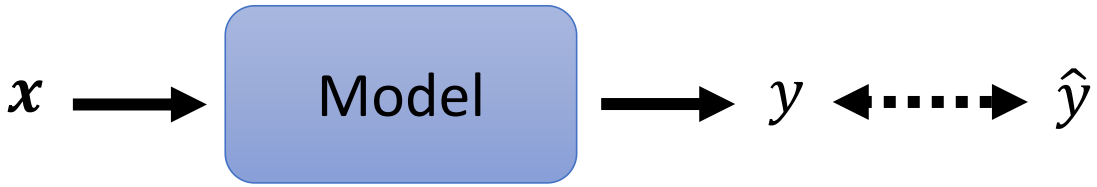
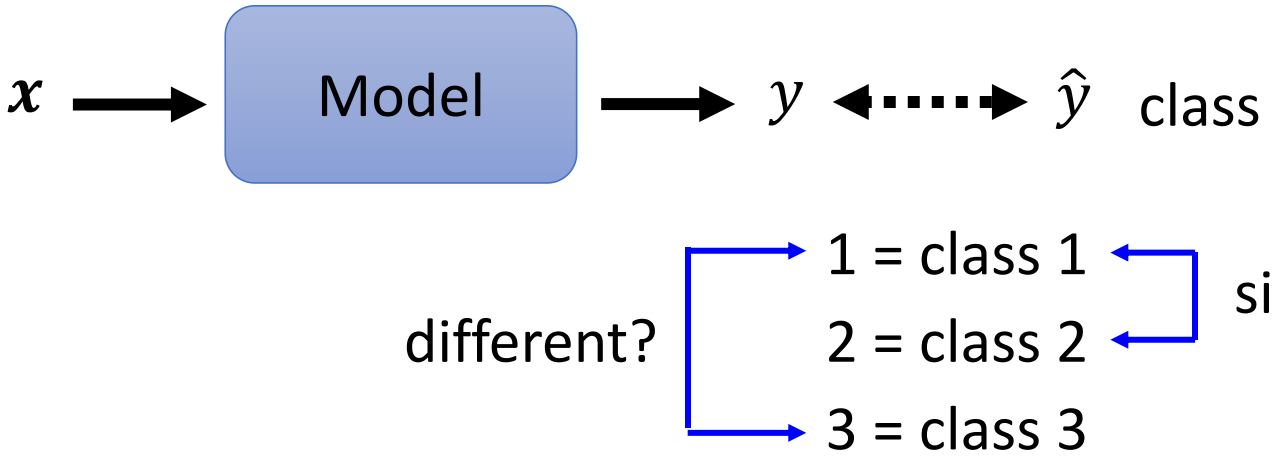
# Deep Learning and Back Propagation

# Classification vs. Regression

- Regression




- Classification as regression?

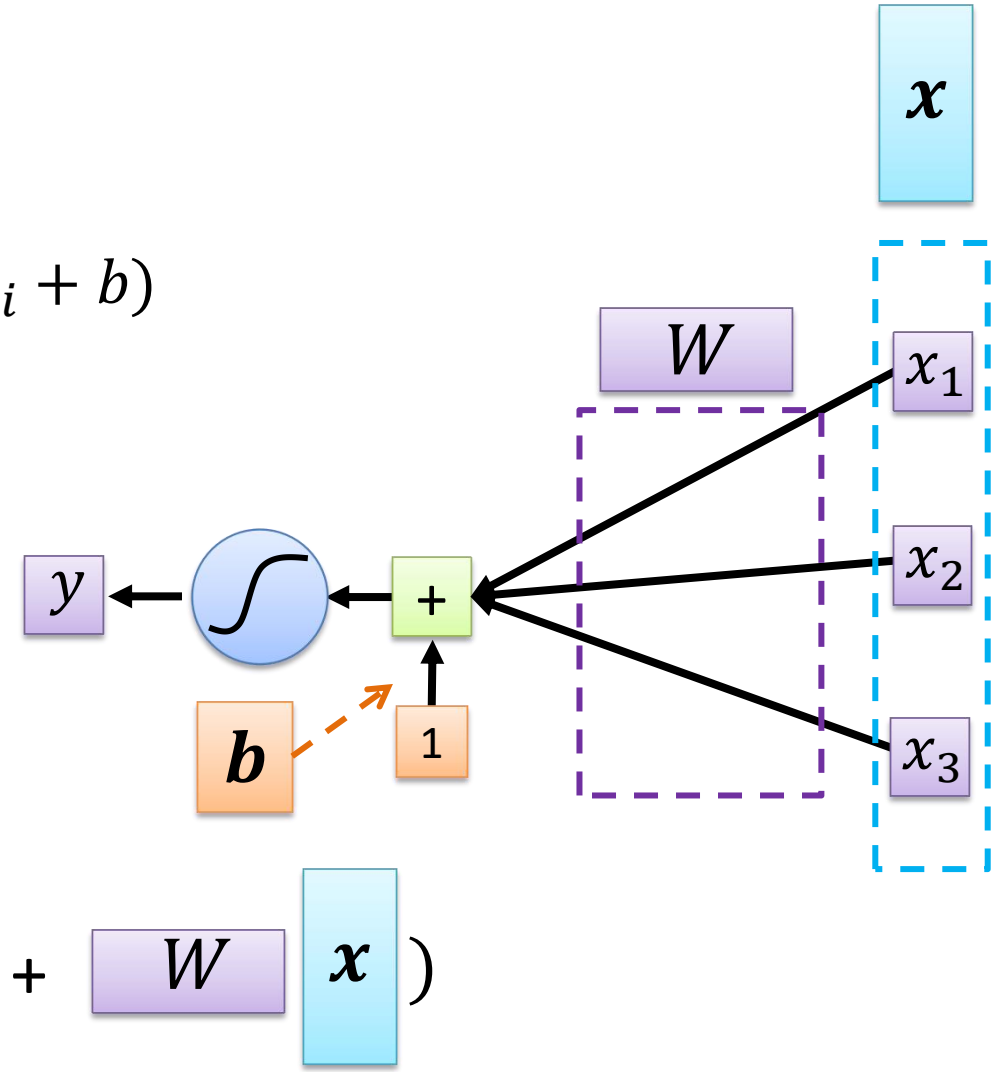


# Logistic Regression

Basic Model

$$y = \delta(\sum_{i=1}^{3} w_i x_i + b)$$

only output
one value

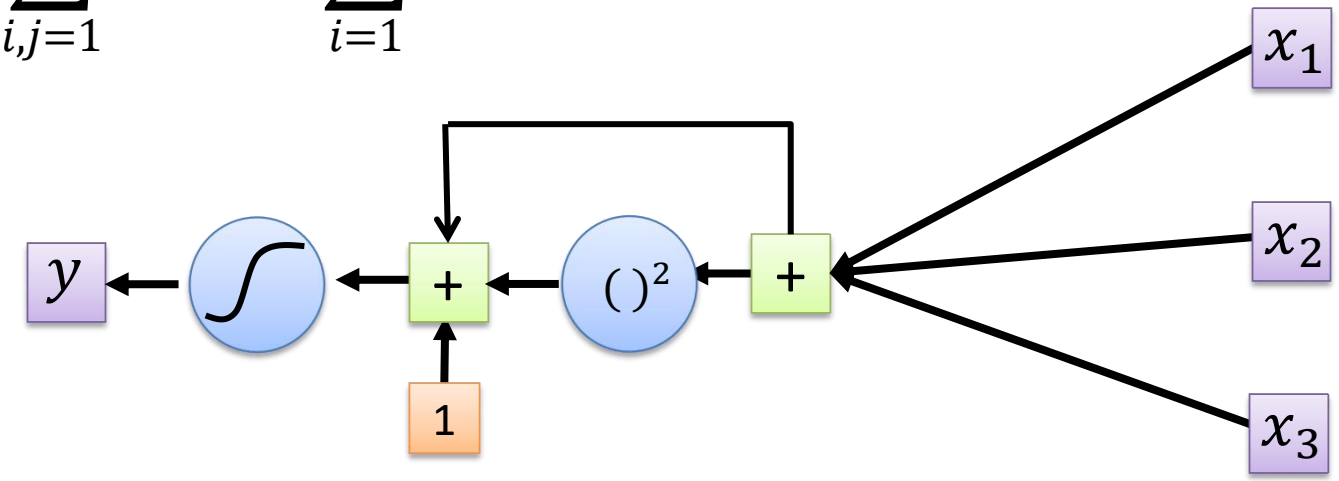



$$y = \sigma(b + W x)$$

# Logistic Regression

How about quadratic model?

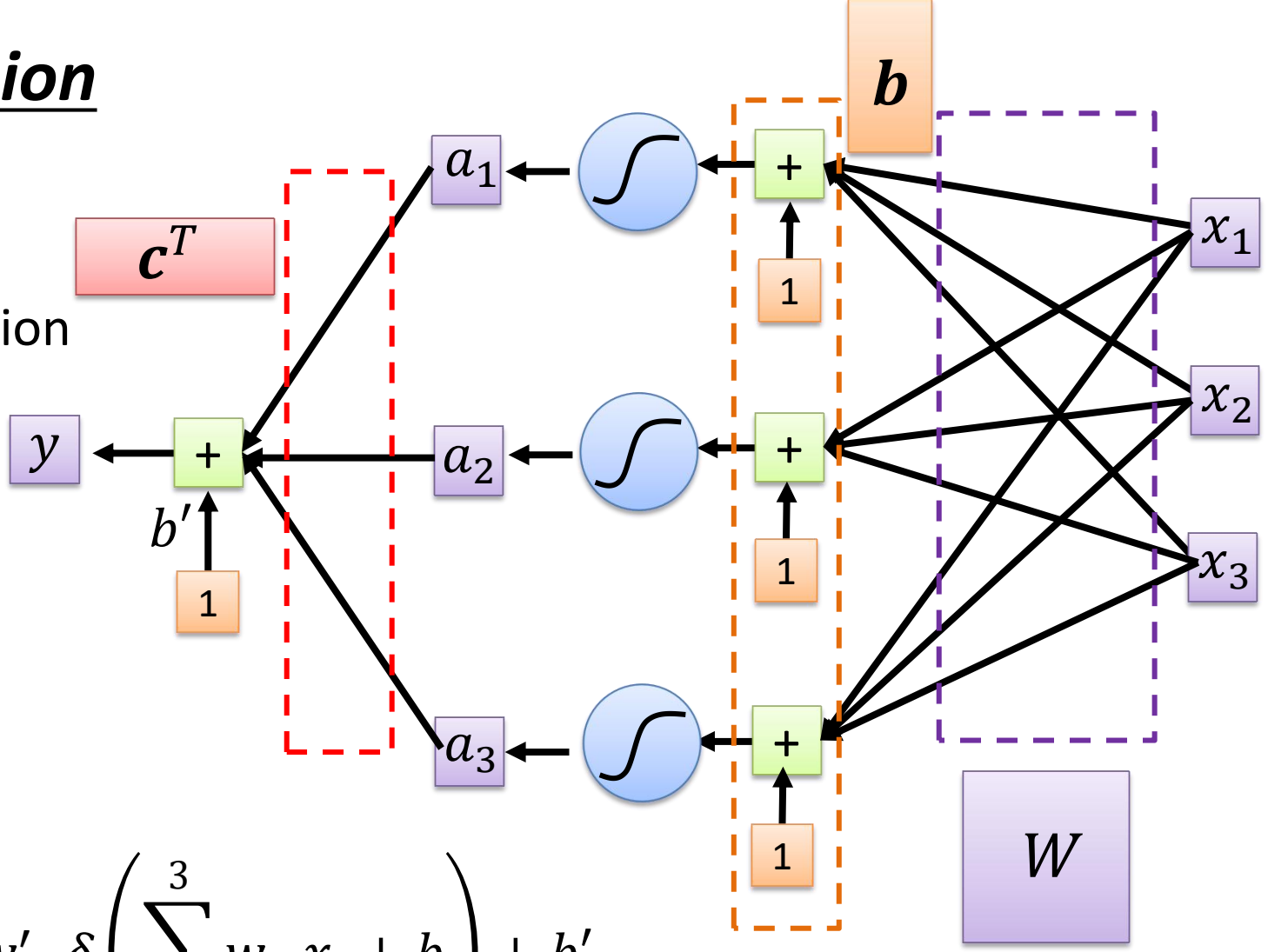
$$y = \delta\left(\sum_{i,j=1}^{3} w_{ij}x_i x_j \sum_{i=1}^{3} w_i x_i + b\right)$$




Modularity: keep the basic module clean and functional

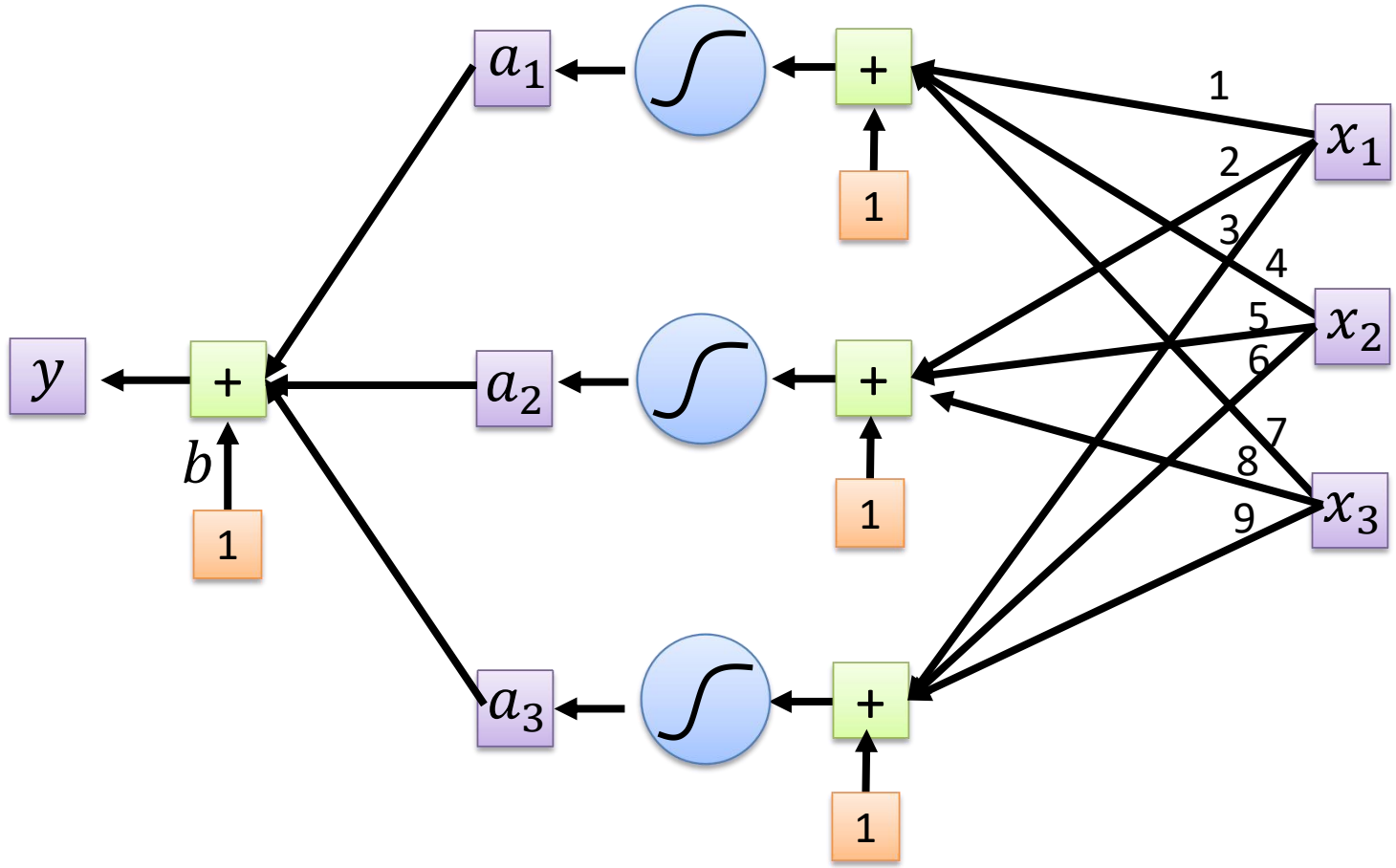
# Regression

Aggregation is Key




$$y = \sum_{j=1}^{3} w'_j \, \delta\left(\sum_{i=1}^{3} w_{ij} x_i + b\right) + b'$$

$$y = b' + c^T \, \sigma(b + W x)$$




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

# 投票 最多可选1项

设置

What is W?

A

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

B

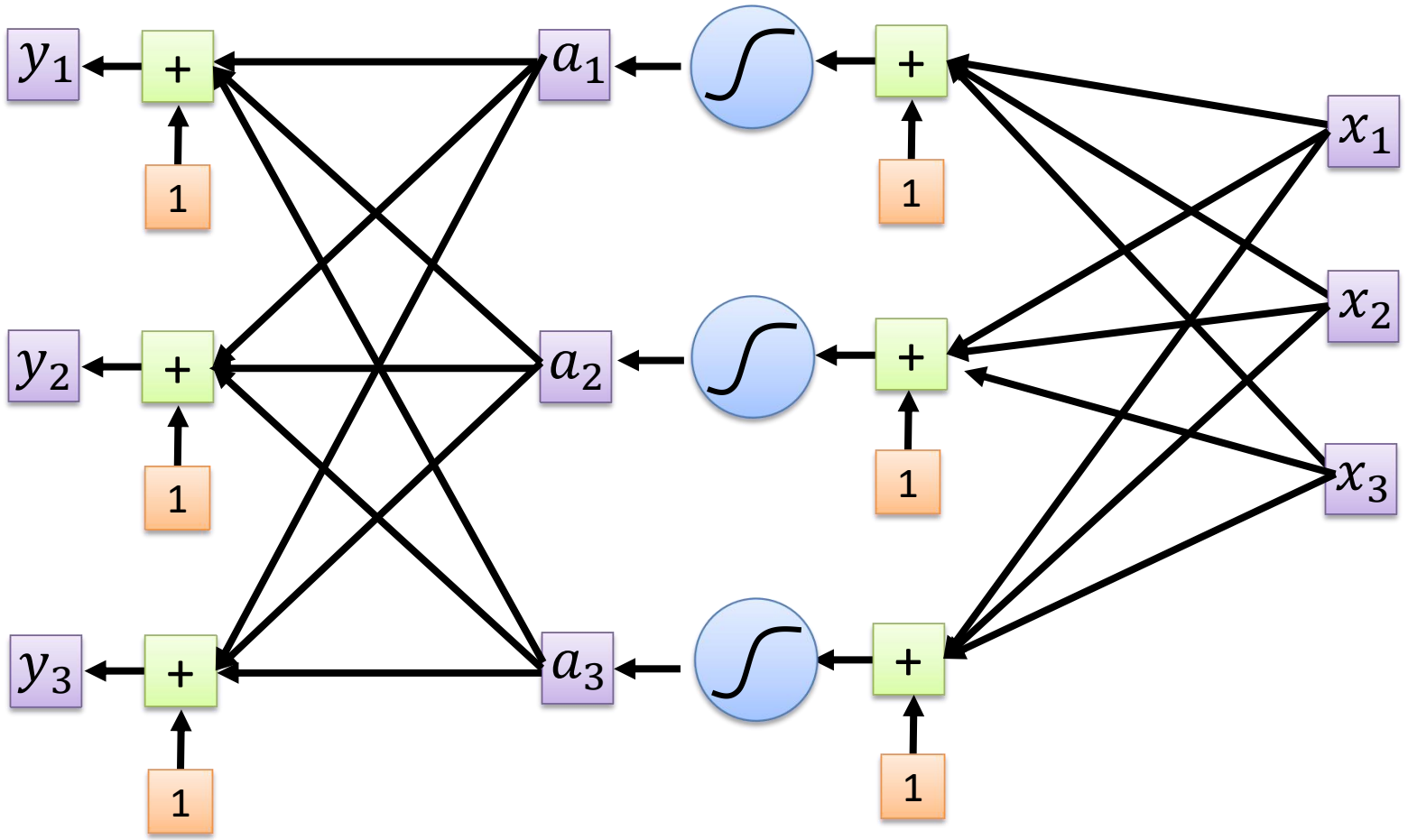
$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

提交

# Class as one-hot vector

Class 1 Class 2 Class 3

$$\hat{y} = \begin{bmatrix}1\\0\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\1\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\0\\1\end{bmatrix}$$



## ***Regression***

label

feature

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

## ***Classification***

feature

$$y = b' + W' \sigma( b + W x )$$

label

$$\hat{y} \longleftrightarrow y' = softmax( y )$$

0 or 1

Make all values between 0 and 1

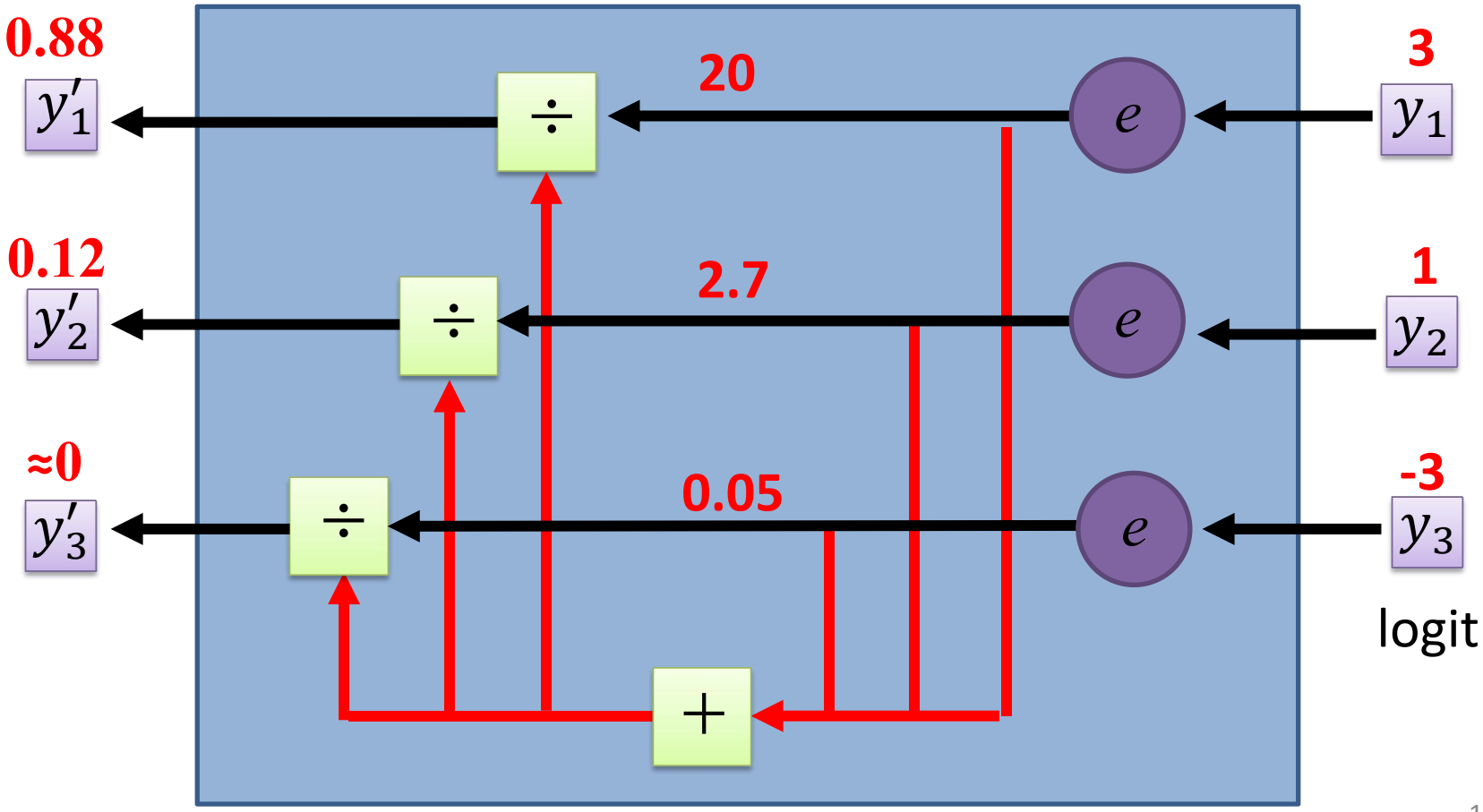
Can have any value

$$y_i' = \frac{exp(y_i)}{\sum_j exp(y_i)}$$

# Soft-max

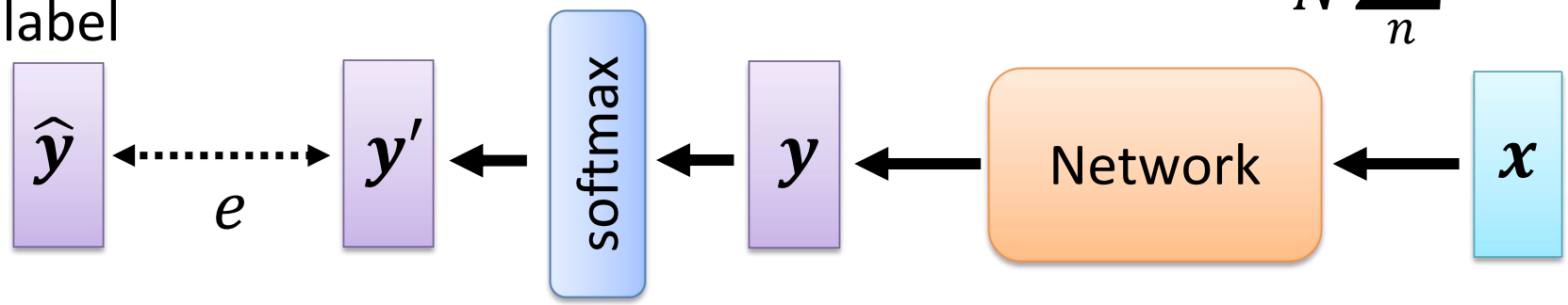
- $1 > y_i' > 0$
- $\sum_i y_i' = 1$

How about **binary classification**? ☺



# Loss of Classification

$$L = \frac{1}{N}\sum_{n} e_n$$




*Regression: Mean Square Error (MSE)*

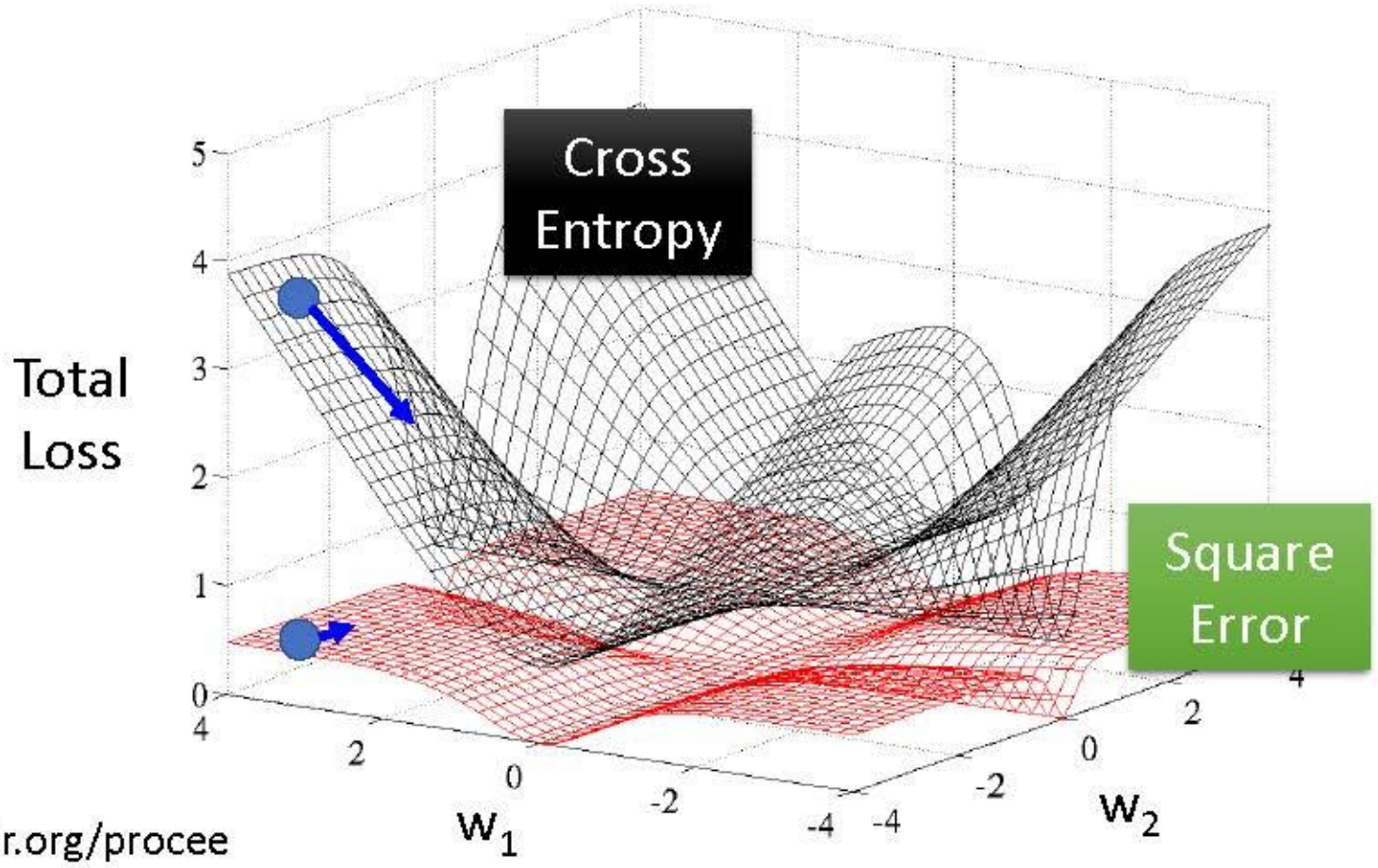
$$e = \sum_{i} (\hat{y}_i - y'_i)^2$$

*Multiclass Classification Cross-entropy*

$$e = -\sum_{i} \hat{y}_i ln y'_i$$

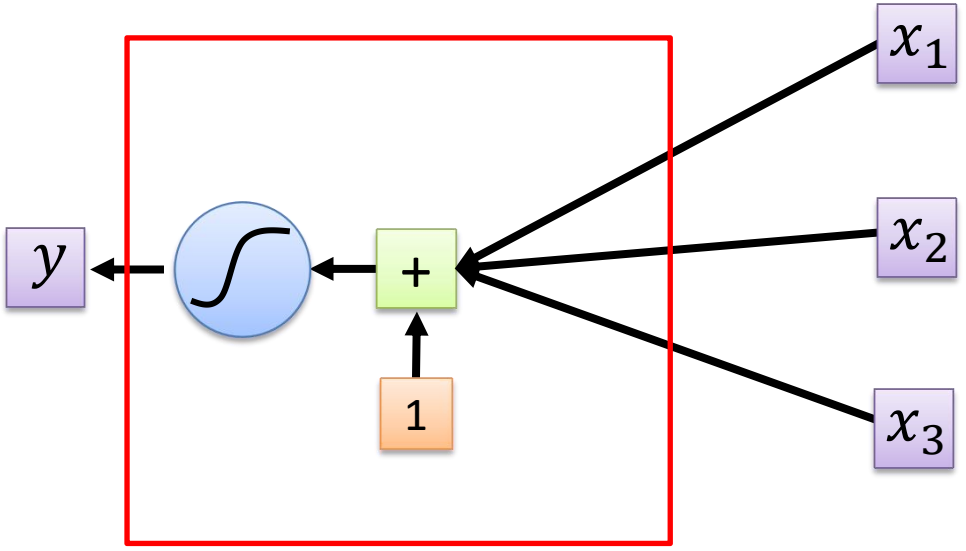
**Minimizing cross-entropy** is equivalent to **maximizing likelihood**.

# Cross Entropy v.s. Square Error

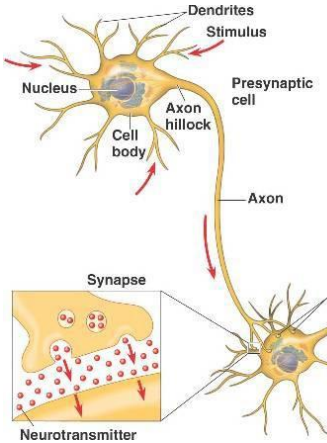


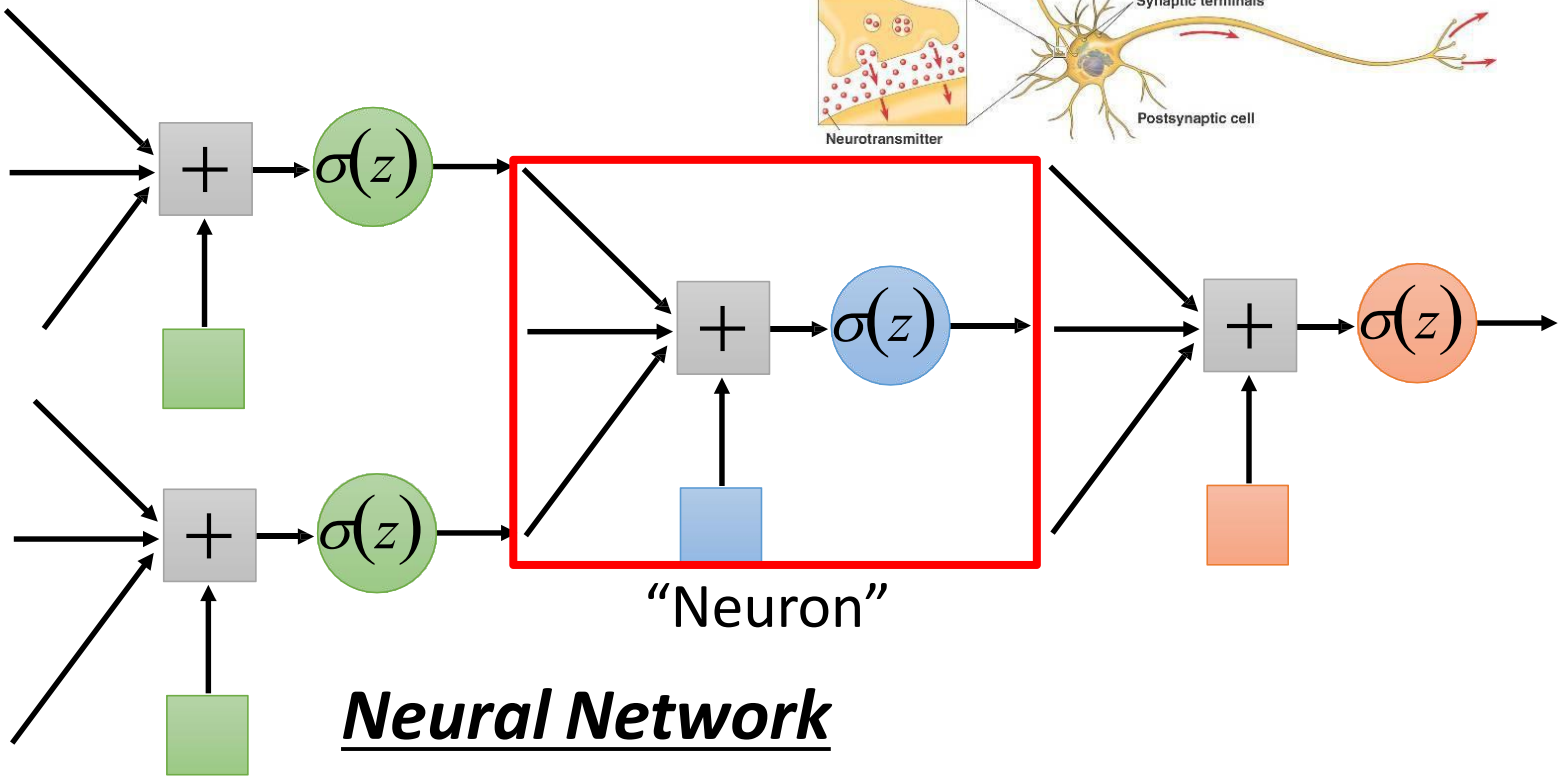

http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

# Basic unit in Neural Network



# Neural Network




"Neuron"

***Neural Network***

Different connection leads to different network structures

Network parameter $\theta$: all the weights and biases in the "neurons"

# 投票 最多可选1项

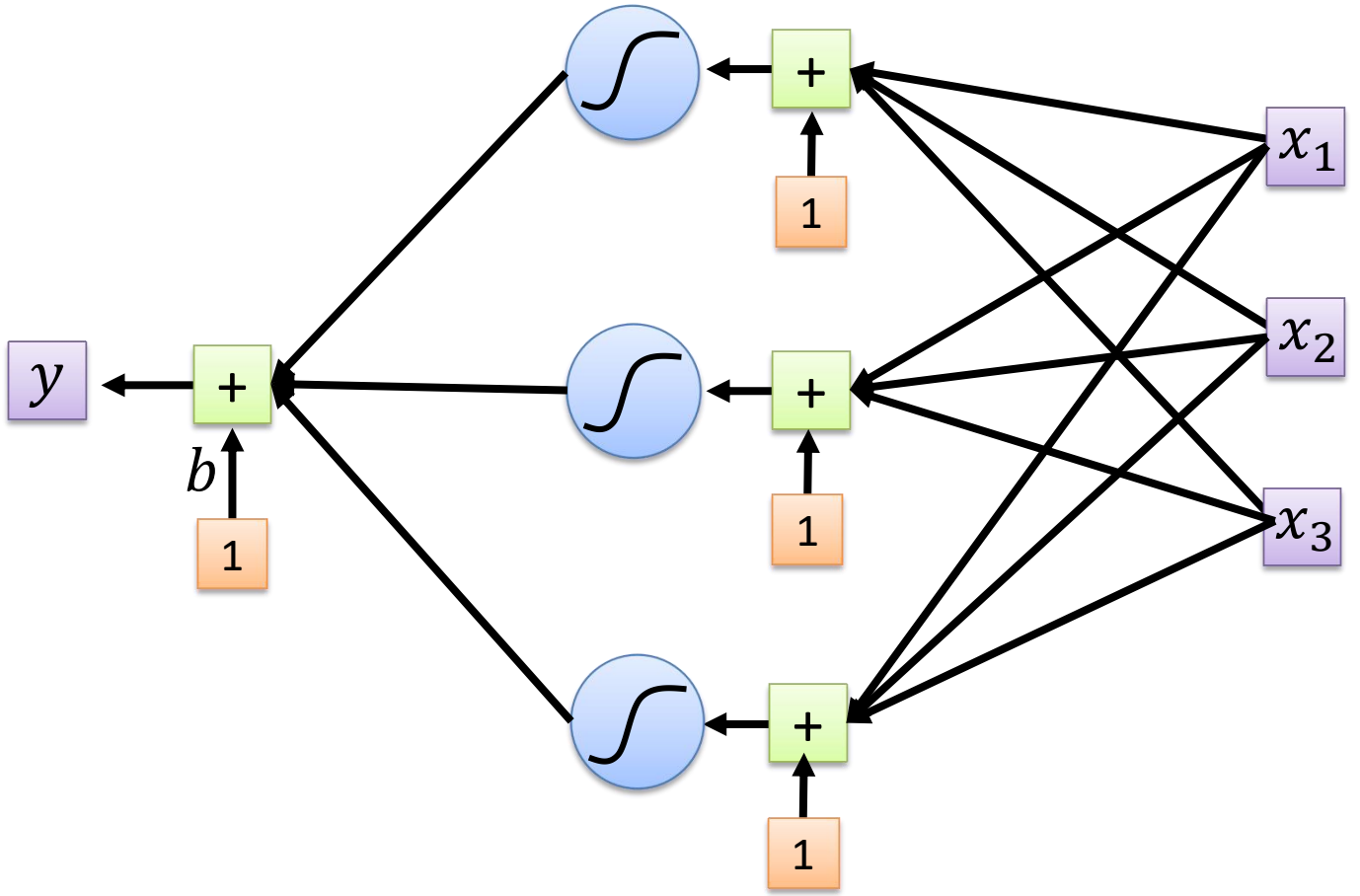
设置

Neuron is a linear function plus non-linear transformation. Is non-linear transformation necessary?

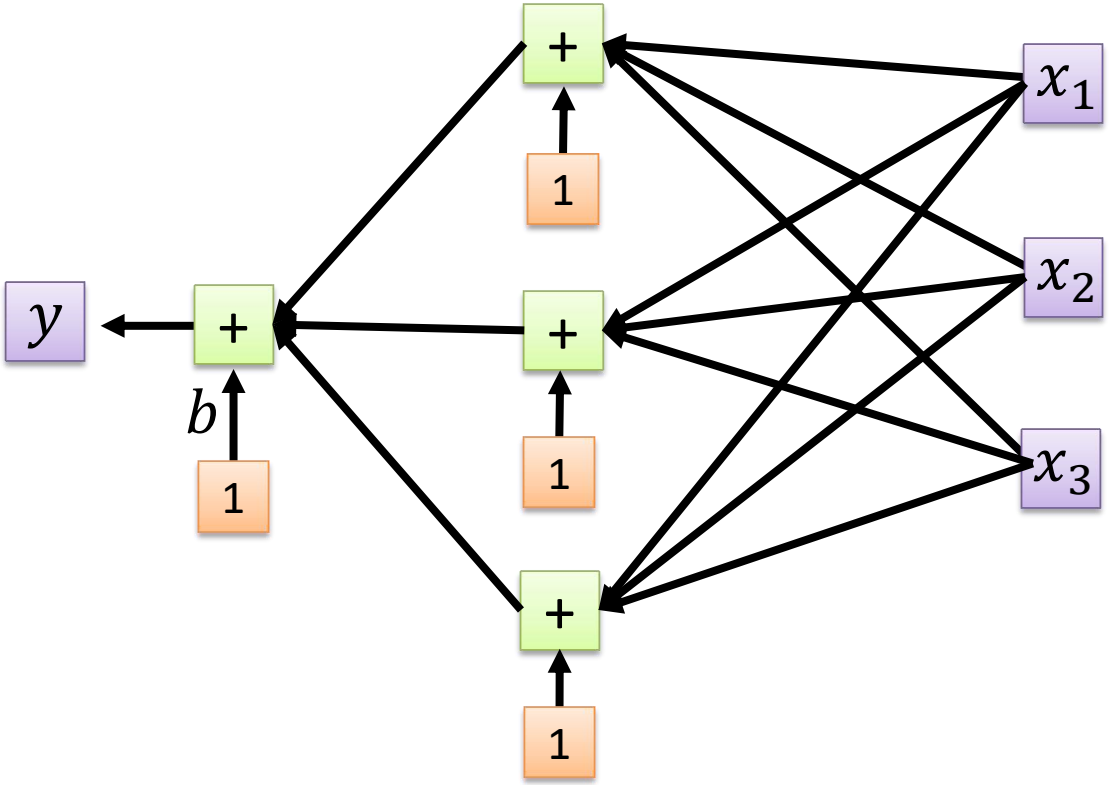
A necessary

B Not necessary

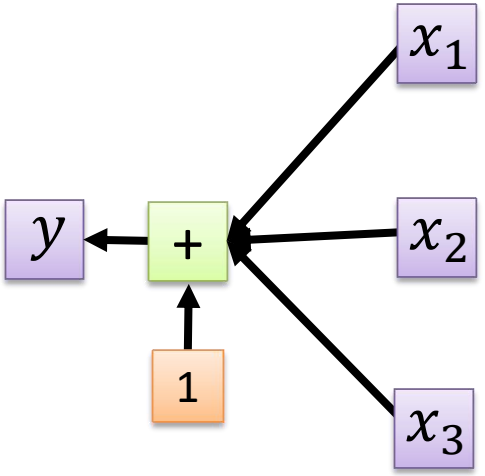
提交




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



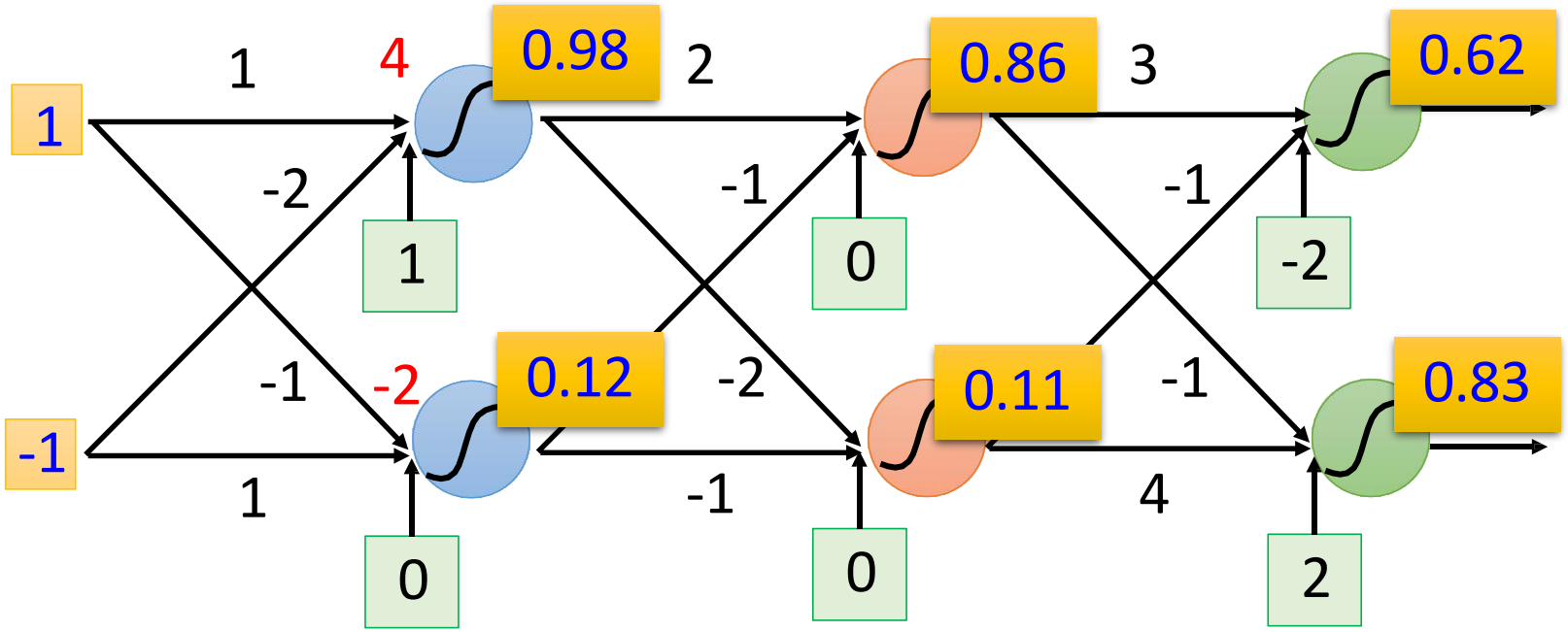

=



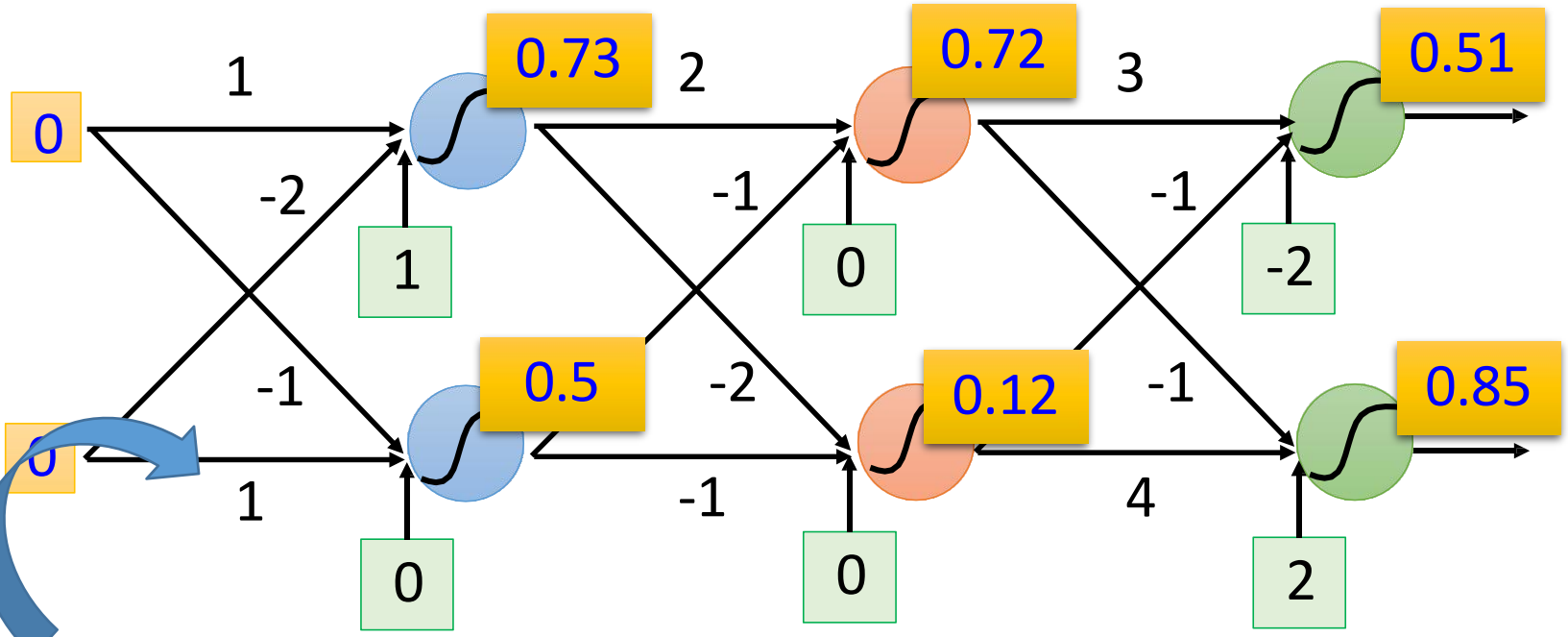

$$y = b + c^T (b + W x)$$

$$y = b + c^T b + c^T W x = b' + W' x$$

# Fully Connect Feedforward Network
# 全连接前馈网络



# Fully Connect Feedforward Network



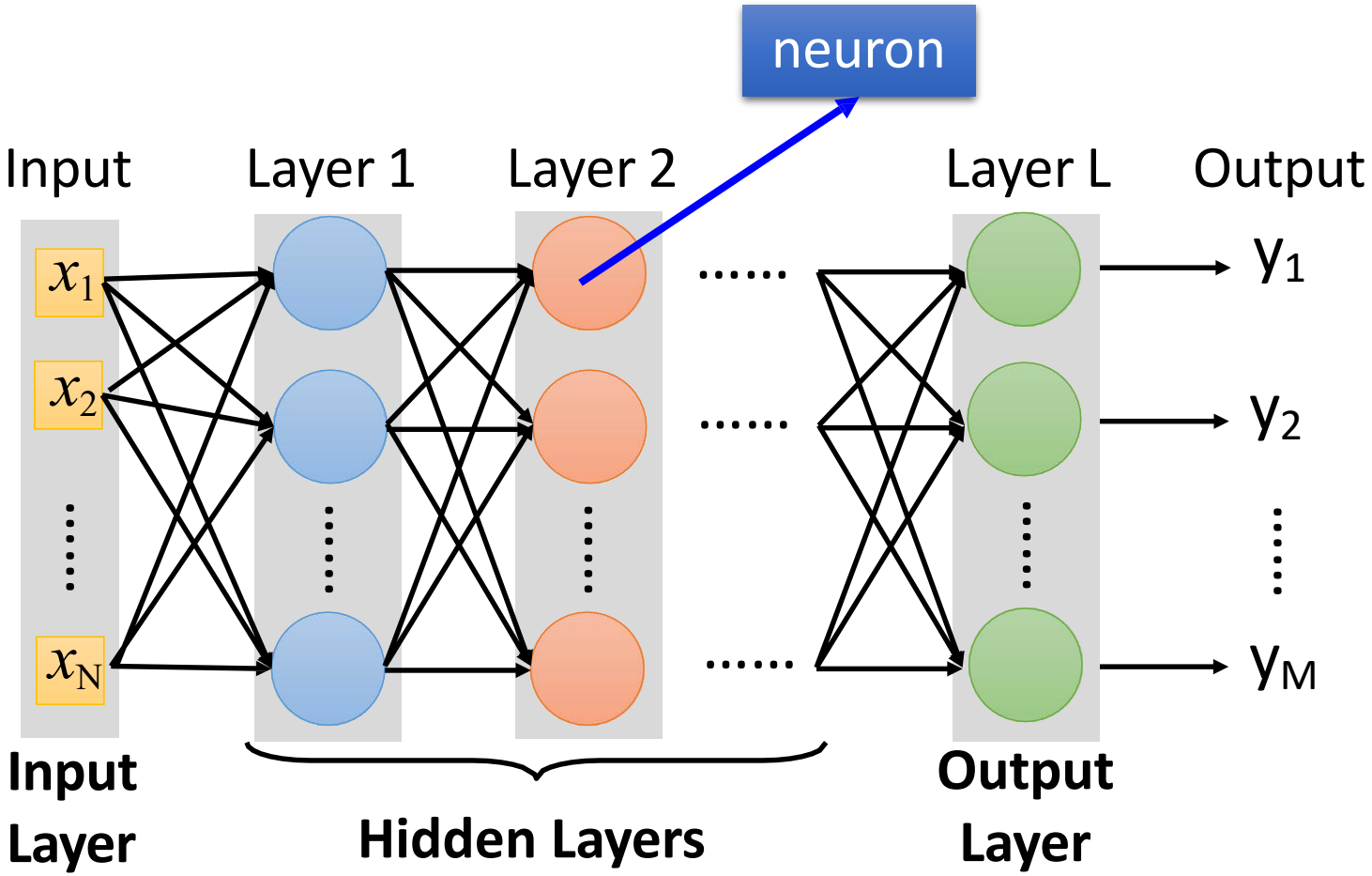

This is a function.

Input vector, output vector

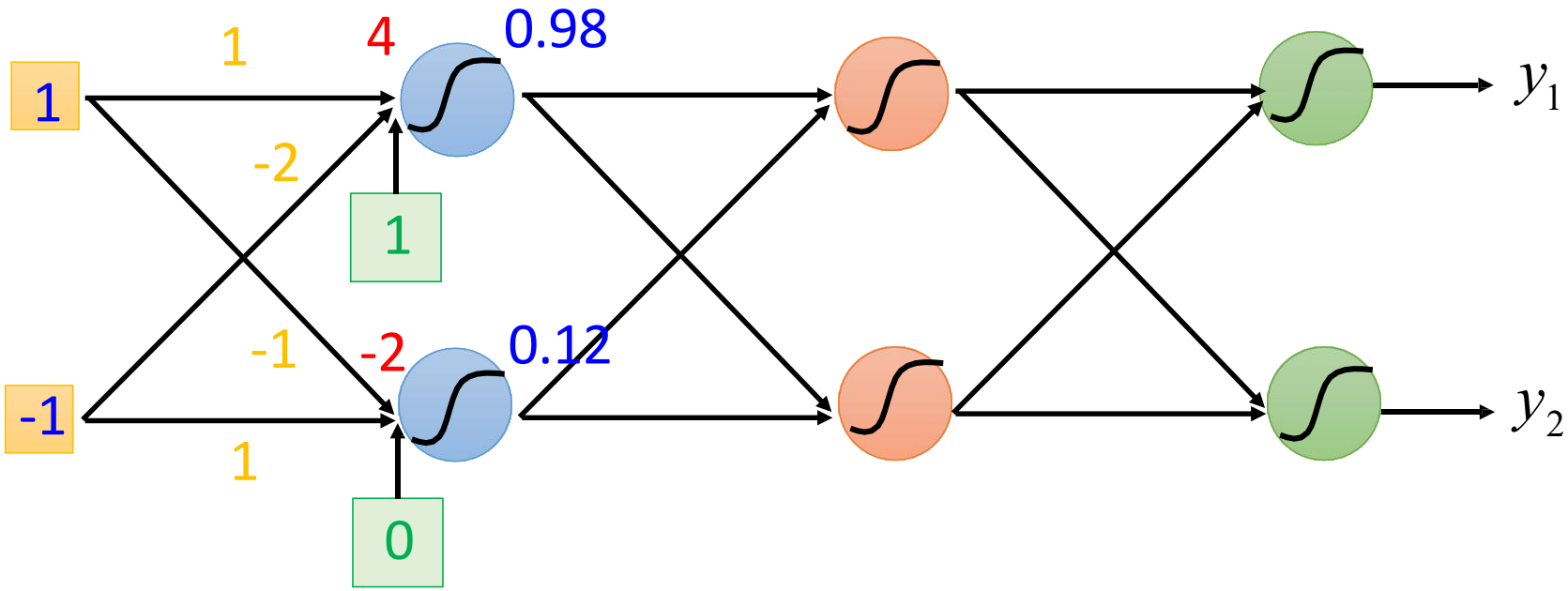
$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

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

# Fully Connect Feedforward Network



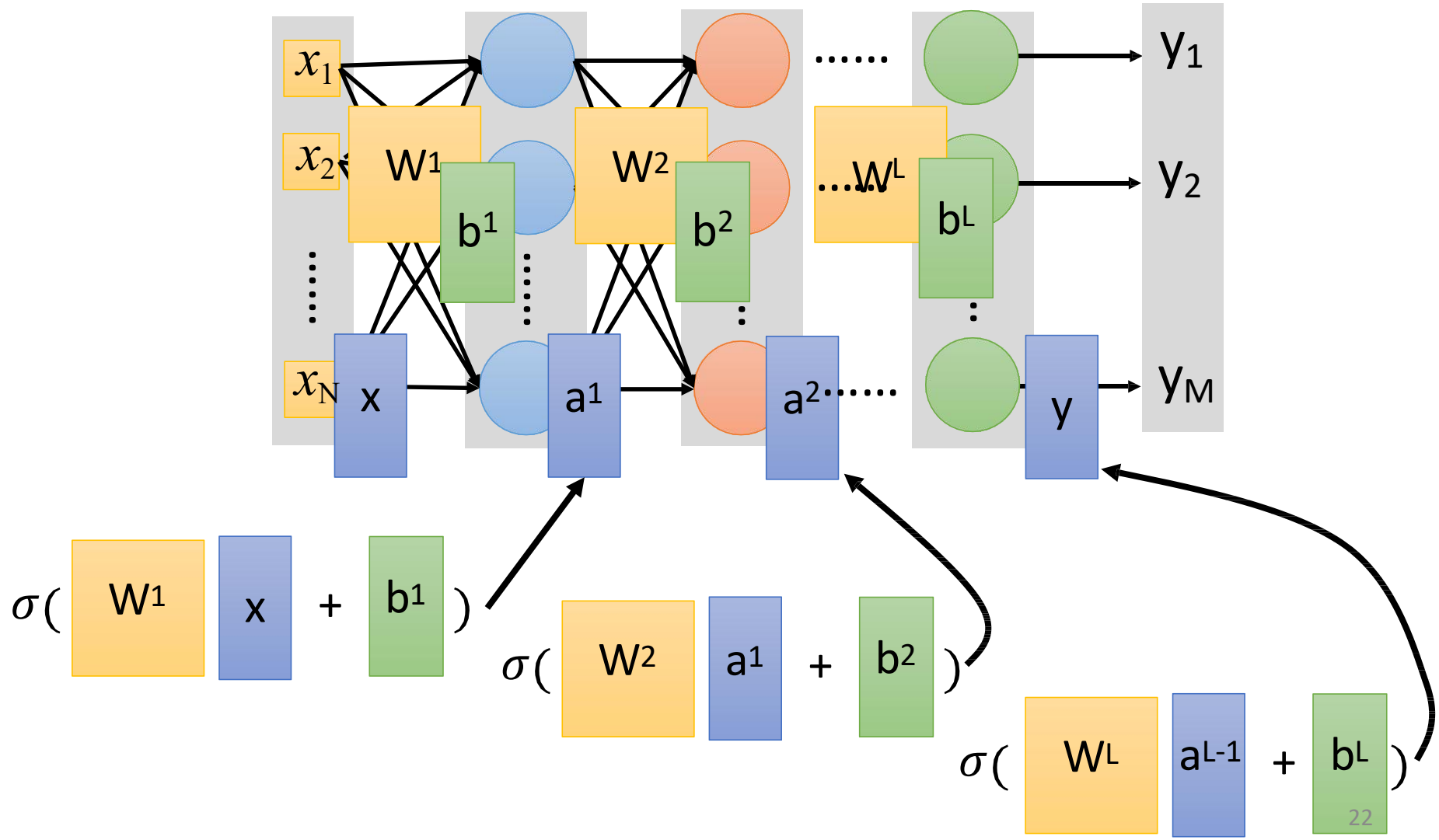
# Matrix Operation



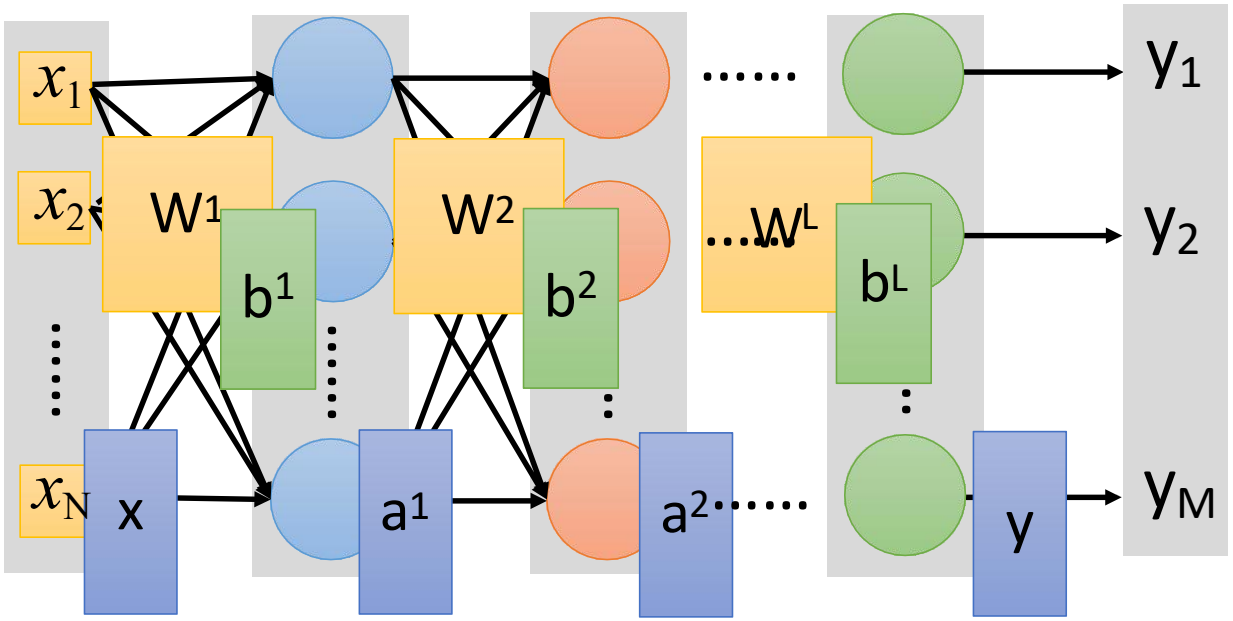

$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

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

# Neural Network



# Neural Network

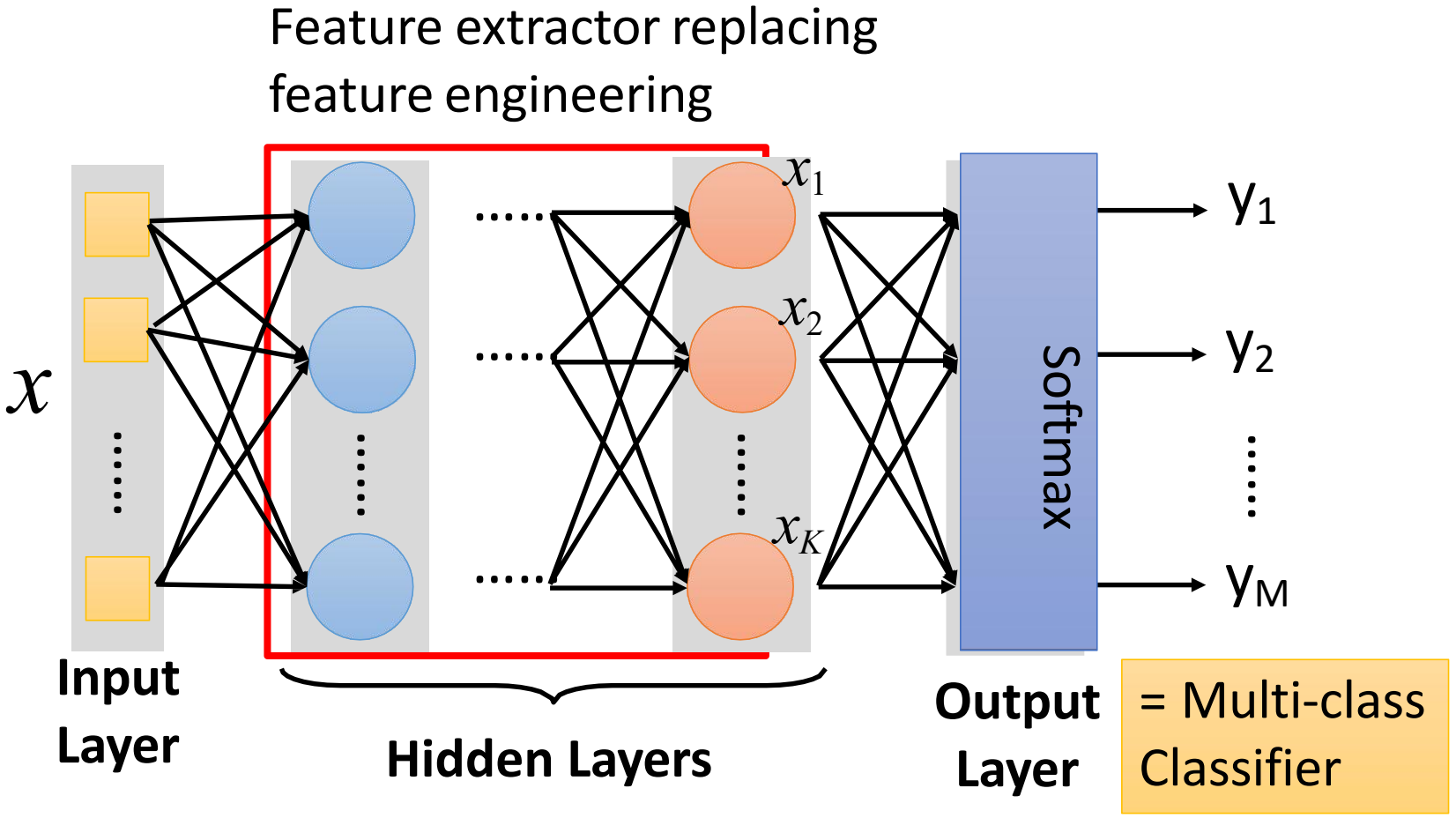



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

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

# Output Layer



# 填空题 1分

设置

Given a 3-layer fully connect feedforward network, input layer 10 nodes, layer1(hidden layer) 20 neurons, layer2(hidden layer) 5 neuron, layer3(output layer) 3 neurons, how many parameters(weight and bias) in this network?

Answer: [填空1]

正常使用填空题需3.0以上版本雨课堂

作答

# 巨量AI模型

- GPT-3 英文语言模型， 1750亿参数
- 源1.0, 浪潮，全球最大规模AI巨量模型，2457亿参数（2021.9.29）
  - 训练集，5年整个中文互联网全部数据，文本分类上5TB中文数据集，2000亿词

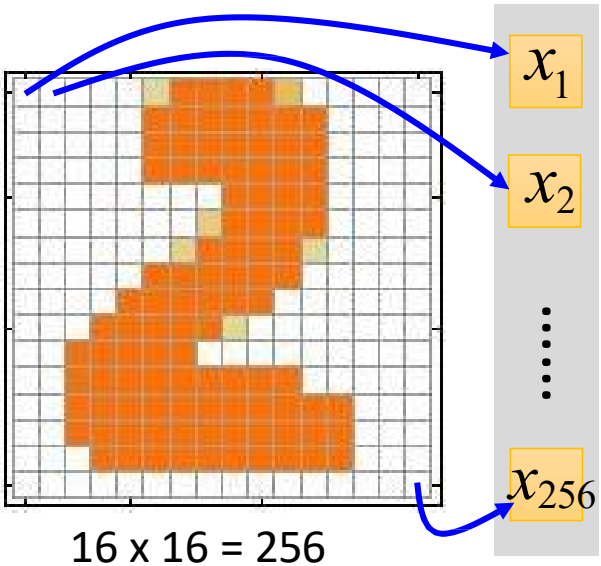
| 对话 | A | B |
| --- | --- | --- |
| 问题1 | 姐姐啥时候回来捏？<br>过年前几天吧。 | 姐姐啥时候回来捏？<br>不知道，你要是想姐姐，就给姐姐写信。 |
| 问题2 | 和喜欢的男生告白了他现在还没回我<br>和喜欢的女生表白了她说我们只适合做朋友 | 和喜欢的男生告白了他现在还没回我<br>我想知道结局 |
| 问题3 | 我买了那套章蜜糖的针织裙<br>哪套？有链接不？ | 我买了那套章蜜糖的针织裙<br>穿了吗？好看不好看？ |
| 问题4 | 不吃东西真的好难啊<br>觉得今天如果真的吃的话，明天铁定会变胖的 | 不吃东西真的好难啊<br>我也觉得！我一定要瘦 |

# 巨量AI模型

- T5
  - Google, 2019.10
  - 110亿
- Megatron-Turing（威震天-图灵）
  - Microsoft and NVIDA, 2021.10.12
  - 5300亿
- M6
  - 阿里达摩研究院，2021.3
  - 10万亿

# Example Application

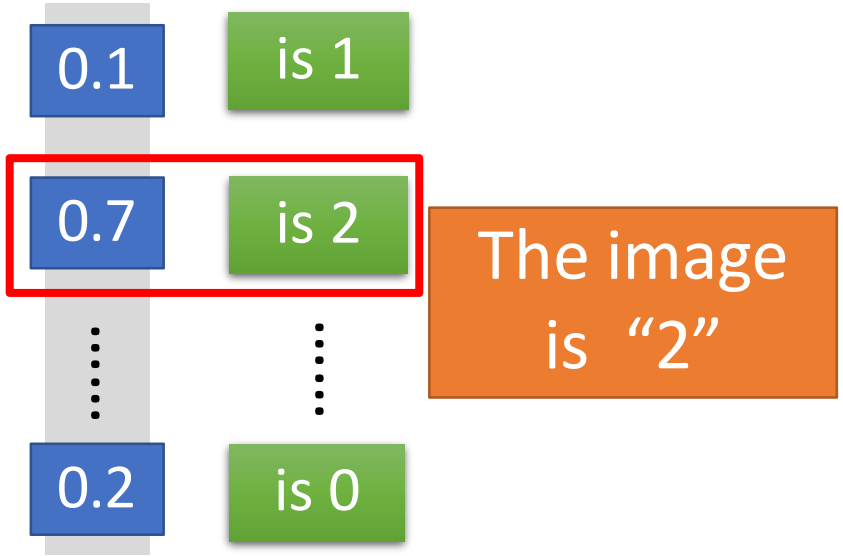

## Input




Ink → 1

No ink → 0

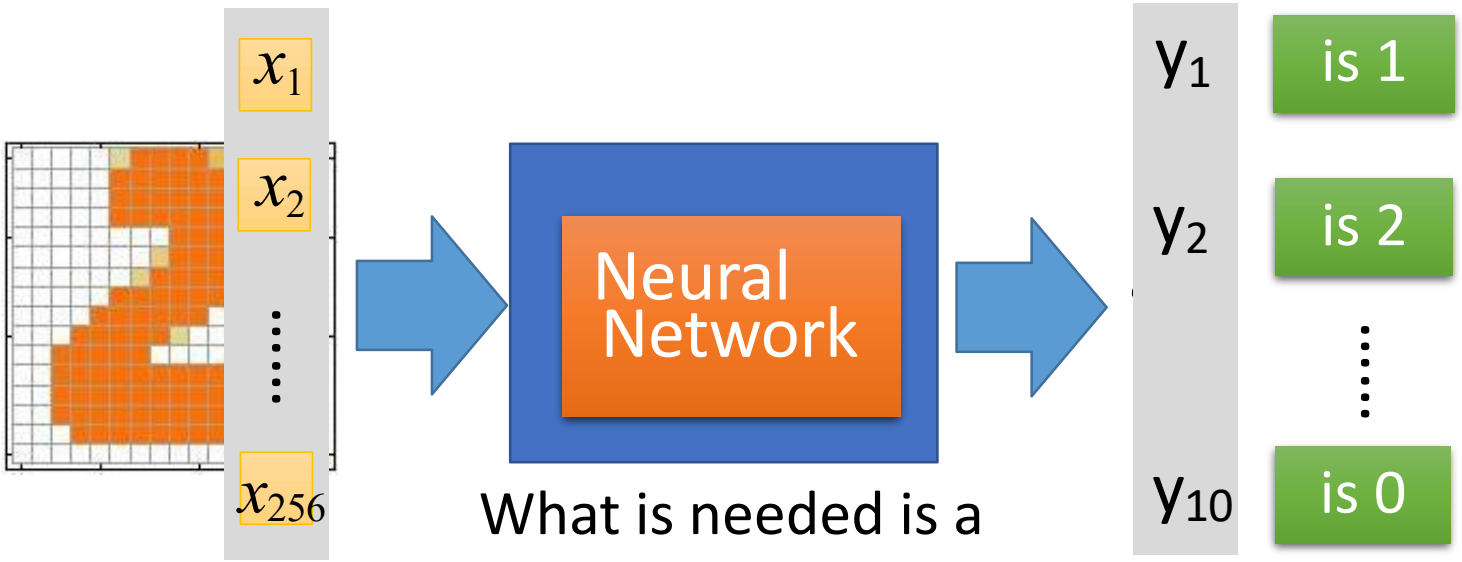
## Output




Each dimension represents the confidence of a digit.

# Example Application

- Handwriting Digit Recognition

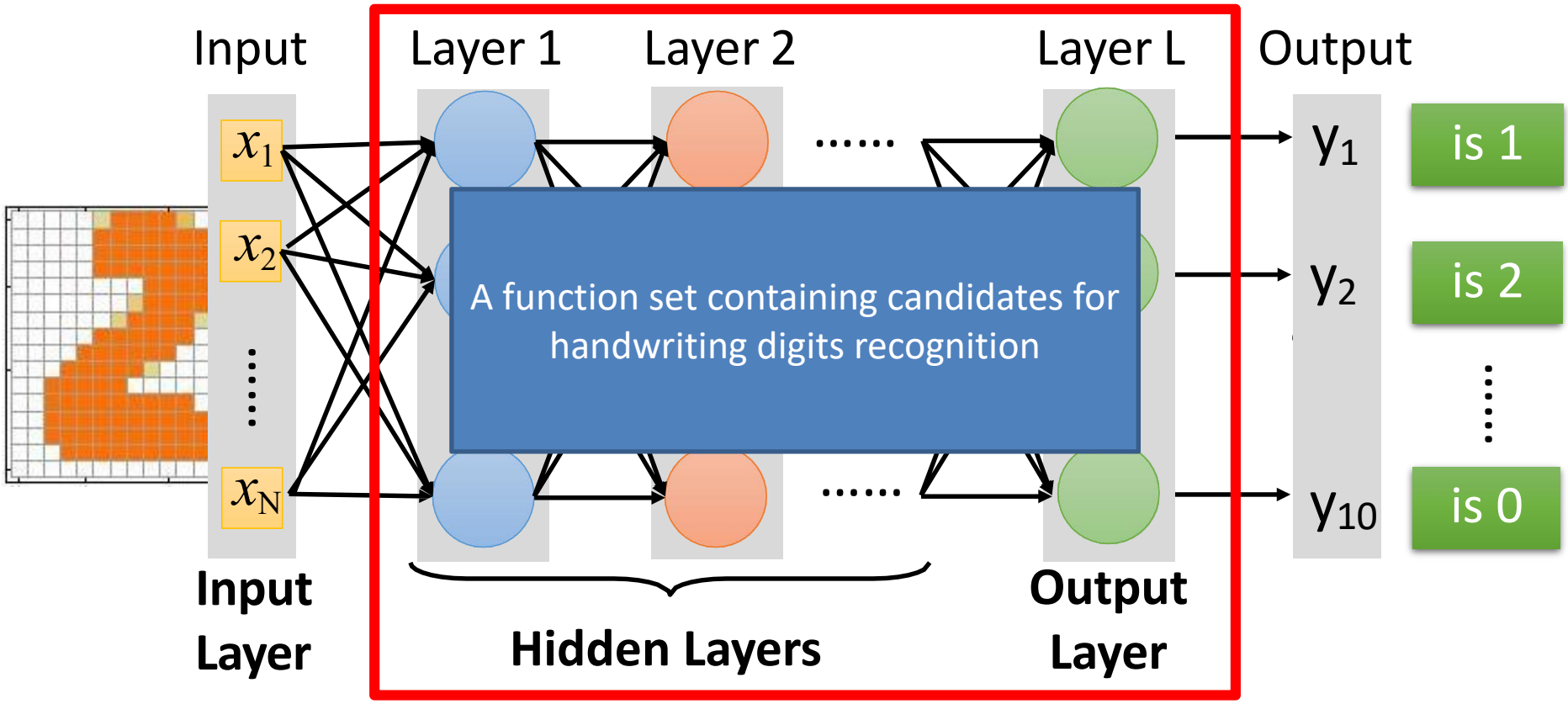



What is needed is a function ……

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.

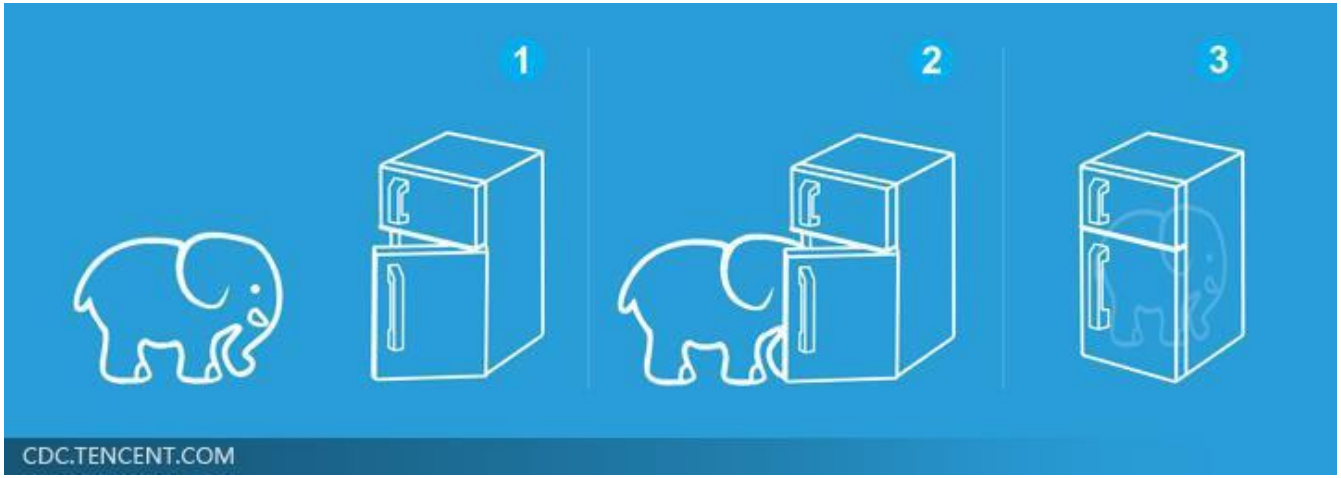
# Three Steps for Deep Learning

Step 1: Neural Network

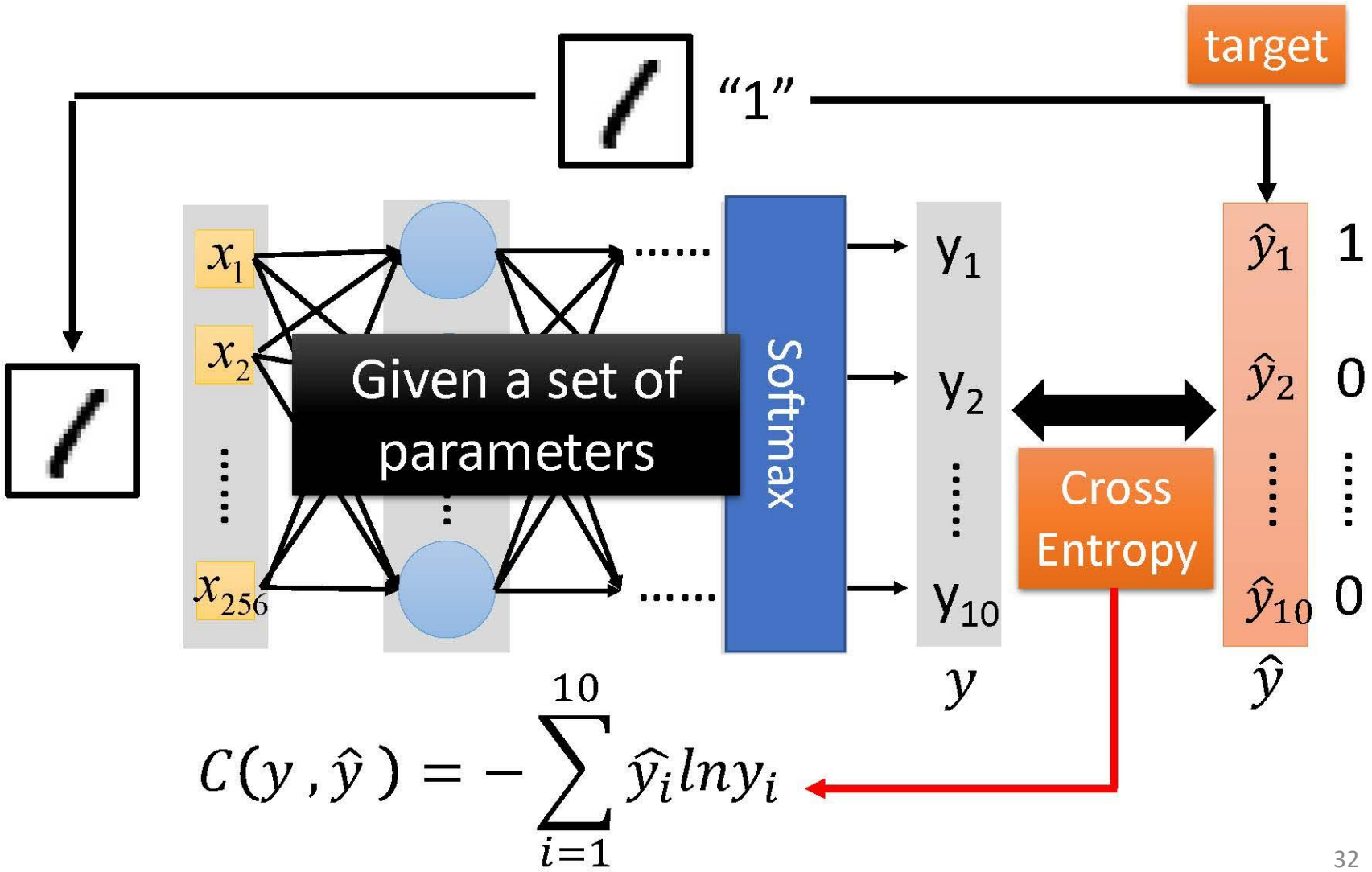
Step 2: goodness of function

Step 3: pick the best function

Deep Learning is so simple ……

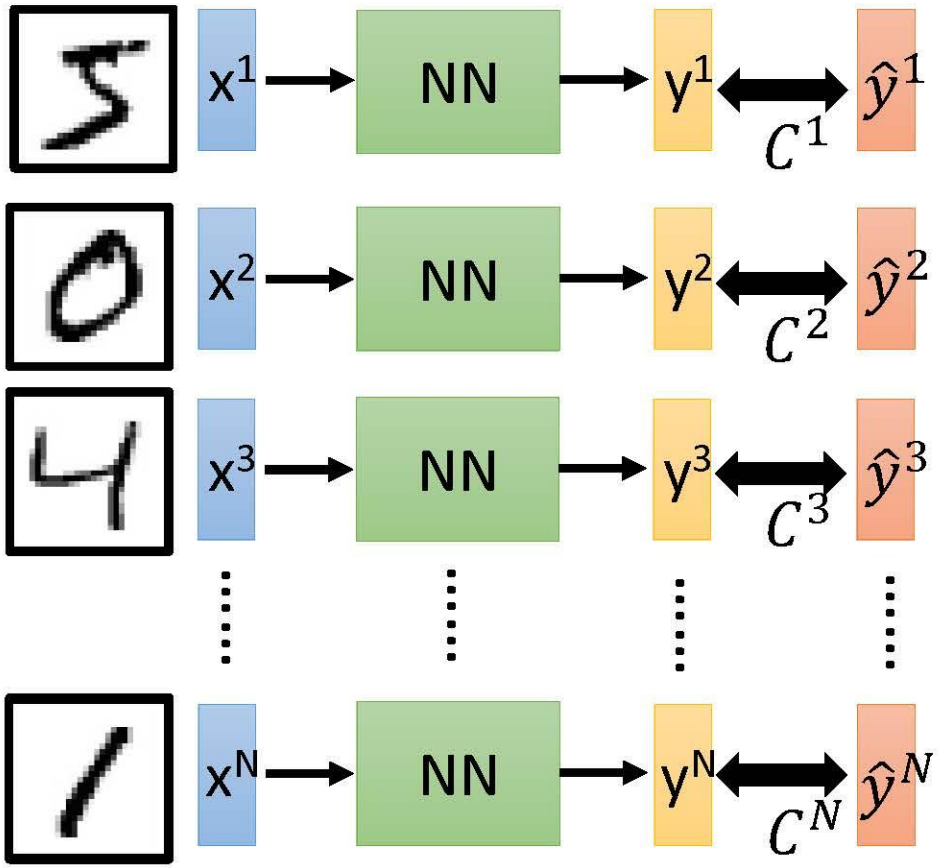


# Loss for an Example



# Total Loss

For all training data …




Total Loss:

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

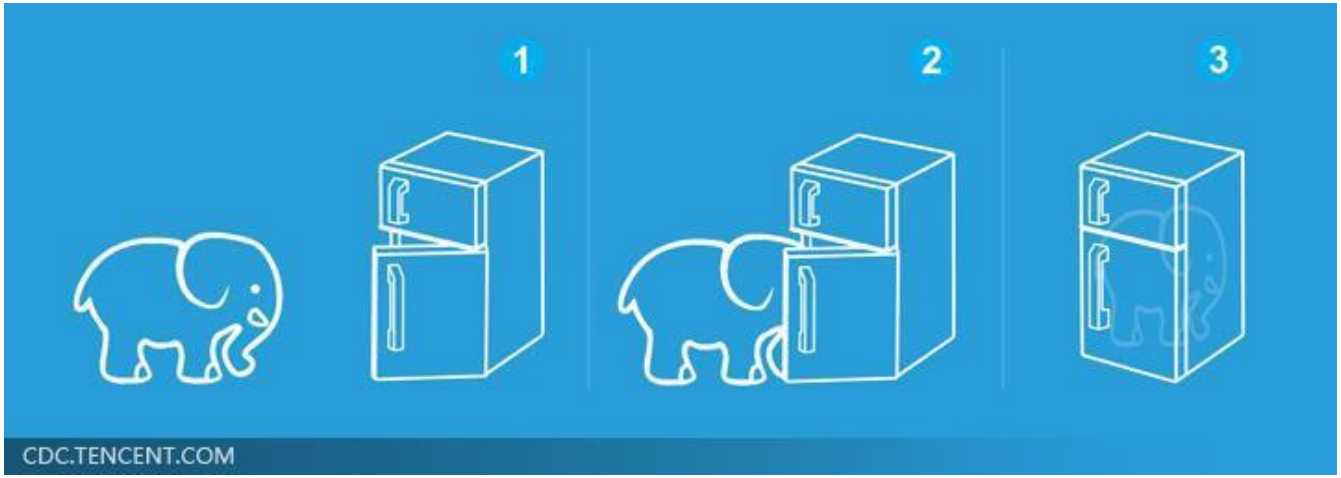
Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters*** $\theta^*$ that minimize total loss L

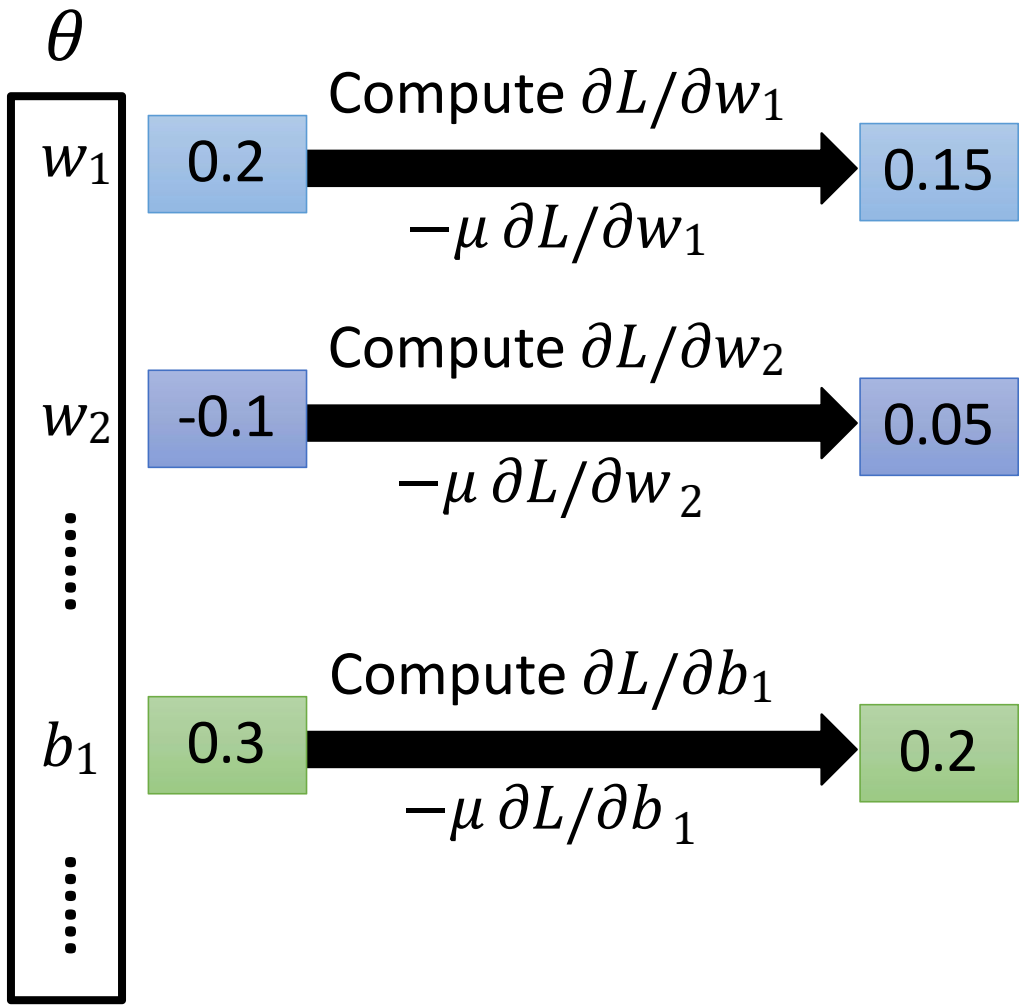
# Three Steps for Deep Learning

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

Deep Learning is so simple ……



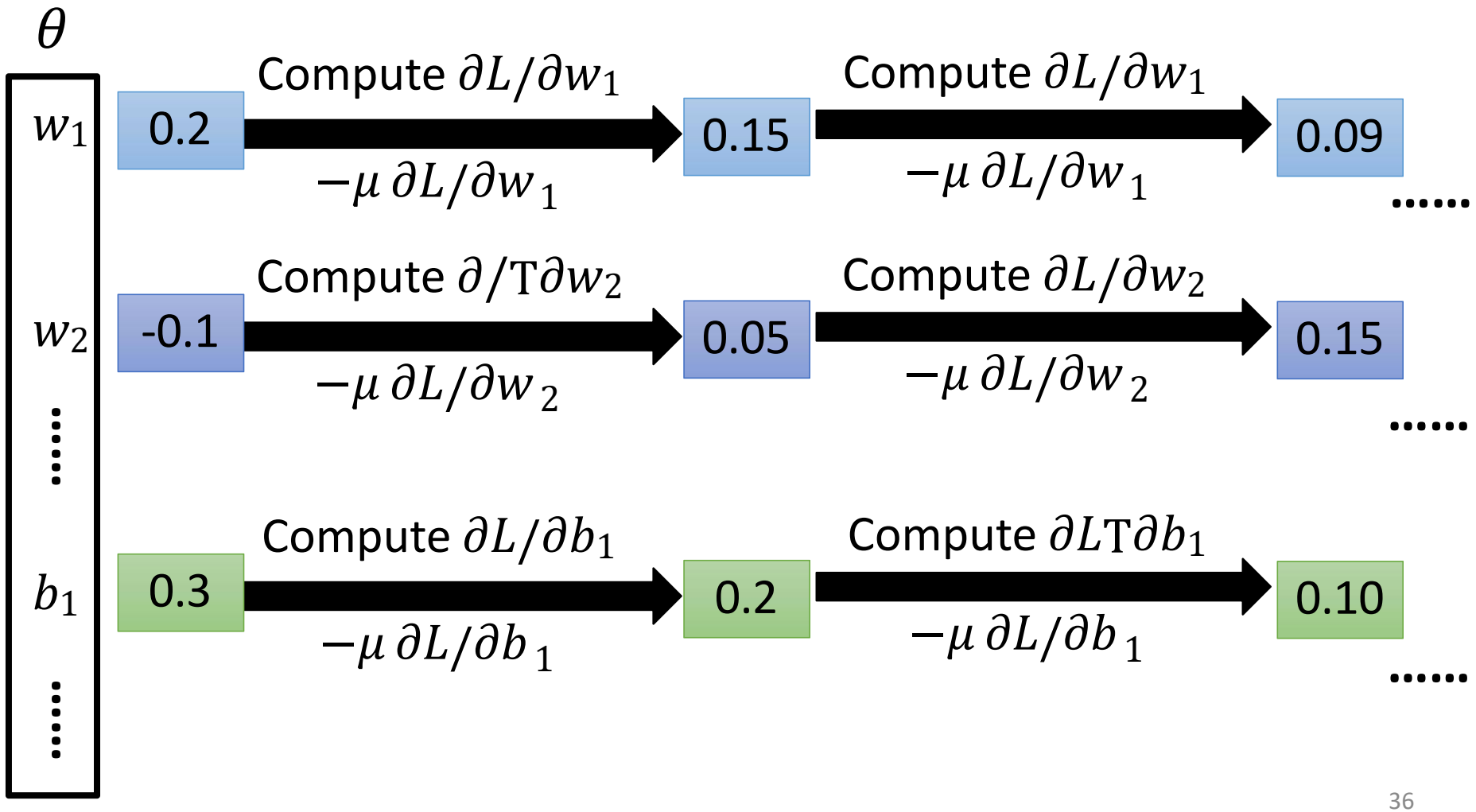
# Gradient Descent




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

gradient

# Gradient Descent



# Backpropagation

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









# Backpropagation
# 反向传播

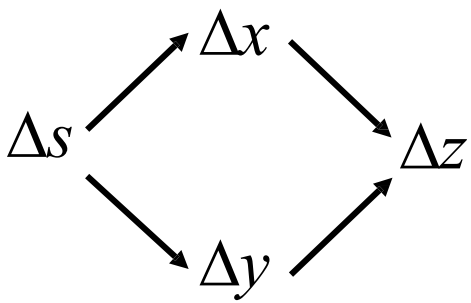
# Chain Rule

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

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

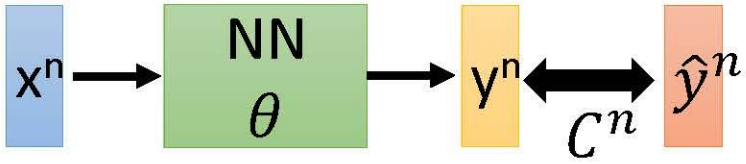
***Case 2***

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

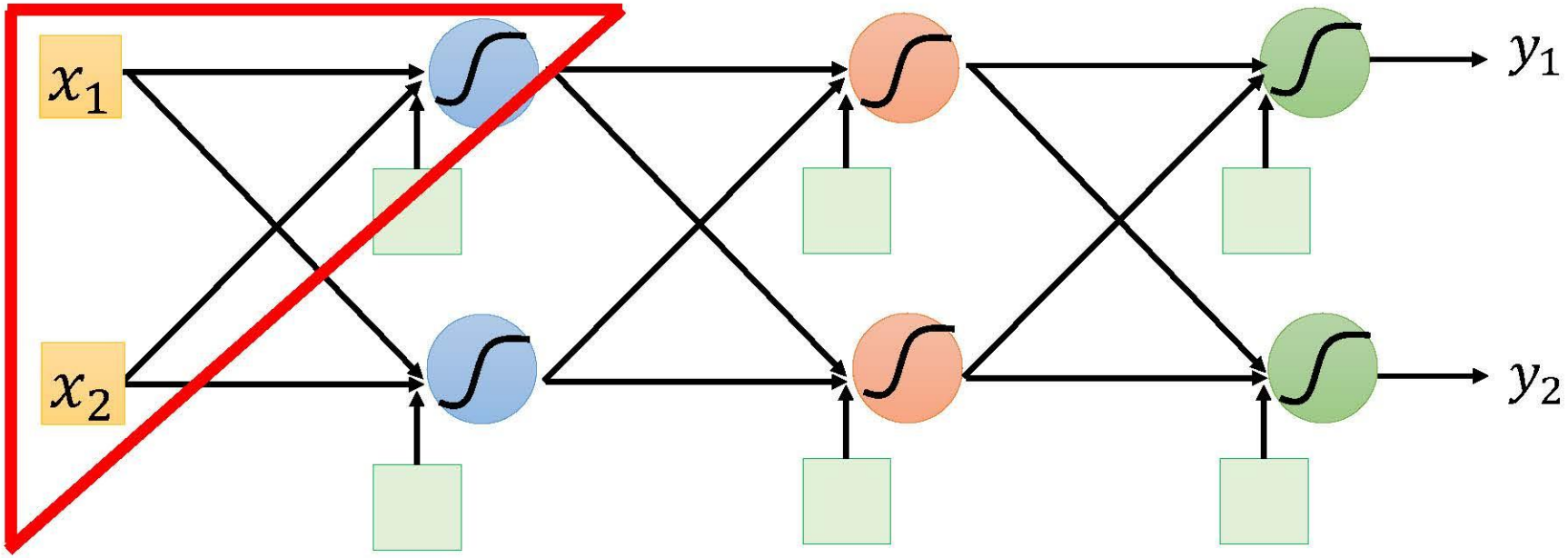



$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

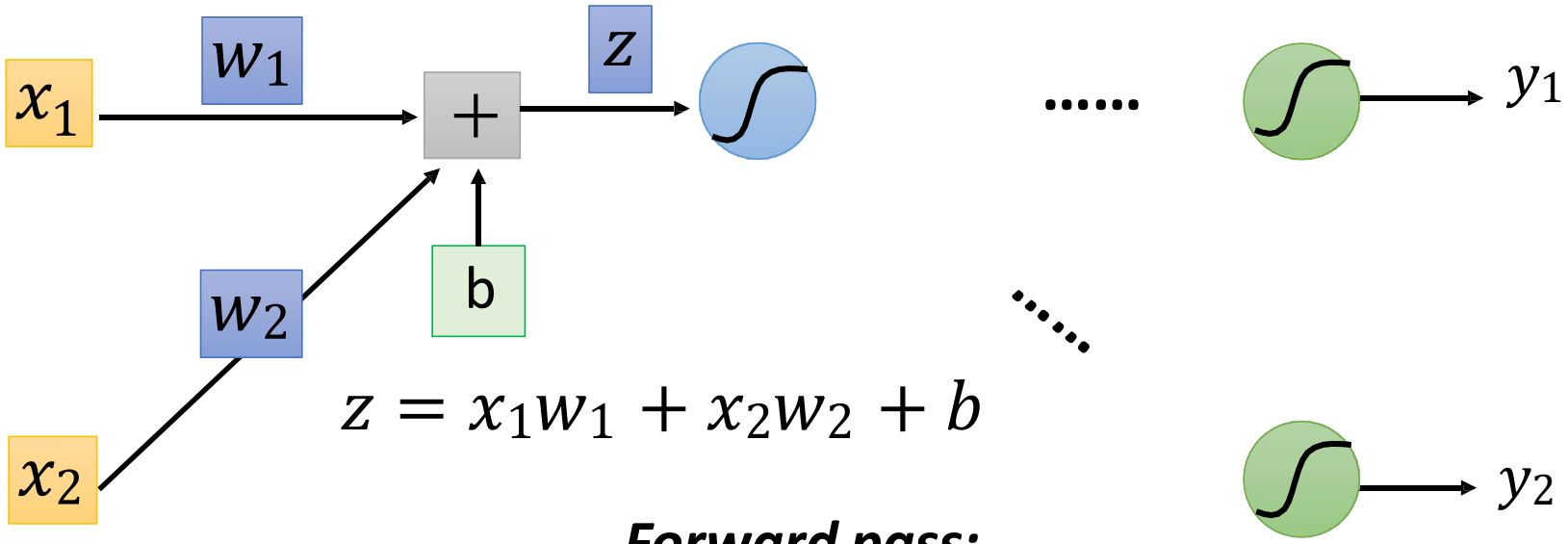
# Backpropagation




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



# Backpropagation




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

***Forward pass:***

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

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

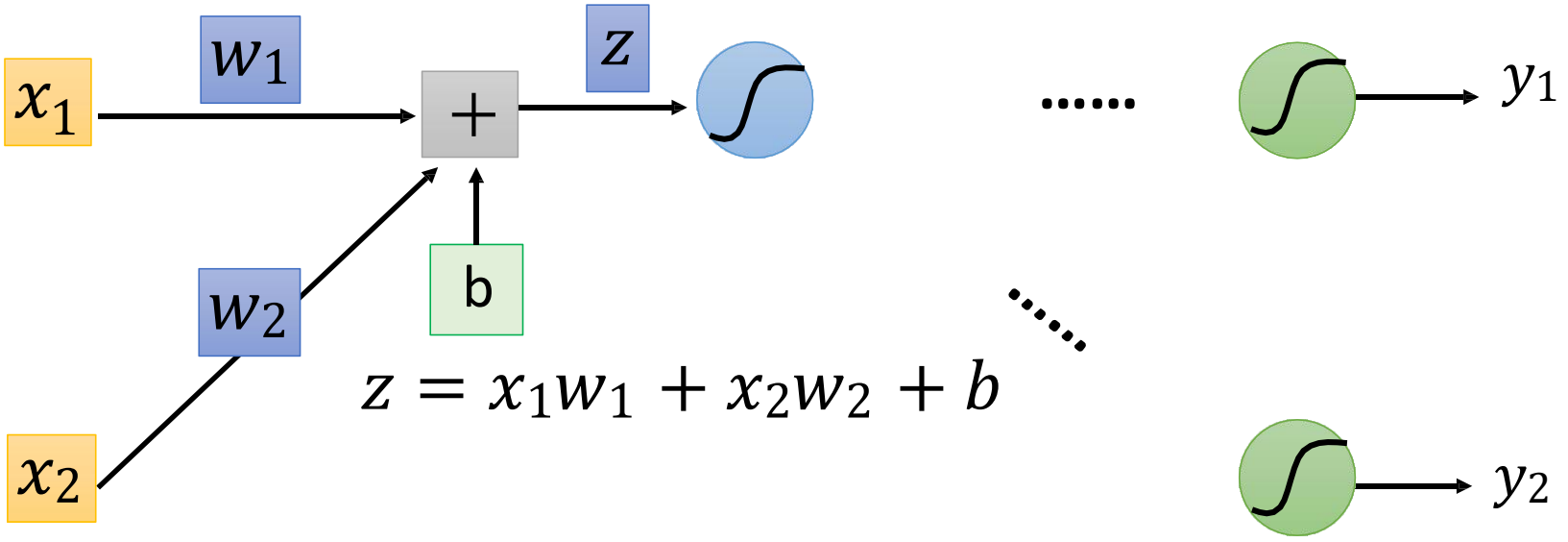
(Chain rule)

***Backward pass:***

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

# Backpropagation – Forward pass

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



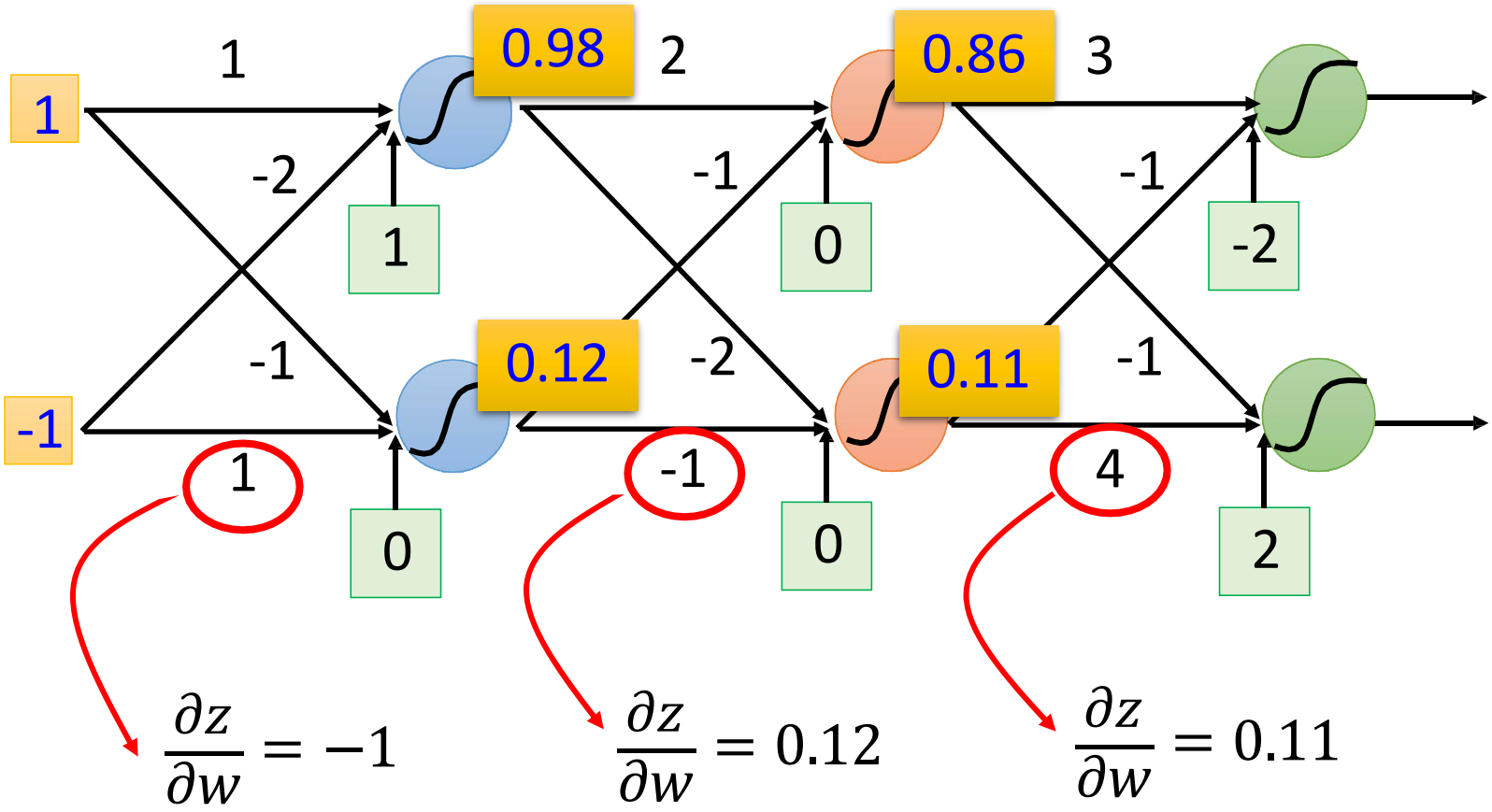

$\partial z/\partial w_1 =?\ \ x_1$

$\partial z/\partial w_2 =?\ \ x_2$

The value of the input connected by the weight

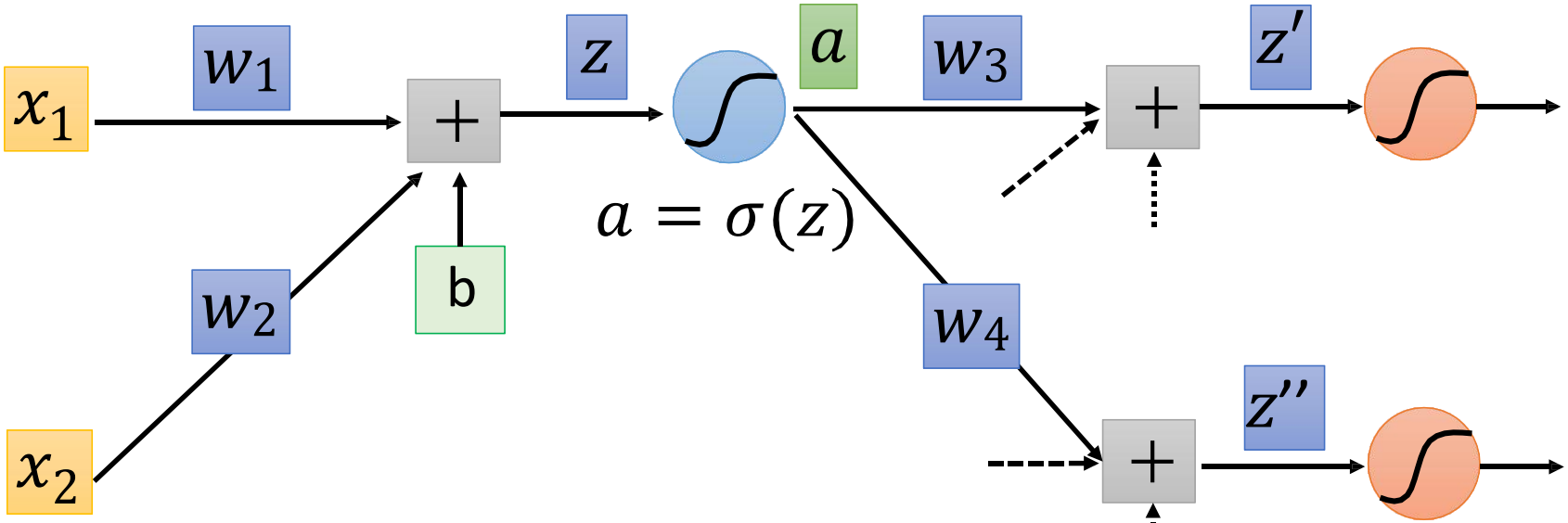
# Backpropagation – Forward pass

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



# Backpropagation – Backward pass

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

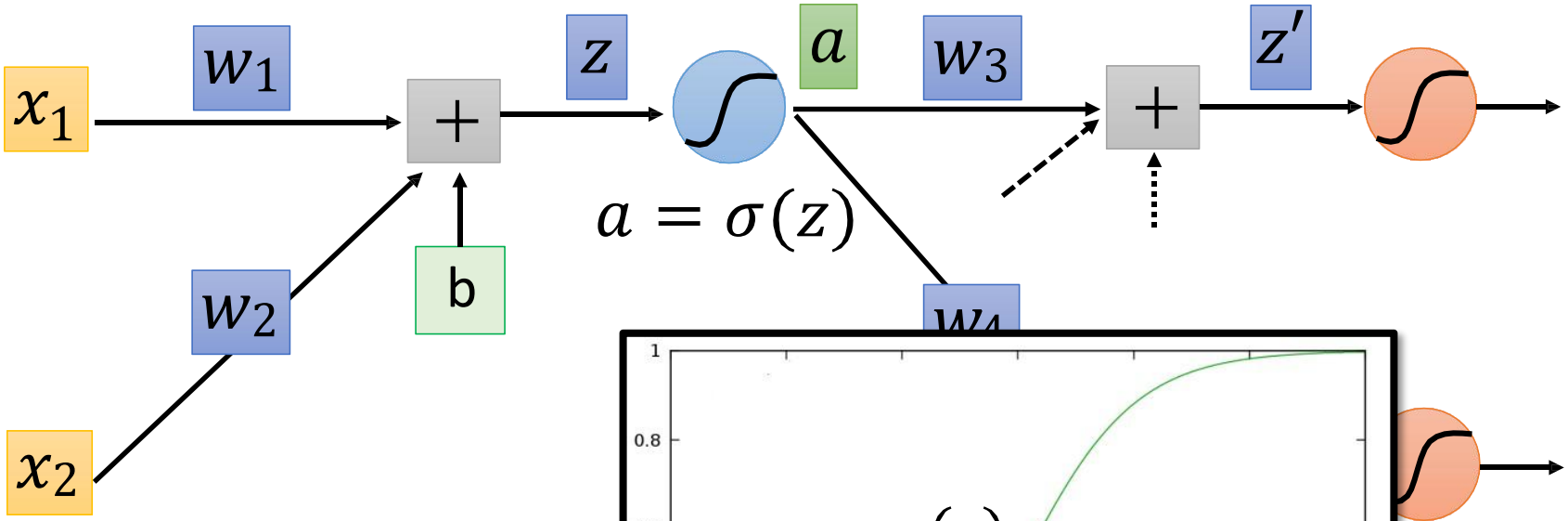



$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

$\sigma'(z)$

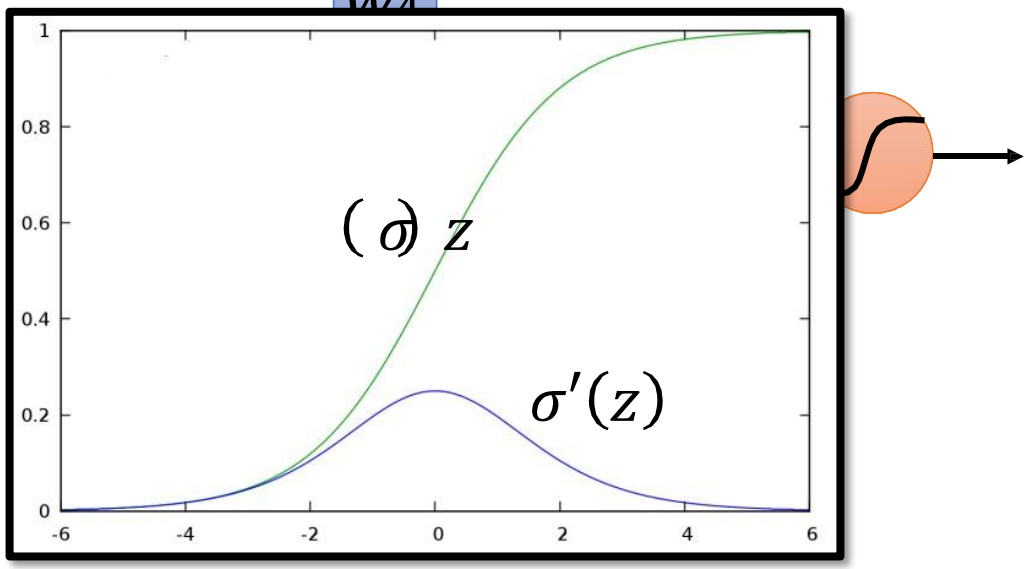
# Backpropagation – Backward pass

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



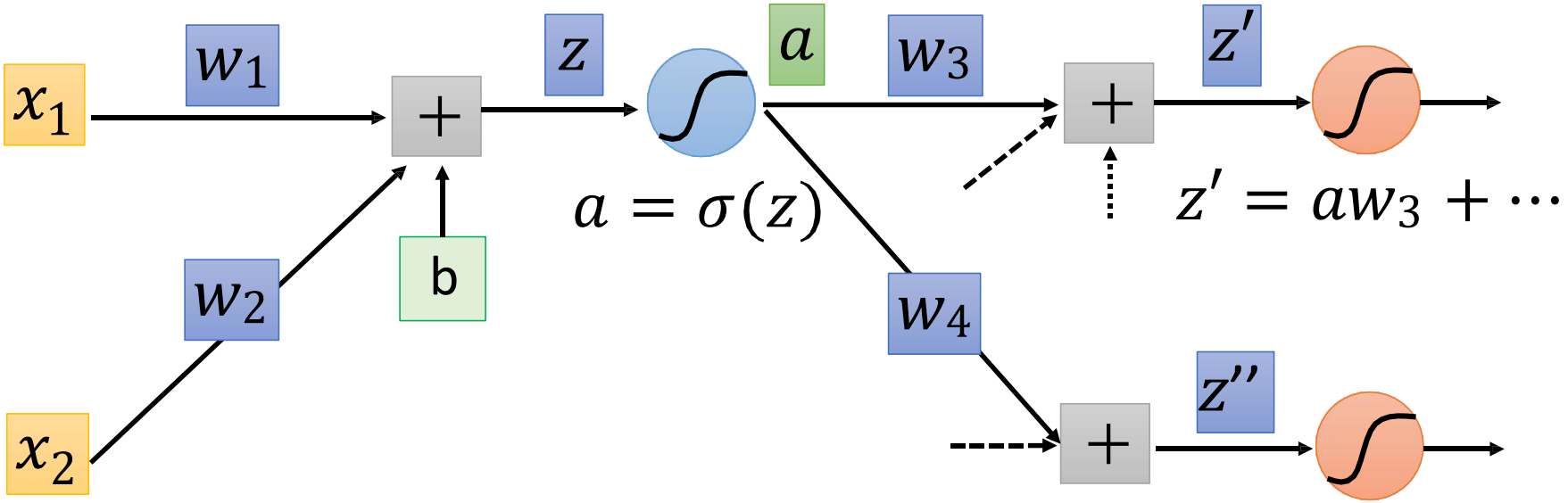

$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

$\Rightarrow \sigma'(z)$



# Backpropagation – Backward pass

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




$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

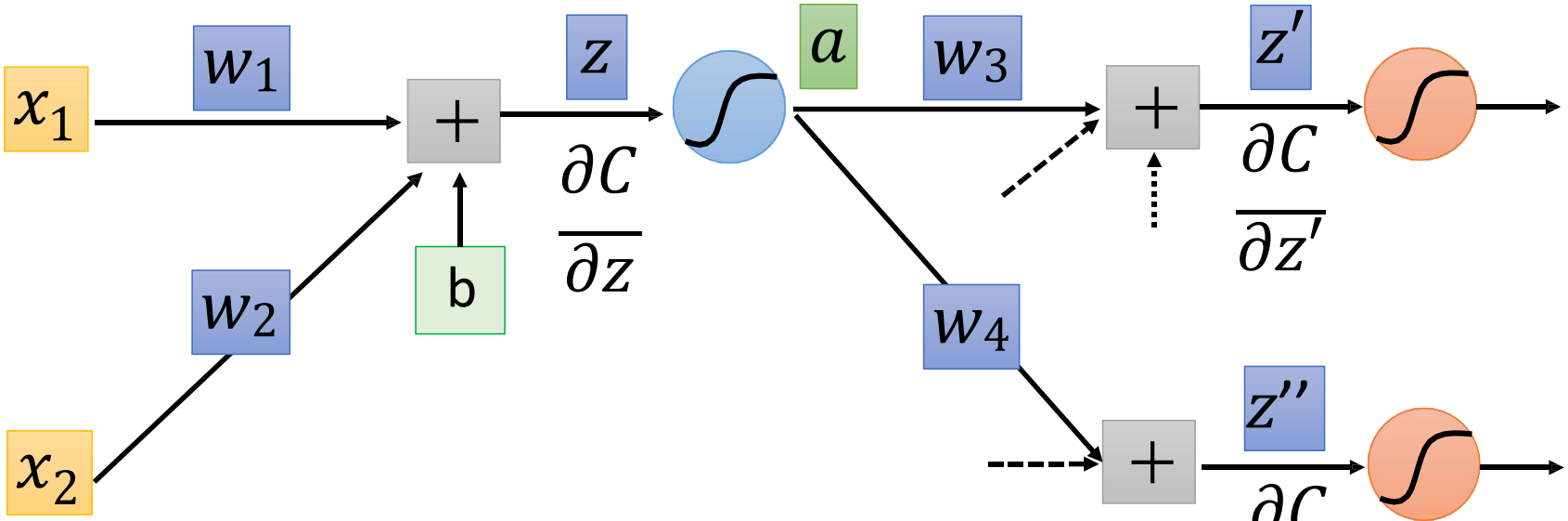
$$\frac{\partial C}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial C}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial C}{\partial z''}$$ (Chain rule)

$\frac{\partial z'}{\partial a} = w_3$, $\frac{\partial C}{\partial z'}$ = ?, $\frac{\partial z''}{\partial a} = w_4$, $\frac{\partial C}{\partial z''}$ = ?

Assumed it's known

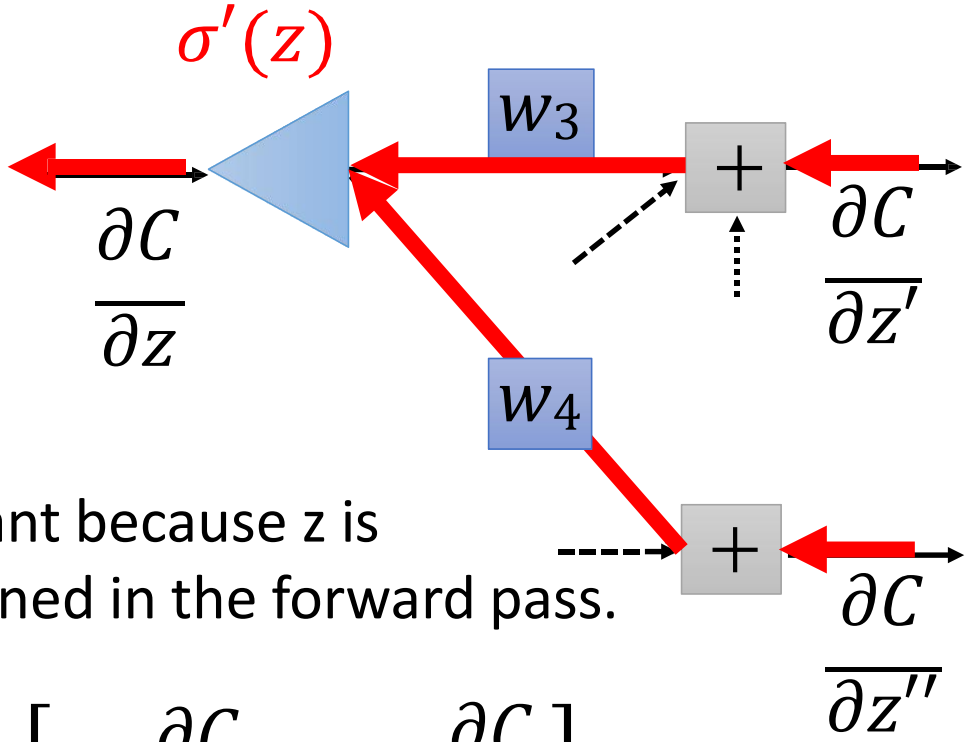
# Backpropagation – Backward pass

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




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

# Backpropagation – Backward pass

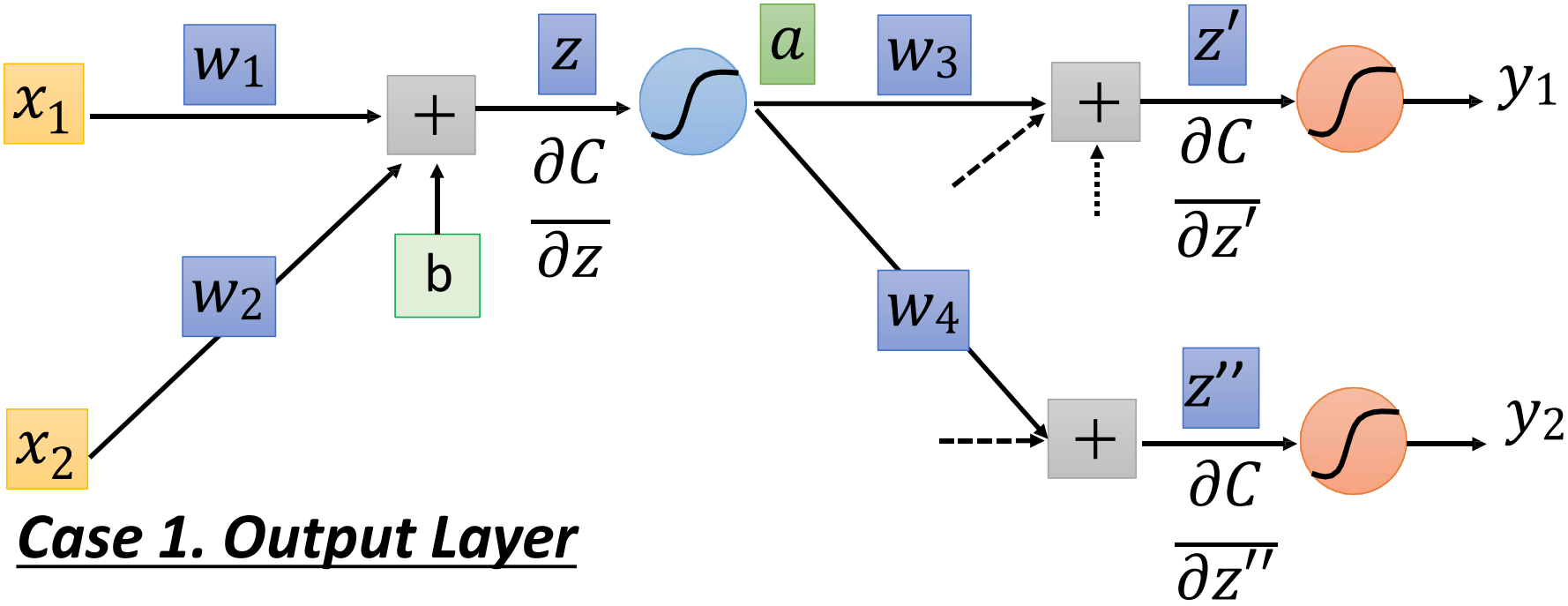



$\sigma'(z)$ is a constant because z is already determined in the forward pass.

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

# Backpropagation – Backward pass

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




***Case 1. Output Layer***

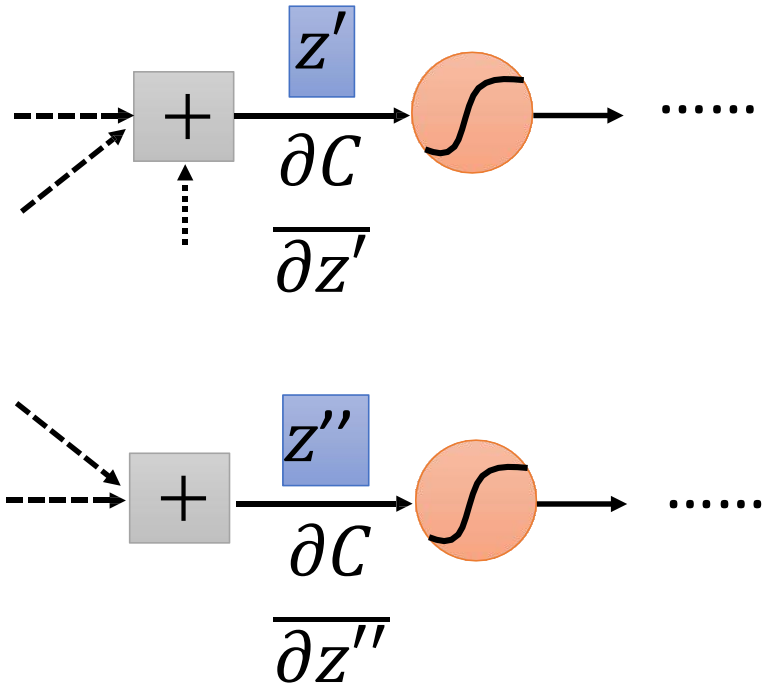
$$\frac{\partial C}{\partial z'} = \frac{\partial y_1}{\partial z'}\frac{\partial C}{\partial y_1} \qquad \frac{\partial C}{\partial z''} = \frac{\partial y_2}{\partial z''}\frac{\partial C}{\partial y_2}$$

Done!

# Backpropagation – Backward pass

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

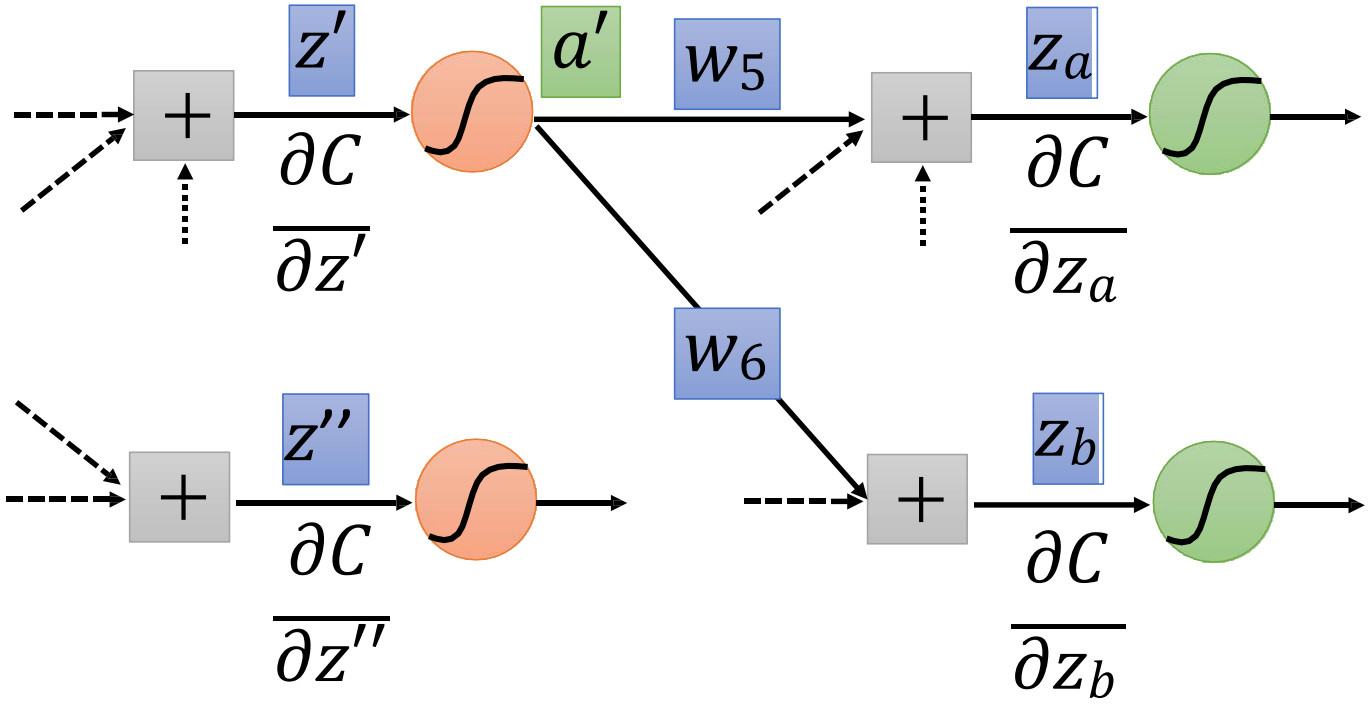
***Case 2. Not Output Layer***



# Backpropagation – Backward pass

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

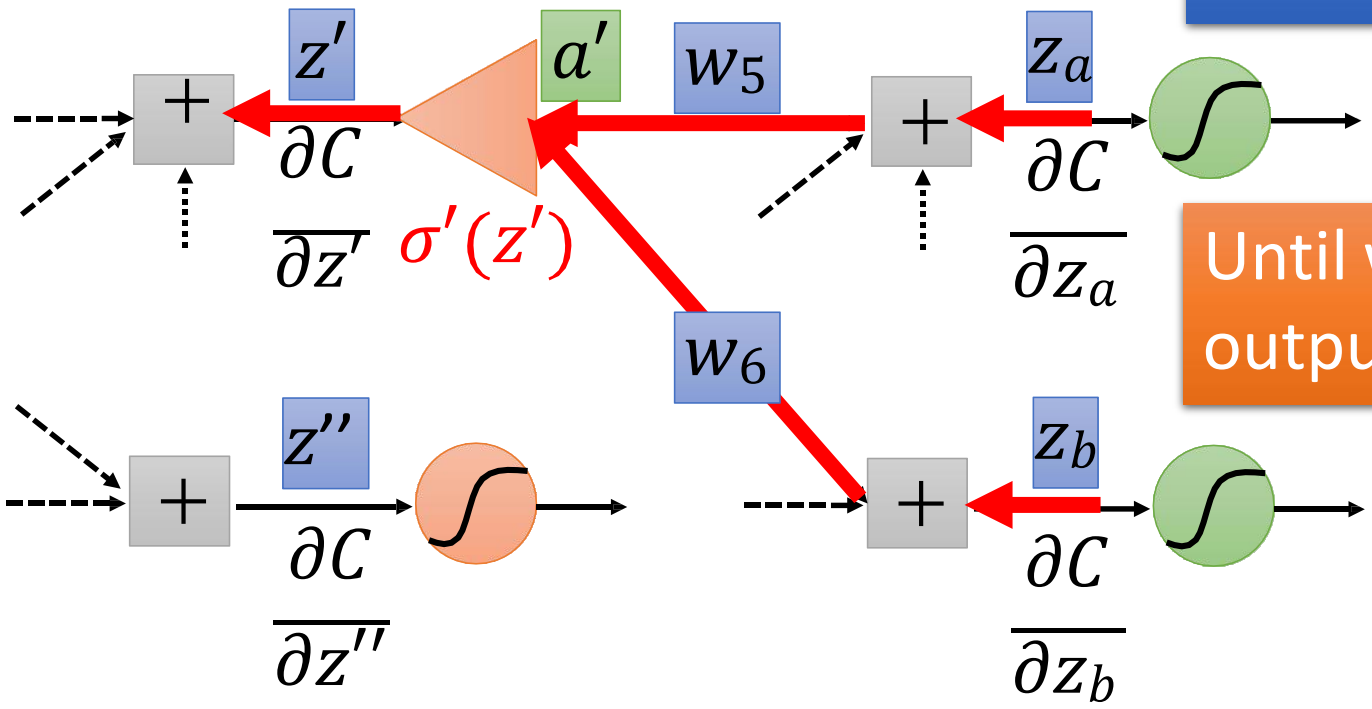
***Case 2. Not Output Layer***



# Backpropagation – Backward pass

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

***Case 2. Not Output Layer***



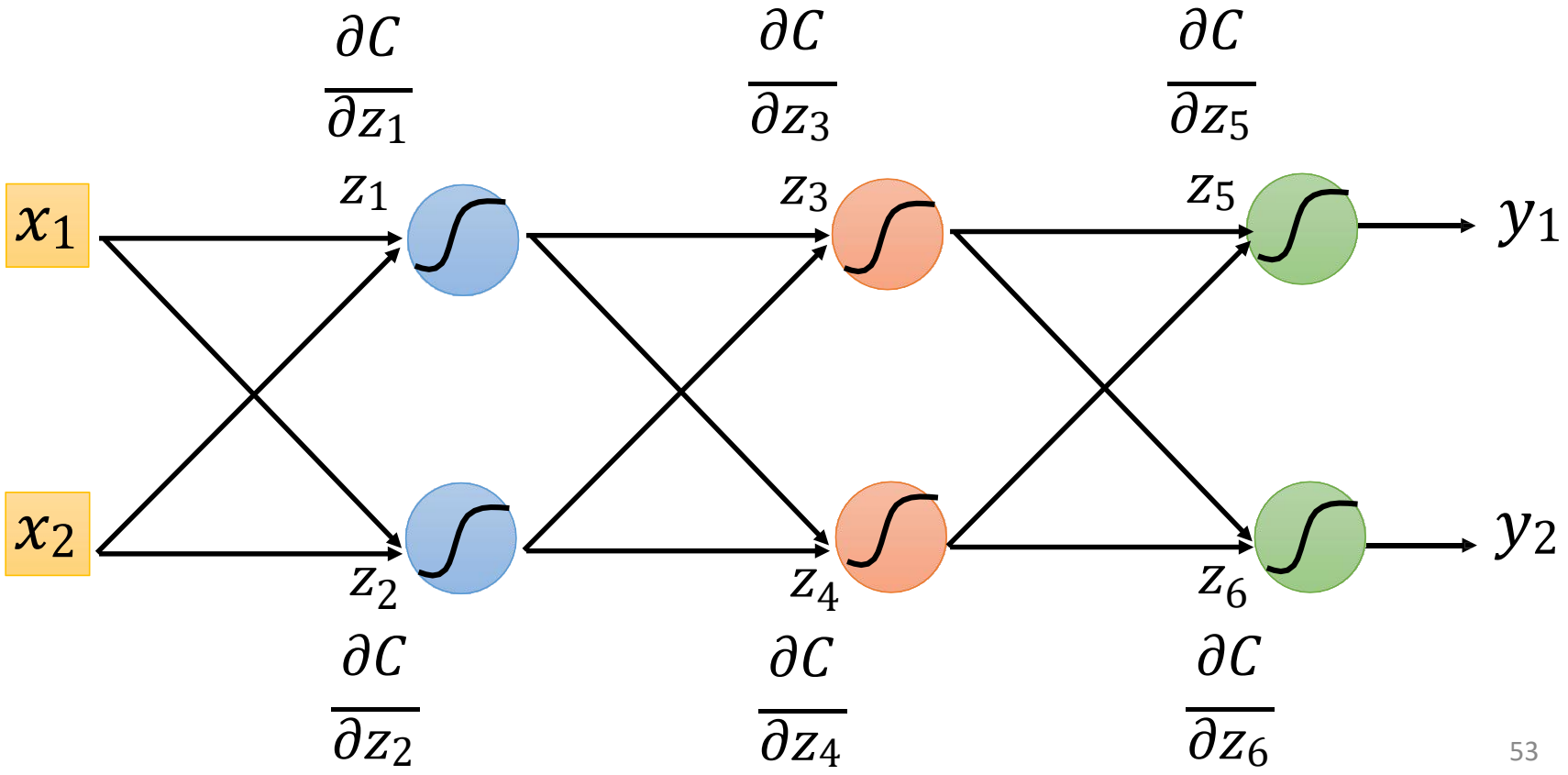

Compute $\partial C/\partial z$ recursively

Until we reach the output layer ……

# Backpropagation – Backward Pass

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

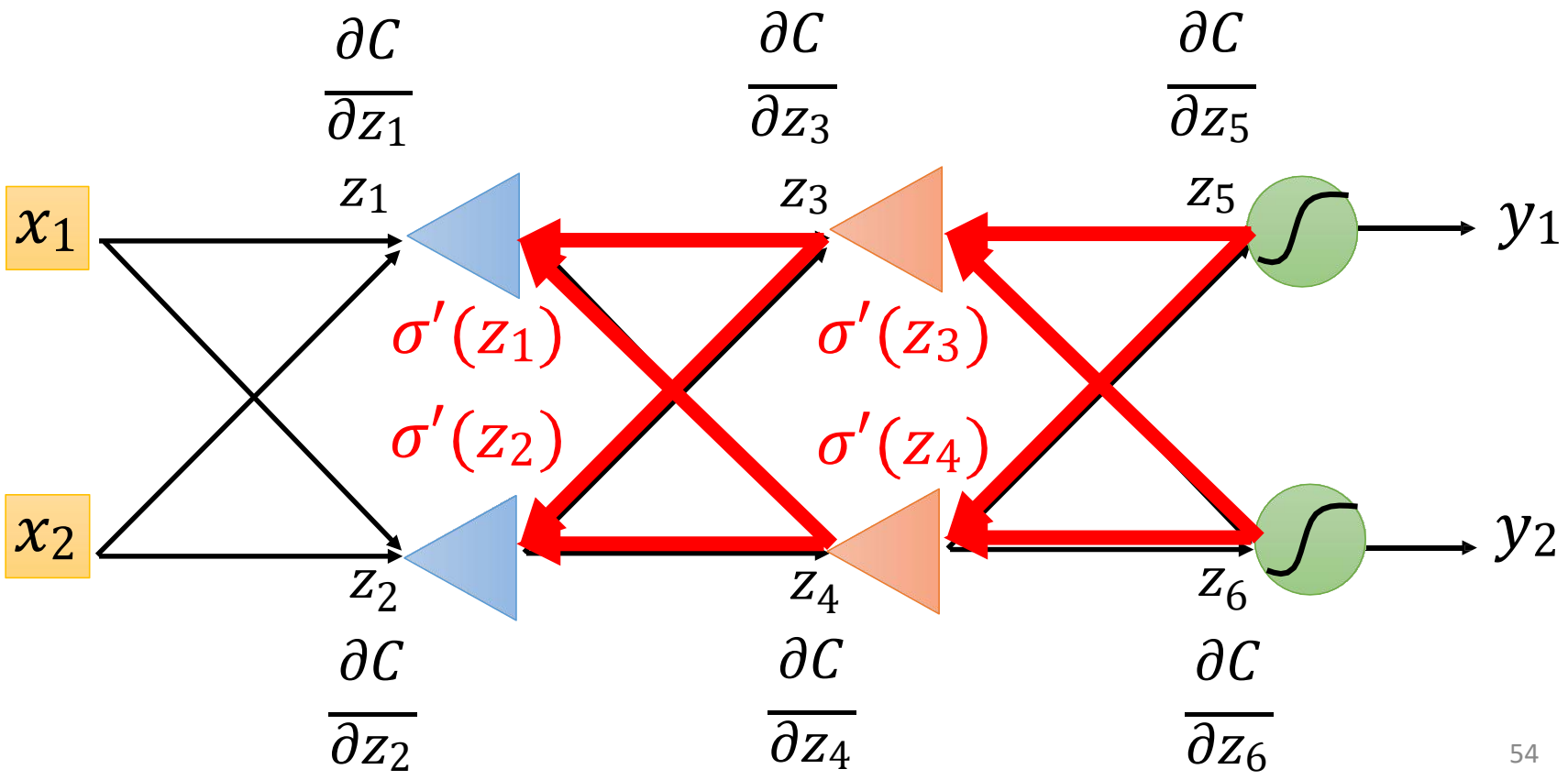
Compute $\partial C/\partial z$ from the output layer



# Backpropagation – Backward Pass

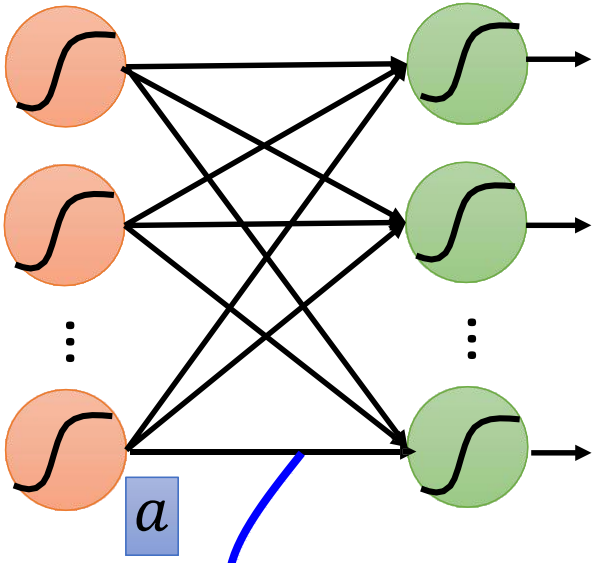
Compute $\partial C/\partial z$ for all activation function inputs z

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



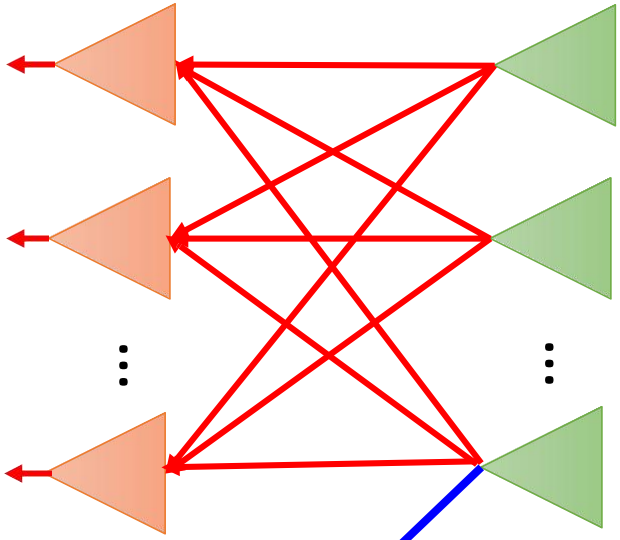
# Backpropagation – Summary

***Forward Pass***




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

***Backward Pass***



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

for all w

# Backpropagation

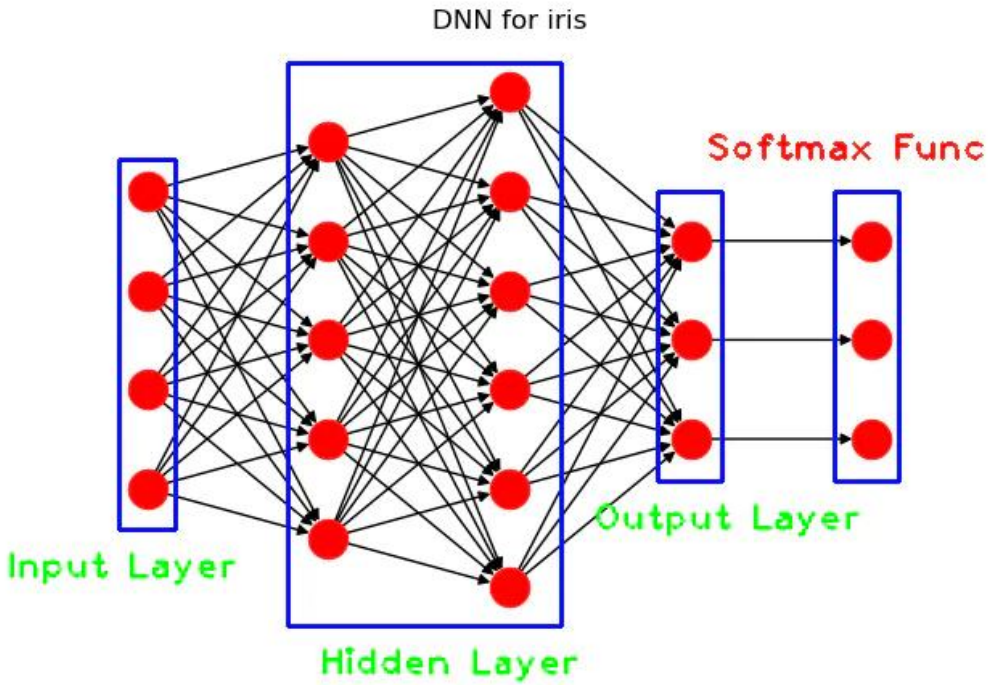

- You don’t do it by your own, just use
  - Tensorflow (google)
  - Keras (on Tensorflow, CNTK, Theano)
  - Pytorch (facebook)
  - Caffe2 (facebook)
  - Paddlepaddle (百度)
  - Mindspore (华为)
  - CNTK(Microsoft)
  - MXNet(Amazon)
  - Deeplearing4j

# DNN实例（IRIS数据集）

- 经典数据集，鸢尾花数据集， 分类问题，http://archive.ics.uci.edu/ml/machine-learning-databases/iris/
- 150个样本，每个样本4个特征，分为3类

| | sepal_length | sepal_width | petal_length | petal_width | class |
|---|---|---|---|---|---|
| 128 | 6.4 | 2.8 | 5.6 | 2.1 | virginica |
| 18 | 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 130 | 7.4 | 2.8 | 6.1 | 1.9 | virginica |
| 105 | 7.6 | 3.0 | 6.6 | 2.1 | virginica |
| 107 | 7.3 | 2.9 | 6.3 | 1.8 | virginica |
| 78 | 6.0 | 2.9 | 4.5 | 1.5 | versicolor |
| 83 | 6.0 | 2.7 | 5.1 | 1.6 | versicolor |
| 14 | 5.8 | 4.0 | 1.2 | 0.2 | setosa |
| 5 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 133 | 6.3 | 2.8 | 5.1 | 1.5 | virginica |

# Keras搭建DNN




```python
import keras as K
# 2. 定义模型
init = K.initializers.glorot_uniform(seed=1)
simple_adam = K.optimizers.Adam()
model = K.models.Sequential()
model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=6, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=simple_adam, metrics=['accuracy'])
```

# 模型训练

```
# 3. 训练模型
b_size = 1
max_epochs = 100
print("Starting training ")
h = model.fit(train_x, train_y, batch_size=b_size, epochs=max_epochs, shuffle=True, verbose=1)
print("Training finished \n")
```

# 模型测试

```
# 4. 评估模型
eval = model.evaluate(test_x, test_y, verbose=0)
print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" \
      % (eval[0], eval[1] * 100) )
```