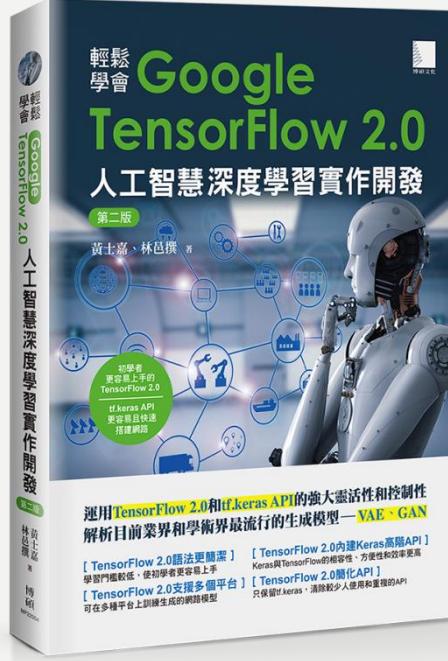


# **GOOGLE TENSORFLOW 2.0**

## **人工智能深度學習**

謝 倩 達

jumbokh@gmail.com



# TensorFlow 2.0

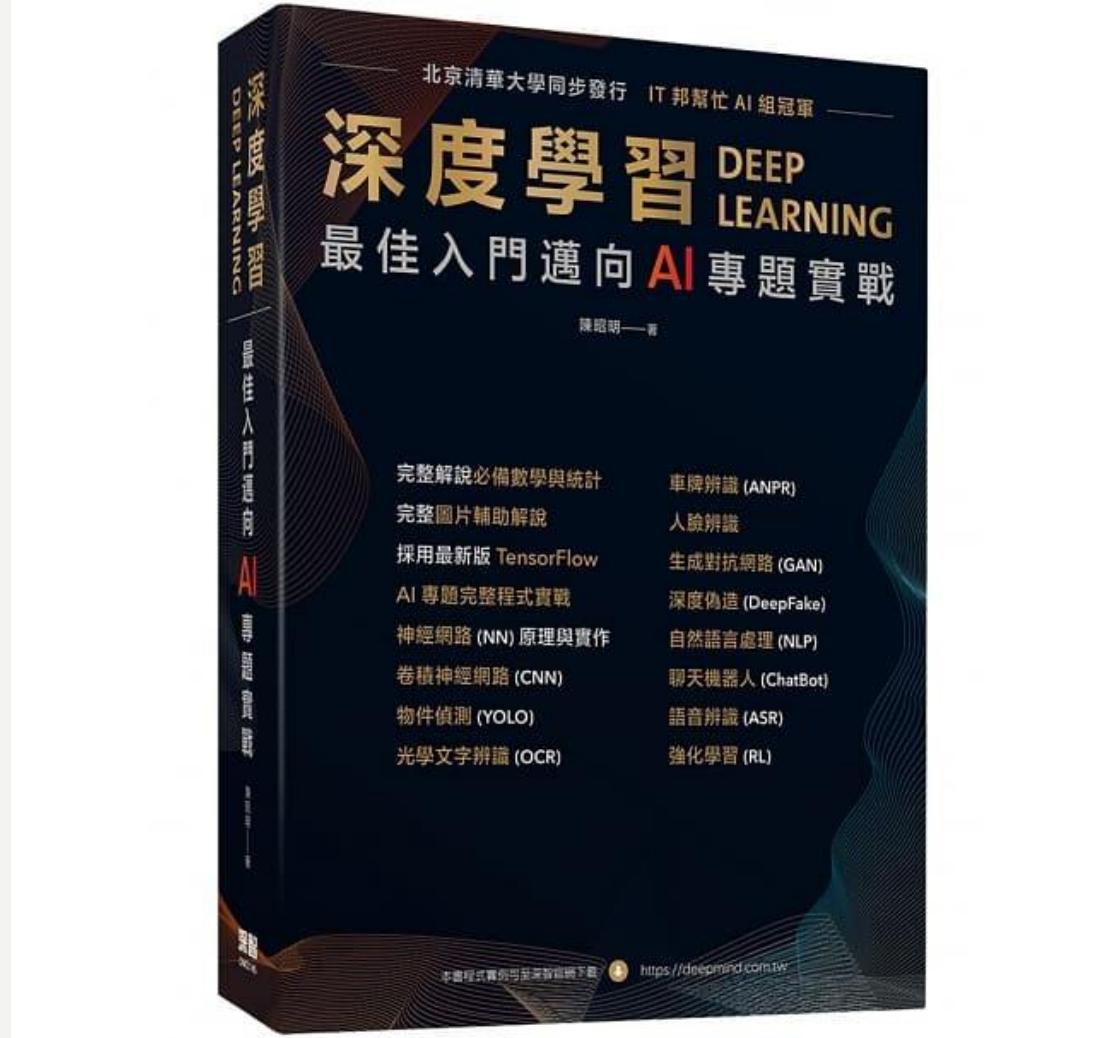
謝坤達

課本：輕鬆學會Google  
TensorFlow 2.0人工智能  
深度學習實作開發(第  
二版)

<https://github.com/taipeitechmmslab/MMSLAB-TF2>

博碩文化出版發行





Program: [https://github.com/mc6666/DL\\_Book](https://github.com/mc6666/DL_Book)

Book: <http://gg.gg/w4net>

# 第一章 TensorFlow 2.0



## 課前指引

- 認識深度學習
- 說明TensorFlow 2.0的重要更新
- 認識Eager Execution模式
- 學習TensorFlow的基本運算
- 學習使用tf.keras搭建網路模型
- 概述tf.data的用途以及基本操作

# 深度學習導論

- 人工智能已歷經三波浪潮，這一波是否又將進入寒冬？
- 人工智能、資料科學、資料探勘、機器學習、深度學習到底有何關聯？
- 機器學習開發流程與一般應用系統開發有何差異？
- 深度學習的學習路徑為何？建議從哪裡開始？
- 為什麼要先學習數學與統計，才能學好深度學習？
- 先學哪一種深度學習框架比較好？TensorFlow 或 PyTorch？
- 如何準備開發環境？

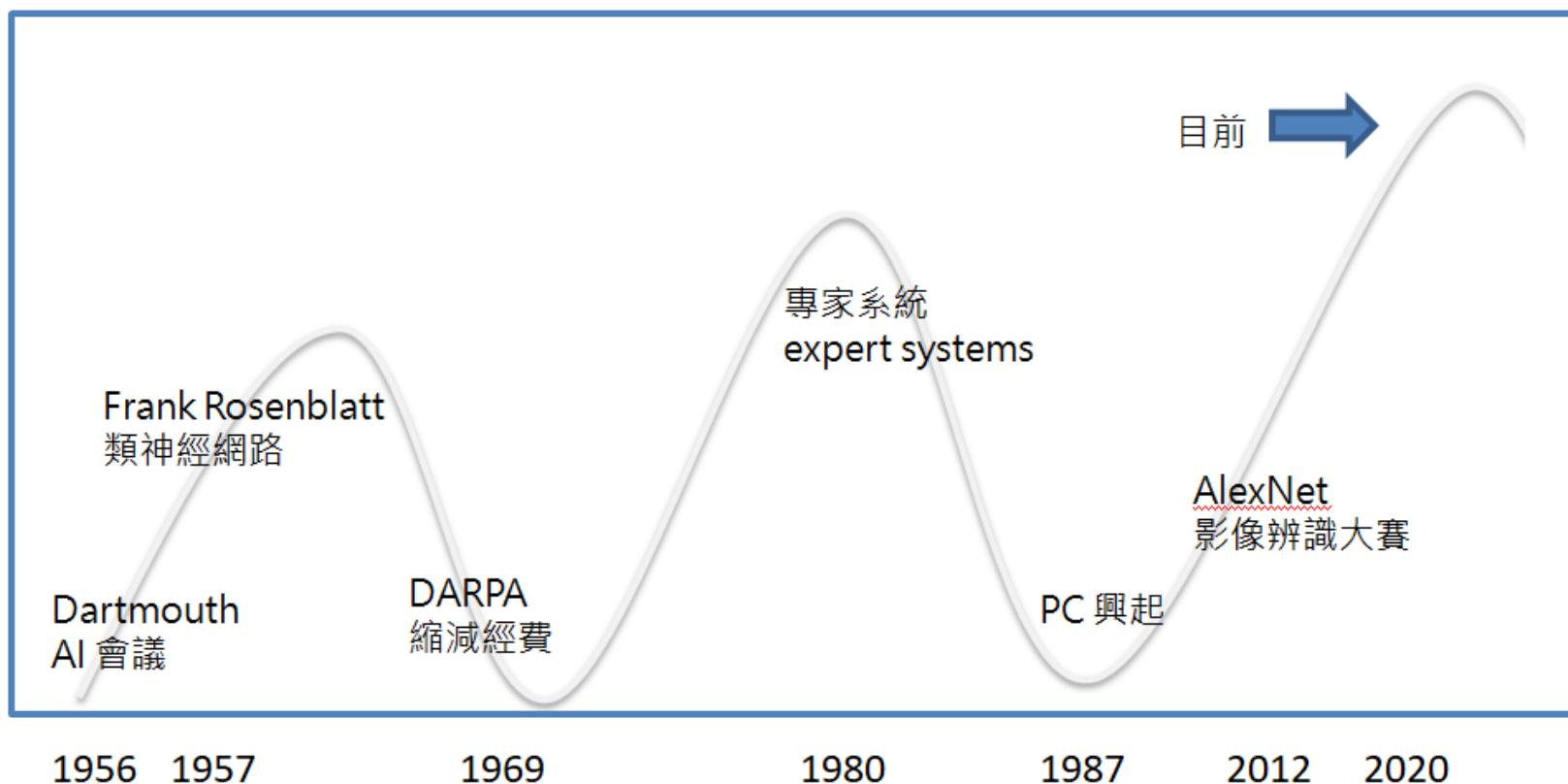
# 1.1 什麼是深度學習？



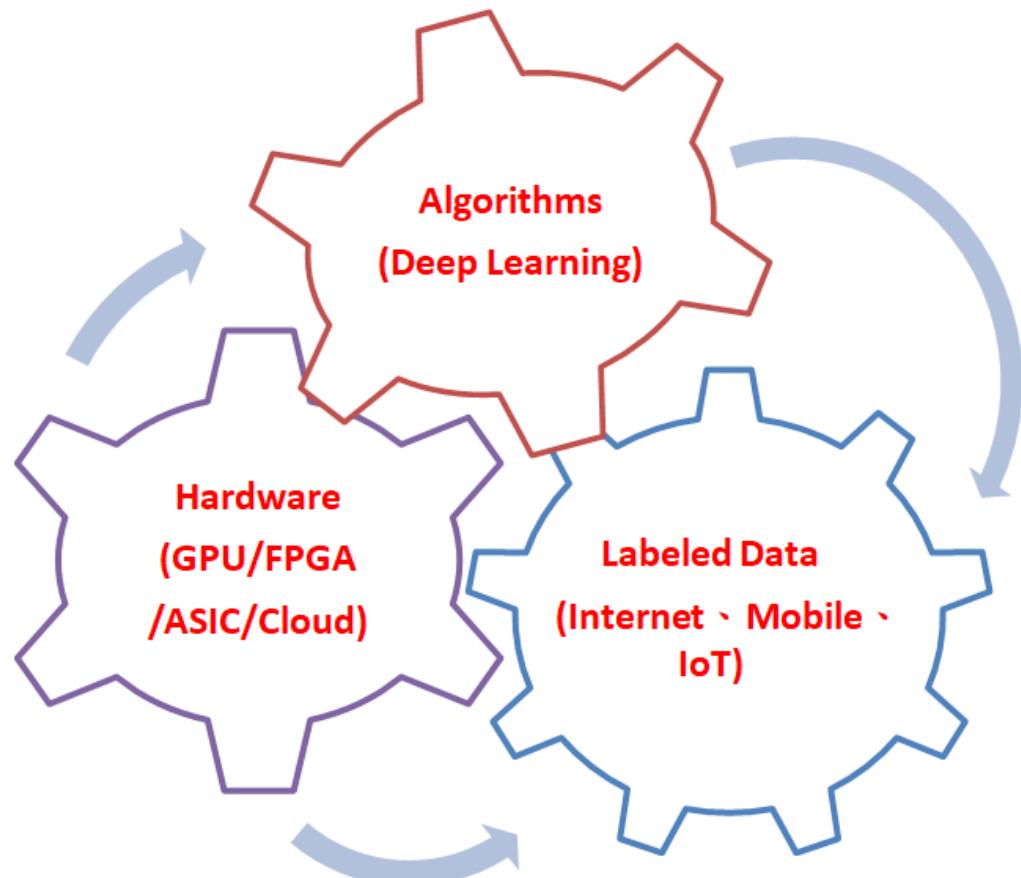
## 1.1 什麼是深度學習？

人工智能（Artificial Intelligence，簡稱為 AI）是指賦予機器思考或學習能力，主要是用來協助人類或取代重複性高、技能含量低的工作，讓人們可以專注在更有意義的事。人工智能是一個綜合的領域，其中包含了演化計算（Evolutionary Computation）、專家系統（Expert Systems）、符號人工智能（Symbolic AI）、支援向量機（Support Vector Machine）、機器學習（Machine Learning）、深度學習（Deep Learning）、增強式學習（Reinforcement Learning）等眾多領域，而深度學習又是機器學習的一個分支領域，如圖 1-1 所示。

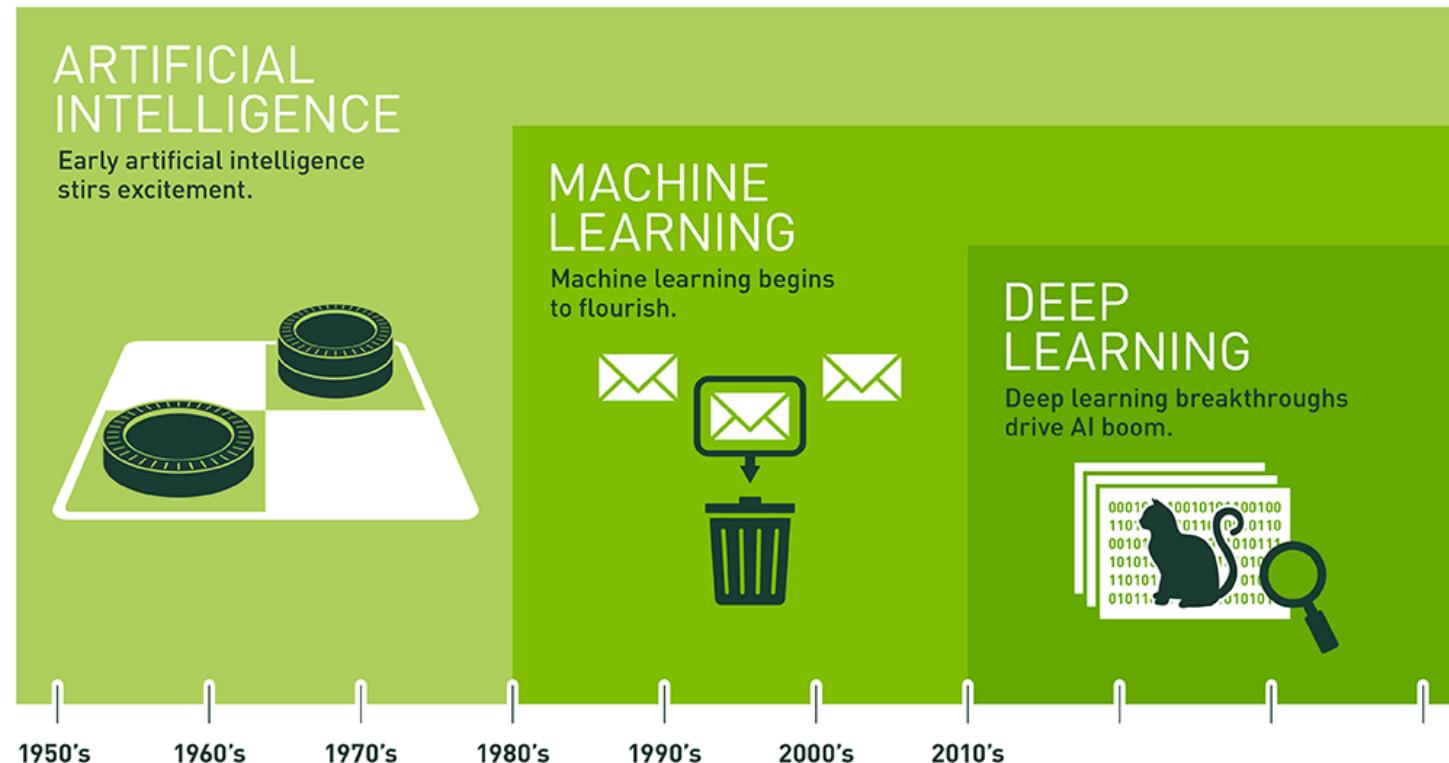
# 人工智慧的三波浪潮



# 第三波優勢

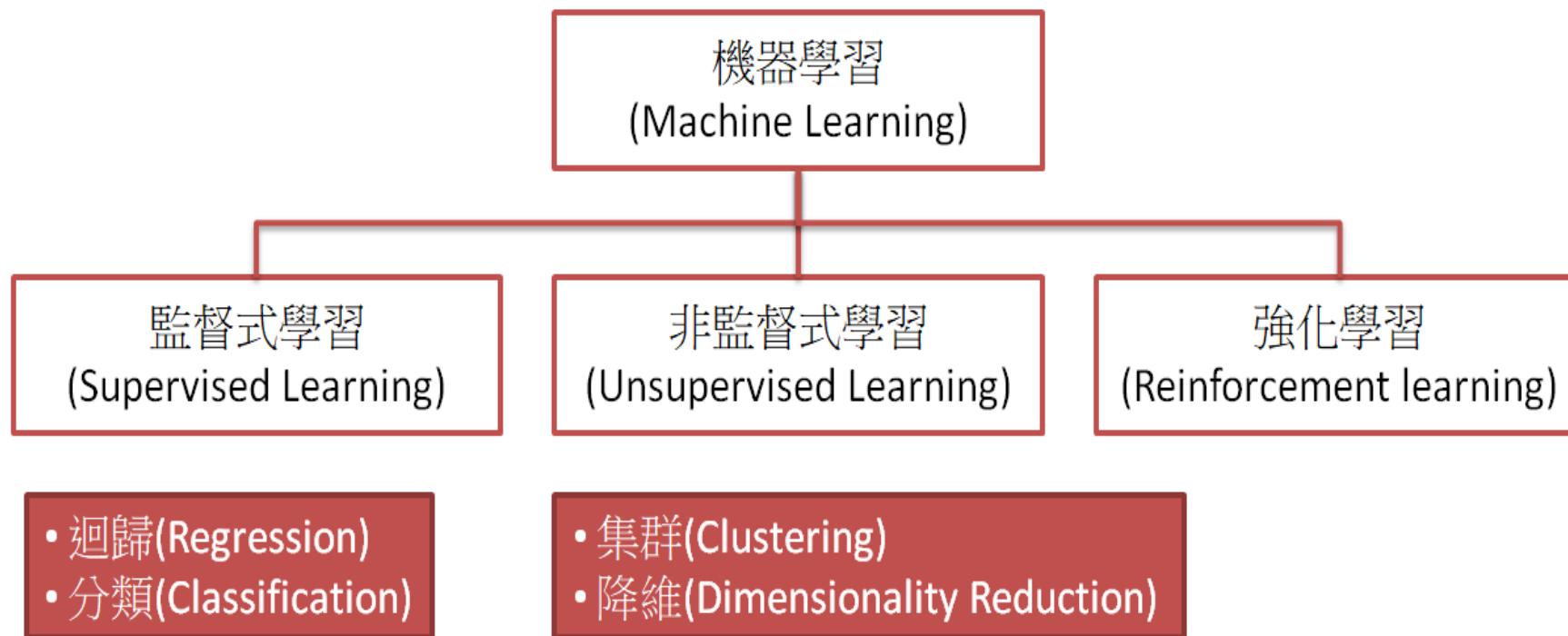


# AI的學習地圖

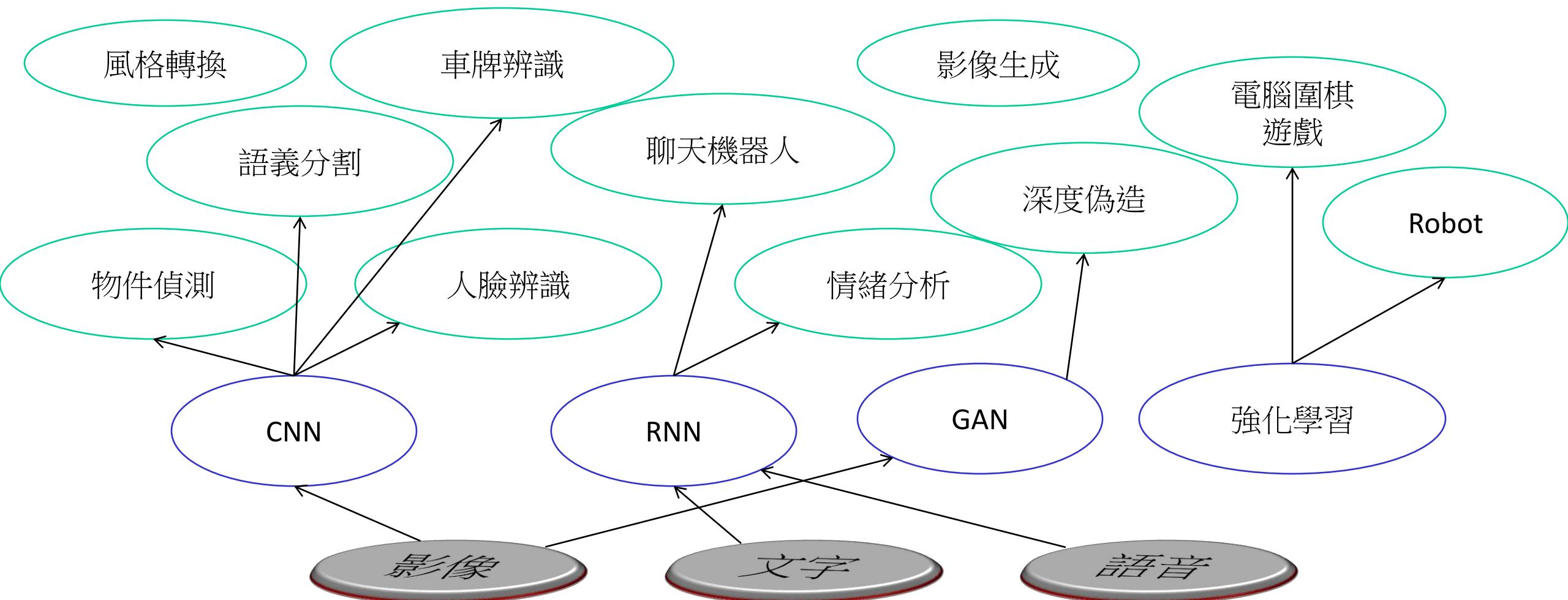


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

# 機器學習(Machine Learning) 分類



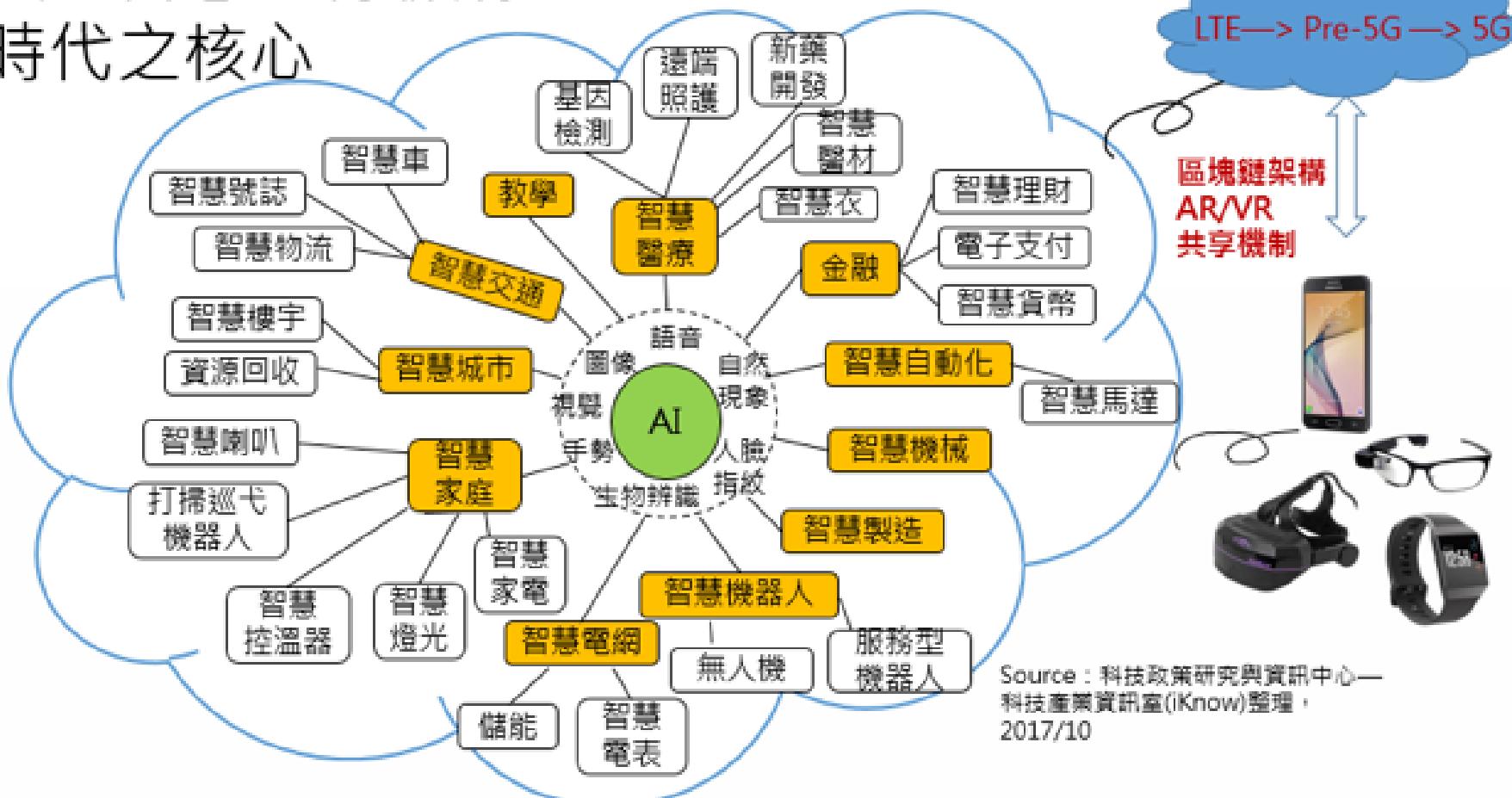
# 演算法應用



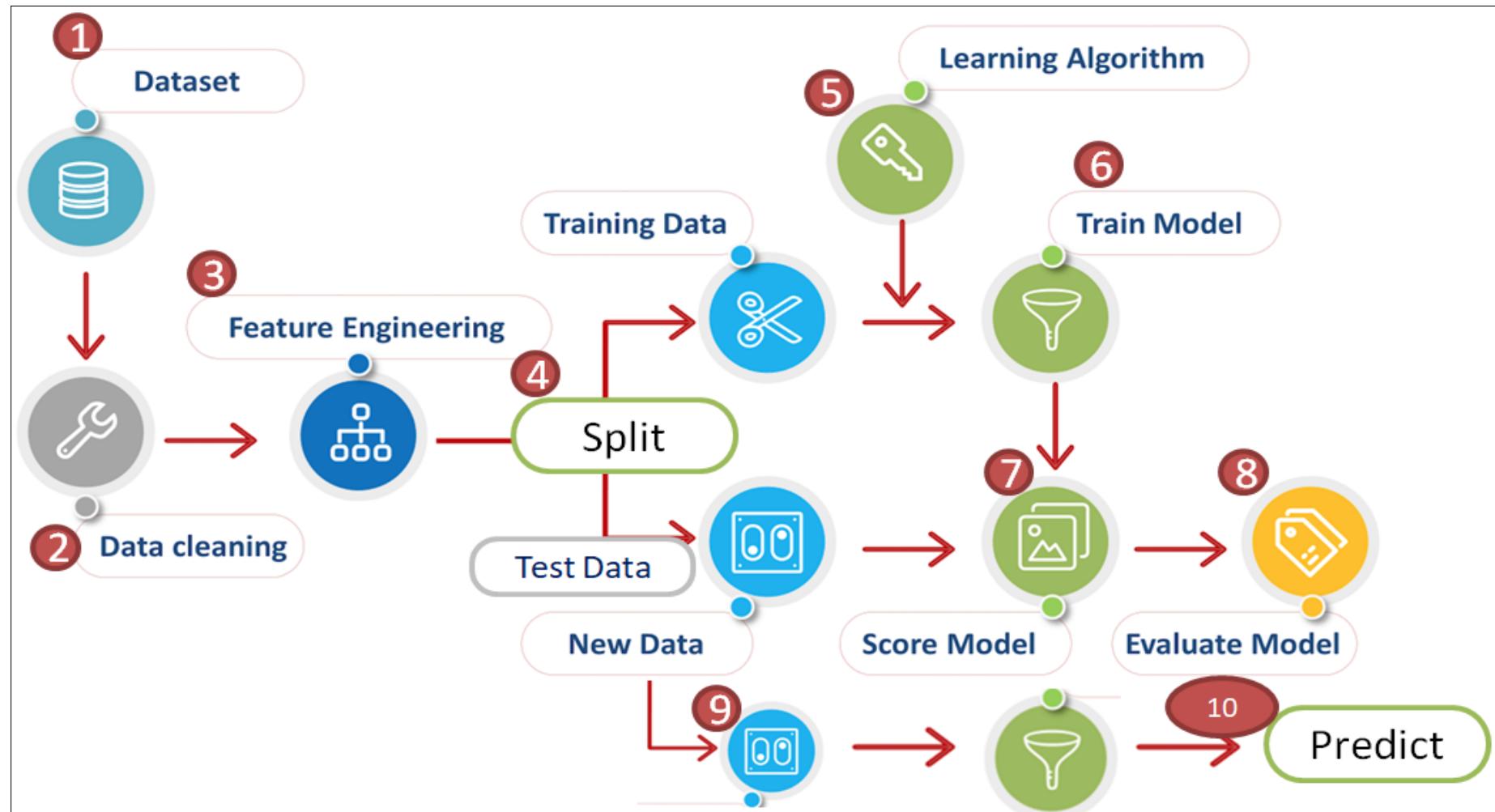
# 人工智慧應用領域



人工智慧是物聯網  
時代之核心



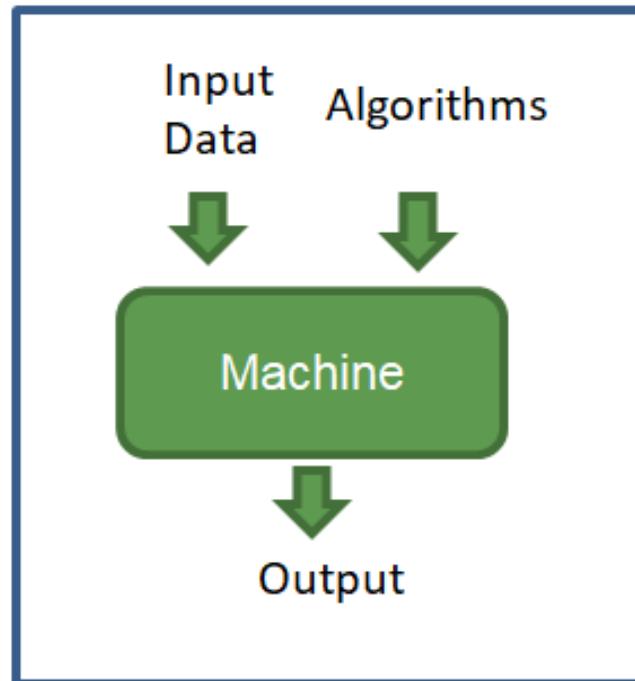
# 機器學習開發流程



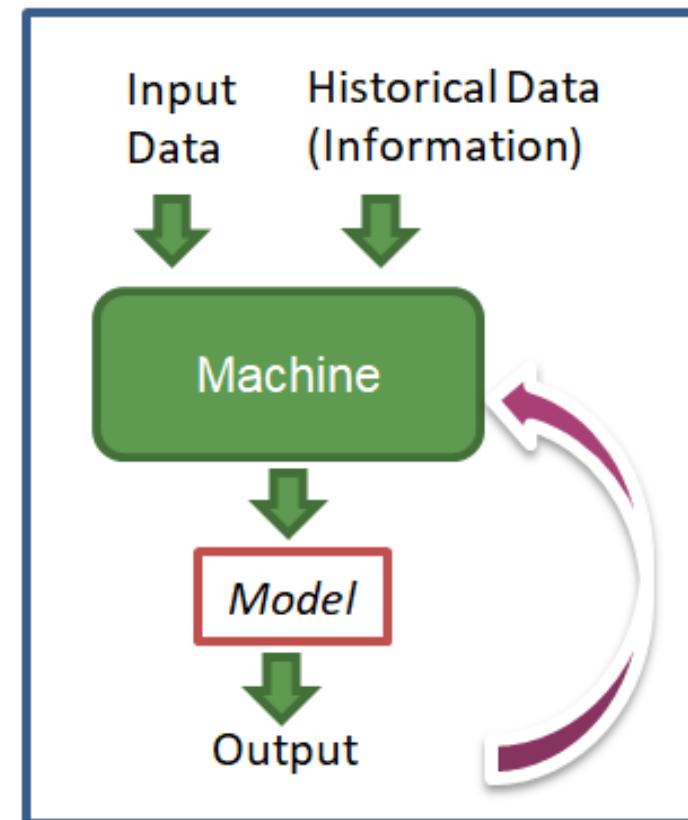
# 機器學習開發流程 VS. 一般應用系統開發

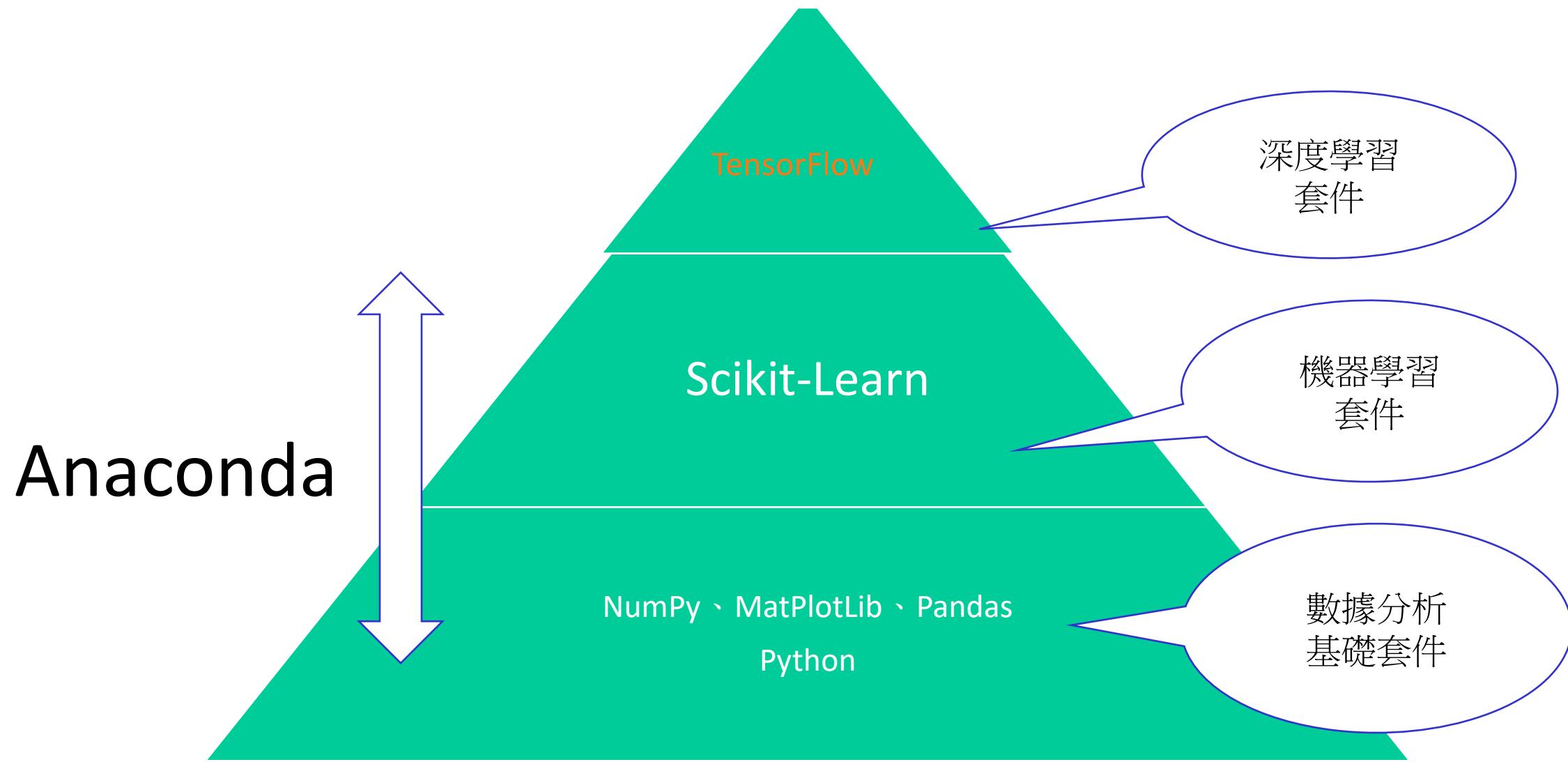


## 傳統的程式開發



## 機器學習





NVidia 顯卡

CUDA

cuDNN

Google Colaboratory

免費的GPU

# 1.1 什麼是深度學習？

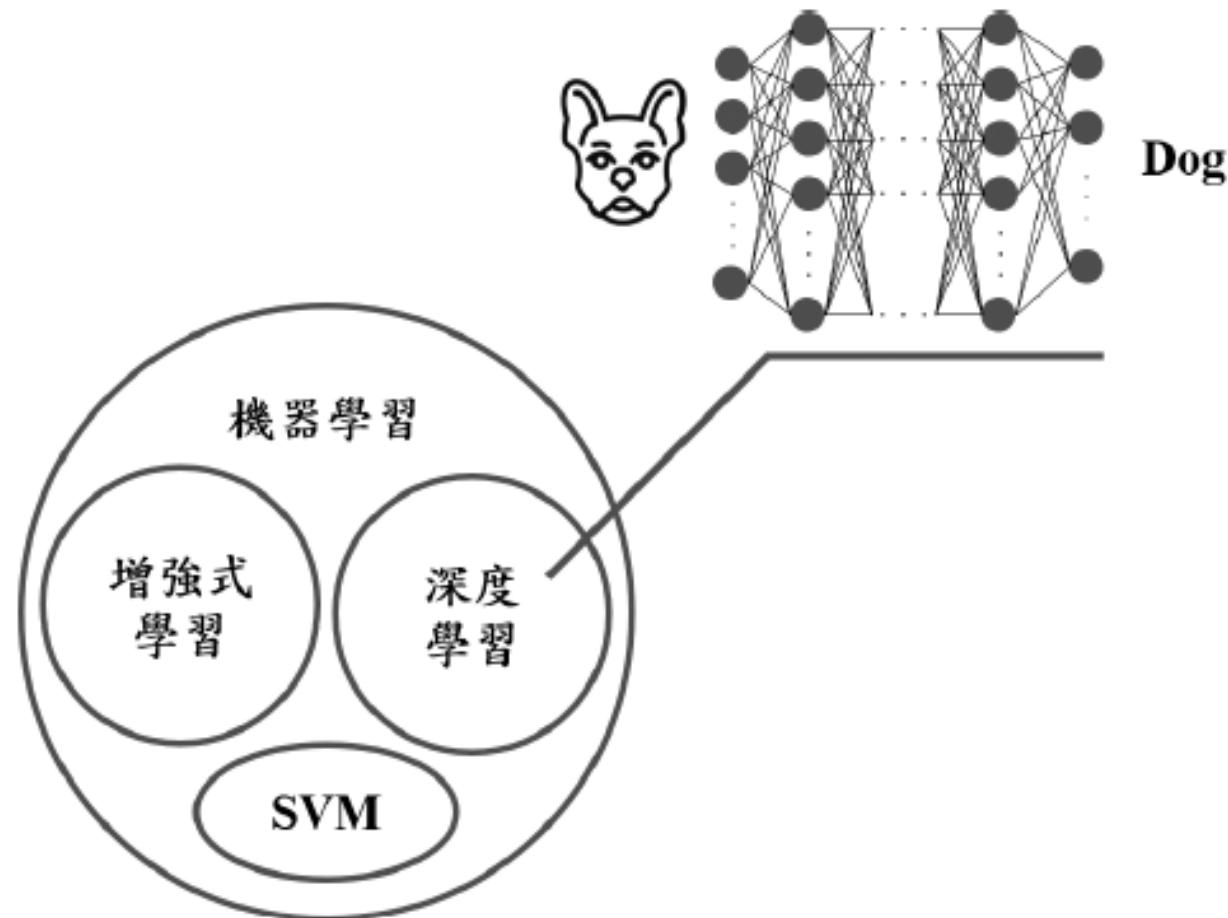
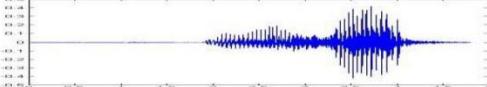


圖 1-1 深度學習是目前機器學習最熱門的領域

# Machine Learning

≈ Looking for a Function

- Speech Recognition

$f($   ) = “How are you”

- Image Recognition

$f($   ) = “Cat”

- Playing Go

$f($   ) = “5-5” (next move)

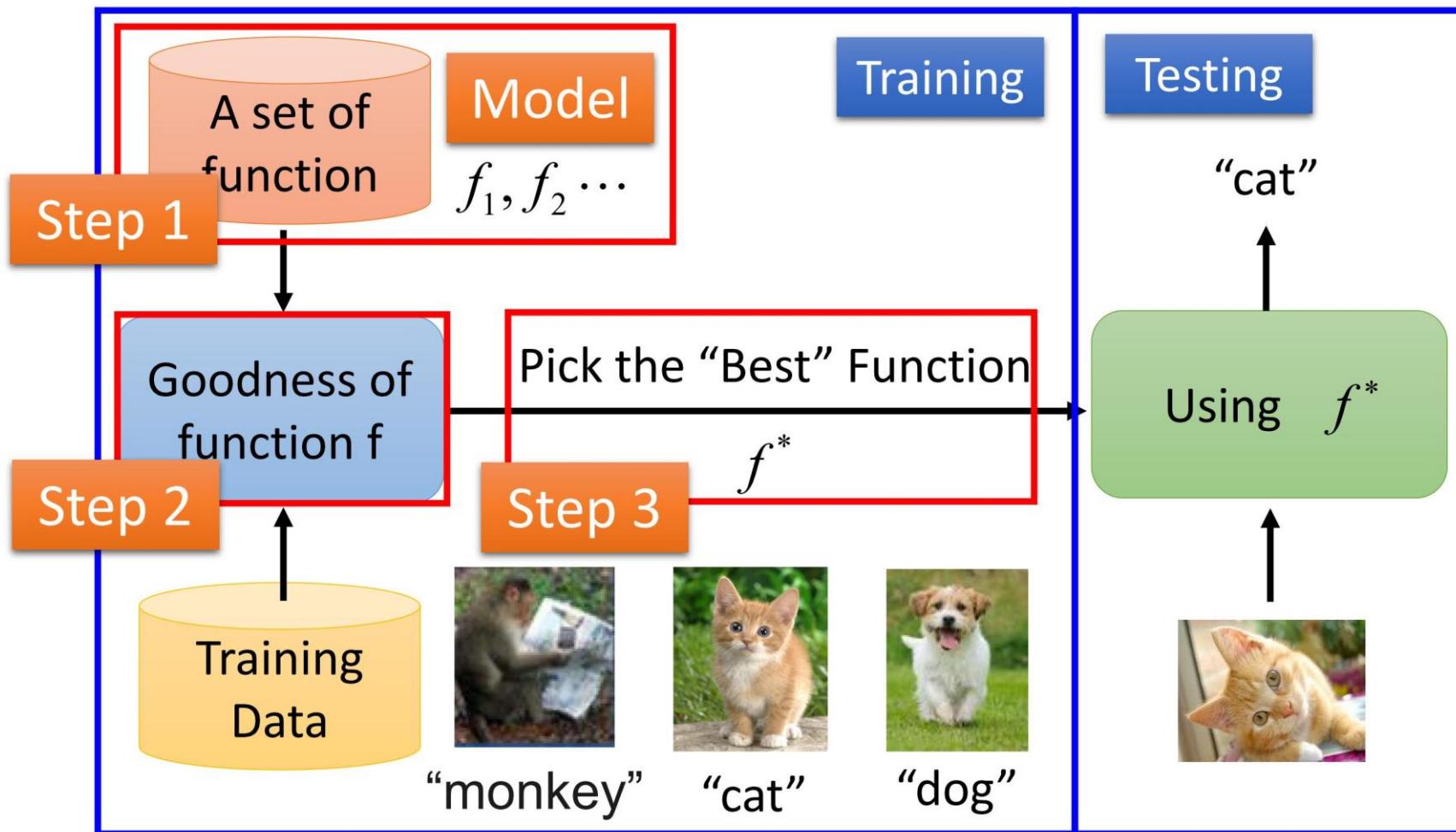
- Dialogue System

$f($  “Hi” (what the user said) = “Hello” (system response)

# Framework

Image Recognition:

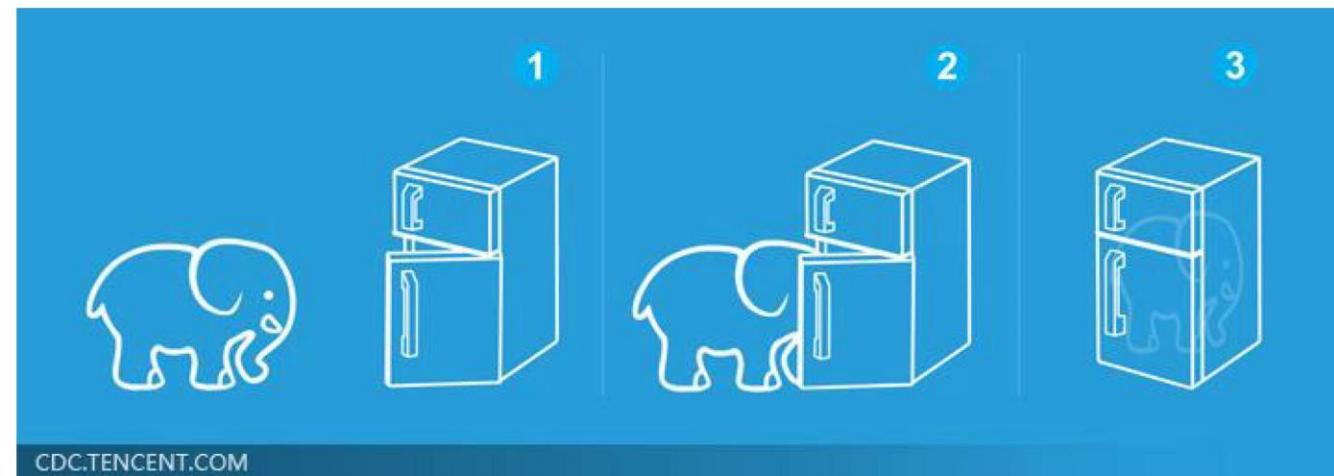
$$f(\text{cat image}) = \text{"cat"}$$



# Three Steps for Deep Learning



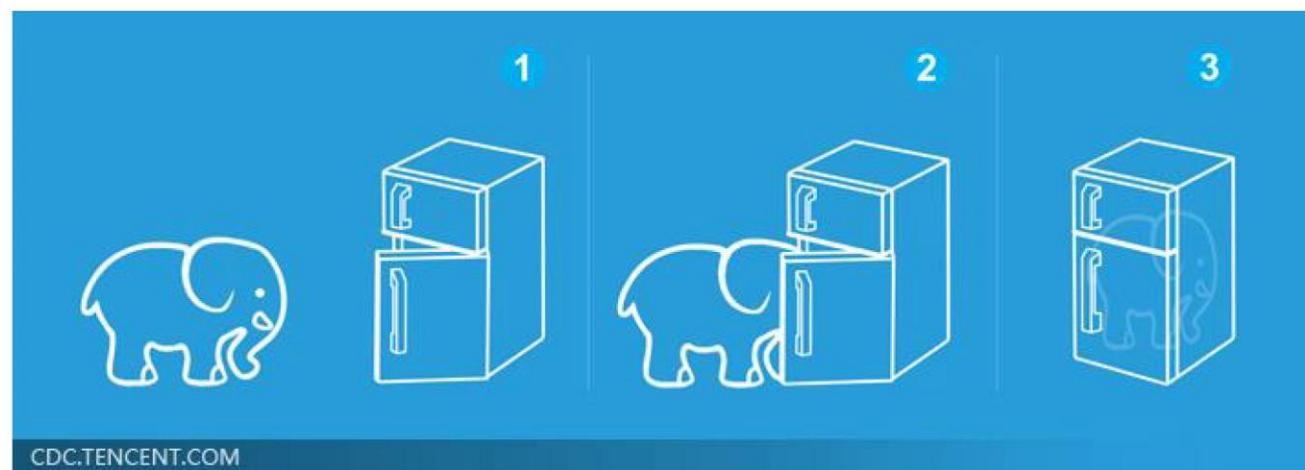
Deep Learning is so simple .....



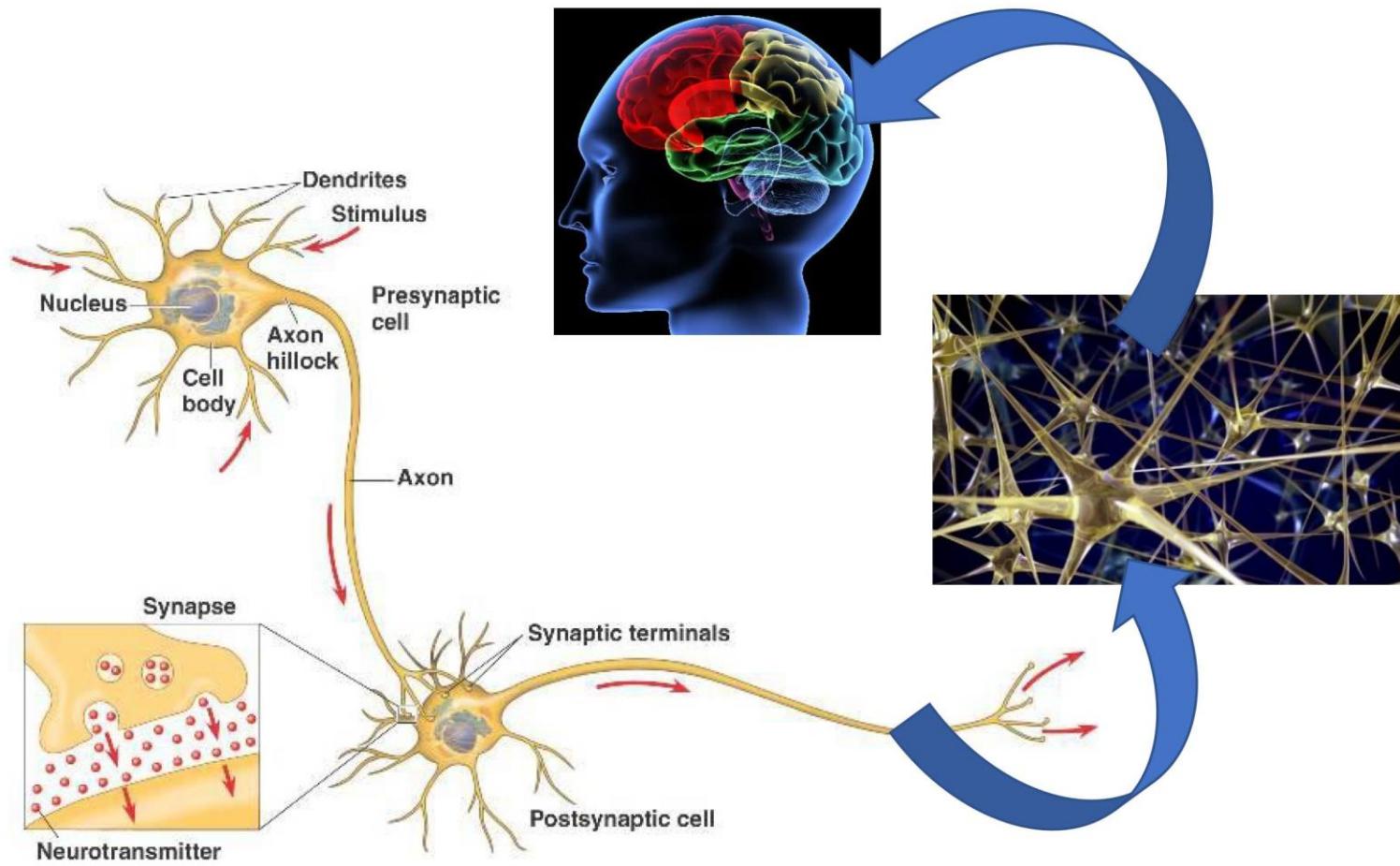
# Three Steps for Deep Learning



Deep Learning is so simple .....



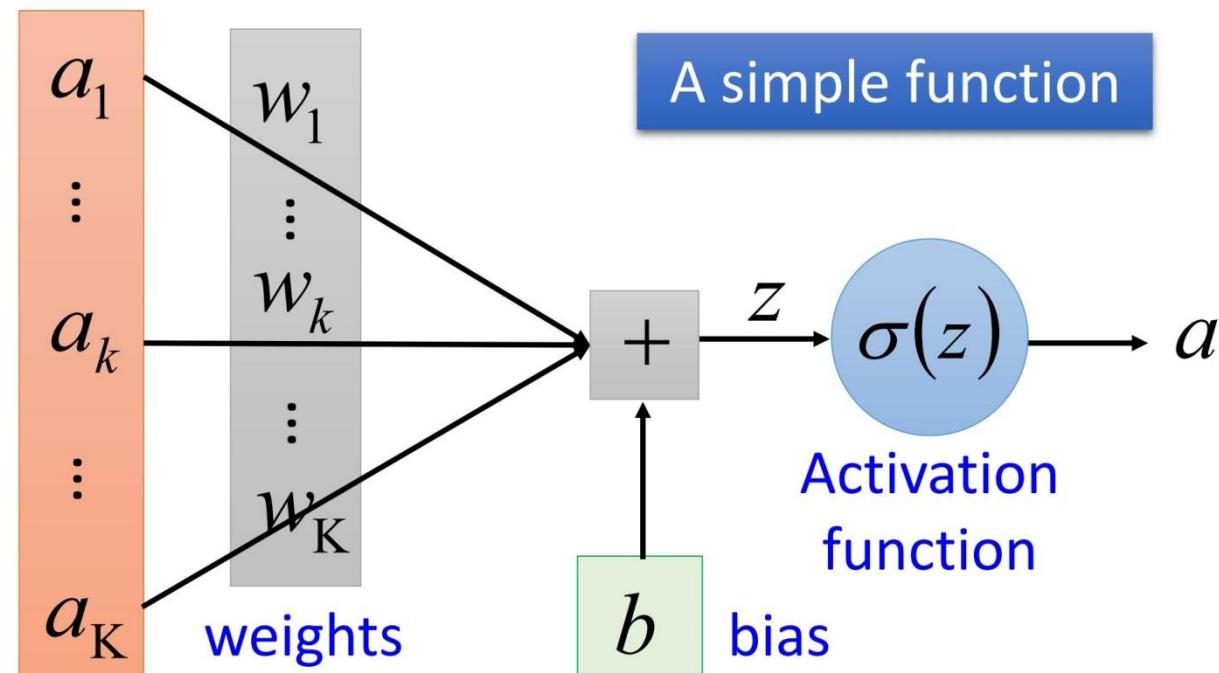
# Human Brains



# Neural Network

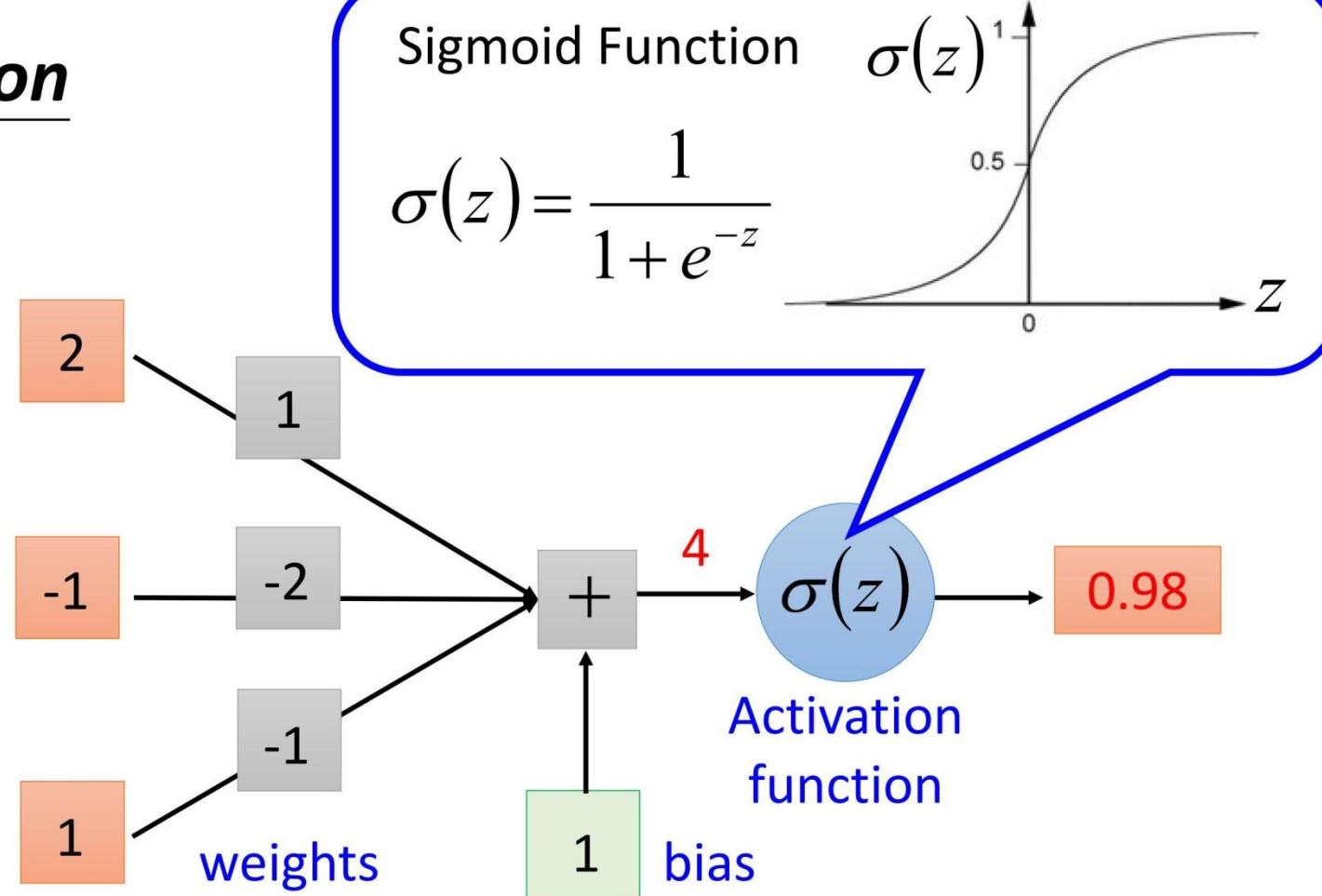
## Neuron

$$z = a_1 w_1 + \dots + a_k w_k + \dots + a_K w_K + b$$



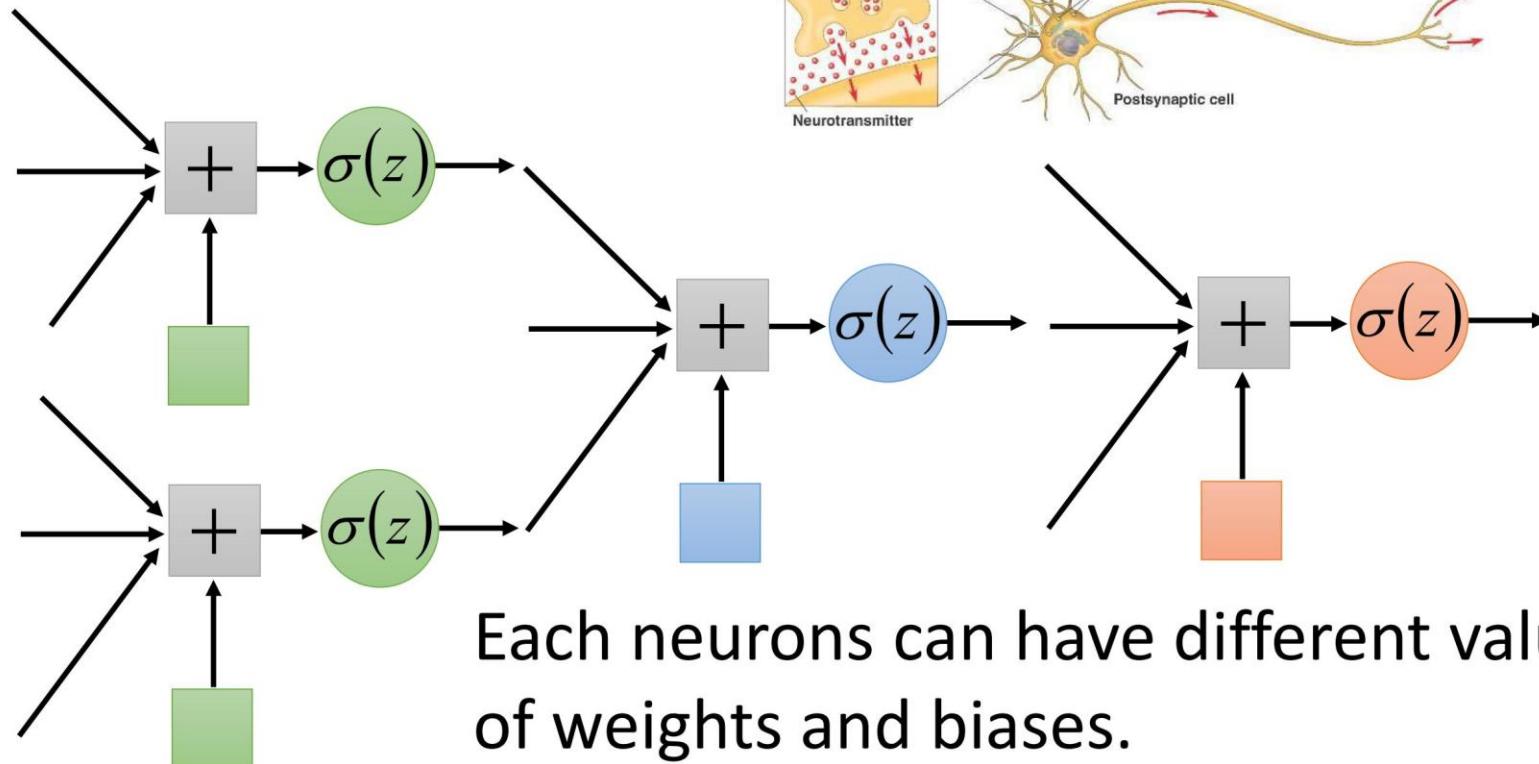
# Neural Network

## Neuron

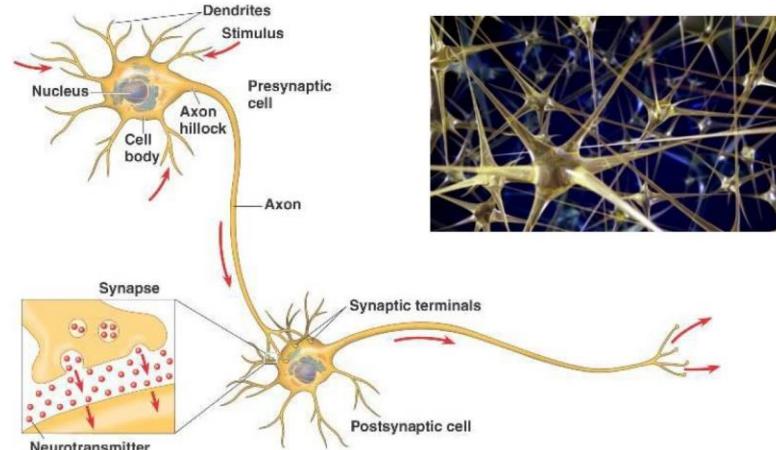


# Neural Network

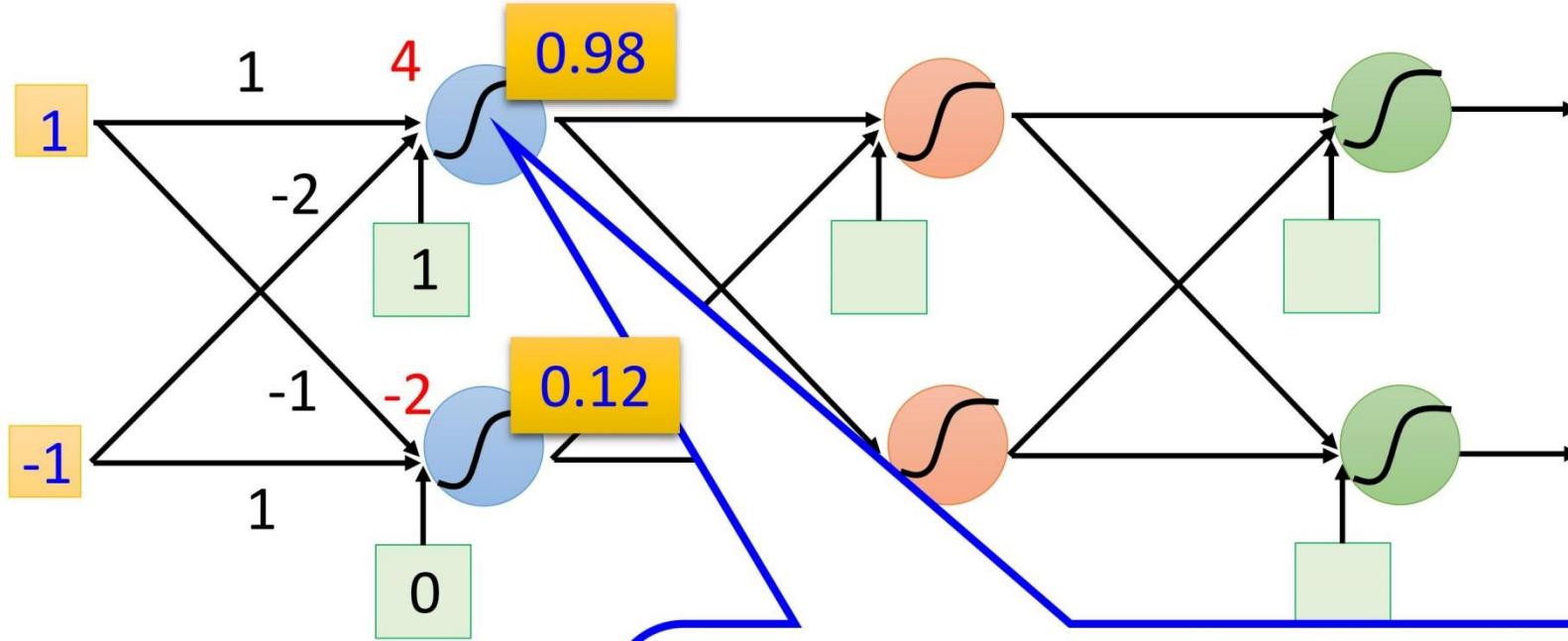
Different connections leads to different network structure



Weights and biases are network parameters  $\theta$

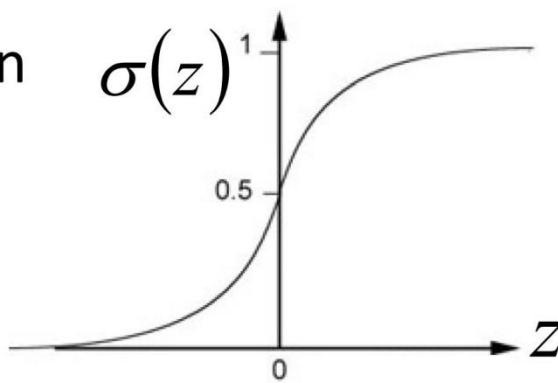


# Fully Connect Feedforward Network

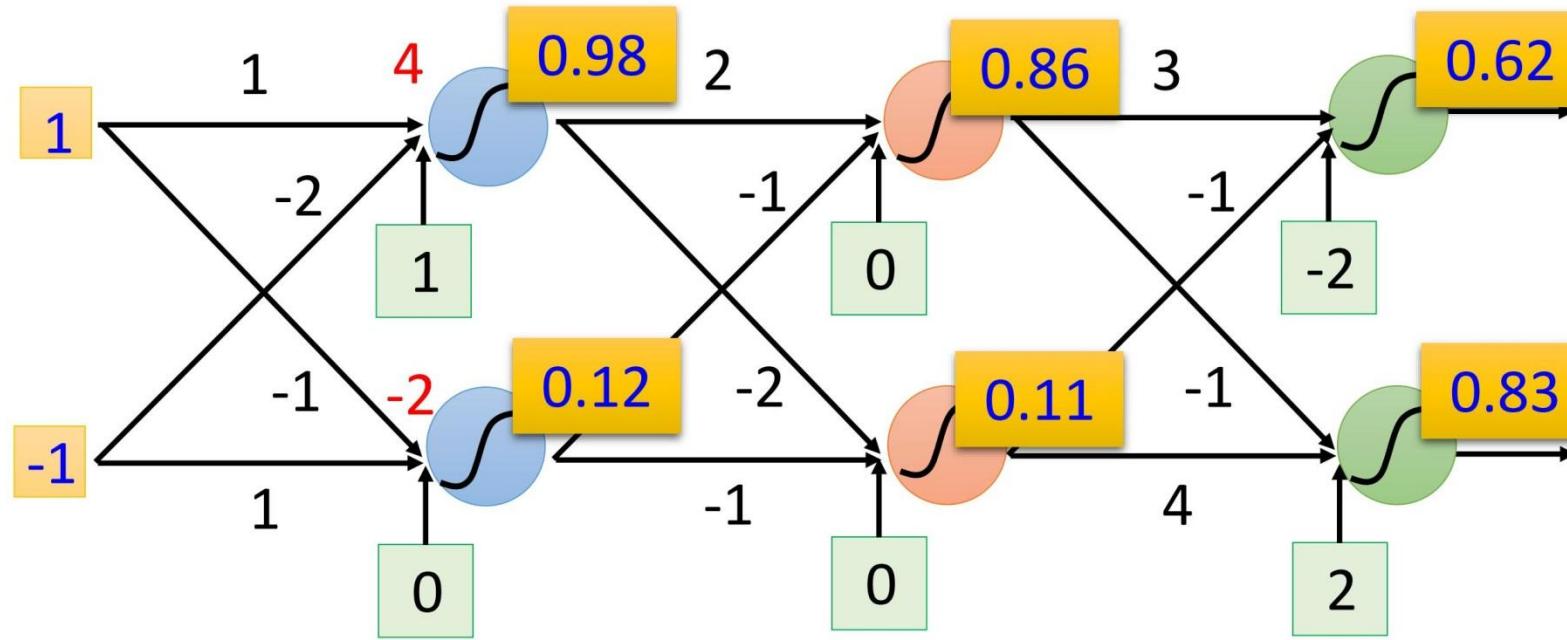


Sigmoid Function

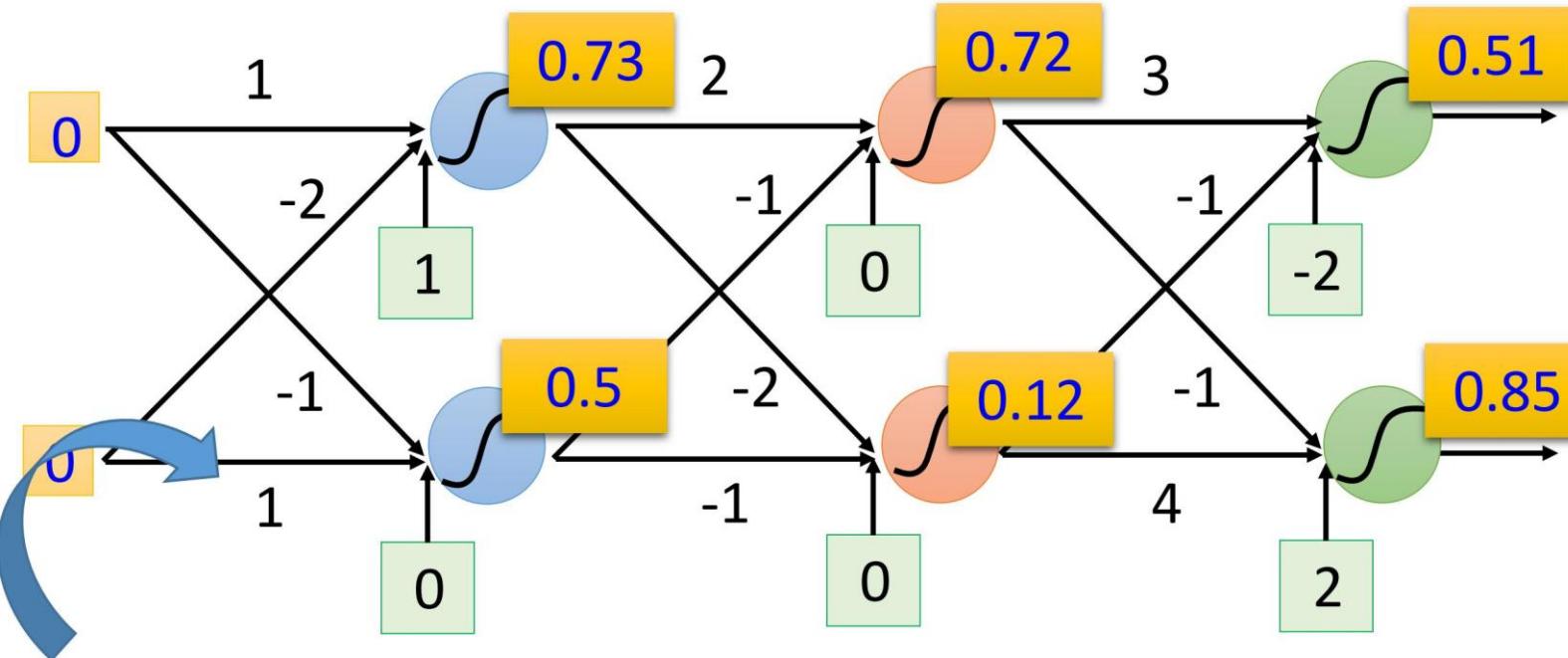
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



# Fully Connect Feedforward Network



# Fully Connect Feedforward Network



This is a function.

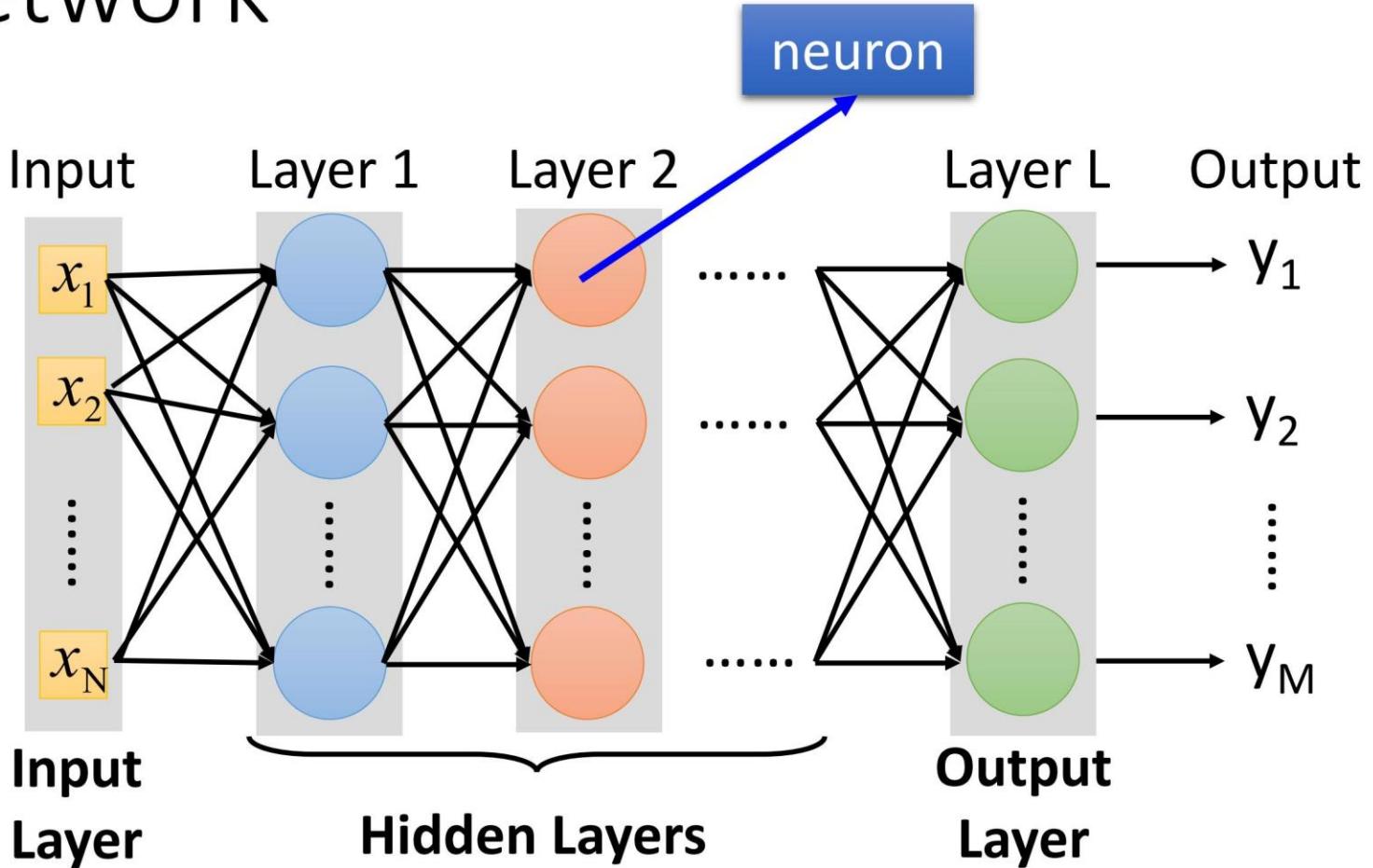
Input vector, output vector

$$f \left( \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

Given parameters  $\theta$ , define a function

Given network structure, define a function set

# Fully Connect Feedforward Network



Deep means many hidden layers

# Output Layer (Option)

- Softmax layer as the output layer

## Ordinary Layer

$$z_1 \rightarrow \sigma \rightarrow y_1 = \sigma(z_1)$$

$$z_2 \rightarrow \sigma \rightarrow y_2 = \sigma(z_2)$$

$$z_3 \rightarrow \sigma \rightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

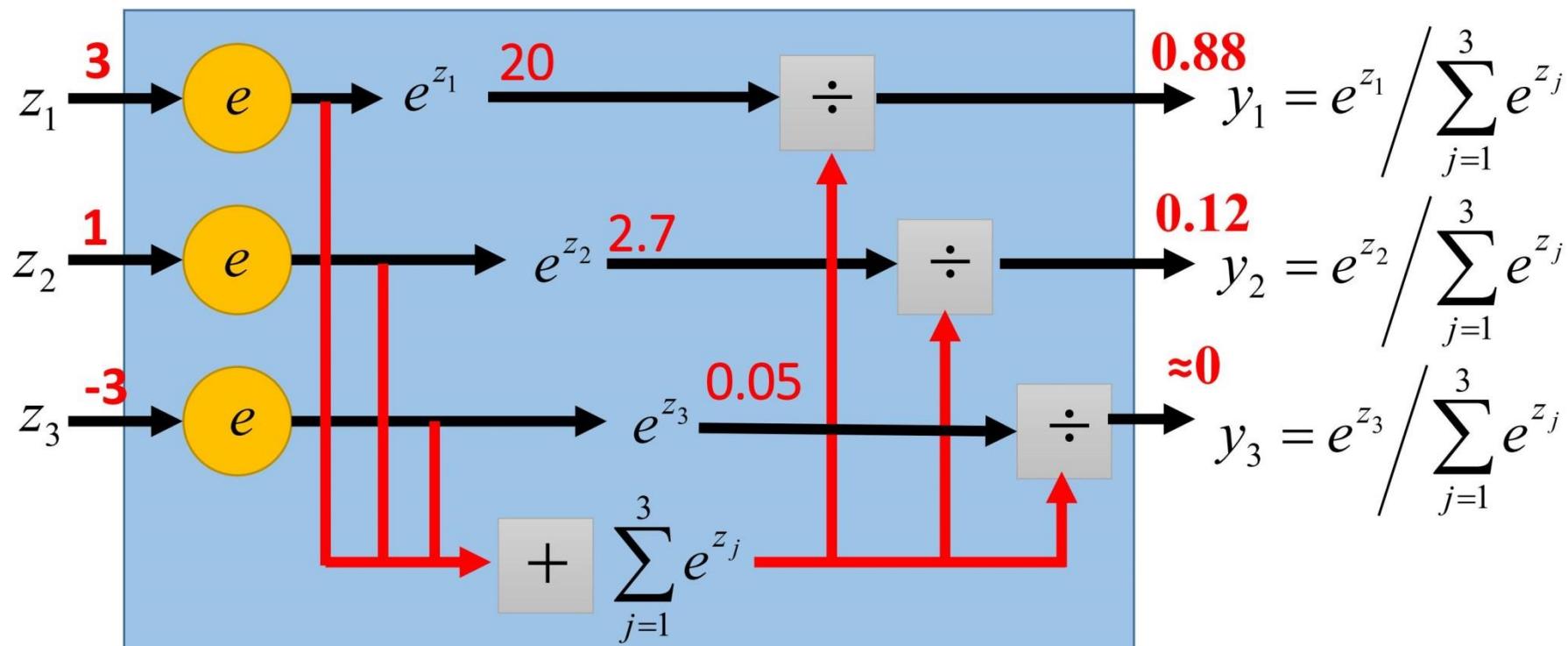
# Output Layer (Option)

- Softmax layer as the output layer

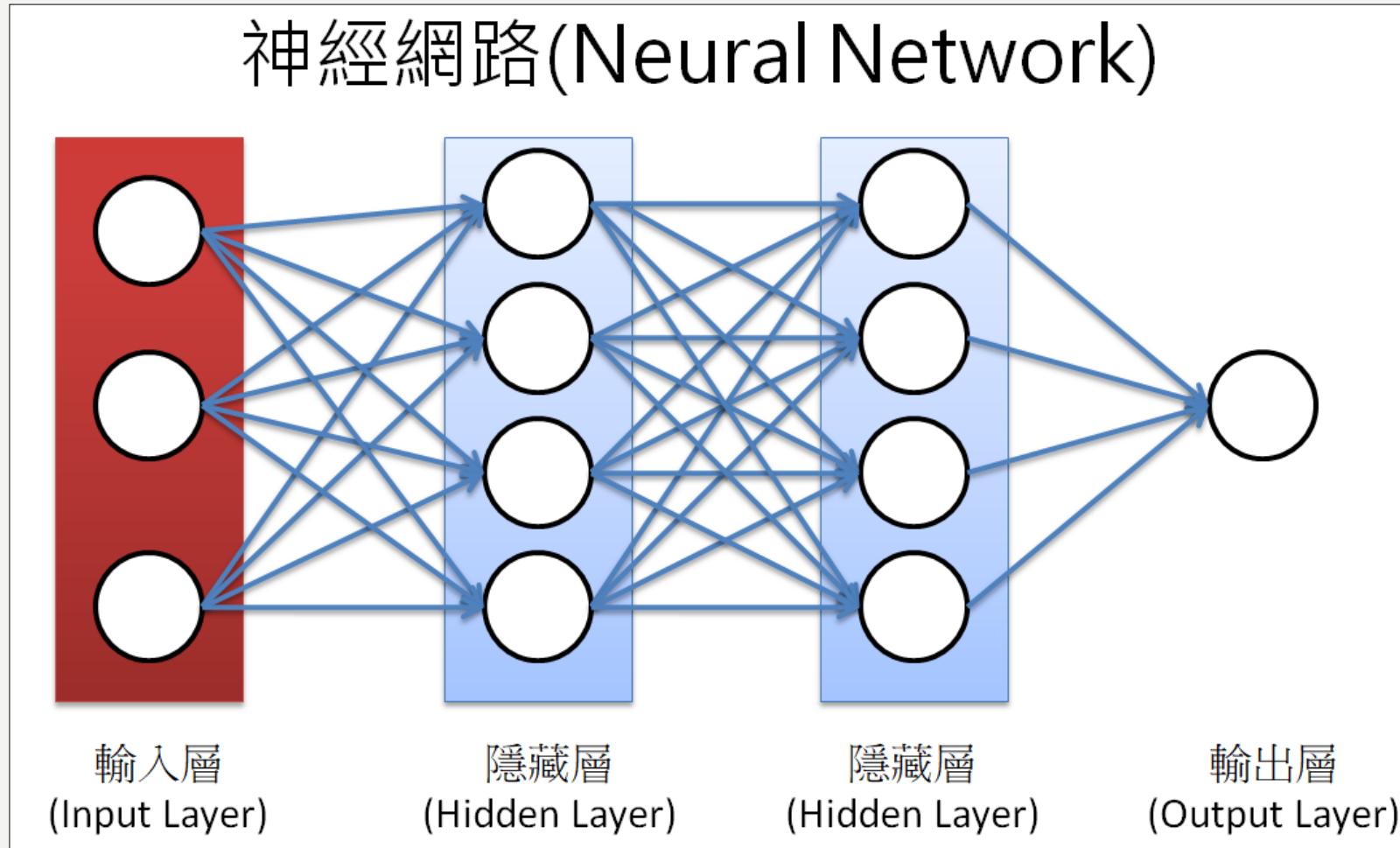
**Probability:**

- $1 > y_i > 0$
- $\sum_i y_i = 1$

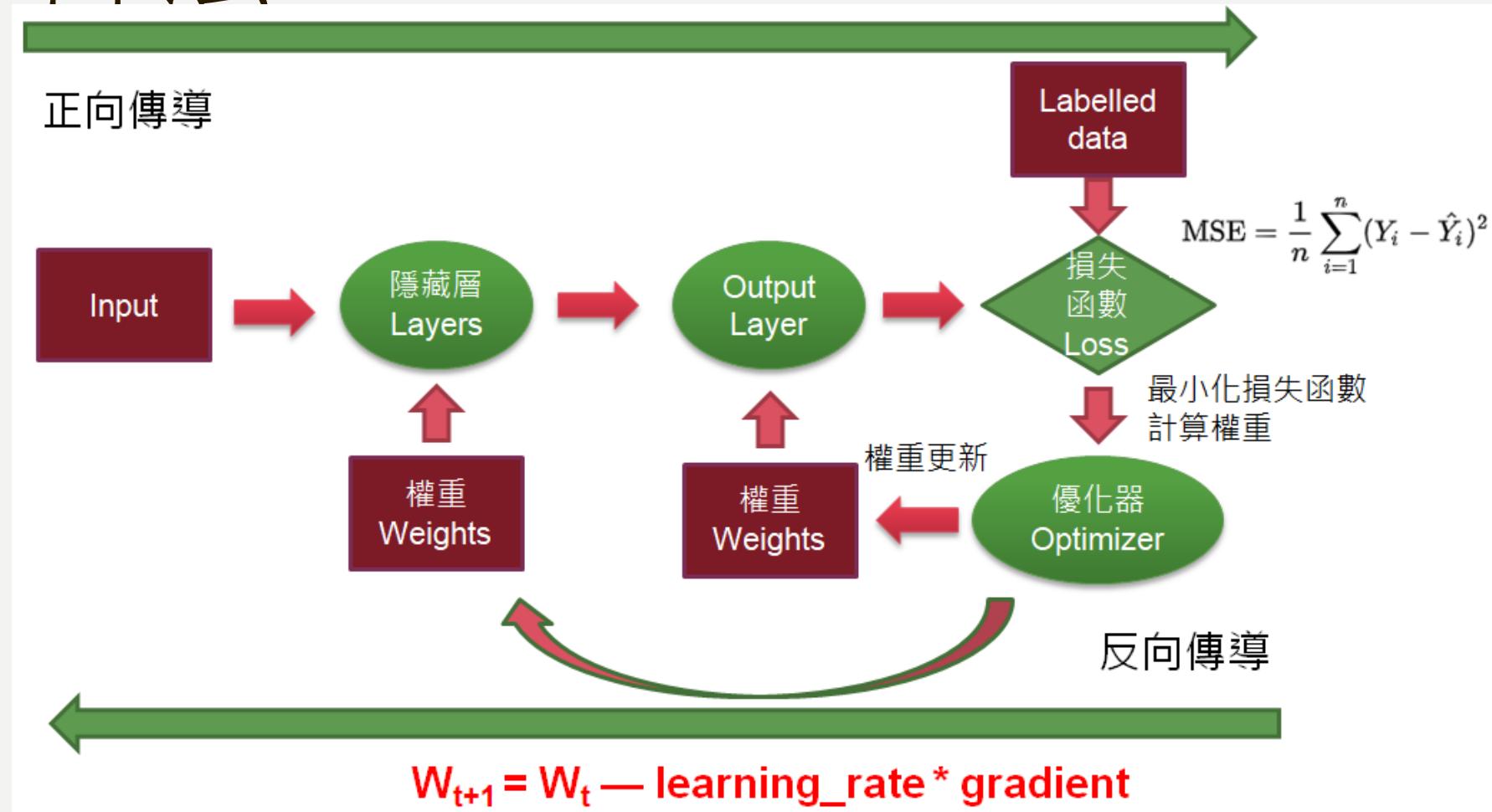
## Softmax Layer



# 神經層(NEURAL NETWORK LAYER)



# 神經網路優化求解的過程 -- 梯度下降法



# 學習路徑

- 張量(Tensor)運算：包括向量、矩陣的運算。
- 自動微分(Automatic Differentiation)：梯度計算、優化(Optimization)、神經網路梯度下降求解，不管是TensorFlow或PyTorch都提供自動微分功能，神經網路才能找到最佳解。
- 各式神經層：包括Dense、卷積(Conv1D、Conv2D、Conv3D)、循環神經層(RNN、LSTM、GRU)等。
- 神經網路：以各種神經層構建的網路模型，其中還包括各式的Activation Function、損失函數(Loss Function)、優化器(Optimizer)、效能衡量指標(Metrics)，模型又分為順序模型(Sequential model)及Functional API。

# 實作

- 03\_1\_張量運算.ipynb
- 03\_2\_自動微分.ipynb
- 03\_3\_簡單線性迴歸.ipynb
- 03\_4\_簡單的完全連階層.ipynb

## 1.2 建立專案



### 1.2 建立專案

本章程式碼建議使用「Jupyter Notebook」執行，操作流程如下：

**ST  
EP 01** 開啟 Jupyter Notebook。

在 Terminal (Ubuntu) 或命令提示字元 (Windows) 中輸入指令。

## 1.2 建立專案



### ST EP 02 建立執行檔。

點擊右上角的「New」，然後點選你安裝的 Python 直譯器（Jupyter 裡都稱作 kernel）來開啟，如圖 1-2 所示，顯示了三個不同的 kernel 分別為：

- Python3：本地端 Python。
- tf2：虛擬機 Python（TensorFlow-cpu 版本）。
- tf2-gpu：虛擬機 Python（TensorFlow-gpu 版本）。

## 1.2 建立專案



**ST  
EP 03** 執行程式碼。

在方框中輸入程式碼「`print("Hello Jupyter Notebook")`」，並透過`Shift` + `Enter`鍵執行單行程式碼，執行結果會顯示在該行程式碼的下方，如圖 1-3 所示。

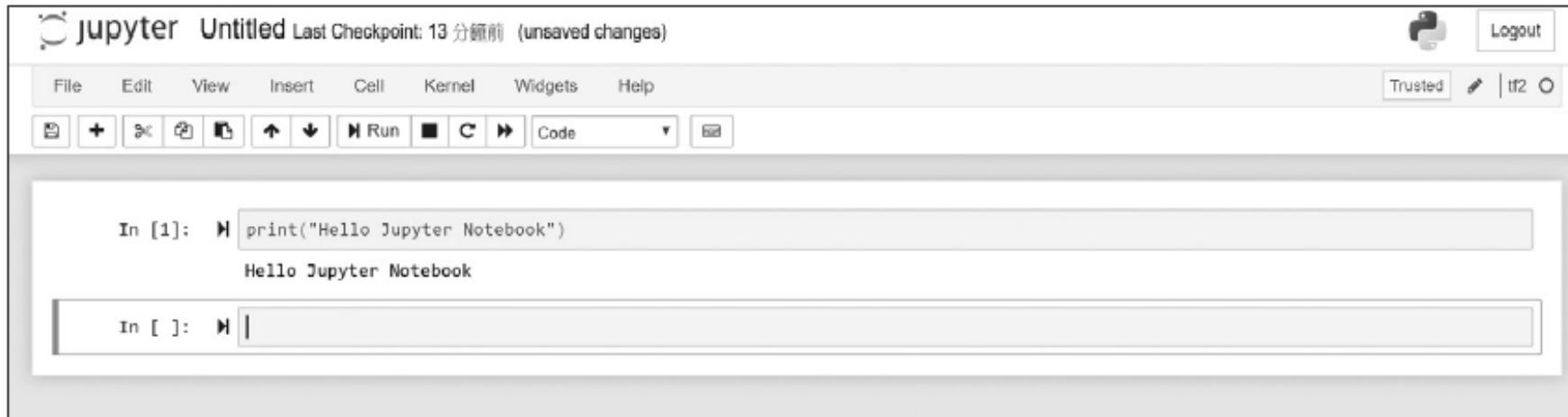


圖 1-3 Jupyter 環境介面

## 1.2 建立專案



**ST  
EP 04** 載入 TensorFlow 套件。

在方框中輸入「`import tensorflow as tf`」，並透過`Shift + Enter`鍵執行單行程式碼，即可載入 TensorFlow 套件，如圖 1-4 所示。

The screenshot shows a Jupyter Notebook window titled "jupyter Untitled Last Checkpoint: 19 分鐘前 (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar below has buttons for New, Run, Cell, Kernel, and Help. The main area displays three code cells:

- In [1]: `print("Hello Jupyter Notebook")`  
Hello Jupyter Notebook
- In [2]: `import tensorflow as tf`
- In [ ]:  (This cell is currently active, indicated by the cursor)

圖 1-4 載入 TensorFlow 套件

接下來，本章出現的程式碼皆可在「Jupyter Notebook」上執行。

# 1.3 TensorFlow 介紹



## 1.3 TensorFlow 介紹

DistBelief為Google Brain團隊一開始所使用的機器學習工具，後來以DistBelief為基礎做改進，並開放原始碼，才有了現在我們所熟知的TensorFlow，目前TensorFlow已然成為最流行的機器學習工具之一，而TensorFlow的命名也直接反應了其功能，可以將名字拆成Tensor和Flow兩部分來解讀。

# 1.3 TensorFlow 介紹



## Tensor (張量)

張量是矩陣向任意維度 (dimension) 的推廣，TensorFlow 的運算都是基於張量進行。

### ○ TensorFlow 的基本型別

#### 結果

```
tf.Tensor(1, shape=(), dtype=int32)
<tf.Variable 'Variable:0' shape=() dtype=int32, numpy=1>
```

# 1.3 TensorFlow 介紹



- 零維張量稱為「標量」。

## 結果

```
tf.Tensor(4, shape=(), dtype=int32)  
0 維 Tensor
```

- 一維張量稱為「向量」。

## 結果

```
1 綴 Tensor
```

- 二維張量稱為「矩陣」。

## 結果

```
2 綴 Tensor
```

# 1.3 TensorFlow 介紹



Flow（流）解釋為資料流動或計算。

TensorFlow的運作方式是透過產生資料流圖（Data Flow Graphs）來進行運算，又稱作「計算圖」（Computation Graph）。計算圖的節點（Nodes）用來表示數學運算，邊（Edges）則表示節點間的關聯性，如圖1-5為數學式  $\text{ReLU}(\mathbf{X}\mathbf{W}+\mathbf{b})$  的計算圖。

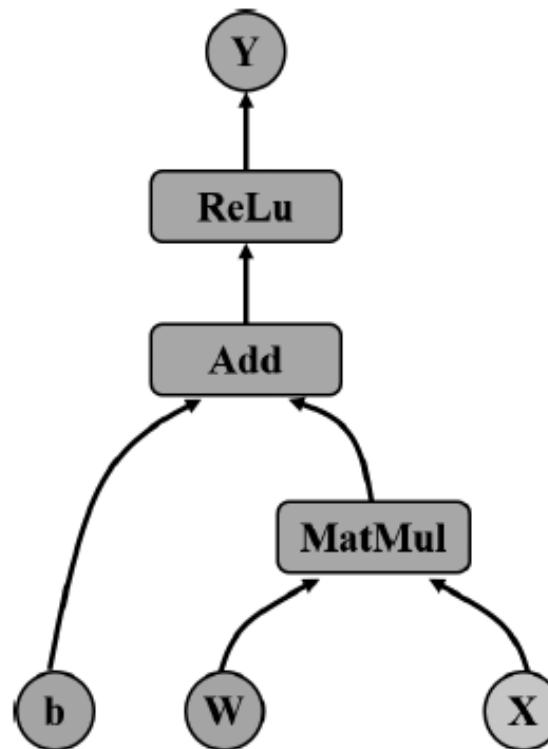


圖 1-5 計算圖

## 1.3 TensorFlow 介紹



在 TensorFlow 1.x 以前的版本都是先產生計算圖，然後再執行如同上述的運算，此架構被稱為靜態圖。而後來的 TensorFlow 2.0 由於預設 Eager Execution 模式為執行模式，因此引入了動態圖的機制，使得執行指令可以立即得到回覆。

# 1.4 TensorFlow 2.0 更動



## 1.4 TensorFlow 2.0 更動

TensorFlow 2.0 相較於前一版簡潔且容易上手，以下簡介了六個目前 TensorFlow 2.0 主要推行的功能，其中 Eager Execution、Keras 和 tf.data 這三個功能將在接下來三個小節詳細介紹，而另外三個在後面的章節才會詳細說明：

### □ Eager Execution

動態圖模式，表示為立即執行的意思，在該模式下執行運算，會立即傳回數值，讓使用者 Debug 更加便利、快速。

### □ Keras

TensorFlow 2.0 加入了 Keras 作為內建高階 API 後，有了更高的相容性。內建 Keras 可透過呼叫 `tf.keras` 使用，其具有簡潔指令、自由組合且容易擴展的模塊化 API 等特性，使得神經網路更為容易搭建。

# 1.4 TensorFlow 2.0 更動



## ❑ tf.data

使用 tf.data 建立資料輸入管道（Input pipeline），速度更快、更簡單。

## ❑ TensorFlow Hub

共享模型權重的 Library，可以從 Hub 上載入預先訓練好的 model，也可以將自己訓練好的模型上傳到 Hub 上，來分享給其他人。

## ❑ Distribution Strategy

新的 API 可以讓你更加輕鬆地完成在多台設備上的分散式訓練，例如：CPU、GPU 或 TPU。

## ❑ SavedModel

TensorFlow 2.0 已經規範好網路模型的儲存格式，使我們可以將訓練好的網路模型放到想要執行的平台上，例如：手機、樹莓派或網頁，同時也支援不同的程式語言，例如：C、Java、Go 或 C# 等。

下圖 1-6 表明了目前 TensorFlow 2.0 已經能包辦從訓練模型到部署模型的流程，意即從讀取資料、訓練模型、儲存模型到部署模型至各種裝置平台上。

# 1.4 TensorFlow 2.0 更動

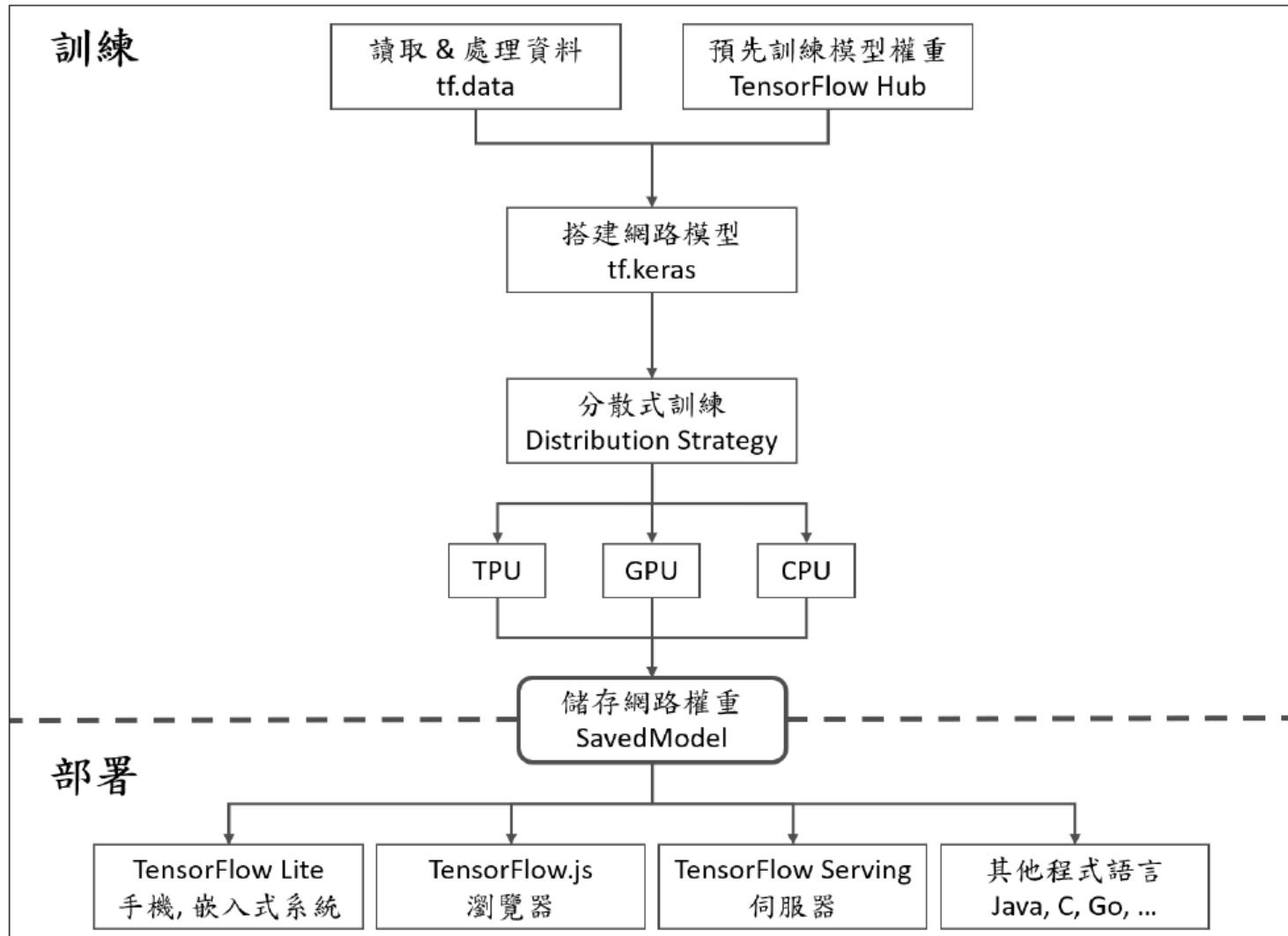


圖 1-6 模型訓練及部署流程所使用的 TensorFlow 模組

# 1.5 Eager Execution



## 1.5.1 Eager Execution 介紹

TensorFlow 引入了「Eager Execution」動態圖模式，這個模式在 TensorFlow 2.0 中為預設的執行模式，一旦運算執行就會立刻傳回數值，有別於以往的靜態圖模式需要建立計算圖才能執行。如此使得 TensorFlow 更容易入門，也使程式開發更直觀。

Eager Execution 模式的優點如下：

- 立即傳回數值，方便除錯。
- 不必定義計算圖。
- 不必初始化參數。
- 無需 `tf.Session.run`，就可以傳回運算結果。

# 1.5 Eager Execution



TensorFlow 1.x 和 TensorFlow 2.0 的比較：

□ TensorFlow 1.x code

tf.constant 在計算圖（Computation Graph）中建立節點，並可以透過 sess.run 從中取得數值：

結果

```
Tensor("Const_5:0", shape=(), dtype=int32)
a = 1
```

□ TensorFlow 2.0 code

tf.constant 會直接傳回數值，相對於 TensorFlow 1.x 省去許多行程式碼：

結果

```
tf.Tensor(1, shape=(), dtype=int32)
```

# 1.5 Eager Execution



## 1.5.2 TensorFlow 基本運算

**ST  
EP 01** 汇入必要套件。

結果

Eager Execution 是否啟動 :True

**ST  
EP 02** 定義常數 Tensor。

結果

```
a = 3  
b = 4
```

# 1.5 Eager Execution



ST  
EP 03 檢查資料型別。

## 結果

```
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
```

ST  
EP 04 基本運算。

## 結果

```
a + b = 7
a * b = 12
```

# 1.5 Eager Execution



ST  
EP 05 二維 Tensor 的運算。

在 Eager Execution 模式下，可以混合 Tensor 和 Numpy 做運算。

## 結果

```
a constant: 2D Tensor
a + b = [[2. 2.]
           [5. 7.]]
a * b = [[ 5.  6.]
           [11. 12.]]
```

# 1.5 Eager Execution



ST  
EP 06

輸出的結果為 Tensor 格式，我們可以將它轉為 Numpy 格式。

## 結果

NumpyArray:

[ [2. 2.]

[5. 7.] ]

說 明

TensorFlow 2.0 中，雖然有 Eager Execution 模式能夠讓 Tensor 格式支援 Python 基本運算 ( ex: for, if...else 等 )，但並非完全兼容，例如：OpenCV 或 Matplotlib 等套件的 API 輸入格式有可能會不支援 Tensor 格式。而當你遇到資料型別錯誤等問題，最快的解決方法是將其轉為 Numpy 格式，因為其歷史悠久、高效能及受歡迎程度，讓 Numpy 格式能適用大部分的 Python 套件。

# 1.5 Eager Execution



ST  
EP 07 計算梯度：假設損失函數為  $w^2$ 。

## 結果

```
tf.Tensor([[2.]], shape=(1, 1), dtype=float32)
```

### 說 明

簡單來說，標量  $x$  的梯度（Gradient）就是計算  $f(x)$  對  $x$  的微分，同理多維的向量  $x$  的梯度就是計算  $f(x)$  對所有元素的偏微分。計算出來的梯度是有方向和大小的向量，而梯度指向的方向為局部最大值，所以在第 2 章介紹的梯度下降法就是往梯度的反方向（局部最小值）更新權重。

## 1.6.1 Keras 介紹

Keras 是 François Chollet 於 2014~2015 年開始編寫的開源高階深度學習 API，主要用於快速搭建和訓練網路模型。Keras 本身並沒有運算能力，它是執行在 TensorFlow、CNTK 和 Theano 等深度學習開源套件上，這些套件稱為 Keras 的後端（backend），剛開始 Keras 後端只支援 Theano，直到 2015 年底 TensorFlow 開源後，Keras 才搭建 TensorFlow 後端，而今天 TensorFlow 已成為 Keras 最常用的後端。總而言之，Keras 將這些深度學習套件封裝為更容易使用的指令。

TensorFlow 2.0 將 Keras 納為內建高階 API，因此不用額外安裝 Keras 套件，直接透過 `tf.keras` 指令即可使用。`tf.keras` 和 Keras 的差別在於，`tf.keras` 更能全面支援 TensorFlow 的指令與模式，例如：支援 Eager Execution、`tf.data`、TPU 訓練等。

# 1.6 Keras



下面將會介紹兩種最常使用的網路搭建方法：

- Sequential Model（序列模型）。
- Function API（函數式模型）。

接下來的範例會使用到以下幾種網路層，這邊會先簡略介紹。如果不懂也沒有關係，詳細的理論和功能會在後面章節提到。以下範例的重點在於，使用 Keras 搭建網路架構的方便性與靈活性。

- Dense：搭建全連接層的指令。
- Conv2d：搭建卷積層的指令。
- Flatten：將輸入攤平，Reshape 成一維張量，大多使用在卷積層與全連接層之間。
- Add：將兩個層的輸出加在一起。
- Concatenate：將兩個層的輸出於指定的維度進行串聯（Concat），這種方法相較於 Add，保留更多資訊量，但計算量較大。

# 1.6 Keras



說 明

使用 tensorflow.keras.utils.plot\_model 需要額外安裝 pydot 和 graphviz 套件。

❑ Ubuntu 安裝方法：開啟 Terminal 並輸入以下指令：

```
pip install pydot  
apt-get install graphviz
```

❑ Windows 安裝方法：

① 開啟命令提示字元並輸入以下指令：

```
pip install pydot
```

# 1.6 Keras



②到 [https://graphviz.gitlab.io/\\_pages/Download/Download\\_windows.html](https://graphviz.gitlab.io/_pages/Download/Download_windows.html)，下載 Windows 安裝檔並安裝，如圖 1-7 所示。



圖 1-7 graphviz 下載頁面

# 1.6 Keras



③最後將 graphviz/bin 安裝路徑加到環境變量裡，執行「控制台→系統及安全性→系統→進階系統設定→環境變數→系統環境變數中的『Path』新增環境變量」，如圖 1-8 所示。

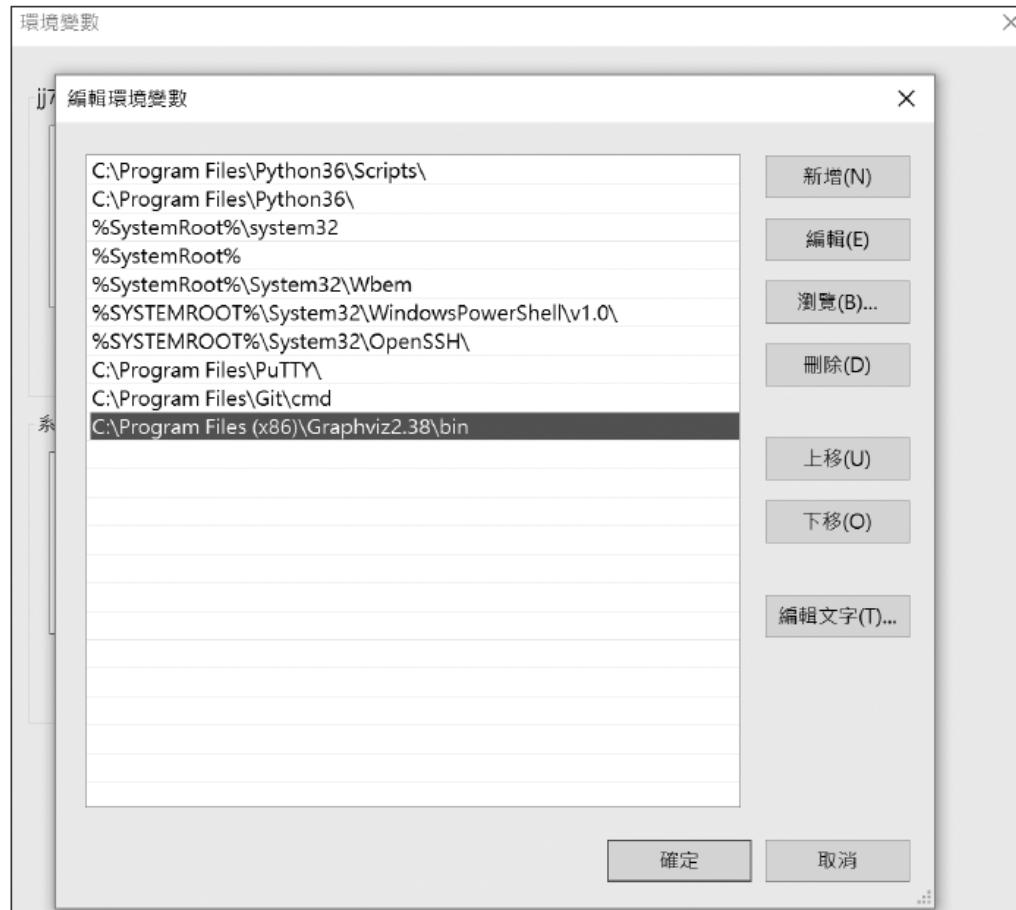
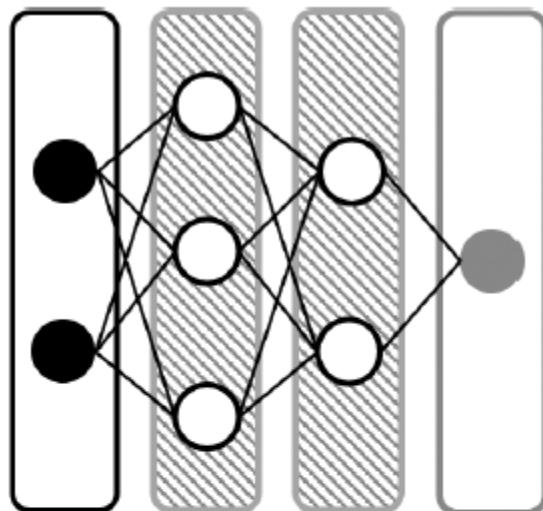


圖 1-8 新增環境變量

## 1.6.2 Sequential Model

Sequential Model（序列模型）搭建方法簡單快速，並且解決大多數的問題以及應用，例如：手寫數字辨識、房價預測或評論分類等，基本上只要是迴歸問題或是分類問題，都可以用 Sequential Model 解決。但是 Sequential Model 的搭建方法有限制，必須要逐層依序搭建網路，且網路模型必須為單個輸入層和單個輸出層，如圖 1-9 中的 Single Input and Output Model。

## 1.6 Keras



Single Input and Output Model

---

● Input Unit      ○ Hidden Unit      ● Output Unit

□ Input Layer      ■ Hidden Layer      □ Output Layer

圖 1-9 網路模型示意圖

# 1.6 Keras



下方示範了 Sequential Model 的兩種搭建方法，並以分類問題為範例：輸入為  $28 \times 28$  的影像並拉平為 784 的一維向量，輸出為 10 的一維向量（分為十個類別），中間使用兩層隱藏層各擁有 64 個神經元，而隱藏層的激活函數為 ReLU，輸出層為 Softmax。

**ST  
EP 01** 汇入必要套件。

# 1.6 Keras



**ST  
EP 02** 建立模型（搭建方法有兩種）。

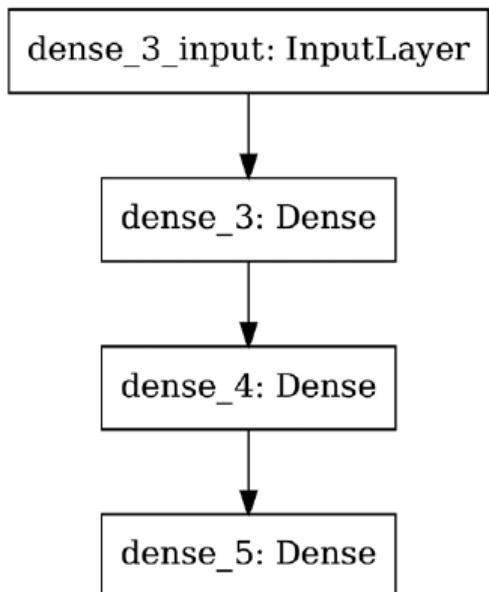
- 方法一
- 方法二

# 1.6 Keras



ST  
EP 03 顯示網路模型。

## 結果



## 1.6.3 Functional API

Keras 中的 Functional (函數式) API 是建立網路模型的另一種方式，它提供了更多的靈活性，能夠建立更複雜的模型，例如：「多輸入單輸出的網路」、「單輸入多輸出的網路」和「多輸入多輸出的網路等」，如圖 1-10 所示。

# 1.6 Keras

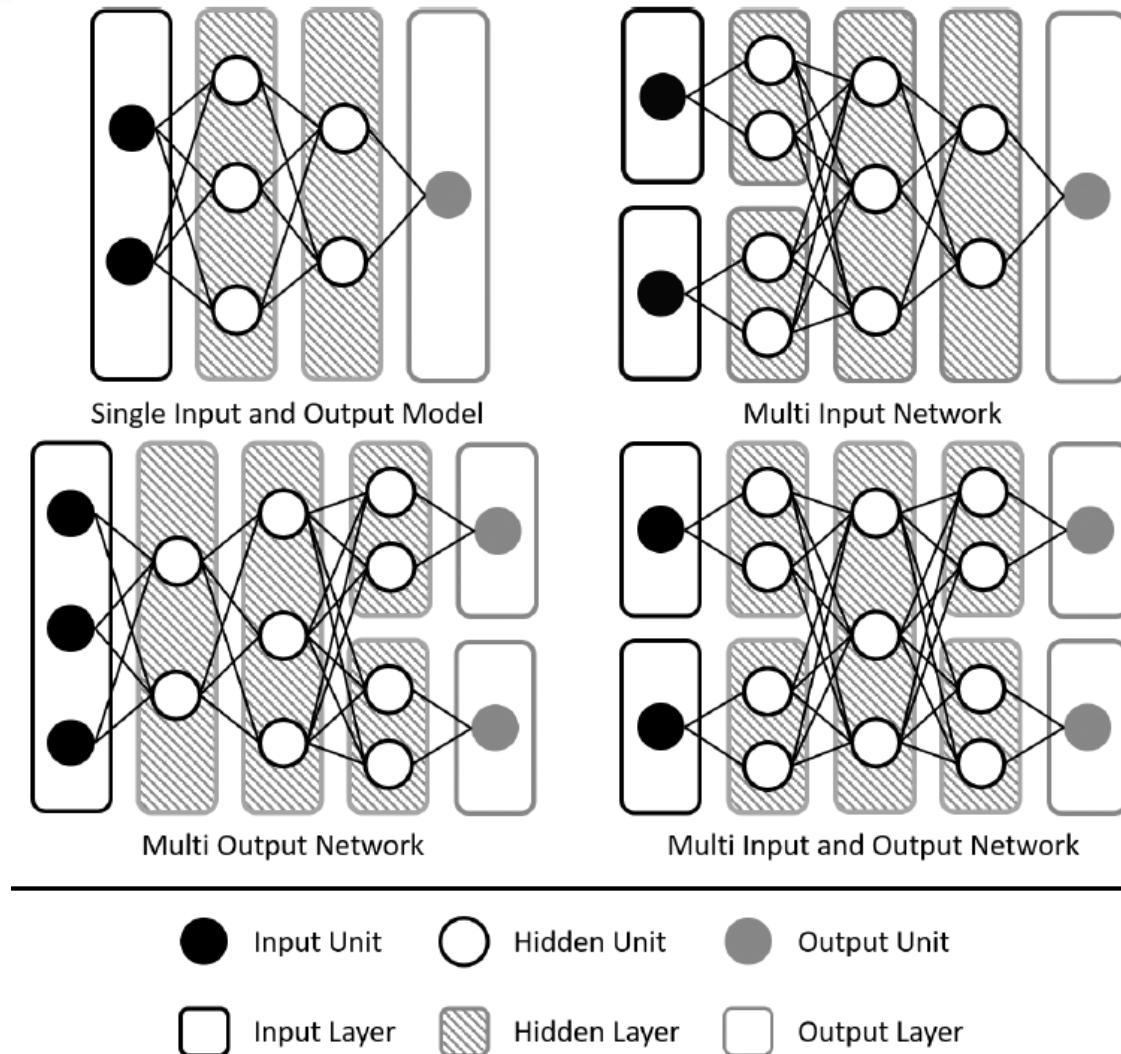


圖 1-10 網路模型示意圖

# 1.6 Keras



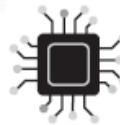
Functional API 在使用上非常的靈活，並且我們在後面介紹到的進階應用，全部都能使用 Functional API 方式搭建實現，以下介紹幾種進階應用：

- 物件偵測（Object detection）。
- 影像分割（Image Segmentation）。
- 生成對抗網路（Generative Adversarial Network）。

下方將會依序簡介幾種網路架構以及應用的場景：

- Single Input and Output Model：單輸入單輸出的網路。
- Multi Input Model：多輸入單輸出的網路。
- Multi Output Model：單輸入多輸出的網路。
- Multi Input and Output Model：多輸入多輸出的網路。

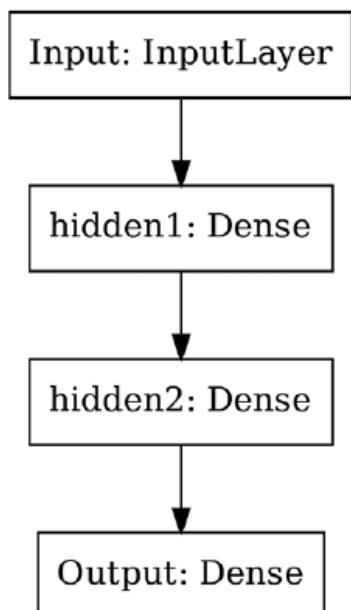
# 1.6 Keras



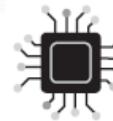
## Single Input and Output Model

單輸入單輸出模型，例如：輸入為  $28 \times 28$  的影像並拉平為 784 的一維向量，輸出為 10 的一維向量（分為十個類別），中間使用兩層隱藏層各擁有 64 個神經元，而隱藏層的激活函數為 ReLU，輸出層為 Softmax。

### 結果



# 1.6 Keras



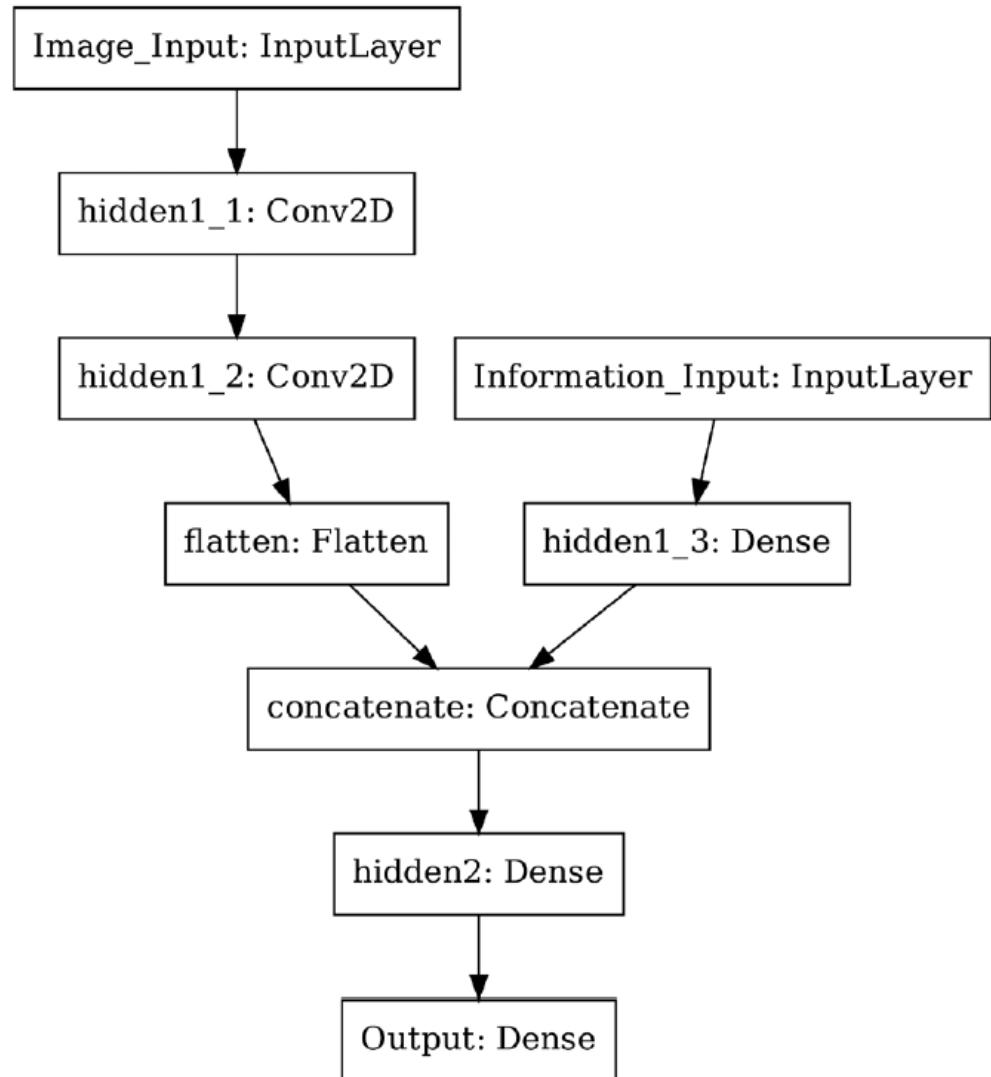
## Multi Input Model

多輸入單輸出模型，例如：商品價格預測為兩個輸入（商品圖片和商品品牌）、一個輸出（價格預測），商品圖片(128, 128, 3)輸入經過三層隱藏層，以及商品品牌(1, )輸入經過一層隱藏層，結合兩個資訊後再經過一層隱藏層，而輸出為價格預測(1, )。

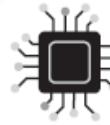
# 1.6 Keras



## 結果



# 1.6 Keras



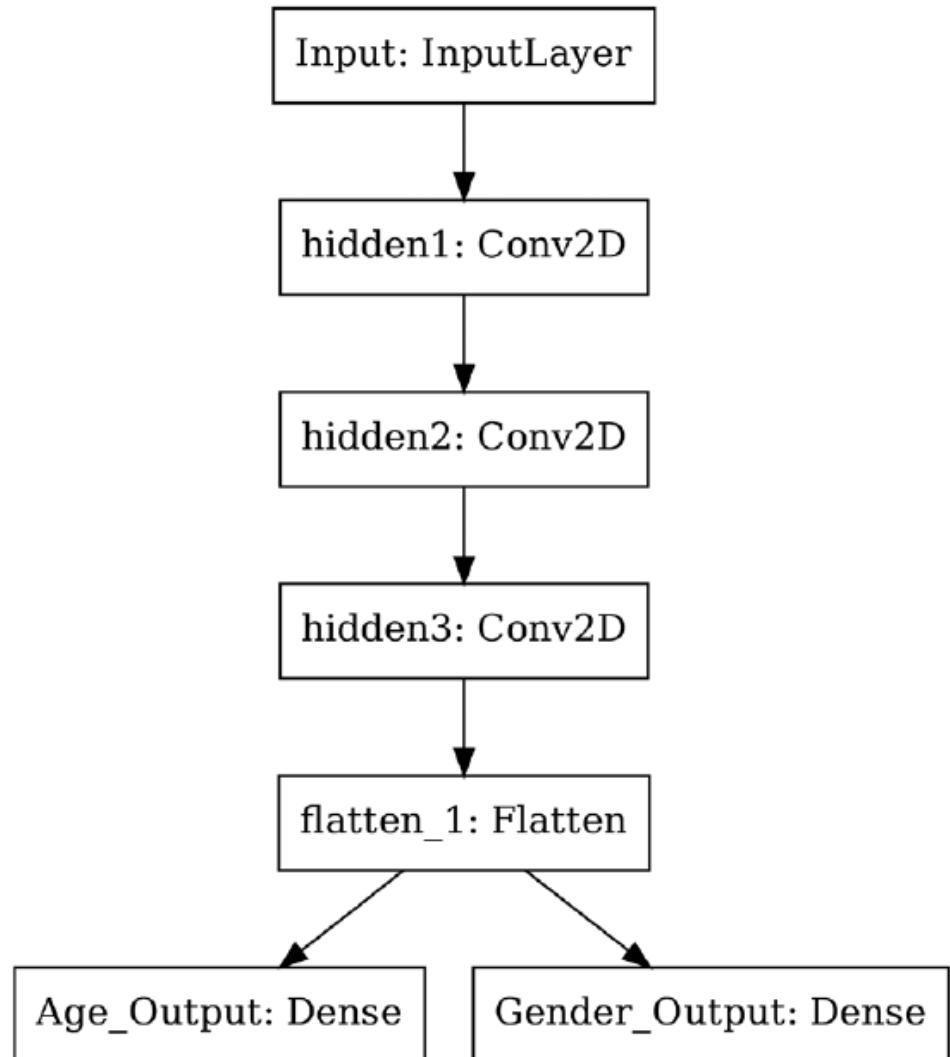
## Multi Output Model

單輸入多輸出模型，例如：人像識別為一個輸入（人物照片）、兩個輸出（年齡和性別），人物照片(128, 128, 3)輸入經過三層隱藏層，而輸出為年齡(1,)和性別(1,)兩種不同資訊。

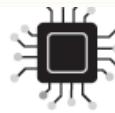
# 1.6 Keras



## 結果



# 1.6 Keras



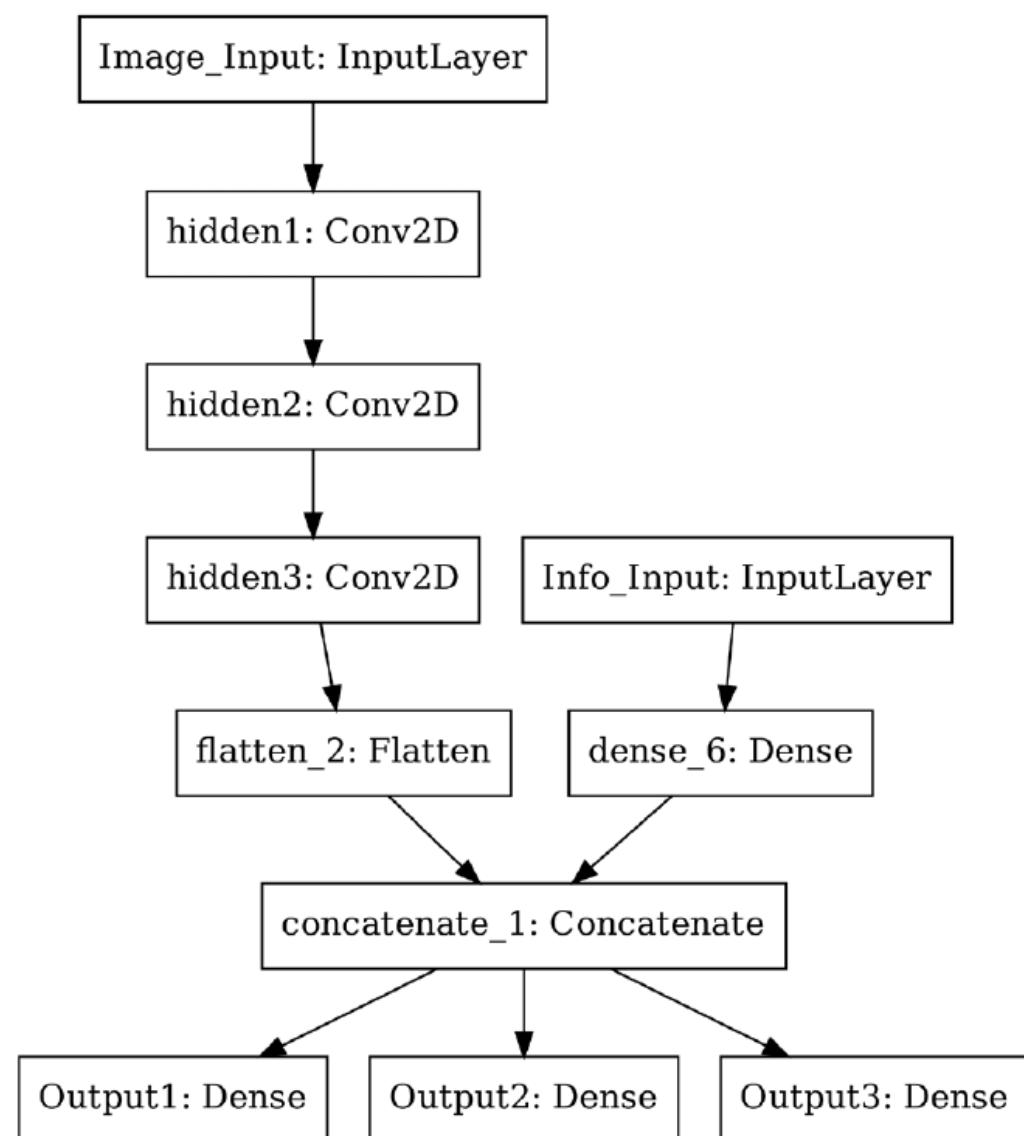
## Multi Input and Output Model

多輸入多輸出模型，例如：天氣預測為兩個輸入（衛星雲圖和氣候資訊）、三個輸出（機率、溫度和濕度），衛星雲圖 ( $256 \times 256 \times 3$ ) 輸入經過三層隱藏層，以及氣候資訊 (10,) 輸入經過一層隱藏層，並結合兩個資訊，而輸出為氣候資訊如降雨機率 (1,)、溫度 (1,) 和濕度 (1,) 等三種不同資訊。

# 1.6 Keras



## 結果



## 1.7.1 tf.data 介紹

從讀取資料到資料傳入加速設備（GPU 或 TPU）的流程，可以被稱為「輸入管道」（Input Pipeline），輸入管道可以分為三個步驟：

**STEP 01** 提取（Extraction）。

從儲存地方（可以是 SSD、HDD 或遠端儲存位置）讀取資料。

**STEP 02** 轉換（Transformation）。

使用 CPU 做資料預處理，例如：對影像做翻轉、裁剪、縮放和正規化等。

**STEP 03** 載入（Loading）。

將轉換後的資料載入到機器學習模型的加速設備。

## 1.7 tf.data



上面這三個步驟主要是裝置讀取資料和 CPU 預處理在消耗時間，如果沒有妥善的分工處理，會造成當 CPU 在準備資料時，GPU 在等待訓練資料產生。反之，當 GPU 在訓練時，CPU 則是空閒狀態，如圖 1-11 所示，如此訓練時間會增加非常多。



圖 1-11 一般訓練情況

## 1.7 tf.data



TensorFlow 提供 tf.data API，可以幫助使用者打造靈活有效的輸入管道。輕鬆處理大量資料、不同資料格式及複雜的轉換。而且透過使用 `tf.data.Dataset.prefetch` 一行指令，就可以讓生成資料與訓練資料同時進行，提升訓練效率，如圖 1-12 所示。



圖 1-12 CPU 和 GPU 可同步進行

## 1.7 tf.data



倘若輸入管道的執行時間遠比訓練時間還久，將發生如圖 1-13 所示的情況，造成 GPU 或 TPU 加速器無法發揮全部的運算力，通常這種情形可能是讀取檔案太大或資料預處理太久所造成。



圖 1-13 讀取與處理時間太長

上述問題可以使用 CPU 多線程來解決，只需在呼叫 map 方法時加入 num\_parallel\_calls 設定，即可開啟並執行處理資料的功能。通常 num\_parallel\_calls 會設定成電腦的核心數，圖 1-14 為改善後的工作情形。

# 1.7 tf.data

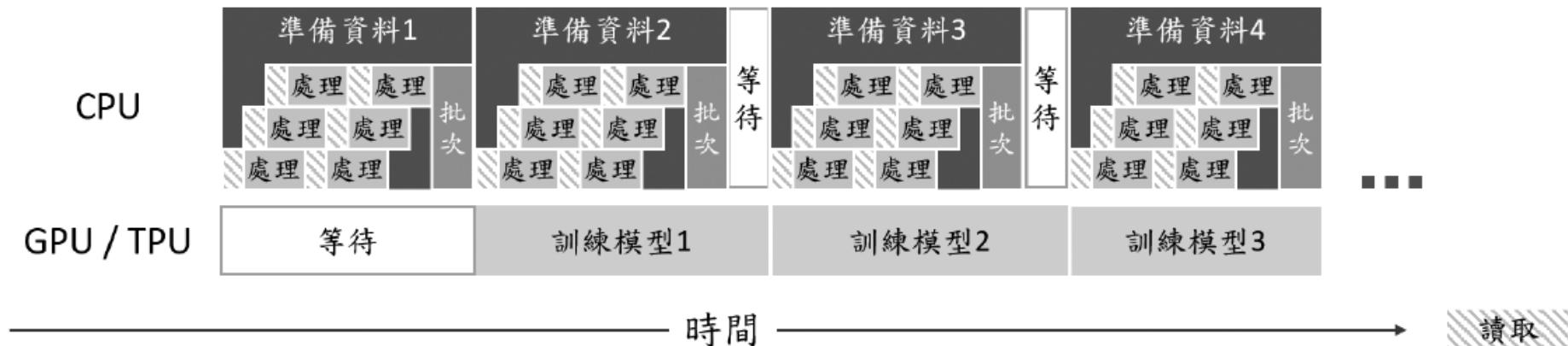


圖 1-14 CPU 可多進程準備資料

## 1.7.2 基本操作

看完上面對 tf.data 的概述後，下面將利用幾個例子帶你了解如何建立 Dataset、設定 Dataset 和提取資料（注意：下方程式請依序執行）。

### □ tf.data.Dataset.from\_tensors

使用 from\_tensors 建立 Dataset。

#### 結果

```
<TensorDataset shapes: (10,), types: tf.int32>
```

# 1.7 tf.data



## □ tf.data.Dataset.from\_tensor\_slices

使用 from\_tensor\_slices 建立 Dataset。

### 結果

```
<TensorSliceDataset shapes: (), types: tf.int32>
<TensorSliceDataset shapes: (), types: tf.int32>
```

## □ for loop

讀取資料。

### 結果

```
tf.Tensor([ 1  2  3  4  5  6  7  8  9 10], shape=(10,), dtype=int32)
```

# 1.7 tf.data



```
for data1, data2 in zip(x_data, y_data):
    print('x: {}, y: {}'.format(data1, data2))
```

## 結果

```
x: 0, y: 0
x: 1, y: 2
x: 2, y: 4
x: 3, y: 6
x: 4, y: 8
x: 5, y: 10
x: 6, y: 12
x: 7, y: 14
x: 8, y: 16
x: 9, y: 18
```

# 1.7 tf.data



## □ take

透過參數可以指定讀取資料數量。

### 結果

```
tf.Tensor([ 1  2  3  4  5  6  7  8  9 10], shape=(10,), dtype=int32)
```

### 結果

```
x: 0, y: 0
x: 1, y: 2
x: 2, y: 4
x: 3, y: 6
x: 4, y: 8
```

# 1.7 tf.data



如果指定讀取數量超過 Dataset 的資料量，則會取得與 Dataset 大小相同的資料量。

## 結果

```
x: 0, y: 0  
x: 1, y: 2  
x: 2, y: 4  
x: 3, y: 6  
x: 4, y: 8  
x: 5, y: 10  
x: 6, y: 12  
x: 7, y: 14  
x: 8, y: 16  
x: 9, y: 18
```

# 1.7 tf.data



## □ tf.data.Dataset.zip

將多個 Dataset 打包成一個。

### 結果

```
<ZipDataset shapes: (((), ()), types: (tf.int32, tf.int32))>
```

## □ map

可以使用 map 來轉換資料。

### 結果

```
<MapDataset shapes: (), types: tf.int64>
```

## □ 命名

能以字典方式為 elements 的組件命名。

### 結果

```
<ZipDataset shapes: {y: (), x: ()}, types: {y: tf.int64, x: tf.int64}>
```

# 1.7 tf.data



## 結果

```
x: 0, y: 0
x: 1, y: 2
x: 2, y: 4
x: 3, y: 6
x: 4, y: 8
x: 5, y: 10
x: 6, y: 12
x: 7, y: 14
x: 8, y: 16
x: 9, y: 18
```

# 1.7 tf.data



- 設定每一批 (batch) 讀取的資料量

## 結果

```
x: [0 1], y: [0 2]  
x: [2 3], y: [4 6]  
x: [4 5], y: [ 8 10]  
x: [6 7], y: [12 14]  
x: [8 9], y: [16 18]
```

- shuffle

Dataset 資料會被載入 buffer 中，並從 buffer 中隨機選取資料出來，取出資料產生的空位會從新的資料替補。而 buffer\_size 是設定 buffer 大小，最好的設定是大於或等於整個 Dataset 資料個數。

## 結果

```
x: [0 1], y: [0 2]  
x: [6 7], y: [12 14]  
x: [4 5], y: [ 8 10]  
x: [8 9], y: [16 18]  
x: [2 3], y: [4 6]
```

# 1.7 tf.data



## repeat

當 Dataset 的資料讀取完後，就會讀取不到資料，透過設定 repeat( $n$ ) 可以重複讀取 Dataset  $n$  次。結果

```
x: [0 1], y: [0 2]
x: [6 7], y: [12 14]
x: [4 5], y: [ 8 10]
x: [8 9], y: [16 18]
x: [2 3], y: [4 6]
```

---

```
x: [6 7], y: [12 14]
x: [0 1], y: [0 2]
x: [4 5], y: [ 8 10]
x: [8 9], y: [16 18]
x: [2 3], y: [4 6]
x: [8 9], y: [16 18]
x: [4 5], y: [ 8 10]
x: [6 7], y: [12 14]
x: [0 1], y: [0 2]
x: [2 3], y: [4 6]
```

本章結束



Q&A 討論時間