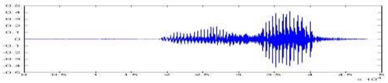
Applied Deep Learning

Neural Network Basics

March 10th, 2020 http://adl.miulab.tw

國立臺灣大學 National Taiwan University

Learning ≈ Looking for a Function

- Speech Recognition $f($  $) =$ "你好"
- Handwritten Recognition $f($  $) =$ "2"
- Weather forecast $f($ Thursday $) =$ "  Saturday"
- Play video games $f($  $) =$ "move left"

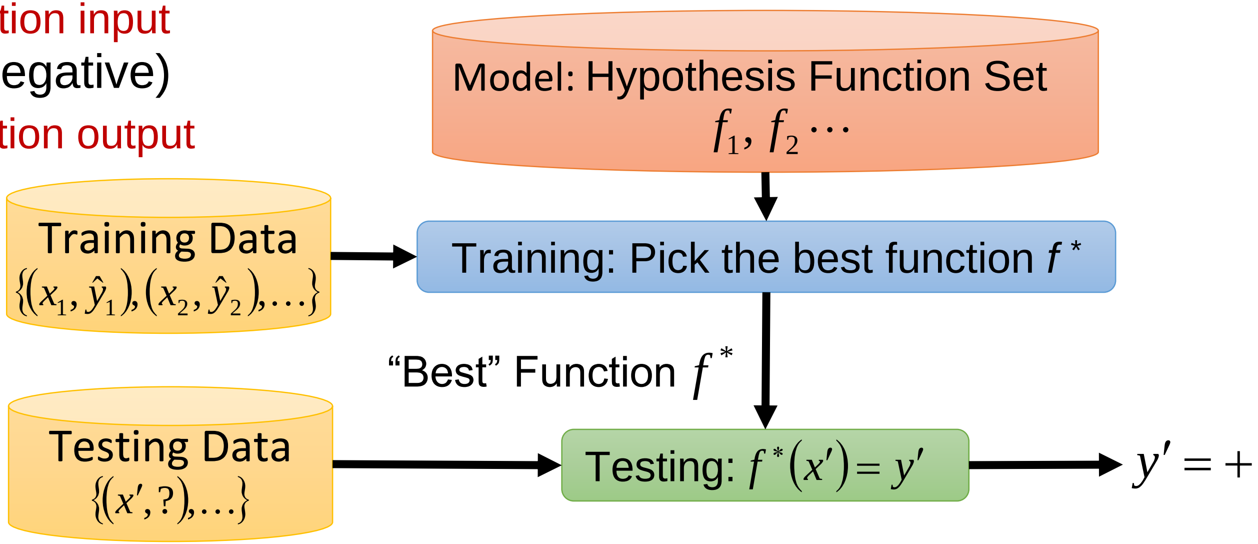
Machine Learning Framework

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 &
Resources

4

How to Train a Model?

實際上我們是如何訓練一個模型的？

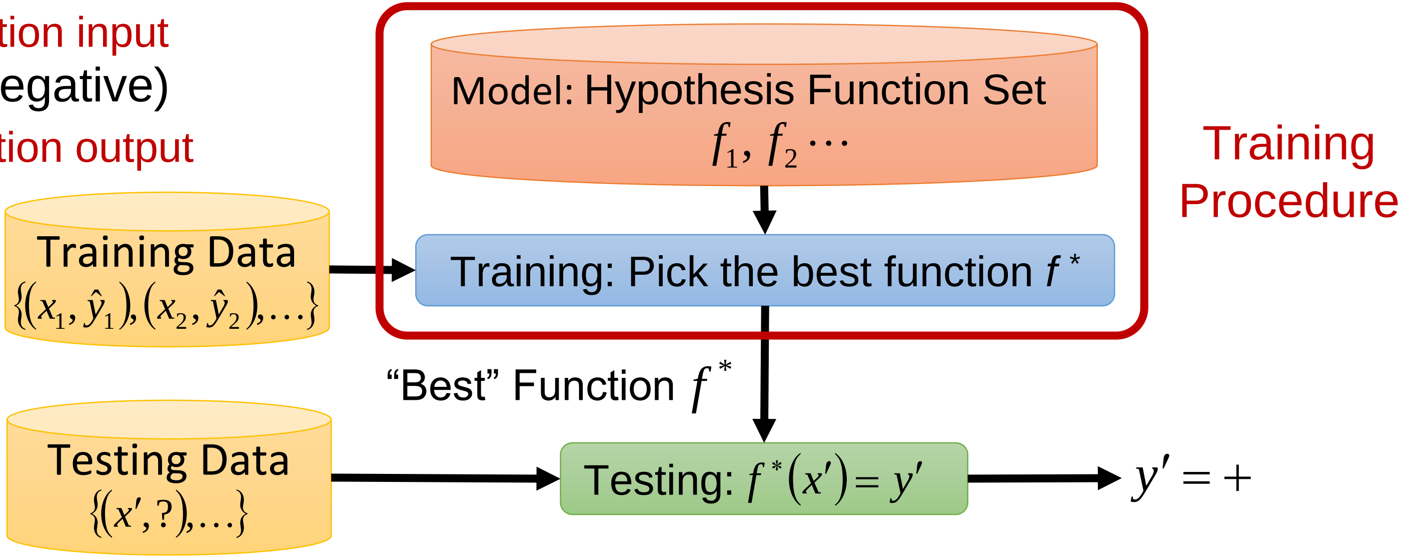
Machine Learning Framework

x : "It claims too much."

function input

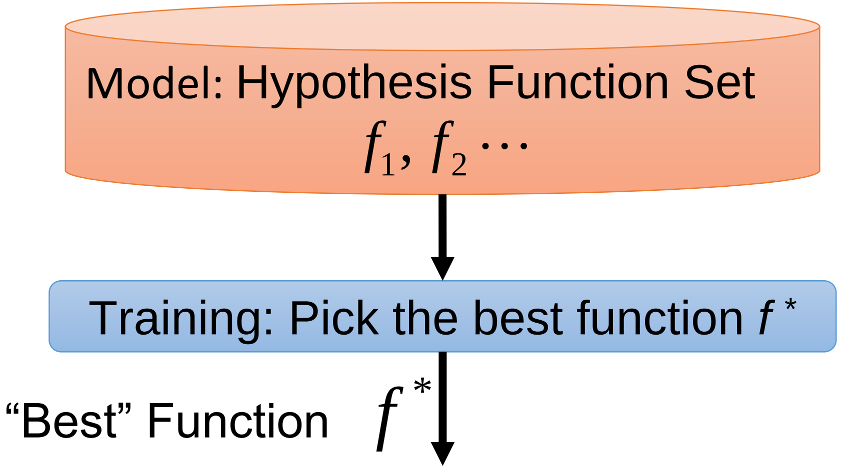
$\hat{y}$: - (negative)

function output



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

6 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?

Training Procedure Outline

① Model Architecture

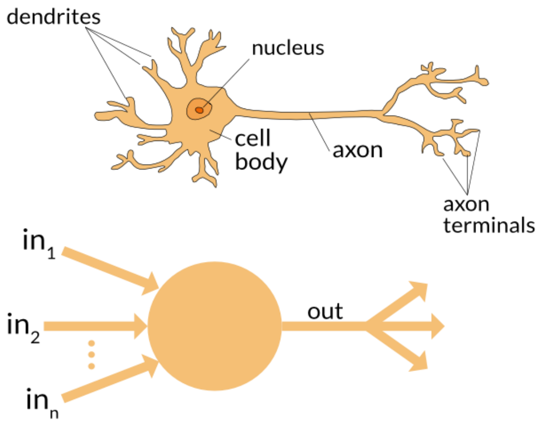
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design

- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips



8

What is the Model?

什麼是模型？

Training Procedure Outline

① Model Architecture

- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design

- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

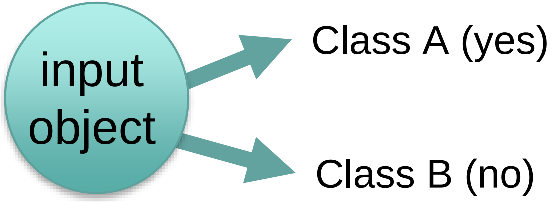
Classification Task

- Sentiment Analysis
- Speech Phoneme Recognition
- Handwritten Recognition

“這規格有誠意!” → +

“太爛了吧~” → -

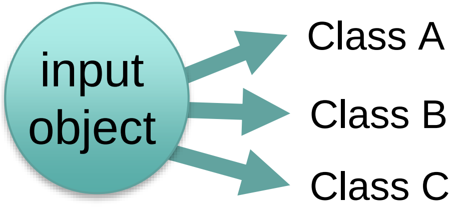
→ Binary Classification



 → /h/

 → 2

→ Multi-class Classification



Some cases are not easy to be formulated as classification problems

Target Function

- Classification Task

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

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

→ a N-dim vector
→ a M-dim vector

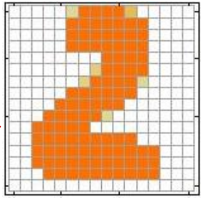
Assume both x and y can be represented as fixed-size vectors

Vector Representation Example

- Handwriting Digit Classification

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

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

y: class/label

10 dimensions for digit recognition

$$\begin{array}{cc} \text{“1”} & \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{array}{c} \text{“1”} \\ \text{“2”} \\ \text{“3”} \\ \end{array} \end{array} \qquad \begin{array}{cc} \text{“2”} & \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} & \begin{array}{l} \text{“1”} \rightarrow \text{“1” or not} \\ \text{“2”} \rightarrow \text{“2” or not} \\ \text{“3”} \rightarrow \text{“3” or not} \\ \end{array} \end{array}$$

Vector Representation Example

- Sentiment Analysis

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

x: word

y: class/label

"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

3 dimensions
(positive, negative, neutral)

$$\begin{array}{cc} \text{“+”} & \text{“-”} \\ \downarrow & \downarrow \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

"+" → "+" or not

"-" → "-" or not

"?" → "?" or not

Target Function

- Classification Task

$$f(x) = y \quad \Longrightarrow \quad f: R^N \to R^M$$

- x: input object to be classified → a N-dim vector
- y: class/label → a M-dim vector

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

Training Procedure Outline

① Model Architecture

- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

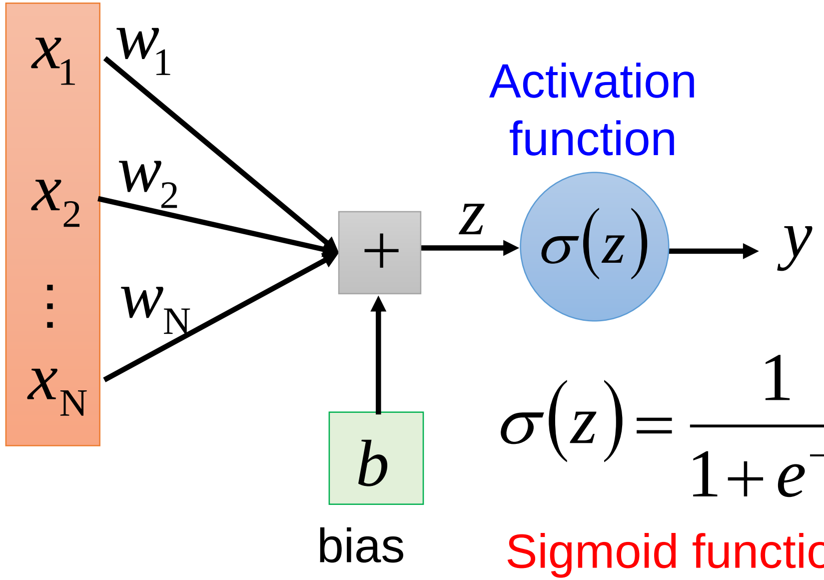
② Loss Function Design

- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

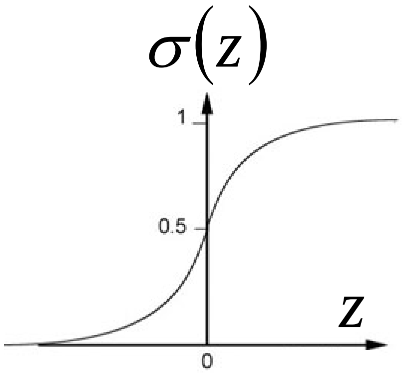
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

A Single Neuron



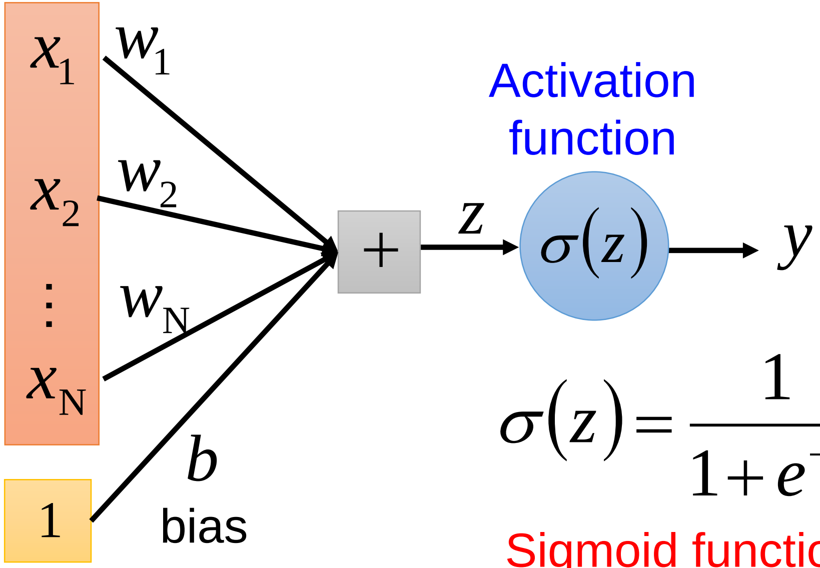
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Sigmoid function



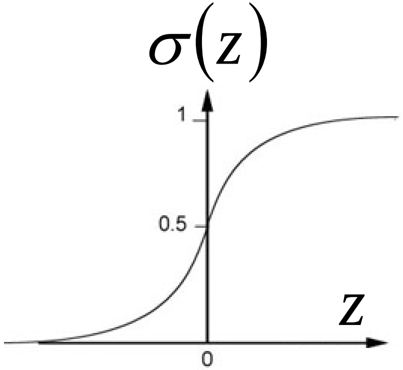
Each neuron is a very simple function

A Single Neuron



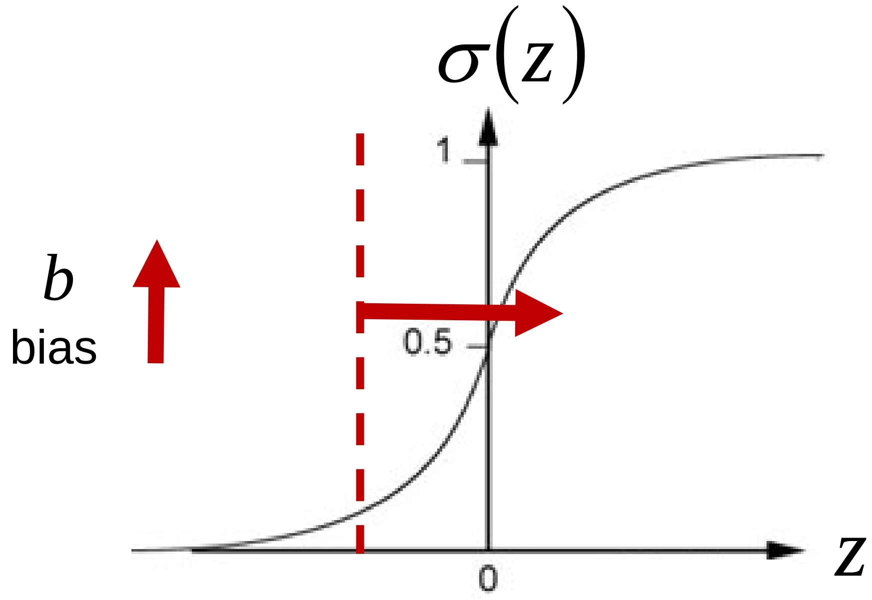
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Sigmoid function



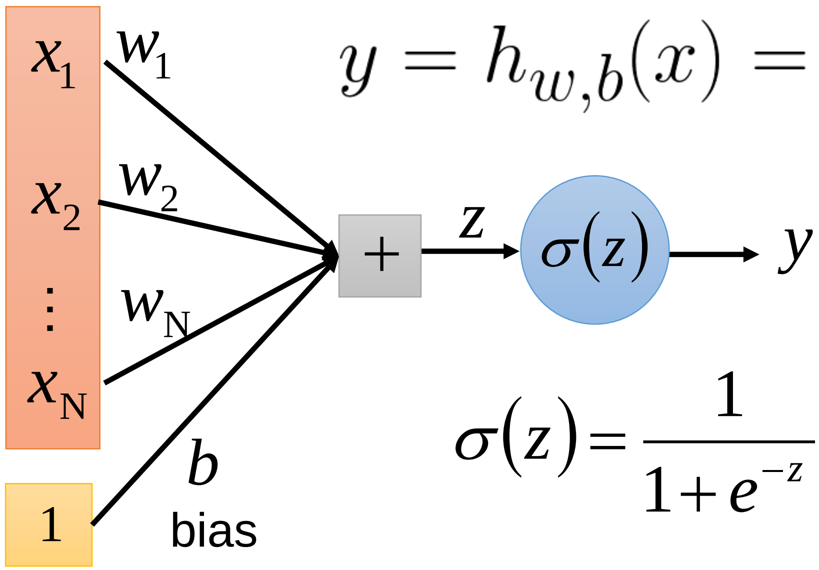
The bias term is an “always on” feature

Why Bias?



The bias term gives a class prior

Model Parameters of A Single Neuron

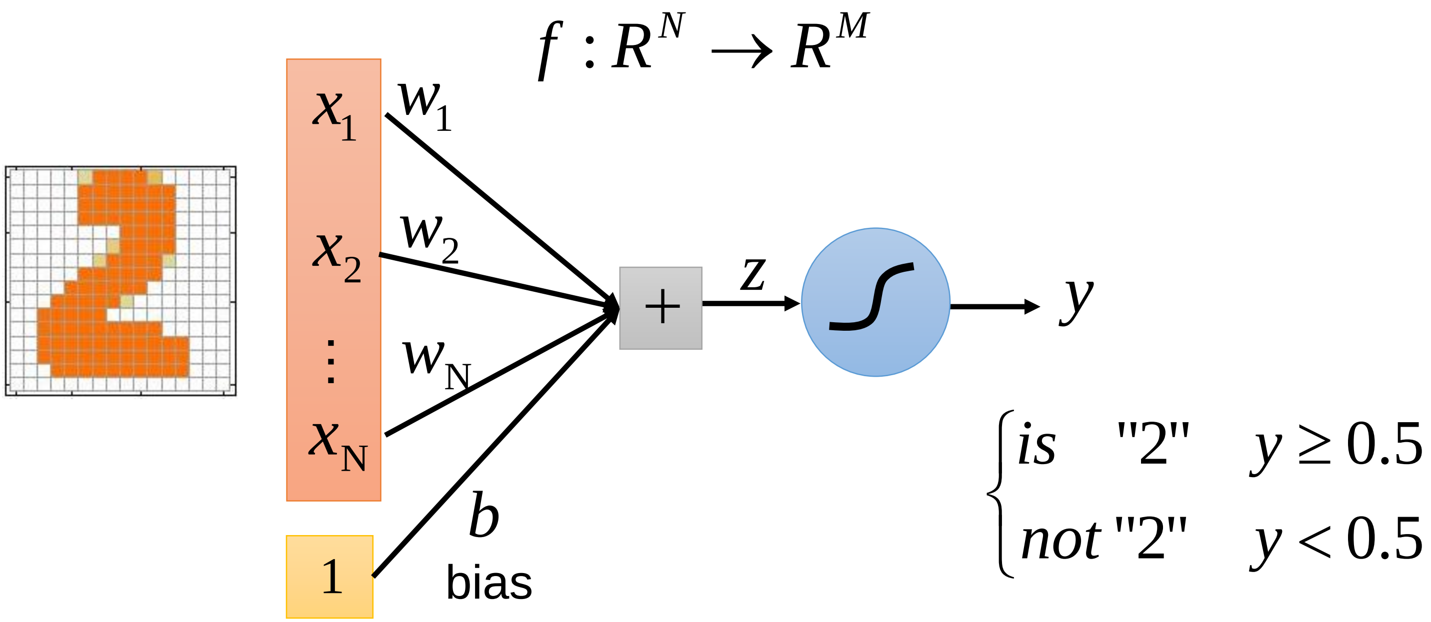


$$y = h_{w,b}(x) = \sigma(w^T x + b)$$

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

w, b are the parameters of this neuron

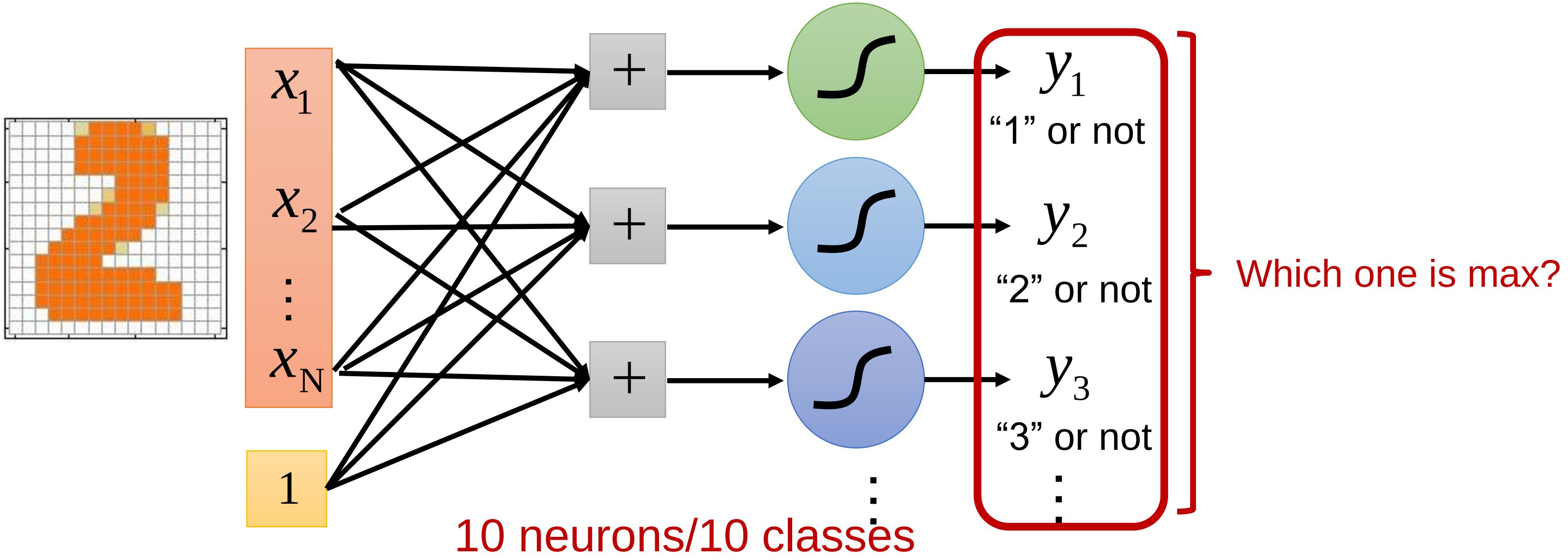
A Single Neuron



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

Training Procedure Outline

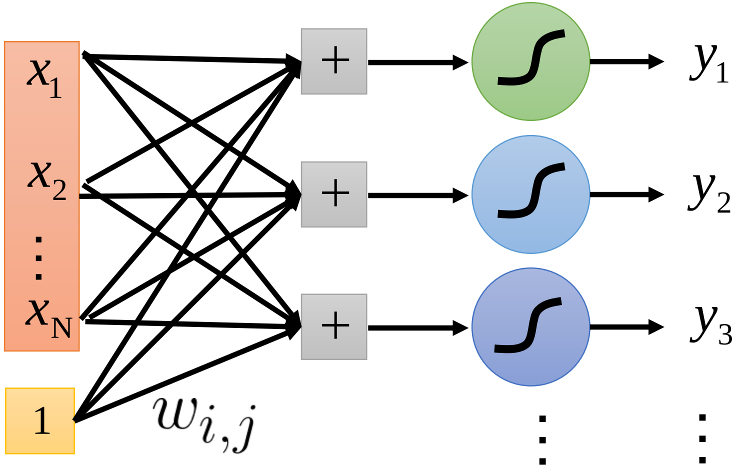
① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

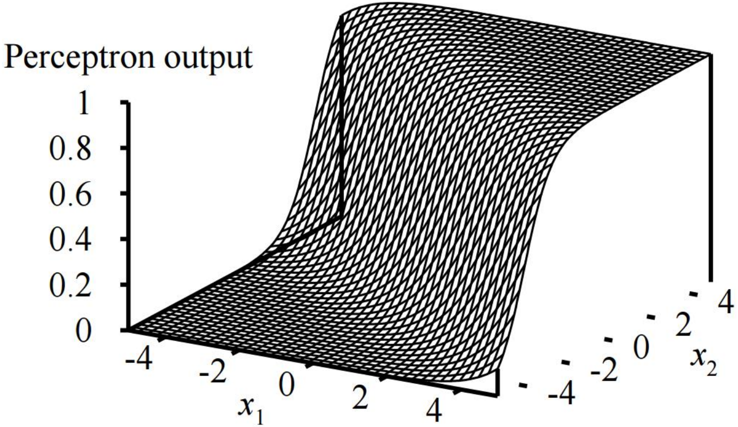
② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

A Layer of Neurons – Perceptron

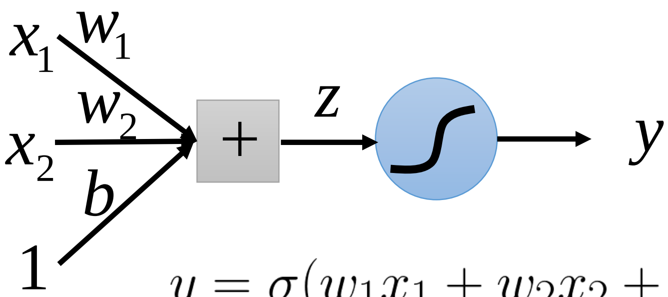
- Output units all operate separately – no shared weights





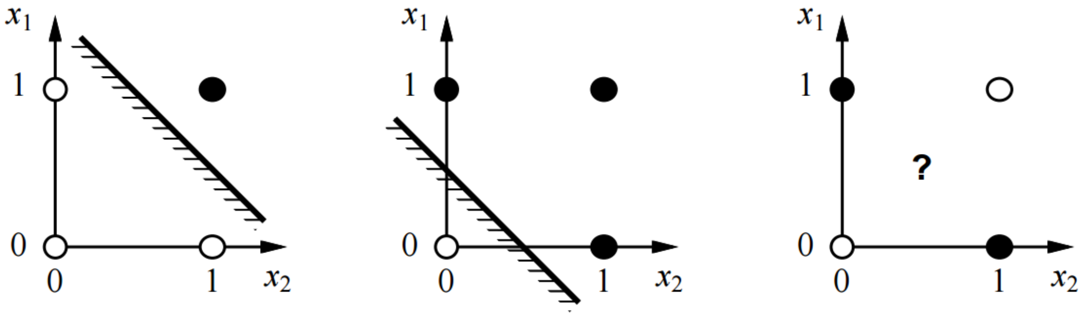
Adjusting weights moves the location, orientation, and steepness of cliff

Expression of Perceptron



$$\begin{cases} \text{yes} & y \geq \theta \ (\text{threshold}) \\ \text{no} & y < \theta \end{cases}$$

$$y = \sigma(w_1 x_1 + w_2 x_2 + b)$$

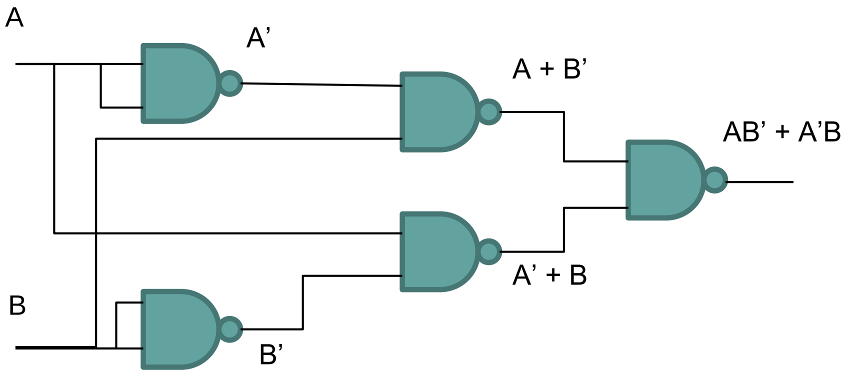


(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

A perceptron can represent AND, OR, NOT, etc., but not XOR → linear separator

How to Implement XOR?

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



A xor B = AB’ + A’B

Multiple operations can produce more complicate output

Training Procedure Outline

① Model Architecture

- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

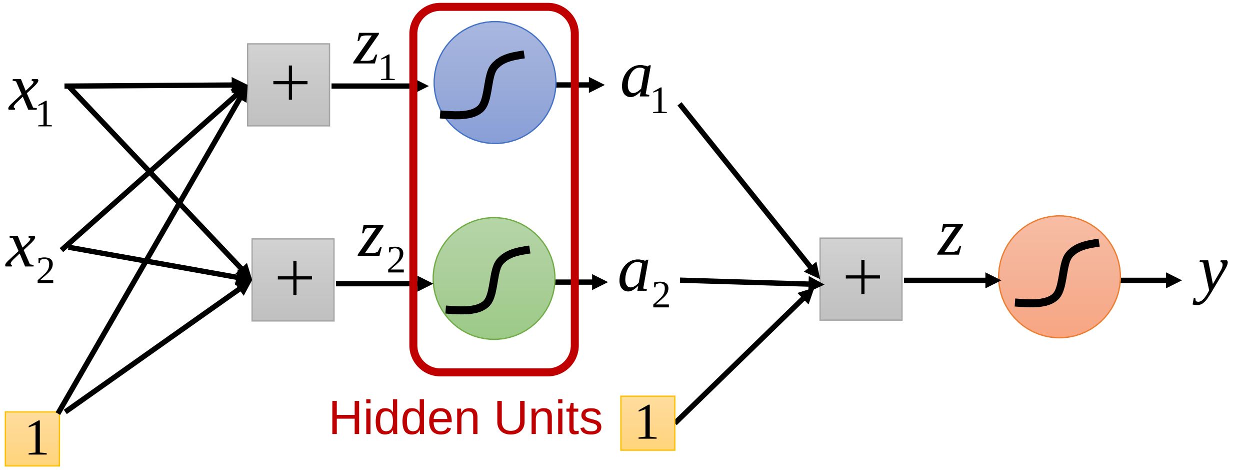
② Loss Function Design

- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

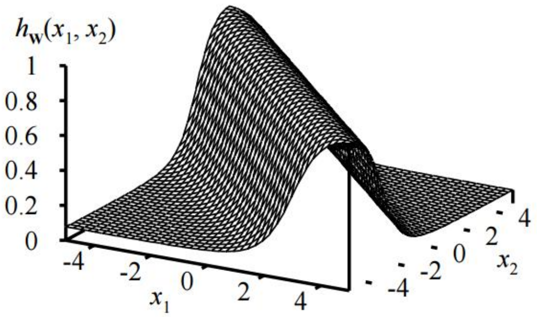
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Neural Networks – Multi-Layer Perceptron



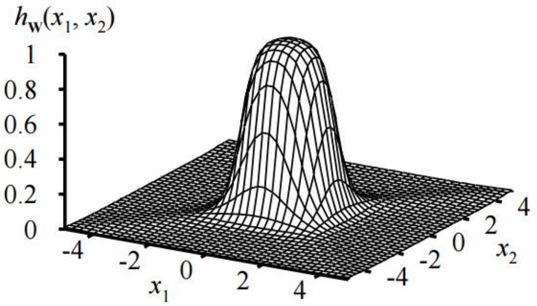
Expression of Multi-Layer Perceptron

- Continuous function w/ 2 layers



- Combine two opposite-facing threshold functions to make a ridge

- Continuous function w/ 3 layers



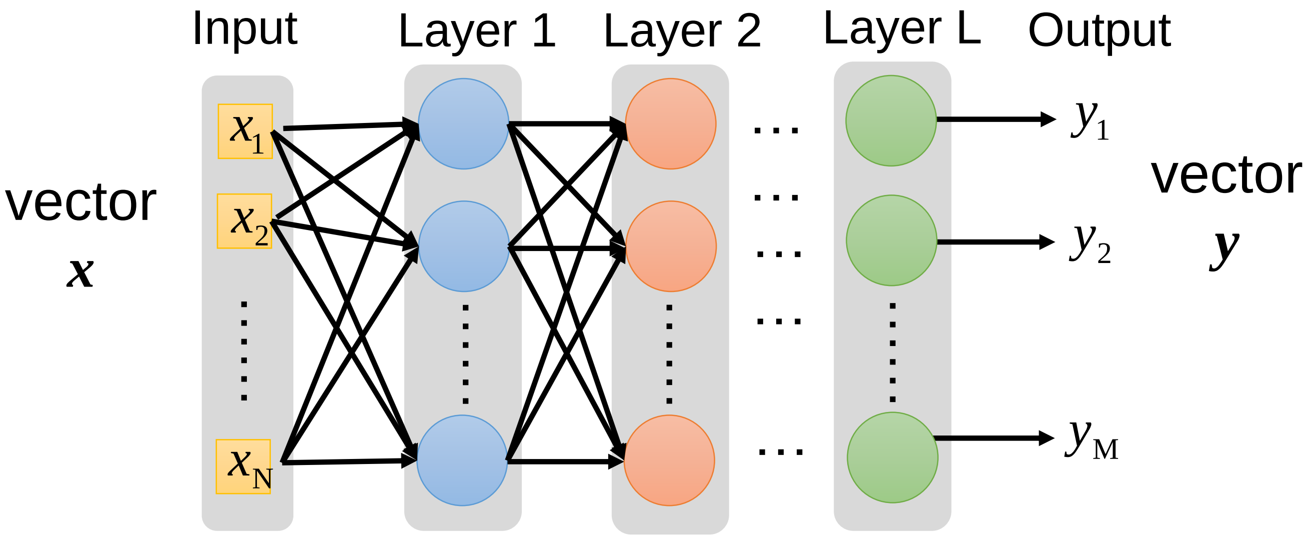
- Combine two perpendicular ridges to make a bump
  - Add bumps of various sizes and locations to fit any surface

multiple layers enhance the model expression
→ the model can approximate more complex functions

http://aima.eecs.berkeley.edu/slides-pdf/chapter20b.pdf

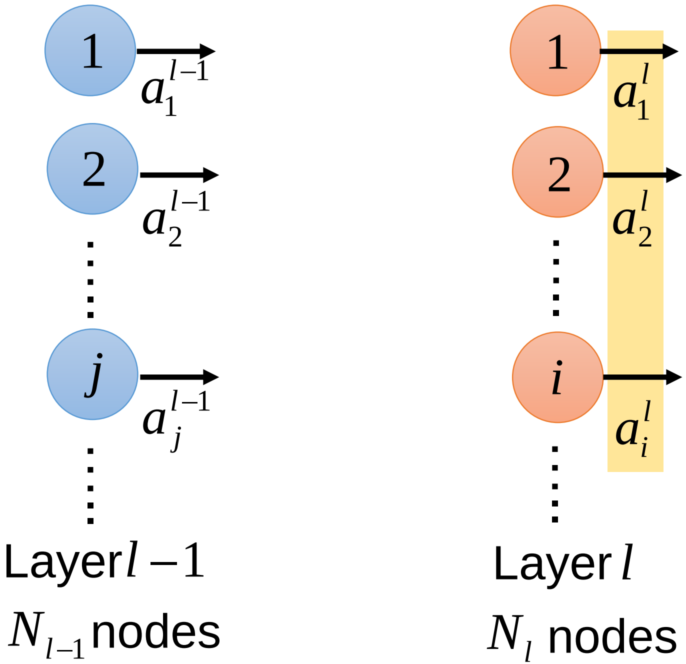
Deep Neural Networks (DNN)

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



Deep NN: multiple hidden layers

Notation Definition



Output of a neuron:

$$a_i^l$$

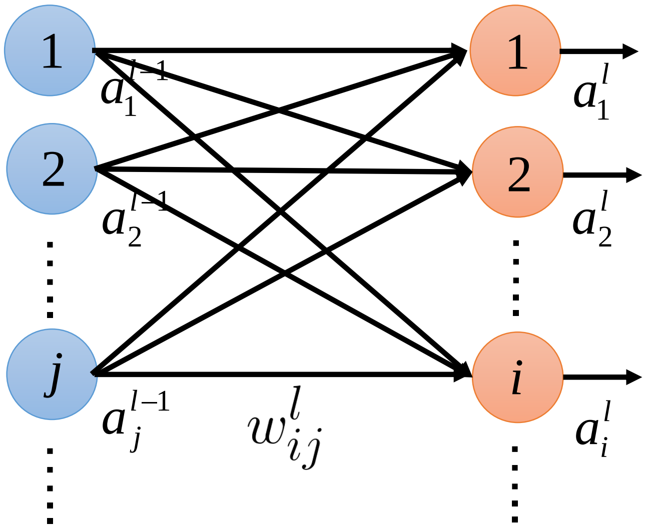
layer l

neuron i

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

output of one layer → a vector

Notation Definition



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

$$w_{ij}^l$$

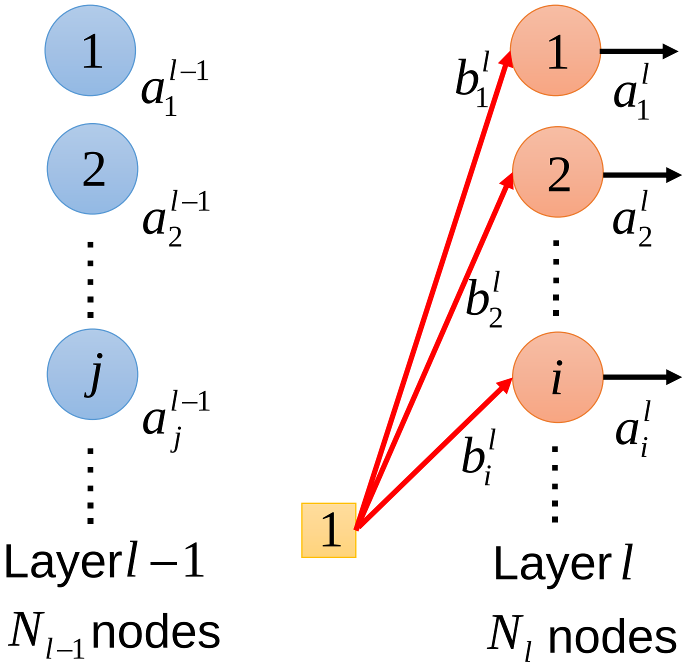
l: layer $l-1$ to layer l

ij: from neuron j (layer l-1) to neuron i (layer l)

$$W^l = \begin{array}{cc} \overleftrightarrow{\quad N_{l-1} \quad} & \\ \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} & N_l \end{array}$$

weights between two layers
→ a matrix

Notation Definition

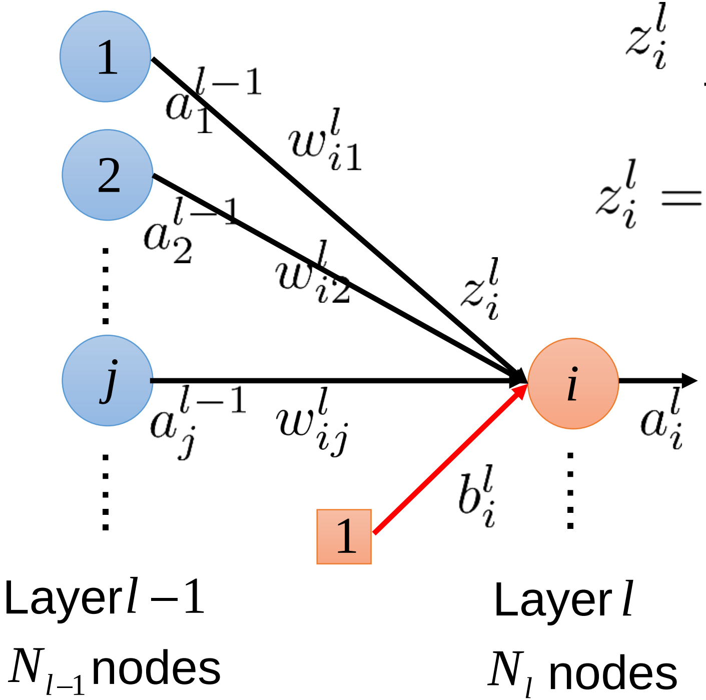


b_i^l : bias for neuron i at layer l

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

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

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

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

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

activation function input at each layer → a vector

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

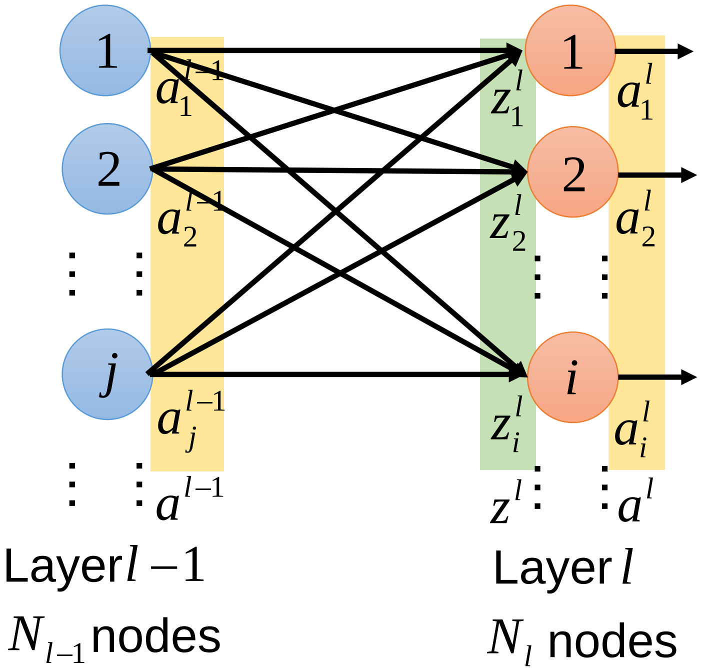
w_{ij}^l : a weight

W^l : a weight matrix

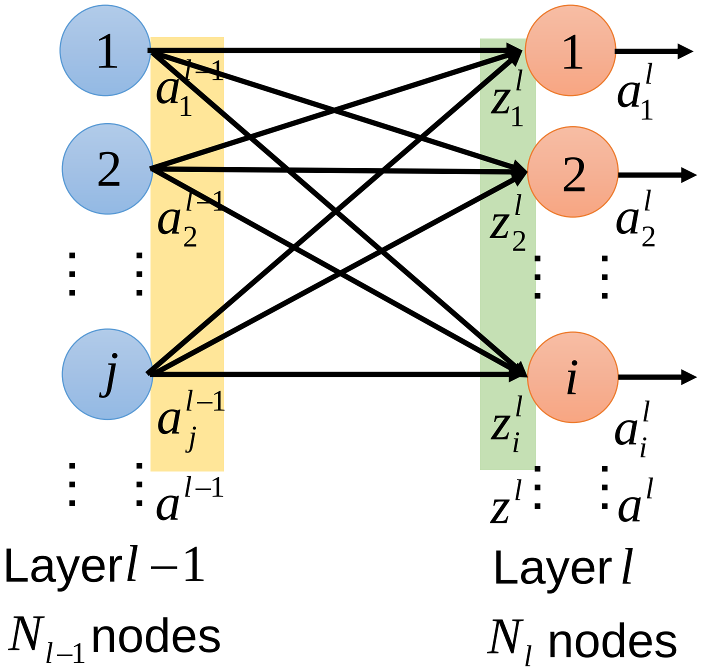
b_i^l : a bias

b^l : a bias vector

Layer Output Relation



Layer Output Relation – from *a* to *z*



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

$$\vdots$$

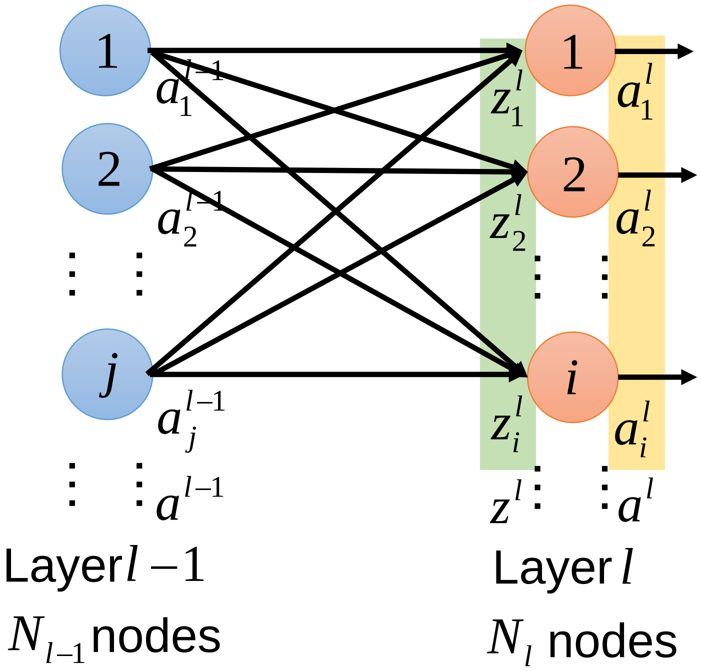
$$z_i^l = w_{i1}^1 a_1^{l-1} + w_{i2}^1 a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

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

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

Layer Output Relation – from z to a

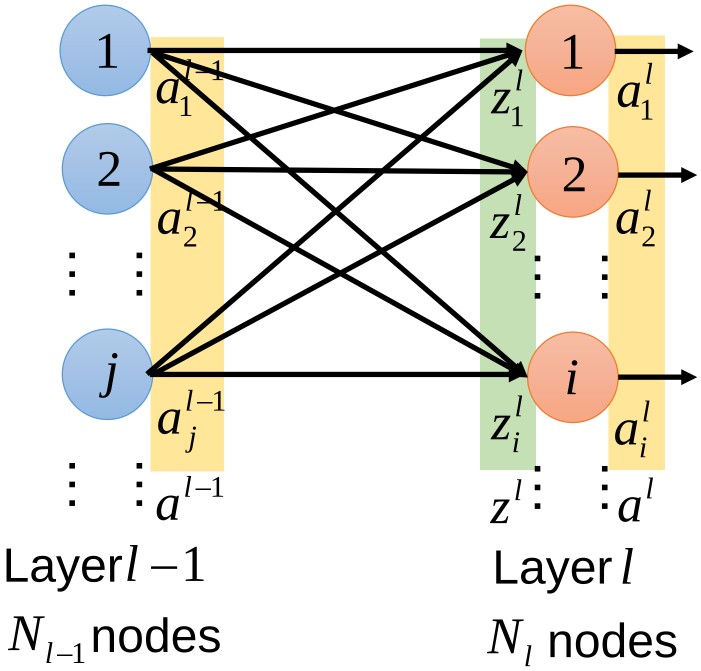


$$a_i^l = \sigma\left(z_i^l\right)$$

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

$$\boxed{a^l = \sigma\left(z^l\right)}$$

Layer Output Relation



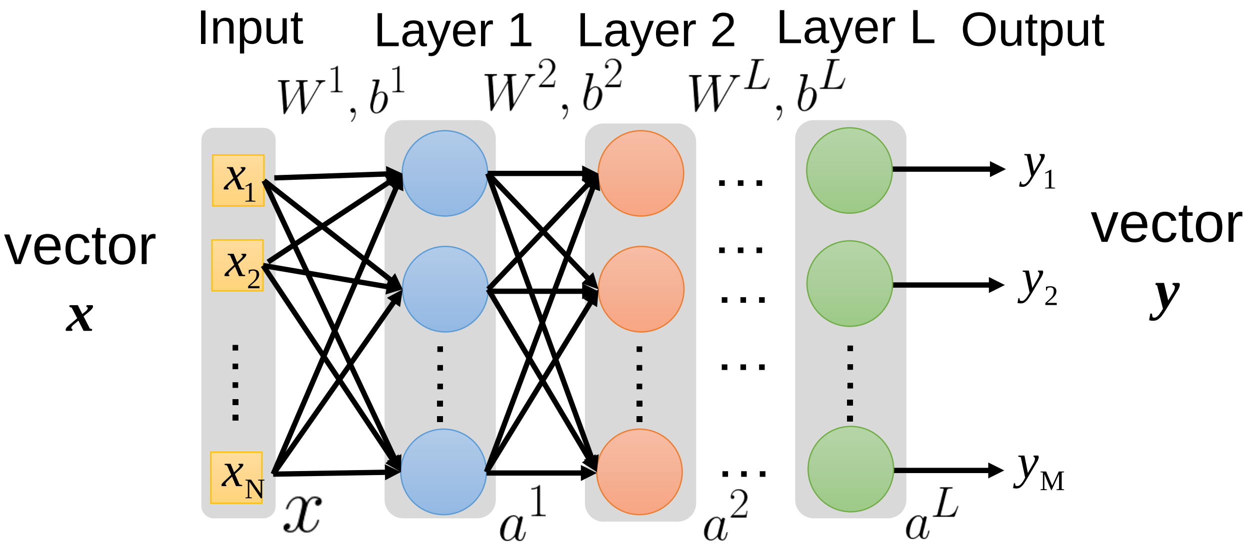
$$z^l = W^l a^{l-1} + b^l$$

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

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

Neural Network Formulation

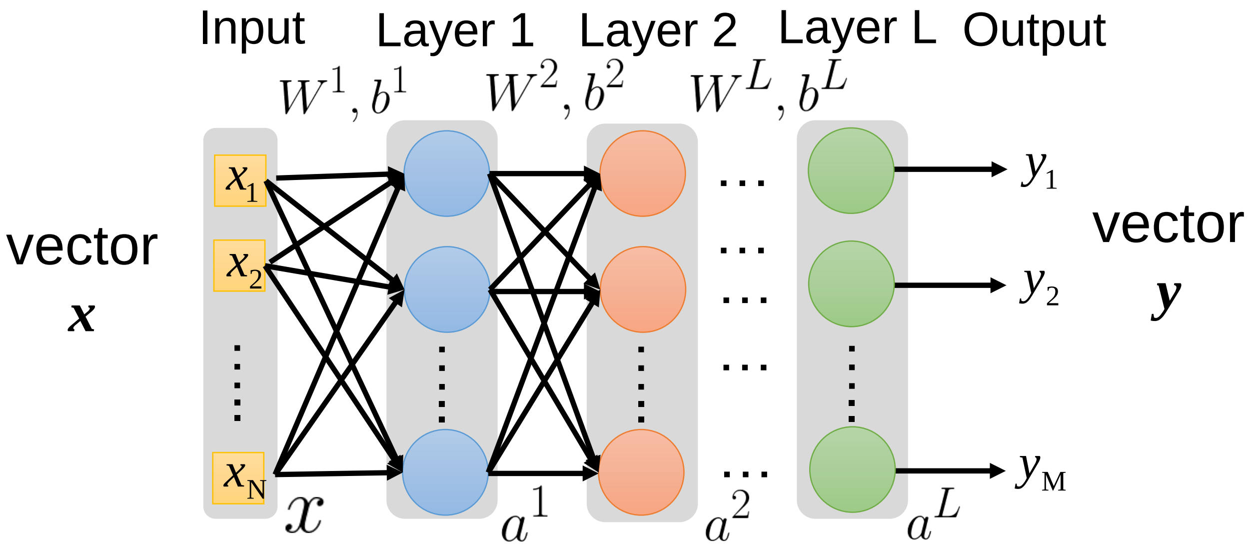
- Fully connected feedforward network $f : R^N \rightarrow R^M$



$$a^1 = \sigma(W^1 x + b^1) \quad a^2 = \sigma(W^2 a^1 + b^2) \quad a^L = \sigma(W^L a^{L-1} + b^L) = y$$

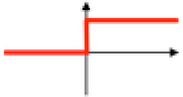
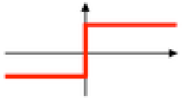
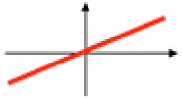
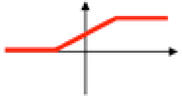
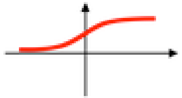
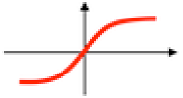
Neural Network Formulation

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



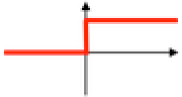
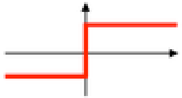
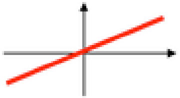
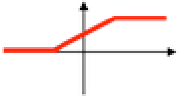
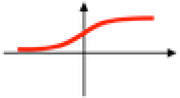
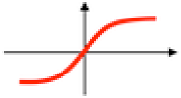
$$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

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)	$\phi(z) = \dfrac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	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

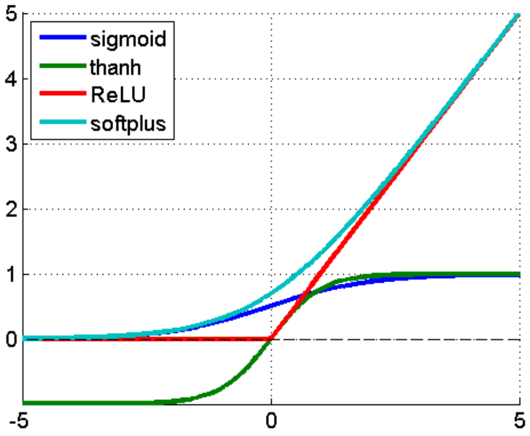
$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

- Tanh

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

- Rectified Linear Unit (ReLU)

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



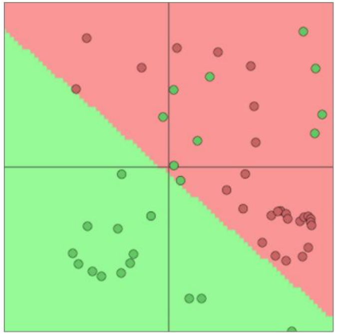
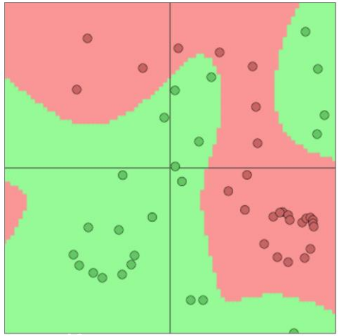
Non-linear functions are frequently used in neural networks

Why Non-Linearity?

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

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

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

45

What does the “Good” Function mean?

什麼叫做“好”的Function呢？

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Training Procedure Outline

① Model Architecture

- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design

- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Function = Model Parameters

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

function set

different parameters W and b → different functions

- Formal definition

$$f(x; \theta)$$

model parameter set

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

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

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

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

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

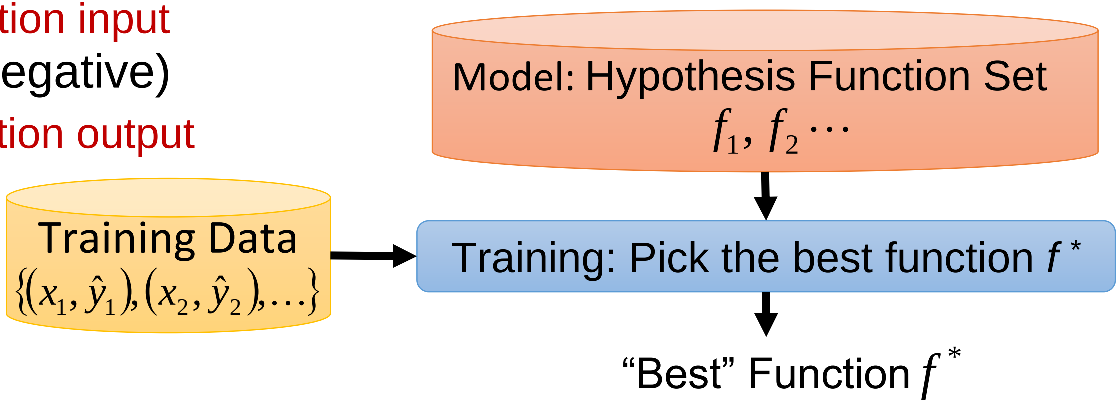
Loss Function Example

x: "It claims too much."

function input

$\hat{y}$: - (negative)

function output



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

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 Function

- Square loss

$$C(\theta) = (1 - \hat{y} f(x;\theta))^2$$

- Hinge loss

$$C(\theta) = \max(0, 1 - \hat{y} f(x;\theta))$$

- Logistic loss

$$C(\theta) = -\hat{y} \log(f(x;\theta))$$

- Cross entropy loss

$$C(\theta) = -\sum \hat{y} \log(f(x;\theta))$$

- Others: large margin, etc.

How can we Pick the "Best" Function?

我們如何找出“最好”的Function呢？

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Problem Statement

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

How to solve this optimization problem?

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

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

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

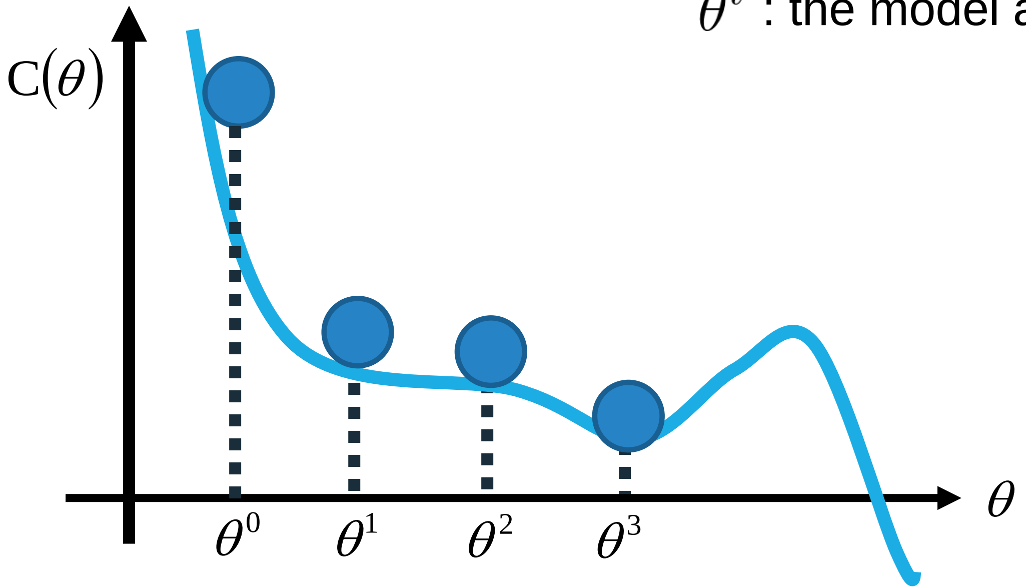
② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Gradient Descent for Optimization

- Assume that θ has only one variable

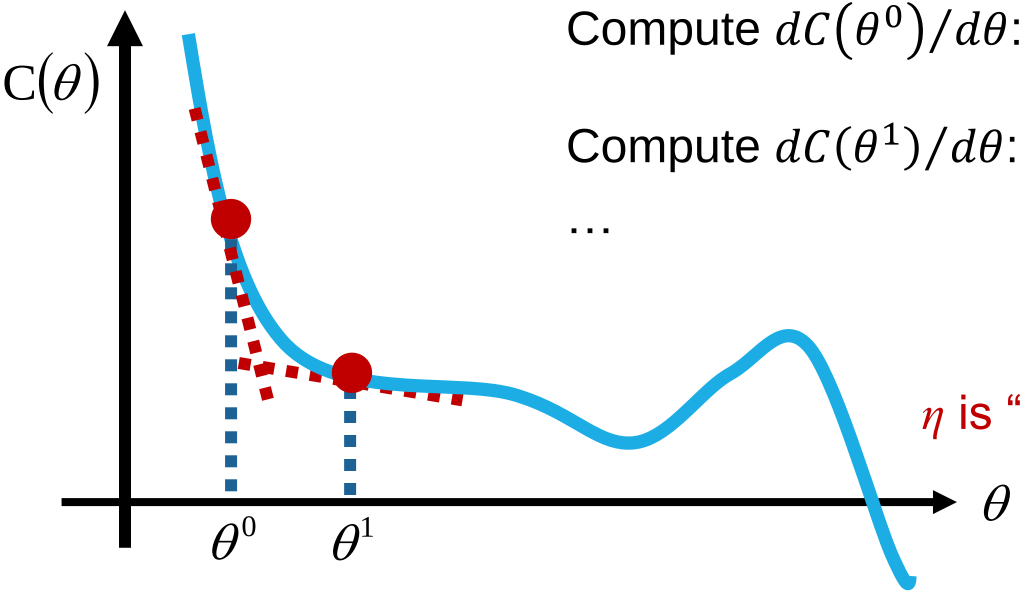
θ^i : the model at the i-th iteration



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

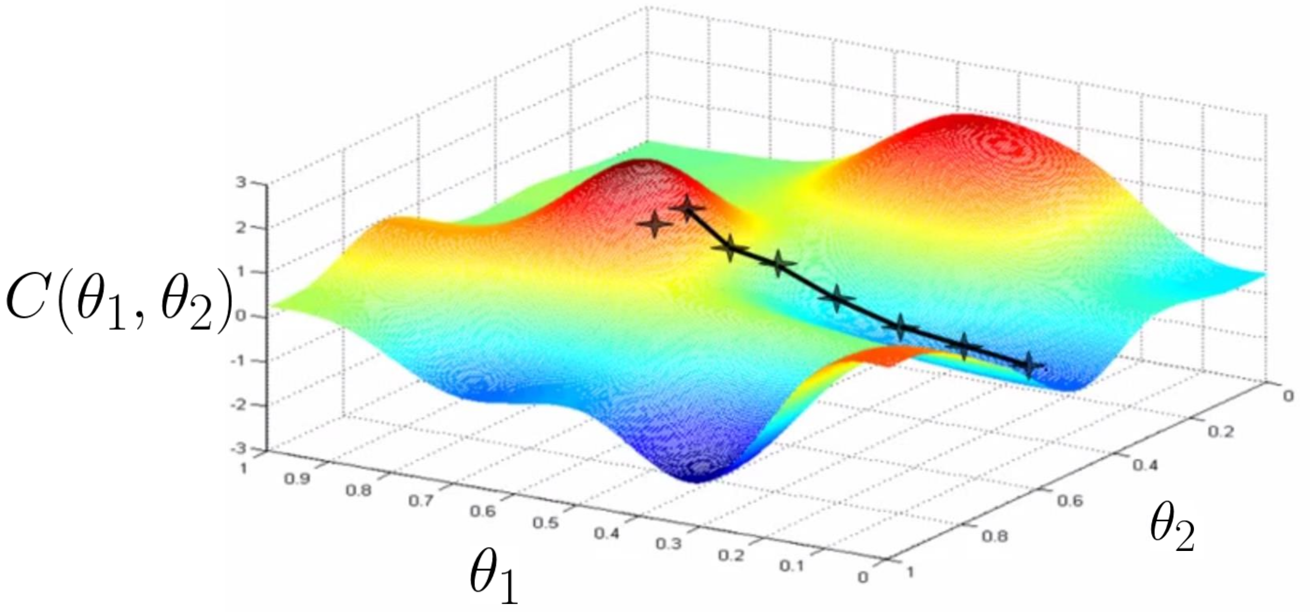
…

η is "learning rate"

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

Gradient Descent for Optimization

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



Gradient Descent for Optimization

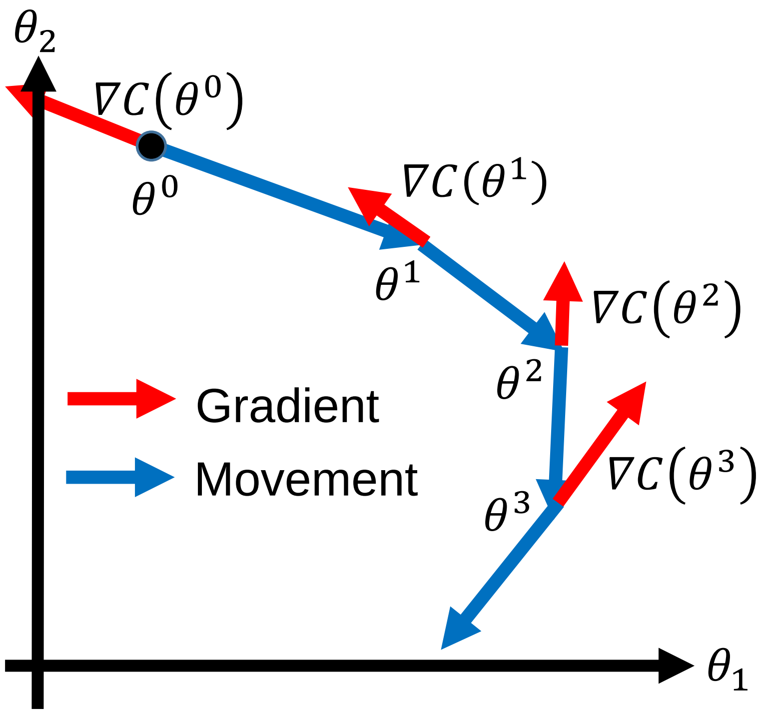
- Assume that θ has two variables $\{\theta_1, \theta_2\}$
  - Randomly start at θ^0: $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$
  - Compute the gradients of $C(\theta)$ at θ^0: $\nabla_\theta C(\theta^0) = \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$
  - Update parameters:

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

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

  - 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

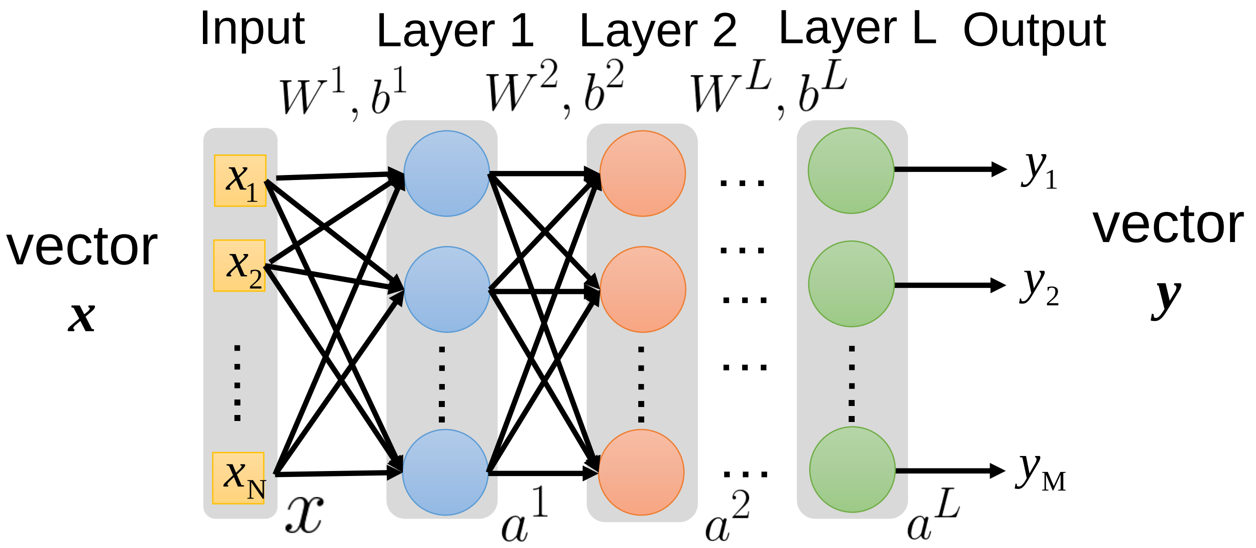
update parameters

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

}

Revisit Neural Network Formulation

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



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

Gradient Descent for Neural Network

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

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

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

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

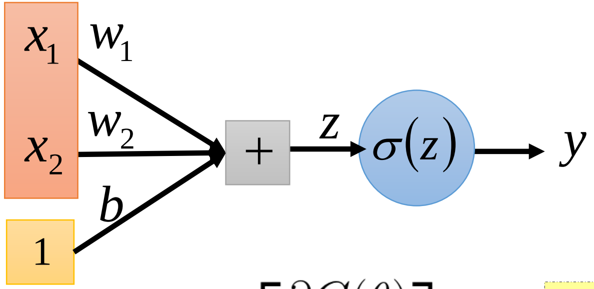
Algorithm
Initialization: start at θ^0
while($\theta^{(i+1)} \neq \theta^i$)
{
    compute gradient at θ^i
    update parameters
    $\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$
}

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

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

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

Gradient Descent for Optimization

Simple Case – Three Parameters & 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

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

Gradient Descent for Optimization

Simple Case – Square Error Loss

- Square Error Loss

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

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

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

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

Gradient Descent for Optimization

Simple Case – Square Error Loss

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

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

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

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

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

Gradient Descent for Optimization

Simple Case – Square Error Loss

- Square Error Loss

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

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

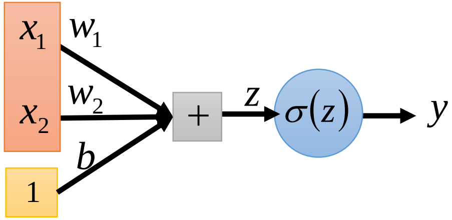
$$\frac{\partial \sigma(Wx+b)}{\partial w_1} = [1 - \sigma(Wx+b)]\sigma(Wx+b)x_1$$

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

Gradient Descent for Optimization

Simple Case – Three Parameters & Square Error Loss

- Update three parameters for t-th iteration



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

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

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

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

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

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

Optimization Algorithm

Algorithm
Initialization: set the parameters θ, b at random
while(stopping criteria not met)
{
    for training sample $\{x, \hat{y}\}$, compute gradient and update parameters

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

}

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

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

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

Gradient Descent for Neural Network

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

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

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

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

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**.

Gradient Descent Issue

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

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

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

Training Data
$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

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

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Stochastic Gradient Descent (SGD)

- Gradient Descent

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

- 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

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

Training Data

$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

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

Epoch Definition

- When running SGD, the model starts θ^0

Training Data
$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

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

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

$\vdots$

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

$\vdots$

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

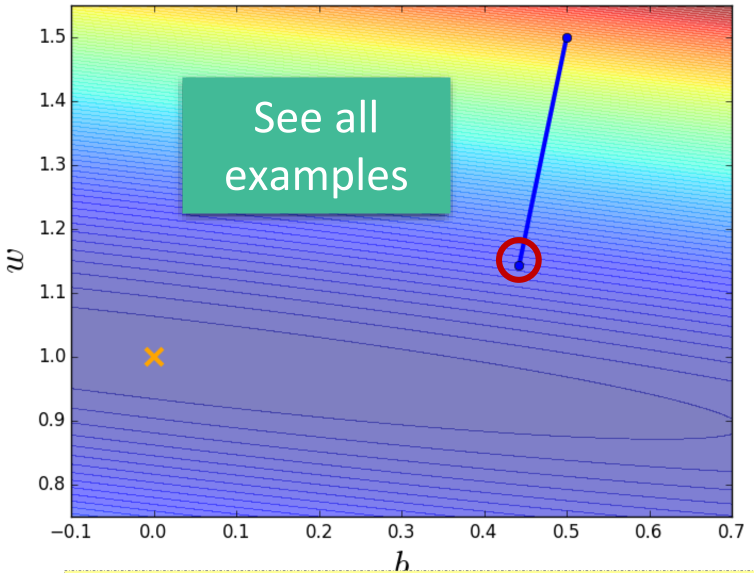
see all training samples once

→ one epoch

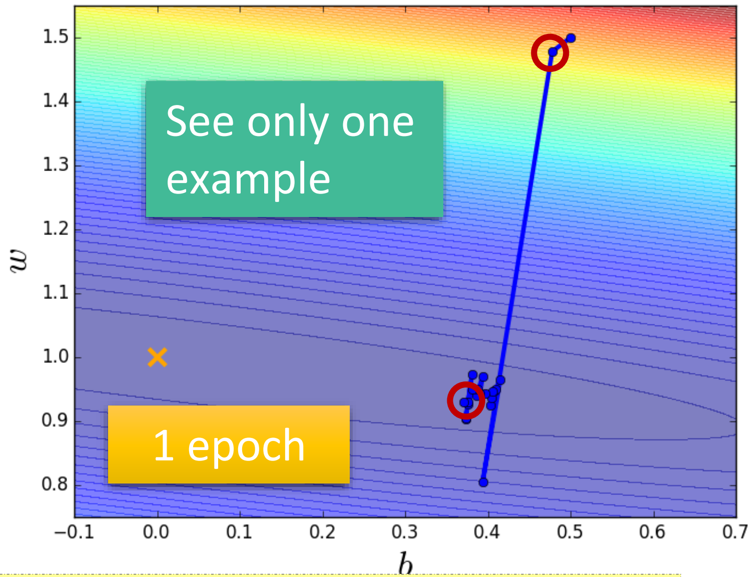
pick x_1 $\quad \theta^{K+1} = \theta^K - \eta \nabla C_1(\theta^K)$

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

Training Procedure Outline

① Model Architecture
- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization
- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Mini-Batch SGD

- Batch Gradient Descent

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

Use all K samples in each iteration

- Stochastic Gradient Descent (SGD)
  - Pick a training sample x_k

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

Use *1* samples in each iteration

- Mini-Batch SGD
  - Pick a set of B training samples as a batch b

B is "batch size"

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

Use all B samples in each iteration

Mini-Batch SGD

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

Require: Learning rate ϵ_k.

Require: Initial parameter $\boldsymbol{\theta}$

while stopping criterion not met **do**

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

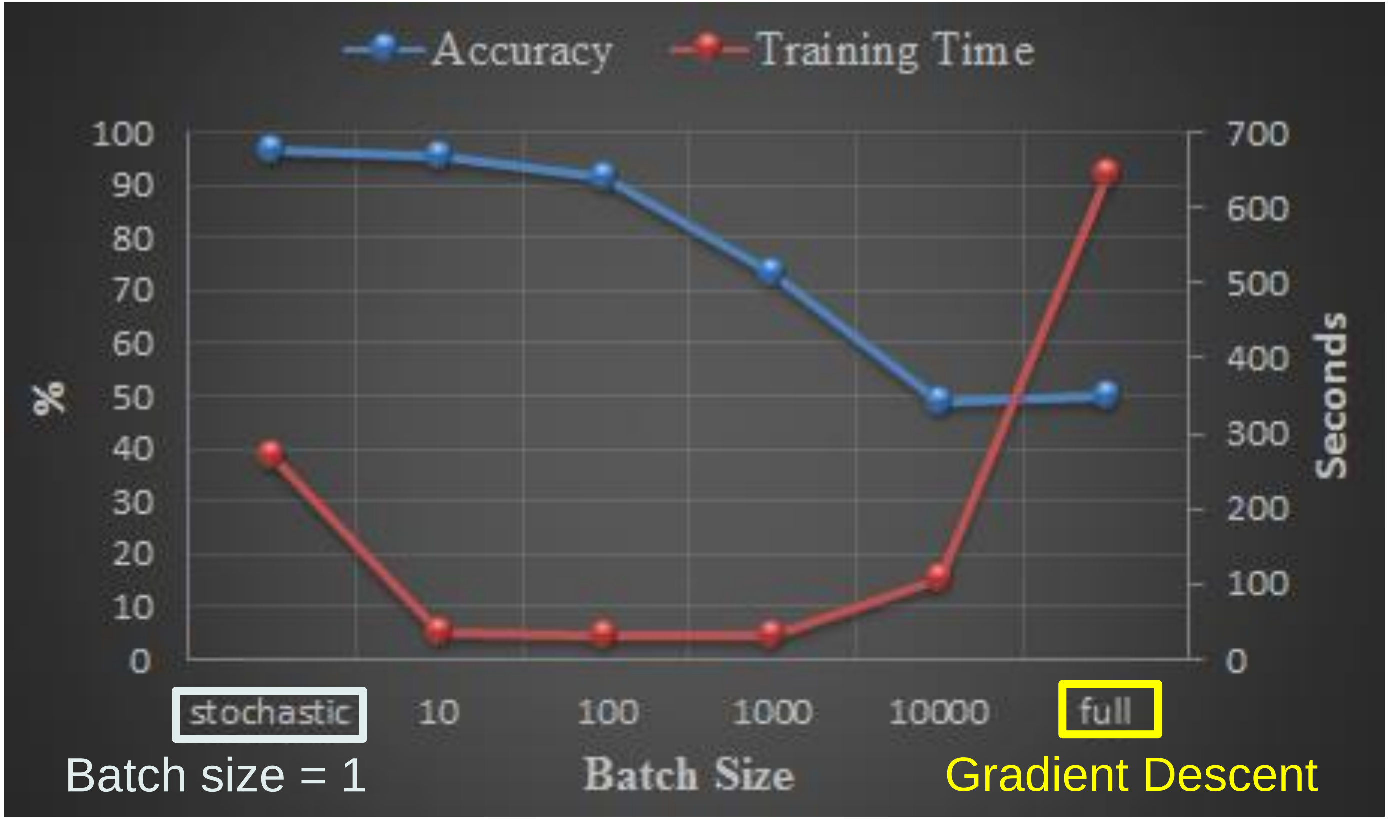
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

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

end while

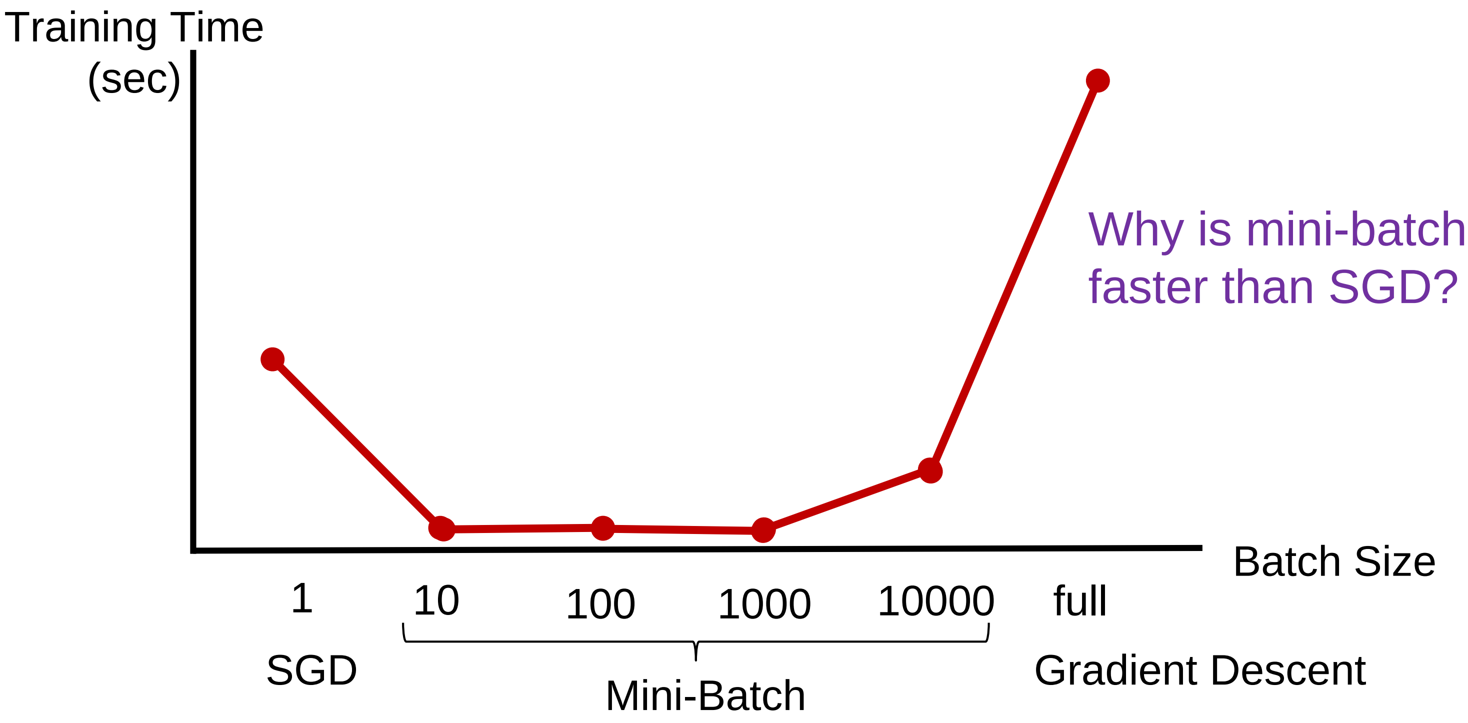
Batch v.s. Mini-Batch

Handwritting Digit Classification



Gradient Descent v.s. SGD v.s. Mini-Batch

Training speed: mini-batch > SGD > Gradient Descent



Why is mini-batch faster than SGD?

SGD v.s. Mini-Batch

- Stochastic Gradient Descent (SGD)

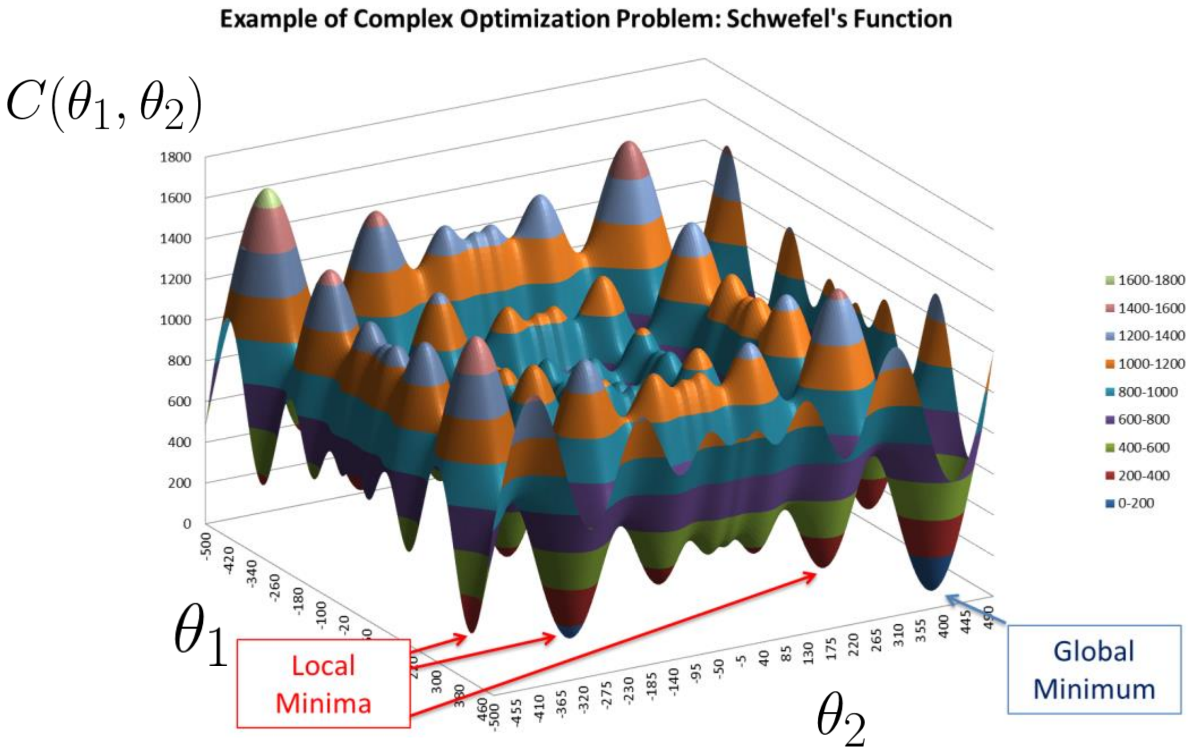
$$z^1 = W^1 \, x \qquad z^1 = W^1 \, x \qquad \ldots\ldots$$

- Mini-Batch SGD

$$\boxed{z^1 \;\; z^1} = W^1 \, \overbrace{\boxed{x \;\; x}}^{\text{matrix}}$$

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

Big Issue: Local Optima



Neural networks has <u>no guarantee</u> for obtaining global optimal solution

Training Procedure Outline

① Model Architecture

- ✓ A Single Layer of Neurons (Perceptron)
- ✓ Limitation of Perceptron
- ✓ Neural Network Model (Multi-Layer Perceptron)

② Loss Function Design

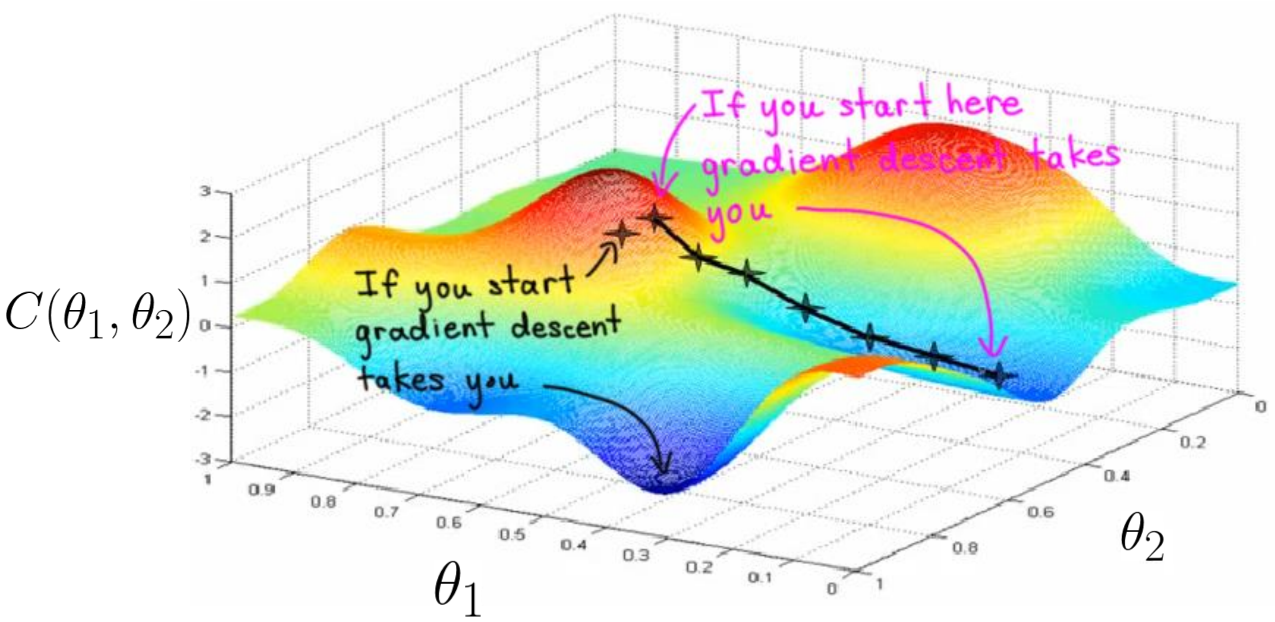
- ✓ Function = Model Parameters
- ✓ Model Parameter Measurement

③ Optimization

- ✓ Gradient Descent
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch SGD
- ✓ Practical Tips

Initialization

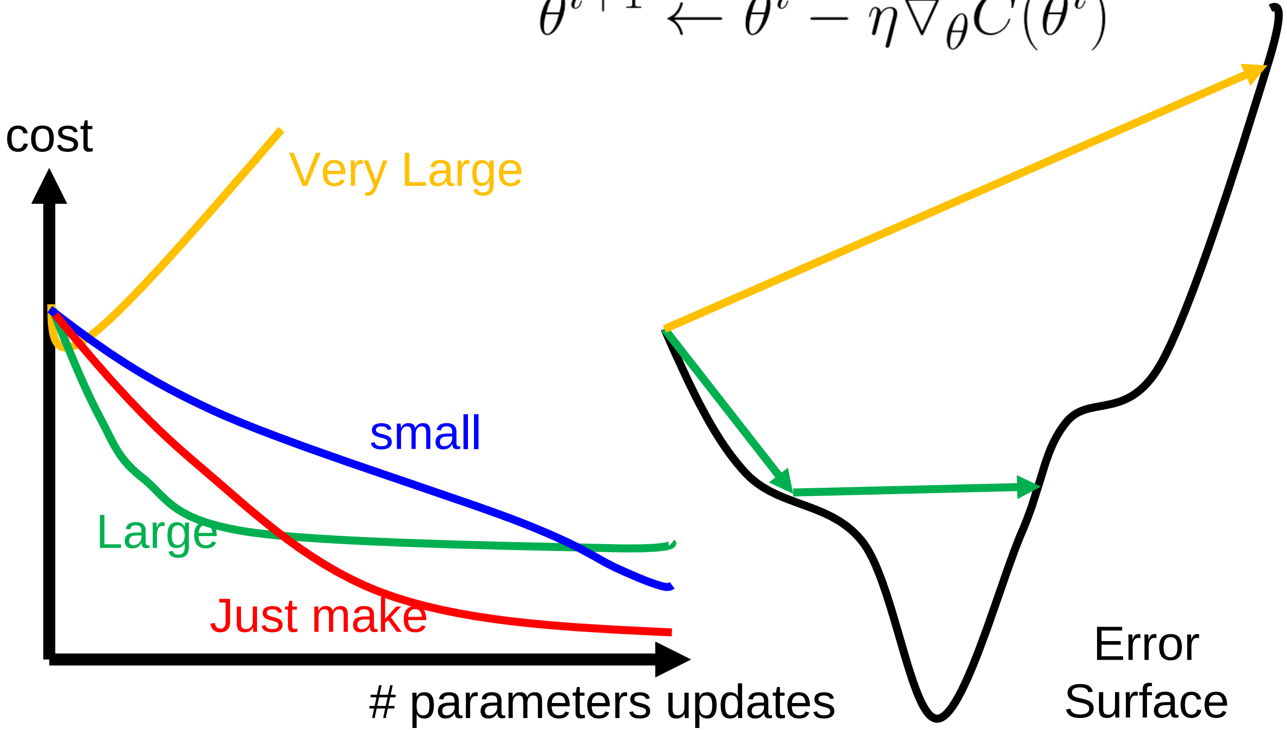
- Different initialization parameters may result in different trained models



Do not initialize the parameters equally → set them randomly

Learning Rate

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

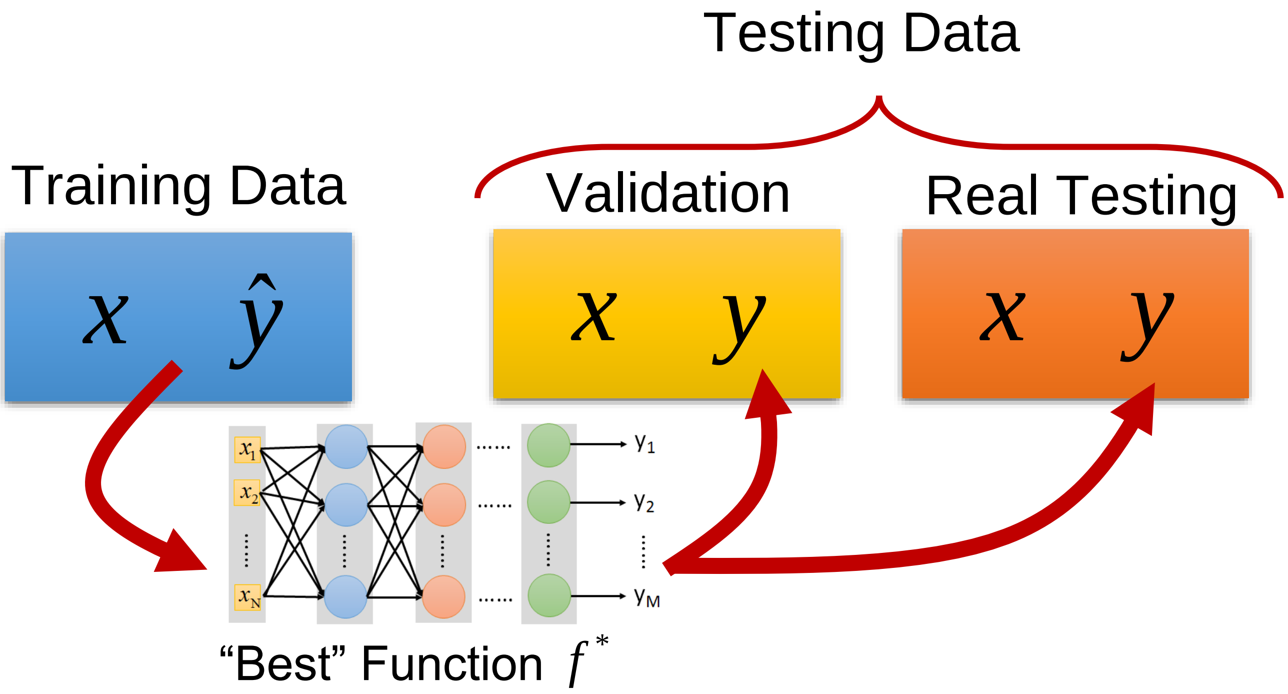


Learning rate should be set carefully

Tips for Mini-Batch Training

- Shuffle training samples before every epoch
  - the network might memorize the order you feed the samples
- Use a fixed batch size for every epoch
  - enable to fast implement matrix multiplication for calculations
- Adapt the learning rate to the batch size
  - larger batch → smaller learning rate

Learning Recipe



Learning Recipe

Testing Data

Training Data

$x \quad \hat{y}$

Validation

$x \quad y$

immediately know the performance

Real Testing

$x \quad y$

Do not know the performance until submission

Learning Recipe

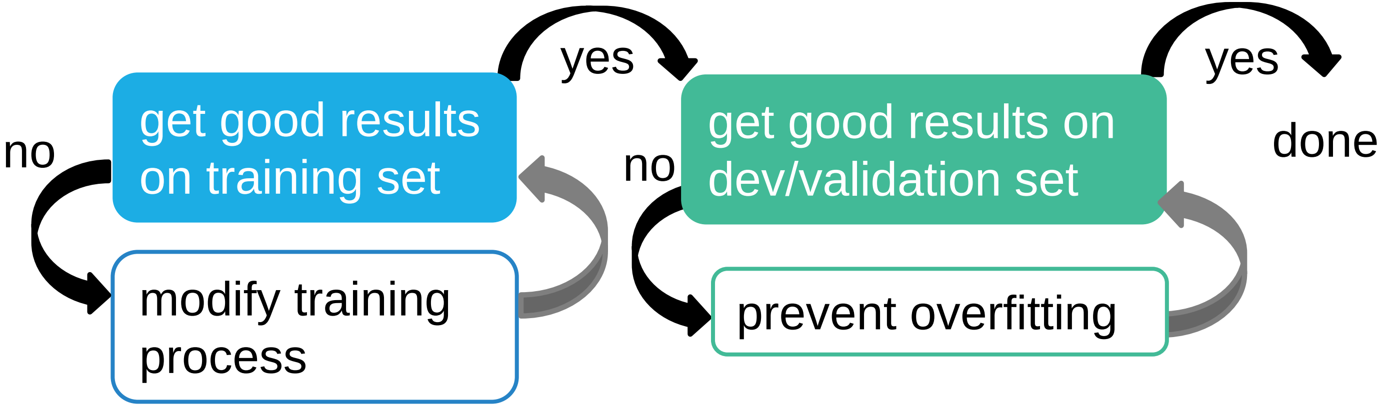
no

get good results on training set

modify training process

- Possible reasons
  - no good function exists: bad hypothesis function set
    → reconstruct the model architecture
  - cannot find a good function: local optima
    → change the training strategy

Learning Recipe



Better performance on training but worse performance on dev → overfitting

Overfitting

- Possible solutions
  - more training samples
  - some tips: dropout, etc.

Concluding Remarks

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

Model Architecture

Loss Function Design

Optimization