

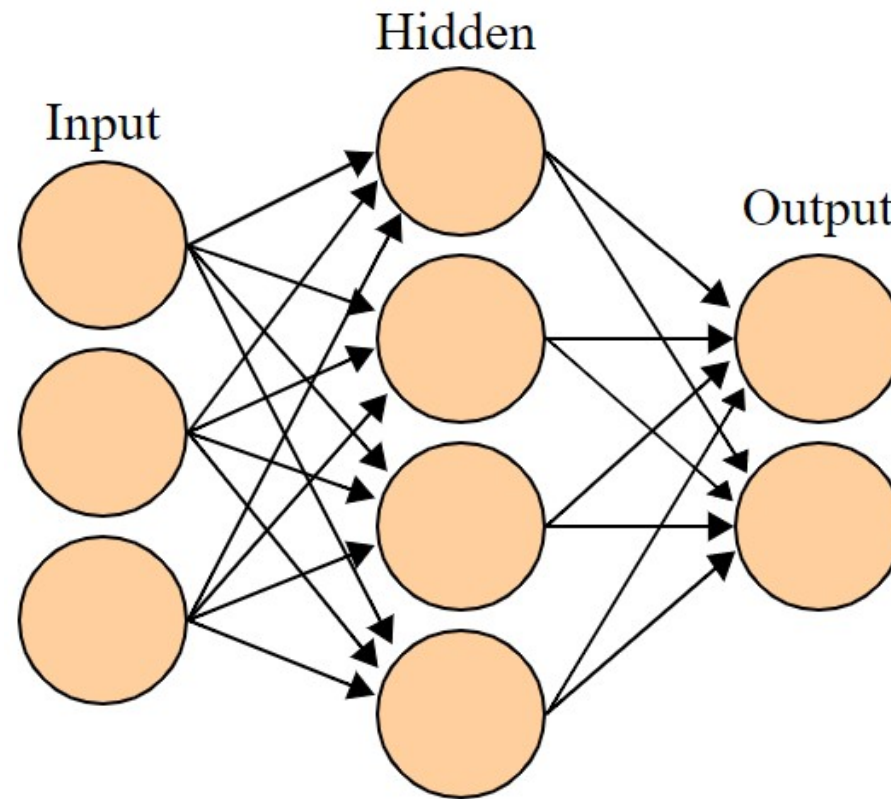
# 類神經網路簡介

Speaker: Malo



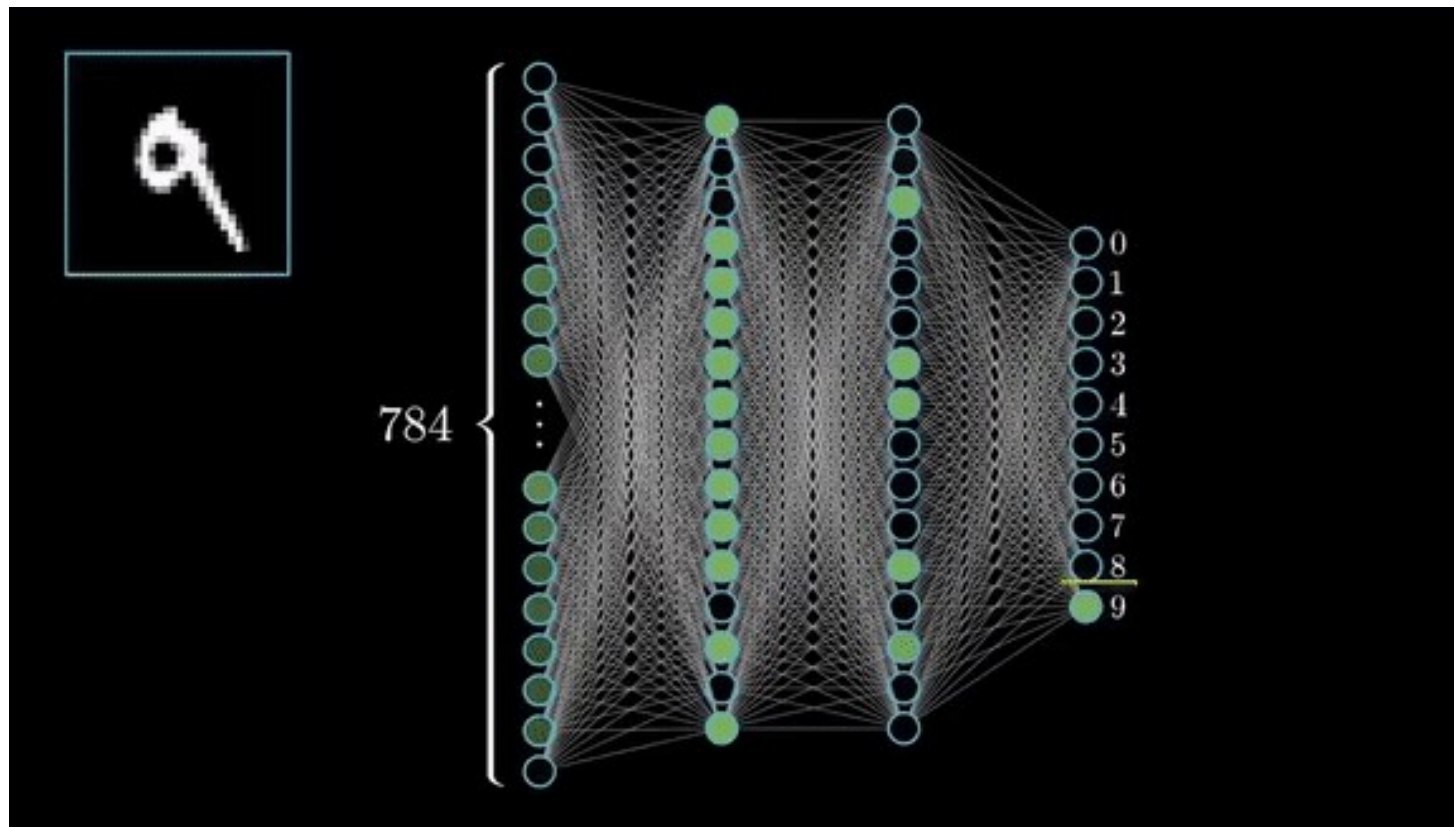
# ANN

- Artificial Neural Network



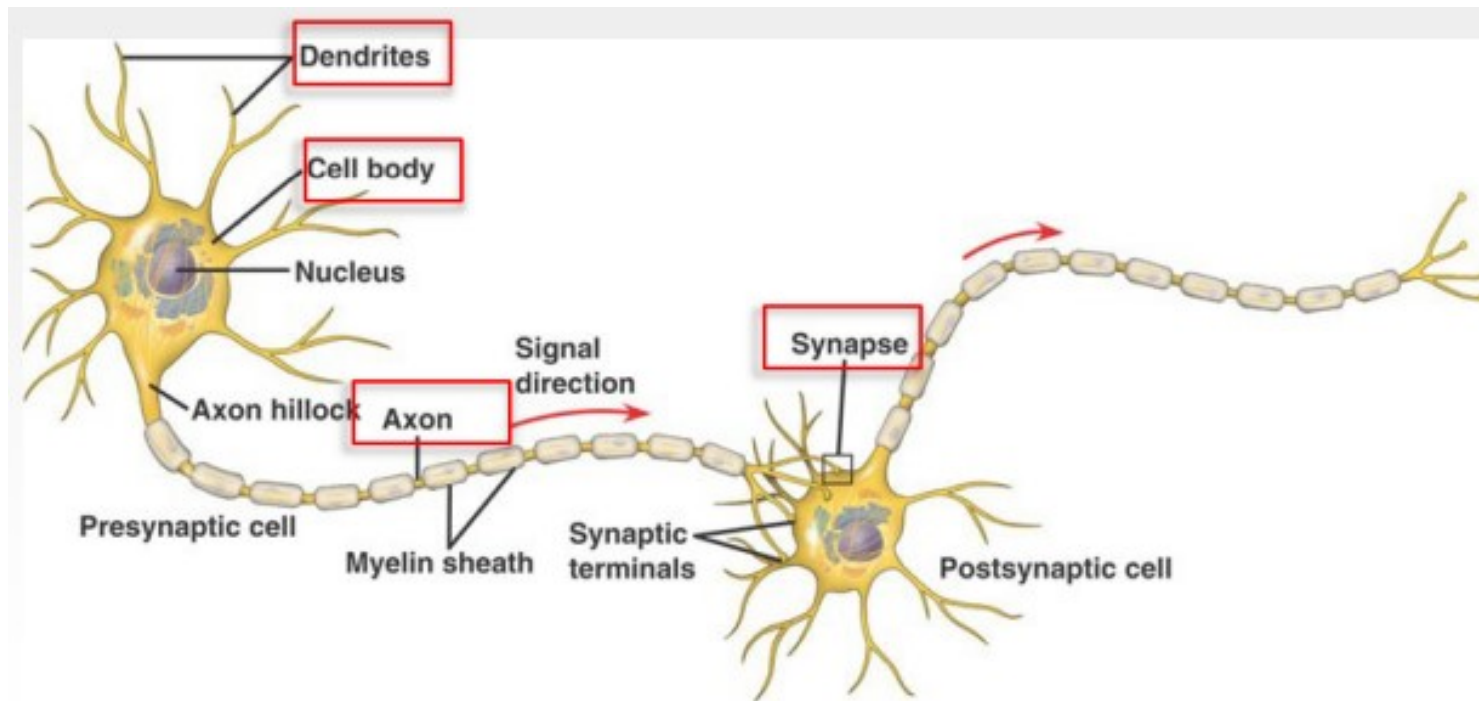
# ANN

- 以MNIST手寫數字為例



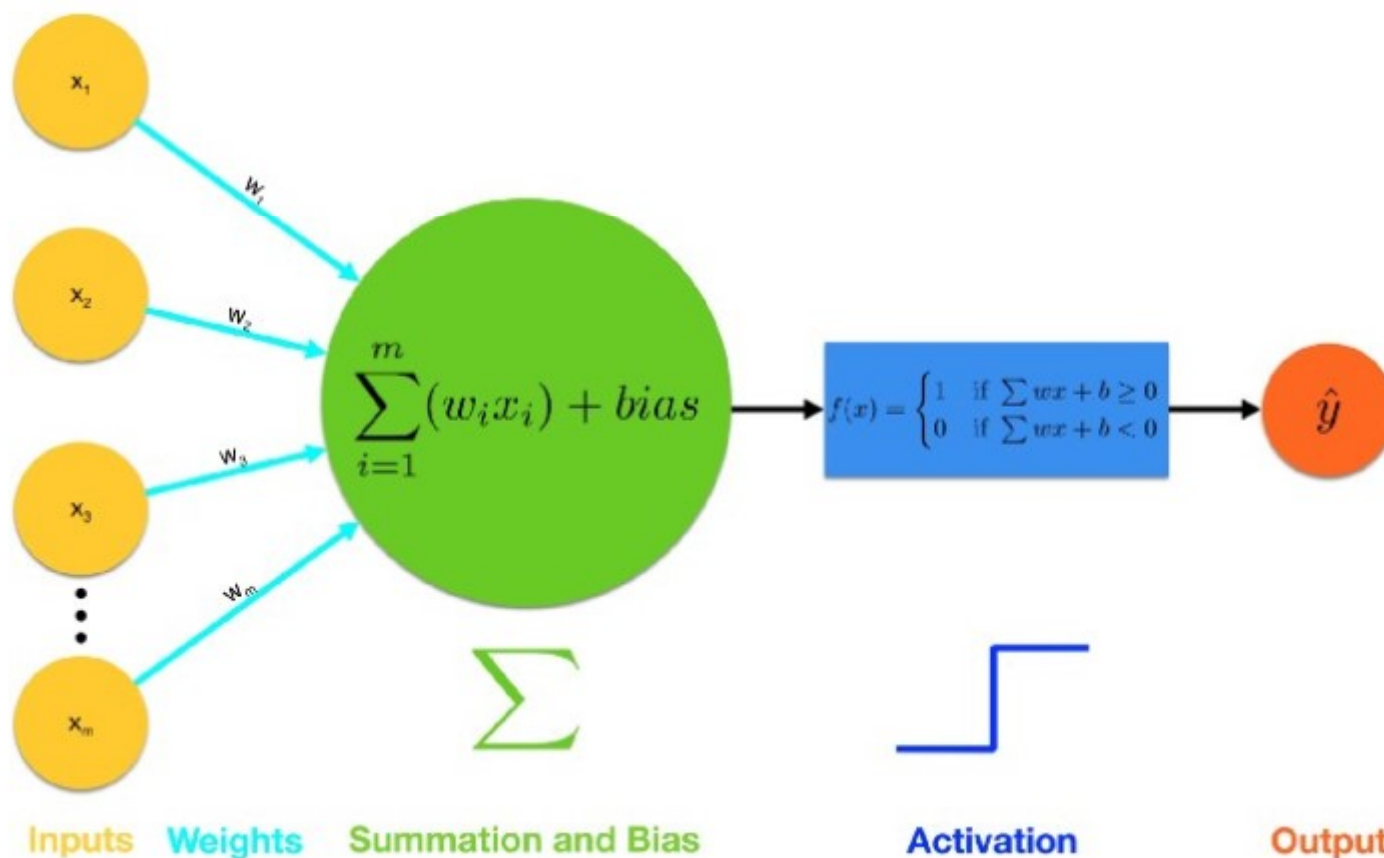
# 人腦的突觸

- 講ANN時常見的圖，



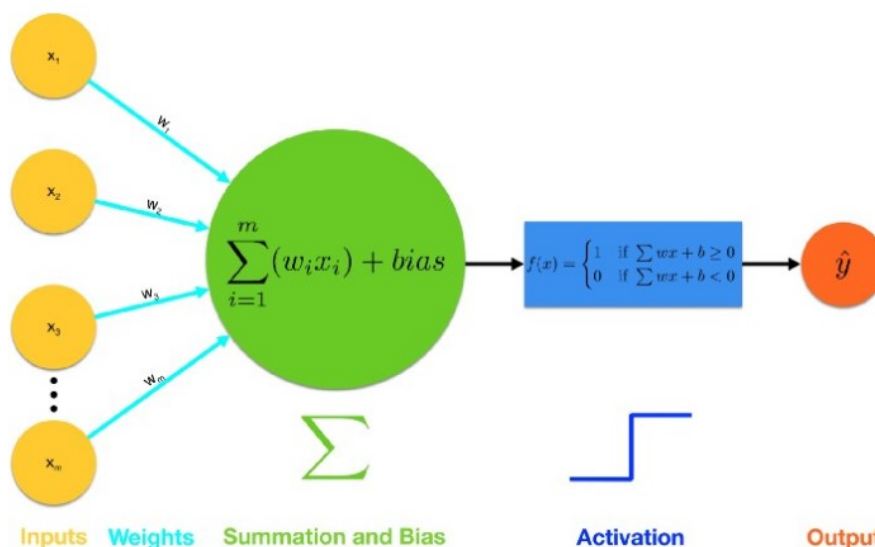
# ANN





- 電腦科學家就是由這樣的概念啟發，發展出人工神經網路的架構








# ANN

- **Input:** 對應突觸收進來的訊號，也就是我們的輸入資訊
- **Weight:** 權重，決定哪一個訊號比較重要
- **Summation and Bias:** 整合訊息
- **Activation:** 依照整合訊息的分數判斷結果，做輸出
- **Output:** 輸出訊號，給下一個神經元



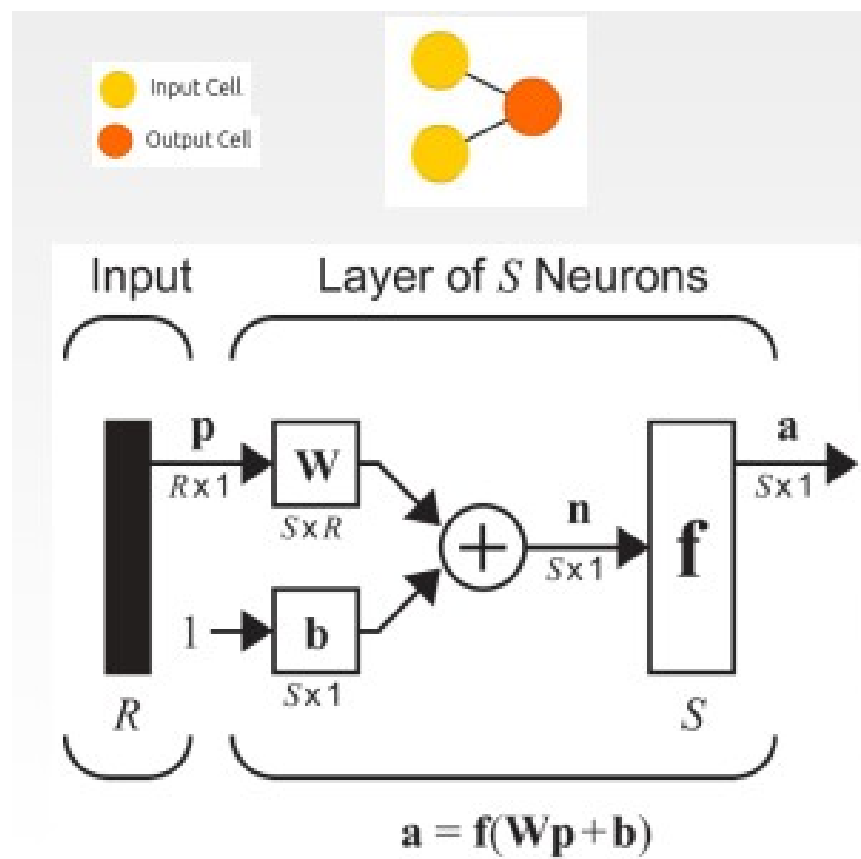
| Name                   | Input/Output Relation   | Icon  |
|------------------------|---|---|
| Hard Limit             | $a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$                            |    |
| Symmetrical Hard Limit | $a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$                          |    |
| Linear                 | $a = n$   |  |
| Saturating Linear      | $a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$ |  |

# 轉換函式

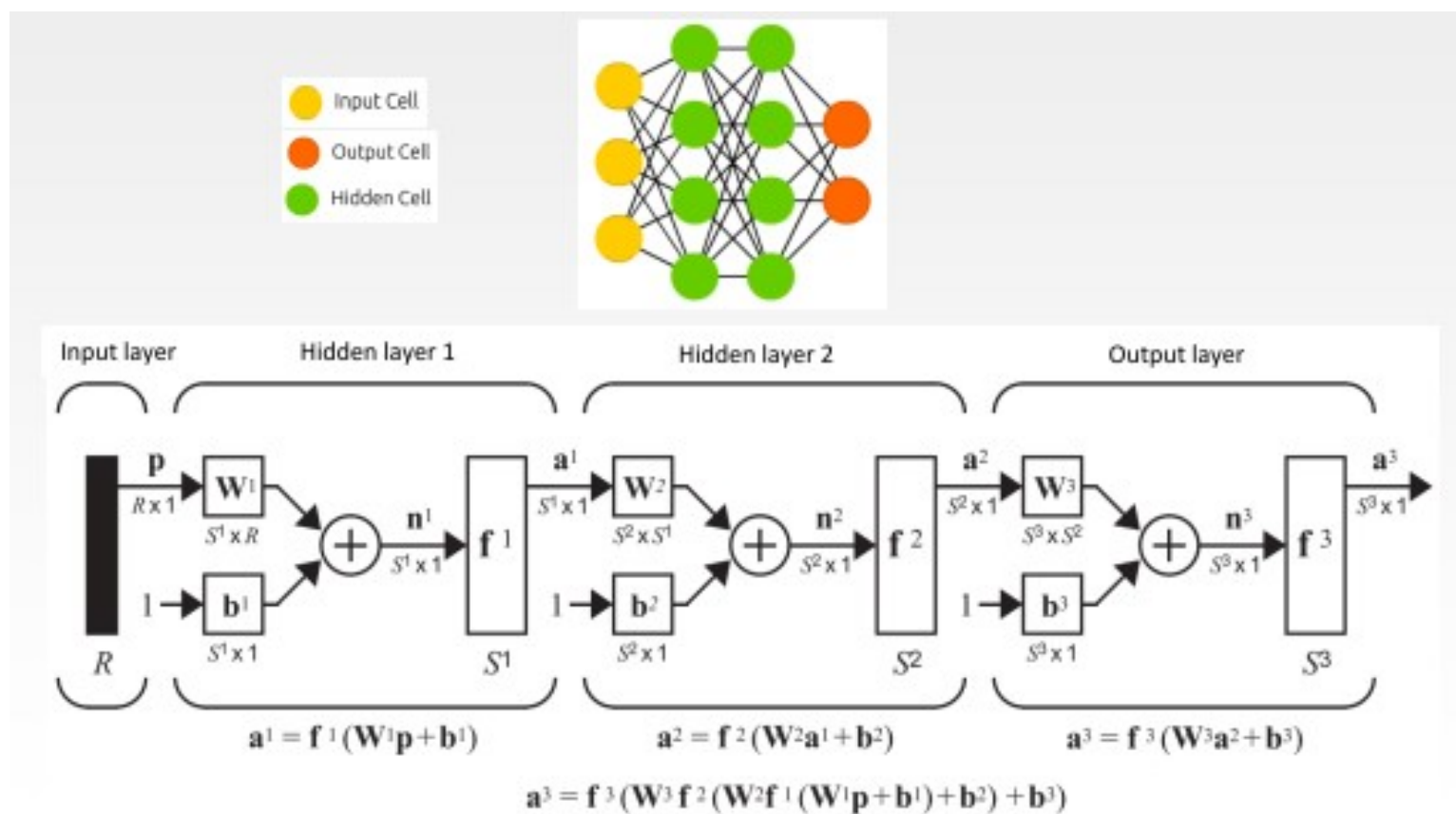
|   |  |   |
|---|--|---|
| Symmetric Saturating Linear                     | $a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$       |    |
| Log-Sigmoid                                     | $a = \frac{1}{1 + e^{-n}}$   |    |
| Hyperbolic Tangent Sigmoid                      | $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$  |    |
| Positive Linear<br>Rectified Linear Unit (ReLU) | $a = 0 \quad n < 0$ $a = n \quad 0 \leq n$                                     |  |
| Competitive                                     | $a = 1 \quad \text{neuron with max } n$ $a = 0 \quad \text{all other neurons}$ |  |



# 單一層

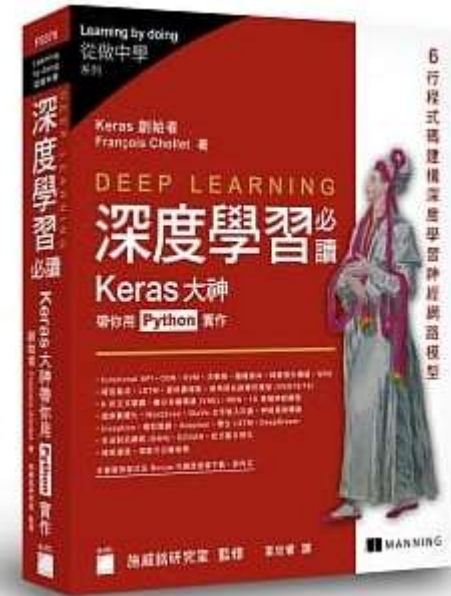


# 多層



# 改以本書介紹

- <https://keras.io/>
- **Python Deep Learning library**
- 可配合TensorFlow, CNTK, or Theano
- [https://www.books.com.tw/products/0010822552:100-1-01-r0vq68ygz\\_D\\_2aabd0\\_B\\_1](https://www.books.com.tw/products/0010822552:100-1-01-r0vq68ygz_D_2aabd0_B_1)



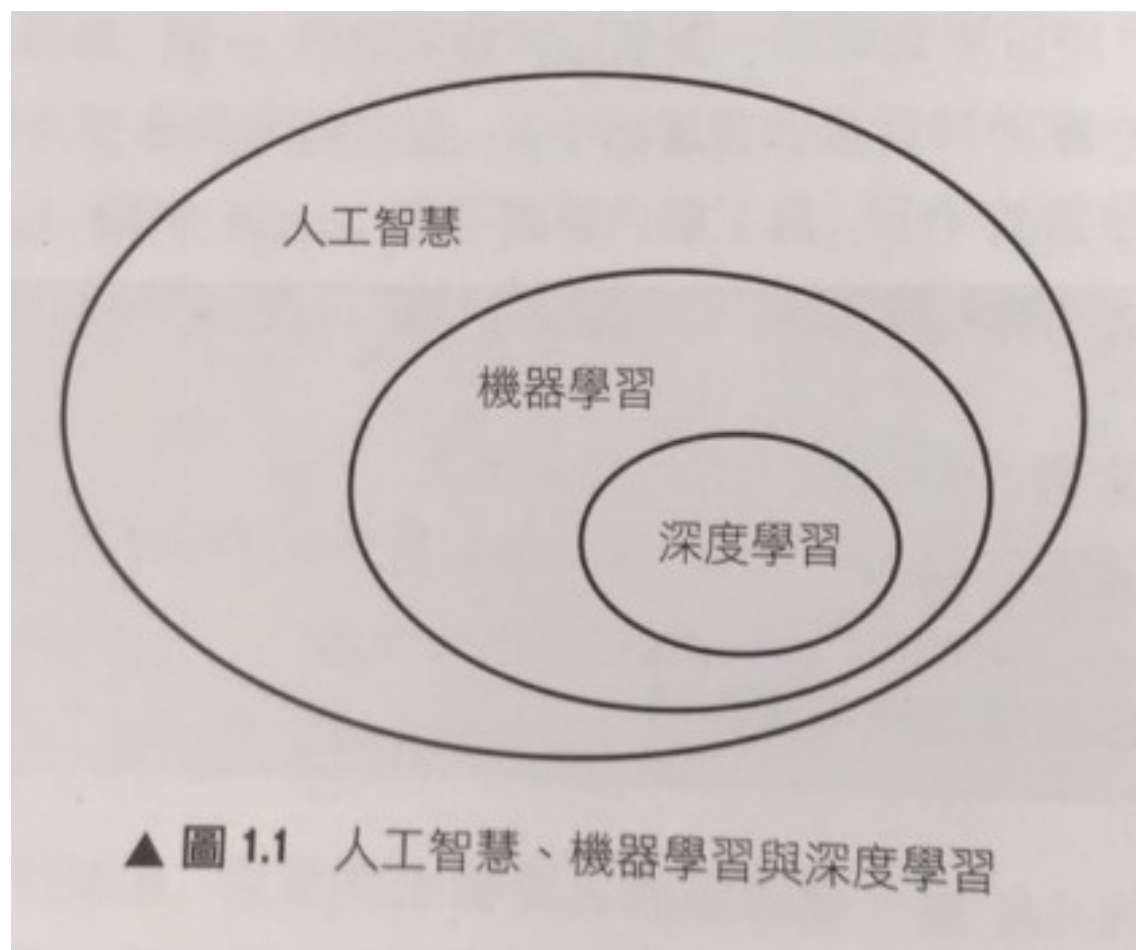
## 關於作者



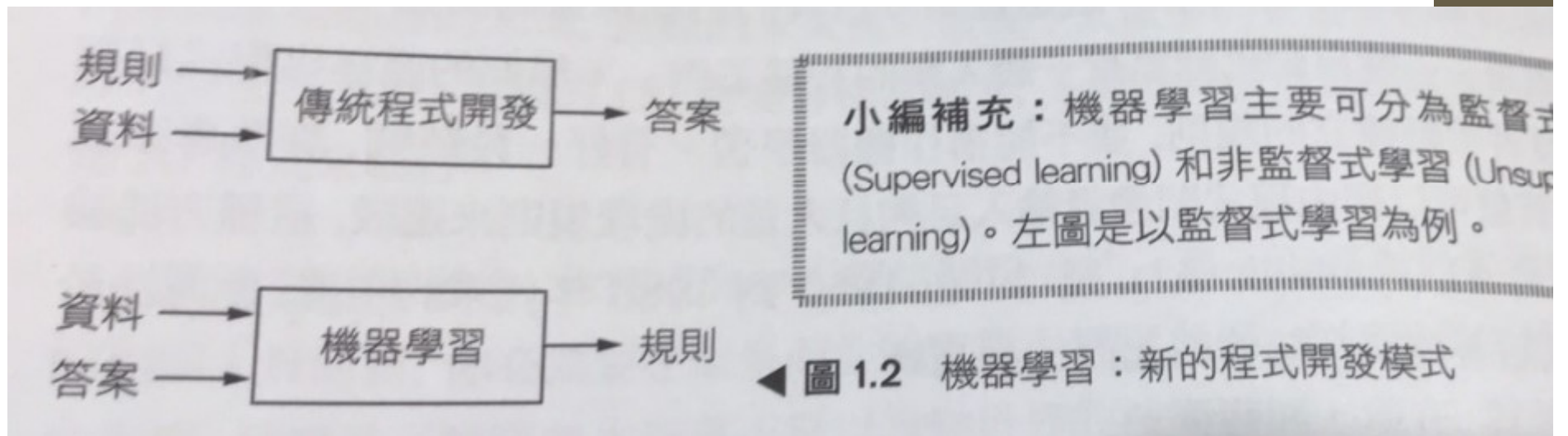
**François Chollet**

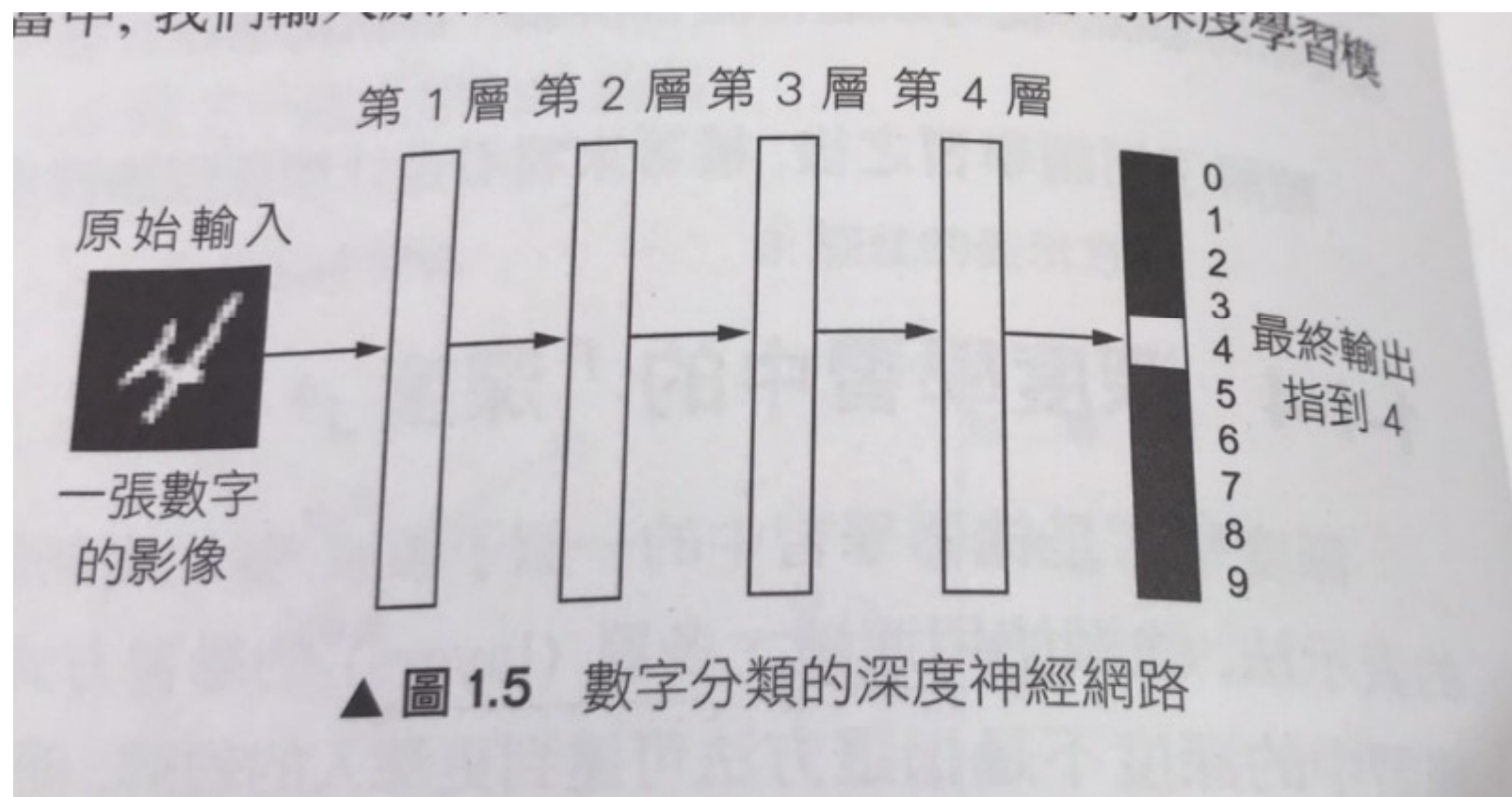
作者為 Keras 之父, 也是 TensorFlow 機器學習框架的貢獻者。作者目前任職於 Google 深度學習小組, 持續投入

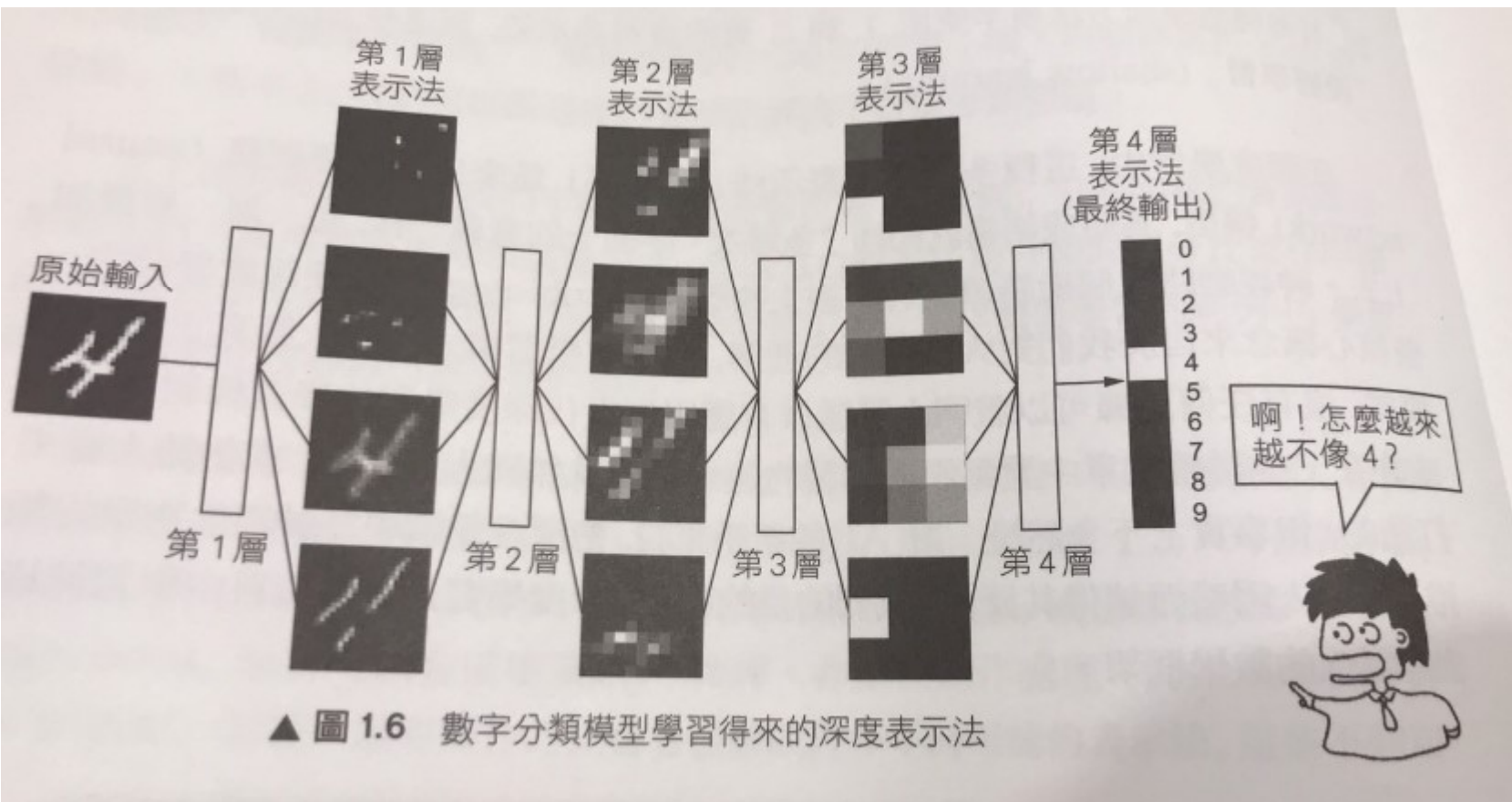
# 深度學習vs人工智慧



# 傳統方法 vs 機器學習

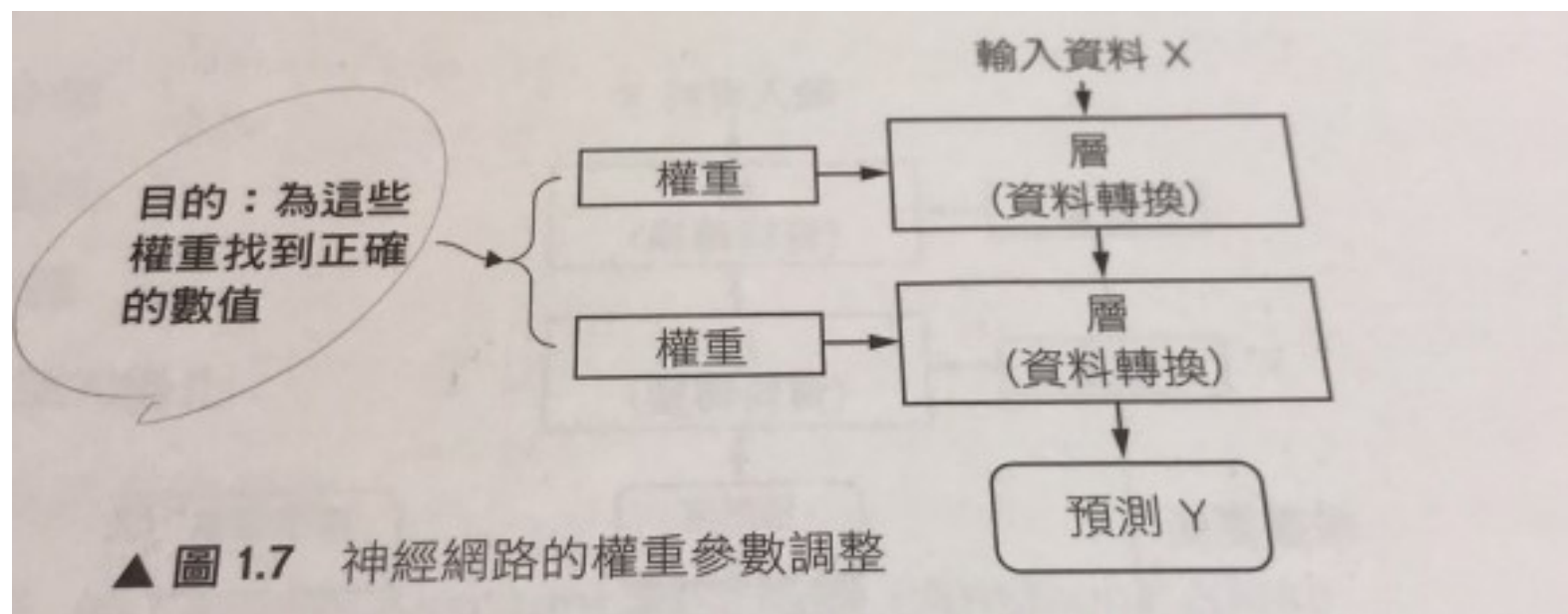






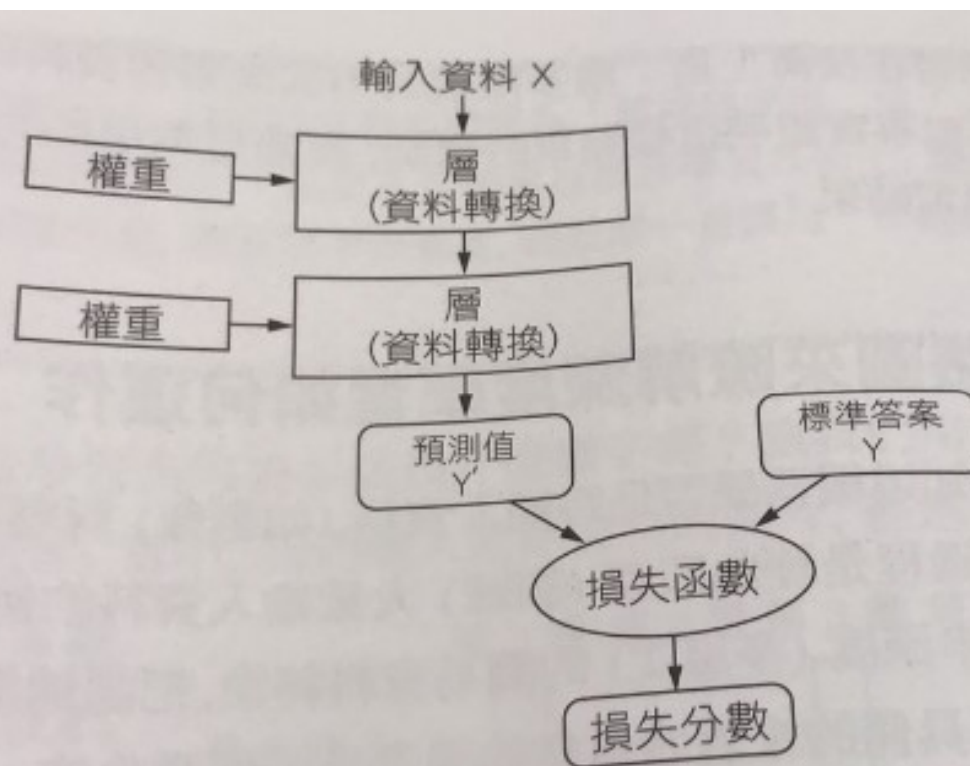
▲ 圖 1.6 數字分類模型學習得來的深度表示法

# 神經網路的目的



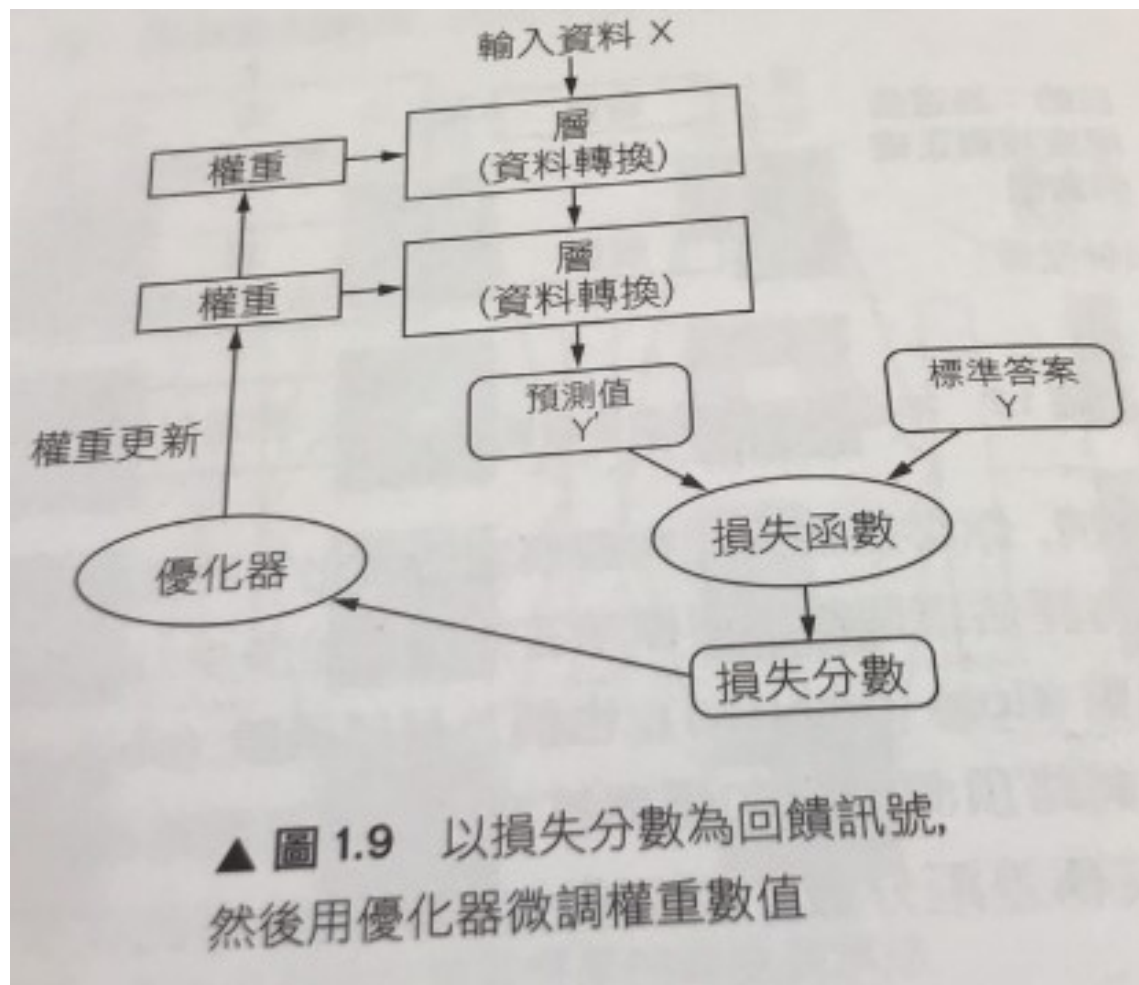


# 損失函數



▲ 圖 1.8 損失函數取得預測值和標準答案之間的差距, 測量神經網路輸出的品質

# 優化器



# NN普及之前

## 1-2-3 Kernel methods 與 SVM

### Kernel methods 與 SVM

正當神經網路在 1990 年代開始贏得研究人員的尊重時，一種新機器學習方法的崛起，又迅速將神經網路踢回被遺忘角落，這方法就是 Kernel

# SVM

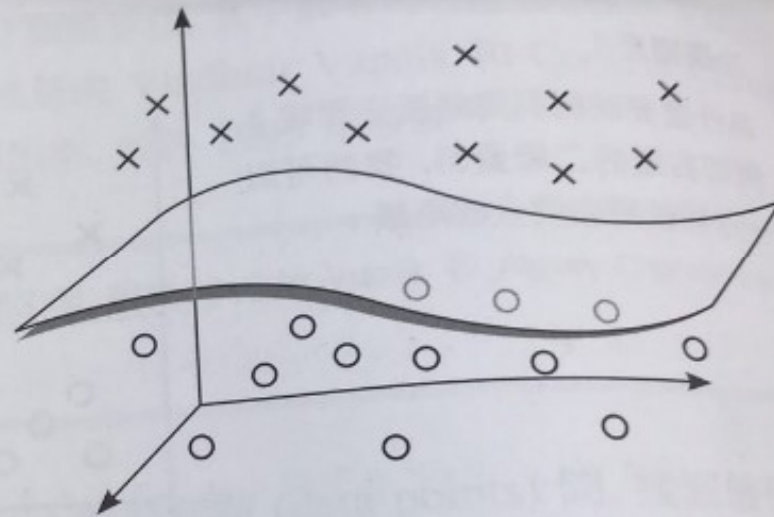
SVM 的目標是在兩種類別 (class) 的資料點 (data points) 間，找到最佳決策邊界 (decision boundaries) (見圖 1.10)。最佳決策邊界就是將你的訓練資料區隔開來的最佳曲面，曲面兩邊分別對應到不同的類別。一旦這樣的決策邊界找到之後，要分類新的資料點，你只需檢查這些新的資料點落在決策邊界的哪一側，就能知道他們是屬於哪一類了。(編註：例如在二維座標中，決策邊界如果是  $x=0$  這條直線，那機器只要辨認資料的  $x$  值是  $> 0$  或  $< 0$  就可以辨認出資料是哪一類了，就像圖 1.4 的分類問題。)



◀ 圖 1.10 決策邊界

# SVM

第二步：在找超曲面時，超曲面要位於兩類資料點之間的中線，如右圖：



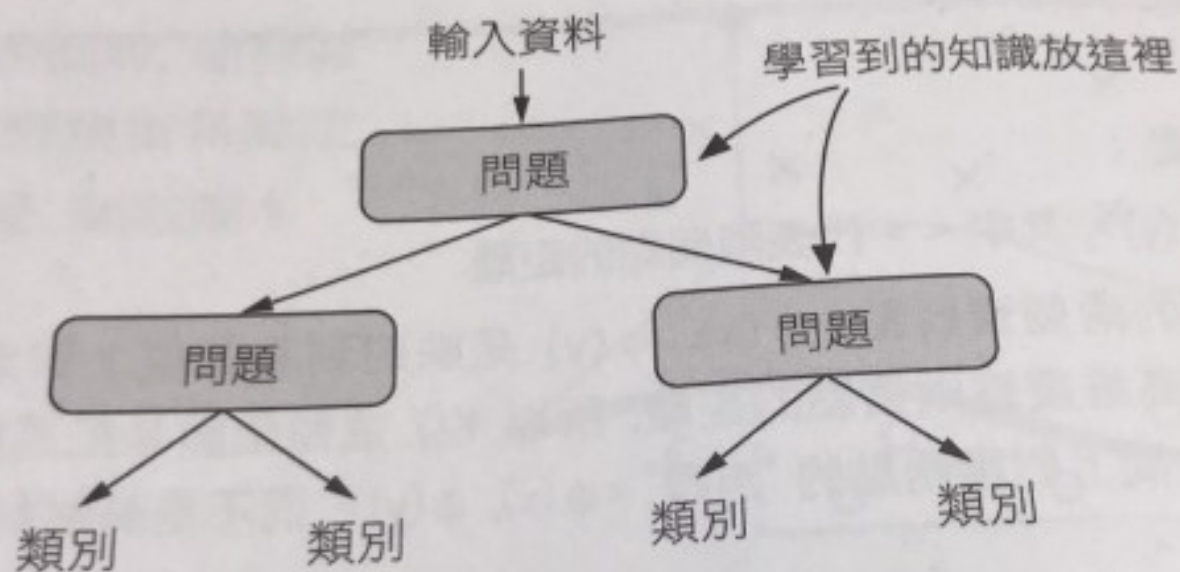
這步驟稱為**最大化邊界** (maximizing the margin), 也就是讓曲面分別和兩類資料點保有最大距離, 這使得當新的資料點被填入空間時 (要分類時), 決策邊界

# 決策樹

## 1-2-4 決策樹、隨機森林和梯度提升機器

### 決策樹 Decision tree

決策樹是類似流程圖的結構，可讓你對輸入資料進行分類或預測輸入資料的輸出值（見圖 1.11）。決策樹很容易以視覺化方式呈現並進行解釋。決策樹是一種從資料中學習的機器學習方法，它在 2000 年代開始獲得研究人員的關注，不過在 2010 年之前，人們還是傾向使用 kernel methods。



▲ 圖 1.11 決策樹：就是學習"問問題"，它把學到的參數用來更新問題，最終建立一個很會經由問問題來把資料做分類的機器。例如，問題可能是"資料中的第 2 個係數是否大於 3.5？"



## 隨機森林 Random Forest

隨機森林演算法 (Random Forest algorithm), 是強大且實用的決策樹學習方法, 它整合了大量專門的決策樹, 然後組合成輸出值。隨機森林適用於各式各樣的問題, 幾乎可以說是所有淺層機器學習中第二好的演算法。當熱門的機器學習競賽網站 Kaggle (<http://kaggle.com>) 在 2010 年開始營運後, 隨機森林很快就成為該平台上的熱門方法, 直到 2014 年才逐漸由梯度提昇機器 (gradient boosting machines) 接手。編註: 有關決策樹 (Decision tree) 和隨機森林 (Random Forest) 可到旗標網站下載 Bonus 補充資料喔!



# 神經網路的抬頭

2011 年，來自 IDSIA 的 Dan Ciresan 開始以 GPU 訓練的深度神經網路贏得學術影像分類競賽，這是現代深度學習第一次實際成功的案例。來到 2012 年，隨著 Hinton 團隊進入年度大規模影像分類 ImageNet 挑戰，神經網路進入分水嶺的時刻。當時 ImageNet 的挑戰非常困難，主要的內容是對 140 萬張影像進行訓練後，將高解析度的彩色影像分類到 1,000 個不同類別。在 2011 年時，以傳統電腦視覺方法獲勝的模型，前五名準確度僅為 74.3%。然而在 2012 年，以 Alex Krizhevsky 為首，由 Geoffrey Hinton 指導的團隊，能夠實現前五名的準確率達 83.6%，這是一項重大突破。從那時候開始，ImageNet 挑戰每年都是由卷積神經網路贏得勝利。到 2015 年，獲勝者已經達到 96.4% 的準確率，ImageNet 上的分類工作被認為是一個已完全解決的問題。

自 2012 年以來，卷積神經網路 (ConvNets) 已成為所有電腦視覺問題的首選演算法，更普遍地說，卷積神經網路可運作於所有感知任務上 (視覺、聽覺等)。在 2015、2016 年主要的電腦視覺研討會上，皆涉及卷積神經網路的論文報告。於此同時，深度學習也被用在許多其他類型的問題上，例如自然語言處理 (NLP)，深度學習已在廣泛的應用中完全取代了 SVM 和決策樹，幾年來，歐洲核子研究組織 (European Organization for Nuclear Research, CERN) 使用以決策樹為基礎的方法來分析大型強子對撞機 (Large Hadron Collider, LHC) 的 ATLAS 探測器的粒子資料，但是 CERN 需要性能更好、更適合用於大型資料集上的訓練方法，最終轉向採用以 Keras 為基礎的深度神經網路。

# 起手式

- 定義訓練資料
- 定義神經網絡模型
- 訓練模型
- 測試模型



# Demo1

- MNIST資料集
- 算是類神經網路的hello world
- 親手建立模組，使用colab實作



# Demo1

先引入需要的module

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

# Demo1

## 載入資料

```
#- 備資料
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

#- 整理資料
train_images = train_images.reshape((60000, 28 * 28))    #reshape 是 NumPy 陣列的 method
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

# Demo1

## 建立模型

```
#- 建立layer, model
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

# Demo1

## 訓練模型

```
#- 準備標籤
```

```
train_labels = to_categorical(train_labels)
```

```
test_labels = to_categorical(test_labels)
```

```
#- training
```

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```



# Demo1

## 測試模型(使用模型預測)

```
#- testing
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
Epoch 1/5
469/469 [=====] - 5s 10ms/step - loss: 0.2582 - accuracy: 0.9256
Epoch 2/5
469/469 [=====] - 5s 10ms/step - loss: 0.1033 - accuracy: 0.9692
Epoch 3/5
469/469 [=====] - 5s 10ms/step - loss: 0.0692 - accuracy: 0.9796
Epoch 4/5
469/469 [=====] - 5s 10ms/step - loss: 0.0494 - accuracy: 0.9849
Epoch 5/5
469/469 [=====] - 5s 10ms/step - loss: 0.0367 - accuracy: 0.9886
313/313 [=====] - 1s 3ms/step - loss: 0.0742 - accuracy: 0.9801
test_acc: 0.9800999760627747
```