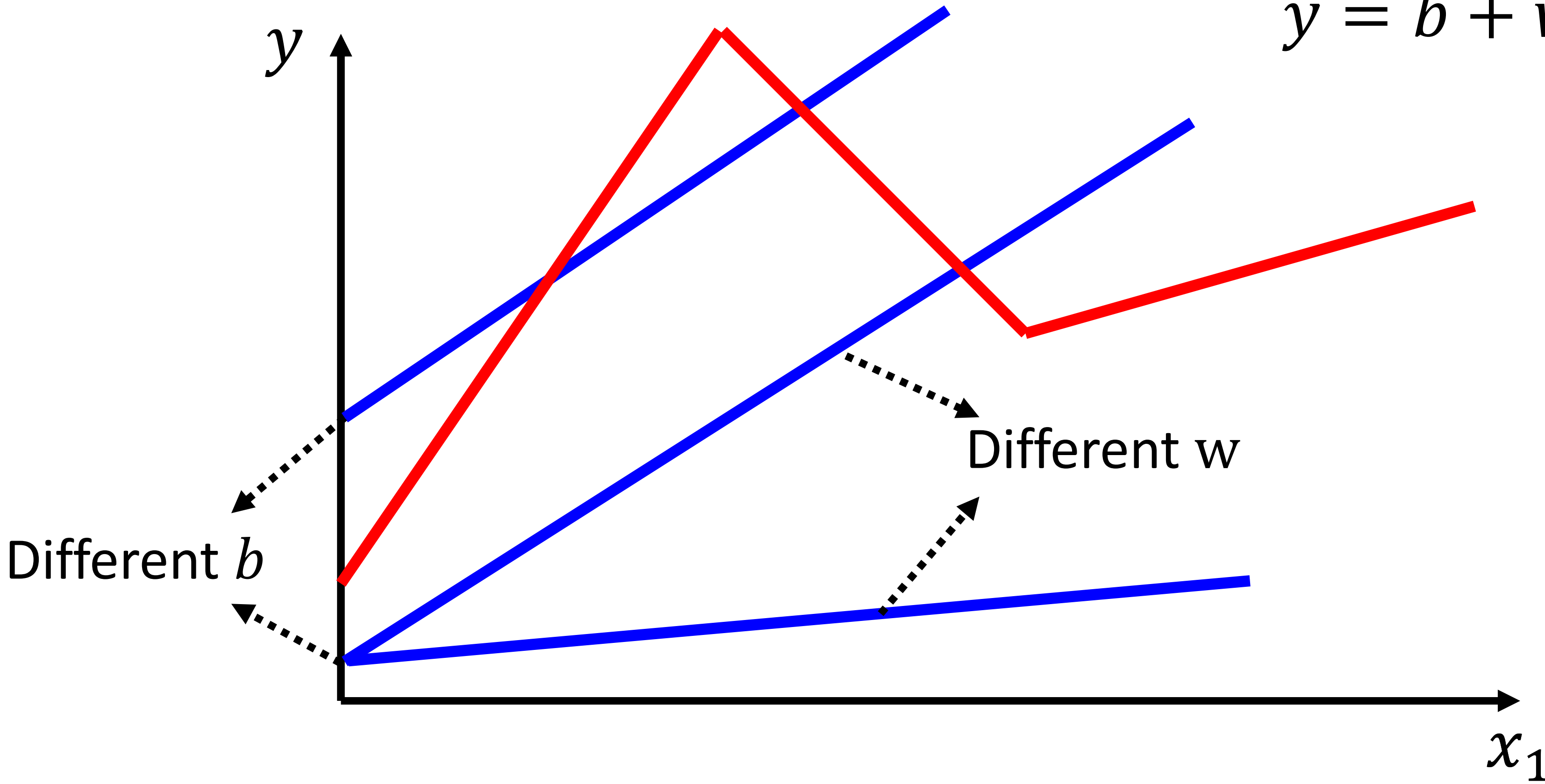
Something before Real Deep Learning
- From Linear Model to **Neural Network**

Yizhen Lao

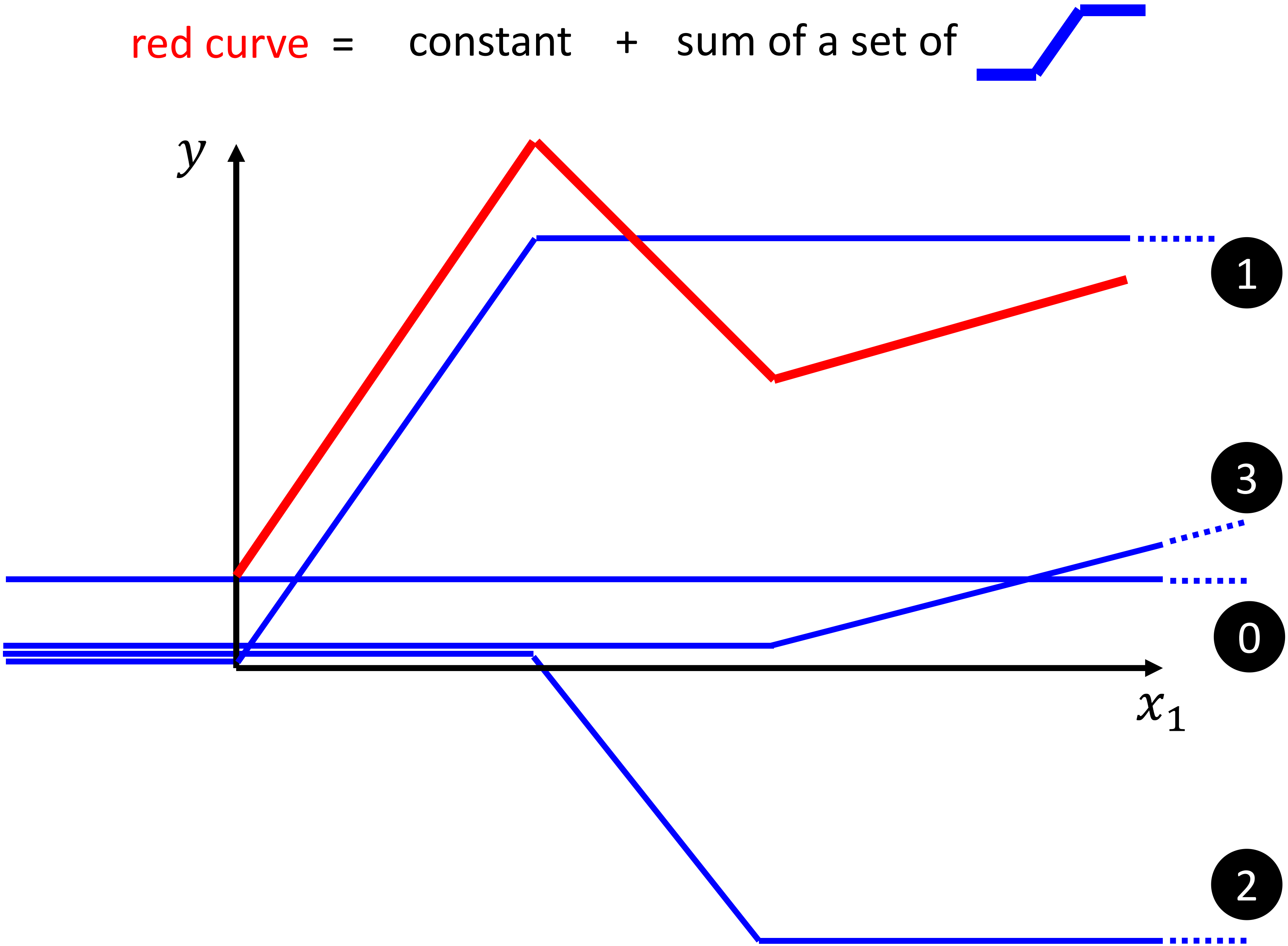
Linear models are too simple … we need more sophisticated modes.

$$y = b + wx_1$$



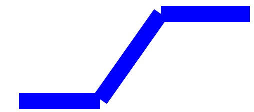
Linear models have severe limitation. ***Model Bias***

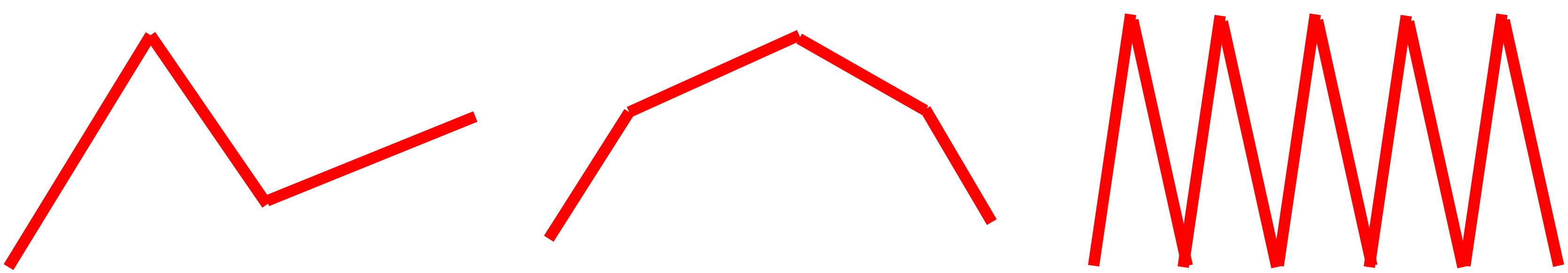
We need a more flexible model!

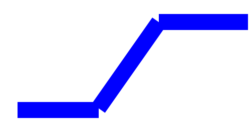


red curve = constant + sum of a set of
y
x_1
1
3
0
2

All Piecewise Linear Curves

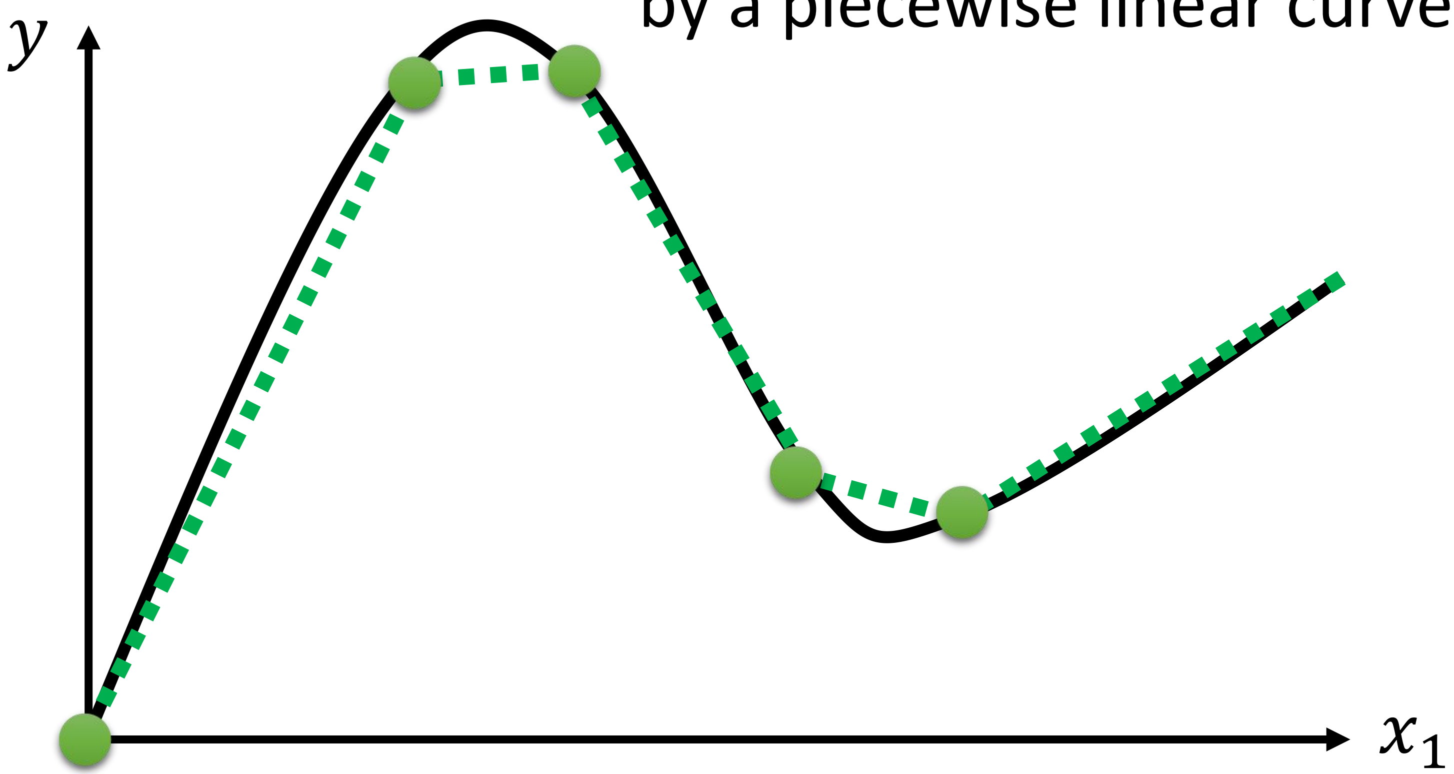
= constant + sum of a set of 



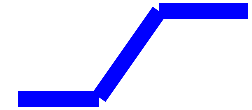
More pieces require more 

Beyond Piecewise Linear?

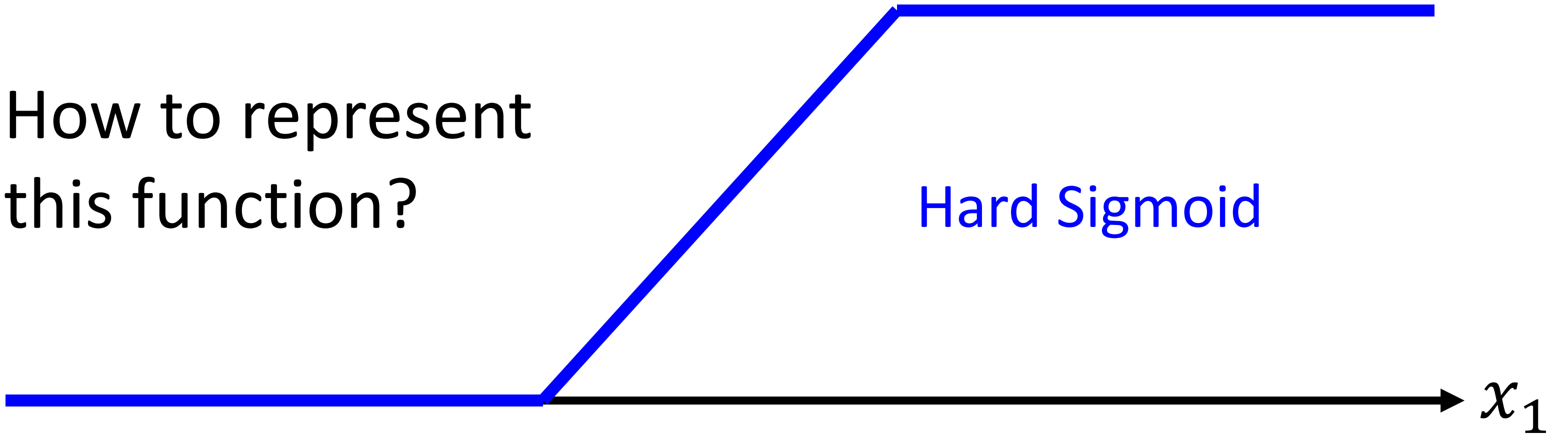
Approximate continuous curve by a piecewise linear curve.



To have good approximation, we need sufficient pieces.

red curve = constant + sum of a set of 

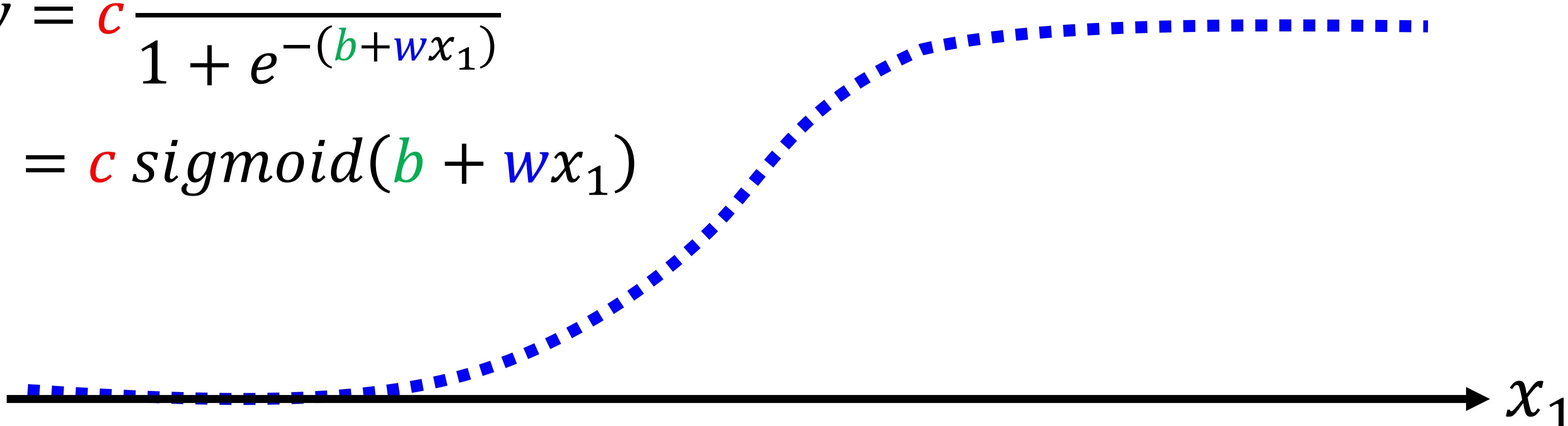
How to represent this function?

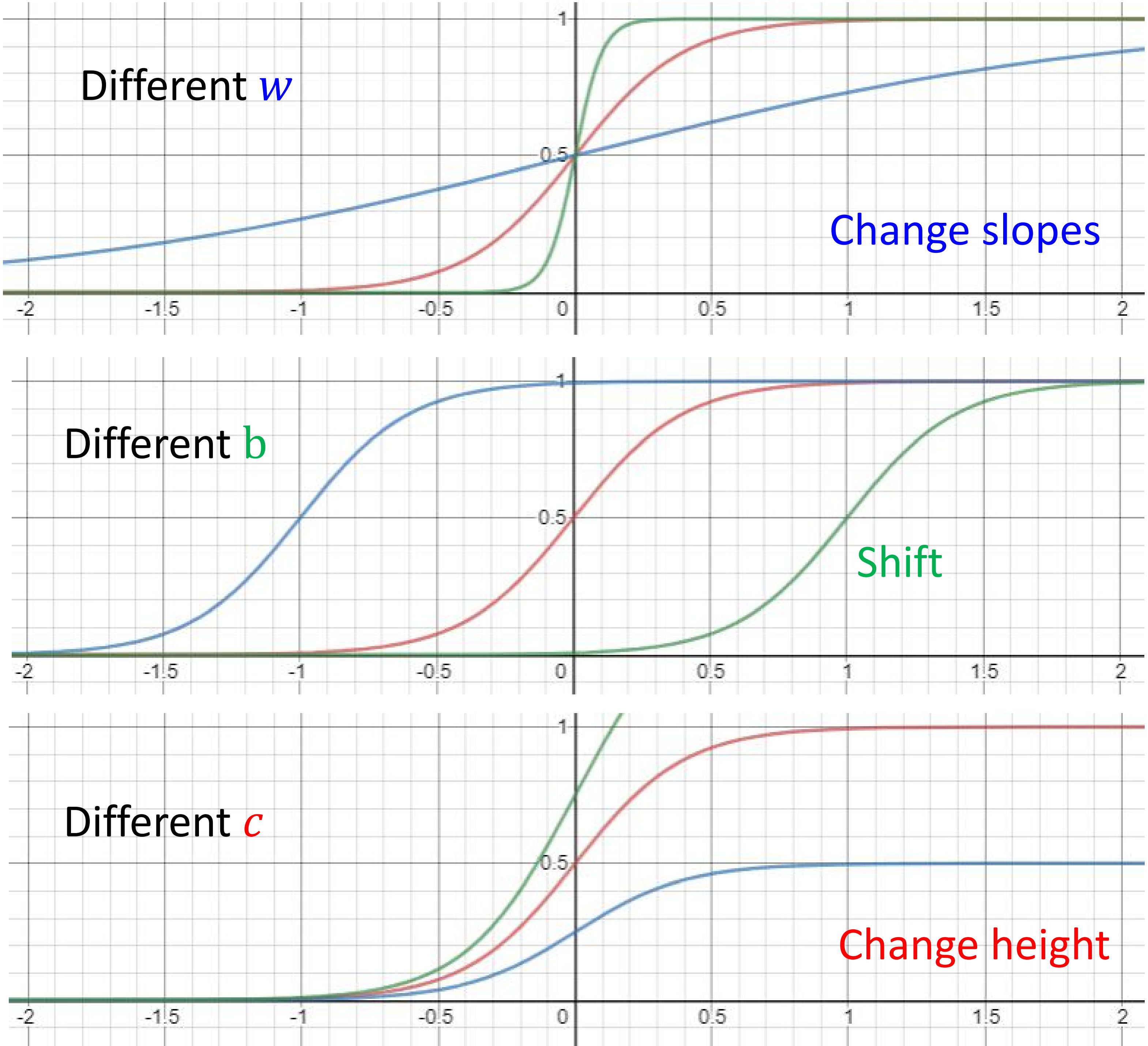


Sigmoid Function

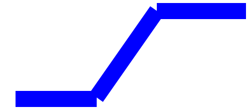
$$y = c\,\frac{1}{1+e^{-(b+wx_1)}}$$

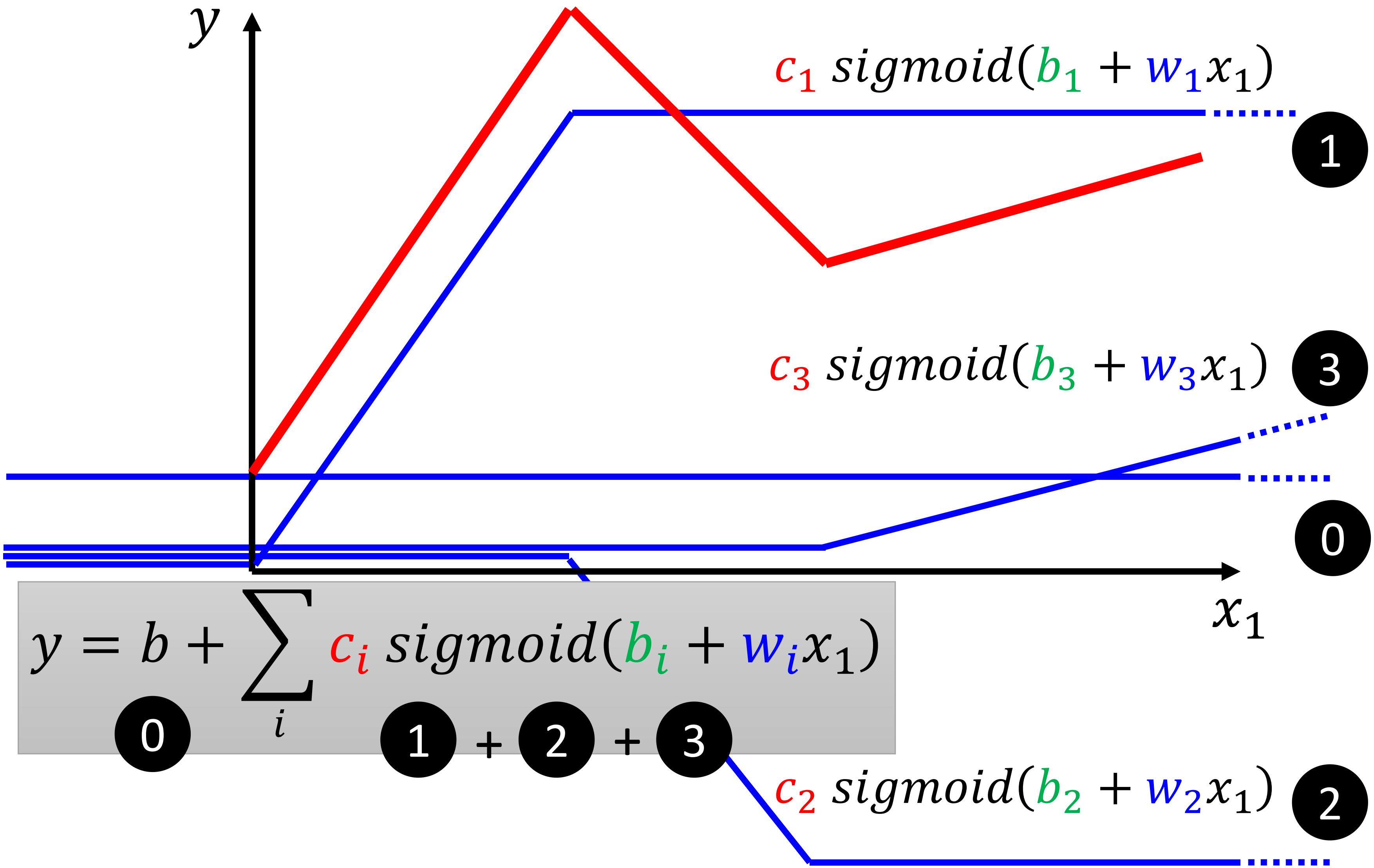
$$= c\ sigmoid(b + wx_1)$$





Different w
Change slopes
Different b
Shift
Different c
Change height
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

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

$$y = \underline{b + w x_1}$$

$$y = b + \sum_i c_i \; sigmoid(\underline{b_i + w_i x_1})$$

$$y = b + \underline{\sum_j w_j x_j}$$

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

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

j: 1,2,3
no. of features

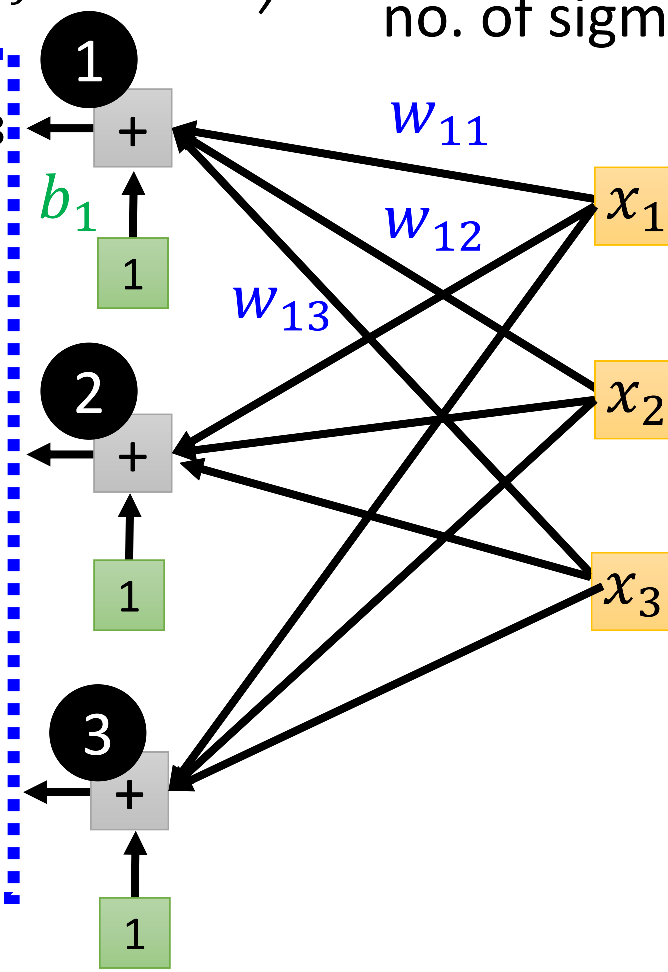
i: 1,2,3
no. of sigmoid

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

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

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

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

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

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

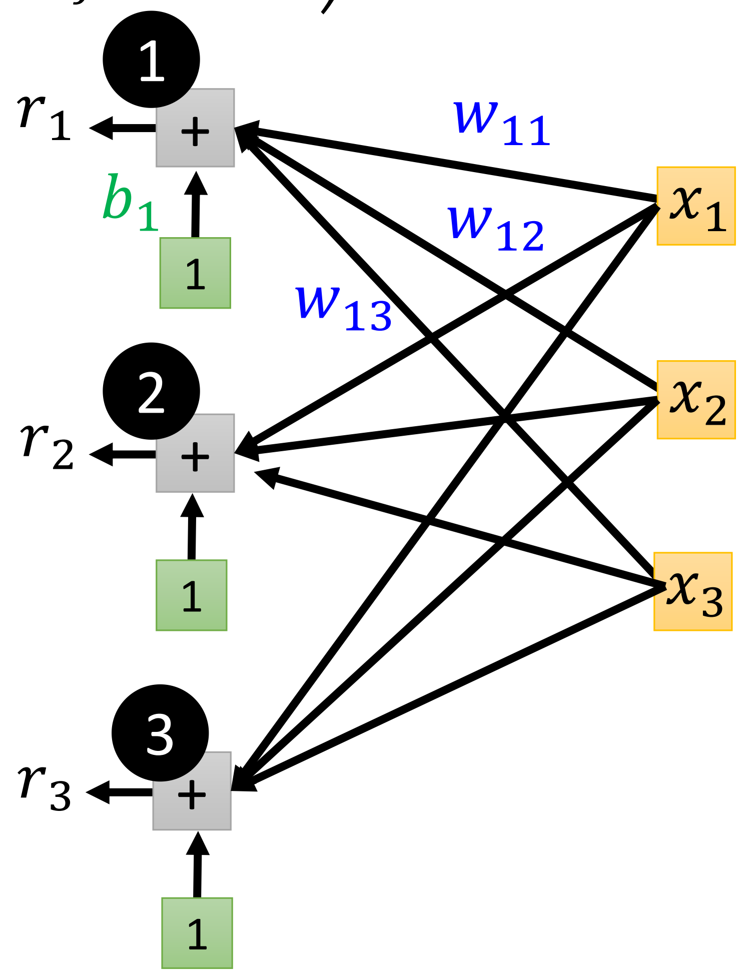
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$

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

i: 1,2,3

j: 1,2,3

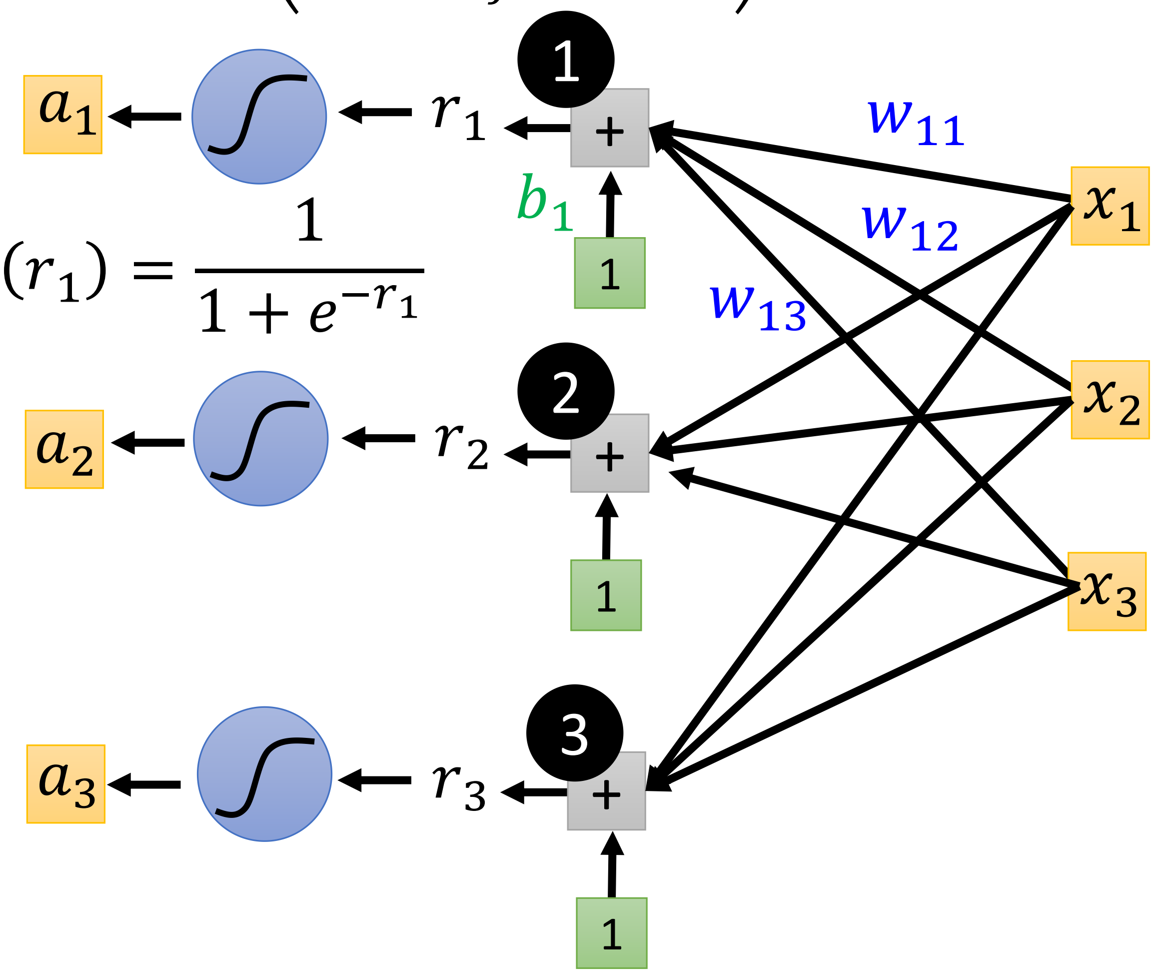
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$



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

i: 1,2,3

j: 1,2,3



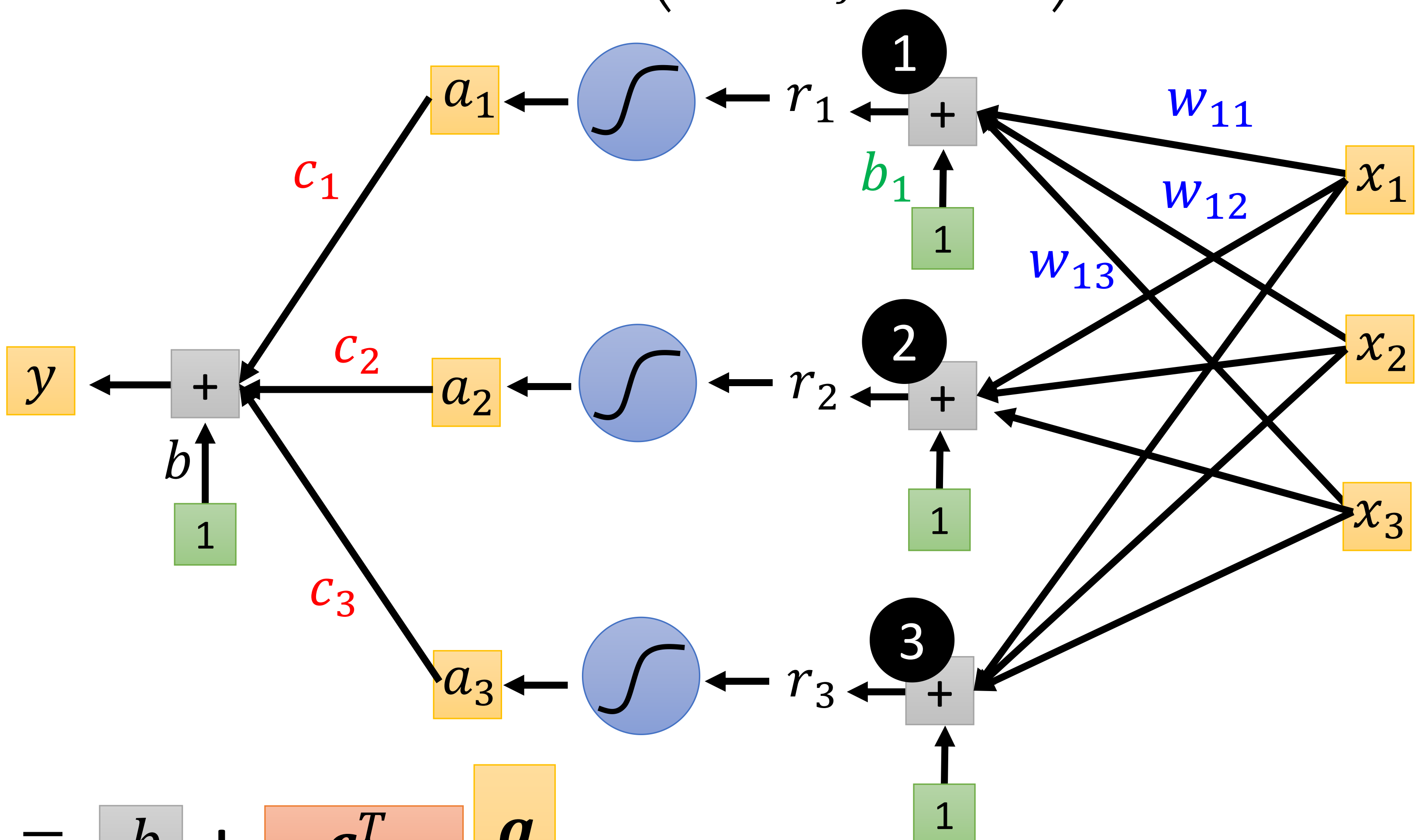
$$a_1 = sigmoid(r_1) = \frac{1}{1 + e^{-r_1}}$$

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

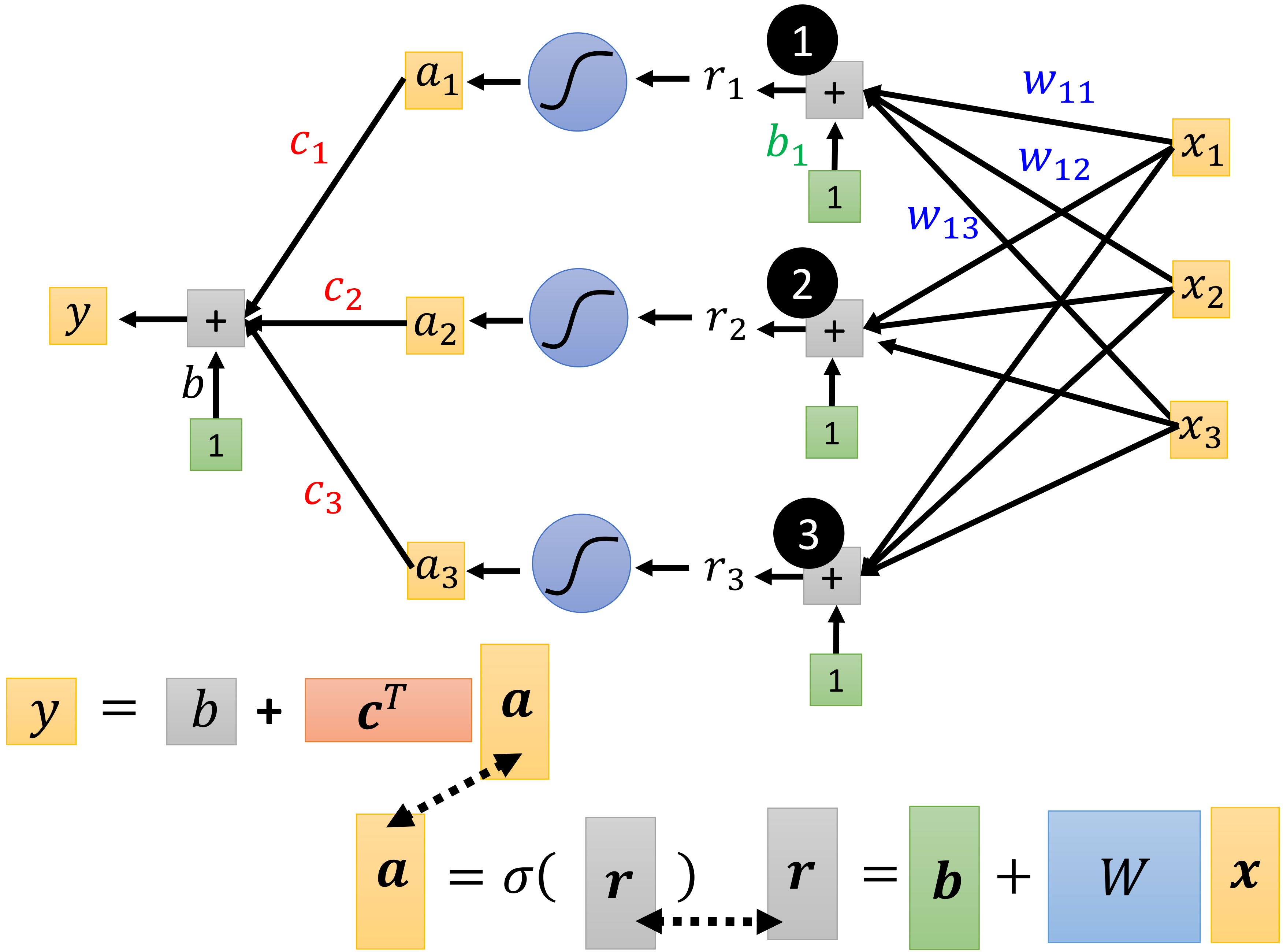
$$y = b + \sum_i c_i \; sigmoid\left(b_i + \sum_j w_{ij} x_j \right)$$

i: 1,2,3

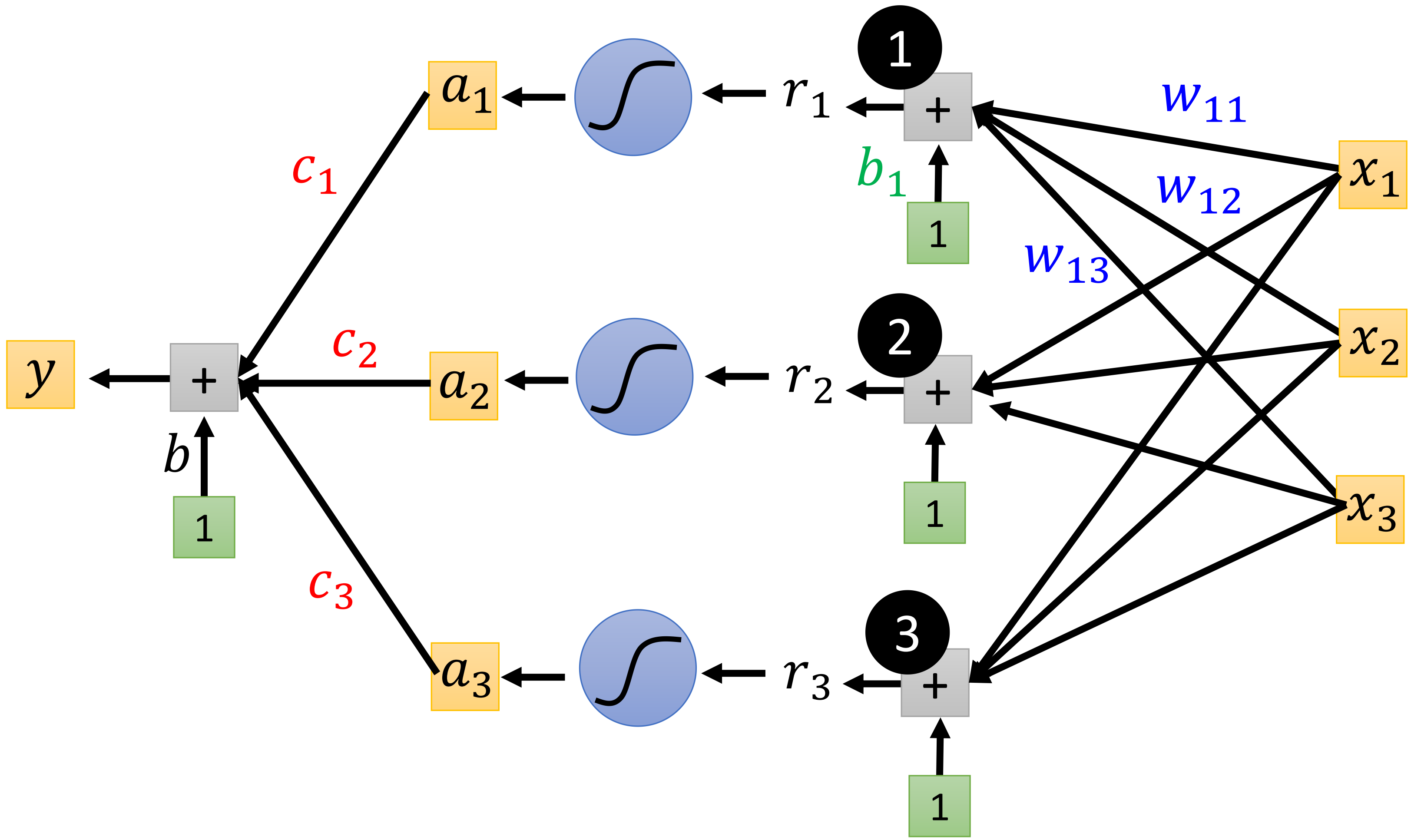
j: 1,2,3



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



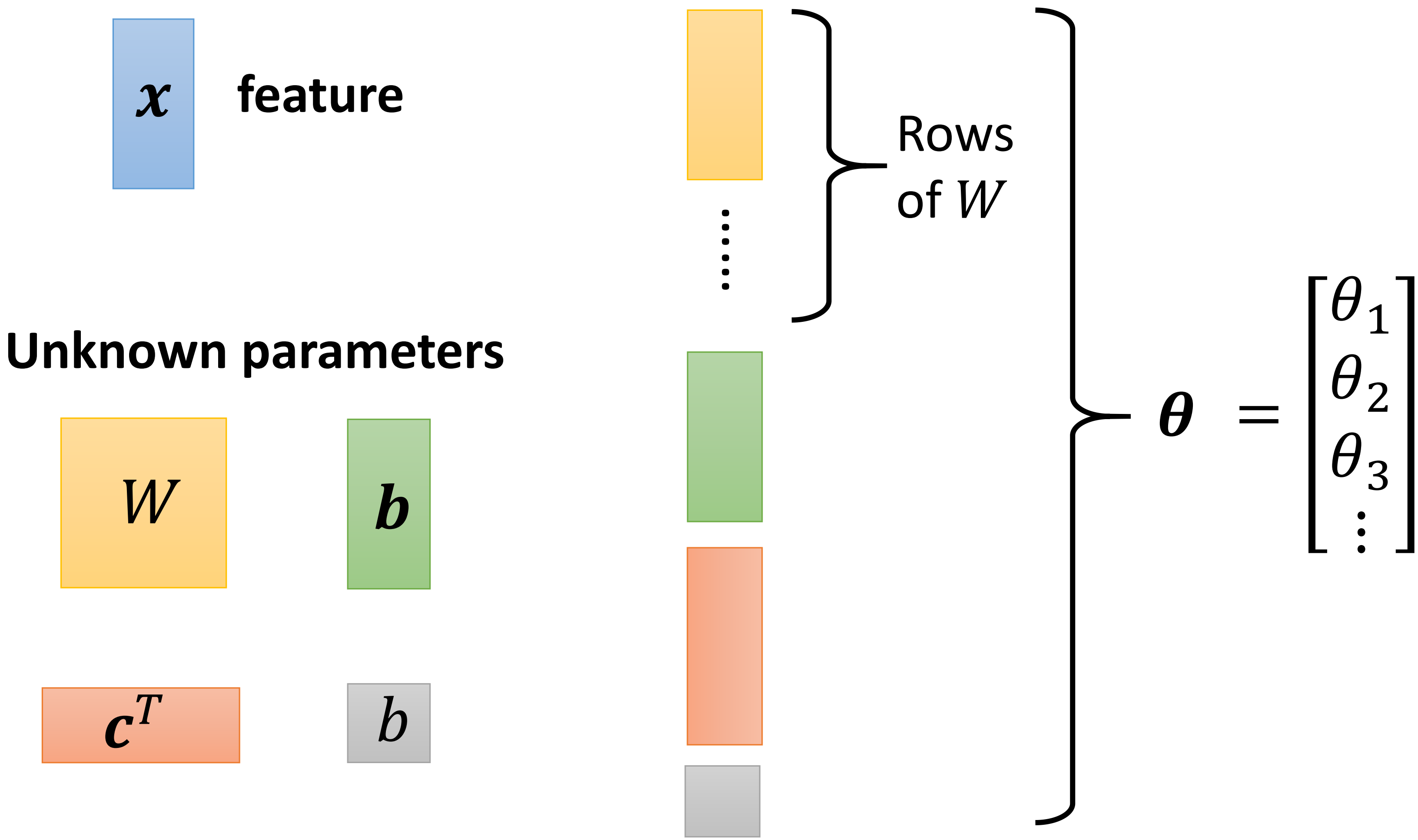
1
2
3
a_1
a_2
a_3
r_1
r_2
r_3
c_1
c_2
c_3
b_1
b
w_{11}
w_{12}
w_{13}
x_1
x_2
x_3
y
+
1
$y = b + \boldsymbol{c}^T \boldsymbol{a}$
$\boldsymbol{a} = \sigma(\boldsymbol{r})$
$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$



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

Function with unknown parameters

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



Back to ML Framework

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

Back to ML Framework

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

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

Optimization of New Model

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

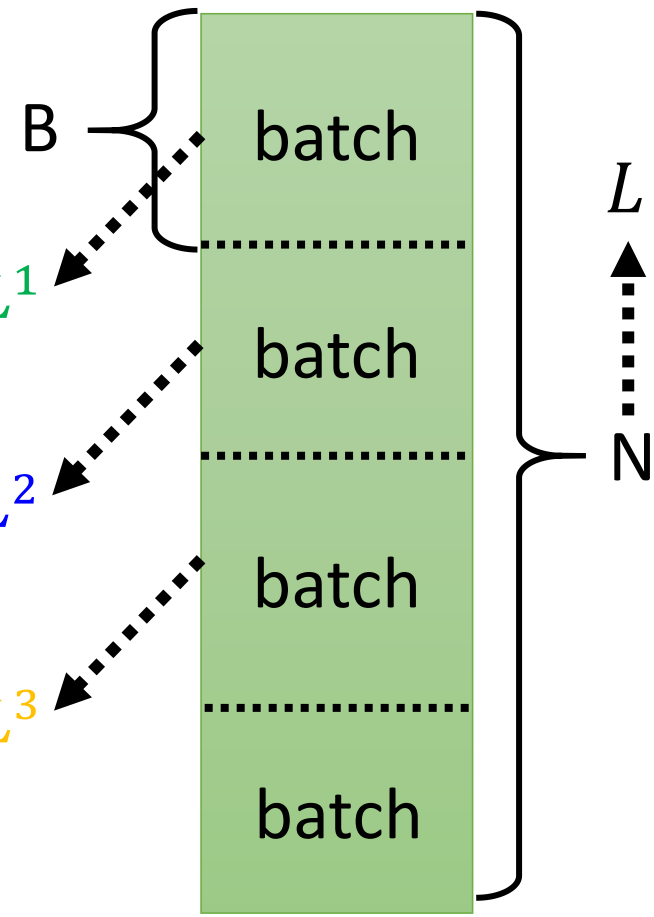
- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g} = \nabla L^1(\boldsymbol{\theta}^0)$ $\quad L^1$

  update $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L^2(\boldsymbol{\theta}^1)$ $\quad L^2$

  update $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L^3(\boldsymbol{\theta}^2)$ $\quad L^3$

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

1 **epoch** = see all the batches once



Optimization of New Model

Example 1

- 10,000 examples (N = 10,000)
- Batch size is 10 (B = 10)

How many update in **1 epoch**?

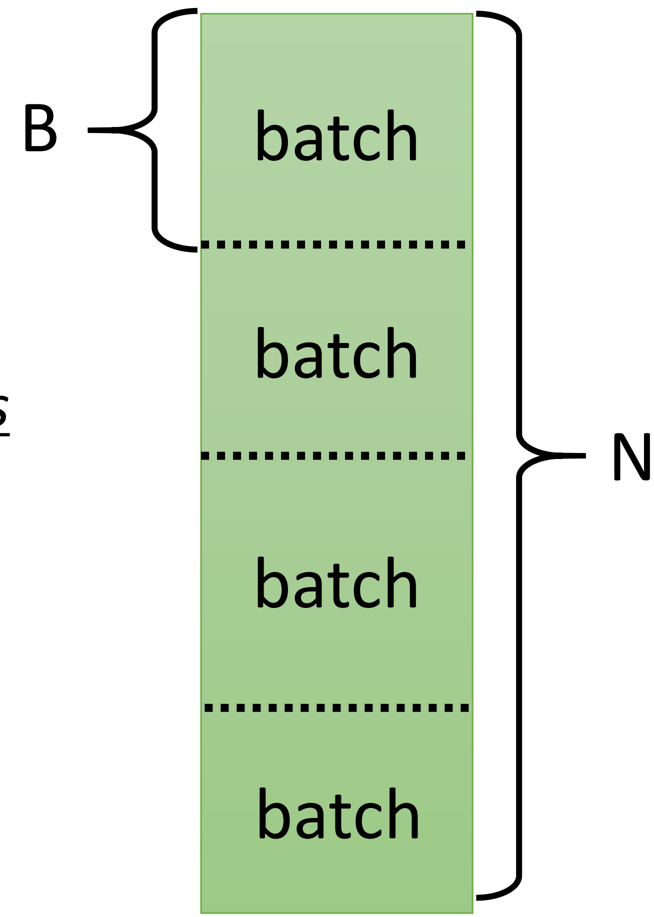
1,000 updates

Example 2

- 1,000 examples (N = 1,000)
- Batch size is 100 (B = 100)

How many update in **1 epoch**?

10 updates



Back to ML Framework

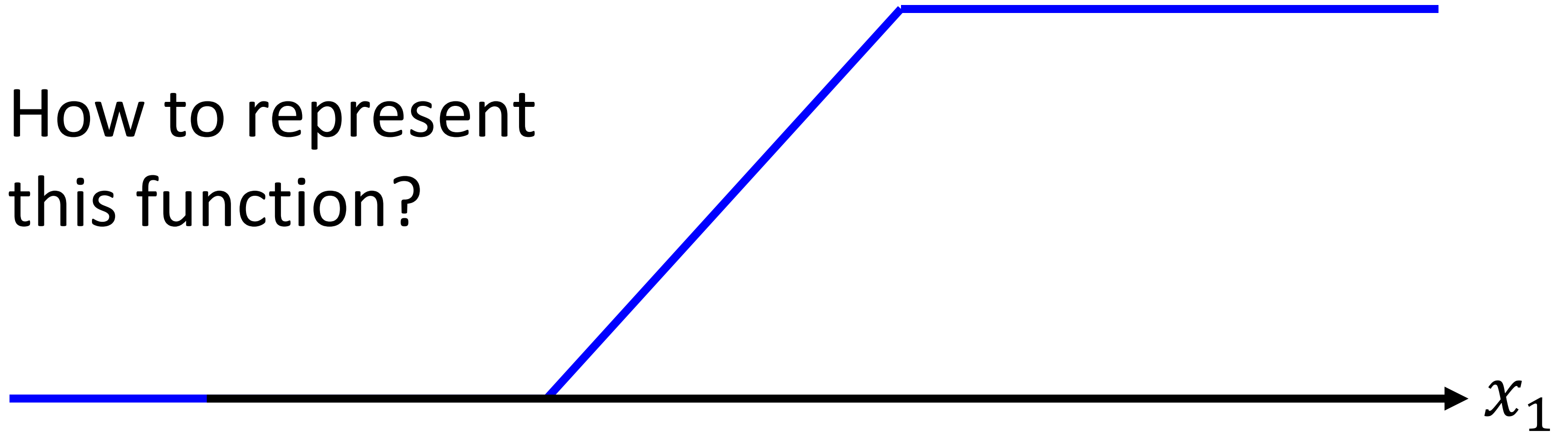
Step 1: function with unknown → Step 2: define loss from training data → Step 3: optimization

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

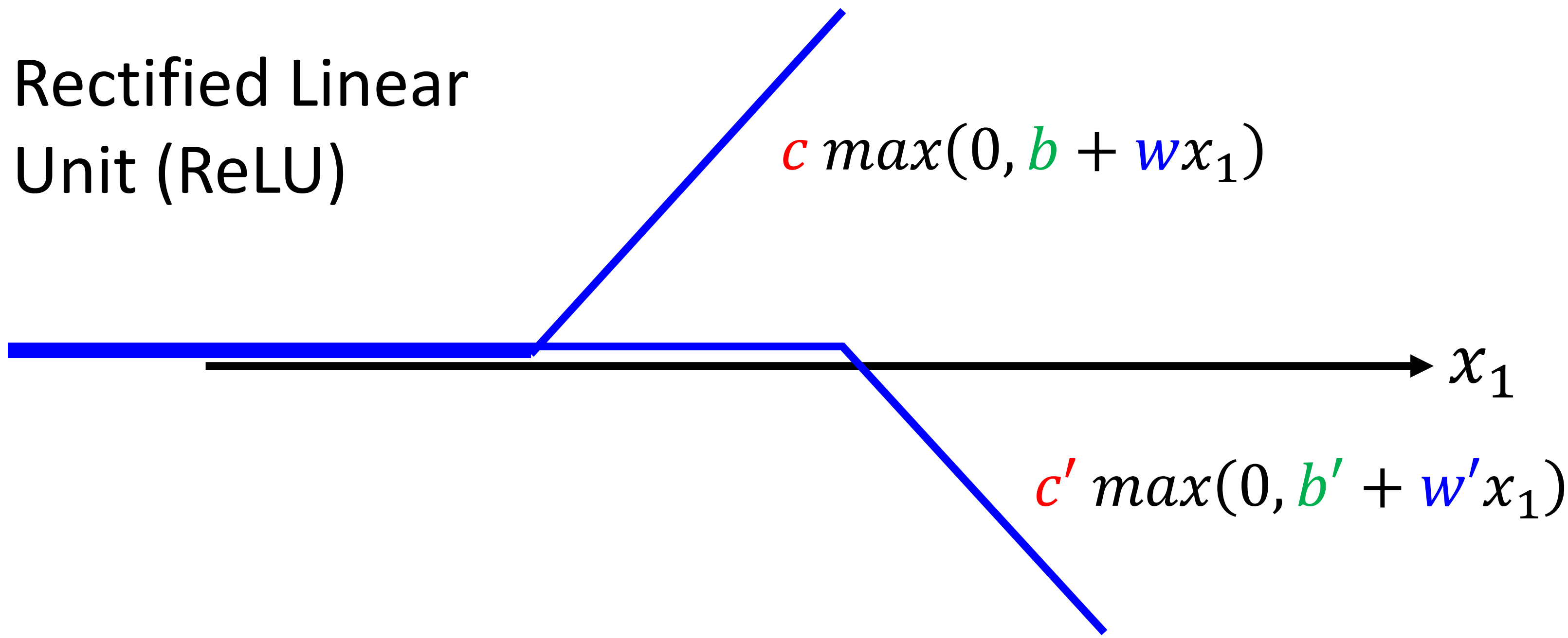
More variety of models …

Sigmoid → ReLU

How to represent this function?



Rectified Linear Unit (ReLU)



Sigmoid → ReLU

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

Activation function

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

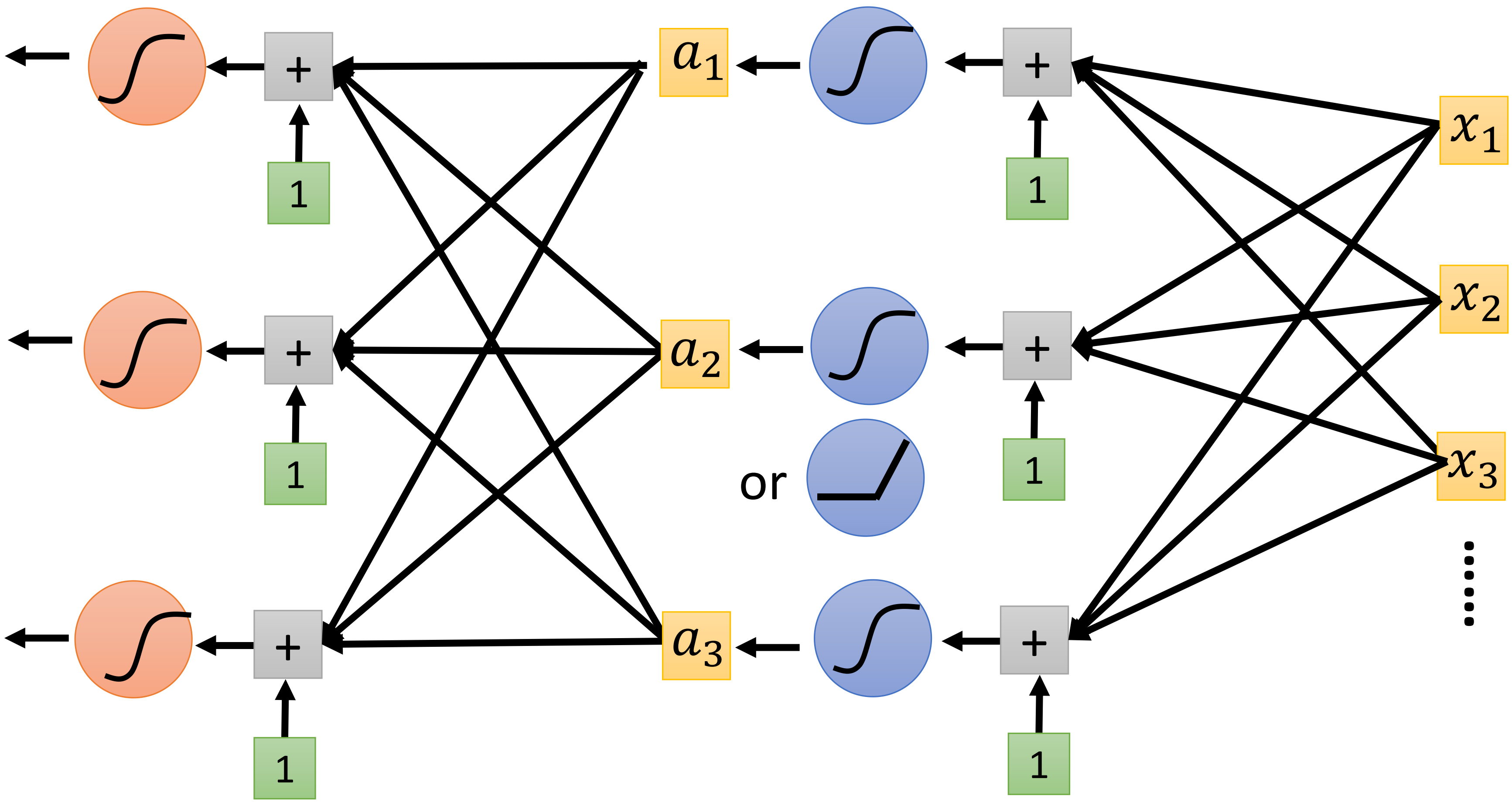
Which one is better?

Back to ML Framework

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

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

Even more variety of models …



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

Back to ML Framework

Step 1: function with unknown

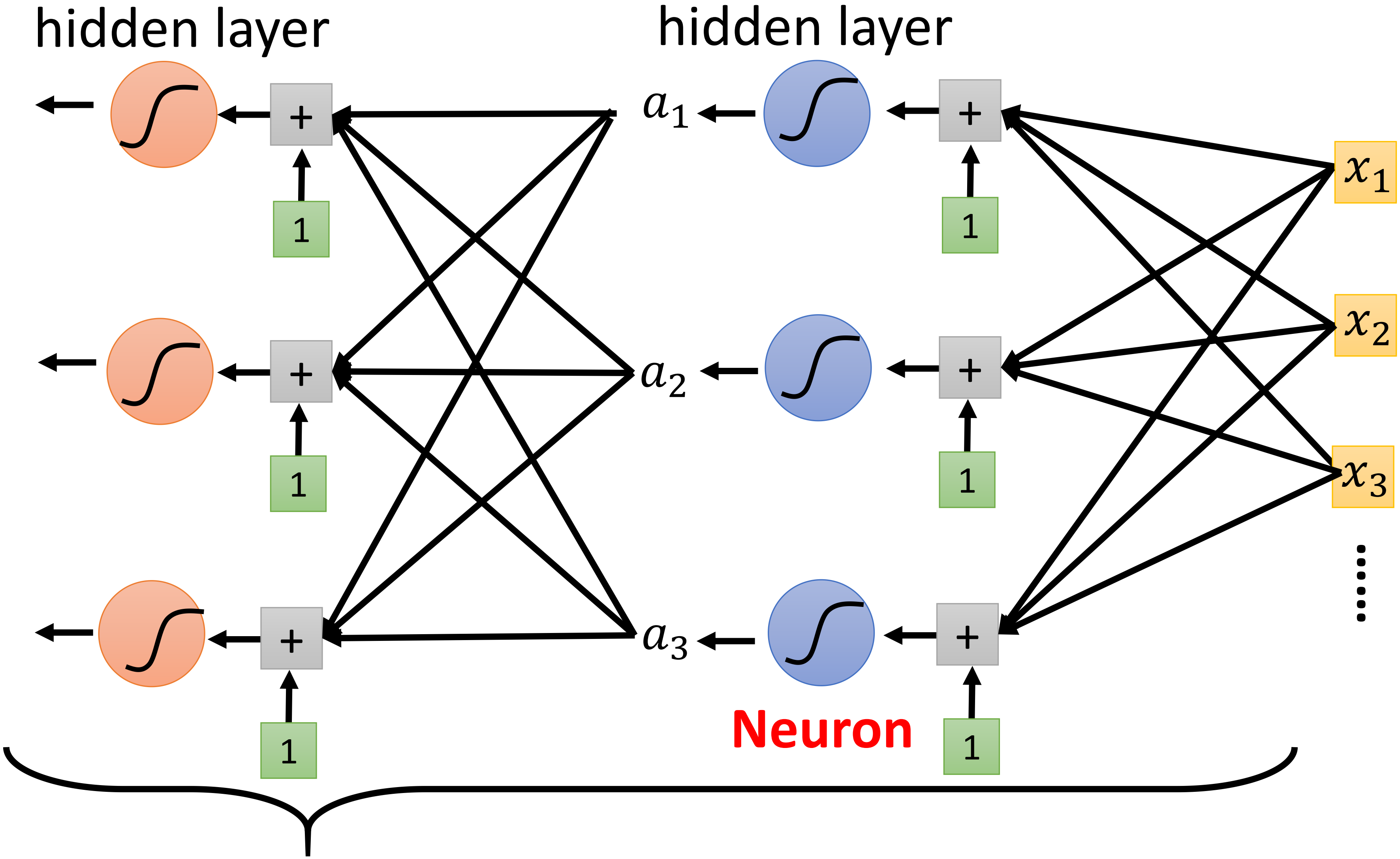
Step 2: define loss from training data

Step 3: optimization

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

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

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



This mimics human brains … (???)

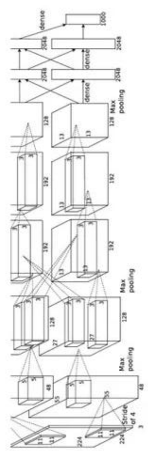
Many layers means **Deep** ➡ Deep Learning

Deep = Many hidden layers

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

8 layers

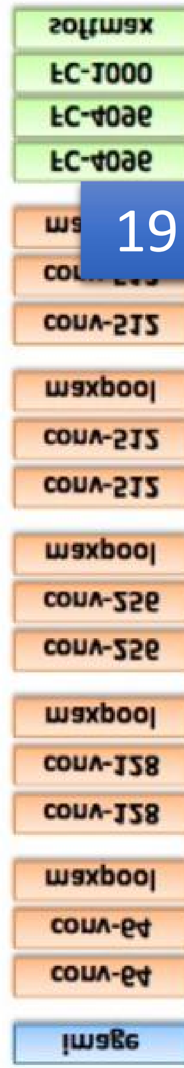
16.4%



AlexNet (2012)

19 layers

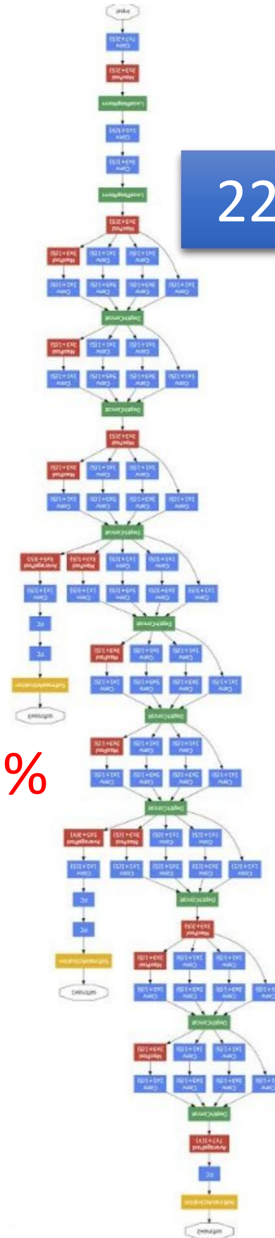
7.3%



VGG (2014)

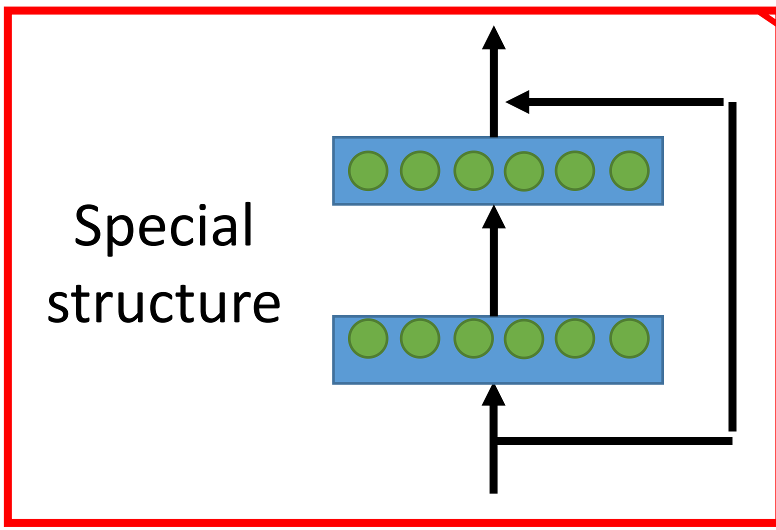
22 layers

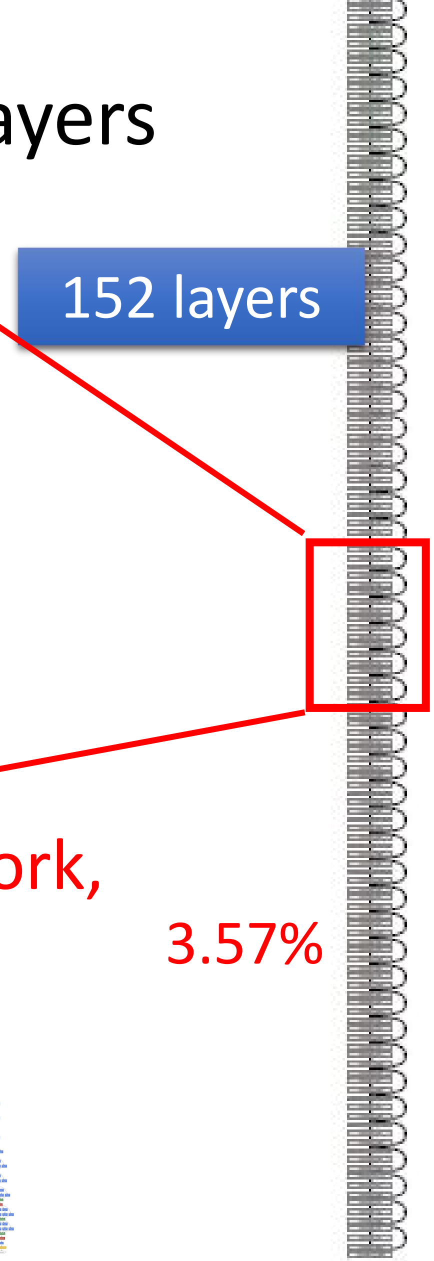
6.7%



GoogleNet (2014)

Deep = Many hidden layers





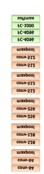
Why we want "***Deep***" network,
not "***Fat***" network?

3.57%

16.4%

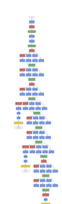
AlexNet
(2012)

7.3%



VGG
(2014)

6.7%



GoogleNet
(2014)

Residual Net
(2015)