

Python深度學習實戰

丘祐璋
David Chiu

關於我



**大數軟體有限公司創辦人
前趨勢科技工程師**

大數學堂

<http://www.largitdata.com/>

粉絲頁

<https://www.facebook.com/largitdata>

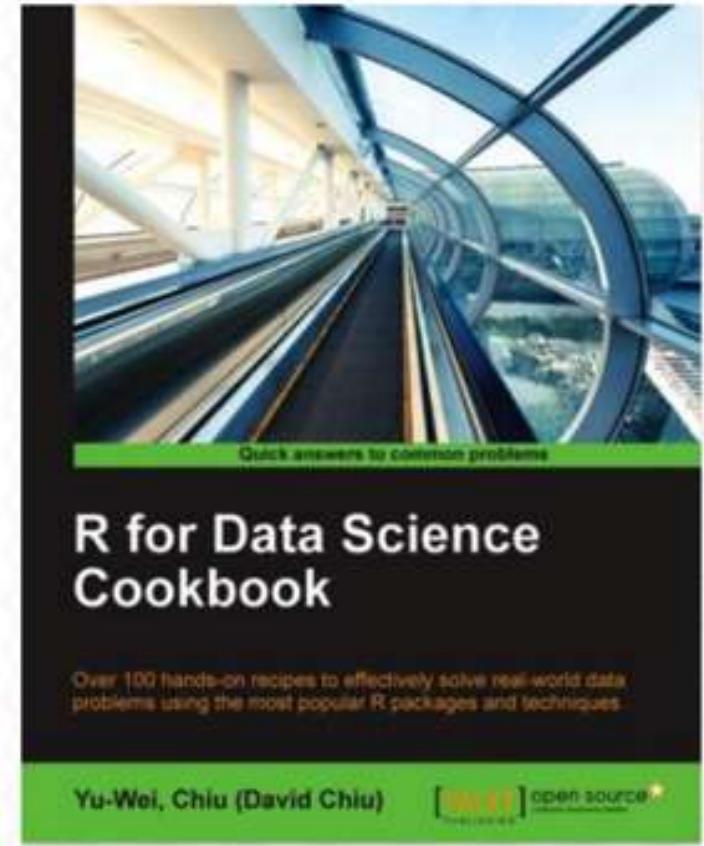
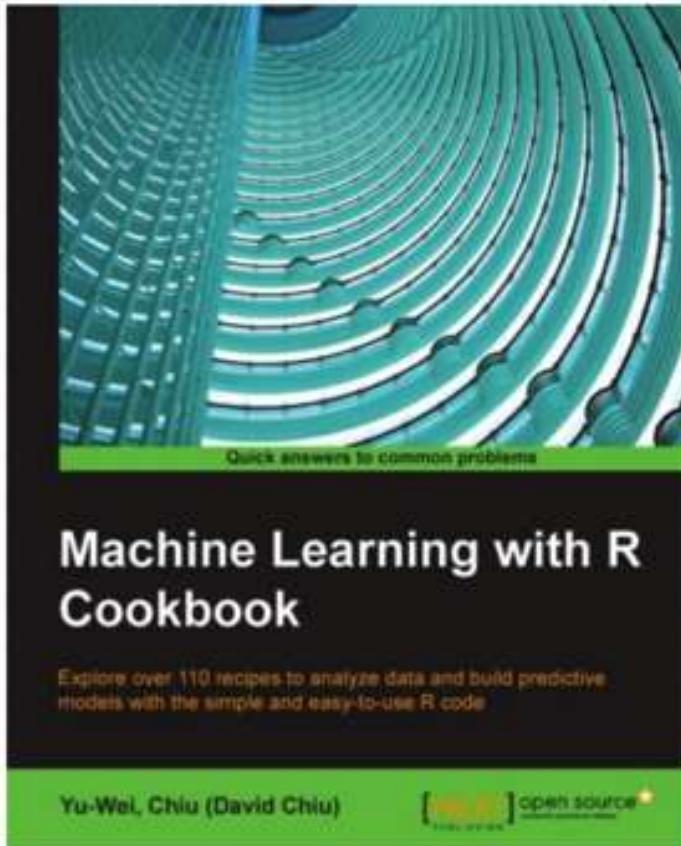
R for Data Science Cookbook

<https://www.packtpub.com/big-data-and-business-intelligence/r-data-science-cookbook>

Machine Learning With R Cookbook

<https://www.packtpub.com/big-data-and-business-intelligence/machine-learning-r-cookbook>

Machine Learning With R Cookbook (机器学习与R语言实战) & R for Data Science Cookbook (数据科学 : R语言实现)



Author: Yu-Wei (David) Chiu

課程補充資料

■ 課程補充資料及程式碼皆放置於以下連結

□ <https://github.com/ywchiu/tibamedl>



AlphaGO
使用深度學習技術打敗頂尖棋手



AlphaGO Zero
只需要三天就能熟悉第一代AlphaGo
所有的圍棋知識

A close-up, slightly blurred photograph of the front right side of a Waymo self-driving car. The car is white with a blue stripe along the bottom edge. The Waymo logo, consisting of a green stylized 'W' above the word 'WAYMO' in a smaller, sans-serif font, is visible on the side of the car. The background shows a road with trees and a clear sky.

Google 成立 Waymo
打造全球第一個真無人車上路測試



什麼是人工智能(A.I.)?

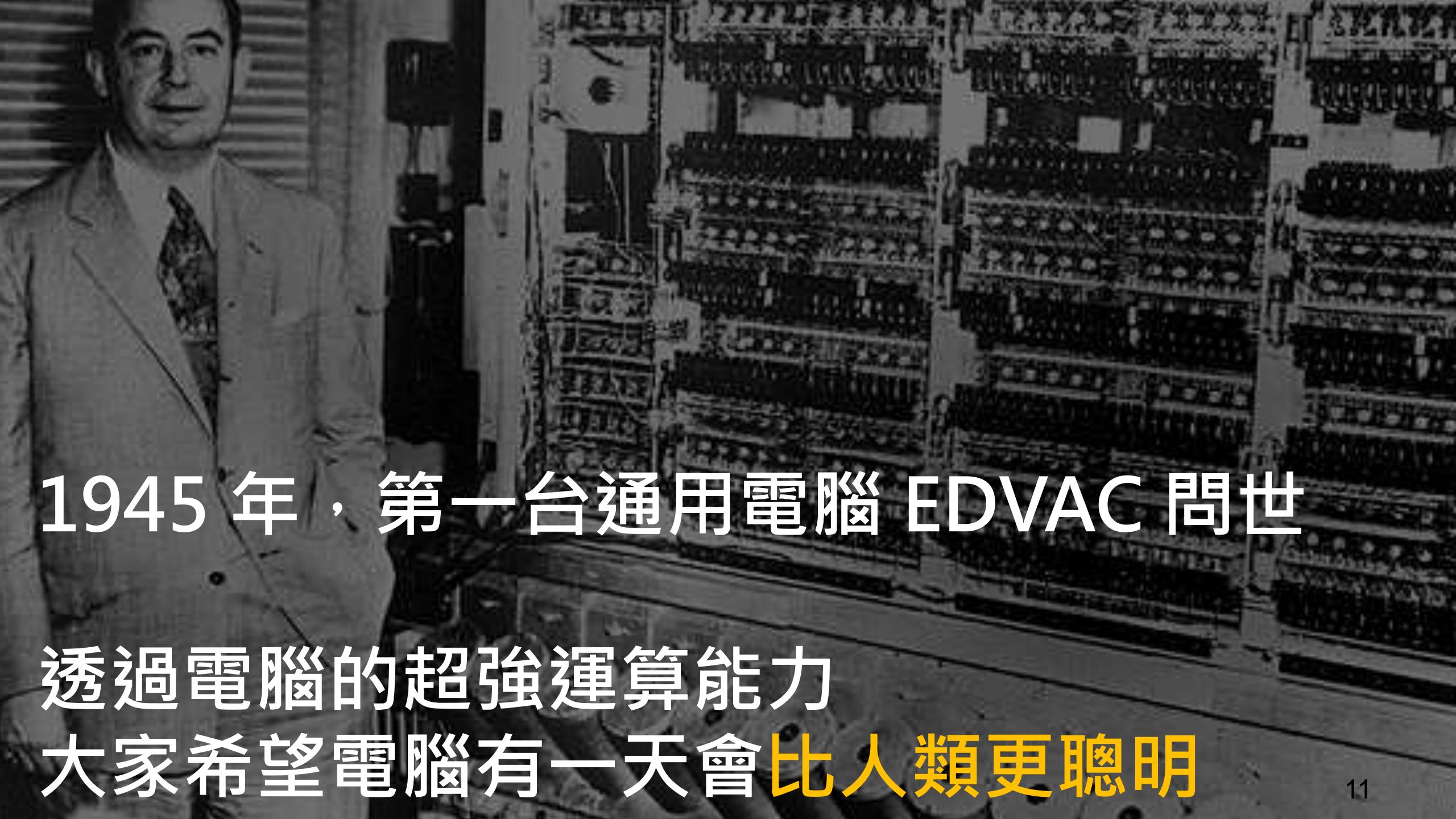


模仿遊戲》(The Imitation Game)，講述了計算機科學之父 艾倫·圖靈的傳奇人生，故事主要聚焦於圖靈協助盟軍破譯德國密碼系統恩尼格瑪密碼機，從而扭轉二戰戰局的經歷

INSPIRATIONAL STORY
OF
ALAN TURING
FATHER OF MODERN COMPUTING

讓我問你一個問題，你必須
作答。根據你的回答，我將
能判定：你是人，還是機器





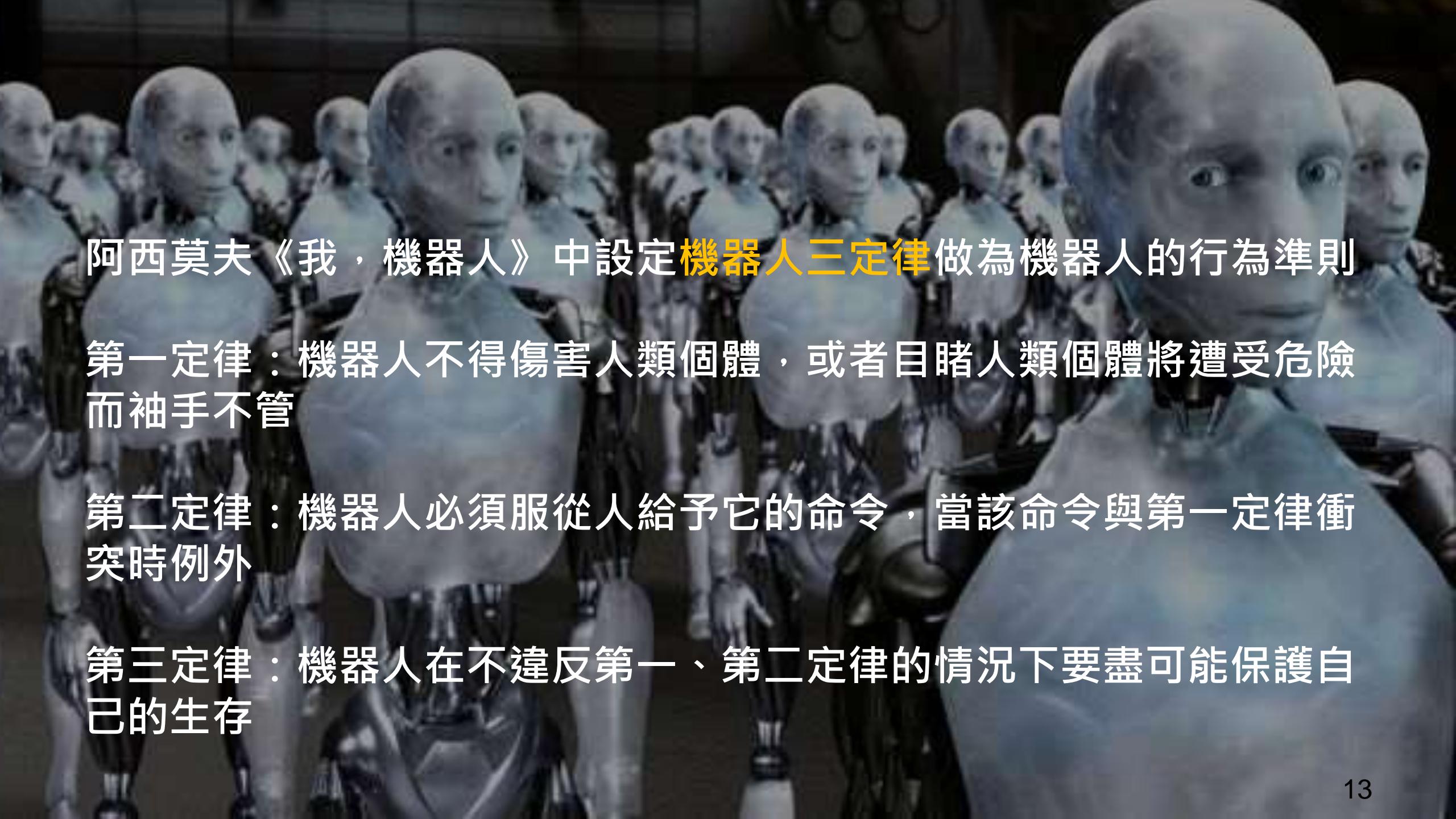
1945 年，第一台通用電腦 EDVAC 問世

透過電腦的超強運算能力
大家希望電腦有一天會**比人類更聰明**



1956 年舉辦于達特茅斯的一場研討會

紐厄爾 (Newell)、西蒙 (Simon) 展示了 “全世界第一個人工智慧程式” 邏輯理論家 (Logic Theorist)



阿西莫夫《我，機器人》中設定**機器人三定律**做為機器人的行為準則

第一定律：機器人不得傷害人類個體，或者目睹人類個體將遭受危險而袖手不管

第二定律：機器人必須服從人給予它的命令，當該命令與第一定律衝突時例外

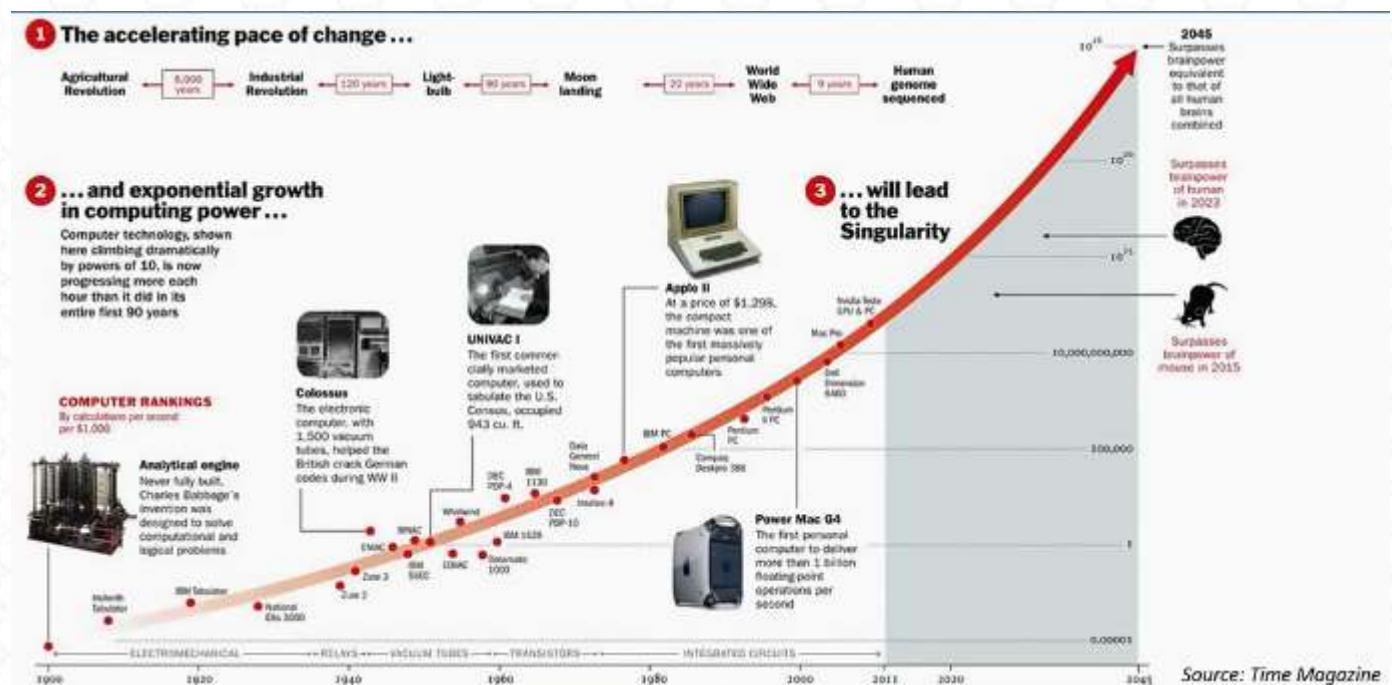
第三定律：機器人在不違反第一、第二定律的情況下要盡可能保護自己的生存

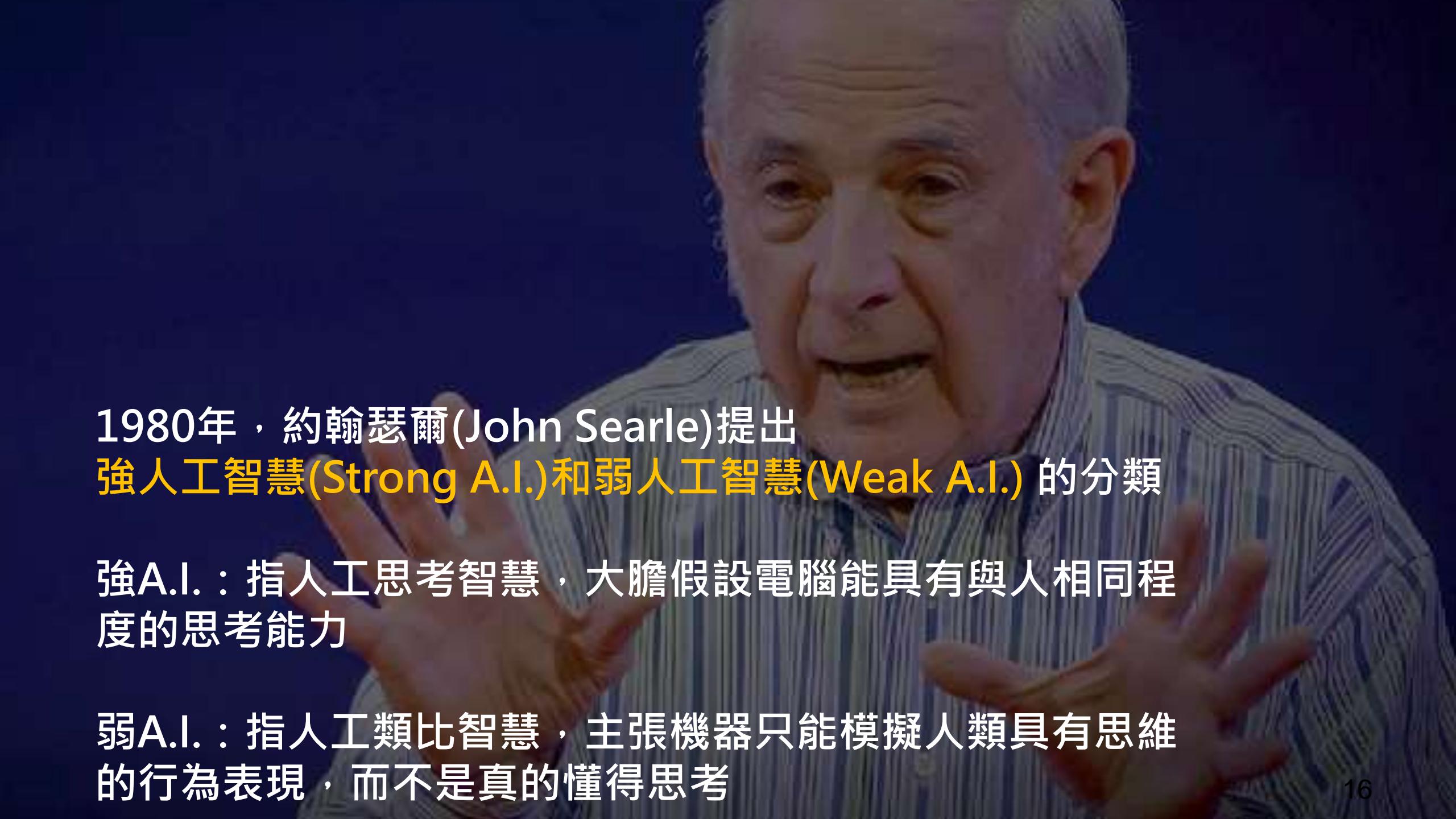
硬體成本逐漸降低



莫爾定律

- 當價格不變時，集成電路上可容納的元器件的數目，約每隔18-24個月便會增加一倍，性能也將提升一倍
- 換言之，每一美元所能買到的電腦性能，將每隔18-24個月翻一倍以上





1980年，約翰瑟爾(John Searle)提出
強人工智慧(Strong A.I.)和弱人工智慧(Weak A.I.) 的分類

強A.I.：指人工思考智慧，大膽假設電腦能具有與人相同程度的思考能力

弱A.I.：指人工類比智慧，主張機器只能模擬人類具有思維的行為表現，而不是真的懂得思考

強人工智能

模仿人類推理過程的思考模式，需要百分之百確定的事實配合，實務上應用困難

弱人工智能 - 預測模型

- 是否能用統計機率學來處理人工智能的問題呢？
- e.g. 根據個人行為的特定機制 (是否點擊廣告)，預測模型把個人特性(身高，性別，職業)當輸入資料，提供預測分數當產出資料，分數越高，產生該行為的機會越高。



讓機器做預測分析

- 從歷史資料建構預測性模型，以預測未知值

The diagram illustrates a dataset for building a predictive model. The table has five columns: Name, Balance, Age, Employed, and Bad debt. The last column, 'Bad debt', is highlighted in green and labeled 'Target', indicating it is the variable to be predicted. The other four columns are labeled 'Attribute'.

Name	Balance	Age	Employed	Bad debt
Mike	20000	42	N	Y
Mary	25000	33	Y	N
Claudio	115000	40	N	N
Robert	29000	23	Y	Y
Dora	72000	31	N	N

用機率與規則產生預測模型

■ 線性模型

e.g. 針對一個化妝品廣告，對女性的吸引力可以給予權重90%，男性權重只有10%，以權重搭配個人點擊機率(15%)可以算出對該使用者推薦的分數(或機率)

女性13.5%，男性1.5%

■ 規則模型

e.g.

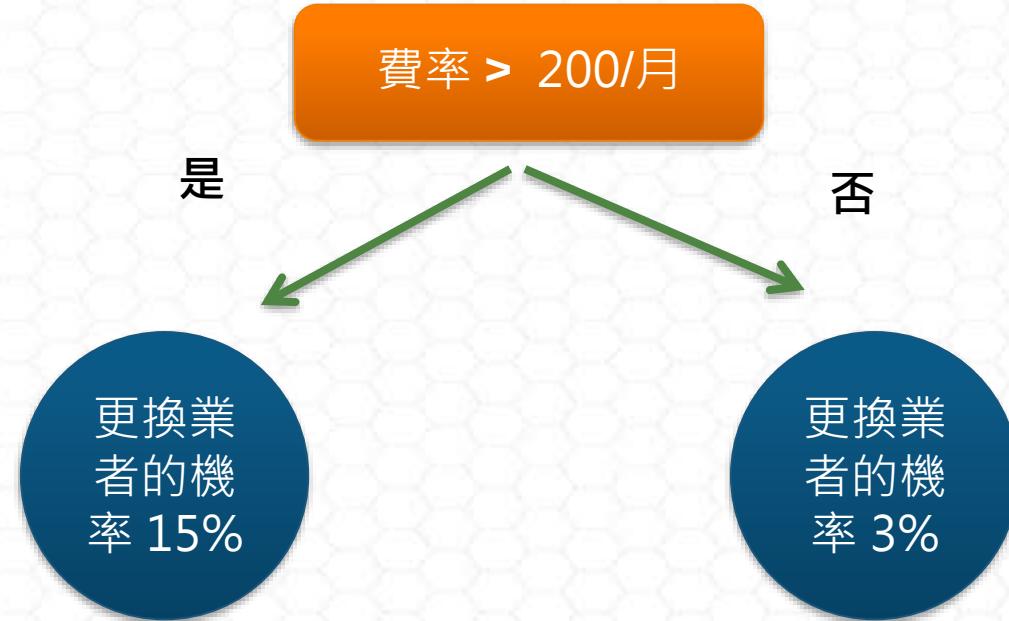
假使使用者為女性

而且月收入高達3萬以上

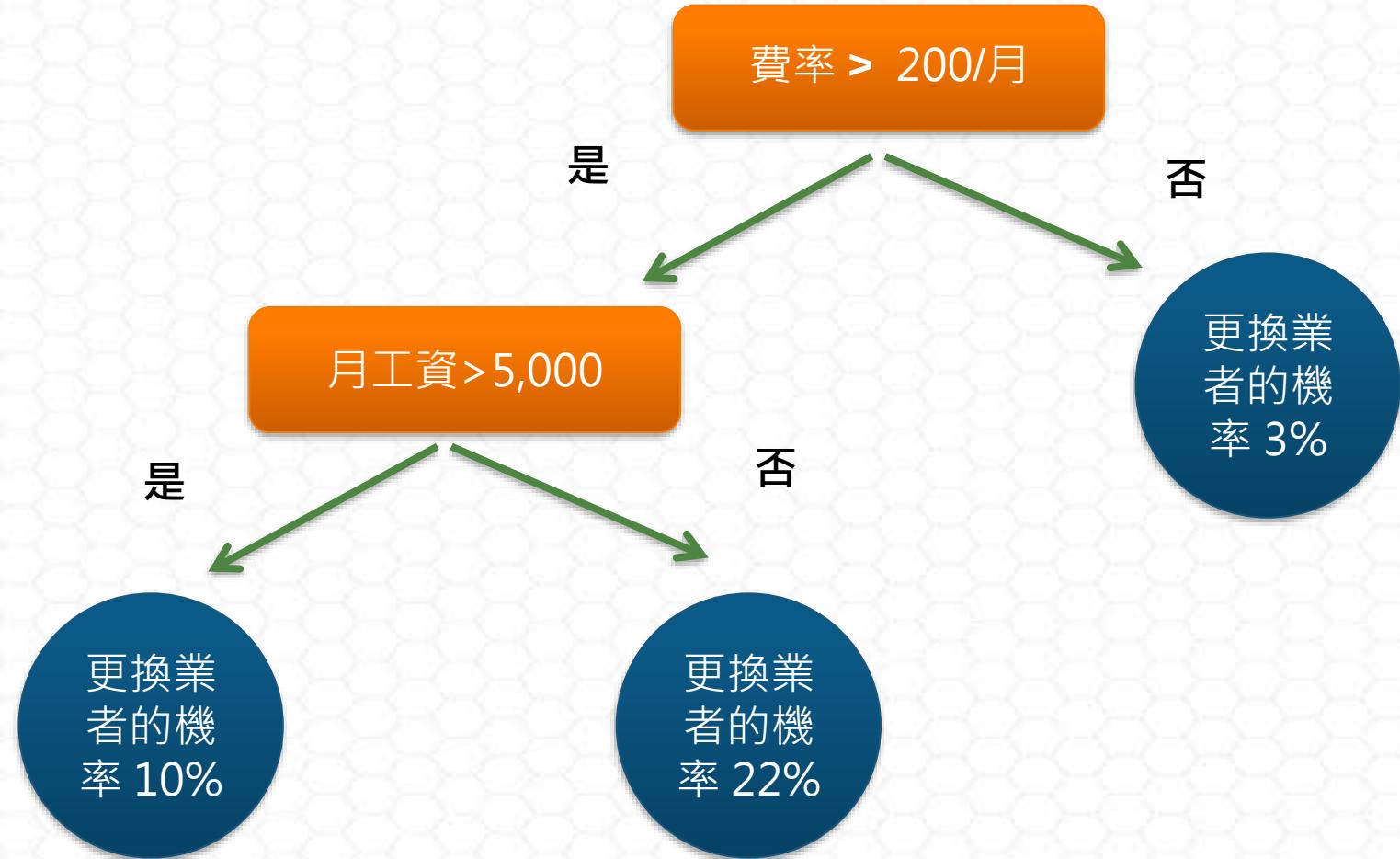
而且還沒看過這廣告

點擊機率為11%

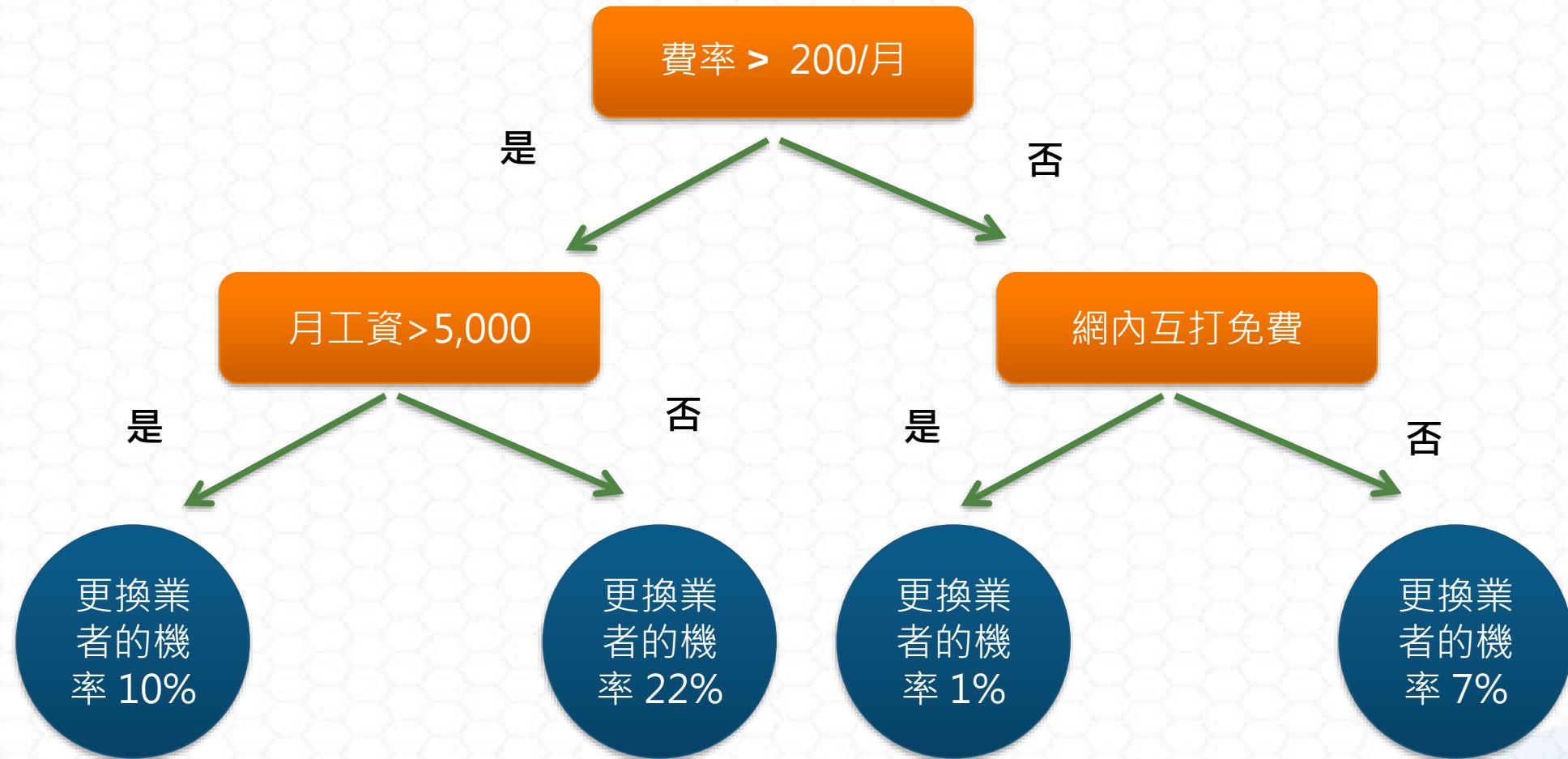
簡單的分類問題(決策樹)



簡單的分類問題(決策樹)



簡單的分類問題(決策樹)



機器學習

■ 機器學習的目的是：歸納（ Induction ）

- 從詳細事實到一般通論

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E

-- Tom Mitchell (1998)

如何找出有效的預測模型

- 一開始都從一個簡單的模型開始
- 藉由不斷輸入訓練資料，修改模型
- 以不斷提升預測績效

機器學習步驟

使用者行為



■ 監督式學習 (Supervised Learning)

- 回歸分析 (Regression)
- 分類問題 (Classification)

■ 非監督式學習 (Unsupervised Learning)

- 降低維度 (Dimension Reduction)
- 分群問題 (Clustering)

■ 半監督式學習 (Semi-supervised Learning)

使用監督式學習進行預測

■ 回歸分析

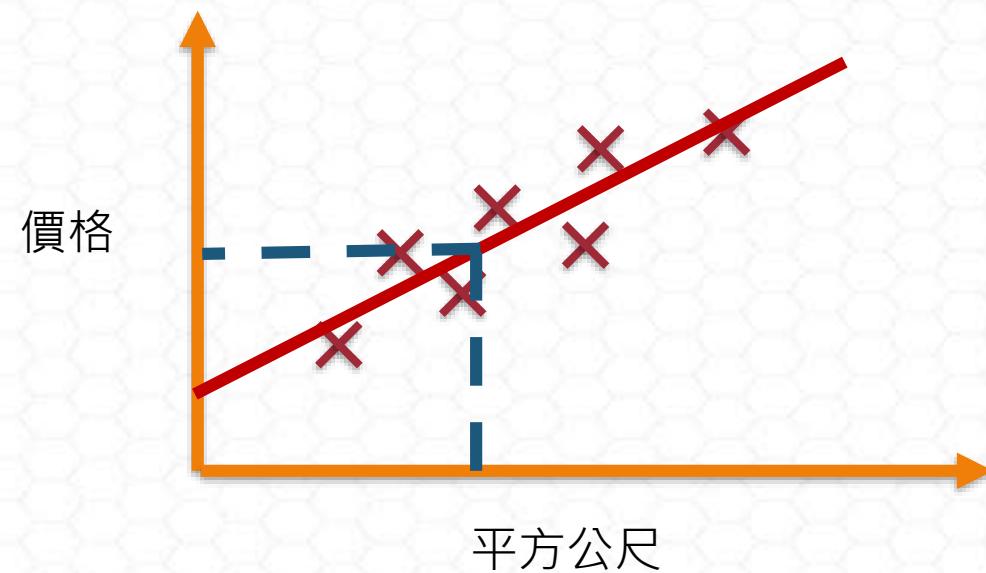
- 使用一組已知對應值的資料產生的模型，預測新資料的對應值
- e.g. 房價預測

■ 分類問題

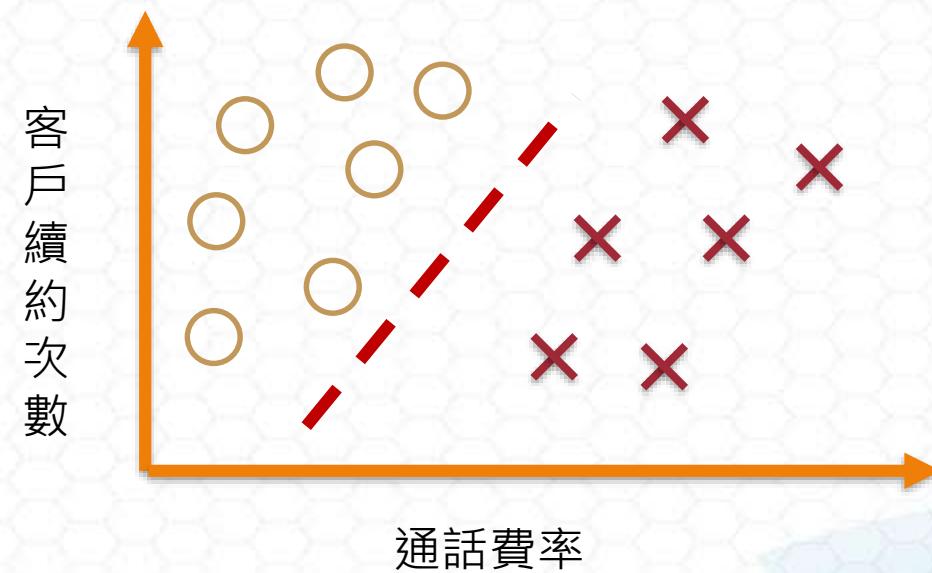
- 根據已知標籤的訓練資料集(Training Set)，產生一個新模型，用以預測測試資料集(Testing Set)的標籤。
- e.g. 客戶流失分析

使用監督式學習進行預測

回歸分析



分類問題



使用非監督式學習找出隱藏的架構

■ 分群問題

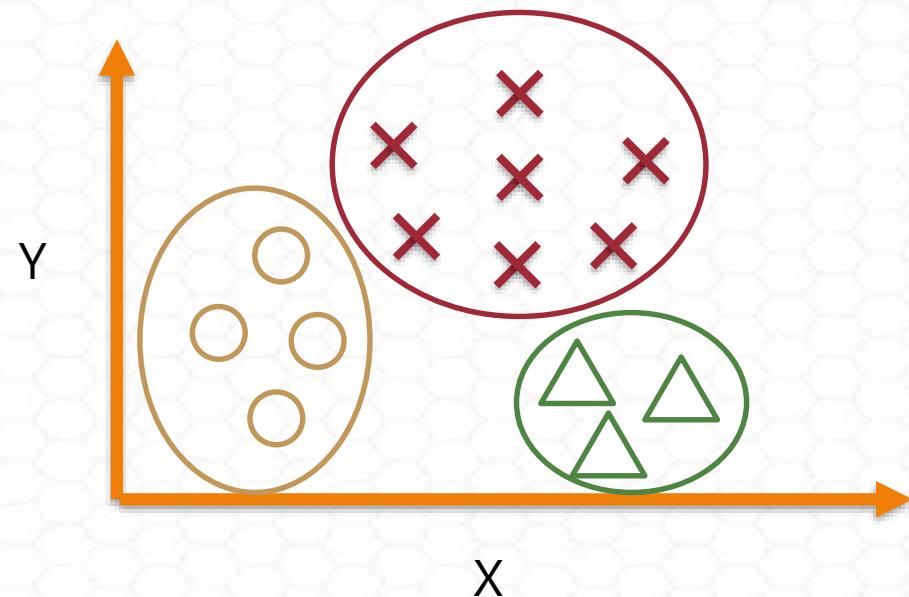
- 物以類聚（近朱者赤，近墨者黑）
- e.g. 將客戶分層

■ 降低維度

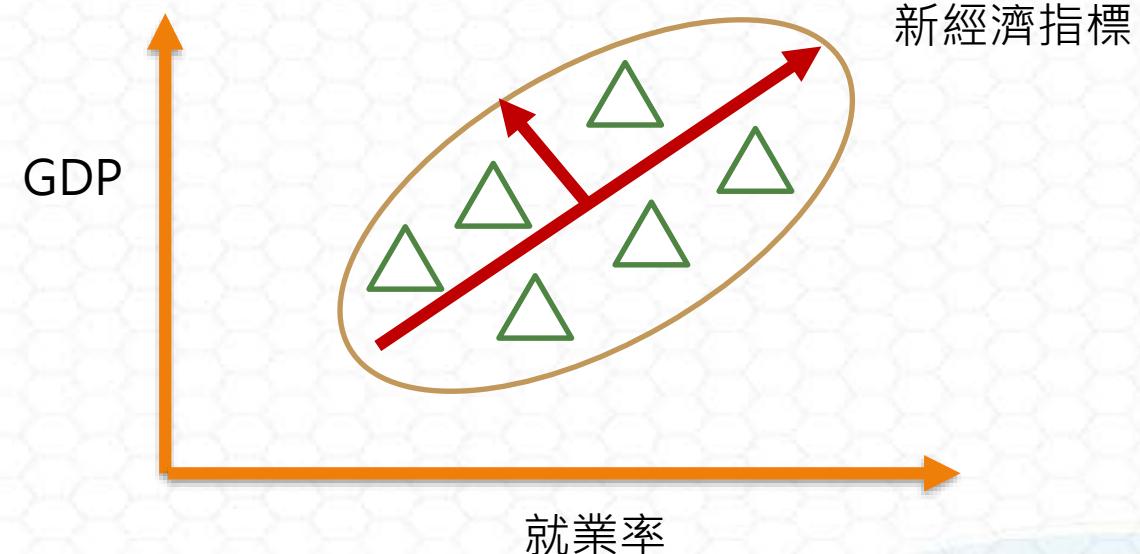
- 產生一有最大變異數的欄位元線性組合，可用來降低原本問題的維度與複雜度
- e.g. 濃縮用到的特徵，編纂成一個新指標

使用非監督式學習找出隱藏的架構

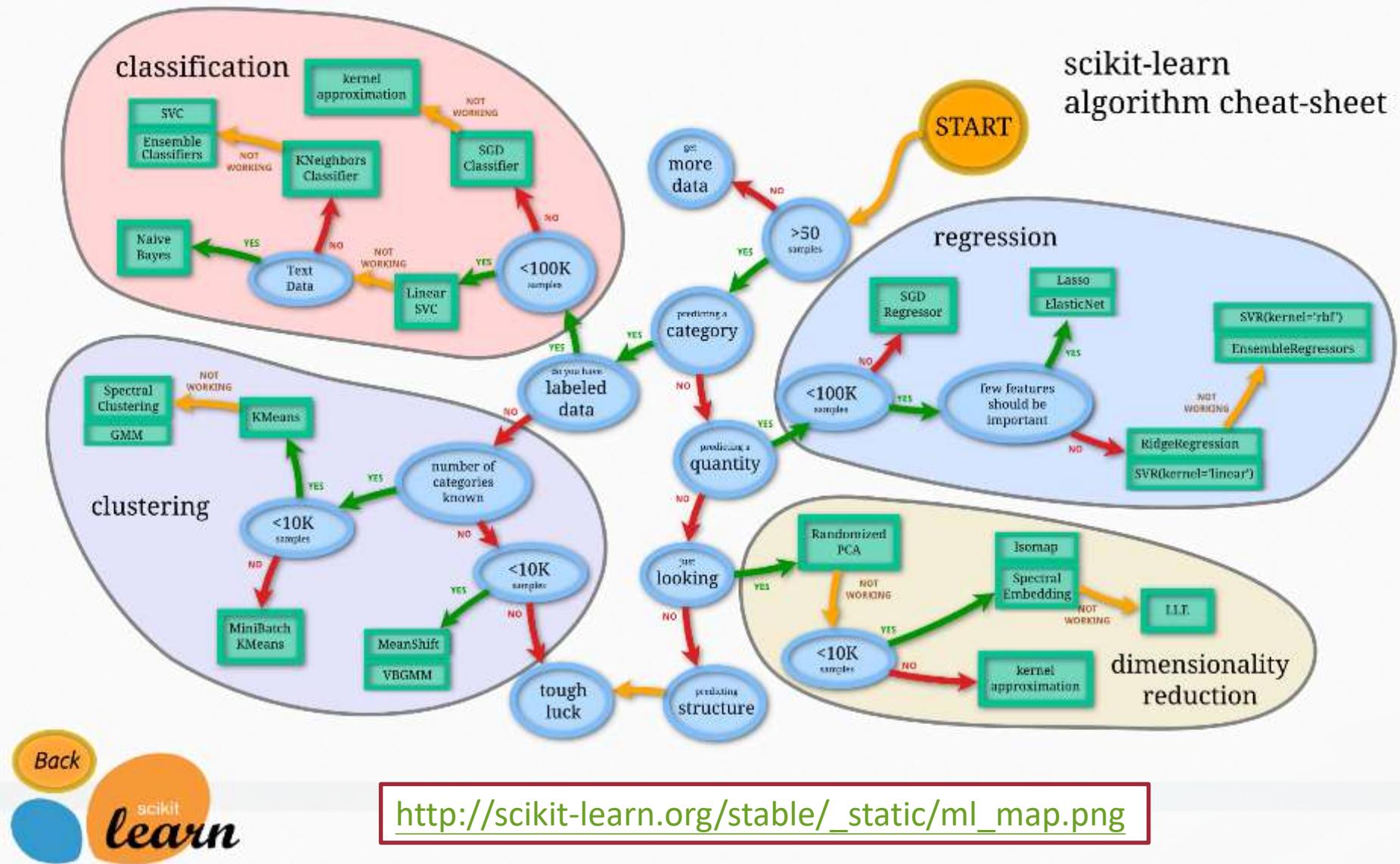
分群問題



降低維度



機器學習地圖



http://scikit-learn.org/stable/_static/ml_map.png

監督式學習 - 規則模型

用機率與規則產生預測模型

■ 規則模型

e.g.

假使使用者為女性

而且月收入高達3萬以上

而且還沒看過這廣告

點擊機率為11%

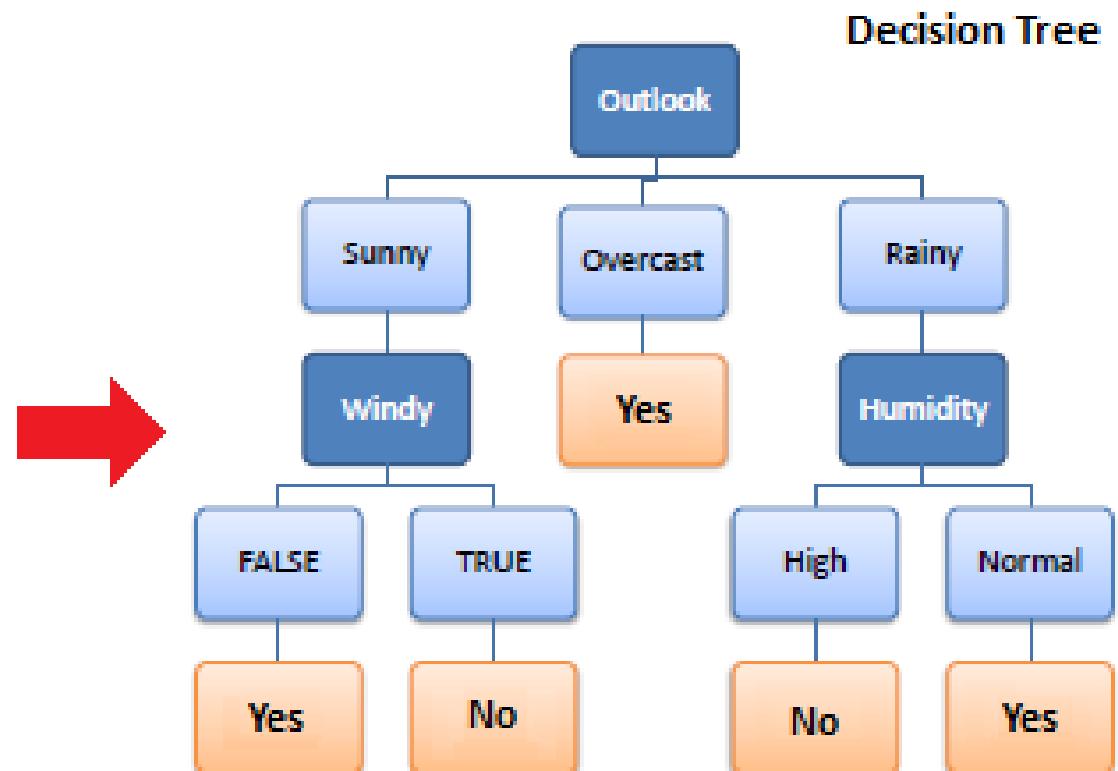
■ 線性模型

e.g. 針對一個化妝品廣告，對女性的吸引力可以給予權重90%，男性權重只有10%，以權重搭配個人點擊機率(15%)可以算出對該使用者推薦的分數(或機率)

女性13.5%，男性1.5%

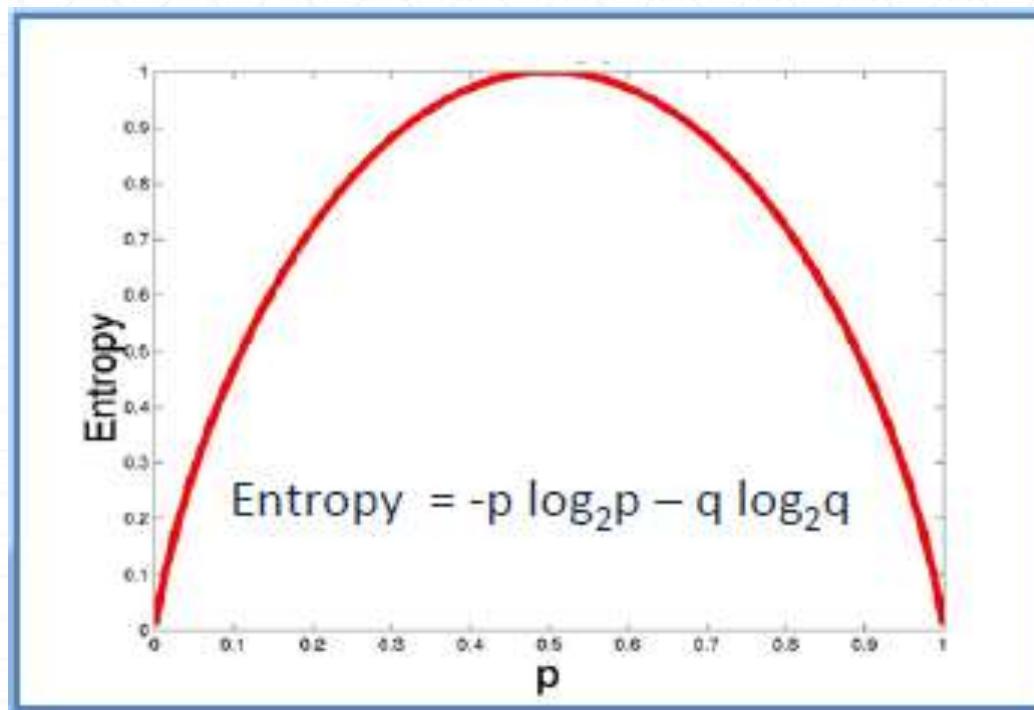
決策樹

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



信息熵(entropy)

- 用於計算一個系統中的失序現象，也就是計算該系統混亂（糾結）的程度



$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

單一變數的計算

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5



$$\begin{aligned}\text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94\end{aligned}$$

多變數的計算

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned} E(\text{PlayGolf}, \text{Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\ &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\ &= 0.693 \end{aligned}$$

Information Gain

- 根據分割 (split) 後，所減少的熵 因此做分割時，會尋找最大的資訊增益
- 計算Entropy

$$\begin{aligned}\text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= - (0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94\end{aligned}$$

計算 Information Gain

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

$$G(\text{PlayGolf}, \text{Outlook}) = E(\text{PlayGolf}) - E(\text{PlayGolf}, \text{Outlook})$$

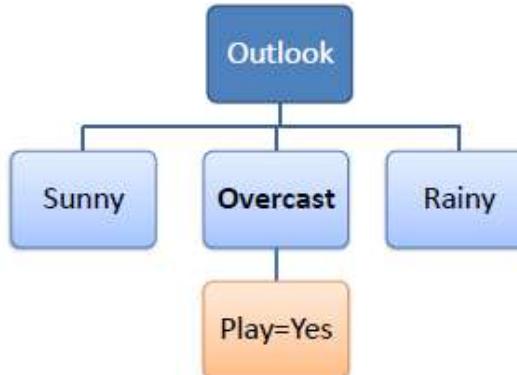
$$= 0.940 - 0.693 = 0.247$$

選擇有最大Information Gain的屬性

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

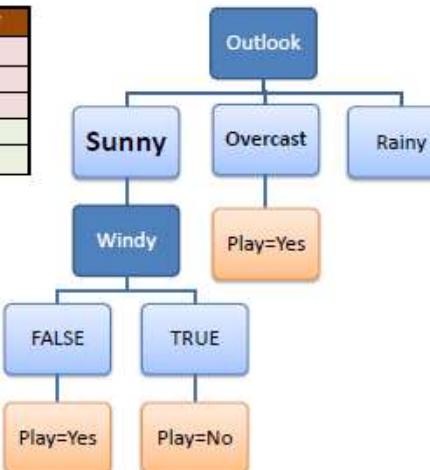
選擇子節點與分割節點

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes
Hot	High	FALSE	Yes



Entropy為0 則為子節點

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Entropy非0 則為分割節點

決策樹如同IF ... ELSE

$R_1: \text{IF} (\text{Outlook}=\text{Sunny}) \text{ AND } (\text{Windy}=\text{FALSE}) \text{ THEN Play}=\text{Yes}$

$R_2: \text{IF} (\text{Outlook}=\text{Sunny}) \text{ AND } (\text{Windy}=\text{TRUE}) \text{ THEN Play}=\text{No}$

$R_3: \text{IF} (\text{Outlook}=\text{Overcast}) \text{ THEN Play}=\text{Yes}$

$R_4: \text{IF} (\text{Outlook}=\text{Rainy}) \text{ AND } (\text{Humidity}=\text{High}) \text{ THEN Play}=\text{No}$

$R_5: \text{IF} (\text{Outlook}=\text{Rain}) \text{ AND } (\text{Humidity}=\text{Normal}) \text{ THEN Play}=\text{Yes}$



如何分類鳶尾花 (iris)

■ https://en.wikipedia.org/wiki/Iris_flower_data_set



Iris setosa



Iris versicolor



Iris virginica

如何從花萼與花瓣長寬分辨花種？

Fisher's Iris Data				
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	<i>I. setosa</i>
4.9	3.0	1.4	0.2	<i>I. setosa</i>
4.7	3.2	1.3	0.2	<i>I. setosa</i>
4.6	3.1	1.5	0.2	<i>I. setosa</i>
5.0	3.6	1.4	0.2	<i>I. setosa</i>
5.4	3.9	1.7	0.4	<i>I. setosa</i>
4.6	3.4	1.4	0.3	<i>I. setosa</i>
5.0	3.4	1.5	0.2	<i>I. setosa</i>



使用sklearn建立決策樹

■ 讀取數據

```
from sklearn.datasets import load_iris  
iris = load_iris()
```

■ 建立模型

```
from sklearn import tree  
clf = tree.DecisionTreeClassifier()  
clf = clf.fit(iris.data, iris.target)
```

呼叫決策樹函式

■ 產生預測結果

```
predicted = clf.predict(iris.data)
```

將分類結果顯示在圖上

- 繪製成樹狀圖

```
from sklearn import tree  
tree.export_graphviz(clf, out_file='tree.dot')
```

- 將tree.dot的內容貼到以下網址做視覺化

<http://webgraphviz.com/>

建立決策邊界 (1)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn import tree

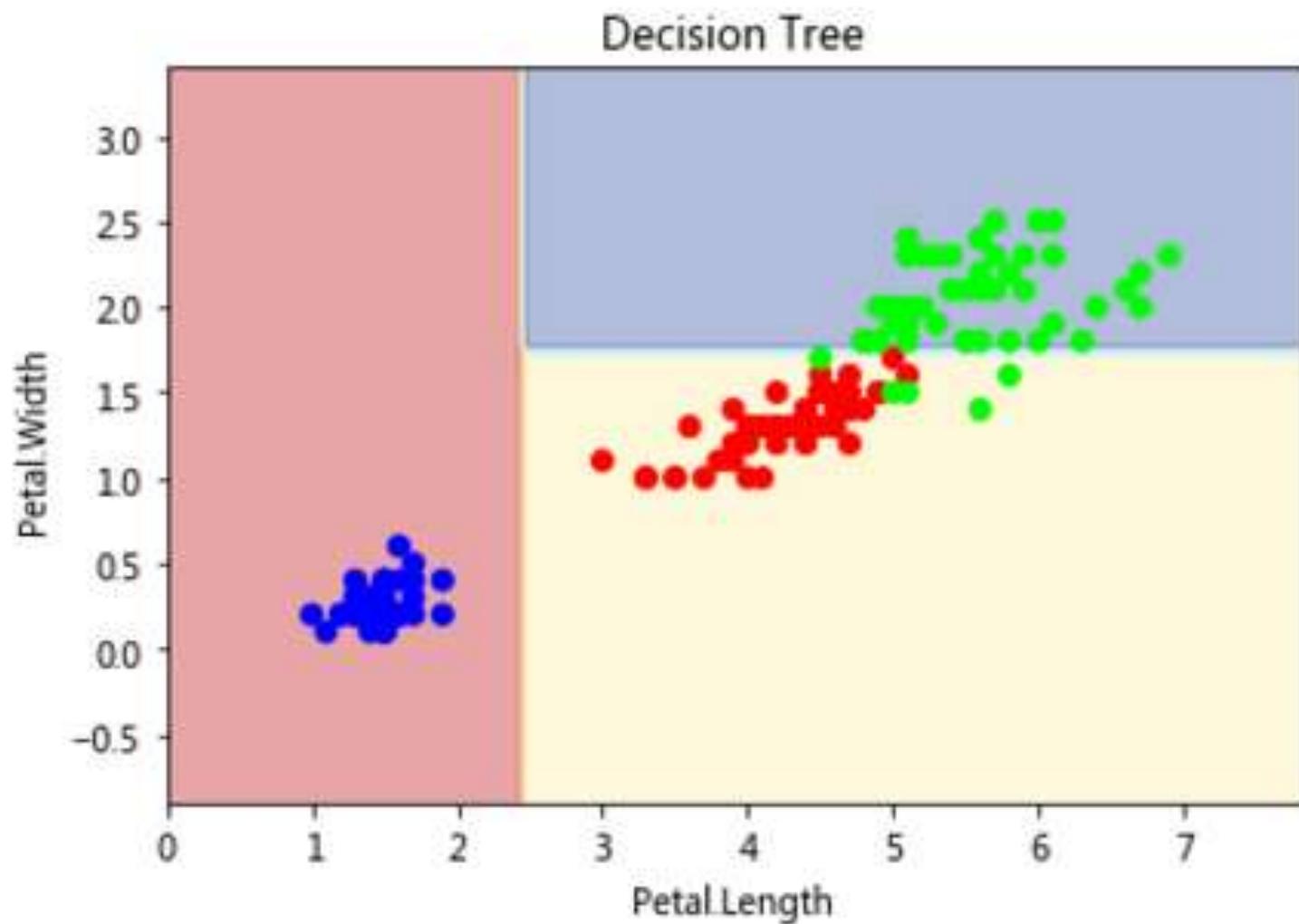
iris = load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

clf = tree.DecisionTreeClassifier(max_depth=2)
clf.fit(X, y)
```

建立決策邊界 (2)

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),  
                      np.arange(y_min, y_max, 0.1))  
  
plt.plot()  
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])  
Z = Z.reshape(xx.shape)  
plt.contourf(xx, yy, Z, alpha=0.4, cmap = plt.cm.rainbow)  
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=1, cmap = plt.cm.RdYlBu)  
plt.title('Decision Tree')  
plt.xlabel('Petal.Length')  
plt.ylabel('Petal.Width')  
plt.show()
```

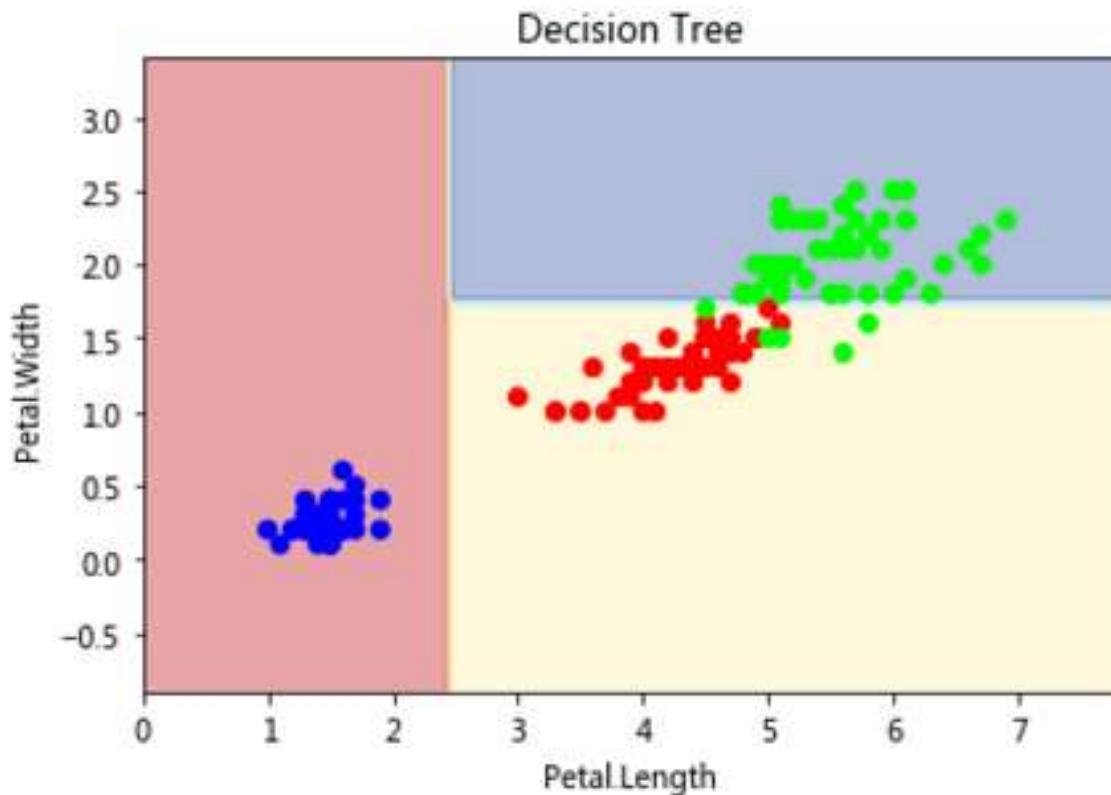
決策樹模型



監督式學習 – 線性模型

決策樹模型

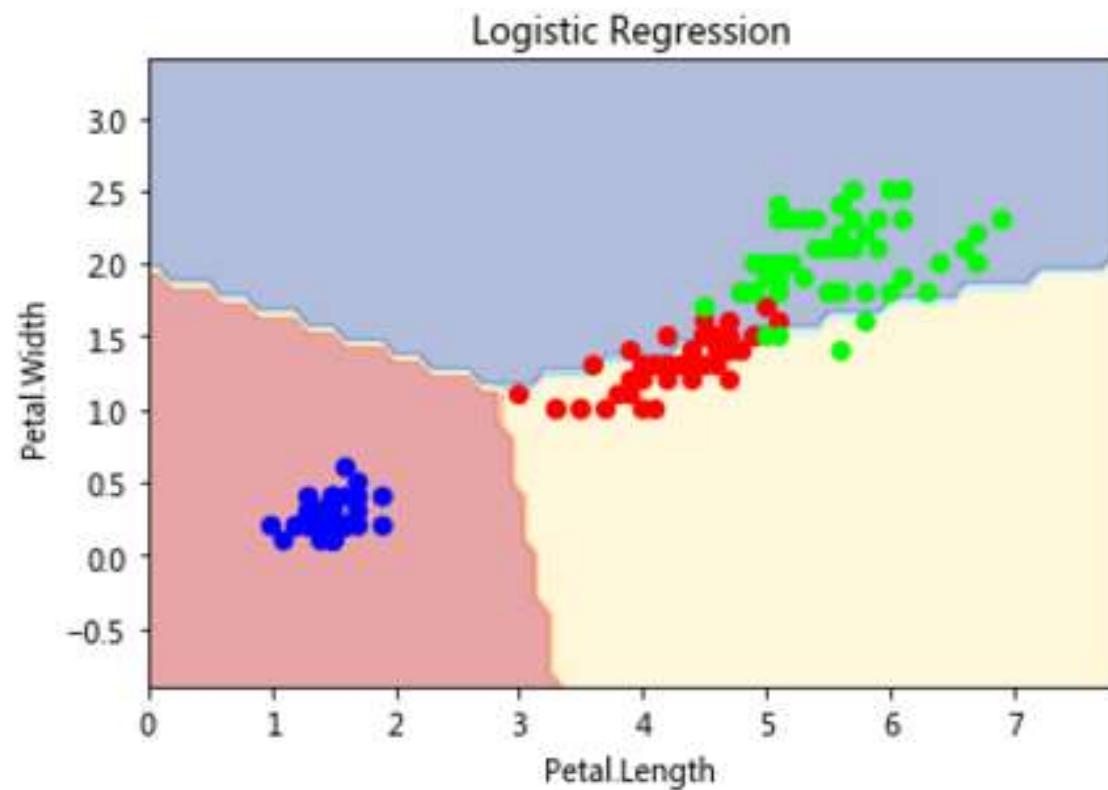
- 每次根據一個變數取決分隔



線性分類法

■ 線性判別

□ 如何切斜的刀



將模型配適到資料

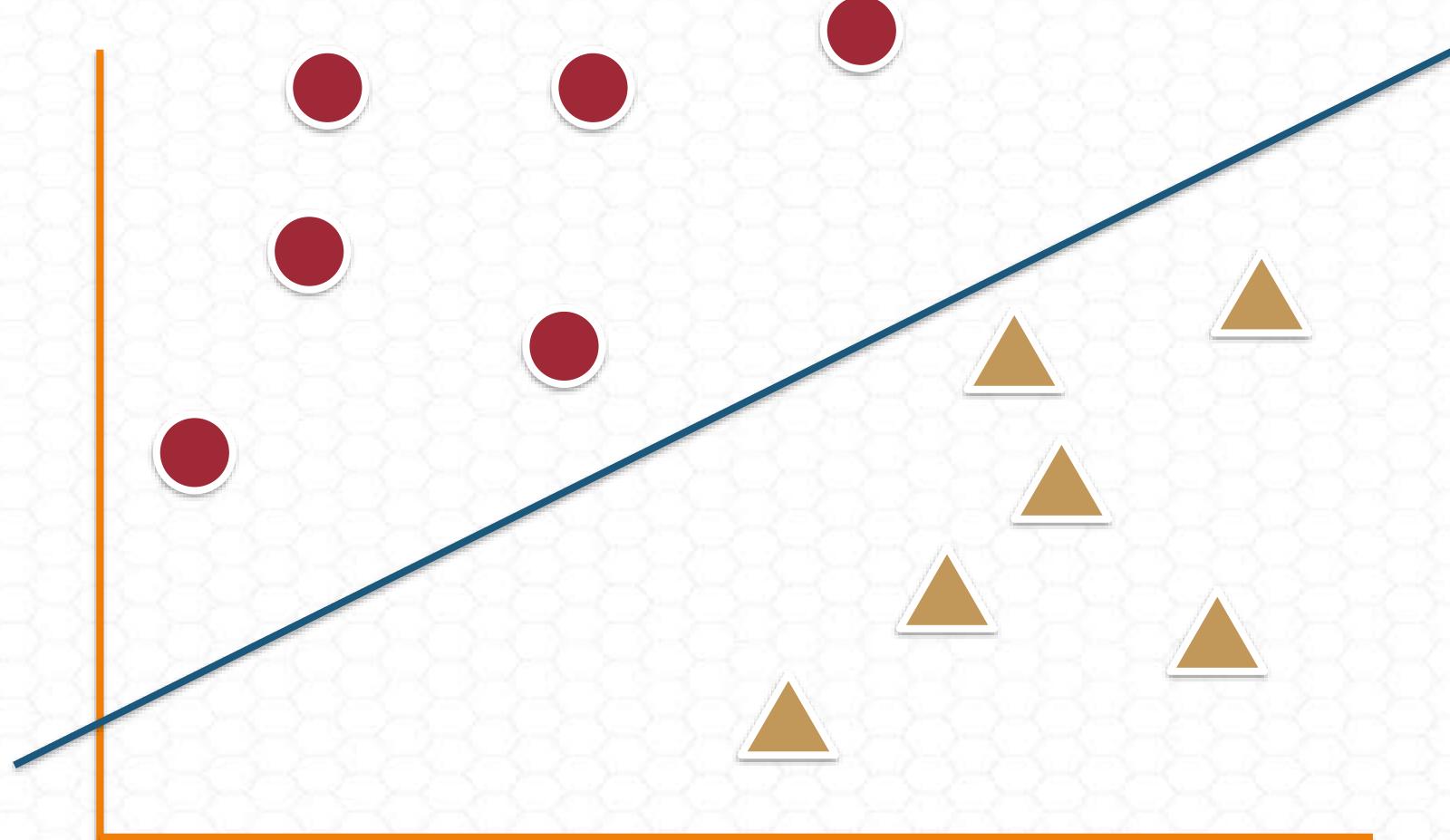
■ 參數學習(Parameter Learning)

□ 或稱為「參數化建模」(Parametric Modeling)

□ 以未確定的數值參數指定模型結構，依特定的訓練資料算出最佳參數值

□ 先根據專業知識挑選屬性，利用演算法調整參數，讓模型盡可能符合資料

參數化建模



邏輯回歸分析 (Logistic Regression)

■ 從對連續依變數的預測轉變為二元的結果(是/否)

- 客戶是否流失?
- 客戶是否買單?
- 腫瘤為良性還惡性?



如果是線性回歸
會不會X值越大會得到
>100% 的預測結果?

邏輯回歸分析 (Logistic Regression)

■ 定義


$$\text{logit}(y) = \ln(\text{odds}) = b_0 + b_1 X_1 + \epsilon$$

■ Odds

Odds

= Probability of event for success (PE)/ failure

= PE/(1-PE)

單純代表獲勝/失敗的機率

■ 推導

$$e^{\ln(\text{odds})} = \text{odds} = e^{(b_0 + b_1 X_1 + \epsilon_i)}$$

$$\text{PE} = \text{odds}/(1+\text{Odds}) = e^{(b_0 + b_1 X_1 + \epsilon_i)} * \frac{1}{1 + e^{(b_0 + b_1 X_1 + \epsilon_i)}}$$

建立邏輯回歸分析模型

```
from sklearn.datasets import load_iris  
from sklearn.linear_model import LogisticRegression  
iris = load_iris()  
clf = LogisticRegression()  
clf.fit(iris.data, iris.target)  
  
clf.predict(iris.data)
```

建立決策邊界 (1)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

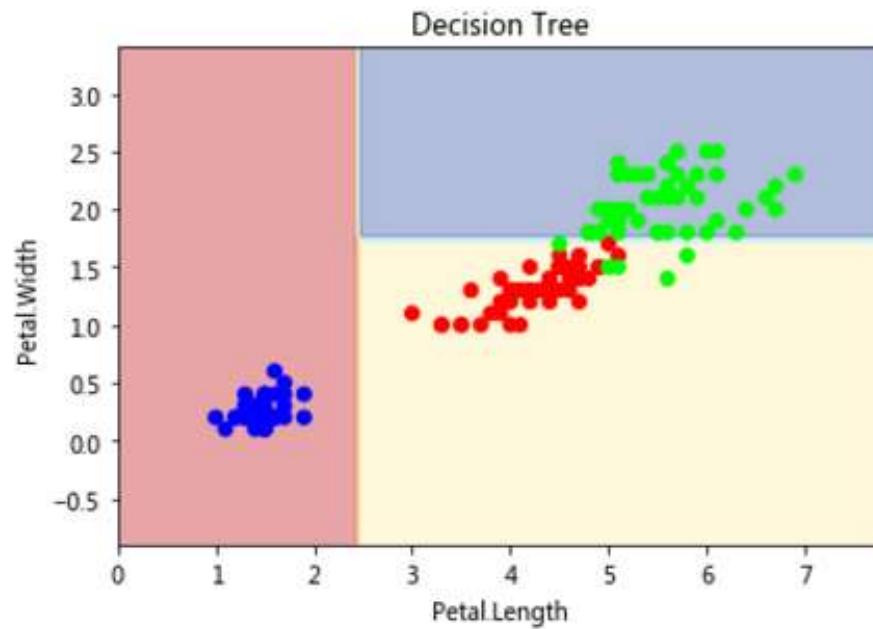
clf = LogisticRegression()
clf.fit(X, y)
```

建立決策邊界 (2)

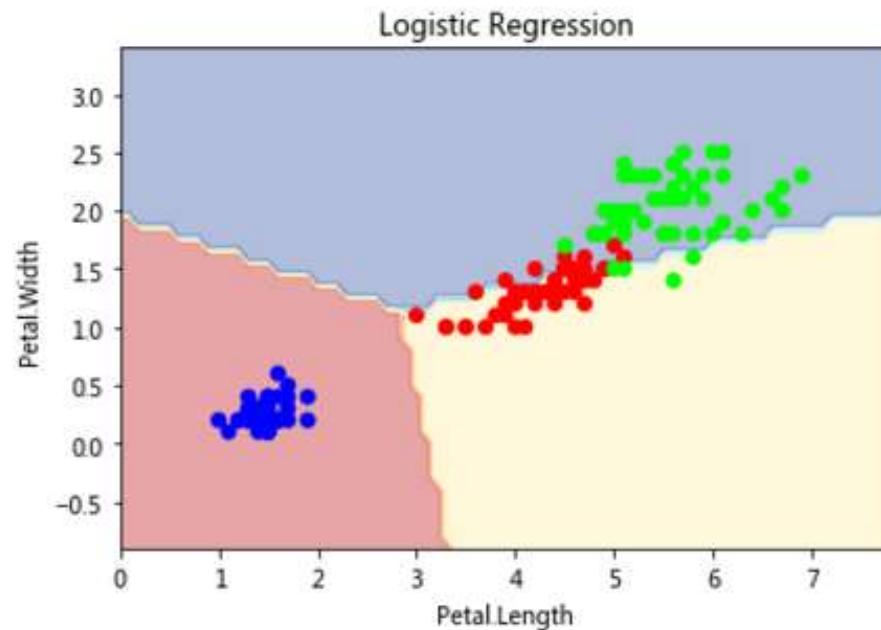
```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),  
                      np.arange(y_min, y_max, 0.1))  
  
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])  
Z = Z.reshape(xx.shape)  
  
plt.plot()  
plt.contourf(xx, yy, Z, alpha=0.4, cmap = plt.cm.RdYlBu)  
plt.scatter(X[:, 0], X[:, 1], c=y, cmap = plt.cm.brg)  
plt.title('Logistic Regression')  
plt.xlabel('Petal.Length')  
plt.ylabel('Petal.Width')  
plt.show()
```

模型比較

決策樹- 規則模型



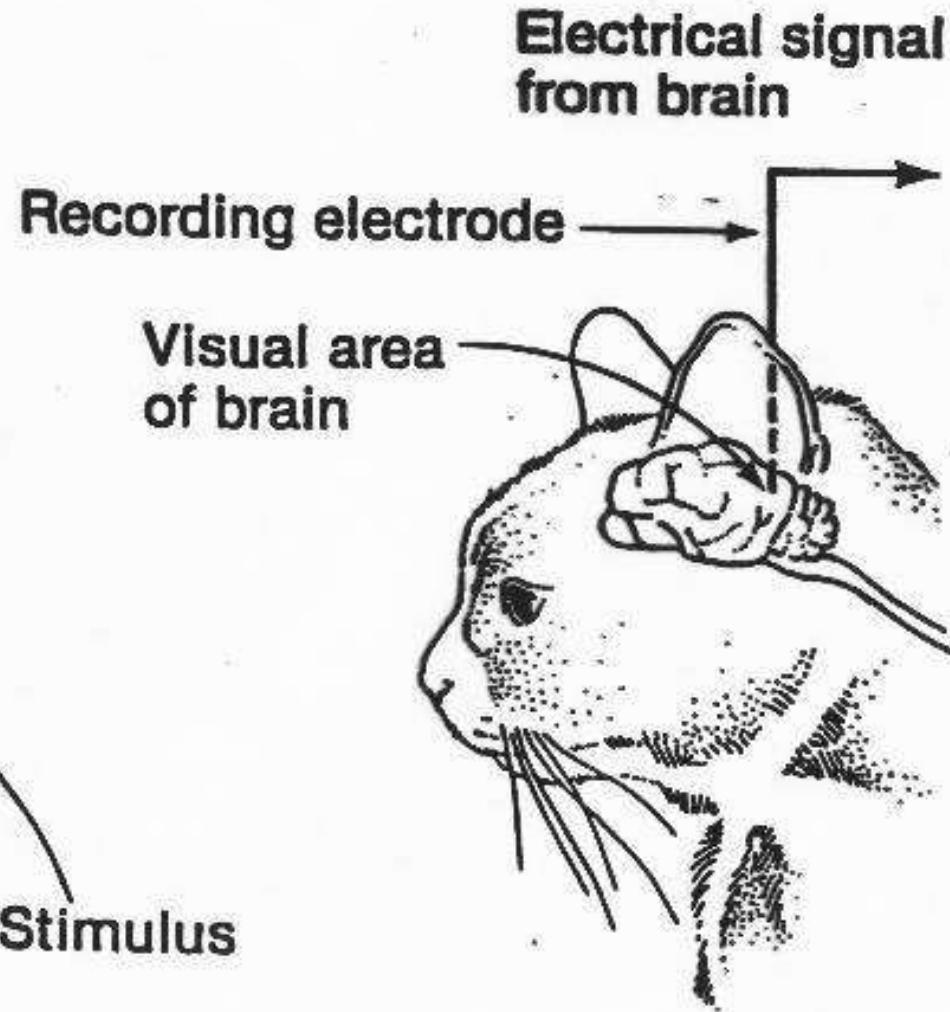
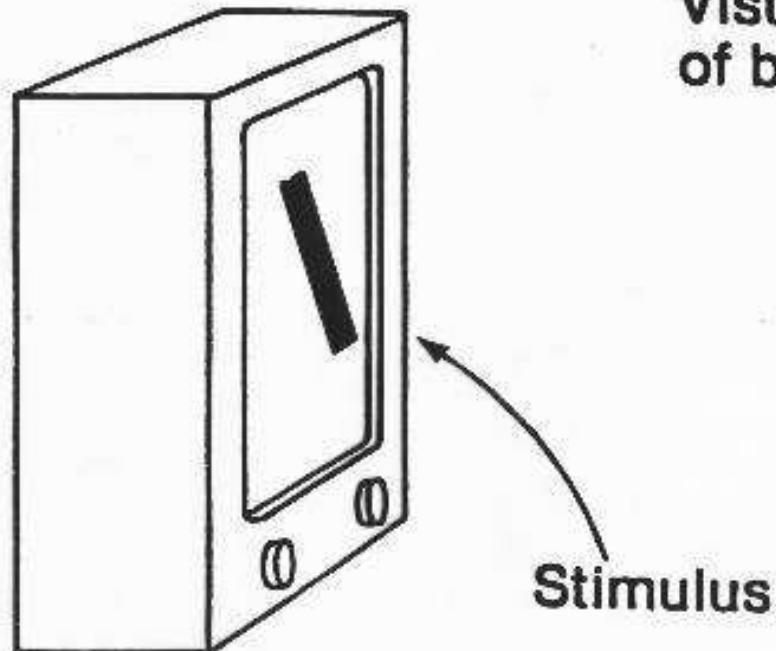
邏輯回歸模型-線性模型



類神經網路

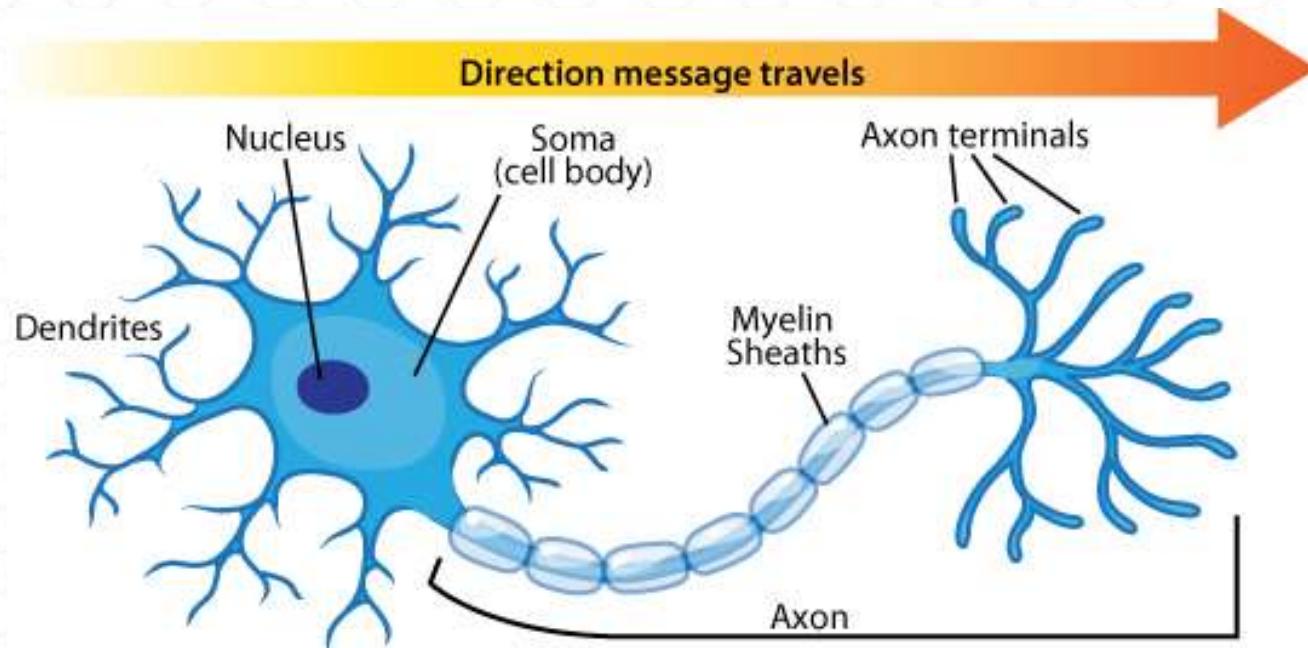
啟發於神經元的實驗

神經生物學家 David Hubel 和 Torsten Wiesel 在 1981 年發現貓的不同視覺神經元對於不同光影的反應不盡相同，在不同情況下的視覺神經元有著不同的活躍程度，甚至只對圖像的某些特定細節有反應，兩人因此發現而獲得了諾貝爾醫學獎。



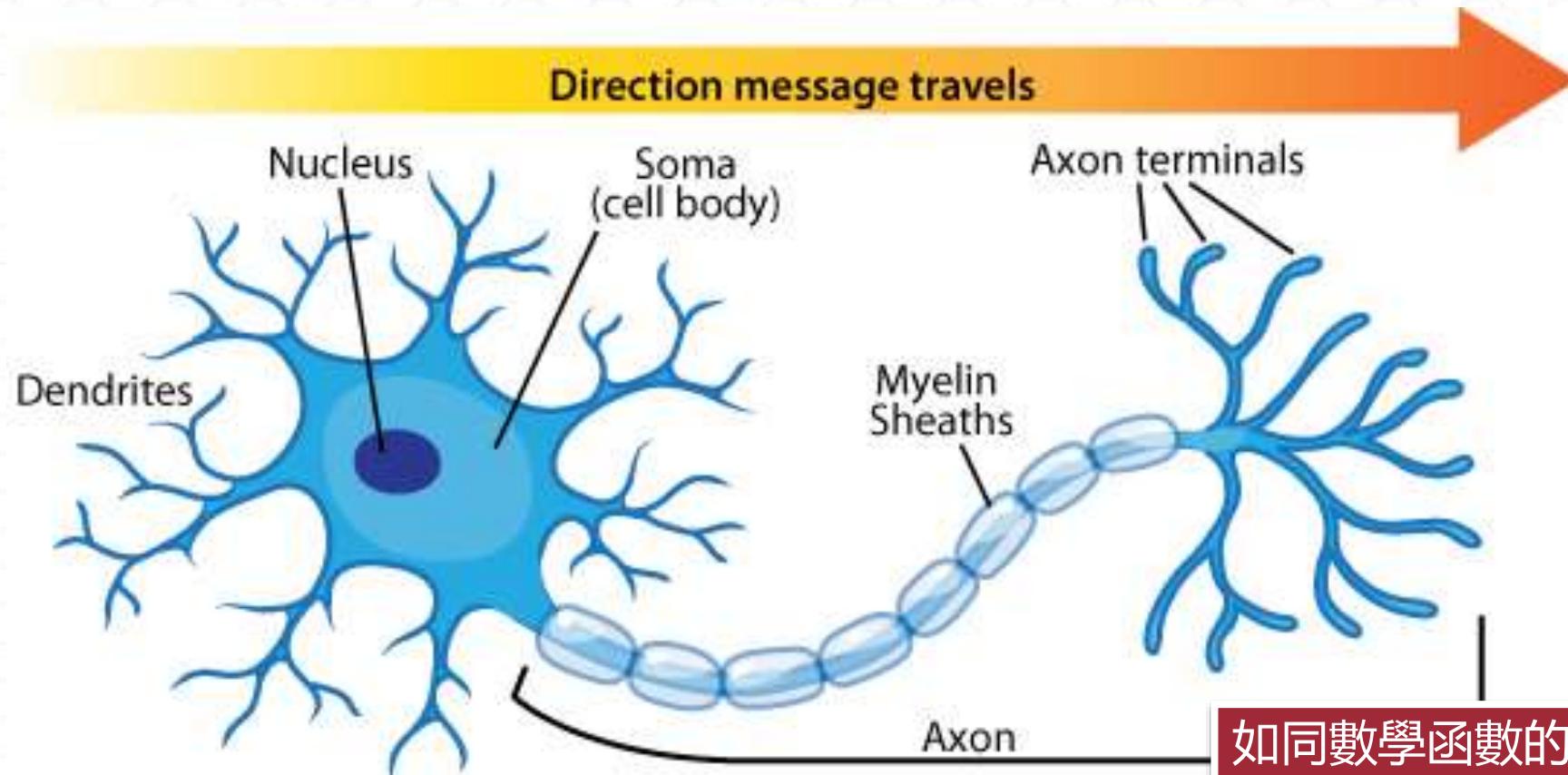
神經元

- 軸突 (Axon) : 連接在神經細胞核上，用來傳送由神經細胞核產生的信號至其它的神經細胞中。
- 樹突 (Dendrites) : 神經樹分為兩種：輸入神經樹及輸出神經樹。在圖1中左邊接到神經核的神經樹是用來接收其他神經細胞傳來的信號，稱為輸入神經樹。
- 突觸 (Synapse) : 輸入神經樹和輸出神經樹相連接的點稱為神經節



人工神經網路 (Artificial Neural Network)

- 神經系統由神經元構成，彼此間透過突觸以電流傳遞訊號。是否傳遞訊號、取決於神經細胞接收到的訊號量，當訊號量超過了某個閾值 (Threshold) 時，細胞體就會產生電流、通過突觸傳到其他神經元



類神經網路跟神經元有關但跟人腦無關



Yann LeCun

12月11日 9:41 ·

...

"Neural networks copy the human brain." I cringe every time I read something like this the press. It is wrong in multiple ways.

First, neural nets are loosely *inspired* by some aspects of the brain, just as airplanes are loosely inspired by birds.

Second the I aspiration doesn't come from the human brain. It comes from *any* animal brain: monkey, cat, rat, mouse, bird, fish, fruit fly, aplysia sea slug, all the way down to caenorhabditis elegans, the 1mm-long roundworm whose brain has exactly 302 neurons.

萬用函數

■ 如果能夠模擬神經元產生一個萬用函數呢？

$$f(\text{[sound波形圖]}) = \text{"How are you"}$$

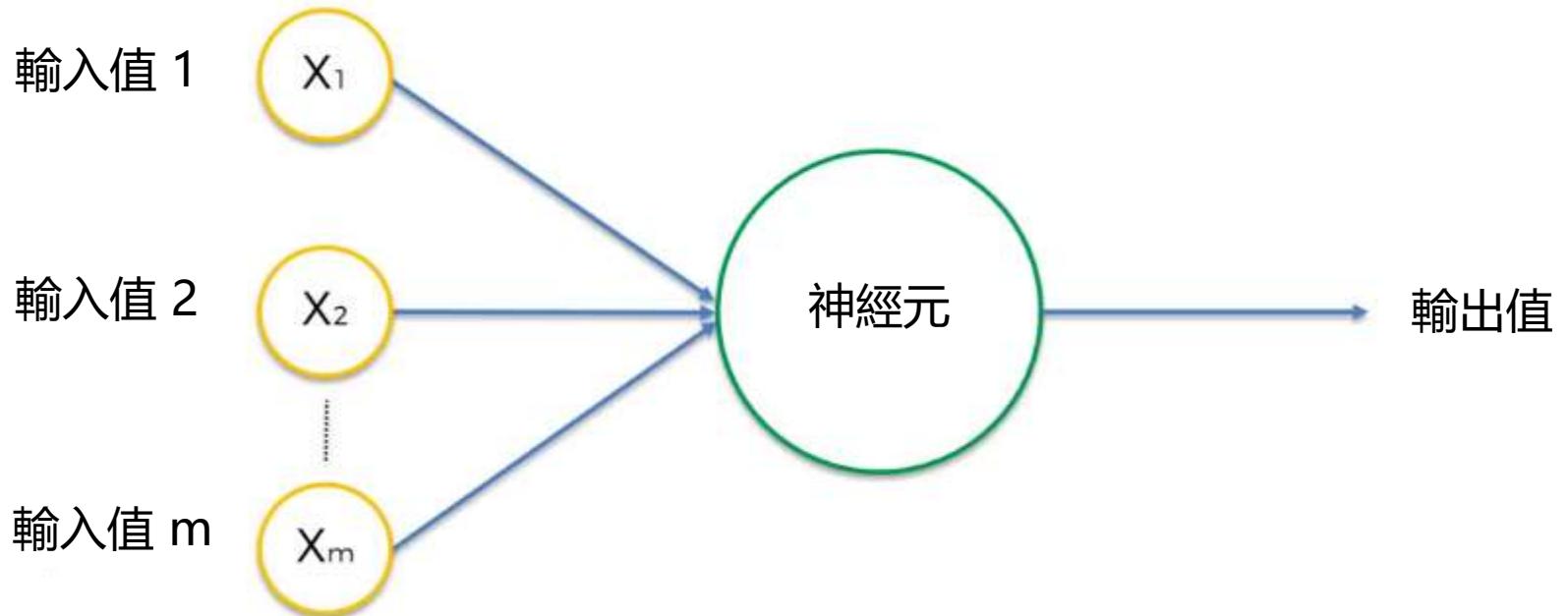
$$f(\text{[貓咪照片]}) = \text{"Cat"}$$

$$f(\text{[棋子照片]}) = \text{"5-5"}$$

是否就可以像人類一樣解決所有問題了？

感知機

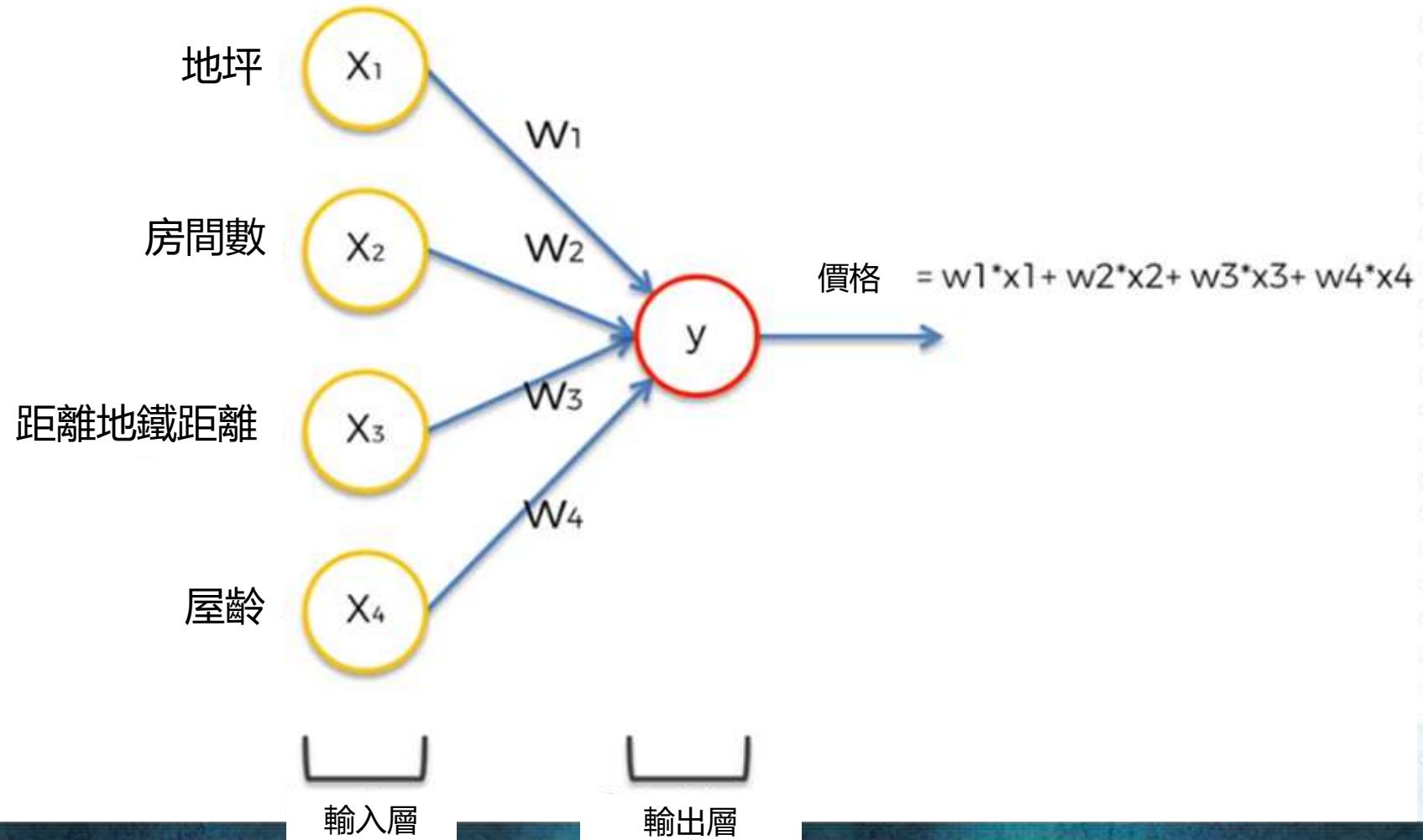
1957 年，Rosenblatt 提出了感知機(Perceptron)模型



1. 加總收集到的訊號
2. 線性或非線性轉換
3. 產生一個新的信號

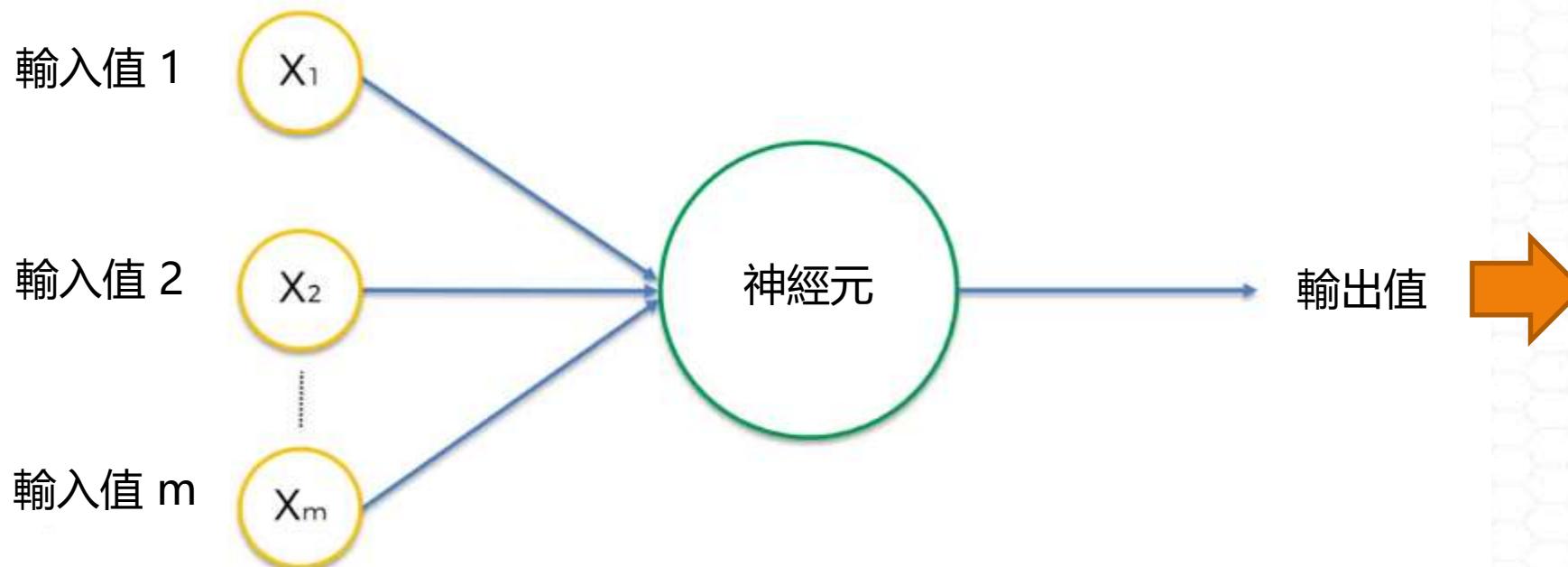
單層感知機

■ 假設要預測房屋價格



邏輯門

■ 邏輯門是組成數位系統的基本結構，通常組合使用實現更為複雜的邏輯運算



AND

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

XOR

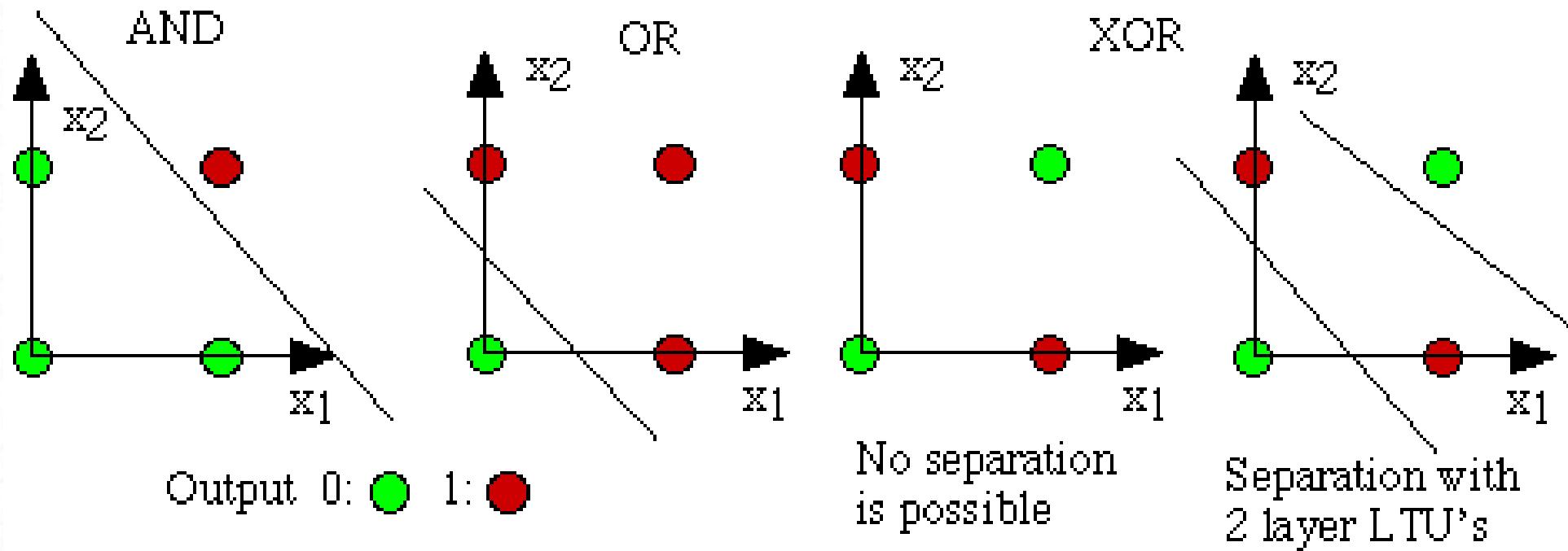
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

NOT

in	out
0	1
1	0

單層感知機的缺點

- 1969年，MIT人工智慧實驗室的Marvin Minsky 和 Seymour 出版了 Perceptrons 一書，證明單層感知機無法處理XOR 問題



利用AND、OR、NOT組合XOR

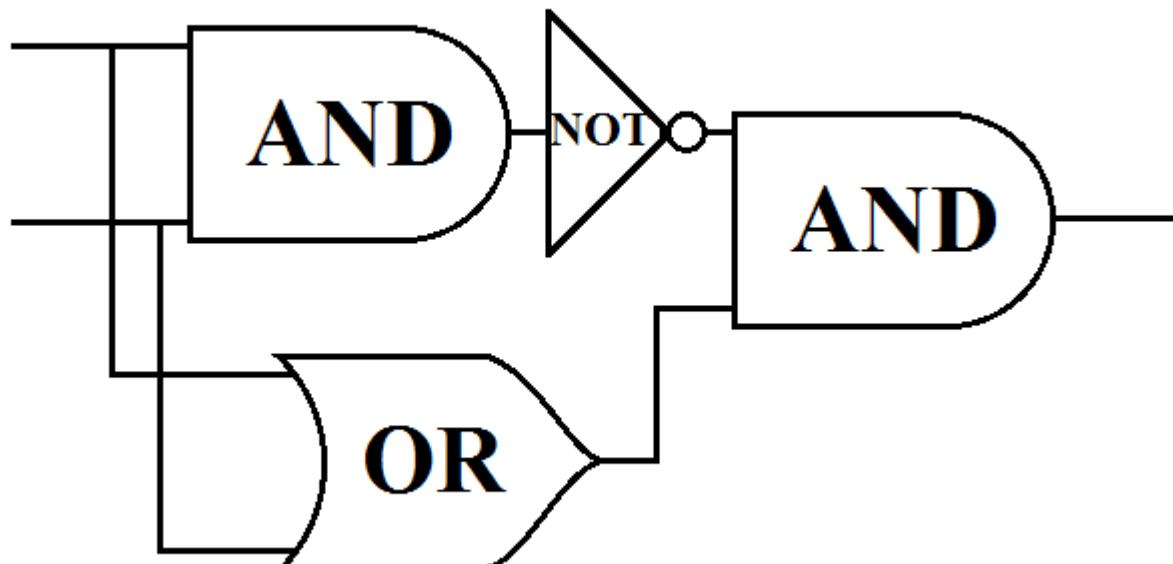


a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

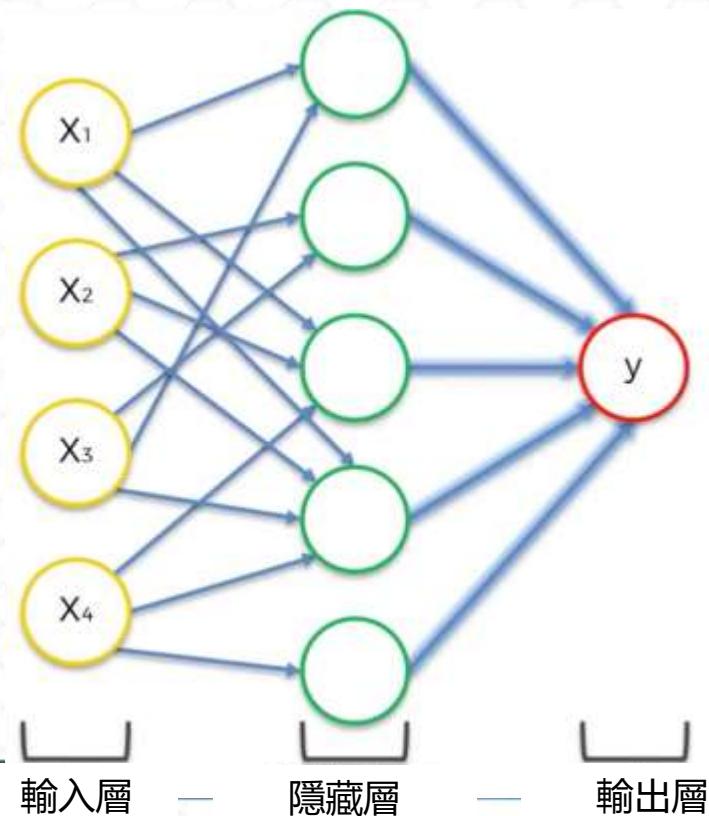
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

in	out
0	1
1	0



多層感知機

- 1986 年，Rumelhart、Hinton 等人提出「反向傳播演算法」
(Backpropagation) 訓練神經網路，催生出具備非線性學習能力的**多層感知機**
(Multi-Layer Perceptron)



Geoffrey Hinton - 深度學習之父

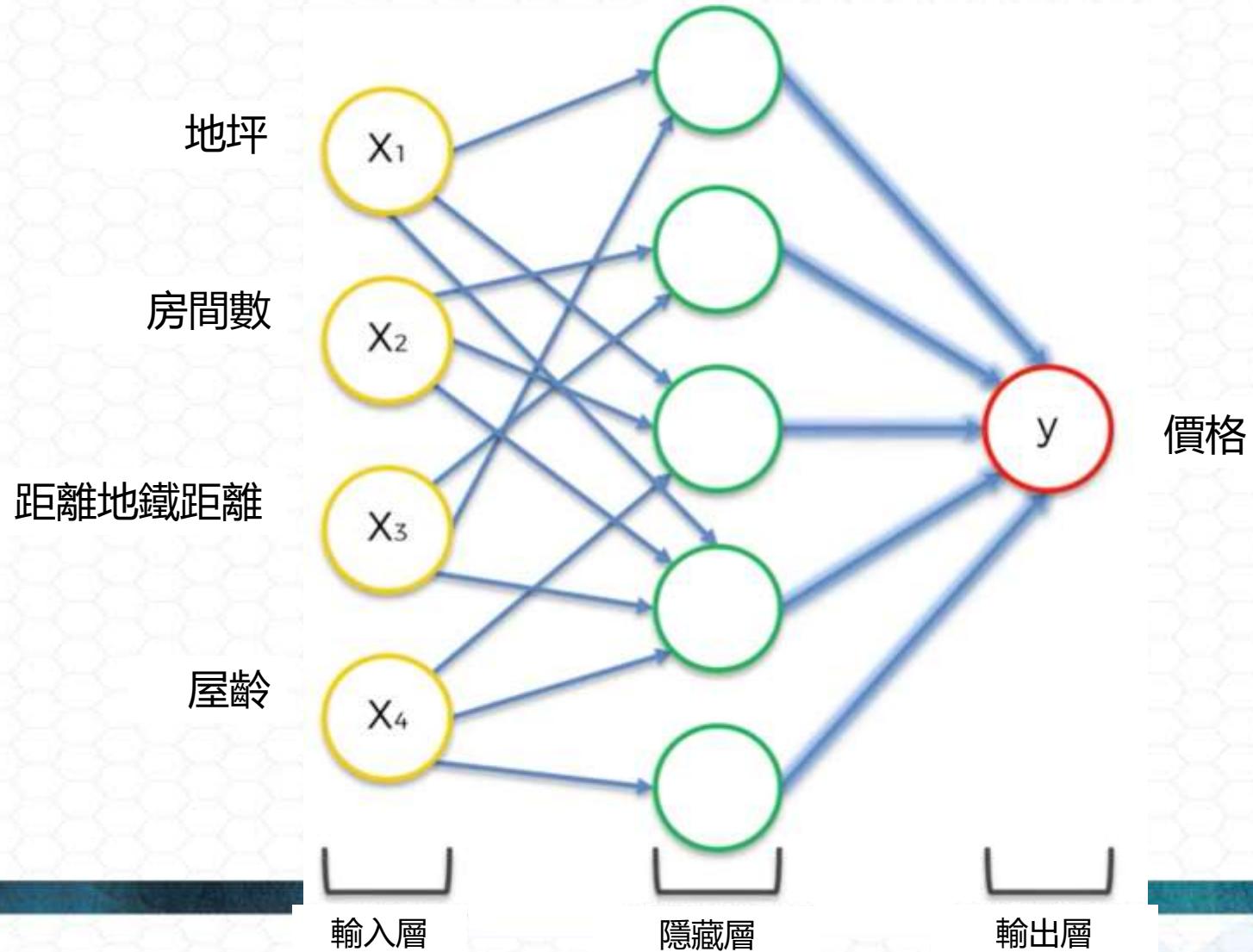


<http://www.cs.toronto.edu/~hinton/>

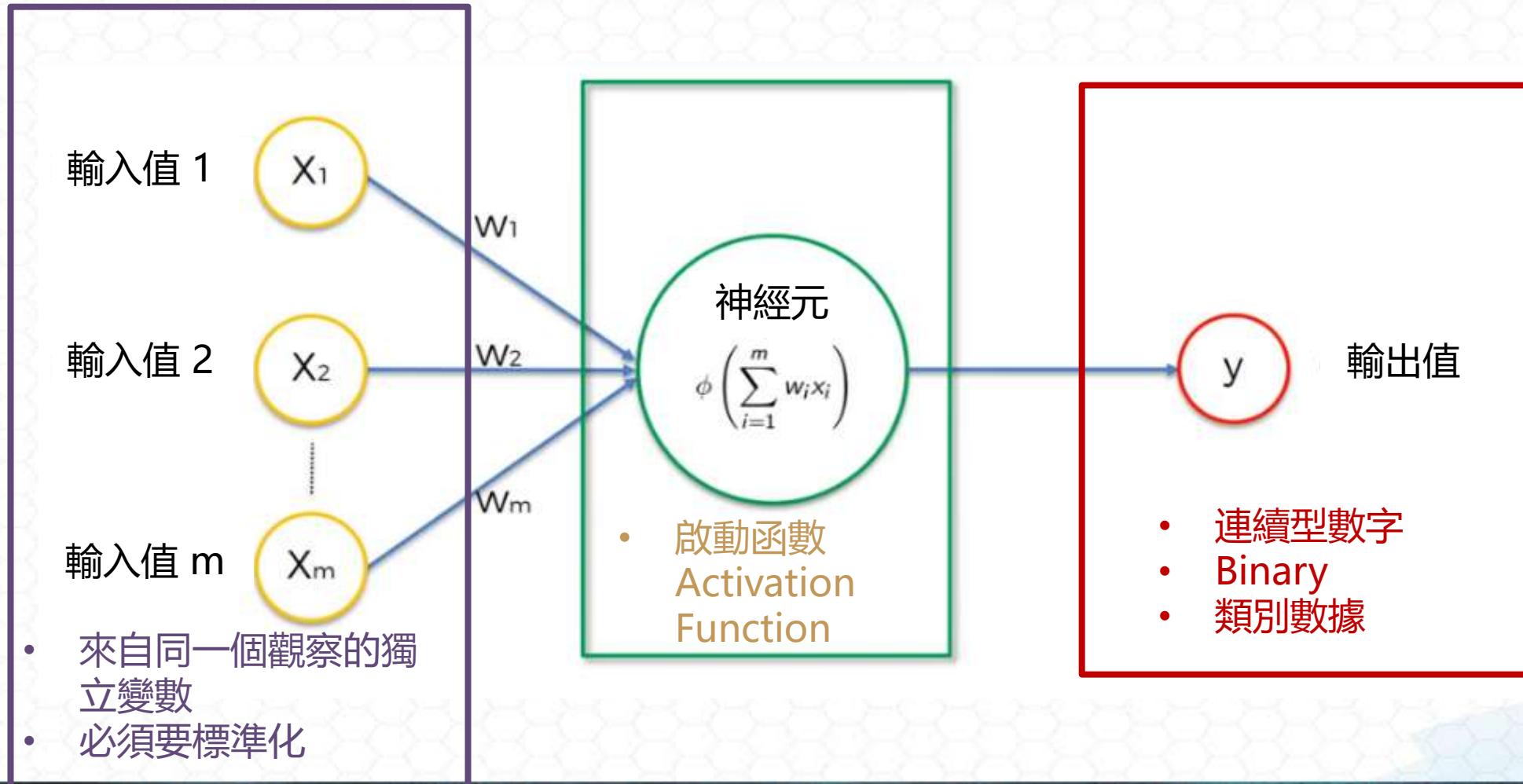


建構神經網路

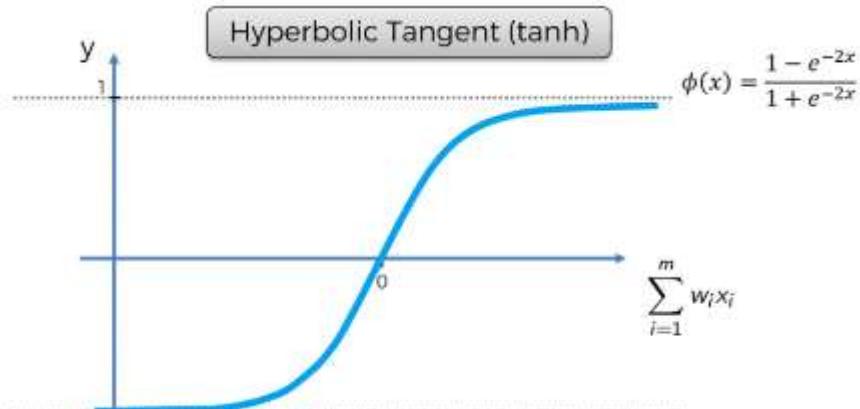
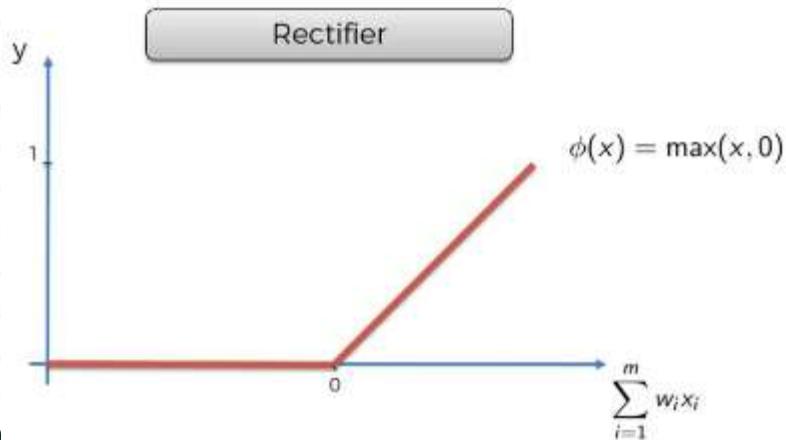
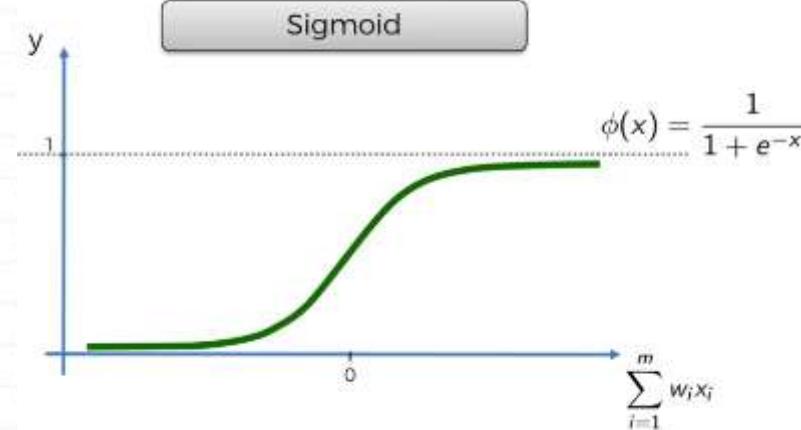
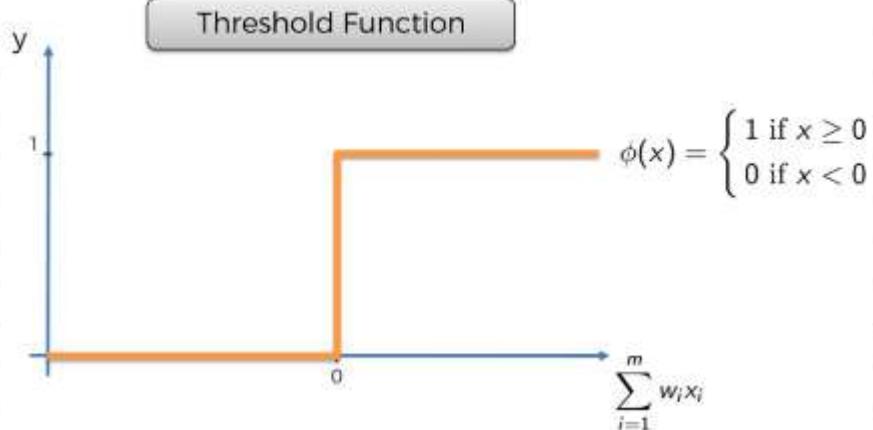
神經網路



使用激活函數調整輸出



激活函數(Activation Function)



Threshold Function

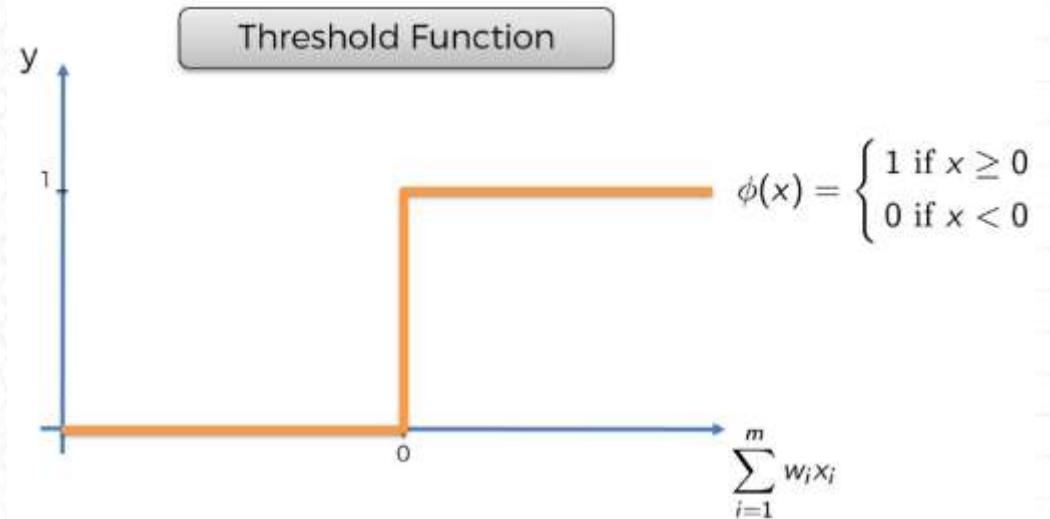
```
def threshold_function(x):
```

```
    y = x > 0
```

```
    return y.astype(int)
```

```
x = np.array([-1,1,2])
```

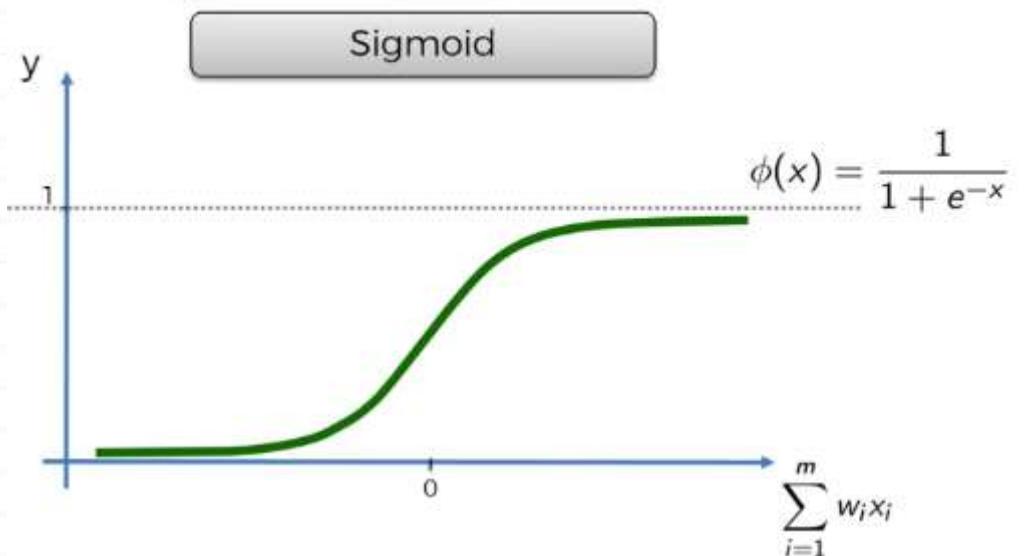
```
threshold_function(x)
```



Sigmoid Function

```
def sigmoid_function(x):  
    return 1/ (1 + np.exp(-x))
```

```
x = np.array([-1,1,2])  
sigmoid_function(x)
```



優點:

- 求導數容易

缺點:

- 容易發生梯度消失問題
- 中心點不為0

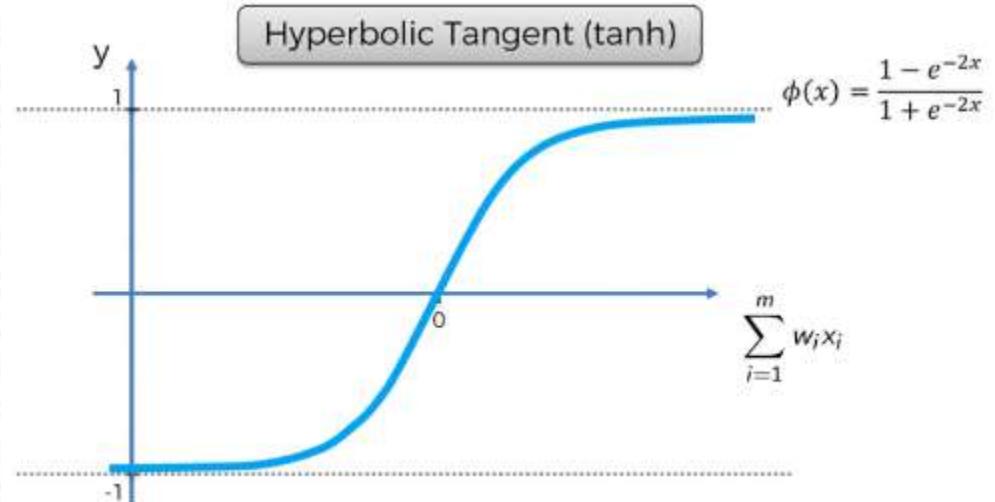
Tangent Function

```
def tangent_function(x):  
    return (1 - np.exp(-2*x)) / (1 +  
    np.exp(-2*x))
```

```
x = np.array([-1,1,2])  
tangent_function(x)
```

OR

```
np.tanh(x)
```



優點:

- 比Sigmoid函數收斂速度更快
- 輸出以0為中心

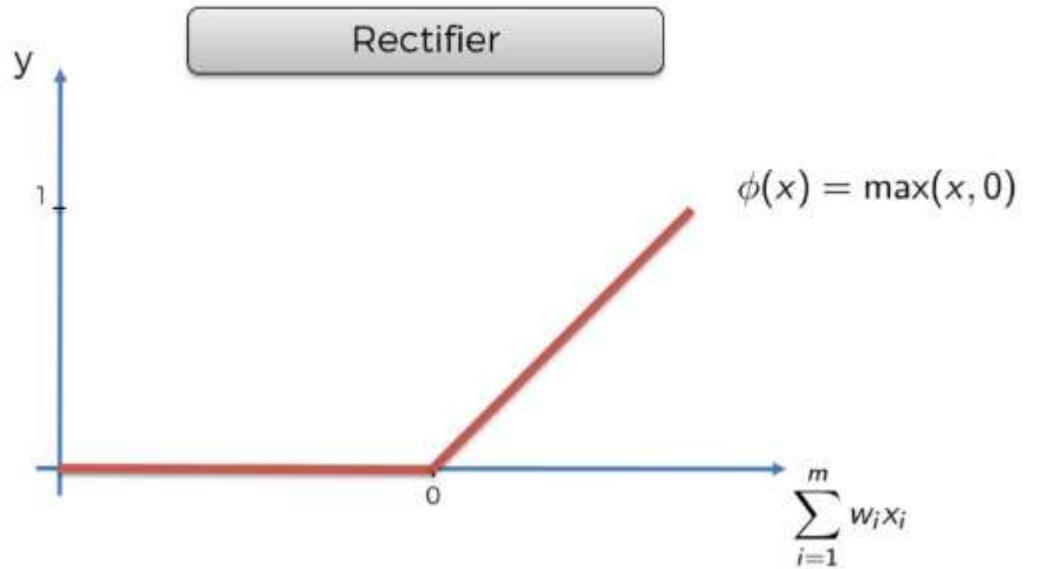
缺點:

- 容易發生梯度消失問題

ReLU Function

```
def relu_function(x):  
    return np.maximum(0,x)
```

```
x = np.array([-1,1,2])  
relu_function(x)
```



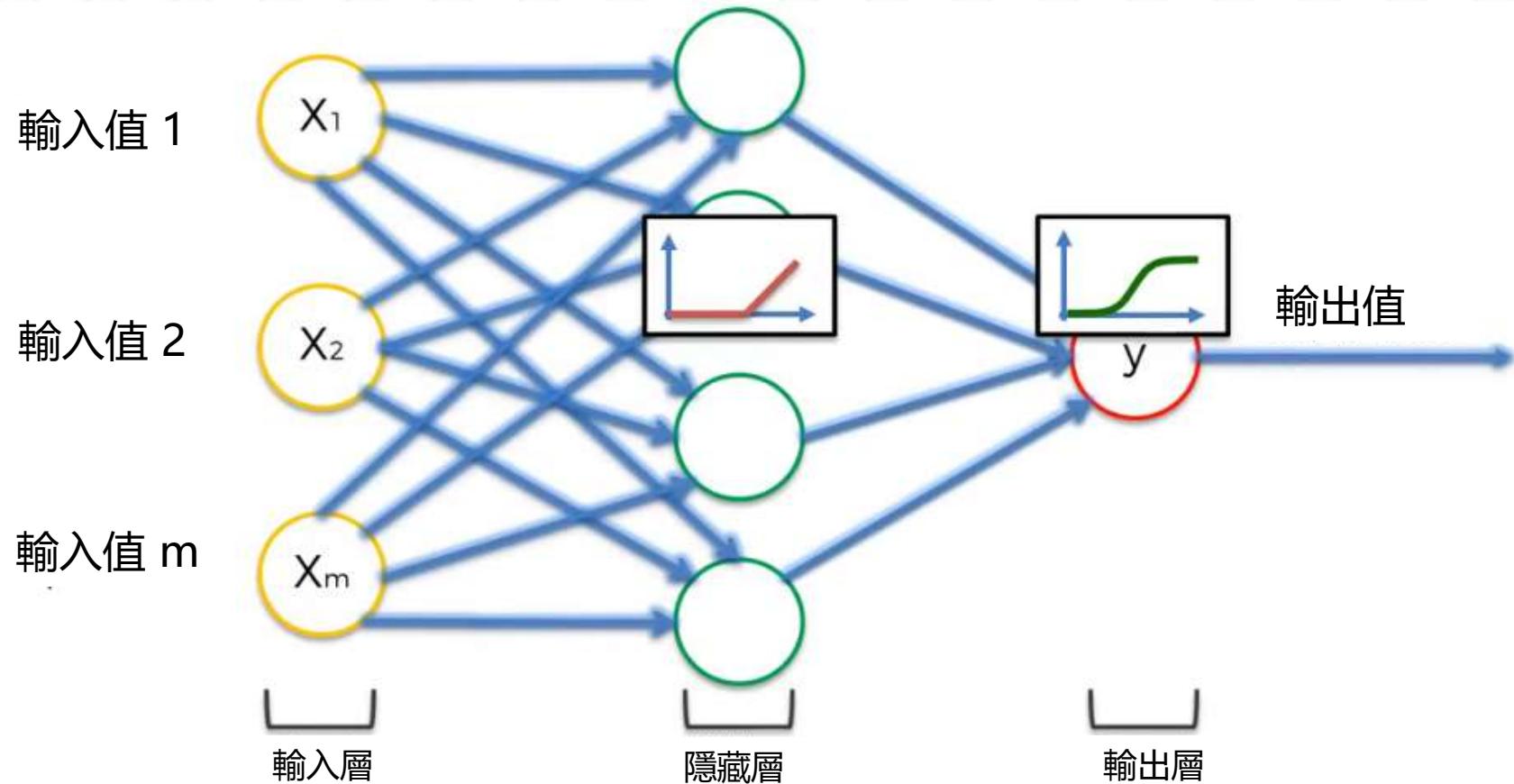
優點:

- 快速收斂
- 解決梯度消失問題

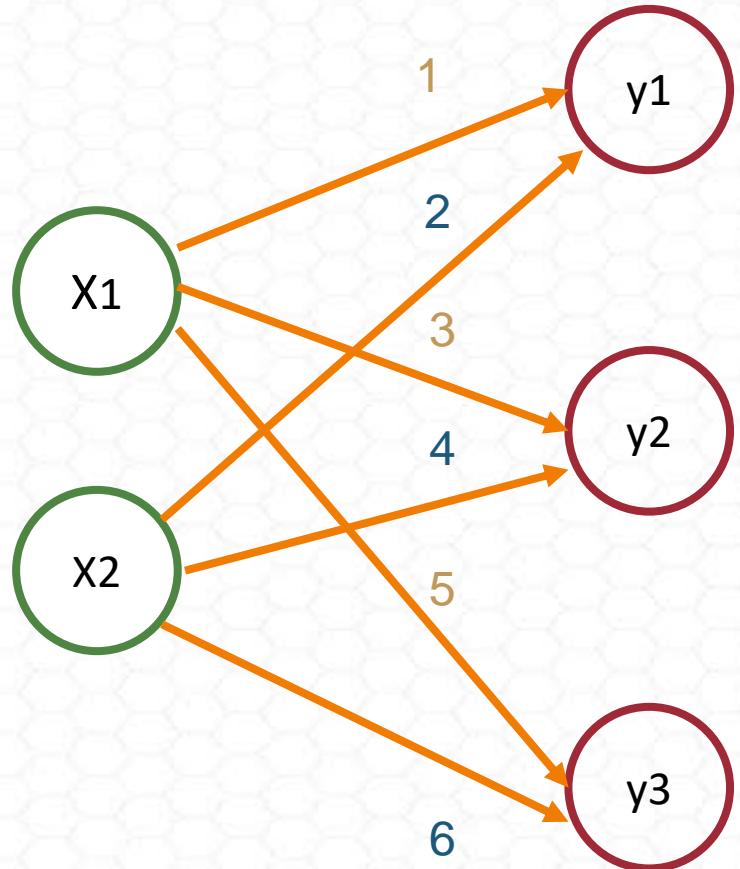
缺點:

- 神經元死亡問題

建構神經網路



單層神經網路前向傳播過程 (Forward Propagation)

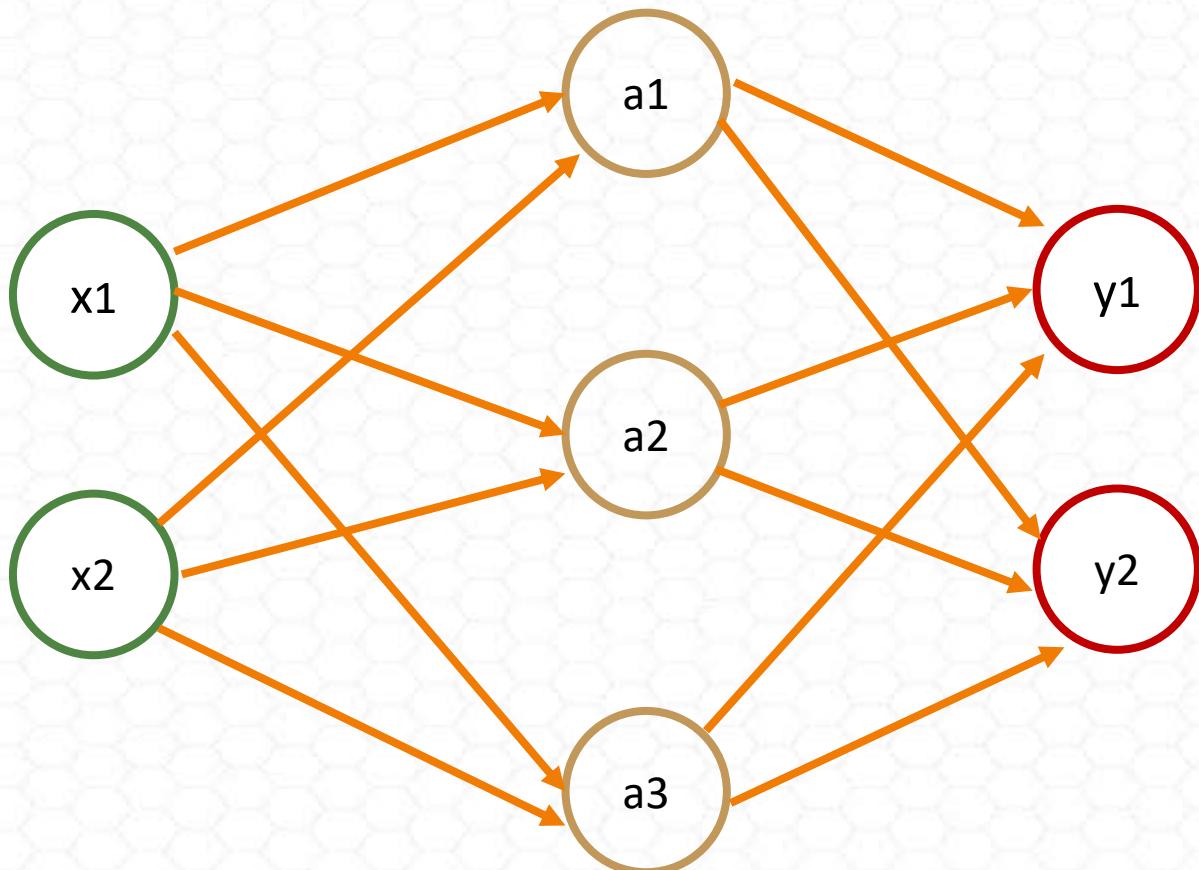


$$\begin{matrix} X \\ 2 \end{matrix} \quad \begin{matrix} W \\ 2 \times 3 \end{matrix} = \begin{matrix} Y \\ 3 \end{matrix}$$

單層神經網路前向傳播過程 (Forward Propagation)

```
import numpy as np  
X = np.array([1,2])  
W = np.array([[1,3,5],[2,4,6]])  
Y = np.dot(X,W)  
Y
```

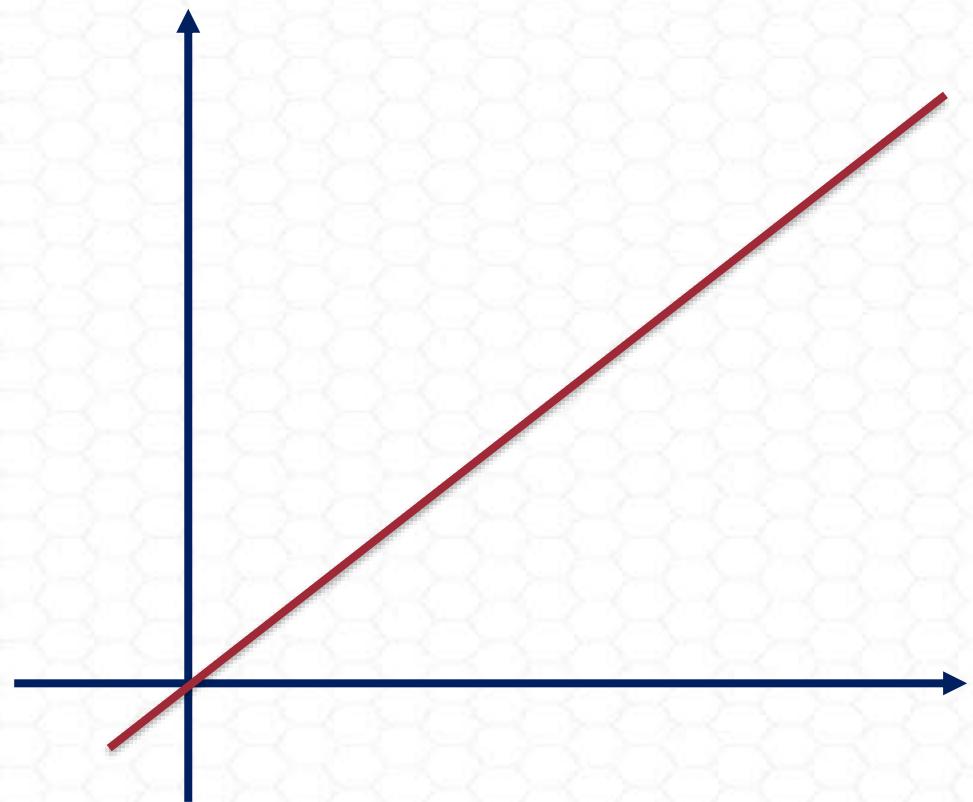
兩層神經網路前向傳播過程 (Forward Propagation)



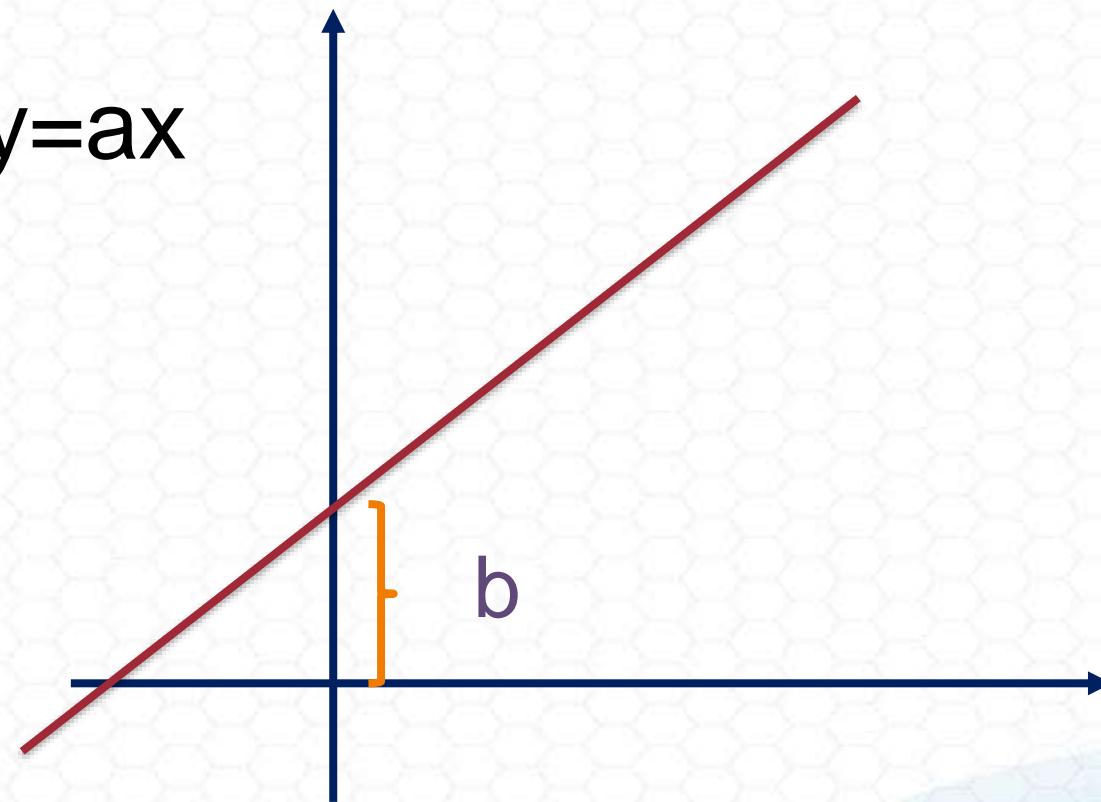
增加偏倚 (bias)

■ 偏倚 (bias) 可以充當閥值調整激活難度

□ e.g. 調整分類結果的參數

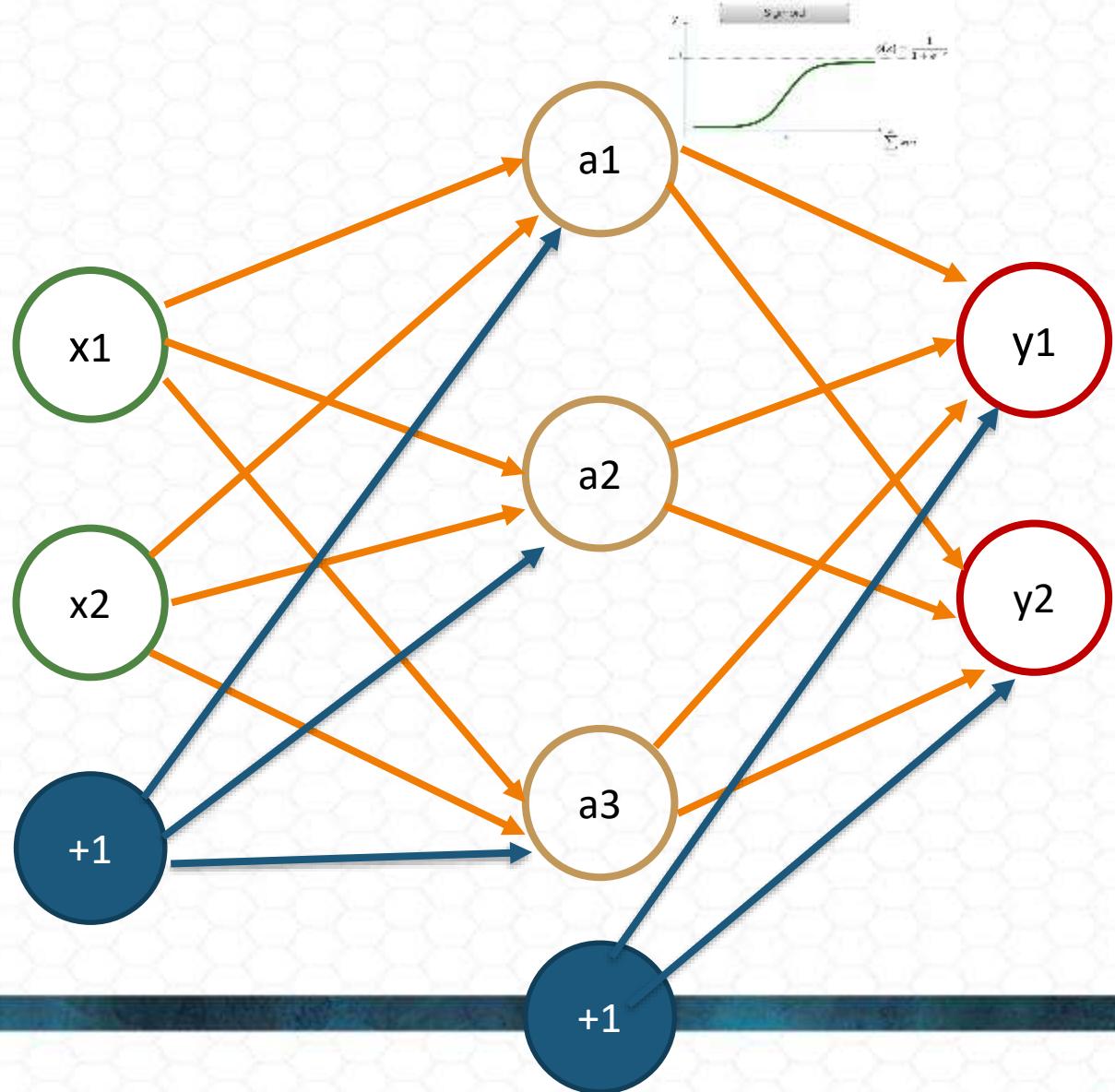


$$y=ax$$



$$y=ax+b$$

增加偏倚 (bias)



初始神經網路

```
network = {}  
network['w1'] = np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])  
network['b1'] = np.array([0.1,0.2,0.3])  
network['w2'] = np.array([[0.1,0.4],[0.2,0.5],[0.3,0.6]])  
network['b2'] = np.array([0.1,0.2])
```

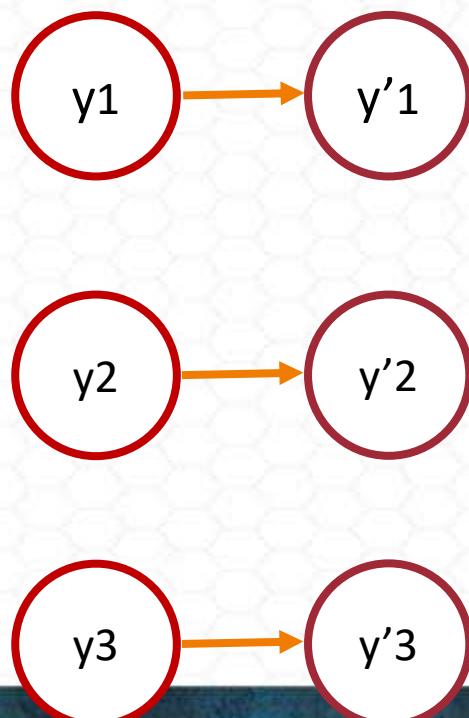
計算傳遞過程

```
x = np.array([1,0.5])
a = np.dot(x, network['w1']) + network['b1']
z = sigmoid_function(a)
y = np.dot(z1, network['w2']) + network['b2']
y
```

Softmax

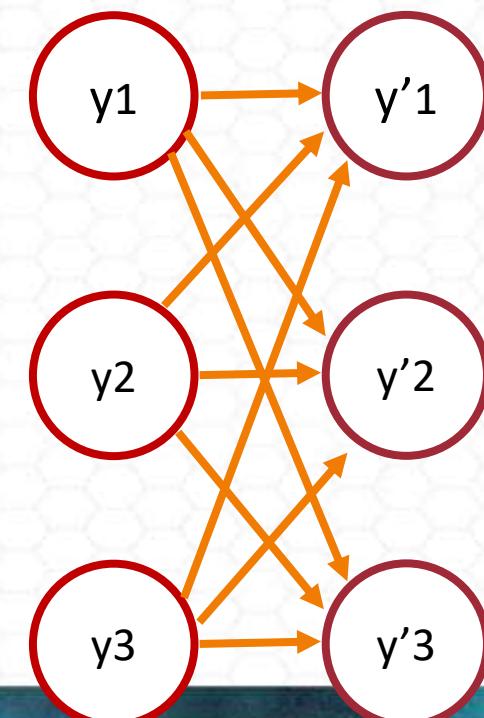
Identity Function

$$y'_k = y_k$$



Softmax Function

$$y'_k = \frac{e^{y_k}}{\sum_{i=1}^n e^{y_i}}$$

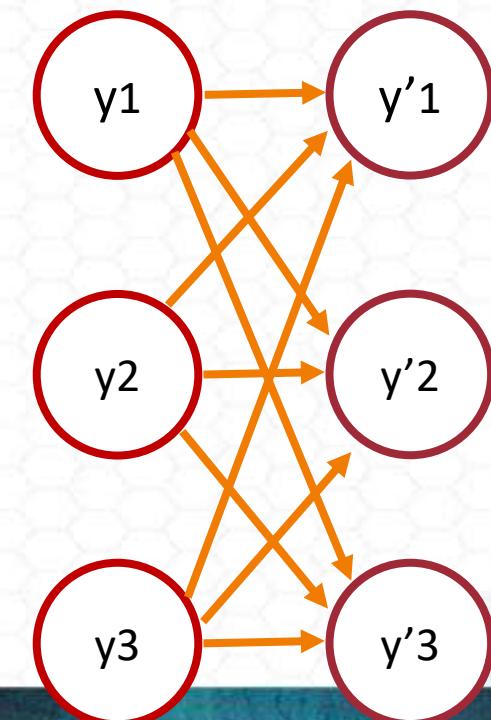


Softmax

```
def softmax_function(x):  
    return np.exp(x) / np.sum(np.exp(x))
```

```
softmax_function(y)
```

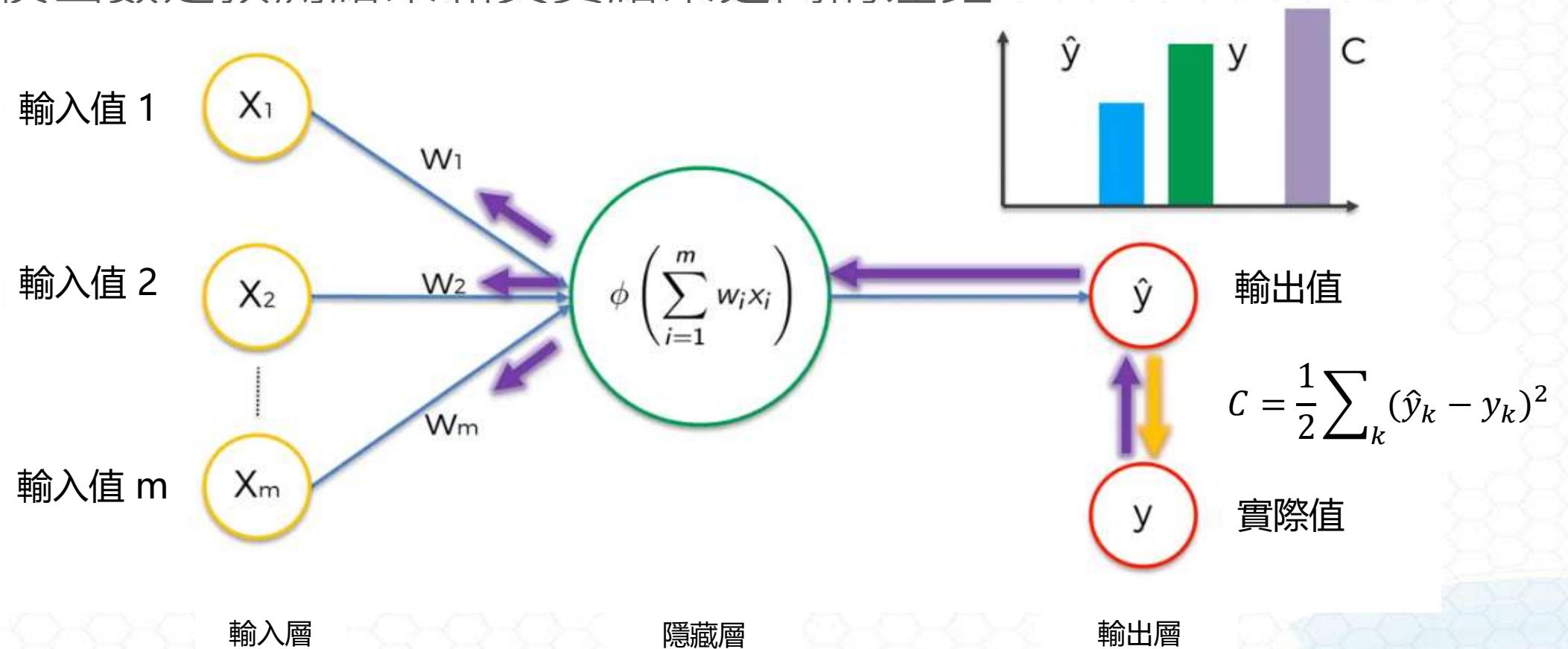
$$y'_k = \frac{e^{y_k}}{\sum_{i=1}^n e^{y_i}}$$



神經網路學習過程

類神經網路如何運作

- 「代價函數」(Cost Function) 或「損失函數」(Loss Function)。
代價函數是預測結果和真實結果之間的差距



均方誤差 Mean squared error

■ 公式

$$C = \frac{1}{2} \sum_k (\hat{y}_k - y_k)^2$$

■ 實作

```
def mean_squared_err(y_hat, y):  
    return 0.5 * np.sum((y_hat - y) ** 2)
```

交叉熵 Cross Entropy

■ 公式

$$C = - \sum_k y_k \log \hat{y}_k$$

■ 實作

```
def cross_entropy_err(y_hat, y):
    delta = 1e-8
    return -np.sum(y*np.log(y_hat + delta))
```

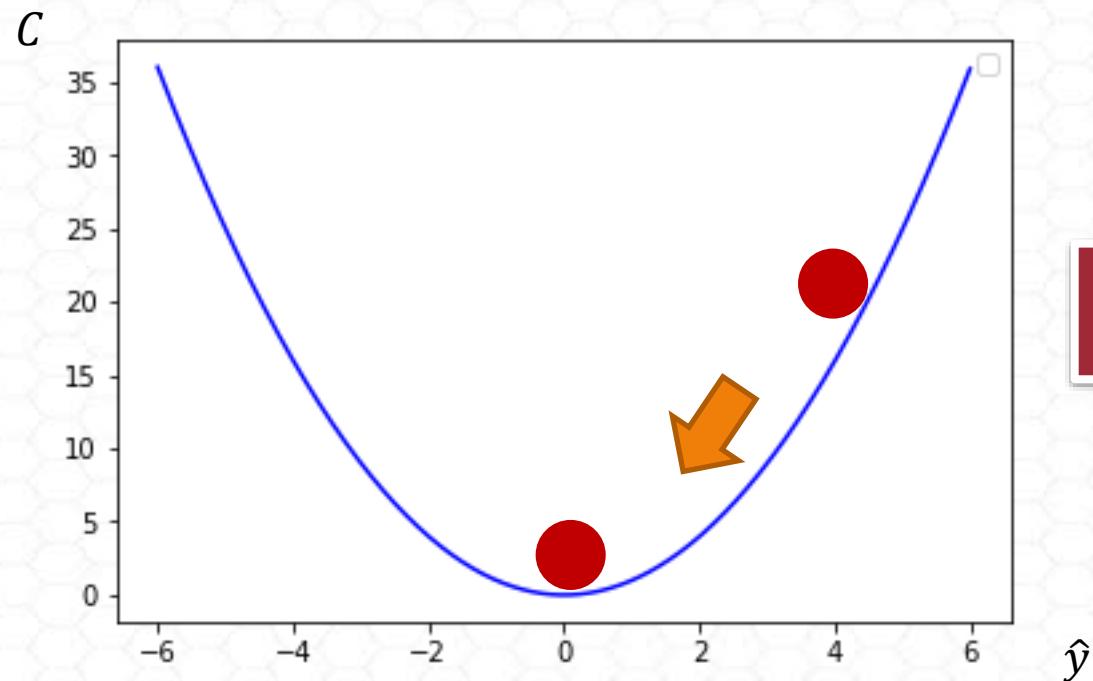
梯度法

■ Gradient Descent Method

- 找出代價函數的最小值

■ Gradient Ascent Method

- 找出代價函數的最大值

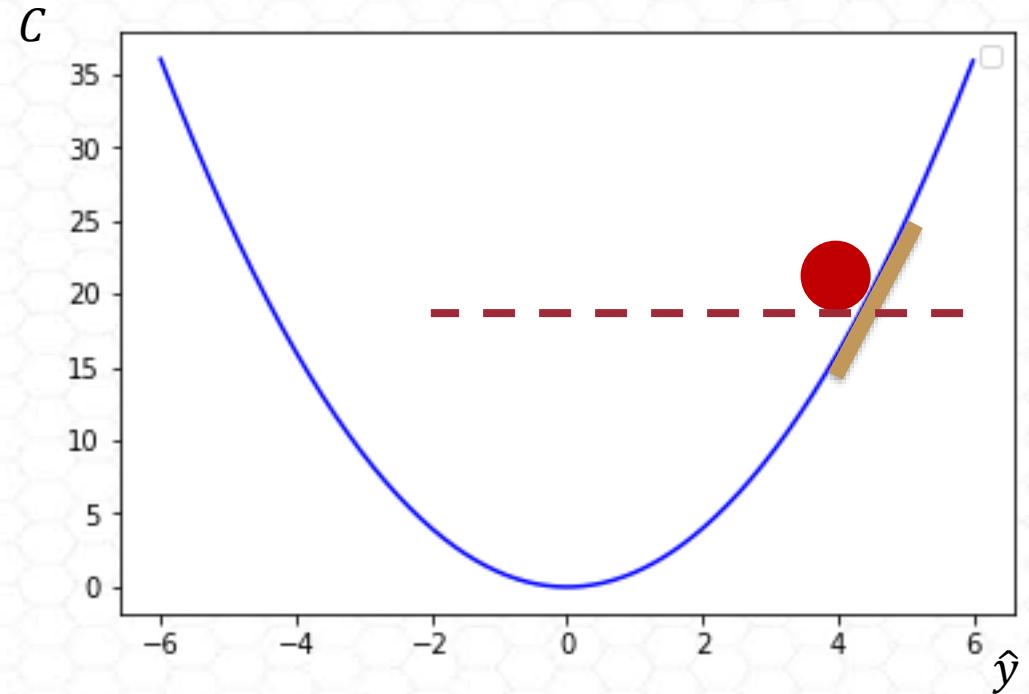


但參數這麼多
要如何有效搜尋代價函數的最小值

計算偏微分

- 要快速到達目的地，就必須知道要往哪個方向做調整？以及調整幅度的範圍？

$$\frac{df(x)}{d(x)} = \lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h}$$



斜率為正, 往左調整
斜率為負, 往右調整

斜率越大, 調整幅度越大

求偏微分

```
def func(x):  
    return x ** 2
```

$$f(x) = x^2$$

```
def dfunc(f, x):  
    h = 1e-4  
    return (f(x+h) - f(x)) / (h)
```

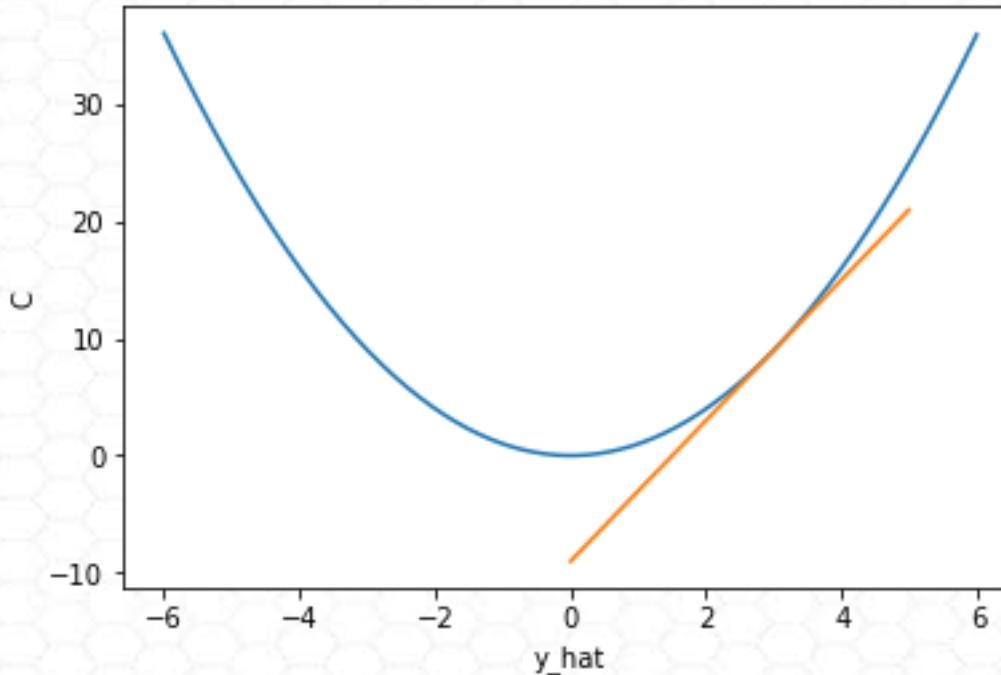
$$\frac{df(x)}{d(x)} = 2x$$

利用微分求出切線

```
# 切線函數
def tfunc(f, x, t):
    d = dfunc(f, x)
    y = f(x) - d*x
    return d*t + y

# 繪製  $x^2$ 
x = np.arange(-6, 6, 0.01)
y = func(x)
plt.plot(x, y)

# 繪製  $x = 3$  時的切線
x2 = np.arange(0, 5, 0.01)
y2 = tfunc(func, 3, x2)
plt.plot(x2, y2)
```



中央差分

$$\blacksquare \frac{df(x)}{d(x)} = \lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h}$$

$$\bullet \frac{df(x)}{d(x)} = \lim_{h \rightarrow 0} \frac{f(x+h)-f(x-h)}{2h}$$

```
def dfunc(f, x):  
    h = 1e-4  
    return (f(x+h) - f(x)) / (h)
```



```
def dfunc(f, x):  
    h = 1e-4  
    return (f(x+h) - f(x-h)) / (2*h)
```

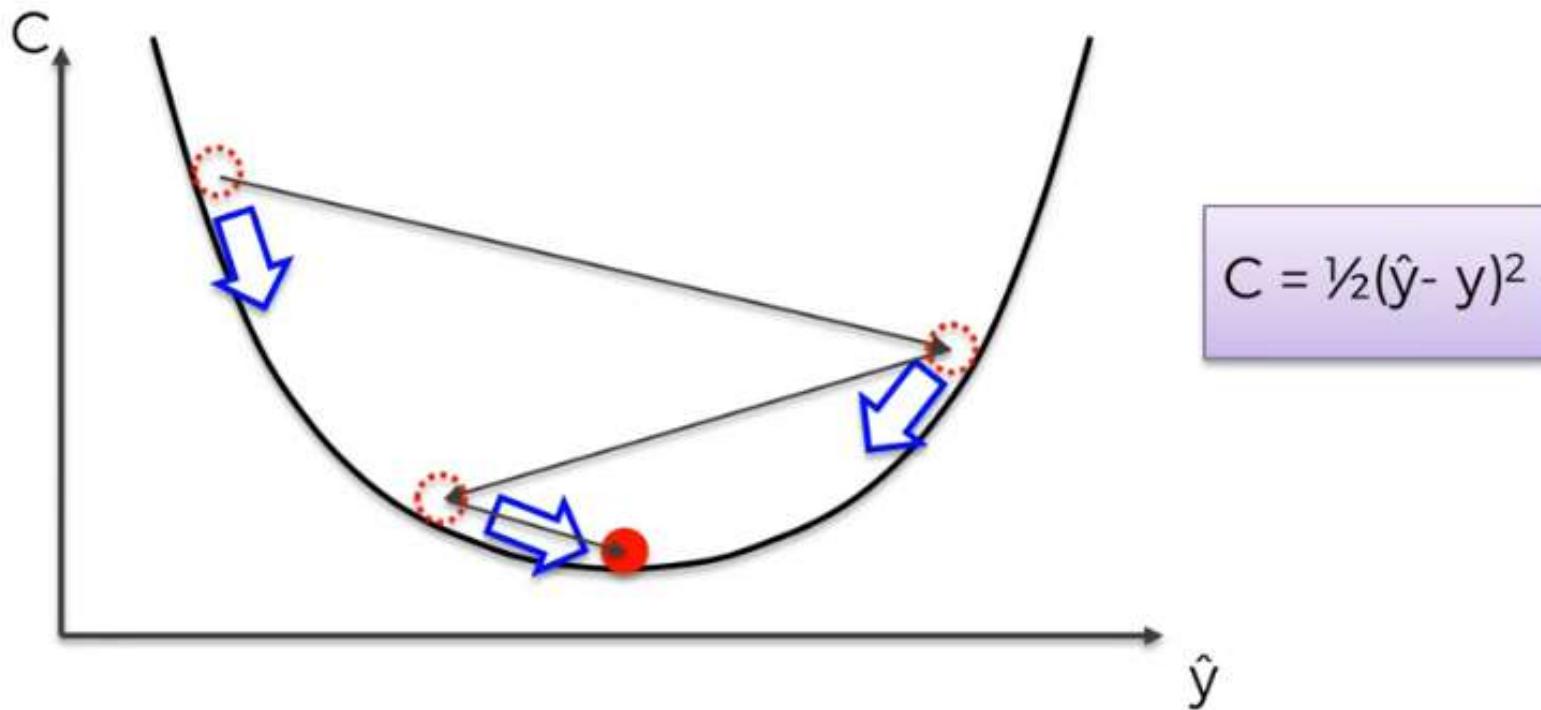
因為 h 不可能逼近無限小

計算偏微分通用公式

```
def dfunc(f, x):
    h = 1e-4
    grad = np.zeros_like(x)
    it = np.nditer(x, flags=['multi_index'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)
        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val
        it.iternext()
    return grad
```

學習率(Learning Rate)

- 有時根據斜率調整會調過頭，所以我們會在調整時乘上一個很小的數值 - 學習率



學習率(Learning Rate)

- ε 代表學習率, 決定在每次的學習過程中, 要變更多參數, 讓函數往最小的地方前進.

$$x_i = x_i - \varepsilon \frac{\delta f}{\delta x}$$

- 學習率的值太大或太小, 都無法達到最小值, 因此必須在學習的過程中, 必須適度調整學習率

梯度下降

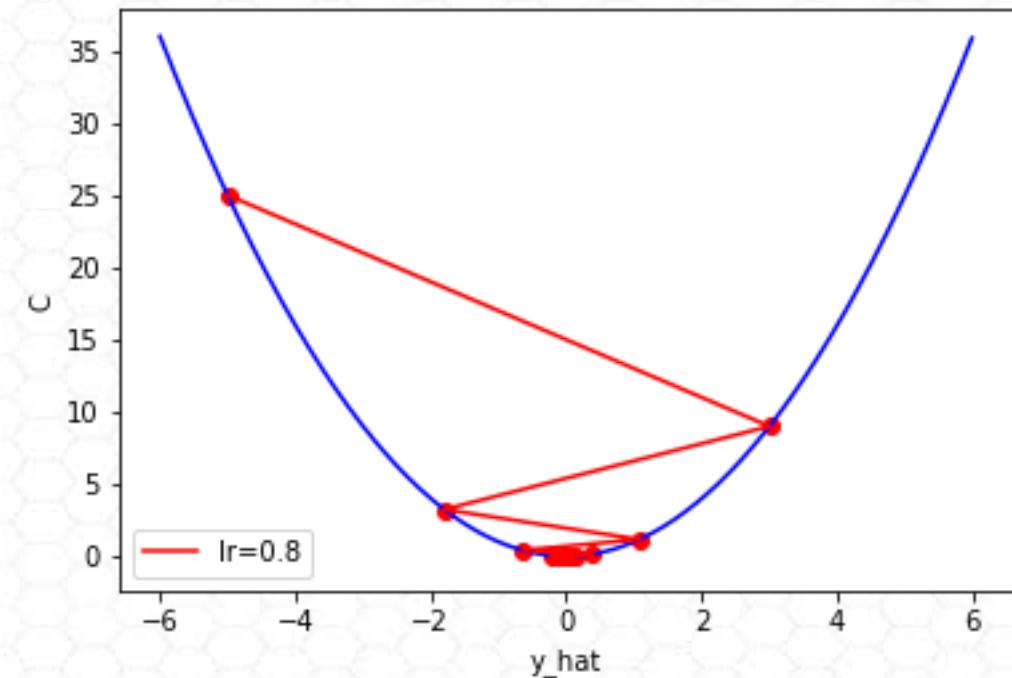
```
def gradient_descent(func, init_x, lr = 0.3, epochs = 100):
    x = init_x
    res = [x]
    for i in range(epochs):
        grad = dfunc(func,x)
        x = x - grad * lr
        res.append(x)
    return np.array(res)
```

$$x_i = x_i - \varepsilon \frac{\delta f}{\delta x}$$

梯度下降

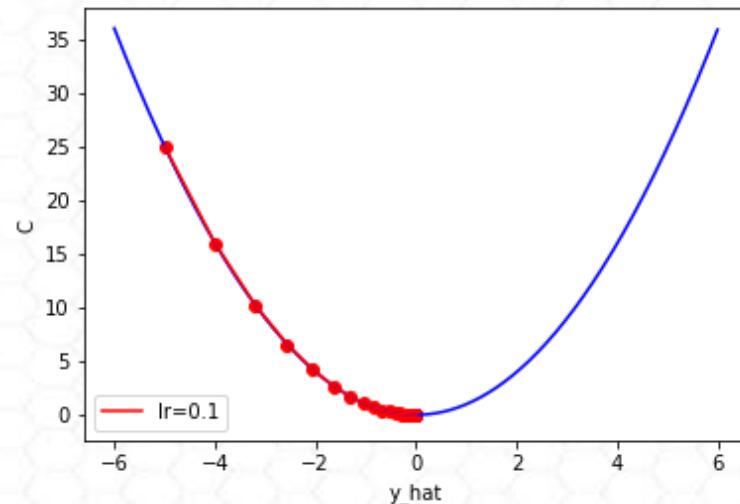
```
x = gradient_descent(func, -5, lr=0.8)
```

```
t = arange(-6.0, 6.0, 0.01)  
plt.plot(t, func(t), c='b')  
plt.plot(x, func(x), c='r')  
plt.scatter(x, func(x), c='r')
```

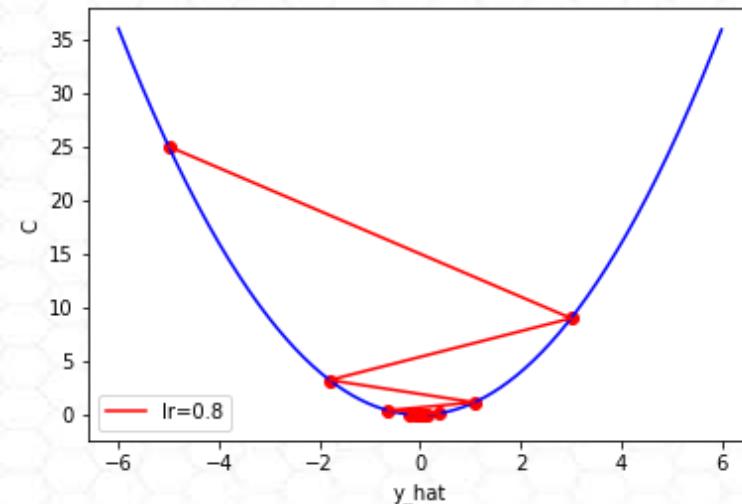


比較不同學習率

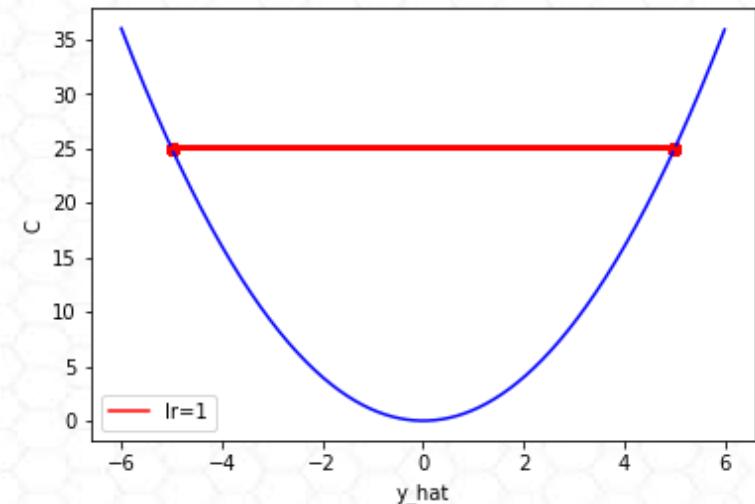
$$\varepsilon = 0.1$$



$$\varepsilon = 0.8$$

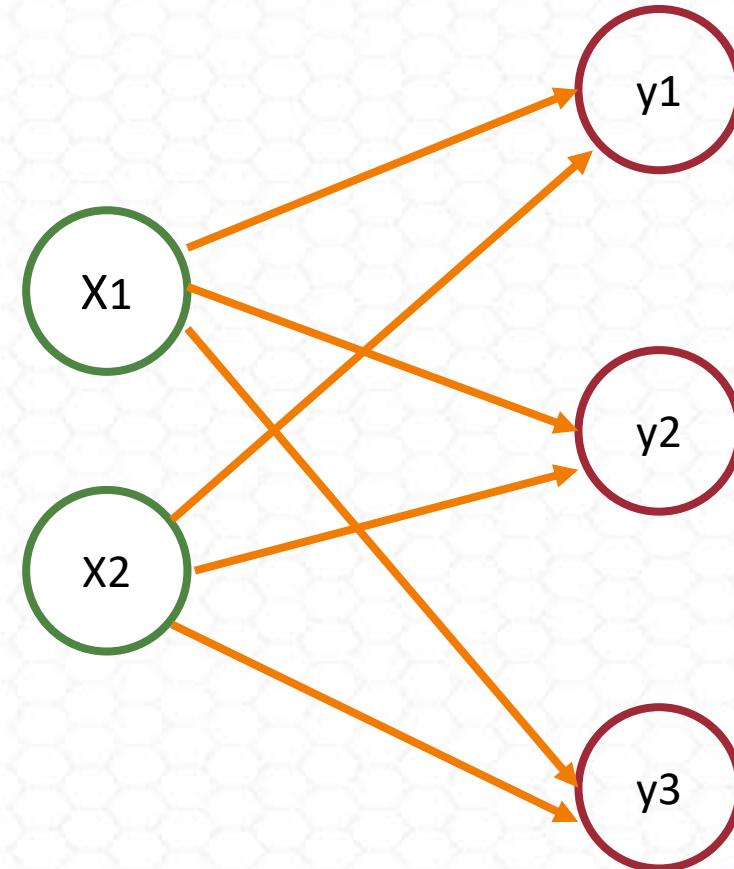


$$\varepsilon = 1$$



如何計算神經網路的梯度

```
# 初始網路  
x = np.array([0.6, 0.9])  
np.random.seed(42)  
weight = np.random.randn(2,3)  
z = np.dot(x, weight)  
# 取得預測值 y_hat  
y_hat = softmax_function(z)  
  
# 計算代價(損失)  
y = np.array([0, 0, 1])  
cross_entropy_err(y_hat, y)
```



計算偏微分通用公式

```
def dfunc(f, x):
    h = 1e-4
    grad = np.zeros_like(x)
    it = np.nditer(x, flags=['multi_index'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)
        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val
        it.iternext()
    return grad
```

計算神經網路梯度

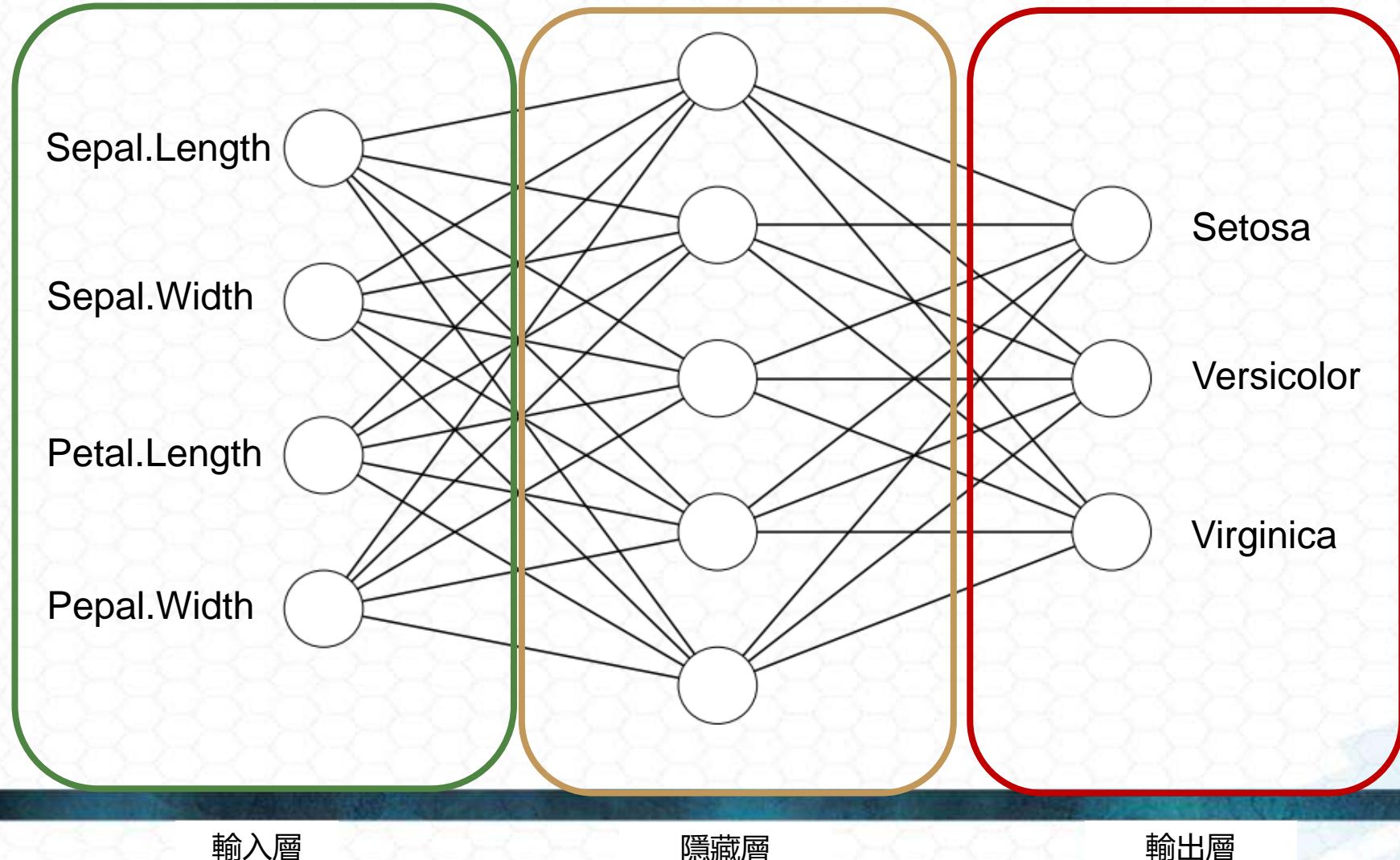
```
def predict(x):
    return np.dot(x, weight)

def loss(x, y):
    z = predict(x)
    y_hat = softmax_function(z)
    loss = cross_entropy_err(y_hat, y)
    return loss

func = lambda w: loss(x, y)
dfunc(func, weight)
```

訓練神經網路

建立一兩層神經網路



準備好建立網路所需資料

- 建立兩層神經網路

```
net = ANN(input_size=4, hidden_size=5, output_size=3)
```

- 準備好輸入值 Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

```
x= iris.data
```

- 建立 one-hot 編碼

```
y = np.zeros((len(iris.target), 3))  
for idx, val in enumerate(iris.target):  
    y[idx, val] = 1
```

訓練類神經網路

```
epochs = 3000
```

```
lr    = 0.01
```

```
train_loss = []
```

```
for i in range(epochs):
```

```
    grad = net.numerical_gradient(x,y)
```

```
    for key in ('W1', 'b1', 'W2', 'b2'):
```

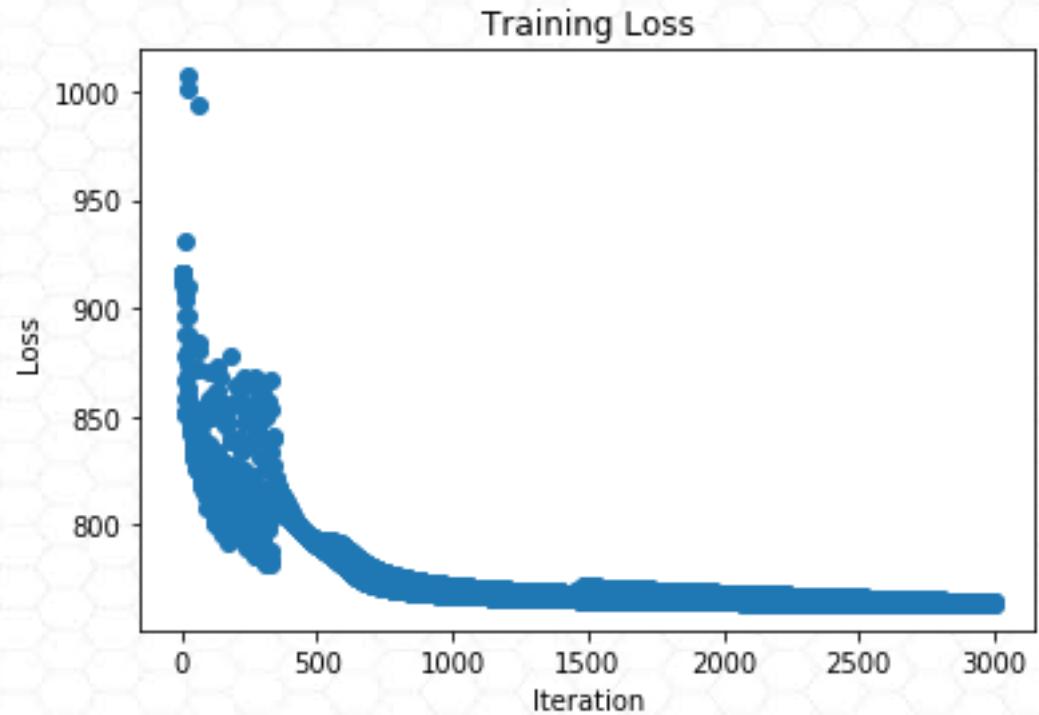
```
        net.params[key] = net.params[key] - lr * grad[key]
```

```
    loss = net.loss(x, y)
```

```
    train_loss.append(loss)
```

訓練損失圖

```
plt.scatter(range(0,3000),train_loss)  
plt.xlabel('Iteration')  
plt.ylabel('Loss')  
plt.title('Training Loss')
```



損失隨多次反覆運算後逐漸下降

驗證模型準確度

```
from sklearn.metrics import accuracy_score, confusion_matrix
predicted = np.argmax(net.predict(x), axis=1)

# accuracy
sum(predicted == iris.target) / len(iris.target)

# accuracy
accuracy_score(iris.target, predicted)

# confusion matrix
confusion_matrix(iris.target, predicted)
```

批次學習

- 如果碰到大資料，利用全部的資料學習相當耗工費時，因此如果只挑選部分的資料進行訓練，即可加快訓練速度。
- 調整交叉熵 Cross Entropy
 - 公式

$$C = -\frac{1}{N} \sum_n \sum_k y_{nk} \log \hat{y}_{nk}$$

- 實作

```
def cross_entropy_err(y_hat, y):  
    y      = y.reshape(1, y.size)  
    y_hat = y_hat.reshape(1, y_hat.size)  
    batch_size = y_hat.shape[0]  
    return -np.sum(y * np.log(y_hat)) / batch_size
```

隨機梯度下降

■ 批量梯度下降 v.s. 隨機梯度下降

- 批量梯度下降(Batch gradient descent)每反覆運算一步，都要用到訓練集所有的資料
- 隨機梯度下降法 (Stochastic Gradient Descent) 是通過隨機樣本來反覆運算更新一次，加速訓練過程

■ 隨機梯度下降演算法：

每一次反覆運算：

1. 隨機挑選批次資料
2. 計算各權重參數的梯度
3. 更新權重參數

批次學習

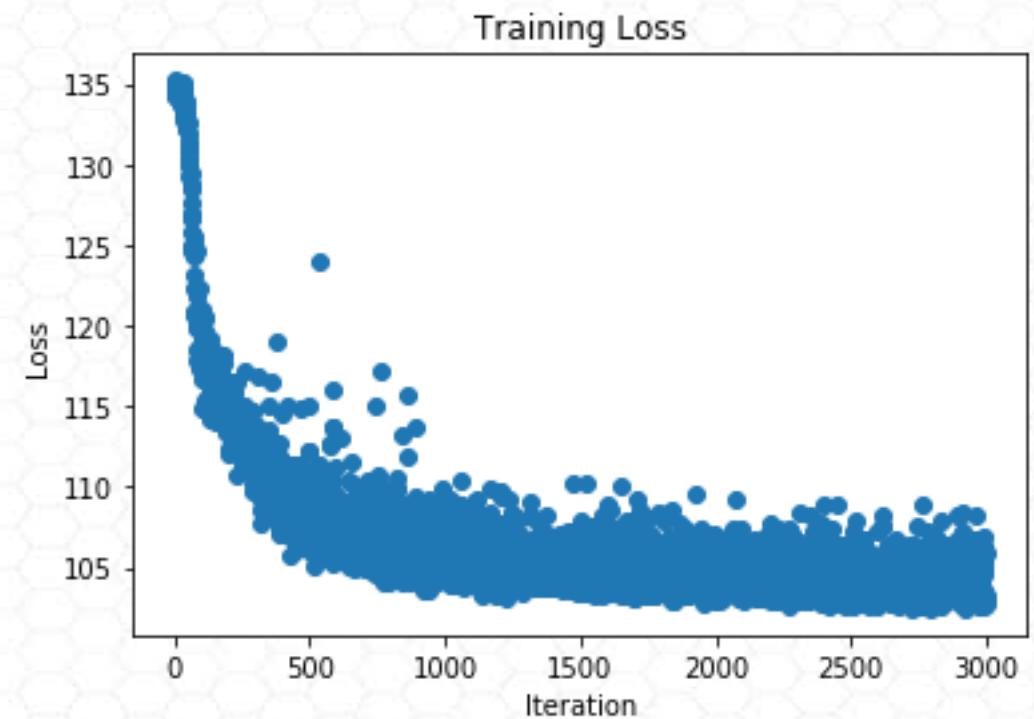
```
net = ANN(input_size=4, hidden_size=5, output_size=3)
```

```
epochs    = 3000  
lr        = 0.01  
batch_size = 30
```

```
train_loss = []  
for i in range(epochs):  
    idx = random.choice(iris.data.shape[0], batch_size)  
    x_batch = iris.data[idx]  
    y_batch = y[idx]  
    grad = net.numerical_gradient(x_batch,y_batch)  
    for key in ('W1', 'b1', 'W2', 'b2'):  
        net.params[key] = net.params[key] - lr * grad[key]  
    loss = net.loss(x_batch, y_batch)  
    train_loss.append(loss)
```

訓練損失圖

```
plt.scatter(range(0,3000),train_loss)  
plt.xlabel('Iteration')  
plt.ylabel('Loss')  
plt.title('Training Loss')
```



驗證批次學習模型準確度

```
from sklearn.metrics import accuracy_score, confusion_matrix
predicted = np.argmax(net.predict(x), axis=1)

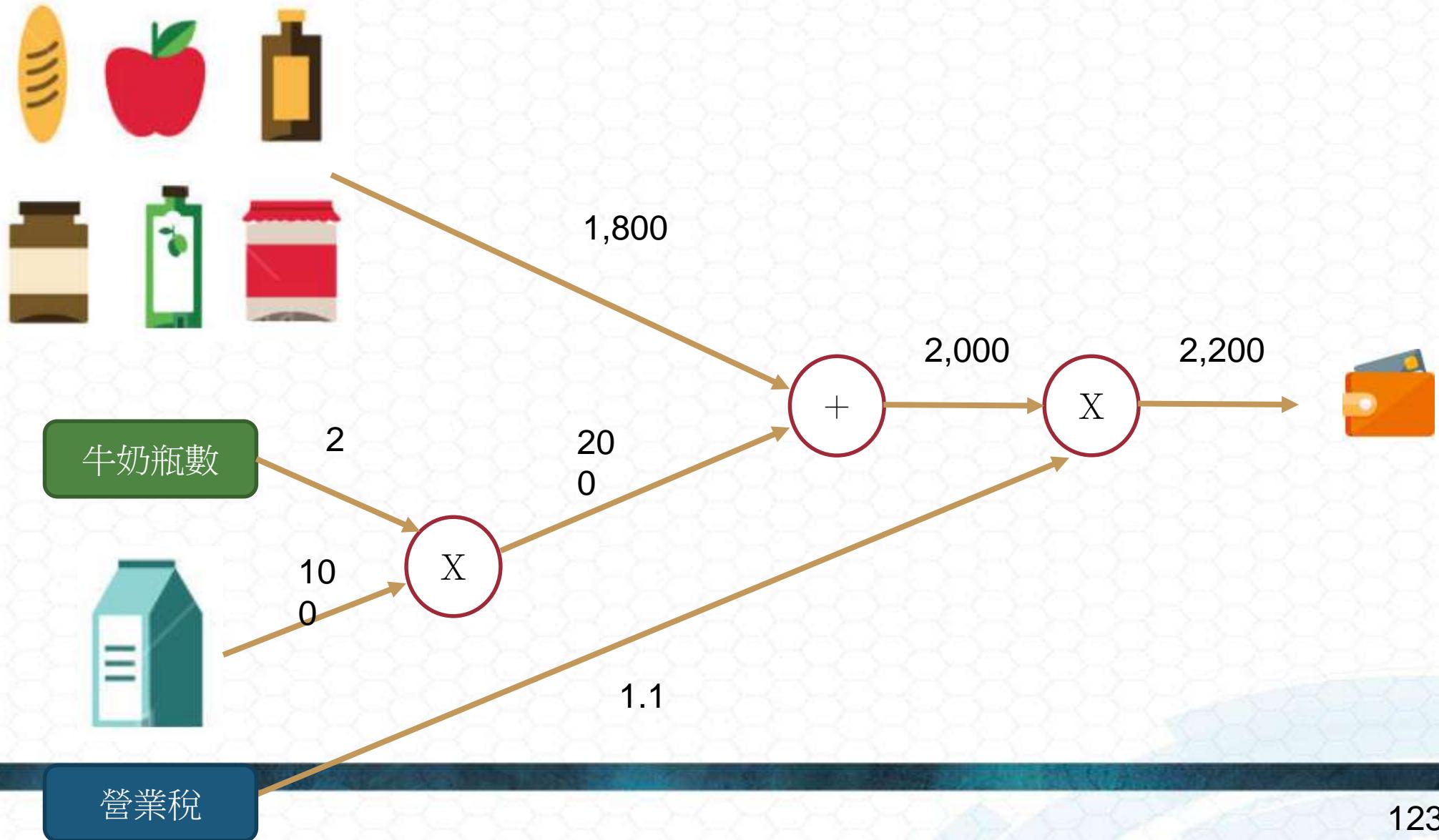
# accuracy
sum(predicted == iris.target) / len(iris.target)

# accuracy
accuracy_score(iris.target, predicted)

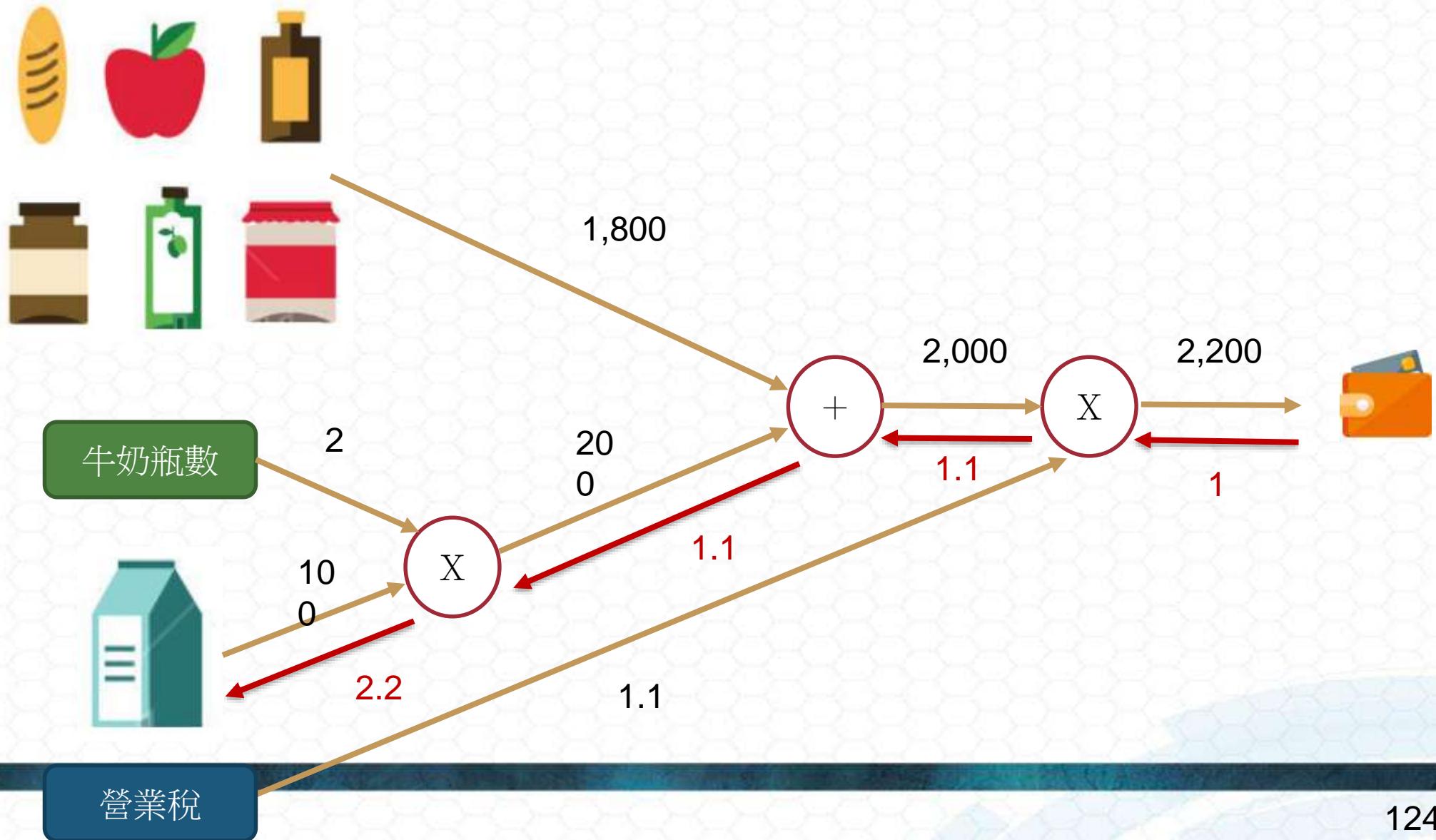
# confusion matrix
confusion_matrix(iris.target, predicted)
```

反向傳播演算法

計算圖 (Computational Graph)



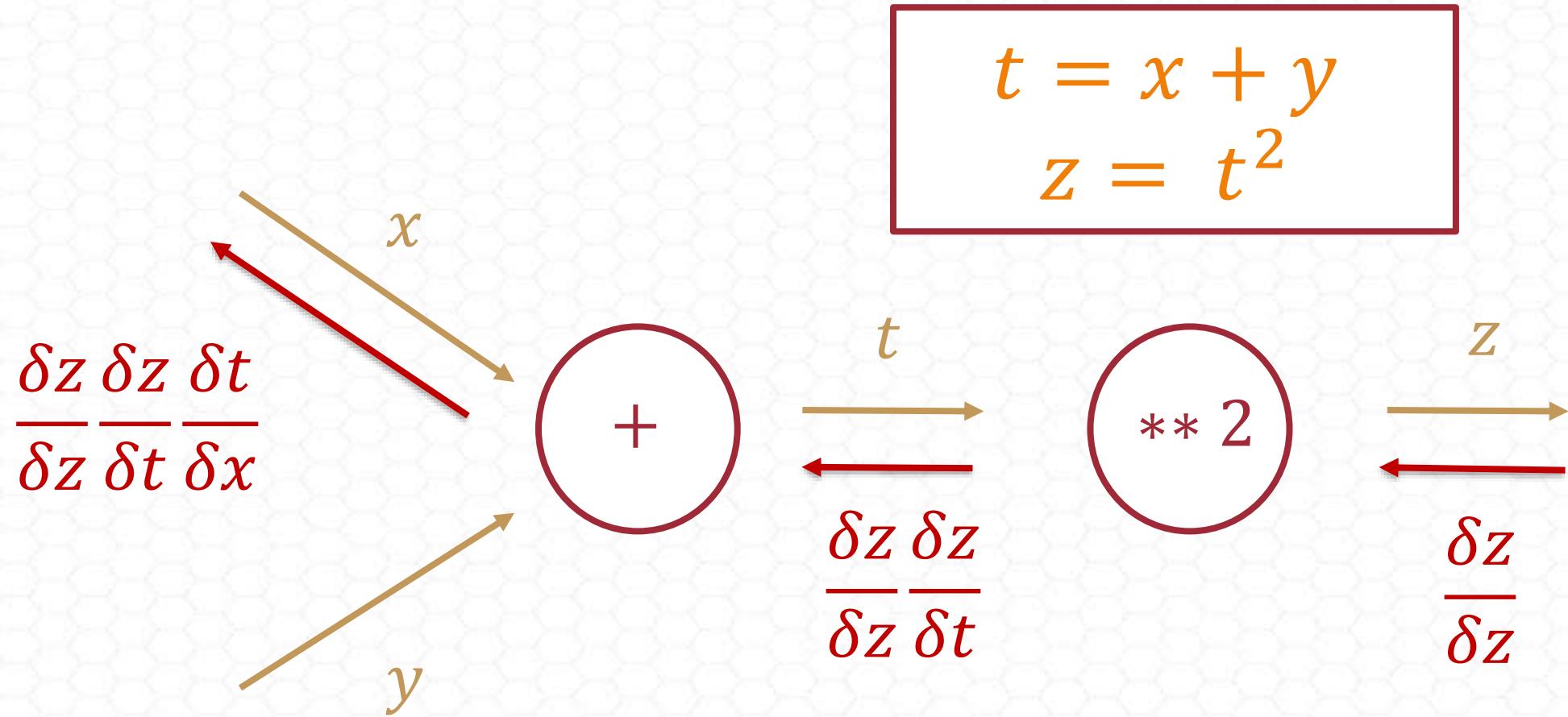
計算圖 (Computational Graph)



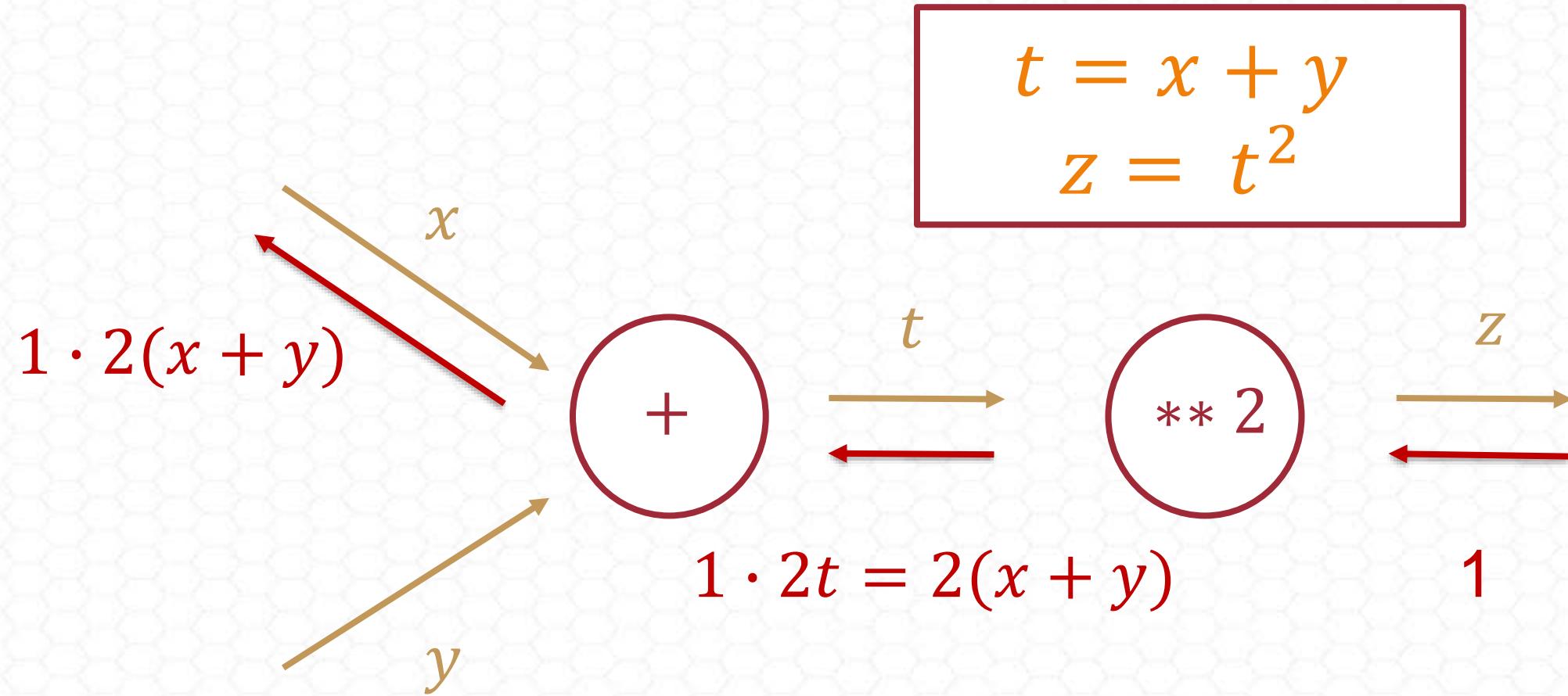
反向傳播



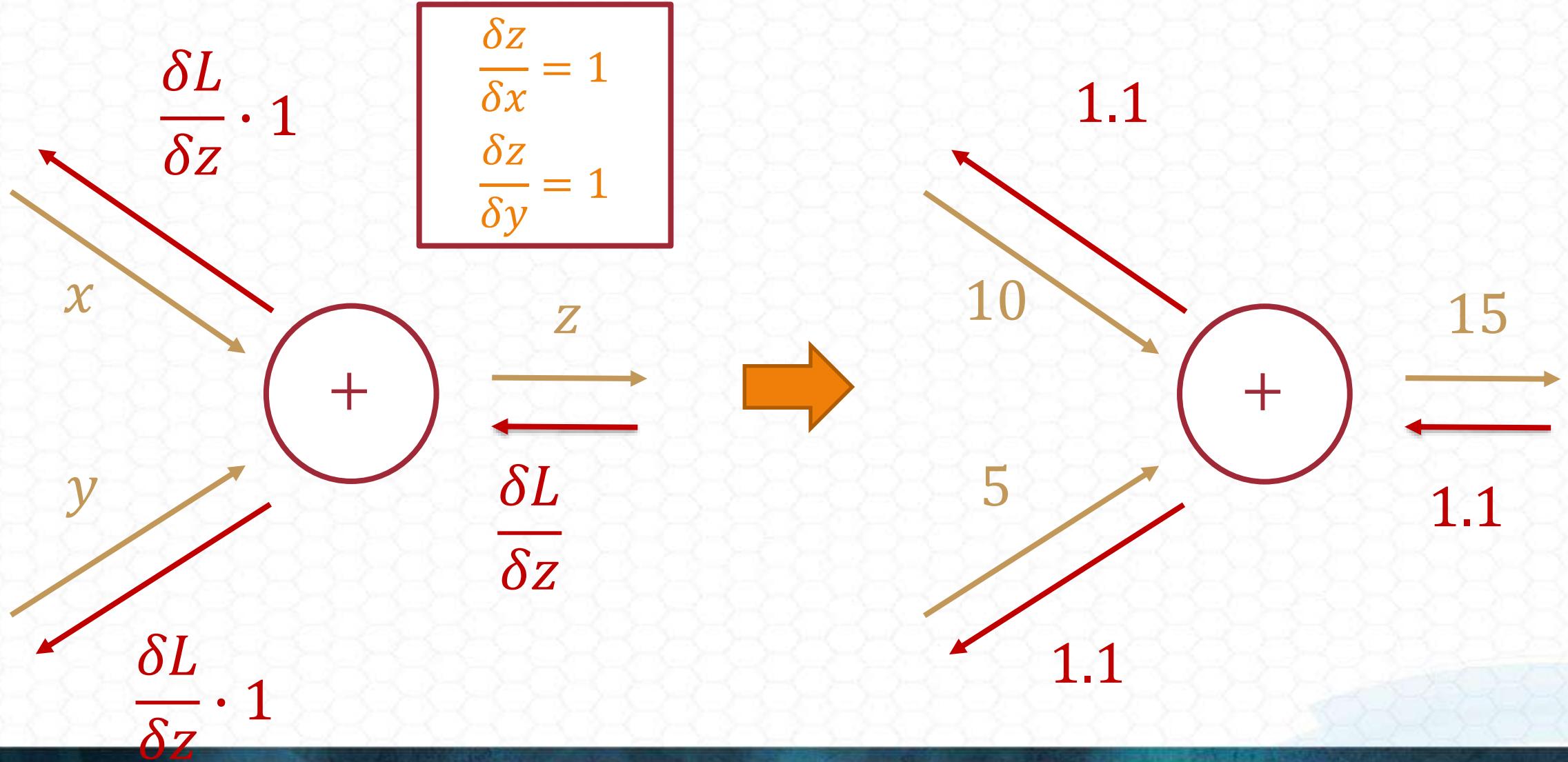
反向傳播



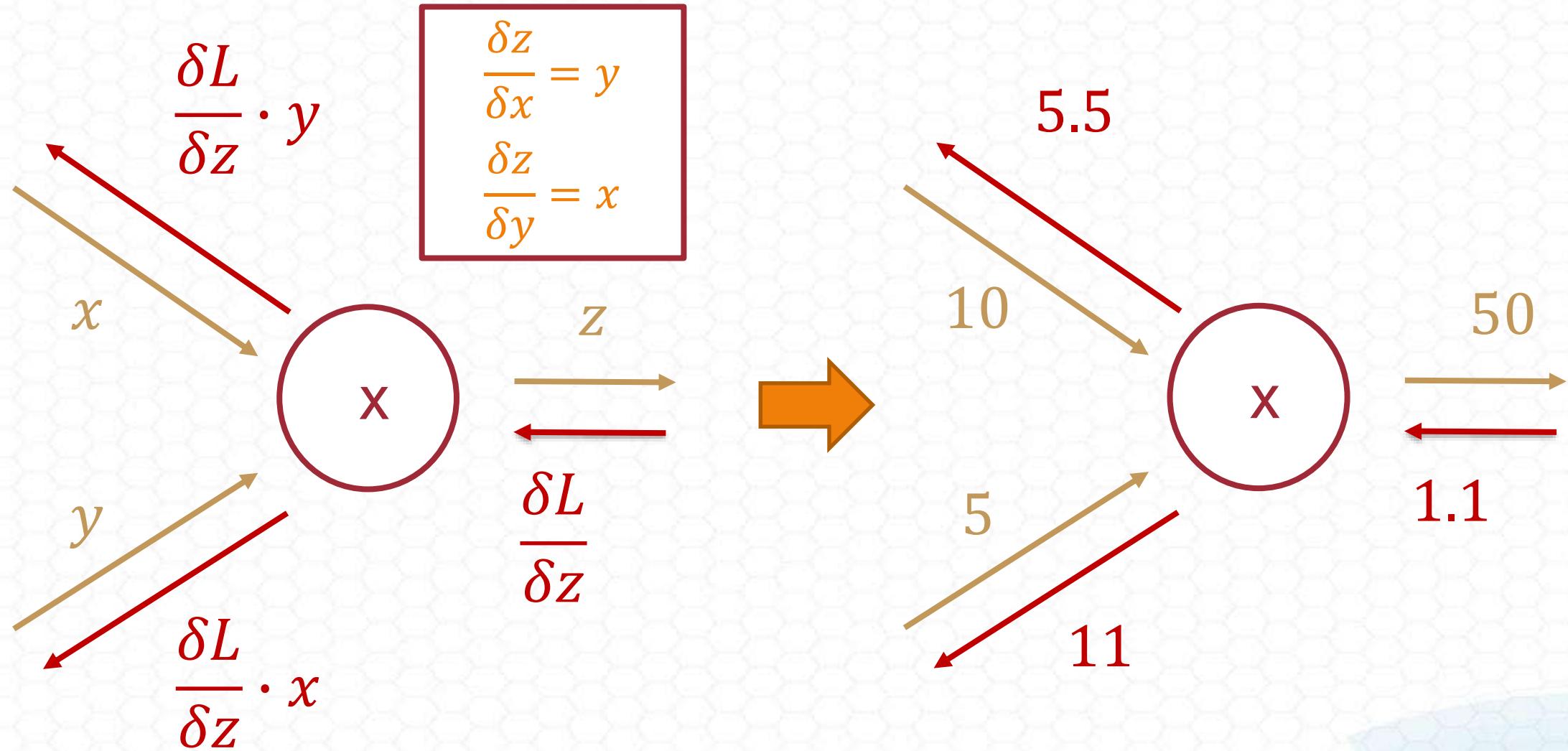
反向傳播



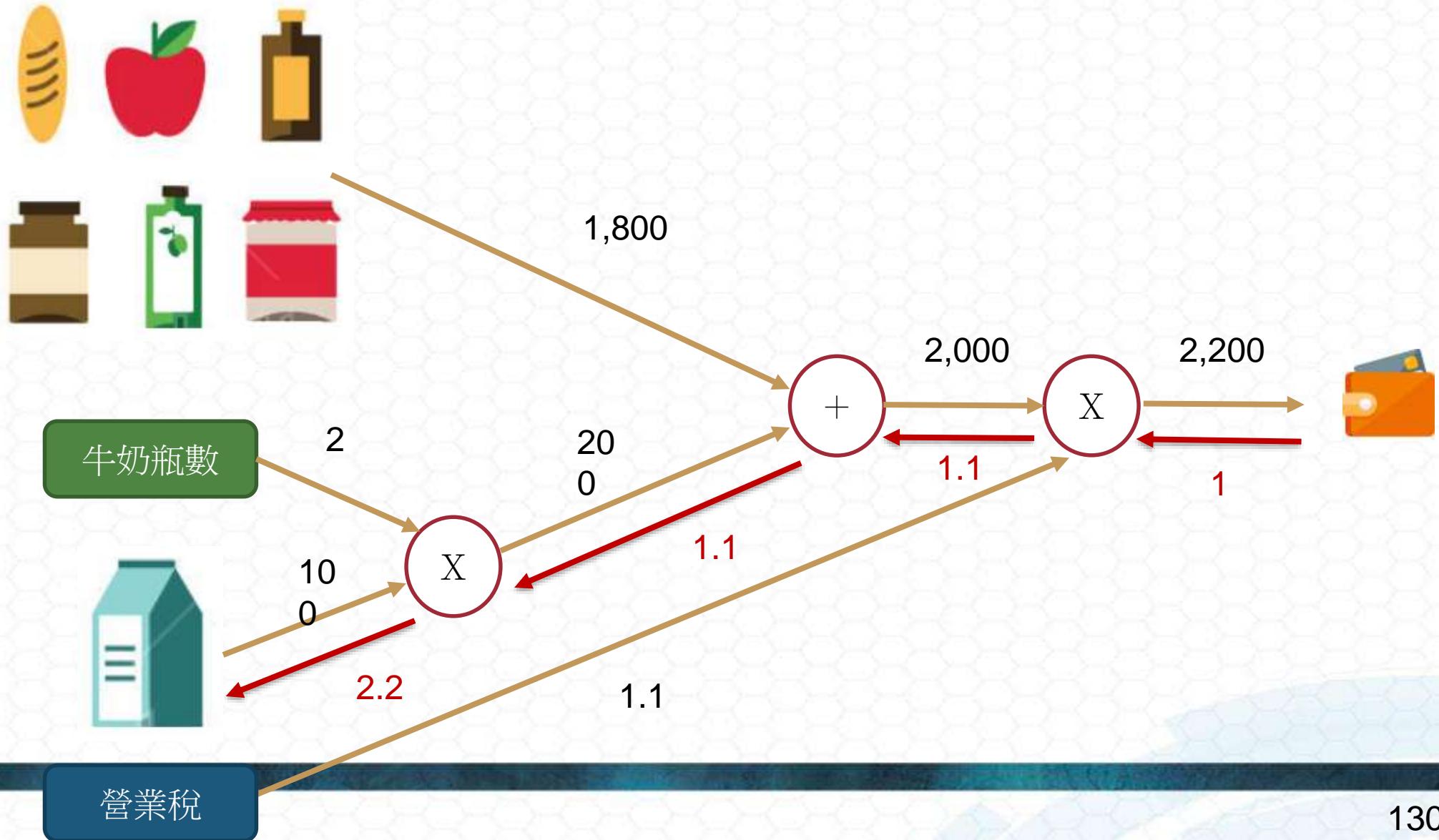
加法反向傳播



乘法反向傳播

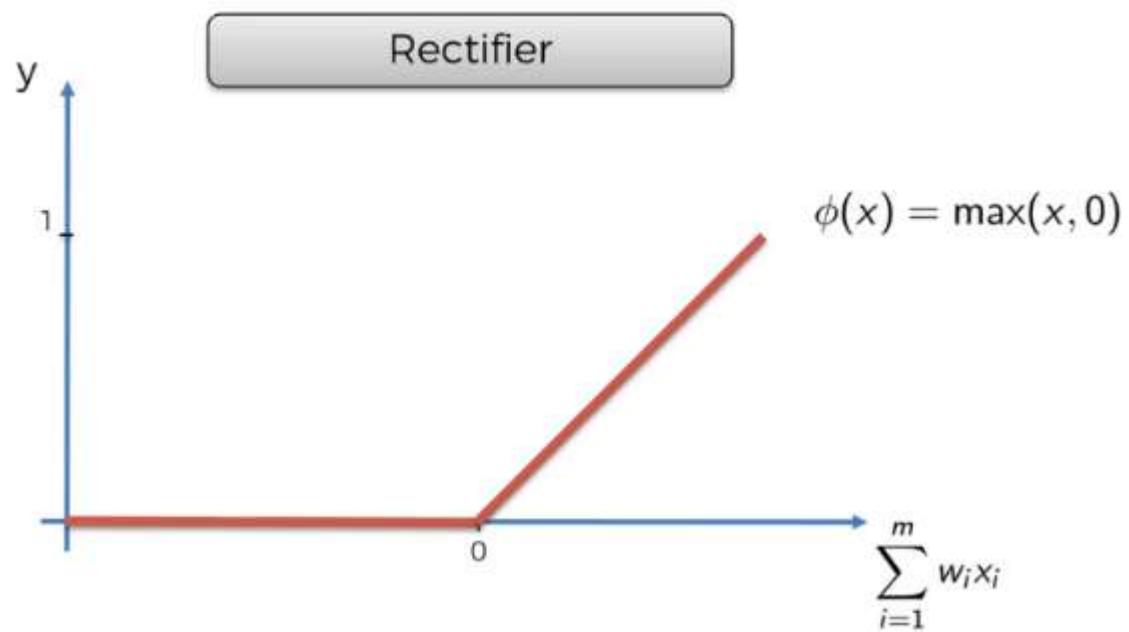


計算圖 (Computational Graph)

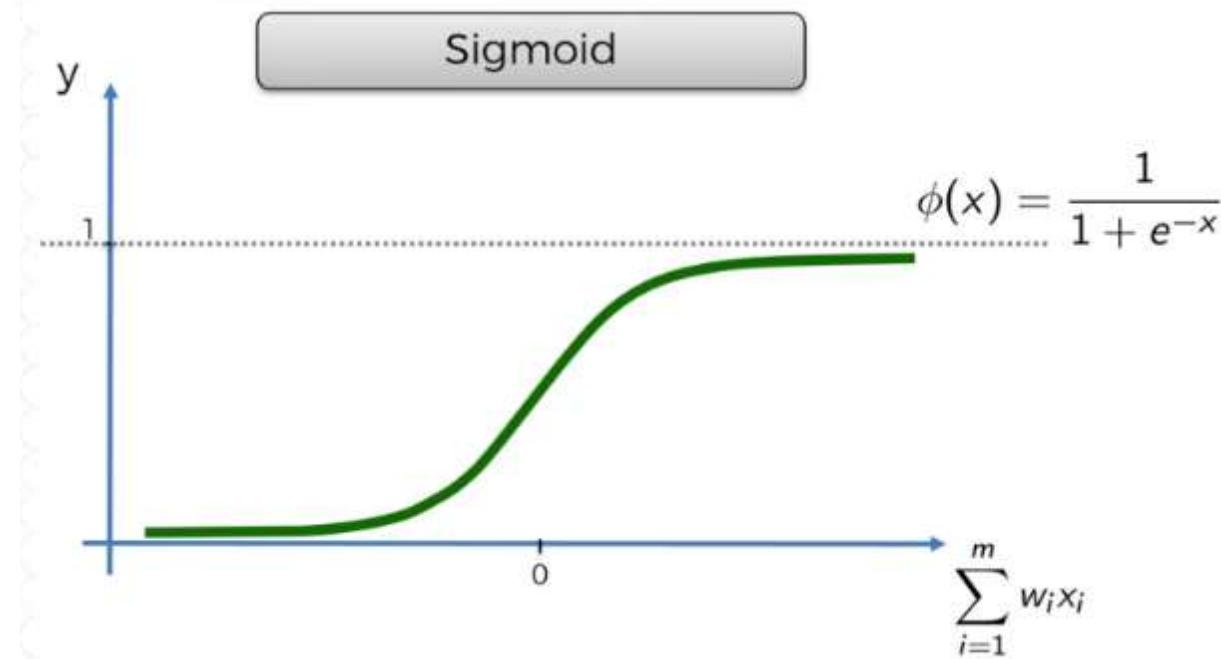


將計算圖套用在啟動函數上

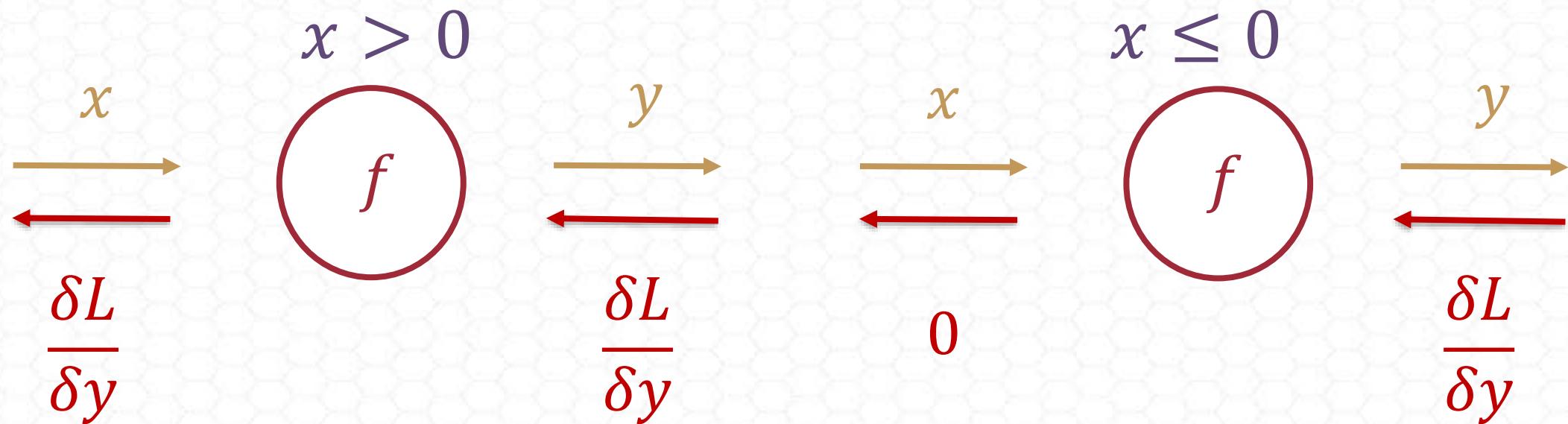
ReLU 層



Sigmoid 層



ReLU 反向傳播

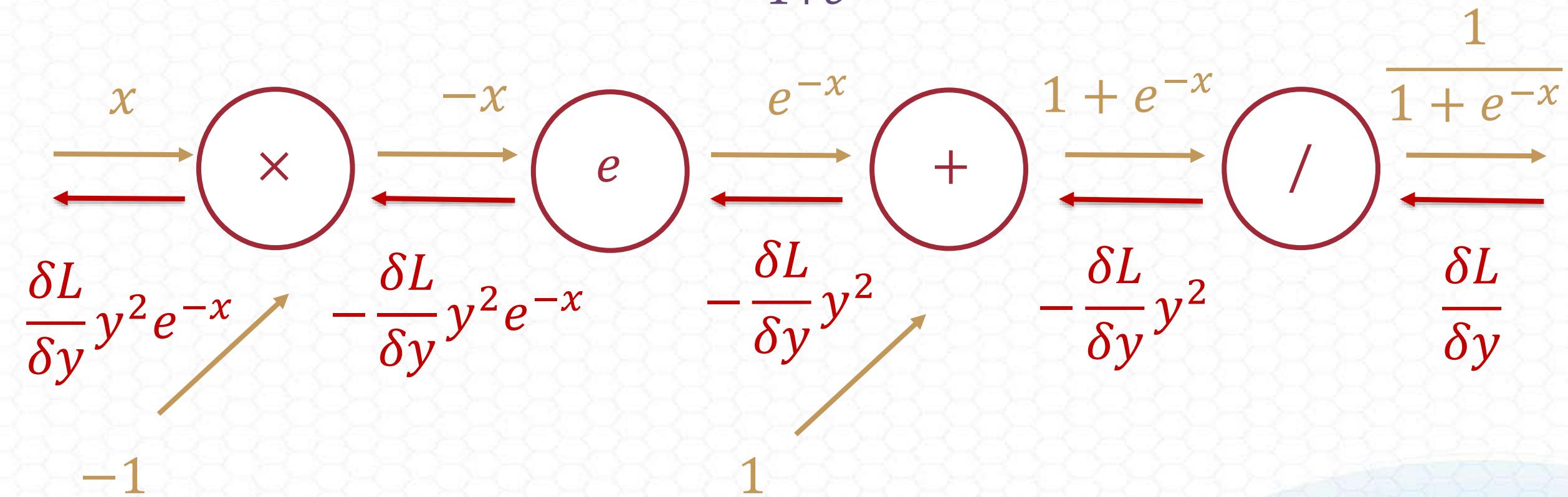


ReLU反向傳播實作

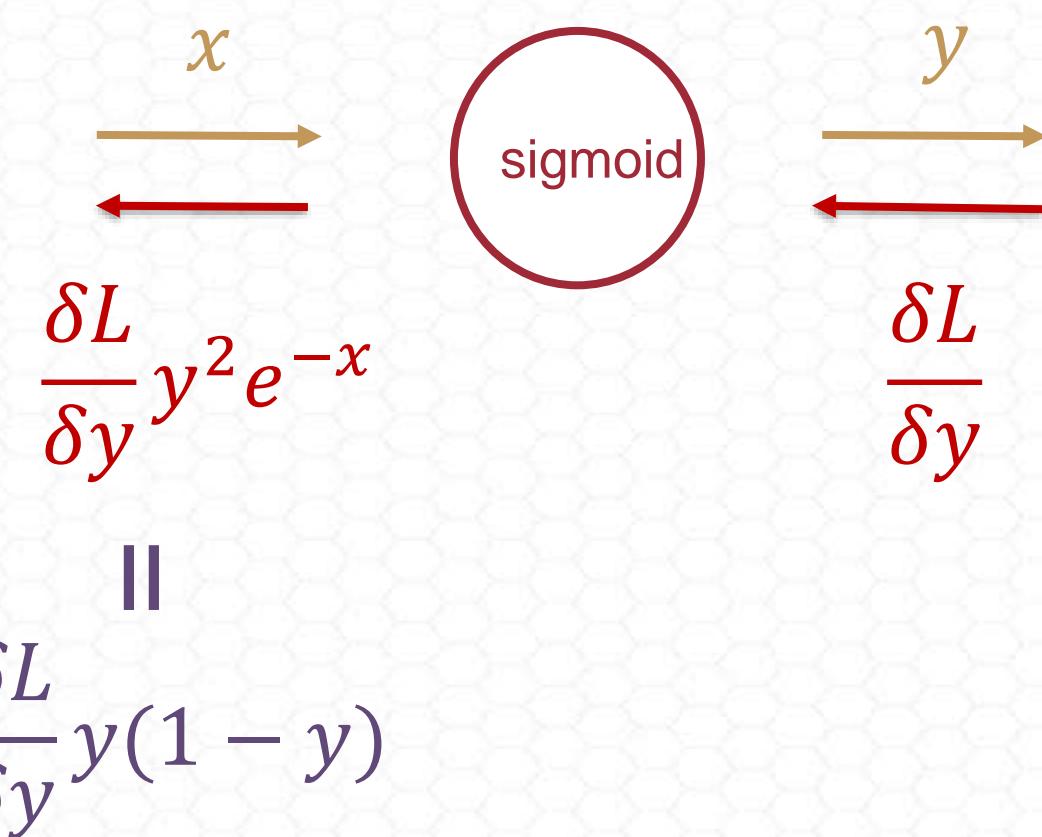
```
class Relu:  
    def __init__(self):  
        self.cache = None  
  
    def forward(self, x):  
        self.cache = (x <=0)  
        out = np.maximum(0,x)  
        return out  
  
    def backward(self, dout):  
        dout[self.cache] = 0  
        dx = dout  
        return dx
```

Sigmoid 反向傳播

$$y = \frac{1}{1+e^{-x}}$$



Sigmoid 反向傳播（簡化版本）

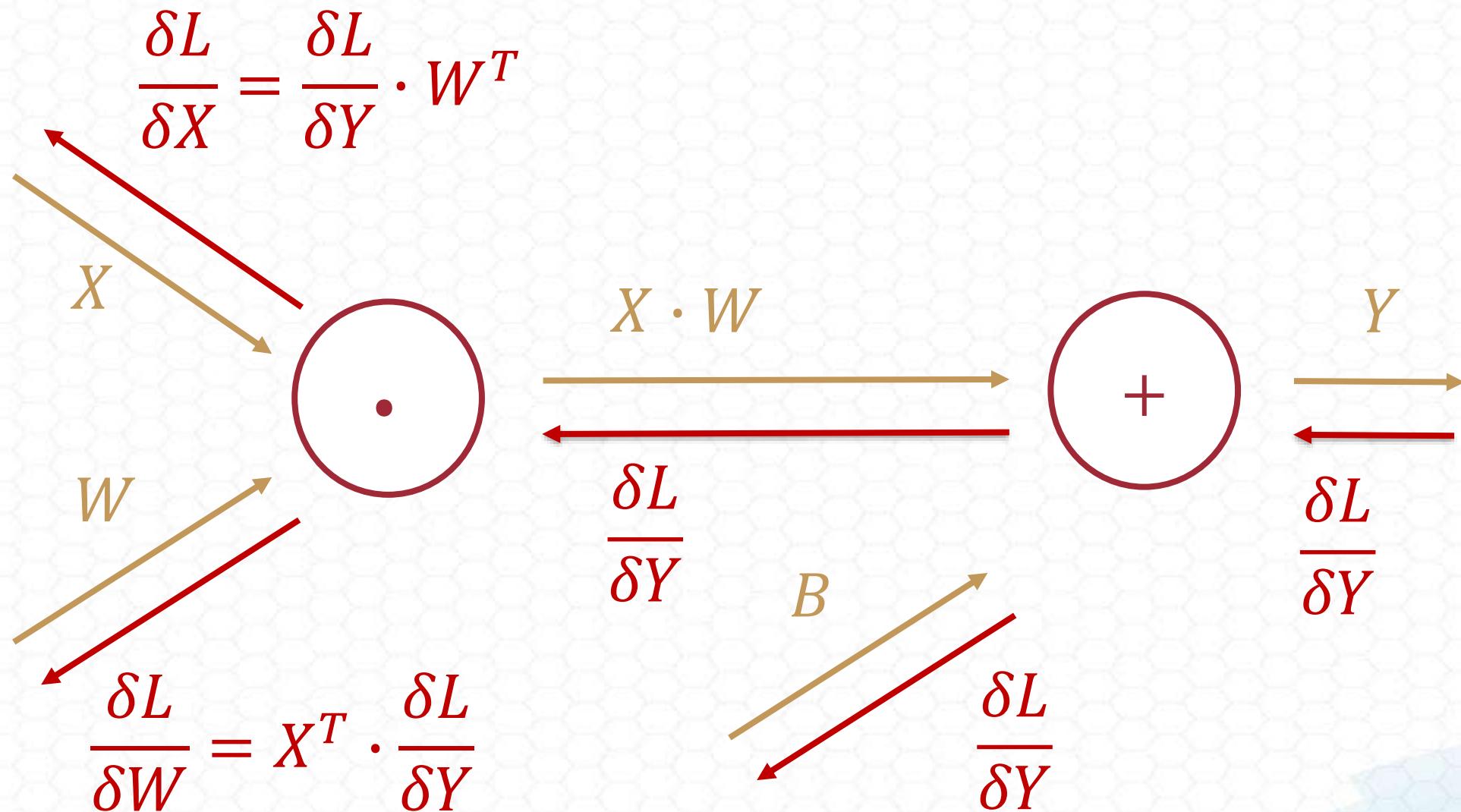


$$\begin{aligned}\frac{\delta L}{\delta y} y^2 e^{-x} &= \frac{\delta L}{\delta y} \left(\frac{1}{1+e^{-x}}\right)^2 e^{-x} \\&= \frac{\delta L}{\delta y} \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} \\&= \frac{\delta L}{\delta y} y(1-y)\end{aligned}$$

Sigmoid反向傳播實作

```
class Sigmoid:  
    def __init__(self):  
        self.out = None  
  
    def forward(self, x):  
        out = 1 / (1 + np.exp(-x))  
        self.out = out  
        return out  
  
    def backward(self, dout):  
        y = self.out  
        dx = dout * y * (1-y)  
        return dx
```

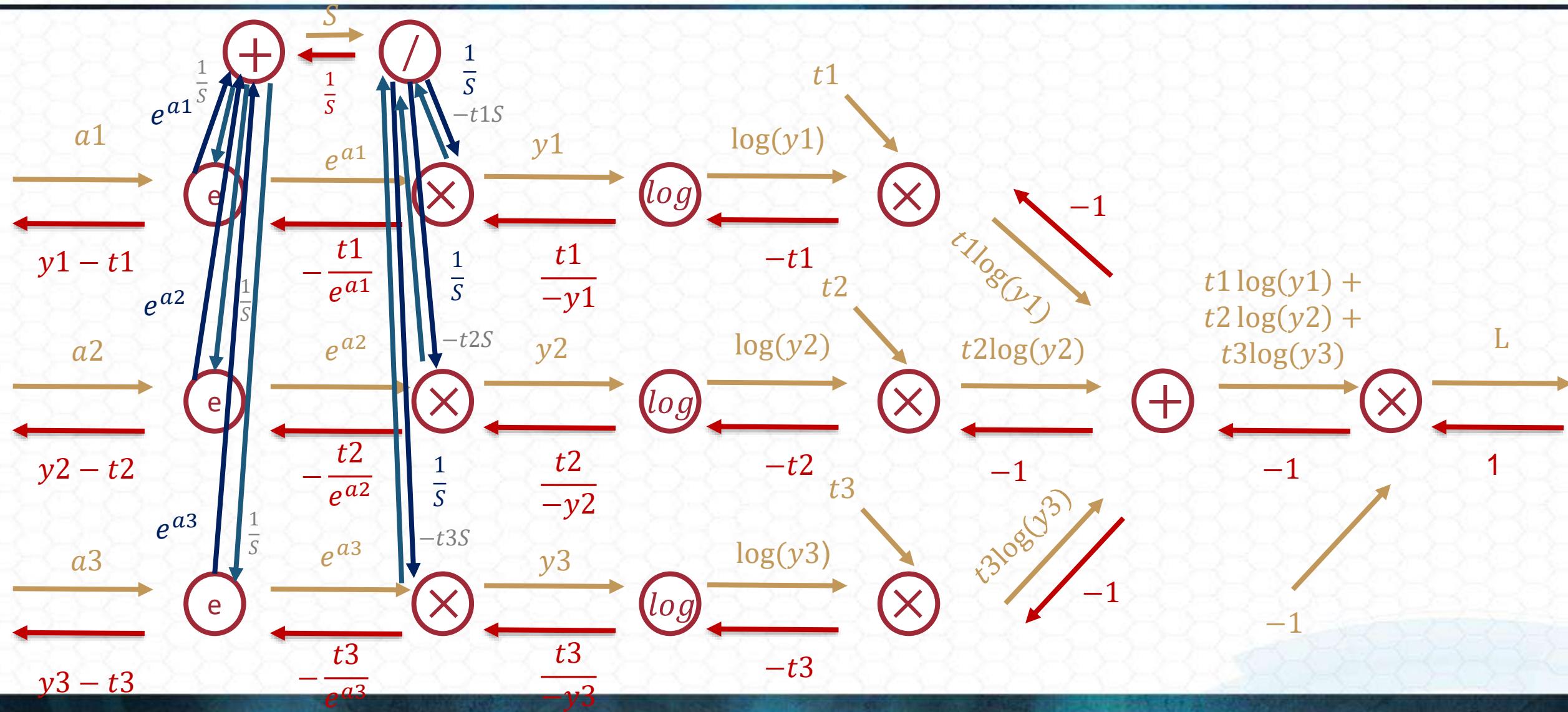
仿射映射 (Affine Transformation)



仿射映射 (Affine Transformation) 反向傳播實作

```
class Affine:  
    def __init__(self, W, b):  
        self.W = W  
        self.b = b  
        self.x = None  
        self.dW = None  
        self.db = None  
  
    def forward(self, x):  
        self.x = x  
        out = np.dot(self.x, self.W) + self.b  
        return out  
  
    def backward(self, dout):  
        dx = np.dot(dout, self.W.T)  
        self.dW = np.dot(self.x.T, dout)  
        self.db = np.sum(dout)  
        return dx
```

Softmax with Loss 推導



Softmax with Loss 反向傳播實作

```
class SoftmaxWithLoss:  
    def __init__(self):  
        self.loss = None  
        self.y_hat = None  
        self.y = None  
  
    def forward(self, x, y):  
        self.y = y  
        self.y_hat = softmax_function(x)  
        self.loss = cross_entropy_err(self.y_hat, self.y)  
        return self.loss  
  
    def backward(self, dout=1):  
        batch_size = self.y.shape[0]  
        dx = (self.y_hat - self.y) / batch_size  
        return dx
```

使用Stochastic Gradient Descent 訓練ANN

1. 隨機設定近似於0 (但不等於0)的權重
2. 將第一筆資料的每個特徵喂進輸入層的每個結點
3. 向前傳遞 (Forward Propagation): 從左至右，神經元透過網路傳遞訊息直到產生預測值
4. 比較預測值與實際值的誤差
5. 向後傳遞 (Backward Propagation): 從右至左，根據錯誤調整權重，透過學習率 (Learning Rate) 決定我們要調整多少權重

使用 SCIKIT-LEARN 訓練神經網路

如何使用Scikit-Learn 建立ANN

label = 5



label = 0



label = 4



label = 1



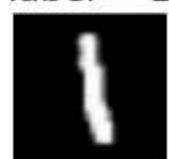
label = 9



label = 2



label = 1



label = 3



label = 1



label = 4



label = 3



label = 5



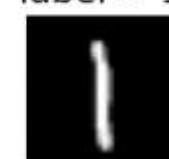
label = 3



label = 6



label = 1



label = 7



label = 2



label = 8



label = 6



label = 9

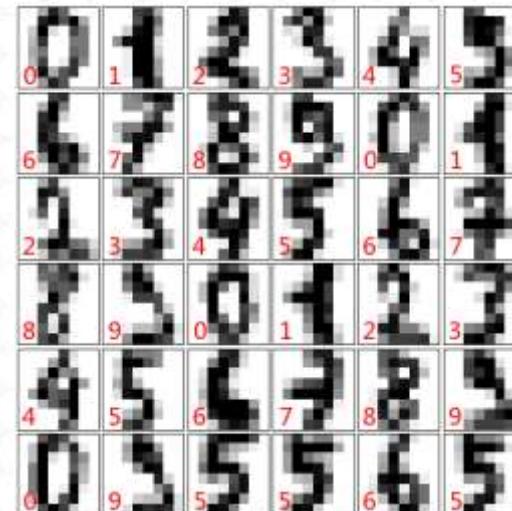


讀取數字資料(MNIST)

```
from sklearn.datasets import load_digits  
import matplotlib.pyplot as plt  
from sklearn.neural_network import MLPClassifier  
from sklearn.preprocessing import StandardScaler  
import numpy as np  
digits = load_digits()
```

繪製數字資料

```
fig = plt.figure(figsize = (8,8))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for i in range(36):
    ax = fig.add_subplot(6, 6, i+1, xticks=[], yticks[])
    ax.imshow(digits.images[i],cmap=plt.cm.binary,interpolation='nearest')
    ax.text(0, 7, str(digits.target[i]), color="red", fontsize = 20)
```



變更數字資料的尺度

```
scaler = StandardScaler()  
scaler.fit(digits.data)  
X_scaled = scaler.transform(digits.data)
```

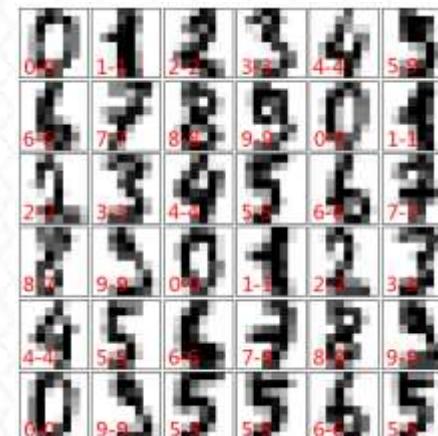
使用MLPClassifier

```
mlp = MLPClassifier(hidden_layer_sizes=(30,30,30),  
                    activation='relu', max_iter = 100,  
                    solver='sgd',learning_rate='constant',  
                    learning_rate_init=0.001)
```

```
mlp.fit(X_scaled,digits.target)
```

產生預測結果

```
predicted = mlp.predict(X_scaled)
fig = plt.figure(figsize = (8,8))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for i in range(36):
    ax = fig.add_subplot(6, 6, i+1, xticks=[], yticks[])
    ax.imshow(digits.images[i],cmap=plt.cm.binary,interpolation='nearest')
    ax.text(0, 7, str('{})-{}'.format(digits.target[i],predicted[i])), color="red", fontsize = 20)
```



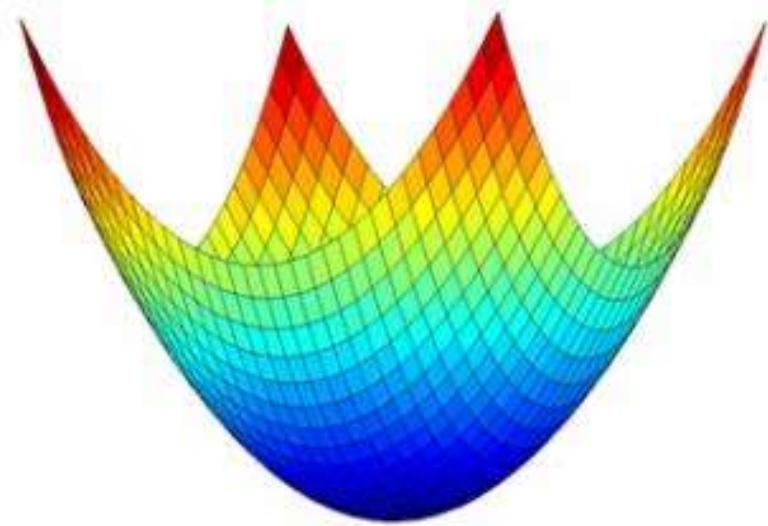
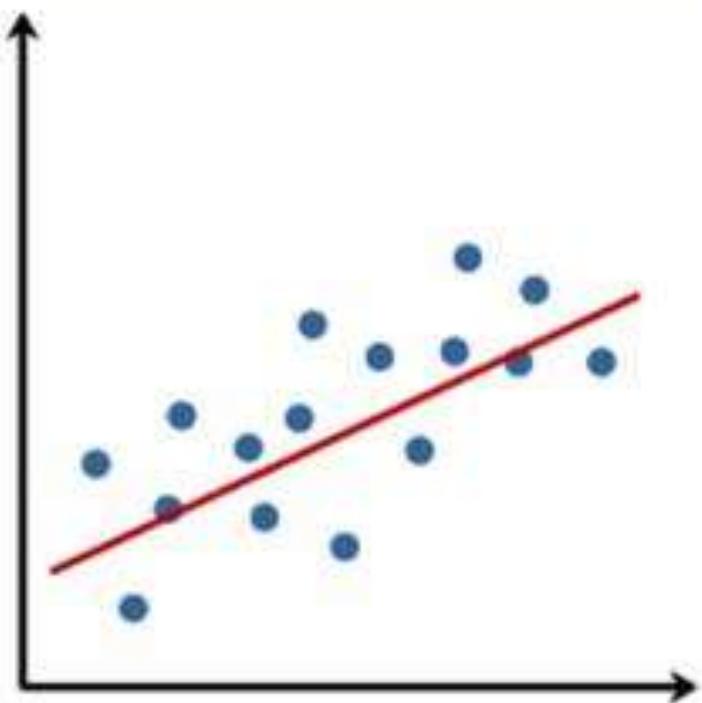
驗證模型準確度

```
from sklearn.metrics import accuracy_score, confusion_matrix  
  
# accuracy  
accuracy_score(digits.target, predicted)  
  
# confusion matrix  
confusion_matrix(digits.target, predicted)
```

梯度消失

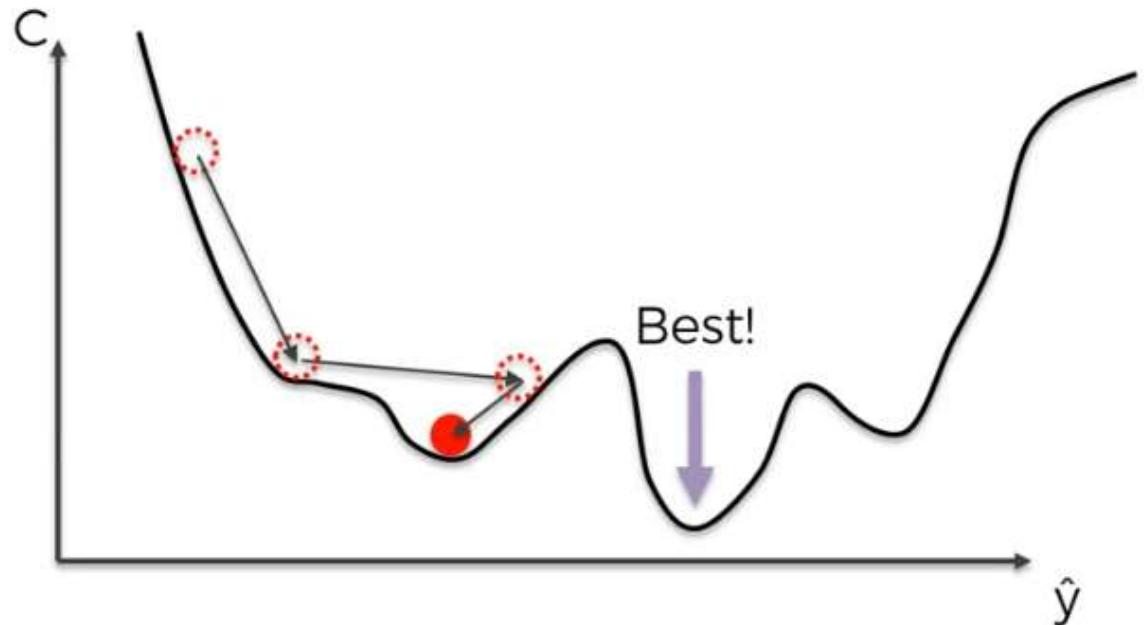
線性問題求解

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



局部最佳解(Local Minimum)

- 一路從頭找到最低點可能會陷入局部最佳解(Local Minimum) 等問題

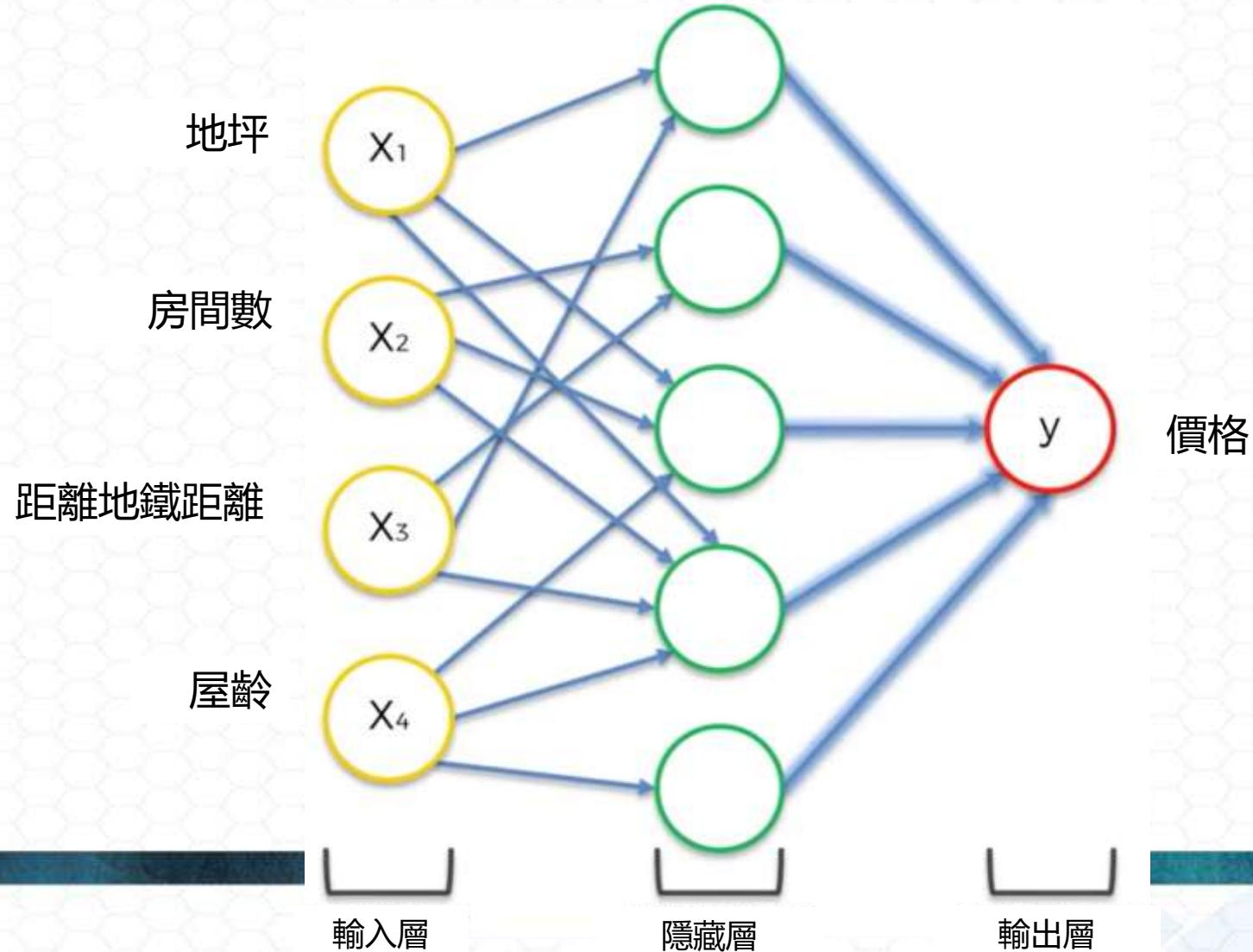


「梯度消失」 (Vanishing Gradient)

代價函數為非凸函數，求解時容易陷入局部最優解、而非全域最優解

傳統類神經網路的瓶頸

■ 神經網路只要超過 3 層以上就幾乎沒有效果

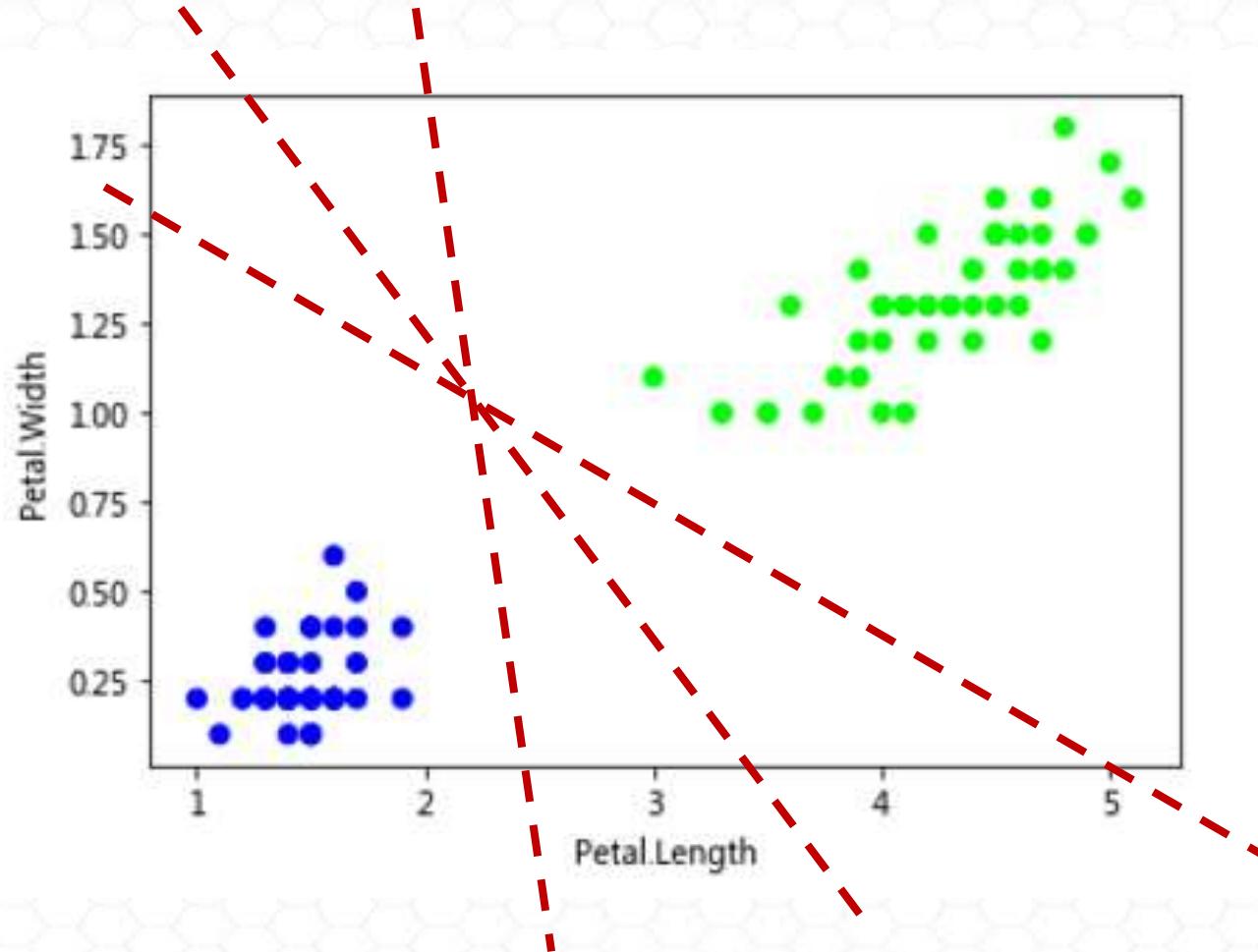


傳統神經網路只能處理淺層結構

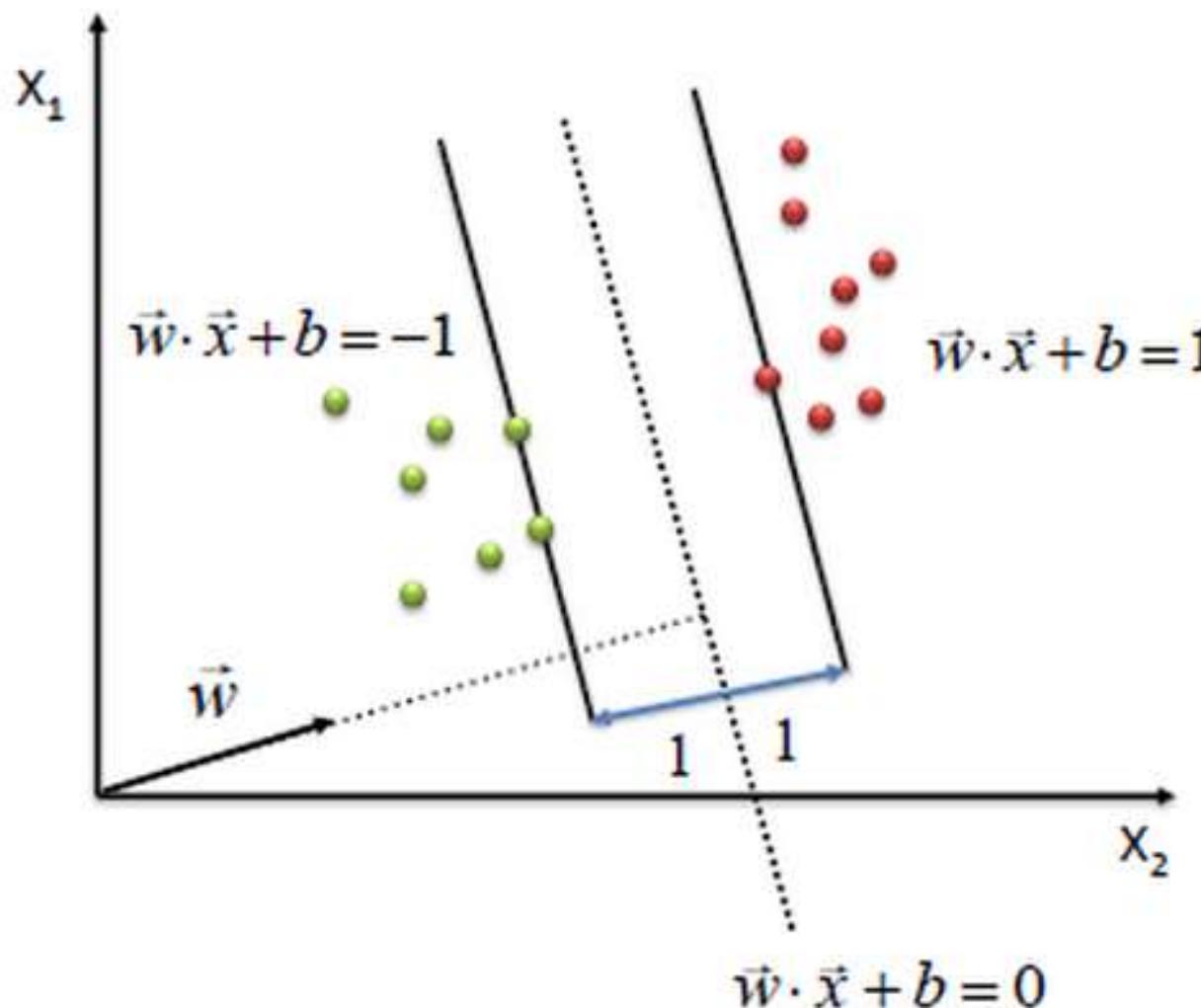
- 梯度消失問題會隨著神經網路層數的增加而更加嚴重，所以傳統類神經網路只能處理淺層結構（比如 2 層）的網路，從而限制了性能
- 單層感知機、和多層感知機失敗，導致1980的學界認為類神經網路是死胡同
- 在 1990 年代，支持向量機 (Support Vector Machine, SVM) 等「淺層機器學習模型」成為主流技術，此為機器學習的第二波浪潮

支持向量機

該選哪一條線做切割？



支持向量機 (Support Vector Machine)



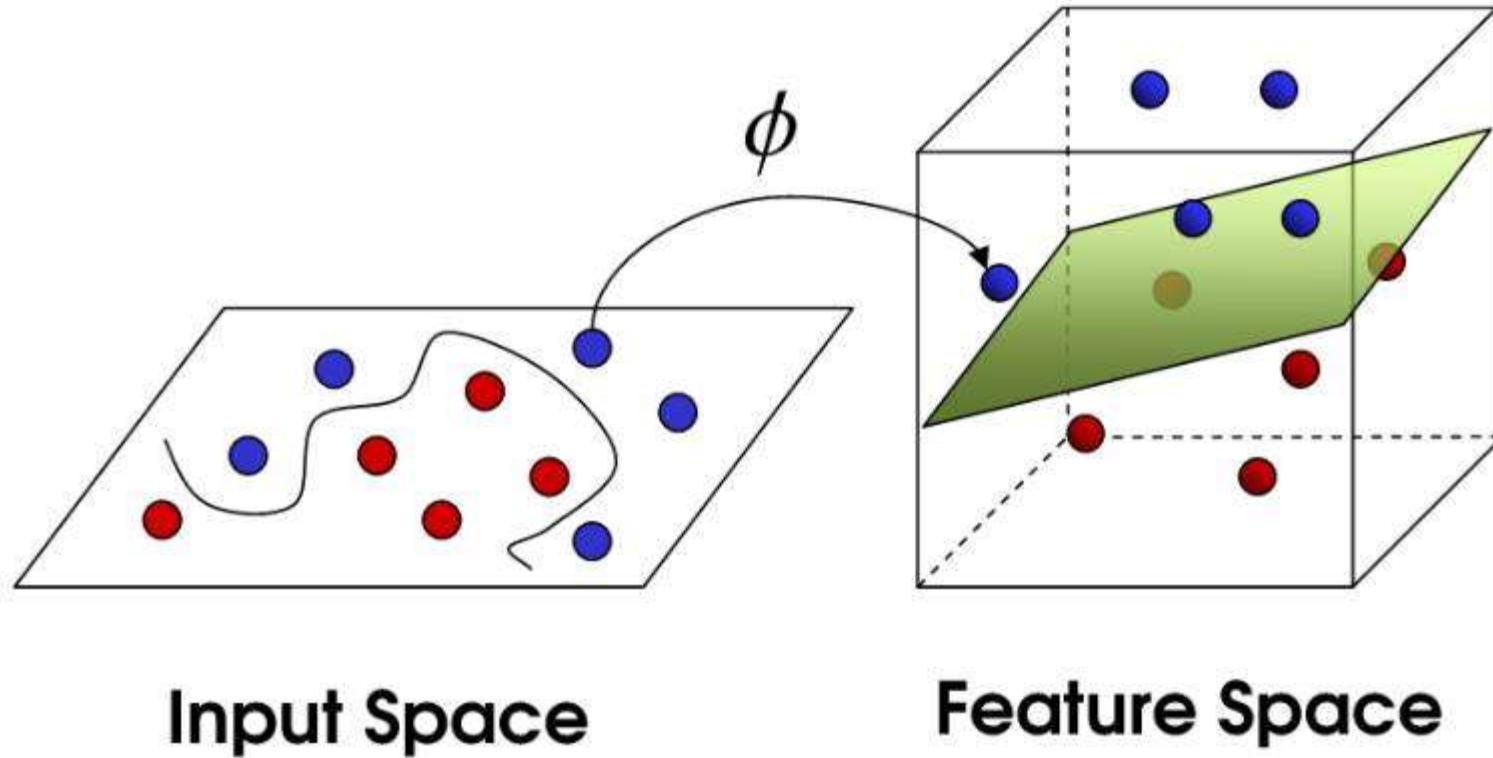
$$\max \frac{2}{\|\vec{w}\|}$$

s.t.

$$(\vec{w} \cdot \vec{x} + b) \geq 1, \forall x \text{ of class 1}$$

$$(\vec{w} \cdot \vec{x} + b) \leq -1, \forall x \text{ of class 2}$$

解決高維度資料切分問題



Input Space

Feature Space

建立支持向量機 (1)

```
from sklearn.datasets import load_iris  
from sklearn.svm import SVC  
from sklearn.linear_model import LogisticRegression  
iris = load_iris()  
  
X = iris.data[0:100,[2,3]]  
y = iris.target[0:100]  
  
clf1 = SVC(kernel="linear")  
clf1.fit(X, y)  
  
clf2 = LogisticRegression()  
clf2.fit(X, y)
```

建立支持向量機 (2)

```
def plot_estimator(estimator, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                          np.arange(y_min, y_max, 0.1))
```

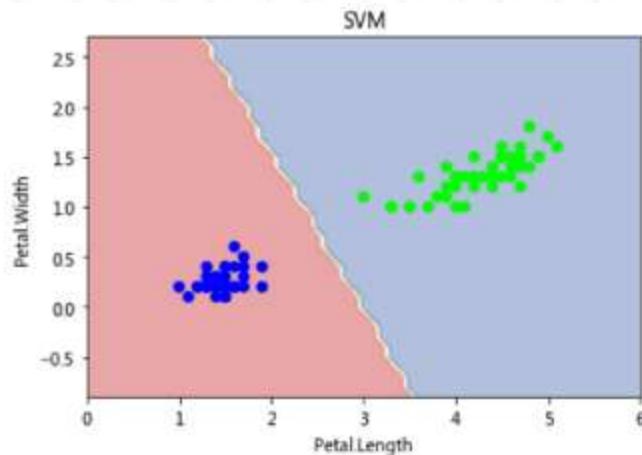
```
Z = estimator.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
plt.plot()
plt.contourf(xx, yy, Z, alpha=0.4, cmap = plt.cm.RdYlBu)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap = plt.cm.brg)
plt.xlabel('Petal.Length')
plt.ylabel('Petal.Width')
plt.show()
```

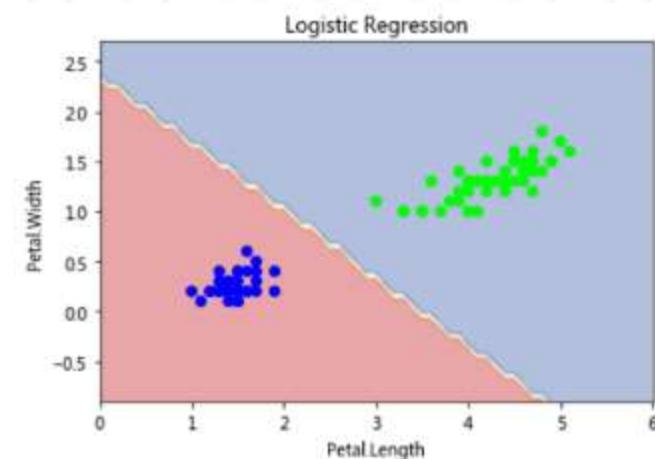
建立一個繪圖函數

SVM v.s. Logistic Regression

`plot_estimator(clf1, X, y)`



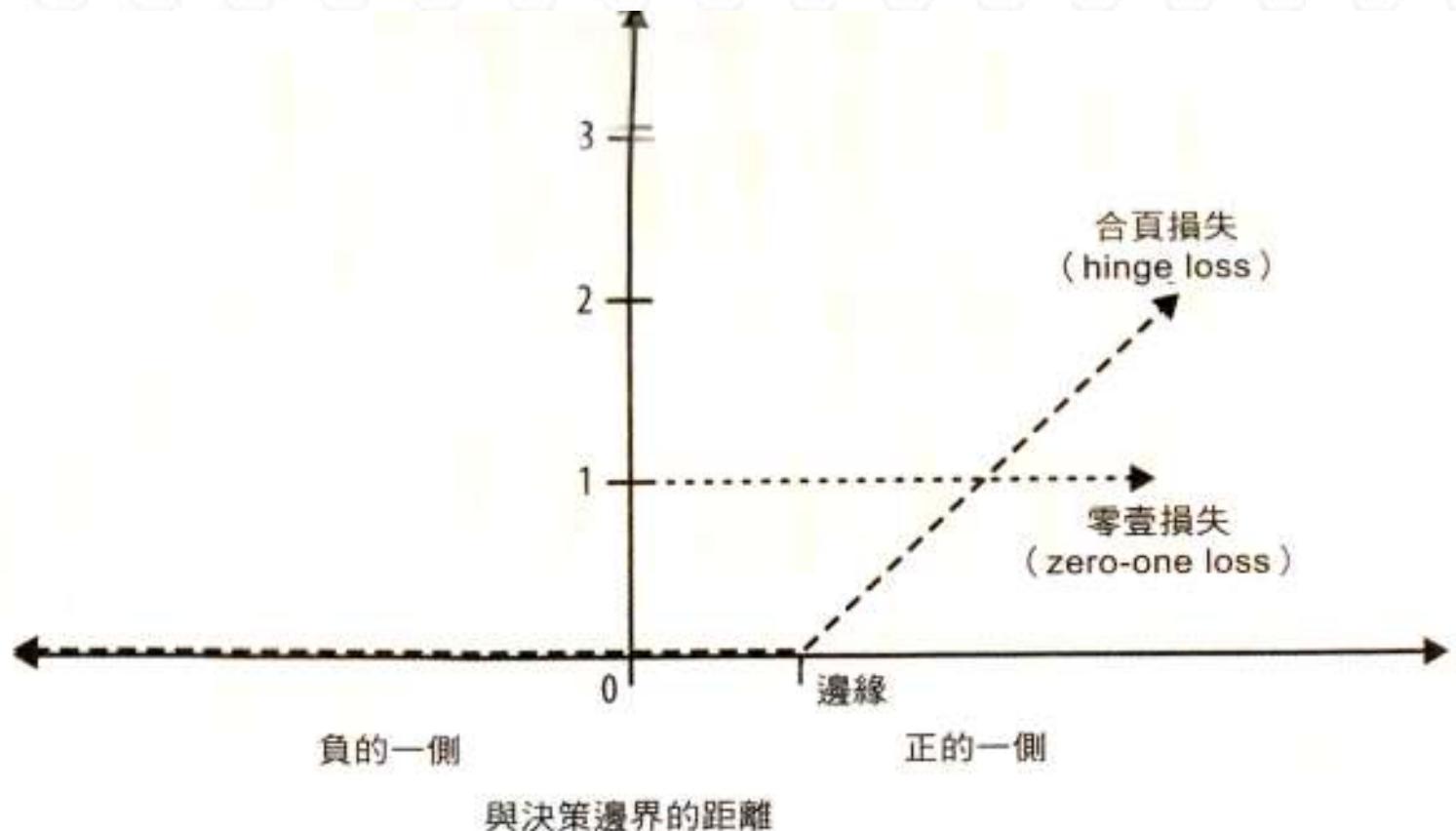
`plot_estimator(clf2, X, y)`



調整Kernel Function 可以做到非線性適配

評估決策邊界

SVM 只會犯小錯

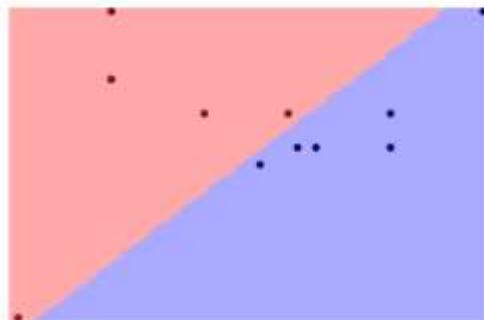
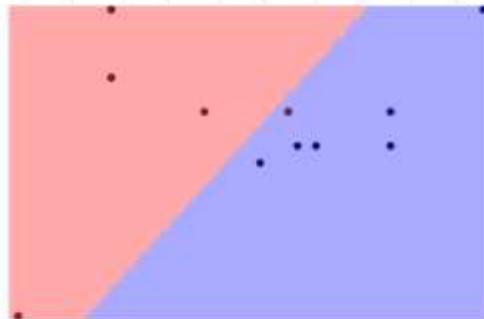


正則項 (Regularization Term)

- 變更正則項 (Regularization Term) C
- 小的 C 可允許界線被忽略 → 寬邊界 (wide margin)
- 大的 C 讓界線難以被忽略 → 窄邊界 (narrow margin)
- $C = \infty$ 必須遵守界線規則，不得越界(hard margin)

設定正則項 (Regularization Term)

```
data = np.array([[-1,2,0],[-2,3,0],[-2,5,0],[-3,-4,0],[-0.1,2,0],[0.2,1,1],[0,1,1],[1,2,1], [1,1,1], [-0.4,0.5,1],[2,5,1]])  
X = data[:, :2]  
Y = data[:,2]  
  
# Large Margin  
clf = SVC(C=1.0, kernel='linear')  
clf.fit(X, Y)  
plot_estimator(clf,X,Y)  
  
# Narrow Margin  
clf = SVC(C=100000, kernel='linear')  
clf.fit(X, Y)  
plot_estimator(clf,X,Y)
```



讀取數據

```
from itertools import product  
import numpy as np  
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris  
from sklearn.svm import SVC
```

```
iris = load_iris()  
X = iris.data[:,[2,3]]  
y = iris.target
```

比較不同Kernel

```
clf1 = SVC(kernel="rbf")  
clf1.fit(X, y)
```

```
clf2 = SVC(kernel="poly")  
clf2.fit(X, y)
```

非線性Kernel

```
clf3 = SVC(kernel="linear")  
clf3.fit(X, y)
```

繪製決策邊界

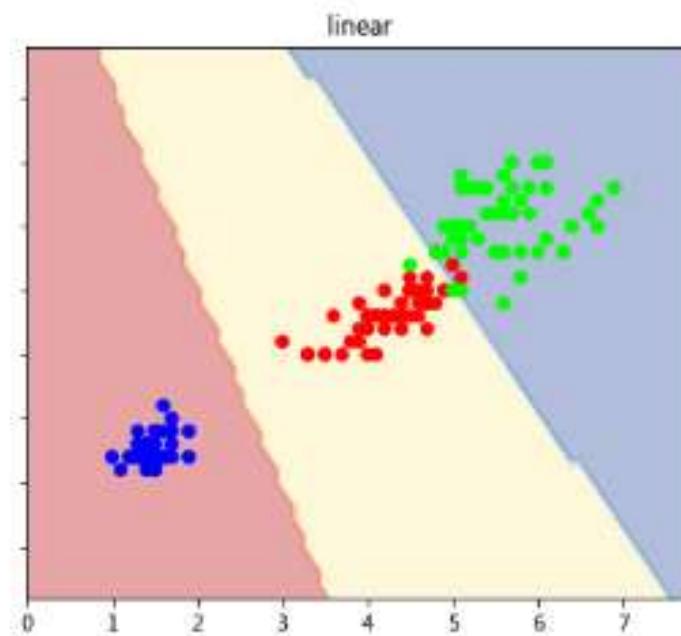
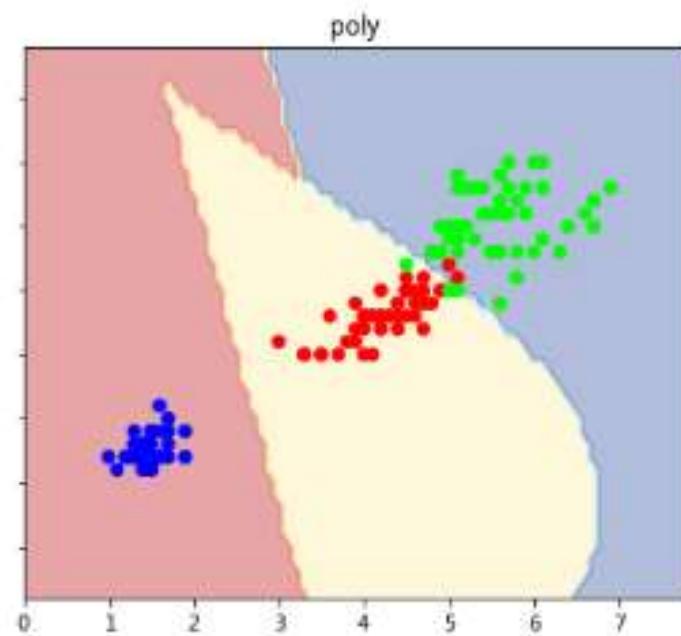
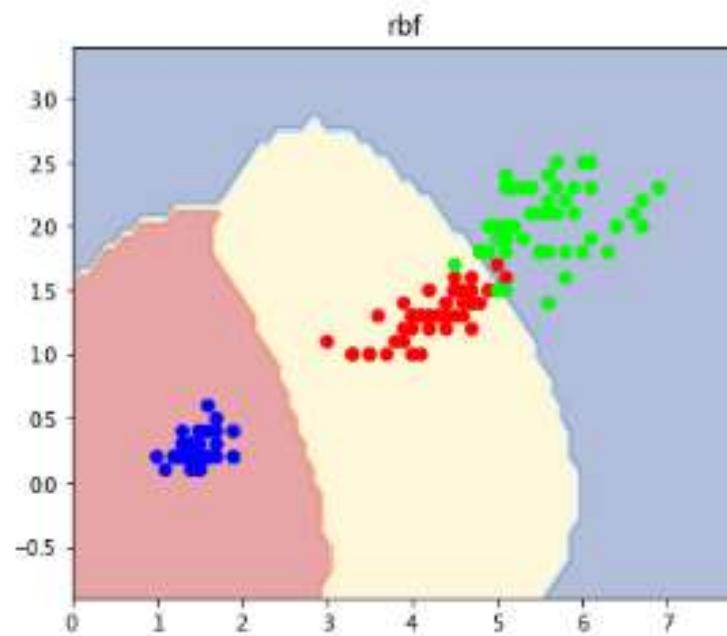
```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                      np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(1, 3, sharex='col', sharey='row', figsize=(20, 5))

for idx, clf, title in zip([0,1,2],[clf1, clf2, clf3], ['rbf', 'poly', 'linear']):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    axarr[idx].contourf(xx, yy, Z, alpha=0.4, cmap = plt.cm.RdYlBu)
    axarr[idx].scatter(X[:, 0], X[:, 1], c=y, cmap = plt.cm.brg)
    axarr[idx].set_title(title)
```

比較不同Kernel Function

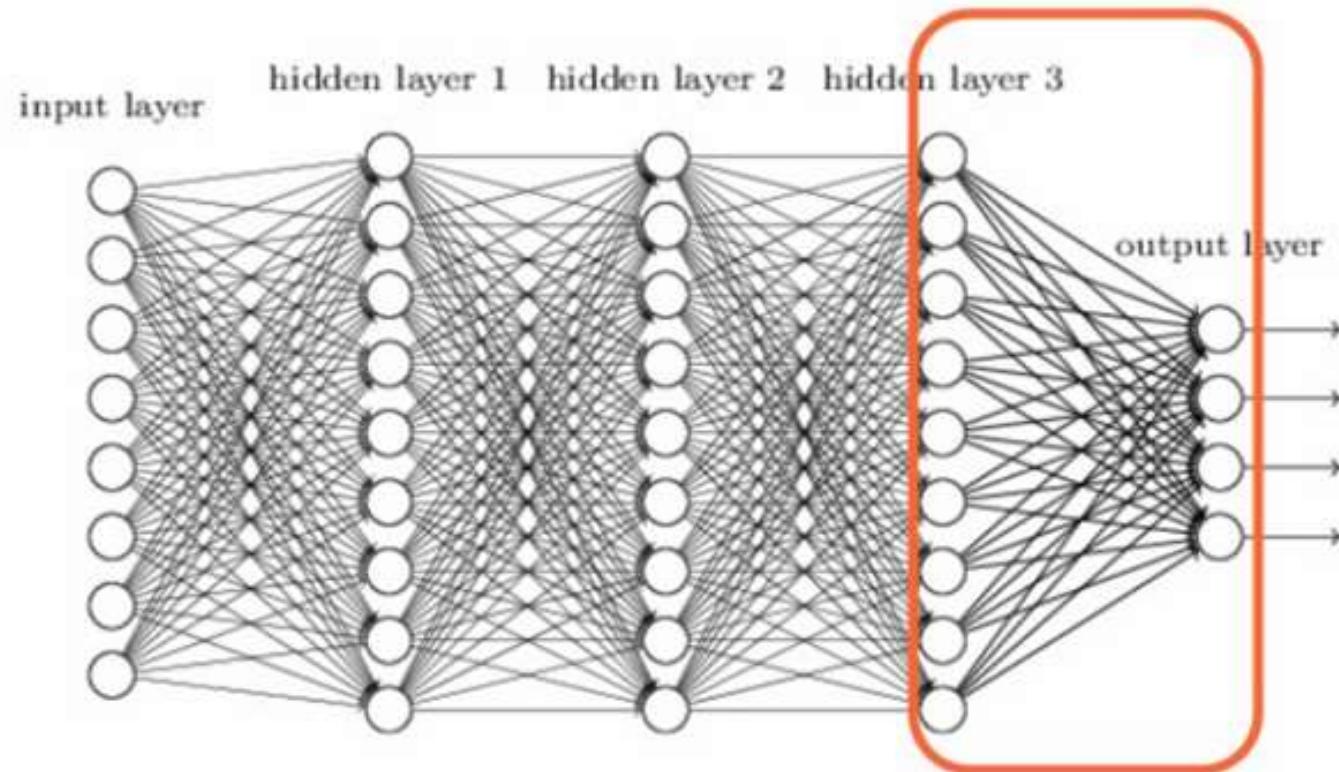


深度學習

類神經網路的曙光

■ 2006 – A fast learning algorithm for deep belief nets

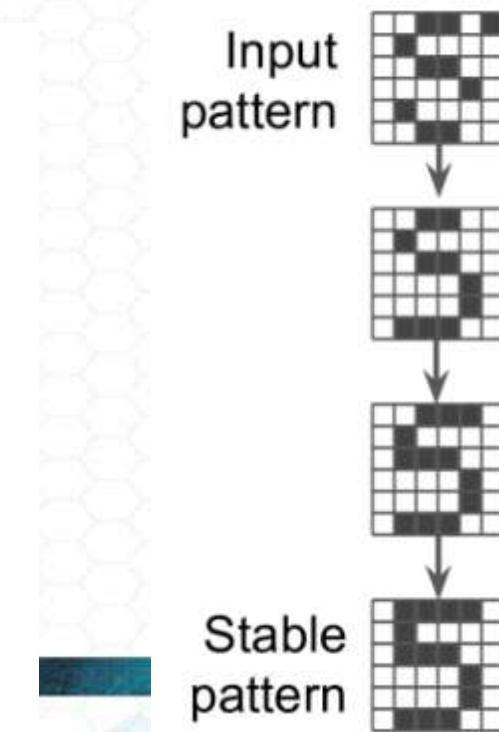
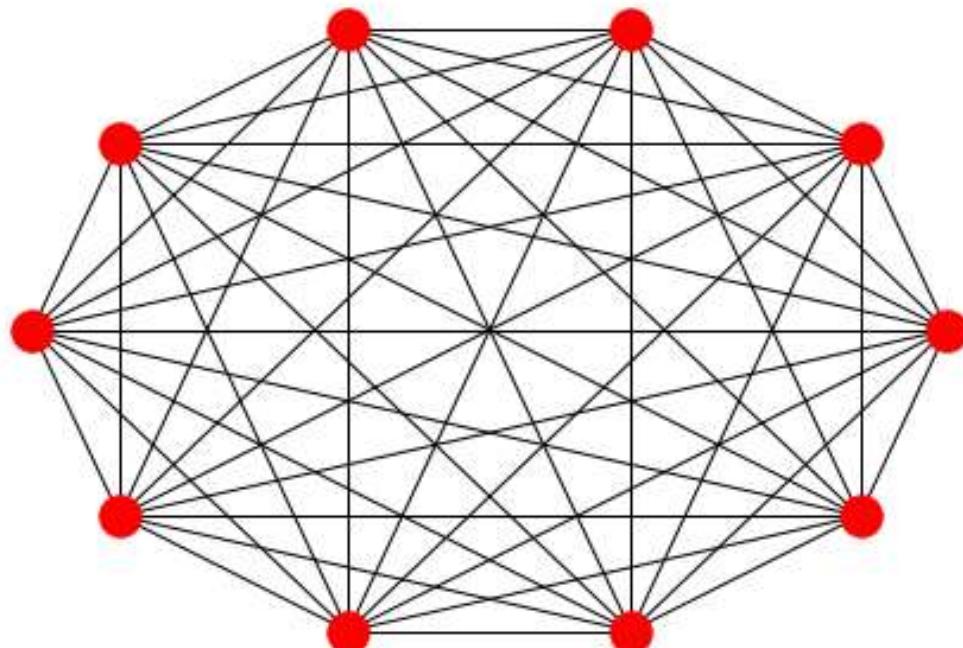
□ Hinton提出用神經網路的非監督式學習來做為神經網路初始權重的指派



非監督式學習

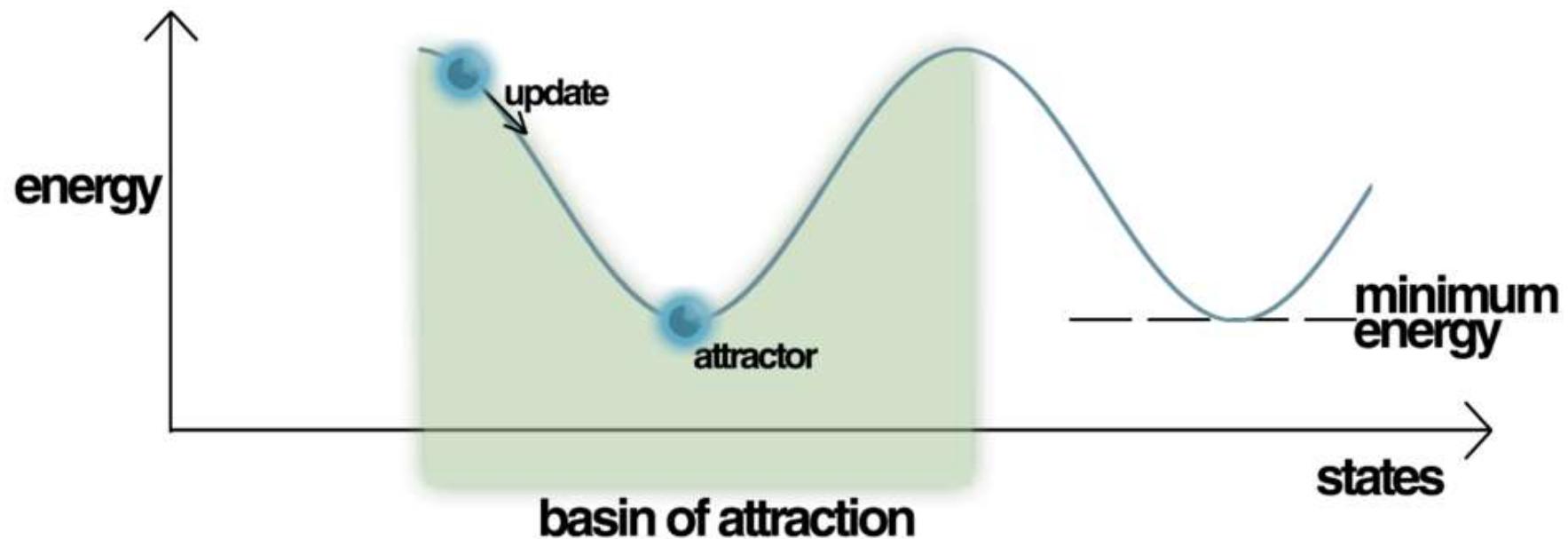
Hopfield 網路

- Hopfield 網路由 J.Hopfield 於 1982年所提出，可以視為一聯想記憶 (Association Memory) 網路
- 教學網路關於數字 5 的圖像後，未來當網路看到一接近數位 5 的模糊圖像，將會將該圖像聯想成數字 5



能量函數

- 每個狀態都會透過一能量函數評估，能量高的狀況發生的次數(或機率)比較少，而狀態維持在低能量狀態的次數(或機率)較多。
- 每經過一個反覆運算，能量即會下降，Hopfield網路最終會保持在一個低能量的穩定狀態

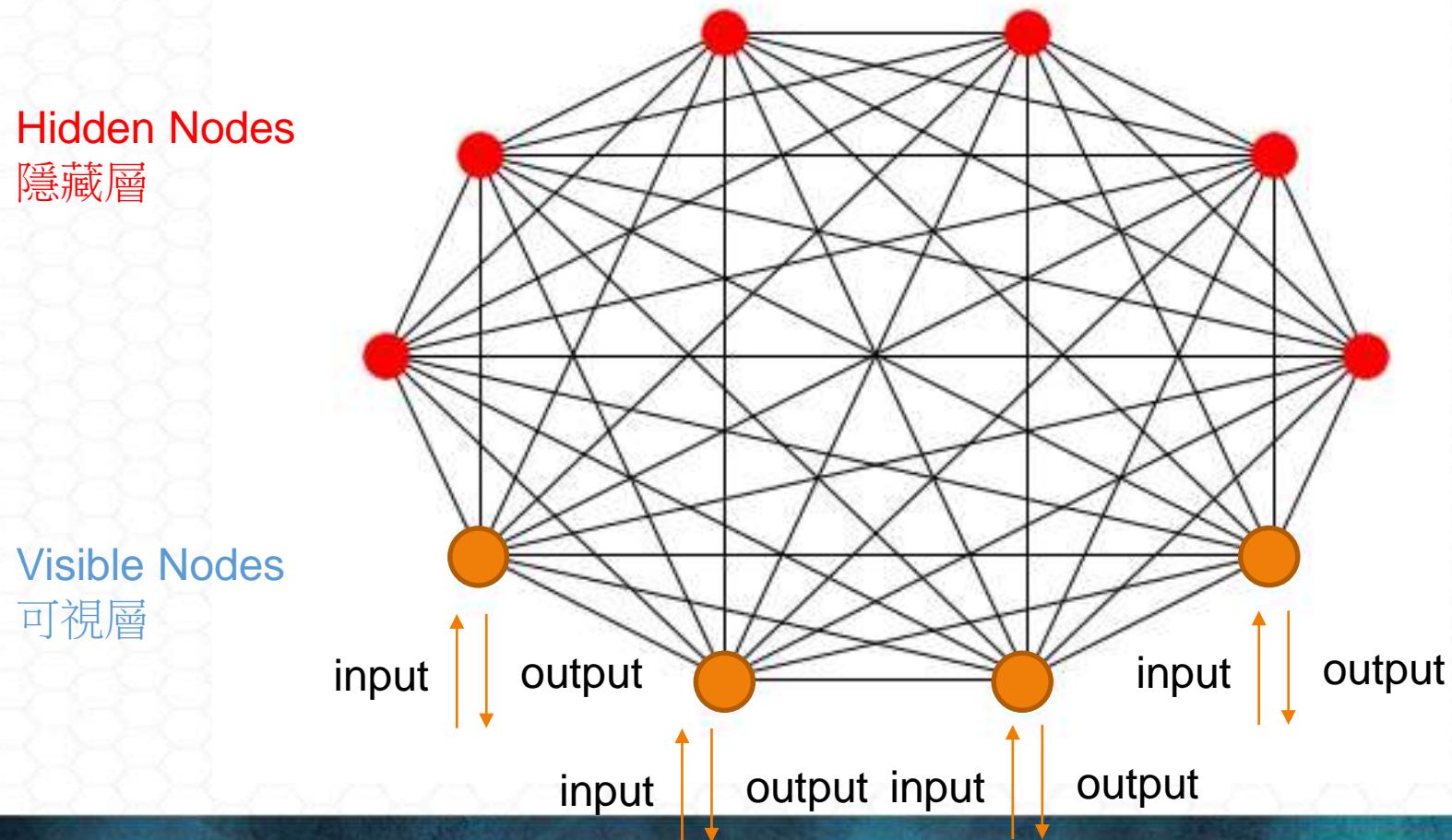


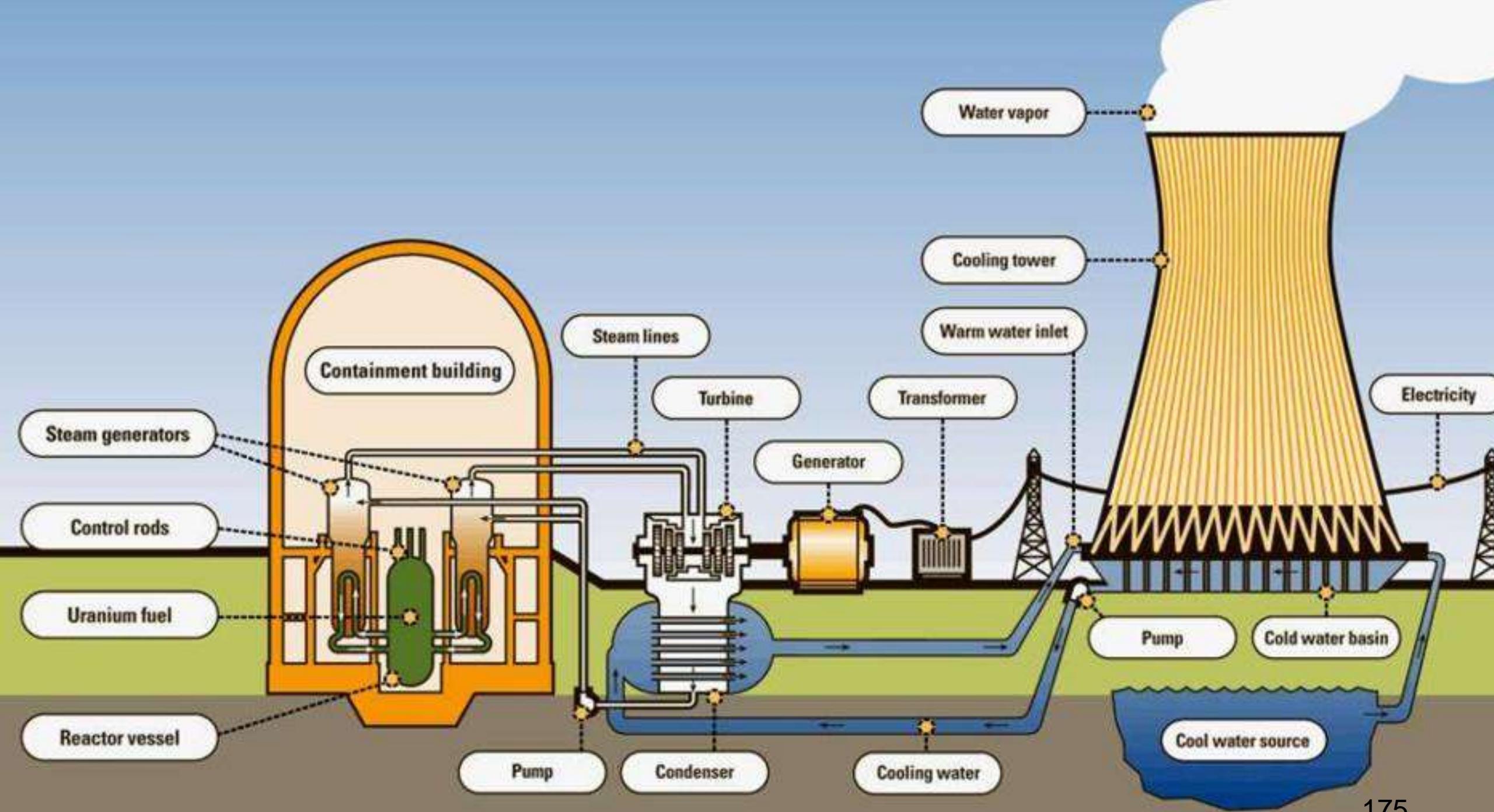
Hopfield 網路的缺陷

- Hopfield 網路保證向局部極小做收斂，但收斂到錯誤的局部極小值（local minimum），而非全域極小（global minimum）的情況也可能發生
- Hopfield 網路計算量過於龐大，如果網路中有784 個節點，將要更新 $306,936(784 * 783 / 2)$ 個權重

玻爾茲曼機 (Boltzmann Machines)

- 玻爾茲曼機(Hinton 1985)類似Hopfield神經網路，但內部的神經元為隨機神經元，代表著每個神經元的產出為機率值而非0或1的二元類別





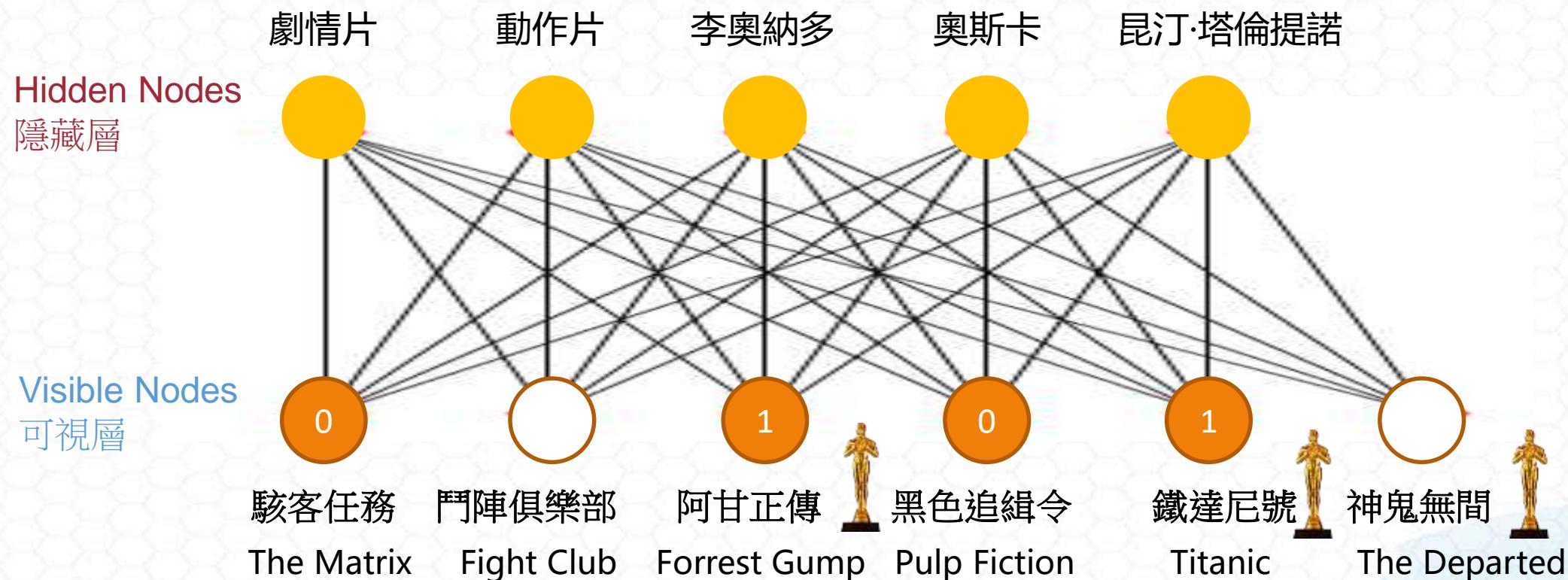
神經元輸出機率

$$P(s_i^{(\text{next step})} = 1) = \sigma\left(\frac{\sum_{j=1}^N w_{i,j} s_j + b_i}{T}\right)$$

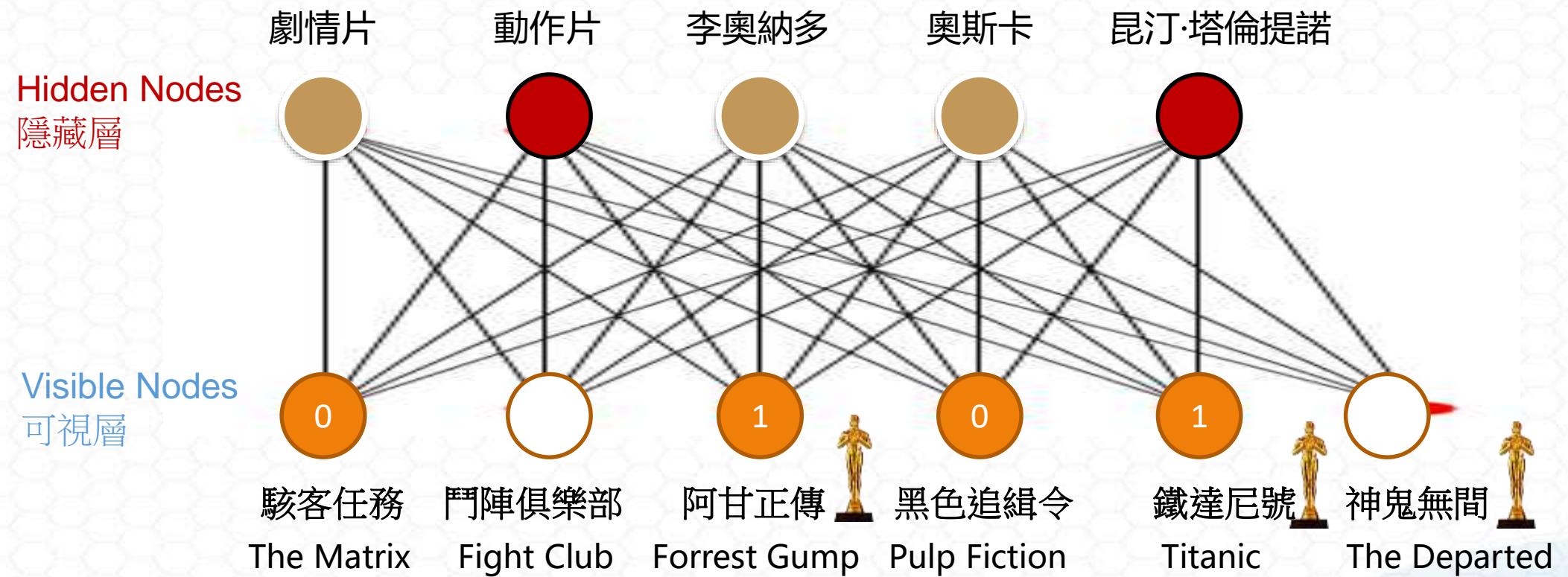
- s_j 是第 j 個神經元的狀態 (0 or 1).
- $w_{i,j}$ 是第 i 與 j 神經元連結的權重.
- b_i 是第 i 神經元的偏倚
- N 是神經元的數量.
- T 代表網路的溫度 溫度越高越可能產生隨機的輸出
- σ 代表邏輯式(logistic) 函數.

限制玻爾茲曼機 (Restricted Boltzmann Machines)

- 同一層的神經元彼此間沒有連結，不同層的神經元彼此間會連接在一起，並採取隨機決策 (stochastic decisions) 來決定一個神經元要傳導或不傳導

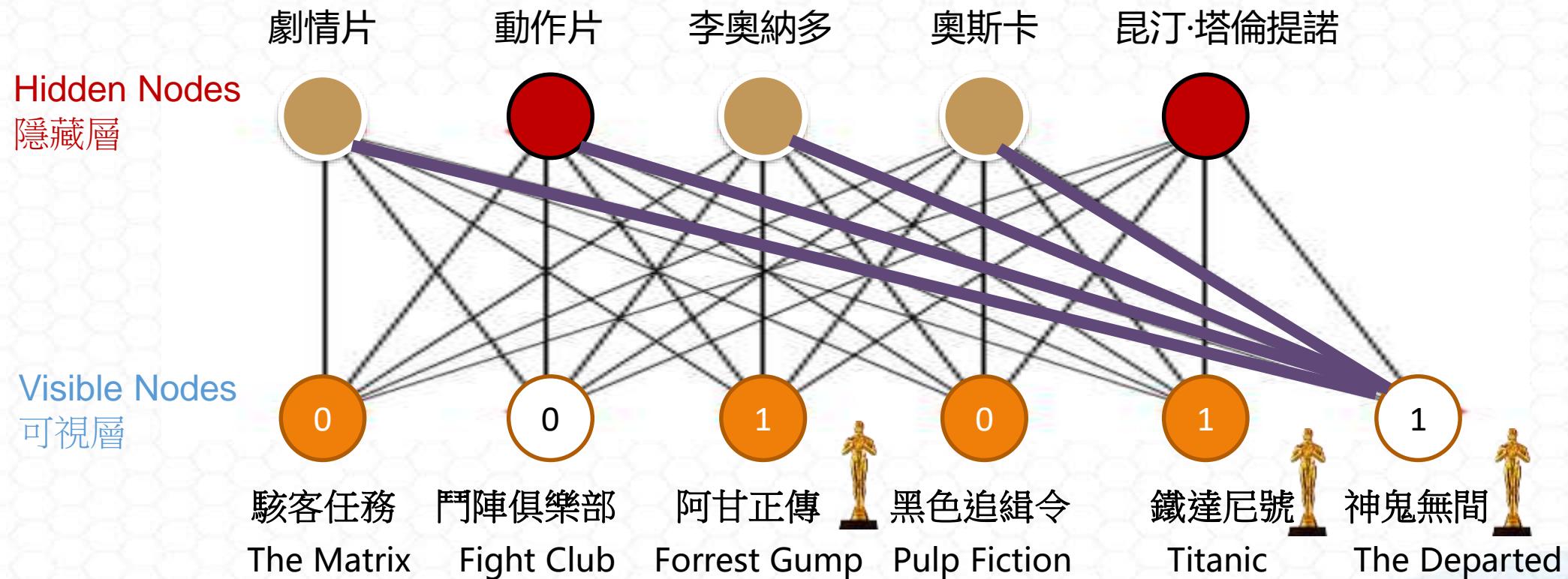


前向傳導 (Forward)



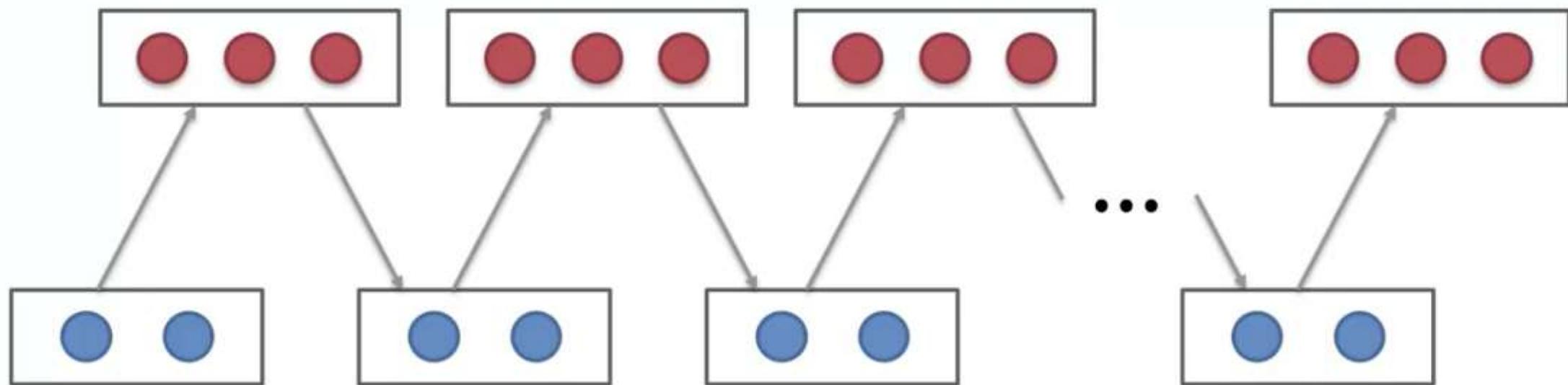
重建 (Reconstruction)

■ 利用Hidden Nodes 推導 Visible Nodes 可能結果



Contrastive Divergence

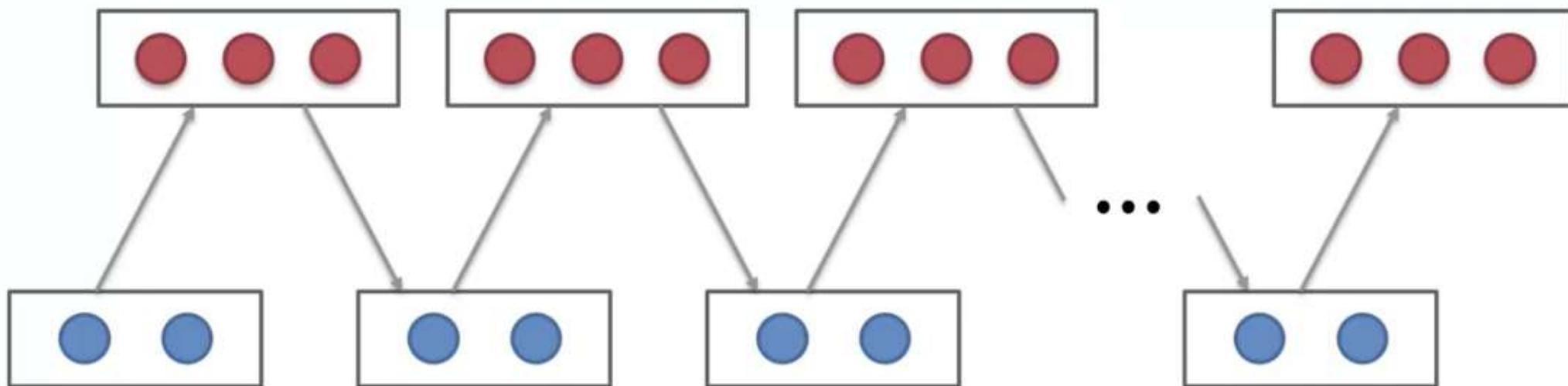
- 整個網路會不斷向前傳遞過去、再向後傳導將原始資料重建回來，期間來回數次、不斷調整每個神經元的權重和偏差值，直到「原始資料」和「重建後資料值」兩者差異被優化到最小值為止



Contrastive Divergence

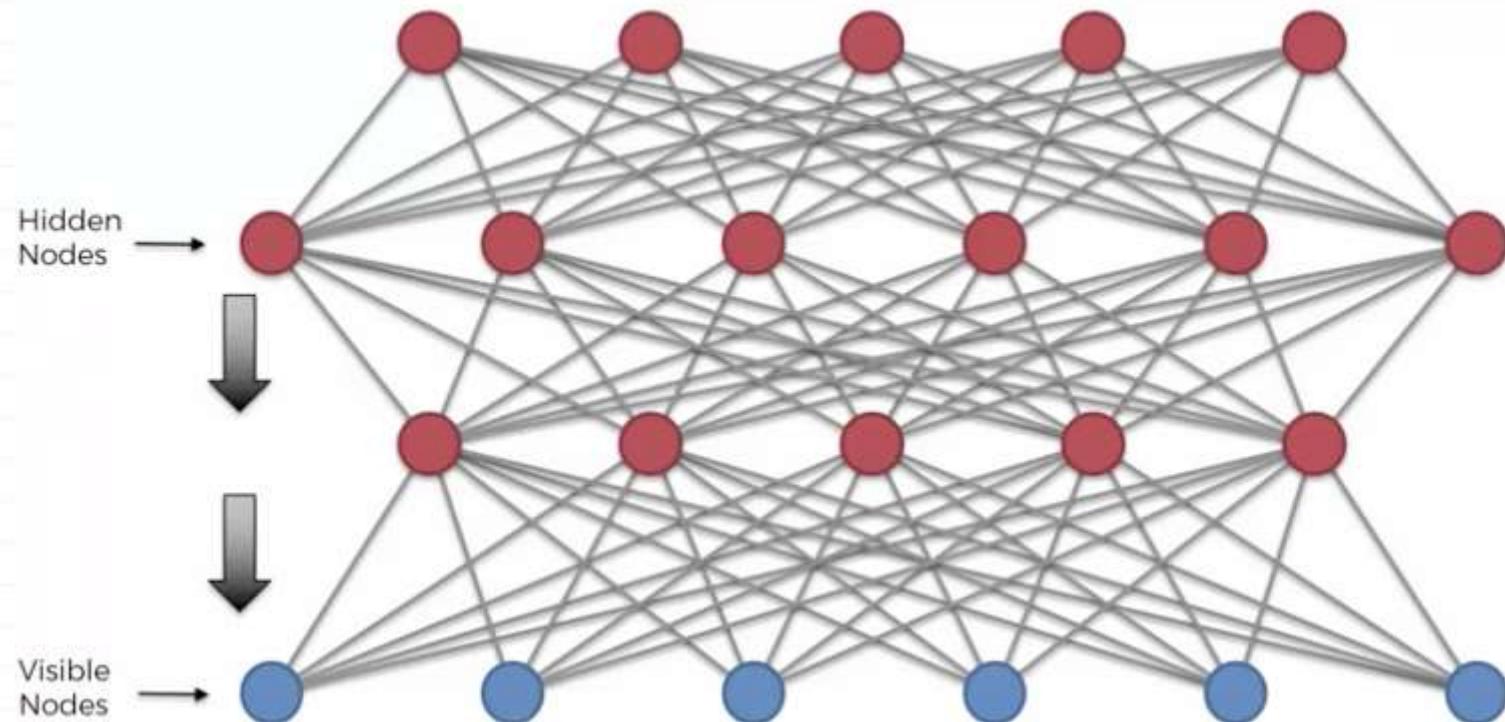
$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta (\mathbf{x}\mathbf{h}^T - \dot{\mathbf{x}}\dot{\mathbf{h}}^T)$$

- \mathbf{x} 代表一開始在可見層的輸入
- \mathbf{h} 代表向前傳導後的隱藏層的向量
- \mathbf{x}' 代表重建後的可見層向量
- \mathbf{h}' 代表再次向前傳導後所得到的隱藏層向量
- η 代表學習率



Deep Belief Networks

- 深度信念網路會一次訓練 2 層RBM；將數個 RBM 模型堆疊起來，每一層模型的輸出值即為下一層模型的輸入值，直到抵達最後一層輸出層為止

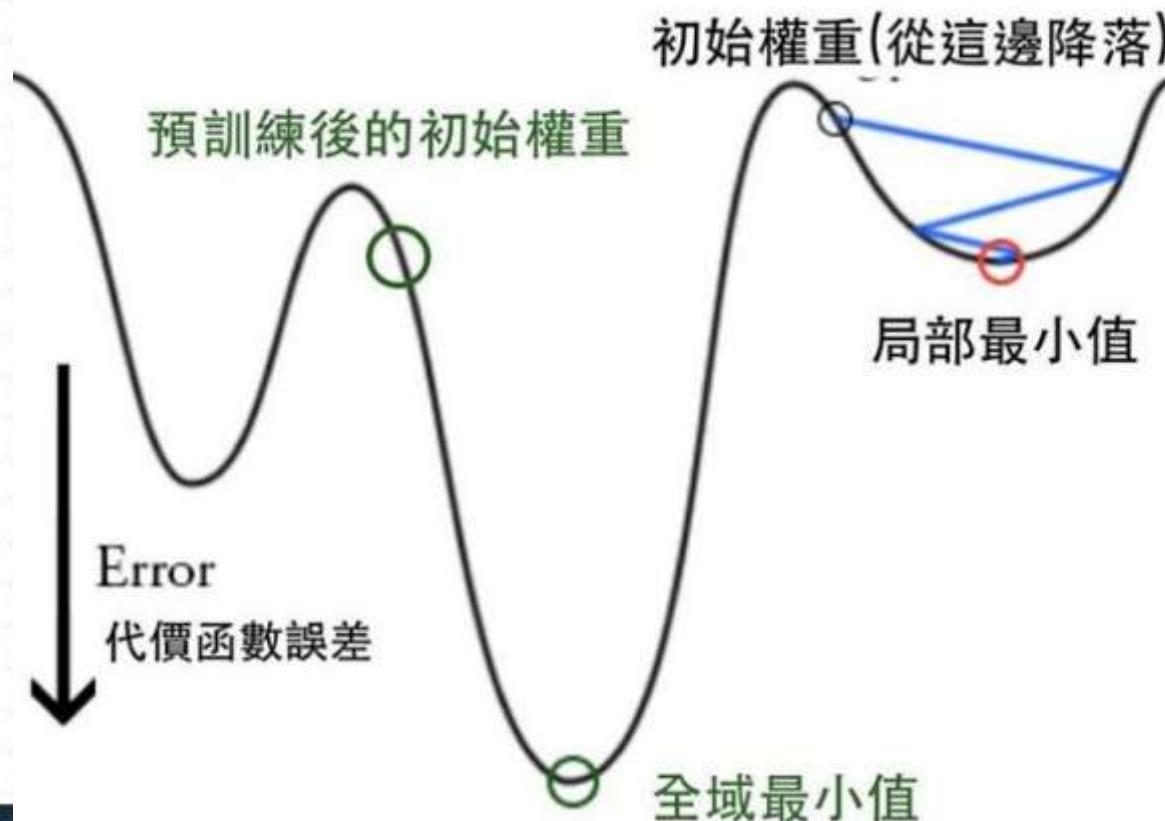


Deep Belief Networks

- 深度信念網路將無標籤 (unlabeled data) 資料放入網路中；是一個無監督學習過程
- 結合半監督學習 - 透過少量的標籤資料 (labeled data)，對深度信念網路的神經元權重和偏差值進行微調，讓精確度更高
- 預訓練(Pre-trained)不直接將資料放進分類器中，而是將資料預先經過 RBM 模型的訓練。深度信念網路能找到輸入資料隱含的規律、具備優異的特徵學習能力

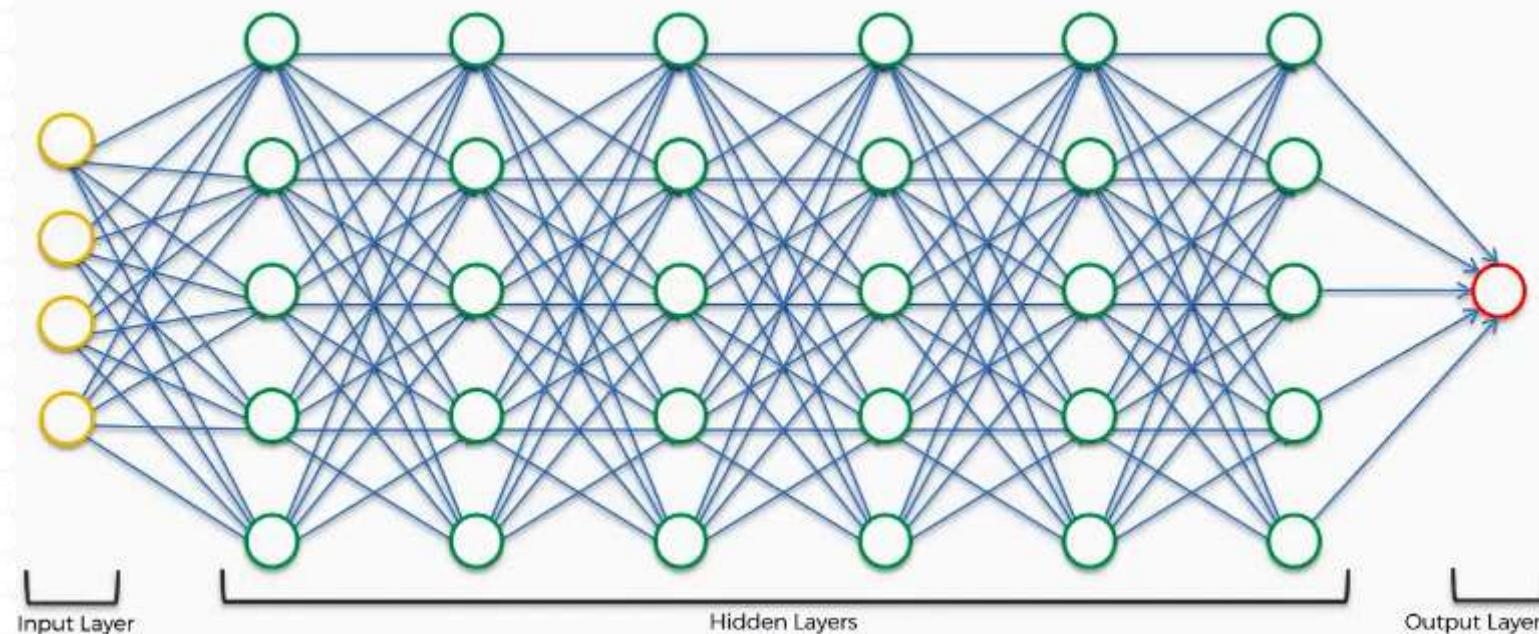
解決反向傳播的優化問題

- 傳統的神經網路隨機初始化網路中的權值，導致網路很容易收斂到局部最小值
找到一個理想的起始點開始梯度下降，將能夠更快、更容易找到全域最小值。



深度學習網路

- 深度學習網路突破以往只能用兩層網路的限制，但由於 Neural Network 長久以來太過惡名昭彰，Hinton 決定把多層的神經網路 (Deep Neural Network) 重命名為深度學習 (Deep Learning)



ImageNet



14,197,122 images, 21841 synsets indexed

[Explore](#) [Download](#) [Challenges](#) [Publications](#) [CoolStuff](#) [About](#)

Not logged in. [Login](#) | [Signup](#)

ImageNet is an image database organized according to the [WordNet](#) hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

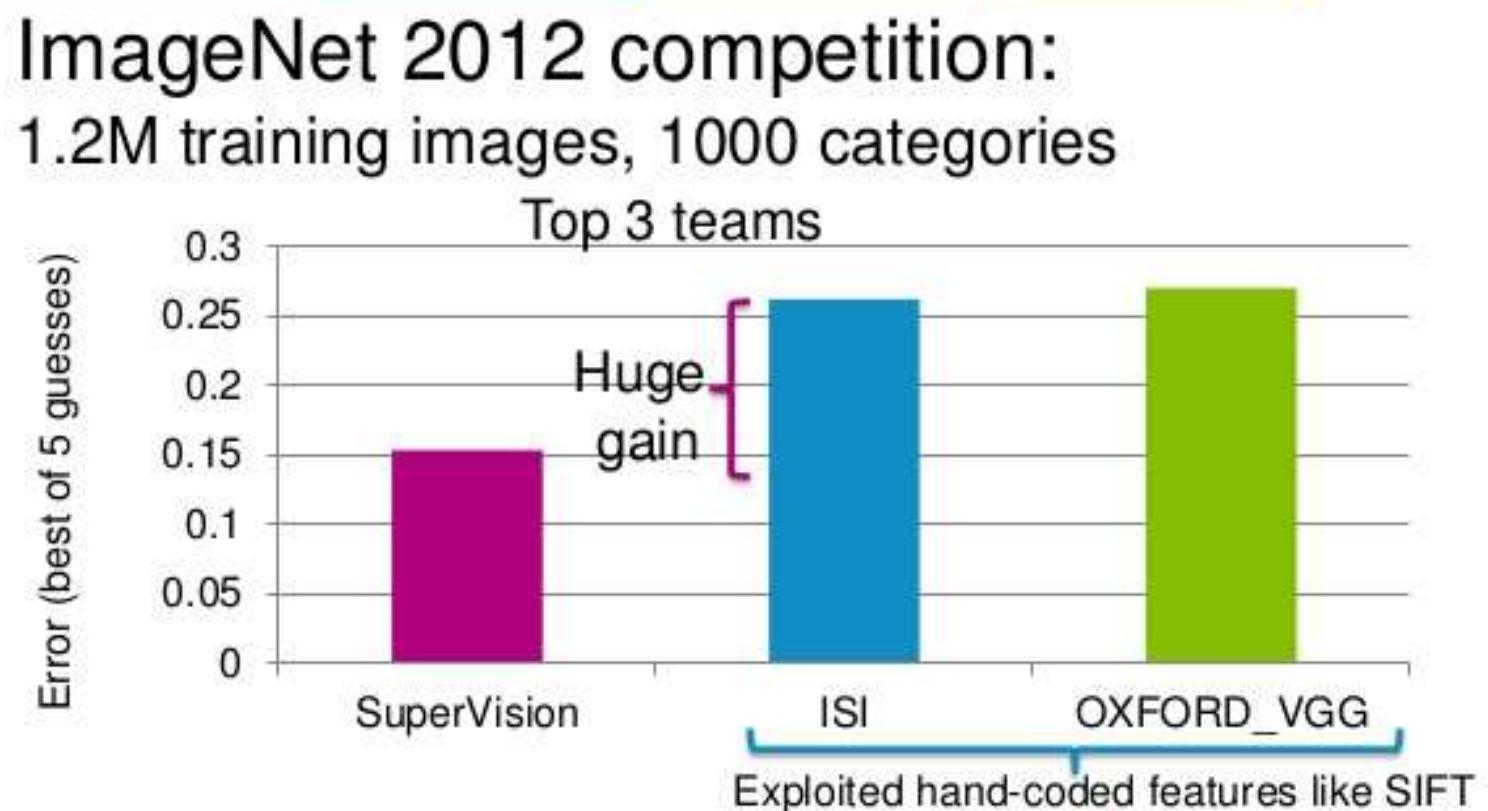
[Click here](#) to learn more about ImageNet, [Click here](#) to join the ImageNet mailing list.



ImageNet 每年都會舉辦圖形識別大賽

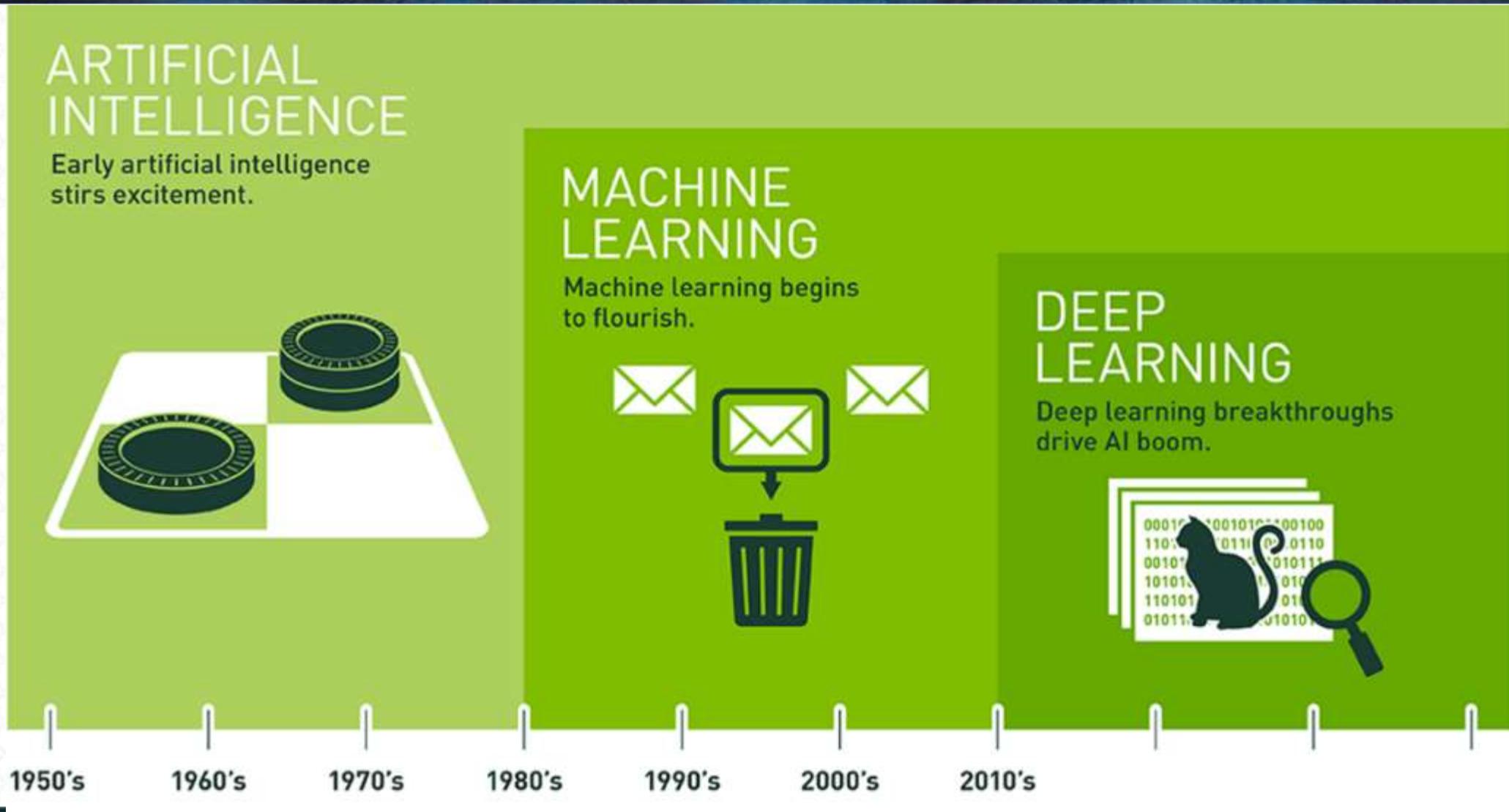
深度學習在ImageNet一戰成名

- Hinton 的兩個學生以 SuperVision 的隊伍名參賽，以 16.42% 的錯誤率遠勝第二名的 26.22%

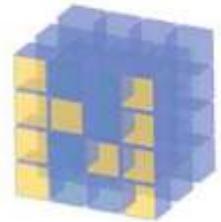


- 深度學習會大量用到矩陣運算，最合適的硬體是負責圖形處理的 GPU
- 直到 NVIDIA 在 2006 – 2007 年間推出全新運算架構 CUDA —讓 GPU 成為深度學習運算必用硬體的關鍵。
- 2012 年 Hinton 的兩位學生利用「深度學習 + GPU」的組合，才真正展現深度學習的威力。

人工智慧 V.S. 深度學習 V.S. 機器學習



Python 與類神經網路



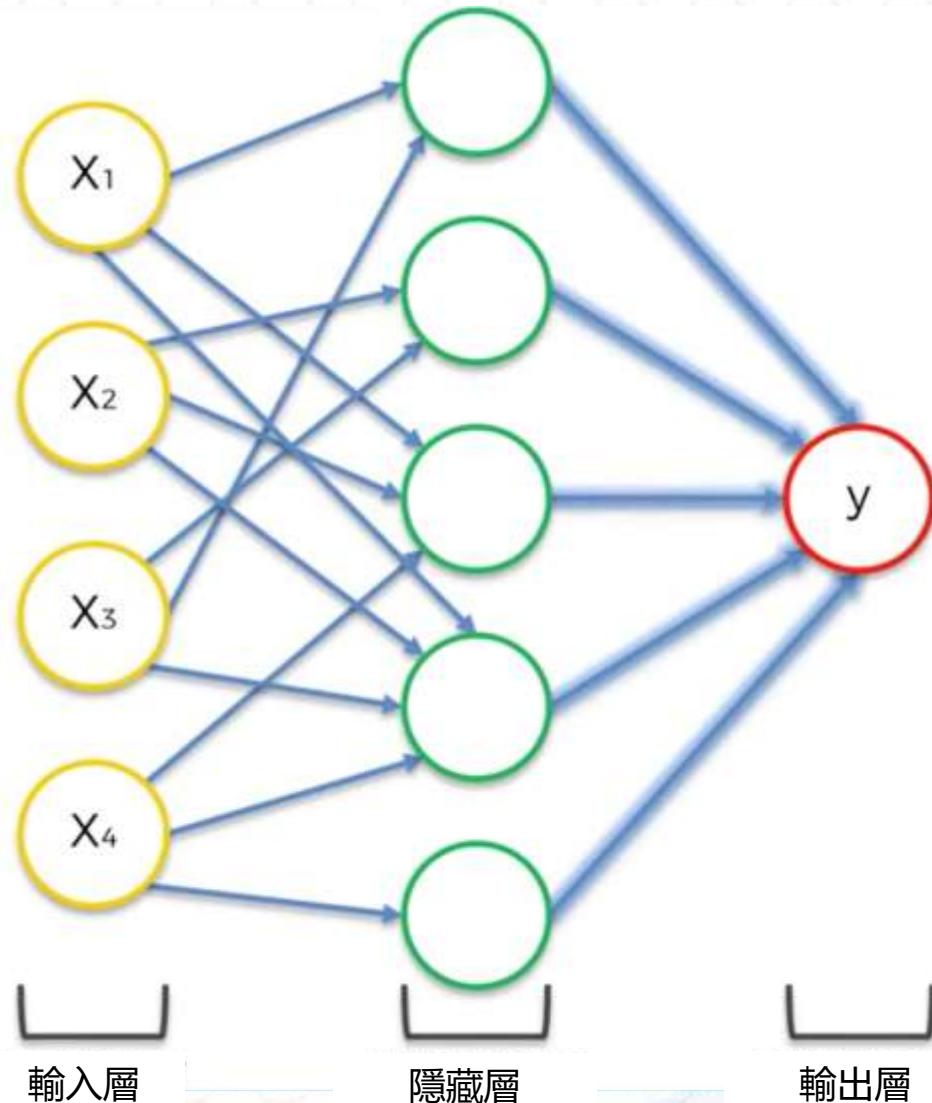
NumPy



Machine Learning with Scikit-Learn



TensorFlow



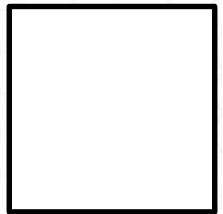
TensorFlow

- Google Brain Team 開源的TensorFlow是最受歡迎的深度學習框架之一，利用開源方式來獲得社群共用的力量，加速深度學習的進展
- TensorFlow 可以運行在 CPU 、GPU 以及TPU上



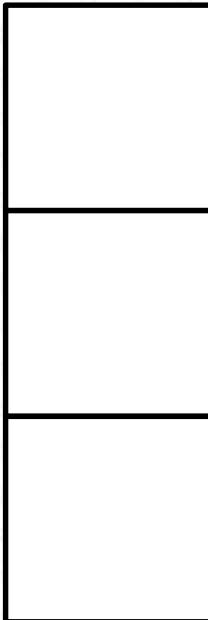
- TensorFlow是利用資料流程圖(Data Flow Graphs)來表達數值運算的開放式原始碼函式庫。資料流程圖中的節點(Nodes)被用來表示數學運算，而邊(Edges)則用來表示在節點之間互相聯繫的多維資料陣列，即張量(Tensors)
- TensorFlow
 - Tensor：張量，其實就是一個n維度的陣列或清單。如一維 Tensor 就是向量，二維 Tensor 就是矩陣
 - Flow：是指 Graph 運算過程中的資料流程

Tensor



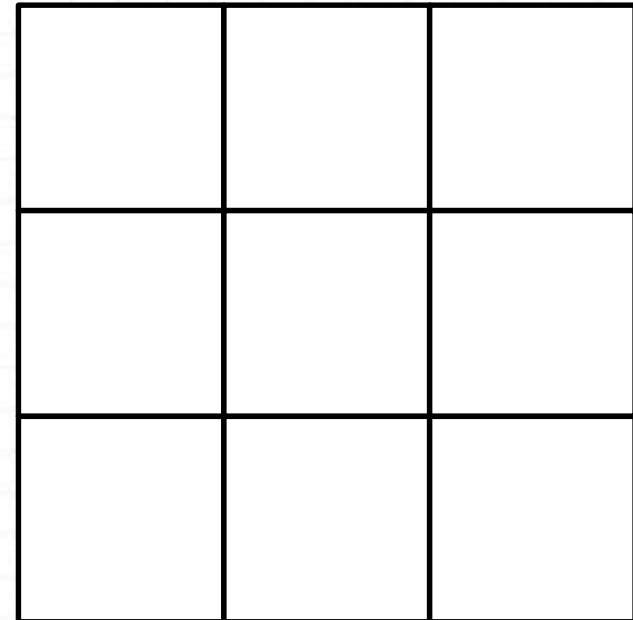
Scalar

Tensor
Rank 0



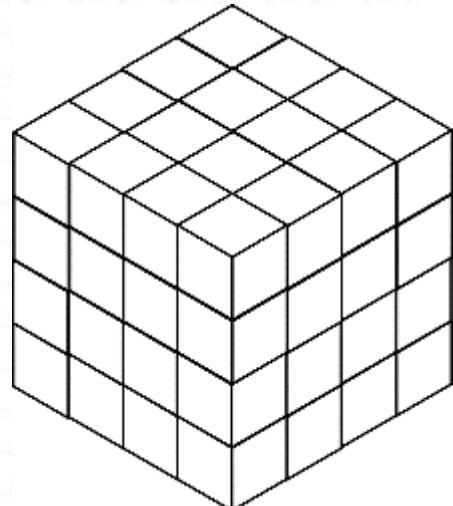
Vector

Tensor
Rank1



Matrix

Tensor
Rank2



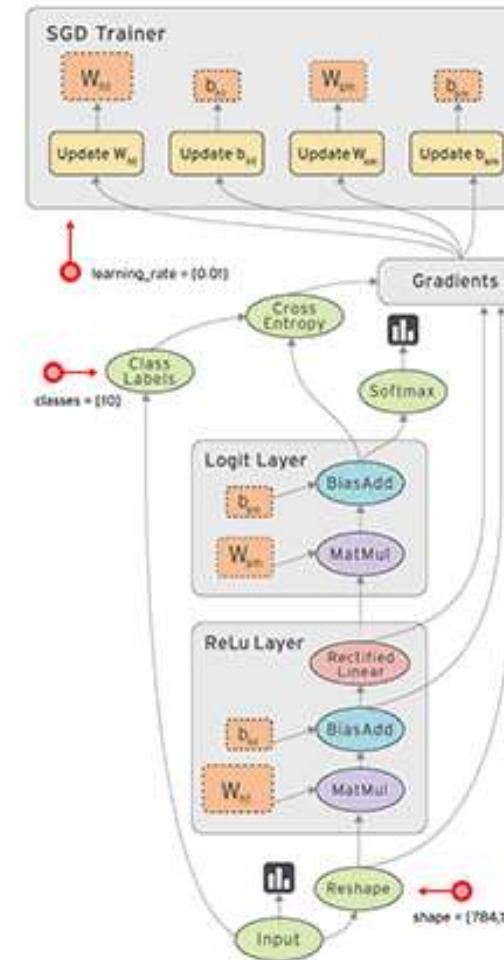
Tensor

資料流程圖 (Data Flow Graphs)

資料流程圖(Data Flow Graphs)是一種有向圖的節點(Node)與邊(Edge)來描述計算過程

節點表示數學操作，亦表示資料 I/O 端點

邊則表示節點之間的關係，用來傳遞操作之間互相使用的多維陣列(Tensors)



安裝 TENSORFLOW

安裝TensorFlow

■ GPU 版本 (非必要)

```
pip install --upgrade tensorflow-gpu
```

■ CPU 版本

```
pip install --upgrade tensorflow
```

可參考 <https://www.tensorflow.org/install/>

第一個TensorFlow 實例

```
import tensorflow as tf  
hello = tf.constant('Hello World')  
sess = tf.Session()  
sess.run(hello)
```

TENSORFLOW 基礎

Tensor 型態

■ Constant (常數)

■ Variable (變數)

□ 可以訓練的變數，比如模型的權重（weights）或者偏倚（bias）

■ Placeholder (預留位置)

□ Tensorflow中是先建好圖再決定資料的輸入與輸出，Placeholder在還沒有資料的時候先占個位，以得到之後傳遞進來的輸入值

運算圖(Computational Graph)

```
x = tf.constant(2)  
y = tf.constant(3)  
with tf.Session() as sess:  
    print(sess.run(x+y))
```



先建立圖，再進行運算

深度學習框架

theano

dmlc

mxnet



Caffe



DL4J
DEELEARNING4J

Lasagne

DSSTNE

Microsoft
CNTK



Keras

Keras是一個由Python編寫而成高階類神經網路API，可接合Tensorflow、Theano以及CNTK等深度學習框架後端

Keras 特性

- 簡易、快速設計模型原型（Keras具模組化，極簡，和可擴充性）
- 支持CNN和RNN，或二者的結合
- 可無縫切換CPU和GPU版本

Keras 安裝

■ Keras 安裝

```
pip install --upgrade keras
```

使用類神經網路建立客戶流失預測模型

客戶流失分析

- 失去一個老客戶會帶來巨大損失，需要企業再開發十個新客戶才能予以彌補，如何能預測客戶即將流失，讓企業採取必要留客手段，是各行業皆須關注的問題
- 目標：
利用類神經網路建構客戶流失分析模型，以預測客戶是否有流失可能

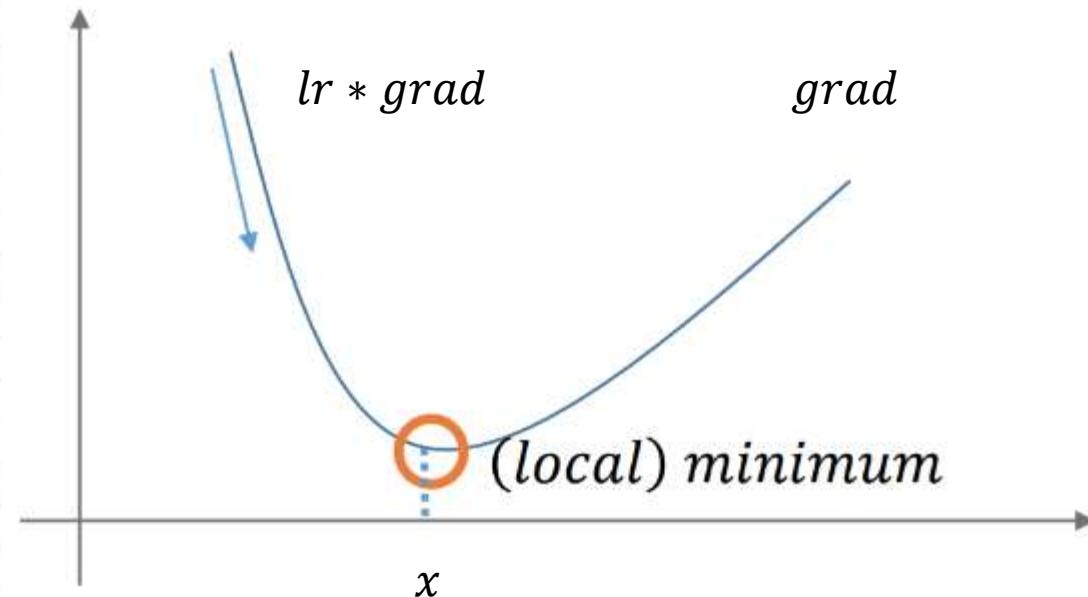
如何選擇優化器

隨機梯度下降 (Stochastic Gradient Descent)

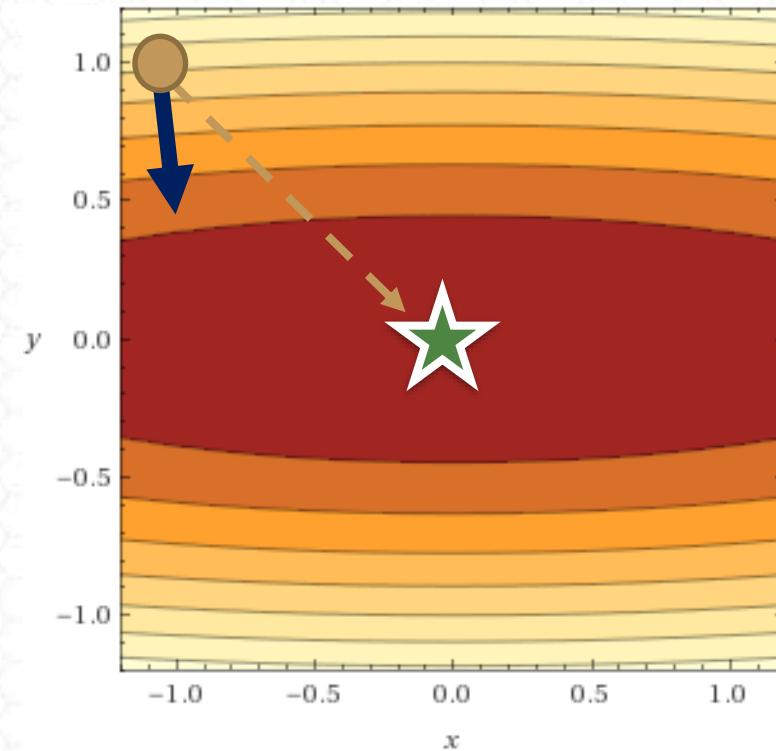
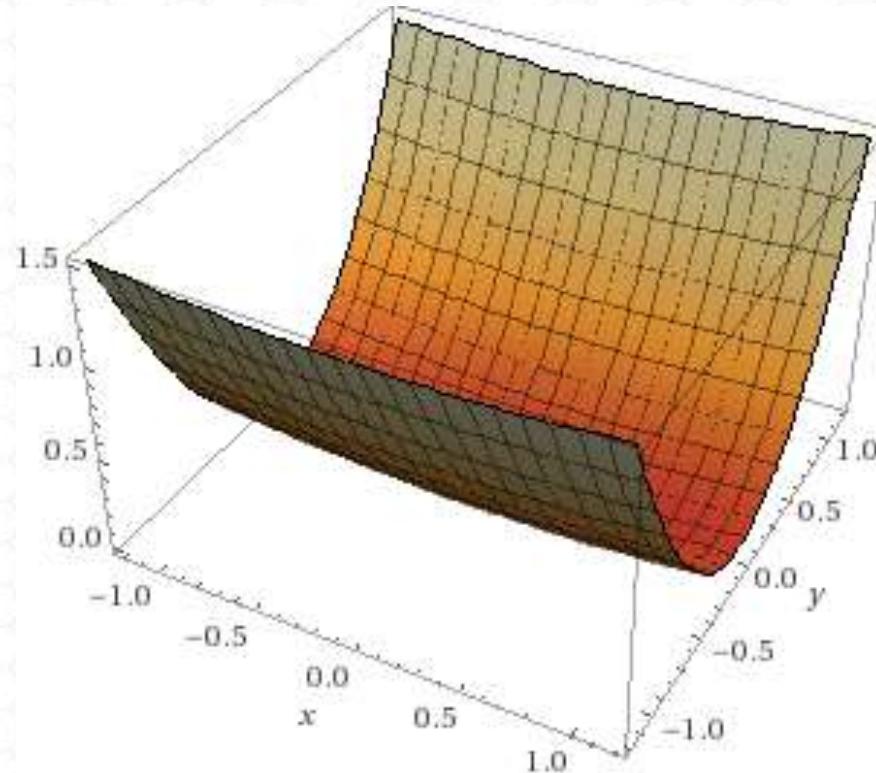
- Gradient Descent Method
 - 找出代價函數的最小值

權重更新函數

$$x_i = x_i - lr * grad$$



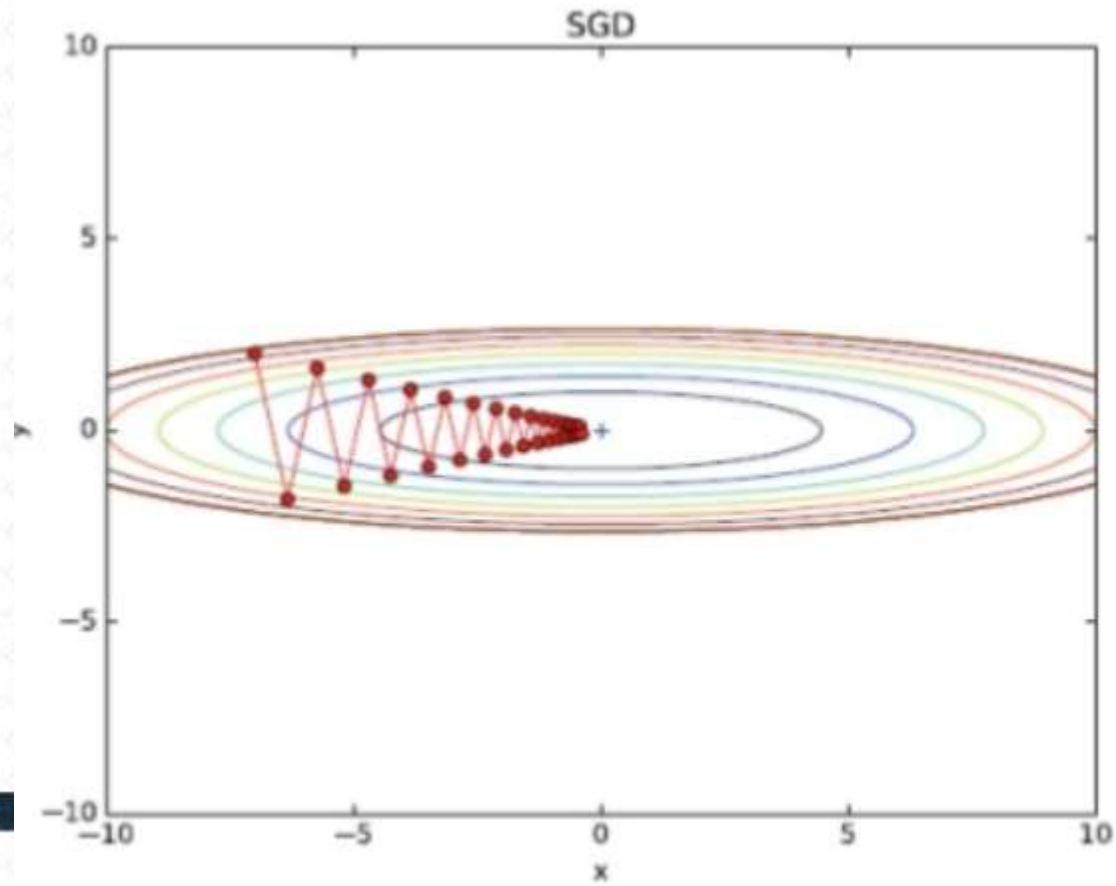
探索最低點





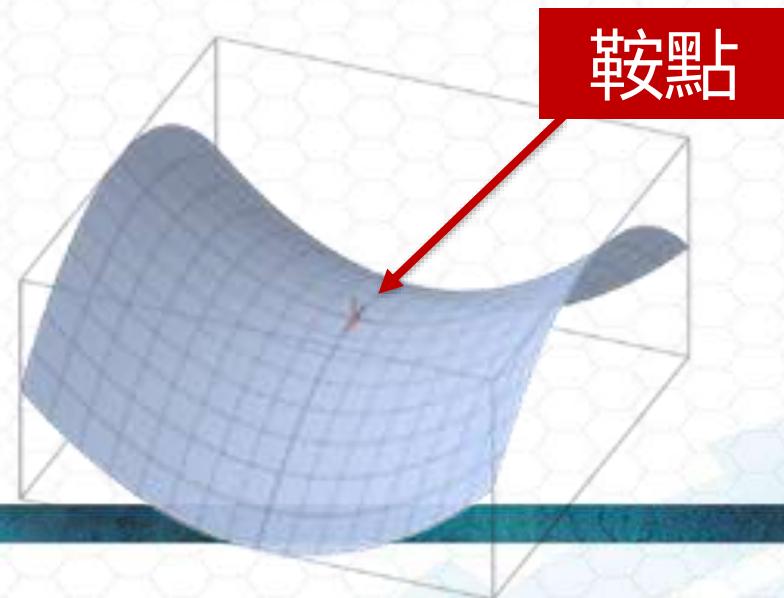
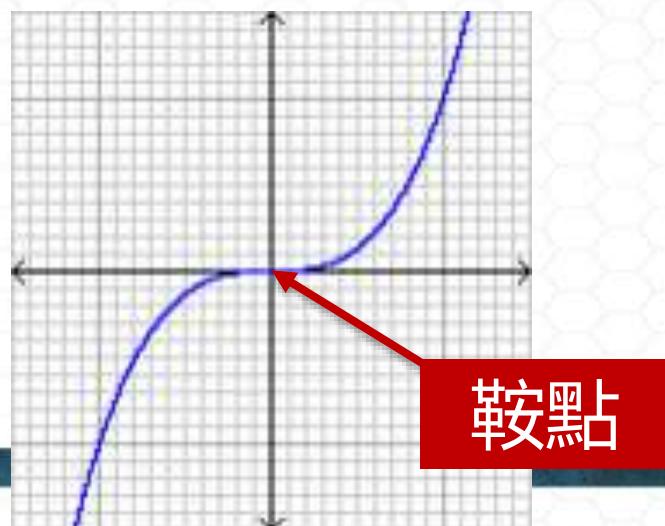
隨機梯度下降的缺點

當梯度不指往最小值時，SGD 會以鋸齒狀方式邁向最低點，相當沒效率



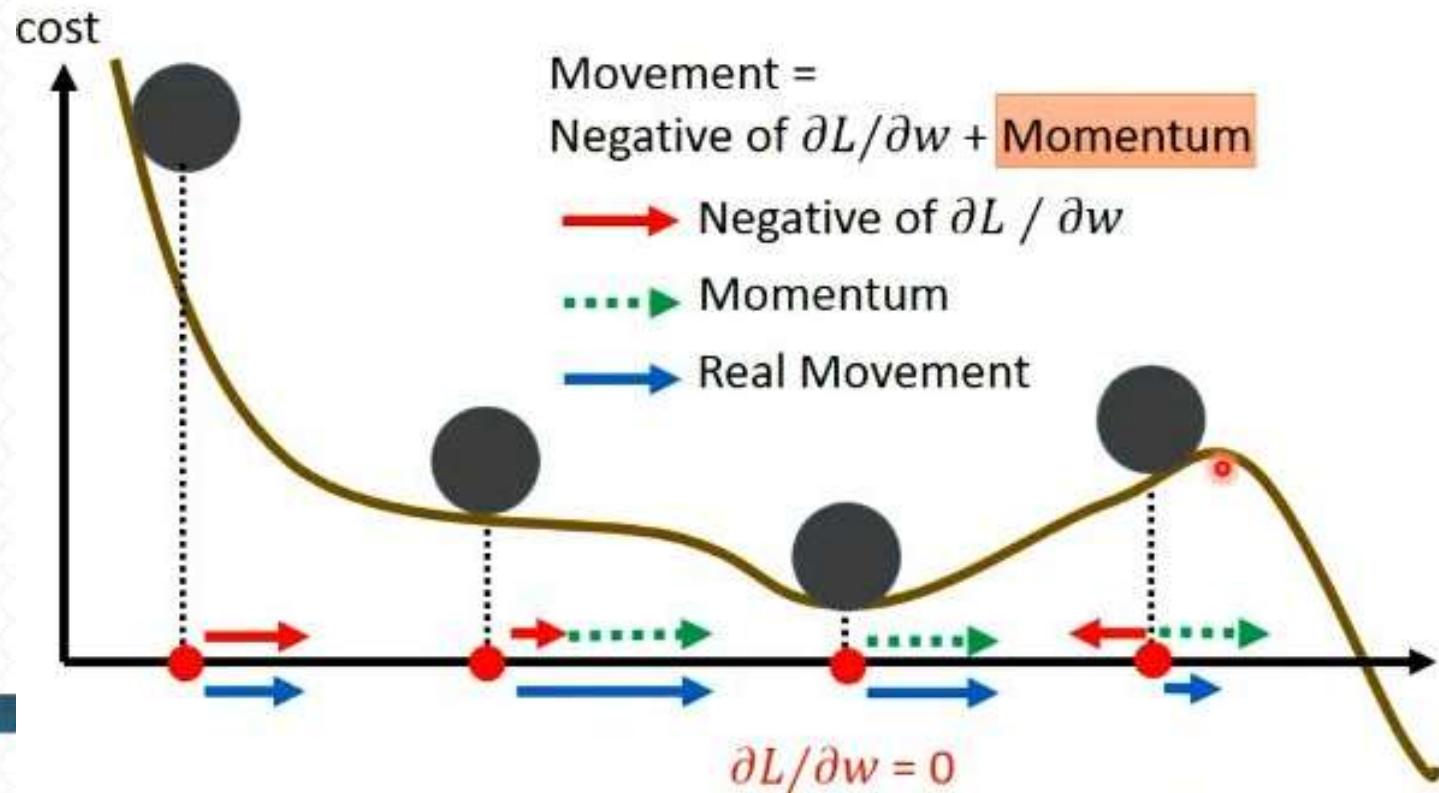
隨機梯度下降的缺點

- 1. 很難選擇出合適的學習率, 太大太小都不合適
- 2. 相同的學習率不應該適用於所有的參數更新 (e.g. 很少出現的特徵, 應使用較大的更新率)
- 3. 隨機梯度下降容易被鞍點困住



動量 (Momentum)

■ 模擬物理學中的動量(Momentum) · 使用積累的動量替代梯度



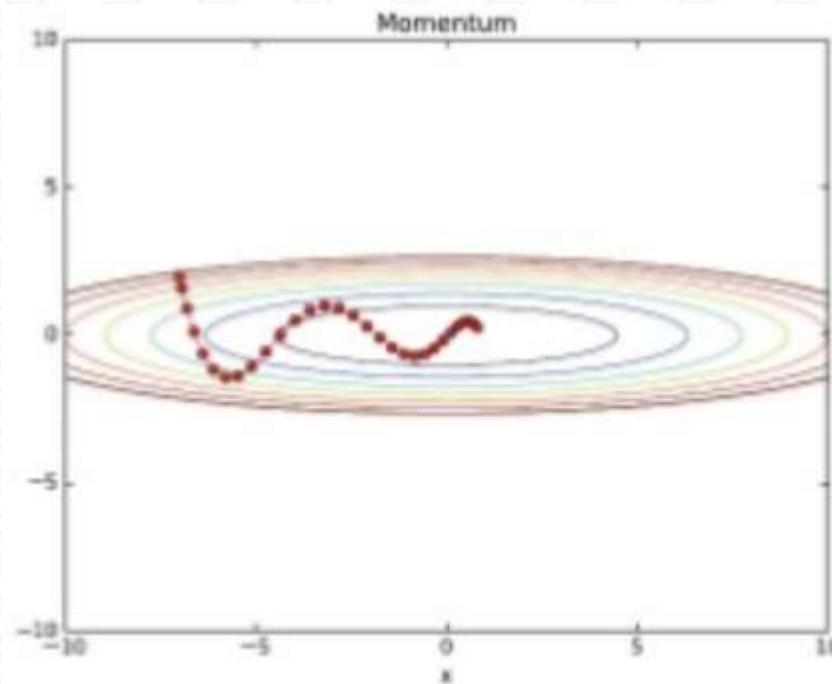
動量 (Momentum)

權重更新函數

$$v = mv - lr * grad$$

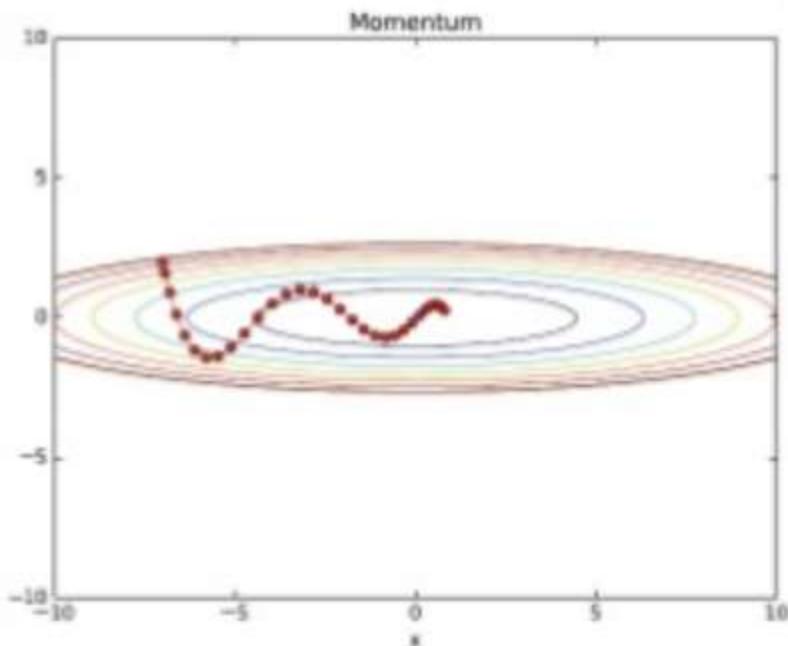
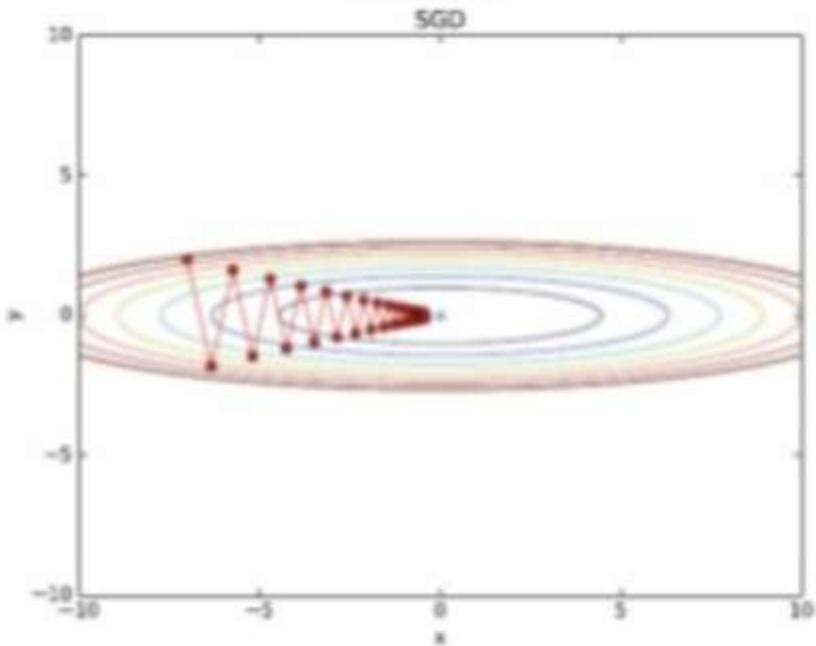
$$x = x + v$$

m (momentum)通常為 0.9



動量 (Momentum)

■ 動量可視為SGD 的加速版本



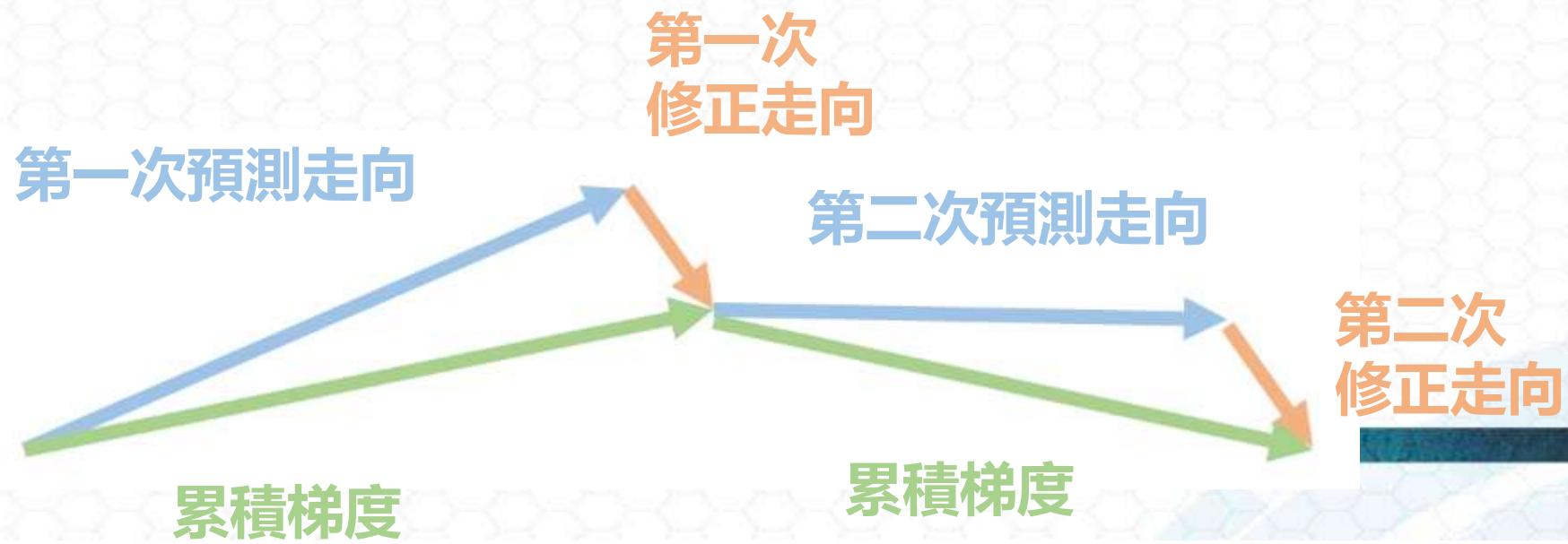
Nesterov accelerated gradient (NAG)

- 梯度更新時，不是盲目追尋最低點，而是產生一個預測指標，預測可能的終點，並且適時減速，避免超過最低點

權重更新函數

$$v = mv - lr * \text{grad}(x + m * v)$$

$$x = x + v$$



自我調整學習率

- 代價函數的梯度大小在不同的層與不同參數可能差異很大，並無法適用同一個學習率，而自我調整學習率的演算法能為每個參數設置不同的學習率，並能在學習過程中自我調整地改變學習率
- 自我調整學習率的演算法：
 - AdaGrad
 - AdaDelta
 - RMSProp
 - Adam

AdaGrad

- 隨著學習過程縮小學習率 (learning rate), 又稱為學習率遞減 (Learning Rate Decay)
- 對於經常變動的參數，學習率會逐漸變小
- 無限學習下去，更新量會降到 0

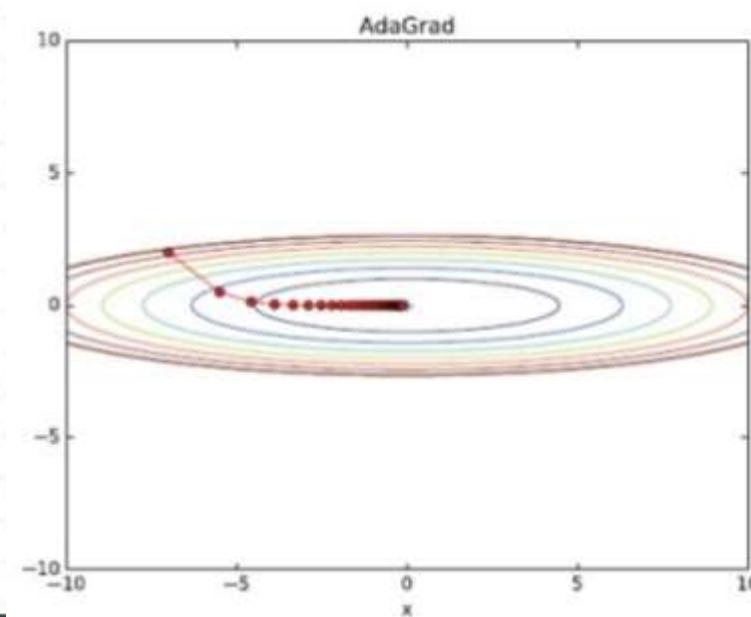
權重更新函數

紀錄過程中所有梯度值的平方和

$$h = h + \text{grad}(x) \odot \text{grad}(x)$$

$$x = x - \ln \frac{1}{\sqrt{h}} \text{grad}$$

調整學習率



AdaDelta

■ Adagrad

- 紀錄過程中的所有梯度值的平方和
- 無限學習下去，更新量會變為 0

■ Adadelta

- 只記錄固定區間（Window）的梯度值平方和

權重更新函數

$$E|grad^2|_t = \gamma E|grad^2|_{t-1} + (1 - \gamma)grad_t^2$$

$$x = x - lr \frac{1}{\sqrt{E|grad^2|_t}} grad$$

γ 類似Momentum

AdaDelta 改良版

■ AdaDelta 甚至不需要設學習率

權重更新函數

$$x = x - lr \frac{1}{\sqrt{E|grad^2|_t}} grad$$



$$x = x - \frac{RMS|x|_{t-1}}{RMS|grad|_t} grad$$

RMSprop

- 與AdaDelta 幾乎同時被提出，其 γ 為一常數，可以當作 Adadelta的一個特例

權重更新函數

$$E|grad^2|_t = 0.9E|grad^2|_{t-1} + 0.1grad_t^2$$

$$x = x - lr \frac{1}{\sqrt{E|grad^2|_t}} grad$$

γ 類似Momentum, Hinton 建議設為0.9

Adam

■ Momentum 與 AdaGrad 的綜合體

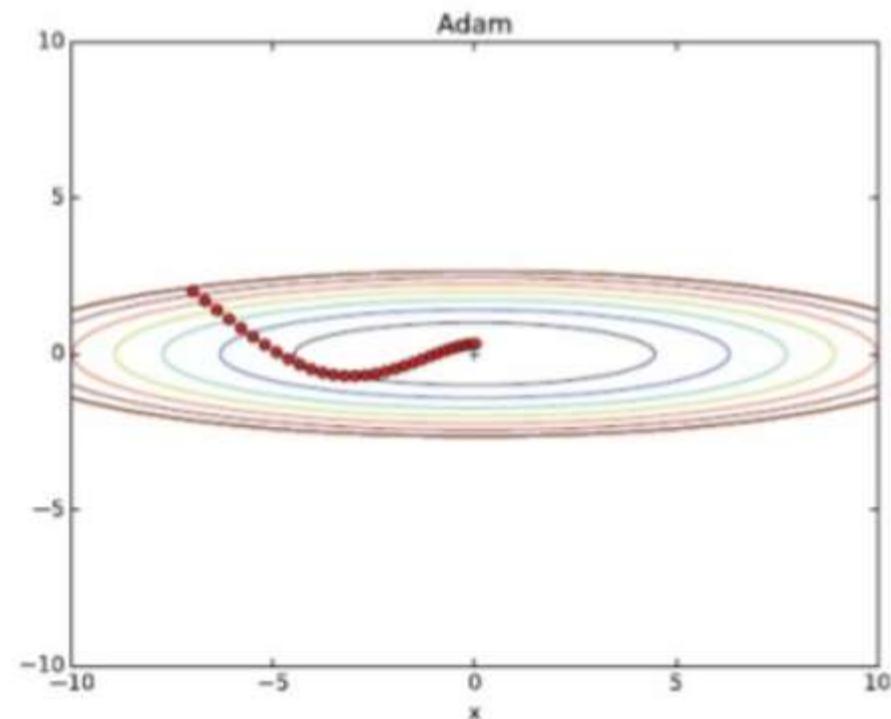
- 透過Momentum 加速收斂
- 透過學習率衰減自動調整學習率

權重更新函數

$$x = x - \frac{lr}{\sqrt{\hat{v}_t}} \hat{m}_t$$

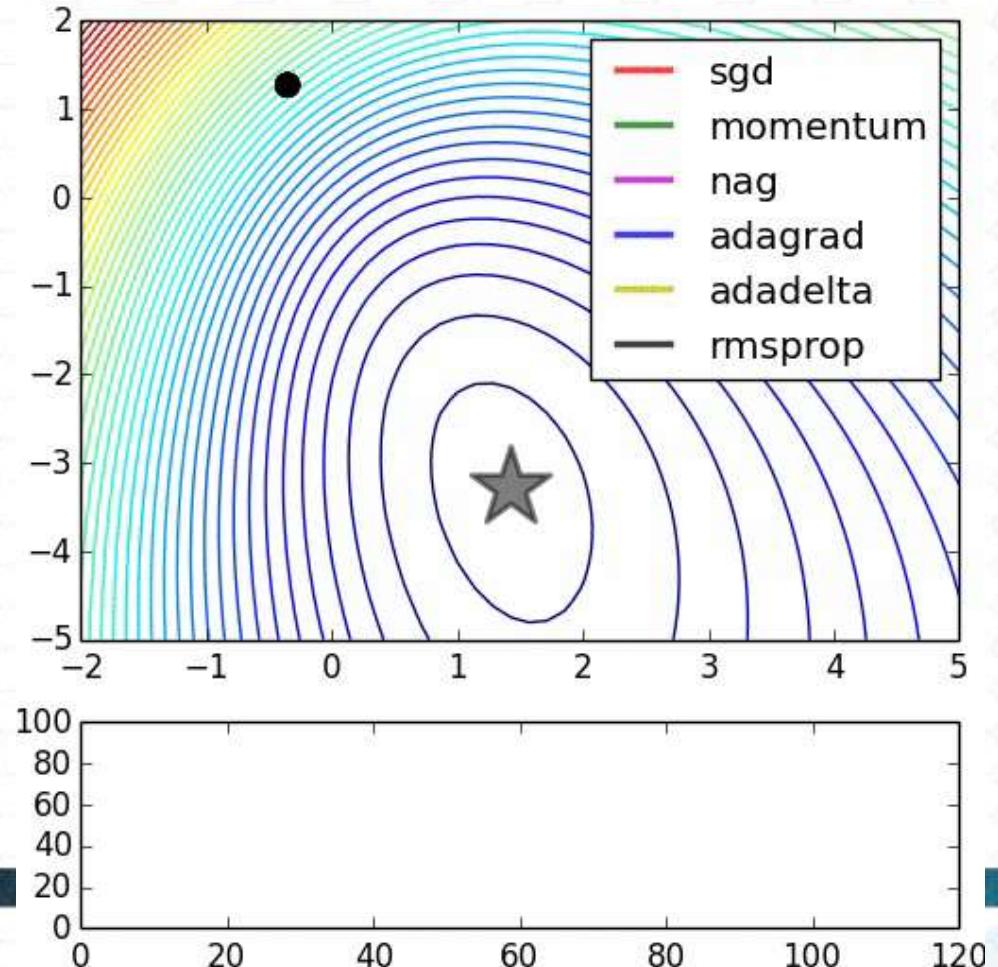
\hat{m}_t : 指數衰減平均值

\hat{v}_t : 平方梯度的指數衰減平均值



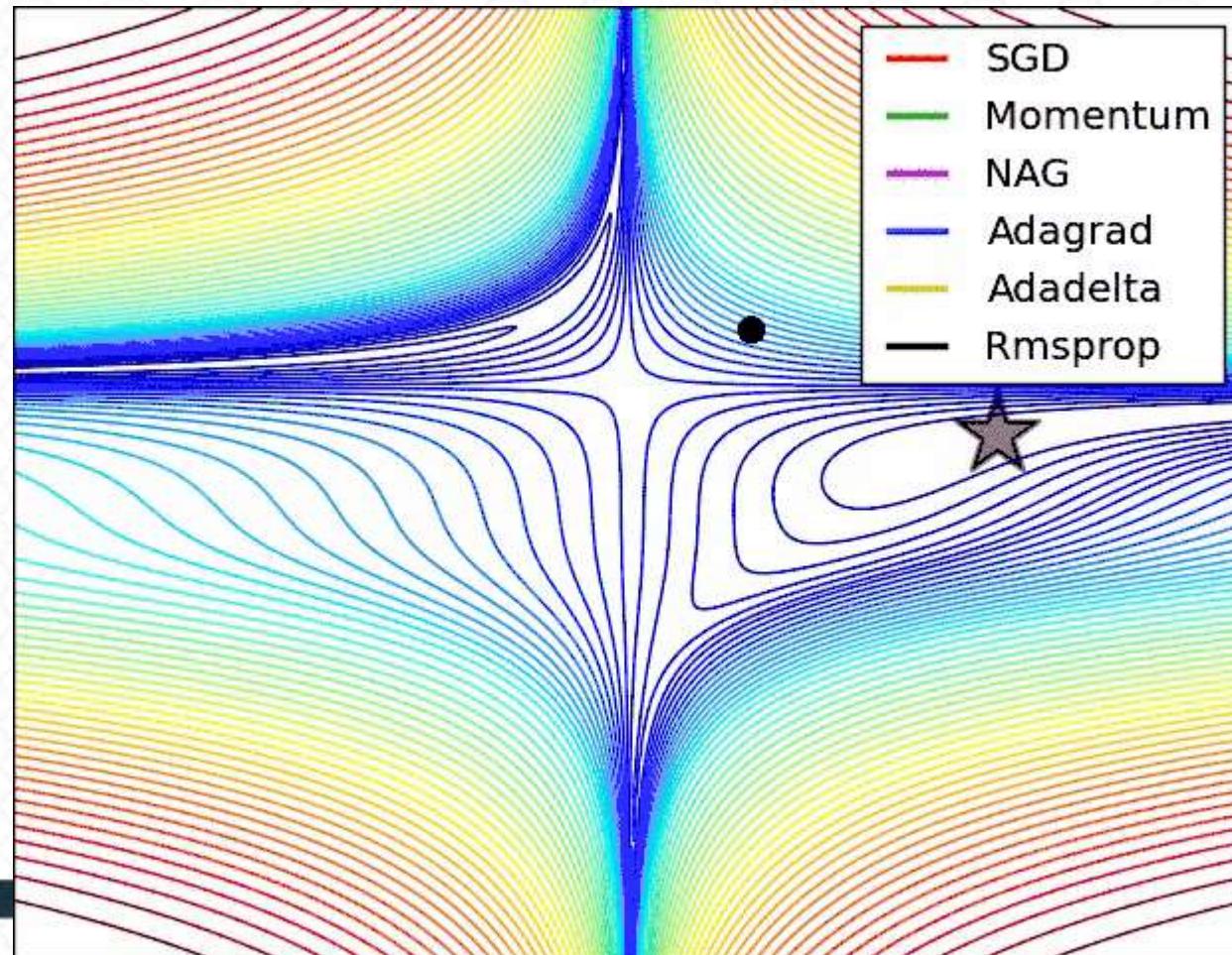
優化演算法動畫

- By Alec Radford
 - <https://bit.ly/2u87PsD>



優化演算法動畫 (二)

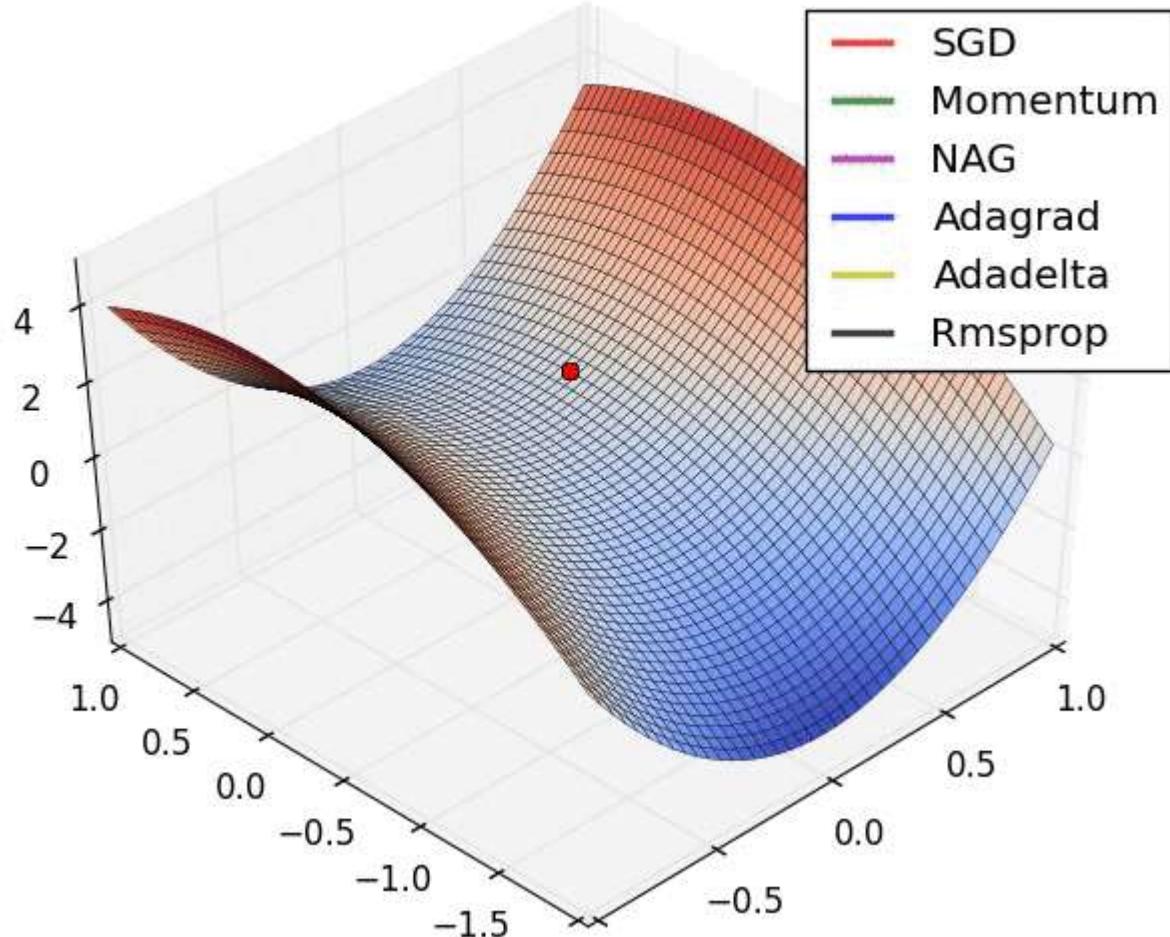
- By Alec Radford
 - <https://bit.ly/2u87PsD>



優化演算法動畫 (三)

■ By Alec Radford

□ <https://bit.ly/2u87PsD>

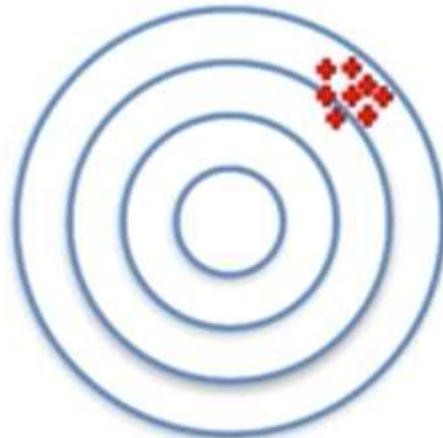


該使用哪種優化器

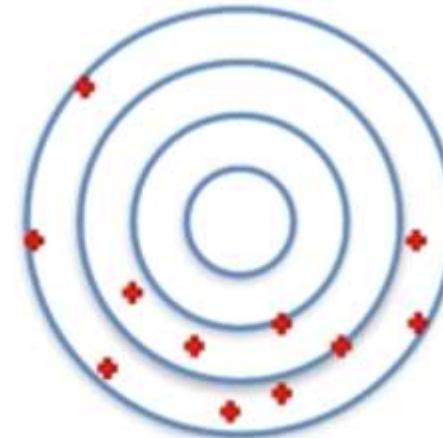
- 對於稀疏資料集，應該使用某種自我調整學習率的方法
 - 自我調整學習率較適用於稀疏資料集
 - 不用選擇學習率
- AdaGrad, RMSprop, AdaDelta 以及 Adam 在多數的表現上非常類似, 不過 Kingma 的比較論文中, Adam 的表現優於 RMSprop
- Adam 在實際應用中效果最好
 - 不過多數的論文都使用 SGD 做優化器!?

評估、調參、優化人工神經網路

Bias (偏差) 和 Variance (方差)



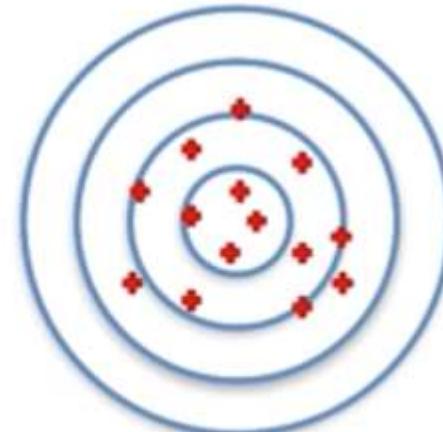
高 Bias (偏差) 低 Variance (方差)



高 Bias (偏差) 高 Variance (方差)



低 Bias (偏差) 低 Variance (方差)



低 Bias (偏差) 高 Variance (方差)

K-fold 交叉驗證

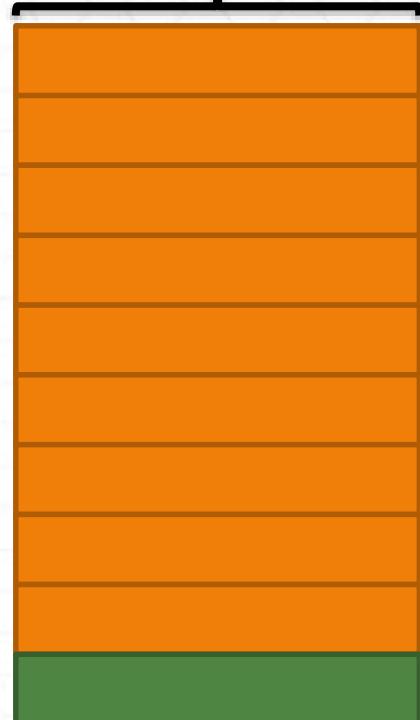


訓練資料集

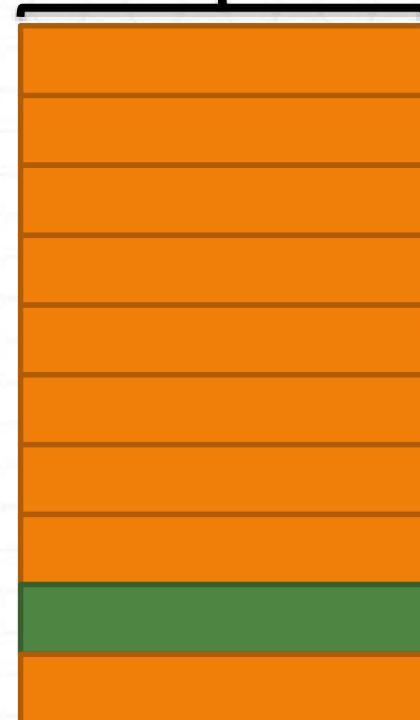


測試資料集

Round 1



Round 2



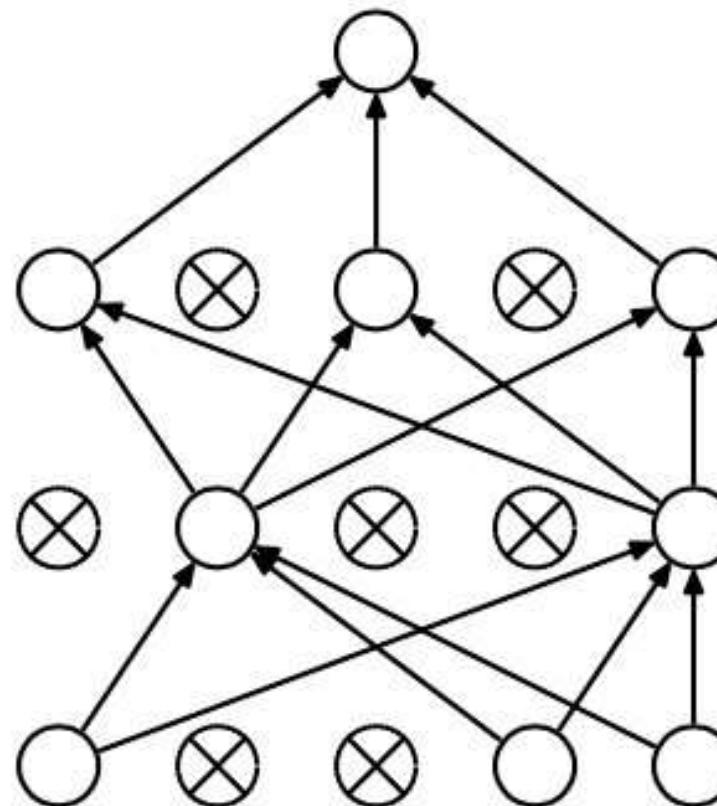
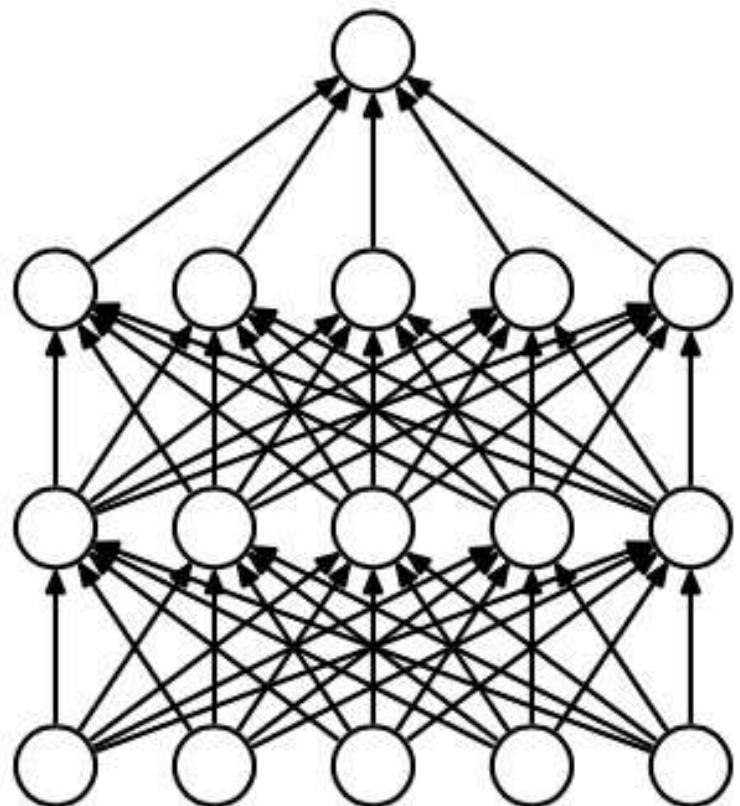
Round 10



⋮ ⋮ ⋮

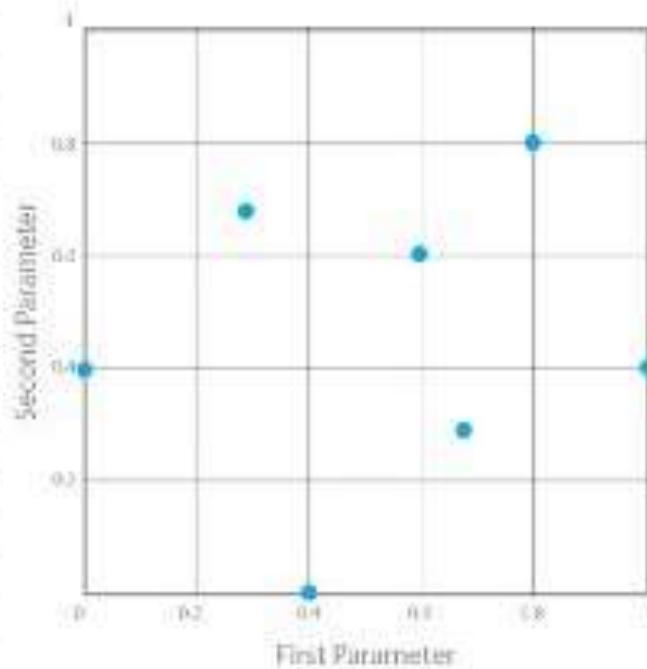
Dropout

■ 隨機剔除神經元解決過擬和問題

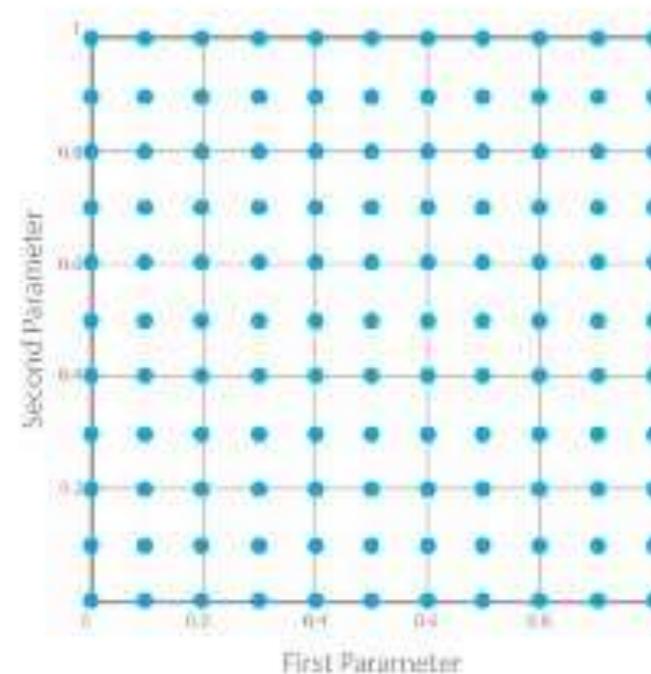


使用Grid Search 調整超參數

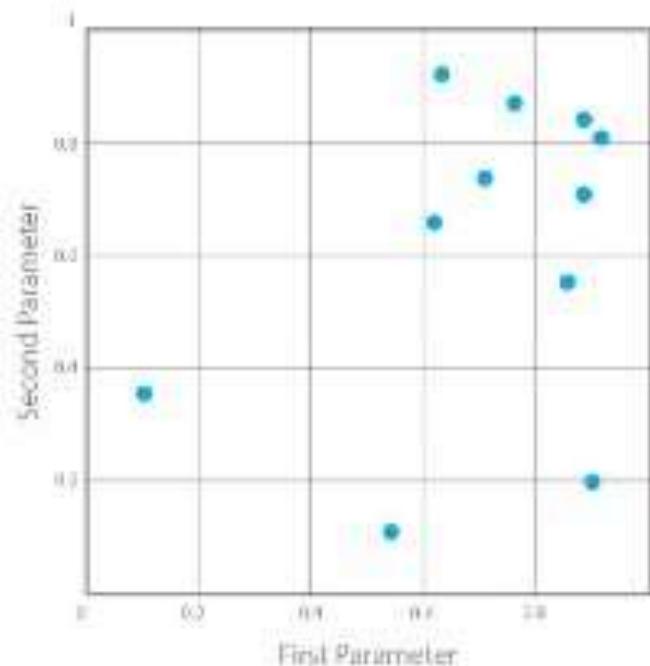
Manual Search



Grid Search

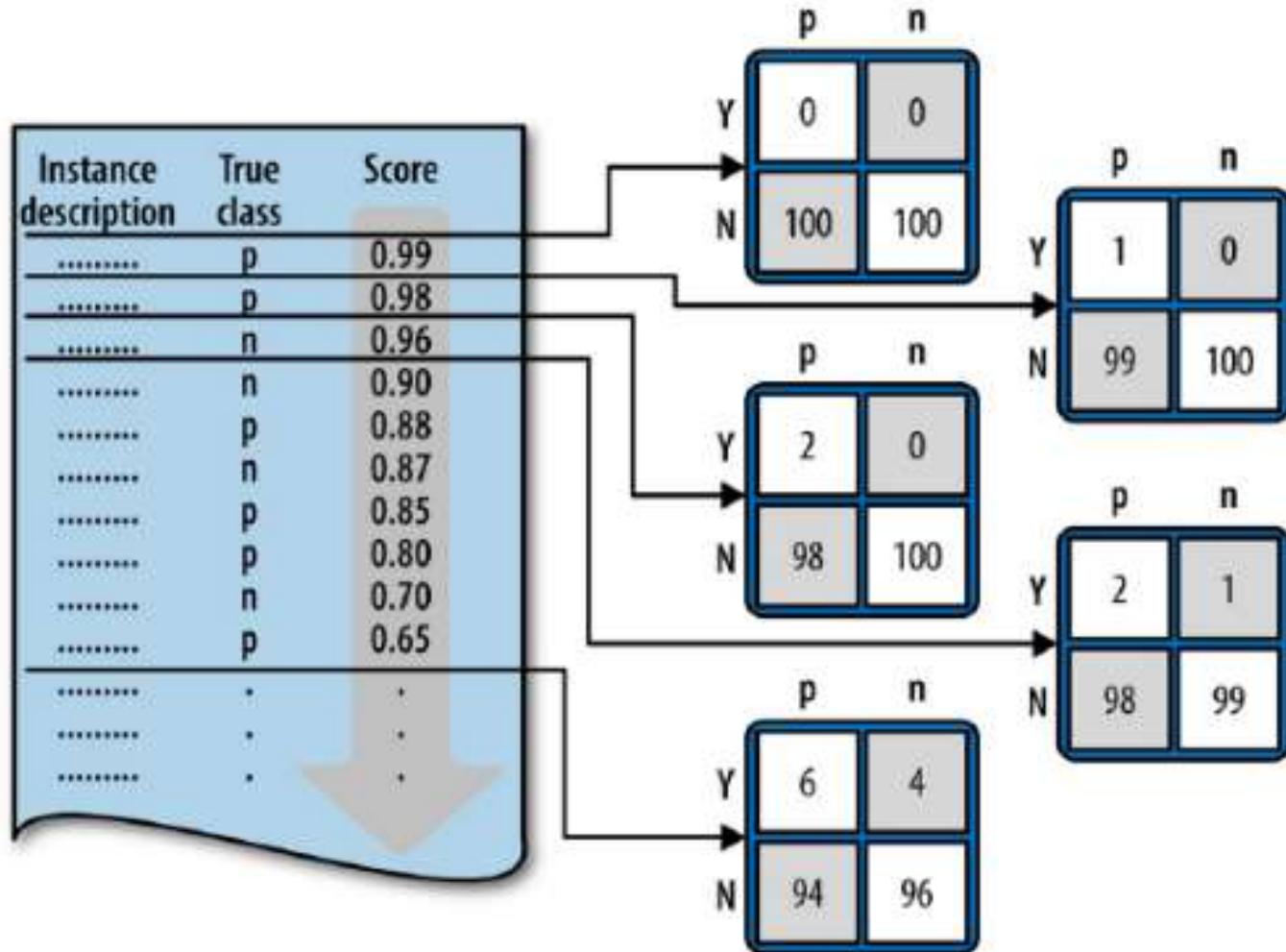


Random Search



比較人工神經網路與其他機器學習模型

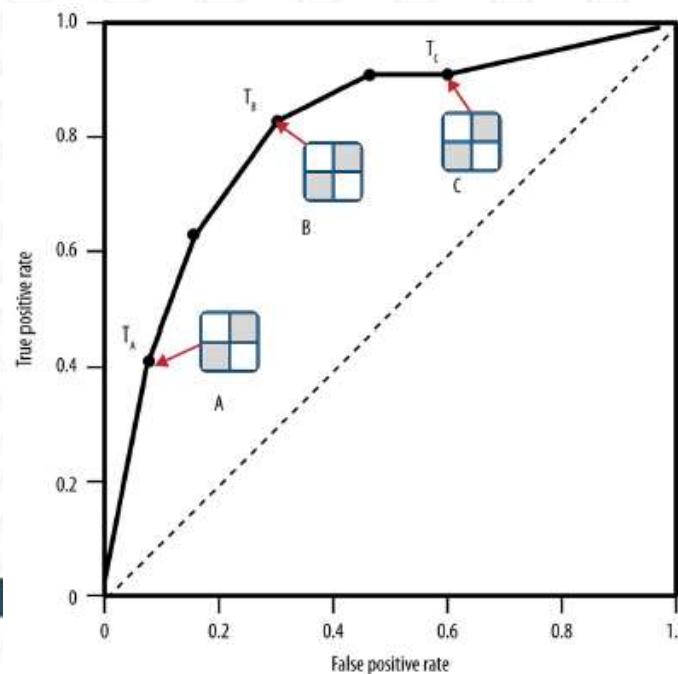
考慮不同成本下的混淆矩陣



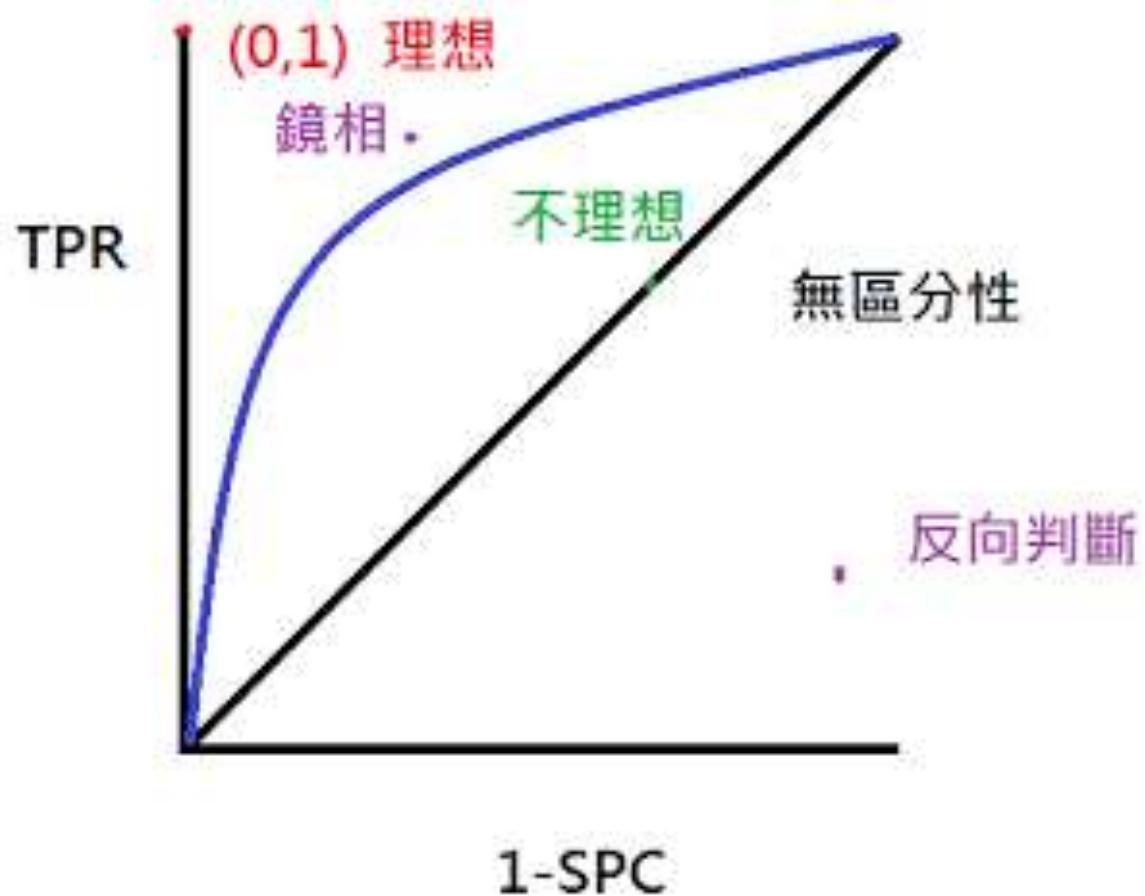
ROC 曲線

■ 接收者操作特徵(receiver operating characteristic, ROC curve)

1. 以假陽性率(False Positive Rate, FPR)為X軸，代表在所有陰性相本中，被判斷為陽性(假陽性)的機率，又寫為(1-特異性)。
2. 以真陽性率(True Positive Rate, TPR)為Y軸，代表在所有陽性樣本中，被判斷為陽性(真陽性)的機率，又稱為敏感性



評估 ROC 曲線



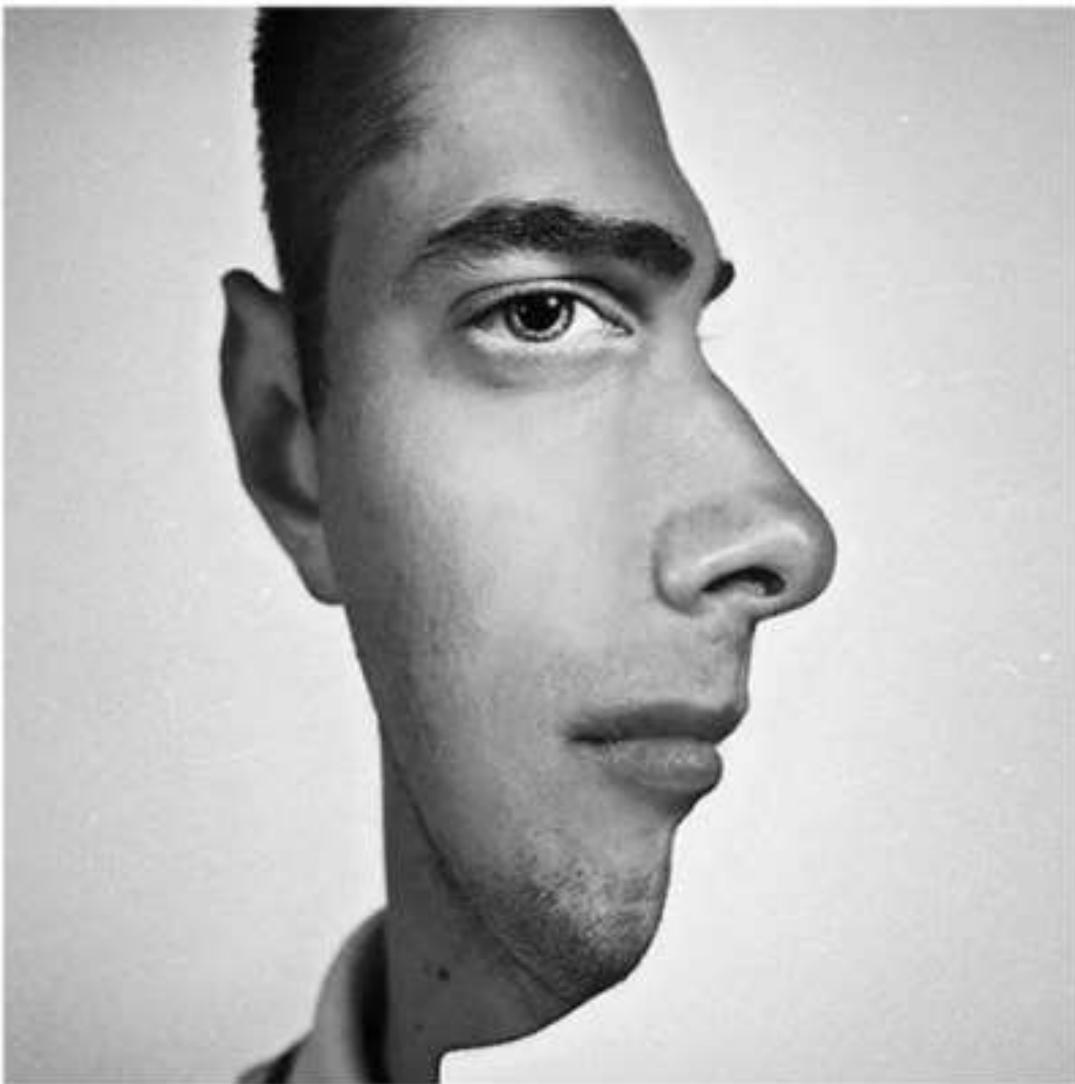
AUC

曲線下面積(Area Under Curve, AUC)為此篩檢方式性能優劣之指標，AUC越接近1，代表此篩檢方式效能越佳。指標可參考以下條件。

AUC數值	解釋
1	完美分類器，無論cut-off point如何設定都可正確預測。 通常不存在
$0.5 < \text{AUC} < 1$	優於隨機，妥善設定可有預測價值
0.5	同隨機，預測訊息沒有價值

什麼是卷積神經網路

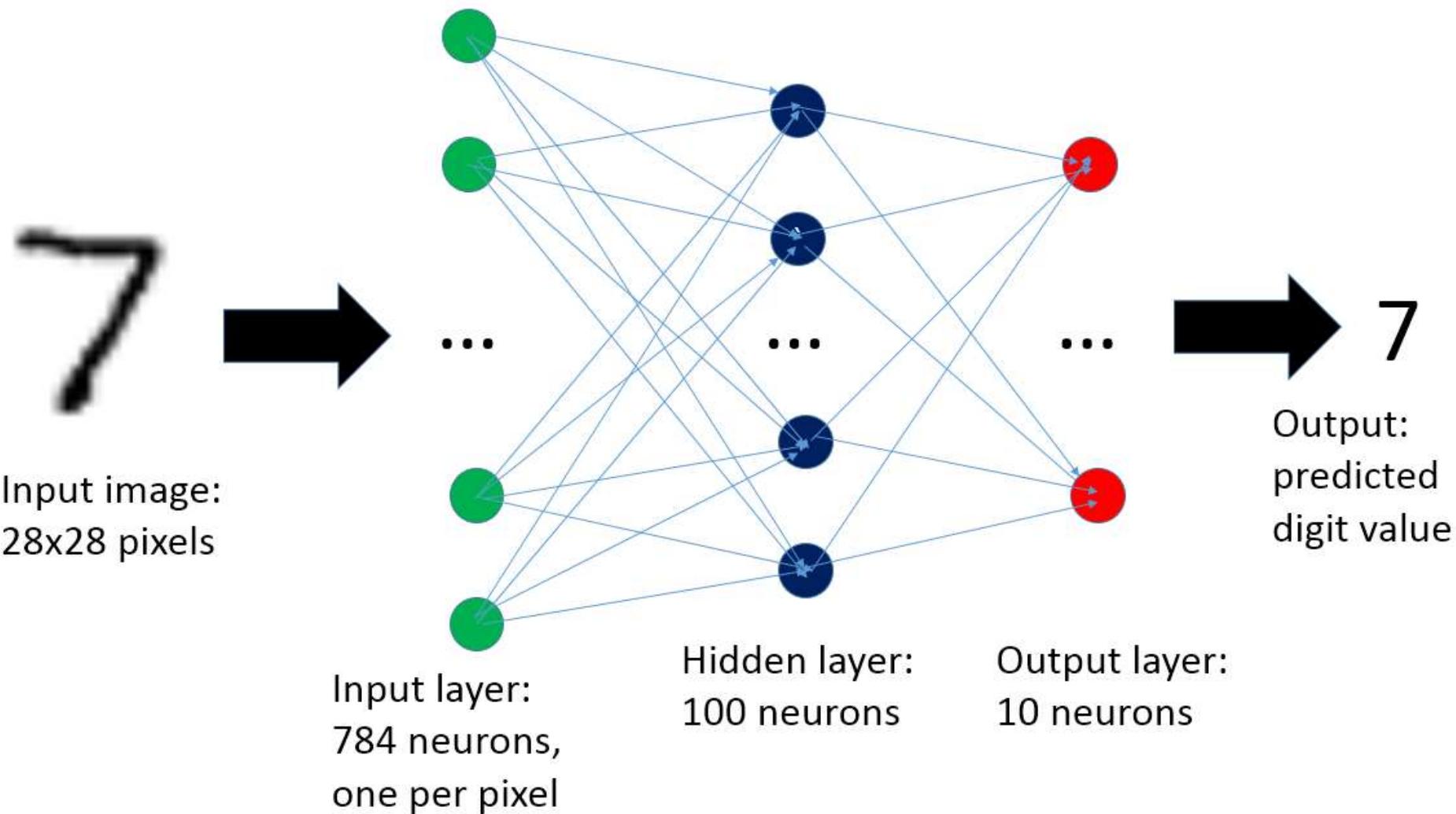
側臉或正面？



老太太或妙齡女子？



如何辨識手寫數字？



ANN在圖形識別時的缺點

■ 參數數量龐大

輸入 28×28 圖元的圖片，輸入層有784節點。假設第一個隱藏層有100個節點，一層就有 $(784+1) \times 100 = 78,500$ 個參數

■ 沒有利用圖像特徵去學習

每個圖元只與周邊的圖元有關，識別出圖像的特徵，可以大幅加速學習過程，提高學習準確率。



■ 網路層數限制

當節點太多，自然無法建立出較多層數的神經網路

Yann LeCun

■ <http://yann.lecun.com/>



根據特徵做學習？

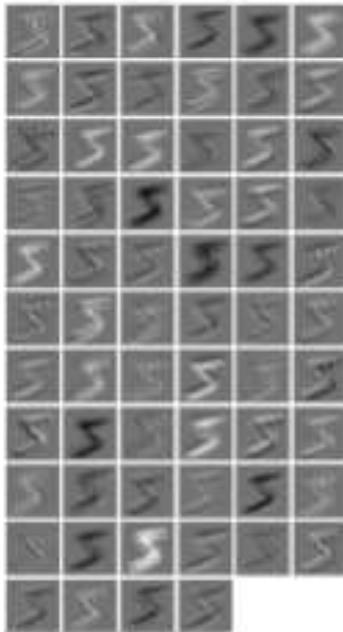


Figure 1: The 64 feature maps after first layer of first block (spatial convolution)



Figure 2: The 64 feature maps after second layer of first block (tanh)



Figure 3: The 64 feature maps after third layer of first block (L2 Pooling)

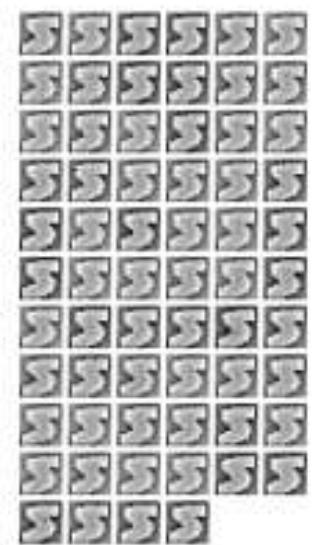
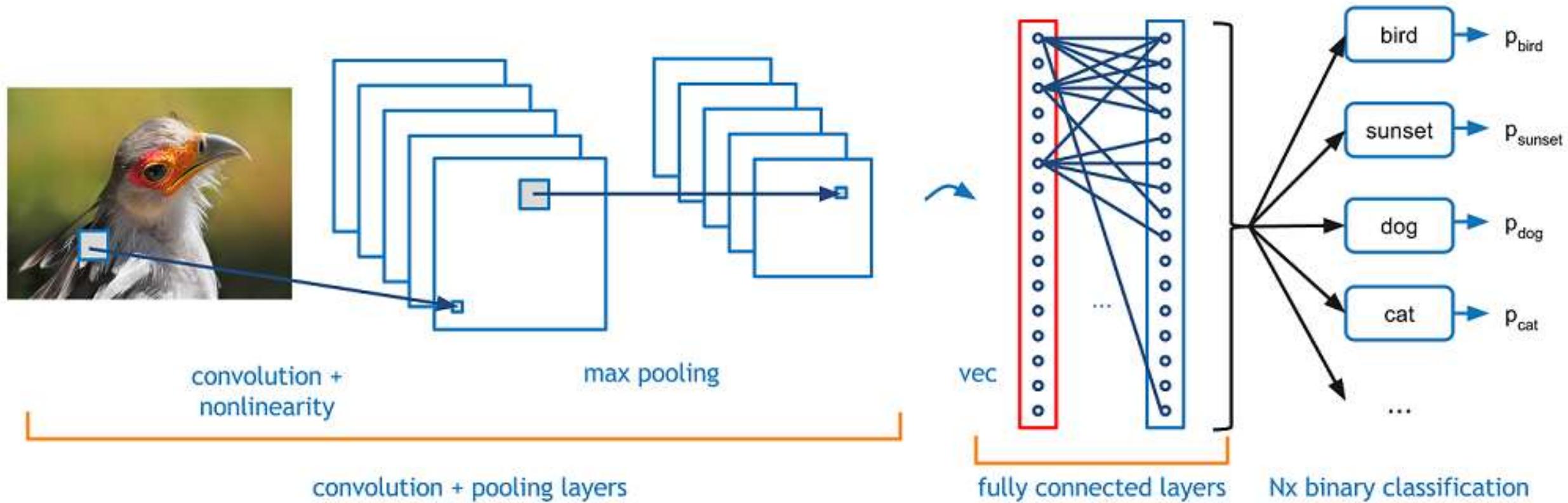


Figure 4: The 64 feature maps after fourth layer of first block (subtractive normalization)

卷積神經網路架構



卷積神經網路方法



卷積特徵提取

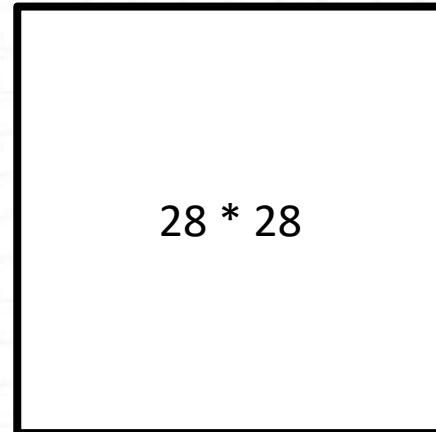
將圖片變為數學矩陣



2d Array



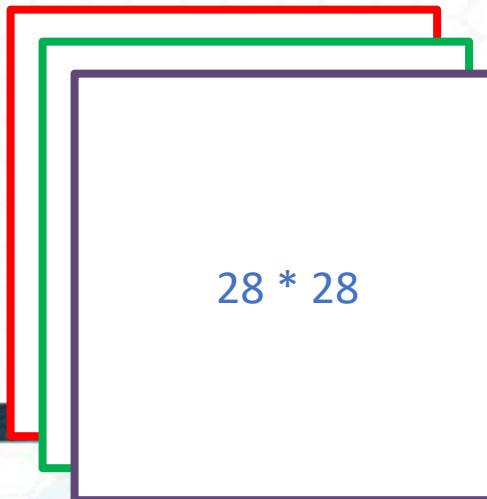
$28 * 28$



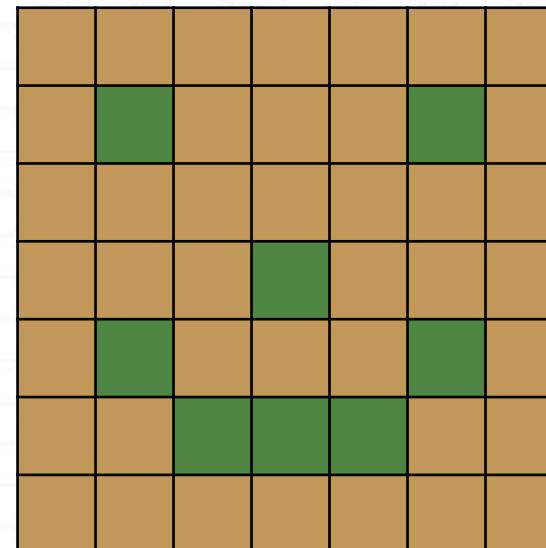
3d Array



$28 * 28$



將圖片變為數學矩陣



0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

卷積特徵提取

0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0



0	0	1
1	0	0
0	1	1

=

0				

輸入圖片

濾鏡
(Filter or Feature Detector)

卷積特徵提取

Stride

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0



0	0	1
1	0	0
0	1	1

=

0	1			

輸入圖片

濾鏡
(Filter or Feature Detector)

卷積特徵提取

0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0



0	0	1
1	0	0
0	1	1

=

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4			

輸入圖片

濾鏡
(Filter or Feature Detector)

卷積特徵提取

0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0



0	0	1
1	0	0
0	1	1

=

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

輸入圖片

濾鏡
(Filter or Feature Detector)

Padding

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0



0	0	1
1	0	0
0	1	1

=

1	1	0	0	1	1	0
0	0	1	0	0	0	1
1	0	1	1	1	0	0
1	1	0	1	2	1	0
0	1	4	2	1	0	1
1	0	0	1	2	1	0
0	0	1	1	1	0	0

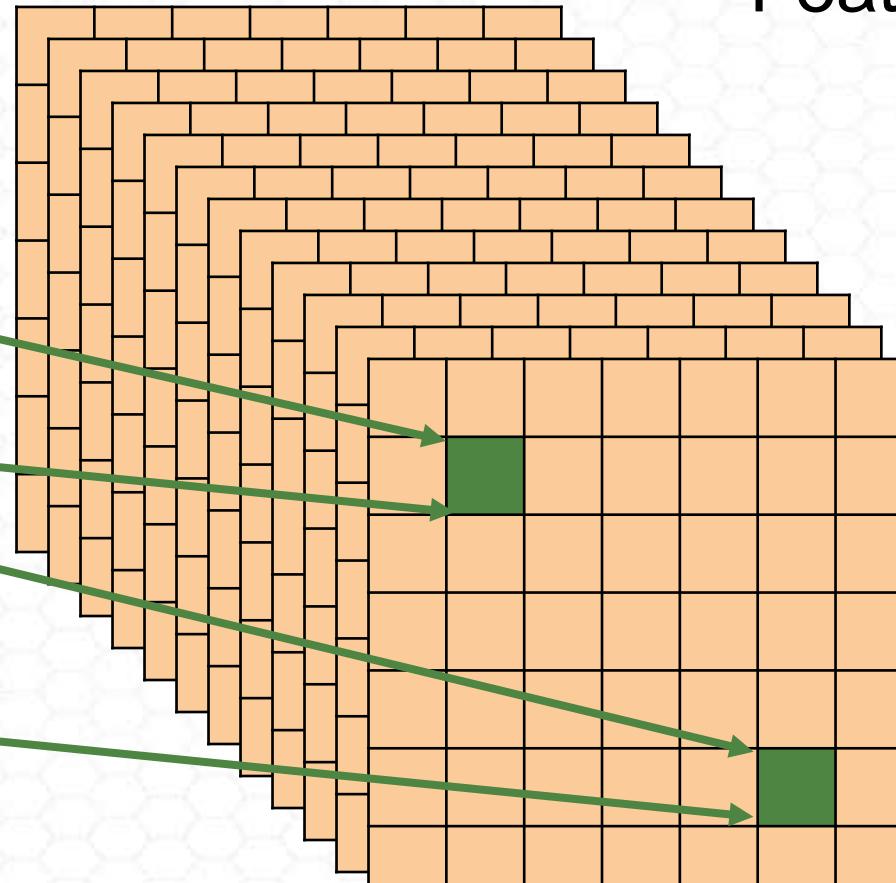
輸入圖片

濾鏡
(Filter or Feature Detector)

Convolution Layer

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

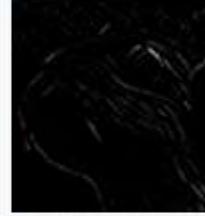
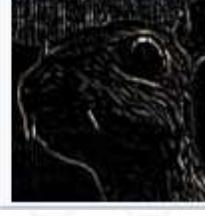
輸入圖片



Feature Map

Convolution Layer

卷積矩陣

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

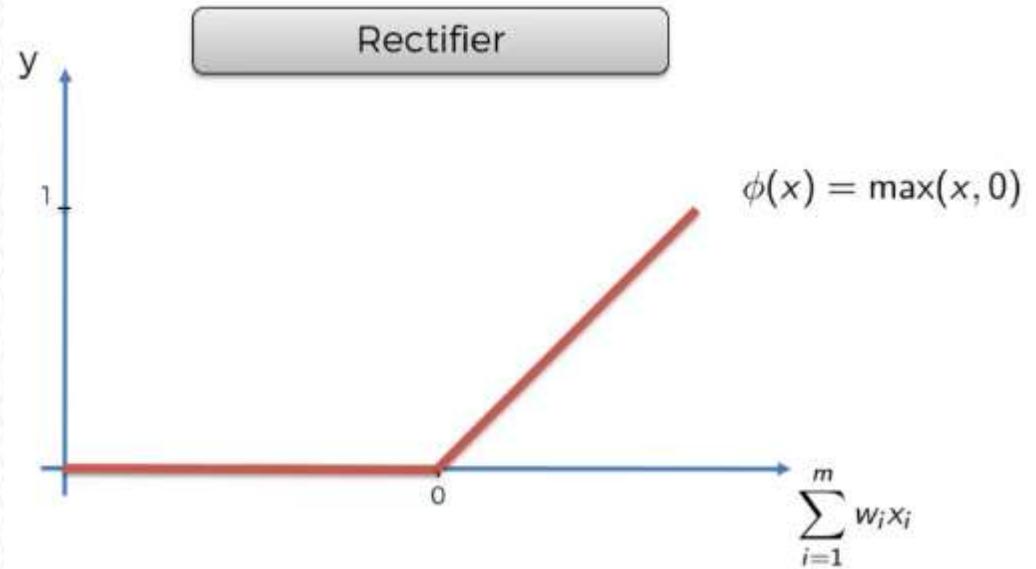
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5×5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5×5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

[https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

RELU層

ReLU 函數

```
def relu_function(x):  
    return np.maximum(0,x)  
  
x = np.array([-1,1,2])  
relu_function(x)
```



優點:

- 快速收斂
- 解決梯度消失問題

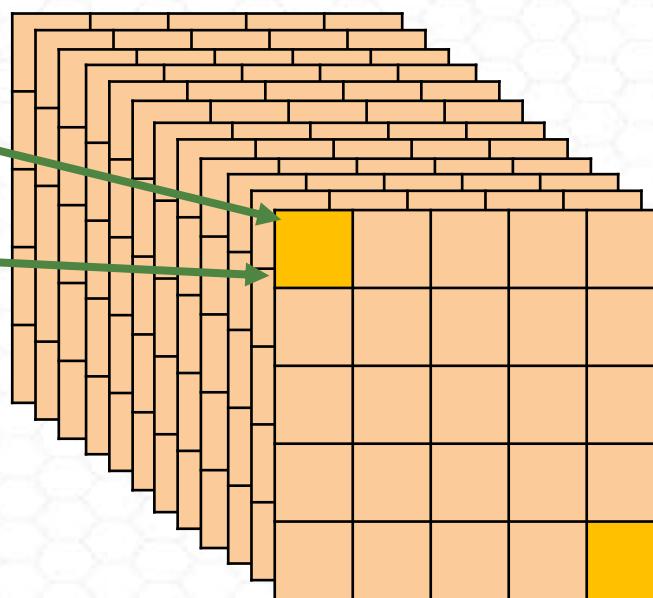
缺點:

- 神經元死亡問題

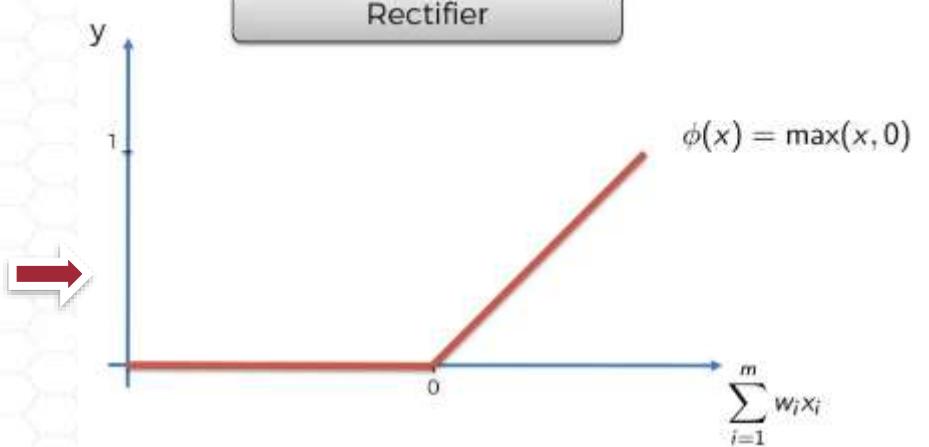
增加ReLU層

Feature Map

0	0	0	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	1	0
0	1	0	0	0



Convolution Layer



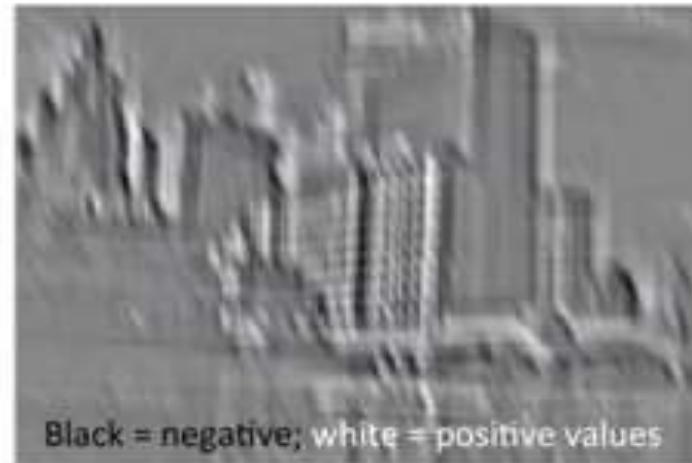
ReLU Layer

ReLU效果

Original:



Feature Detector:



ReLU:



[https://github.com/Achronus/Machine-Learning-101/wiki/Convolutional-Neural-Networks-\(CNN\)](https://github.com/Achronus/Machine-Learning-101/wiki/Convolutional-Neural-Networks-(CNN))

池化層

如何用不同角度辨認老鷹？



如何用不同角度辨認老鷹？



池化層 (Pooling layer)

1. 保留重要的特徵
2. 具有抗干擾的作用
3. 減少過擬合(Over-Fitting)

池化過程

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

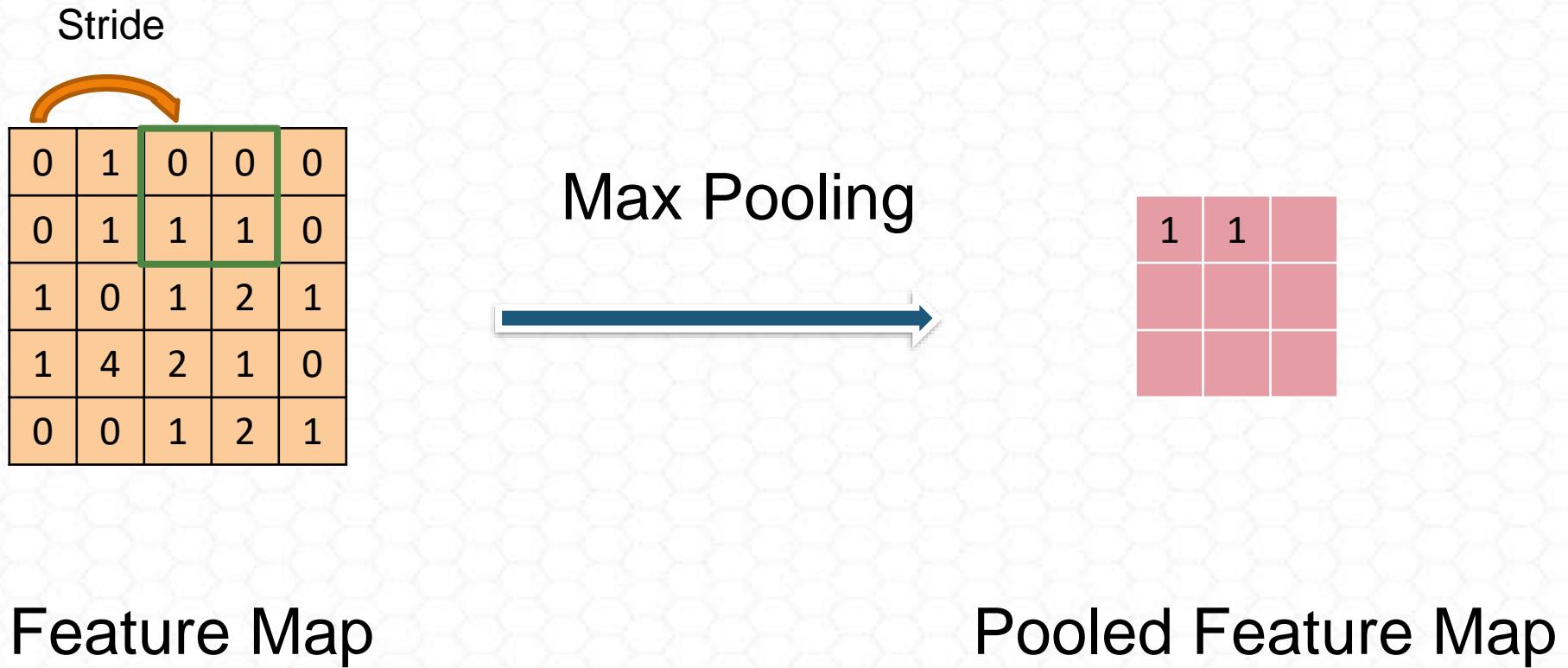
Max Pooling



1		

Pooled Feature Map

池化過程



池化過程

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Max Pooling



1	1	0
4	2	1
0	2	1

Pooled Feature Map

平化 (FLATTENING)

平化 (Flattening)

1	1	0
4	2	1
0	2	1

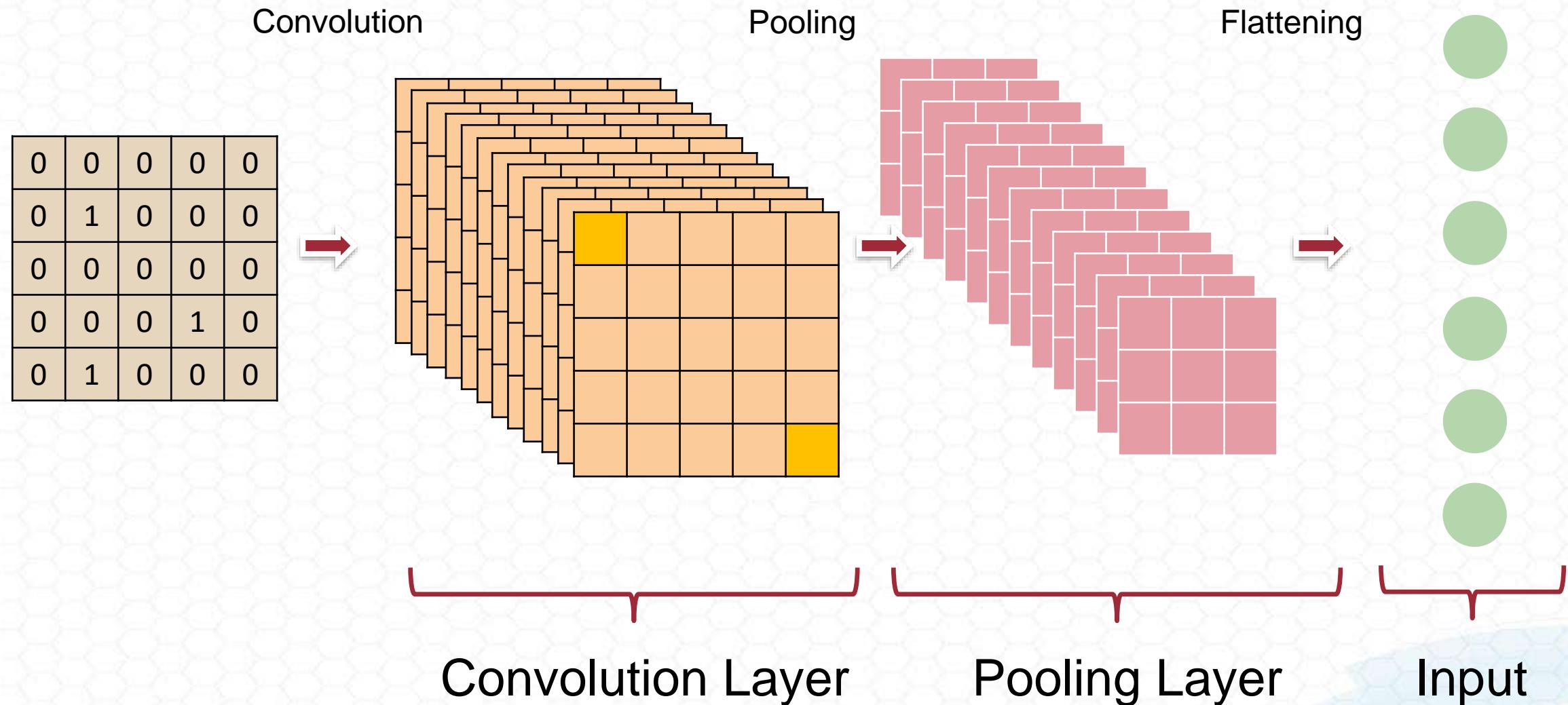
Flattening



1
1
0
4
2
1
0
2
1

Pooled Feature Map

平化 (Flattening)

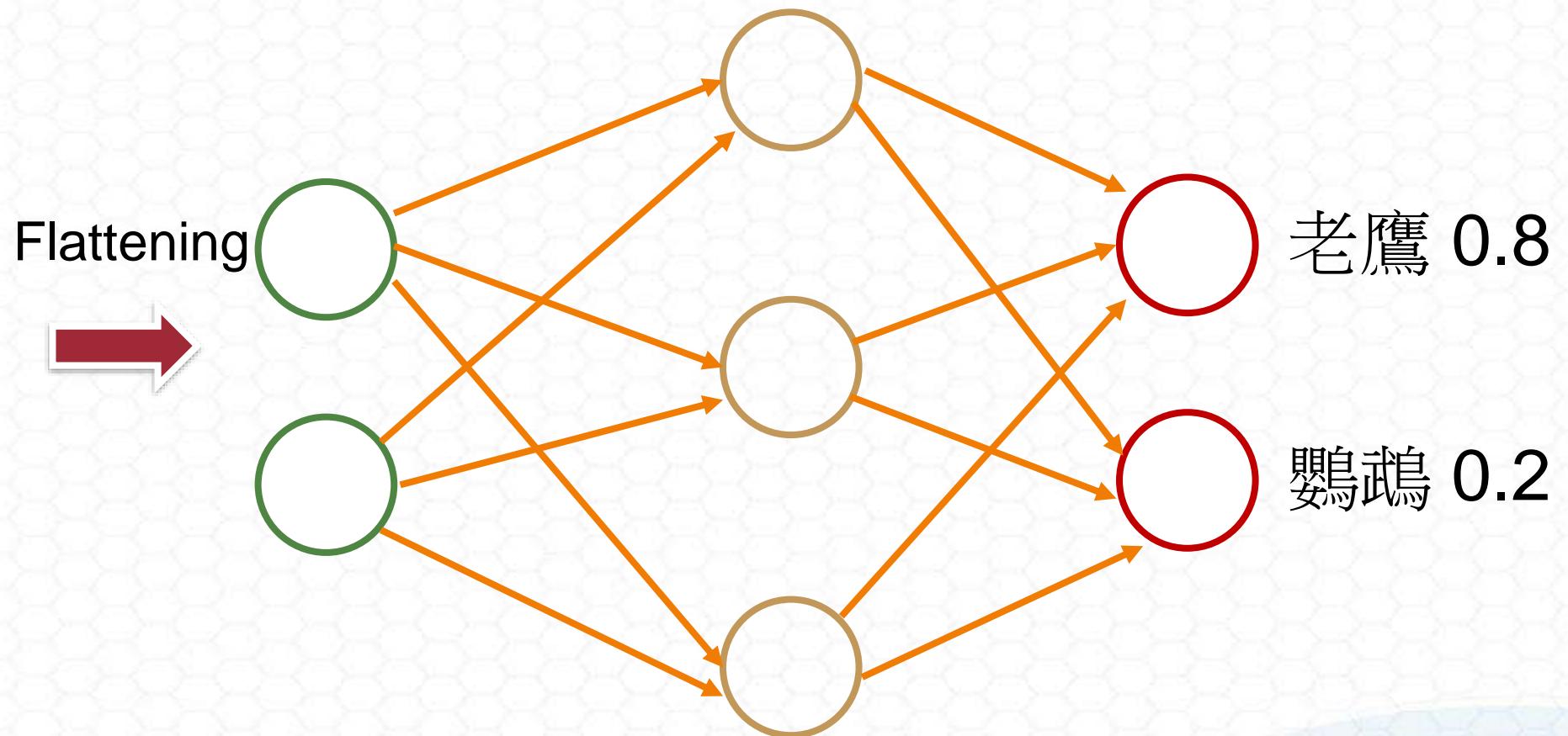


建立卷積神經網路

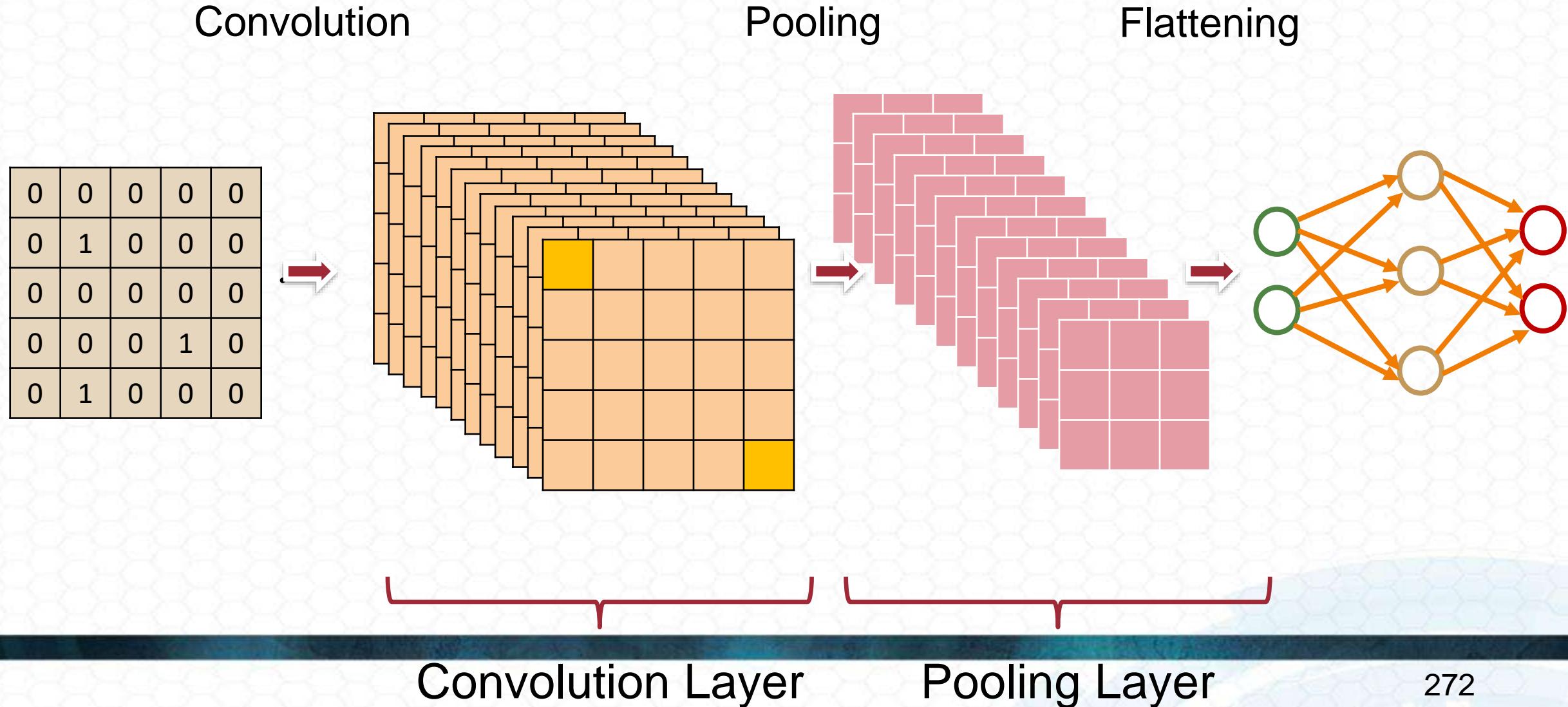
卷積神經網路方法



建立卷積神經網路



建立卷積神經網路



[實例] 利用卷積神經網路辨識圖片

利用卷積神經網路辨識明星圖片



使用網路爬蟲抓取明星圖片



使用OpenCV 擷取人臉



Mac 安装OpenCV

```
# add opencv  
brew tap homebrew/science
```

```
# install opencv  
brew install opencv
```

```
# install opencv3  
brew install opencv3
```

Windows 安裝OpenCV

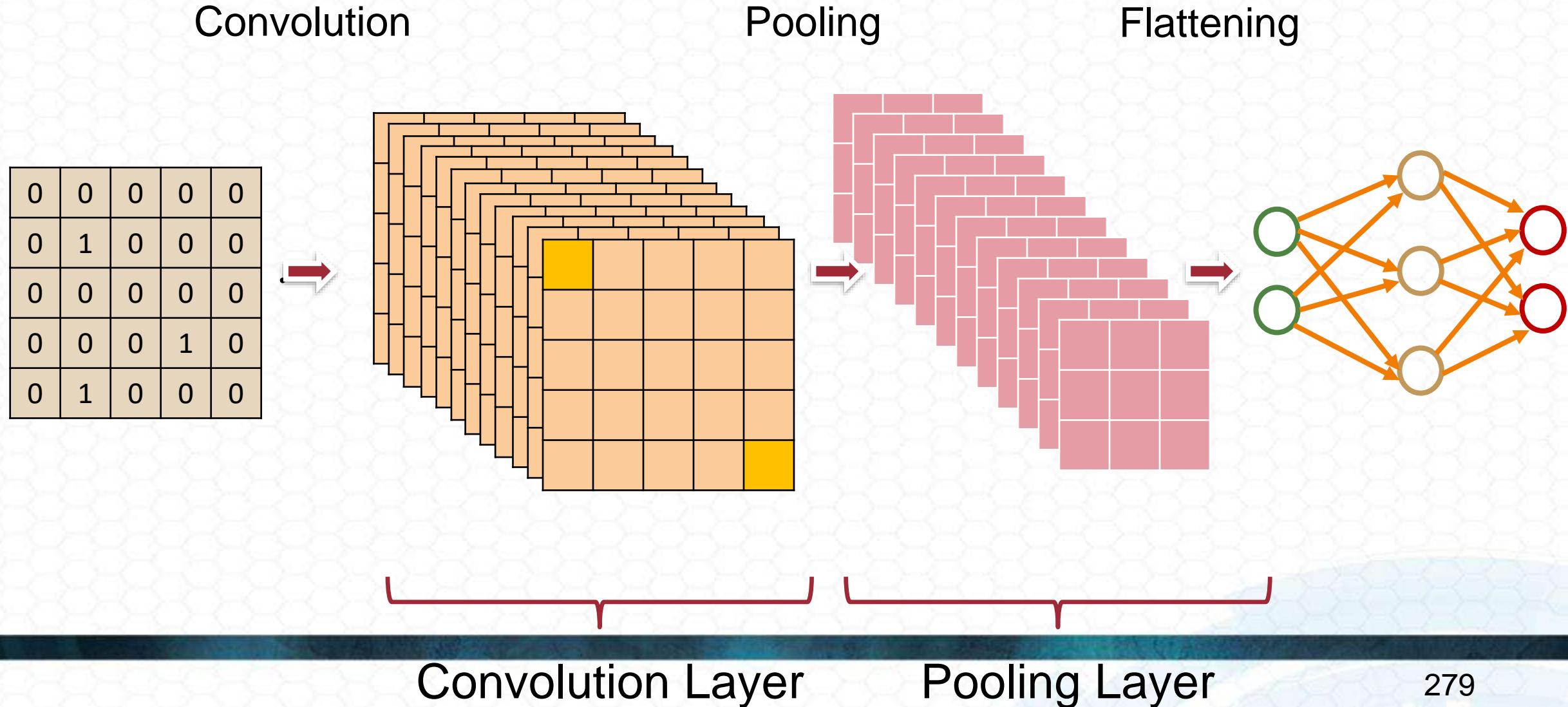
■ 下載OpenCV

□ <https://www.lfd.uci.edu/~gohlke/pythonlibs/>

OpenCV, a real time computer vision library.

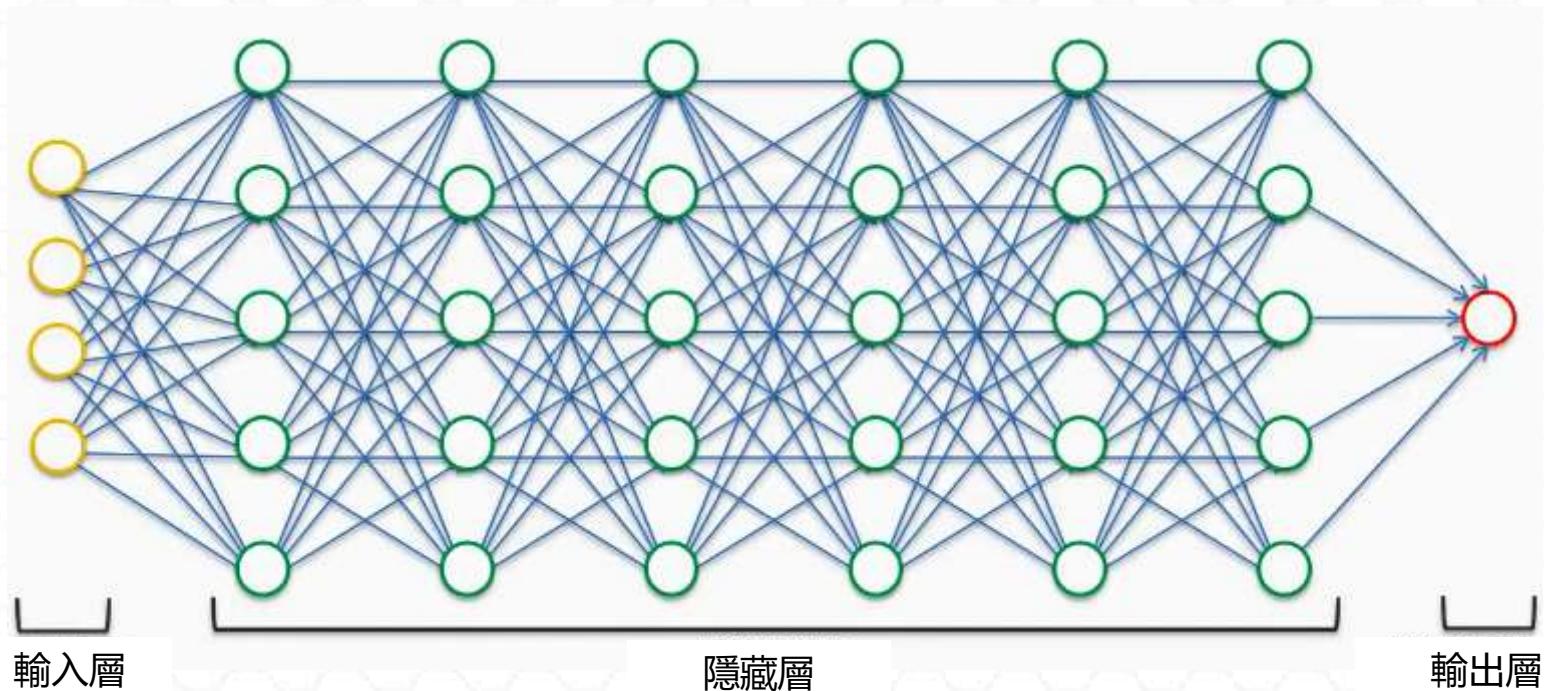
[opencv_python-2.4.13.5-cp27-cp27m-win32.whl](#)
[opencv_python-2.4.13.5-cp27-cp27m-win_amd64.whl](#)
[opencv_python-3.1.0-cp34-cp34m-win32.whl](#)
[opencv_python-3.1.0-cp34-cp34m-win_amd64.whl](#)
[opencv_python-3.4.2+contrib-cp35-cp35m-win32.whl](#)
[opencv_python-3.4.2+contrib-cp35-cp35m-win_amd64.whl](#)
[opencv_python-3.4.2+contrib-cp36-cp36m-win32.whl](#)
[opencv_python-3.4.2+contrib-cp36-cp36m-win_amd64.whl](#)
[opencv_python-3.4.2+contrib-cp37-cp37m-win32.whl](#)
[opencv_python-3.4.2+contrib-cp37-cp37m-win_amd64.whl](#)
[opencv_python-3.4.2-cp35-cp35m-win32.whl](#)
[opencv_python-3.4.2-cp35-cp35m-win_amd64.whl](#)
[opencv_python-3.4.2-cp36-cp36m-win32.whl](#)
[opencv_python-3.4.2-cp36-cp36m-win_amd64.whl](#)
[opencv_python-3.4.2-cp37-cp37m-win32.whl](#)
[opencv_python-3.4.2-cp37-cp37m-win_amd64.whl](#)

建立卷積神經網路



什麼是遞迴神經網路

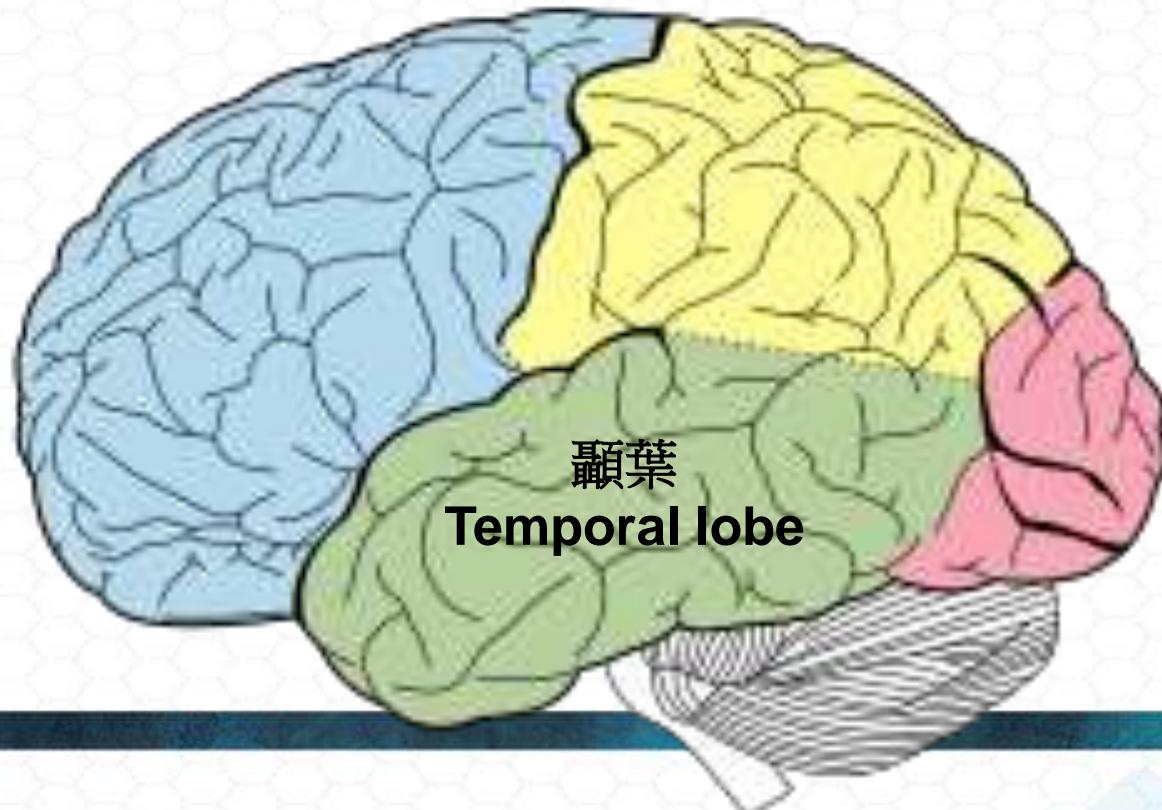
一般神經網路的缺陷



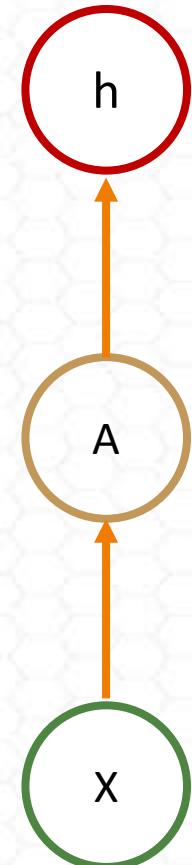
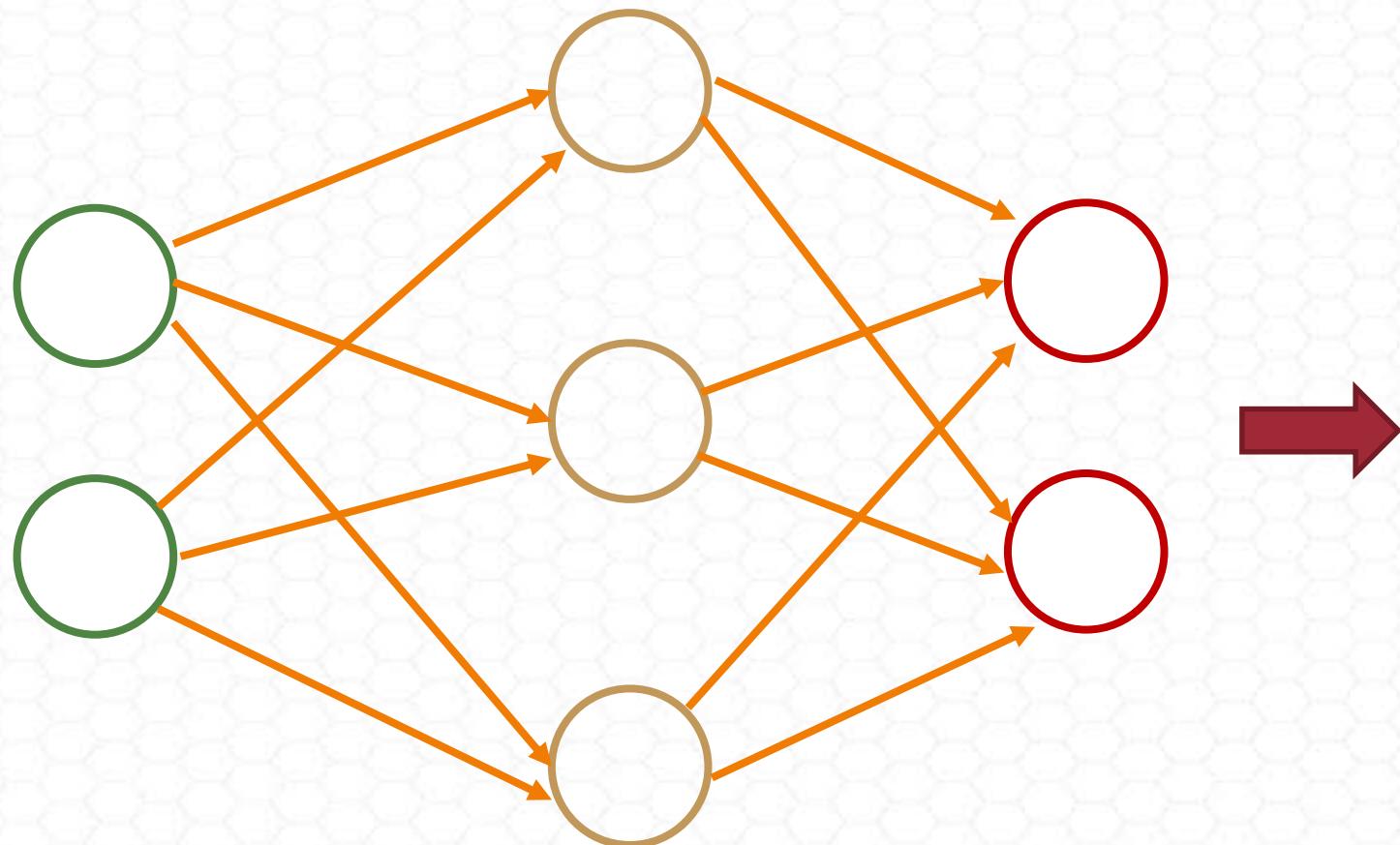
原文: no one else loves me like you do
譯文: 沒有 人 其他 愛 我 像 你 做

大腦的記憶能力

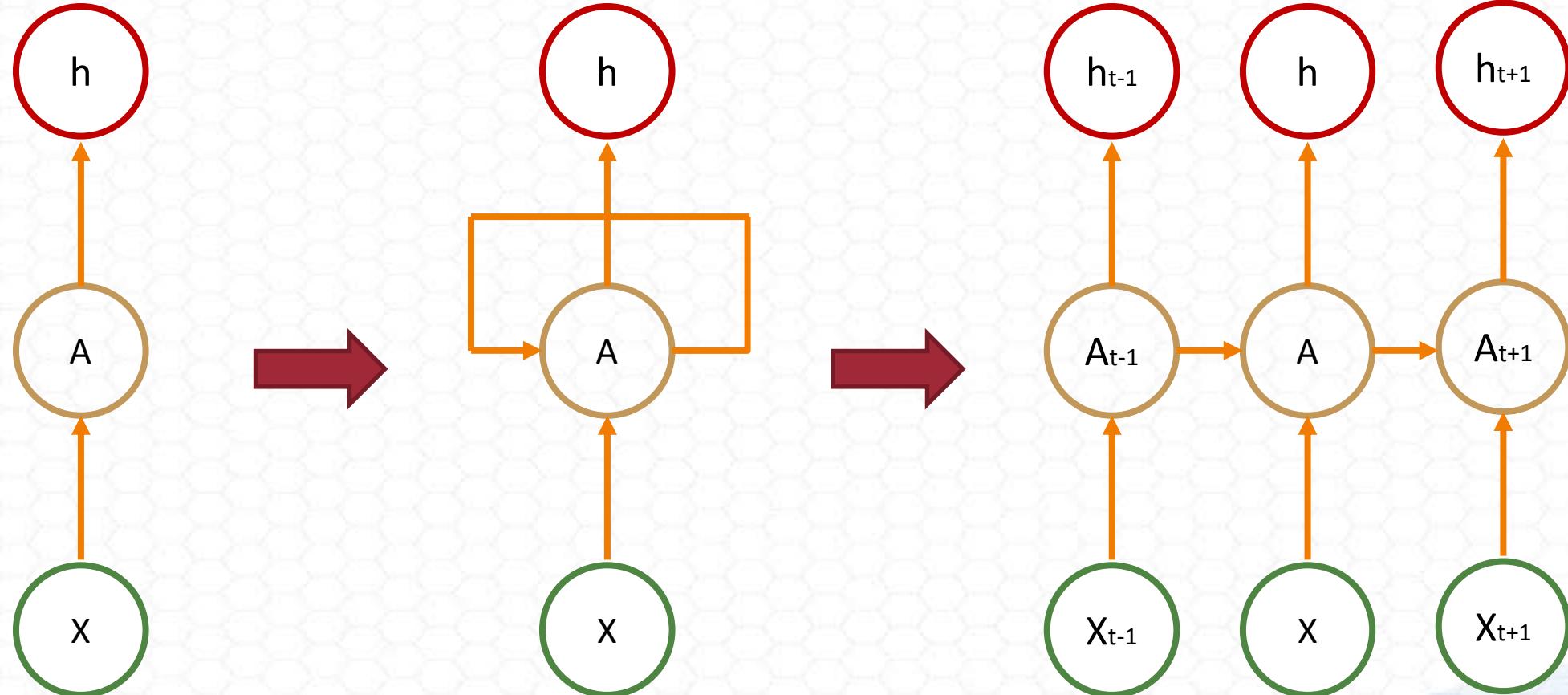
- 顳葉位於兩耳附近的位置，主要提供聽覺與嗅覺上的分辨能力，還有提供新信息的整理能力
 - 右顳葉腦主要在負責視覺化的記憶能力，例如對圖片與臉型的記憶
 - 左顳葉腦則負責語言上的記憶能力，例如單字與名字的記憶



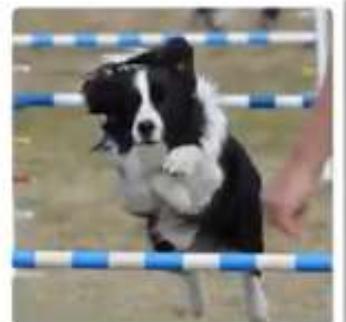
遞迴神經網路



遞迴神經網路

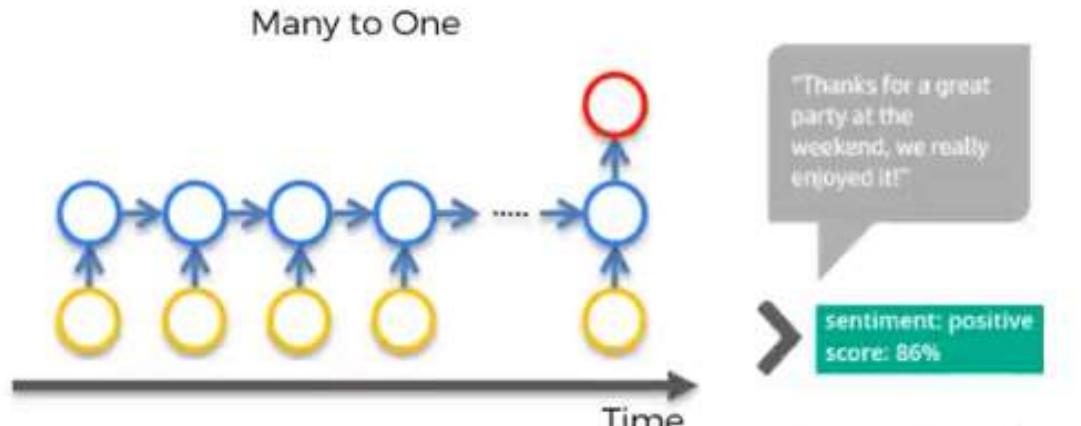
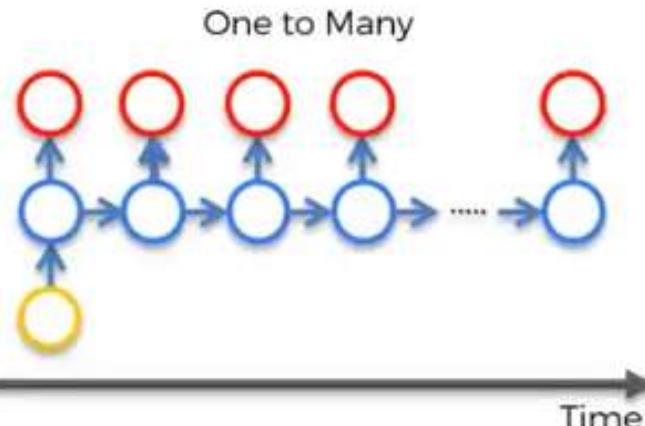


遞迴神經網路的應用



"black and white
dog jumps over
bar."

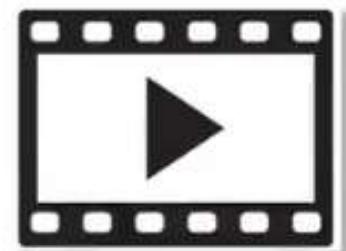
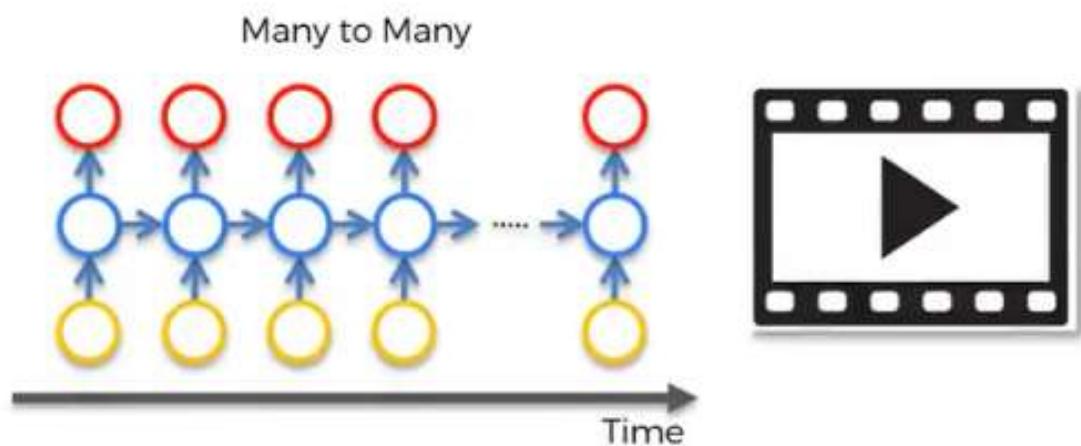
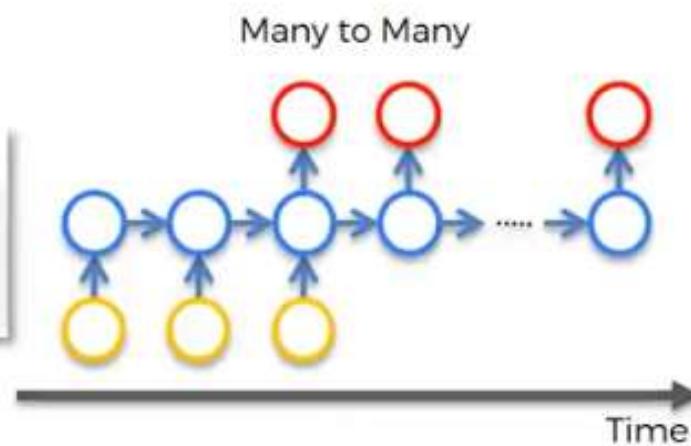
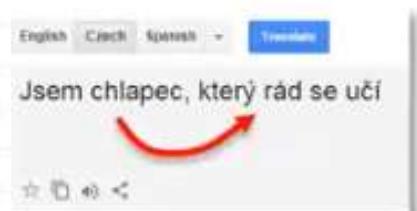
karpathy.github.io



"Thanks for a great
party at the
weekend, we really
enjoyed it!"

> sentiment: positive
score: 86%

dev.havenondemand.com

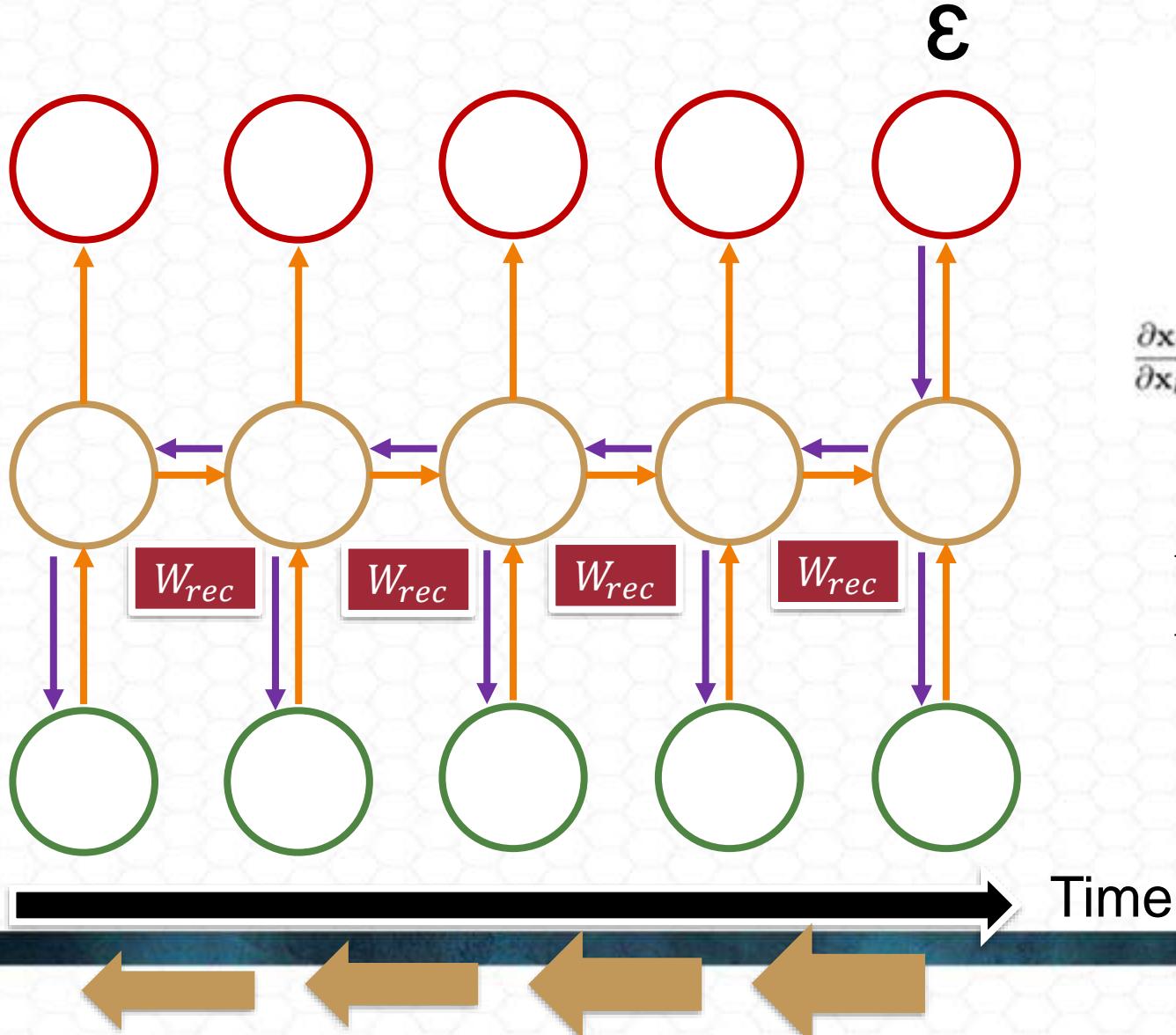


梯度消失的問題

長期依賴問題 (Long Term Dependencies)

- 會議中，董事長問總經理，要怎麼把進度落後的產品如期上線完成？
- 總經理回答：要10個工程師每週上班7天，每天工作16小時，持續半年，還要吊死工友養的流浪狗小白。
- 董事長驚問：為什麼要吊死小白????
- 總經理回頭對技術長說：你們看吧 !! 就跟你說沒有人會在乎工程師的死活的.....

長期依賴會導致梯度消失或梯度爆炸



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

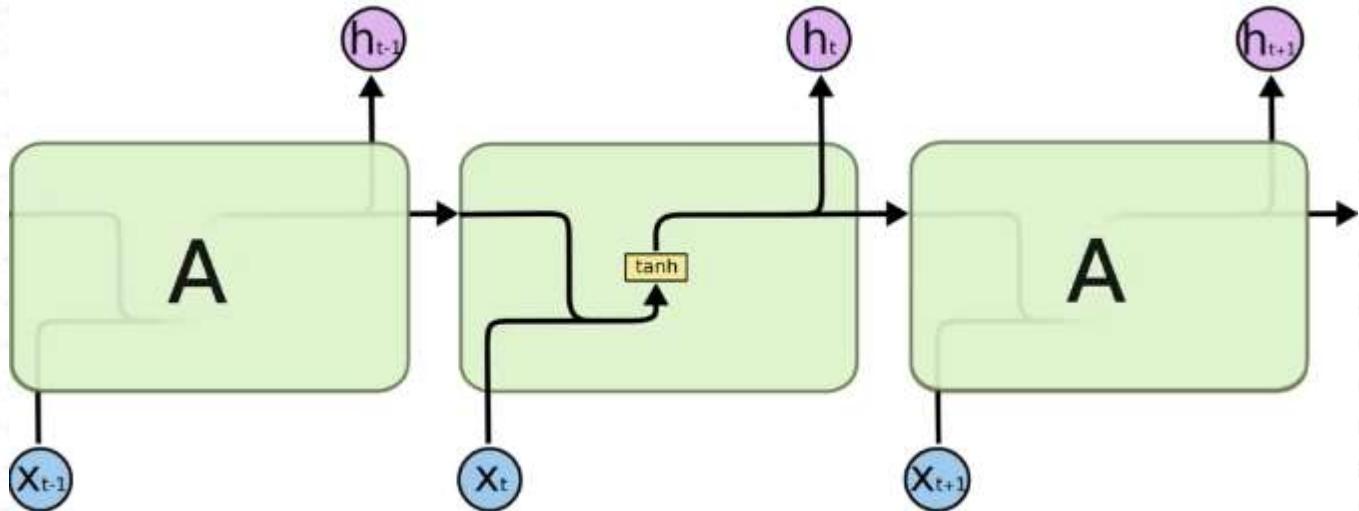
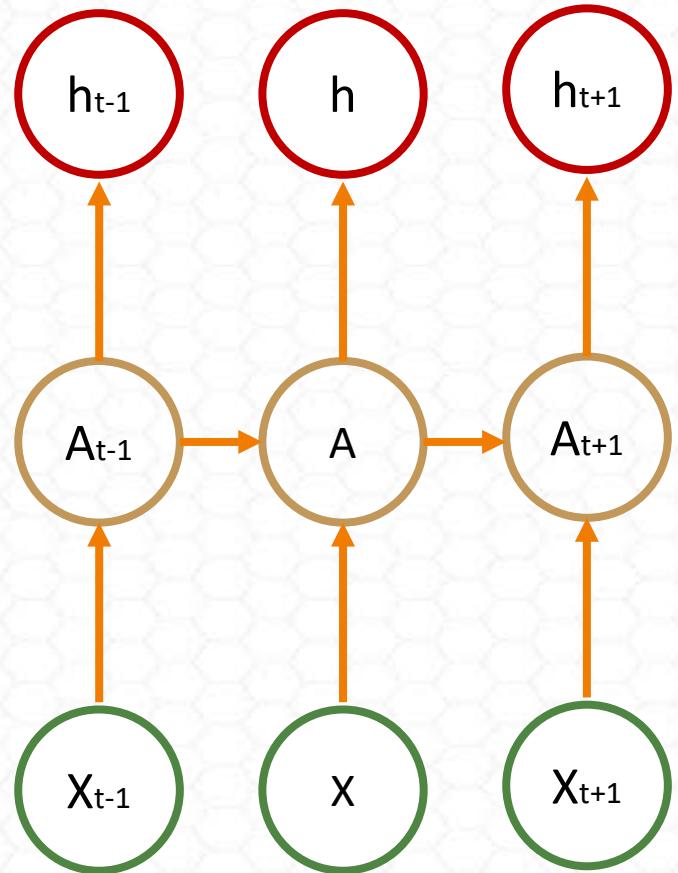
$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

$W_{rec} \sim small$ 梯度消失
 $W_{rec} \sim large$ 梯度爆炸

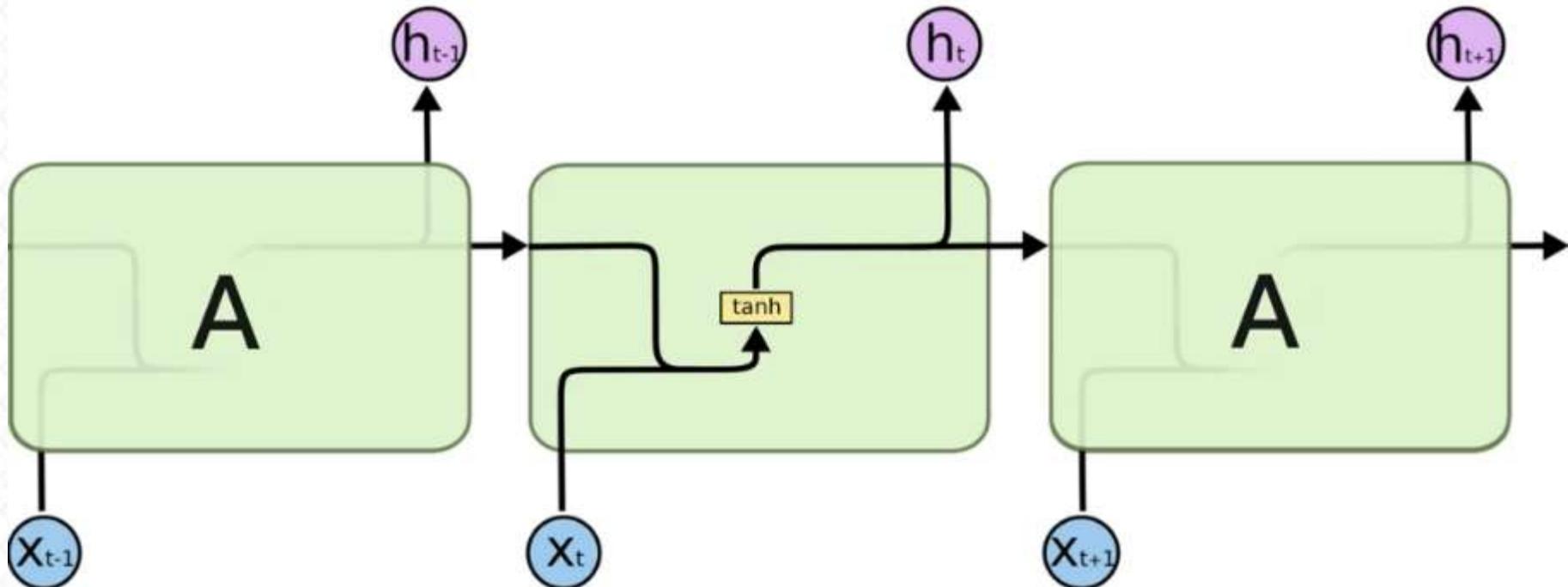
長短期記憶網路 LSTM

遞迴神經網路

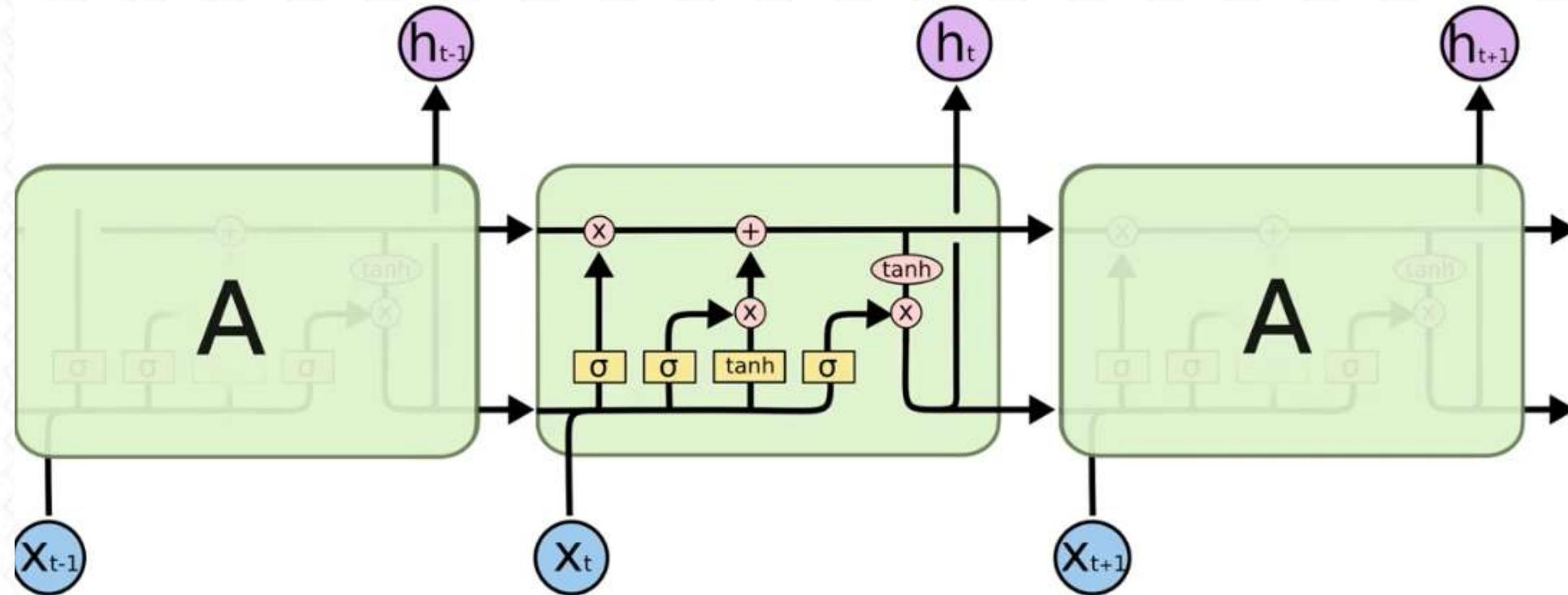


tanh的二階導數會讓梯度能撐過長時間的傳遞才接近0

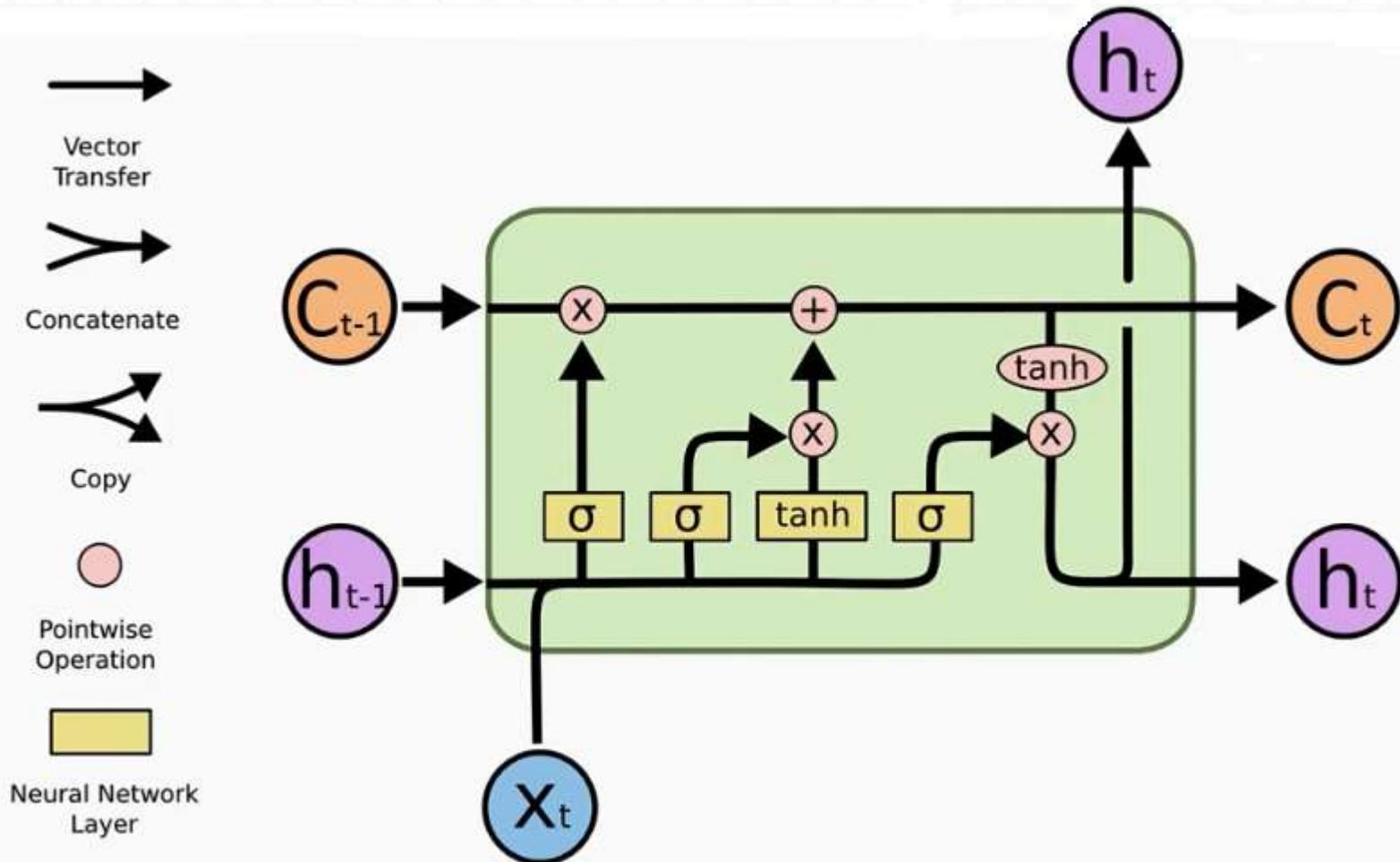
遞迴神經網路



長短期記憶網路 (Long Short-Term Memory)

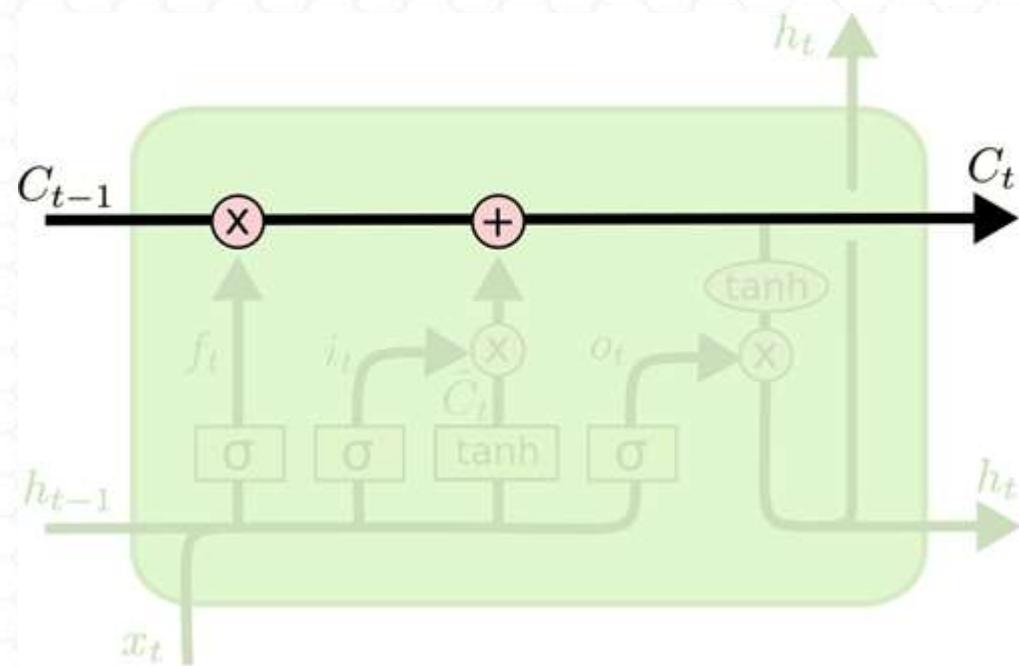


長短期記憶網路 (Long Short-Term Memory)

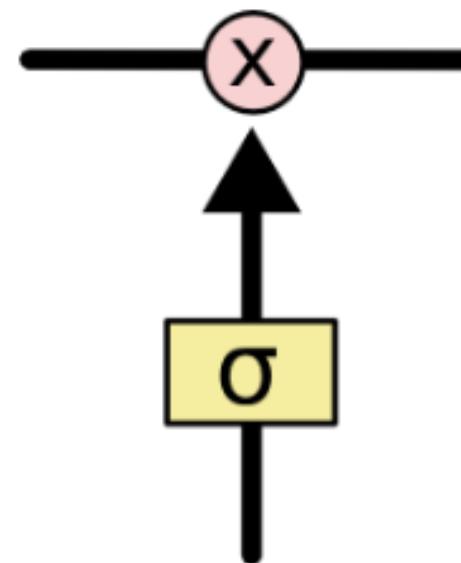


長短期記憶網路 LSTM 詳解

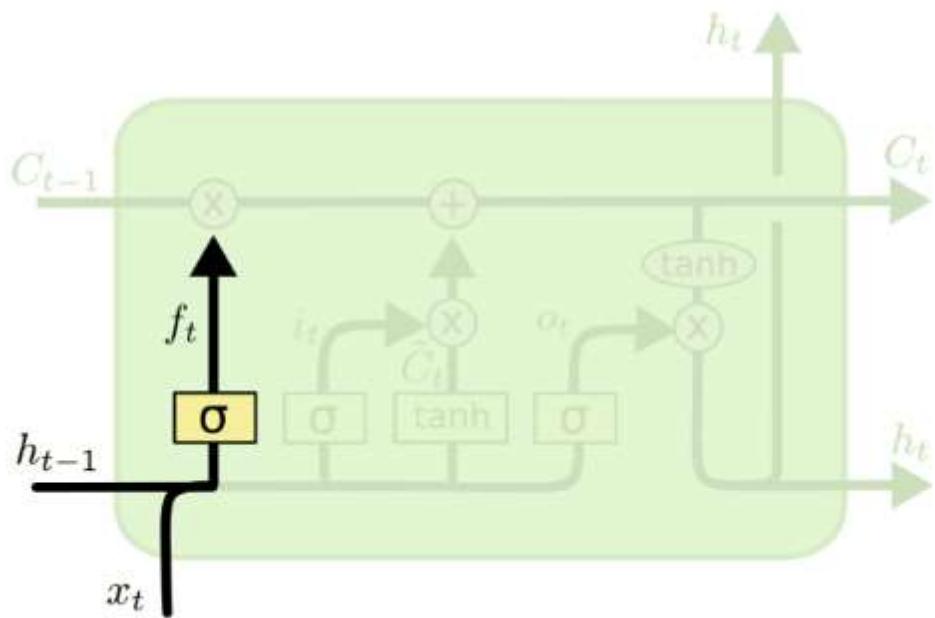
Cell State 傳遞過程



Sigmoid 閘門

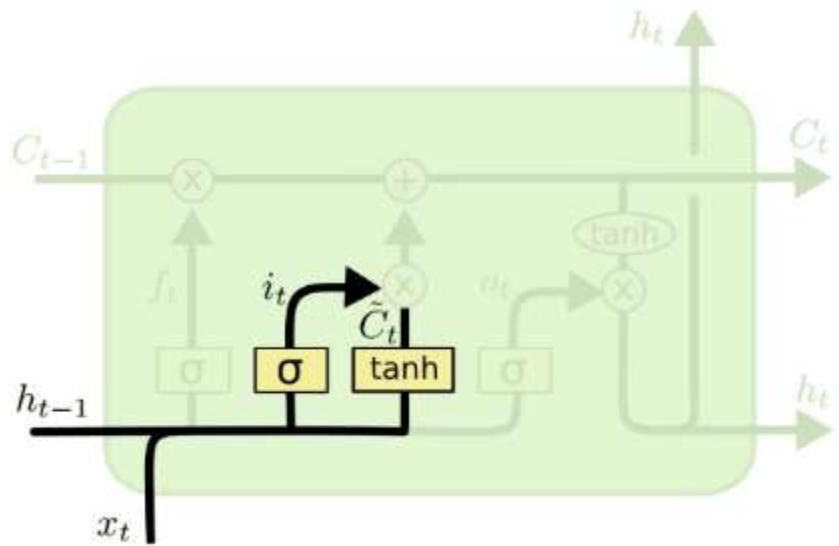


遺忘門 (Forget Gate)



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

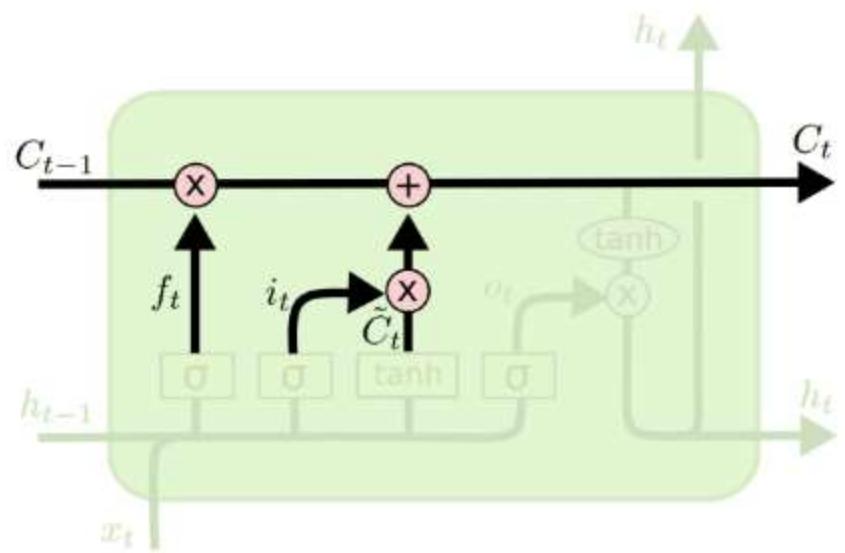
輸入門 (Input Gate)



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

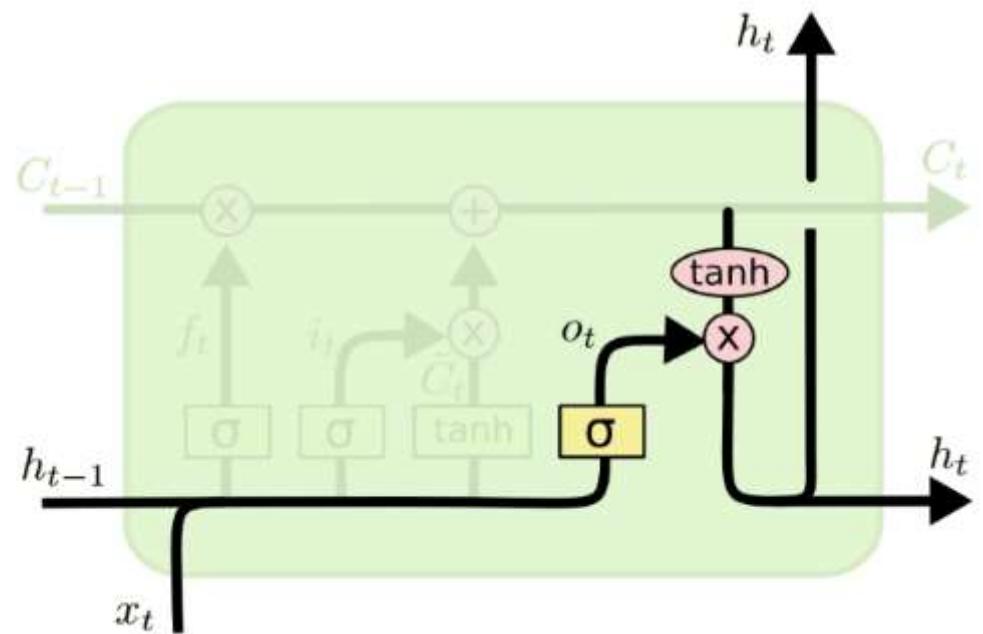
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

更新細胞狀態 (Cell State)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

輸出門 (Output Gate)



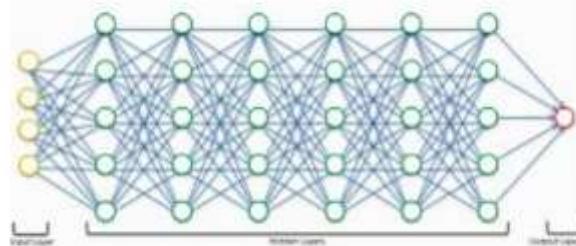
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

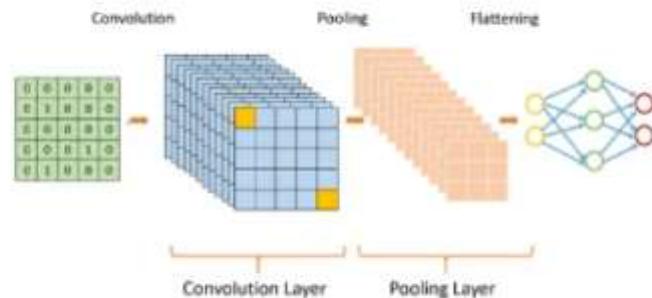
什麼是自編碼網路

神經網路的種類

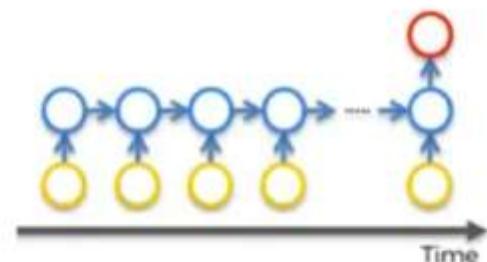
人工神经网络



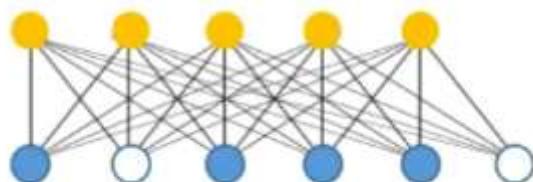
卷积神经网络



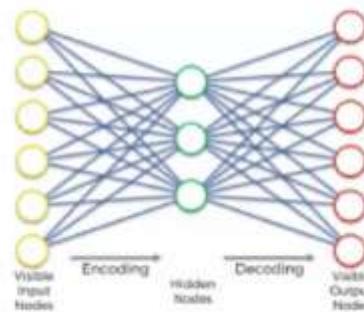
递归神经网络



限制波兹曼机

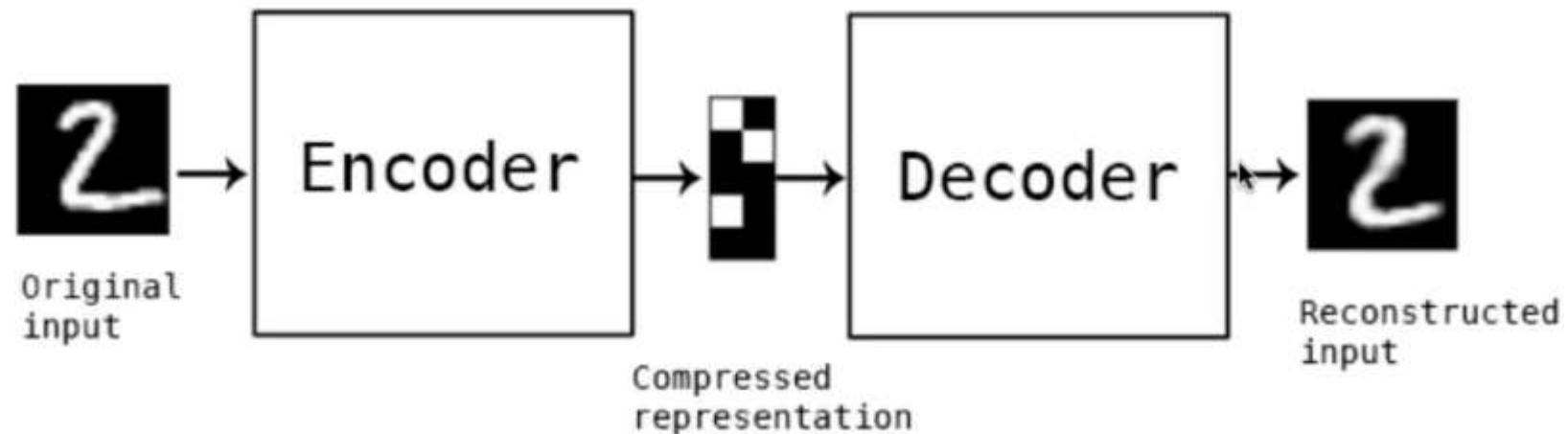


自编码网络

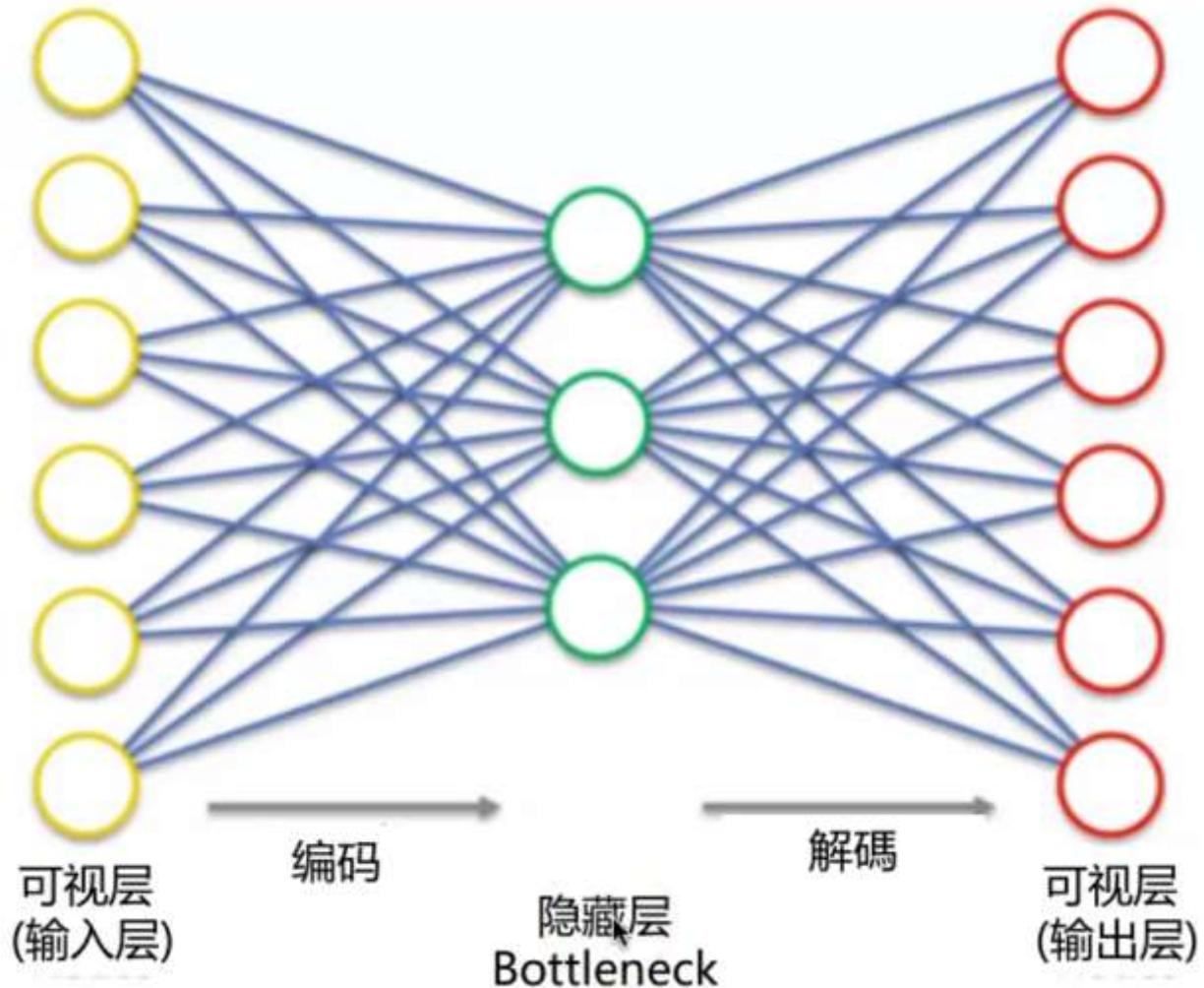


什麼是自編碼器

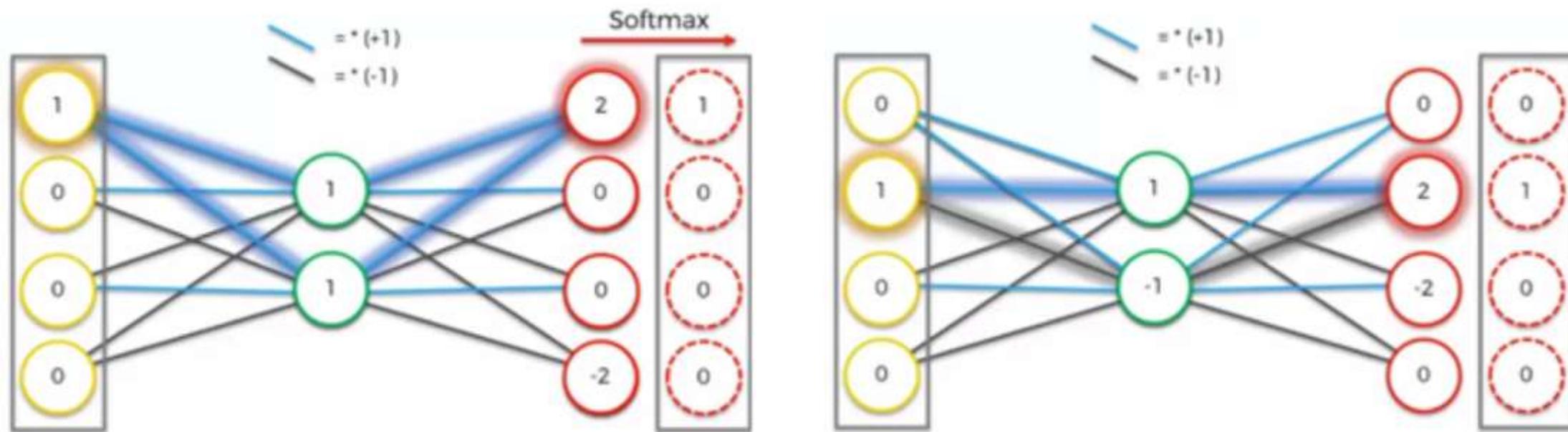
- 自编码器是一种压缩(compression)算法，包含以下特性：
 - 数据相关 (Data-specific)
 - 有损的 (Lossy)
 - 从样本中自动学习(Learned automatically from examples)



Autoencoders

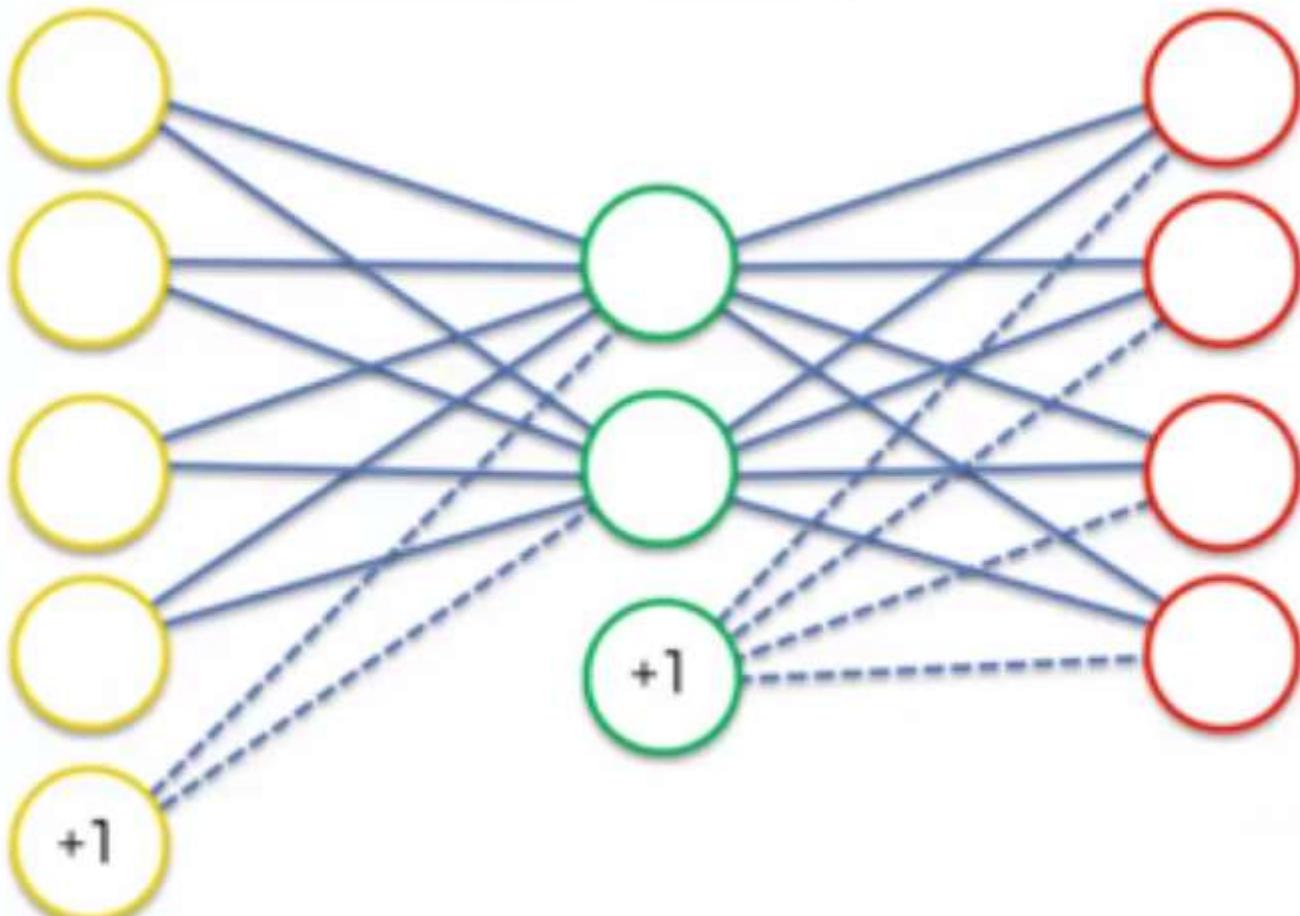


Autoencoders



增加偏倚 (Bias)

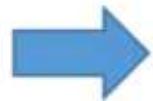
$$z = f(Wx + b)$$



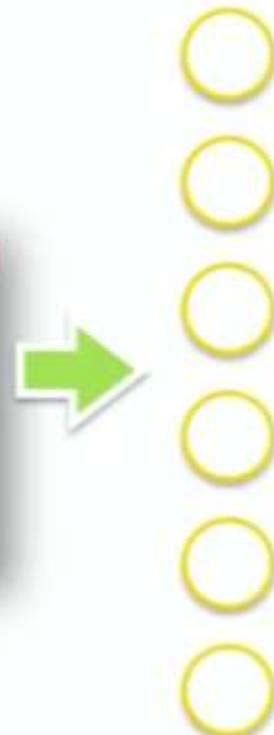
訓練自編碼網路

訓練自編碼器 (Autoencoders)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	1	0		1	1	1
User 2	0	1	0	0	1	0
User 3	1	1	0		0	
User 4	1	0	1	1	0	1
User 5	0		1	1		1
User 6	0	0	0	0	1	
User 7	1	0	1	1	0	1
User 8	0	1	1		0	1
User 9	0	1	1	1	1	
User 10	1		0	0		0
User 11	0	1	1	1	0	1



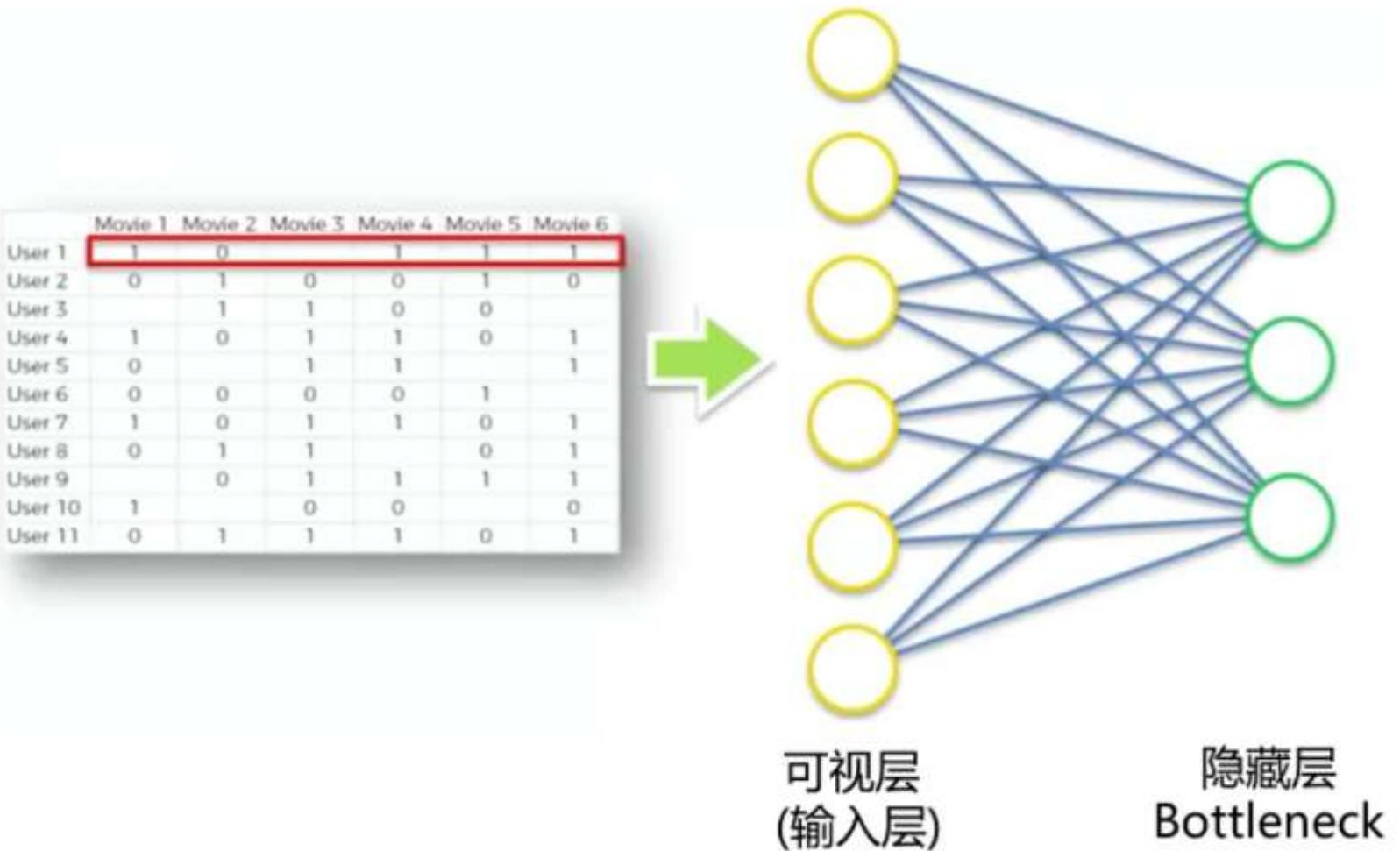
	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	1	0		1	1	1
User 2	0	1	0	0	1	0
User 3	1	1	0	0	0	
User 4	1	0	1	1	0	1
User 5	0		1	1		1
User 6	0	0	0	0	1	
User 7	1	0	1	1	0	1
User 8	0	1	1		0	1
User 9	0	1	1	1	1	
User 10	1		0	0		0
User 11	0	1	1	1	0	1



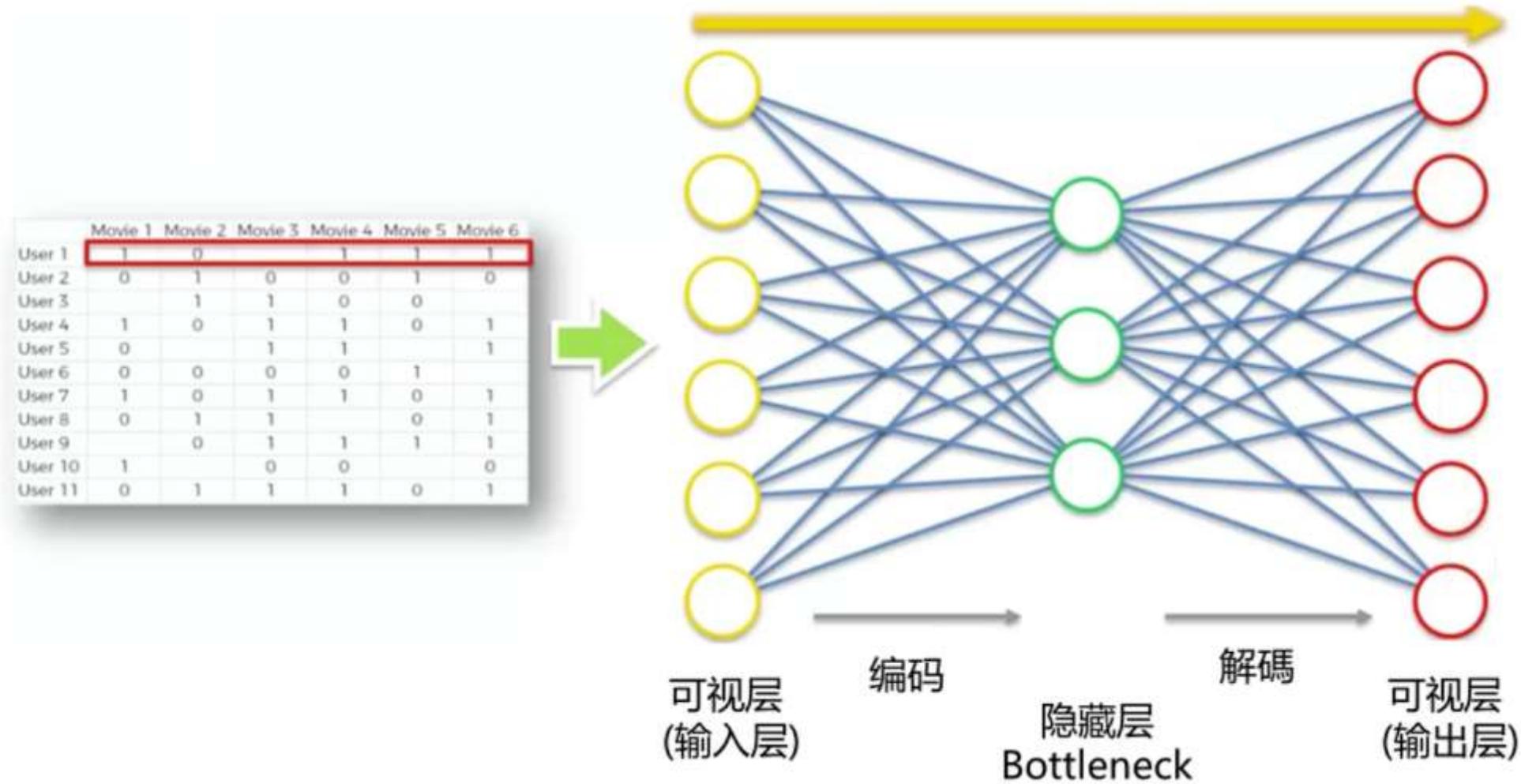
可视层
(输入层)



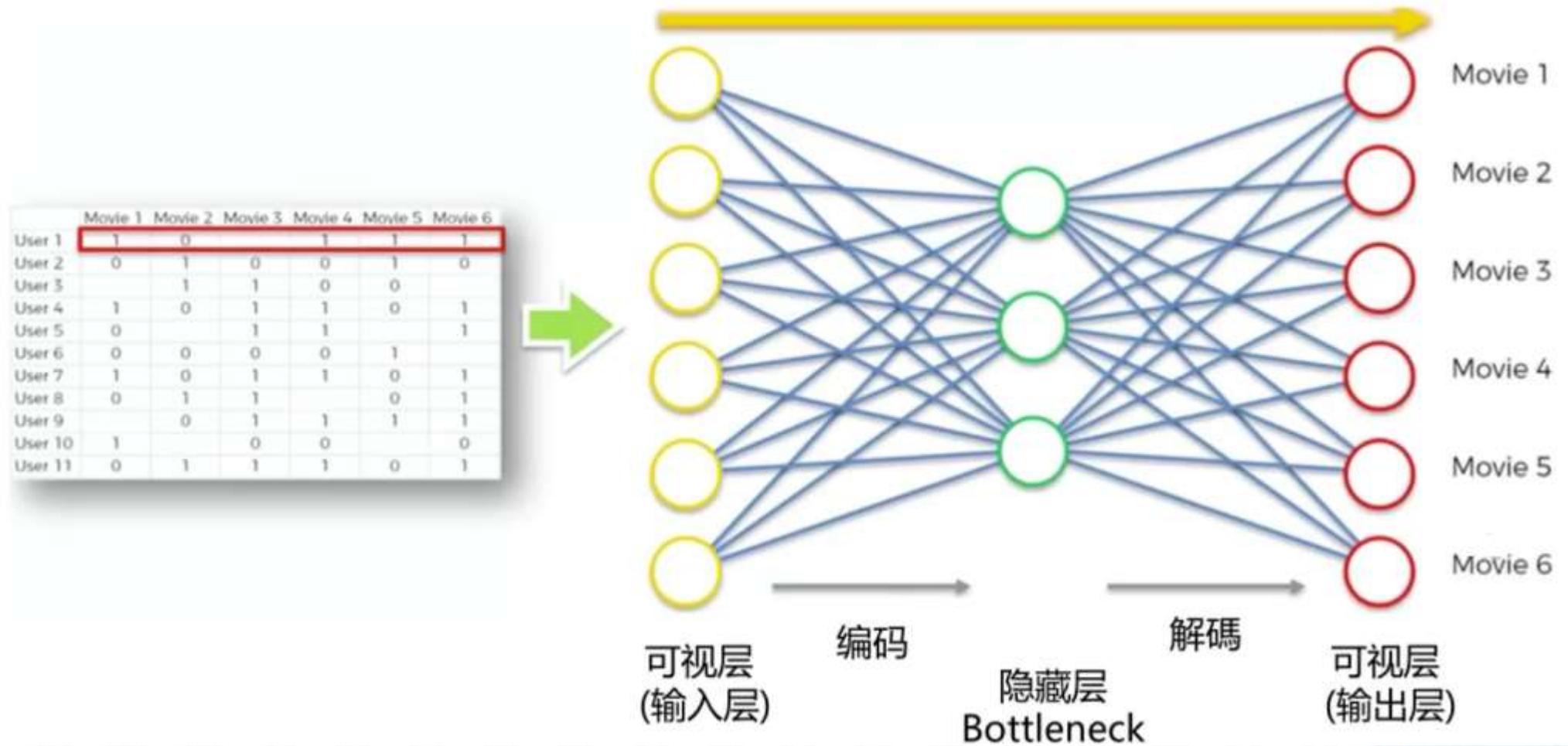
訓練自編碼器 (Autoencoders)



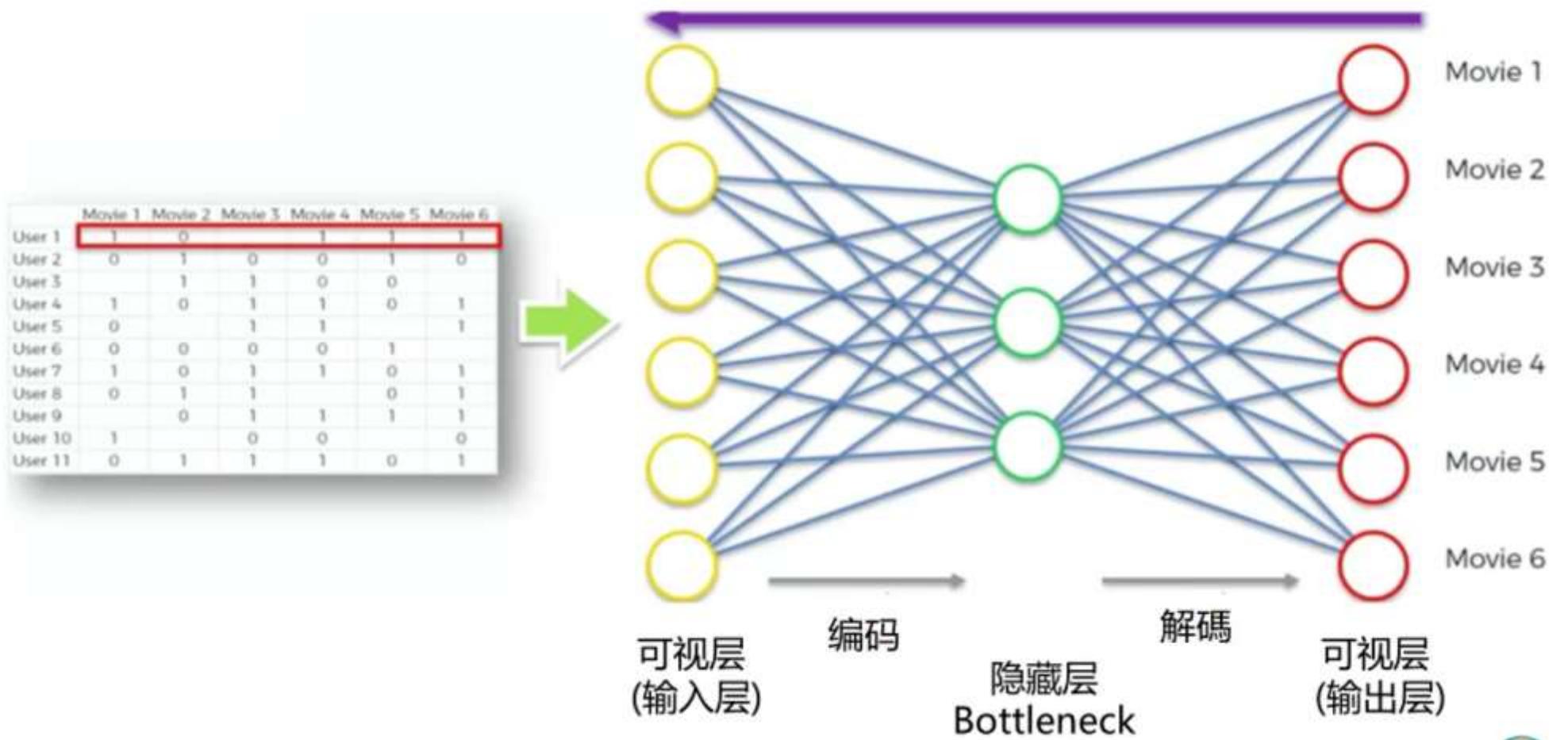
訓練自編碼器 (Autoencoders)



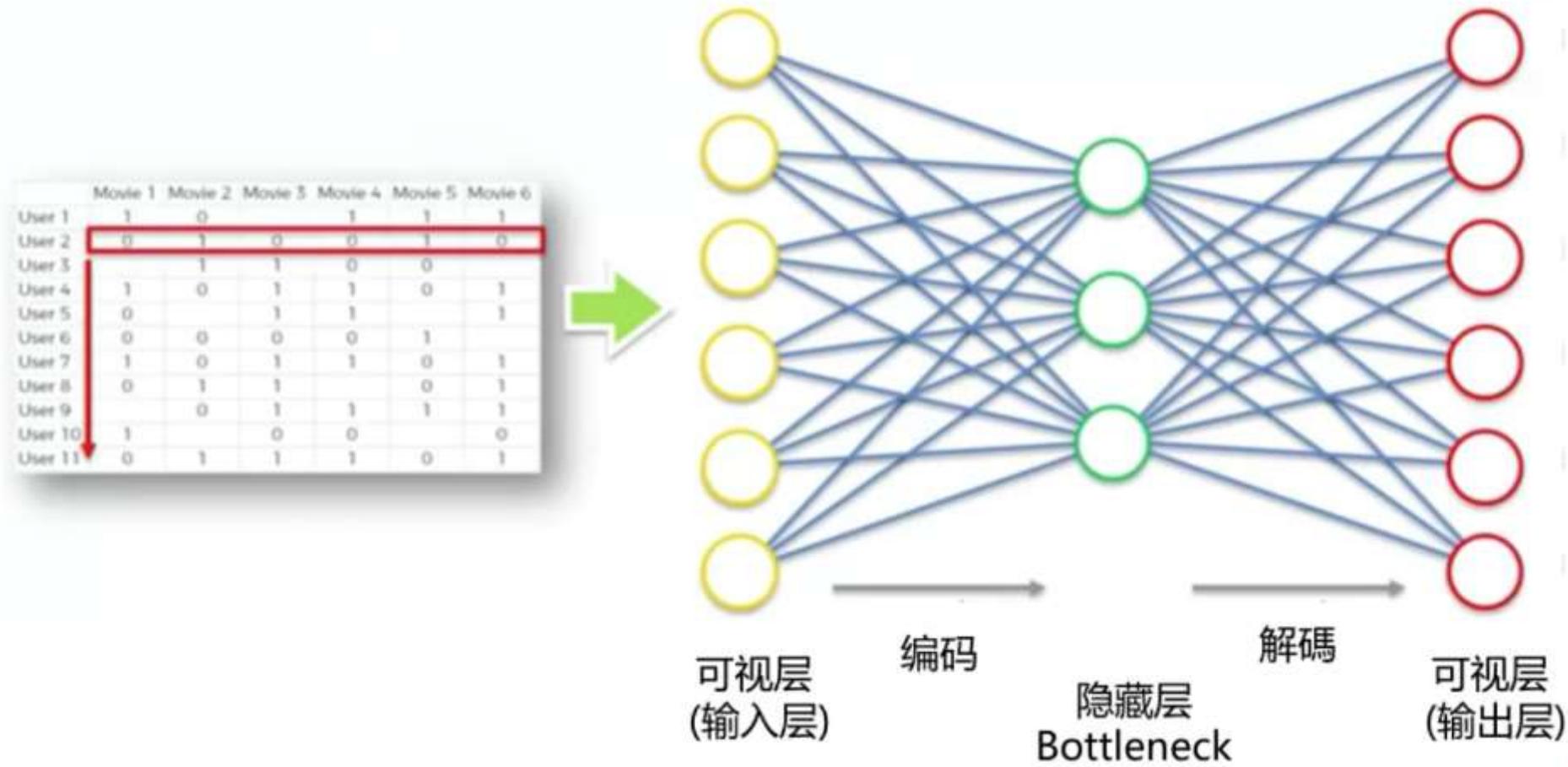
訓練自編碼器 (Autoencoders)



訓練自編碼器 (Autoencoders)

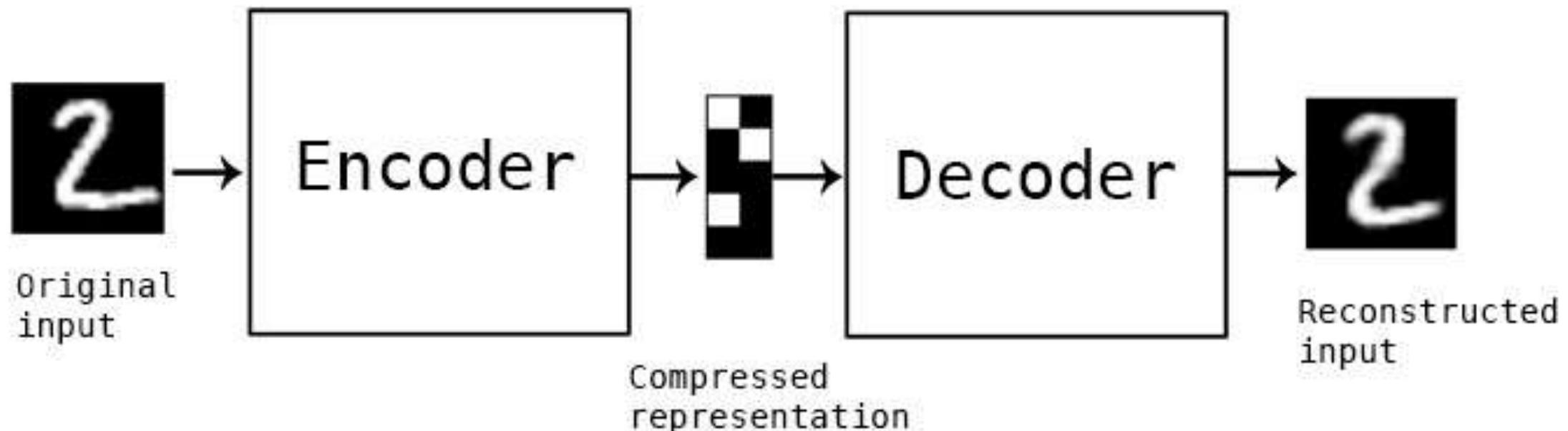


訓練自編碼器 (Autoencoders)



利用自編碼網路重建影像

利用自編碼網路重建影像



什麼是生成對抗網路

男女朋友間的GAN 話



男：你看這張拍的好不好？
女：這是在拍誰，是拍我還是在拍景物？
男：哦

.....

男：那這張呢？
女：不行，拍得太胖了
男：哦

.....

男：這張總算好點了吧？
女：角度再高一點，看起來比較好看

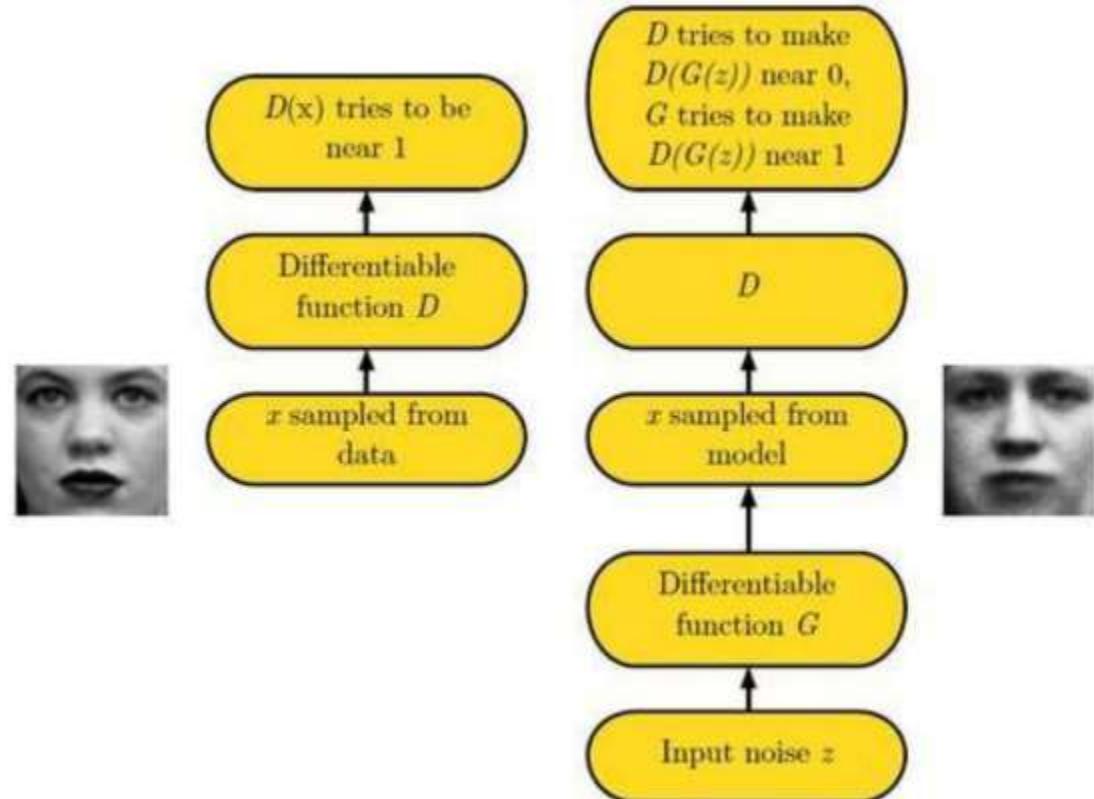
.....

男：這樣呢？
女：Good 我待會發IG

Generative Adversarial Network (GAN)

■ Generator和Discriminator

- Generator 是一個生成圖片的網路，它接收一個隨機的雜訊 z ，通過這個雜訊生成圖片，記做 $G(z)$
- Discriminator 是一個判別網路，判別一張圖片是不是**真實的**。它的輸入參數是 x ， x 代表一張圖片，輸出 $D(x)$ 代表 x 為真實圖片的概率，如果為1，就代表100%是真實的圖片，而輸出為0，就代表不可能是真實的圖片
- 在訓練過程中，Generator的目標就是儘量生成真實的圖片去欺騙Discriminator。而Discriminator的目標就是儘量把Generator生成的圖片和真實的圖片分別開來



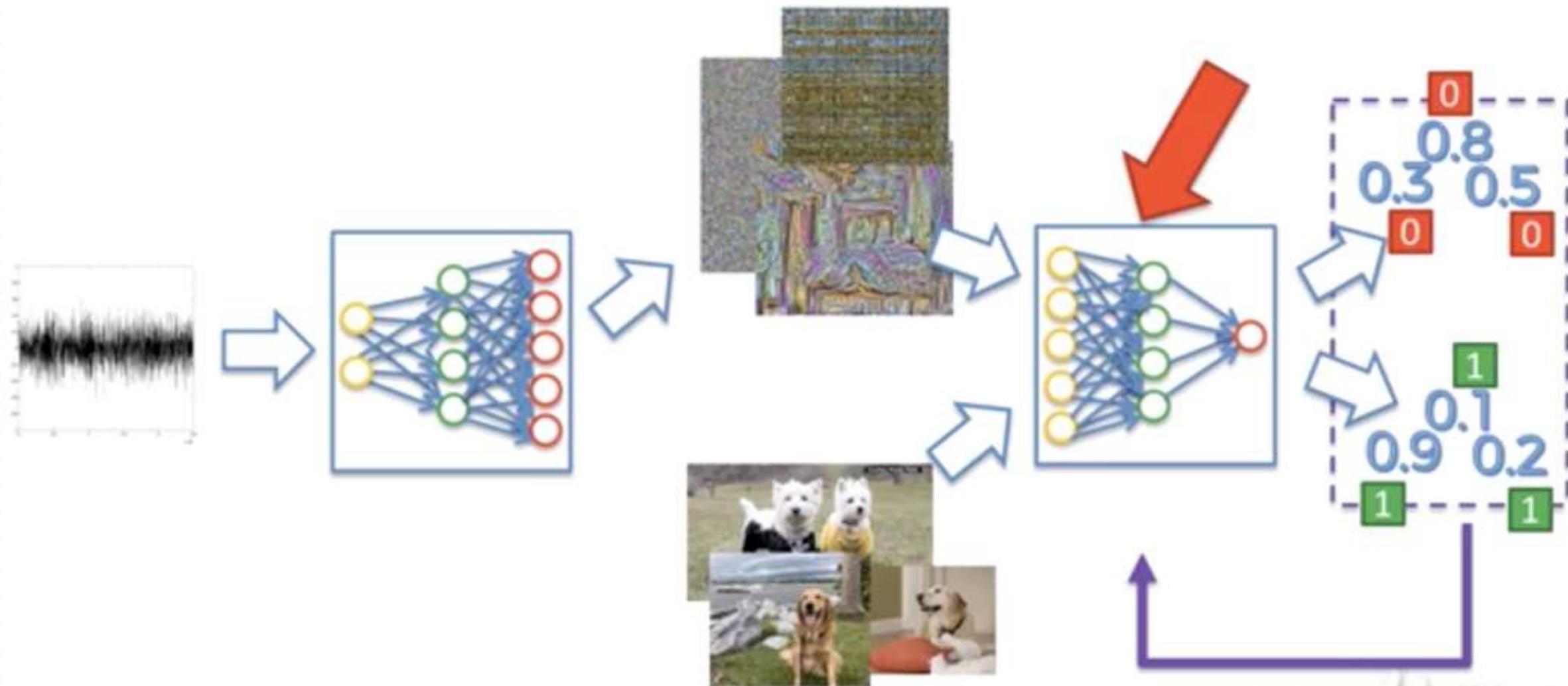
(Goodfellow 2016)



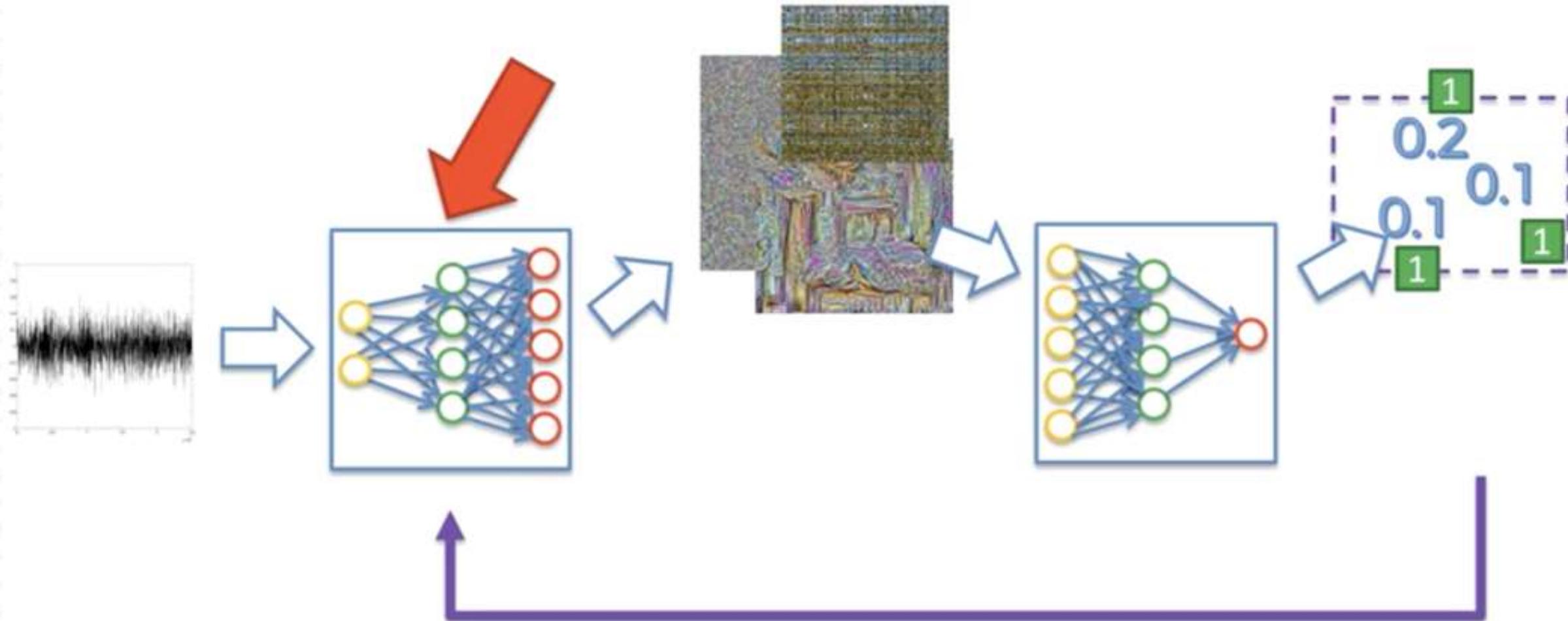
Ian Goodfellow
Google Brain

GAN 的訓練過程

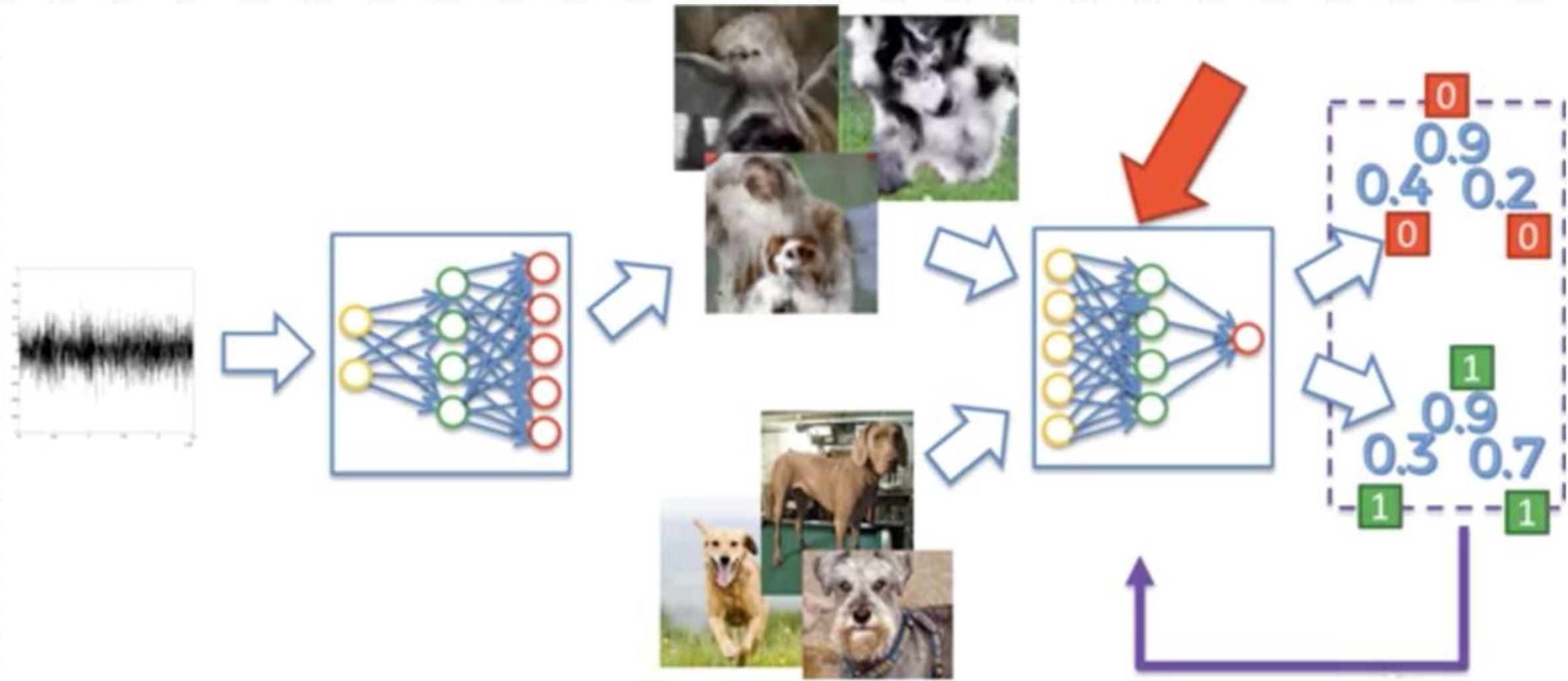
第一步: 訓練 Discriminator



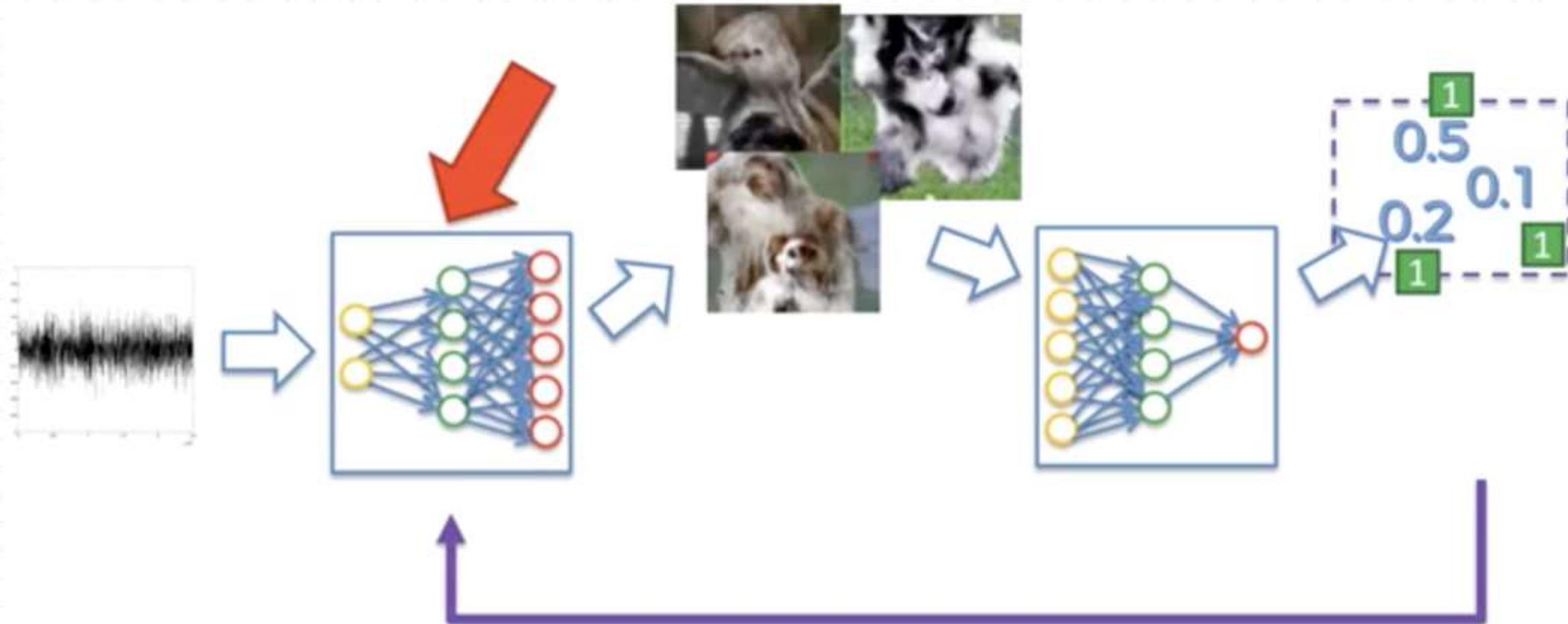
第一步: 訓練 Generator



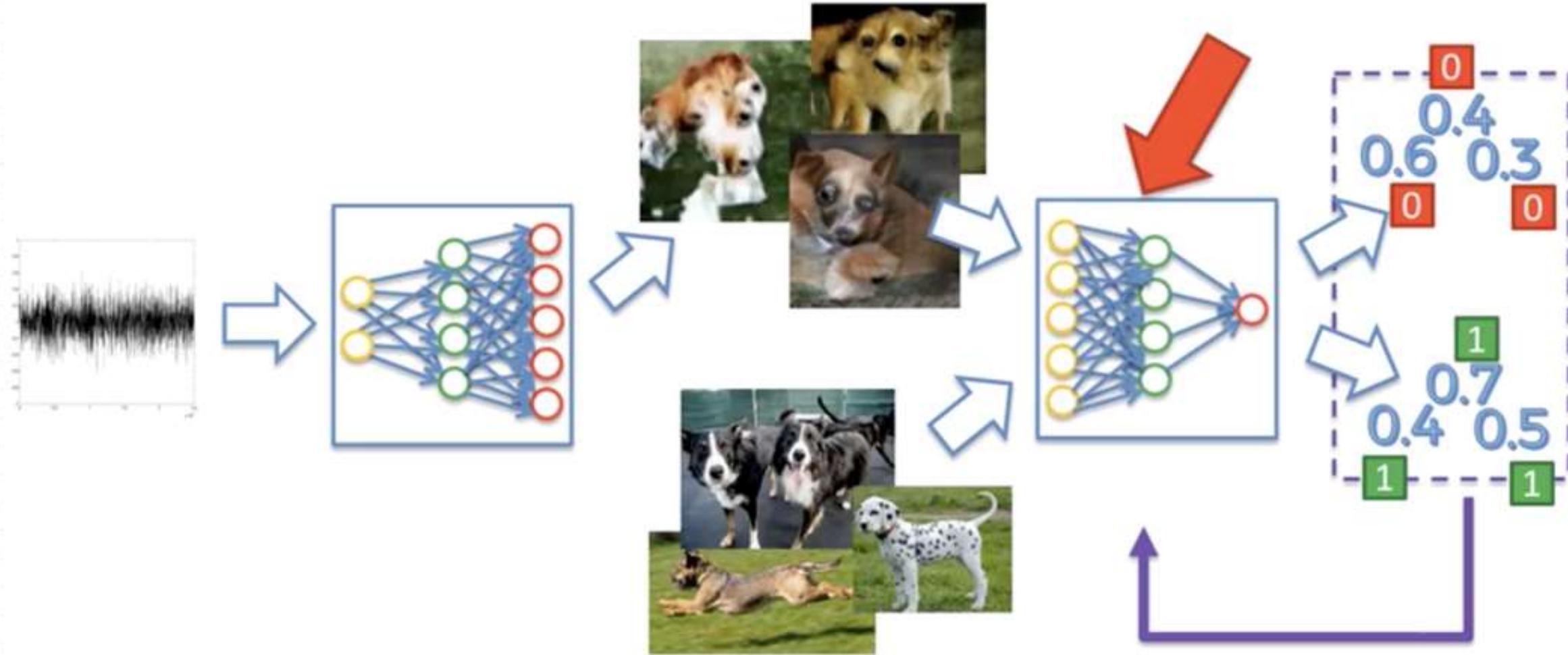
第二步: 訓練 Discriminator



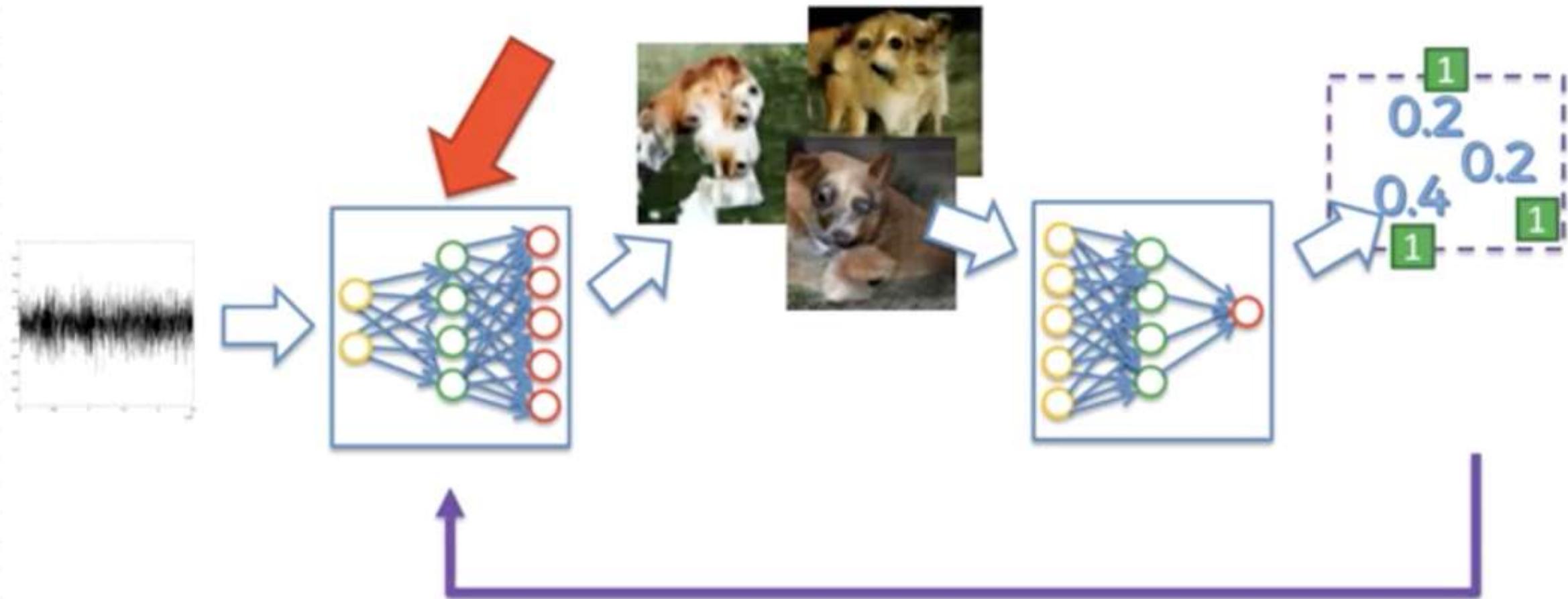
第二步: 訓練 Generator



第三步: 訓練 Discriminator



第三步: 訓練 Generator



GAN 的特性與優缺點

零和遊戲

■ 價值函數

$$V(\theta^{(G)}, \theta^{(D)}) = E_{x \sim P_{data}} \log(D(x)) + E_{x \sim P_{model}} \log(1 - D(x))$$

G 生成器

D 鑒別器

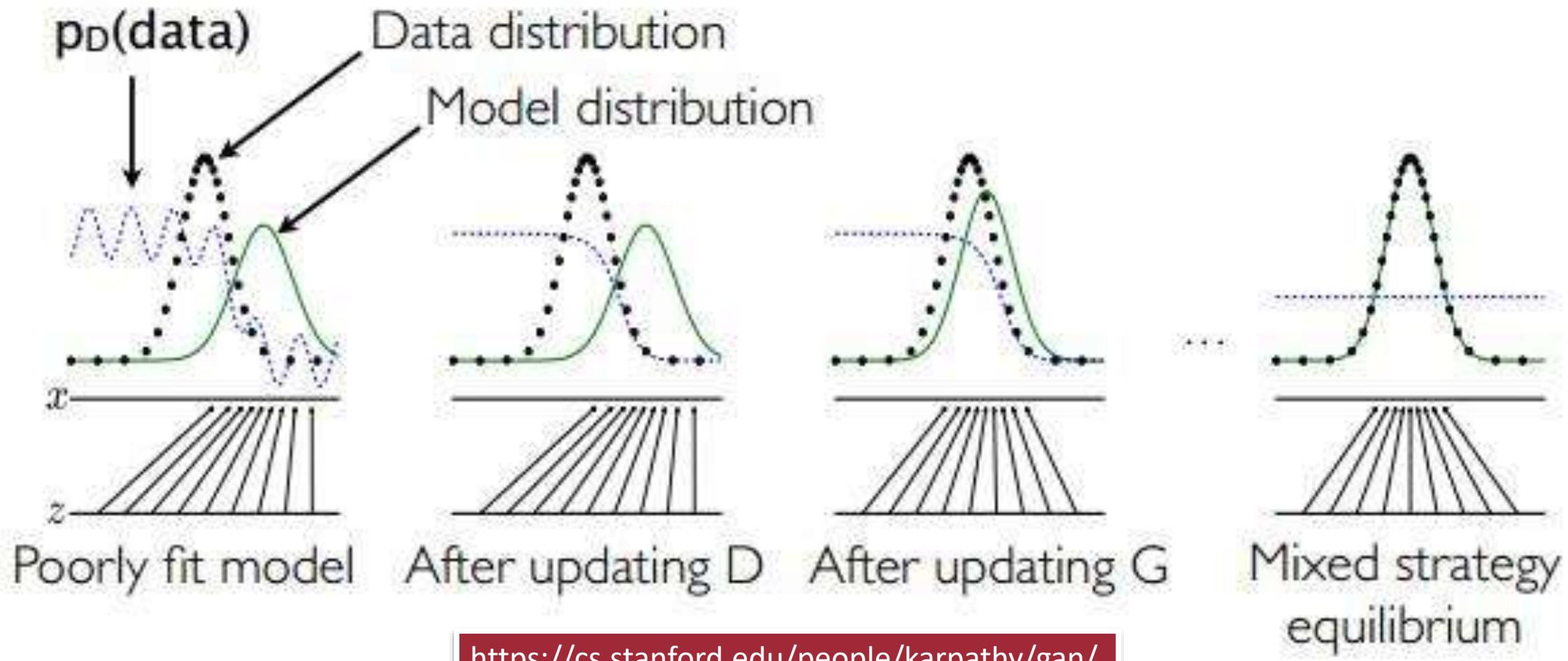
P_{data} 真實樣本分佈

P_{model} 生成樣本分佈

■ 優化價值函數

$$\arg \min_G \max_D (V(\theta^{(G)}, \theta^{(D)}))$$

GAN 生成概念



<https://cs.stanford.edu/people/karpathy/gan/>

GAN 的特性

- 相比較傳統的模型，他存在兩個不同的網路，而不是單一的網路，並且訓練方式採用的是對抗訓練方式
- GAN中G的梯度更新資訊來自判別器D，而不是來自資料樣本

GAN 的優點

- GAN是一種生成式模型，相比較其他生成模型(如玻茲曼機) 只用到了反向傳播,而不需要複雜的馬可夫鏈(Markov Chain)
- GAN採用的是一種無監督的學習方式訓練，可以被廣泛用在無監督學習和半監督學習領域
- GANs可以產生更加清晰，真實的樣本，而且生成的圖片是一致的,但是VAE是有偏差的
- GAN 不用設計損失函數，只要有一個的基準，便可交給對抗訓練了

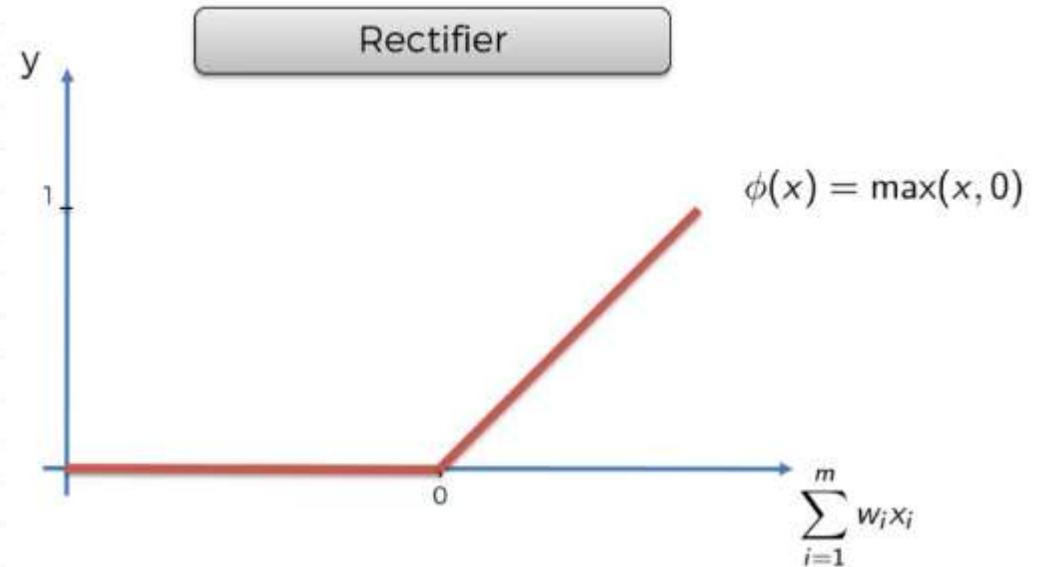
GAN 的缺點

- 訓練GAN需要達到Nash Equilibrium,有時候可以用梯度下降法做到,但有時候做不到,所以訓練GAN相比VAE是不穩定的
- GAN不適合處理離散形式的資料，比如文本資料

LEAKY RELU

ReLU

```
def relu_function(x):  
    return np.maximum(0,x)  
  
x = np.array([-1,1,2])  
relu_function(x)
```



優點:

- 快速收斂
- 解決梯度消失問題

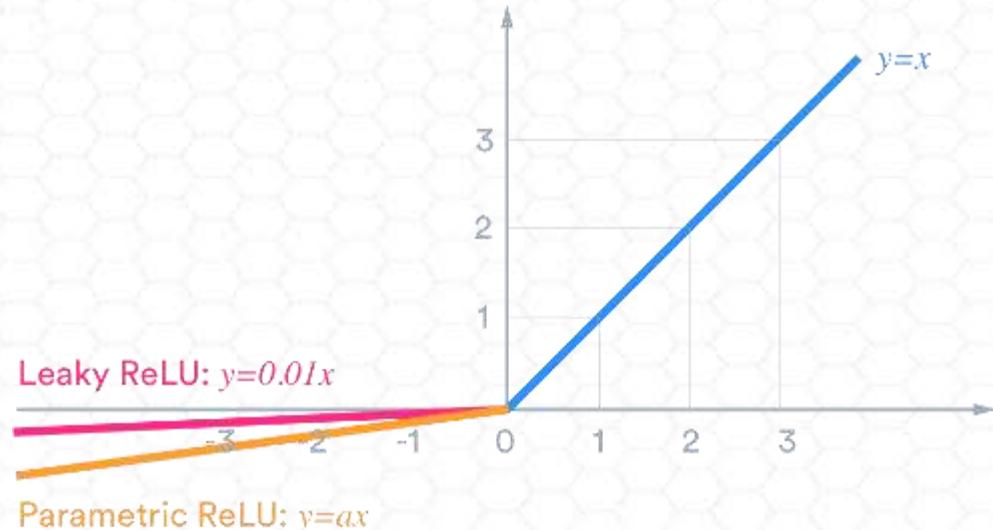
缺點:

- 神經元死亡問題

Leaky ReLU

```
def leaky_relu(z, alpha=0.01):  
    return np.maximum(alpha*z, z)
```

```
x = np.array([-1,1,2])  
leaky_relu(x)
```

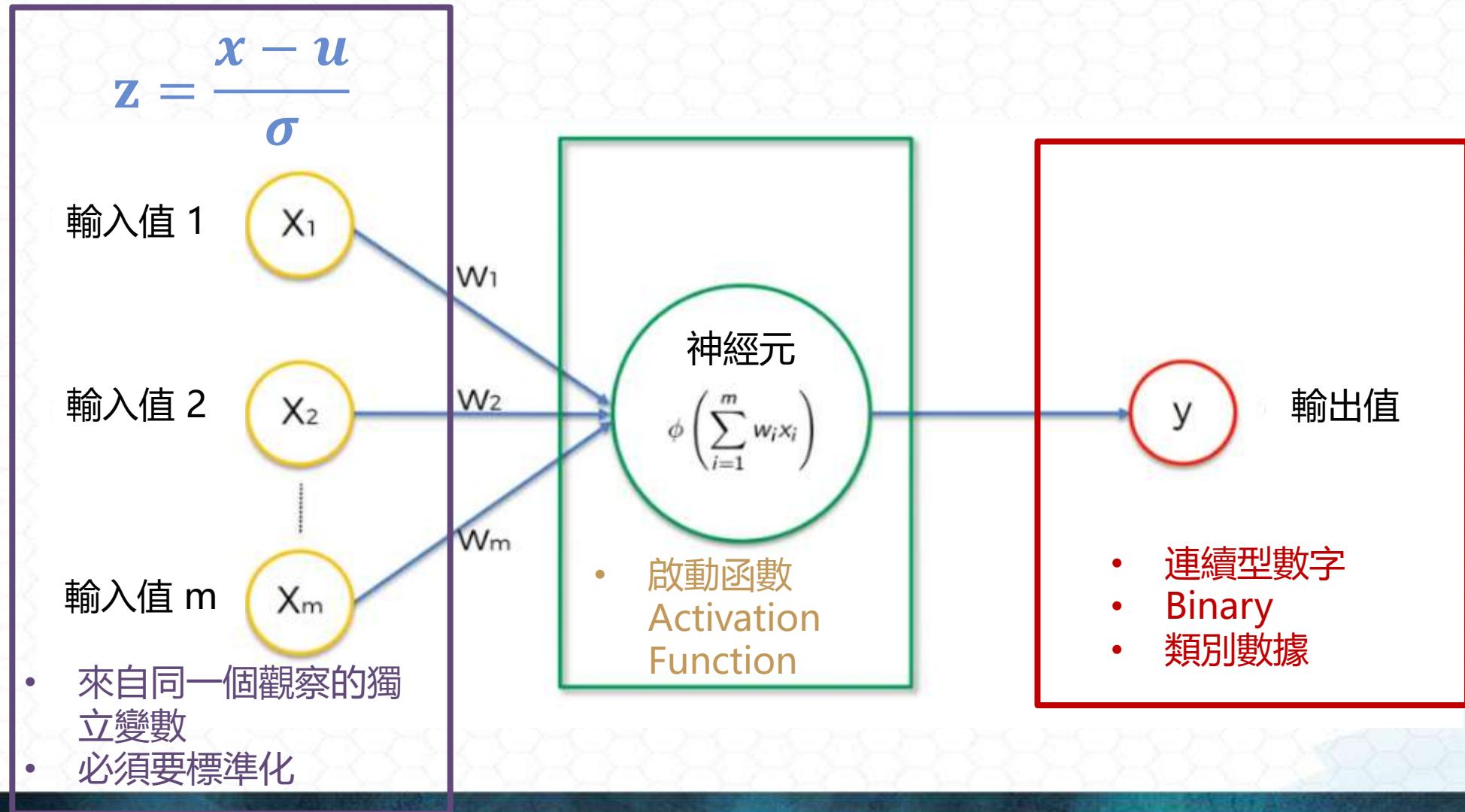


優點:

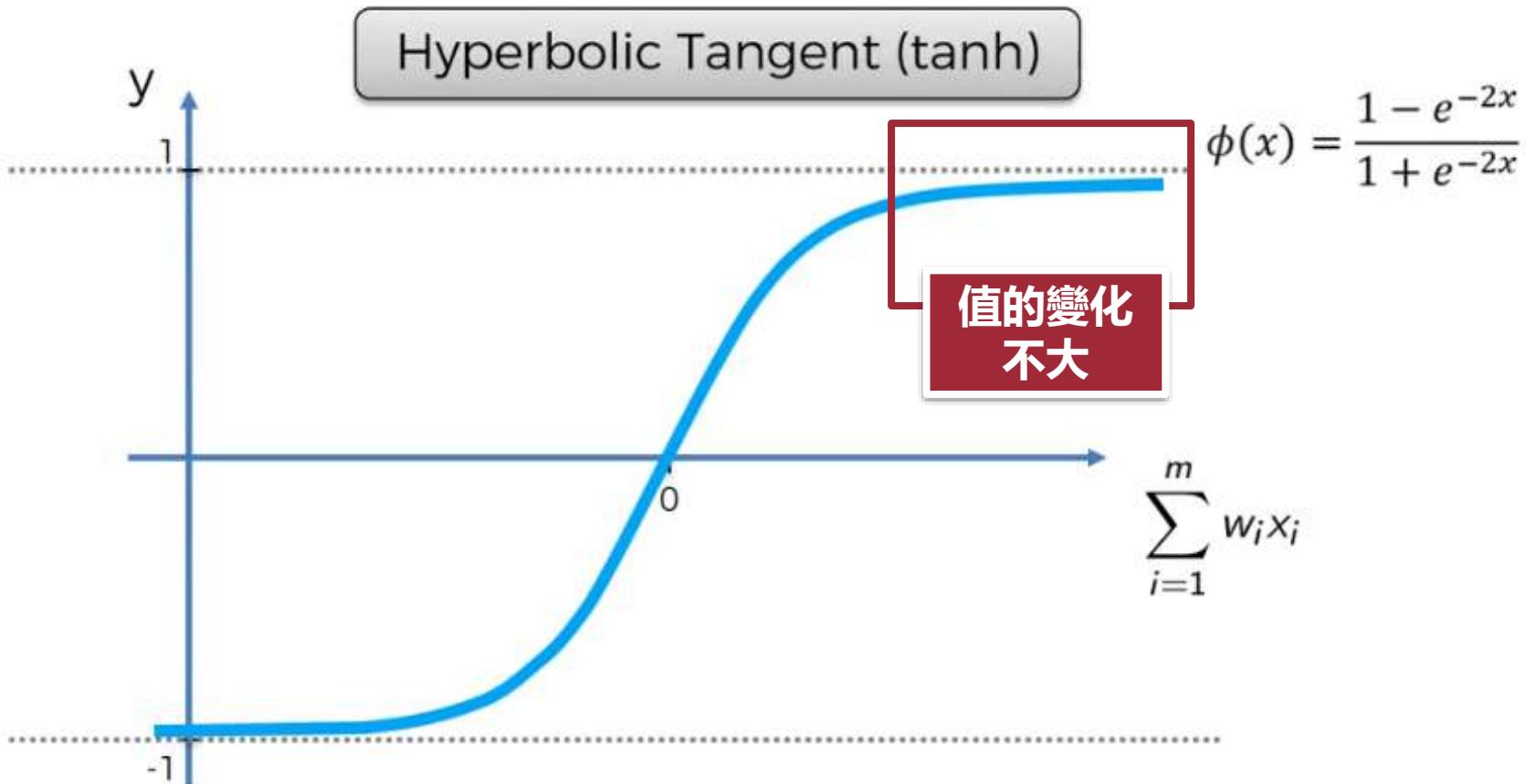
- 快速收斂
- 解決梯度消失問題
- 解決神經元死亡問題

批標準化

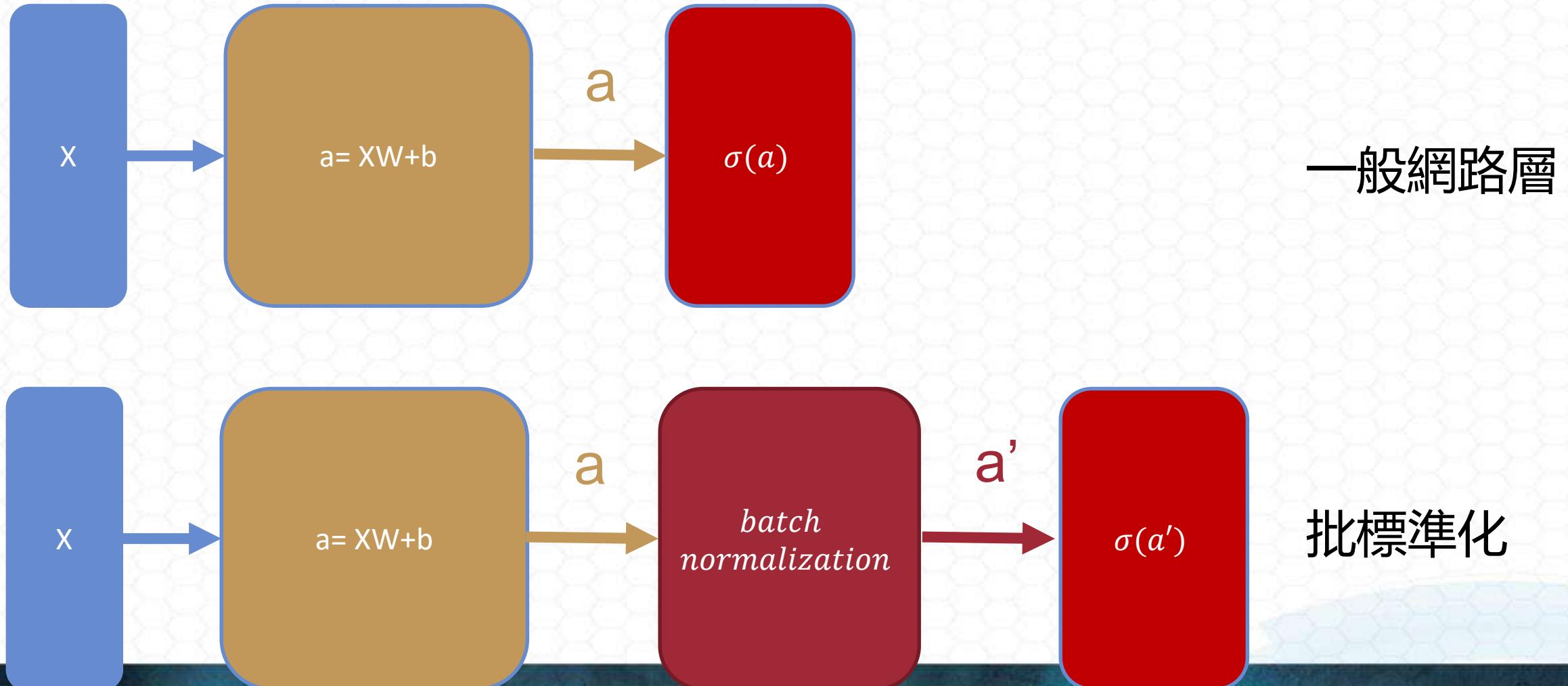
標準化



飽和區域 (Saturation Region)



批標準化 (Batch Normalization)



反標準化

$$\blacksquare \hat{X}_{Batch} = \frac{X_{Batch} - \mu_{Batch}}{\sigma_{Batch}}$$

□ 其中 μ_{Batch} 代表平均數

□ σ_{Batch} 代表標準差

$$Y = \gamma \hat{X}_{Batch} + \beta$$

處理測試資料

- 測試資料集沒有批(Batch)的概念
- 解決方案
 - 使用所有訓練資料的平均與標準差 (數據太大時不可行)
 - 使用在訓練階段時的移動平均 σ 與移動平均 μ
- 對每個批次(Batch) 做指數移動平滑
- $\mu = \text{decay} * \mu + (1 - \text{decay}) * \mu_{Batch}$
- $\sigma = \text{decay} * \sigma + (1 - \text{decay}) * \sigma_{Batch}$

批標準化的優點

- 避免梯度消失或爆炸

- 尤其是受到sigmoid 或 tanh 影響的資料

- 減少資料初始化的影響

- 逐層標準化

- 大幅減少訓練時間

- 可以使用較大的學習率

GAN 的應用

圖片生成 (Image Generation)

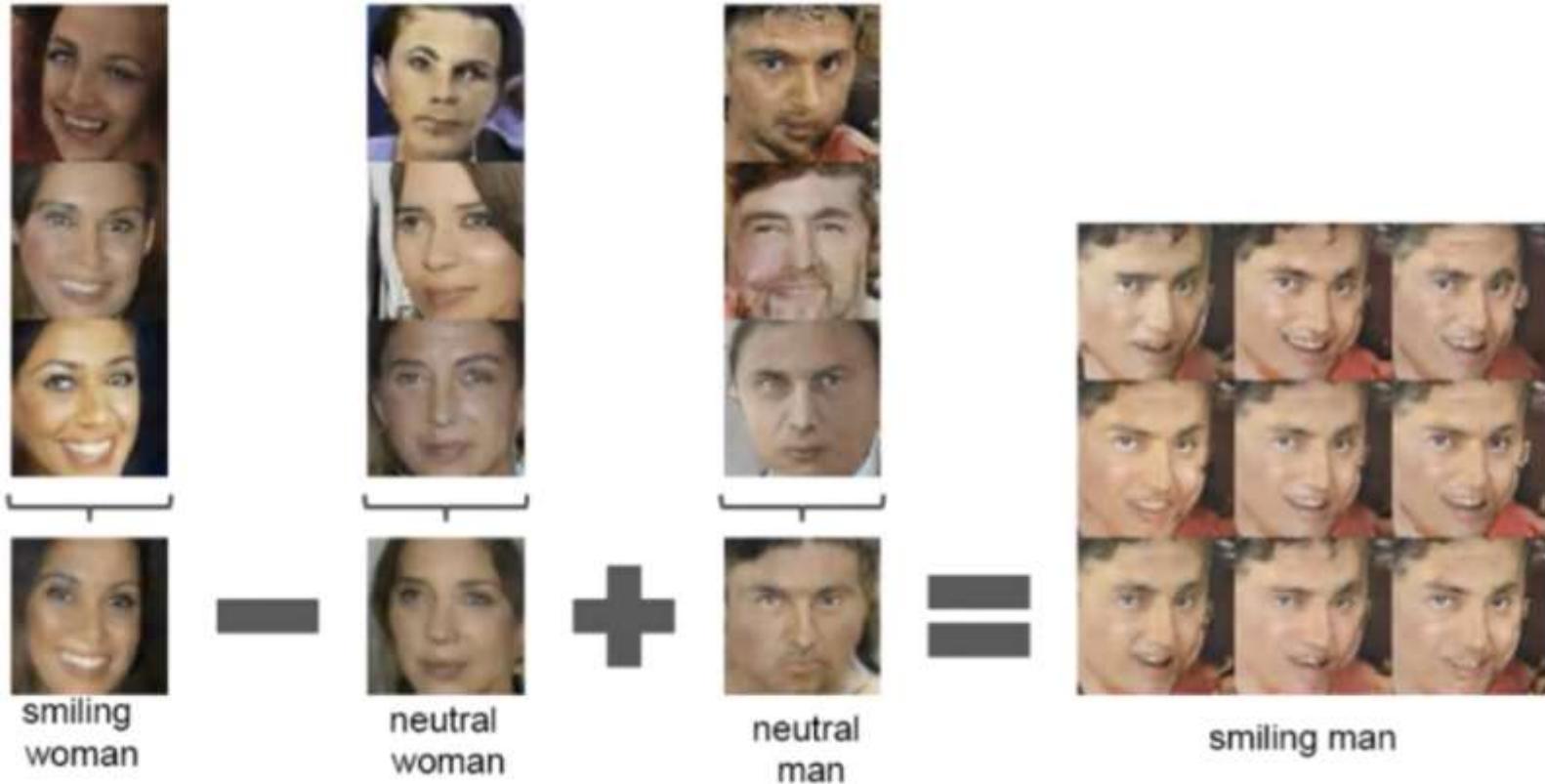
原圖片



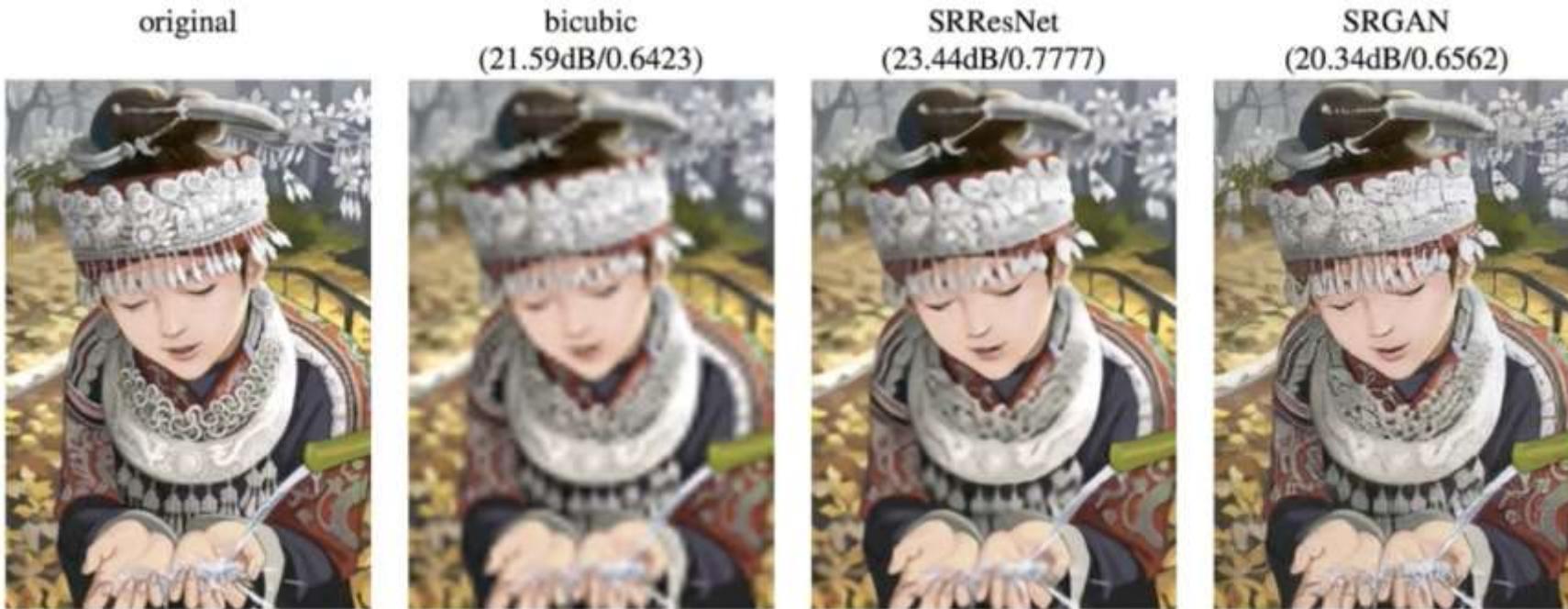
生成圖片



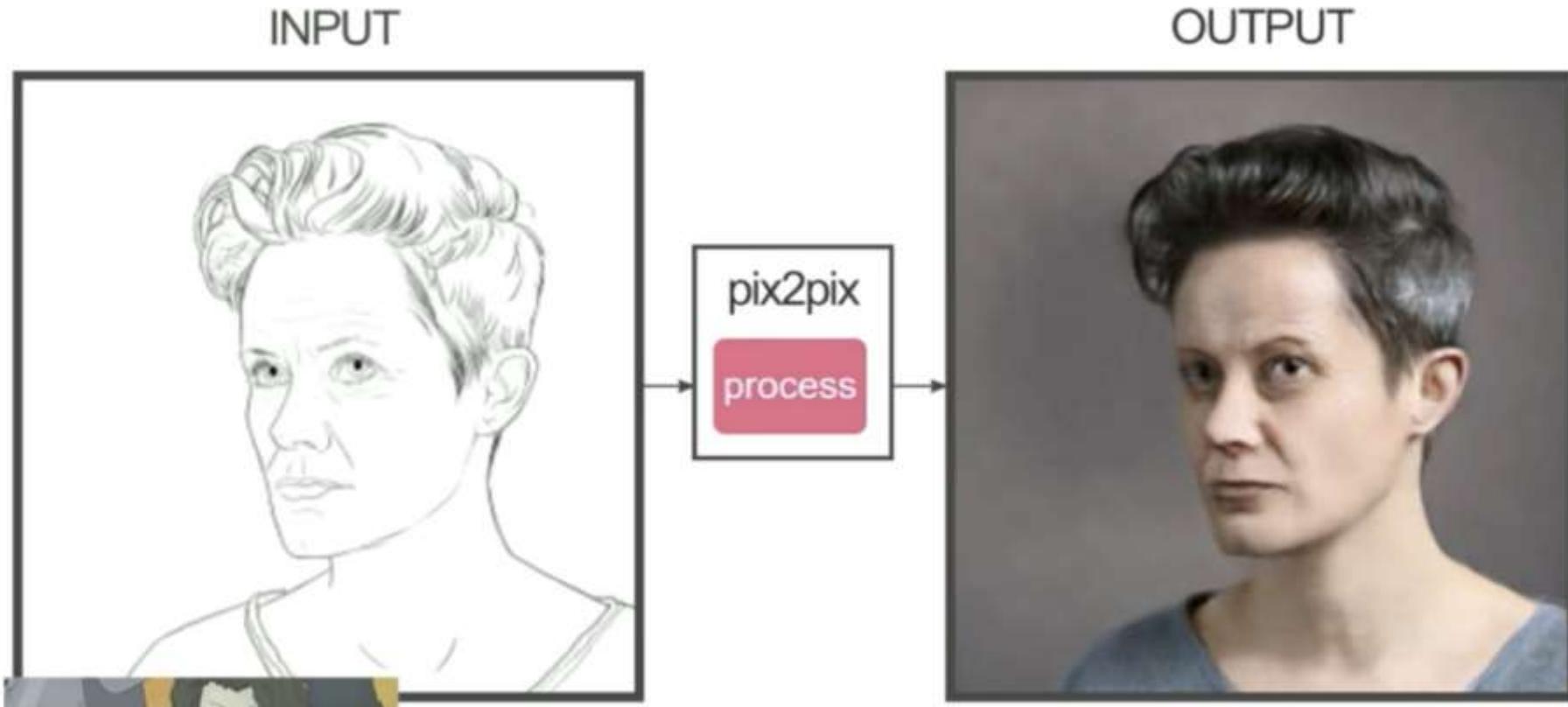
修改影像 (Image Modification)



增強解析度 (Super Resolution)

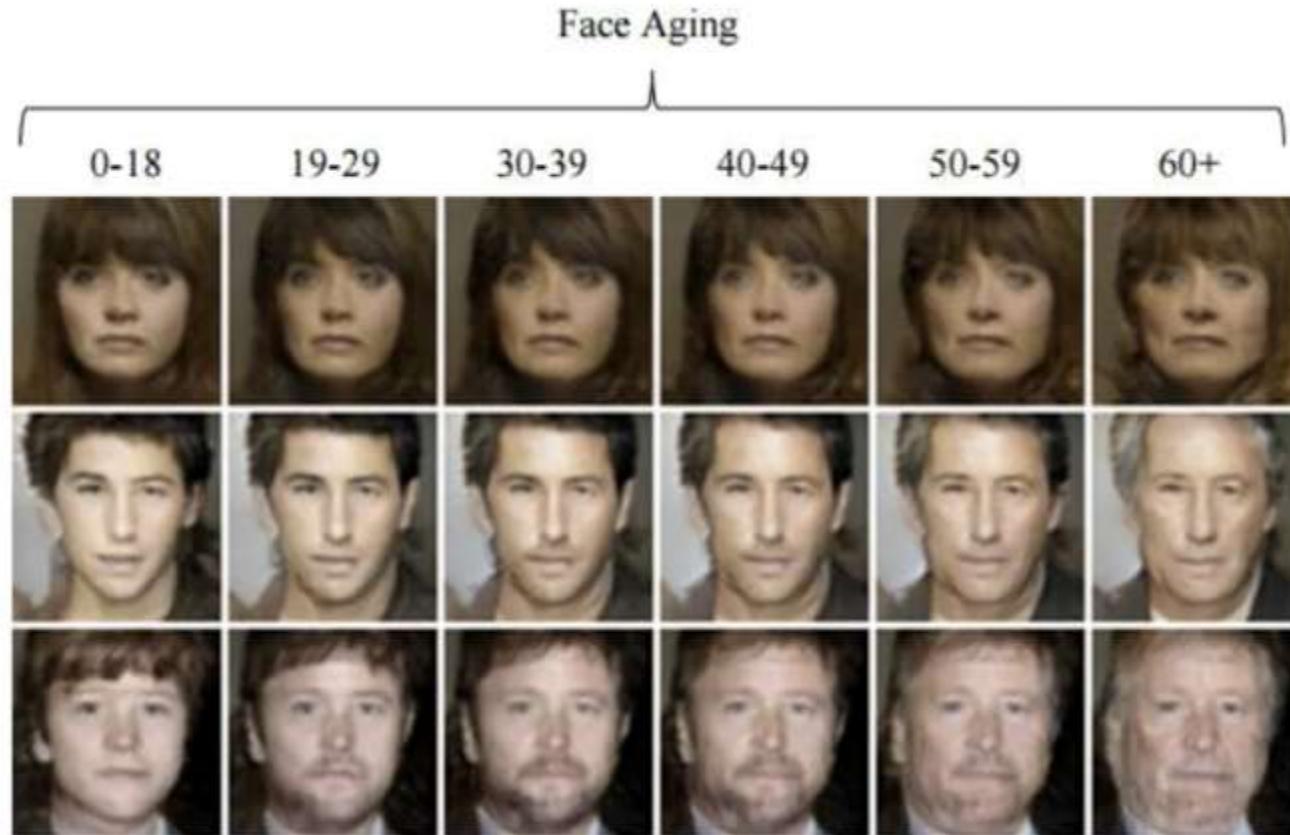


製作出如相片般的影像 (Photo Realistic Image)



<https://github.com/NVIDIA/vid2vid>

改變臉部年紀 (Face Aging)



Keras-GAN

■ <https://github.com/eriklindernoren/Keras-GAN>



Keras-GAN

Collection of Keras implementations of Generative Adversarial Networks (GANs) suggested in research papers. These models are in some cases simplified versions of the ones ultimately described in the papers, but I have chosen to focus on getting the core ideas covered instead of getting every layer configuration right. Contributions and suggestions of GAN varieties to implement are very welcomed.

See also: [PyTorch-GAN](#)

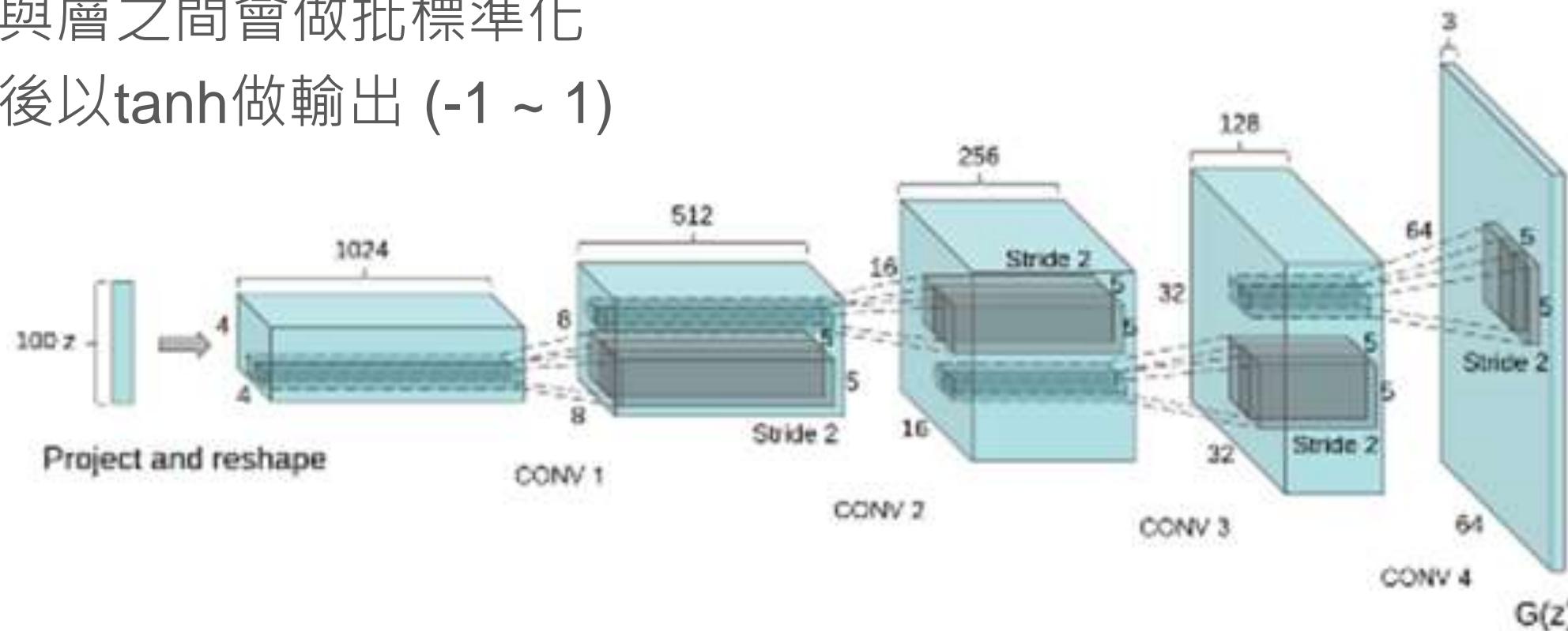
DCGAN

DCGAN (Deep Convolutional Generative Adversarial Networks)

- GAN訓練起來非常不穩定，經常會使得生成器產生沒有意義的輸出
- A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In International Conference on Learning Representations (ICLR), 2016,
- 基於卷積神經網路的對抗生成網路
 - 卷積網路架構
 - 批標準化

DCGAN 生成器

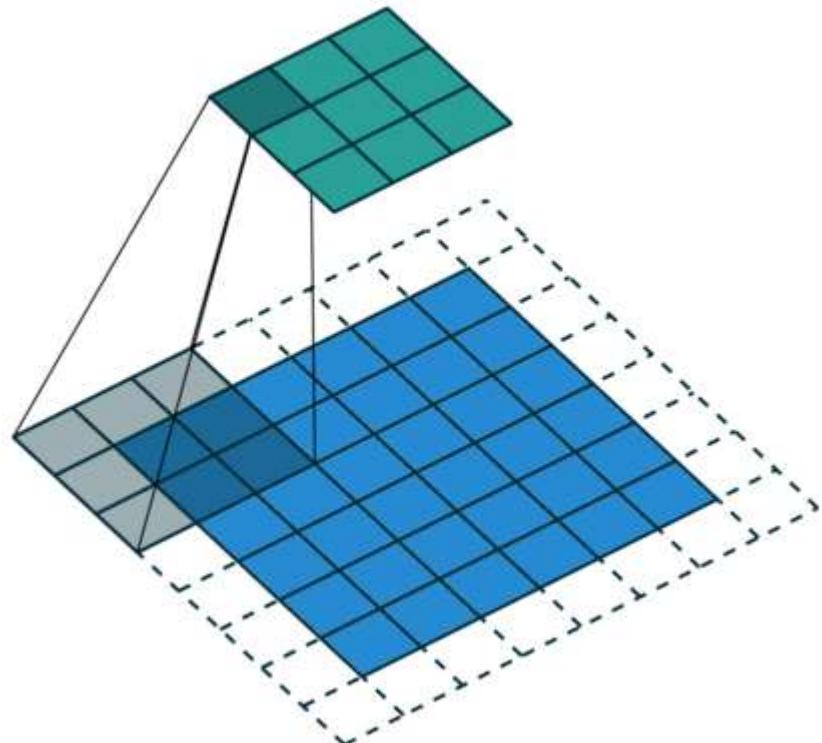
- 使用ReLU 做非線性啟動
- 沒有Max Pooling 但不斷做轉置卷積 (Transposed Convolution)
- 層與層之間會做批標準化
- 最後以tanh做輸出 (-1 ~ 1)



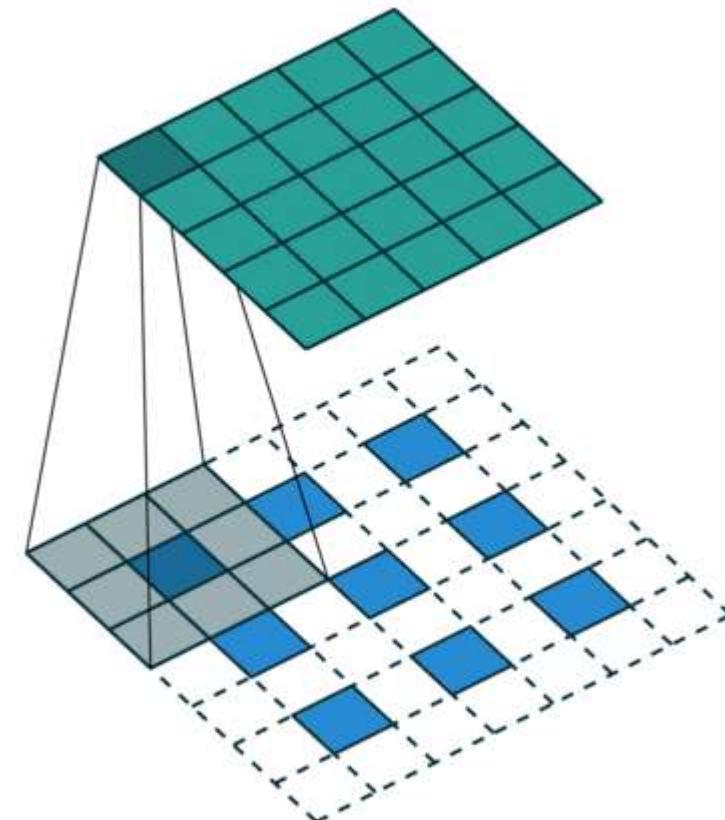
小步長卷積 (Fractionally-Strided Convolution)

一般卷積

Stride = 2

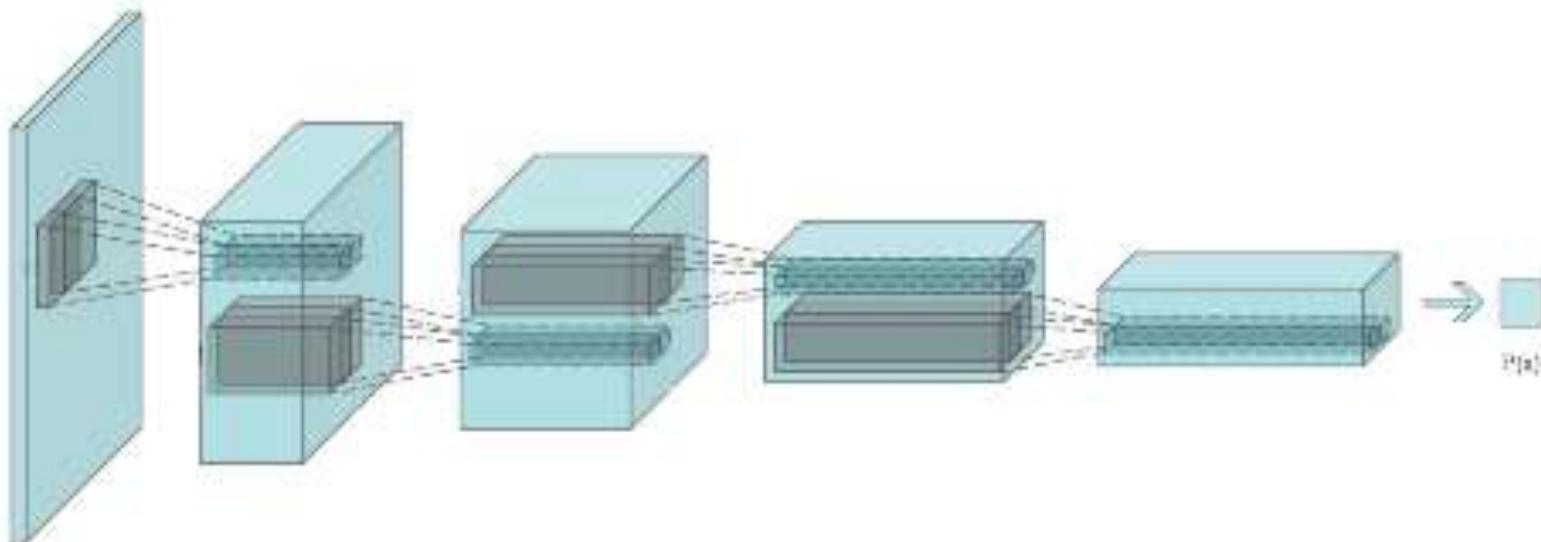


轉置卷積



DCGAN 鑑別器

- 使用Leaky ReLU 做非線性啟動
- 沒有Max Pooling 層與層之間做卷積
- 層與層之間會做批標準化
- 最後以sigmoid做輸出 (0 ~ 1)



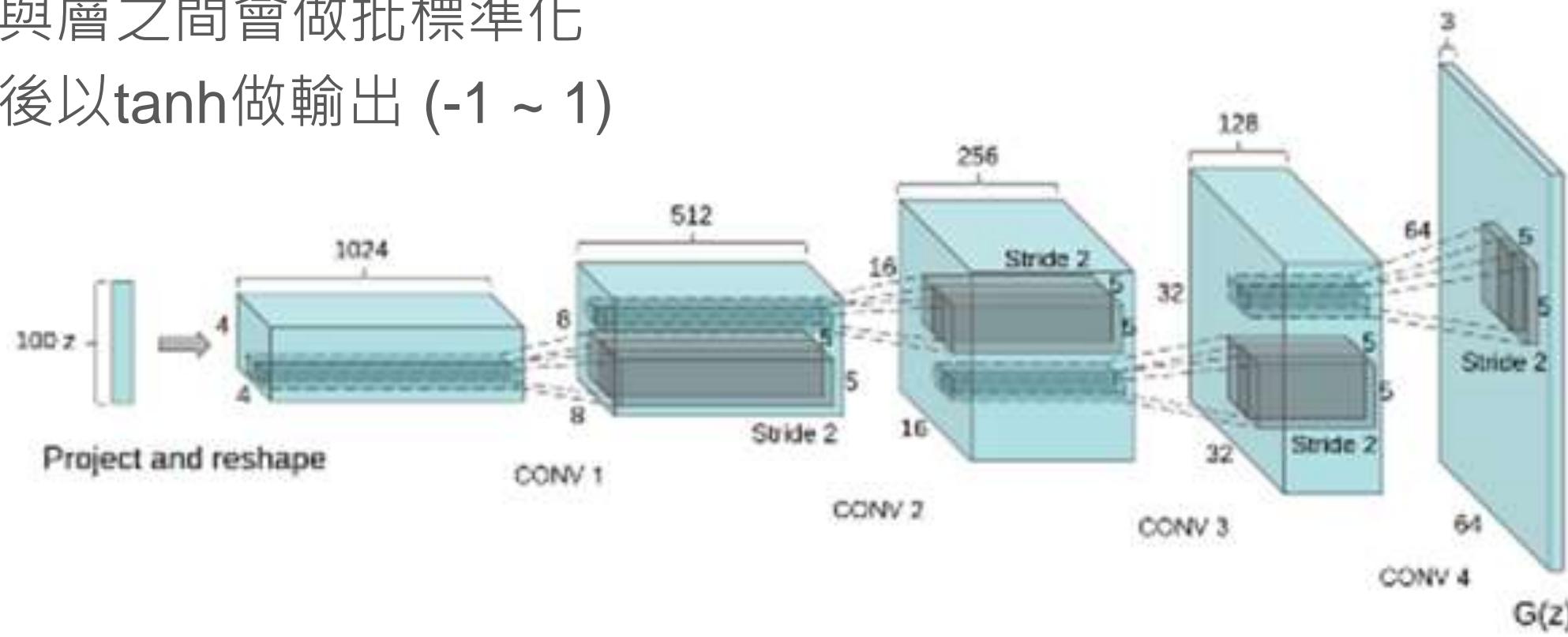
DCGAN 特色

- 使用小步卷積取代池化層
 - 鑑別器用卷積
 - 生成器用轉置卷積
- 層與層之間都會做批標準化
- 移除隱藏層的架構.
- 生成器使用 ReLU 啟動，輸出層使用 tanh 啟動
- 鑑別器使用 LeakyReLU 啟動，輸出層使用 sigmoid 啟動

使用 DCGAN 生成臉部圖片

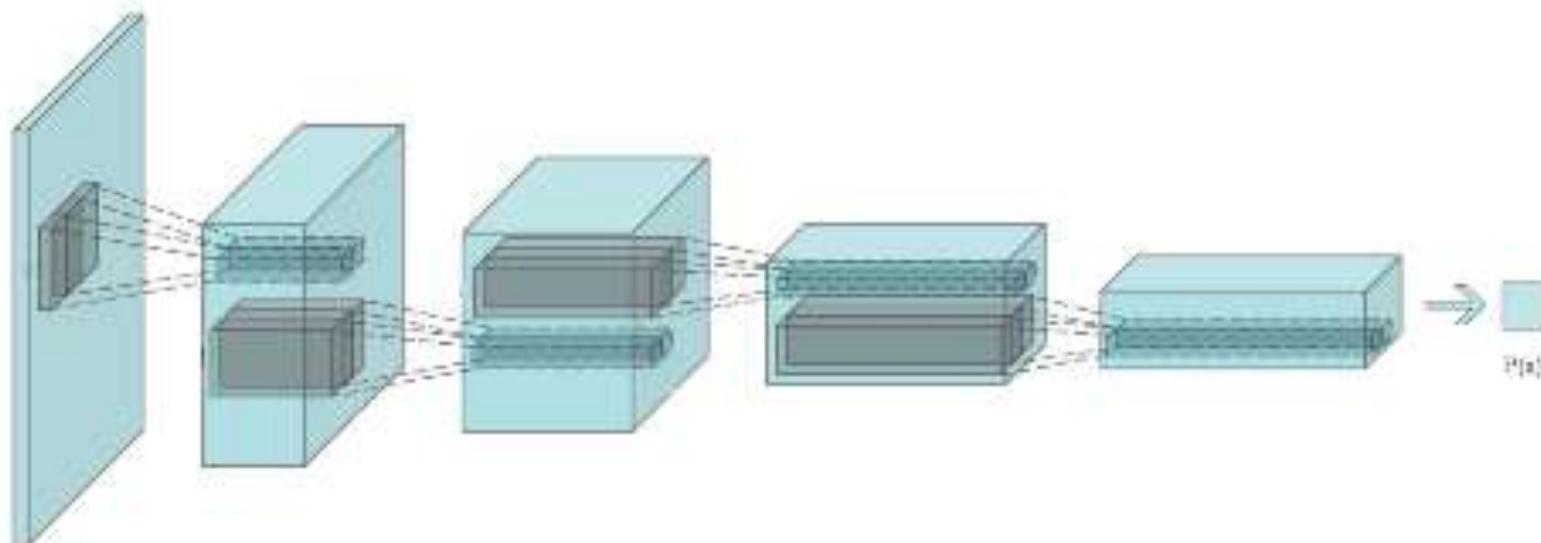
DCGAN 生成器

- 使用ReLU 做非線性啟動
- 沒有Max Pooling 但不斷做轉置卷積 (Transposed Convolution)
- 層與層之間會做批標準化
- 最後以tanh做輸出 (-1 ~ 1)



DCGAN 鑑別器

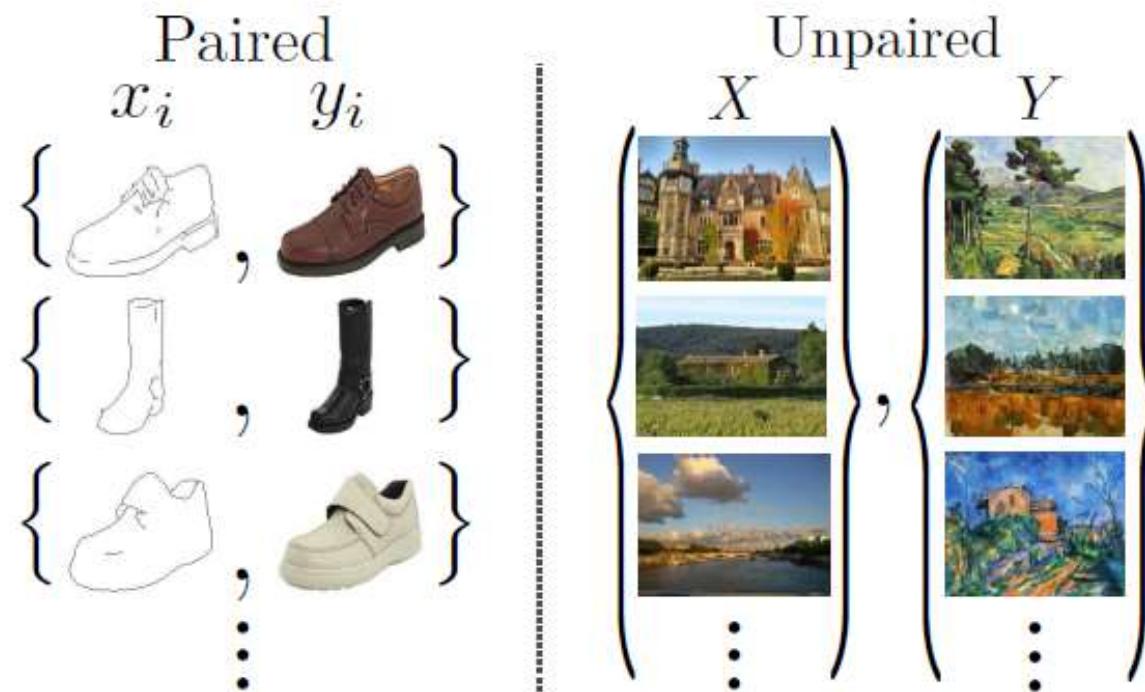
- 使用Leaky ReLU 做非線性啟動
- 沒有Max Pooling 層與層之間做卷積
- 層與層之間會做批標準化
- 最後以sigmoid做輸出 (0 ~ 1)



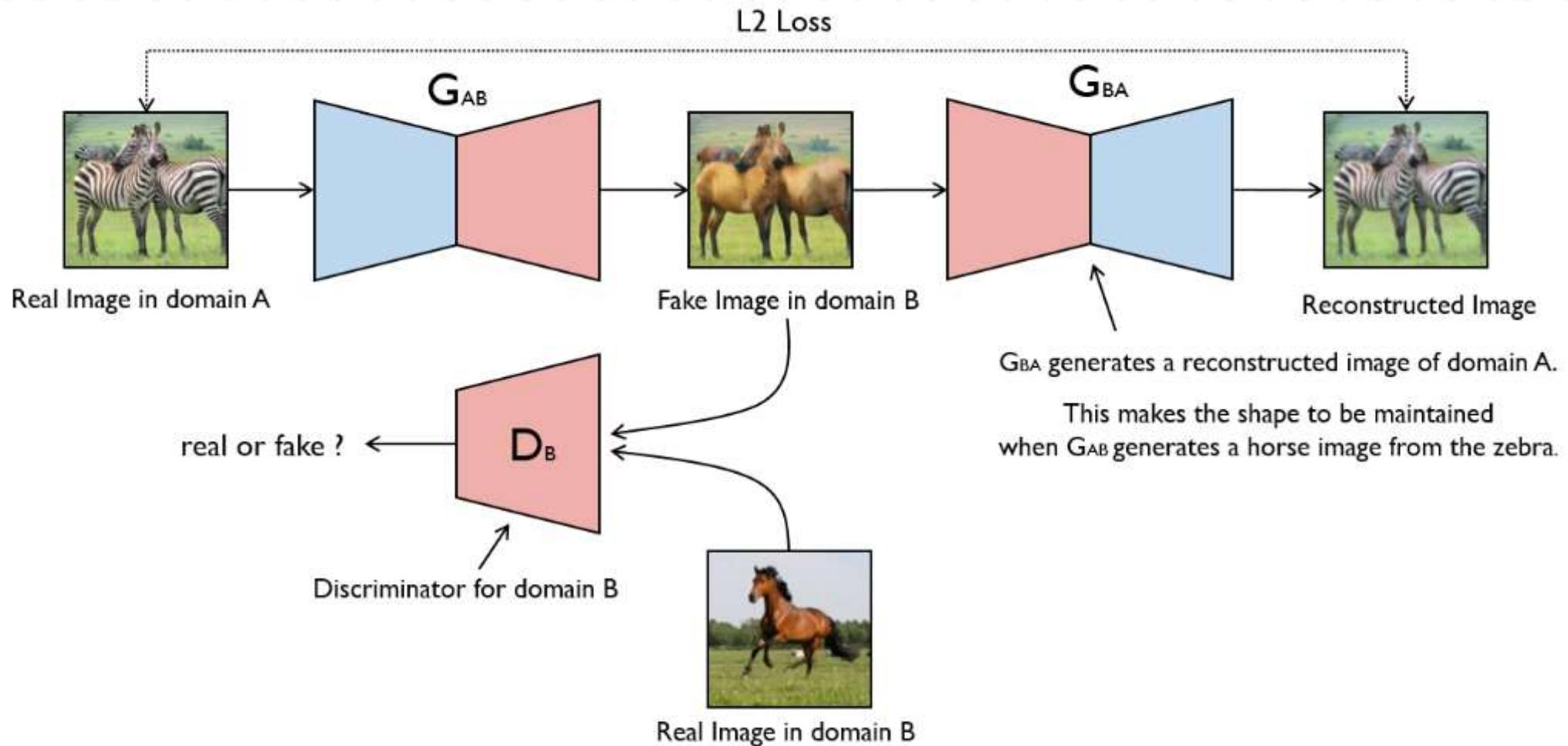
CYCLEGAN

Paired v.s. Unpaired

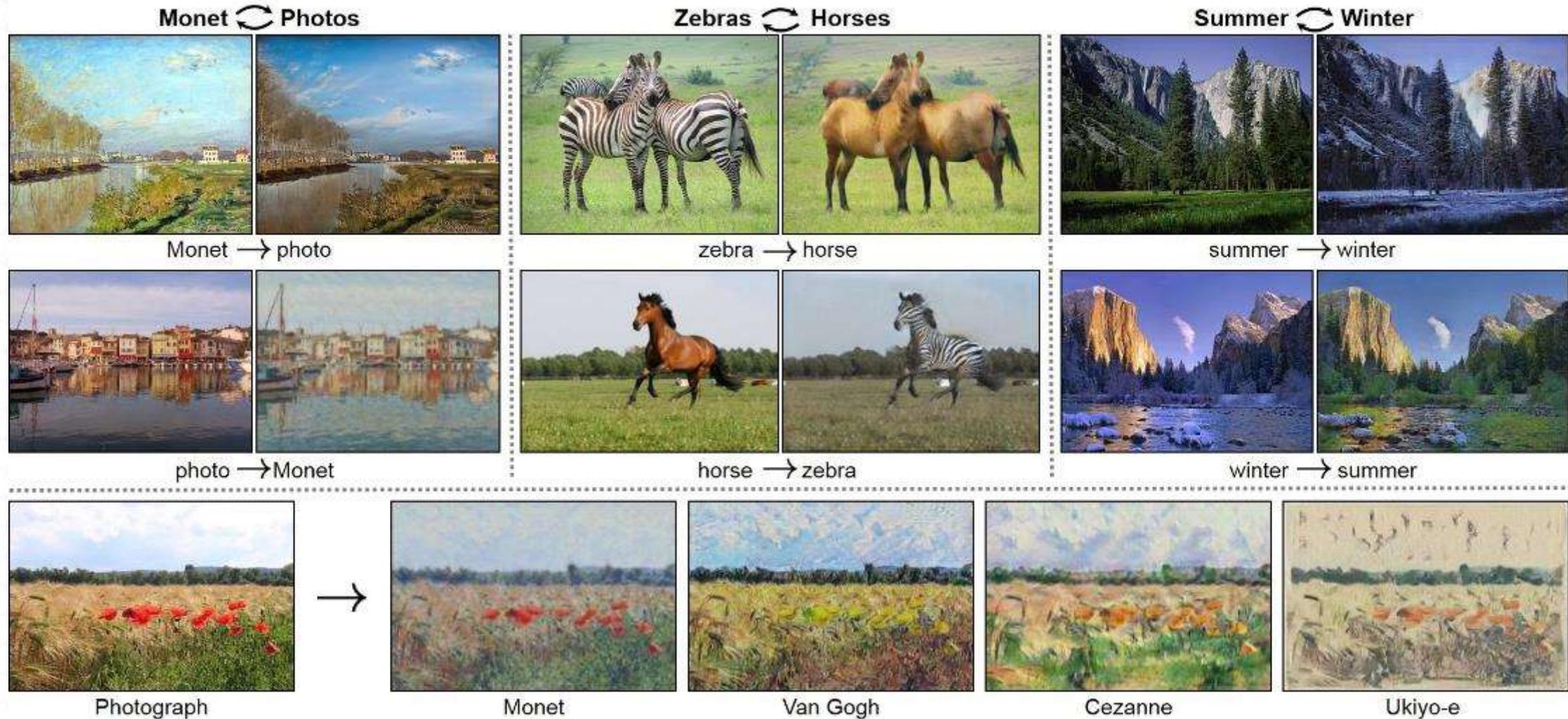
- pix2pix利用Conditional GAN去訓練一個已經配對好（ Paired ）的資料集，CycleGAN 可以用在任意（Unpaired）兩組圖上



CycleGAN



CycleGAN



使用 CYCLEGAN 將馬變斑馬

Batch v.s. Instance Normalization

■ Batch Normalization

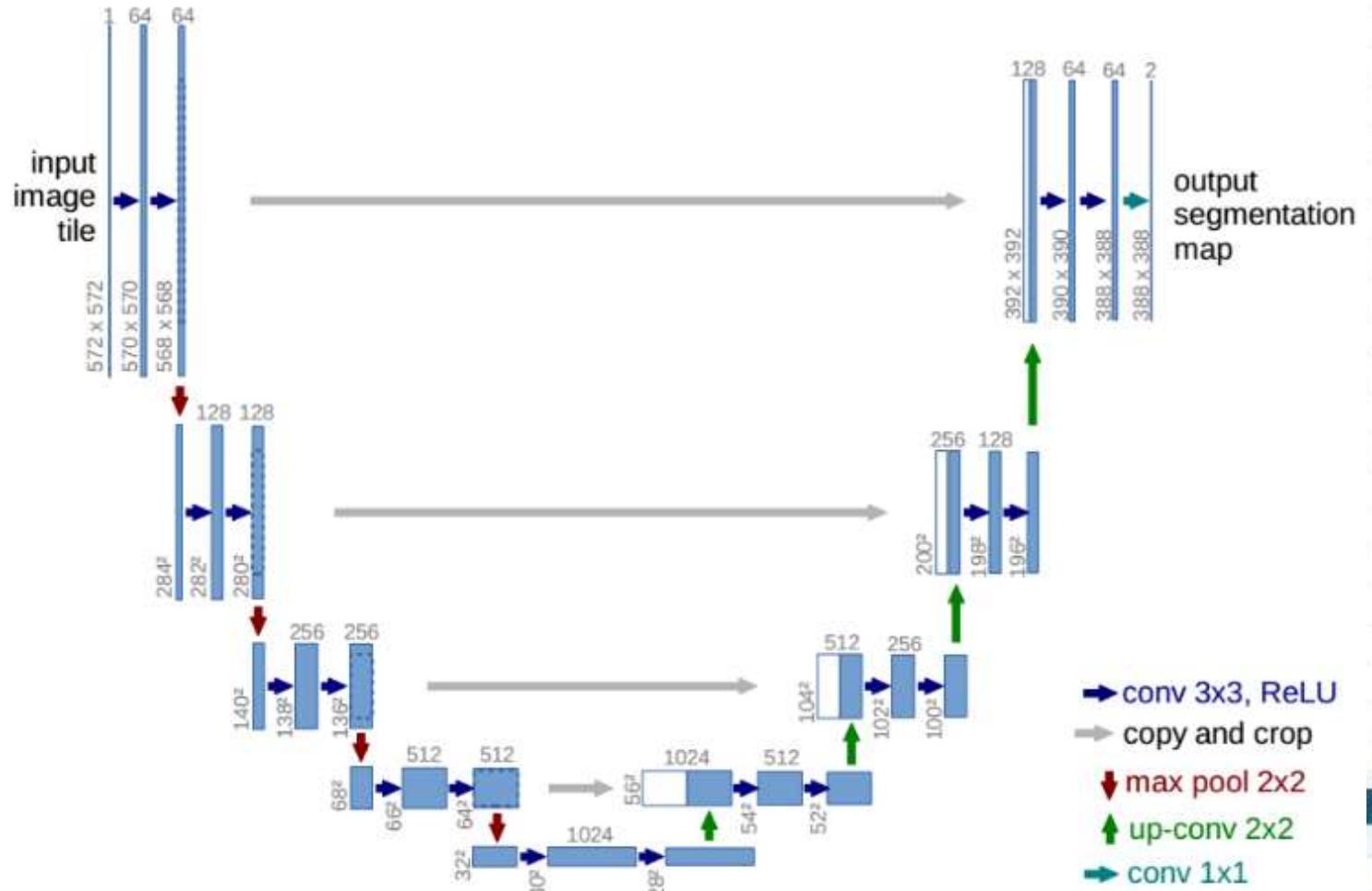
- 對一批資料進行標準化
- 產生一批樣本的統計量，容易受到其他樣本影響
- 適合用在圖片分類

■ Instance Normalization

- 對一筆資料進行標準化
- 產生單一樣本的統計量
- 適合用在風格轉移(Style Transfer)

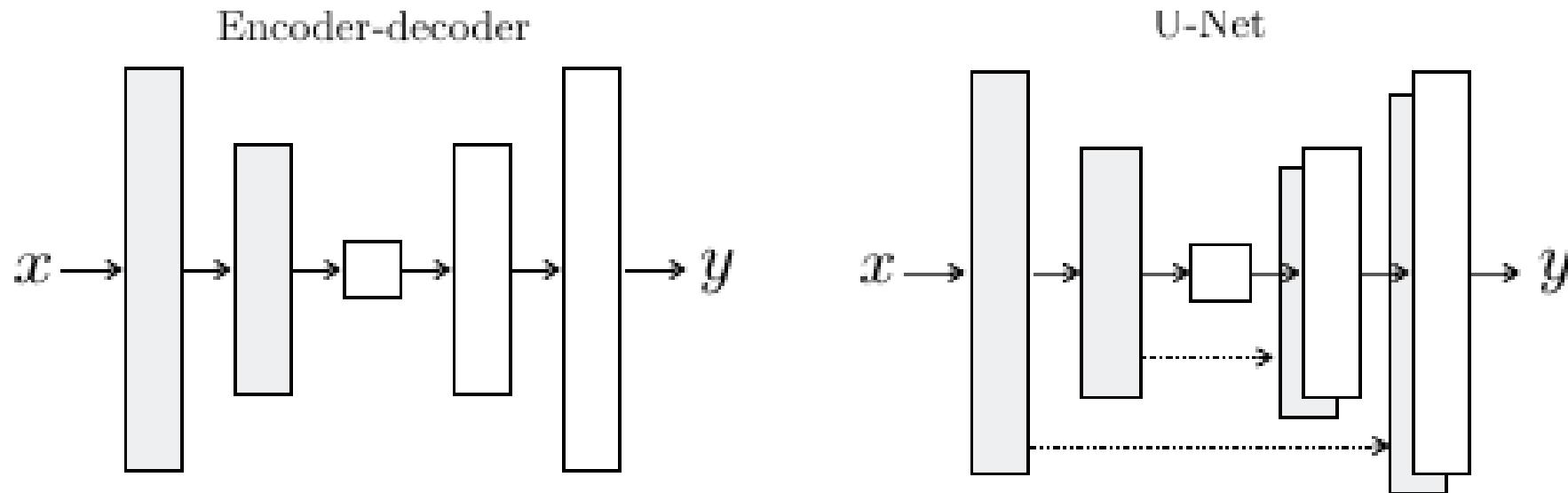
U-Net

- 先下採樣(Downsampling)到低維度，再上採樣(Upsampling)到原始解析度的網路結構
- U-Net加入跨層連結(skip-connection)，將對應的編碼層與解碼層連結一起，詳細保留不同解析度的細節資訊



Encoder-Decoder 網路架構 v.s. U-Net 網路架構

- U-Net加入跨層連結(skip-connection)，將對應的編碼層與解碼層連結一起，詳細保留不同解析度的細節資訊



PatchGAN

- PatchGAN 是一種卷積網路，專注計算材質/風格的損失
- 鑒別器鑒別 $N \times N$ 的patch的真偽。又稱Markovian discriminator，因為在鑒別時將每個patch視為互相獨立
- 降低輸入維度，大幅加速計算速度





THANK YOU