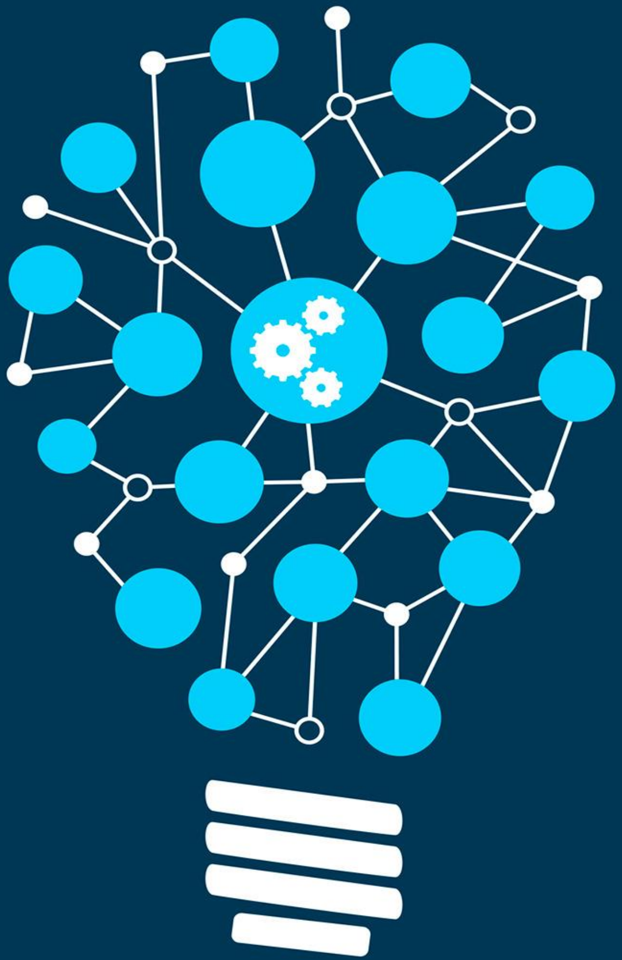
人工智能技术及应用

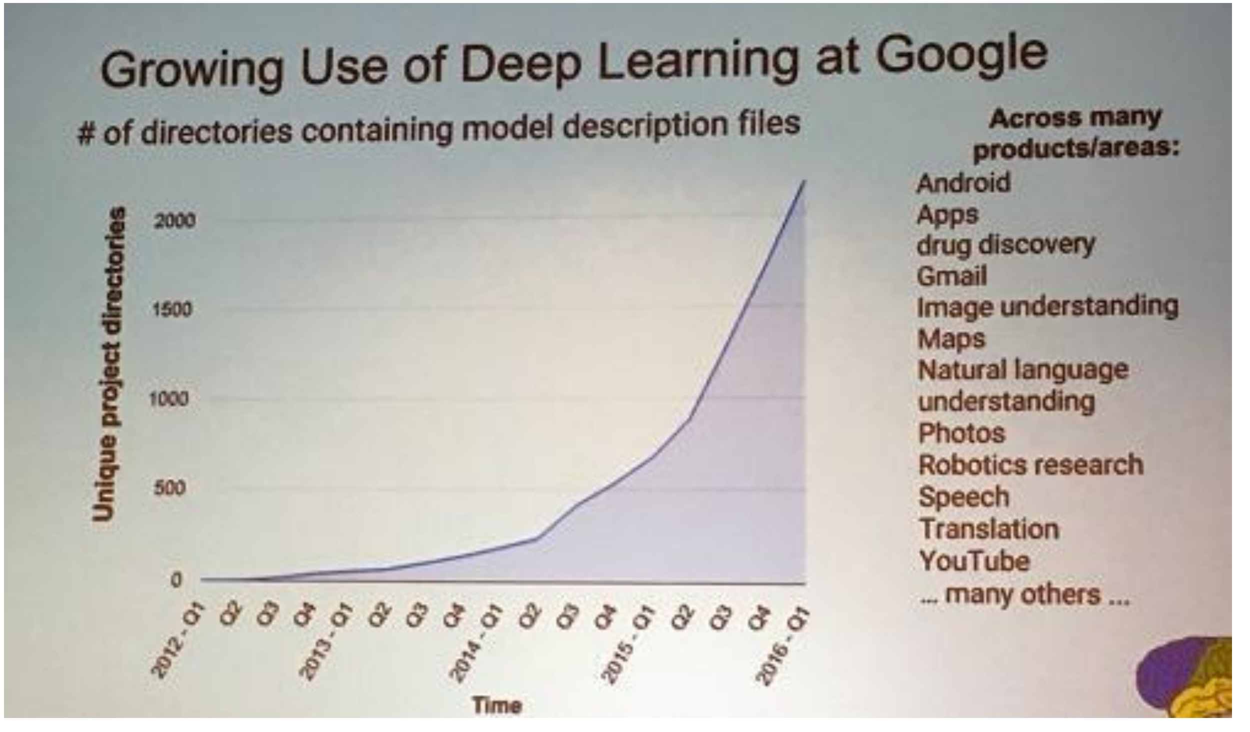
Artificial Intelligence and Application

Deep learning



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

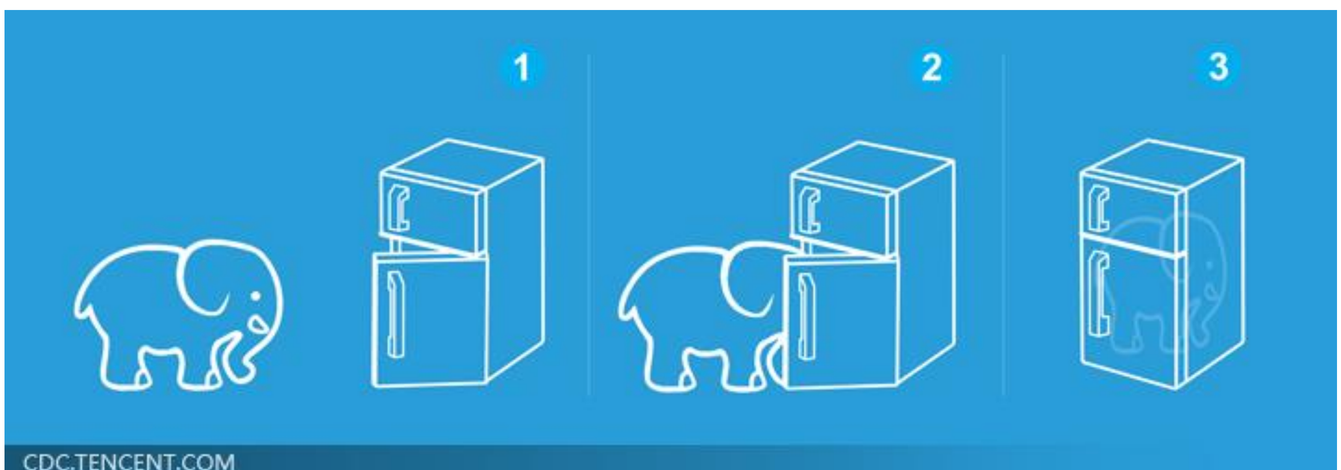
Ups and downs of Deep Learning

- 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

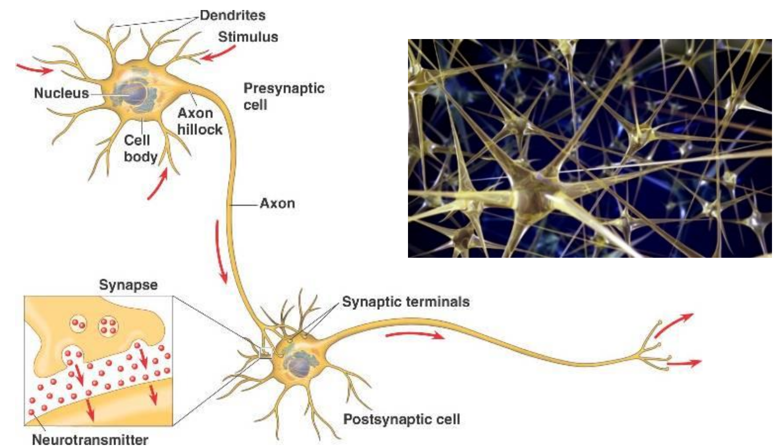
Three Steps for Deep Learning

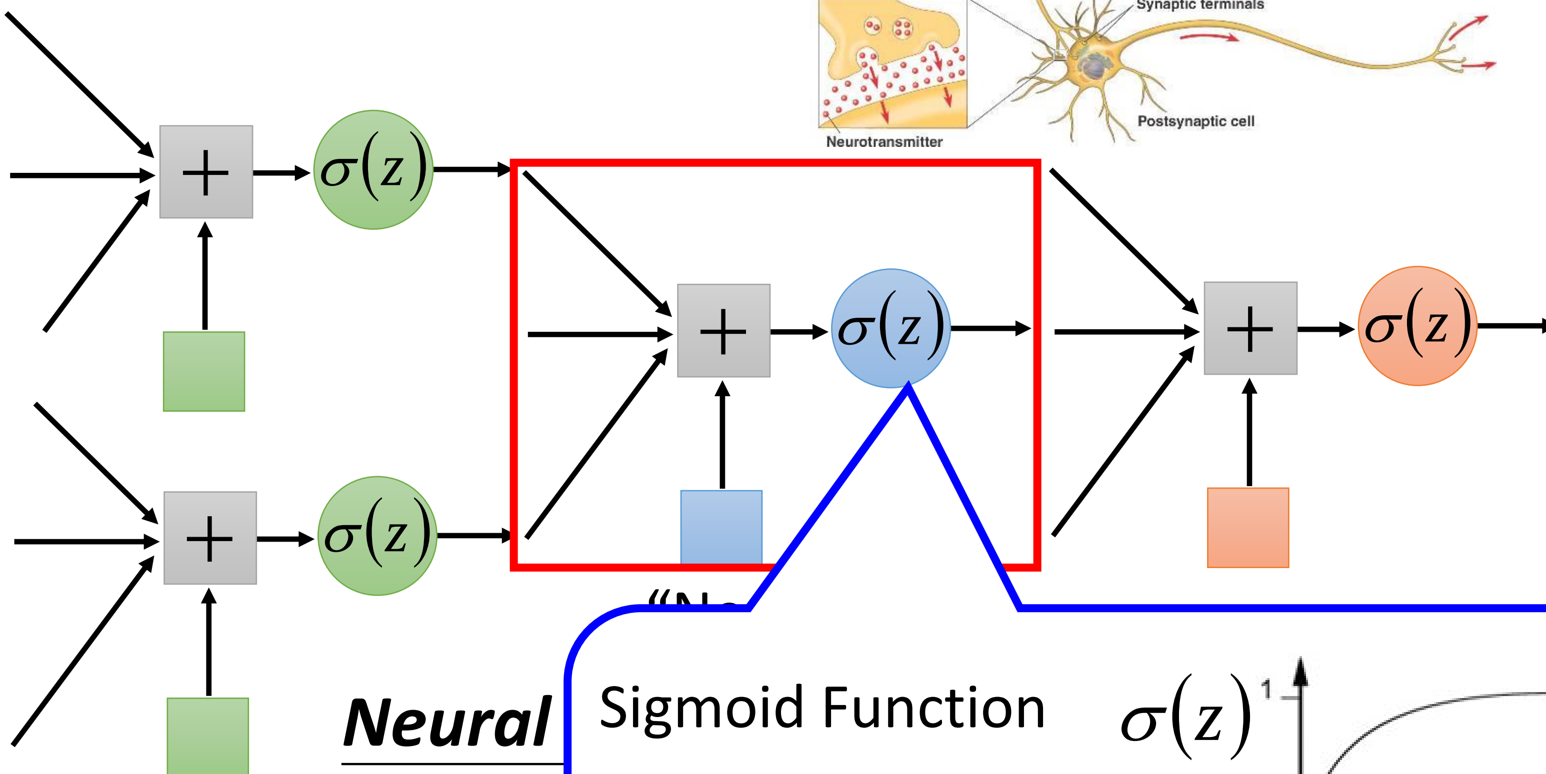
Step 1: Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



Neural Network





“Ne

Neural

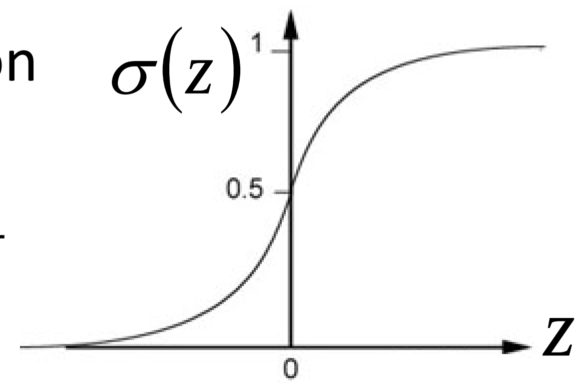
Different

structure

Network parameter θ: all

Sigmoid Function

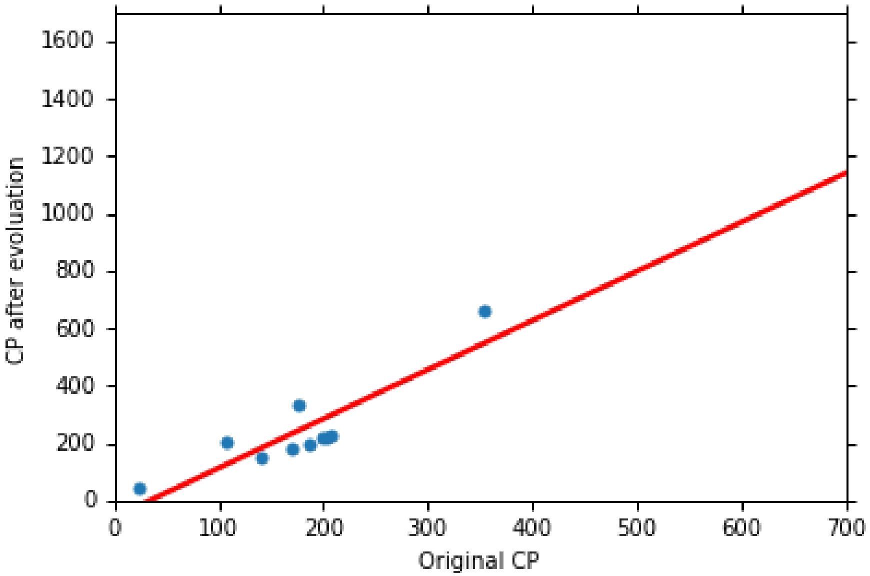
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Parallel Universes

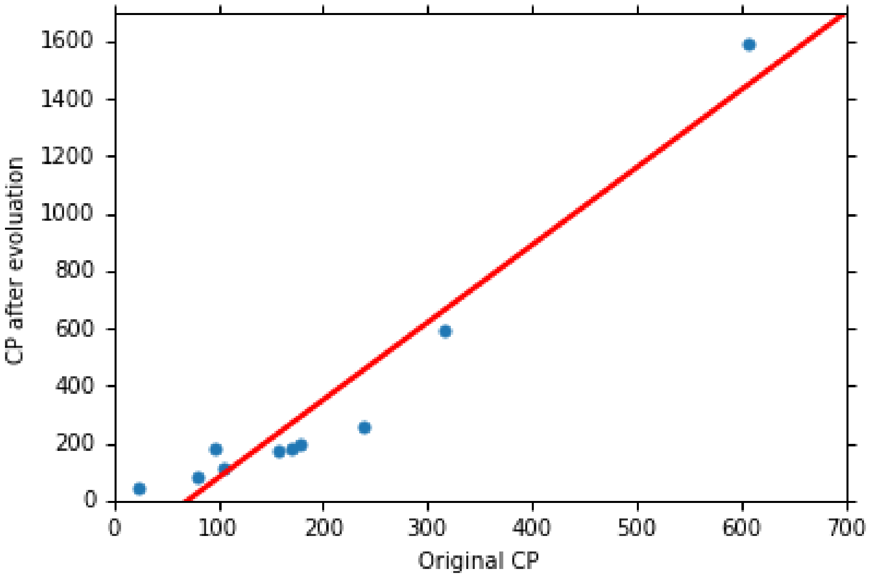
- In different universes, we use the same model, but obtain different f^*

Universe 123



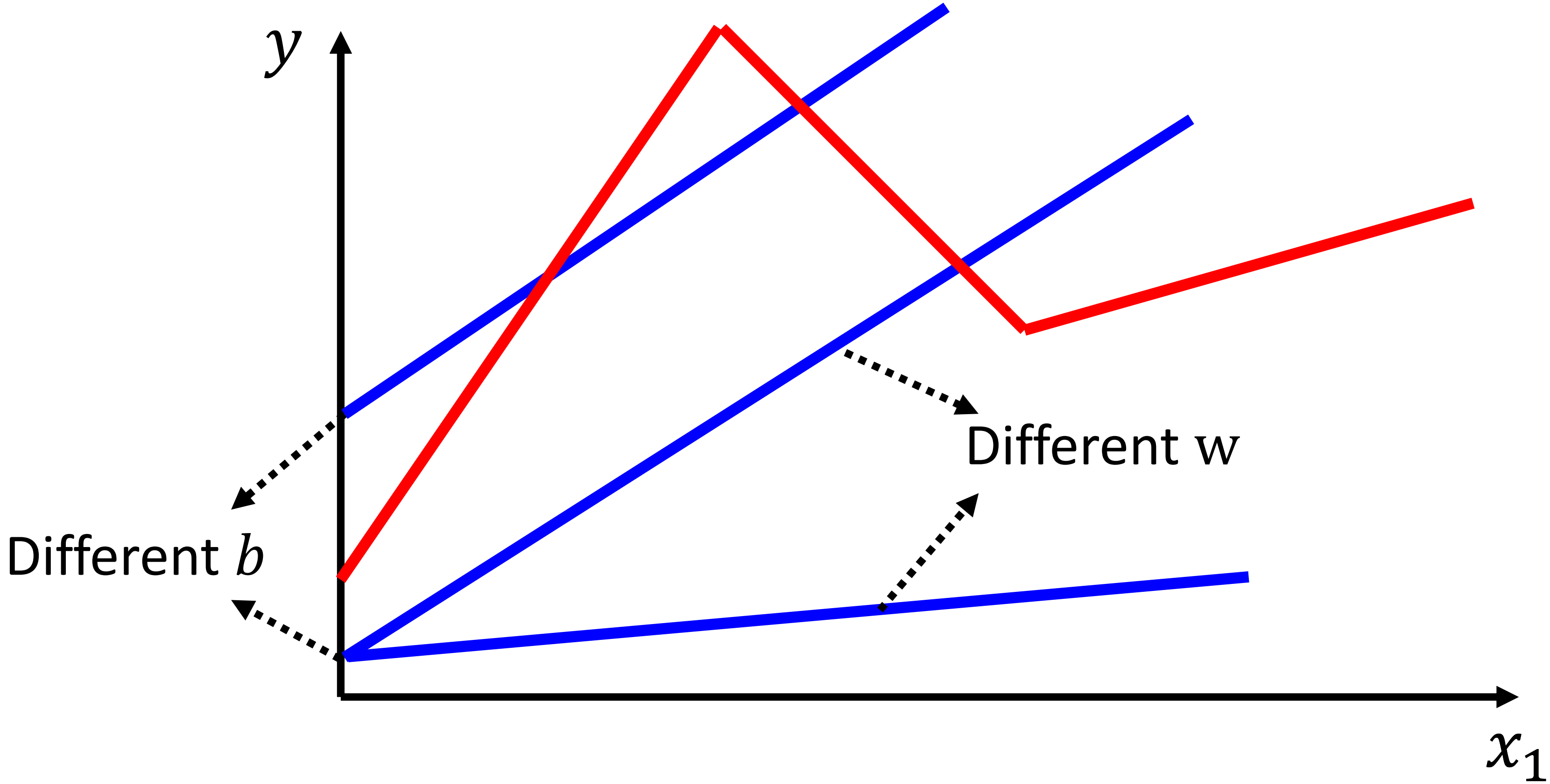
$$y = b + w \cdot x_{cp}$$

Universe 345



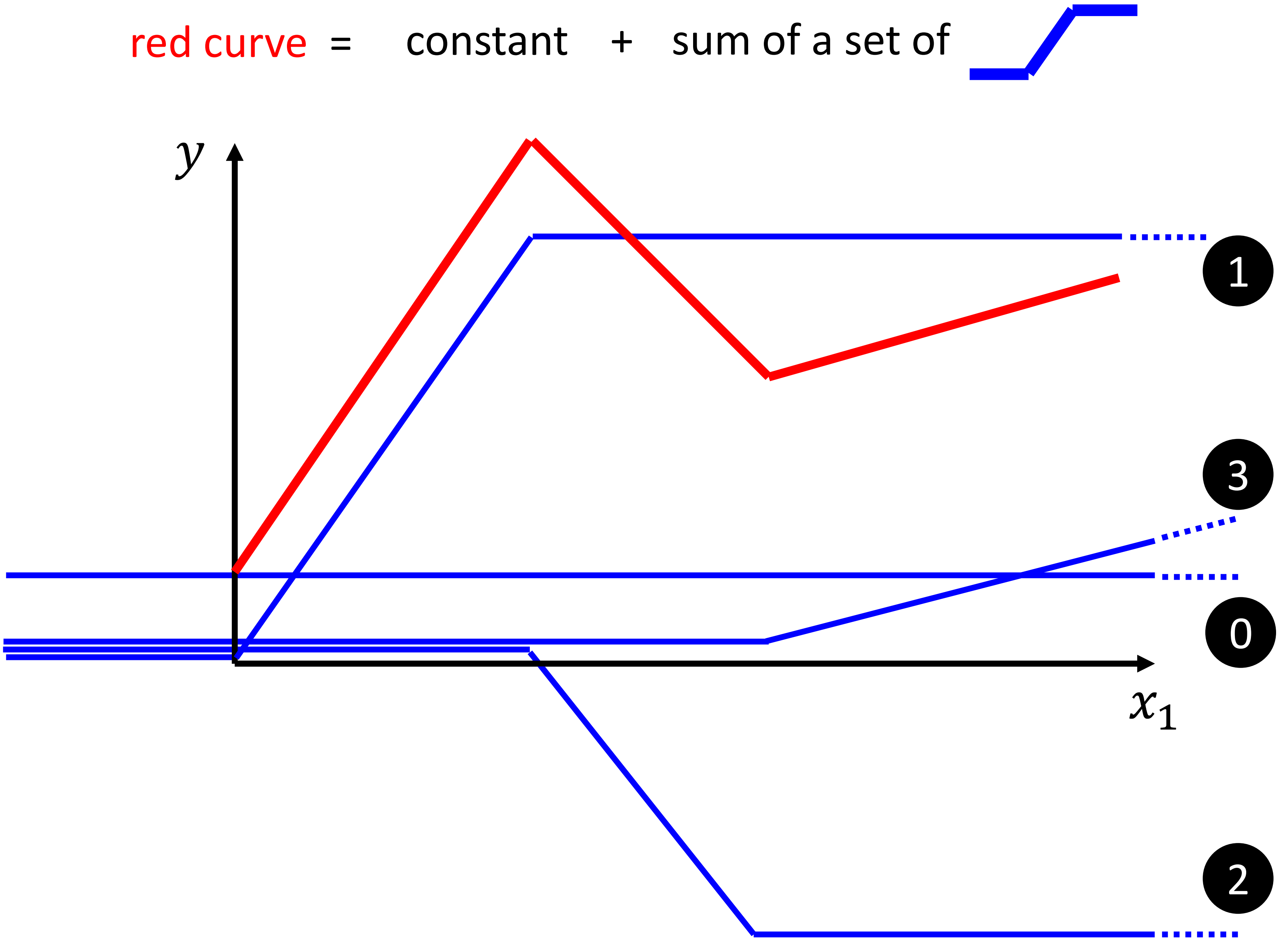
$$y = b + w \cdot x_{cp}$$

Linear models are too simple … we need more sophisticated modes.



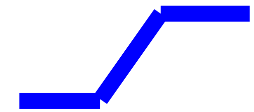
Linear models have severe limitation. ***Model Bias***

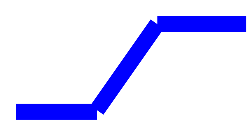
We need a more flexible model!



red curve = constant + sum of a set of
y
x1
1
3
0
2

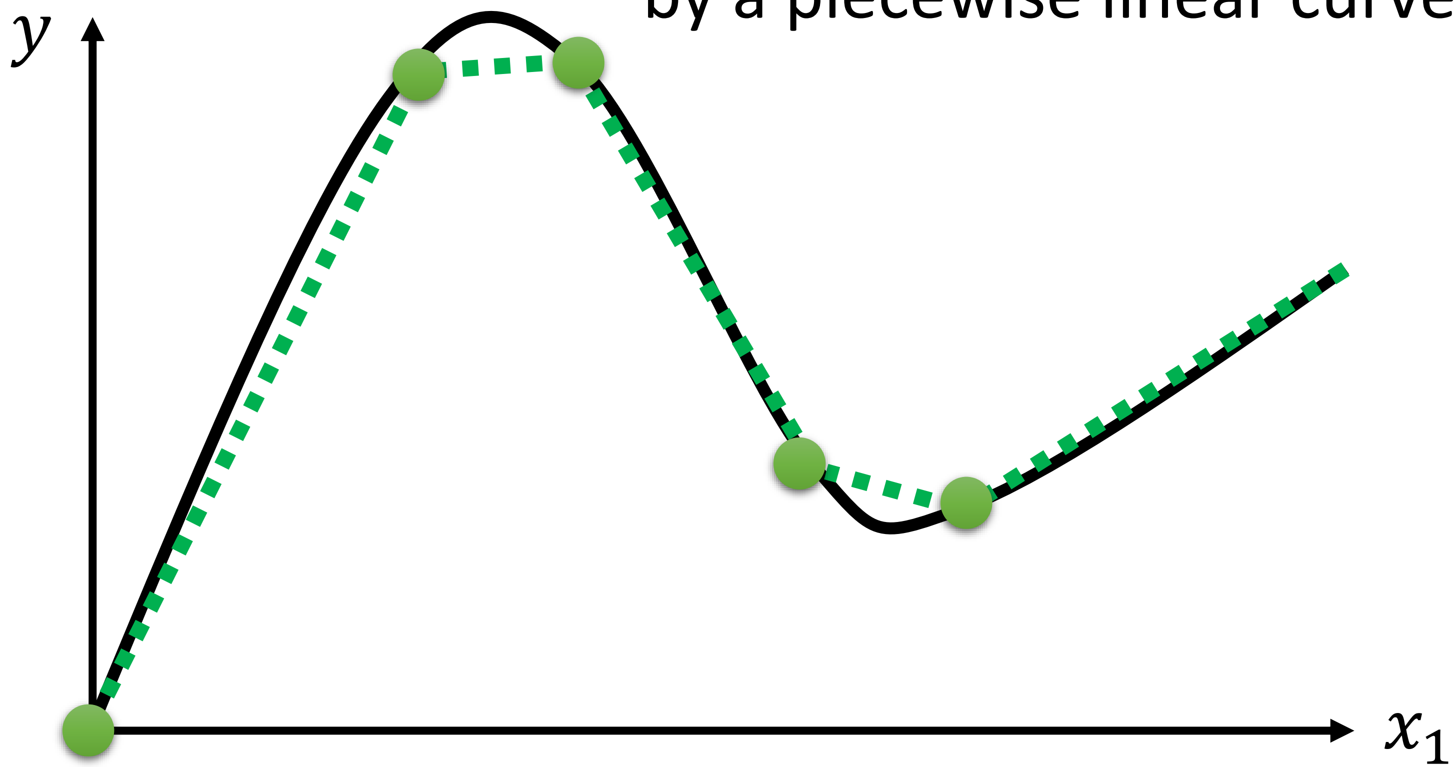
All Piecewise Linear Curves

= constant + sum of a set of 

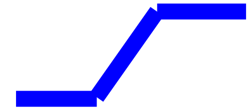
More pieces require more 

Beyond Piecewise Linear?

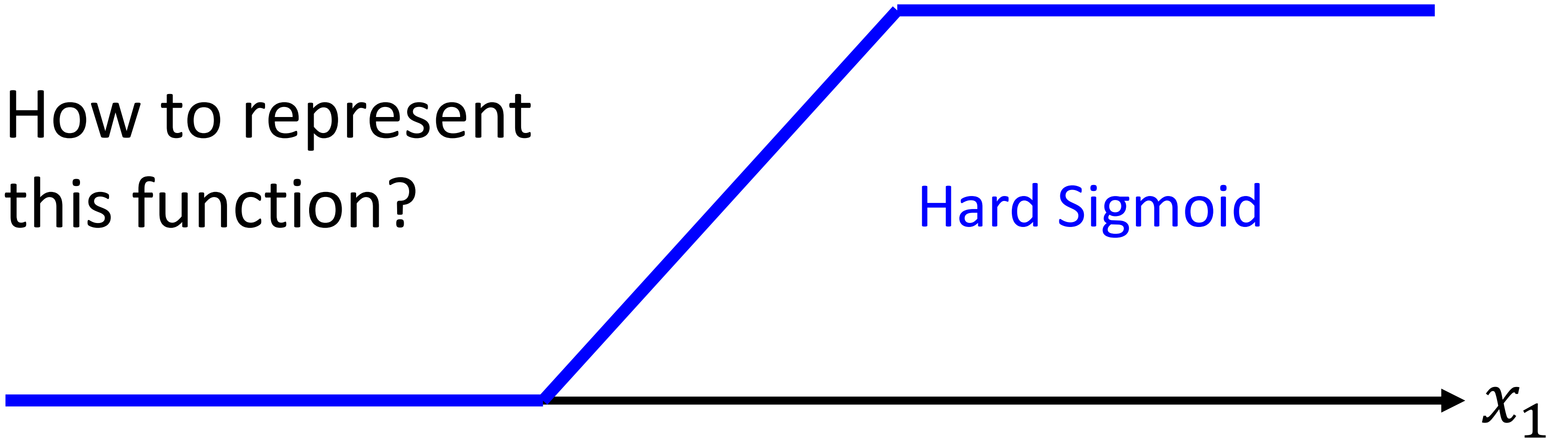
Approximate continuous curve by a piecewise linear curve.



To have good approximation, we need sufficient pieces.

red curve = constant + sum of a set of 

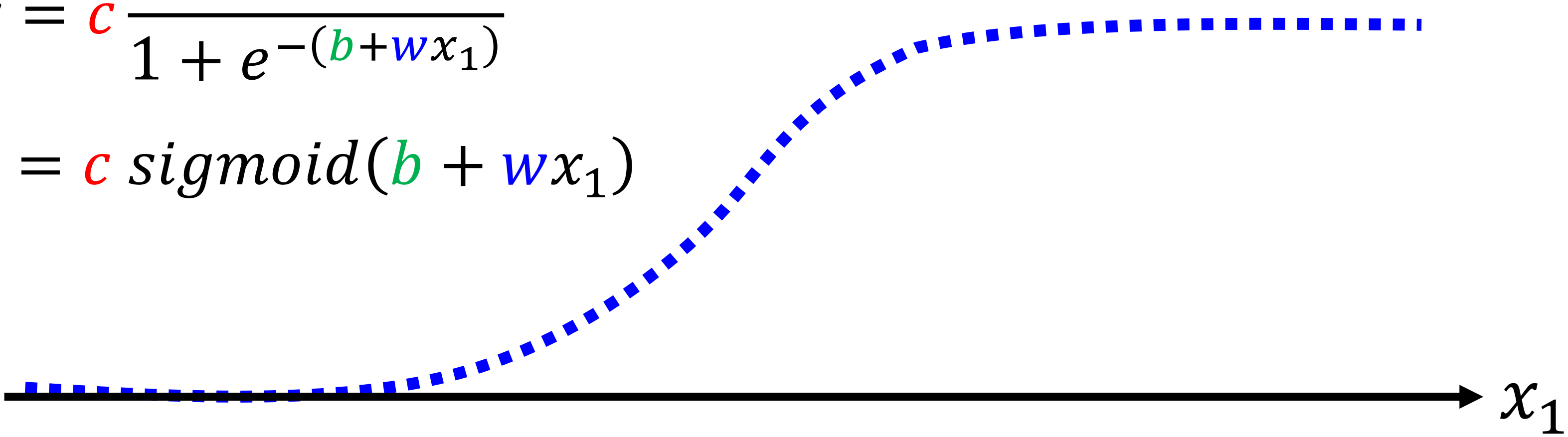
How to represent this function?

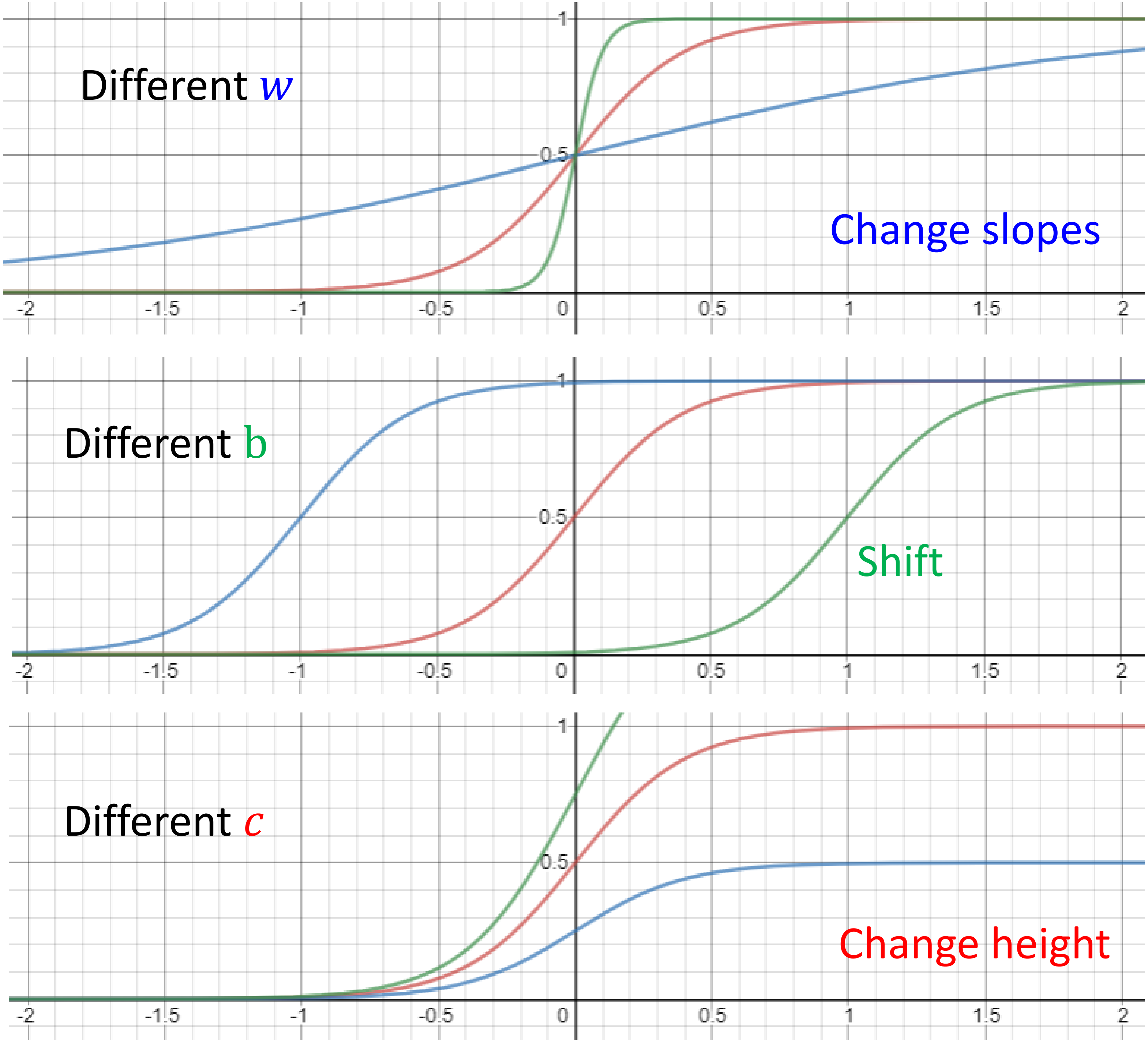


Sigmoid Function

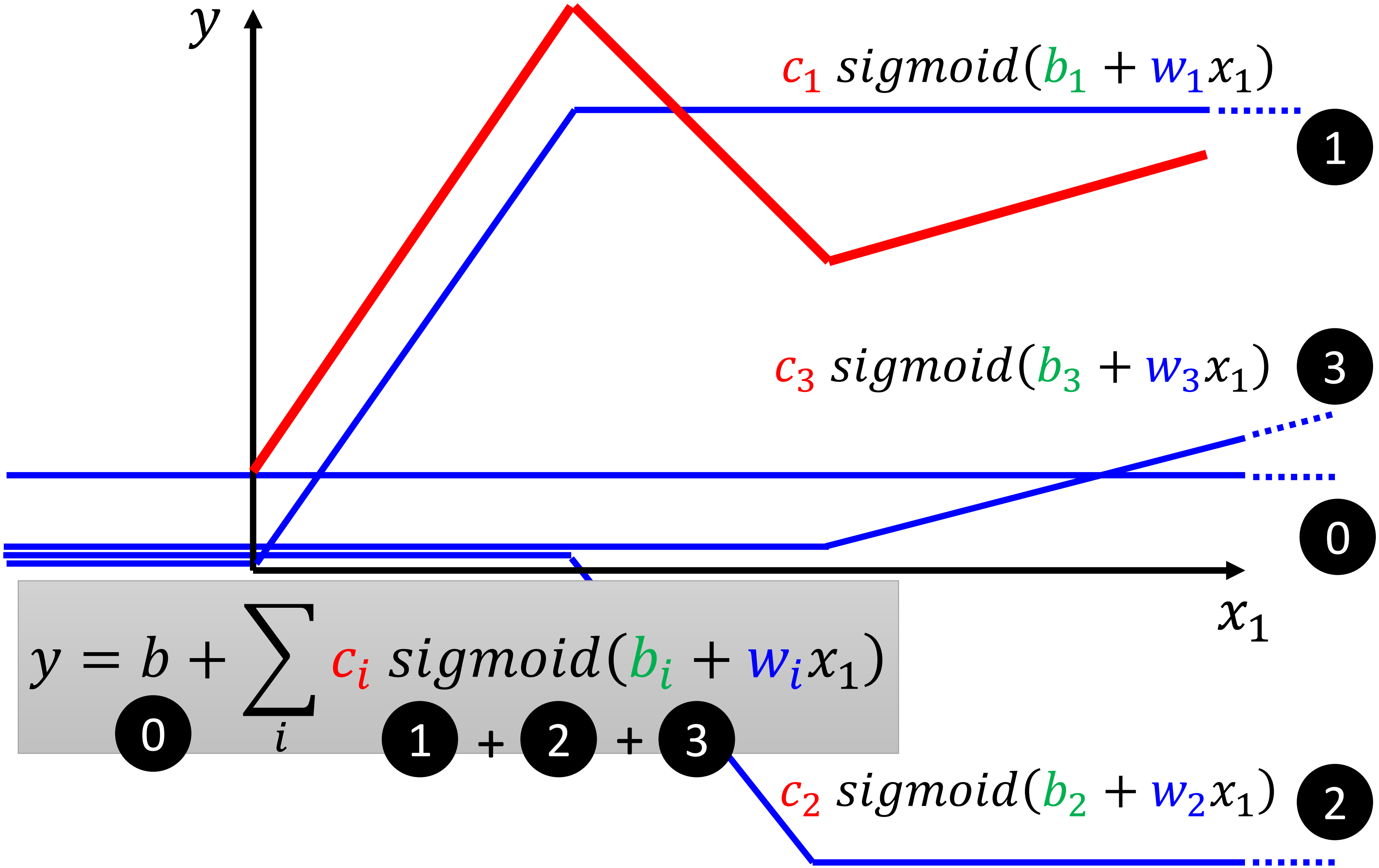
$$y = c \frac{1}{1 + e^{-(b + w x_1)}}$$

$$= c \; sigmoid(b + w x_1)$$




Different w
Change slopes
Different b
Shift
Different c
Change height

red curve = sum of a set of ⎯/⎺ + constant



$$y = b + \sum_i c_i\ sigmoid(b_i + w_i x_1)$$

0 1 + 2 + 3

New Model: More Features

$$y = b + wx_1$$

$$y = b + \sum_i c_i \, sigmoid(b_i + w_i x_1)$$

$$y = b + \sum_j w_j x_j$$

$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right)$$

$$y = b + \sum_i c_i \; sigmoid\left(b_i + \sum_j w_{ij} x_j\right)$$

j: 1,2,3
no. of features

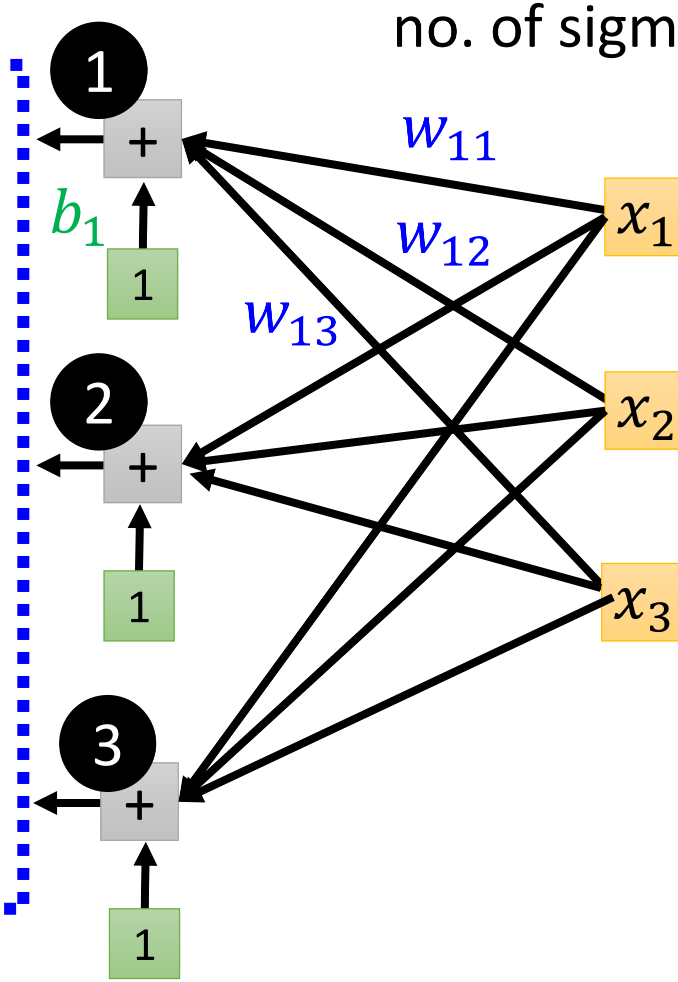
i: 1,2,3
no. of sigmoid

$$r_1 = b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

w_{ij}: weight for x_j for i-th sigmoid

$$r_2 = b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

$$r_3 = b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3$$



$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j \right)$$

i: 1,2,3

j: 1,2,3

$$r_1 = b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

$$r_2 = b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

$$r_3 = b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3$$

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

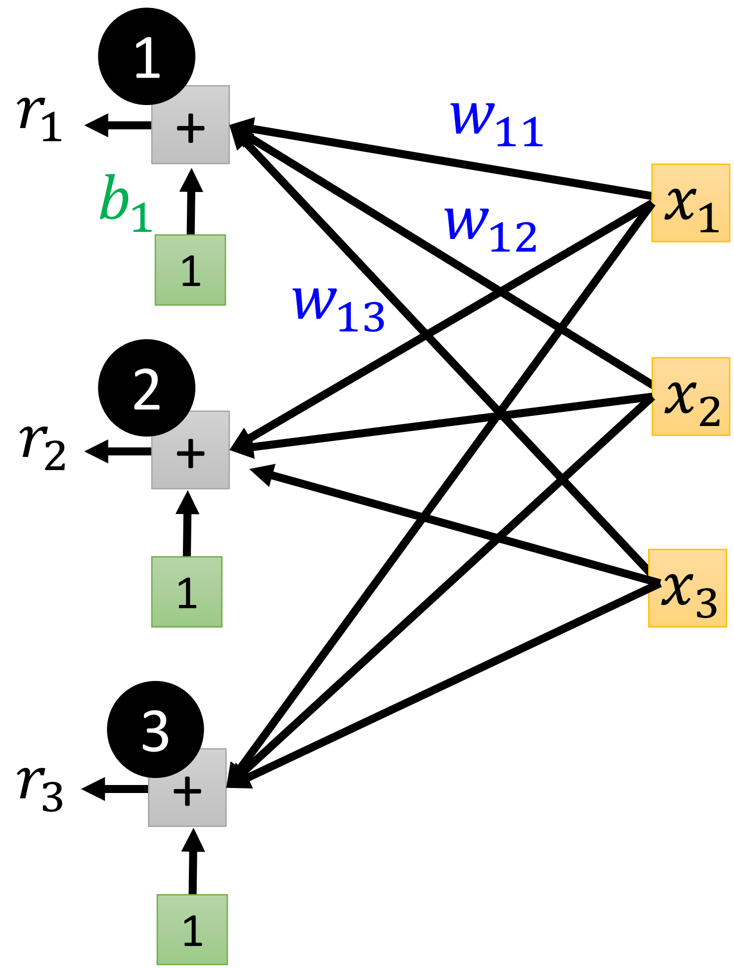
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$

$$y = b + \sum_i c_i \, sigmoid\left(\boxed{b_i + \sum_j w_{ij} x_j}\right)$$

i: 1,2,3

j: 1,2,3

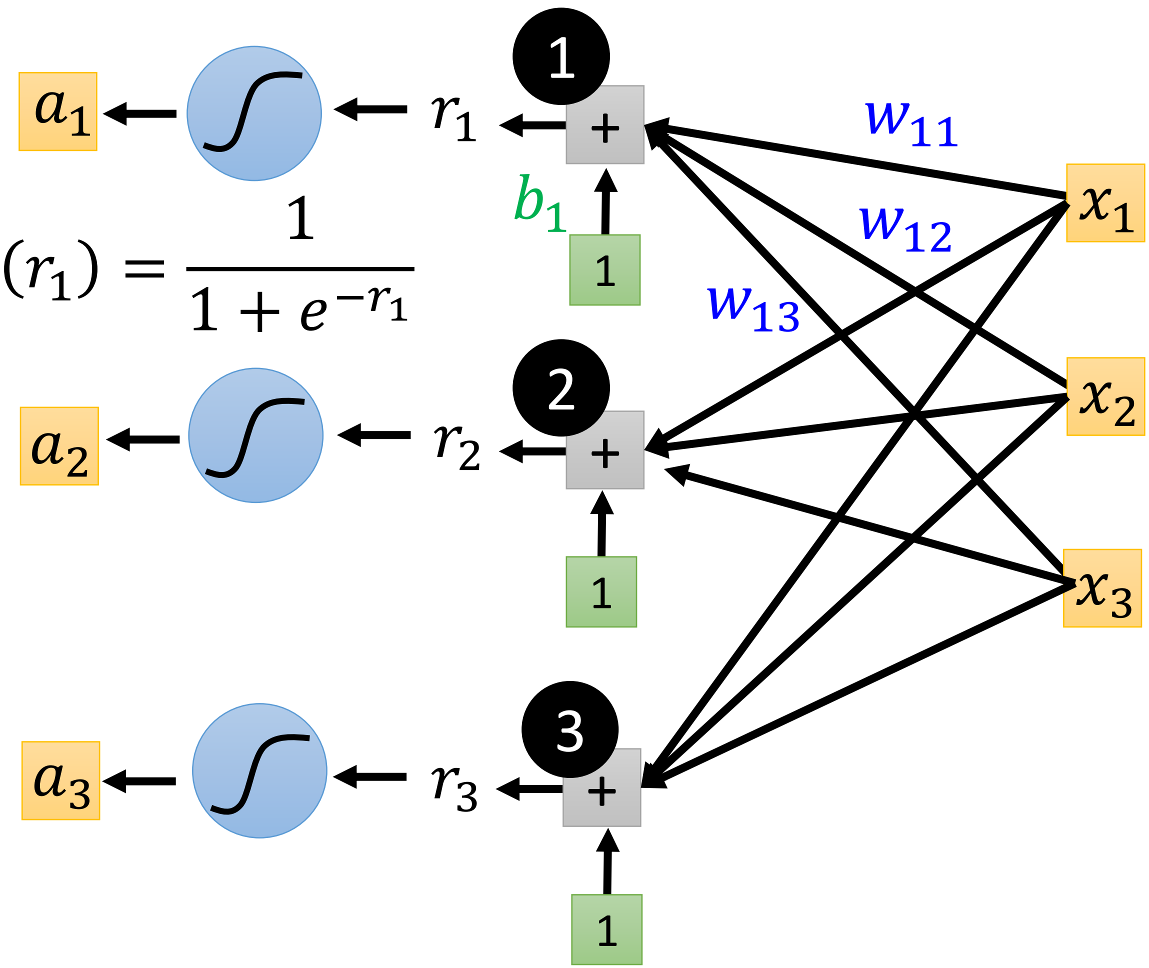
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$



$$y = b + \sum_i c_i \boxed{sigmoid\left(b_i + \sum_j w_{ij} x_j\right)}$$

i: 1,2,3

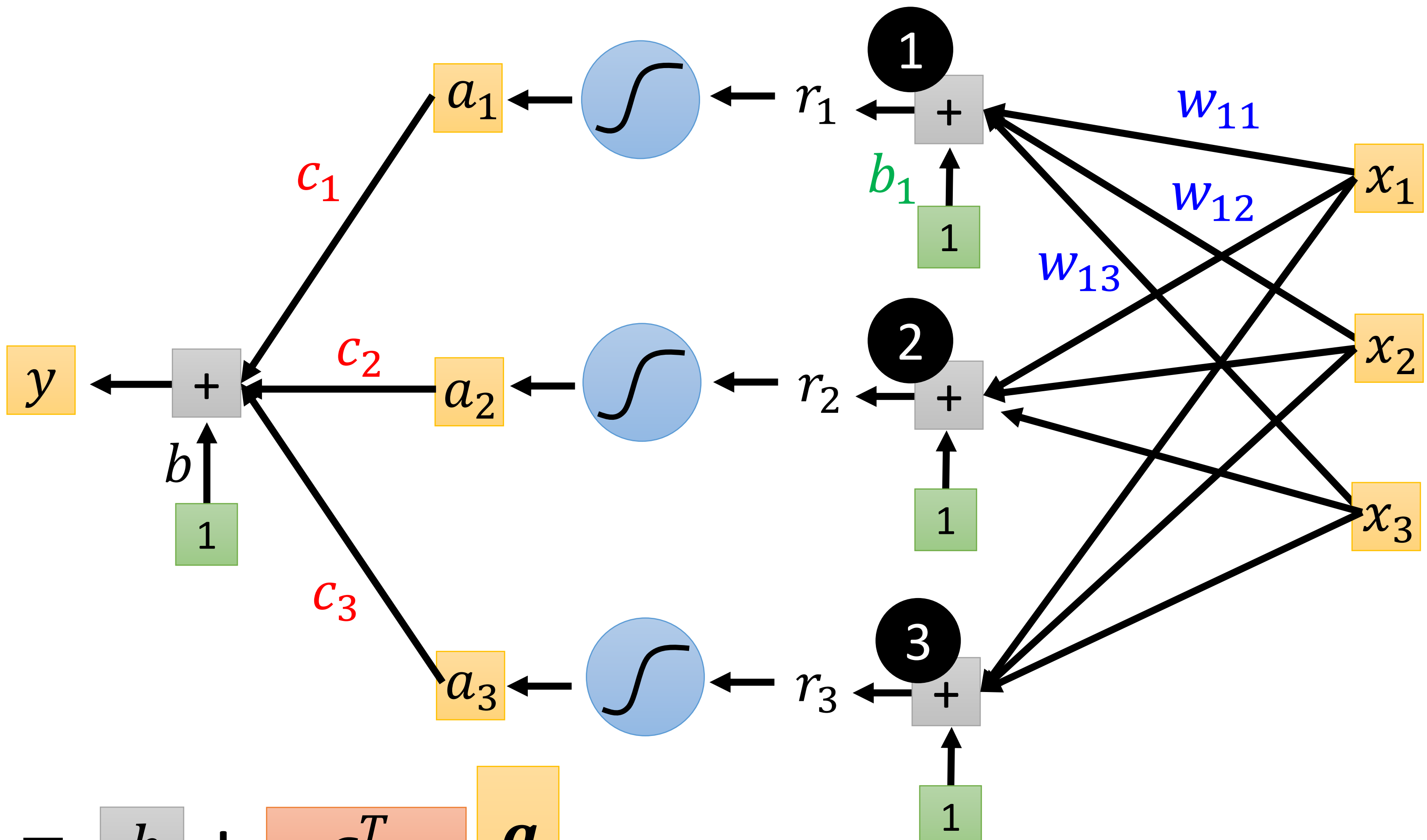
j: 1,2,3



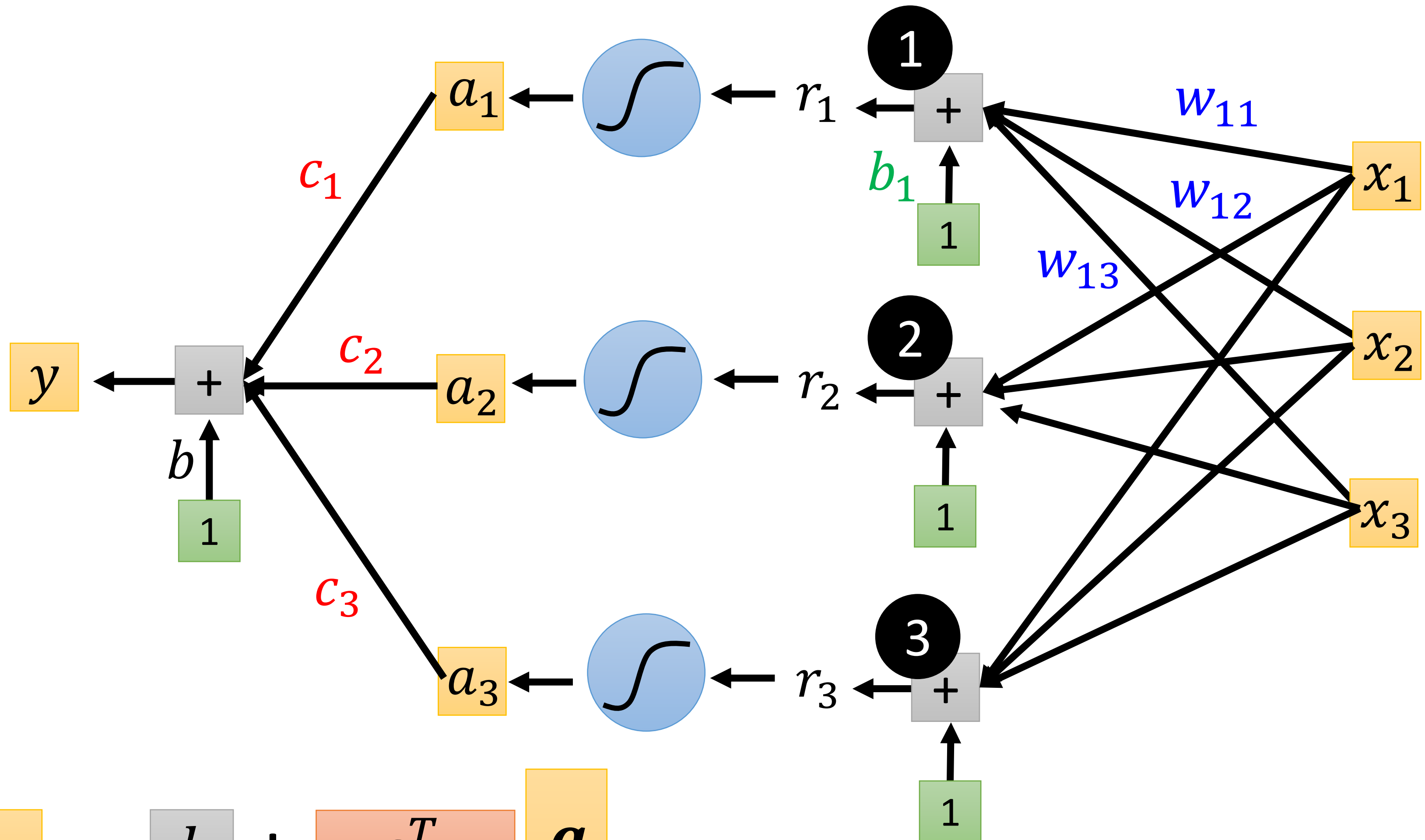
$$a_1 = sigmoid(r_1) = \frac{1}{1 + e^{-r_1}}$$

$$\boldsymbol{a} = \sigma(\ \boldsymbol{r}\ )$$

$$y = \boxed{b + \sum_i c_i \; sigmoid\left(b_i + \sum_j w_{ij} x_j \right)} \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$

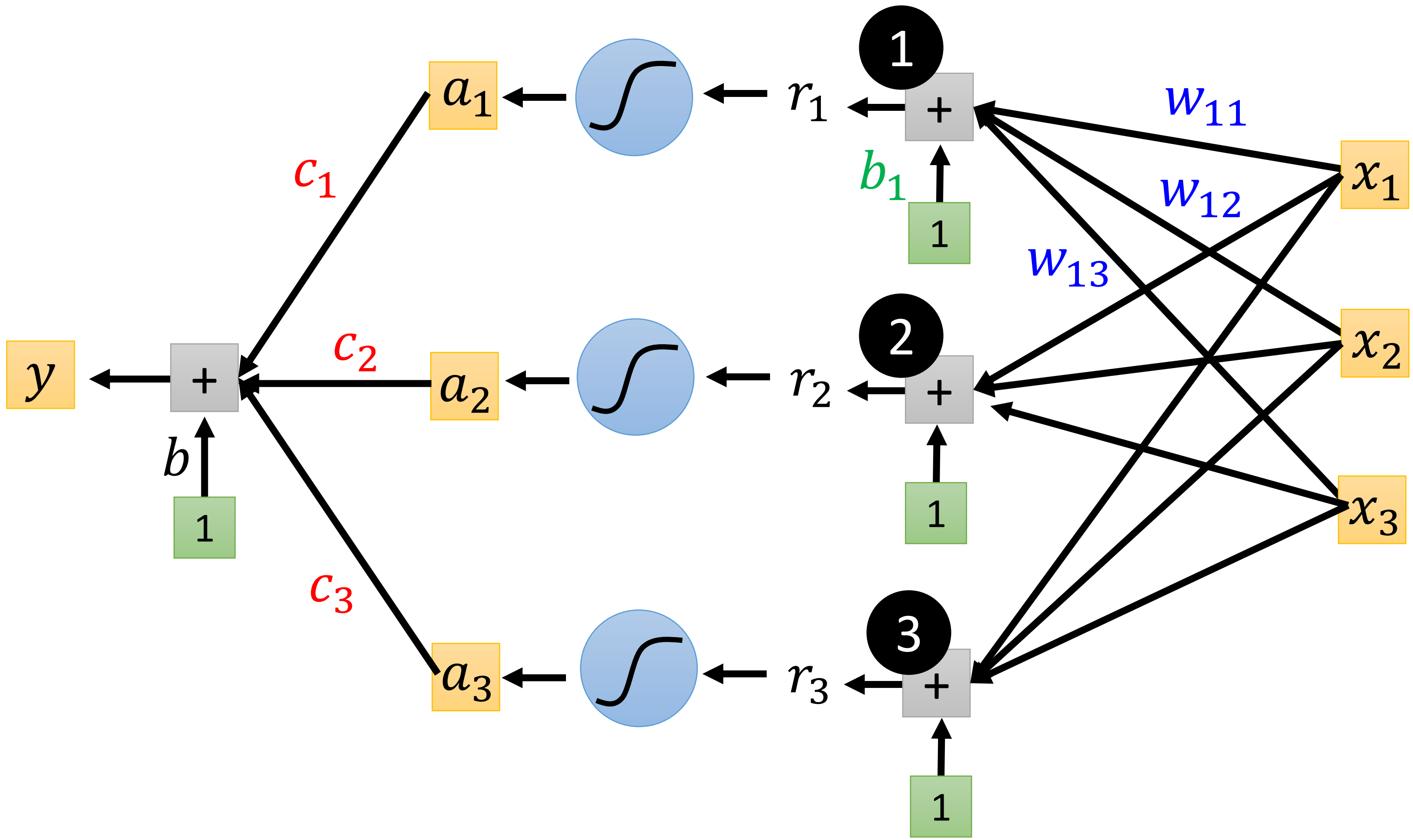


$$y = b + \boldsymbol{c}^T \boldsymbol{a}$$



$$y = b + \boldsymbol{c}^T \boldsymbol{a}$$

$$\boldsymbol{a} = \sigma(\boldsymbol{r}) \qquad \boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$



$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W \boldsymbol{x})$$

Function with unknown parameters

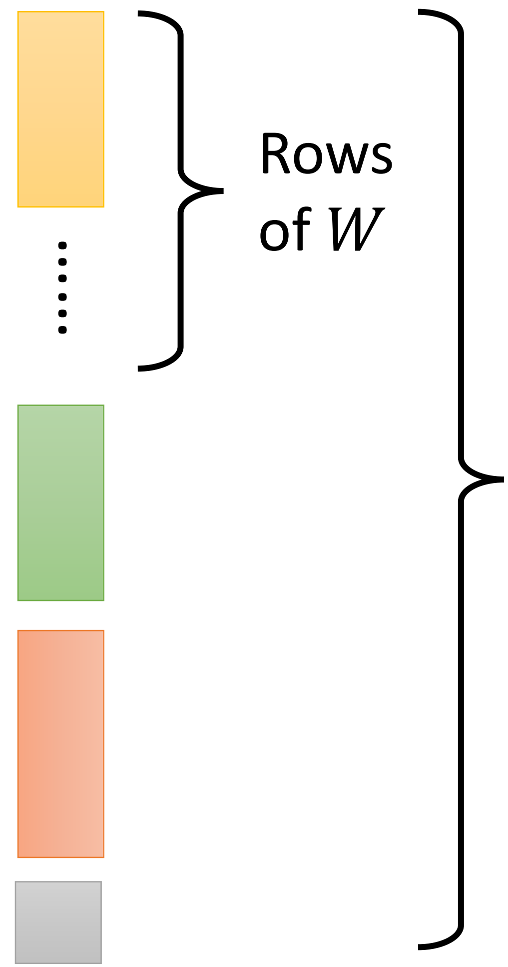
$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W\boldsymbol{x})$$

$\boldsymbol{x}$ **feature**

Unknown parameters

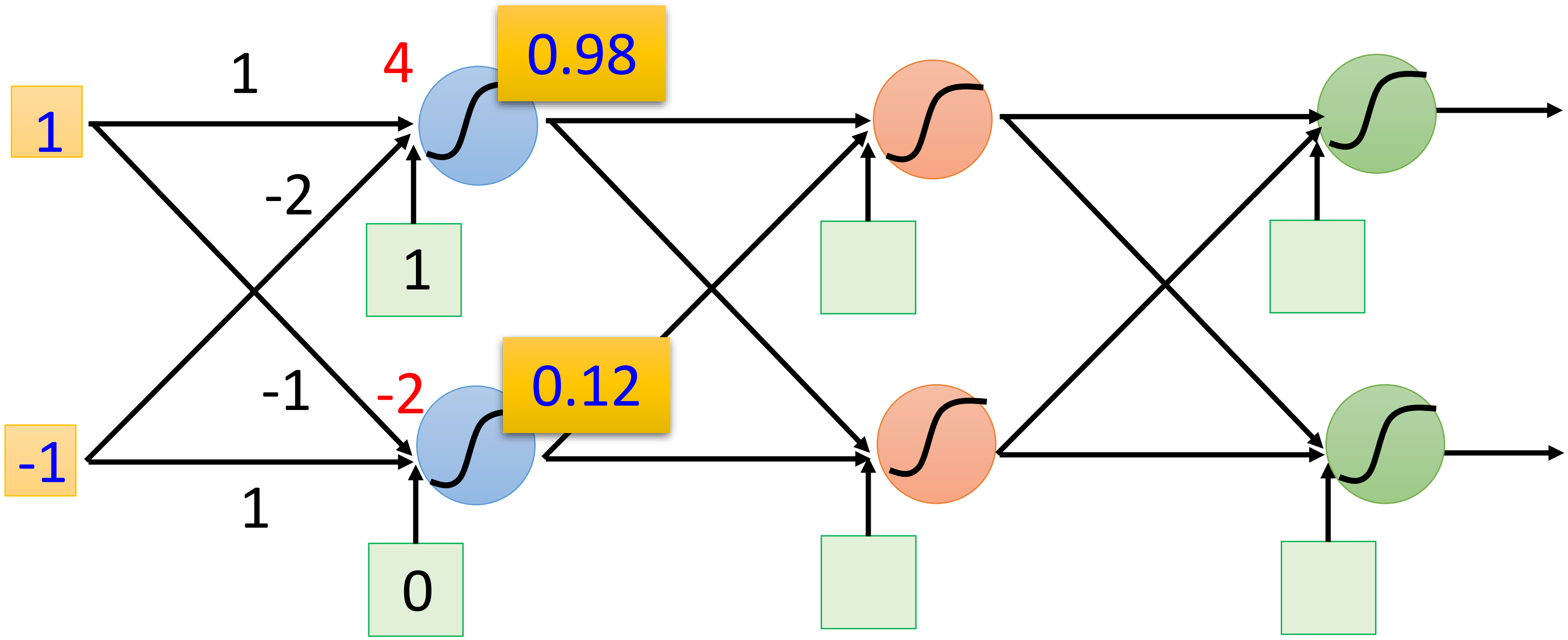
W $\boldsymbol{b}$

$\boldsymbol{c}^T$ b

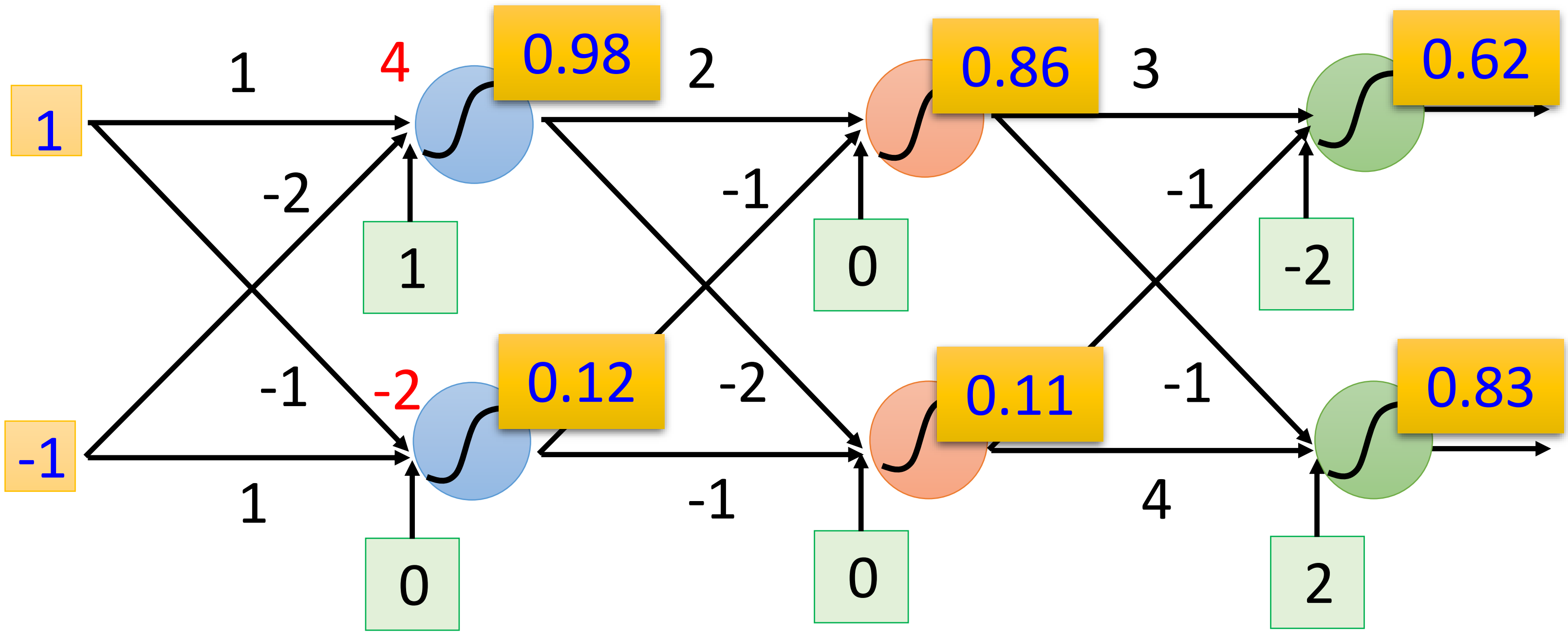


$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \end{bmatrix}$$

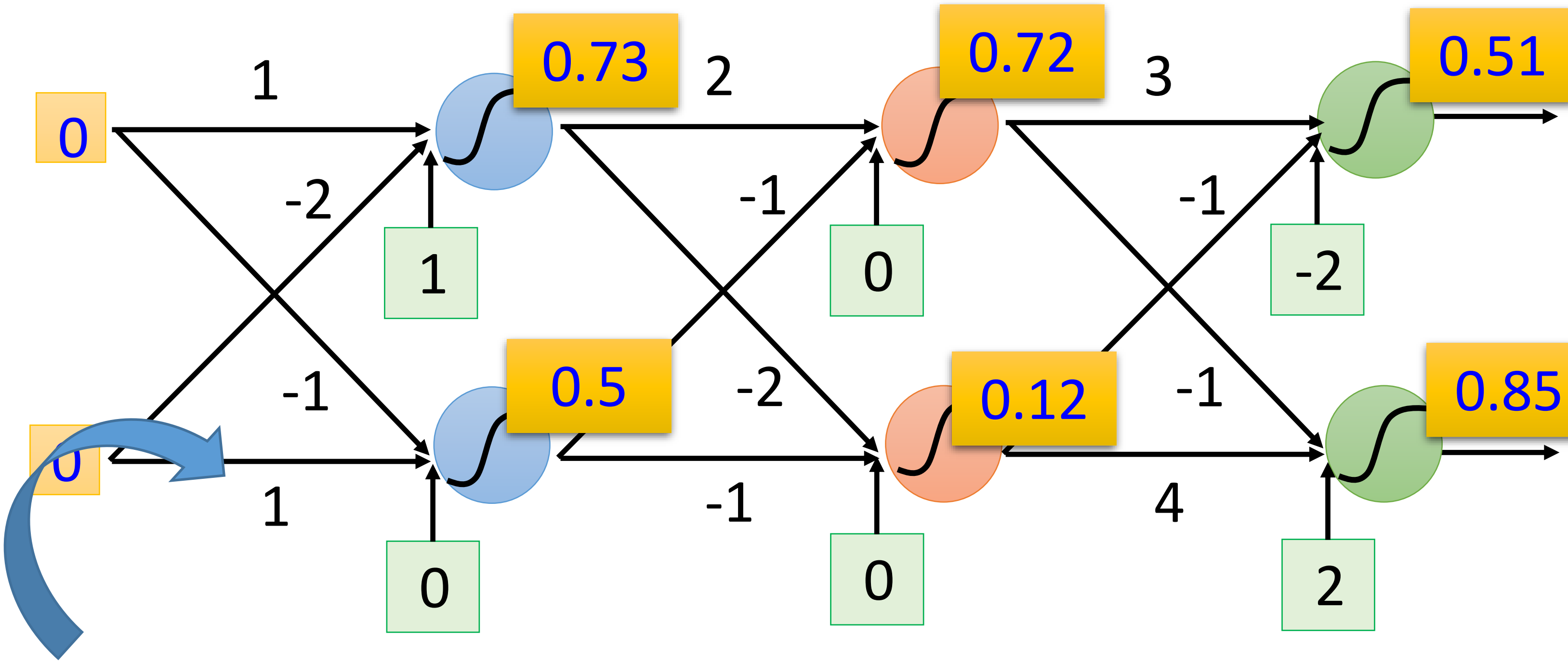
Fully Connect Feedforward Network



Fully Connect Feedforward Network



Fully Connect Feedforward Network



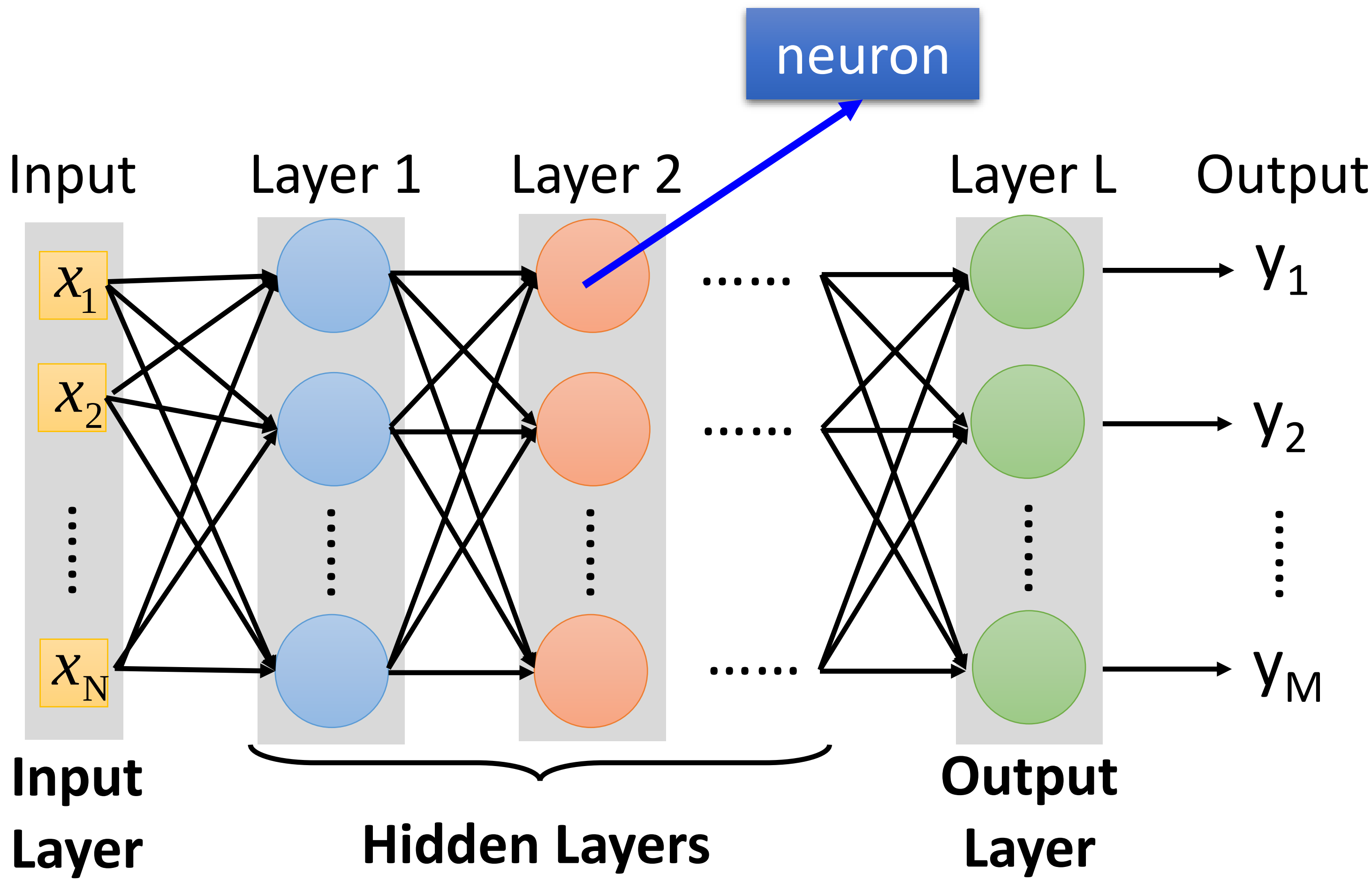
This is a function.

Input vector, output vector

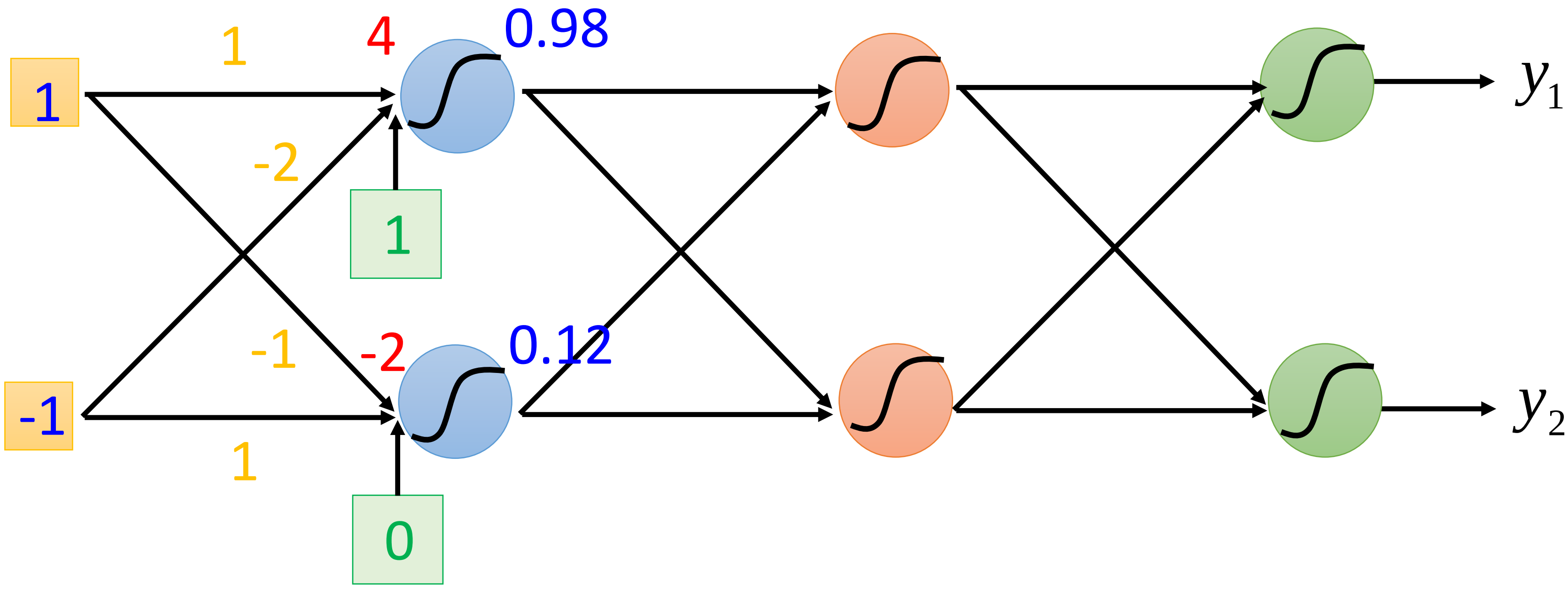
$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

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

Fully Connect Feedforward Network

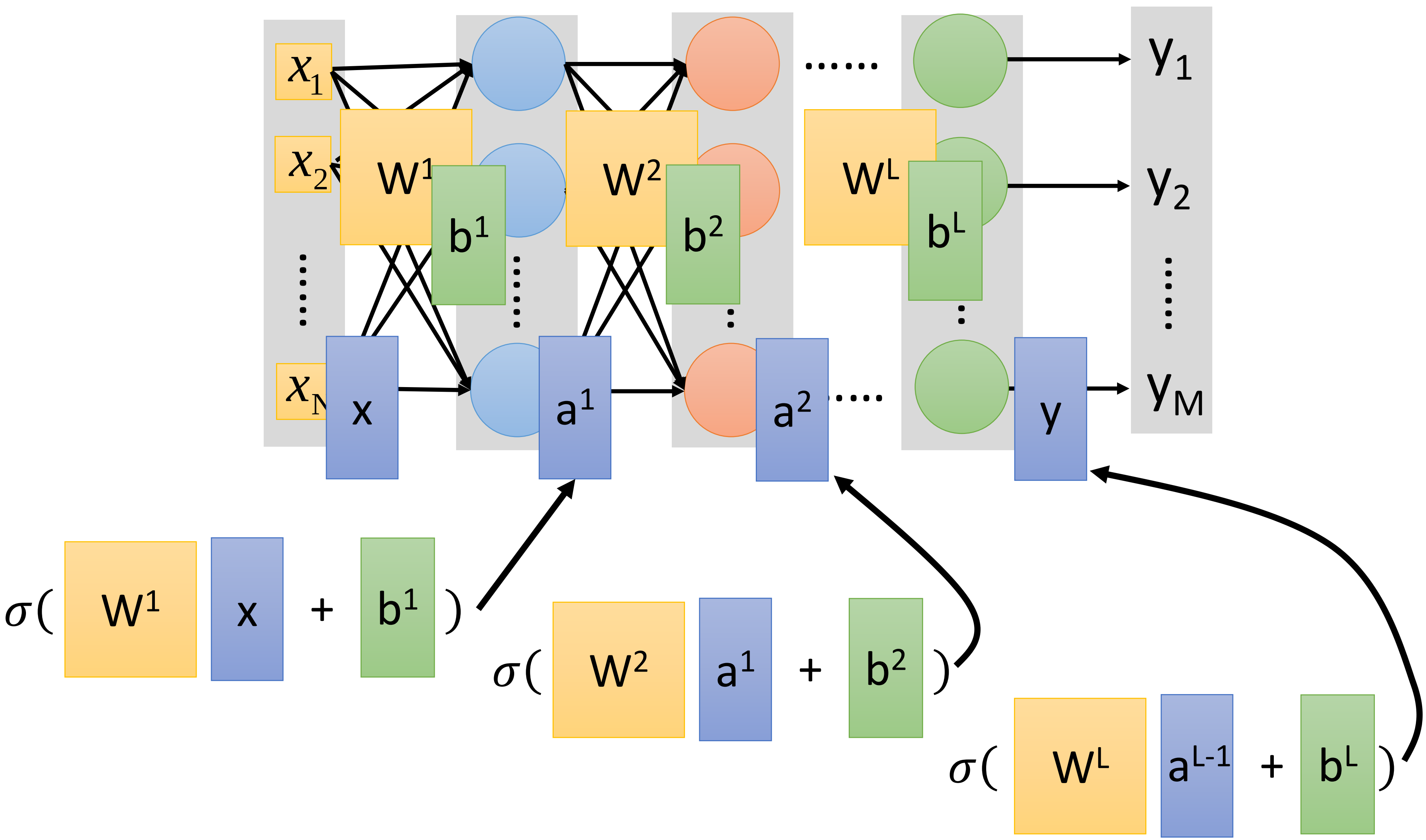


Matrix Operation

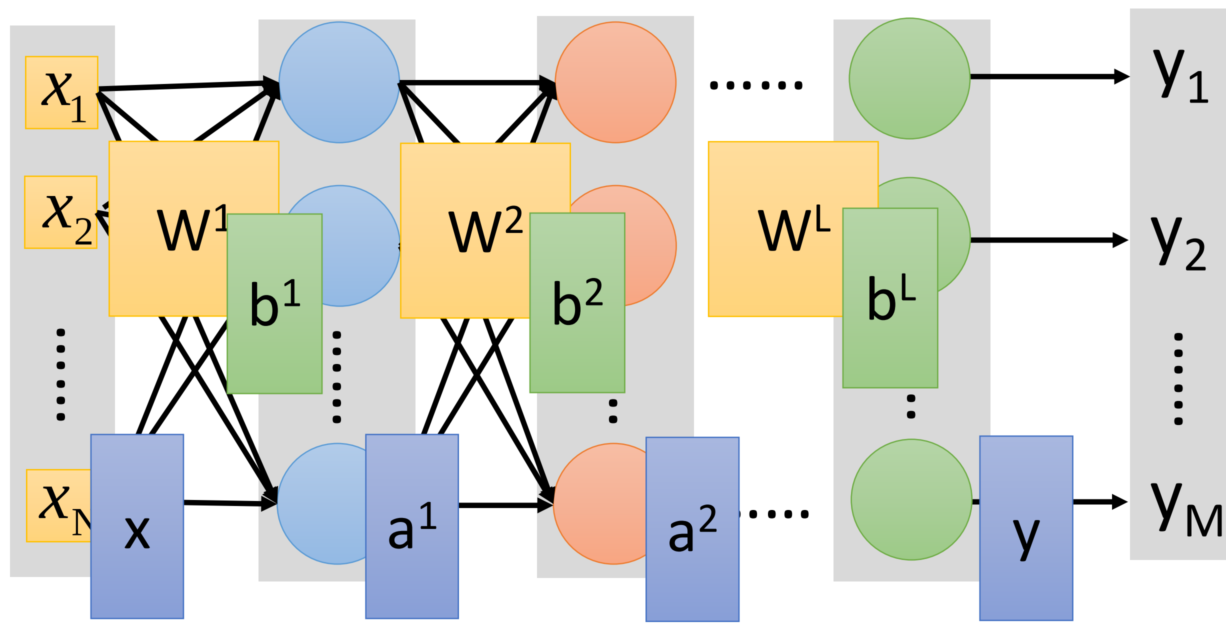


$$\sigma\Big(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\Big) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network

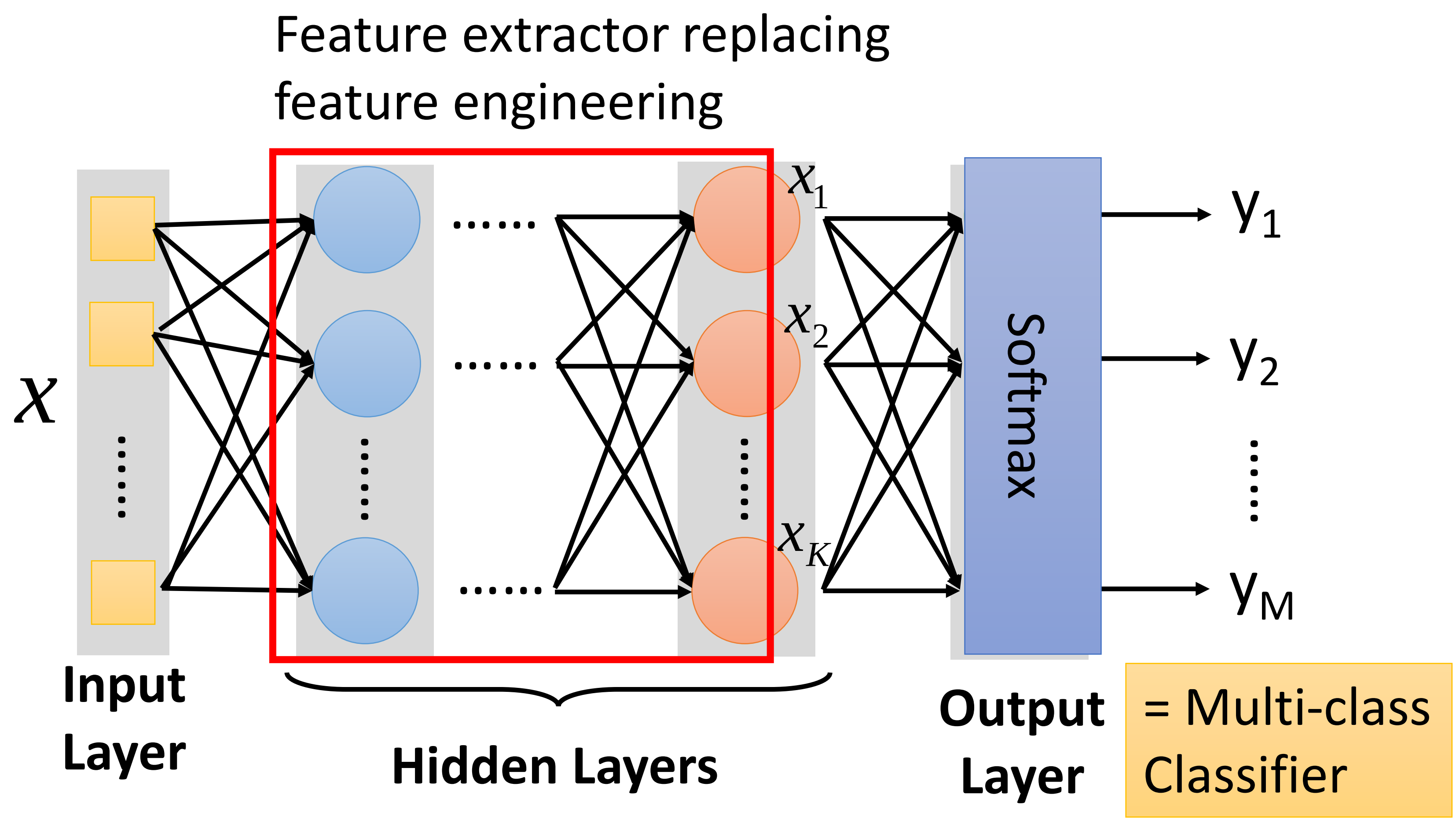


$$y = f(x)$$

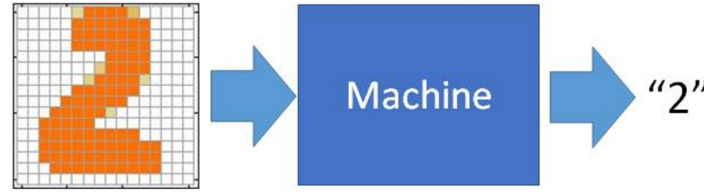
Using parallel computing techniques to speed up matrix operation

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

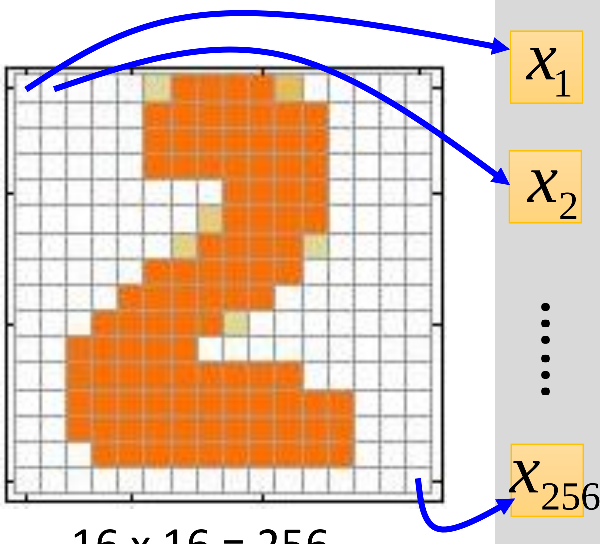
Output Layer as Multi-Class Classifier



Example Application



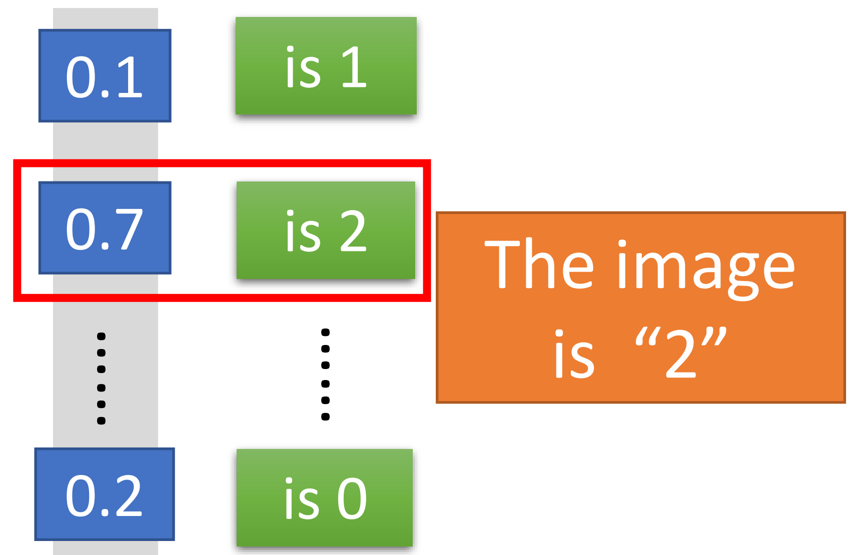
Input



16 x 16 = 256

Ink → 1
No ink → 0

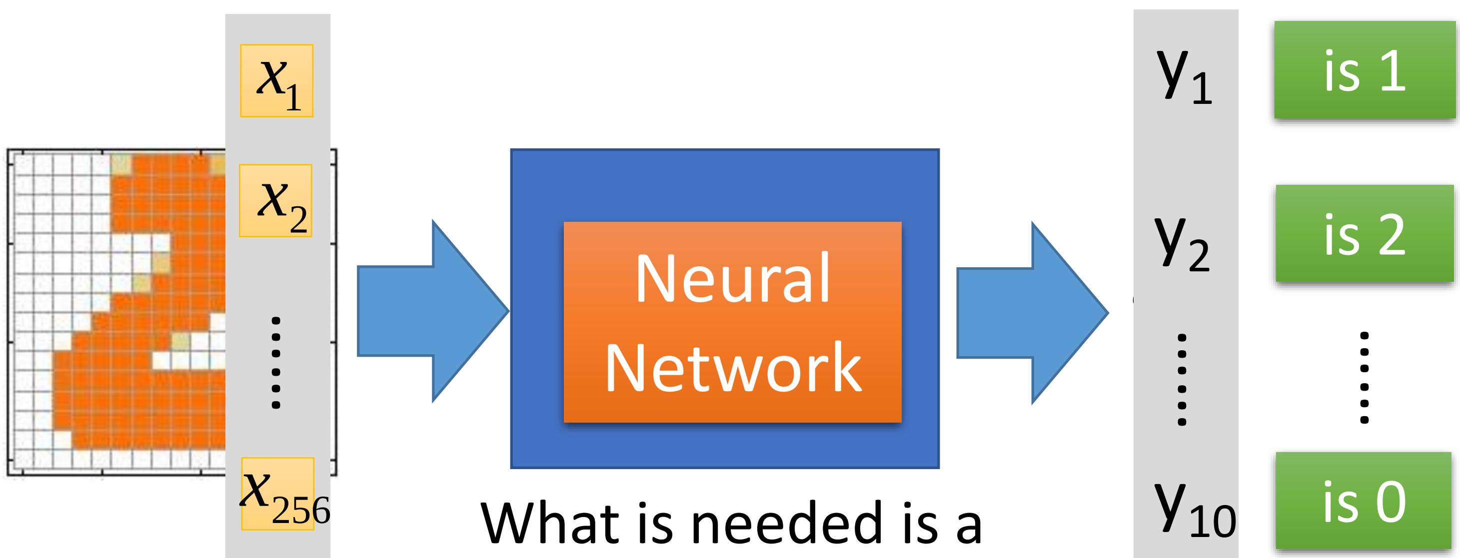
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition

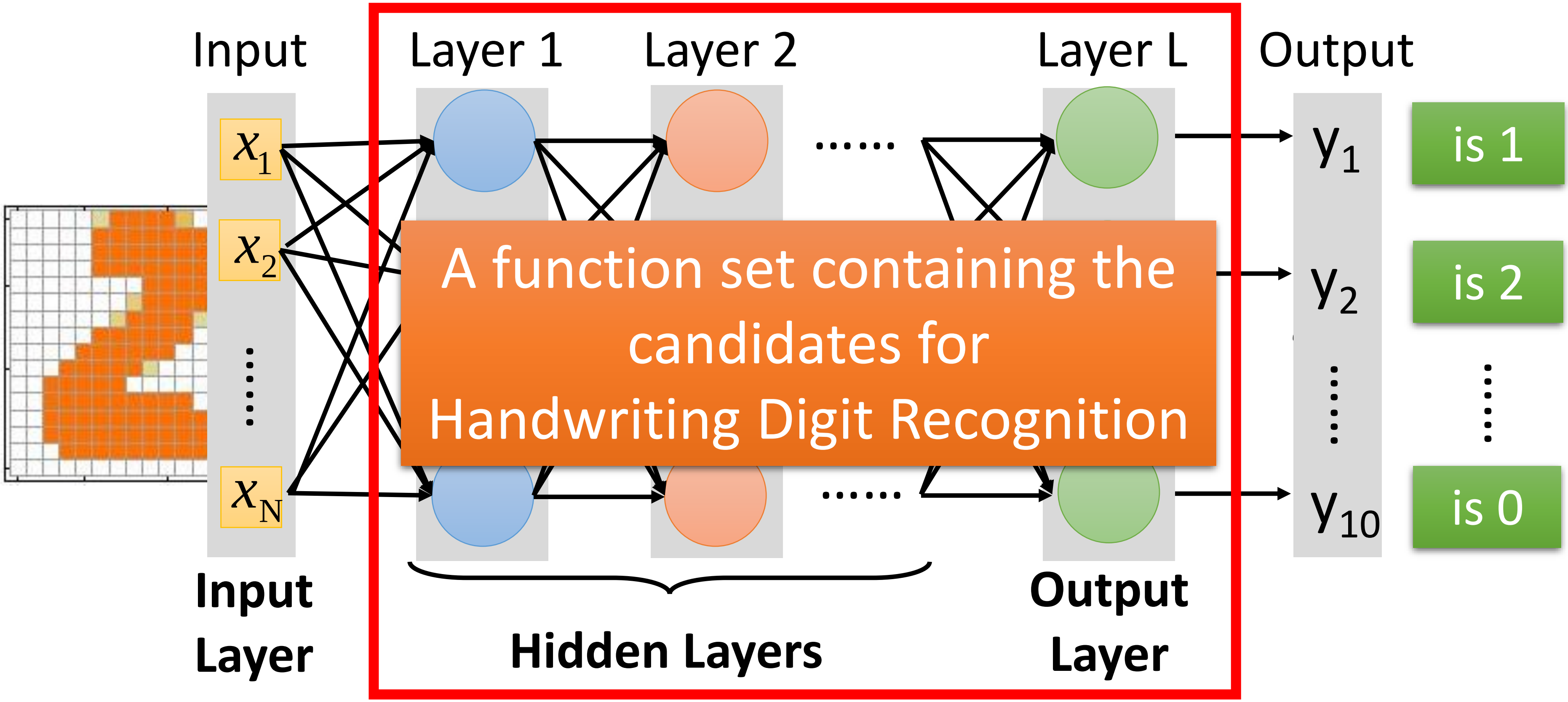


What is needed is a function ……

Input:
256-dim vector

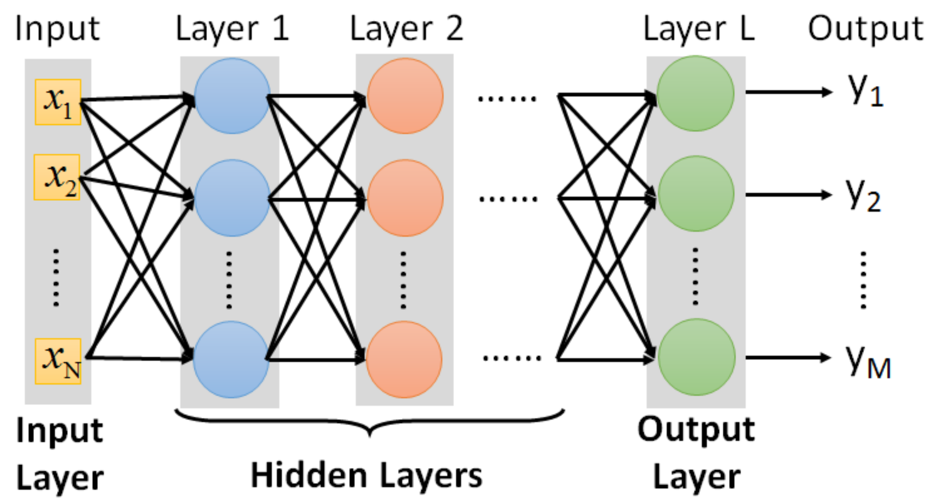
output:
10-dim vector

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?

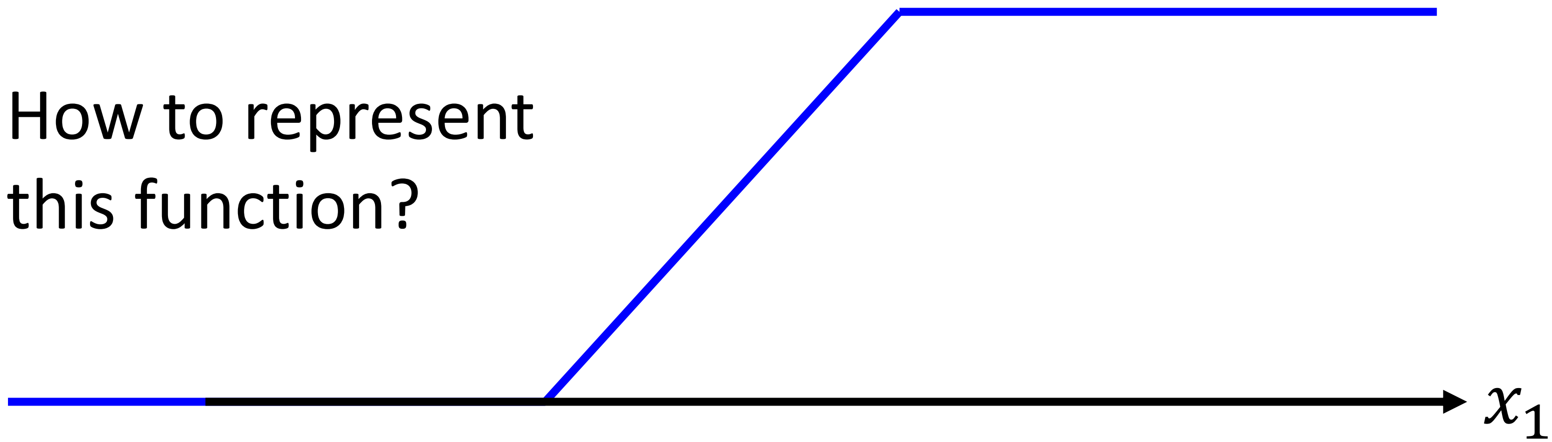
Trial and Error + Intuition

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

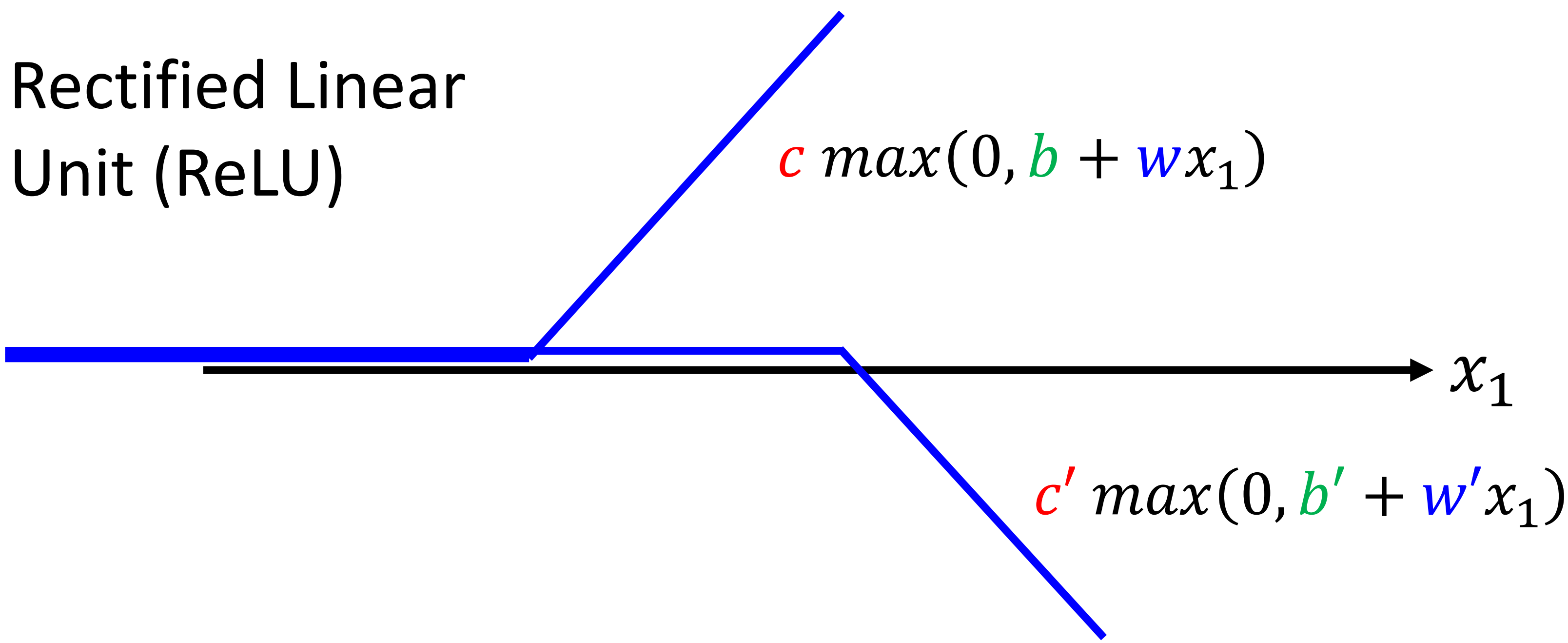
Convolutional Neural Network (CNN)

Sigmoid → ReLU

How to represent this function?



Rectified Linear Unit (ReLU)



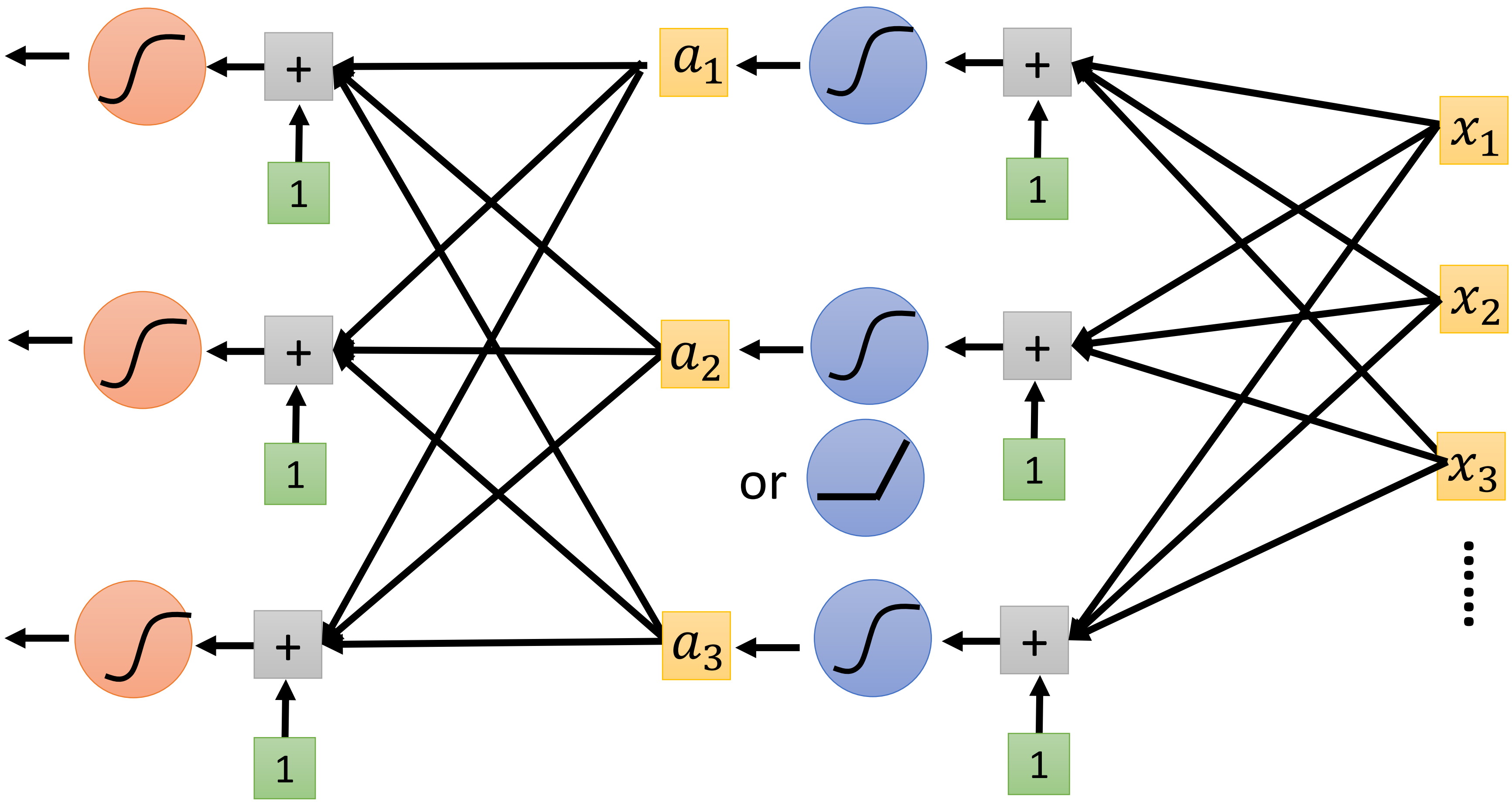
Sigmoid → ReLU

$$y = b + \sum_{\boxed{i}} c_i \underline{sigmoid}\left(b_i + \sum_j w_{ij} x_j\right)$$

Activation function

$$y = b + \sum_{\boxed{2i}} c_i \underline{max}\left(0, b_i + \sum_j w_{ij} x_j\right)$$

Which one is better?



$$\boldsymbol{a}' = \sigma(\boldsymbol{b}' + W' \boldsymbol{a}) \qquad \boldsymbol{a} = \sigma(\boldsymbol{b} + W \boldsymbol{x})$$

Three Steps for Deep Learning

Step 1: function with unknown → **Step 2: define loss from training data** → Step 3: optimization

$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W \boldsymbol{x})$$

Loss

- Loss is a function of parameters $L(\theta)$
- Loss means how good a set of values is.

feature

$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W \boldsymbol{x})$$

e

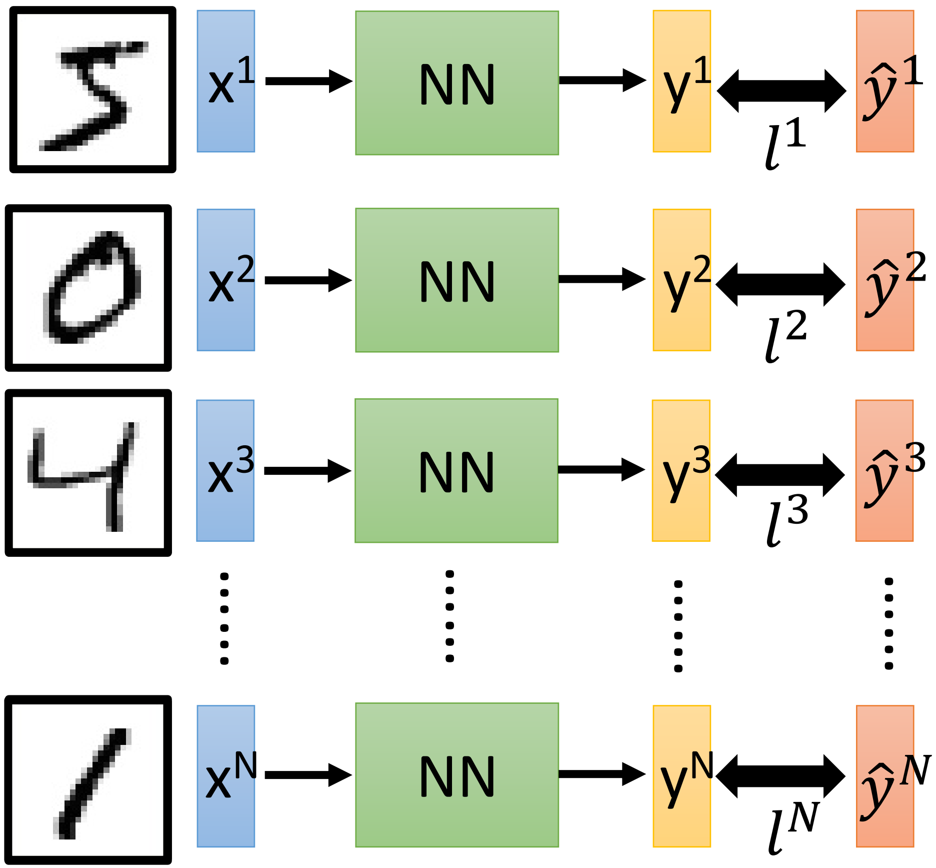
label $\hat{y}$

Given a set of values

Loss: $$L = \frac{1}{N}\sum_n e_n$$

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: function with unknown → Step 2: define loss from training data → Step 3: optimization

$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W\boldsymbol{x})$$

Optimization of New Model

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \end{bmatrix}$$

$$\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$

$$\boldsymbol{g} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \frac{\partial L}{\partial \theta_2}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \vdots \end{bmatrix}$$

gradient

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \vdots \end{bmatrix} \leftarrow \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \\ \vdots \end{bmatrix} - \begin{bmatrix} \eta \frac{\partial L}{\partial \theta_1}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \eta \frac{\partial L}{\partial \theta_2}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \vdots \end{bmatrix}$$

$$\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^0)$$

$$\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$$

Optimization of New Model

$$\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^0)$

$$\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$$

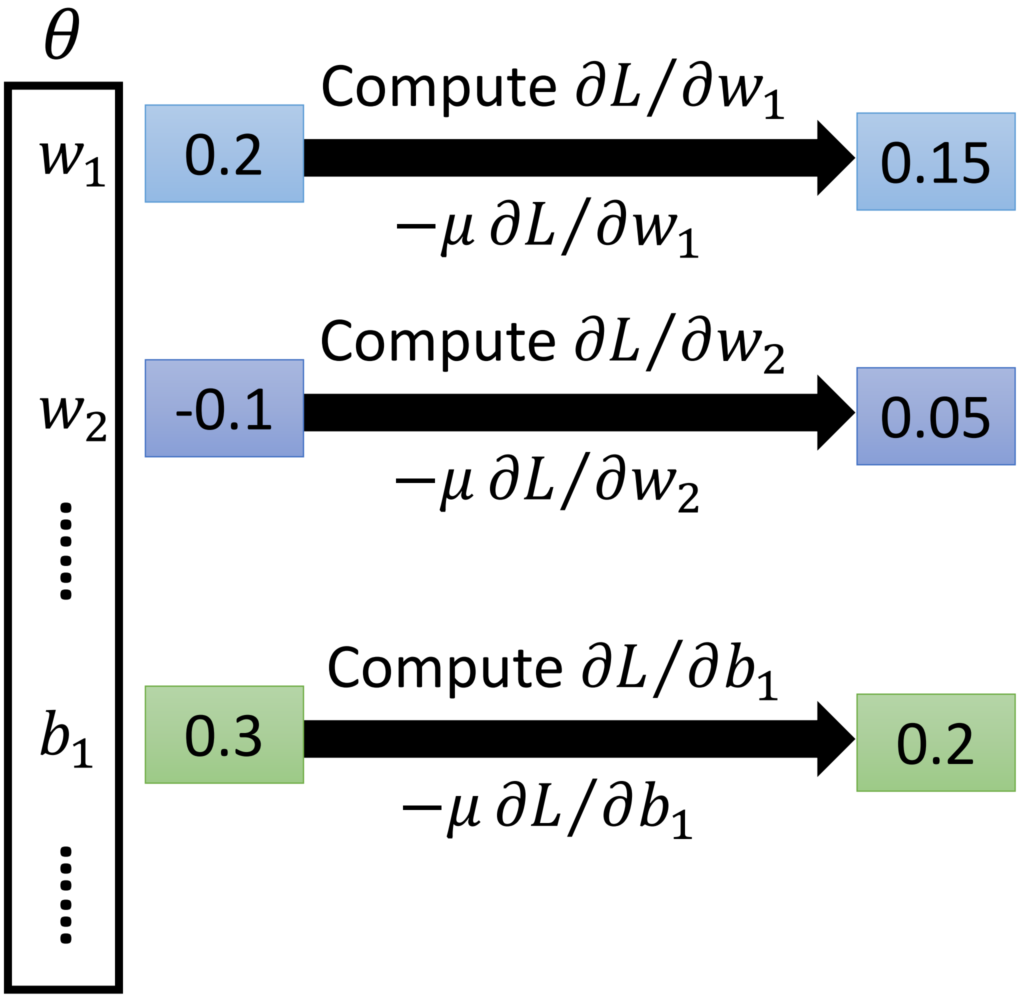
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^1)$

$$\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$$

- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^2)$

$$\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}$$

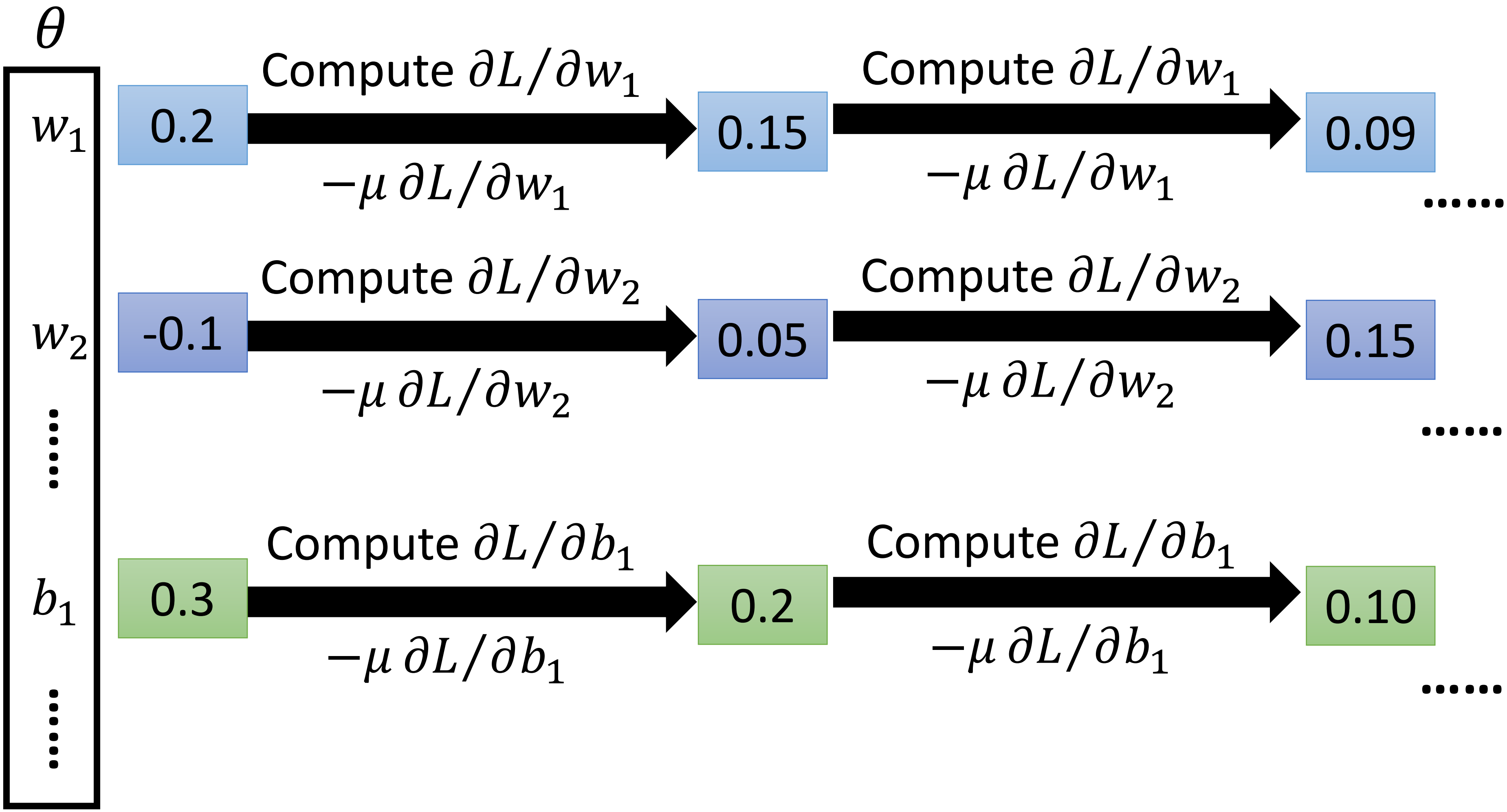
Gradient Descent



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

gradient

Gradient Descent



Backpropagation

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

TensorFlow

PYTORCH

MATLAB

torch

theano

Caffe

Microsoft CNTK

Chainer

Deep Learning library produced by Amazon
DSSTNE

mxnet

libdnn

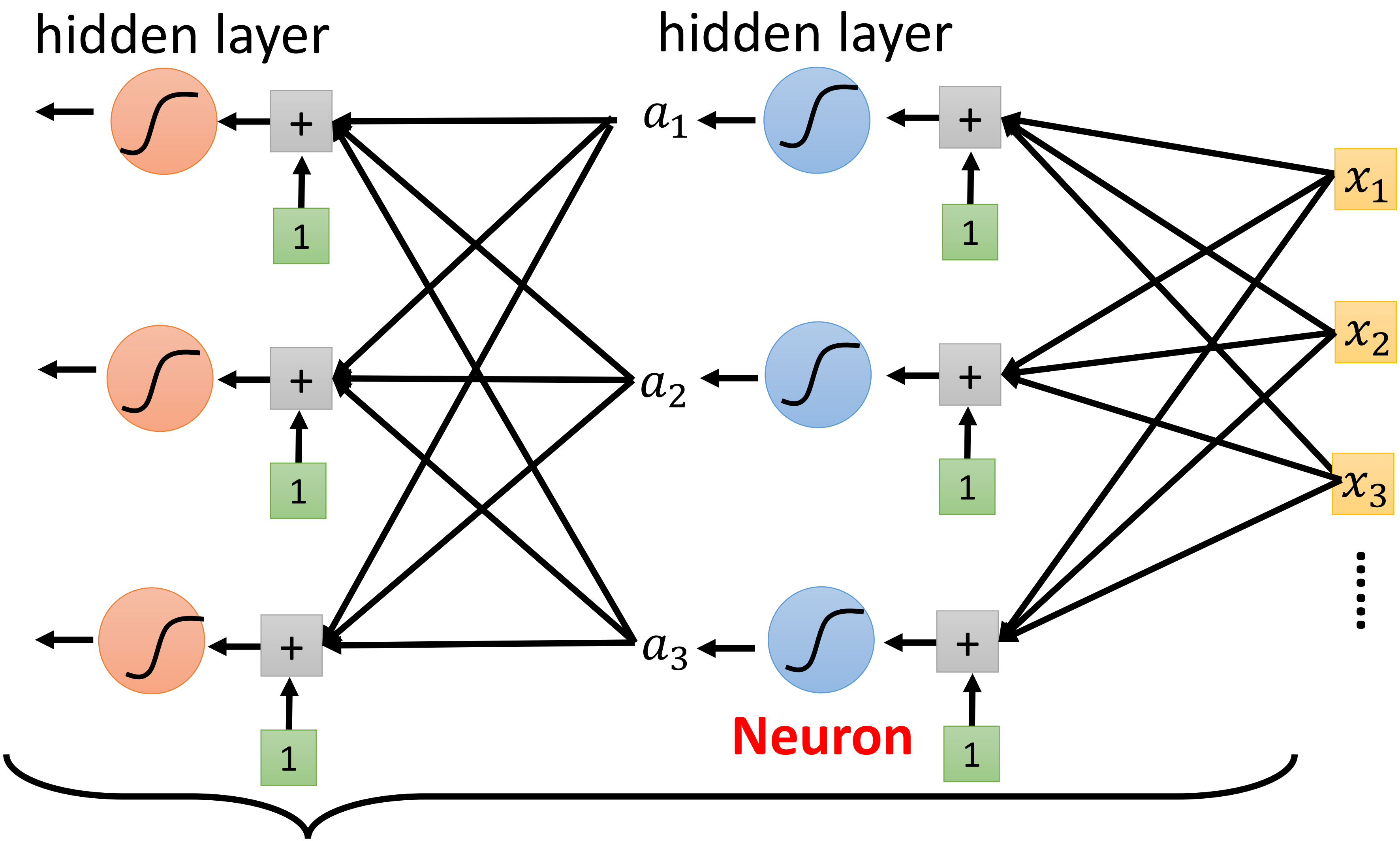
Three Steps for Deep Learning

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

Deep Learning is so simple ……

It is not ***fancy*** enough.

Let's give it a ***fancy*** name!



Neural Network   This mimics human brains … (???)

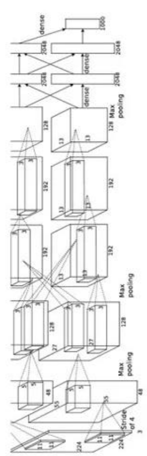
Many layers means **Deep** ➡ Deep Learning

Deep = Many hidden layers

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

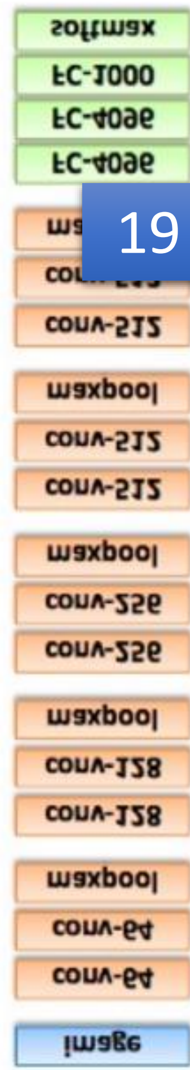
8 layers

16.4%



AlexNet (2012)

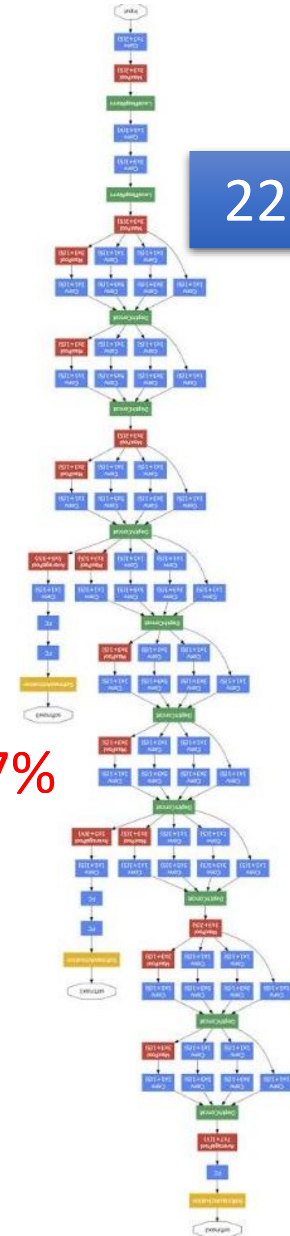
7.3%



19 layers

VGG (2014)

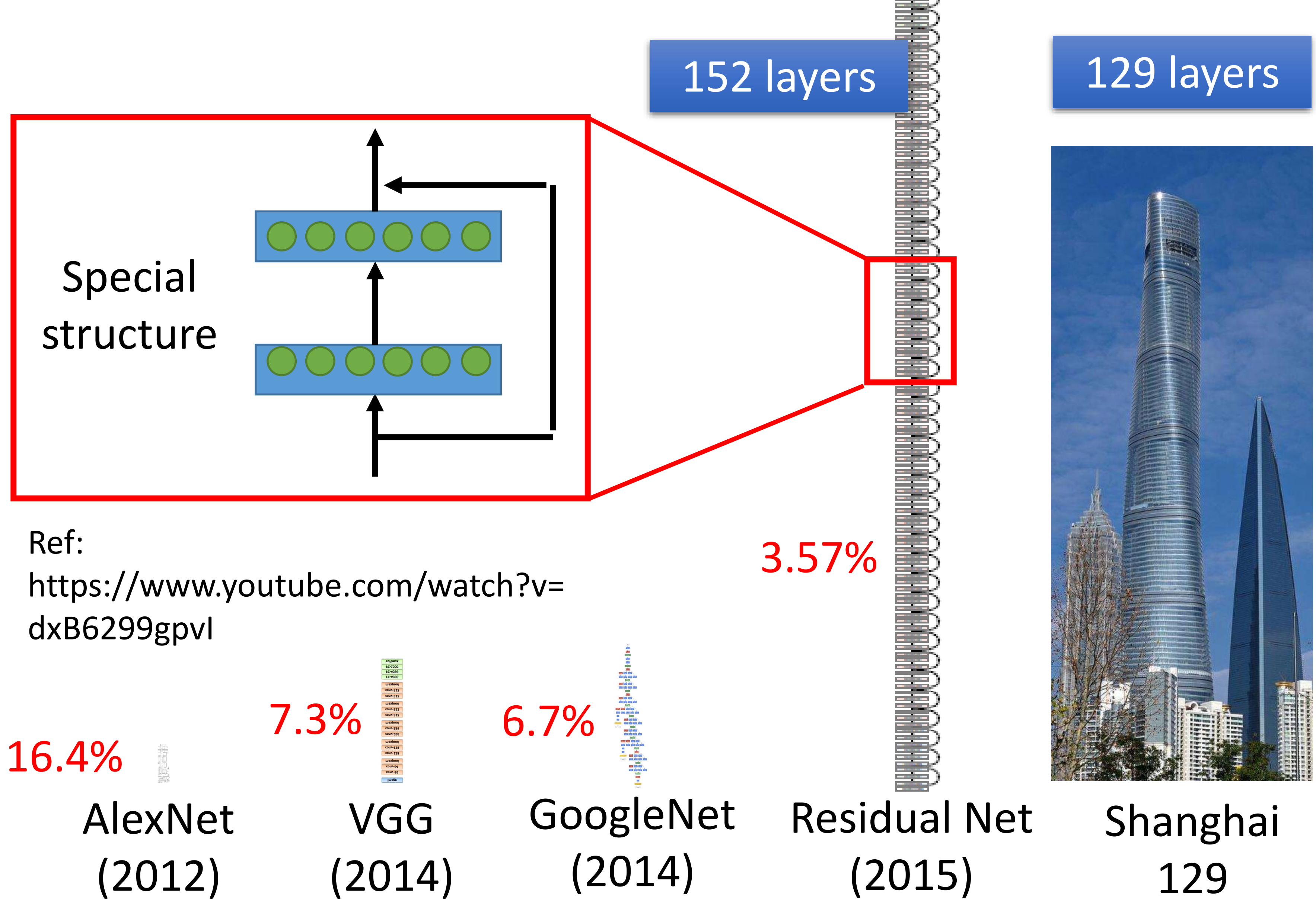
6.7%



22 layers

GoogleNet (2014)

Deep = Many hidden layers



Deeper is Better?

Layer X Size	Word Error Rate (%)
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
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

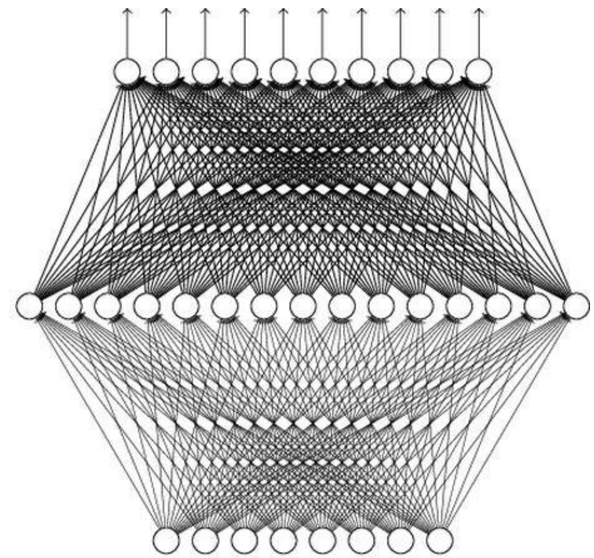
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Why "Deep" neural network not "Fat" neural network?

(next lecture)

Backpropagation

Gradient Descent

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

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

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

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

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

Millions of parameters ……

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

Chain Rule

Case 1

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

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

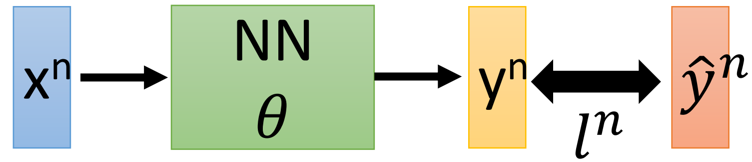
Case 2

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

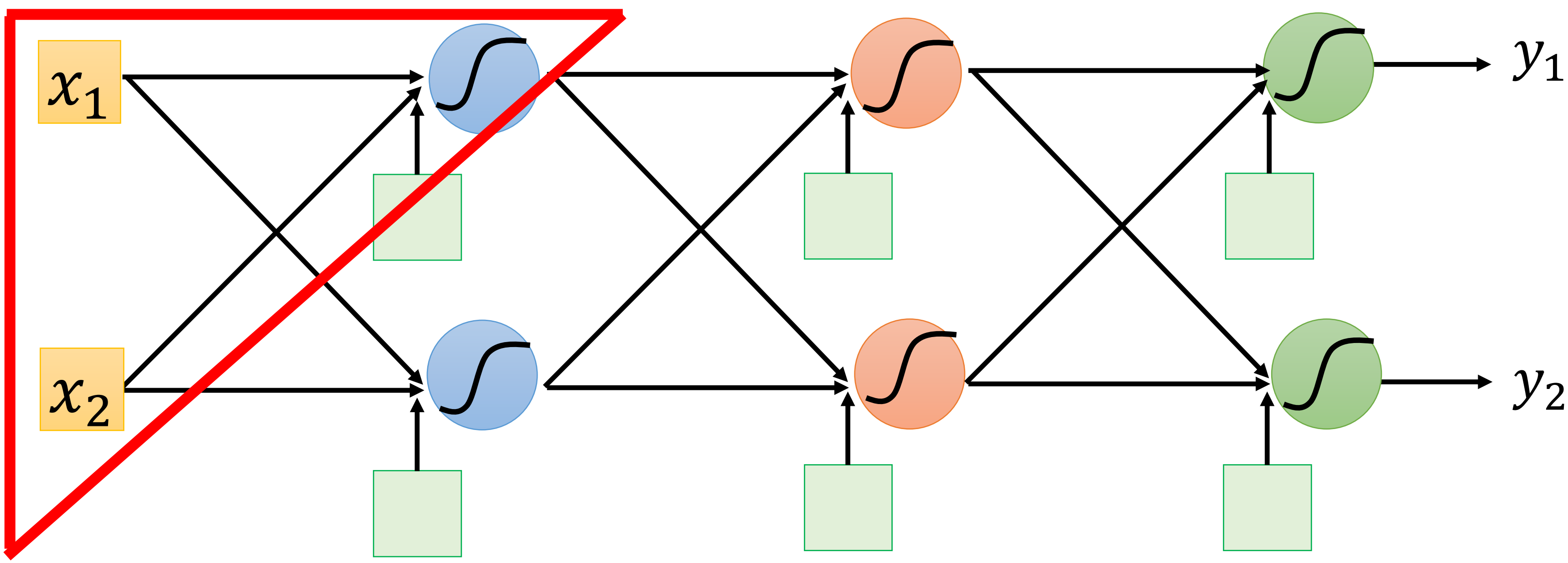
$$\begin{tikzcd}
 & \Delta x \arrow[dr] & \\
\Delta s \arrow[ur] \arrow[dr] & & \Delta z \\
 & \Delta y \arrow[ur] &
\end{tikzcd}$$

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

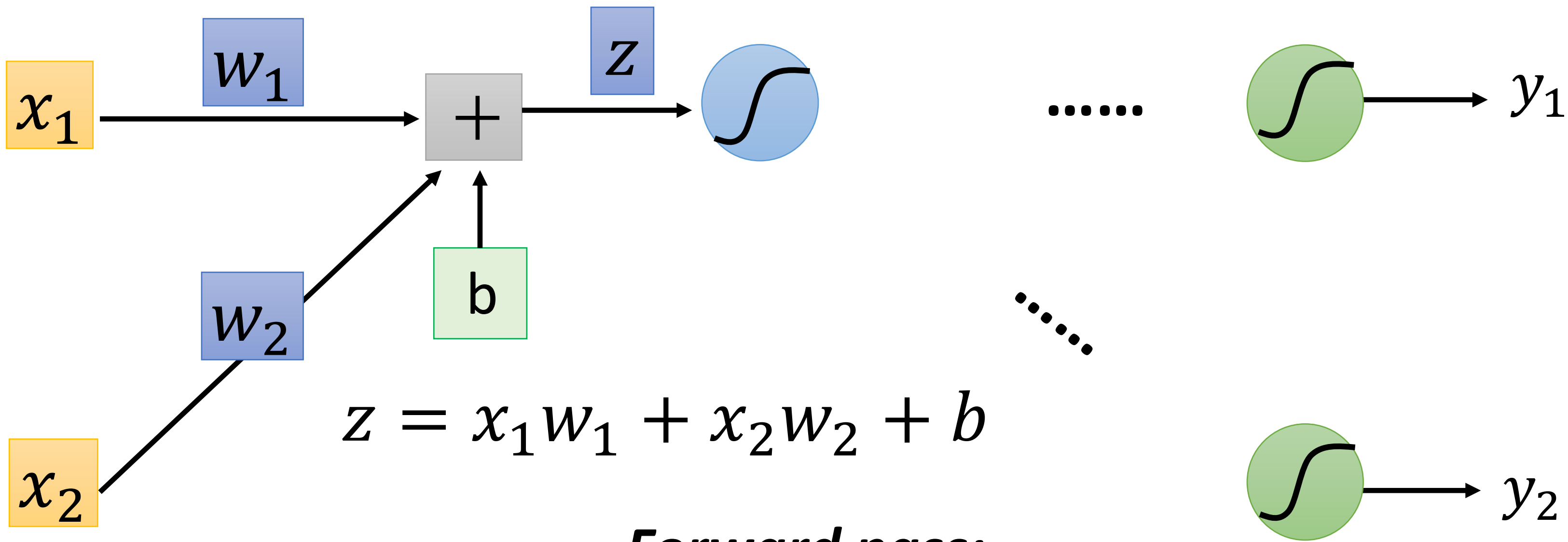
Backpropagation



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



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)

<u>Forward pass:</u>

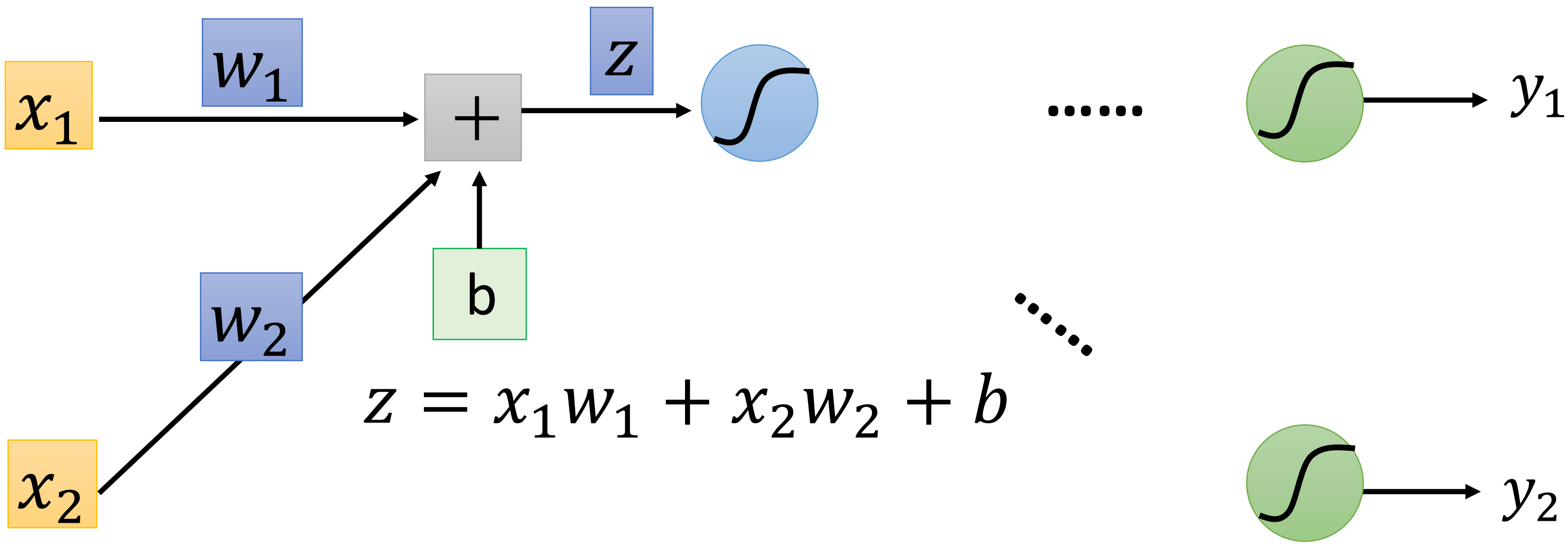
Compute $\partial z / \partial w$ for all parameters

<u>Backward pass:</u>

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

Backpropagation – Forward pass

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



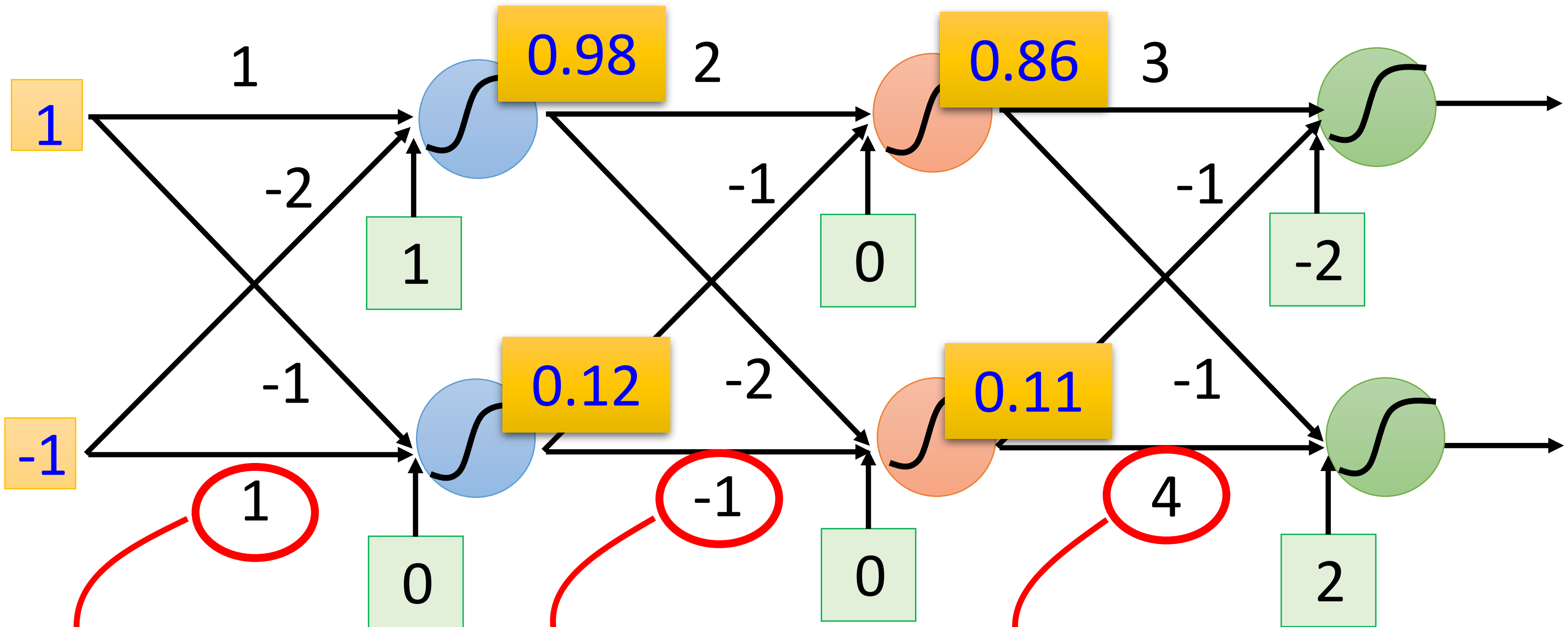
$$z = x_1 w_1 + x_2 w_2 + b$$

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

The value of the input connected by the weight

Backpropagation – Forward pass

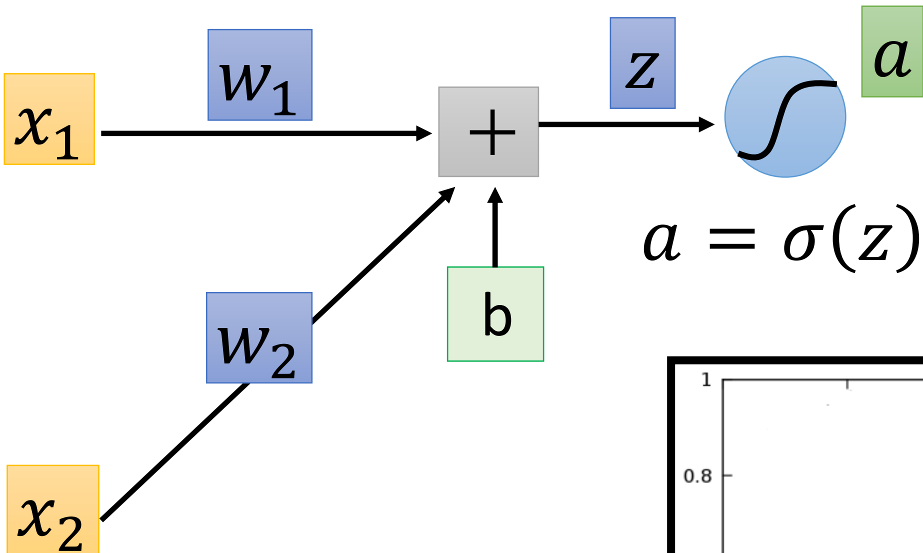
Compute $\partial z/\partial w$ for all parameters



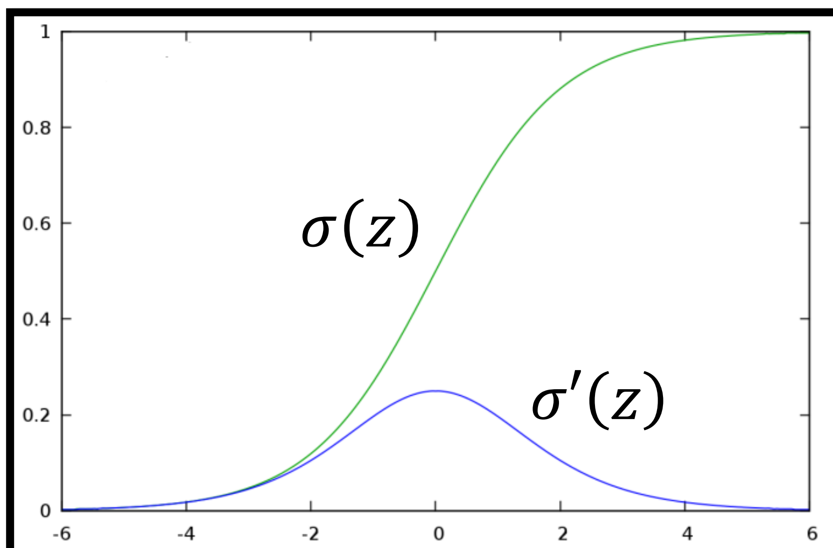
$$\frac{\partial z}{\partial w} = -1 \qquad \frac{\partial z}{\partial w} = 0.12 \qquad \frac{\partial z}{\partial w} = 0.11$$

Backpropagation – Backward pass

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



$$a = \sigma(z)$$

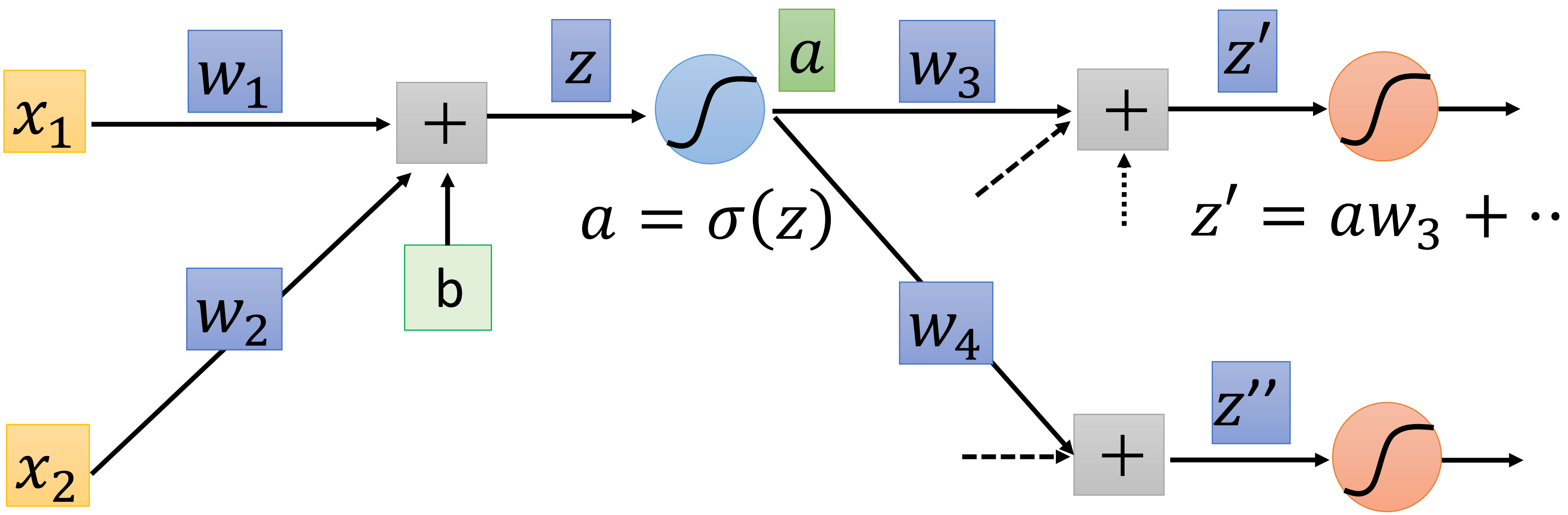


$$\frac{\partial l}{\partial z} = \underline{\frac{\partial a}{\partial z}} \frac{\partial l}{\partial a}$$

$\sigma'(z)$

Backpropagation – Backward pass

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



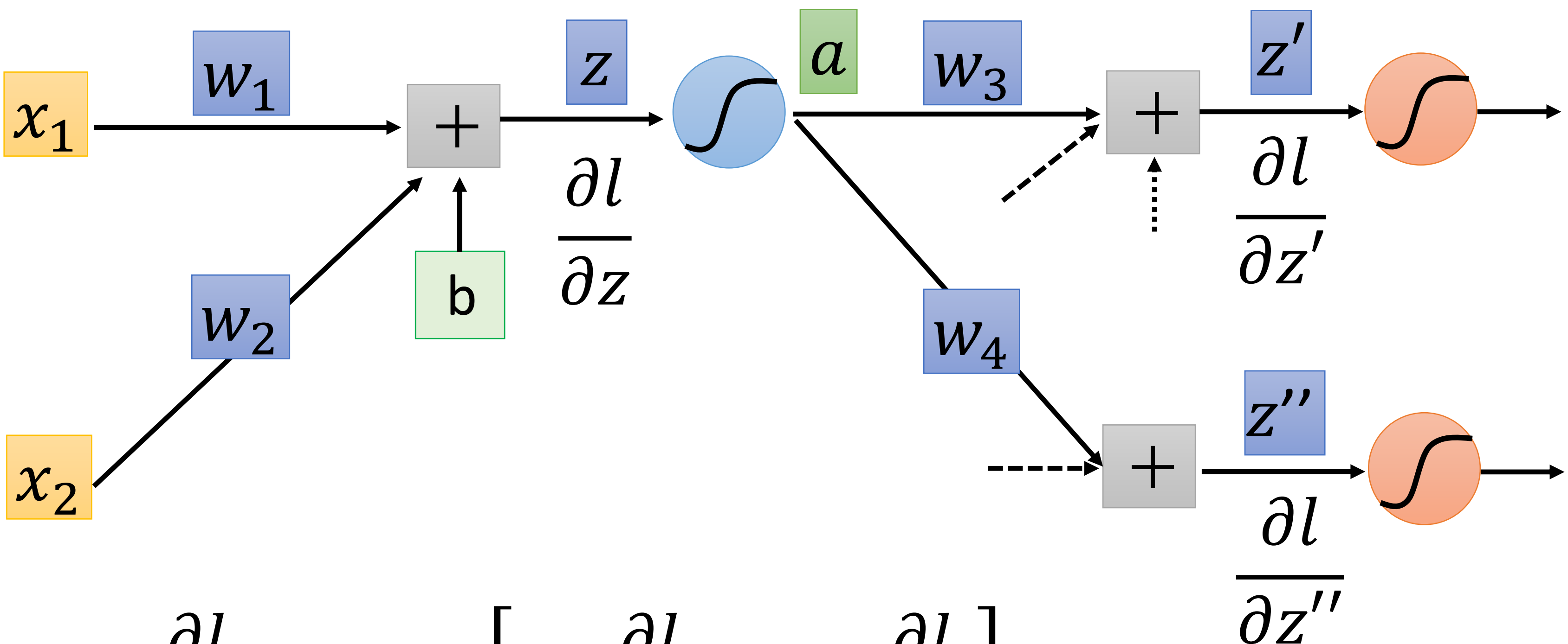
$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial l}{\partial a} \qquad \frac{\partial l}{\partial a} = \frac{\partial z'}{\partial a} \frac{\partial l}{\partial z'} + \frac{\partial z''}{\partial a} \frac{\partial l}{\partial z''}$$ (Chain rule)

w_3 ? w_4 ?

Assumed it's known

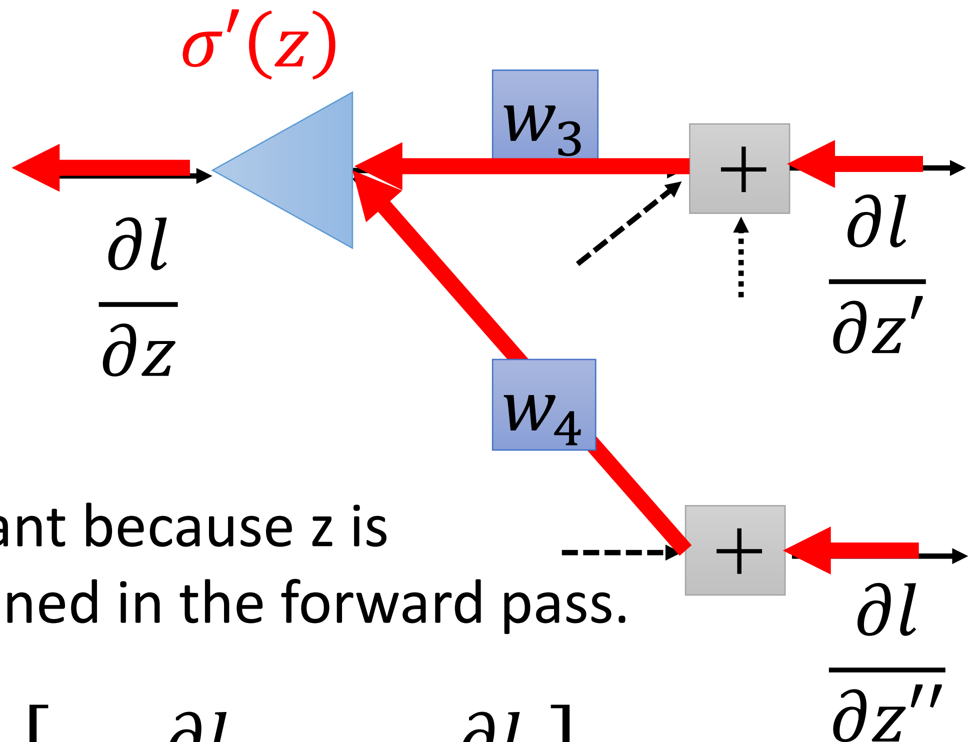
Backpropagation – Backward pass

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

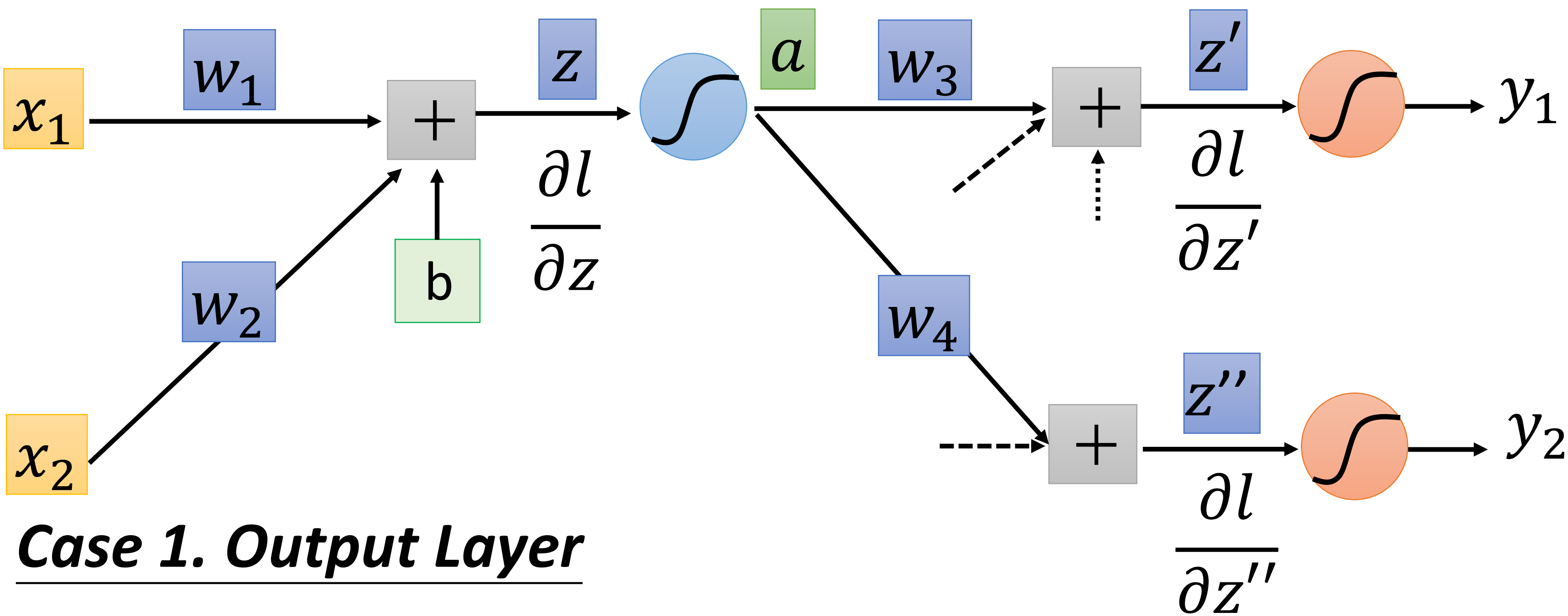


$\sigma'(z)$ is a constant because z is already determined in the forward pass.

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

Backpropagation – Backward pass

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



Case 1. Output Layer

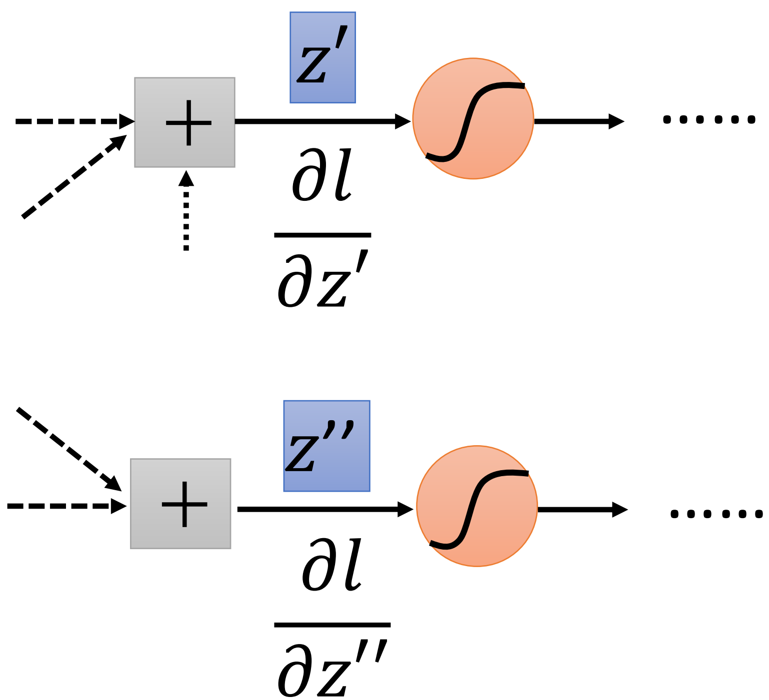
$$\frac{\partial l}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial l}{\partial y_1} \qquad \frac{\partial l}{\partial z''} = \frac{\partial y_2}{\partial z''} \frac{\partial l}{\partial y_2}$$

Done!

Backpropagation – Backward pass

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

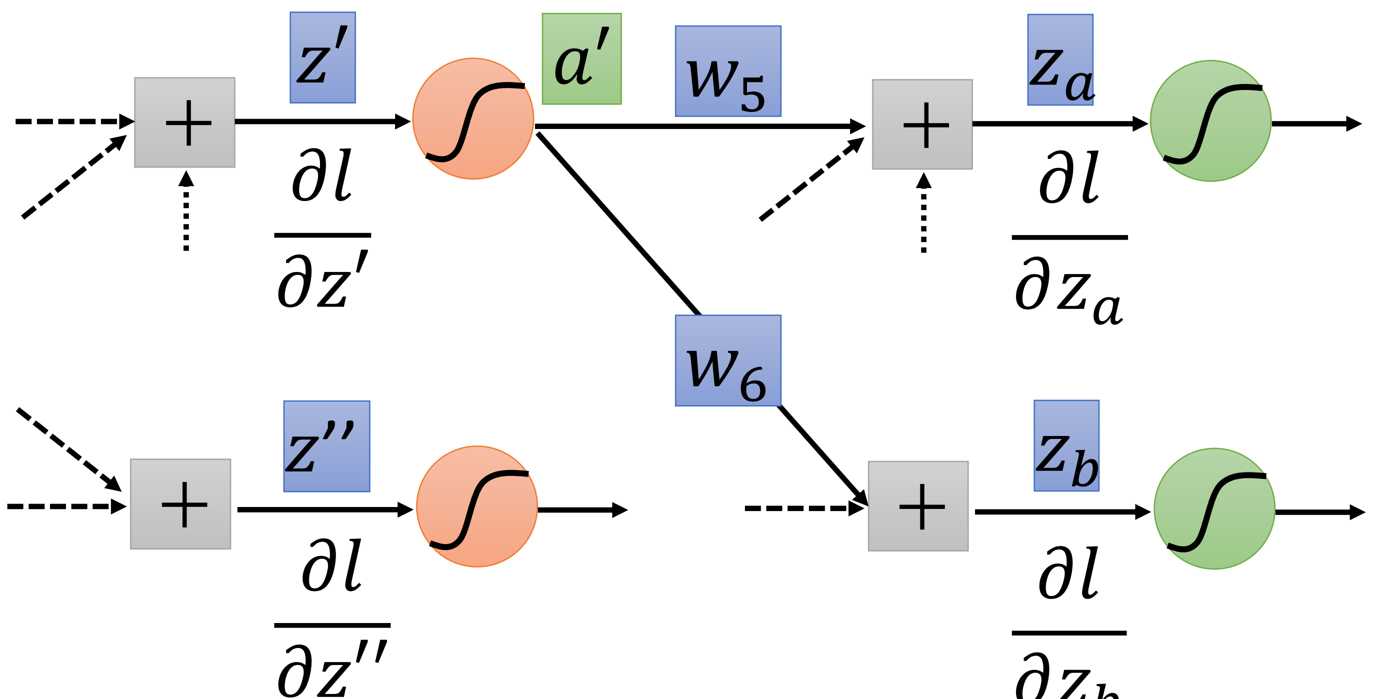
Case 2. Not Output Layer



Backpropagation – Backward pass

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

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

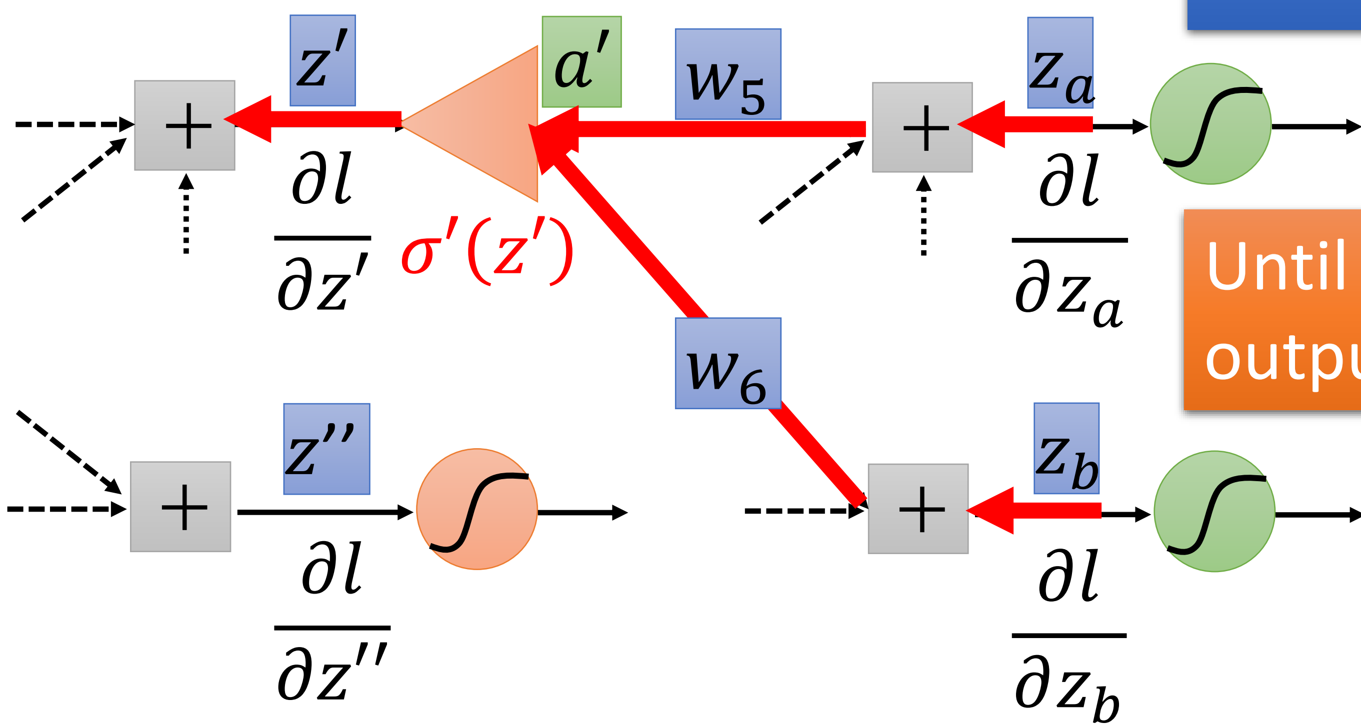


Backpropagation – Backward pass

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

Case 2. Not Output Layer

Compute $\partial l/\partial z$ recursively

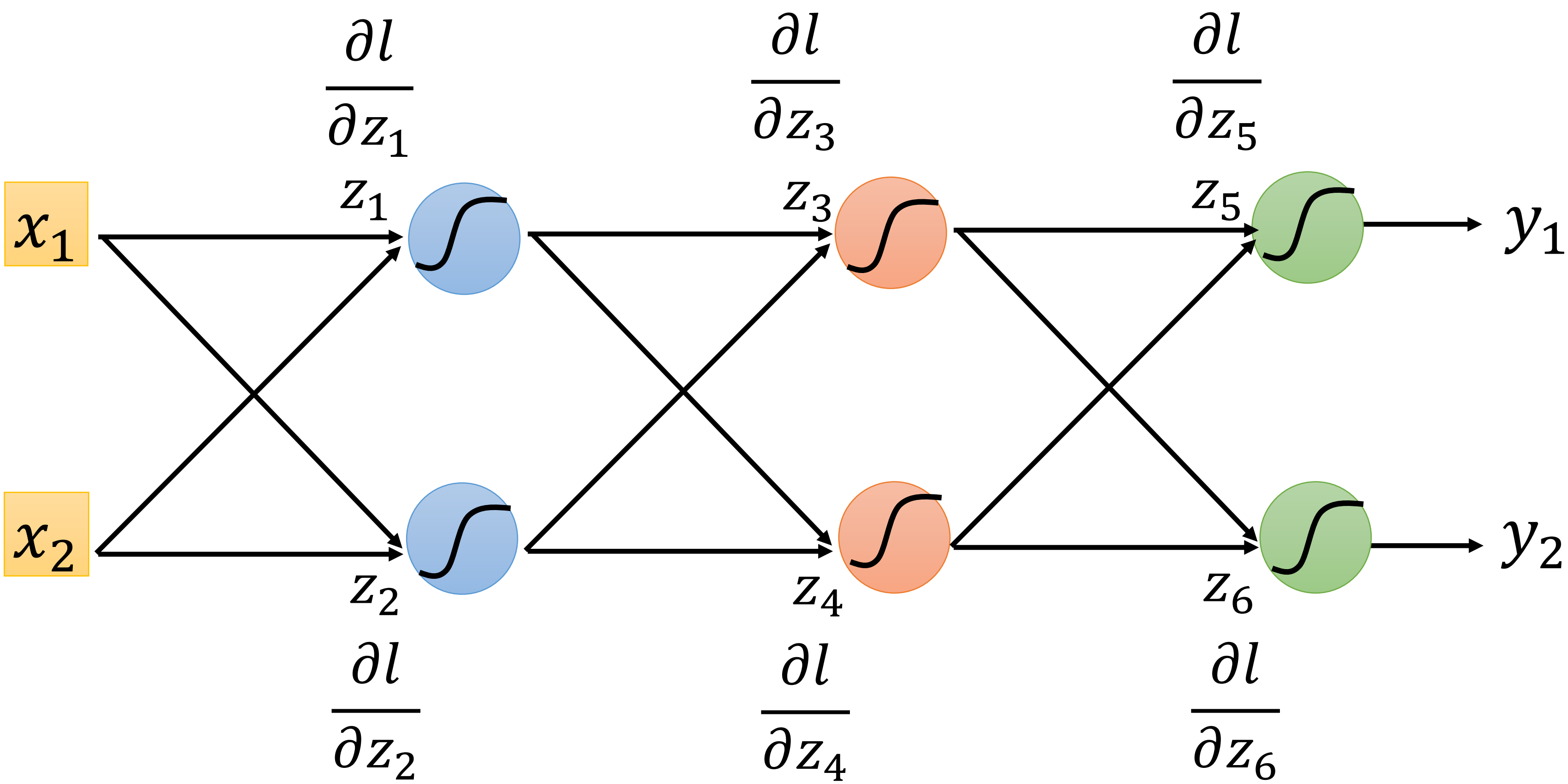


Until we reach the output layer ……

Backpropagation – Backward Pass

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

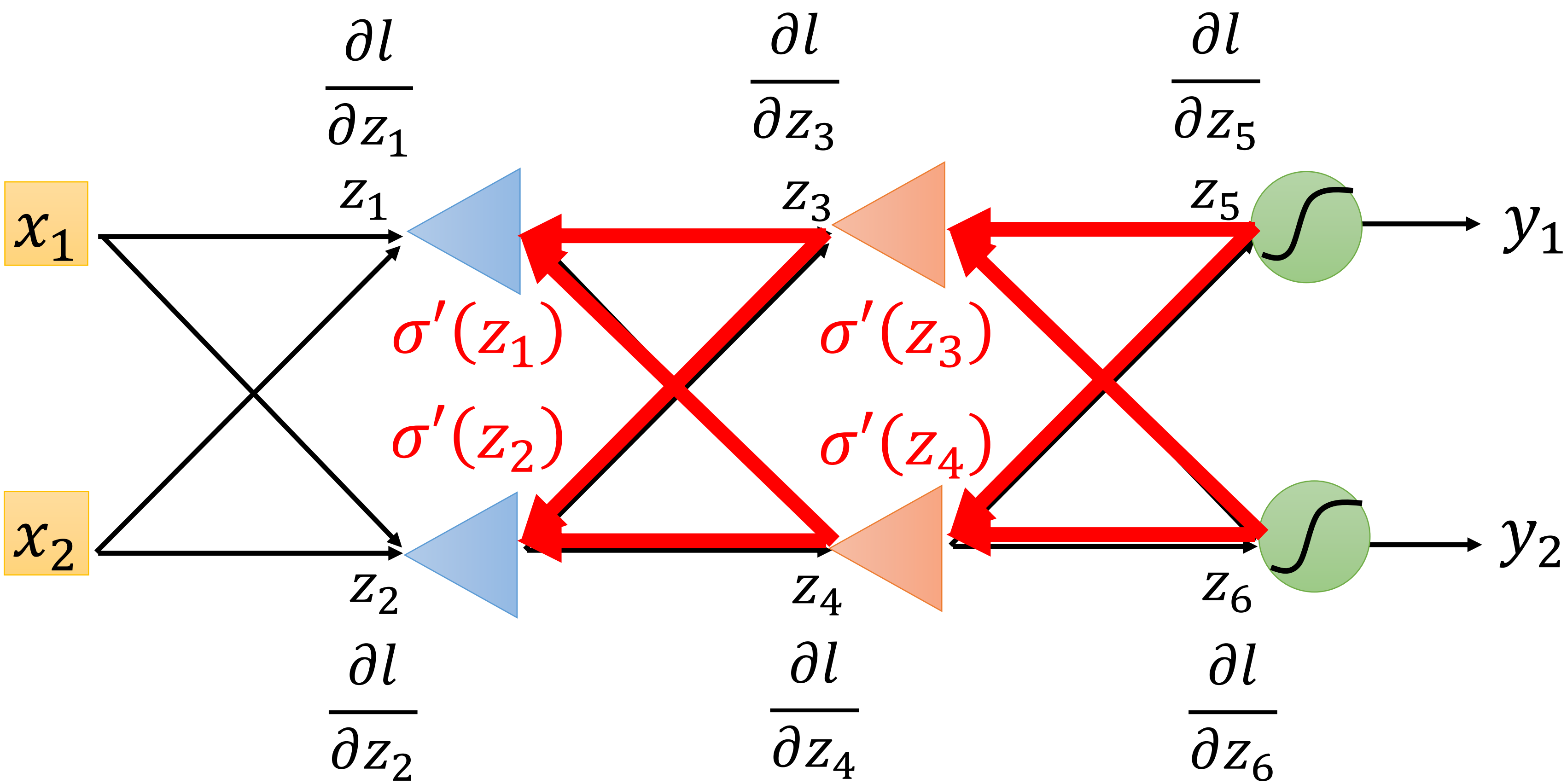
Compute $\partial l / \partial z$ from the output layer



Backpropagation – Backward Pass

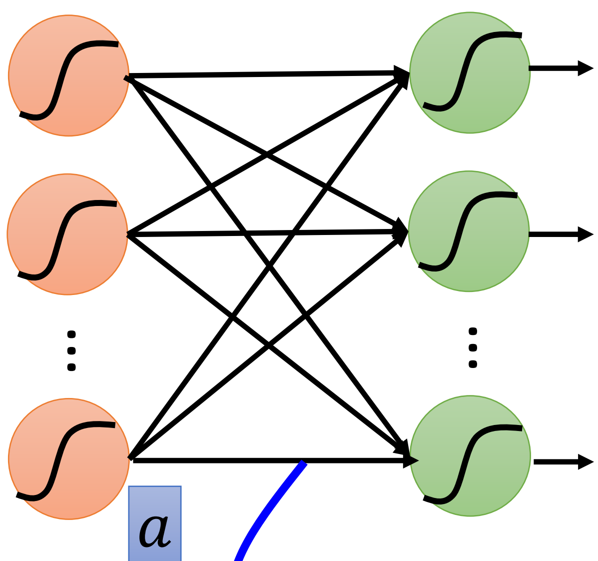
Compute $\partial l / \partial z$ for all activation function inputs z

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

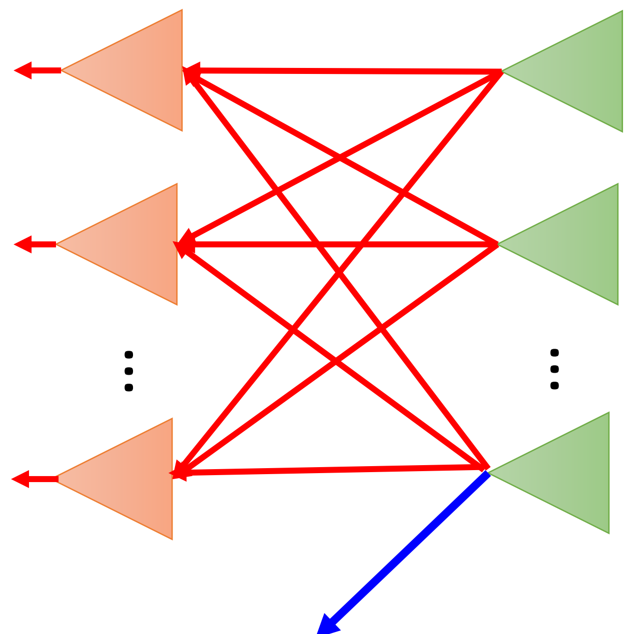


Backpropagation – Summary

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