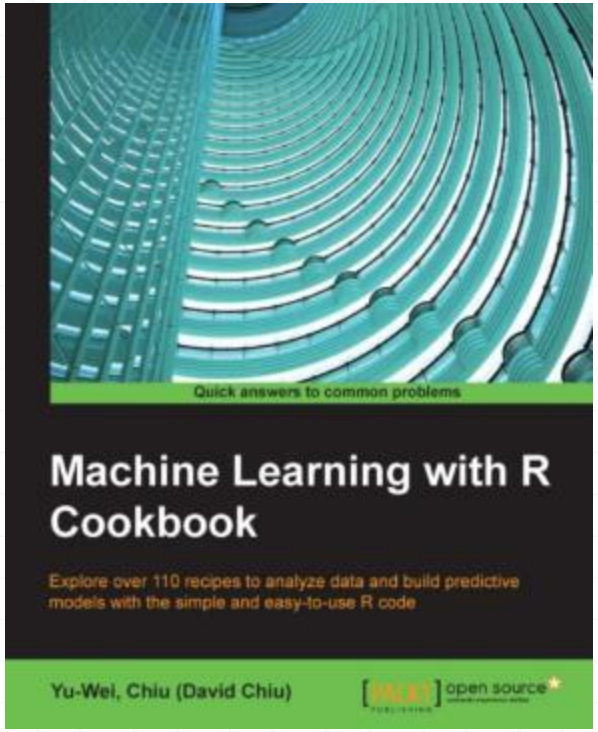
Python深度學習實戰-邁向A.I.的第一步

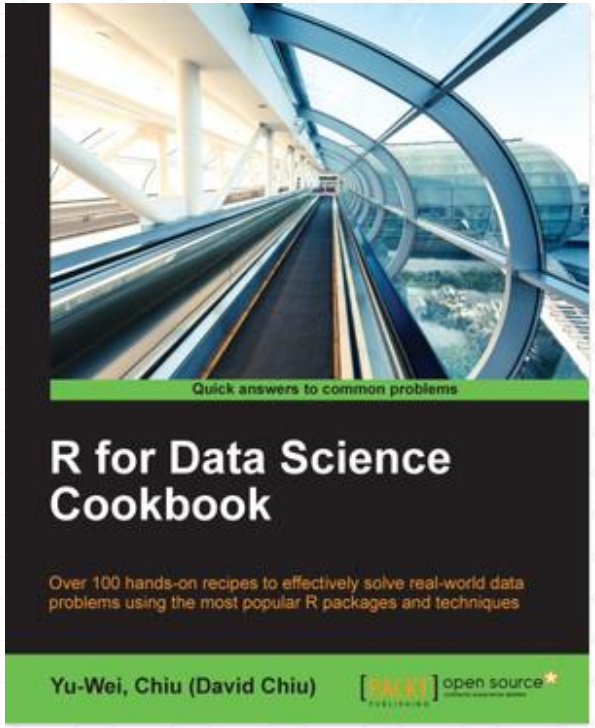
丘祐瑋
David Chiu

關於我

- 大數軟體有限公司創辦人
- 前趨勢科技工程師
- ywchiu.com
- 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 (機器學習與R語言實戰) & R for Data Science Cookbook (資料科學：R語言實現)





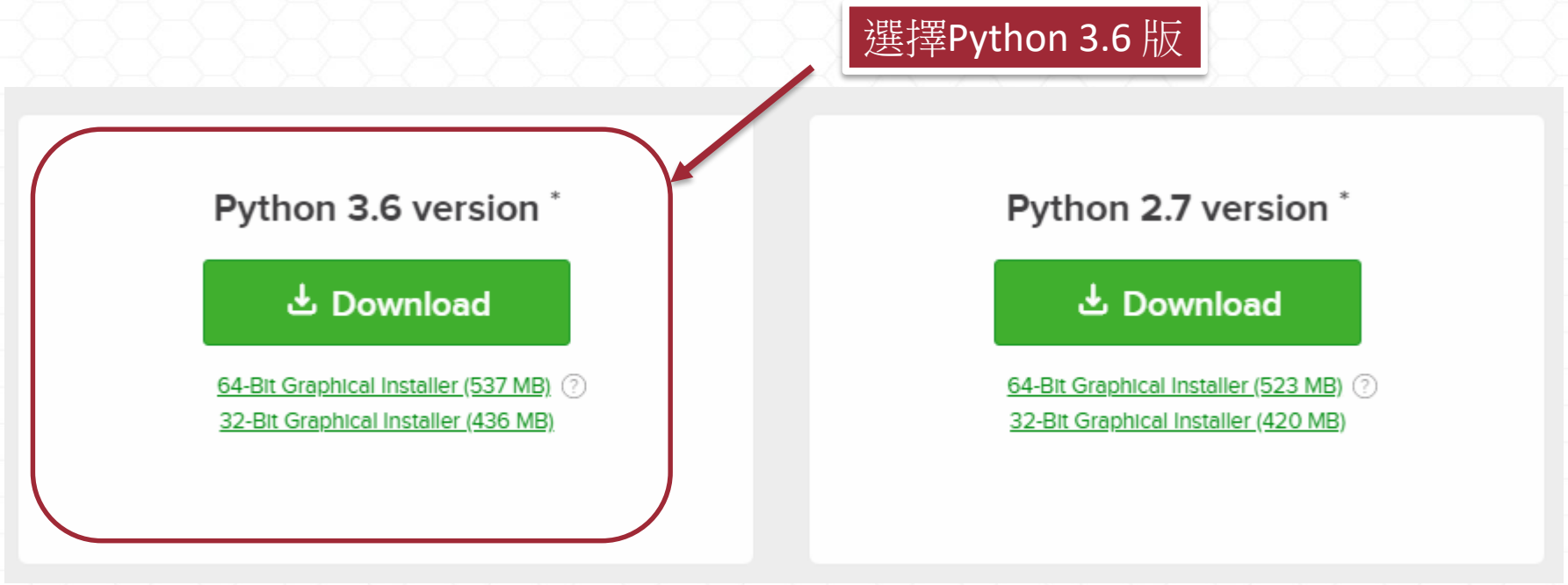
Author: David (YU-WEI CHIU) Chiu

課程資料

- 所有課程補充資料、投影片皆位於
  - https://github.com/ywchiu/tibamedl

Python 開發工具簡介

安裝Anaconda

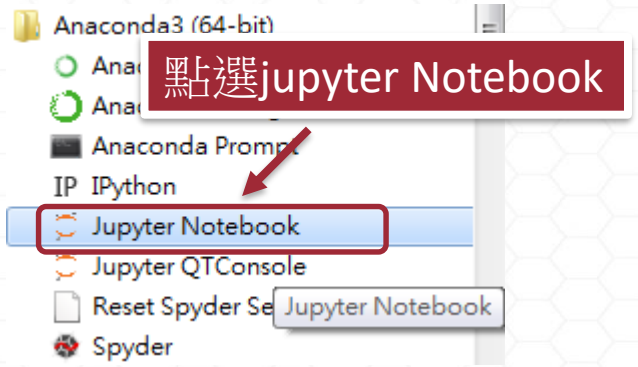


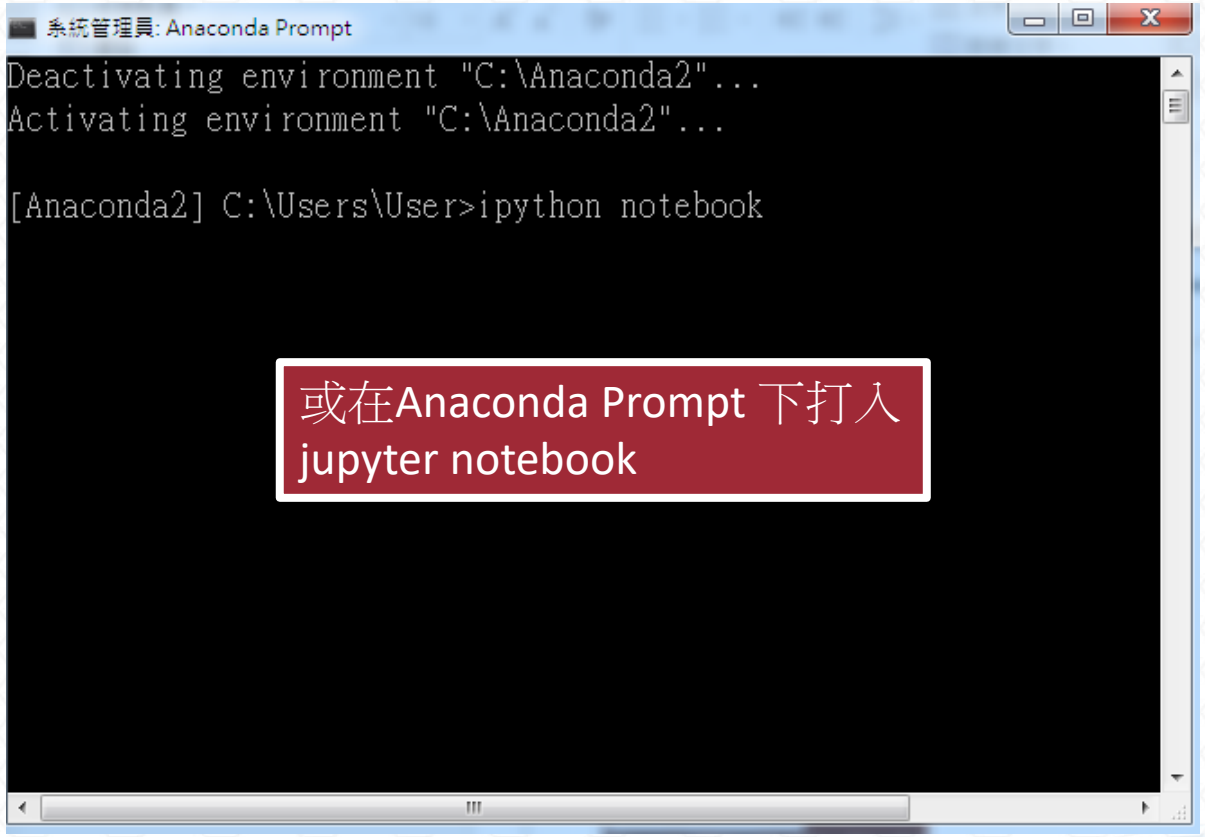
https://www.anaconda.com/download/

Jupyter Notebook

- 使用於教學與資料探索，不適合開發大型專案

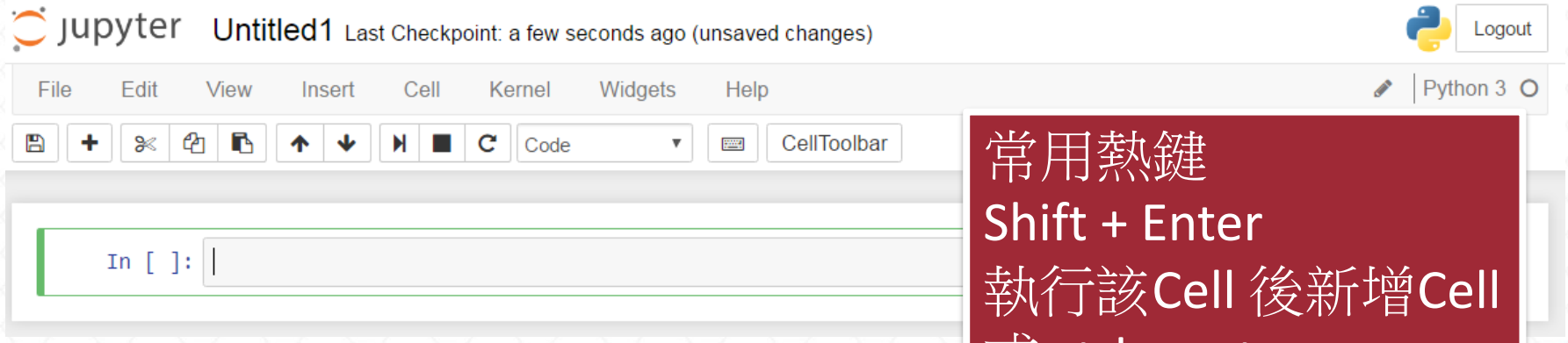
使用 Jupyter Notebook





啟用 Jupyter (jupyter Notebook)

- 在命令列下打:
  - jupyter notebook
  - 自動開啟瀏覽器後便可瀏覽 (預設為localhost:8888)
- 可匯出.ipynb, .py 各種不同格式檔案
- 瀏覽快捷鍵 Help -> Keyboard Shortcuts



常用熱鍵
Shift + Enter
執行該Cell 後新增Cell
或 ctrl + enter
執行該Cell

人工智慧的發展歷史

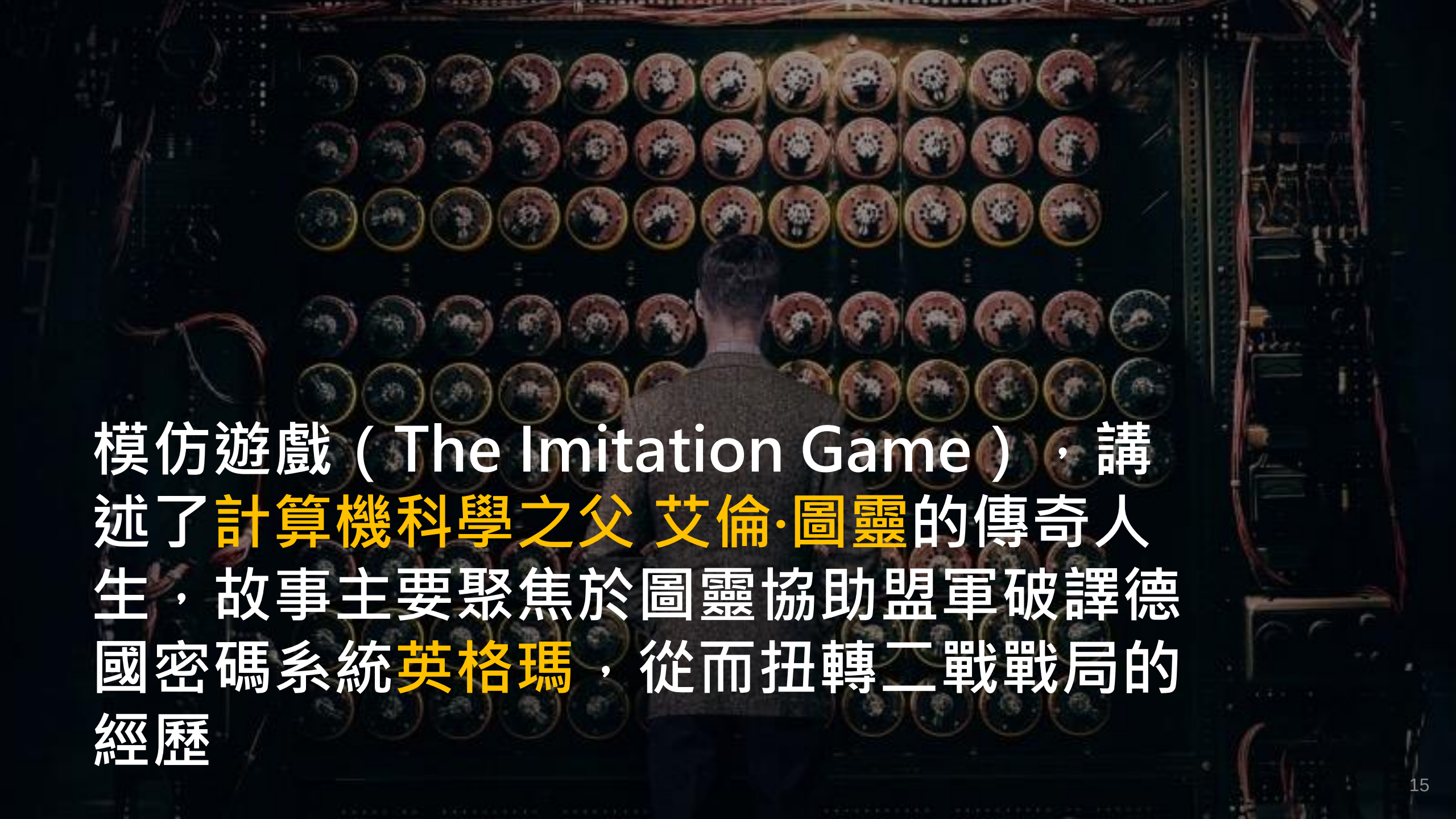
AlphaGO
使用深度學習技術打敗頂尖棋手

AlphaGO Zero

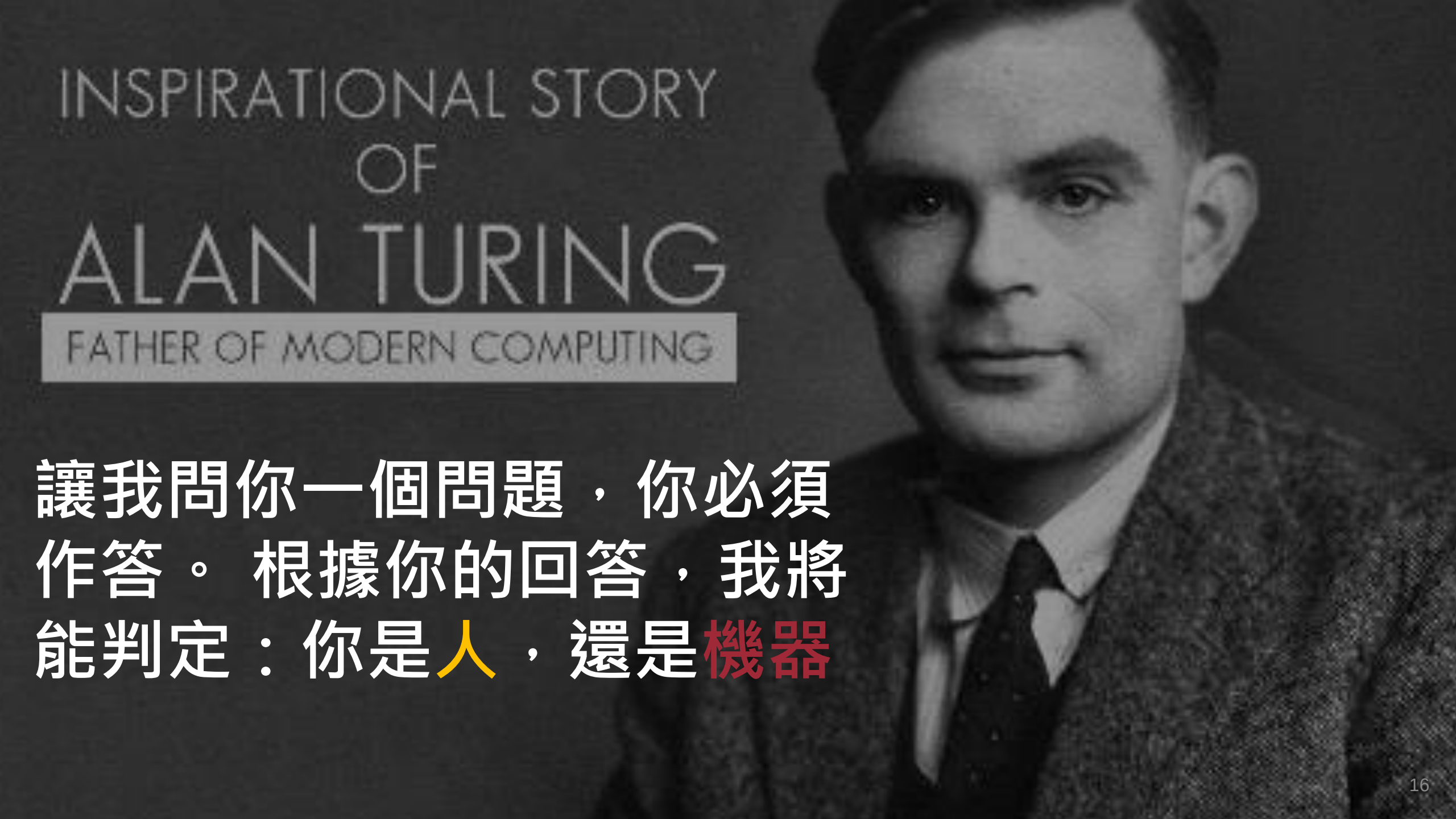
只需要三天就能熟悉第一代AlphaGo所有的圍棋知識

Google 成立 Waymo
打造全球第一個無人車上路測試

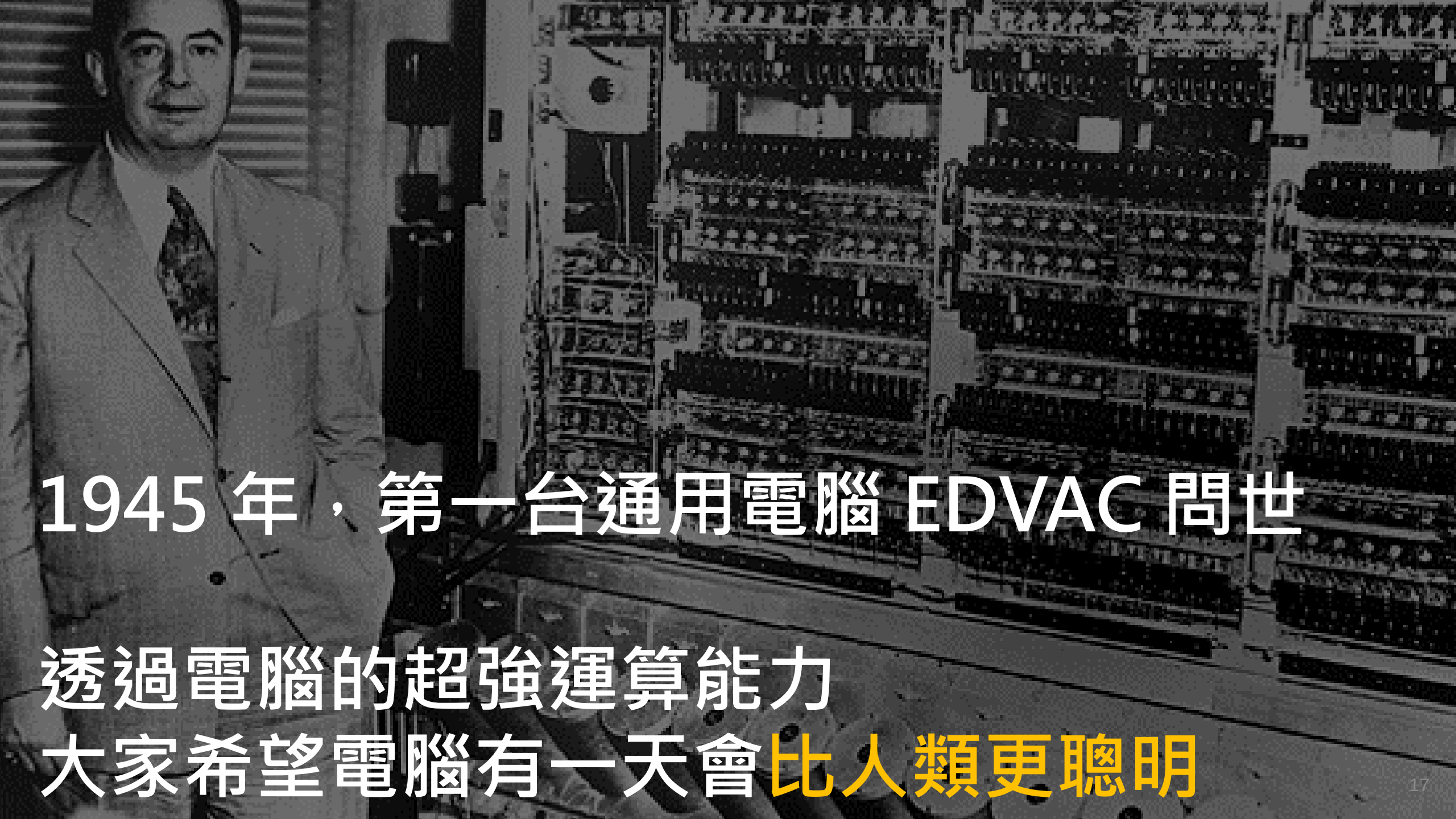
什麼是人工智能(A.I)?



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**英格瑪**，從而扭轉二戰戰局的經歷



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)

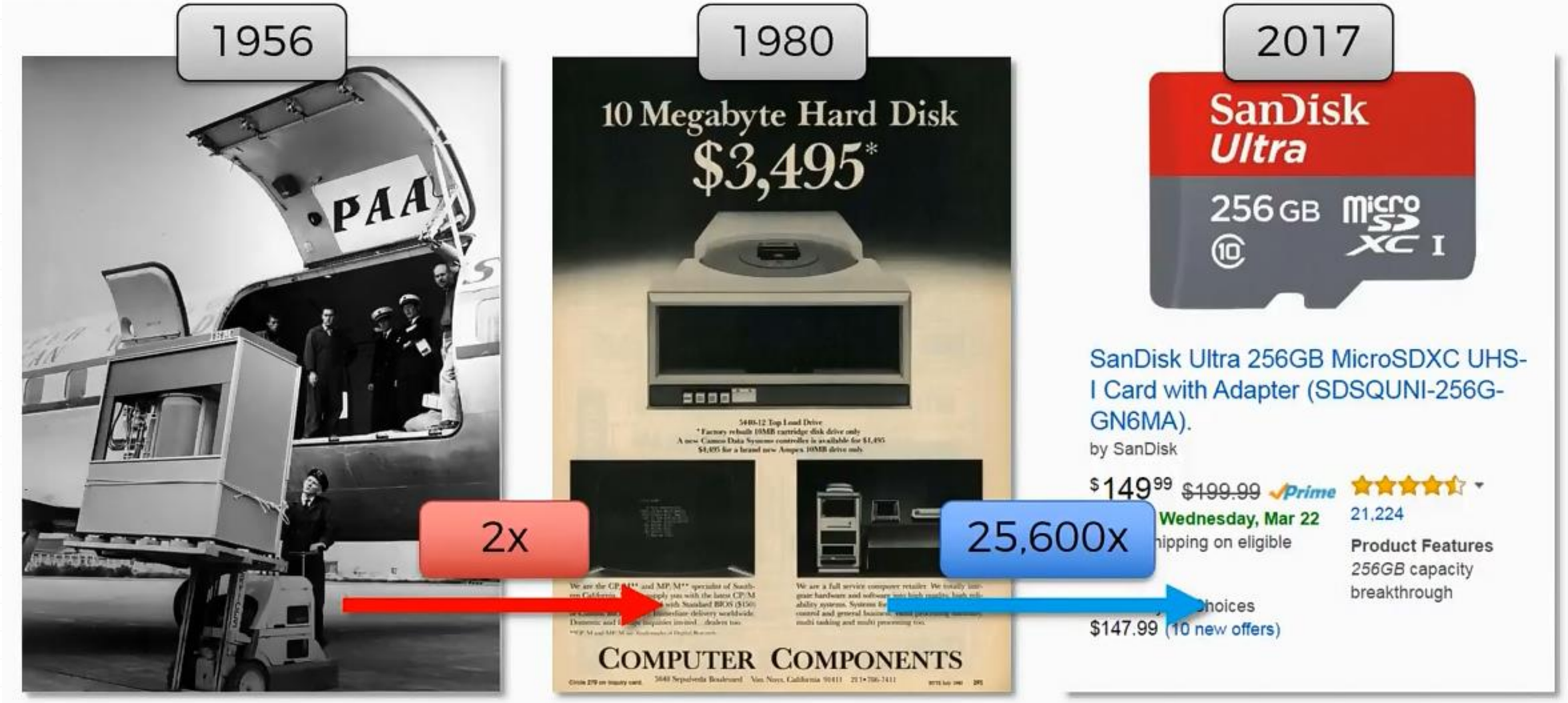
阿西莫夫《我，機器人》中設定機器人三定律做為機器人的行為準則

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

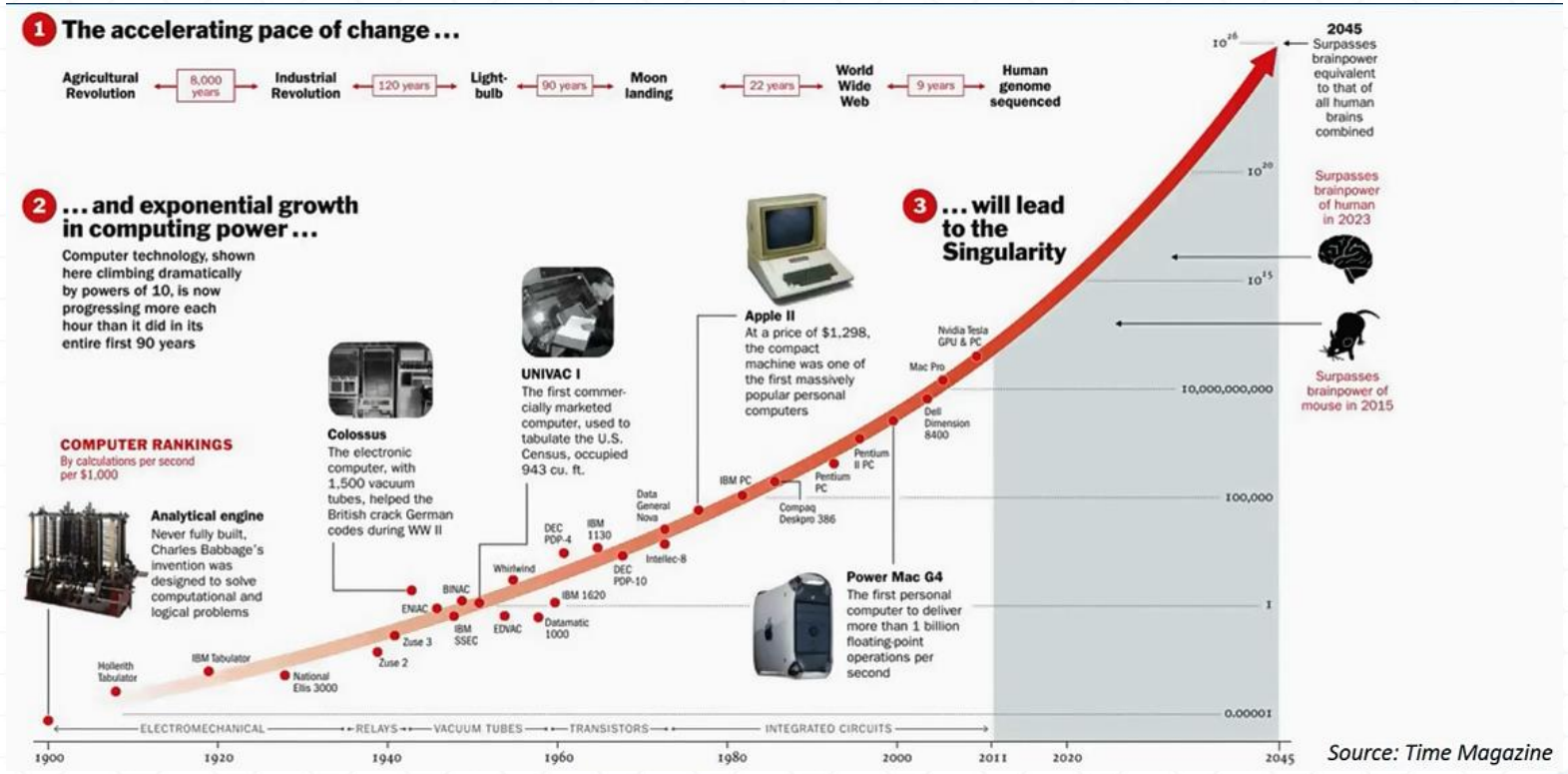
第三定律：機器人在不違反第一、第二定律的情況下要盡可能保護自己的生存

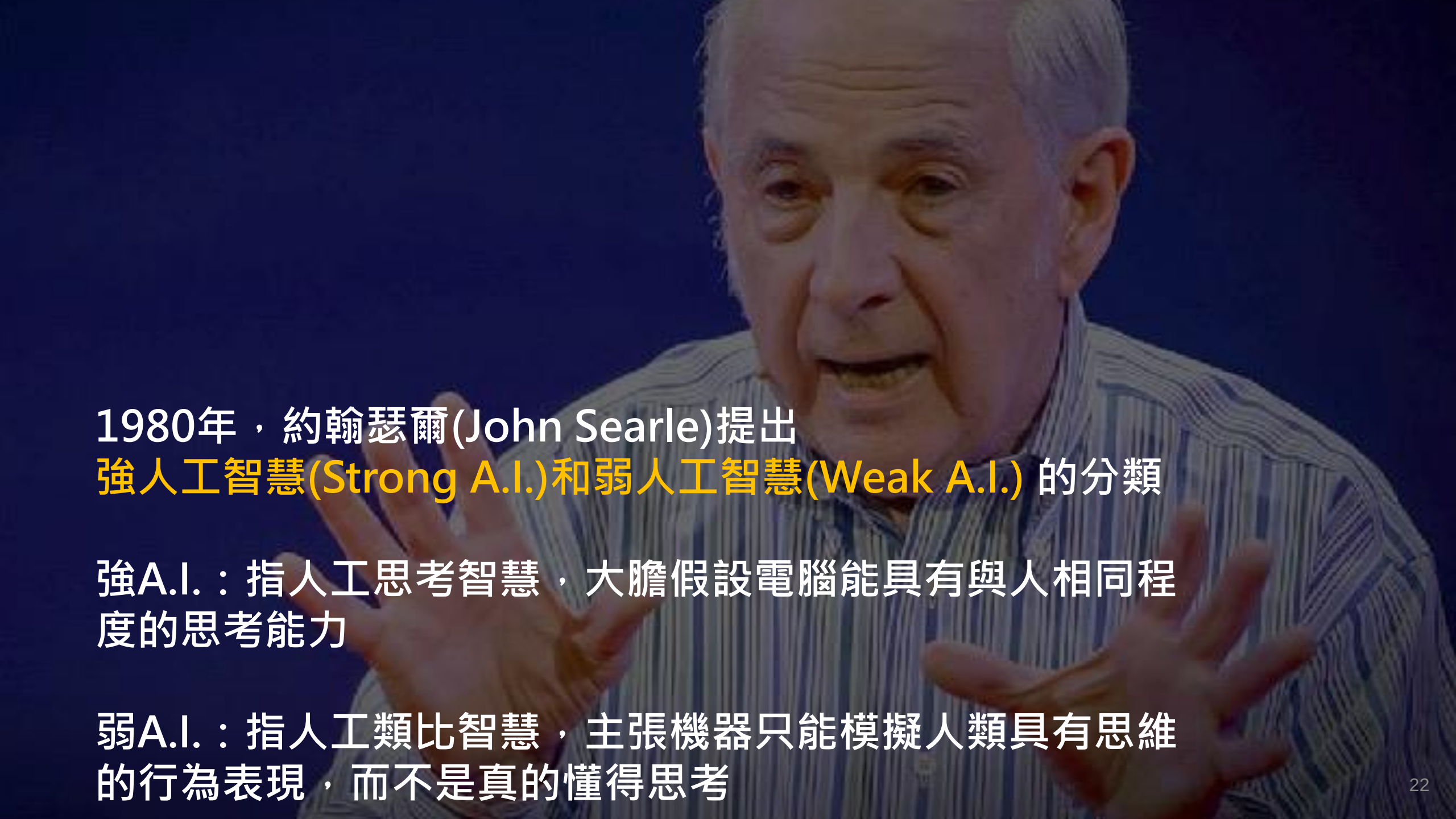
硬體成本逐漸降低


1956
1980
2017
10 Megabyte Hard Disk
$3,495*
COMPUTER COMPONENTS
SanDisk
Ultra
256 GB
SanDisk Ultra 256GB MicroSDXC UHS-I Card with Adapter (SDSQUNI-256G-GN6MA).
by SanDisk
$149.99
$199.99
Prime
Wednesday, Mar 22
21,224
Product Features
256GB capacity
breakthrough
$147.99 (10 new offers)
2x
25,600x

莫爾定律

- 當價格不變時，**電晶體**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

強人工智慧

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

弱人工智慧 - 預測模型

- 是否能用統計機率學來處理人工智慧的問題呢？

- 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

Attribute

Target

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.

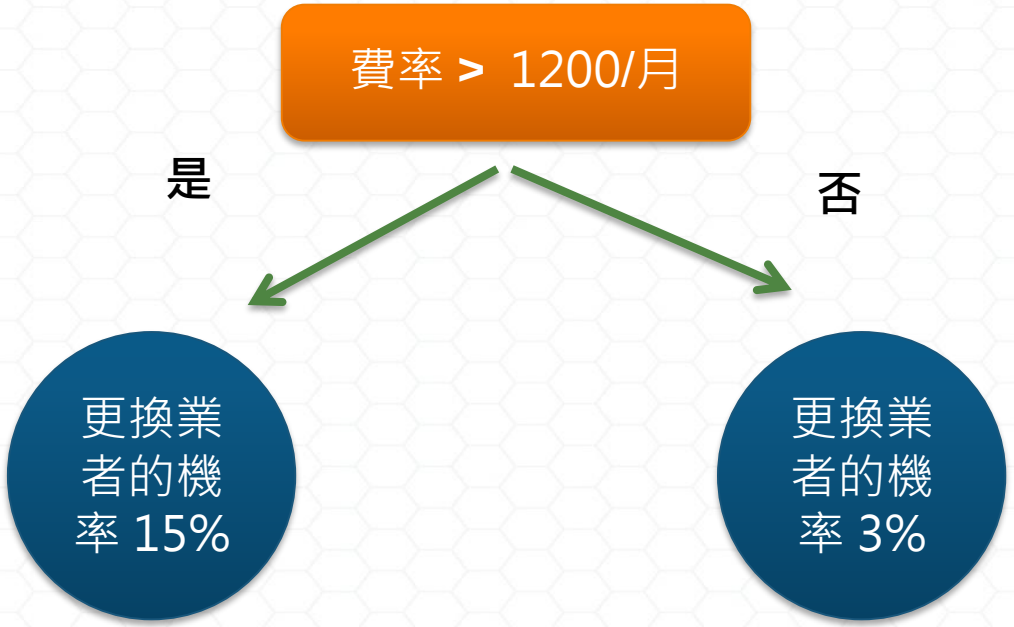
假使使用者為女性

而且月收入高達3萬以上

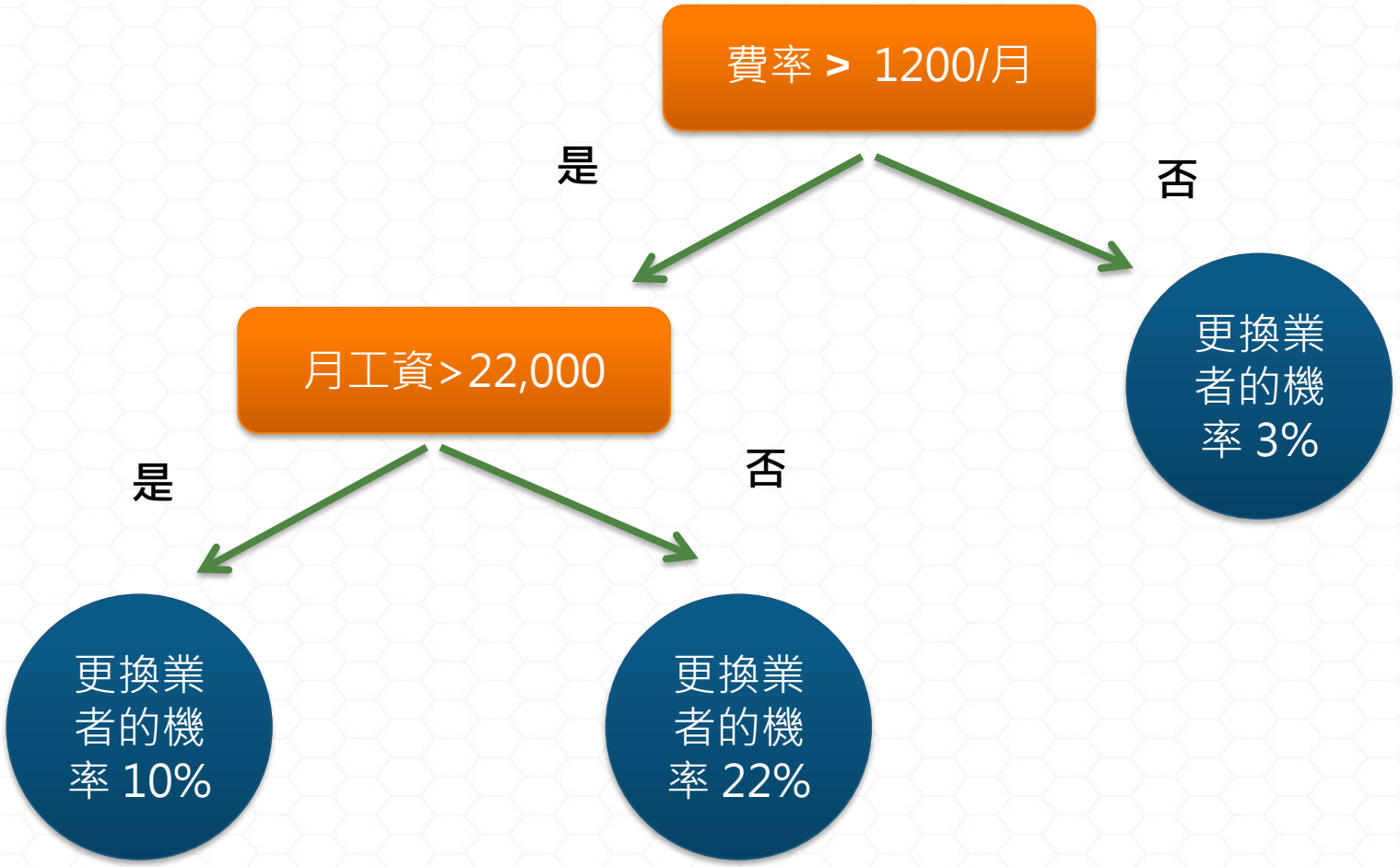
而且還沒看過這廣告

點擊機率為11%

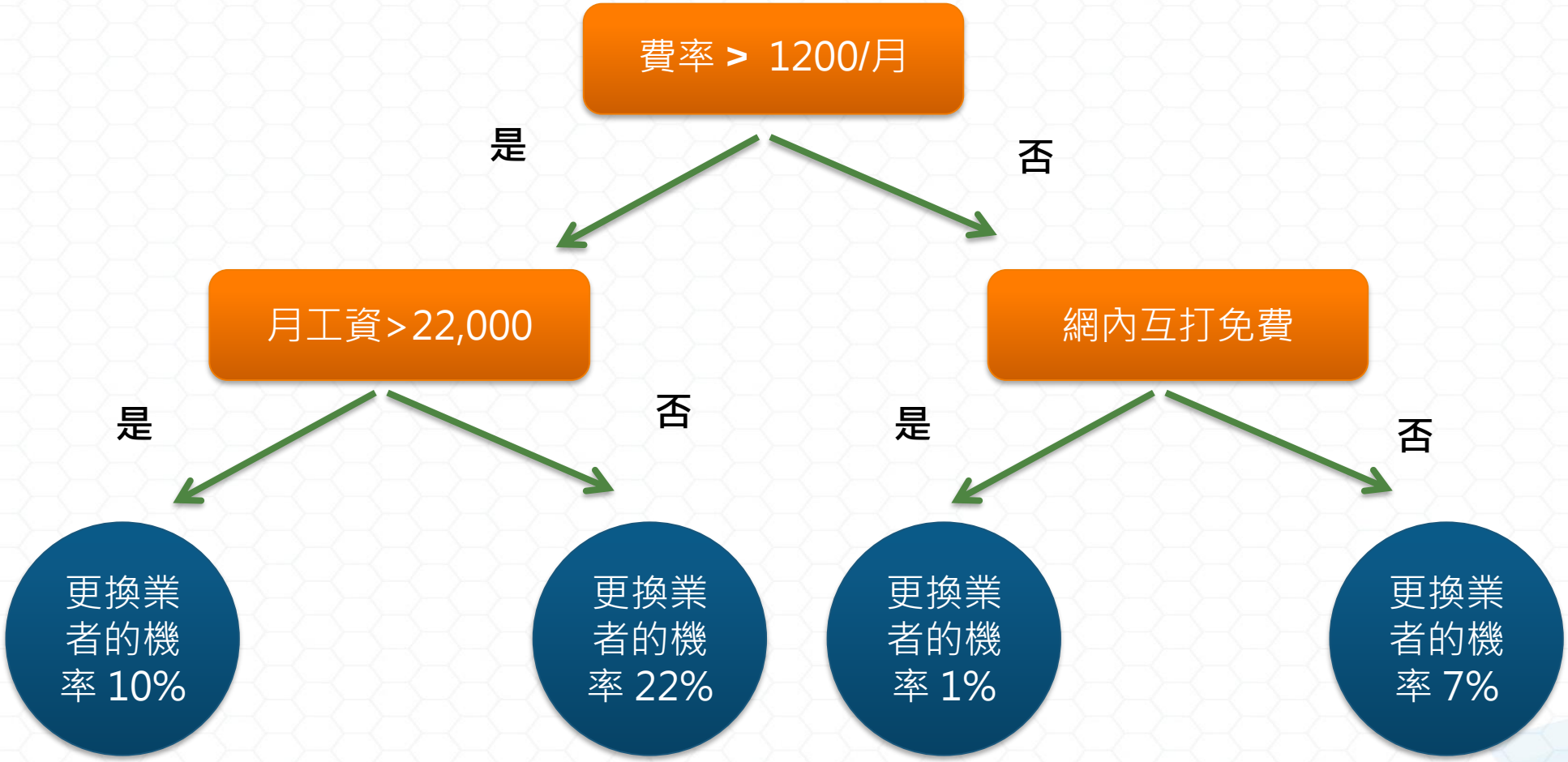
簡單的分類問題(決策樹)



簡單的分類問題(決策樹)



簡單的分類問題(決策樹)


費率 > 1200/月
是
否
月工資>22,000
網內互打免費
是
否
是
否
更換業者的機率 10%
更換業者的機率 22%
更換業者的機率 1%
更換業者的機率 7%

機器學習

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

搜集資料 → 數據轉換與清洗 → 建立模型 → 驗證模型 → 佈署模型

反復訓練與驗證

機器學習問題分類

- 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）

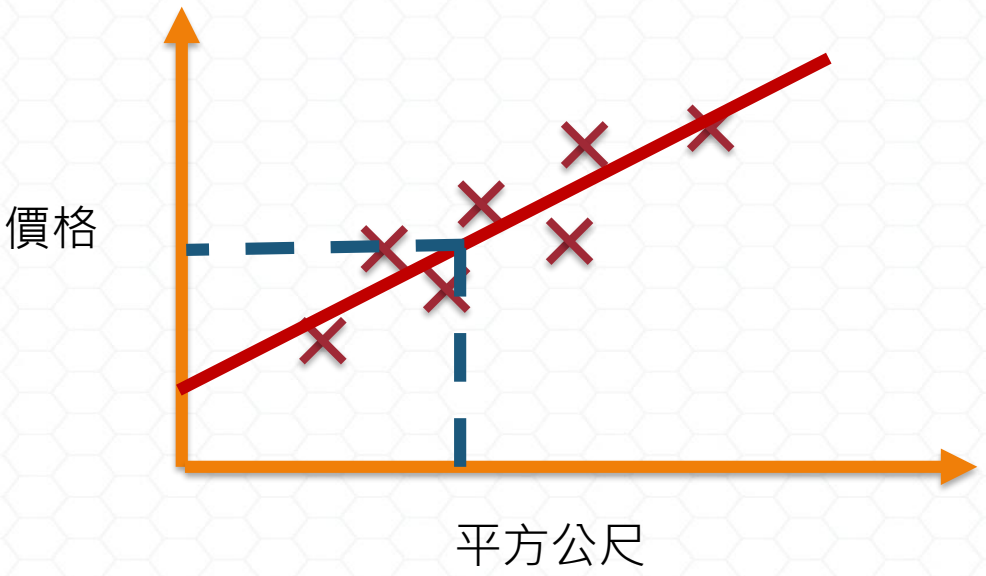
使用監督式學習進行預測

- 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

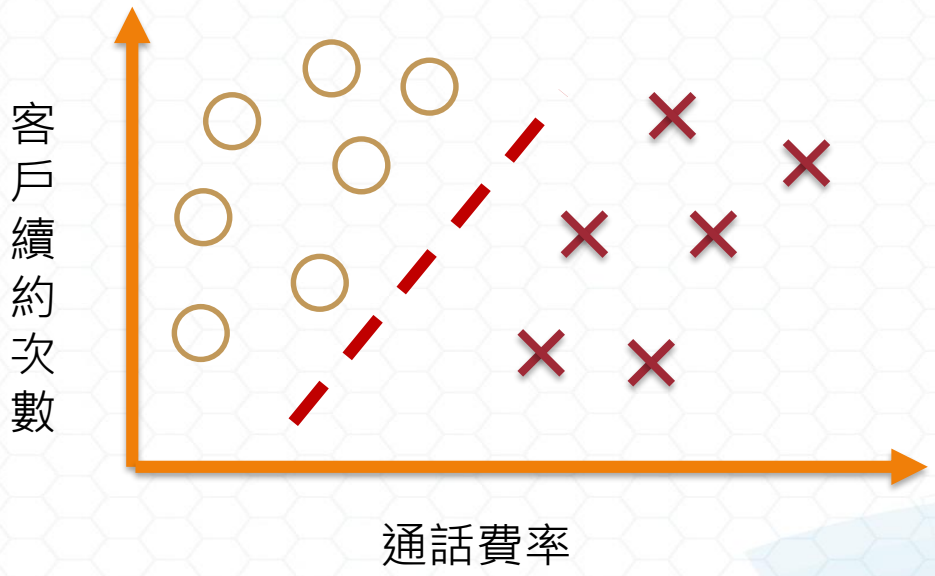
- 分類問題
  - 根據已知標籤的訓練資料集(Training Set)，產生一個新模型，用以預測測試資料集(Testing Set)的標籤。
  - e.g. 客戶流失分析

使用監督式學習進行預測

回歸分析



分類問題

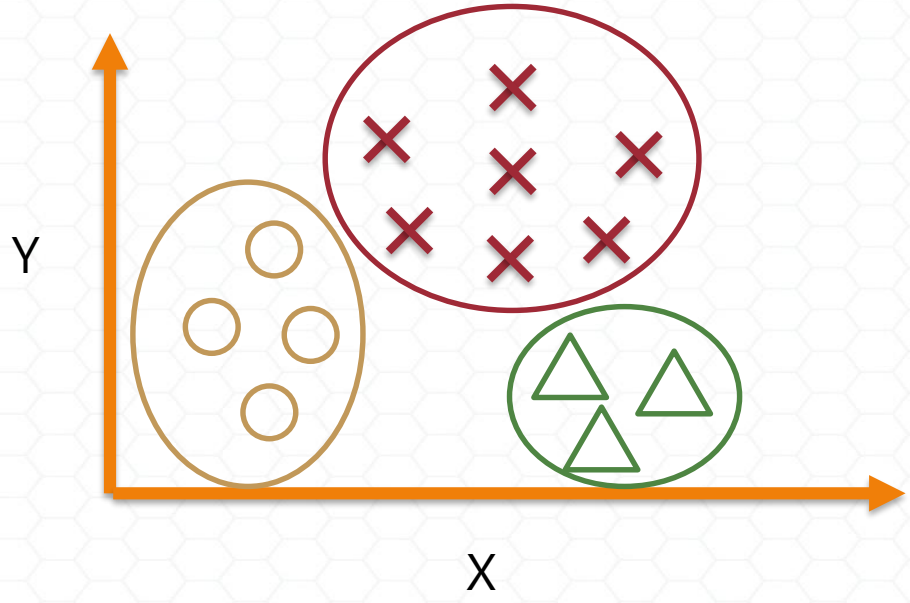


使用非監督式學習找出隱藏的架構

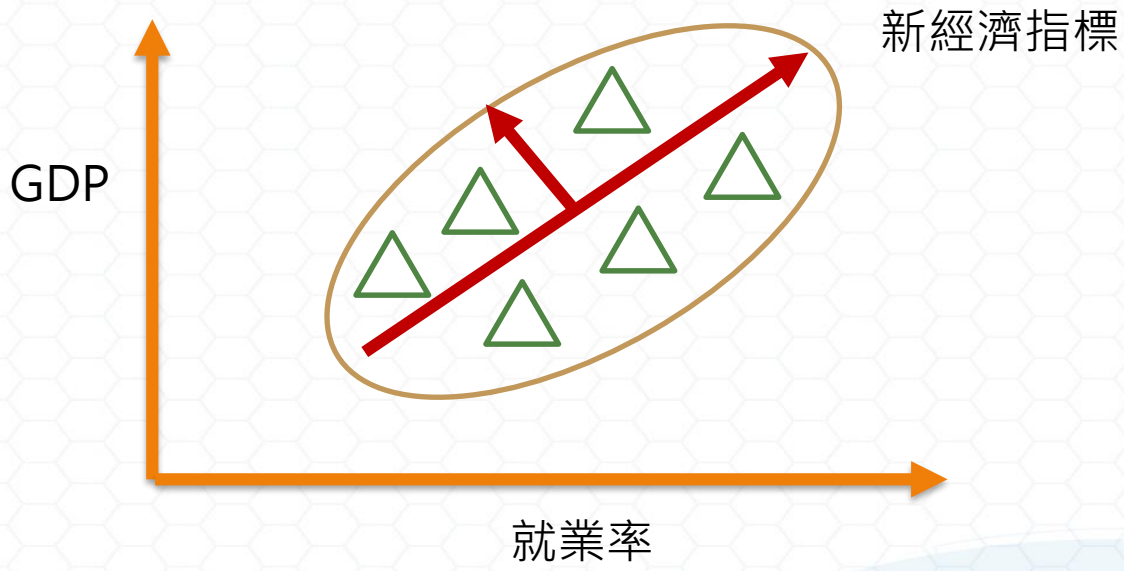
- 分群問題
  - 物以類聚（近朱者赤，近墨者黑）
  - e.g.將客戶分層
- 降低維度
  - 產生一有最大變異數的欄位元線性組合，可用來降低原本問題的維度與複雜度
  - e.g.濃縮用到的特徵，編纂成一個新指標

使用非監督式學習找出隱藏的架構

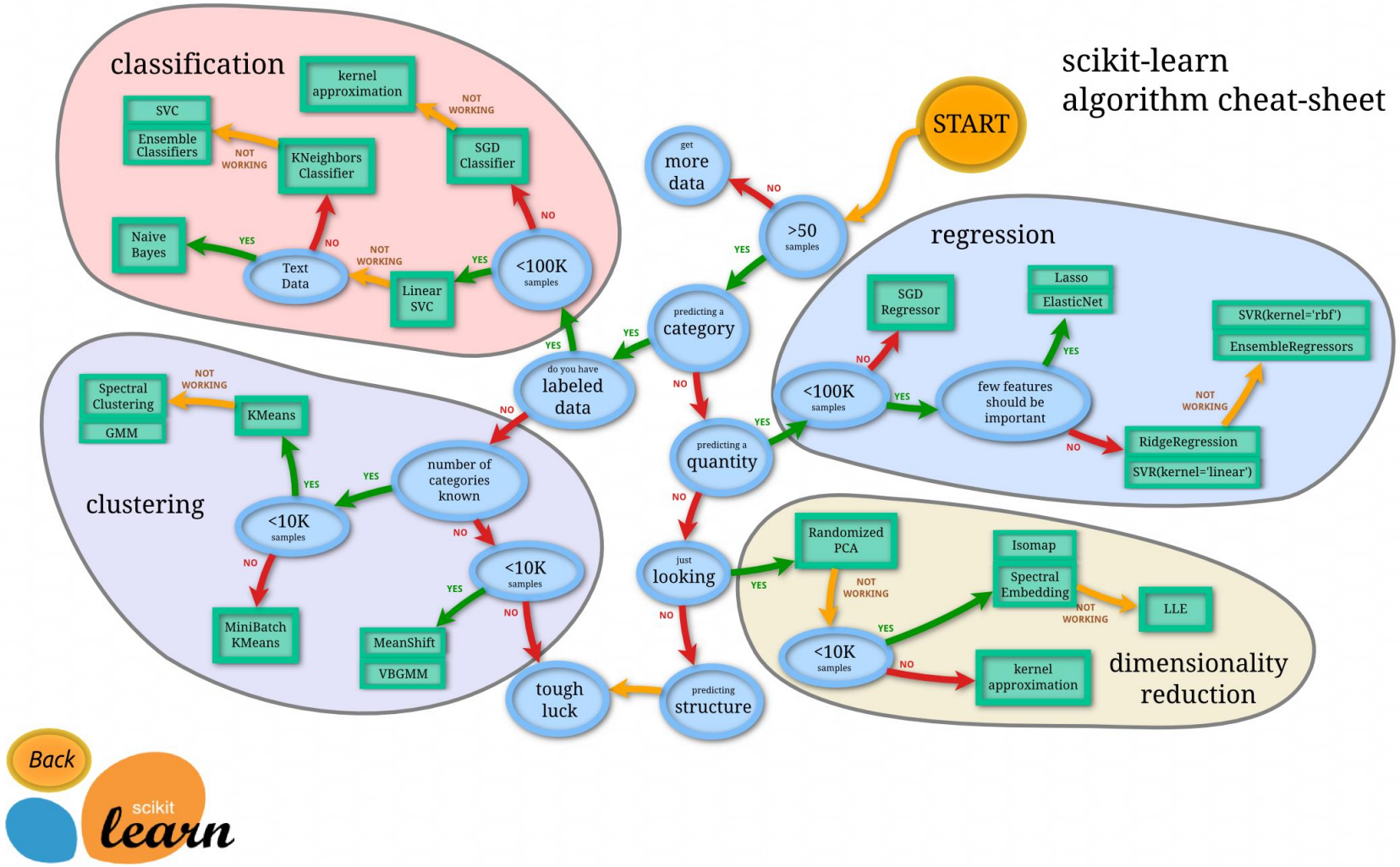
分群問題



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

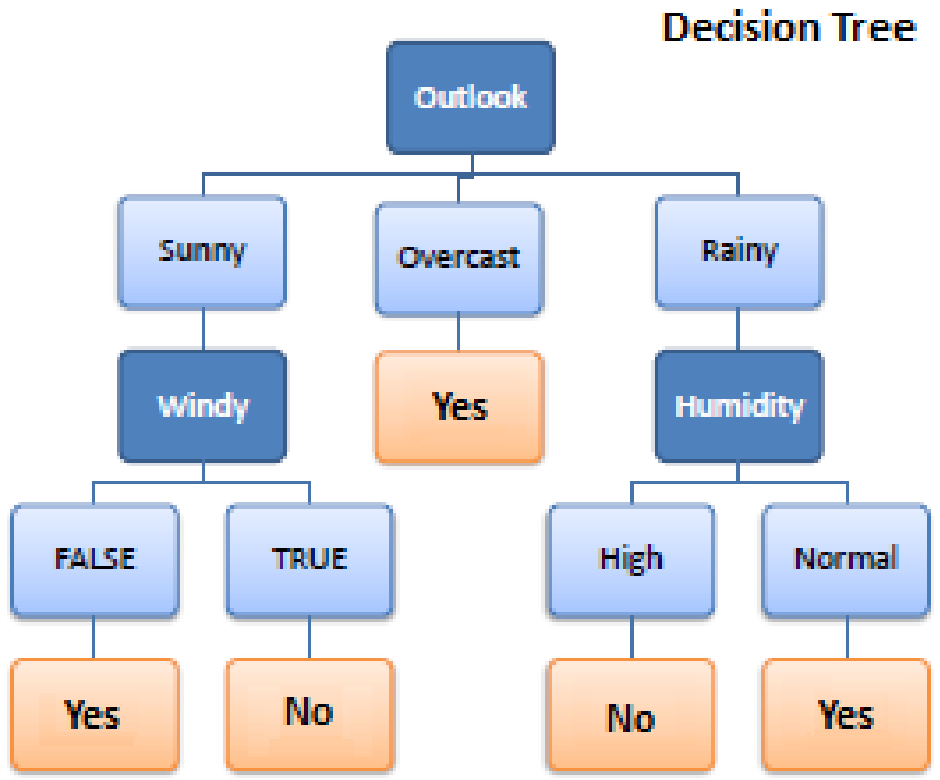
e.g. 針對一個化妝品廣告，對女性的吸引力可以給予權重90%，男性權重只有10%，以權重搭配個人點擊機率 (15%)可以算出對該使用者推薦的分數(或機率)

女性13.5%，男性1.5%

決策樹

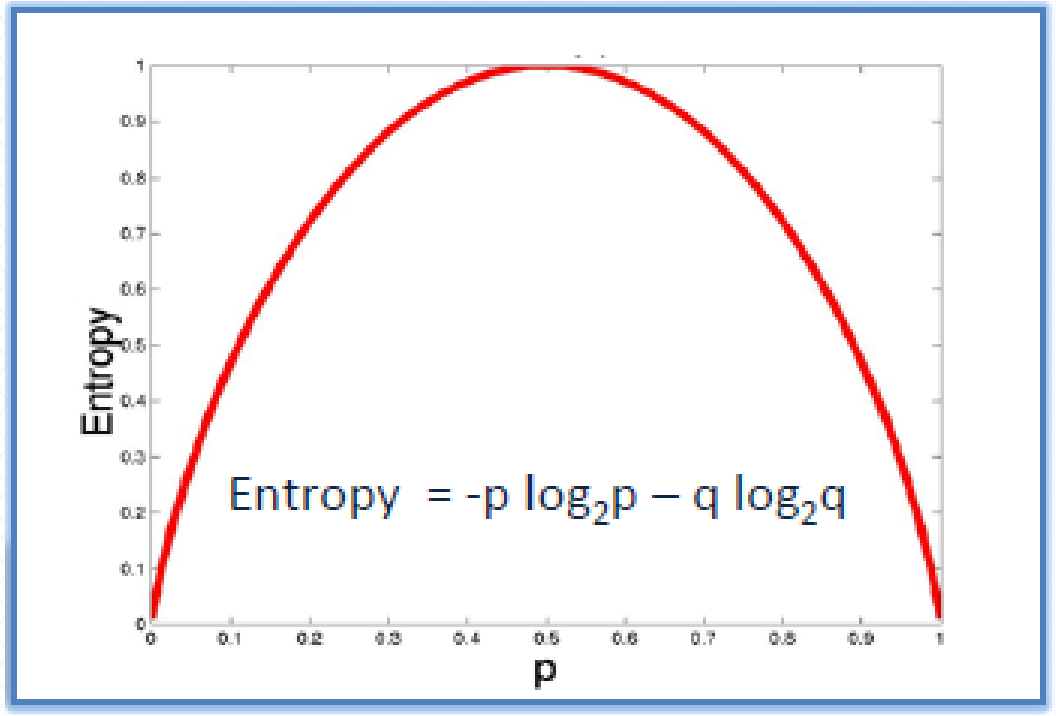
Predictors | Target

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



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



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

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

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

E(PlayGolf, Outlook) = **P**(Sunny)*__E__(3,2) + **P**(Overcast)*__E__(4,0) + **P**(Rainy)*__E__(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

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

$$\text{Entropy(PlayGolf)} = \text{Entropy}\,(5,9)$$

$$= \text{Entropy}\,(0.36, 0.64)$$

$$= -\,(0.36 \log_2 0.36) - (0.64 \log_2 0.64)$$

$$= 0.94$$

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(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

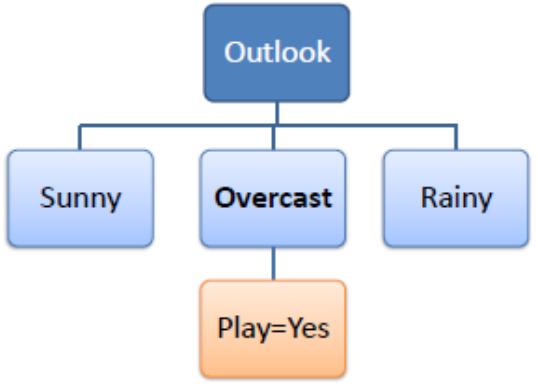
= 0.940 – 0.693 = 0.247

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			

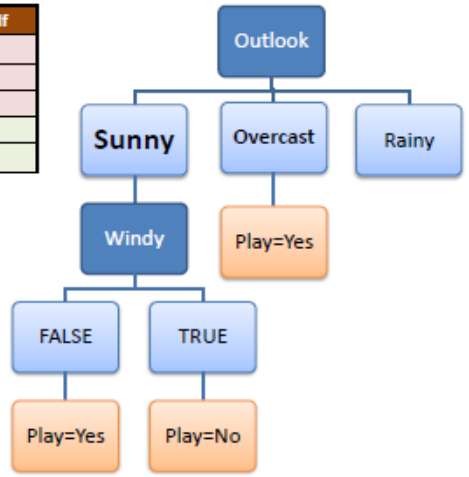
選擇子節點與分割節點

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

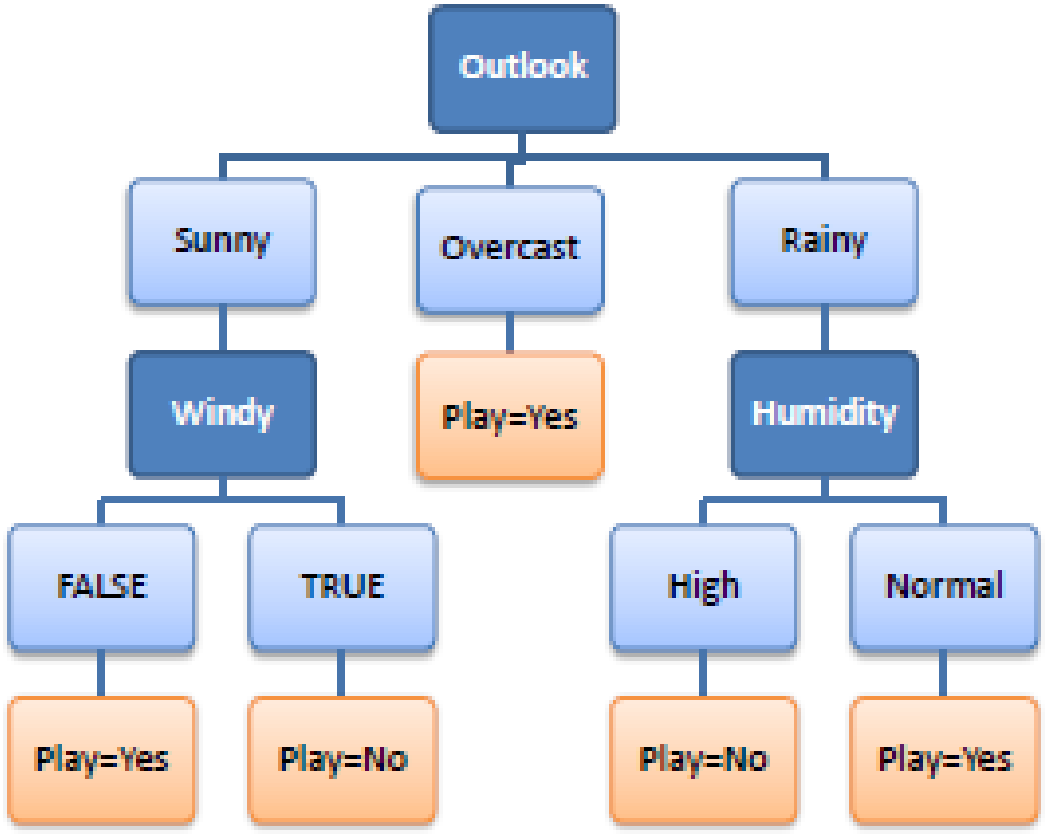
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



如何分類鳶尾花（iris）

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

Iris setosa

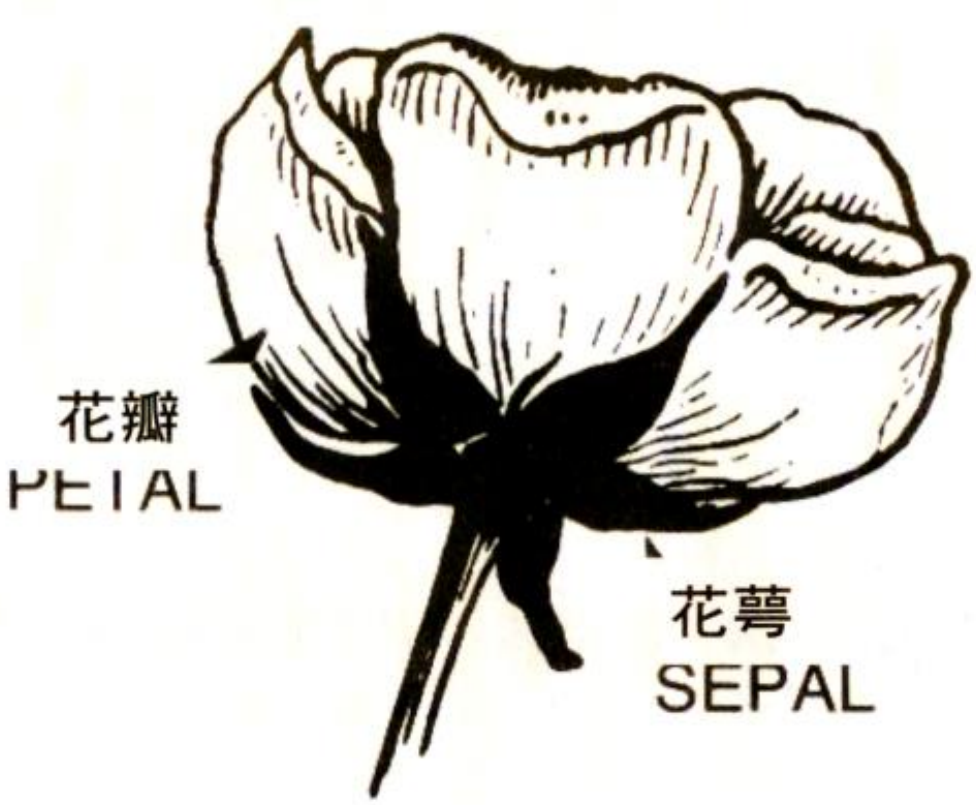
Iris versicolor

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. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*



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/

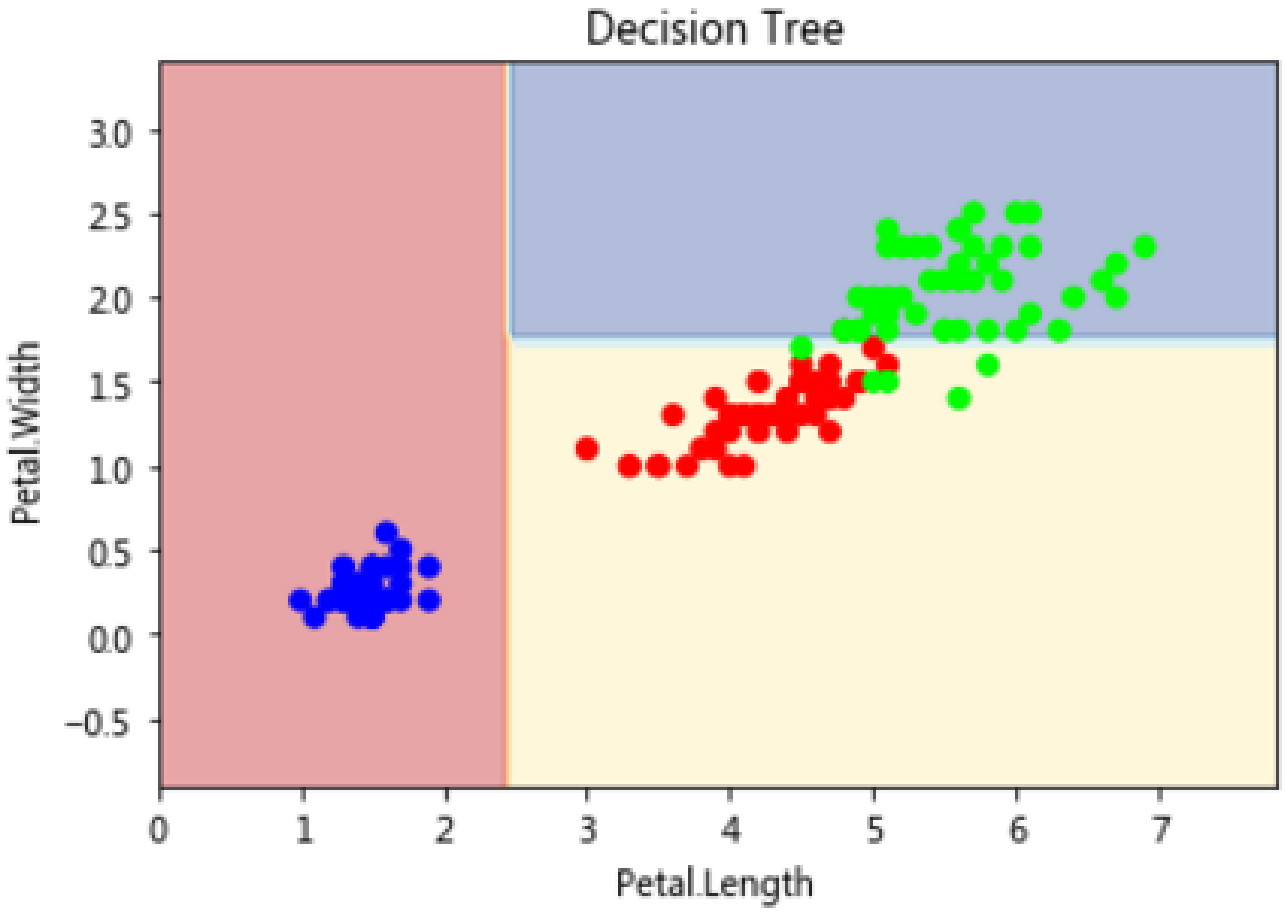
建立決策邊界（1）

```
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()
```

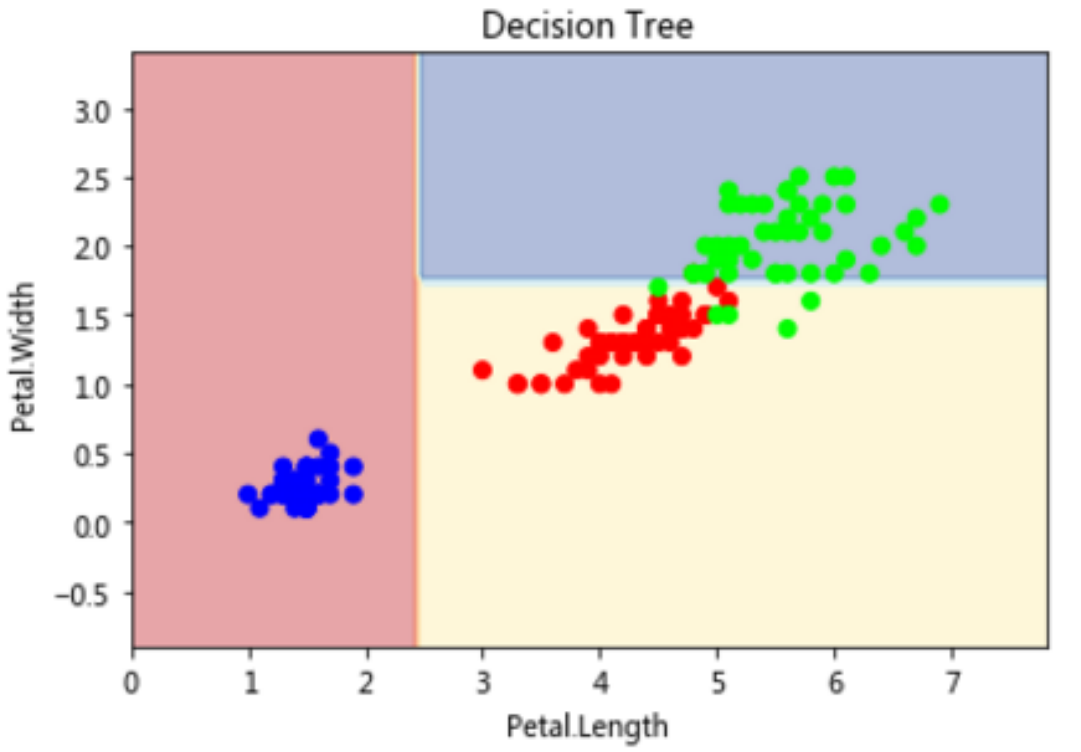
決策樹模型


Decision Tree
Petal.Width
Petal.Length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
6
7

監督式學習 – 線性模型

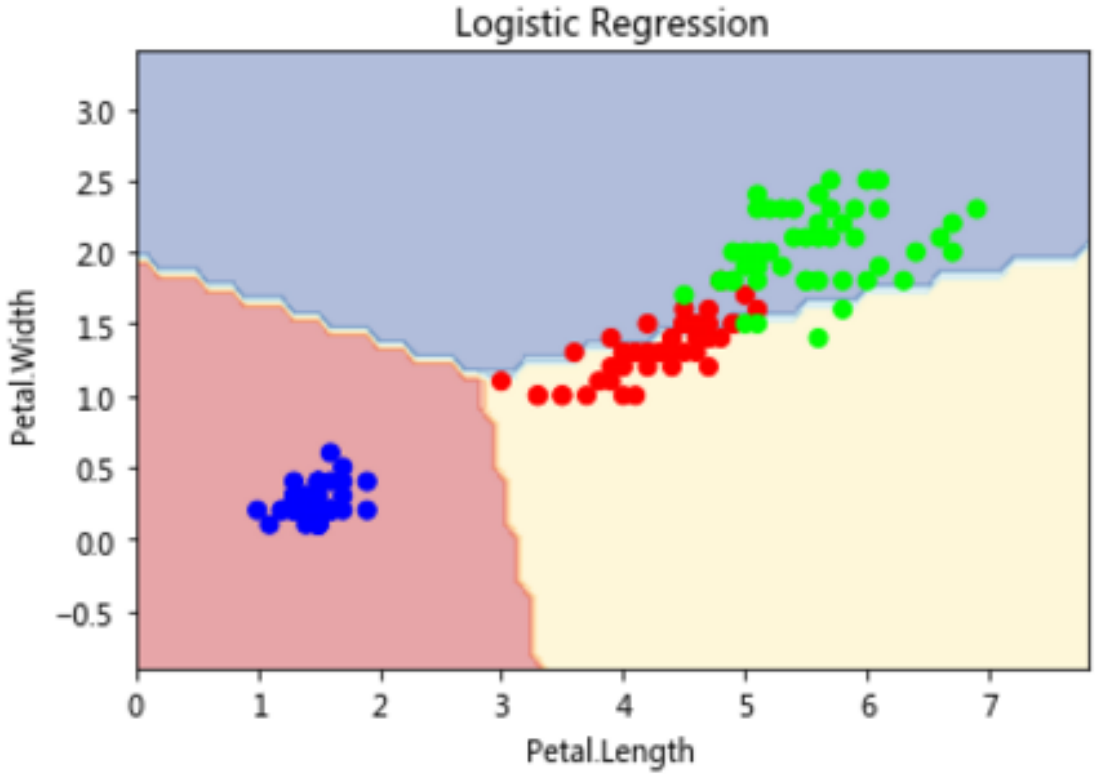
決策樹模型

- 每次根據一個變數取決分隔



線性分類法

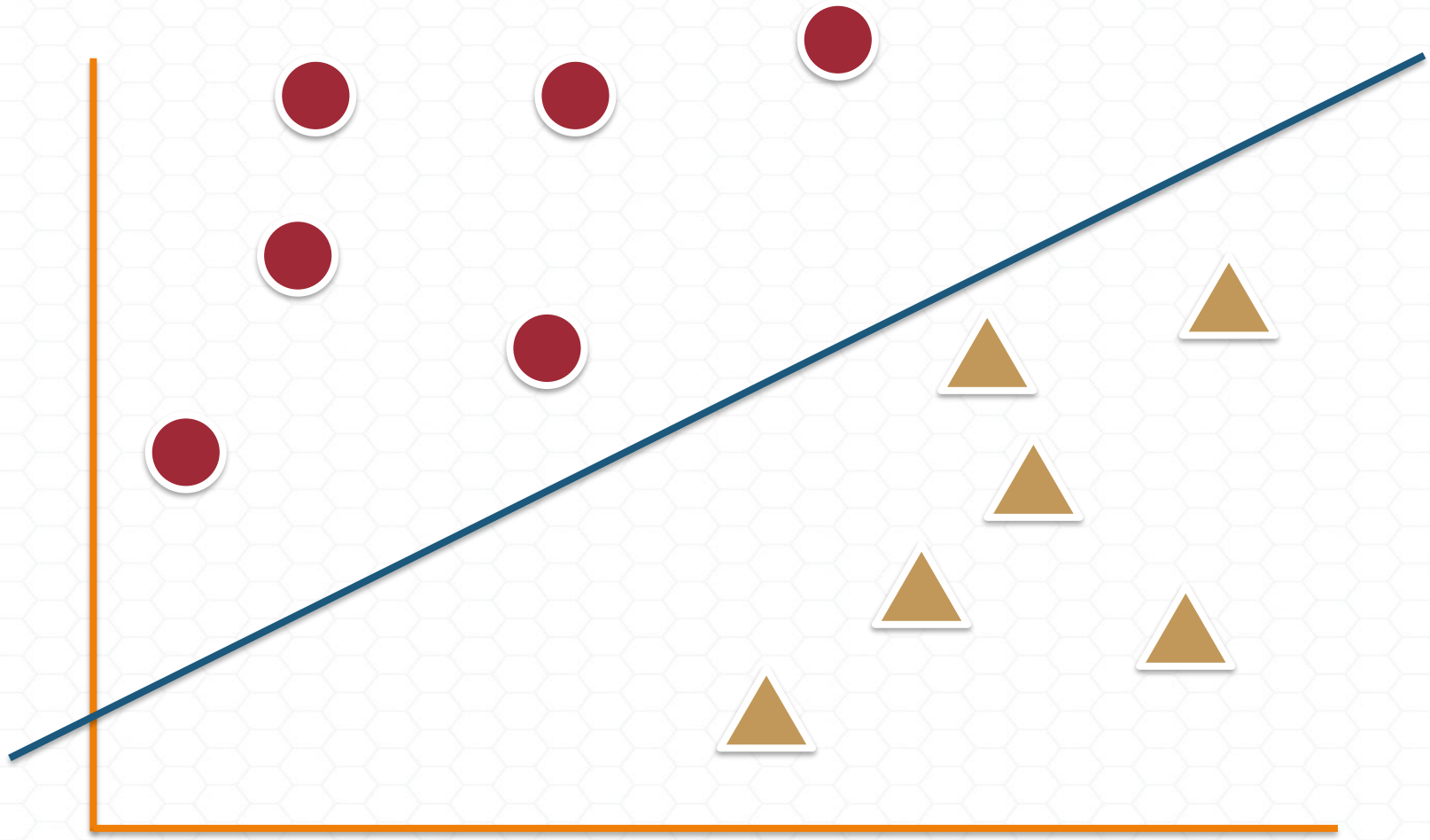
- 線性判別
  - 如何切斜的刀



將模型配適到資料

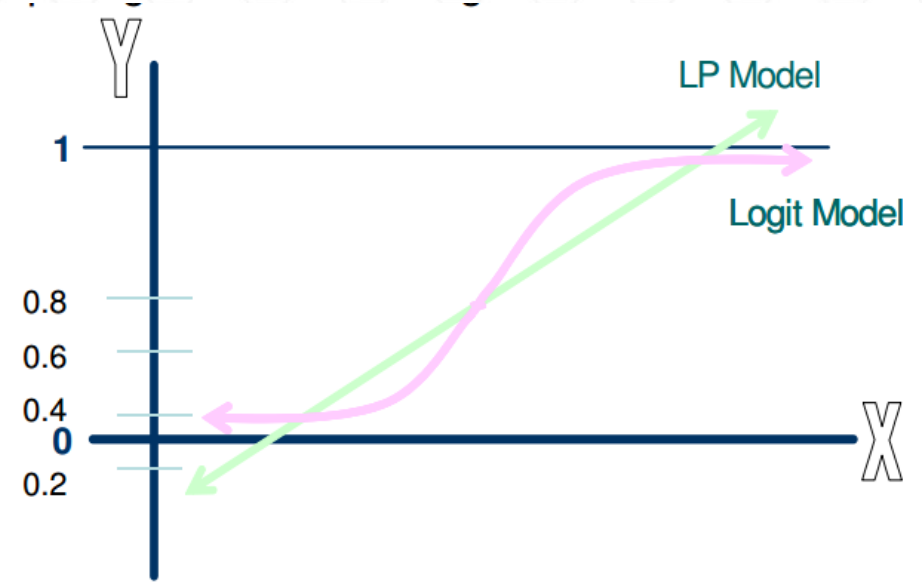
- 參數學習(Parameter Learning)
  - 或稱為「參數化建模」(Parametric Modeling)
  - 以未確定的數值參數指定模型結構，依特定的訓練資料算出最佳參數值
  - 先根據專業知識挑選屬性，利用演算法調整參數，讓模型盡可能符合資料

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

Logistic Regression

$$\log it(y) = \ln(odds) = b_0 + b_1 X_1 + \varepsilon$$

- 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

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

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

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)

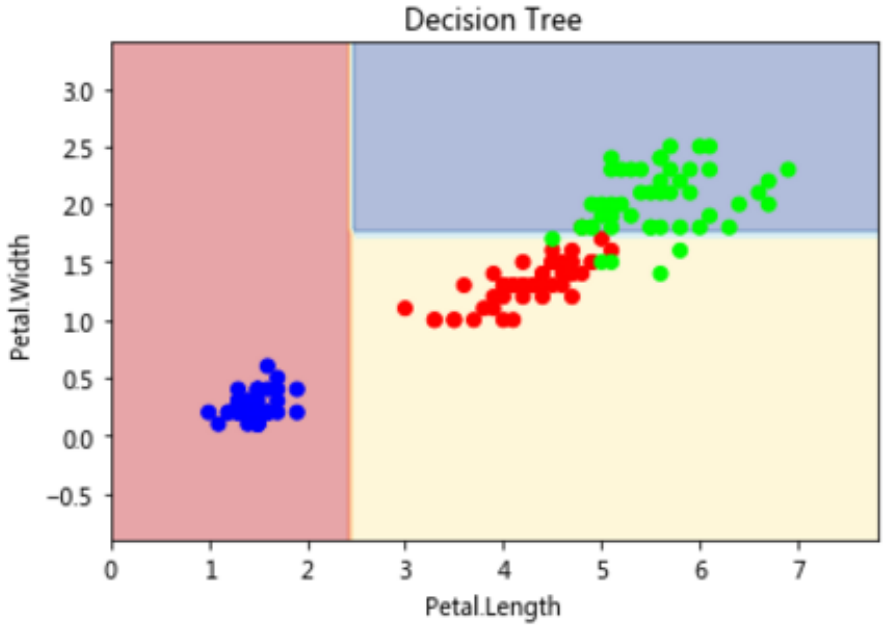
```
import numpy as np
import matplotlib.pyplot as plt

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)

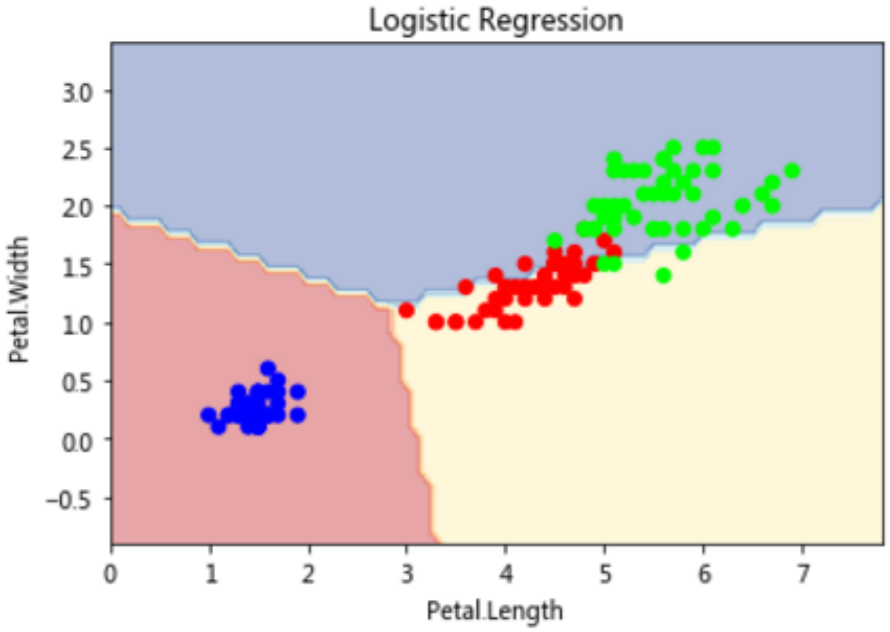
```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

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



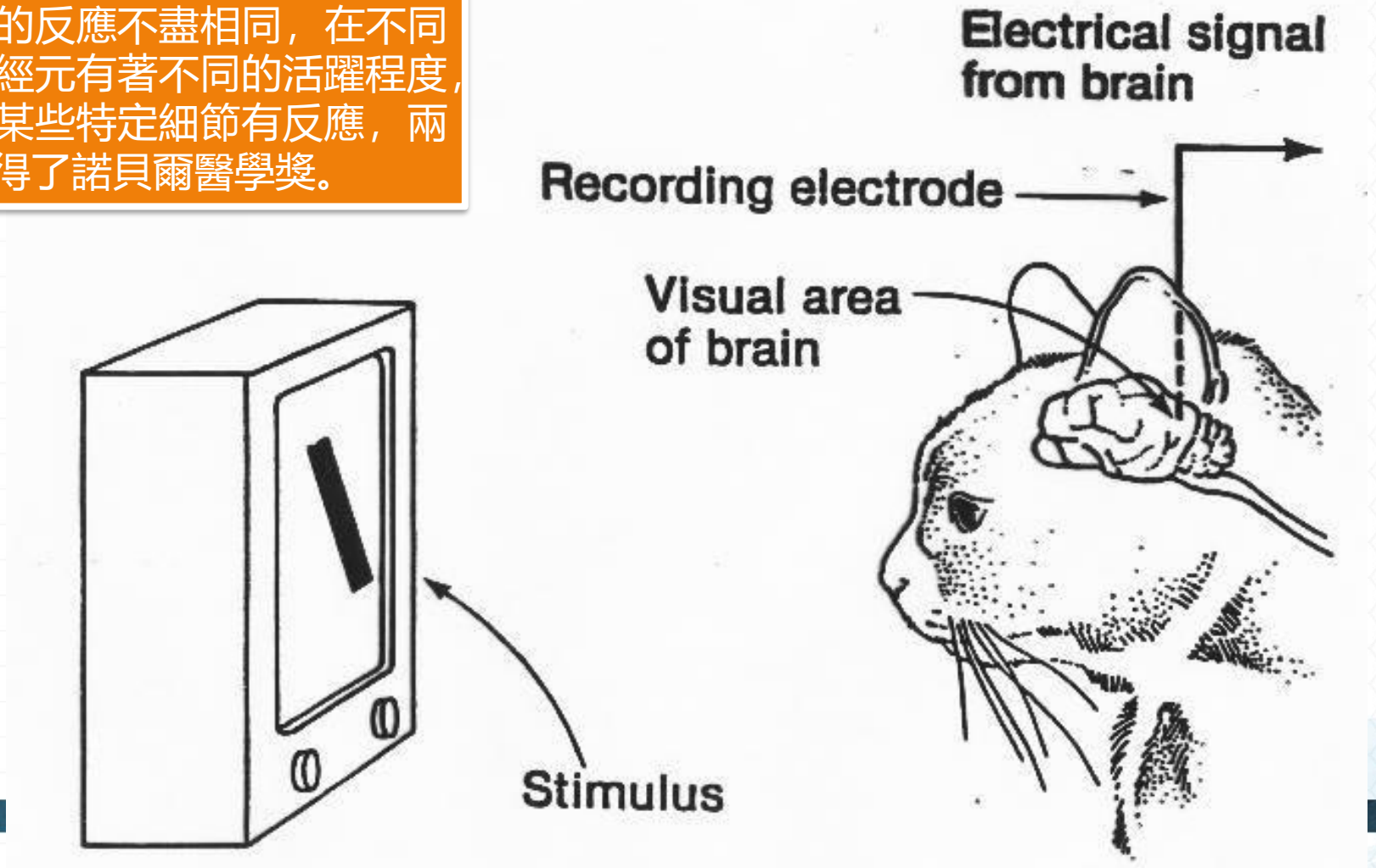
邏輯回歸模型-線性模型



類神經網路

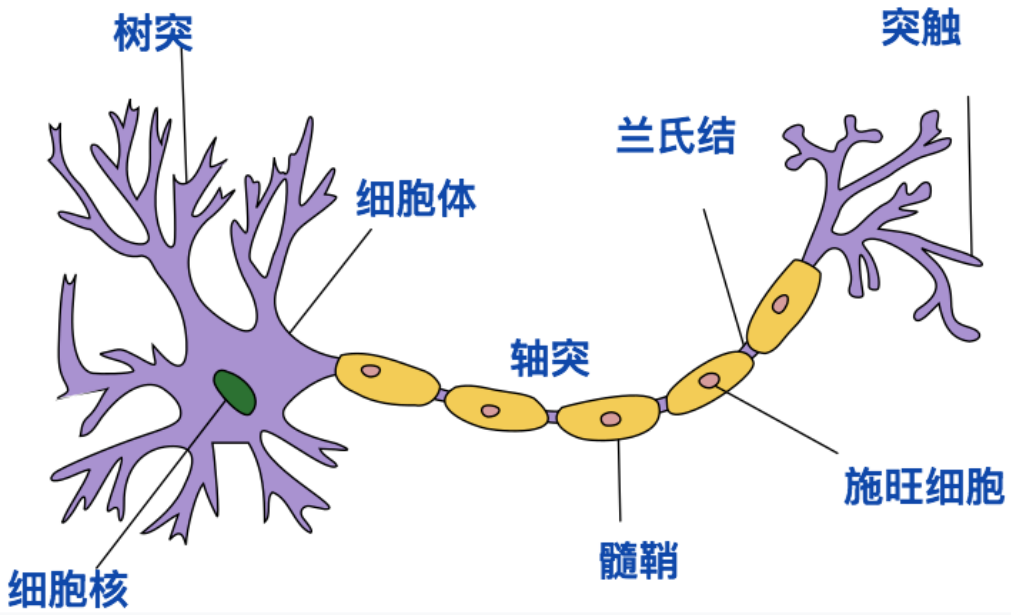
啟發於神經元的實驗

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

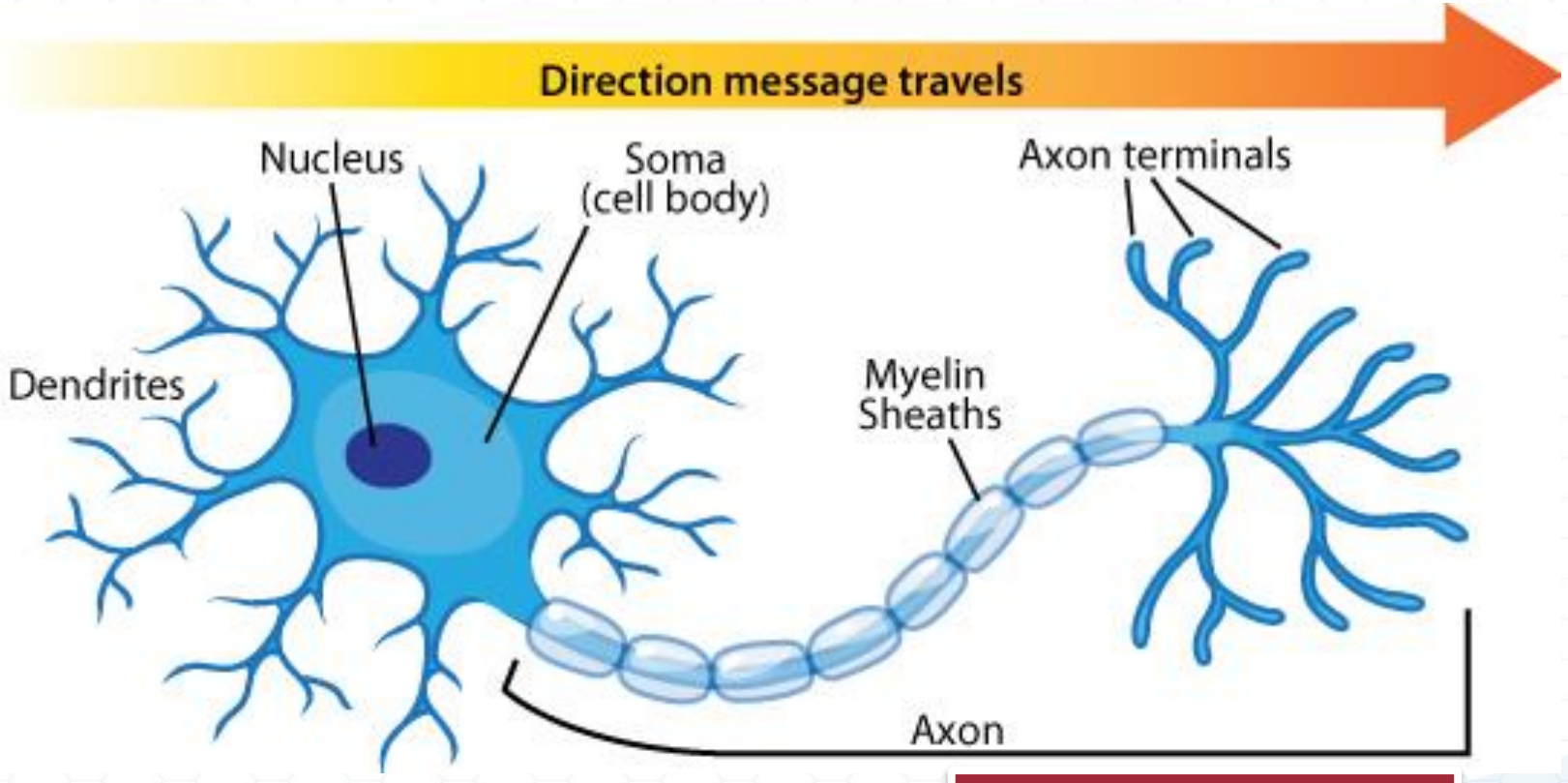
- 軸突（Axon）：連接在神經細胞核上，用來傳送由神經細胞核產生的信號至其它的神經細胞中。
- 樹突（Dendrites）：神經樹分為兩種：輸入神經樹及輸出神經樹。在圖1中左邊接到神經核的神經樹是用來接收其他神經細胞傳來的信號，稱為輸入神經樹。
- 突觸（Synapse）：輸入神經樹和輸出神經樹相連接的點稱為神經節



典型神经元的结构

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



如同數學函數的產出

類神經網路跟神經元有關但跟人腦無關

Yann LeCun

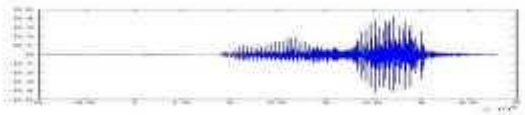
12月11日 9:41 ·

"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.

https://www.facebook.com/yann.lecun

萬用函數

- 如果能夠模擬神經元產生一個萬用函數呢?

$f($  $)=$ “How are you”

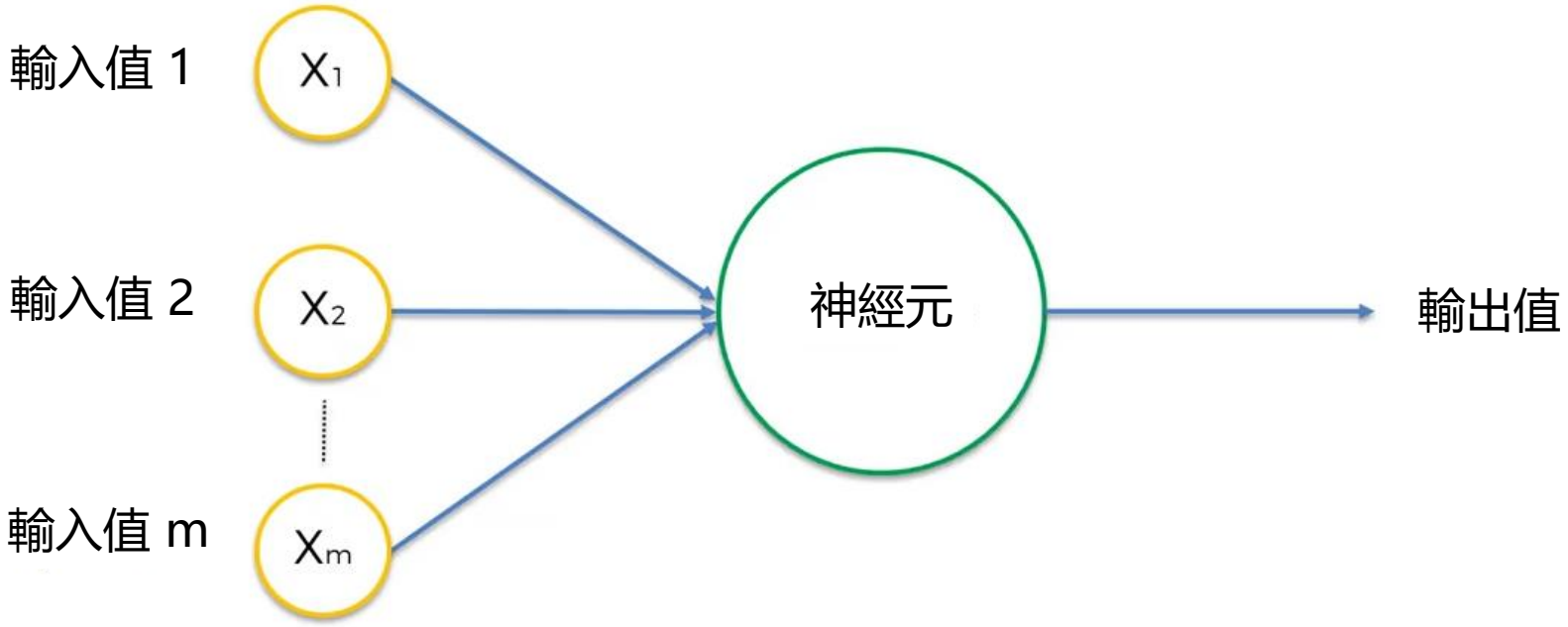
$f($  $)=$ “Cat”

$f($  $)=$ “5-5”

是否就可以像人類一樣解決所有問題了?

感知機

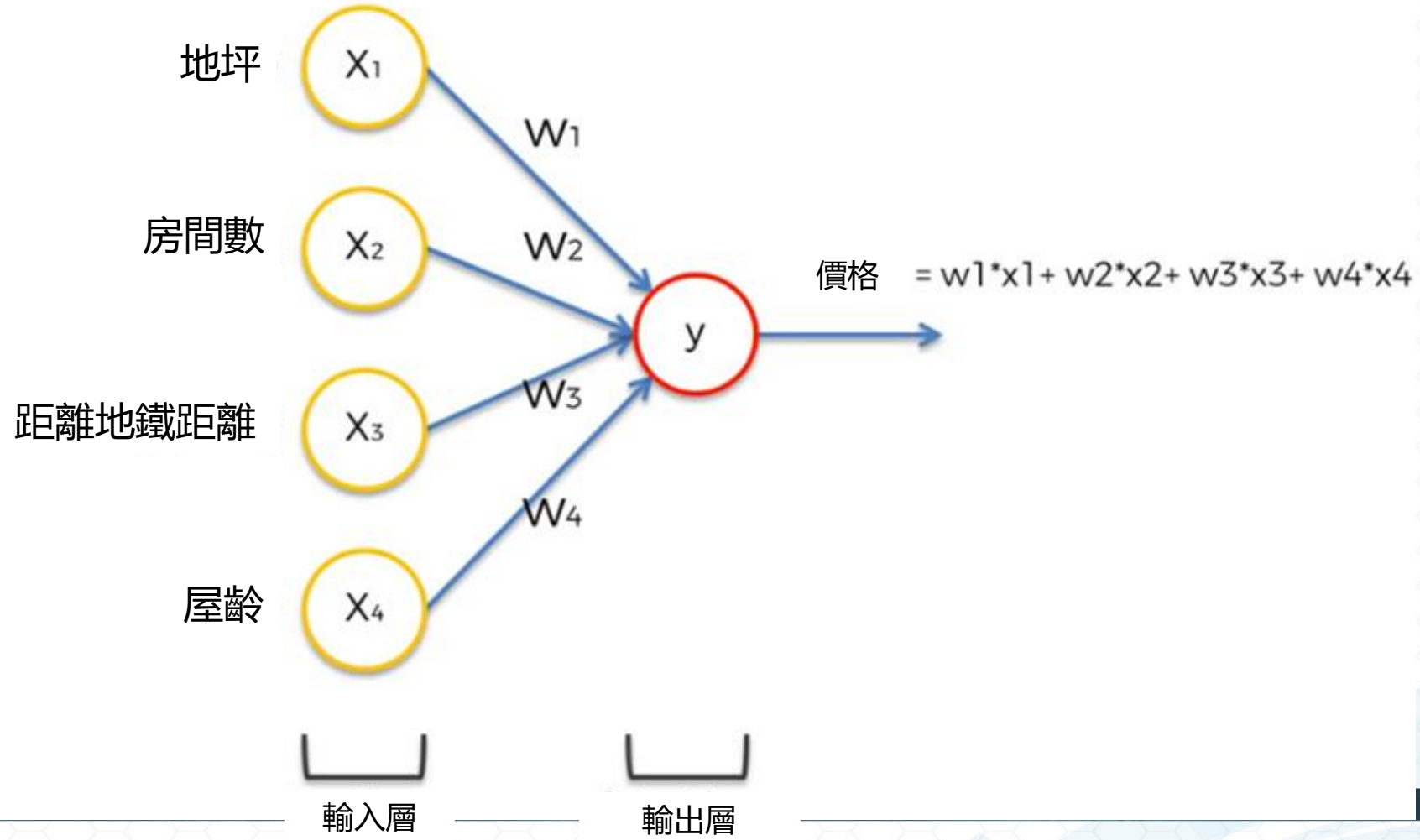
1957 年，Rosenblatt 提出了感知機(Perceptron)模型



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

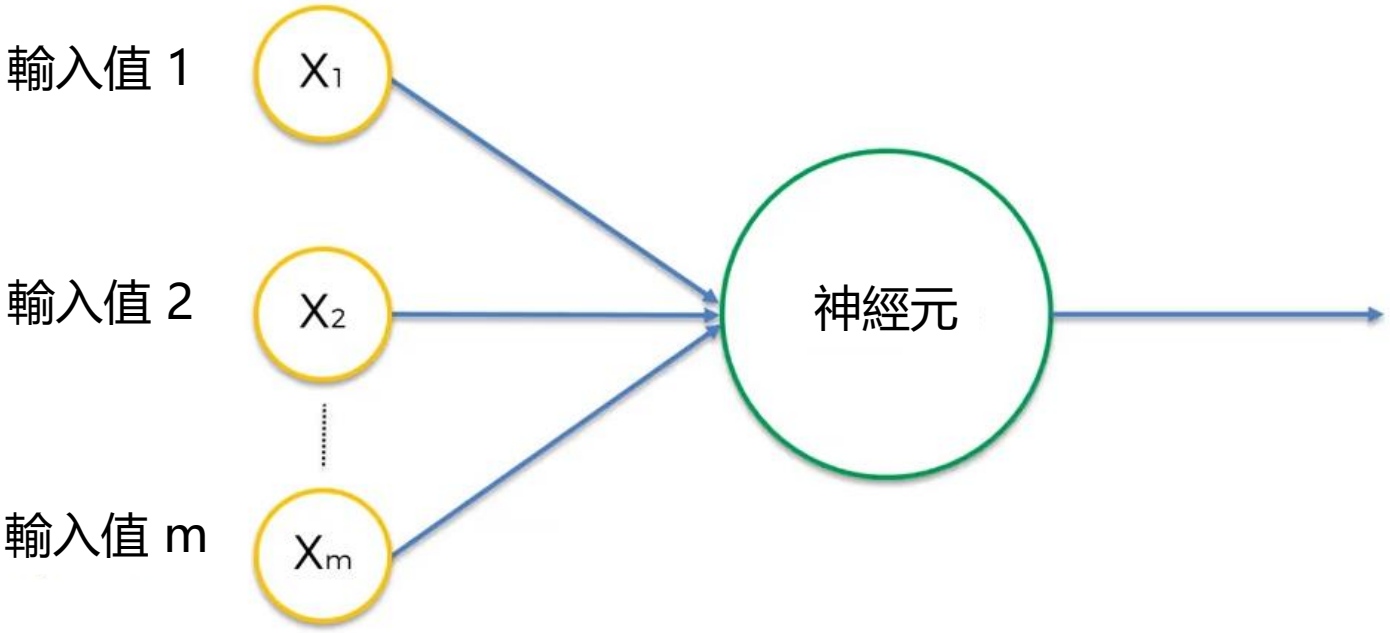
單層感知機

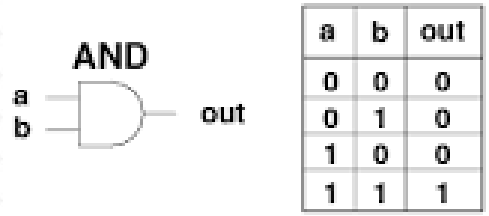
- 假設要預測房屋價格



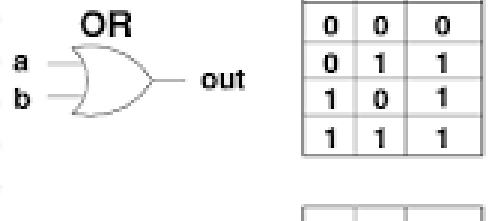
邏輯門

- 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

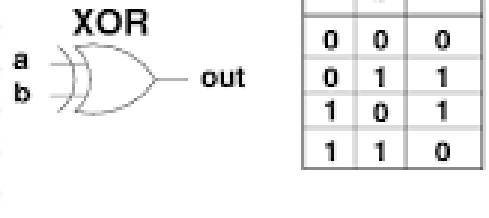




a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



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

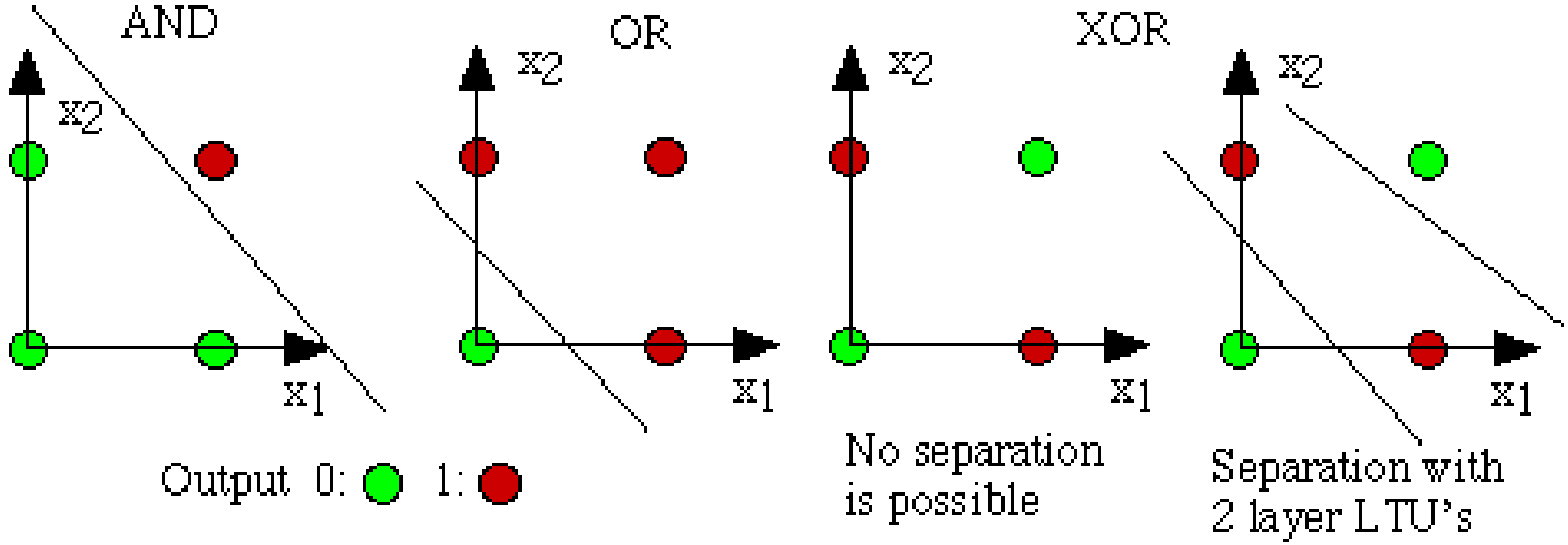


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

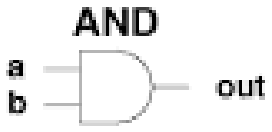
in	out
0	1
1	0

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

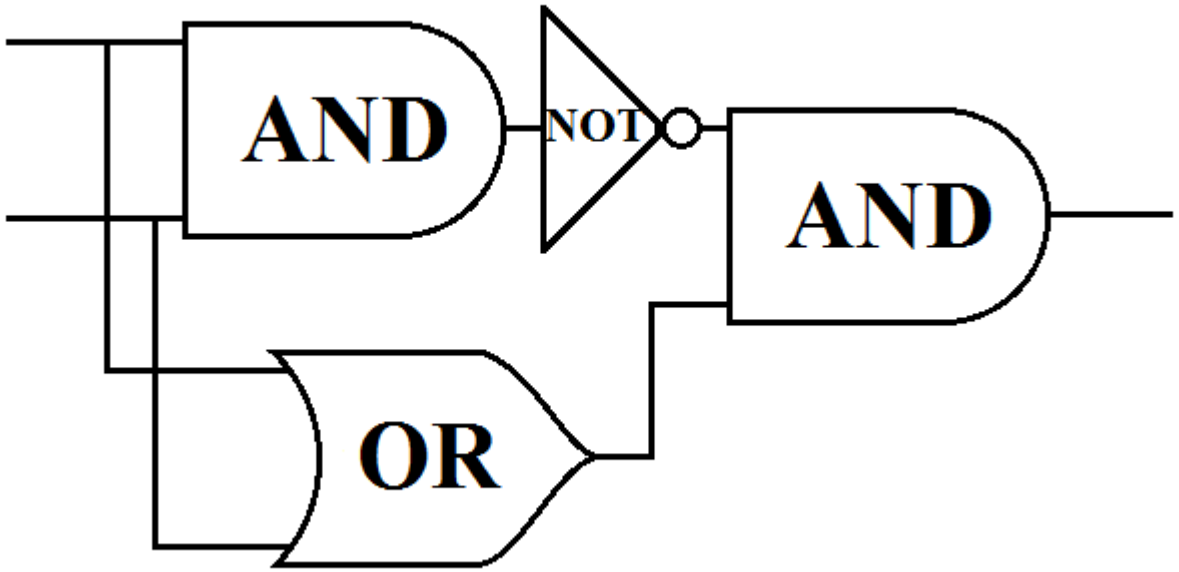


a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

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

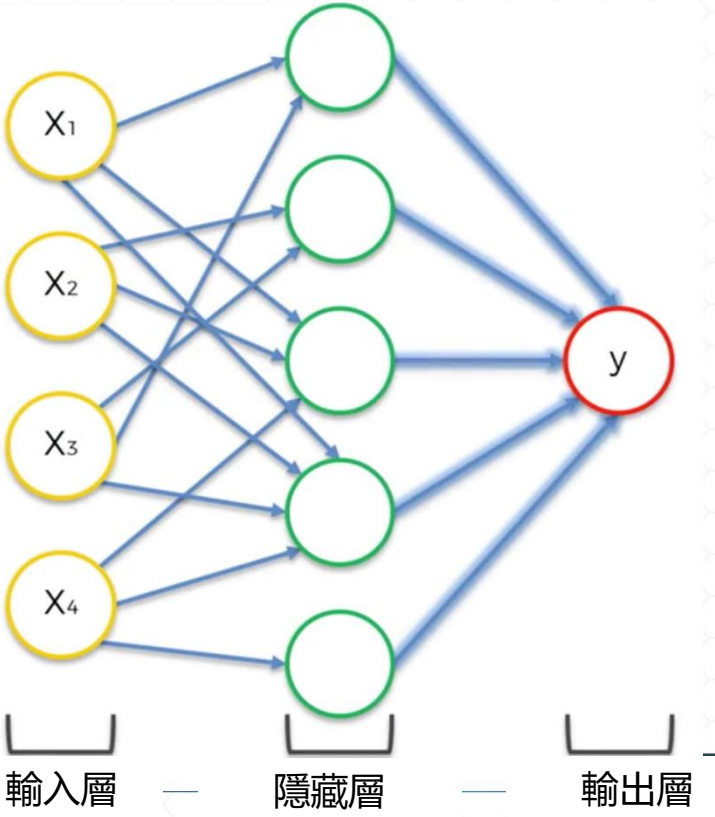
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

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)**

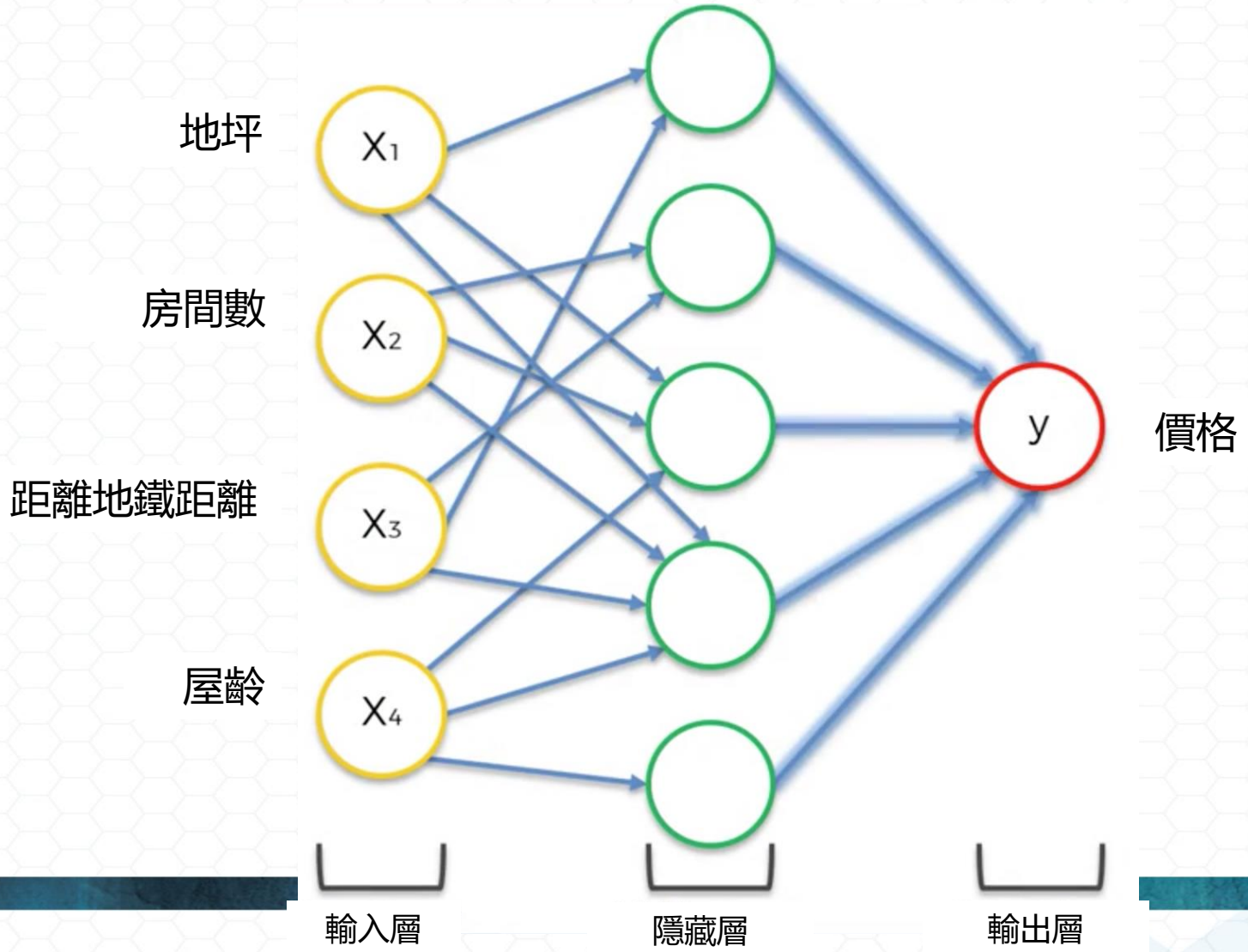


Geoffrey Hinton - 深度學習之父

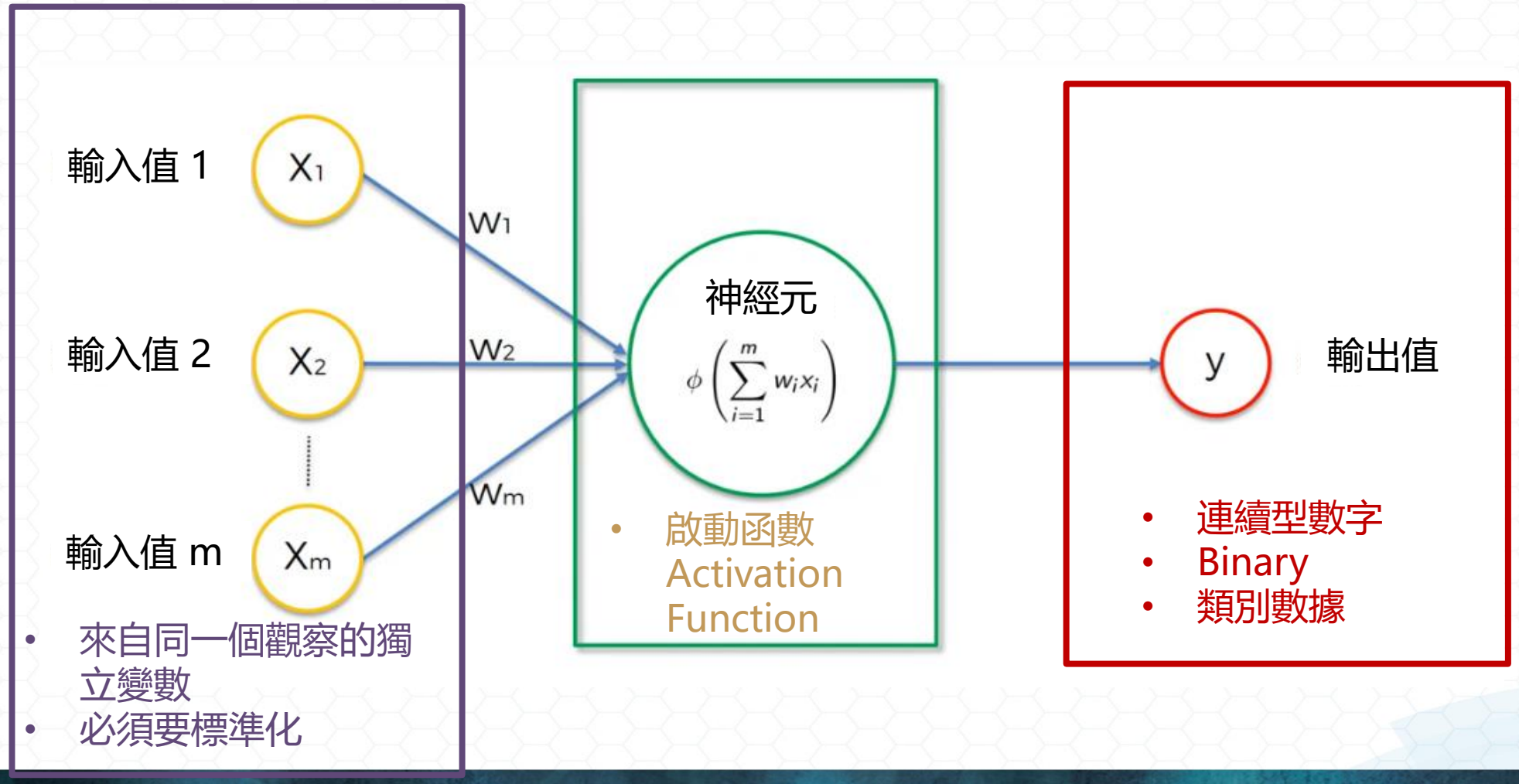
http://www.cs.toronto.edu/~hinton/

建構神經網路

神經網路

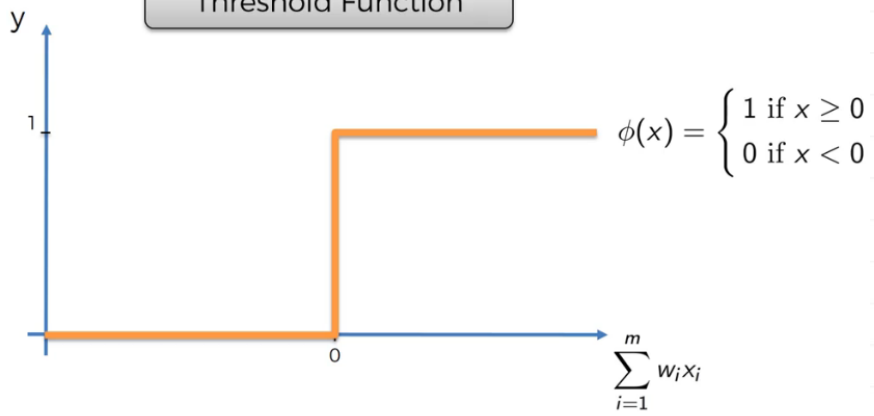


使用激活函數調整輸出

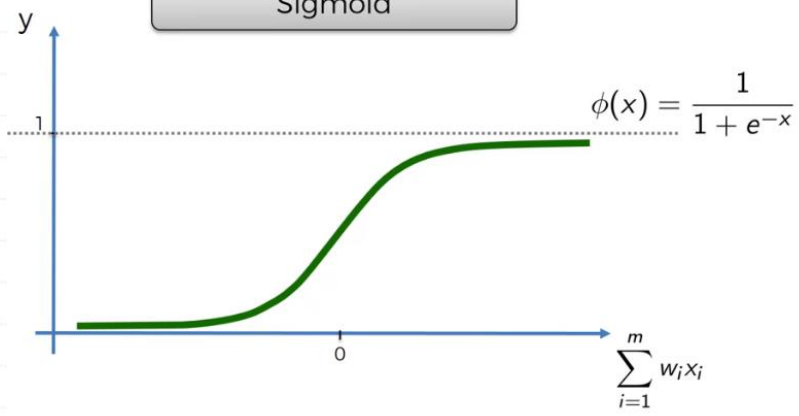

輸入值 1
X1
輸入值 2
X2
輸入值 m
Xm
W1
W2
Wm
神經元
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
y
輸出值
• 來自同一個觀察的獨立變數
• 必須要標準化
• 啟動函數 Activation Function
• 連續型數字
• Binary
• 類別數據

激活函數(Activation Function)

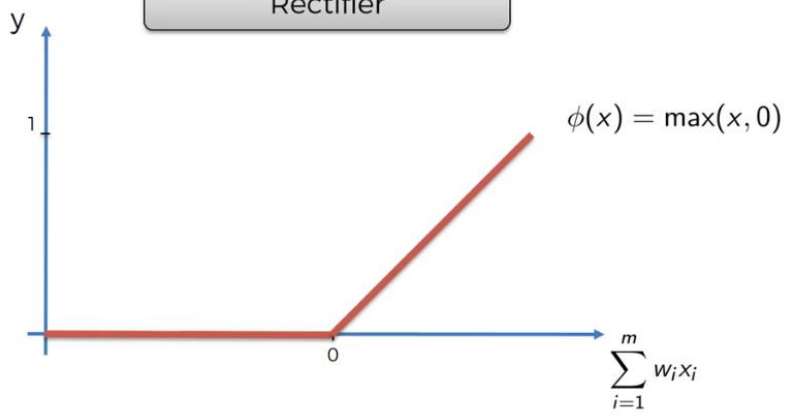
Threshold Function



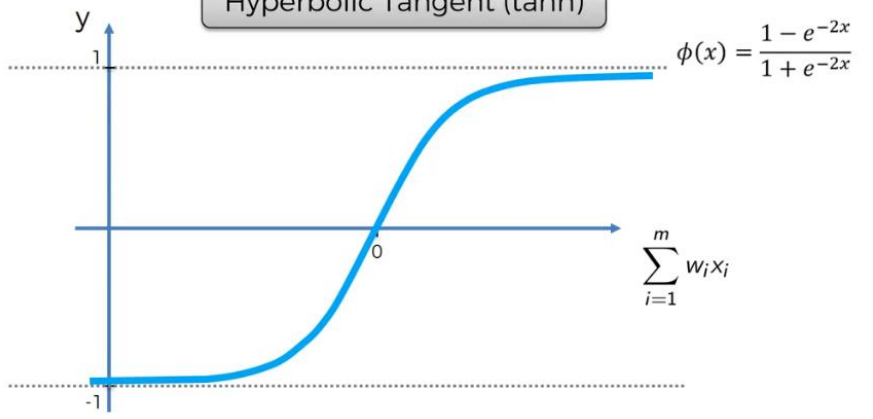
Sigmoid



Rectifier

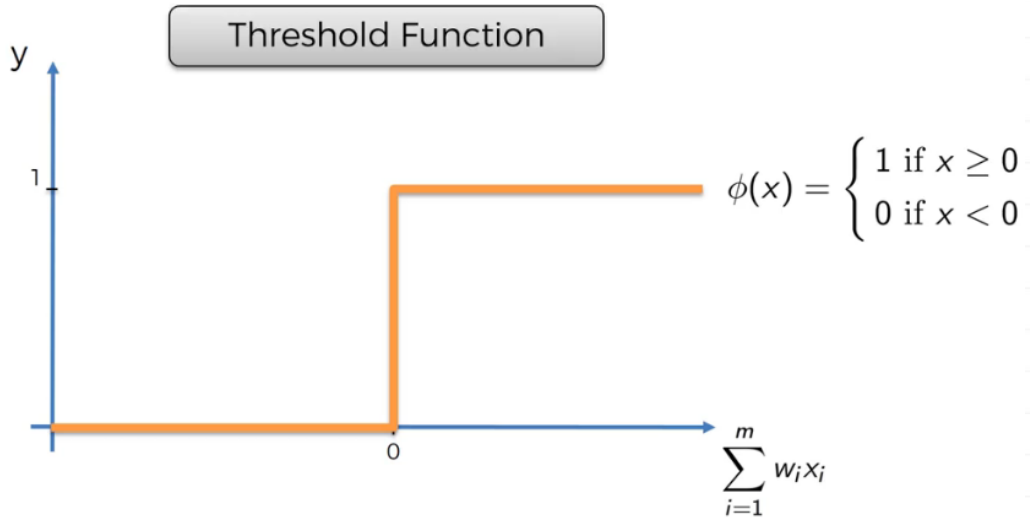


Hyperbolic Tangent (tanh)



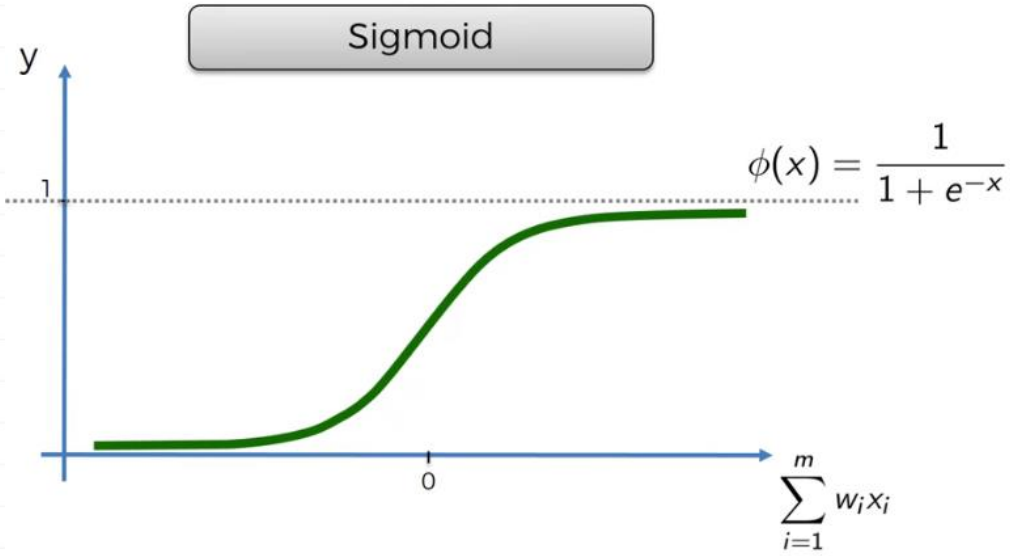
Threshold Function

```
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

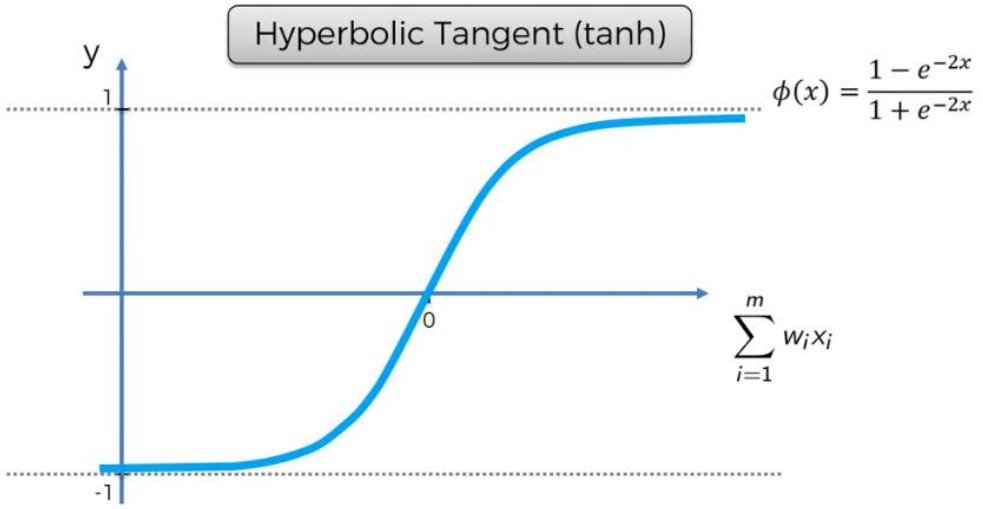
缺點:
- 容易發生梯度消失問題
- 中心點不為0

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)
```

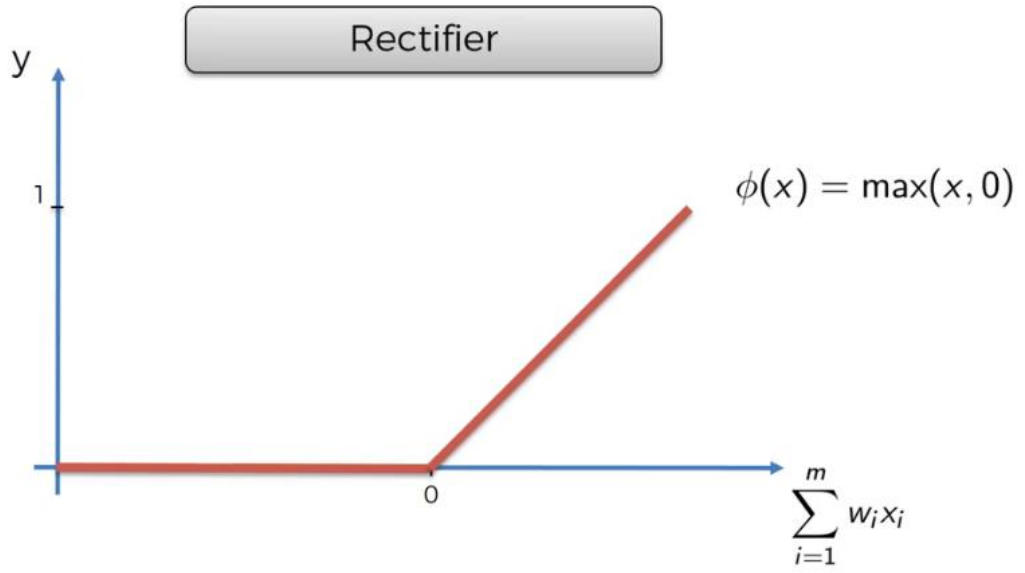


優點:
- 比Sigmoid函數收斂速度更快
- 輸出以0為中心

缺點:
- 容易發生梯度消失問題

ReLU Function

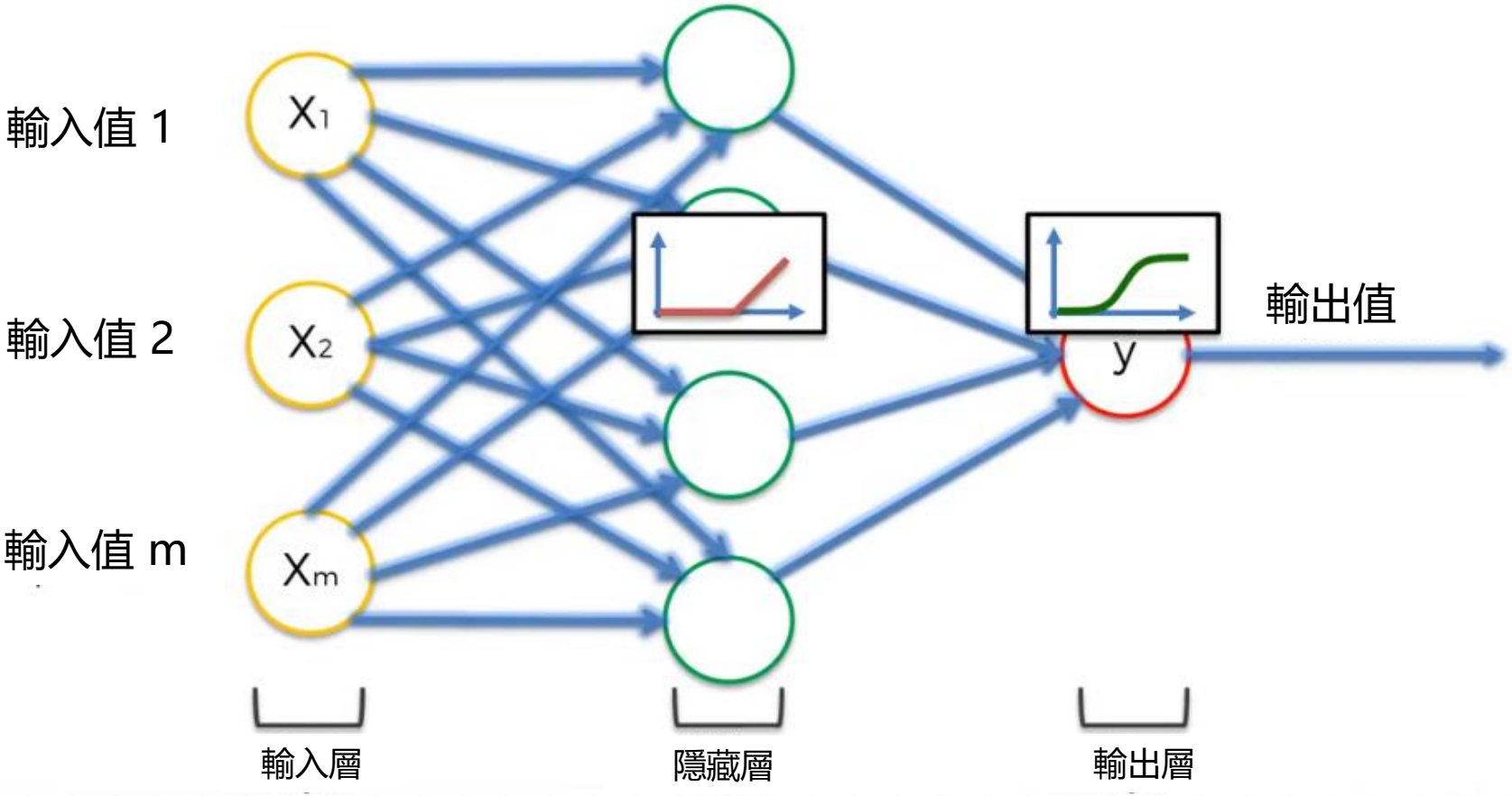
```
def relu_function(x):
    return np.maximum(0,x)

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



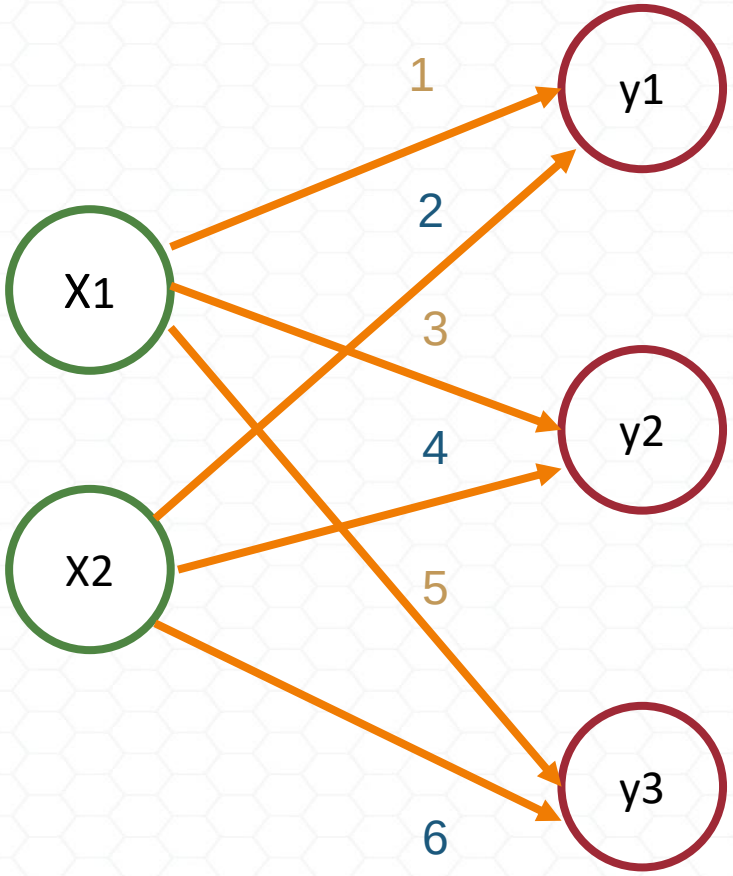
優點:
- 快速收斂
- 解決梯度消失問題

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）

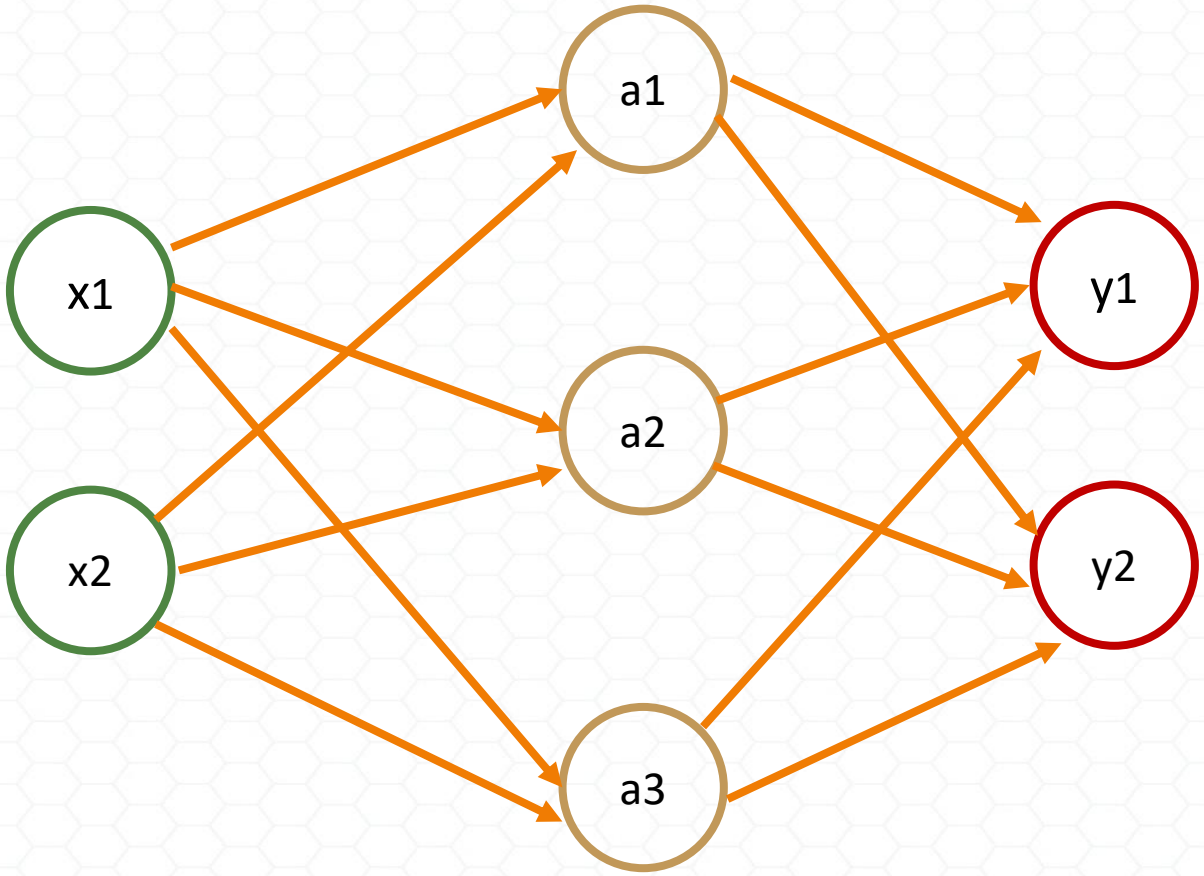


X	W	=	Y
2	2X3		3

單層神經網路前向傳播過程（Forward Propagation）

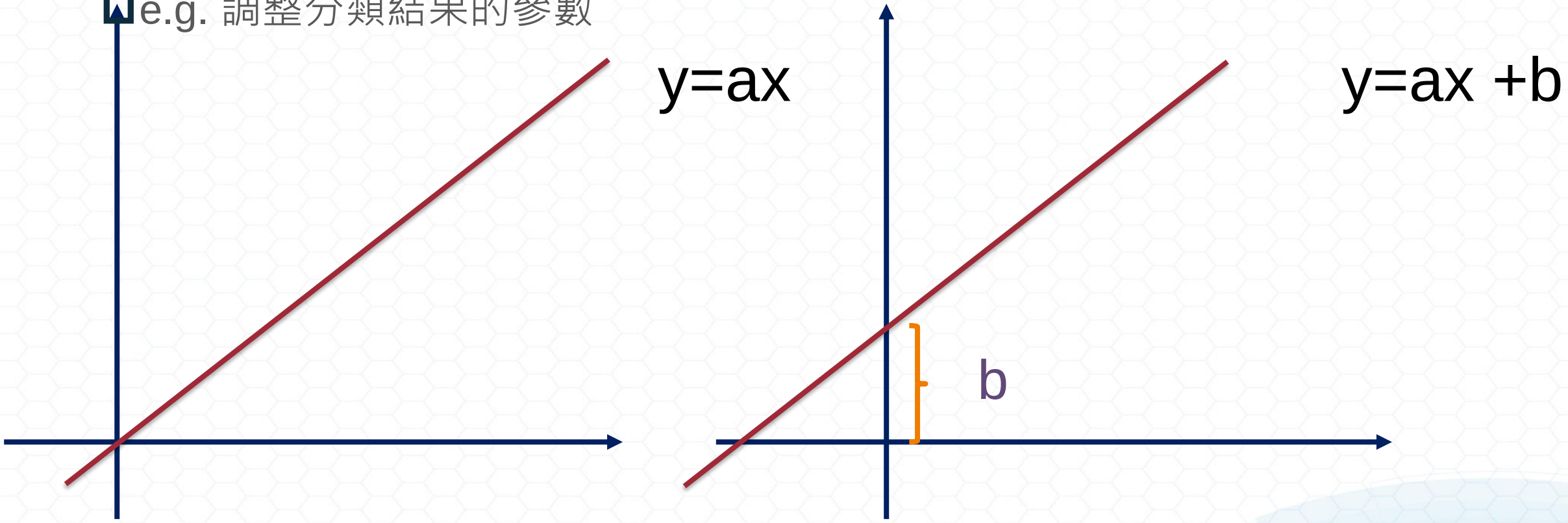
```
import numpy as np
X = np.array([1,2])
W = np.array([[1,3,5],[2,4,6]])
Y = np.dot(X,W)
Y
```

兩層神經網路前向傳播過程（Forward Propagation）

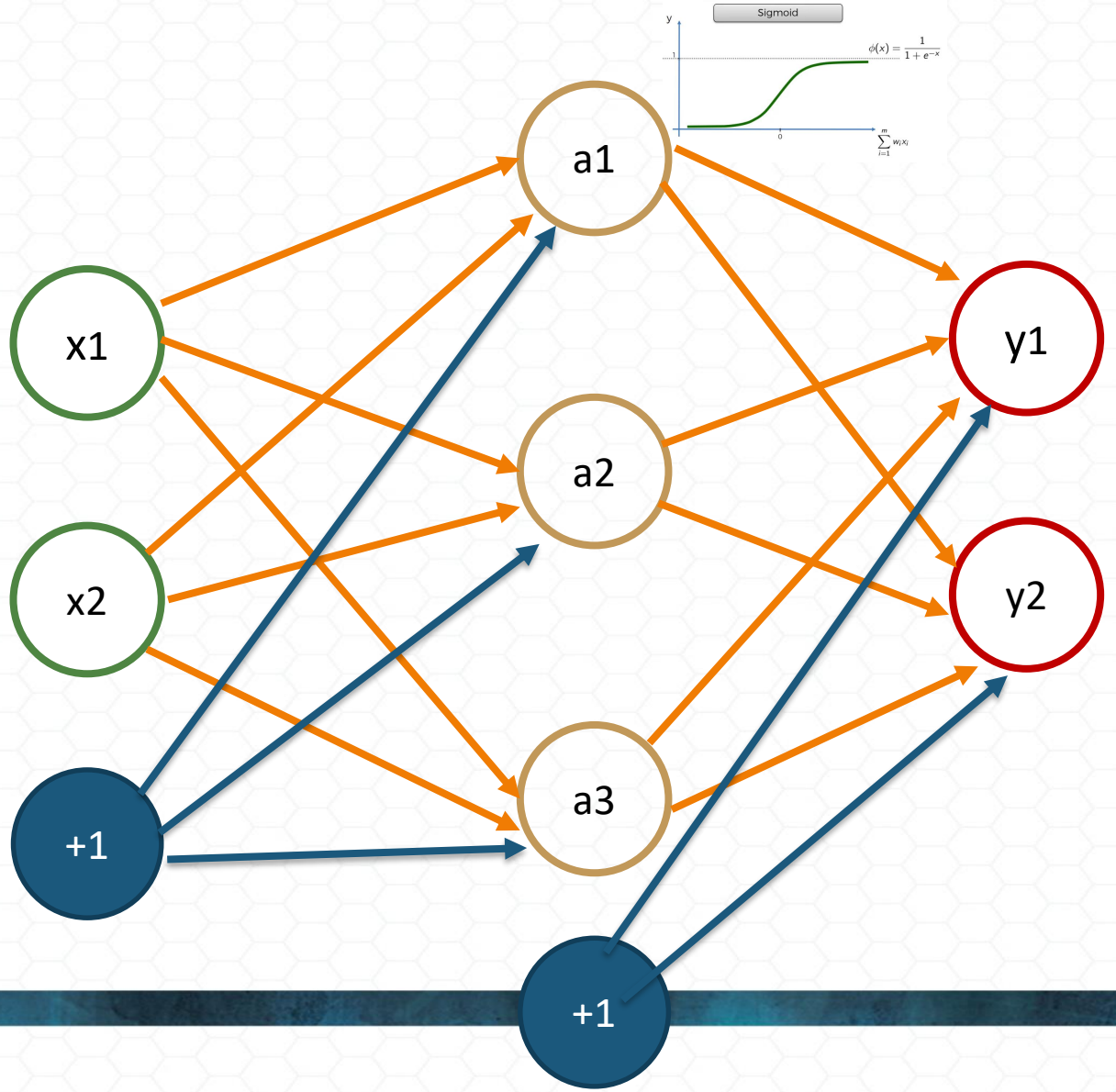


增加偏倚（bias）

- 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



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])
```

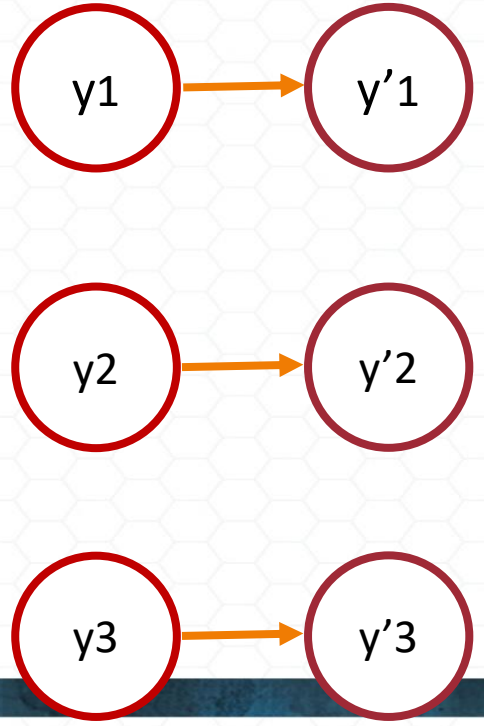
計算傳遞過程

```
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

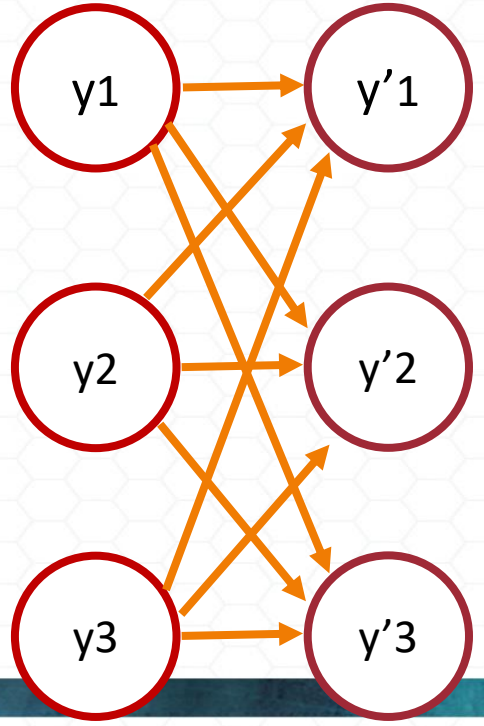
Identity Function

$$y'_k = y_k$$



Softmax Function

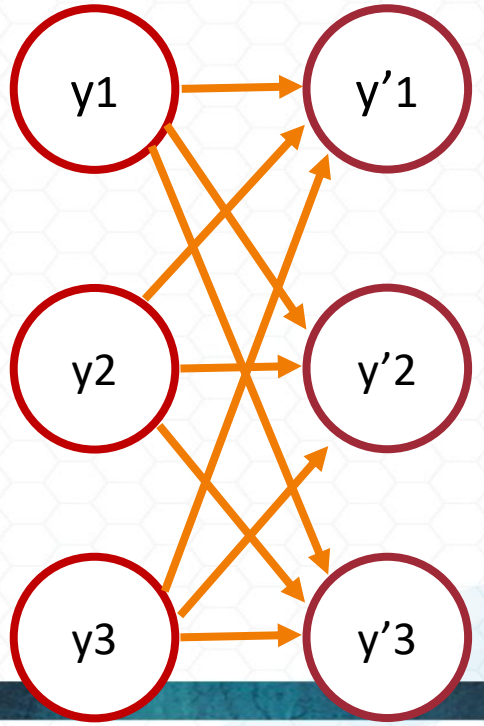
$$y'_k = \frac{e^{y_k}}{\sum_{i=1}^{n} e^{y_i}}$$



Softmax

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

softmax_function(y)
```

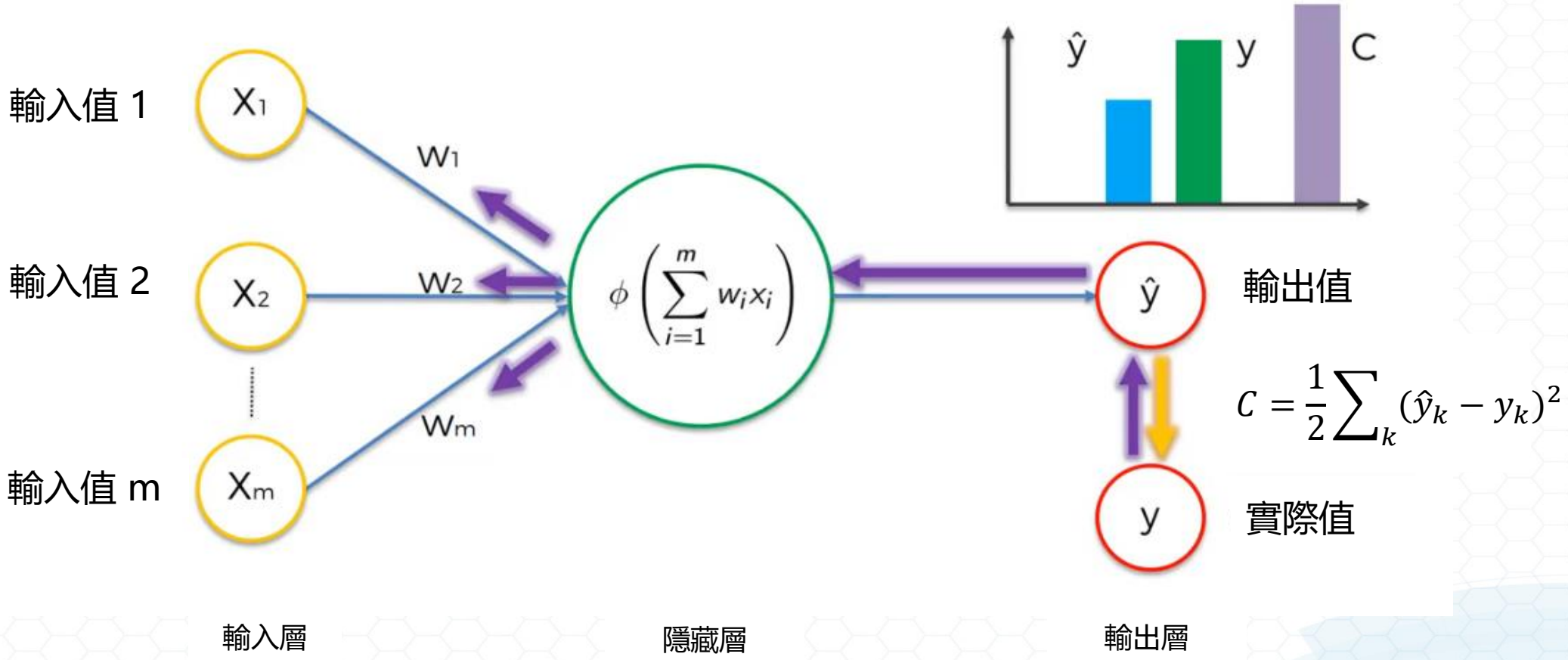
$$y'_k = \frac{e^{y_k}}{\sum_{i=1}^{n} e^{y_i}}$$



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

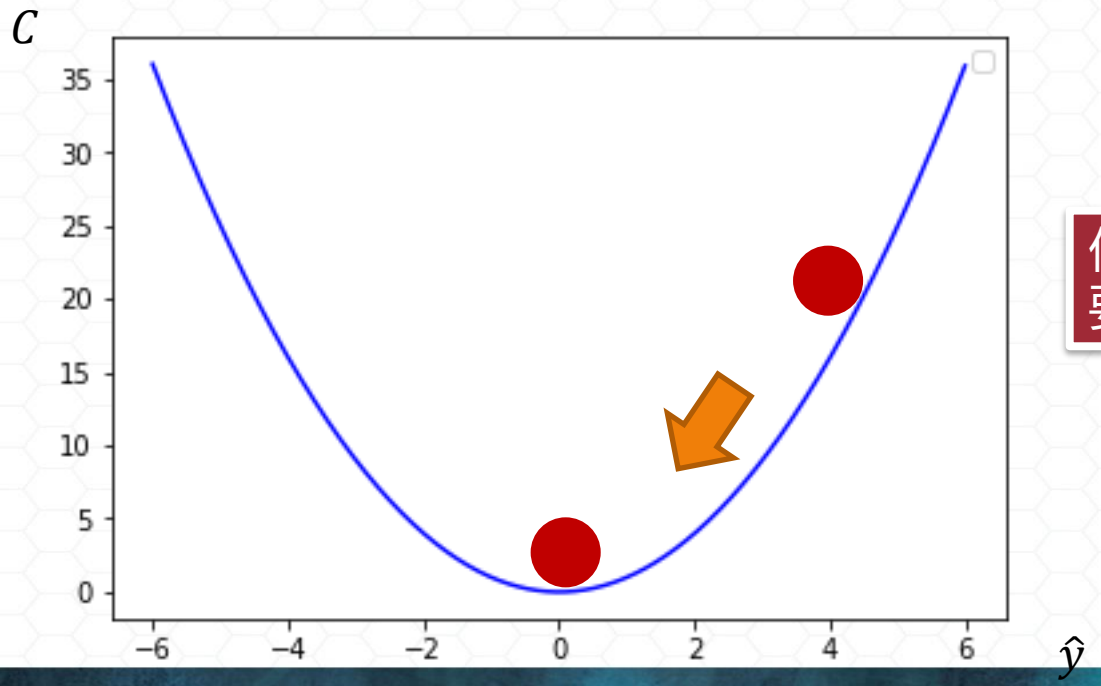
$$C = -\sum_k y_k \log \hat{y}_k$$

- 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

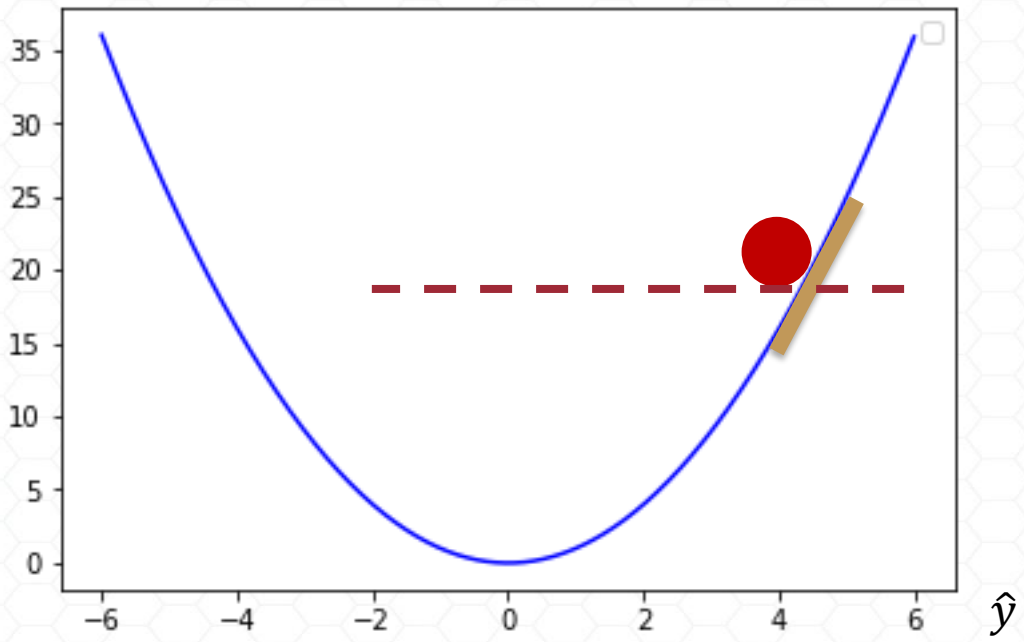


但參數這麼多
要如何有效搜尋代價函數的最小值

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？

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



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

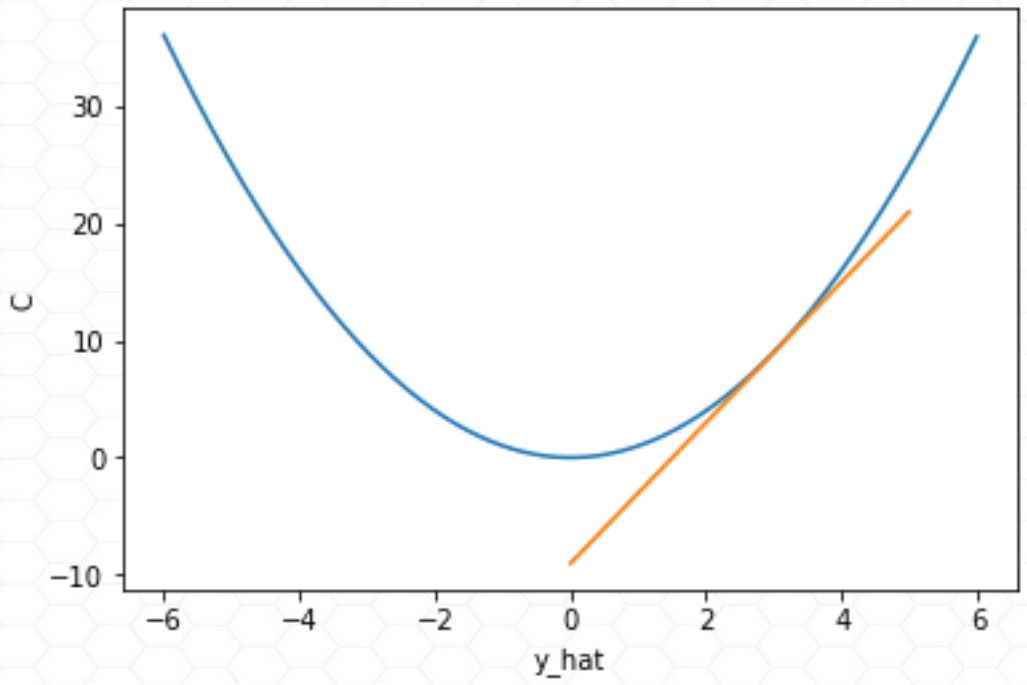
```
def func(x):
    return x ** 2

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

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

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

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

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

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

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

因為h 不可能逼近無限小

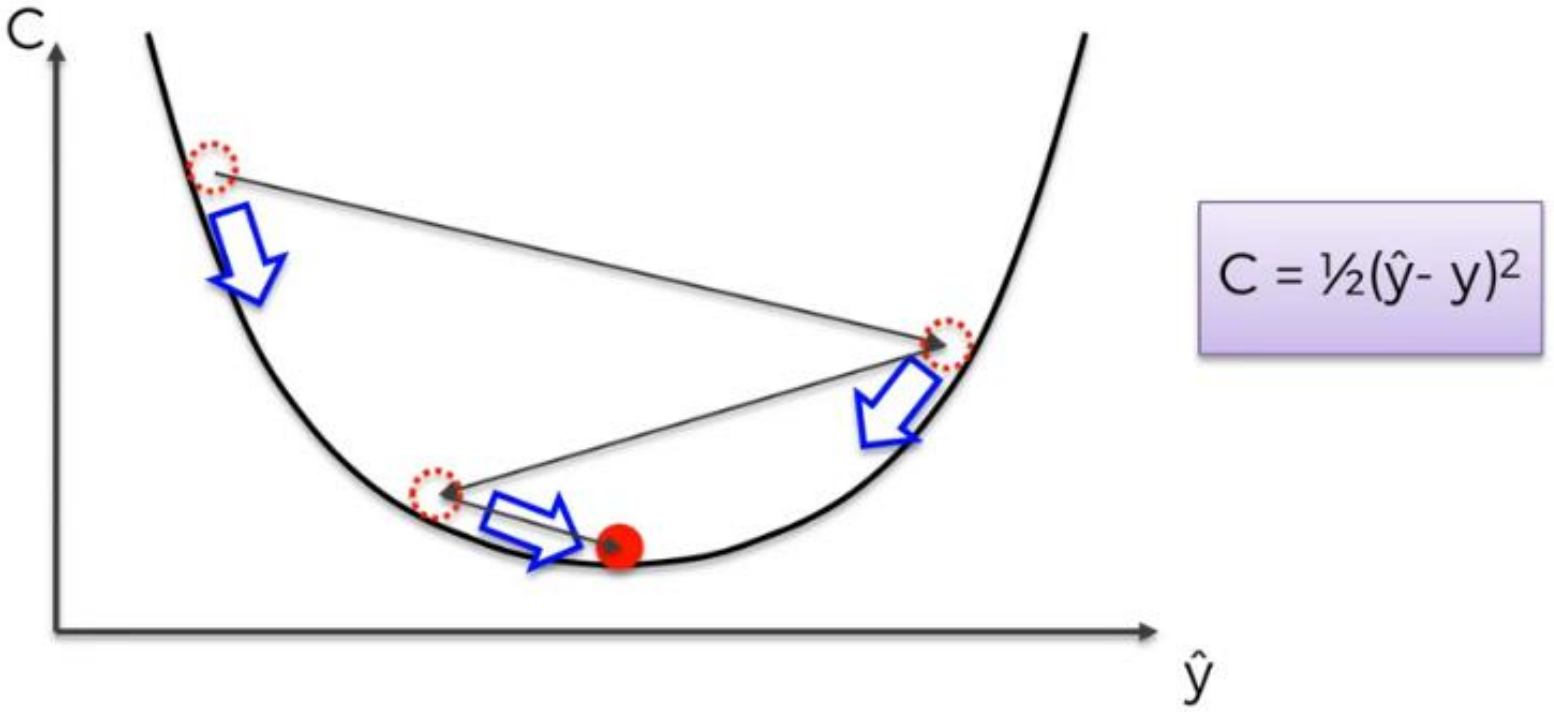
```
def dfunc(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)
```

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,要變更多少參數, 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

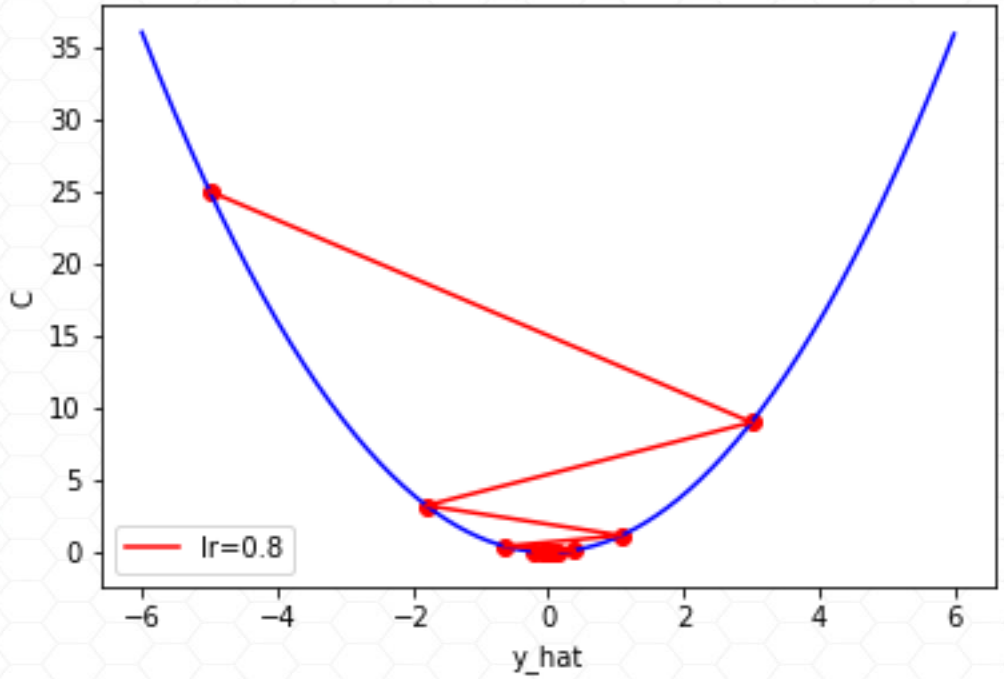
梯度下降

```
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)
```

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

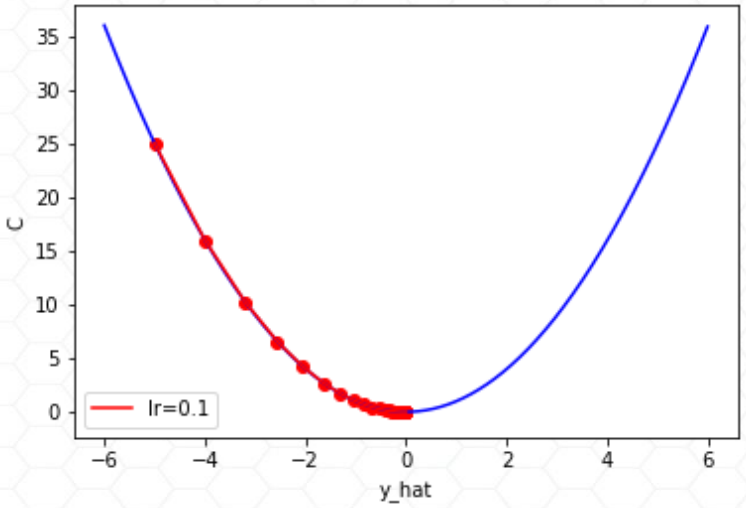
梯度下降

```
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')
```

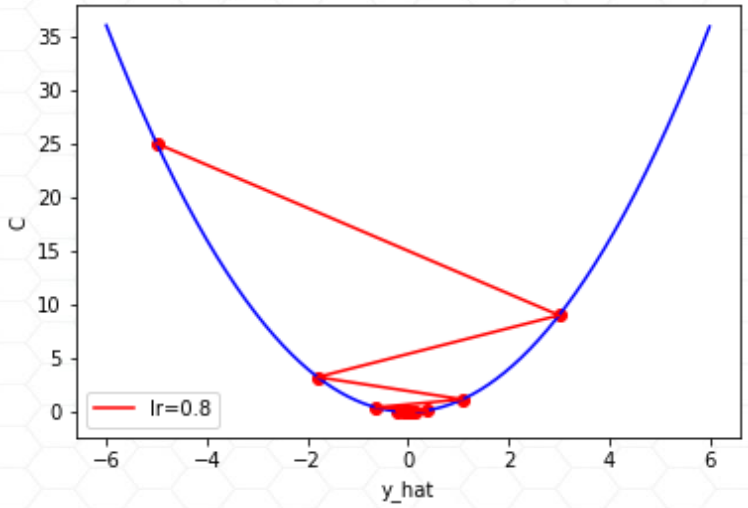


比較不同學習率

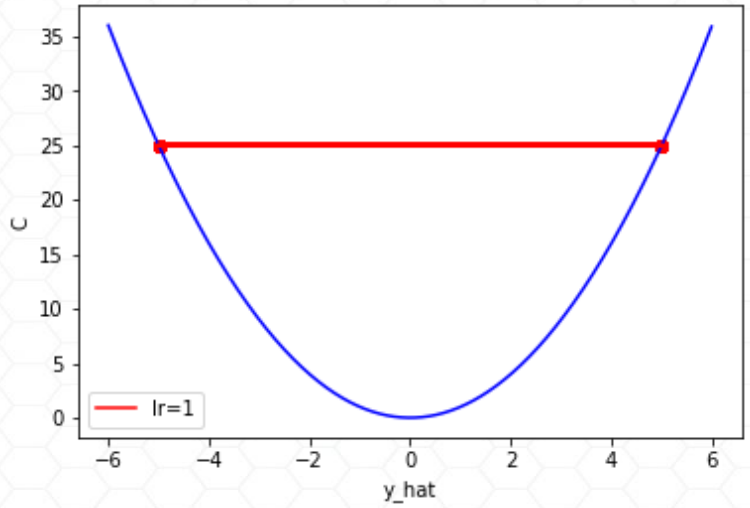
$\varepsilon = 0.1$



$\varepsilon = 0.8$

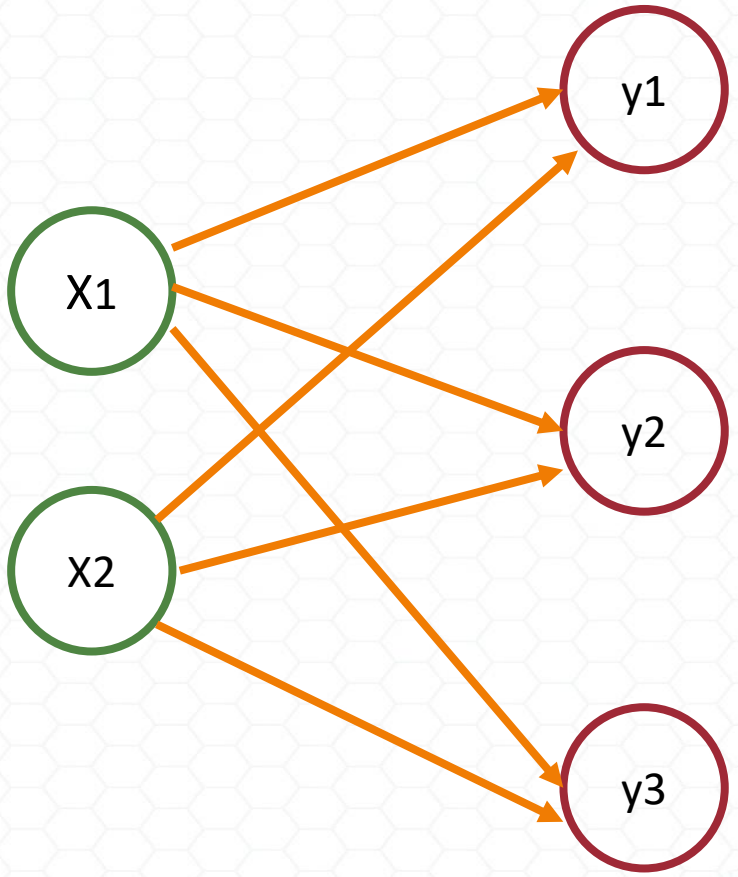


$\varepsilon = 1$



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

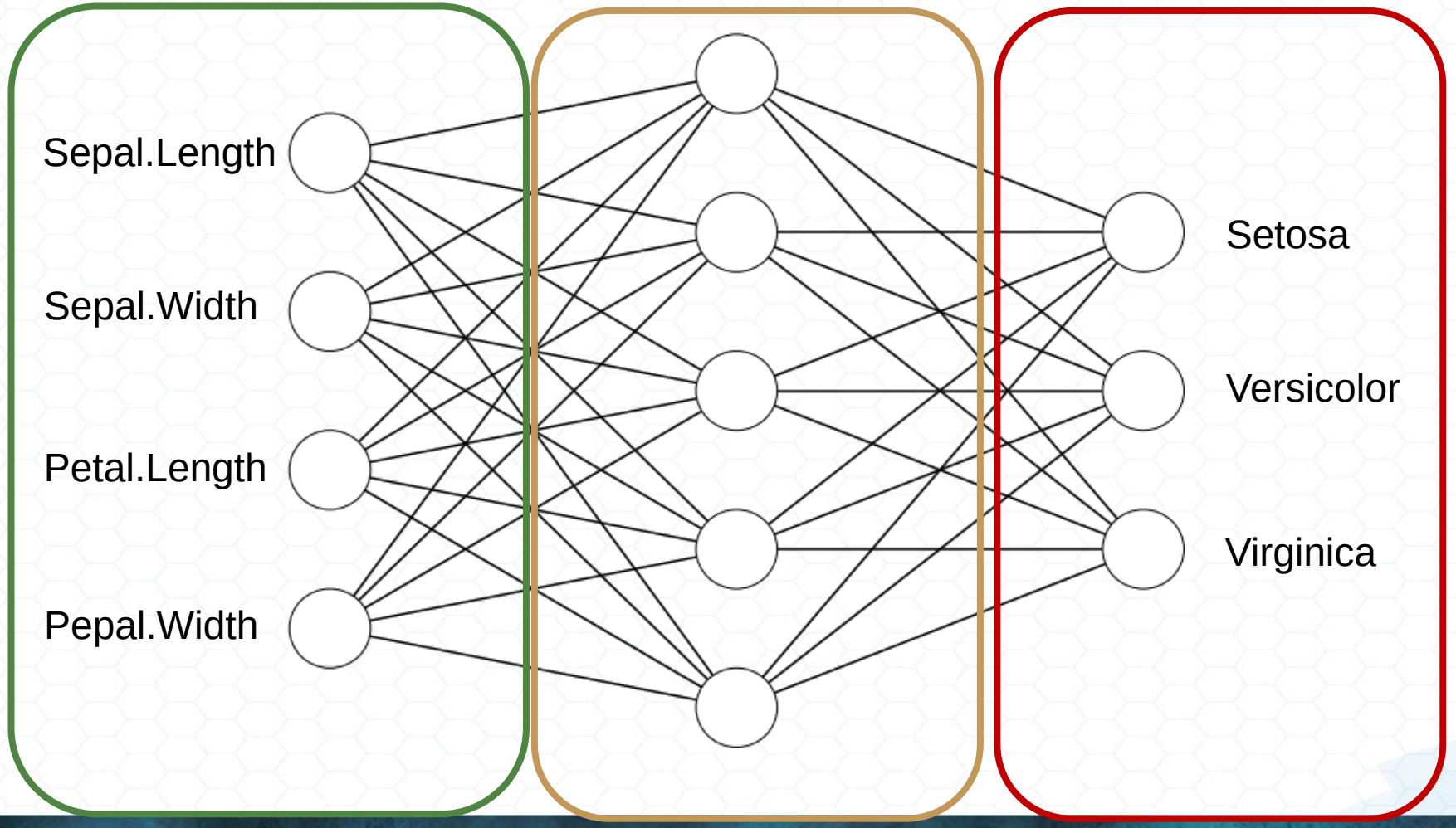
```
def dfunc(f, x):
    h = 1e-4
    grad = np.zeros_like(x)
    it = np.nditer(x, flags=['multi_index'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)
        x[idx] = tmp_val - h
        fxh2 = f(x)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



輸入層　　隱藏層　　輸出層

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

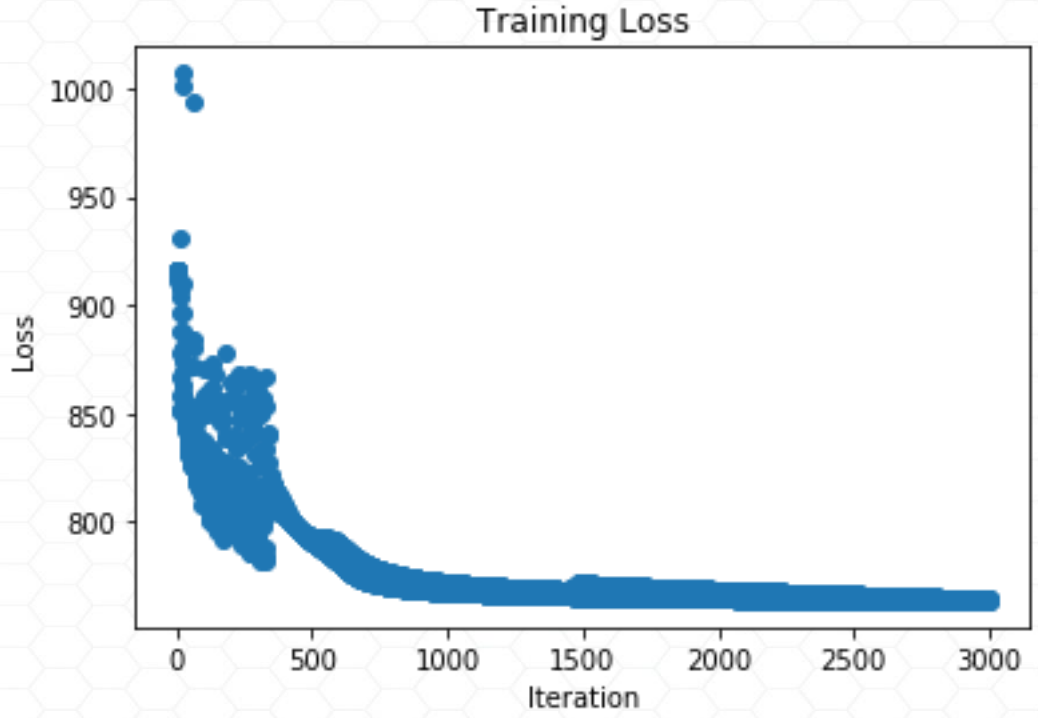
```
y = np.zeros((len(iris.target), 3))
for idx, val in enumerate(iris.target):
    y[idx, val] = 1
```

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:

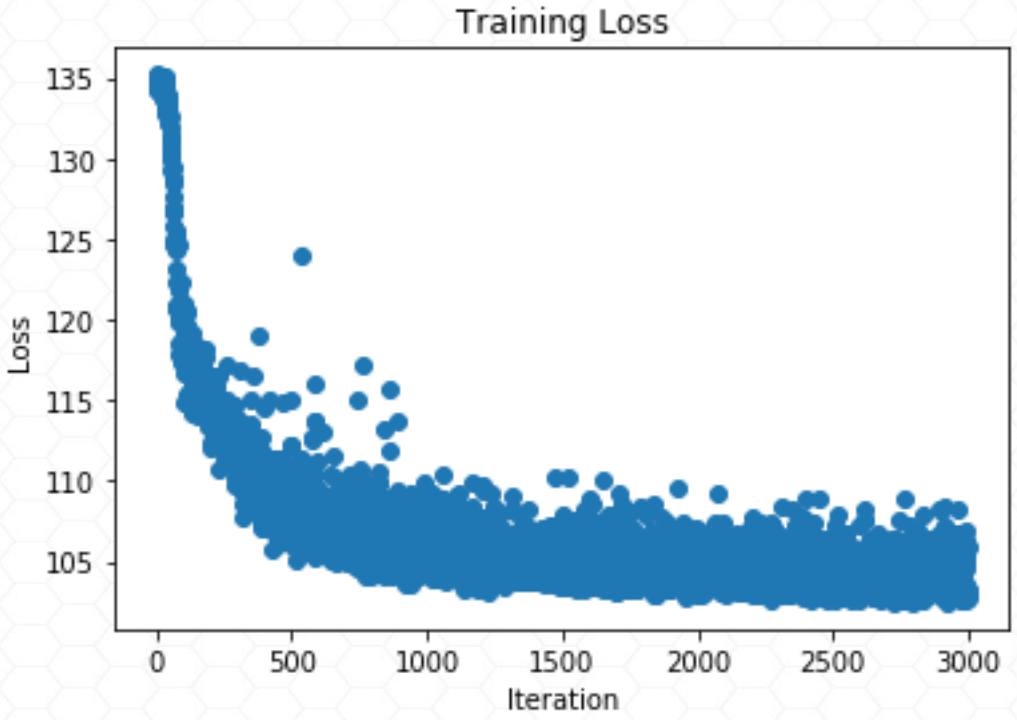
1. 隨機挑選批次資料
2. 計算各權重參數的梯度
3. 更新權重參數

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')
```

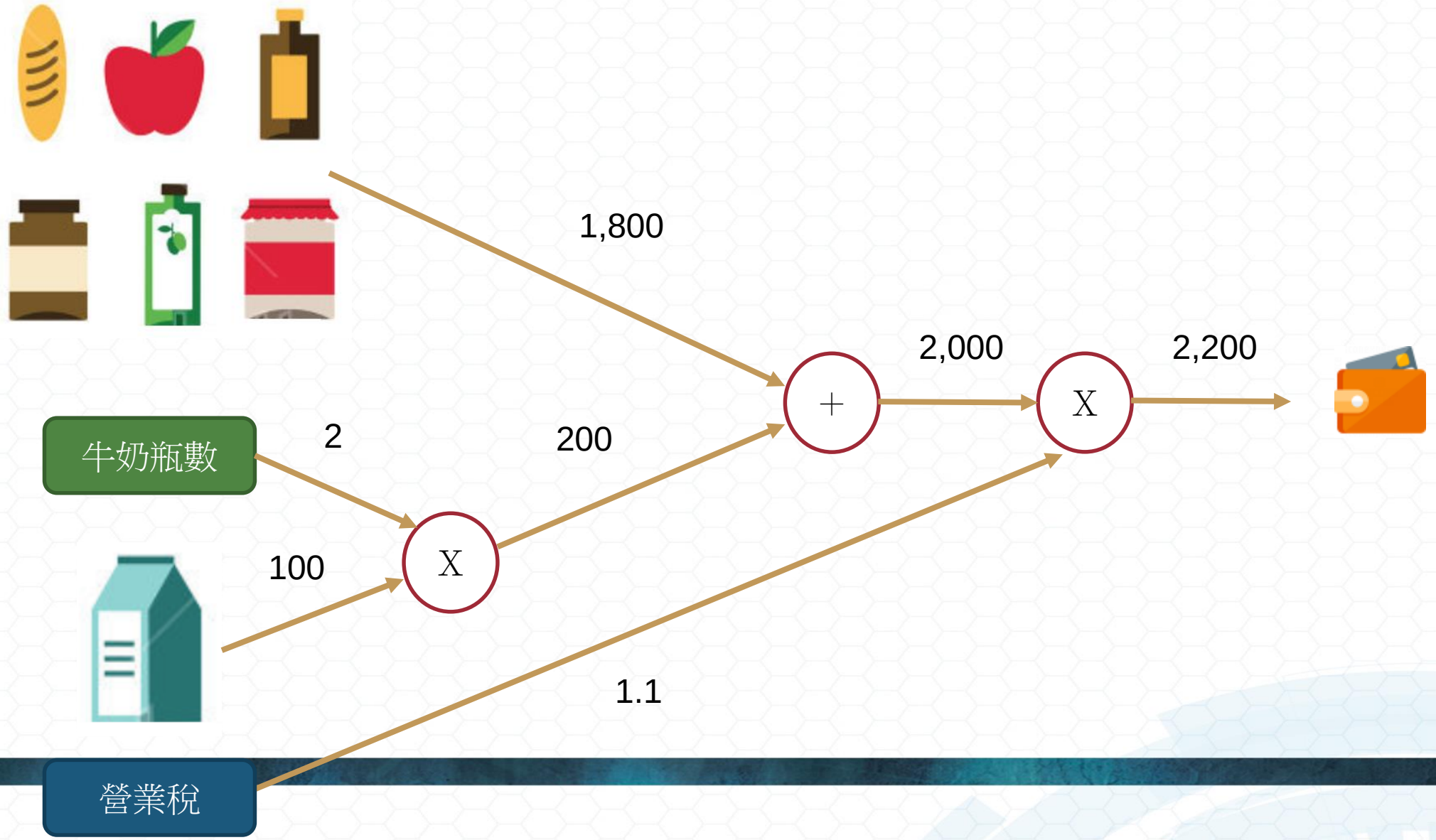


驗證批次學習模型準確度

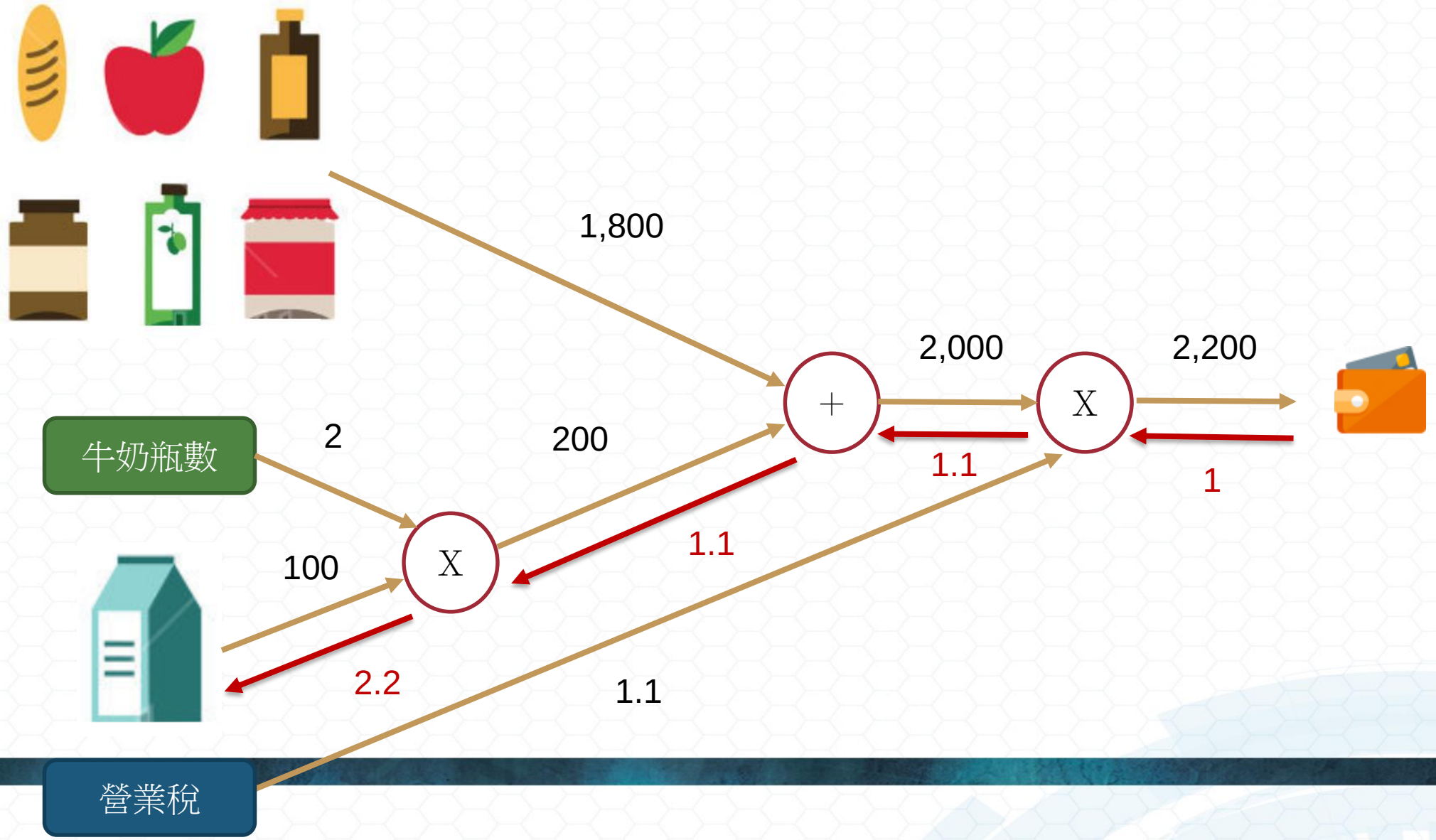
```
from sklearn.metrics import accuracy_score, confusion_matrix
predicted = np.argmax(net.predict(x), axis=1)

# 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)



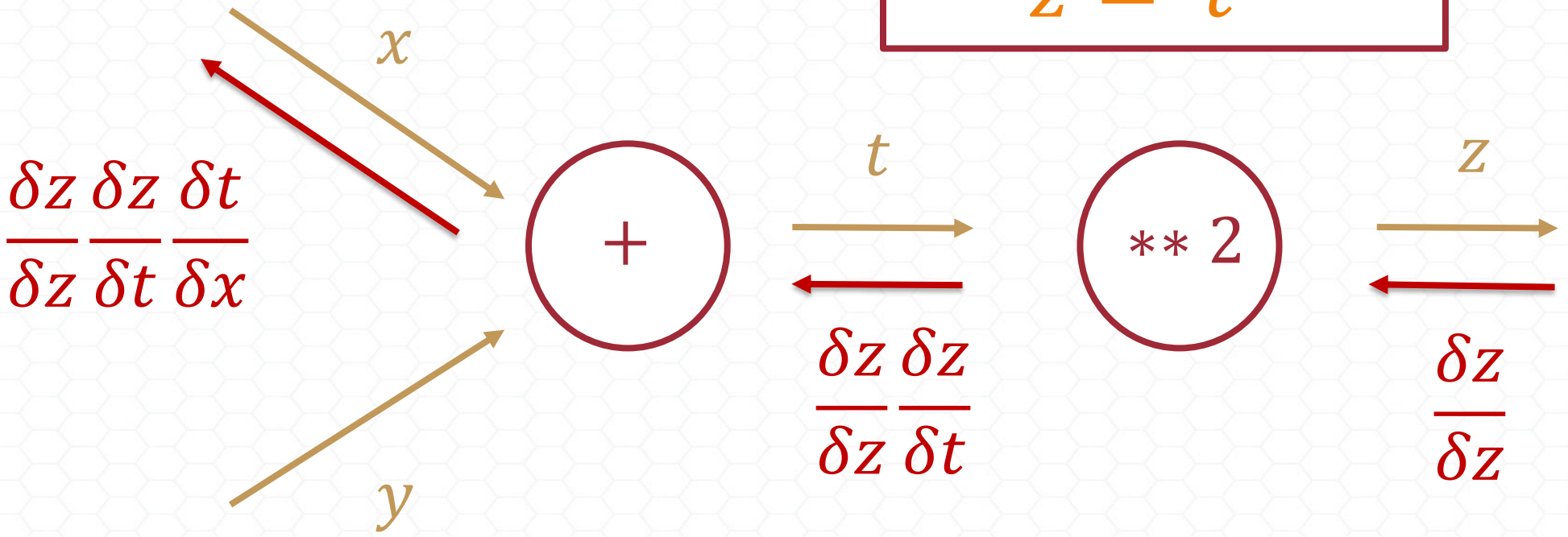
計算圖 (Computational Graph)



反向傳播

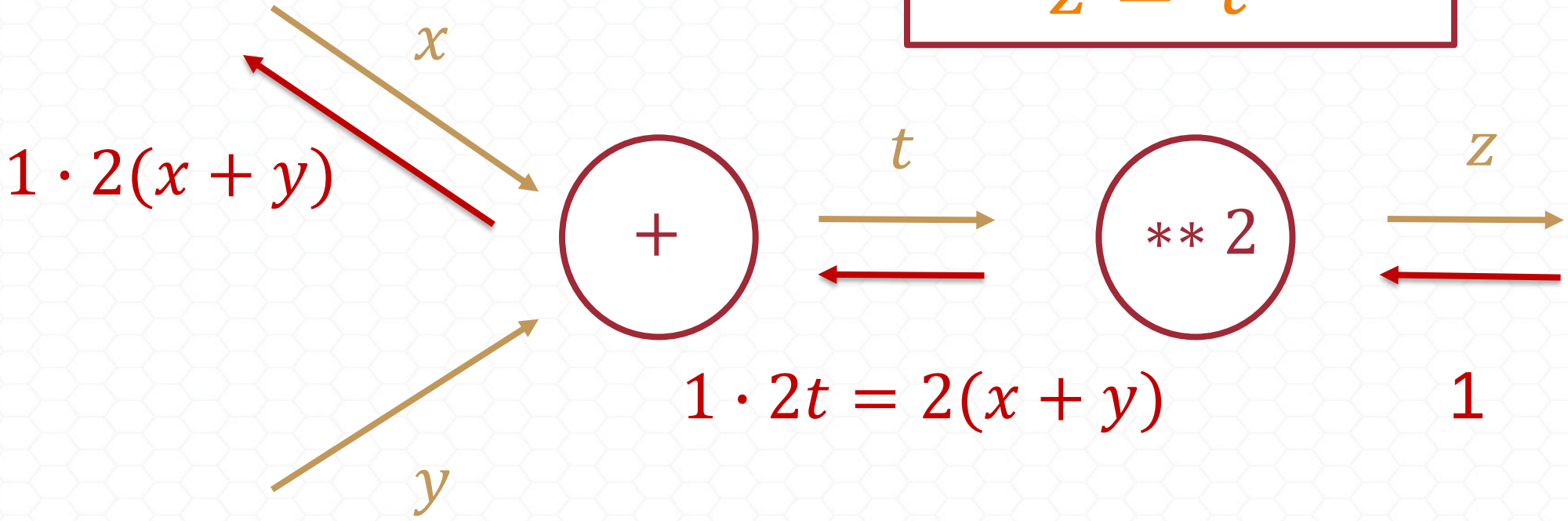
反向傳播

$$t = x + y$$
$$z = t^2$$

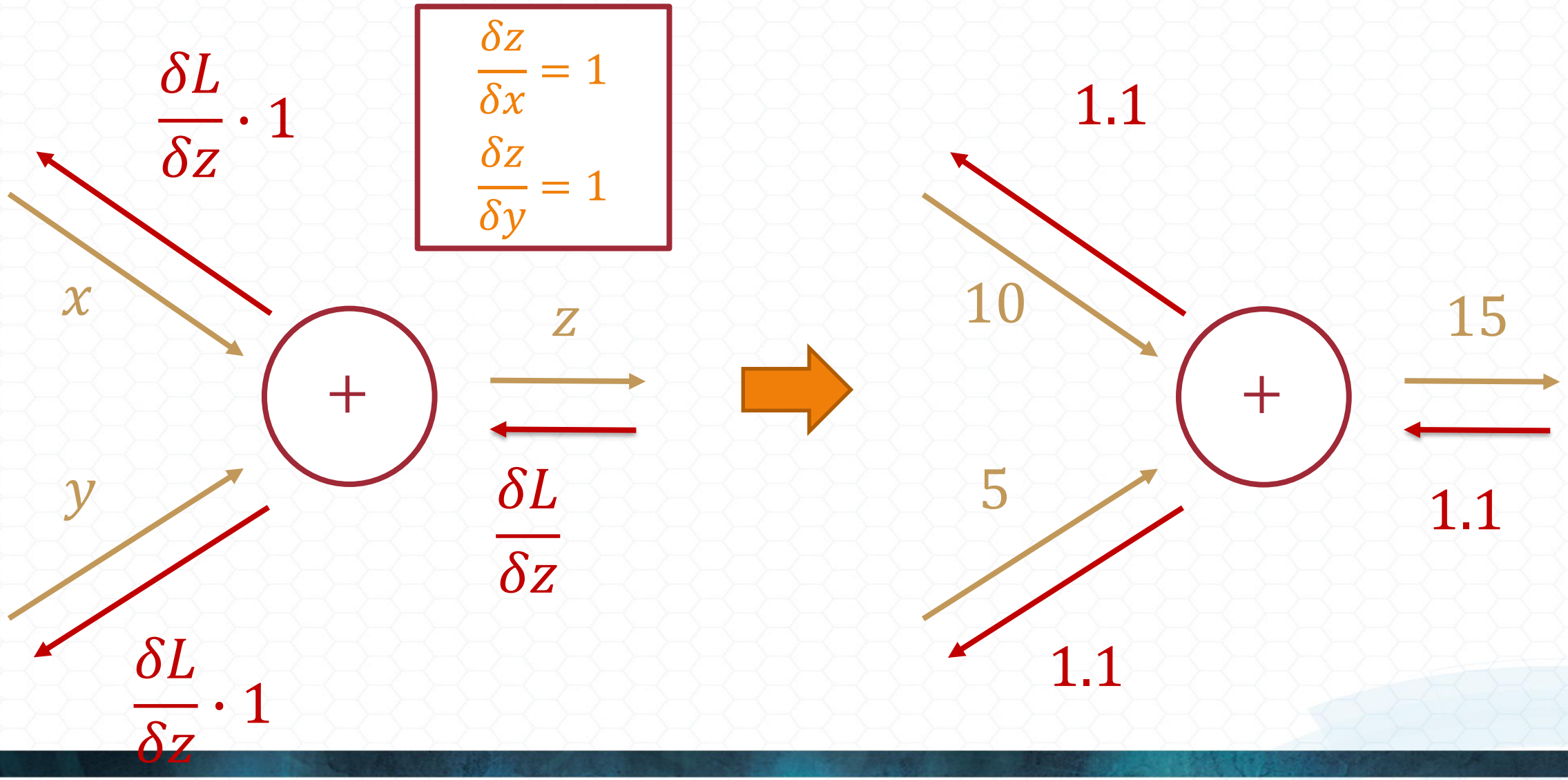


反向傳播

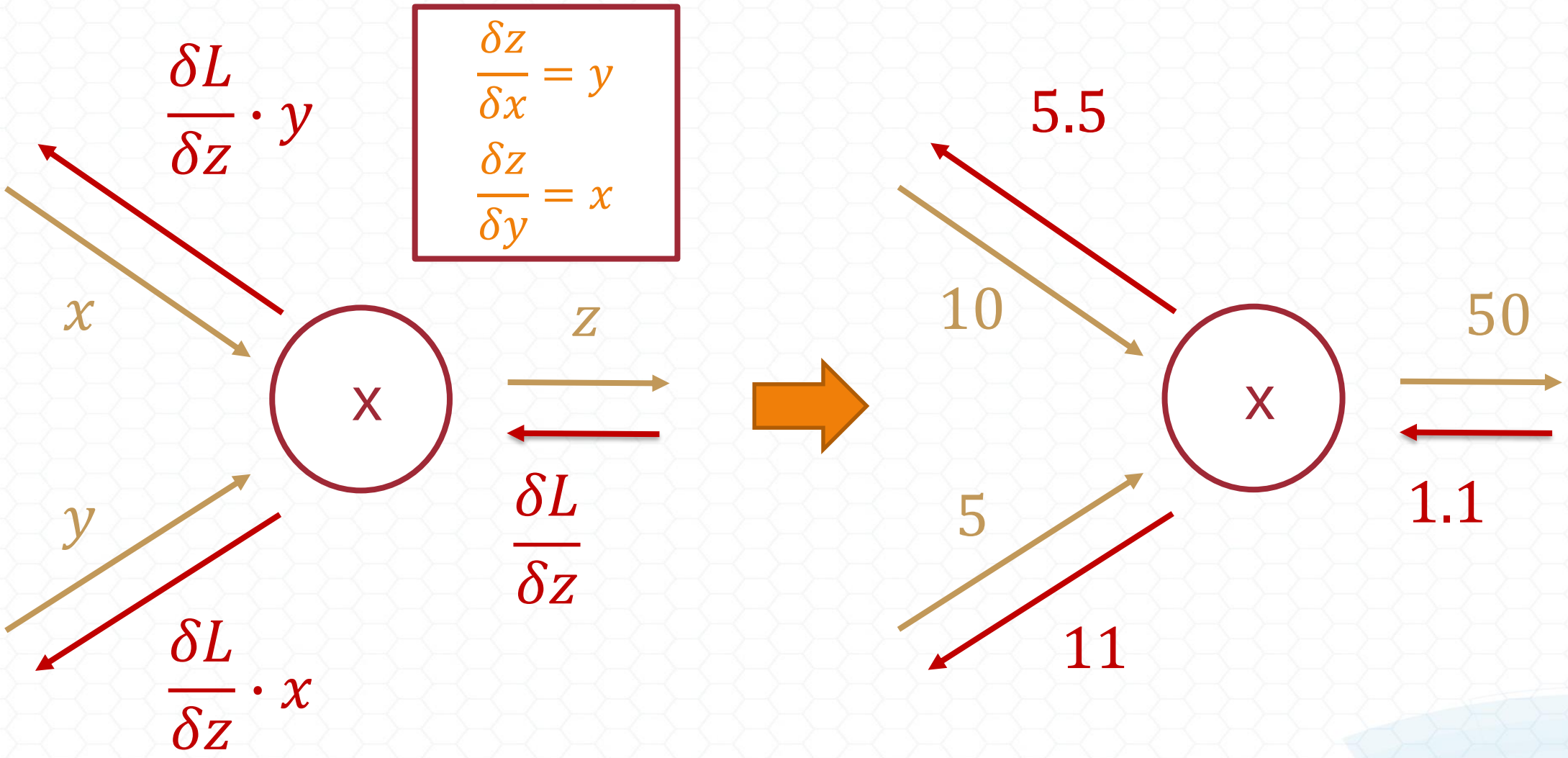
$$t = x + y$$
$$z = t^2$$



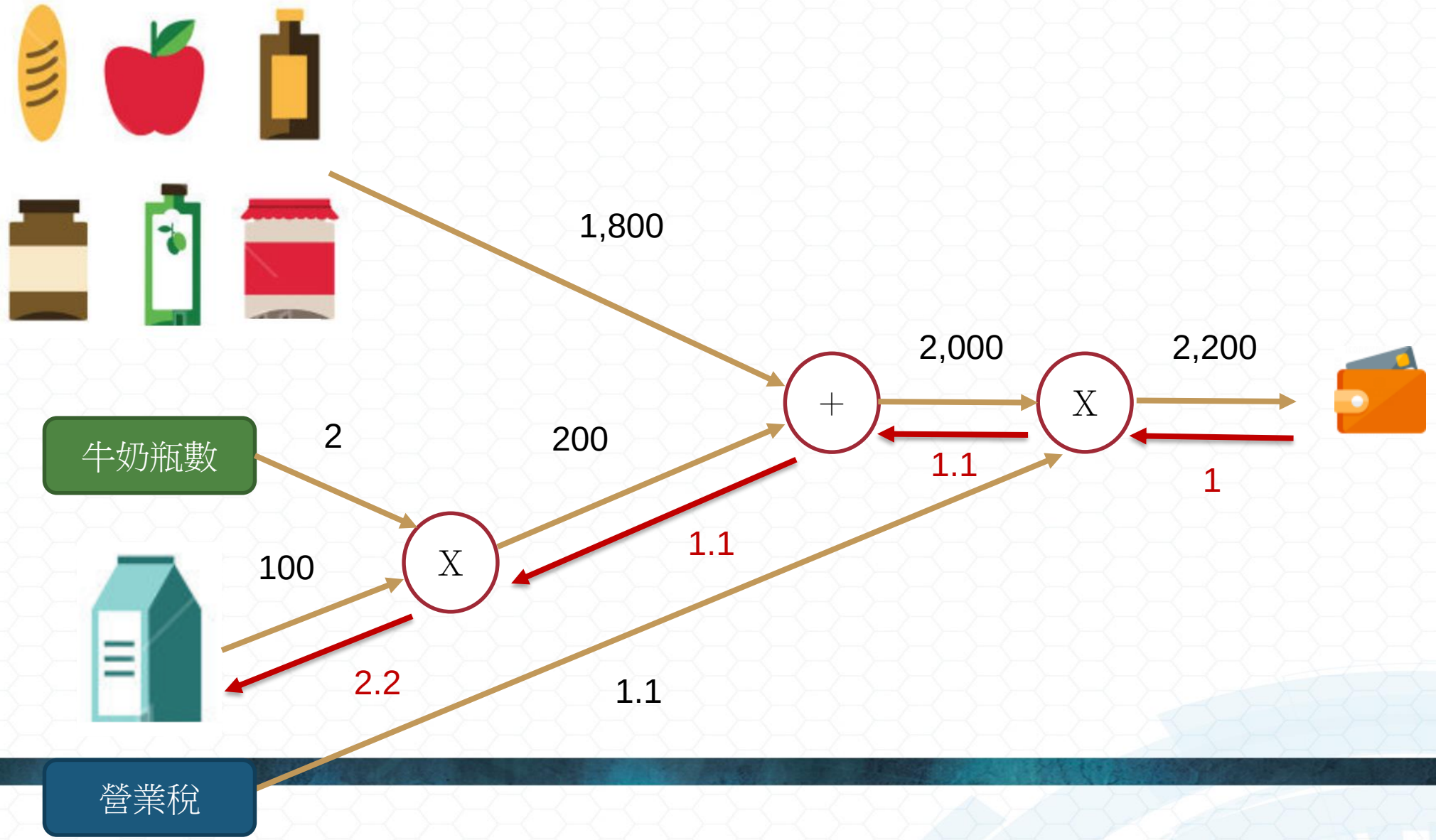
加法反向傳播


$\frac{\delta z}{\delta x} = 1$
$\frac{\delta z}{\delta y} = 1$
$\frac{\delta L}{\delta z} \cdot 1$
x
y
z
$\frac{\delta L}{\delta z}$
$\frac{\delta L}{\delta z} \cdot 1$
1.1
10
5
15
1.1
1.1

乘法反向傳播

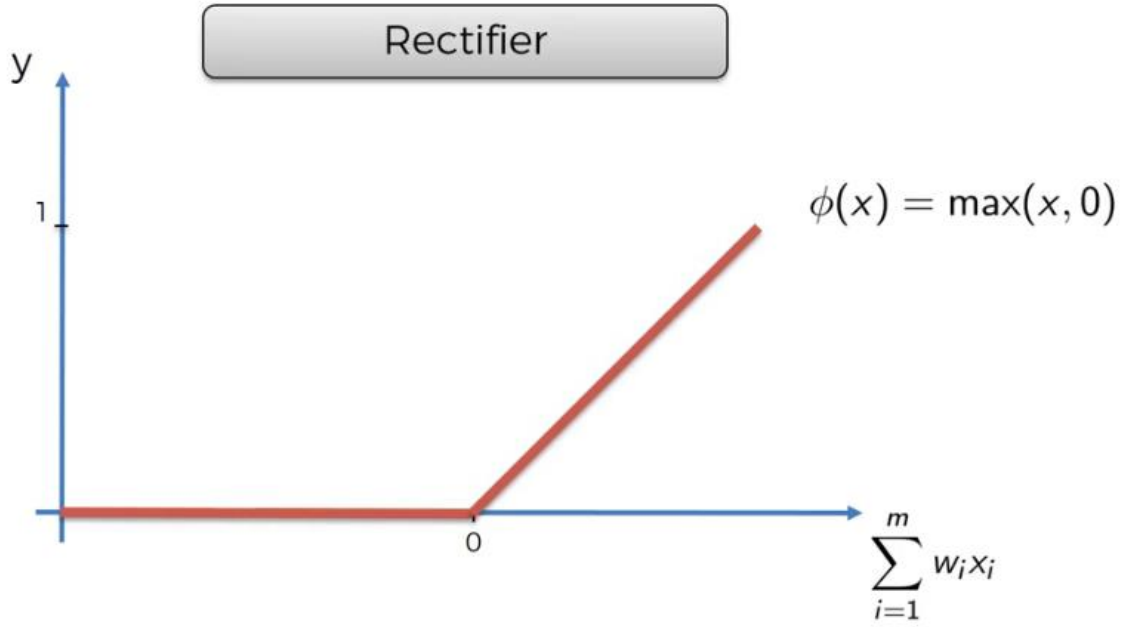

$\frac{\delta z}{\delta x} = y$
$\frac{\delta z}{\delta y} = x$
$\frac{\delta L}{\delta z} \cdot y$
x
x
z
$\frac{\delta L}{\delta z}$
y
$\frac{\delta L}{\delta z} \cdot x$
5.5
10
x
50
1.1
5
11

計算圖 (Computational Graph)

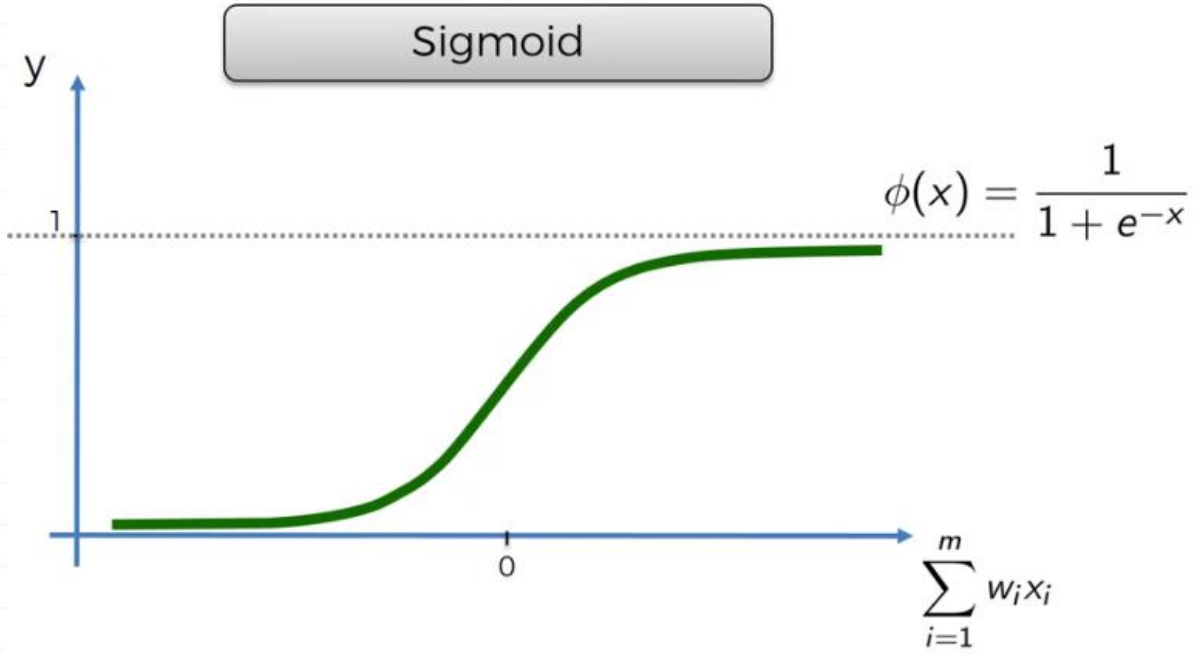


將計算圖套用在啟動函數上

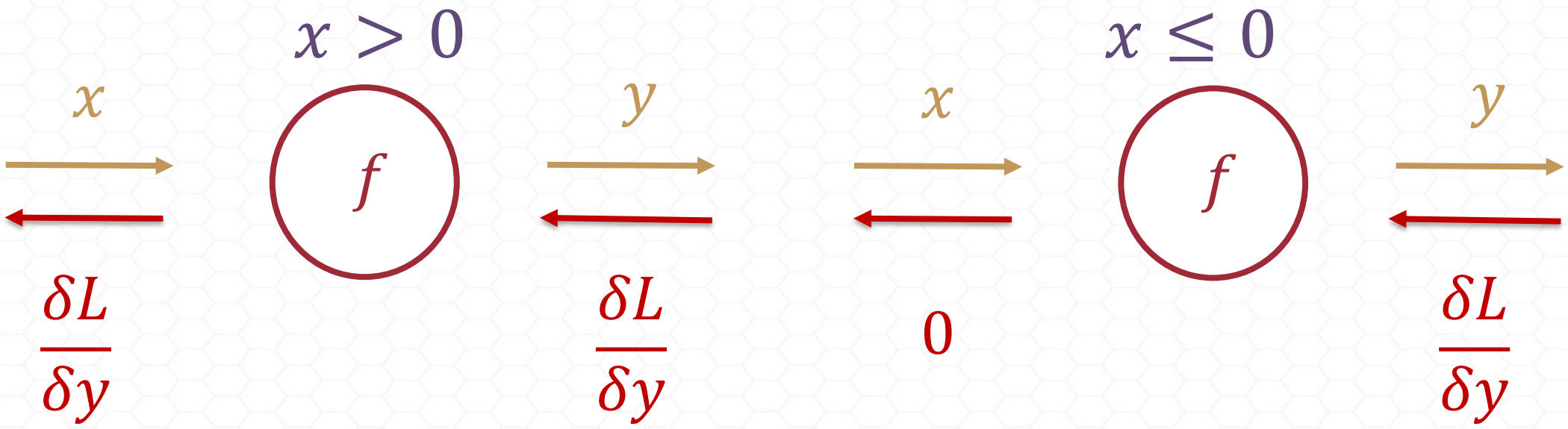
ReLU 層



Sigmoid 層



ReLU 反向傳播

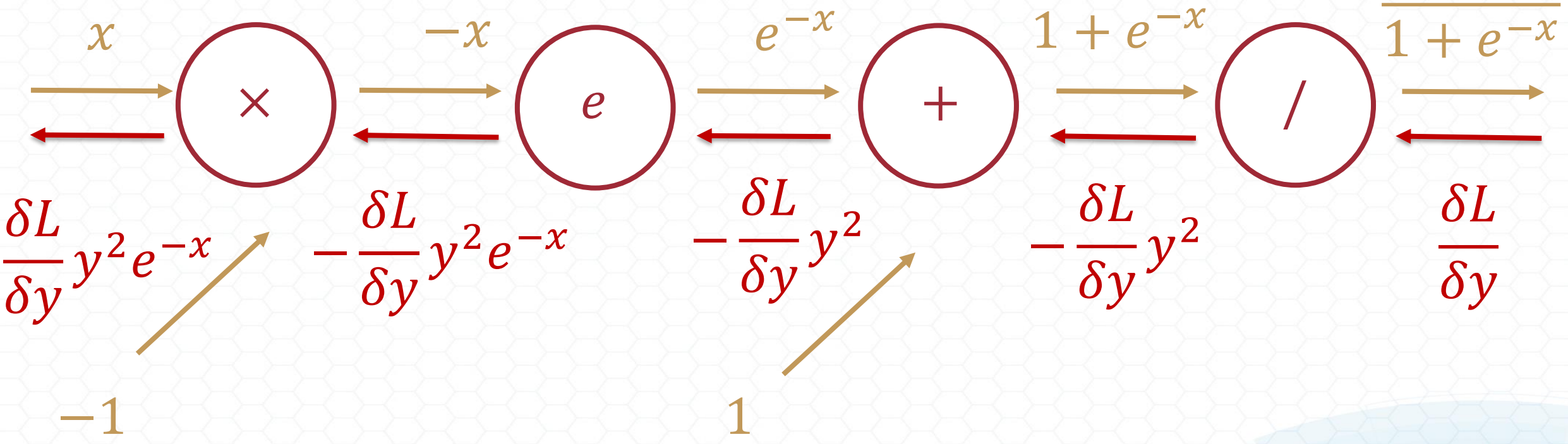

$x > 0$
x
f
y
$\frac{\delta L}{\delta y}$
$\frac{\delta L}{\delta y}$
$x \leq 0$
x
f
y
0
$\frac{\delta L}{\delta y}$

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）

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

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

$$= \frac{\delta L}{\delta y} y(1-y)$$

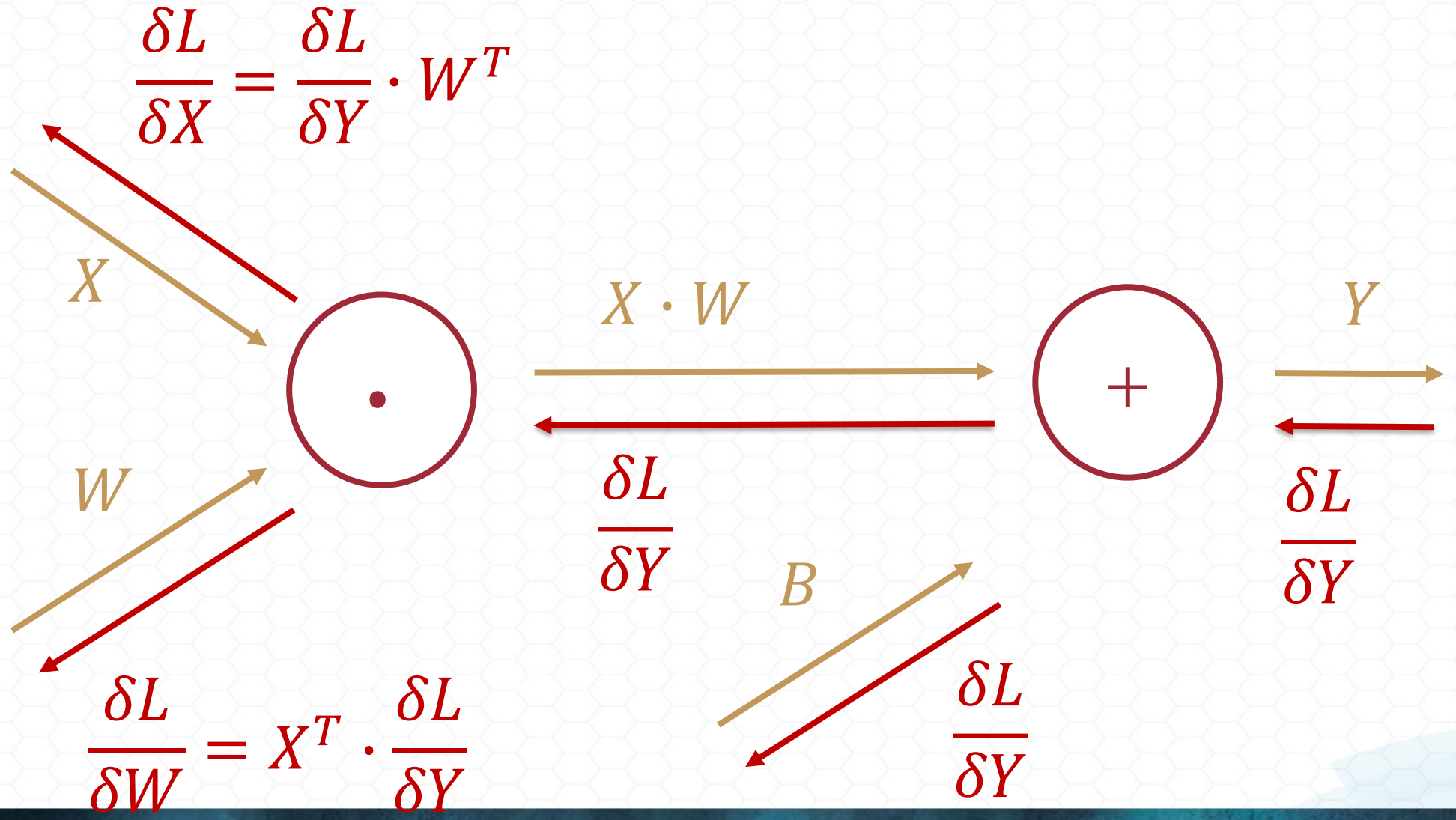
$$\| $$

$$\frac{\delta L}{\delta y} y(1-y)$$

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
```

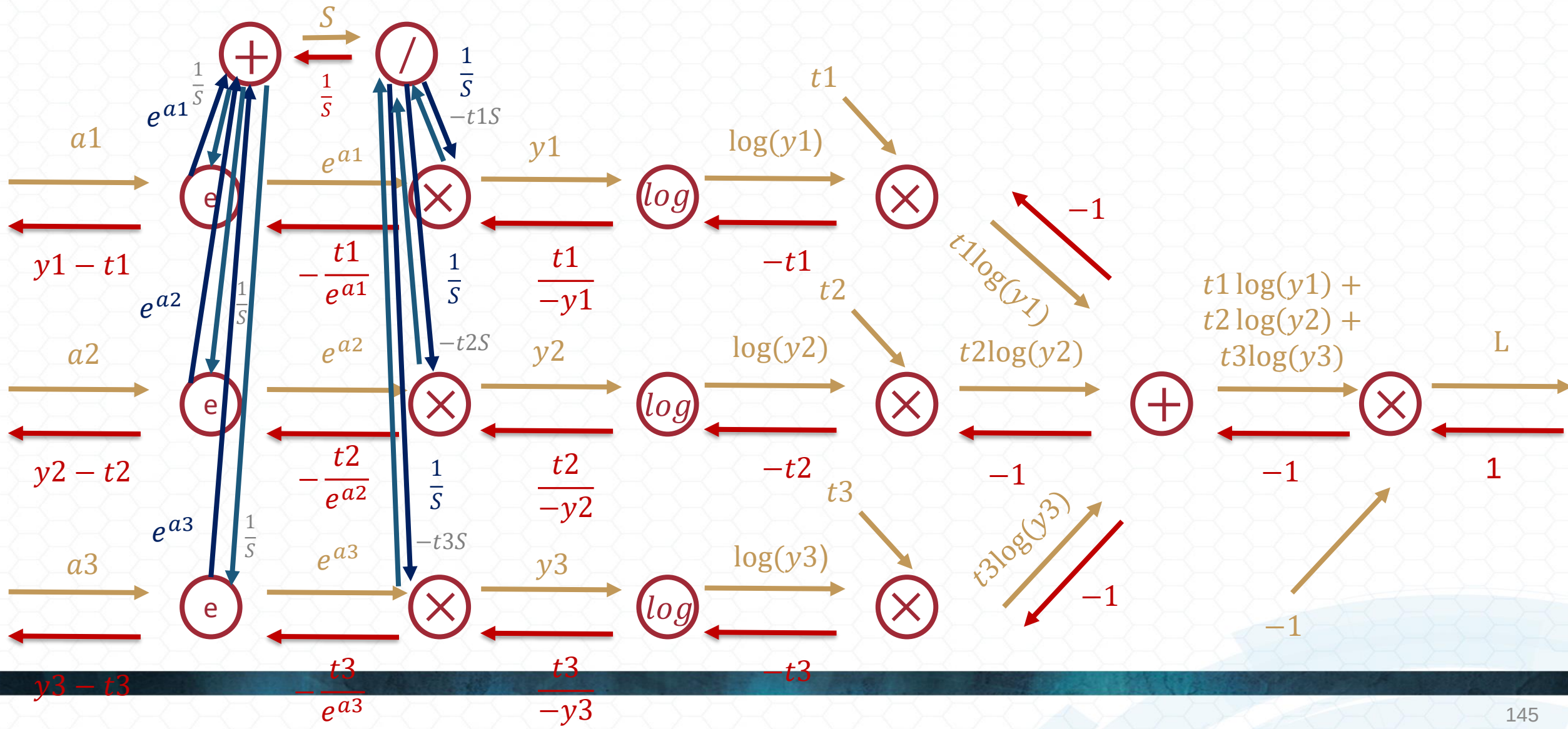
仿射映射 (Affine Transformation)



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

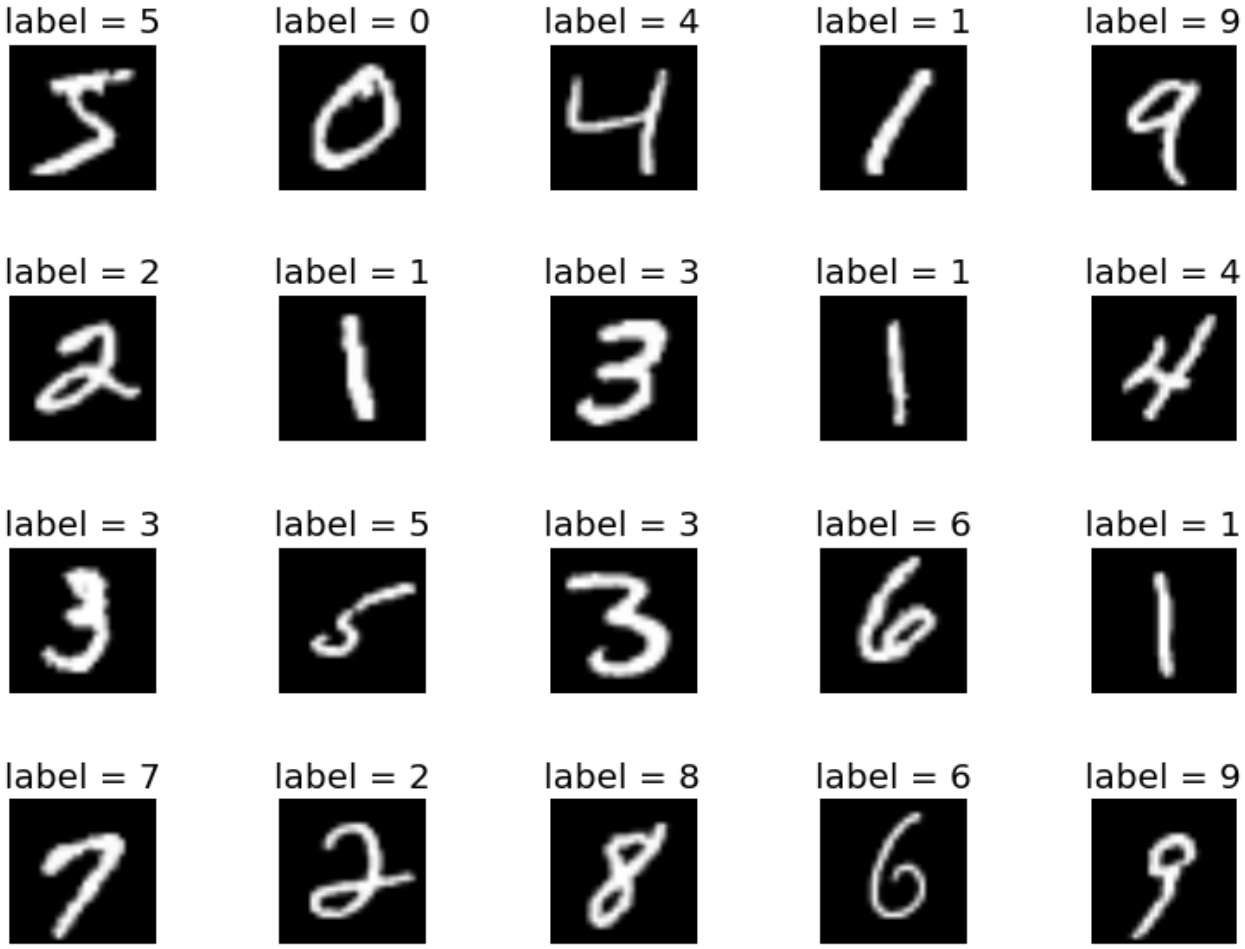
```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y_hat = None
        self.y = None

    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

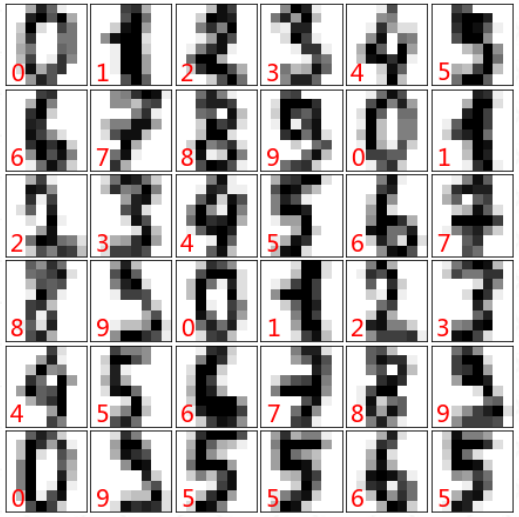


讀取數位資料(MNIST)

```
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()
```

繪製數位資料

```
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)
```



變更數位資料的尺度

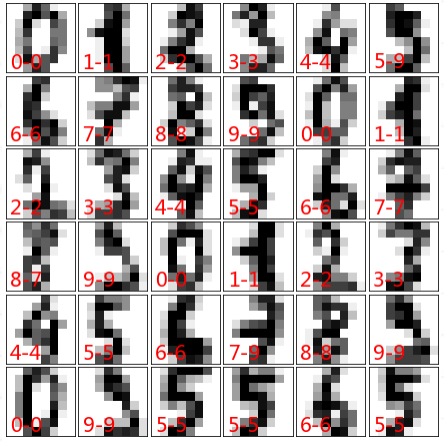
```
scaler = StandardScaler()
scaler.fit(digits.data)
X_scaled = scaler.transform(digits.data)
```

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)
```



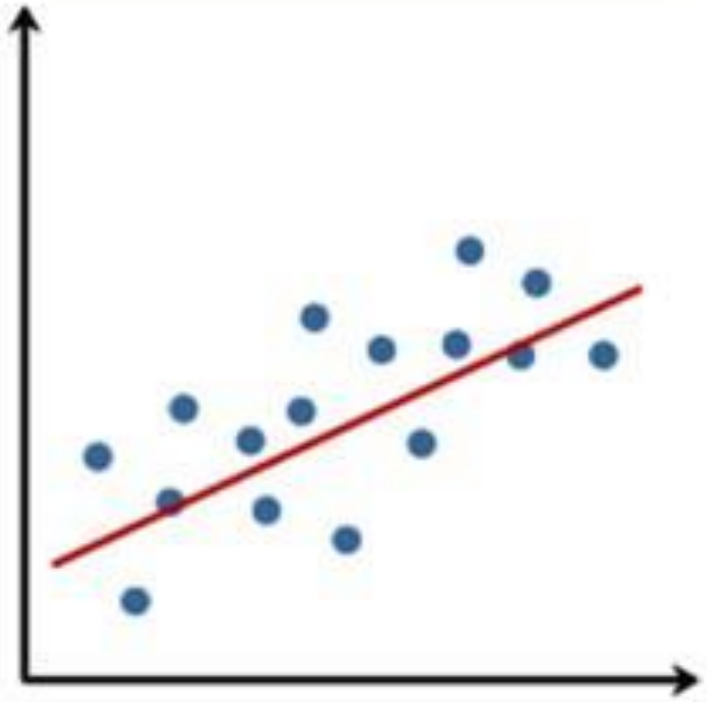
驗證模型準確度

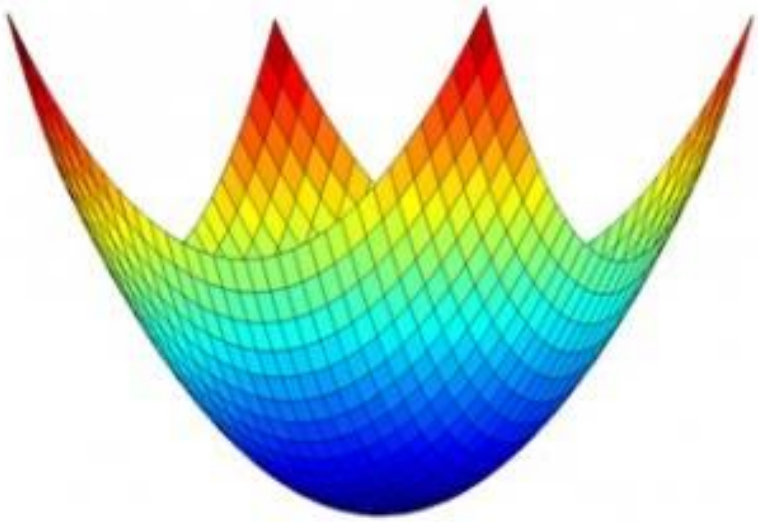
```
from sklearn.metrics import accuracy_score, confusion_matrix

# accuracy
accuracy_score(digits.target, predicted)

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

梯度消失

線性問題求解

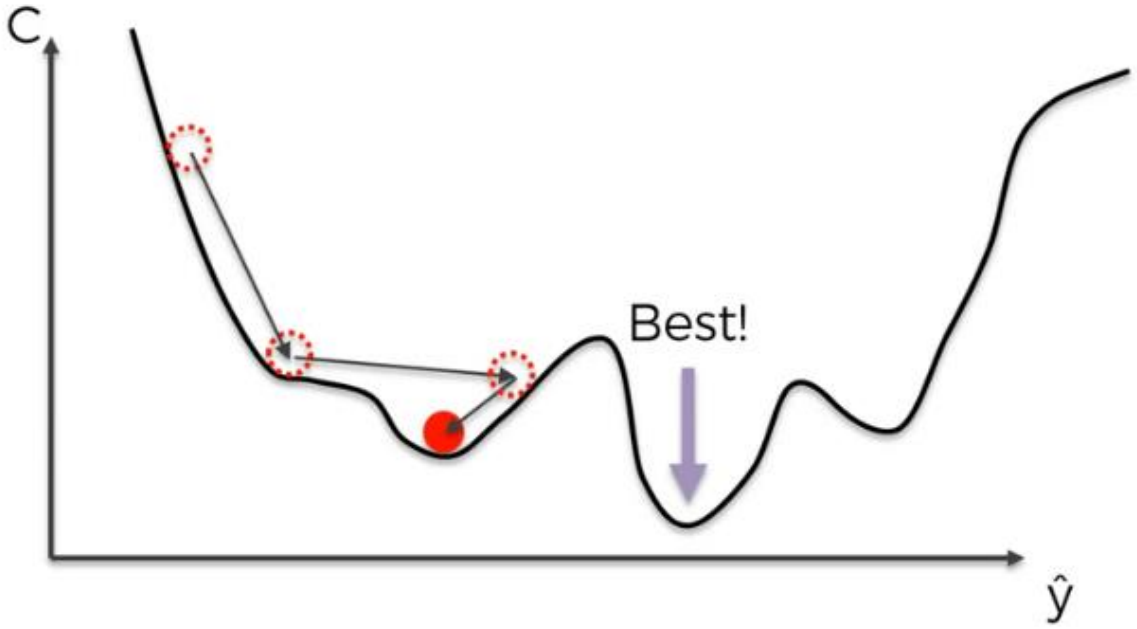
$$h_\theta(x) = \theta_0 + \theta_1 x$$





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

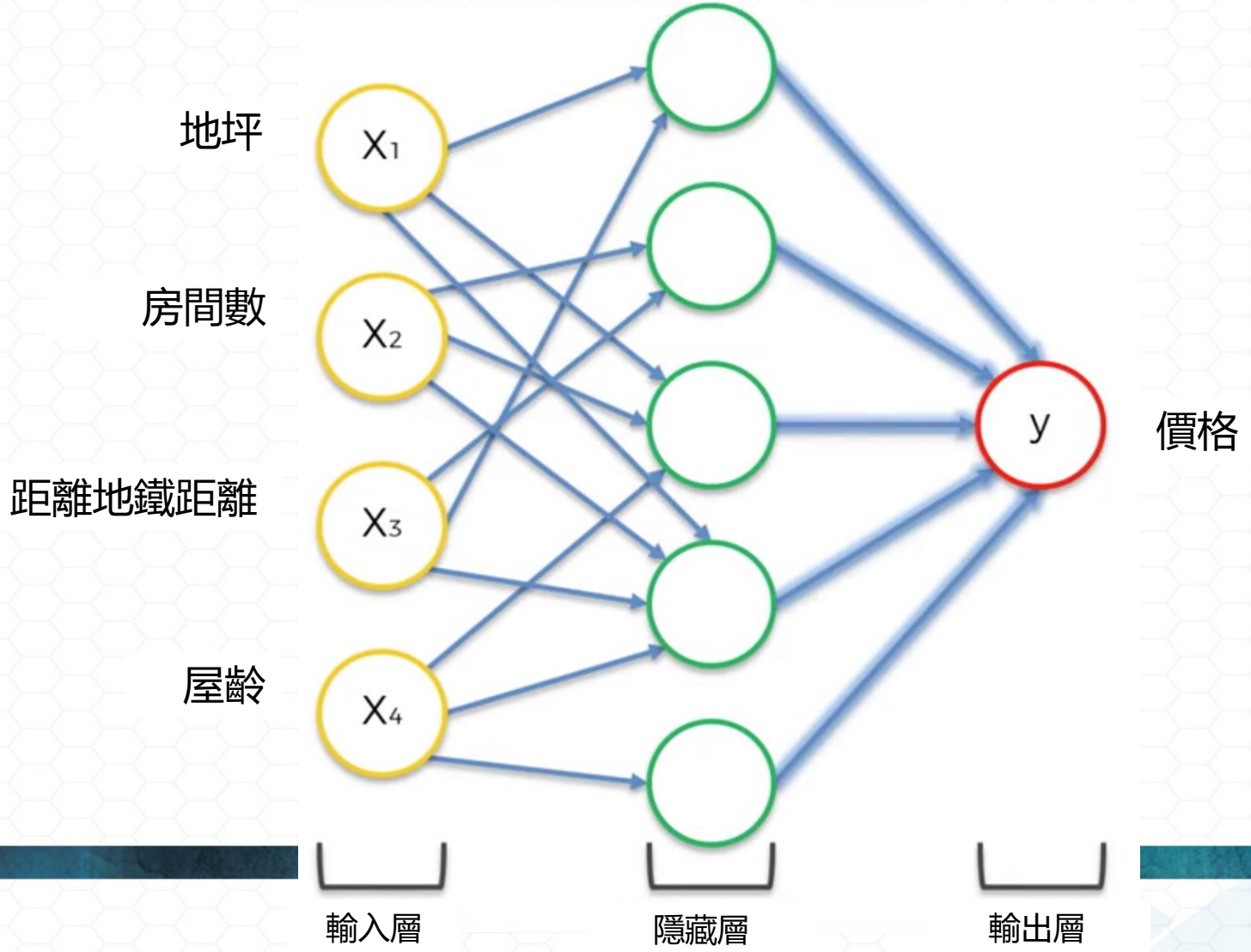


「梯度消失」(Vanishing Gradient)

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

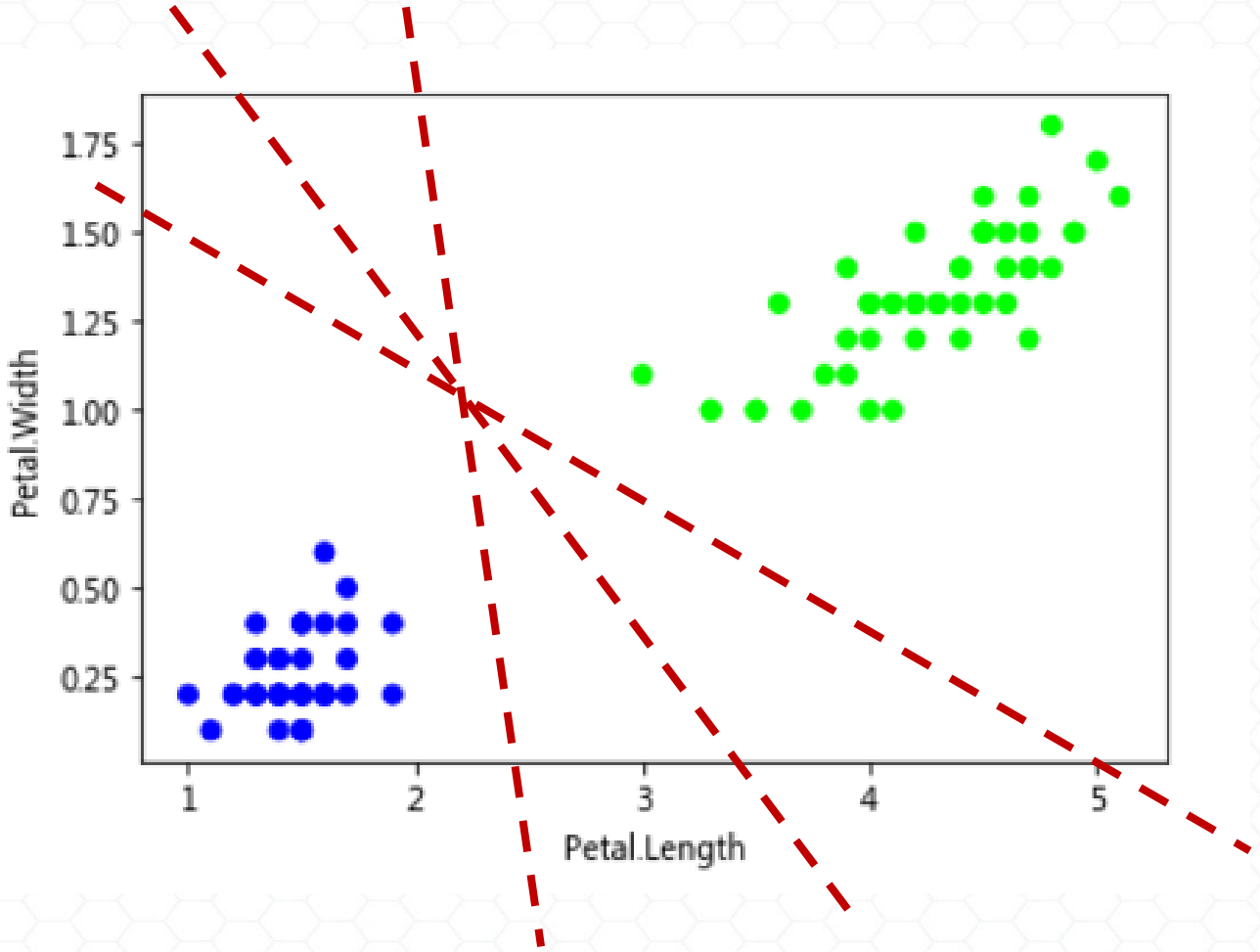


傳統神經網路只能處理淺層結構

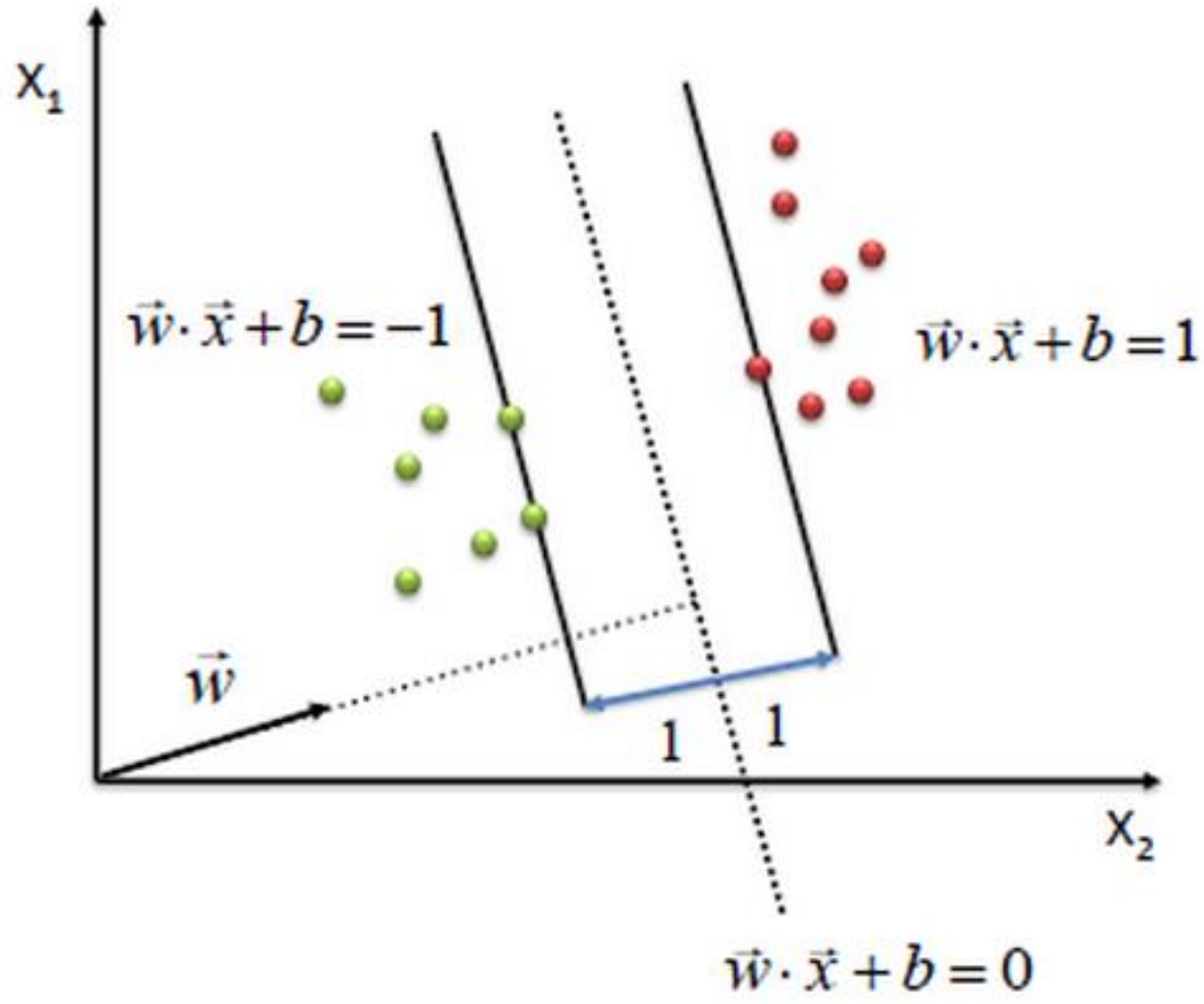
- 梯度消失問題會隨著神經網路層數的增加而更加嚴重，所以傳統類神經網路只能處理淺層結構（比如 2 層）的網路，從而限制了性能
- 單層感知機、和多層感知機失敗，導致1980的學界認為類神經網路是死胡同
- 在 1990 年代，支持向量機 (Support Vector Machine, SVM) 等「淺層機器學習模型」成為主流技術，此為機器學習的第二波浪潮

支持向量機

該選哪一條線做切割?



支持向量機 (Support Vector Machine)



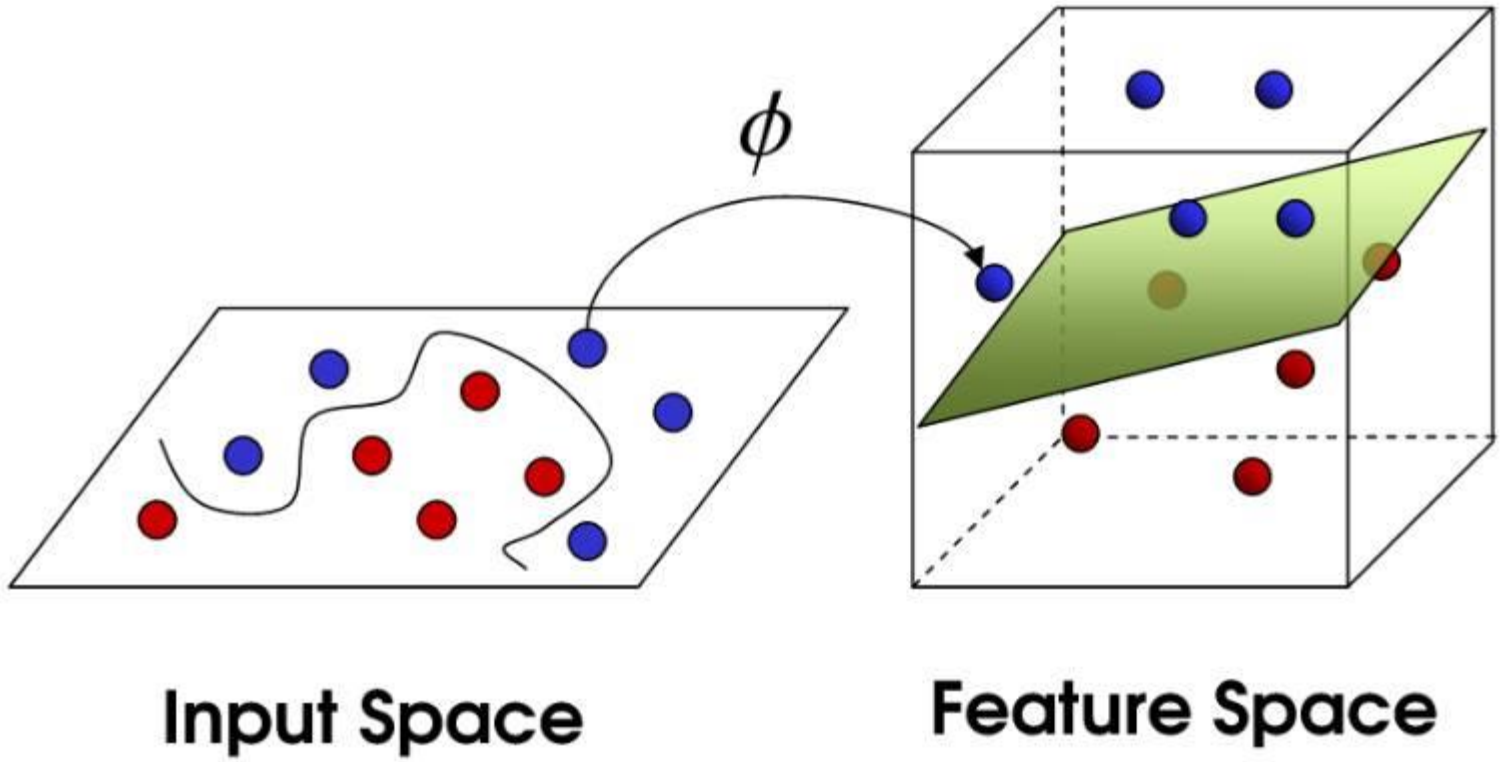
$$\max \frac{2}{\|w\|}$$

s.t.

$(w\cdot x+b)\geq 1, \forall x$ of class 1

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

解決高維度資料切分問題



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)
```

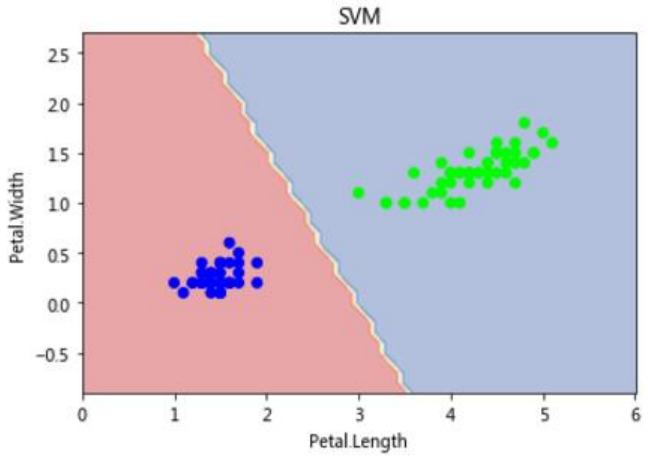
建立支持向量機 (2)

```
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()
```

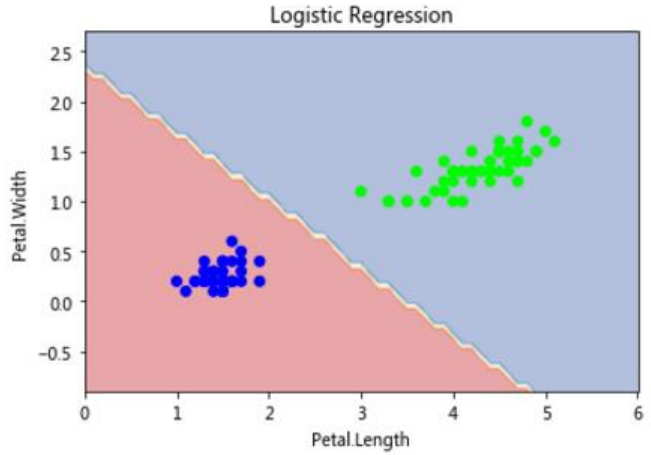
建立一個繪圖函數

SVM v.s. Logistic Regression

plot_estimator(clf1, X, y)



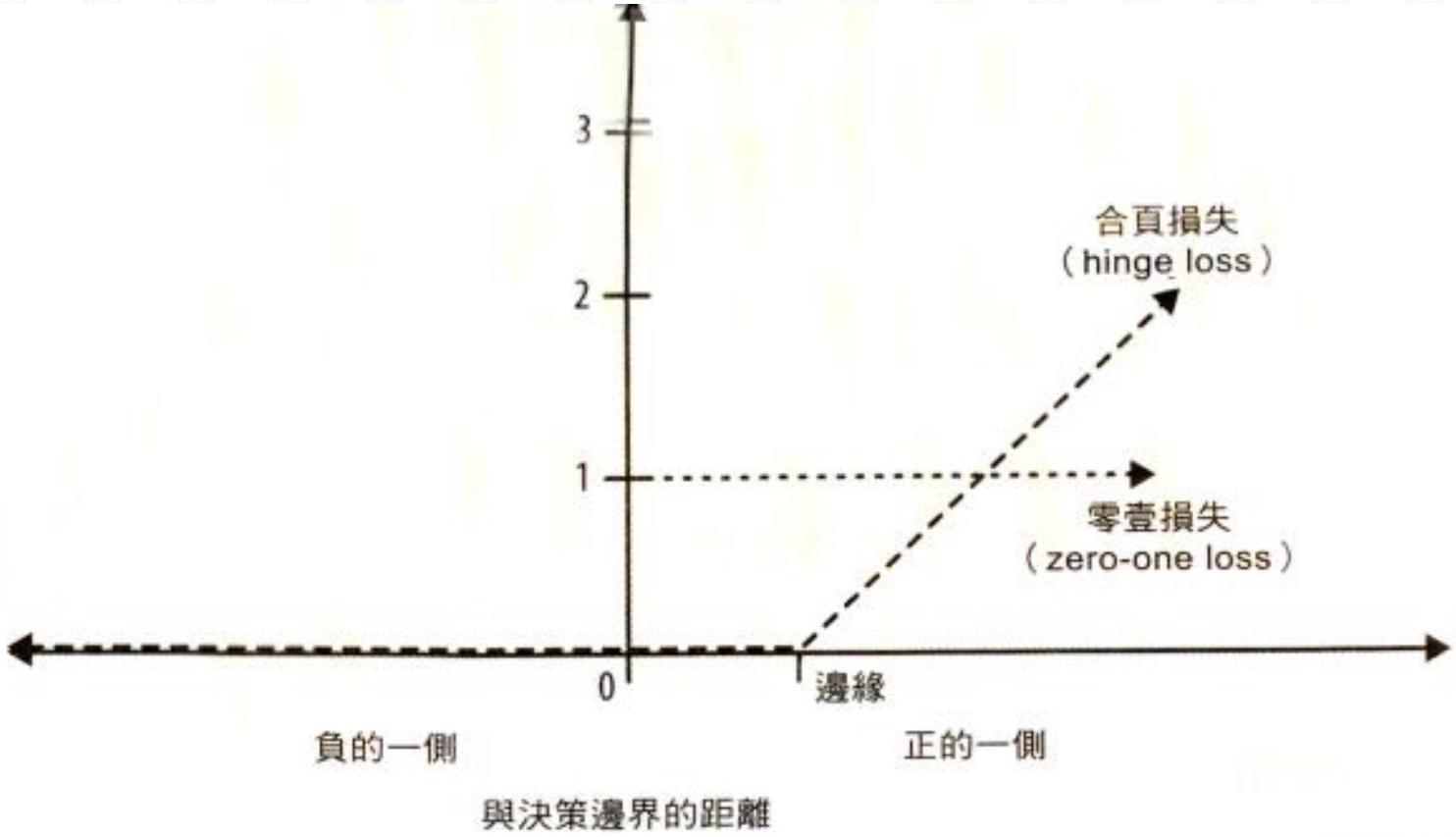
plot_estimator(clf2, X, y)



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

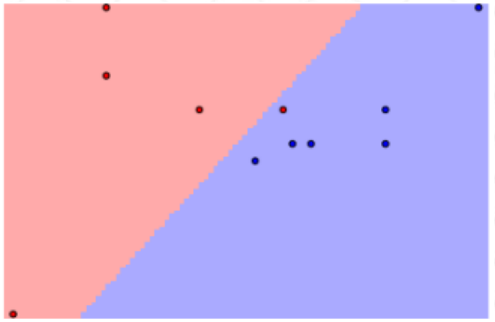


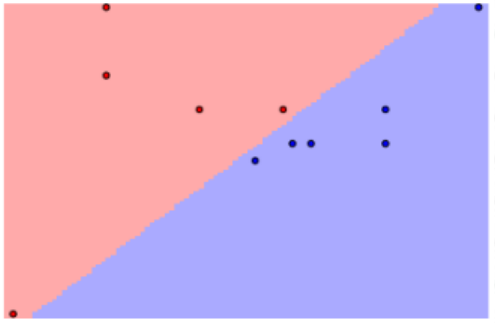
正則項 (Regularization Term)

- **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 = ∞ 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

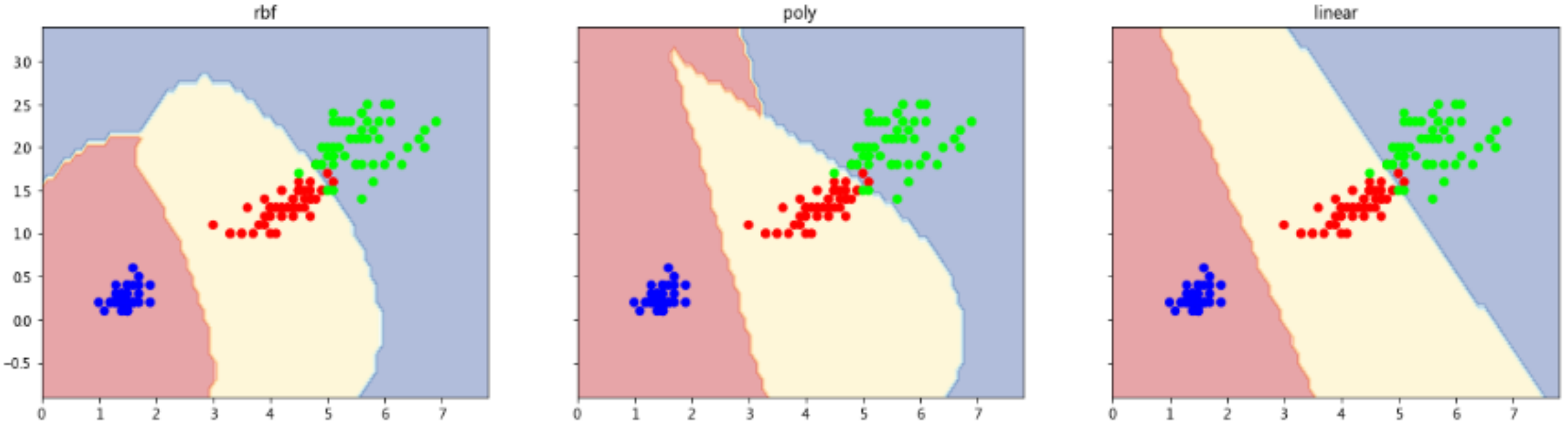
```
clf1 = SVC(kernel="rbf")
clf1.fit(X, y)

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

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

非線性Kernel

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)
```

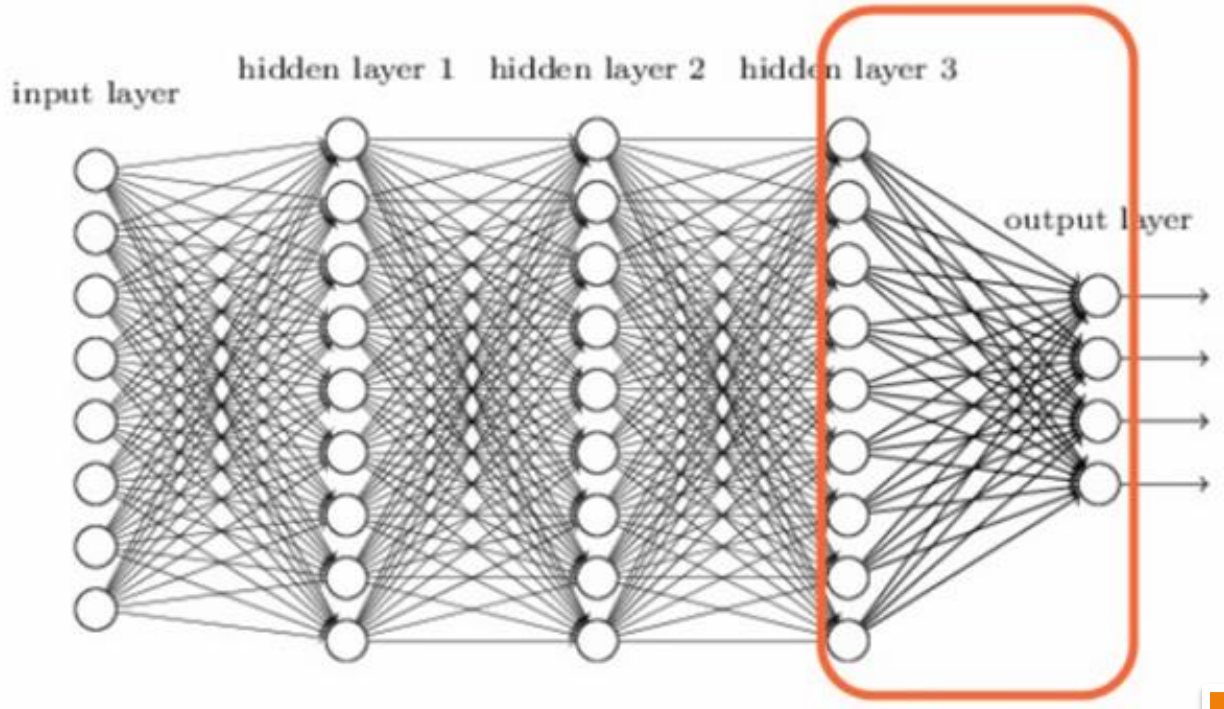
比較不同Kernel Function



深度學習

類神經網路的曙光

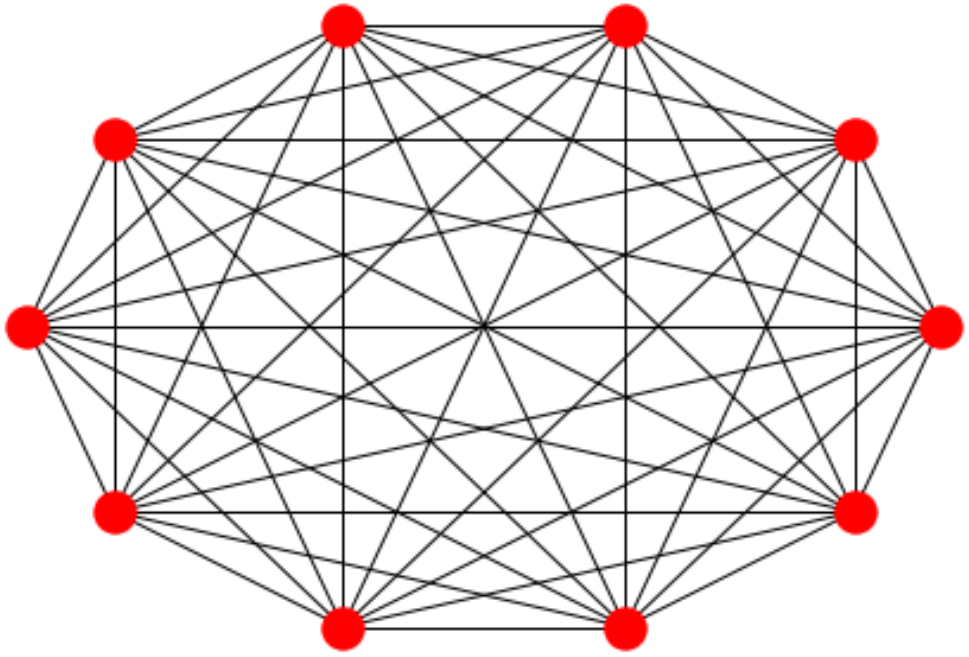
- 2006 – A fast learning algorithm for deep belief nets
  - Hinton提出用神經網路的非監督式學習來做為神經網路初始權重的指派

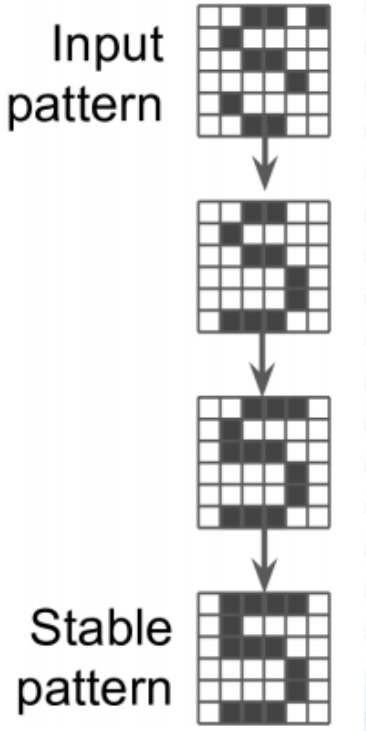


非監督式學習

Hopfield 網路

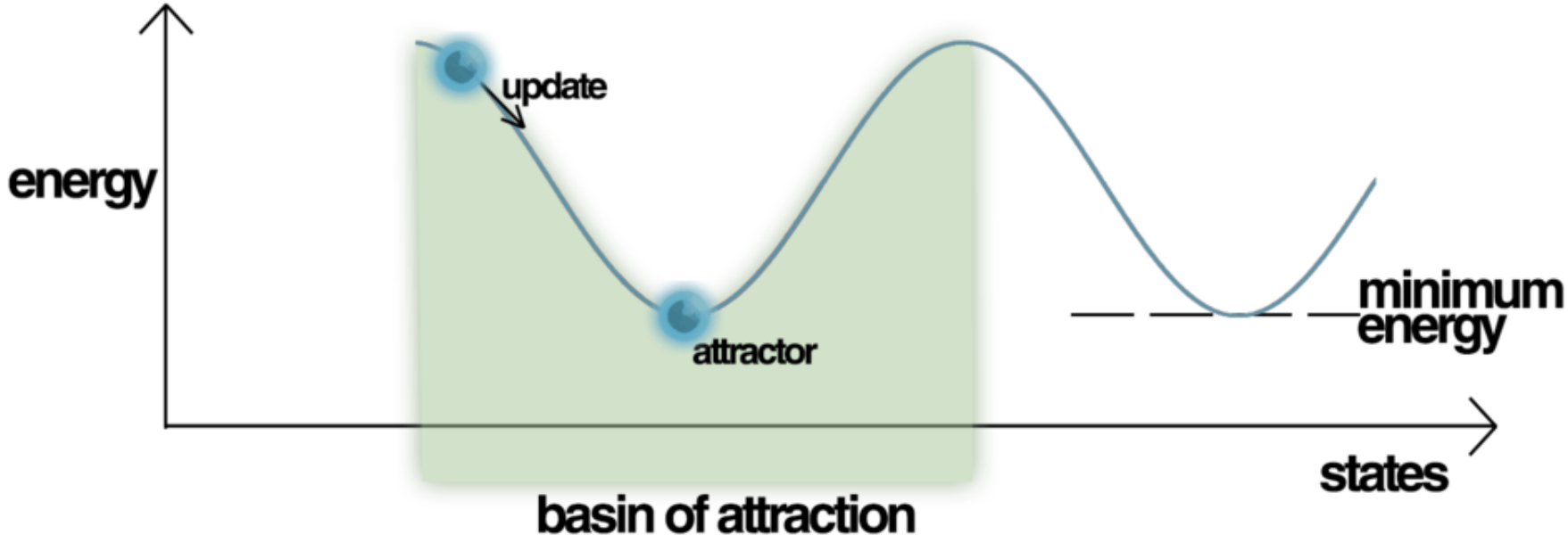
- Hopfield 網路由 J.Hopfield 於 1982年所提出，可以視為一聯想記憶(Association Memory)網路
- 教學網路關於數位 5 的圖像後，未來當網路看到一接近數位 5 的模糊圖像，將會將該圖像聯想成數位 5





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



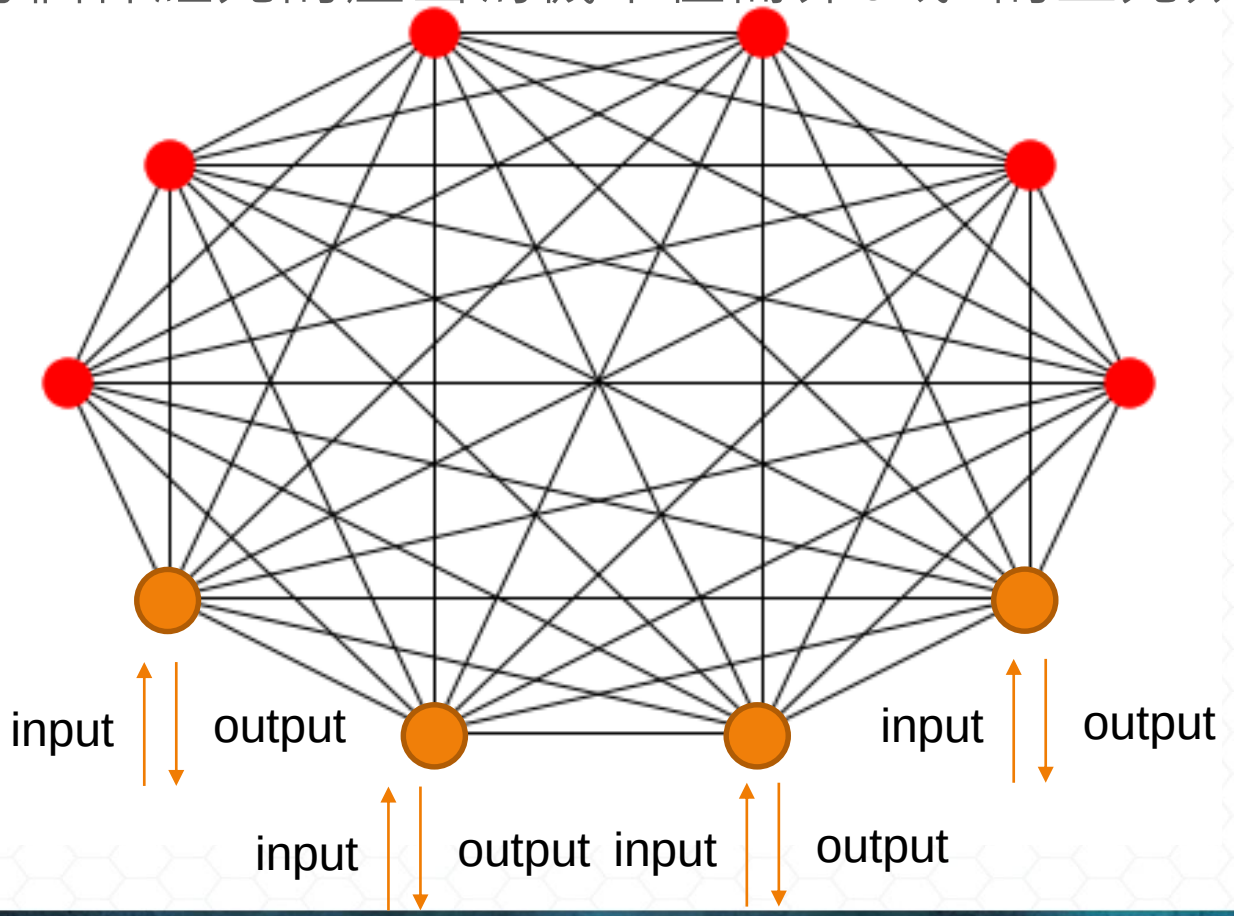
Hopfield 網路的缺陷

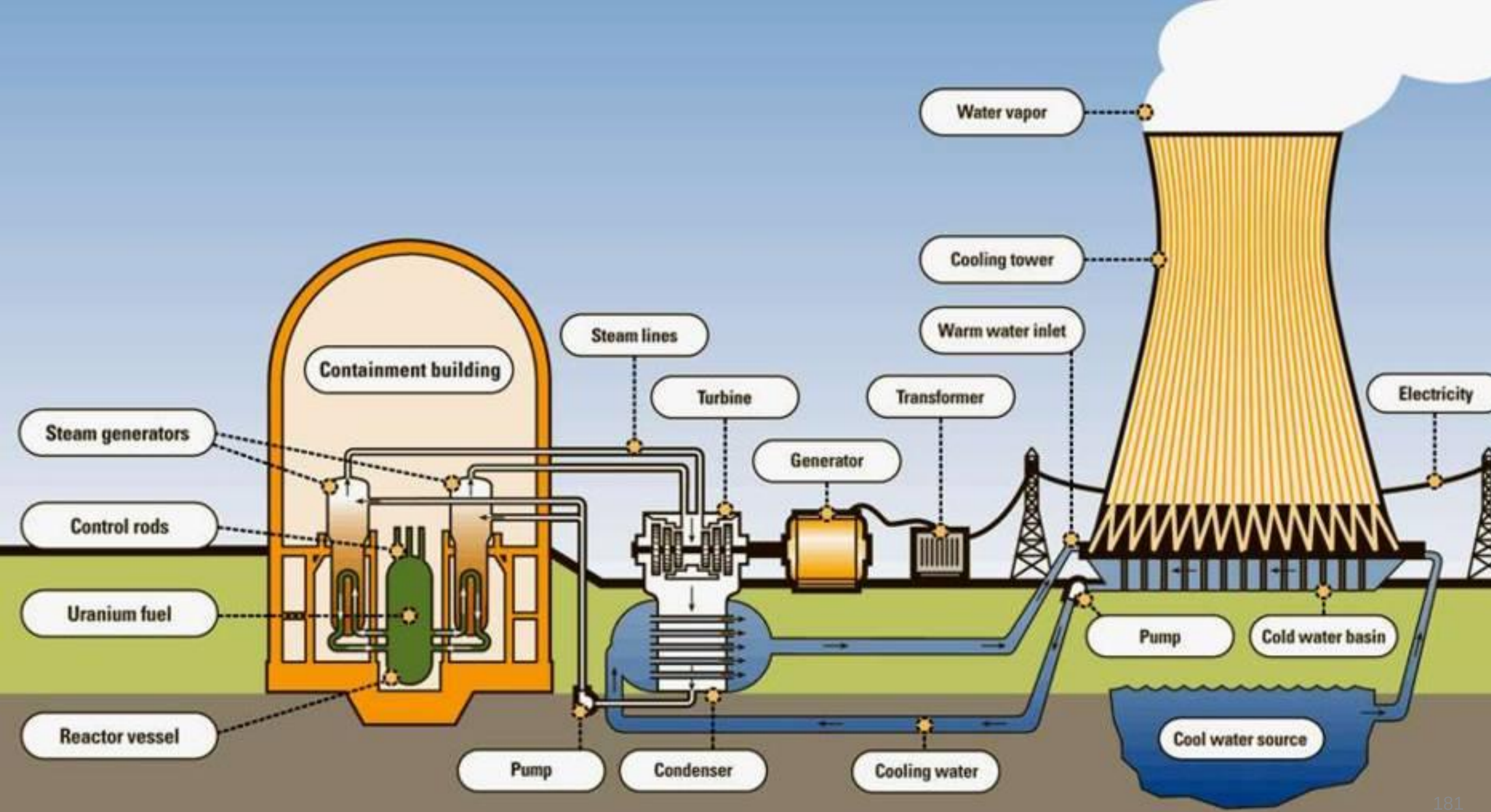
- Hopfield 網路保證向局部極小做收斂，但收斂到錯誤的局部極小值（local minimum），而非全域極小（global minimum）的情況也可能發生

- 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




Water vapor
Cooling tower
Warm water inlet
Steam lines
Containment building
Turbine
Transformer
Electricity
Steam generators
Generator
Control rods
Uranium fuel
Pump
Cold water basin
Reactor vessel
Cool water source
Pump
Condenser
Cooling water

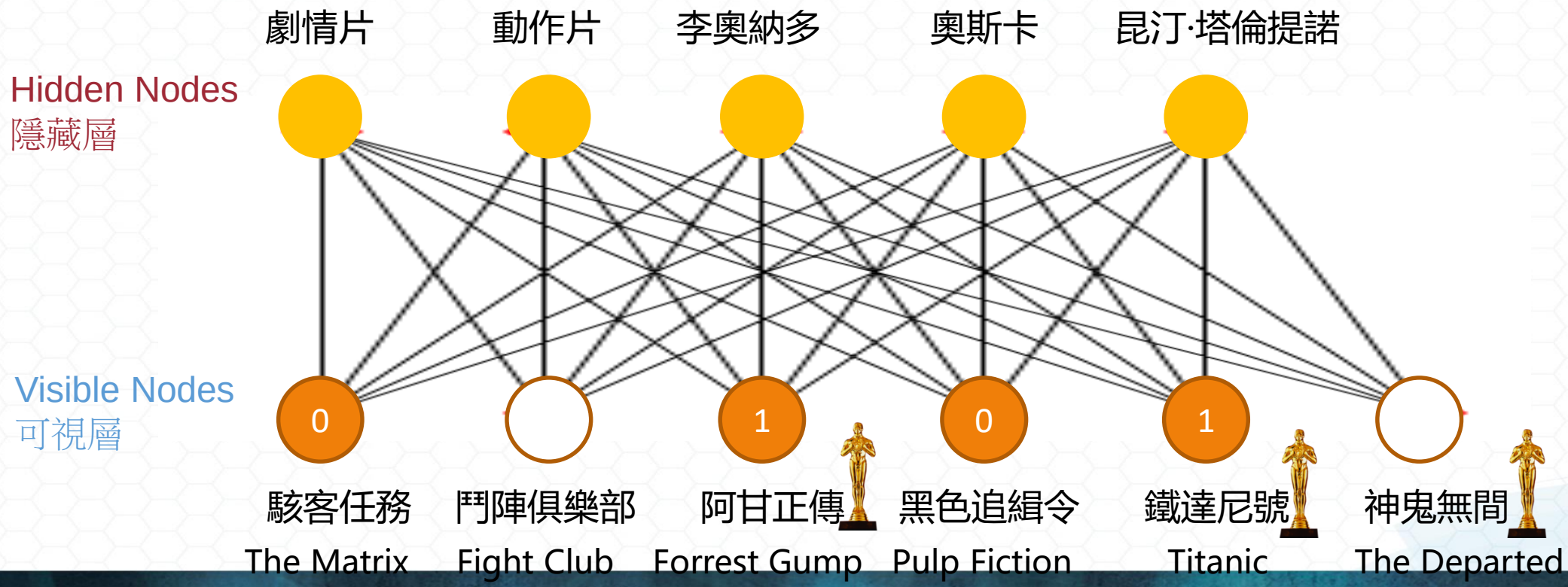
神經元輸出機率

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

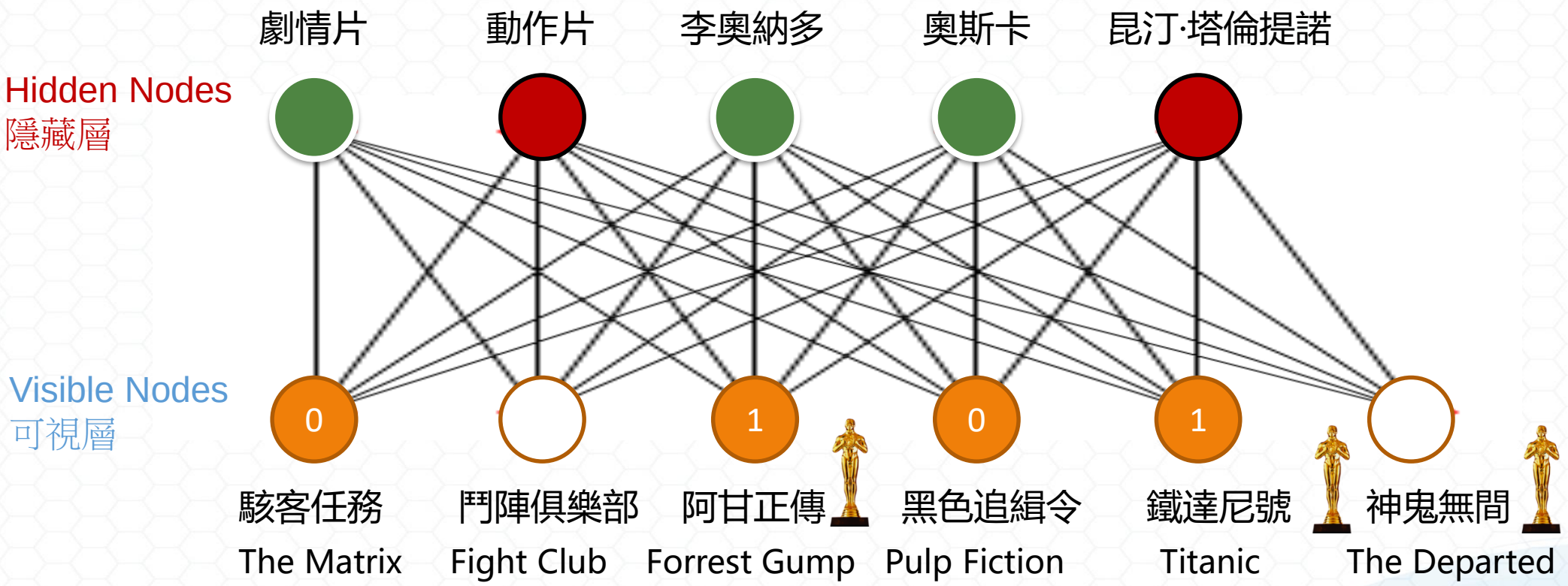
- s_j 是第 j 個神經元的狀態 (0 or 1).
- $w_{i,j}$ 是第 i 與 j 神經元連結的權重.
- b_i 是第 I 神經元的偏倚
- N 是神經元的數量.
- T 代表網路的溫度 溫度越高越可能產生隨機的輸出
- σ 代表邏輯式(logistic) 函數.

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

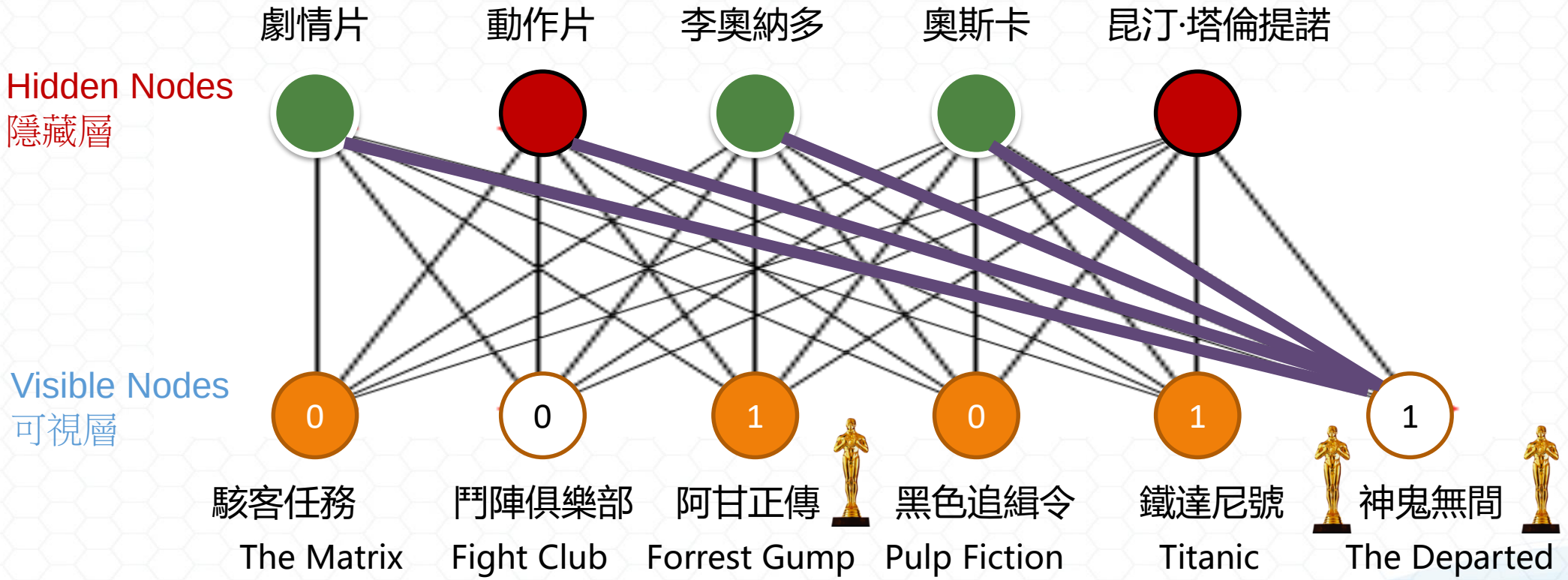


前向傳導 (Forward)



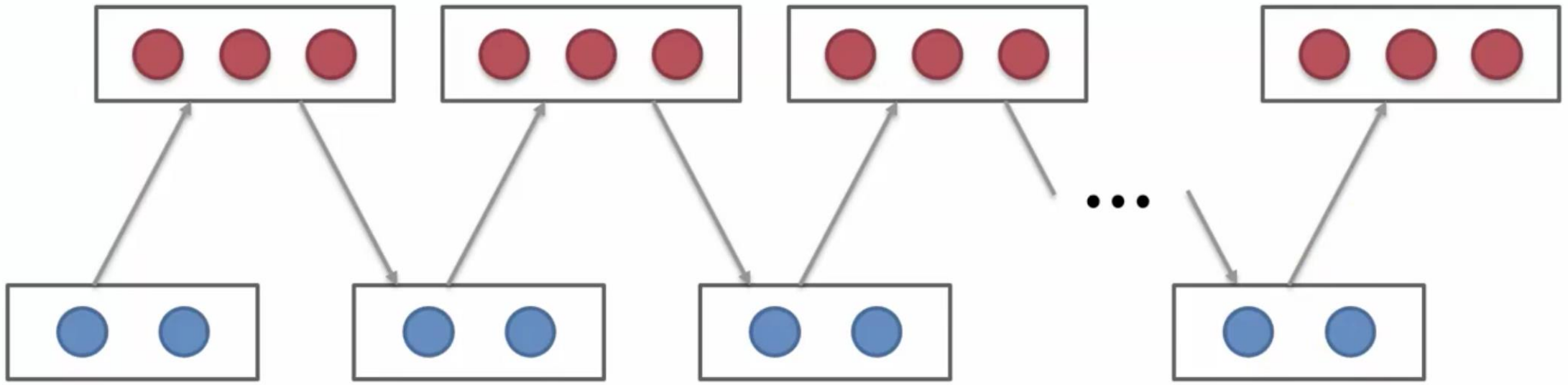
重建 (Reconstruction)

- 利用Hidden Nodes 推導 Visible Nodes 可能結果



Contrastive Divergence

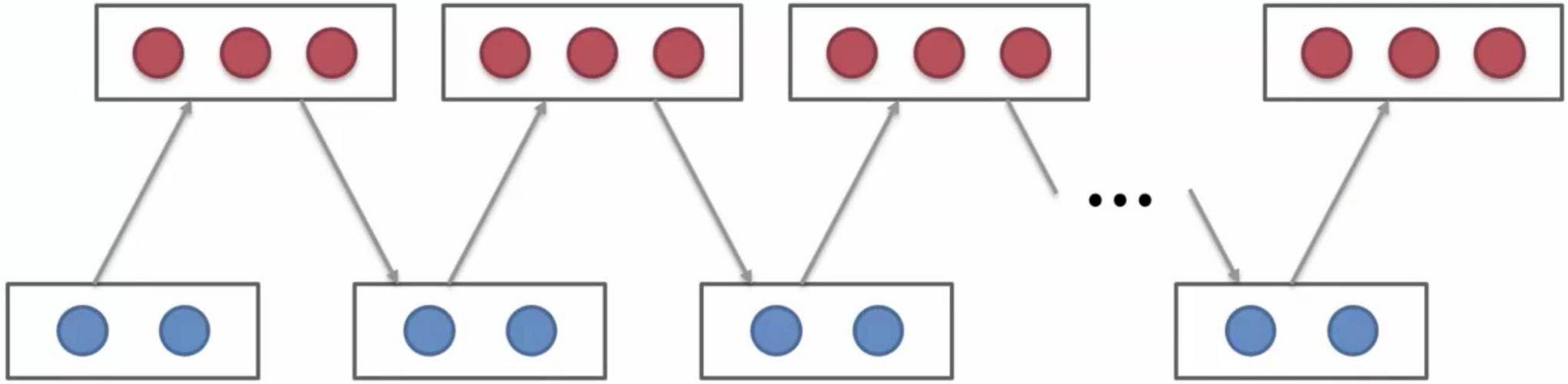
- 整個網路會不斷向前傳遞過去、再向後傳導將原始資料重建回來，期間來回數次、不斷調整每個神經元的權重和偏差值，直到「原始資料」和「重建後資料值」兩者差異被優化到最小值為止



Contrastive Divergence

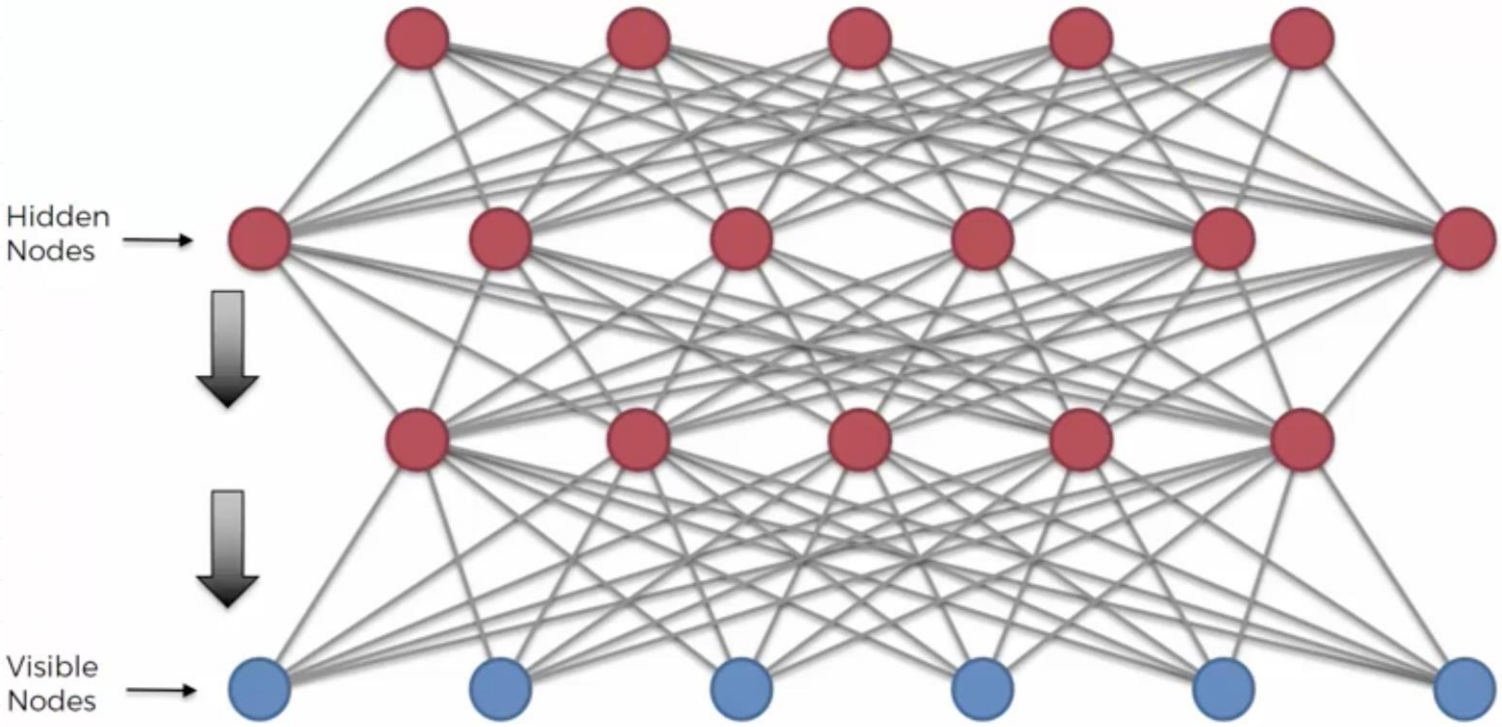
$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\mathbf{x}\mathbf{h}^T - \dot{\mathbf{x}}\dot{\mathbf{h}}^T\right)$$

- x　代表一開始在可見層的輸入
- h　代表向前傳導後的隱藏層的向量
- x' 代表重建後的可見層向量
- 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

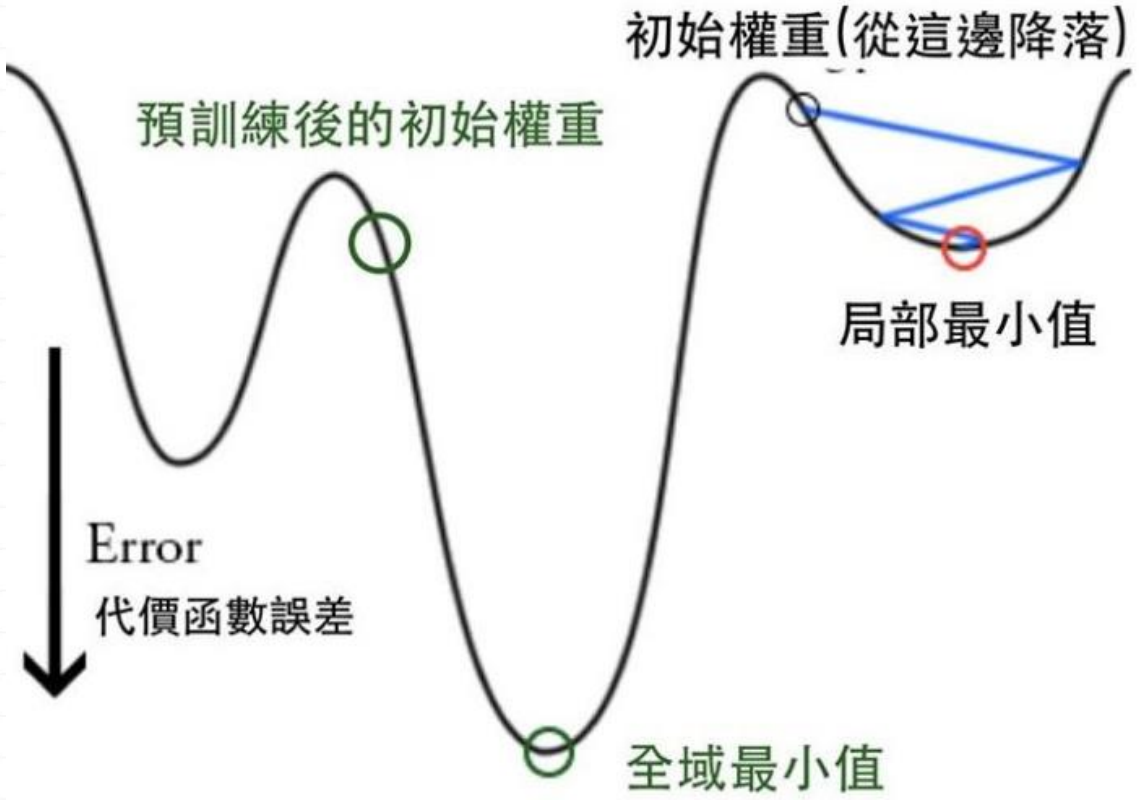
- 結合半監督學習 - 透過少量的標籤資料 (labeled data)，對深度信念網路的神經元權重和偏差值進行微調 ，讓精確度更高

- 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

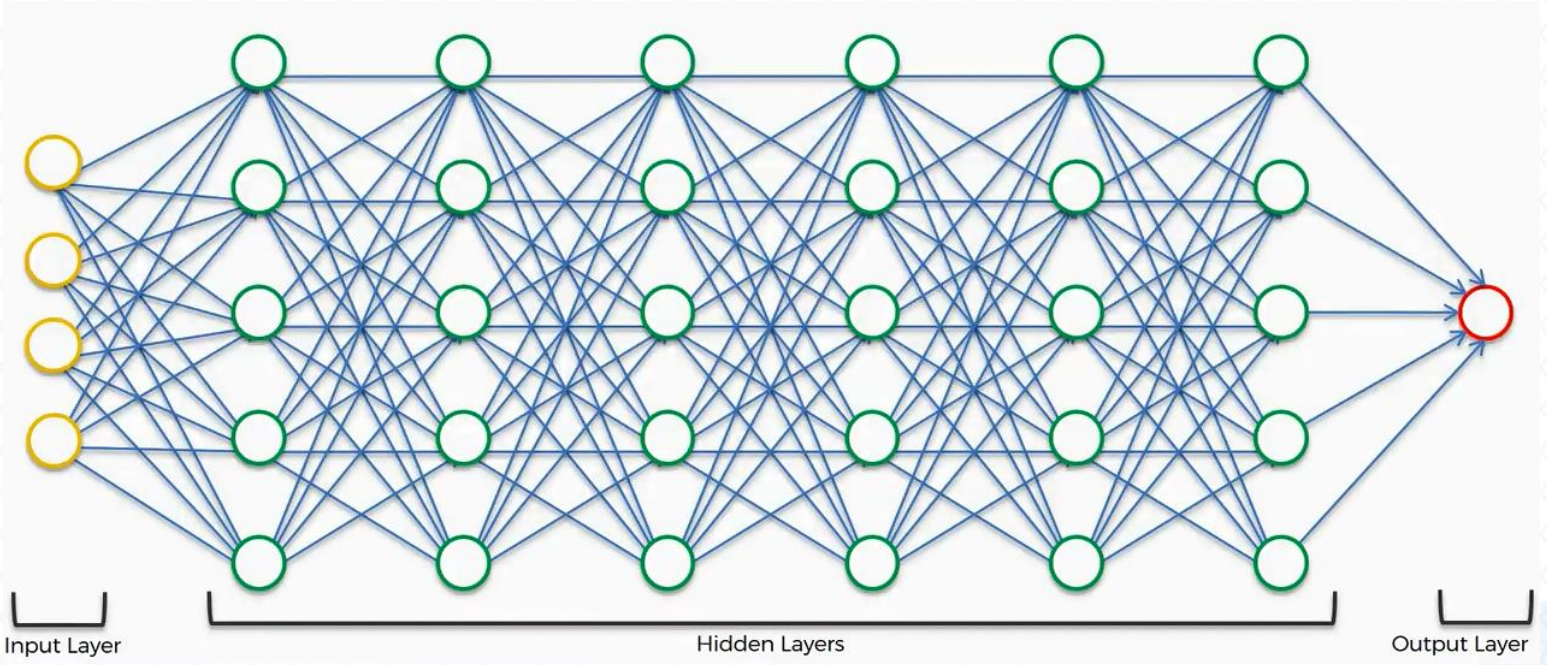
- **傳統的神經網路隨機初始化網路中的權值，導致網路很容易收斂到局部最小值**

  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絡的限制，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

IMAGENET

14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications CoolStuff About

Not logged in. Login | Signup

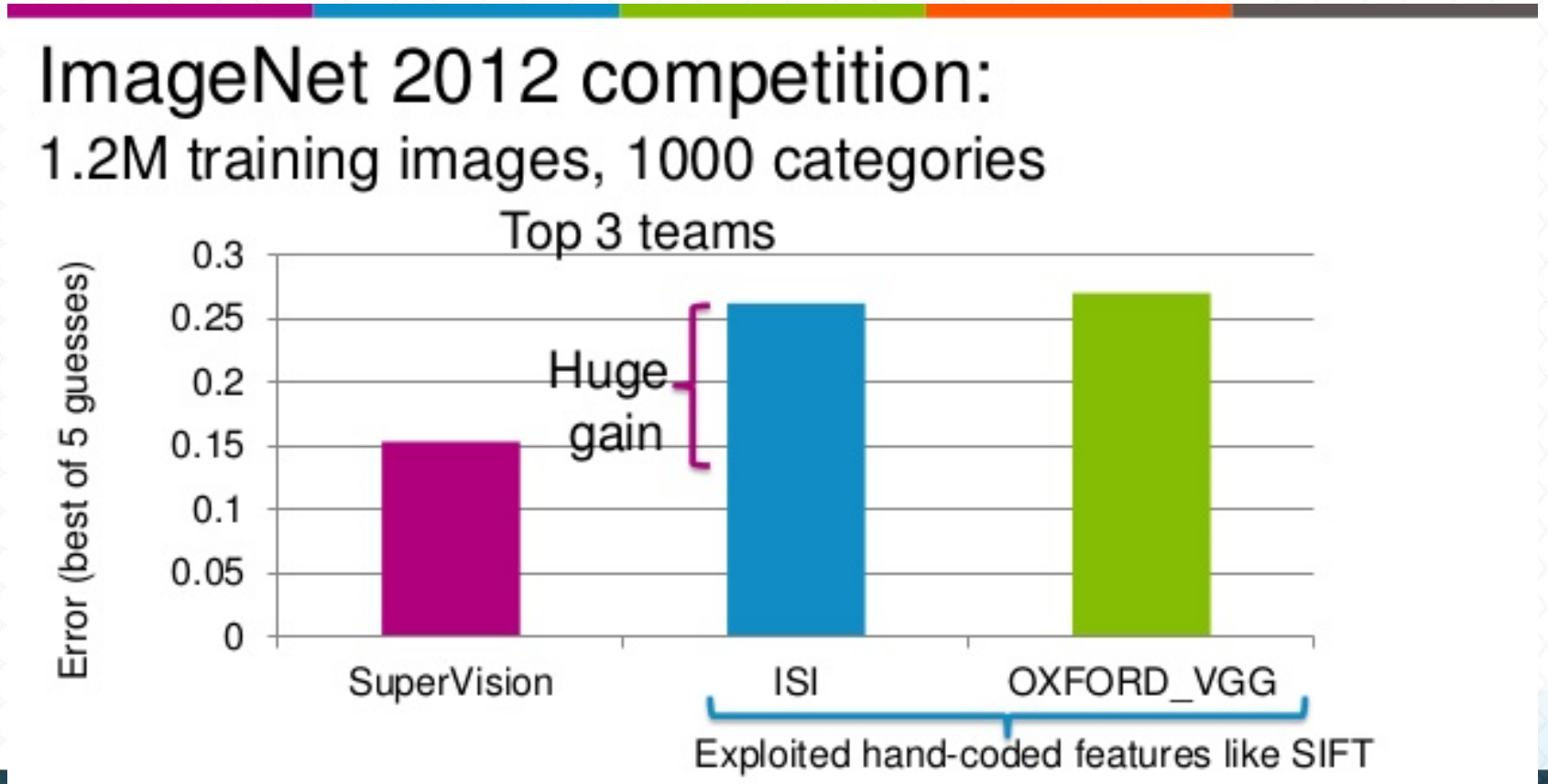
ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource 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

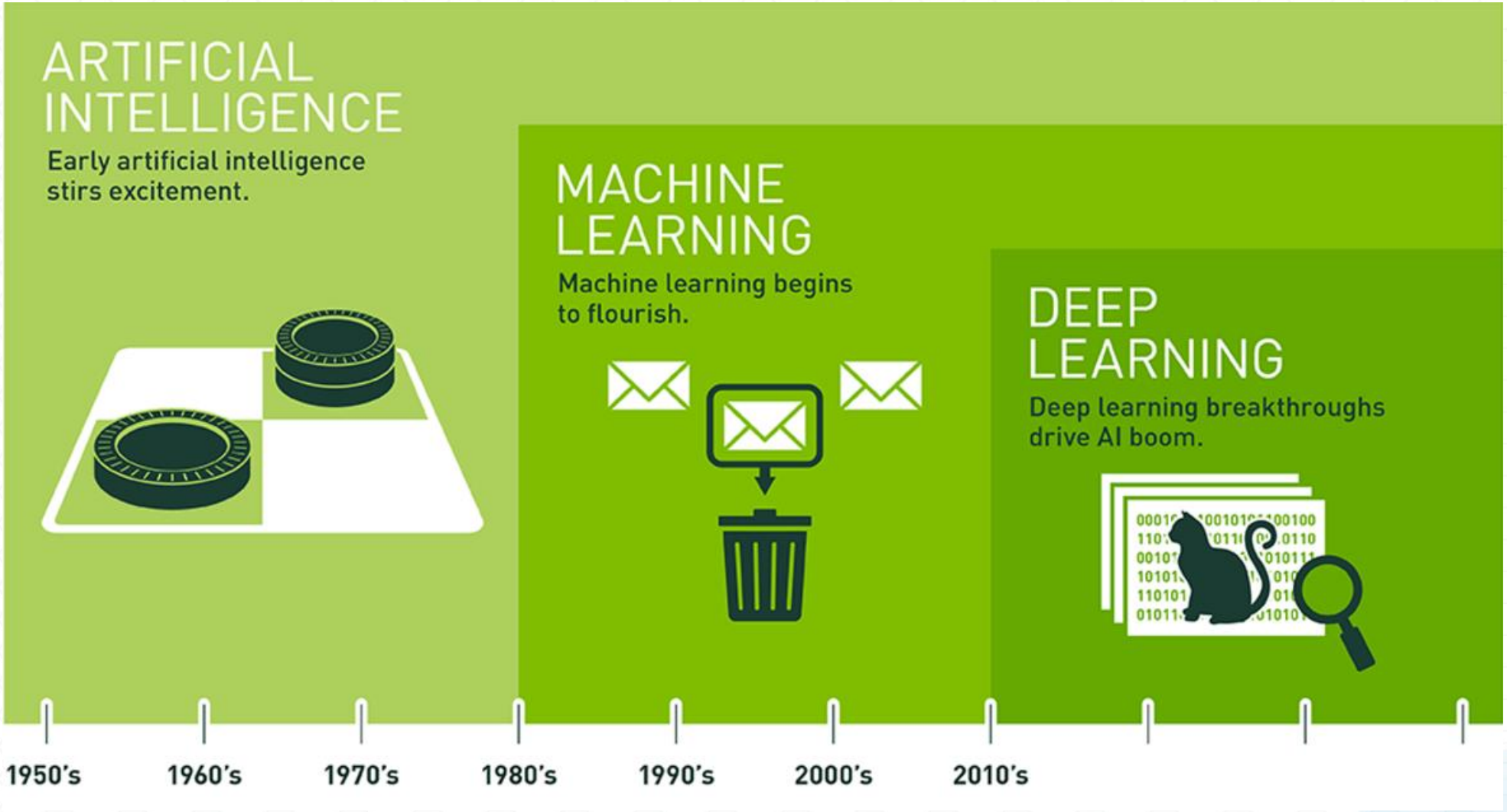
- Hinton 的兩個學生以 SuperVision 的隊伍名參賽，以 16.42% 的錯誤率遠勝第二名的 26.22%



GPU + 深度學習

- 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



TensorFlow & Keras

深度學習框架


Caffe
Lasagne
theano
TensorFlow
DSSTNE
dmlc
mxnet
Microsoft
CNTK
torch
DL4J
DEEPLEARNING4J
K

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享的力量，加速深度學習的進展

TensorFlow

- TensorFlow是利用資料流圖(Data Flow Graphs)來表達數值運算的開放式原始碼函式庫。資料流圖中的節點(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)

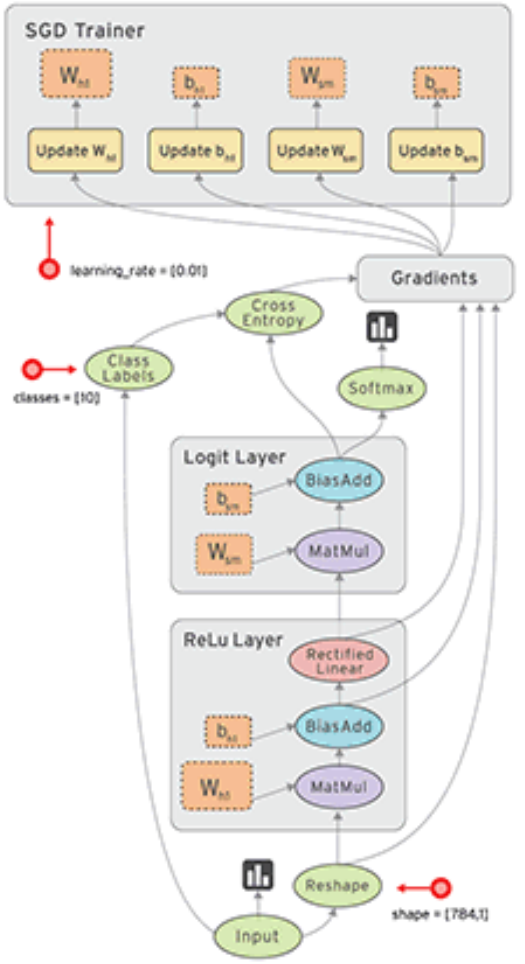
- TensorFlow
  - Tensor：是中文翻譯是張量，其實就是一個n維度的陣列或列表。如一維 Tensor 就是向量，二維 Tensor 就是矩陣等等
  - Flow：是指 Graph 運算過程中的資料流

Data Flow Graphs

資料流圖(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析，用來傳遞操作之間互相使用的多維陣列(Tensors)



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

安裝TensorFlow 與 Keras

- GPU 版本 (非必要)

>>> pip install --upgrade tensorflow-gpu

- CPU 版本

>>> pip install --upgrade tensorflow

- Keras 安裝

>>> pip install keras -U --pre

使用者流失分析 (Churn Analysis)

- 預測哪些使用者會流失?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	RowNumb	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfPro	HasCrCard	IsActiveMe	EstimatedS	Exited
2	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.9	1
3	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.6	0
4	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	11393..6	1
5	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
6	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8	1	1	1	79084.1	0
7	6	15574012	Chu	645	Spain	Male	44	8	113755.8	2	1	0	149756.7	1
8	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
9	8	15656148	Obinna	376	Germany	Female	29	4	115046.7	4	1	0	119346.9	1
10	9	15792365	He	501	France	Male	44	4	142051.1	2	0	1	74940.5	0
11	10	15592389	H?	684	France	Male	27	2	134603.9	1	1	1	71725.73	0
12	11	15767821	Bearce	528	France	Male	31	6	102016.7	2	0	0	80181.12	0
13	12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390.01	0
14	13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26260.98	0

資料預先處理

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

轉換類別資料

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

將資料分為訓練與測試資料集

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

標準化特徵

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

建立 ANN

```
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initializing the ANN
classifier = Sequential()
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

預測與驗證模型

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

評估 ANN

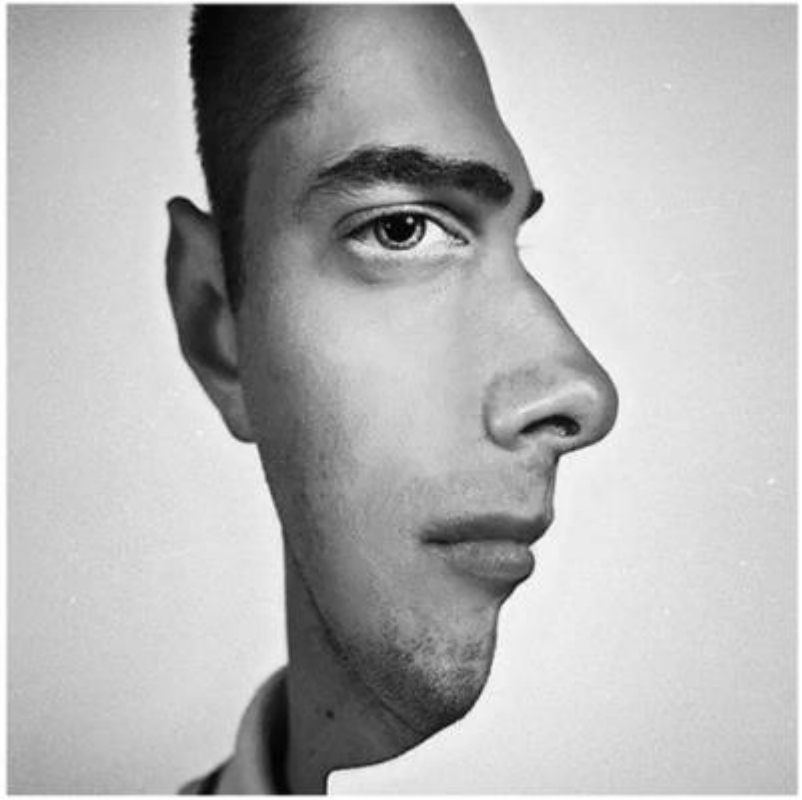
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense
def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, epochs = 100)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_jobs = -1)
mean = accuracies.mean()
variance = accuracies.std()
```

使用GridSearch 調整 ANN

```
from sklearn.model_selection import GridSearchCV

classifier = KerasClassifier(build_fn = build_classifier)
parameters = {'batch_size': [25, 32],
              'epochs': [100, 500],
              'optimizer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

卷積神經網路
(Convolution Neural Network)

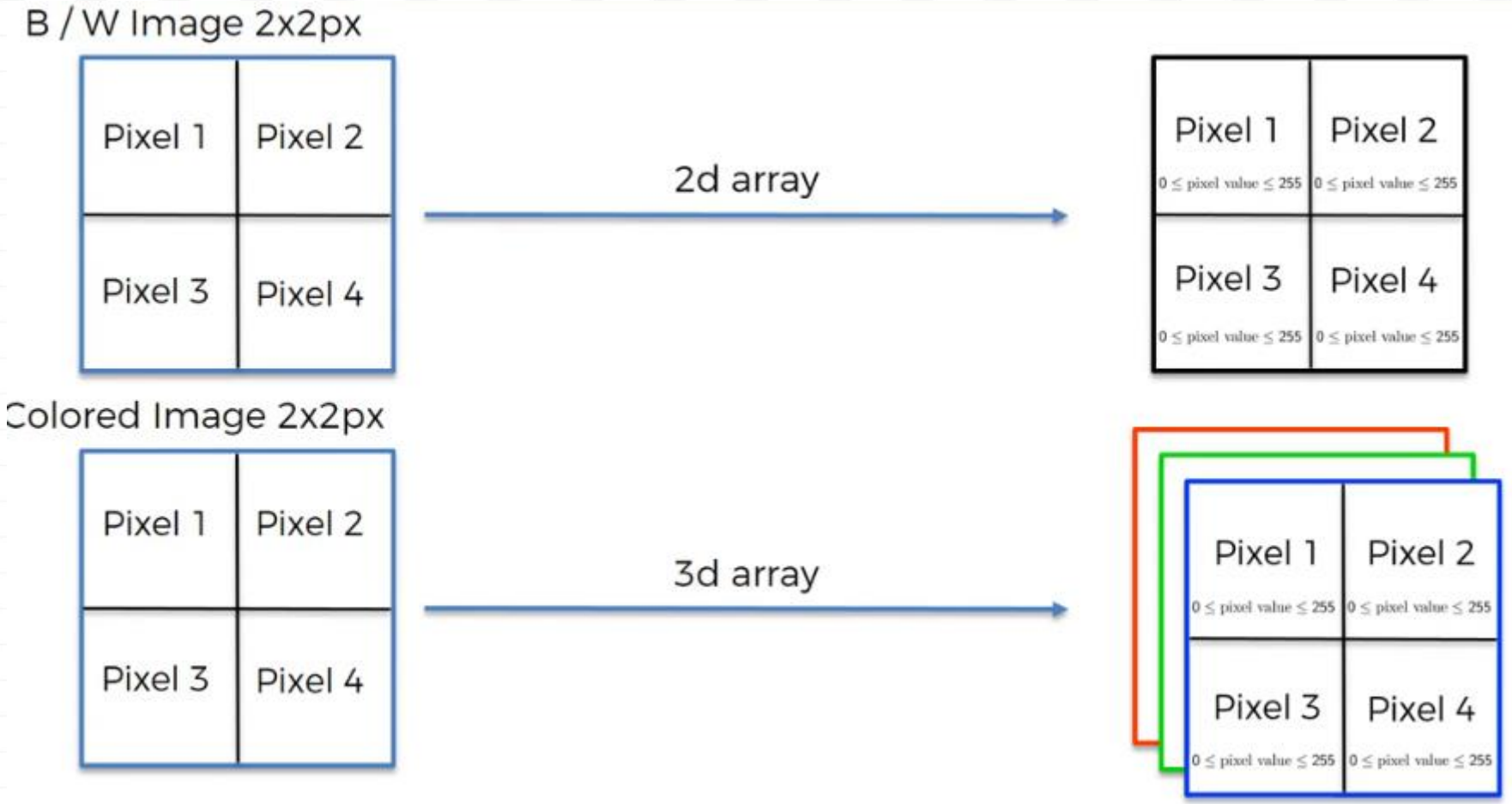
你看到了什麼?



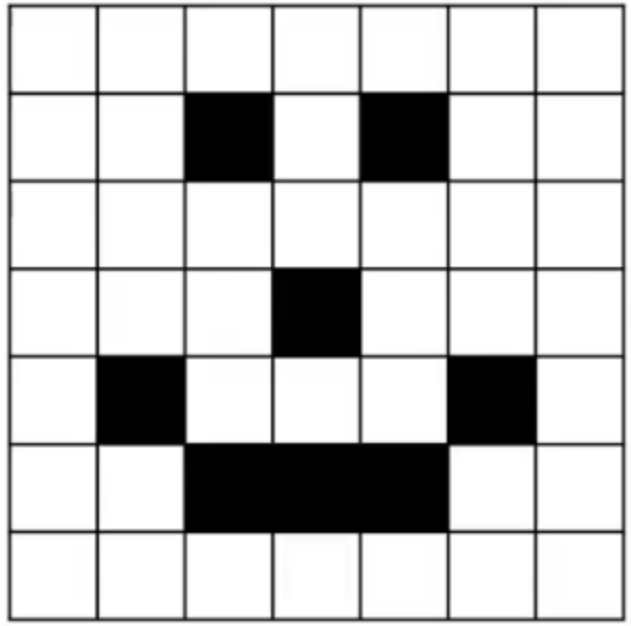
你又看到了什麼?

人腦是透過特徵分類影像

如何解讀圖片


B / W Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
2d array
Pixel 1
0 ≤ pixel value ≤ 255
Pixel 2
0 ≤ pixel value ≤ 255
Pixel 3
0 ≤ pixel value ≤ 255
Pixel 4
0 ≤ pixel value ≤ 255
Colored Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
3d array
Pixel 1
0 ≤ pixel value ≤ 255
Pixel 2
0 ≤ pixel value ≤ 255
Pixel 3
0 ≤ pixel value ≤ 255
Pixel 4
0 ≤ pixel value ≤ 255

把圖片變成數學矩陣



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

CNN 方法

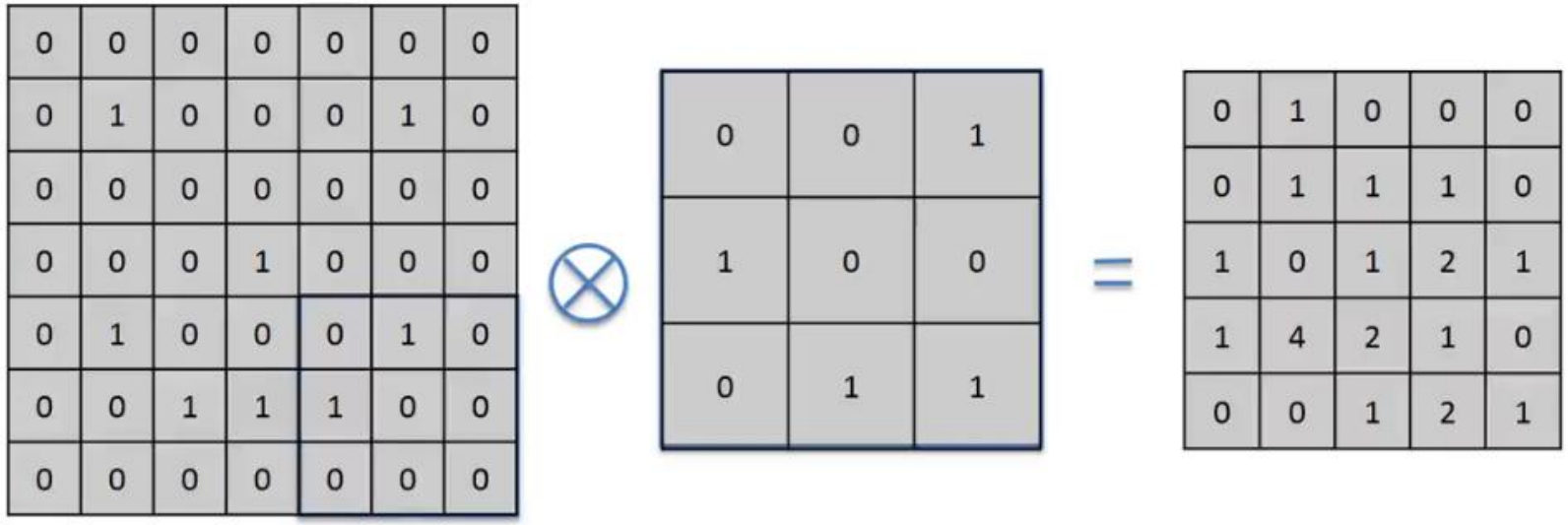
STEP 1: Convolution

STEP 2: Max Pooling

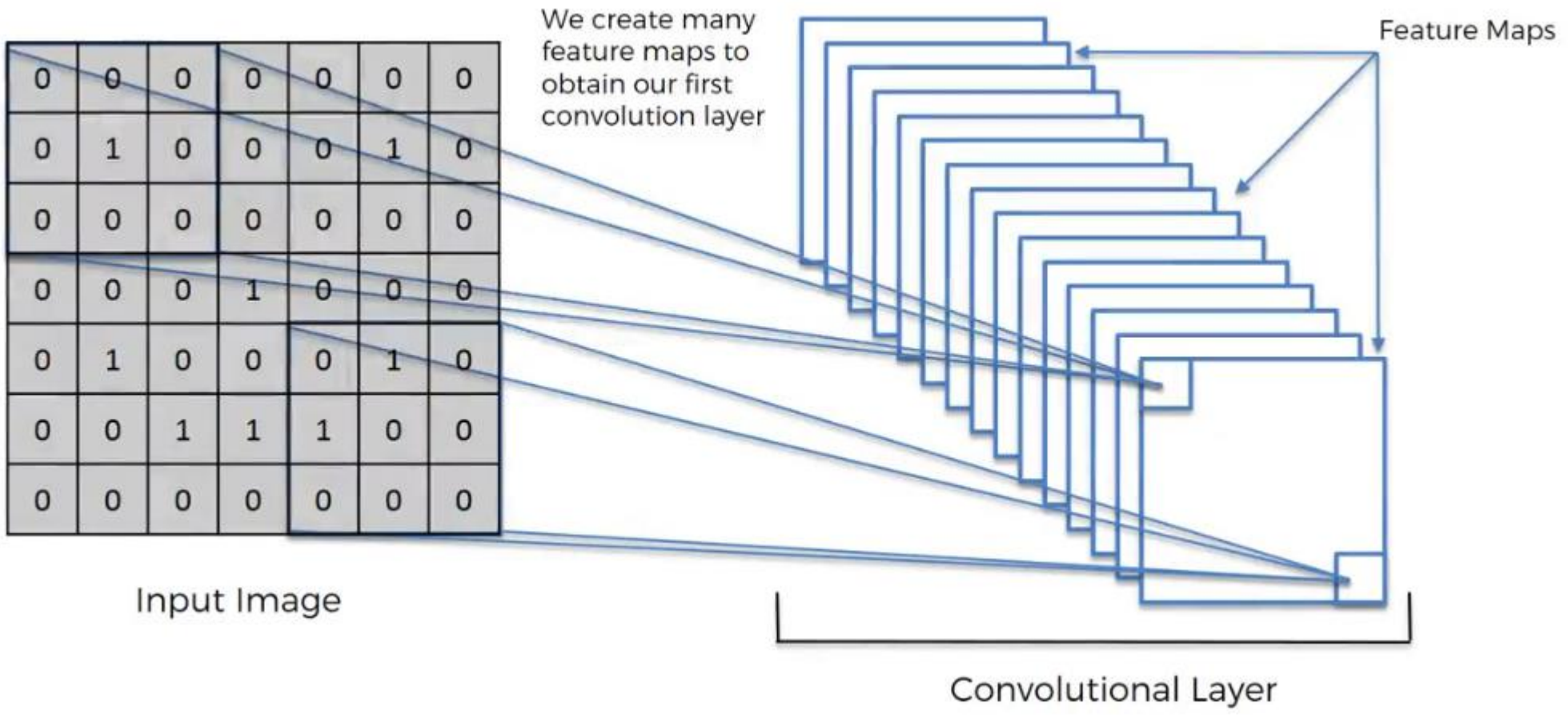
STEP 3: Flattening

STEP 4: Full Connection

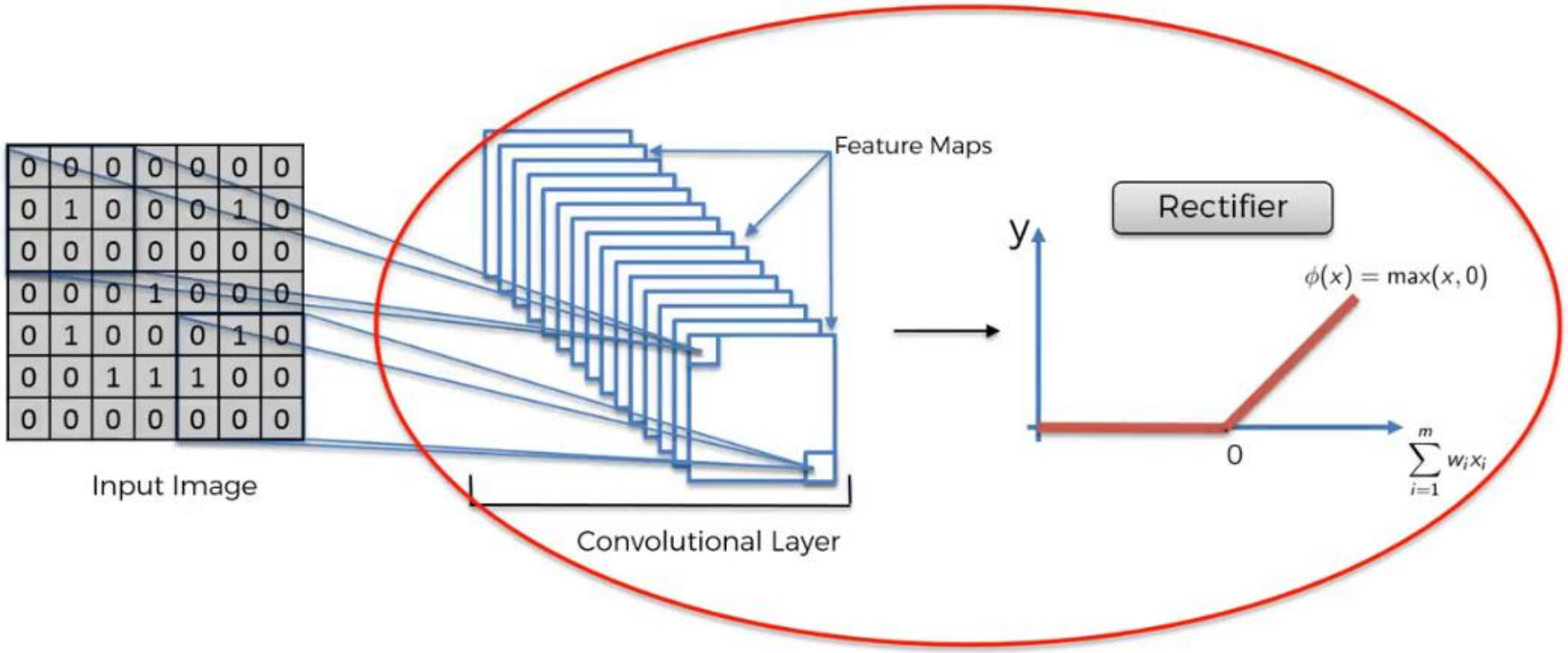
Step1. Convolution



Step1. Convolution



Step1. Convolution (ReLu)

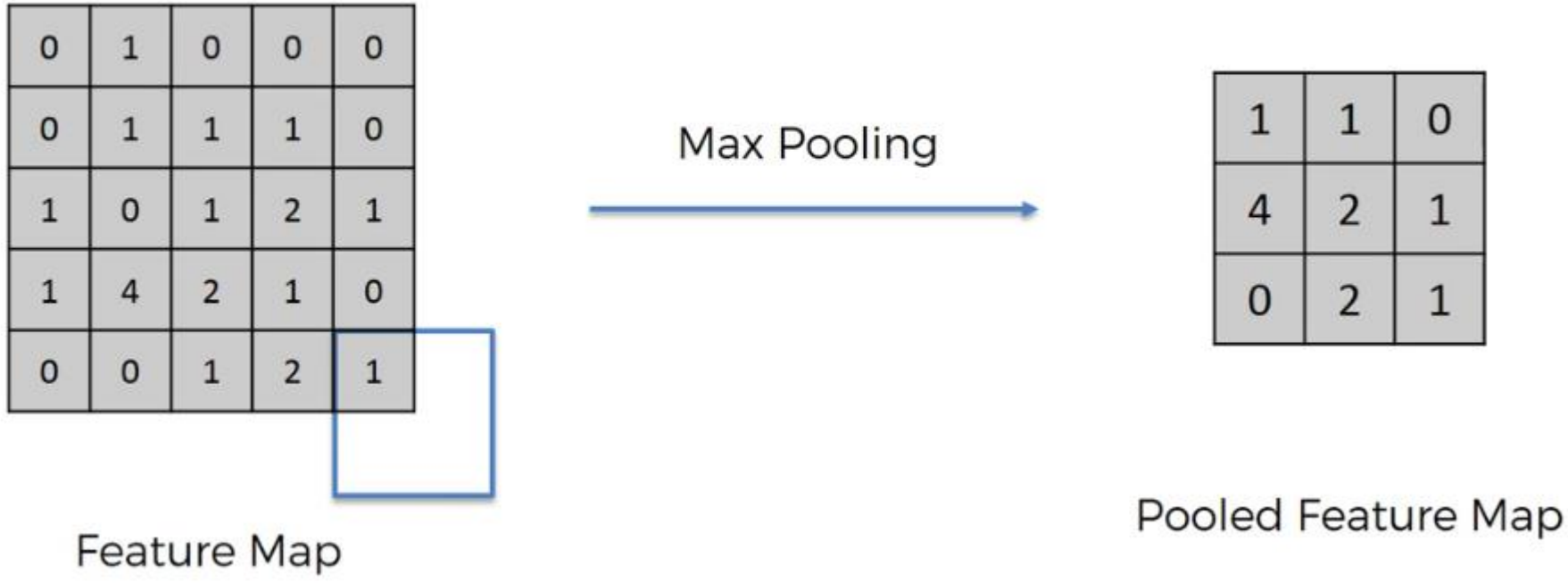


Step1. Convolution (ReLu)

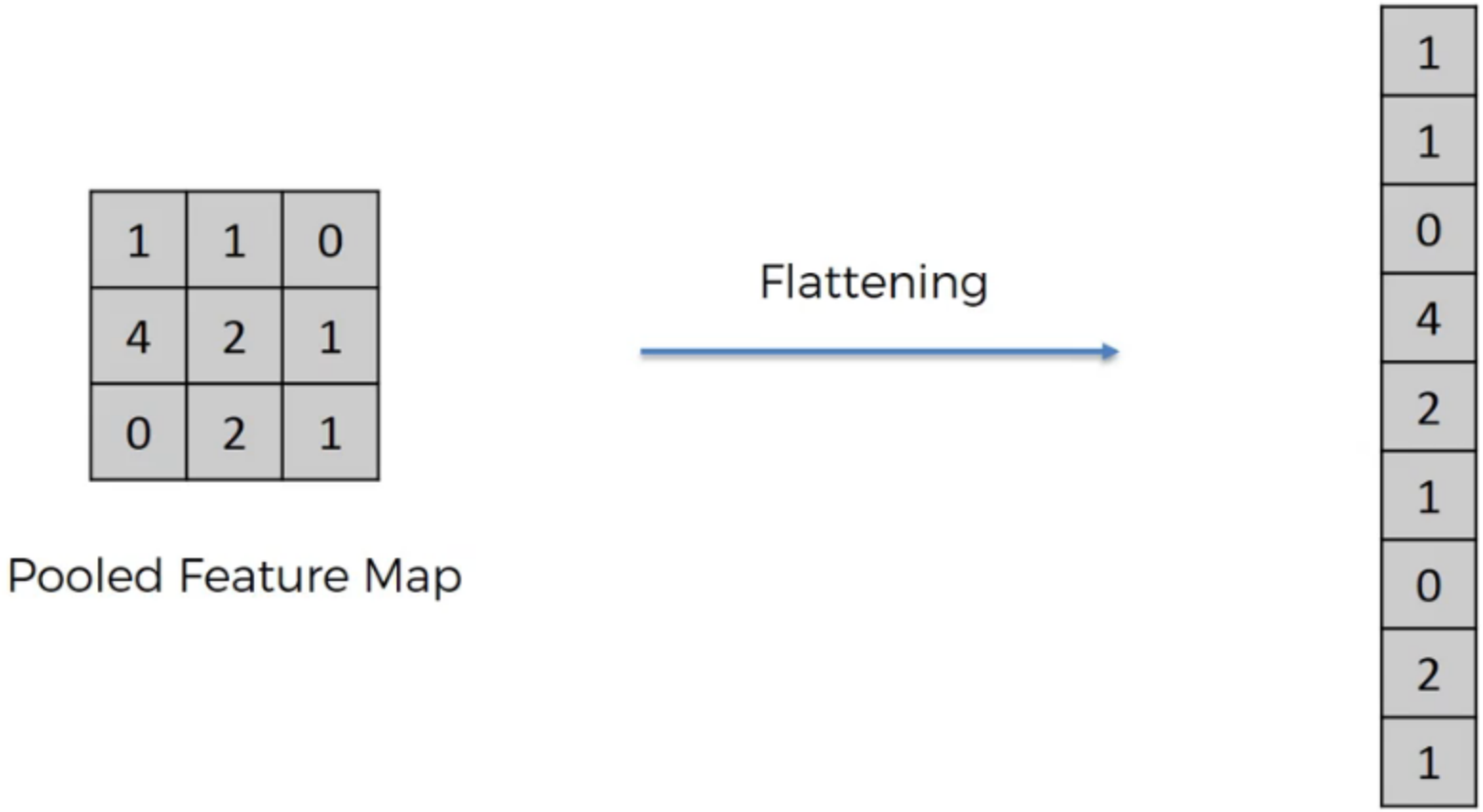
如何用不同角度辨認獵豹?

如何用不同角度辨認獵豹?

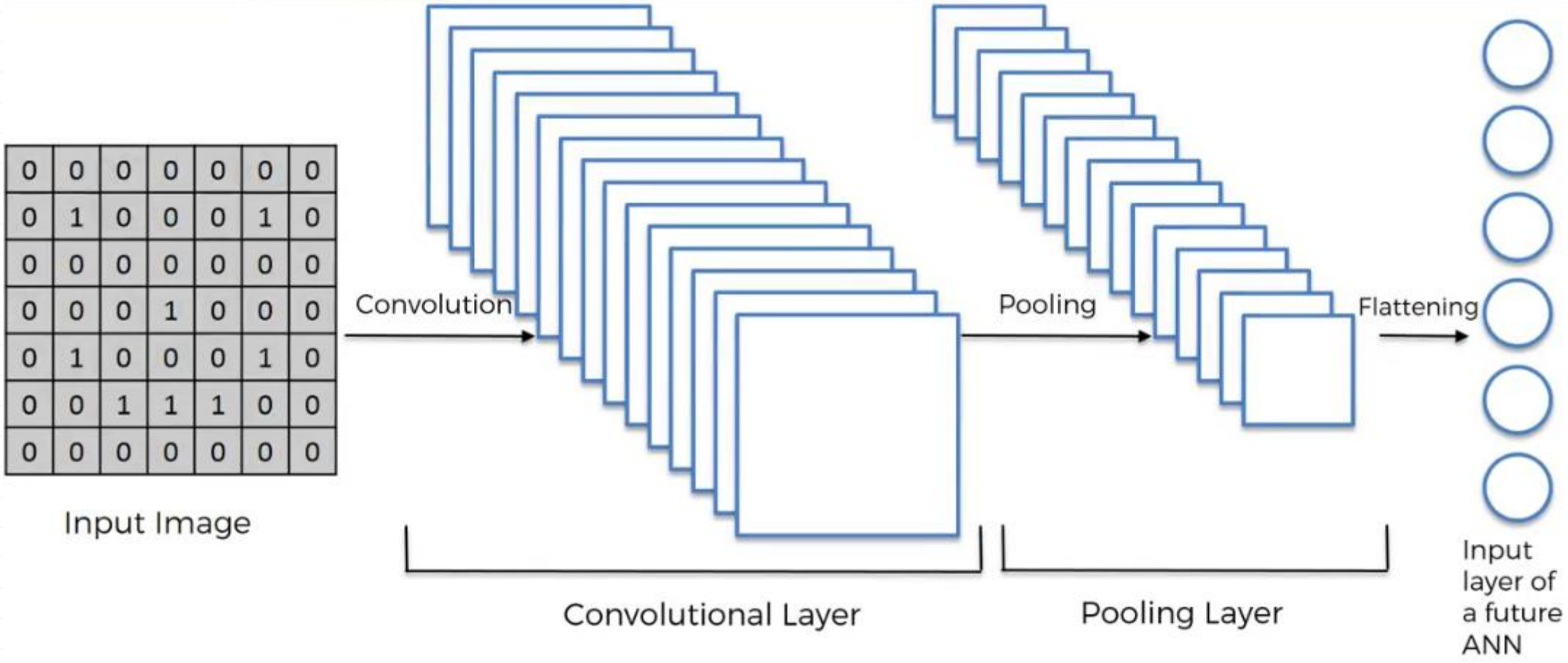
Step 2. Max Pooling



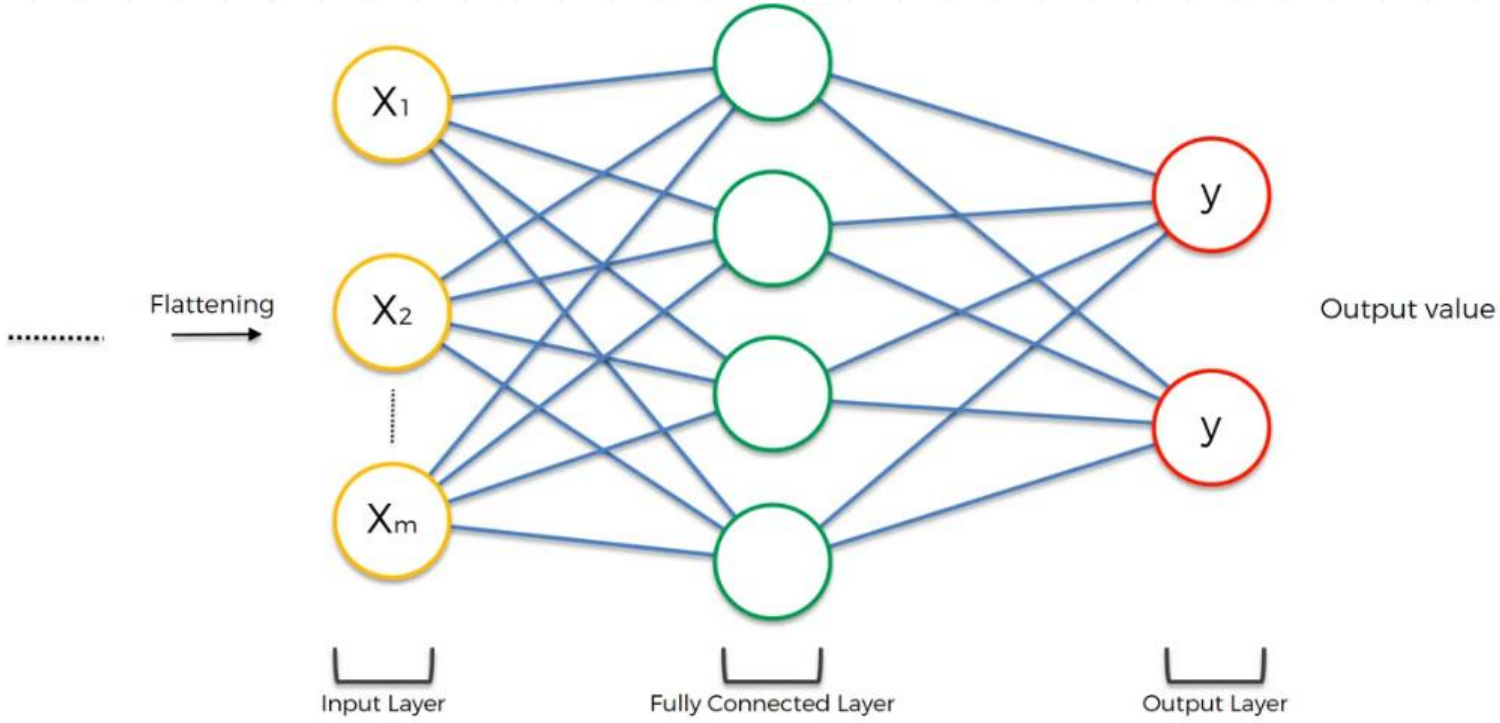
Step 3. Flattening



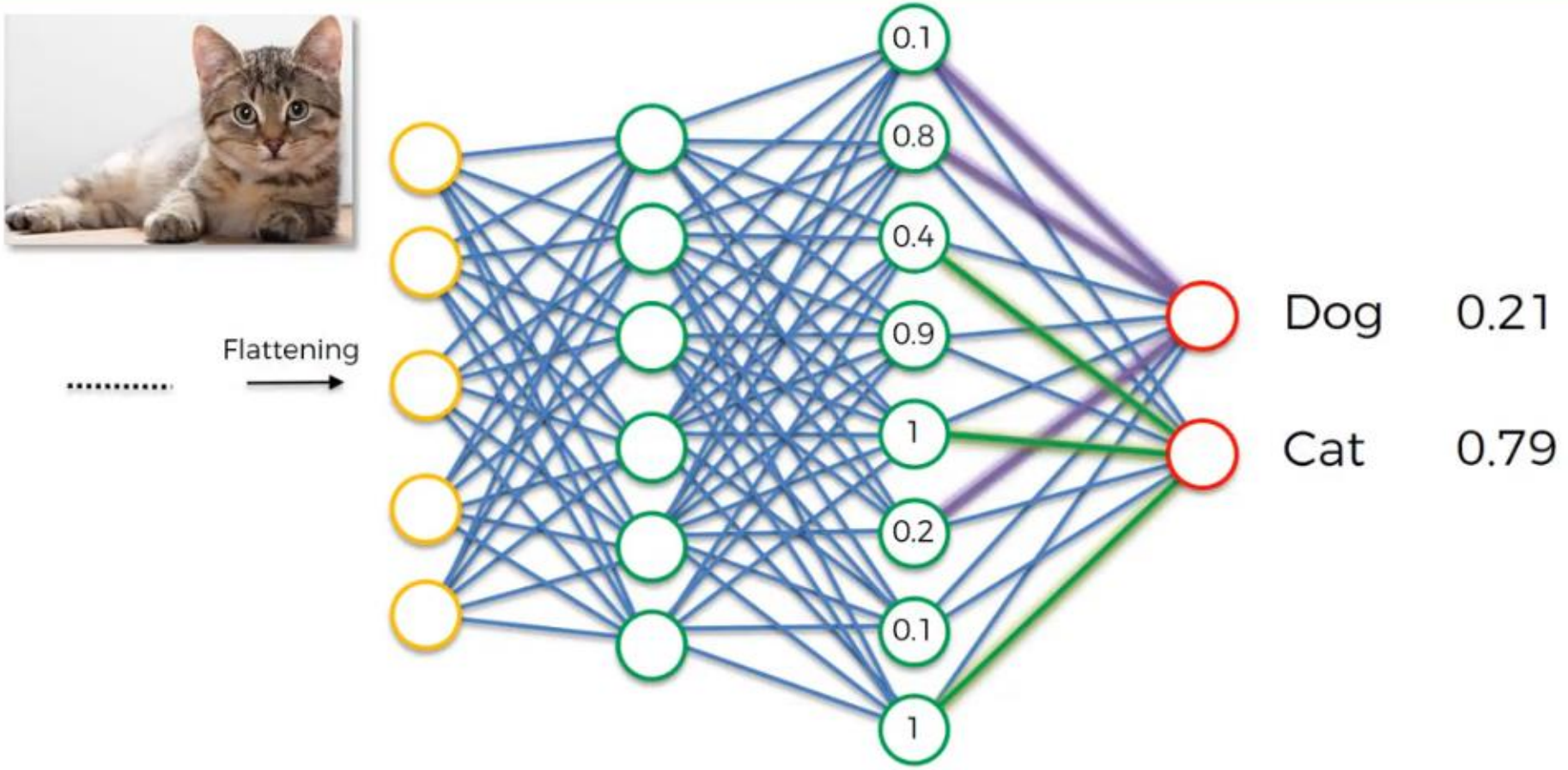
Step 3. Flattening



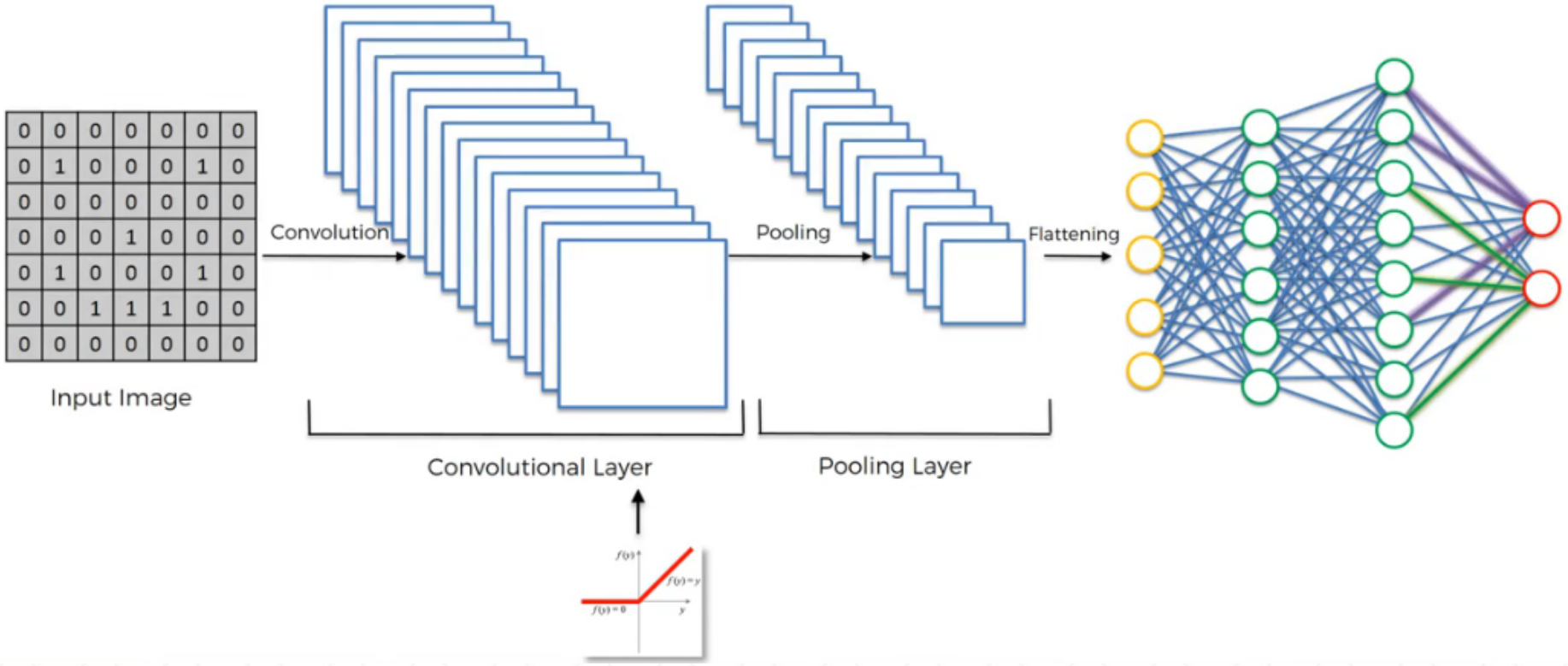
Step 4. Fully Connected Network



建立神經網路分類影像



Convolution Neural Network



讓電腦識別狗或貓

建立CNN

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Initialising the CNN
classifier = Sequential()
```

Convolution & Pooling

```
# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation =
'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Flattening & Full Connection

```
# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics
= ['accuracy'])
```

建立模型

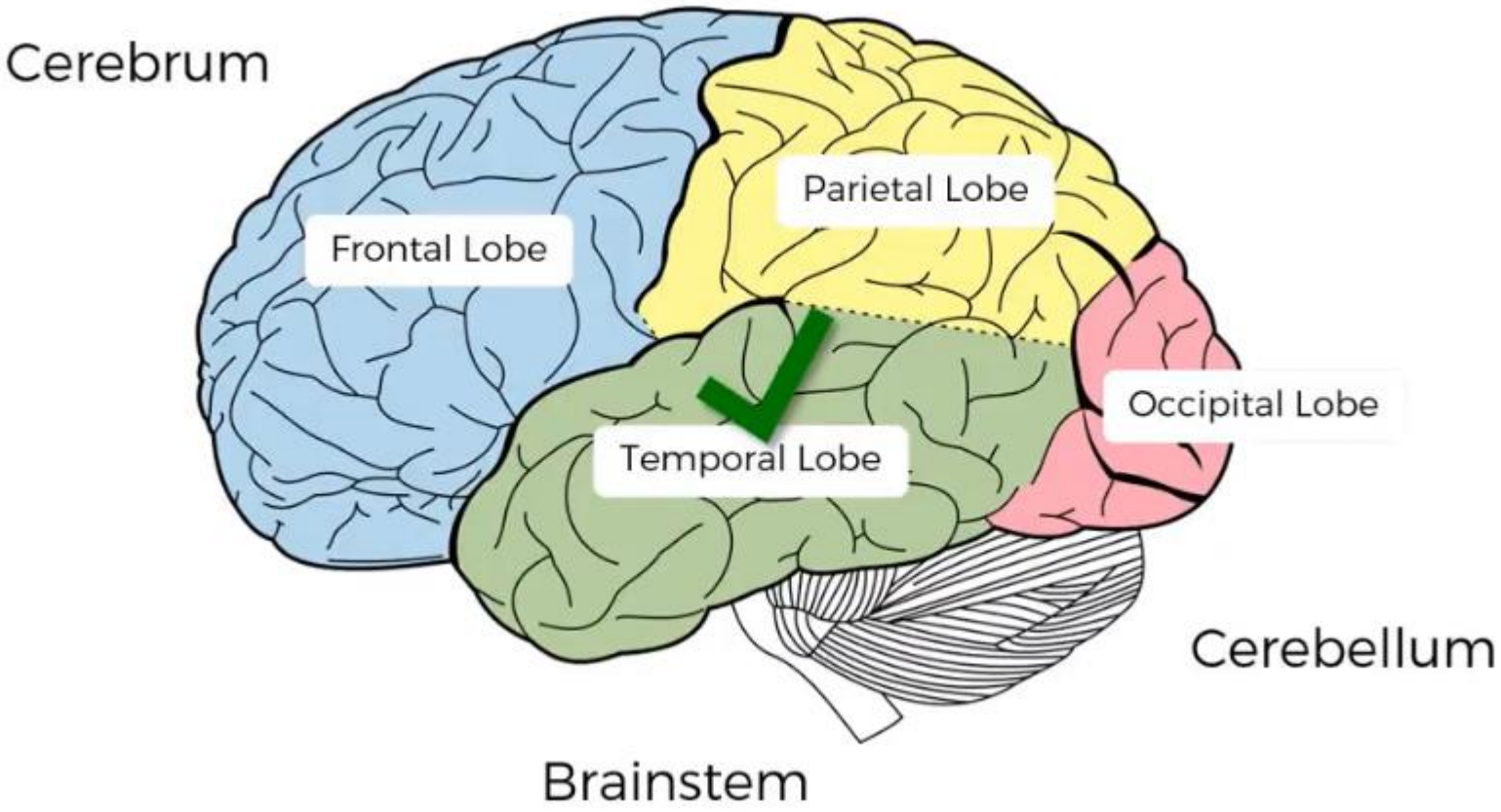
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

產生預測結果

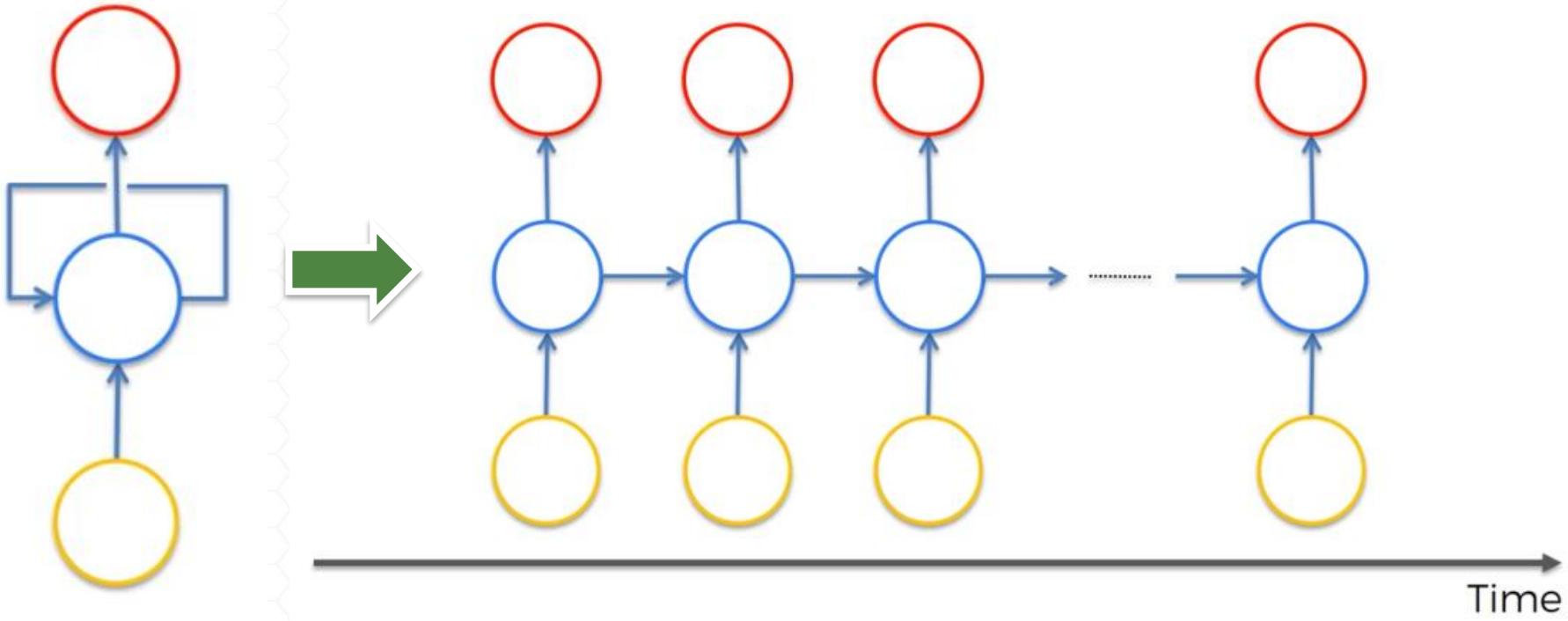
```
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = 8000,
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = 2000)
```

循環神經網路
(Recurrent Neural Networks)

腦的運作方式


Cerebrum
Frontal Lobe
Parietal Lobe
Occipital Lobe
Temporal Lobe
Cerebellum
Brainstem

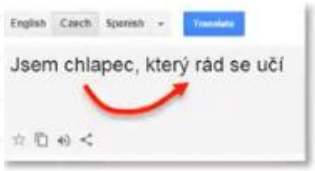
循環神經網路 (Recurrent Neural Networks)

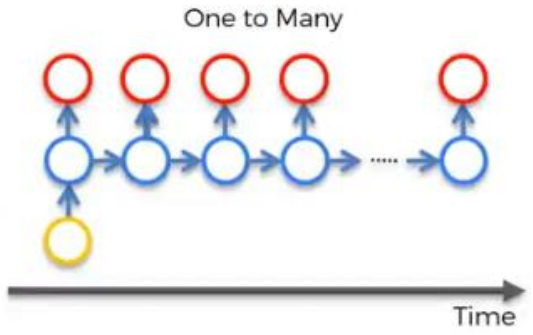


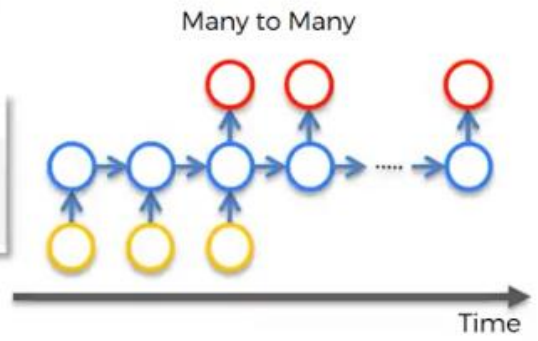
循環神經網路的應用

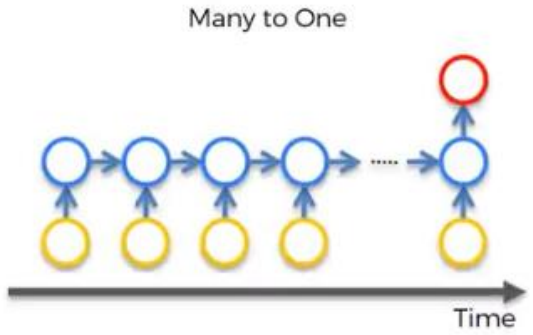
"black and white dog jumps over bar."

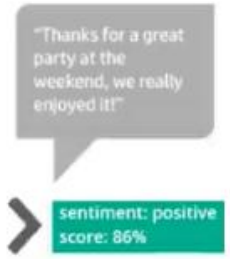
karpathy.github.io



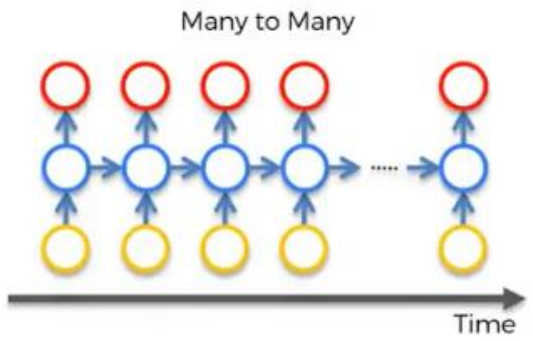


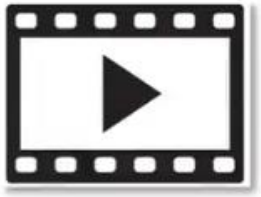




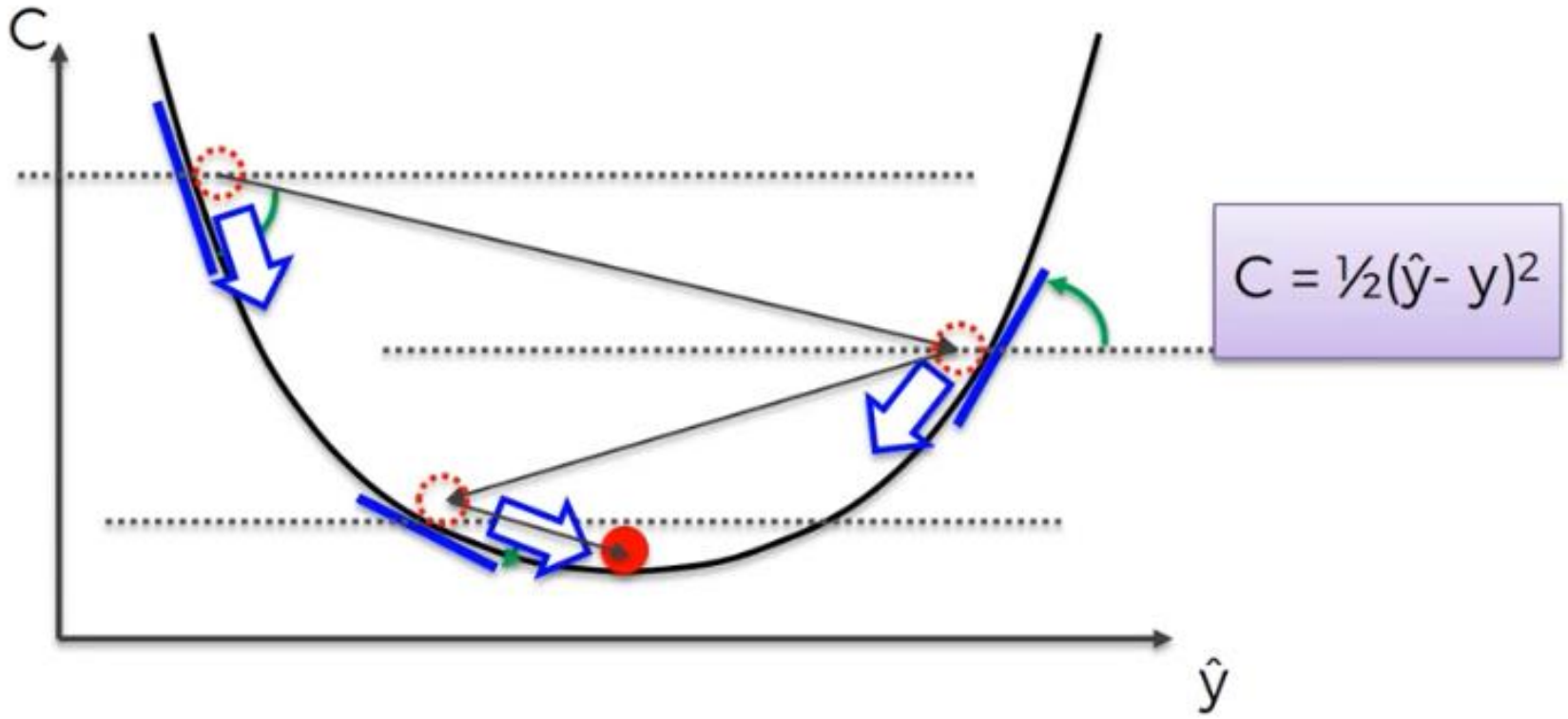


dev.havenondemand.com

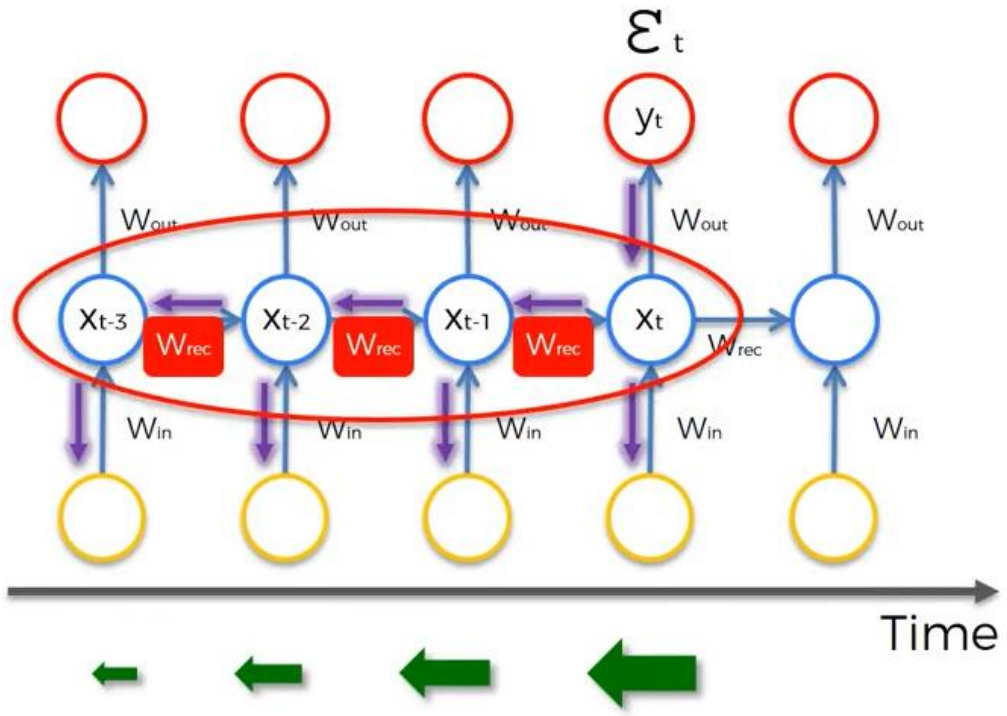




梯度下降問題



梯度消失問題



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

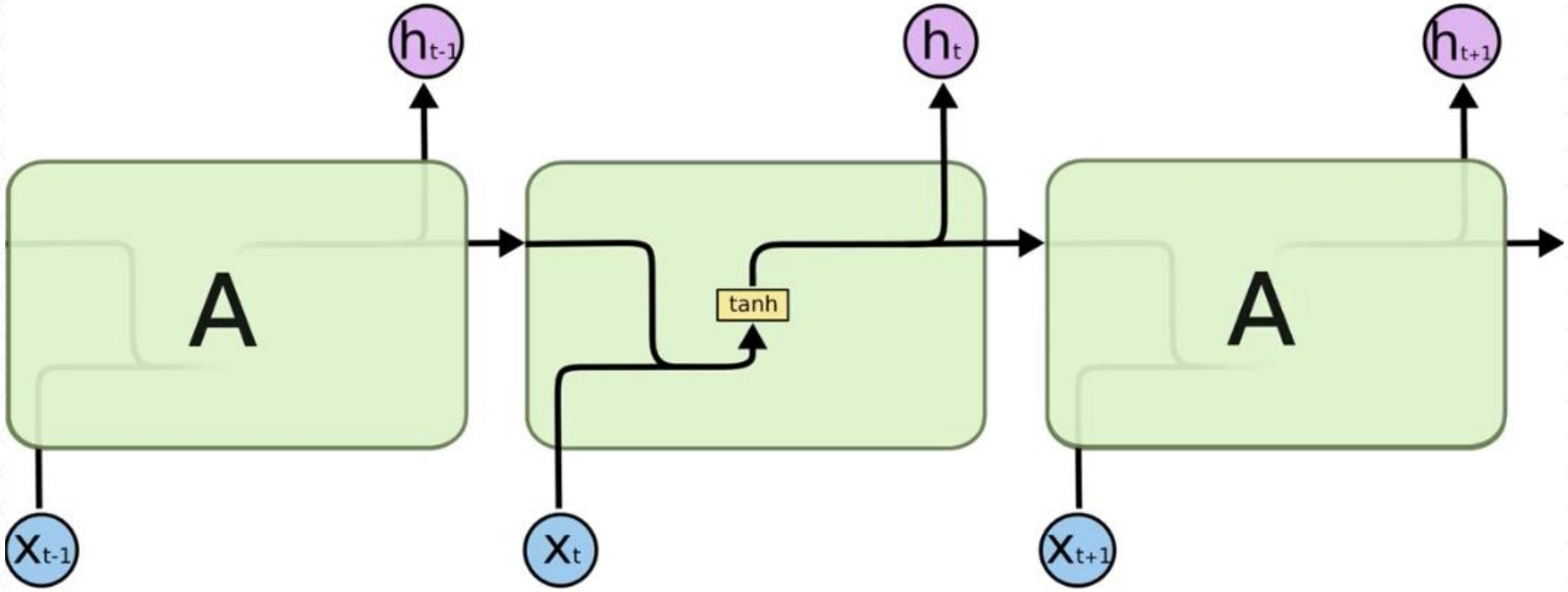
$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

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

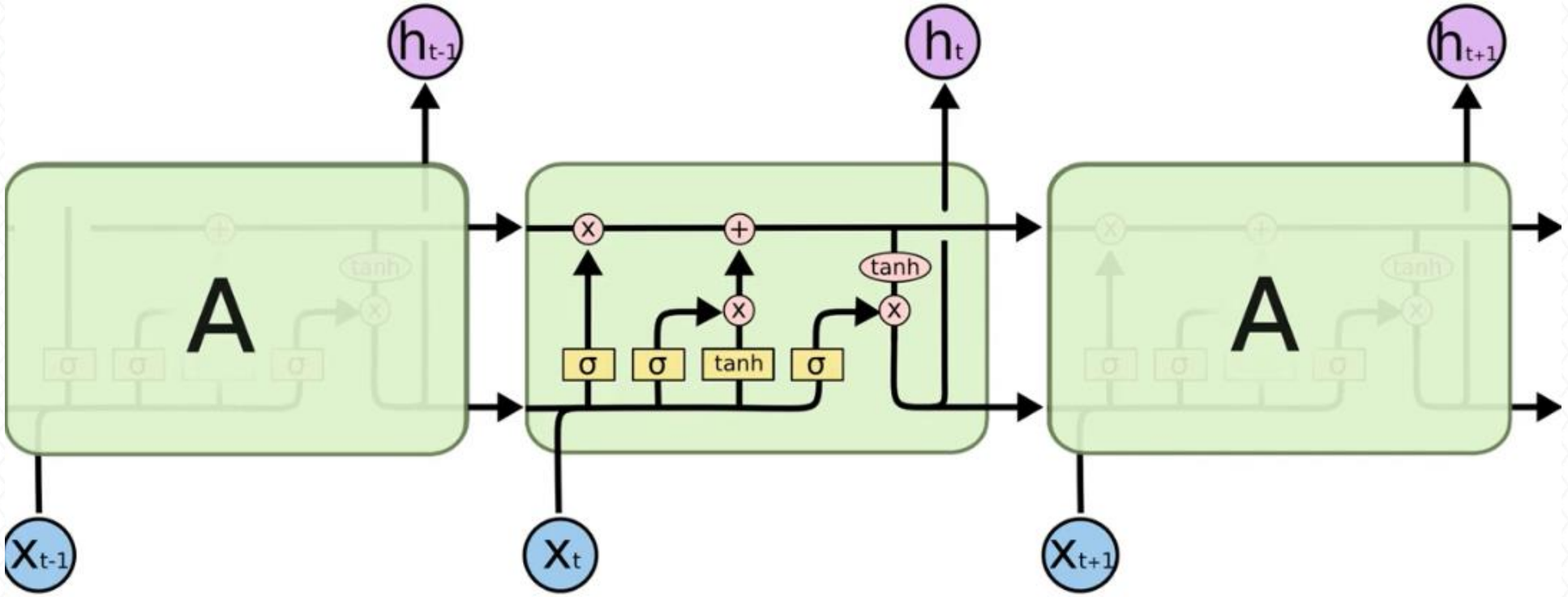
W_{rec} ~ small ⇨ Vanishing

W_{rec} ~ large ⇨ Exploding

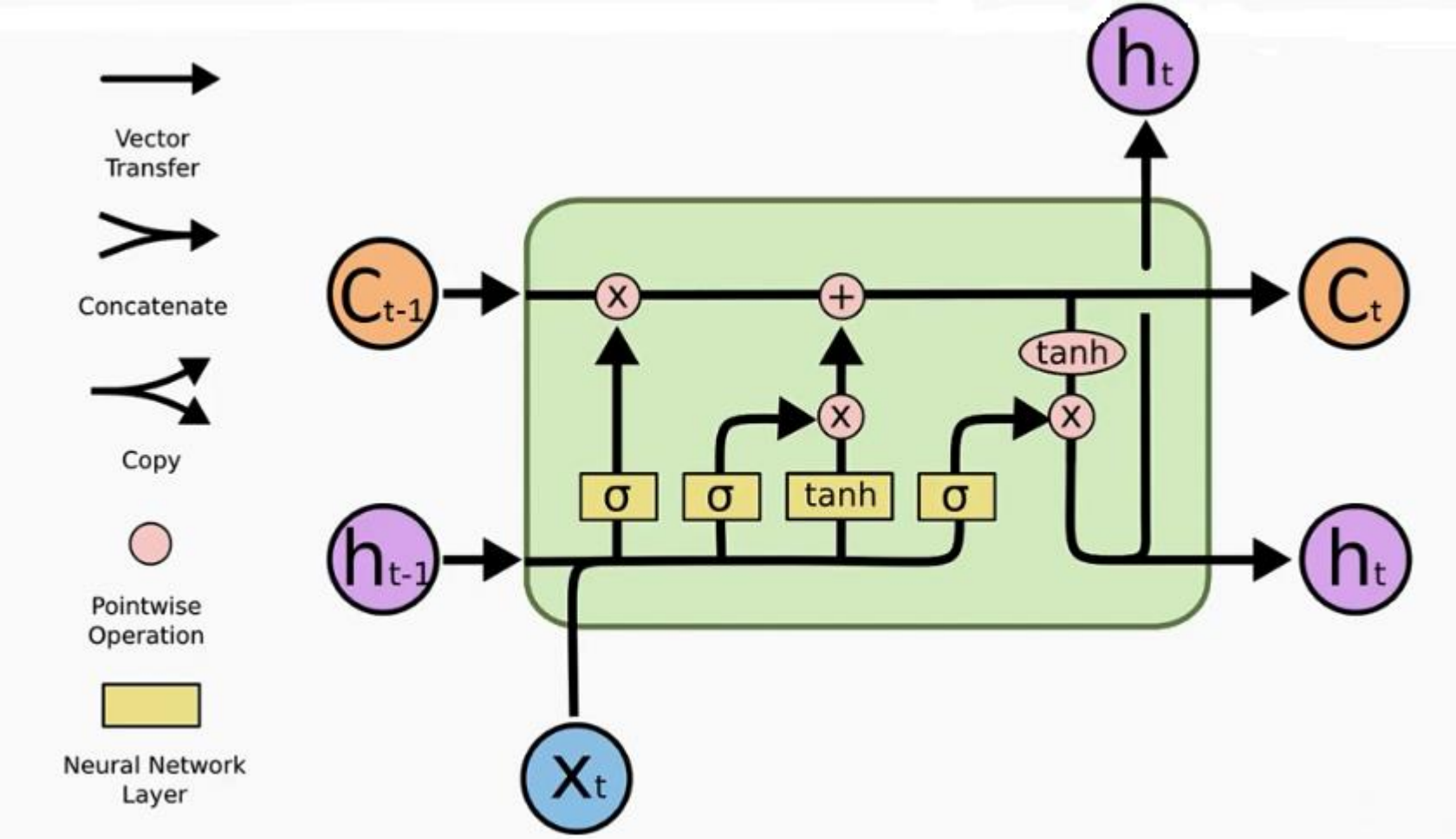
Long Short-Term Memory



Long Short-Term Memory

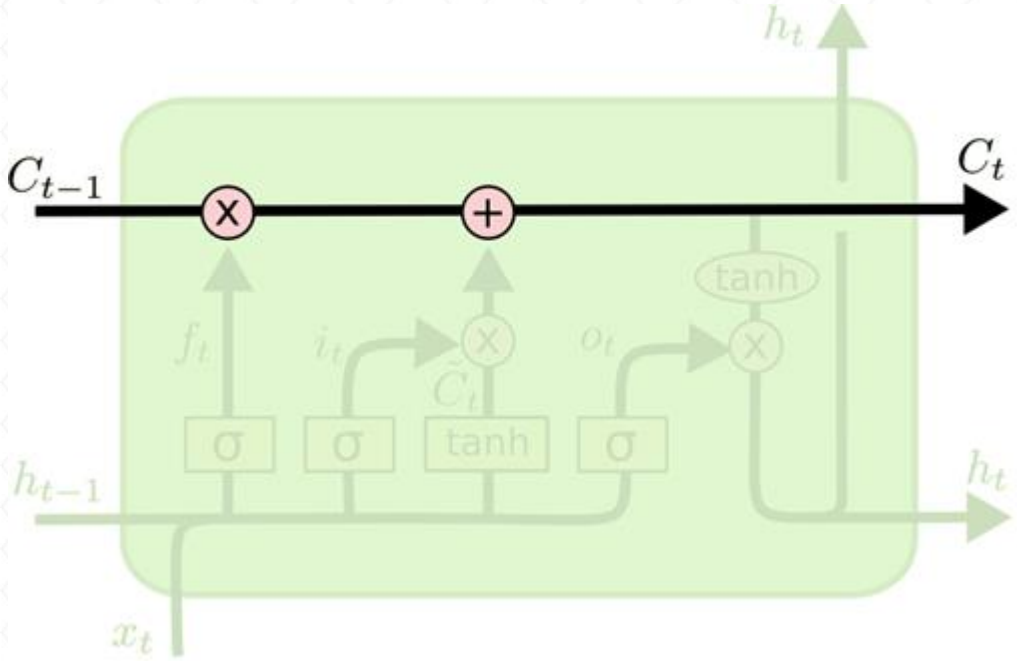


Long Short-Term Memory

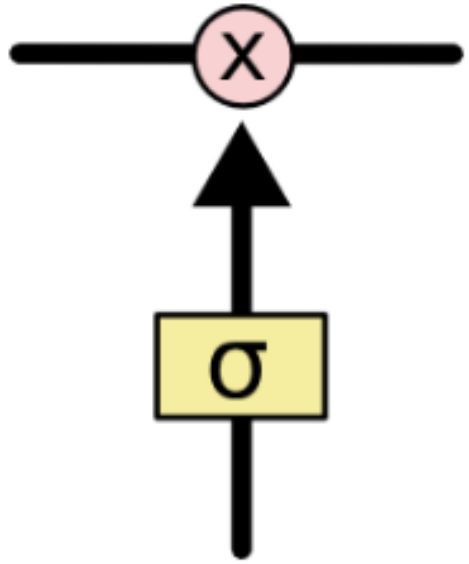


Long Short-Term Memory 詳解

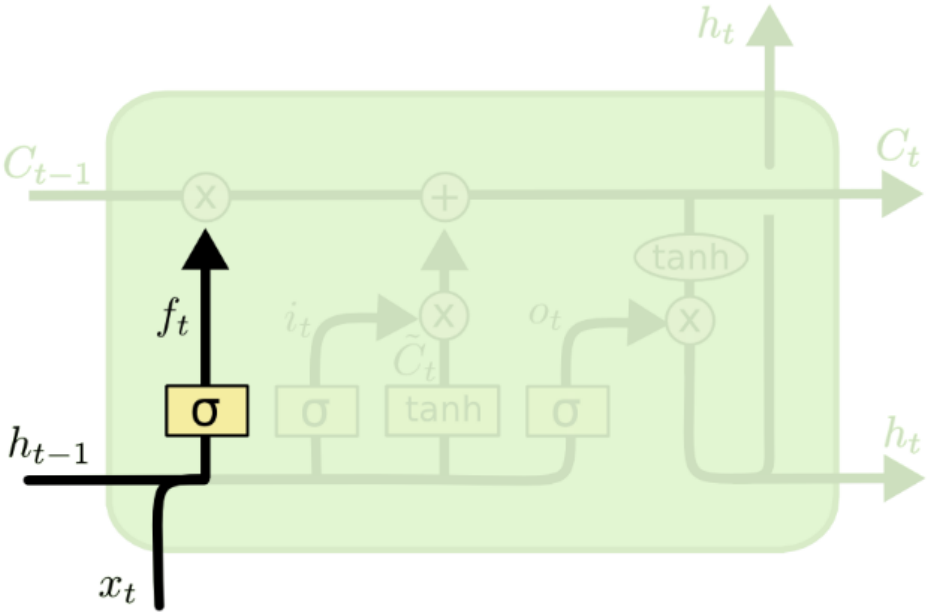
RNN 傳遞過程



Sigmoid 閘門

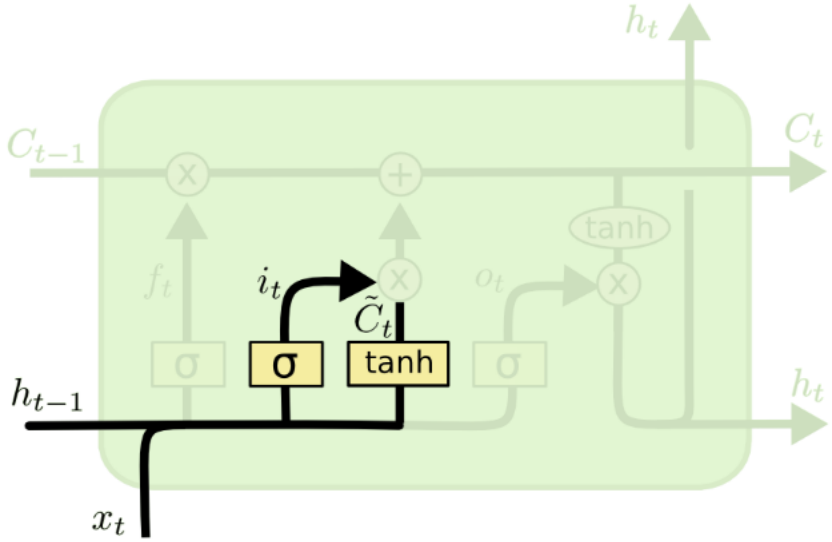


遺忘門 (Forget Gate)



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

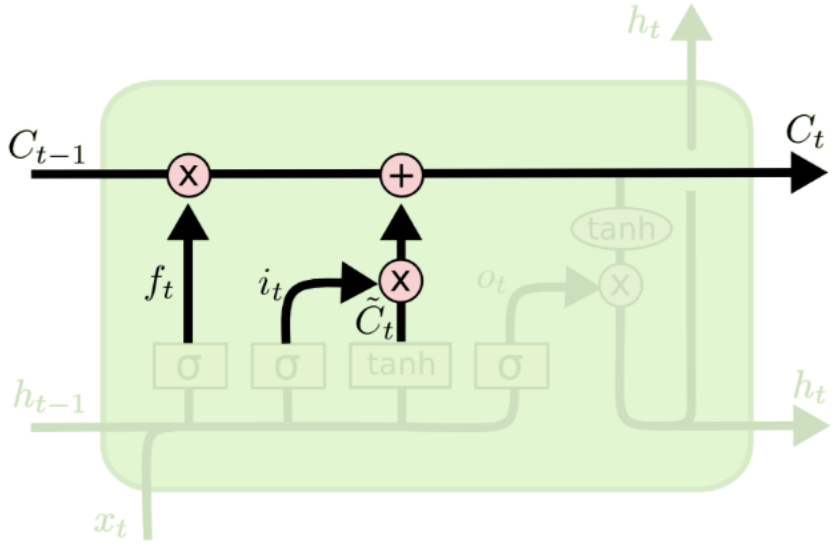
輸入門 (Input Gate)



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

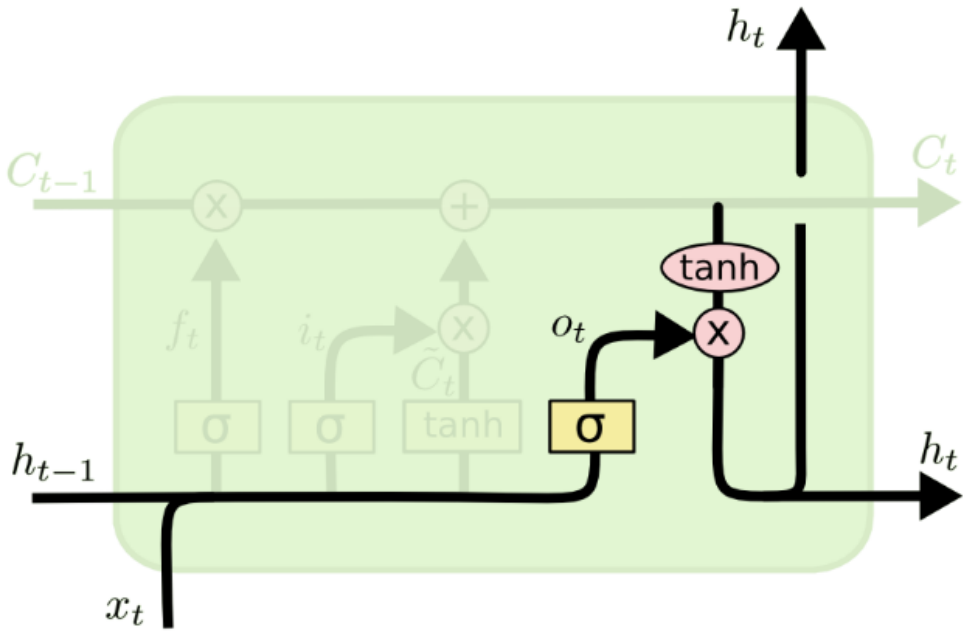
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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)



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

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

股價預測


GOOG (Google, Inc.) Nasdaq GS
29-Oct-2007 11:52am
© StockCharts.com
Open 677.77 High 679.51 Low 672.09 Last 676.19 Volume 1.6M Chg +1.59 (+0.24%)
GOOG (Daily) 676.19
MA(50) 571.62
MA(20) 631.33
Price Gap 1
Price Gap 2
Price Gap 3
680
660
640
620
600
580
560
540
520
500
480
Jun 11 18 25 Jul 9 16 23 Aug 6 13 20 27 Sep 10 17 24 Oct 8 15 22 29
Volume 1.57M, EMA(60) 5.44M
15M
10M
5M
Jun 11 18 25 Jul 9 16 23 Aug 6 13 20 27 Sep 10 17 24 Oct 8 15 22 29

讀取訓練資料

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the training set
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
training_set = dataset_train.iloc[:, 1:2].values
```

特徵規一化

```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

資料轉換

```
# Creating a data structure with 60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, 1258):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

建立RNN

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Initialising the RNN
regressor = Sequential()
```

建立 RNN

```
# Adding the first LSTM layer and some Dropout regularization
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))
# Adding a second LSTM layer and some Dropout regularization
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
# Adding a third LSTM layer and some Dropout regularization
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
# Adding a fourth LSTM layer and some Dropout regularization
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))
# Adding the output layer
regressor.add(Dense(units = 1))
```

編譯RNN

```
# Compiling the RNN
regressor.compile(optimizer = 'adam', loss =
'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

讀取測試資料集

```
# Getting the real stock price of 2017
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

產生預測價格

```
# Getting the predicted stock price of 2017
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

視覺化呈現預測結果

```
# Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google
Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```

THANK YOU