left.\begin{array}{l} \partial z / \partial w_1 = ? \;\; x_1 \\ \partial z / \partial w_2 = ? \;\; x_2 \end{array}\right\}$$

The value of the input connected by the weight

Backpropagation – Forward pass

Compute $\partial z / \partial w$ for all parameters



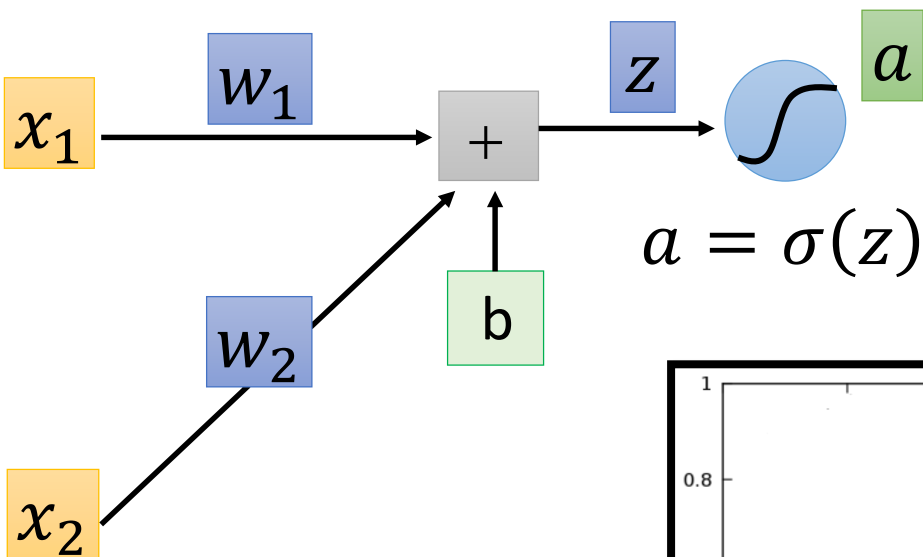
$$\frac{\partial z}{\partial w} = -1$$

$$\frac{\partial z}{\partial w} = 0.12$$

$$\frac{\partial z}{\partial w} = 0.11$$

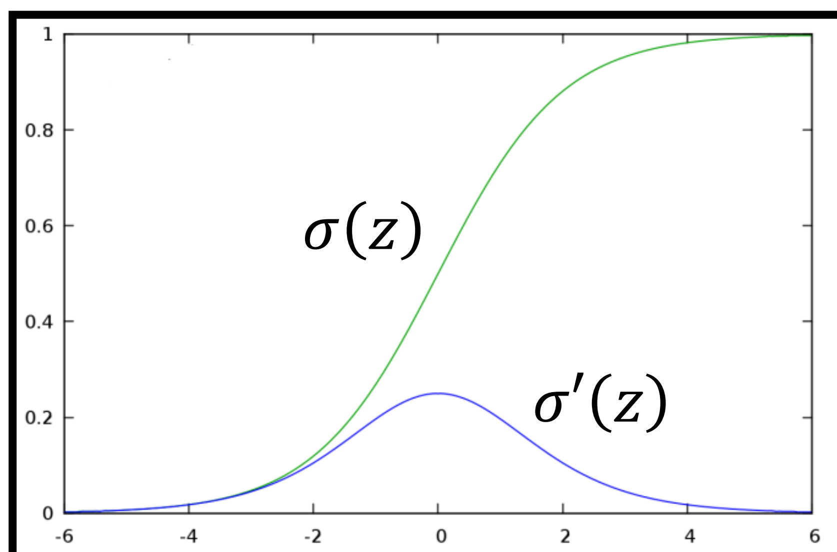
Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z



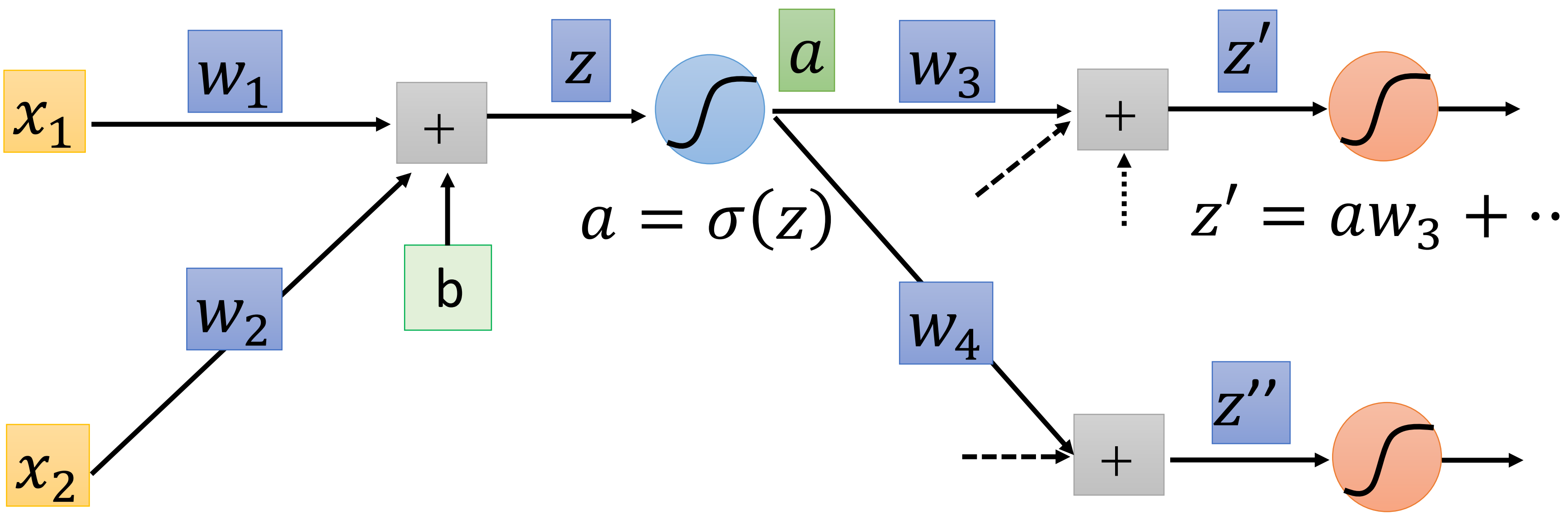
$$\frac{\partial l}{\partial z} = \underline{\frac{\partial a}{\partial z}} \frac{\partial l}{\partial a}$$

$\Rightarrow \sigma'(z)$



Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z



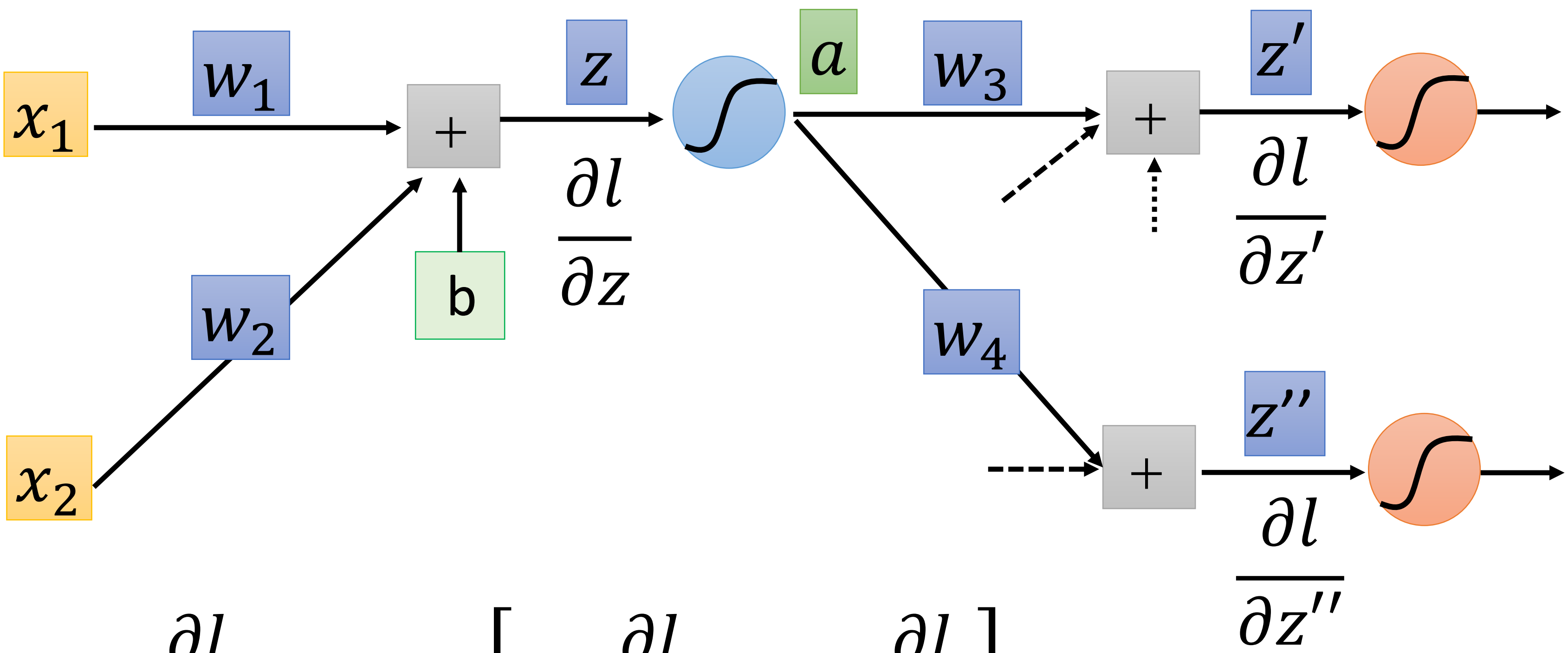
$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial l}{\partial a} \qquad \frac{\partial l}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial l}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial l}{\partial z''}$$ (Chain rule)

w_3 ? w_4 ?

Assumed it's known

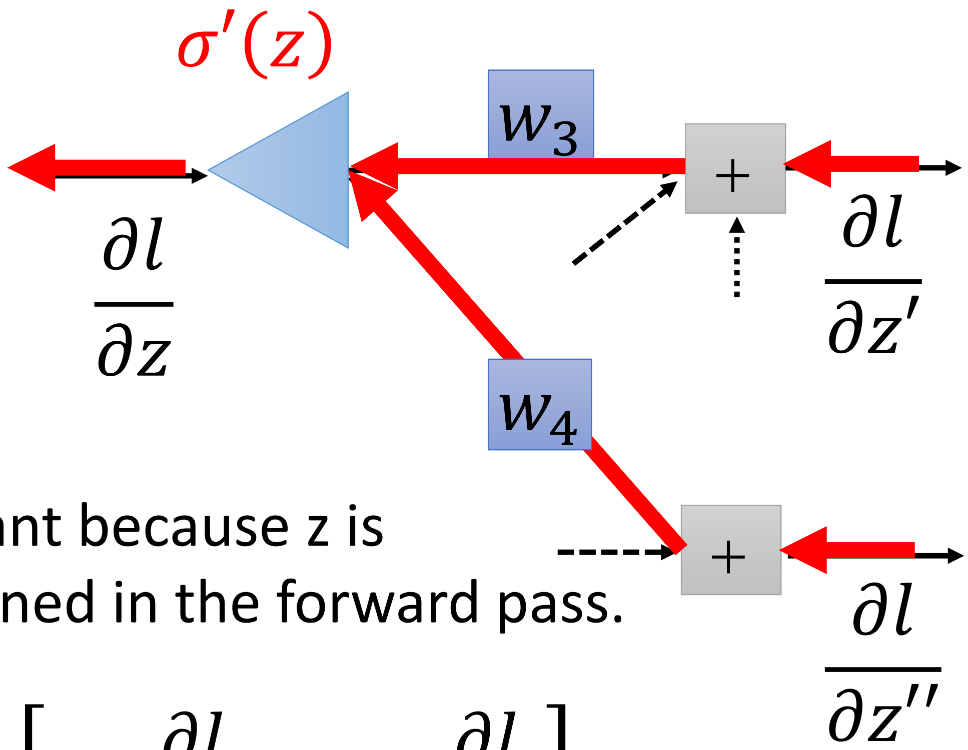
Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



$$\frac{\partial l}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''}\right]$$

Backpropagation – Backward pass

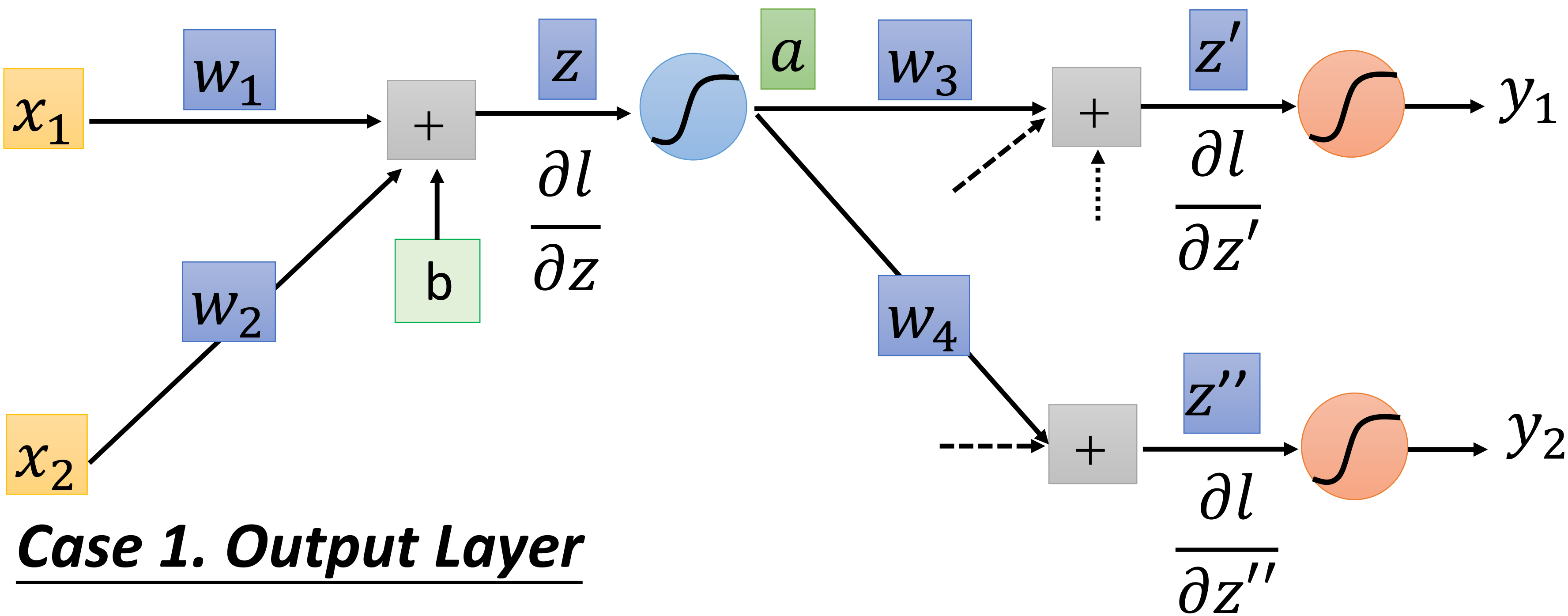


$\sigma'(z)$ is a constant because z is already determined in the forward pass.

$$\frac{\partial l}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''}\right]$$

Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



Case 1. Output Layer

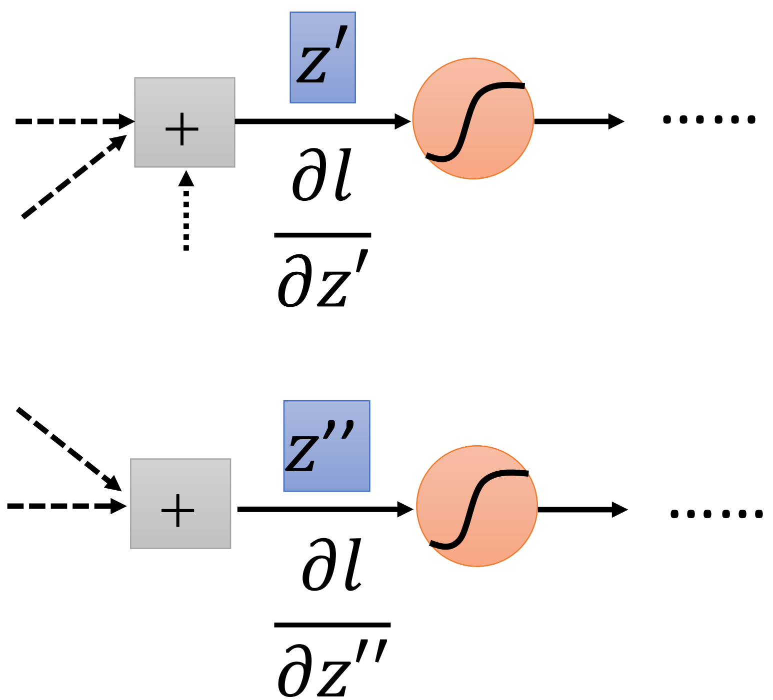
$$\frac{\partial l}{\partial z'} = \frac{\partial y_1}{\partial z'}\frac{\partial l}{\partial y_1} \qquad \frac{\partial l}{\partial z''} = \frac{\partial y_2}{\partial z''}\frac{\partial l}{\partial y_2}$$

Done!

Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z

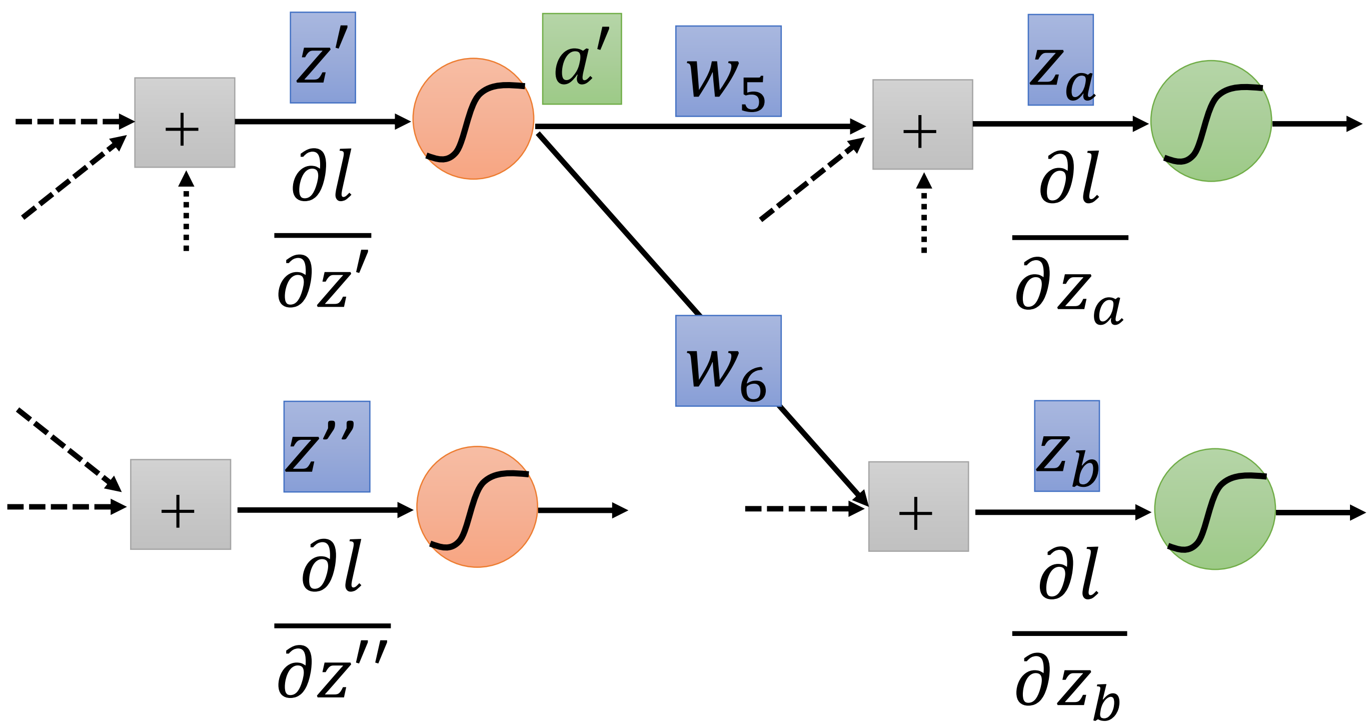
Case 2. Not Output Layer



Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z

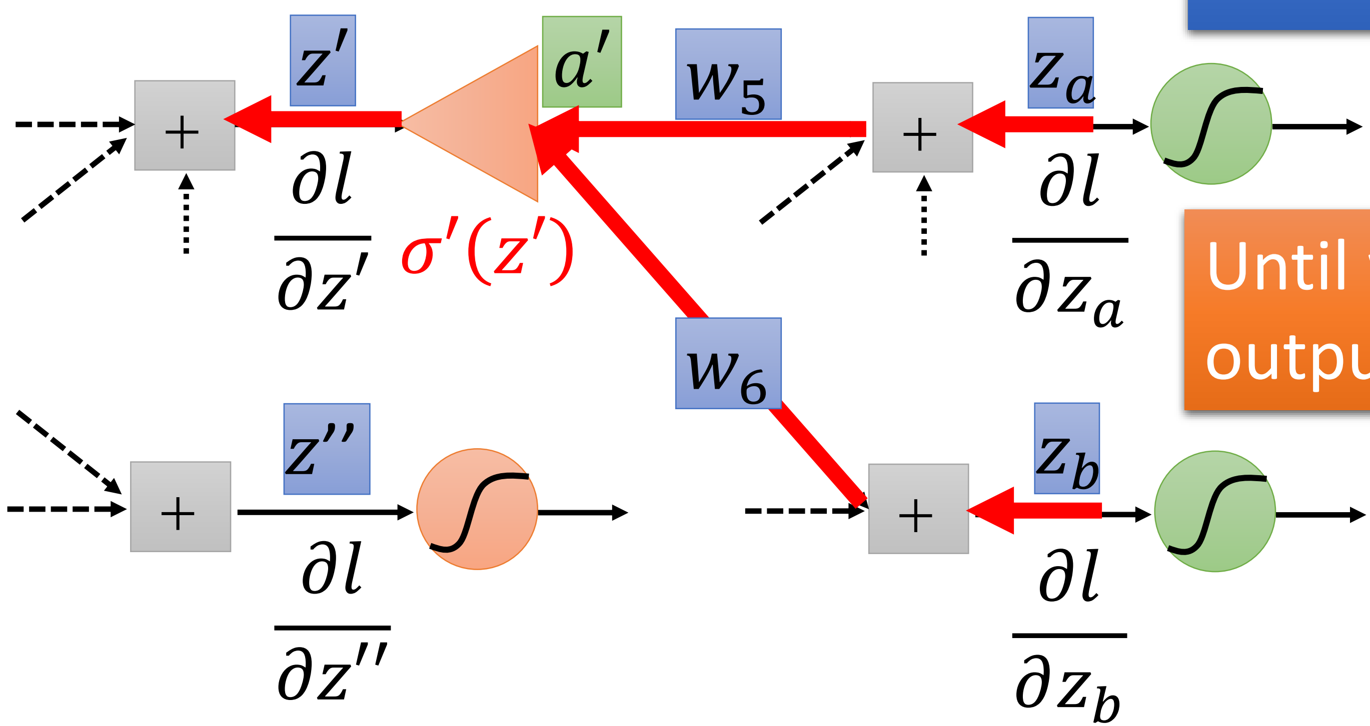
<u>Case 2. Not Output Layer</u>



Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z

Case 2. Not Output Layer



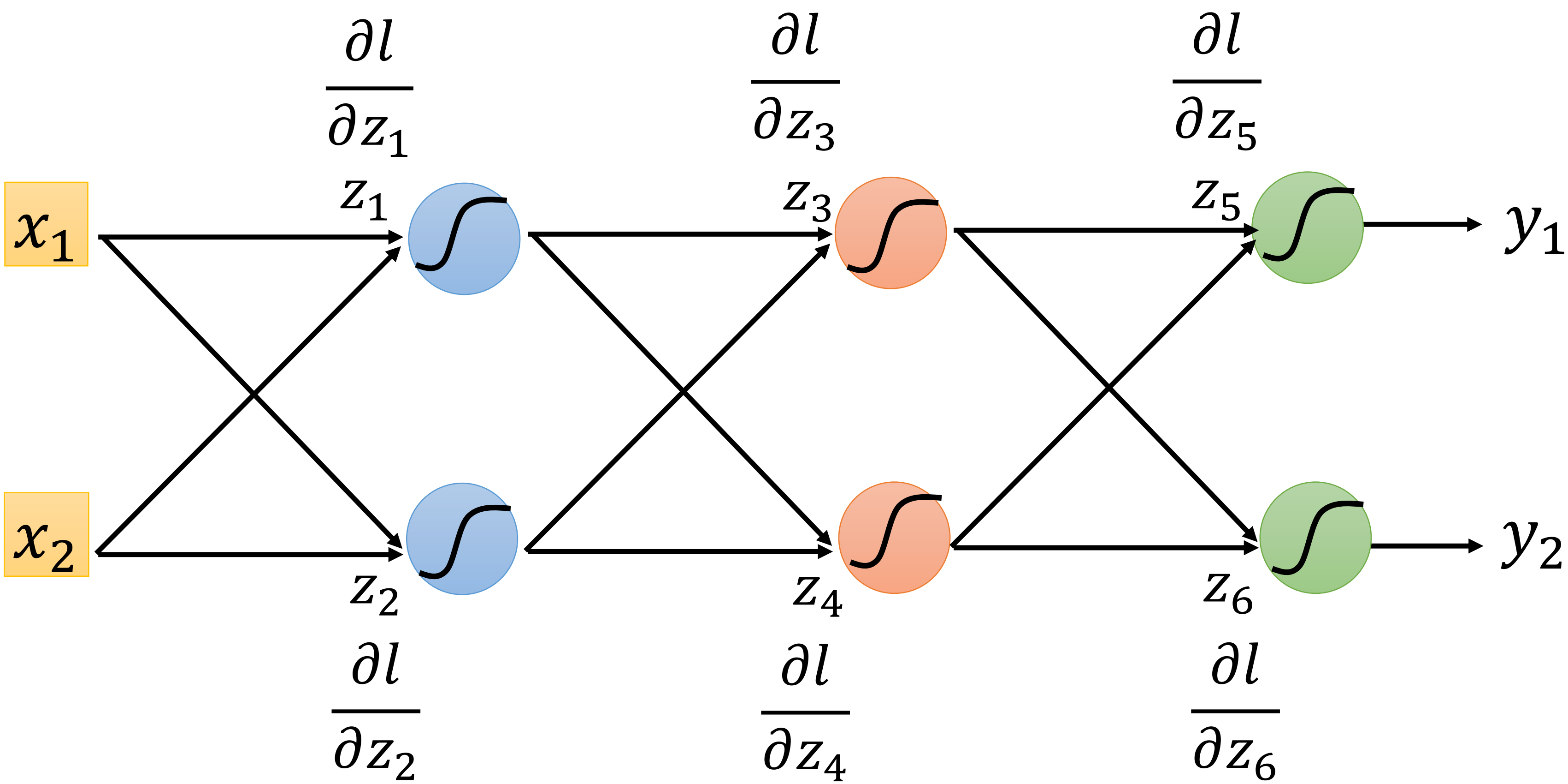
Compute $\partial l / \partial z$ recursively

Until we reach the output layer ……

Backpropagation – Backward Pass

Compute $\partial l / \partial z$ for all activation function inputs z

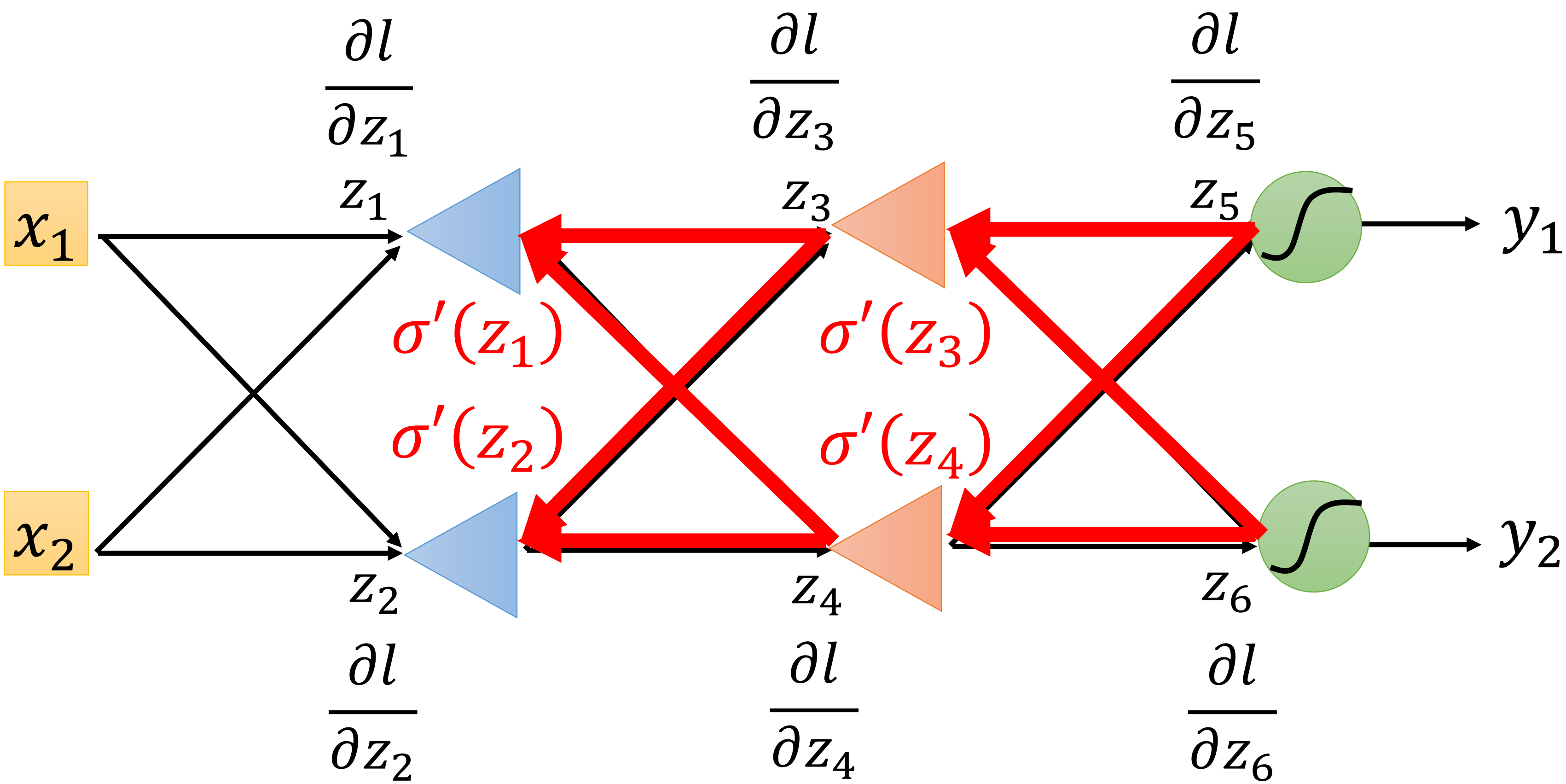
Compute $\partial l / \partial z$ from the output layer



Backpropagation – Backward Pass

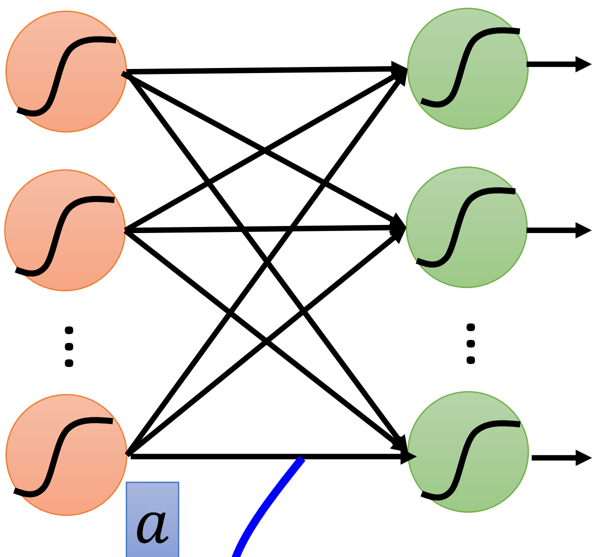
Compute $\partial l/\partial z$ for all activation function inputs z

Compute $\partial l/\partial z$ from the output layer

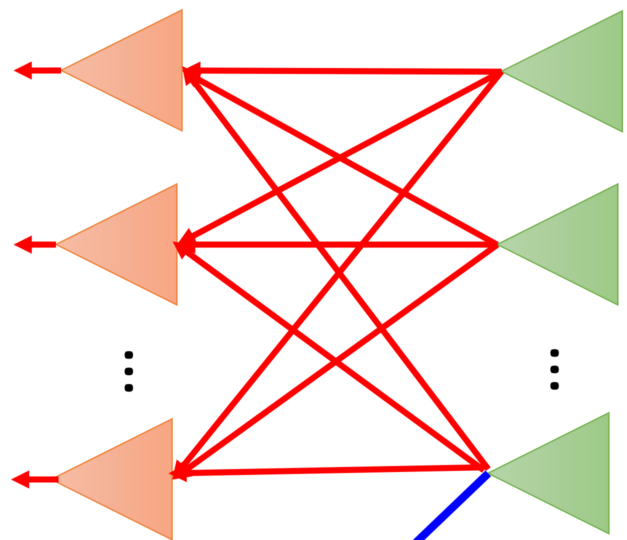


Backpropagation – Summary

Forward Pass



Backward Pass



$$\frac{\partial z}{\partial w} = a \quad \times \quad \frac{\partial l}{\partial z} \quad = \frac{\partial l}{\partial w}$$

for all w

Acknowledgment

- 感謝 Victor Chen 發現投影片上的打字錯誤