Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

AI Training Course Series

Multilayer Perceptron (MLP)

Lecture 2-1

Architecture Development & Algorithm Refinement for Intelligent Computing

Student: Jye-En Wu

Advisor: Juinn-Dar Huang, Ph.D.

July 11, 2024

本份講義大部分內容取材自台大李宏毅教授之課程教材

Outline (1/2)

- Machine learning framework
- Task category
  - Linear regression
  - Binary classification
  - Multiclass classification
- Neural networks and activation functions
  - Example: handwriting recognition

Outline (2/2)

3 steps for deep learning

- Step 1: **Define a set of functions**
  - Why “deep” neural network
- Step 2: **Evaluate goodness of function**
  - Supervised learning
  - Example: classification
- Step 3: **Pick the best function**
  - Backward propagation
  - Batch and epoch
  - Overfitting and validation

Machine Learning

機器學習
≈ 找一個函數的能力 **根據資料**

- Speech Recognition

$f($  $)=$ “How are you”

- Image Recognition

$f($  $)=$ “Cat”

- Playing Go

$f($  $)=$ “5-5” (next move)

- Dialogue System

$f($ “How are you?” $)=$ “I am fine.”

(what the user said) (system response)

Machine Learning Framework (1/3)

Framework

Image Recognition:

$f($  $)=$ "cat"

A set of function

Model

$f_1, f_2 \cdots$

$f_1($  $)=$ "cat"

$f_2($  $)=$ "monkey"

$f_1($  $)=$ "dog"

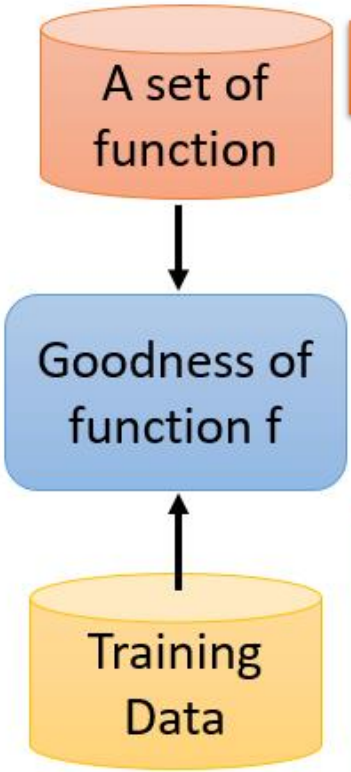
$f_2($  $)=$ "snake"

Machine Learning Framework (2/3)

Framework

Image Recognition:

$f($  $)=$ "cat"



$f_1($  $)=$ "cat"

$f_2($  $)=$ "monkey"

Better!

$f_1($  $)=$ "dog"

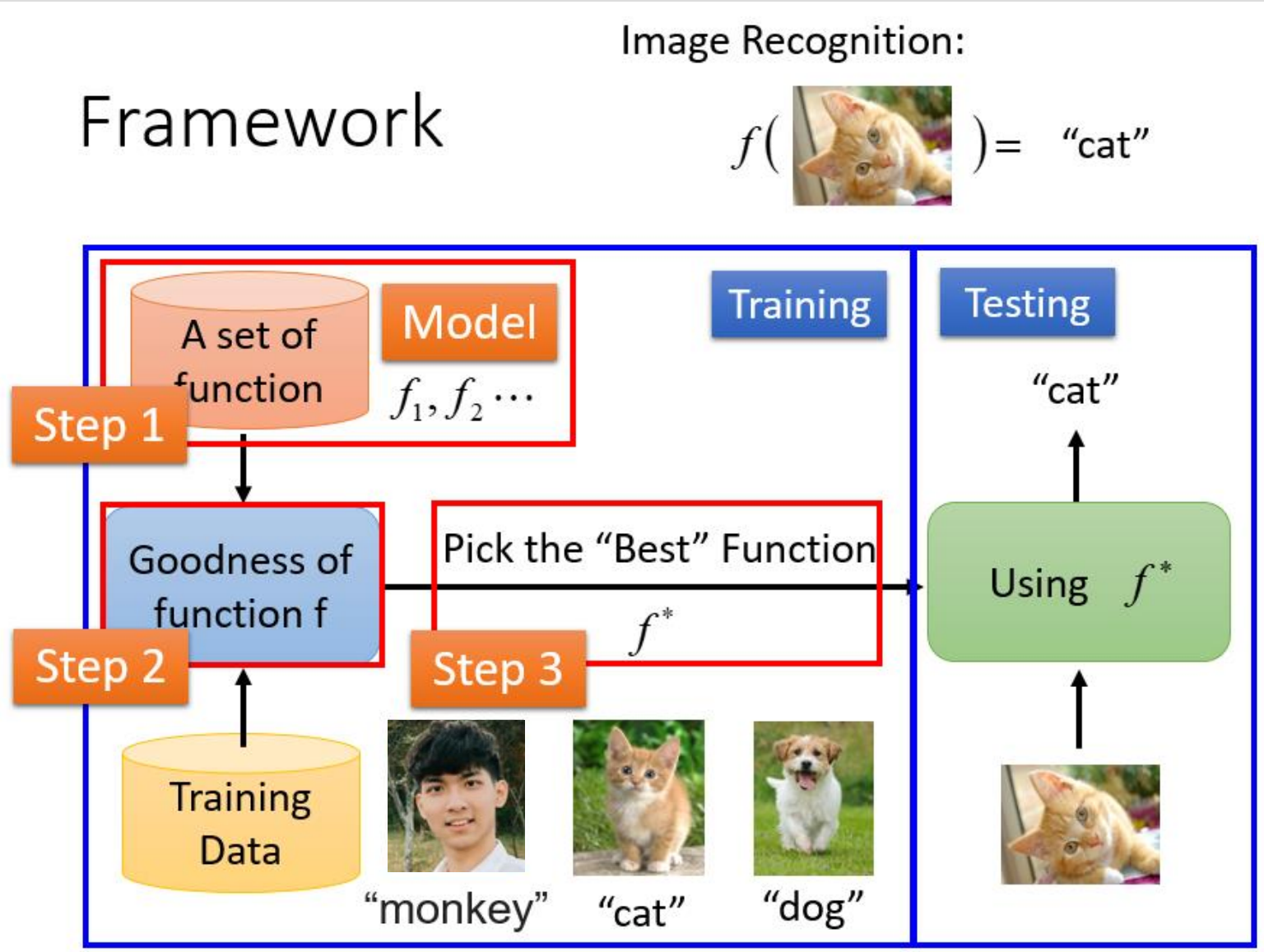
$f_2($  $)=$ "snake"

Supervised Learning (督導式學習)

function input:   

function output: "monkey" "cat" "dog"

Machine Learning Framework (3/3)



Three Steps for Deep Learning – Step 0

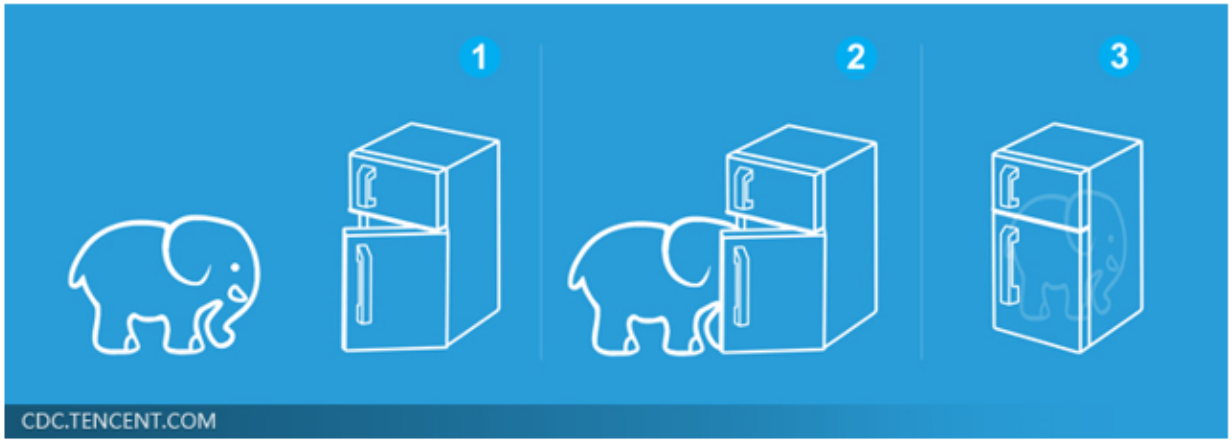
機器學習好簡單 .....

Different Tasks (任務)

Step 0: What kind of function do you want to find?

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

就好像把大象放進冰箱 .....



Task Category – Regression

Regression

The output of the target function f is "scalar".

Predict PM2.5

Training Data:

Input:

9/01 PM2.5 = 63 9/02 PM2.5 = 65

Output:

9/03 PM2.5 = 100

Input:

9/12 PM2.5 = 30 9/13 PM2.5 = 25

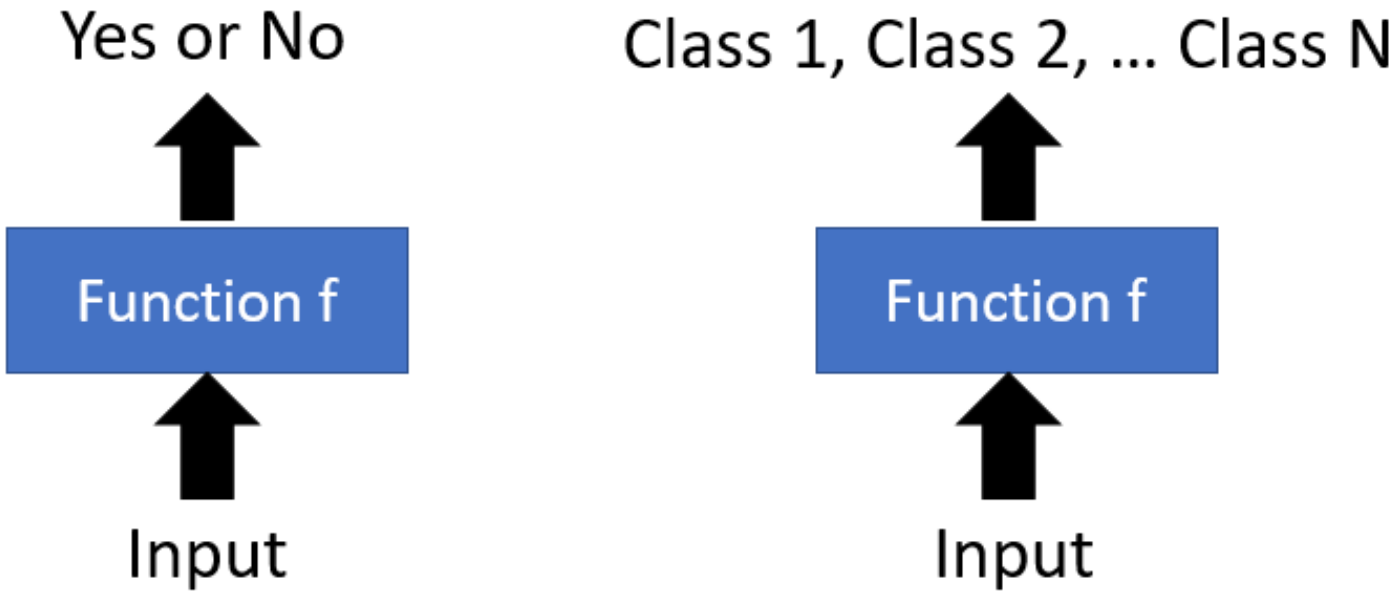
Output:

9/14 PM2.5 = 20

Task Category – Classification

Classification (分類)

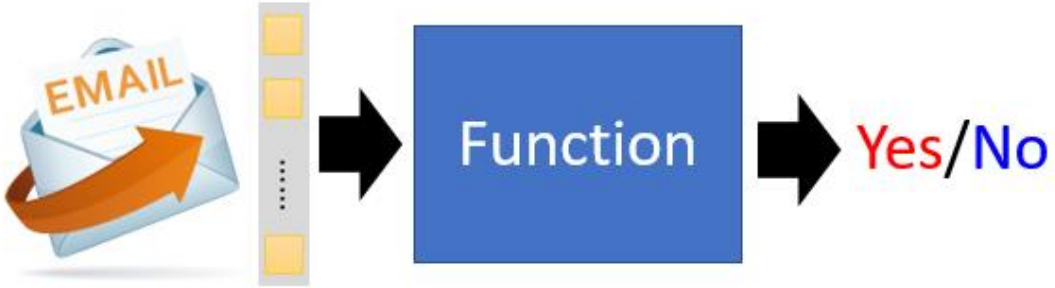
- Binary Classification (二元分類)
- Multi-class Classification (多類別分類)



Task Category – Binary Classification

二元分類

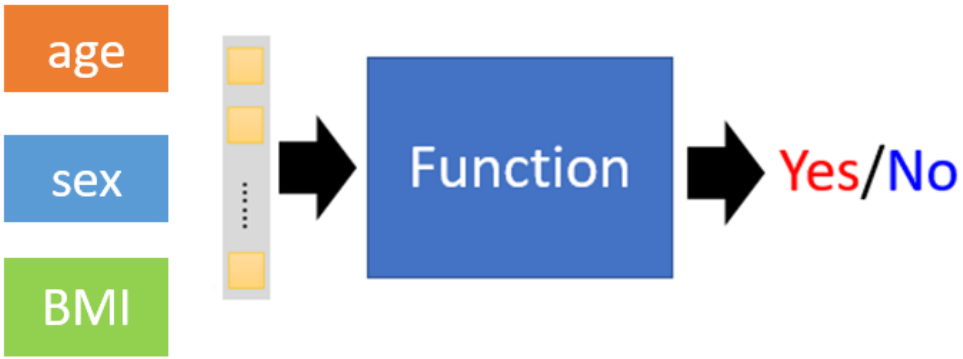
Spam filtering



(http://spam-filter-review.toptenreviews.com/)

Task Category – Binary Classification

二元分類

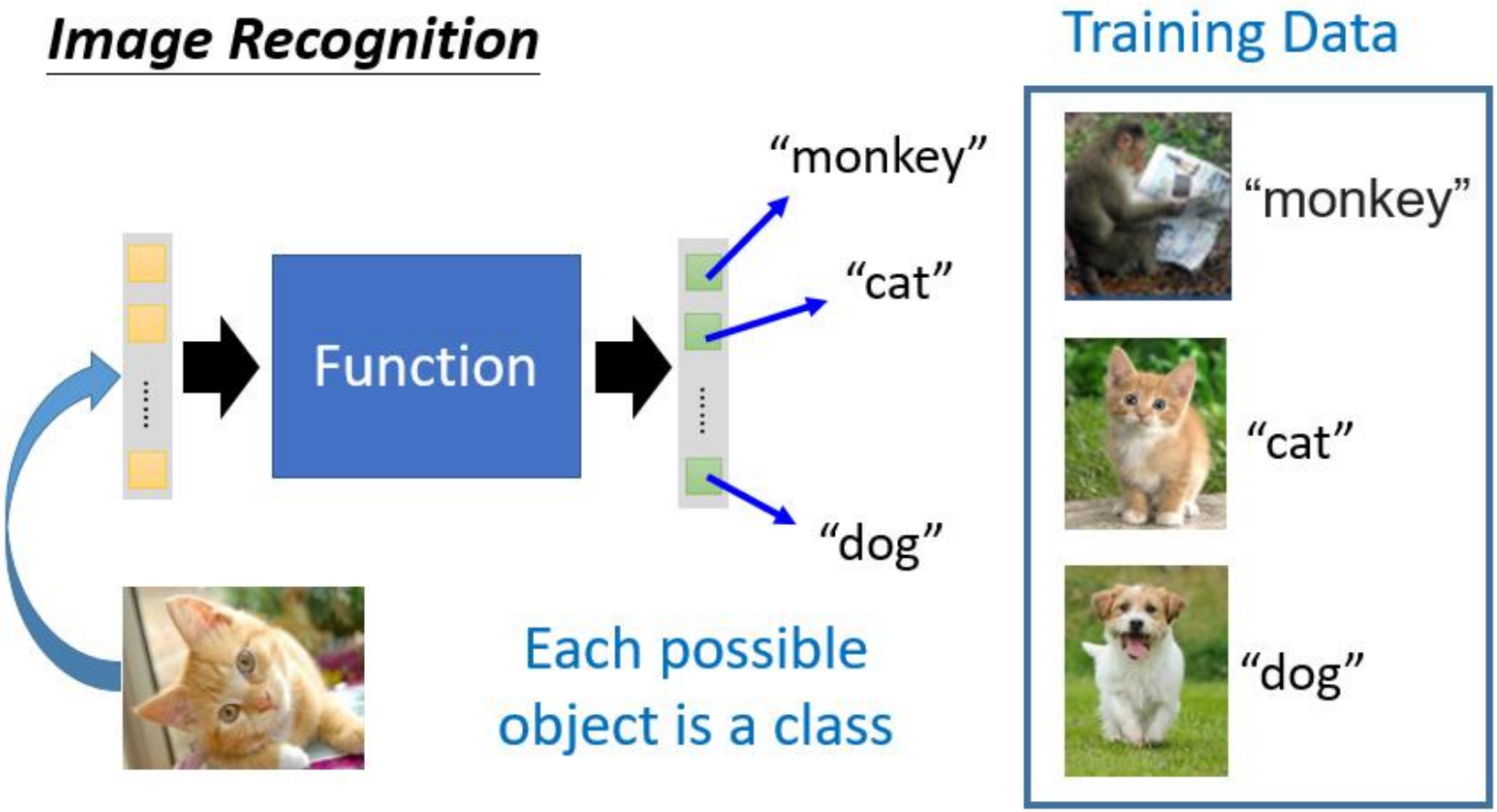


Training Data

ID	age	sex	BMI	HT
01	70	0	23.2	1
02	76	1	23.3	0
03	84	1	23.4	0
04	85	1	23	0
…	…	…	…	…
…	…	…	…	…
62	55	0	24.4	1
63	71	0	24.6	0
64	56	0	23.7	0
65	70	0	24	1

Task Category – Multiclass Classification

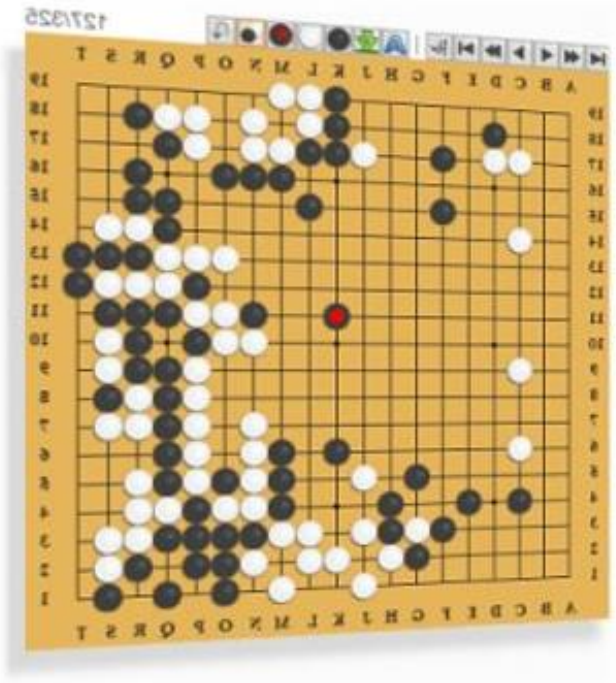
多類別分類



Task Category – Multiclass Classification

多類別分類

Playing GO



Each position is a class

(19 x 19 classes)

Function

右上星位

左下星位

天元

Next move

Three Steps for Deep Learning – Step 1

機器學習好簡單 .....

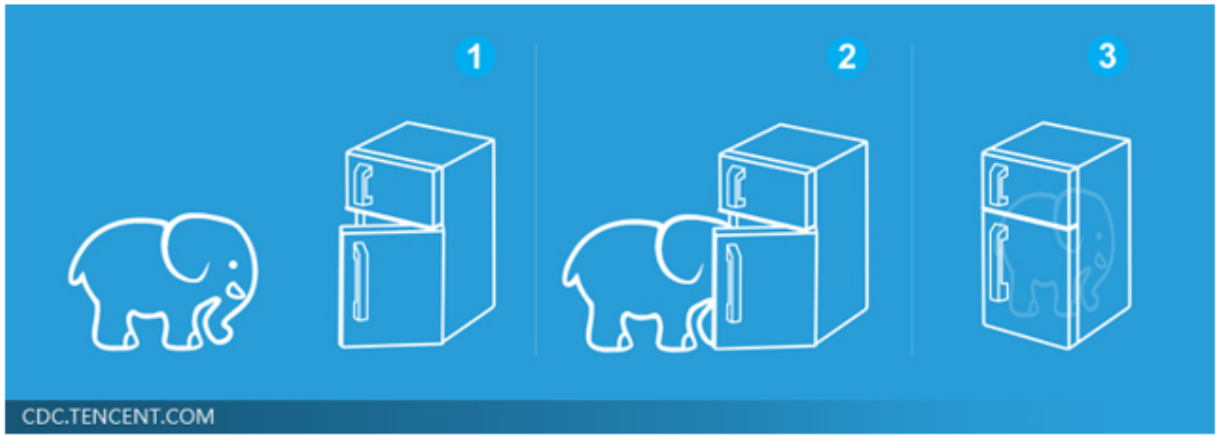
Step 0: What kind of function do you want to find?

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

就好像把大象放進冰箱 .....



Deep Learning

Deep Learning (深度學習)

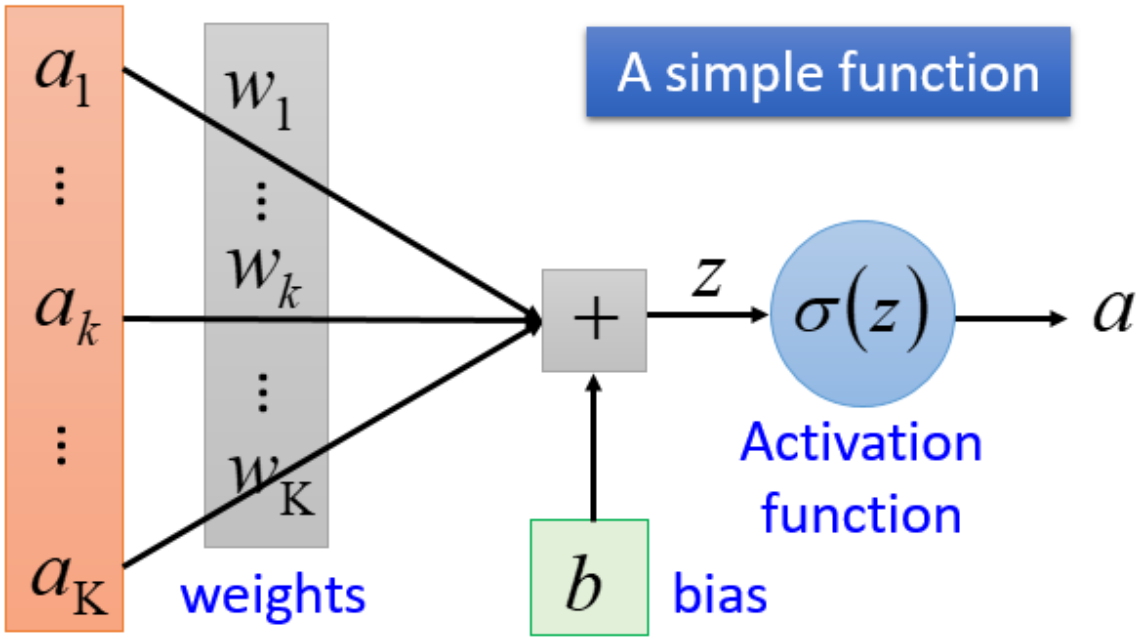
- Deep learning, SVM, decision tree ….
  - →using different ways to represent a function
- Using neural network (神經網路) to represent a function

Neural Network (1/5)

Neural Network (神經網路)

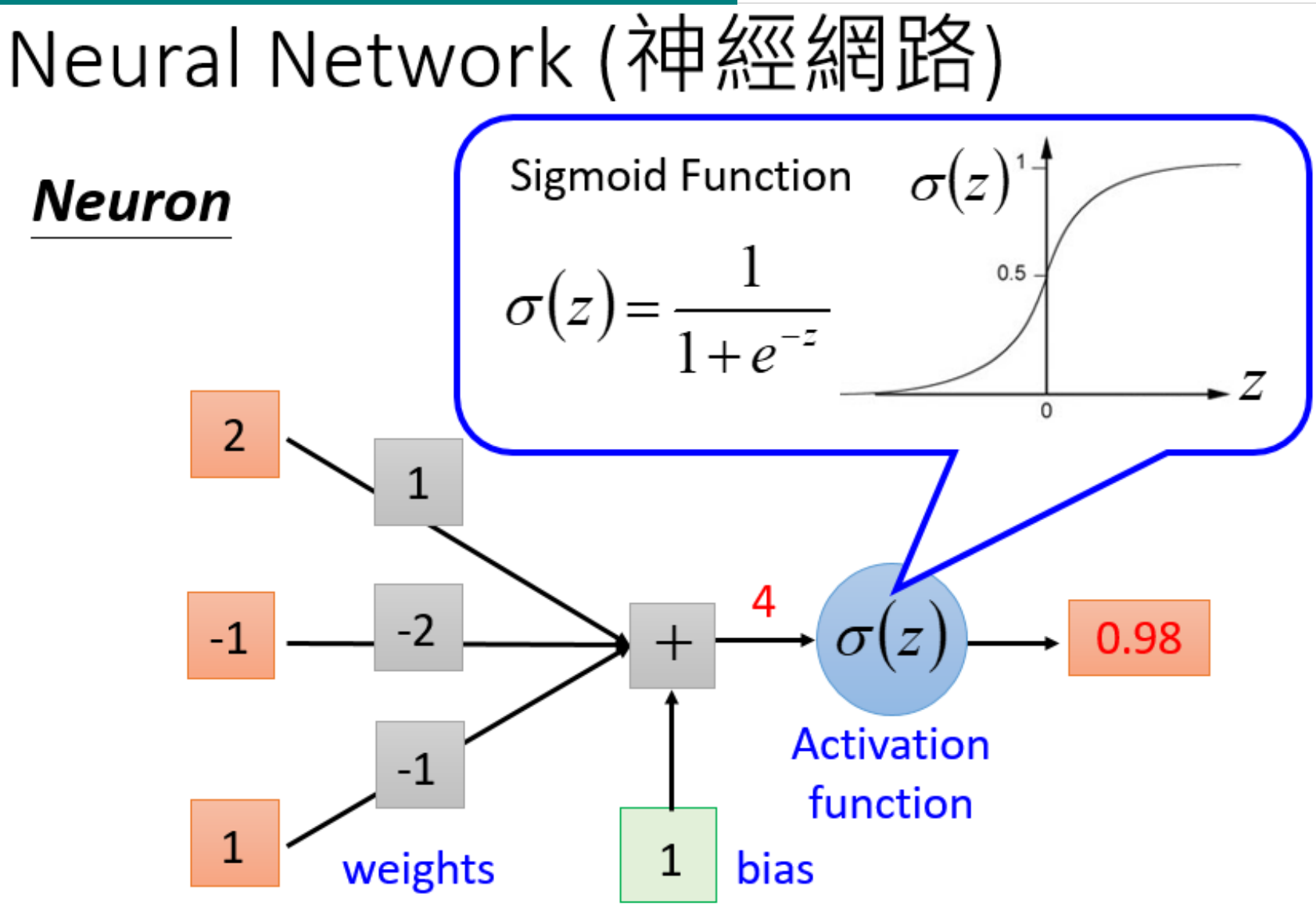
Neuron (神經元)

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



Weights and biases are called network parameters

Neural Network (2/5)



Common Activation Functions (1/5)

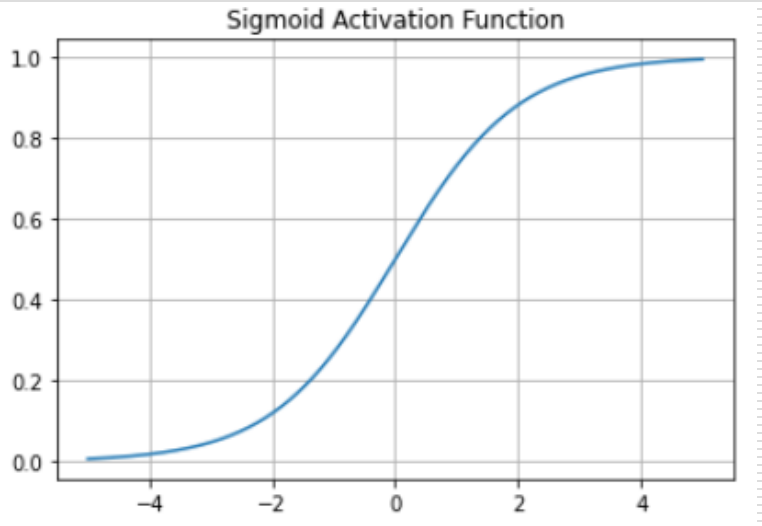
- Increase Non-linearity
  - (w1x+b1) * w2 + b2 = w1w2x + (w2b1 + b2) = w'x + b'
  - Two linear layers will merge to one layer if not use activation function

- Common activation functions
  - Sigmoid, Tanh
  - ReLU, ReLU6, Leaky ReLU
  - GELU
  - Swish, Hardswish

Common Activation Functions (2/5)

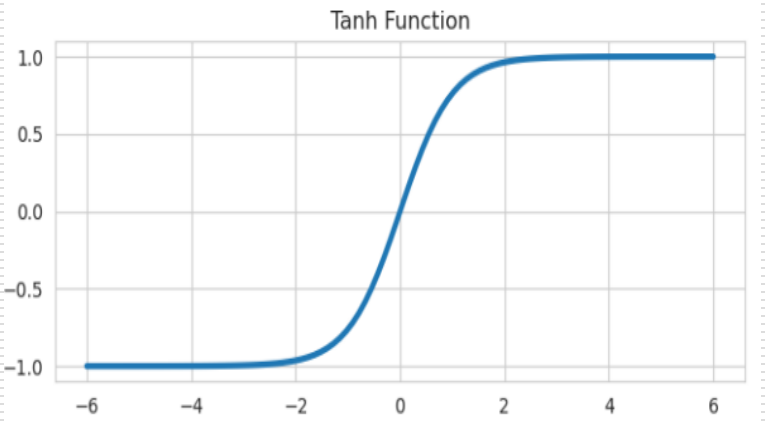
- Sigmoid
  - torch.nn.Sigmoid
  - Output from 0 to 1
  - x = 0 has the maximum slop

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$



- Tanh
  - torch.nn.Tanh
  - Output from -1 to 1
  - The mean of tanh is 0

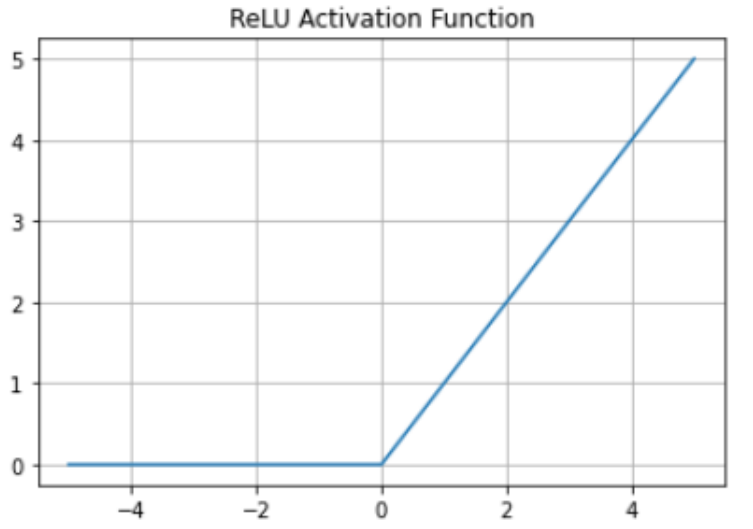
$$\text{Tanh}(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



Common Activation Functions (3/5)

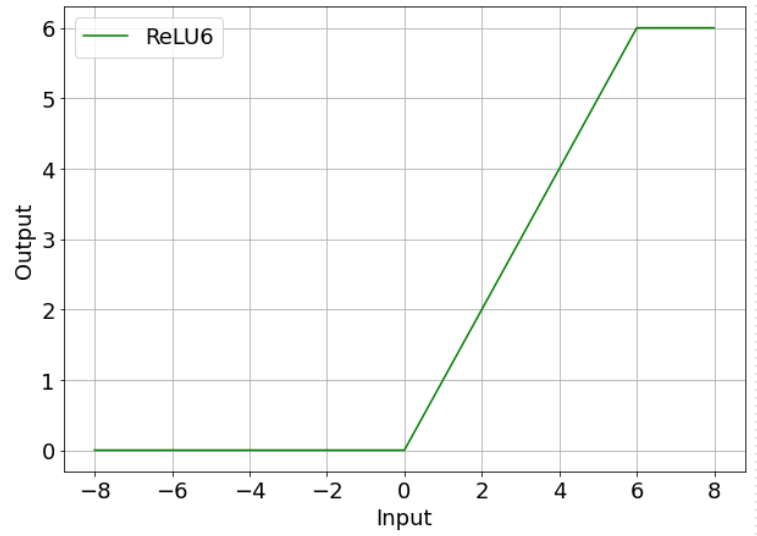
- ReLU
  - torch.nn.ReLU

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



- ReLU6
  - torch.nn.ReLU6
  - If output is beyond 6, then output 6
  - Used in lightweight networks
  - To meet the low precision needs of float16/int8

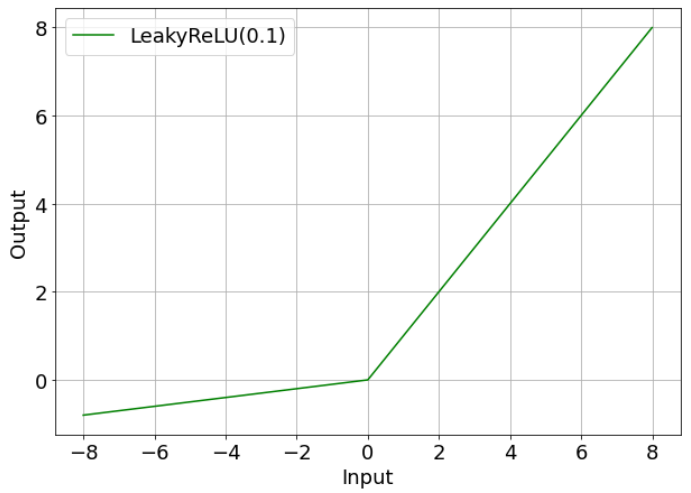
$$\mathrm{ReLU6}(x) = \min(\max(0, x), 6)$$



Common Activation Functions (4/5)

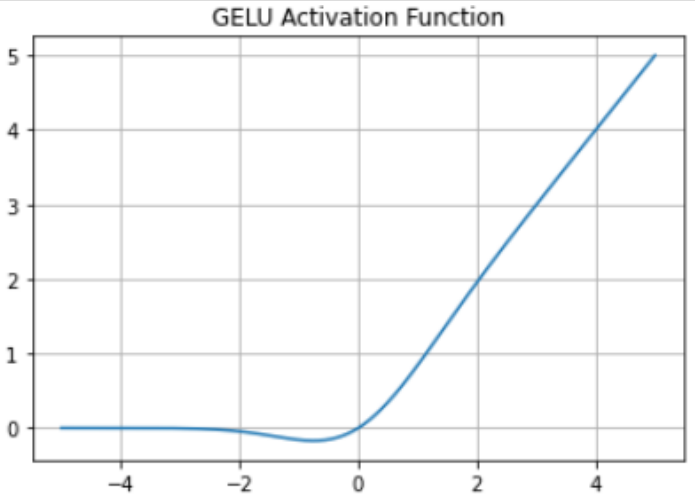
- Leaky ReLU
  - torch.nn.LeakyReLU(negative_slope)
  - If output is under 0,
    then output negative_slope * x
  - Solve the problem of
    vanishing gradient

$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$



- GELU
  - torch.nn.GELU
  - Provide well defined gradient
    for negative inputs
  - Transformer (BERT, GPT-2)

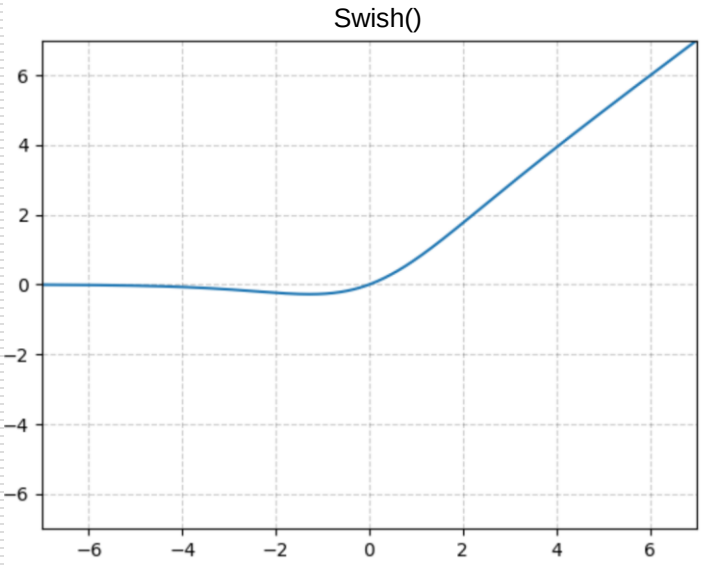
$$GELU(x) = 0.5x(1 + tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$



Common Activation Functions (5/5)

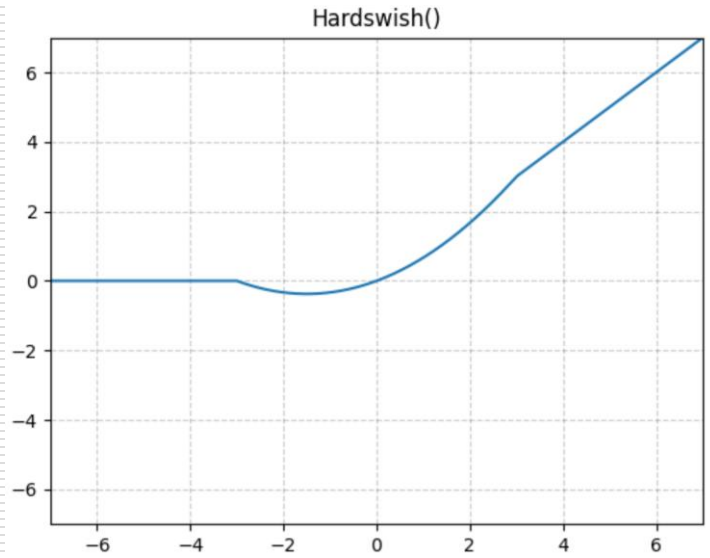
- Swish
  - torch.nn.SiLU (if β = 1)
  - β is a learnable parameter

$$\mathrm{Swish}(x) = x \cdot \mathrm{Sigmoid}(\beta x)$$

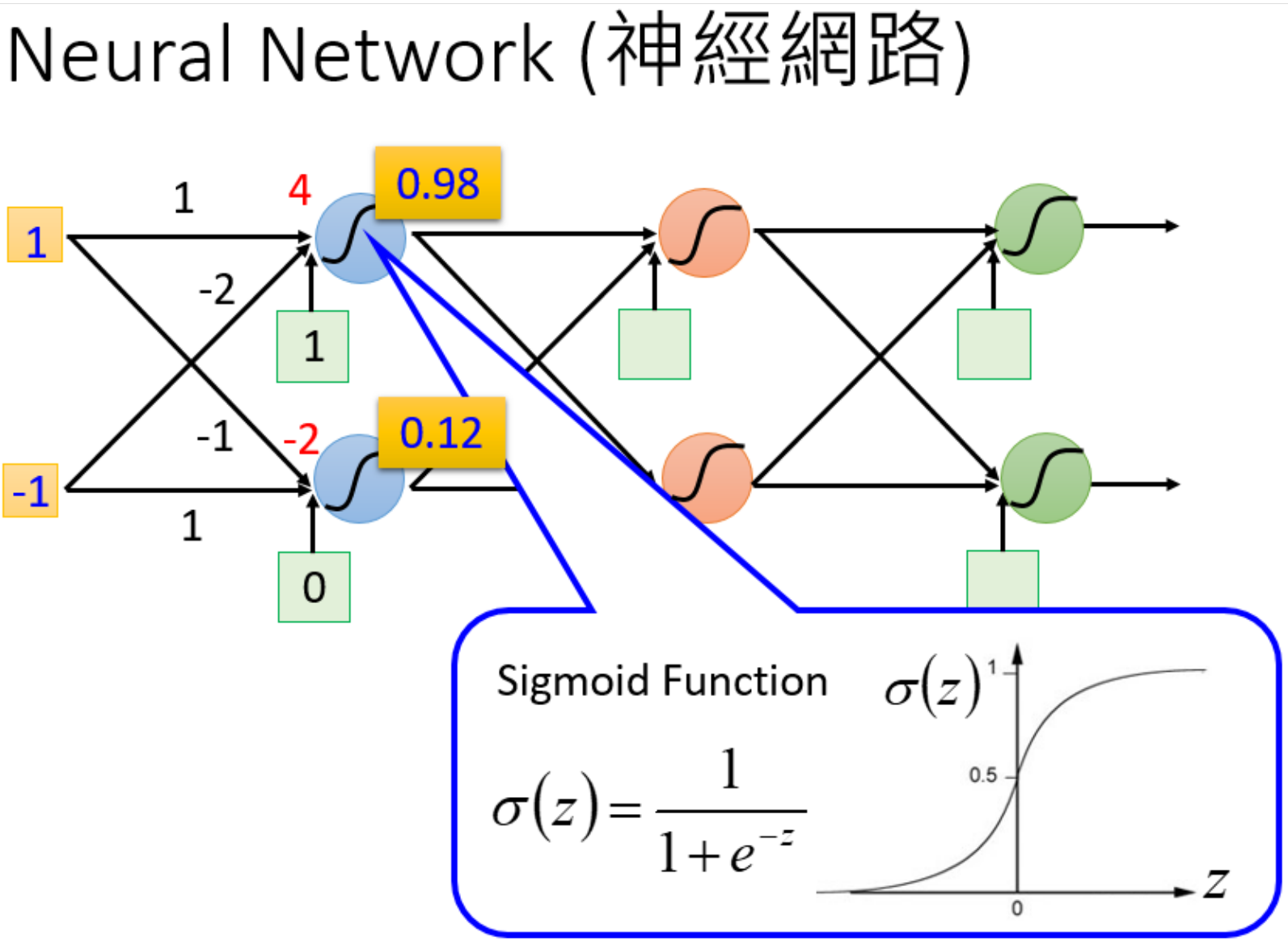


- Hardswish
  - torch.nn.Hardswish

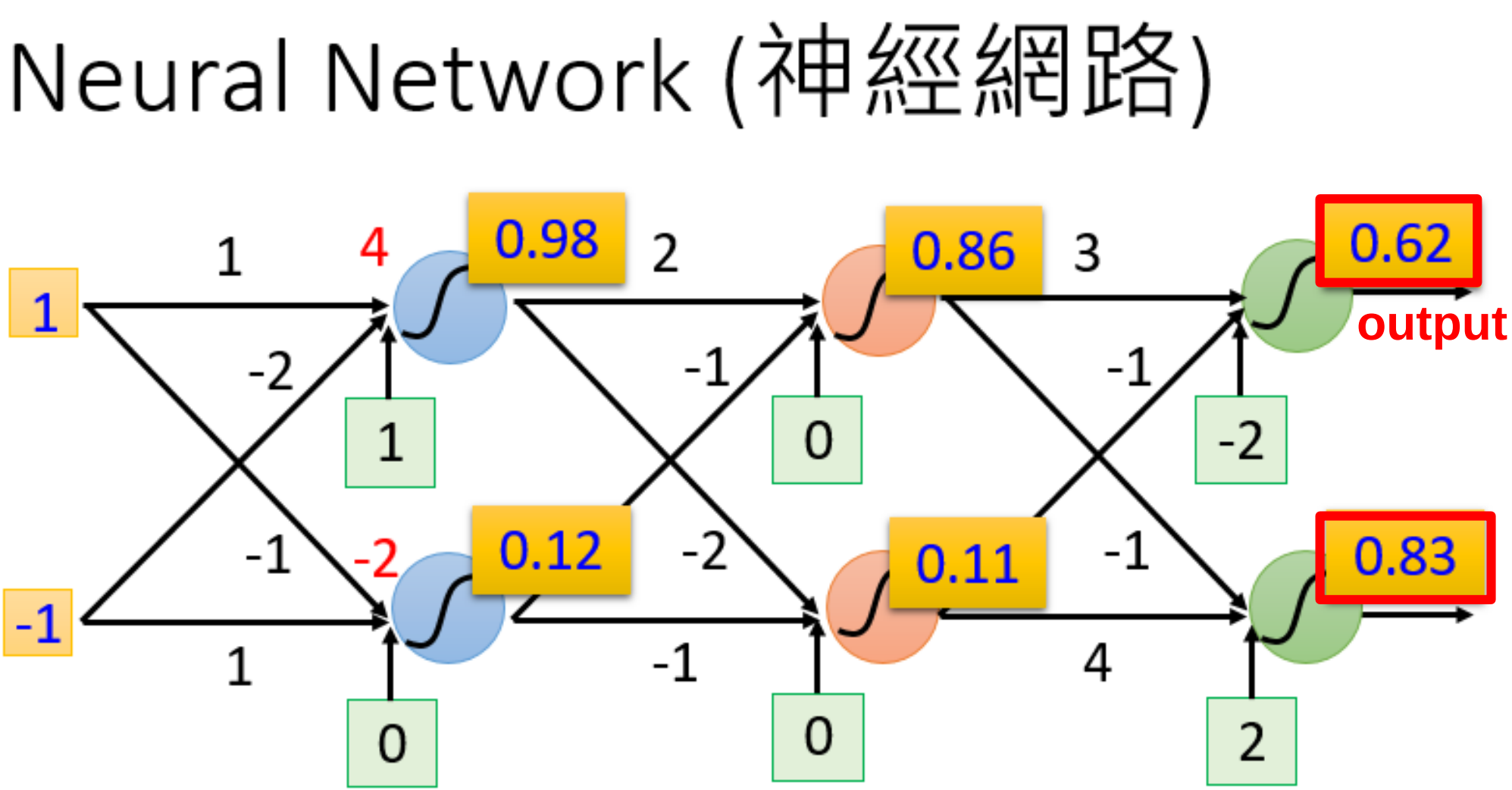
$$\mathrm{Hardswish}(x) = \begin{cases} 0 & \text{if } x \leq -3, \\ x & \text{if } x \geq +3, \\ x \cdot (x+3)/6 & \text{otherwise} \end{cases}$$



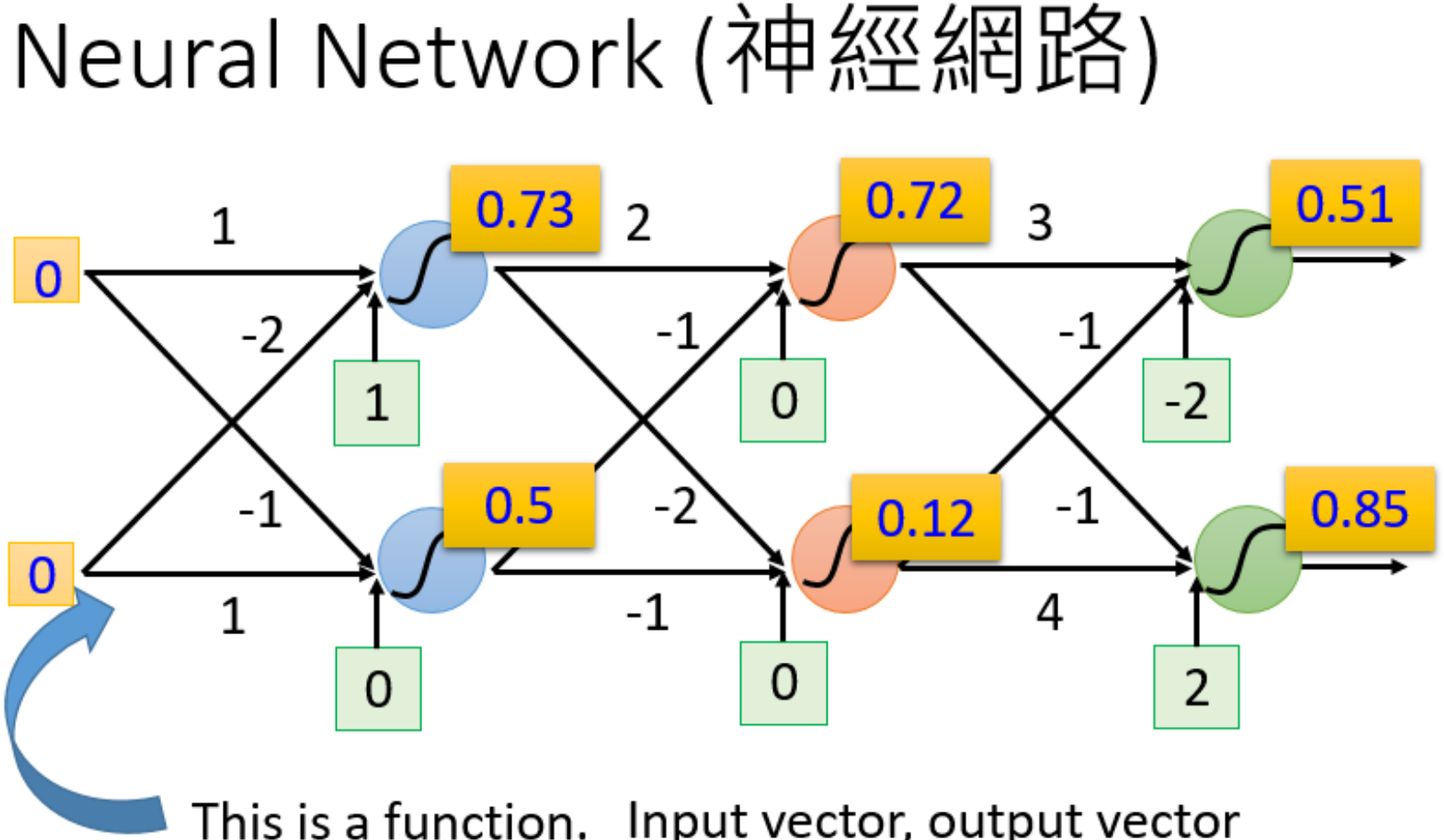
Neural Network (3/5)



Neural Network (4/5)



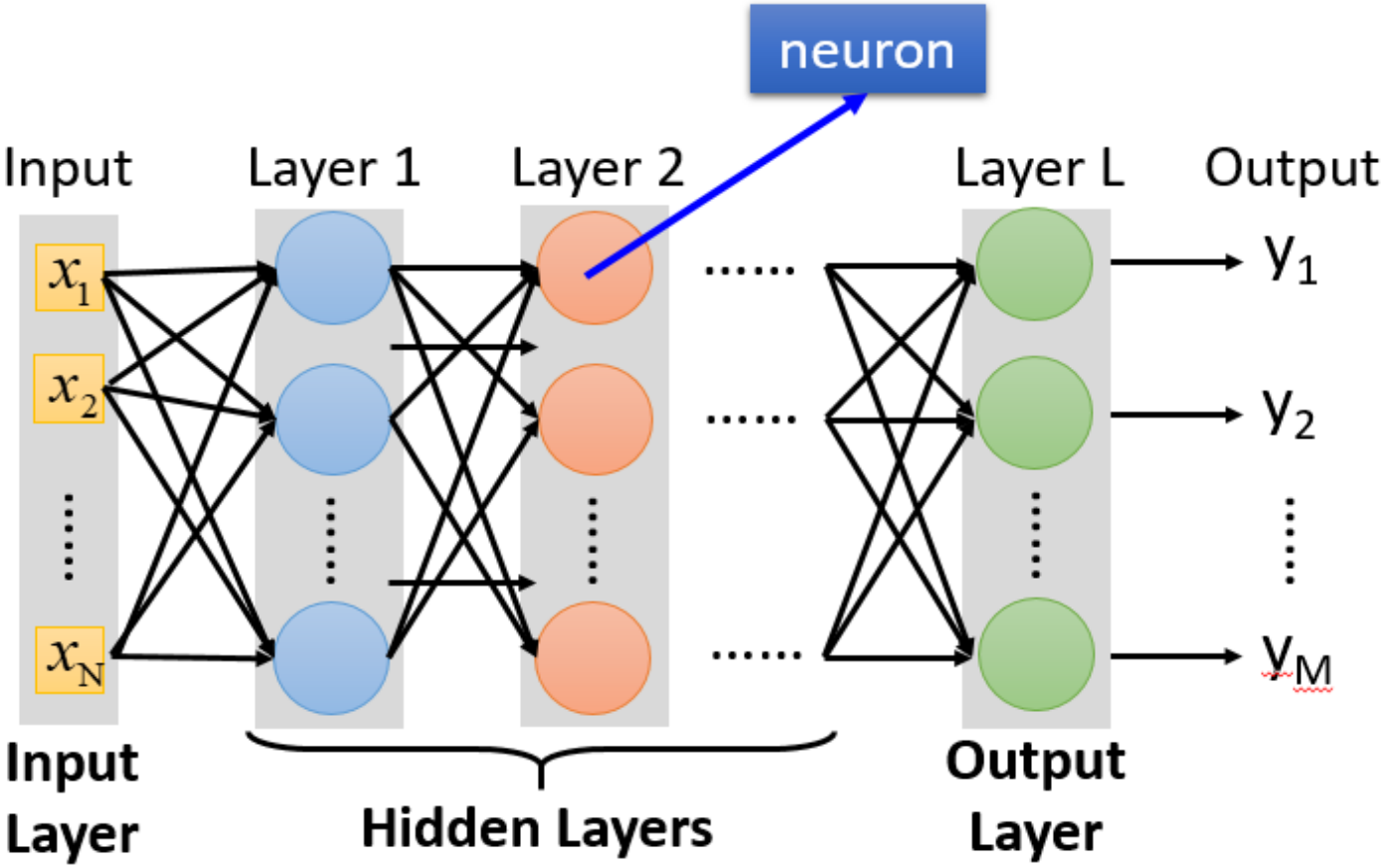
Neural Network (5/5)



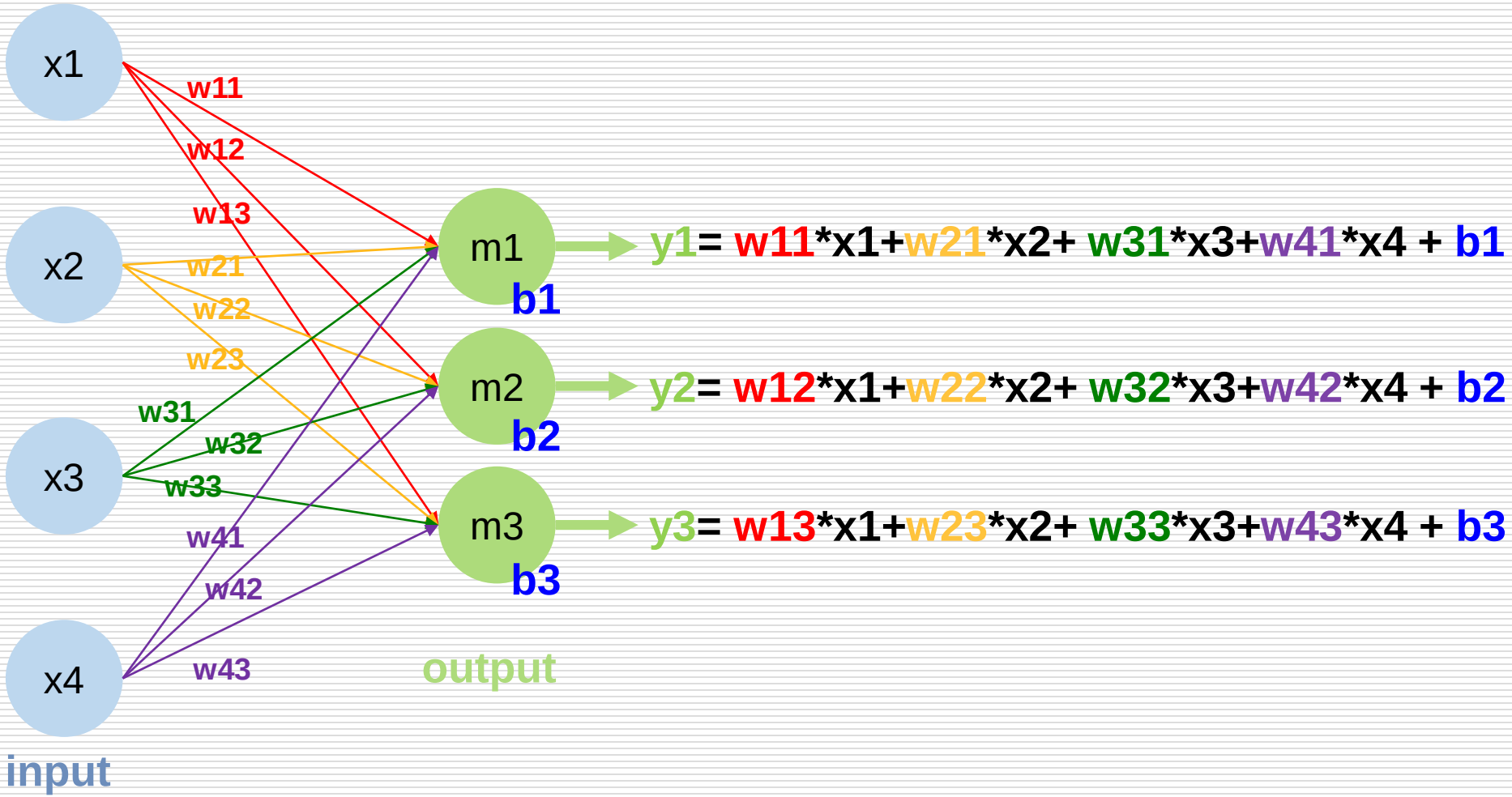
Fully Connected Feedforward Network

Fully Connected Feedforward Network

a.k.a.
Multilayer Perceptron (MLP)



Computation Cost (1/2)



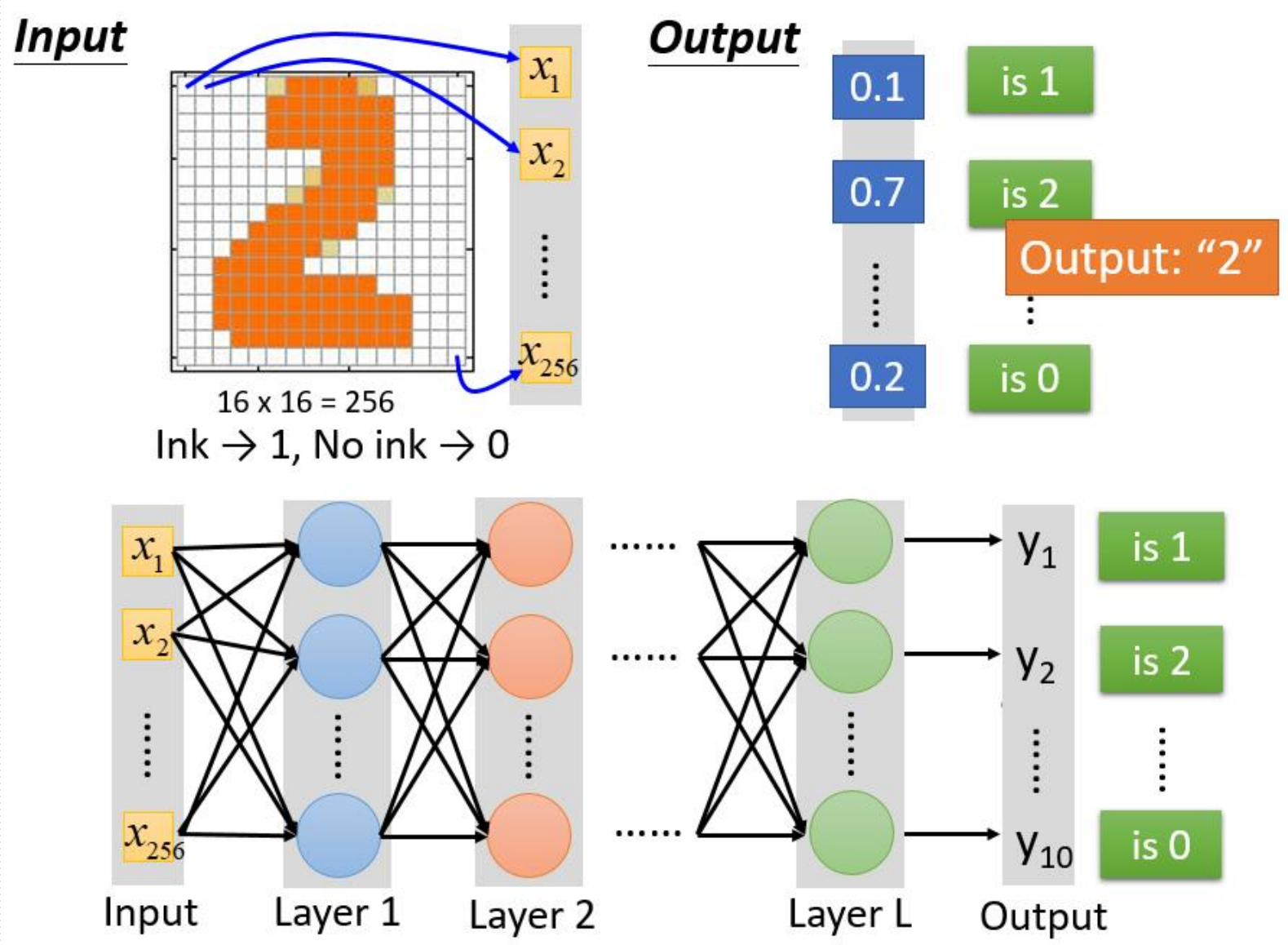
Computation Cost (2/2)

- Input dimension: 4
- Output dimension: 3
  → Weight: 4x3 matrix
  → a Fully-Connected layer can change dimension

- Need **4x3 MACs** to compute FC for **one** input vector
  – Need **nx4x3 MACs** to compute **n** input vectors

$$\begin{bmatrix} w11 & w21 & w31 & w41 \\ w12 & w22 & w32 & w42 \\ w13 & w23 & w33 & w43 \end{bmatrix} x \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix} = \begin{bmatrix} y1 \\ y2 \\ y3 \end{bmatrix}$$

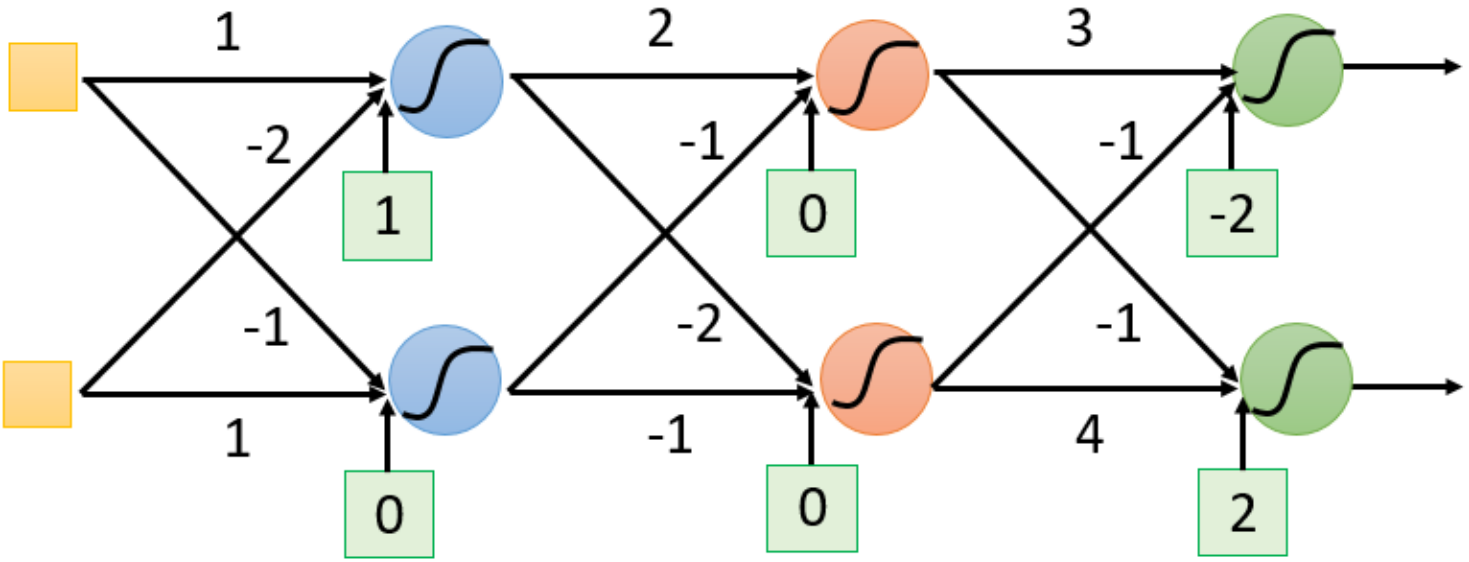
Example: Handwriting Recognition



What is define a set of functions

Define a Set of Functions (1/2)

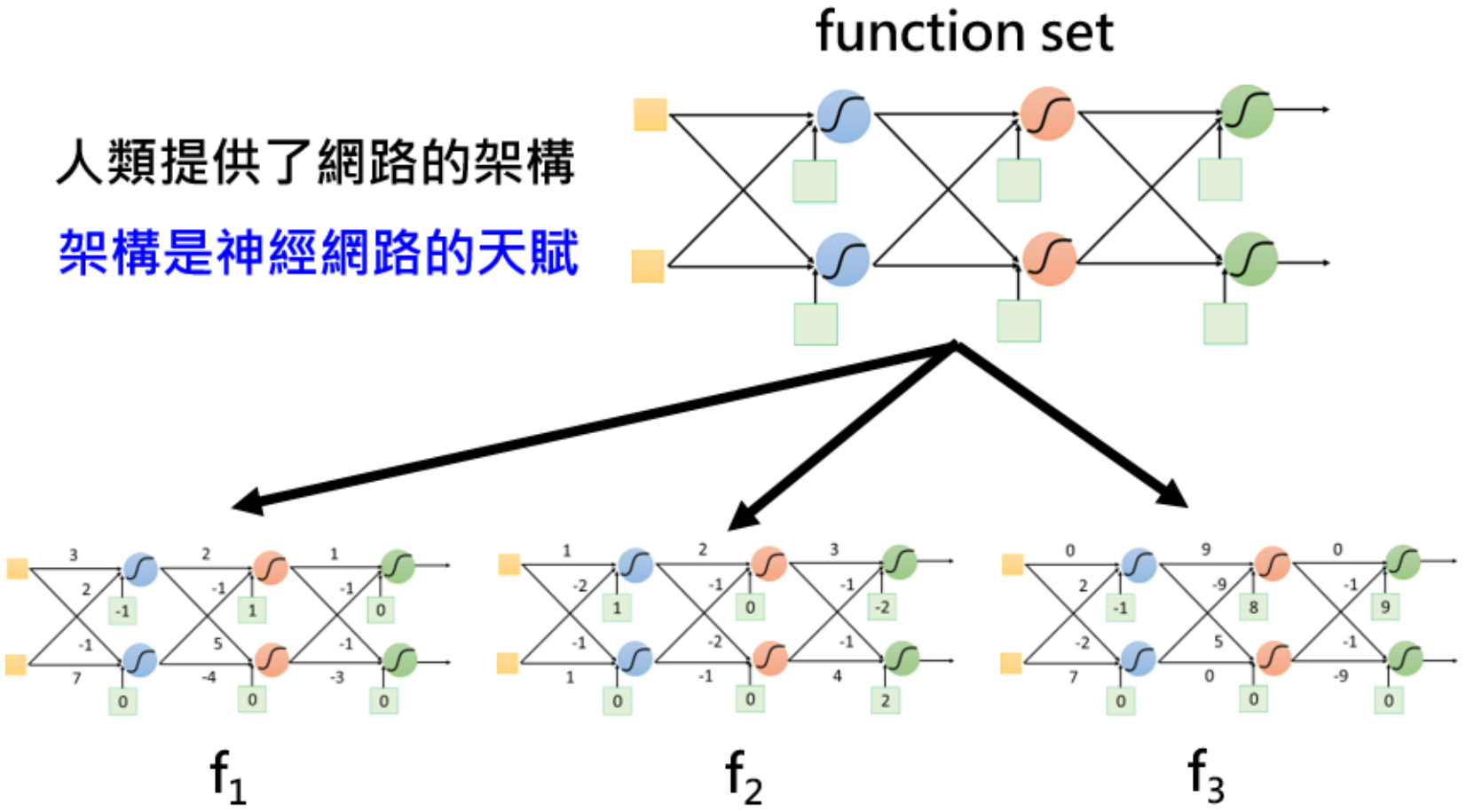
Neural Network (神經網路)



The developer only has to provide network structure (架構).

The parameters are found automatically from data.

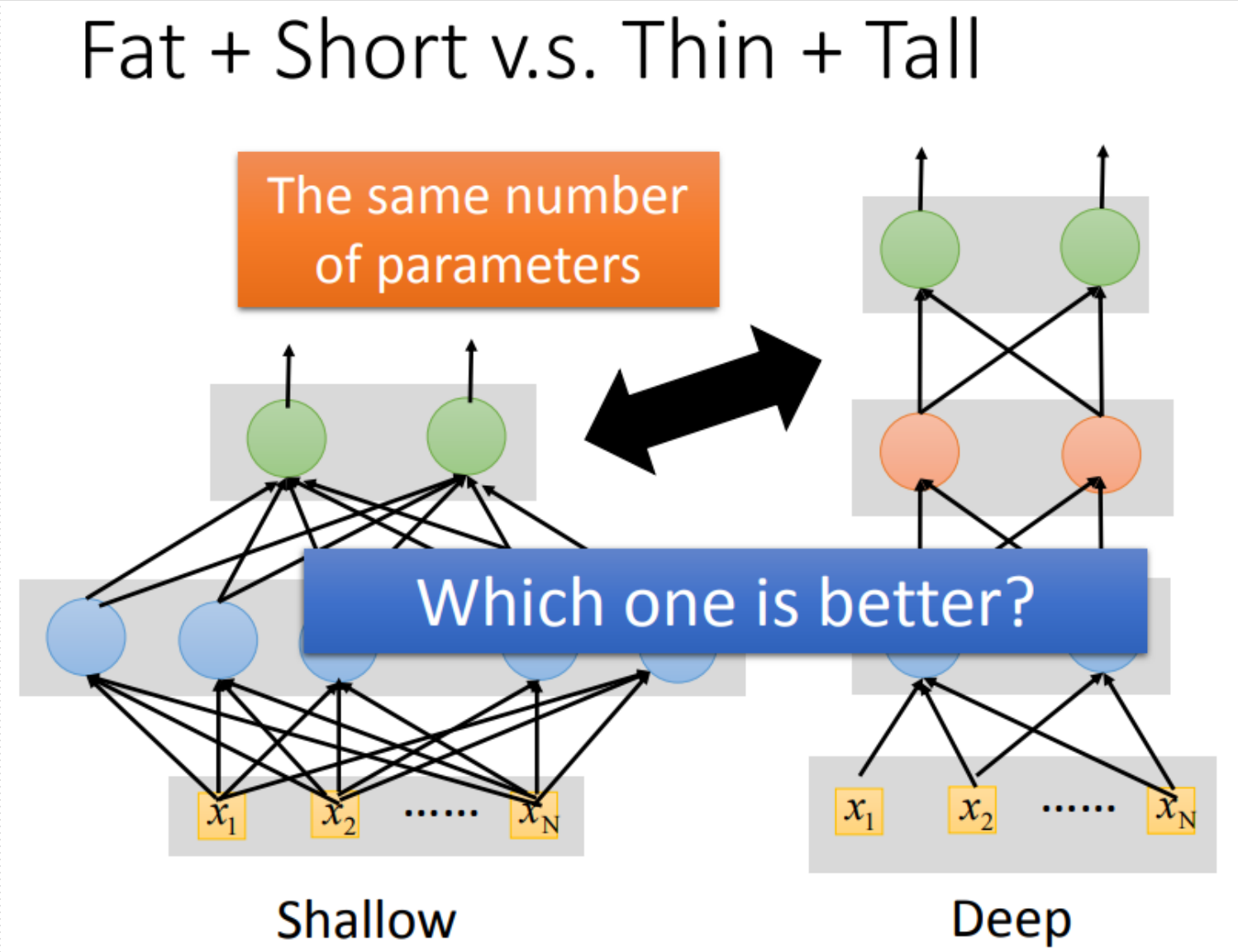
Define a Set of Functions (2/2)



機器自己根據資料找出參數 (也就是選擇了某一個 function)

機器自己後天學習的成果

Why “Deep” Neural Network (1/8)



Why "Deep" Neural Network (2/8)

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?

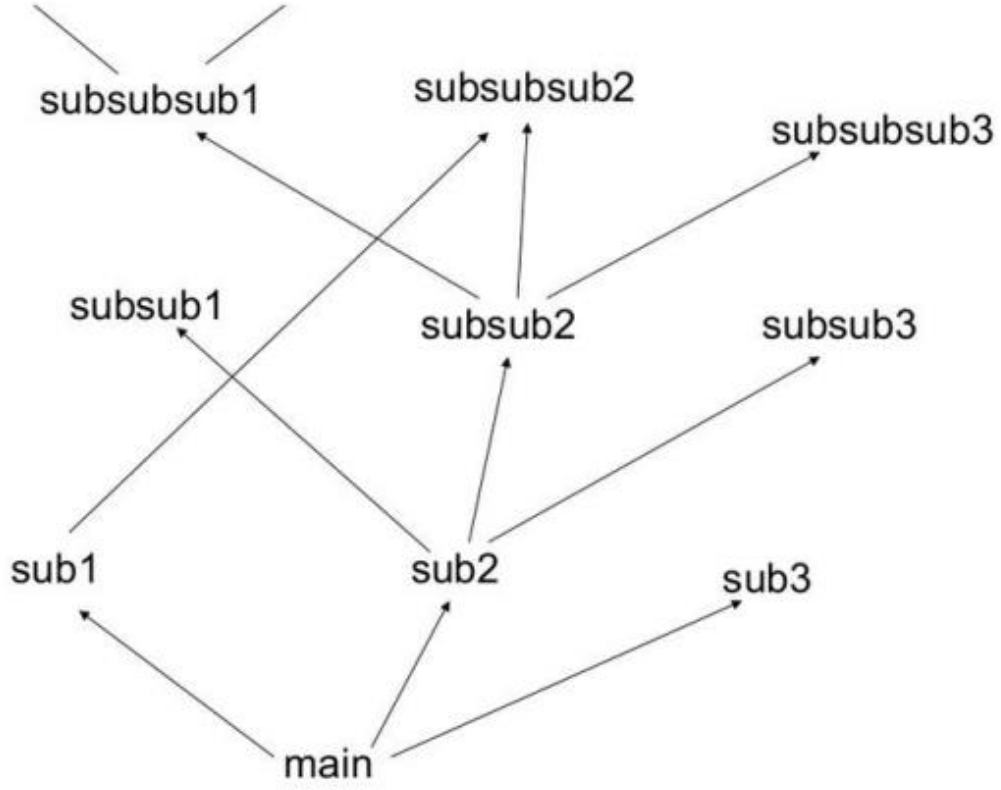
Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Why “Deep” Neural Network (3/8)

Modularization

- Deep → Modularization

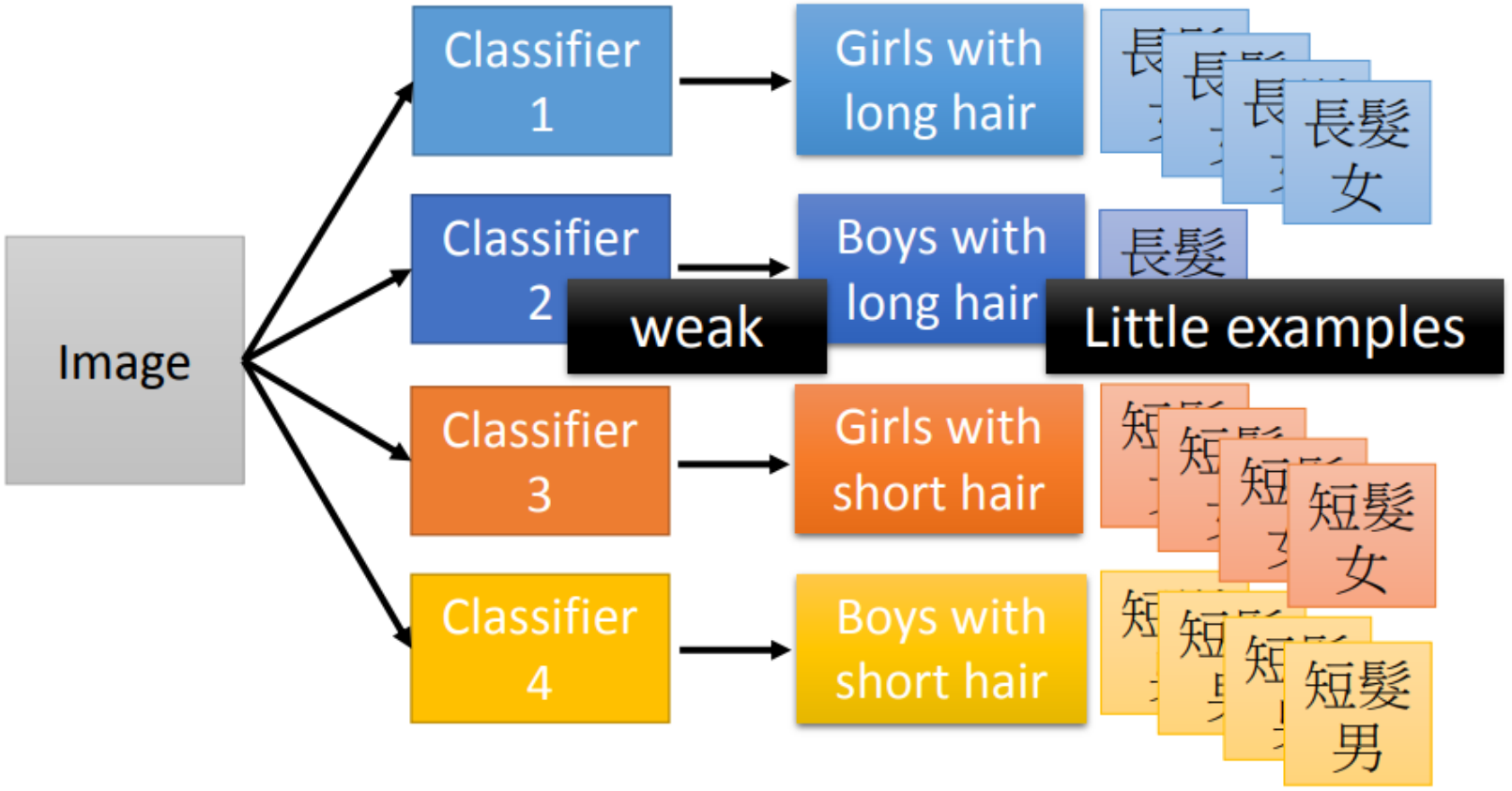
Don’t put everything in your main function.



Why “Deep” Neural Network (4/8)

Modularization

- Deep → Modularization

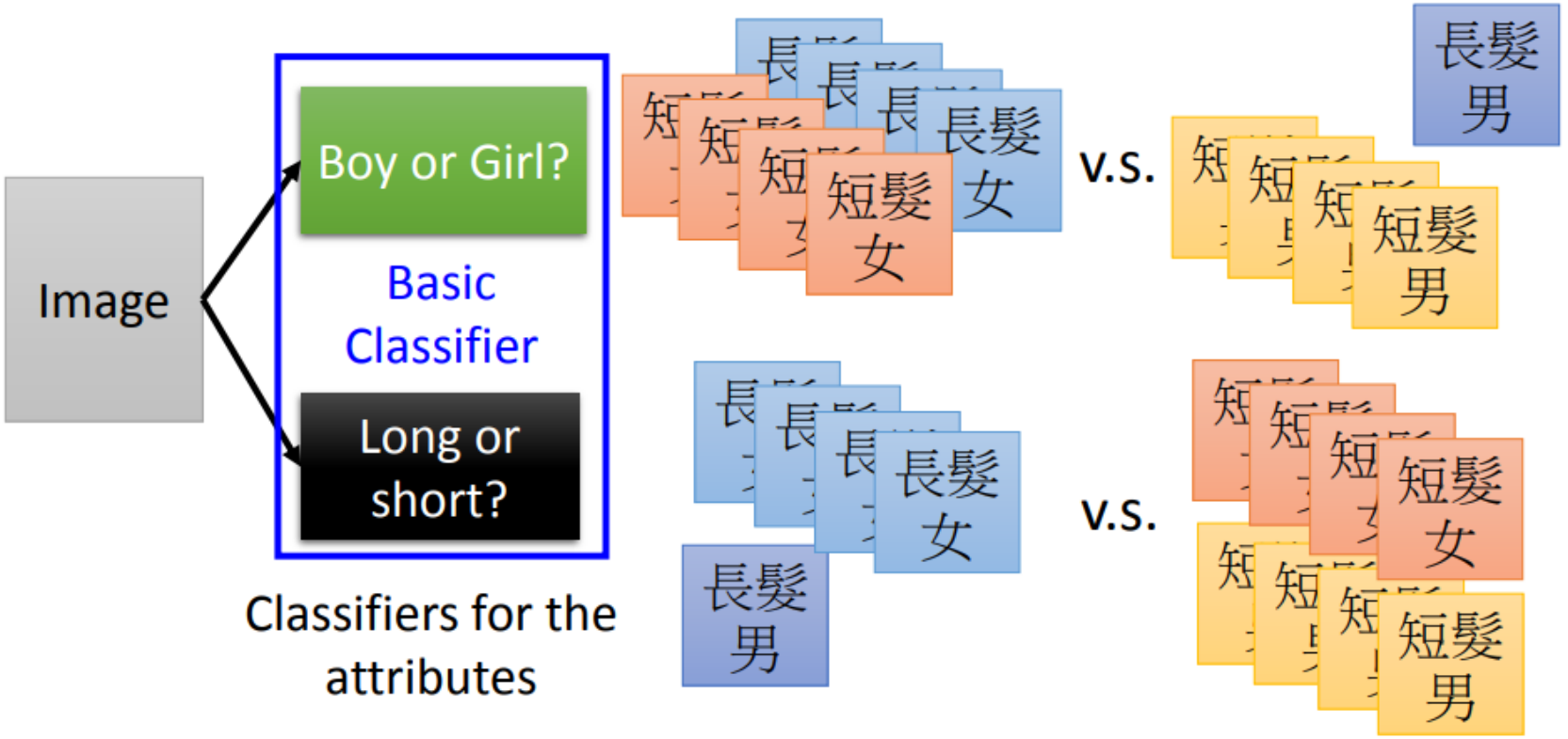


Why "Deep" Neural Network (5/8)

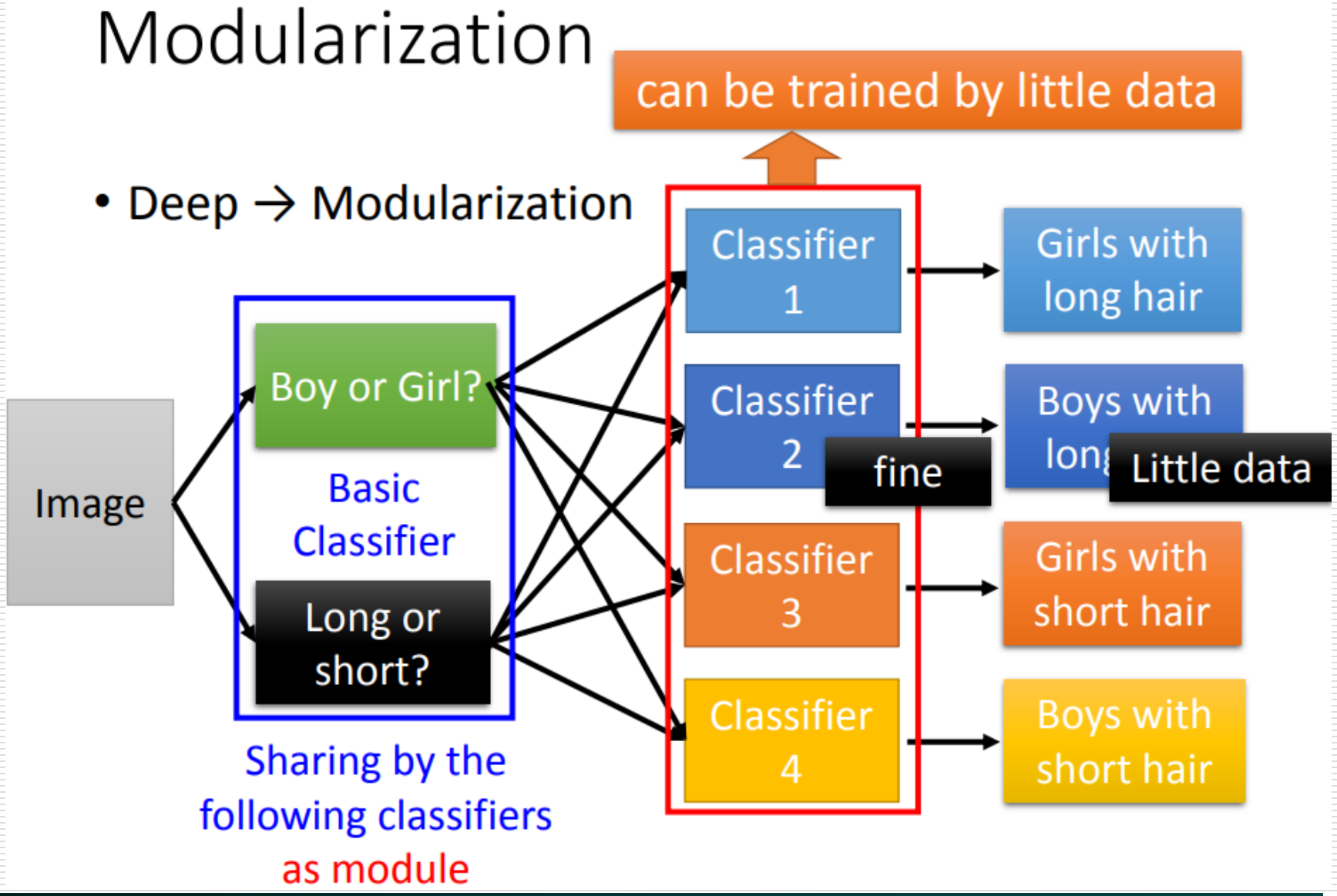
Modularization

Each basic classifier can have sufficient training examples.

- Deep → Modularization



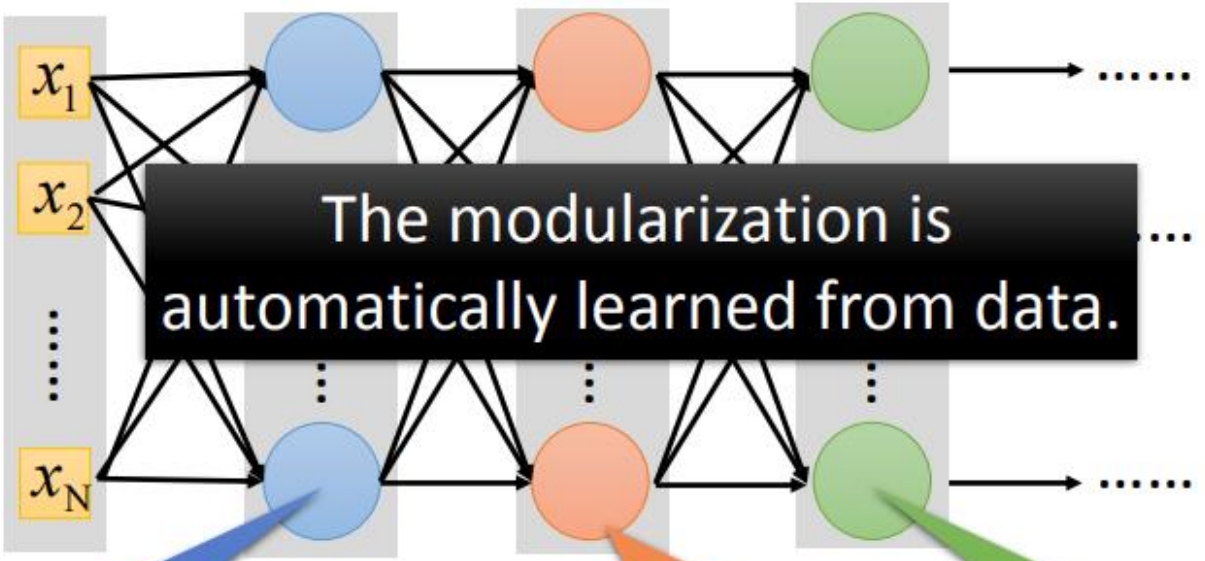
Why "Deep" Neural Network (6/8)



Why "Deep" Neural Network (7/8)

Modularization

- Deep → Modularization → Less training data?



The most basic classifiers

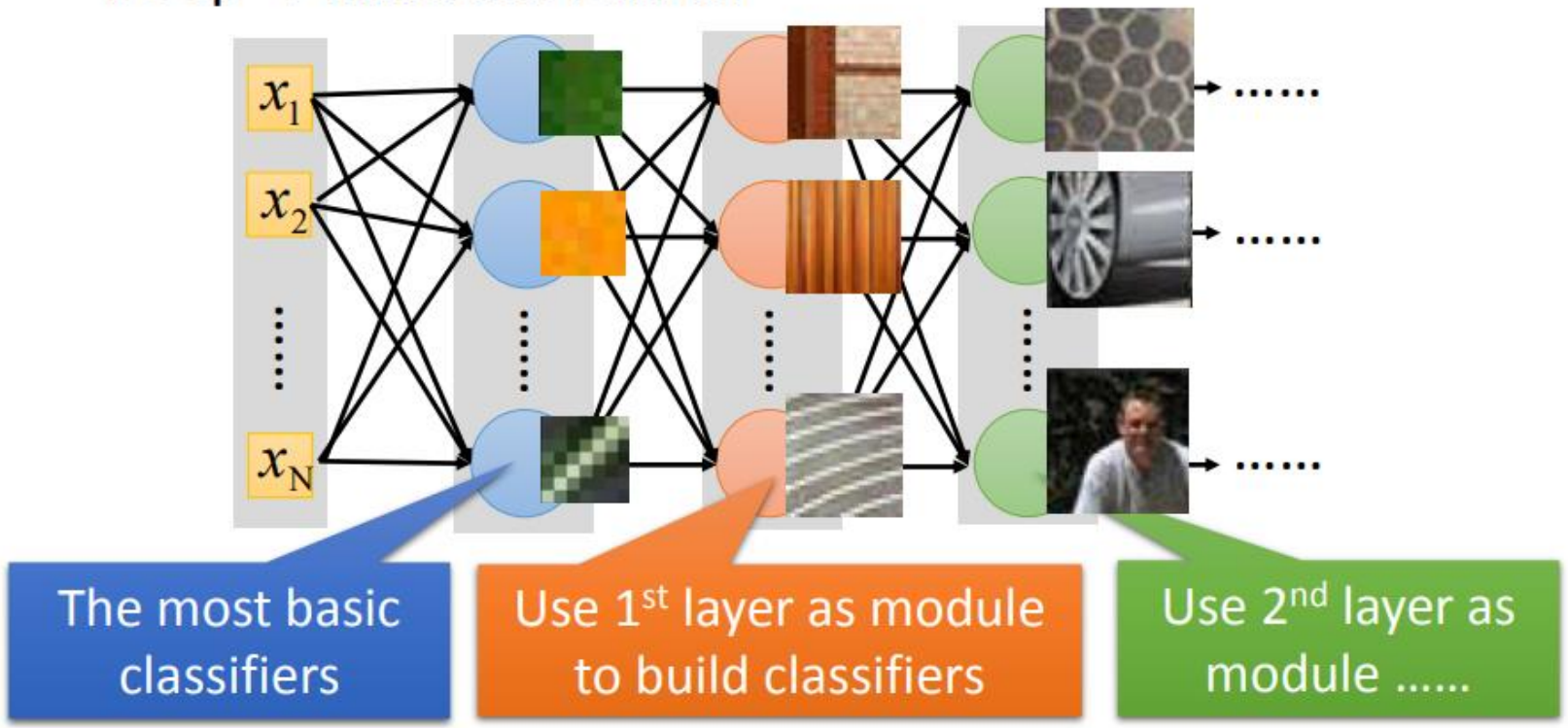
Use 1st layer as module to build classifiers

Use 2nd layer as module .....

Why "Deep" Neural Network (8/8)

Modularization - Image

- Deep → Modularization

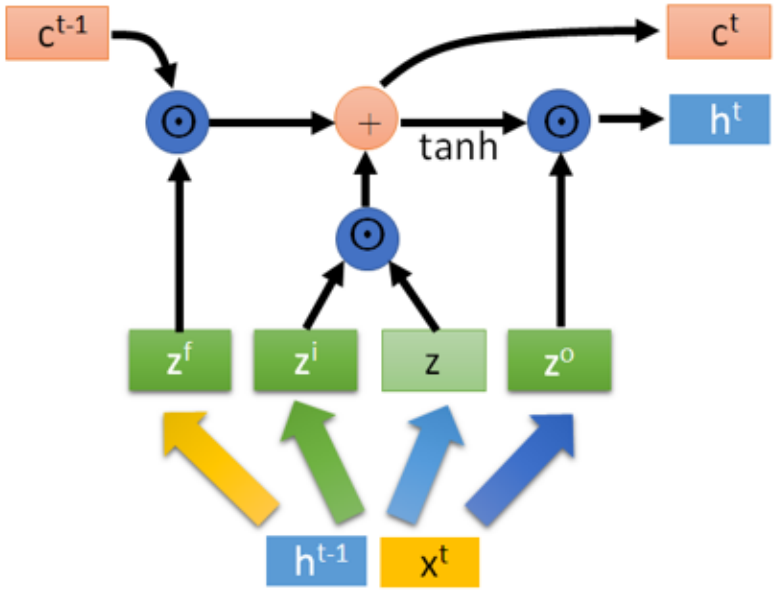


Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)

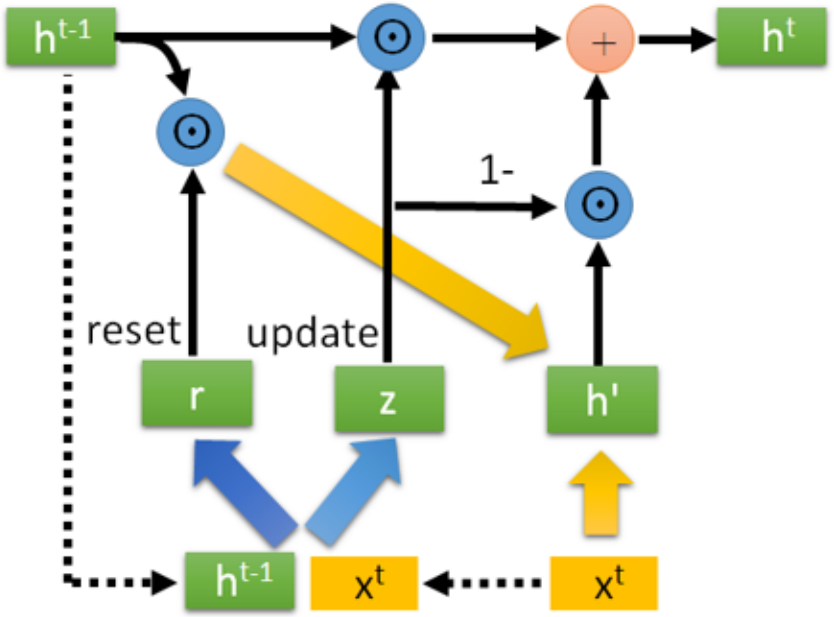
Different Network Structures

Different Network Structures

- CNN, LSTM, GRU, etc. are just different ways to connect neurons.



LSTM



GRU

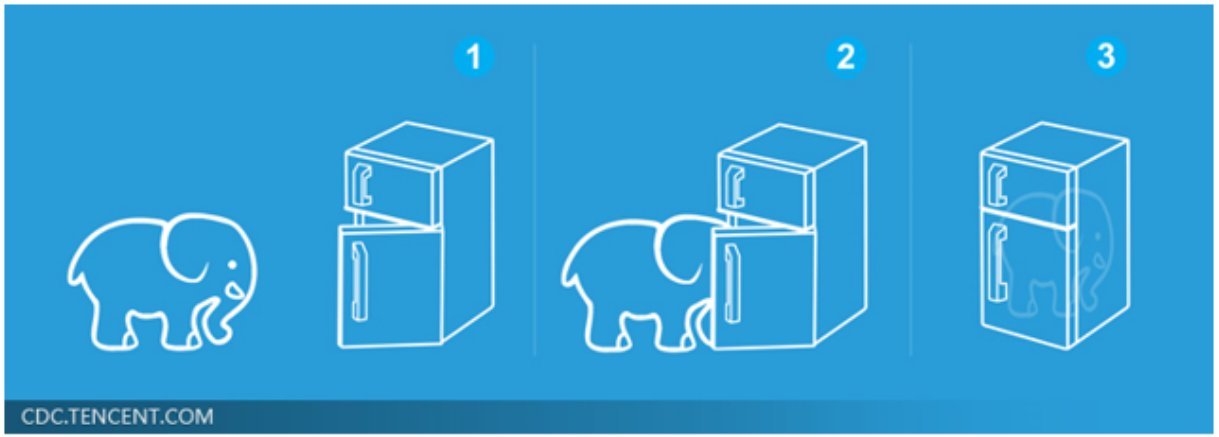
Three Steps for Deep Learning – Step 2

機器學習好簡單 .....

Step 0: What kind of function do you want to find?

Step 1: define a set of function → **Step 2: goodness of function** → Step 3: pick the best function

就好像把大象放進冰箱 .....

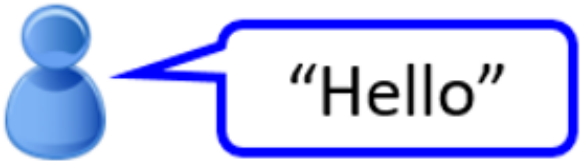


Supervised Learning

- 監督式學習

- Supervised

Learning from teacher



Say "Hi"

Say "Good bye"

x

Speech Recognition

"How are you"

y

Supervised

x_1:

x_2:

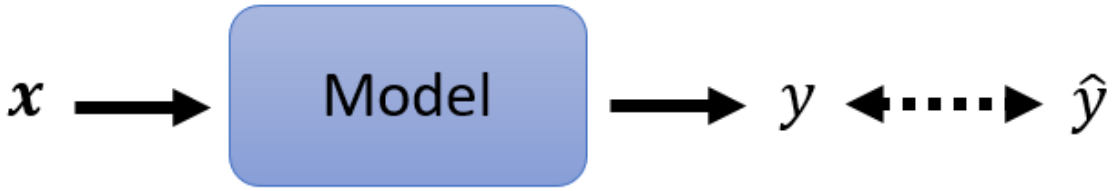
x_3:

y_1: Hello y_2: Good y_3: I am fine

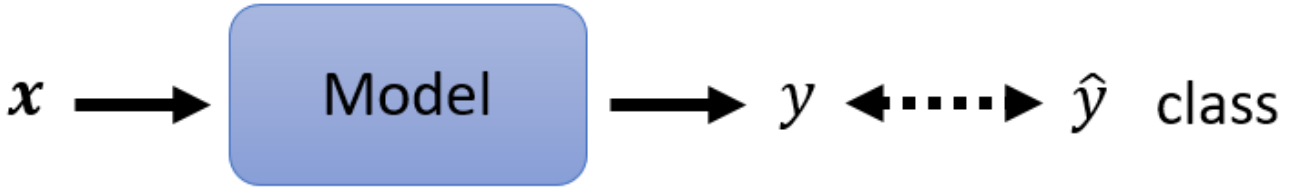
Example – Classification (1/11)

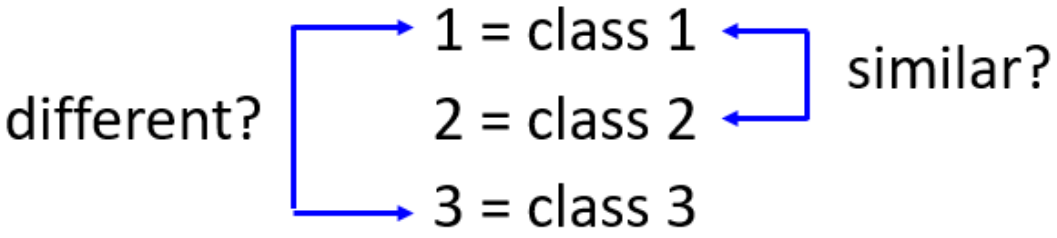
Classification as Regression?

- Regression



- Classification as regression?





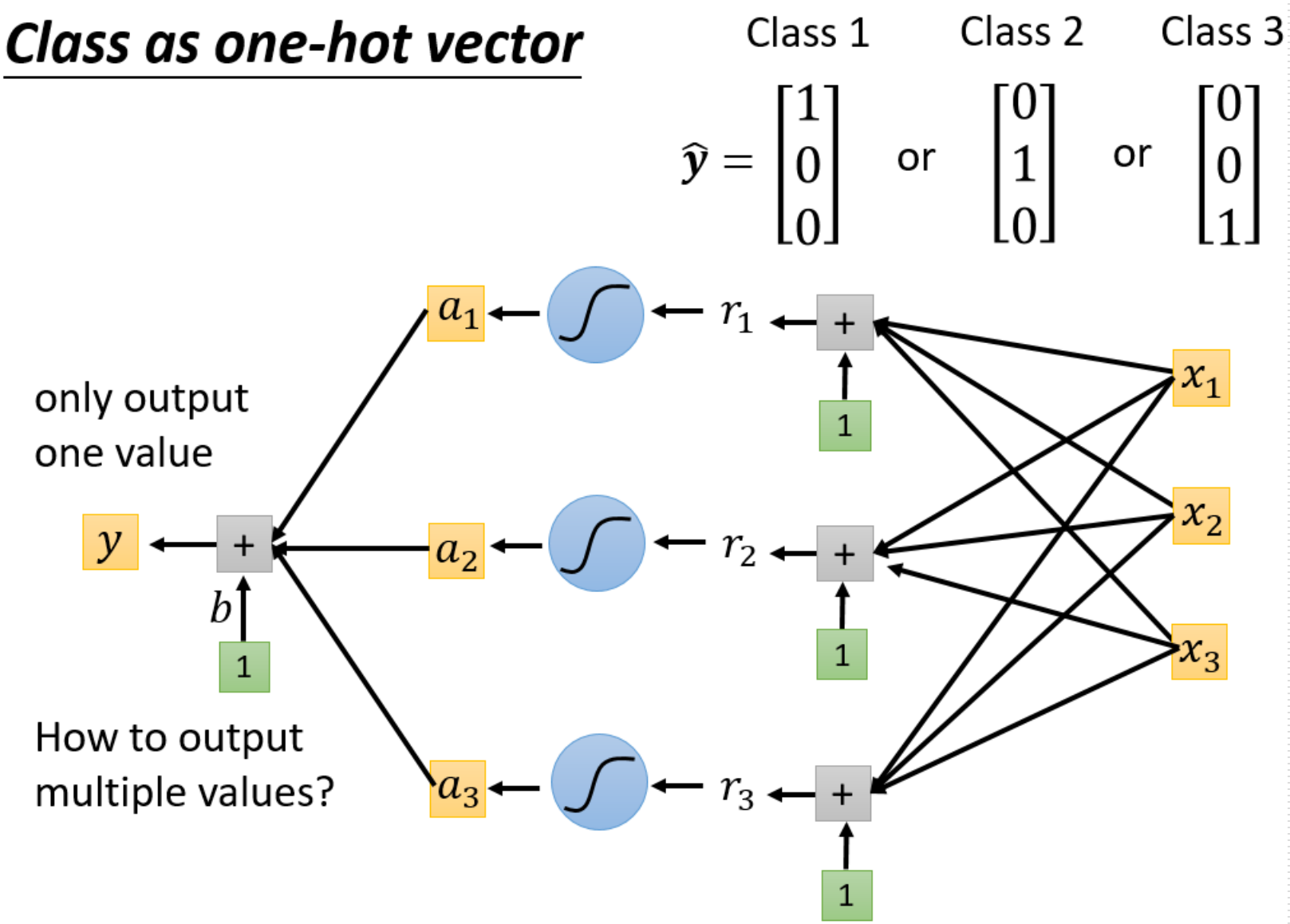
Example – Classification (2/11)

- One-hot Encoding

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Example – Classification (3/11)

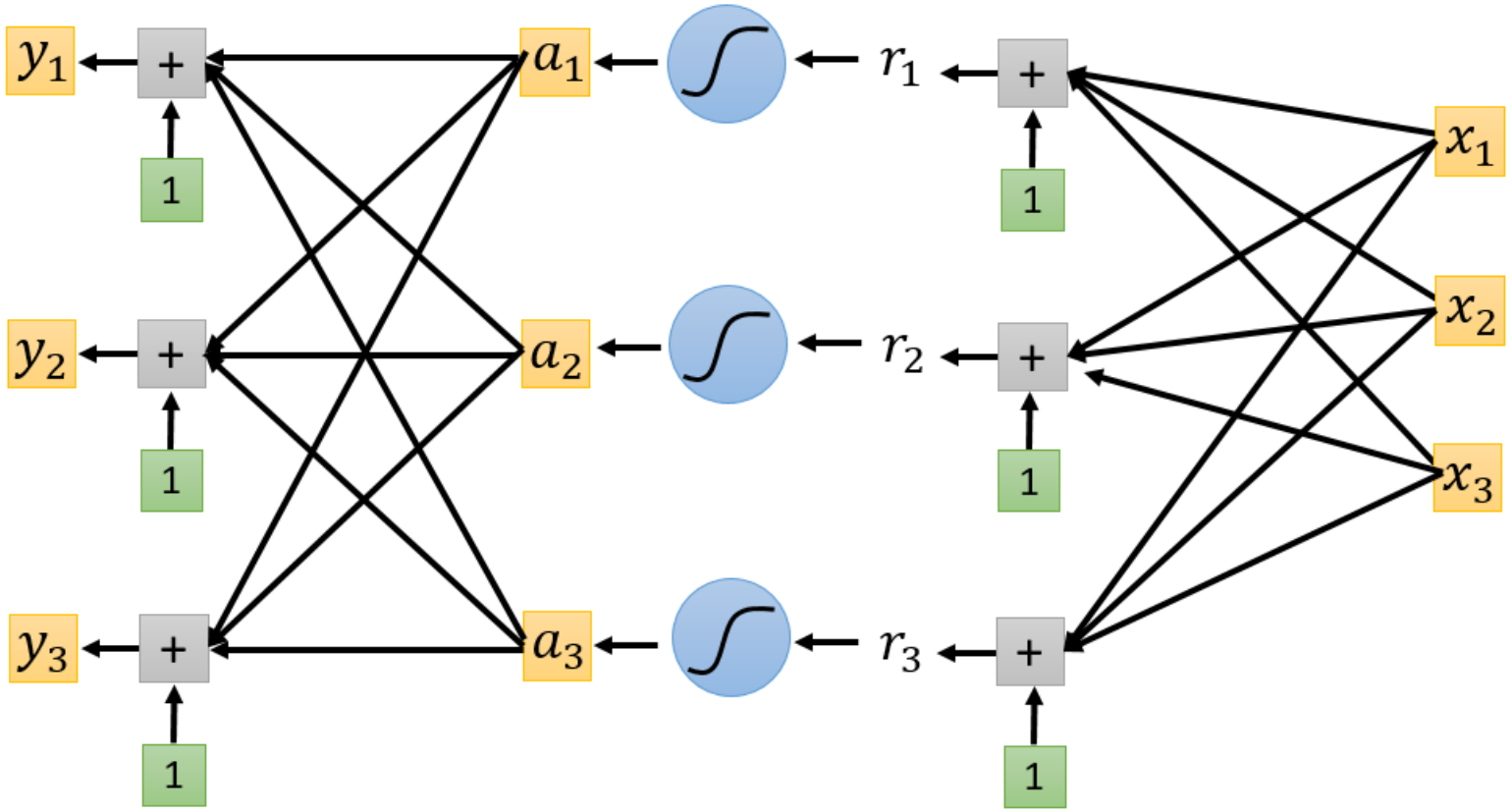


Example – Classification (4/11)

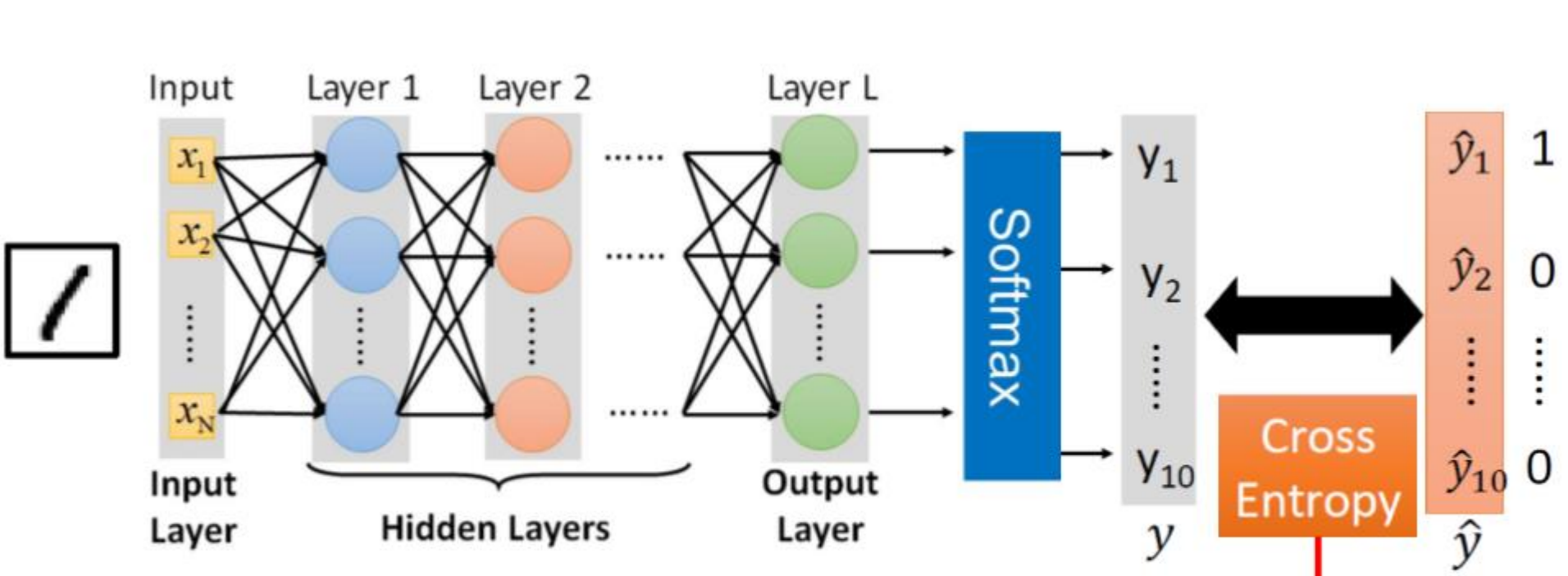
Class as one-hot vector

Class 1 Class 2 Class 3

$$\hat{y} = \begin{bmatrix}1\\0\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\1\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\0\\1\end{bmatrix}$$



Example – Classification (5/11)



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

Example – Classification (6/11)

Regression

label

feature

$$\hat{y} \longleftrightarrow y = b + c^T \sigma(b + W x)$$

Classification

feature

$$y = b' + W' \sigma(b + W x)$$

label

$$\hat{y} \longleftrightarrow y' = softmax(y)$$

0 or 1

Make all values between 0 and 1

Can have any value

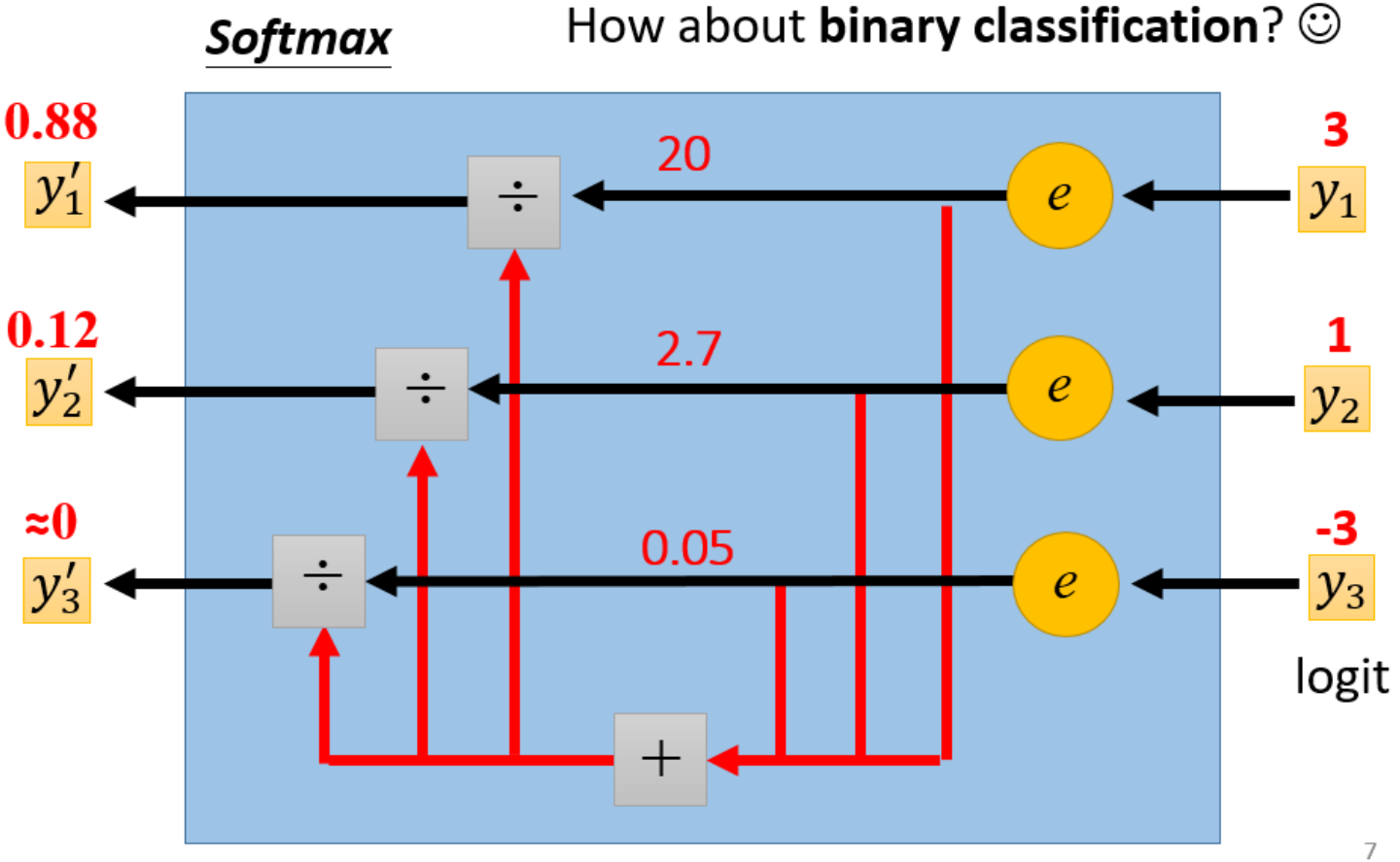
6

Example – Classification (7/11)

Soft-max $y_i' = \frac{exp(y_i)}{\sum_j exp(y_i)}$

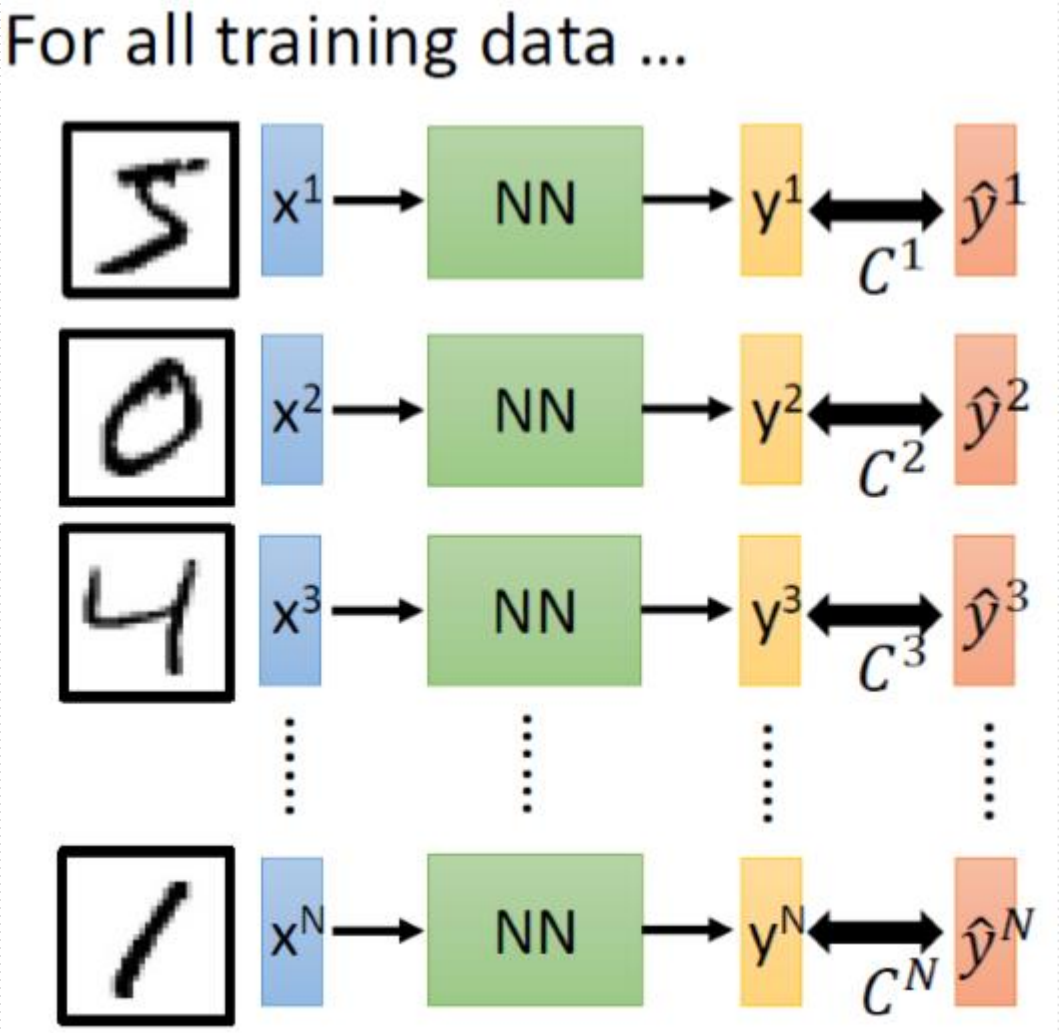
- $1 > y_i' > 0$
- $\sum_i y_i' = 1$

How about **binary classification**? ☺



7

Example – Classification (8/11)



Total Loss:

$$L = \sum_{n=1}^{N} C^n$$

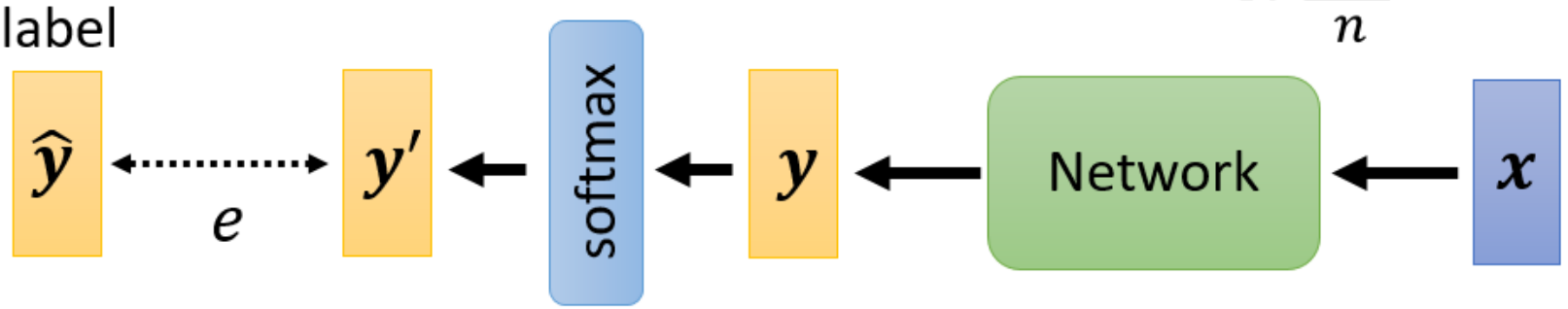
Find ***a function in function set*** that minimizes total loss L

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

Example – Classification (9/11)

Loss of Classification

$$L = \frac{1}{N}\sum_{n} e_n$$



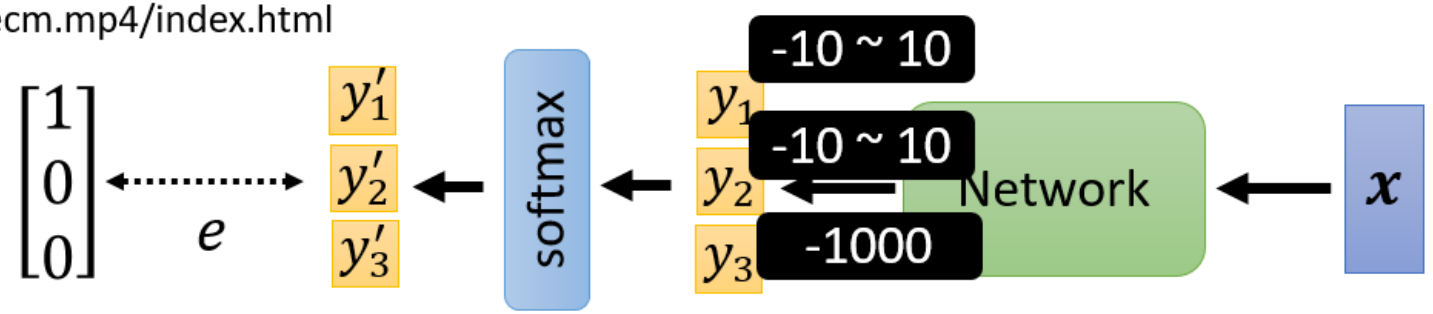
Mean Square Error (MSE) $e = \sum_{i} (\hat{y}_i - y'_i)^2$

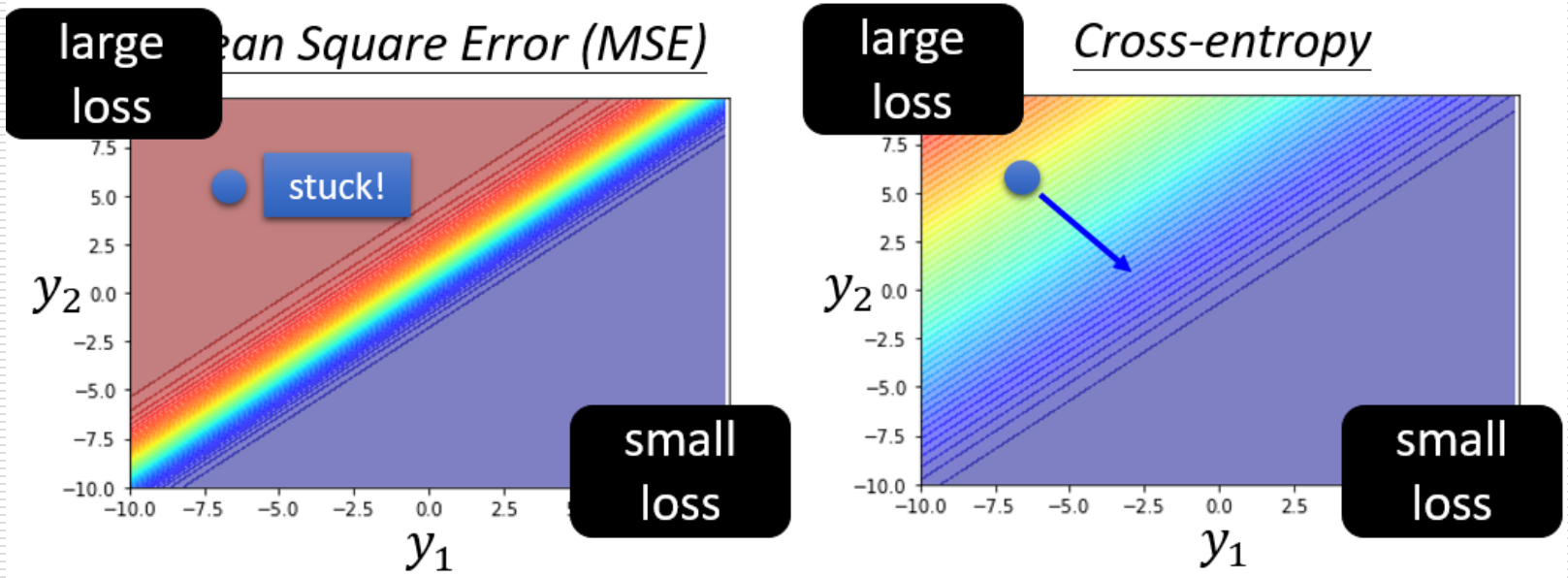
Cross-entropy WIN $e = -\sum_{i} \hat{y}_i ln y'_i$

Minimizing cross-entropy is equivalent to **maximizing likelihood**.

Example – Classification (10/11)

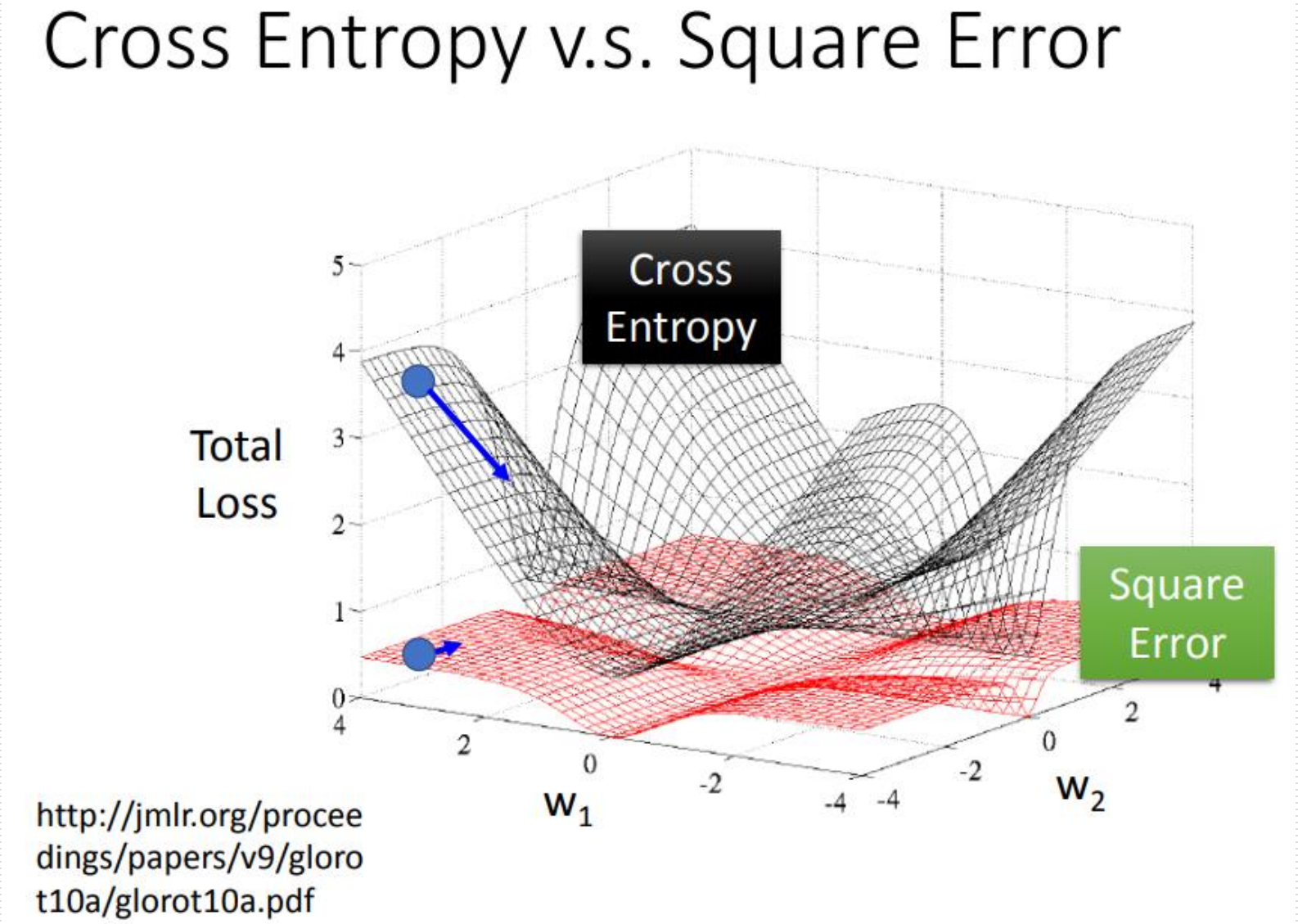
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Deep%20More%20(v2).ecm.mp4/index.html





Changing the loss function can change the difficulty of optimization.

Example – Classification (11/11)



Three Steps for Deep Learning – Step 3

機器學習好簡單 .....

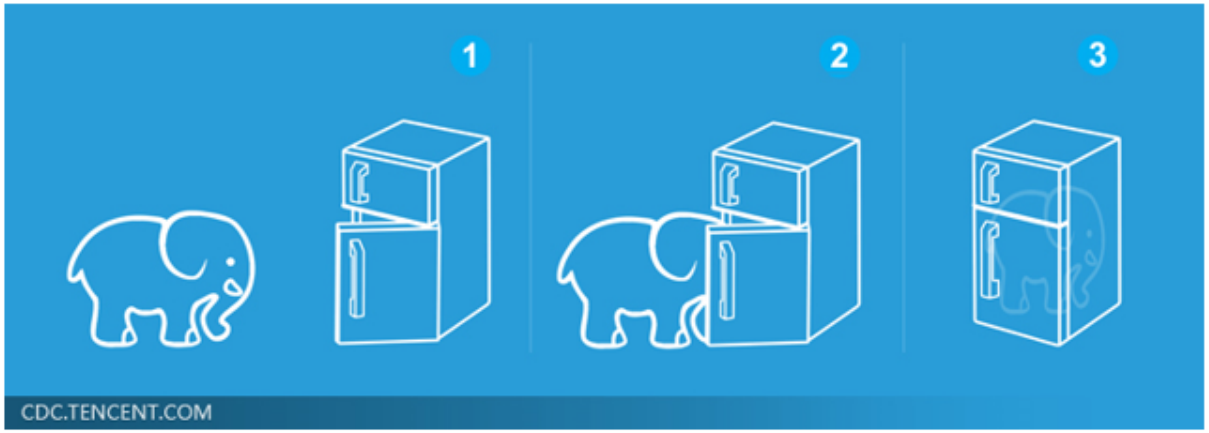
Step 0: What kind of function do you want to find?

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

就好像把大象放進冰箱 .....



Pick the Best Function (1/2)

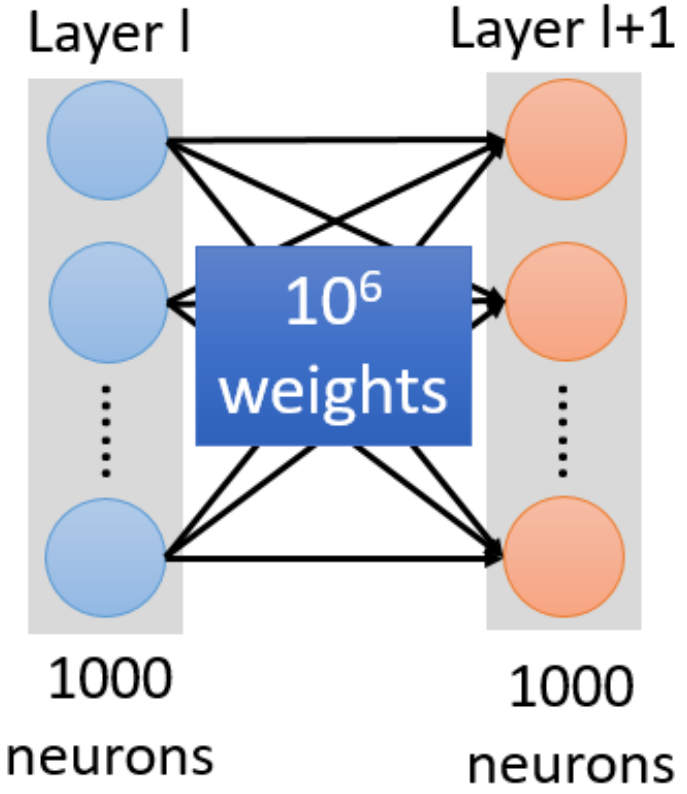
如何找出最好的 function ?

Enumerate all possible values

Network parameters $\theta = \{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}$

Millions of parameters

Today a network can have more than 100M parameters.



Pick the Best Function (2/2)

Gradient Descent

Backward propagation

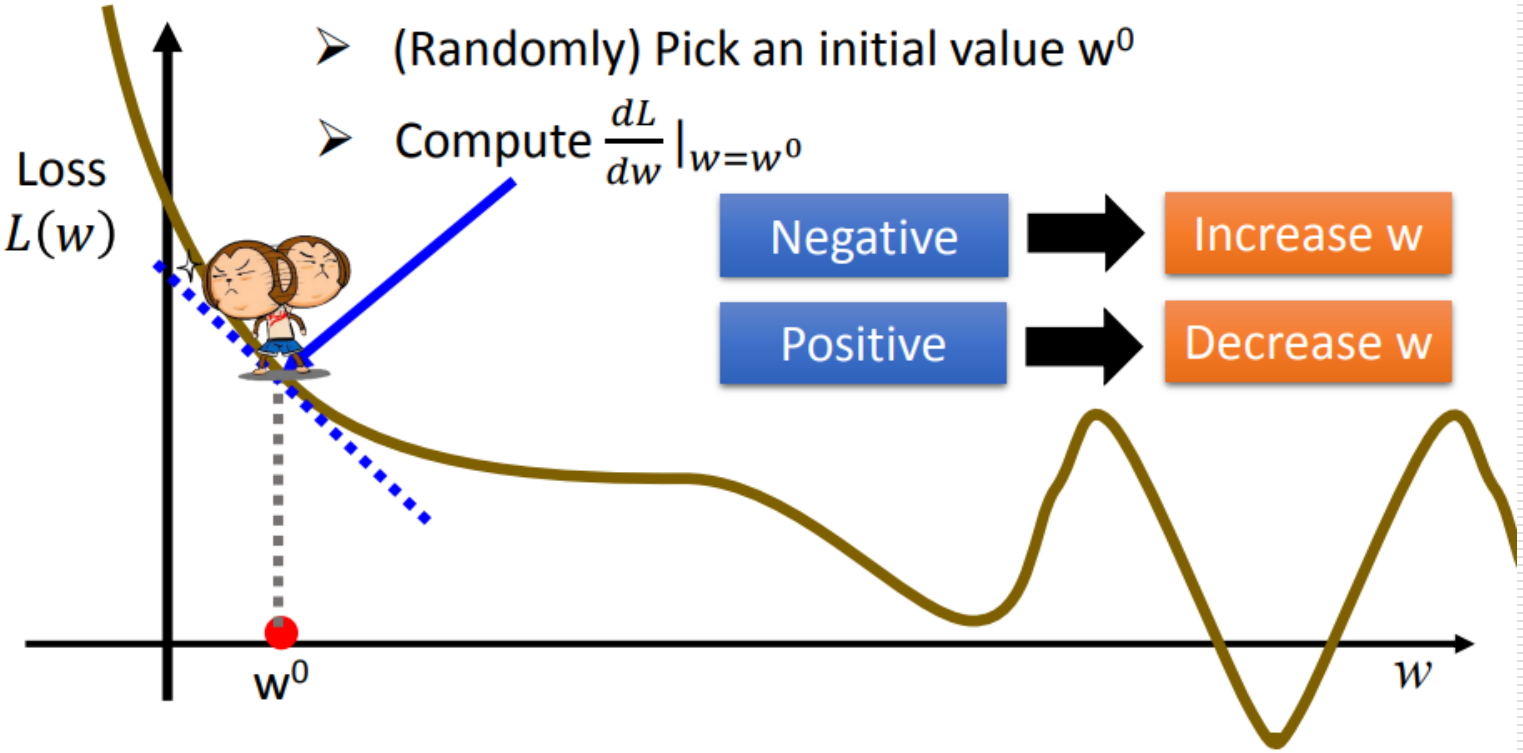
Backward Propagation (1/5)

http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:
  - (Randomly) Pick an initial value w^0
  - Compute $\frac{dL}{dw}|_{w=w^0}$



Backward Propagation (2/5)

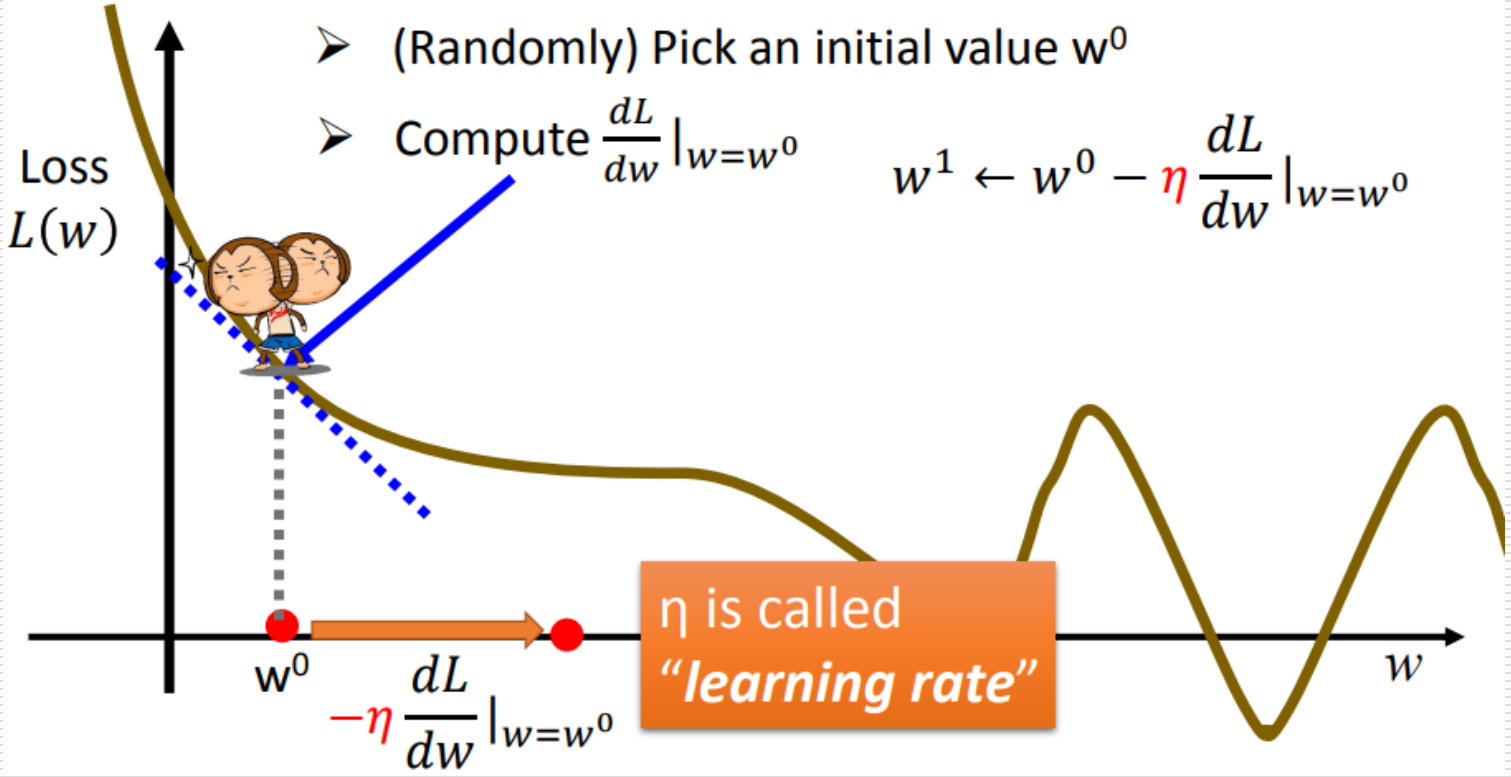
http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:
  - (Randomly) Pick an initial value w^0
  - Compute $\frac{dL}{dw}|_{w=w^0}$

$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$$

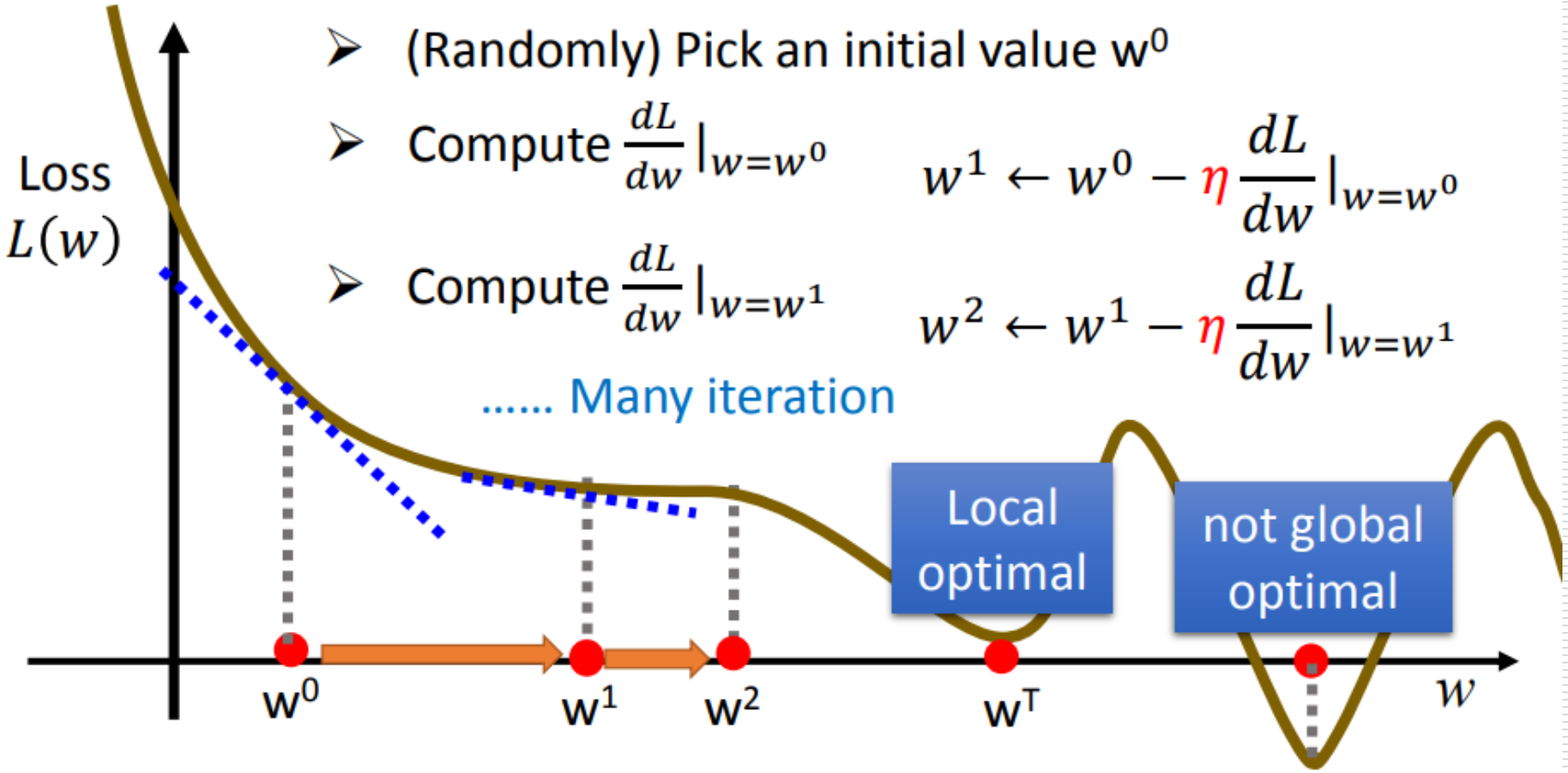


Backward Propagation (3/5)

Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:



Backward Propagation (4/5)

Step 3: Gradient Descent

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$ gradient

- How about two parameters? $w^*, b^* = arg \min_{w,b} L(w, b)$
  - (Randomly) Pick an initial value w^0, b^0
  - Compute $\frac{\partial L}{\partial w}|_{w=w^0,b=b^0}, \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$

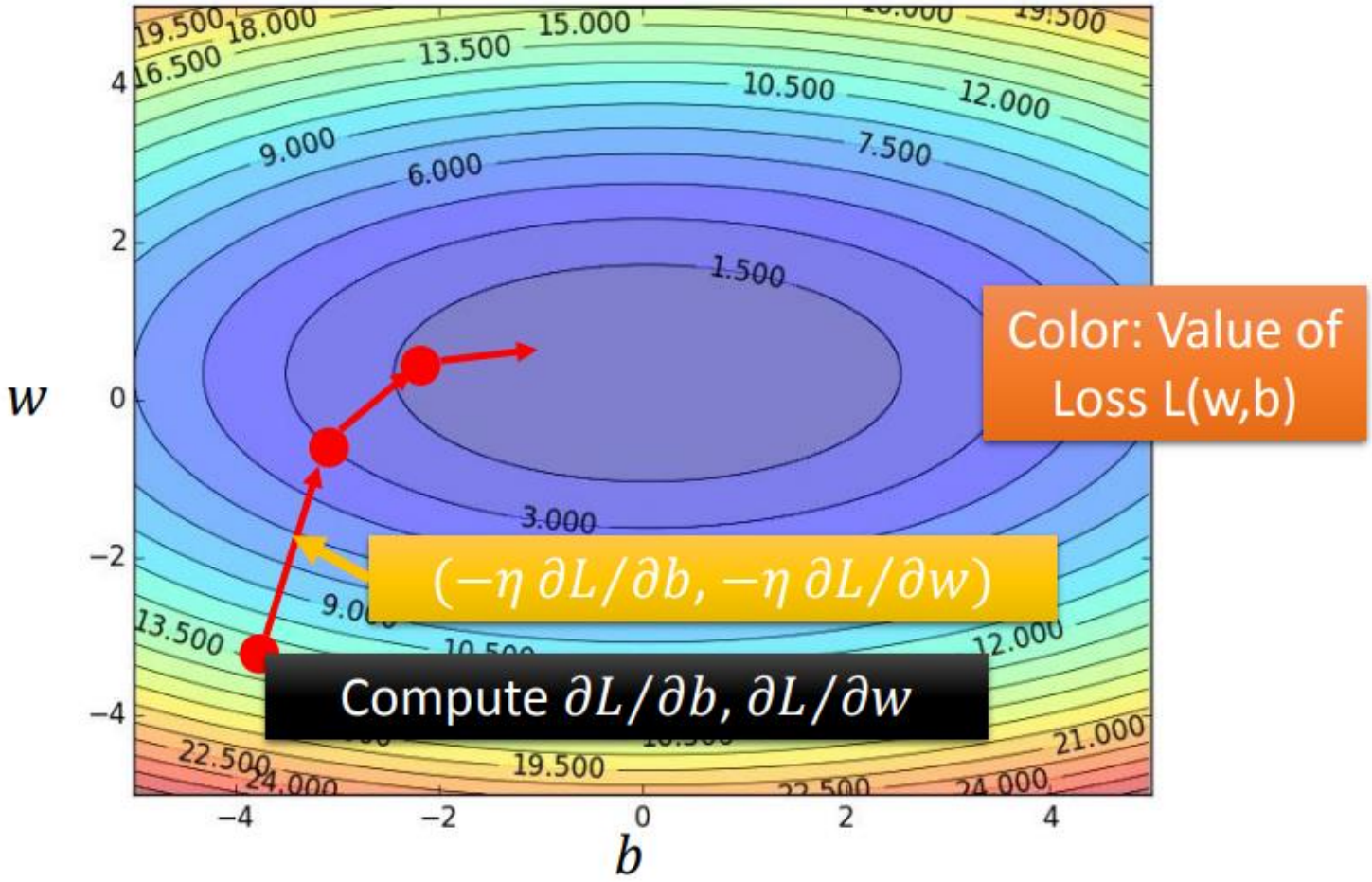
$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0,b=b^0} \qquad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$

  - Compute $\frac{\partial L}{\partial w}|_{w=w^1,b=b^1}, \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$

$$w^2 \leftarrow w^1 - \eta \frac{\partial L}{\partial w}|_{w=w^1,b=b^1} \qquad b^2 \leftarrow b^1 - \eta \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$$

Backward Propagation (5/5)

Step 3: Gradient Descent



Batch and Epoch

Batch and Epoch (1/3)

Review: Optimization with Batch

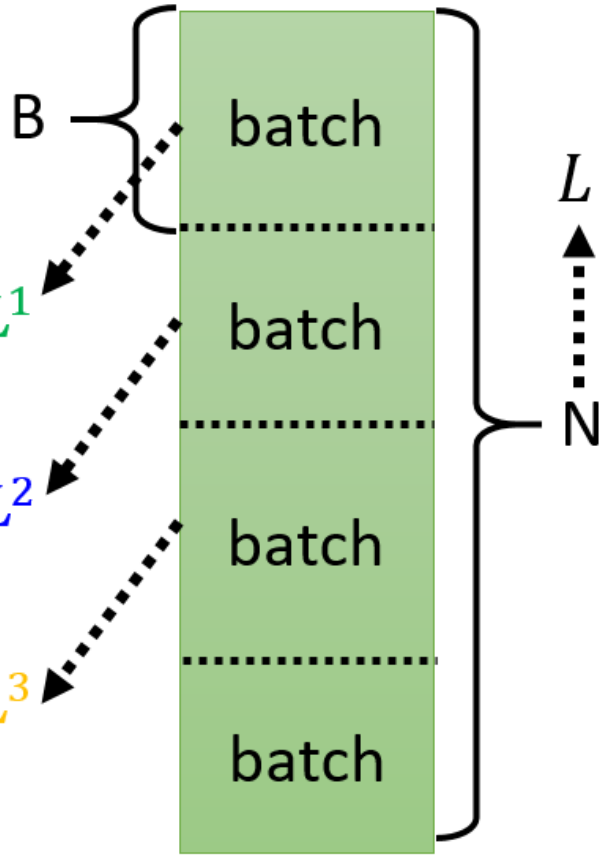
$$\boldsymbol{\theta}^* = arg\min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g^0} = \nabla L^1(\boldsymbol{\theta}^0)$ L^1

  update $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g^0}$
- Compute gradient $\boldsymbol{g^1} = \nabla L^2(\boldsymbol{\theta}^1)$ L^2

  update $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g^1}$
- Compute gradient $\boldsymbol{g^3} = \nabla L^3(\boldsymbol{\theta}^2)$ L^3

  update $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g^3}$



1 **epoch** = see all the batches once → **Shuffle** after each epoch

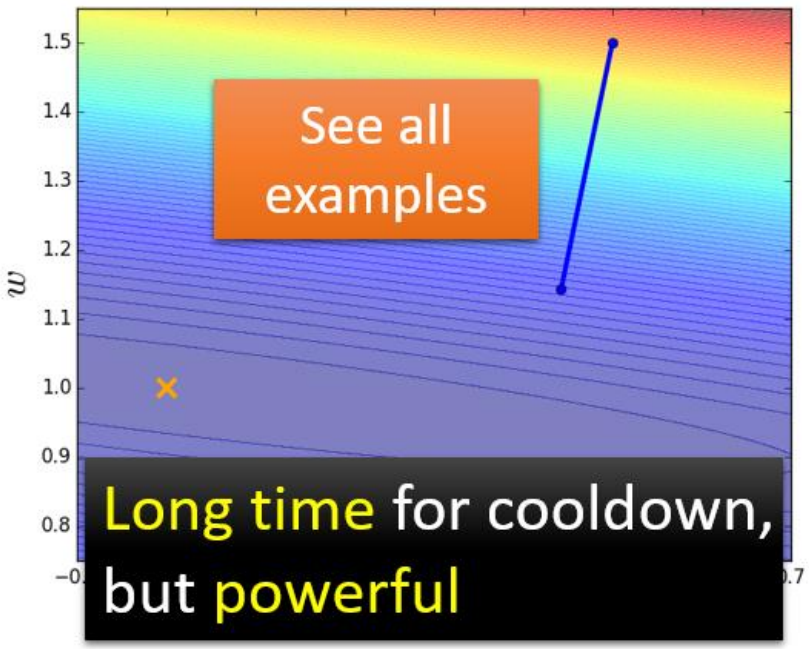
Batch and Epoch (2/3)

Small Batch v.s. Large Batch

Consider 20 examples (N=20)

Batch size = N (Full batch)

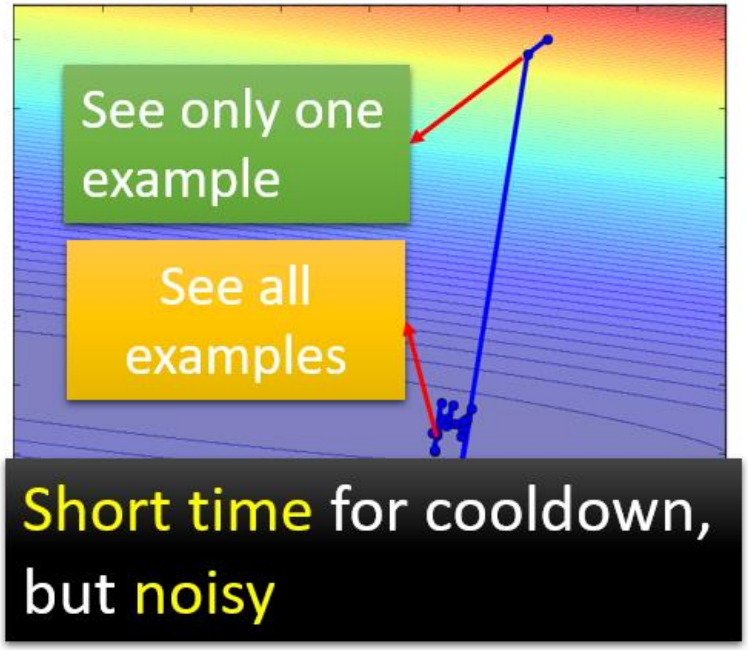
Update after seeing all the 20 examples



Batch size = 1

Update for each example

Update 20 times in an epoch

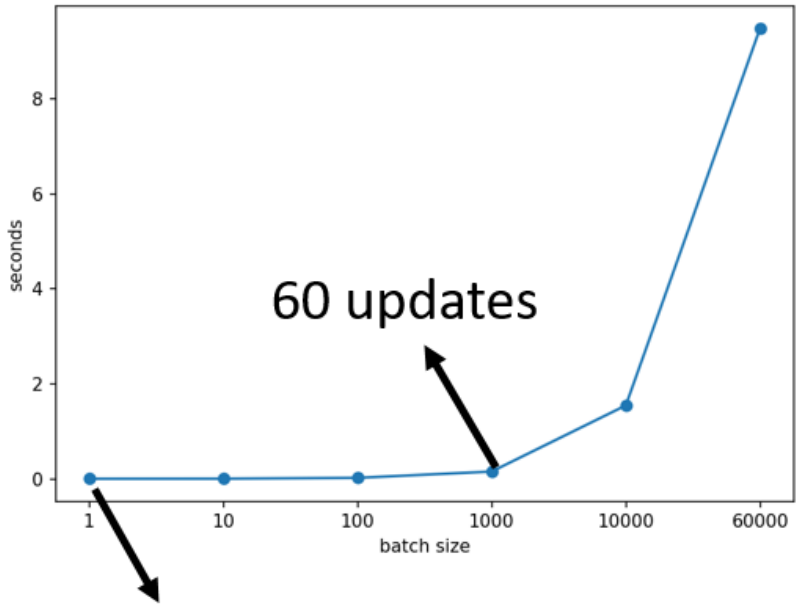


Batch and Epoch (3/3)

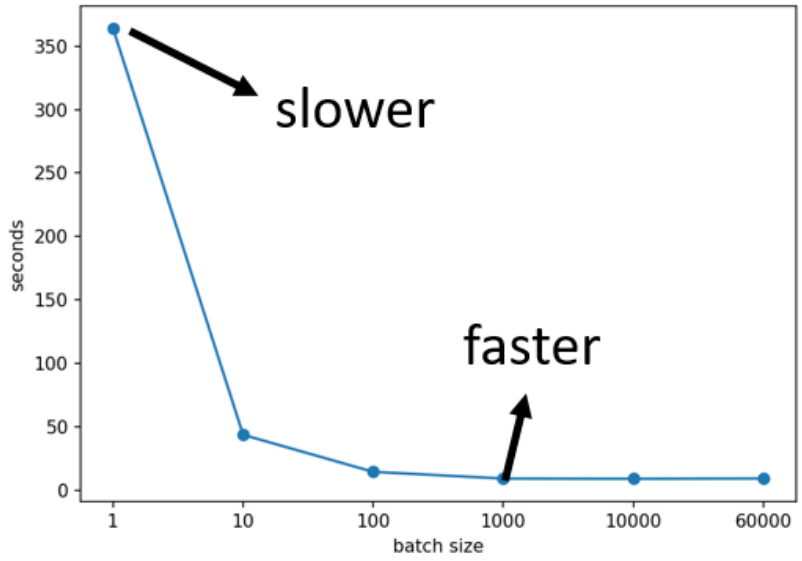
Small Batch v.s. Large Batch

- Smaller batch requires longer time for one epoch (longer time for seeing all data once)

Time for one **update**



Time for one **epoch**



Overfitting and validation

Overfitting (1/11)

How's the results?

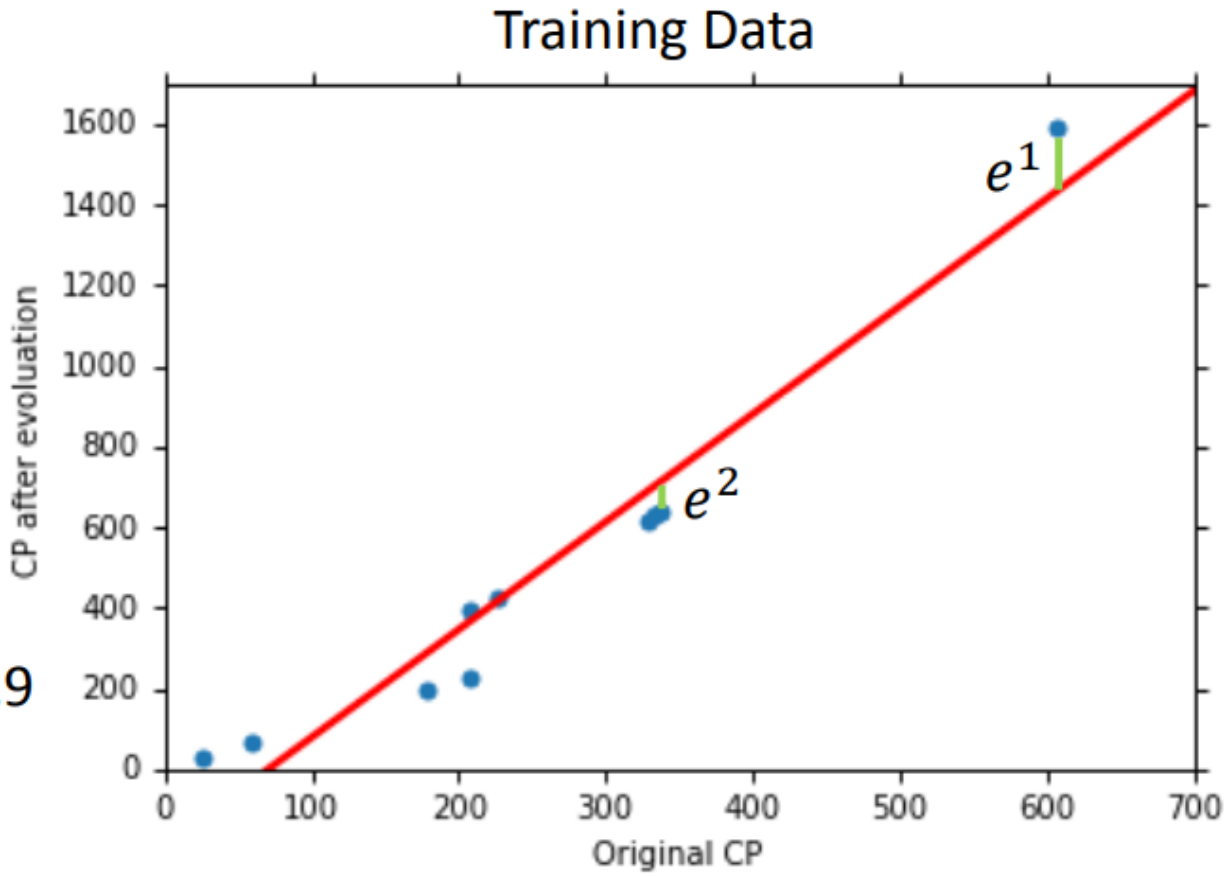
$y = b + w \cdot x_{cp}$

b = -188.4

w = 2.7

Average Error on Training Data

$$= \frac{1}{10}\sum_{n=1}^{10} e^n = 31.9$$



Overfitting (2/11)

How's the results? - Generalization

What we really care about is the error on new data (testing data)

$y = b + w \cdot x_{cp}$

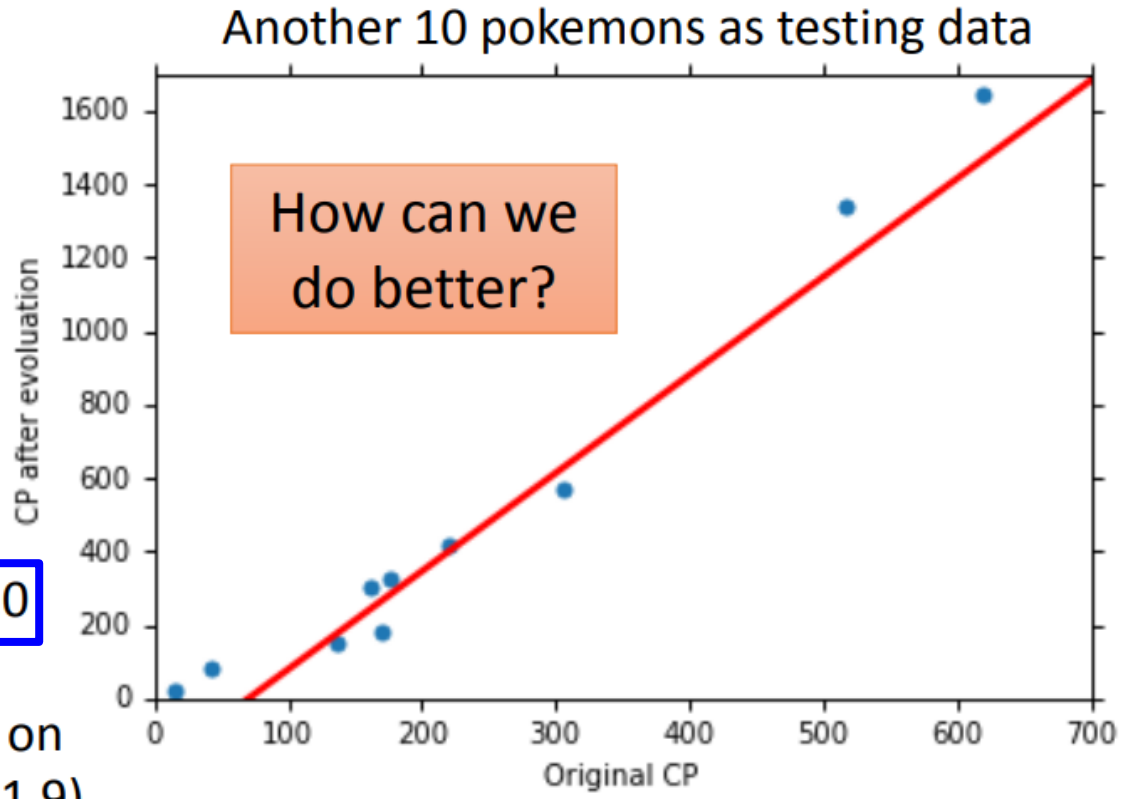
b = -188.4

w = 2.7

Average Error on Testing Data

$$= \frac{1}{10}\sum_{n=1}^{10} e^n = 35.0$$

> Average Error on Training Data (31.9)



Overfitting (3/11)

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$$

Best Function

b = -10.3

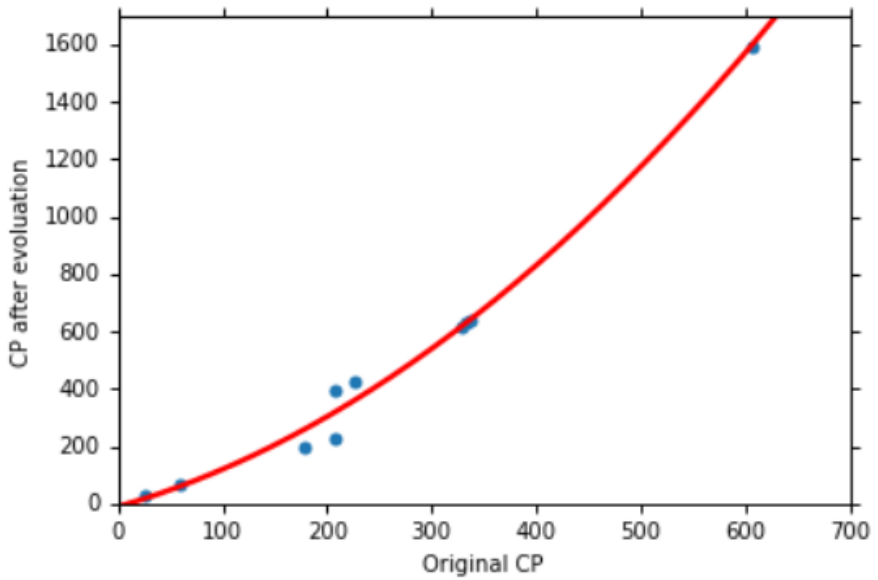
$w_1 = 1.0$, $w_2 = 2.7 \times 10^{-3}$

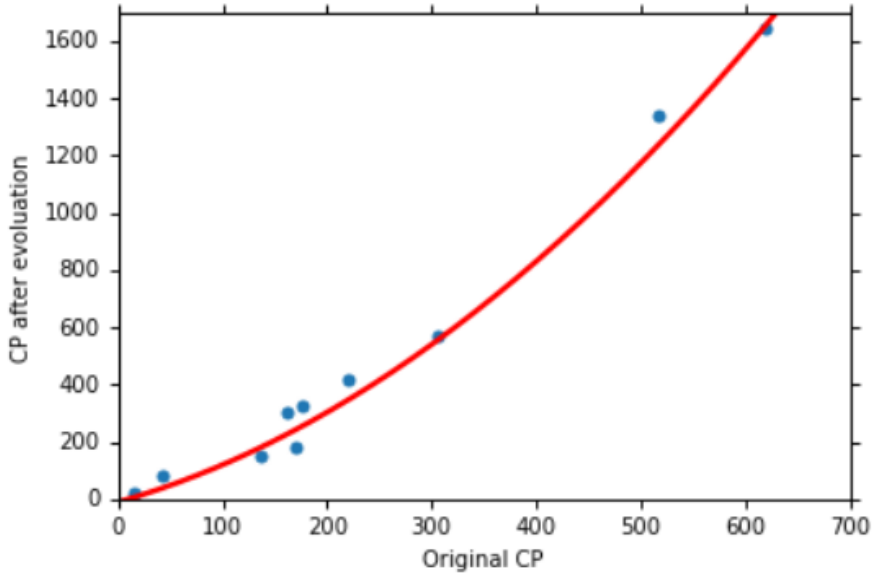
Average Error = 15.4

Testing:

Average Error = 18.4

Better! Could it be even better?





Overfitting (4/11)

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$$

Best Function

$b = 6.4$, $w_1 = 0.66$

$w_2 = 4.3 \times 10^{-3}$

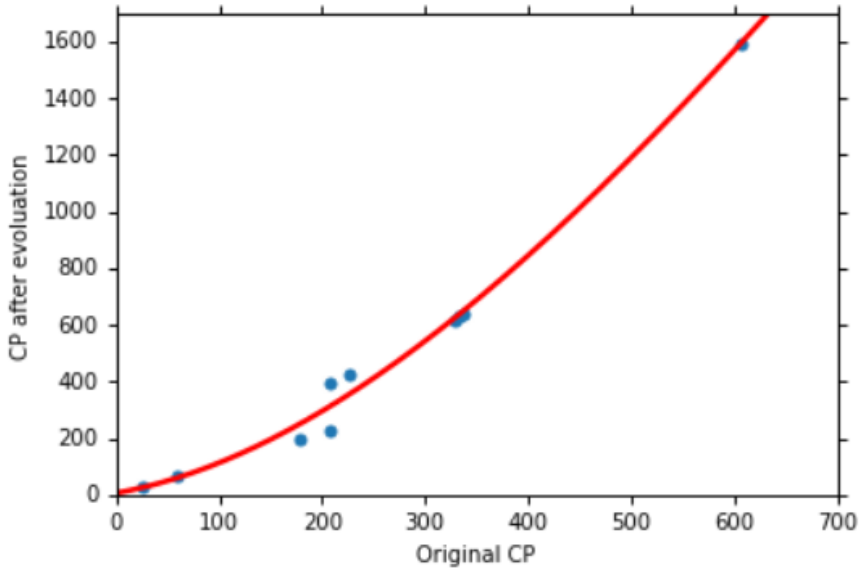
$w_3 = -1.8 \times 10^{-6}$

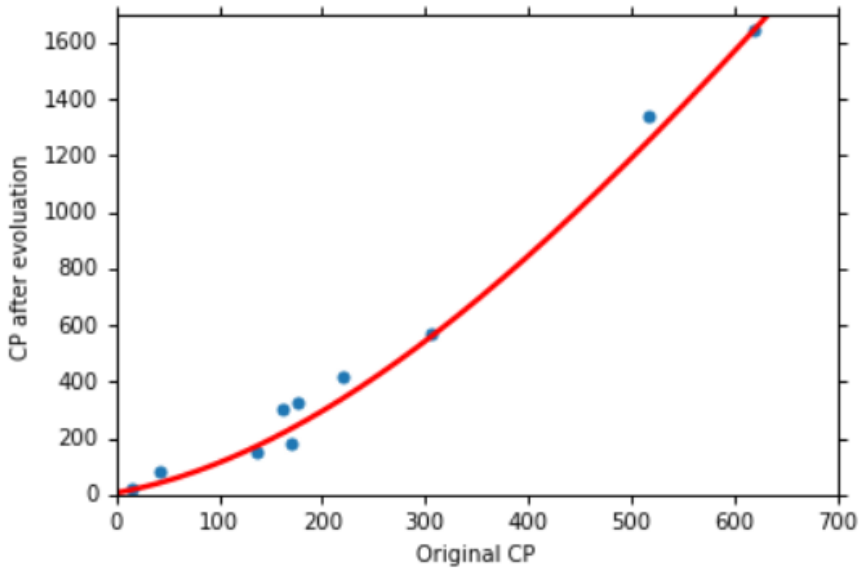
Average Error = 15.3

Testing:

Average Error = 18.1

Slightly better.
How about more complex model?





Overfitting (5/11)

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$$

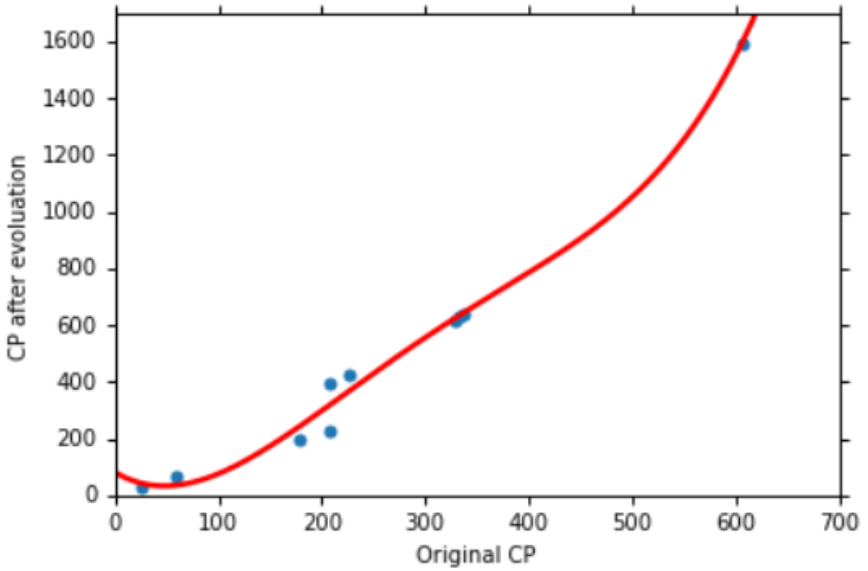
Best Function

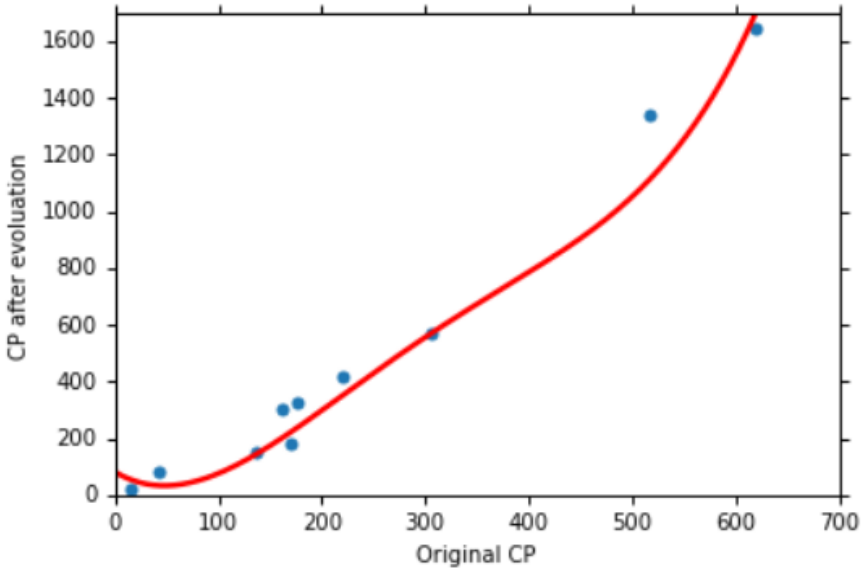
Average Error = 14.9

Testing:

Average Error = 28.8

The results become worse ...

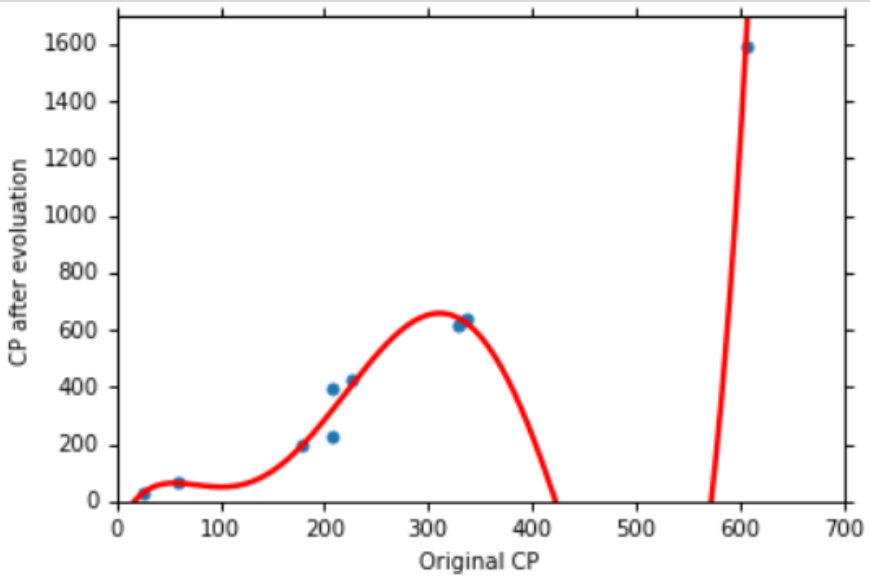




Overfitting (6/11)

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$$



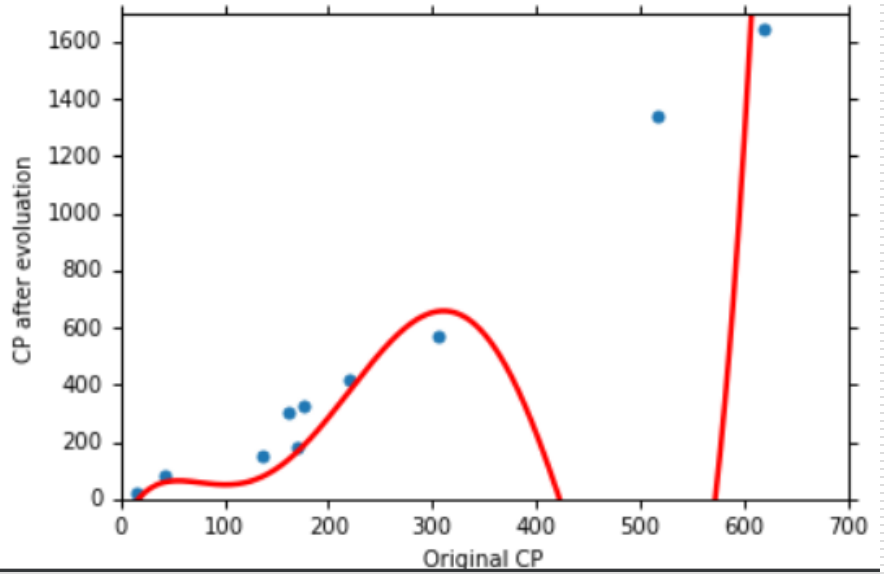
Best Function

Average Error = 12.8

Testing:

Average Error = 232.1

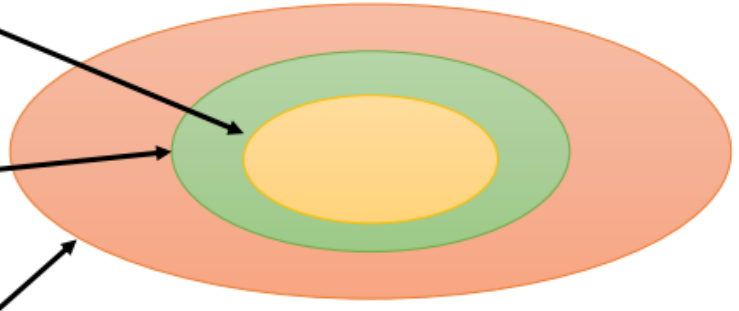
The results are so bad.



Overfitting (7/11)

Model Selection

1. $y = b + w \cdot x_{cp}$
2. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$
3. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$
4. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$
5. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$

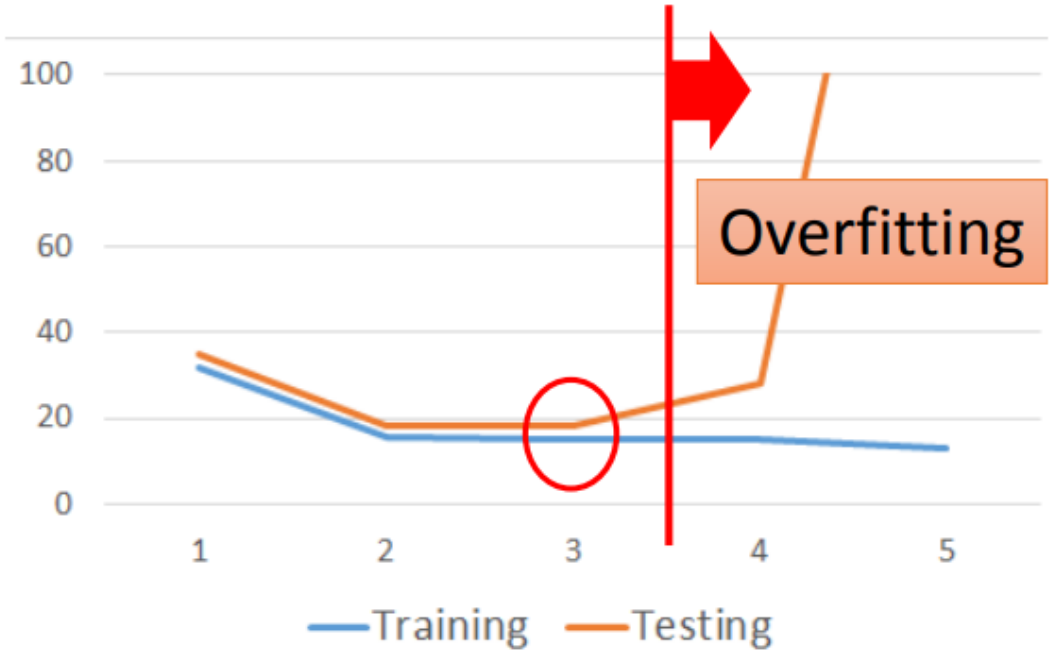


A more complex model yields lower error on training data.

If we can truly find the best function

Overfitting (8/11)

Model Selection

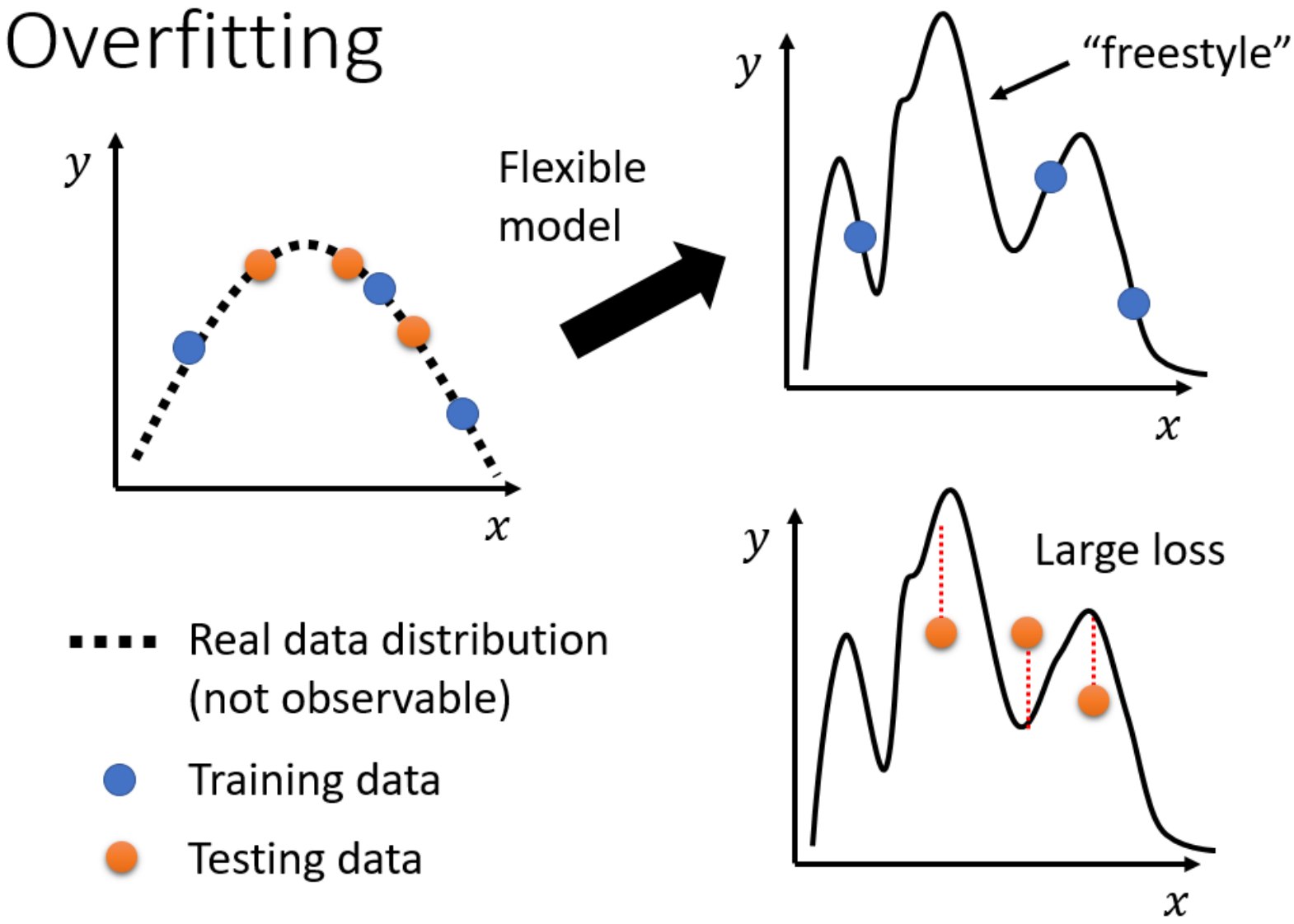


	Training	Testing
1	31.9	35.0
2	15.4	18.4
3	15.3	18.1
4	14.9	28.2
5	12.8	232.1

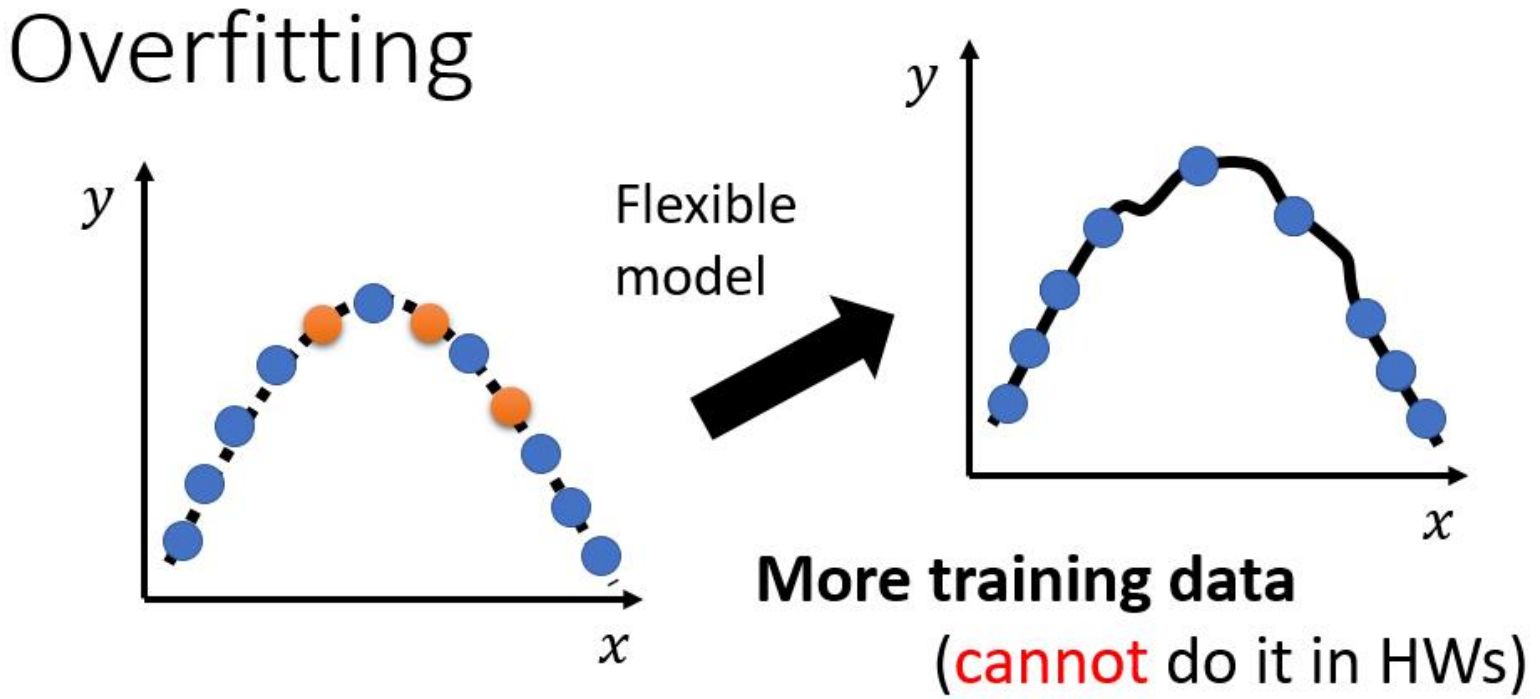
A more complex model does not always lead to better performance on ***testing data***.

This is ***Overfitting***. ➡ Select suitable model

Overfitting (9/11)



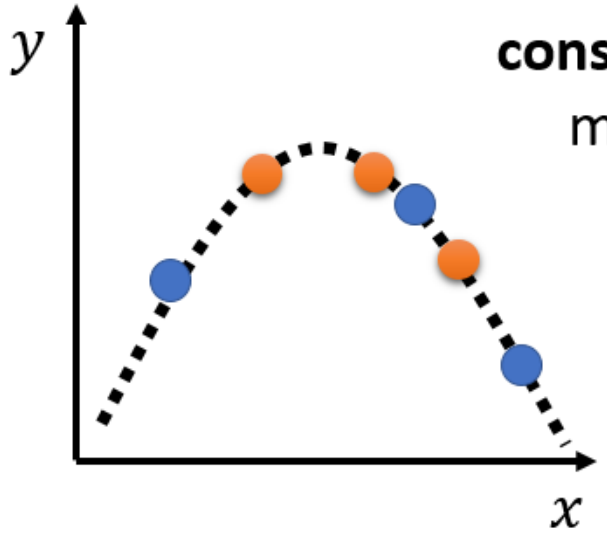
Overfitting (10/11)



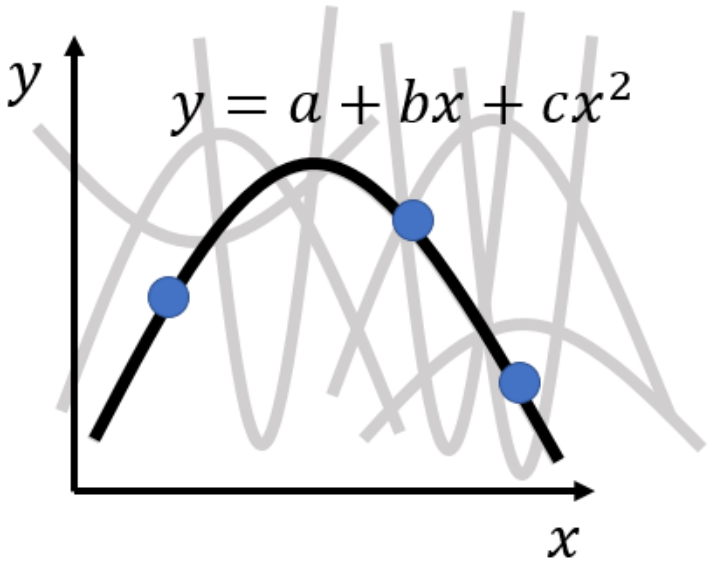
Data augmentation (you can do that in HWs)

Overfitting (11/11)

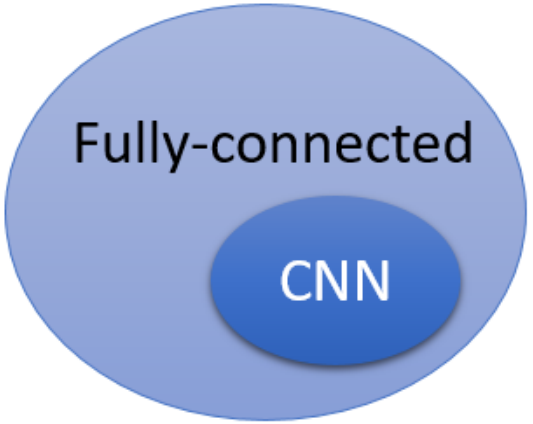
Overfitting



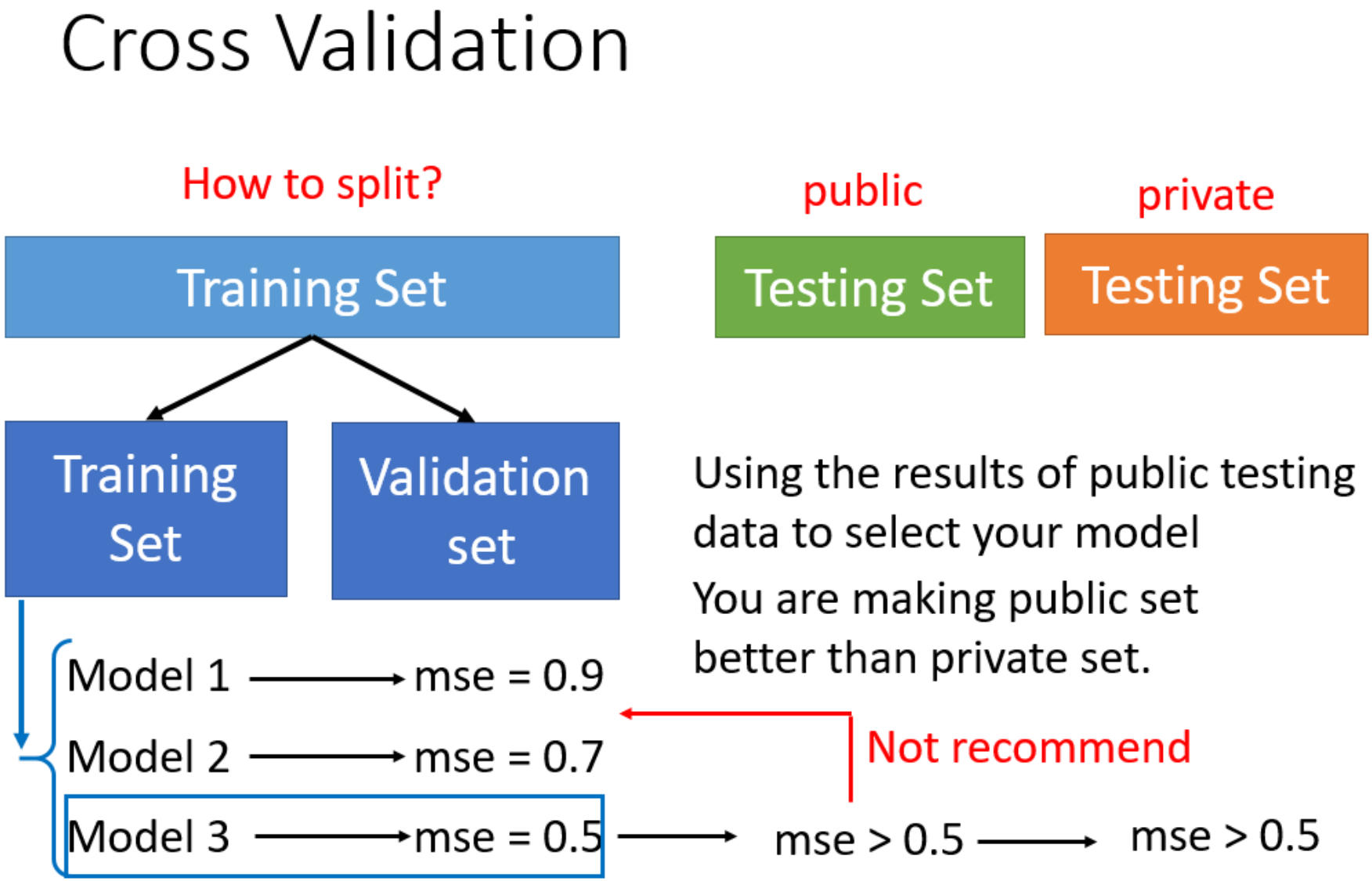
constrained model



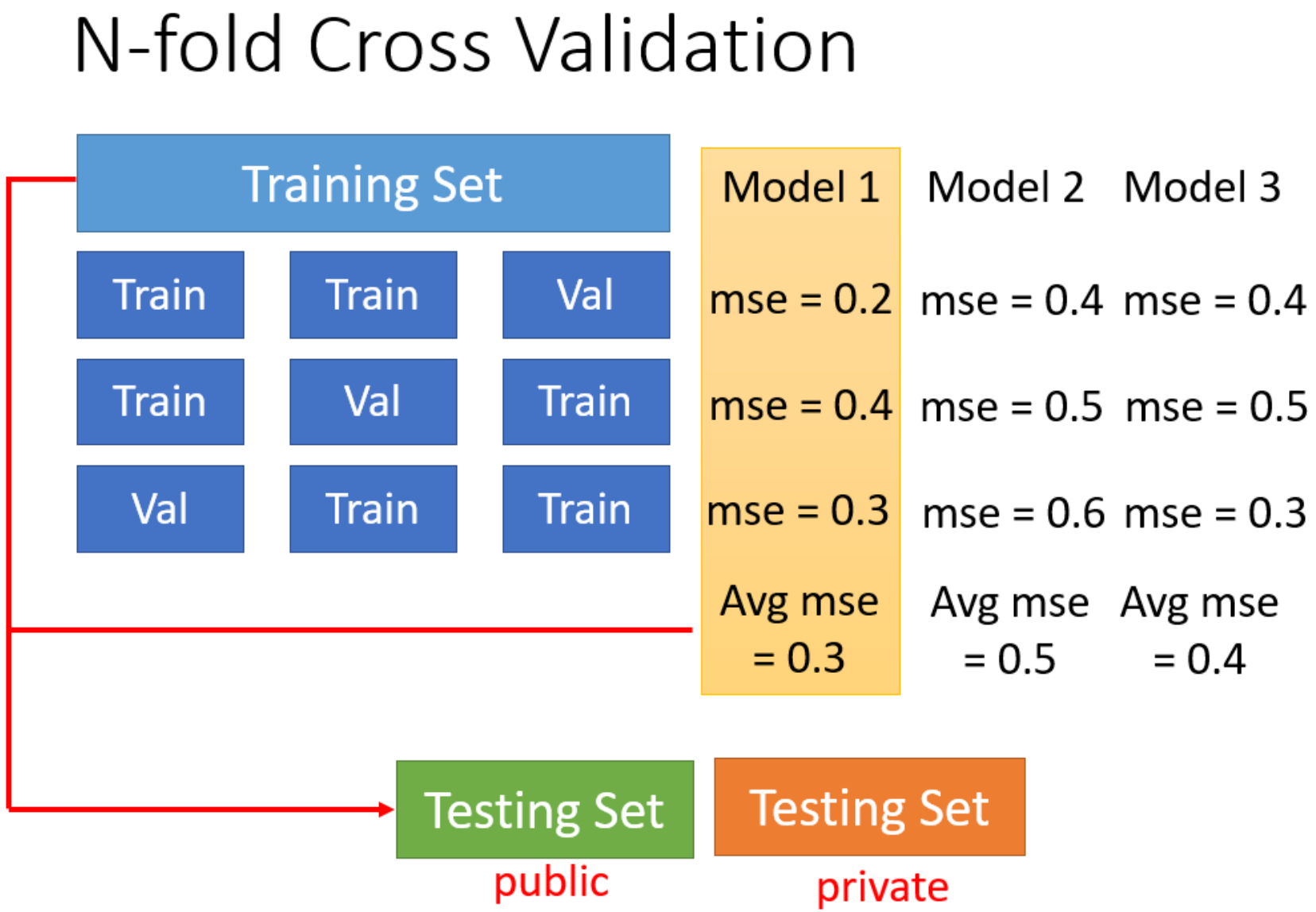
- Less parameters, sharing parameters
- Less features
- Early stopping
- Regularization
- Dropout



Validation (1/2)



Validation (2/2)



References

- 李宏毅教授 Machine Learning Online Course
  - http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML17_2.html
  - https://www.youtube.com/playlist?list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49
- 從 ReLU 到 GeLU, 一文概覽神經網路的激活函數
  - https://zhuanlan.zhihu.com/p/98863801
- 常用激活函數(relu,glu,gelu,swish等)
  - https://blog.csdn.net/weixin_39529413/article/details/123071764

Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

AI Training Course Series

Convolutional Neural Networks (CNNs)

Lecture 2-2

Student: Chih-Yao Liang
Advisor: Juinn-Dar Huang, Ph.D.

July 11, 2024

Outline (1/2)

- **Why CNNs?**
- **Details of Convolution**
  - Algorithm
  - Padding
  - Stride
  - Dilation
  - # of parameters
- **Common Layers in a CNN**
  - Pooling
  - Batch normalization
  - Connection method
  - Activation function

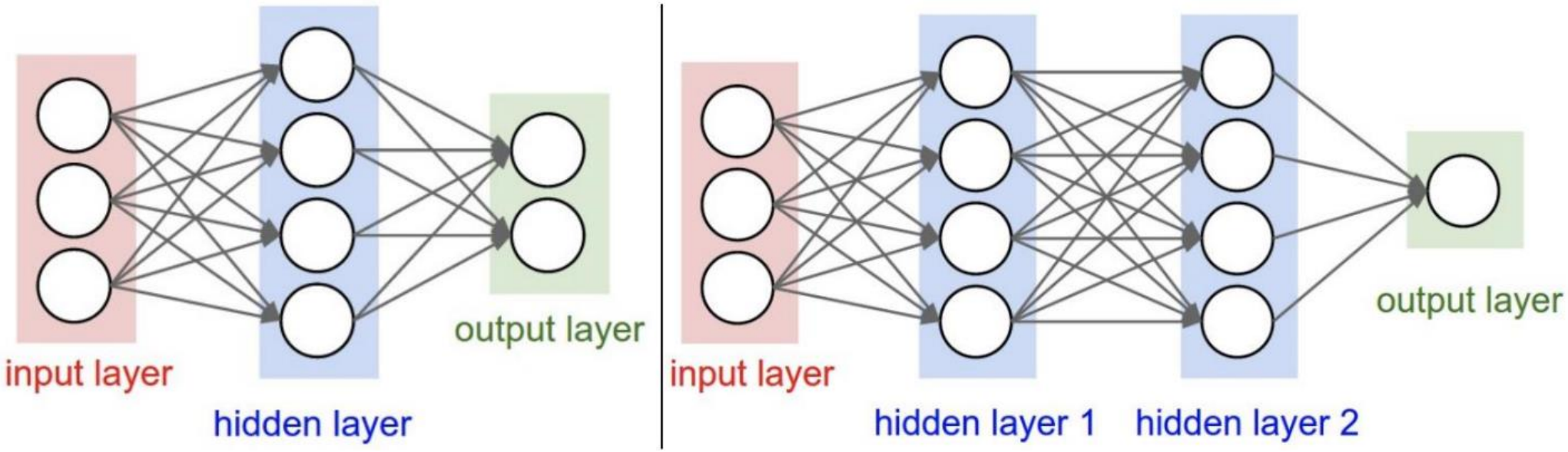
Outline (2/2)

- **Various Convolution Styles**
  - 1x1 convolution
  - Depth-wise separable convolution
  - Transpose convolution
- **Homework**
  - HW1
  - HW2
  - Submission
- **Reference**

Why CNNs?

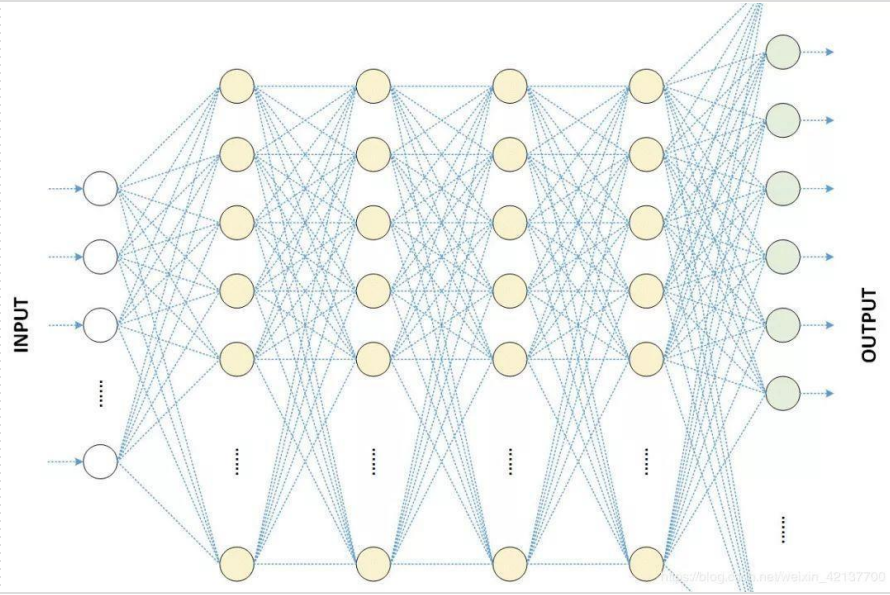
Recap on MLP

- Basic MLP architecture



The Problem with MLP (1/2)

- Add more layers to accomplish complex tasks
  - larger number of parameters ➔ storage problem & back propagation computation
  - need high quality training dataset ➔ expensive (otherwise overfitting)

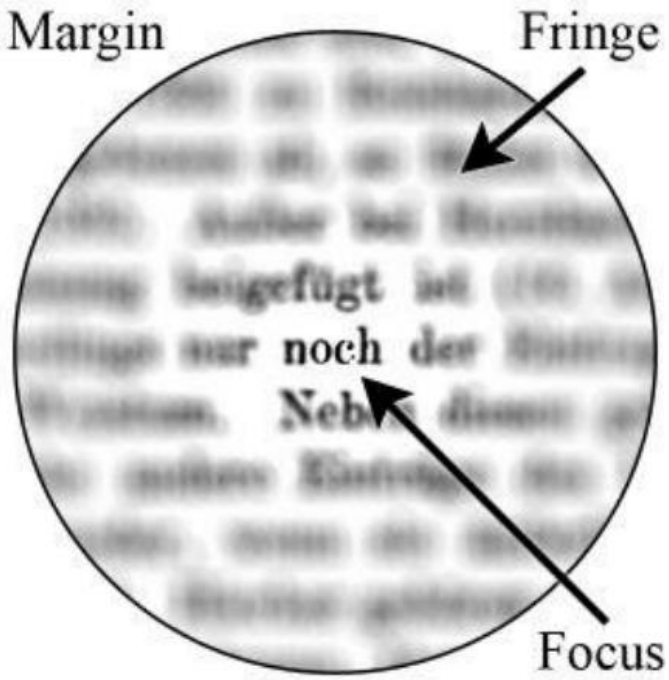


The Problem with MLP (2/2)

- Some tasks include spatial coherence (e.g., image related)
  - an output of an MLP neuron is determined by all input neurons ➔ NO spatial locality
  - thus MLPs may not be the best solution

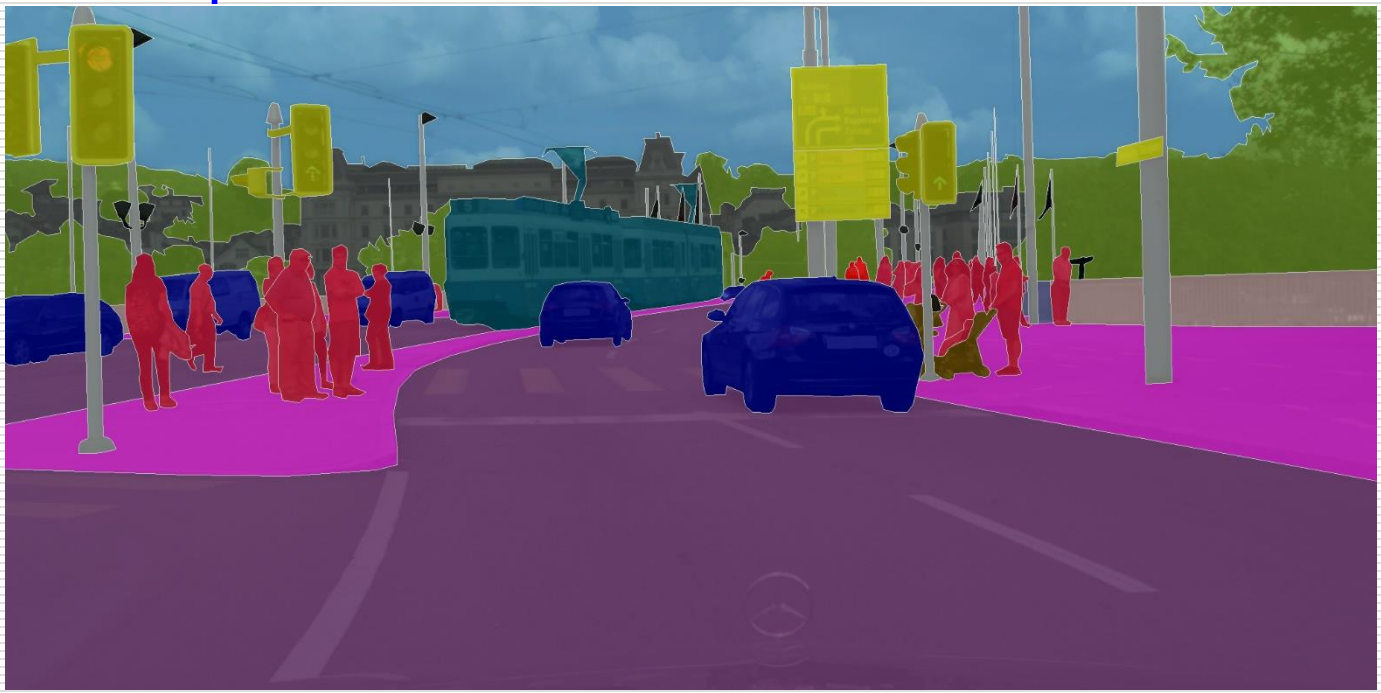
Local Receptive Field

- Neuroscientists have studied visual attention as an important cognitive process
- Humans select a subset of available information upon which to focus for enhanced processing and integration

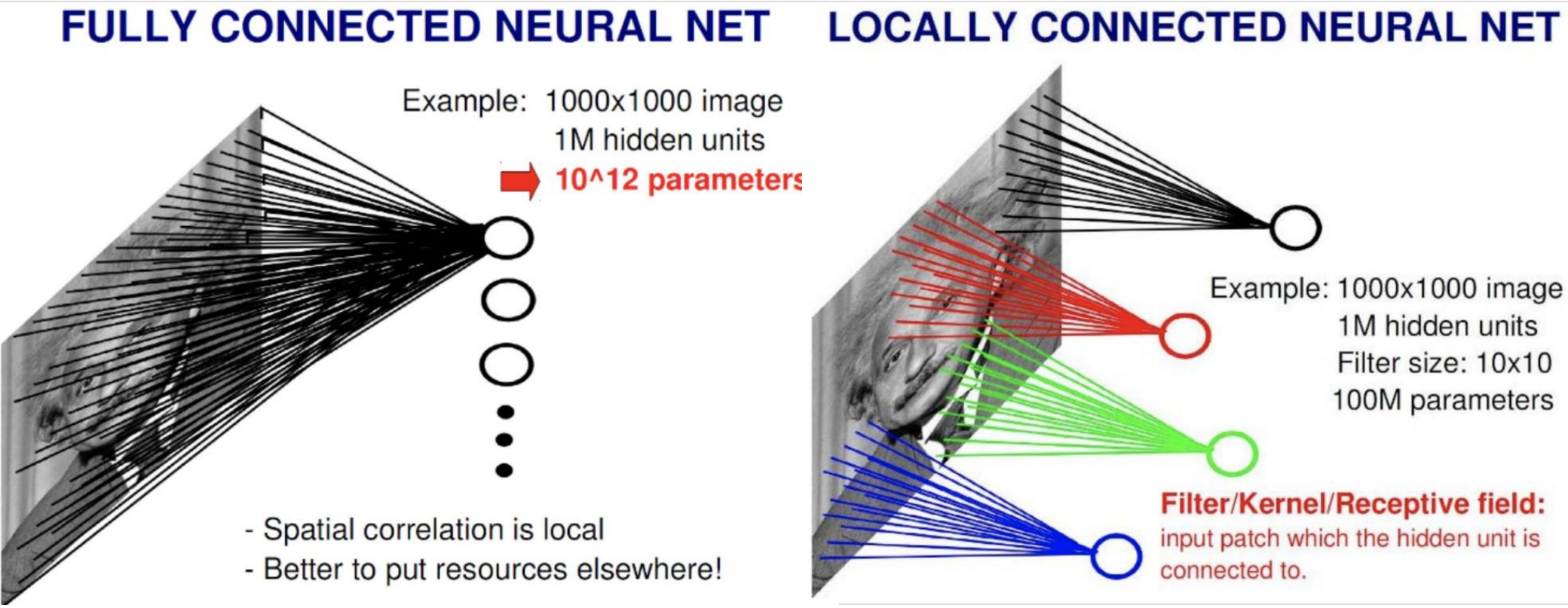


Spatial Coherence in Image

- Each pixel in an image can be classified to a category
  - which is called semantics (tree, road, sky, etc.)
- Neighboring pixels are likely from the same visual semantic region
  - local receptive field is better in such case

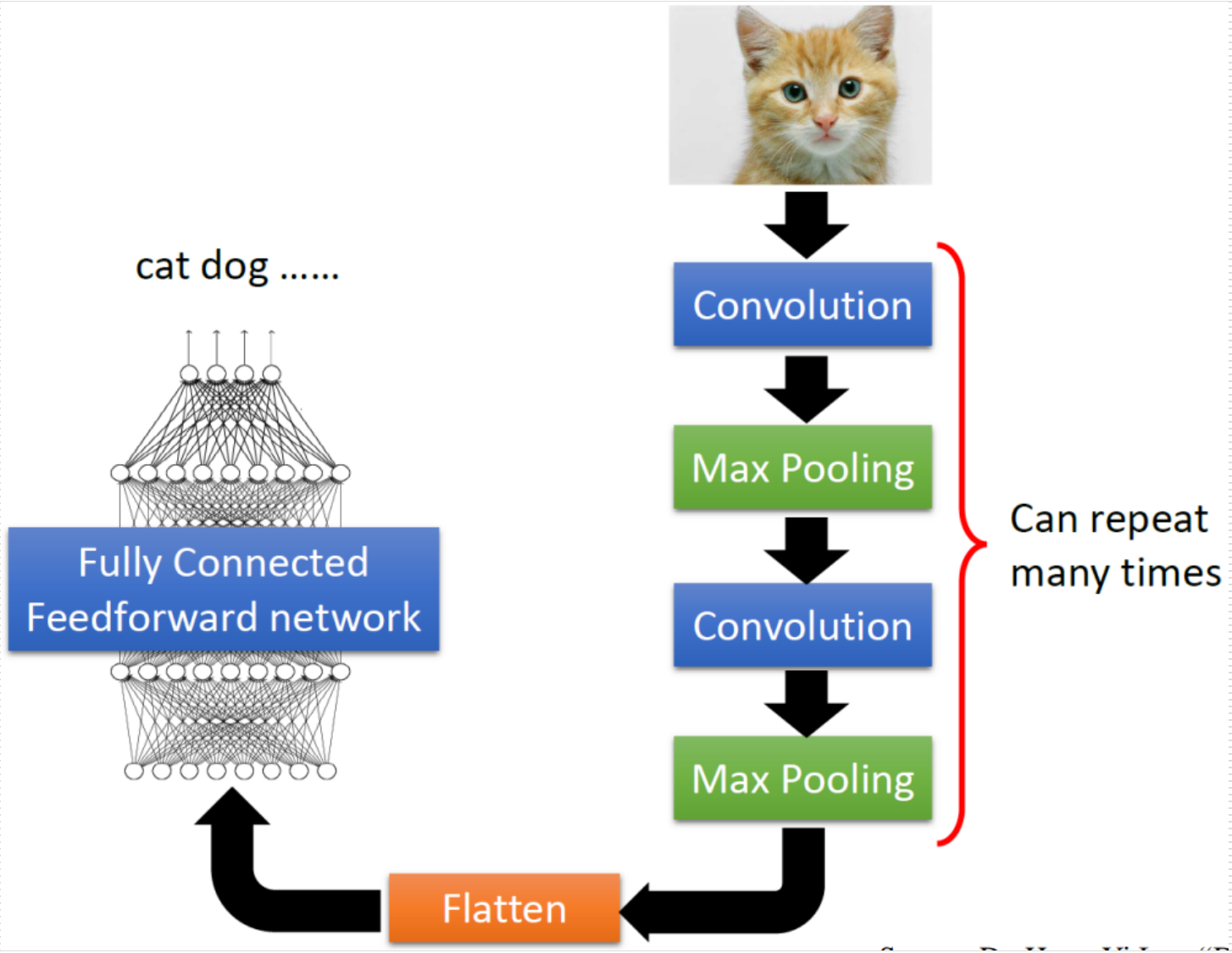


MLP vs. CNN



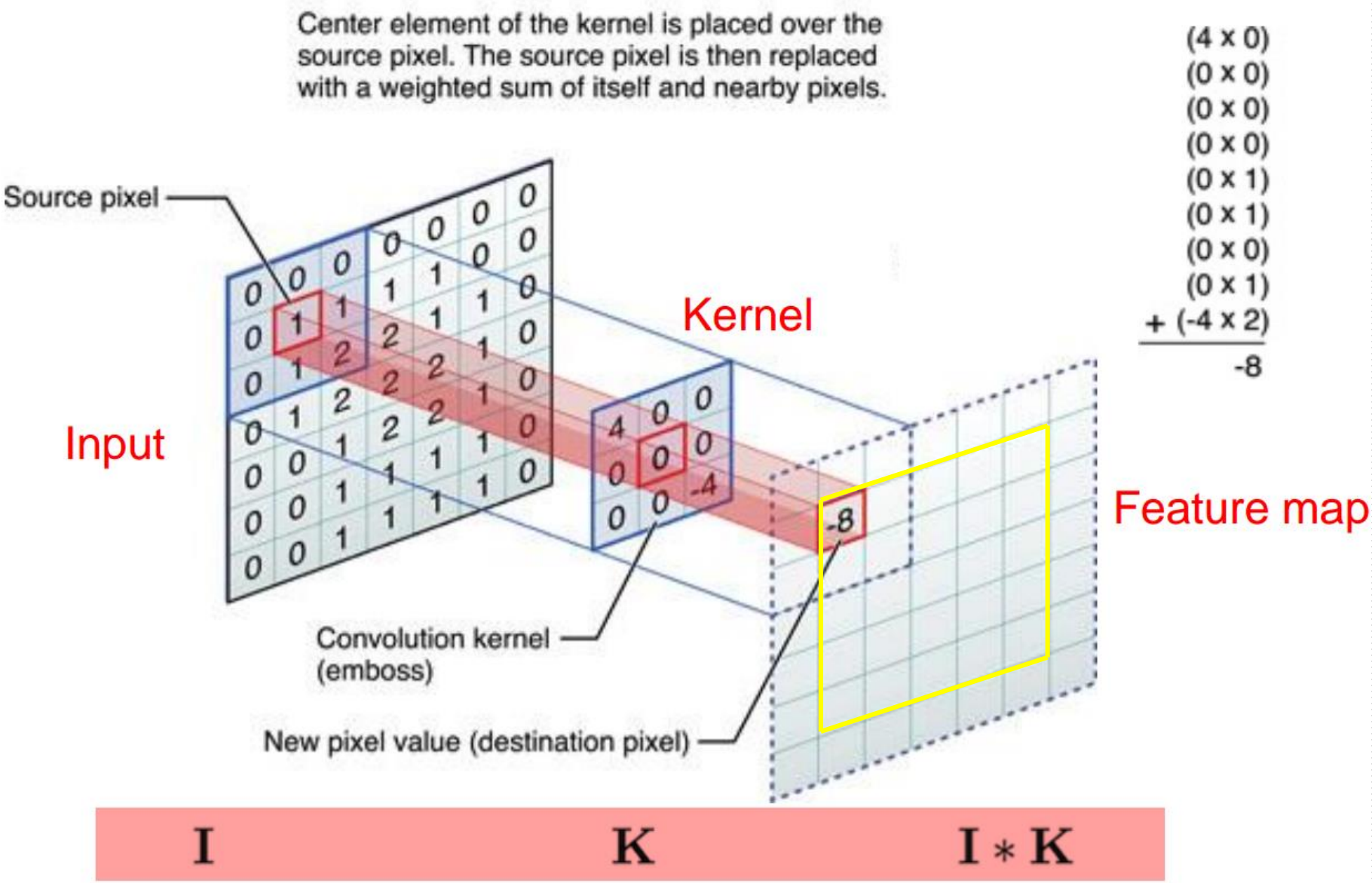
- Locally connected neural net is called CNN
  - stands for Convolutional Neural Network

Typical CNN Architecture



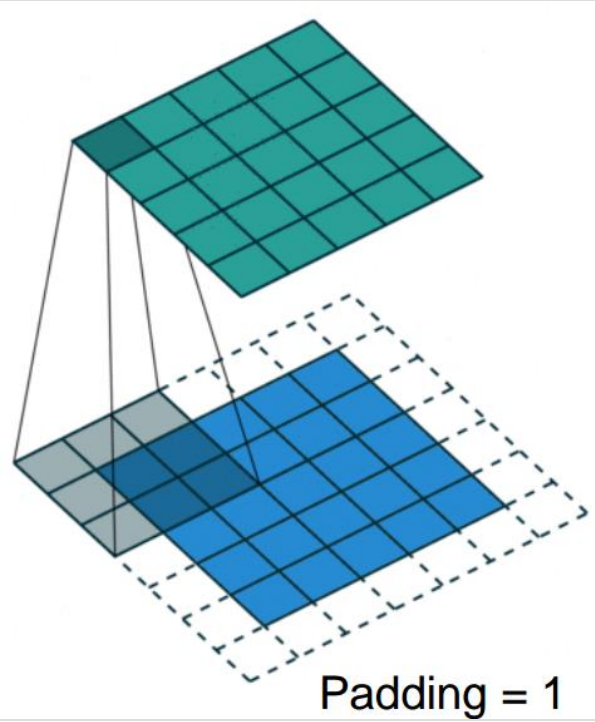
Details of Convolution

Convolution Operations



Zero Padding (1/3)

- Input image is padded with an additional border of a certain number of zeros around the edge



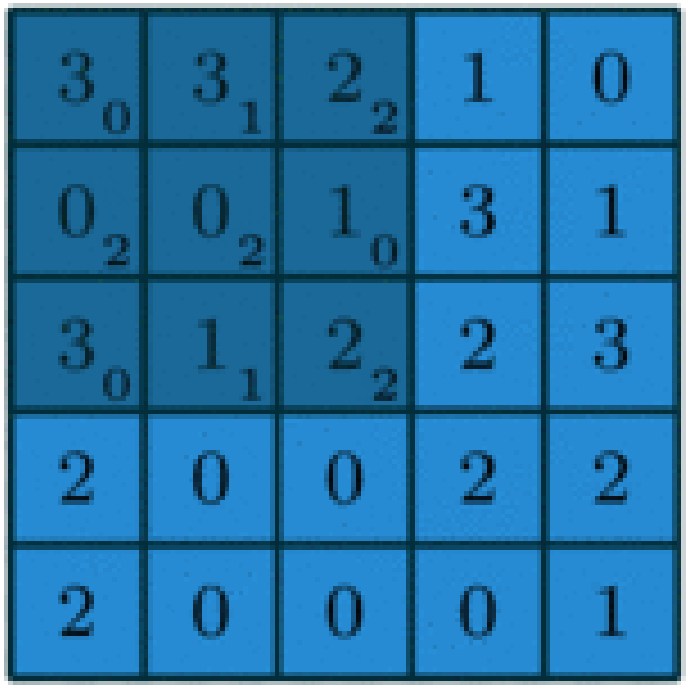
5	6	7	8
0	6	7	8
5	6	15	8
5	6	7	8

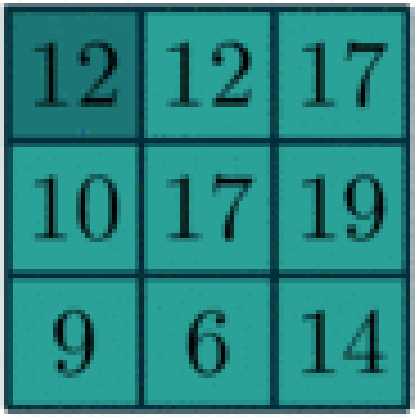
zero-padding →

0	0	0	0	0	0
0	5	6	7	8	0
0	0	6	7	8	0
0	5	6	15	8	0
0	5	6	7	8	0
0	0	0	0	0	0

Zero Padding (2/3)

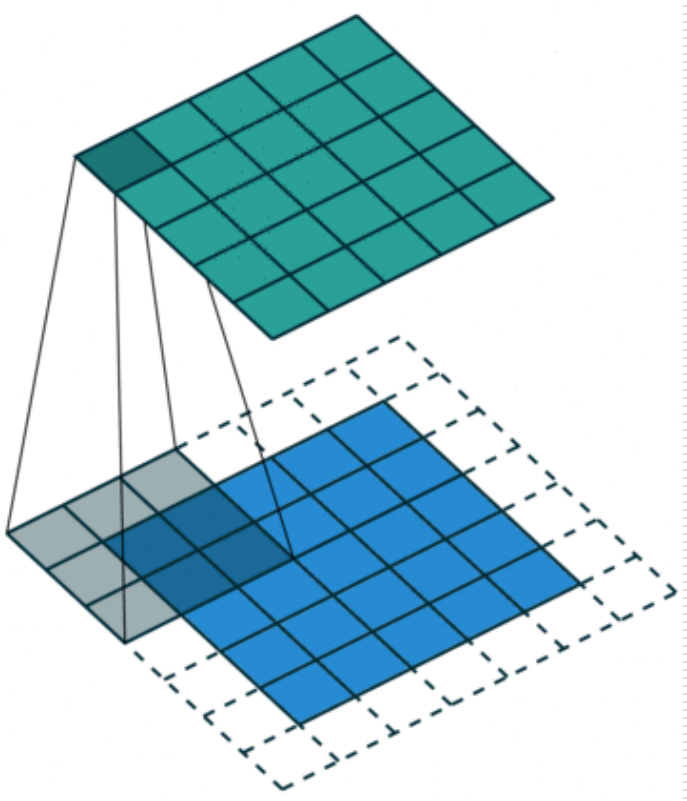
- Why zero padding?
  - 以5x5 feature map和3x3 kernel做convolution為例
    - 若**沒有做** zero padding 的話，3x3 kernel可在H方向和W方向各移動3步，故output feature map的大小為3x3。





Zero Padding (3/3)

- Why zero padding?
  - 以5x5 feature map和3x3 kernel做convolution為例
    - 若想讓output feature map的長寬和input feature map的長寬**一樣大**，可以zero padding一整圈，讓convolution的輸入變成7x7，則輸出就會變成5x5。

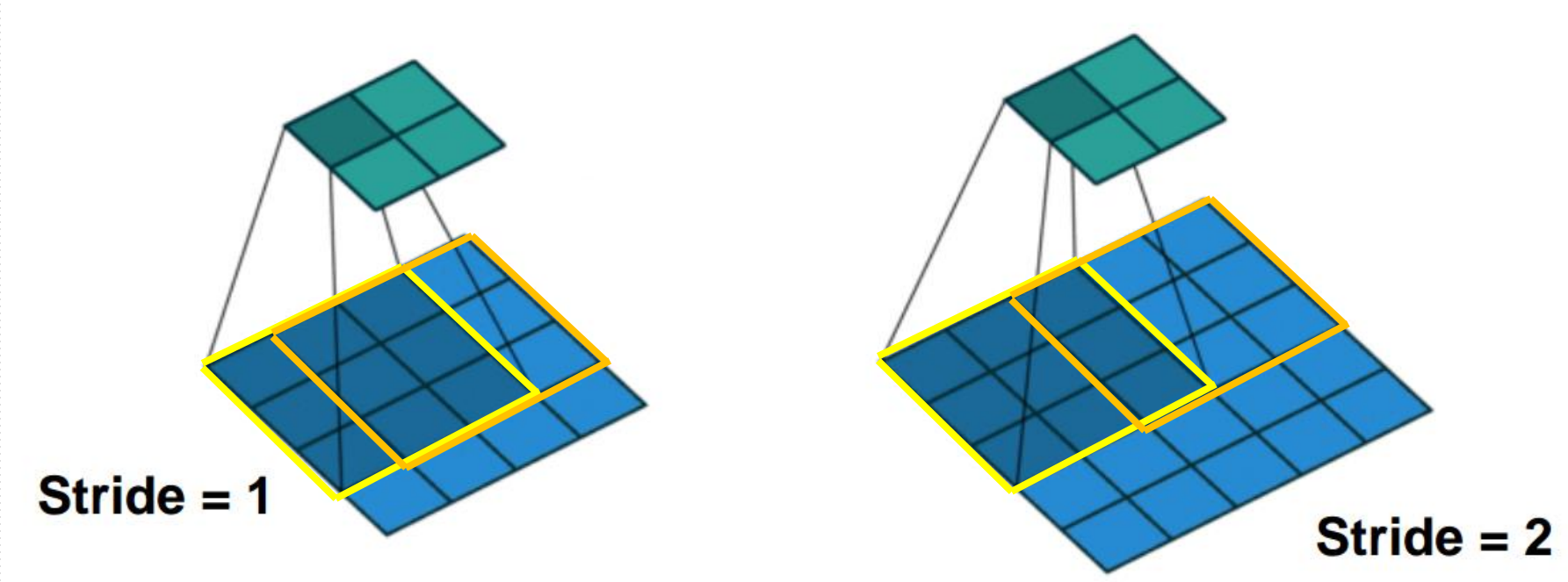


Kernel Size Convention

- 我們通常使用**odd x odd**的方形kernel。
  - EX
    - 3x3、5x5、7x7
  - Why odd?
    - If we want to keep **the same size** of feature map after convolution with the following setting.
      - Input feature map size = m x m
      - Kernel size = k x k
      - Stride = 1, **padding = p (圈)**
    - Calculation：
      - $\frac{m+2p-k}{1}+1=m \Rightarrow p=\frac{k-1}{2}$

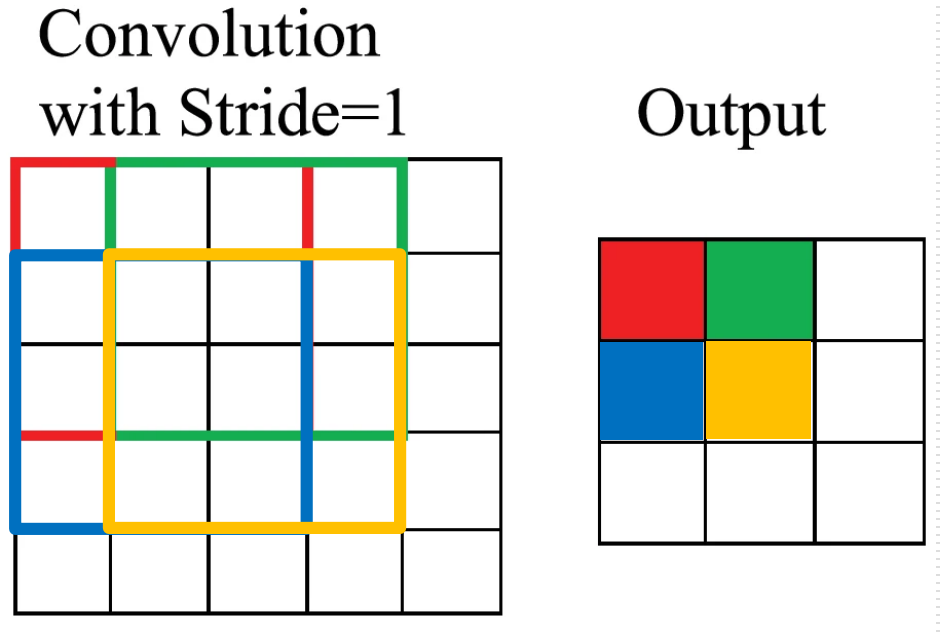
Stride (1/4)

- Stride is the number of pixels shifts over the input image.
- Larger stride has the effect of down-sampling the image but causes some feature losses of output.



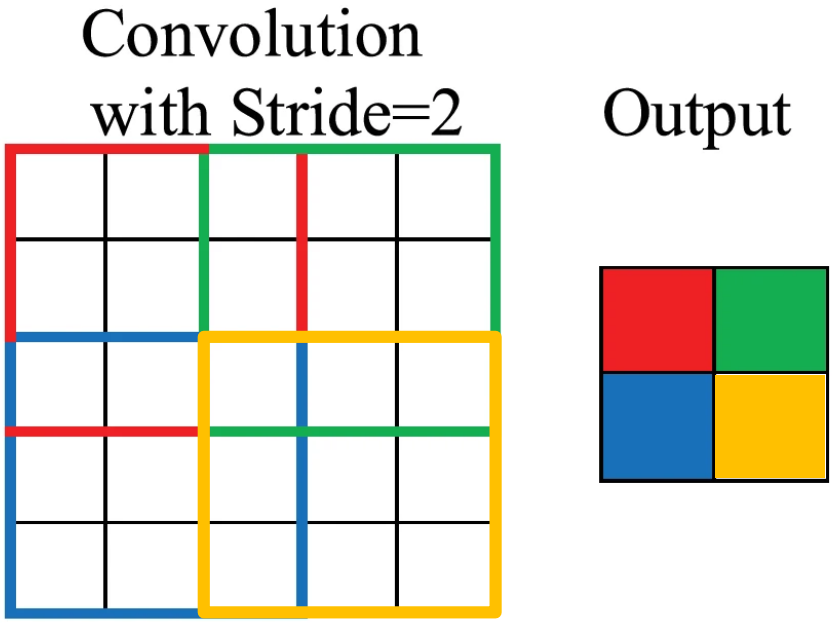
Stride (2/4)

- Why we use stride = 2 sometimes?
  - 縮圖
    - EX_1：
      - Input feature map：5x5
      - Kernel size：3x3, stride = **1**
      - Output feature map：3x3

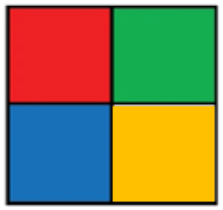
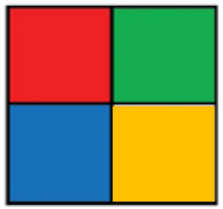


Stride (3/4)

- Why we use stride = 2 sometimes?
  - 縮圖
    - EX_2：
      - Input feature map：5x5
      - Kernel size：3x3, stride = **2**
      - Output feature map：2x2
    - 和上一頁的例子相比，
      使用stride = 2，可進一步縮圖。

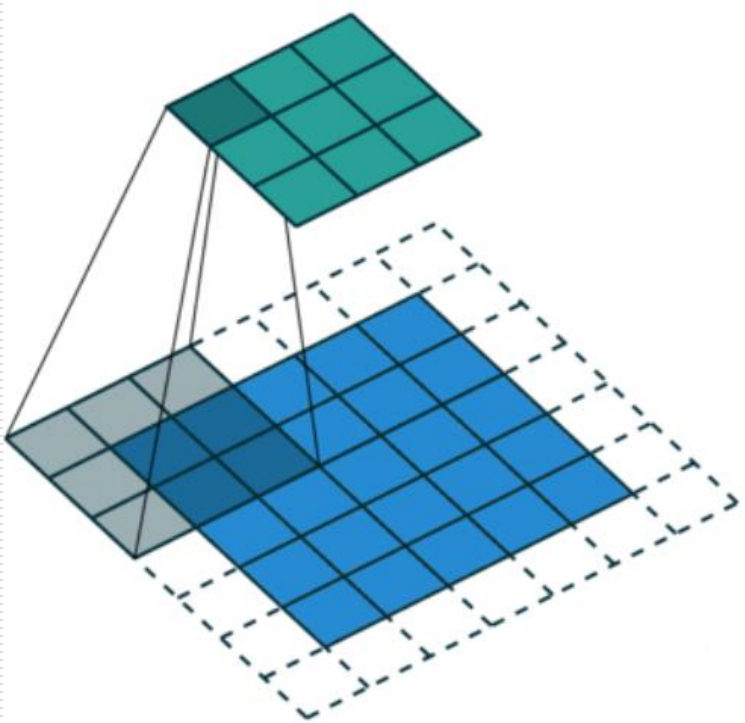


Stride (4/4)

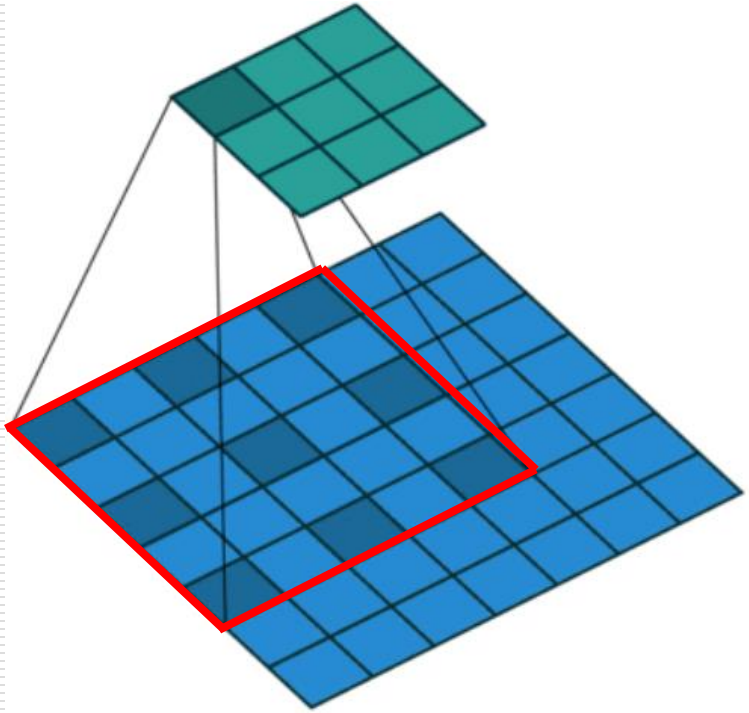
- Why we use stride = 2 sometimes?
  - **擴大receptive field (感知範圍)**
    - 第一個例子output feature map中的  囊括了原本input feature map中**4x4**的資訊。
    - 第二個例子output feature map中的  囊括了原本input feature map中**5x5**的資訊。
    - 可知增大stride能有效擴大receptive field。

Dilation

- Increase receptive field with smaller kernel
  - used in semantic segmentation



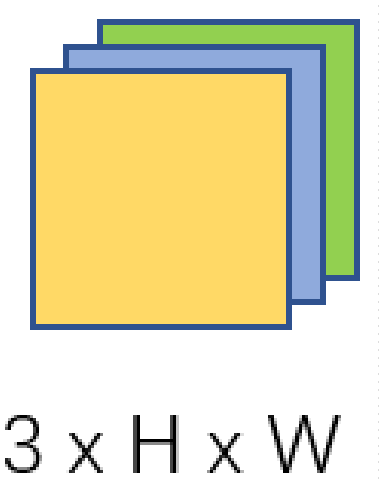
dilated rate = 1 (default)



dilated rate = 2

Channel (1/3)

- 就是thickness (厚度)
  - 可以理解為除了H、W之外的第三個維度。
    - 圖片(or kernel)的厚度



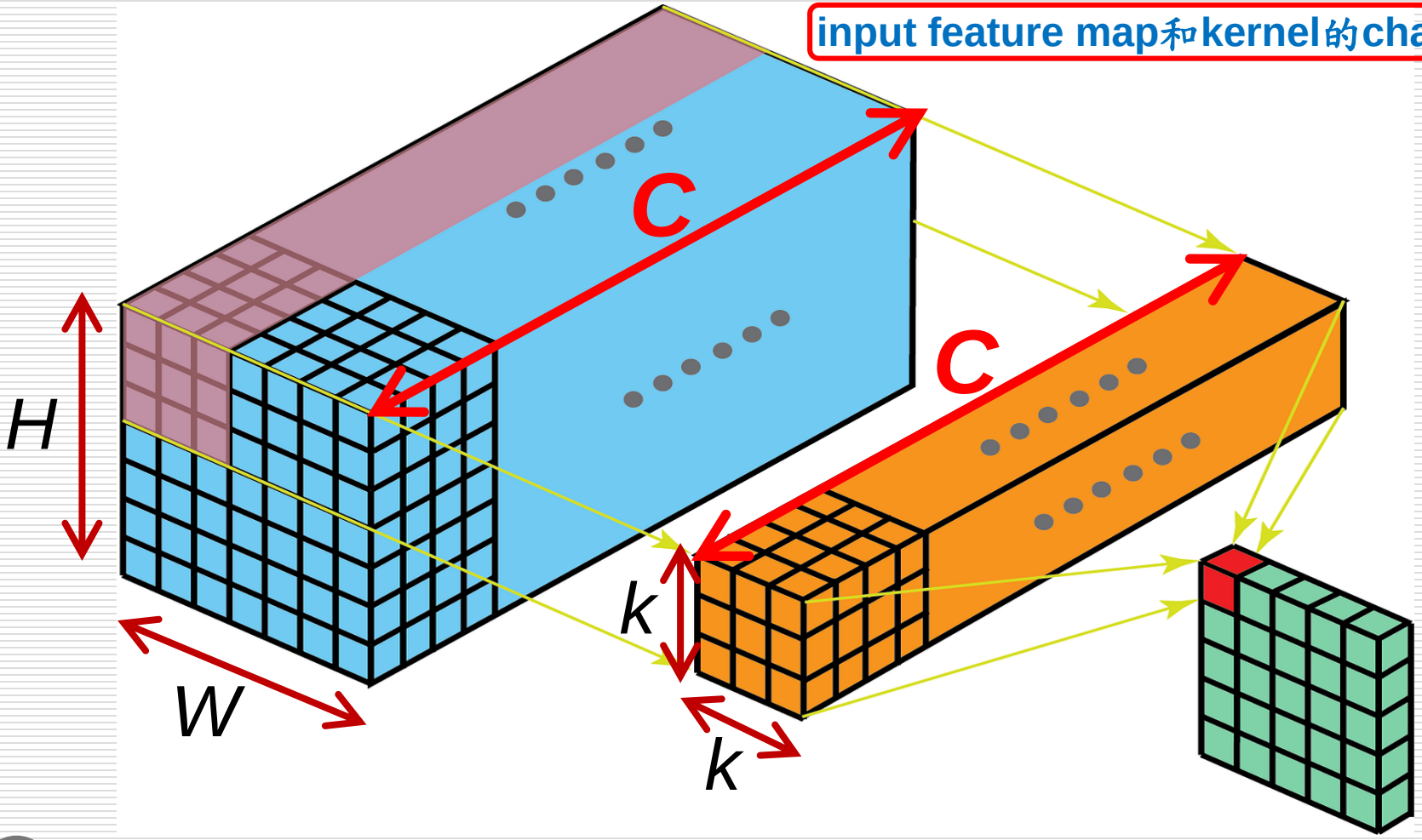
  - 以最常見的圖片來說，維度一般都是
    - (C, H, W) = (**3**, H, W)，因為顏色是由三原色組成，故可以用3個channels表達出所有顏色。

Channel (2/3)

在pytorch中，常寫成# of input channels * k * k，只是維度順序不同而已，本質一樣。

- Convolution with "**channel**"
  - 一組kernel $(k * k * \#\ of\ input\ channels)$ 決定一"片"output feature map

input feature map和kernel的channel數必須相同

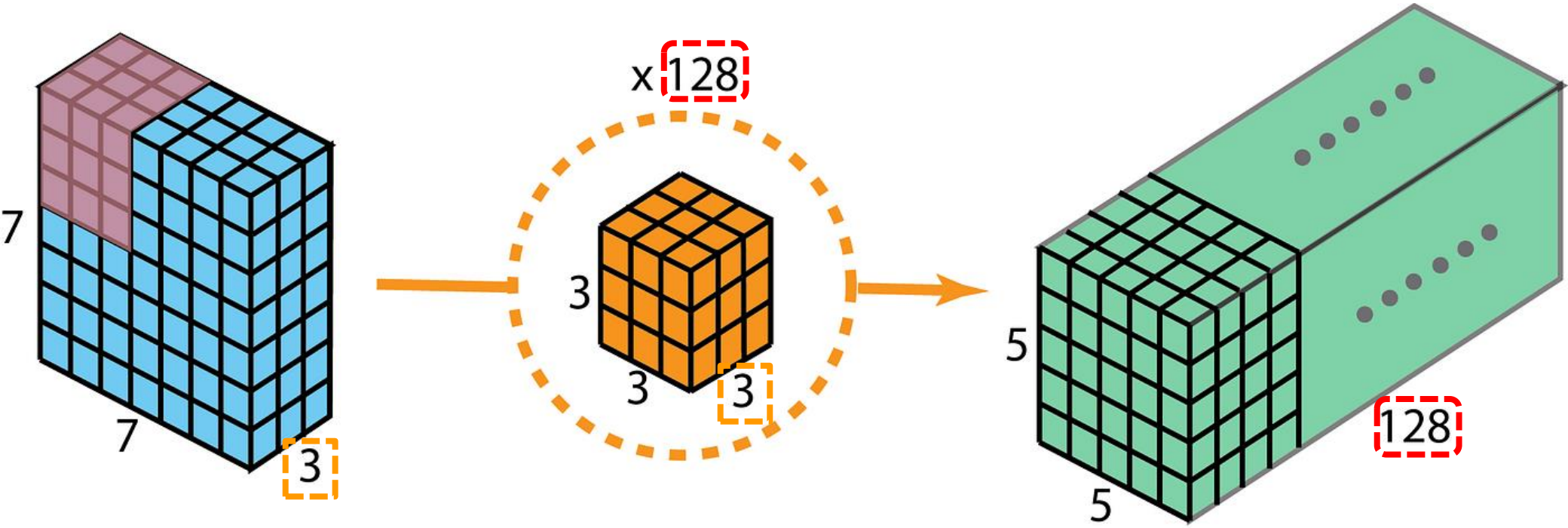


Channel (3/3)

- Convolution with "**channel**"
  - **# of output channels** = 共有**幾組**kernels
  - EX: **128組**kernels
    每組有3片3x3 kernels，
    一組稱為一個filter，共**128**個filters

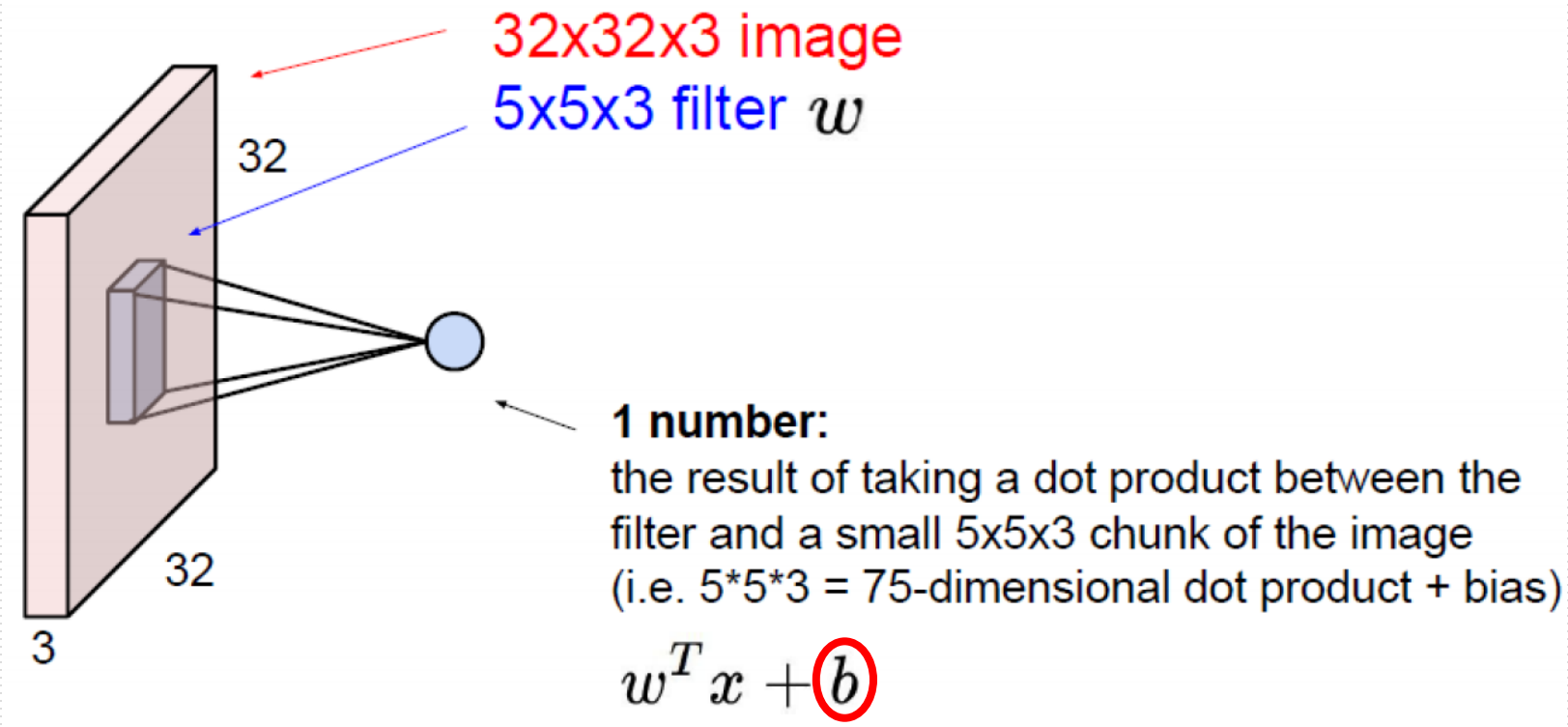
一組kernel = 複數片kernels

# of input channels 片



Bias

- Add bias to increase the flexibility of the prediction
  - ***a filter*** (一組kernel) ***has only one bias***
  - biases are tuned alongside weights in learning process
  - no bias ➔ no noticeable difference in big network training



Output Size Calculation

- With dilation rate = 1

$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

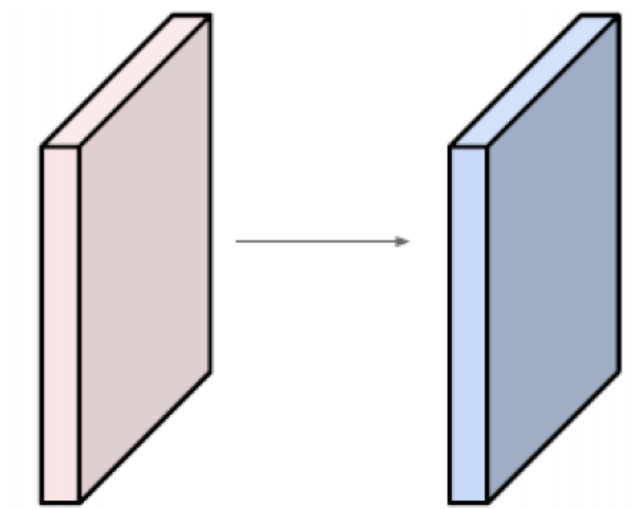
$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

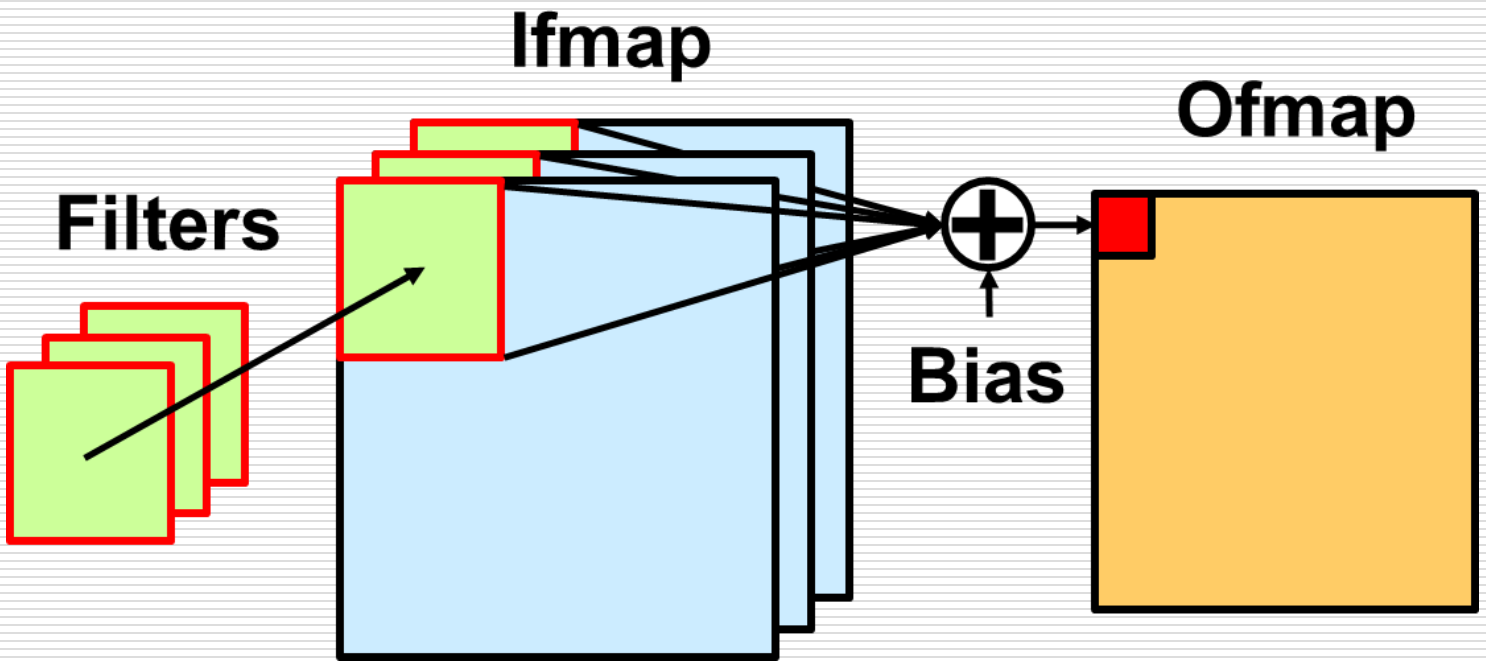
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



Parameter Size Calculation (1/2)

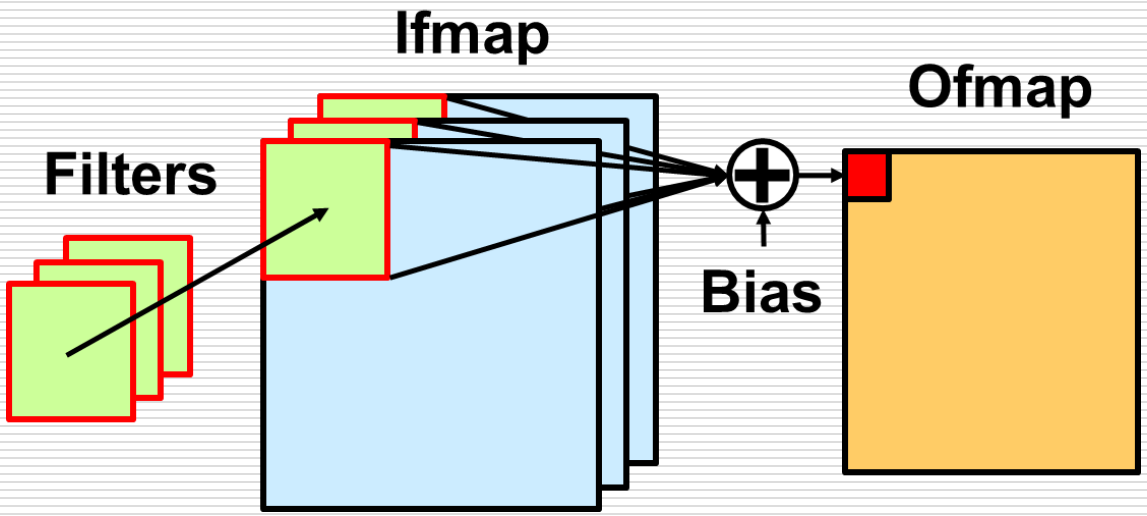
- Dilation rate = 1, padding = 2, bias = True
- Input channel = 3, output channel = 10
- Kernel size：(5, 5)



Parameter Size Calculation (2/2)

- Calculation :

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params
=> 76*10 = **760**
(+1 for bias)



Convolution Layer in PyTorch (1/2)

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)`

- Apply 2D convolution operations over several input planes

```
class Conv2D(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride, padding):
        super(Conv2D, self).__init__()

        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        self.batchnorm = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, input_tensor):
        output = self.conv(input_tensor)
        output = self.batchnorm(output)
        output = self.relu(output)
        return output
```

Convolution Layer in PyTorch (2/2)

Input channel = 1, Output Channel = 20
Kernel Size = 3, Stride = 1, Padding = 1

```
# Creating an instance of the Conv2D class
in_channels = 1
out_channels = 20
kernel_size = 3
stride = 1
padding = 1

conv2d_layer = Conv2D(in_channels, out_channels, kernel_size, stride, padding)

# Random input tensor
input_tensor = torch.randn(1, in_channels, 32, 32)

# Forward pass
output = conv2d_layer(input_tensor)

# Print the output shape
print(output.shape)
```

Common Layers in a CNN

Common Layers in a CNN (1/2)

- Activation Function
  - 用於引入非線性特性到神經網絡中，以增加其表示能力。
- Pooling Layer
  - 池化層主要用於減少特徵圖的空間尺寸，同時保留重要的特徵。最常見的池化方式是最大池化（Max Pooling），它選擇區域內的最大值作為特徵圖的代表。
- Fully Connected Layer
  - 就是MLP。
  - 也被稱為密集連接層（Dense Layer）。全連接層將前一層的所有特徵圖都連接到下一層的每個神經元上。這一層通常用於最後的分類任務，將提取的特徵映射到不同的類別。

Common Layers in a CNN (2/2)

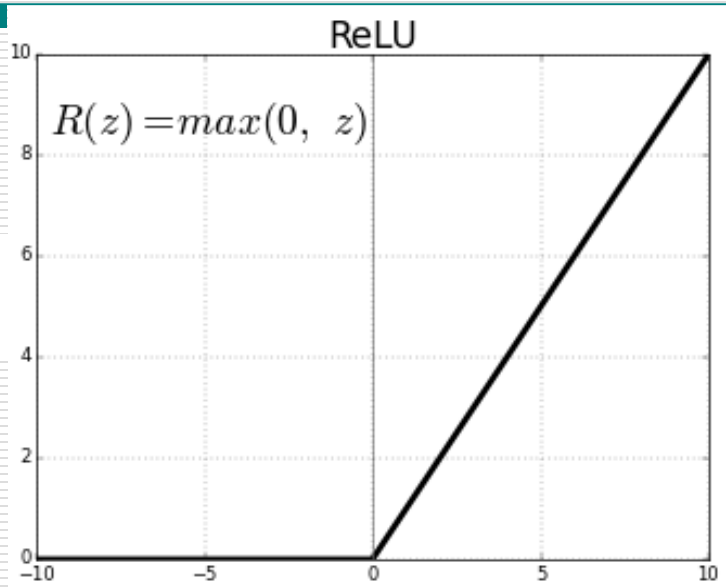
- Normalization Layer
  - 常見的正規化層包括批量正規化（Batch Normalization）和層正規化（Layer Normalization），它們可以有效地調節特徵的尺度，提高模型的收斂速度和性能。

- Skip Connection
  - Aka residual connection。
  - 它可以有效地解決深層網絡中的梯度消失問題，幫助資訊在網絡中更好地流動。

Common Activation Functions in CNNs

- nn.ReLU()
  [most commonly used in CNNs]

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



- Others
  - Leaky ReLU
  - PReLU
  - ReLU6 (ReLUx)
  - Mish(在HarDNet中常使用)

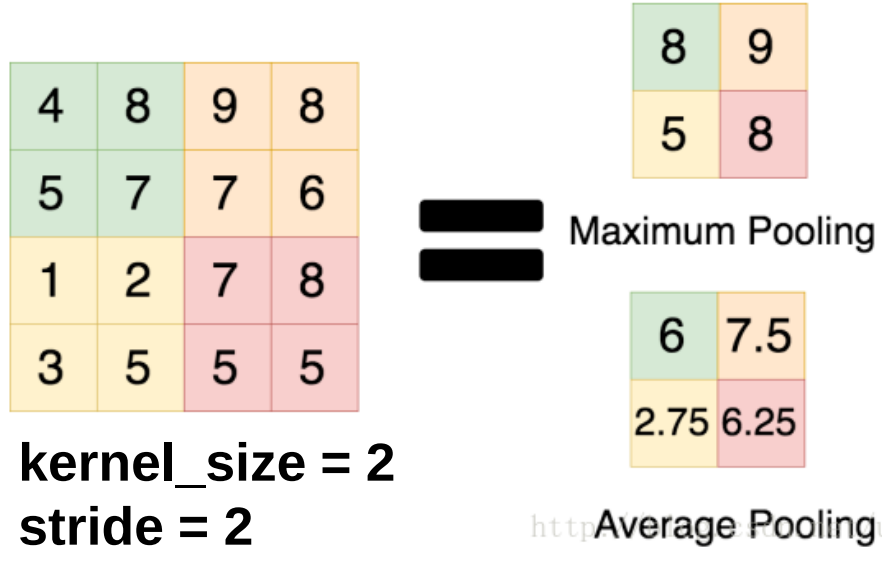
Pooling

- Reduce the output feature map size after convolution
- 2 common types of pooling
  - **max pooling**: find the **max value** in the kernel region

CLASS torch.nn.MaxPool2d(*kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False*) [SOURCE]

  - **average pooling**: find the **average value** in the kernel region

CLASS torch.nn.AvgPool2d(*kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None*) [SOURCE]



AdaptiveAvgPool2d

CLASS torch.nn.AdaptiveAvgPool2d(*output_size*)

- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, S_0, S_1) or (C, S_0, S_1), where $S = \text{output\_size}$.

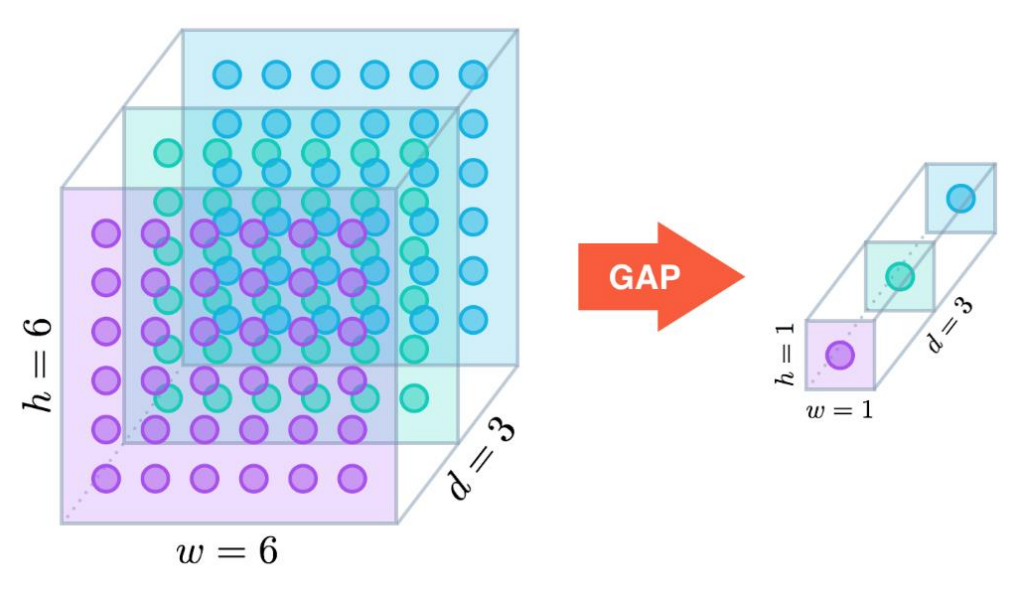
- 對input feature map依照指定的輸出大小作Average Pooling，不用自己設定kernel size、padding等參數，會依照指定大小算出對應的參數值。

```
# target output size of 5x7
m = nn.AdaptiveAvgPool2d((5, 7))
input = torch.randn(1, 64, 8, 9)
output = m(input)
```

```
# target output size of 7x7 (square)
m = nn.AdaptiveAvgPool2d(7)
input = torch.randn(1, 64, 10, 9)
output = m(input)
```

Global Average Pooling (GAP)

- GAP is an operation designed to replace FC layers in CNNs
  - calculate the average of each whole feature map
  - no parameter to optimize in the GAP (avoid overfitting)



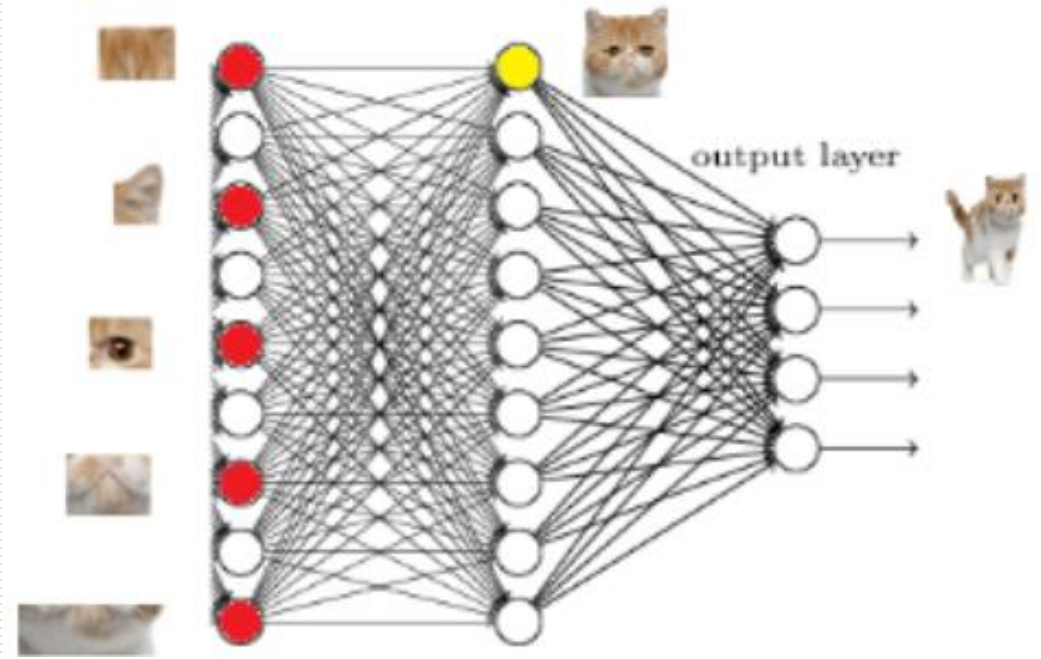
- AdaptiveAvgPool2d applies average pooling to each input plane with given output size
  - *output_size* is set to (1,1) in GAP

Flatten & Fully Connected Layer

- Most CNNs have some MLP layers at the end
  - flatten the output of the last convolution layer
  - combine the features extracted from the previous layer
  - # of neurons at the last layer = # of classes (classification)

```
self.classifier = nn.Sequential(
    nn.Linear(512 * 7 * 7, 4096),
    nn.ReLU(True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(True),
    nn.Dropout(),
    nn.Linear(4096, num_classes),
)
def forward(self, x: torch.Tensor)
    x = self.features(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.classifier(x)
    return x
```

VGG16



Batch Normalization (1/4)

CLASS torch.nn.BatchNorm2d(*num_features*, *eps=1e-05*, *momentum=0.1*, *affine=True*, *track_running_stats=True*, *device=None*, *dtype=None*) [SOURCE]

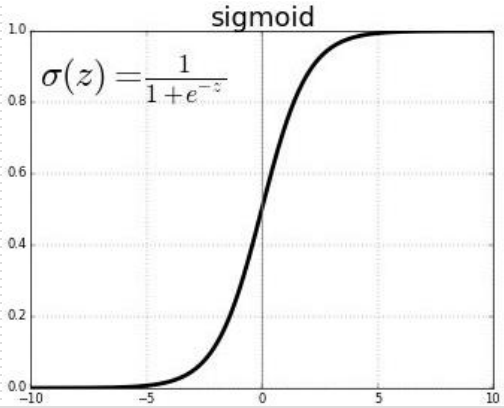
- Applies batch normalization over a 4D input
  - *num_features:* (batch_size * height * width * channel_num)
- Learnable parameters γ & β are updated during training
  - *affine:* γ & β is unavailable when affine = False
- Mean & variance(from training) are fixed during inference
  - *momentum:* used to calculate running mean & variance
  - *track_running_stats:* update running mean & variance in training
- *EPS:* ε to prevent division by zero

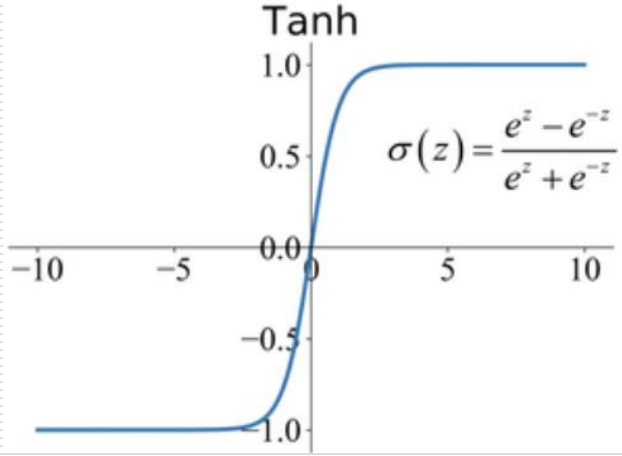
$$y = \frac{z - E[z]}{\sqrt{\mathrm{Var}[z] + \varepsilon}} * \gamma + \beta$$

Batch Normalization (2/4)

CLASS torch.nn.BatchNorm2d(*num_features*, *eps=1e-05*, *momentum=0.1*, *affine=True*, *track_running_stats=True*, *device=None*, *dtype=None*) [SOURCE]

- Typical position in NN
  - After convolution layer
  - Before activation function
  - Why?
    - 有些 activation function (tanh 或是 sigmoid) 會有 saturation region，所以先做 normalization 再丟進 activation function，確保值不會掉進 saturation region。





Batch Normalization (3/4)

- Fused in convolution layer
  - BN formula：

$$y = \frac{z - E[z]}{\sqrt{\mathrm{Var}[z] + \varepsilon}} * \gamma + \beta$$

  - $z = wx + b$
  - $\Rightarrow y = \left(\frac{wx+b-mean}{\sqrt{\mathrm{Var}[z]+\varepsilon}}\right) * \gamma + \beta = \left(\frac{w*\gamma}{\sqrt{\mathrm{Var}[z]+\varepsilon}}\right) * x + \left(\frac{b-mean}{\sqrt{\mathrm{Var}[z]+\varepsilon}} * \gamma + \beta\right)$
  - $\Rightarrow new\ w = \left(\frac{w*\gamma}{\sqrt{\mathrm{Var}[z]+\varepsilon}}\right), new\ b = \left(\frac{b-mean}{\sqrt{\mathrm{Var}[z]+\varepsilon}} * \gamma + \beta\right)$
  - ⇒ 故在**inference**(統計量不再變動)時，可以**不用通過BN** layer，因為BN operation可以被融進前面的**Convolution layer**中。

Batch Normalization (4/4)

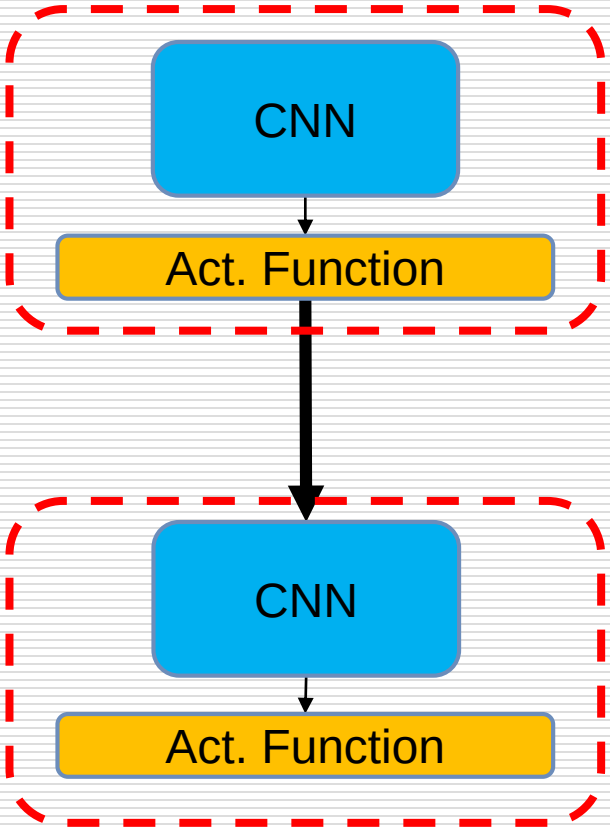
- Fused in convolution layer
  - "**torch.ao.quantization.fuse_modules**"

CLASS torch.ao.quantization.fuse_modules(*model*, *modules_to_fuse*, *inplace=False*, *fuser_func=<function fuse_known_modules>*, *fuse_custom_config_dict=None*) [SOURCE]

```
m = M().eval()
# m is a module containing the sub-modules below
modules_to_fuse = [ ['conv1', 'bn1', 'relu1'], ['submodule.conv', 'submodule.relu']]
fused_m = torch.ao.quantization.fuse_modules(m, modules_to_fuse)
output = fused_m(input)

m = M().eval()
# Alternately provide a single list of modules to fuse
modules_to_fuse = ['conv1', 'bn1', 'relu1']
fused_m = torch.ao.quantization.fuse_modules(m, modules_to_fuse)
output = fused_m(input)
```

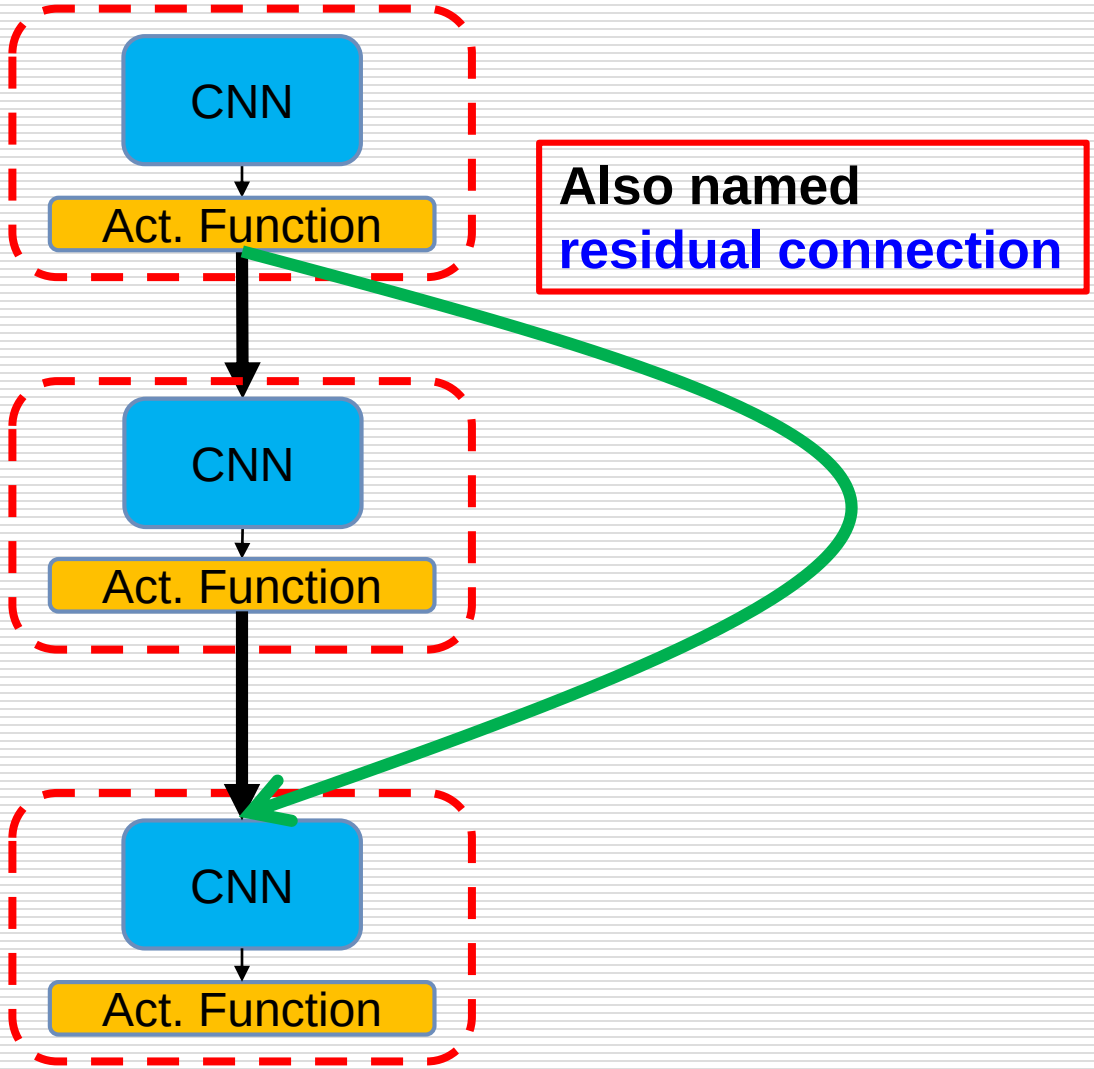
Connection Method (1/4)

- **Normal connection**



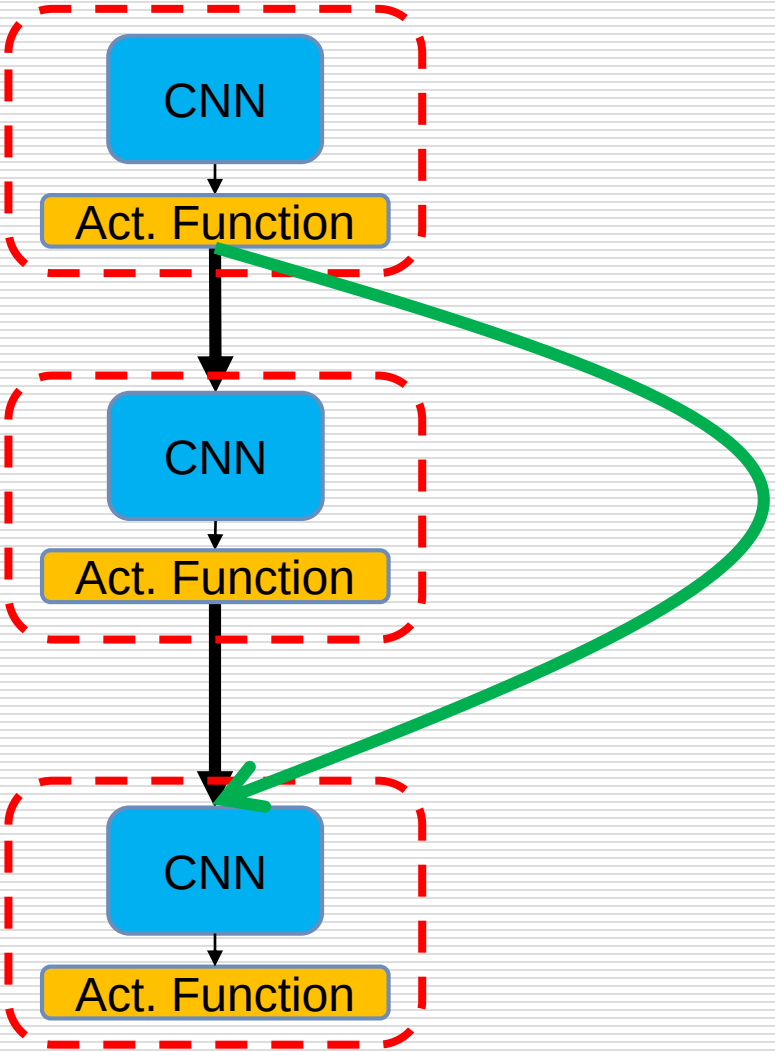
Connection Method (2/4)

- **Skip connection**
  - Addition
  - Concatenation



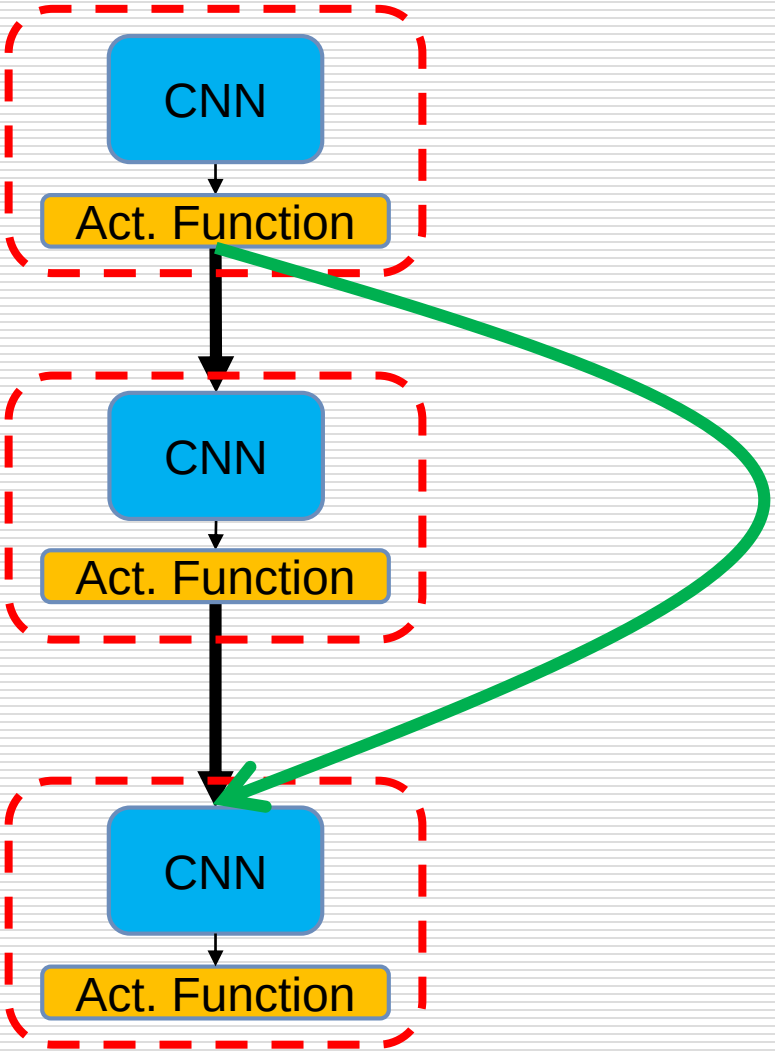
Connection Method (3/4)

- Addition
  - 在normal connection & skip connection的交會處，採直接element-wise相加(or 加權相加)，但相加前提是tensor的維度要相同。
  - 若不同則須調整維度
    - Padding
    - Up-sample
    - Down-sample
    - 1x1 Conv. (for channel dimension)



Connection Method (4/4)

- Concatenation
  - 在normal connection & skip connection 的交會處，將兩個feature maps在 **channel dimension**上串接起來，但兩者的H、W必須相同。
  - EX：(B, C, H, W)
    - Concatenate (B, 8, 24, 24) & (B, 4, 24, 24) at channel dimension.
      - Outcome：(B, 12, 24, 24)

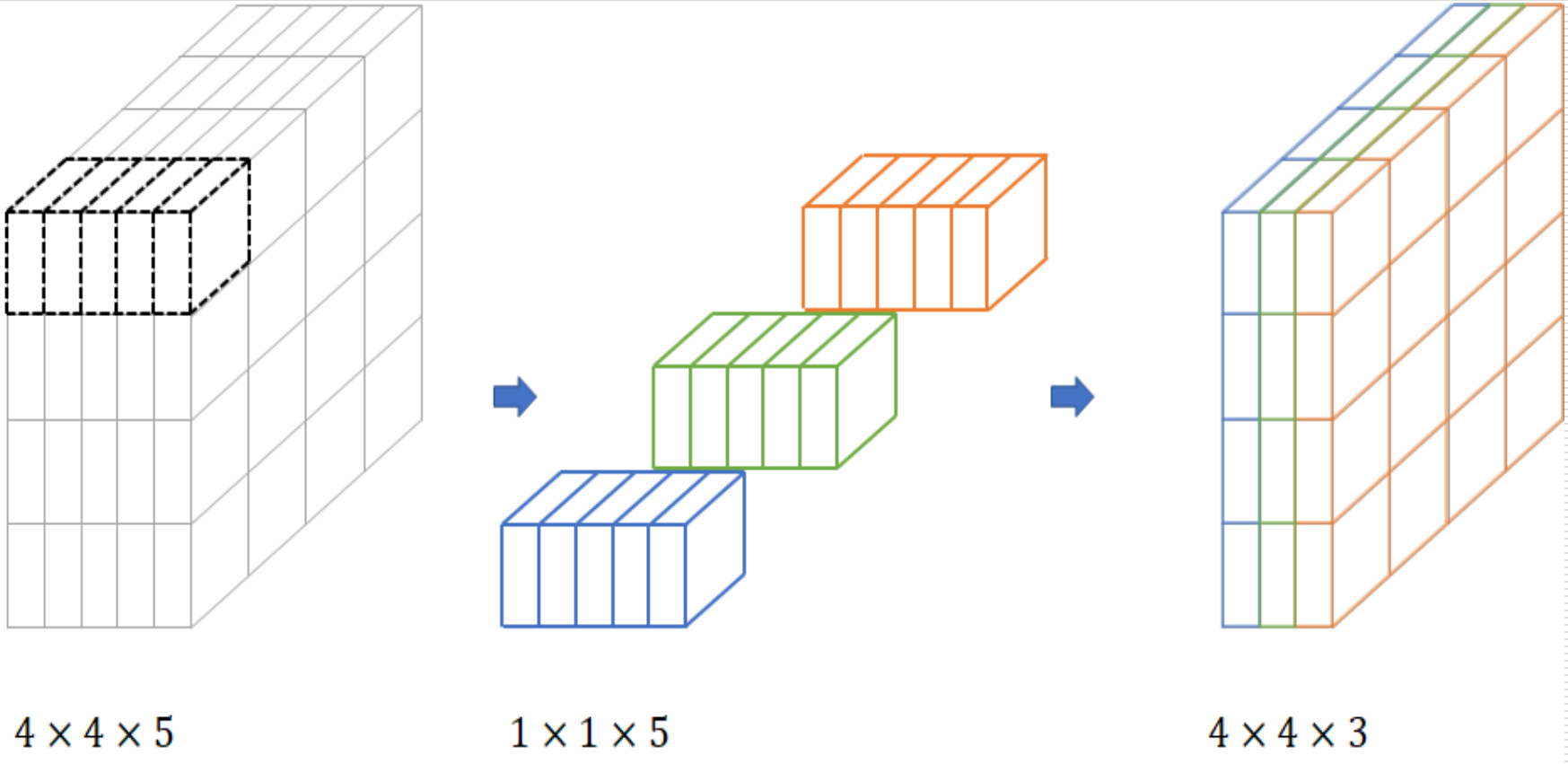



```
A = torch.randn(B, 8, 24, 24)
B = torch.randn(B, 12, 24, 24)
concatenated_tensor = torch.cat((A, B), dim=1)
```

Various Convolution Styles

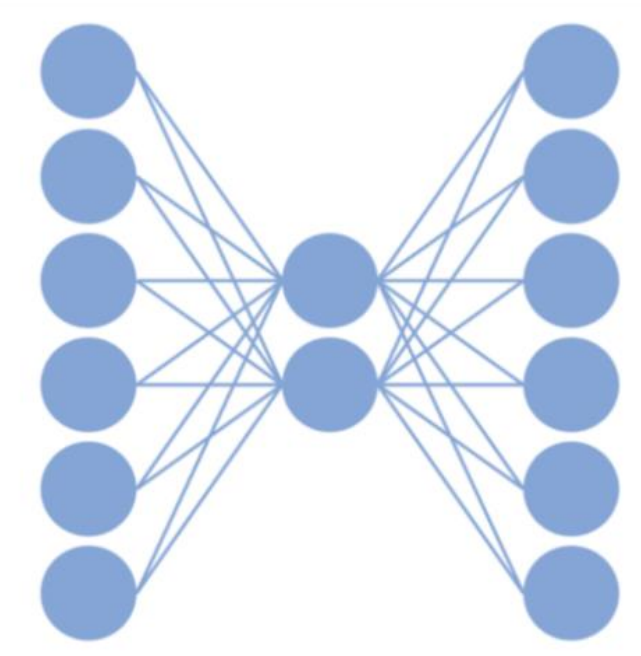
1x1 Convolution (1/3)

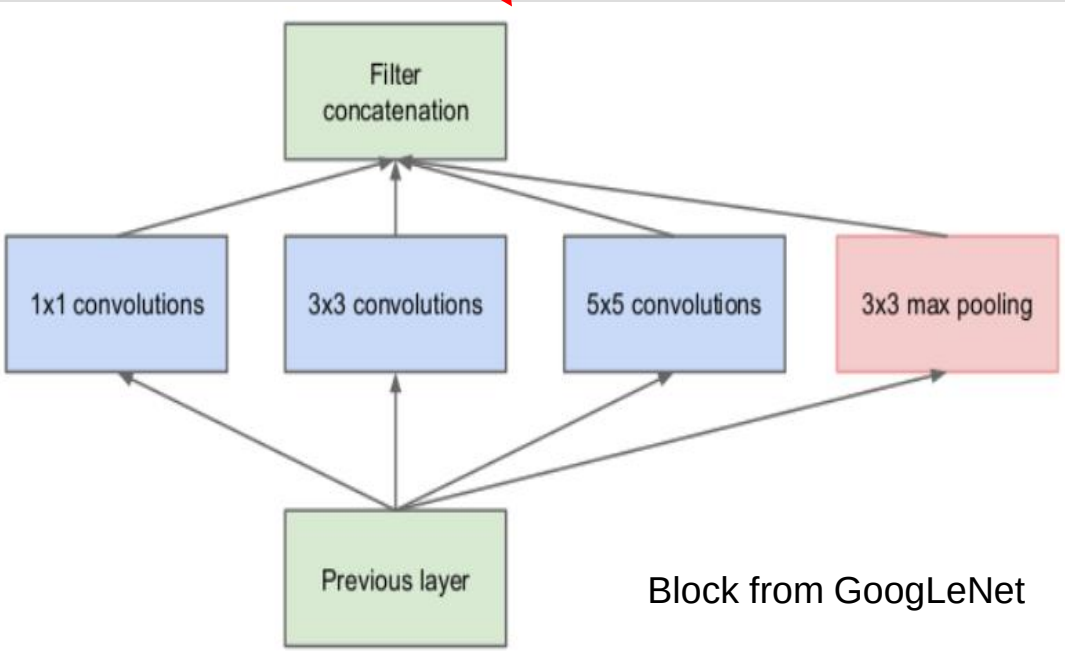
- Adjust the number of output channels by convolution
  - linear combination with different channels



1x1 Convolution (2/3)

- Adjust the number of output channels by convolution
  - add bottlenecks to compress data bandwidth
  - extract features with various scales from input





Block from GoogLeNet

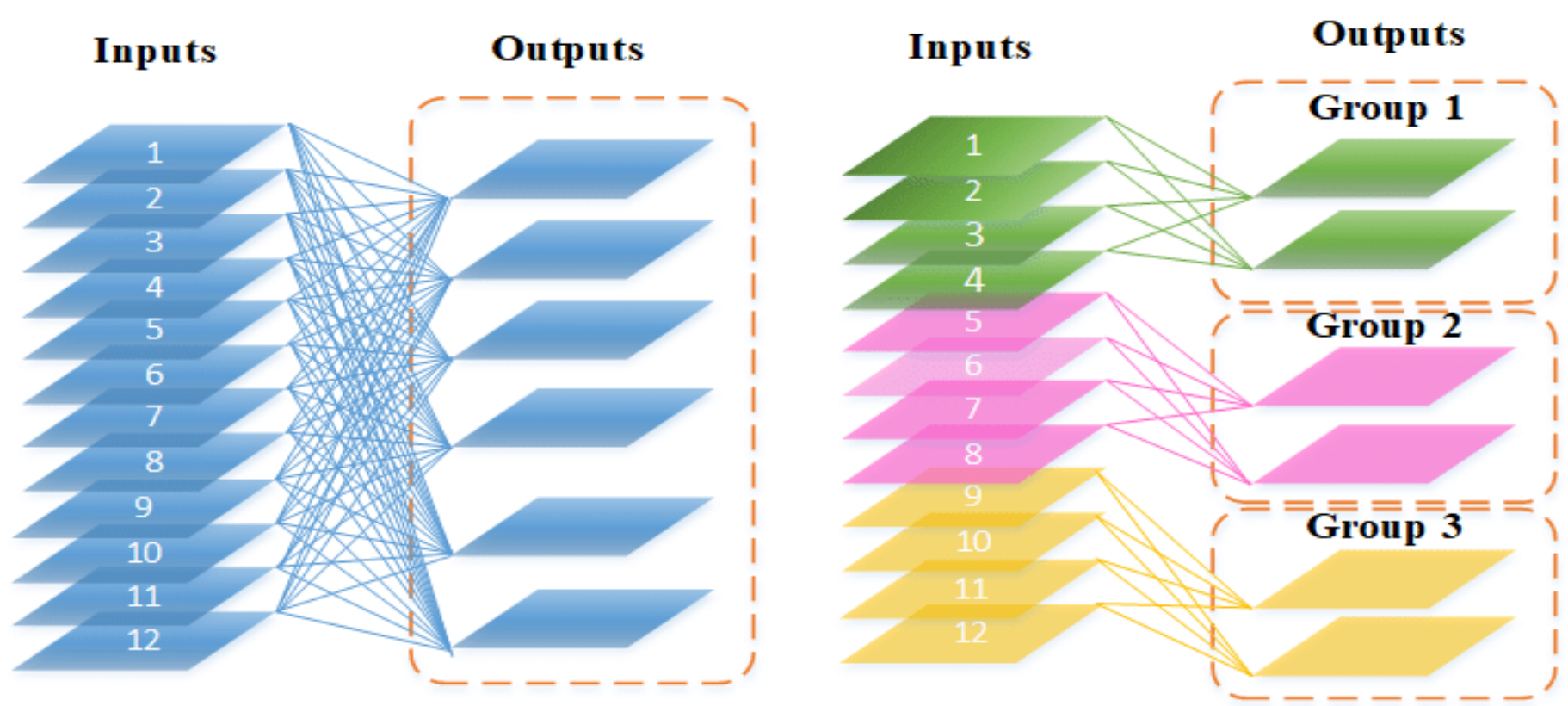
1x1 Convolution (3/3)

- Implementation

```
class AdjustChannels(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(AdjustChannels, self).__init__()

        self.conv_layer = nn.Conv2d(in_channels, out_channels, kernel_size=1)
        self.batchnorm_layer = nn.BatchNorm2d(out_channels)

    def forward(self, input_tensor):
        output_tensor = self.batchnorm_layer(self.conv_layer(input_tensor))
        return output_tensor
```

Group Convolution (1/3)

- 將input feature maps在channel維度上進行分組，每組以不同kernel (輸出之H、W需每組相同) 來做convolution，最後將各組的output feature maps在channel維度上做concatenation，得到最終輸出。



Group Convolution (2/3)

- 假設進行convolution前後之H、W相同，以下為其他設定：
  - Input：(B, H, W, C0)
  - # of group = **G**
  - Kernel size：(K, K)
  - Output：(B, H, W, C1)

- 參數量計算：
  - Shape of kernel in each group：(K, K, $\frac{C0}{G}$)
  - # of parameters in each group：$K * K * \left(\frac{C0}{G}\right) * \left(\frac{C1}{G}\right)$
  - # of total parameters： $K * K * \left(\frac{C0}{G}\right) * \left(\frac{C1}{G}\right) * G = \frac{\boldsymbol{K^2 * C0 * C1}}{\boldsymbol{G}}$

Group Convolution (3/3)

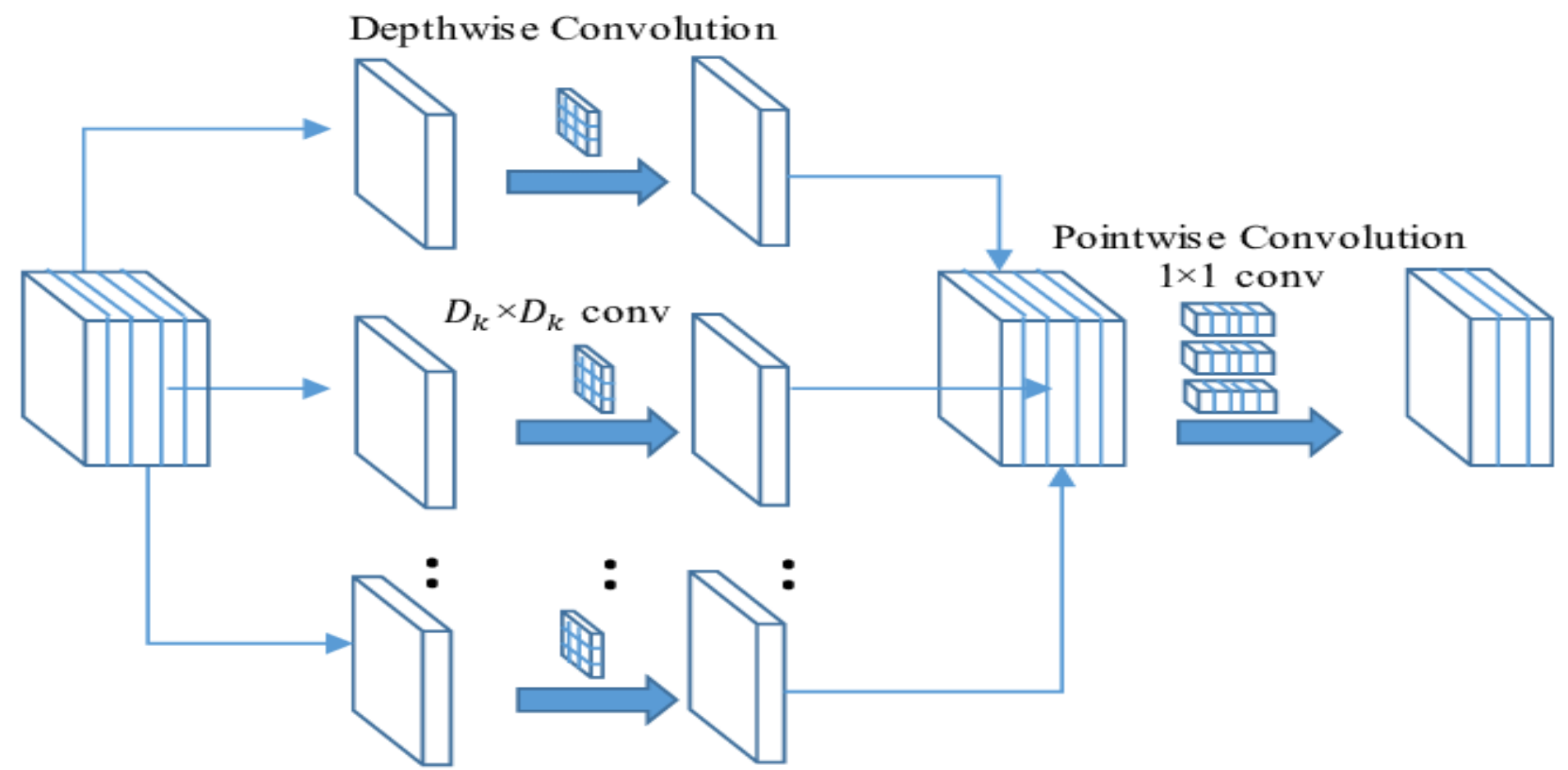
- 很明顯可看出來當# of group = G = 1時，此為一般的convolution。
- 故一般的convolution可視為group convolution之特例。

- 一般convolution之參數量計算：
  - 承上頁，# of total parameters： $K * K * \left(\frac{C0}{G}\right) * \left(\frac{C1}{G}\right) * G = \frac{K^2 * C0 * C1}{G}$
  - $G = 1$， # of total parameters：$K^2 * C0 * C1$

- 由上可知，使用Group Convolution可有效減少參數量，變為原本的 $\frac{1}{G}$ 倍。

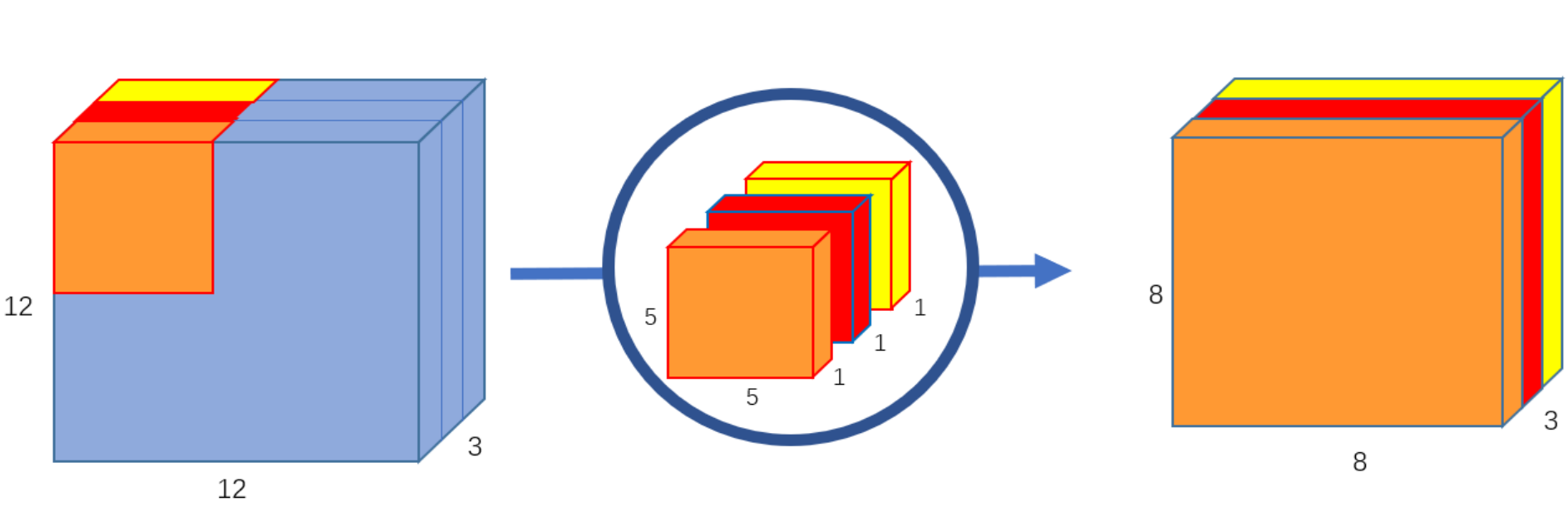
Depth-Wise Separable Convolution (1/4)

- Split a normal convolution to 2 parts to reduce computation cost
  - **Depth-wise convolution + Point-wise convolution**



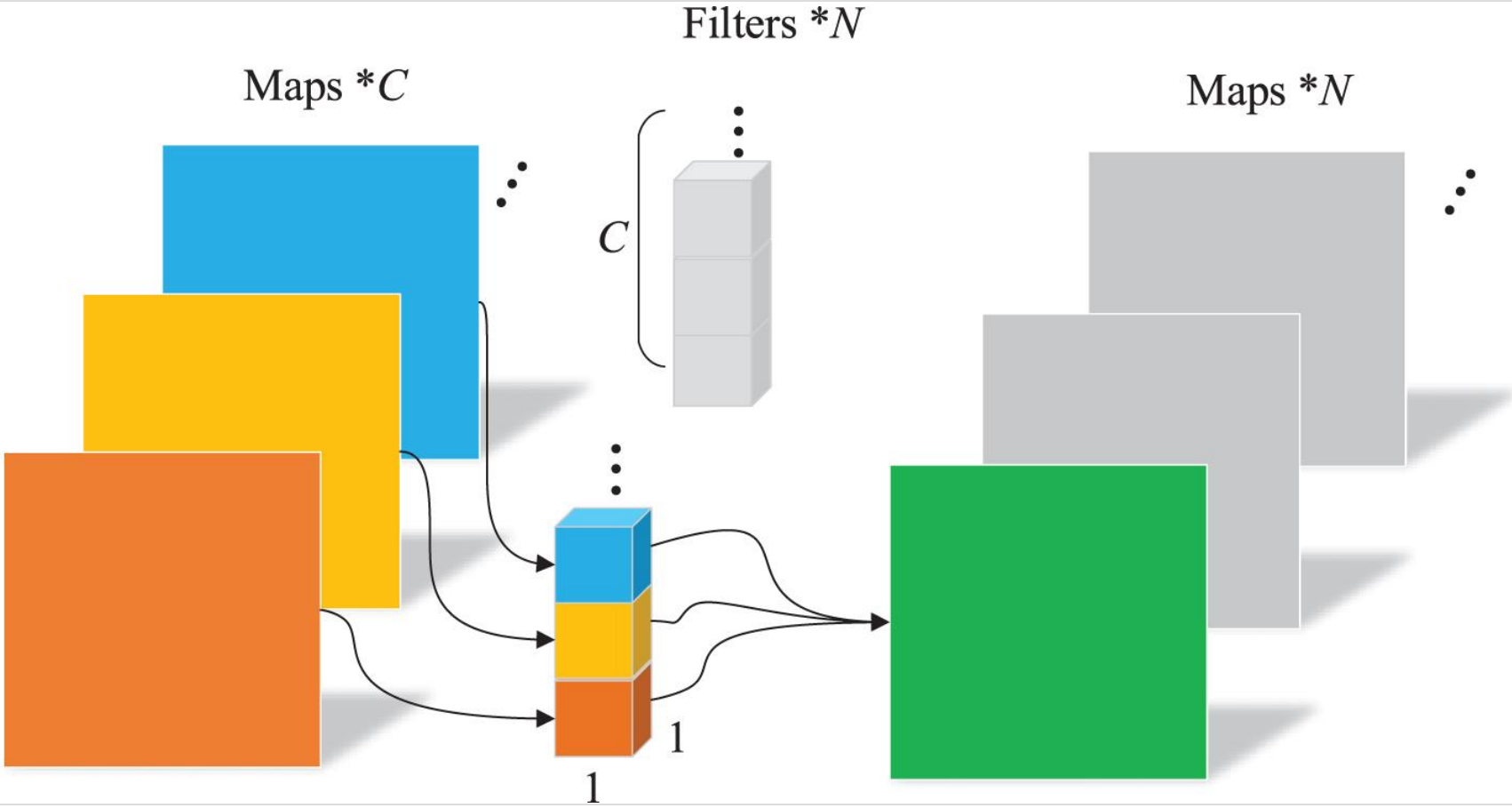
Depth-Wise Separable Convolution (2/4)

- **Depth-wise convolution**
  - 是一種group convolution
    - **# of groups = # of input channels**
    - 將每“片”feature maps都用一”片”kernel來做convolution
      - “片”：厚度 = 1
    - # of input channels = # of output channels



Depth-Wise Separable Convolution (3/4)

- **Point-wise convolution**
  - 就是**1x1 convolution**
    - 將channel數調整成目標channel數



Depth-Wise Separable Convolution (4/4)

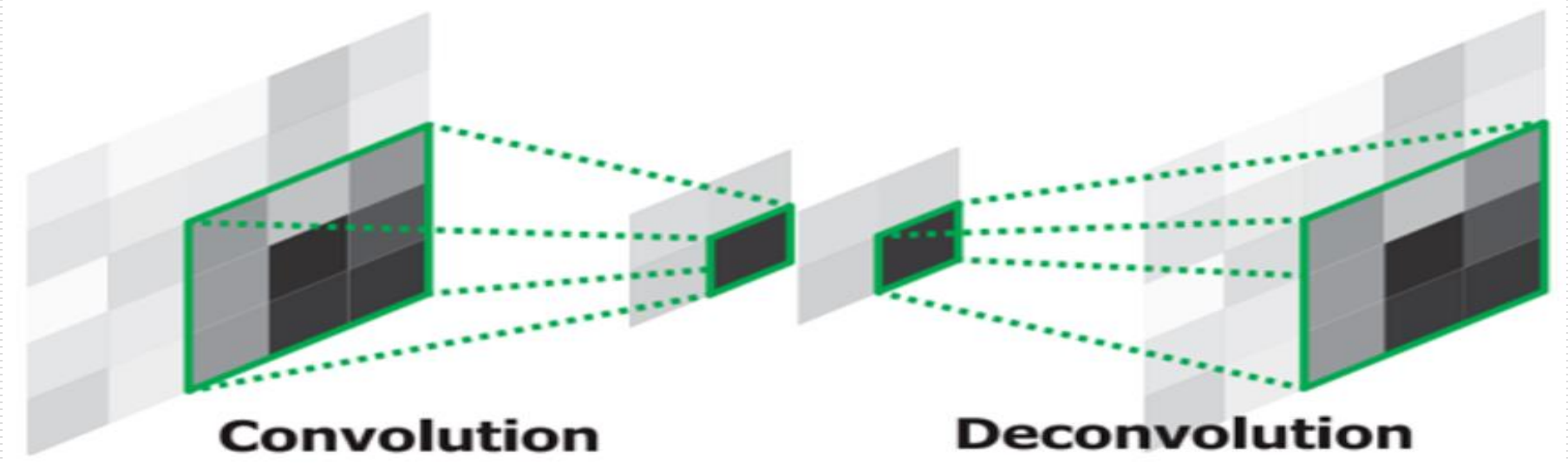
- Implement via the *groups* argument of nn.Conv2d

```
class DepthwiseSeparableConv(nn.Module):
    def __init__(self, inp_ch, out_ch, k, strd, pad):
        super(DepthwiseSeparableConv, self).__init__()

        self.depthwise_conv = nn.Conv2d(inp_ch, inp_ch, kernel_size=k, stride=strd, padding=pad, groups=inp_ch)
        self.pointwise_conv = nn.Conv2d(inp_ch, out_ch, kernel_size=1)

    def forward(self, input_tensor):
        output = self.depthwise_conv(input_tensor)
        output = self.pointwise_conv(output)
        return output
```

Transpose Convolution (1/4)

CLASS torch.nn.ConvTranspose2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

- Convolution which perform **up-sampling**
- 一般convolution是用於縮小feature map大小(H、W)，但Transpose convolution 是用於放大feature map大小，故常稱為deconvolution。



Transpose Convolution (2/4)

- Why named “Transpose”?
  - Normal convolution
    - ifmap: 4x4, kernel(**w**): 3x3, stride: 1, ofmap: 2x2
      - After flatten
        - Ifmap(***x***): 16x1, ofmap(***y***): 4x1
    - Linear algebra
      - Convolution operation equals to a matrix *C*
        - $\boldsymbol{C}x = y$
        - $\boldsymbol{C}$: *4x16*

$$\mathbf{w} = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,2} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

$$\begin{matrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{matrix}$$

Transpose Convolution (3/4)

- Why named “Transpose”?
  - Transpose convolution
    - ifmap: 2x2, kernel(**w**): 3x3, stride: 1, ofmap: 4x4
      - After flatten
        - Ifmap(***y***): 4x1, ofmap(***x***): 16x1
    - Linear algebra
      - Transpose convolution operation equals to a matrix ***C’***
        - $\boldsymbol{C}'y = x$
        - $\boldsymbol{C}'$: *16x4*
      - 因為$\boldsymbol{C}'$的shape和$\boldsymbol{C}$的轉置相同，所以稱為transpose convolution。

Transpose Convolution (4/4)

- Detail implementation
  - First generate new ifmap by **interpolation**
  - And then do the convolution with the same kernel

```
>>> # exact output size can be also specified as an argument
>>> input = torch.randn(1, 16, 12, 12)
>>> downsample = nn.Conv2d(16, 16, 3, stride=2, padding=1)
>>> upsample = nn.ConvTranspose2d(16, 16, 3, stride=2, padding=1)
>>> h = downsample(input)
>>> h.size()
torch.Size([1, 16, 6, 6])
>>> output = upsample(h, output_size=input.size())
>>> output.size()
torch.Size([1, 16, 12, 12])
```

References

References (1/2)

- 台大李宏毅老師機器學習課程
  - https://speech.ee.ntu.edu.tw/~tlkagk/
- 聊一聊深度學習的activation function
  - https://zhuanlan.zhihu.com/p/25110450
- Gradient descent with momentum
  - https://zhuanlan.zhihu.com/p/34240246
- Gradient descent optimization algorithms
  - https://reurl.cc/nz8Kkn
- CNN中的數據增強簡單總結
  - https://zhuanlan.zhihu.com/p/104992391

References (2/2)

- PyTorch 官方文件
  - PyTorch documentation — PyTorch 2.0
- Group Convolution
  - CSDN Group Convolution
- Transpose Convolution
  - CSDN Transpose Convolution
  - 搞懂 deconvolution、transposed convolution、sub-pixel or fractional convolution - shine-lee (cnblogs.com)
- Depth-wise Separable Convolution
  - 深度學習-MobileNet (Depthwise separable convolution) | by Tommy Huang | Medium

Homework

HW1

- HW1 (25%) :
  - 於eng05 server(140.113.225.245)上執行
    - $ tar -xvf /DATA/AI_training_HW_2024/HW2.tar
  - In lec2_hw1.py, calculate the input/output size of each layer in NetHW and the total # of parameters of this model.
  - 直接把答案打在"lec2_hw1.py"的註解中
  - File name should be "帳號_lec2_hw1.py"

HW2

- Report (75%, 25% for each)
- Problem1:
  - Please explain what is the residual block (two types) and give the pros and cons of each.
- Problem2:
  - Please explain what is the receptive field and how to adjust the receptive field in the neural network.
- Problem3:
  - Please give some methods to achieve feature map upsampling. Explain them with codes and images.
- File name should be “帳號_lec2_hw2.pdf”

Submission

- Deadline
  - July 15, 2024, 23:59
- Format
  - 帳號_lec2_hw1.py
  - 帳號_lec2_hw2.pdf

- 繳交至FB社團公告中的google表單

Thank you