

Natural Language Processing with Deep Learning

IFT6289, Winter 2022

Lecture 2: Neural Networks and Backpropagation

Bang Liu

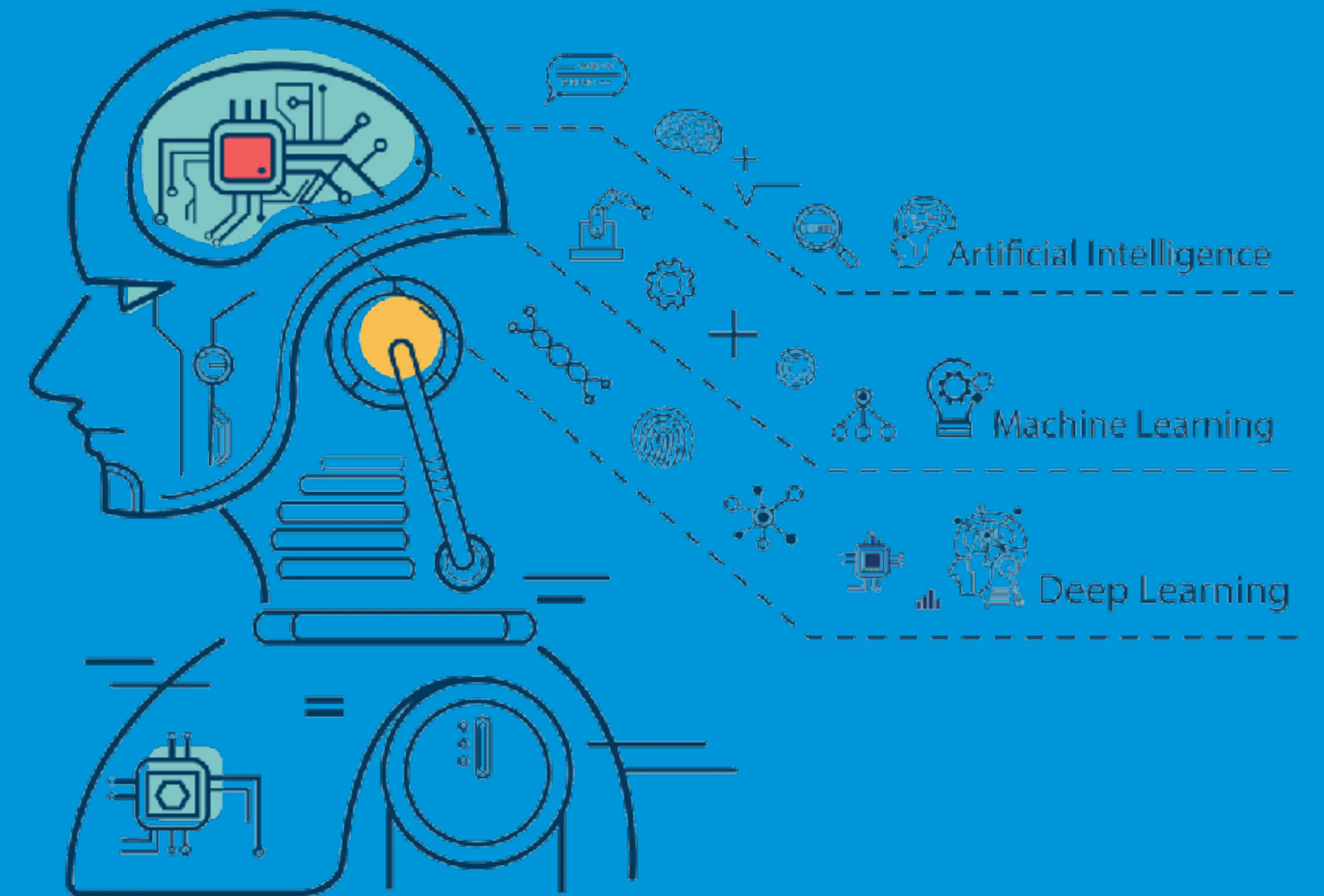
2 Lecture outline

1. Machine Learning and Deep Learning
2. Training Deep Neural Networks by Gradient Descent
3. Computing Gradients by Backpropagation

3 Certain Slides Adapted From or Referred To...

- ◎ Slides from NTU S-108, Yun-Nung (Vivian) Chen
 - **Applied Deep Learning, Spring 2020:** <https://www.csie.ntu.edu.tw/~miulab/s108-adl/syllabus>, lecture 2

Machine Learning and Deep Learning



What is Machine Learning

6 What Computers Can Do

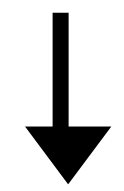
- Programs can do the things you ask them to do.



7 Program for Solving Tasks

- Task: predicting positive or negative given a product review

“I love this product!”



program.py

+

if input contains “love”, “like”, etc.
output = positive

“It claims too much.”

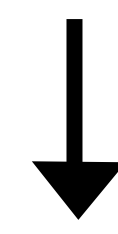


program.py

-

if input contains “too much”, “bad”, etc.
output = negative

“It’s a little expensive.”



program.py

?

Some tasks are complex, and we don’t know how to write a program to solve them.

8 Learning \approx Looking for a Function

- Task: predicting positive or negative given a product review

“I love this product!”

↓ f
+

if input contains “love”, “like”, etc.
output = positive



Positive

“It claims too much.”

↓ f
-

if input contains “too much”, “bad”, etc.
output = negative



Negative

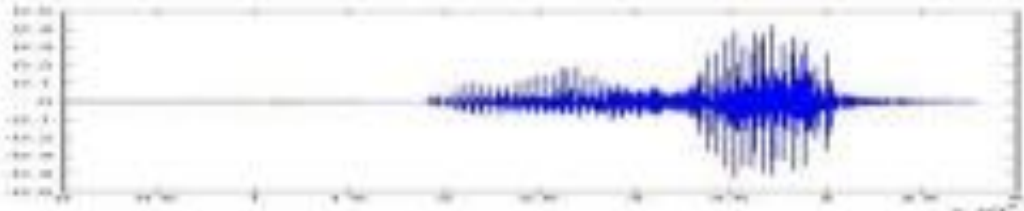
“It’s a little expensive.”

↓ f
?

Given a large amount of data, the machine learns what the function f should be

9 Learning \approx Looking for a Function



- Speech Recognition

$$f(\text{ ) = \textit{“hello”}$$

- Handwritten Recognition

$$f(\text{ ) = \textit{“2”}$$

- Weather Forecast

$$f(\text{  Thursday) = \textit{“  Saturday”}$$

- Play video games

$$f(\text{ ) = \textit{“move left”}$$

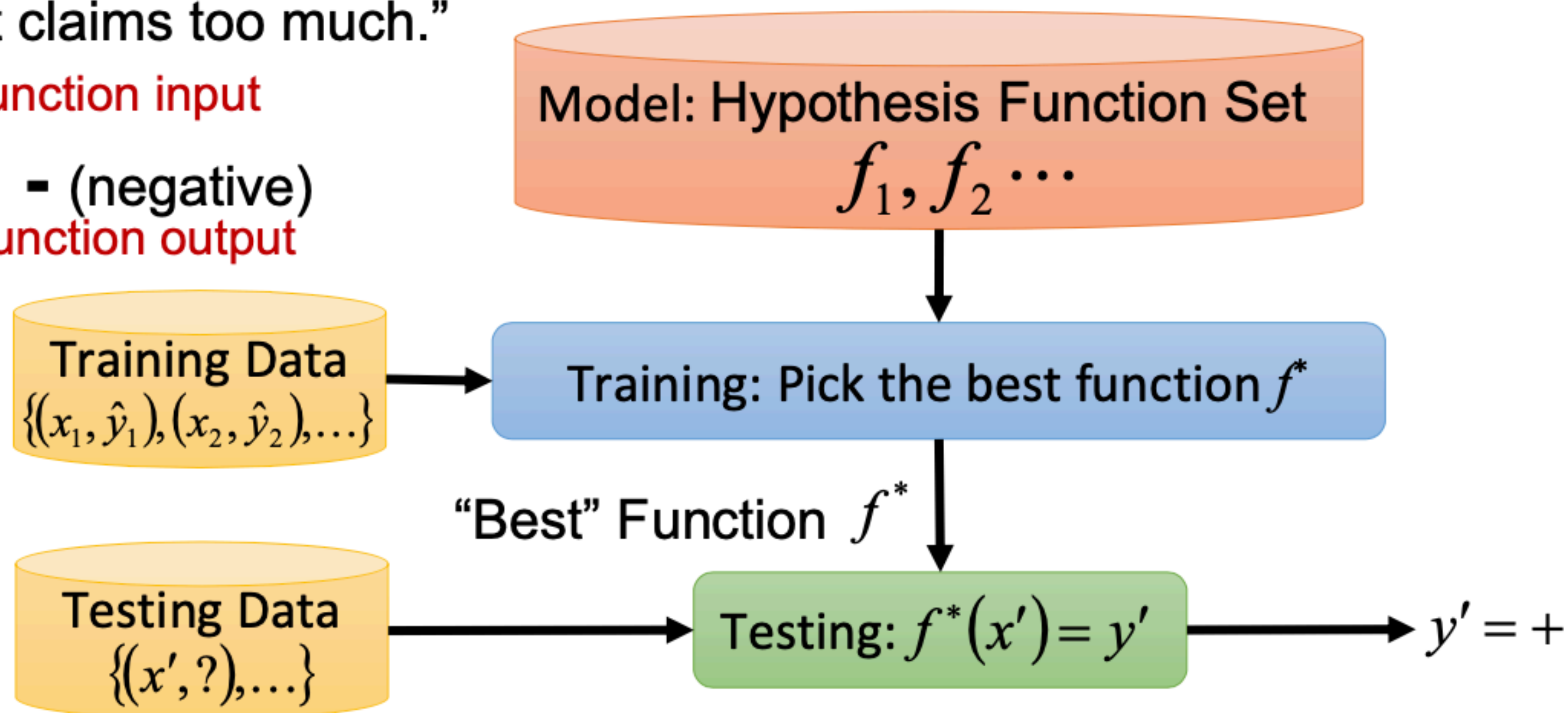
10 Machine Learning Framework

x : "It claims too much."

function input

\hat{y} : - (negative)

function output

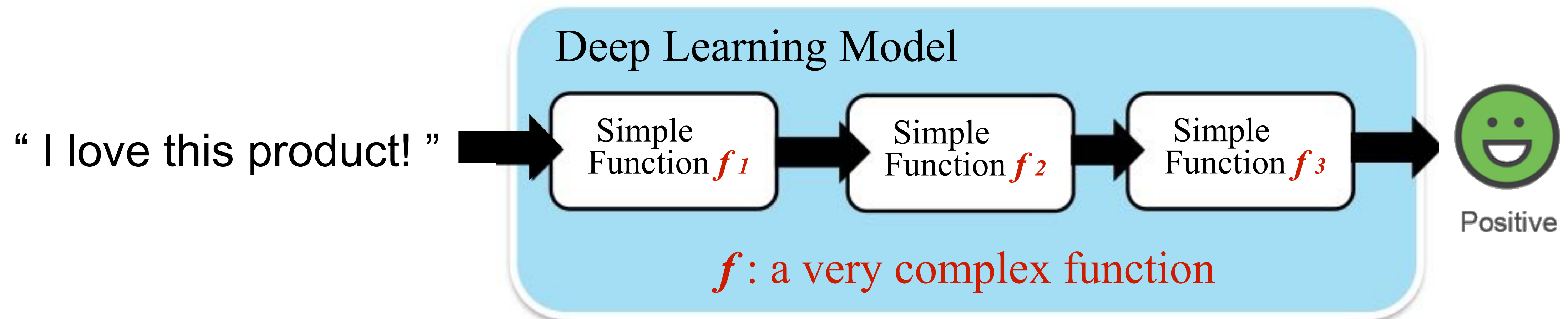


Training is to pick the best function given the observed data
Testing is to predict the label using the learned function

What is Deep Learning

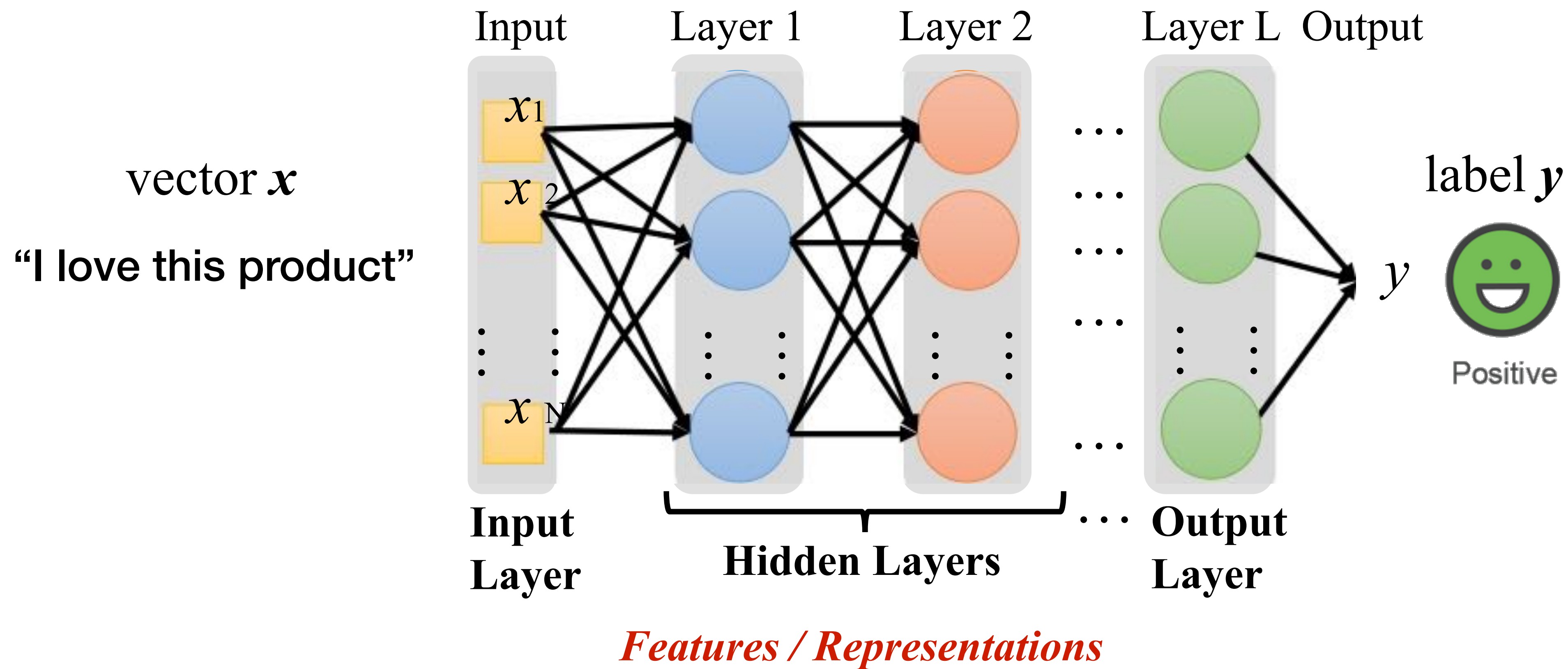
12 Stacked Functions Learned by Machine

- Production line



End-to-end training: what each function should do is learned automatically
Deep learning usually refers to **neural network** based model

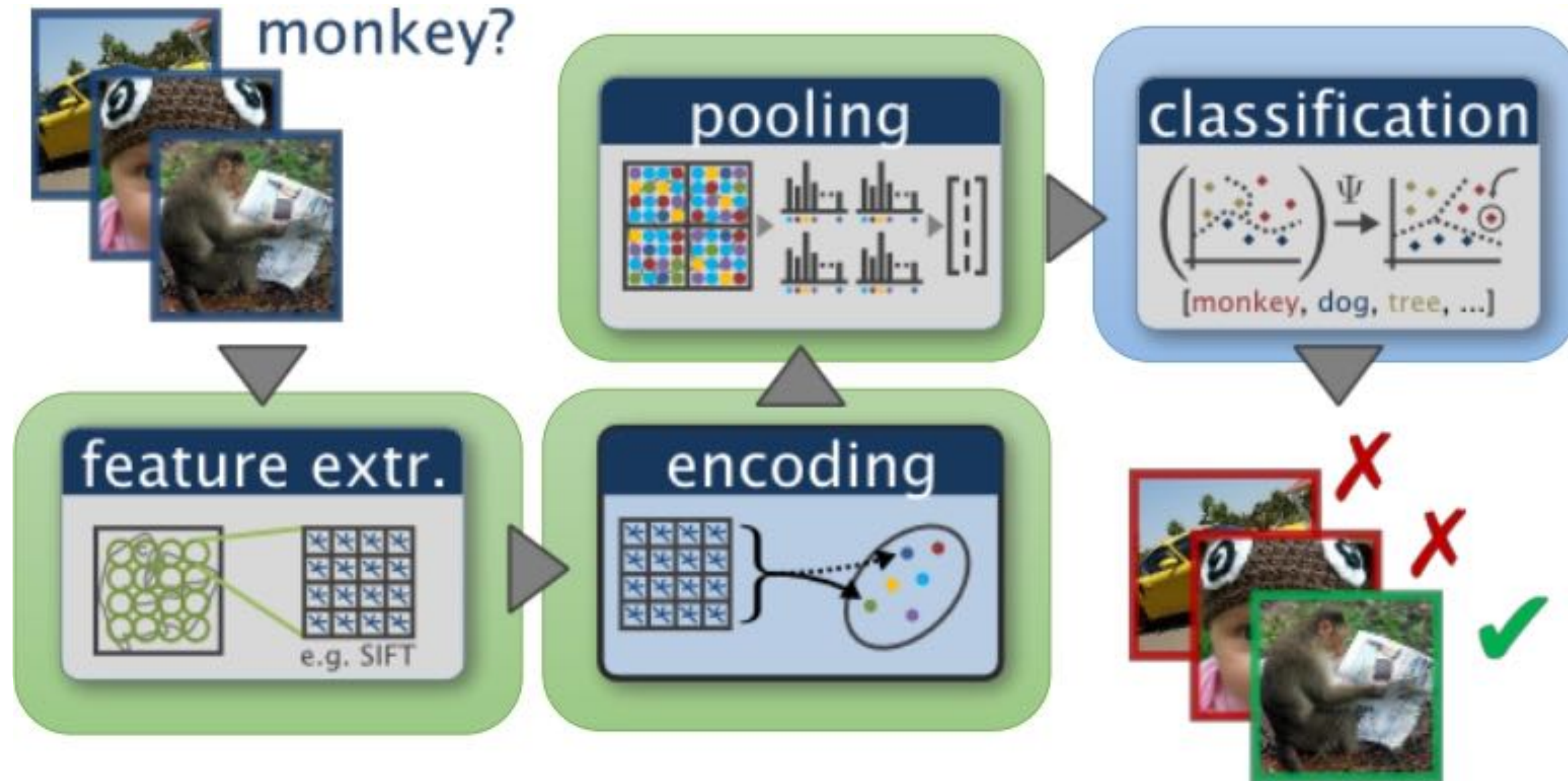
13 Stacked Functions Learned by Machine



Representation Learning attempts to learn good features/representations
Deep Learning attempts to learn (multiple levels of) representations and an output

14 Deep v.s. Shallow: Image Recognition

- Shallow model



:hand - crafted



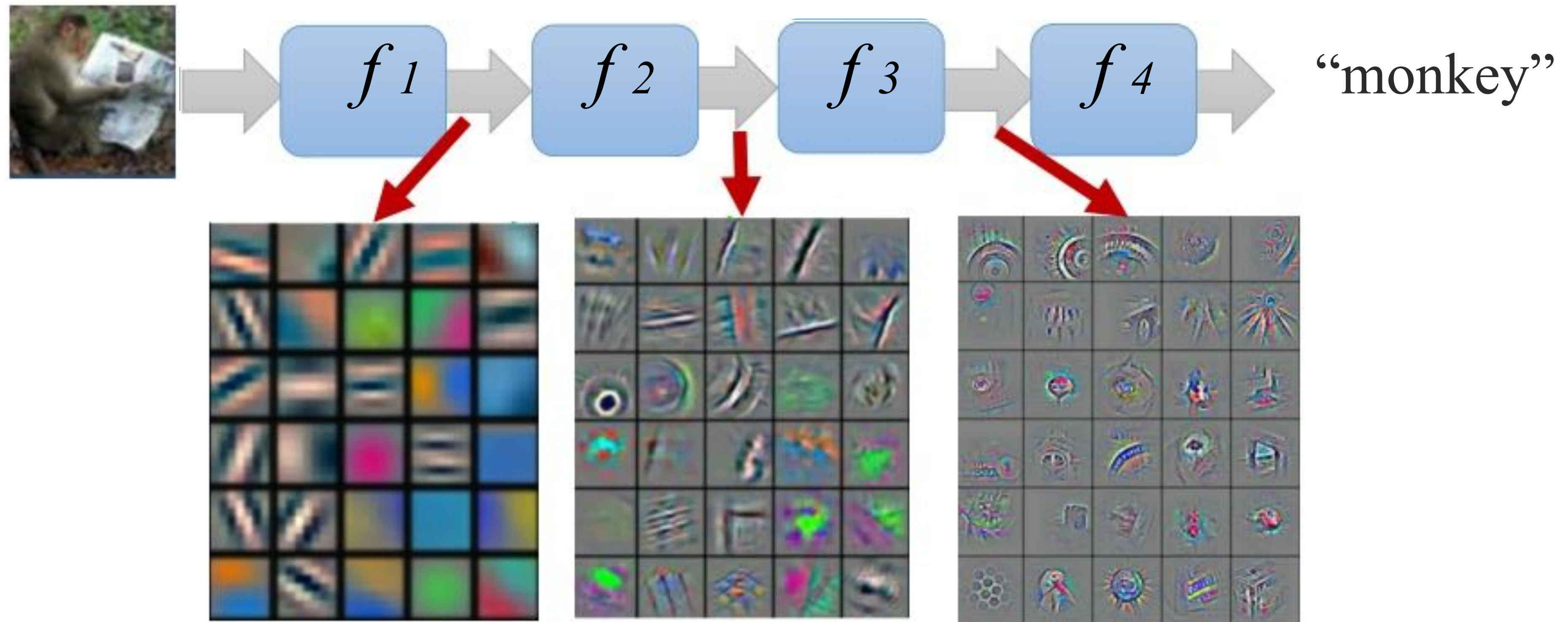
:learned from data

15 Deep v.s. Shallow: Image Recognition

- Deep model

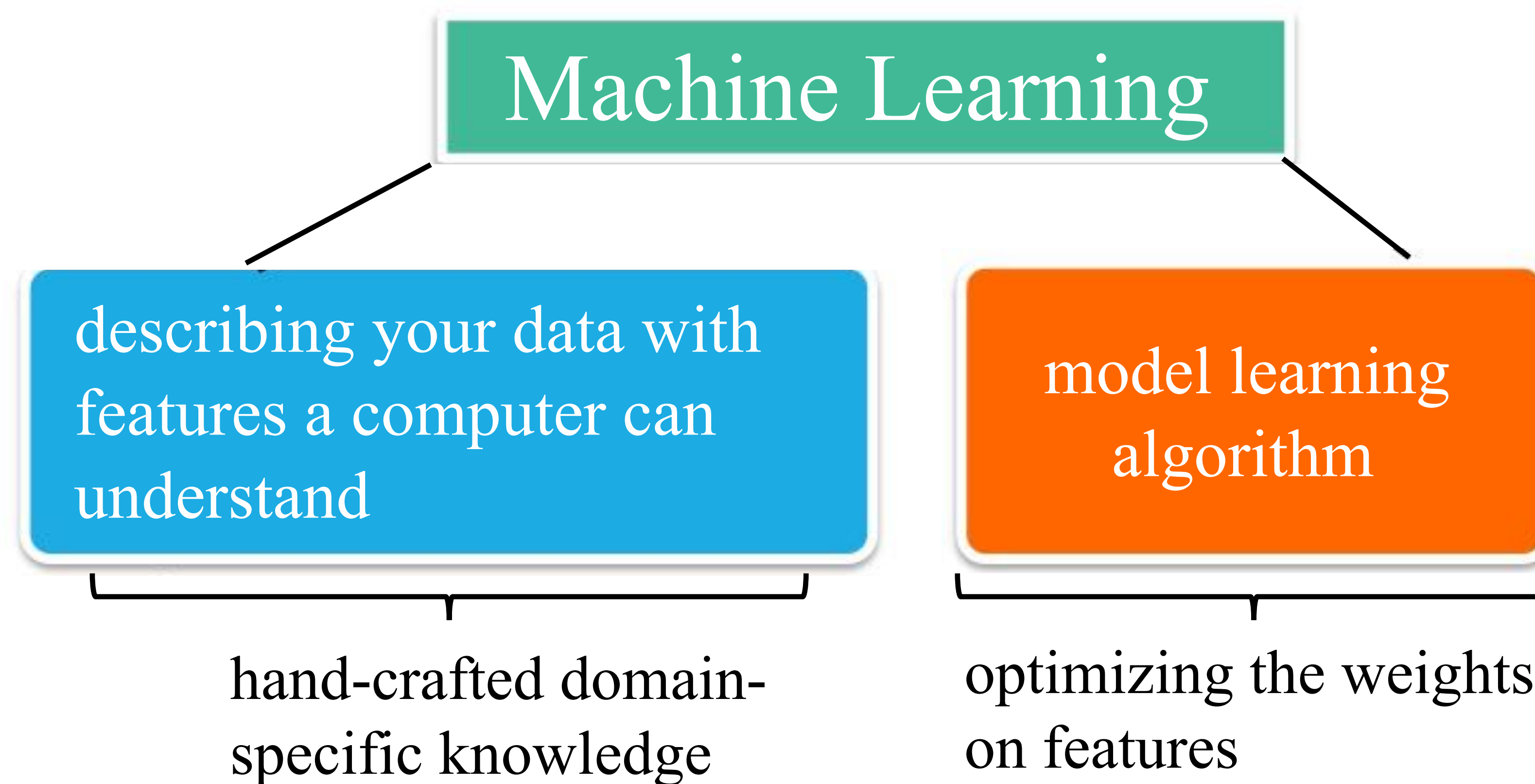
Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014* (pp. 818- 833)

All functions are learned from data

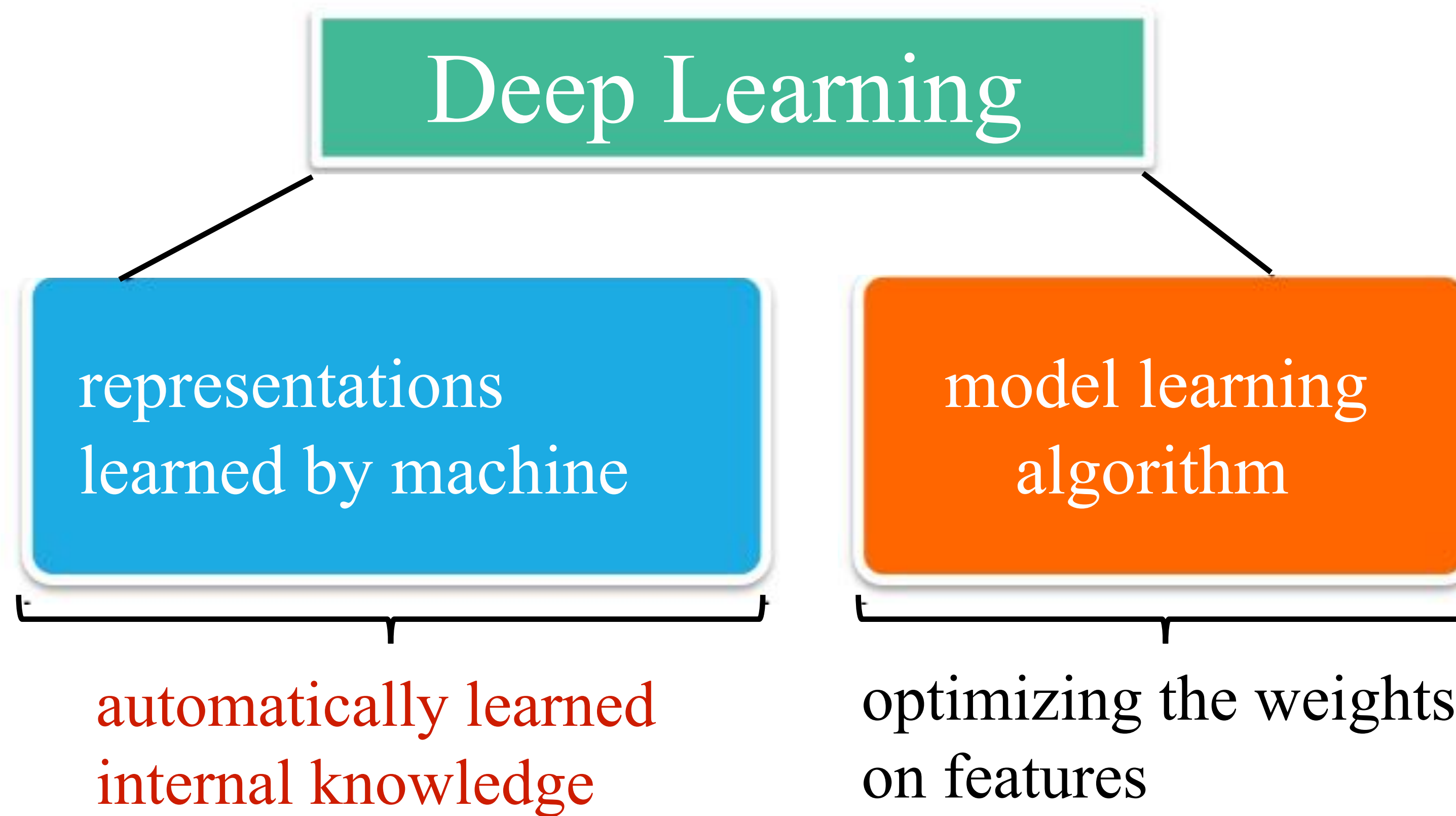


Features / Representations

16 Machine Learning v.s. Deep Learning

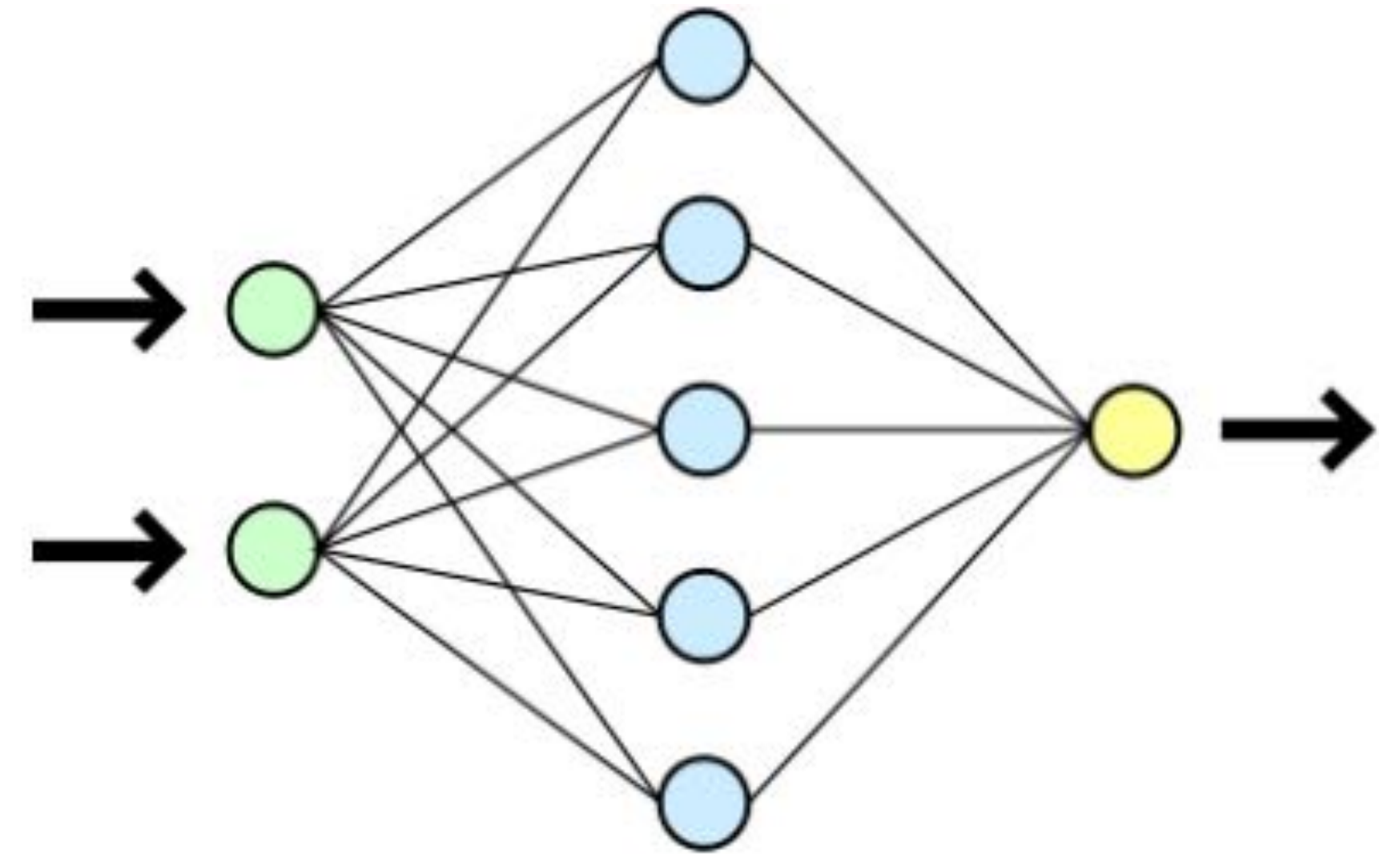
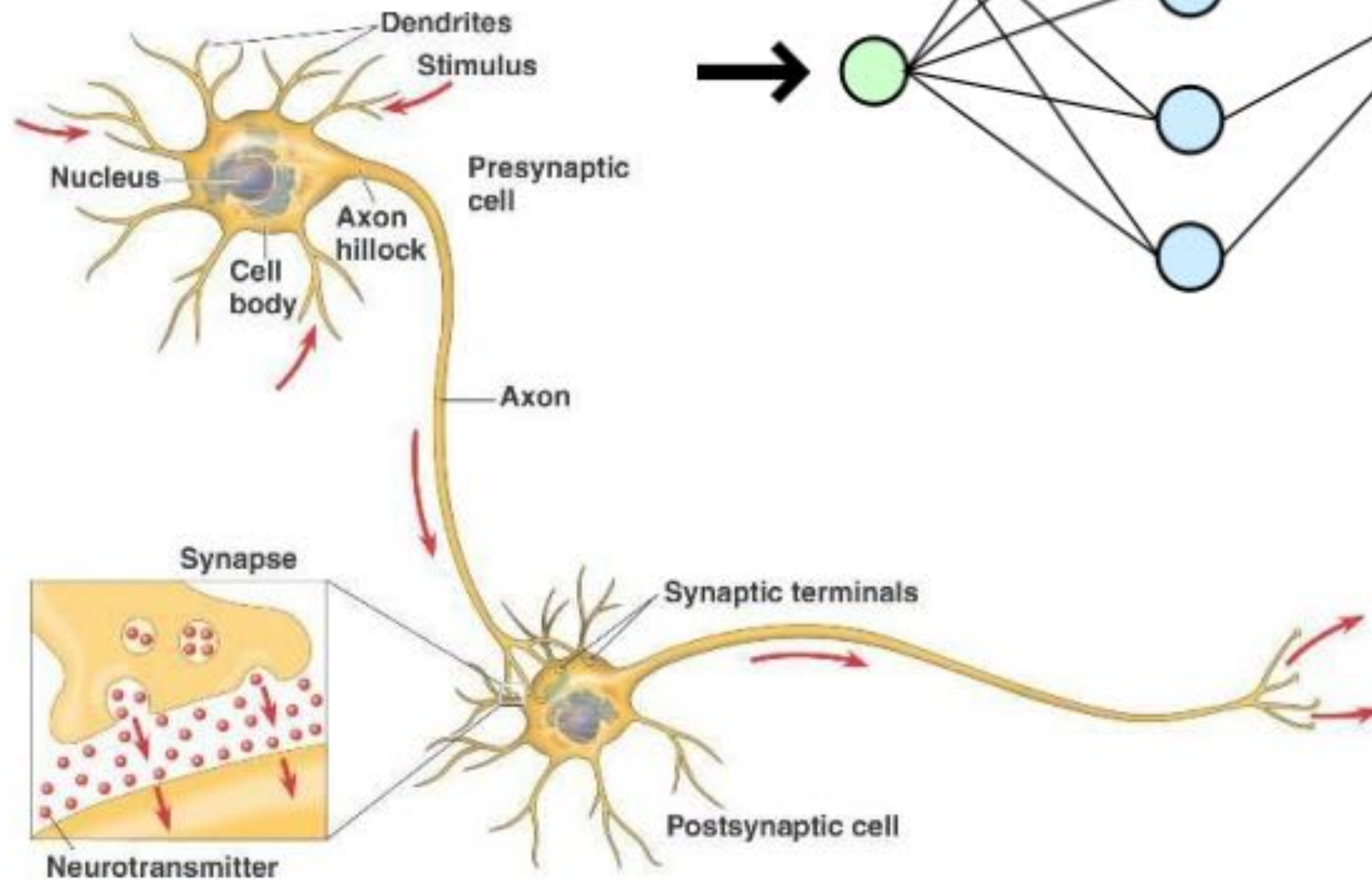


17 Machine Learning v.s. Deep Learning

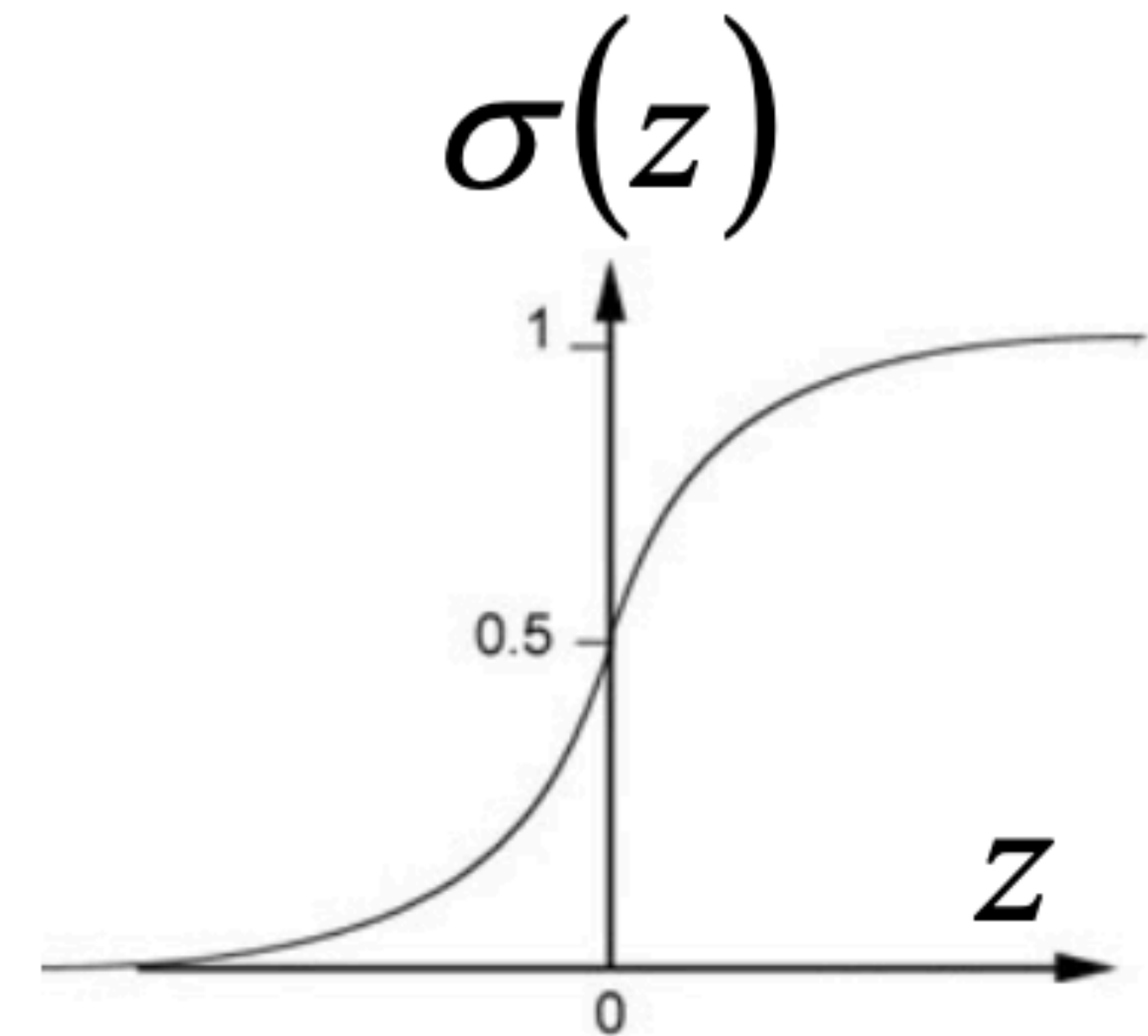
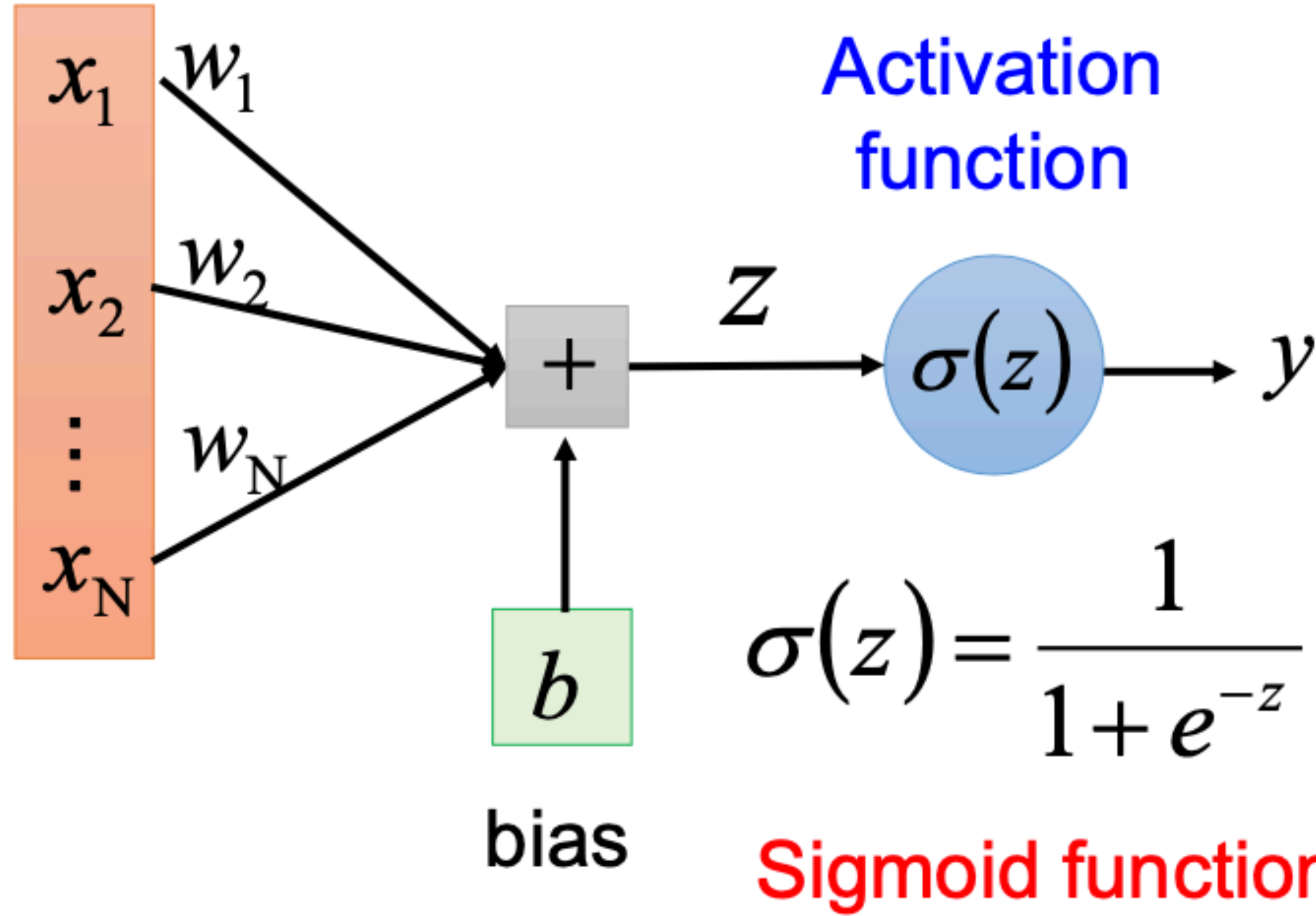


Deep learning usually refers to **neural network** based model

18 Inspired by Human Brain



19 A Single Neuron

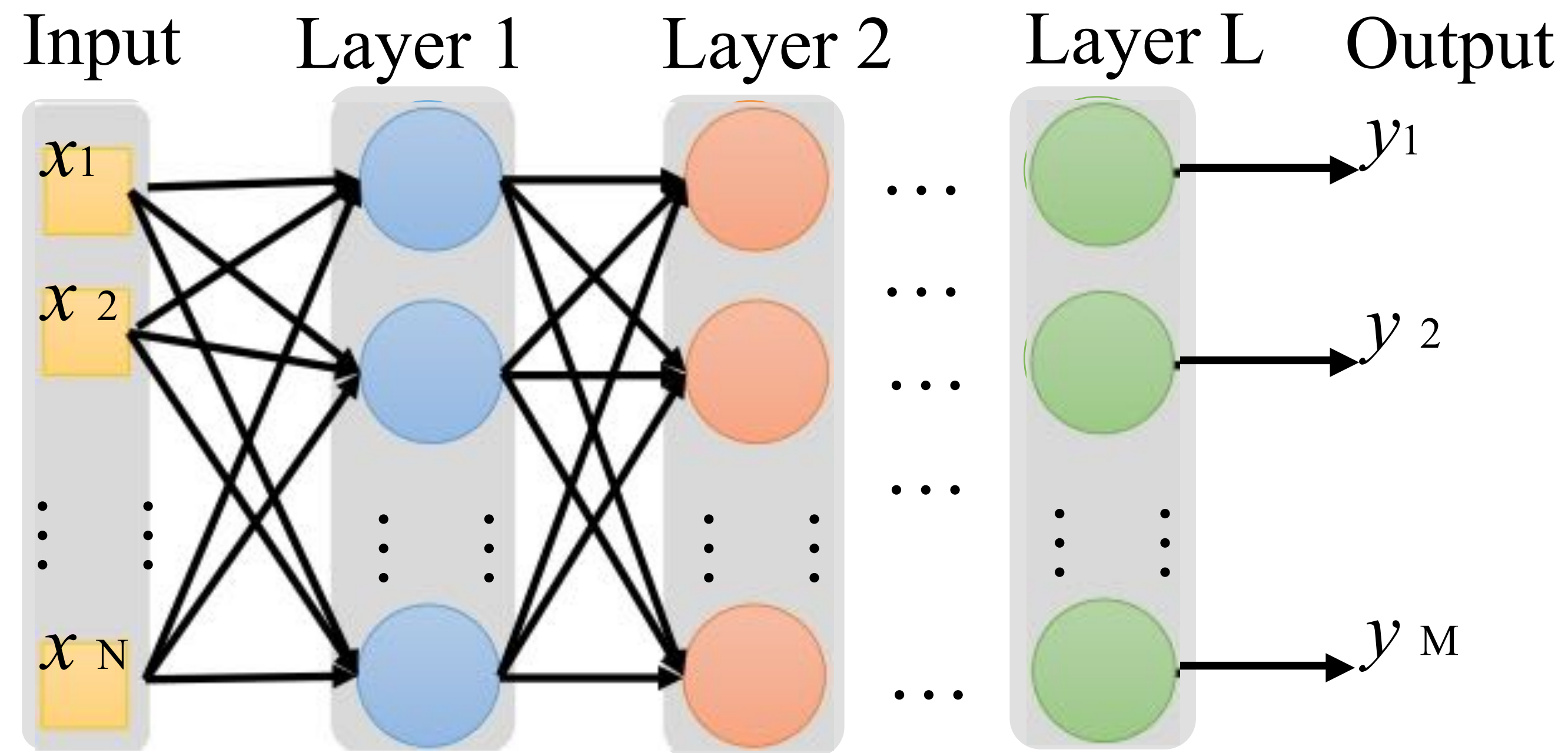


Each neuron is a very simple function

20 Deep Neural Network

- Cascading the neurons to form a neural network

A neural network is a complex function: $f : R^N \rightarrow R^M$



Each layer is a simple function in the production line

Why Deep Learning Works



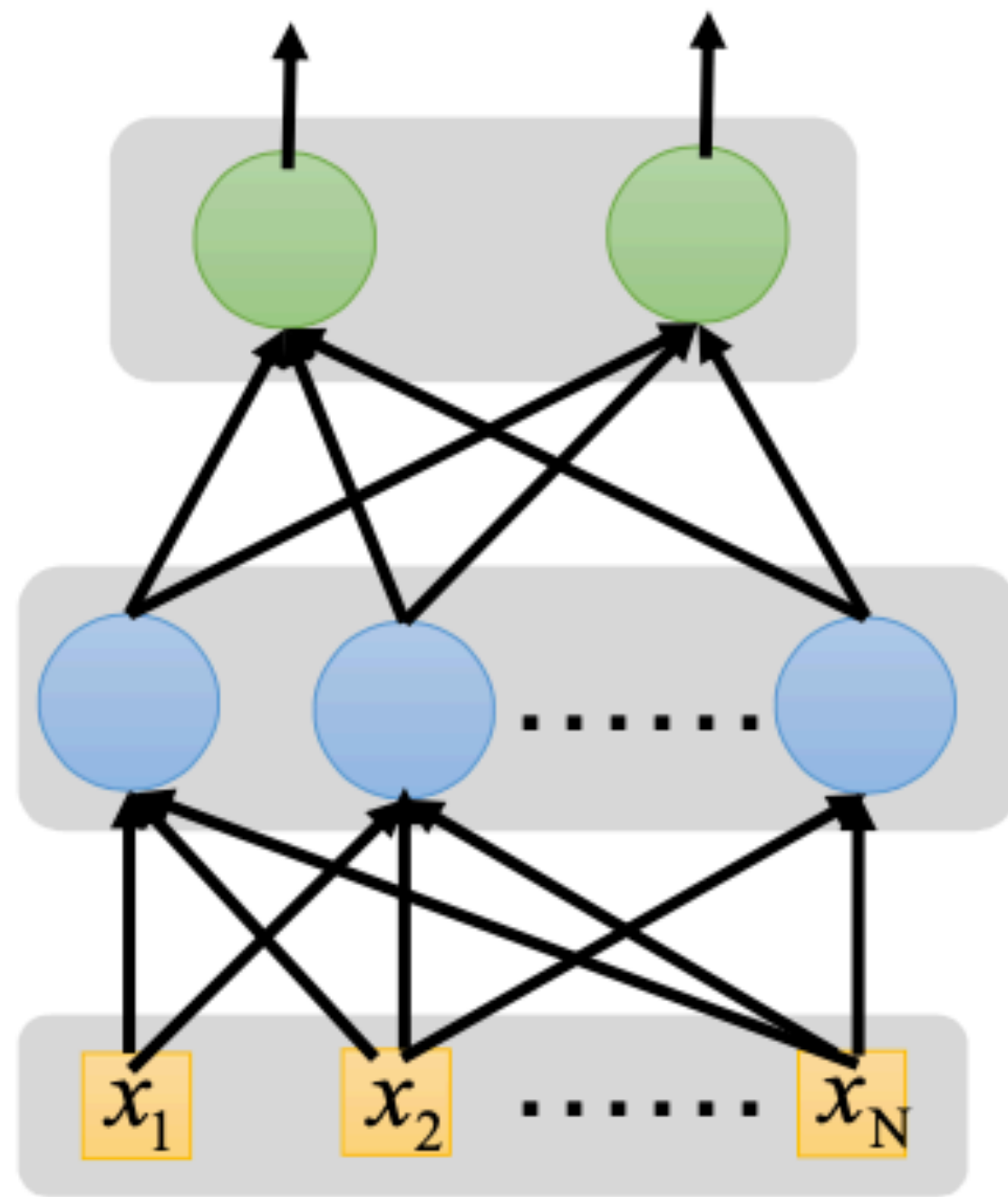
Big Data



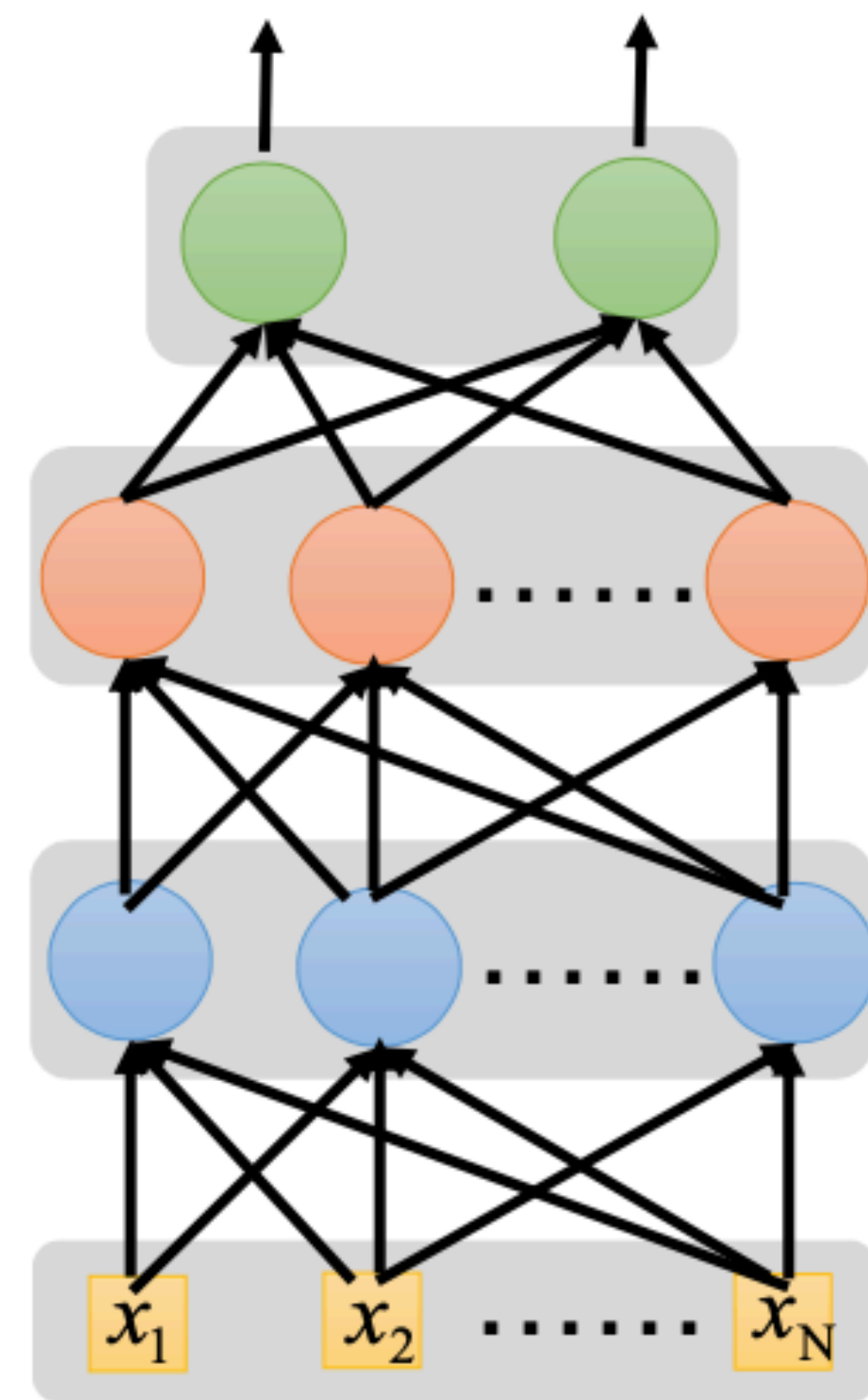
GPU

22 Why Deeper is Better

Deeper \rightarrow More parameters



Shallow



Deep

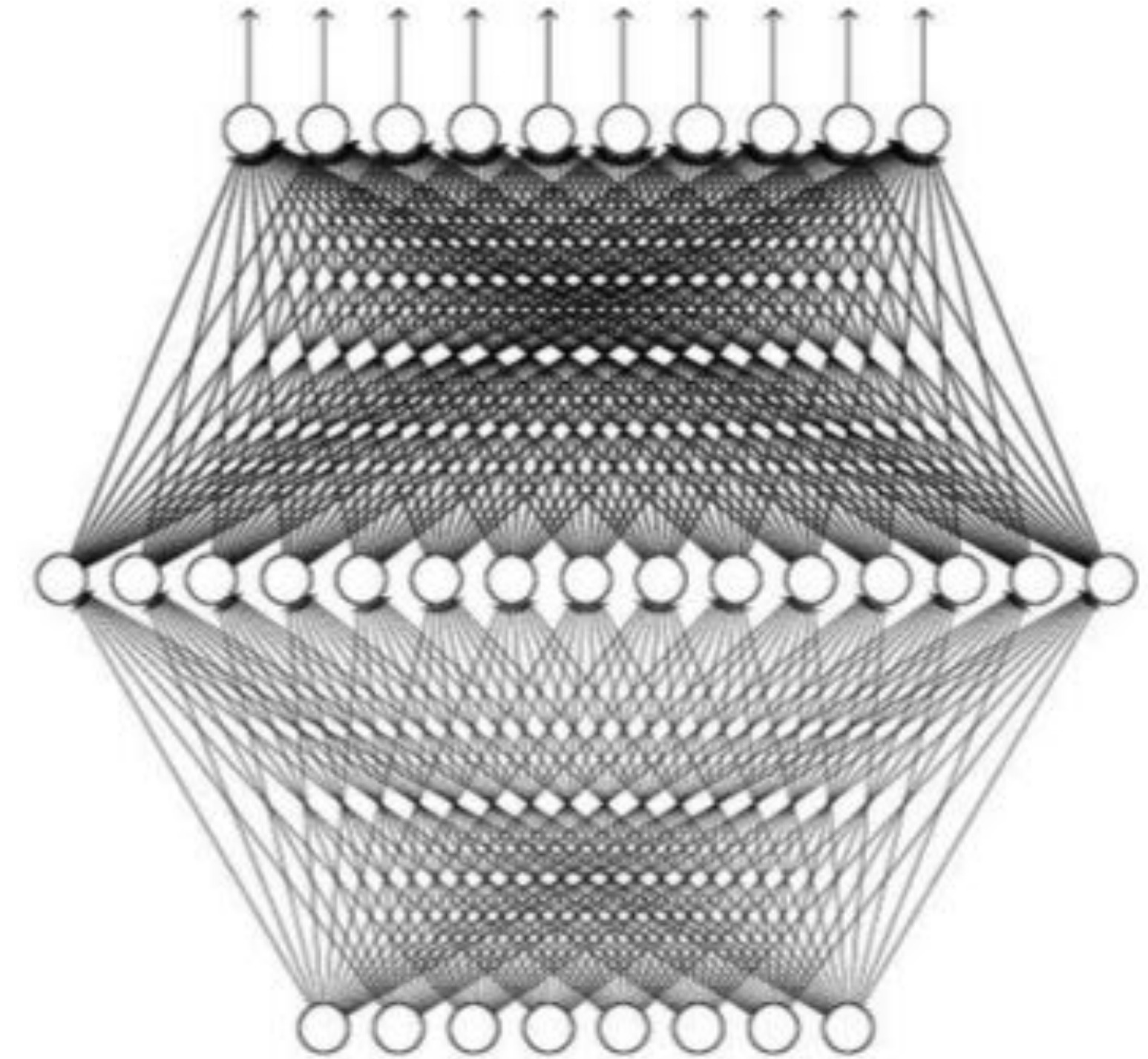
Universal Approximation Theorem

- Any continuous function f

$$f : \mathbb{R}^N \rightarrow \mathbb{R}^M$$

- can be realized by a network with one hidden layer

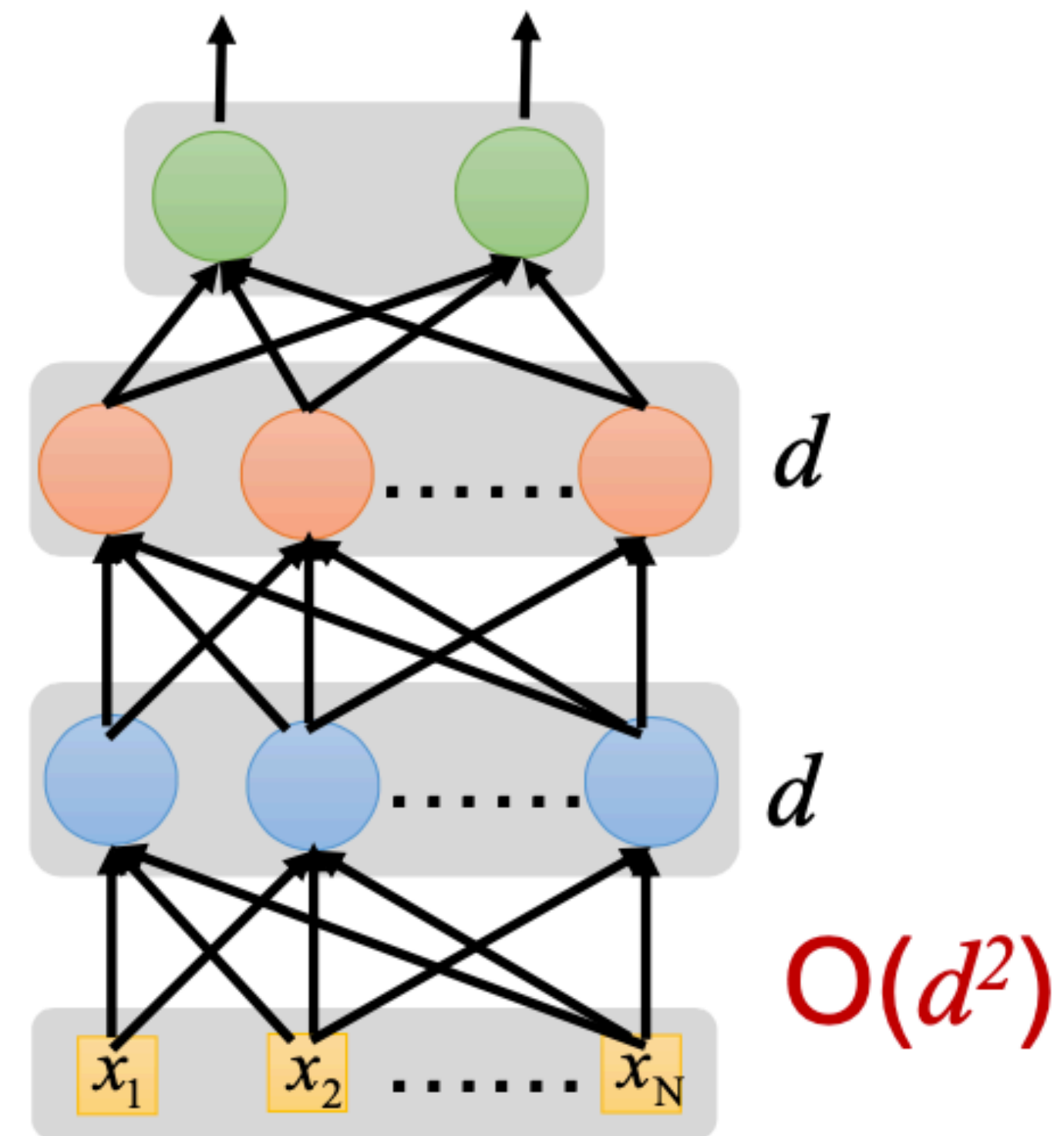
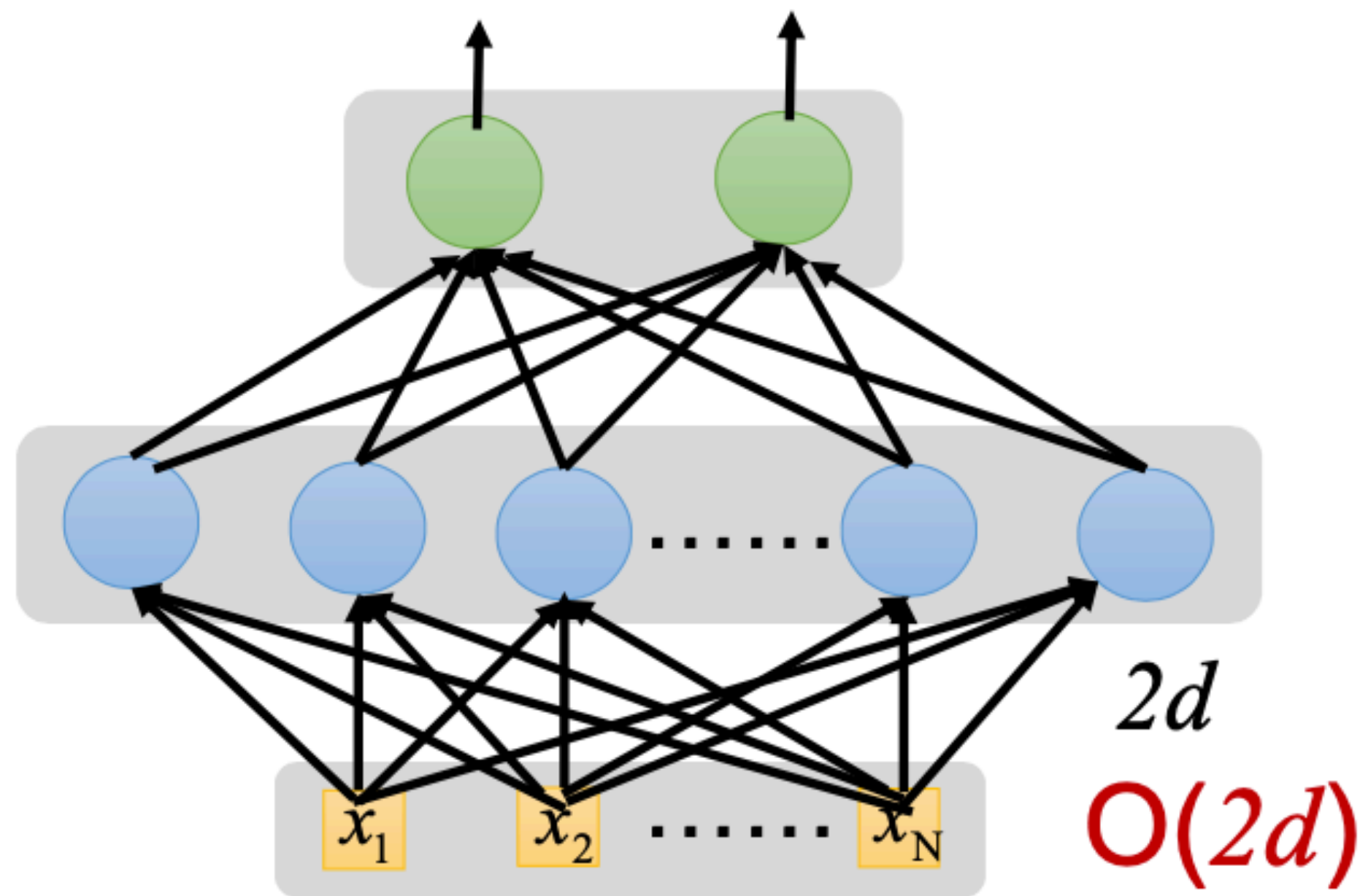
<http://neuralnetworksanddeeplearning.com/chap4.html>



Why “deep” not “fat” ?

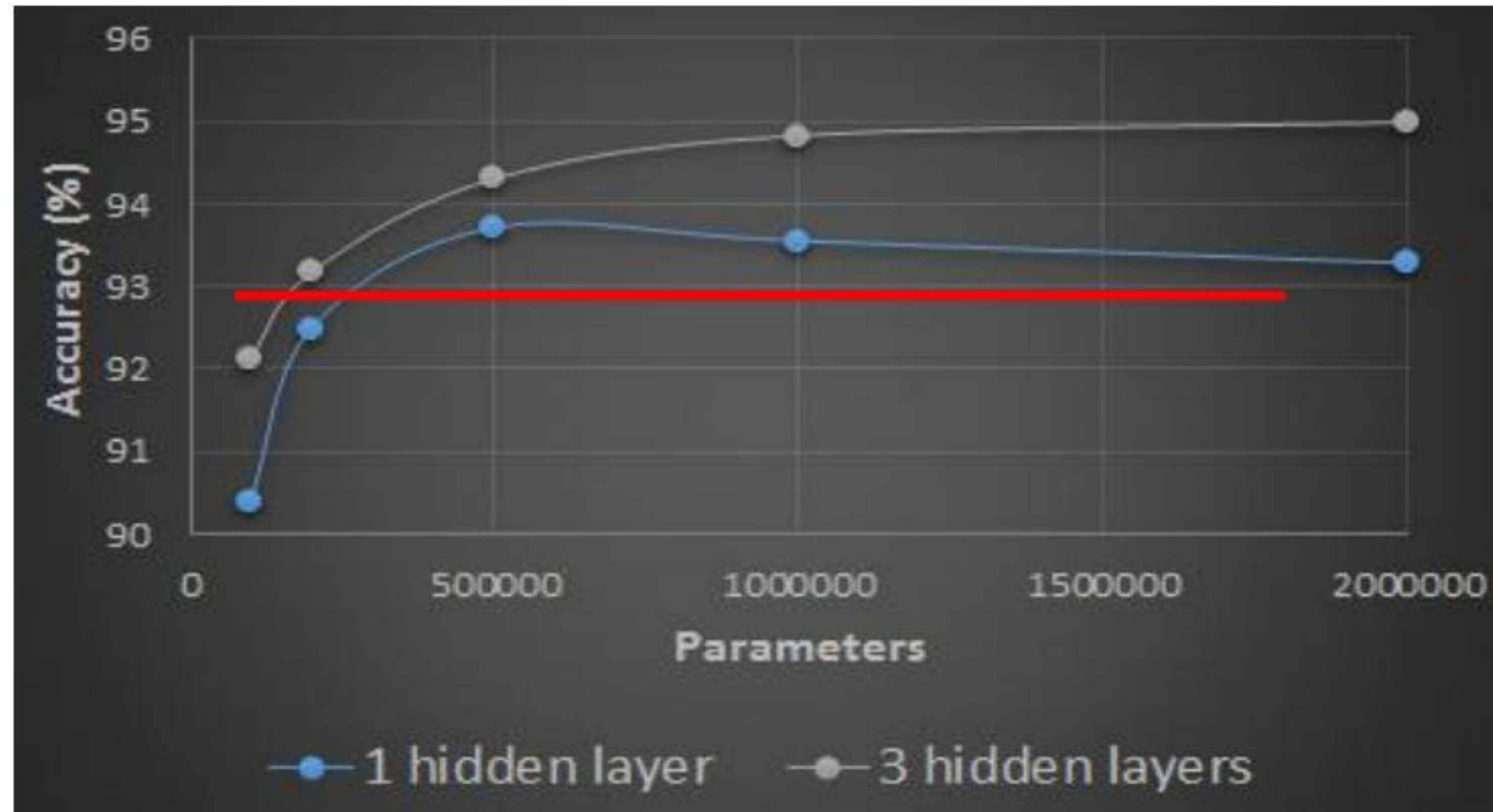
24 Fat Shallow v.s. Thin Deep

- Two networks with the same number of parameters
- same number of parameters, deep can model more complex function
 - Left is like “plus”, right is like “multiply”



25 Fat Shallow v.s. Thin Deep

- © E.g., Hand-Written Digit Classification



The deeper model uses less parameters to achieve the same performance

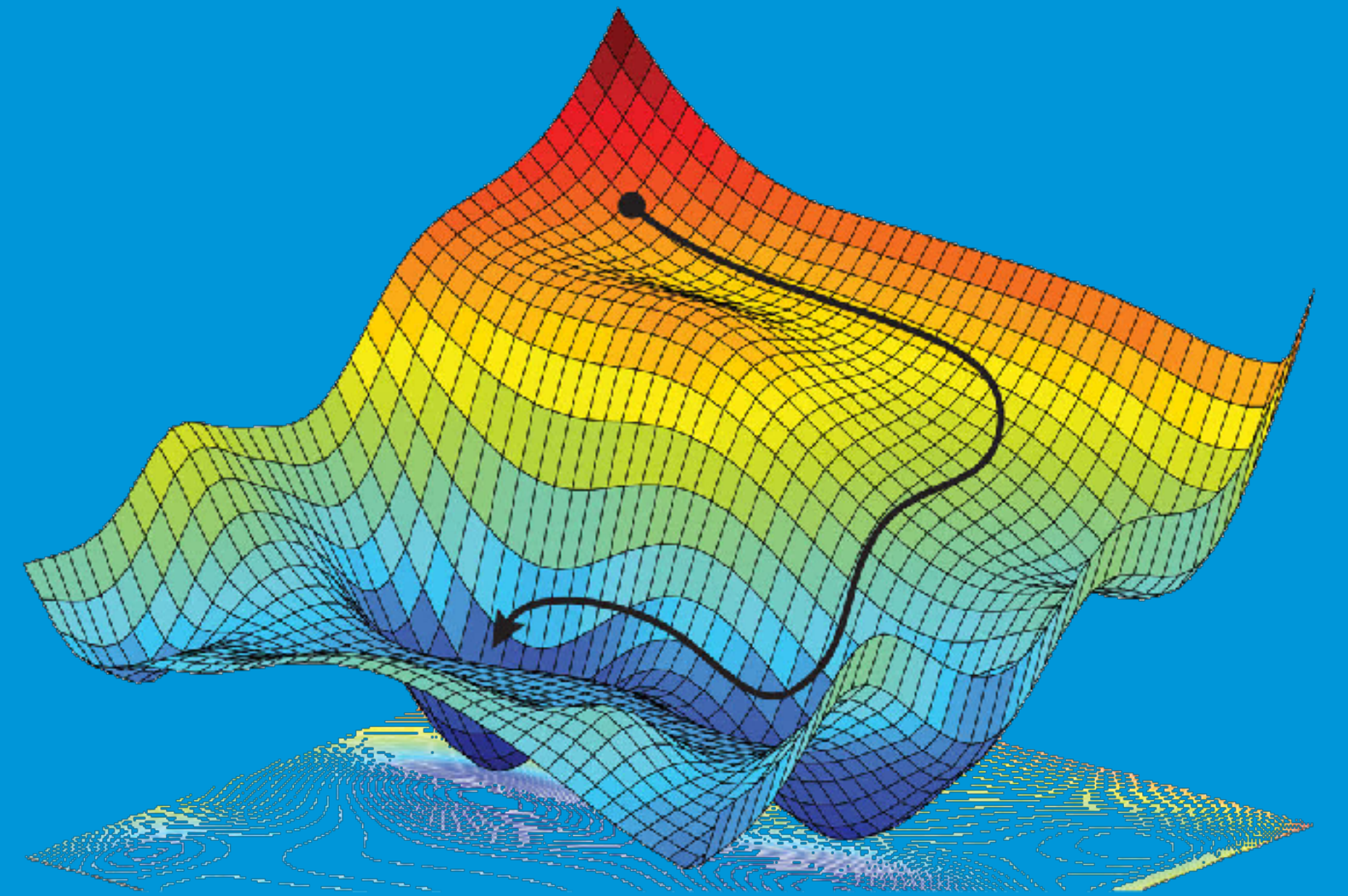
26 How to Frame the Learning Problem

- The **learning algorithm** f is to **map** the input domain X to output domain Y

$$f : X \rightarrow Y$$

- **Input domain:** word, word sequence, image, audio, video, click logs, brain signal ...
- **Output domain:** single label, tag sequence, tree structure, probabilistic distribution ...

Training Deep Neural Networks by Gradient Descent



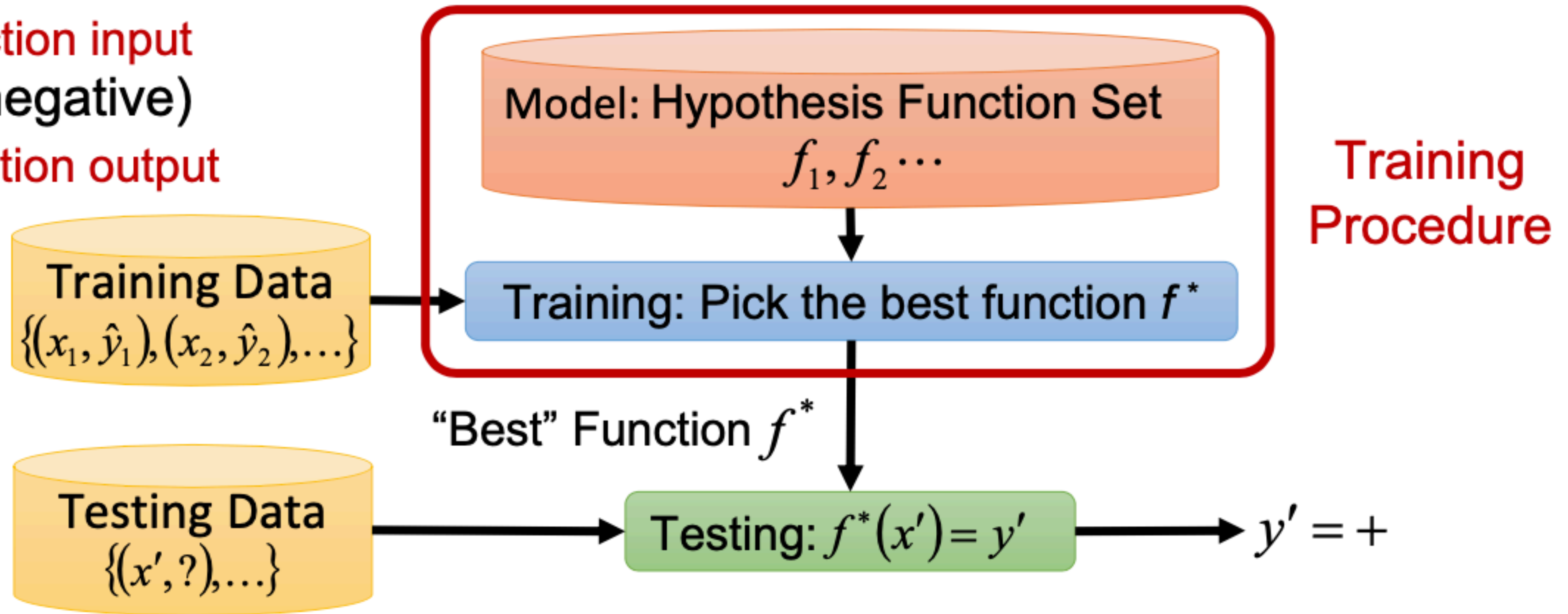
Machine Learning Framework

x : "It claims too much."

function input

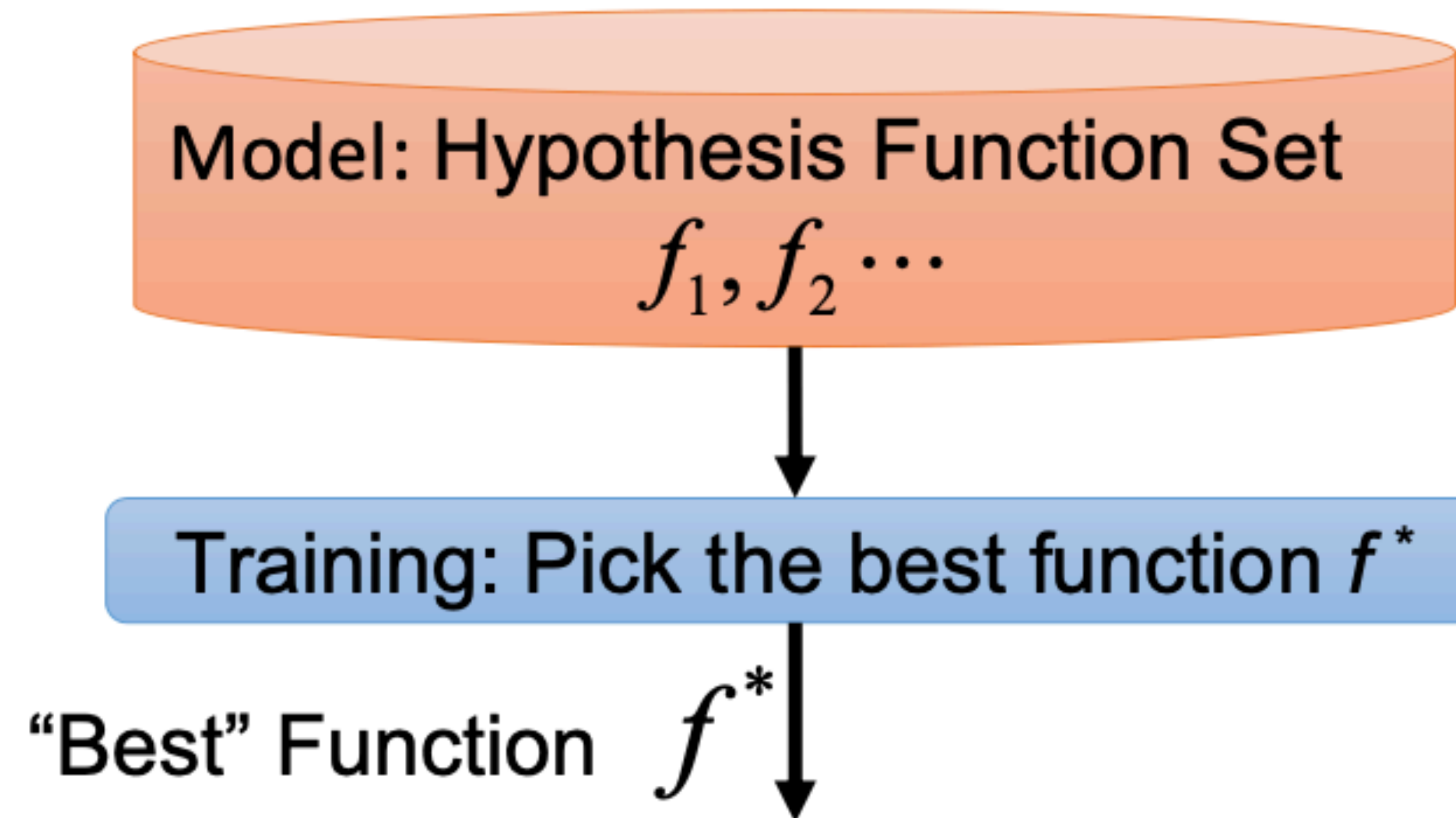
\hat{y} : - (negative)

function output



Training is to pick the best function given the observed data
 Testing is to predict the label using the learned function

Training Procedure



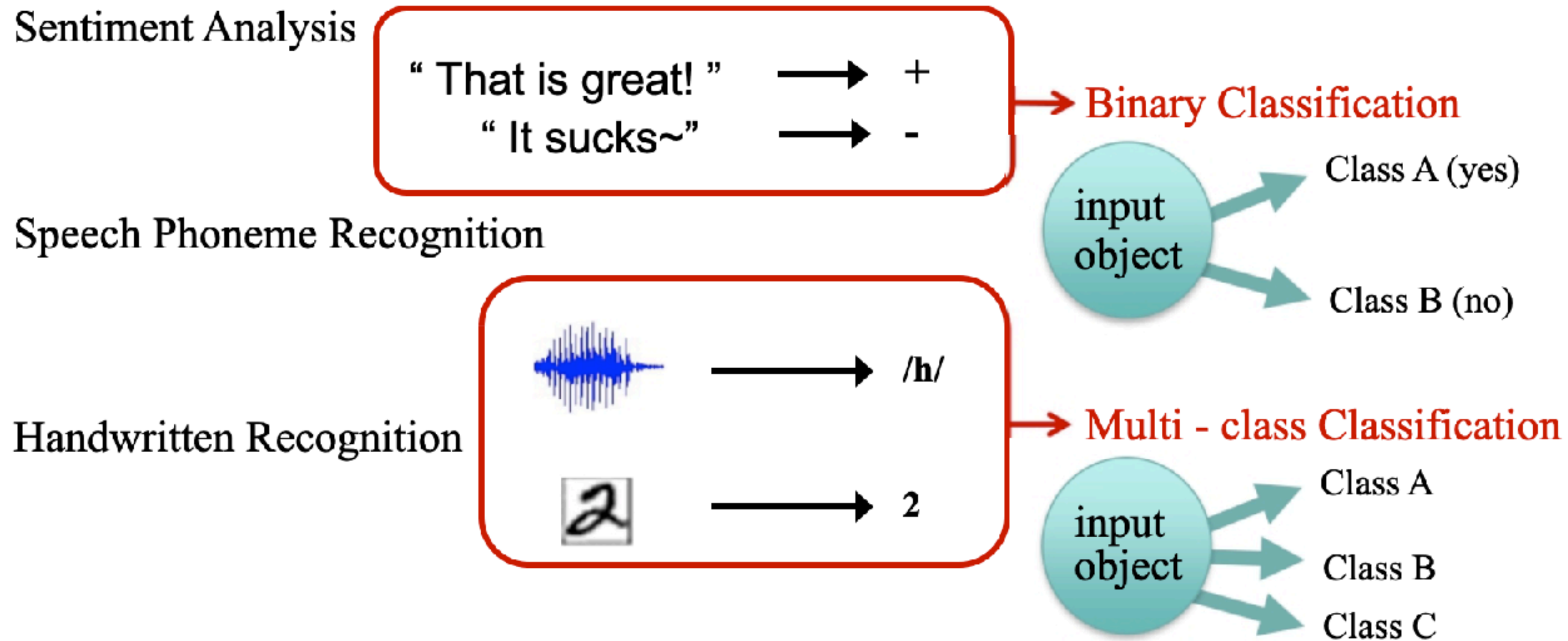
- Q1: What is the **model**? (function hypothesis set)
- Q2: What does a **good function** mean?
- Q3: How do we **pick** the “best” function?

What is the model?

Neural Networks

Example: Classification Tasks

- Many problems can be formulated as a classification problem



Some cases are not easy to be formulated as classification problems

Target Function

Classification Task

$$f(x) = y \quad \longrightarrow \quad f : R^N \rightarrow R^M$$

- x : input object to be classified
- y : class/label

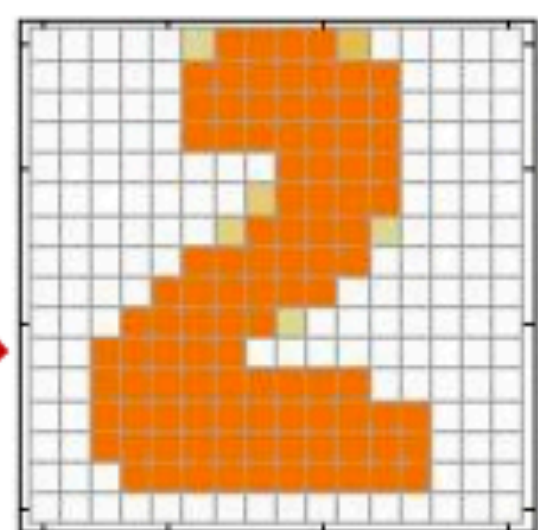
- a N -dim vector
- a M -dim vector

Assume both x and y can be represented as fixed-size vectors

Vector Representation Example

Handwriting Digit Classification

x: image



16 x 16

Each pixel corresponds to an element in the vector

$\begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$

1: for ink
0: otherwise

16 x 16 = 256 dimensions

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

y: class/label

10 dimensions for digit recognition

“1”

$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$

“1”
“2”
“3”

“2”

$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$

“1” → “1” or not
“2” → “2” or not
“3” → “3” or not

Vector Representation Example

Sentiment Analysis

x: word

“love” Each element in the vector corresponds to a word in the vocabulary



$$\begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$$

1: indicates the word
0: otherwise
dimensions = size of vocab

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

y: class/label

3 dimensions
(positive, negative, neutral)

“+”



1

“+”

0

“-”

0

“?”

⋮

“-”



0

1

0

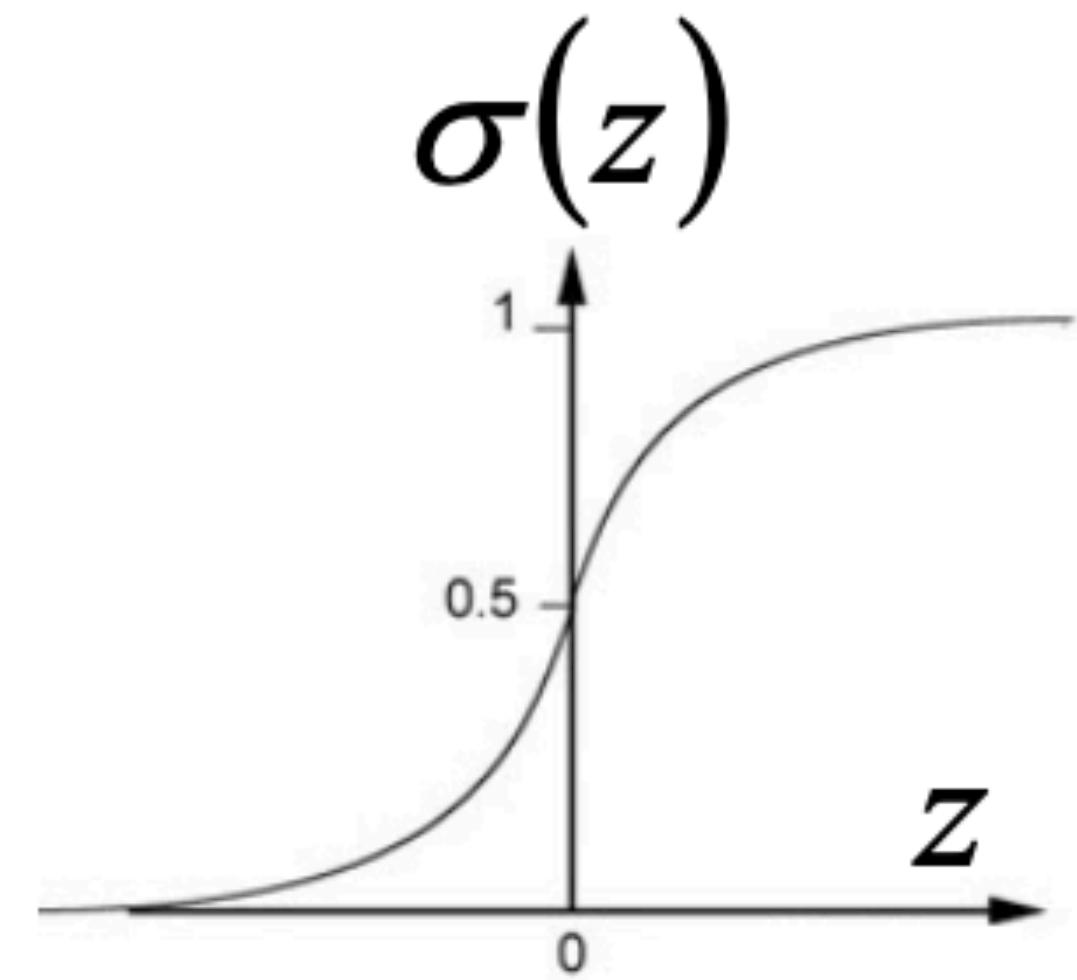
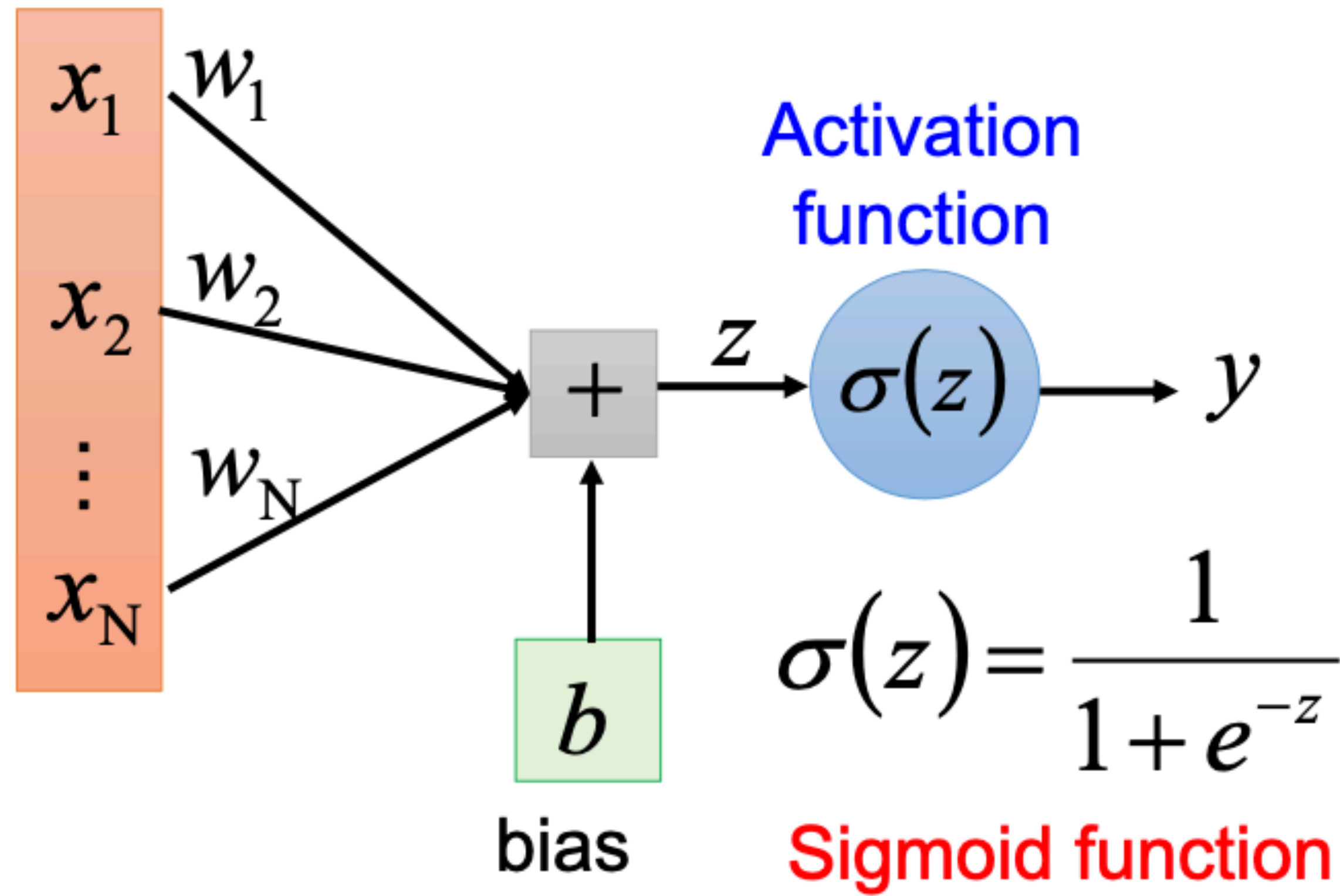
⋮

“+” → “+” or not

“-” → “-” or not

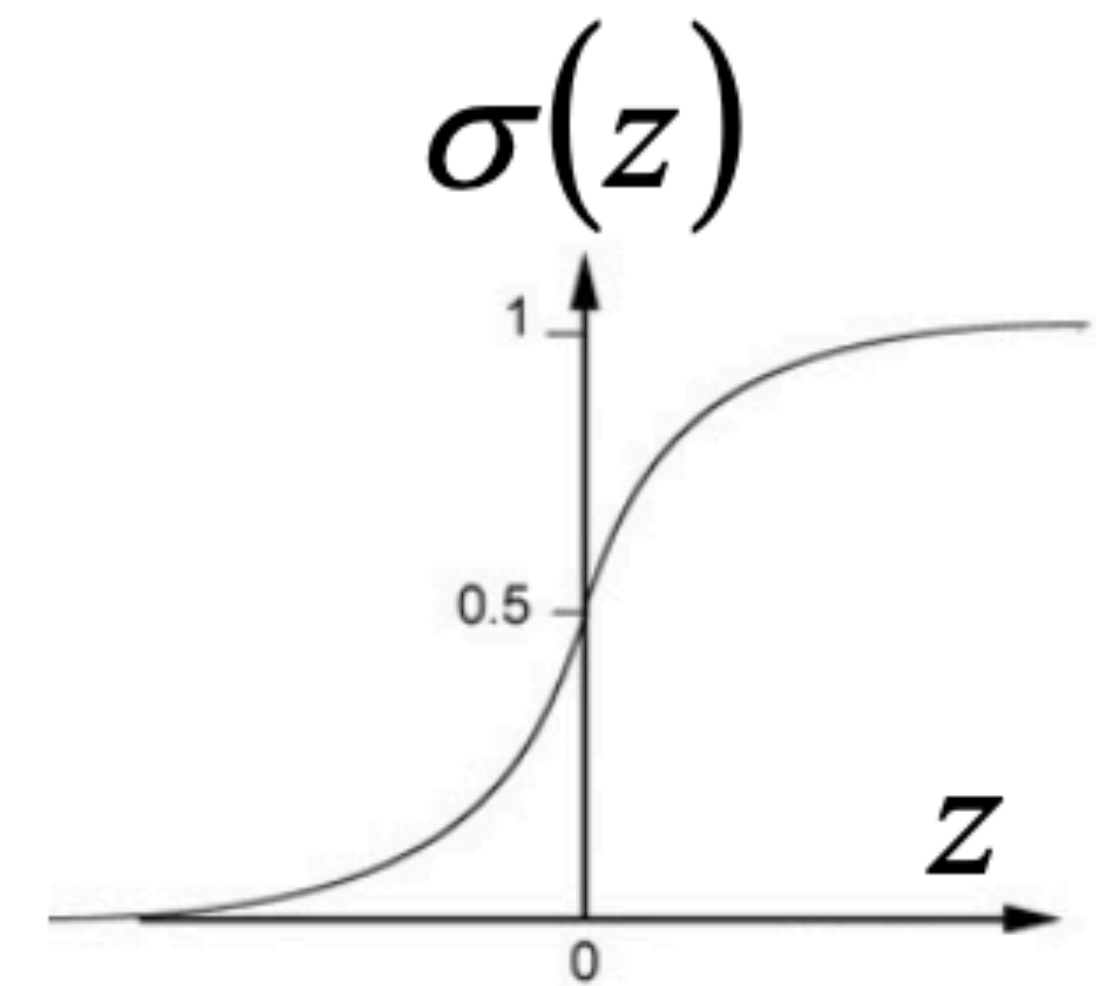
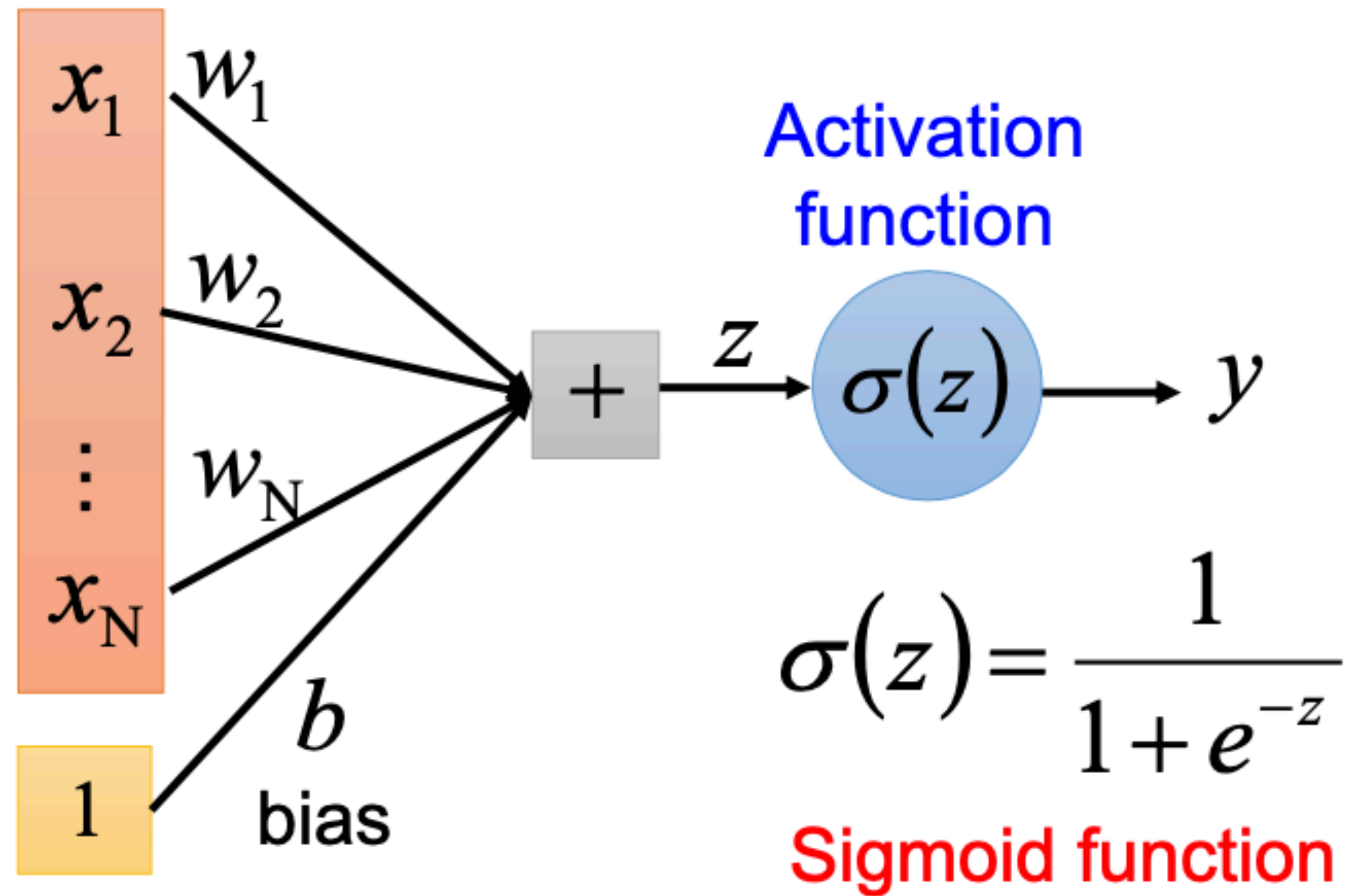
“?” → “?” or not

A Single Neuron



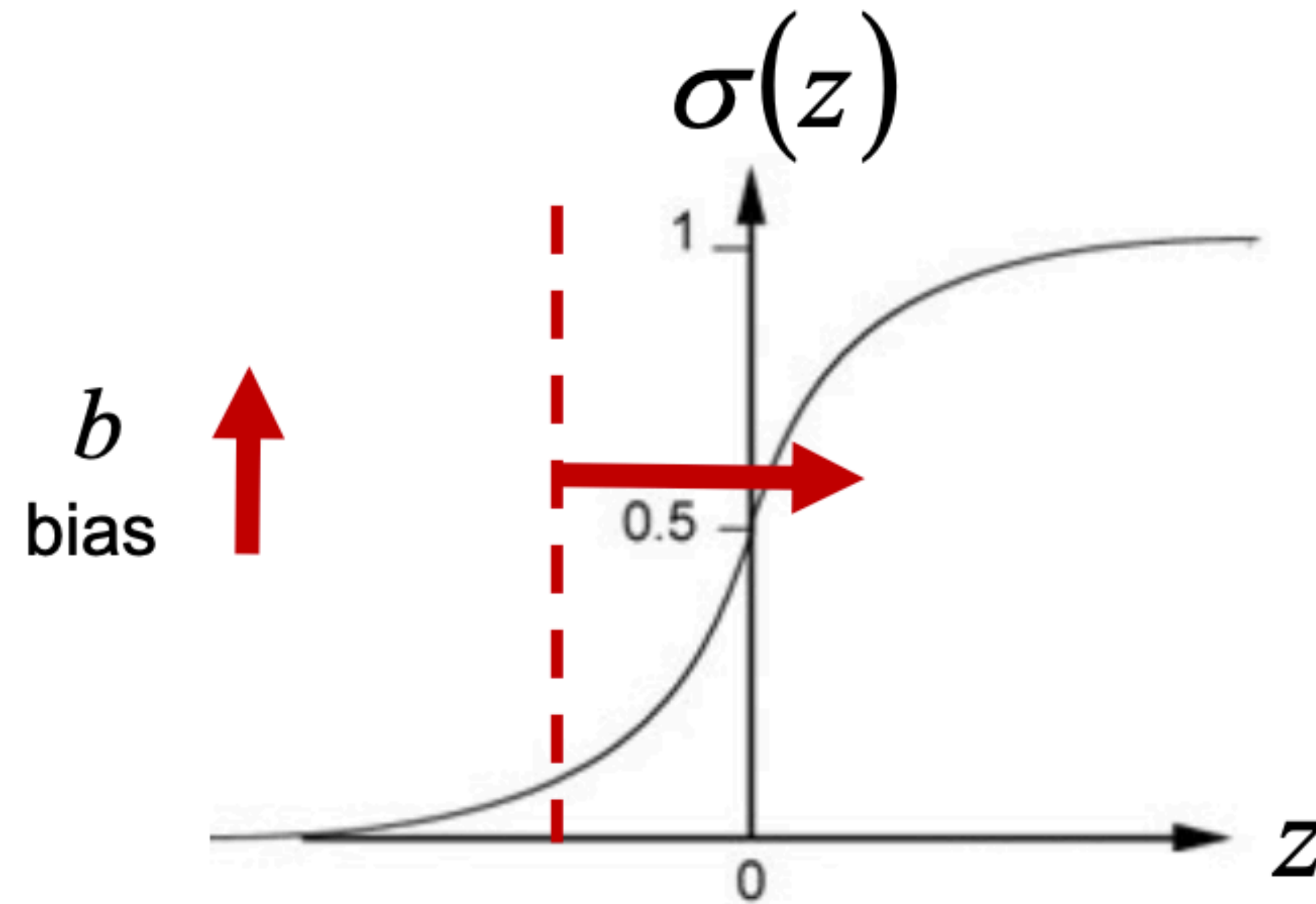
Each neuron is a very simple function

A Single Neuron



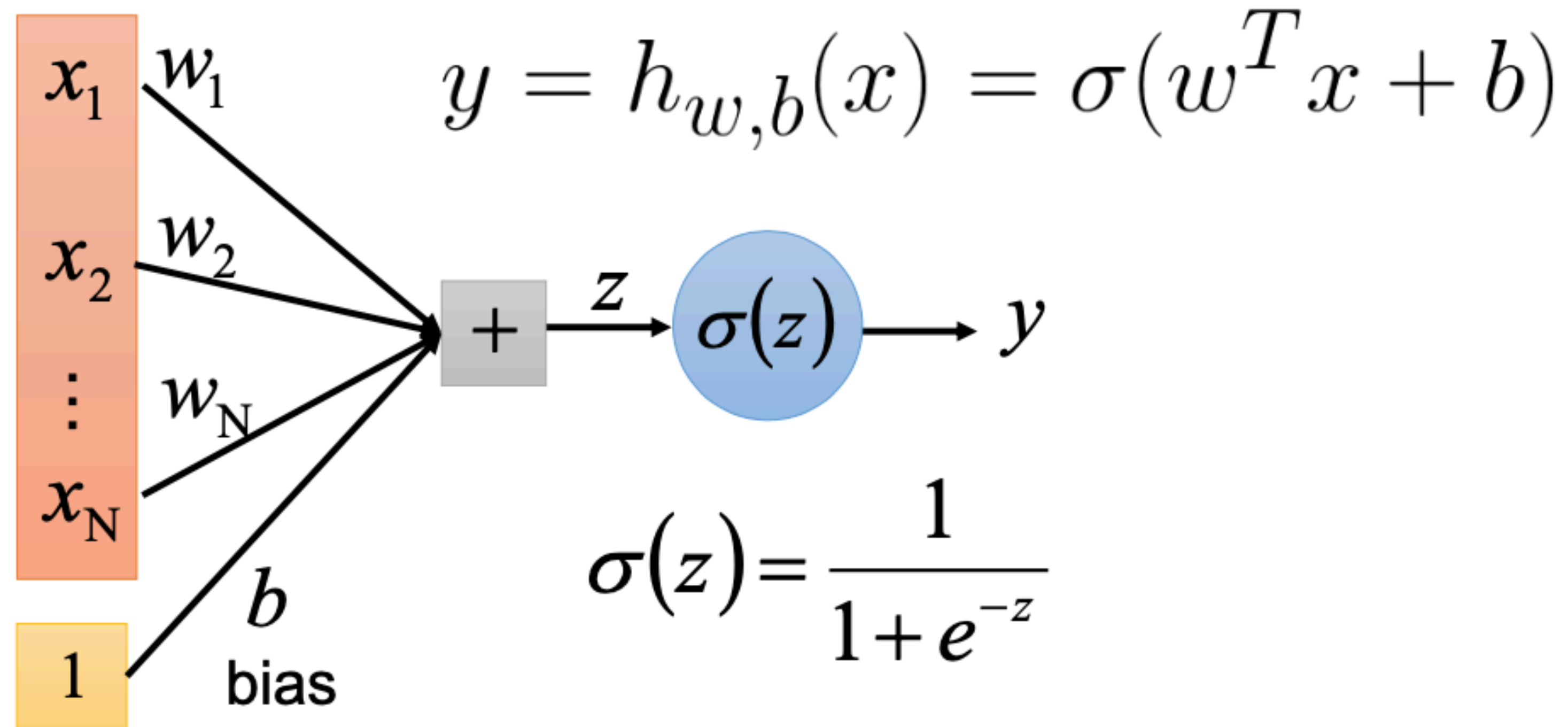
The bias term is an “always on” feature

37 Why Bias



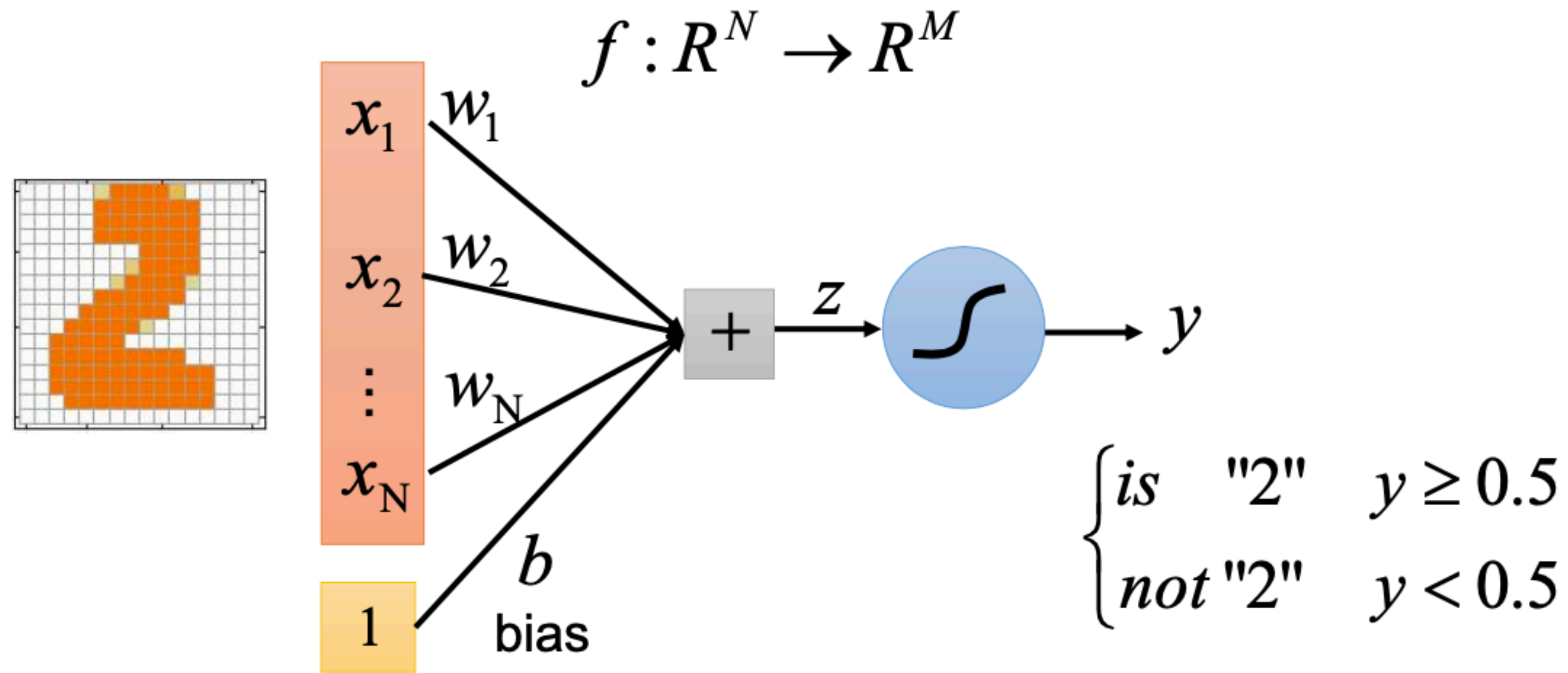
The bias term gives a class prior

38 Model Parameters of a Single Neuron



w, b are the parameters of this neuron

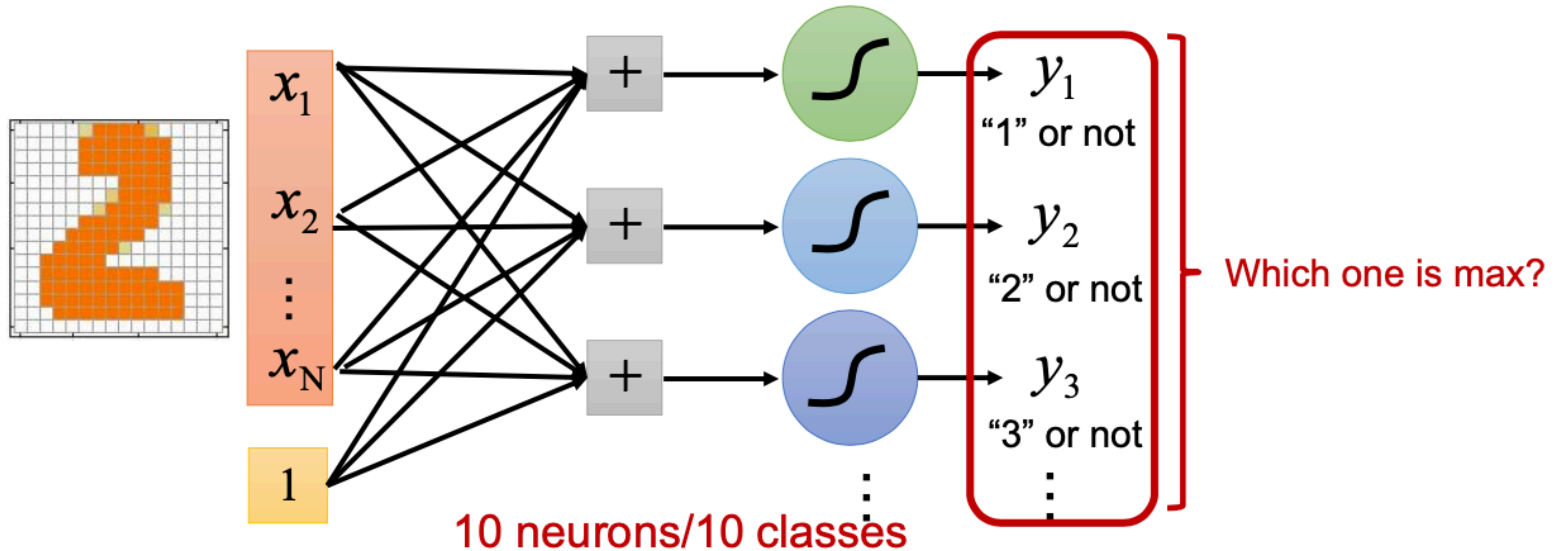
A Single Neuron



A single neuron can only handle binary classification

A Layer of Neurons

Handwriting digit classification $f : R^N \rightarrow R^M$

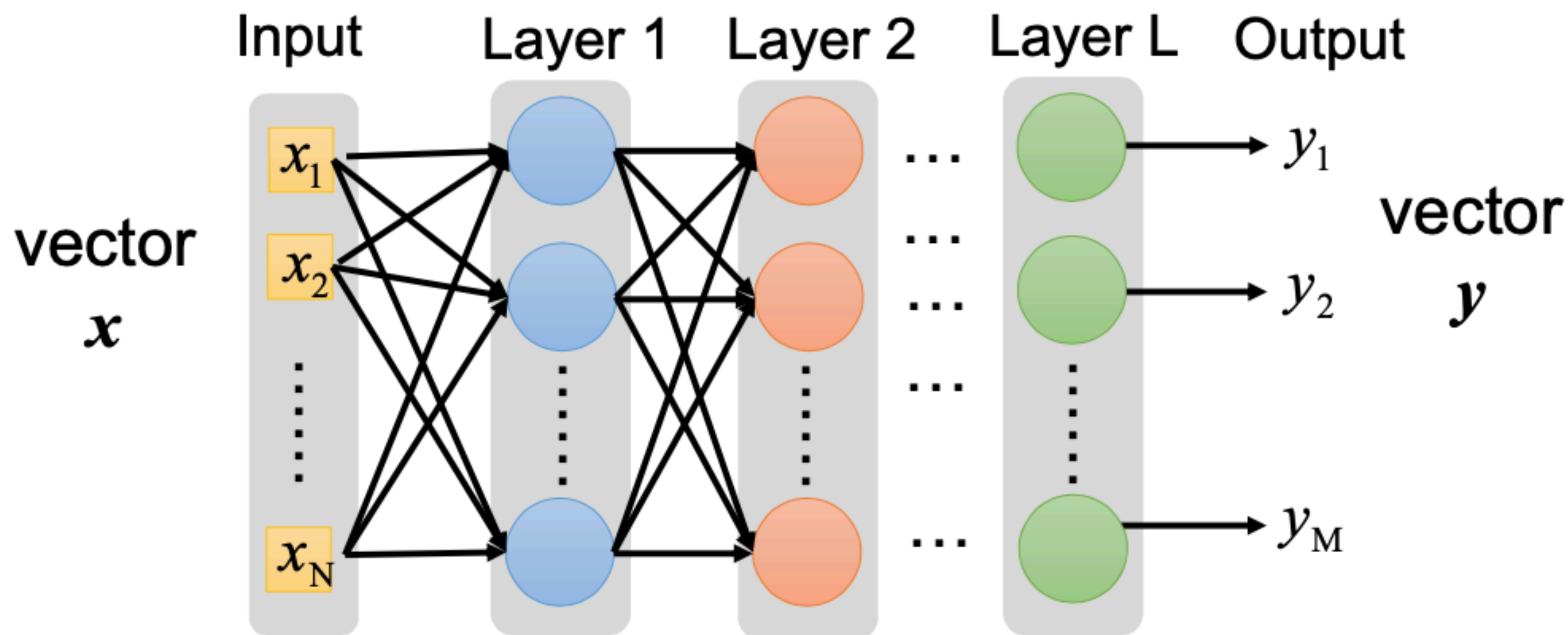


A layer of neurons can handle multiple possible output, and the result depends on the max one

41 Deep Neural Networks (DNN)

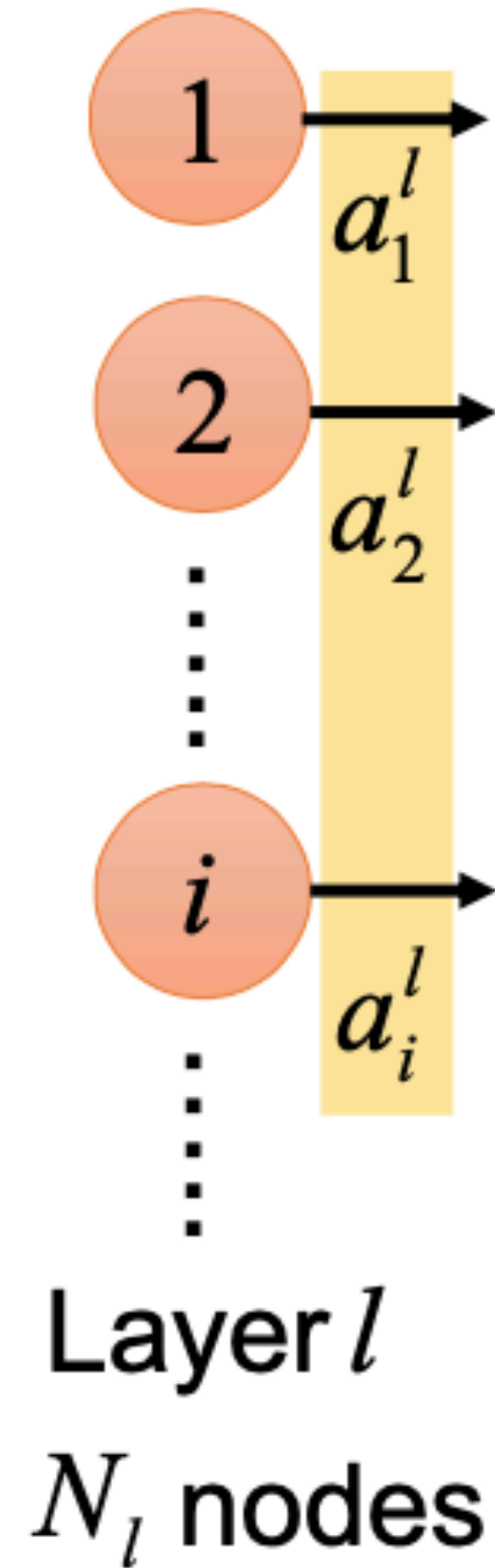
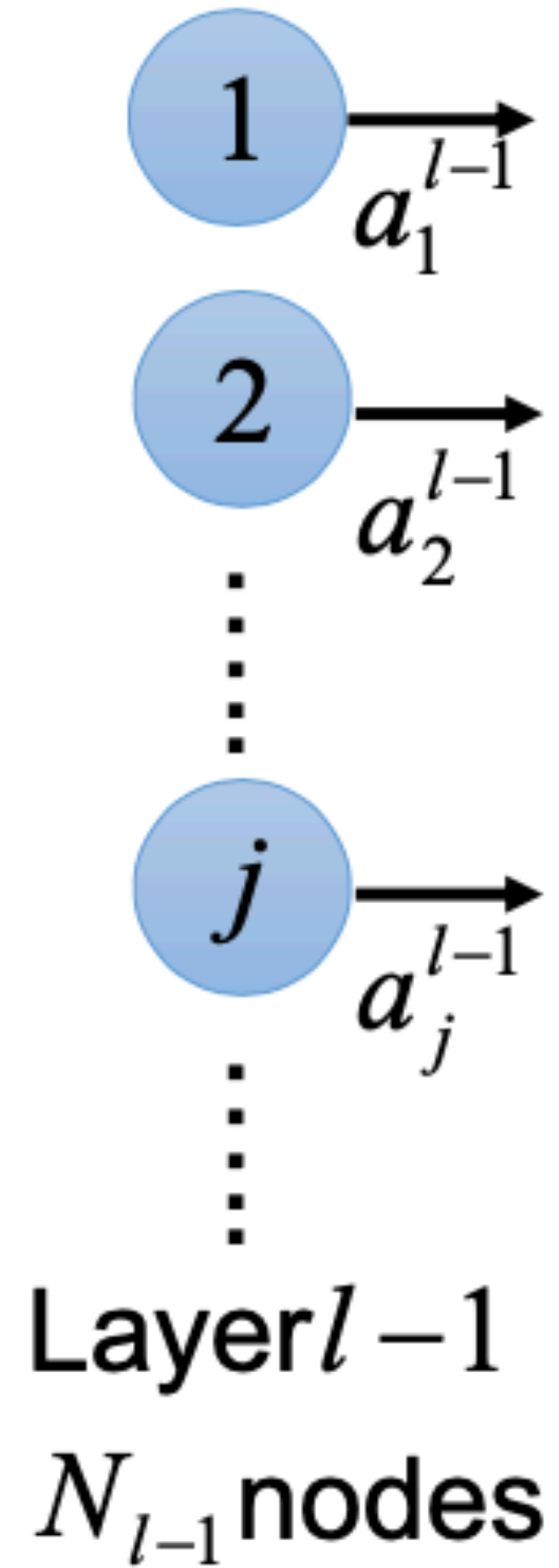
Fully connected feedforward network

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



Deep NN: multiple hidden layers

42 Notation Definition



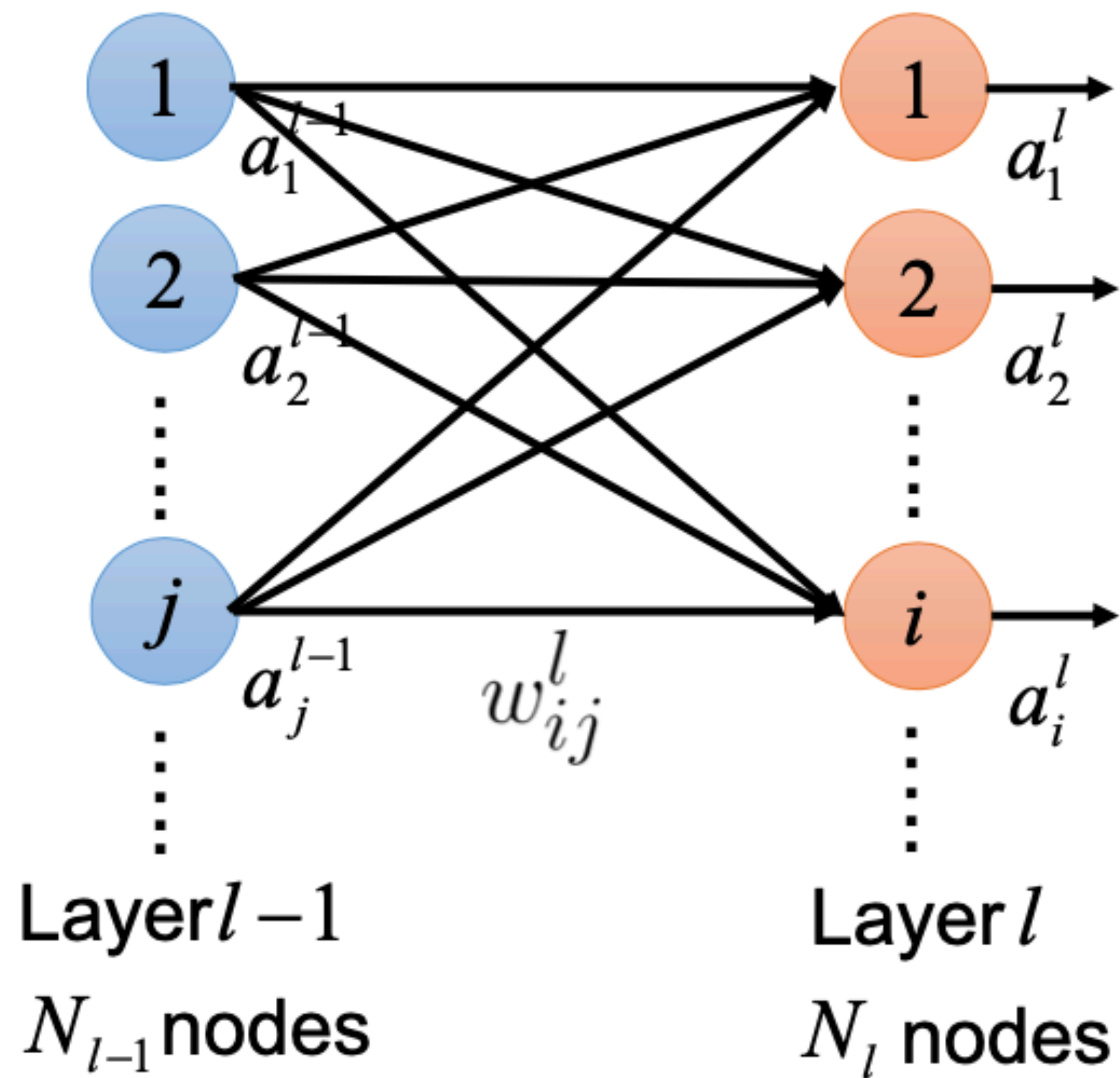
Output of a neuron:

a_i^l → layer l
 a_i^l → neuron i

$$a^l = \begin{bmatrix} \vdots \\ a_i^l \\ \vdots \end{bmatrix}$$

output of one layer → a vector

Notation Definition

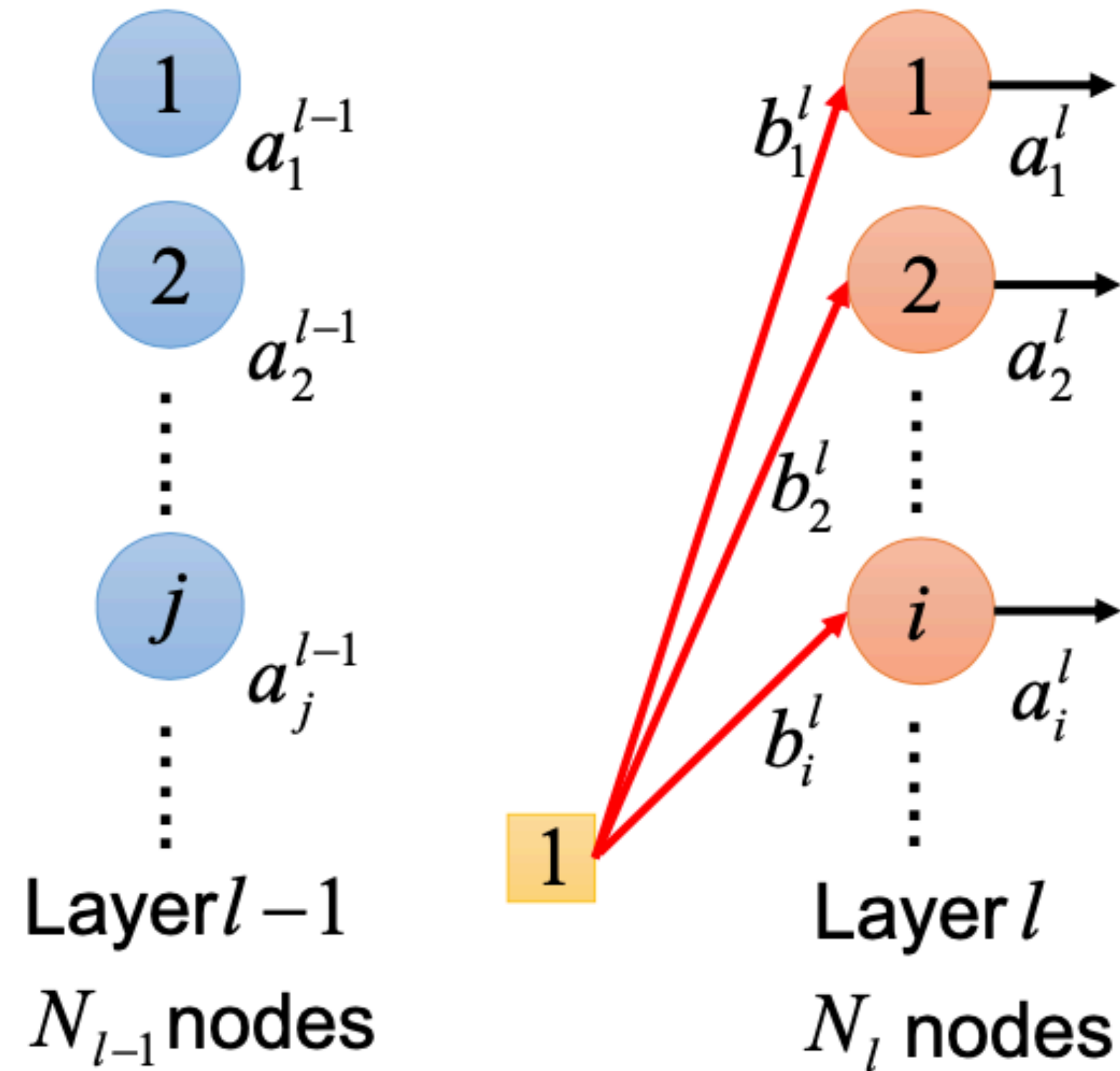


w_{ij}^l → layer $l-1$ to layer l
 → from neuron j (layer $l-1$) to neuron i (layer l)

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & & \ddots \end{bmatrix} \begin{matrix} \leftarrow N_{l-1} \rightarrow \\ \uparrow N_l \downarrow \end{matrix}$$

weights between two layers
 → a matrix

44 Notation Definition

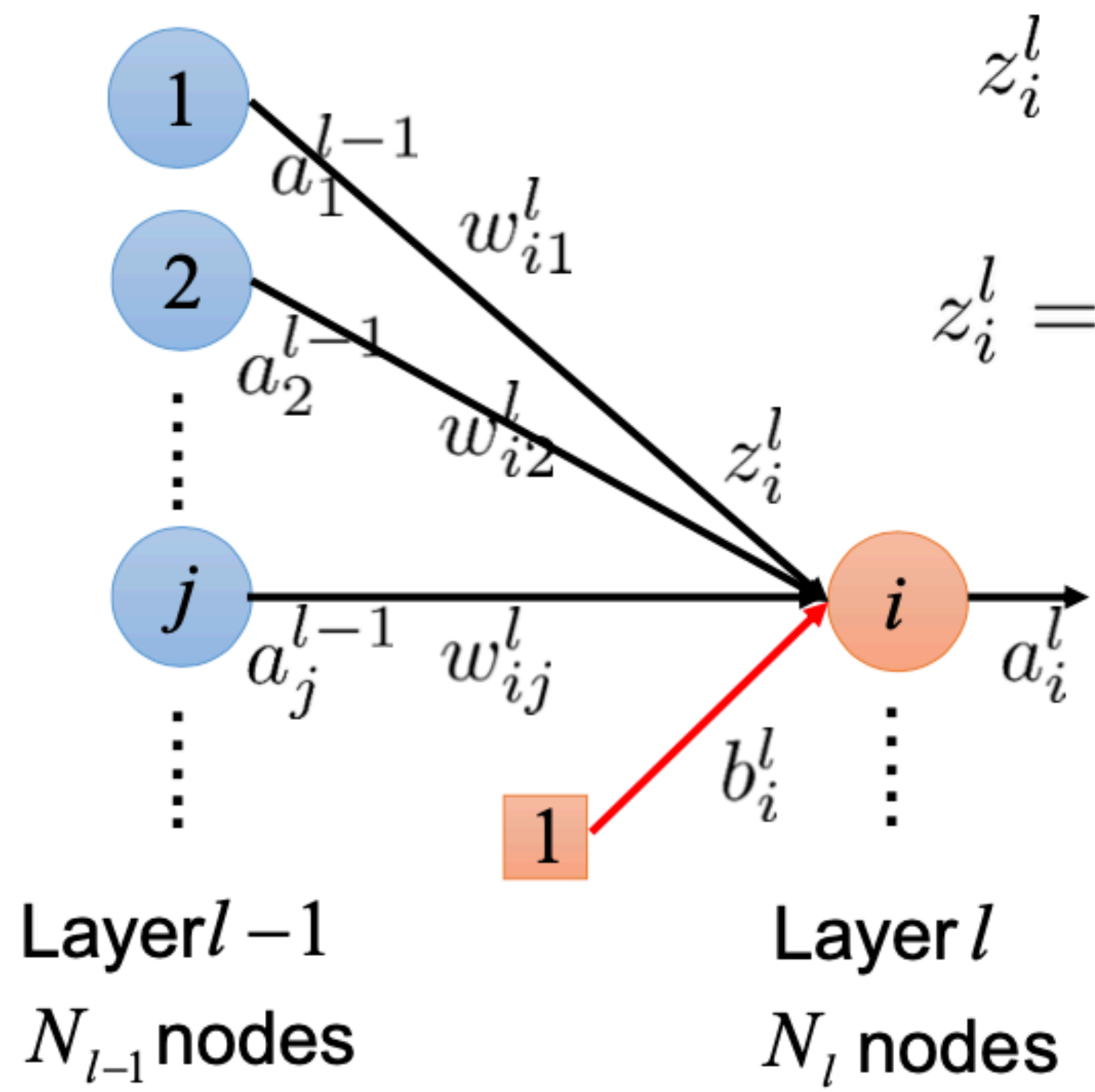


b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias of all neurons at each layer
→ a vector

Notation Definition



z_i^l : input of the activation function for neuron i at layer l

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \dots + b_i^l$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$

$$z^l = \begin{bmatrix} \vdots \\ z_i^l \\ \vdots \end{bmatrix}$$

activation function input at each layer \rightarrow a vector

46 Notation Summary

a_i^l : output of a neuron

a^l : output vector of a layer

z_i^l : input of activation function

z^l : input vector of activation function
for a layer

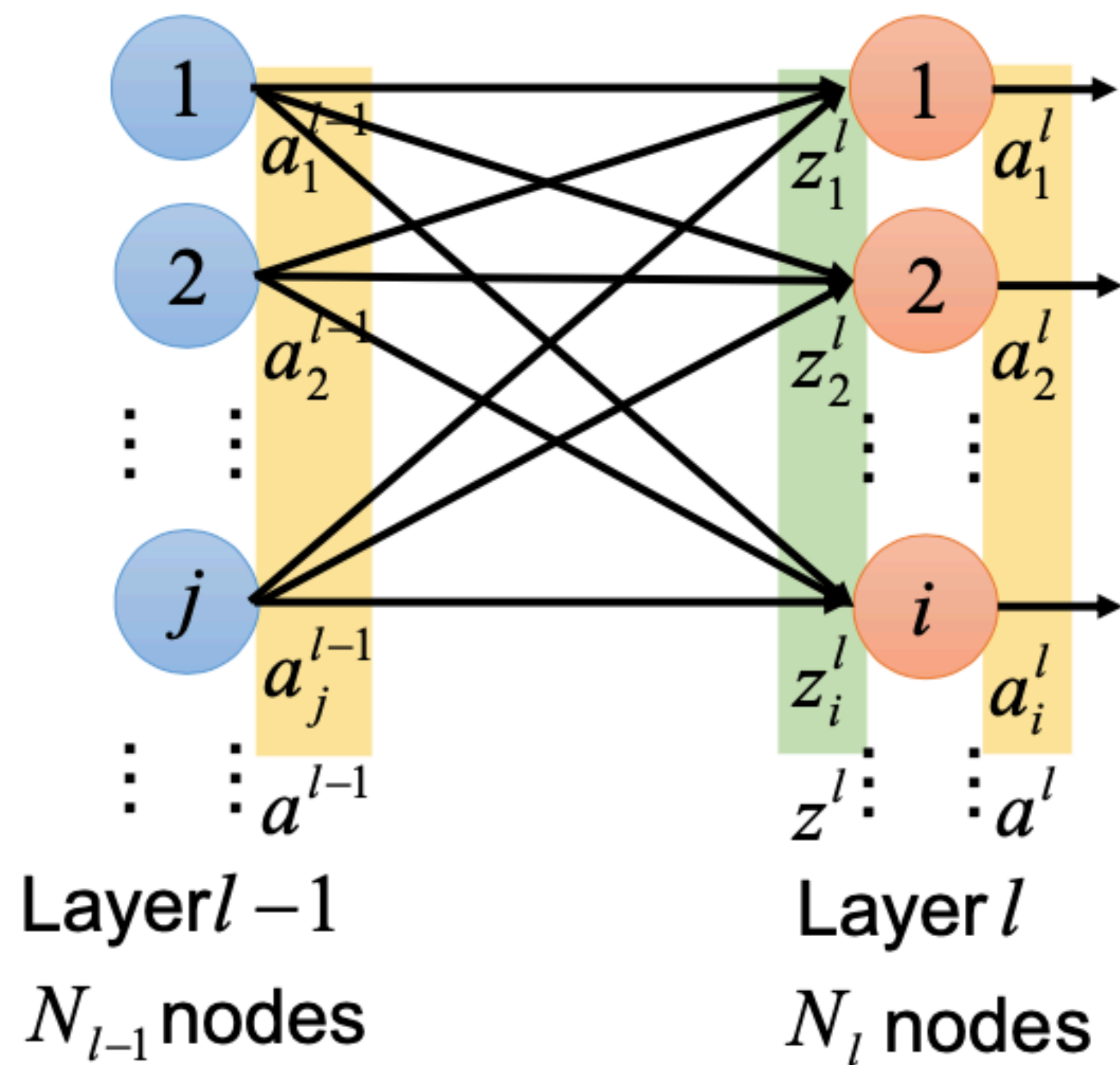
w_{ij}^l : a weight

W^l : a weight matrix

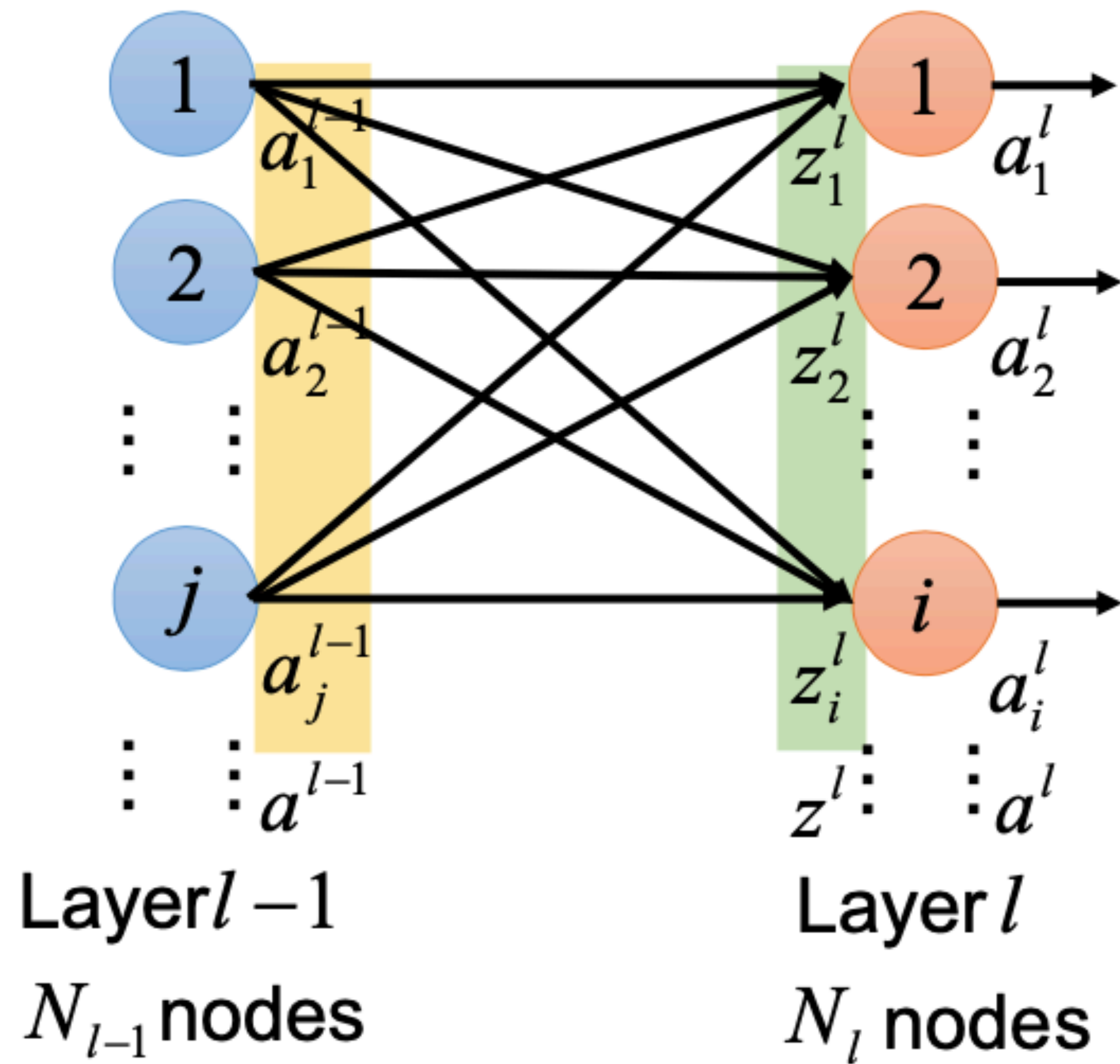
b_i^l : a bias

b^l : a bias vector

47 Layer Output Relation



Layer Output Relation: from a to z



$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \dots + b_1^l$$

$$\vdots$$

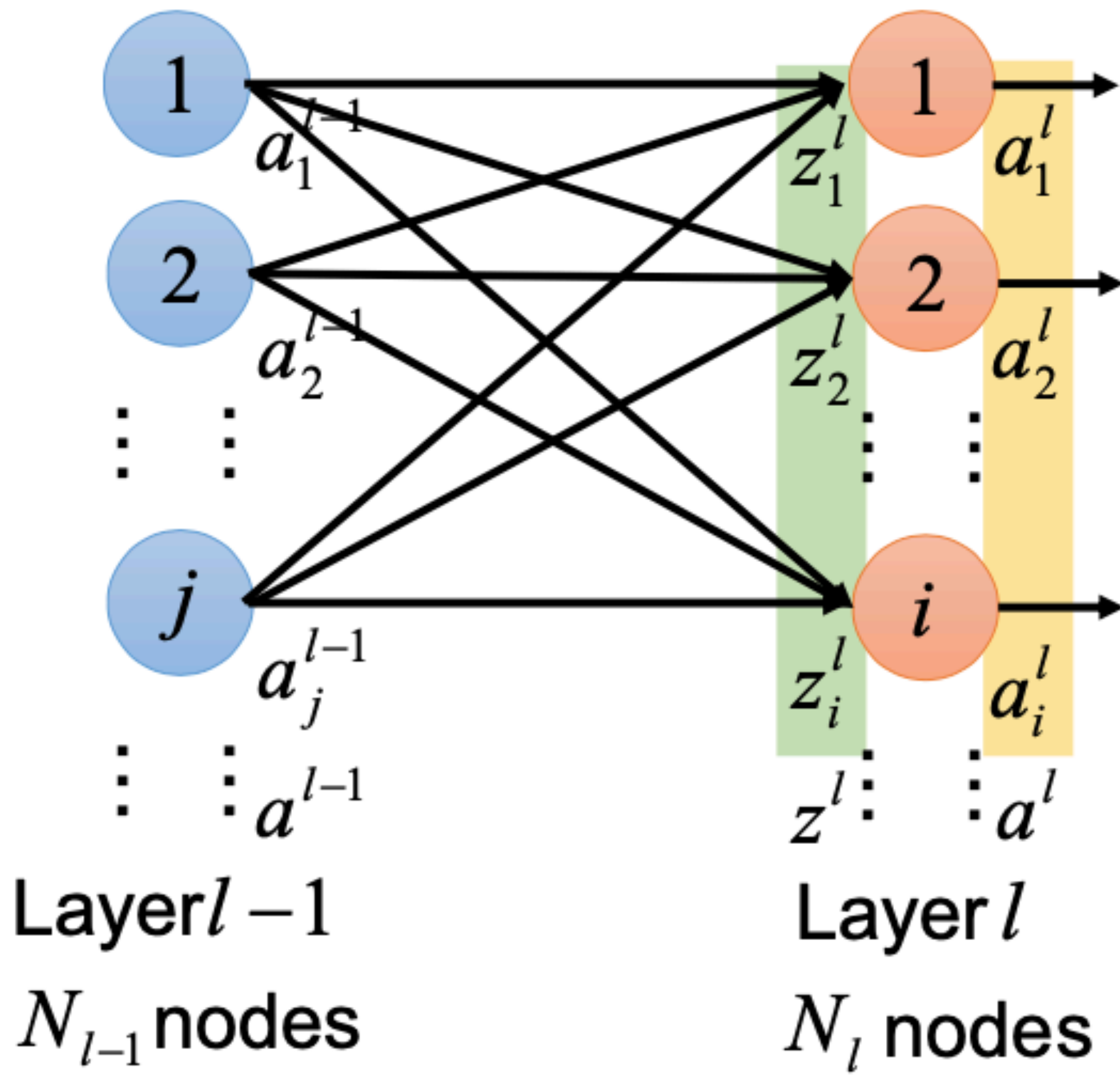
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \dots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \dots \\ \vdots & & \dots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

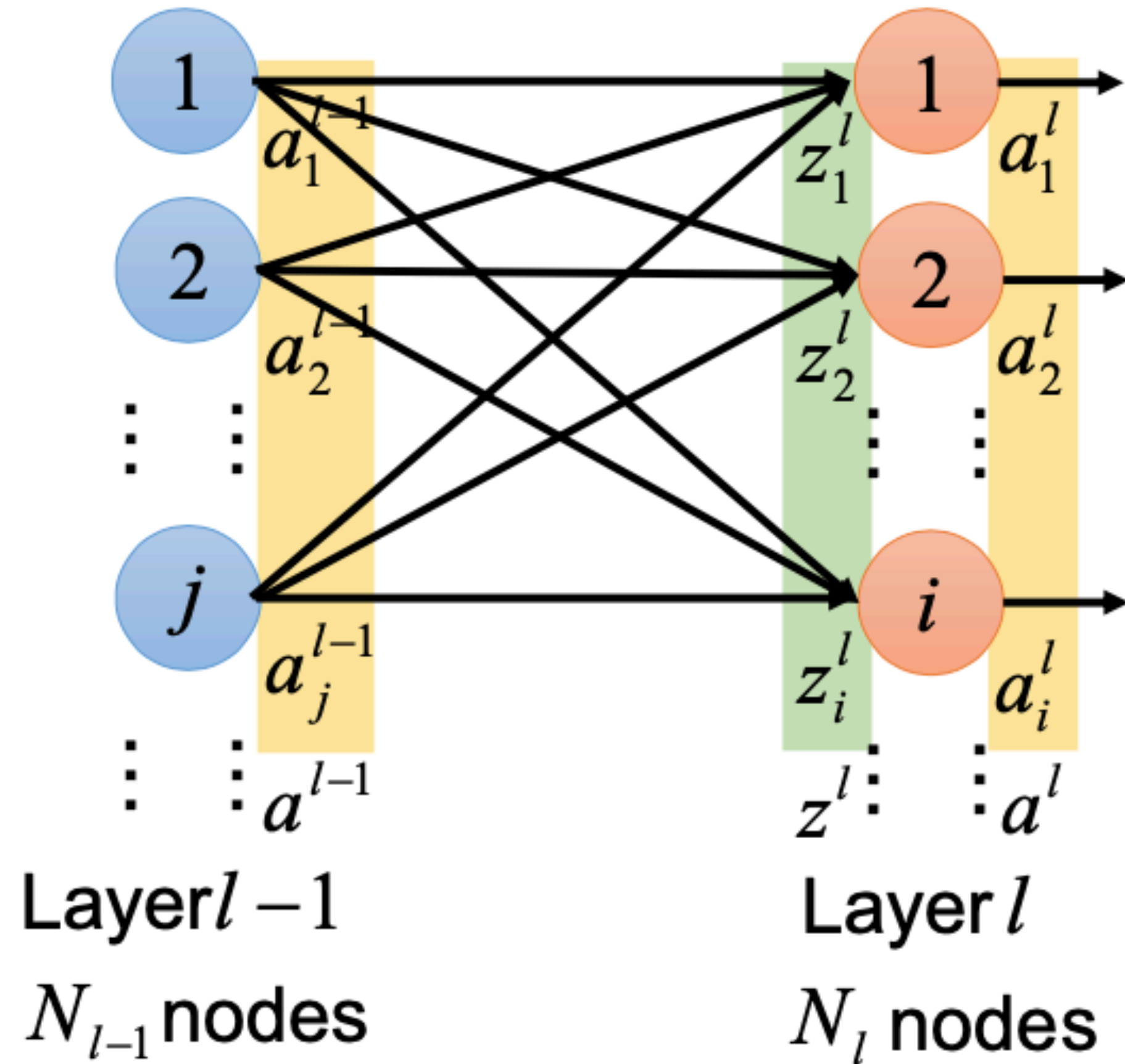
49 Layer Output Relation: from z to a



$$a_i^l = \sigma(z_i^l)$$
$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma(z_1^l) \\ \sigma(z_2^l) \\ \vdots \\ \sigma(z_i^l) \\ \vdots \end{bmatrix}$$

$$a^l = \sigma(z^l)$$

50 Layer Output Relation



$$z^l = W^l a^{l-1} + b^l$$

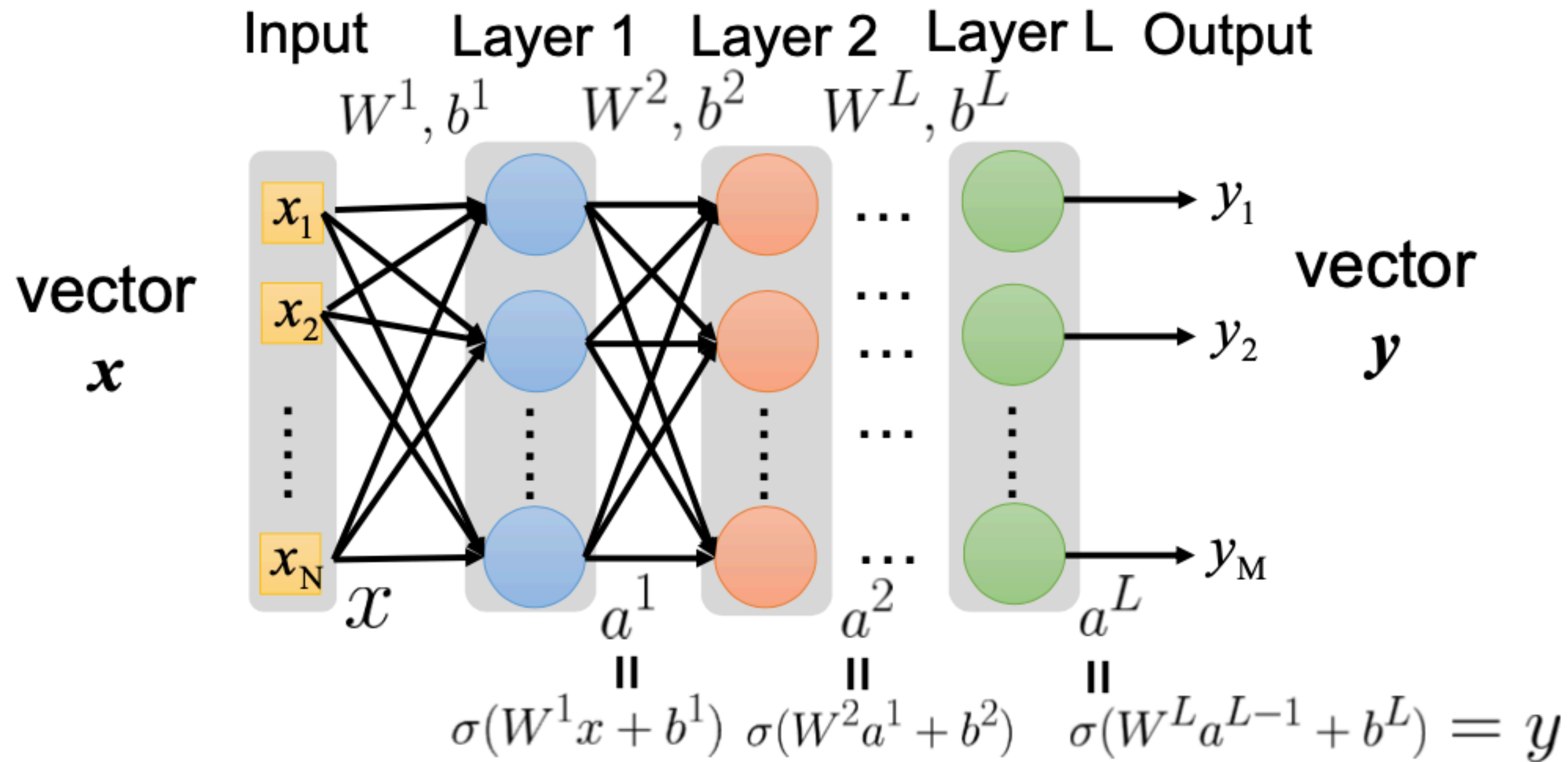
$$a^l = \sigma(z^l)$$



$$a^l = \sigma(W^l a^{l-1} + b^l)$$

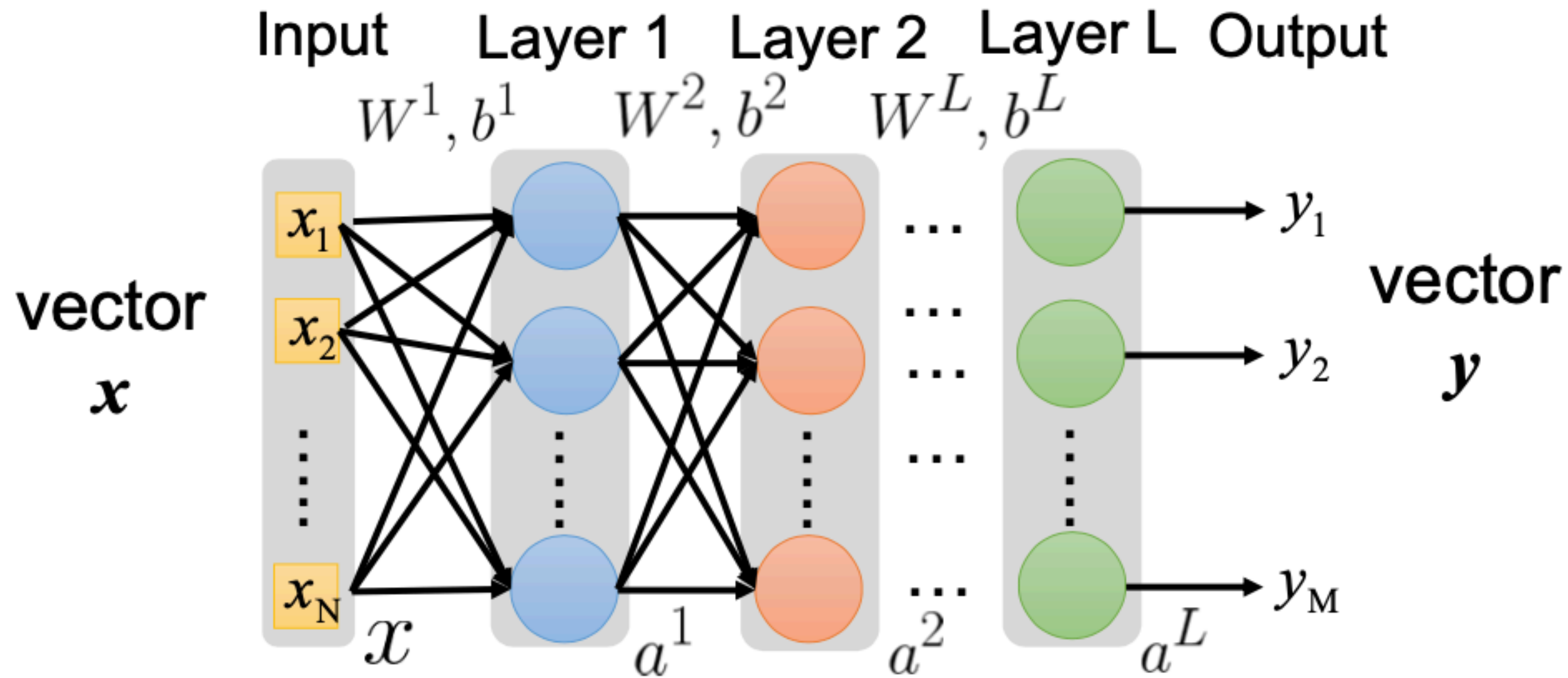
51 Neural Network Formulation

Fully connected feedforward network $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$



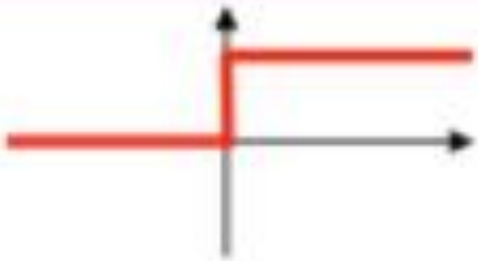
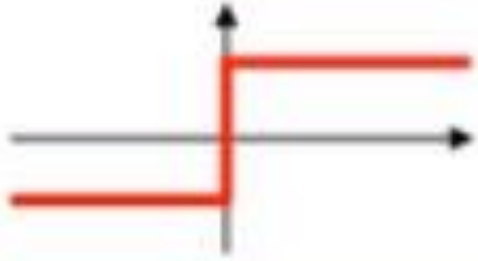
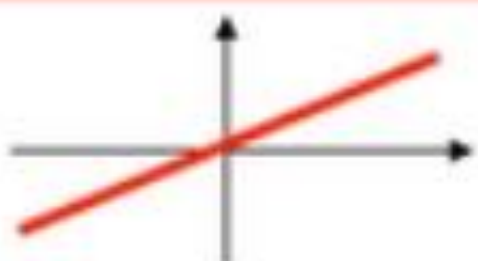


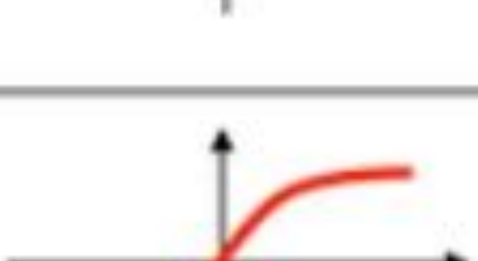
52 Neural Network Formulation

Fully connected feedforward network $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$



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

Activation Function $\sigma(\cdot)$

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

bounded function

Activation Function $\sigma(\cdot)$

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

boolean

linear

non - linear

Non-Linear Activation Function

Sigmoid

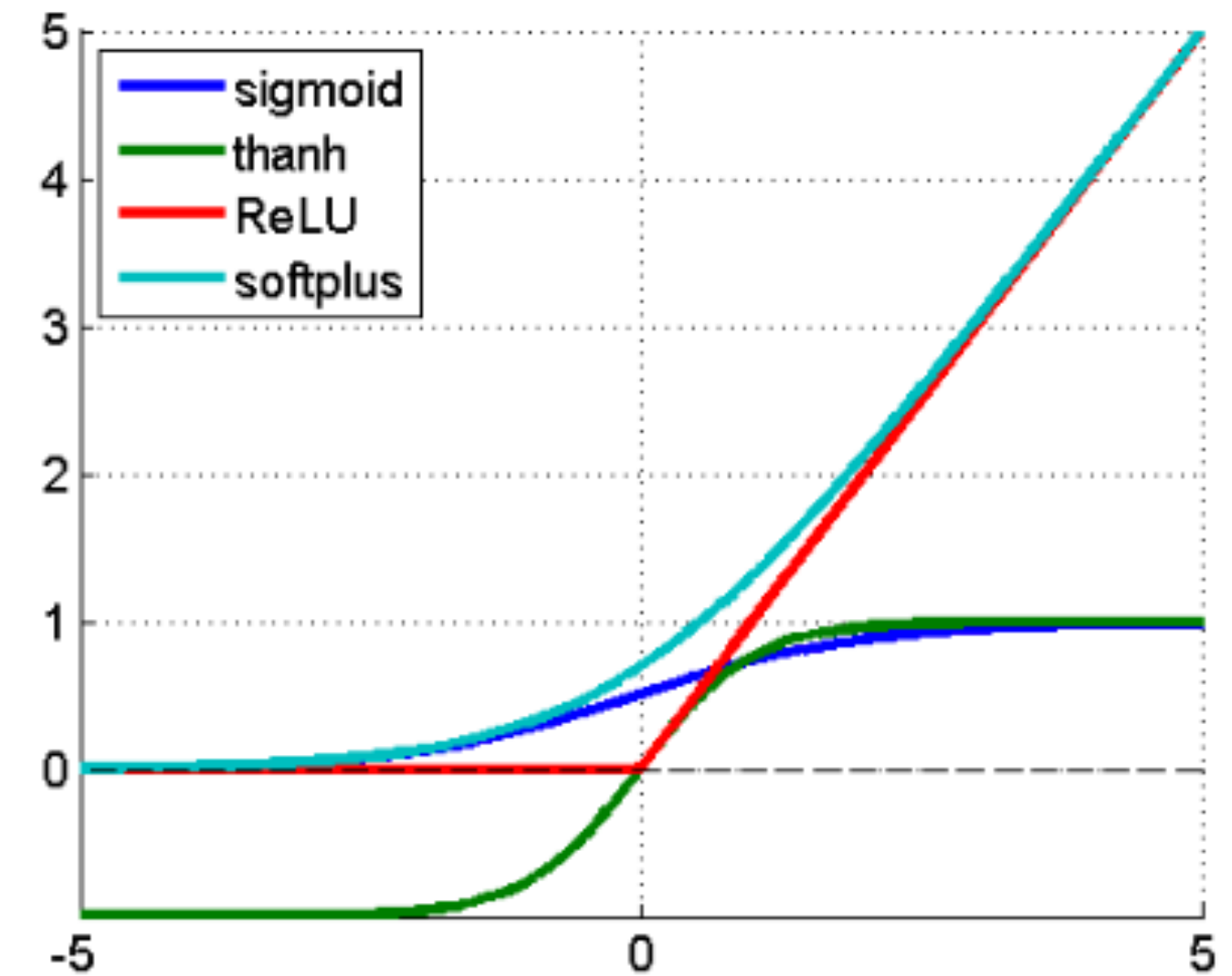
$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Tanh

$$\text{tanh}(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Rectified Linear Unit (ReLU)

$$\text{ReLU}(x) = \max(x, 0)$$



Non-linear functions are frequently used in neural networks

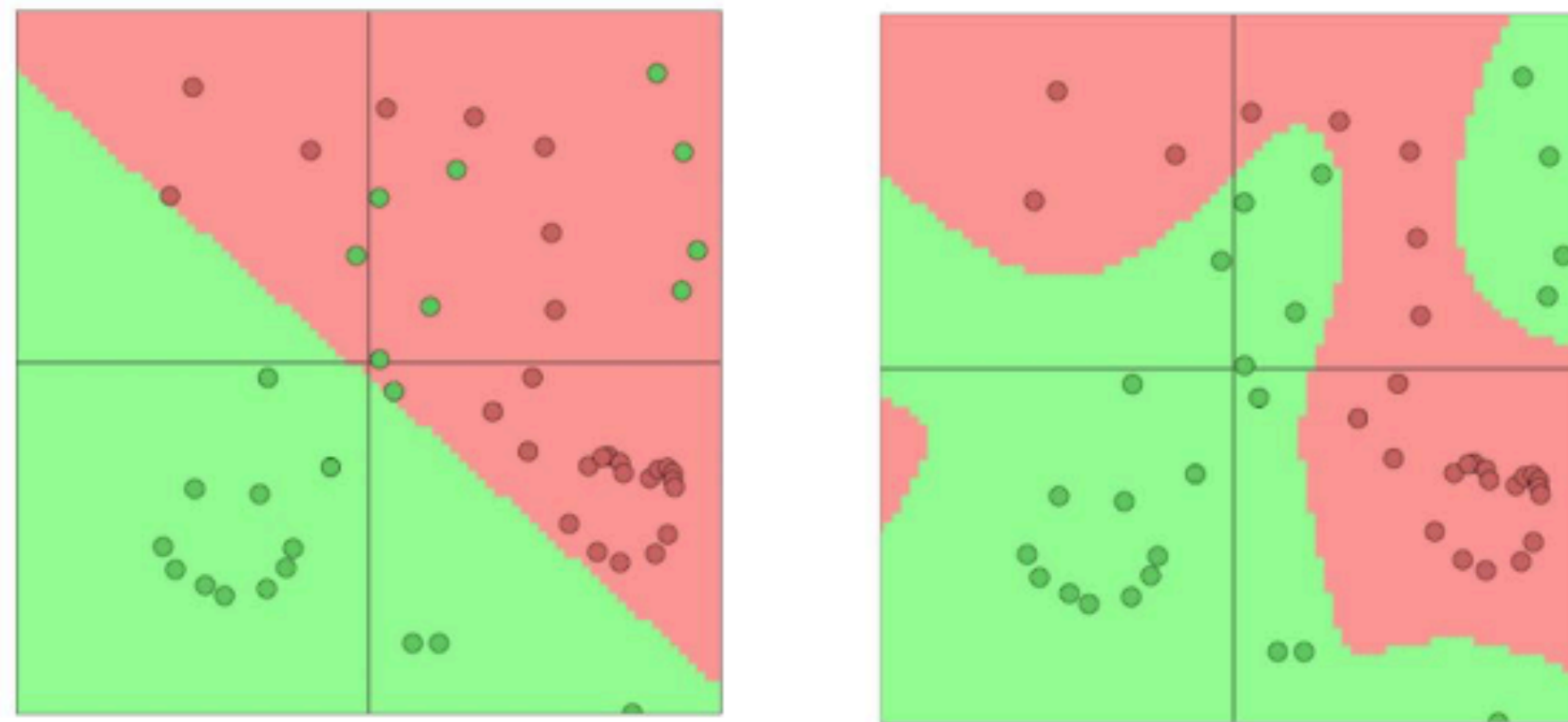
Why Non-Linearity?

Function approximation

- **Without non-linearity**, deep neural networks work the same as linear transform

$$W_1(W_2 \cdot x) = (W_1W_2)x = Wx$$

- **With non-linearity**, networks with more layers can approximate more complex functions



**What does the “good”
function mean?
Loss Function**

Function = Model Parameters

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

function set

different parameters W and $b \rightarrow$ different functions

Formal definition

$f(x; \theta)$ model parameter set

$$\theta = \{W^1, b^1, W^2, b^2, \cdots, W^L, b^L\}$$

pick a function $f =$ pick a set of model parameters θ

59 Model Parameter Measurement

- Define a function to measure the quality of a parameter set θ
 - Evaluating by **a loss/cost/error function** $C(\theta)$ → how bad θ is
 - Best model parameter set

$$\theta^* = \arg \min_{\theta} C(\theta)$$

- Evaluating by **an objective/reward function** $O(\theta)$ → how good θ is
- Best model parameter set

$$\theta^* = \arg \max_{\theta} O(\theta)$$

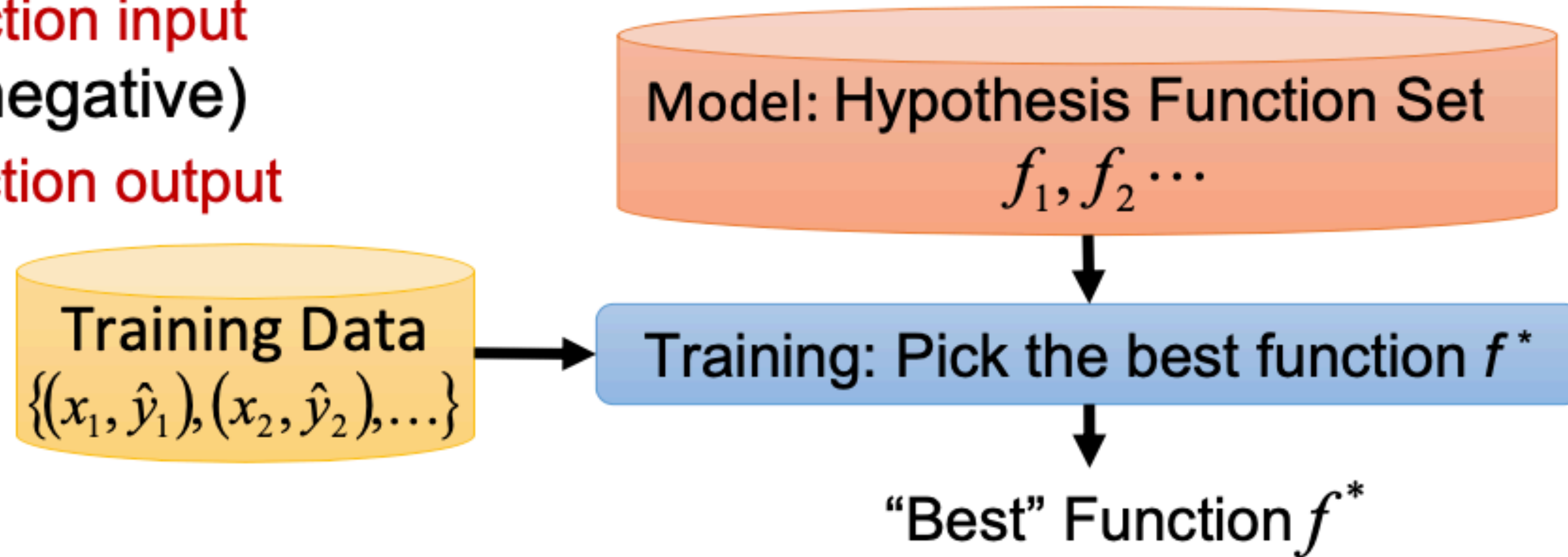
60 Loss Function Example

x : “It claims too much.”

function input

\hat{y} : - (negative)

function output



A “Good” function: $f(x; \theta) \sim \hat{y} \Rightarrow \|\hat{y} - f(x; \theta)\| \approx 0$

Define an example loss function: $C(\theta) = \sum_k \|\hat{y}_k - f(x_k; \theta)\|$

sum over the error of all training samples

Frequent Loss Functions

● Squared Error Loss

$$L = (y - f(x))^2$$

● Hinge Loss

$$L = \max(0, 1 - y * f(x))$$

● Binary Cross Entropy Loss

$$L = -y * \log(p) - (1 - y) * \log(1 - p)$$

● Multi-Class Cross Entropy Loss

$$L(X_i, Y_i) = - \sum_{j=1}^c y_{ij} * \log(p_{ij})$$

where Y_i is one – hot encoded target vector $(y_{i1}, y_{i2}, \dots, y_{ic})$,

$$y_{ij} = \begin{cases} 1, & \text{if } i_{th} \text{ element is in class } j \\ 0, & \text{otherwise} \end{cases}$$

$p_{ij} = f(X_i) = \text{Probability that } i_{th} \text{ element is in class } j$

**How can we pick the
“best” function?
Optimization**

Problem Statement

- Given a loss function and several model parameter sets
 - Loss function: $C(\theta)$
 - Model parameter sets: $\{\theta_1, \theta_2, \dots\}$
- Find a model parameter set that minimizes $C(\theta)$

How to solve this optimization problem?

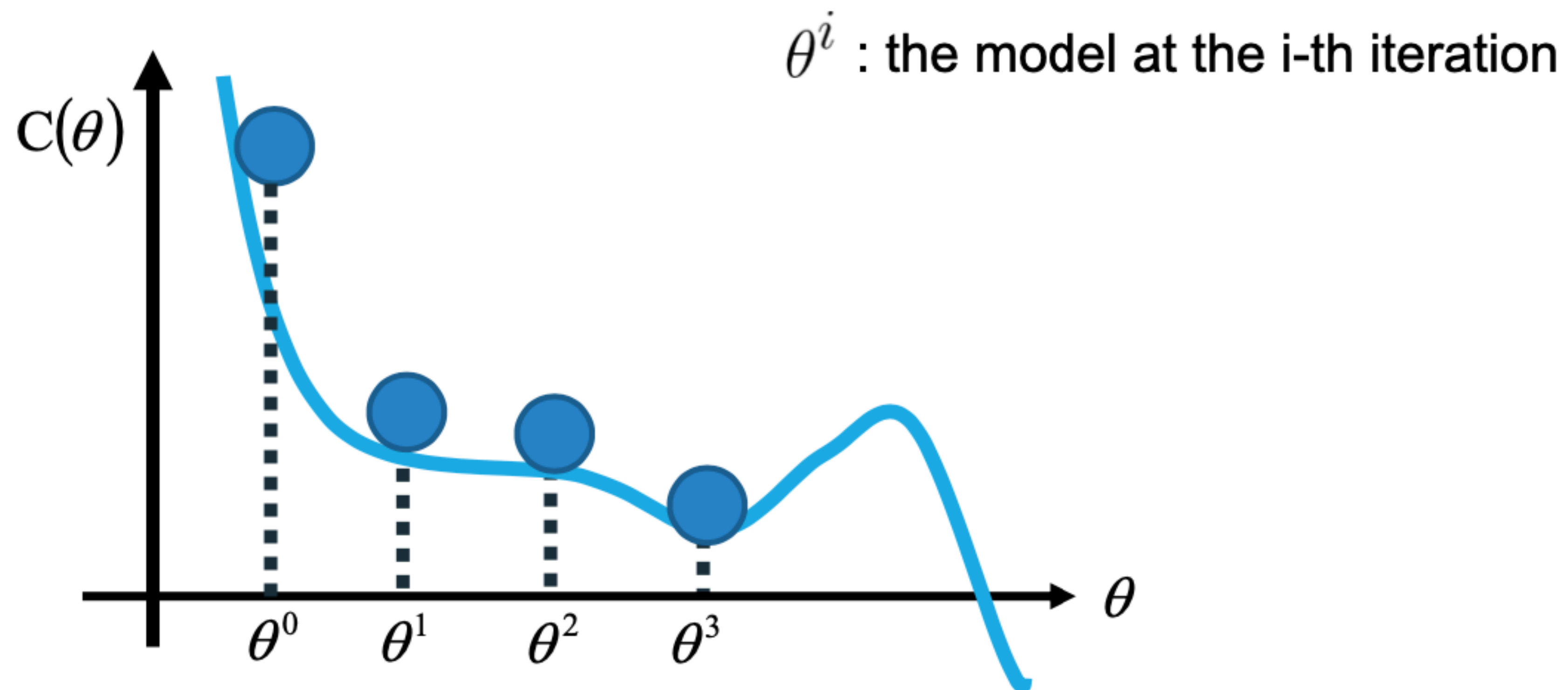
- 1) Brute force – enumerate all possible θ
- 2) Calculus – $\frac{\partial C(\theta)}{\partial \theta} = 0$

Issue: whole space of $C(\theta)$ is unknown



64 Gradient Descent for Optimization

- Assume that θ has only one variable



Idea: drop a ball and find the position where the ball stops rolling (local minima)

Gradient Descent for Optimization

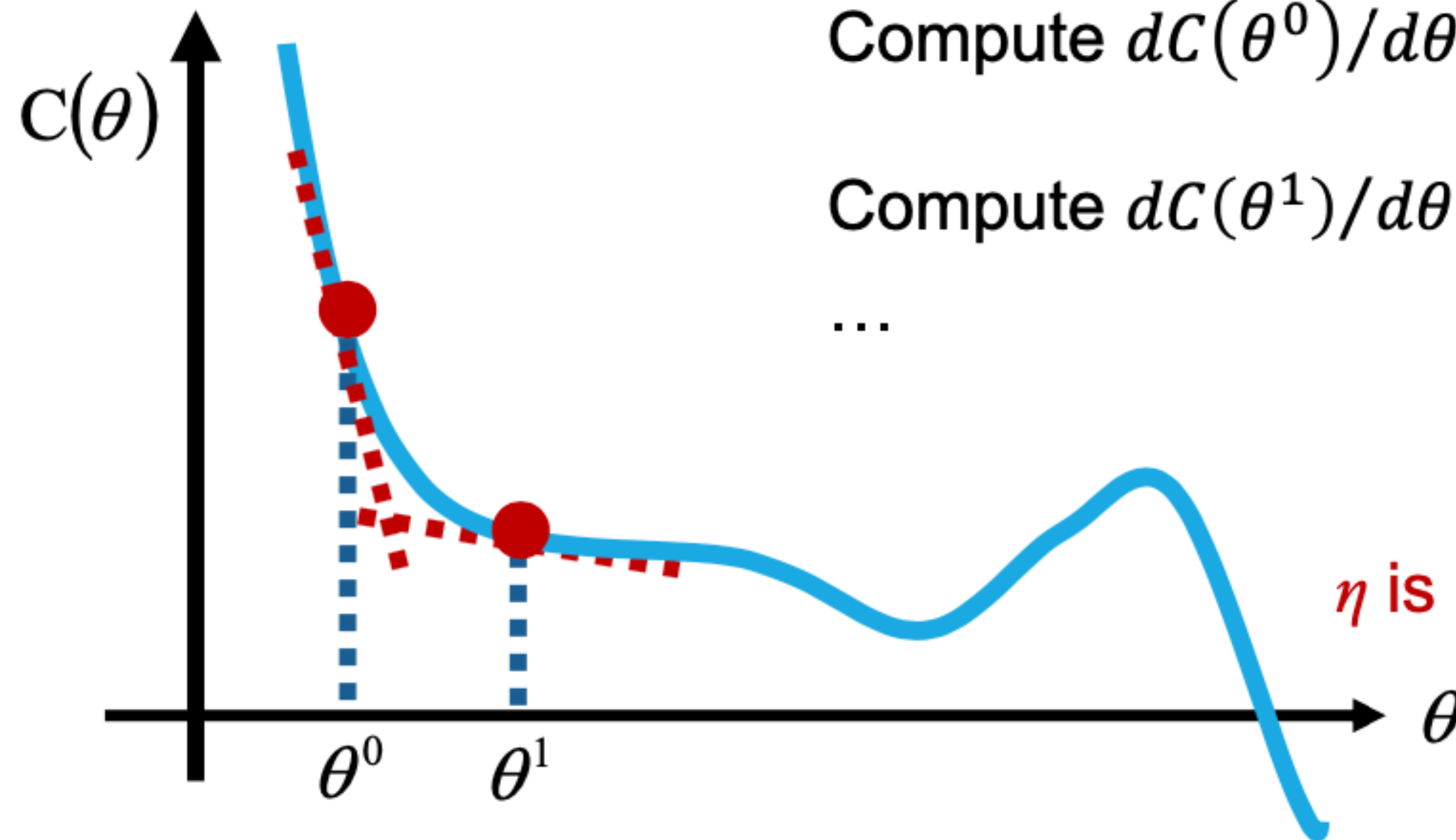
- Assume that θ has only one variable

Randomly start at θ^0

Compute $dC(\theta^0)/d\theta$: $\theta^1 \leftarrow \theta^0 - \eta \frac{\partial C(\theta^0)}{\partial \theta}$

Compute $dC(\theta^1)/d\theta$: $\theta^2 \leftarrow \theta^1 - \eta \frac{\partial C(\theta^1)}{\partial \theta}$

...

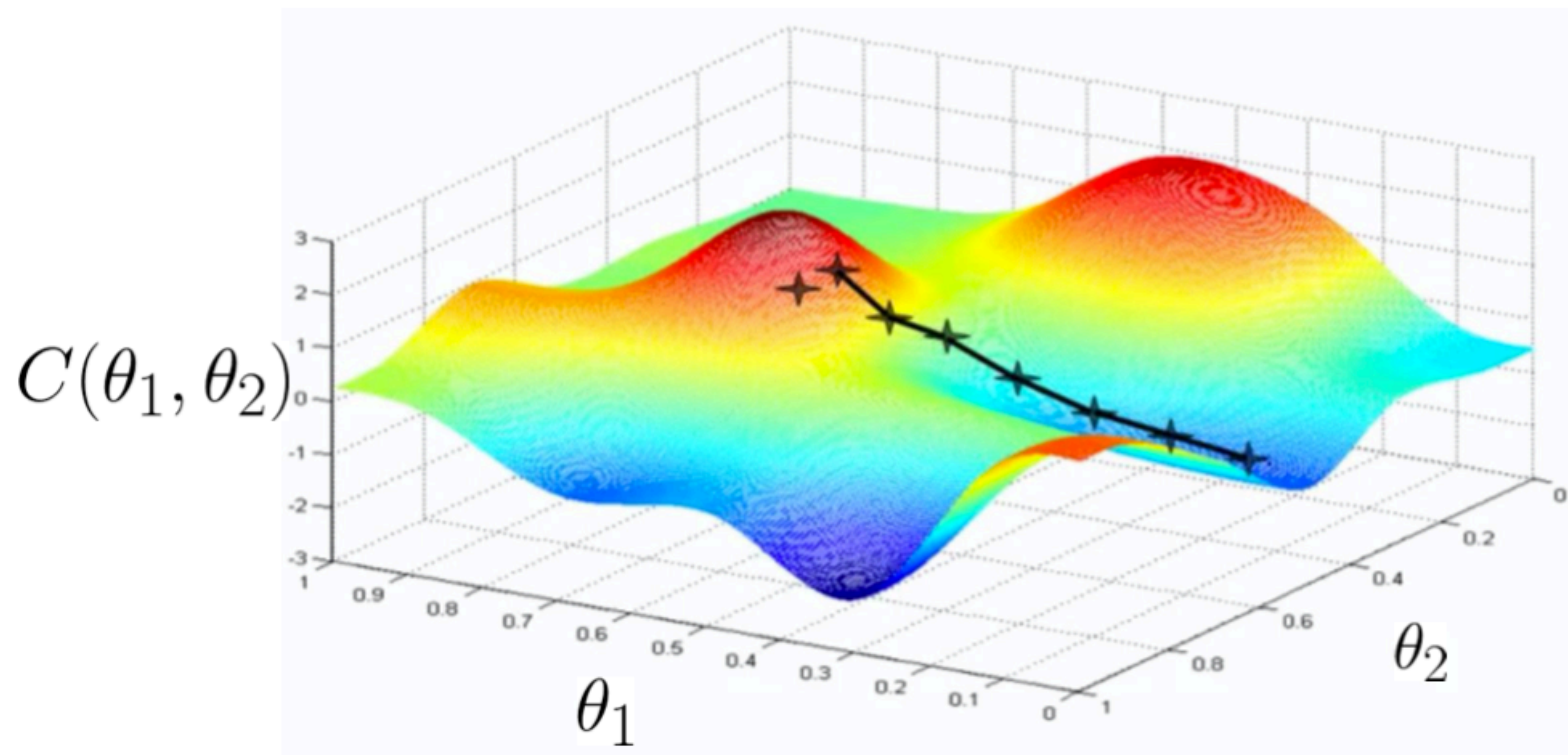


η is "learning rate"

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$$

66 Gradient Descent for Optimization

- Assume that θ has two variables $\{\theta_1, \theta_2\}$



67 Gradient Descent for Optimization

Assume that θ has two variables $\{\theta_1, \theta_2\}$

- Randomly start at θ^0 : $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

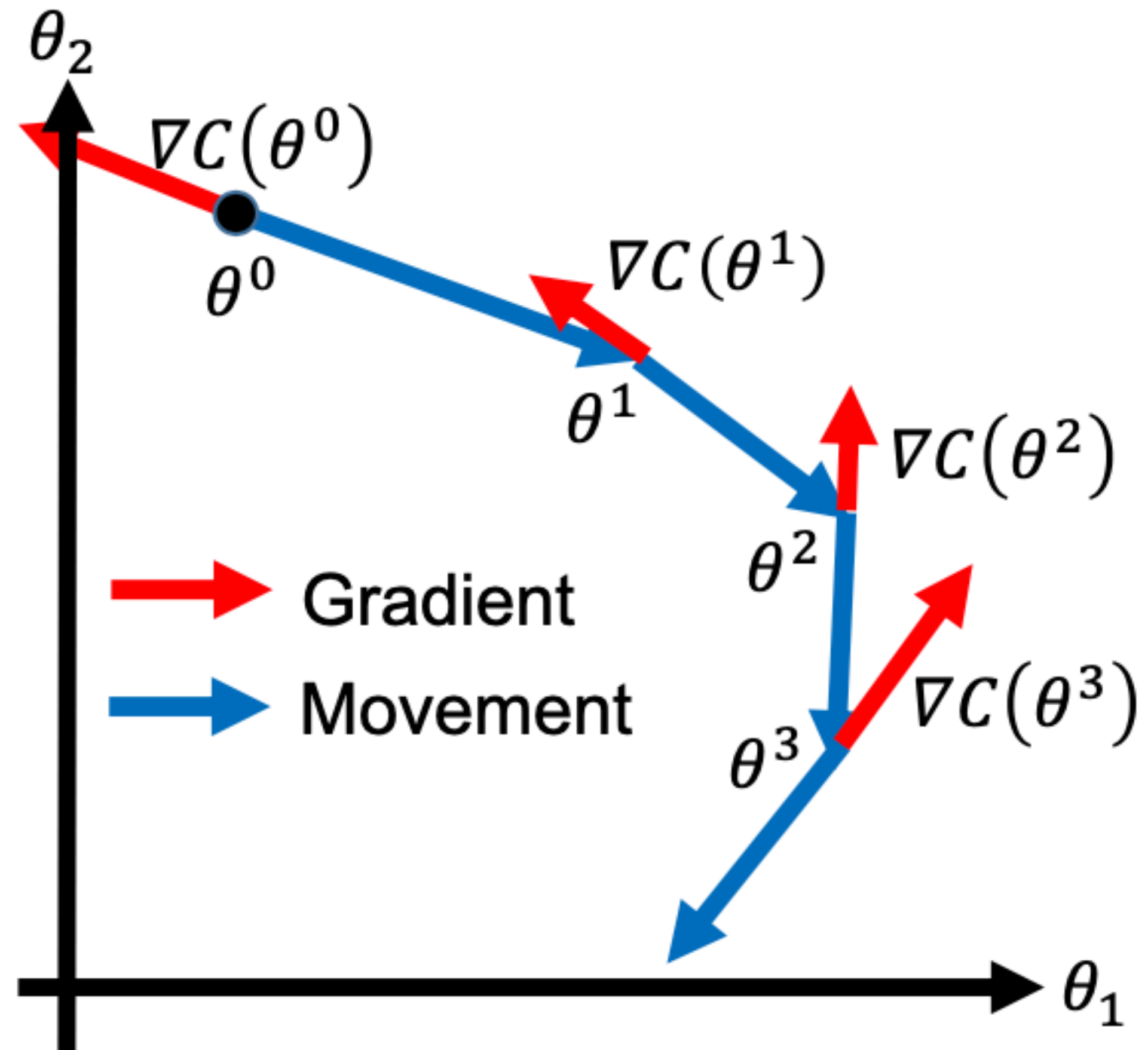
- Compute the gradients of $C(\theta)$ at θ^0 : $\nabla_{\theta} C(\theta^0) = \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$
- Update parameters:

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$$

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$$

- Compute the gradients of $C(\theta)$ at θ^1 : $\nabla_{\theta} C(\theta^1) = \begin{bmatrix} \frac{\partial C(\theta_1^1)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^1)}{\partial \theta_2} \end{bmatrix}$

Gradient Descent for Optimization



Algorithm

Initialization: start at θ^0
while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

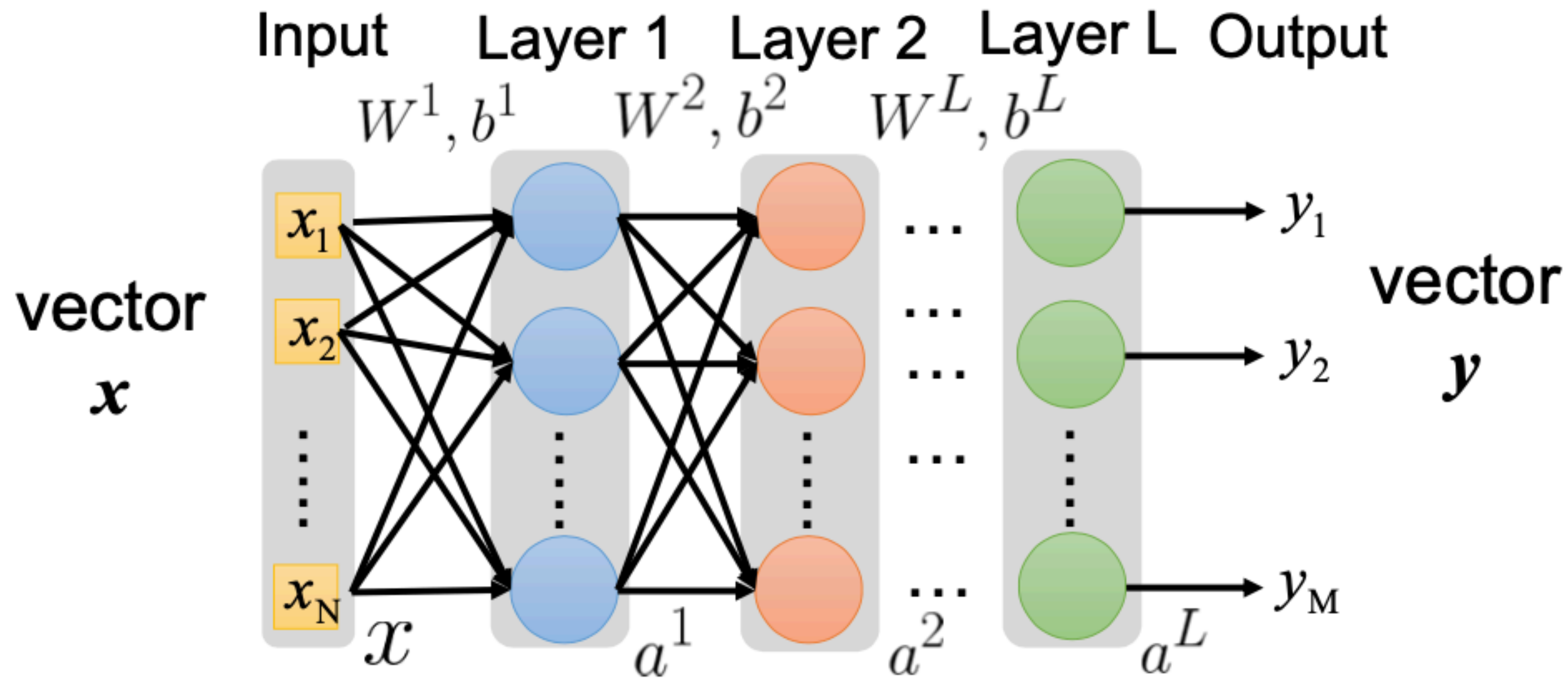
 update parameters

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$$

}

69 Revisit Neural Network Formulation

- Fully connected feedforward network $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$



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

70 Gradient Descent for Neural Network

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

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & & \cdots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

 update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$

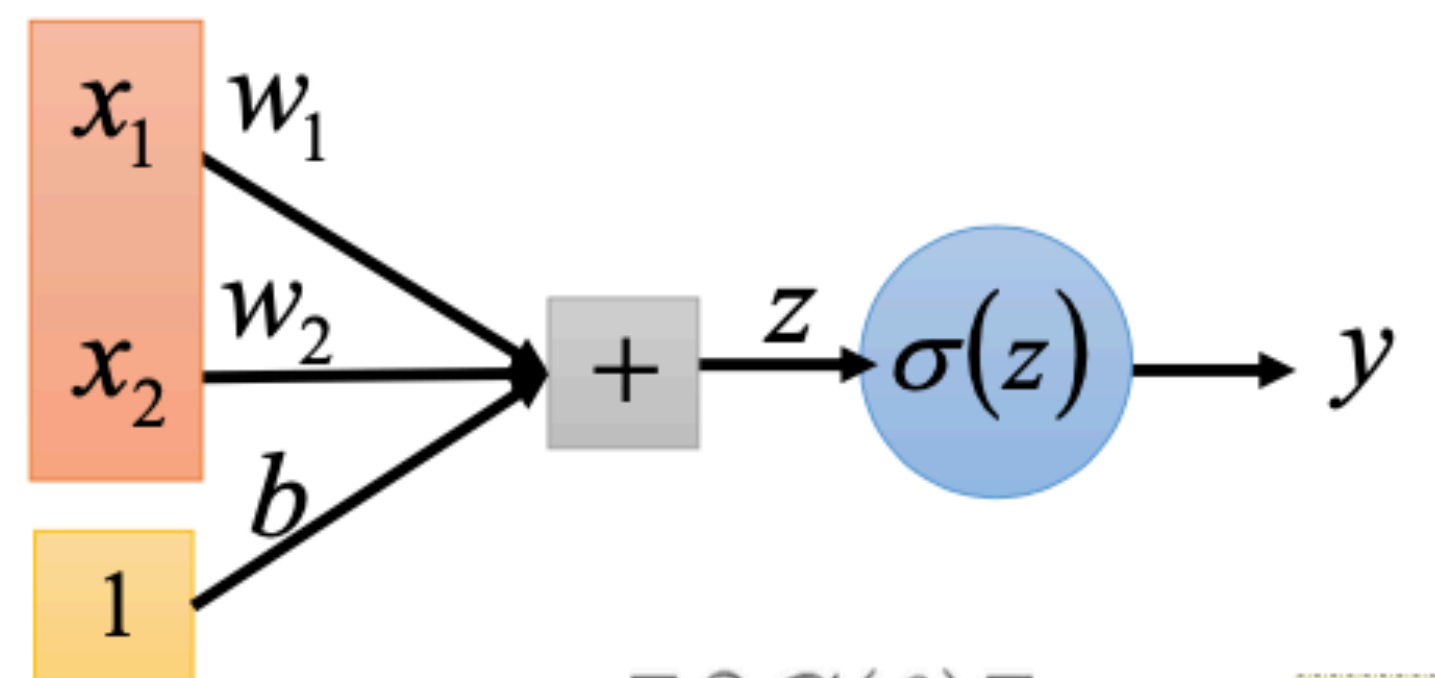
}

Gradient Descent for Optimization

Simple Case

$$y = f(x; \theta) = \sigma(Wx + b)$$

$$\theta = \{W, b\} = \{w_1, w_2, b\}$$



```

Algorithm
Initialization: start at  $\theta^0$ 
while( $\theta^{(i+1)} \neq \theta^i$ )
{
    compute gradient at  $\theta^i$ 
    update parameters
     $\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$ 
}
    
```

$$\nabla_{\theta} C(\theta) = \begin{bmatrix} \frac{\partial C(\theta)}{\partial w_1} \\ \frac{\partial C(\theta)}{\partial w_2} \\ \frac{\partial C(\theta)}{\partial b} \end{bmatrix}$$

$$\begin{bmatrix} w_1^{i+1} \\ w_2^{i+1} \\ b^{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} w_1^i \\ w_2^i \\ b^i \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta)}{\partial w_1} \\ \frac{\partial C(\theta)}{\partial w_2} \\ \frac{\partial C(\theta)}{\partial b} \end{bmatrix}$$

Gradient Descent for Optimization

Simple Case: Three Parameters and Square Error Loss

- Update three parameters for t -th iteration

$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_1}$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_2}$$

$$b^{(t+1)} = b^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial b}$$

$$\begin{bmatrix} w_1^{i+1} \\ w_2^{i+1} \\ b^{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} w_1^i \\ w_2^i \\ b^i \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta)}{\partial w_1} \\ \frac{\partial C(\theta)}{\partial w_2} \\ \frac{\partial C(\theta)}{\partial b} \end{bmatrix}$$

- Square error loss

$$C(\theta) = \sum_{\forall x} \|\hat{y} - f(x; \theta)\|^2 = (\hat{y} - f(x; \theta))^2$$

Gradient Descent for Optimization

Simple Case: Square Error Loss

● Square Error Loss

$$\frac{\partial C(\theta)}{\partial w_1} = \frac{\partial}{\partial w_1} (f(x; \theta) - \hat{y})^2$$

$$= 2(f(x; \theta) - \hat{y}) \frac{\partial}{\partial w_1} f(x; \theta)$$

$$= 2(\sigma(Wx + b) - \hat{y}) \frac{\partial}{\partial w_1} \sigma(Wx + b)$$

$$f(x; \theta) = \sigma(Wx + b)$$

Gradient Descent for Optimization

Simple Case: Square Error Loss

$$\frac{\partial \sigma(Wx + b)}{\partial w_1} = \frac{\partial \sigma(Wx + b)}{\partial (Wx + b)} \frac{\partial (Wx + b)}{\partial w_1}$$

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x} \text{ chain rule } \frac{\partial g(z)}{\partial z} = [1 - g(z)]g(z) \text{ sigmoid func } g(z) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial \sigma(Wx + b)}{\partial w_1} = [1 - \sigma(Wx + b)]\sigma(Wx + b) \frac{\partial (Wx + b)}{\partial w_1}$$

$$\frac{\partial (Wx + b)}{\partial w_1} = \frac{\partial (w_1x_1 + w_2x_2 + b)}{\partial w_1} = x_1$$

$$\frac{\partial \sigma(Wx + b)}{\partial w_1} = [1 - \sigma(Wx + b)]\sigma(Wx + b)x_1$$

Gradient Descent for Optimization

Simple Case: Square Error Loss

● Square Error Loss

$$\begin{aligned}\frac{\partial C(\theta)}{\partial w_1} &= \frac{\partial}{\partial w_1} (f(x; \theta) - \hat{y})^2 \\ &= 2(f(x; \theta) - \hat{y}) \frac{\partial}{\partial w_1} f(x; \theta) \\ &= 2(\sigma(Wx + b) - \hat{y}) \frac{\partial}{\partial w_1} \sigma(Wx + b)\end{aligned}$$

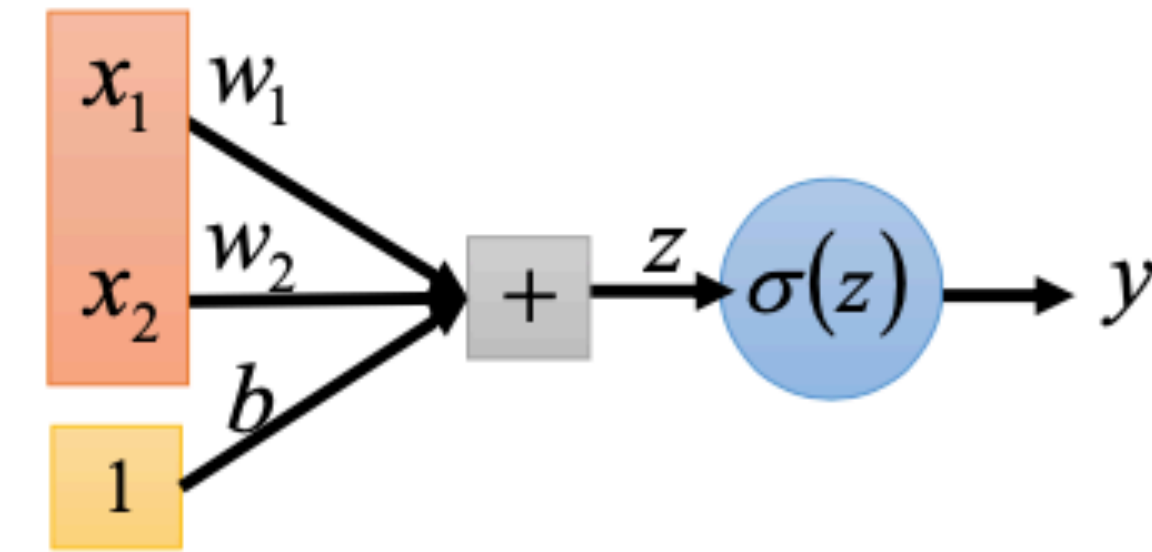
$$f(x; \theta) = \sigma(Wx + b)$$

$$\frac{\partial \sigma(Wx + b)}{\partial w_1} = [1 - \sigma(Wx + b)] \sigma(Wx + b) x_1$$

Gradient Descent for Optimization

Simple Case: Three Parameters and Square Error Loss

- Update three parameters for t -th iteration



$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_1}$$

$$\frac{\partial C(\theta)}{\partial w_1} = 2(\sigma(Wx + b) - \hat{y})[1 - \sigma(Wx + b)]\sigma(Wx + b)x_1$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_2}$$

$$\frac{\partial C(\theta)}{\partial w_2} = 2(\sigma(Wx + b) - \hat{y})[1 - \sigma(Wx + b)]\sigma(Wx + b)x_2$$

$$b^{(t+1)} = b^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial b}$$

$$\frac{\partial C(\theta)}{\partial b} = 2(\sigma(Wx + b) - \hat{y})[1 - \sigma(Wx + b)]\sigma(Wx + b)$$

77 Gradient Descent for Neural Network

$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \dots \\ \vdots & & \dots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

 update parameters

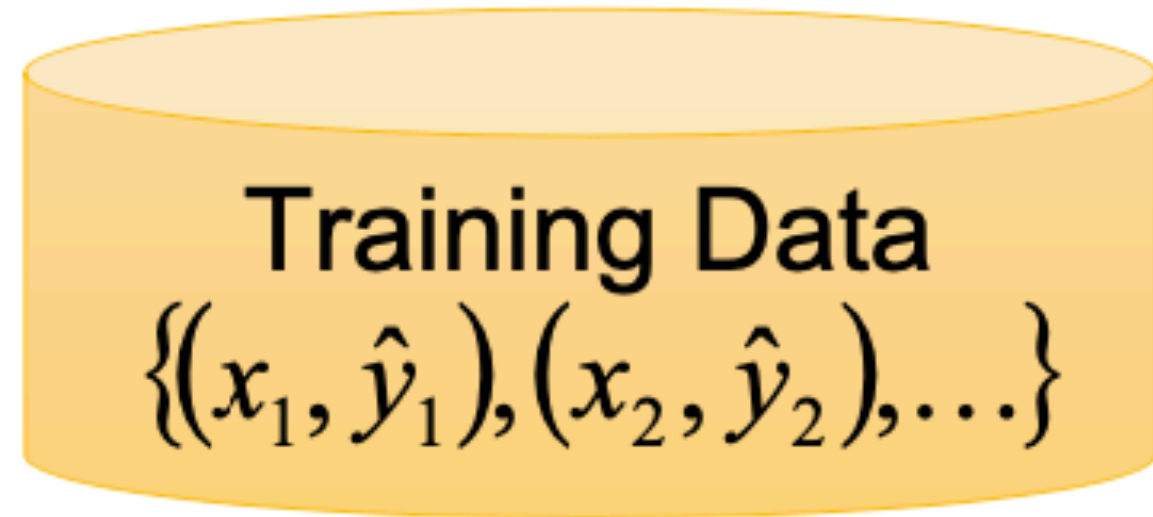
$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$

}

Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

Gradient Descent Issue

$$\theta^{i+1} = \theta^i - \eta \nabla C(\theta^i)$$



$$C(\theta) = \frac{1}{K} \sum_k \|f(x_k; \theta) - \hat{y}_k\| = \frac{1}{K} \sum_k C_k(\theta)$$

$$\nabla C(\theta^i) = \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$

After seeing all training samples, the model can be updated → slow

79 Stochastic Gradient Descent (SGD)

Gradient Descent

$$\theta^{i+1} = \theta^i - \eta \nabla C(\theta^i) \quad \nabla C(\theta^i) = \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$

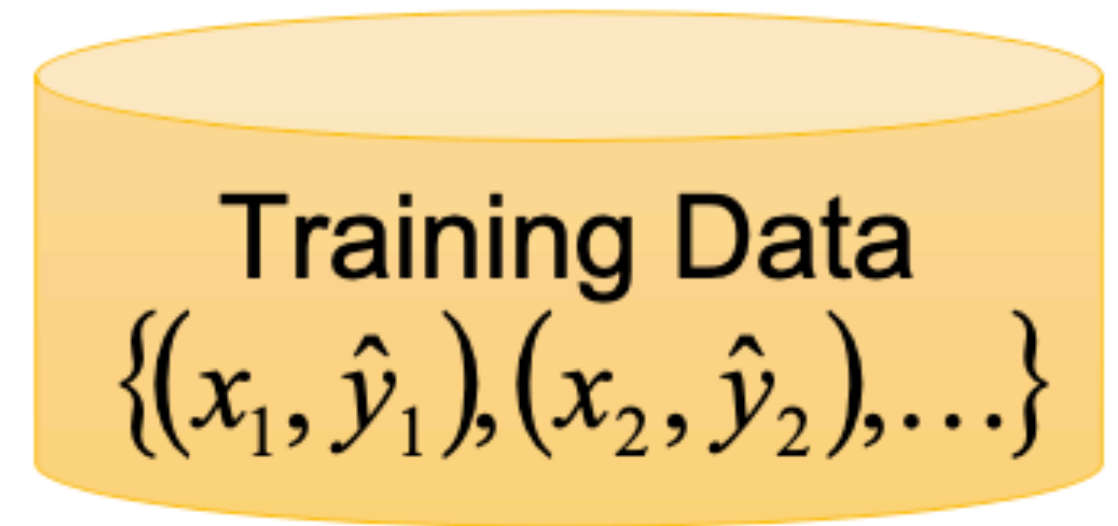
Stochastic Gradient Descent (SGD)

- Pick a training sample x_k

$$\theta^{i+1} = \theta^i - \eta \nabla C_k(\theta^i)$$

- If all training samples have same probability to be picked

$$E[\nabla C_k(\theta^i)] = \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$



The model can be updated after seeing one training sample → faster

80 Epoch Definition

- When running SGD, the model starts θ^0

pick x_1 $\theta^1 = \theta^0 - \eta \nabla C_1(\theta^0)$

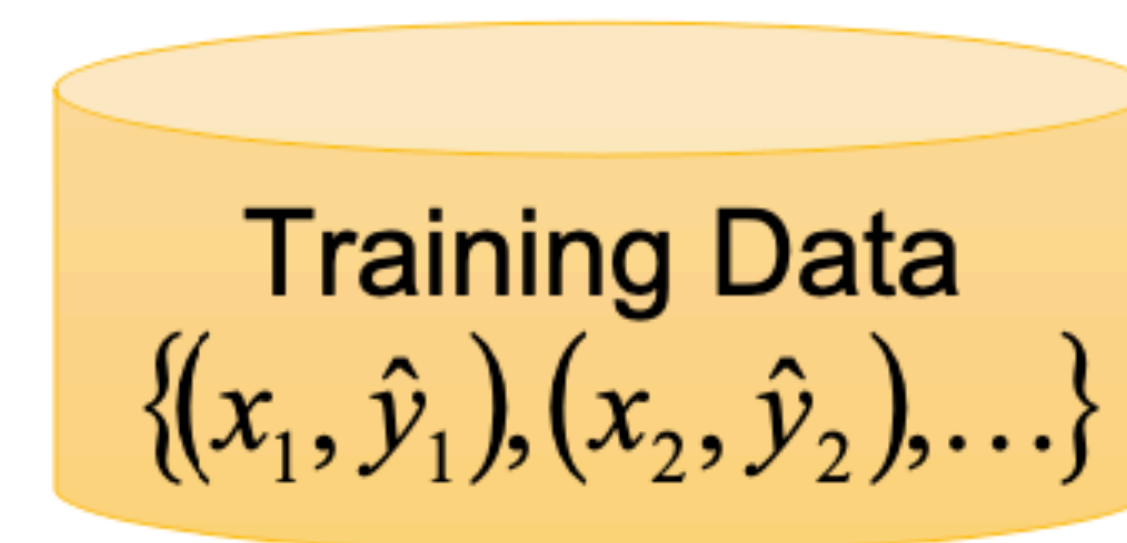
pick x_2 $\theta^2 = \theta^1 - \eta \nabla C_2(\theta^1)$

\vdots

pick $\theta^k = \theta^{k-1} - \eta \nabla C_k(\theta^{k-1})$

x_k \vdots

pick x_K $\theta^K = \theta^{K-1} - \eta \nabla C_K(\theta^{K-1})$



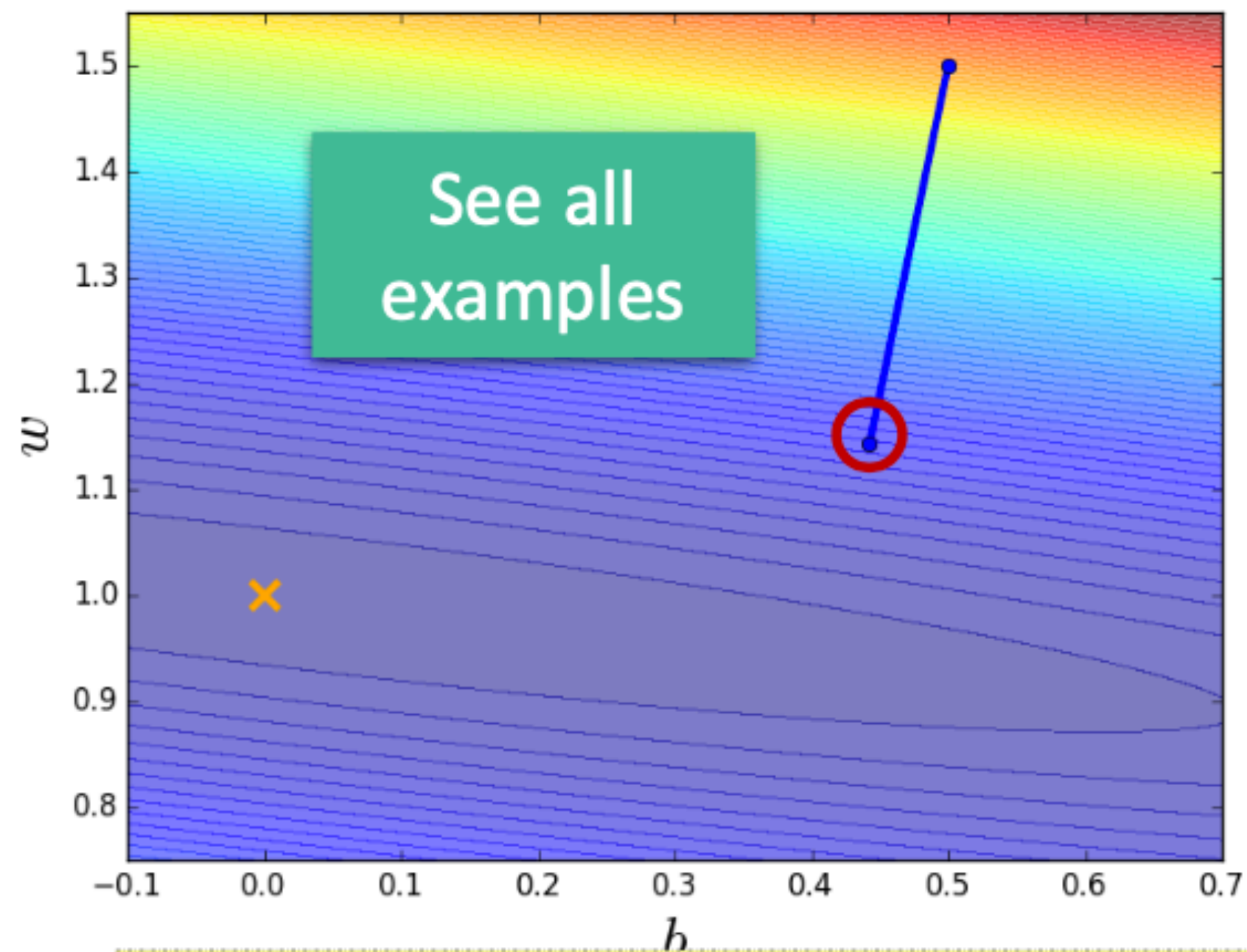
see all training samples once

→ one epoch

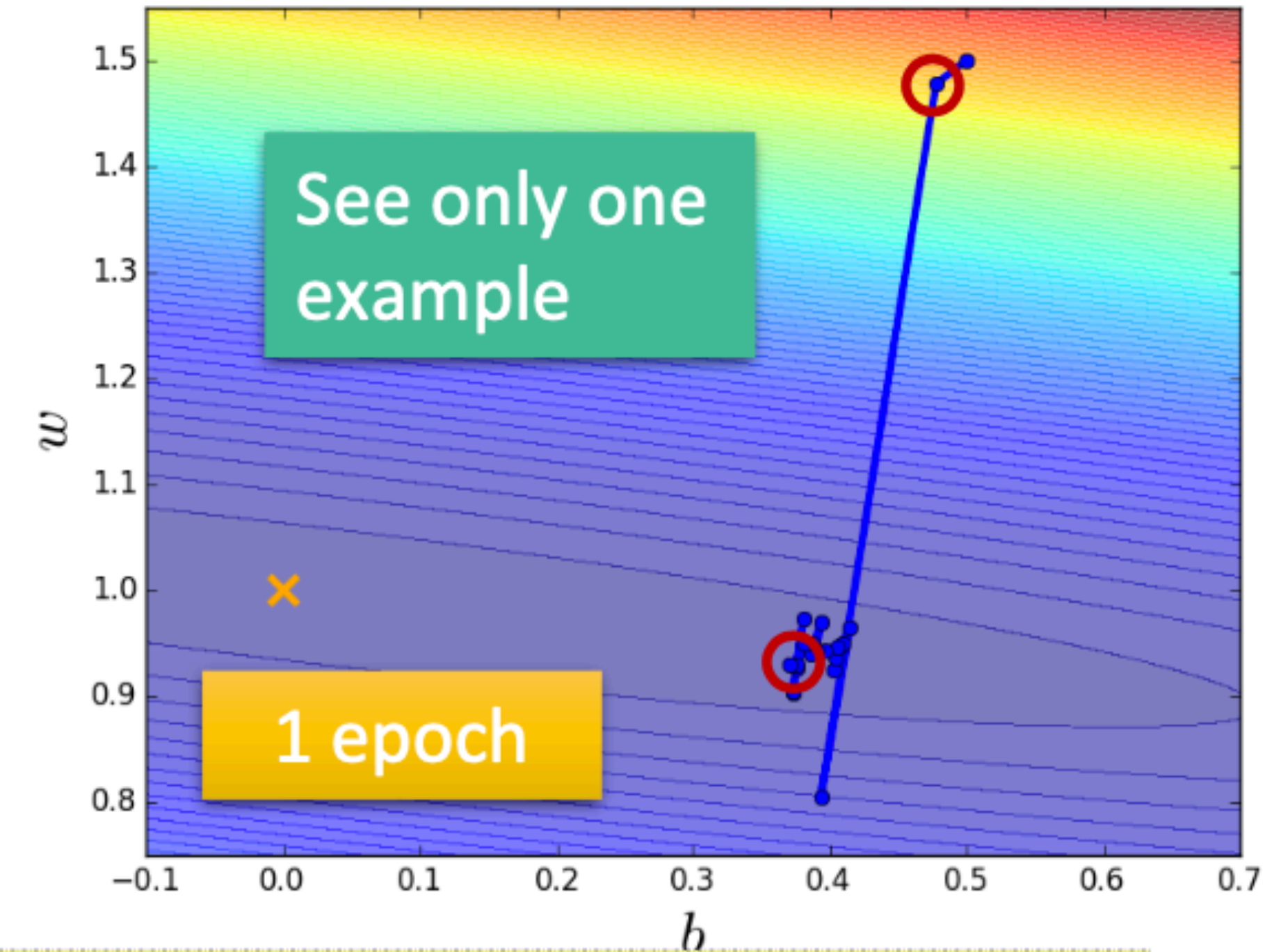
pick x_1 $\theta^{K+1} = \theta^K - \eta \nabla C_1(\theta^K)$

Gradient Descent v.s. SGD

- 🕒 Gradient Descent
- ✓ Update after seeing all examples



- 🕒 Stochastic Gradient Descent
- ✓ If there are 20 examples, update 20 times in one epoch.



SGD approaches to the target point faster than gradient descent

Mini-Batch SGD

Batch Gradient Descent

Use all K samples in each iteration

$$\theta^{i+1} = \theta^i - \eta \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$

Stochastic Gradient Descent (SGD)

- Pick a training sample x_k

Use 1 samples in each iteration

$$\theta^{i+1} = \theta^i - \eta \nabla C_k(\theta^i)$$

Mini-Batch SGD

- Pick a set of B training samples as a batch b

B is “batch size”

Use all B samples in each iteration

$$\theta^{i+1} = \theta^i - \eta \frac{1}{B} \sum_{x_k \in b} \nabla C_k(\theta^i)$$

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

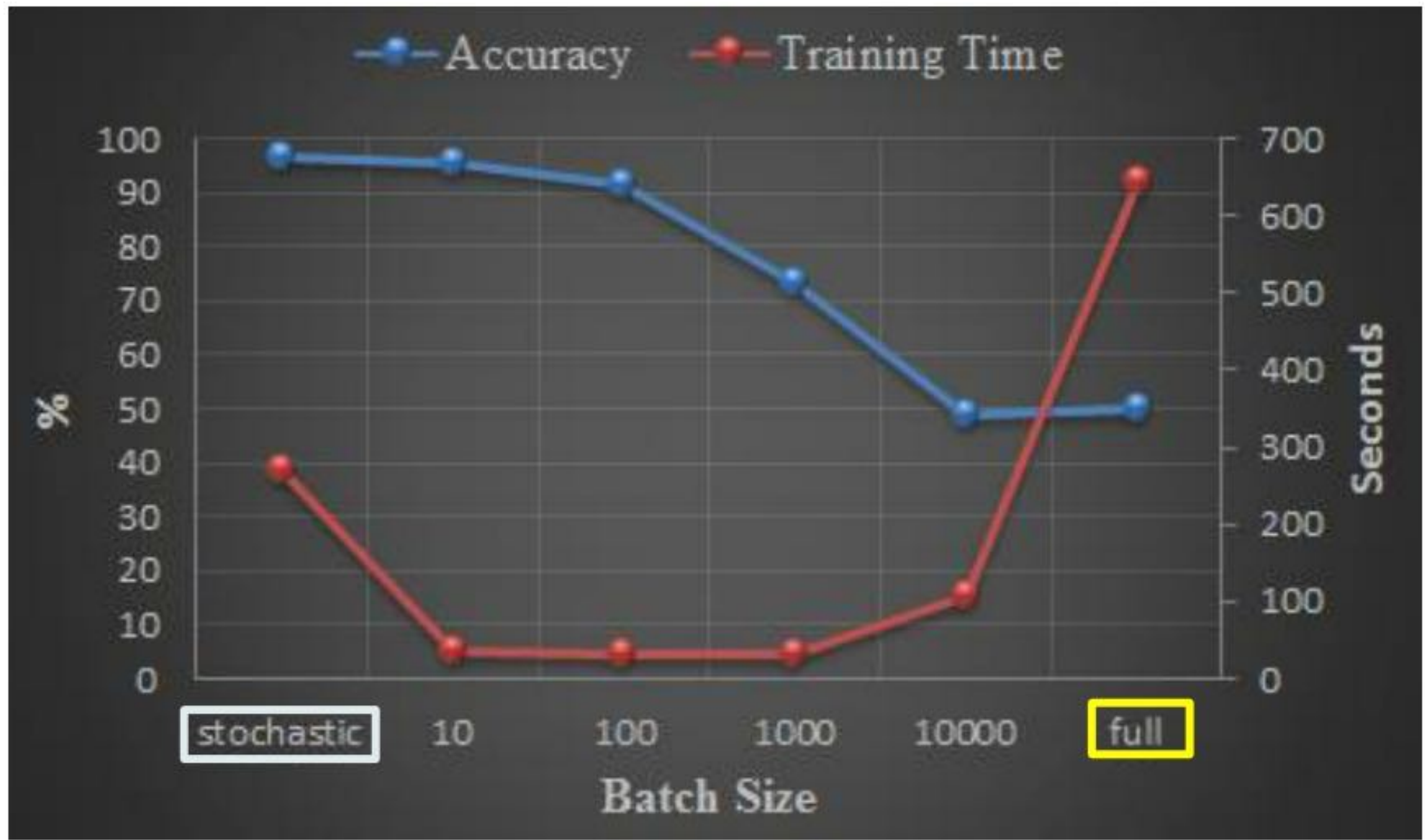
Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

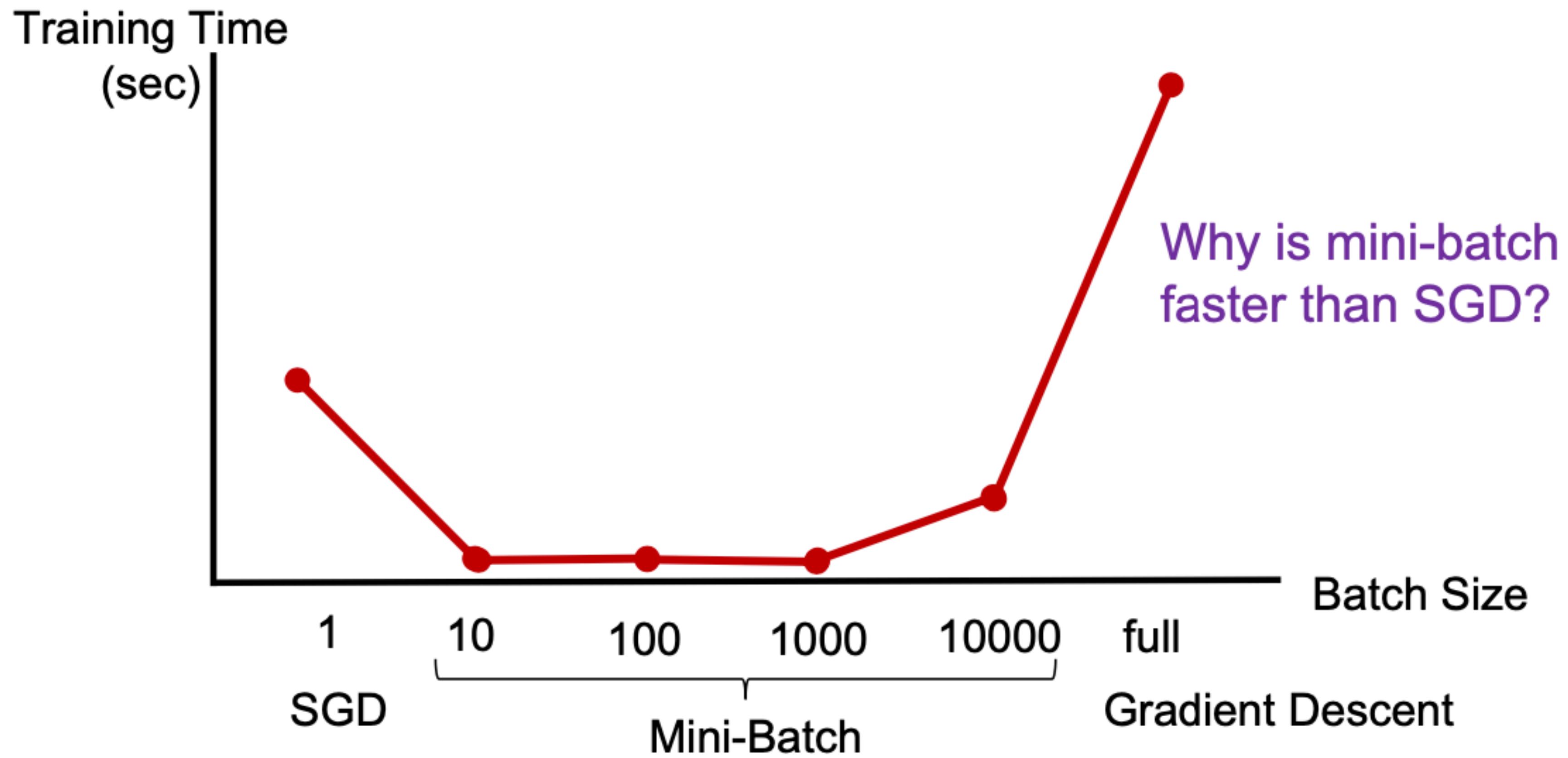
end while

Batch v.s. Mini-Batch Handwriting Digit Classification



GD v.s. SGD v.s. Mini-Batch SGD

Training speed: mini-batch > SGD > Gradient Descent



86 SGD v.s. Mini-Batch SGD

Stochastic Gradient Descent (SGD)

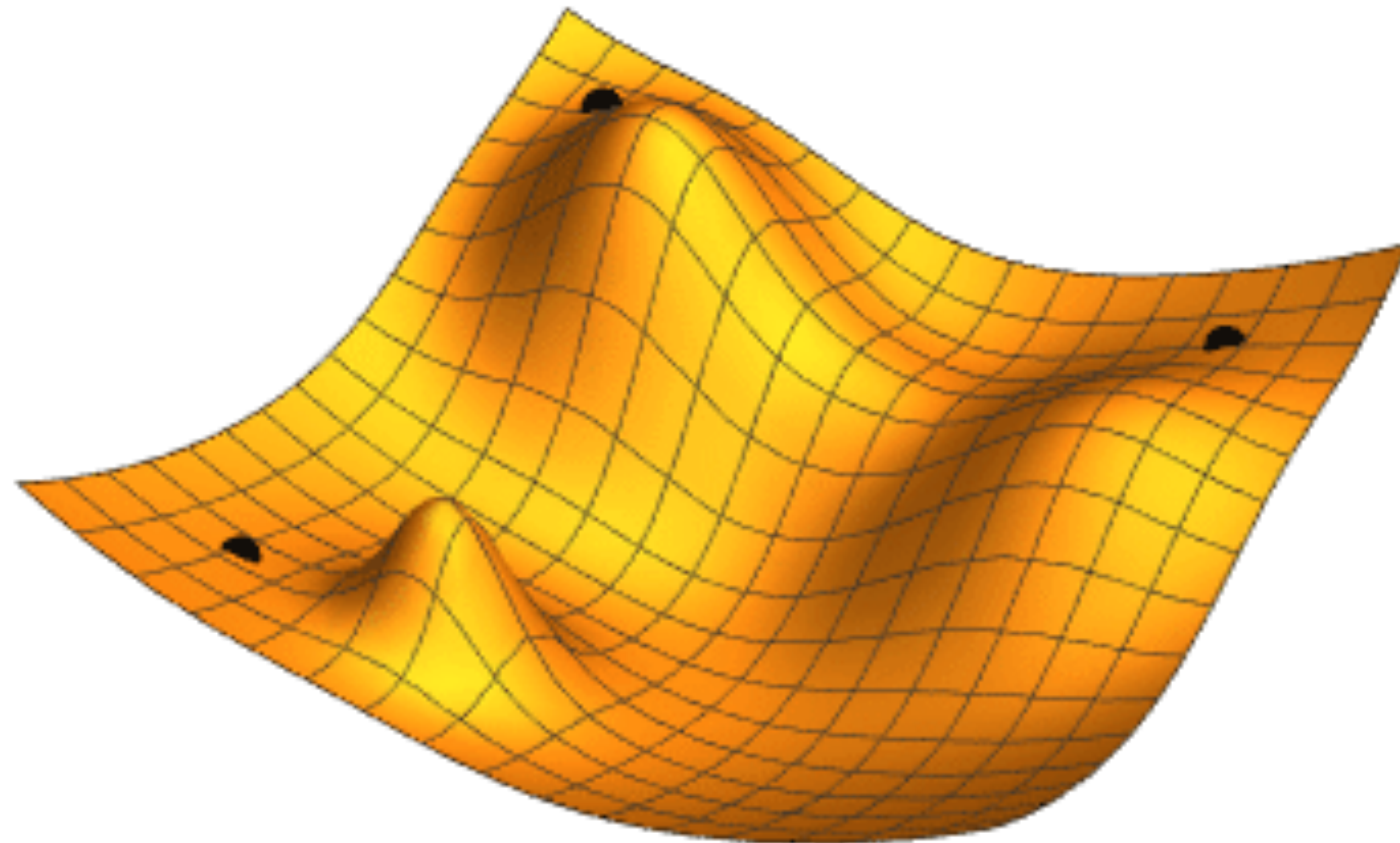
The diagram illustrates the Stochastic Gradient Descent (SGD) process. It shows two separate operations. The first operation is represented by an orange vertical bar labeled z^1 on the left, followed by an equals sign, a blue rectangular box labeled W^1 , and a yellow vertical bar labeled x on the right. The second operation is represented by a blue vertical bar labeled z^1 on the left, followed by an equals sign, a blue rectangular box labeled W^1 , and a green vertical bar labeled x on the right. To the right of the second operation, there are three dots indicating that this process repeats for many different samples.

Mini-Batch SGD

The diagram illustrates the Mini-Batch SGD process. It shows a single operation where a red-bordered box on the left contains two vertical bars, one orange labeled z^1 and one blue labeled z^1 . This is followed by an equals sign, a blue rectangular box labeled W^1 , and another red-bordered box on the right. This second box is labeled "matrix" above it and contains two vertical bars, one yellow labeled x and one green labeled x .

Modern computers run matrix-matrix multiplication faster than matrix-vector multiplication

87 Big Issue: Local Optima



Neural networks has no guarantee for obtaining global optimal solution

Tips for Neural Network Training

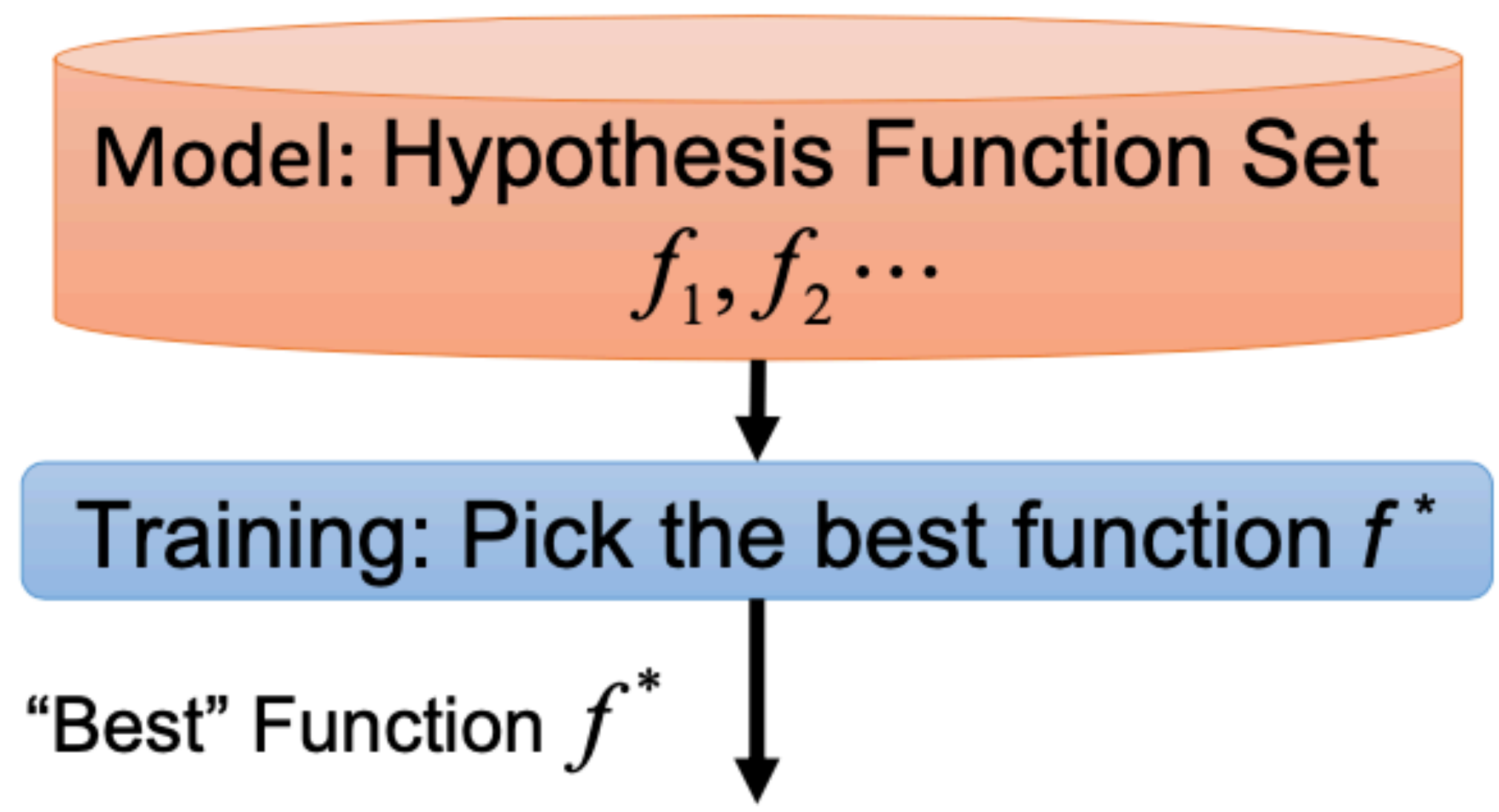
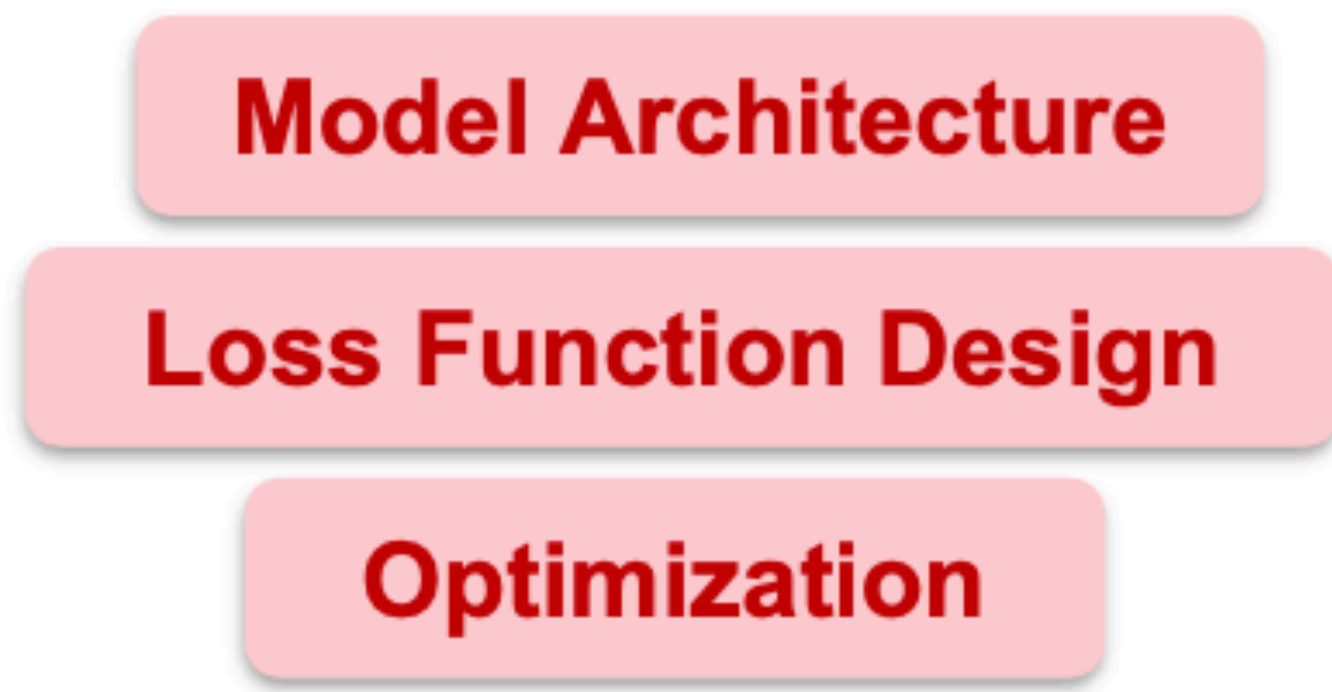
<http://karpathy.github.io/2019/04/25/recipe/>

<https://www.lri.fr/~gcharpia/deeppractice/2020/tips.pdf>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#>

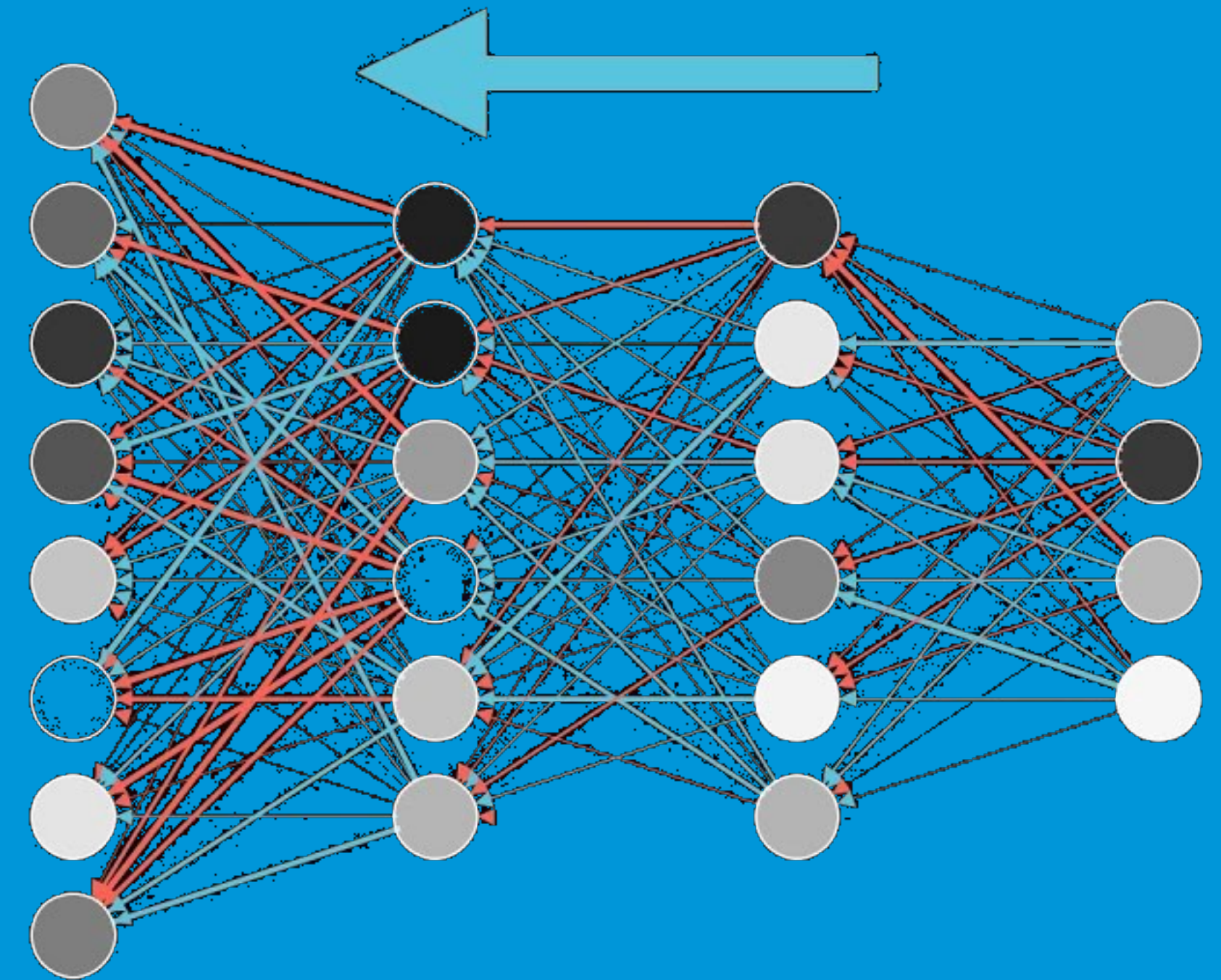
Concluding Remarks

- Q1. What is the model?
- Q2. What does a “good” function mean?
- Q3. How do we pick the “best” function?



Computing Gradients by Backpropagation

Backpropagation



91 Recap: Notation Summary

a_i^l : output of a neuron

a^l : output vector of a layer

z_i^l : input of activation function

z^l : input vector of activation function
for a layer

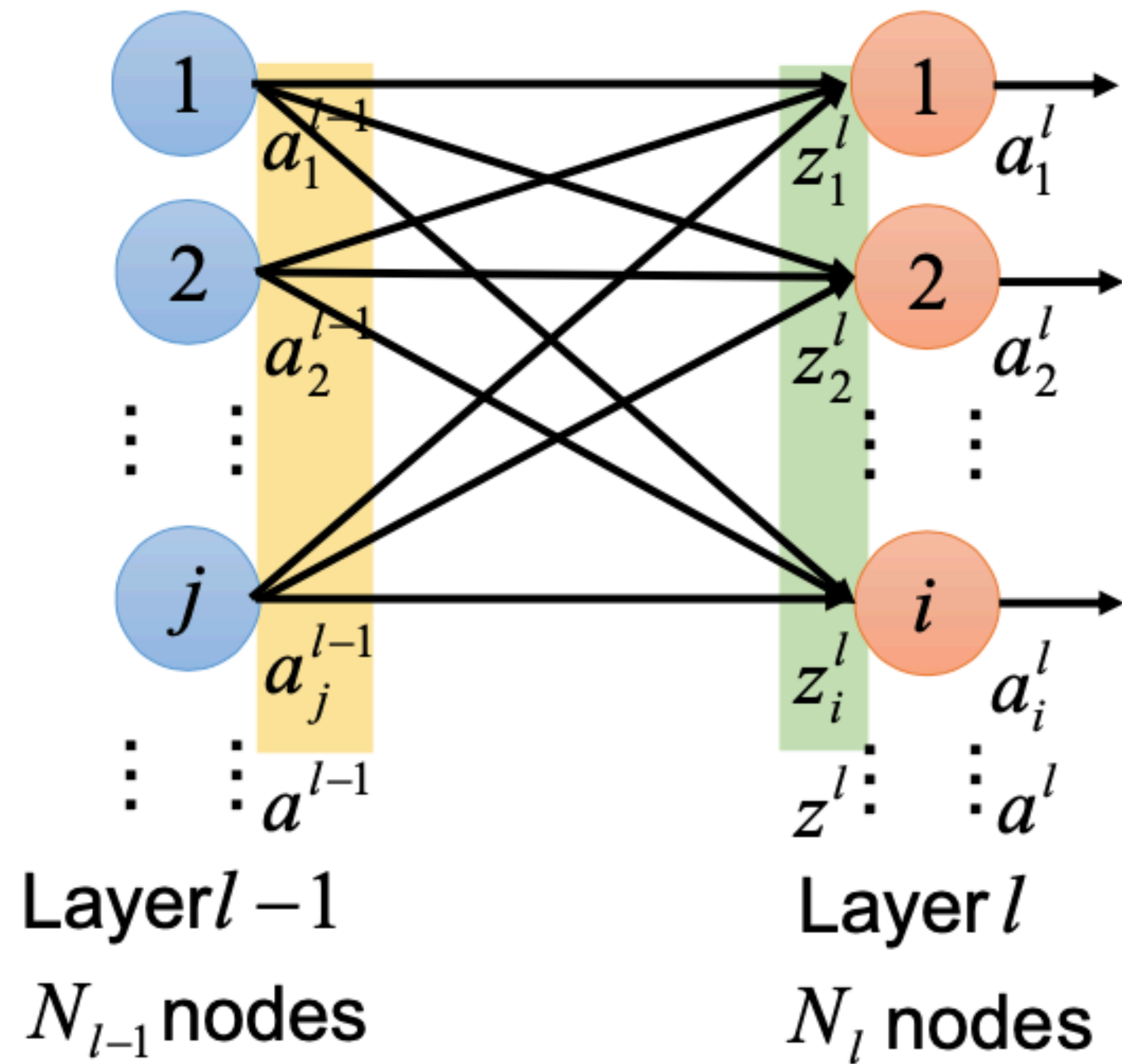
w_{ij}^l : a weight

W^l : a weight matrix

b_i^l : a bias

b^l : a bias vector

92 Recap: Layer Output Relation: from a to z



$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \dots + b_1^l$$

$$\vdots$$

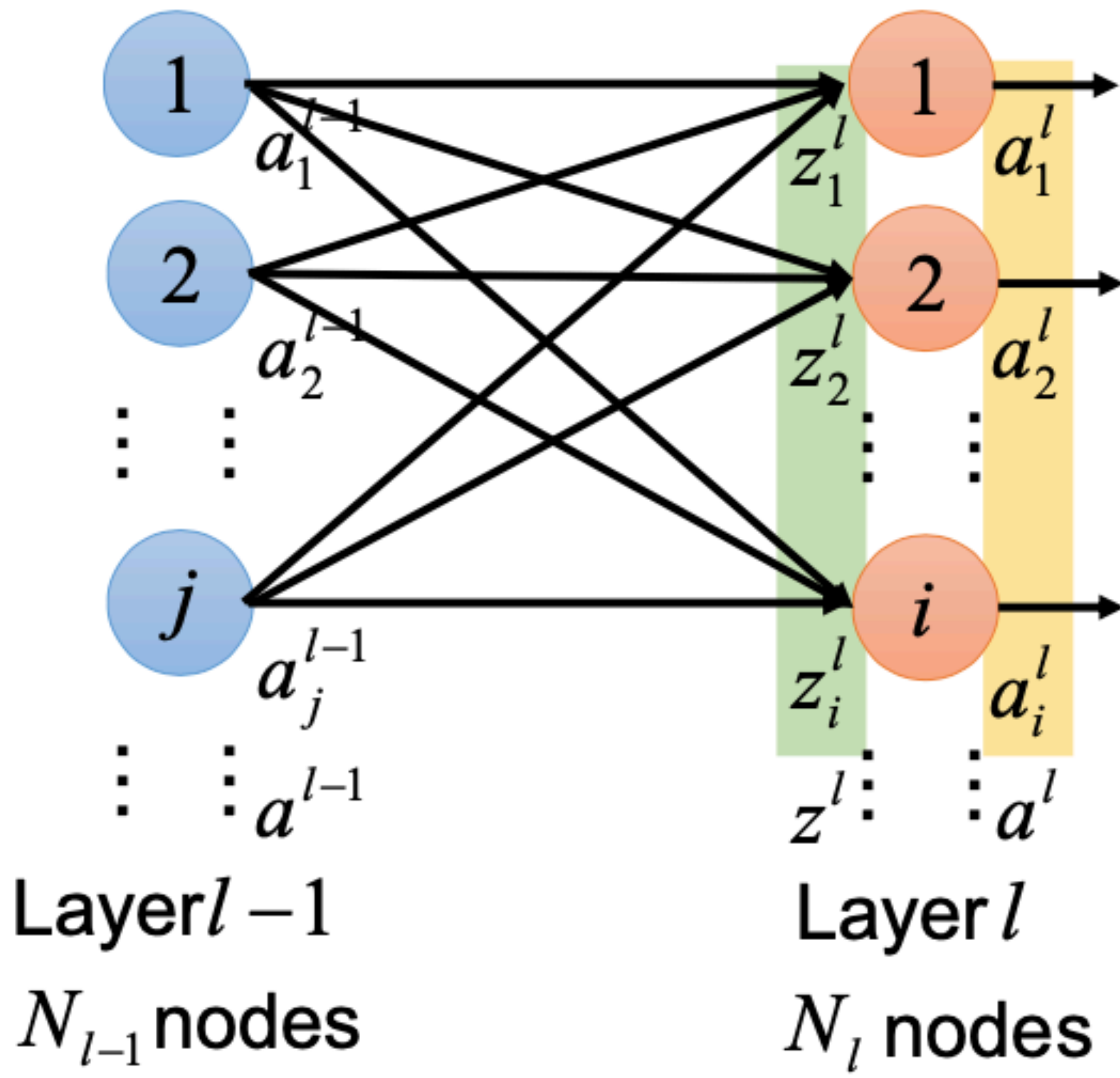
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \dots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \dots \\ \vdots & & \dots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

Layer Output Relation: from z to a

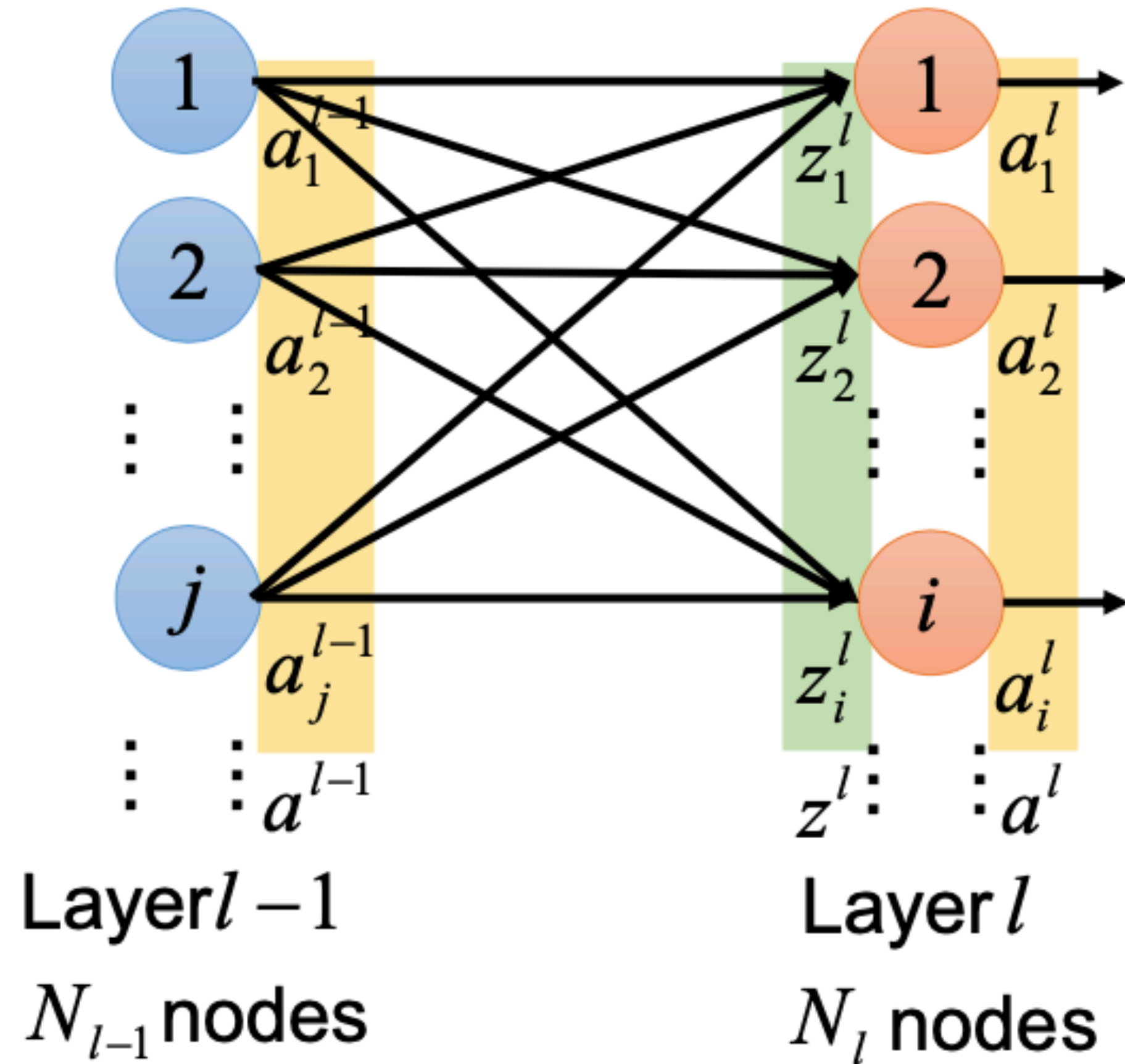


$$a_i^l = \sigma(z_i^l)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma(z_1^l) \\ \sigma(z_2^l) \\ \vdots \\ \sigma(z_i^l) \\ \vdots \end{bmatrix}$$

$$a^l = \sigma(z^l)$$

94 Layer Output Relation



$$z^l = W^l a^{l-1} + b^l$$

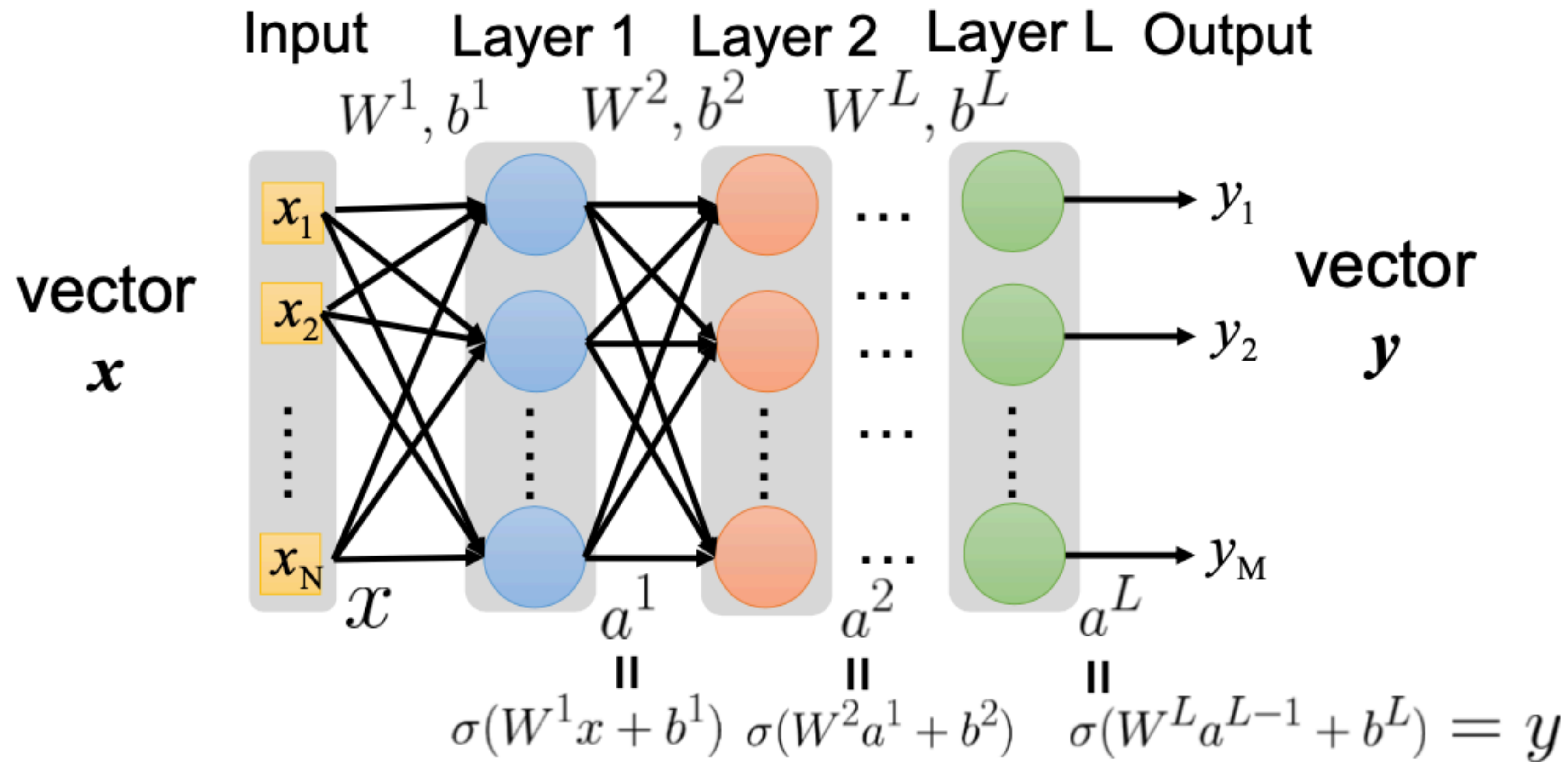
$$a^l = \sigma(z^l)$$



$$a^l = \sigma(W^l a^{l-1} + b^l)$$

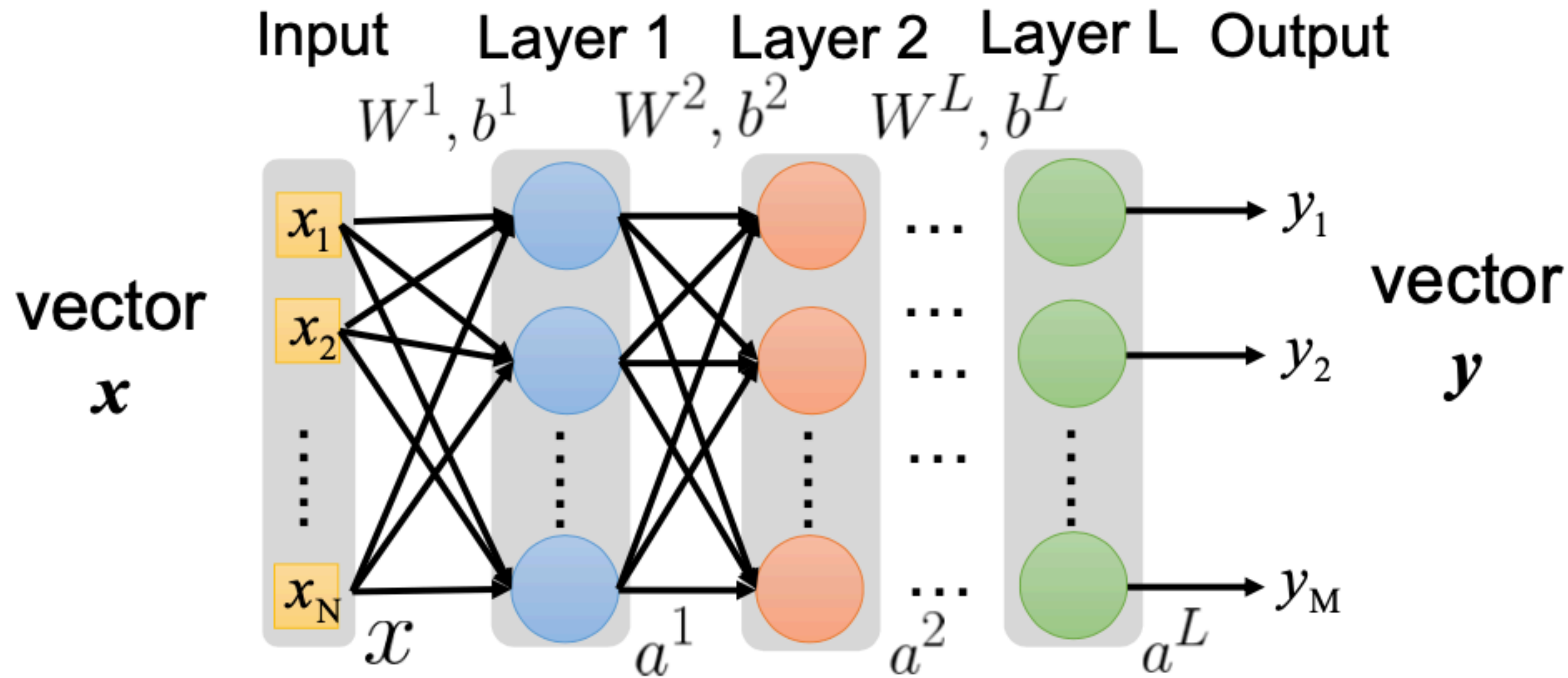
Neural Network Formulation

Fully connected feedforward network $f : R^N \rightarrow R^M$



Neural Network Formulation

Fully connected feedforward network $f : R^N \rightarrow R^M$



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

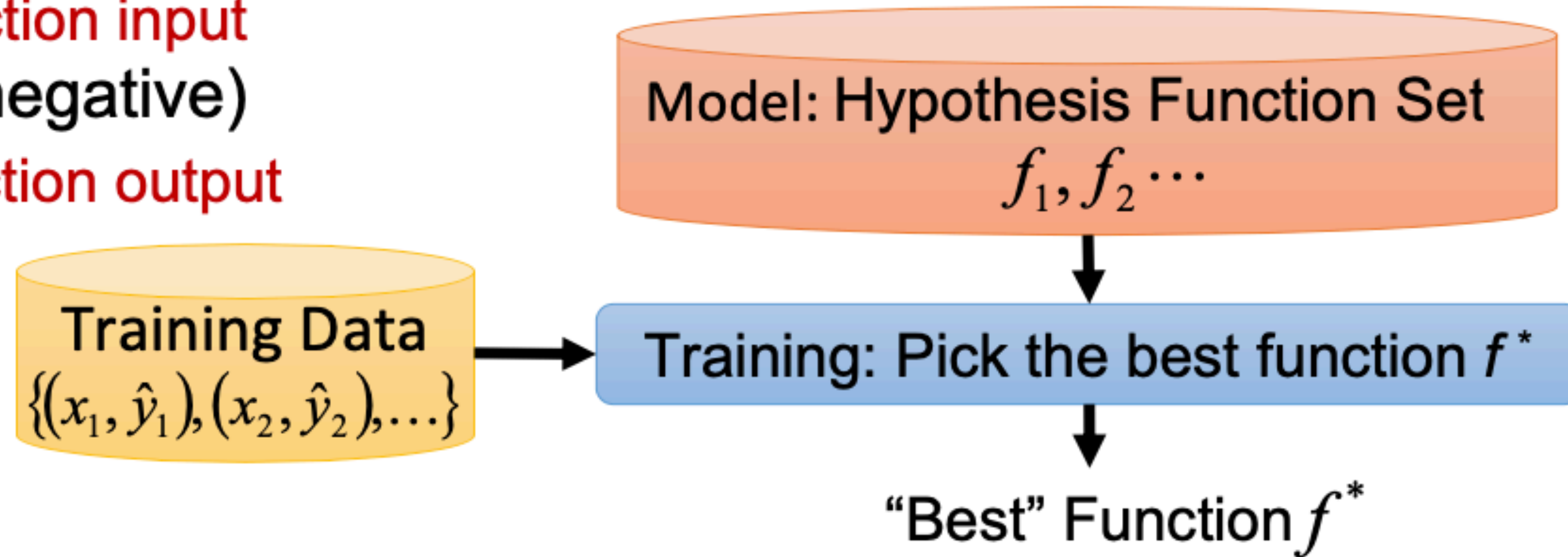
97 Loss Function for Training

x : “It claims too much.”

function input

\hat{y} : - (negative)

function output



A “Good” function: $f(x; \theta) \sim \hat{y} \Rightarrow \|\hat{y} - f(x; \theta)\| \approx 0$

Define an example loss function:
$$C(\theta) = \sum_k \|\hat{y}_k - f(x_k; \theta)\|$$

sum over the error of all training samples

98 Gradient Descent for Neural Network

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

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

 update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$

}

Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

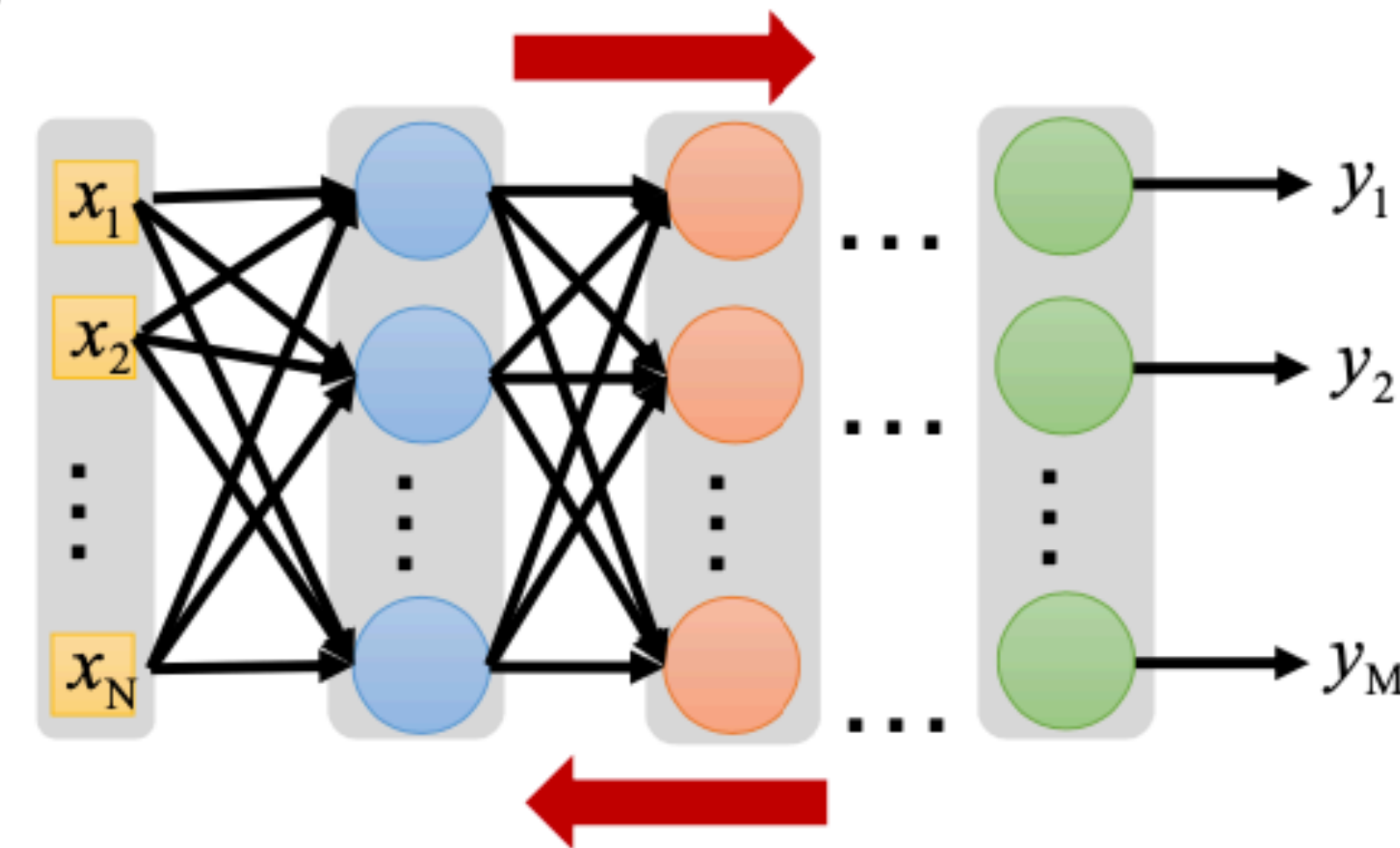
**How to efficiently
compute the
gradients?
Backpropagation**

100 Forward v.s. Backpropagation

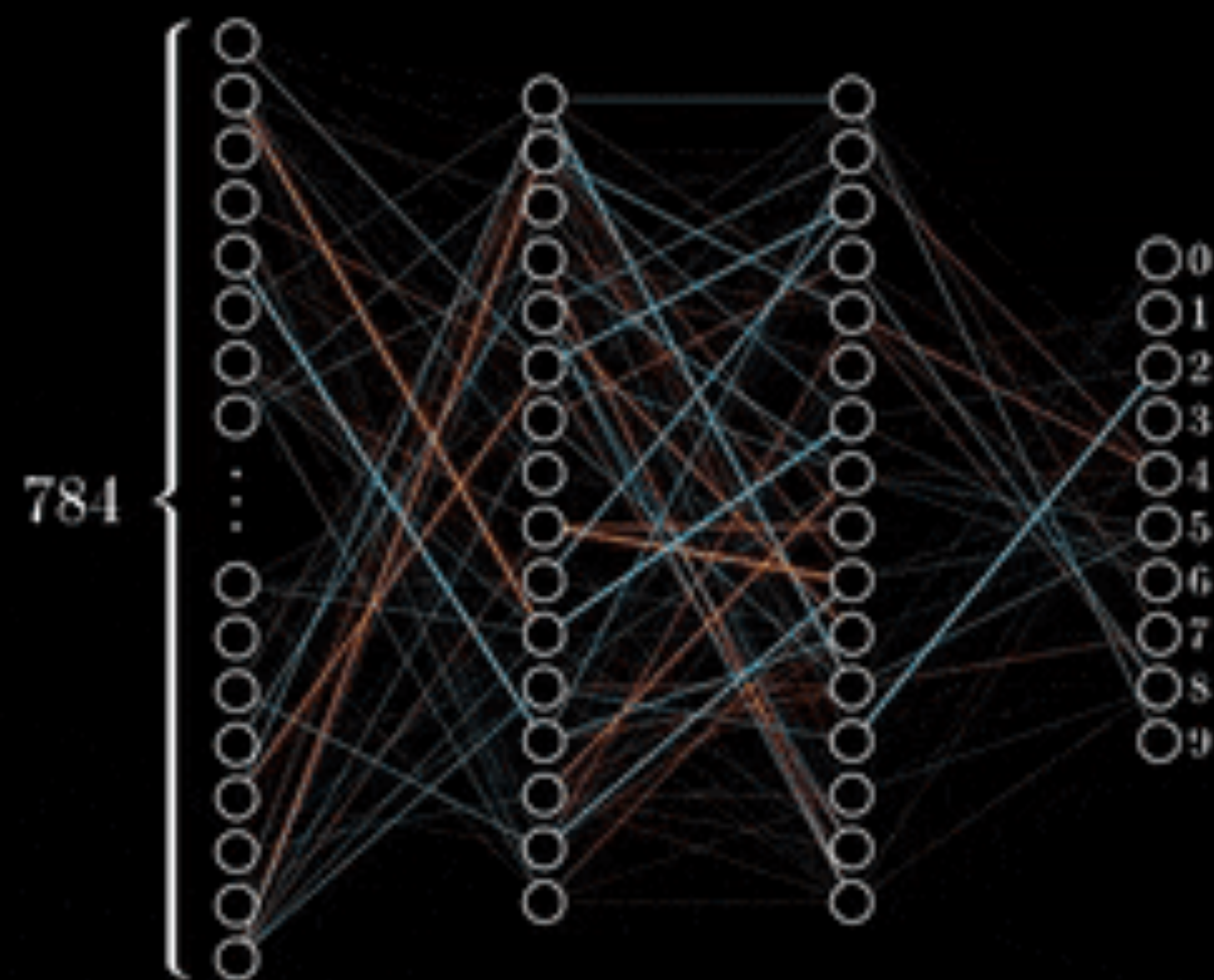
- In a feedforward neural network
 - forward propagation
 - from input x to output y information flows forward through the network
 - during training, forward propagation can continue onward until it produces a scalar cost $C(\theta)$
 - back-propagation
 - allows the information from the cost to then flow backwards through the network, in order to compute the **gradient**
 - can be applied to any function

Why backward?

Reduce the redundant computations
(same computation repeated for different parameters. e.g., w , b)



Training in
progress...



102 Chain Rule

$$\Delta w \rightarrow \Delta x \rightarrow \Delta y \rightarrow \Delta z$$

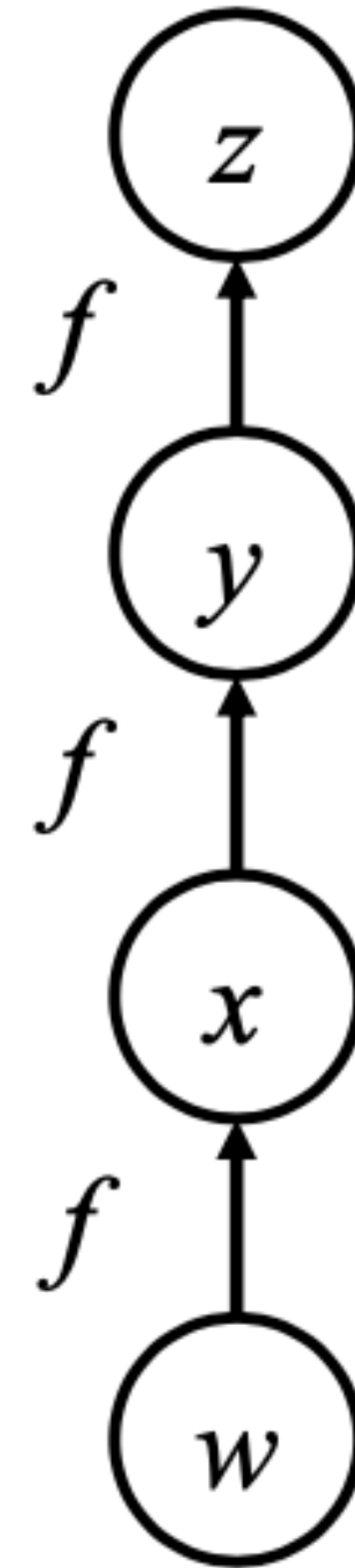
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w}$$

$$= f'(y) f'(x) f'(w)$$

$$= f'(f(f(w))) f'(f(w)) f'(w)$$

forward propagation for cost

back-propagation for gradient



Gradient Descent for Neural Networks

$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \dots \\ \vdots & & \dots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

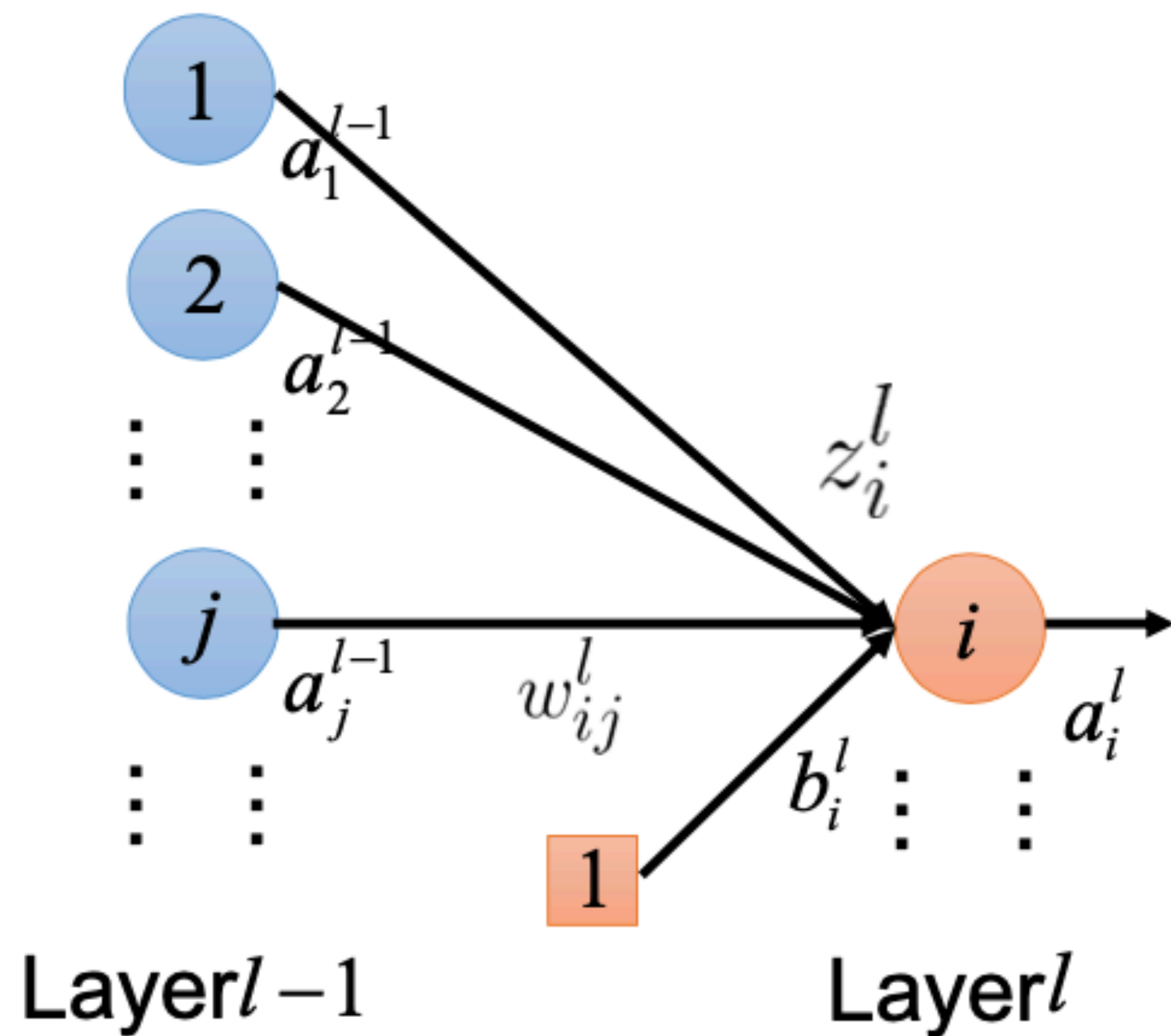
 update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$

}

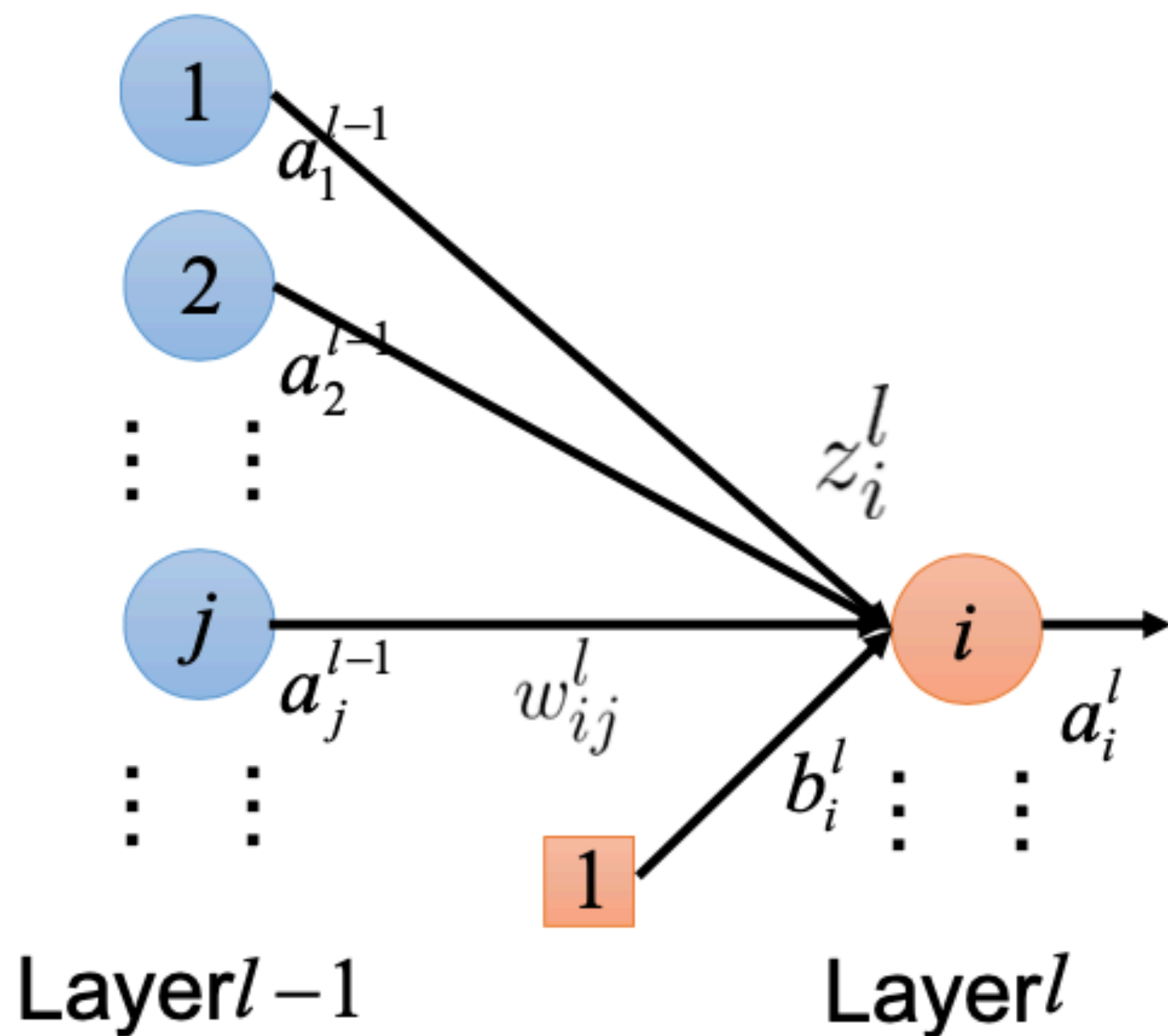
Computing the gradient includes millions of parameters.
To compute it efficiently, we use **backpropagation**.

$$\frac{\partial C(\theta)}{\partial w_{ij}^l}$$



$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} \quad (l > 1)$$

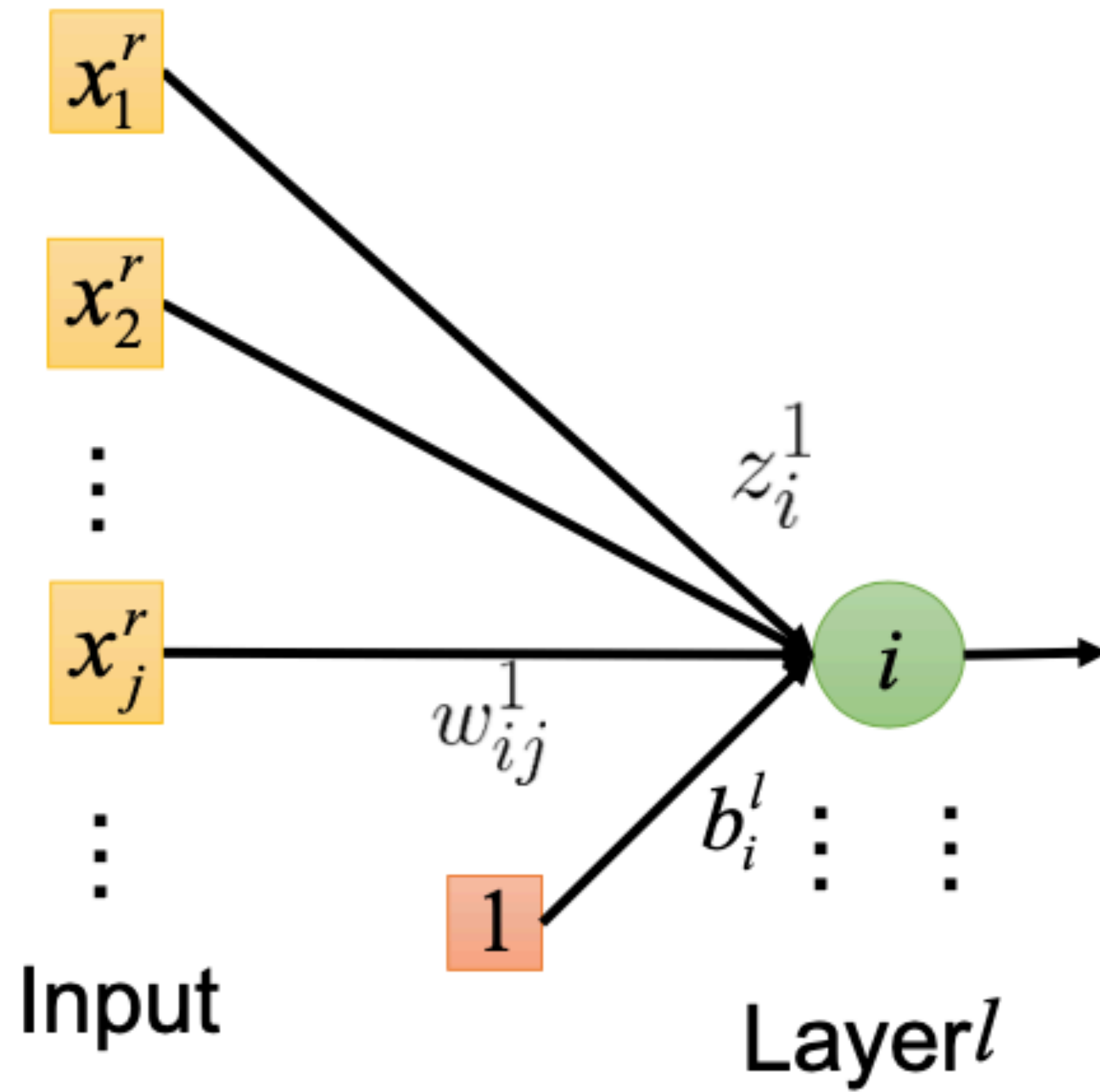


$$z^l = W^l a^{l-1} + b^l$$

$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} \quad (l = 1)$$

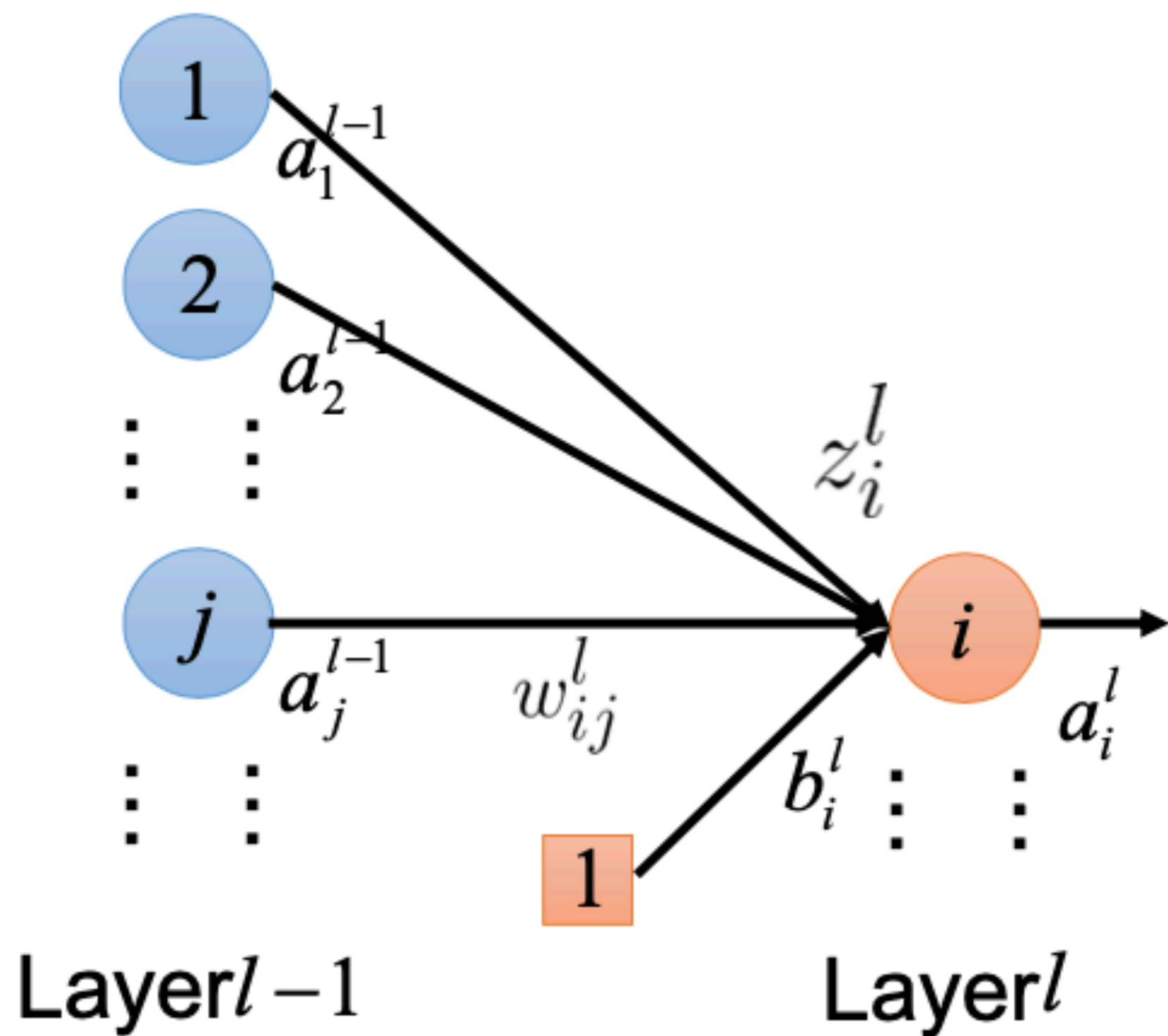


$$z^1 = W^1 x + b^1$$

$$z_i^1 = \sum_j w_{ij}^1 x_j + b_i^1$$

$$\frac{\partial z_i^1}{\partial w_{ij}^1} = x_j$$

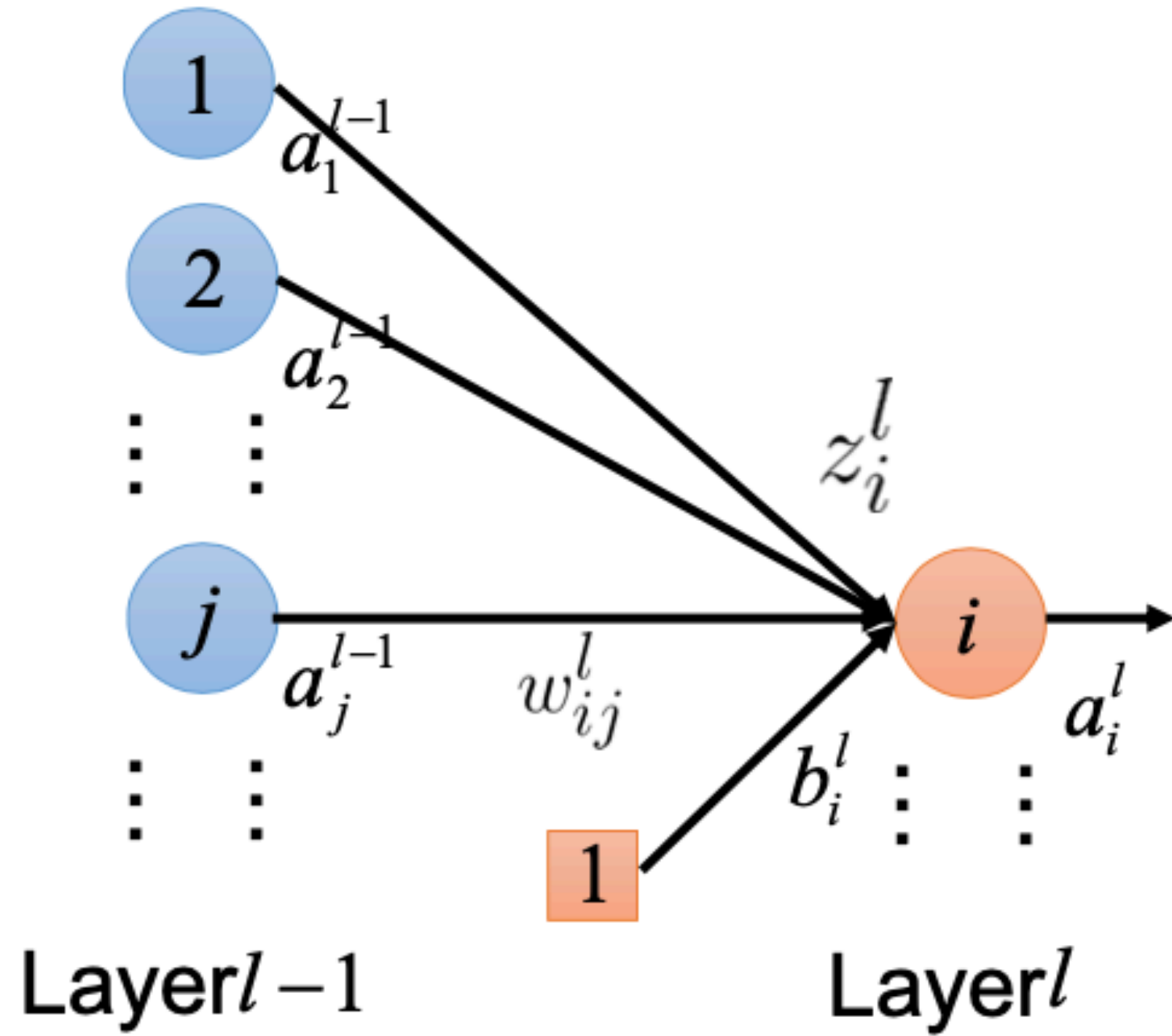
$$\frac{\partial C(\theta)}{\partial w_{ij}^l}$$



$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$

$$\frac{\partial C(\theta)}{\partial w_{ij}^l}$$

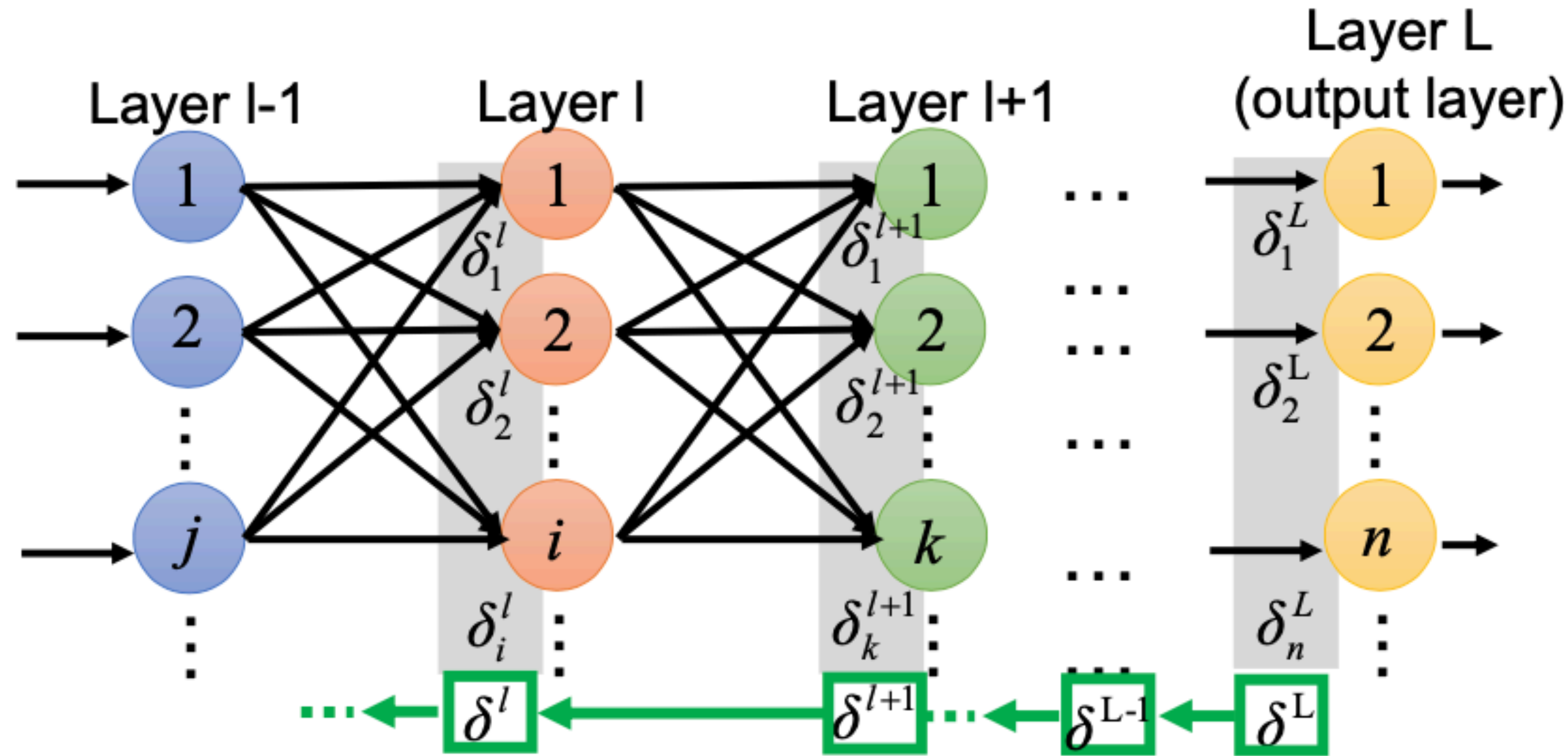


$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial C(\theta)}{\partial z_i^l}$$

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l : the propagated gradient corresponding to the l -th layer



Idea: computing δ^l layer by layer (from δ^L to δ^1) is more efficient

$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

- Idea: from L to 1
 - ① Initialization: compute δ^L
 - ② Compute δ^l based on δ^{l+1}

$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

● Idea: from L to 1

① **Initialization: compute δ^L**

② Compute δ^l based on δ^{l+1}

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} \quad \Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$
$$= \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^L}$$

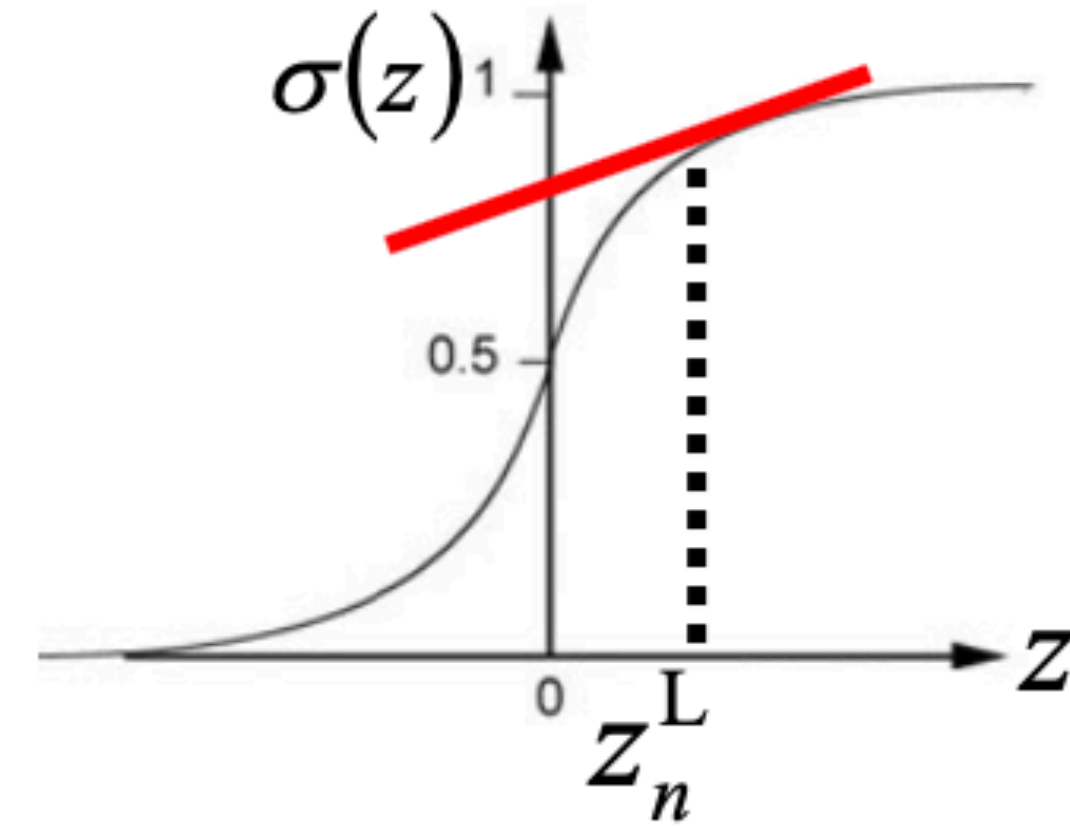
$\partial C / \partial y_i$ depends on the loss function

$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

● Idea: from L to 1

① Initialization: compute δ^L

② Compute δ^l based on δ^{l+1}



$$\delta_i^L = \frac{\partial C}{\partial z_i^L} \quad \Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

$$= \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^L} = a_i^L = \sigma(z_i^L) \quad \sigma'(z^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_i^L) \\ \vdots \end{bmatrix} \quad \nabla C(y) = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_i} \\ \vdots \end{bmatrix}$$

$$= \frac{\partial C}{\partial y_i} \sigma'(z_i^L)$$

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

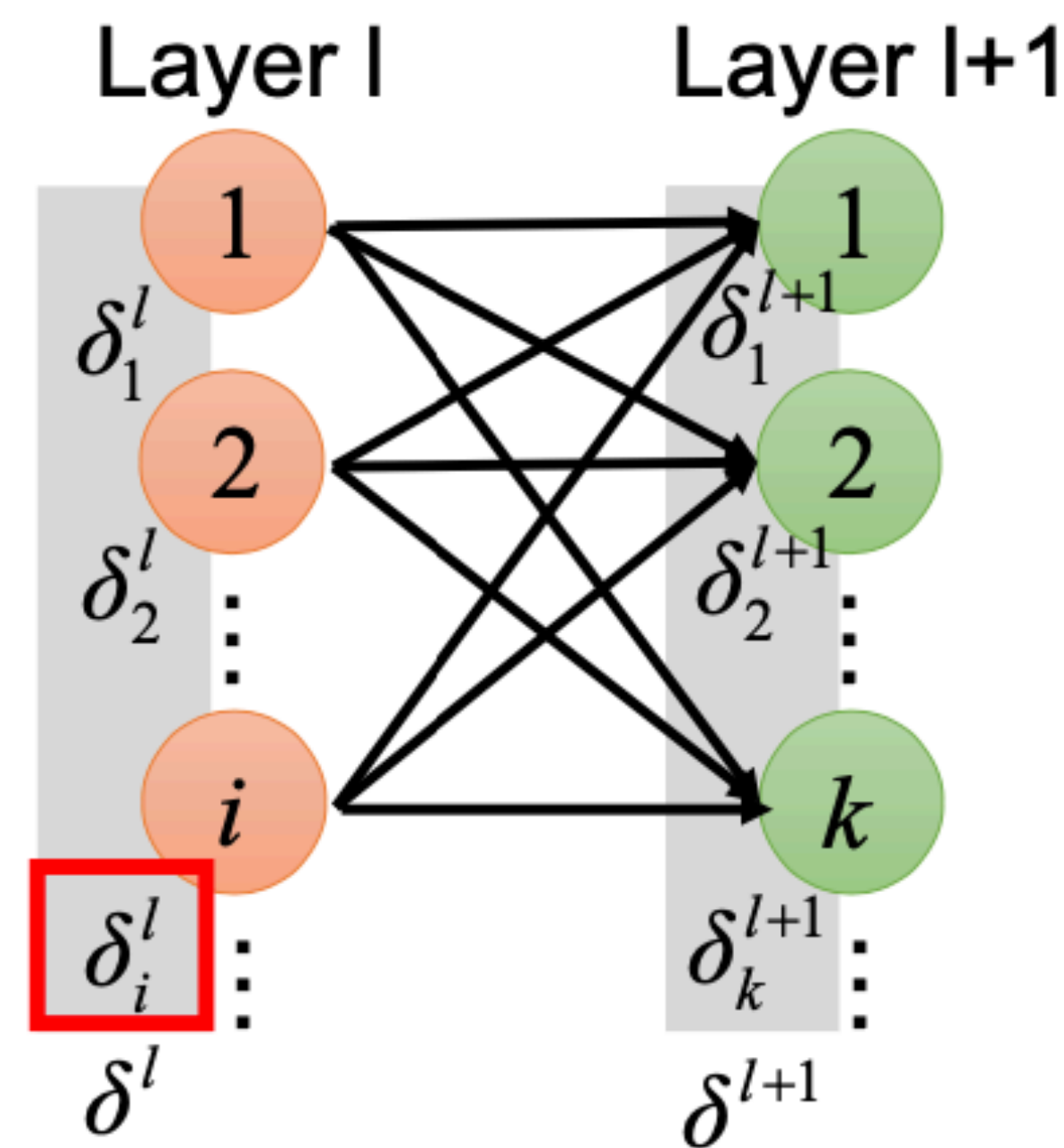
$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

- Idea: from L to 1
 - ① Initialization: compute δ^L
 - ② **Compute δ^l based on δ^{l+1}**

$$\Delta z_i^l \rightarrow \Delta a_i^l \rightarrow \begin{matrix} \Delta z_1^{l+1} \\ \Delta z_2^{l+1} \\ \vdots \\ \Delta z_k^{l+1} \end{matrix} \rightarrow \Delta C$$

$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \right)$$

$$= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \right) \delta_i^{l+1}$$

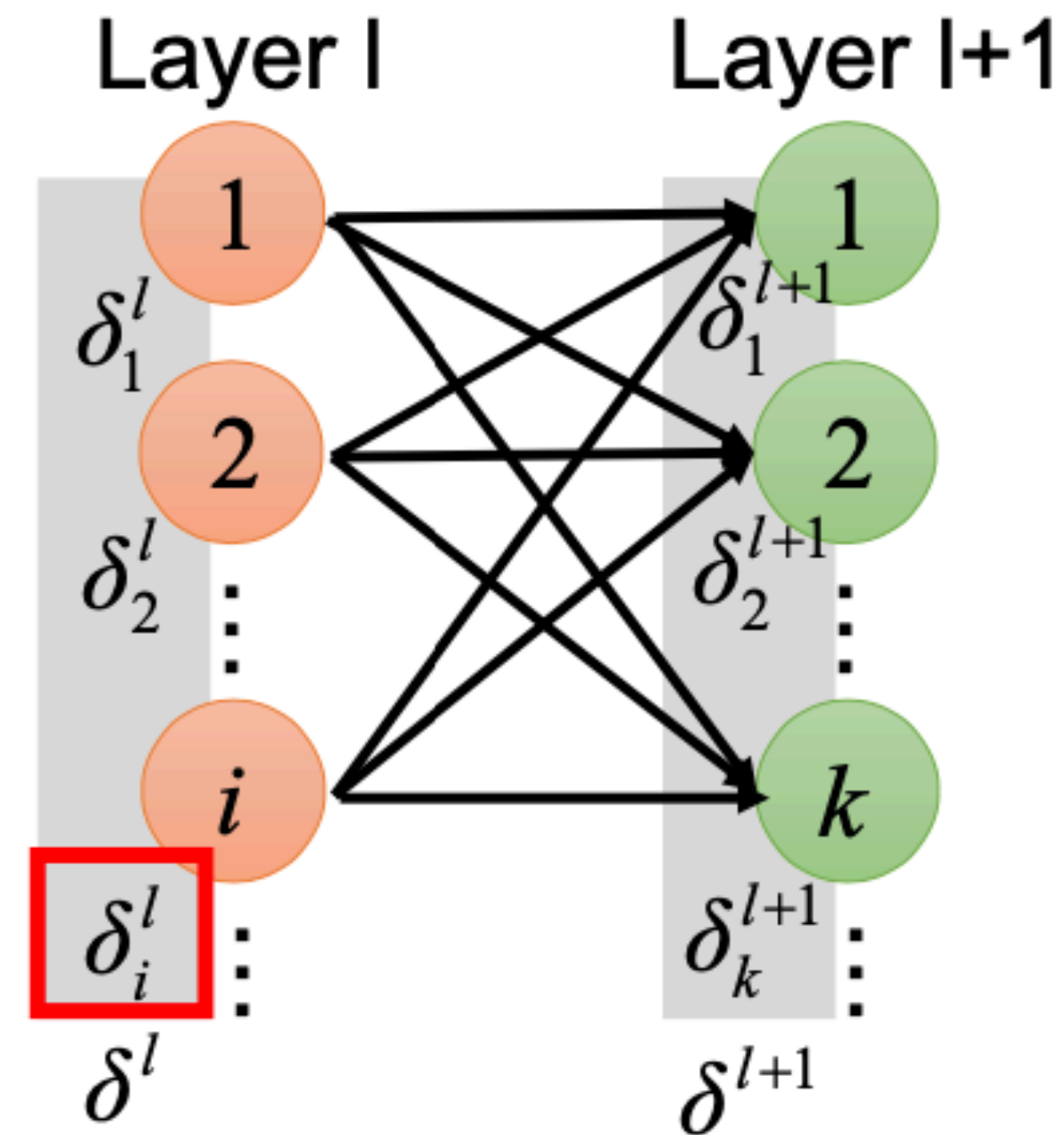
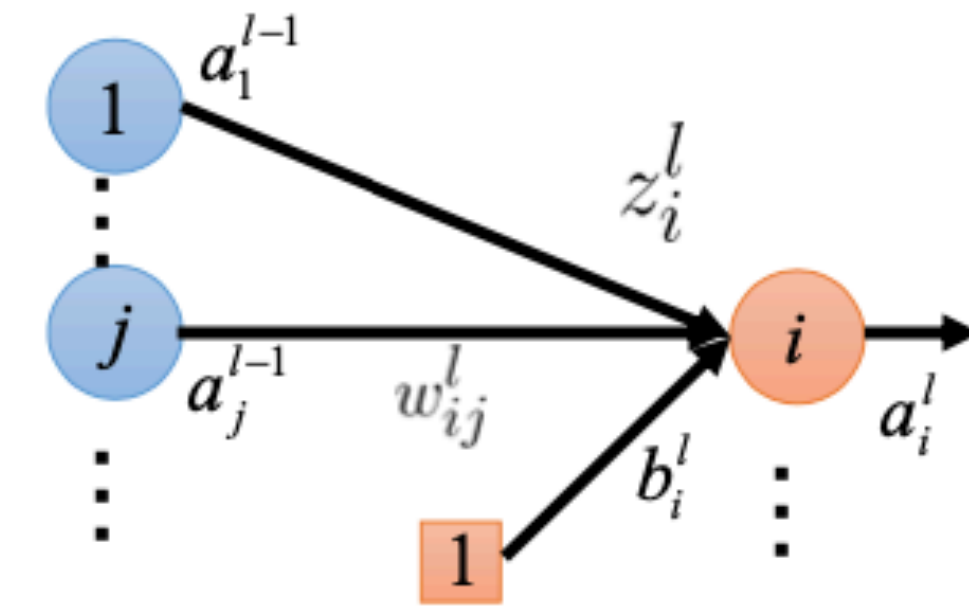


$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

● Idea: from L to 1

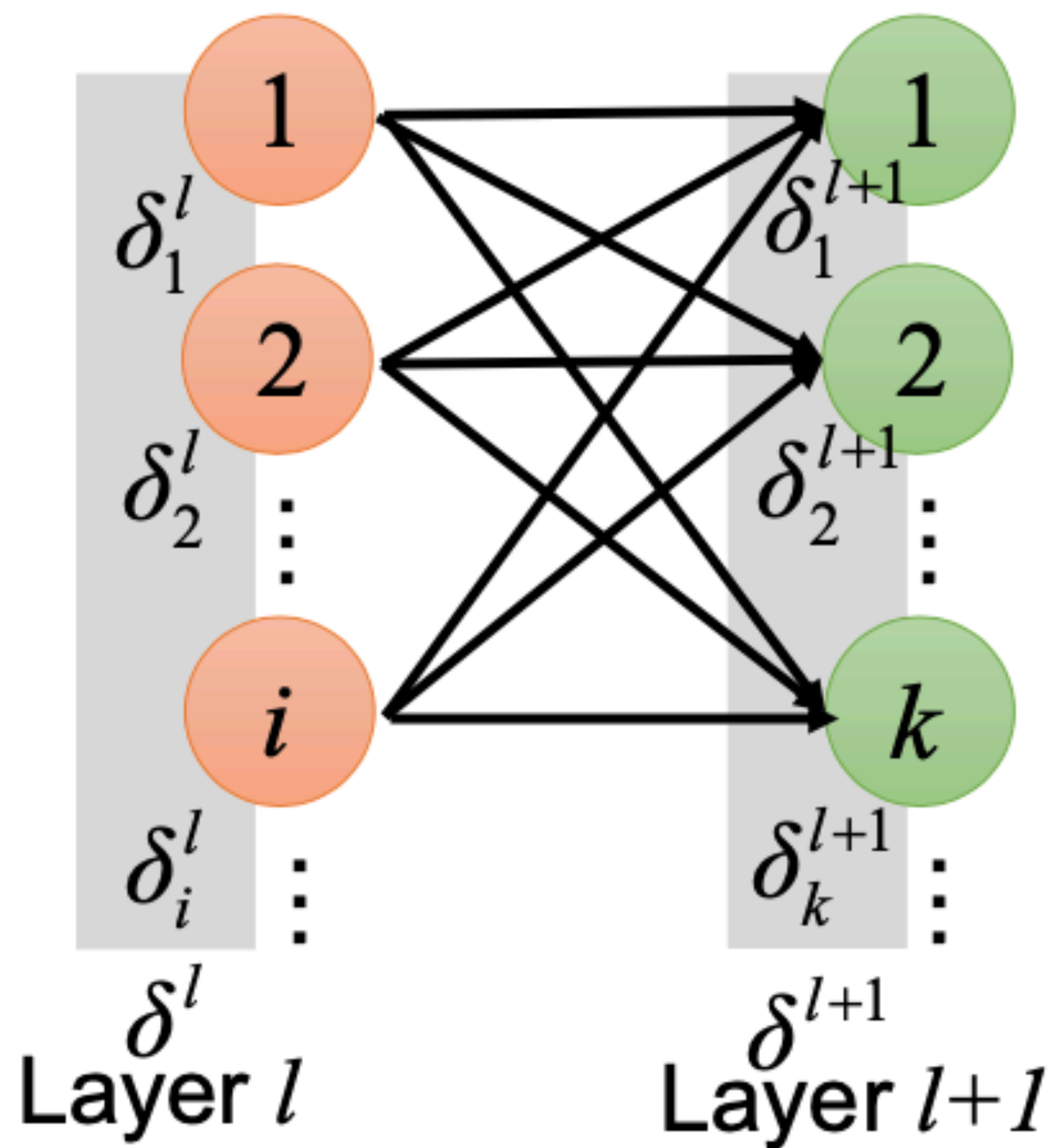
- ① Initialization: compute δ^L
- ② **Compute δ^l based on δ^{l+1}**

$$\begin{aligned} \delta_i^l &= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1} \\ &= \sigma'(z_i) \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1} \\ &= \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1} \end{aligned}$$

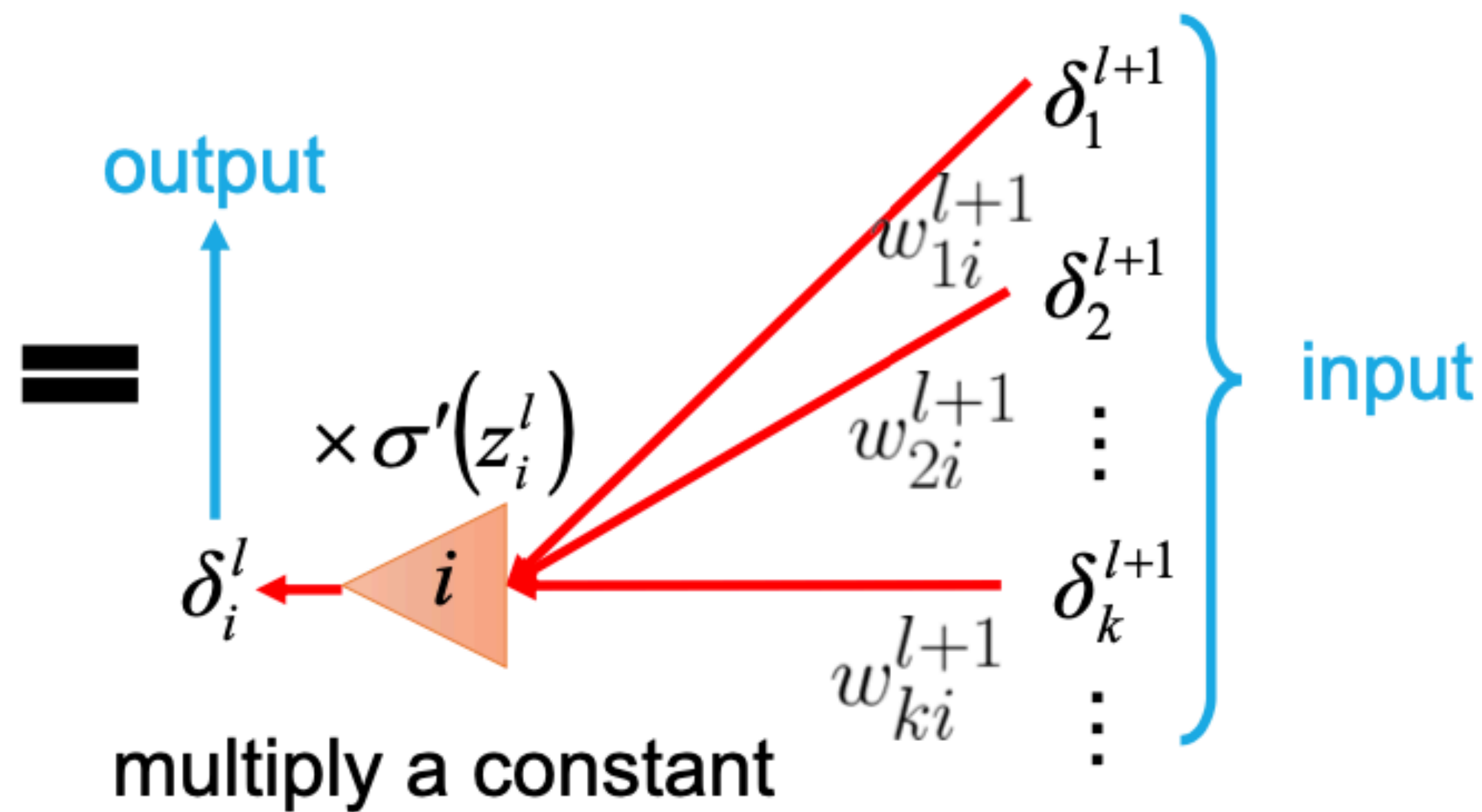


$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

🕒 Rethink the propagation



$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

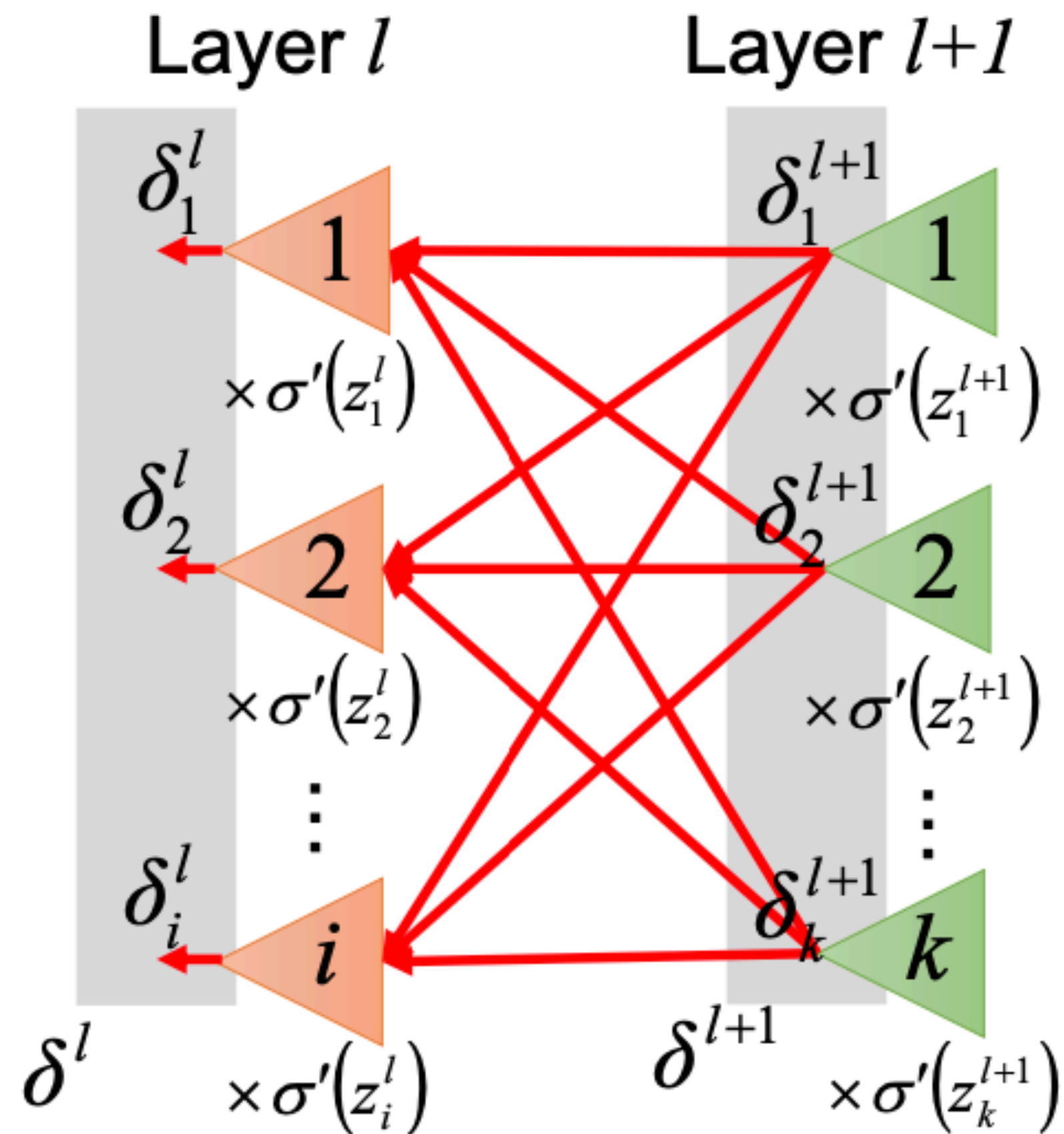


$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

$$\sigma'(z^l) = \begin{bmatrix} \sigma'(z_1^l) \\ \sigma'(z_2^l) \\ \vdots \\ \sigma'(z_i^l) \\ \vdots \end{bmatrix}$$

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$



$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

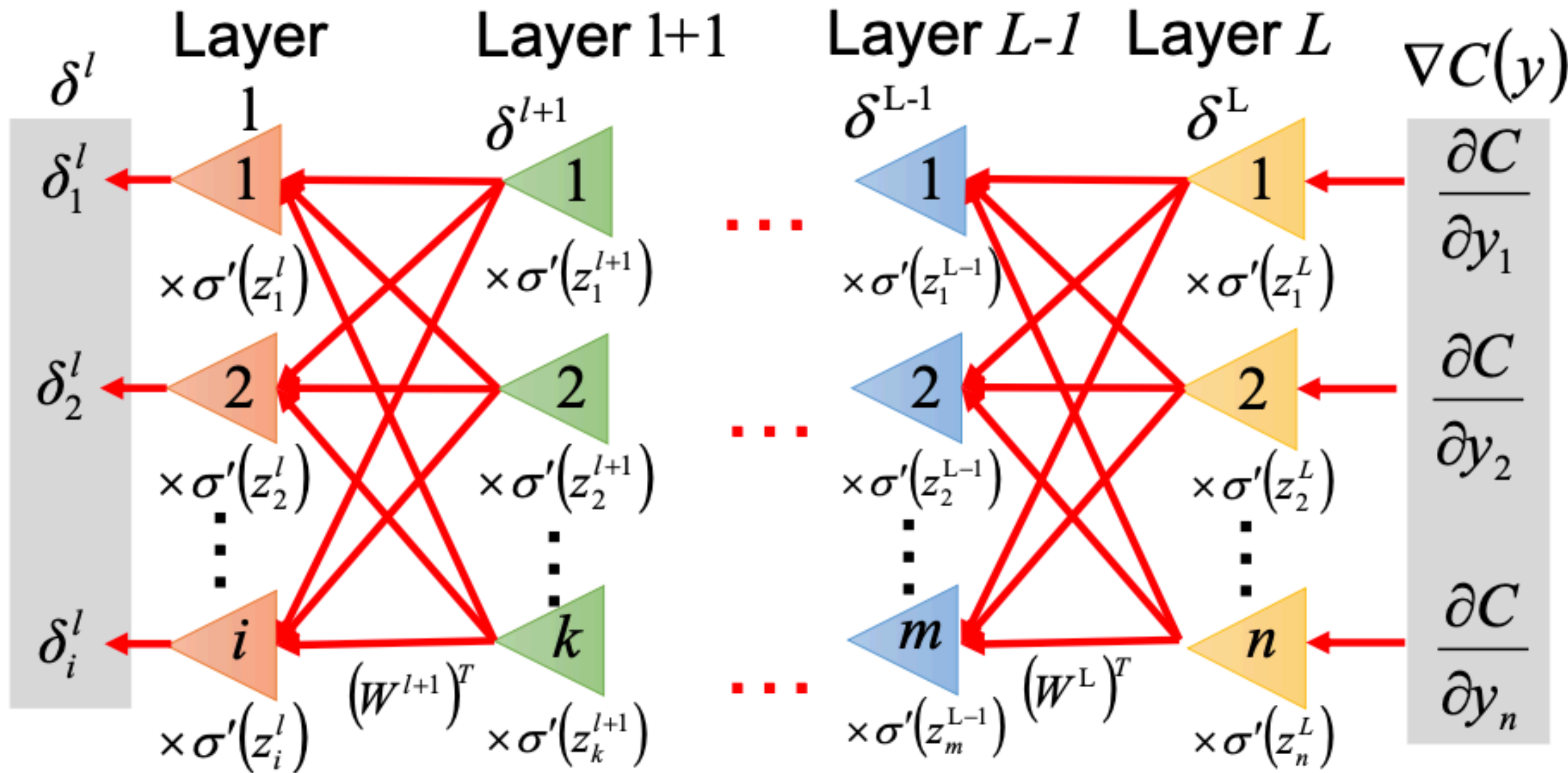
$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

Idea: from L to 1

- ① Initialization: compute δ^L
- ② Compute δ^{l-1} based on δ^l

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$



Backpropagation

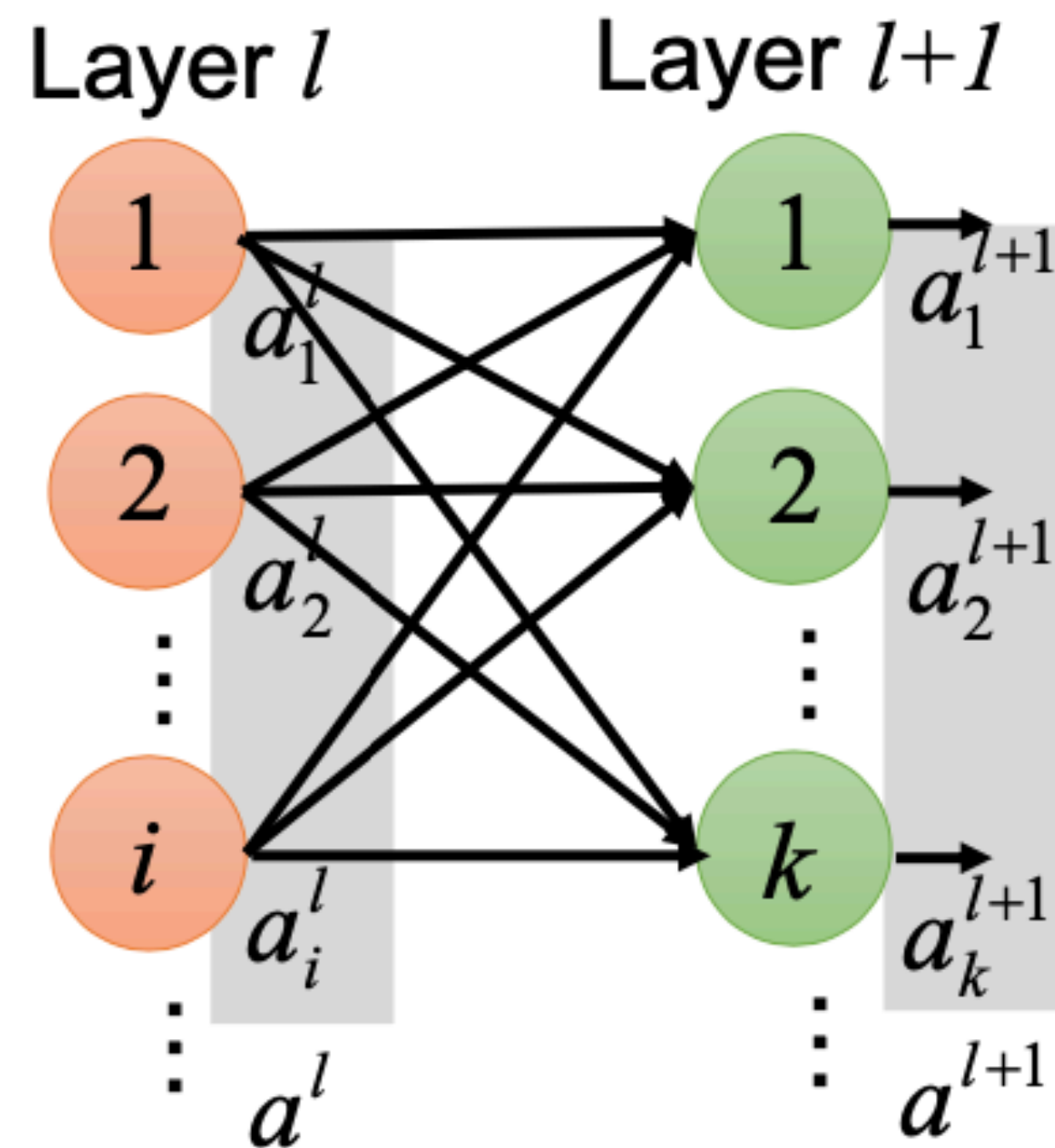
$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$

Forward Pass

$$z^1 = W^1 x + b^1 \quad a^1 = \sigma(z^1)$$

$$z^l = W^l a^{l-1} + b^l \quad a^l = \sigma(z^l)$$



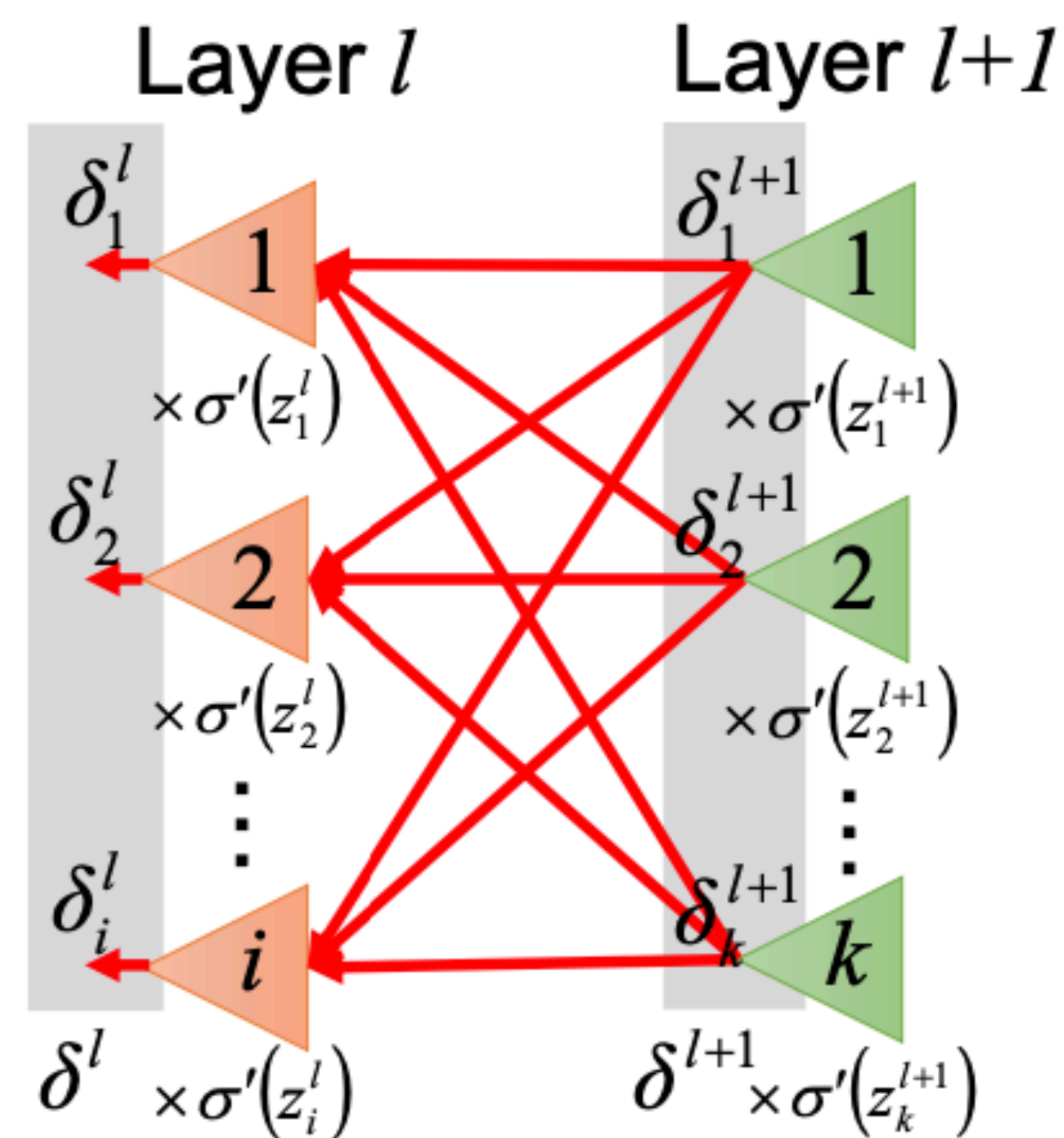
Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

Backward Pass

$$\begin{aligned} \delta^L &= \sigma'(z^L) \odot \nabla C(y) \\ \delta^{L-1} &= \sigma'(z^{L-1}) \odot (W^L)^T \delta^L \\ &\vdots \\ \delta^l &= \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1} \\ &\vdots \end{aligned}$$



Gradient Descent for Optimization

- Remember: our problem is we have too many partial derivatives to calculate for different w

$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \dots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

 compute gradient at θ^i

 update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_{\theta} C(\theta^i)$

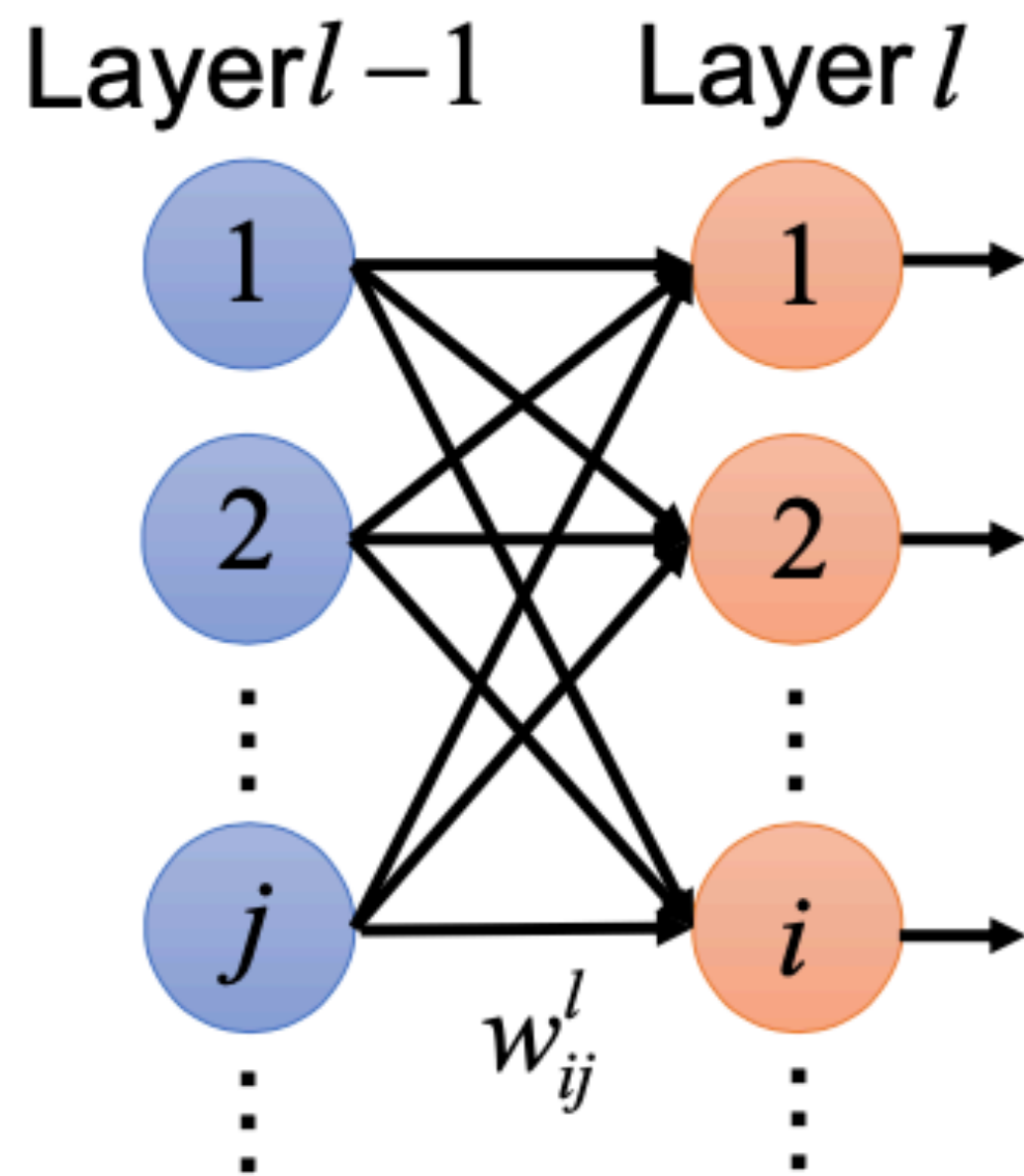
}

Concluding Remarks

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\delta_i^l$$

$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$



Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$

⋮

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$

⋮

Forward Pass

$$z^1 = W^1 x + b^1$$

$$a^1 = \sigma(z^1)$$

⋮

$$z^l = W^l a^{l-1} + b^l$$

$$a^l = \sigma(z^l)$$

⋮

Compute the gradient based on two pre-computed terms from backward and forward passes

Todo

- **Reading assignment 1:** Deep learning
<https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf>
Due date: 23:59 pm, January 21th, 2021
- **Recommended reading:** Automatic Differentiation in Machine Learning: a Survey
<https://jmlr.org/papers/v18/17-468.html>
- **Next lecture:** Recurrent Neural Networks and Language Modeling

Candidate course projects

**You can have your
own proposal**

124 Project 1: SQuAD Question Answering

- SQuAD competition: <https://rajpurkar.github.io/SQuAD-explorer/>
- Objectives
 - Get familiar with the question answering task in NLP;
 - Implement one or several QA models;
 - Test some novel ideas: e.g., data augmentation, model visualization and analysis, model compression, multi-task learning, knowledge augmentation, etc.
- We will review your work based on the idea, novelty, implementation and so on, not based on your rank on the leaderboard.

Project 2: Question Generation

- ◎ Generating questions based on the input context with/without the answer
- ◎ The input can be multimodal, e.g., visual question generation
- ◎ Objectives
 - Investigate text generation and evaluation methods, with emphasize on question generation tasks;
 - Implement at least one QG method on a specific dataset;
 - Investigate how to improve the quality of questions, evaluation metrics, combine with QA, etc.
- ◎ Reference papers include but not limited to the following:
 - <http://www-labs.iro.umontreal.ca/~liubang/files/bliu-www20.pdf>
 - <http://www-labs.iro.umontreal.ca/~liubang/files/yCheng-acl21.pdf>

Project 3: Event Causality Identification (ECI)

- ◎ Identifying causal relations of events in texts. For example, in a sentence: “The **earthquake** generates a **tsunami** that rose up to 135 feet”, an ECI system should identify that a causal relationship holds between the two mentioned events.
- ◎ Reference papers:
 - Gao, Lei, Prafulla Kumar Choubey, and Ruihong Huang. "Modeling document-level causal structures for event causal relation identification." Proc. NAACL 2019.
 - Liu, Jian, Yubo Chen, and Jun Zhao. "Knowledge enhanced event causality identification with mention masking generalizations." Proc. IJCAI 2021.
 - Phu, Minh Tran, and Thien Huu Nguyen. "Graph Convolutional Networks for Event Causality Identification with Rich Document-level Structures." Proc. NAACL 2021.
- ◎ Datasets:
 - EventStoryLine: "The event storyline corpus: A new benchmark for causal and temporal relation extraction." Proceedings of the Events and Stories in the News Workshop. 2017.
 - Causal-Timebank: "Catena: Causal and temporal relation extraction from natural language texts." COLING 2016.

Project 4: Story Forest

- ◎ **Story Forest System** is originally implemented in Java and it aims to cluster documents into fine-grained events (a cluster of documents talking about the same event), and organize events by tree structure to track and visualize related events into stories (which called **story tree**).
- ◎ Objectives:
 - Reimplement the story forest system by Python: write modularized, well-organized code, so that different components in it can be easily replaced with better ideas;
 - Apply it to a English news articles;
 - Visualize the results;
 - Try your own ideas in implementing the system, no need to follow the original paper.
- ◎ Reference:
 - <http://www-labs.iro.umontreal.ca/~liubang/files/bliu-tkdd20.pdf>
 - <https://github.com/BangLiu/StoryForest>

Project 5: Text to Image Generation

- Generate image based on textual inputs.
- The input can be keywords, sentence descriptions, etc.
- The output can be realistic images, human faces, anime images, etc.
- Objective:
 - Survey a text-to-image task;
 - Implement a baseline published in recent years;
 - Test some novel ideas, e.g., controllable text-to-image generation, story generation (generate several images that forms a story), 3D image generation, etc.

More...

- ⊙ Music/poetry/novel/story generation
- ⊙ Machine translation
- ⊙ Chatbot
- ⊙ Document understanding
- ⊙ Electronic Health Records analysis
- ⊙ Mental health prediction based on social media posts
- ⊙ Visual question answering
- ⊙ Embodied question answering
- ⊙ ...

What you should do next

- Find a topic you are interested in: what we have proposed, your own proposal, tasks on this website: <https://nlpprogress.com/>, and so on.
- Define your objective: re-implement an existing but not open-sourced algorithm? Improving an existing method? Design a new task/dataset/model? Take part in a competition? etc.
- Choose one of a few most related articles to read, and form your baselines (can be changed after proposal). Determine your dataset, evaluation criteria, etc. You can read the proposal instruction.
- Write your proposal with your team.
 - Can come and discuss with me during the office hours

Thanks! Q&A

Bang Liu

Email: bang.liu@umontreal.ca

Homepage: <http://www-labs.iro.umontreal.ca/~liubang/>

Github: <https://github.com/BangLiu/>