

AI與深度學習簡介

吳庭育

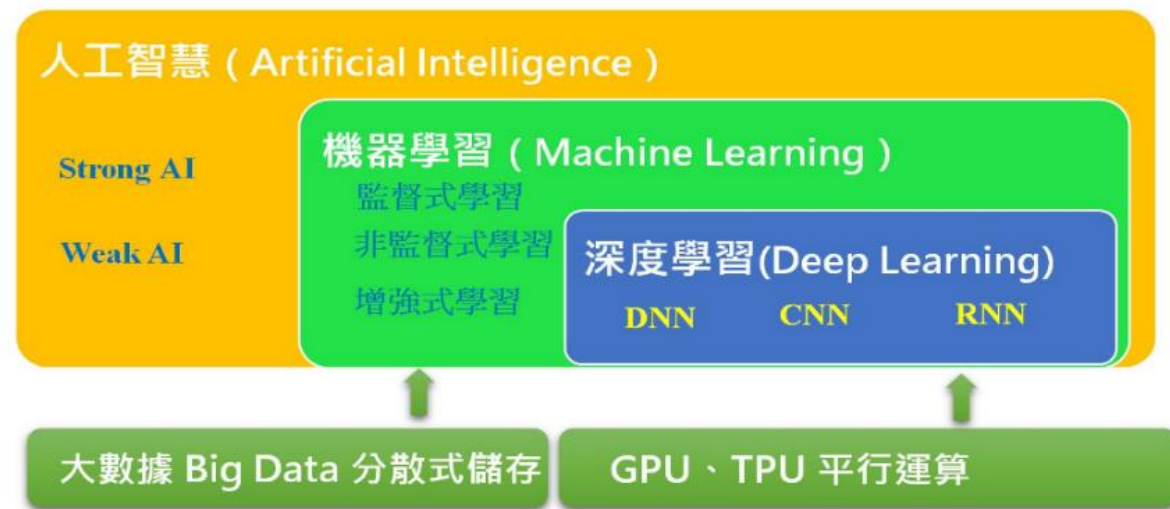
tyw429@gmail.com

Outline

- 人工智慧(AI)與深度學習
- 深度學習的函數集組成
- 深度學習的神經網路計算圖
- 深度學習的資料-張量

人工智慧(AI)的發展

- 人工智慧(Artificial Intelligence)的概念最早可以溯及1950年代，到了1980年，機器學習開始受到歡迎，大約到了2010年，深度學習在弱人工智慧系統方面終於有了重大突破，在2012年Toronto大學Geoffrey Hinton主導的團隊提出基於深度學習的AlexNet，一舉將ImageNet圖片資料集的識別準確率提高十幾個百分比，讓機器的影像識別率正式超越人類。



GPU

- 顯示卡它負責了電腦中一大部分的圖形運算，玩3D遊戲、繪圖、深度學習等都是靠它。而一張顯示卡是由 GPU、記憶體、電路板、散熱器等零件組成，而其中，GPU為其最重要的運算核心。



規格

	GEFORCE RTX 3090	GEFORCE RTX 3080 Ti	GEFORCE RTX 3080	GEFORCE RTX 3070 Ti	GEFORCE RTX 3070	GEFORCE RTX 3060 Ti	GEFORCE RTX 3060	GEFORCE RTX 3050
NVIDIA CUDA 核心	10496	10240	8960 / 8704	6144	5888	4864	3584	2560
加速時脈 (GHz)	1.70	1.67	1.71	1.77	1.73	1.67	1.78	1.78
記憶體大小	24 GB	12 GB	12 GB / 10 GB	8 GB	8 GB	8 GB	12 GB	8 GB
記憶體類型	GDDR6X	GDDR6X	GDDR6X	GDDR6X	GDDR6	GDDR6	GDDR6	GDDR6

NVIDIA v.s AMD



- GPU 分為兩大陣營 **NVIDIA(輝達) GeForce & AMD(超微) Radeon**。
- 這兩家廠商是專門設計 GPU 並將設計好 GPU 後會交給晶圓廠製作，如：台積電。GPU 做好後，交給板卡廠商去完成一張顯示卡，如：華碩、技嘉等。那簡單解釋了這些廠商們的關係後，我們看回 NVIDIA 與 AMD 這兩家。
- 這兩大陣營各有各的特色
 - NVIDIA 陣營優點就是性能較好，且多數專業軟體只能用 NVIDIA 的卡加速，但是價格也比較高。
 - AMD 陣營則是較便宜，但之前的驅動穩定度有點不一。
 - 簡單來說如果是遊戲需求，兩家都可以；
 - 因為 NVIDIA 已經用 CUDA 技術根基這領域很久了如果有在用專業軟體，像是跑分析、3D 渲染、深度學習等專業需求，就選 NVIDIA

GPU等級 比較 - NVIDIA

RTX3080 / RX5700XT

- NVIDIA 的 遊戲顯卡為 Geforce 系列，其中 RTX 為光追系列，GTX 則為不支援"光線追蹤"的系列。
- NVIDIA GPU 的命名規則為 **新舊+等級**，而字尾如有Ti代表加強版，如：RTX 2080 Ti 與 RTX 3060 Ti，3060 Ti 雖然為30系列，但是字尾80及Ti 代表 2080 Ti 比 3060 Ti 定位還要更高階。另外 Super 字尾則是為小升級的意思，性能較一般版本高。



GPU等級 比較 - AMD

RTX3080 / RX5700XT

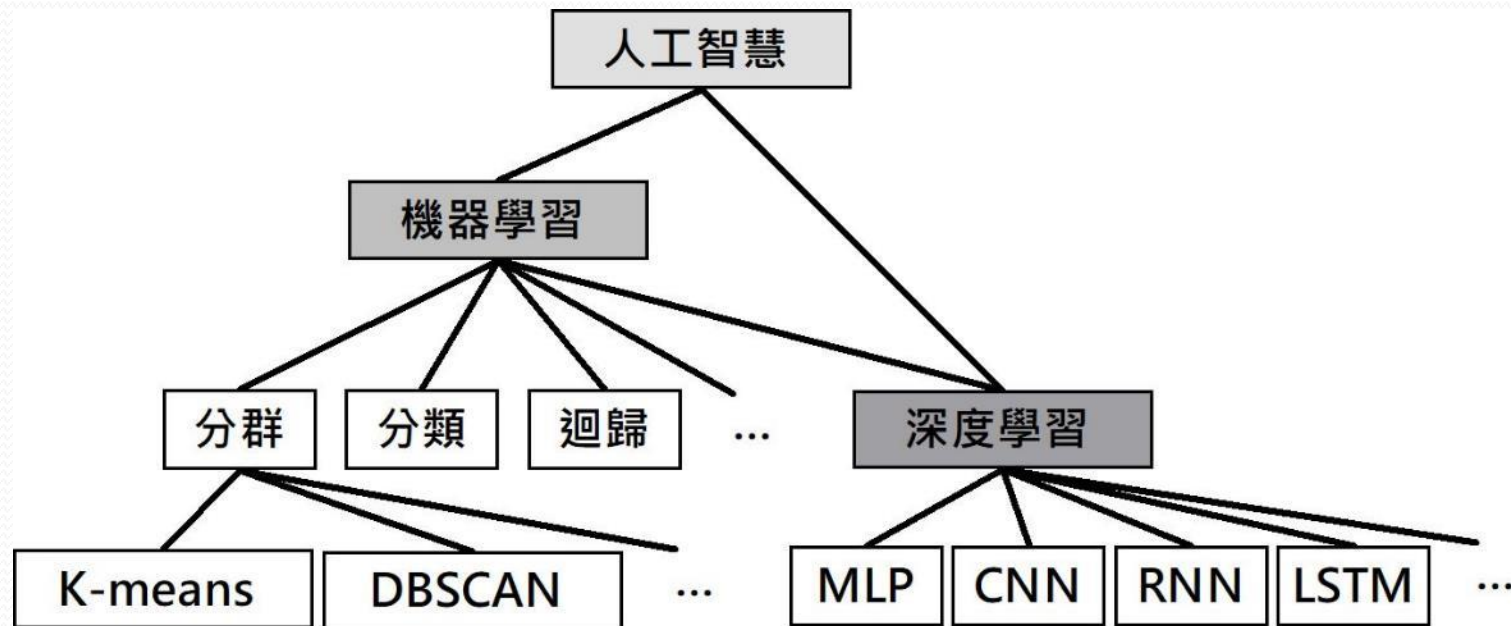
- AMD的 遊戲顯卡為 RADEON RX 系列。
- AMD的命名規則分為兩種，第一種為新款GPU的命名，有四位數。第一個數字為第幾代，後面的數字代表等級，而有XT字尾則是代表加強版。



AMD VS NVIDIA

等級	NVIDIA	AMD	解析度
高	RTX 3090	X	4K 順玩
	RTX 3080 Ti	X	
	RTX 3080	RX 6900 XT	
		RX 6800 XT	
	RTX 3070 Ti	RX 6800	2K 順玩
	RTX 3070		
	RTX 3060 Ti	RX 6700 XT	
	RTX 3060	RX 6600 XT	FHD 順玩
	RTX 2060	RX 5700	
	GTX 1660 S	RX 5600 XT	FHD 可玩
	GTX 1650 S	RX 5500 XT	
	GTX 1650	RX 580/590	
低	X	X	別玩
	GT 1030		

人工智慧、機器學習與深度學習

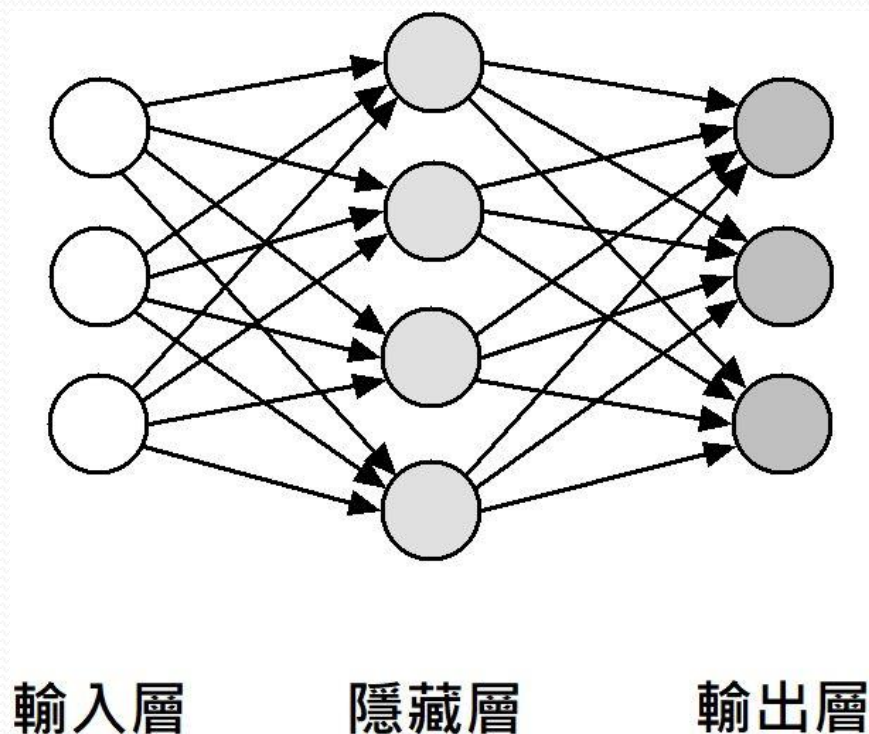


深度學習 (Deep Learning)

- 深度學習(DL)是機器學習的一種方式，深度學習的概念早在 90 年代就已經存在，但是因為當時的電腦運算能力不佳，效率不彰，因此沒有獲得太多的關注。直到近年來全球電腦設備因為網路的串連，分散式儲存技術的成熟，產生了龐大的數據，再加上電腦硬體進步以及大量伺服器平行運算能力，而使得深度學習捲土重來，並迅速地蓬勃發展。
- 深度學習是在訓練機器的直覺，請注意！這是直覺訓練，並非知識學習。
 - 訓練深度學習辨識一張狗圖片，我們是訓練機器知道這張圖片是狗，並不是訓練機器學習到狗有4隻腳、會叫或是一種哺乳類動物等關於狗的相關知識。

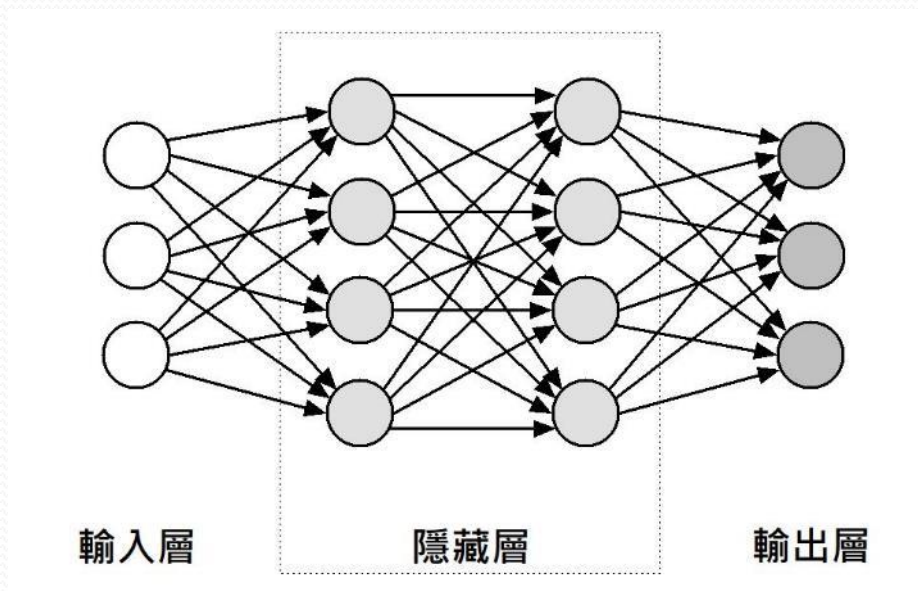
深度學習是一種神經網路 (1/3)

- 深度學習就是模仿人類大腦神經元(Neuron)傳輸的一種神經網路架構(Neural Network Architectures)。



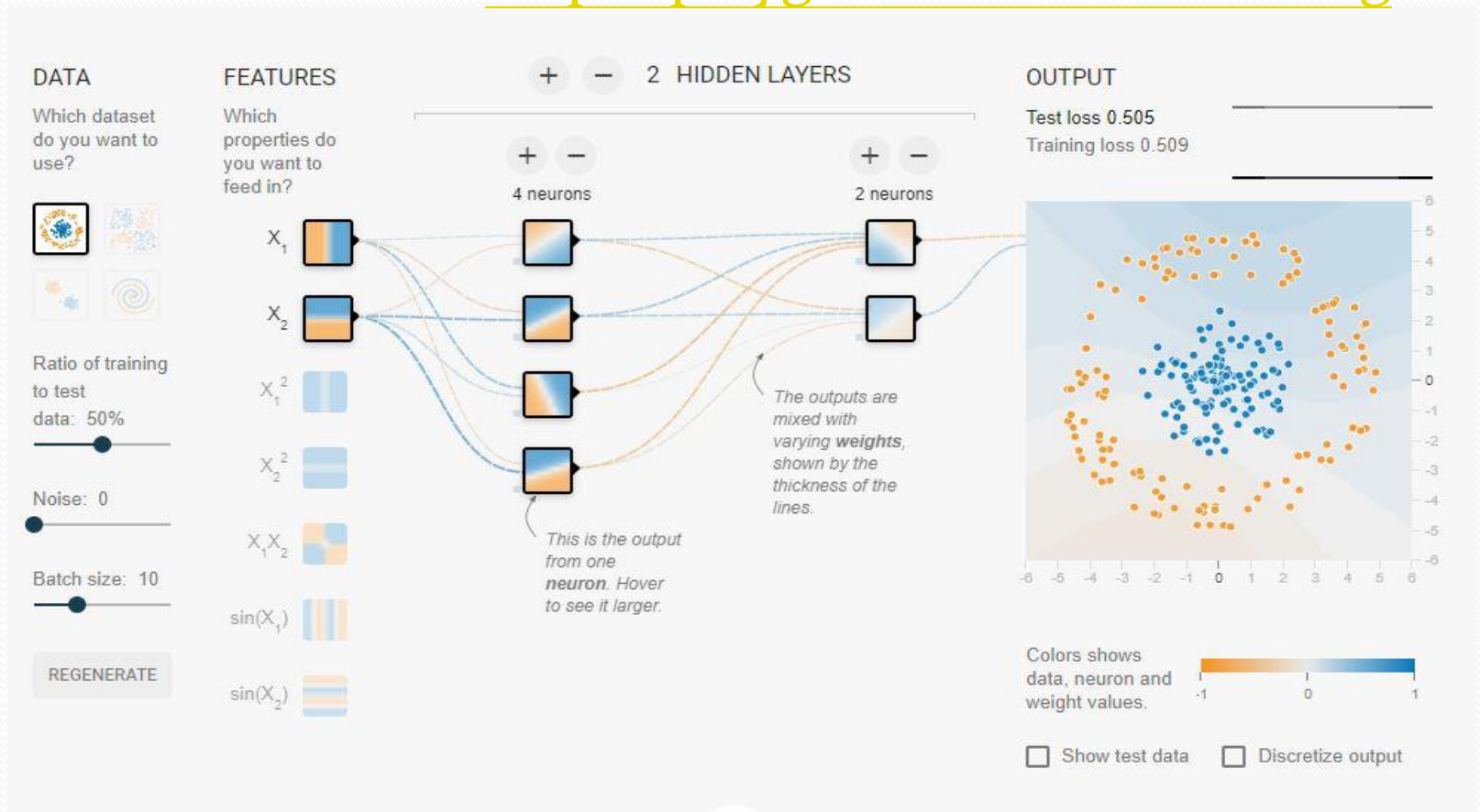
深度學習是一種神經網路(2/3)

- 深度學習使用的神經網路稱為「深度神經網路」(Deep Neural Networks, DNNs)，其中間的隱藏層有很多層，意味著整個神經網路十分的深(Deep)，可能高達150層隱藏層。
 - 神經網路只需擁有2層隱藏層，加上輸入層和輸出層共四層之上，就可以稱為深度神經網路。



深度學習是一種神經網路(3/3)

- 深度學習實作上分成三個步驟
 - 建構神經網路 / 設定目標 / 匯入資料進行學習
- TensorFlow網站 <http://playground.tensorflow.org>



深度學習就是一個函數集

- 深度學習的神經網路是使用一個**函數集**來執行多次矩陣相乘的非線性轉換運算。
- 這個函數集如果已經完成訓練，當我們將**特徵資料**送入神經網路，經過每一層神經元的輸入和輸出(將一個向量映射至另一個向量的過程)，整個神經網路可以輸出最後結果的最佳解。



TensorFlow (1/2)



- TensorFlow是Google Brain Team開發，在2005年底開放專案後，2017年推出第一個正式版本，稱為TensorFlow。
- 因為其輸入/輸出的運算資料是向量、矩陣等多維度的數值資料，稱為張量(Tensor)。
- Tensor張量就是經過這些流程Flow的數值運算還產生輸出結果，稱為 $\text{Tensor} + \text{Flow} = \text{TensorFlow}$ 。

TensorFlow 是屬於比較低階的深度學習 API，開發者可以自由配置運算環境進行深度學習神經網路研究。TensorFlow 的特點如下：

- 處理器：可以在 CPU、GPU、TPU 上執行
- 跨平台：可在 Windows、Linux、Android、iOS、Raspberry Pi 執行。
- 分散式執行：具有分散式運算能力，可以同時在數百台電腦上執行訓練，大幅縮短訓練的時間
- 前端程式語言：Tensorflow 可以支援多種前端程式語言，例如：Python、C++、Java 等，目前以Python的表現最佳。
- 高階 API：Tensorflow 可以開發許多種高階的 API，例如：Keras、TFLearn、TF-Slim、TF-Layer 等，其中以 Keras 功能最完整。

Keras (1/2)



- Keras是Google工程師Francois Chollet使用Python 開發的一套開放原始碼的高階神經網路函式庫，支援多種後台(Backend)的神經網路計算引擎，其預設引擎是TensorFlow。
- Keras的特色
 - Keras能夠使用相同的Python程式碼在CPU或GPU上執行。
 - Keras提供高階APIs來快速建構深度學習模型的神經網路。
 - Keras 預建全連接、卷積、池化、RNN、LSTM和GRU等多種神經層，來建立多層感知器、卷積神經網路和循環神經網路。

Keras (2/2)



Keras 已經將訓練模型的輸入層、隱藏層、輸出層架構做好，只需要加入正確的參數如輸入層神經元個數、隱藏層神經元個數、輸出層神經元個數、激發函式等，訓練上較 TensorFlow 容易許多。

Keras 可說是最適合初學者及研究人員的深度學習套件，不像 TensorFlow 必須自行設計一大堆的計算公式，讓使用者可以在很短的時間內學習並開發應用。當然 Keras 也有小小的缺點，就是自由度不如 TensorFlow，且沒有辦法使用到底層套件的全部功能。

Tensor Flow vs. Keras

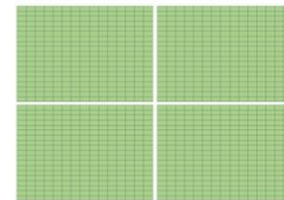
Tensorflow 架構圖說明:



Keras

Tensorflow-GPU 版本

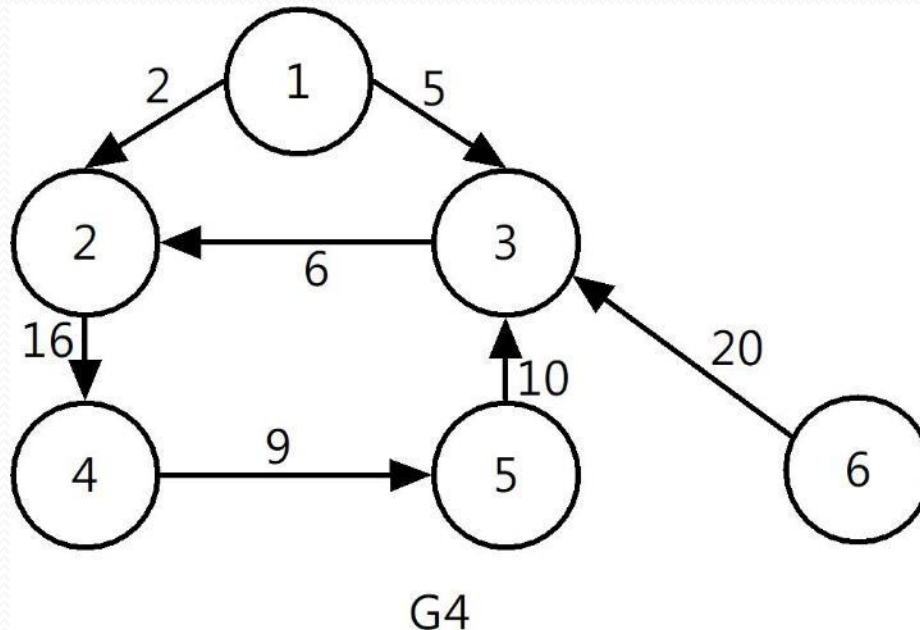
CUDA 與 cuDNN



GPU 具有數千個核心

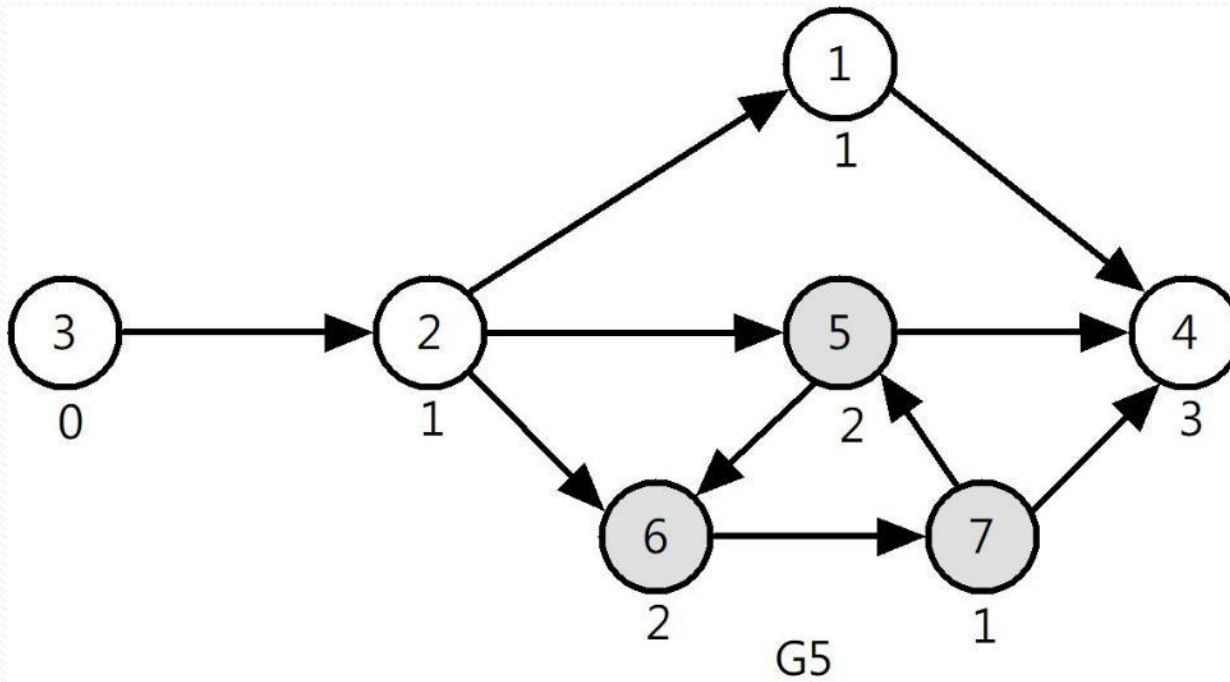
加權圖形結構

- 圖形在解決問題時通常需要替邊線加上一個數值，這個數值稱為「權重」(Weights)。
 - 常見權重有：時間、成本或長度等。
- 如果圖形擁有權重，稱為「加權圖形」(Weighted Graph)。
 - 方向性圖形G4在邊線上的數值是權重，這就是一個加權圖形。



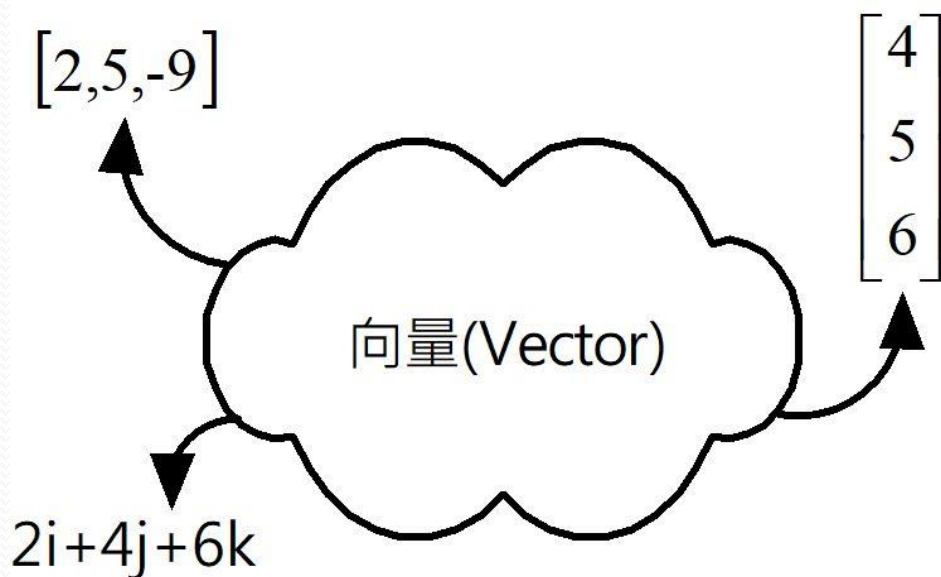
方向性循環圖

- 方向性圖形的頂點之間可能會有循環路徑的迴圈。
 - 方向性圖形G5擁有迴圈 $5 \rightarrow 6 \rightarrow 7 \rightarrow 5$



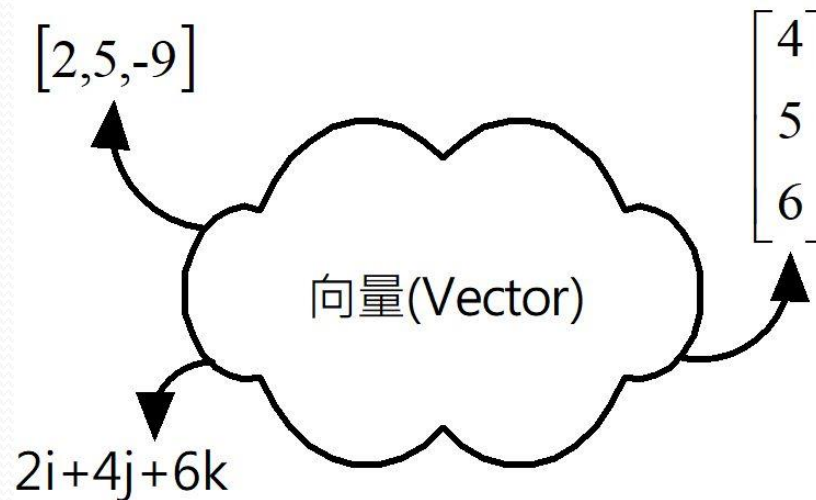
向量

- 向量(Vector)
 - 向量是方向和大小值(如速度、加速度和動力等)。
 - 向量是一序列數值。
 - 在程式語言是使用一維陣列來表示。



矩陣

- 矩陣(Matrix)
 - 矩陣類似向量，只是形狀是二維表格的列(Rows)和欄(Columns)，我們需要使用列和欄來取得指定元素值。
 - 程式語言是使用二維陣列方式來表示。



微分(Differentiation)

- 微分(Differentiation)的目的在求瞬間的變化量

- 照相機的瞬間變化量就是照片。
- 用水的體積對水的重量作微分，找出的瞬間變化量，就是水的密度，因為當水的體積增加時，重量也會增加，而變動水的一個單位體積，可以讓水的重量有多少變化，這就產生了瞬間變化量，而水的密度就是單位體積的重量。

函數 $f(x)$ 的微分主要有兩種表示方法： $f'(x)$ 或 $\frac{df(x)}{dx}$

- 微分的運算並不困難，常數的微分是0，單一變數函數的微分

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

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

$$f(x) = 2x^3 + 5x + 2 +$$

$$\frac{df(x)}{dx} = 6x^2 + 5 +$$

- 函數微分的範例

偏微分(Partial Differentiation)

- 一個多變數的函數，可以分別針對x和y來進行微分，稱為 偏微分(Partial Differentiation)。

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

對變數 x 偏微分，就是將變數 y 視為常數來微分，如下所示：

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

- 對變數y偏微分，也就是將變數x視為常數來微分。

$$\frac{\partial f(x,y)}{\partial y} = 12xy + 4$$

連鎖率

因為深度學習的神經網路有很多層，其產生的函數是一種合成函數： $f(x) = g(h(x))$ ，函數 $h(x)$ 是 $g(x)$ 函數的引數，例如： $f(x)$ 範例合成函數，如下所示：

$$f(x) = (2x^3 + 5x + 2)^4$$

$$\text{函數 } g(x) : f(h) = h^4$$

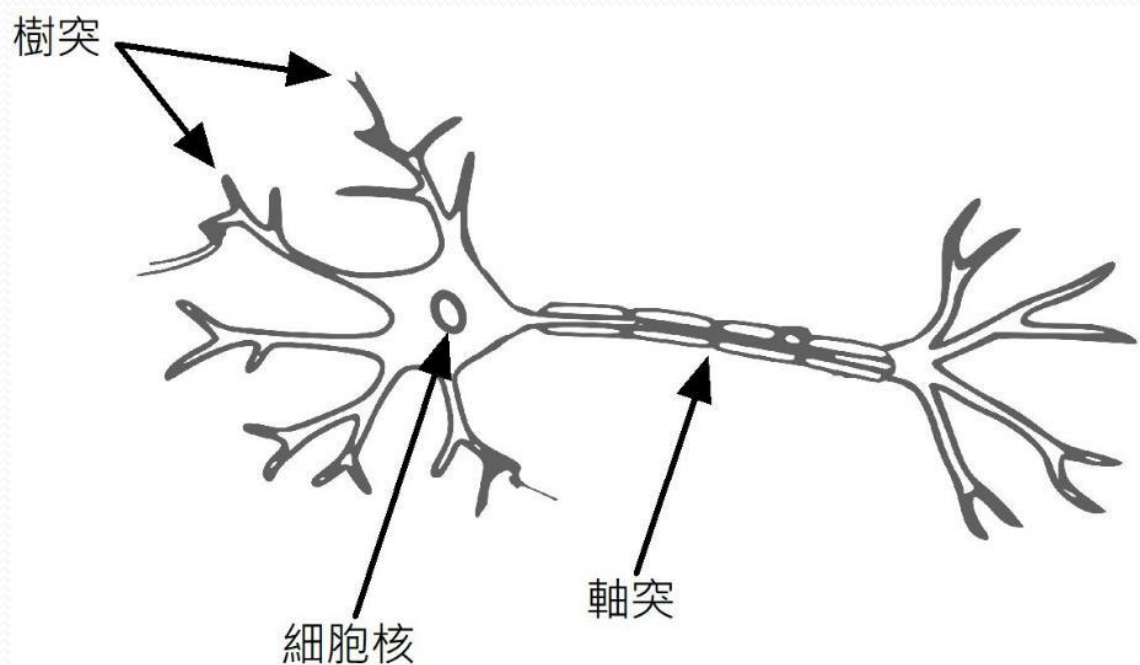
$$\text{函數 } h(x) : h(x) = 2x^3 + 5x + 2$$

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

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

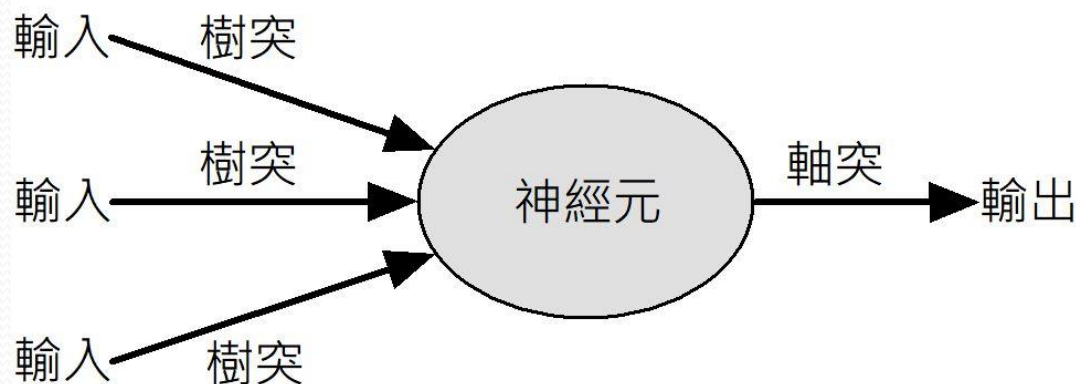
神經元

- 人類的大腦十分複雜，據估計大腦擁有超過300億個使用各種方式連接的「神經元」(Neuron)，能夠使用神經元之間的連接來傳遞訊息和處理訊息(電子脈衝訊號)，可以讓人類產生記憶、思考、計算和識別的能力



神經元

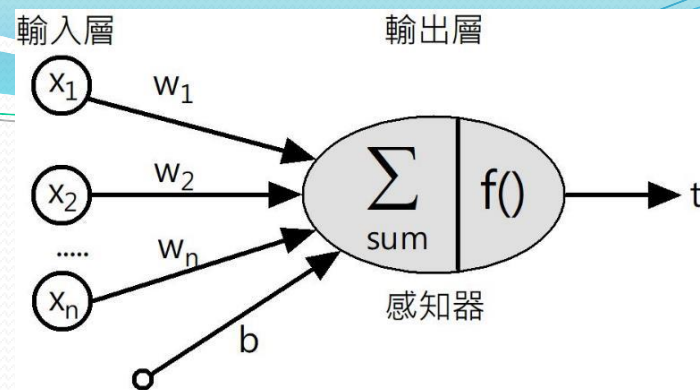
- 神經元擁有多個輸入，但是只有一個輸出。就算同時有多個訊號輸入，如果訊號不夠強，神經元也不會有任何輸出。
- 當短時間有大量高強度訊號輸入時，神經元就會被激活，然後透過軸突向其他神經元輸出電子脈衝訊號。



人工神經元

- 輸入1~3的值是0或1；權重是浮點數2.5、3.0或-0.2等，然後，我們可以計算出訊號強度，如下所示：
 - 訊號強度 = 輸入1*權重1+輸入2*權重2+輸入3*權重3
- 人工神經元的輸出是判斷加總後的訊號強度是否大於等於「閾值」(Thresholds)的數值，以決定是否激活神經元，其規則如下所示：
 - 訊號強度 \geq 閾值 \rightarrow 輸出1
 - 訊號強度 $<$ 閾值 \rightarrow 輸出0
- ○

感知器 (Perceptions)



- 感知器 (Perceptions) 是 1957 年 Frank Rosenblatt 在康奈爾航空實驗室 (Cornell Aeronautical Laboratory) 所提出，一個可以模擬人類感知能力的機器，一種進化版的人工神經元和二元線性分類器。
- 感知器是神經網路的基本組成元素，單一感知器就是一種最簡單形式的二層神經網路，擁有輸入層和輸出層。
- $f(n)$ 啟動函數/激活函數 (Activation Function)。

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

感知器 (Perceptions)

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

- 上述激活函數 $f(n)$ 在傳統的感知器是使用階梯函數 (Step Function)

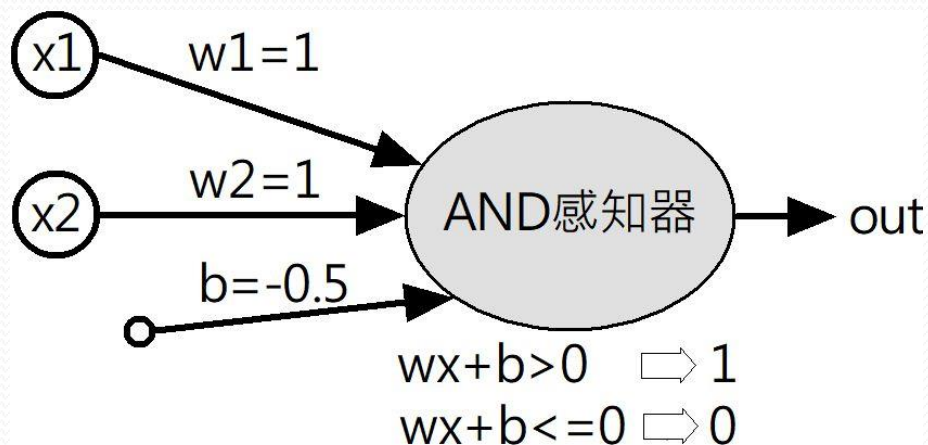
$$f(x) = \begin{cases} 1 & \text{如果 } wx + b > s \\ 0 & \text{否則} \end{cases}$$

感知器 - AND邏輯閘

- AND邏輯閘有2個輸入x1與x2，和一個輸出out，其符號和真值表。



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



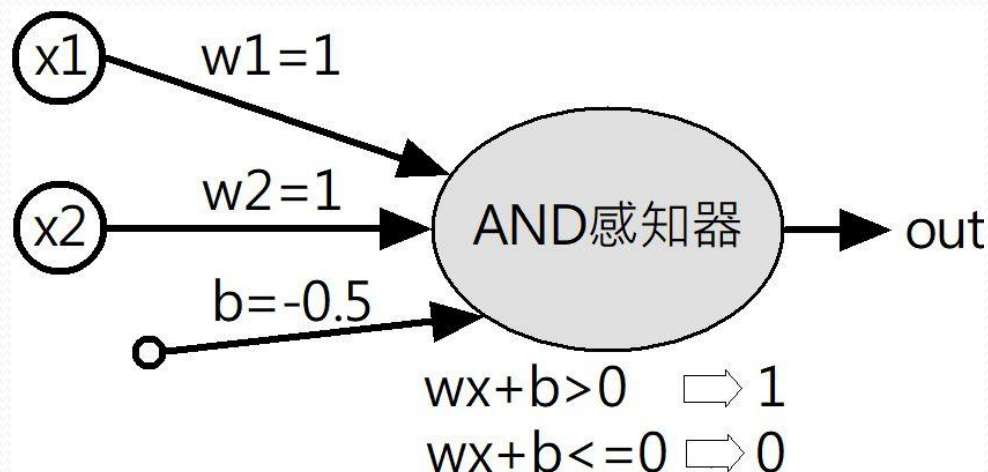
感知器 - AND邏輯閘

第一列輸入值：x1=0和x2=0

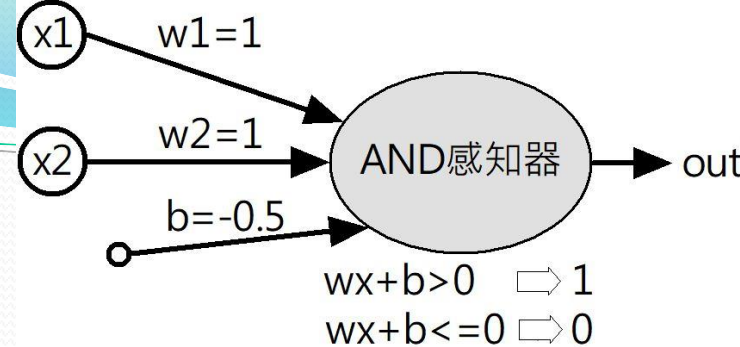
- 請使用輸入值、權重和偏向量來計算感知器的訊號強度，如下所示：

$$w1*x1+w2*x2+b \rightarrow 1*0+1*0+-0.5 = -0.5$$

- 上述運算式的計算結果是-0.5，因為值 ≤ 0 ，所以輸出0，和真值表的輸出0相同，輸出正確，不用調整權重和偏向量。



感知器 - AND邏輯閘



第二列輸入值：x1=0和x2=1

- 感知器訊號強度的計算如下所示：

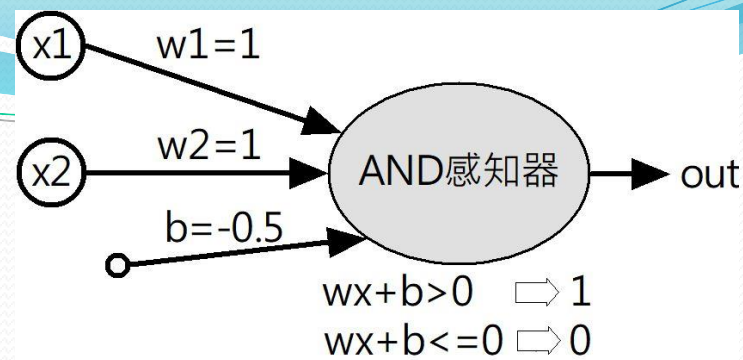
$$w_1 * x_1 + w_2 * x_2 + b \rightarrow 1 * 0 + 1 * 1 + -0.5 = 0.5$$

- 上述運算式的計算結果是0.5，因為值 > 0 ，所以輸出1，和真值表的輸出0不同，因為輸出不正確，我們需要調整權重或偏向量。因為需要減少訊號強度，讓感知器不激活，而不是增加訊號強度來激活感知器，所以需要減少偏向量b值，將b值減0.5成為-1，然後重新計算感知器的訊號強度，如下所示：

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

- 上述運算式的計算結果是0，成功減少訊號強度從0.5至0，因為值 ≤ 0 ，所以輸出0，和真值表的輸出0相同，輸出正確，不用再調整權重或偏向量。

感知器 - AND邏輯閘



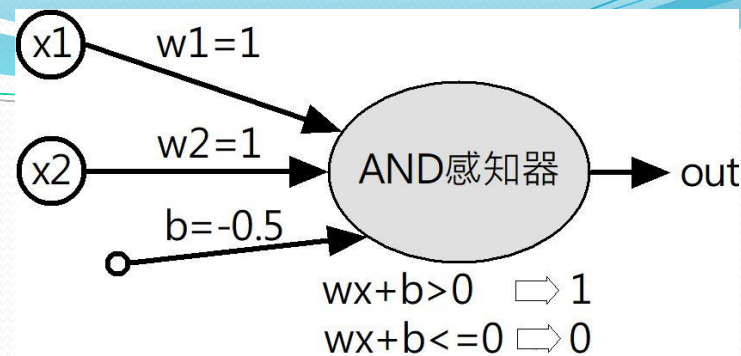
第三列輸入值：x1=1和x2=0

- 感知器訊號強度的計算，目前偏向量b的值是-1，如下所示：

$$w1*x1+w2*x2+b \rightarrow 1*1+1*0+-1 = 0$$

- 上述運算式的計算結果是0，因為值 ≤ 0 ，所以輸出0，和真值表的輸出0相同，輸出正確，不用再調整權重和偏向量。

感知器 - AND邏輯閘



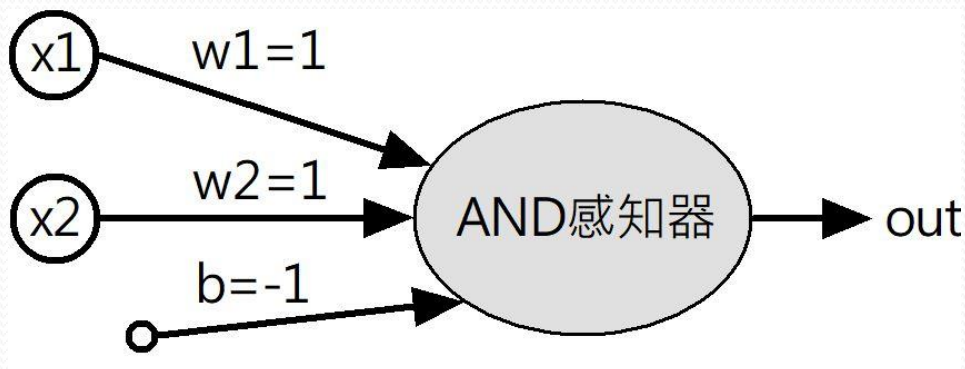
- 第四列輸入值： $x_1=1$ 和 $x_2=1$
- 感知器訊號強度的計算，目前偏向量 b 的值是-1，如下所示：

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

- 上述運算式的計算結果是1，因為值 > 0 ，所以輸出1，和真值表的輸出1相同，輸出正確，不用再調整權重和偏向量。

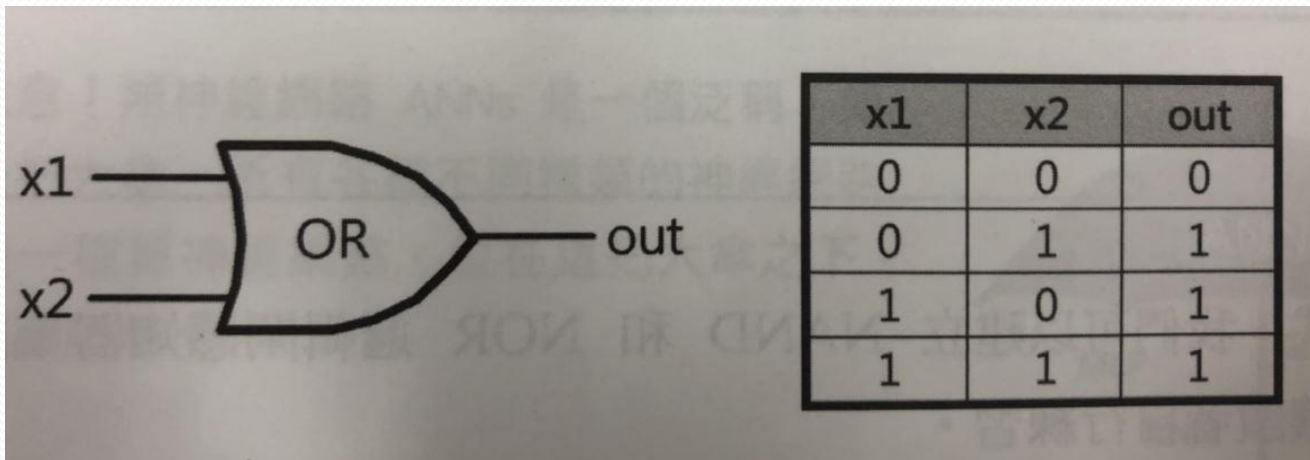
感知器 - AND邏輯閘

- 經過4次輸入/輸出資料的訓練過程，我們可以找出AND邏輯閘感知器的權重 $w_1=1$ ； $w_2=1$ 和偏向量 $b=-1$ 。



練習

感知器 - OR邏輯



張量的種類

- 張量就是不同大小維度(Dimension)，也稱為軸(Axis)的多維陣列，其基本形狀(Shape)如下所示：

(樣本數, 特徵1, 特徵2....)

- 上述「,」逗號分隔的是維度，第1維是樣本數，即送入神經網路訓練的資料數，之後是資料特徵數的維度，視處理問題而不同。
- 例如：圖片資料是4D張量，擁有3個特徵寬、高和色彩數，如下所示：

(樣本數, 寬, 高, 色彩數)

- 上述張量的維度數是4，也稱為軸數，或稱為等級(Rank)。

0D張量

- 0D張量就是純量值(Scalar)，或稱為純量值張量(Scalar Tensor)，以NumPy來說，0D張量就是float32或float64的數值資料(Python程式：Ch3_4_1.py)，如下所示：

```
import NumPy as np
```

```
x = np.array(10.5)
```

```
print(x)
```

```
print(x.ndim)
```

- 上述程式碼建立值10.5的純量值，其執行結果可以看到值是一個純量值10.5，軸數是0(ndim屬性)。

1D張量

- 1D張量就是向量，即一個維度的一維陣列 (Python程式：Ch3_4_1a.py)，如下所示：

```
x = np.array([1.2, 5.5, 8.7, 10.5])
```

```
print(x)
```

```
print(x.ndim)
```

- 上述程式碼使用清單(List，或稱列表、串列)建立4個元素的一維陣列，其執行結果可以看到一維陣列的向量共有4個項目，軸數是1(ndim屬性)。

2D張量

- 2D張量就是矩陣，即二個維度的二維陣列，有2個軸 (Python程式：Ch3_4_1b.py)，如下所示：

```
x = np.array([[1.2, 5.5, 8.7, 8.5],  
              [2.2, 4.3, 6.5, 9.5],  
              [6.2, 7.3, 1.5, 3.5]])
```

```
print(x)
```

```
print(x.ndim)
```

```
print(x.shape)
```

- 上述程式碼使用巢狀清單建立二維陣列，其執行結果可以看到一個一維陣列，每一個元素是一個向量，軸數是2(ndim屬性)，最後顯示形狀(shape屬性)。

3D張量

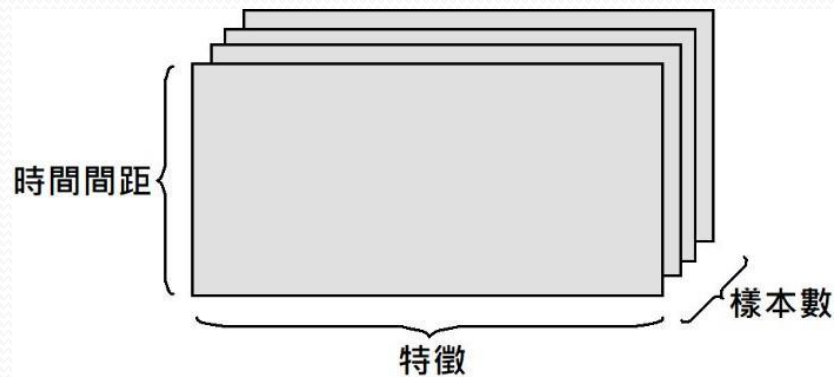
- 3D張量就是三個維度的三維陣列，即一維的矩陣陣列，每一個元素是一個矩陣，共有3個軸(Python程式：Ch3_4_1c.py)，如下所示：

```
x = np.array([[[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]]])
```

```
print(x)
```

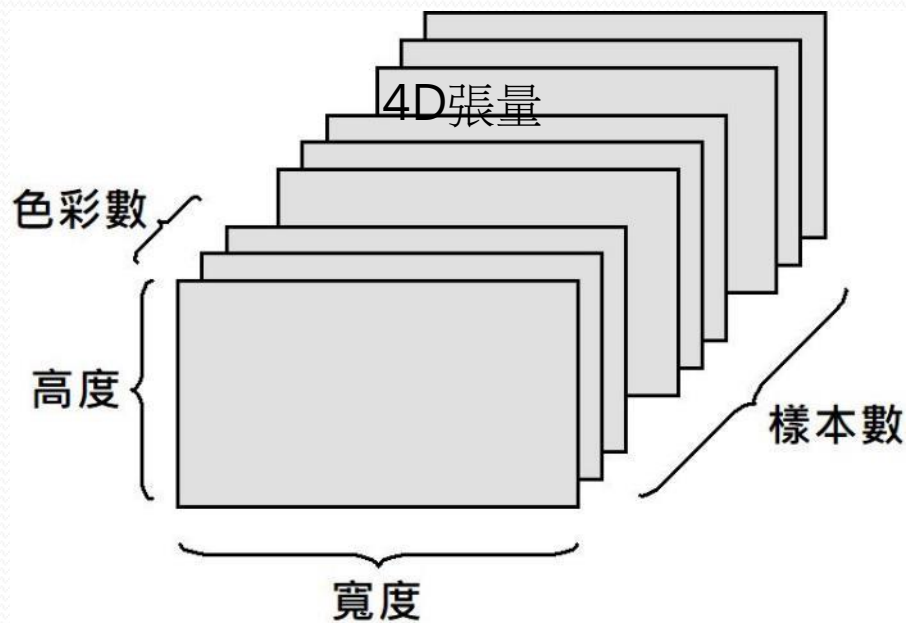
```
print(x.ndim)
```

```
print(x.shape)
```



4D張量

- 4D張量就是四個維度的四維陣列，共有4個軸，真實的特徵資料圖片就是一種4D張量。

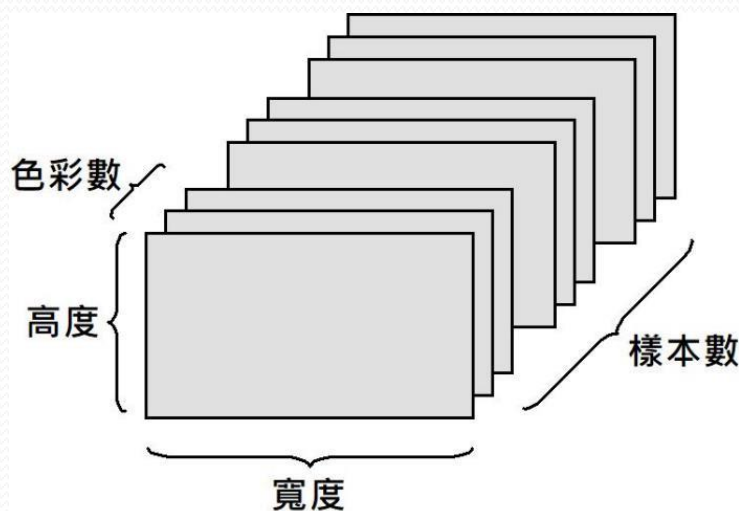


5D張量與更高維度的張量

- 5D張量是五個維度的五維陣列，共有5個軸，更高維度擁有更多軸，以5D張量來說，真實的特徵資料影片就是一種5D張量，比圖片多了一個軸，即每一秒有多少個畫面 (Frames)，如下所示：

(樣本數, 畫面數, 寬度, 高度, 色彩數)

- 一部256x144的YouTube影片，每秒有240個畫面，現在有10部YouTube影片，其5D張是：(10, 240, 256, 144, 3)。



張量運算

- 感知器輸出 t 的計算是一個張量運算的運算式，最後使用 啟動函數判斷是否激活神經元/感知器，如下所示：

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

- 上述 $w_i x_i$ 是張量的點積運算， $+b$ 是張量加法，換句話說，神經網路就是使用張量運算，從輸入資料開始，一層神經層接著一層神經層來逐步計算出神經網路的輸出結果。

逐元素運算

- 逐元素運算(Element-wise Operations)是指運算套用在每一個張量的元素，可以執行張量的加、減、乘和除四則運算，以2D張量(即矩陣)為例，2D張量的對應元素可以執行加、減、乘和除的四則運算。
- 準備使用加法的張量運算為例，
 - 2D張量a有a1~a4個元素，s有s1~s4。

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

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

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

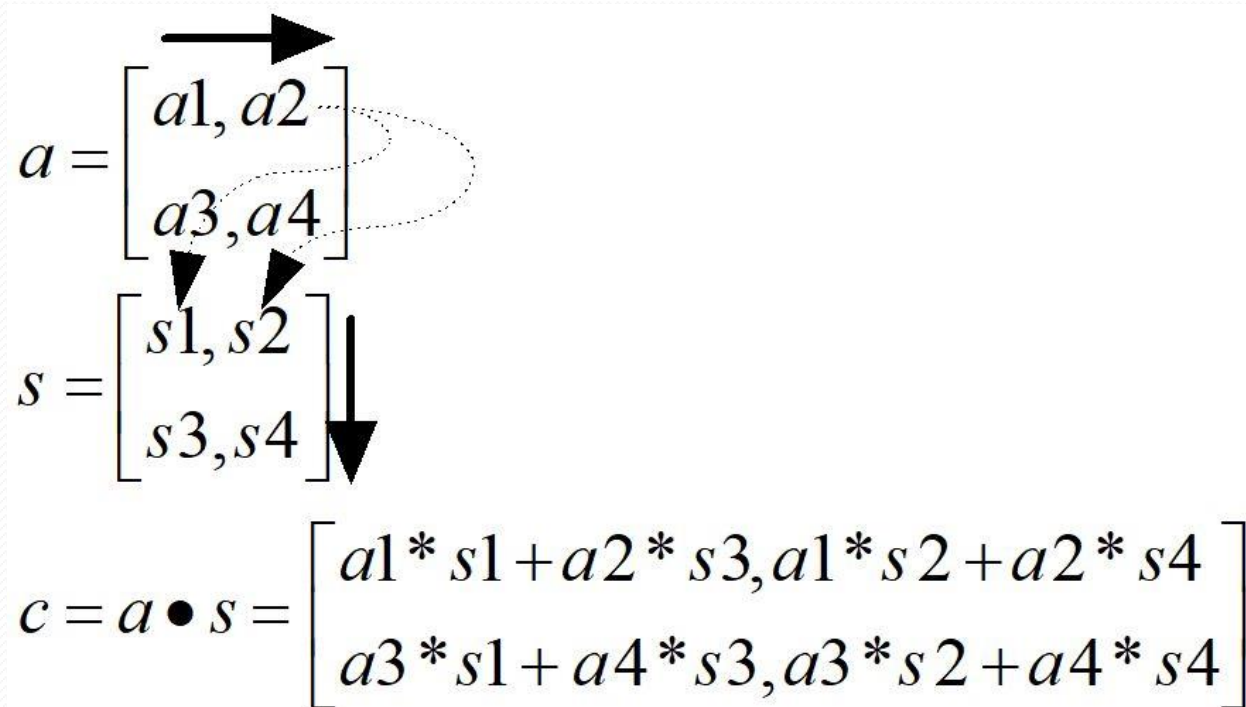
$$a = \begin{bmatrix} 1, 2 \\ 3, 4 \end{bmatrix}$$

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

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

點積運算

- 點積運算(Dot Product)是兩個張量對應元素的列和行的乘積和，例如：使用之前相同的2個2D張量來執行點積運算。



The diagram illustrates the dot product operation between two 2D tensors, a and s , to produce a scalar result c . Tensor a is represented as a column vector $a = \begin{bmatrix} a1, a2 \\ a3, a4 \end{bmatrix}$ with a horizontal arrow above it indicating its row dimension. Tensor s is represented as a column vector $s = \begin{bmatrix} s1, s2 \\ s3, s4 \end{bmatrix}$ with a vertical arrow to its right indicating its column dimension. Dotted lines with arrows show the pairing of elements: $a1$ with $s1$, $a2$ with $s3$, $a3$ with $s2$, and $a4$ with $s4$. The resulting scalar c is calculated as the sum of these products:

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

動腦時間

1. 請問什麼是神經網路？人工神經元？感知器？
2. 請簡述神經網路的結構有哪幾種？
3. 請問什麼是張量？張量的種類有哪些？如何執行張量運算？

