

TensorFlow與Keras

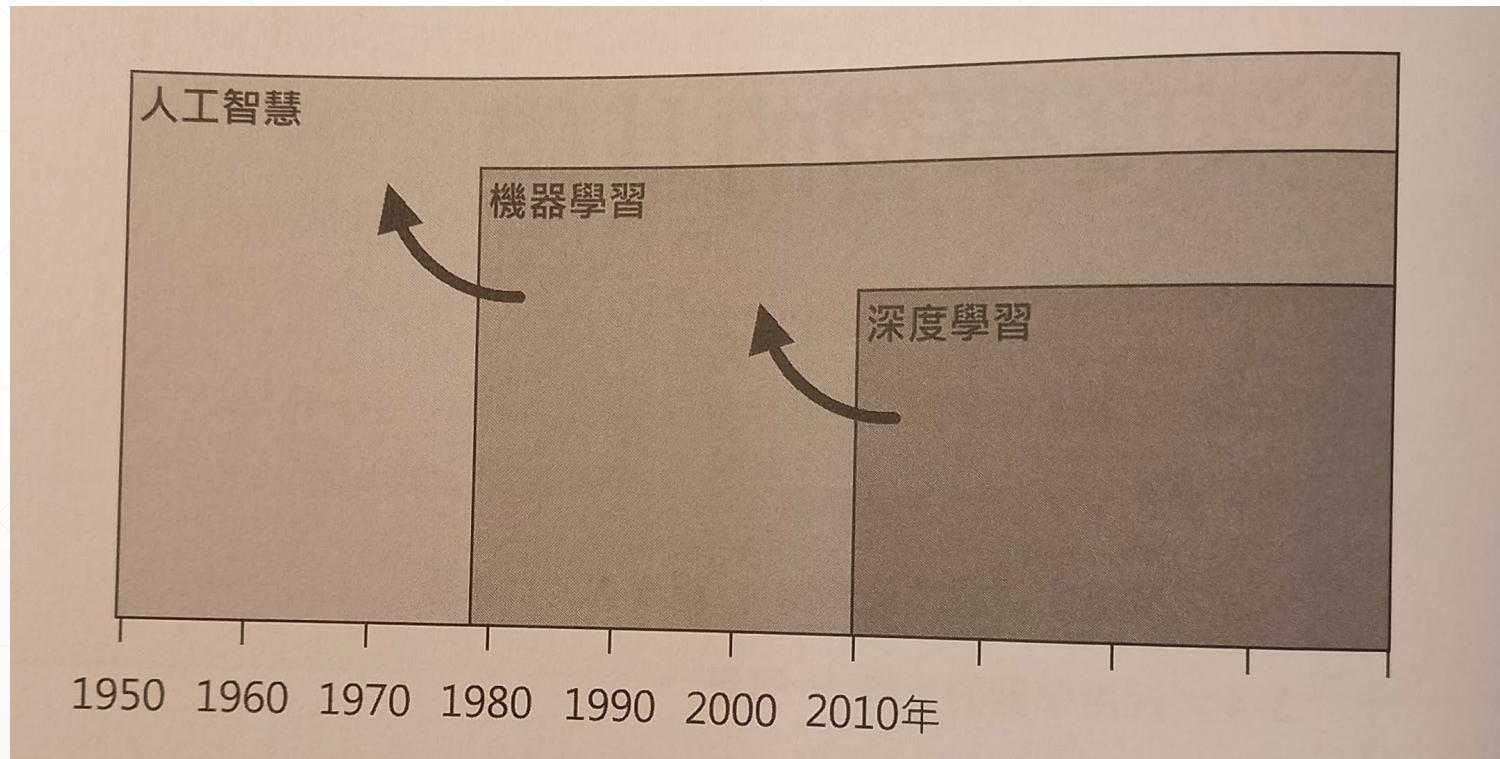
第三章 深度學習的基礎

目錄

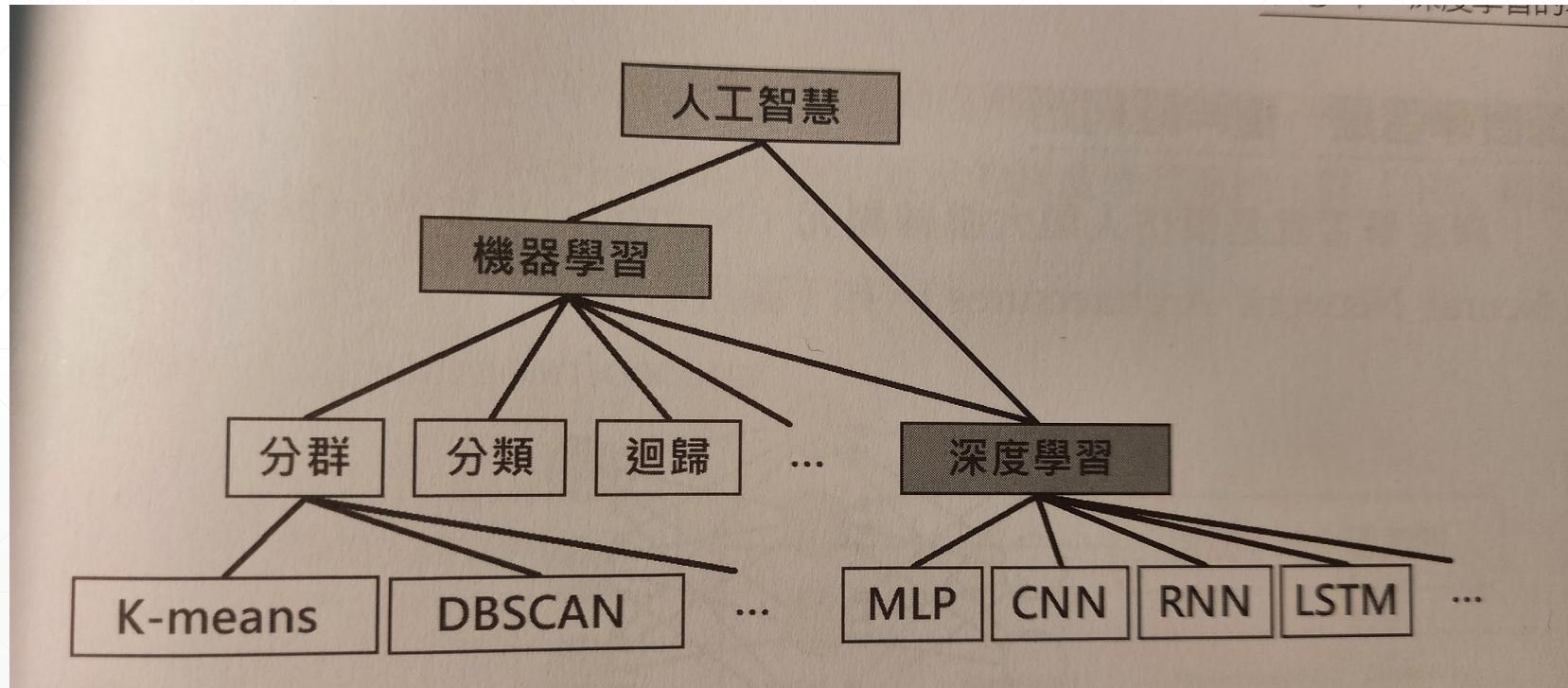
- 3-1 認識深度學習
 - 3-2 深度學習的基礎知識
 - 3-3 深度學習的精神網路 – 建構你的計算圖
 - 3-4 深度學習的資料 - 張量
-

3-1 認識深度學習

3-1-1 人工智慧、機器學習與深度學習的關係

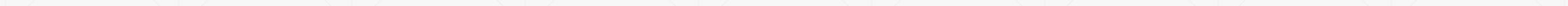


3-1-1 人工智慧、機器學習與深度學習的關係



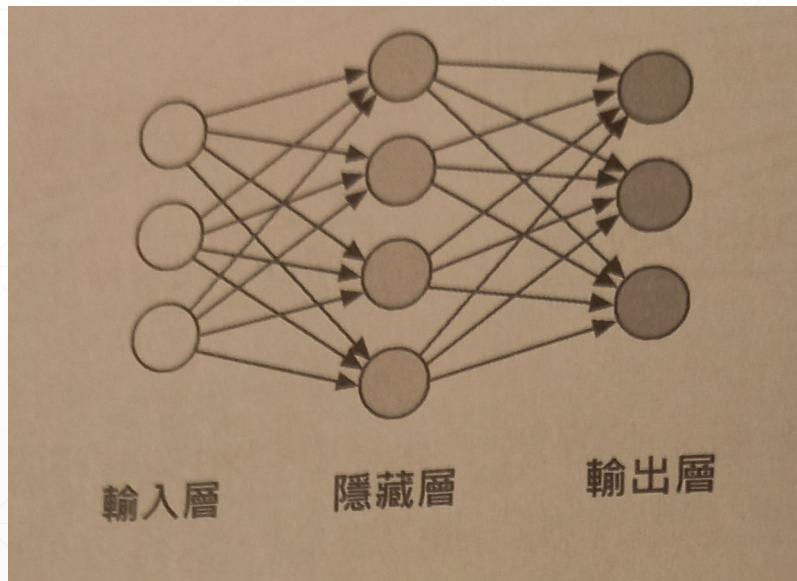
3-1-2 什麼是深度學習？

- 定義：「一種實現機器學習的技術」 = 「我吃過的鹽比你吃過的米還多」
- 需要大量的資料

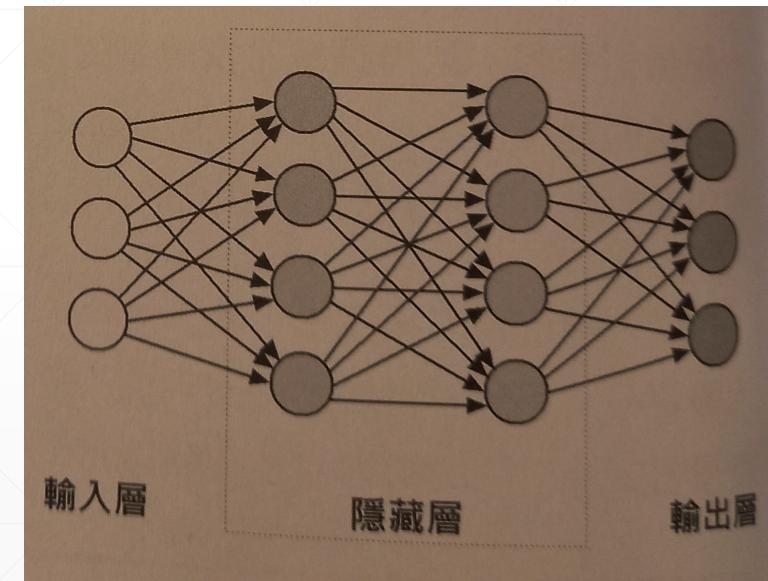


3-1-2 什麼是深度學習？

- 深度學習就是模仿人類大腦「神經元」傳輸的一種「神經網路架構」



多神經網路



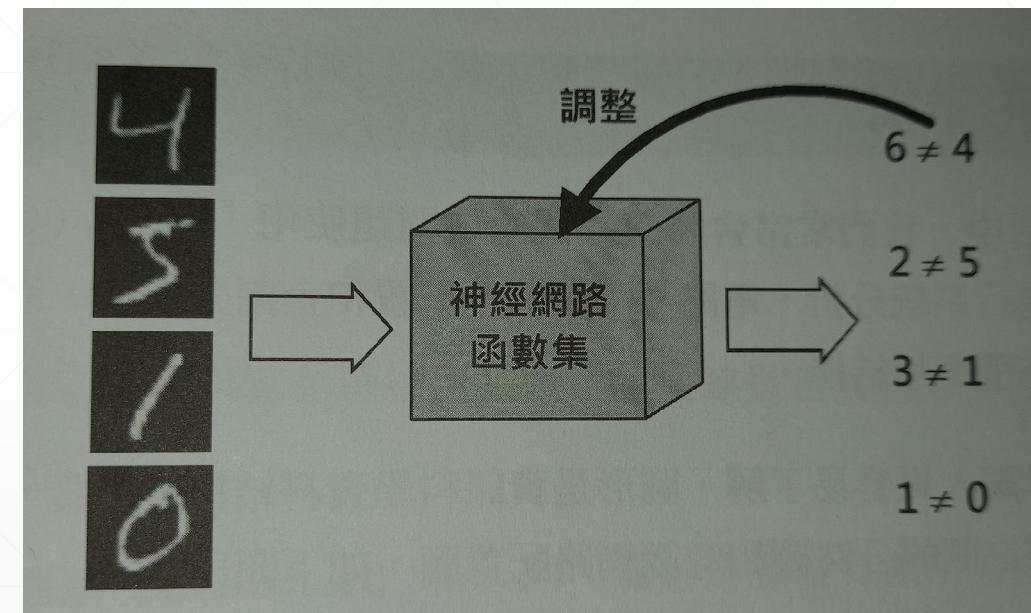
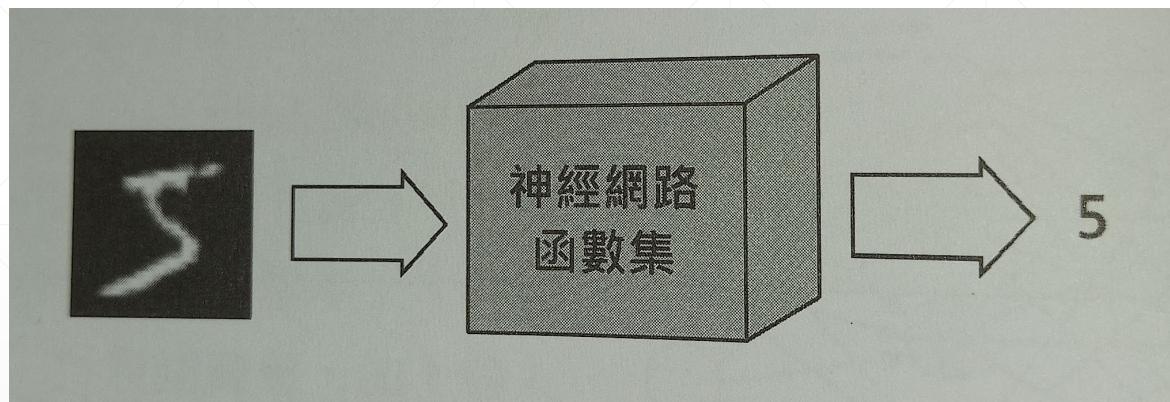
深度神經網路

3-1-2 什麼是深度學習？

- Q：深度學習能做什麼？
- A：處理所有「感知問題」
- Ex. (1) 模仿人類的圖片分類、語音識別、手寫辨識和自動駕駛
 - (2) 大幅改進機器翻譯和文字轉語音的正確率
 - (3) 大幅改進樹會助理、搜尋引擎和網頁廣告投放的效果
 - (4) 自然語言對話的問答系統，例如：聊天機器人

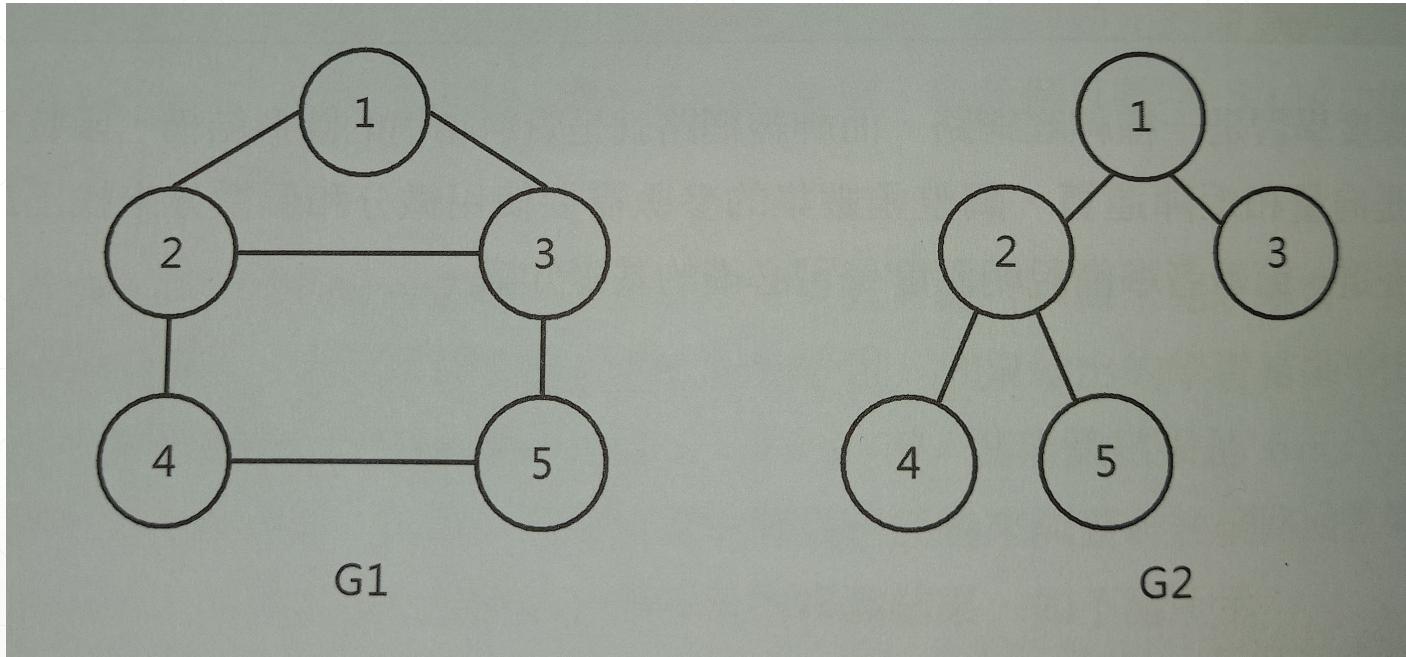


3-1-3 深度學習就是一個函數集



3-2 深度學習的基礎知識

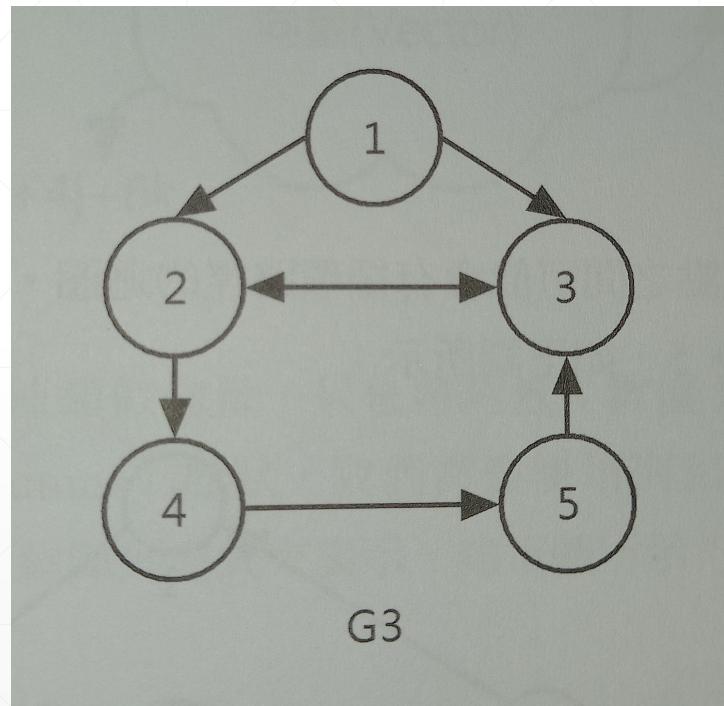
3-2-1 圖形結構



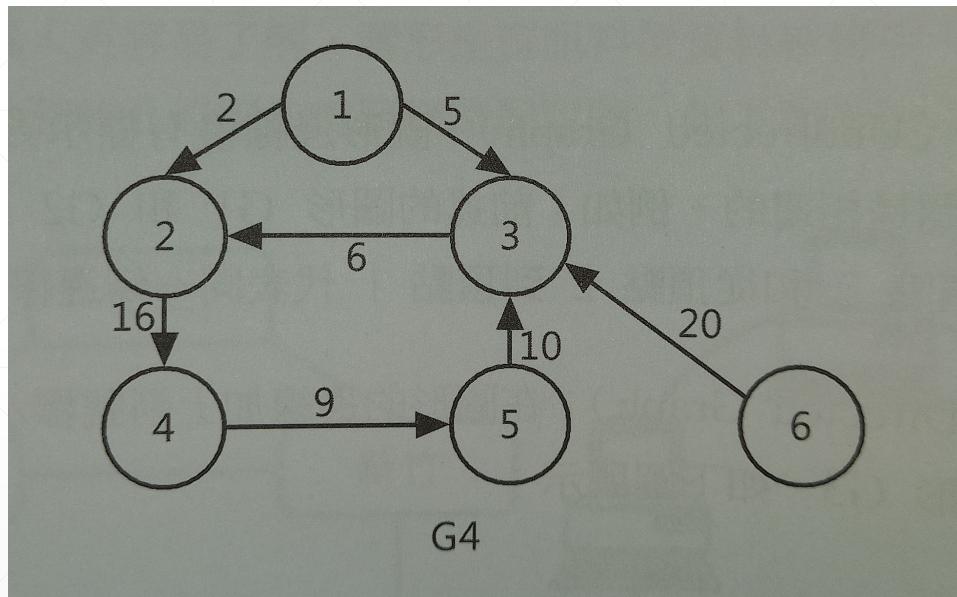
- 圖形基本上是有限「頂點」和「邊線」集合所組成

3-2-1 圖形結構

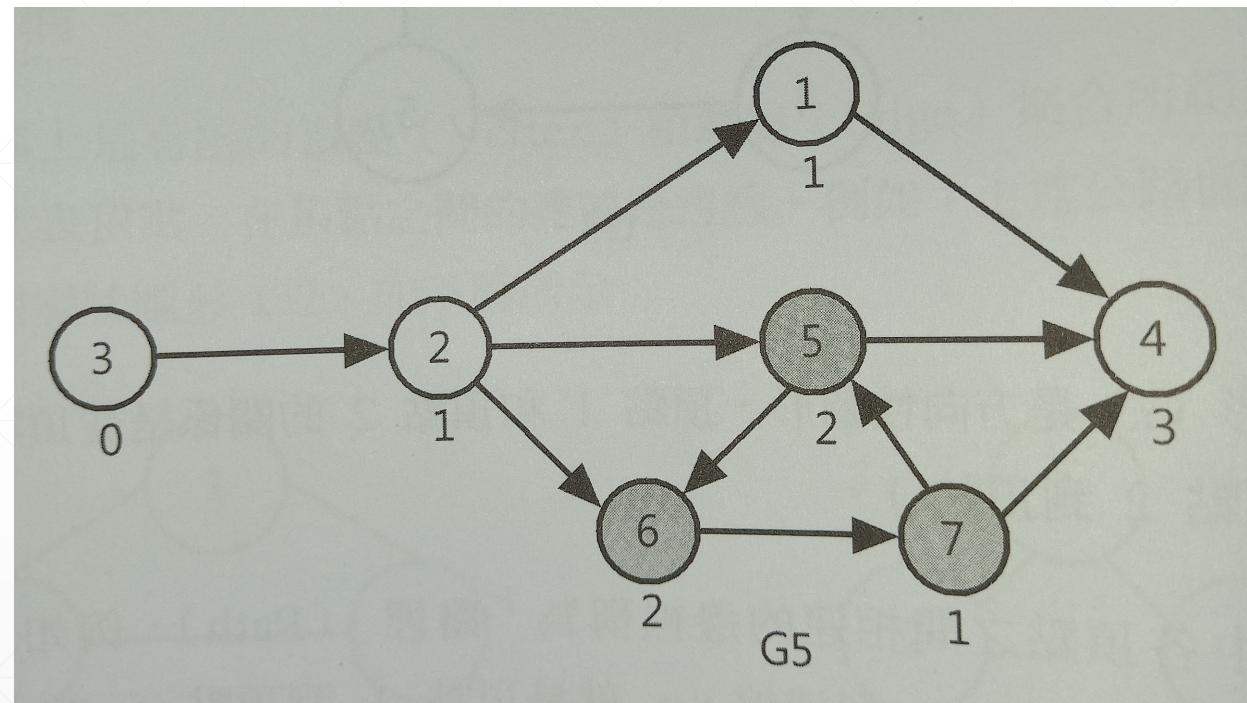
- 圖形種類：
 - a.無方向性圖形
 - b.方向性圖形



3-2-1 圖形結構



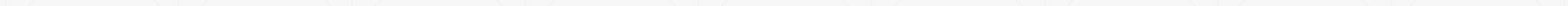
加權圖形



方向性循環圖

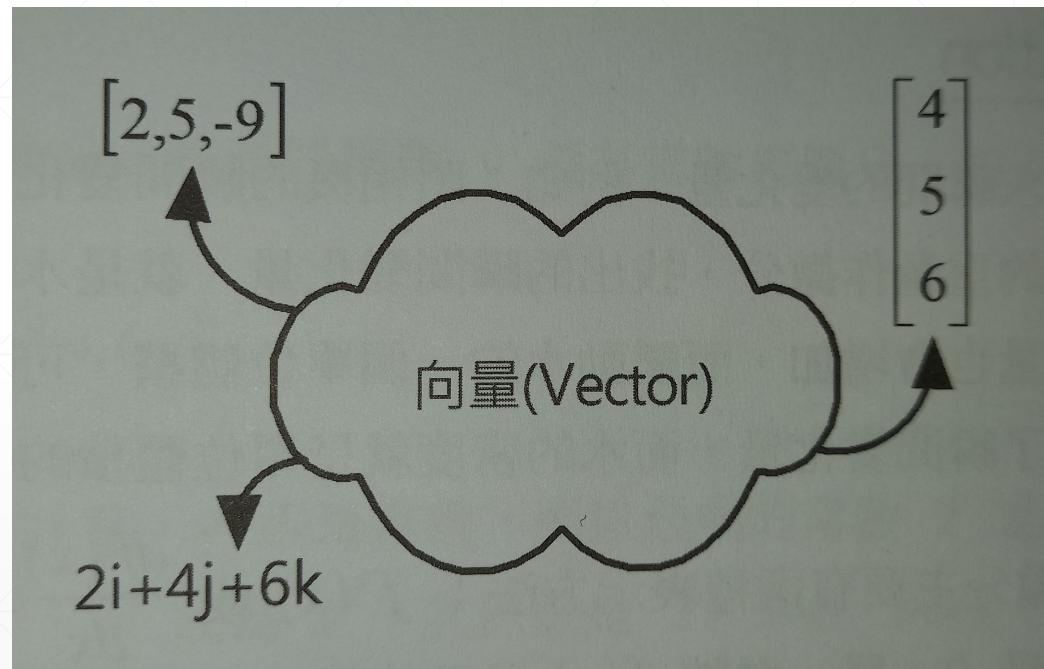
3-2-2 向量與矩陣

- 深度學習從輸入層送入的資料，和輸出層輸出的資料都稱為「張量」，數學上的向量與矩陣就是一種張量。



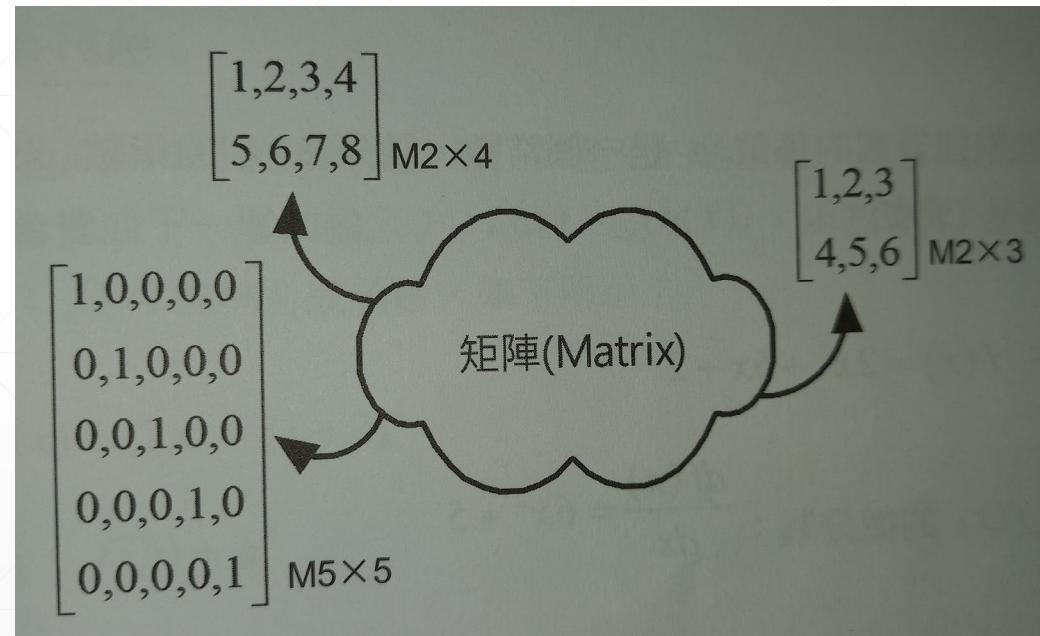
3-2-2 向量與矩陣

- **向量**：具有大小質與方向性的數學表現，一般在程式語言皆以一維陣列表示。



3-2-2 向量與矩陣

- **矩陣**：類似於向量，是將純量排列成二維表格的「列」和「欄」形狀，在程式語言皆以二維陣列表示。



3-2-3 微分與偏微分

$$f(x) = ax^n$$

$$\frac{df(x)}{dx} = anx^{n-1}$$

假設： $f(x) = 2x^3 + 5x + 2$

那麼 $f(x)$ 的微分為： $\frac{df(x)}{dx} = 6x^2 + 5$

■ 微分

$$f(x, y) = 2x^3 + 6xy^2 + 4y + 2$$

對變數 x 偏微分，就是將變數 y 視為常數來微分，如下所示：

$$\frac{\partial f(x, y)}{\partial x} = 6x^2 + 6y^2$$

■ 偏微分

3-2-3 微分與偏微分

假設： $f(x) = (2x^3 + 5x + 2)^4$

函數 $g(x)$ ： $f(h) = h^4$

函數 $h(x)$ ： $h(x) = 2x^3 + 5x + 2$

■ 連鎖率

合成函數 $f(x) = g(h(x))$ 的微分需要使用連鎖律，如下所示：

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(h)}{\partial h} \frac{\partial h(x)}{\partial x}$$

上述連鎖律基本上是從外向內一層一層的進行微分，首先是最外層函數的微分，然後是內層函數的微分，如下所示：

函數 $g(x)$ 微分： $\frac{\partial f(h)}{\partial h} = 4h^3$

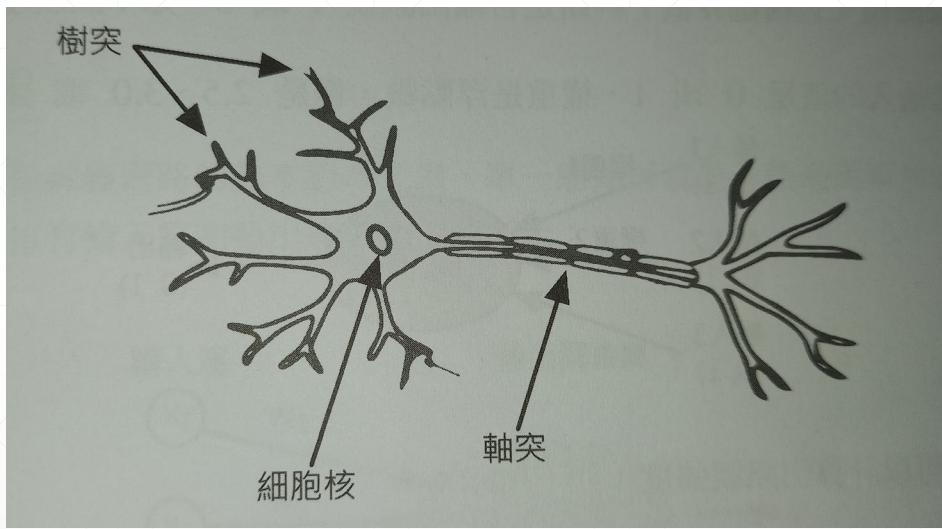
函數 $h(x)$ 微分： $\frac{\partial h(x)}{\partial x} = 6x^2 + 5$

最後，我們可以使用連鎖律執行 $f(x)$ 函數的微分，如下所示：

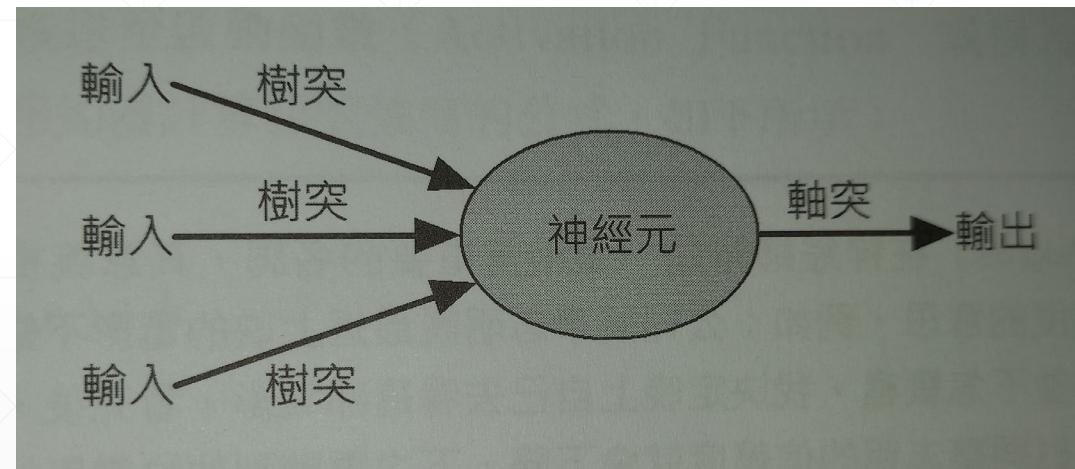
$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(h)}{\partial h} \frac{\partial h(x)}{\partial x} = (4h^3)(6x^2 + 5) = 4(2x^3 + 5x + 2)^3(6x^2 + 5)$$

3-3 深度學習的精神網路 – 建構你的計算圖

3-3-1 神經元



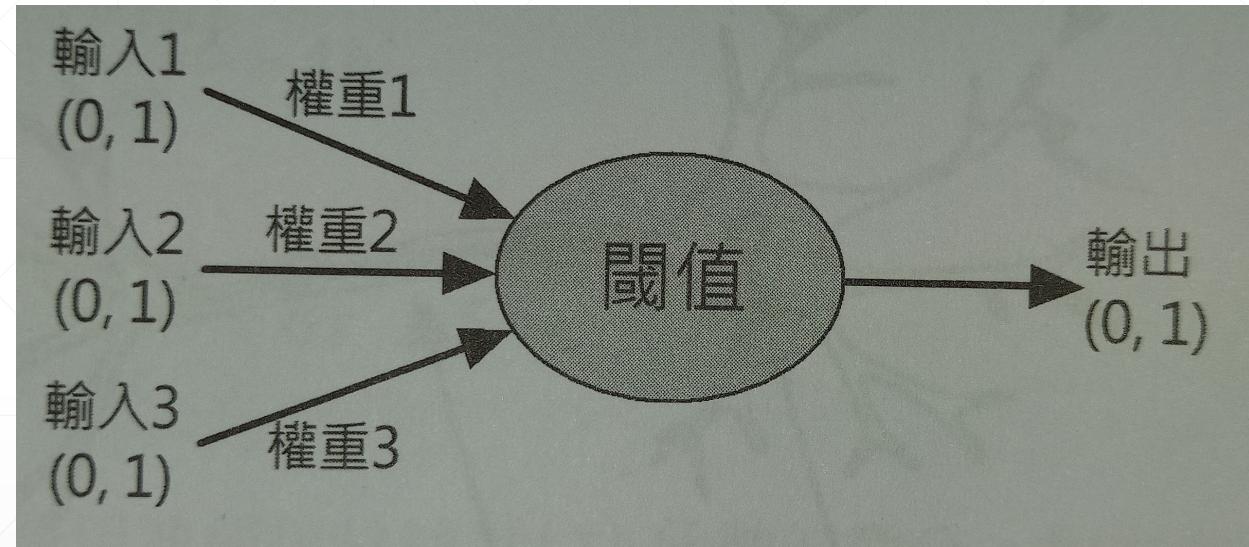
腦神經細胞的神經元



深度學習的神經網路

3-3-1 神經元

- 以0或1的數值代表訊號。
- 輸出是以訊號強度是否大於等於「閾值」(臨界值)，以決定是否繳活神經元。

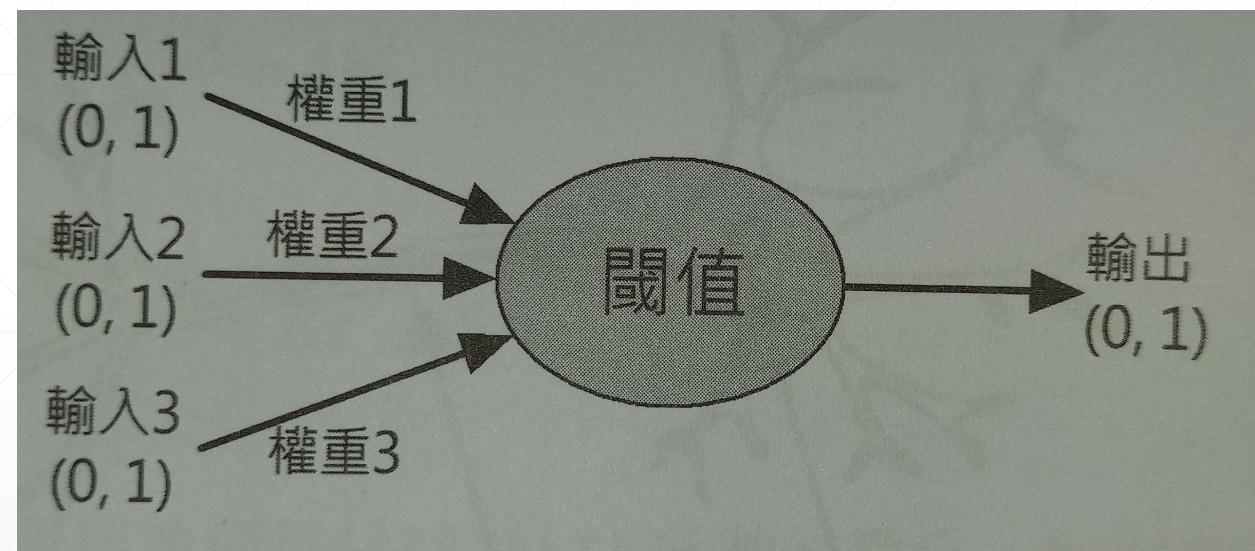


人工神經網路

$$\text{訊號強度} = \text{輸入 } 1 * \text{權重 } 1 + \text{輸入 } 2 * \text{權重 } 2 + \text{輸入 } 3 * \text{權重 } 3$$

3-3-1 神經元

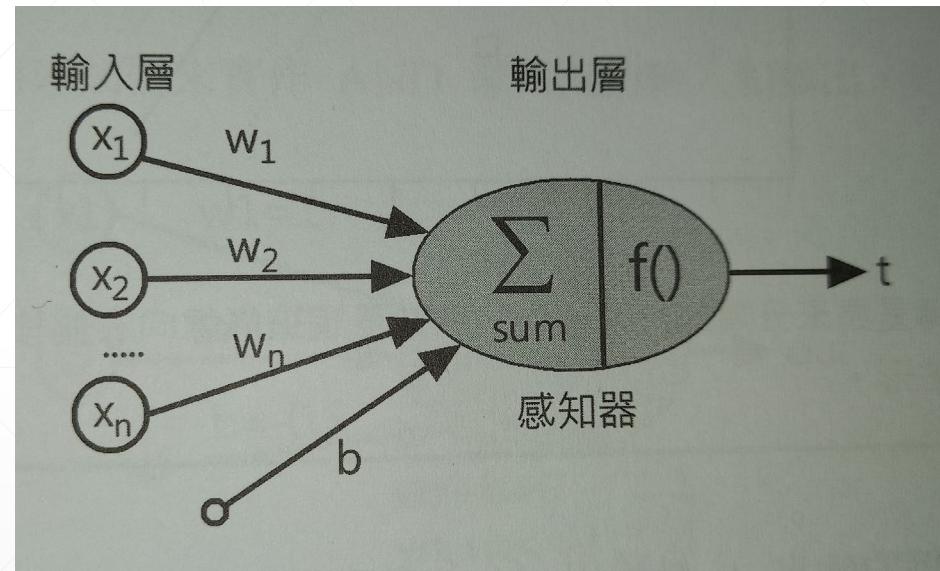
- 訊號強度 \geq 閾值 \rightarrow 輸出 1
訊號強度 $<$ 閾值 \rightarrow 輸出 0
- 權重相當於「信賴度」。



人工神經網路

3-3-2 感知器

- 感知器：一個可以模擬人類感知能力的機器，是神經網路的基本組成元素。



3-3-2 感知器

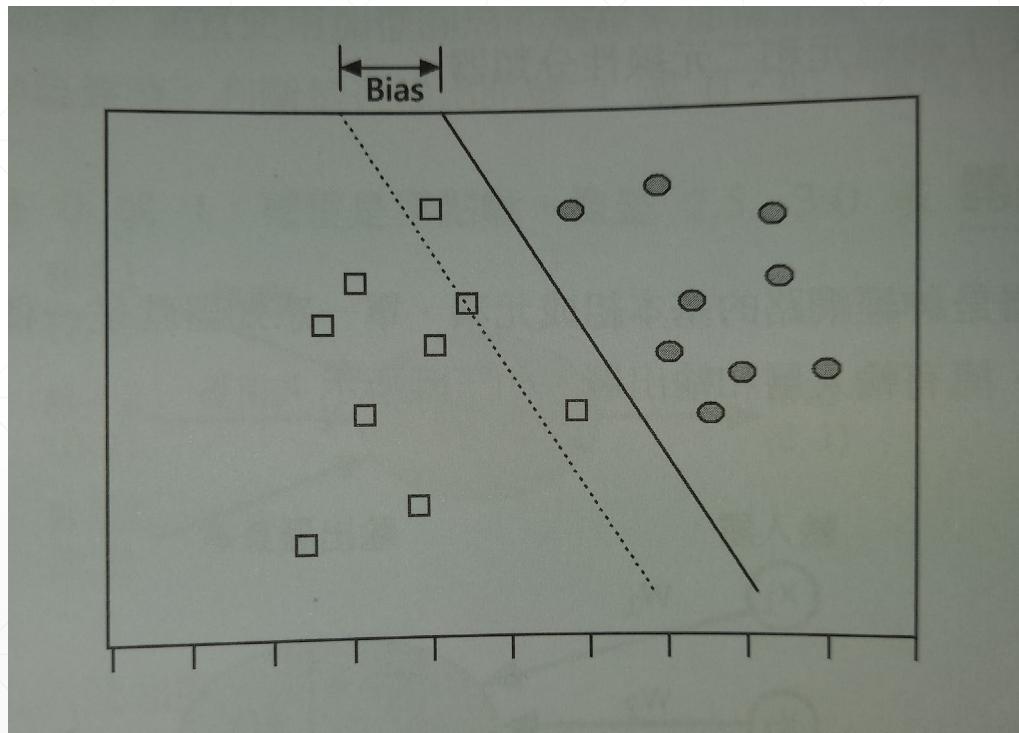
- w : 感知器輸入向量
- X : 權重向量
- b : 偏向量(偏量)

$$z = \left(\sum_{i=1}^n w_i x_i \right) + b$$

感知器計算強度公式

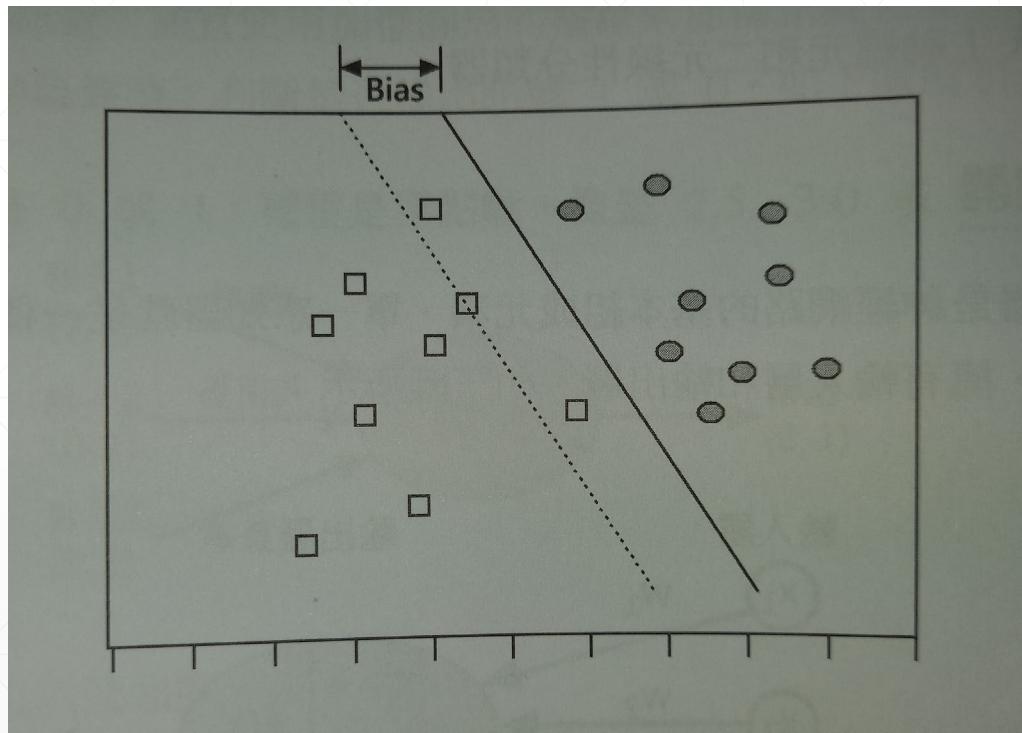
3-3-2 感知器

- 偏向量可以位移「決策邊界」(圖上虛線)來得到正確的分類結果。



3-3-2 感知器

- 偏向量 b 可以位移「決策邊界」(圖上虛線)來得到正確的分類結果。



3-3-2 感知器

- 感知層的輸出 t 是使用 $f(n)$ 啟動函數來判斷是否繳活神經元。
- s = 闕值，初始值為 0，但可根據情況自行定義門檻。

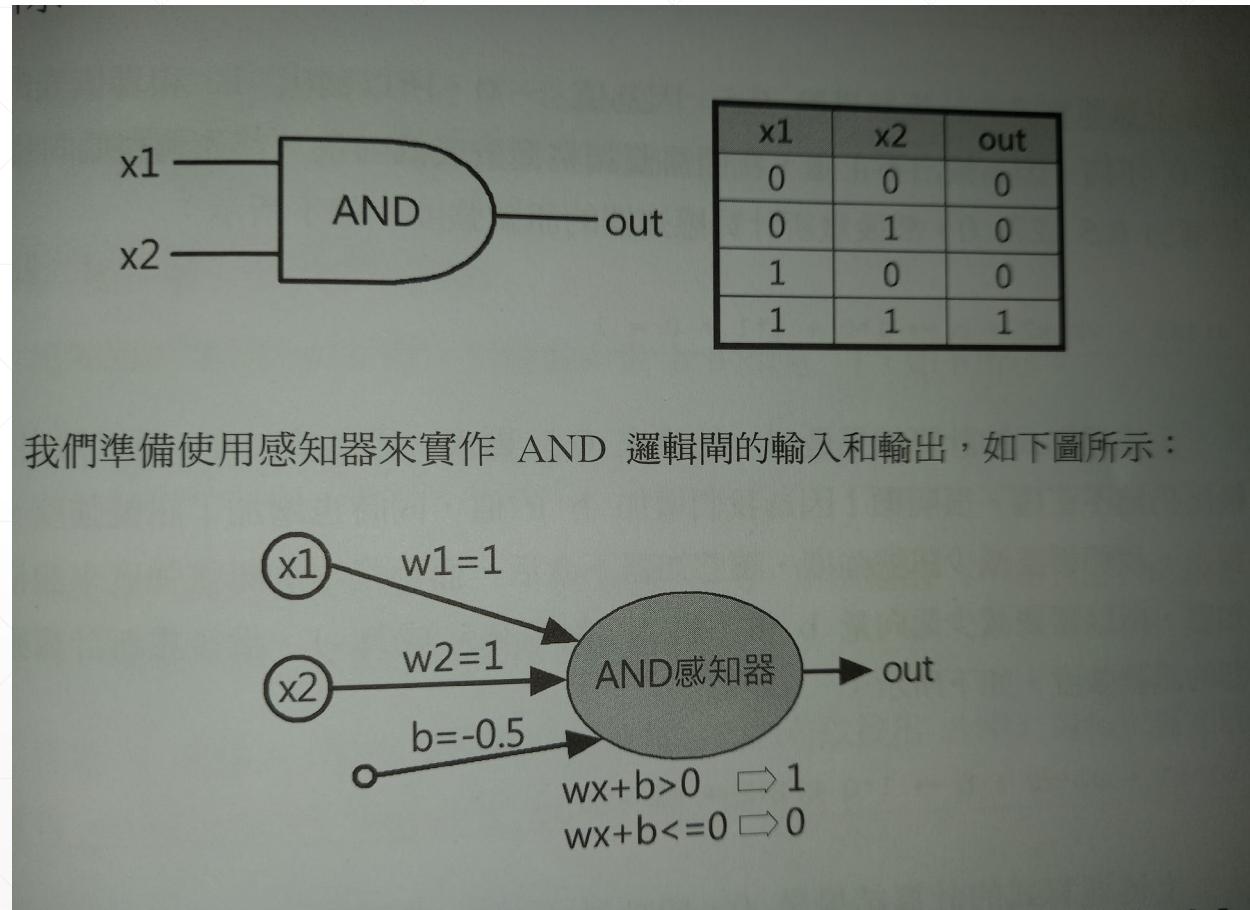
$$t = f(z) = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

上述啟動函數在傳統的感知器是使用階梯函數 (Step Function)，其定義如右所示：

$$f(x) \begin{cases} 1 & \text{如果 } wx + b > s \\ 0 & \text{否則} \end{cases}$$

3-3-2 感知器

- 範例：AND邏輯閘



3-3-2 感知器

第一列輸入值： $x_1=0$ 和 $x_2=0$

請使用輸入值、權重和偏向量來計算感知器的訊號強度，如下所示：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 0 + 1 \cdot 0 + -0.5 = -0.5$$

x_1	x_2	out
0	0	0
0	1	0
1	0	0
1	1	1

- 因為 $-0.5 \leq 0$ ，所以輸出 0，符合真值表。

3-3-2 感知器

第二列輸入值： $x_1=0$ 和 $x_2=1$

感知器訊號強度的計算如下所示：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 0 + 1 \cdot 1 + -0.5 = 0.5$$

- 因為 $0.5 > 0$ ，所以輸出 1，不符合真值表。

將偏量(b)調整為 0 再次運算：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 0 + 1 \cdot 1 + 0 = 1$$

- 因為 $1 > 0$ ，所以輸出 1，不符合真值表。

x_1	x_2	out
0	0	0
0	1	0
1	0	0
1	1	1

3-3-2 感知器

將偏量(b)調整為 1 再次運算：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 0 + 1 \cdot 1 + -1 = 0$$

- 因為 $0 \leq 0$ ，所以輸出 0，符合真值表。

使用新的偏量(b)來計算第一個感知信號強度：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 0 + 1 \cdot 0 + -1 = -1$$

- 因為 $-1 \leq 0$ ，所以輸出 0，符合真值表。

x1	x2	out
0	0	0
0	1	0
1	0	0
1	1	1

3-3-2 感知器

第三列輸入值： $x_1=1$ 和 $x_2=0$

感知器訊號強度的計算，目前偏向量 b 的值是 -1，如下所示：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 1 + 1 \cdot 0 + -1 = 0$$

第3章 深

x_1	x_2	out
0	0	0
0	1	0
1	0	0
1	1	1

- 因為 $-1 \leq 0$ ，所以輸出 0，符合真值表。

3-3-2 感知器

第四列輸入值： $x_1=1$ 和 $x_2=1$

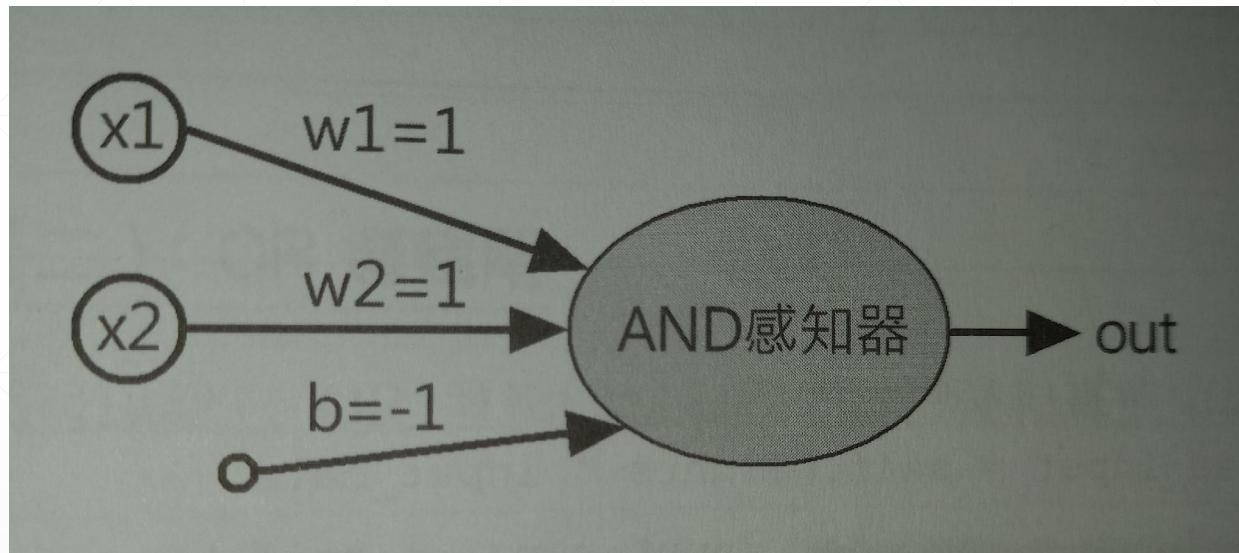
感知器訊號強度的計算，目前偏向量 b 的值是 -1，如下所示：

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b \rightarrow 1 \cdot 1 + 1 \cdot 1 + -1 = 1$$

x_1	x_2	out
0	0	0
0	1	0
1	0	0
1	1	1

- 因為 $1 > 0$ ，所以輸出 1，符合真值表。

3-3-2 感知器



依上述的訓練過程，我們可以找出AND邏輯閘感知器如圖。

3-3-2 感知器

- python範例：AND邏輯閘

執行結果為：[0 0] 0

[0 1] 0

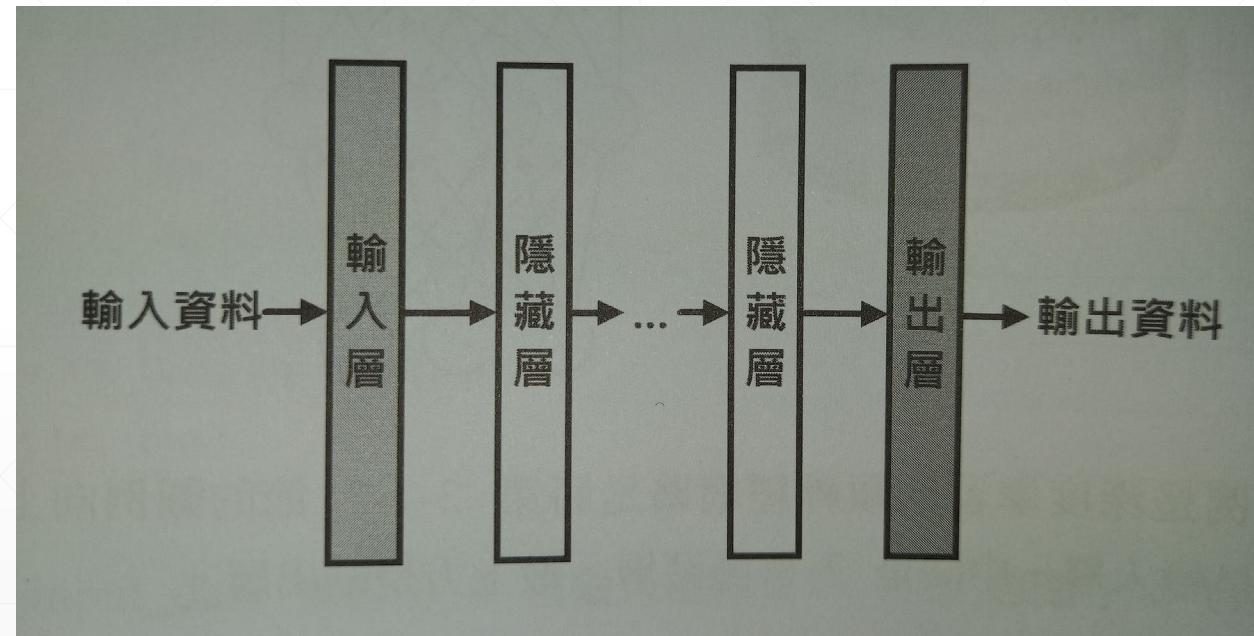
[1 0] 0

[1 1] 1

```
1 import numpy as np
2
3 class Perceptron:
4     def __init__(self, input_length, weights=None, bias=None):
5         if weights is None:
6             self.weights = np.ones(input_length) * 1
7         else:
8             self.weights = weights
9         if bias is None:
10            self.bias = -1
11        else:
12            self.bias = bias
13
14     @staticmethod
15     def activation_function(x):
16         if x > 0:
17             return 1
18         return 0
19
20     def __call__(self, input_data):
21         weighted_input = self.weights * input_data
22         weighted_sum = weighted_input.sum() + self.bias
23         return Perceptron.activation_function(weighted_sum)
24
25 weights = np.array([1, 1])
26 bias = -1
27 AND_Gate = Perceptron(2, weights, bias)
28
29 input_data = [np.array([0, 0]), np.array([0, 1]),
30               np.array([1, 0]), np.array([1, 1])]
31 for x in input_data:
32     out = AND_Gate(np.array(x))
33     print(x, out)
```

3-3-3 深度學習的神經網路種類

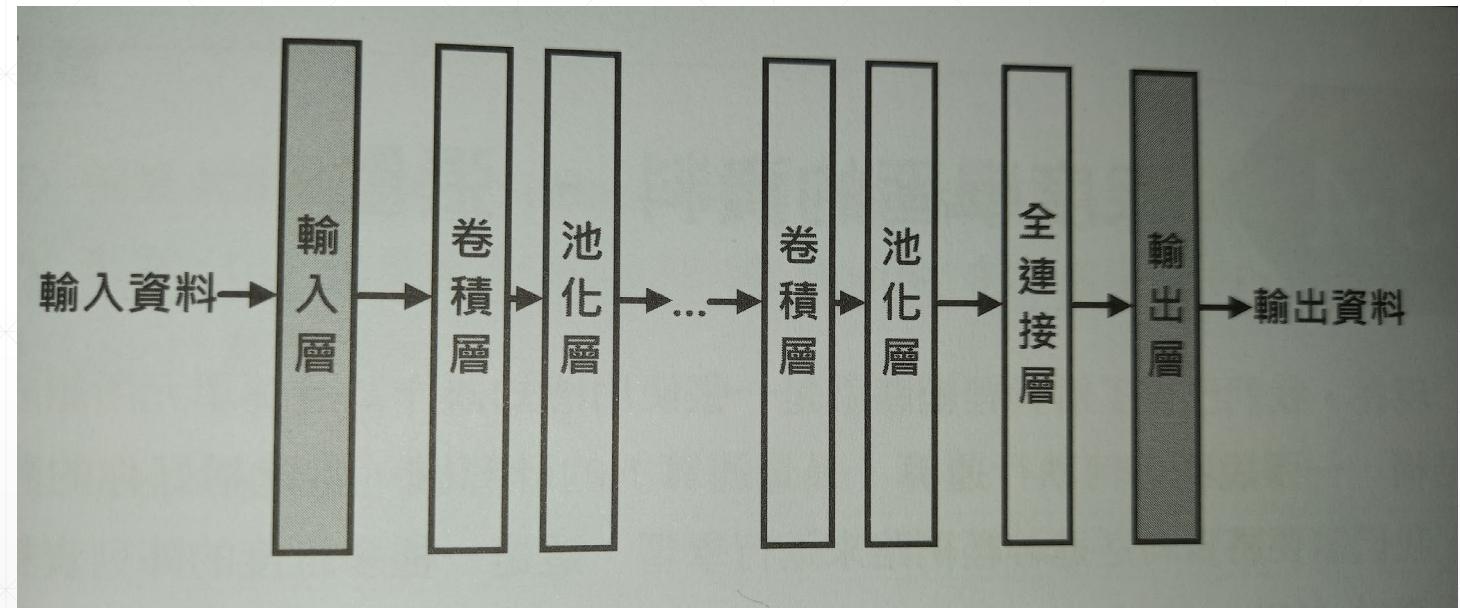
- 多層感知器：多加一層隱藏層來解決感知器無法處理的線性不可分問題。若隱藏層 ≥ 2 層，這個感知器就是一種深度精神網路。



3-3-3 深度學習的神經網路種類

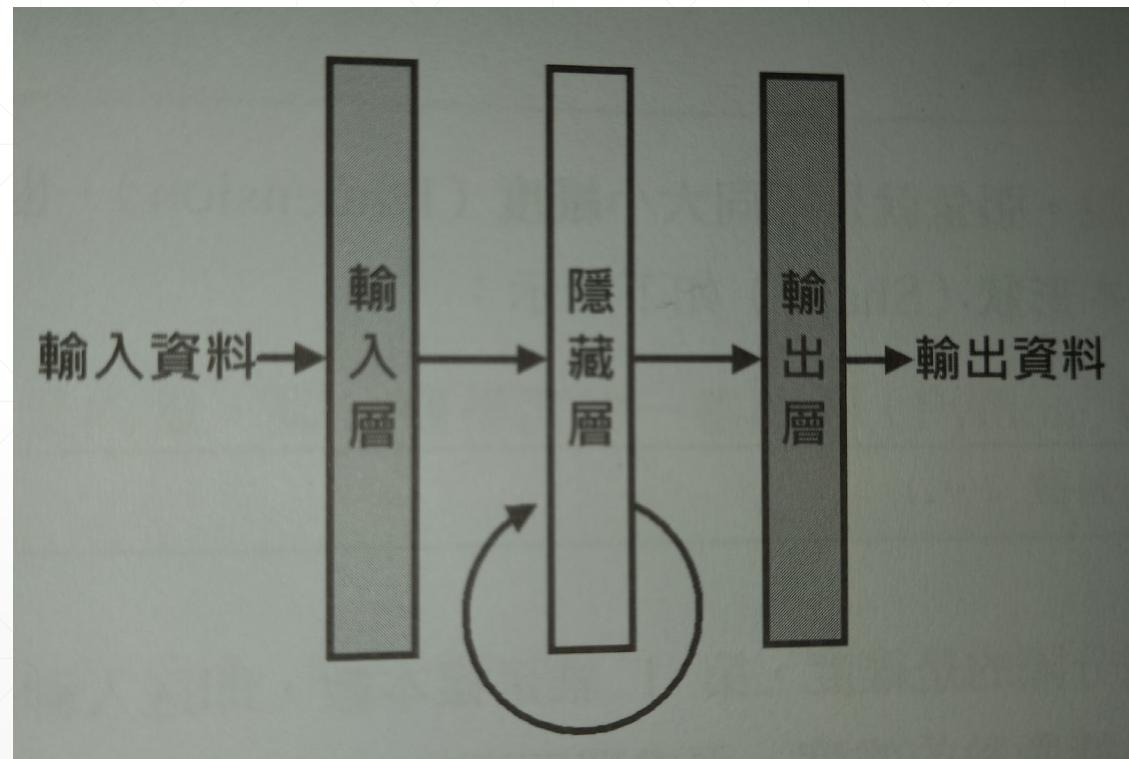
- 卷積神經網路：模仿人腦視覺處理區域的精神迴路，相當適合進行影像辨識。

會利用卷積層和池化層自動萃取圖像特徵，再傳送至全連接層。



3-3-3 深度學習的神經網路種類

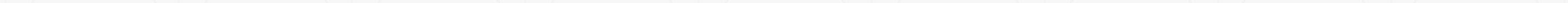
- 循環神經網路：是一種處理聲音、語言和影片等序列資料的精神網路，擁有短期記憶能力。



3-4 深度學習的資料 - 張量

3-4-1 張量的種類

- 我們送入機器學習進行學習的特徵資料，都是使用張量為基本資料結構。
- 以程式設計來說，張量就是不同大小維度(也稱為軸)的多維陣列。
- 基本形狀：(樣本數, 特徵1, 特徵2...)



3-4-1 張量的種類

- 0D張量=純量值=純量值張量

```
10.5  
0
```

🐍 Ch3_4_1.py

```
1 import numpy as np  
2  
3 x = np.array(10.5)  
4 print(x)  
5 print(x.ndim)
```

3-4-1 張量的種類

- 1D張量：一個一維度的一維陣列

```
[1.2 5.5 8.7 10.5]
```

```
1
```

```
Ch3_4_1a.py ×  
Ch3_4_1a.py  
1 import numpy as np  
2  
3 x = np.array([1.2, 5.5, 8.7, 10.5])  
4 print(x)  
5 print(x.ndim)
```

3-4-1 張量的種類

- 2D張量：就是矩陣，即維度
維 2 的陣列，有 2 個軸。

```
[[1.2 5.5 8.7 10.5]
```

```
[2.2 4.3 6.5 9.5]
```

```
[6.2 7.3 1.5 3.5]]
```

```
2
```

```
(3, 4)
```

```
Ch3_4_1b.py ×
```

```
Ch3_4_1b.py
```

```
1 import numpy as np
2
3 x = np.array([[1.2, 5.5, 8.7, 8.5],
4 | | | | [2.2, 4.3, 6.5, 9.5],
5 | | | | [6.2, 7.3, 1.5, 3.5]])
6 print(x)
7 print(x.ndim)
8 print(x.shape)
```

3-4-1 張量的種類

- 3D張量：就是矩陣，即維度
維 3 的陣列，有 3 個軸。

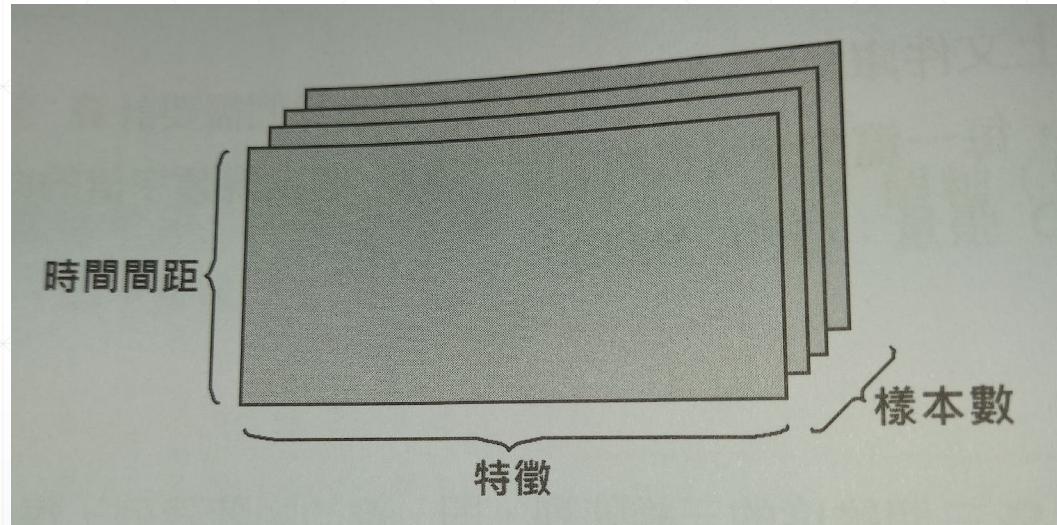
```
[[[1.2 5.5 3.3][8.7 8.5 4.4]]  
 [2.2 4.3 5.5][6.5 9.5 6.6]]  
 [6.2 7.3 7.7][1.5 3.5 8.8]]]  
  
3  
(3, 2, 3)
```

Ch3_4_1c.py

```
1 import numpy as np  
2  
3 x = np.array([[ [1.2, 5.5, 3.3],  
4 | | | | [8.7, 8.5, 4.4]],  
5 | | | | [[2.2, 4.3, 5.5],  
6 | | | | [6.5, 9.5, 6.6]],  
7 | | | | [[6.2, 7.3, 7.7],  
8 | | | | [1.5, 3.5, 8.8]]])  
9 print(x)  
10 print(x.ndim)  
11 print(x.shape)
```

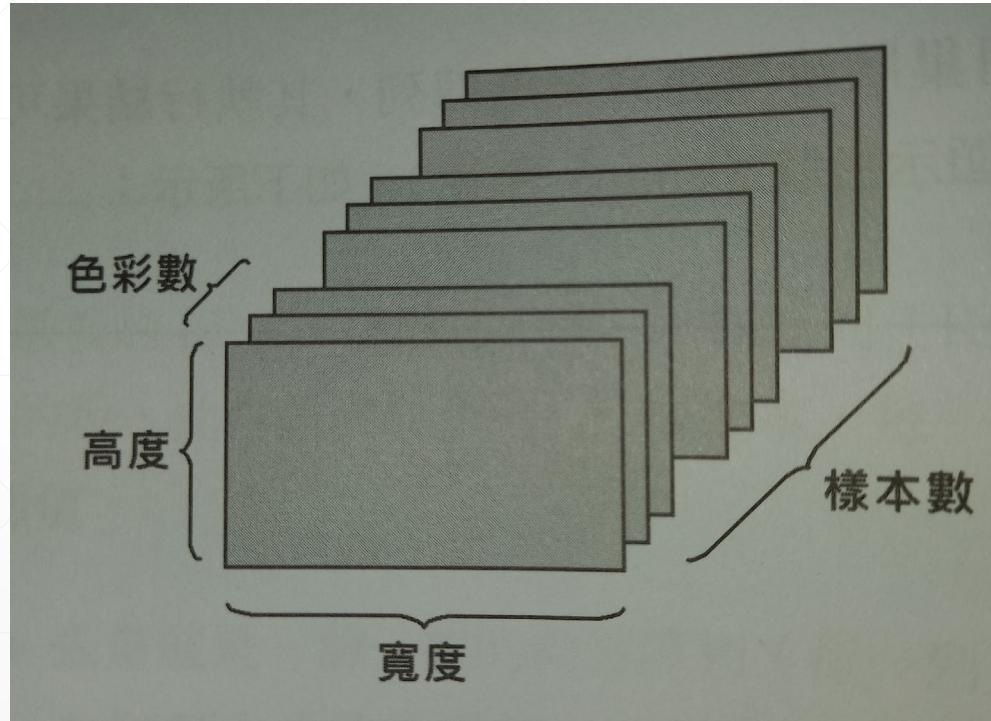
3-4-1 張量的種類

3D張量通常是特徵資料擁有
「時間間距」和循序性。



3-4-1 張量的種類

- 4D張量



3-4-2 張量運算

- $w_i x_i$ 是張量的積點運算
- $+b$ 是張量加法

$$t = f(z) = f\left(\sum_{i=1}^n w_i x_i\right) + b$$

3-4-2 張量運算

$$a = \begin{bmatrix} a1, a2 \\ a3, a4 \end{bmatrix}$$

$$s = \begin{bmatrix} s1, s2 \\ s3, s4 \end{bmatrix}$$

$$c = a + s = \begin{bmatrix} a1 + s1, a2 + s2 \\ a3 + s3, a4 + s4 \end{bmatrix}$$

$$a = \begin{bmatrix} 1, 2 \\ 3, 4 \end{bmatrix}$$

$$s = \begin{bmatrix} 5, 6 \\ 7, 8 \end{bmatrix}$$

$$c = a + s = \begin{bmatrix} 1+5, 2+6 \\ 3+7, 4+8 \end{bmatrix}$$

▪ 逐元素運算

3-4-2 張量運算

Ch3_4_2.py

```
1 import numpy as np
2
3 a = np.array([[1,2],[3,4]])
4 print("a=")
5 print(a)
6 s = np.array([[5,6],[7,8]])
7 print("s=")
8 print(s)
9 b = a + s
10 print("a+s=")
11 print(b)
12 b = a - s
13 print("a-s=")
14 print(b)
15 b = a * s
16 print("a*s=")
17 print(b)
18 b = a / s
19 print("a/s=")
20 print(b)
21
```

```
a=
[[1 2]
 [3 4]]
s=
[[5 6]
 [7 8]]
a+s=
[[ 6  8]
 [10 12]]
a-s=
[[-4 -4]
 [-4 -4]]
a*s=
[[ 5 12]
 [21 32]]
a/s=
[[ 0.2           0.33333333]
 [ 0.42857143   0.5         ]]
```

3-4-2 張量運算

The diagram shows two vectors, a and s , represented as column matrices. Vector a has components a_1, a_2, a_3, a_4 . Vector s has components s_1, s_2, s_3, s_4 . A horizontal arrow above vector a indicates its direction. Dotted arrows point from the components of a to the corresponding components of s in the formula for c . A large downward-pointing arrow indicates the result of the multiplication.

$$a = \begin{bmatrix} a_1, a_2 \\ a_3, a_4 \end{bmatrix}$$
$$s = \begin{bmatrix} s_1, s_2 \\ s_3, s_4 \end{bmatrix}$$
$$c = a \bullet s = \begin{bmatrix} a_1 * s_1 + a_2 * s_3, a_1 * s_2 + a_2 * s_4 \\ a_3 * s_1 + a_4 * s_3, a_3 * s_2 + a_4 * s_4 \end{bmatrix}$$

■ 點積運算

3-4-2 張量運算

Ch3_4_2a.py

```
1 import numpy as np  
2  
3 a = np.array([[1,2],[3,4]])  
4 print("a=")  
5 print(a)  
6 s = np.array([[5,6],[7,8]])  
7 print("s=")  
8 print(s)  
9 b = a.dot(s)  
10 print("a.dot(s)=")  
11 print(b)  
12
```

$$\begin{bmatrix} 1 \times 5 + 2 \times 7, 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7, 3 \times 6 + 4 \times 8 \end{bmatrix}$$

```
a=  
[[1 2]  
 [3 4]]  
s=  
[[5 6]  
 [7 8]]  
a.dot(s)=  
[[19 22]  
 [43 50]]
```

Thanks~
