Lecture 6: Introduction to Deep Learning

Hong-Han Shuai

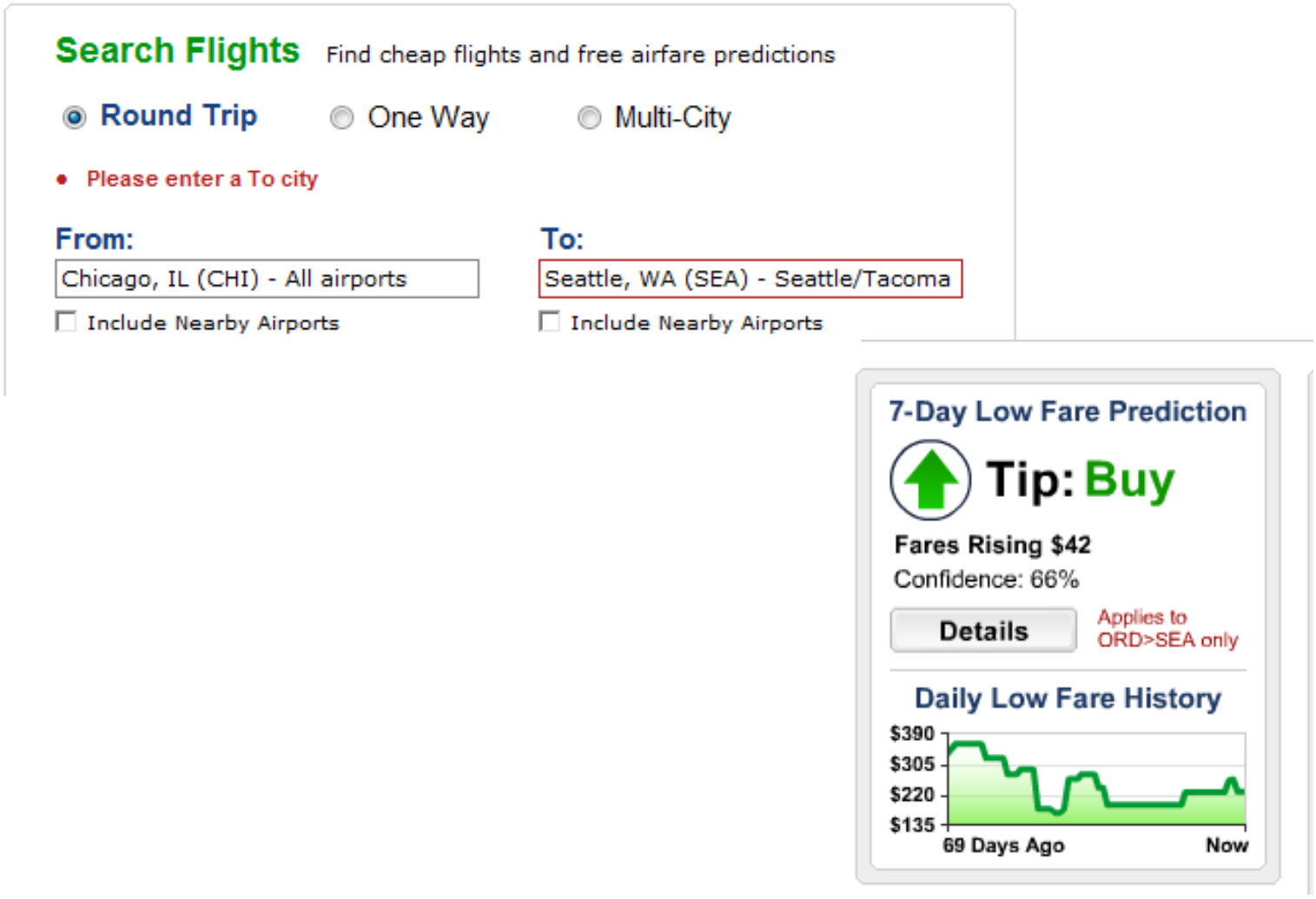
National Chiao Tung University

2018 Fall

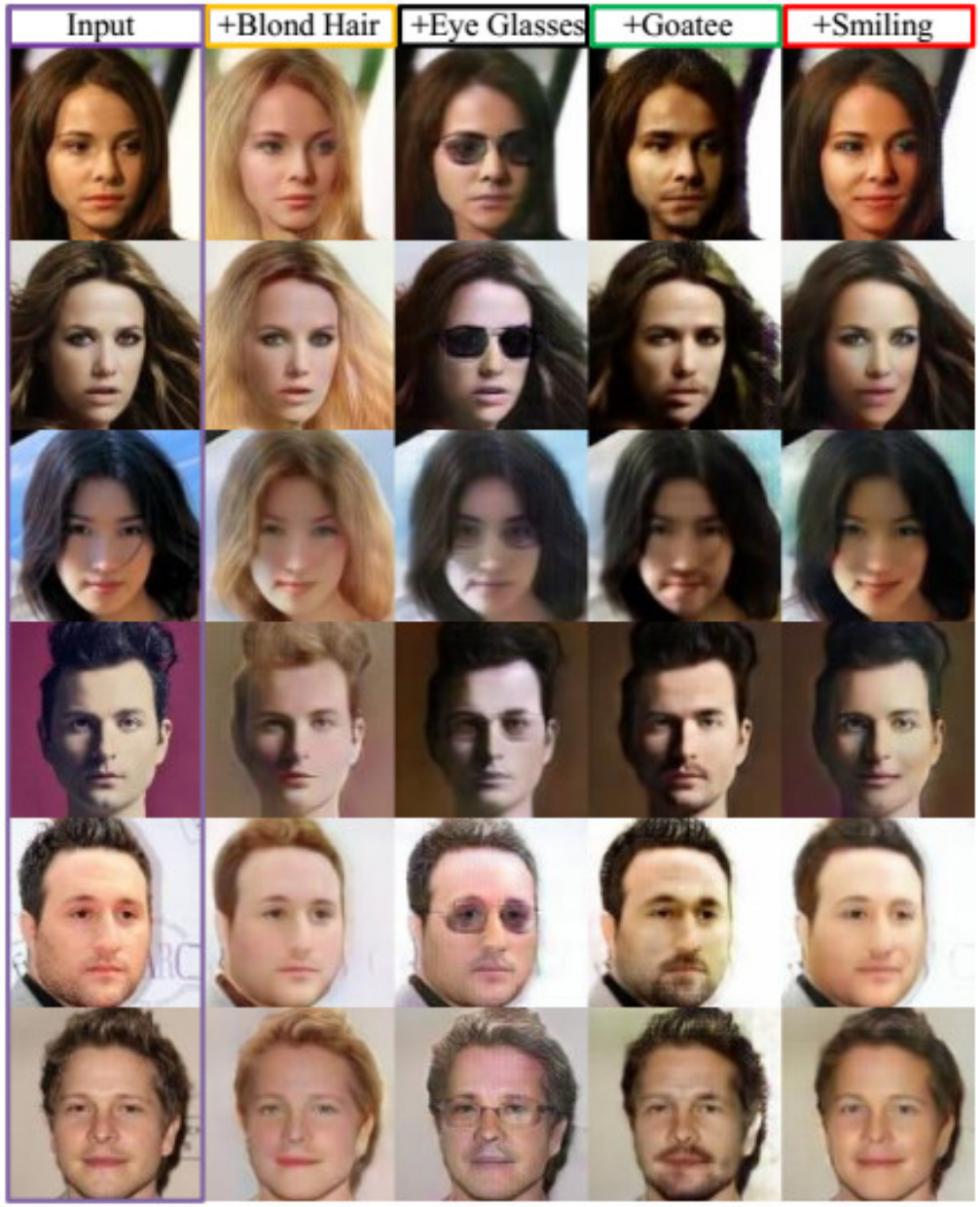
Thanks to the slides of Prof. P. Domingos from Washington University, Prof. H.-T. Lin and Prof. Lee Hung-Yi Lee from NTU.

COURSE MAP: Deep Learning

- Multi-Layer Perceptron (General)
- CNN and why it works (Structural)
- RNN models (Sequential), Attention models
- Social Network Analysis
  - Embedding
  - Group Query
  - Recommendation System

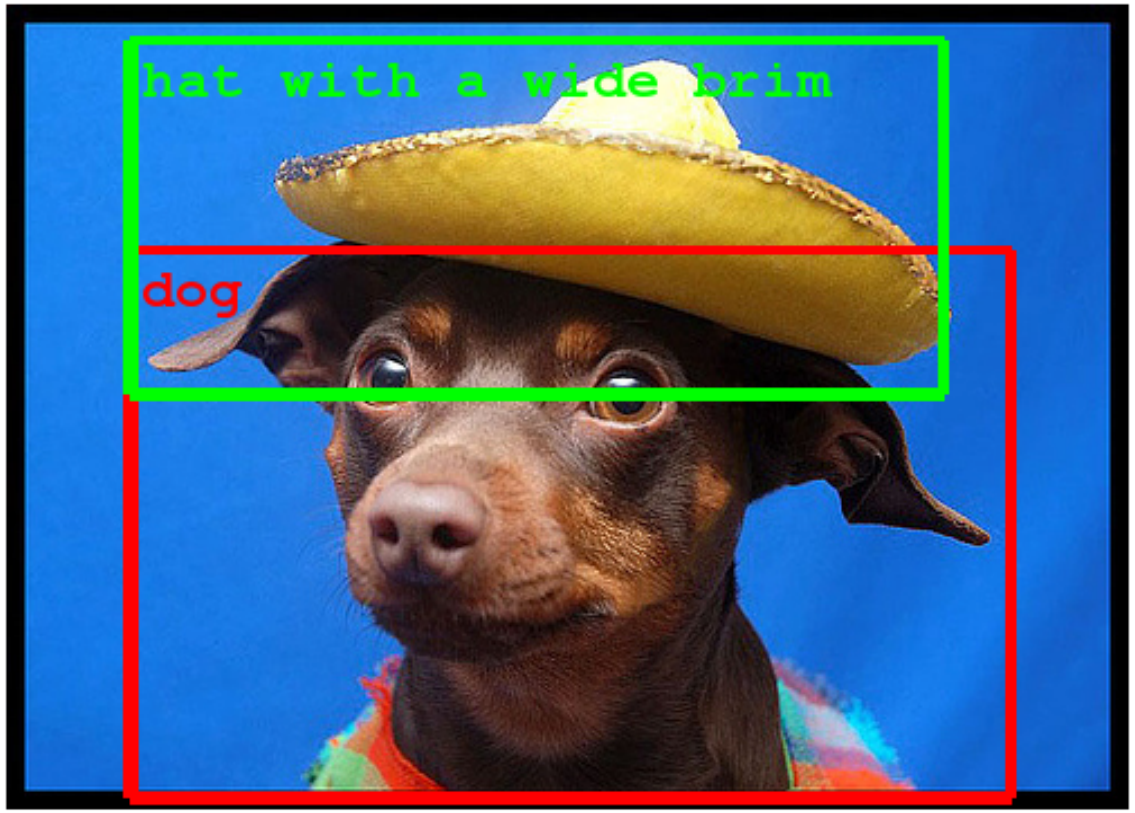

Search Flights
Find cheap flights and free airfare predictions
Round Trip
One Way
Multi-City
Please enter a To city
From:
Chicago, IL (CHI) - All airports
Include Nearby Airports
To:
Seattle, WA (SEA) - Seattle/Tacoma
Include Nearby Airports
7-Day Low Fare Prediction
Tip: Buy
Fares Rising $42
Confidence: 66%
Details
Applies to ORD>SEA only
Daily Low Fare History
$390
$305
$220
$135
69 Days Ago
Now

Applications



Applications

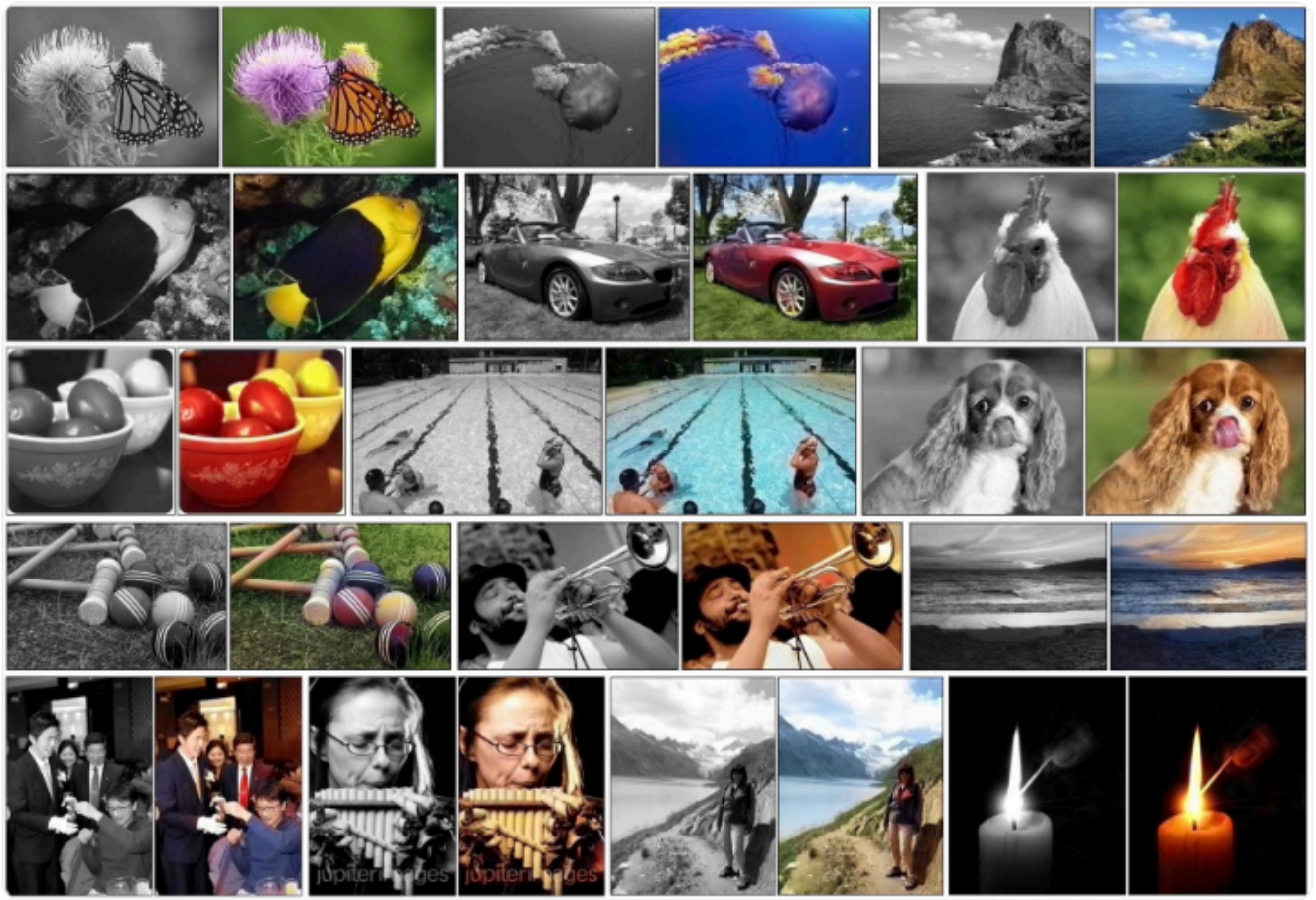
- Object Classification and Detection in Photographs



Source:http://machinelearningmastery.com/inspirational-applications-deep-learning

Applications

- Automatic Colorization of Black and White Images



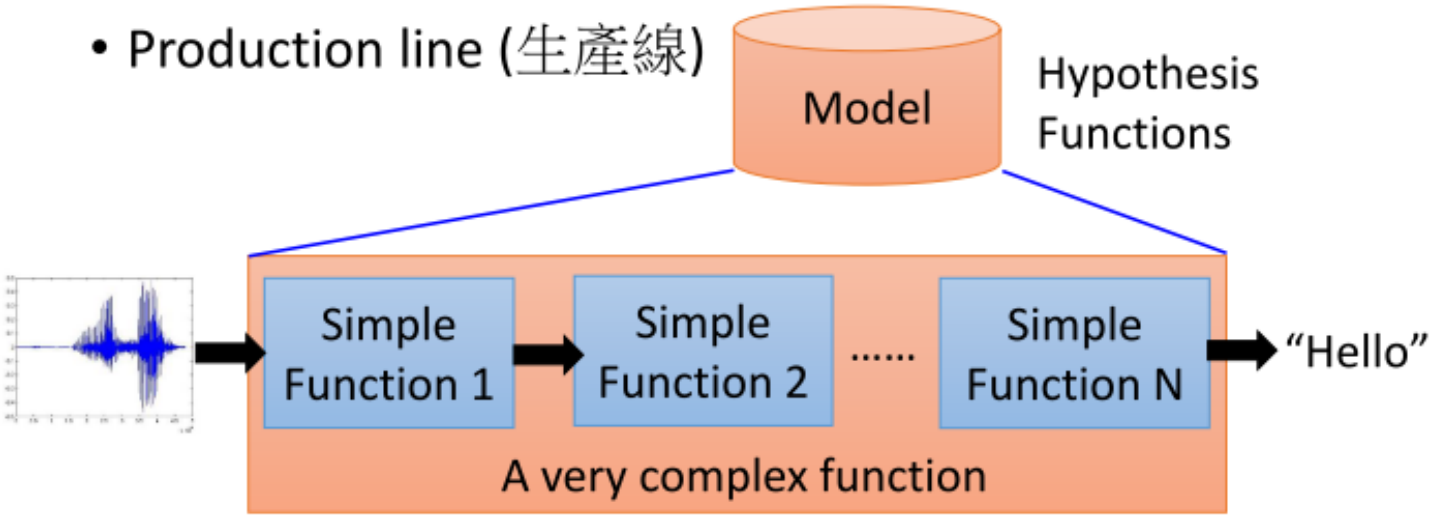
Source:http://machinelearningmastery.com/inspirational-applications-deep-learning/

Applications

- Automatic Speech Recognition
- Playing Go

What is Deep Learning?

- Production line (生產線)

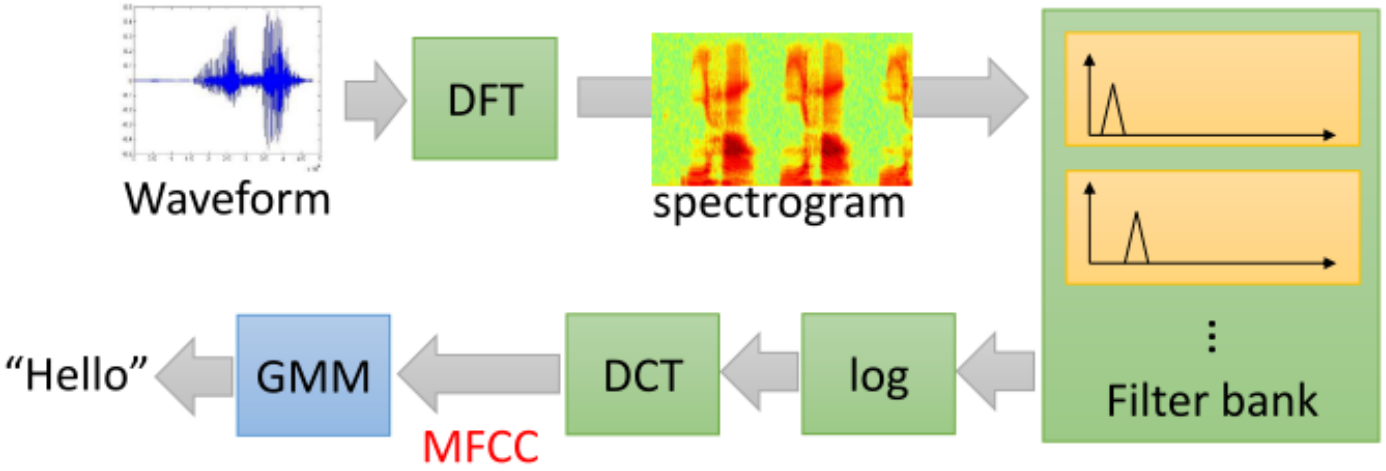


End-to-end training:

What each function should do is learned automatically

Deep v.s. Shallow
- Speech Recognition

- Shallow Approach



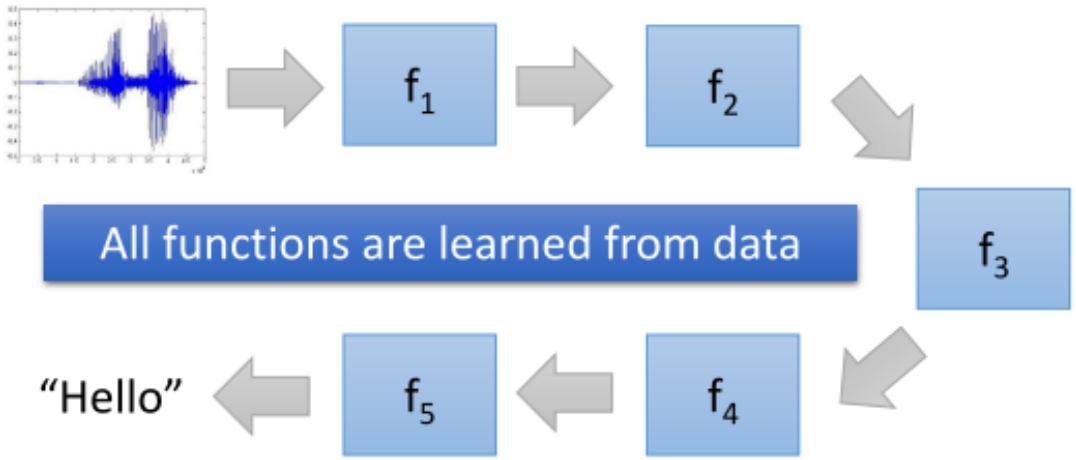
Each box is a simple function in the production line:

:hand-crafted :learned from data

Deep v.s. Shallow - Speech Recognition

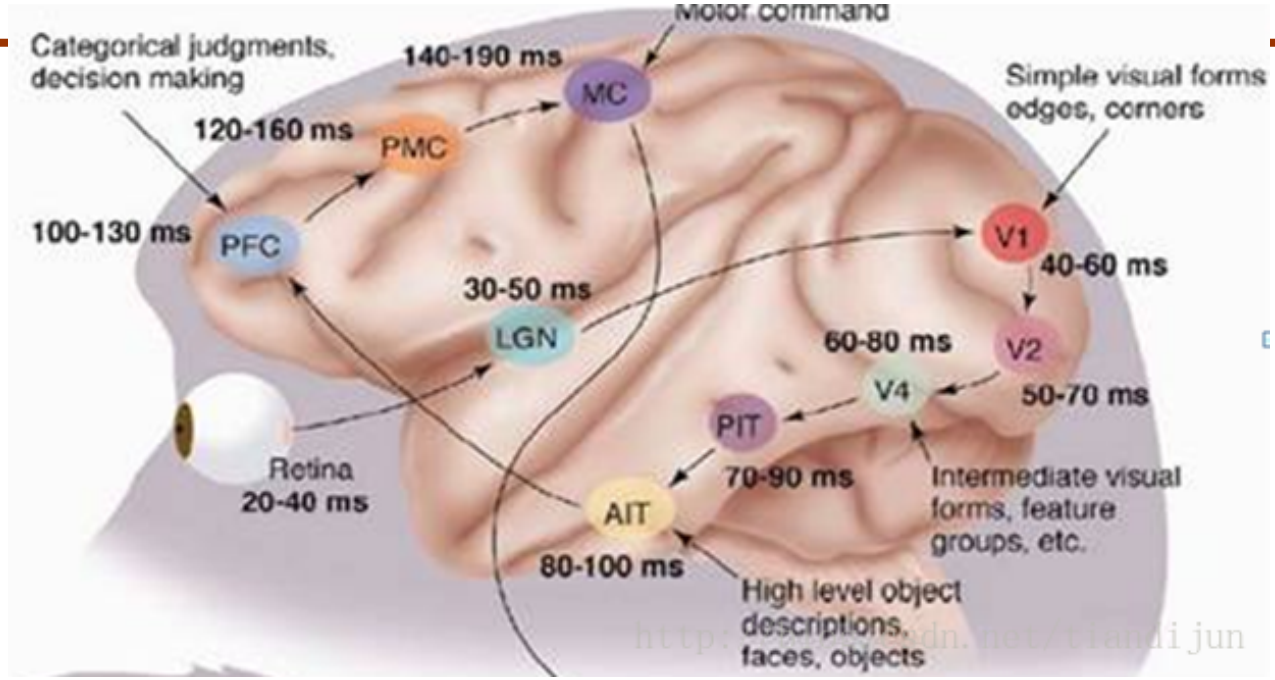
- Deep Learning

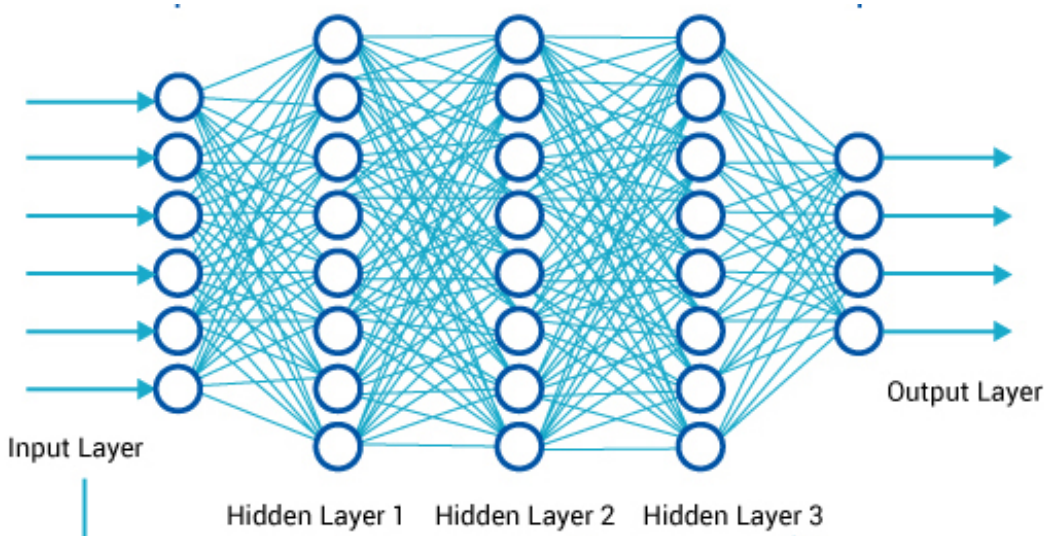
"Bye bye, MFCC"
- Deng Li in Interspeech 2014



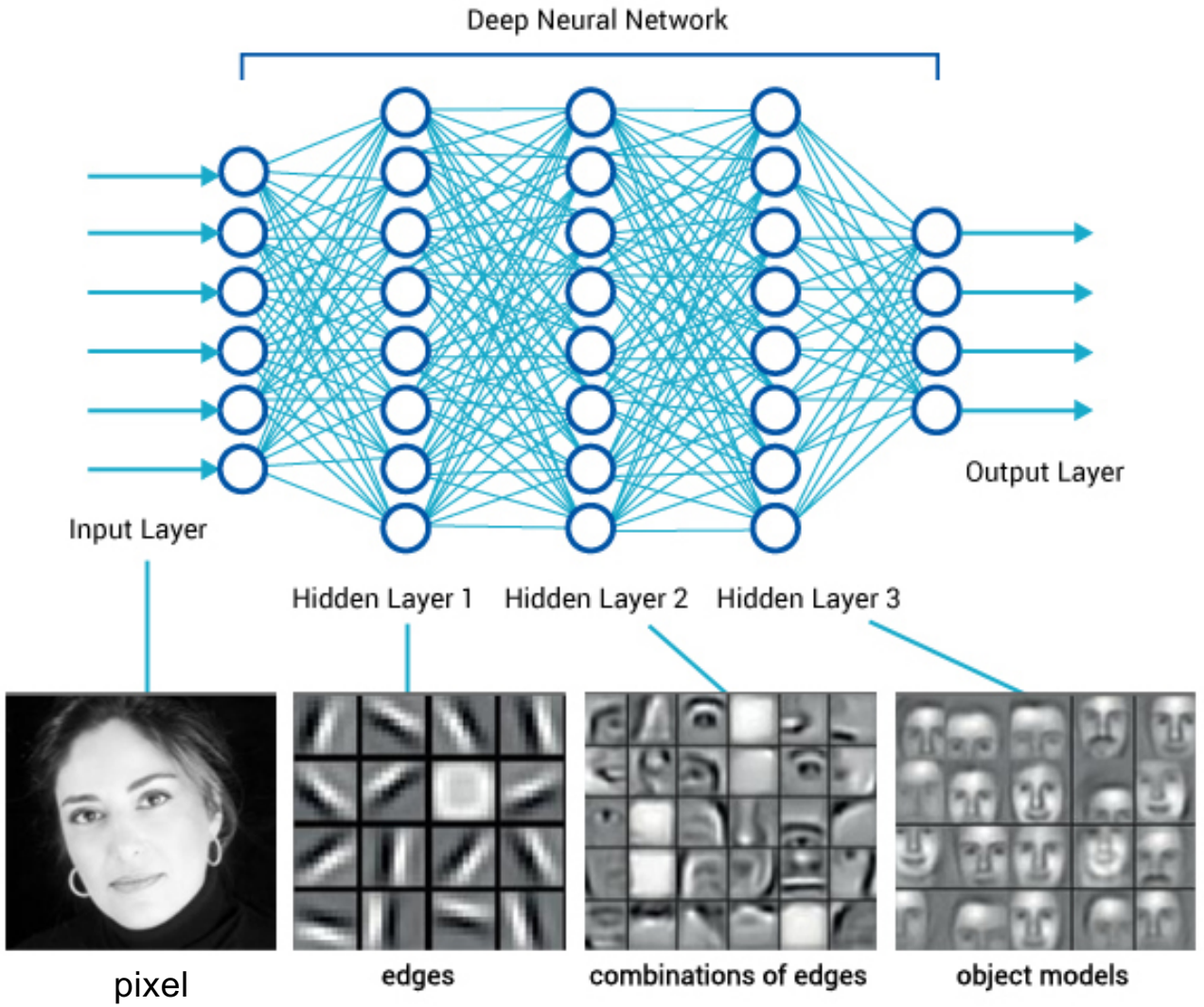
Less engineering labor, but machine learns more

Idea from human brain



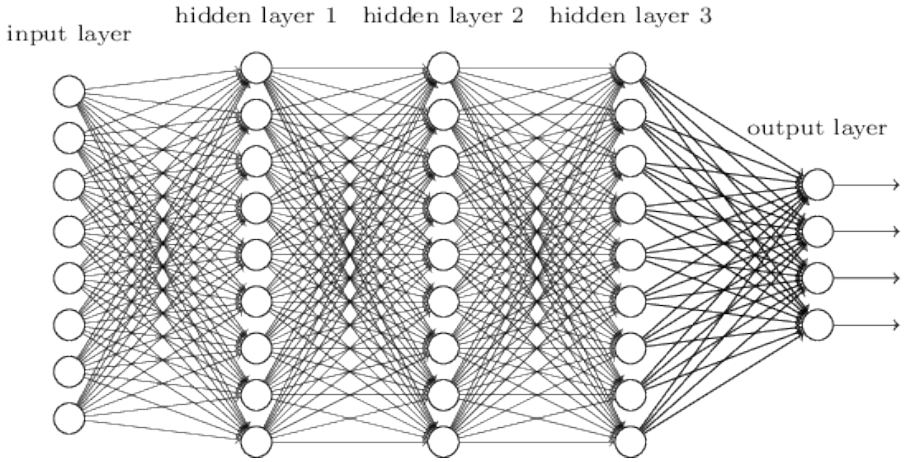


Artificial Neuron Network(ANN)

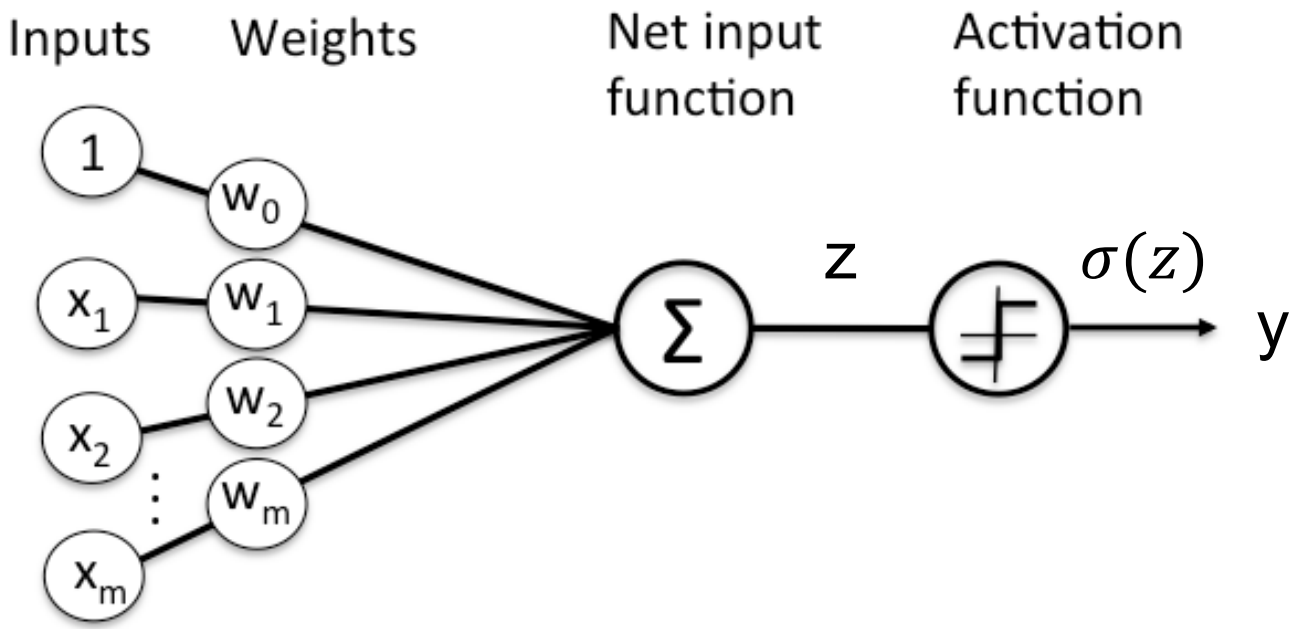


Source:http://www.amax.com/blog/?p=804

Neuron Network



Neuron

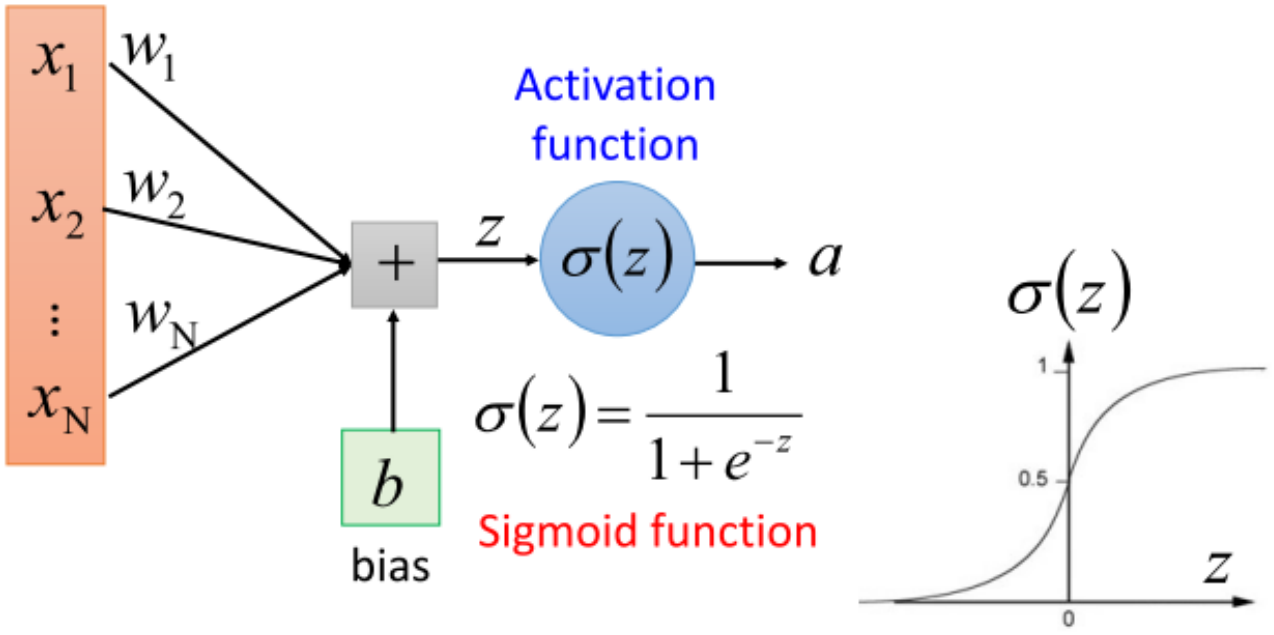


$z = 1^{*}w_0 + x_1{}^{*}w_1 + x_2{}^{*}w_2 + \ldots + x_m{}^{*}w_m$

$y = \sigma(z)$

A Neuron for Machine

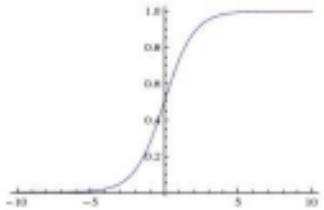
Each neuron is a very simple function



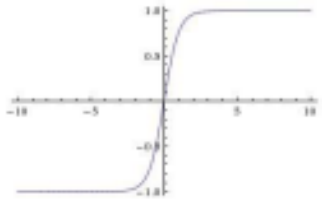
Activation Functions

Sigmoid

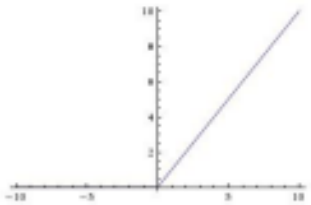
$$\sigma(x) = 1/(1+e^{-x})$$



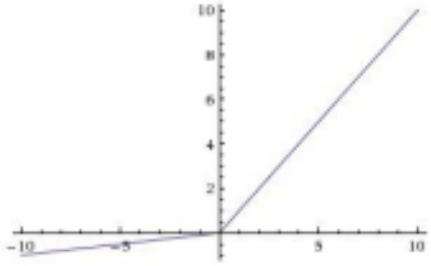
tanh tanh(x)



ReLU max(0,x)



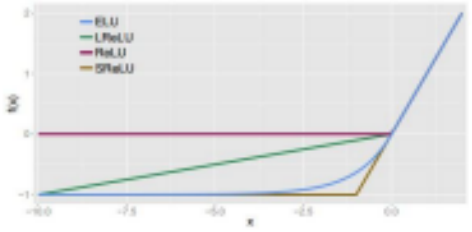
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



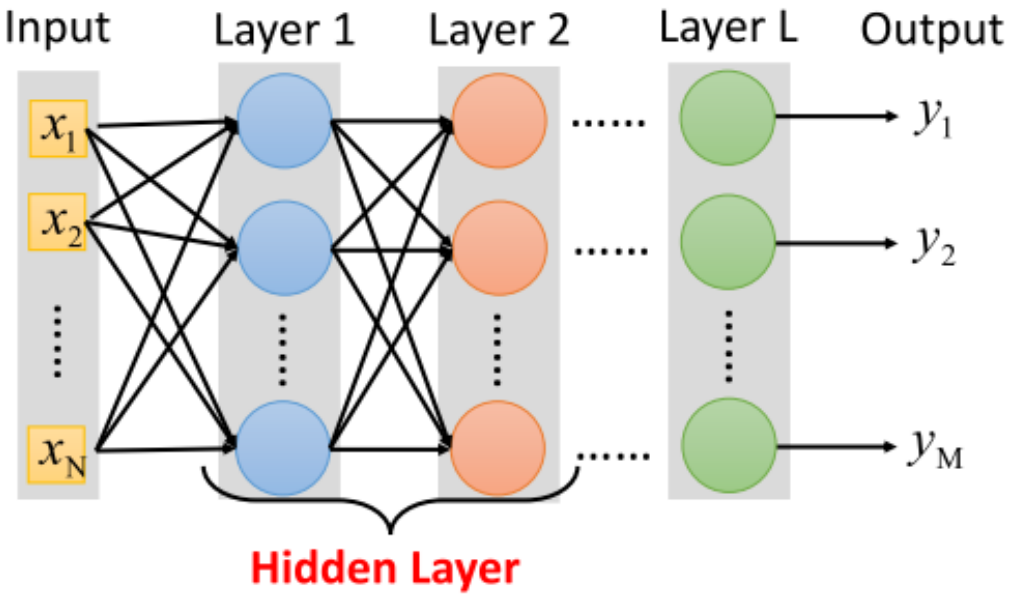
Deep Learning

A neural network is a complex function:

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

- Cascading the neurons to form a neural network.

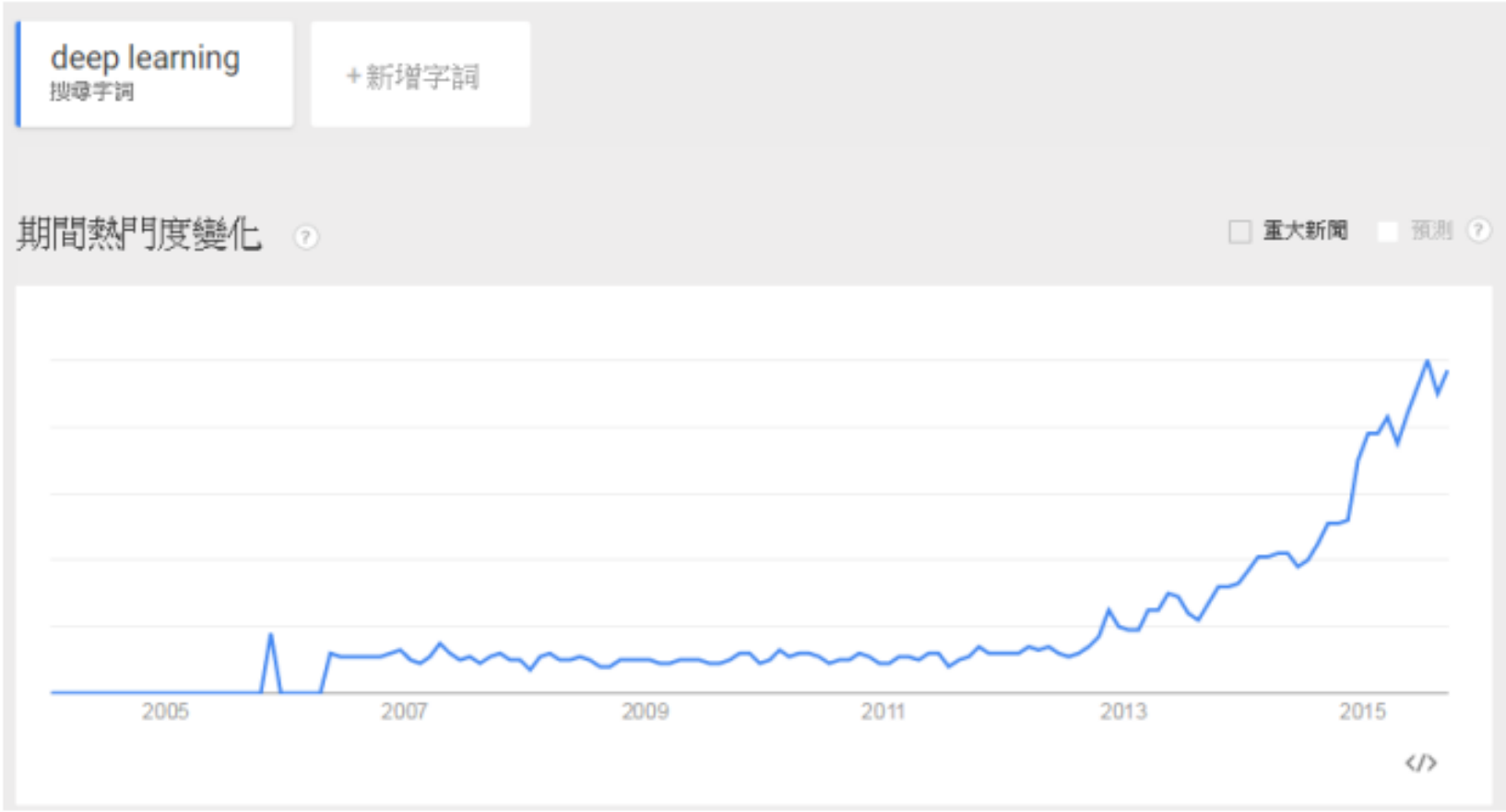
  Each layer is a simple function in the production line.



Ups and downs of deep learning

- 1958: Perceptron (Linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-Layer perceptron
- 1986: Backpropagation
- 1989: 1 hidden layer is "good enough",why deep?
- 2006: RBM initialization (breakthrough)
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition

Become very Popular



Why Deep Learning?

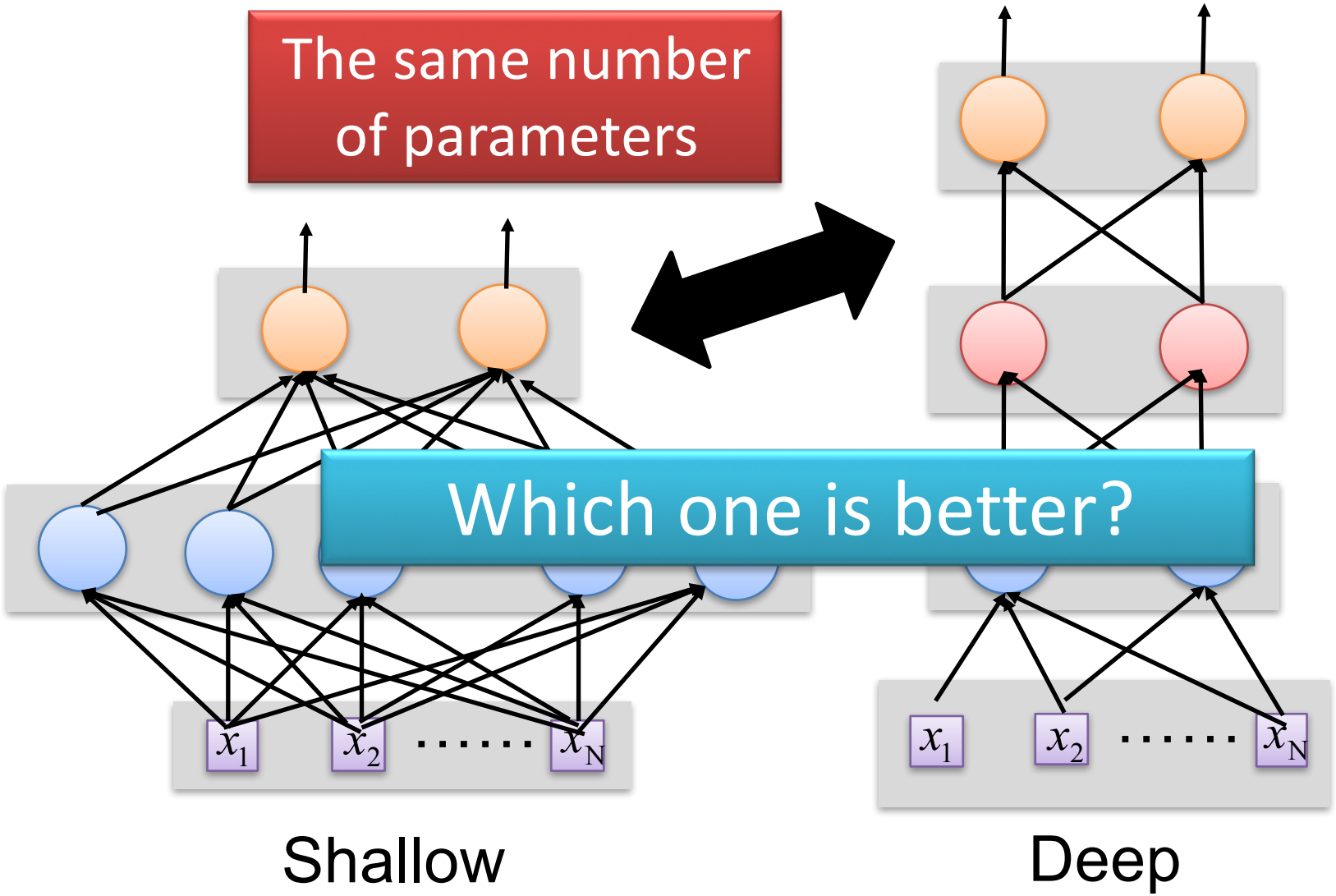
Deeper is Better?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

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

Fat + Short v.s. Thin + Tall



Fat + Short v.s. Thin + Tall

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

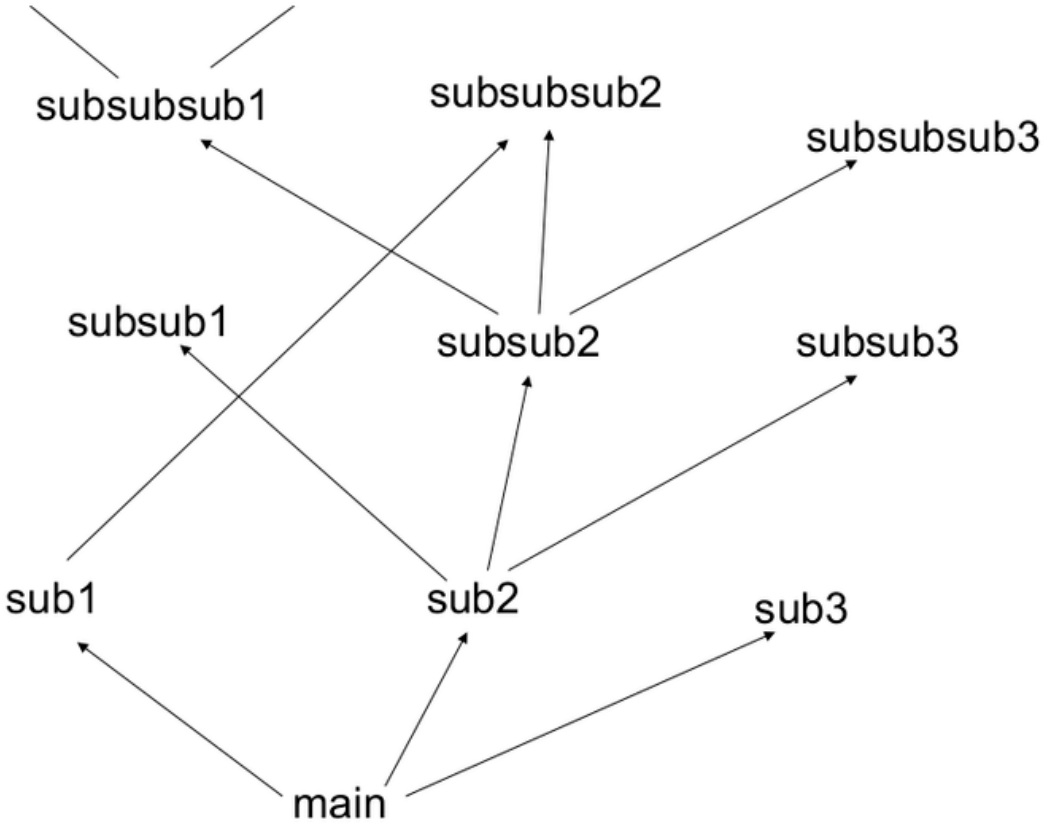
Why?

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

Modularization

- Deep → Modularization

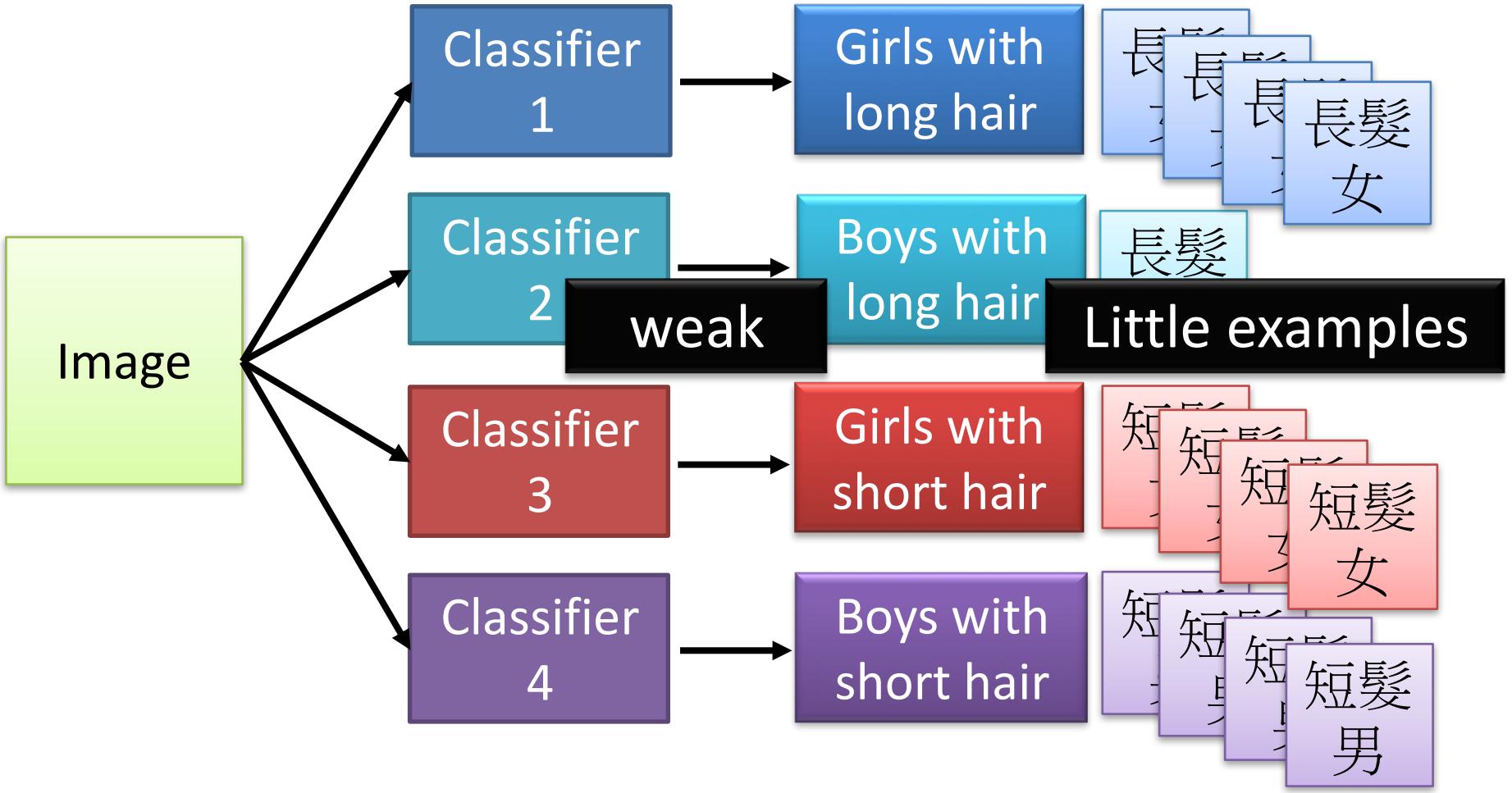
Don’t put everything in your main function.



http://rinuboney.github.io/2015/10/18/theoretical-motivations-deep-learning.html

Modularization

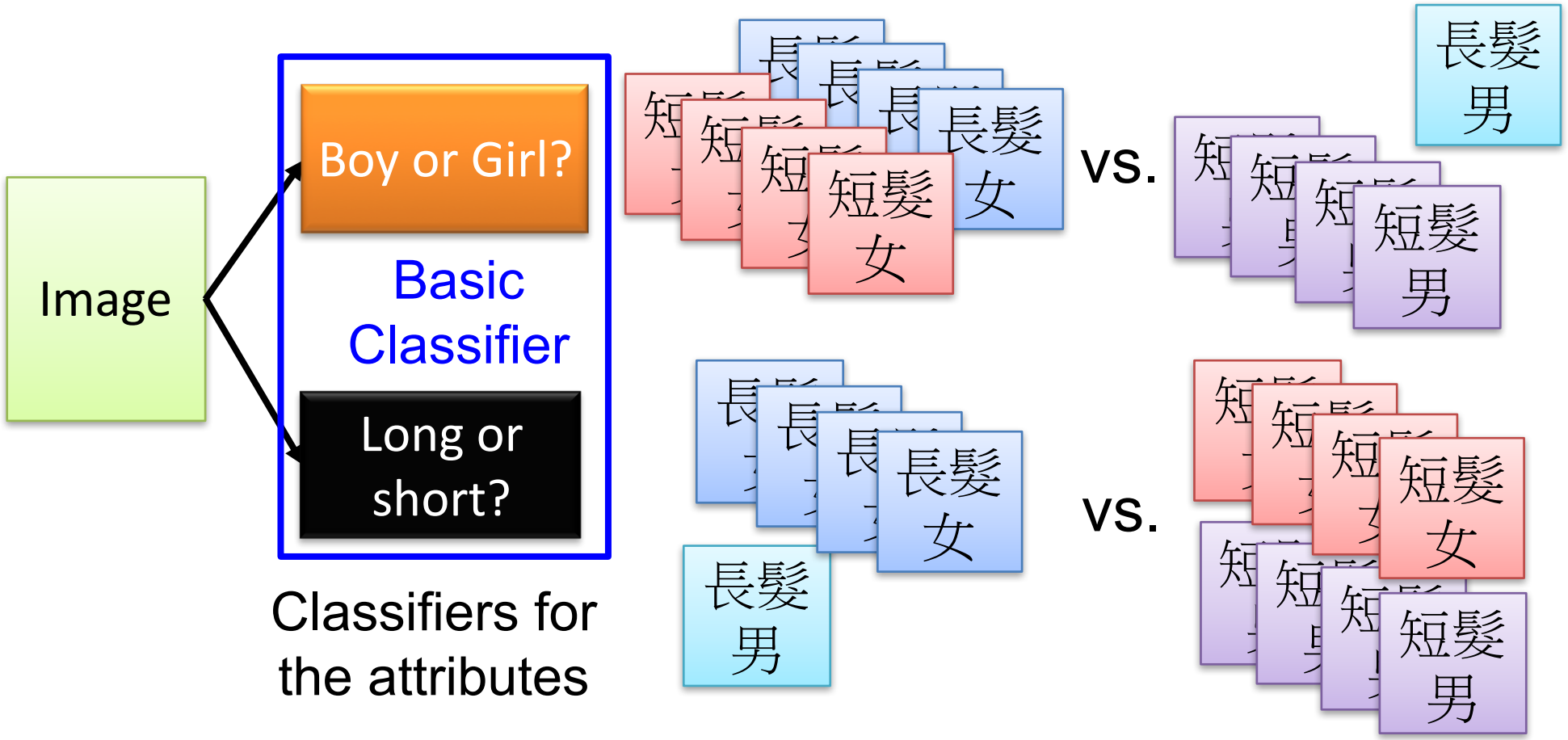
- Deep → Modularization



Modul

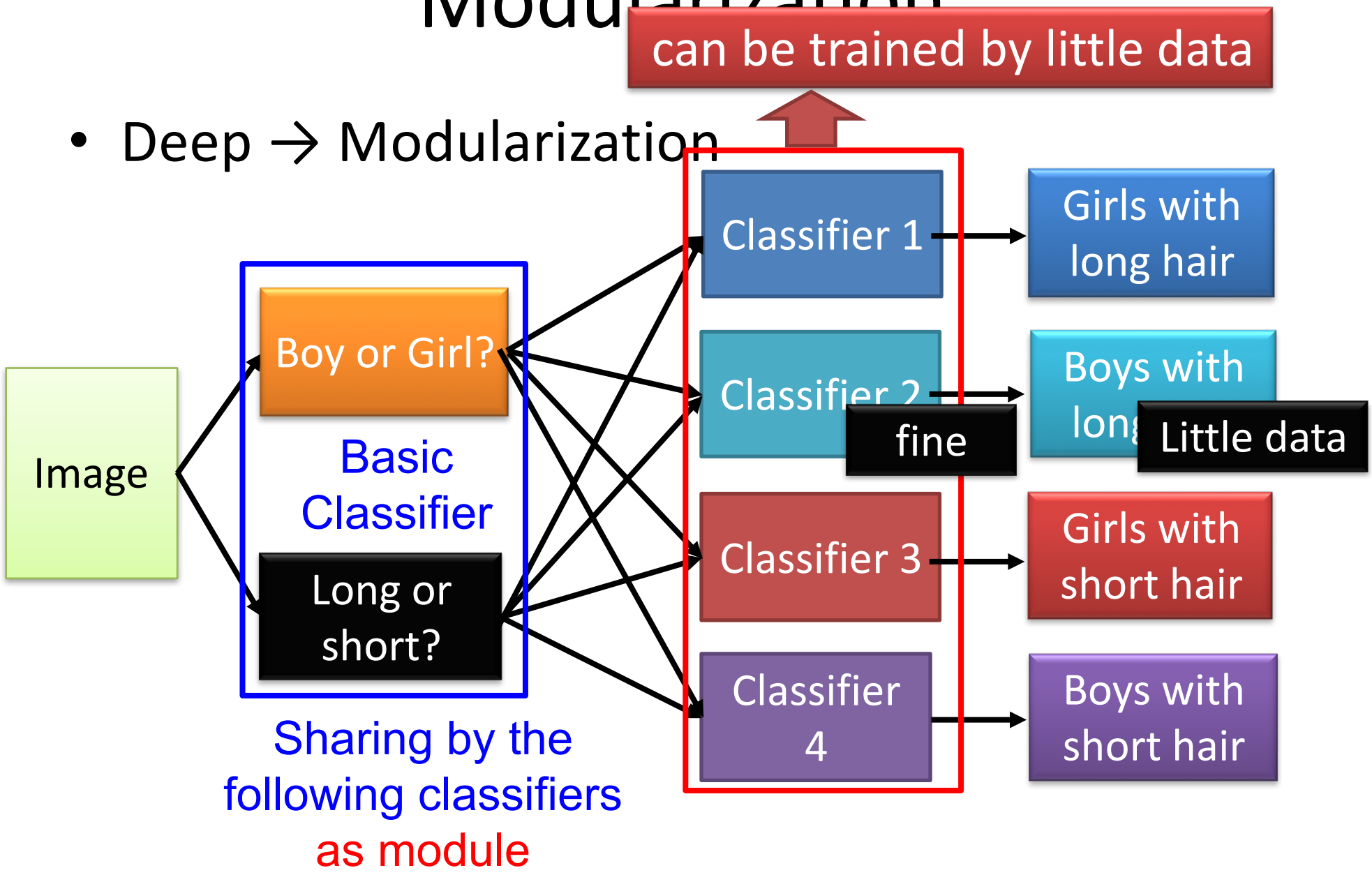
Each basic classifier can have sufficient training examples.

- Deep → Modularization



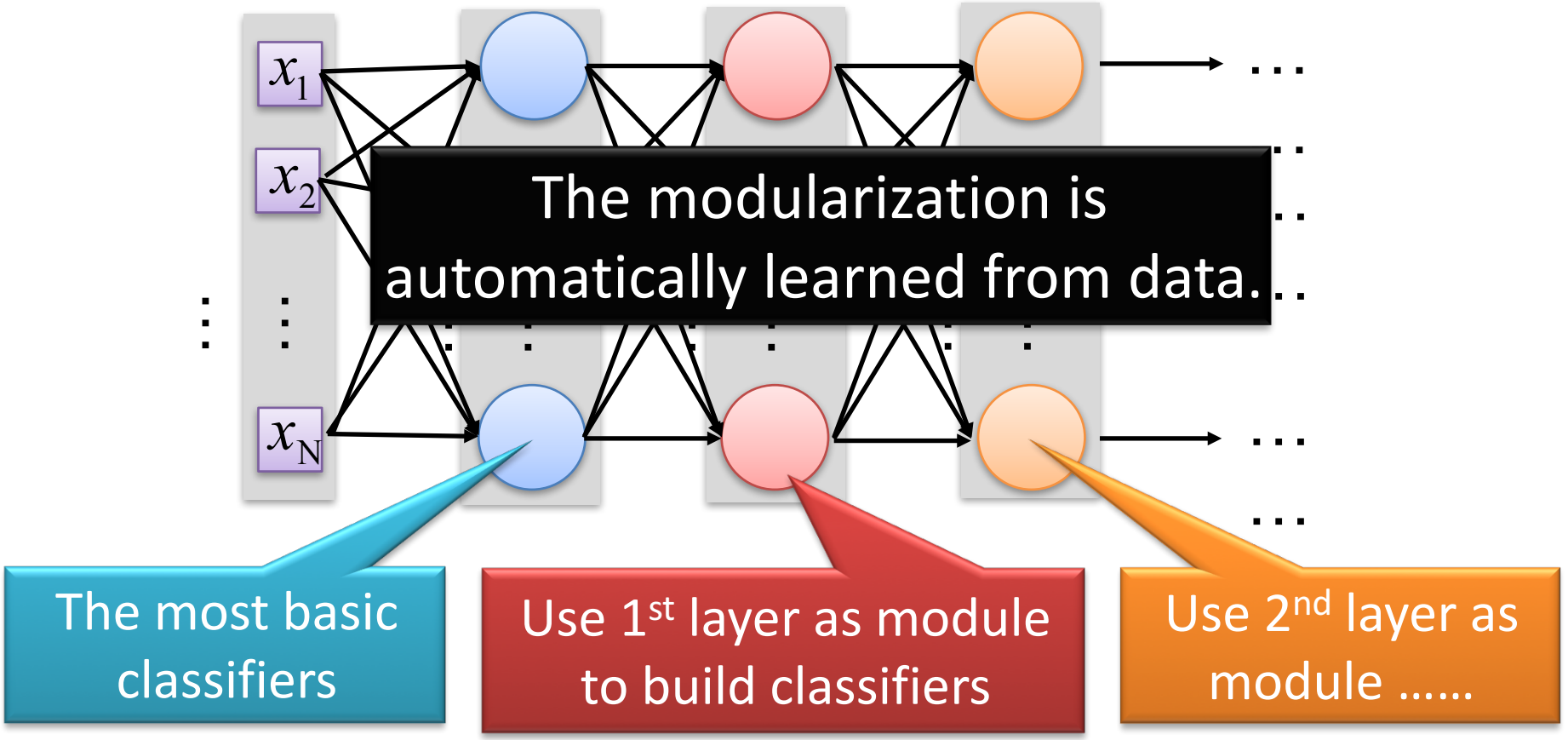
Modularization

- Deep → Modularization



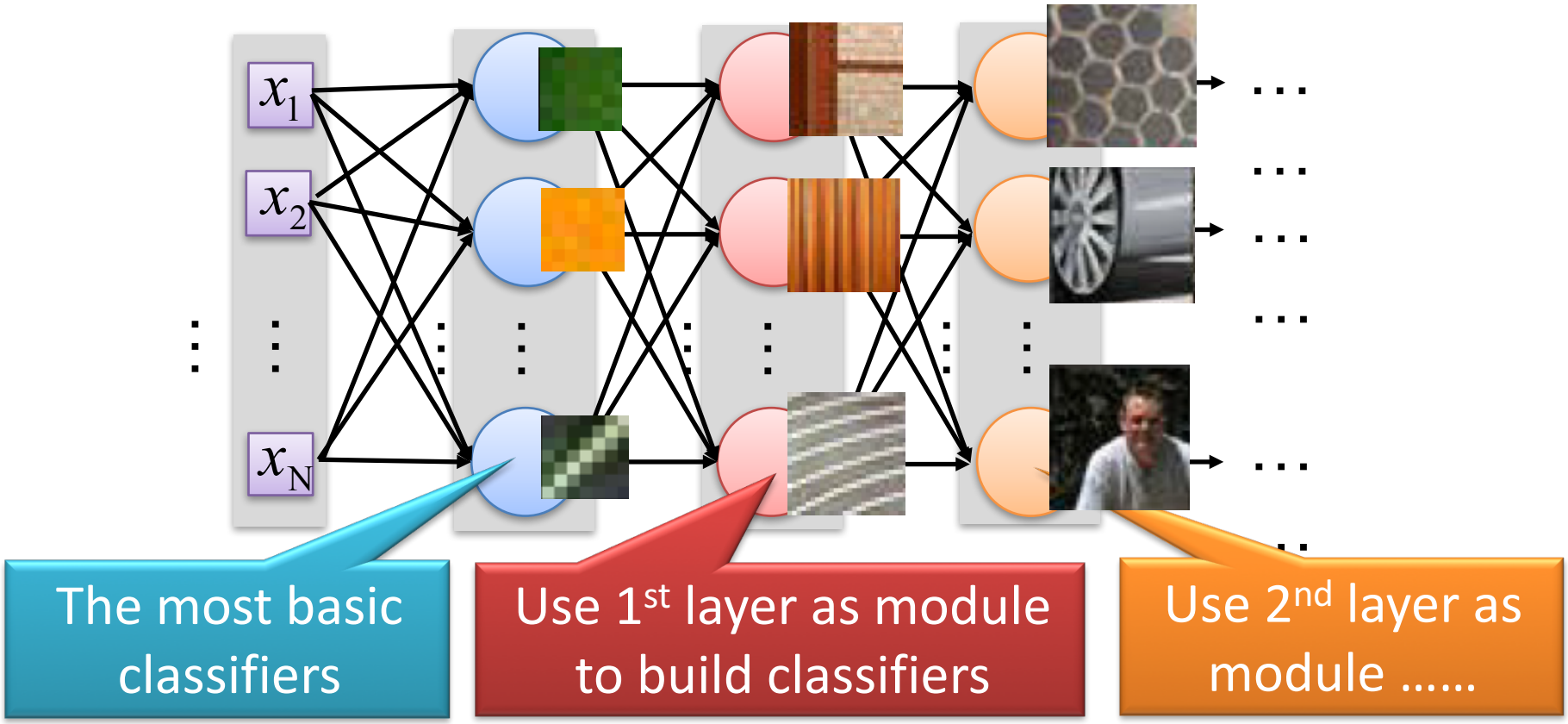
Modularization

- Deep → Modularizati → Less training data?

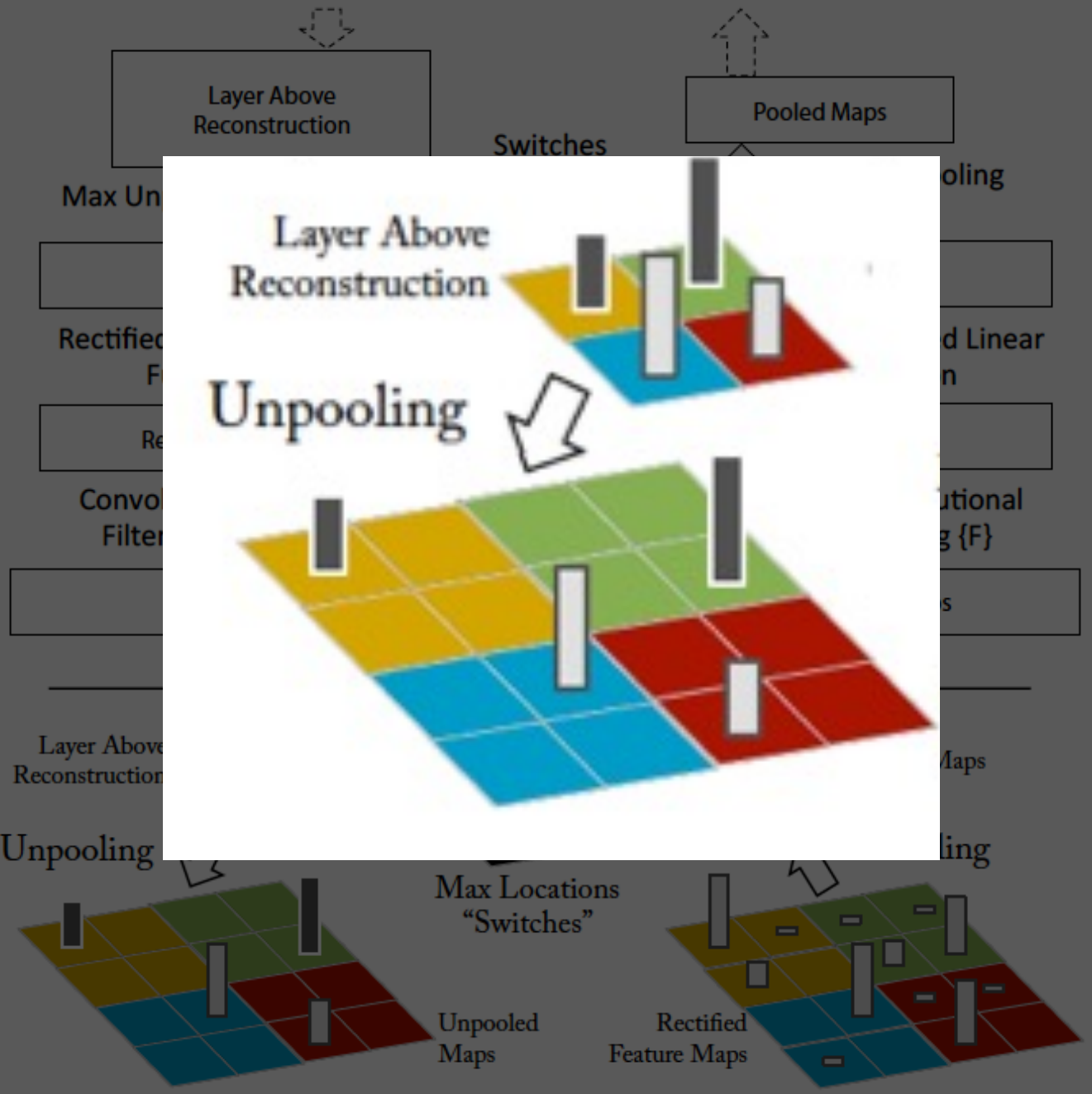


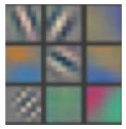
Modularization - Image

- Deep → Modularization

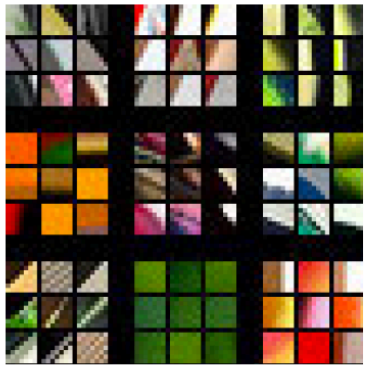


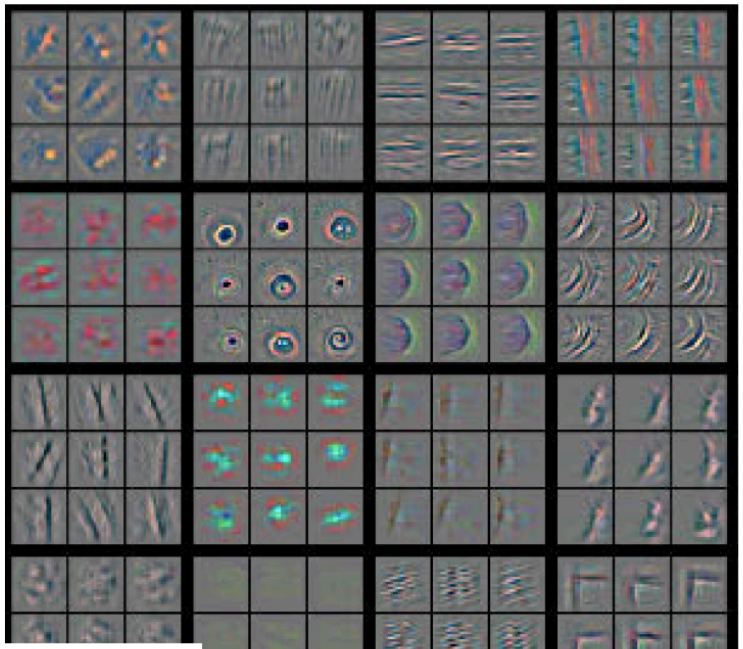
Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)


Layer Above Reconstruction
Pooled Maps
Switches
Layer Above Reconstruction
Unpooling
Max Locations "Switches"
Unpooled Maps
Rectified Feature Maps

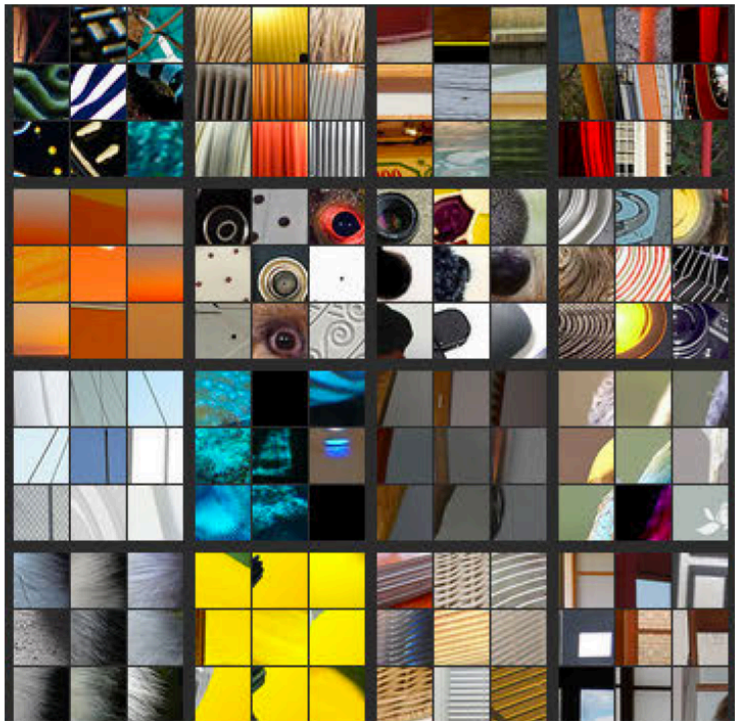


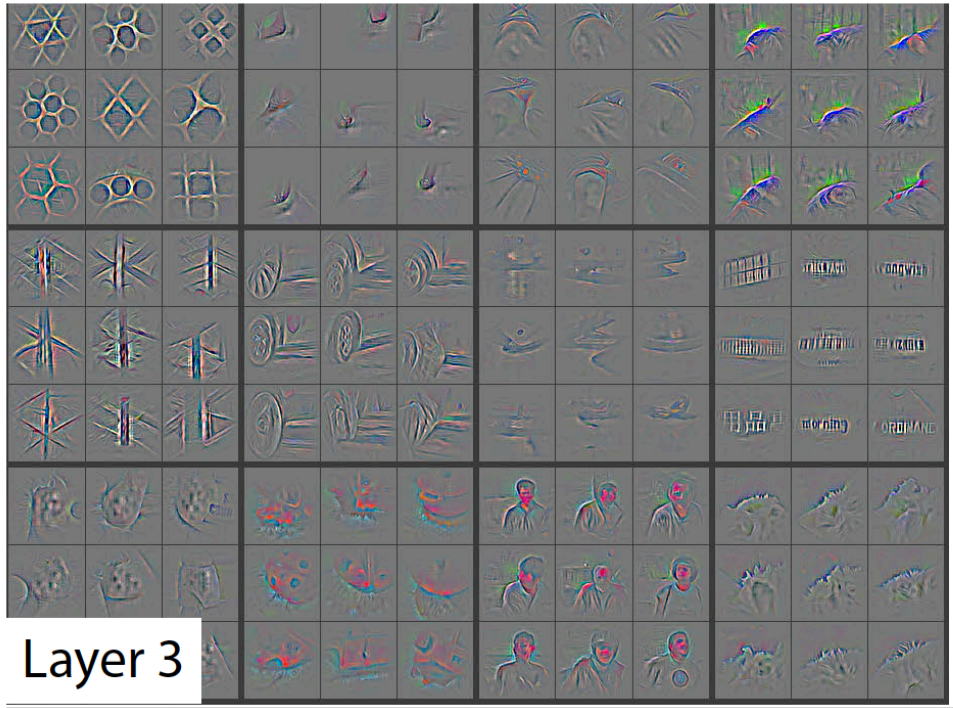
Layer 1



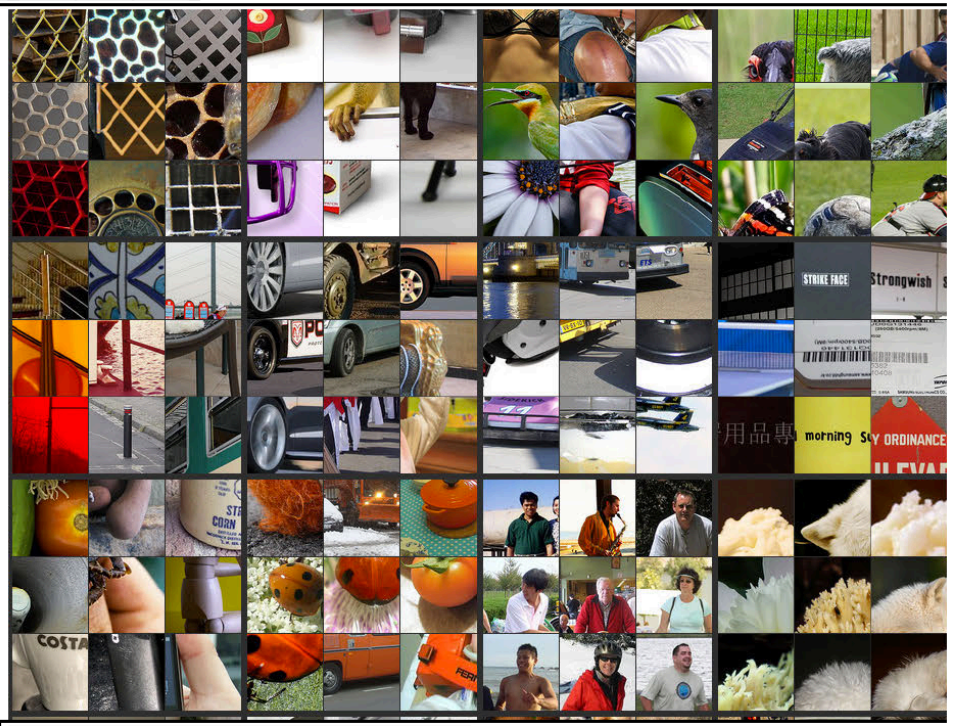


Layer 2





Layer 3



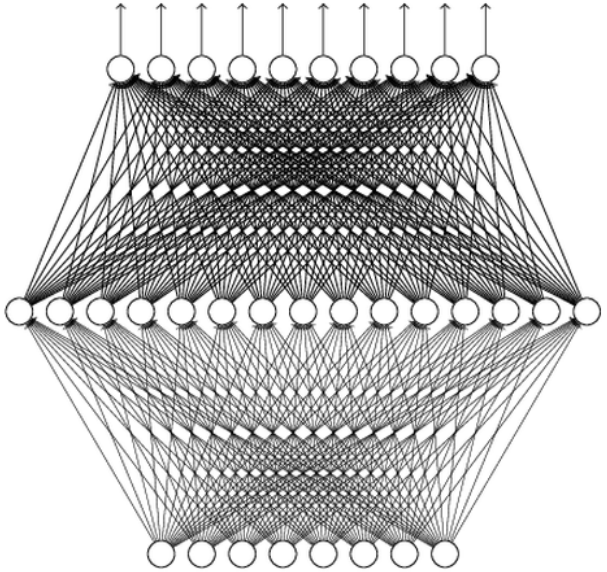
Universality Theorem

Any continuous function f

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

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)

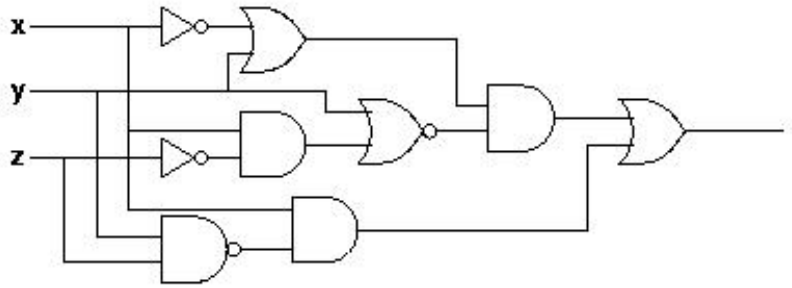


Reference for the reason:
http://neuralnetworksanddeeplearning.com/chap4.html

Yes, shallow network can represent any function.

However, using deep structure is more effective.

Anal



Logic circuits

- Logic circuits consists of **gates**
- **A two layers of logic gates** can represent **any Boolean function.**
- Using multiple layers of logic gates to build some functions are much simpler

→ less gates needed

Neural network

- Neural network consists of **neurons**
- **A hidden layer network** can represent **any continuous function.**
- Using multiple layers of neurons to represent some functions are much simpler

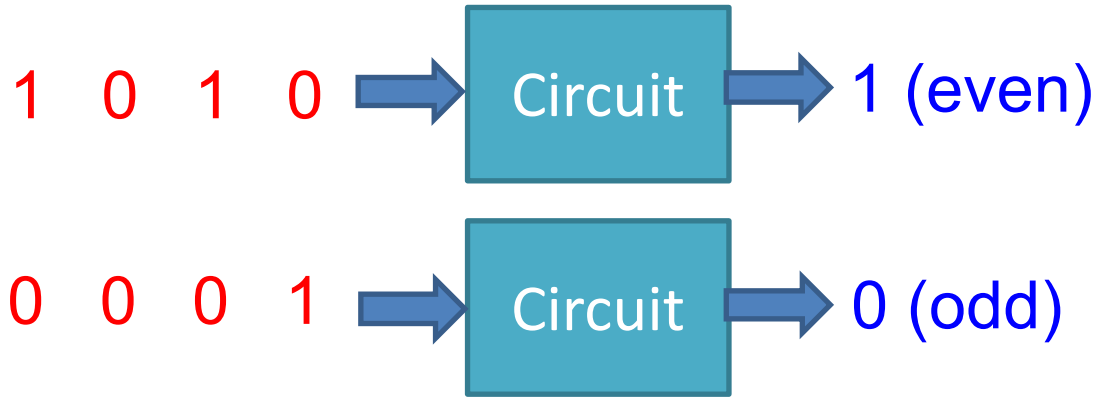
→ less parameters → less data?

This page is for EE background.

Analo

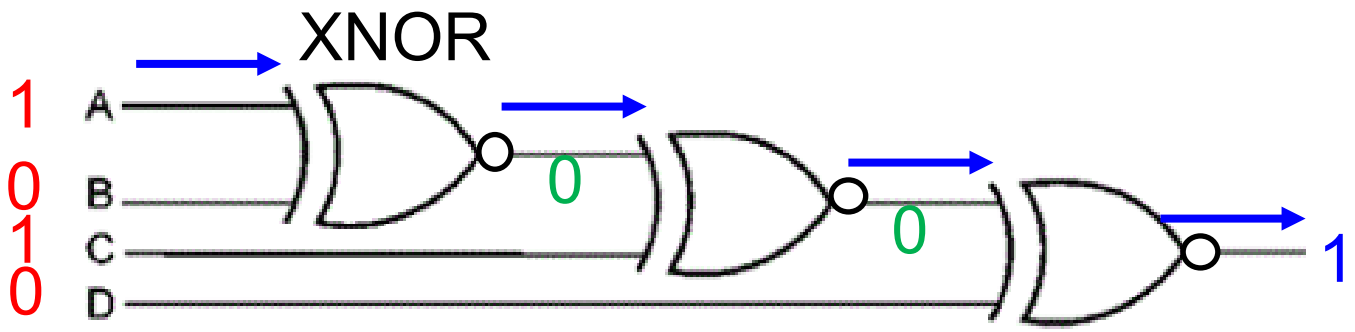
A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

- E.g. ***parity check***



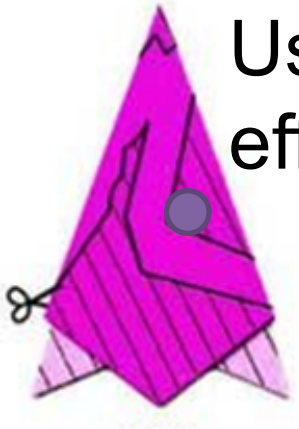
For input sequence with d bits,

Two-layer circuit need $O(2^d)$ gates.

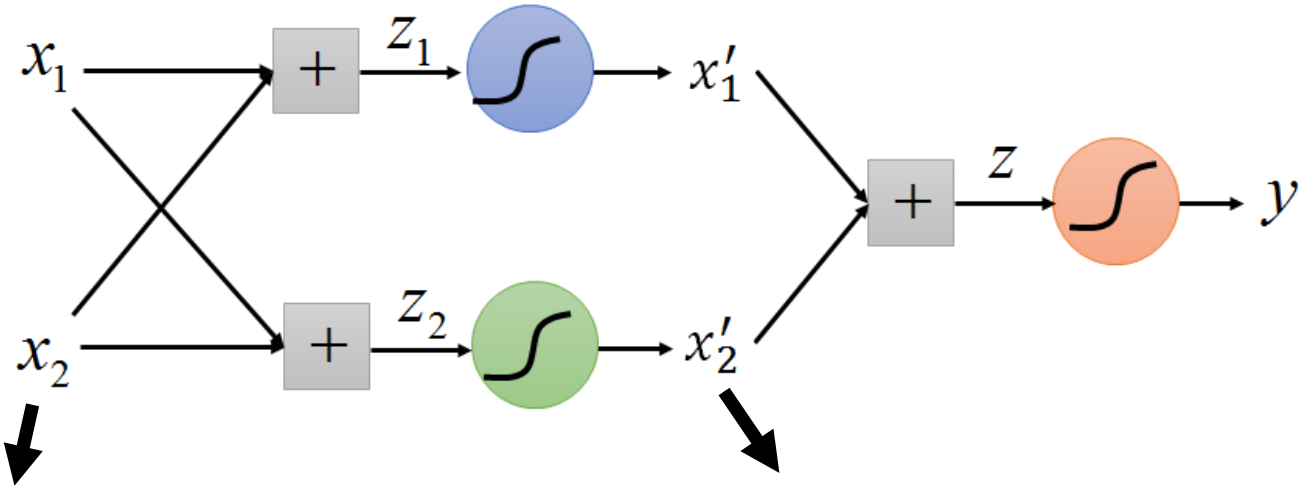


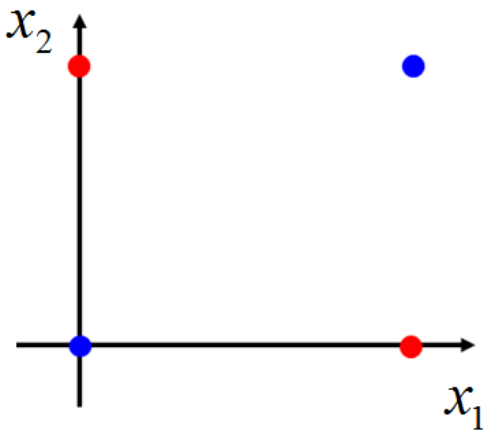
With multiple layers, we need only O(d) gates.

More A

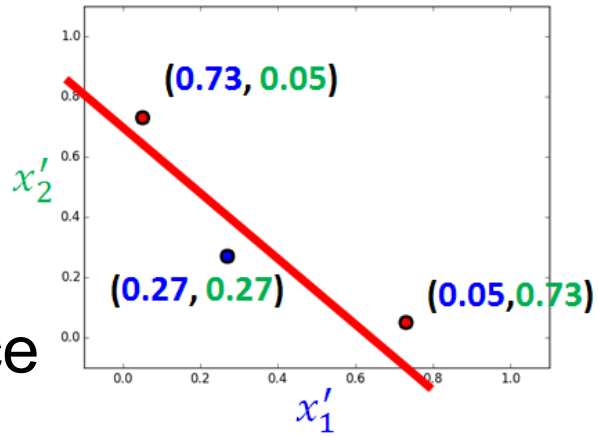


Use data effectively





Folding the space

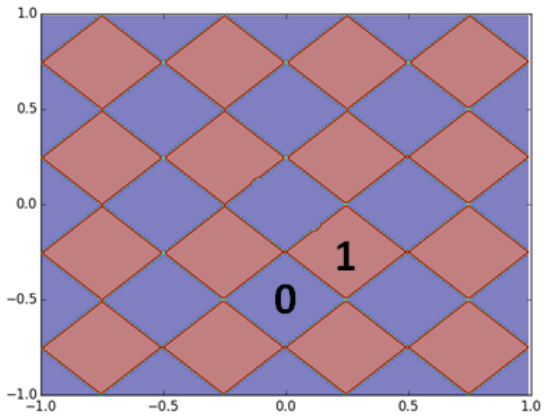


More Analogy - Experiment

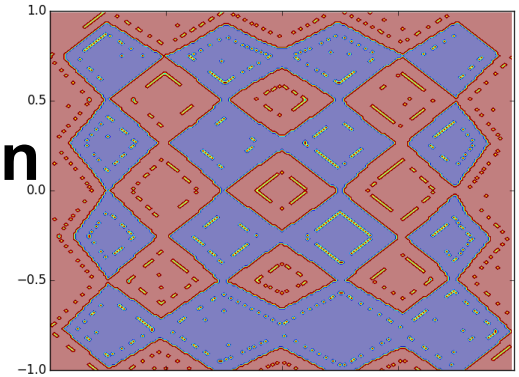
Different numbers of training examples

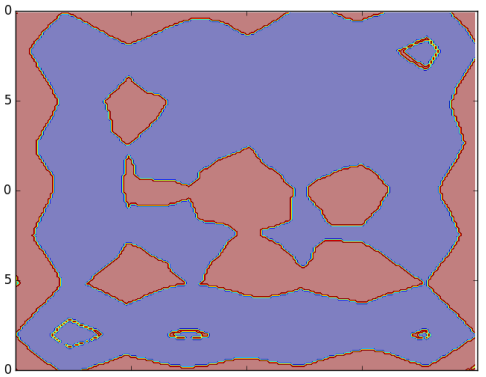
10,0000 2,0000

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

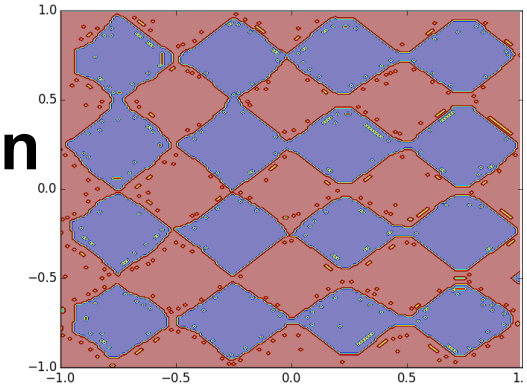


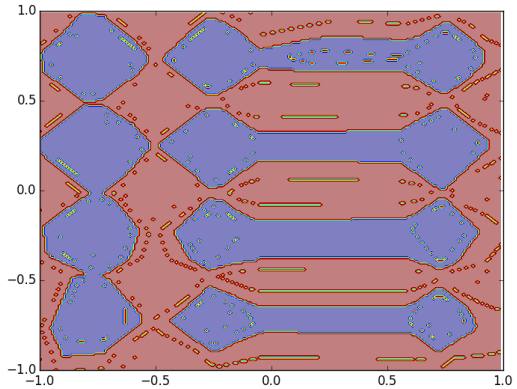
1 hidden layer





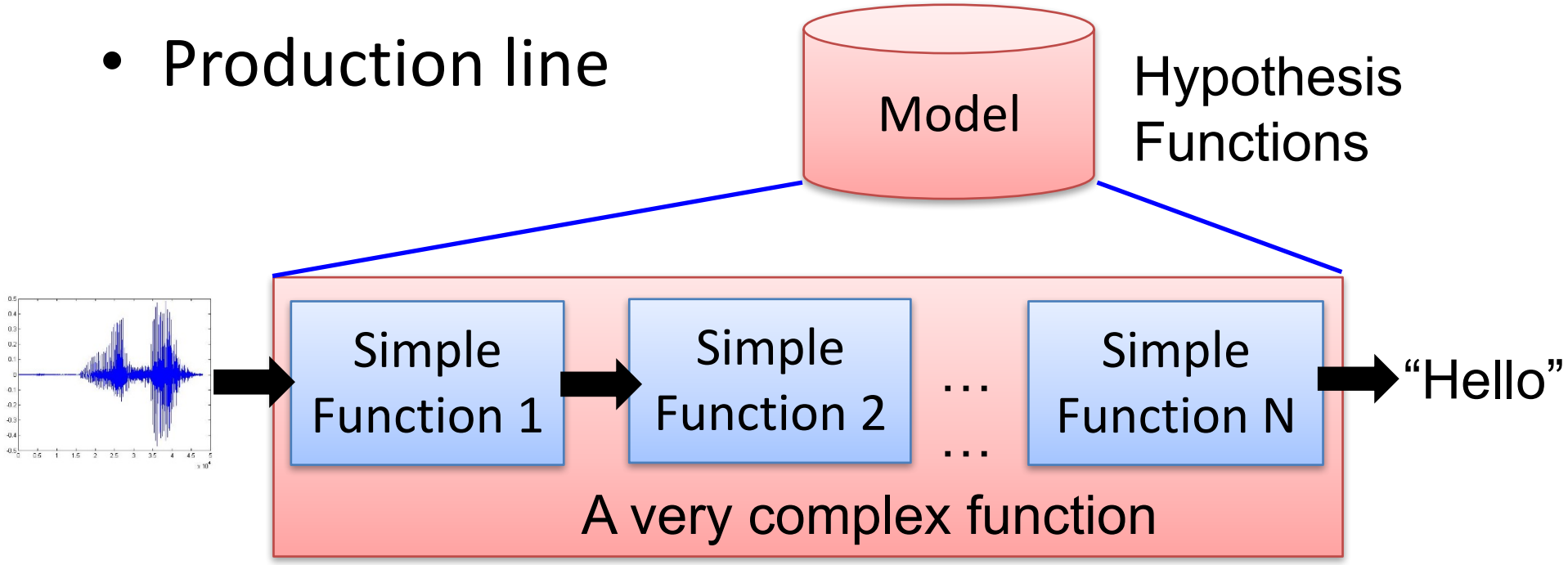
3 hidden layers





End-to-end Learning

- Production line

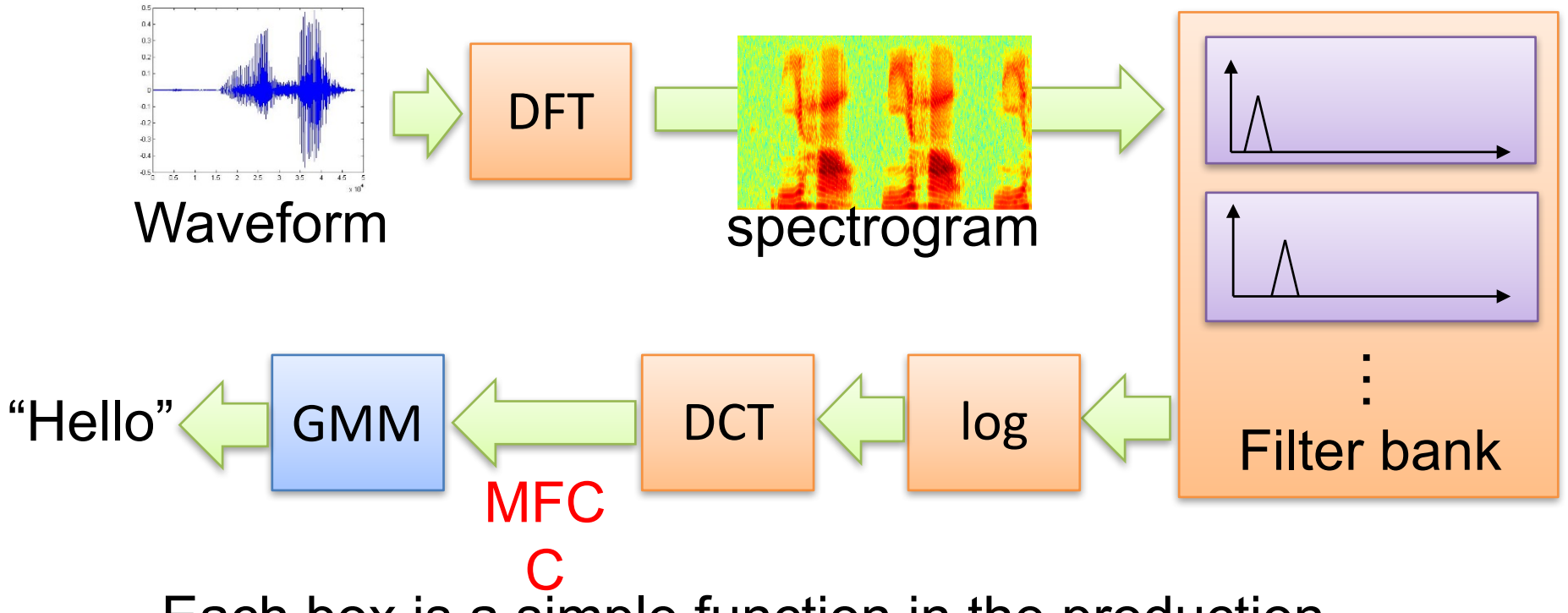


End-to-end training:

What each function should do is learned automatically

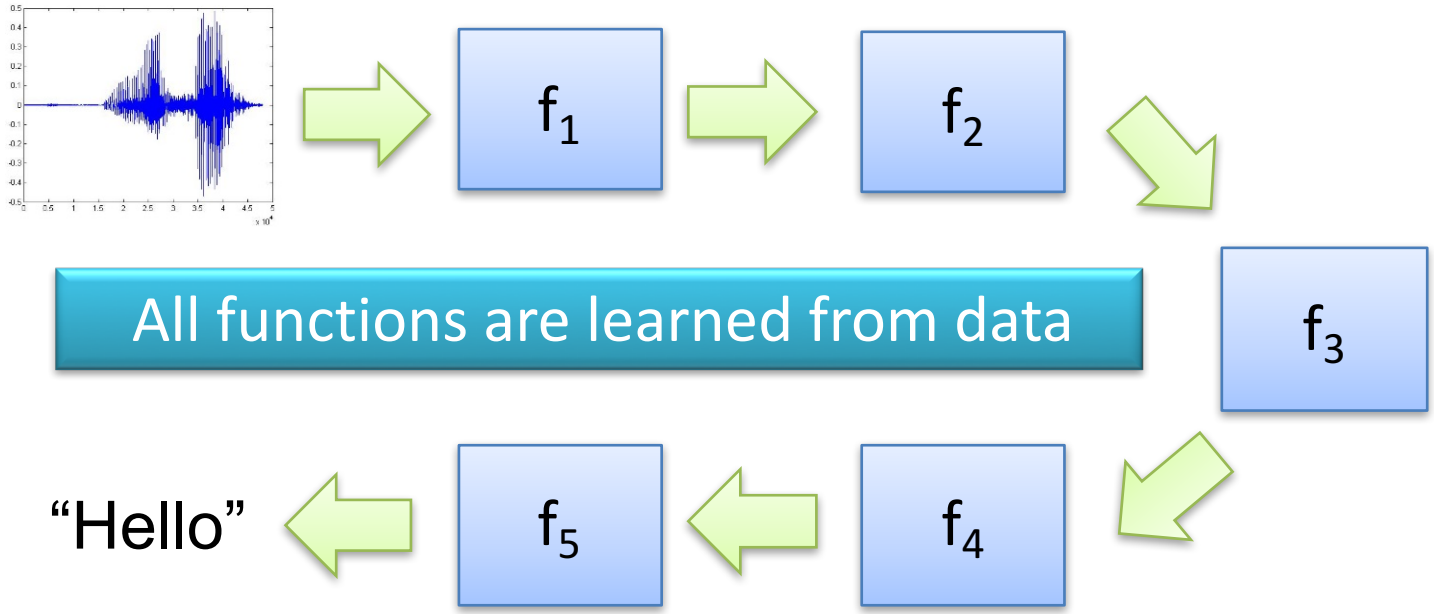
End-to-end Learning - Speech Recognition

- Shallow Approach



End-to-end Learning - Speech Recognition

- Deep Learning

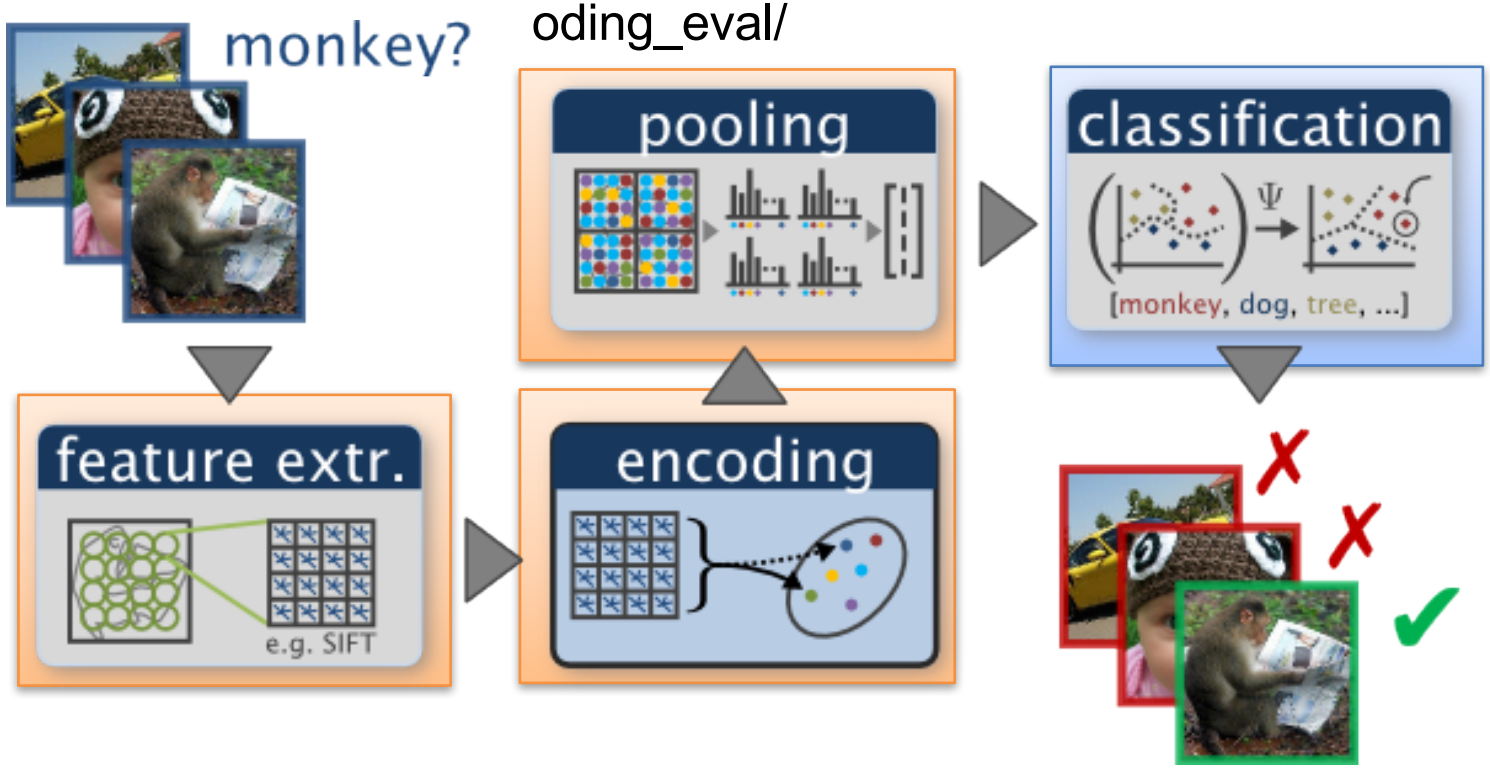


Less engineering labor, but machine learns more

End-to-end Learning - Image Recognition

- Shallow Approach

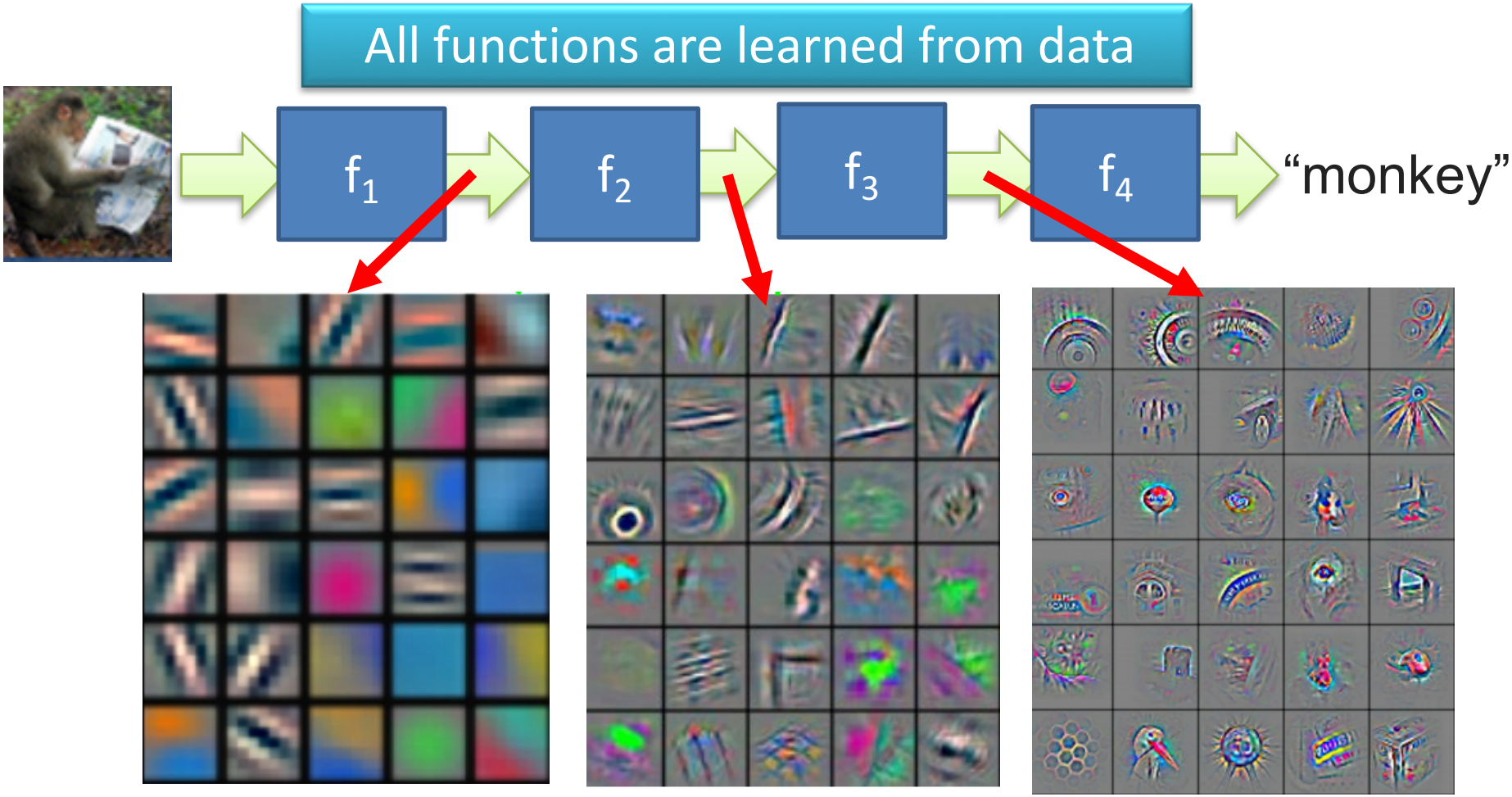
http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/



End-to-end Learning - Image Recognition

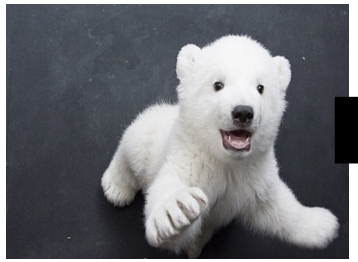
Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)

- Deep Learning



Complex Task …

- Very similar input, different output

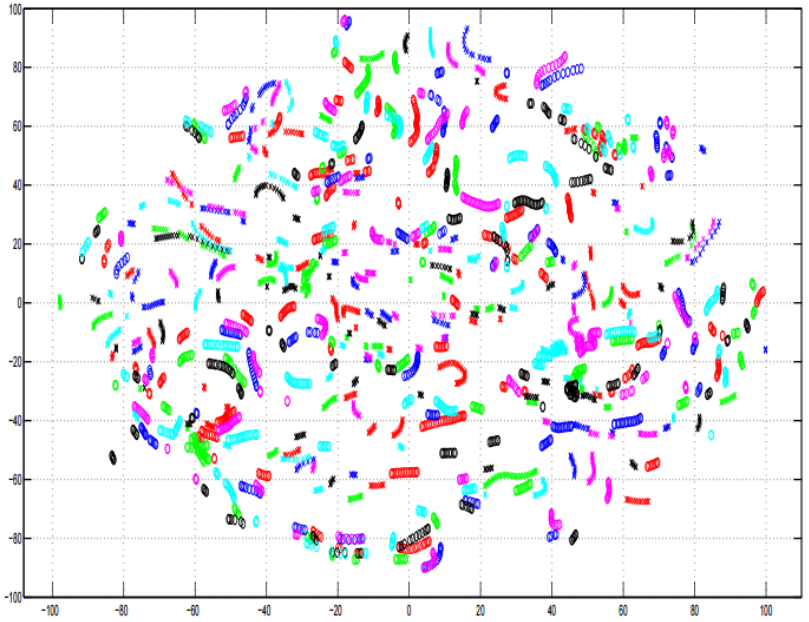


- Very different input, similar output

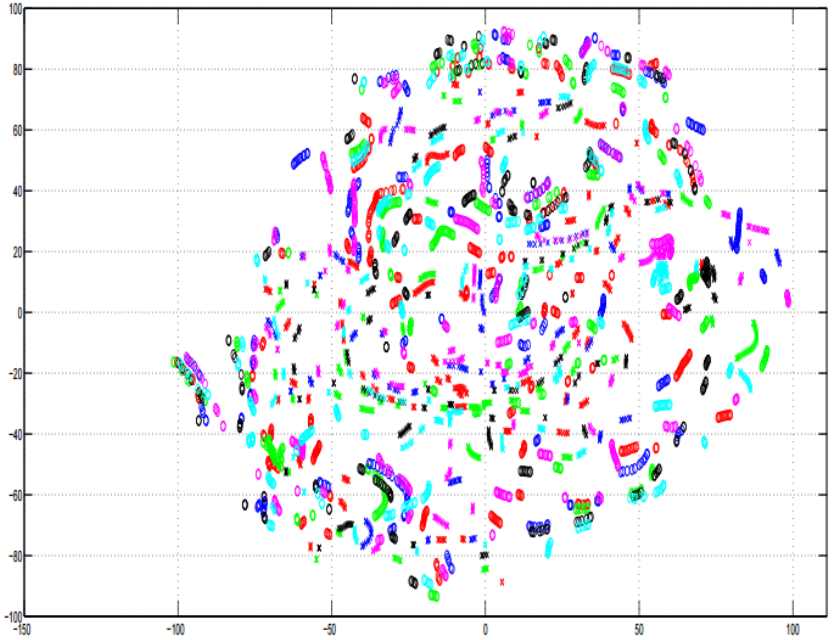
Complex Task …

A. Mohamed, G. Hinton, and G. Penn, "Understanding how Deep Belief Networks Perform Acoustic Modelling," in ICASSP, 2012.

- Speech recognition: Speaker normalization is automatically done in DNN

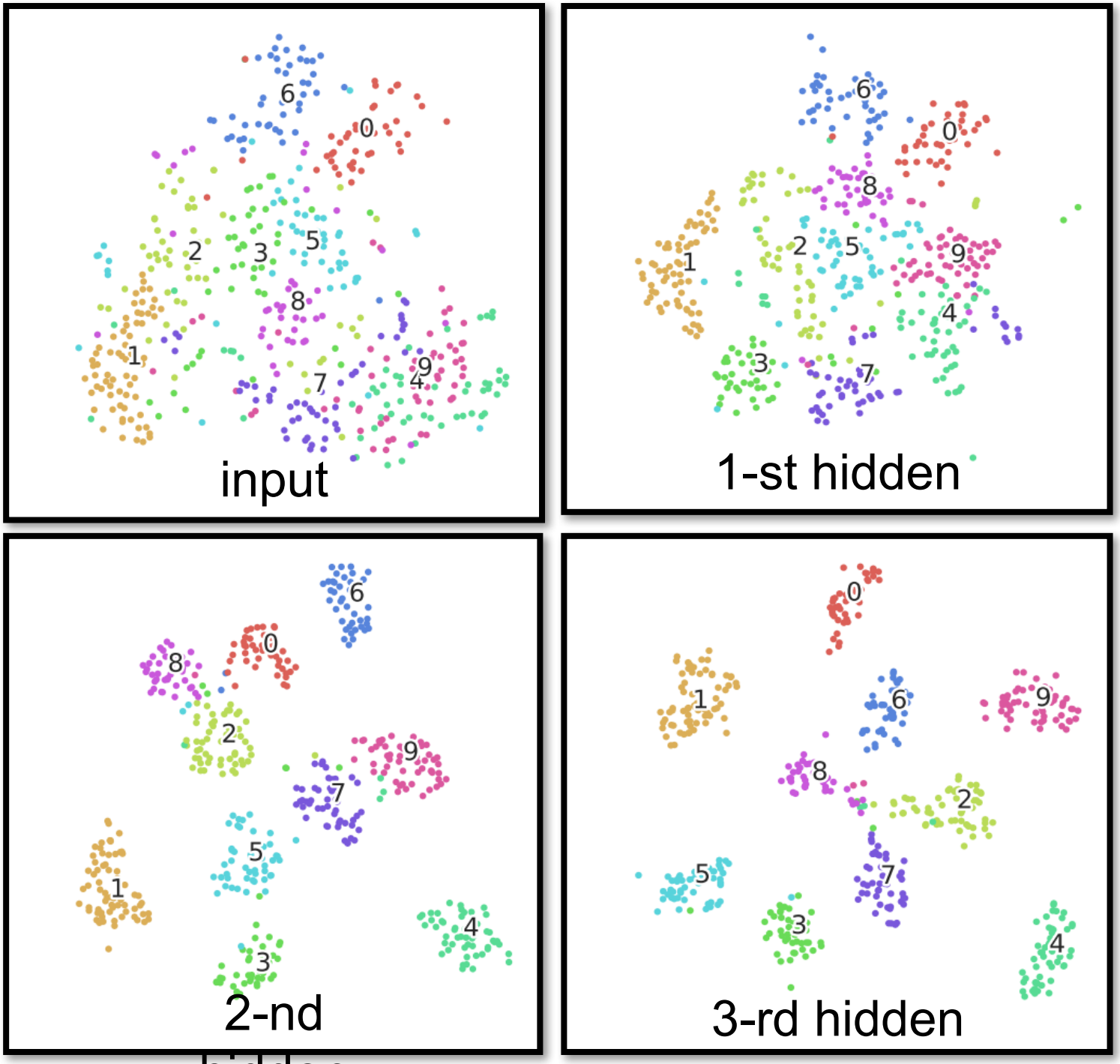


Input Acoustic Feature (MFCC)



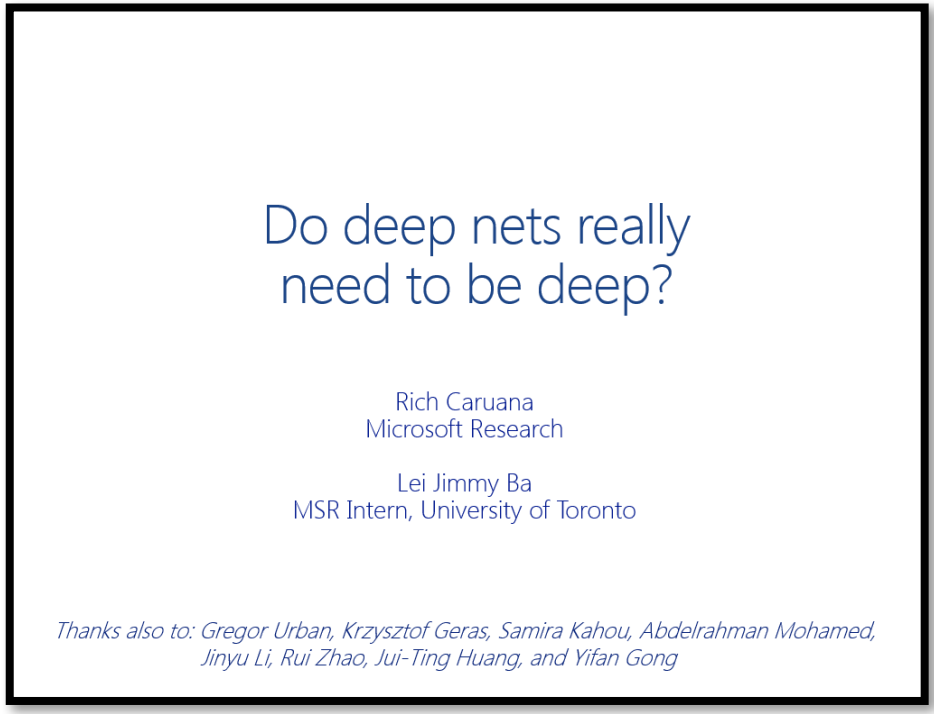
1-st Hidden Layer

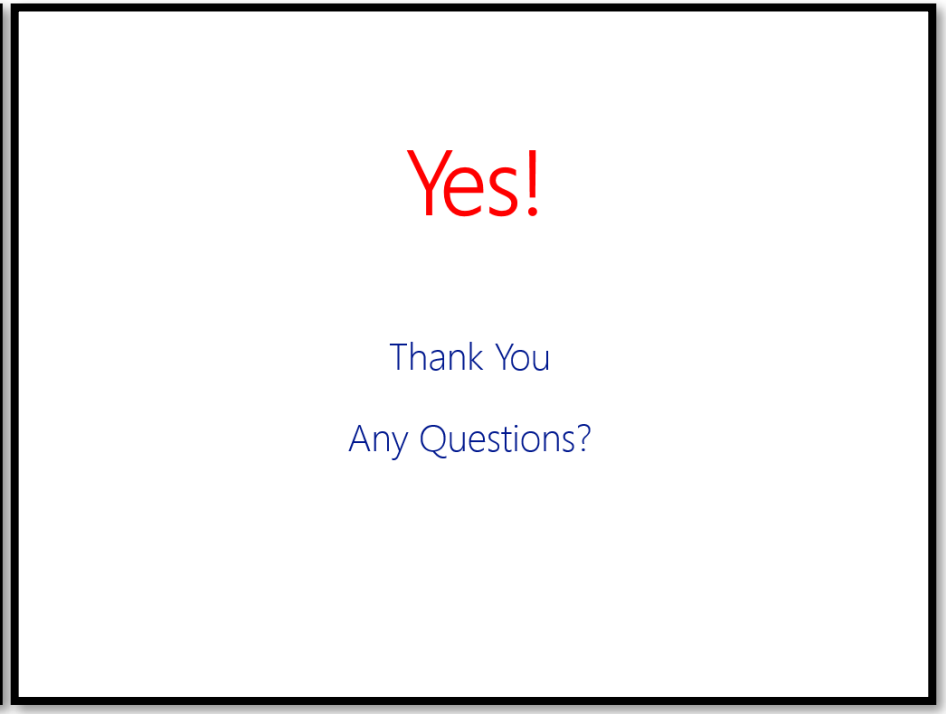
MNIST



To learn more …

- Do Deep Nets Really Need To Be Deep? (by Rich Caruana)
- http://research.microsoft.com/apps/video/default.aspx?id=232373&r=1



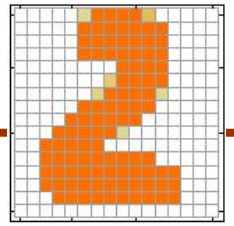


keynote of Rich Caruana at ASRU 2015

To learn more …

- Deep Learning: Theoretical Motivations (*Yoshua Bengio*)
  - http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/
- Connections between physics and deep learning
  - https://www.youtube.com/watch?v=5MdSE-N0bxs
- Why Deep Learning Works: Perspectives from Theoretical Chemistry
  - https://www.youtube.com/watch?v=kIbKHIPbxiU

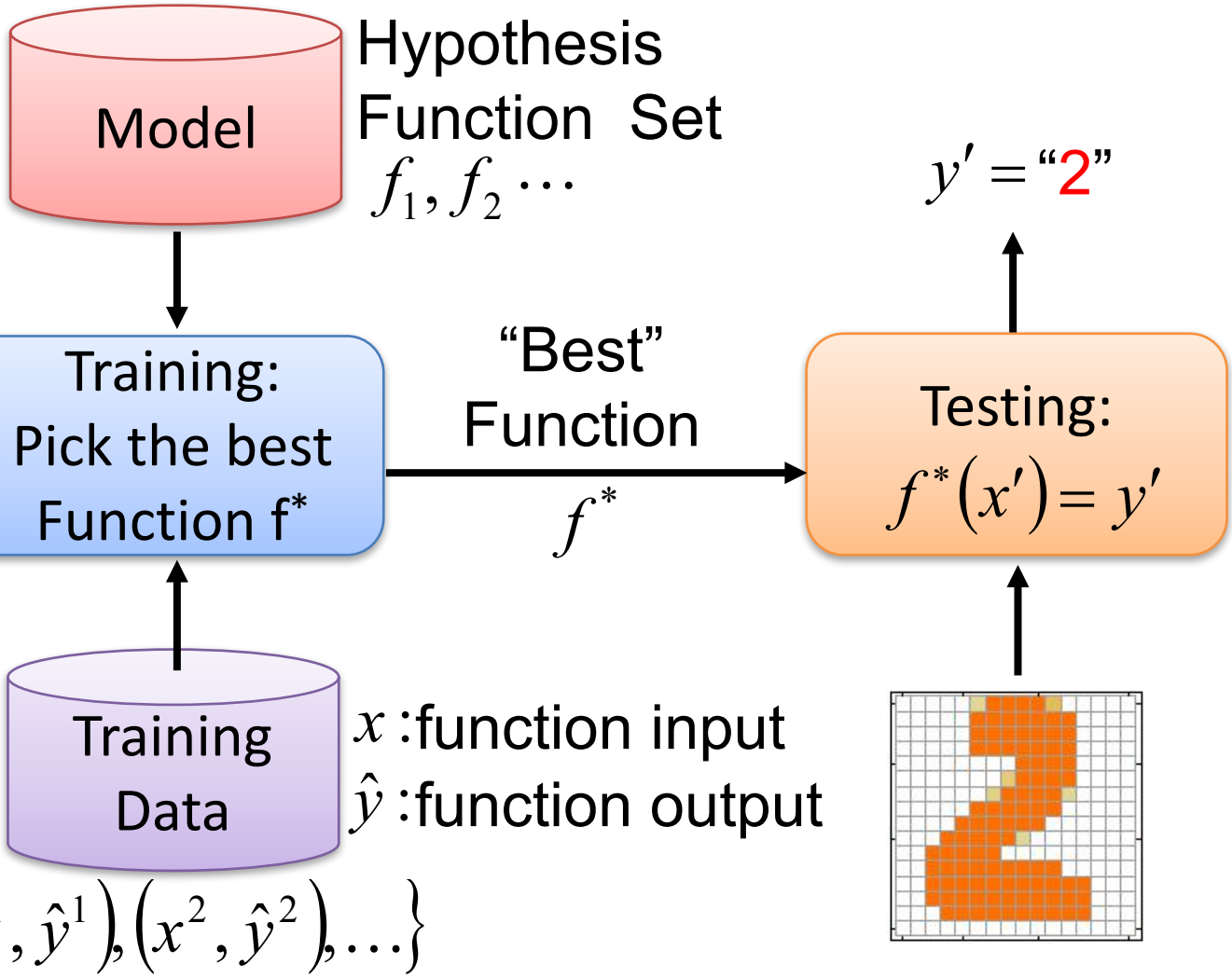
Framework



x :

$\hat{y}$: "2"

(label)



Outline

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

2. What is the “best” function?

3. How to pick the “best” function?

1. What is the model?

What is the function we are looking for?

- ***classification***

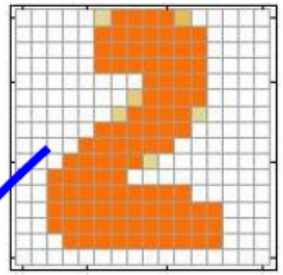
$$y = f(x) \quad \Rightarrow \quad f: R^N \rightarrow R^M$$

  - 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?

- ***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

10 dimensions for digit recognition

"1" "2"

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \text{“1”} \\ \text{“2”} \\ \text{“3”} \end{matrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

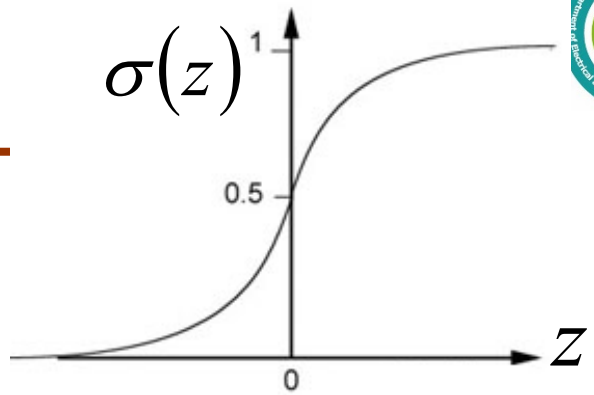
“1” → “1” or not
“2” → “2” or not
“3” → “3” or not

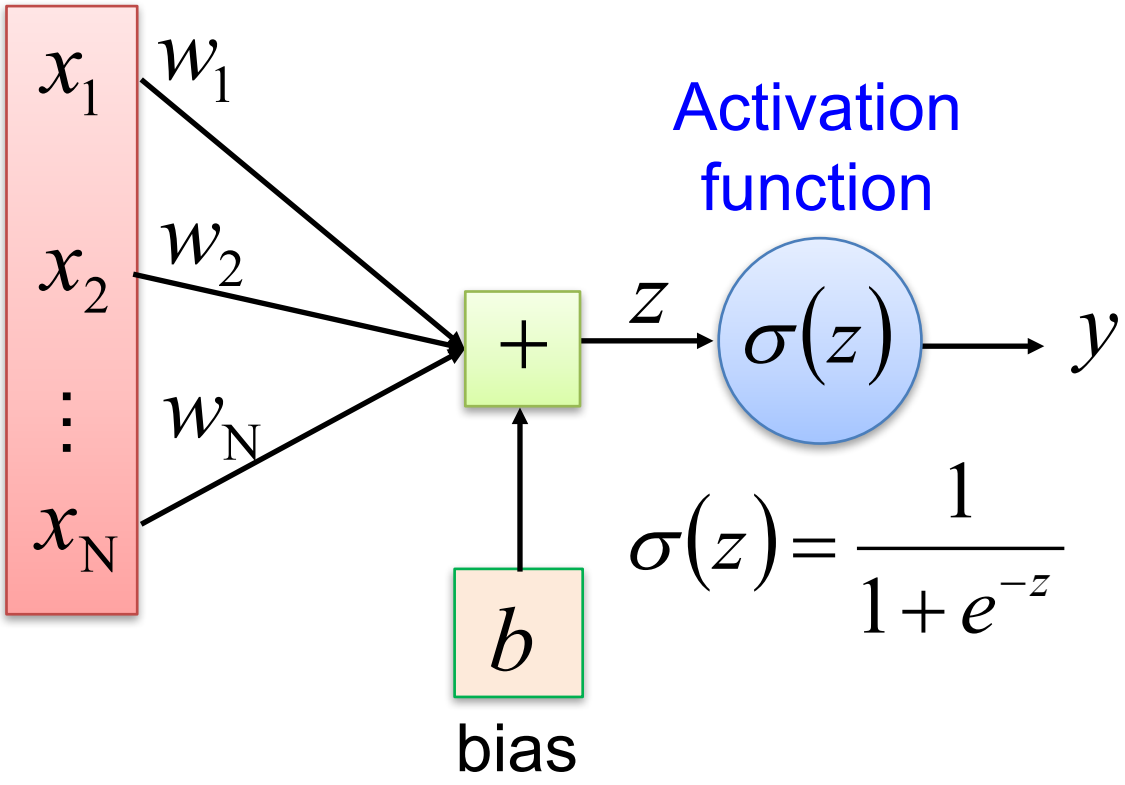
1. What is the model?

A Layer of Neuron

Single Neuron

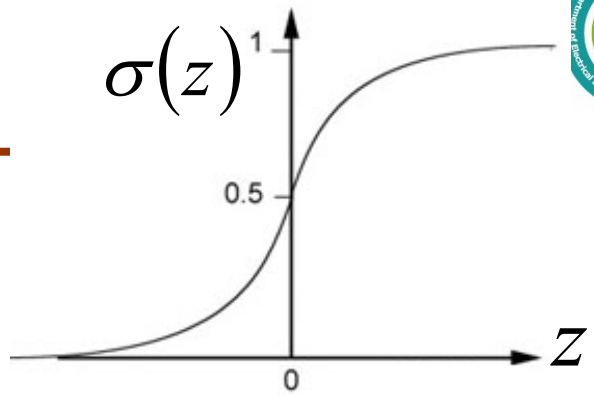
$$f: R^N \rightarrow R$$

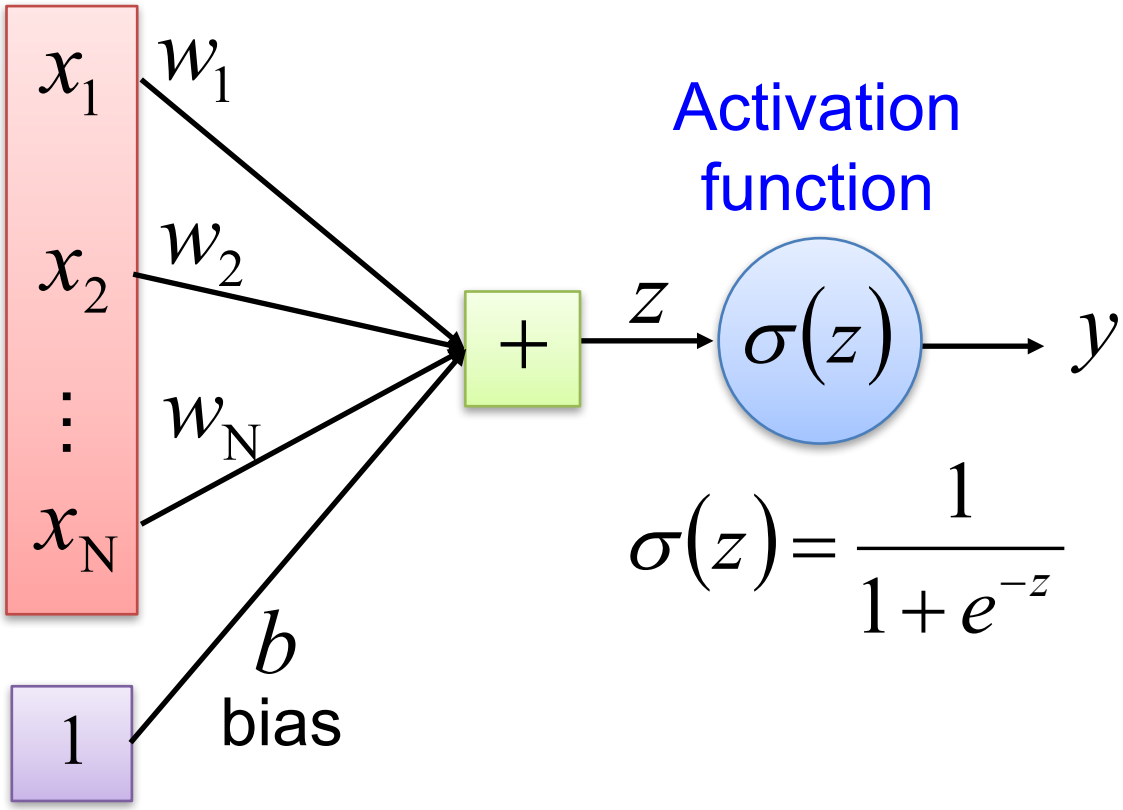




Single Neuron

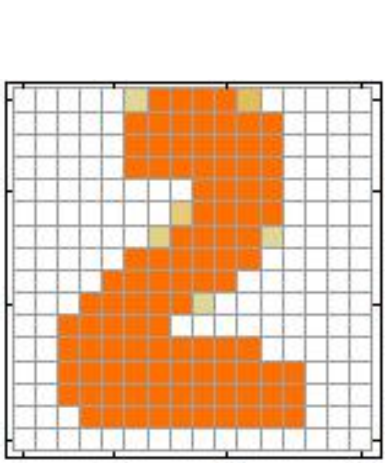
$$f: R^N \rightarrow R$$

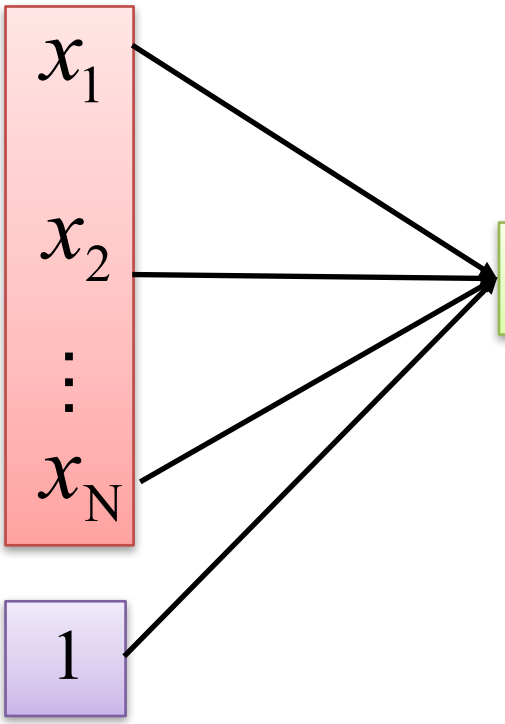




Single Neuron $f: R^N \rightarrow R$

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





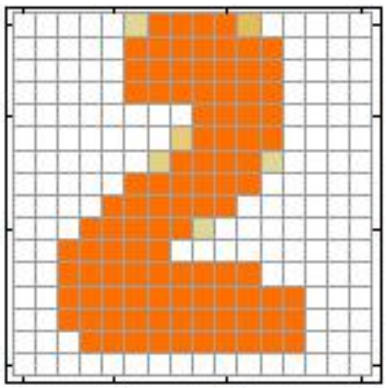
y

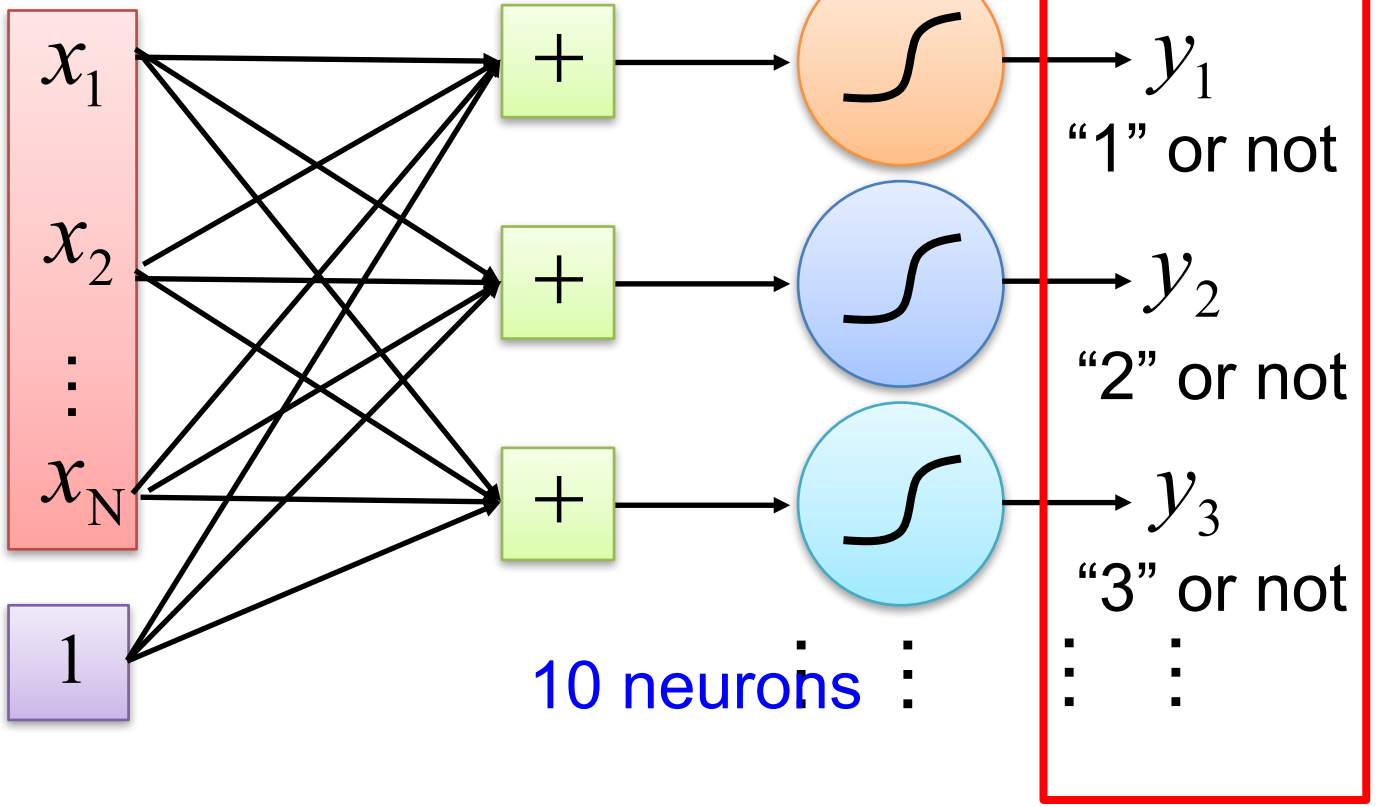
$$\begin{cases} is \quad "2" & y \geq 0.5 \\ not\ "2" & y < 0.5 \end{cases}$$

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”.



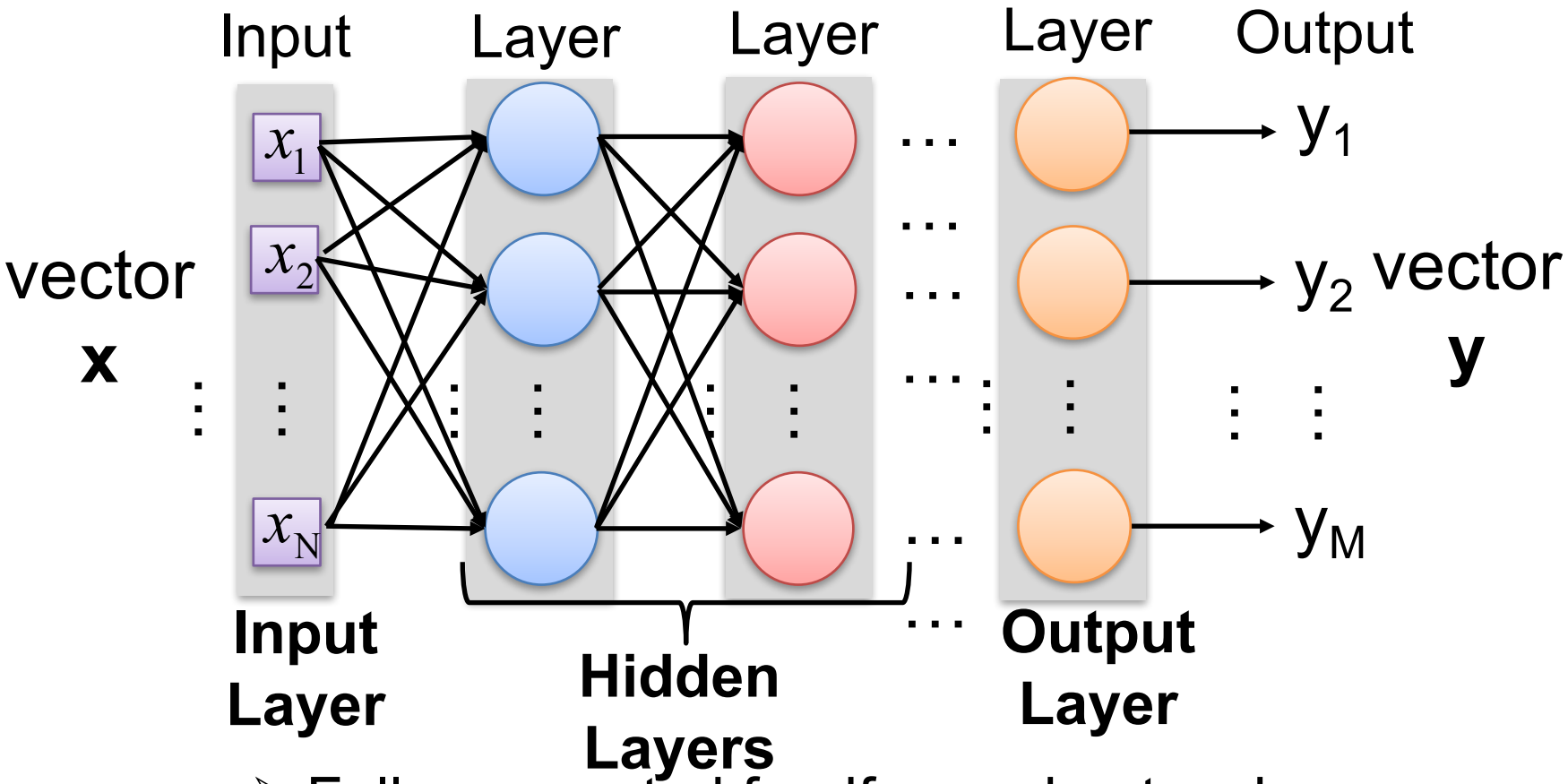


1. What is the model?

Neural Network

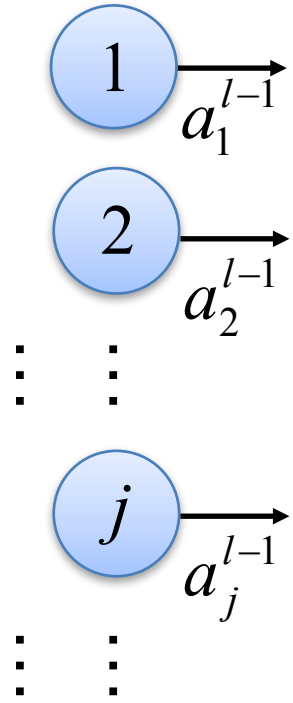
Neural Network as Model

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



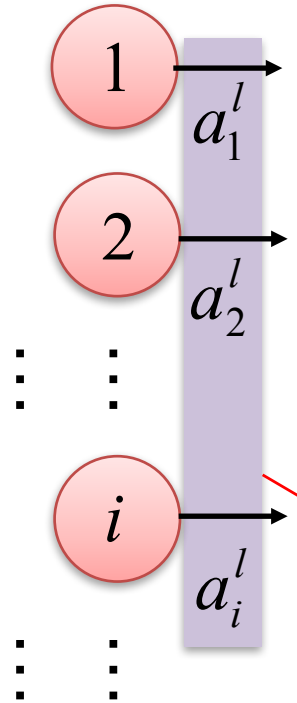
- Fully connected feedforward network
- Deep Neural Network: many hidden layers

Notation



Layer $l-1$

N_{l-1} nodes



Layer l

N_l nodes

Output of a neuron:

a_i^l

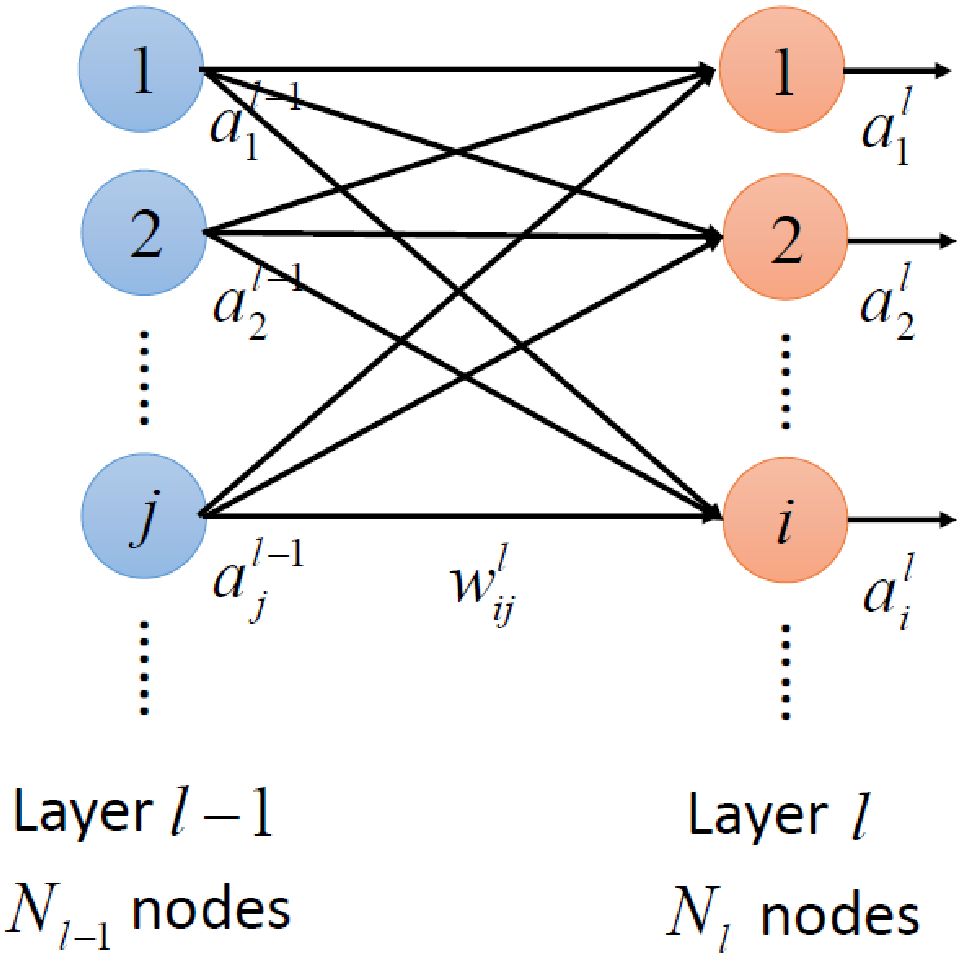
Layer l

Neuron i

Output of one layer:

a^l : a vector

Notation

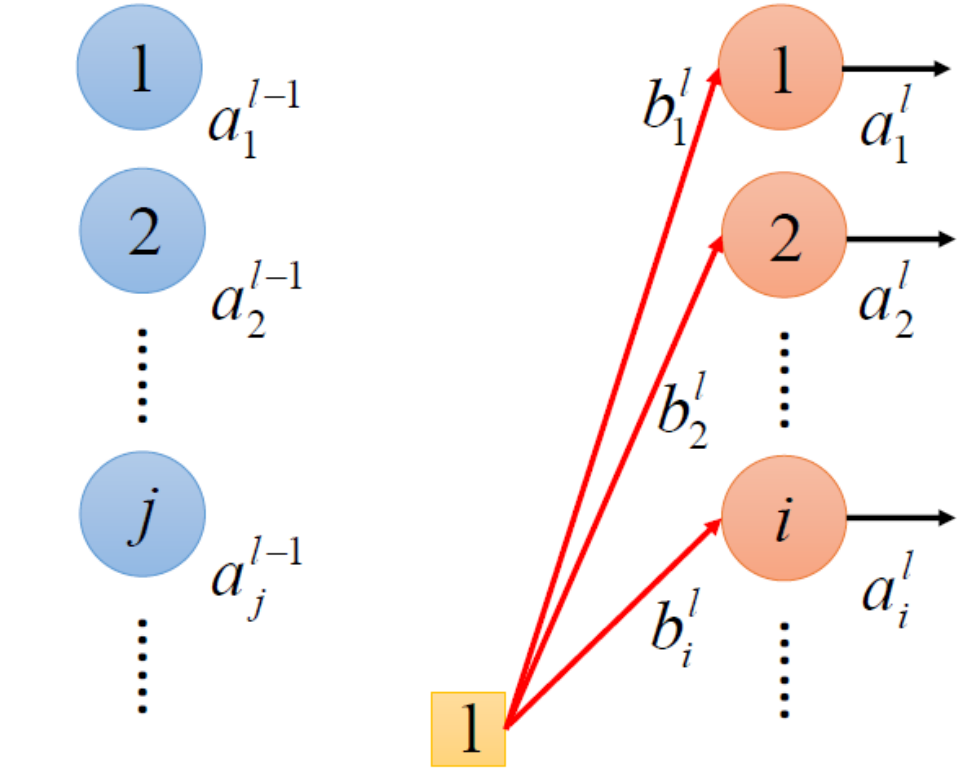


w_{ij}^l → Layer $l-1$ to Layer l

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

$$W^l = \underbrace{\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \\ & & \end{bmatrix}}_{\mathrm{N}_{l-1}} \Bigg\} \mathrm{N}_l$$

Notation



b_i^l : bias for neuron i at layer l

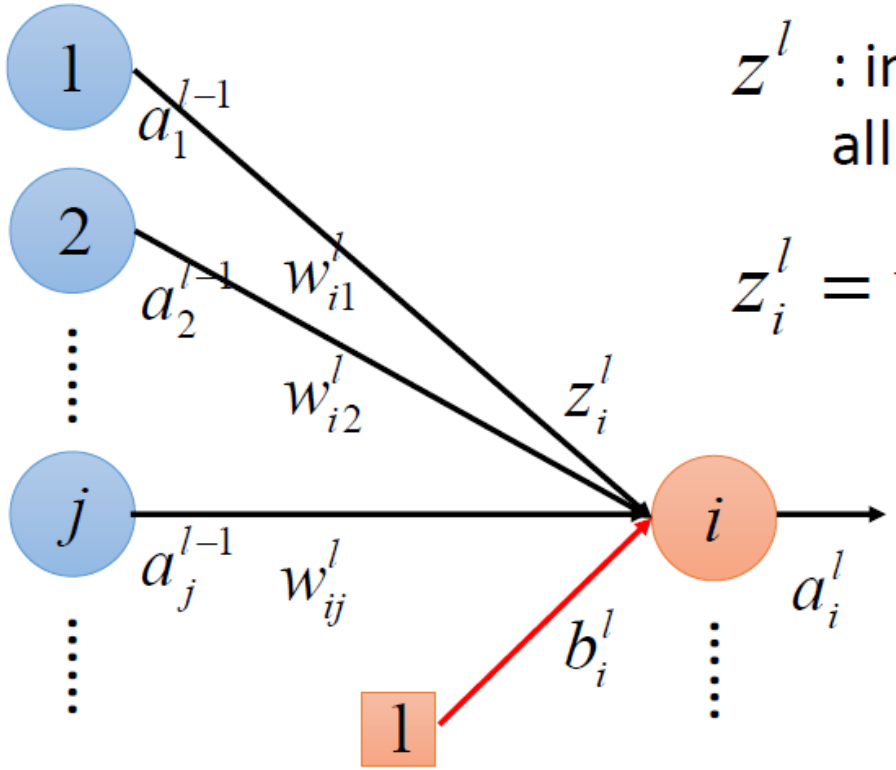
$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

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

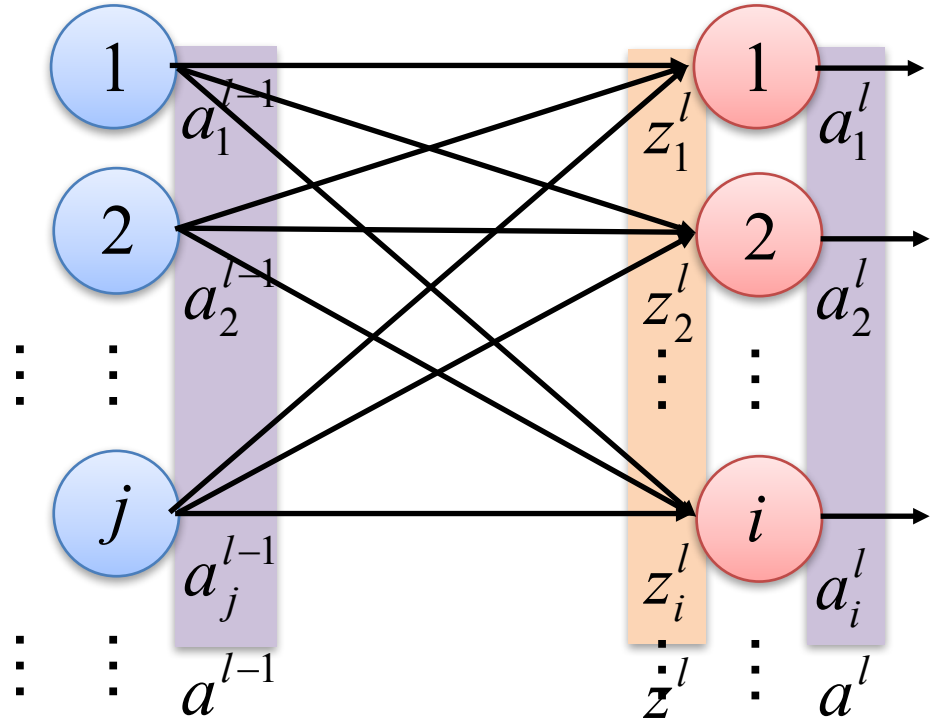
w_{ij}^l : a weight

$\mathbf{W}^l$: a weight matrix

b_i^l : a bias

b^l : a bias vector

Relations between Layer Outputs



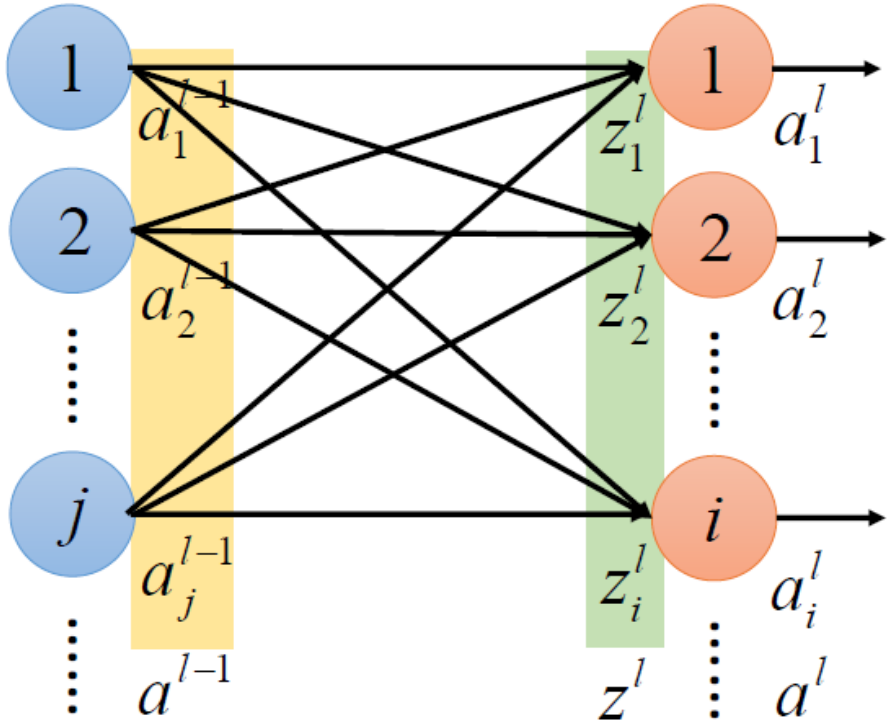
Layer $l-1$

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

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

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

$$\vdots$$

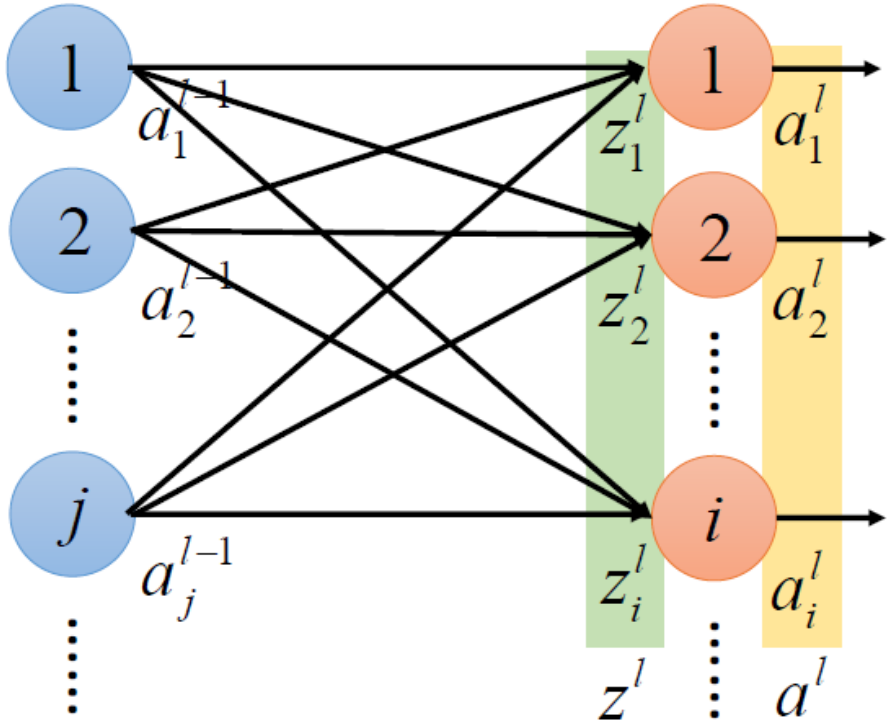
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

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

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

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

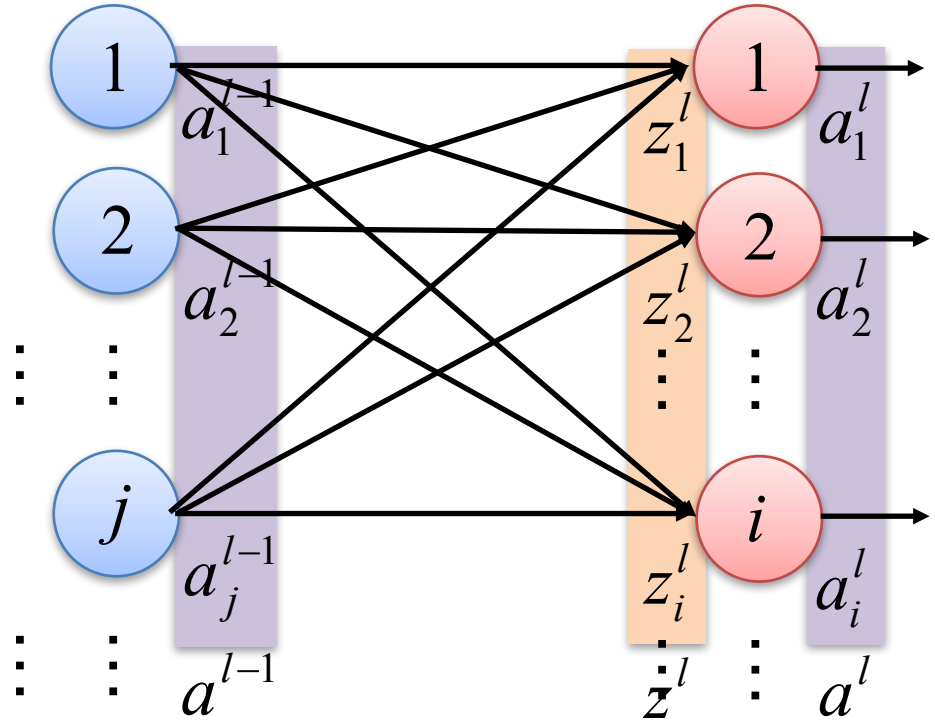
N_l nodes

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

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

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

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

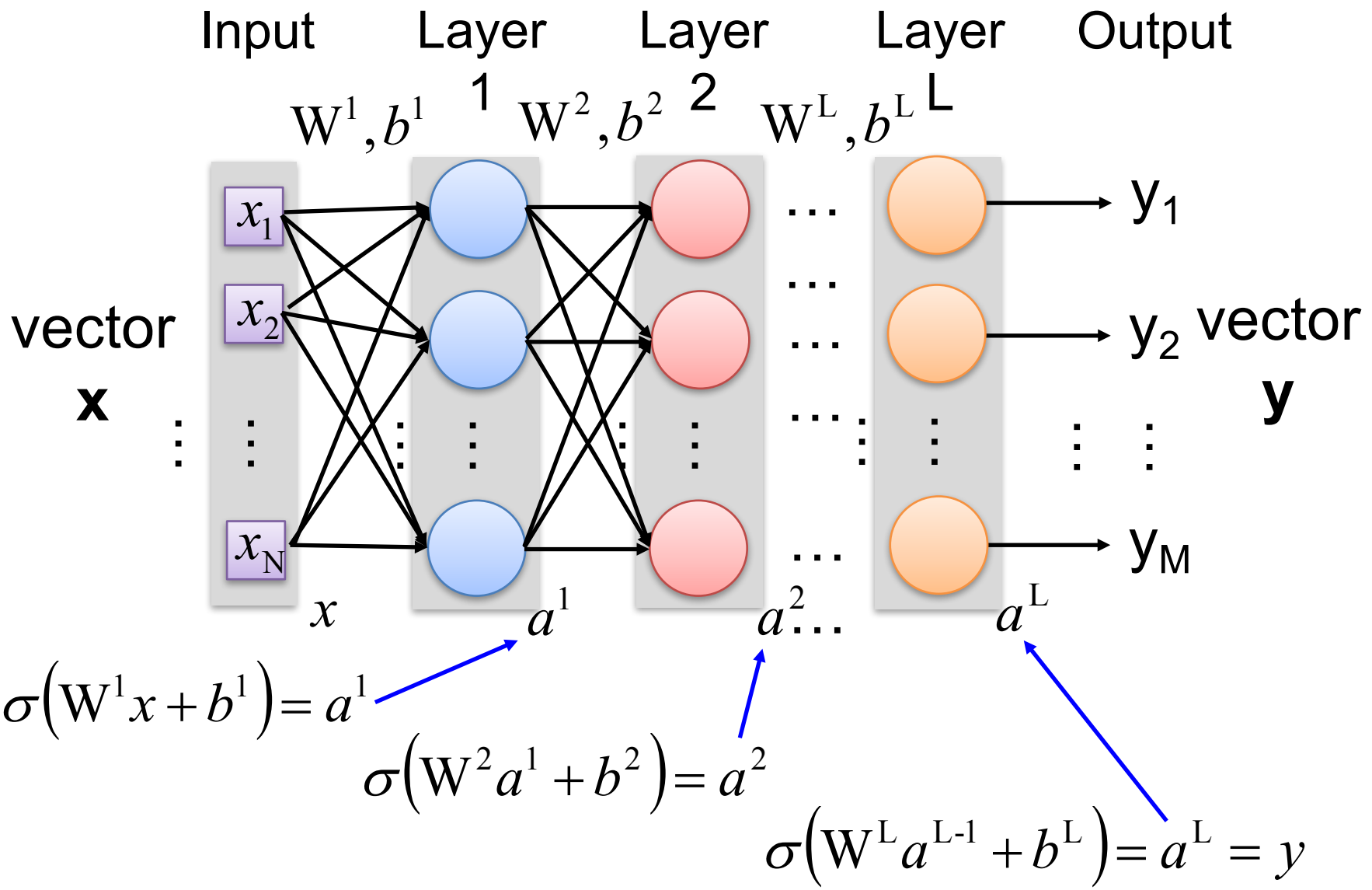
N_l nodes

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

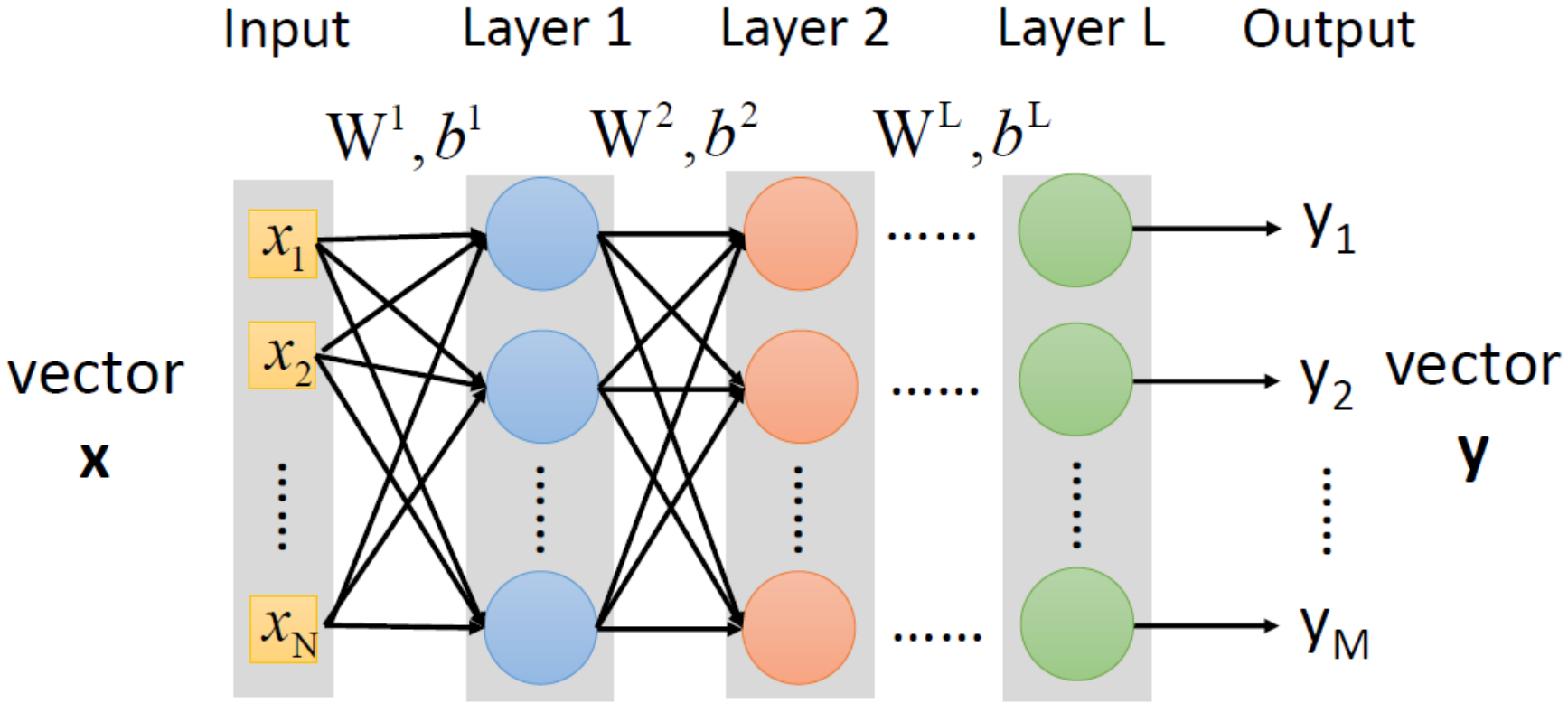
$$a^l = \sigma\left(z^l\right)$$

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

Function of Neural Network



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

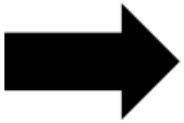
because different parameters W and b lead to different function

Formal way to define a function set:

$f(x;\theta)$ → 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

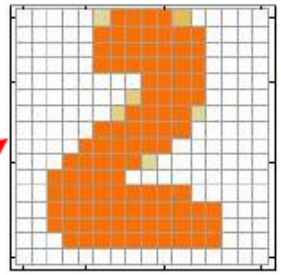
Cost Function

Given training data:

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

- ***Handwriting Digit Classification***

sum over all training examples



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

Minimize distance

$$\begin{bmatrix} 0.1 \\ 0.4 \\ 0.2 \\ \vdots \end{bmatrix} \begin{matrix} \text{“1”} \\ \text{“2”} \\ \text{“3”} \end{matrix}$$

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \text{“1”} \\ \text{“2”} \\ \text{“3”} \end{matrix}$$

3. How to pick the “best” function?

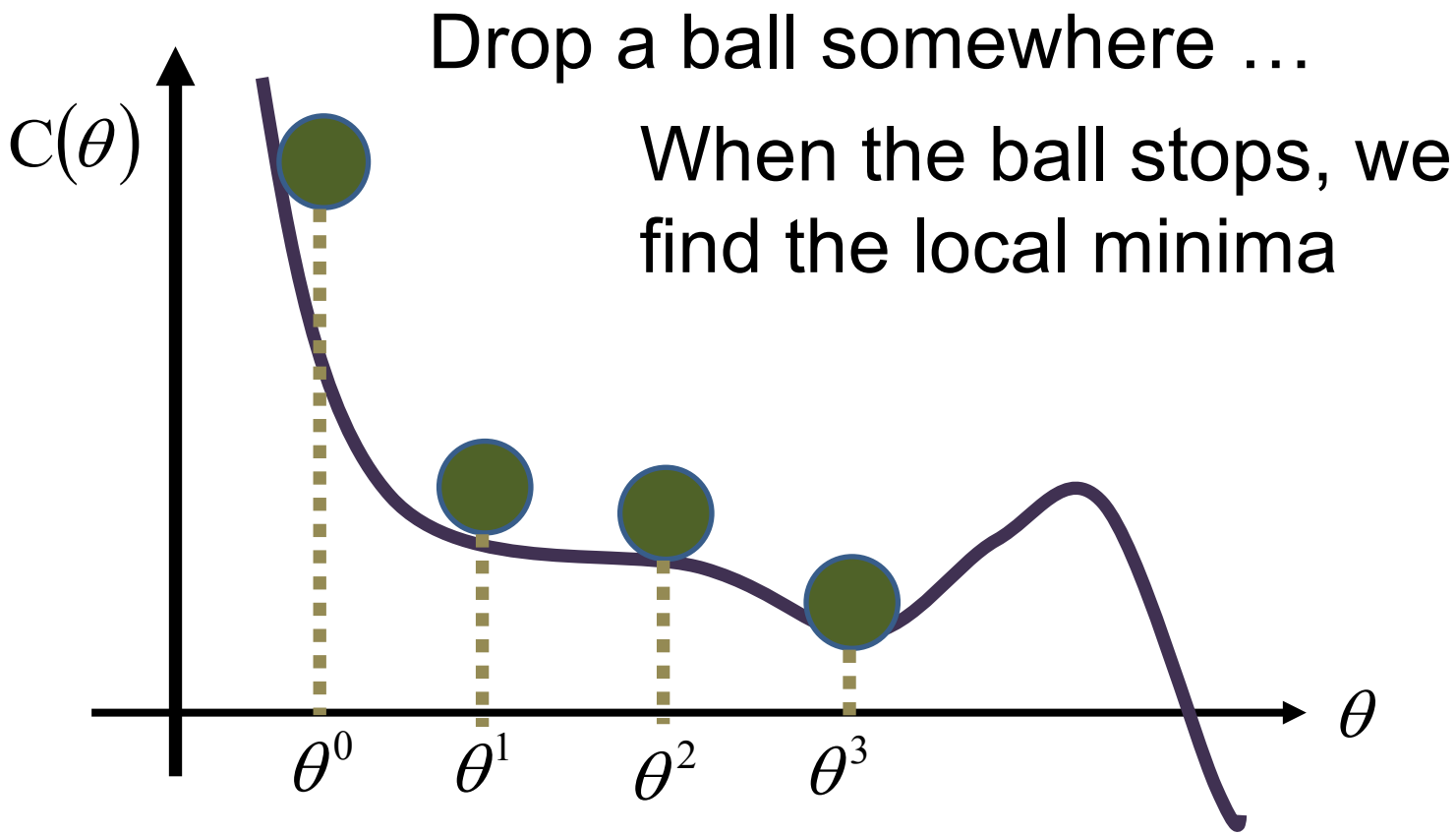
Gradient Descent

Statement of Problems

- Statement of problems:
  - There is a function C(θ)
    - θ represents parameter set
    - $\theta = \{\theta_1, \theta_2, \theta_3, \ldots\ldots\}$
  - Find θ^* that minimizes C(θ)
- 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

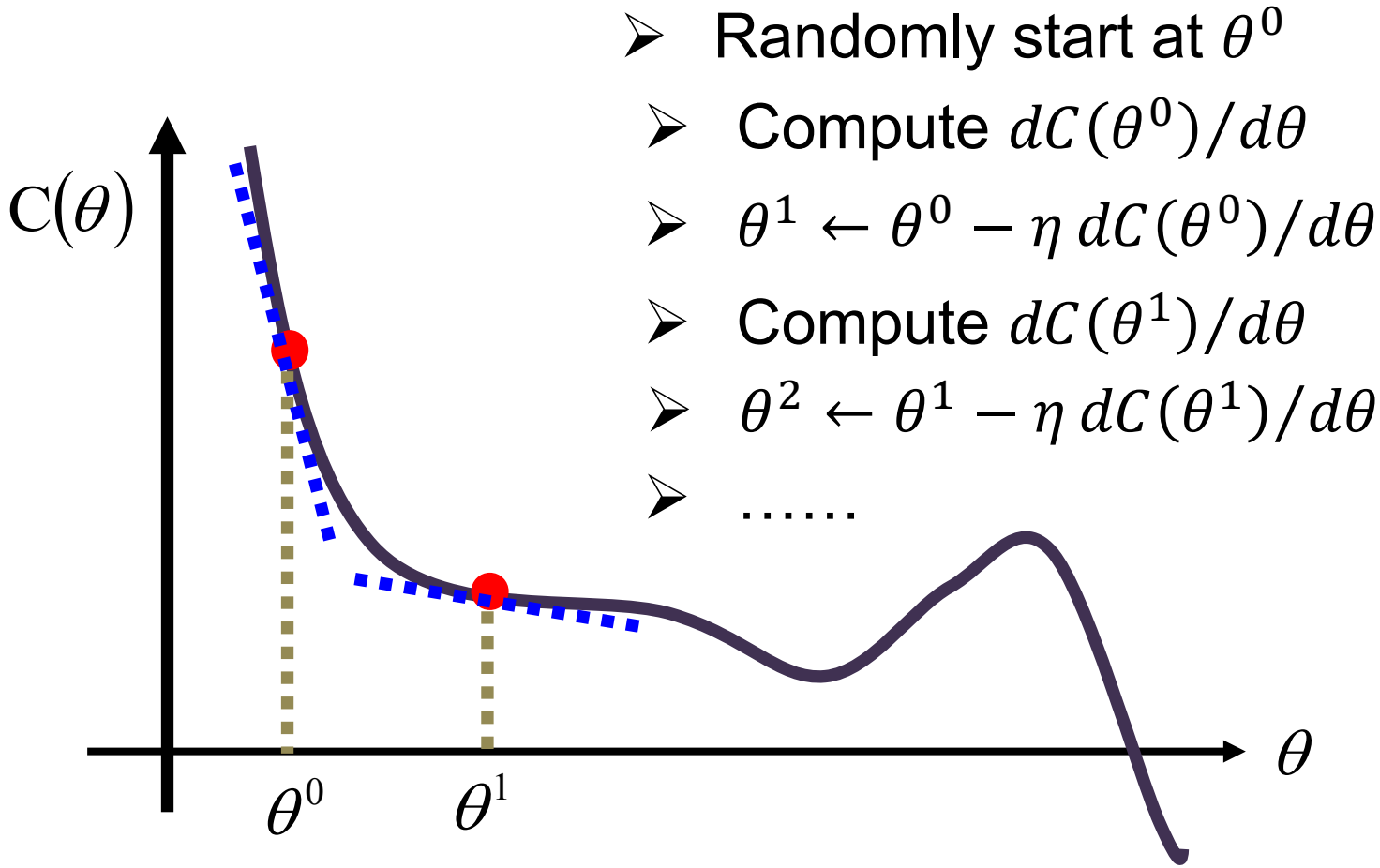
- For simplification, first consider that θ has only one variable



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: $\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta_1^0)/\partial\theta_1 \\ \partial C(\theta_2^0)/\partial\theta_2 \end{bmatrix}$

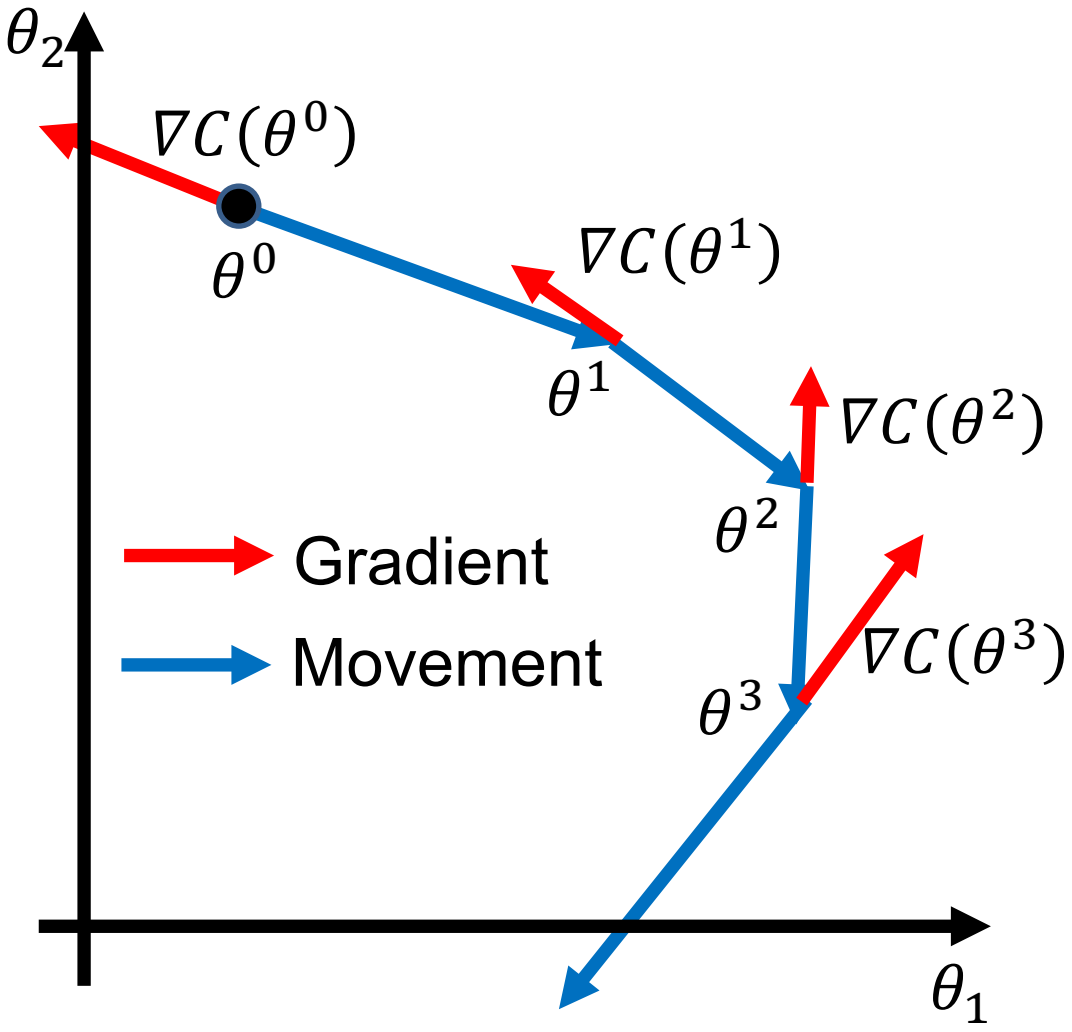
➢ Update parameters

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

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

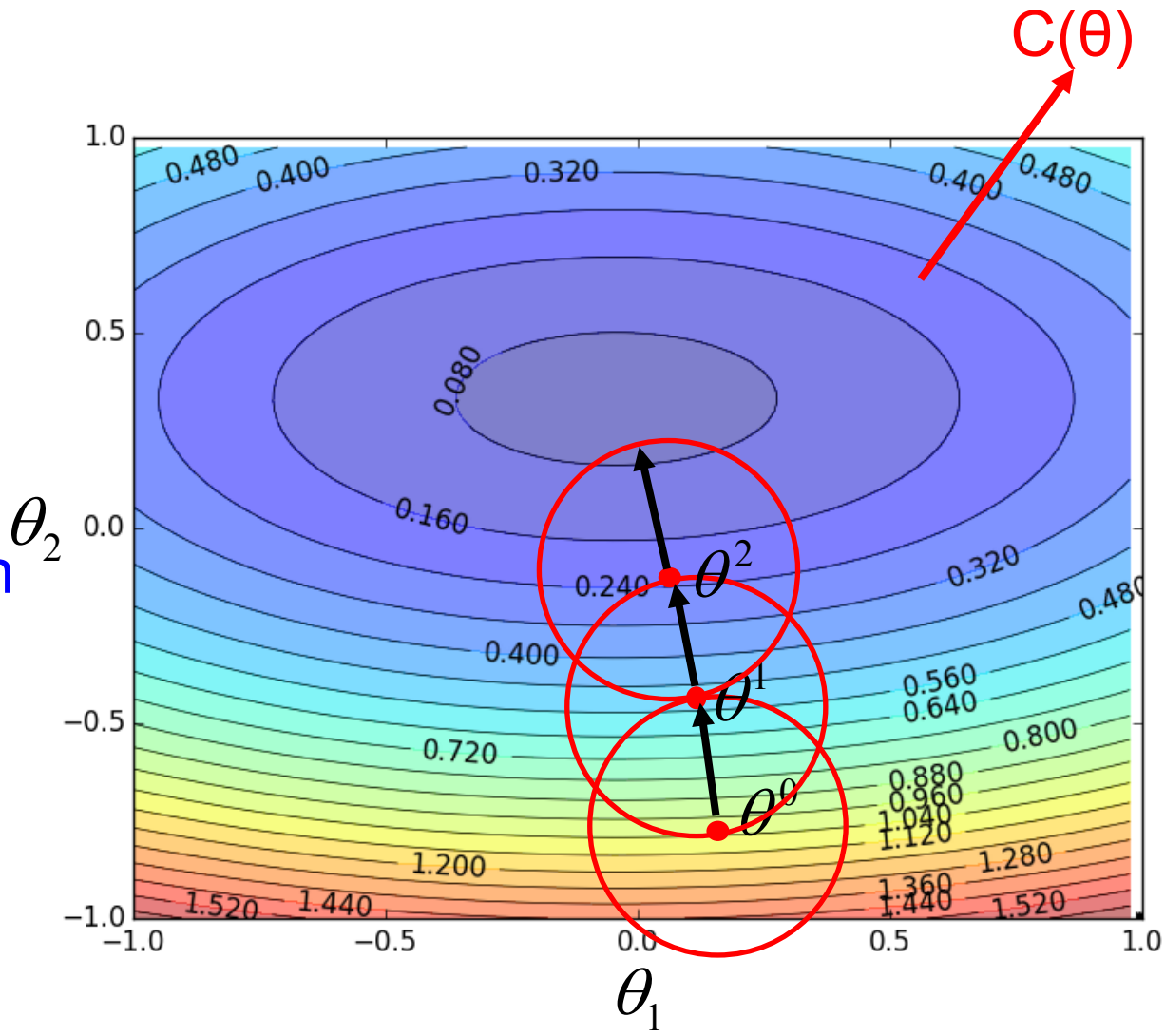
Compute gradient at θ^1

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. How?


80
10
100
200
9/10
黑暗时代
— 村民 已生产 —
蘋果即時

Formal Derivation of Gradient Descent

- **Taylor series**: Let h(x) be infinitely differentiable around $x = x_0$.

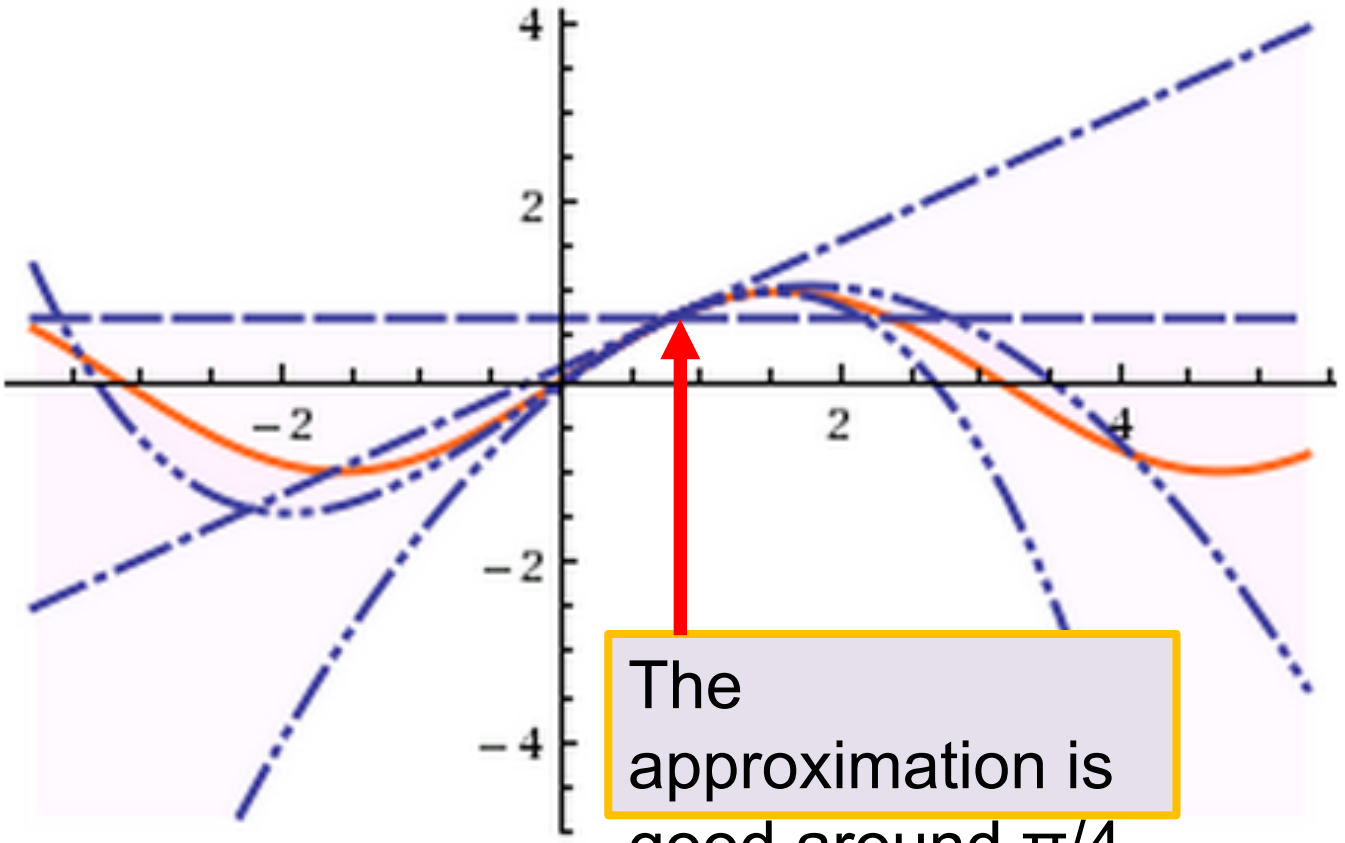
$$h(x) = \sum_{k=0}^{\infty} \frac{h^{(k)}(x_0)}{k!}(x - x_0)^k$$

$$= h(x_0) + h'(x_0)(x - x_0) + \frac{h''(x_0)}{2!}(x - x_0)^2 + \dots$$

When x is close to x_0 $\Rightarrow$ $h(x) \approx h(x_0) + h'(x_0)(x - x_0)$

E.g. Taylor series for h(x)=sin(x) around $x_0=\pi/4$

$$\sin(x)=\frac{1}{\sqrt{2}}+\frac{x-\frac{\pi}{4}}{\sqrt{2}}-\frac{\left(x-\frac{\pi}{4}\right)^2}{2\sqrt{2}}-\frac{\left(x-\frac{\pi}{4}\right)^3}{6\sqrt{2}}+\frac{\left(x-\frac{\pi}{4}\right)^4}{24\sqrt{2}}+\frac{\left(x-\frac{\pi}{4}\right)^5}{120\sqrt{2}}-\frac{\left(x-\frac{\pi}{4}\right)^6}{720\sqrt{2}}-$$
$$\frac{\left(x-\frac{\pi}{4}\right)^7}{5040\sqrt{2}}+\frac{\left(x-\frac{\pi}{4}\right)^8}{40\,320\sqrt{2}}+\frac{\left(x-\frac{\pi}{4}\right)^9}{362\,880\sqrt{2}}-\frac{\left(x-\frac{\pi}{4}\right)^{10}}{3\,628\,800\sqrt{2}}+\cdots\cdots$$



The approximation is good around π/4.

Multivariable Taylor series

$$h(x,y)=h(x_0,y_0)+\frac{\partial h(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$$

+ something related to $(x-x_0)^2$ and $(y-y_0)^2$

+ ……

When x and y is close to x_0 and y_0

$$h(x,y)\approx h(x_0,y_0)+\frac{\partial h(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$$

Formal Derivation of Gradient Descent

Based on Taylor Series:

If the red circle is ***small enough***, in the red circle

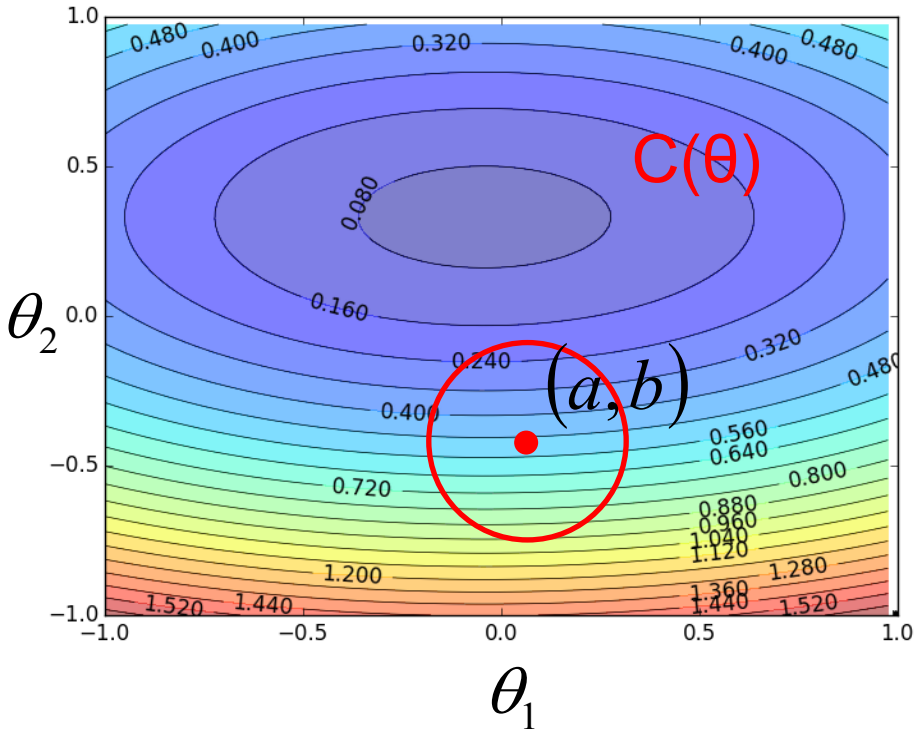
$$C(\theta) \approx C(a,b) + \frac{\partial C(a,b)}{\partial \theta_1}(\theta_1 - a) + \frac{\partial C(a,b)}{\partial \theta_2}(\theta_2 - b)$$

$$s = C(,b)$$

$$u = \frac{\partial C(a,b)}{\partial \theta_1}, v = \frac{\partial C(a,b)}{\partial \theta_2}$$

$$C(\theta)$$

$$\approx s + u(\theta_1 - a) + v(\theta_2 - b)$$



Formal Derivation of Gradient Descent

$$u=\frac{\partial C(a,b)}{\partial \theta_1}, v=\frac{\partial C(a,b)}{\partial \theta_2}$$

Based on Taylor Series:

If the red circle is ***small enough***, in the red circle

$$C(\theta)\approx s+u(\theta_1-a)+v(\theta_2-b)$$

Find θ_1 and θ_2 yielding the smallest value of $C(\theta)$ in the circle

$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}=\begin{bmatrix}a\\ b\end{bmatrix}-\eta\begin{bmatrix}u\\ v\end{bmatrix}=\begin{bmatrix}a\\ b\end{bmatrix}-\eta\begin{bmatrix}\frac{\partial C(a,b)}{\partial \theta_1}\\ \frac{\partial C(a,b)}{\partial \theta_2}\end{bmatrix}=\begin{bmatrix}a\\ b\end{bmatrix}-\eta\nabla C(a,b)$$

Its value depending on the radius of the circle, u and v.

This is how gradient descent updates parameters.

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

$$\nabla C(\theta) \qquad \theta = \left\{ W^1, b^1, W^2, b^2, \cdots, W^l, b^l, \cdots, W^L, b^L \right\}$$

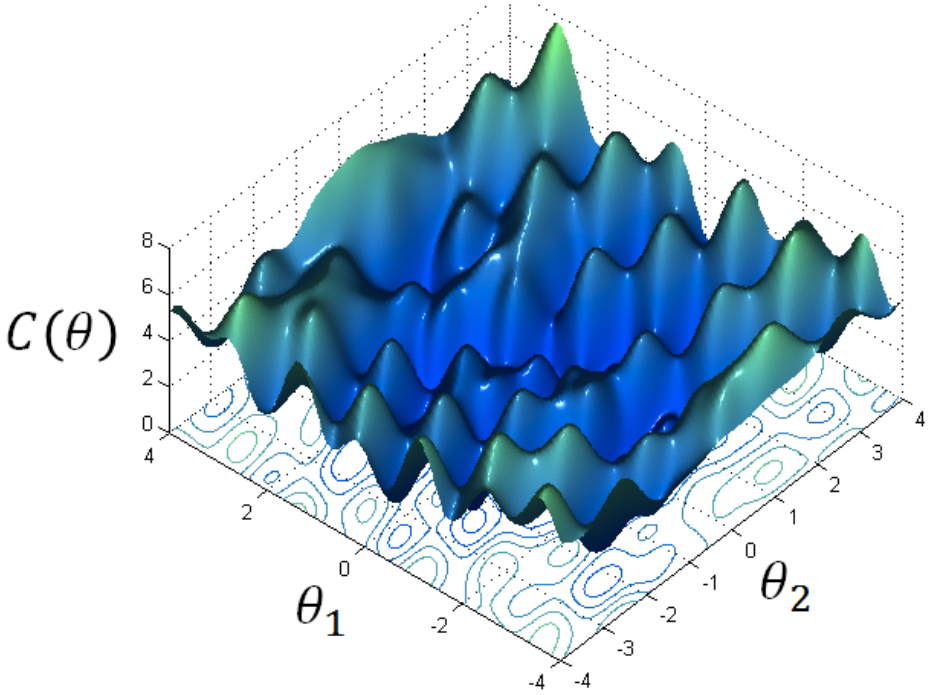
$$= \begin{bmatrix} \vdots \\ \dfrac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \dfrac{\partial C(\theta)}{\partial b_i^l} \\ \vdots \end{bmatrix}$$

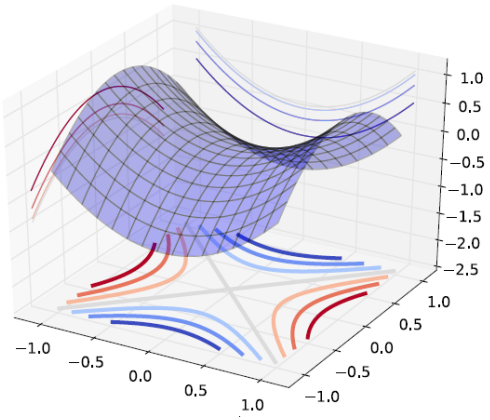
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \\ & & \end{bmatrix} \qquad b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

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

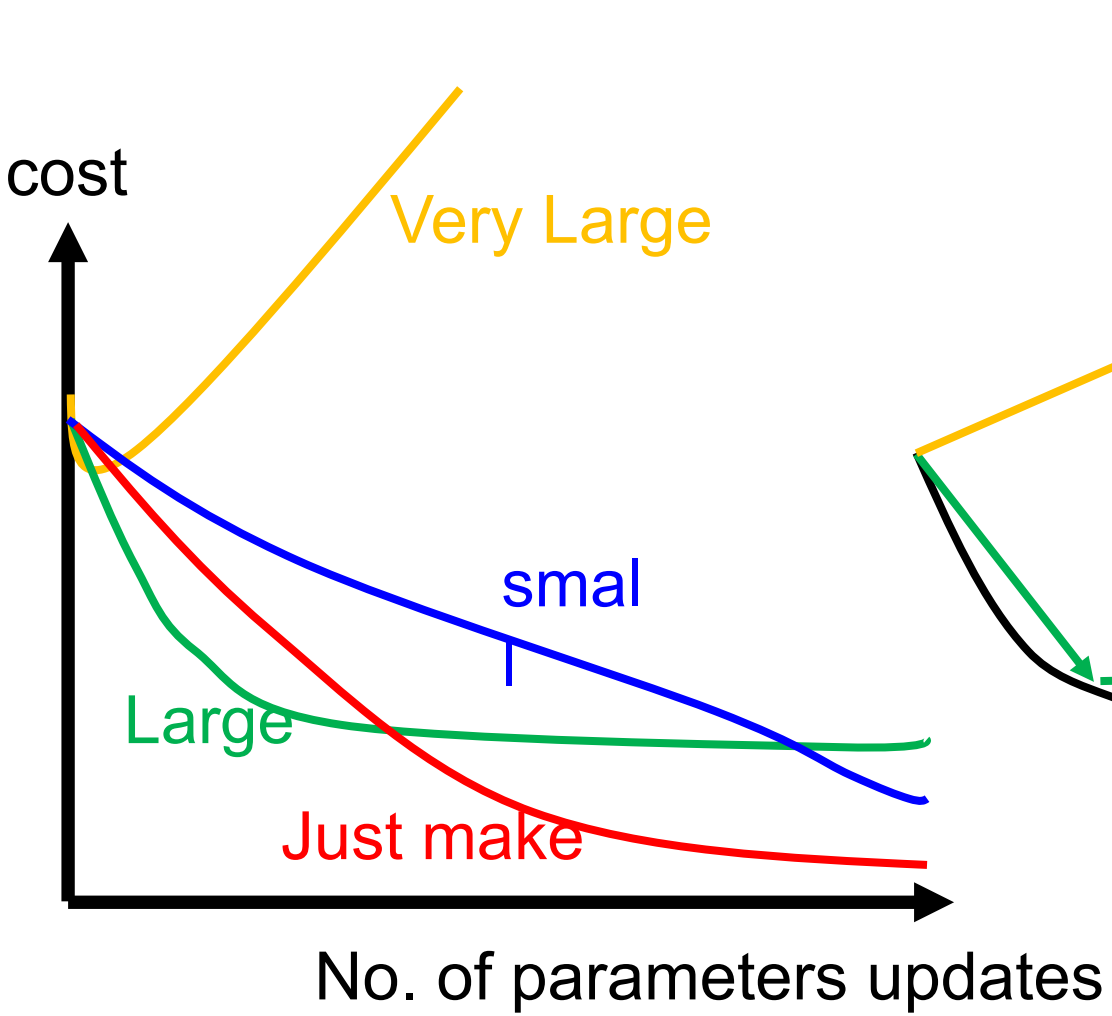
Parameter Initialization

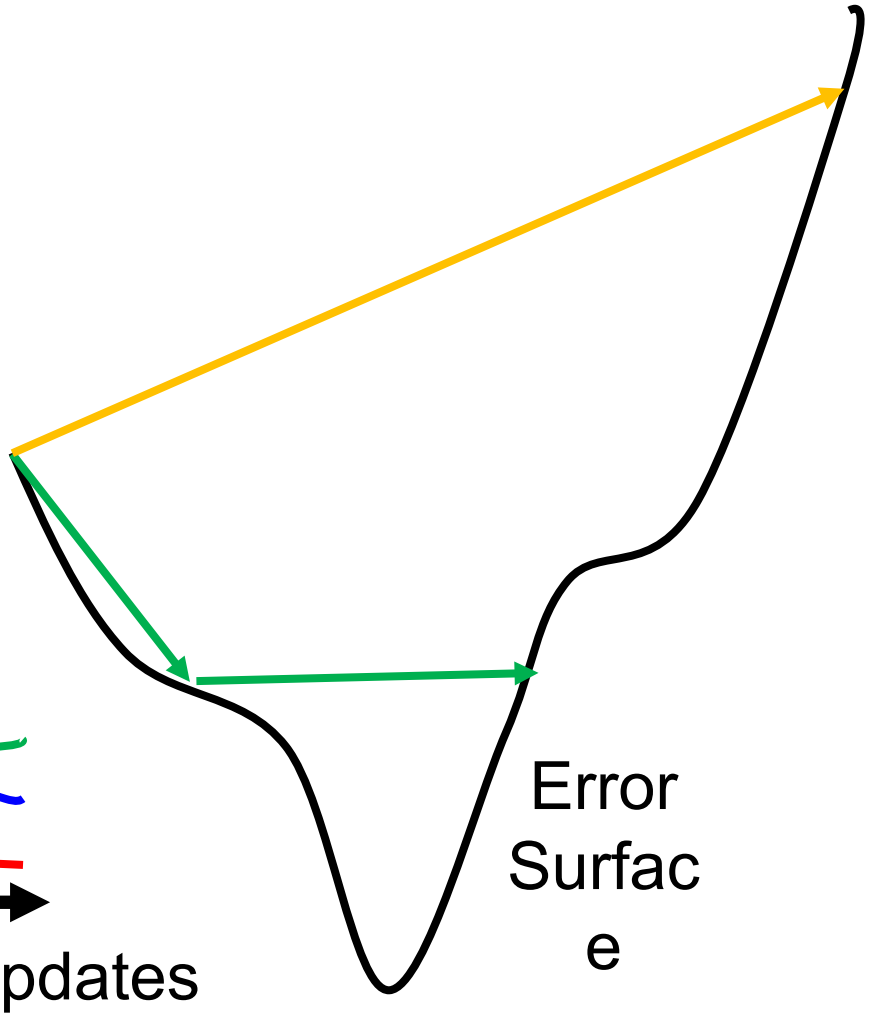
- For gradient Descent, we need to pick an initialization parameter θ^0.
- The initialization parameters have some influence to the training.
  - We will go back to this issue in the future.
- Suggestion today:
  - Do not set all the parameters θ^0 equal
  - Set the parameters in θ^0 randomly

Learning Rate

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

- Set the learning rate η carefully



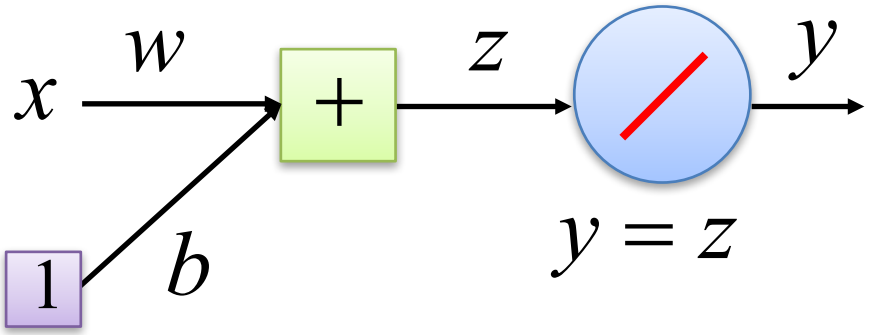


Learning Rate

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

- Set the learning rate η carefully

- ***Toy Example***



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

Training Data (20 examples)

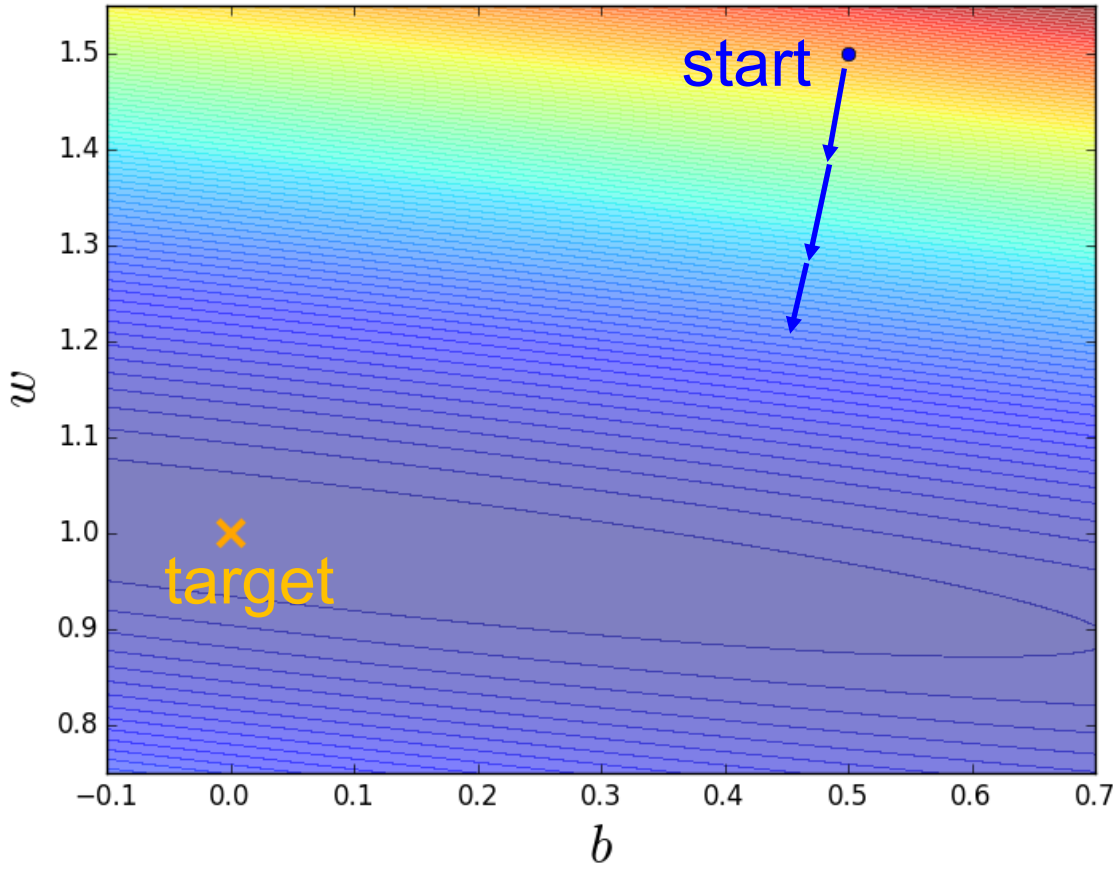
x = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5]
y = [0.1, 0.4, 0.9, 1.6, 2.2, 2.5, 2.8, 3.5, 3.9, 4.7, 5.1, 5.3, 6.3, 6.5, 6.7, 7.5, 8.1, 8.5, 8.9, 9.5]

Learning Rate

- ***Toy Example***

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

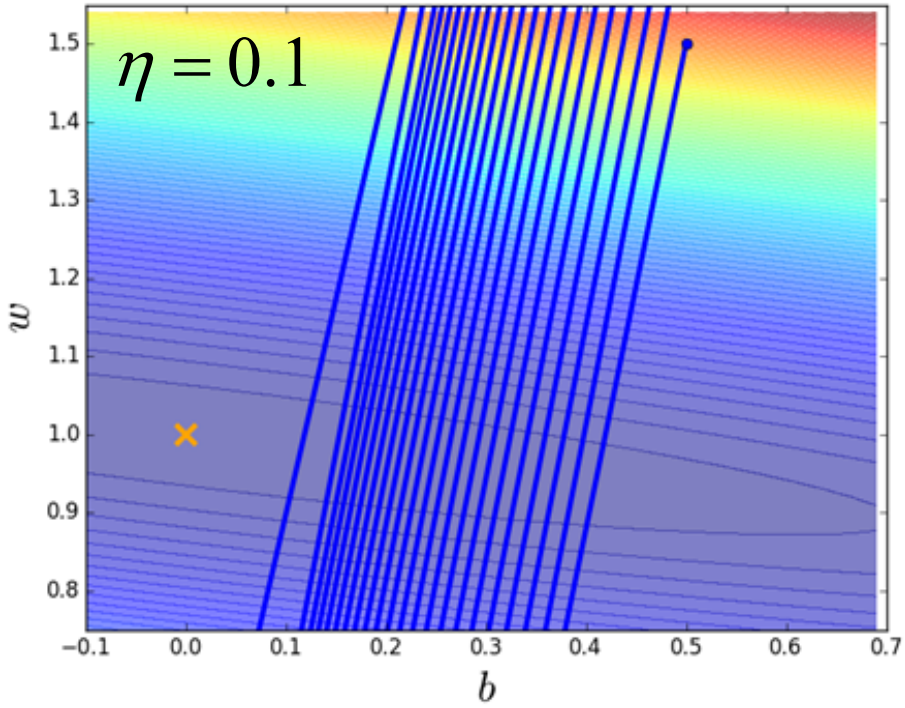
Error Surface: C(w,b)

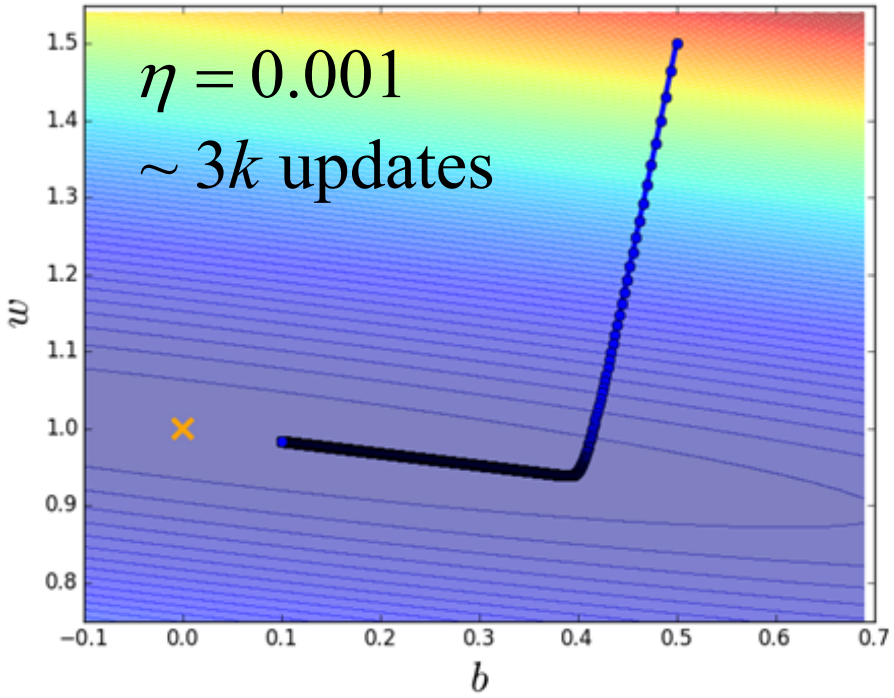


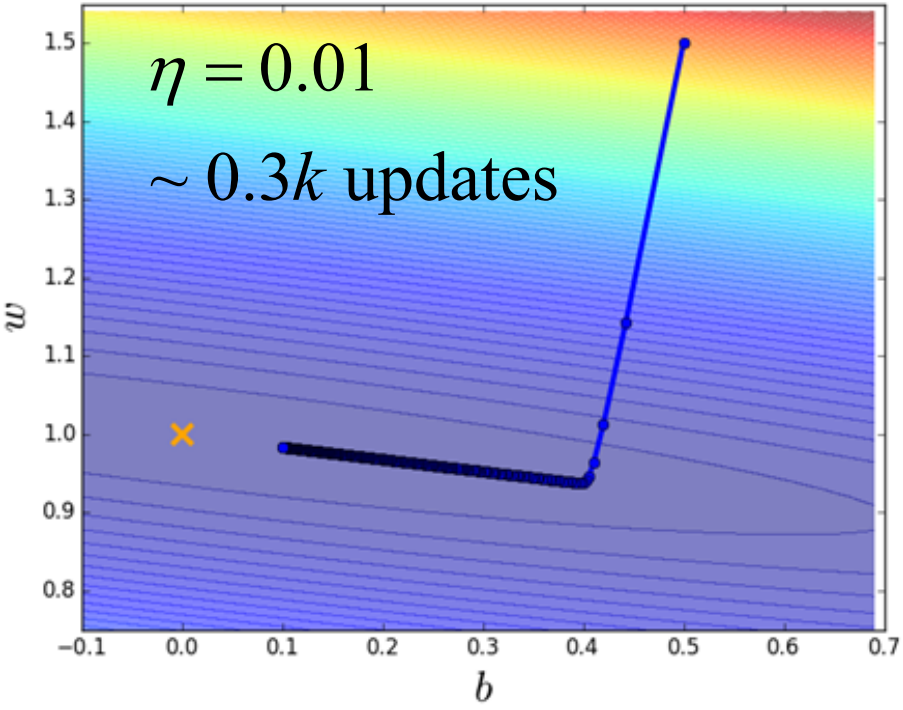
Learning Rate

- ***Toy Example***

Different learning rate η

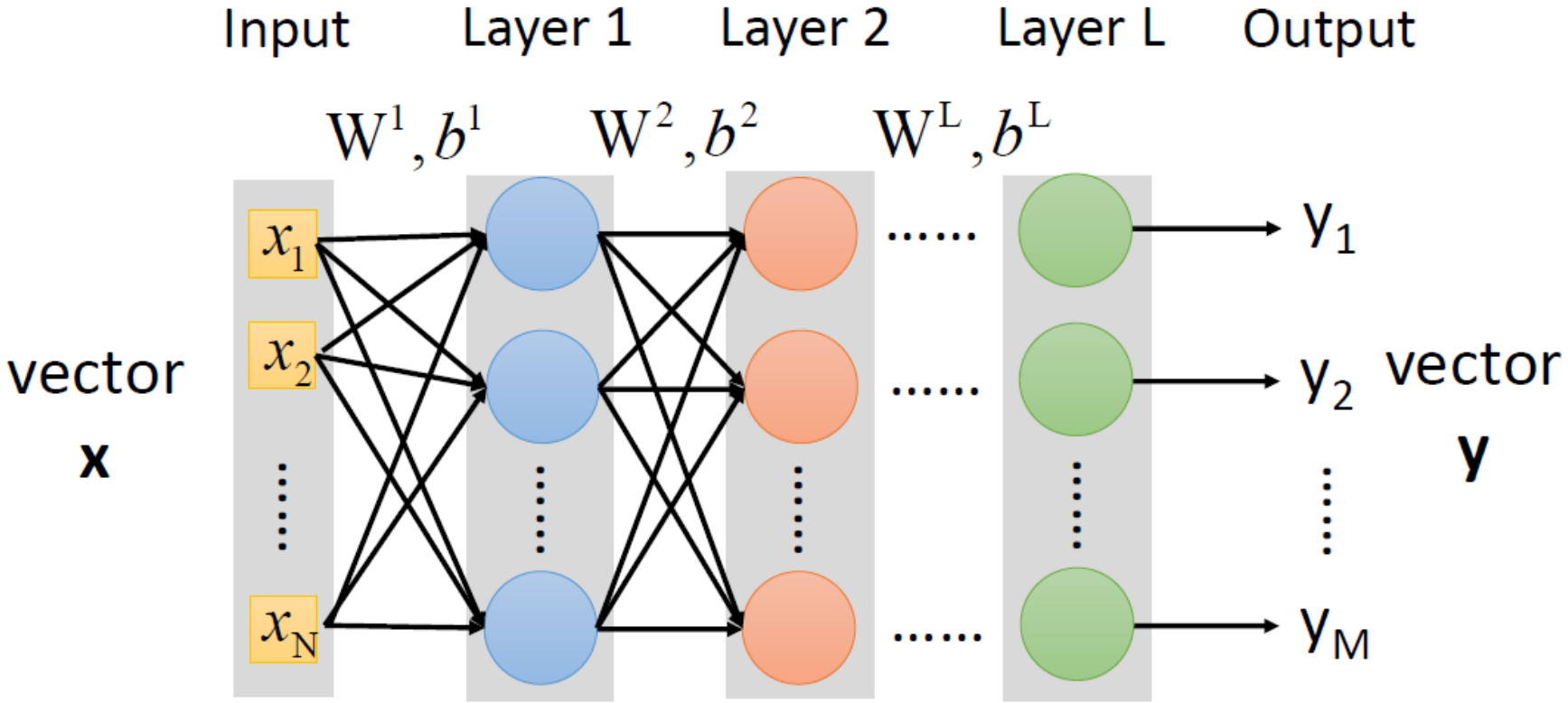






Quick Review

Function of Neural Network



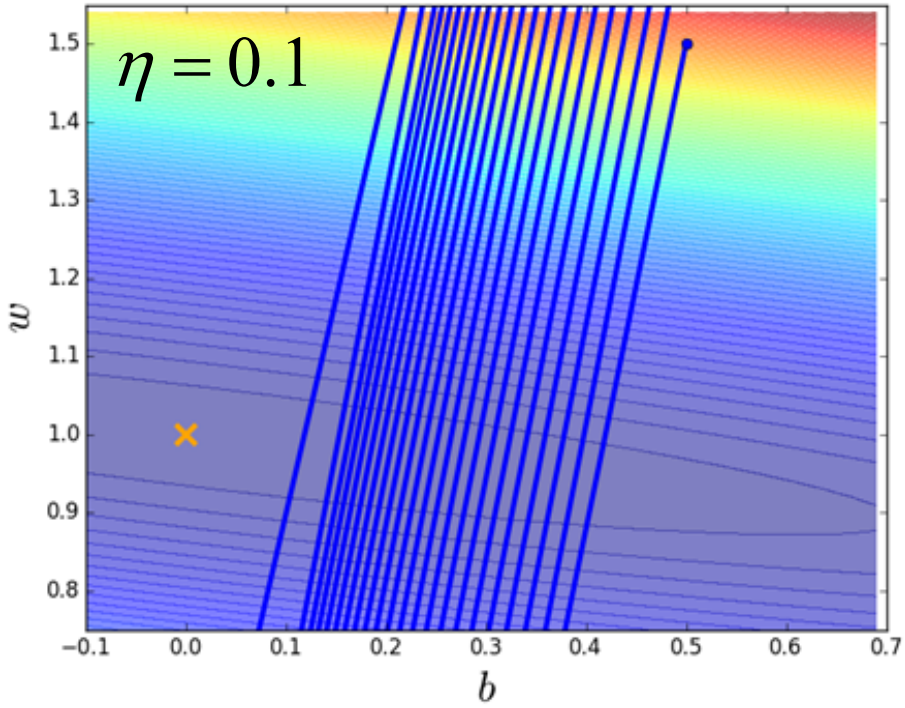
$$y = f(x)$$

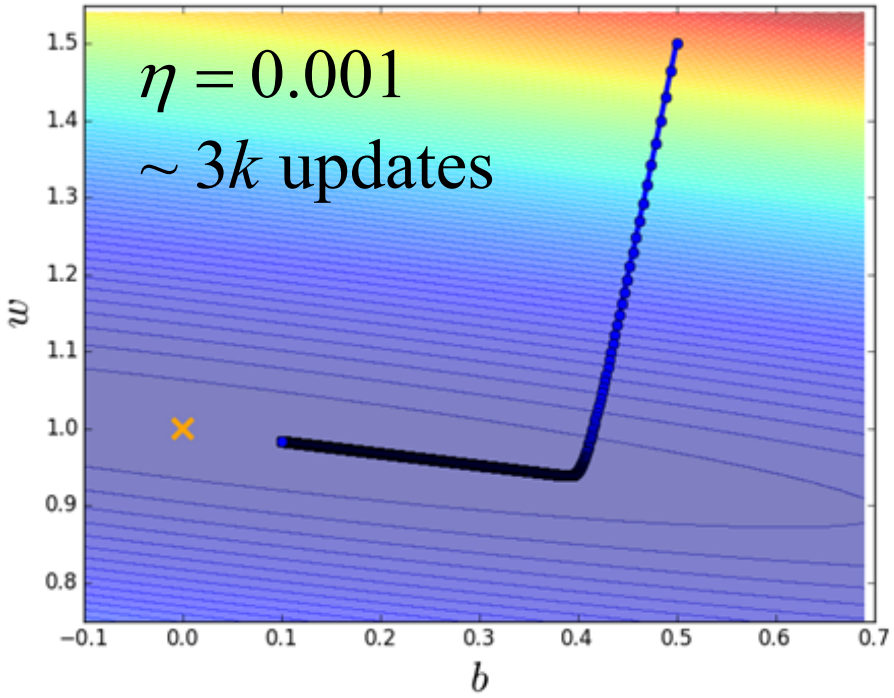
$$= \sigma\left(W^L \ldots \sigma\left(W^2 \sigma\left(W^1 x + b^1\right) + b^2\right) \ldots + b^L\right)$$

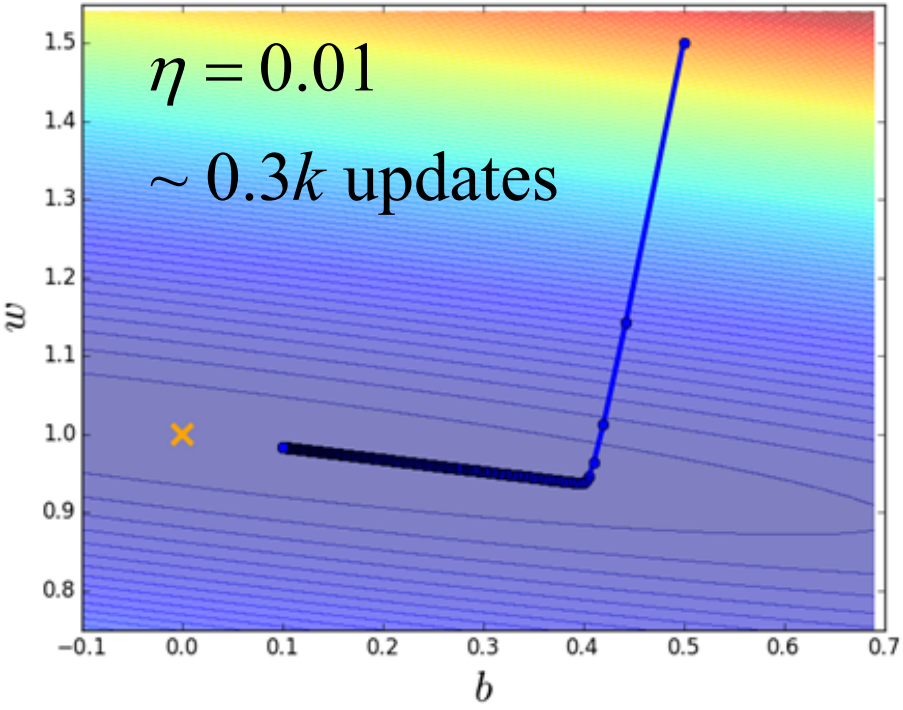
Learning Rate

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

$\vdots$ $\qquad\qquad \vdots$

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

$\vdots$ $\qquad\qquad \vdots$

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

---

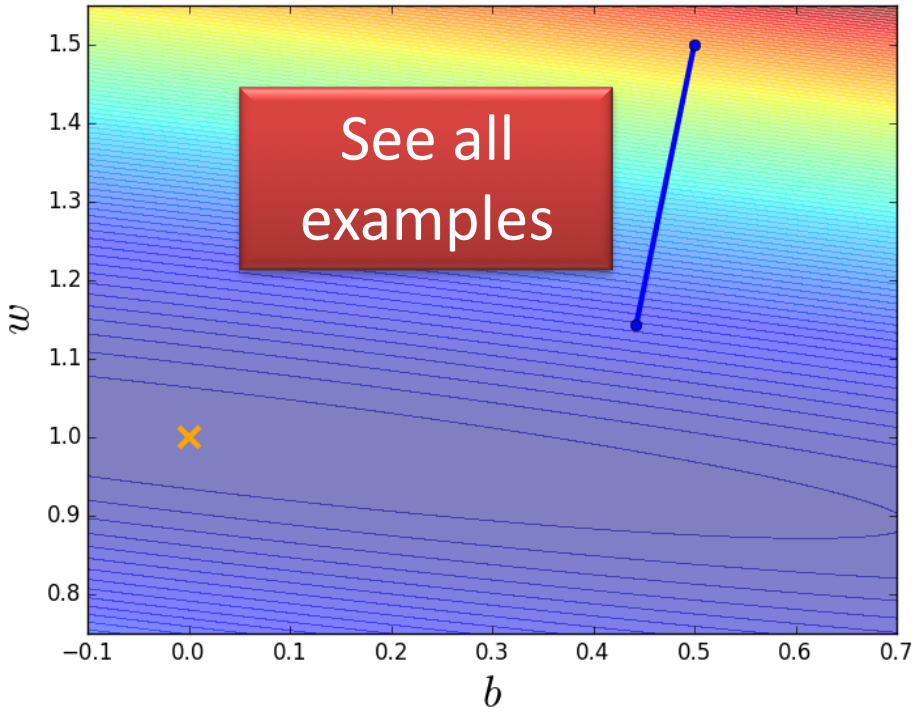
pick x^1 $\quad \theta^{R+1} = \theta^R - \eta \nabla C^1\left(\theta^R\right)$

Stochastic Gradient Descent and Mini-batch

- ***Toy Example***

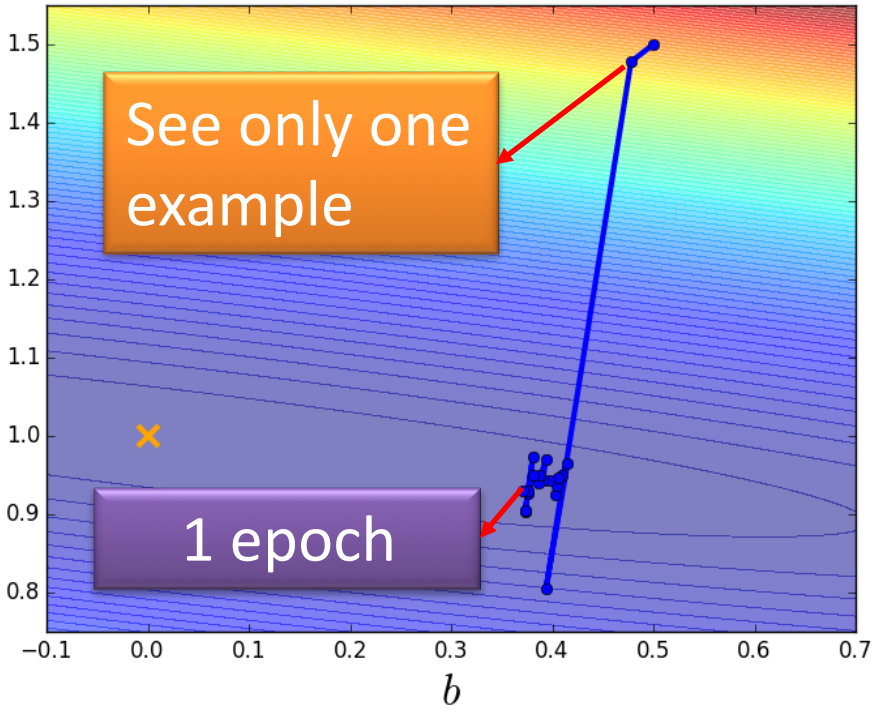
Gradient Descent

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

◆ ***Mini Batch Gradient Descent***

Pick B examples as a batch b

B is batch size

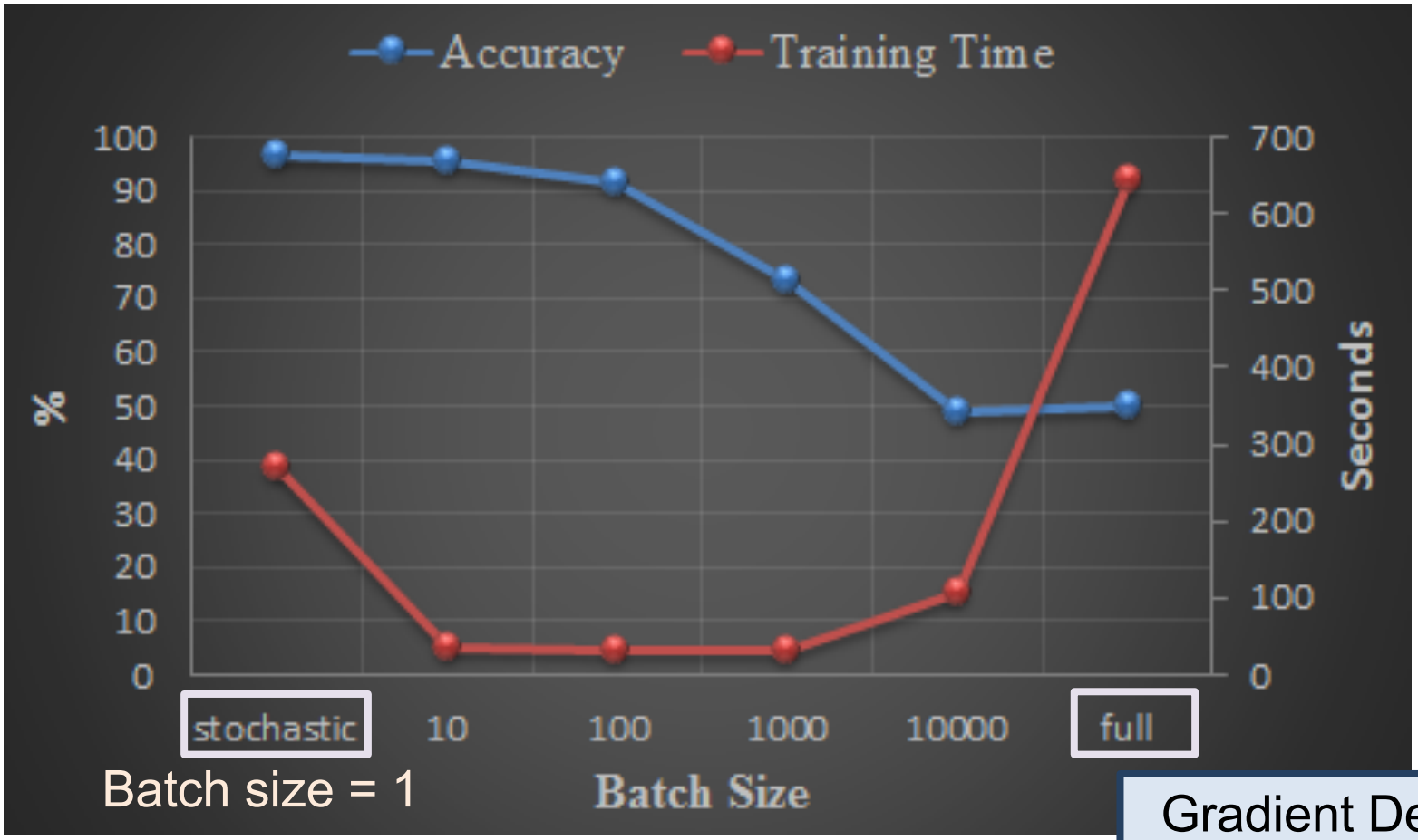
Shuffle your data

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

Average the gradient of the examples in the batch b

Stochastic Gradient Descent and Mini-batch

- ***Handwriting Digit Classification***

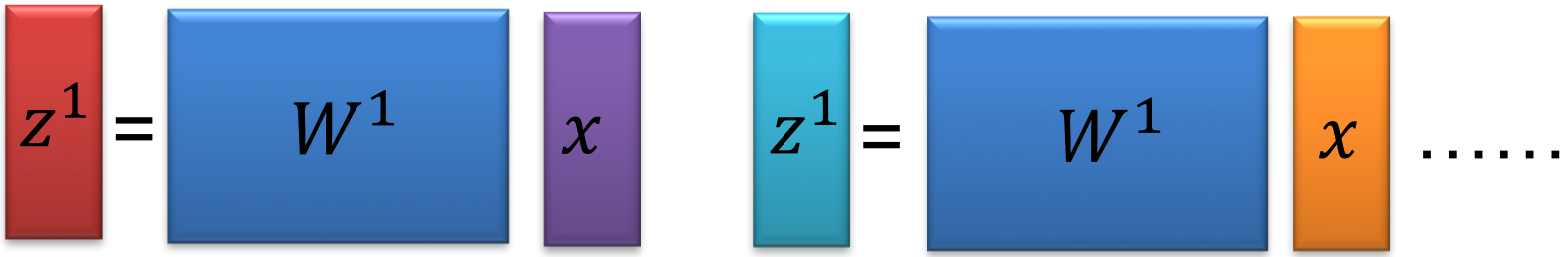


Gradient Descent

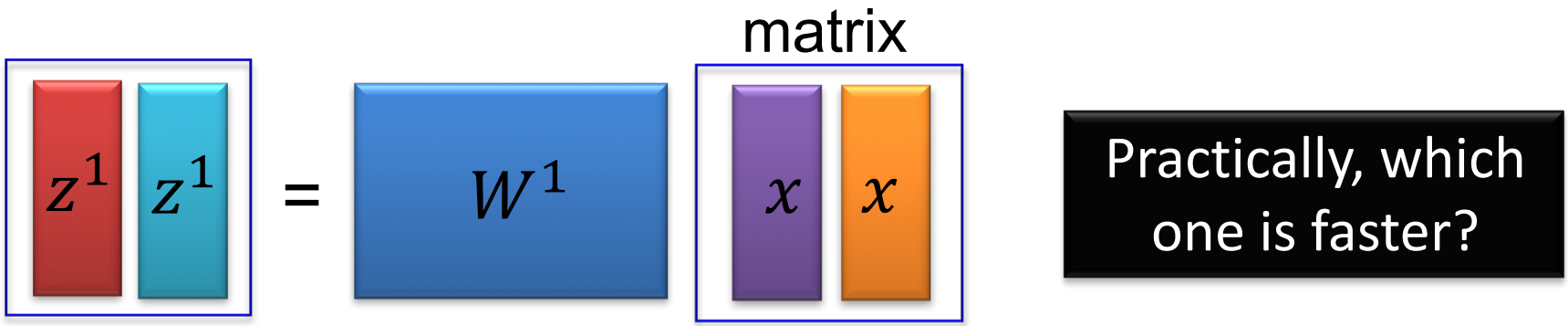
Stochastic Gradient Descent and Mini-batch

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

Stochastic Gradient Descent



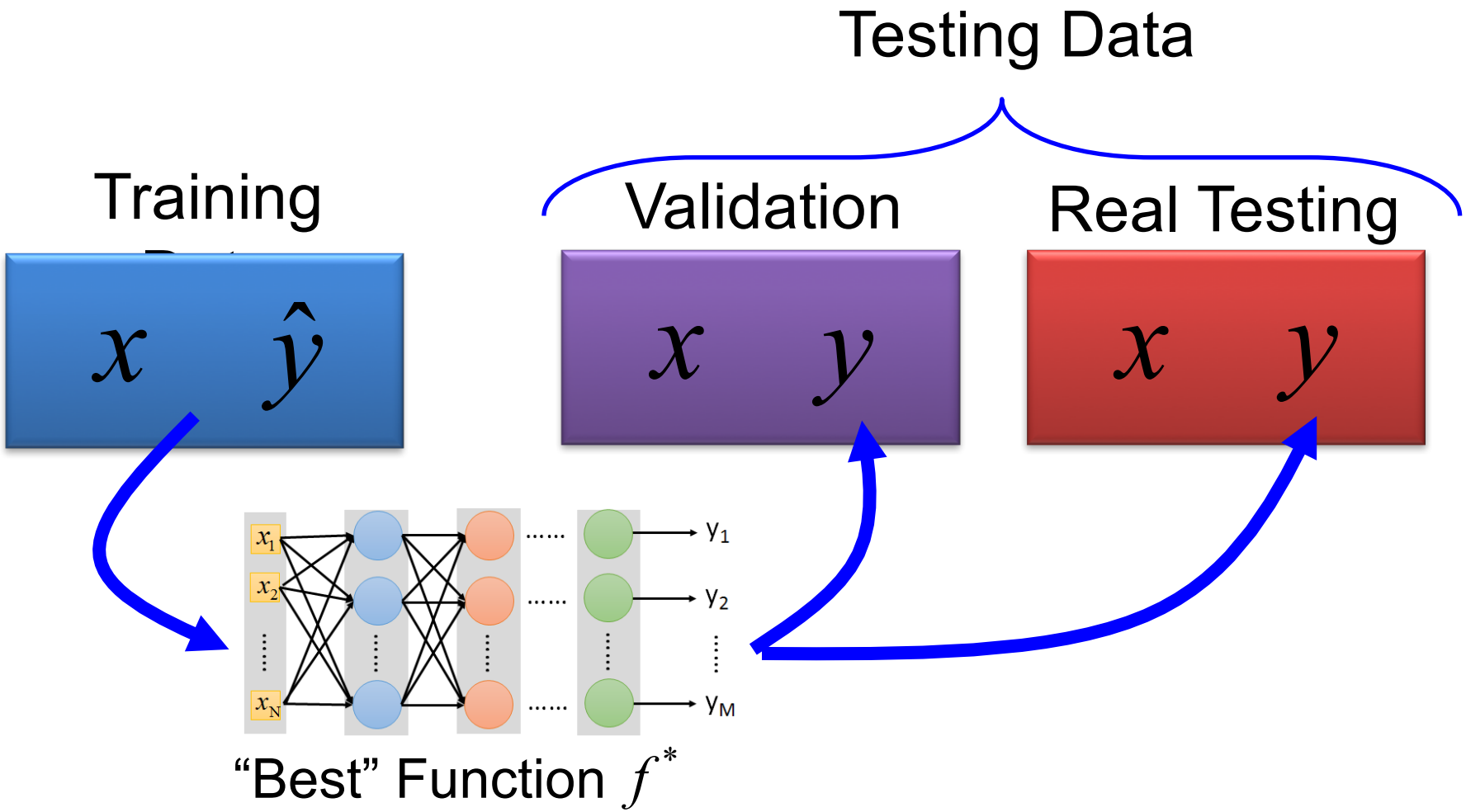
Mini-batch



Practically, which one is faster?

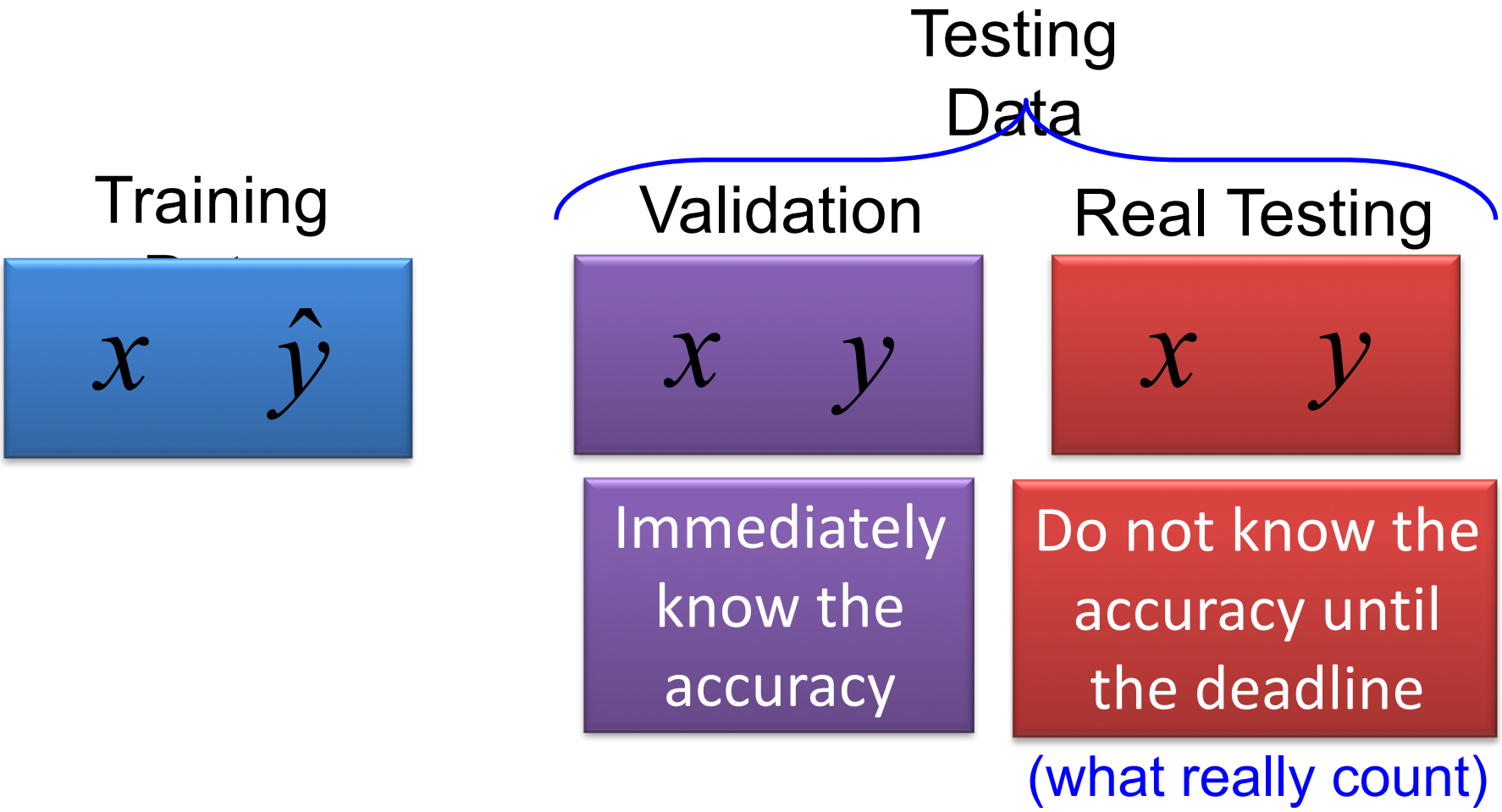
Recipe for Learning

- Data provided in homework



Recipe for Learning

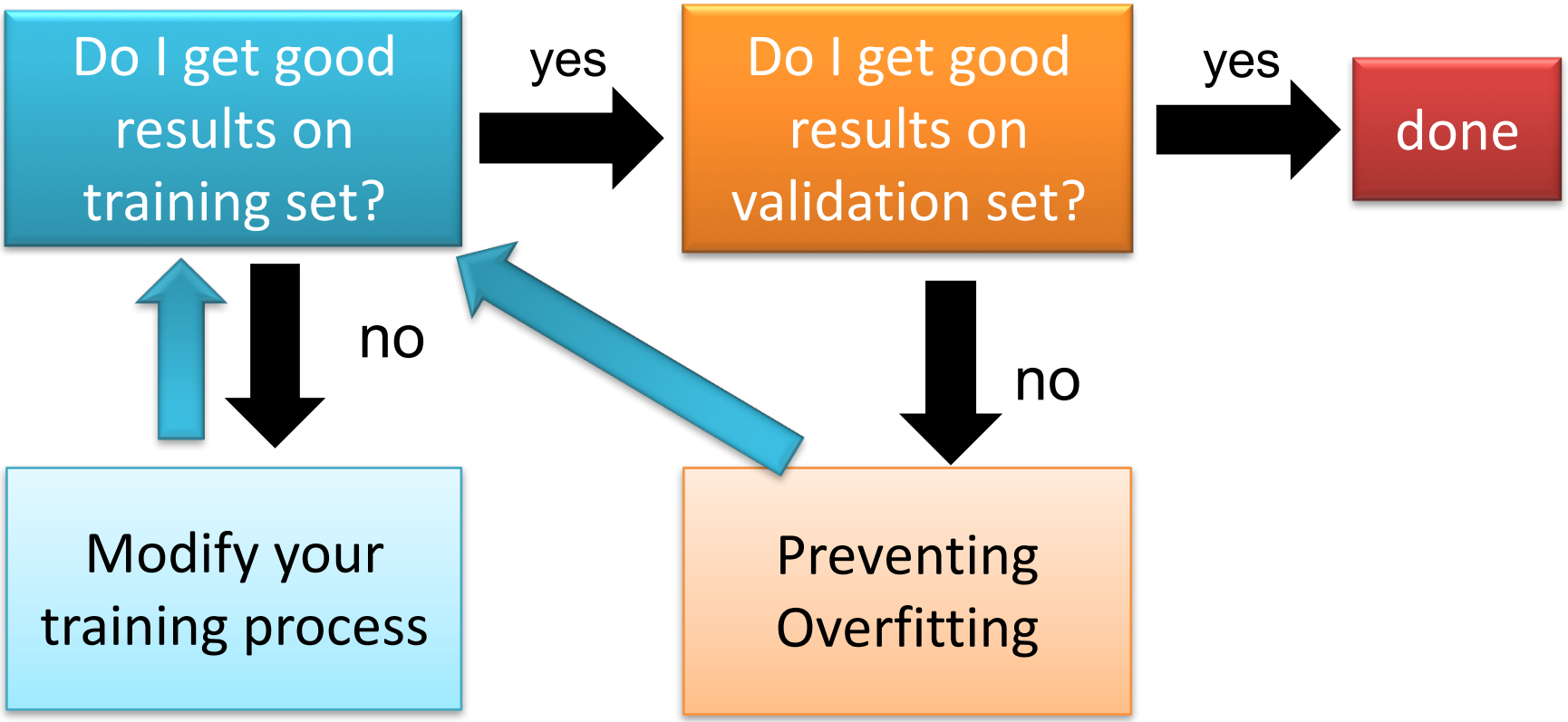
- Data provided in homework



Recipe for Learning

- Your code has bug.
- Can not find a good function
  - Stuck at local minima, saddle points ….
  - Change the training strategy
- Bad model
  - There is no good function in the hypothesis function set.
  - Probably you need bigger network

Recipe for Learning



- Your code usually do not have bug at this situation.

Recipe for Learning - Overfitting

- You pick a "best" parameter set θ^*

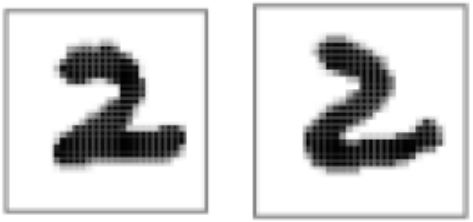
Training Data: $\{\ldots(x^r, \hat{y}^r)\ldots\} \Rightarrow \forall r : f(x^r; \theta^*) = \hat{y}^r$

However,

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

Training data and testing data have different distribution.

Training Data:

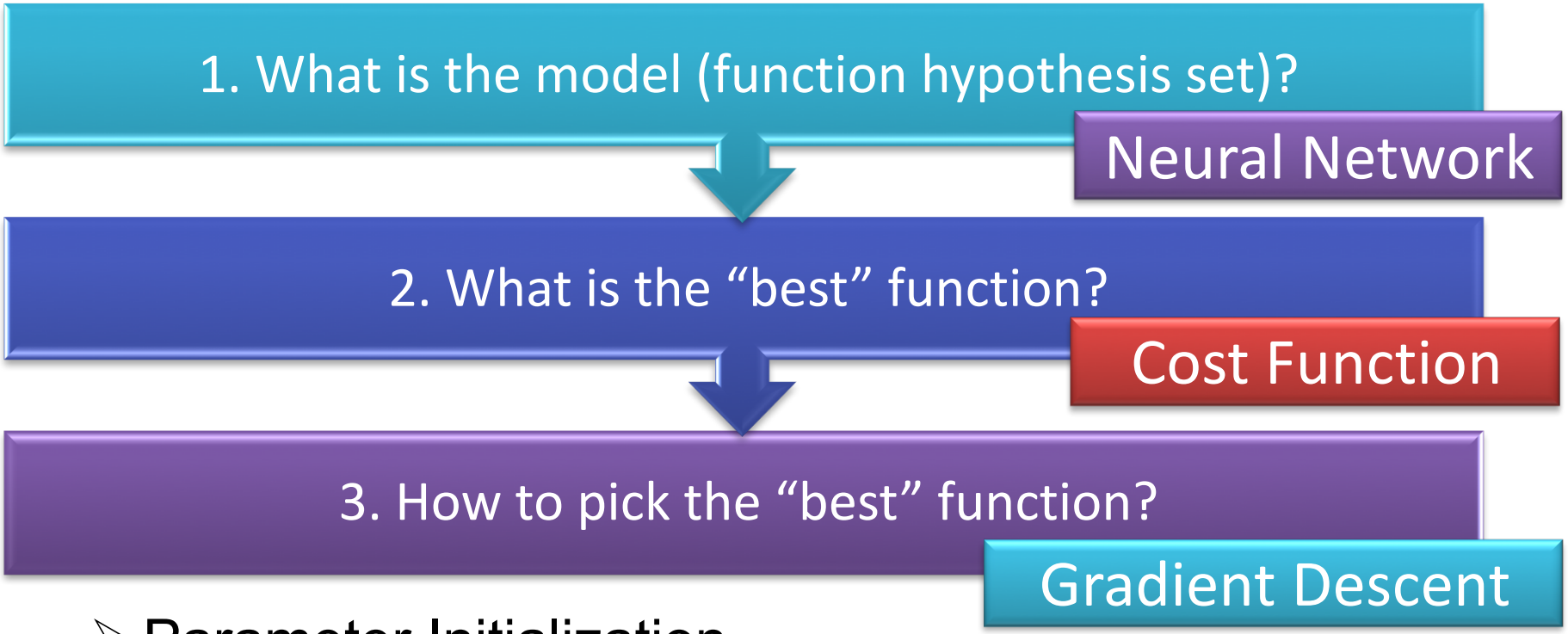


Testing Data:

Recipe for Learning - Overfitting

- Panacea: Have more training data
  - You can do that in real application, but you can't do that in homework.
- We will go back to this issue in the future.

Concluding Remarks



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

感謝 沈昇勳 同學提供圖檔

使用 Keras 心得

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

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

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

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

Millions of parameters ……

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

Background

- Cost Function $C(\theta)$
  - Given training examples: $\{(x^1, \hat{y}^1), \cdots, (x^r, \hat{y}^r), \cdots, (x^R, \hat{y}^R)\}$
  - Find a set of parameters θ^* minimizing C(θ)
    - $C(\theta) = \frac{1}{R}\sum_r C^r(\theta), C^r(\theta) = \|f(x^r;\theta) - \hat{y}^r\|$
- Gradient Descent
  - $\nabla C(\theta) = \frac{1}{R}\sum_r \nabla C^r(\theta)$
  - Given w_{ij}^l and b_i^l, we have to compute $\partial C^r/\partial w_{ij}^l$ and $\partial C^r/\partial b_i^l$
- There is an efficient way to compute the gradients of the network parameters – ***backpropagation***.

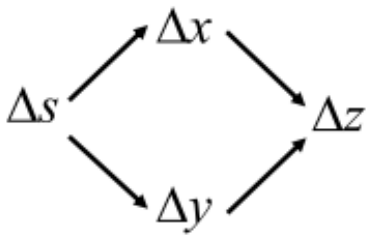
Chain Rule

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

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

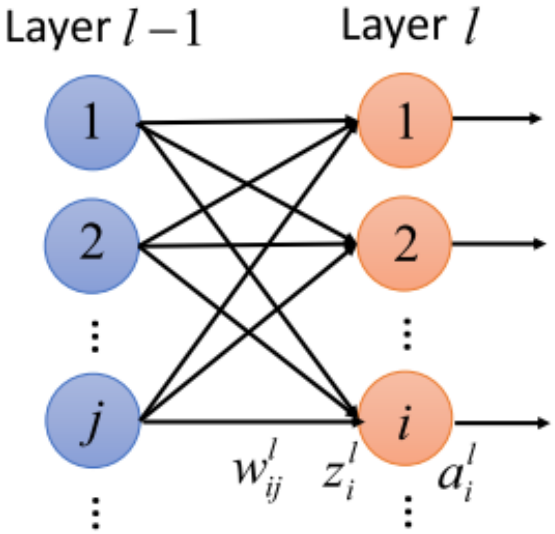
Case 2

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



$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

$\partial C^r / \partial w_{ij}^l$

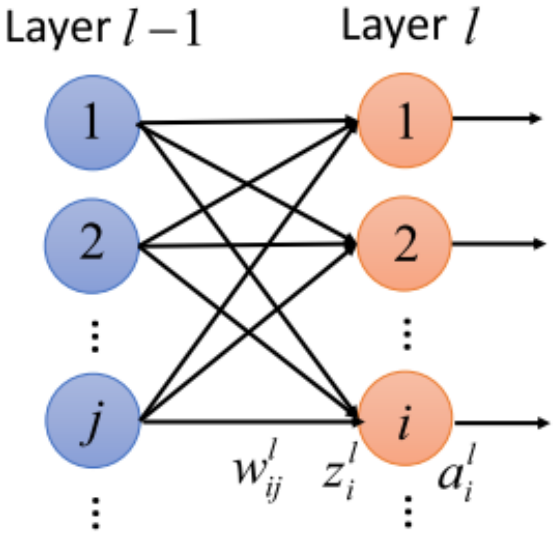


$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \cdots\cdots \rightarrow \Delta C^r$$

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$

- $\frac{\partial C^r}{\partial w_{ij}^l}$ is the multiplication of two terms

$\partial C^r / \partial w_{ij}^l$ - First Term



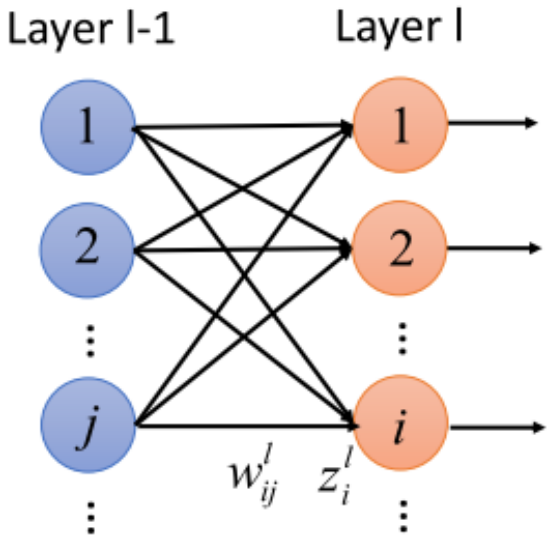
$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \cdots\cdots \rightarrow \Delta C^r$$

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$

- $\dfrac{\partial C^r}{\partial w_{ij}^l}$ is the multiplication of two terms

$\partial C^r / \partial w_{ij}^l$ - First Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$

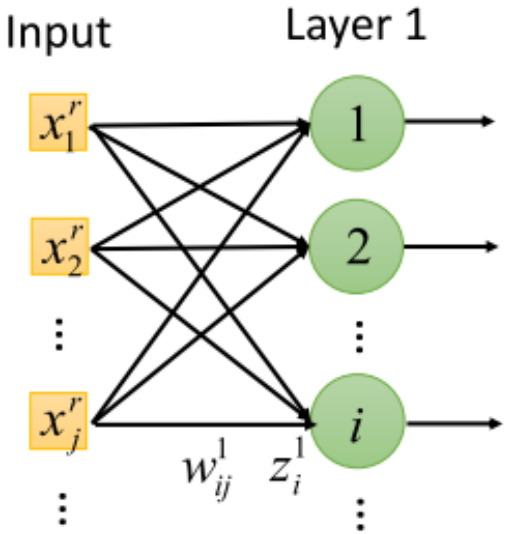


If l > 1

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

$\partial C^r / \partial w_{ij}^l$ - First Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$



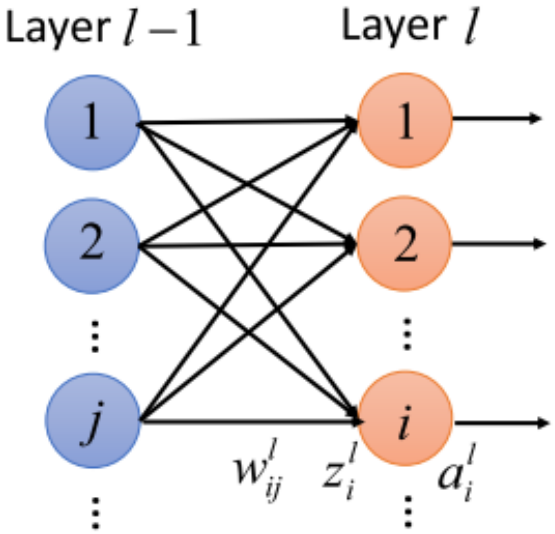
If l > 1

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

If l = 1

$$z_i^1 = \sum_j w_{ij}^1 x_j^r + b_i^1 \qquad \frac{\partial z_i^1}{\partial w_{ij}^1} = x_j^r$$

$\partial C^r / \partial w_{ij}^l$ - Second Term



$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \cdots\cdots \rightarrow \Delta C^r$$

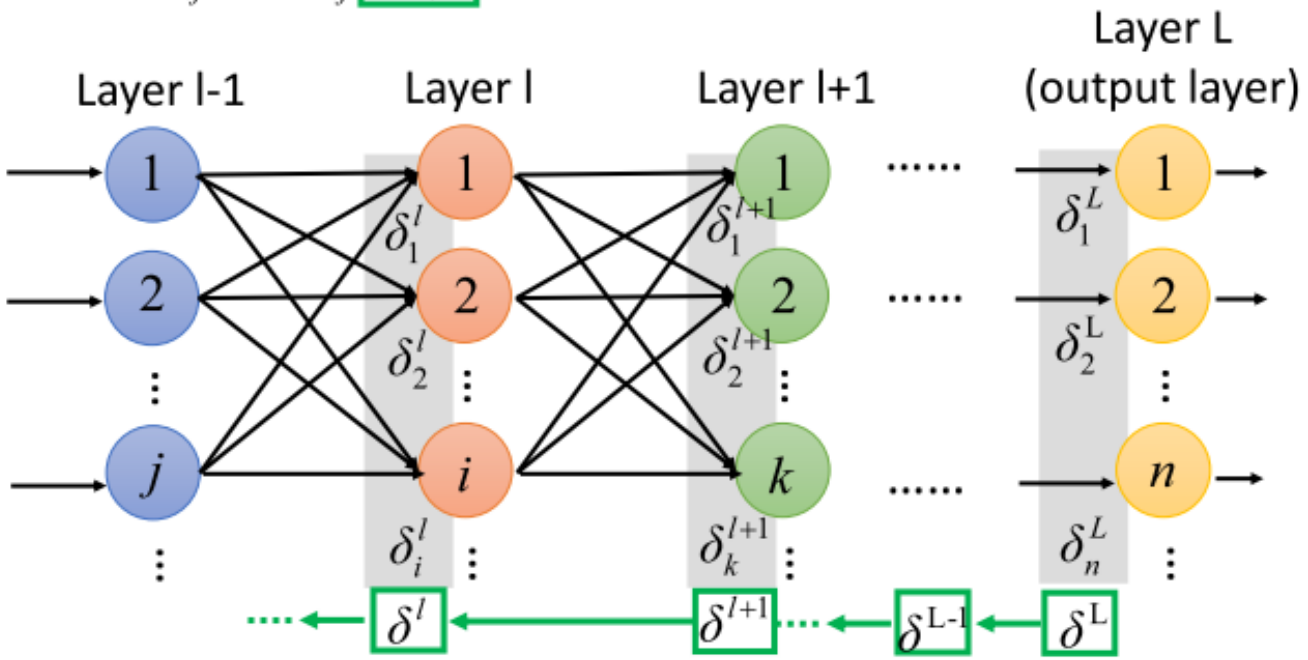
$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$

- $\dfrac{\partial C^r}{\partial w_{ij}^l}$ is the multiplication of two terms

$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$

1. How to compute δ^L
2. The relation of δ^l and δ^{l+1}



$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$

1. How to compute δ^L
2. The relation of δ^l and δ^{l+1}

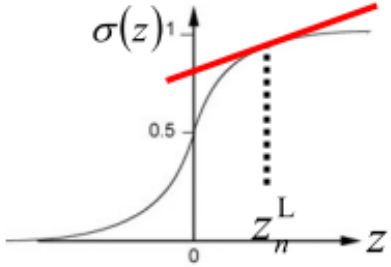
$$\delta_n^{\mathrm{L}} = \frac{\partial C^r}{\partial z_n^{\mathrm{L}}}$$

$$\Delta z_n^{\mathrm{L}} \rightarrow \Delta a_n^{\mathrm{L}} = \Delta y_n^{\mathrm{r}} \rightarrow \Delta C^{\mathrm{r}}$$

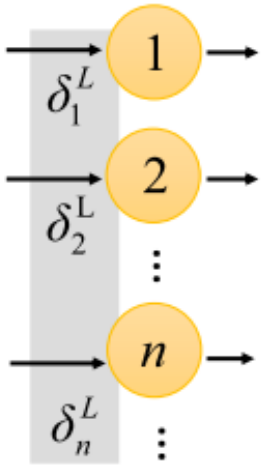
$$= \frac{\partial y_n^{\mathrm{r}}}{\partial z_n^{\mathrm{L}}} \frac{\partial C^r}{\partial y_n^{\mathrm{r}}}$$

Depending on the definition of cost function

$$\sigma'\left(z_n^{\mathrm{L}}\right)$$



Layer L
(output layer)



$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l} \rightarrow \delta_i^l$$

1. How to compute δ^L

2. The relation of δ^l and δ^{l+1}

δ^L?

$$\delta_n^L = \frac{\partial C^r}{\partial z_n^L}$$

$$= \frac{\partial y_n^r}{\partial z_n^L} \frac{\partial C^r}{\partial y_n^r}$$

$$= \sigma'\left(z_n^L\right) \frac{\partial C^r}{\partial y_n^r}$$

$$\sigma'\left(z^L\right) = \begin{bmatrix} \sigma'\left(z_1^L\right) \\ \sigma'\left(z_2^L\right) \\ \vdots \\ \sigma'\left(z_n^L\right) \\ \vdots \end{bmatrix} \quad \nabla C^r\left(y^r\right) = \begin{bmatrix} \partial C^r / \partial y_1^r \\ \partial C^r / \partial y_2^r \\ \vdots \\ \partial C^r / \partial y_n^r \\ \vdots \end{bmatrix}$$

$$\delta^L = \sigma'\left(z^l\right) \bullet \nabla C^r\left(y^r\right)$$

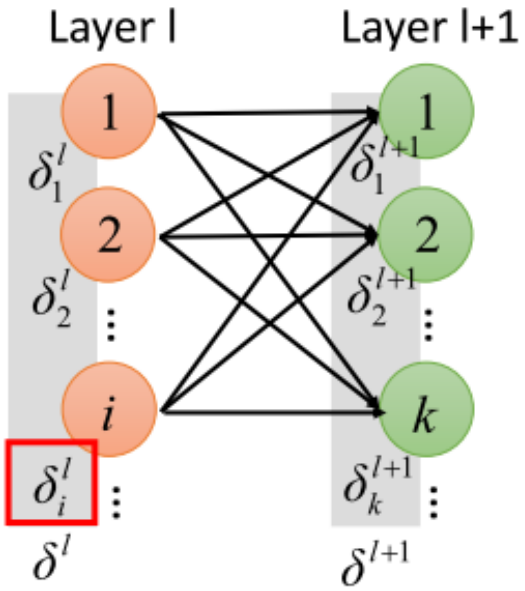
element-wise multiplication

$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$

1. How to compute δ^L

2. The relation of δ^l and δ^{l+1}



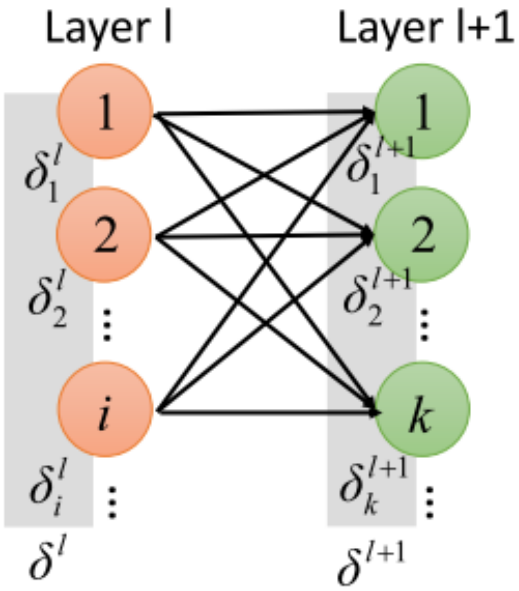
$$\delta_i^l = \frac{\partial C^r}{\partial z_i^l}$$

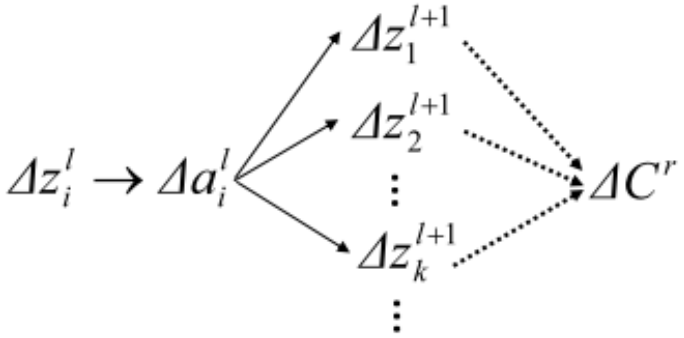
$$\Delta z_i^l \rightarrow \Delta a_i^l \rightarrow \Delta z_1^{l+1}, \Delta z_2^{l+1}, \ldots, \Delta z_k^{l+1}, \ldots \rightarrow \Delta C^r$$

$$\delta_i^l = \frac{\partial C^r}{\partial z_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \boxed{\frac{\partial C^r}{\partial z_k^{l+1}}} \rightarrow \delta_k^{l+1}$$

$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$





$$\delta_i^l = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

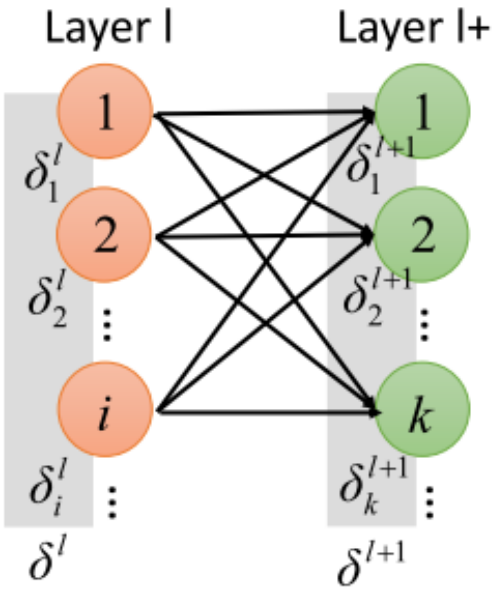
$$\sigma'\left(z_i^l\right) \qquad z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

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

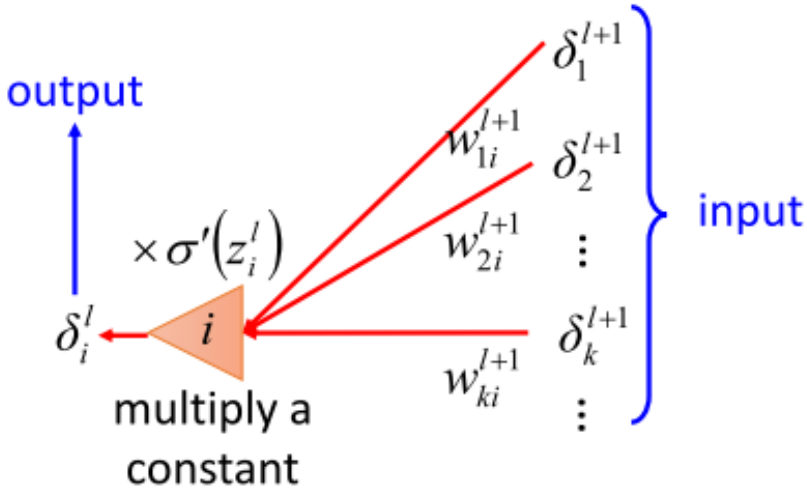
$\partial C^r / \partial w_{ij}^l$ - Second Term

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \rightarrow \delta_i^l$$

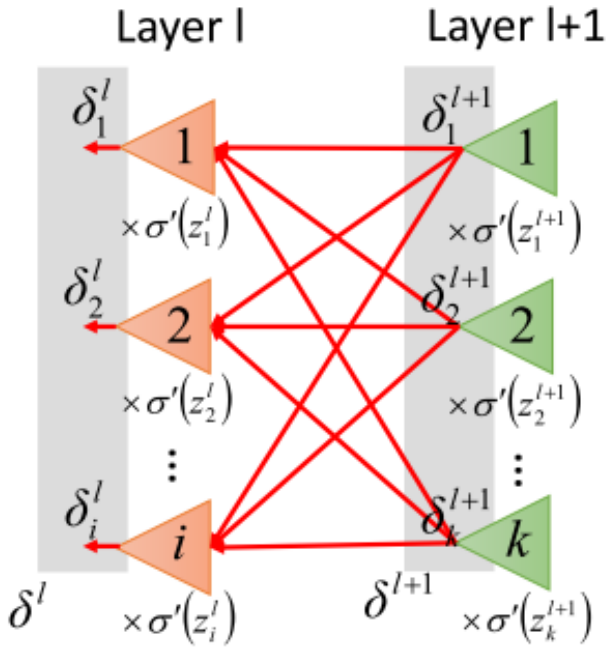
$$\delta_i^l = \sigma'\left(z_i^l\right) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$



new type of neuron



$\partial C^r / \partial w_{ij}^l$ - Second Term

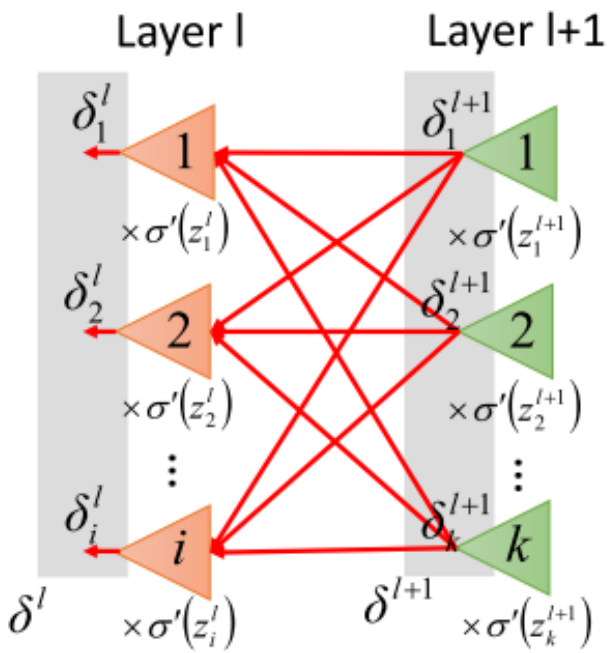


$$\delta_i^l = \sigma'\left(z_i^l\right)\sum_k w_{ki}^{l+1}\delta_k^{l+1}$$

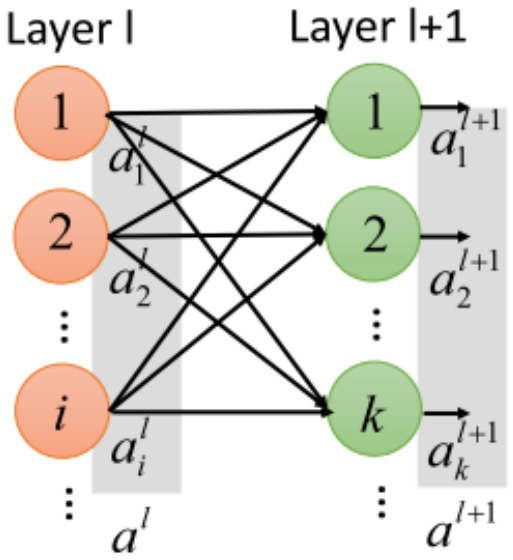
$$\sigma'\left(z^l\right) = \begin{bmatrix} \sigma'\left(z_1^l\right) \\ \sigma'\left(z_2^l\right) \\ \vdots \\ \sigma'\left(z_i^l\right) \\ \vdots \end{bmatrix}$$

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

$\partial C^r / \partial w_{ij}^l$ - Second Term



Compare



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

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

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}}$$

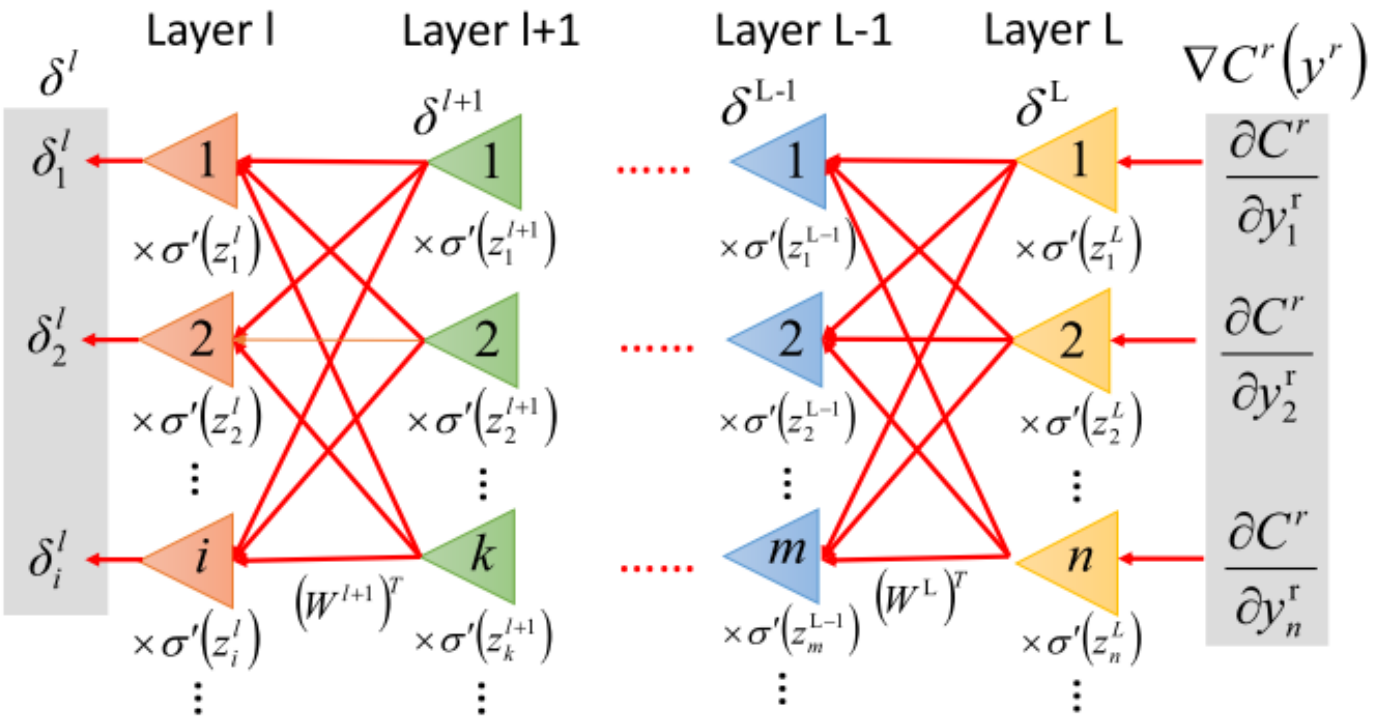
δ_i^l

1. How to compute δ^L

$$\Rightarrow \delta^L = \sigma'\left(z^L\right) \bullet \nabla C^r\left(y^r\right)$$

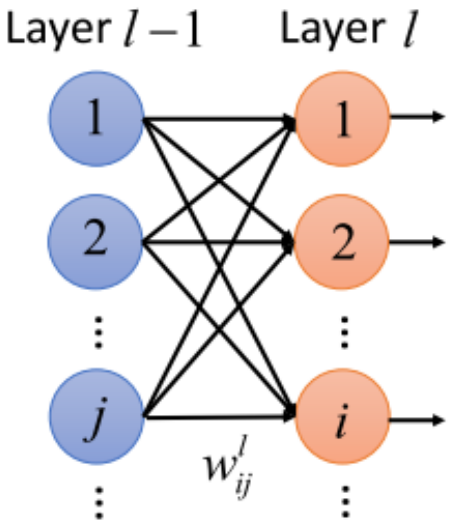
2. The relation of δ^l and δ^{l+1}

$$\Rightarrow \delta^l = \sigma'\left(z^l\right) \bullet \left(W^{l+1}\right)^T \delta^{l+1}$$



Concluding Remarks

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$



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

$$\delta_i^l$$

Forward Pass

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

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

$$\cdots\cdots$$

$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

$$a^{l-1} = \sigma(z^{l-1})$$

Backward Pass

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

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

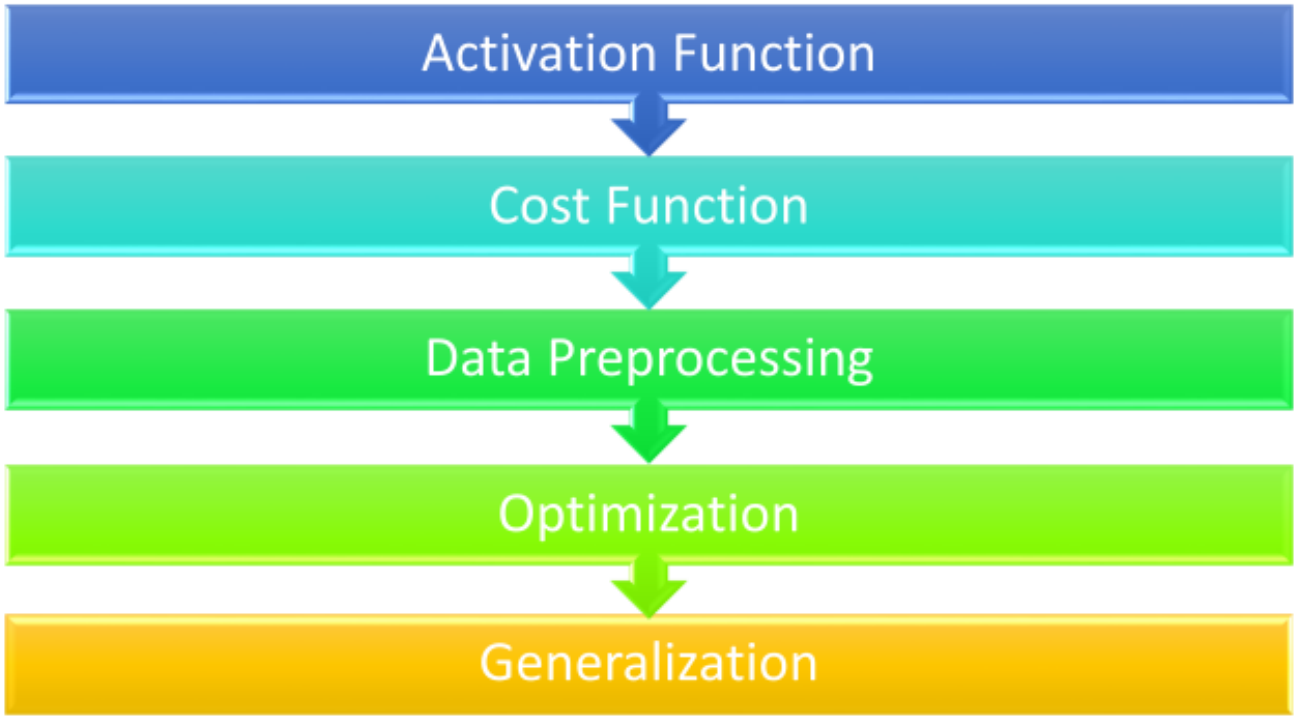
$$\cdots\cdots$$

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

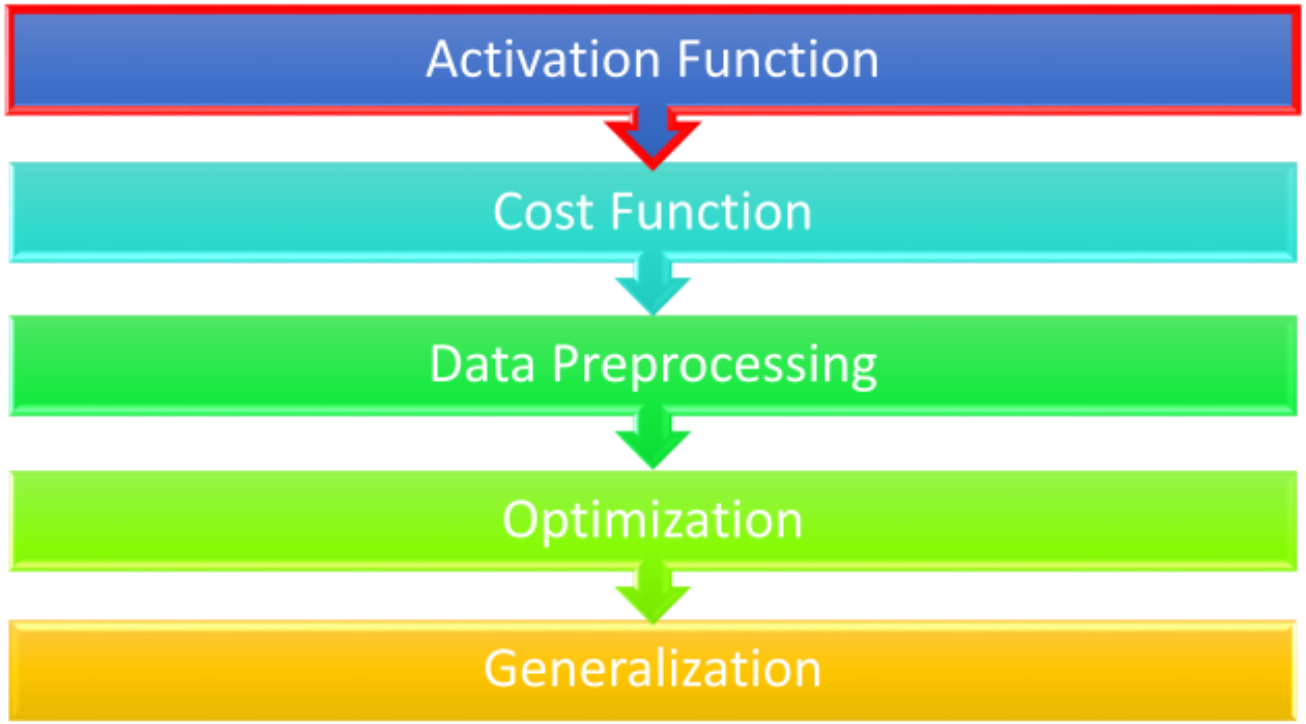
$$\cdots\cdots$$

Tips for training Deep Neural Network

Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

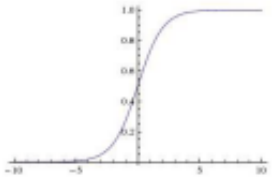
Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

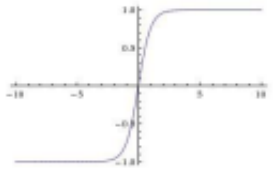
Activation Functions

Sigmoid

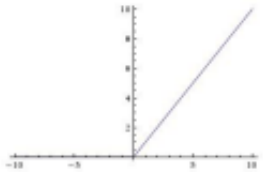
$$\sigma(x) = 1/(1+e^{-x})$$



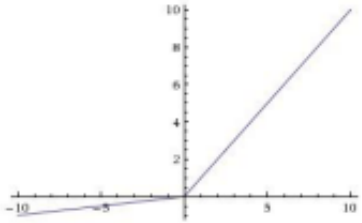
tanh tanh(x)



ReLU max(0,x)



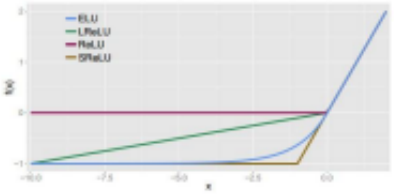
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



Goals?

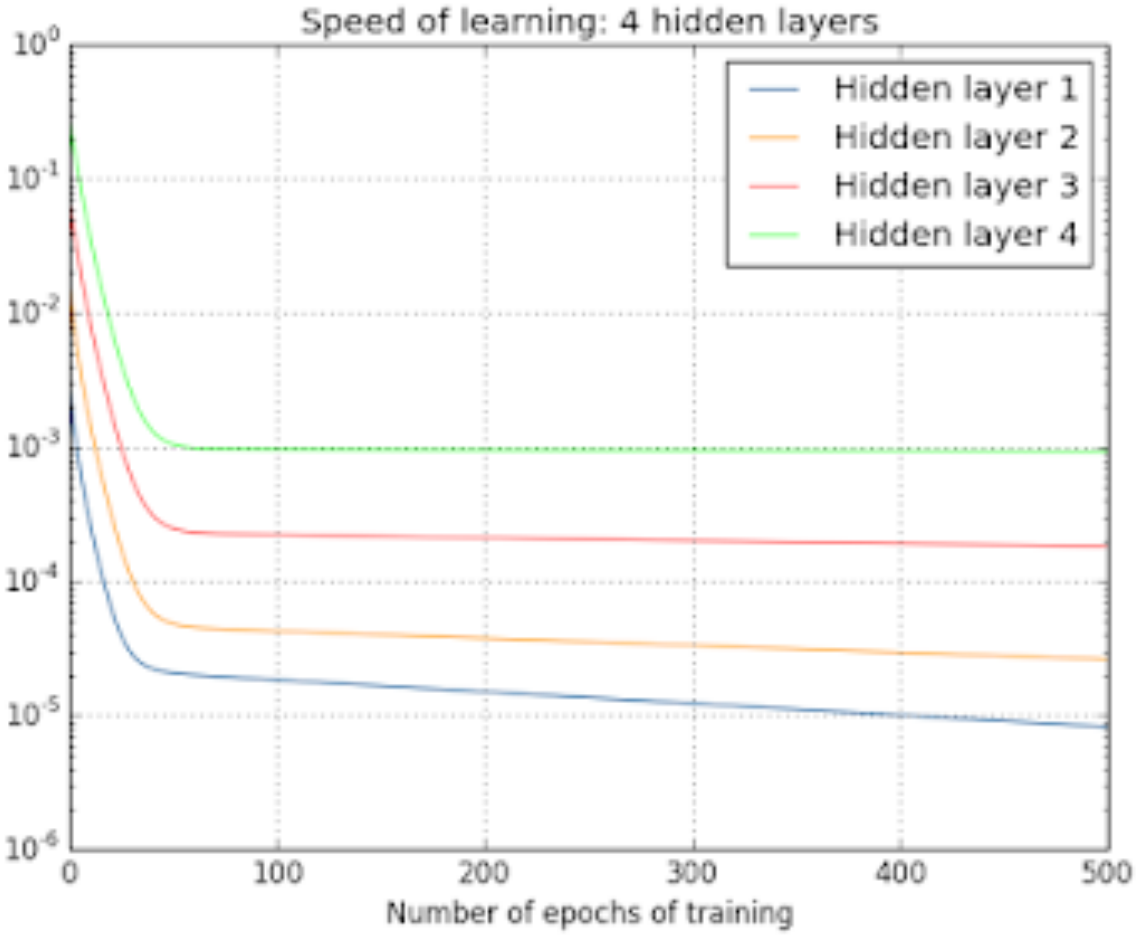
主要作用是引入非線性。

Basic Concepts

- 實用上最常使用 ReLU
  - Leaky ReLU, Maxout 也可以試試
  - tanh 和 sigmoid 盡量別用。

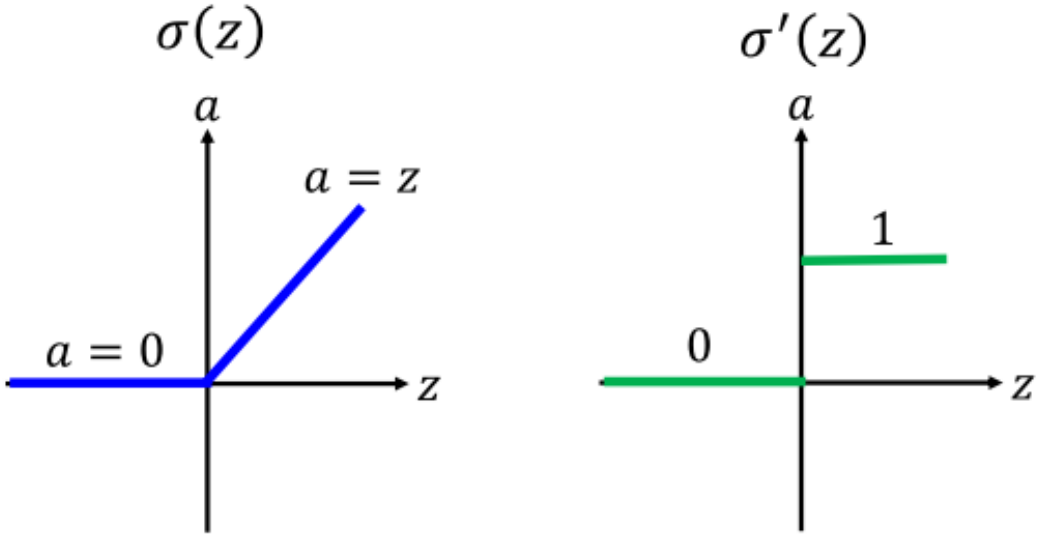
Why?

- Vanishing gradient



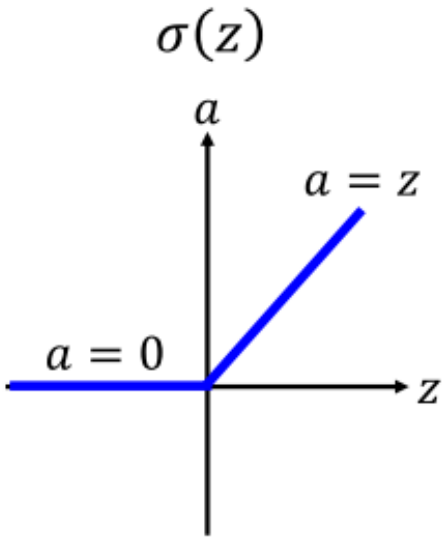
ReLU

- Rectified Linear Unit (ReLU)



ReLU

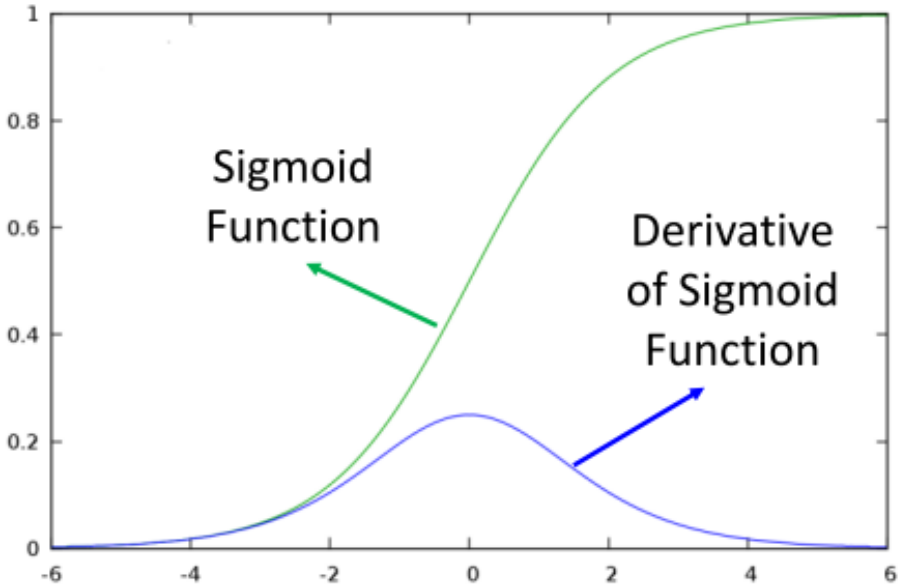
- Rectified Linear Unit (ReLU)



Reason:

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

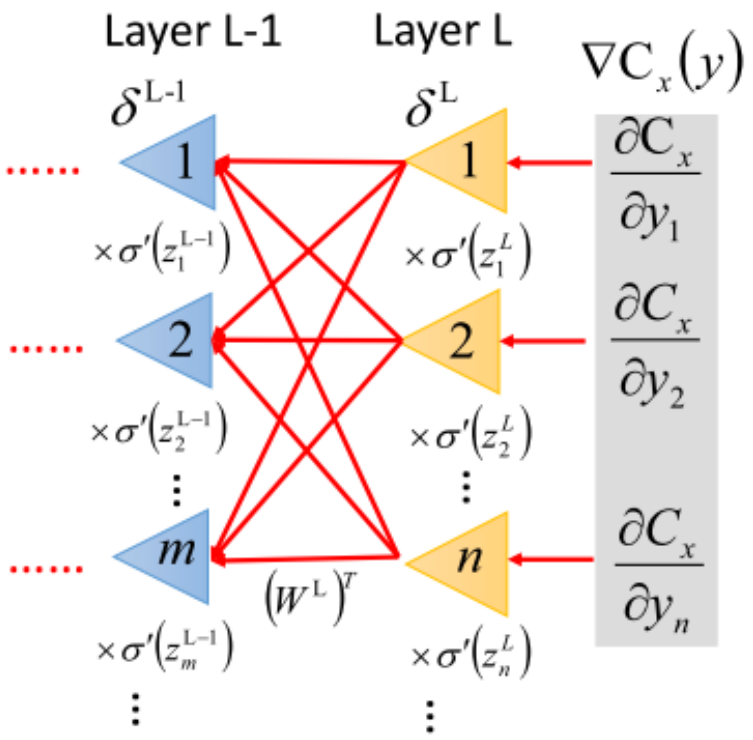
Problem of Sigmoid



Derivative of Sigmoid Function is always smaller than 1

Vanishing Gradient Problem

Backward Pass:

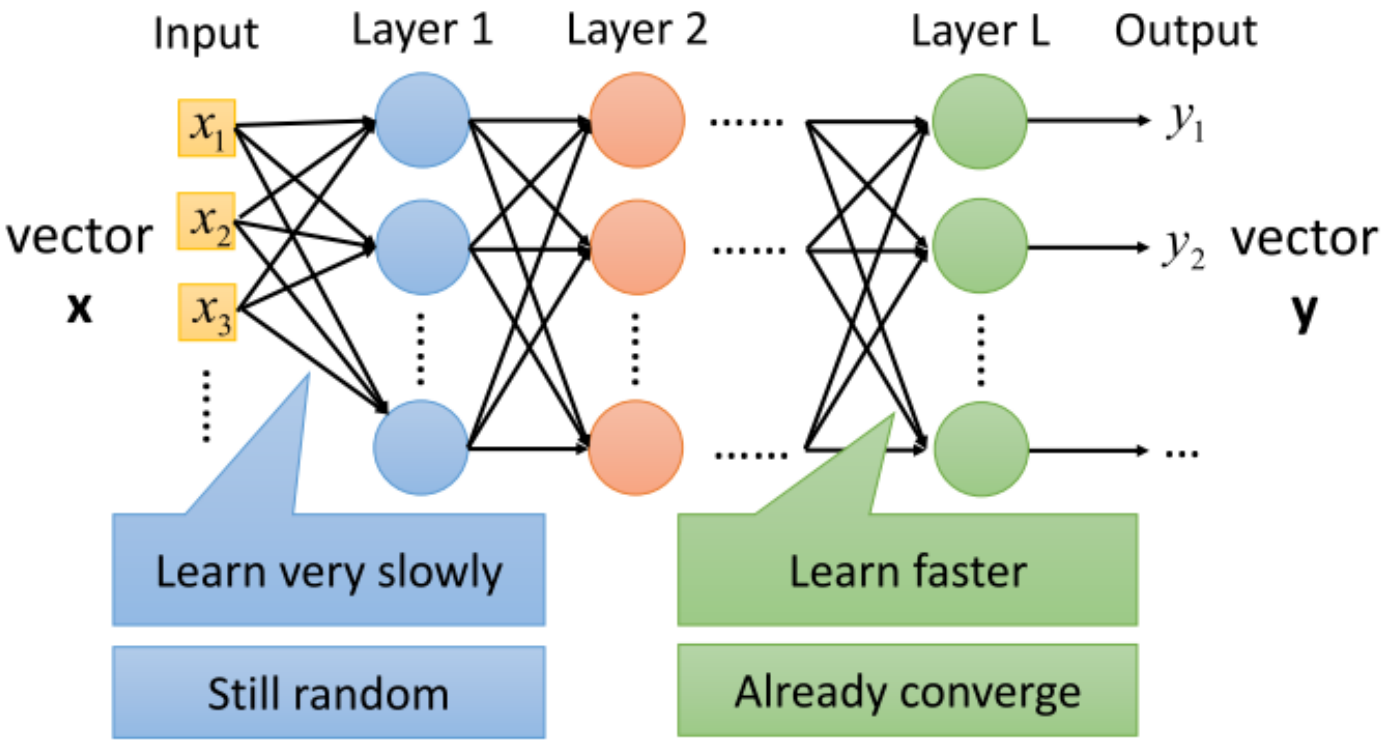


- For sigmoid function, $\sigma'(z)$ always smaller than 1
- Error signal is getting smaller and smaller

Gradient is smaller

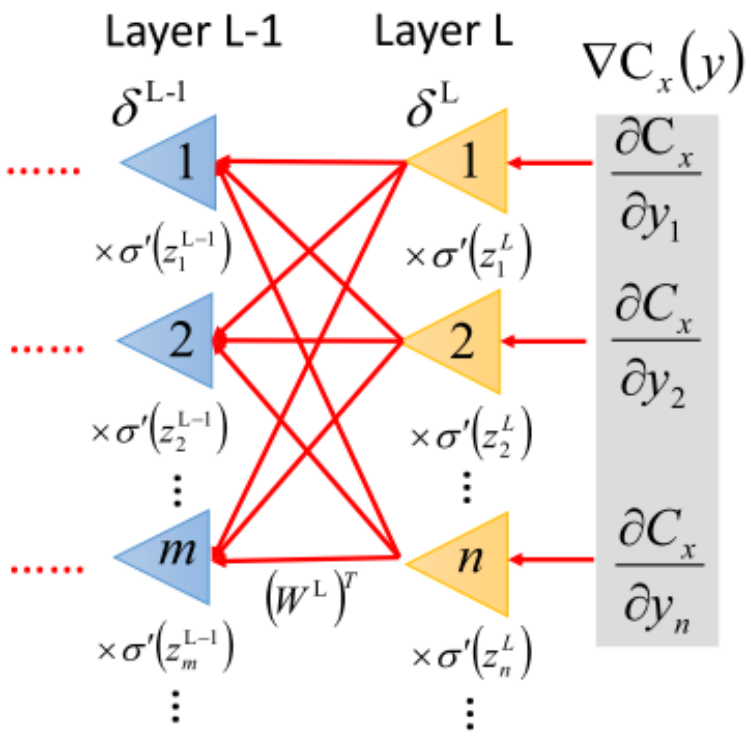
$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l}\frac{\partial C_x}{\partial z_i^l} \rightarrow \delta_i^l$$

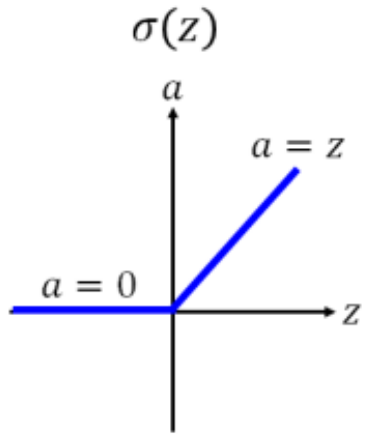
Vanishing Gradient Problem

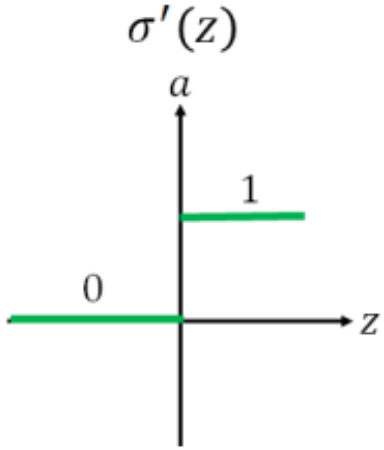


ReLU

Backward Pass:

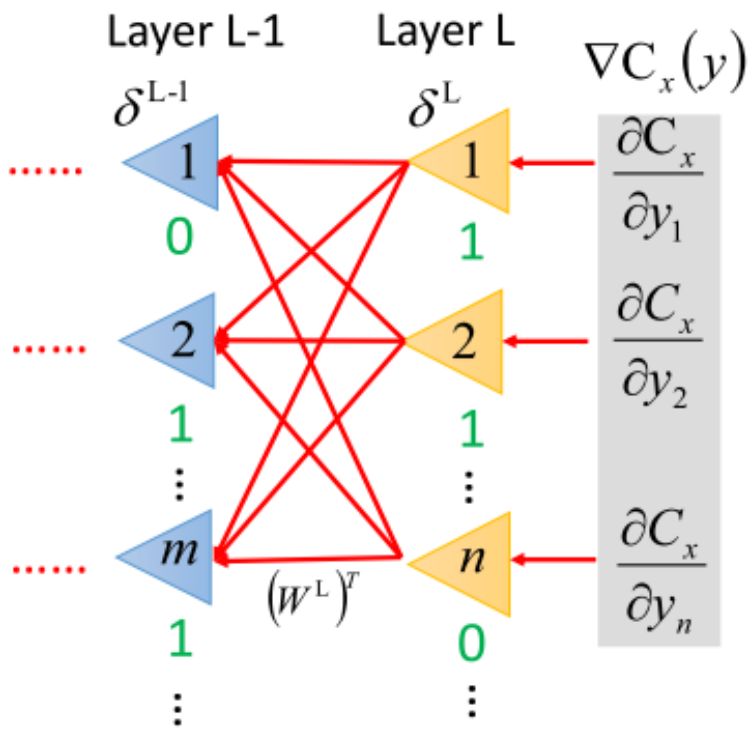


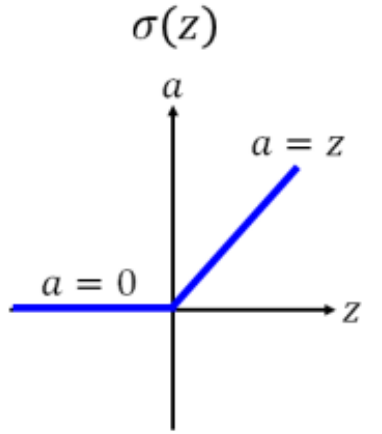


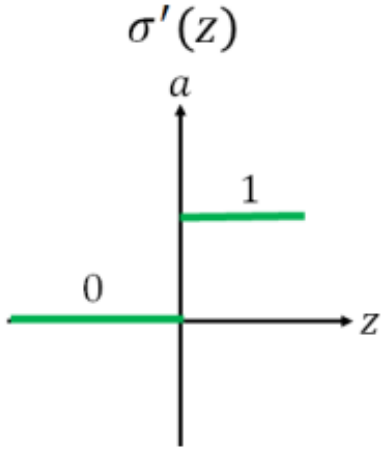


ReLU

Backward Pass:

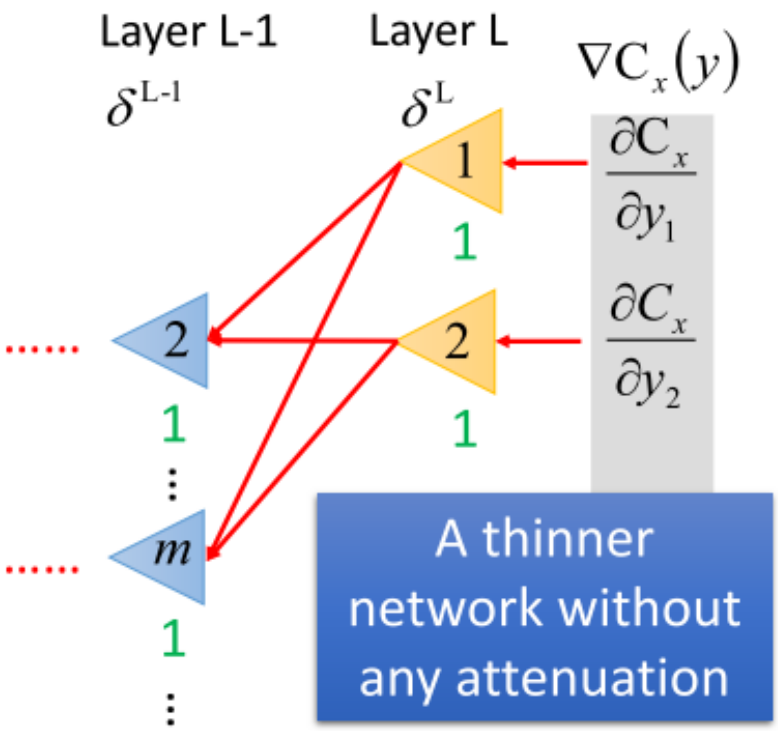


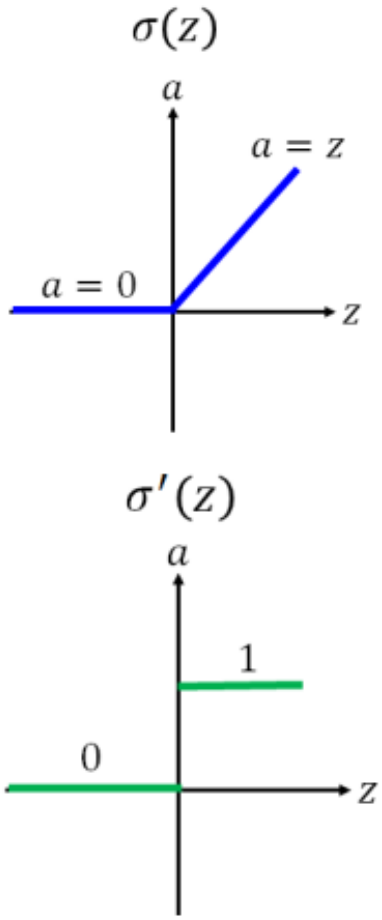




ReLU

Backward Pass:





ReLU

$$\frac{\partial C_x}{\partial w_{nj}^L} = \frac{\partial z_n^L}{\partial w_{nj}^L} \frac{\partial C_x}{\partial z_n^L}$$

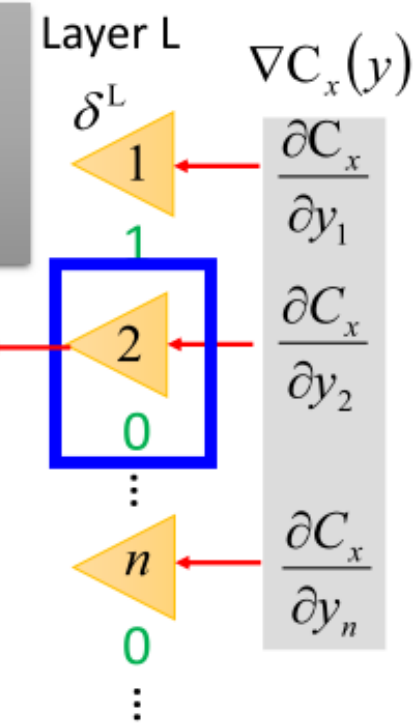
Backward Pass:

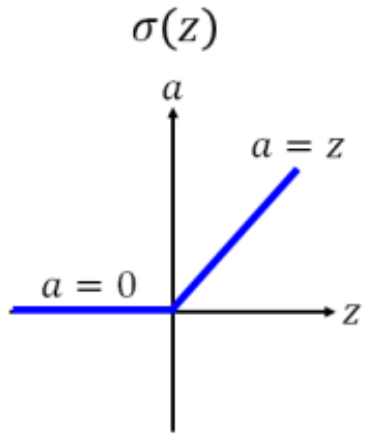
All the weights connected to this neuron will not update.

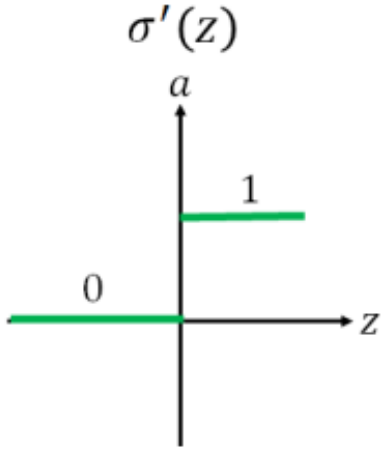
$$\delta_n^L = \frac{\partial C_x}{\partial z_n^L} = 0$$

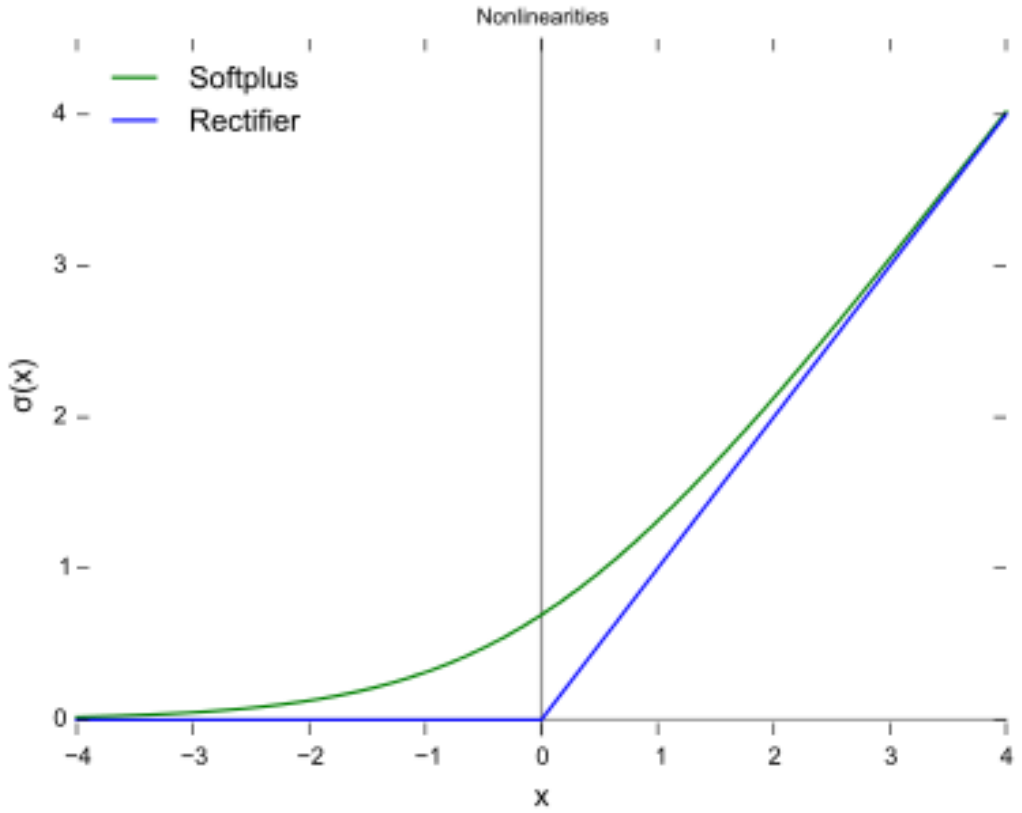
Possible solution:

1. softplus
2. Initialize with large bias



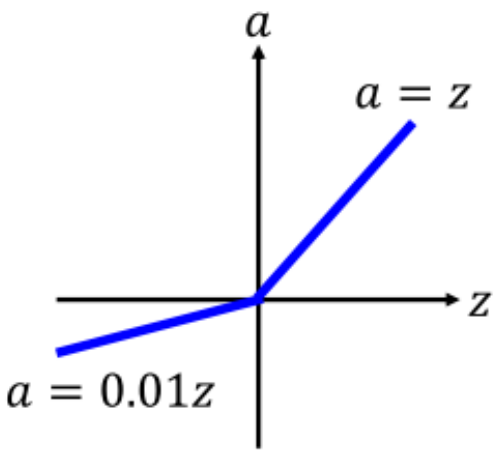




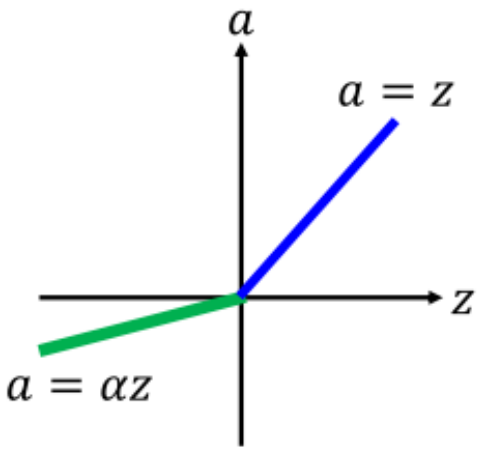

Nonlinearities
Softplus
Rectifier
σ(x)
x
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4

ReLU - variant

Leaky ReLU



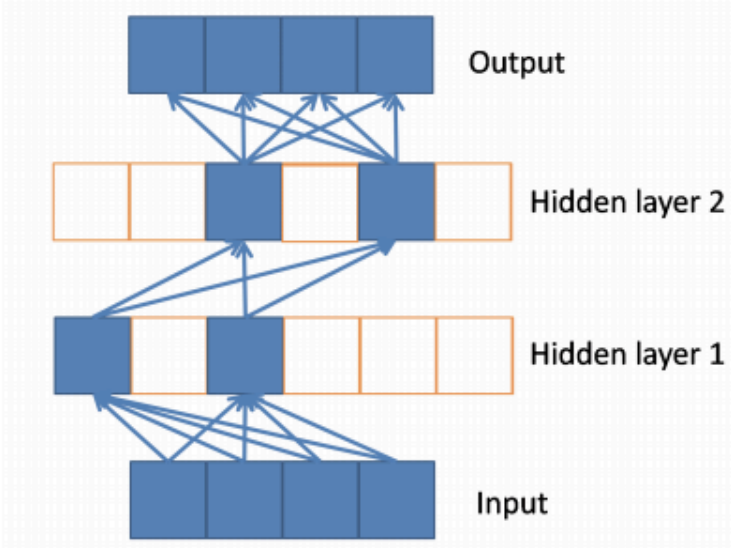
Parametric ReLU



α also learned by gradient descent

Why?

- Occam's razor



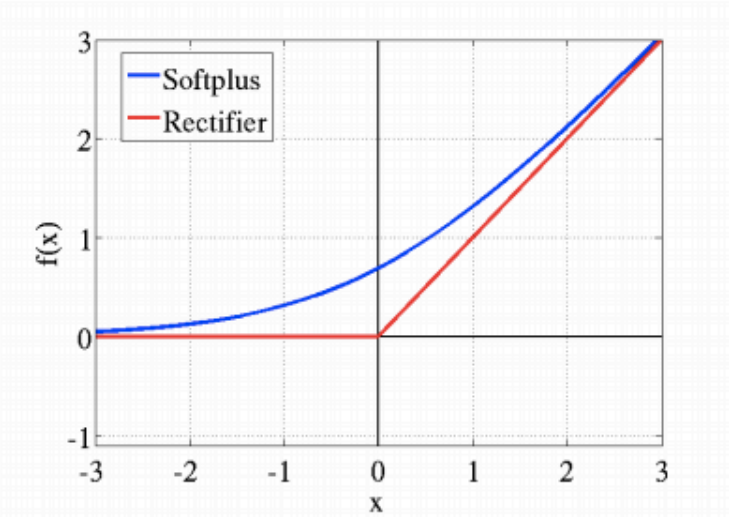
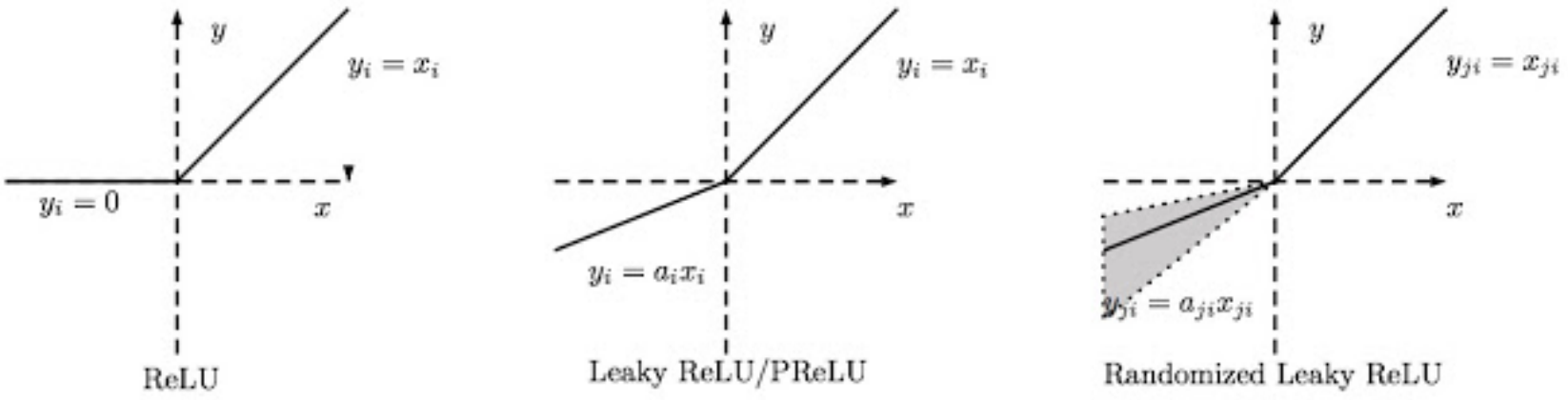
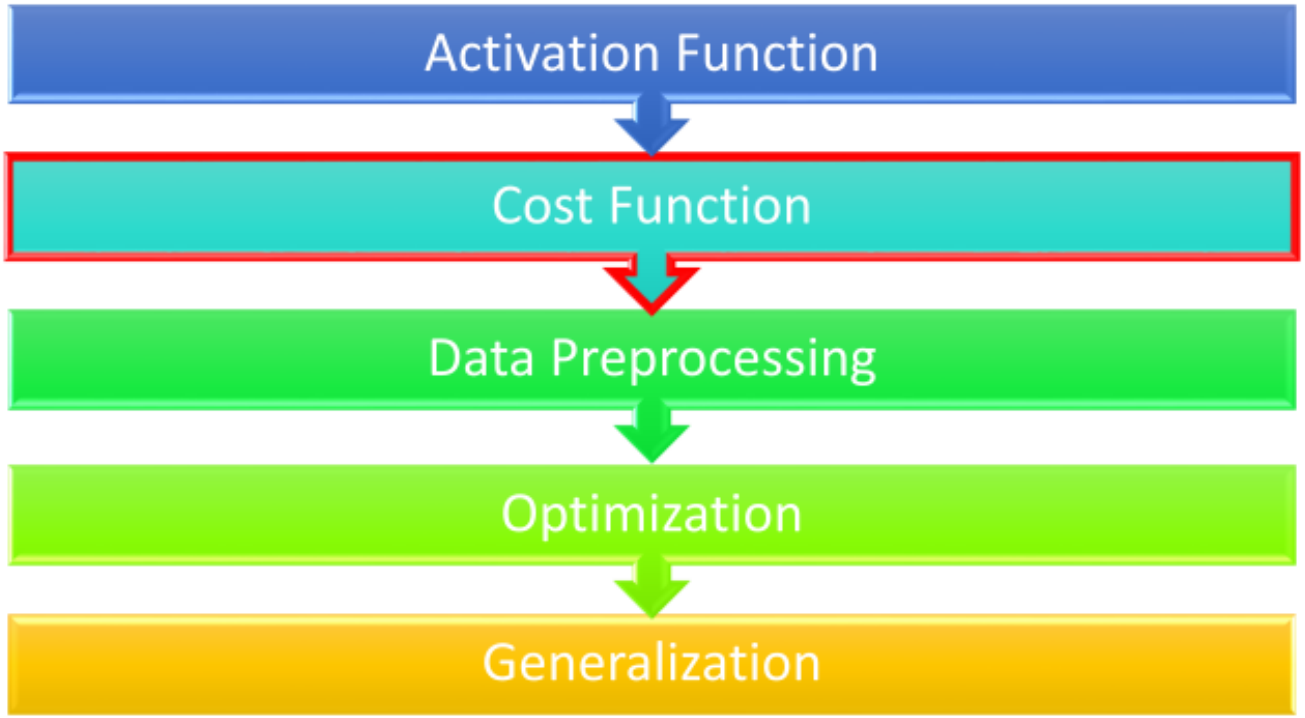


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

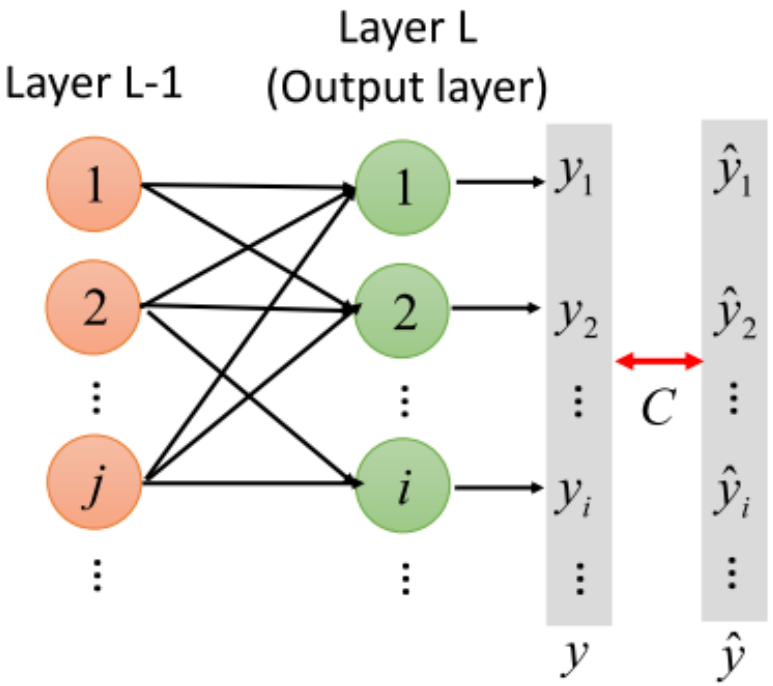
Dead ReLU Problem



Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

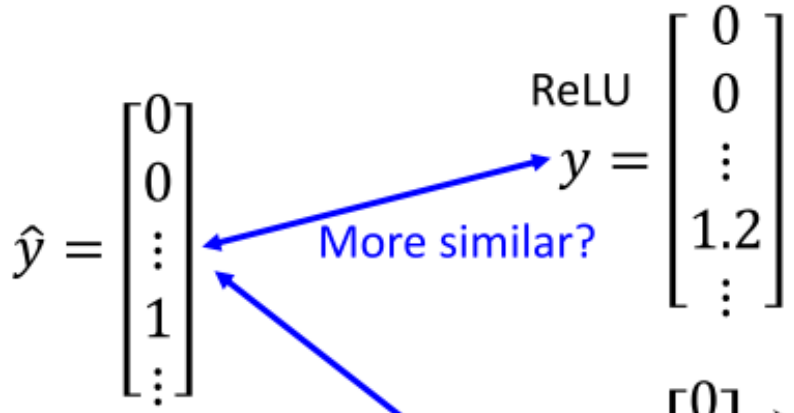
Cost Function



$$C = \frac{1}{2}\|y - \hat{y}\|^2$$

$$= \frac{1}{2}\sum_n (y_n - \hat{y}_n)^2$$

Output Layer



Classification Task:

Only one dimension is 1, and others are all 0

- Larger output means larger confidence

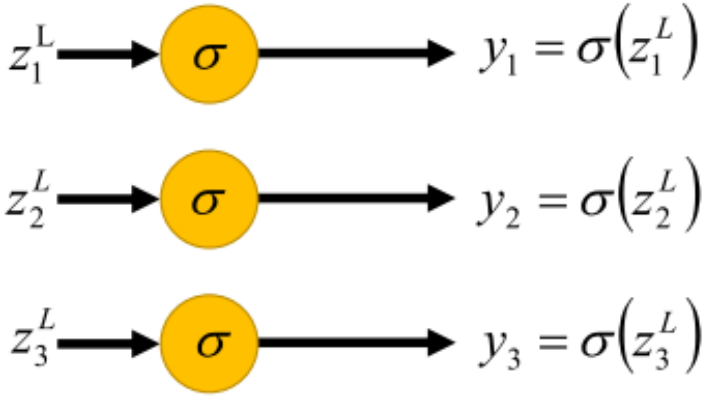
Better?

It is better to let the output bounded.

Softmax

- Softmax layer as the output layer

Ordinary Output layer

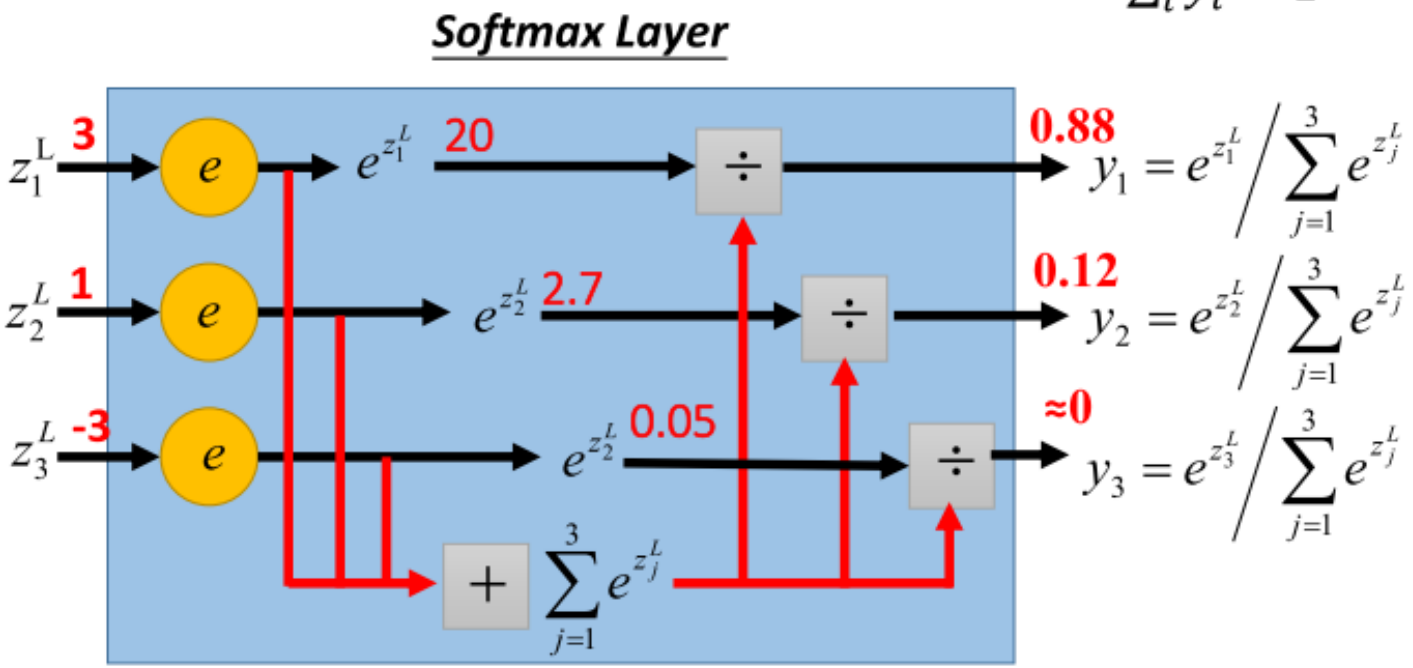


Softmax

- Softmax layer as the output layer

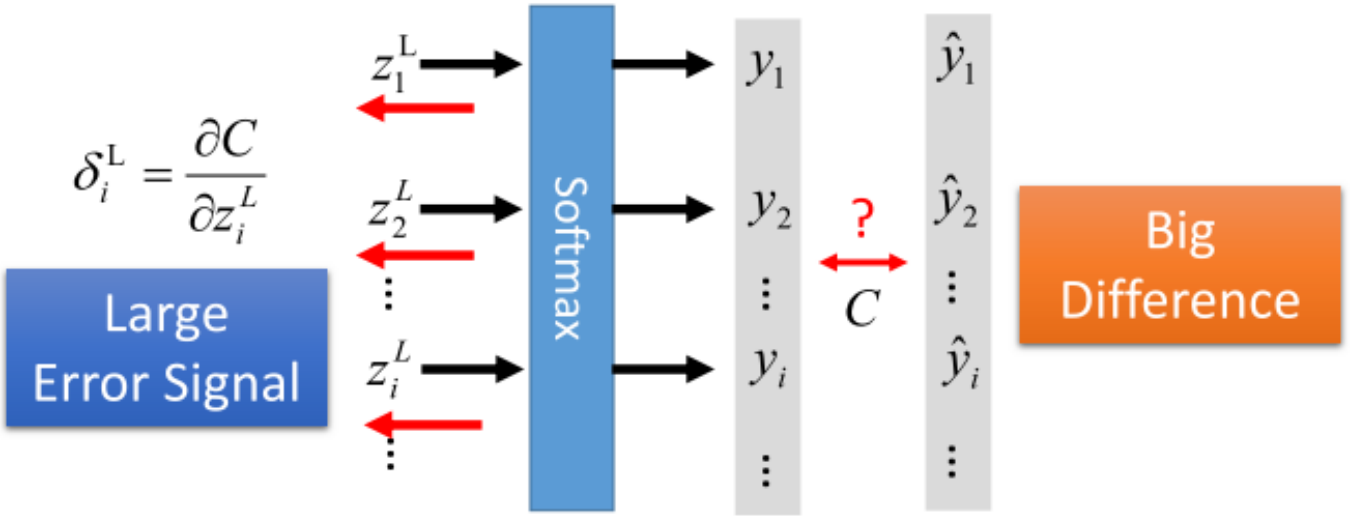
Probability:

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



Softmax

- What kind of cost function should we used for softmax layer output?



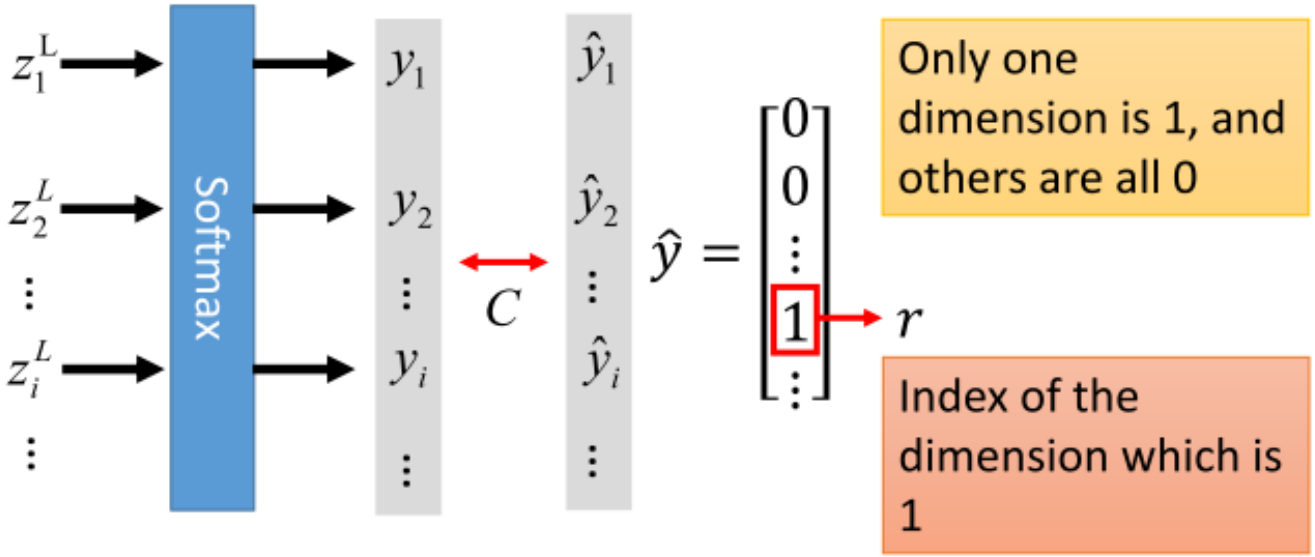
For discrete p and q this means

$$H(p,q) = -\sum_x p(x)\,\log q(x).$$

Softmax

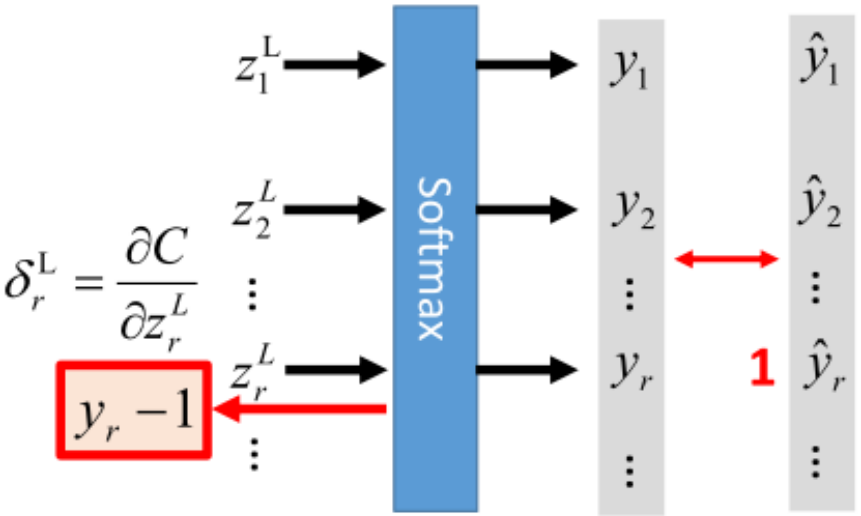
$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

Do we have to consider other dimensions?



$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

$$C = -\log y_r$$



$$\frac{\partial C}{\partial y_r}\frac{\partial y_r}{\partial z_r^L}$$

$$\delta_r^L = \frac{\partial C}{\partial z_r^L} = -\frac{1}{y_r}\frac{\partial y_r}{\partial z_r^L} = -\frac{1}{y_r}\left(y_r - y_r^2\right) = y_r - 1$$

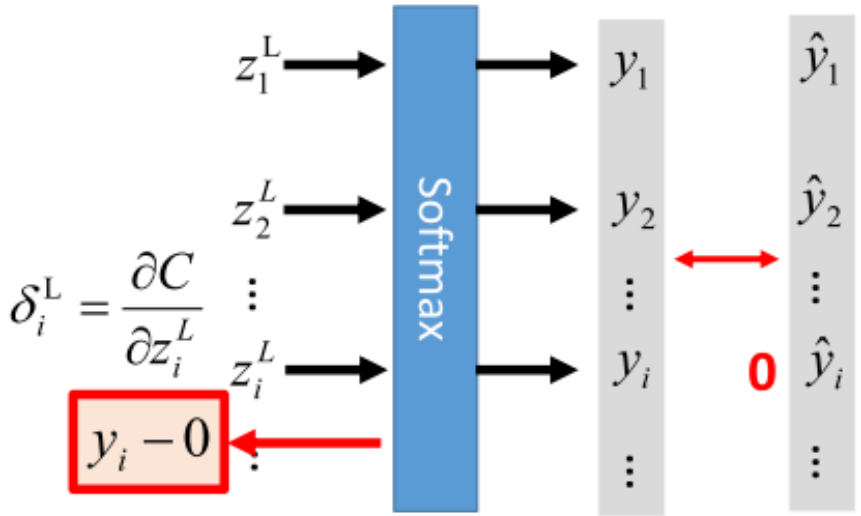
$$y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}}$$

z_r^L appears in both numerator and denominator

The absolute value of δ_r^L is larger when y_r is far from 1

$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

$$C = -\log y_r$$



$$\frac{\partial C}{\partial y_r}\frac{\partial y_r}{\partial z_i^L}$$

$i \neq r$

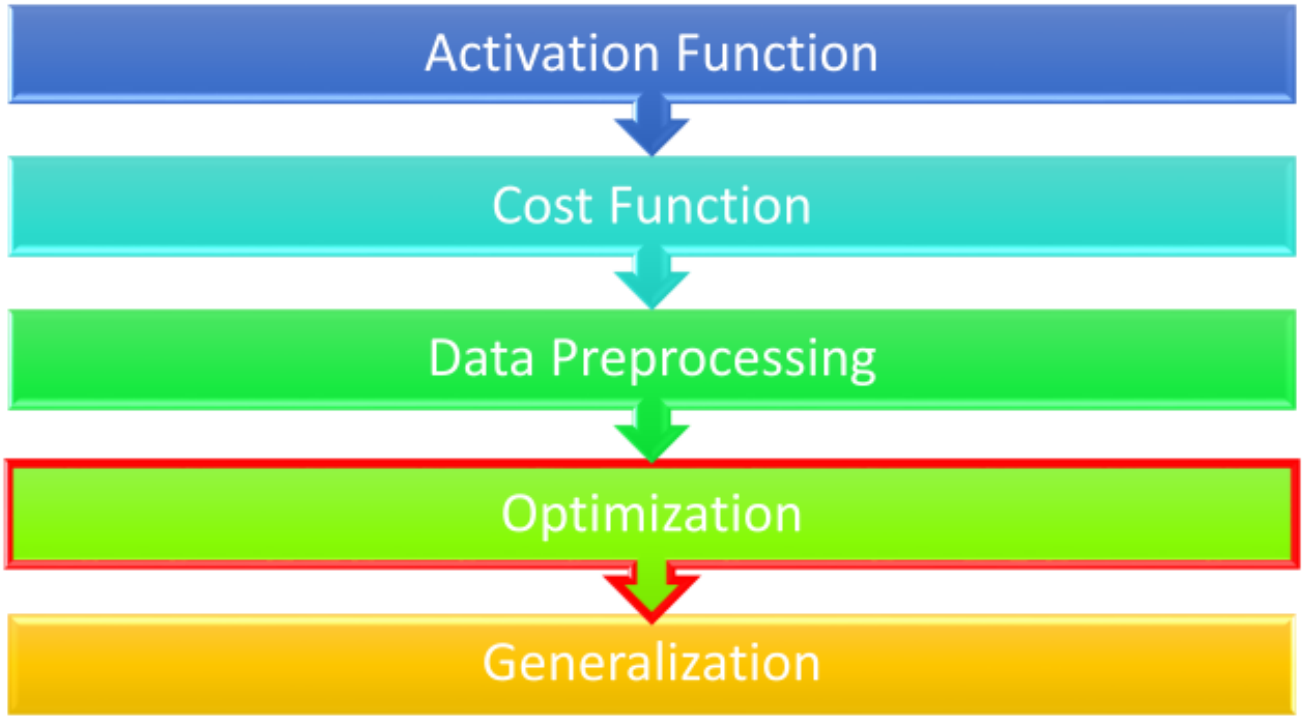
$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = -\frac{1}{y_r}\frac{\partial y_r}{\partial z_i^L} = -\frac{1}{y_r}\left(-y_r y_i\right) = y_i$$

$$y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}}$$

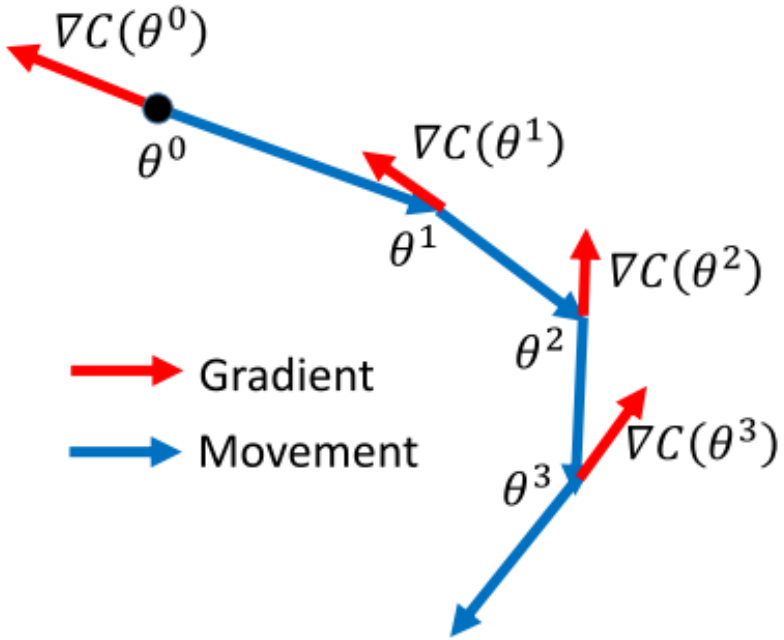
z_i^L appears only in denominator

The absolute value of δ_i^L is larger when y_i is larger

Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

Vanilla 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

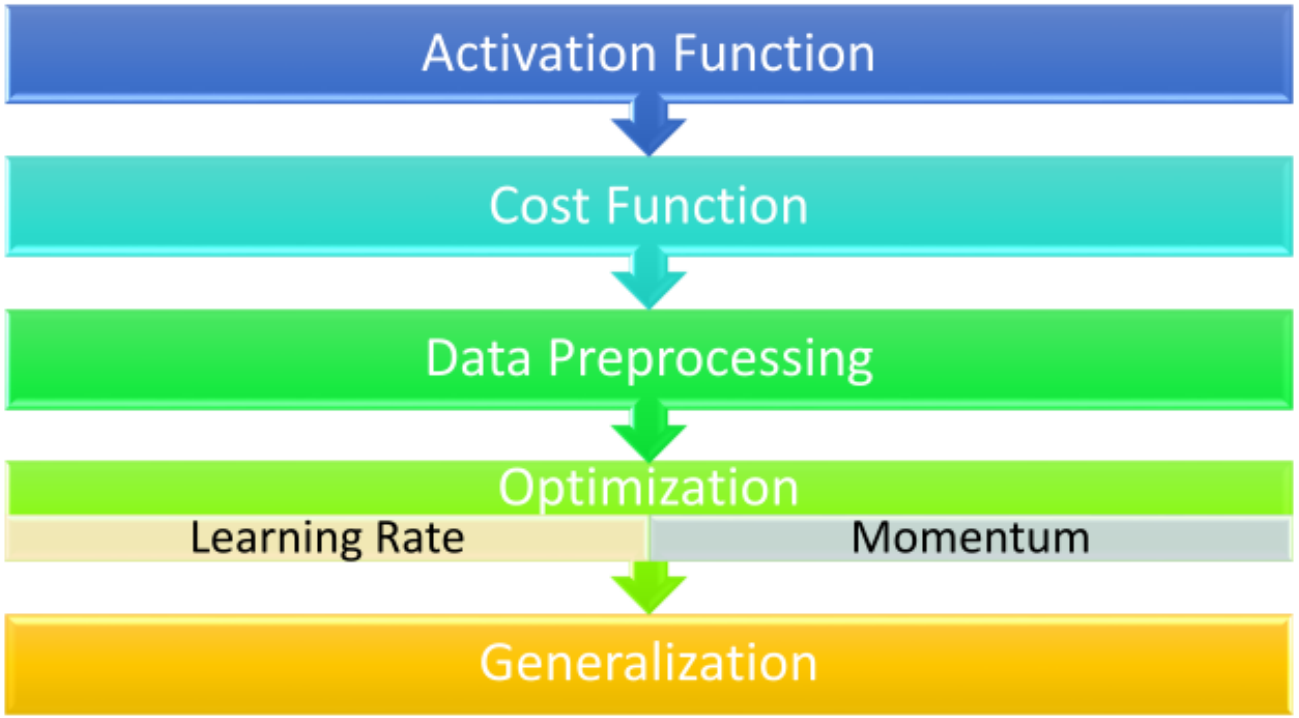
Move to $\theta^2 = \theta^1 - \eta\nabla C(\theta^1)$

⋮

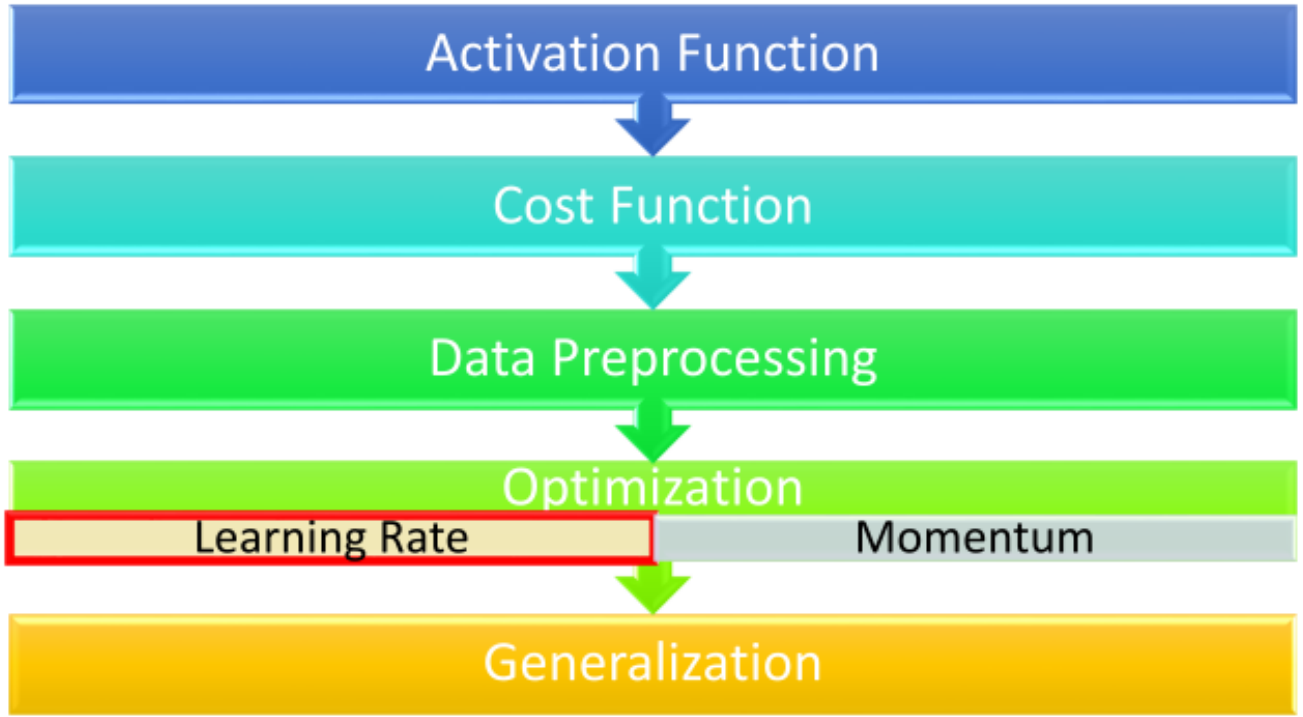
Stop until $\nabla C(\theta^t) \approx 0$

1. How to determine the learning rates
2. Stuck at local minima or saddle points

Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Learning Rate
Momentum
Generalization

Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Learning Rate
Momentum
Generalization

Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay: $\eta^t = \eta/\sqrt{t+1}$
- Learning rate cannot be one-size-fits-all
  - Give different parameters different learning rates

Adagrad

$$g^t = \frac{\partial C(\theta^t)}{\partial w} \qquad \eta^t = \frac{\eta}{\sqrt{t+1}}$$

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

w is one parameters

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

σ^t: ***root mean square*** of the previous derivatives of parameter w

Parameter dependent

Adagrad

σ^t: ***root mean square*** of the previous derivatives of parameter w

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}\left[(g^0)^2 + (g^1)^2\right]}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}\left[(g^0)^2 + (g^1)^2 + (g^2)^2\right]}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t} (g^i)^2}$$

Adagrad

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

$$\eta^t = \frac{\eta}{\sqrt{t+1}}$$ 1/t decay

$$\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

Contradiction?

$$g^t = \frac{\partial C(\theta^t)}{\partial w} \quad \eta^t = \frac{\eta}{\sqrt{t+1}}$$

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

Larger gradient, larger step

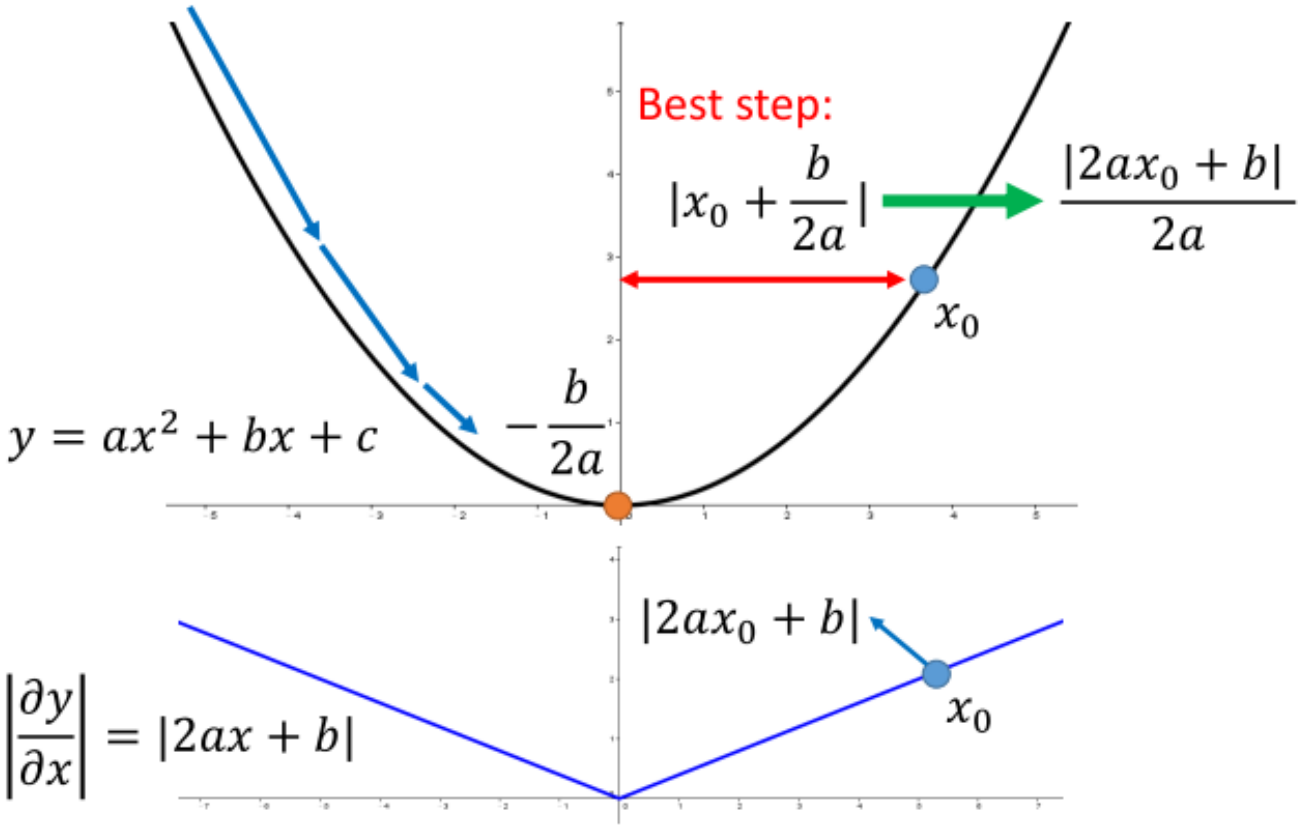
Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

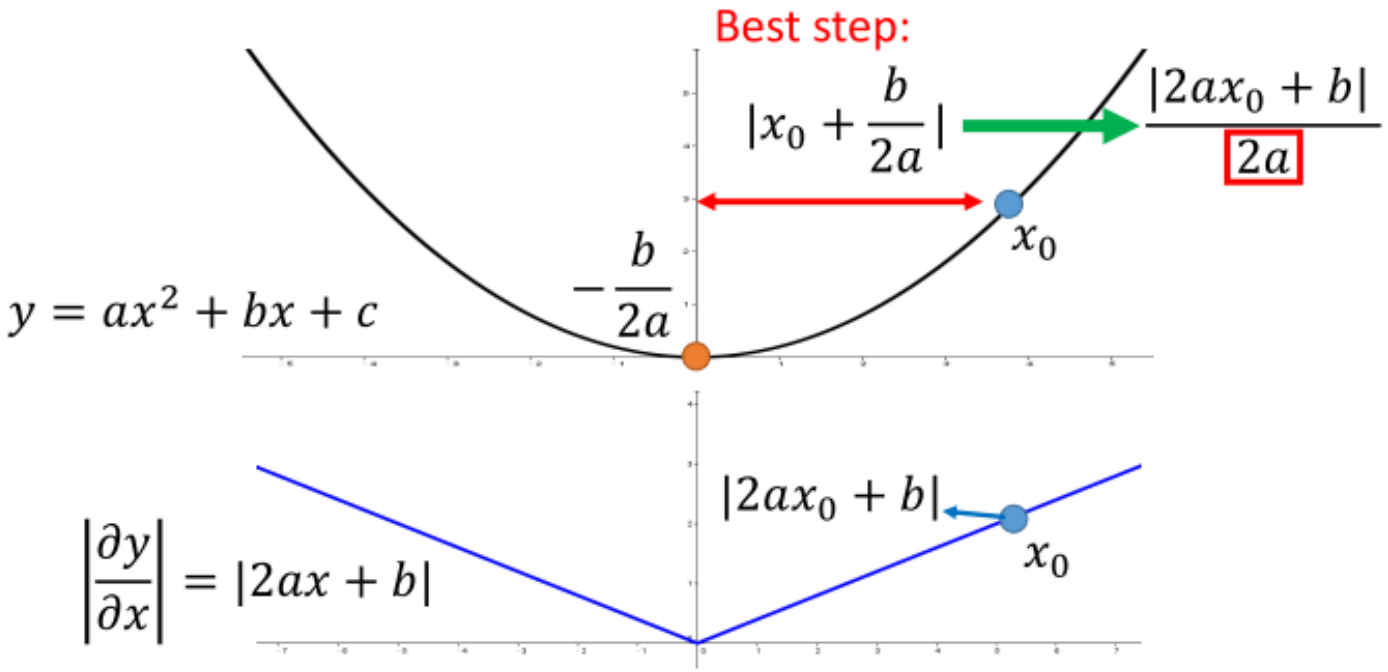
Larger gradient, larger step

Larger gradient, smaller step

Larger gradient, larger steps?



Second Derivative

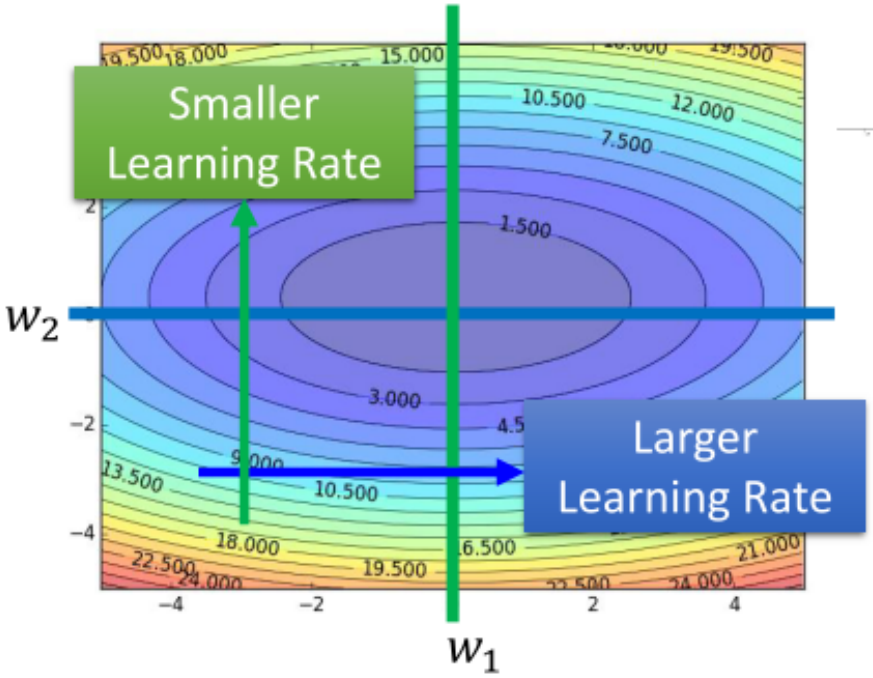


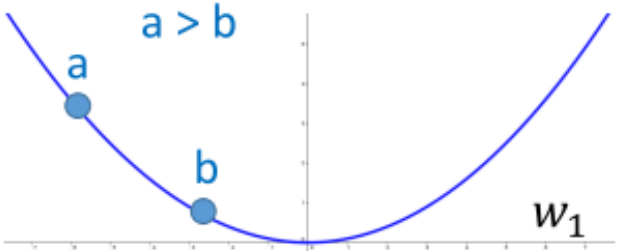
$$\frac{\partial^2 y}{\partial x^2} = 2a$$

The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$

More than one parameters

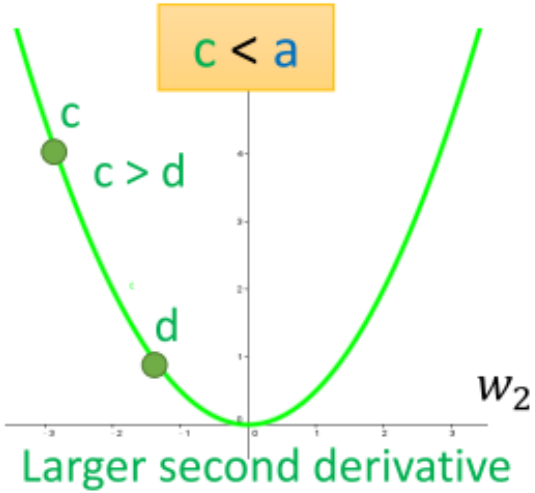
The best step is $\frac{|\text{First derivative}|}{\text{Second derivative}}$





smaller second derivative

c < a



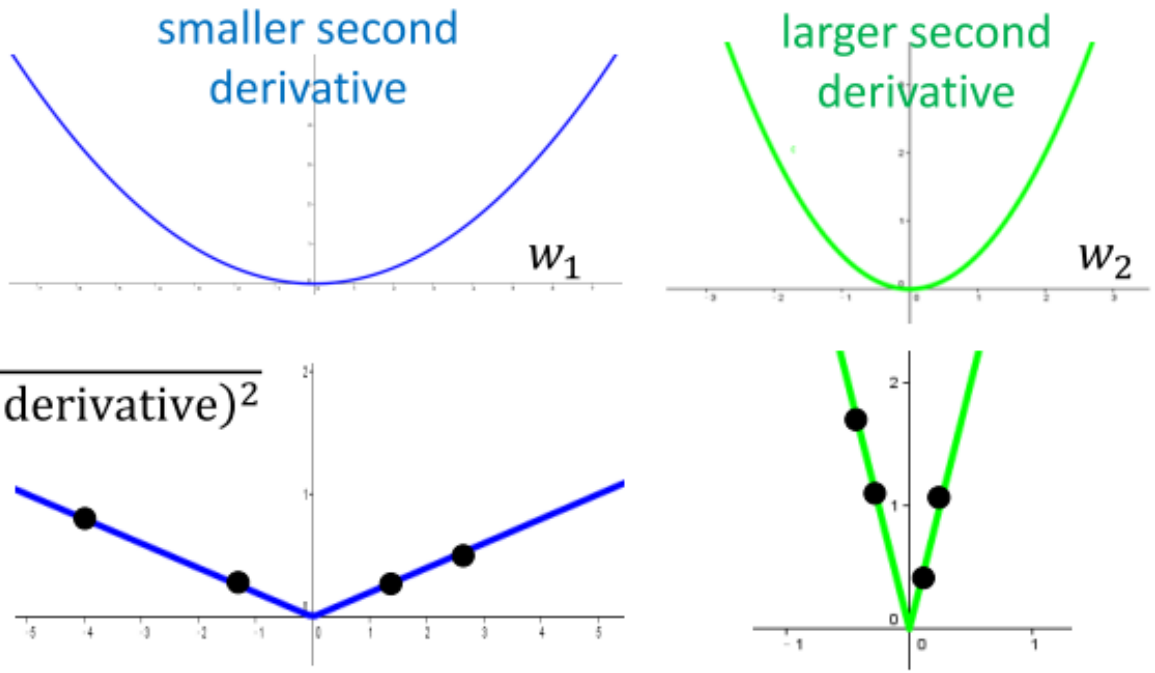
Larger second derivative

What to do with Adagrad?

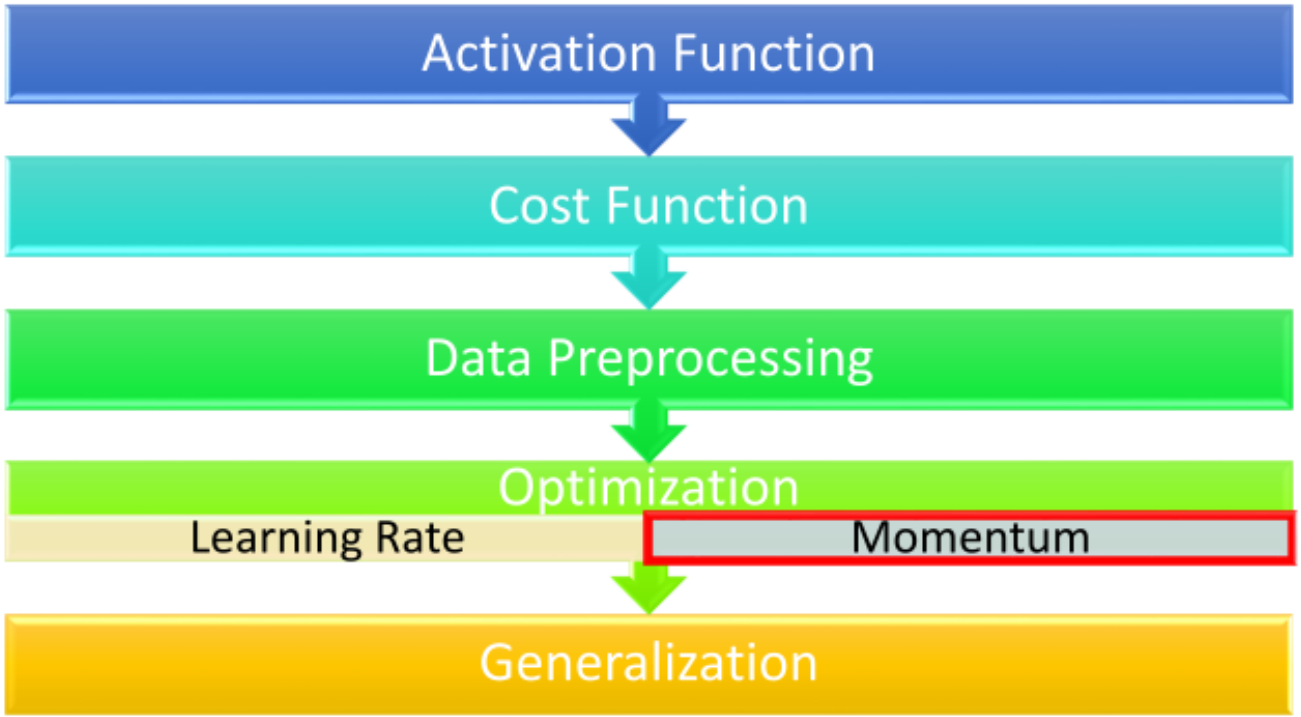
The best step is

$$\frac{|\text{First derivative}|}{\text{Second derivative}}$$

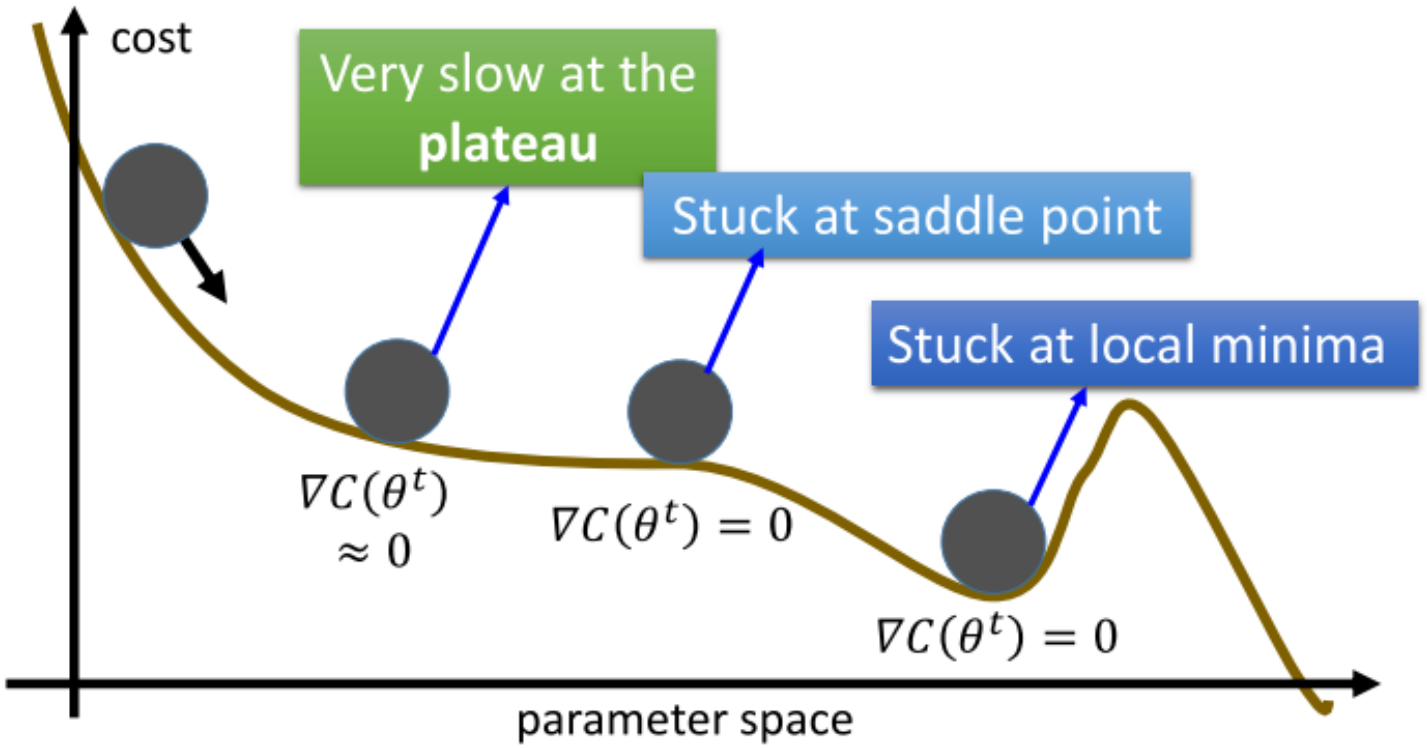
Use *first derivative* to estimate *second derivative*



Outline

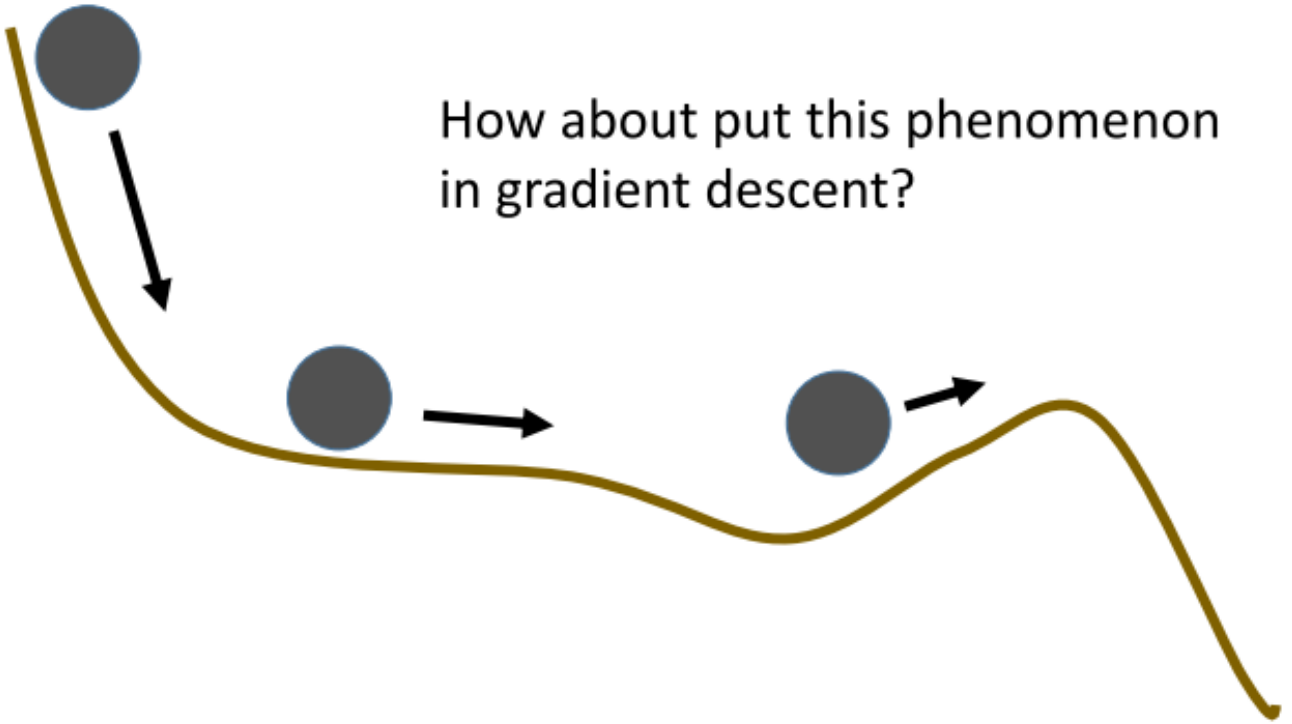

Activation Function
Cost Function
Data Preprocessing
Optimization
Learning Rate
Momentum
Generalization

Easy to stuck



In physical world ……

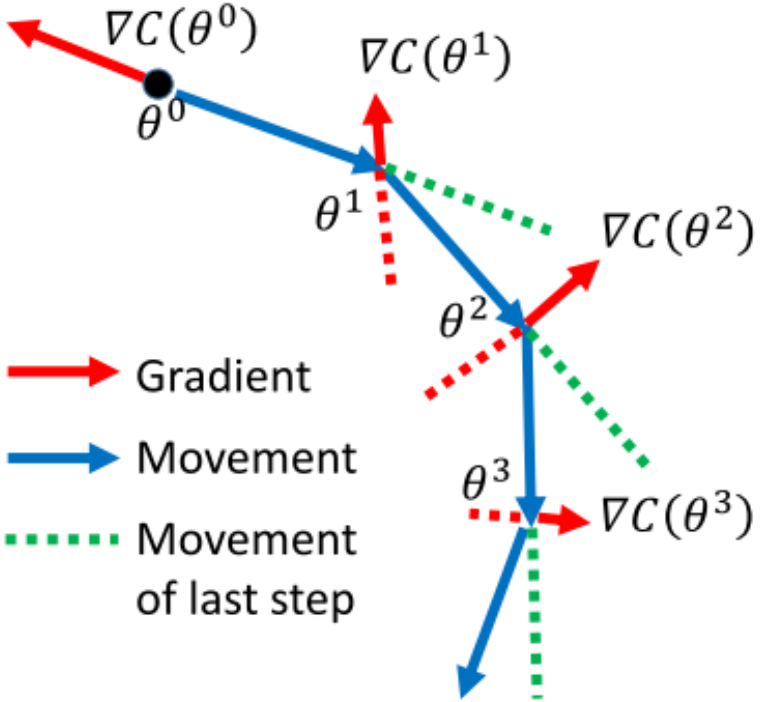
- Momentum



How about put this phenomenon in gradient descent?

Momentum

Movement: movement of last step minus gradient at present



Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla C(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta \nabla C(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement.

Momentum

Movement: movement of last step minus gradient at present

v^i is actually the weighted sum of all the previous gradient: $\nabla C(\theta^0), \nabla C(\theta^1), \ldots \nabla C(\theta^{i-1})$

$v^0 = 0$

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

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

⋮

Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla C(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

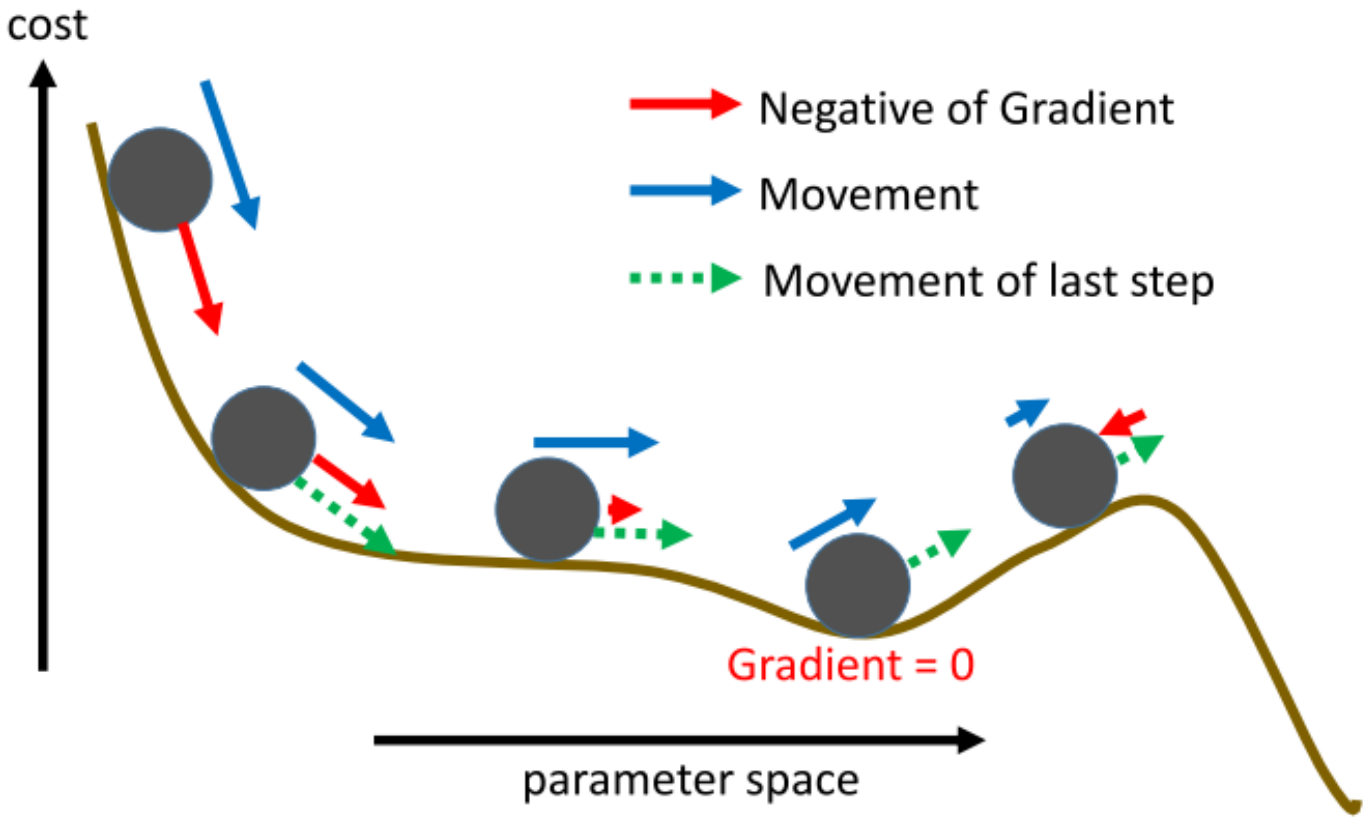
Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta \nabla C(\theta^1)$

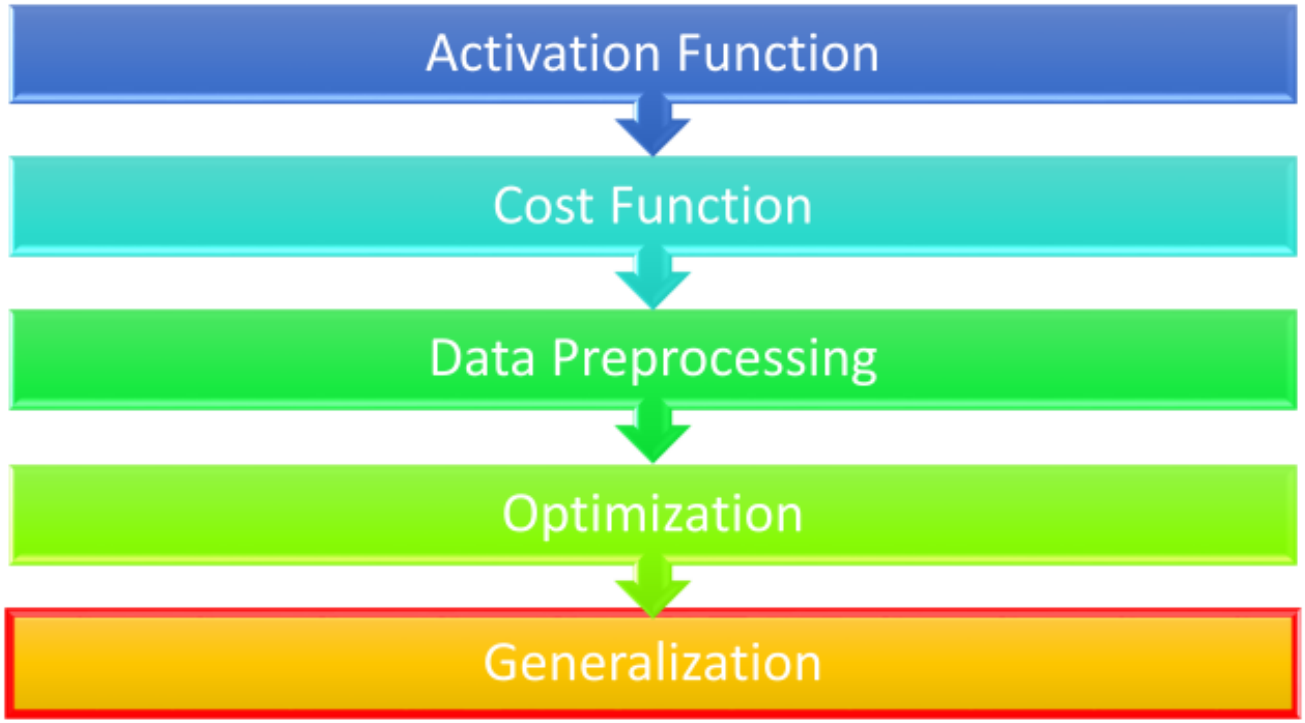
Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement

Momentum


cost
Negative of Gradient
Movement
Movement of last step
Gradient = 0
parameter space

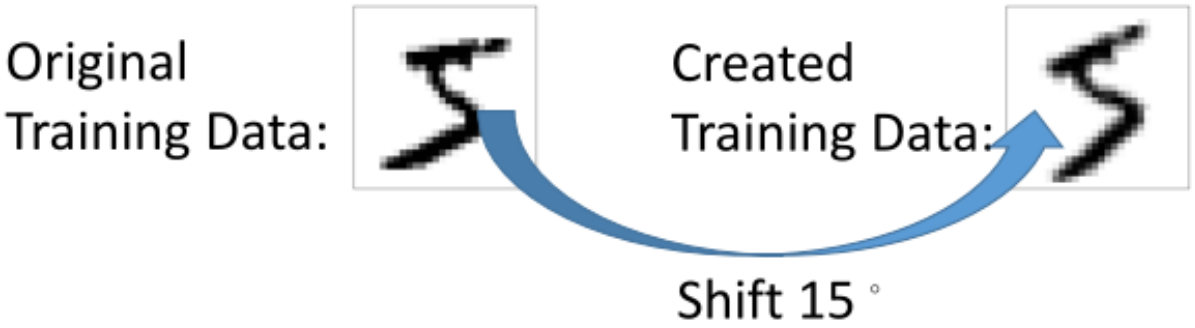
Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

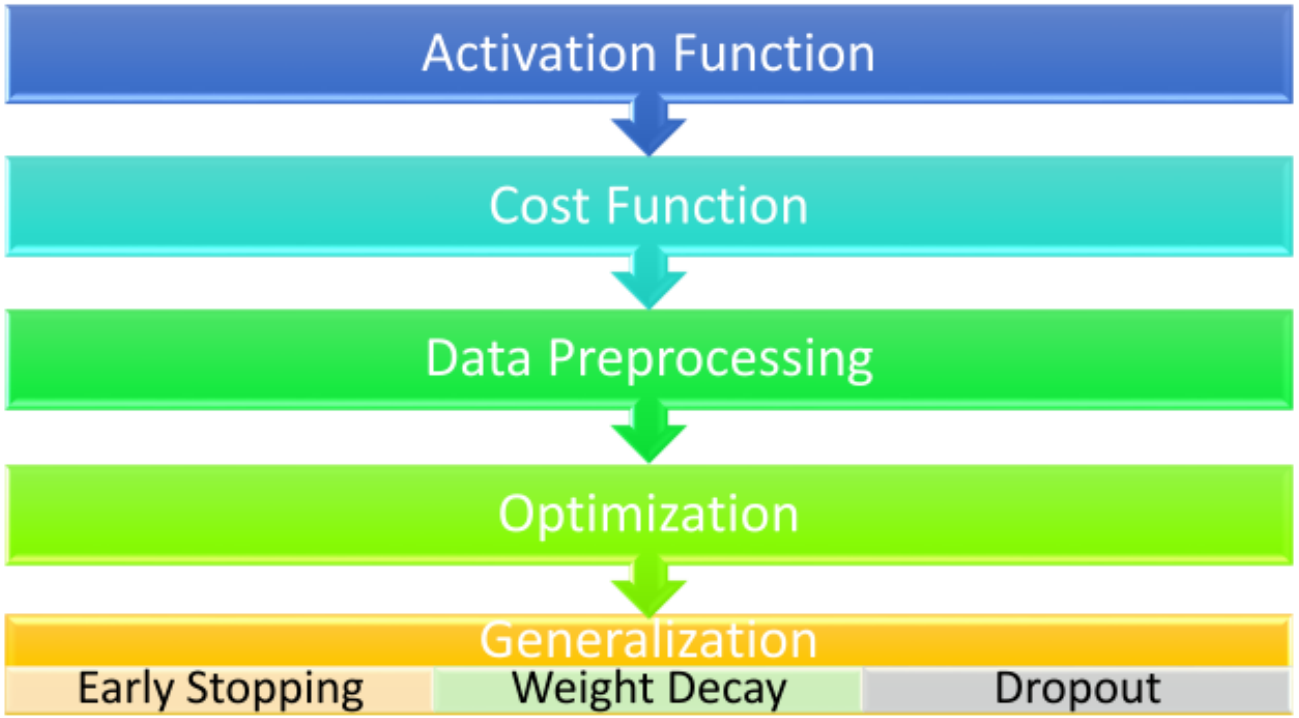
Panacea

- Have more training data
- ***Create*** more training data (?)

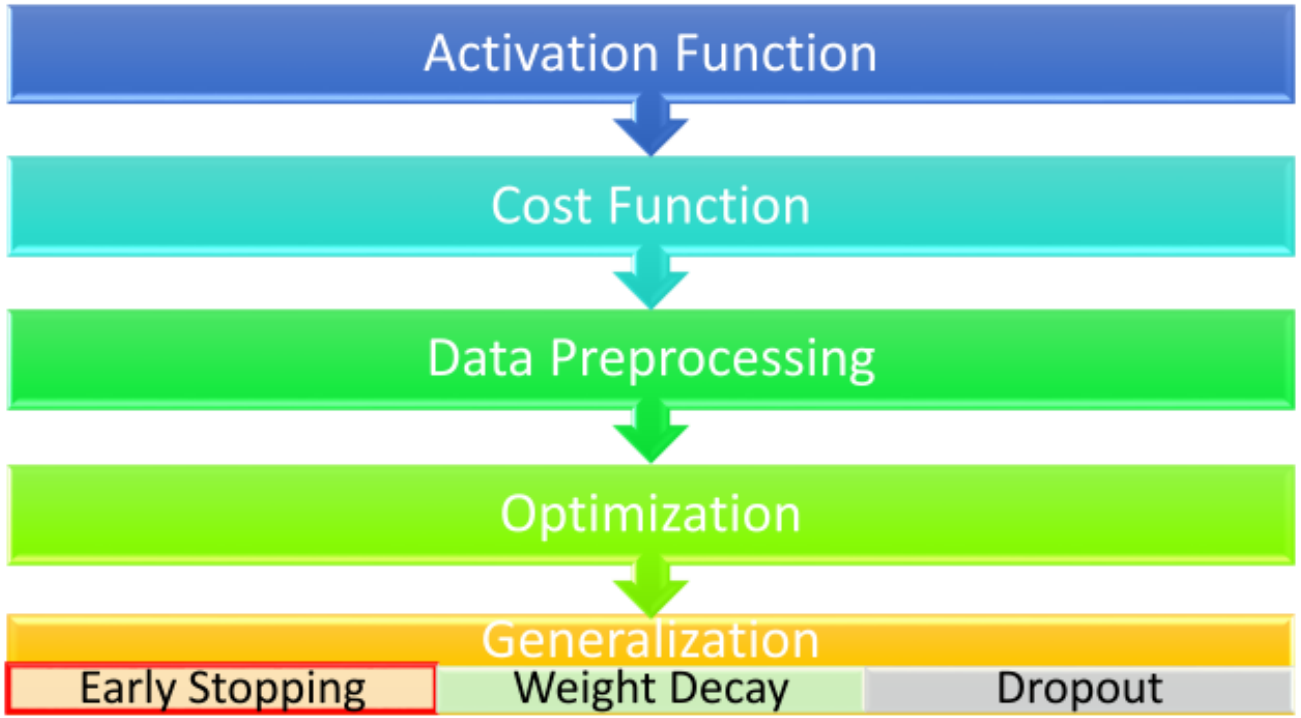
Handwriting recognition:



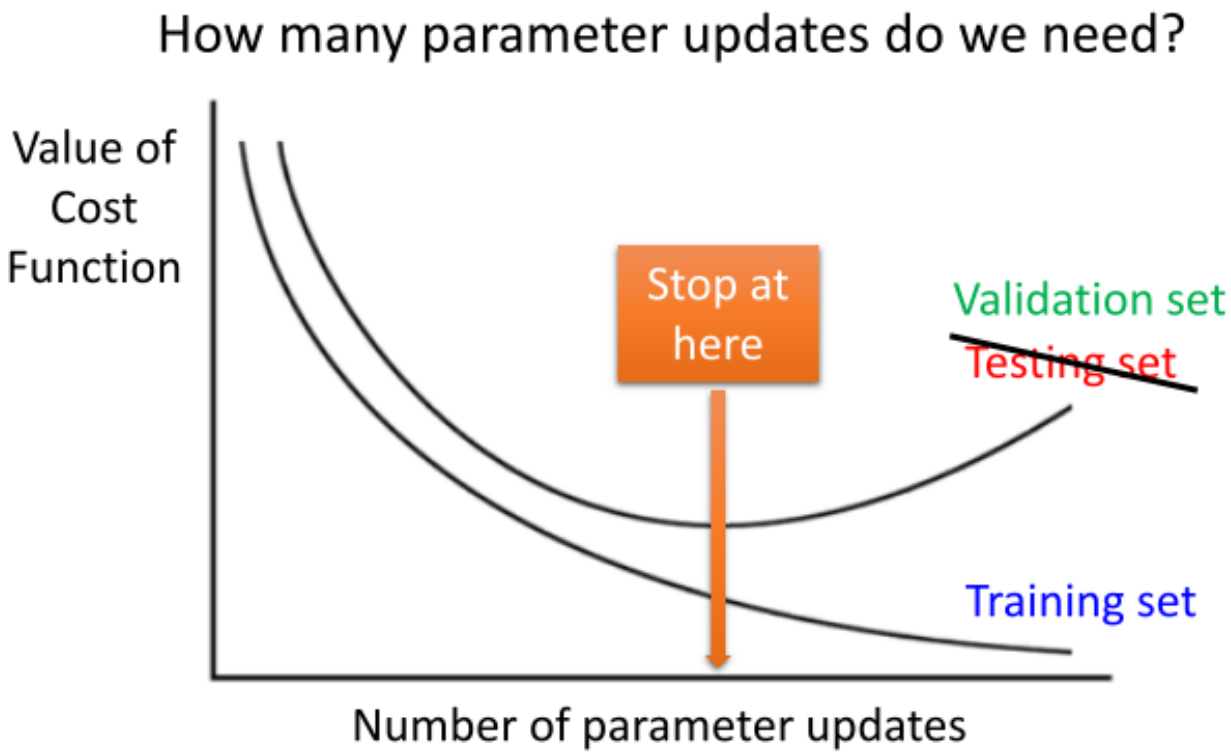
Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization
Early Stopping
Weight Decay
Dropout

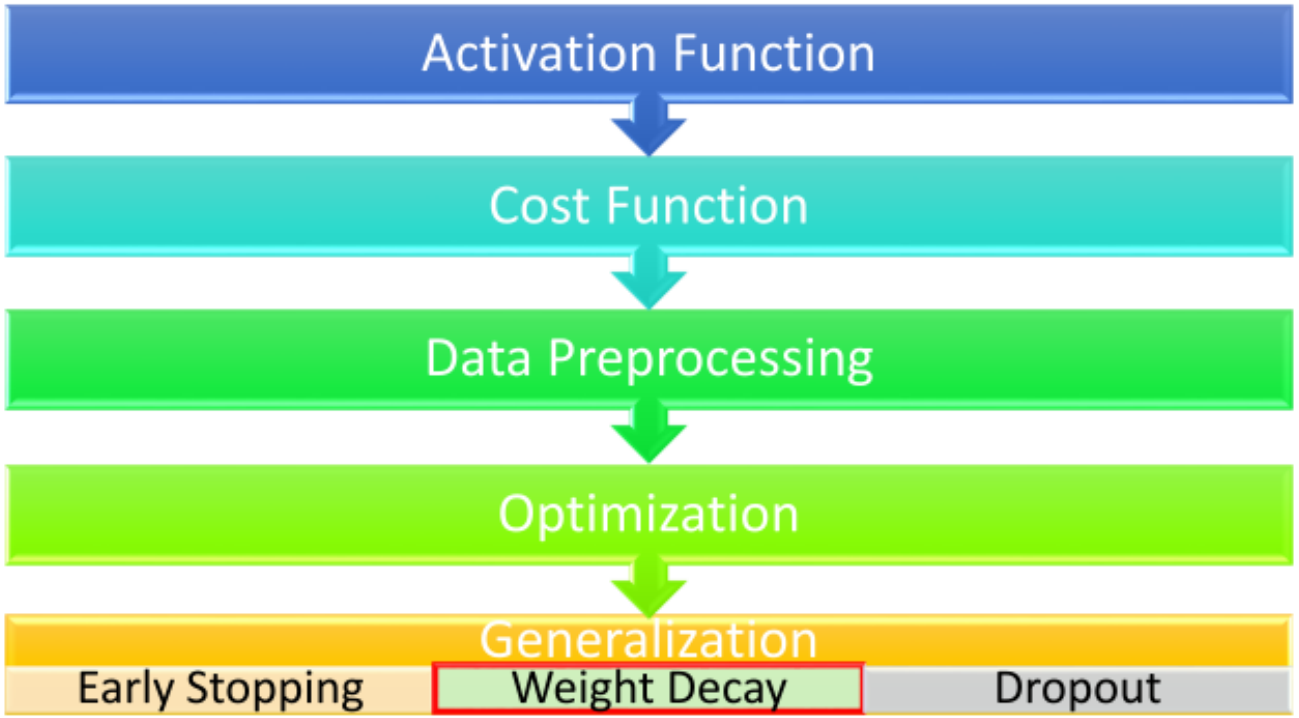
Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization
Early Stopping
Weight Decay
Dropout

Early Stopping



Outline

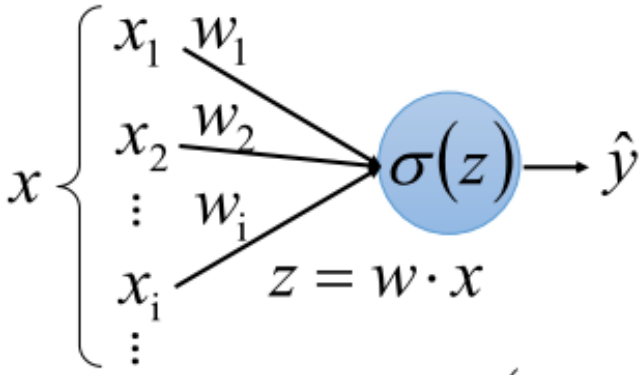

Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization
Early Stopping
Weight Decay
Dropout

Weight Decay

- The parameters closer to zero is preferred.

Training data:

$$\{(x, \hat{y}), \ldots\}$$



Testing data:

$$\{(x', \hat{y}), \ldots\} \qquad x' = x + \varepsilon$$

$$\begin{aligned} z' &= w \cdot (x + \varepsilon) \\ &= w \cdot x + w \cdot \varepsilon \\ &= z + w \cdot \varepsilon \end{aligned}$$

To minimize the effect of noise, we want w close to zero.

Weight Decay

- New cost function to be minimized
  - Find a set of weight not only minimizing original cost but also close to zero

$$C'(\theta) = C(\theta) + \lambda \frac{1}{2} \|\theta\|^2$$

→ Regularization term:

Original cost
(e.g. minimize square error, cross entropy …)

$$\theta = \{W^1, W^2, \ldots\}$$

$$\|\theta\|^2 = \left(w_{11}^1\right)^2 + \left(w_{12}^1\right)^2 + \ldots + \left(w_{11}^2\right)^2 + \left(w_{12}^2\right)^2 + \ldots$$

(not consider biases. why?)

Weight Decay

$$\|\theta\|^2 = \left(w_{11}^1\right)^2 + \left(w_{12}^1\right)^2 + \ldots + \left(w_{11}^2\right)^2 + \left(w_{12}^2\right)^2 + \ldots$$

- New cost function to be minimized

$$C'(\theta) = C(\theta) + \lambda \frac{1}{2}\|\theta\|^2 \quad \text{Gradient:} \quad \frac{\partial C'}{\partial w} = \frac{\partial C}{\partial w} + \lambda w$$

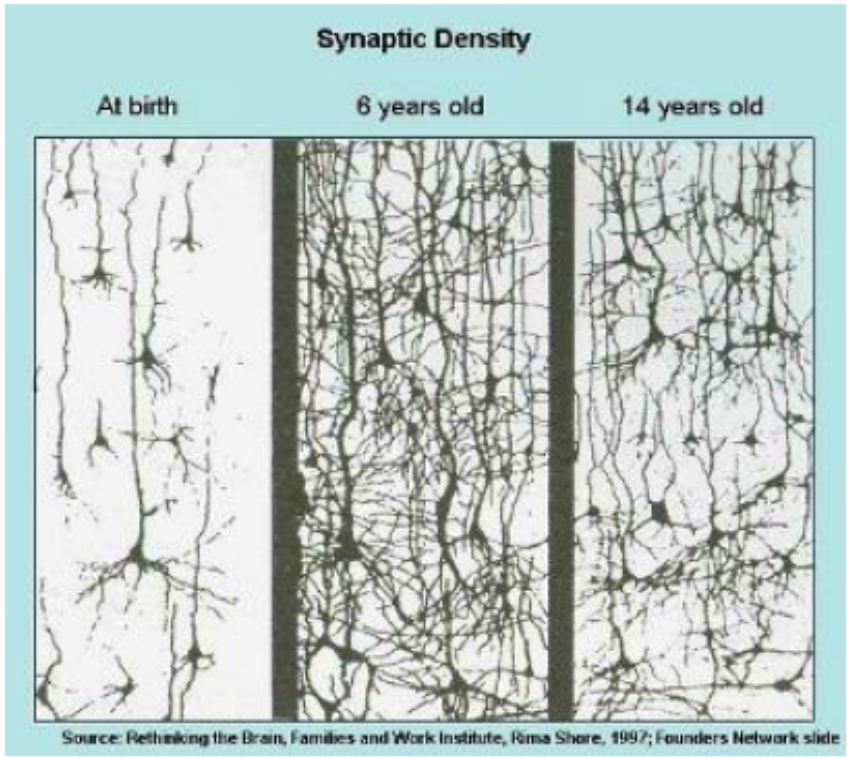
Update: $$w^{t+1} \rightarrow w^t - \eta \frac{\partial C'}{\partial w} = w^t - \eta\left(\frac{\partial C}{\partial w} + \lambda w^t\right)$$

$$= \underbrace{(1-\eta\lambda)w^t} - \eta \underbrace{\frac{\partial C}{\partial w}}$$

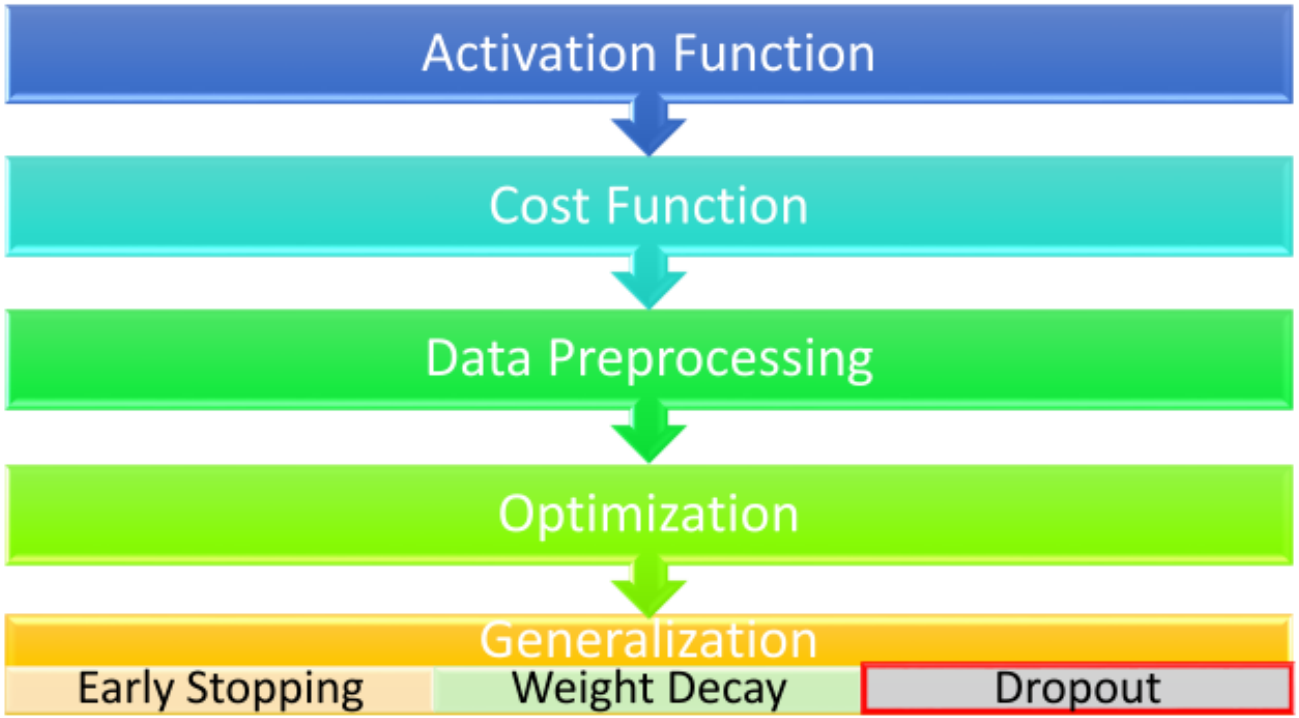
Smaller and smaller

Weight Decay

- Our Brain



Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization
Early Stopping
Weight Decay
Dropout

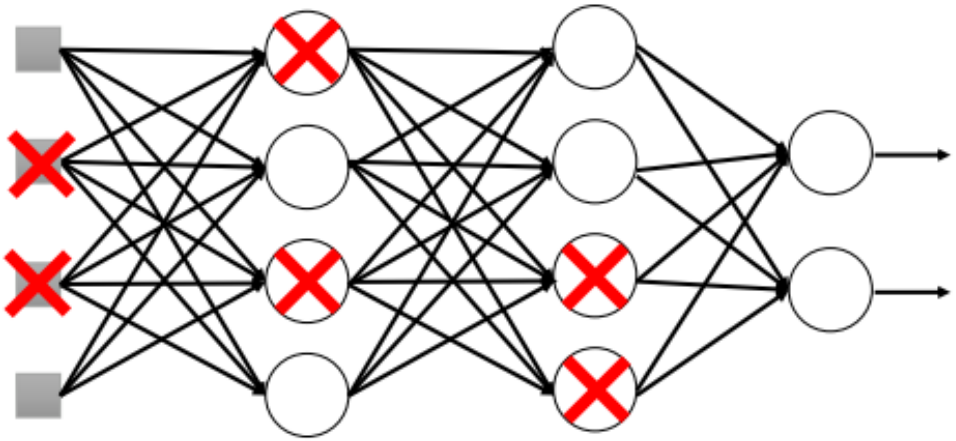
Dropout

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})$$

Training:

x

(stochastic gradient descent)



- In each ***iteration***
  - Each neuron has p% to dropout

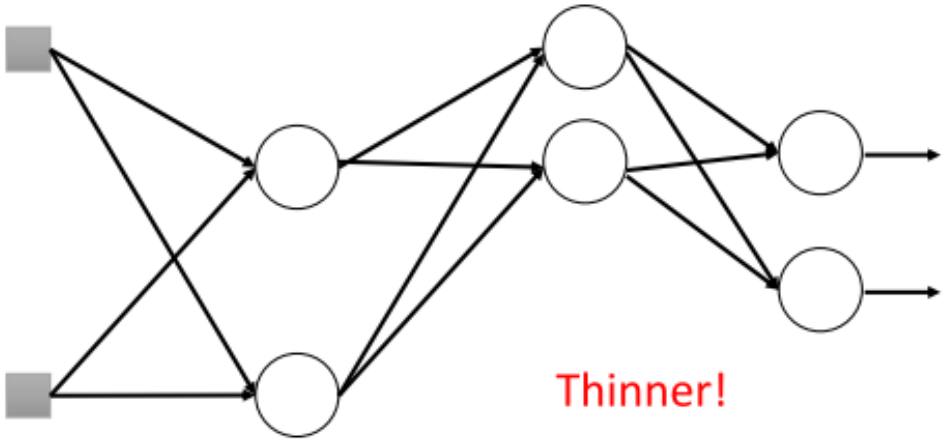
Dropout

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})$$

Training:

x

(stochastic gradient descent)

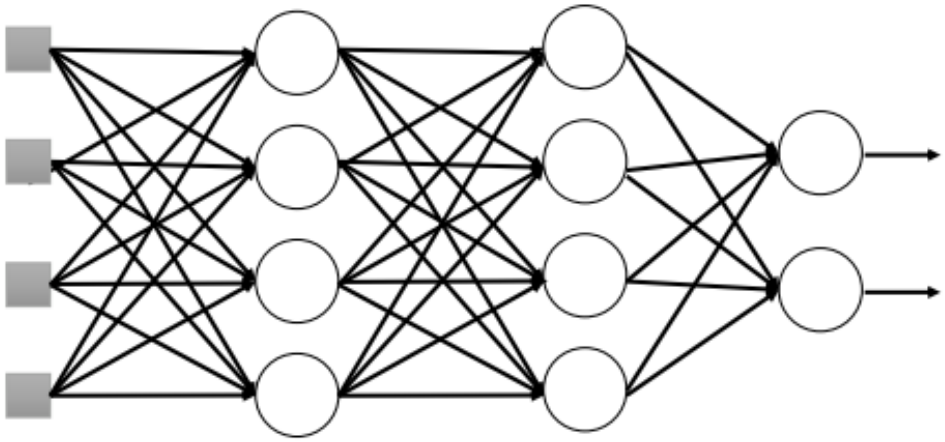


- In each ***iteration***
  - Each neuron has p% to dropout
    - **The structured of the network is changed.**
  - Using the new network for training

For each iteration, we resample the dropout neurons

Dropout

Testing:



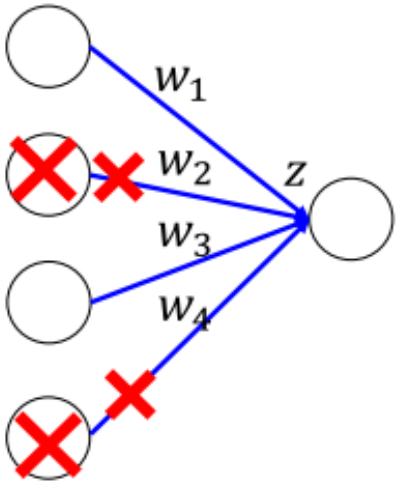
- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If $w_{ij}^{l} = 1$ from training, set $w_{ij}^{l} = 0.5$ for testing.

Dropout
- Intuitive Reason

- Why the weights should multiply (1-p)% (dropout rate) when testing?

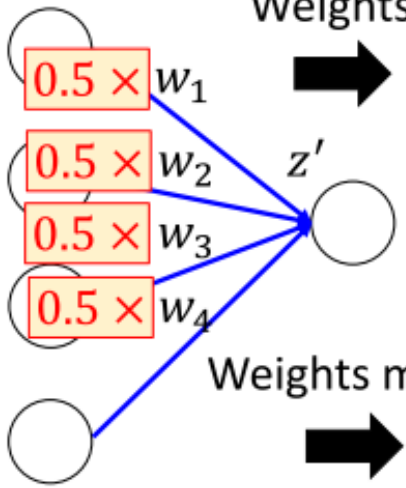
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



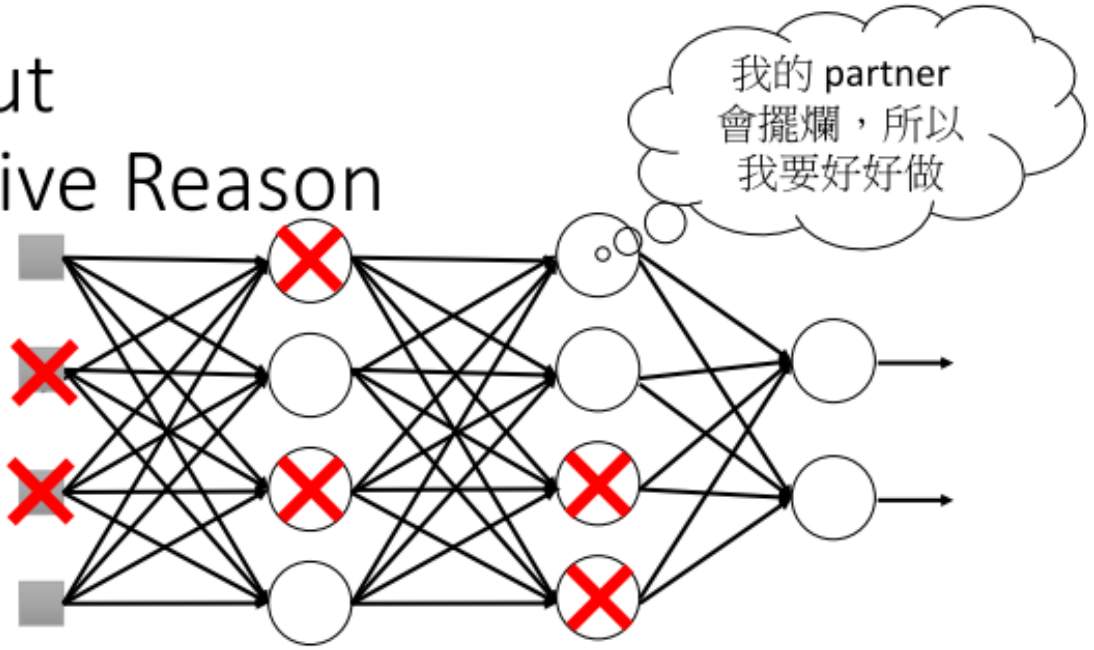
Weights from training

$z' \approx 2z$

Weights multiply (1-p)%

$z' \approx z$

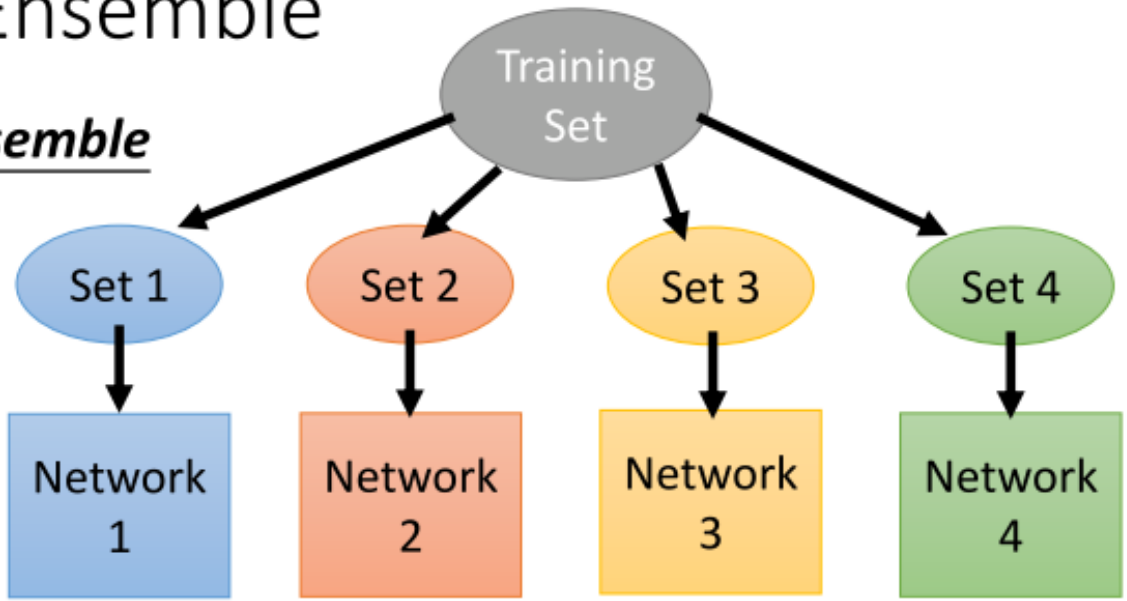
Dropout
- Intuitive Reason



- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout
- Ensemble

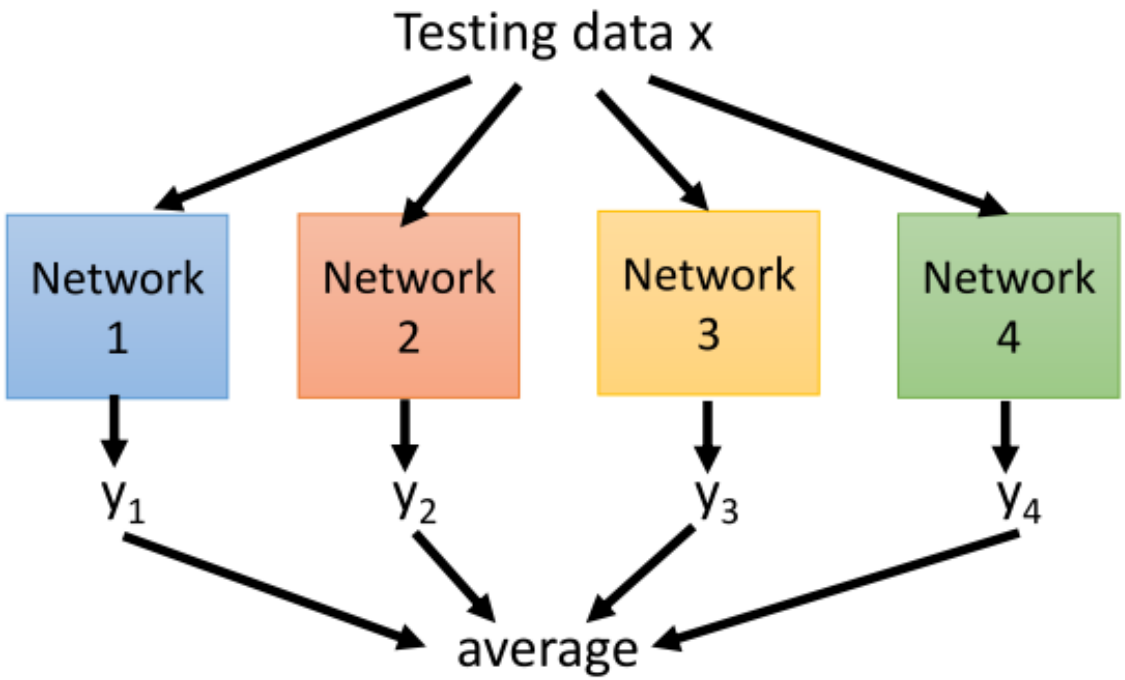
Ensemble



Train a bunch of networks with different structures

Dropout
- Ensemble

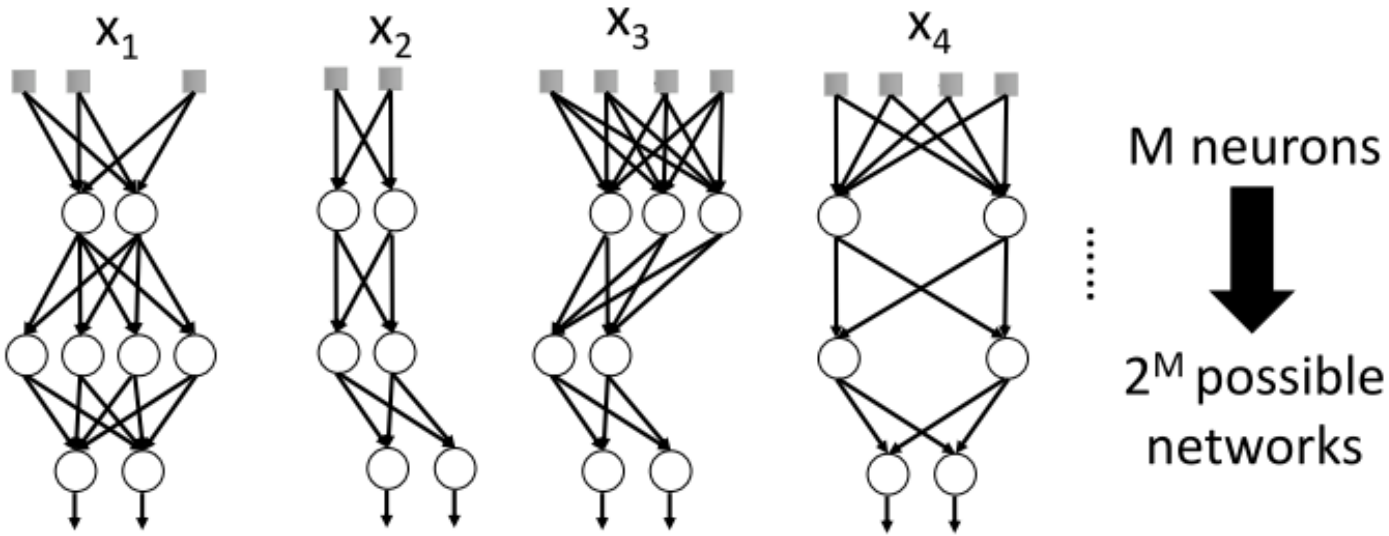
Ensemble



Dropout - Ensemble

Dropout ≈ ***Ensemble***.

Training of Dropout



- Using one data to train one network
- Some parameters in the network are shared

Dropout - Ensemble

Dropout ≈ ***Ensemble***.

Testing of Dropout