

Lecture I: Introduction to Deep Learning

National Cheng Kung University (國立成功大學)
Dept. of Computer Science and Information Engineering
Robotics Lab.

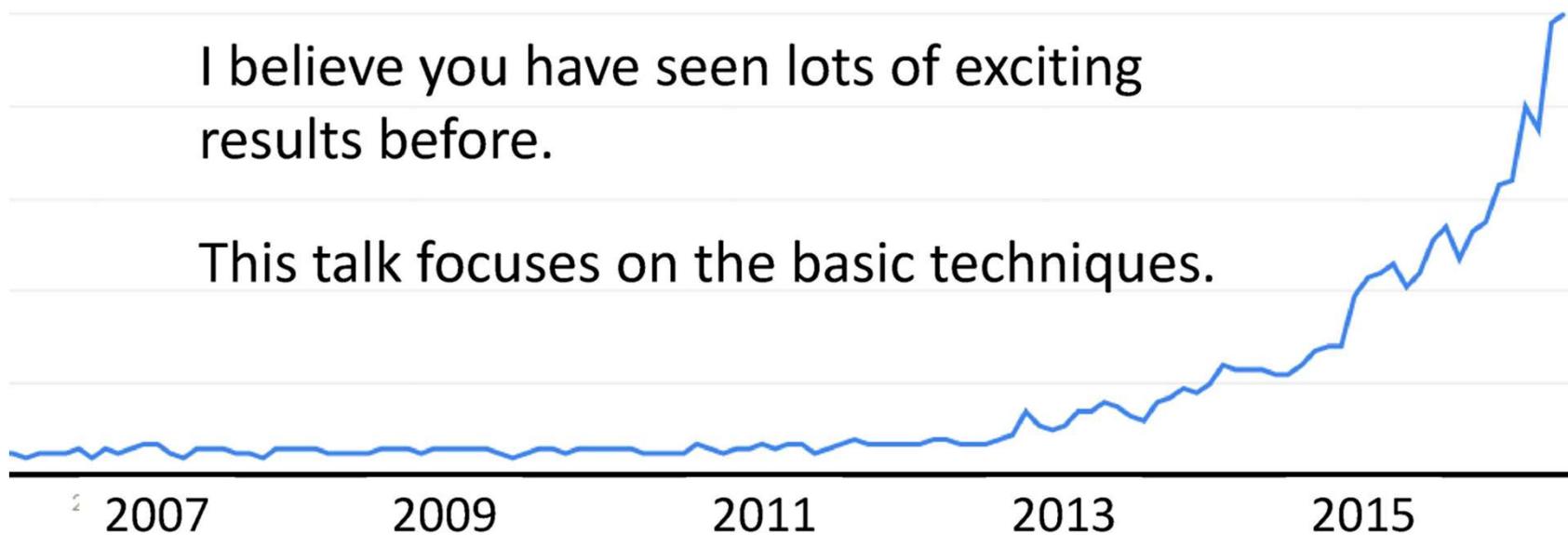
Jenn-Jier James Lien / 連震杰
<http://robotics.csie.ncku.edu.tw>
jjlien@csie.ncku.edu.tw

Deep learning attracts lots of attention.

- Google Trends

I believe you have seen lots of exciting results before.

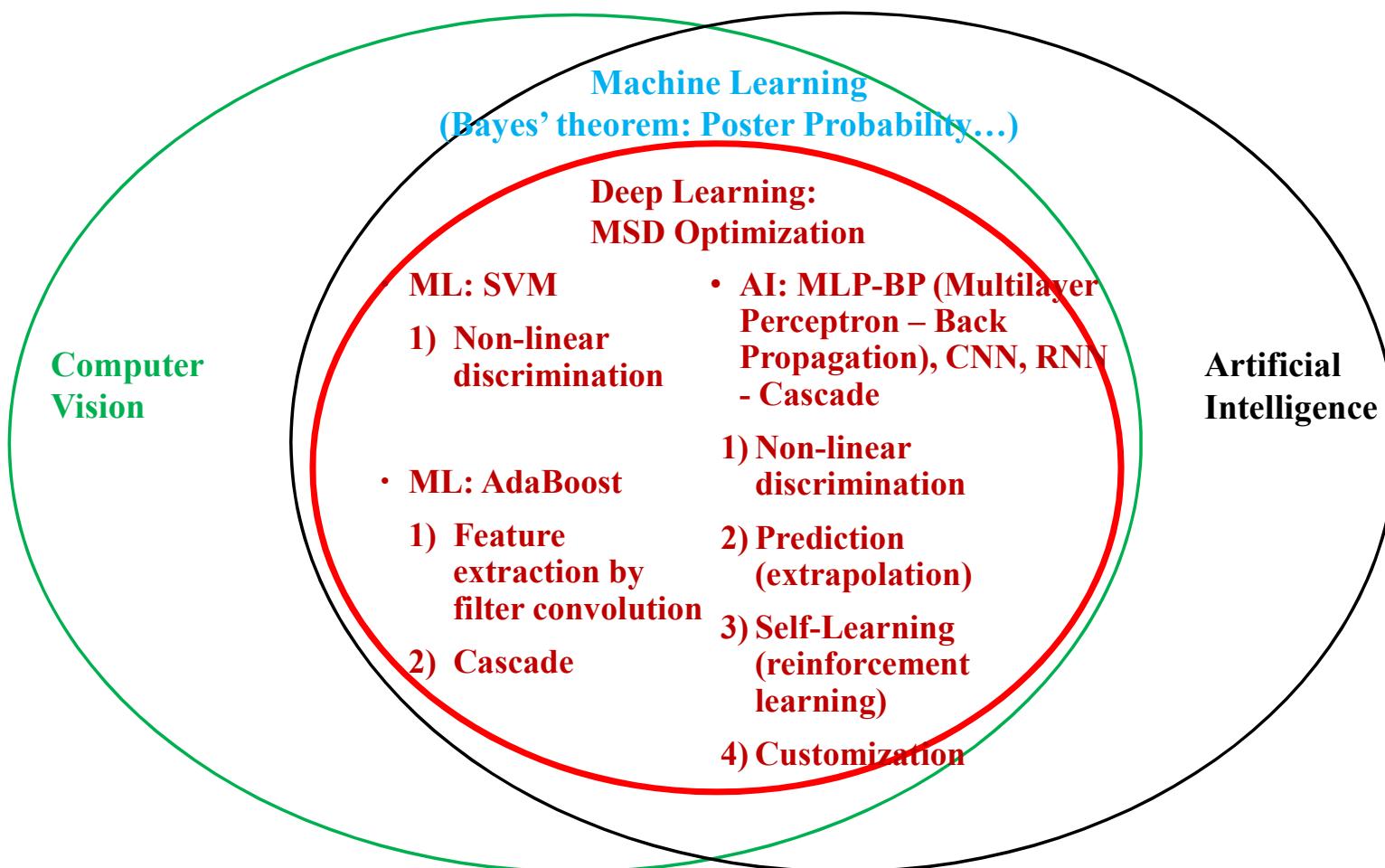
This talk focuses on the basic techniques.



Deep Learning: Computer Vision Vs. Artificial Intelligence Network

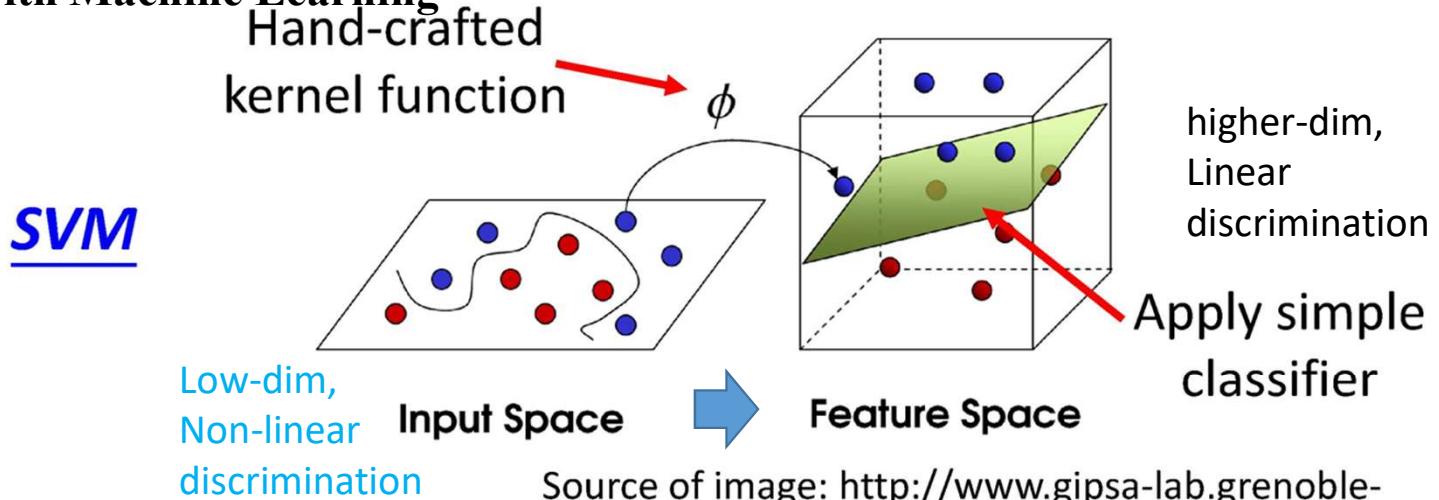
Machine Learning: Need Data + Label

Deep Learning: Need Big Data + Label

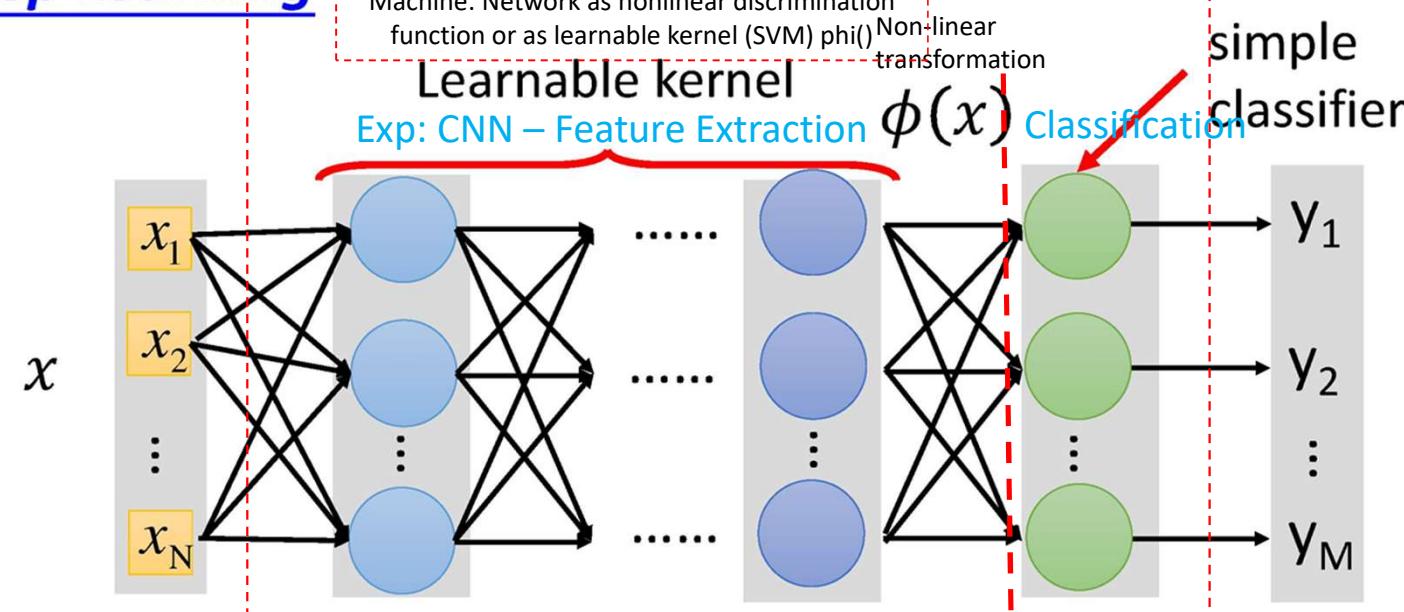


Deep Learning with Machine Learning

J: Without
cascade
concept



J: With cascade
concept



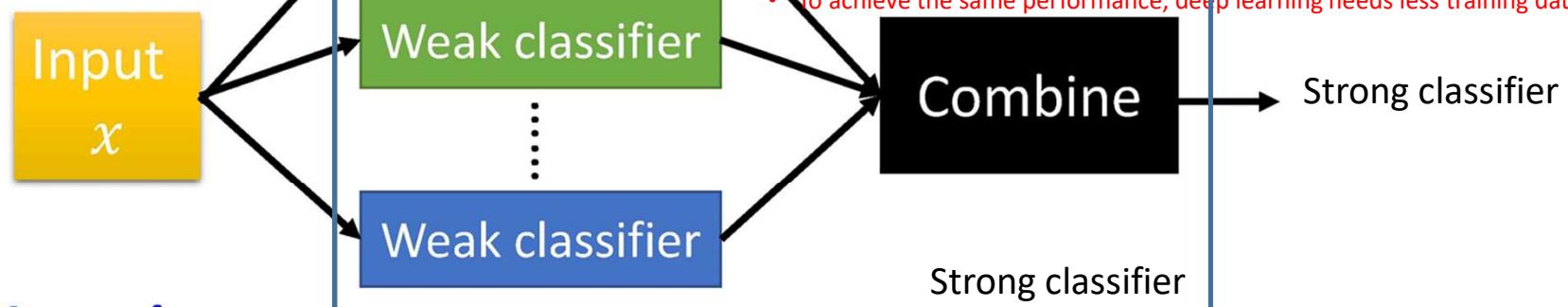
Deep Learning from ML:
Networks as

- 1) Non-linear discrimination:
As learnable kernel (SVM)
 $\phi()$
- 2) AdaBoost Cascade: As
boosted weak classifier to
be strong classifier

Deep Learning with Machine Learning

Cascade: Strong classifiers are cascaded

Boosting



Deep Learning

- One weak classifier is one haar filter
- is also as one neuron unit
- One strong classifier is as one layer

Machine: Network as nonlinear discrimination function or as learnable kernel (SVM) phi()

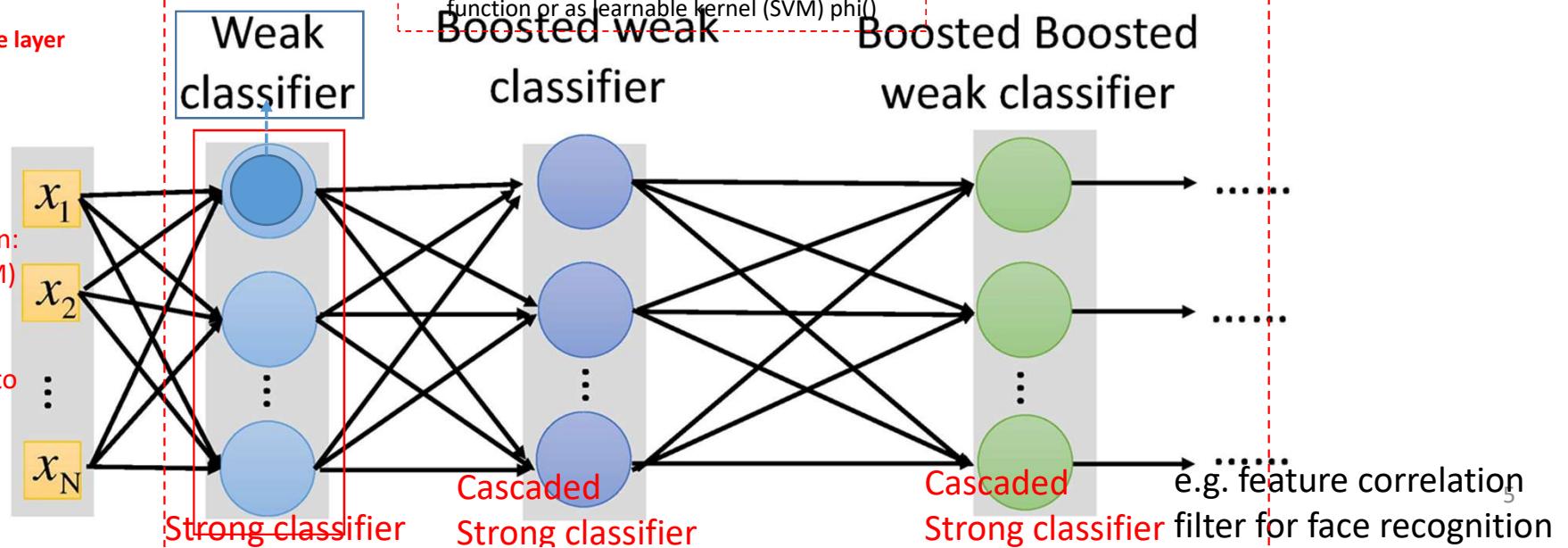
$\phi()$
Boosted
weak classifier

Deep Learning from ML:

Networks as

- 1) Non-linear discrimination:
As learnable kernel (SVM)
 $\phi(\cdot)$

2) AdaBoost Cascade: As boosted weak classifier to be strong classifier

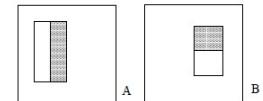


ML: AdaBoost - Feature Extraction Using Haar-Like Filter

1) Filter:

- Filter type: Ex. Parameters $-1, +1$,
- can use II (Integral Image)

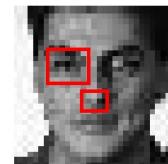
Ex: Haar-like filter



2) Feature:

- a. Filter 的座標位置 (position)
- b. Filter 的大小 (size)

Ex: Eye , nose



1+2) Feature Value (this is one weak classifier):

*
-Feature Value =

Filter \downarrow Feature(Position, Size)

Ex:



Convolution: Linear combination

ML: AdaBoost - Cascade

- ◆ Advantage: Reducing testing computation time.
- ◆ Method: Cascade stages.
 - Each weak classifier: Consist of one Feature Value (this is one weak classifier):
Feature Value = Filter * Feature(Position,Size) -- convolution
 - Each stage corresponds to one strong classifier: Consists of several weak classifiers
 - The training data is the same for each weak classifier at the same strong classifier.
 - After one strong classifier, the training data is reduced – reduce as many negative data as possible.
- ◆ Idea: Reject as many negatives as possible at the earliest stage. More complex classifiers were then used in later stages.
- ◆ The detection process is that of a degenerate decision tree, so called “cascade”.

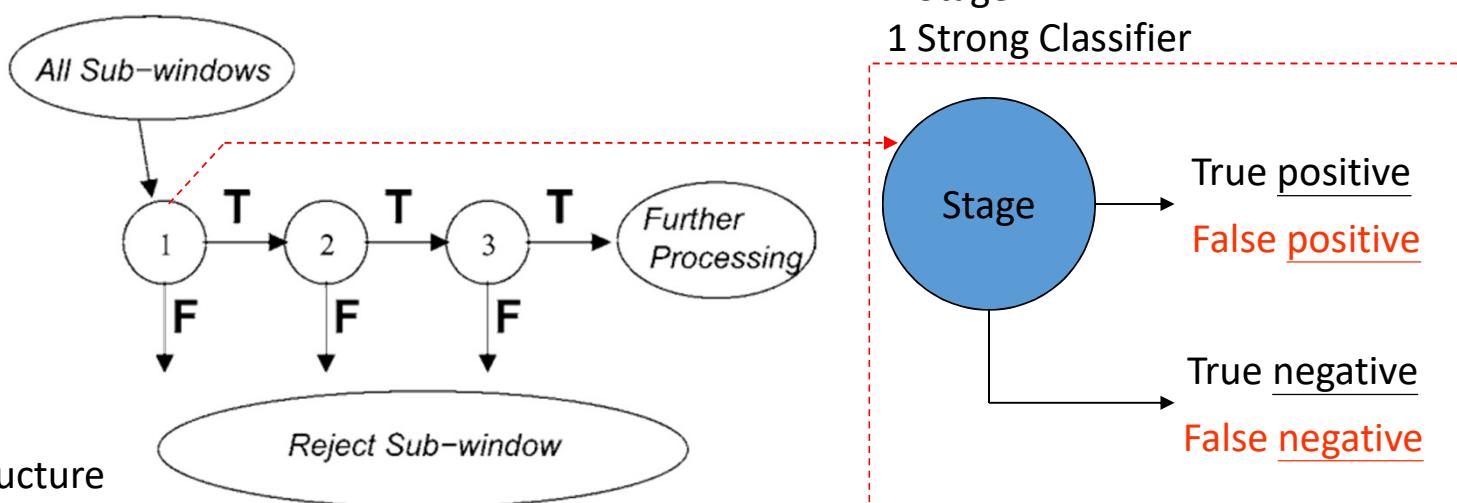
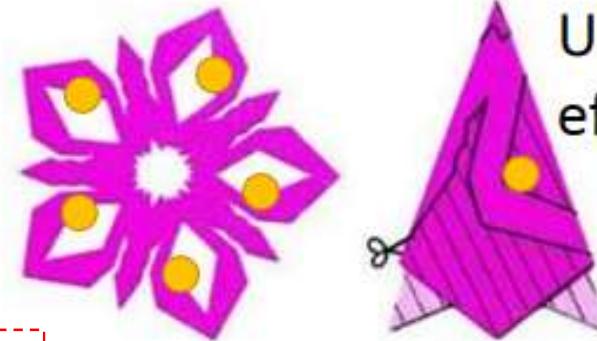


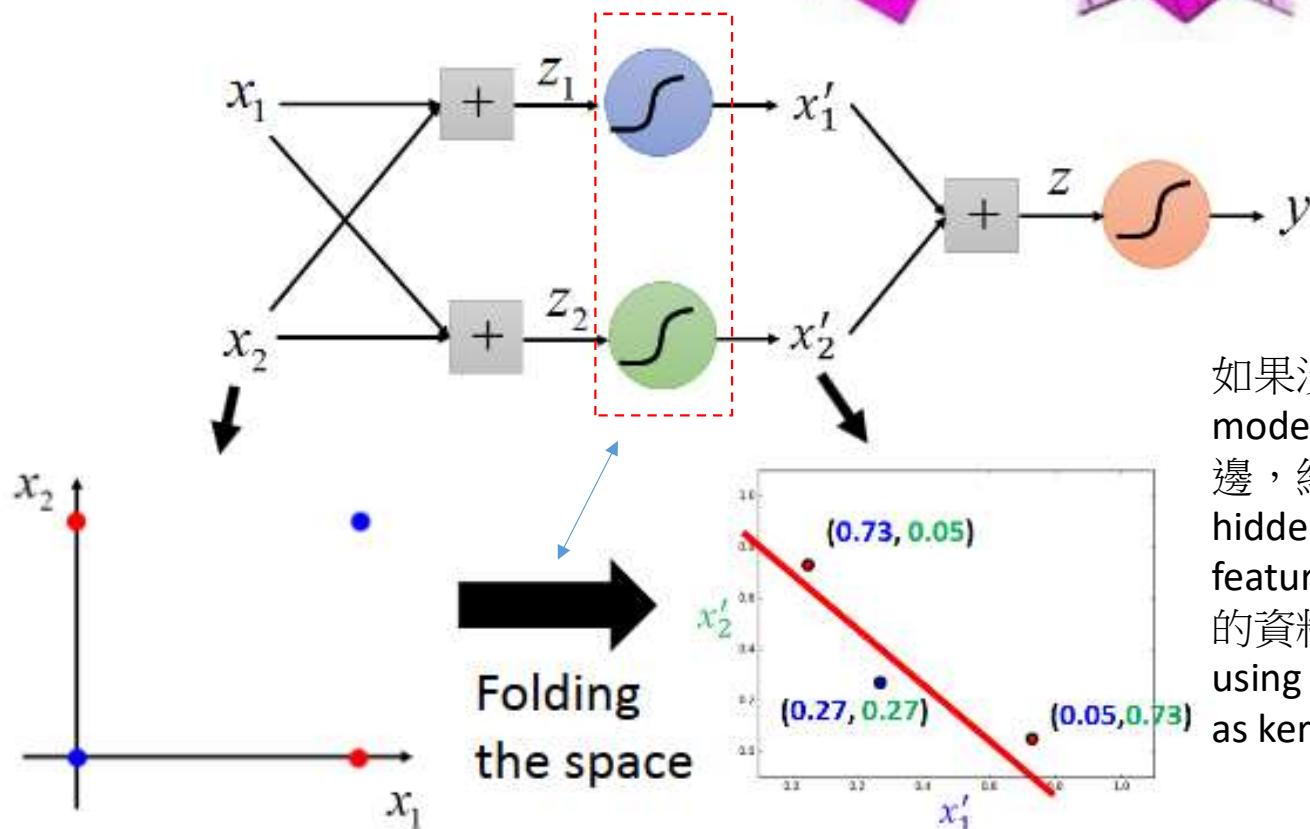
Figure 4: Cascade Structure

□ Important Concept!!

More Analogy



Use data effectively



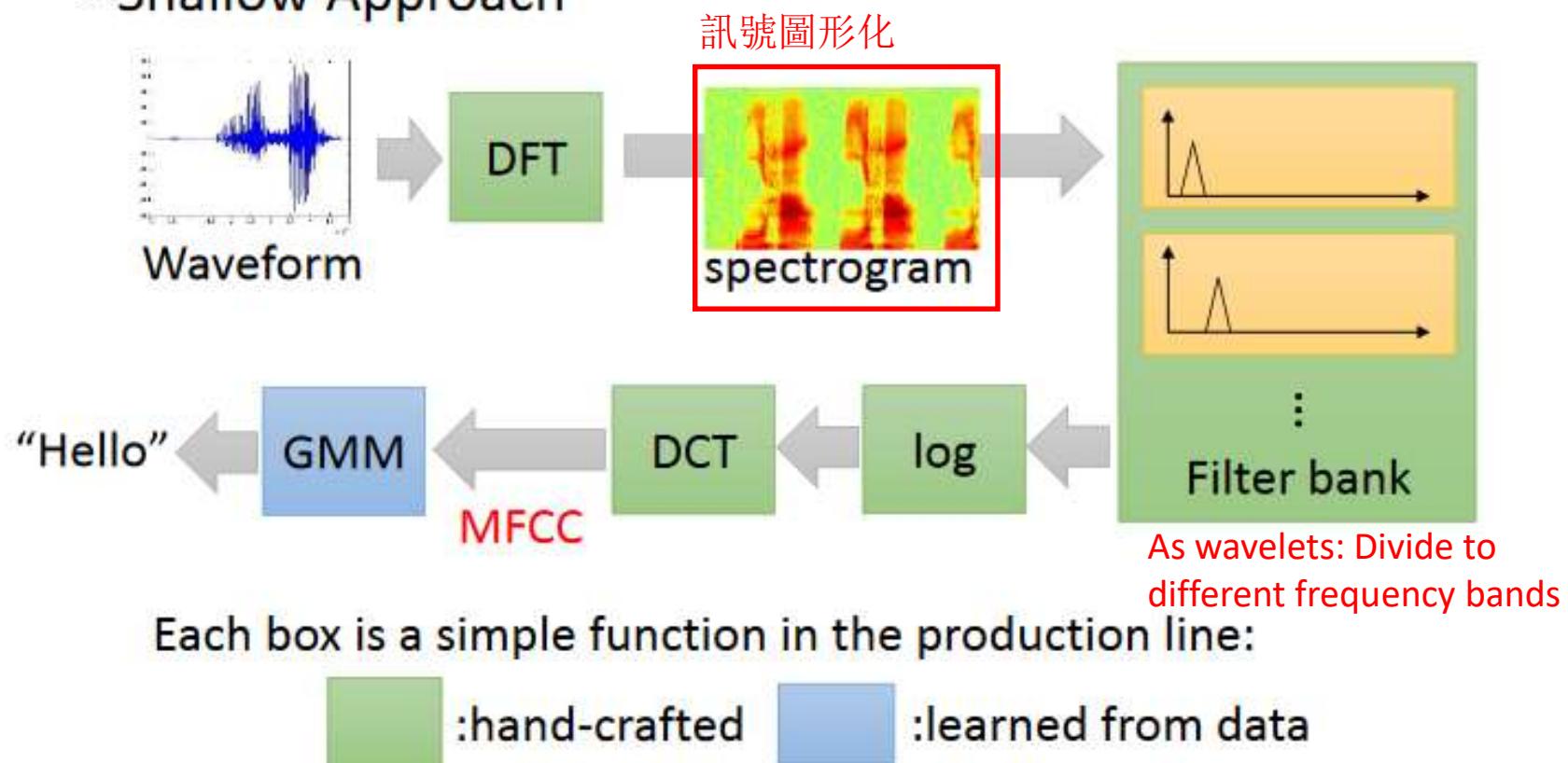
Input: 4 data point ,
the color represent the
classification

如果沒有hidden layer，linear的
model沒有辦法把藍色分在一
邊，紅色分在一邊，當加入了
hidden layer相當於做了一個
feature transformation，將原先
的資料project到另一個space by
using non-linear transformation
as kernel function of SVM。

End-to-end Learning - Speech Recognition

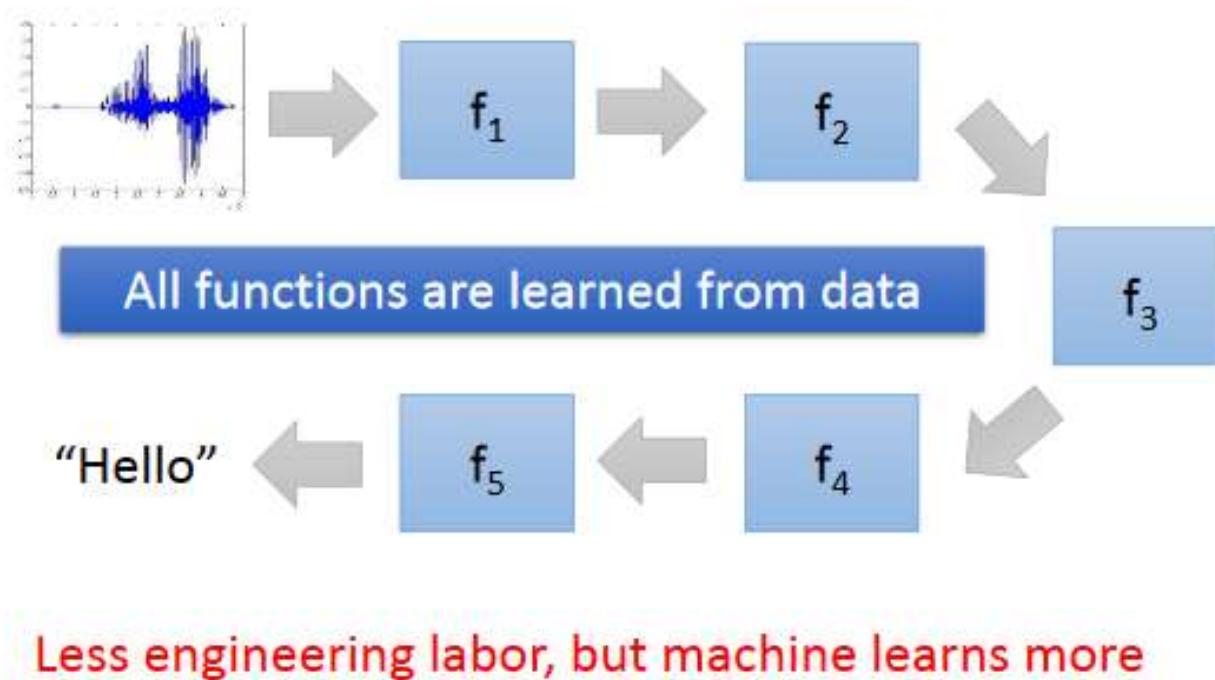
1/2

- Shallow Approach



End-to-end Learning - Speech Recognition

- Deep Learning 2/2



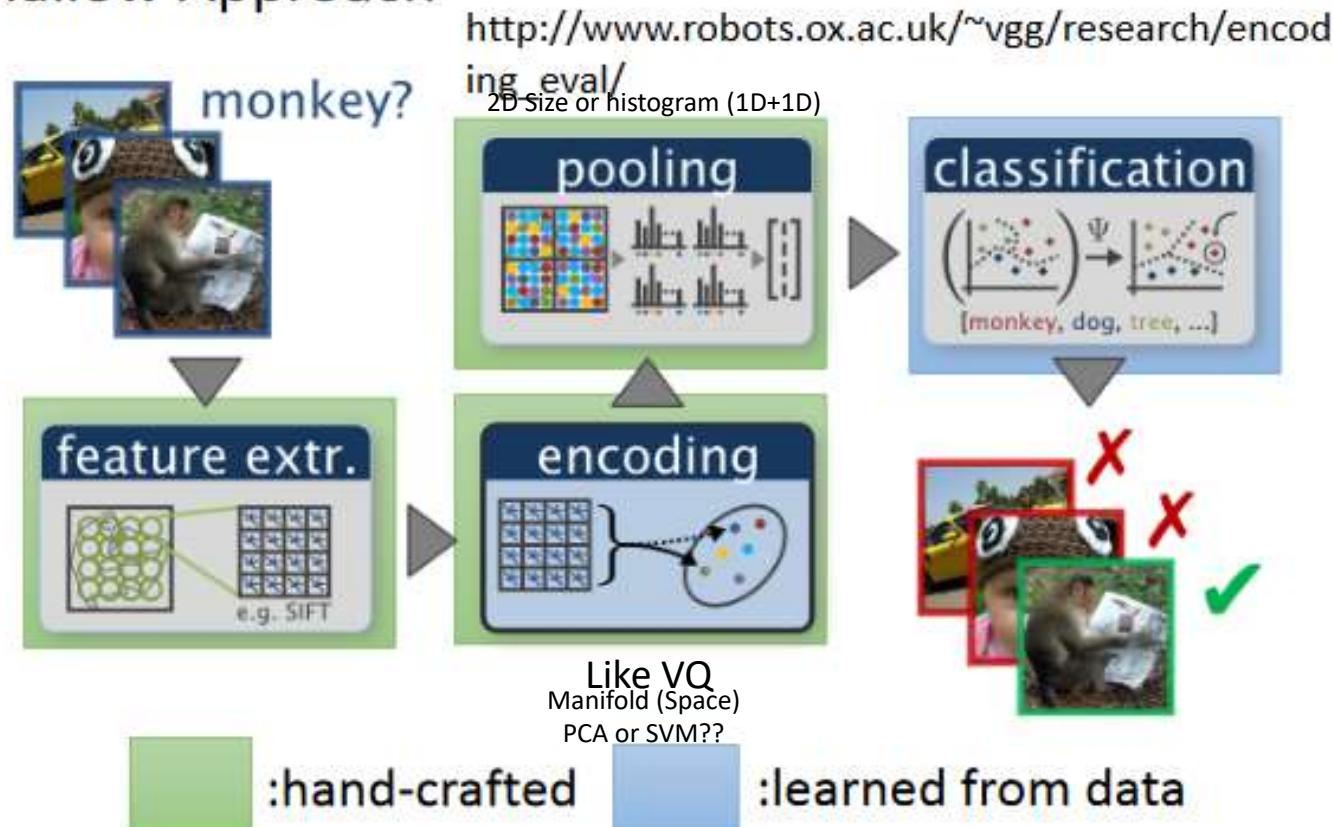
Q: 以聲音訊號為例，有學者提出能不能透過 deep learning，即只有input(聲音訊號)與 output(對應的文字)，完全不對輸入的聲音訊號 (time domain)做feature transform(signal processing)，是否能夠不需要signal processing
A: 有學者提出結論，能夠training出一個model與有將input經過 feature transform的結果持平，但沒有更好。

End-to-end Learning

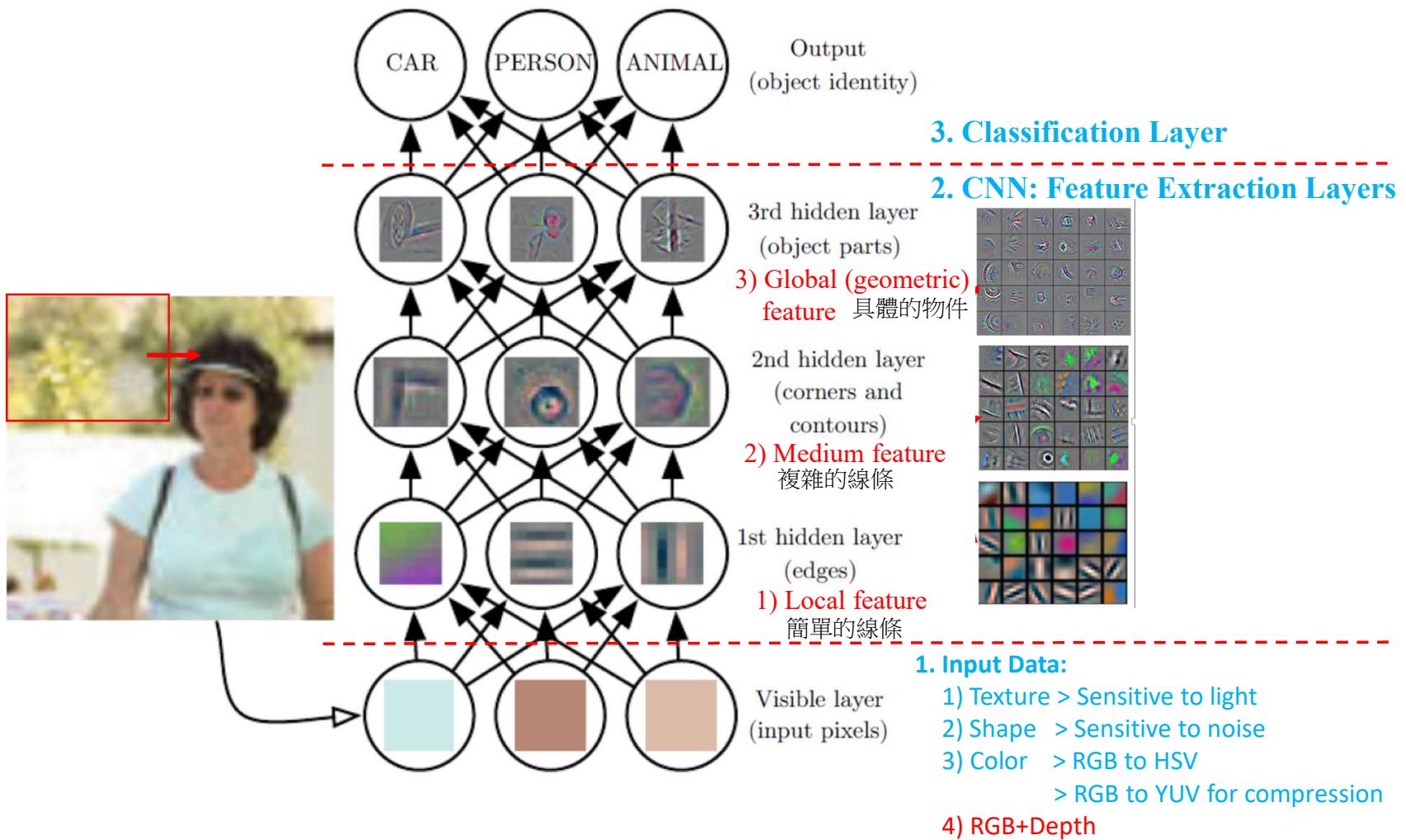
- Image Recognition

1/2

- Shallow Approach



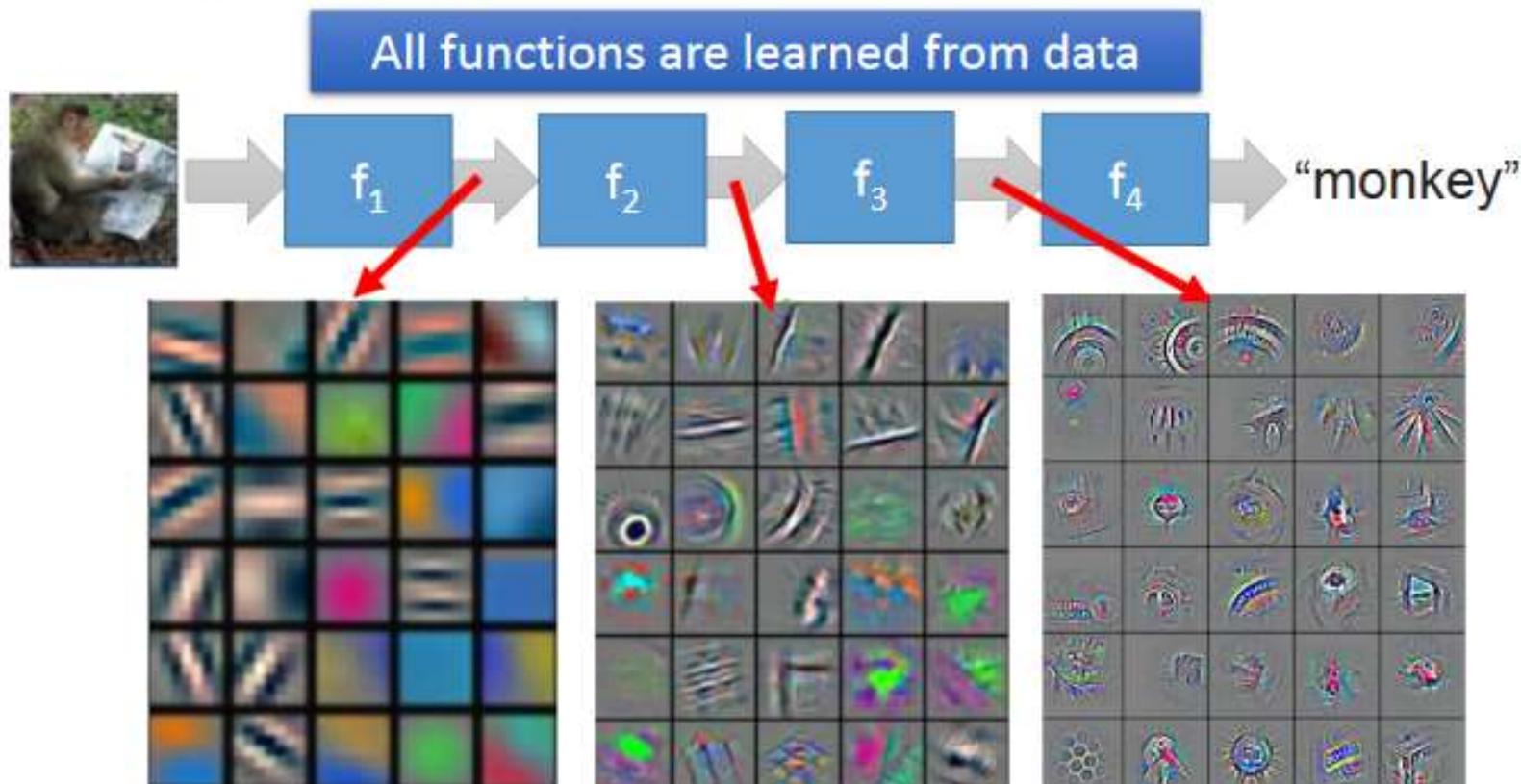
Deep Learning: CNN Classification 2/2



End-to-end Learning - Image Recognition

- Deep Learning

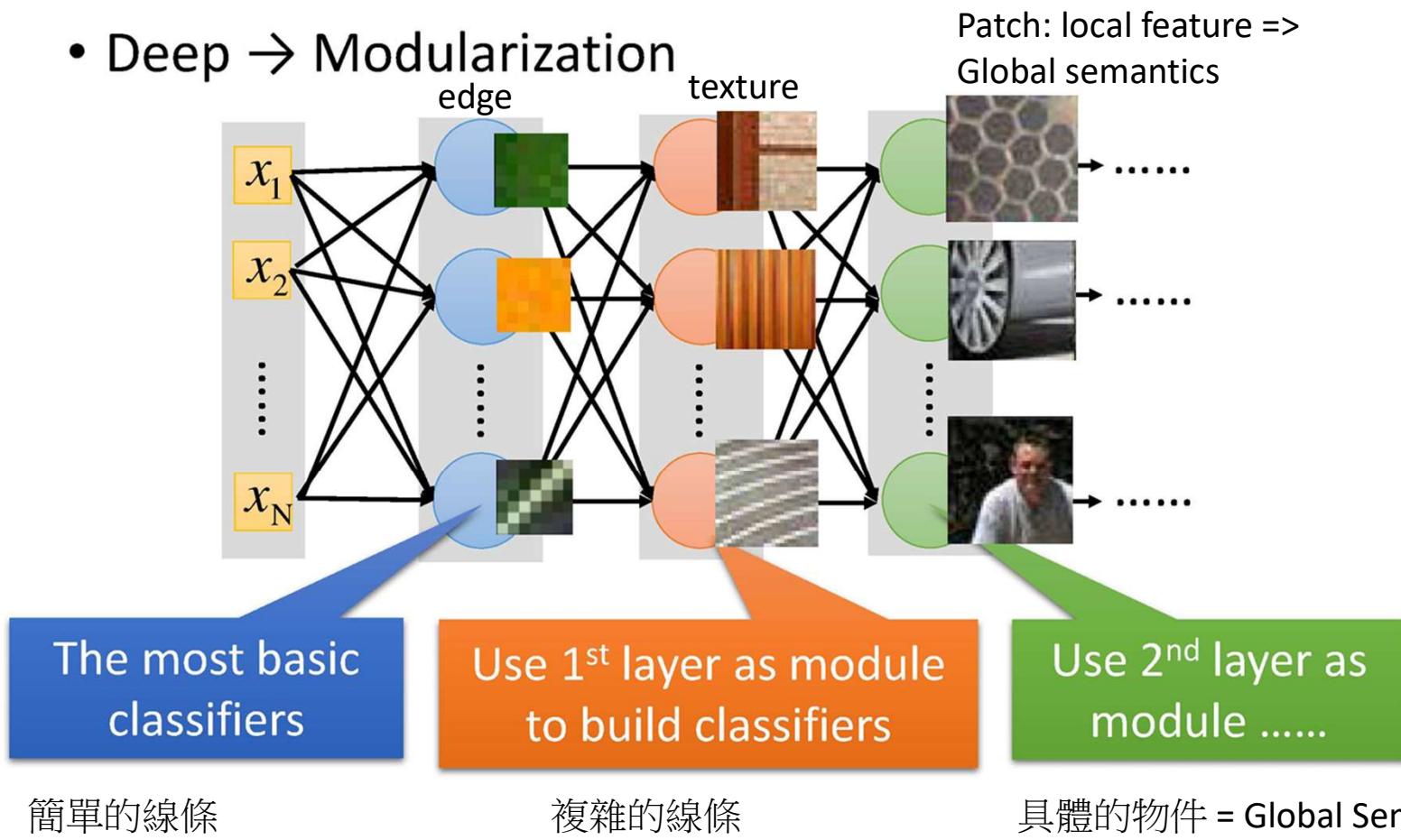
Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818–833)



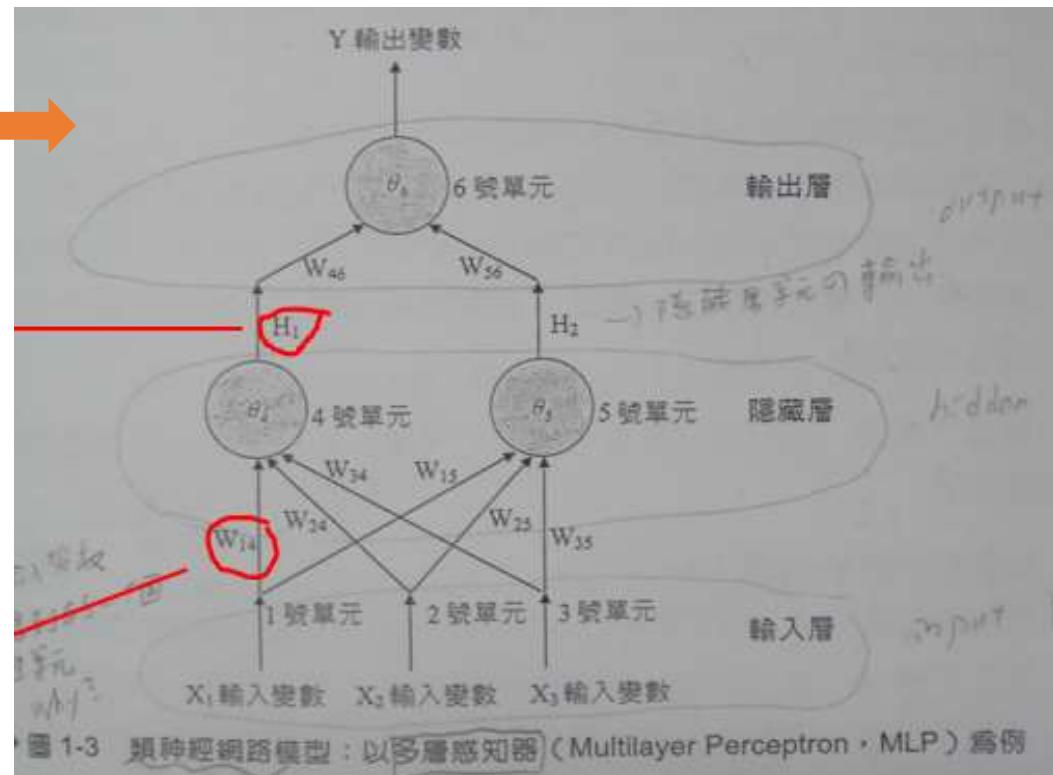
Modularization

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

- Deep → Modularization



MLP-BP



- One weak classifier is one haar filter is also as one neuron unit
- One strong classifier is as one layer

$$y_j = f(\sum_i W_{ij}x_i - \theta_j)$$

Deep Learning: CNN

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

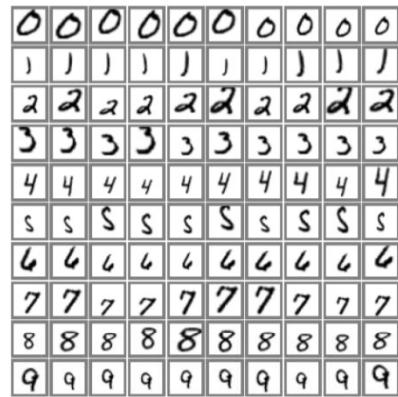


Fig. 7. Examples of distortions of ten training patterns.

Training

CNN:
Extraction
-2 convolution layers
-2 Pooling Layers

Test

Classification



Fig. 4. Size-normalized examples from the MNIST database.

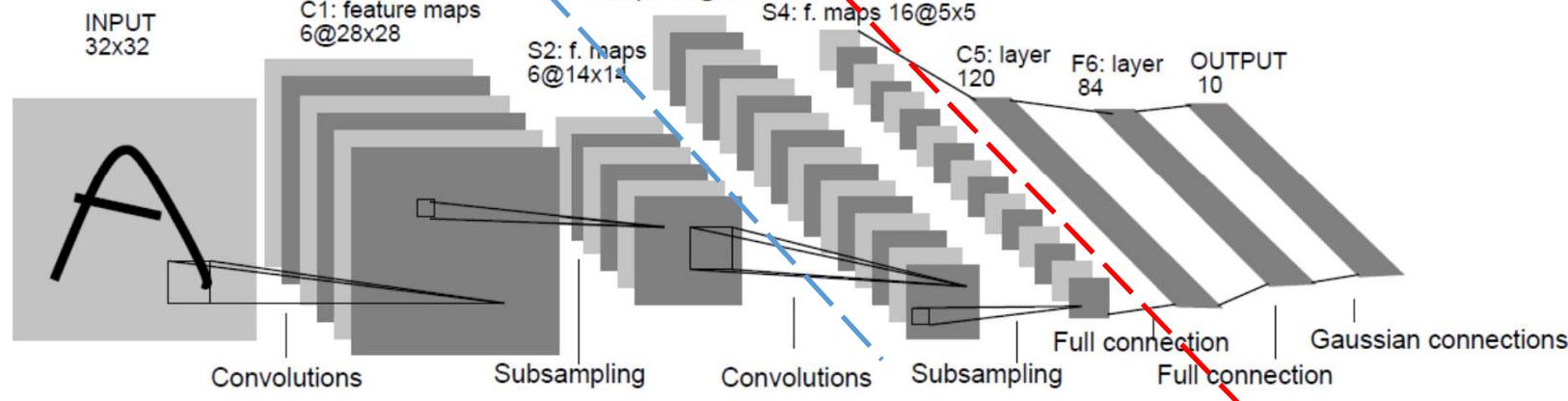


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Outline

- I. **Lecture I: Introduction to Deep Learning**
- II. **Lecture II: Tips for Training Deep Neural Network**
- III. **Lecture III: Variants of Neural Network**
- IV. **Lecture IV: Next Wave**

Lecture I: Introduction ^{to} ~~of~~ Deep Learning

Outline of Lecture 1

I.1

Three Steps for Deep Learning

I.2

Why Deep Learning?

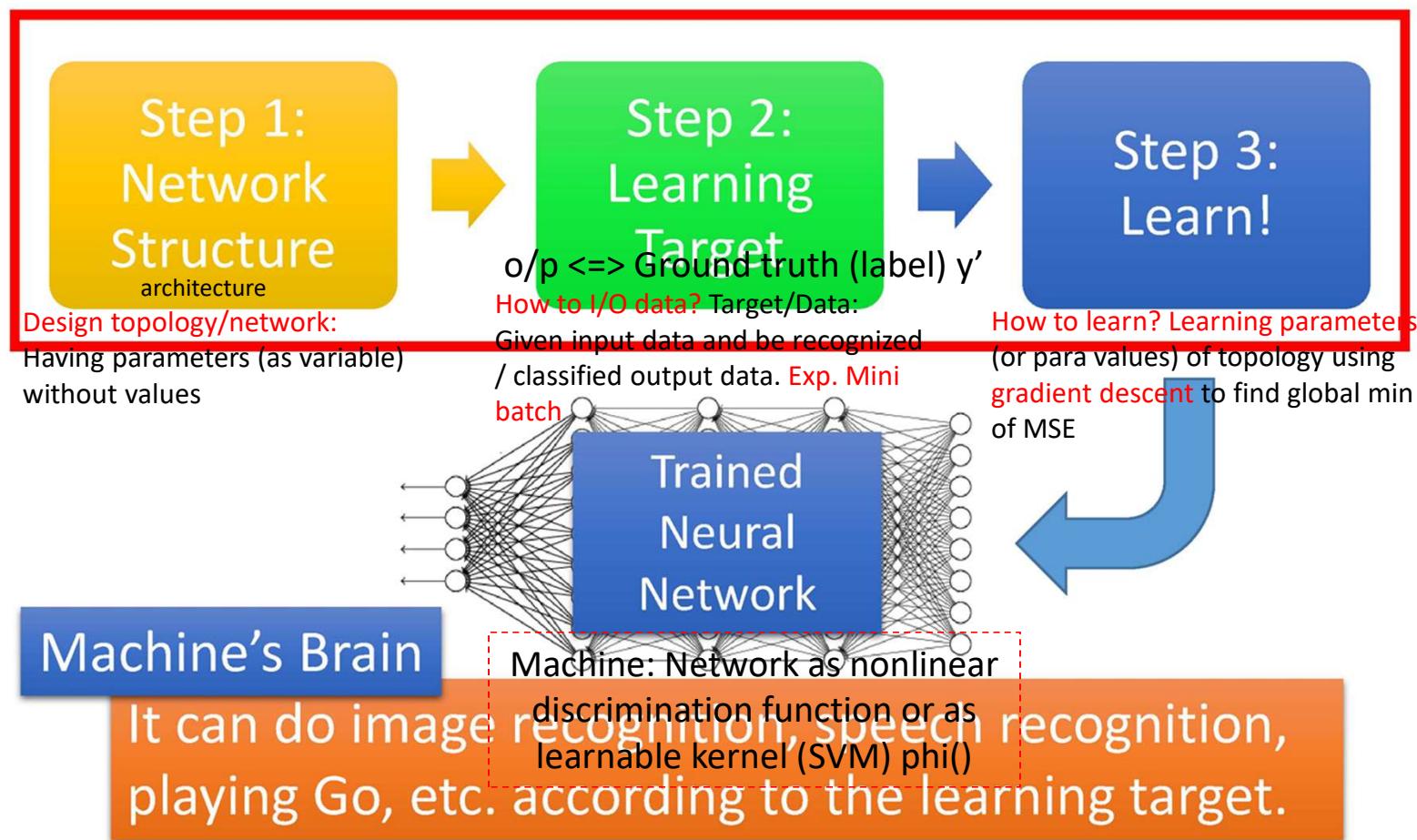
- 1) Fat+Short Vs. Thin+Tall
- 2) Multi-Task:
 - (1) How to reduce para. in order to use less data
 - (2) Modularization (logic circuit) – Combine network / Circuit
- 3) End to End Learning

I.3

“Hello world” for deep learning

Coding
Example

Three Steps for Deep Learning



Three Steps for Deep Learning



天生的腦



Expected Target: 100



http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Three Steps for Deep Learning

Design topology/network:

Having parameters (as variable)
without values

**Step 1:
Network
Structure**

Topology: Refrigerator

How to I/O data? Target/Data:

Given input data and be recognized / classified output data. **Exp. Mini batch**

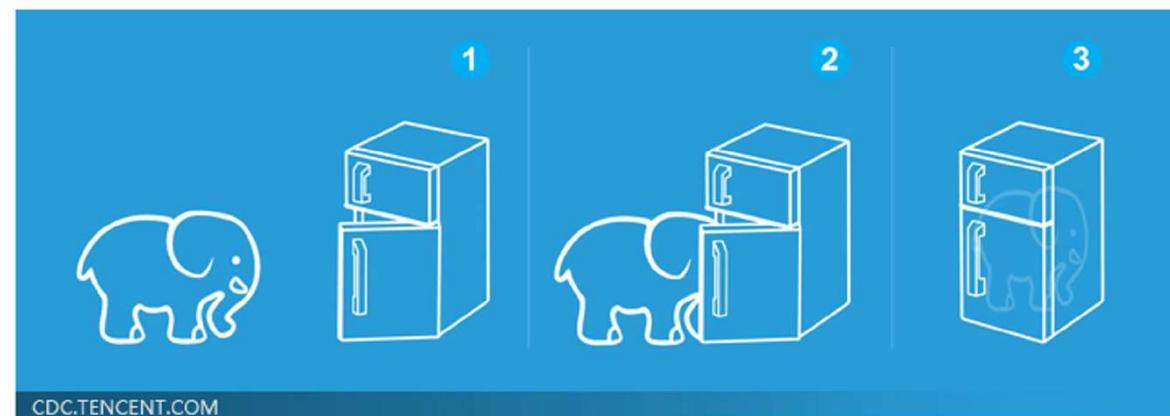
**Step 2:
Learning
Target**

$o/p \Leftrightarrow$ Ground truth (label) y'
Target/Data: Given
elephant to recognize it.

How to learn? Learning parameters (or para values) of topology using **gradient descent** to find global min of MSE

**Step 3:
Learn!**

Deep Learning is so simple

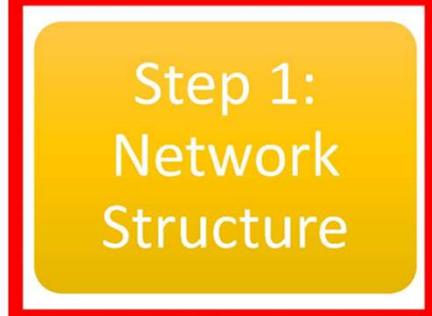


I.1. Step1

Three Steps for Deep Learning

Design topology/network:

Having parameters (as variable)
without values



How to I/O data? Target/Data:

Given input data and be recognized
/ classified output data. Exp. Mini
batch



How to learn? Learning parameters

(or para values) of topology using
gradient descent to find global min
of MSE

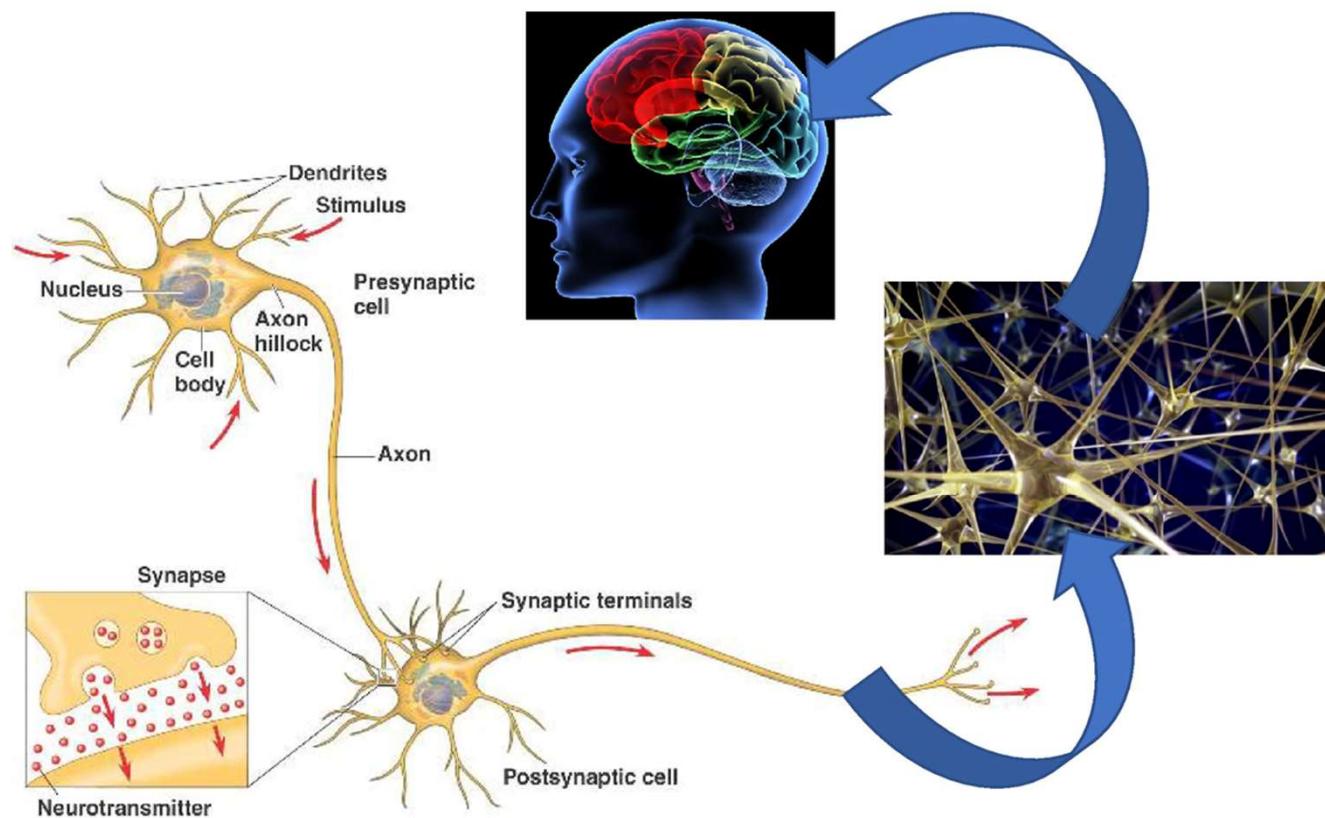


天生的腦



http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Human Brains



EX. 人臉 detection

input :

(20X20個pixel的圖)

$x_1, x_2, x_3, \dots, x_{400}$

output :

(人臉=1, 非人臉=0)

$y_1 = 1$ or $y_1 = 0$

Why not as SVM

$y_1 = 1$ or $y_1 = -1$

-One weak classifier is one haar filter
is also as one neuron unit

-One strong classifier is as one layer

類神經網路應用與實作

當神經細胞透過突觸與樹突從其他神經元輸入脈波訊號後，經過體細胞處理，產生一個新的脈波訊號，如果脈波訊號夠強，將產生一個約 100 毫伏 0.001 秒的脈波訊號，這個訊號再經過軸索傳送到其樹突，再透過突觸與樹突成為其他神經元的輸入脈波訊號。如果脈波訊號是經過興奮突觸(excitatory synapse)，則會增加脈波訊號的速率(pulse rate)，如果脈波訊號是經過抑制突觸(inhibitory synapse)，則會減少脈波訊號的速率。因此，脈波訊號的速率是同時取決於輸入脈波訊號的速率，以及突觸的強度(strength)。而突觸的強度可視為神經網路儲存資訊之所在，神經網路的學習即在調整突觸的強度。*權重 node*

類神經網路是由許多的人工神經細胞(artificial neuron)所組成，人工神經細胞又稱類神經元、人工神經元、處理單元(processing element)(圖 1-2)。介於處理單元之間的訊號傳遞路徑稱為連結(connection)。每一個連結上有一個數值的加權值(W_{ij})，用以表示第 i 處理單元對第 j 個處理單元之影響強度。每一個處理單元的輸出以扇狀送出，成為其他許多處理單元的輸入。處理單元其輸出值與輸入值的關係式，一般可用輸入值的加權乘積和之函數來表示：

1. Learning $y_j = f(\sum_i W_{ij}x_i - \theta_j)$
2. Test 其中 y_j = 模仿軸索功能的輸出訊號。
 f = 模仿體細胞處理功能的轉換函數(transfer function)，是一個可將從其他處理單元輸入的輸入值之加權乘積和，轉換成處理單元輸出值的數學公式。
 x_i = 模仿樹突功能的輸入訊號。
 W_{ij} = 模仿突觸強度功能的連結加權值。
 θ_j = 模仿體細胞閾值功能的門限值。

圖 1-2 人工神經元模型

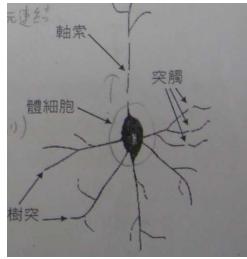
Y_j 輸出變數
j 處理單元
 $W_{i1}, W_{i2}, W_{i3}, W_{i4}, W_{i5}, W_{i6}, W_{i7}, W_{i8}, W_{i9}, W_{i10}$
 $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$
 x_i 輸入變數
 θ_j

圖 1-2 人工神經元模型

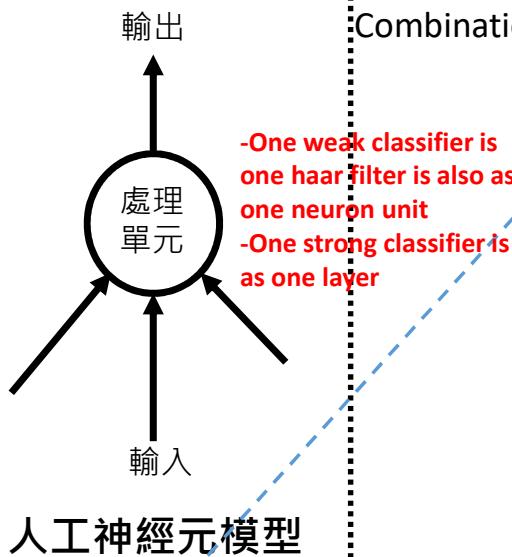
1-1

1-4

Review NN (Ch1) :



生物神經元模型



$$y_i = f(\sum_i w_i x_i - \theta)$$

bias
constant

A neuron unit

Neural Network

One Neuron

Linear Combination:

$$z = a_1 w_1 + \dots + a_k w_k + \dots + a_K w_K + b$$

Constant, bias

$$\begin{matrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_K \end{matrix}$$

weights

Linear

Combination:

J input data vectors
(here J=1)

$$Aw = z$$

?

$$\begin{bmatrix} w_{11} w_{21} \dots w_{k1} \dots w_{N1} b_1 \\ w_{12} w_{22} \dots w_{k2} \dots w_{N2} b_2 \\ \vdots \\ w_{1j} w_{2j} \dots w_{kj} \dots w_{Nj} b_j \end{bmatrix}_{J \times (N+1)}$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ \vdots \\ a_N \\ 1 \end{bmatrix}_{(N+1) \times 1} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \\ \vdots \\ z_j \\ 1 \end{bmatrix}_{J \times 1}$$

1 J-dim output result vector, here J=1)

A simple function

Activation function

$$b$$

bias

$$\sigma(z)$$

a

$$a = \sigma(z)$$

Here, the output is a, which is one input element for next neuron

Non-Linear Discrimination by Kernel Function / Activation Function as SVM phi()

input : (20X20個pixel的圖) $x_1, x_2, x_3, \dots, x_{k=400}$
One neuron
Linear Discrimination by Linear Combination

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

Neural Network

One **Neuron**

- One weak classifier is one haar filter is also as one neuron unit
- One strong classifier is as one layer

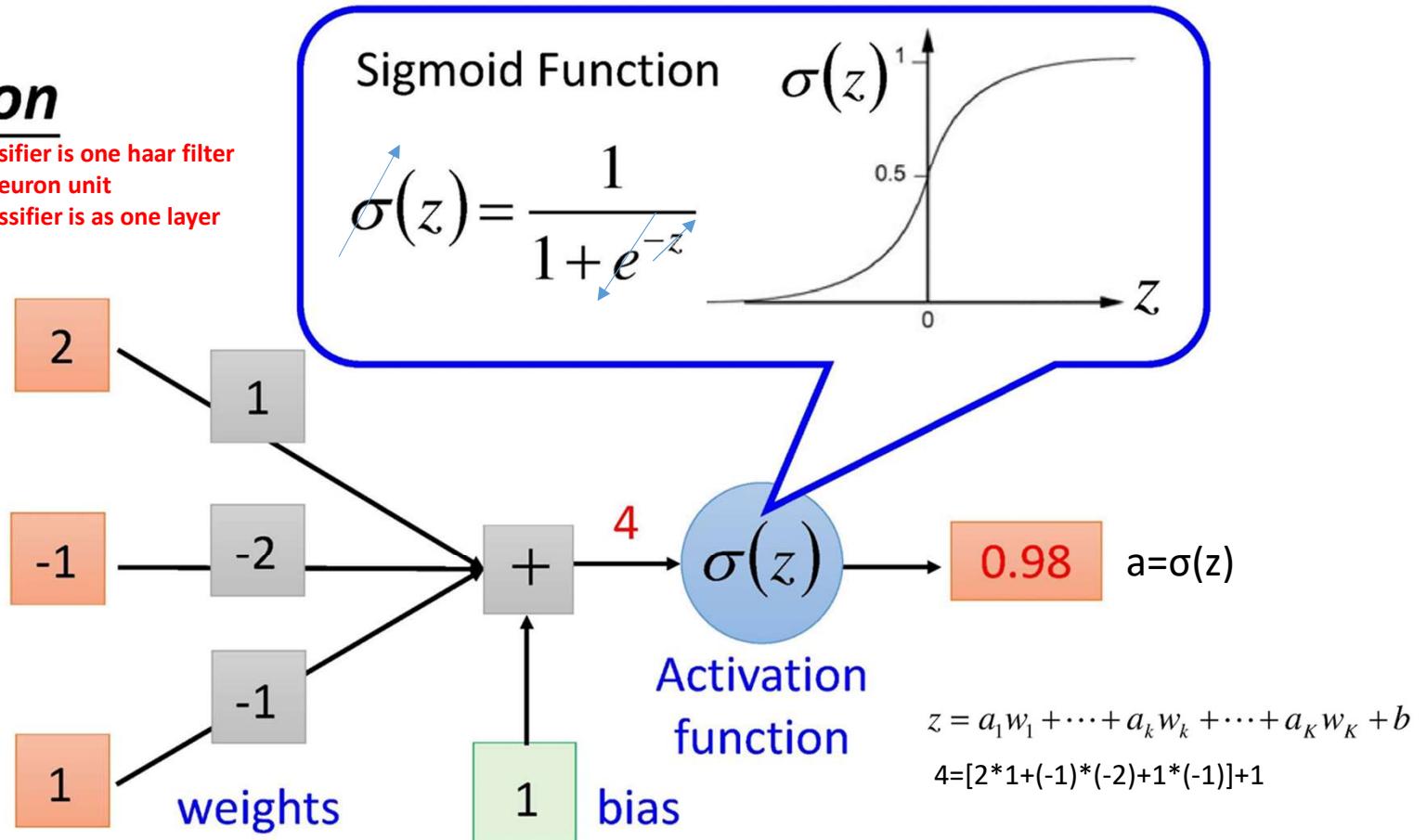
Input pixel of image vector is normalized btw -255 ~ +255 by:

- 1) Ave.(R,G,B) over ImageNet
- 2) Unbiased each each pixel(x,y) by

`Pixel_R(x,y) := Ave_R(x,y)`

`Pixel_G(x,y) := Ave_G(x,y)`

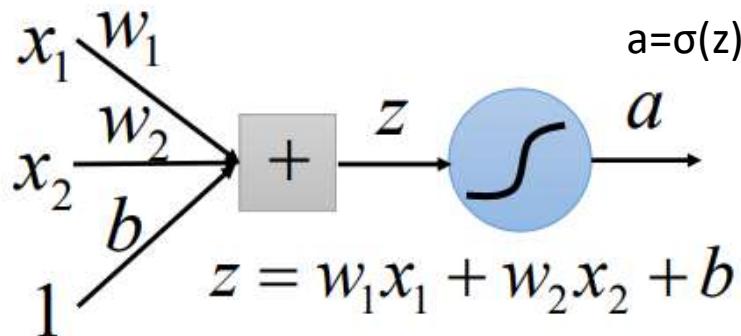
`Pixel_B(x,y) := Ave_B(x,y)`



Reference:

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20\(v4\).pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20(v4).pdf)

Limitation of Single Layer



$$\begin{cases} \text{yes} & a \geq \text{threshold} \\ \text{no} & a < \text{threshold} \end{cases}$$

XOR問題

Input		Output
x_1	x_2	
0	0	No (0)
0	1	Yes (1)
1	0	Yes (1)
1	1	No (0)

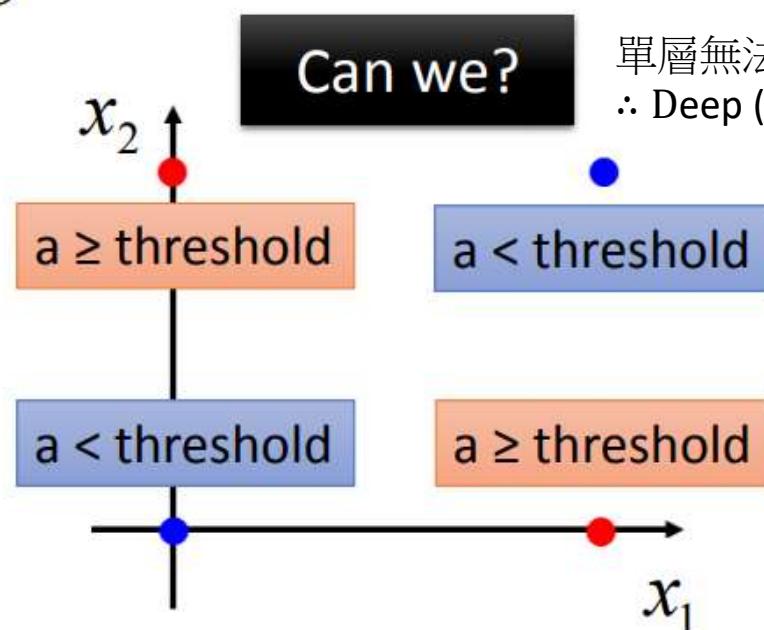
Output: 2 classes 0 or 1,

problem: You cannot discrimination to 2 classes

J: You can solve it by using MLP non-linear discrimination. But here it doesn't connect to MLP yet.

Can we?

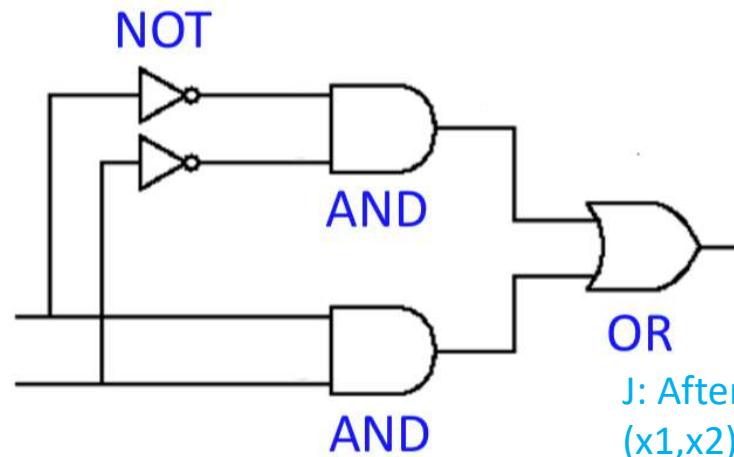
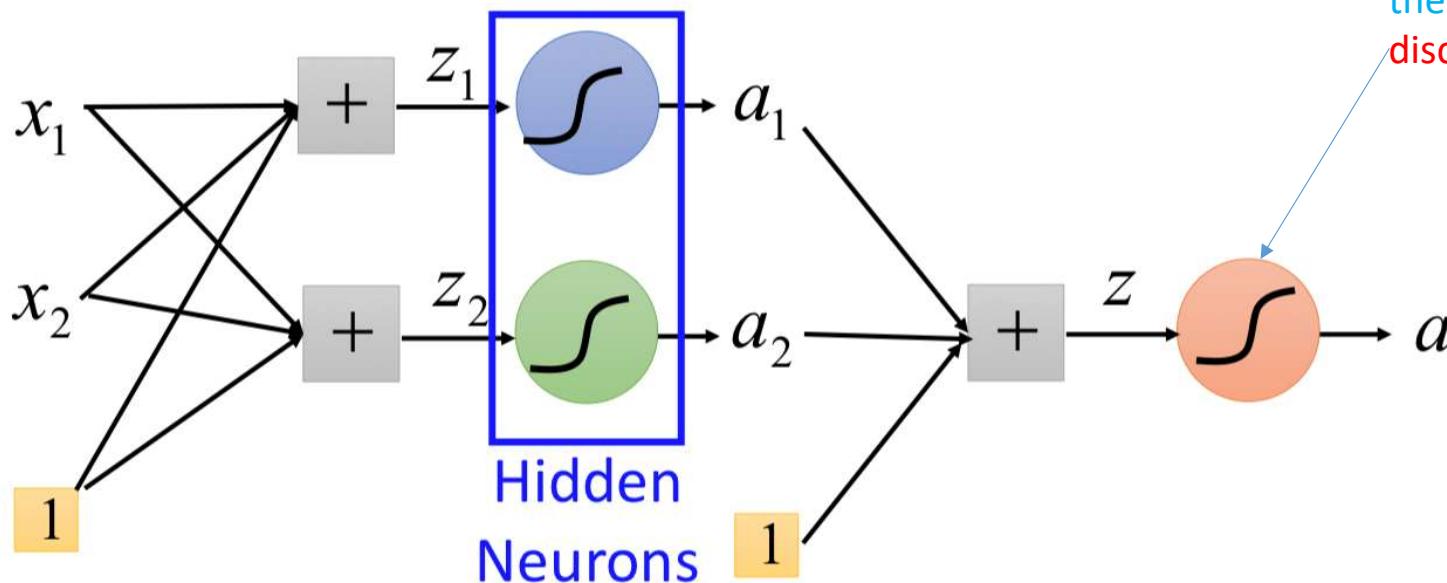
單層無法解決
∴ Deep (多層)



??

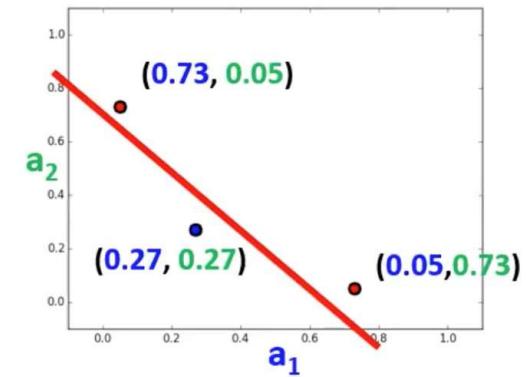
Neural Network 1/2

Neural Network

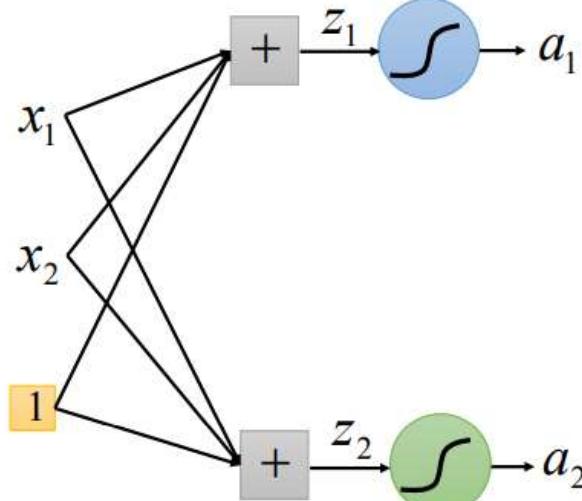


XOR問題
單層無法解決
 \therefore Deep (多層)

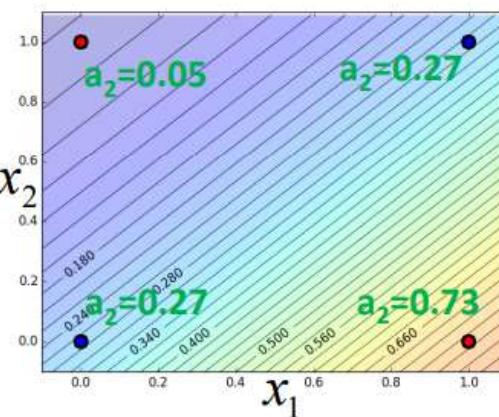
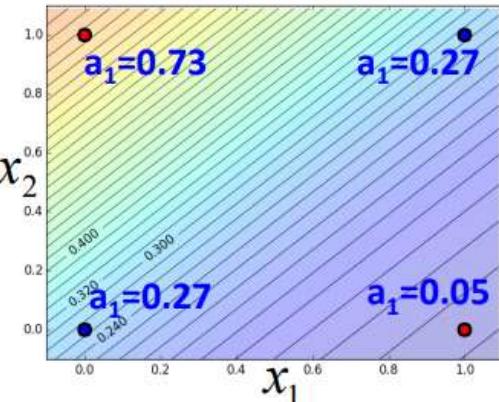
J: After we transform from (x_1, x_2) space to (a_1, a_2) space, then we can separate by linear discrimination



Limitation of Single Layer 2/2

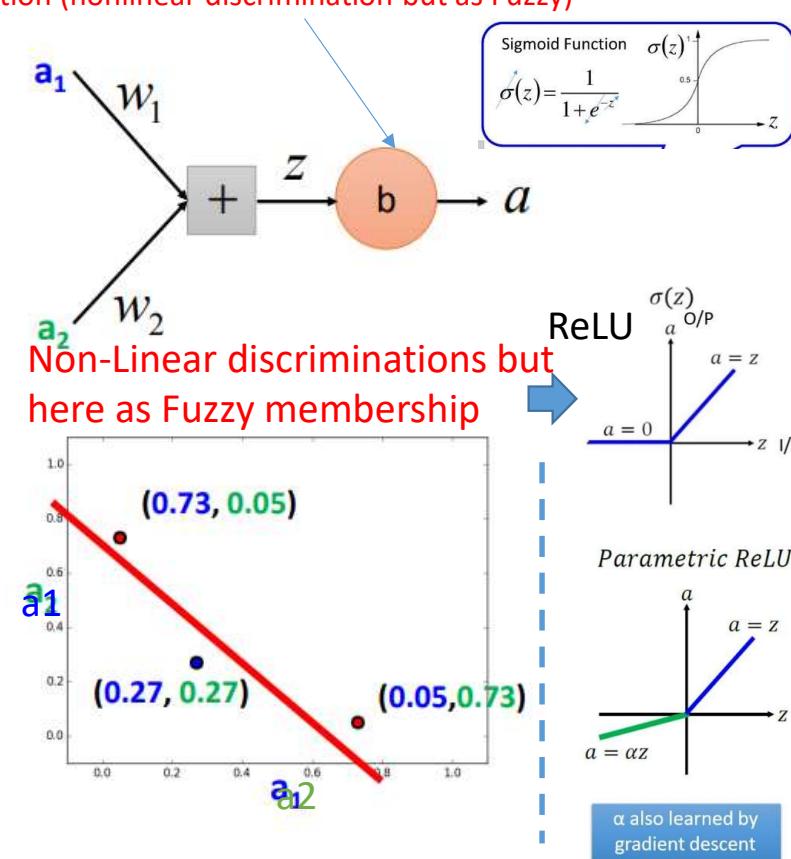


Space domain transform



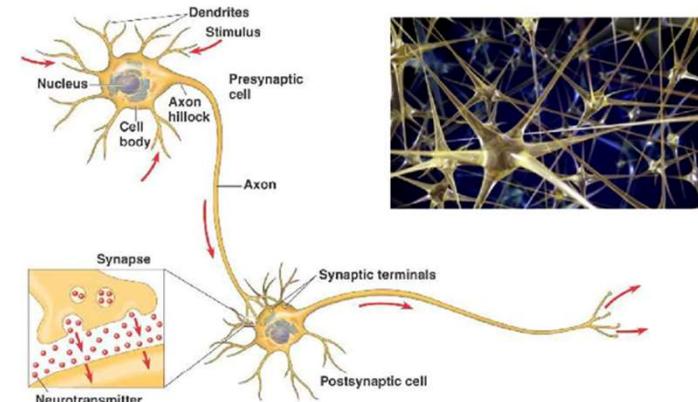
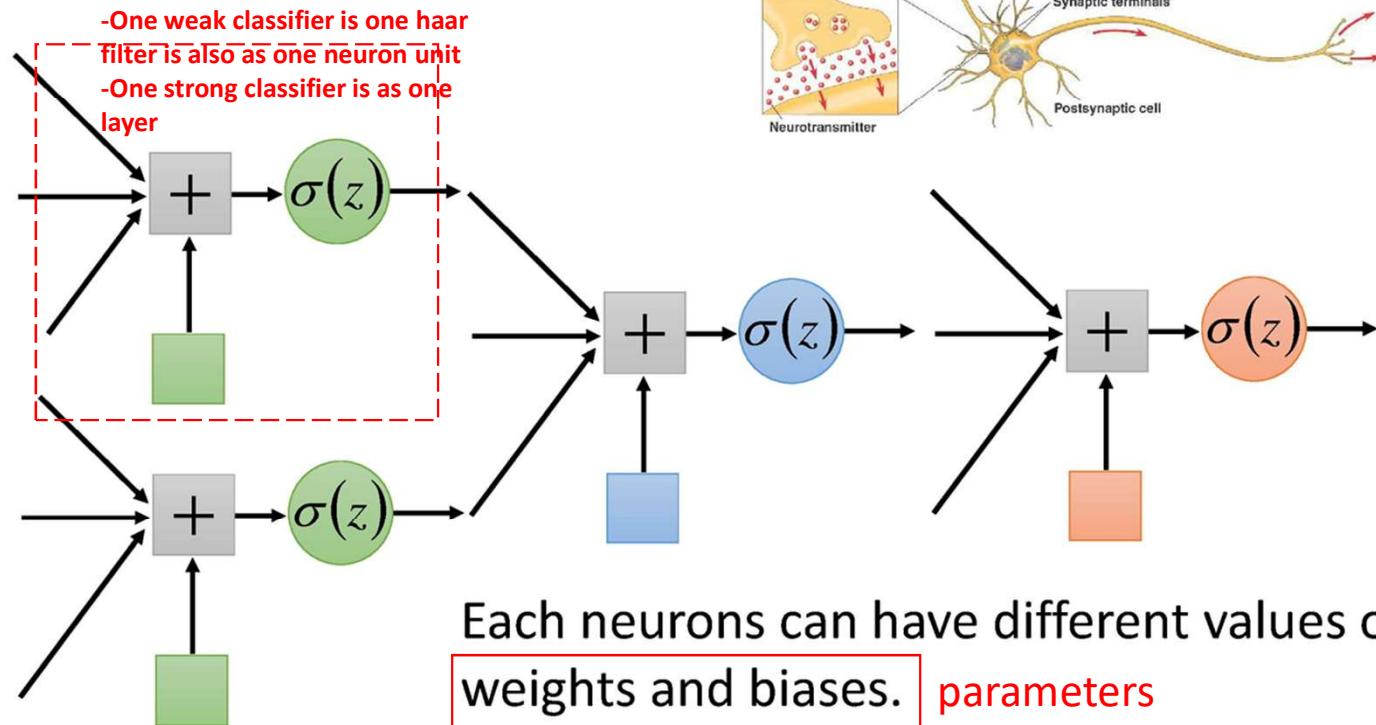
J:

- 1) After we transform from (x_1, x_2) space to (a_1, a_2) space,
- 2) then we can separate by linear discrimination
- 3.1) That is, instead of using SVM directly backproject to very high dimension by using kernel function or high dimensional activation function,
- 3.2) we can apply more layers (deeper network) to reach the same goal – space transformation layer by layer (step by step) and not so high dimension transform/activation function (nonlinear discrimination but as Fuzzy)



Neural Network

: AdaBoost - Cascade



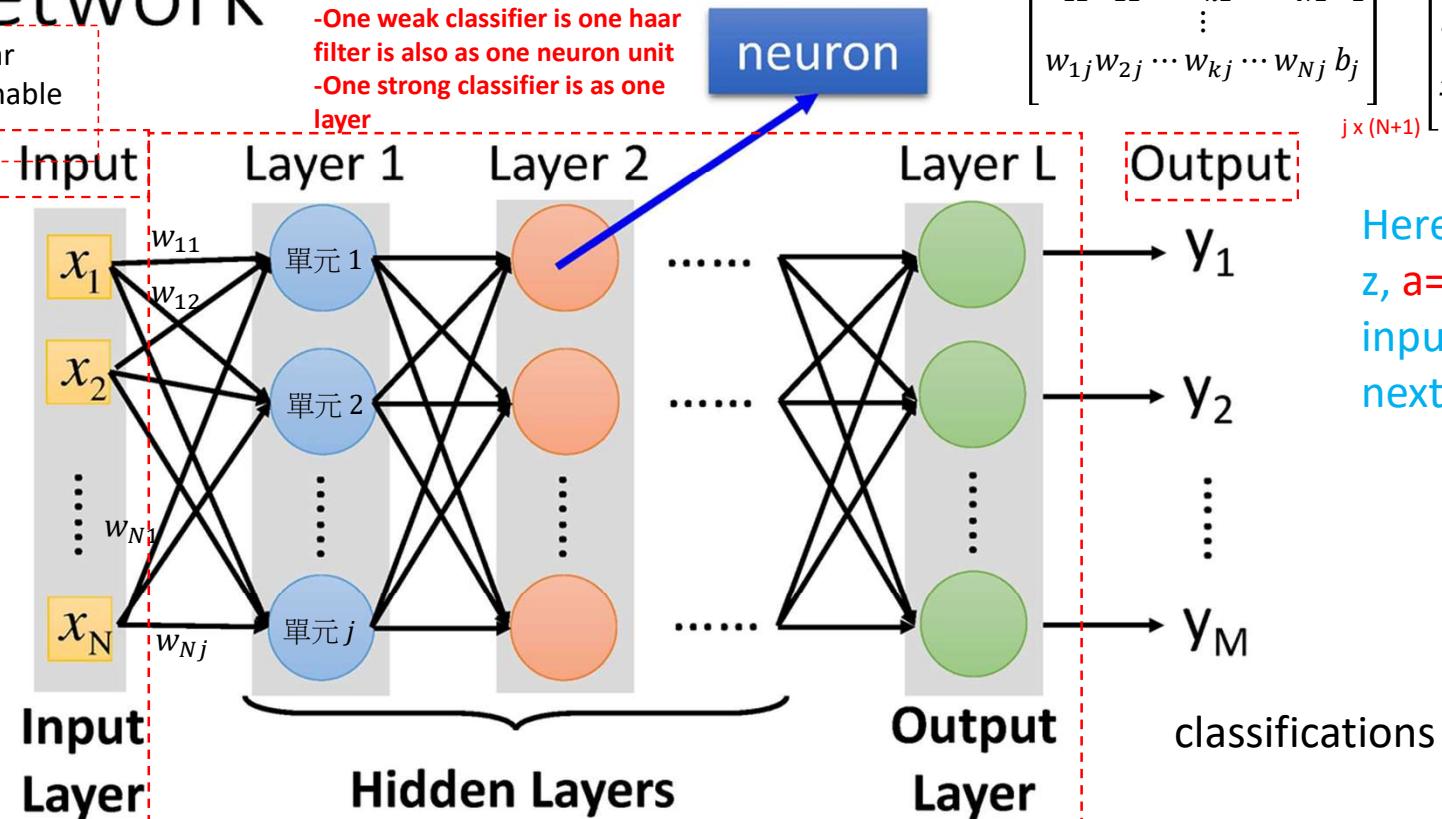
Each neurons can have different values of weights and biases. parameters

The values of weights and biases will be obtained by learning (step 3).

Ax = b 型式 :

Fully Connect Feedforward Network : AdaBoost - Cascade

Machine: Network as nonlinear discrimination function or as learnable kernel (SVM) phi()



AdaBoost
Cascade:

Deep means many hidden layers

Layer 1 Parameters
(known?)

$$\begin{bmatrix} w_{11}w_{21} \cdots w_{k1} \cdots w_{N1} b_1 \\ w_{12}w_{22} \cdots w_{k2} \cdots w_{N2} b_2 \\ \vdots \\ w_{1j}w_{2j} \cdots w_{kj} \cdots w_{Nj} b_j \end{bmatrix}$$

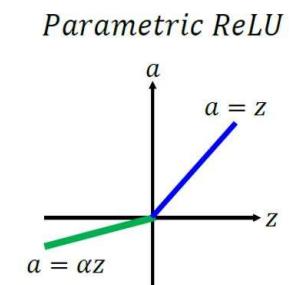
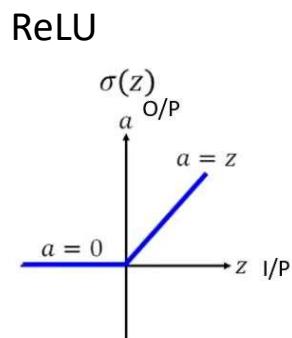
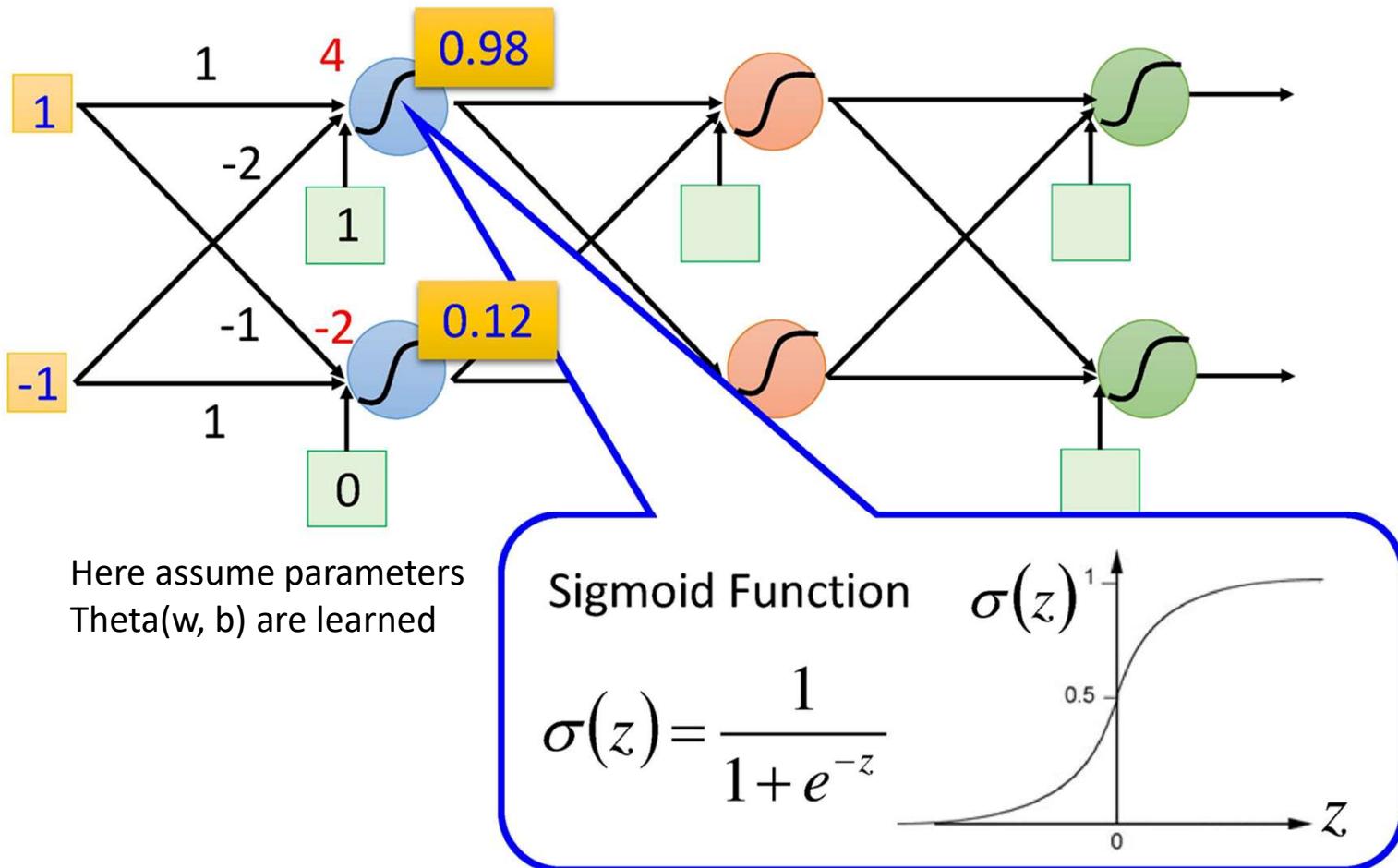
I/P O/P

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_N \\ 1 \end{bmatrix}_{j \times (N+1)} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \\ \vdots \\ z_j \end{bmatrix}_{(N+1) \times 1}^{j \times 1}$$

Here, the output is z , $a=\sigma(z)$ is one input element for next neuron

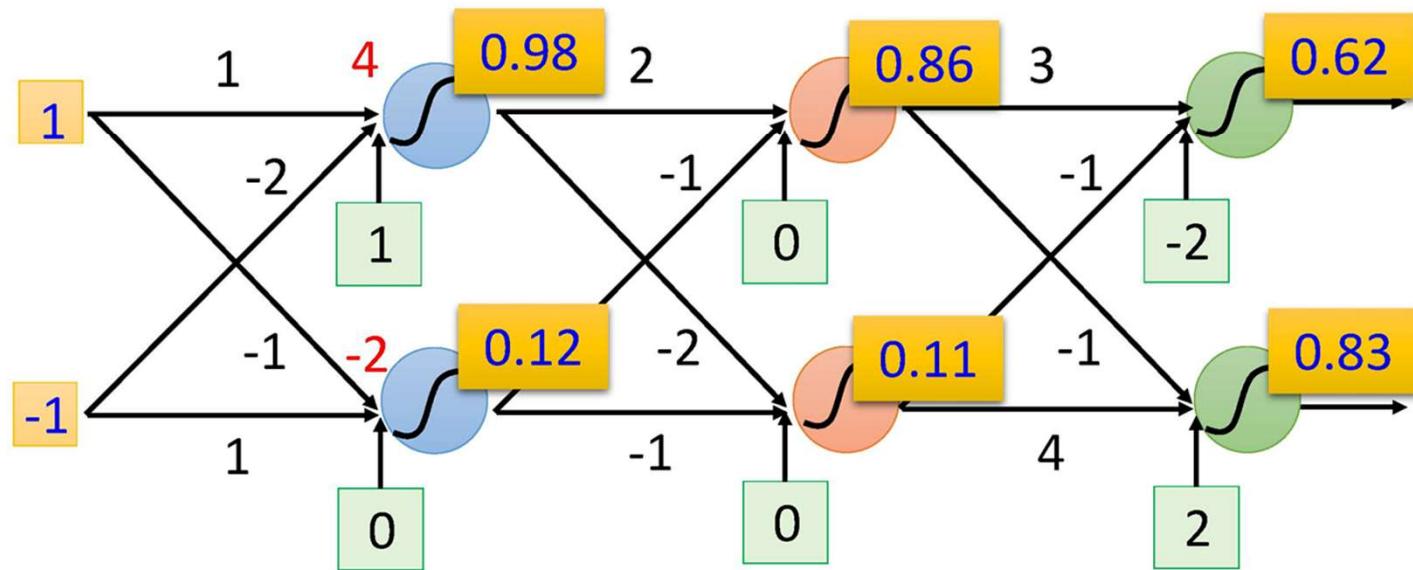
classifications

Fully Connect Feedforward Network : AdaBoost - Cascade



α also learned by gradient descent

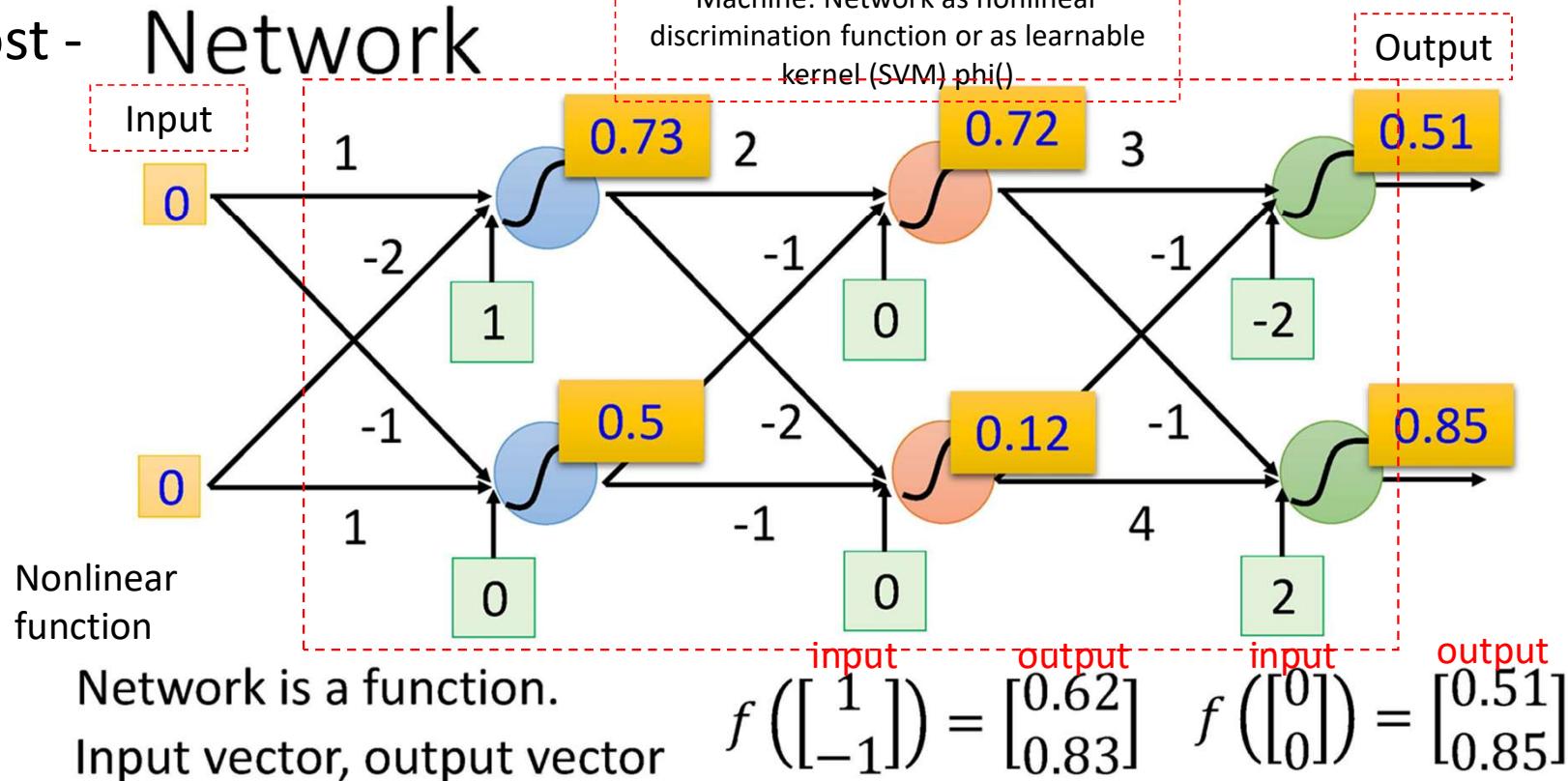
Fully Connect Feedforward Network : AdaBoost - Cascade



Here assume parameters
 $\Theta(w, b)$ are learned

: AdaBoost -
Cascade

Fully Connect Feedforward Network

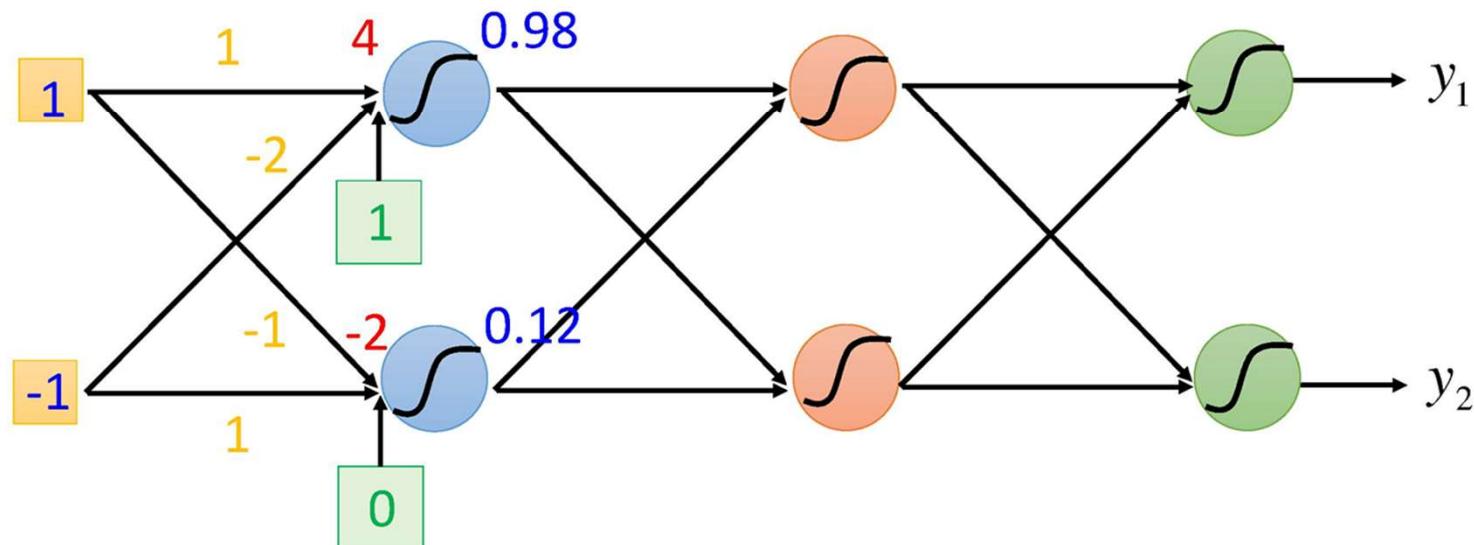


Here assume parameters
Theta(w, b) are learned

Weights and biases are network parameters θ

Different parameters θ define different functions

Fully Connect Feedforward Network - Matrix Operation



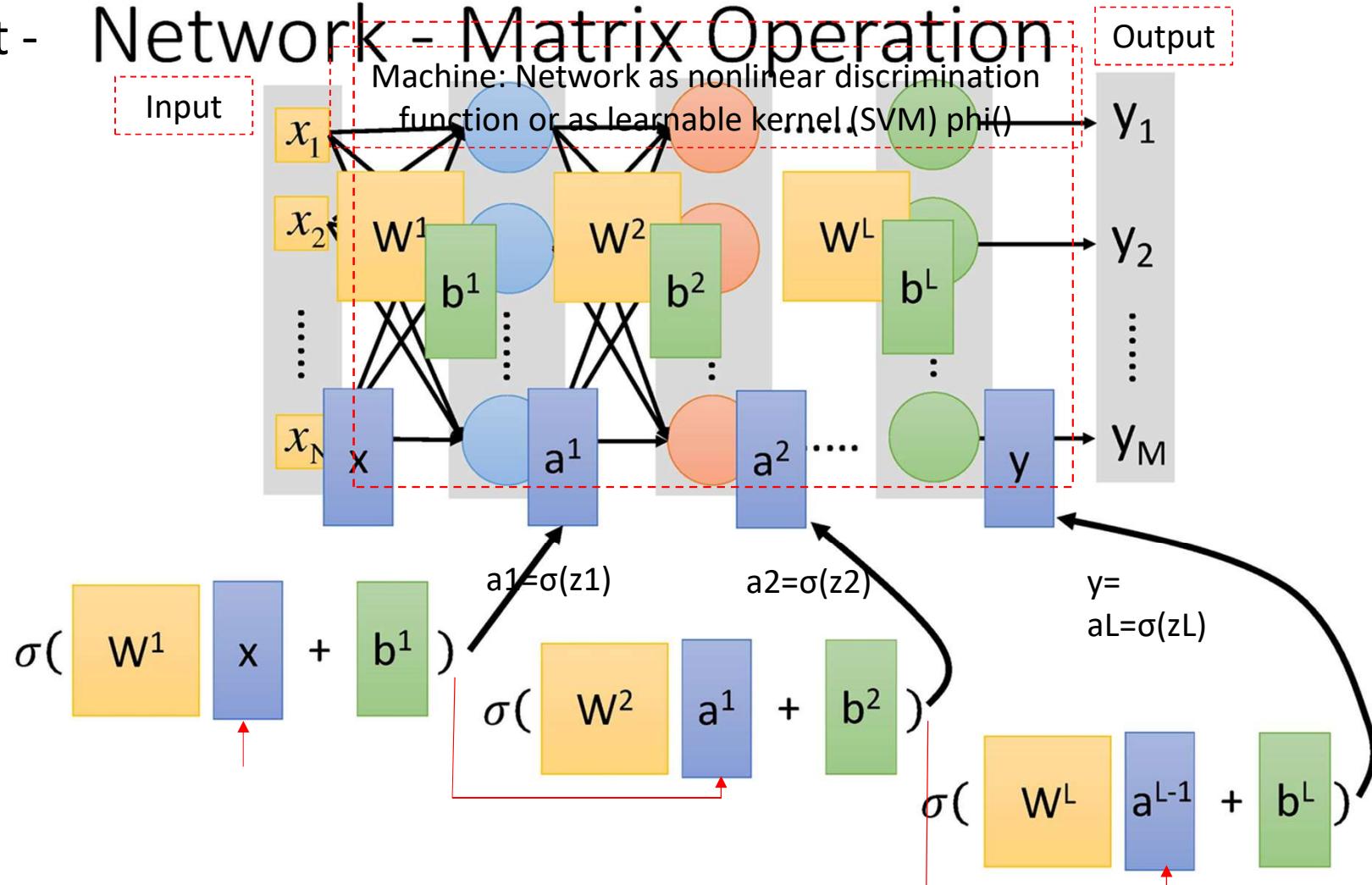
$$\sigma \left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}}_w \underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{\text{input}} + \underbrace{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}_b \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix} \quad \begin{array}{l} \text{output} \\ [a_1 = \sigma(z_1) \\ a_2 = \sigma(z_2)] \end{array}$$

Here assume parameters
Theta(w, b) are learned

$$\begin{bmatrix} 4 \\ -2 \end{bmatrix} \quad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Fully Connect Feedforward Network - Matrix Operation

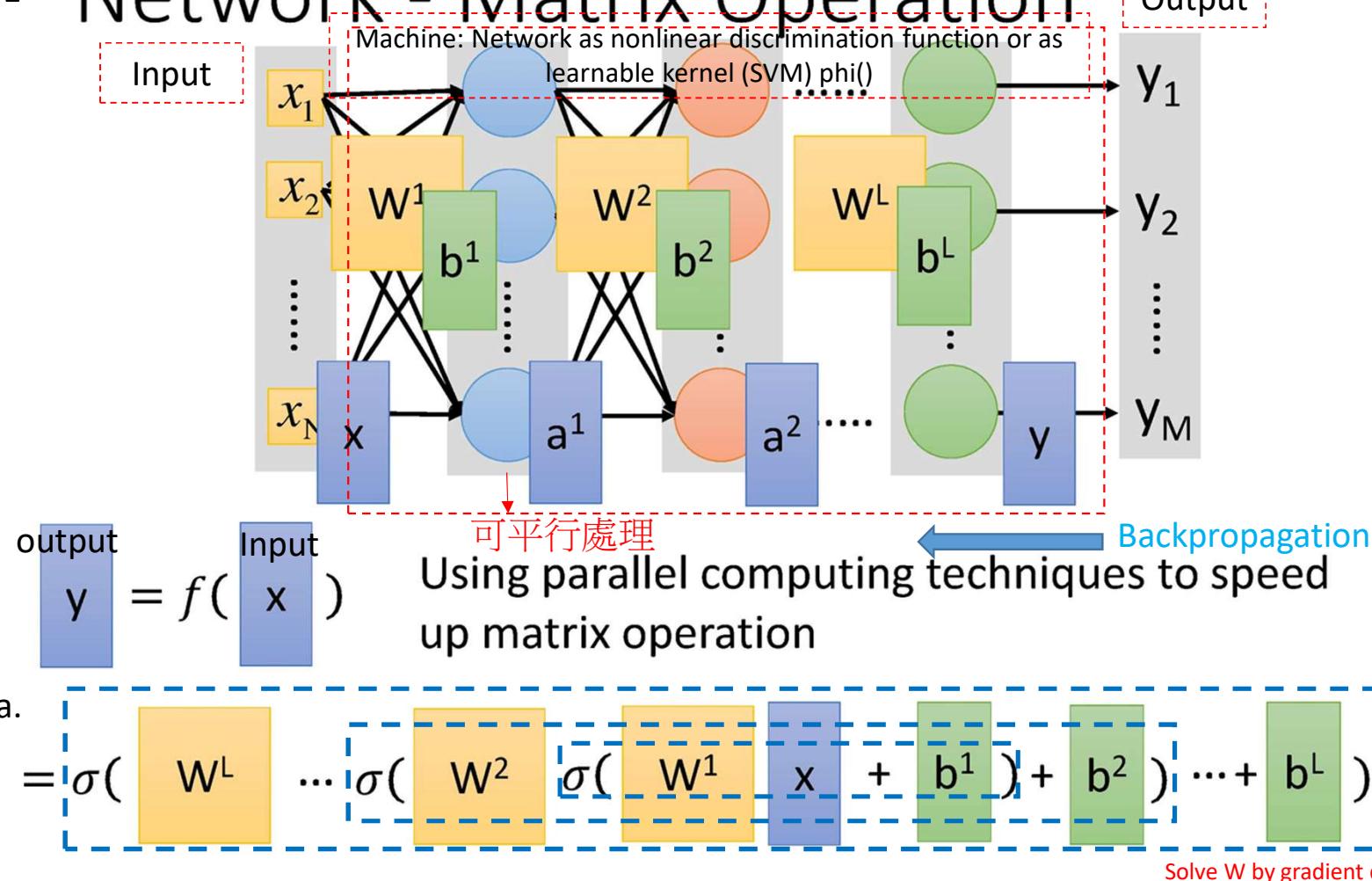
: AdaBoost - Cascade



: AdaBoost -
Cascade

Fully Connect Feedforward Network - Matrix Operation

$$\begin{bmatrix} w_{11}w_{21} \dots w_{k1} \dots w_{N1} b_1 \\ w_{12}w_{22} \dots w_{k2} \dots w_{N2} b_2 \\ \vdots \\ w_{1j}w_{2j} \dots w_{kj} \dots w_{Nj} b_j \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_N \\ 1 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \\ \vdots \\ z_N \\ z_j \end{bmatrix}$$



Output Layer: Softmax Layer

: AdaBoost -
Cascade

Output Layer (Option)

: Softmax Layer as normalization

- **Softmax layer** as the output layer

p.26 +4~ p.27+4

Ordinary Layer

$$z_1 \rightarrow \sigma \rightarrow y_1 = \sigma(z_1)$$

$$z_2 \rightarrow \sigma \rightarrow y_2 = \sigma(z_2)$$

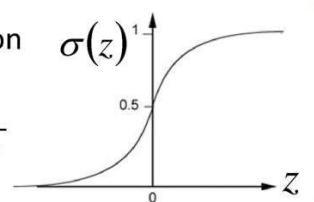
$$z_3 \rightarrow \sigma \rightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

Sigmoid Function

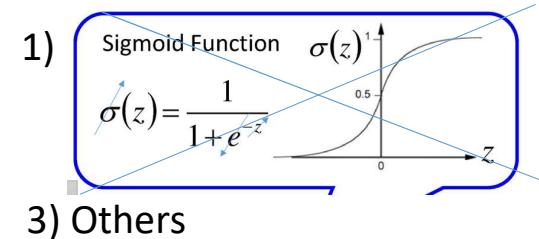
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Output Layer: Softmax Layer

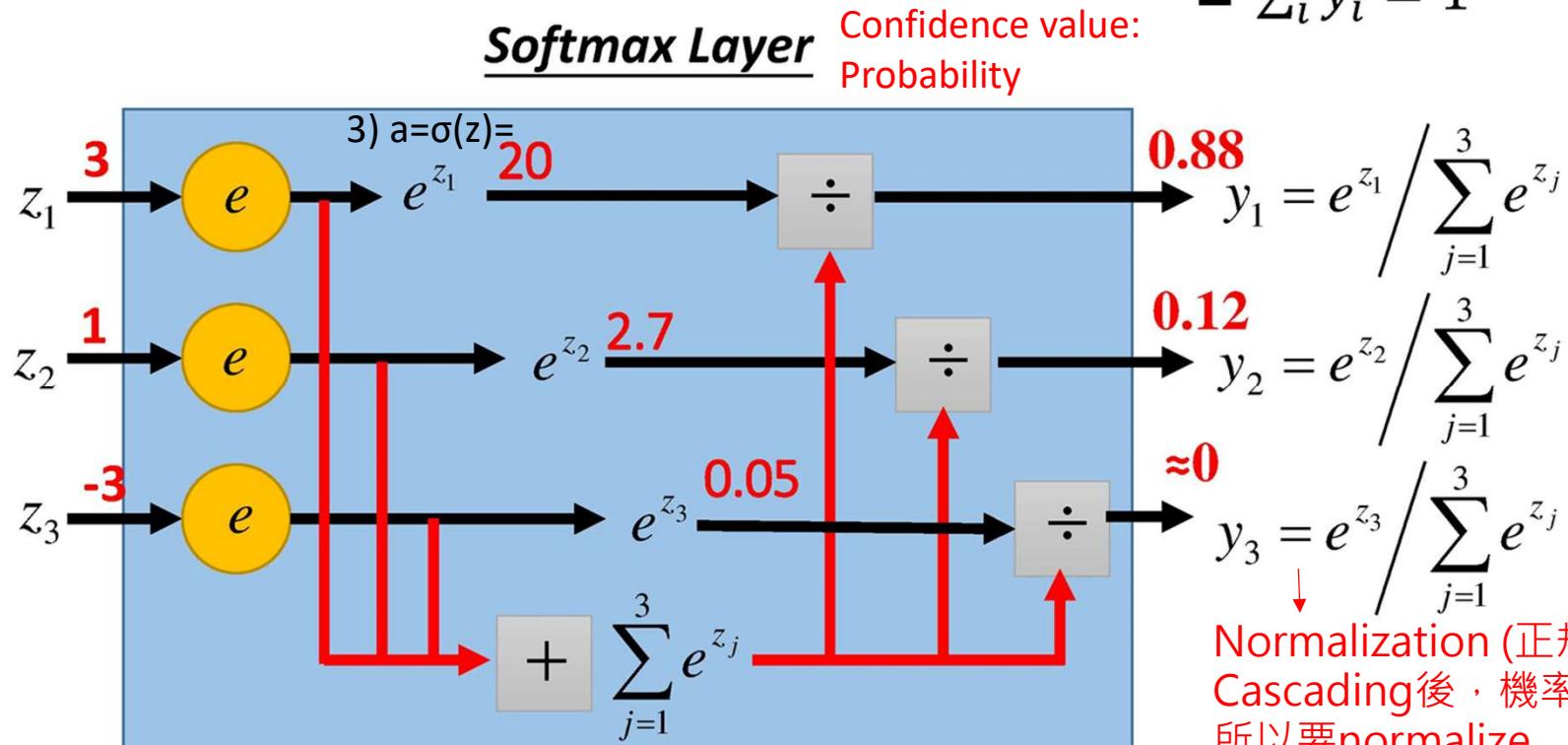
Output Layer (Option)

- : Softmax Layer as normalization
- Softmax layer as the output layer



Probability:

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

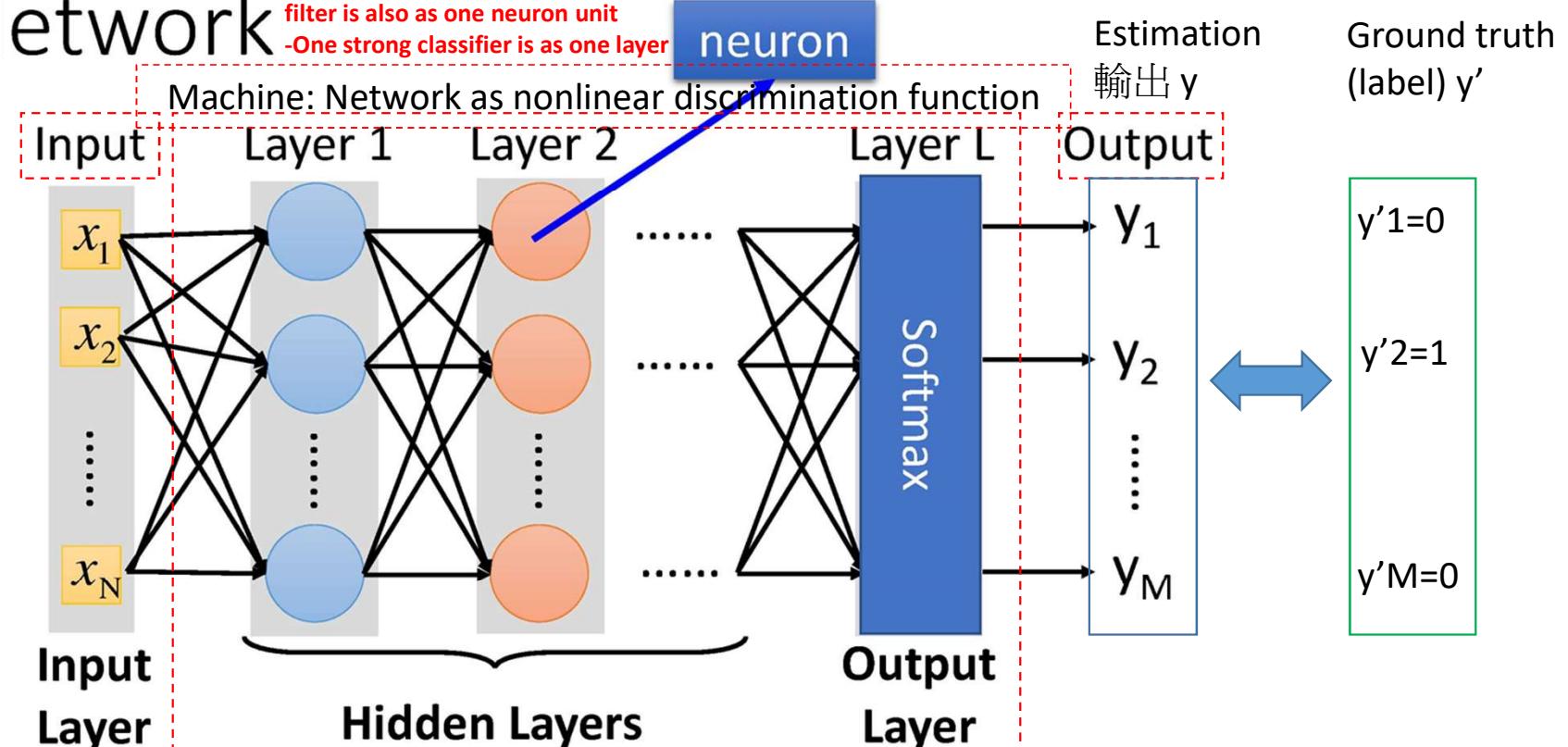


Output Layer: Softmax Layer Fully Connect Feedforward Network

: AdaBoost -
Cascade

-One weak classifier is one haar
filter is also as one neuron unit
-One strong classifier is as one layer

neuron



Q: How many layers? How many neurons for each layer?

Q: Can the structure be automatically determined?

Output Layer: Softmax Layer (1/2) ?? Confidence value: Probability

Softmax函數 : Softmax Layer as normalization +

維基百科，自由的百科全書 enhance or reduce

在數學，尤其是概率論和相關領域中，**Softmax**函數，或稱歸一化指數函數^{[1]:198}，是邏輯函數的一種推廣。它能將一個含任意實數的K維的向量 \mathbf{z} 的「壓縮」到另一個K維實向量 $\sigma(\mathbf{z})$ 中，使得每一個元素的範圍都在(0, 1)之間，並且所有元素的和為1。該函數的形式通常按下面的式子給出：

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

J: Why exponential? Bigger is getting much bigger, smaller is getting much smaller

Softmax函數實際上是有限項離散概率分布的梯度對數歸一化。因此，Softmax函數在包括 多項邏輯回歸^{[1]:206–209}，多項線性判別分析，樸素貝葉斯分類器和人工神經網絡等的多種基於概率的多分類問題方法中都有着廣泛應用。^[2] 特別地，在多項邏輯回歸和線性判別分析中，函數的輸入是從K個不同的線性函數得到的結果，而樣本向量 \mathbf{x} 屬於第 j 個分類的概率為：

Prob. of o/p y

classify to j?

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^K e^{\mathbf{x}^\top \mathbf{w}_k}}$$

I/P: \mathbf{x}
O/P: y

??

這可以被視作K個線性函數 $\mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w}_1, \dots, \mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w}_K$ softmax函數的複合 ($\mathbf{x}^\top \mathbf{w} \mathbf{x} \mathbf{w}$)。

Output Layer: Softmax Layer (2/2)

例子

輸入向量 $[1,2,3,4,1,2,3]$ 對應的Softmax函數的值為 $[0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$ 。輸出向量中擁有最大權重的項對應着輸入向量中的最大值「4」。這也顯示了這個函數通常的意義：對向量進行歸一化，凸顯其中最大的值並抑制遠低於最大值的其他分量。

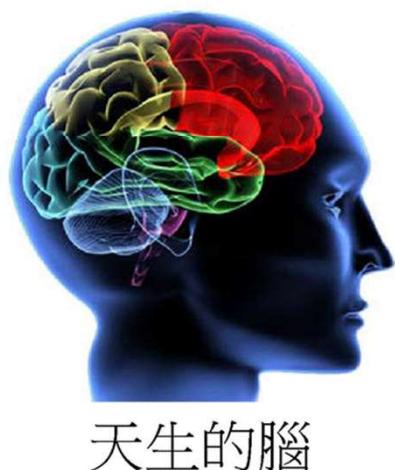
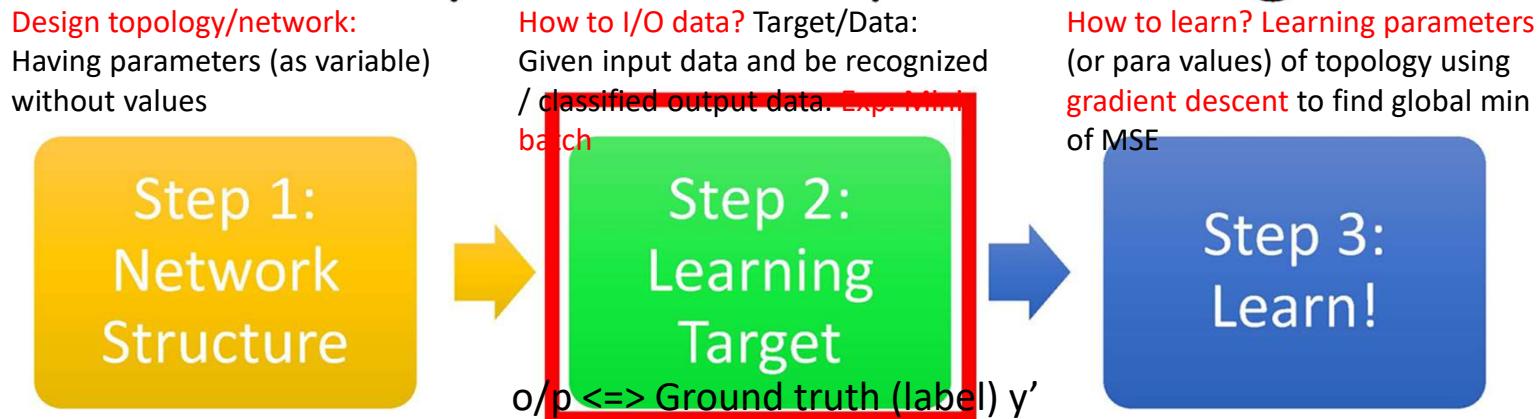
參考資料

1. Bishop, Christopher M. (2006).
2. ai-faq What is a softmax activation function? (<http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html>)

取自 "<https://zh.wikipedia.org/w/index.php?title=Softmax函数&oldid=43648521>"

Three Steps for Deep Learning

I.1.
Step2

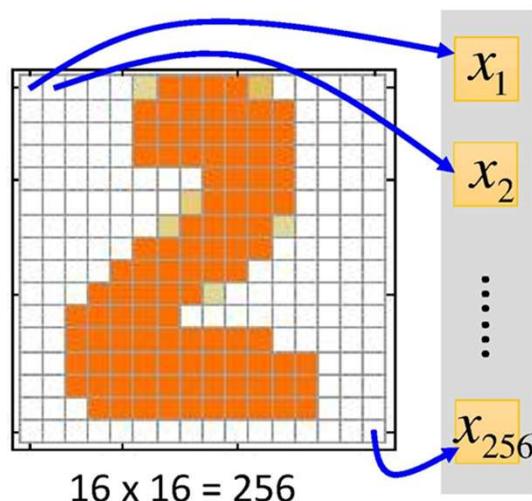


http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

?? So input is binary or gray value??

Example Application

Input



input 二值化
(也可能有非二值化的輸入 like Grayvalue input)

Ink \rightarrow 1

No ink \rightarrow 0

?? So input is binary or gray value??

$x: 0 \sim 255$

Output

Estimation
輸出 y

0.1

0.7

0.2

Ground truth
(label) y'

is 1
 $y'=0$

is 2
 $y'=1$

is 0
 $y'=0$

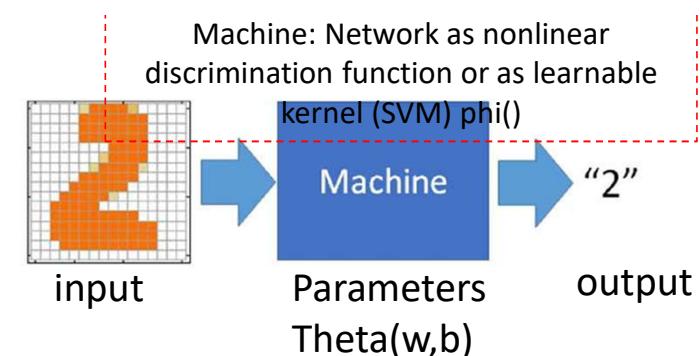
J: Confidence is not the similarity measure.

decision
The image is "2"

confidence 信心值(專有名詞) :
通常有lower bound (threshold)
(ex: 放一張貓的圖做為 input
顯示此張 input 圖為貓的confidence 為 0.6
該相信他為貓? Is it false positive??)

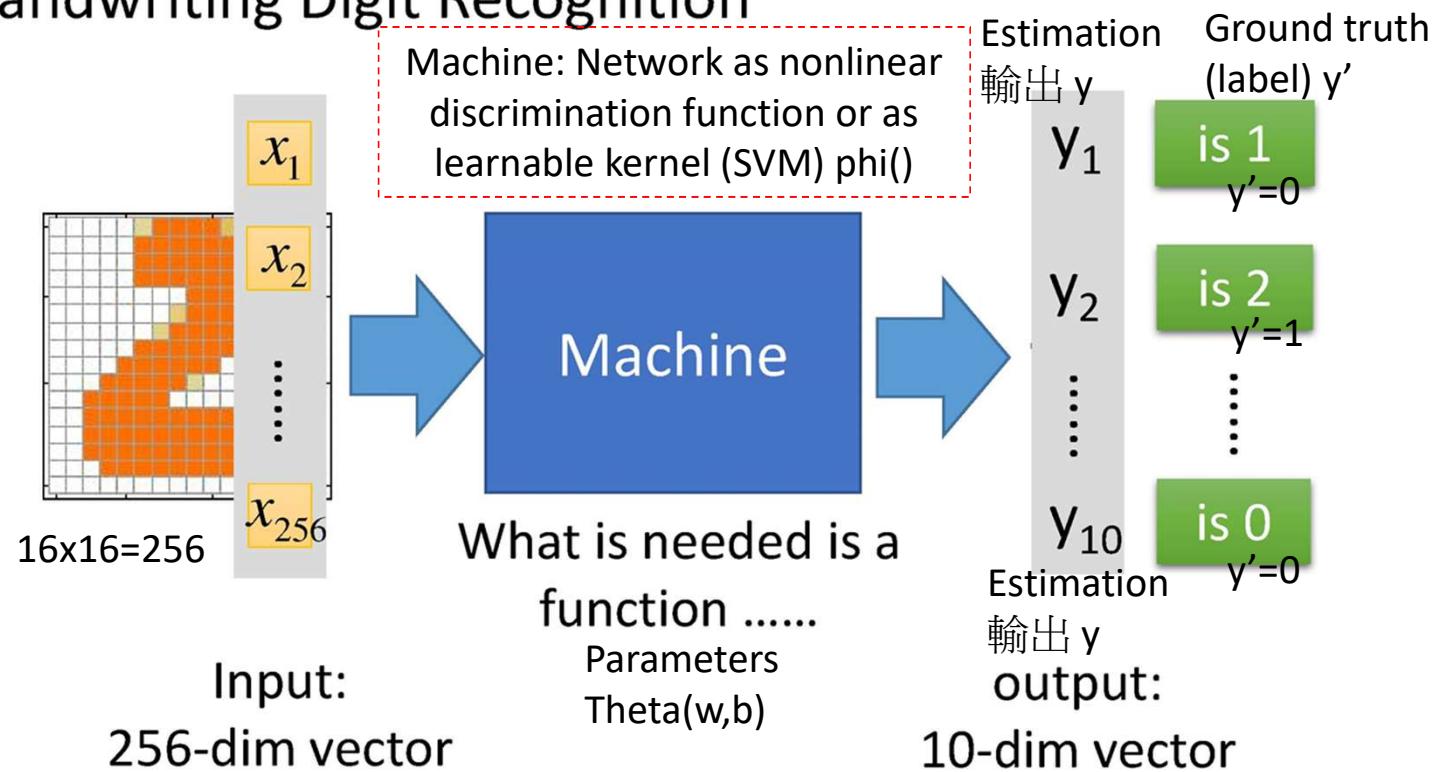
Each dimension represents the confidence of a digit.

= Similarity measure value ??



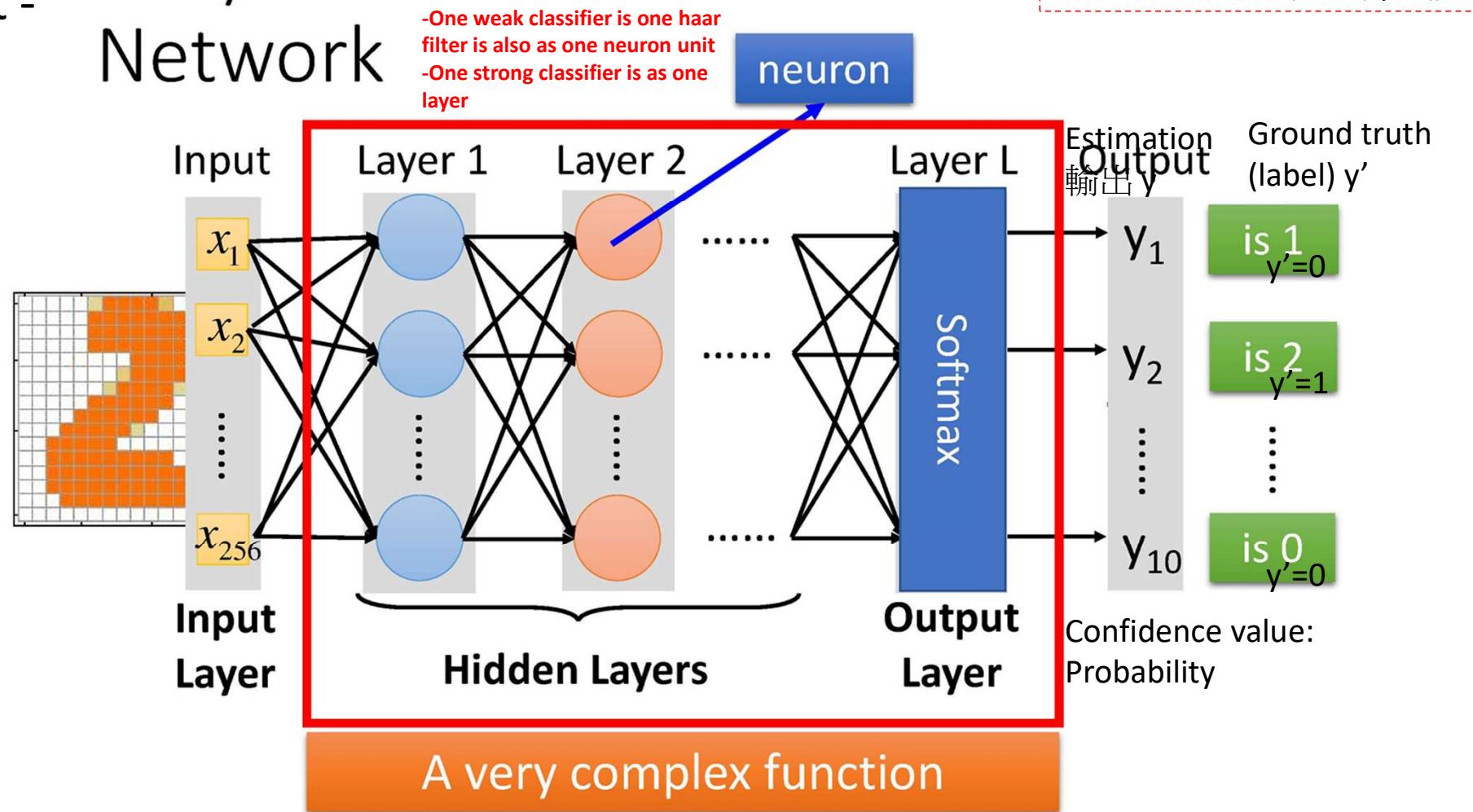
Example Application

- Handwriting Digit Recognition



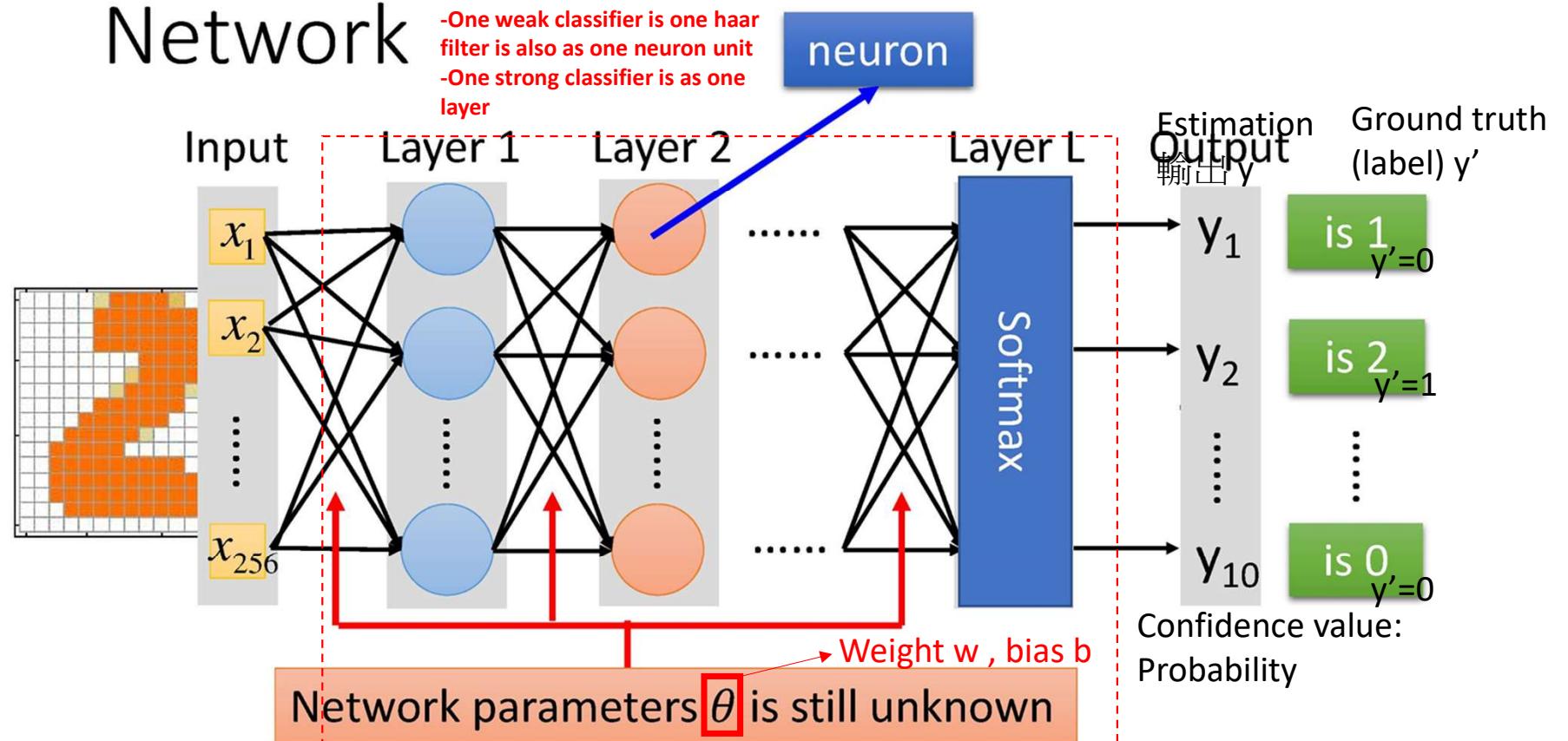
: AdaBoost -
Cascade

Fully Connect Feedforward Network



: AdaBoost -
Cascade

Fully Connect Feedforward Network



-Given I/P x ...
and o/p y ,
label y'
-Parameters
are learned

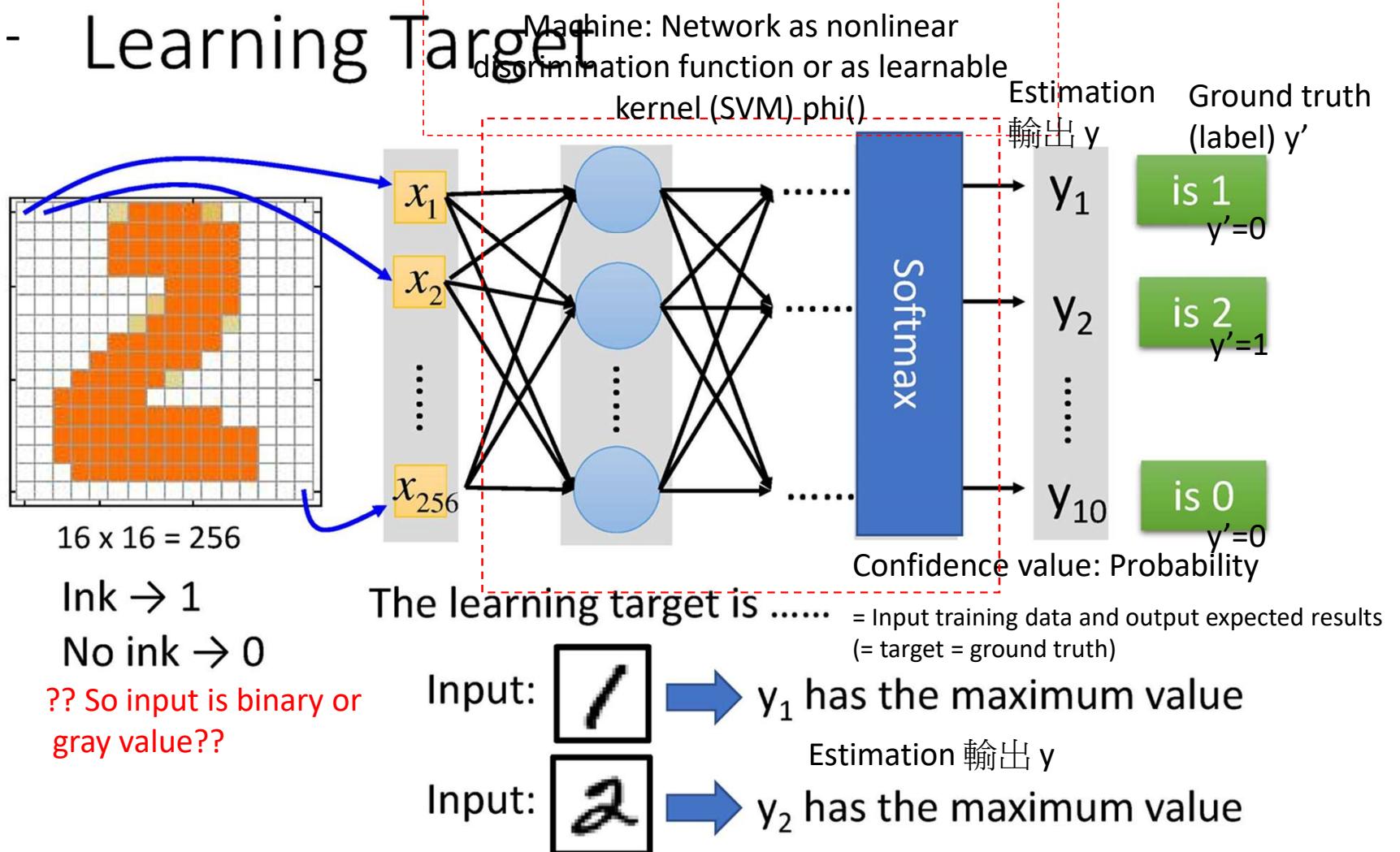
Tell the machine its learning target Step 2
= Ground truth (label) y'

Then machine find the parameters θ^* that achieve the target. Step 3

= Input training data and output expected results
(= target=ground truth)

: AdaBoost -
Cascade

Learning Target

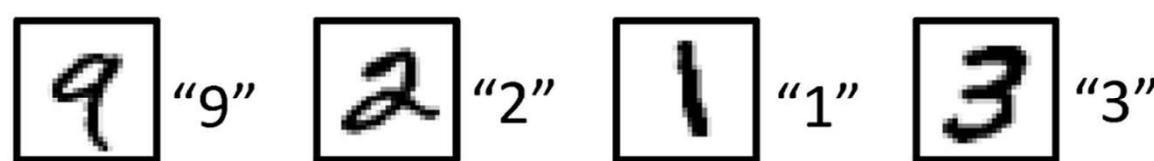
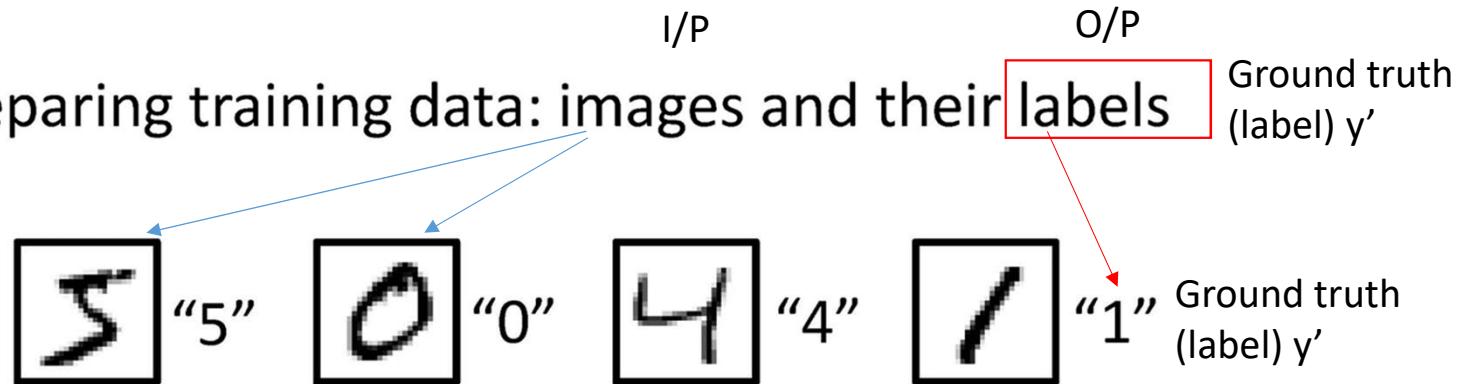


Training Data

Interpolation??
Extrapolation??

Estimation
輸出 y

- Preparing training data: images and their **labels**



Label

The learning target is **defined** on
the training data.

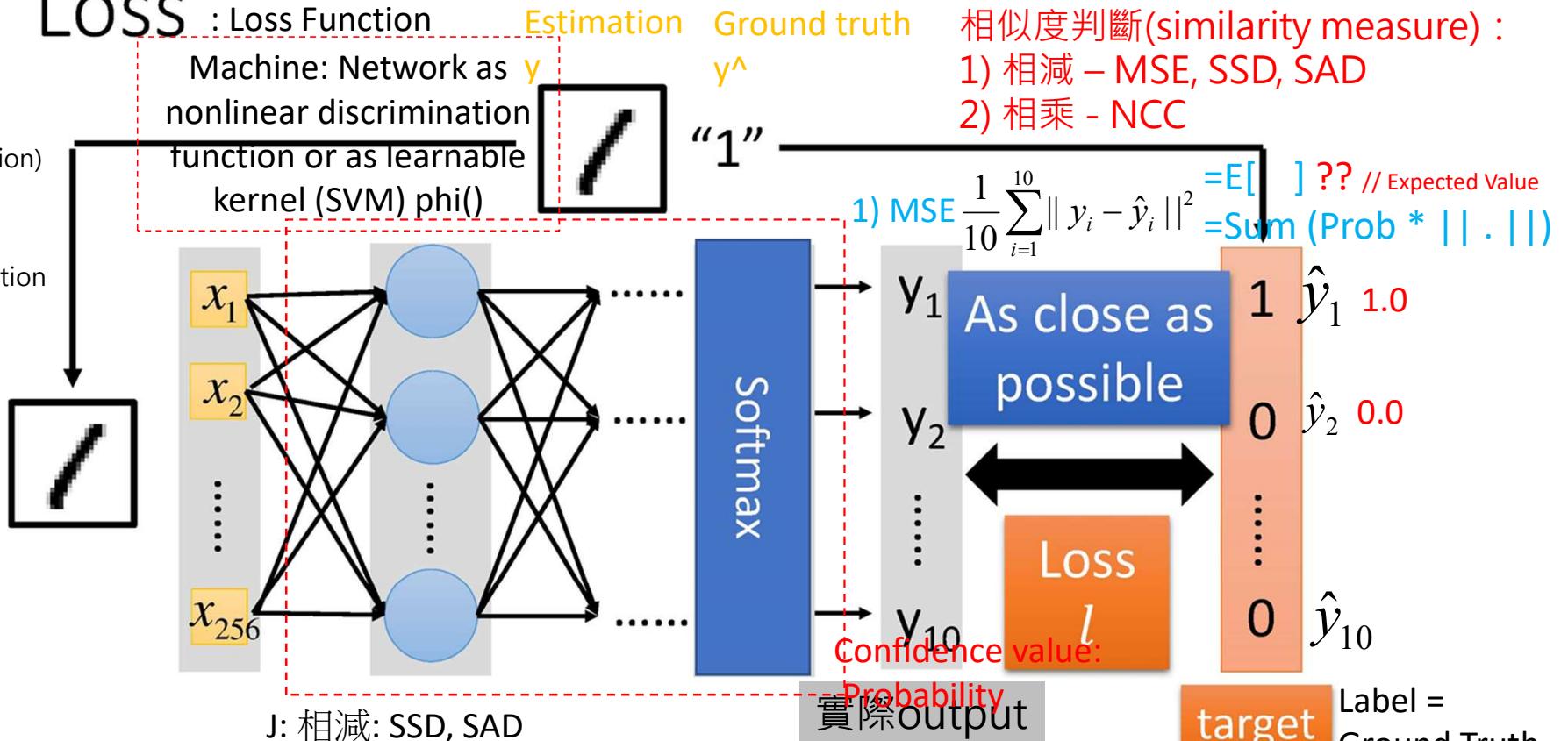
$$Ax = b$$

?? ?

最佳化方式：

1. EM (Expectation-maximization)
 - (1) 初始值 = mean (expected)
 - (2) 求max likelihood
 - (3) 不斷進行(1)~(2)iteration
2. Maximum likelihood estimation
3. 基因演算法
4. Random: Particle filter

LOSS



Loss can be square error or cross entropy

between the network output and target

Estimation O/P
probability y

Ground truth y^{\wedge}

2) Cross entropy
= Input training data and output expected results (= target)
 $0.0 < - \sum_{i=1}^{10} (\hat{y}_i \log y_i)$:- Because y is prob. 0.0~1.0, so log y will be -, therefore -- = +
 $y^{\wedge}=1$ Yes, $y^{\wedge}=0$ No

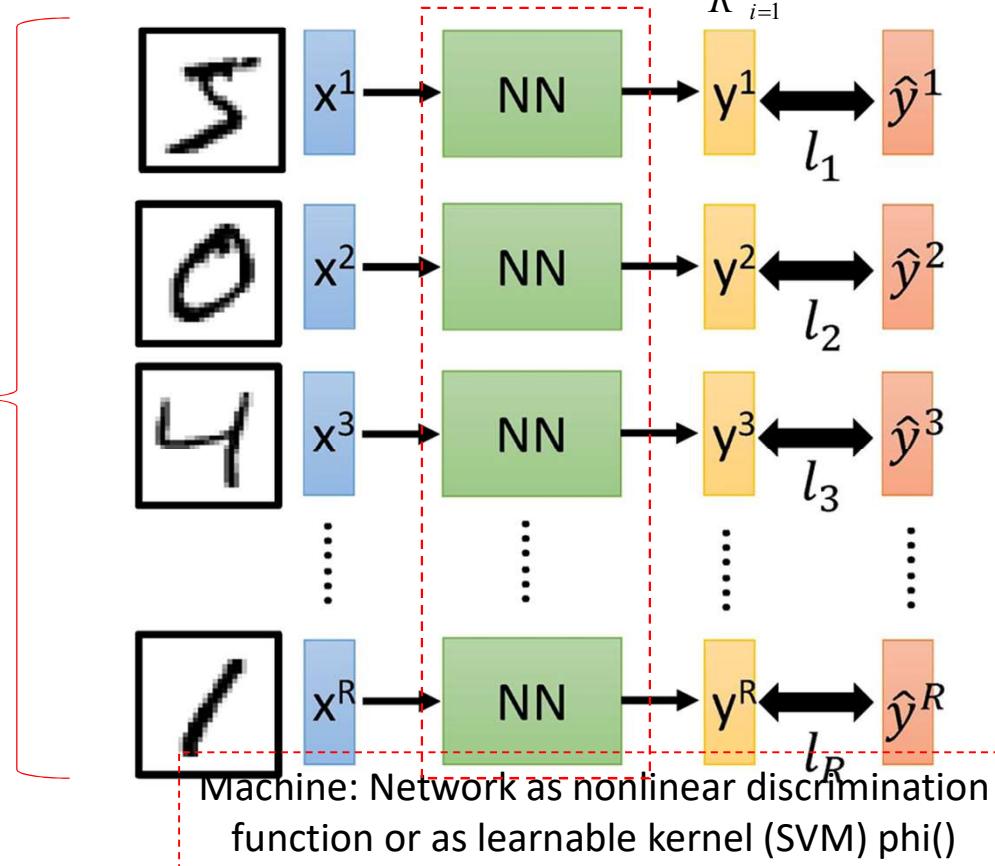
Loss function

= cost function

= Entropy

Total Loss

For all training data ... $\frac{1}{R} \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$



1) MSD: $1/R \sum (y - y')^2$

Where estimation $y = Ax$ by computation
 y' : target value, expected value, ground truth
 $A(w,b)$ (=theta): Unknown parameters,

Total Loss: $L = \min \frac{1}{R} \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$

$$L = \sum_{r=1}^{R=10} l_r$$

=E[] ?? // Expected Value
=Sum (Prob * || . ||)

Learning Target

Find the network parameters θ^* that minimize total loss L

累積所有training data，計算total loss，根據total loss及梯度下降法更新parameters θ^*

逐例學習 VS 批次學習(batch learning)

NN book提到的逐例學習和批次學習

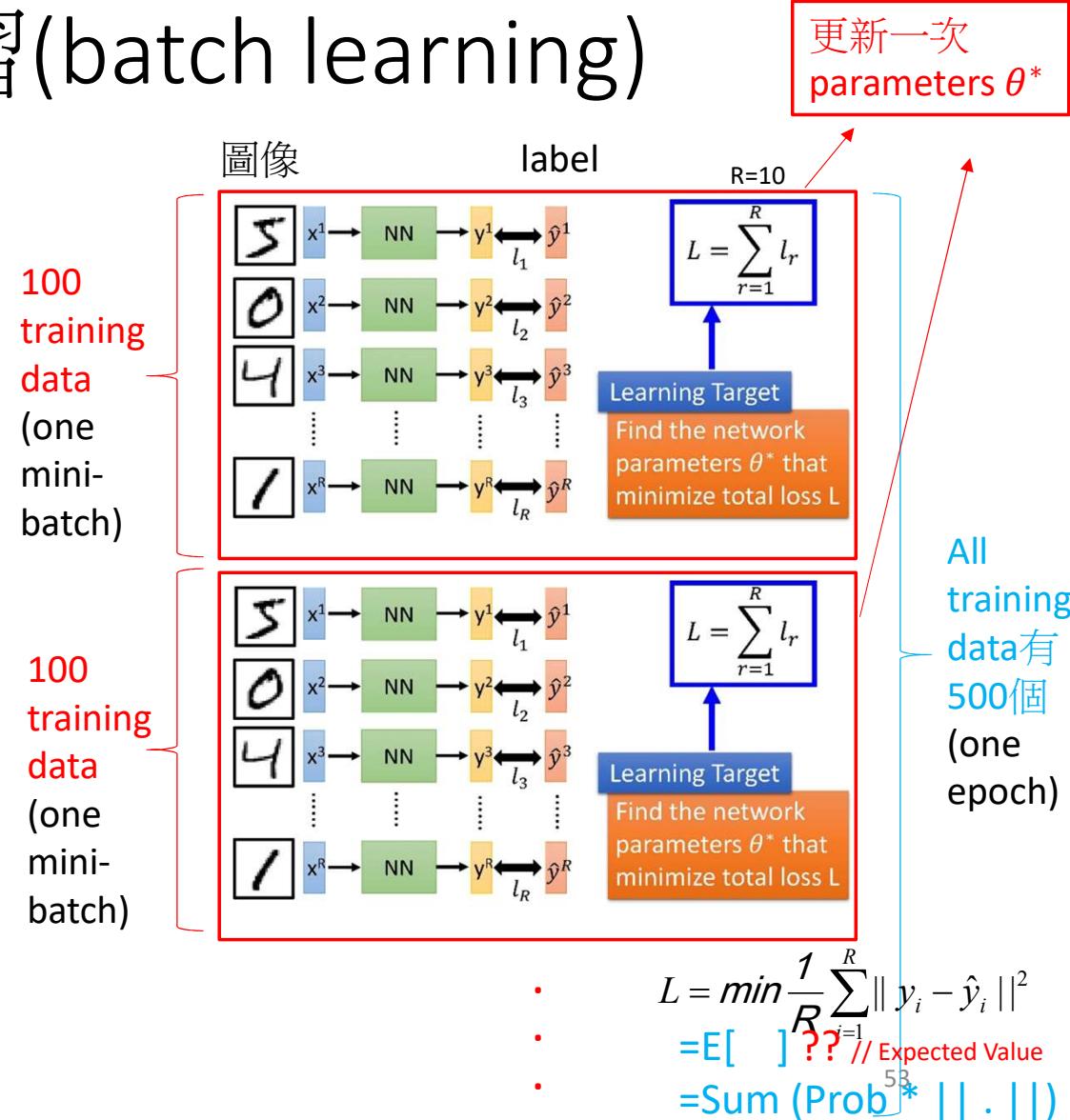
五、加權值更新方式

BP 學習過程通常是每載入一個範例，計算加權值的修正量後，立即更新加權值，稱之為「逐例學習」。如果學習過程改成不立即更新加權值，而是等到所有訓練範例均載入一次後，才將各範例的加權值修正量累加，更新加權值，稱之為「加權值累積式更新」或稱「批次學習」(batch learning)。如果訓練範例數衆多，加權值的修正量累加值可能很大，因此批次學習必須取較小的學習速率，一般可取原學習速率除以訓練範例數數目平方根。實證證明，對 BP 算法而言，通常「逐例學習」優於「批次學習」。

這個講義提到的mini-batch：

一些訓練範例載入網路，根據誤差總和
計算parameters θ^* 修正量

(parameters θ^* 的修正量不是根據
所有訓練範例(500 training data)計算的total loss，
而是根據每個mini-batch(100 training data)計算的loss總和)



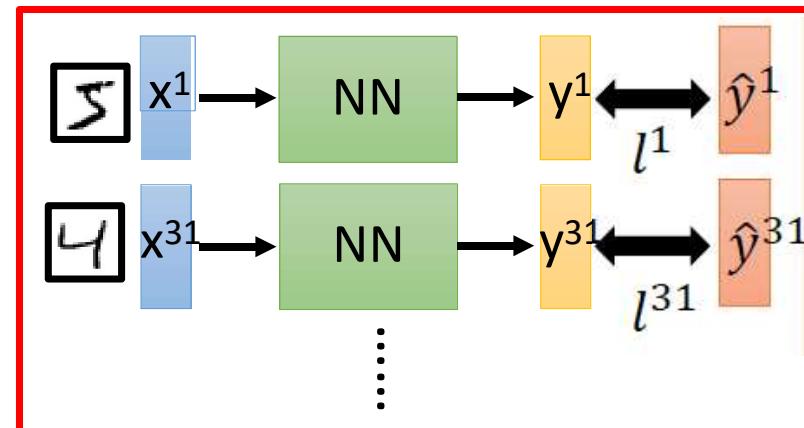
We do not really minimize total loss!

Mini-batch

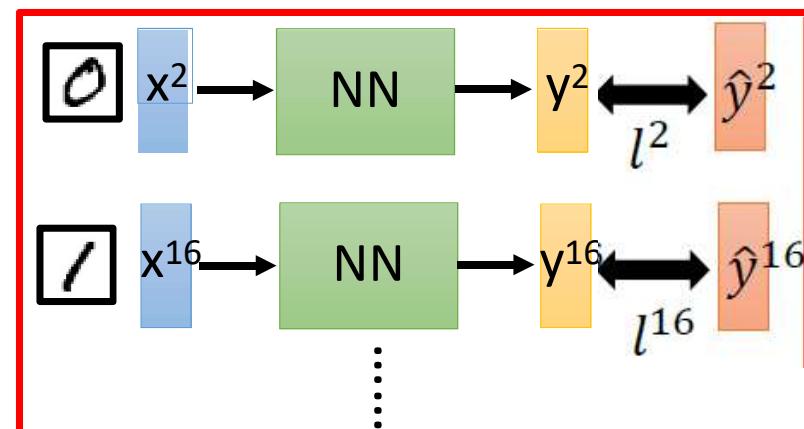
Originally 500
training samples:

Each mini-batch:
uses 100 training
samples

Mini-batch



Mini-batch



(1) Randomly initialize
network parameters

(2) one epoch

- Pick the 1st batch
 $L' = l^1 + l^{31} + \dots$
Update parameters once
- Pick the 2nd batch
 $L'' = l^2 + l^{16} + \dots$
Update parameters once
⋮
- Until all mini-batches
have been picked

(3)~(x) Repeat the above epoch

Three Steps for Deep Learning

I.1.
Step3

Design topology/network:
Having parameters (as variable)
without values

Step 1:
Network
Structure

How to I/O data? Target/Data:
Given input data and be recognized
/ classified output data. *Exp. Mini
batch*

Step 2:
Learning
Target
 $o/p \Leftrightarrow$ Ground truth (label) y'

How to learn? Learning parameters
(or para values) of topology using
gradient descent to find global min
of MSE

Step 3:
Learn!



天生的腦



$$L = \frac{1}{R} \min \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$$

How to Learn?

Solution for this problem:
 1) Gradient decent
 2) Random approximation
 Particle filter, andy....

Step 2:

Learning Target

Find the network parameters θ^* that minimize total loss

Step 3:

Learning

Solving the optimization problem

Brute force:
 列舉出 parameter 的所有可能組合
 → 不可行！

Enumerate all possible values

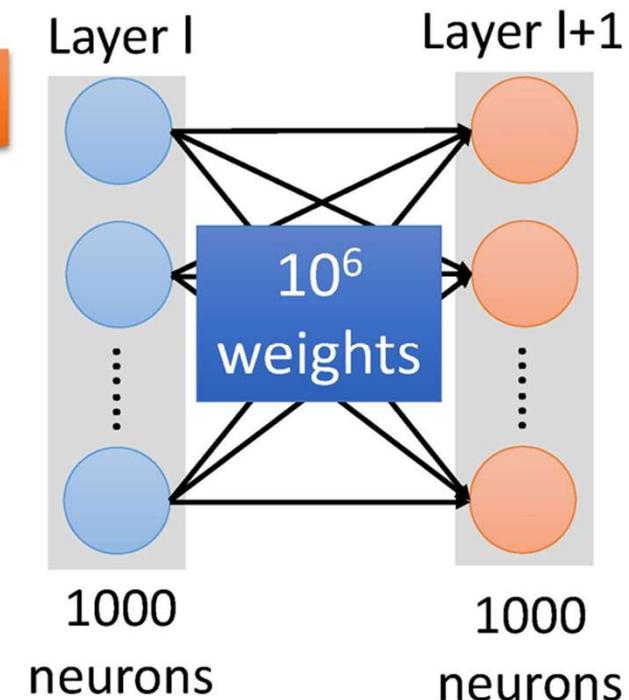
Network parameters $\theta =$

$\{w_1, w_2, w_3, \dots, b_1, b_2, b_3, \dots\}$

Millions of parameters

-One weak classifier is one
 haar filter is also as one
 neuron unit
 -One strong classifier is as one
 layer

E.g. speech recognition: 8 layers and
 1000 neurons each layer

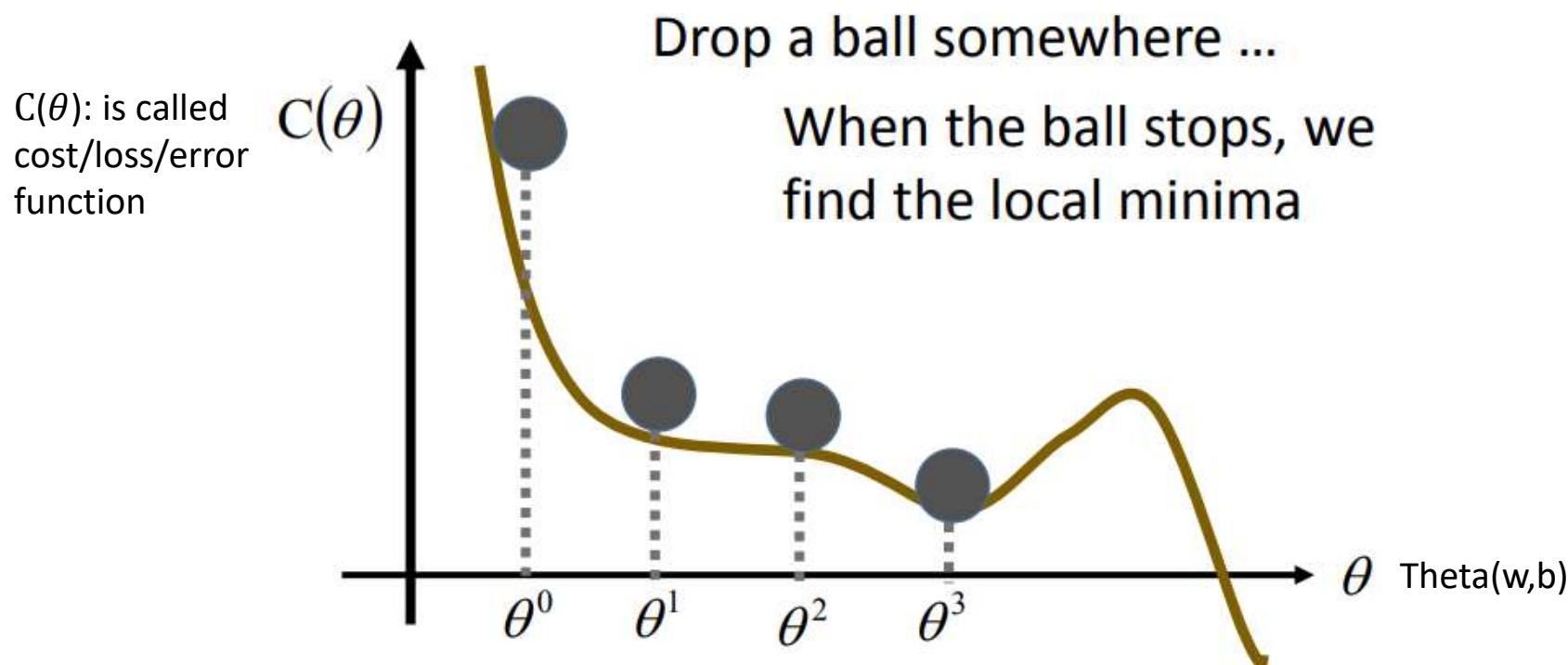


Reference:

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20\(v4\).pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20(v4).pdf)

Gradient Descent – Idea

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



Gradient Descent

Network parameters $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

Step 2:

Learning Target

Find the network parameters θ^* that minimize total loss

其他方法：
Monte-Carlo(蒙地卡羅法)：
是一種隨機模擬方法，以機率
和統計理論方法為基礎的一種
計算方法。



??

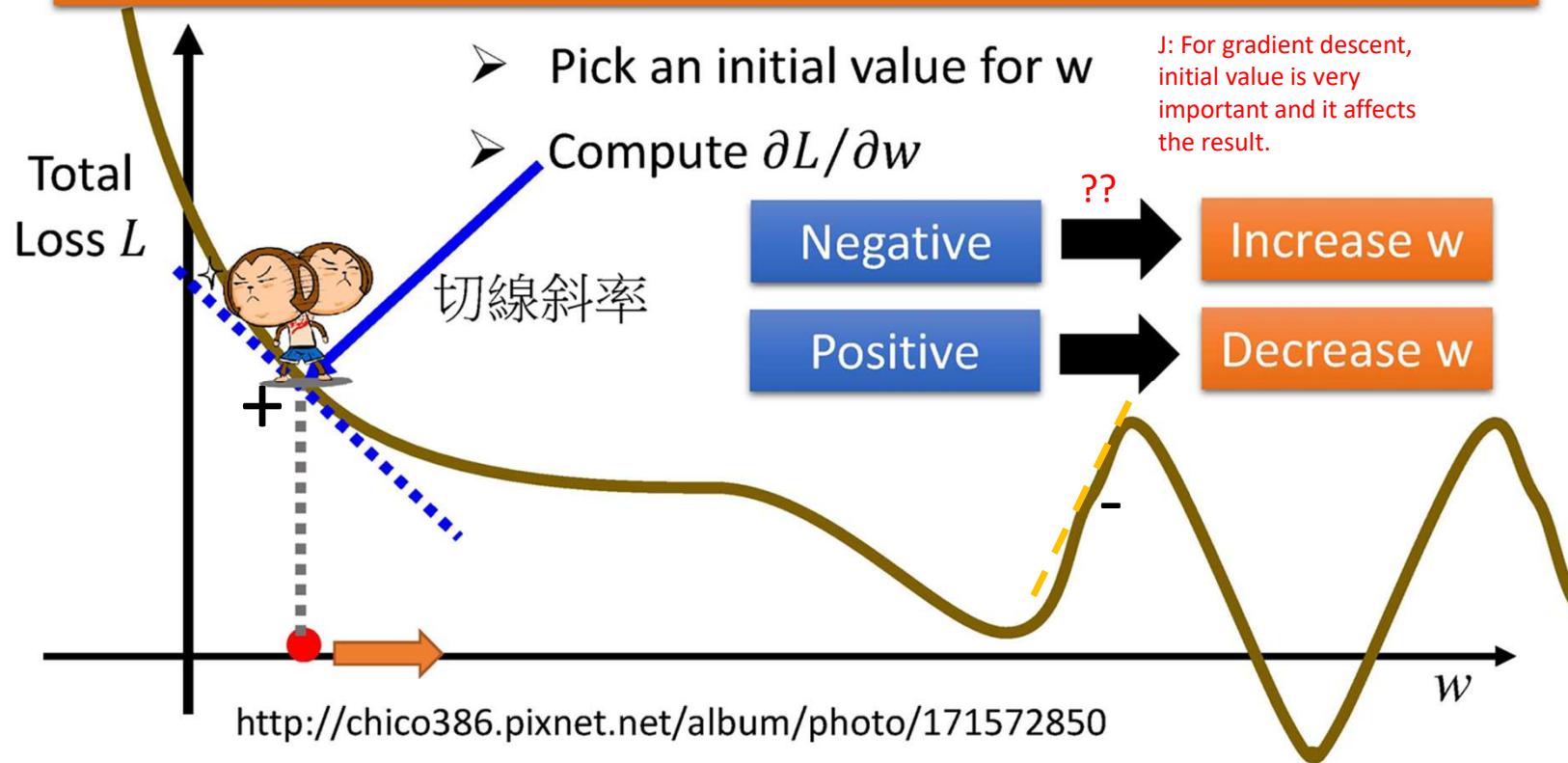
Gradient Descent

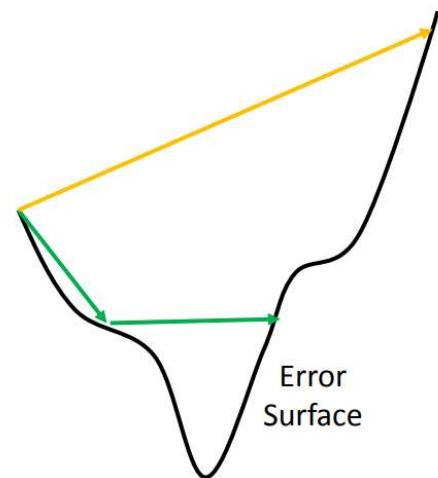
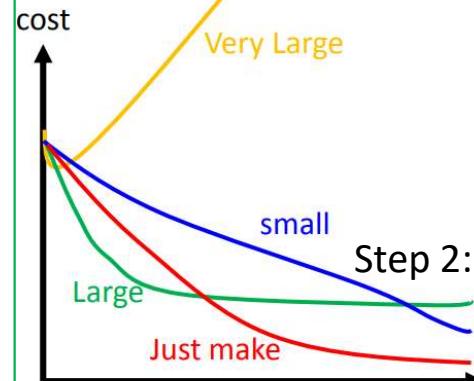
Network parameters $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

Step 2:

Learning Target

Find the network parameters θ^* that minimize total loss





Set the learning rate η carefully

Learning rate: (movement)

取的少，跳很慢

取的大，可能跳過global min

Learning rate is given by experience

Gradient Descent

Learning Target

Find the network parameters θ^* that minimize total loss

➤ Pick an initial value for w

➤ Compute $\partial L / \partial w$

$$w \leftarrow w - \eta \partial L / \partial w$$

Repeat

η is called
"learning rate"

$$-\eta \partial L / \partial w$$

60

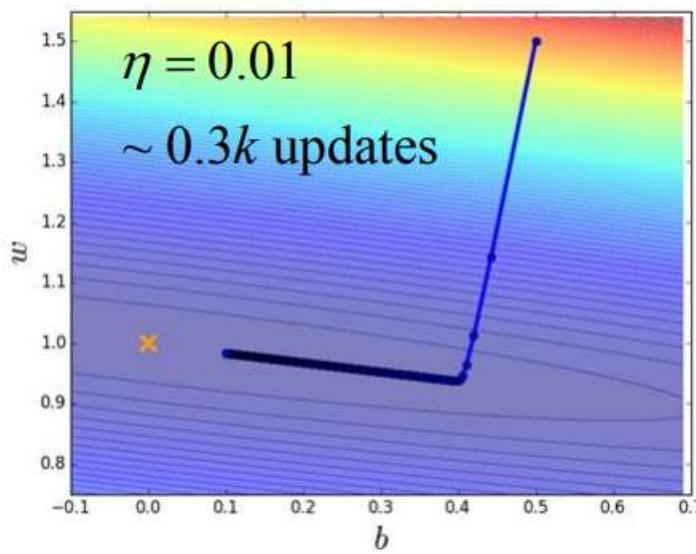
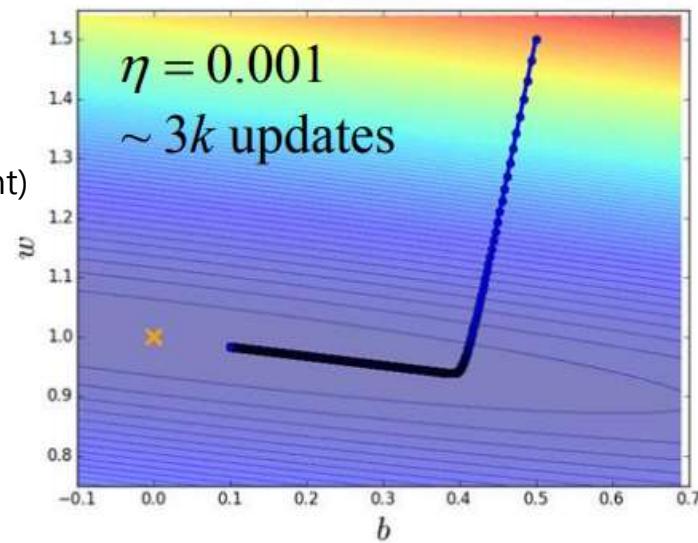
?? Learning Rate

- Toy Example

Different learning rate η

??
收敛速度快

J: Learning rate: (movement)
取的很小，跳的很慢
收敛速度很慢，
iteration many times



震盪，到不了min

J: Learning rate: (movement)
取的大，可能跳過global min

??
收敛速度較快

J: Learning rate: (movement)
取的適中，收斂速度ok，
iteration ok times

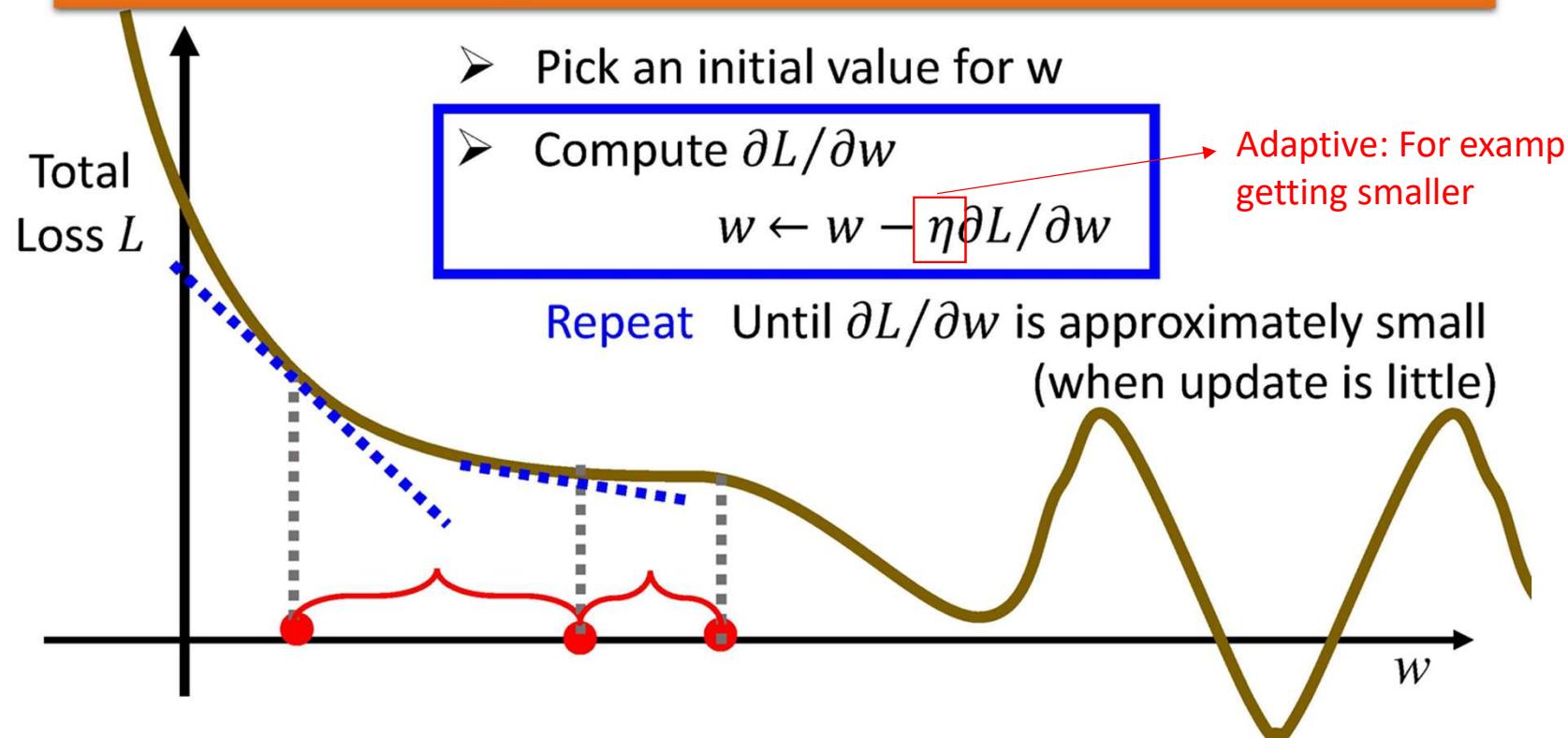
Gradient Descent

Network parameters $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

Step 2:

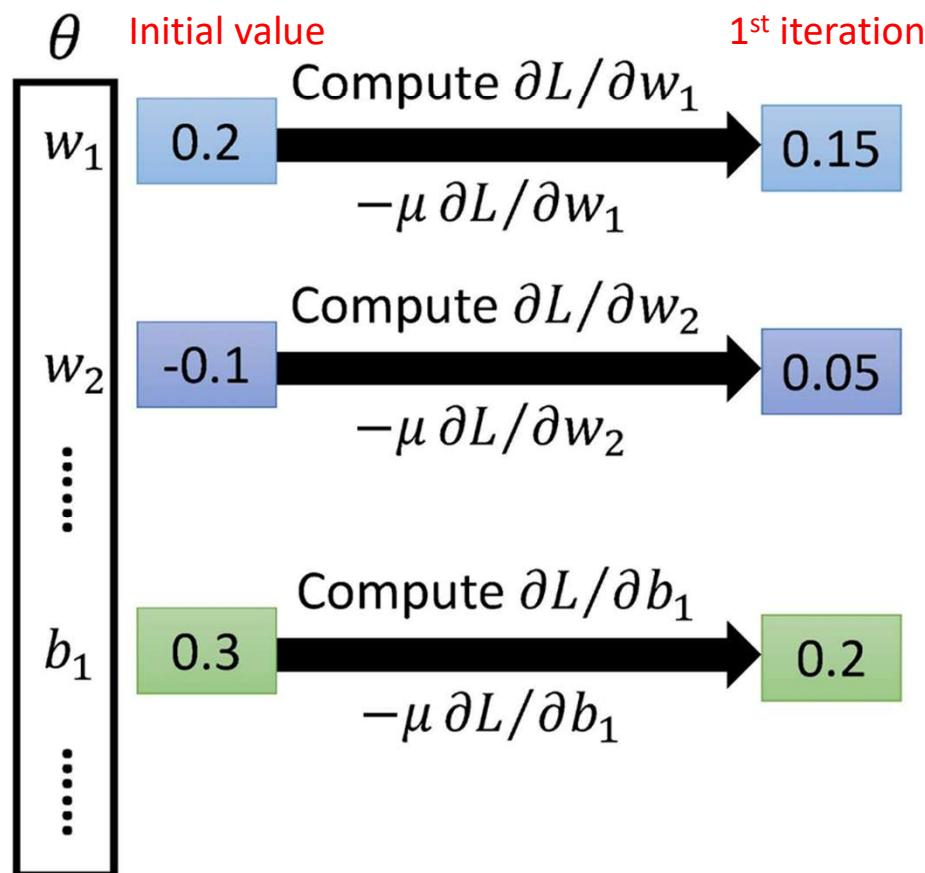
Learning Target

Find the network parameters θ^* that minimize total loss



$$Ax=b$$

Gradient Descent



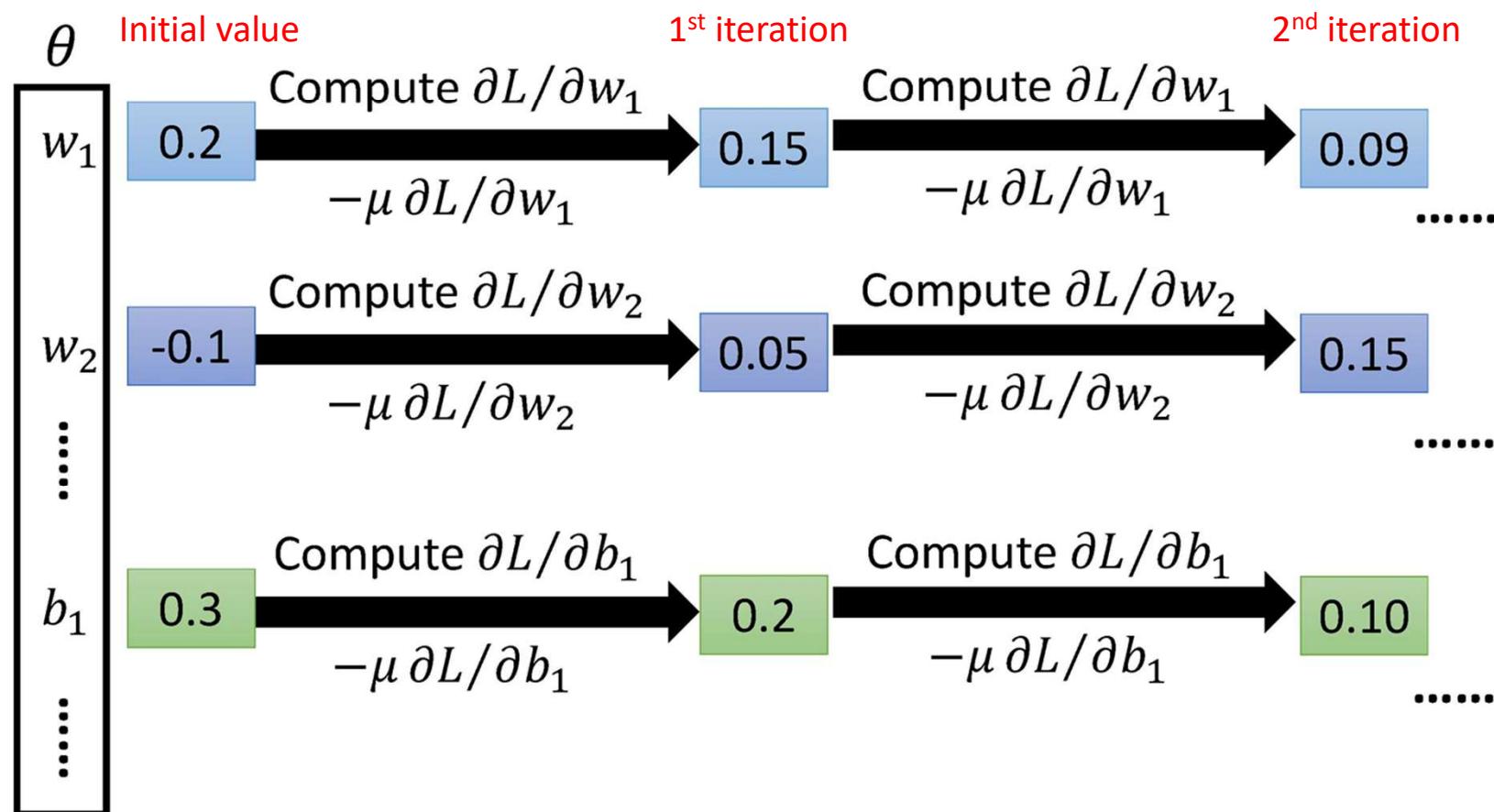
$$\begin{bmatrix} w_1 & w_2 & w_3 & \cdots & w_K & 1 \end{bmatrix}_{J(-1) \times (k+1)}^T = z$$

$$\begin{bmatrix} b \\ b \\ b_3 \\ \vdots \\ b_K \\ 1 \end{bmatrix}_{(k+1) \times 1} = z$$

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

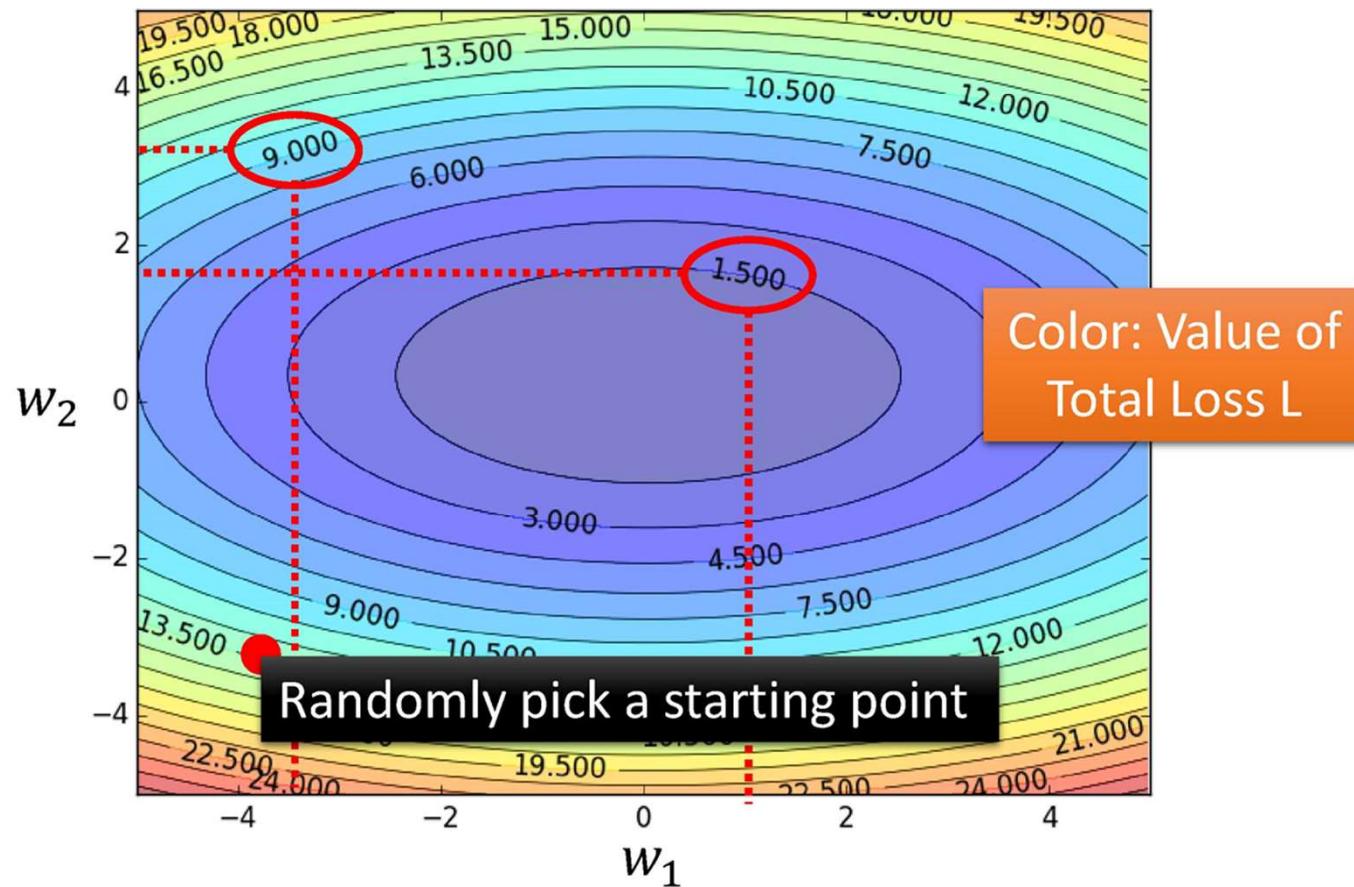
gradient

Gradient Descent



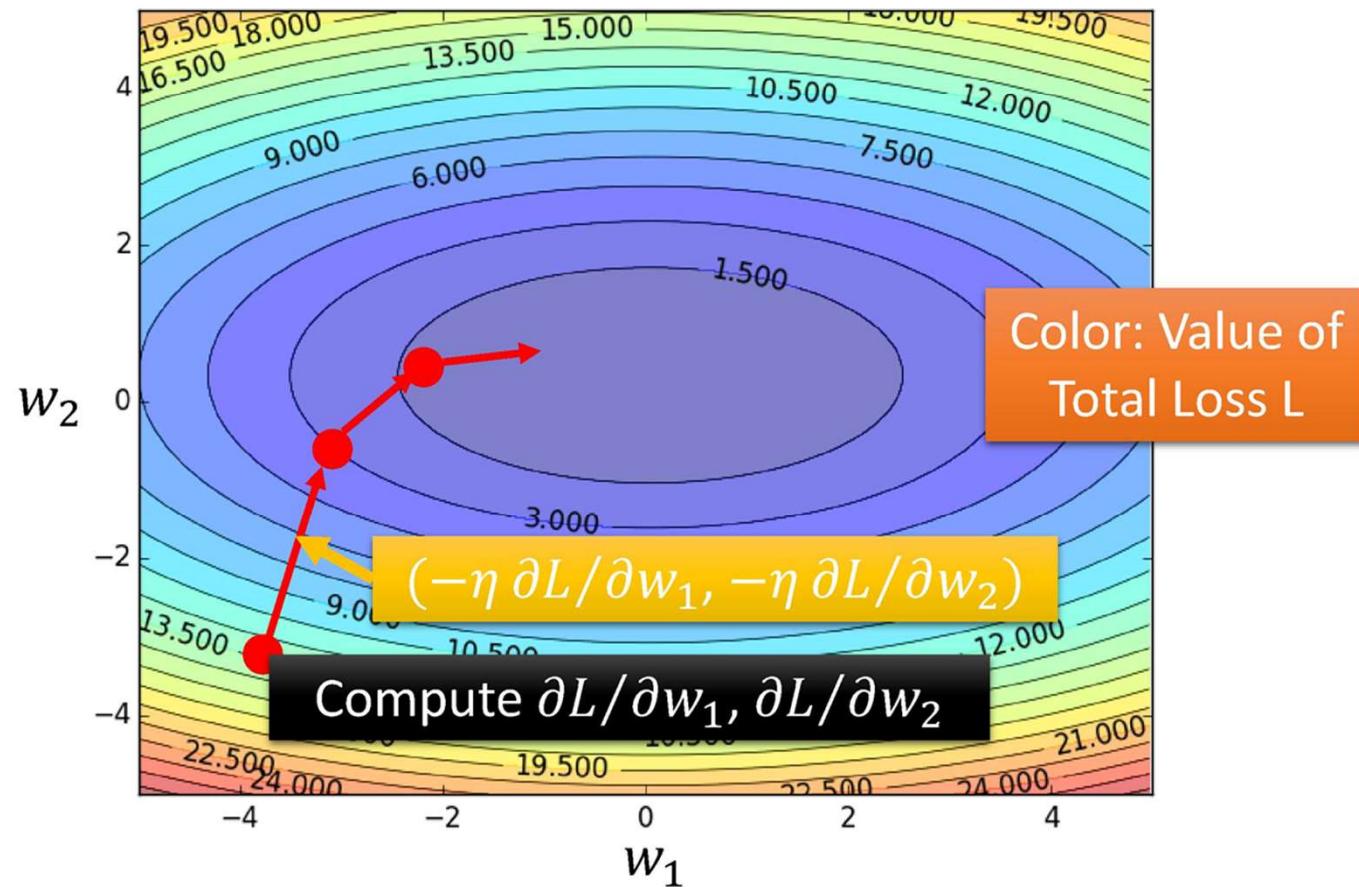
Gradient Descent

Suppose that θ has two variables $\{w_1, w_2\}$



Gradient Descent

Hopfully, we would reach
a minima



??

Reference: [http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20\(v4\).pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20(v4).pdf)
<https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent>

Stochastic Gradient Descent

and Mini-batch

$$C(\theta) = \frac{1}{R} \sum_r \|f(x^r; \theta) - \hat{y}^r\| = E[\hat{y}^r] // \text{Expected Value}$$
$$= \frac{1}{R} \sum_r C^r(\theta) = \text{Sum (Prob * || . ||)}$$

◆ Gradient Descent

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

◆ Stochastic Gradient Descent

Faster!

Better!

Pick an example x^r

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

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

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

run through **ALL** the samples in your training set to do a single update

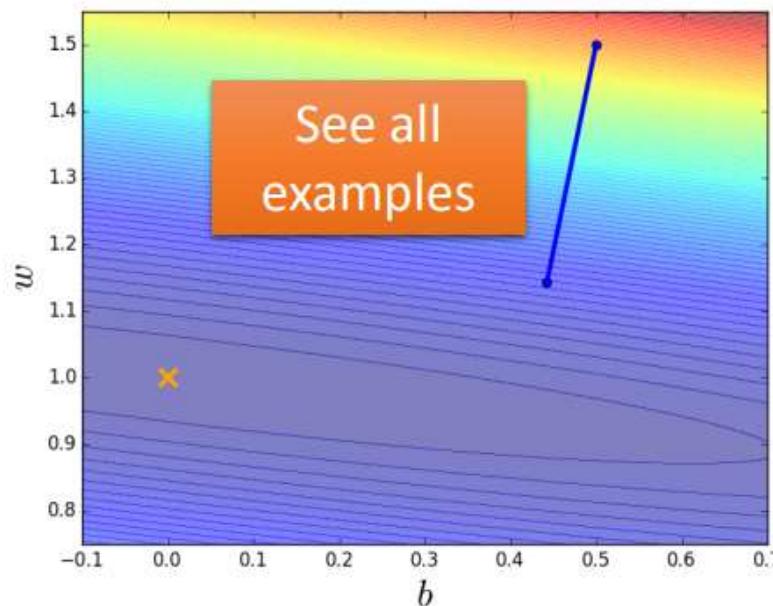
use **ONLY ONE** of training sample from your training set to do the update

Stochastic Gradient Descent and Mini-batch

- Toy Example

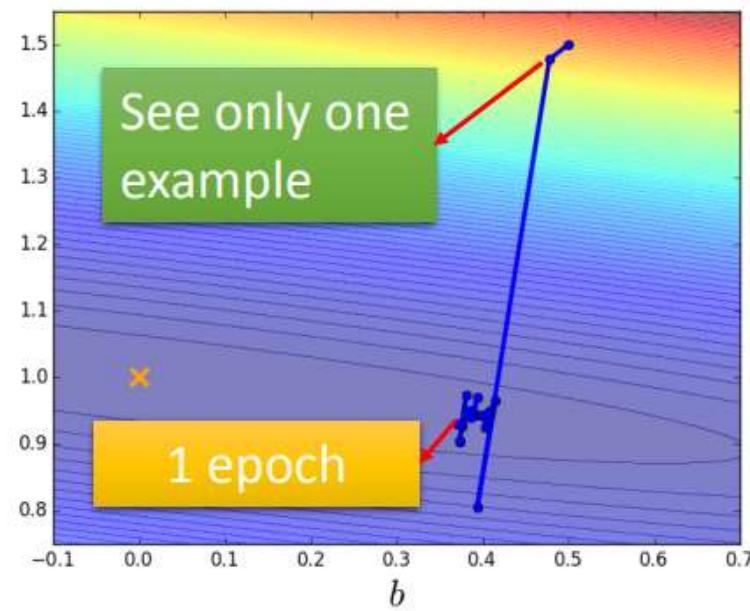
Gradient Descent

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

What is epoch?

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

When using stochastic gradient descent

Starting at θ_0 pick x^1 $\theta^1 = \theta^0 - \eta \nabla C^1(\theta^0)$

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

\vdots

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

\vdots

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

Seen all the
examples once
One epoch

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

Stochastic Gradient Descent and Mini-batch

1. Performance order: ???
- 1) Stochastic Gradient Descent
- 2) Mini Batch Gradient
- 3) Gradient Descent

◆ **Gradient Descent** (Batch Gradient Descent) $= \text{Sum} (\text{Prob} * || \cdot ||)$

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

◆ **Stochastic Gradient Descent**

Pick an example x_r

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

◆ **Mini Batch Gradient Descent**

Pick B examples as
a batch b

B is batch size

Shuffle your data

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

Average the gradient of the
examples in the batch b

➤ Why??

1. Speed order: ???
- 1) Mini Batch Gradient
- 2) Stochastic Gradient Descent
- 3) Gradient Descent

➤ Why??

Stochastic Gradient Descent and Mini-batch

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

Stochastic Gradient Descent

$$z^1 = \begin{matrix} W^1 \\ x \end{matrix}$$
$$z^1 = \begin{matrix} W^1 \\ x \end{matrix} \dots\dots$$

Mini-batch

$$\begin{matrix} z^1 & z^1 \end{matrix} = \begin{matrix} W^1 \\ \text{matrix} \end{matrix} \begin{matrix} x & x \end{matrix}$$

b = Ax

Practically, which one is faster?

Gradient Descent

- When considering multiple parameters together, do you see any problem?



Gradient Descent

This is the “learning” of machines in deep learning

Keep finding min value??

→ Even alpha go using this approach.

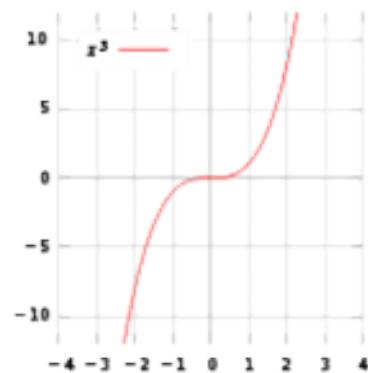
大家以為 Learning 是



其實 Learning 只是



I hope you are not too disappointed :p



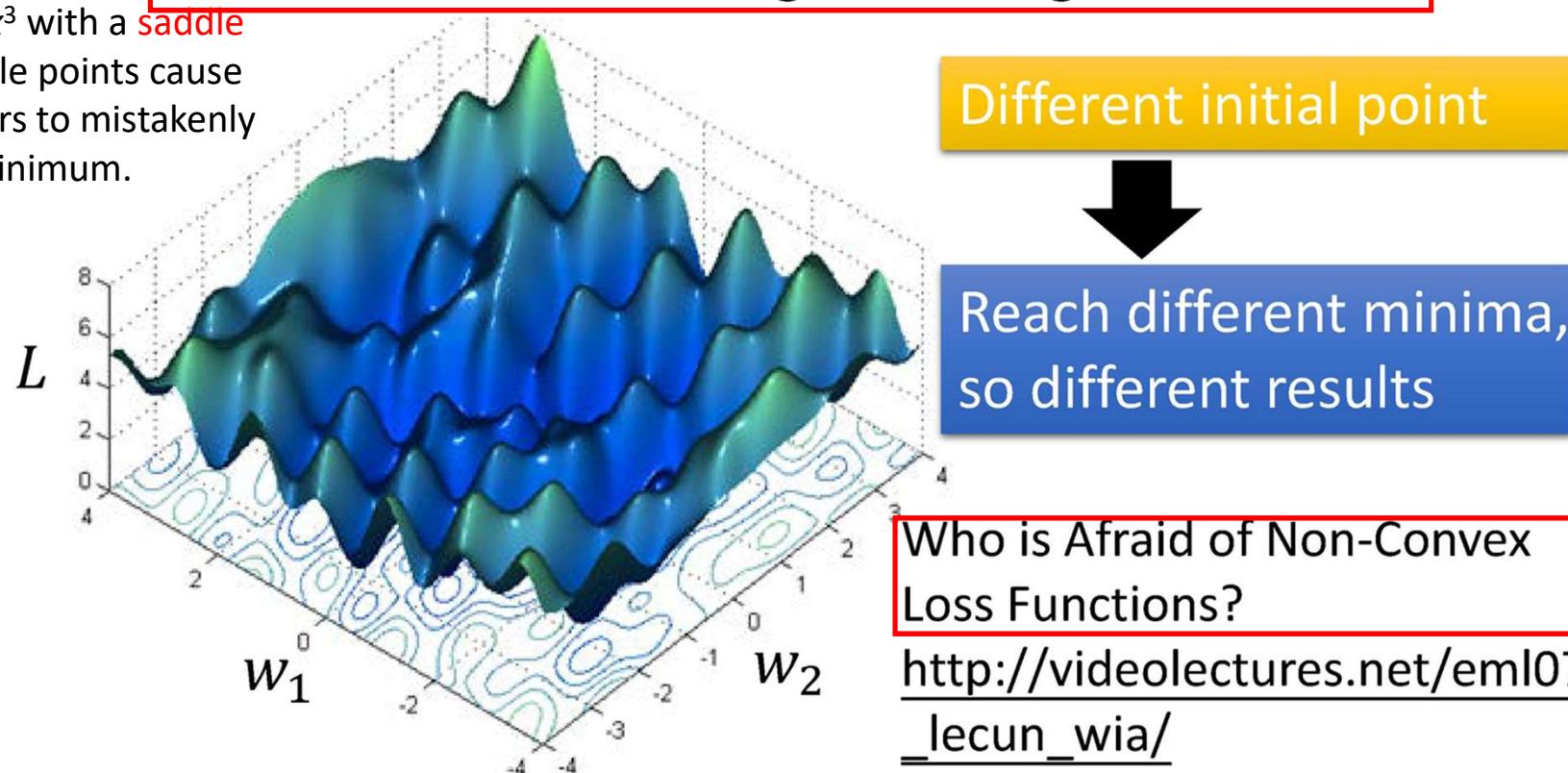
Local Minima

- 1) Saddle Point Problem
- 2) Non-Convex Problem

- Still find **saddle point**, which is worse than local minimum.
- Solution is trying to use different initial points.
- So solution....

- Gradient descent never guarantee global minima

The plot of $y = x^3$ with a **saddle point** at 0. Saddle points cause the programmers to mistakenly find the local minimum.



?? Why call BP??

How ??

Backpropagation

??

- Backpropagation: an efficient way to compute $\partial L / \partial w$
- Ref:
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html



theano

libdnn
台大周伯威
同學開發

Caffe

Microsoft
CNTK

The Chainer logo features a white molecular-like icon next to the word "Chainer" in a white sans-serif font.

The mxnet logo features a blue rounded rectangle containing the letters "mxnet" in white, with a white circular icon to the left of the 'm'.

Don't worry about $\partial L / \partial w$, the toolkits will handle it.

Backpropagation: Background

If we want to train a neural network, we need to **define**
a cost function first.

- Cost Function $C(\theta)$

- Given training examples:

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

- Find a set of parameters θ^* minimizing $C(\theta)$

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

- Gradient Descent $= \text{Sum} (\text{Prob} * ||.||)$

$$\nabla C(\theta) = \frac{1}{R} \sum_r \nabla C^r(\theta)$$

Calculated value Expected value = target
If we want to find a set of parameter θ^* , we
use the **gradient descent** to solve it.

- Given w_{ij}^l and b_i^l , we have to compute $\partial C^r / \partial w_{ij}^l$ and $\partial C^r / \partial b_i^l$

Network parameters

Trained neural network to get
the estimation result $y=f()$ for
each epoch

- There is an efficient way to compute the gradients of the network parameters – **backpropagation**.

因為今天我們要訓練一個neural network而network parameter非常非常的多，一個一個去計算cost function的偏微分是不切實際的。

Review 1

Total Loss

MSD: $1/R \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$
 where $y \leftarrow Ax$ by computation
 y' : target value, expected value
 $A(w,b)$ ($=\theta$): Unknown parameters,

Loss function
 = cost function
 = Entropy

For all training data ...

$$\frac{1}{R} \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$$

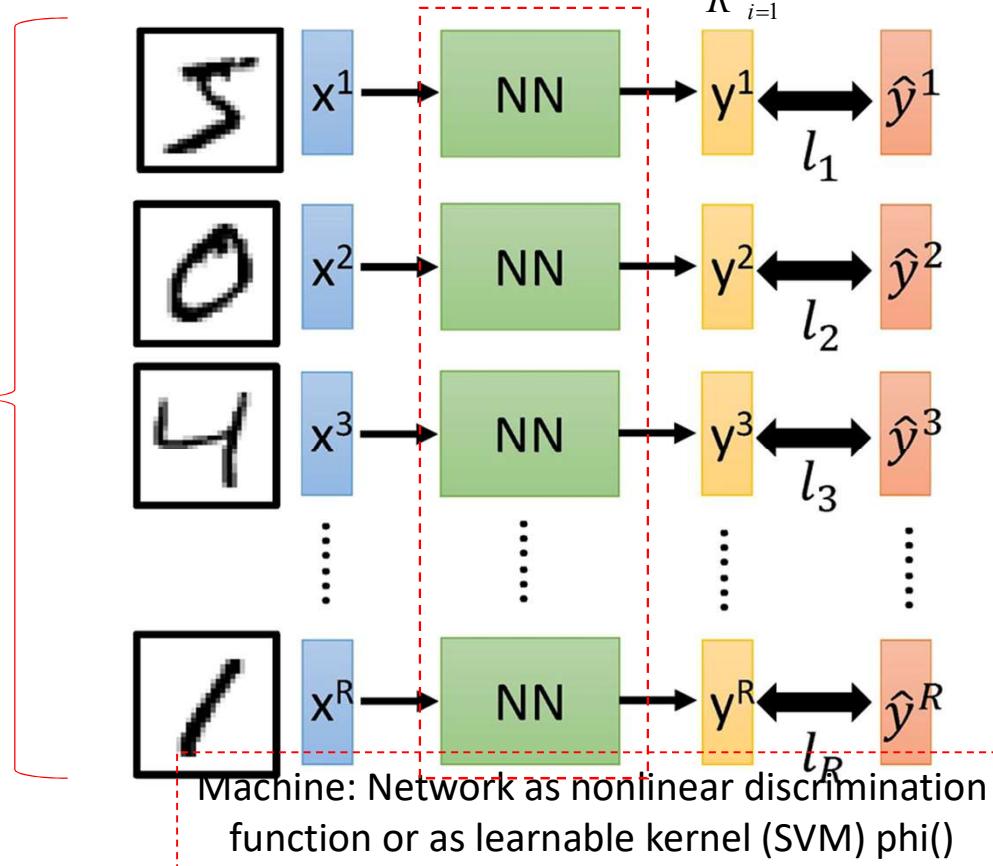
Total Loss:

$$L = \min \frac{1}{R} \sum_{i=1}^R \|y_i - \hat{y}_i\|^2$$

$$= E[\quad] ?? // \text{Expected Value}$$

$$= \text{Sum} (\text{Prob} * || \cdot ||)$$

500 training data



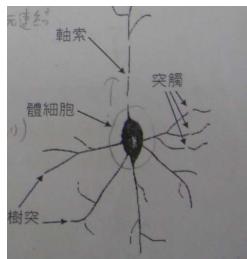
Learning Target

Find the network parameters θ^* that minimize total loss L

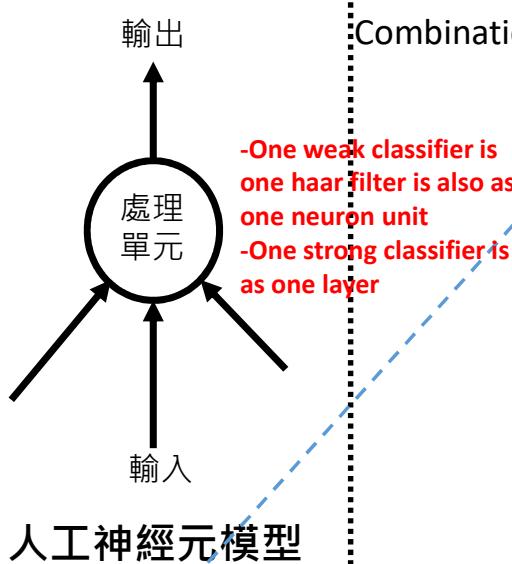
$$L = \sum_{r=1}^R l_r$$

累積所有training data，計算total loss，根據total loss及梯度下降法更新parameters θ^*

Review NN (Ch1) :



生物神經元模型



$$y_i = f(\sum_i w_i x_i - \theta) \quad \text{constant}$$

bias

A neuron unit

Review 2

Neural Network

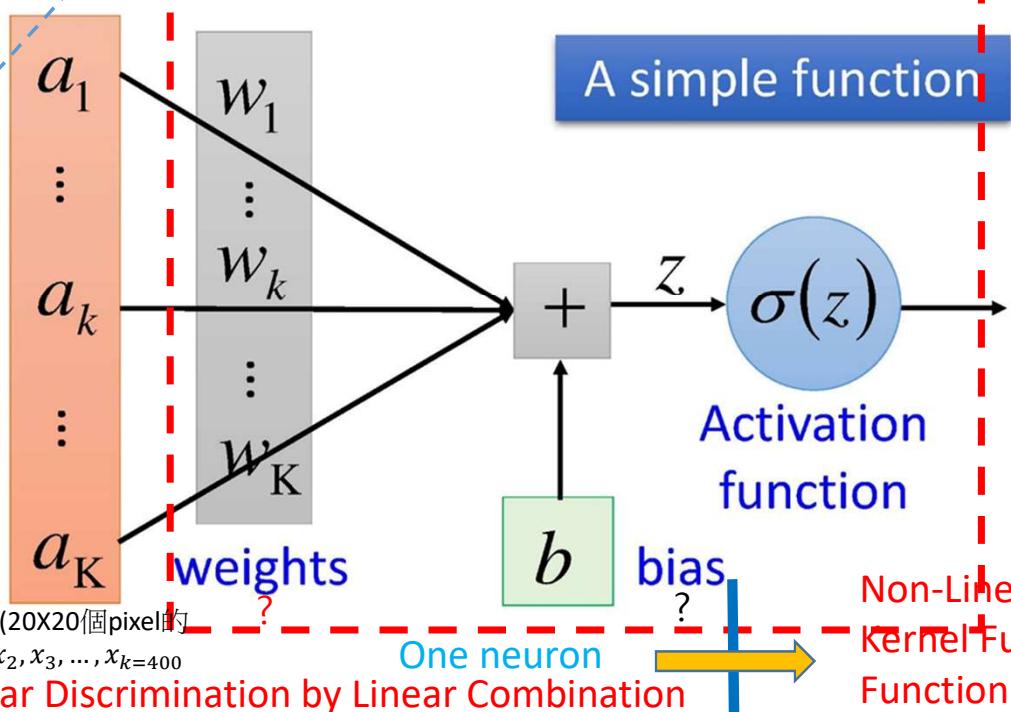
One **Neuron**

- One weak classifier is one haar filter is also as one neuron unit
- One strong classifier is as one layer

Linear Combination:

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$

Constant, bias



寫程式不會以 equation 的方式，會以 vector, matrix ($Ax = b$), homogenous matrix, 的方式撰寫。

Linear

Combination: $Aw = z$

?

J input data vectors
(here J=1)

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

Review 3-1

Reference: [http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20\(v4\).pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20(v4).pdf)
<https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent>

Stochastic Gradient Descent and Mini-batch

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

=E[] ?? // Expected Value
=Sum (Prob * || . ||)

◆ Gradient Descent

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

◆ Stochastic Gradient Descent

Faster!

Better!

Pick an example x^r

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

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

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

run through **ALL** the
samples in your
training set to do a
single update

use **ONLY ONE** of
training sample from
your training set to do
the update

Stochastic Gradient Descent and Mini-batch

◆ Gradient Descent

(Batch Gradient Descent) $=\text{Sum}(\text{Prob} * || \cdot ||)$

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

◆ Stochastic Gradient Descent

Pick an example x_r

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

◆ Mini Batch Gradient Descent

Pick B examples as
a batch b

B is batch size

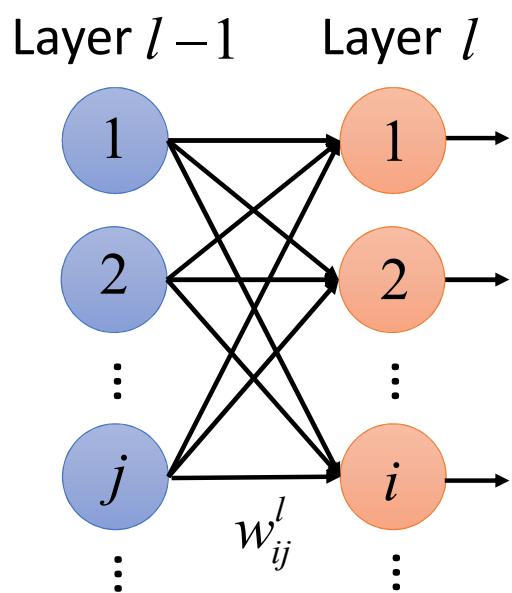
Shuffle your data

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

Average the gradient of the
examples in the batch b

Review 4

Concluding Remarks



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

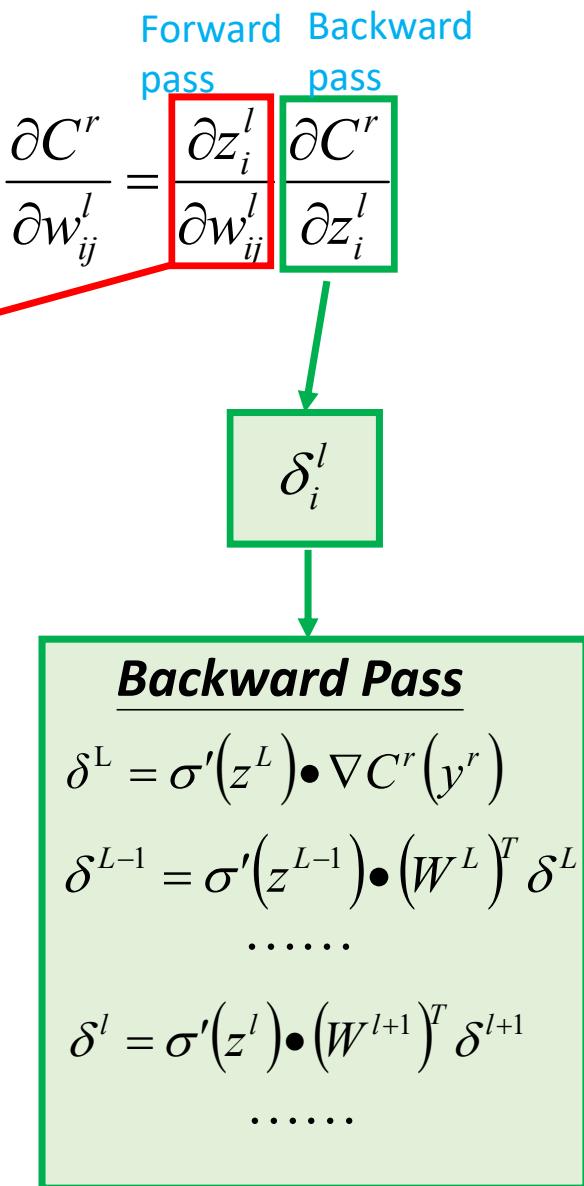
Forward Pass

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

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

$$\dots\dots$$

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

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


Review: Chain Rule

Case 1

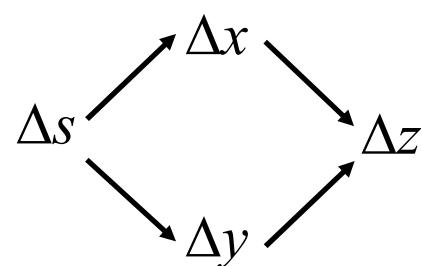
$$y = g(x) \quad z = h(y)$$

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

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Case 2

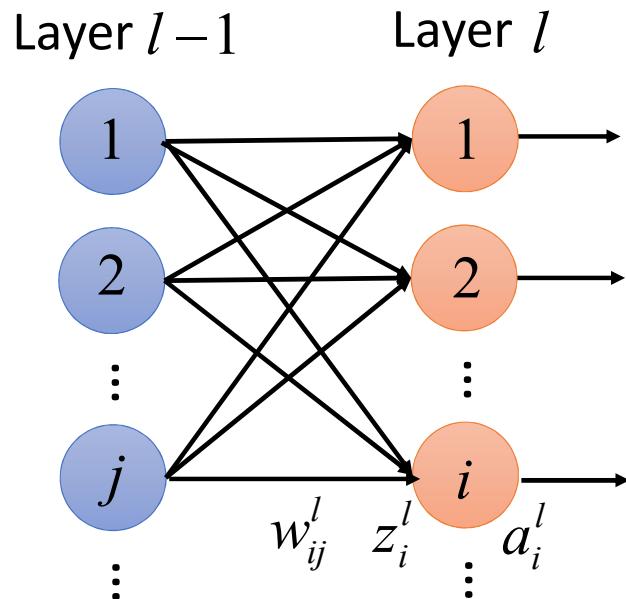
$$x = g(s) \quad y = h(s) \quad z = k(x, y)$$



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

舉例: 以 neural network 中的一個 network parameter (weighting)
對 cost function 做 partial derivative

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



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

According to the chain rule, we can get the following equation:

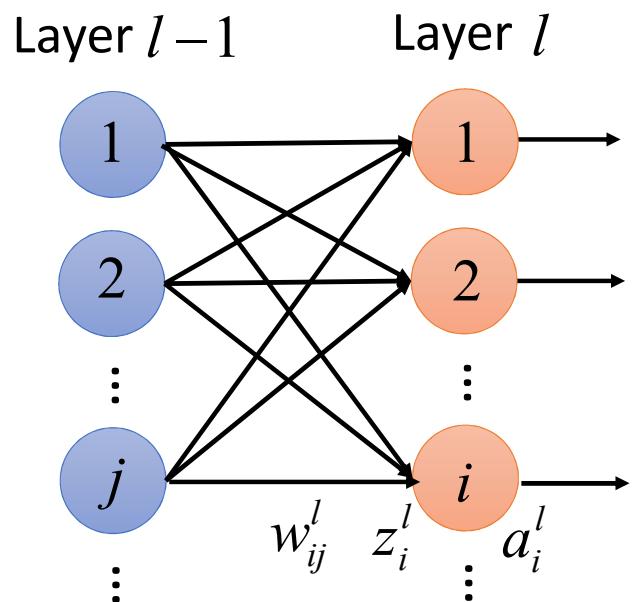
$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l}$$

Forward pass Backward pass

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

We will talk about how to calculate this two terms in the following lecture

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



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

The calculation of the first term will be easier than the second terms.

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

Forward pass Backward pass

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

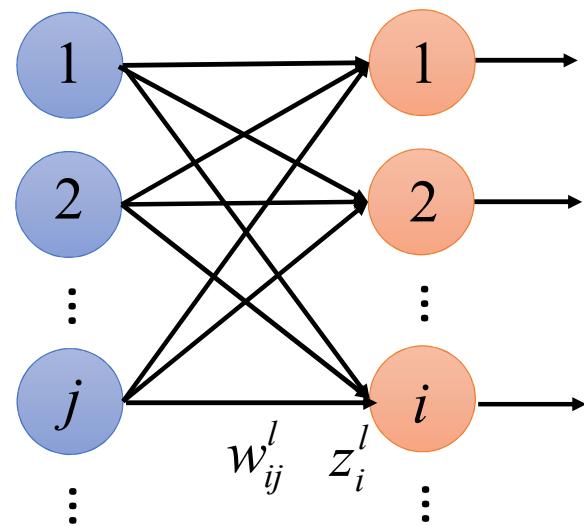
$\partial C^r / \partial w_{ij}^l$ - First Term

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

Forward Backward
pass pass

Layer l-1

Layer l



If $l > 1$

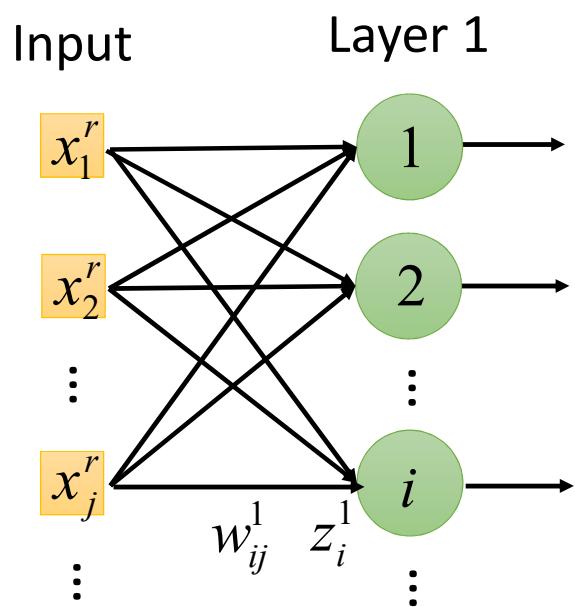
$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

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

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

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

Forward pass Backward pass



The output of the previous layer

If $l > 1$

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

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

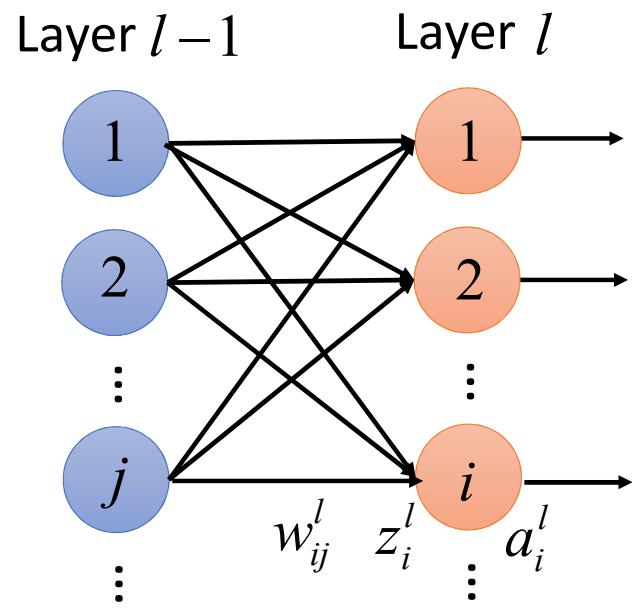
If $l = 1$

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

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

The input layer

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



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

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

Forward pass Backward pass

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

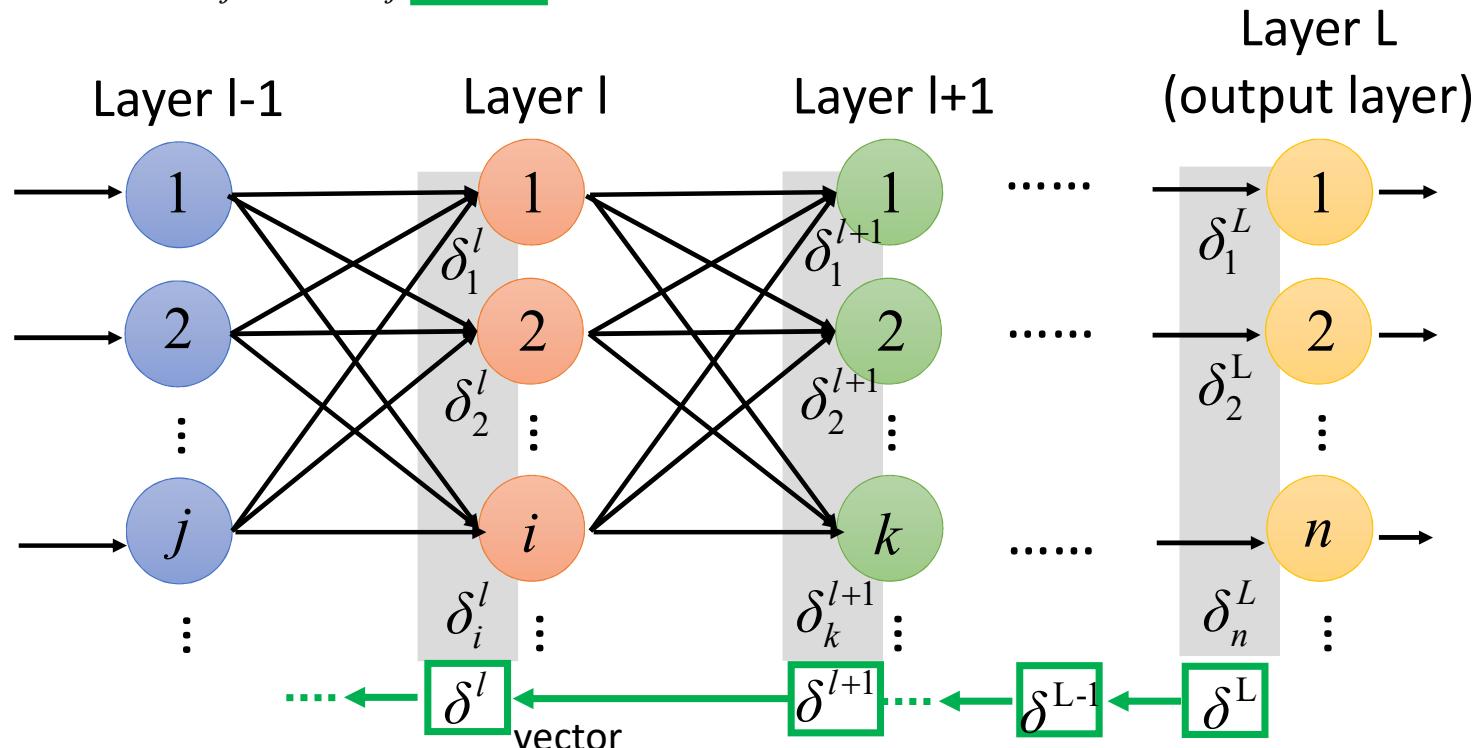
$\partial C^r / \partial w_{ij}^l$ - Second Term

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

Forward pass Backward pass

To fulfill this process (backpropagation), we need to solve the following problem:

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

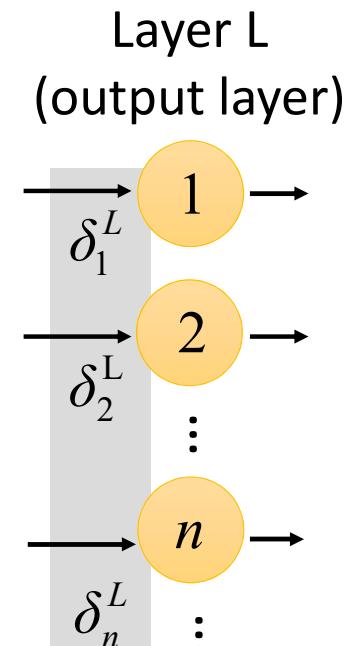
$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \xrightarrow{\text{pass}} \delta_i^l$$

Before calculate the δ^L , we first demonstrate how to calculate one of element δ_n^L of the δ^L

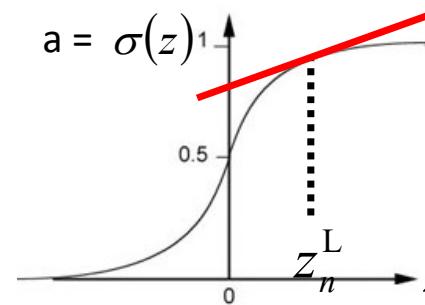
$$\begin{aligned}\delta_n^L &= \frac{\partial C^r}{\partial z_n^L} \\ &= \frac{\partial y_n^r}{\partial z_n^L} \frac{\partial C^r}{\partial y_n^r} \\ &\downarrow \\ &\sigma'(z_n^L)\end{aligned}$$

1. How to compute δ^L

2. The relation of δ^l and δ^{l+1}



Depending on the definition of cost function



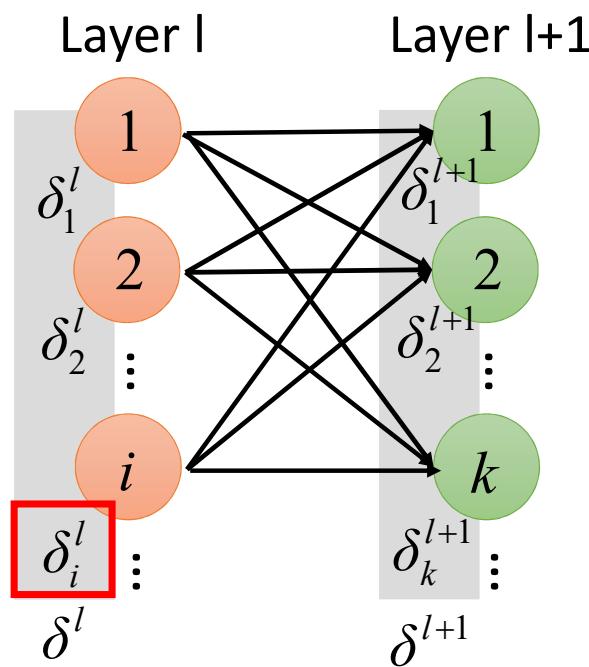
$$\underline{z_k^{l+1}} = \sum_i w_{ki}^l a_i^l + b_k^{l+1}$$

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

$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l} \rightarrow \delta_i^l$ <p style="margin-top: 10px;">$\delta_n^L = \frac{\partial C^r}{\partial z_n^L}$</p> $= \frac{\partial y_n^r}{\partial z_n^L} \frac{\partial C^r}{\partial y_n^r}$ $= \sigma'(z_n^L) \frac{\partial C^r}{\partial y_n^r}$	<p style="color: blue; font-weight: bold;">Forward Backward</p> <p style="margin-top: 10px;">$\delta^L?$</p> $\sigma'(z^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_n^L) \\ \vdots \end{bmatrix} \quad \nabla C^r(y^r) = \begin{bmatrix} \partial C^r / \partial y_1^r \\ \partial C^r / \partial y_2^r \\ \vdots \\ \partial C^r / \partial y_n^r \\ \vdots \end{bmatrix}$ <div style="border: 1px dashed red; padding: 5px; margin-top: 10px;"> $\delta^L = \underline{\sigma'(z^l)} \bullet \underline{\nabla C^r(y^r)}$ </div> <div style="background-color: #ADD8E6; border-radius: 5px; padding: 5px; margin-top: 10px;"> element-wise multiplication </div>
--	---

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

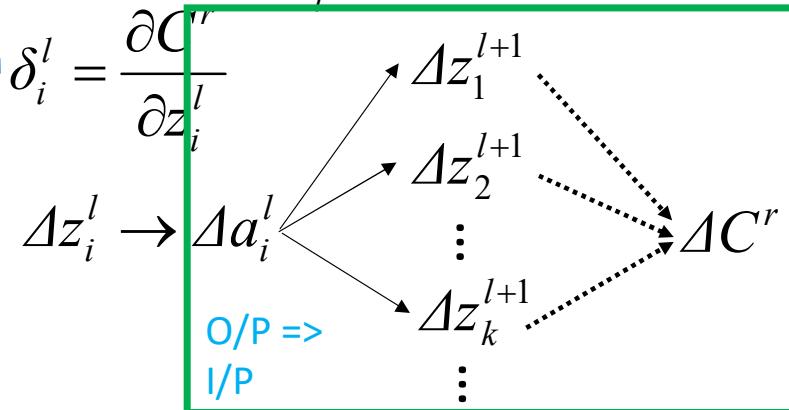
$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\text{Forward pass}}{\text{Backward pass}} \frac{\partial C^r}{\partial z_i^l} \rightarrow \delta_i^l$$



1. How to compute δ^L

2. The relation of δ^l and δ^{l+1}

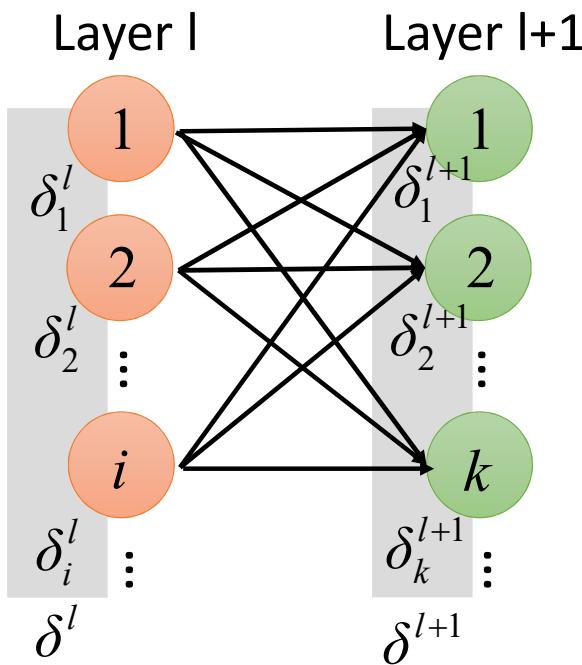
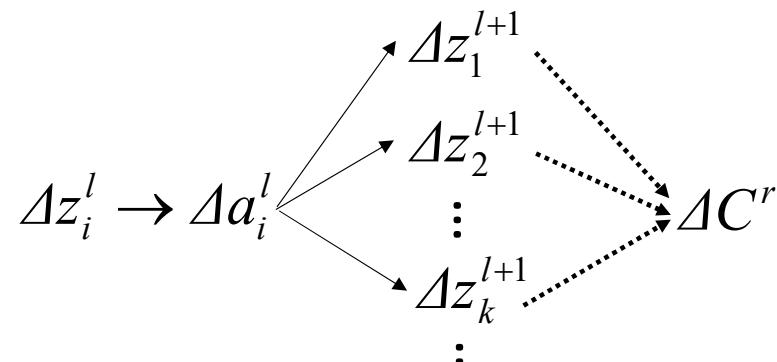
The definition of δ_i^l



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

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

$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C^r}{\partial z_i^l} \quad \begin{matrix} \text{Forward pass} \\ \text{Backward pass} \end{matrix}$$



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

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

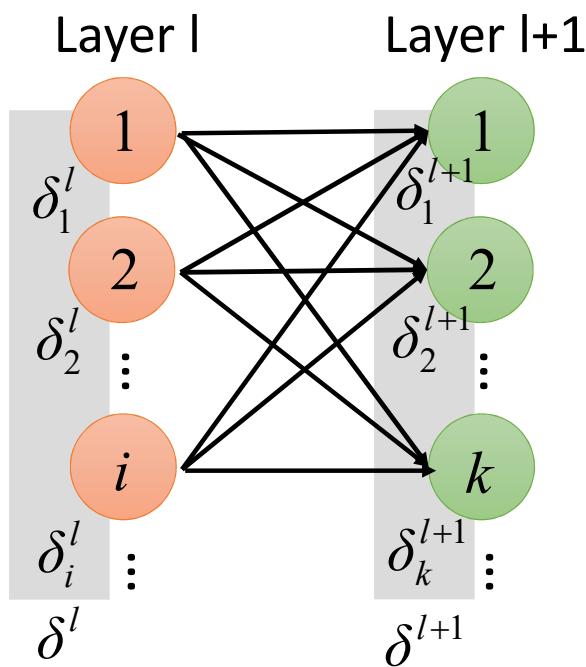
$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

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

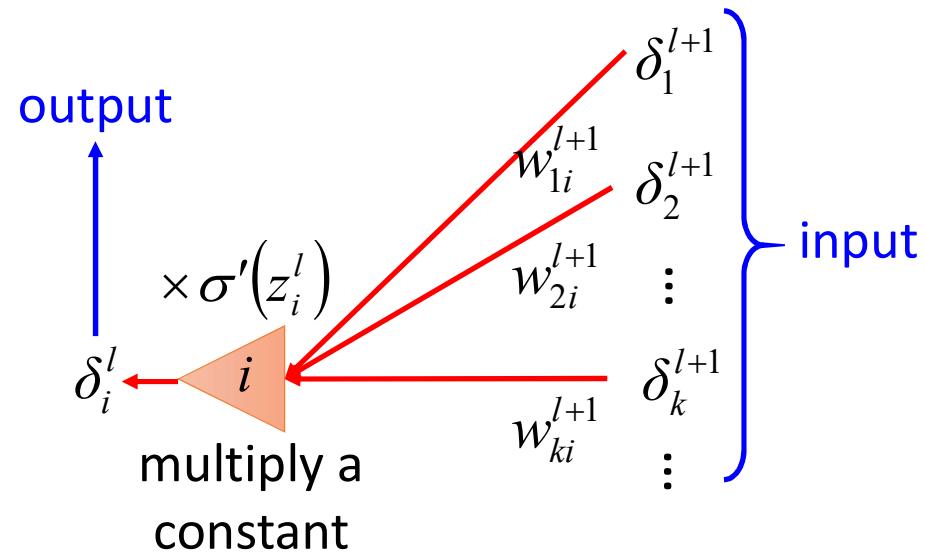
Forward pass Backward pass

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

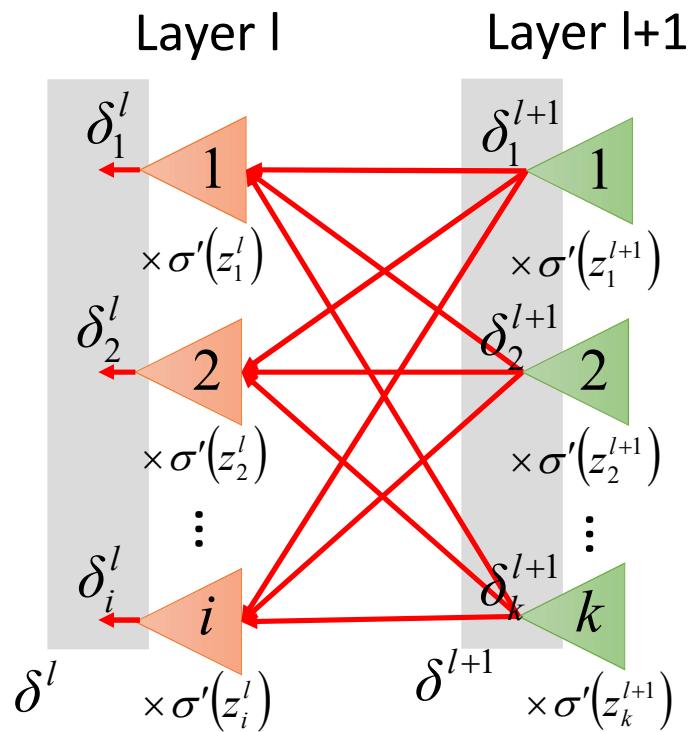
$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$



new type of neuron



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



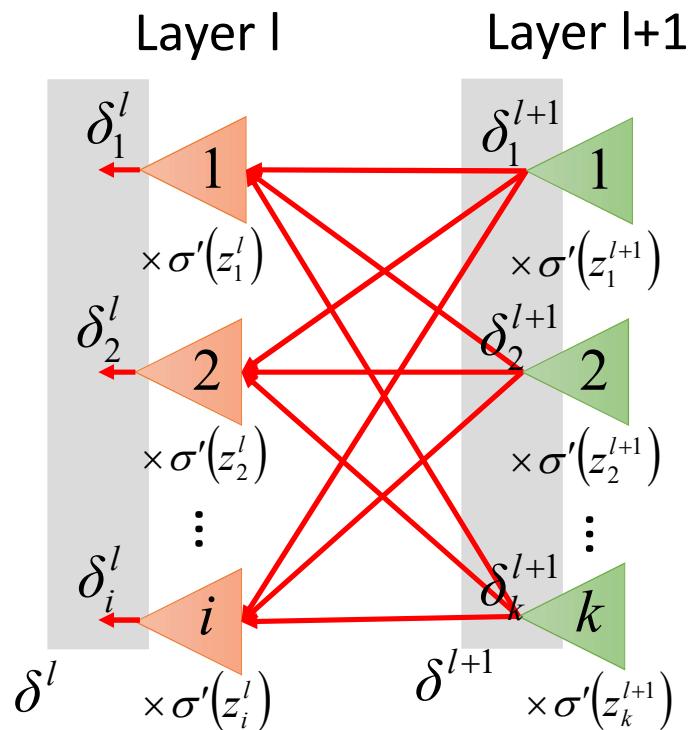
$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

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

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

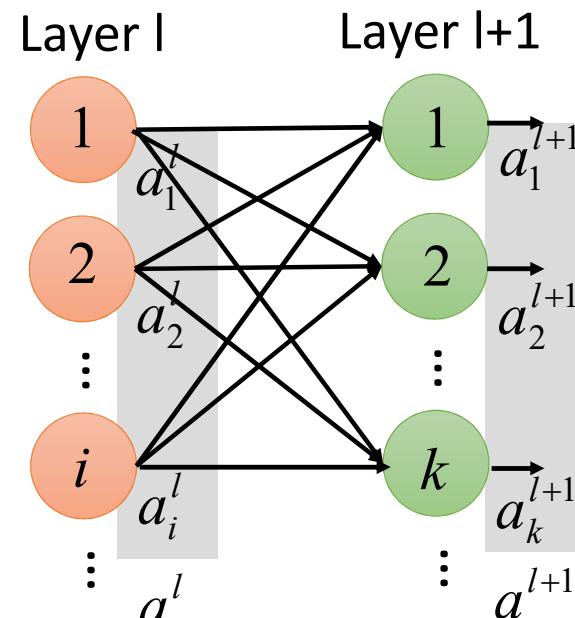
Weight matrix

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



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

Compare



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

Forward Backward

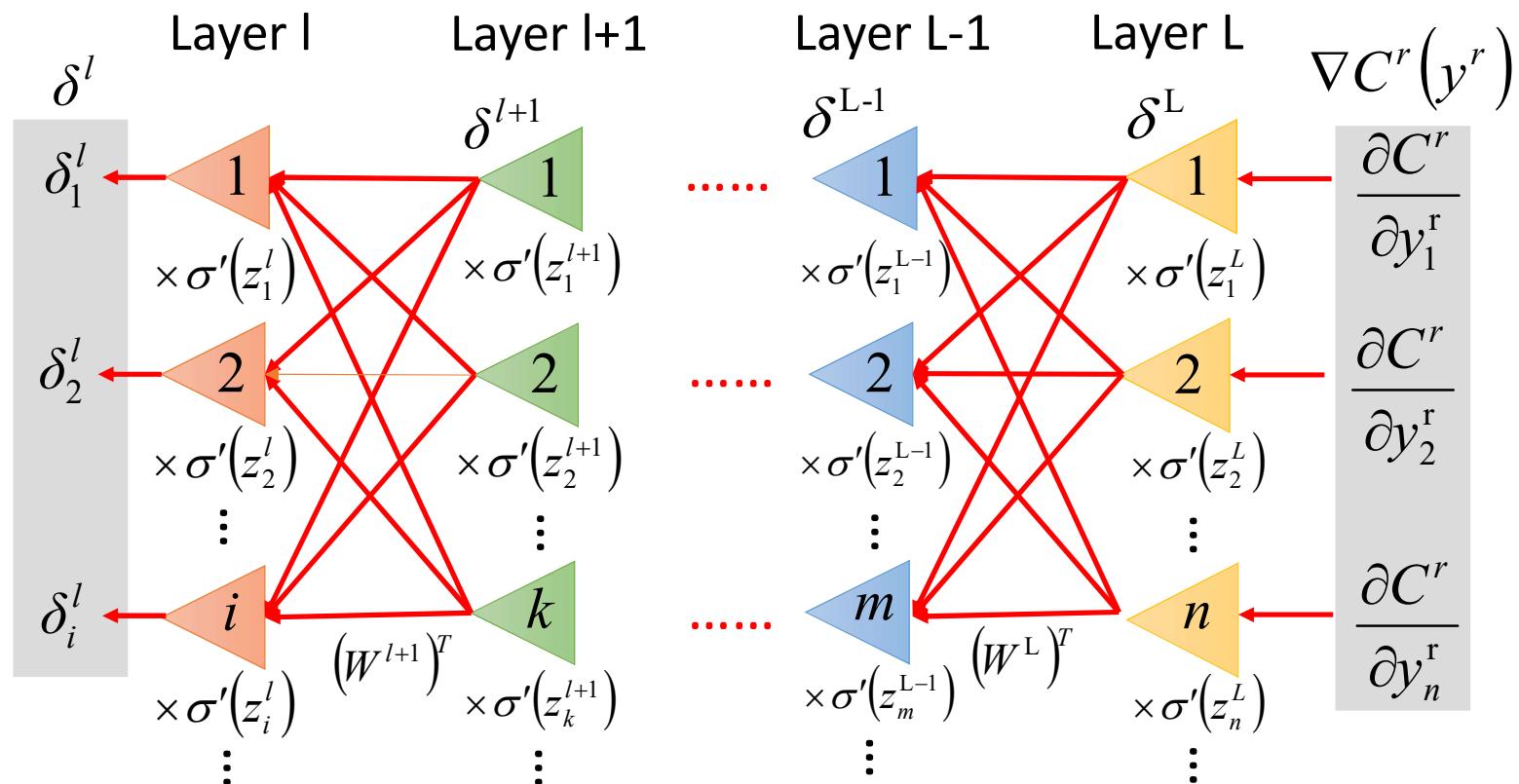
$$\frac{\partial C^r}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C^r}{\partial z_i^l}} \downarrow \delta_i^l$$

1. How to compute δ^L

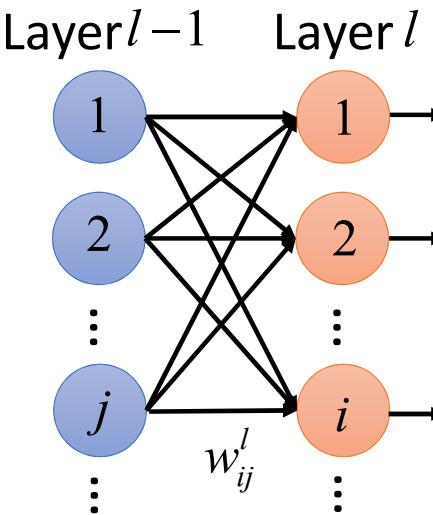
$$\rightarrow \delta^L = \sigma'(z^L) \bullet \nabla C^r(y^r)$$

2. The relation of δ^l and δ^{l+1}

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



Concluding Remarks



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

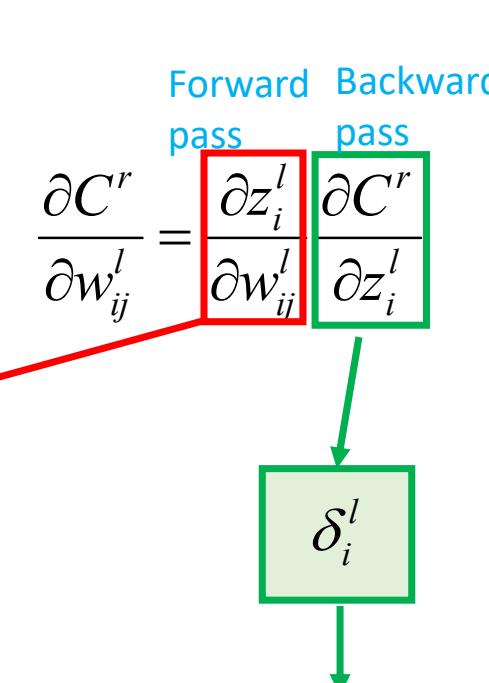
Forward Pass

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

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

$$\dots$$

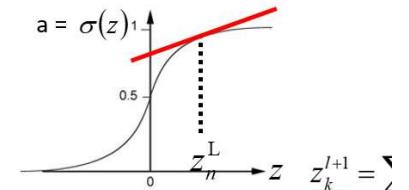
$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

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


$$\begin{aligned} \delta_n^L &= \frac{\partial C^r}{\partial z_n^L} \\ &= \frac{\partial y_n^r}{\partial z_n^L} \frac{\partial C^r}{\partial y_n^r} \\ &\downarrow \\ &\sigma'(z_n^L) \end{aligned}$$

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

Depending on the definition of cost function



$$z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

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

Backward Pass

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

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

$$\dots$$

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

$$\dots$$

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

$\sigma'(z_i^l)$

$$z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

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

Define a function set: $f(x; \theta)$

x : input

θ : network parameter

Define the total loss function L

That's it!

Design topology/network:

Having parameters (as variable)
without values

Step 1:
Network
Structure

How to I/O data? Target/Data:

Given input data and be recognized
/ classified output data. Exp. Mini
batch

Step 2:
Learning
Target

$o/p \Leftrightarrow$ Ground truth (label) y'

How to learn? Learning parameters

(or para values) of topology using
gradient descent to find global min
of MSE

Step 3:
Learn!



The network parameters
 θ^* minimizing total loss

If the input/output of the machine in your task
are both vectors, you know how to deal with it.

Machine: Network as nonlinear discrimination
function or as learnable kernel (SVM) $\phi()$

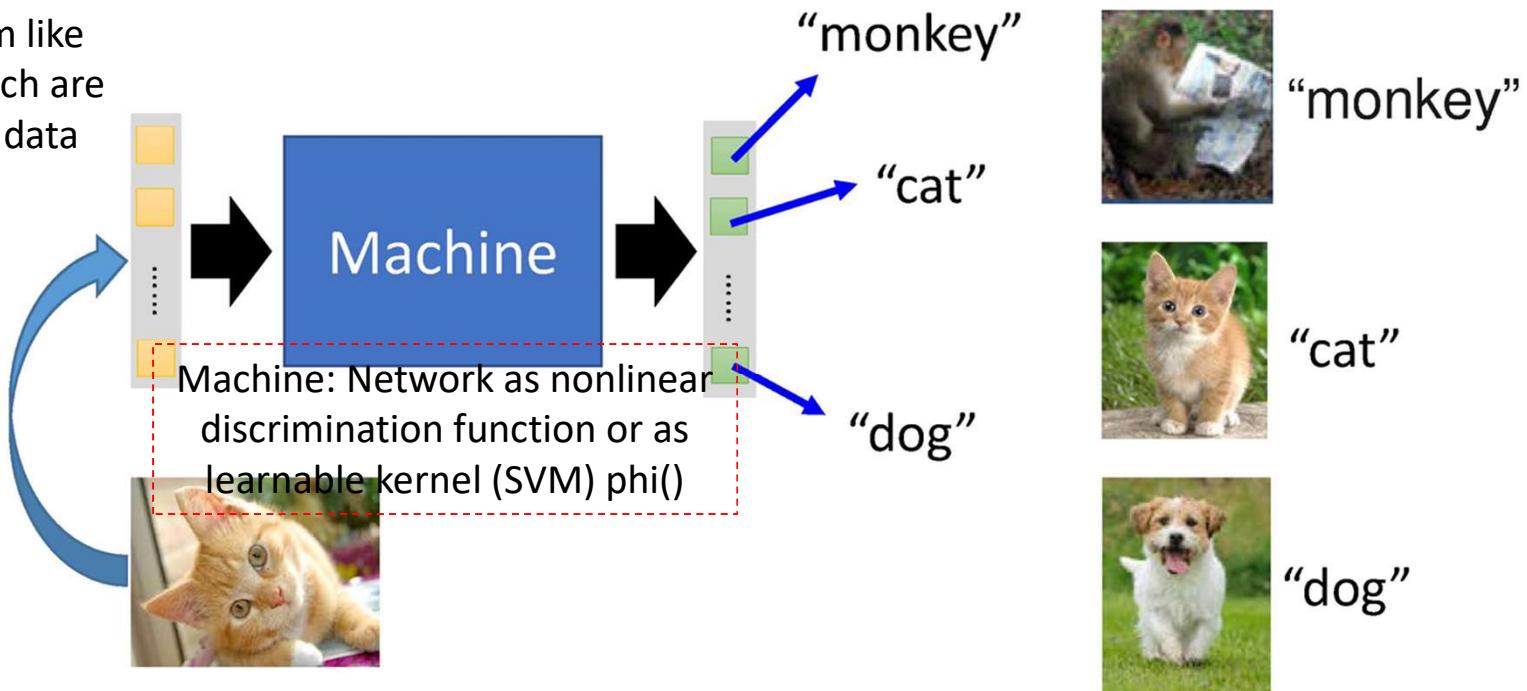
Only change the
learning target

JJ: Learning, NN is one by one input image, but the parameter can estimate either 1) one time by one image input 2) one time by one batch, or 3) wait until input all then estimate once.

For example, you can do

- Image Recognition

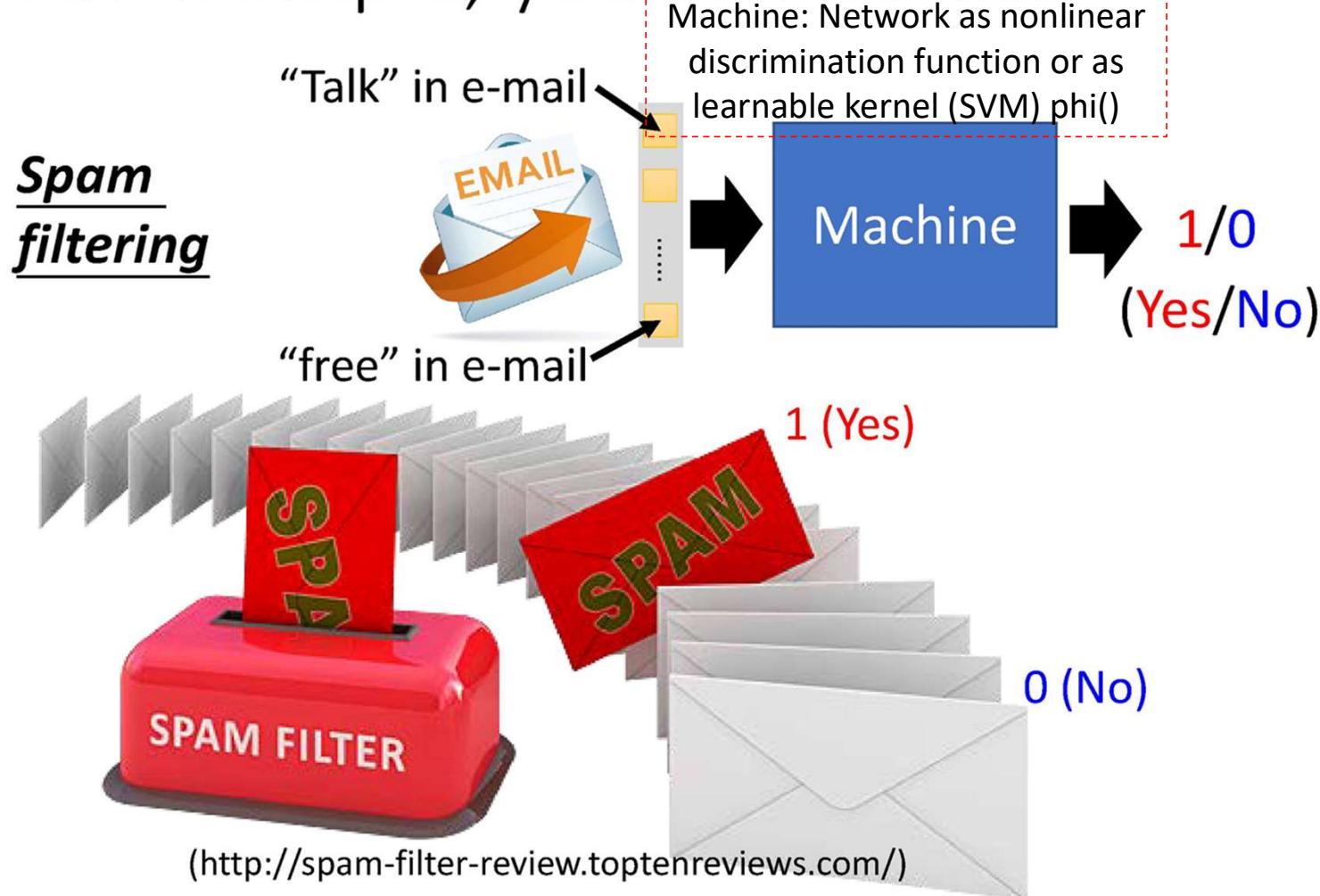
This is different form like PCA, LDA, SVM, which are input by all training data



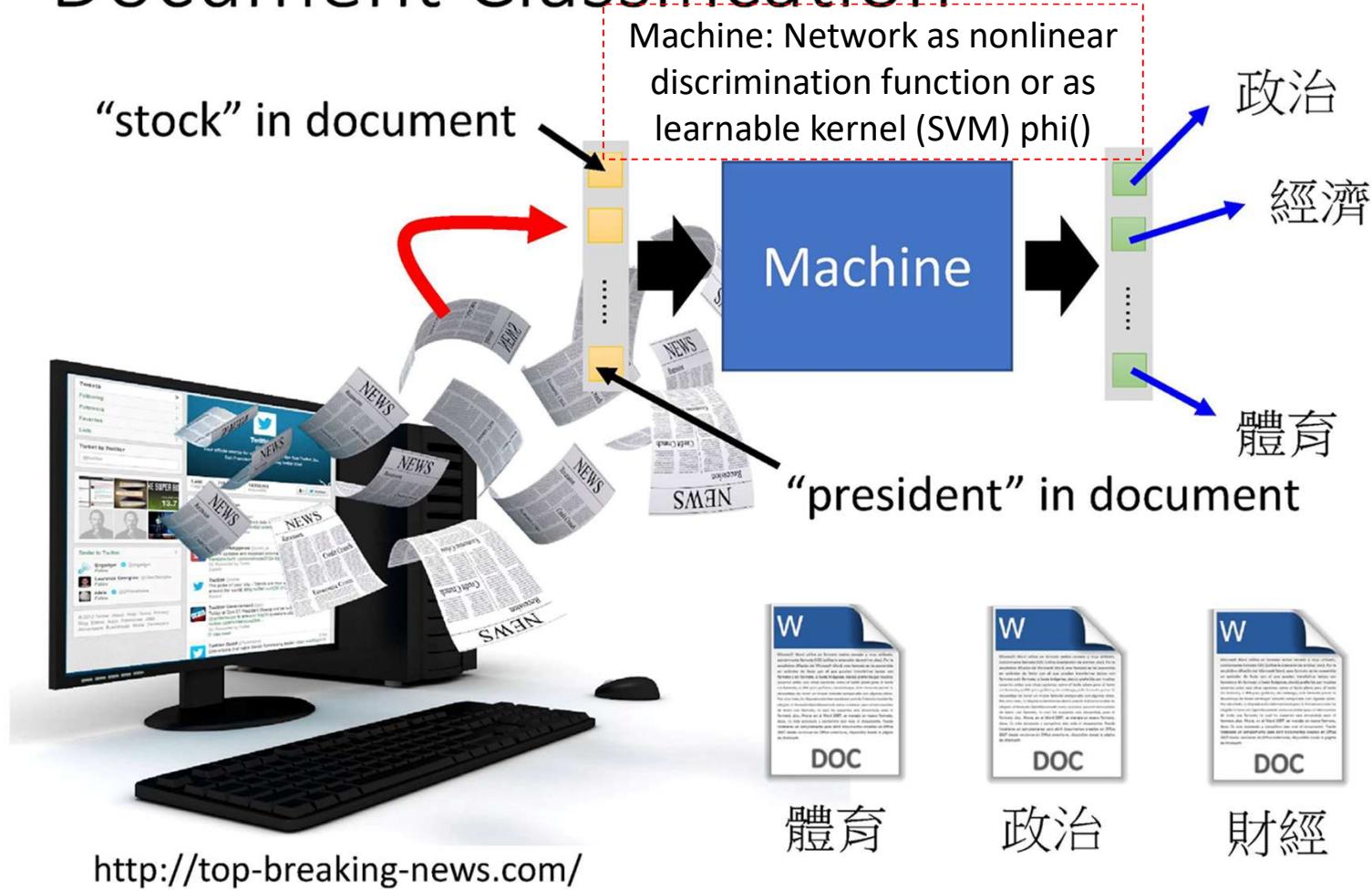
Application:

1. Image Recognition
2. Spam Filtering
3. Documentation Classification

For example, you can do



Document Classification



Outline of Lecture 1

I.1

Three Steps for Deep Learning

I.2

Why Deep Learning?

- 1) Fat+Short Vs. Thin+Tall
- 2) Multi-Task:
 - (1) How to reduce para. in order to use less data
 - (2) Modularization (logic circuit) – Combine network / Circuit
- 3) End to End Learning

I.3

“Hello world” for deep learning

Coding
Example

1) NN Vs. Deep NN:
In the beginning of deep learning, compare regular NN with Deep, deep works better simply because it uses more parameters.

Deeper is Better?

2) Deep NN Vs. Deeper NN:

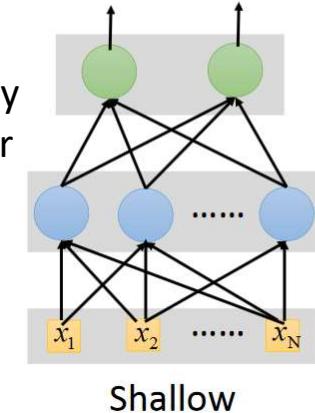
Later, deeper NN using 3x3 filter has less para. than deep NN using 7x7 filter.

Exp. For para numbers:
AlexNet (5+3) >
VGG16 (13+3) >
ResNet50 or 101 >
DenseNet

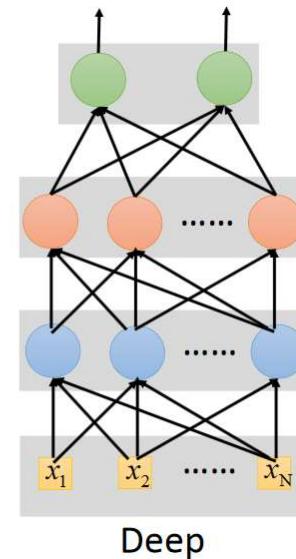
Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1
layer neuron	

Deep works better simply because it uses more parameters.

Shallow neural network has only one hidden layer



Shallow



Deep

Not surprised, more parameters, better performance

Deep neural network (DNN) has multiple hidden layers. It has more network parameter than shallow neural network. Suppose we have enough training data, the result of DNN will get better performance.

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

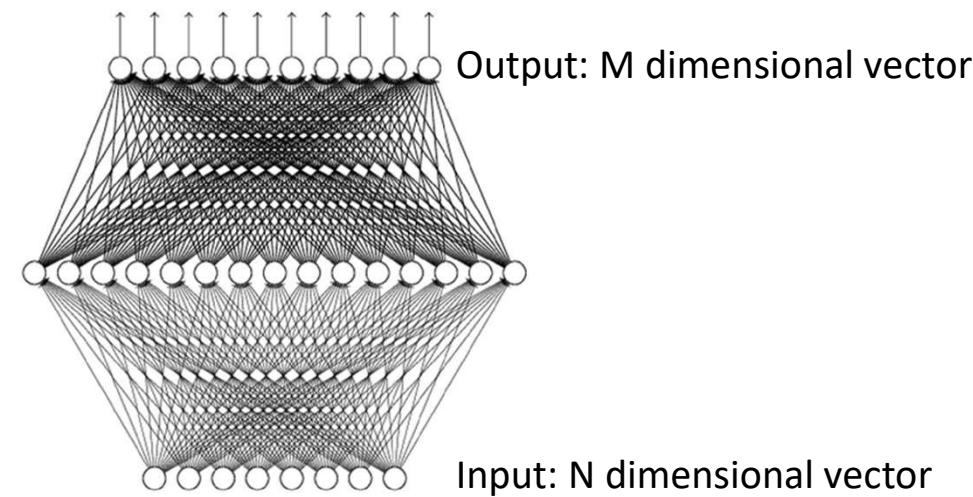
Universality Theorem

Any continuous function f

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

Can be realized by a network
with one hidden layer

(given **enough** hidden
neurons)

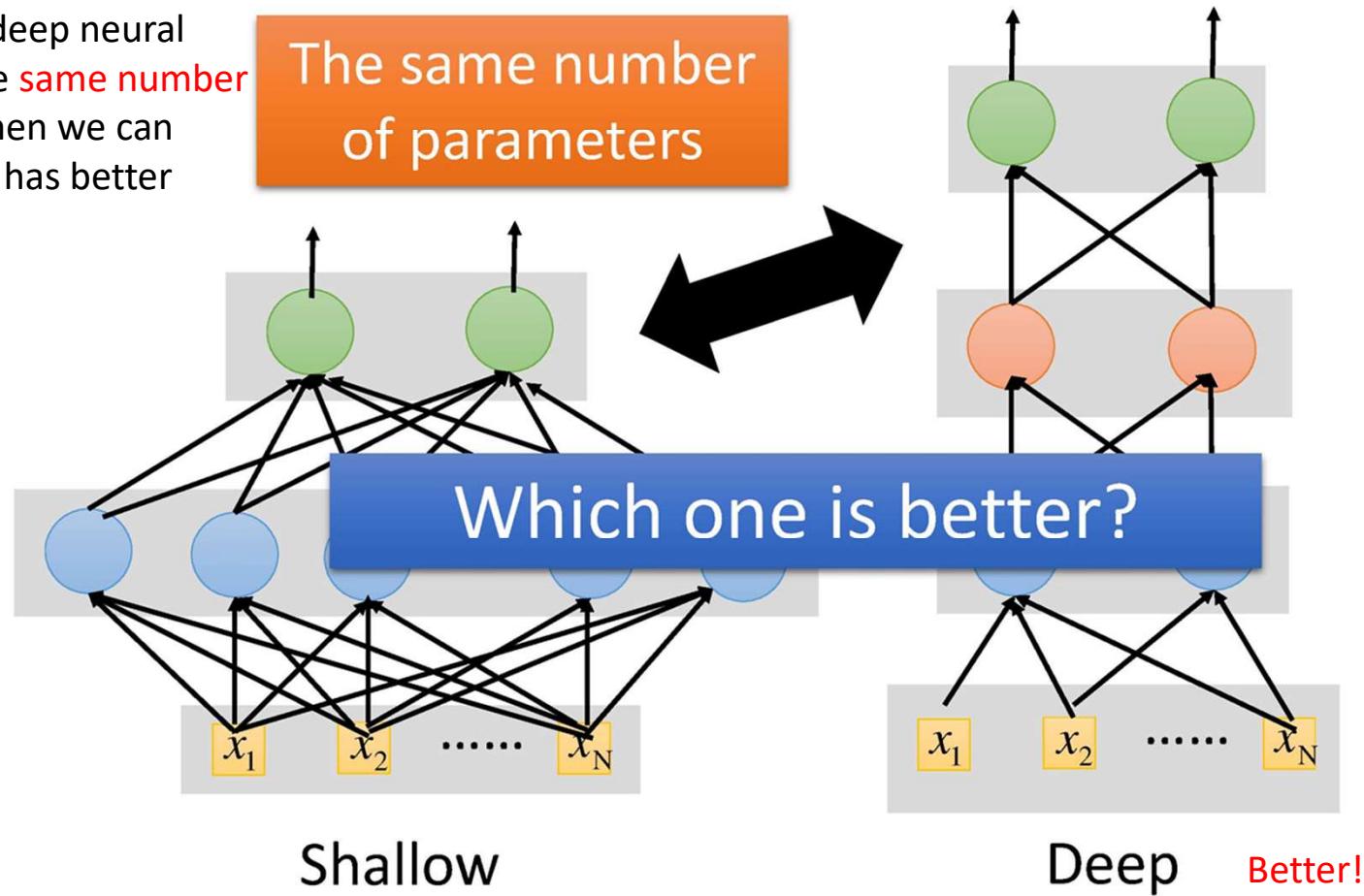


Reference for the reason:
<http://neuralnetworksanddeeplearning.com/chap4.html>

Why “Deep” neural network not “Fat” neural network?

Fat + Short v.s. Thin + Tall

Let shallow and deep neural network have the **same number of parameters**, then we can check which one has better performance



Fat + Short v.s. Thin + Tall

AdaBoost: Cascade

Multiple Layer		1 Hidden Layer	
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

Multiple layer and 1 hidden layer in the same row has the same number of the parameters.

Why?

1 X 16k is better than 1 X 2k, but not better than 2 X 2k.

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

Modularization

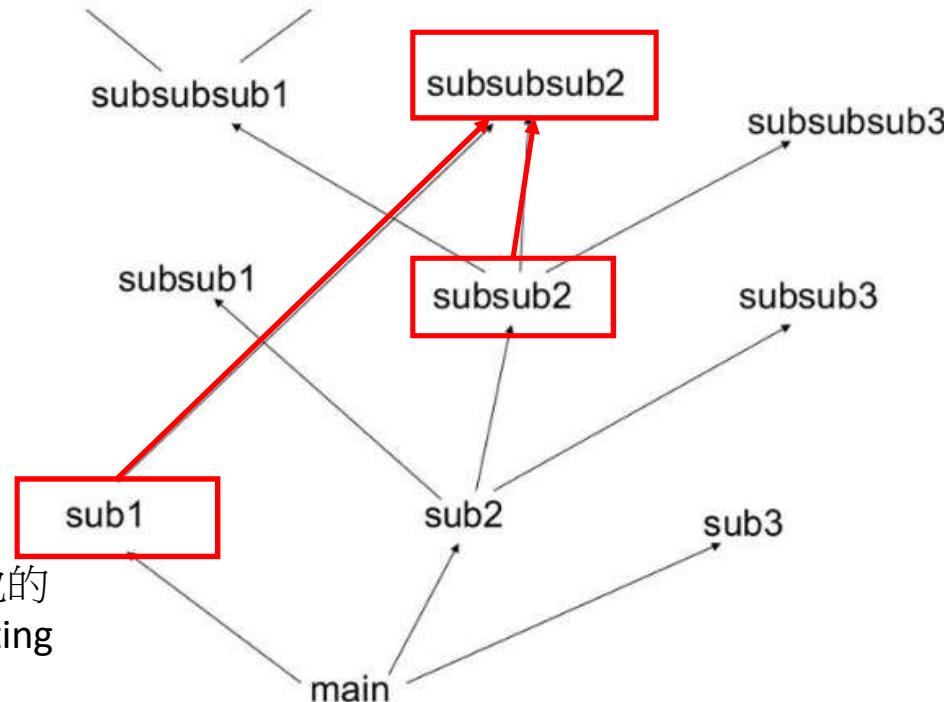
- Deep → Modularization

Don't put
everything in your
main function.

Advantage of modularization in programming:

1. 有一些函式是可以共用的
2. 減少程式的複雜度

舉例: 當subsubsub2函數是sorting，只要在其他的函式(sub1, subsub2)使用到，就不用在需要sorting的時候在自己的函式中重新implement一個。

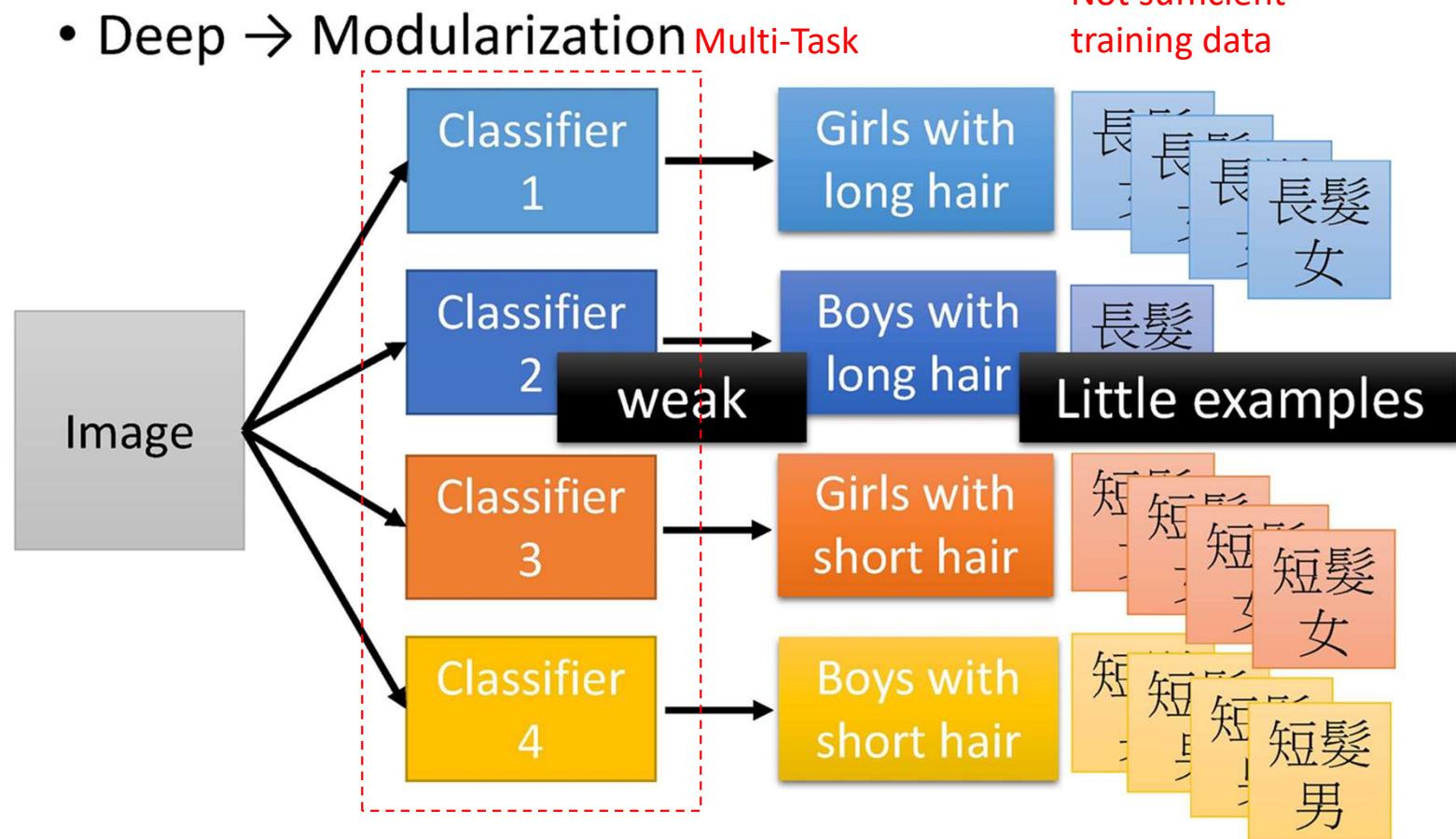


Modularization

Example = sample

Similar work as
Faster R-CNN for
multi-task:
ROI + classification

- Deep → Modularization Multi-Task

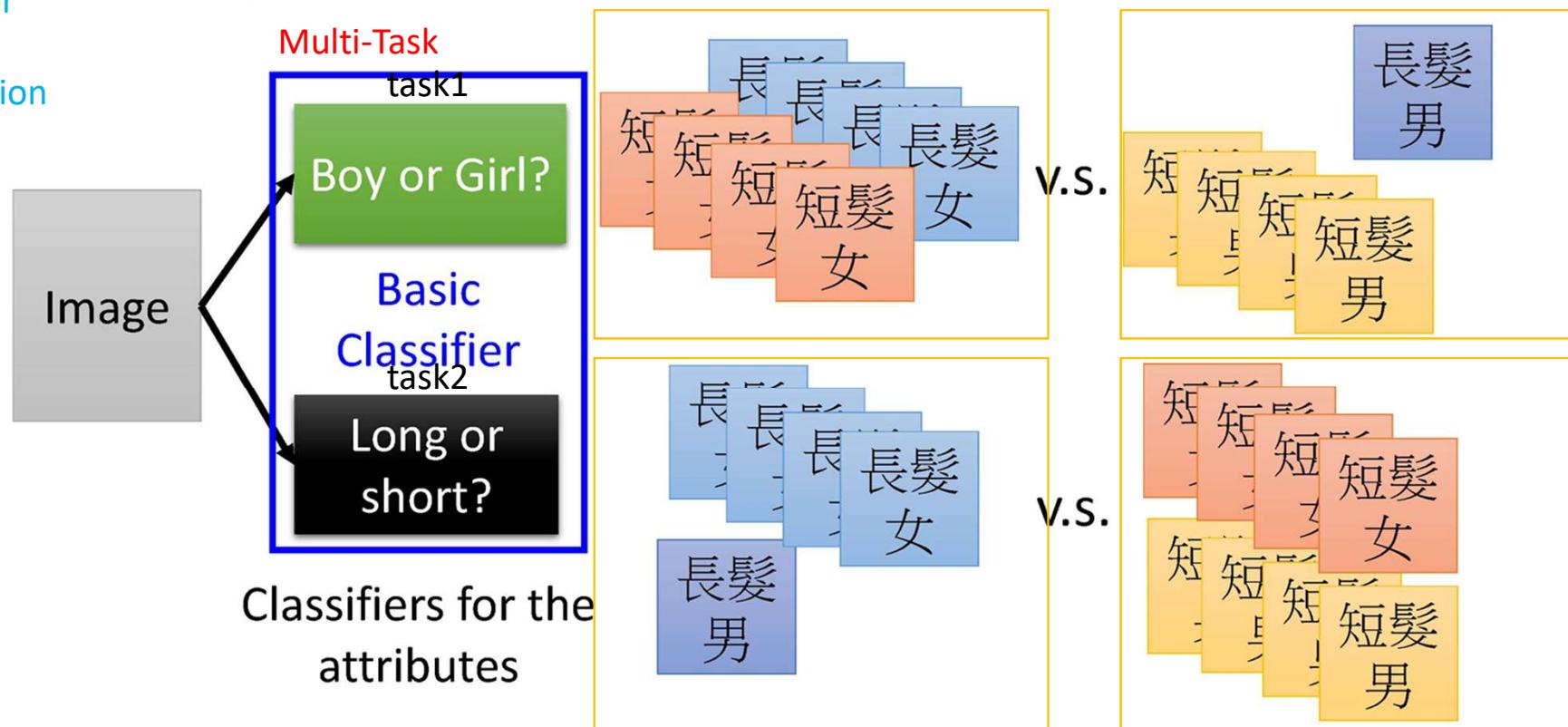


Modularization

Each basic classifier can have sufficient training examples.

Similar work as
Faster R-CNN for
multi-task:
ROI + classification

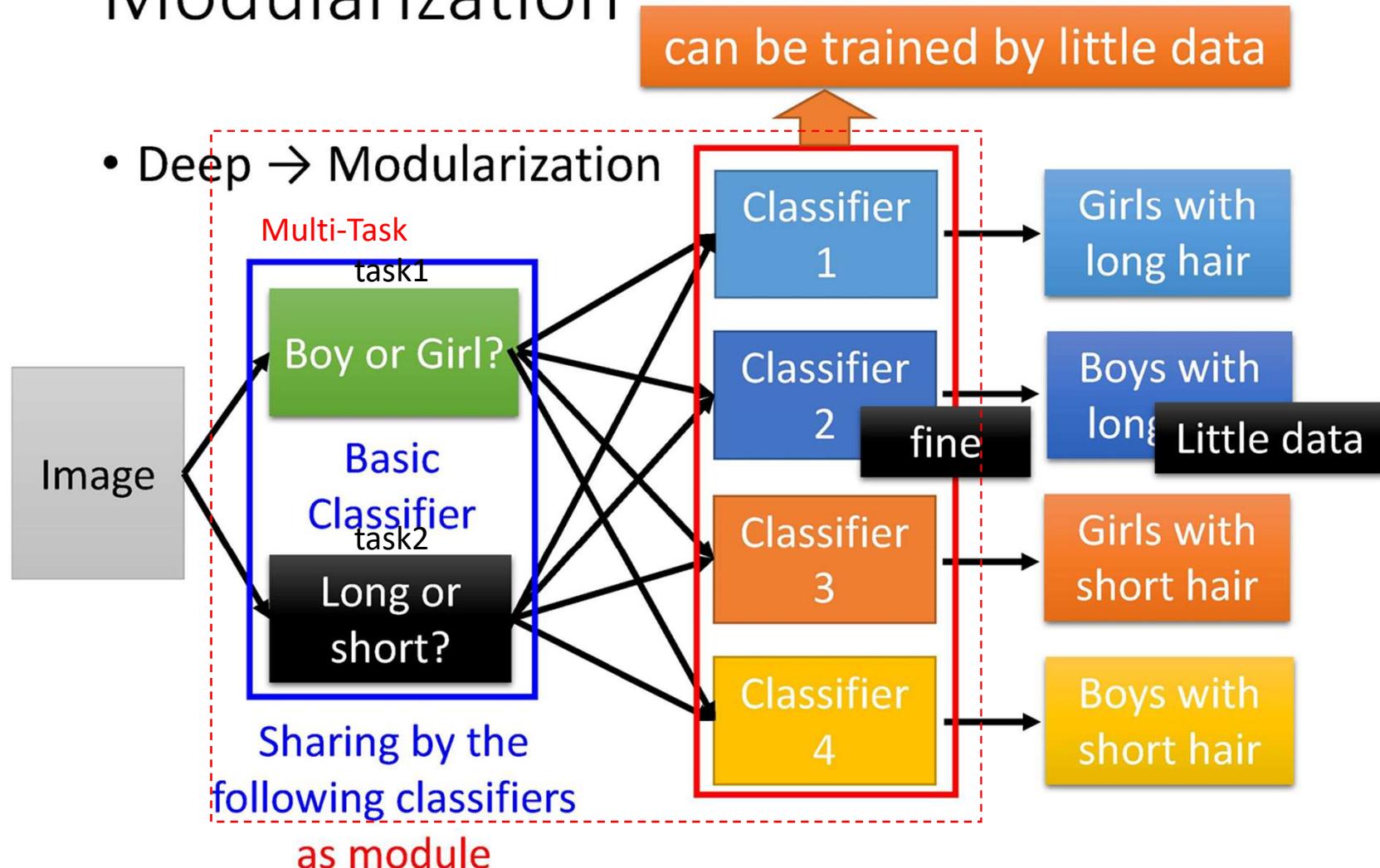
- Deep → Modularization



Modularization

J: Less parameters need less data

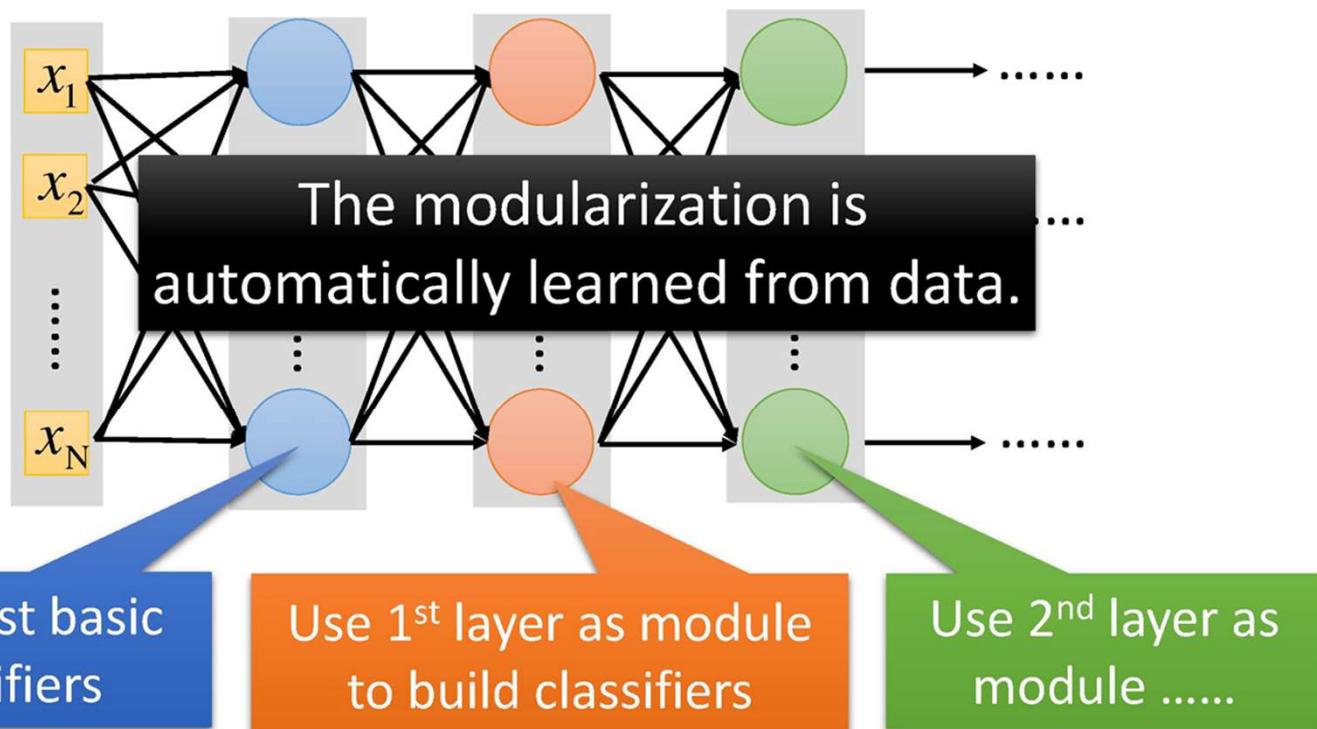
- Deep → Modularization



Modularization

Coarse to Fine

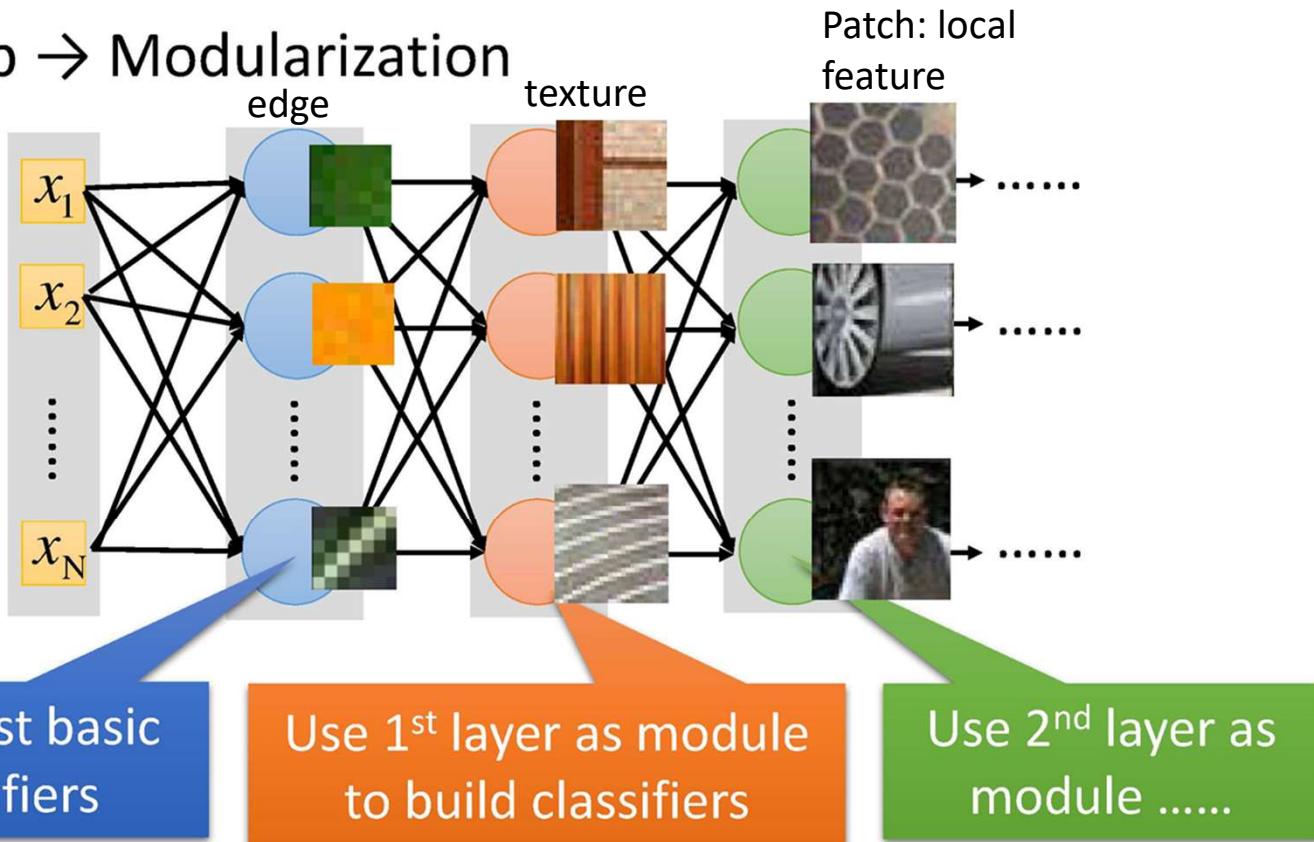
- Deep \rightarrow Modularization \rightarrow Less training data?



Modularization

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

- Deep → Modularization



The most basic classifiers

簡單的線條

Use 1st layer as module
to build classifiers

複雜的線條

Use 2nd layer as module

具體的物件

More Reasons

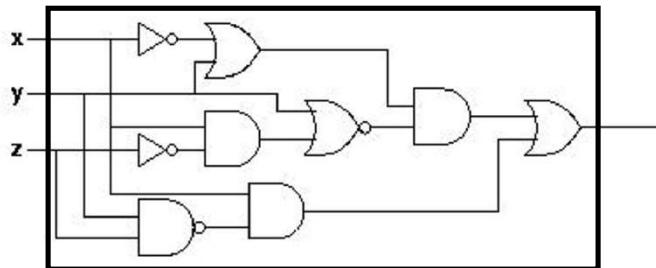
modularization

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



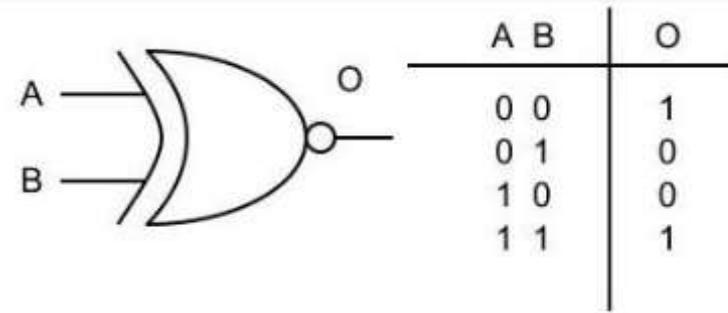
less data?

Fat Vs. Thin
and Tall

Shallow Vs.
Deep

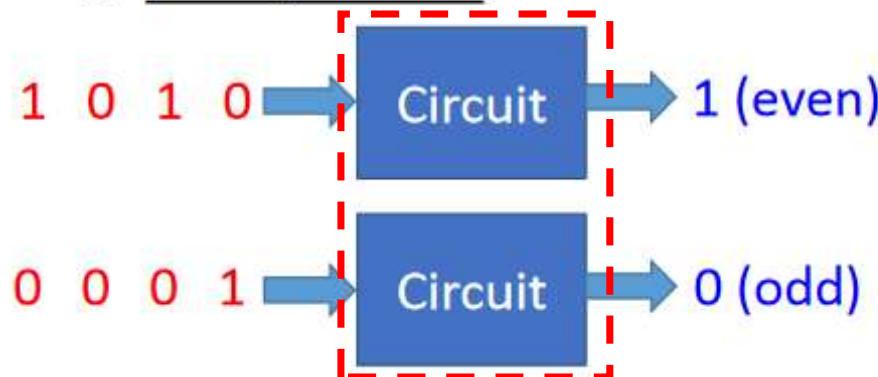
This page is for EE background.

Analogy



- E.g. parity check

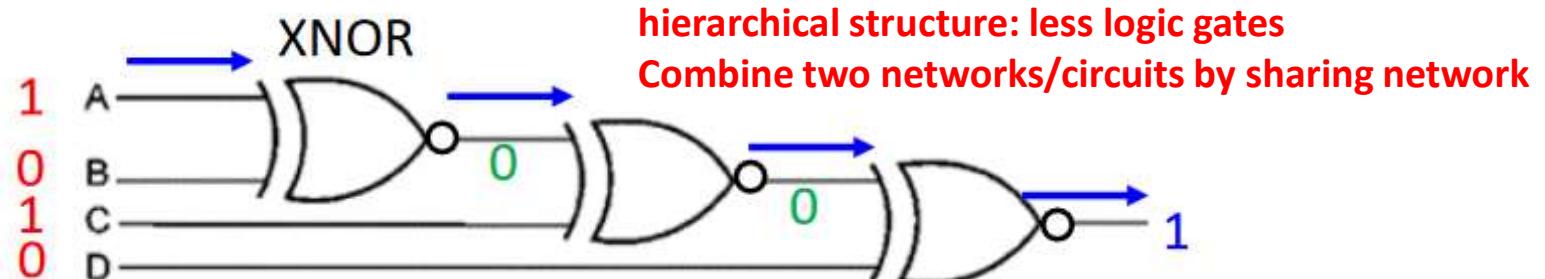
目的: 設計一個電路，使輸入出現1的數目是偶數的話，output是1，出現1的數目是奇數的話，output是0



For input sequence with d bits,

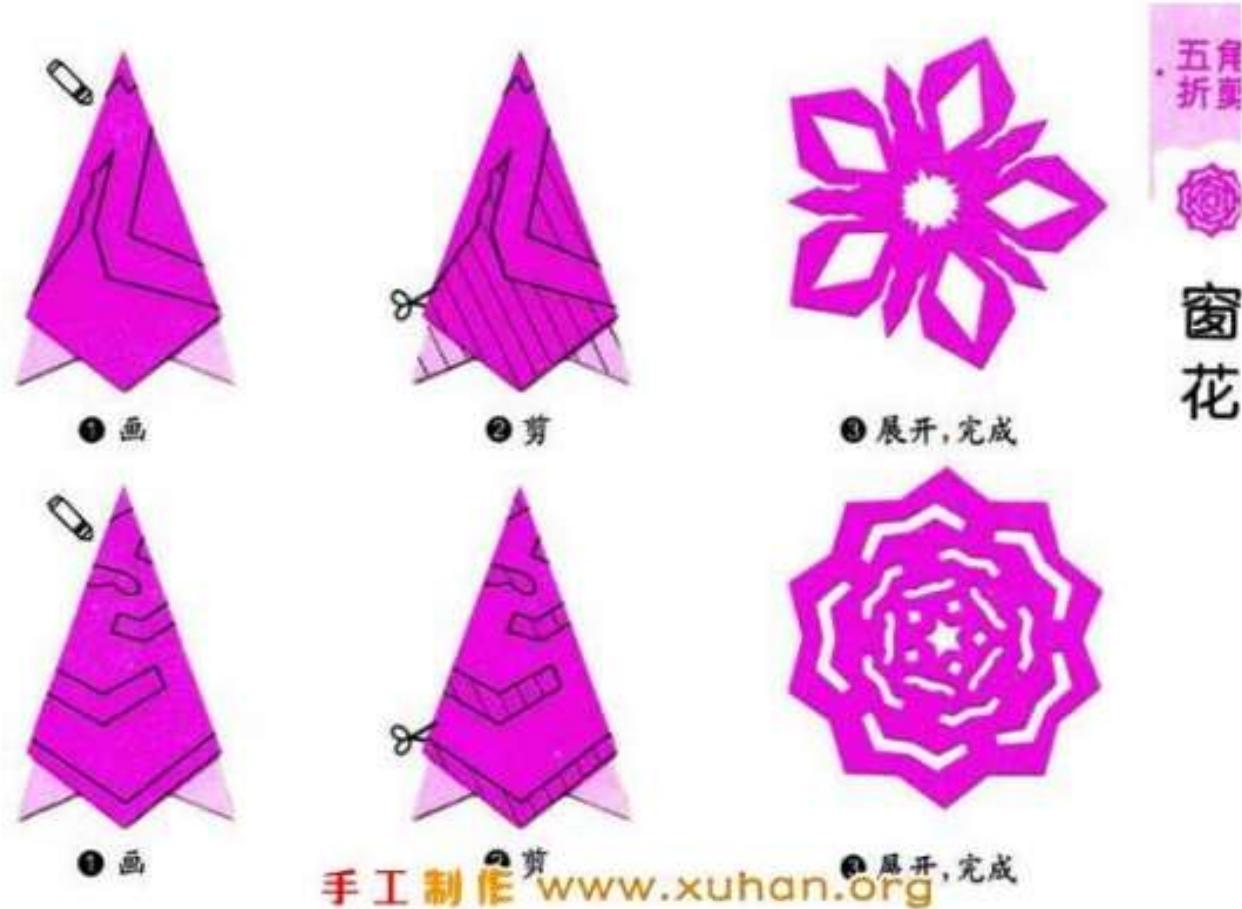
Two-layer circuit need $O(2^d)$ gates.

more logic gates

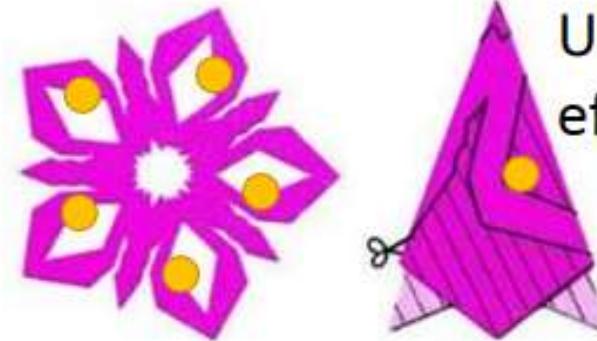


With multiple layers, we need only $O(d)$ gates.

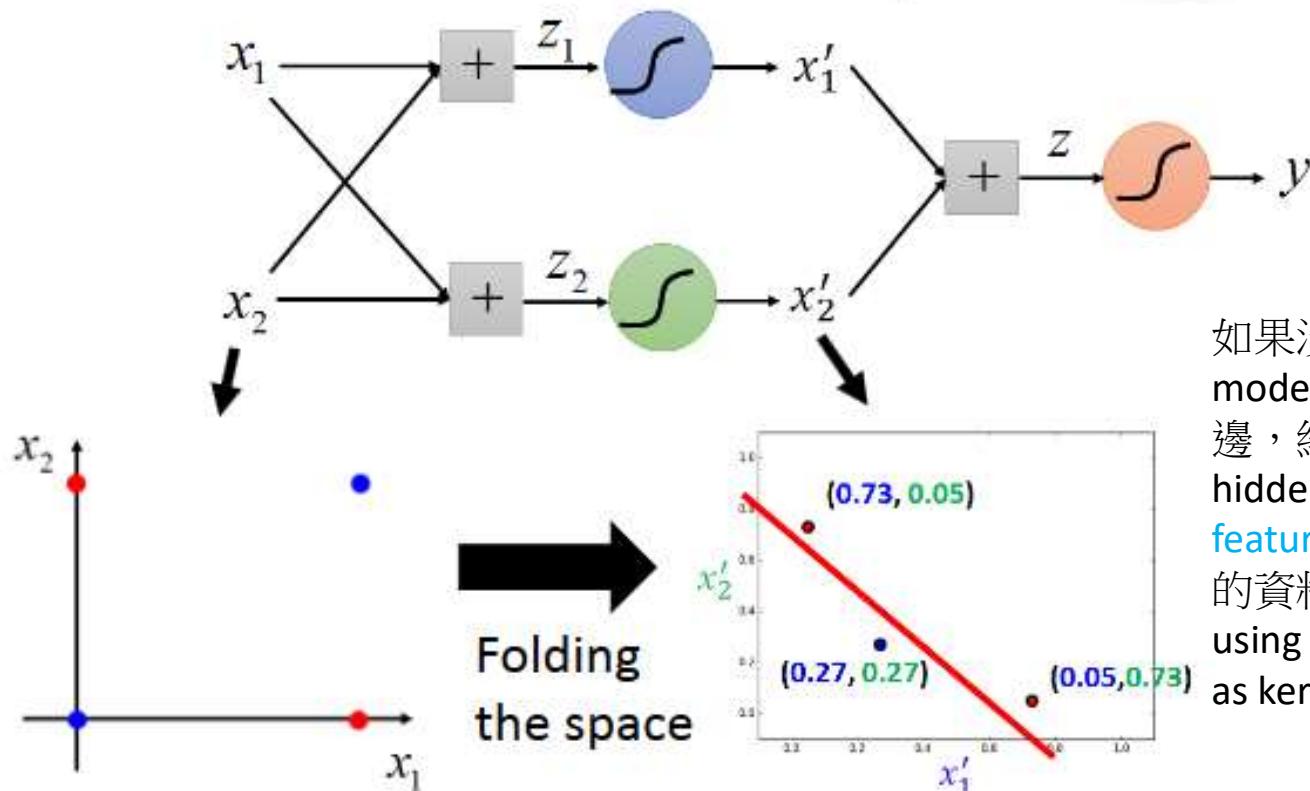
More Analogy



More Analogy



Use data effectively



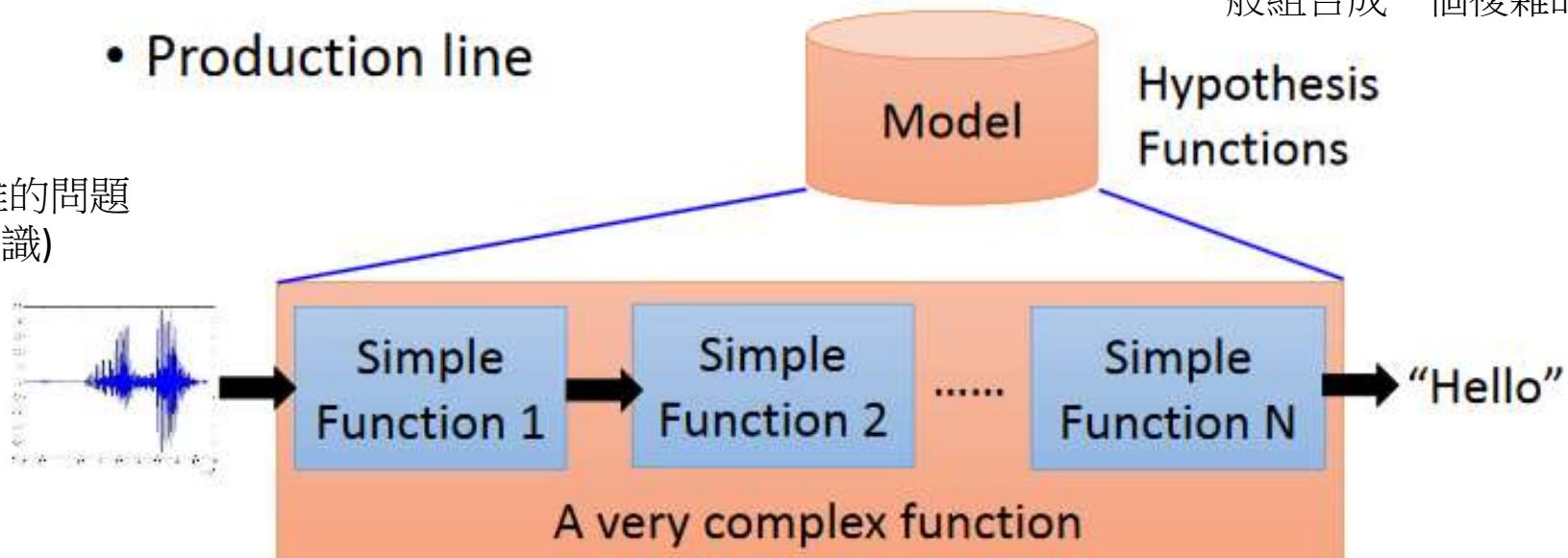
如果沒有hidden layer，linear的model沒有辦法把藍色分在一邊，紅色分在一邊，當加入了hidden layer相當於做了一個feature transformation，將原先的資料project到另一個space by using non-linear transformation as kernel function of SVM。

Another reason to use deep learning

End-to-end Learning

- Production line

欲解決複雜的問題
(例: 語音辨識)



End-to-end training:

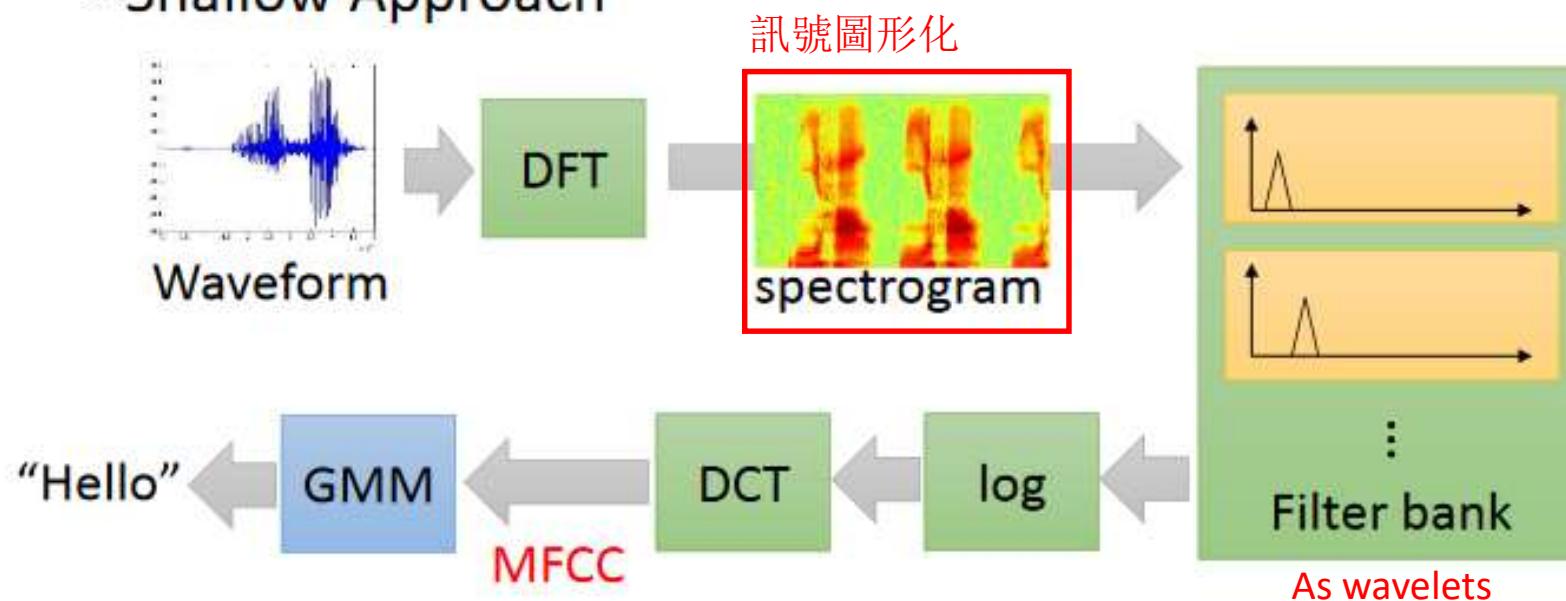
只給input與output，不決定中
間的function如何分工

What each function should do is learned automatically

➤ End-to-End for visual-guided robot arm control

End-to-end Learning - Speech Recognition

- Shallow Approach



Each box is a simple function in the production line:



:hand-crafted



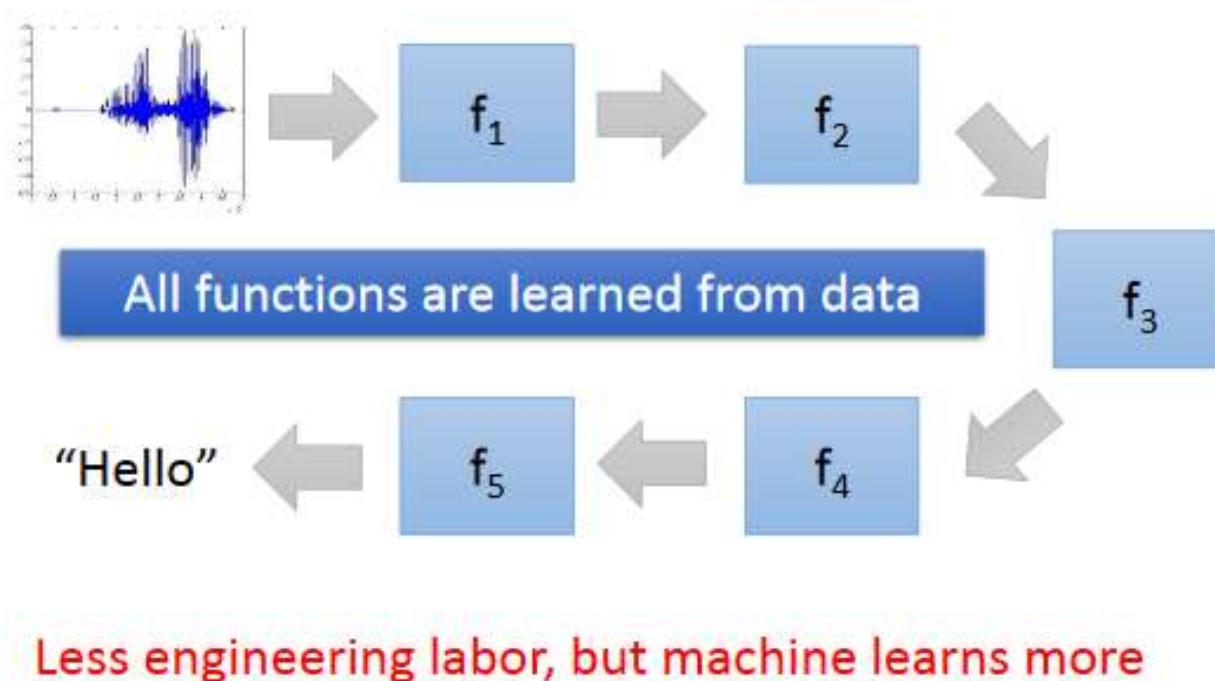
:learned from data

J: End-to-End, users just care of input and output, deep learning will take care of all processing for you.

J: My experience is we still need to design the number of output for each layers, especially for the last few layers.

End-to-end Learning - Speech Recognition

- Deep Learning

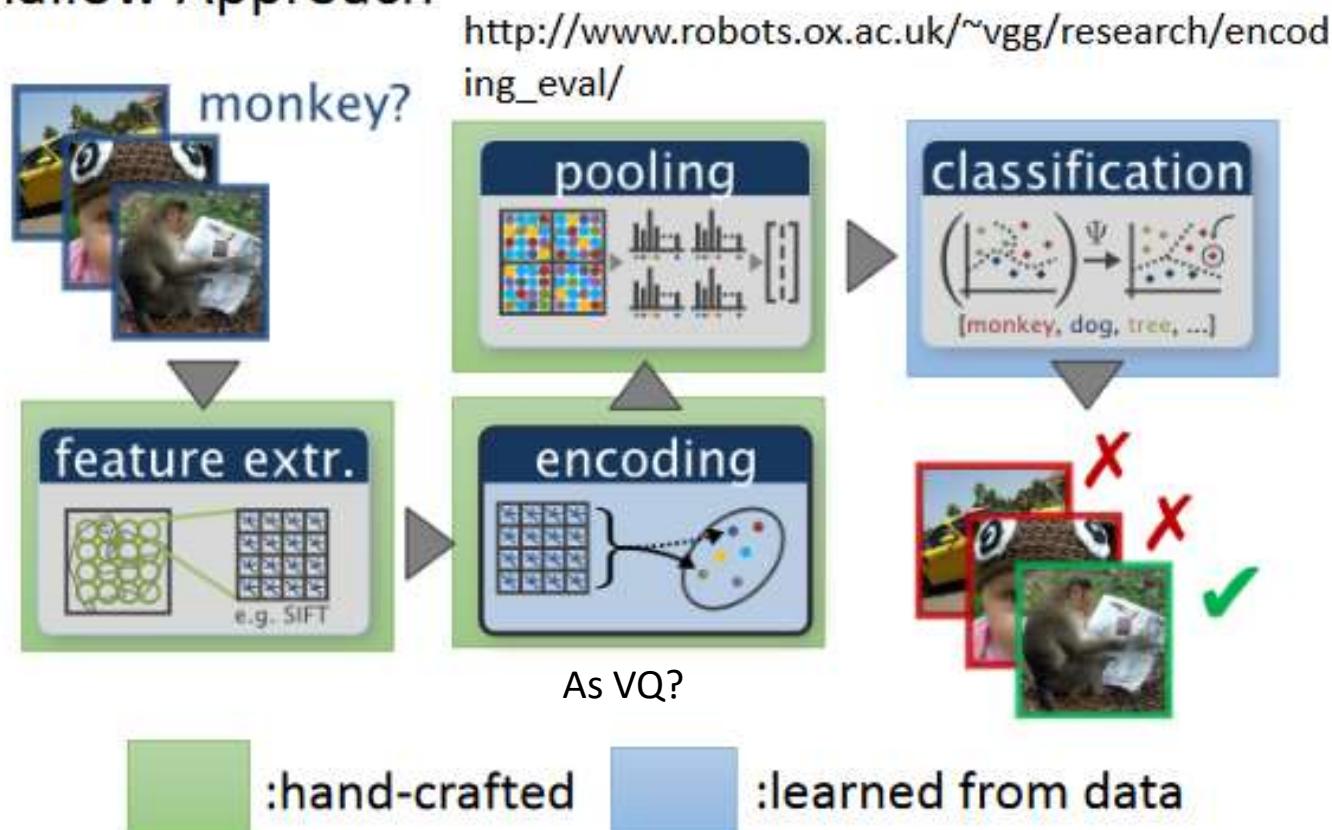


Q: 以聲音訊號為例，有學者提出能不能透過 deep learning，即只有input(聲音訊號)與 output(對應的文字)，完全不對輸入的聲音訊號 (time domain)做feature transform (signal processing)，是否能夠不需要signal processing
A: 有學者提出結論，能夠training出一個model與有將input經過 feature transform的結果持平，但沒有更好。

End-to-end Learning

- Image Recognition

- Shallow Approach

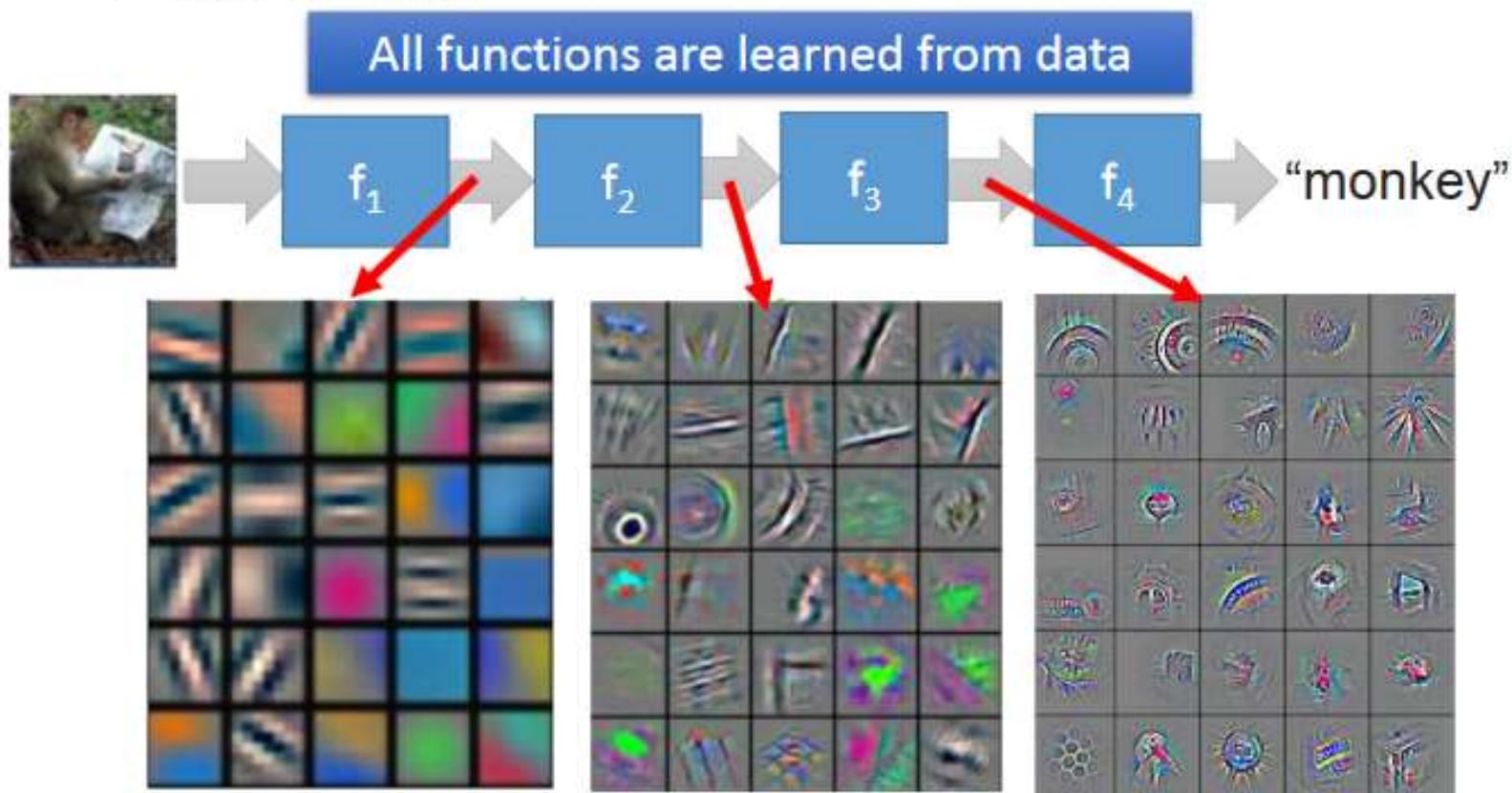


End-to-end Learning

- Image Recognition

- Deep Learning

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



More Reasons

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

The slide is divided into two equal-sized rectangular panels by a vertical line. The left panel contains a video thumbnail with the title 'Do deep nets really need to be deep?' in blue text, the names 'Rich Caruana' and 'Microsoft Research' below it, and 'Lei Jimmy Ba' and 'MSR Intern, University of Toronto' at the bottom. A small note at the bottom left reads: 'Thanks also to: Gregor Urban, Krzysztof Geras, Samira Kahou, Abdelrahman Mohamed, Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong'. The right panel is mostly white space, except for the word 'Yes!' in large red capital letters near the top center.

Do deep nets really
need to be deep?

Rich Caruana
Microsoft Research

Lei Jimmy Ba
MSR Intern, University of Toronto

Thanks also to: Gregor Urban, Krzysztof Geras, Samira Kahou, Abdelrahman Mohamed, Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong

Yes!

Thank You

Any Questions?

Outline of Lecture 1

I.1

Three Steps for Deep Learning

I.2

Why Deep Learning?

- 1) Fat+Short Vs. Thin+Tall
- 2) Multi-Task:
 - (1) How to reduce para. in order to use less data
 - (2) Modularization (logic circuit) – Combine network / Circuit
- 3) End to End Learning

I.3

“Hello world” for deep learning

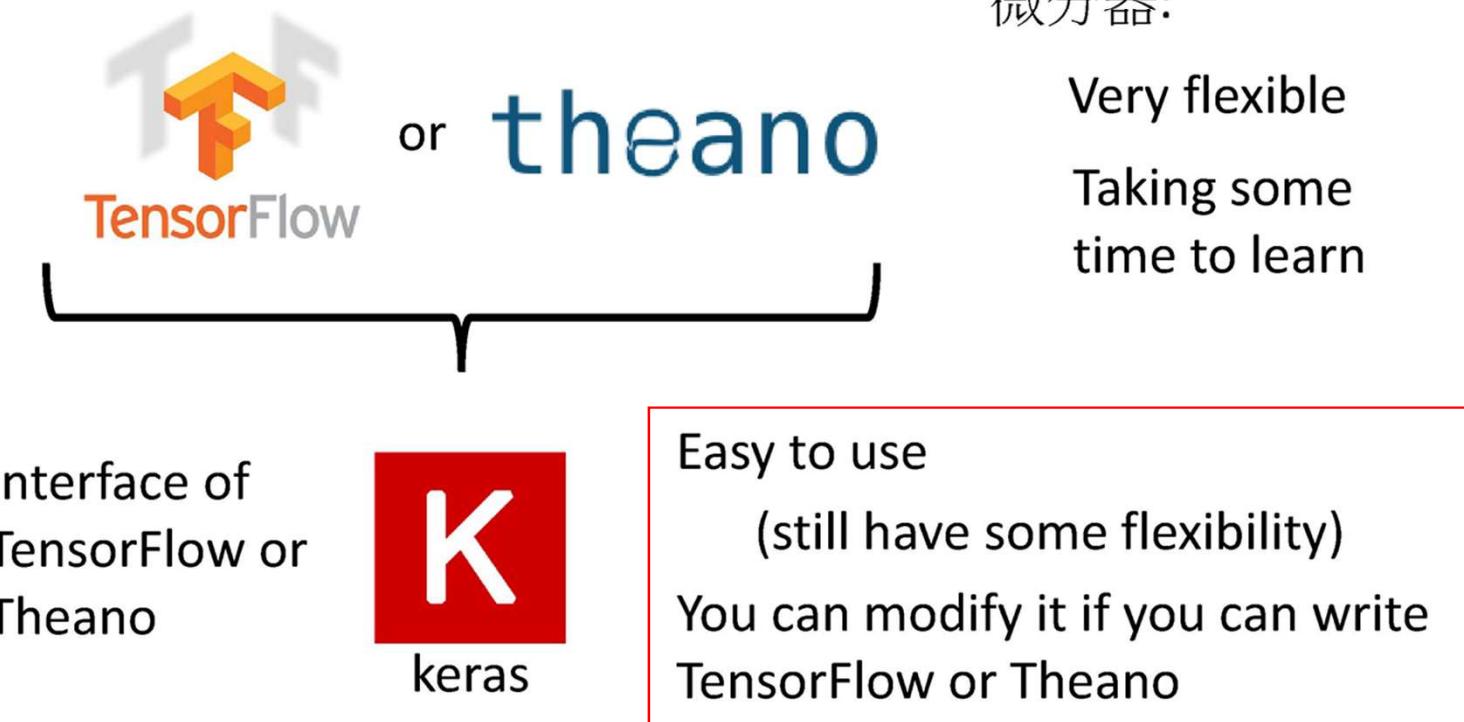
Coding
Example

Keras

If you want to learn theano:

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/RNN%20training%20\(v6\).ecm.mp4/index.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/RNN%20training%20(v6).ecm.mp4/index.html)



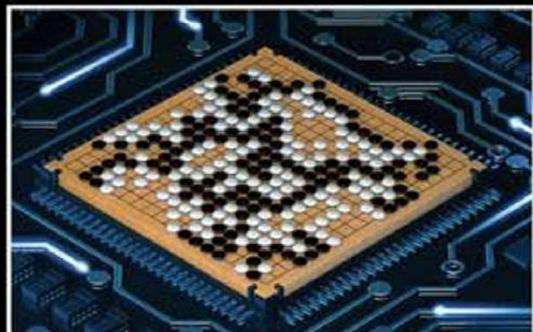
Keras

- François Chollet is the author of Keras.
 - He currently works for Google as a deep learning engineer and researcher.
- Keras means *horn* in Greek
- Documentation: <http://keras.io/>
- Example:
<https://github.com/fchollet/keras/tree/master/examples>

感謝 沈昇勳 同學提供圖檔

使用 Keras 心得

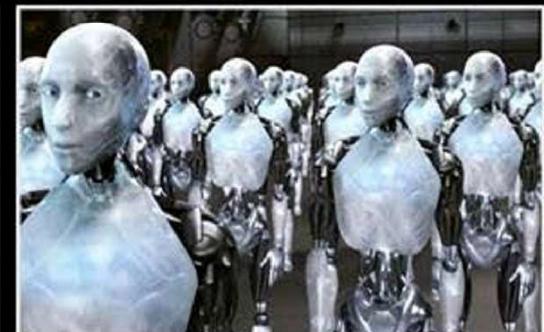
Deep Learning研究生



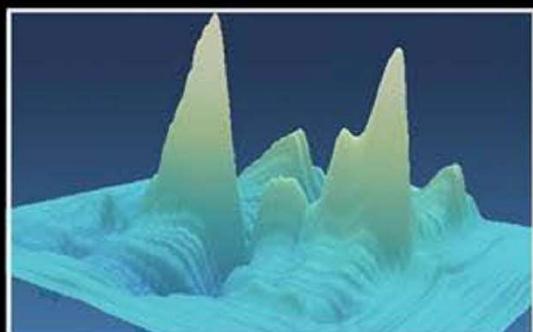
朋友覺得我在



我媽覺得我在



大眾覺得我在



指導教授覺得我在



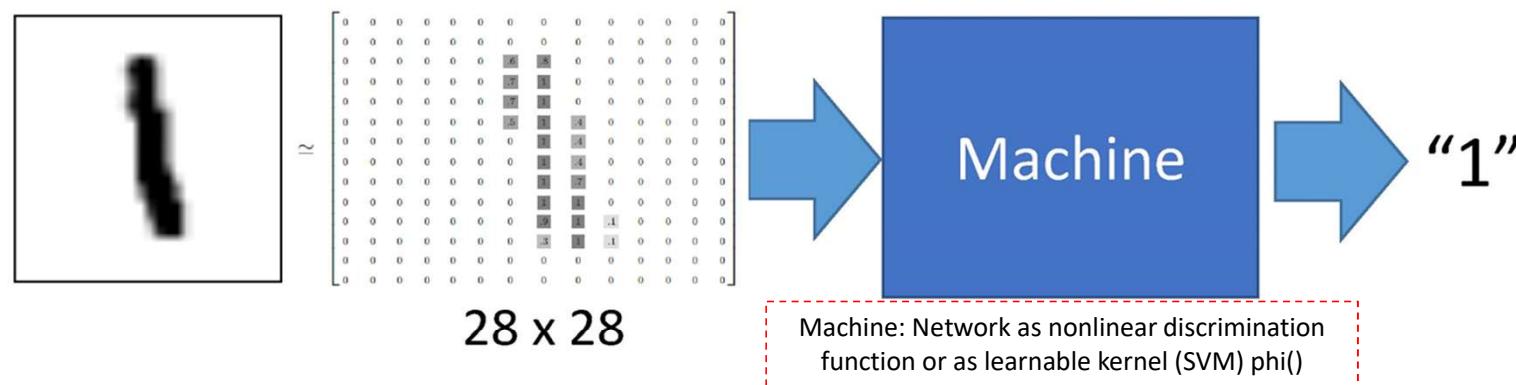
我以為我在



事實上我在

Example Application

- Handwriting Digit Recognition



MNIST Data: <http://yann.lecun.com/exdb/mnist/>

“Hello world” for deep learning

Keras provides data sets loading function: <http://keras.io/datasets/>

Training Process:

Keras

Design topology/network:
Having parameters (as variable)
without values

Step 1:
Network
Structure

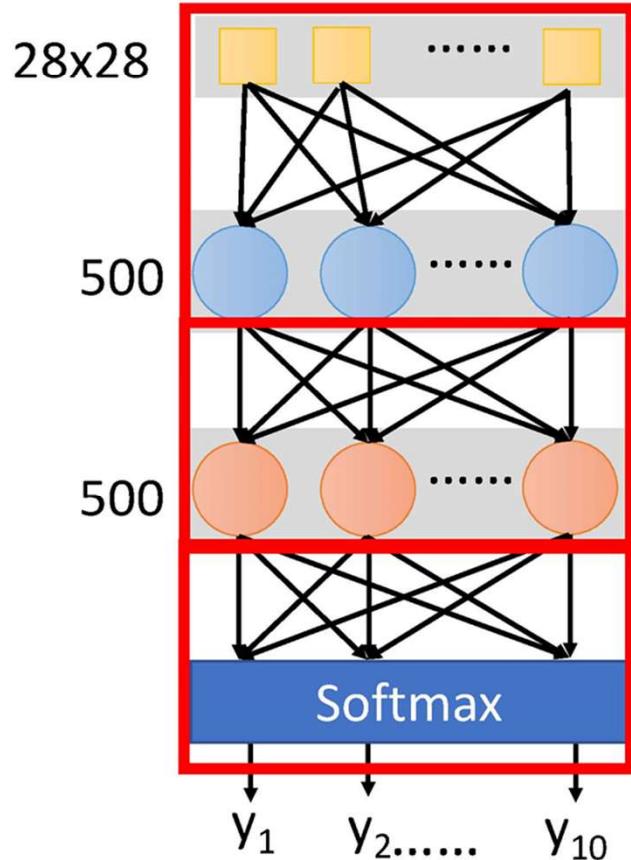
How to I/O data? Target/Data:
Given input data and be recognized
/ classified output data. Exp. Mini
batch

Step 2:
Learning
Target

o/p \Leftrightarrow Ground truth (label) y'

How to learn? Learning parameters
(or para values) of topology using
gradient descent to find global min
of MSE

Step 3:
Learn!



```
model = Sequential()
```

“Dense” represents a fully connected layer
model.add(Dense(input_dim=28*28, output_dim=500))
model.add(Activation('sigmoid'))

Activation function can also be chose as softplus,
softsign, relu, tanh, hard_sigmoid, linear

```
model.add( Dense( output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

Ten digits
model.add(Dense(output_dim=10))
model.add(Activation('softmax'))

Output layer can be seem as a multi-class
classifier, so we choose softmax as our activation

Training Process:

1) Batch = input all, update once

2) Mini – batch =

3) SGD belongs to stochastic = 隨機 = input 1 data, update once

10000 training data, based on stochastic, probably you just need 1000 data to train, then parameter reaches convergence.

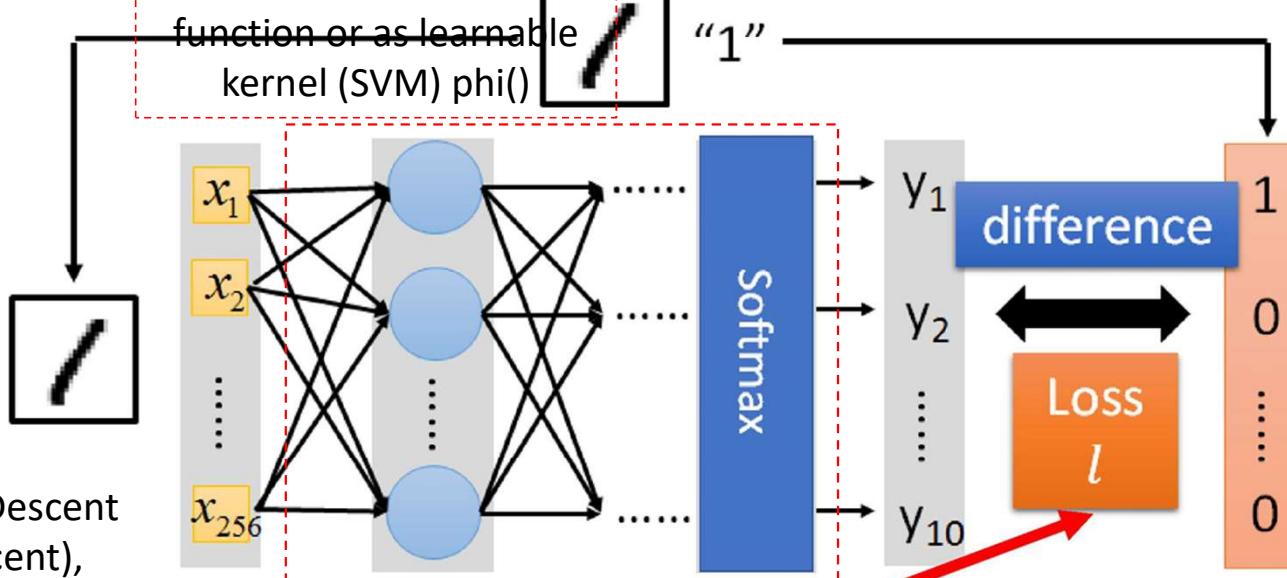
Keras

Step 1: Network Structure

Step 2: Learning Target

Step 3: Learn!

Machine: Network as nonlinear discrimination function or as learnable kernel (SVM) $\phi()$



SGD: Stochastic Gradient Descent
(incremental gradient descent),
Loss: Loss function

MSE: Mean squared error

lr: Learning rate

```
model.compile(loss='mse',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

Vs. Entropy

$$\frac{1}{10} \sum_{i=1}^{10} \|y_i - \hat{y}_i\|^2$$

$$-\sum_{i=1}^{10} (\hat{y}_i \log y_i)$$

$y^1=1$ Yes, $y^0=0$ No

Training Process:

1) Batch = input all, update once

2) Mini – batch =

3) SGD belongs to stochastic = 隨機 = input 1 data, update once

10000 training data, based on stochastic, probably you just need 1000 data to train, then parameter reaches convergence.

1. Introduction - 2) Gradient-Based Learning

Gradient-Based Learning algorithms can use one of two classes of methods to update the parameters. The first method, dubbed “Batch Gradient”, is the classical one: the gradients are accumulated over the entire training set, and the parameters are updated after the exact gradient has been so computed. In the second method, called “Stochastic Gradient”, a partial, or noisy, gradient is evaluated on the basis of one single training sample (or a small number of samples), and the parameters are updated using this approximate gradient. The training samples can be selected randomly or according to a properly randomized sequence. In the stochastic version, the gradient estimates are noisy, but the parameters are updated much more often than with the batch version. An empirical result of considerable practical importance is that on tasks with large, redundant data sets, the stochastic version is considerably faster than the batch version, sometimes by orders of magnitude [117]. Although the reasons for this are not totally understood theoretically, an intuitive explanation can be found in the following extreme example. Let us take an example where the training database is composed of two copies of the same subset. Then accumulating the gradient over the whole set would cause redundant computations to be performed. On the other hand, running Stochastic Gradient once on this training set would amount to performing two complete

As gradient decent:
Learning rate = Epsonlon,
K = kth iteration

Epsilon: weight for current k situation

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}$$

ϵ is a scalar constant. More sophisticated procedures use variable ϵ , or substitute it for a diagonal matrix, or substitute it for an estimate of the inverse Hessian matrix as in Newton or Quasi-Newton methods.

Batch gradient or
stochastic gradient

Stochastic gradient algorithm (on-line update):

$$W_k = W_{k-1} - \epsilon \frac{\partial E^{P_k}(W)}{\partial W}$$

It consists in updating the parameter vector using a noisy, or approximated, version of the average gradient. In the most common instance of it, W is updated on the basis of a single sample:

J: Pk for example, each pk=1000 samples for ith iteration, total P=10000 samples

Training Process:

??

Keras



Step 3.1: Configuration

```
model.compile(loss='mse',  
               optimizer=SGD(lr=0.1),  
               metrics=['accuracy'])
```

$$w \leftarrow w - \eta \partial L / \partial w$$

0.1
??

$$\frac{1}{10} \sum_{i=1}^{10} \|y_i - \hat{y}_i\|^2$$

$$-\sum_1^{10} (\hat{y}_i \log y_i)$$

$y^{\wedge}=1$ Yes, $y^{\wedge}=0$ No

Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training data
(Images)

Labels
(digits)

等一下再講 ??

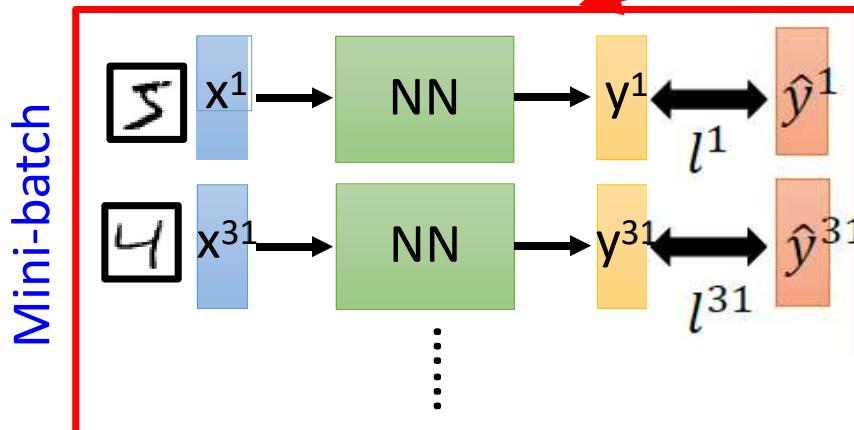
Stochastic gradient descent : Batch size = 1

Full batch gradient descent: Batch size = number of training data

Mini batch gradient descent: Batch size between stochastic and full batch

Mini-batch

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



Ex. All 500 examples → 5 mini-batch

Repeat 20 times = repeat 20 epochs

Repeat 20 times

For each epoch, entire samples are randomly shuffled to have new 5 mini-batches (Shuffle: a method to select the training examples for each epoch, detail will be talked in lecture 2)

- Pick the 1st batch
 $L' = l^1 + l^{31} + \dots$
Update parameters once
- Pick the 2nd batch
 $L'' = l^2 + l^{16} + \dots$
Update parameters once
- ⋮
- Until all mini-batches have been picked

one epoch

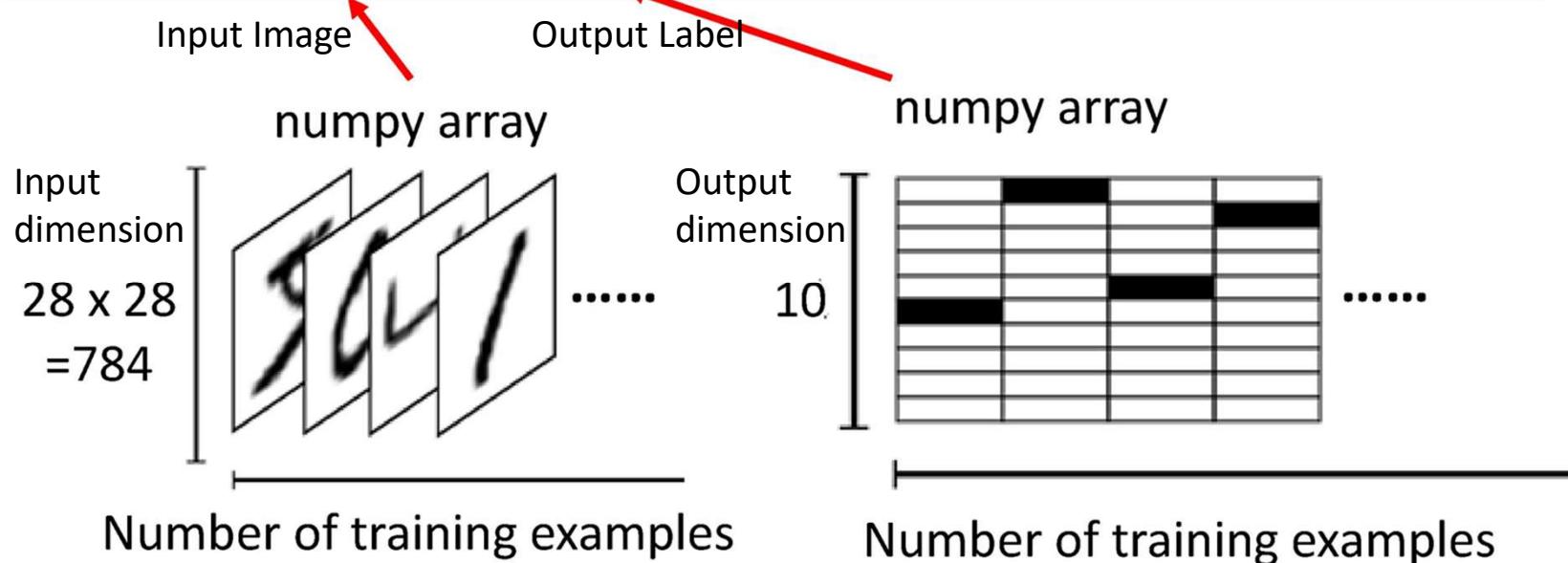
Training Process:

Keras



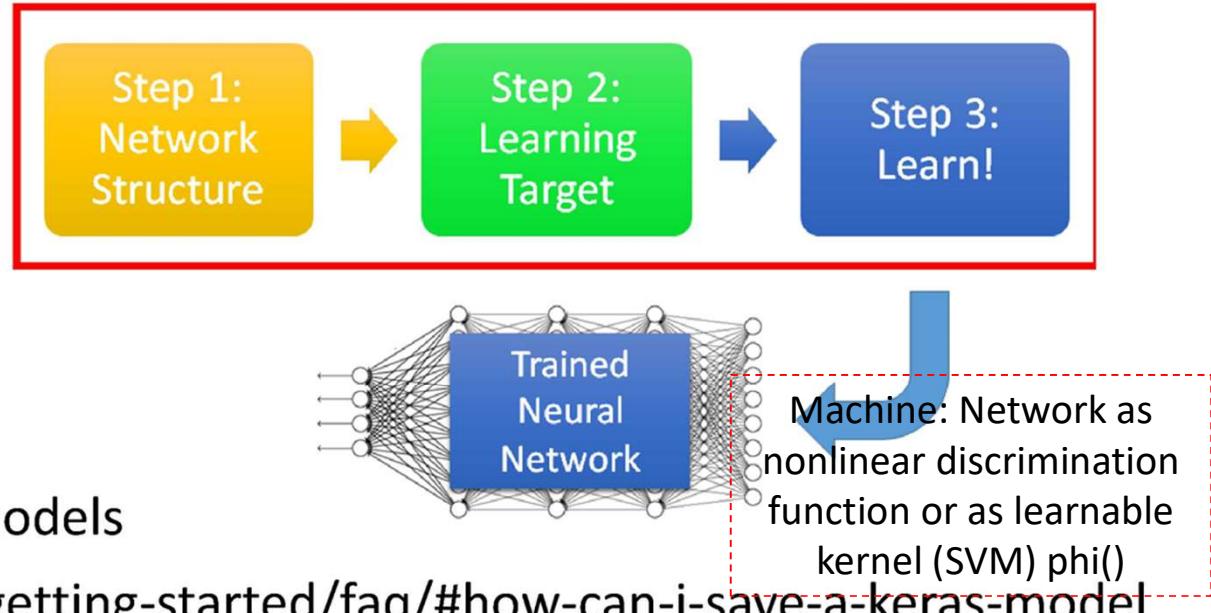
Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



Test Process: ??

Keras



How to use the neural network (testing):

```
score = model.evaluate(x_test, y_test)
case 1: print('Total loss on Testing Set:', score[0])
          print('Accuracy of Testing Set:', score[1])
```

```
case 2: result = model.predict(x_test)
```

Get your hand dirty

- Using GPU
 - THEANO_FLAGS=device=gpu0 python YourCode.py

Concluding Remarks of Lecture I

- Deep Learning is simple!
- Deep Learning is powerful!

但 Deep Learning 就像 雷神之槌



無法輕易被舉起來

<http://ent.ltn.com.tw/news/breakingnews/1144545>

Reference ??

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