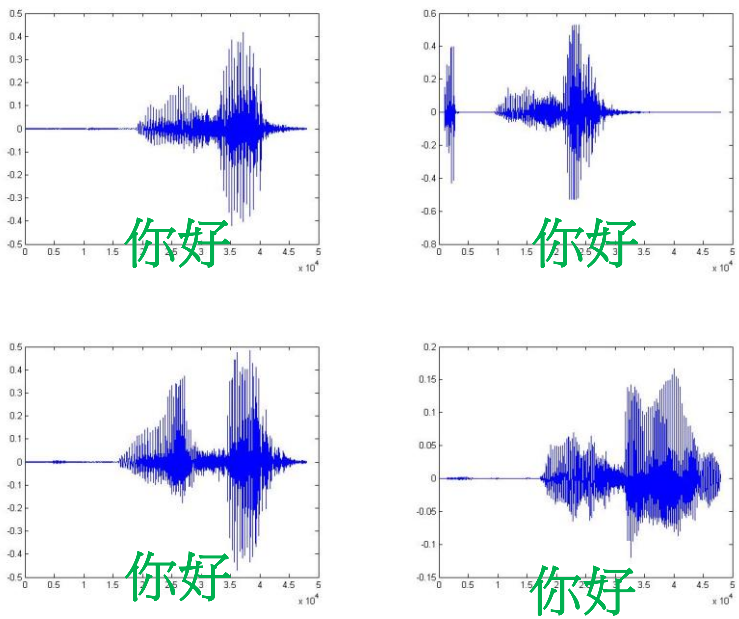
What is
Machine Learning?

You know how to program …

- You can ask computers to do lots of things for you.
- However, computer can only do what you ask it to do.
- Computer can never solve the problem you can’t solve.

Some tasks are very complex

- One day, you are asked to write a program for speech recognition.

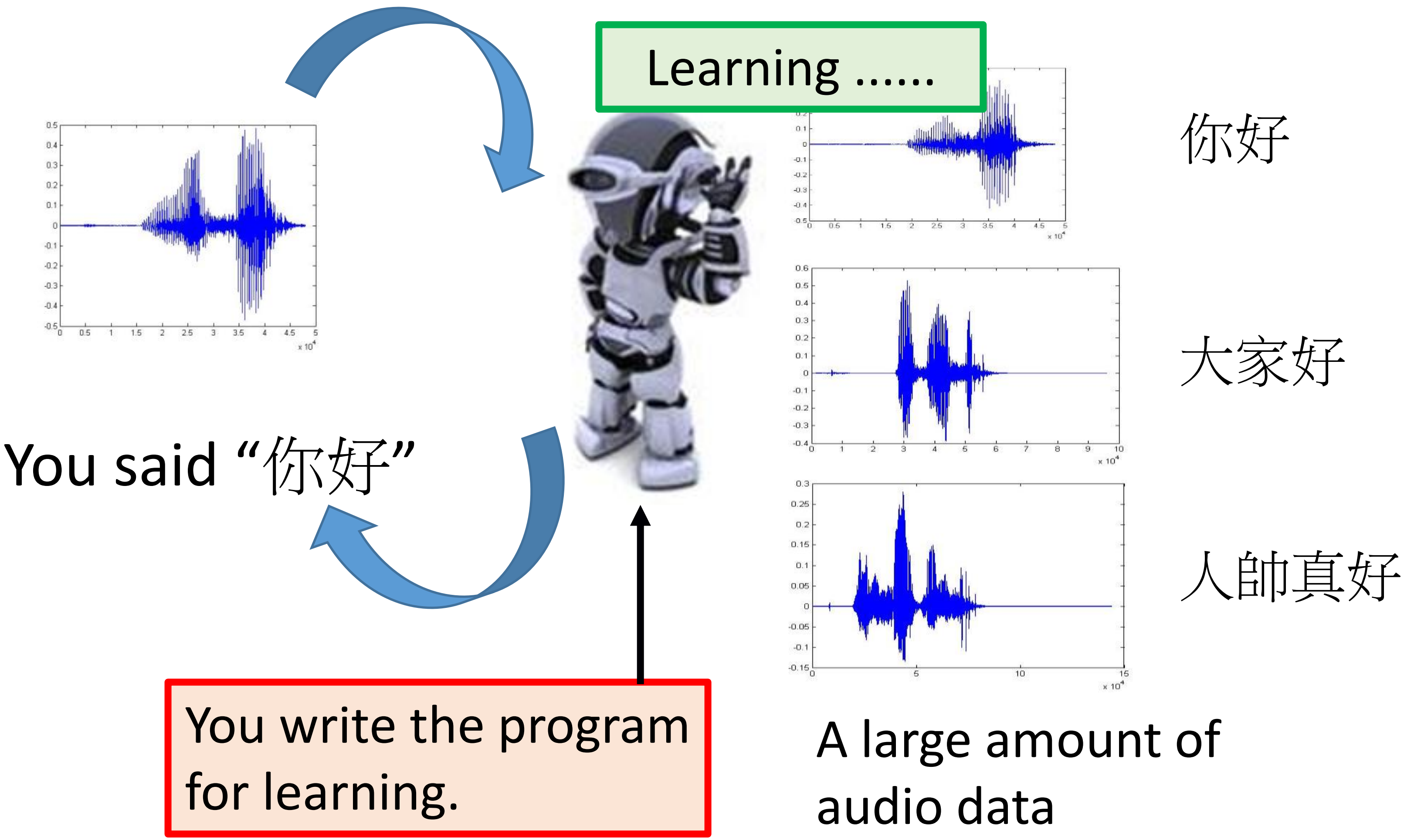


Find the common patterns from the left waveforms.

You quickly get lost in the exceptions and special cases.

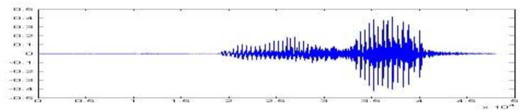
It seems impossible to write a program for speech recognition.

Let the machine learn by itself



Learning ≈ Looking for a Function

- Speech Recognition

$f($  $)=$ “你好”

- Handwritten Recognition

$f($  $)=$ “2”

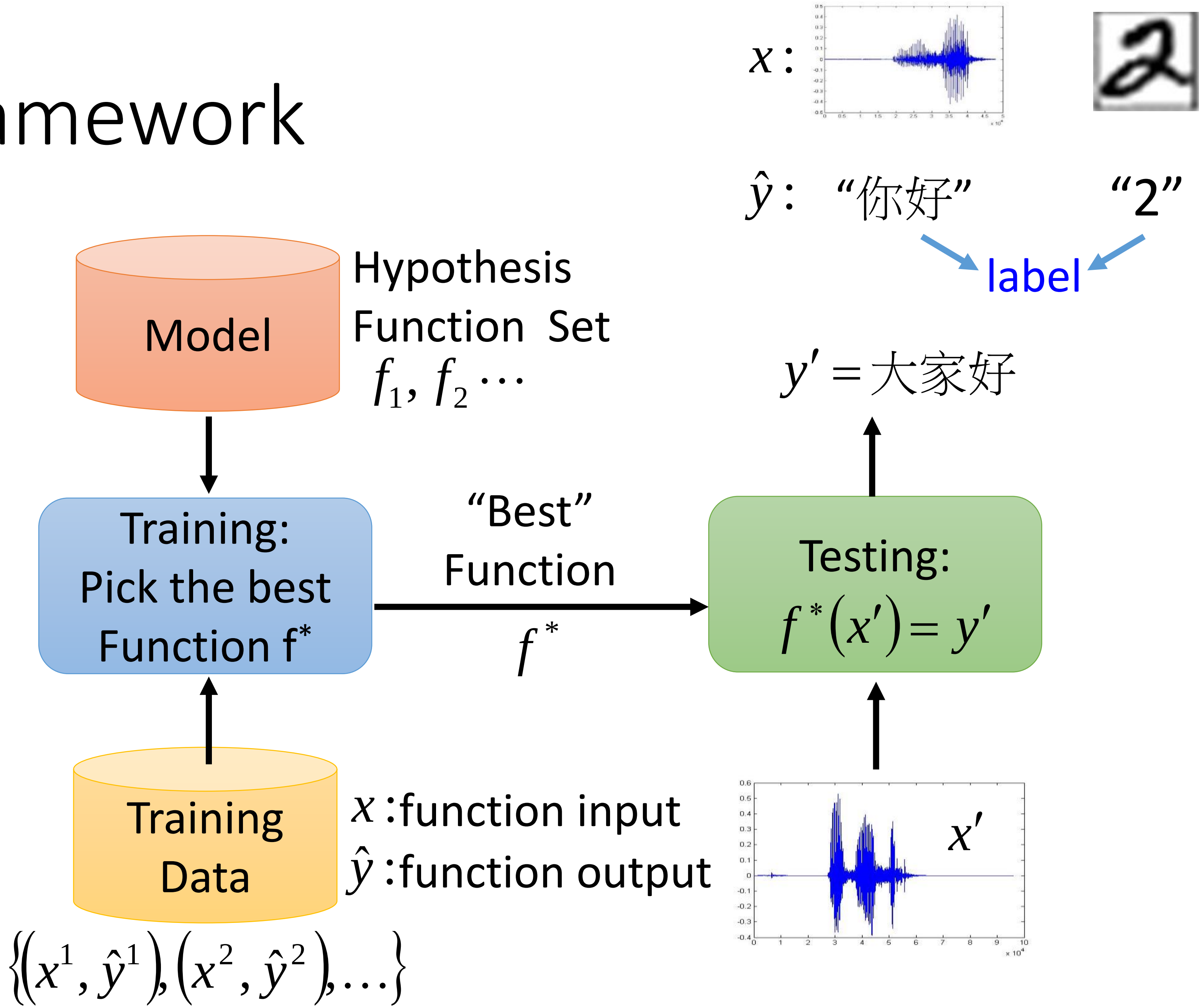
- Weather forecast

$f($ weather today $)=$ “sunny tomorrow”

- Play video games

$f($ Positions and number of enemies $)=$ “fire”

Framework

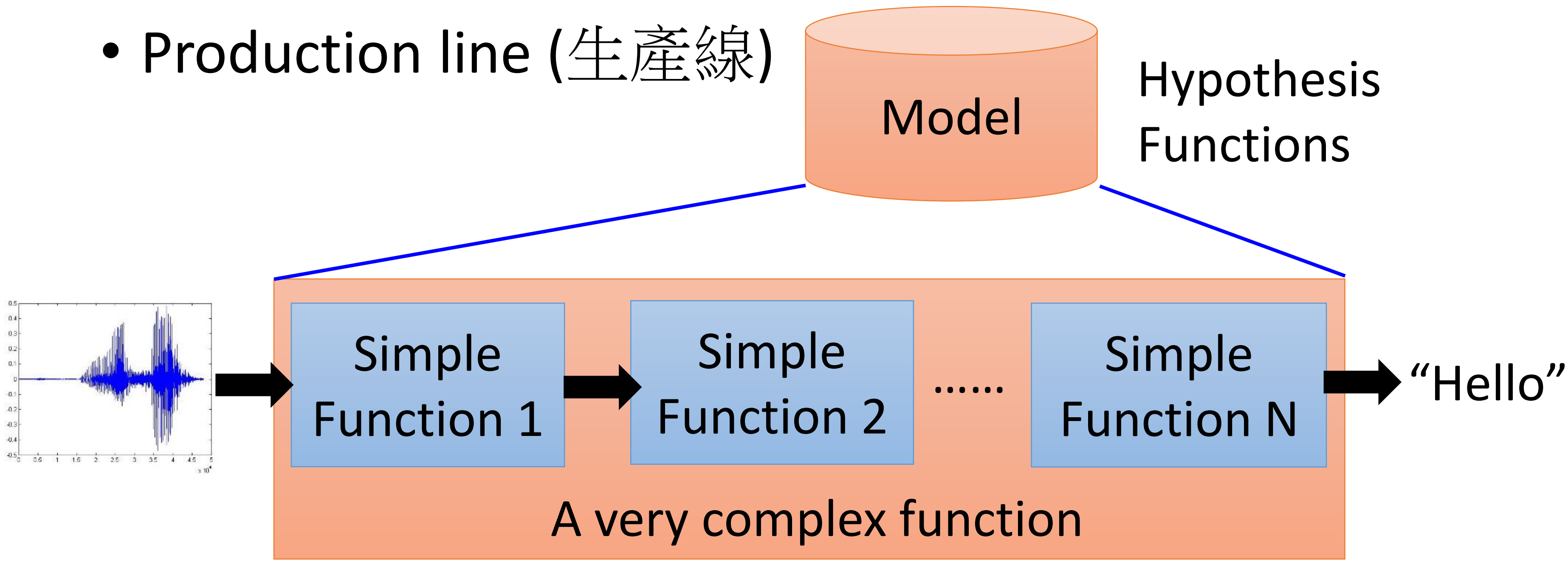


x :
$\hat{y}$: "你好" "2"
label
Model
Hypothesis Function Set
$f_1, f_2 \cdots$
Training: Pick the best Function f*
"Best" Function
f^*
Testing:
$f^*(x') = y'$
$y' =$ 大家好
Training Data
x :function input
$\hat{y}$:function output
$\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \ldots\}$
x'

Deep Learning

What is Deep Learning?

- Production line (生產線)



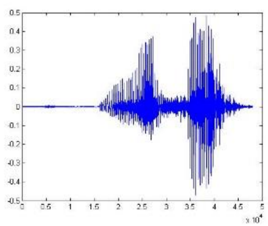
End-to-end training:

What each function should do is learned automatically

Deep v.s. Shallow
- Speech Recognition

- Deep Learning

"Bye bye, MFCC"
- Deng Li in Interspeech 2014



f_1 → f_2 → f_3 → f_4 → f_5 → "Hello"

All functions are learned from data

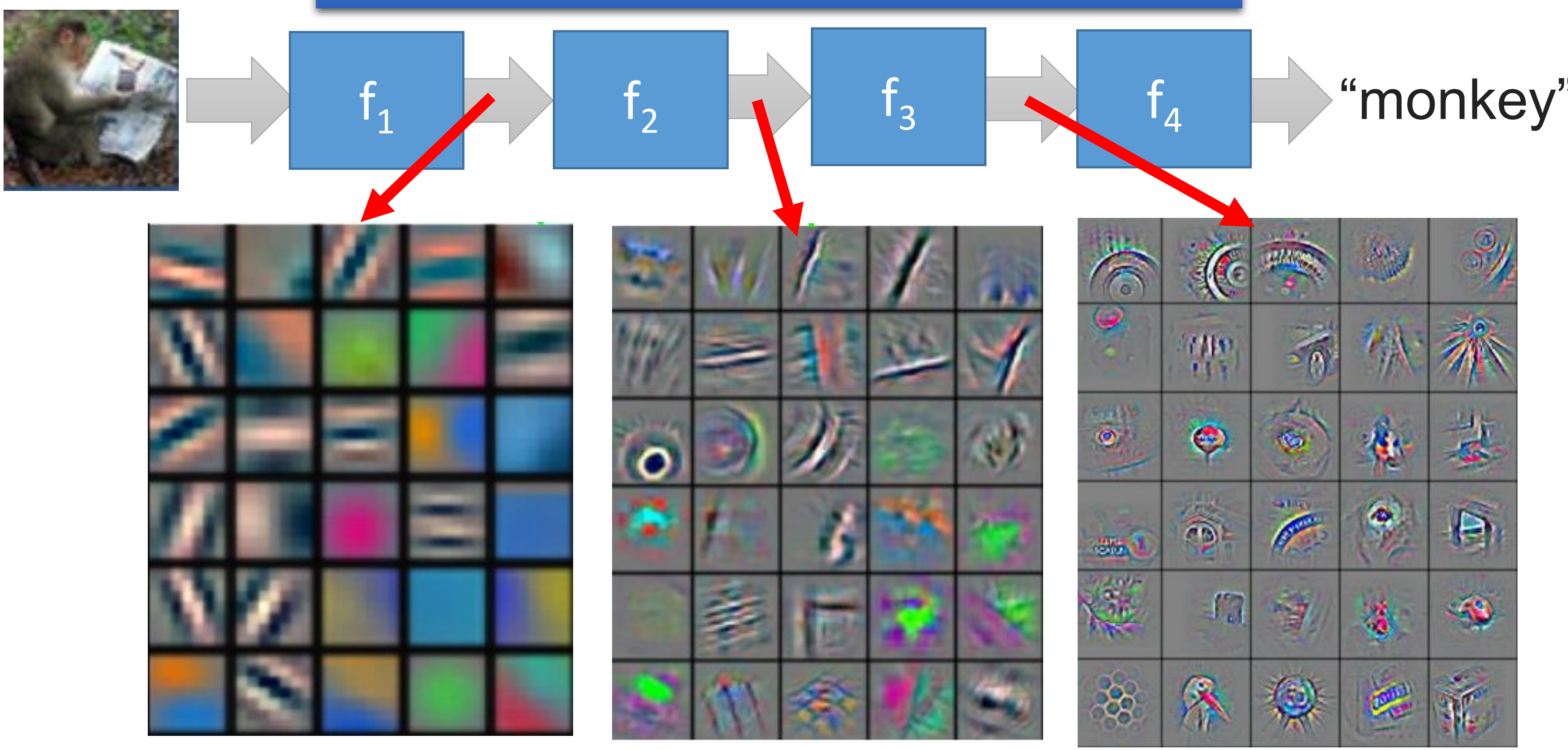
Less engineering labor, but machine learns more

Deep v.s. Shallow - Image Recognition

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

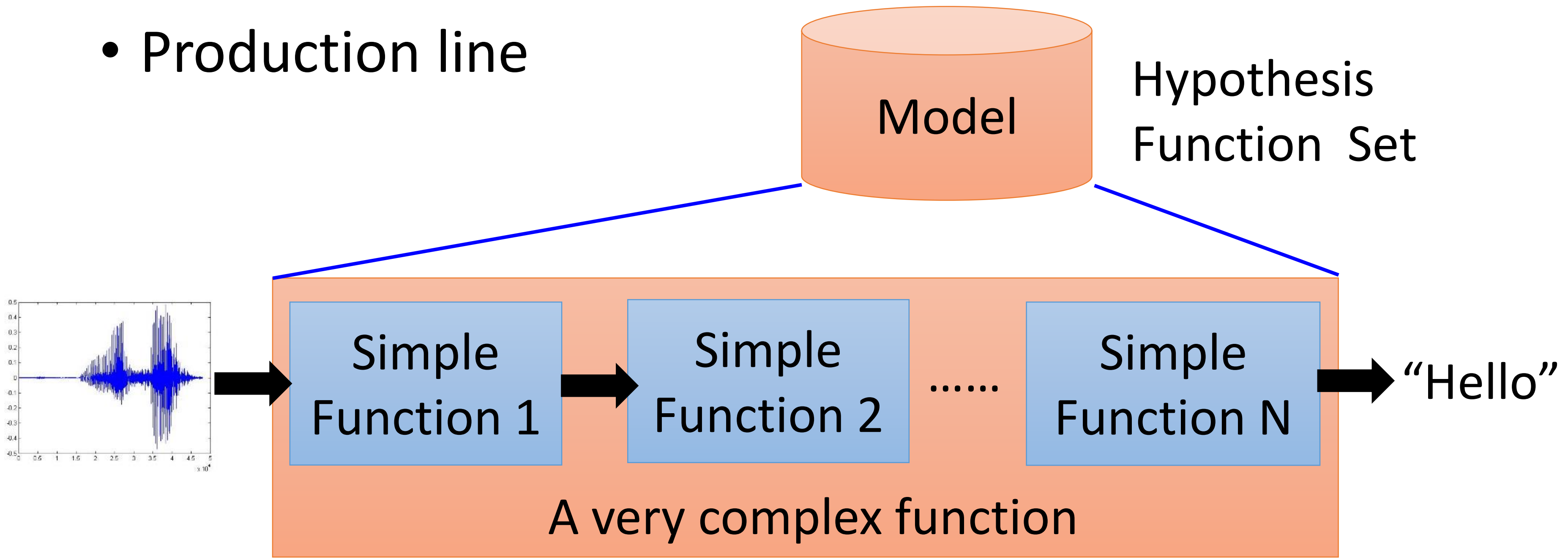
- Deep Learning

All functions are learned from data



What is Deep Learning?

- Production line



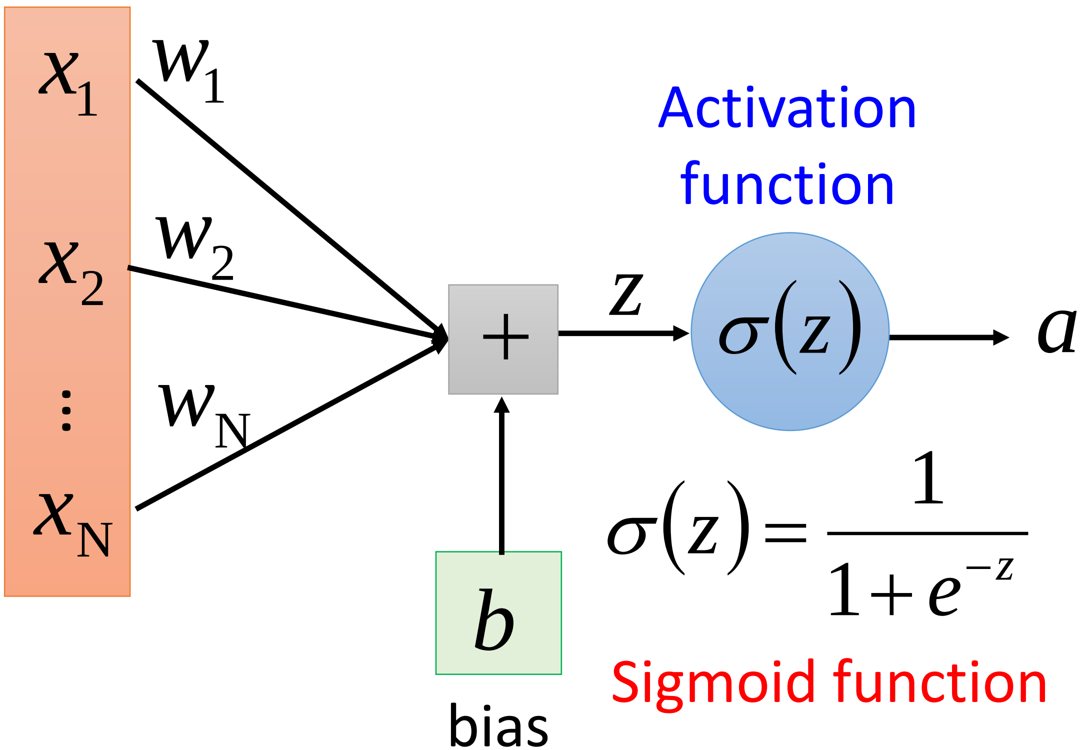
End-to-end training:

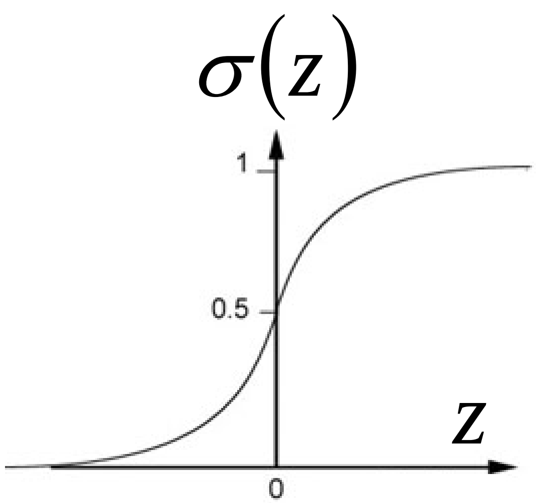
What each function should do is learned automatically

- Deep learning usually referred to neural network based approach

A Neuron for Machine

Each neuron is a very simple function





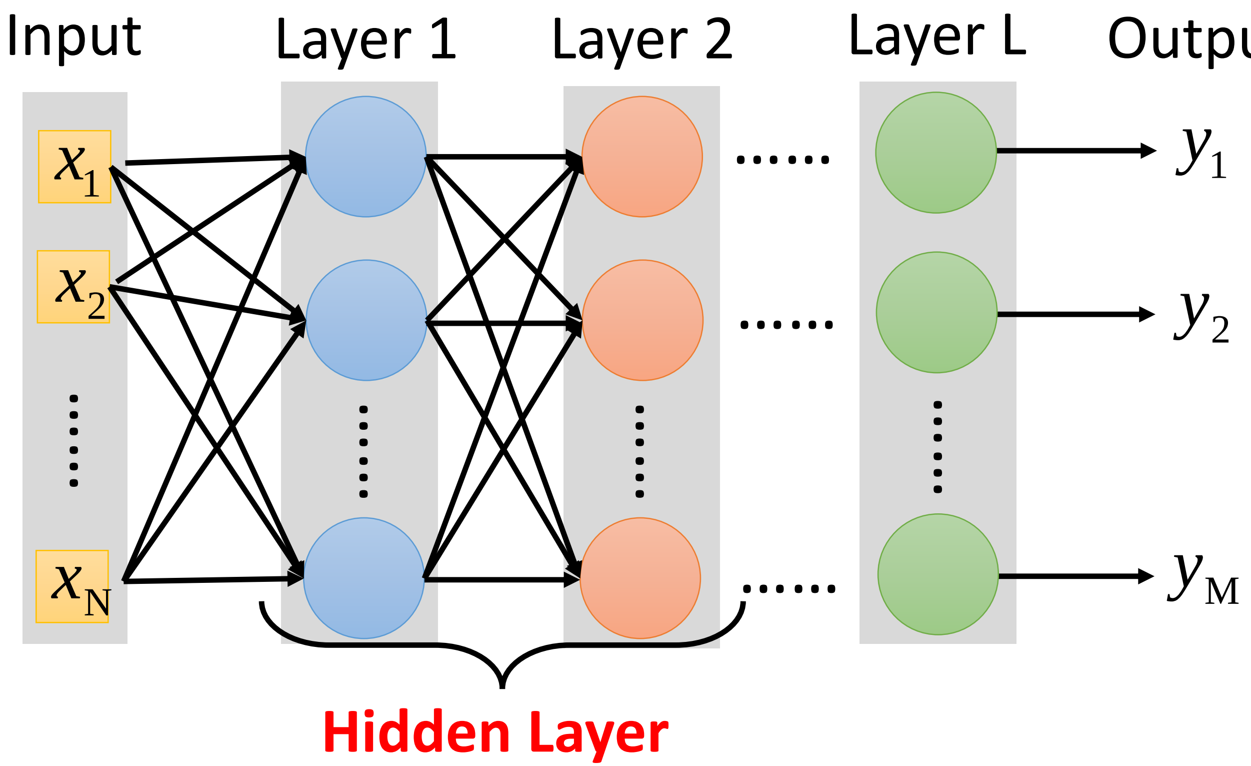
Deep Learning

A neural network is a complex function:

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

- Cascading the neurons to form a neural network.

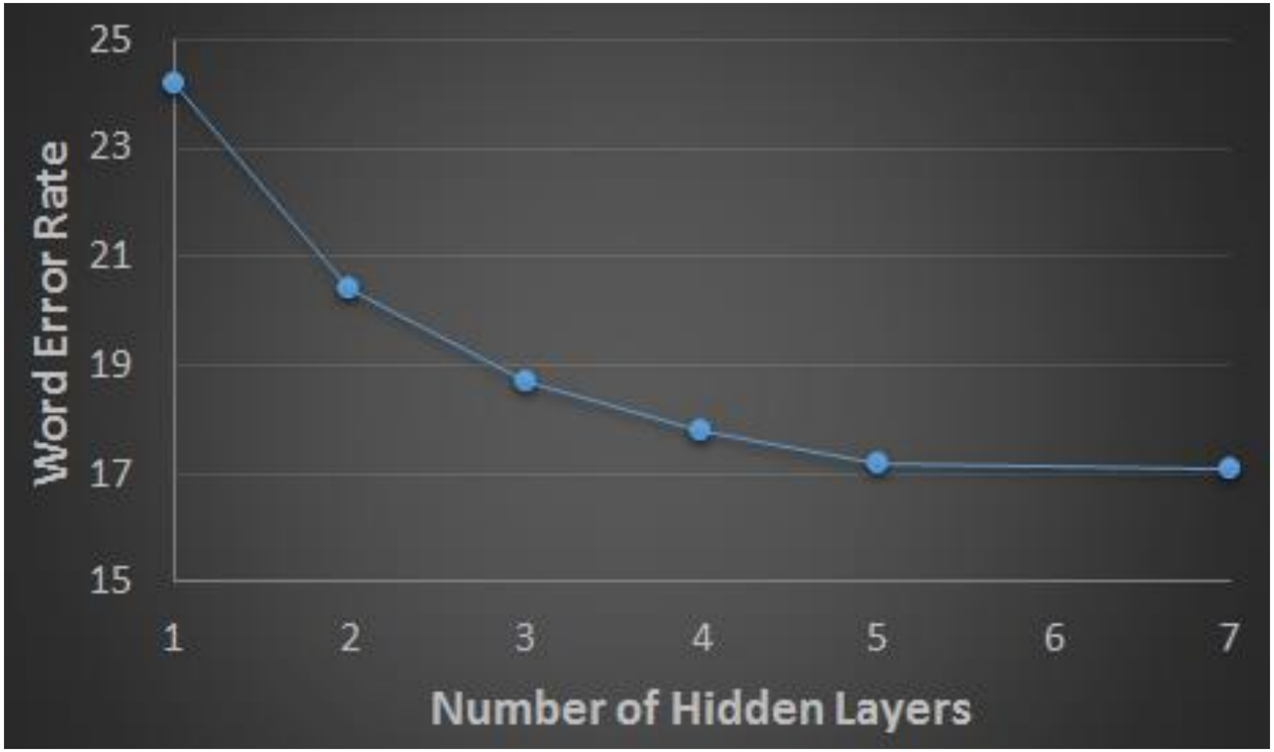
  Each layer is a simple function in the production line.



Why Deep Learning?

Deeper is Better.

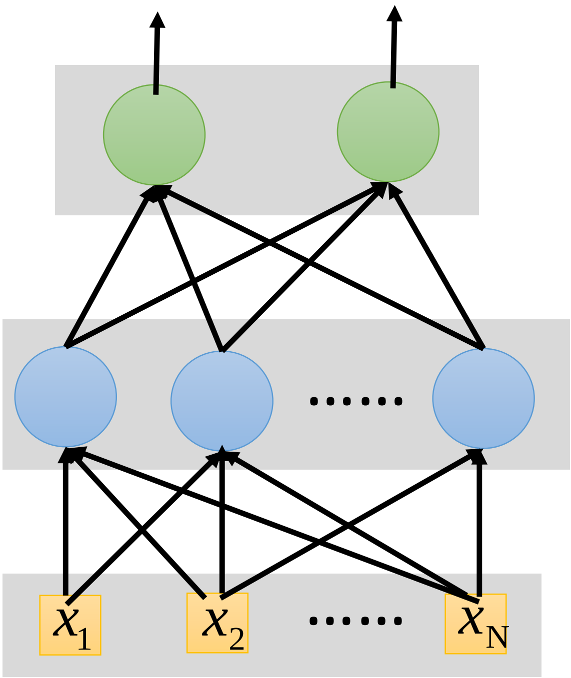
- Speech recognition



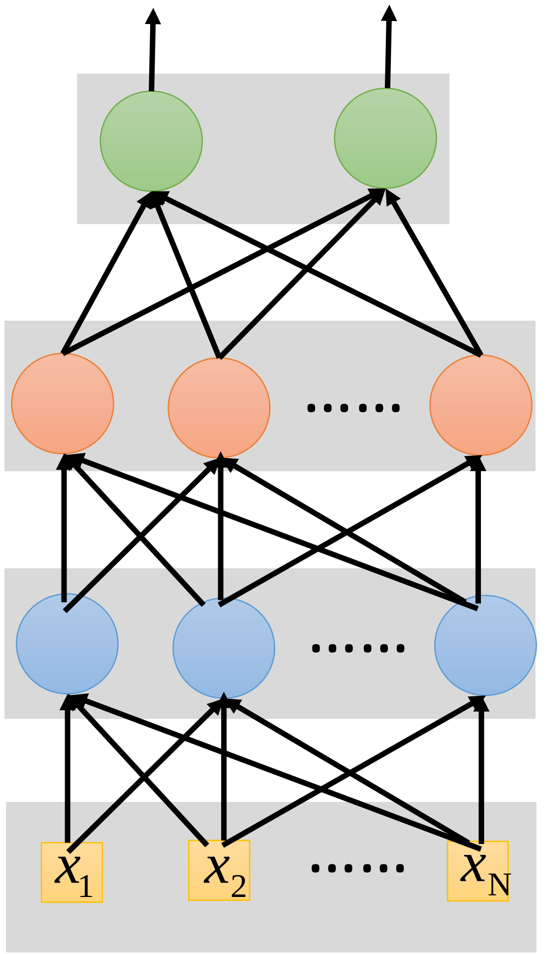
Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Why Deeper is Better?

Deep works better simply because it uses more parameters.



Shallow

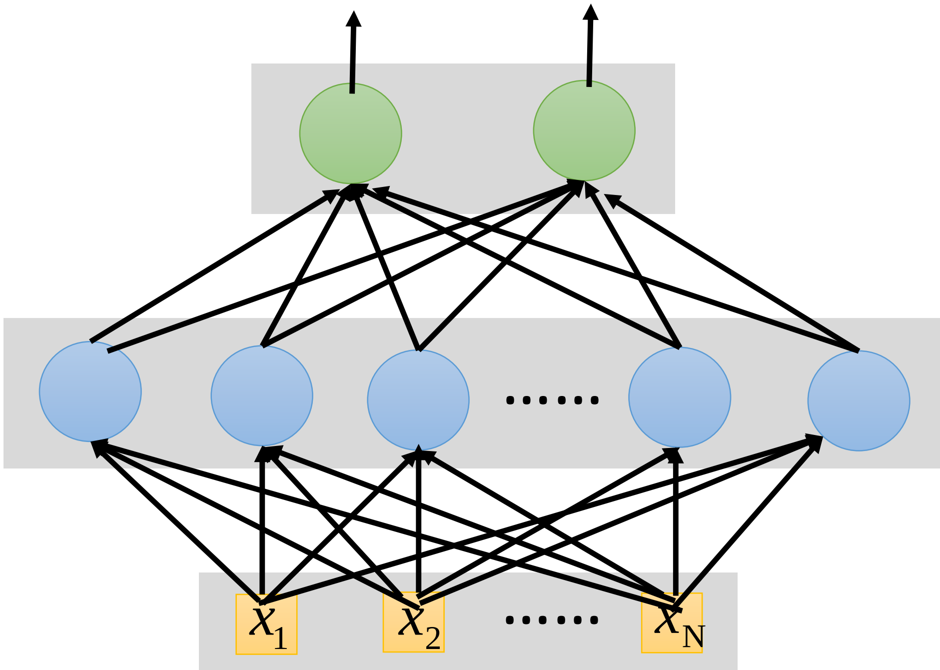


Deep

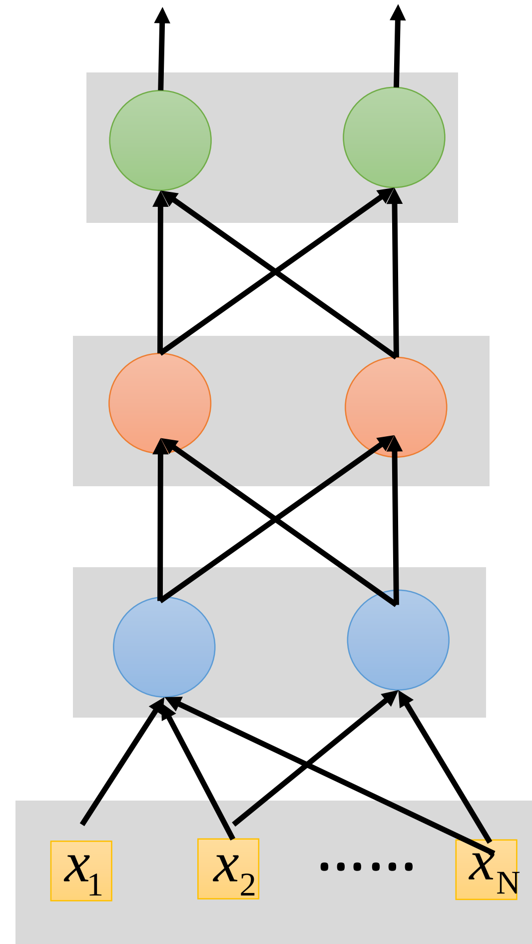
Fat + Short v.s. Thin + Tall

If they have the same parameters,

Which one is better?



Shallow



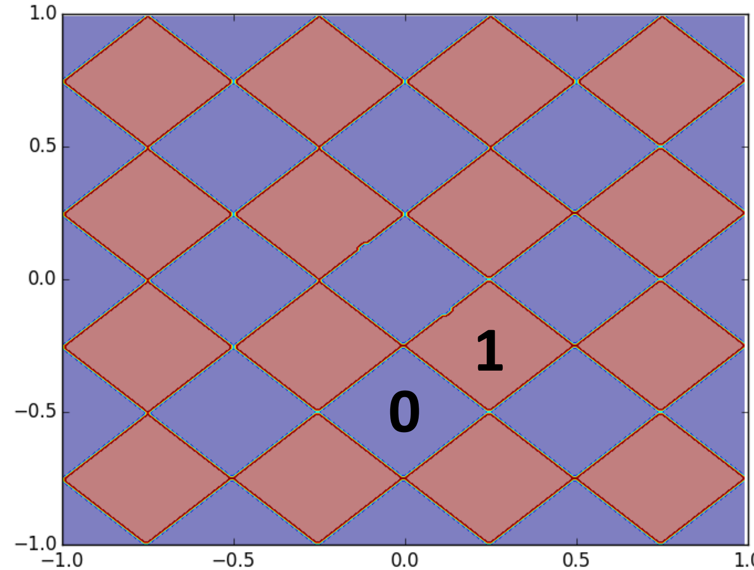
Deep

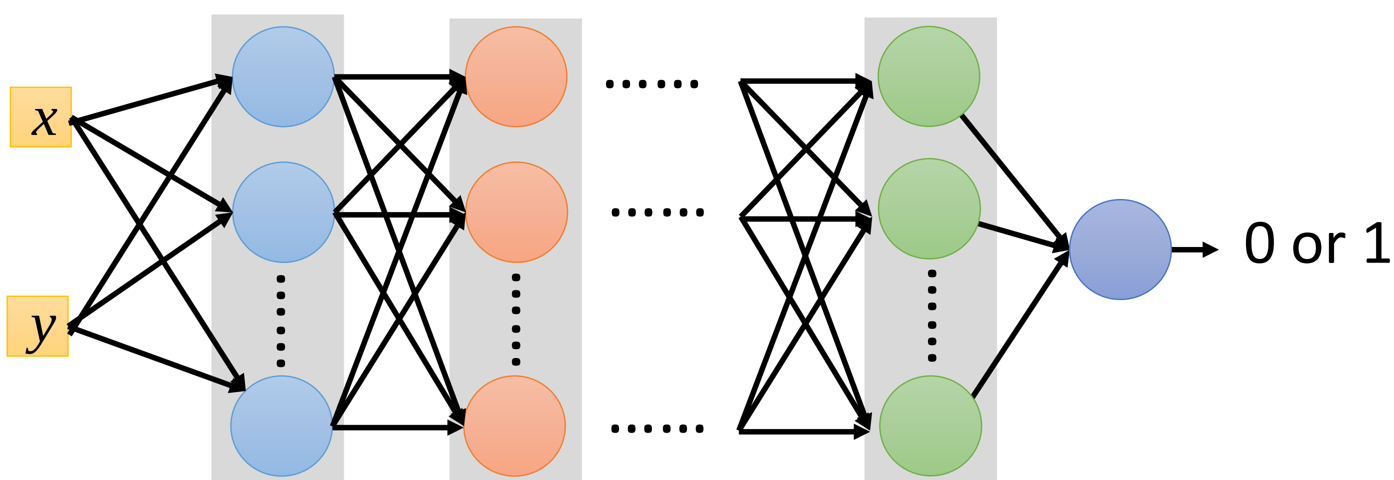
Fat + Short v.s. Thin + Tall

Toy Example

$$f : R^2 \rightarrow \{0,1\}$$

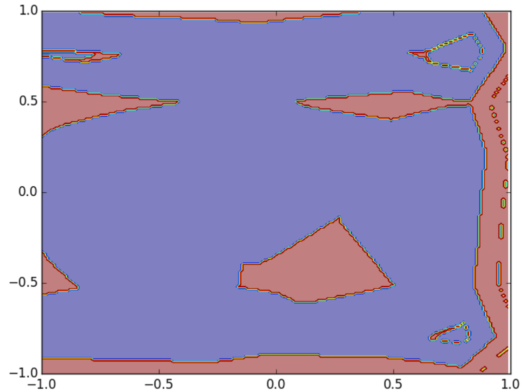
Sample 10,0000 points as training data



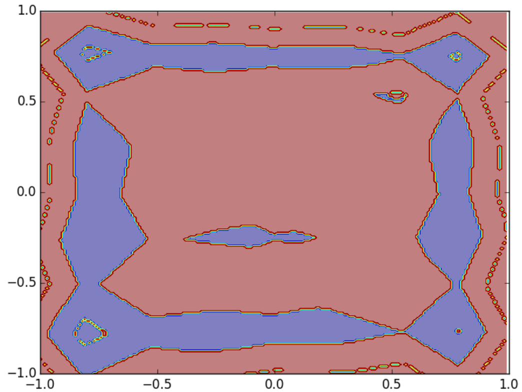


Fat + Short v.s. Thin + Tall
Toy Example

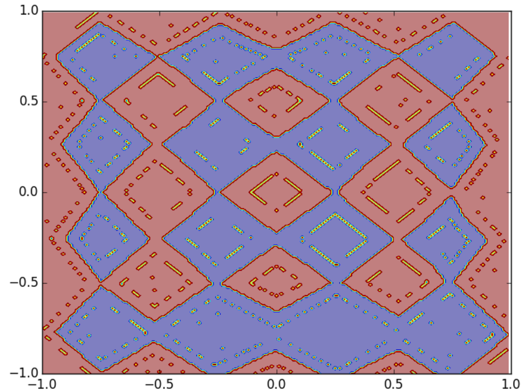
1 hidden layer:



(A) 125 neurons

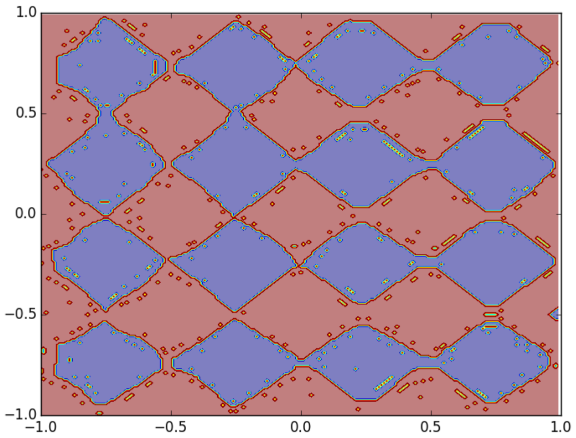


(B) 500 neurons



(C) 2500 neurons

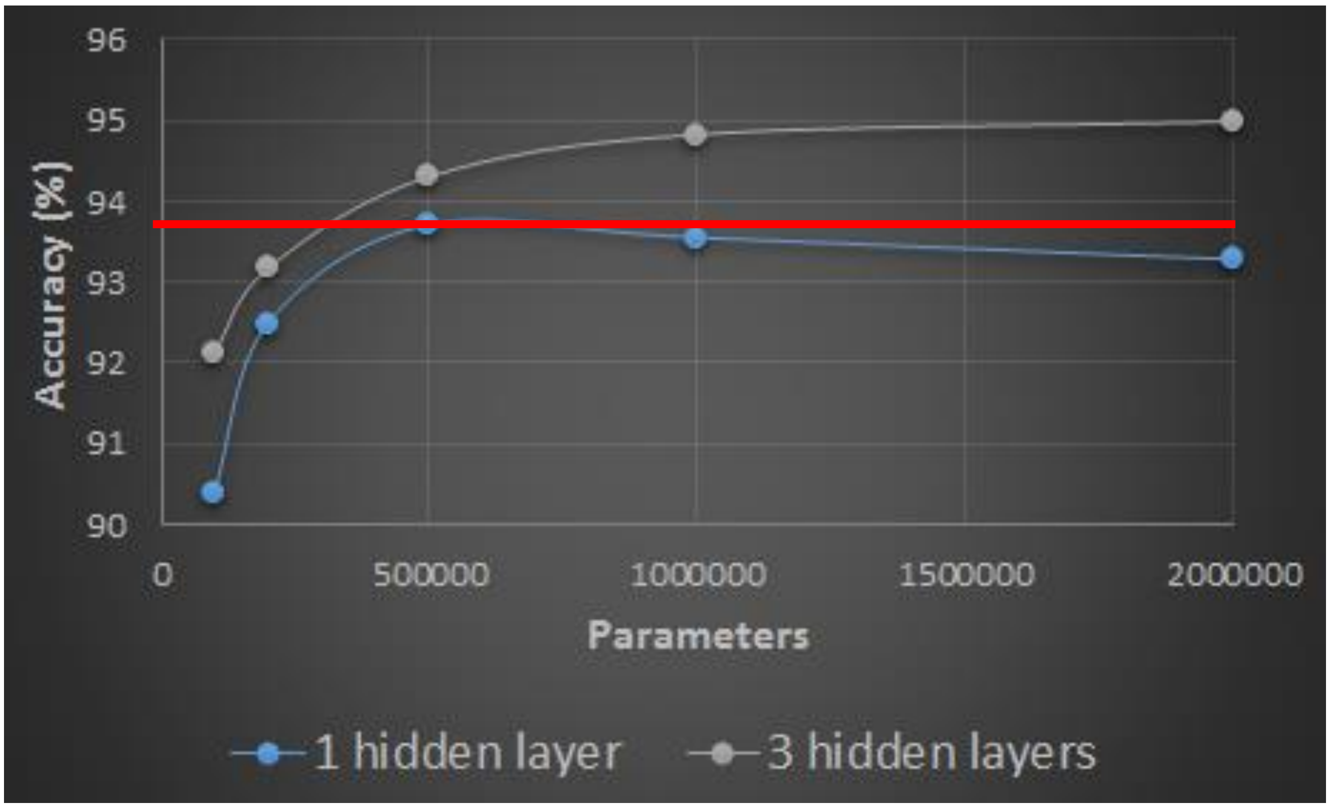
3 hidden layers:



Q: the number of parameters close to (A), (B) or (C)?

Fat + Short v.s. Thin + Tall
Hand-writing digit classification

- Same parameters



Deeper: Using less parameters to achieve the same performance

Fat + Short v.s. Thin + Tall
Speech Recognition

- Word error rate (WER)

Multiple layers | 1 hidden layer

LxN	DBN-PT (%)	1xN	DBN-PT (%)
1×2k	24.2		
2×2k	20.4		
3×2k	18.4		
4×2k	17.8		
5×2k	17.2	1×3,772	22.5
7×2k	17.1	1×4,634	22.6
		1×16K	22.1

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

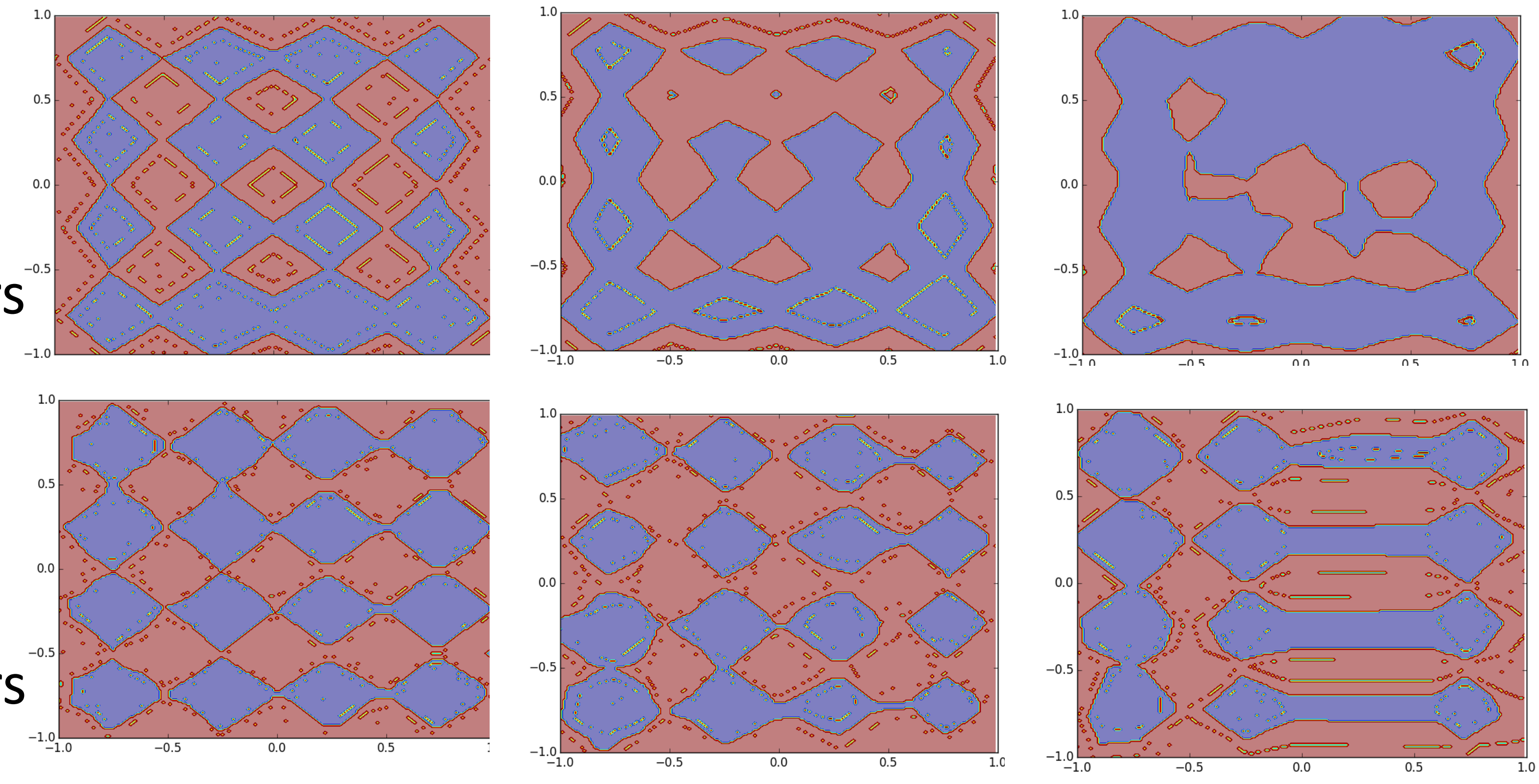
Size of Training Data

- Different numbers of training examples

10,0000 | 5,0000 | 2,0000

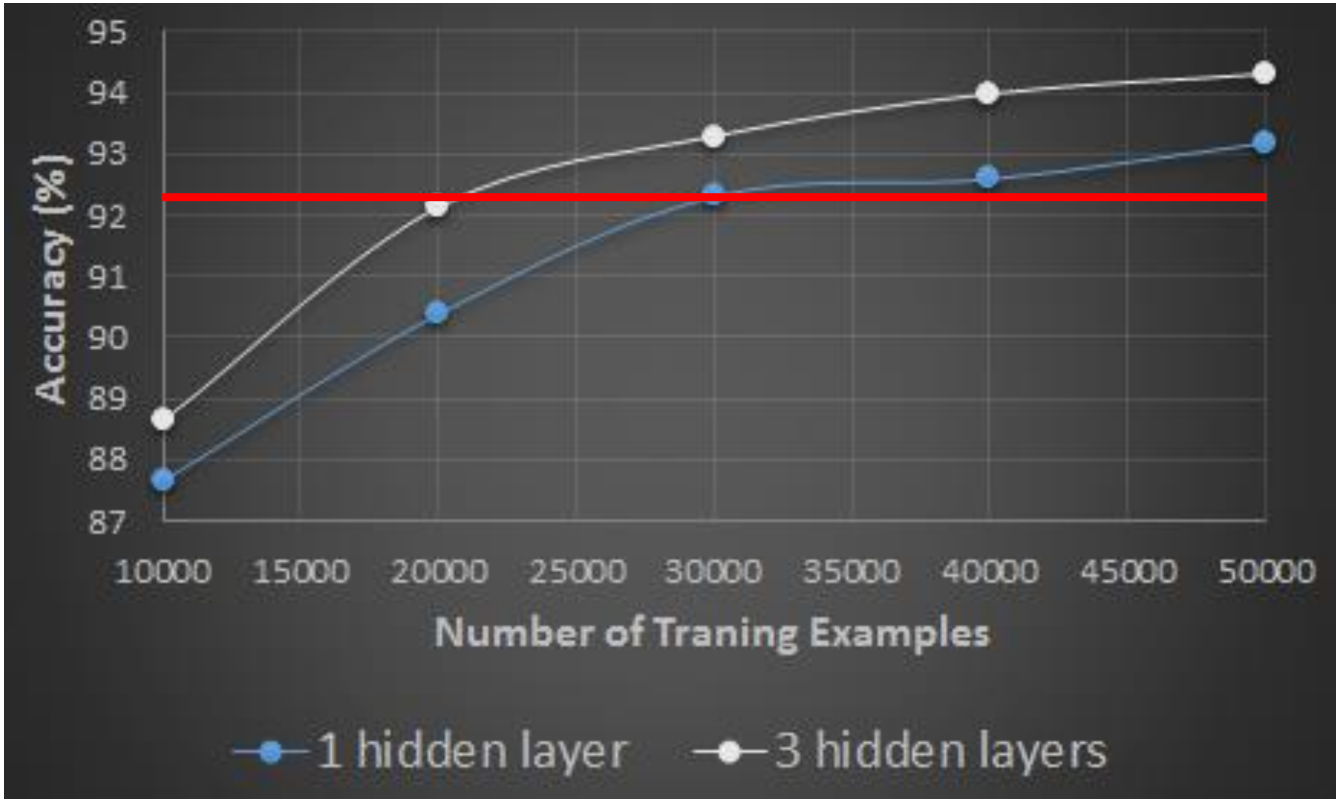
1 hidden layer
More parameters

3 hidden layers
Less parameters



Size of Training Data

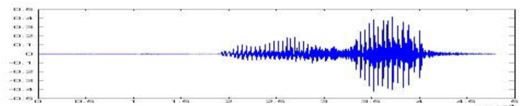
- Hand-writing digit classification



Deeper: Using less training data to achieve the same performance

Learning ≈ Looking for a Function

- Speech Recognition

$f($  $)=$ “你好”

- Handwritten Recognition

$f($  $)=$ “2”

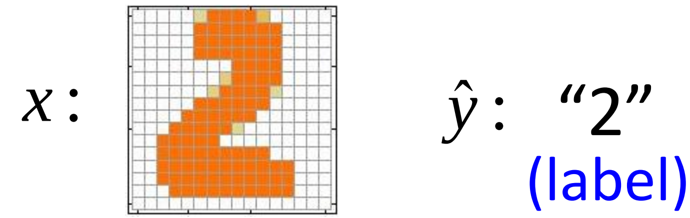
- Weather forecast

$f($ weather today $)=$ “sunny tomorrow”

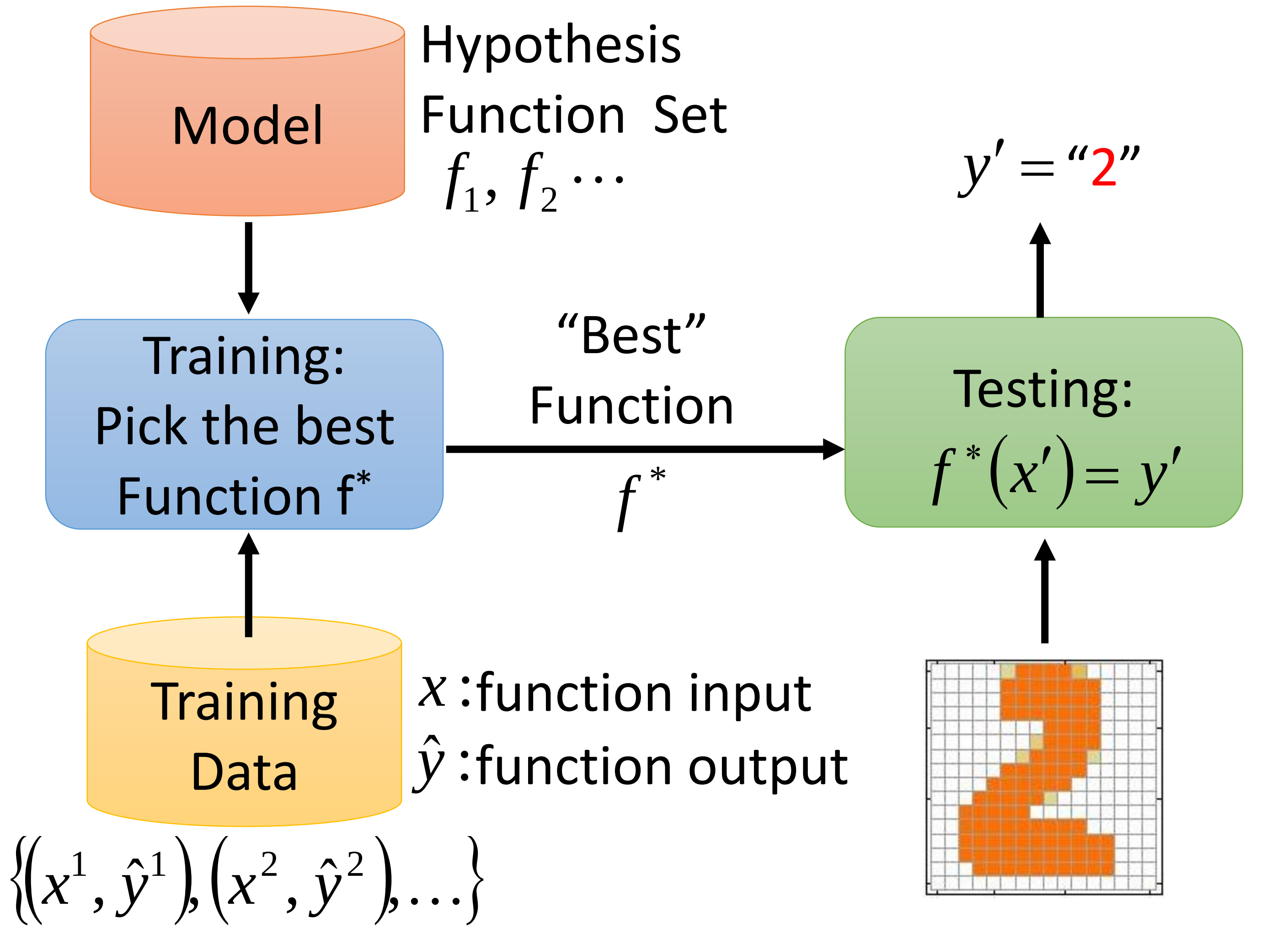
- Play video games

$f($ Positions and number of enemies $)=$ “fire”

Framework



$x:$
$\hat{y}:$ "2"
(label)



Model
Hypothesis Function Set
$f_1, f_2 \cdots$
Training: Pick the best Function f*
"Best" Function
f^*
Testing:
$f^*(x') = y'$
$y' = $ "2"
Training Data
x :function input
$\hat{y}$:function output
$\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \ldots\}$

Outline

1. What is the model (function hypothesis set)?

2. What is the "best" function?

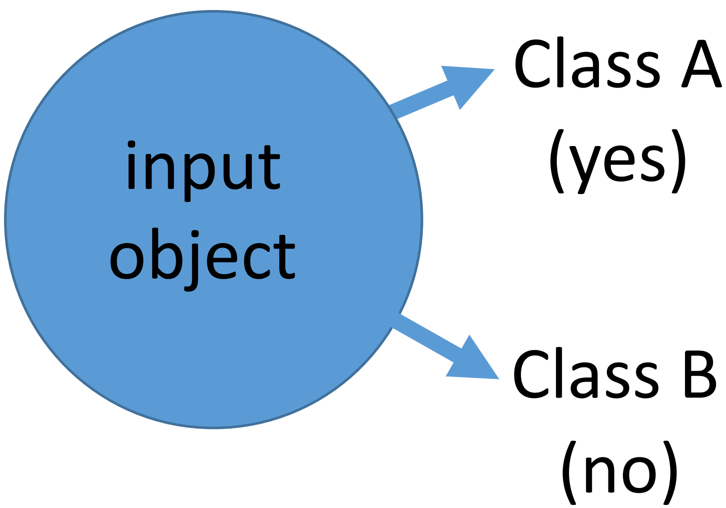
3. How to pick the "best" function?

Task Considered Today

- ***Classification***

Binary Classification

Only two classes



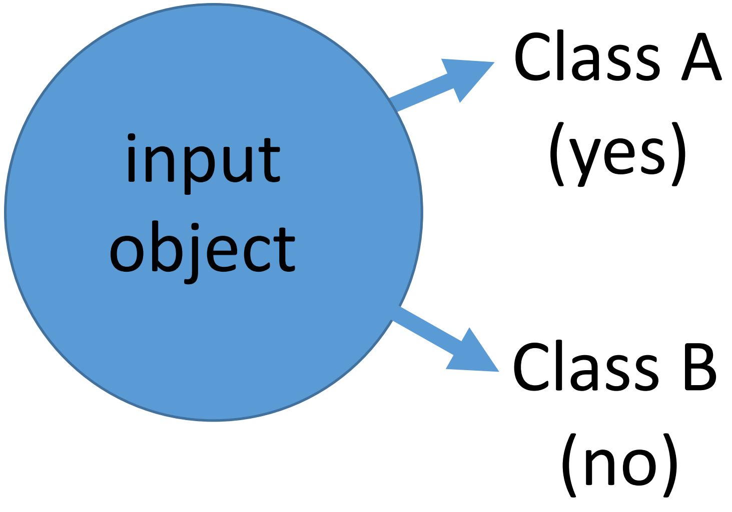
- **Spam filtering**
  - Is an e-mail spam or not?
- **Recommendation systems**
  - recommend the product to the customer or not?
- **Malware detection**
  - Is the software malicious or not?
- **Stock prediction**
  - Will the future value of a stock increase or not?

Task Considered Today

- ***Classification***

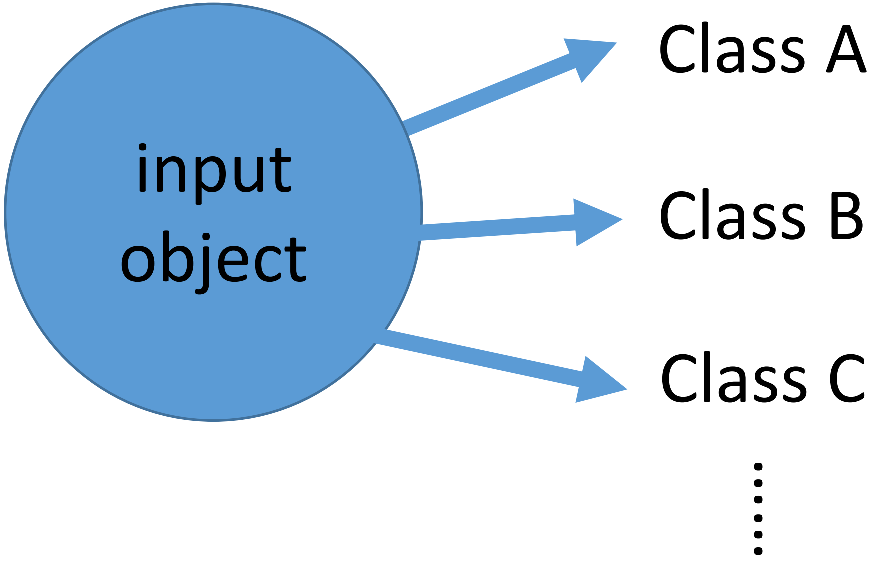
Binary Classification

Only two classes



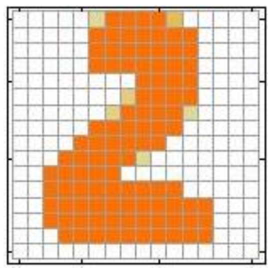
Multi-class Classification

More than two classes



Multi-class Classification

- Handwriting Digit Classification

Input:  Class: “1”, “2”, …., “9”, “0”

10 classes

- Image Recognition

Input:  Class: “dog”, “cat”, “book”, ….

Thousands of classes

1. What is the model?

What is the function we are looking for?

- ***classification***

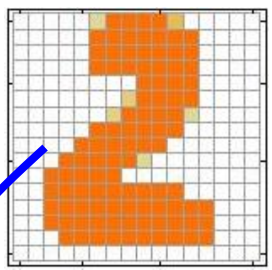
$$y = f(x) \quad \Rightarrow \quad f: R^N \to R^M$$

- x: input object to be classified
- y: class
- ***Assume both x and y can be represented as fixed-size vector***
  - x is a vector with N dimensions, and y is a vector with M dimensions

What is the function we are looking for?

- ***<u>Handwriting Digit Classification</u>*** $f: R^N \rightarrow R^M$

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

10 dimensions for digit recognition

$$\begin{array}{cc} \text{“1”} & \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{array}{c} \text{“1”} \\ \text{“2”} \\ \text{“3”} \\ \\ \end{array} \end{array} \qquad \begin{array}{c} \text{“2”} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

“1” → “1” or not

“2” → “2” or not

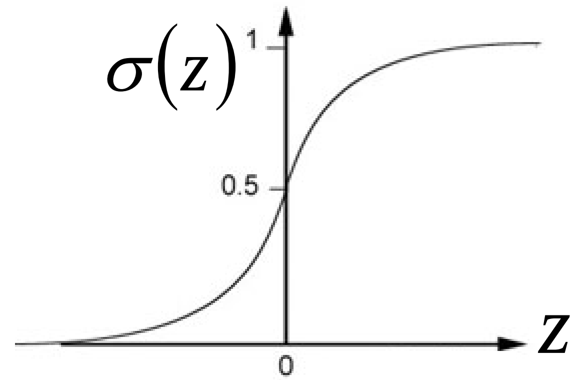
“3” → “3” or not

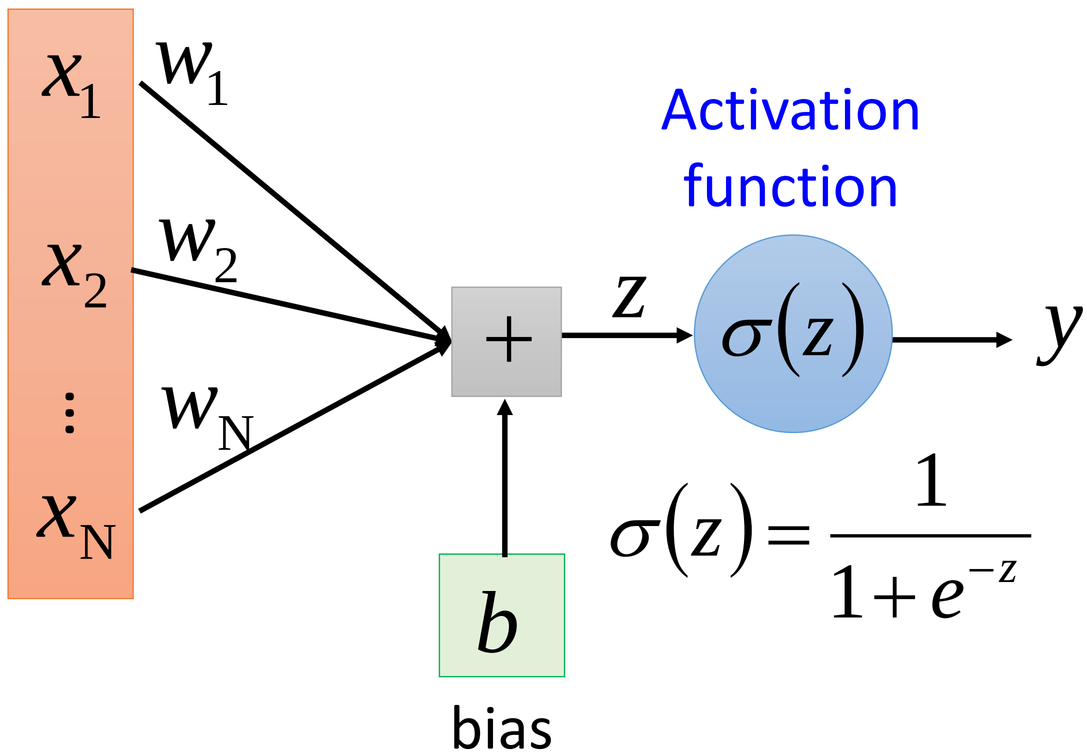
1. What is the model?

A Layer of Neuron

Single Neuron

$$f: R^N \to R$$

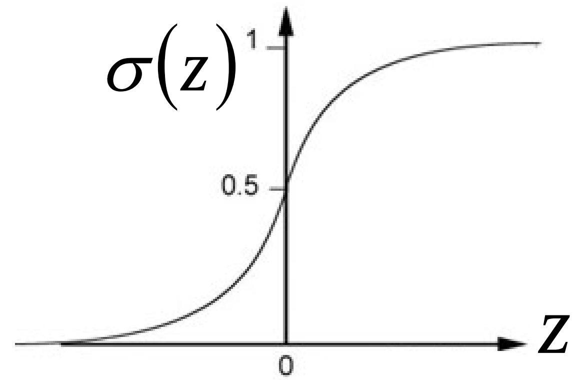


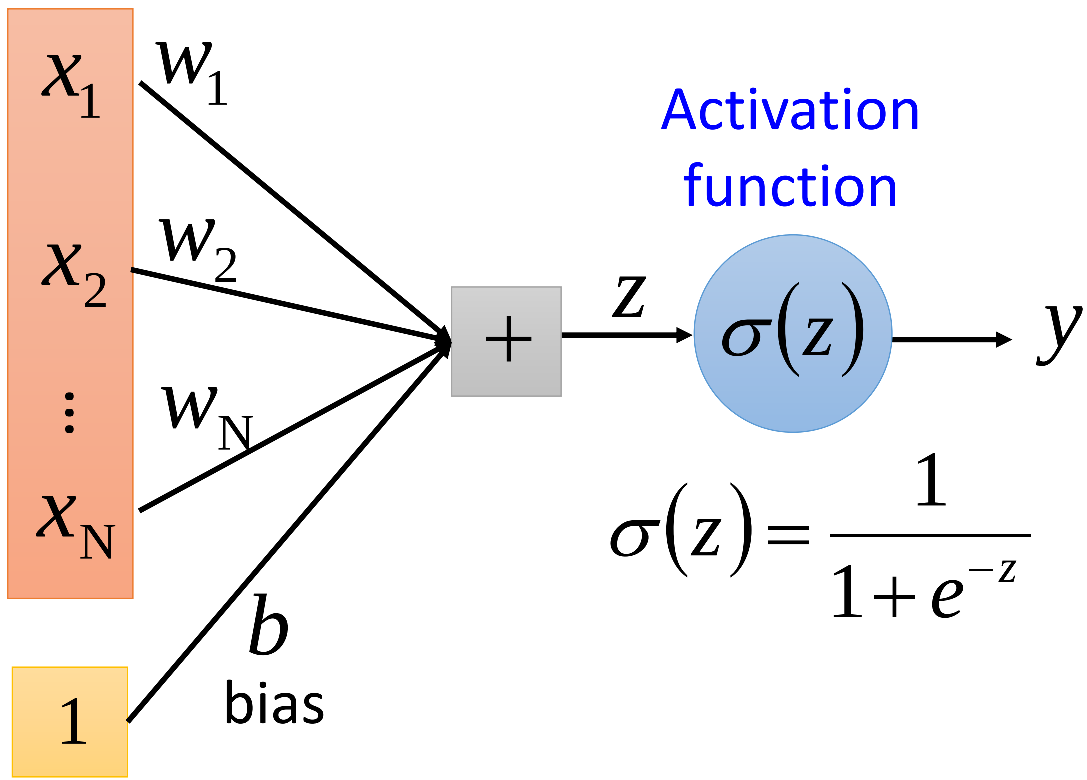


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Single Neuron

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

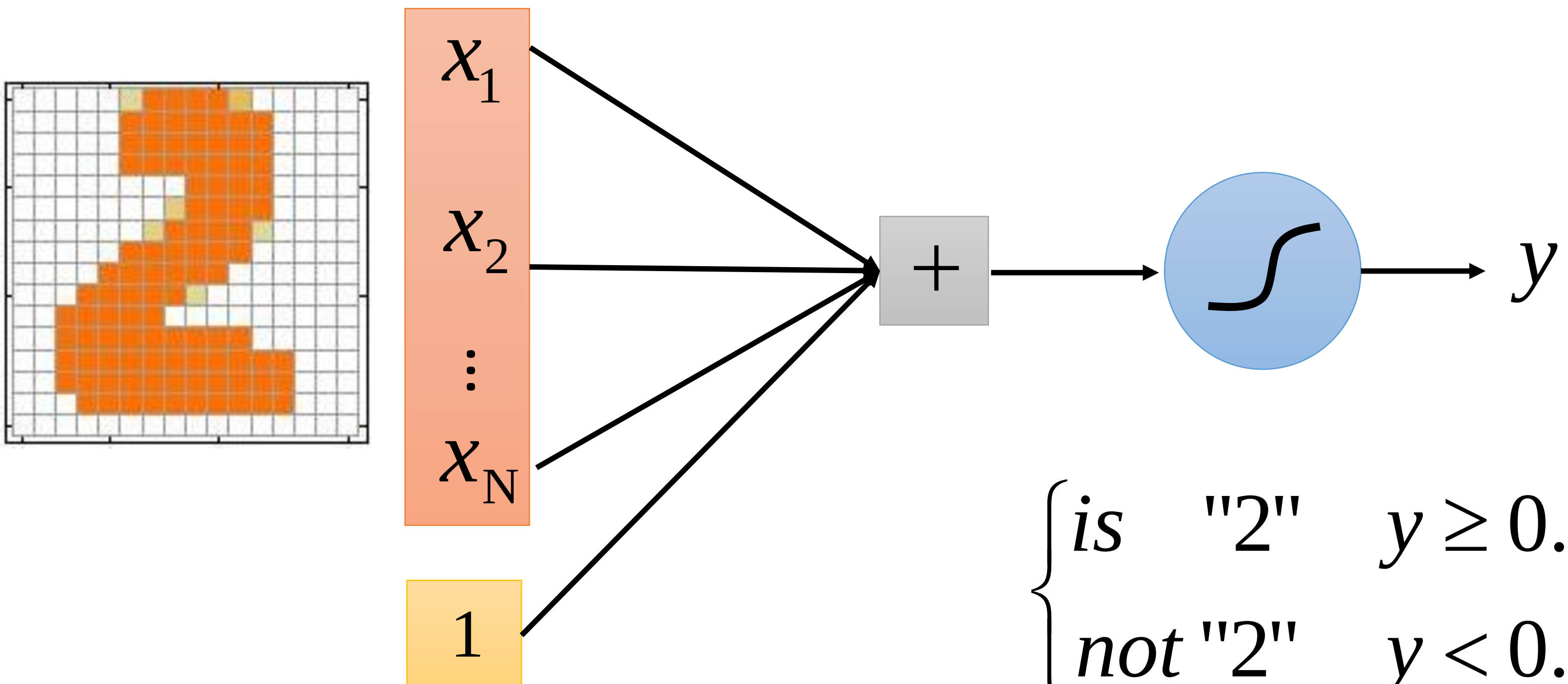




$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Single Neuron $f: R^N \to R$

- Single neuron can only do binary classification, cannot handle multi-class classification

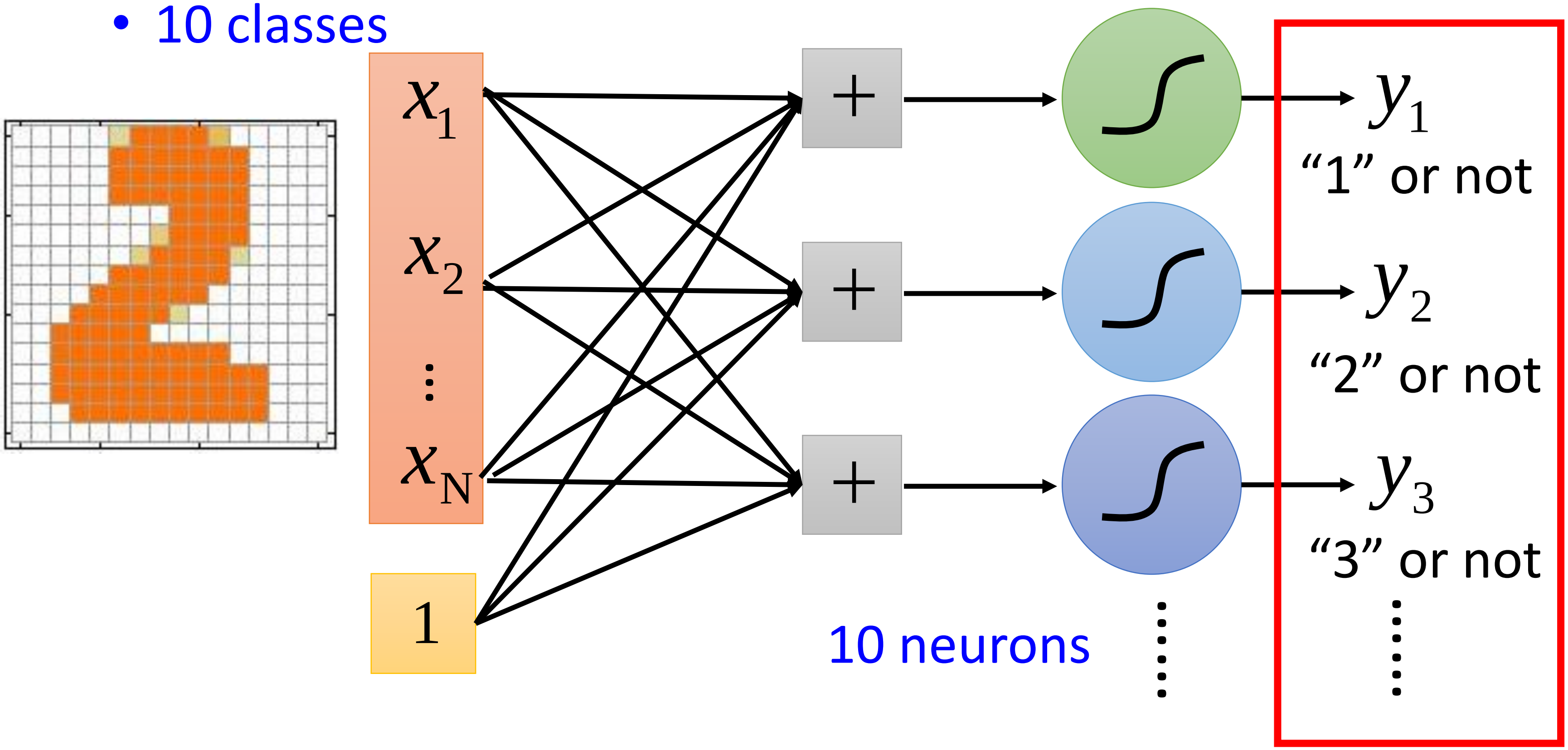


$$\begin{cases} is \quad "2" & y \geq 0.5 \\ not\ "2" & y < 0.5 \end{cases}$$

A Layer of Neuron $f: R^N \rightarrow R^M$

- Handwriting digit classification
  - Classes: “1”, “2”, …., “9”, “0”
  - 10 classes

If y_2 is the max, then the image is “2”.

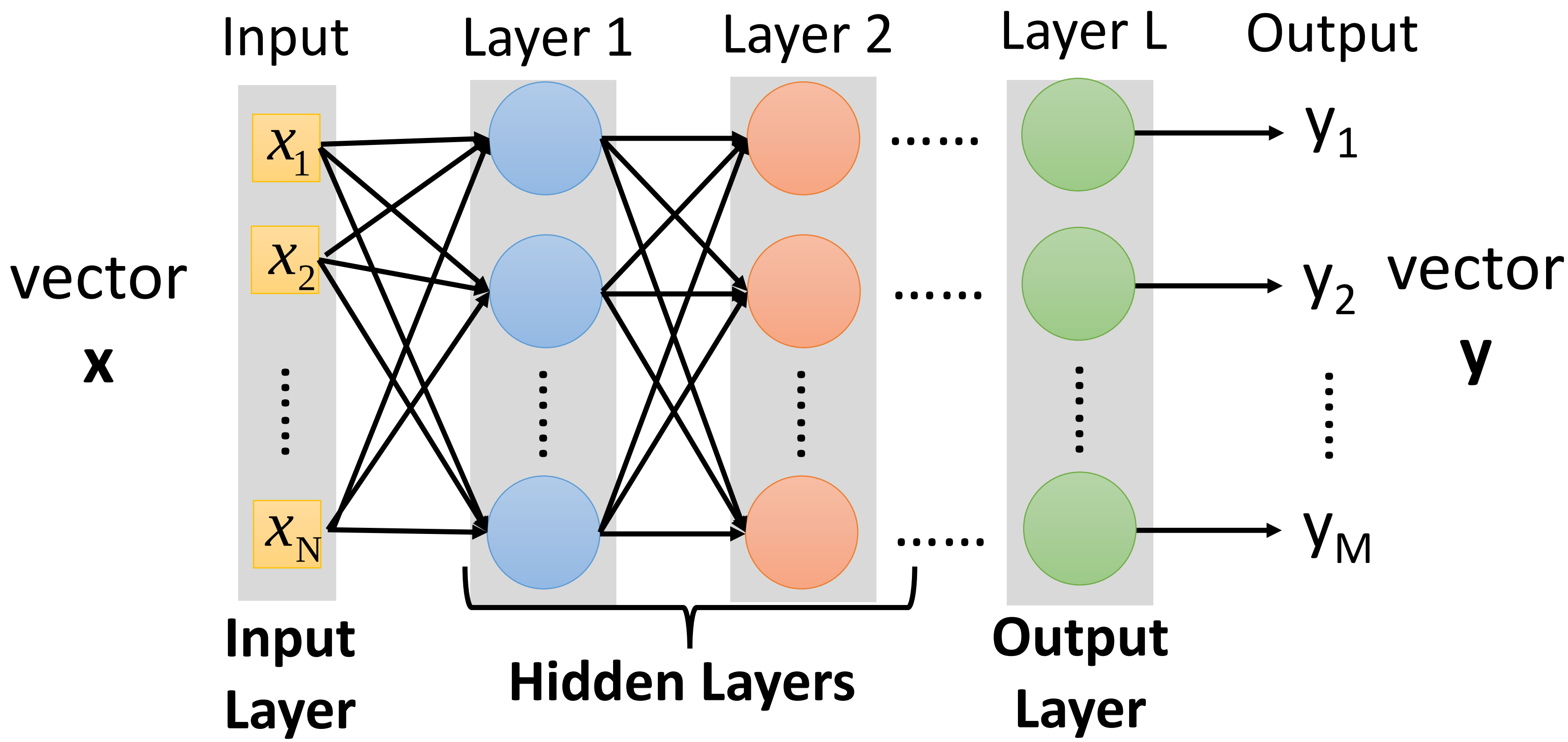


1. What is the model?

Neural Network

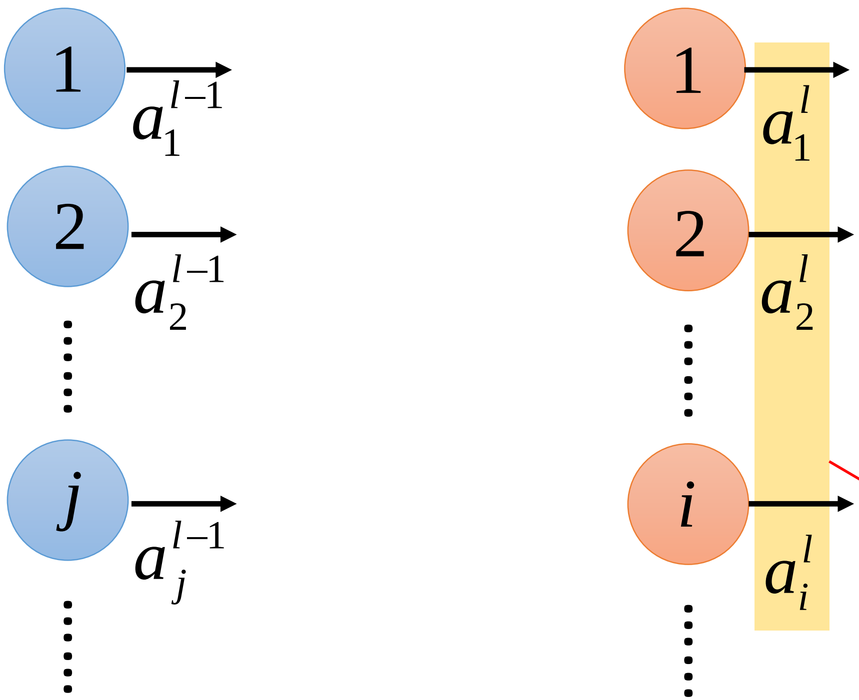
Neural Network as Model

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



- Fully connected feedforward network
- Deep Neural Network: many hidden layers

Notation



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

Output of a neuron:

a_i^l

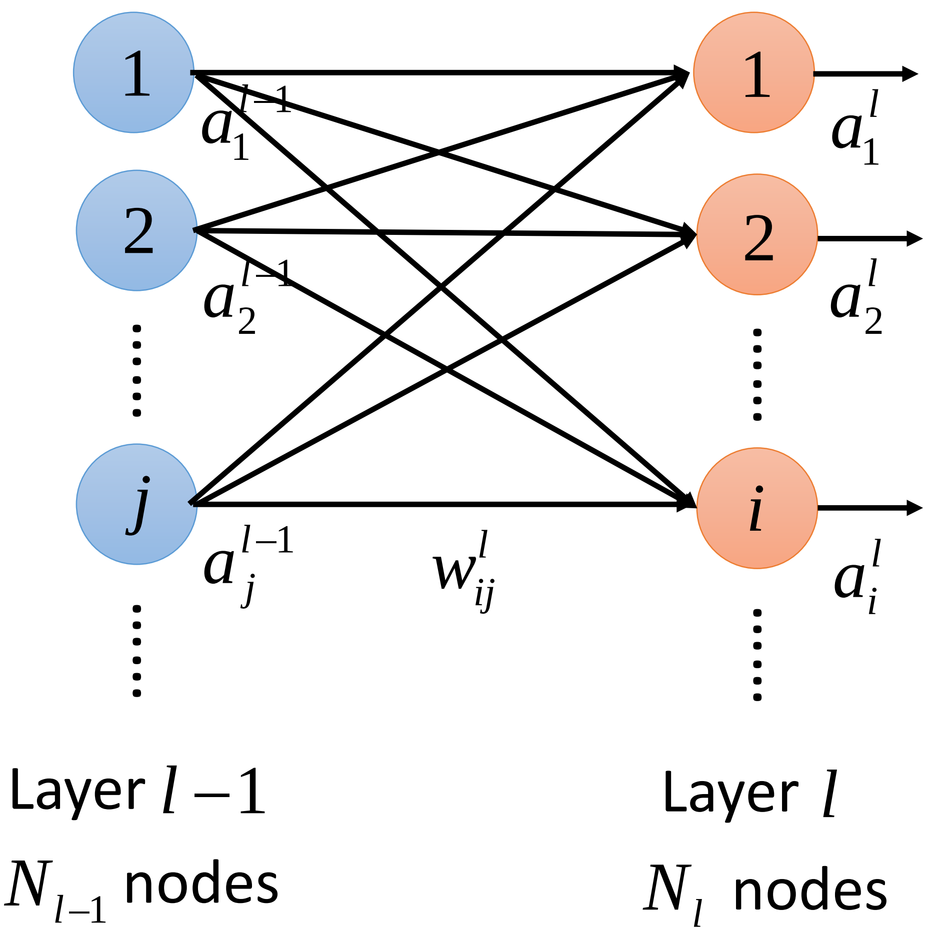
l → Layer l

i → Neuron i

Output of one layer:

a^l : a vector

Notation



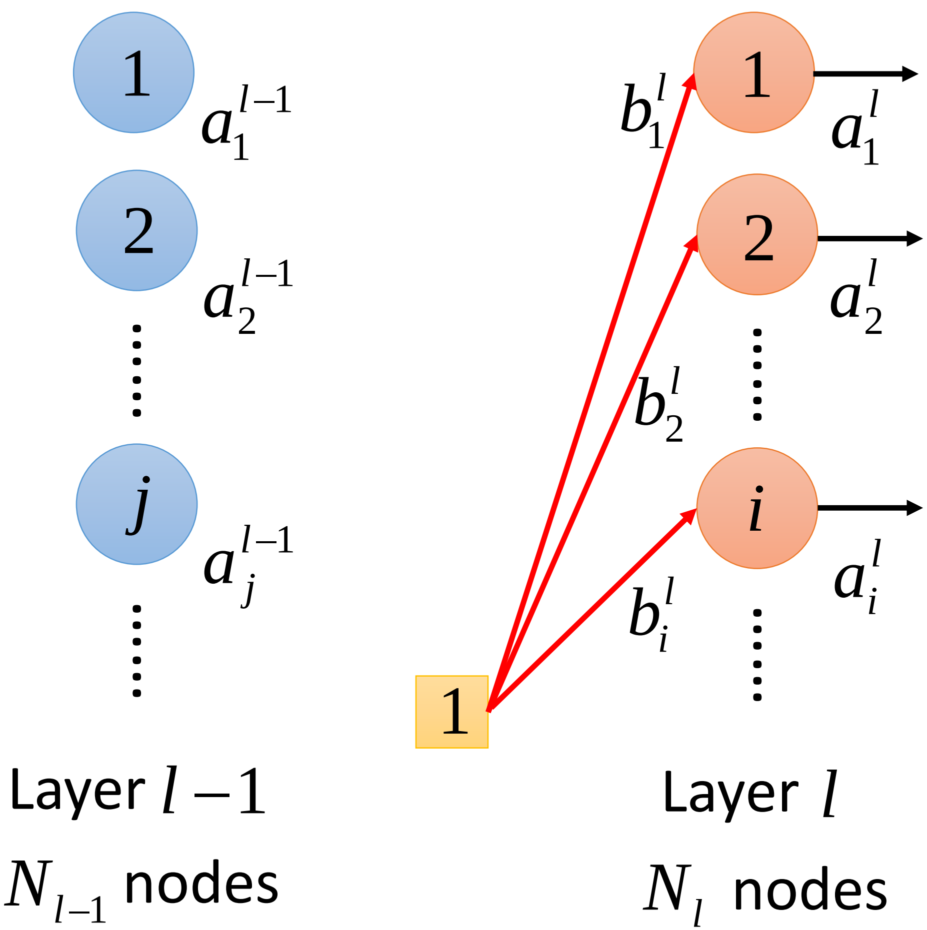
w_{ij}^l

Layer $l-1$ to Layer l

from neuron j (Layer $l-1$)
to neuron i (Layer l)

$$W^l = \overbrace{\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \\ & & \end{bmatrix}}^{N_{l-1}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}}\right\} N_l$$

Notation



b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias for all neurons in layer l

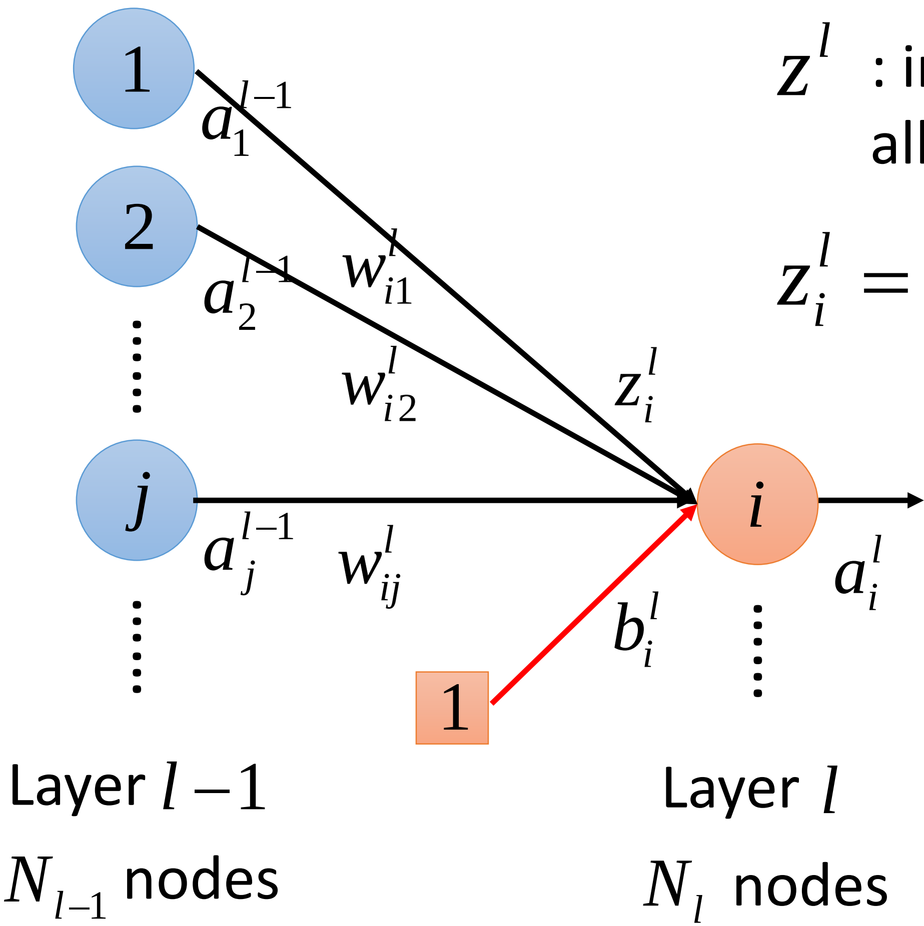
Notation

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

z^l : input of the activation function all the neurons in layer l

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

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



Notation - Summary

a_i^l :output of a neuron

a^l :output of a layer

z_i^l : input of activation function

z^l : input of activation function for a layer

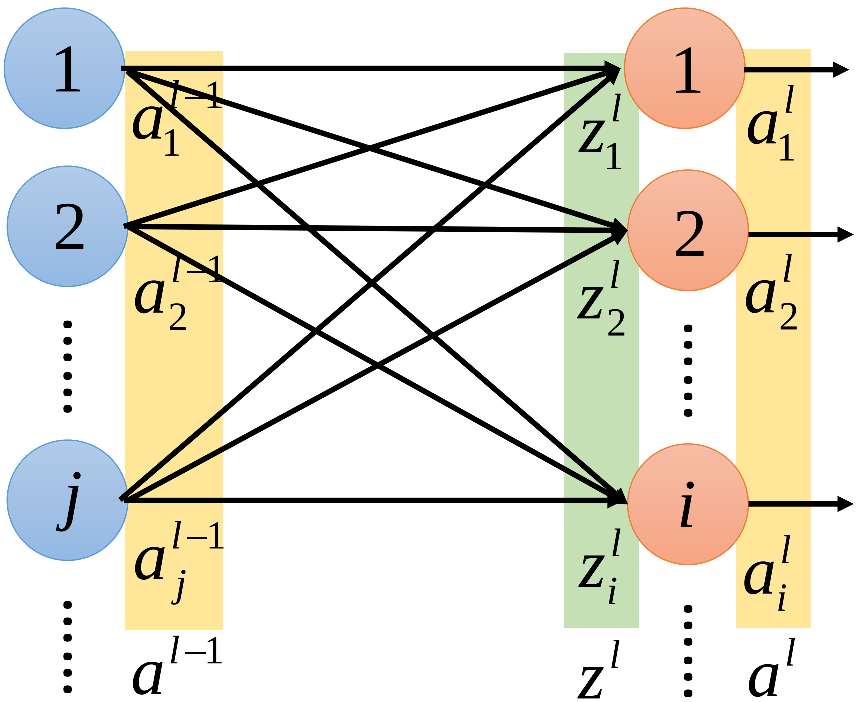
w_{ij}^l : a weight

W^l : a weight matrix

b_i^l : a bias

b^l : a bias vector

Relations between Layer Outputs



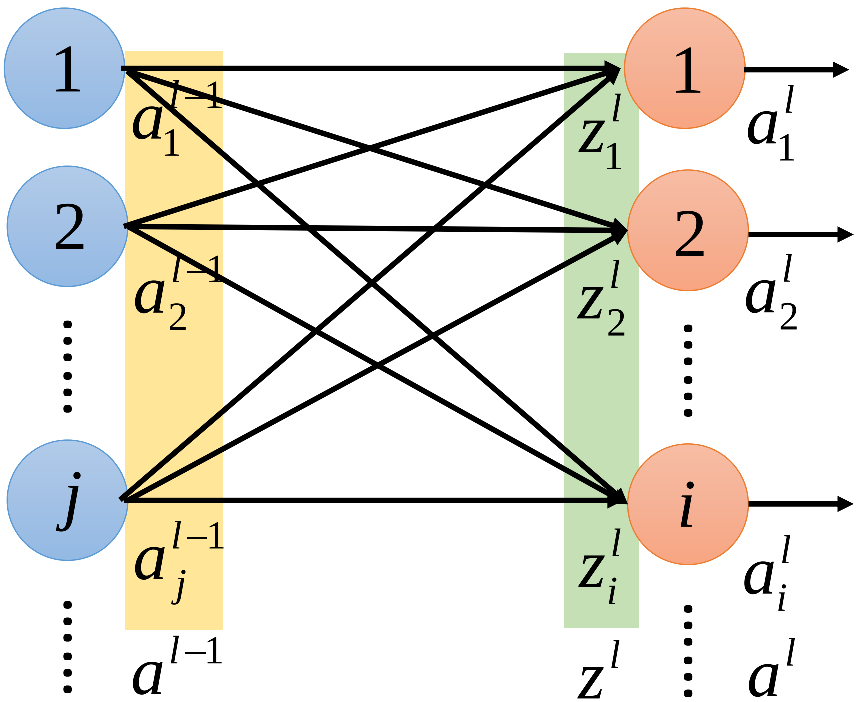
Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

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

$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \cdots + b_2^l$$

$$\vdots$$

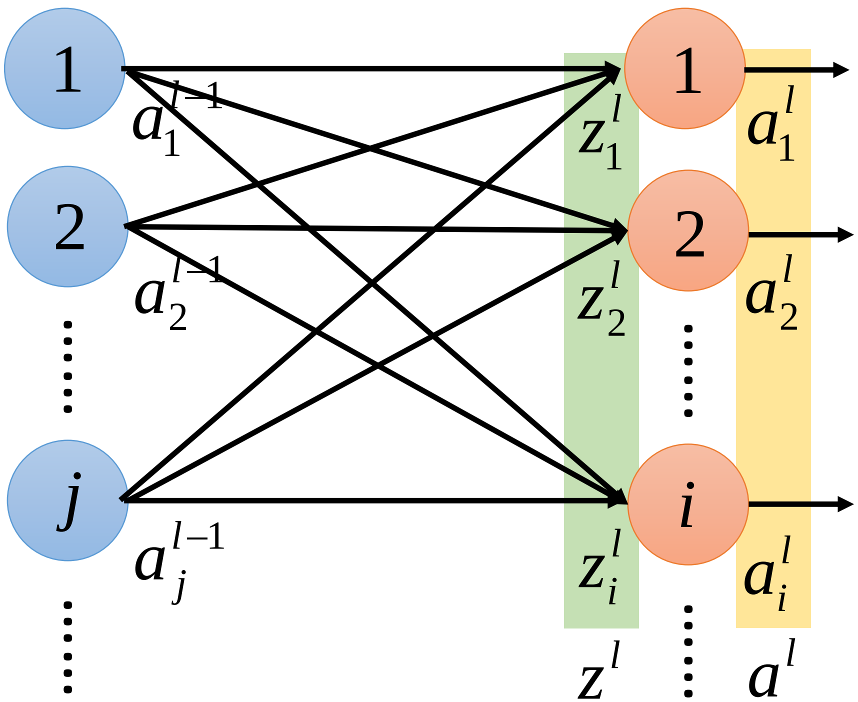
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

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

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

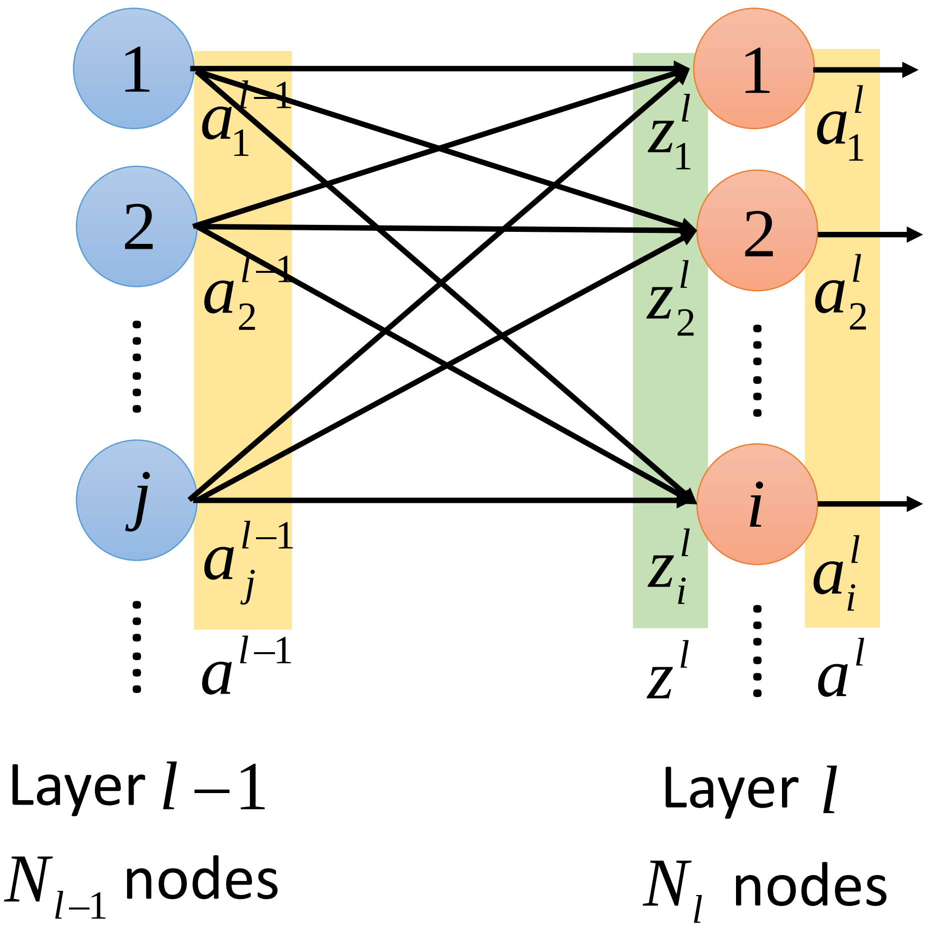
N_l nodes

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

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

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

Relations between Layer Outputs

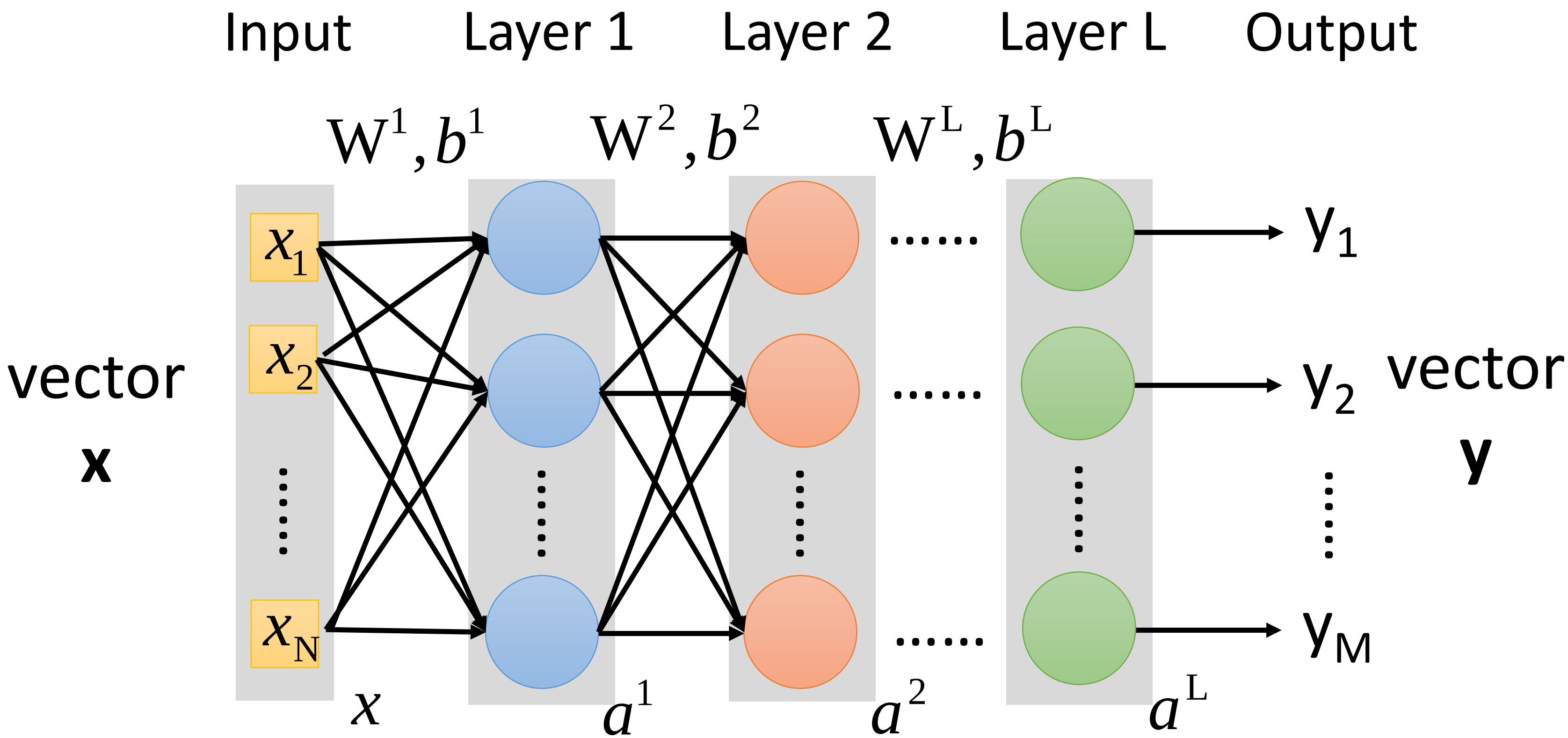


$$z^l = W^l a^{l-1} + b^l$$

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

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

Function of Neural Network

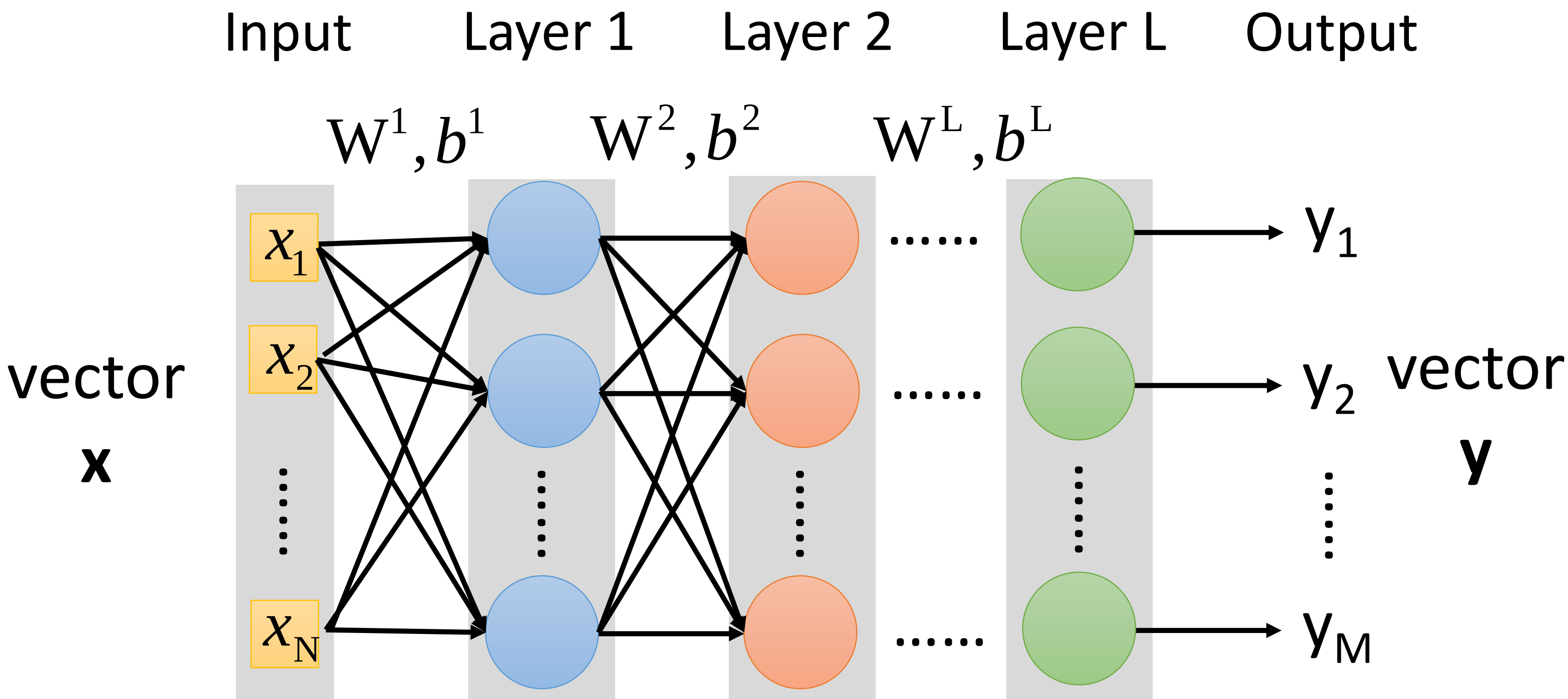


$$\sigma\left(W^1 x + b^1\right) = a^1$$

$$\sigma\left(W^2 a^1 + b^2\right) = a^2$$

$$\sigma\left(W^L a^{L-1} + b^L\right) = a^L = y$$

Function of Neural Network



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

2. What is the “best” function?

Best Function = Best Parameters

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

function set

because different parameters W and b lead to different function

Formal way to define a function set:

$$f(x; \underline{\theta}) \rightarrow \text{parameter set}$$

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

Pick the "best" function f*

Pick the "best" parameter set θ*

Cost Function

- Define a function for parameter set $C(\theta)$
  - $C(\theta)$ evaluate how bad a parameter set is
  - The best parameter set θ^* is the one that minimizes $C(\theta)$

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

- $C(\theta)$ is called ***cost/loss/error function***
  - If you define the goodness of the parameter set by another function $O(\theta)$
  - $O(\theta)$ is called objective function

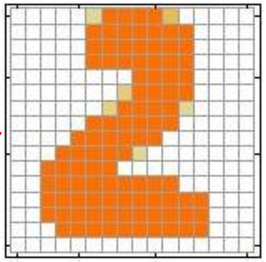
Cost Function

Given training data:

$$\left\{\left(x^1, \hat{y}^1\right) \ldots \left(x^r, \hat{y}^r\right) \ldots \left(x^R, \hat{y}^R\right)\right\}$$

- ***Handwriting Digit Classification***

sum over all training examples



$$C(\theta) = \frac{1}{R} \sum_r \left\| f\left(\boxed{x^r}; \theta\right) - \hat{y}^r \right\|$$

Minimize distance

$$\begin{array}{cc} \begin{bmatrix} 0.1 \\ 0.4 \\ 0.2 \\ \vdots \end{bmatrix} & \begin{array}{l} \text{“1”} \\ \text{“2”} \\ \text{“3”} \\ \end{array} \end{array} \qquad \begin{array}{cc} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} & \begin{array}{l} \text{“1”} \\ \text{“2”} \\ \text{“3”} \\ \end{array} \end{array}$$

3. How to pick the “best” function?

Gradient Descent

Statement of Problems

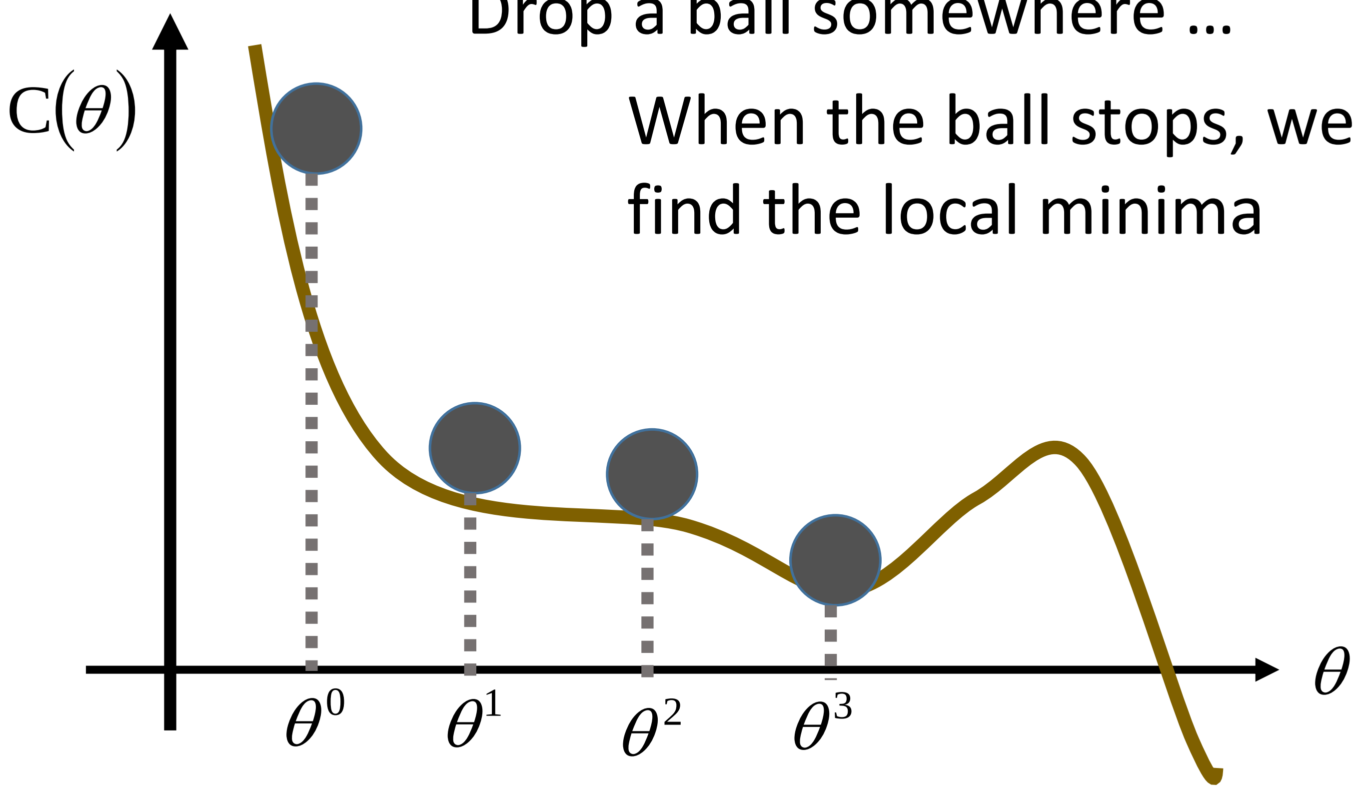
- Statement of problems:
  - There is a function $C(\theta)$
    - θ represents parameter set
    - $\theta = \{\theta_1, \theta_2, \theta_3, \ldots\ldots\}$
  - Find θ^* that minimizes $C(\theta)$
- Brute force?
  - Enumerate all possible θ
- Calculus?
  - Find θ^* such that $\left.\frac{\partial C(\theta)}{\partial \theta_1}\right|_{\theta=\theta^*} = 0, \left.\frac{\partial C(\theta)}{\partial \theta_2}\right|_{\theta=\theta^*} = 0, \ldots$

Gradient Descent – Idea

- For simplification, first consider that θ has only one variable

Drop a ball somewhere …

When the ball stops, we find the local minima

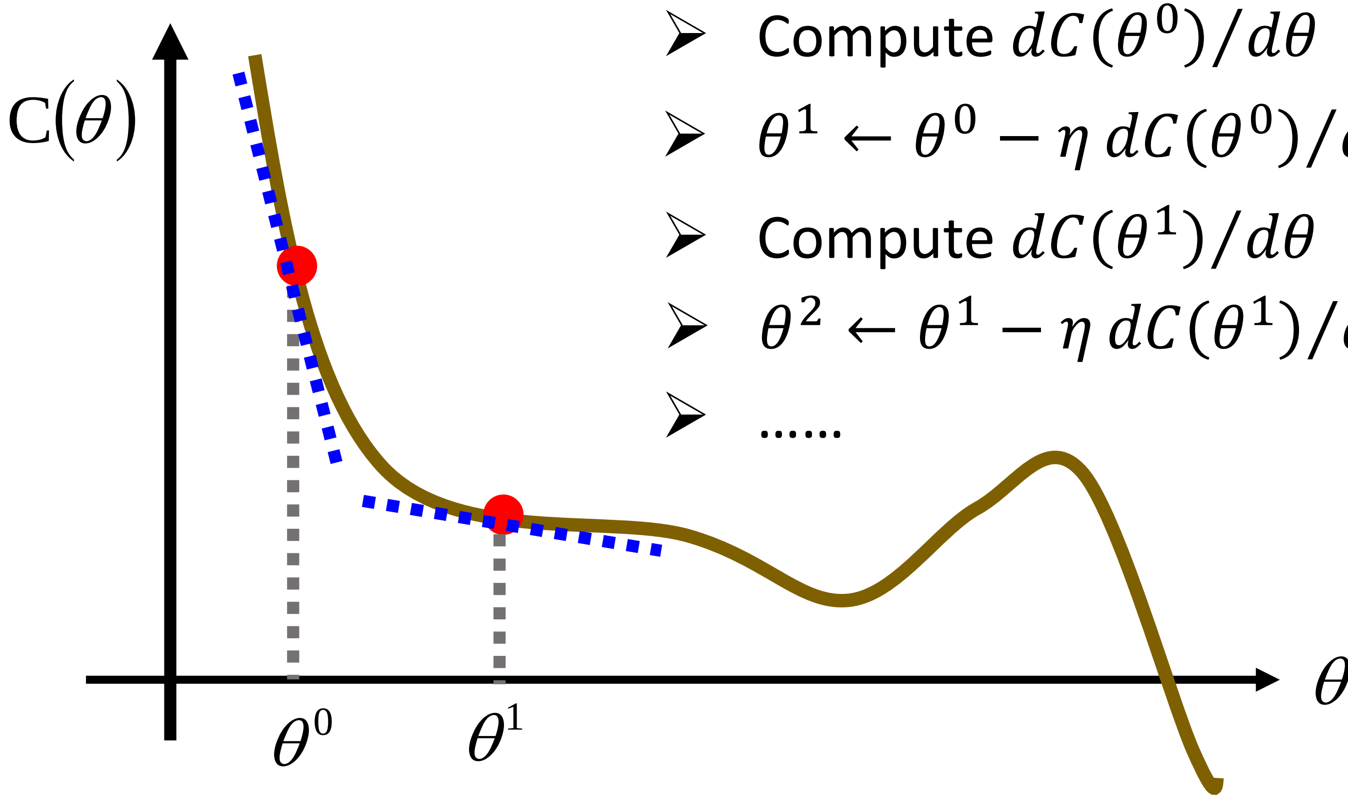


Gradient Descent – Idea

η is called ***"learning rate"***

- For simplification, first consider that θ has only one variable

  - Randomly start at θ^0
  - Compute $dC(\theta^0)/d\theta$
  - $\theta^1 \leftarrow \theta^0 - \eta\, dC(\theta^0)/d\theta$
  - Compute $dC(\theta^1)/d\theta$
  - $\theta^2 \leftarrow \theta^1 - \eta\, dC(\theta^1)/d\theta$
  - ……



Gradient Descent

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

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

➢ Compute the gradients of $C(\theta)$ at θ^0: $\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta_1^0)/\partial \theta_1 \\ \partial C(\theta_2^0)/\partial \theta_2 \end{bmatrix}$

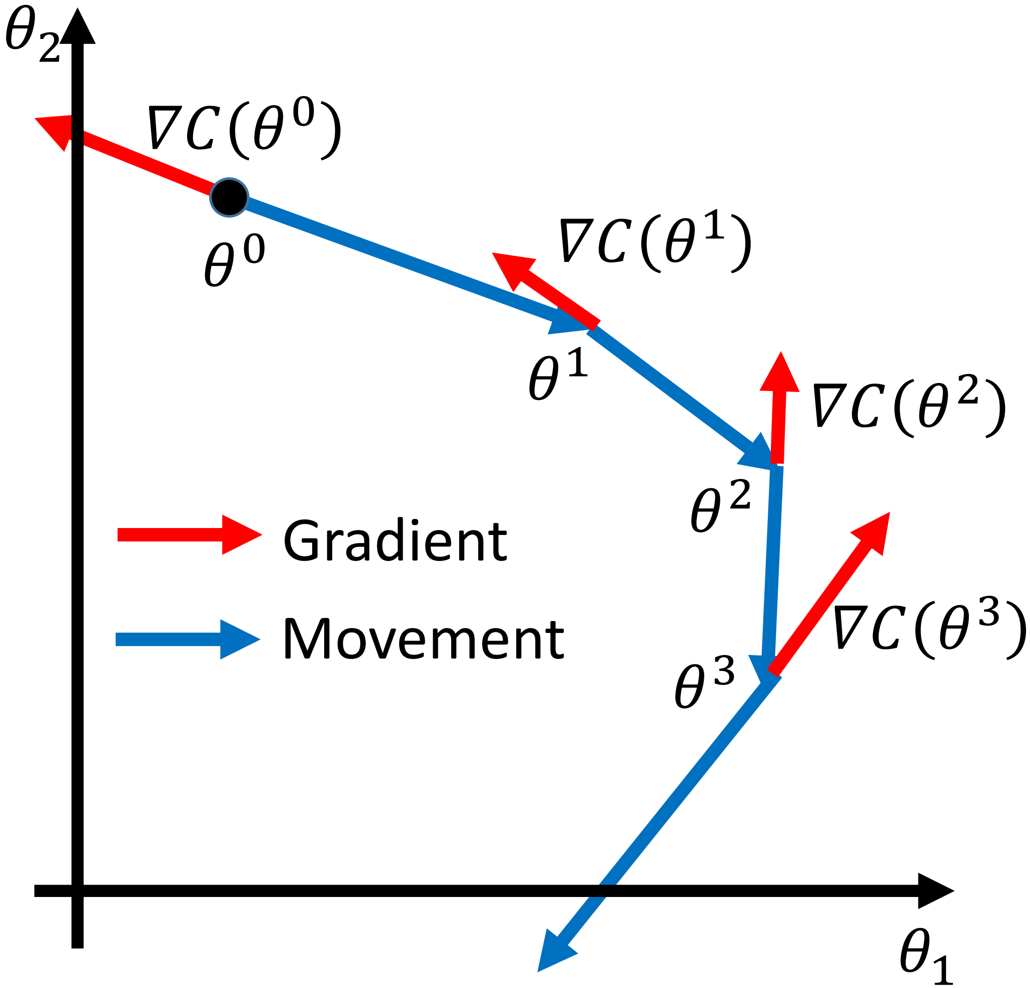
➢ Update parameters

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial C(\theta_1^0)/\partial \theta_1 \\ \partial C(\theta_2^0)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta \nabla C(\theta^0)$$

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

➢ ……

Gradient Descent



Start at position θ^0

Compute gradient at θ^0

Move to $\theta^1 = \theta^0 - \eta\nabla C(\theta^0)$

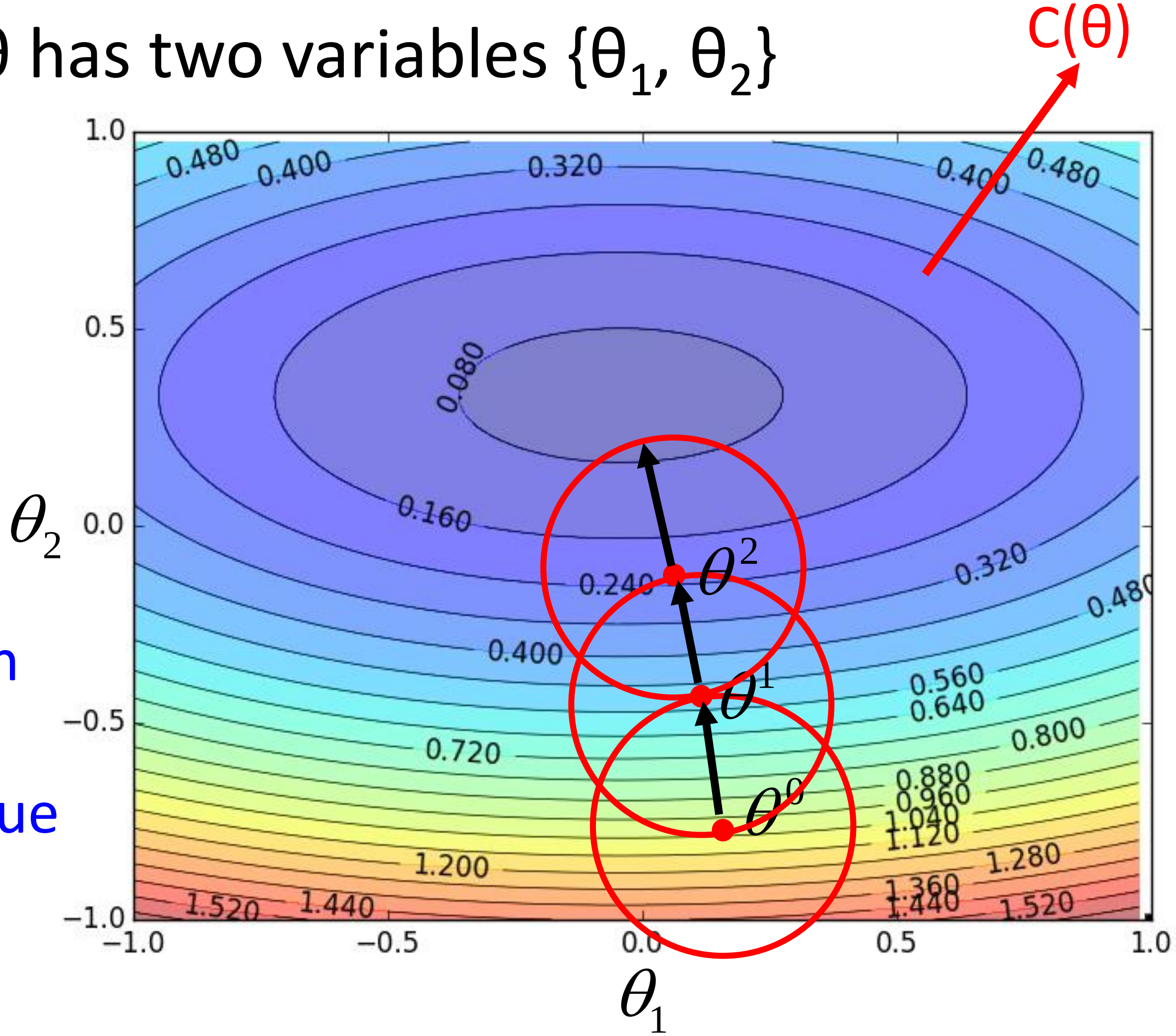
Compute gradient at θ^1

Move to $\theta^2 = \theta^1 - \eta\nabla C(\theta^1)$

⋮

Formal Derivation of Gradient Descent

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



Given a point, we can easily find the point with the smallest value nearby.

Gradient Descent for Neural Network

$$Compute\ \nabla C(\theta^0)$$
$$\theta^1 = \theta^0 - \eta \nabla C(\theta^0)$$
$$Compute\ \nabla C(\theta^1)$$
$$\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$$

Starting Parameters $\theta^0 \longrightarrow \theta^1 \longrightarrow \theta^2 \longrightarrow \cdots\cdots$

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

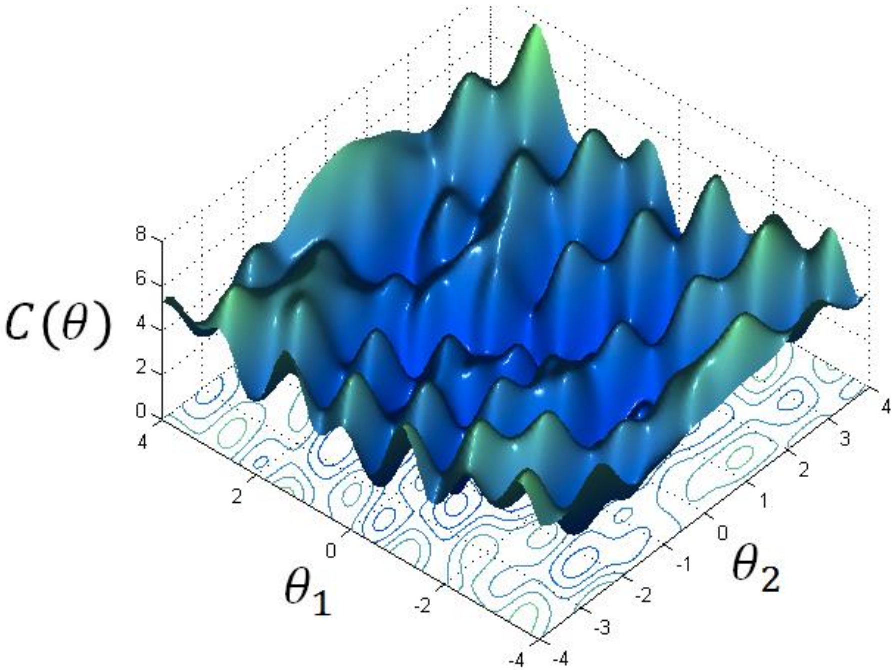
$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \\ \vdots \end{bmatrix}$$

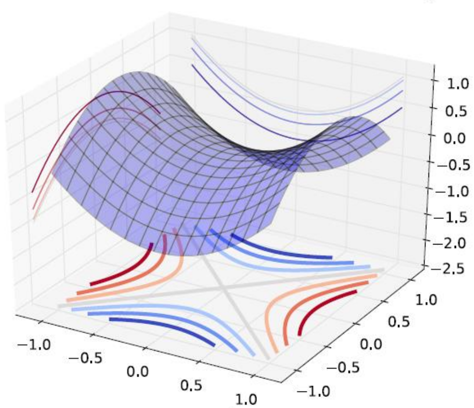
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \qquad b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

Millions of parameters ……

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

Stuck at local minima?





Saddle point

- Who is Afraid of Non-Convex Loss Functions?
- http://videolectures.net/eml07_lecun_wia/
- Deep Learning: Theoretical Motivations
- http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/

3. How to pick the "best" function?

Practical Issues for neural network

Practical Issues for neural network

- Parameter Initialization
- Learning Rate
- Stochastic gradient descent and Mini-batch
- Recipe for Learning

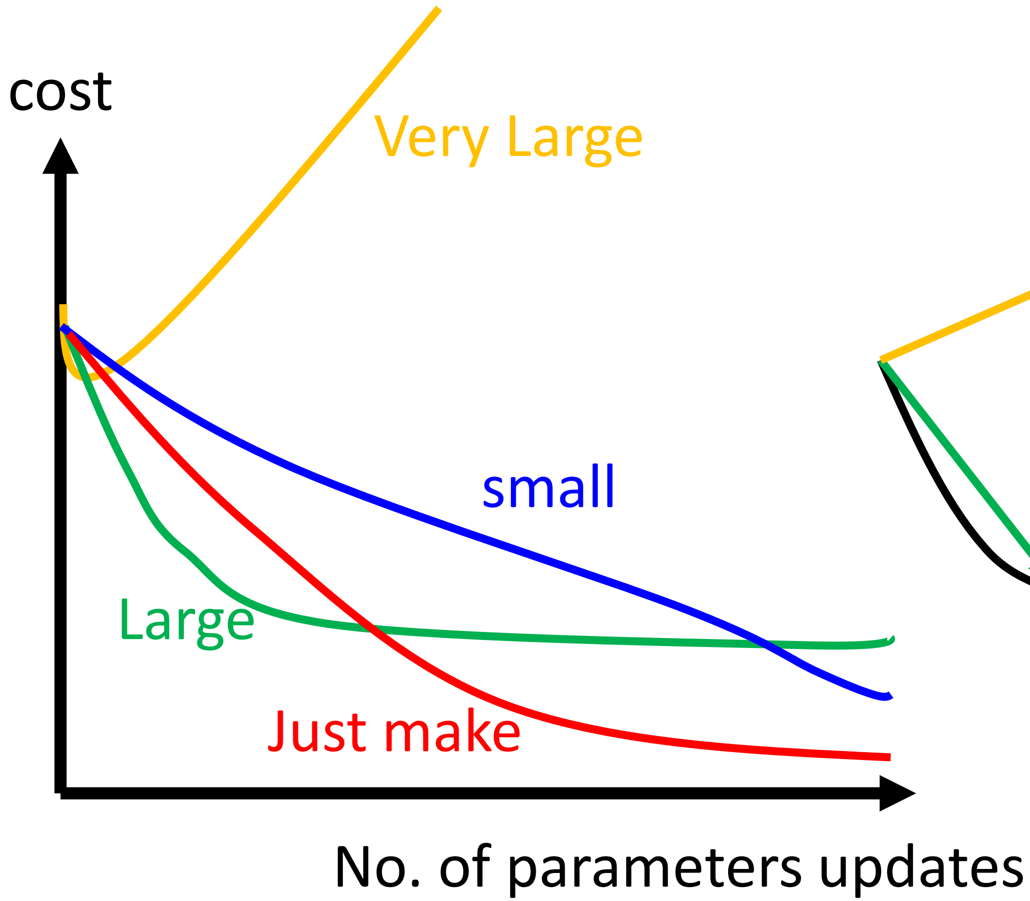
Parameter Initialization

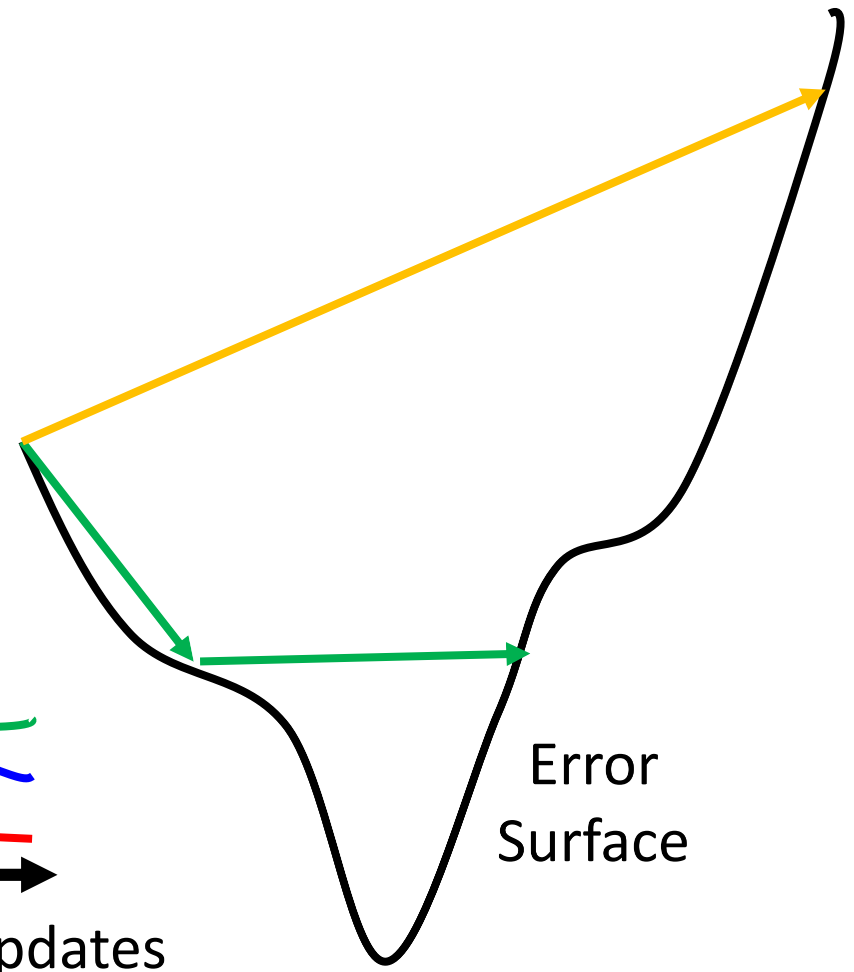
- For gradient Descent, we need to pick an initialization parameter θ^0.
- The initialization parameters have some influence to the training.
  - We will go back to this issue in the future.
- Suggestion today:
  - Do not set all the parameters θ^0 equal
  - Set the parameters in θ^0 randomly

Learning Rate

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

- Set the learning rate η carefully



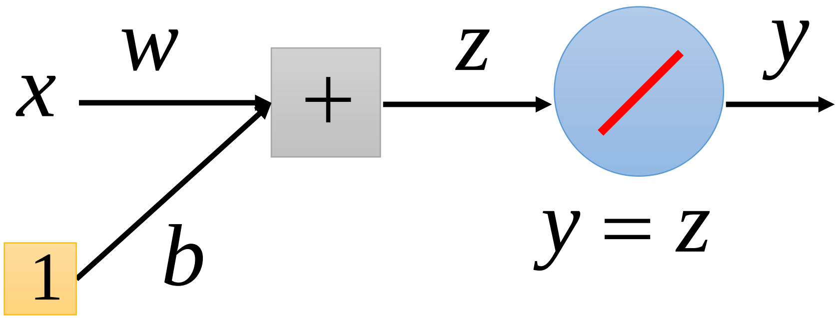


$$\theta^i = \theta^{i-1} - \eta \nabla C\left(\theta^{i-1}\right)$$

Learning Rate

- Set the learning rate η carefully

- ***<u>Toy Example</u>***



$$\theta^* = \begin{bmatrix} w = 1 \\ b = 0 \end{bmatrix}$$

Training Data (20 examples)

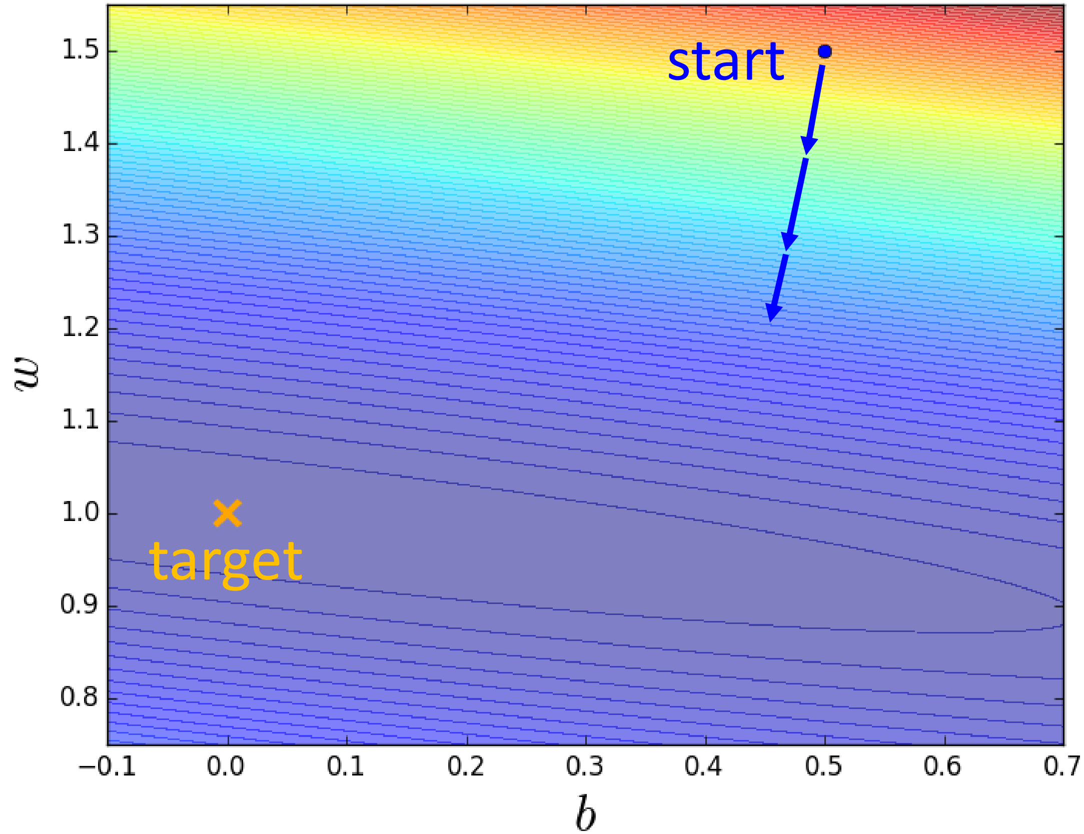
x = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5]
y = [0.1, 0.4, 0.9, 1.6, 2.2, 2.5, 2.8, 3.5, 3.9, 4.7, 5.1, 5.3, 6.3, 6.5, 6.7, 7.5, 8.1, 8.5, 8.9, 9.5]

Learning Rate

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

- ***Toy Example***

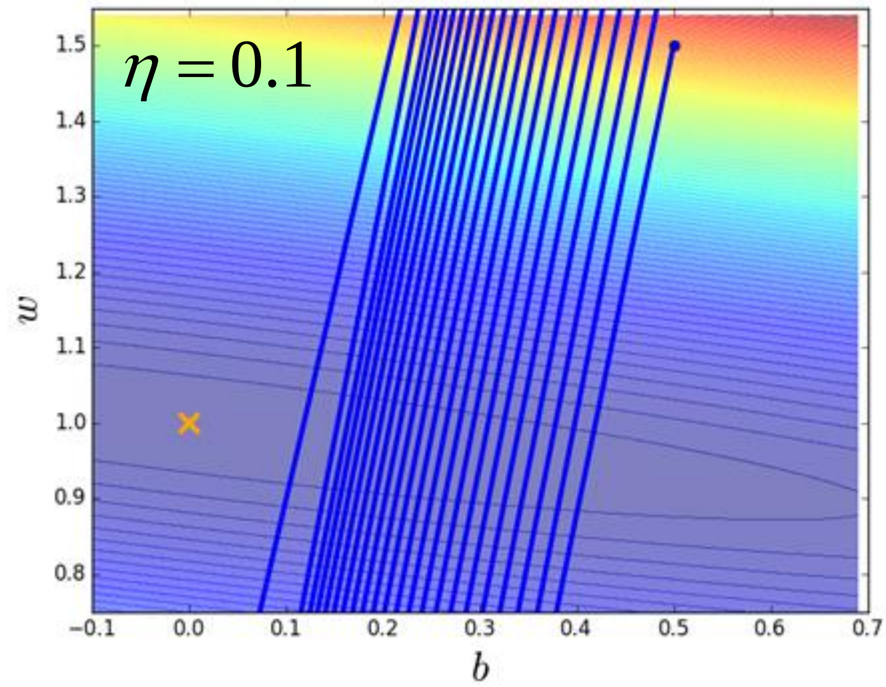
Error Surface: C(w,b)

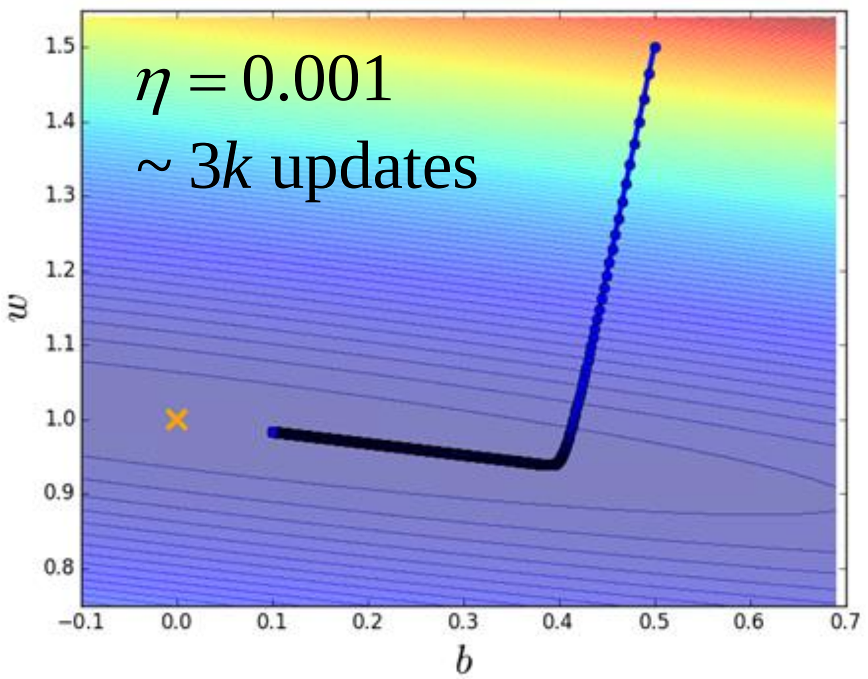


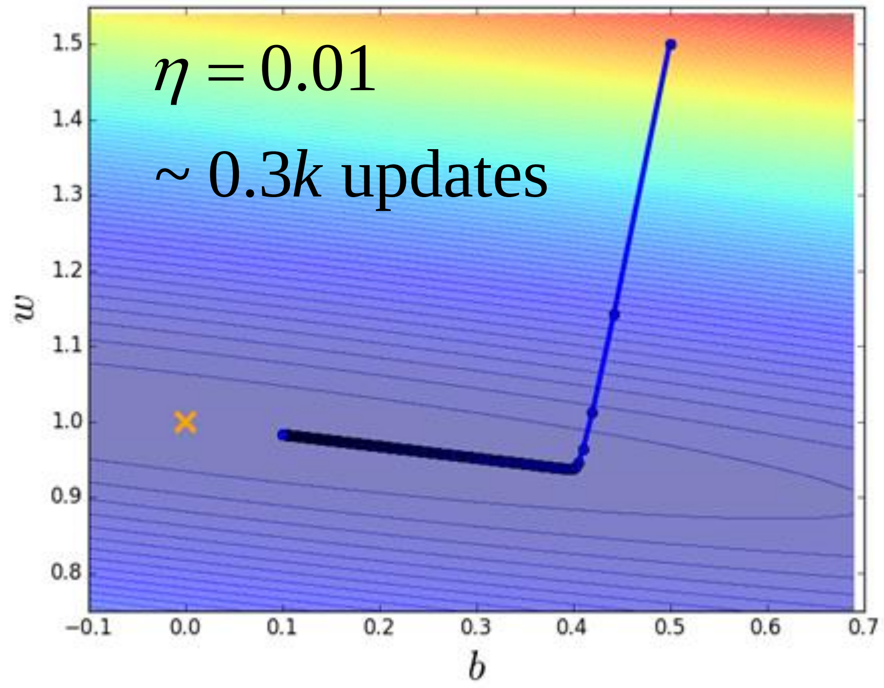
Learning Rate

- ***Toy Example***

  Different learning rate η







Stochastic Gradient Descent and Mini-batch

$$C(\theta)=\frac{1}{R}\sum_r \left\| f\left(x^r;\theta\right)-\hat{y}^r \right\|$$
$$=\frac{1}{R}\sum_r C^r(\theta)$$

◆ ***Gradient Descent***

$$\theta^i=\theta^{i-1}-\eta\nabla C\left(\theta^{i-1}\right) \qquad \nabla C\left(\theta^{i-1}\right)=\frac{1}{R}\sum_r \nabla C^r\left(\theta^{i-1}\right)$$

◆ ***Stochastic Gradient Descent*** Faster! Better!

Pick an example x^r

$$\theta^i=\theta^{i-1}-\eta\nabla C^r\left(\theta^{i-1}\right)$$

If all example x^r have equal probabilities to be picked

$$E\left[\nabla C^r\left(\theta^{i-1}\right)\right]=\frac{1}{R}\sum_r \nabla C^r\left(\theta^{i-1}\right)$$

Stochastic Gradient Descent and Mini-batch

What is epoch?

Training Data: $\left\{\left(x^1,\hat{y}^1\right),\left(x^2,\hat{y}^2\right),\cdots\left(x^r,\hat{y}^r\right),\cdots\left(x^R,\hat{y}^R\right)\right\}$

When using stochastic gradient descent

Starting at θ_0

pick x^1 $\quad \theta^1=\theta^0-\eta\nabla C^1\left(\theta^0\right)$

pick x^2 $\quad \theta^2=\theta^1-\eta\nabla C^2\left(\theta^1\right)$

⋮ ⋮

pick x^r $\quad \theta^r=\theta^{r-1}-\eta\nabla C^r\left(\theta^{r-1}\right)$

⋮ ⋮

pick x^R $\quad \theta^R=\theta^{R-1}-\eta\nabla C^R\left(\theta^{R-1}\right)$

Seen all the examples once

One epoch

---

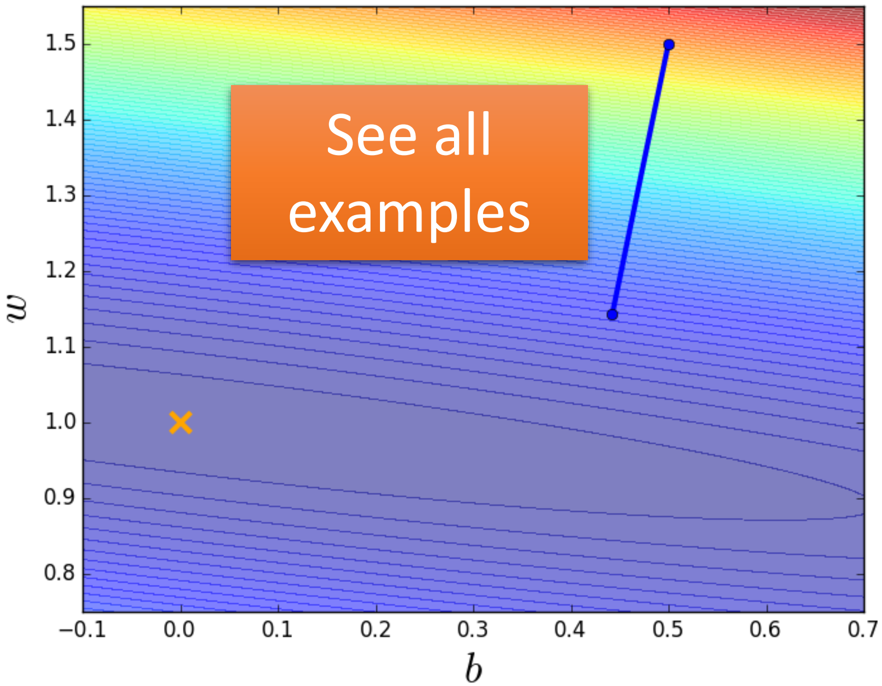
pick x^1 $\quad \theta^{R+1}=\theta^R-\eta\nabla C^1\left(\theta^R\right)$

Stochastic Gradient Descent and Mini-batch

- ***Toy Example***

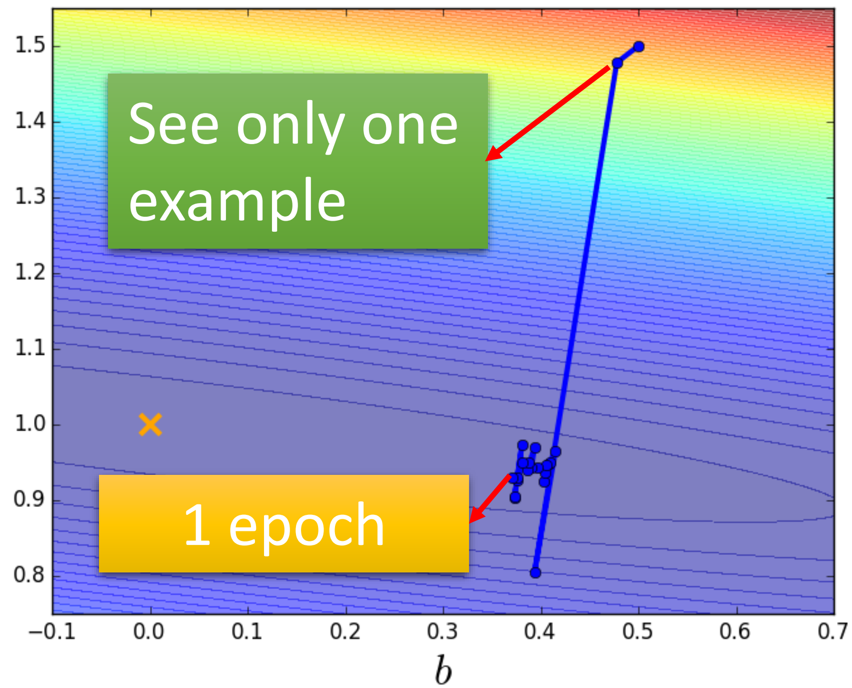
Gradient Descent

Update after seeing all examples



Stochastic Gradient Descent

If there are 20 examples, update 20 times in one epoch.



Stochastic Gradient Descent and Mini-batch

◆ ***Gradient Descent***

$$\theta^i = \theta^{i-1} - \eta \nabla C\left(\theta^{i-1}\right) \qquad \nabla C\left(\theta^{i-1}\right) = \frac{1}{R}\sum_r \nabla C^r\left(\theta^{i-1}\right)$$

◆ ***Stochastic Gradient Descent***

Pick an example x_r

$$\theta^i = \theta^{i-1} - \eta \nabla C^r\left(\theta^{i-1}\right)$$

◆ ***Mini Batch Gradient Descent***

Pick B examples as a batch b

B is batch size

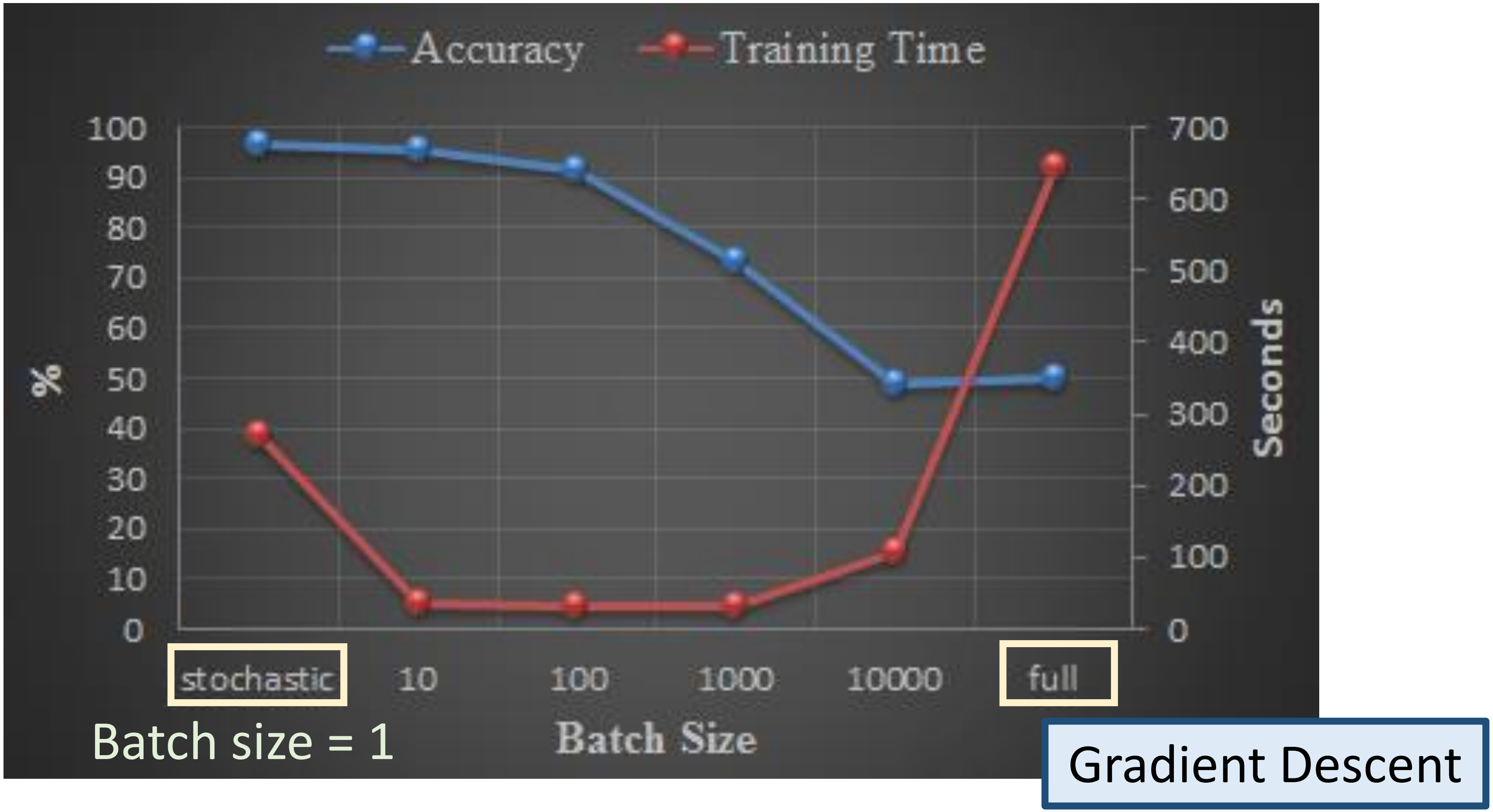
Shuffle your data

$$\theta^i = \theta^{i-1} - \eta \frac{1}{B}\sum_{x_r \in b} \nabla C^r\left(\theta^{i-1}\right)$$

Average the gradient of the examples in the batch b

Stochastic Gradient Descent and Mini-batch

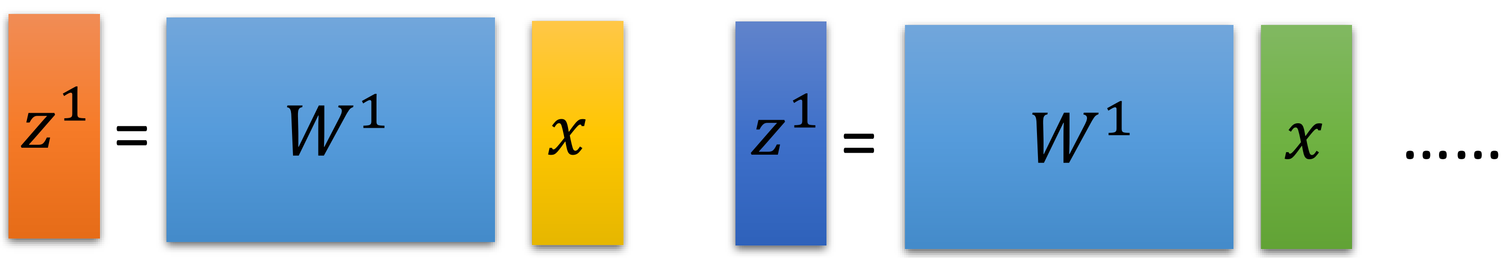
- ***<u>Handwriting Digit Classification</u>***



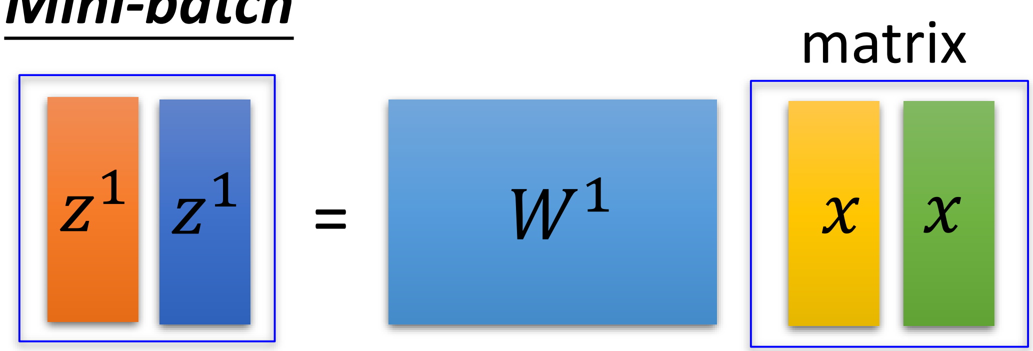
Stochastic Gradient Descent and Mini-batch

- Why mini-batch is faster than stochastic gradient descent?

<u>Stochastic Gradient Descent</u>

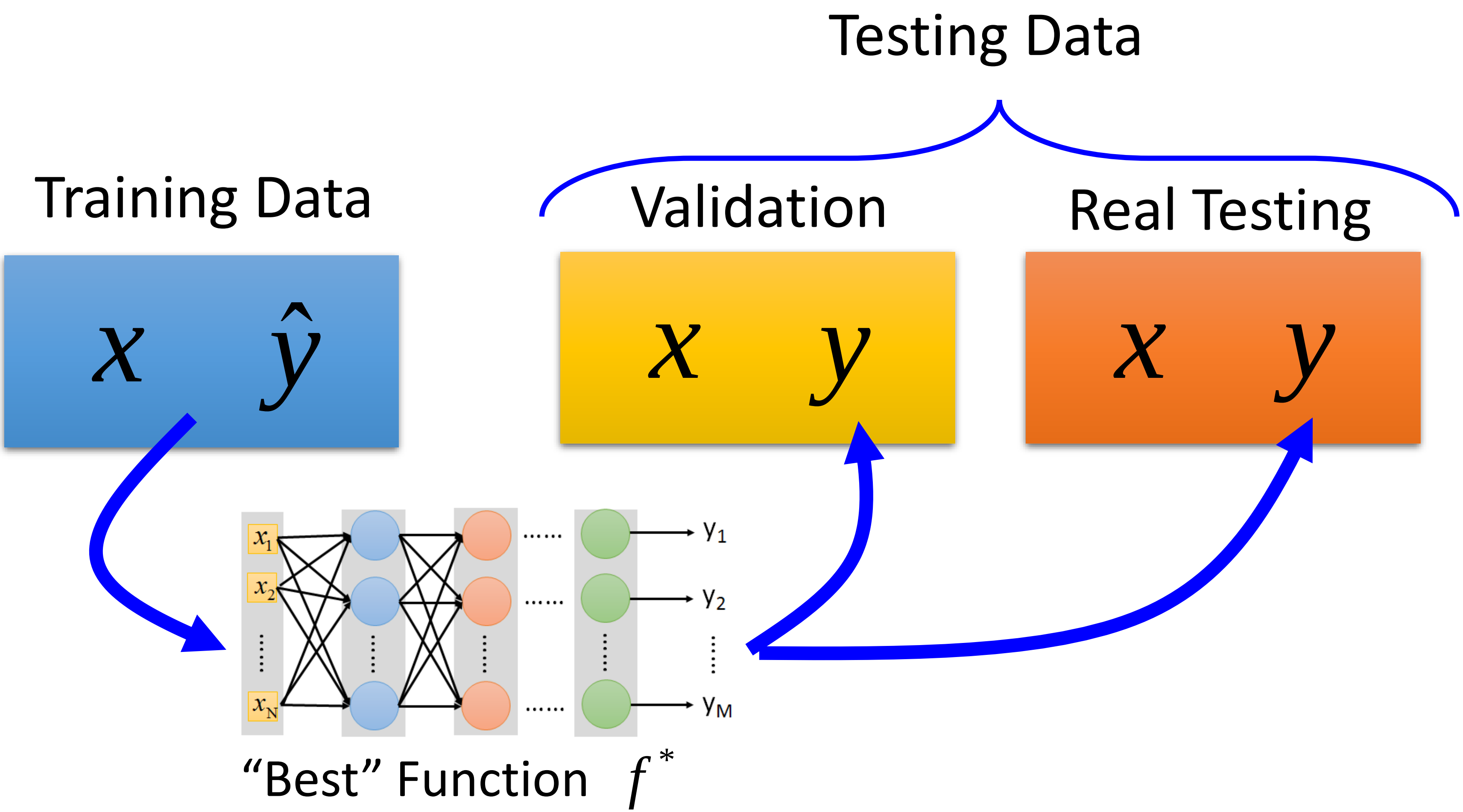


<u>Mini-batch</u>



Practically, which one is faster?

Recipe for Learning



Recipe for Learning - Overfitting

- You pick a "best" parameter set θ^*

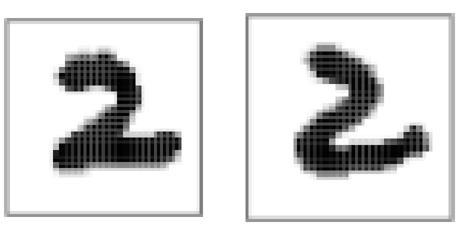
Training Data: $\left\{\ldots\left(x^r, \hat{y}^r\right)\ldots\right\} \Rightarrow \forall r: f\left(x^r;\theta^*\right) = \hat{y}^r$

However,

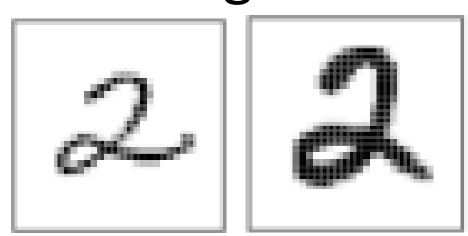
Testing Data: $\left\{\ldots x^u \ldots\right\} \Rightarrow f\left(x^u;\theta^*\right) \neq \hat{y}^u$

Training data and testing data have different distribution.

Training Data:



Testing Data:



Recipe for Learning - Overfitting

- Panacea: Have more training data

- We will go back to this issue in the future.

Concluding Remarks

1. What is the model (function hypothesis set)?

Neural Network

2. What is the "best" function?

Cost Function

3. How to pick the "best" function?

Gradient Descent

- Parameter Initialization
- Learning Rate
- Stochastic gradient descent, Mini-batch
- Recipe for Learning