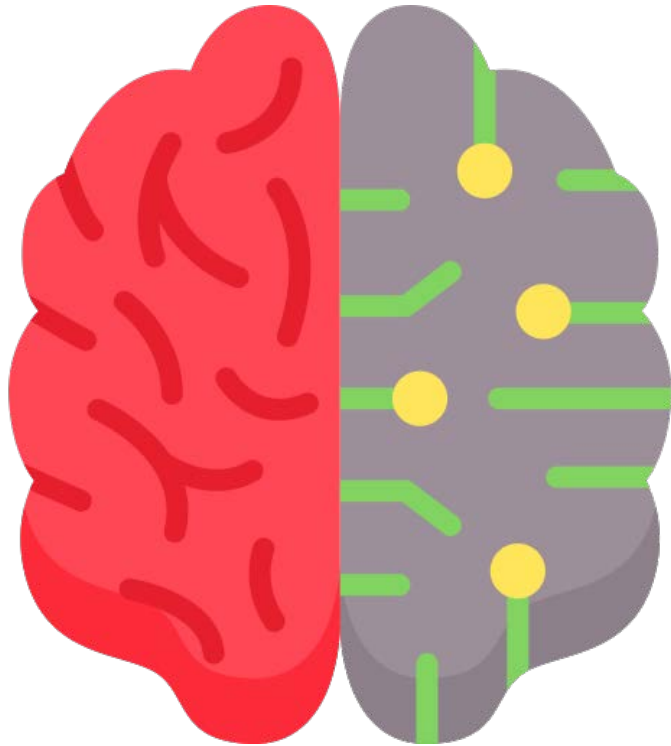


# 何謂AI

AI已在你我生活之中

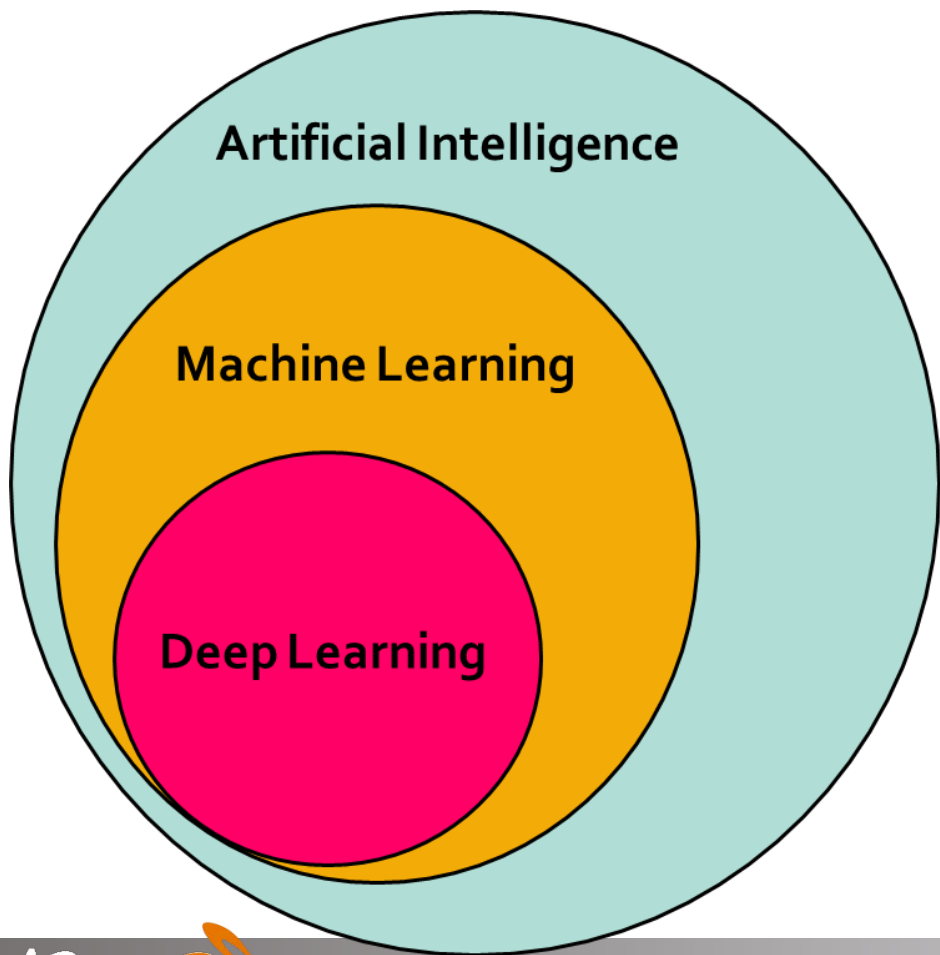
講師:隋建德

# AI - Artificial Intelligence 人工智慧



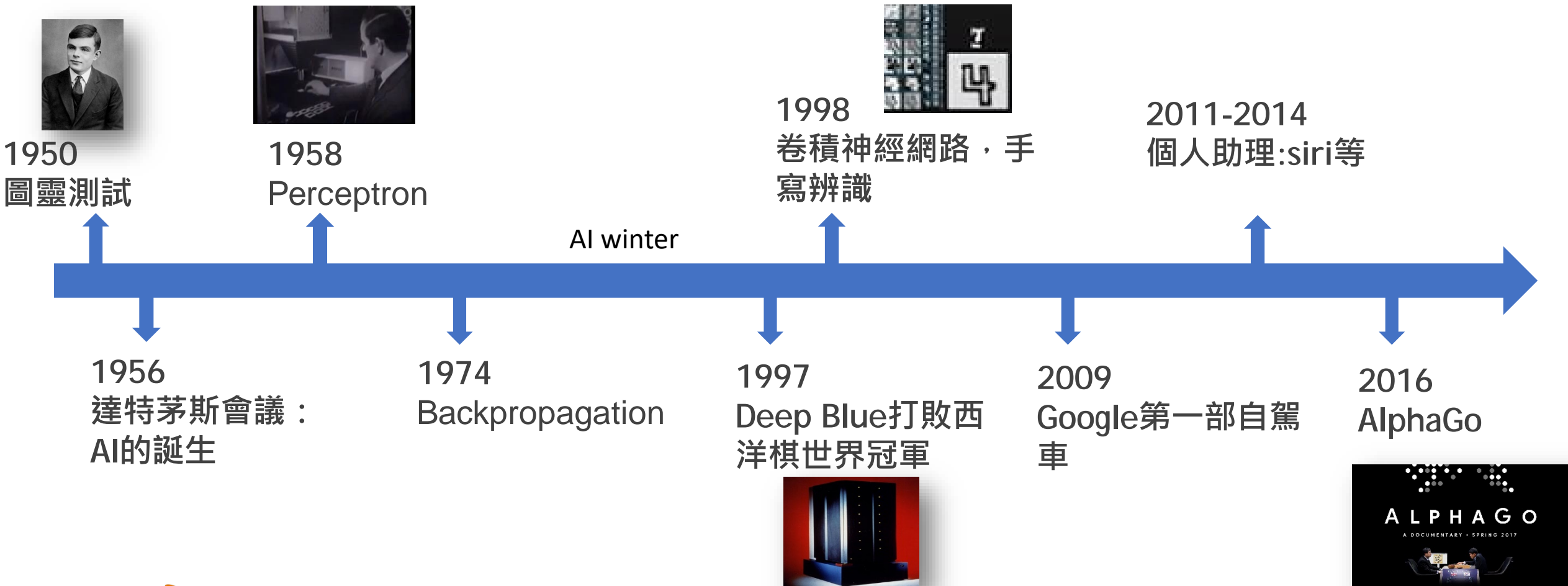
- 模擬人類智慧
- 代替人類做決定、工作

# AI? 機器學習(Machine Learning)? 深度學習(Deep Learning)?



- AI: 模擬人類智慧
  - 涵蓋廣
  - 有智慧就算
  - 一個擁有非常詳盡的 **規則** 系統也可以是 AI
- Machine learning是達成 AI 的一種方法
  - 從龐大資料當中學習出 **規則**
  - 依照**規則**找到 **方程式** 能解決特定的問題
- Deep learning 是machine learning的一種
  - 從**feature engineering** 走向**architecture engineering**
  - 不再人工萃取特徵
  - 深層網路萃取更抽象特徵

# AI 發展史



# AI 三大應用領域

- 語音辨識
- 影像辨識
  - 圖像辨識
  - 人臉辨識
  - 自動駕駛
- 自然語言處理(Natural language processing : NLP)
  - 理解人類文字與話語
  - 語意語法分析
  - 聊天機器人

小知識:  
Siri 為語音辨識  
+NLP的技術

# 圖像辨識

Function(



)



貓咪

Function(

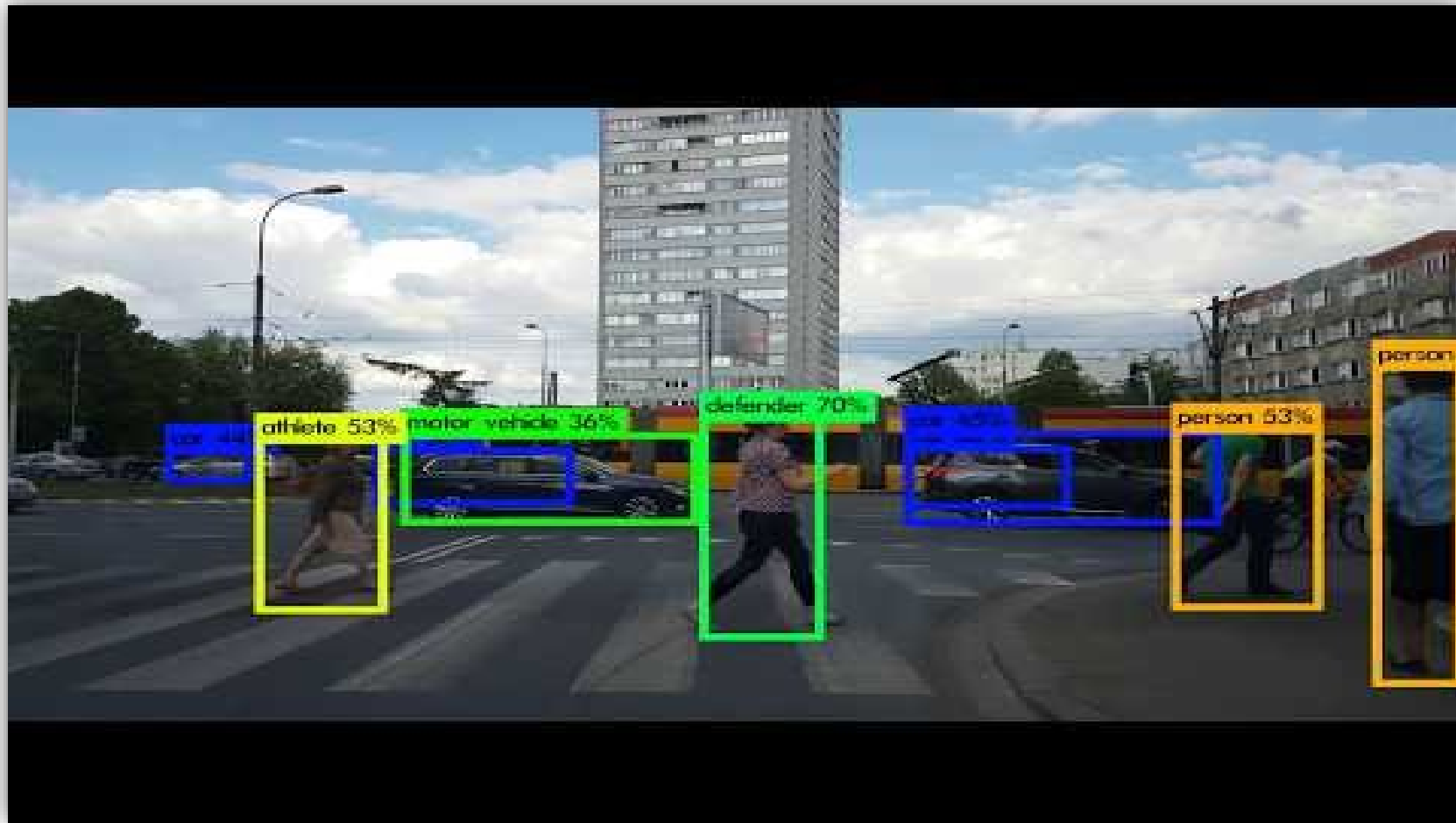


)



狗

# 圖像辨識範例: Object detection and classification



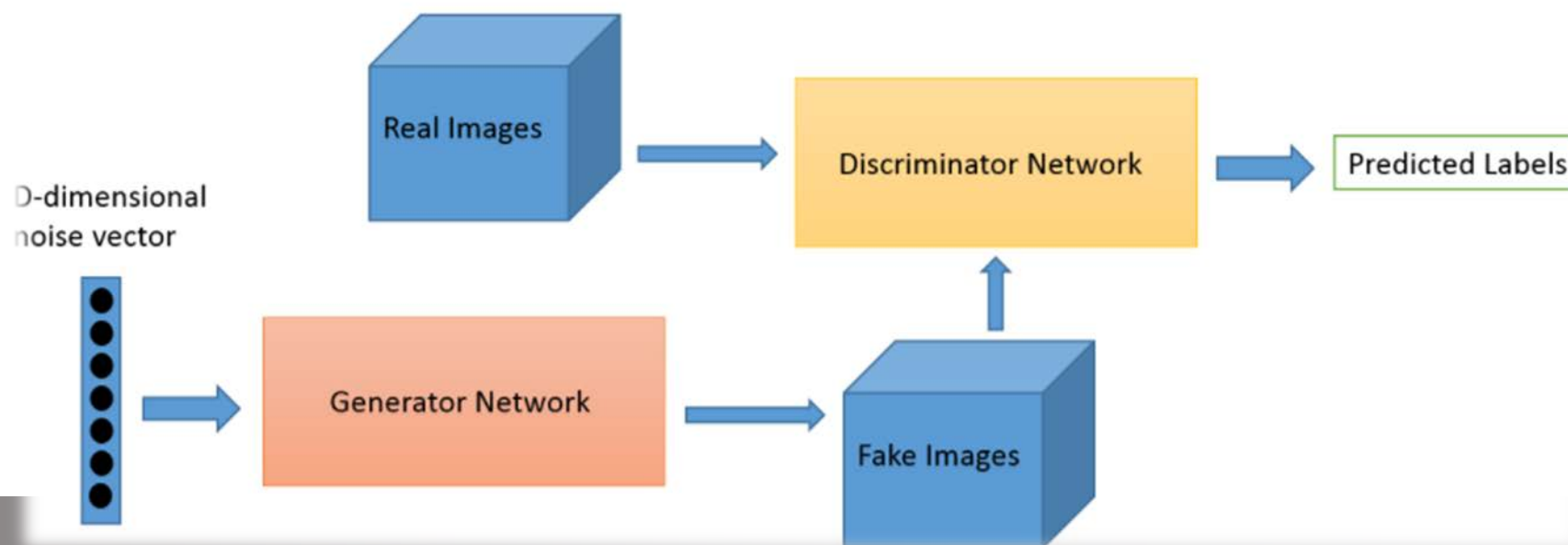
# NLP與語音辨識範例-Google Duplex





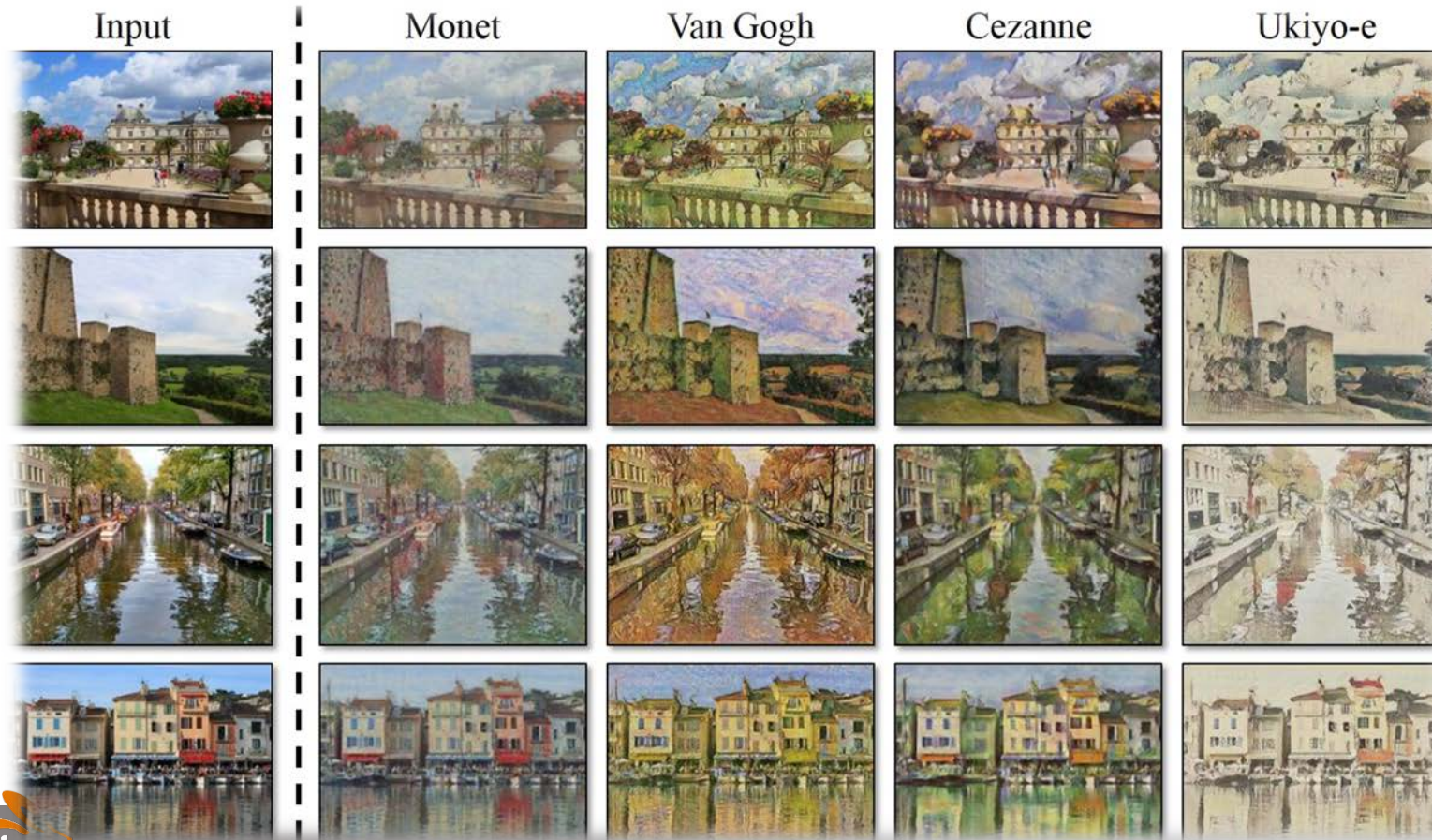
# GAN-Generative Adversarial Network 生成對抗網路

- 近年來最有趣的想法
- 概念:
  - 偽造者與警察的訓練，偽造者製造假鈔給警察判斷真假，再得到警察的回饋訓練製造假鈔的能力



# GAN-風格轉換?

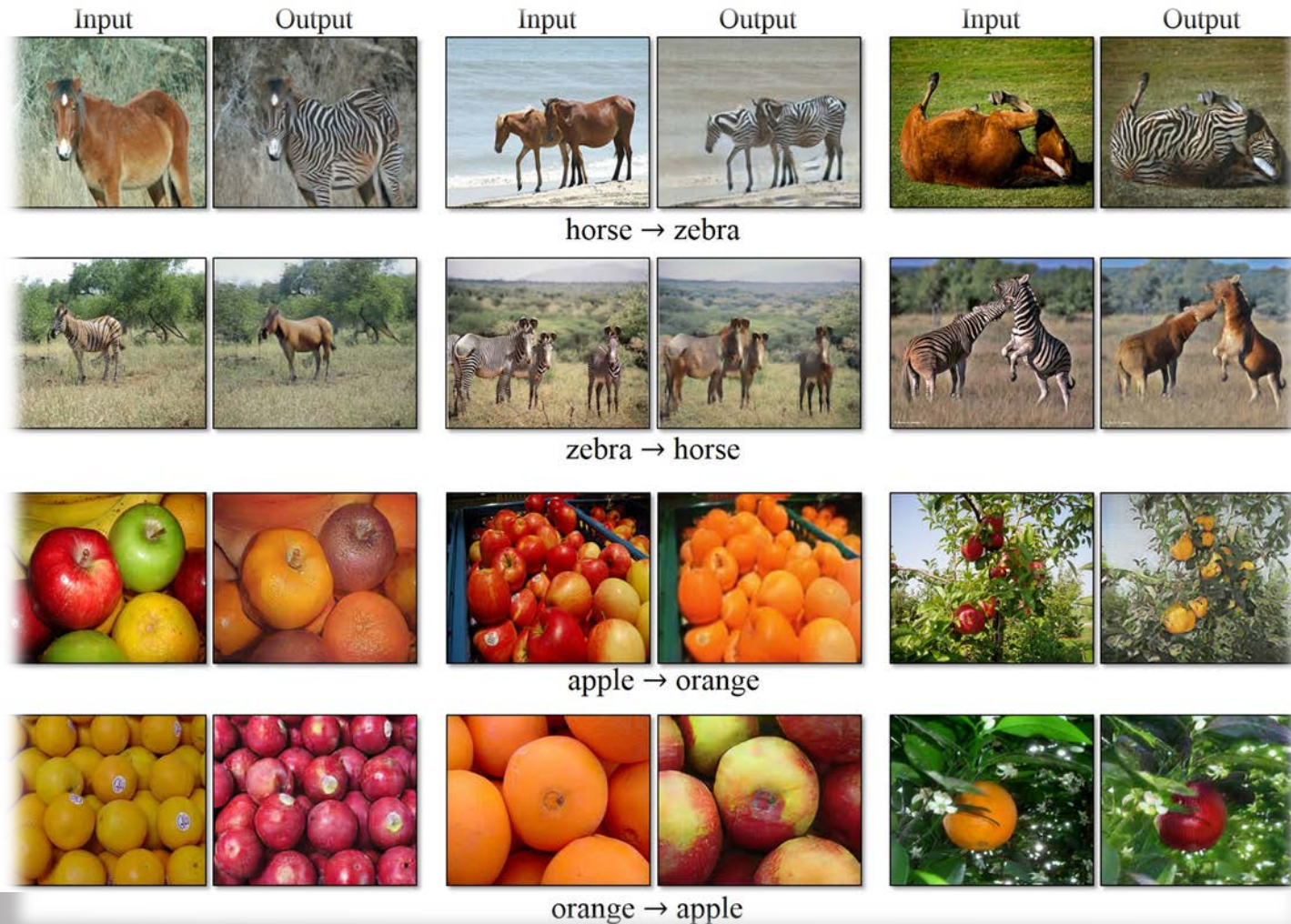
- 將一般照片轉換成莫內、梵谷風格





# GAN-偷天換日?

- 創造以假亂真的圖片
  - 可以增加資料量



# GAN-生成高解析度影像

- 生成比原圖更高解析度圖像

original



bicubic  
(21.59dB/0.6423)



SRGAN  
(20.34dB/0.6562)



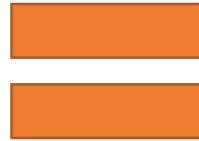
# GAN範例-Deepfake



<https://www.youtube.com/watch?v=cQ54GDm1eL0>



# Rabboni



加速度計  
(Accelerometer)

可以測量  
Rabboni動的  
加速度

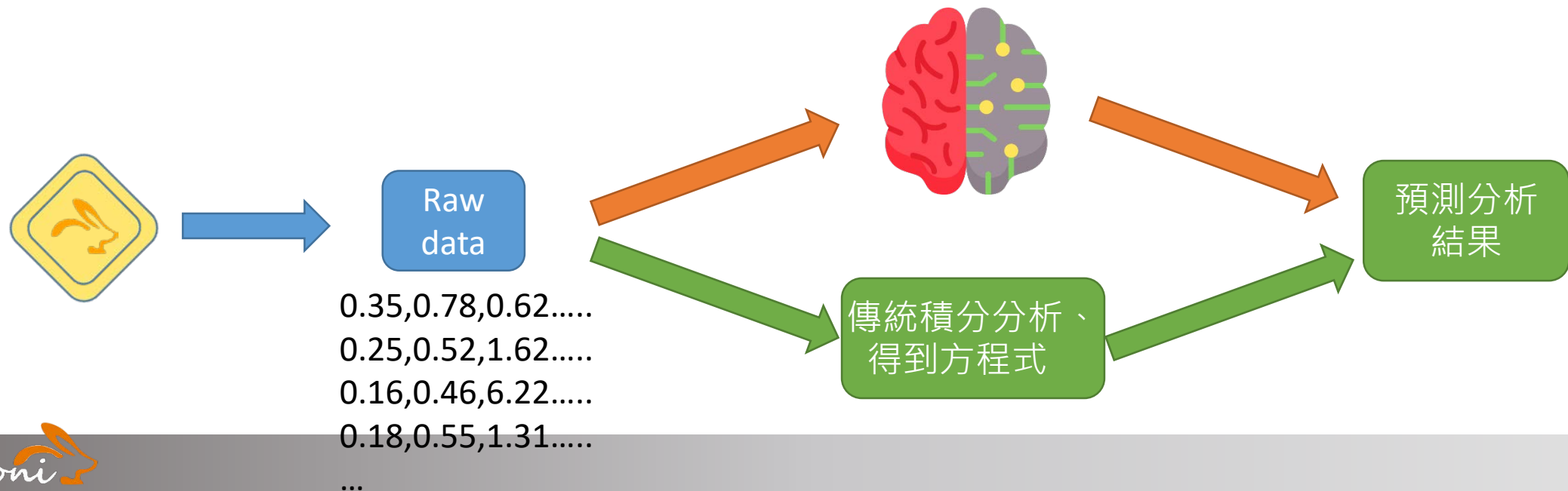


陀螺儀  
(Gyroscope)

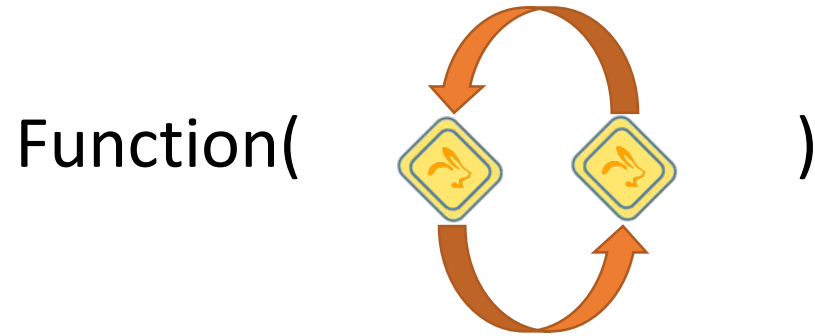
可以測量  
Rabboni轉的  
加速度

# Rabboni

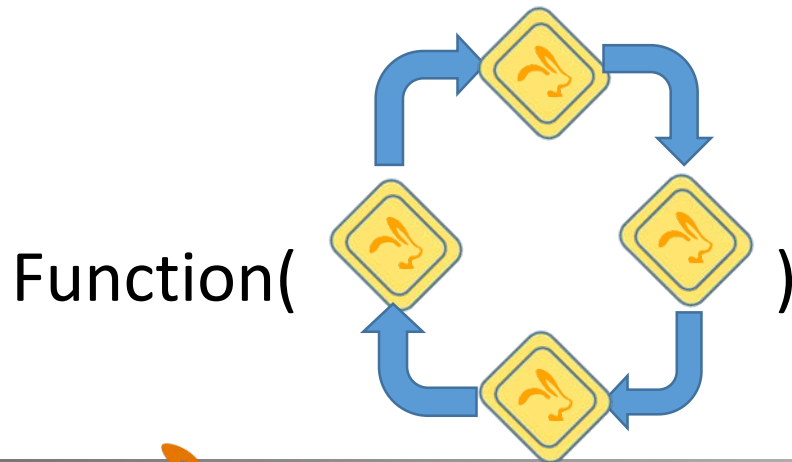
- 不同的運動方式有不同的資料
- 擁有高精確度的 加速度三軸 與 角加速度三軸
- 將sensor的原始資料(raw data)，一堆數字給AI判斷



# Rabboni



圓圈運動?



方形運動?

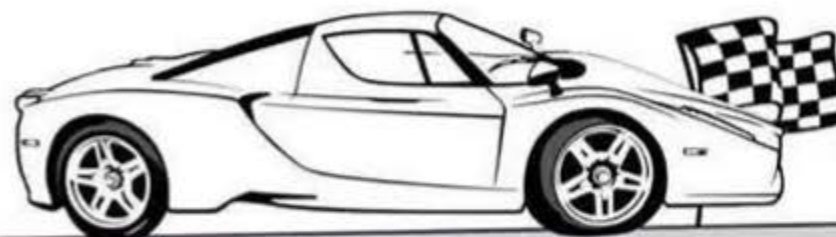


## 你眼中的大数据分析



数据提取

模型建立



深度学习, 人工智能

## 真实的大数据分析



需求讨论

提取数据



数据清洗

数据整合



缺失值处理



特征工程

模型评估

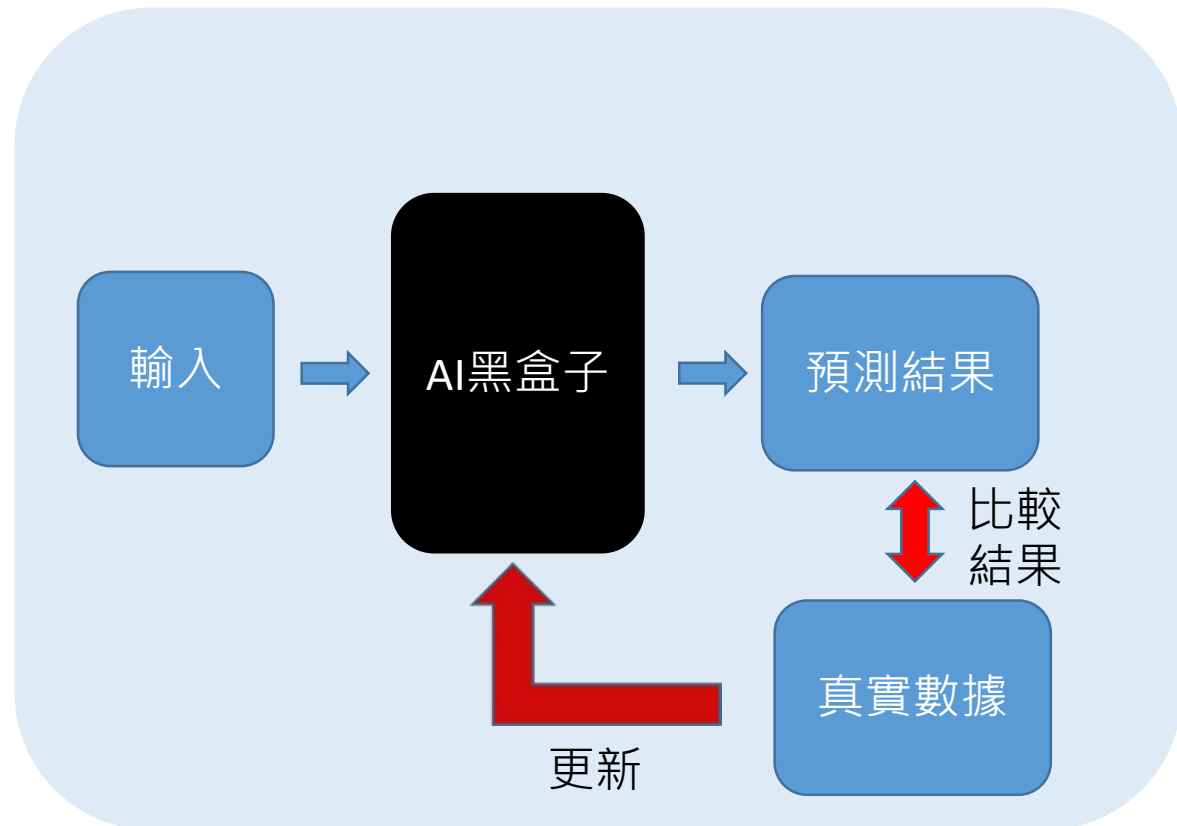


# 機器學習

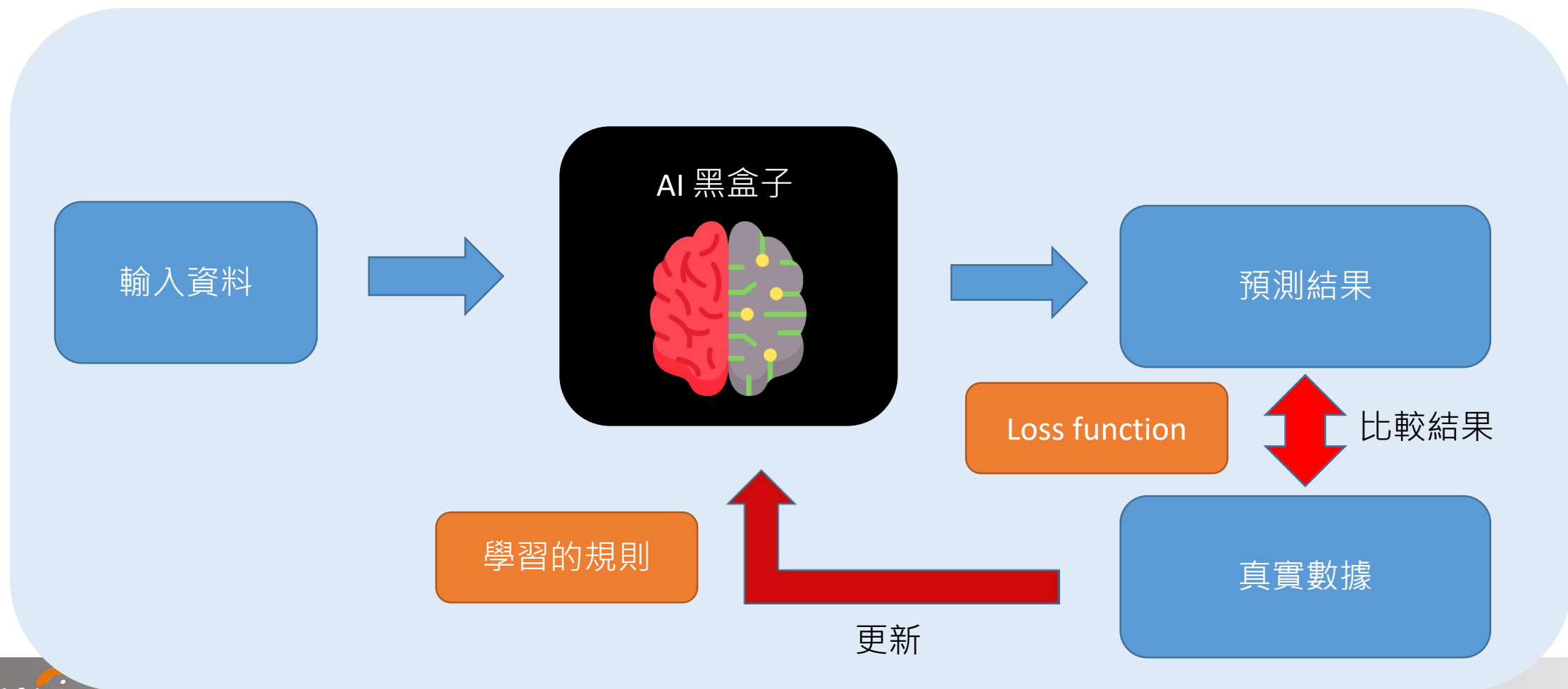
Machine learning

# 概念

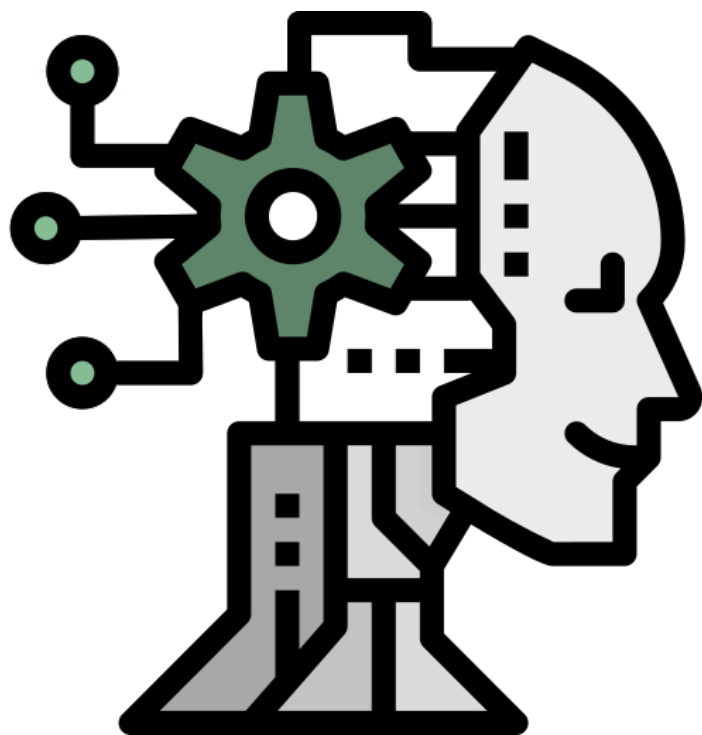
- 給予機器大量資料，使其學習規則
- 資料可能內涵...
  - 輸入(input data)
  - 真實數據(ground truth)
  - 損失函數(loss function)
- 設定機器學習的規則...
  - Optimizer
  - 學習率



# 概念



# 概念



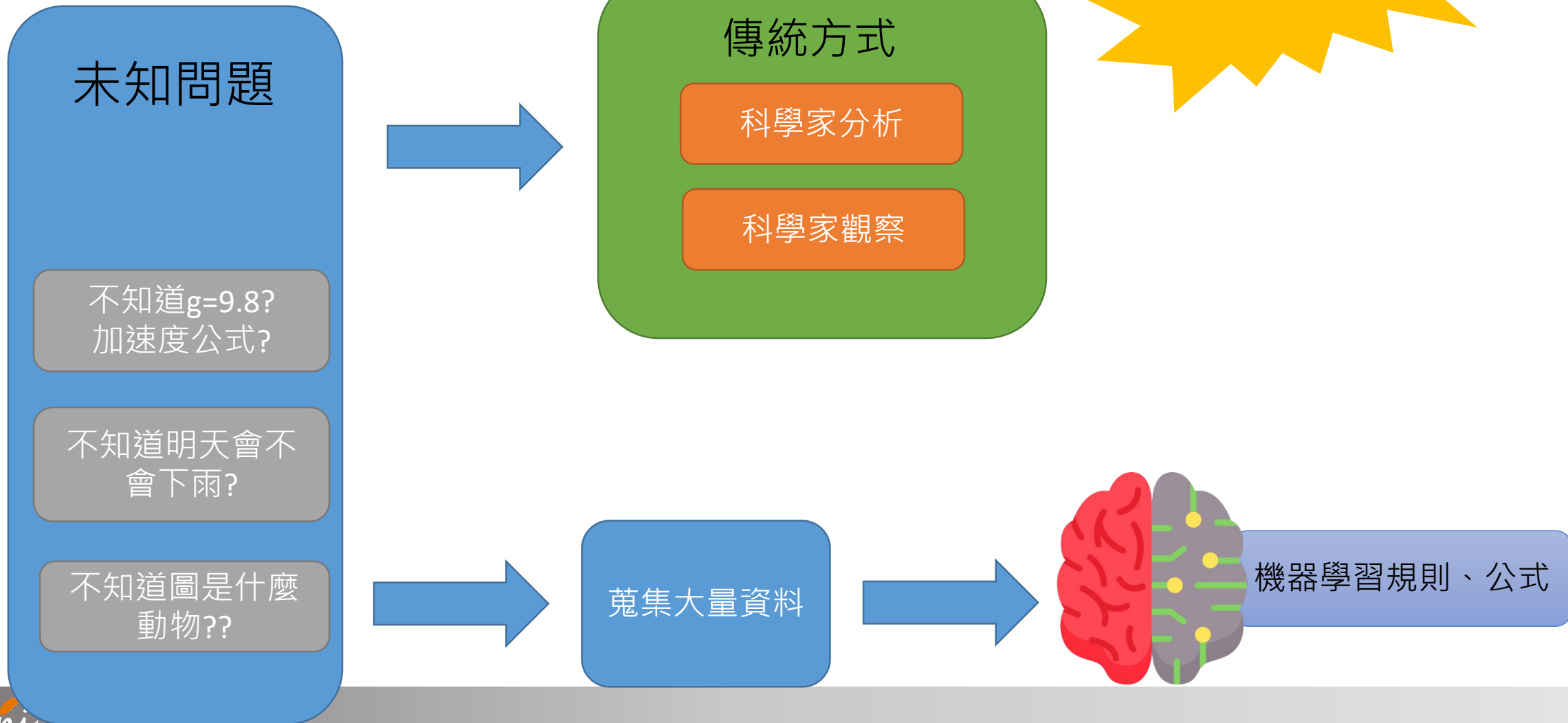
## 好處

- 免複雜假設與各種預設模型
- 適合複雜的分析
- 適用範圍廣

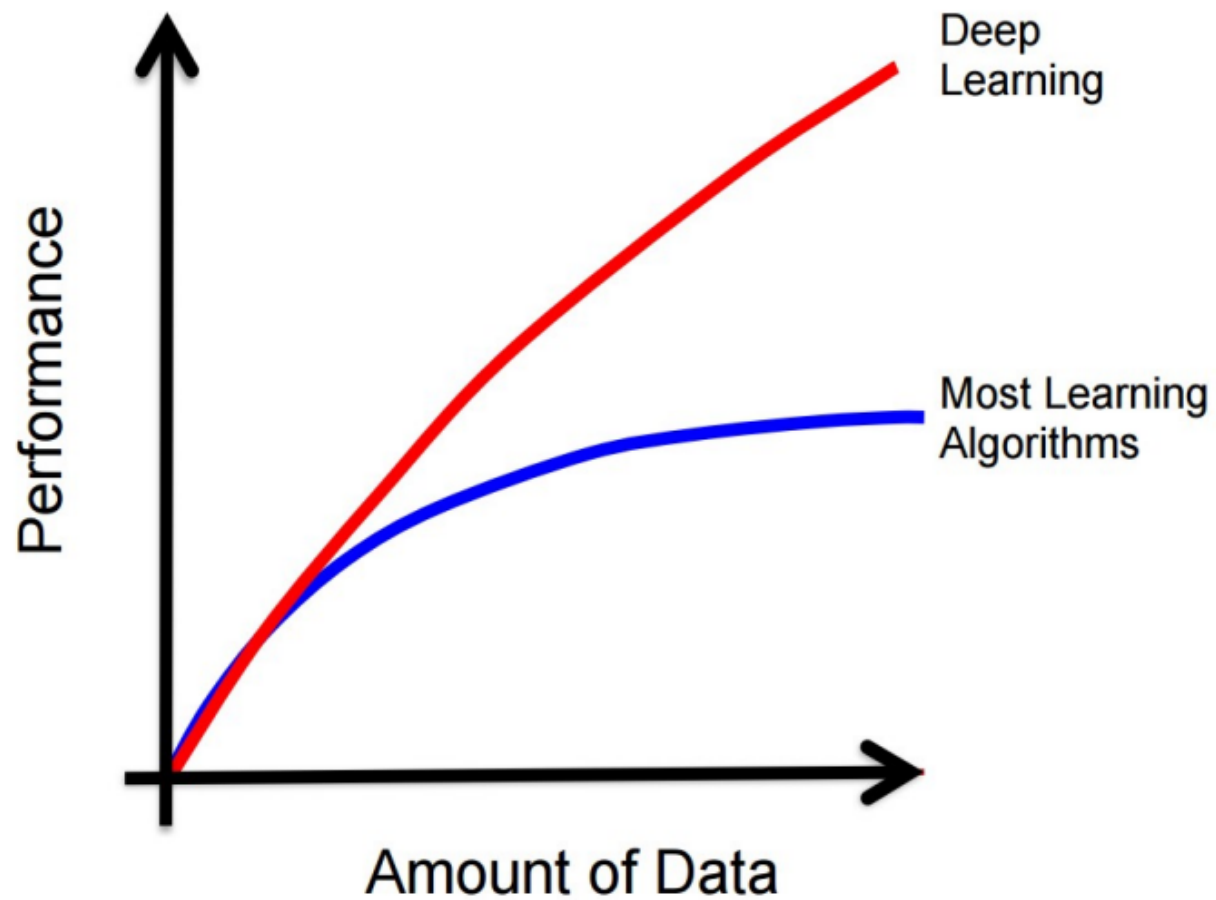
## 挑戰

- 資料收集
- 模型建立與選擇

# 概念



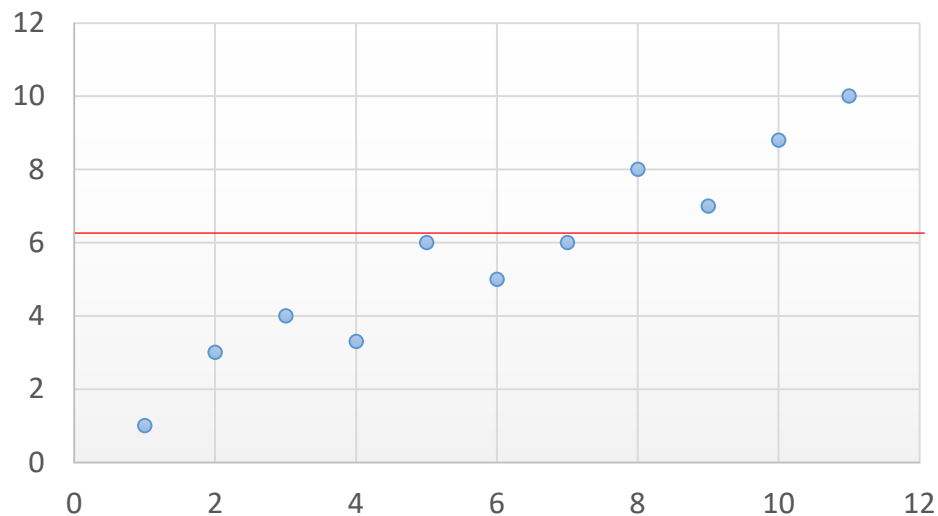
# 概念



# 簡單例子-回歸直線

- XY座標散佈，尋找回歸直線
  - 定義出直線 $y = a + bx$
  - 誤差函數(loss function) = 最小平方法

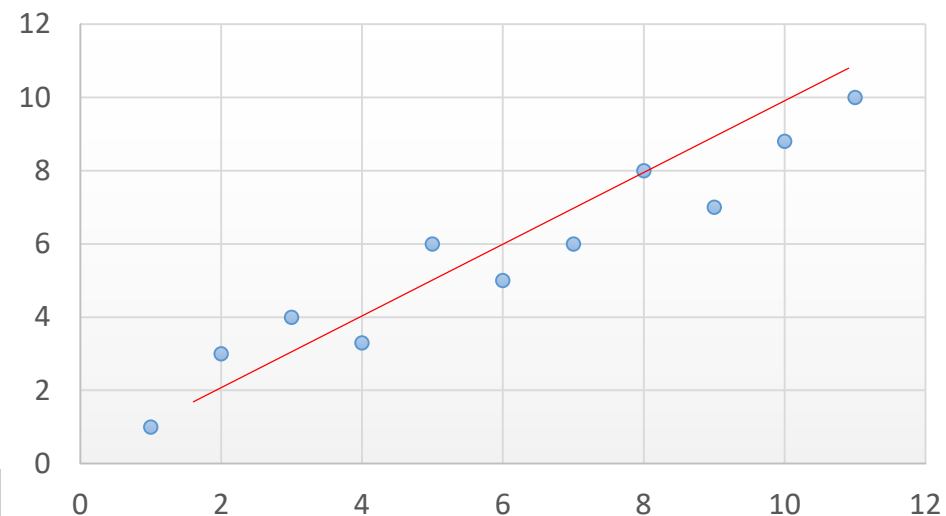
一開始



訓練  
尋找最小誤差

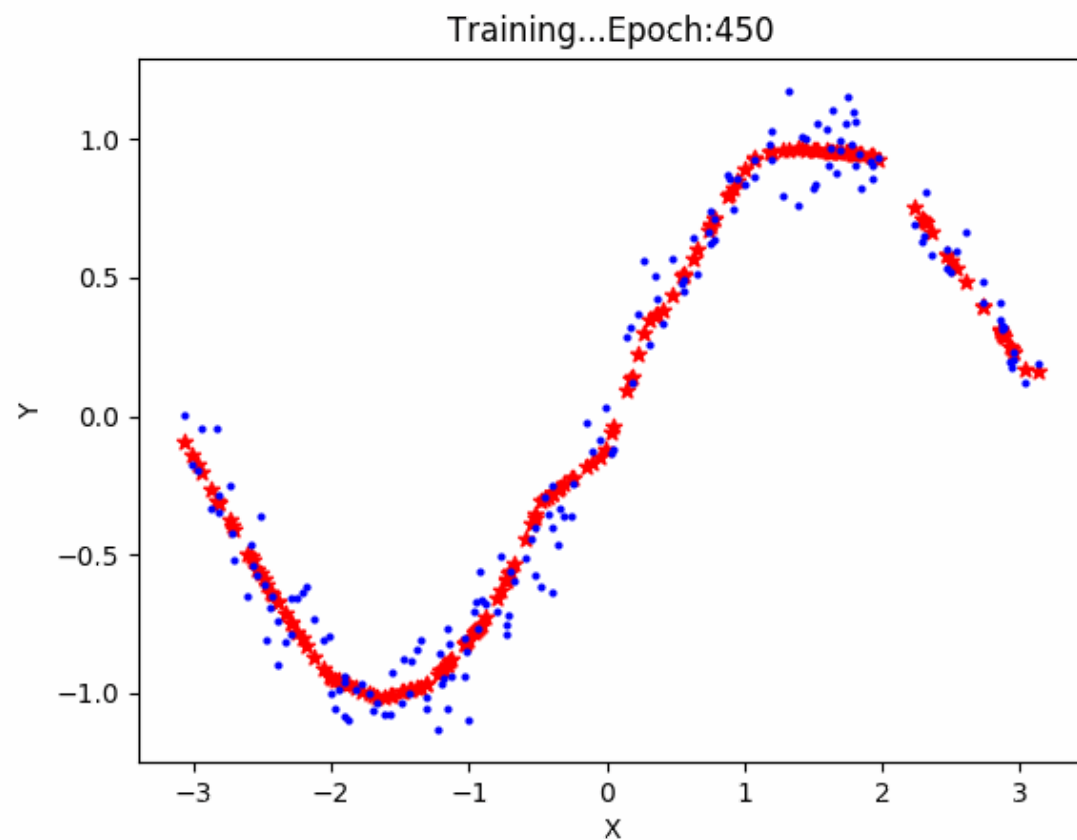


最終





# 簡單例子－回歸線



# 機器學習種類

真實數據(Ground truth):以  
分類為例就是標籤數據

## 監督式學習 supervised learning

- 輸入
- 真實數據

## 非監督式學習 unsupervised learning

- 輸入
- 無真實數據

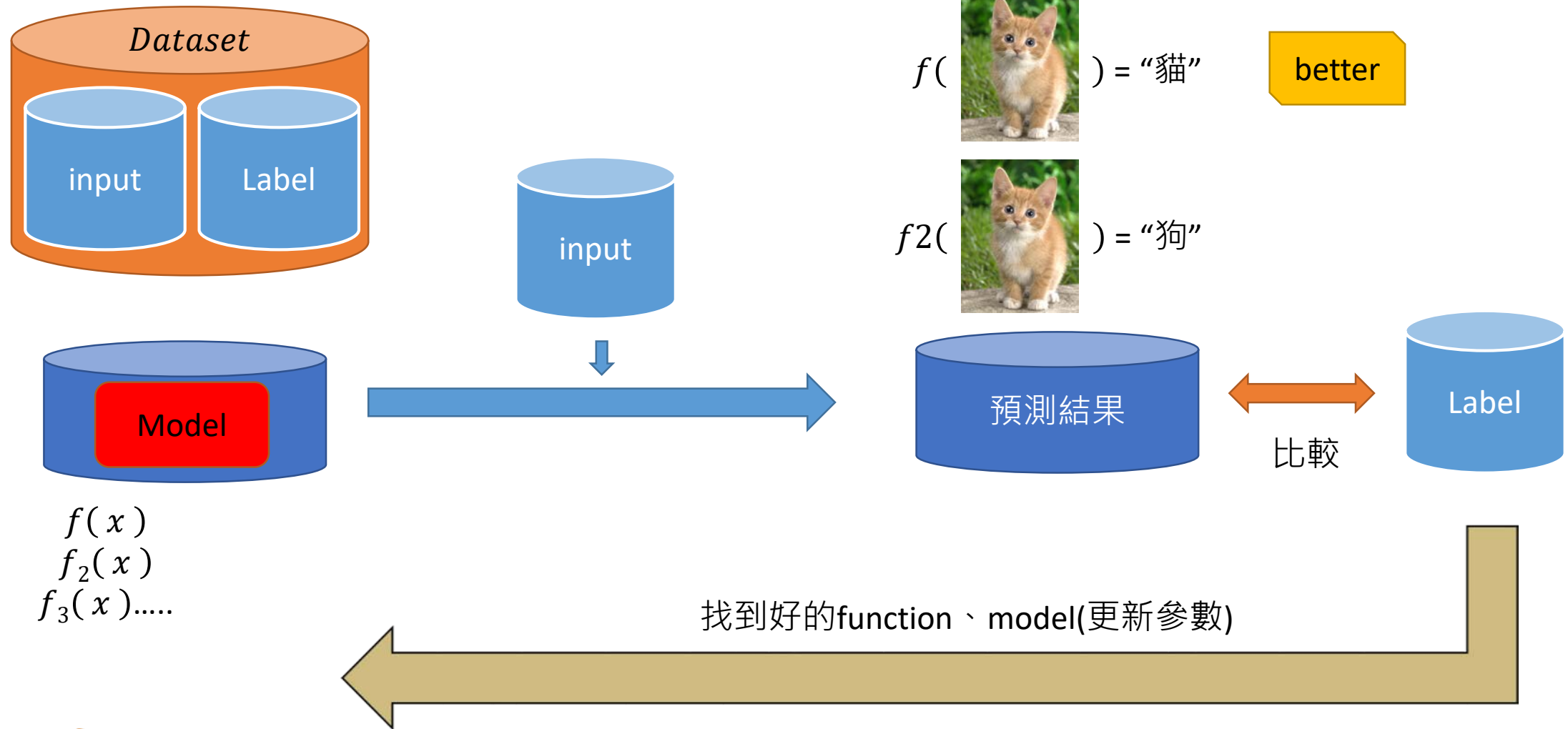
## 半監督式學習 semi-supervised learning

- 輸入
- 少部分擁有真實數據
- 大部分沒有真實數據

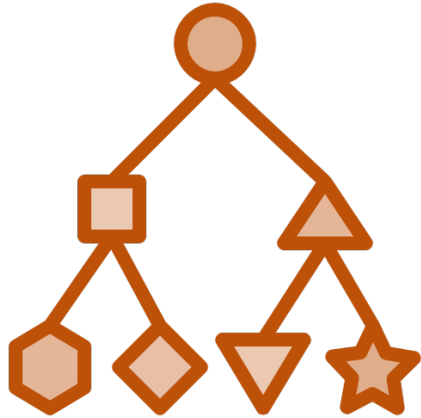
## 強化式學習 reinforcement learning

- 設定得分條件
- 從錯誤中學習
- 學習出最佳得分行為

# 監督式學習(supervised learning)

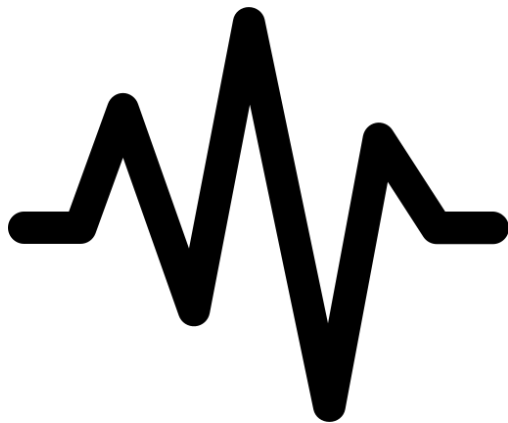


# 監督式學習(supervised learning)



分類

- 圖像分類
- 行為分類

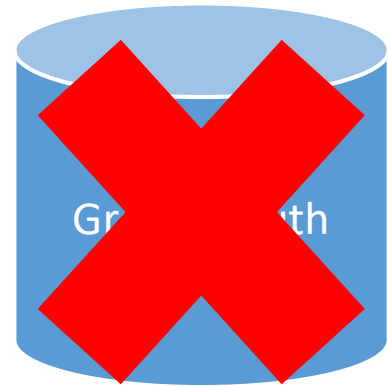


回歸

- 年齡預測
- 股市預測

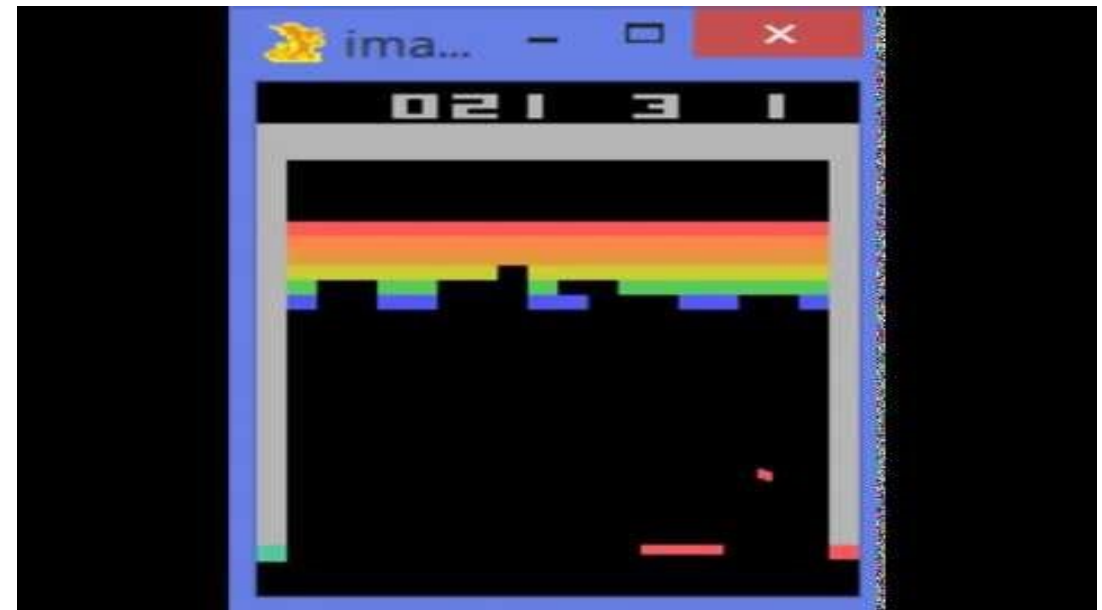
# 非監督式學習(unsupervised learning)

- Dataset 中 label(ground truth) 很難蒐集
- 非監督式不用label
- 讓網路自動學習特徵，進行分類



# 強化式學習(reinforcement learning)

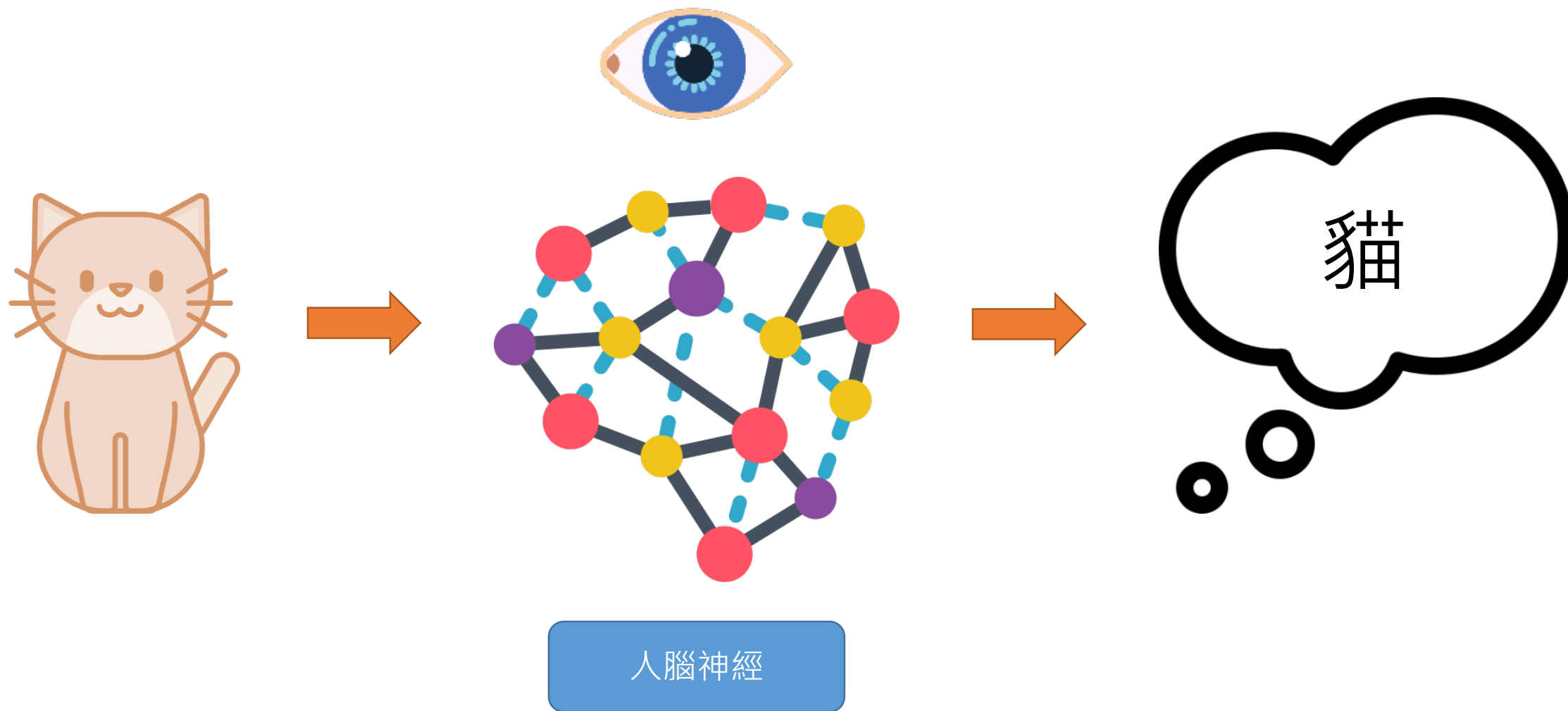
- 以評分條件來訓練
- 應用
  - Alphago
  - AI玩遊戲
- 舉例
  - Game over -> 低分
  - 破關 -> 高分
  - AI學習期中會導致破關的行為



# 機器學習

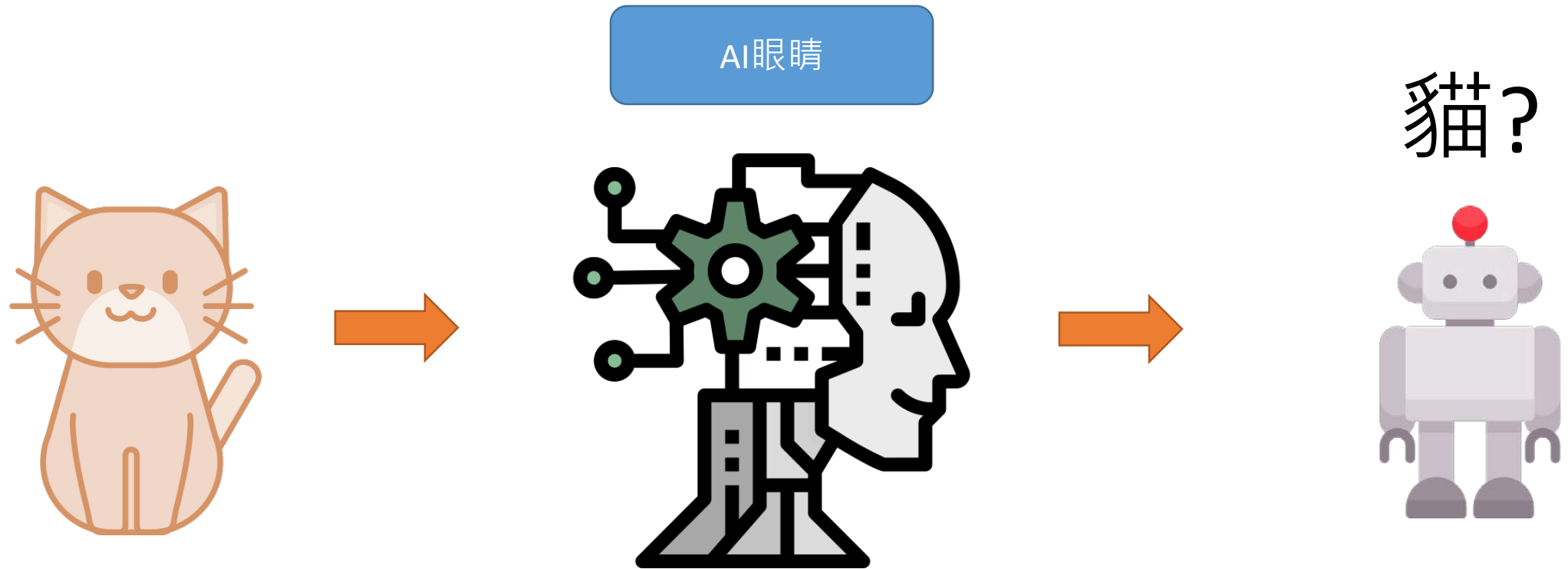
計算原理-分類舉例

# NN類神經原理



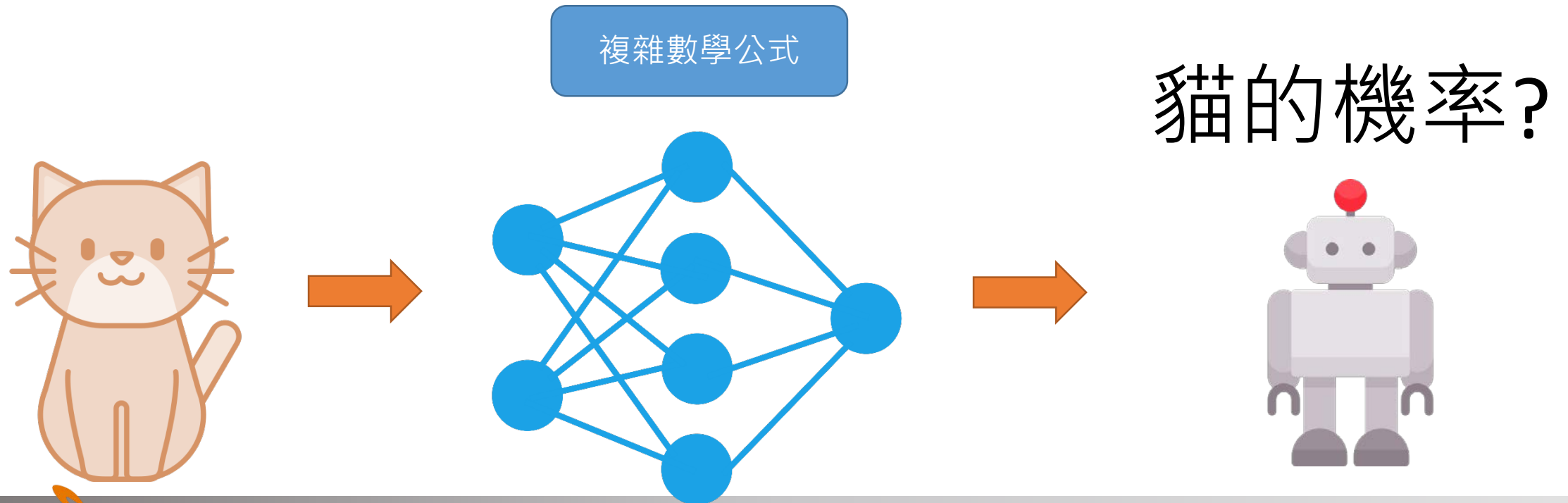


# NN類神經原理

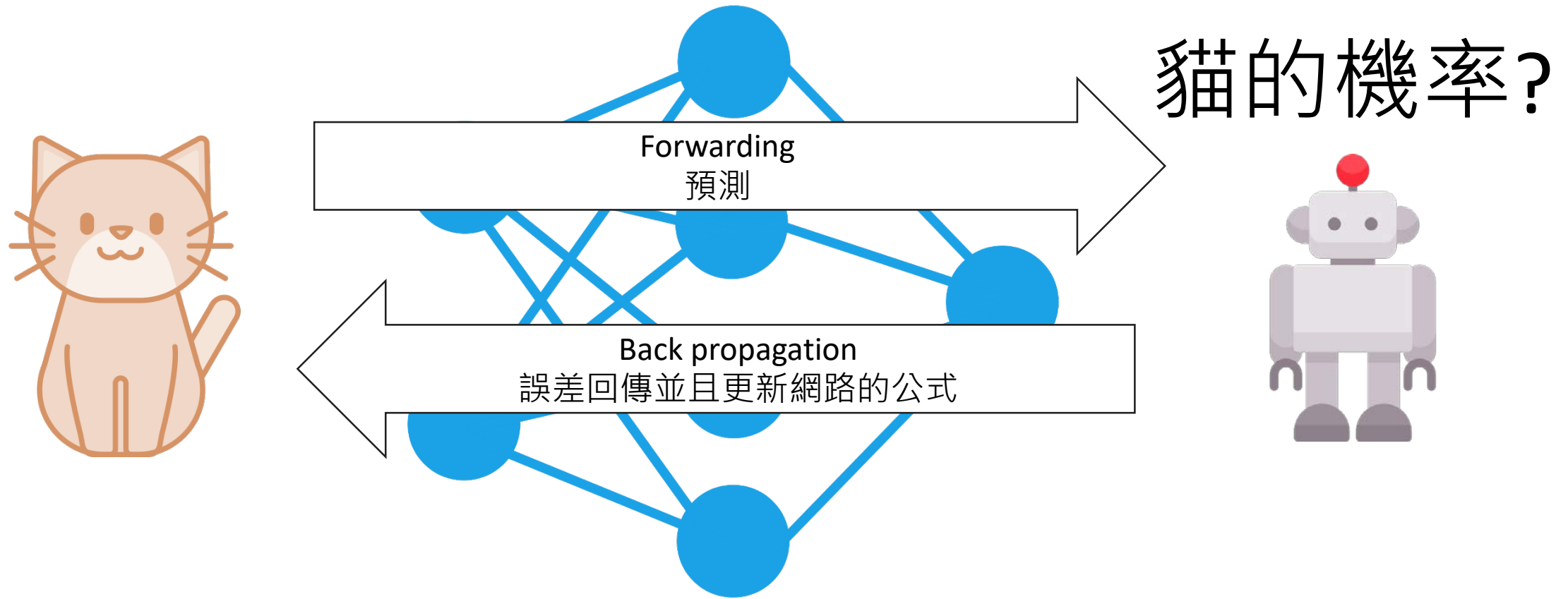


# NN類神經原理

- 將人類神經傳遞模擬成數學算式(類神經網路)
- 有公式、規則現象可用類神經網路推估

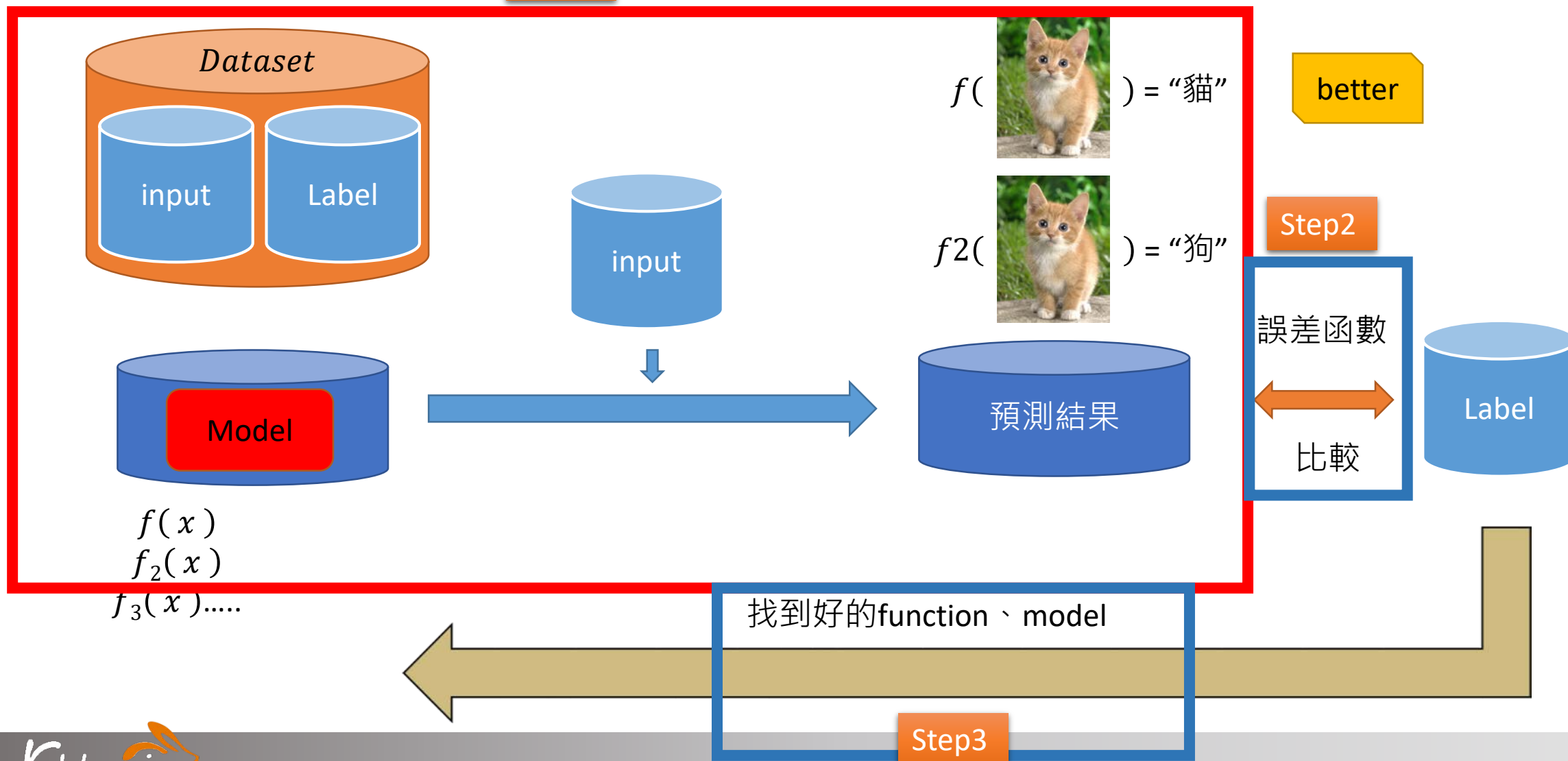


# Forwarding? Back propagation?



# 架構

Step1



# NN類神經-Forward Pass

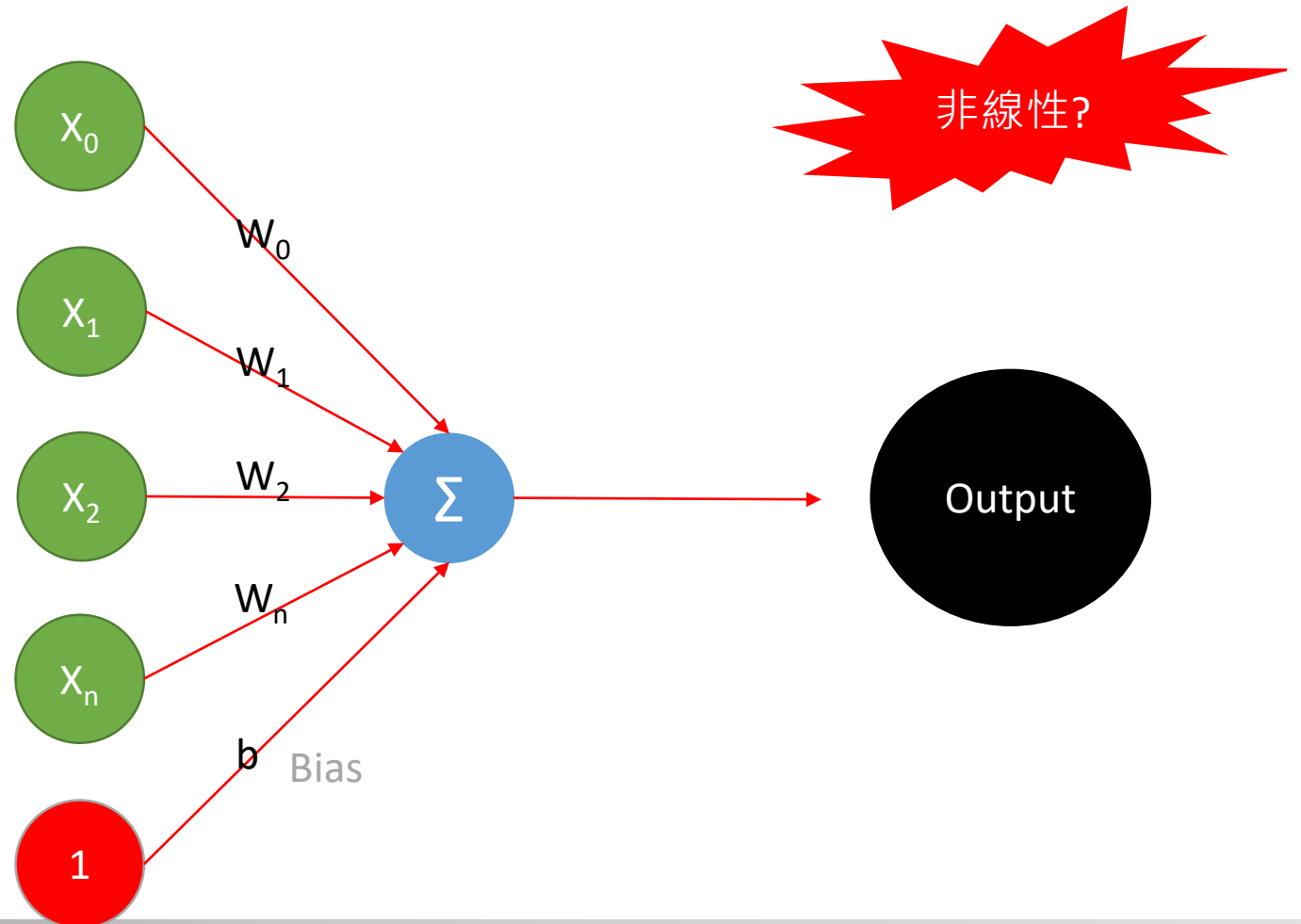
- NN基本公式

$$\text{Output} = \sum_{i=0}^N x_i \times W_i + b$$

$$X = x_0, x_1, \dots, x_n$$

$$W = w_0, w_1, \dots, w_n$$

Inputs      weights      sum



# NN類神經-Forward Pass

為了加上非線性模擬

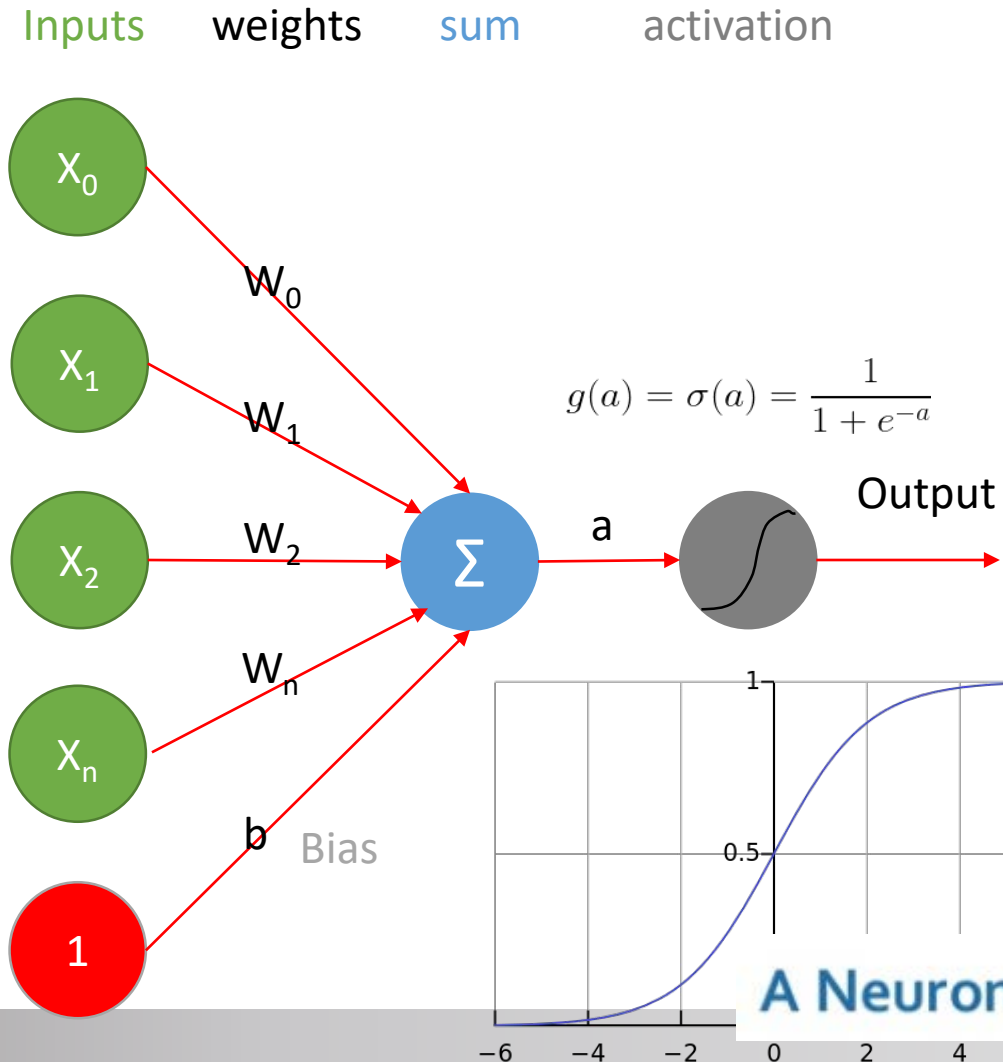
**Activation Function**

$$output = g\left(\sum_{i=0}^N x_i * w_i + b\right)$$

$$X = x_0, x_1, \dots, x_n$$

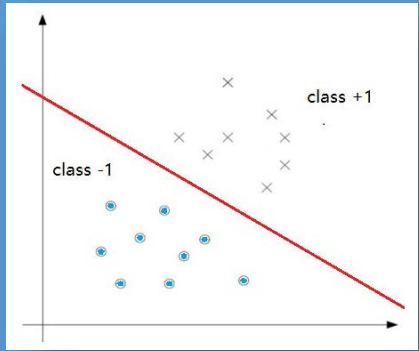
$$W = w_0, w_1, \dots, w_n$$

$$a = W_0X_0 + W_1X_1 + W_2X_2 + \dots + W_nX_n + b$$

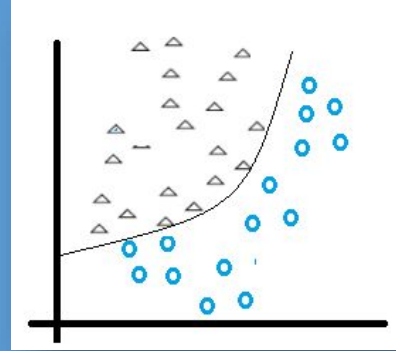


# 激發函數-Activation Function

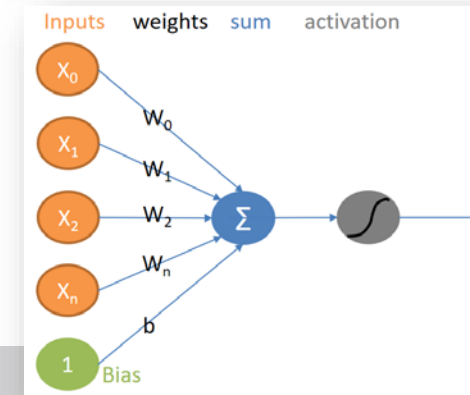
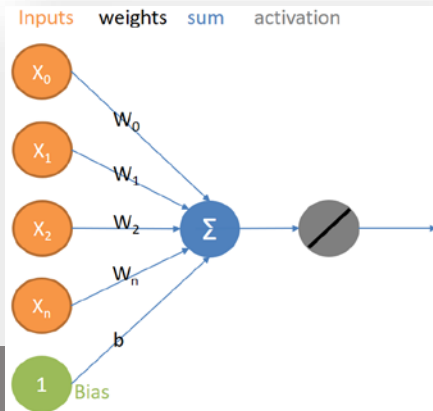
- 現實世界通常為非線性



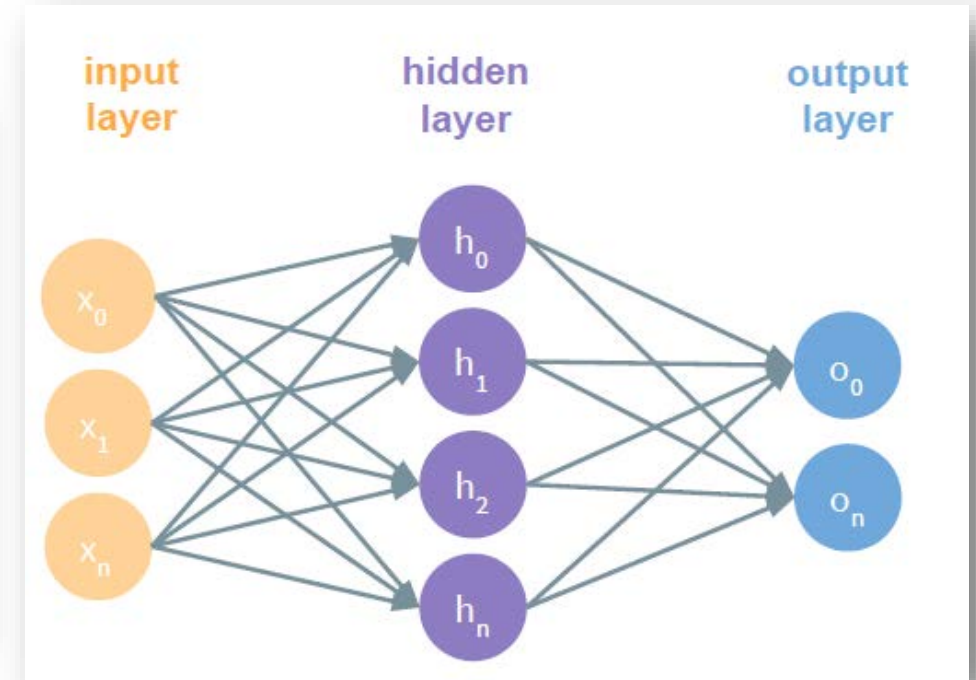
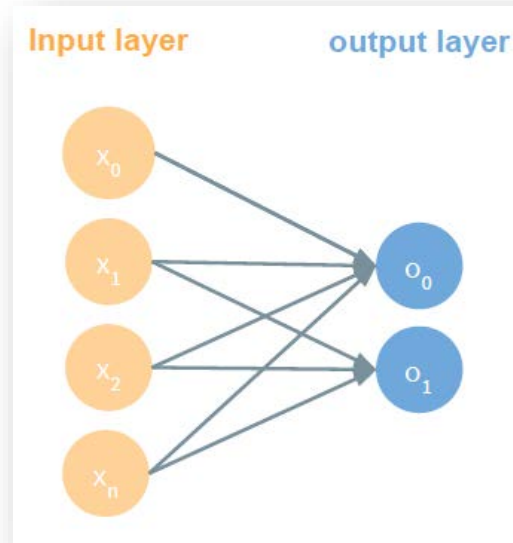
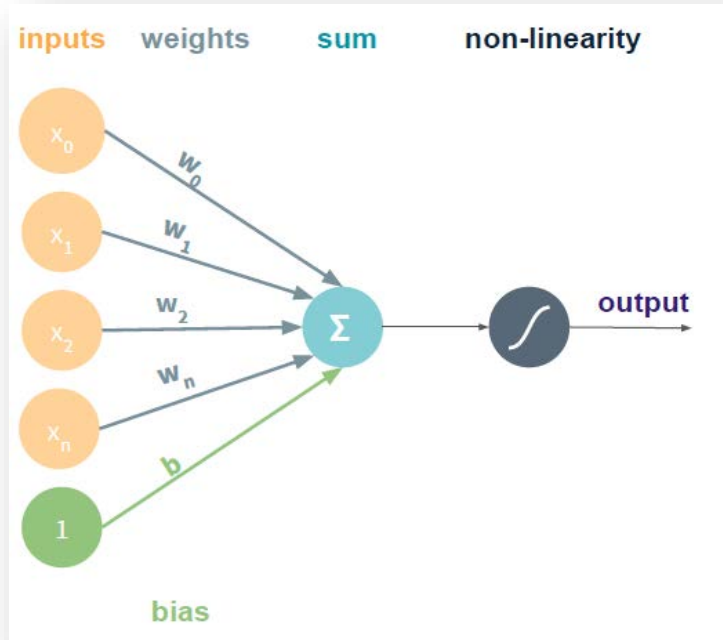
- 線性分類
- 可以利用 linear activation function 很簡單的將其分開



- 線性不可分
- 須利用 non-linear activation function 才能將其分開



# Fully Connected Neural Network

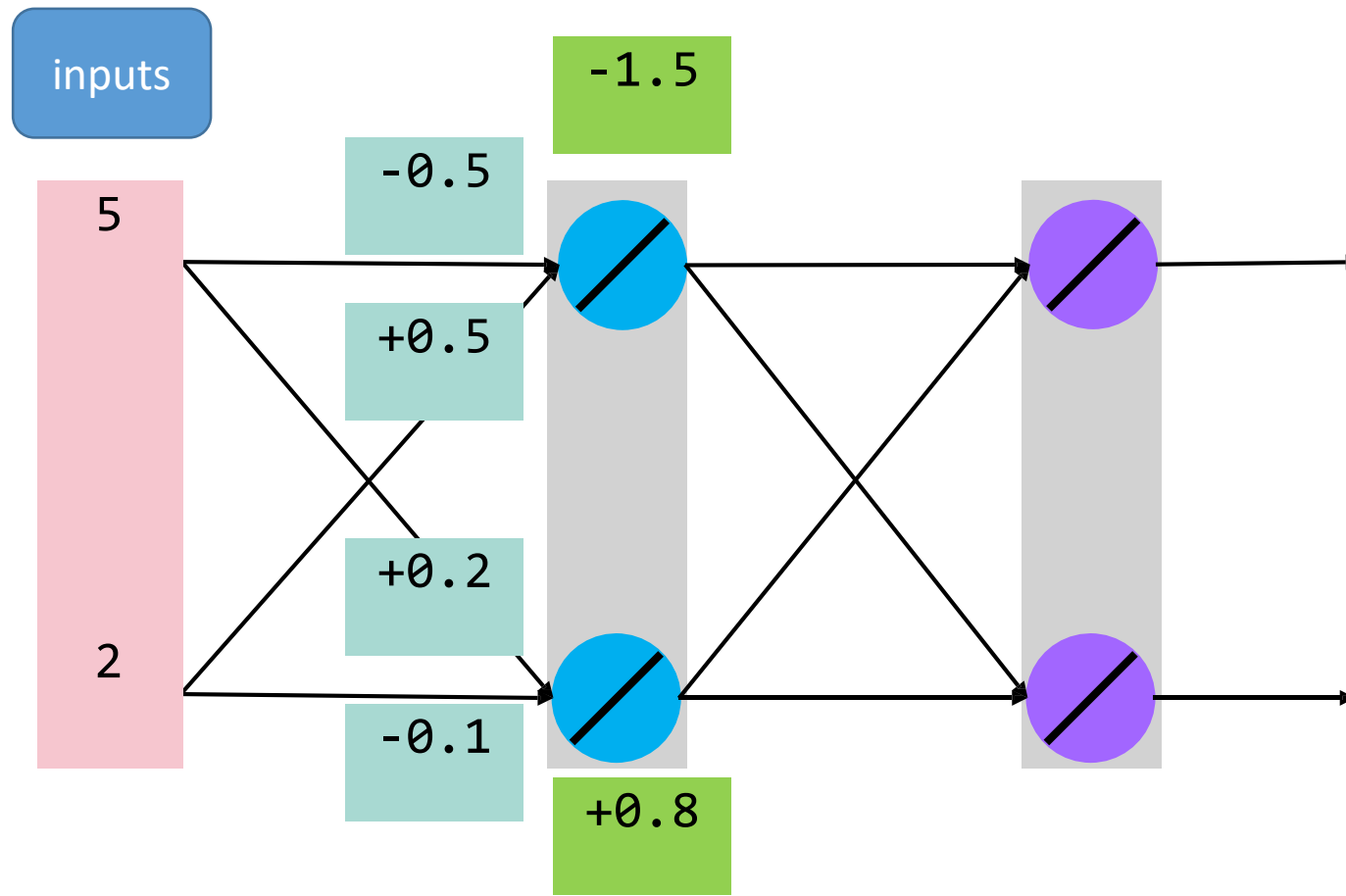


□ 很多個 neurons 連接成 network

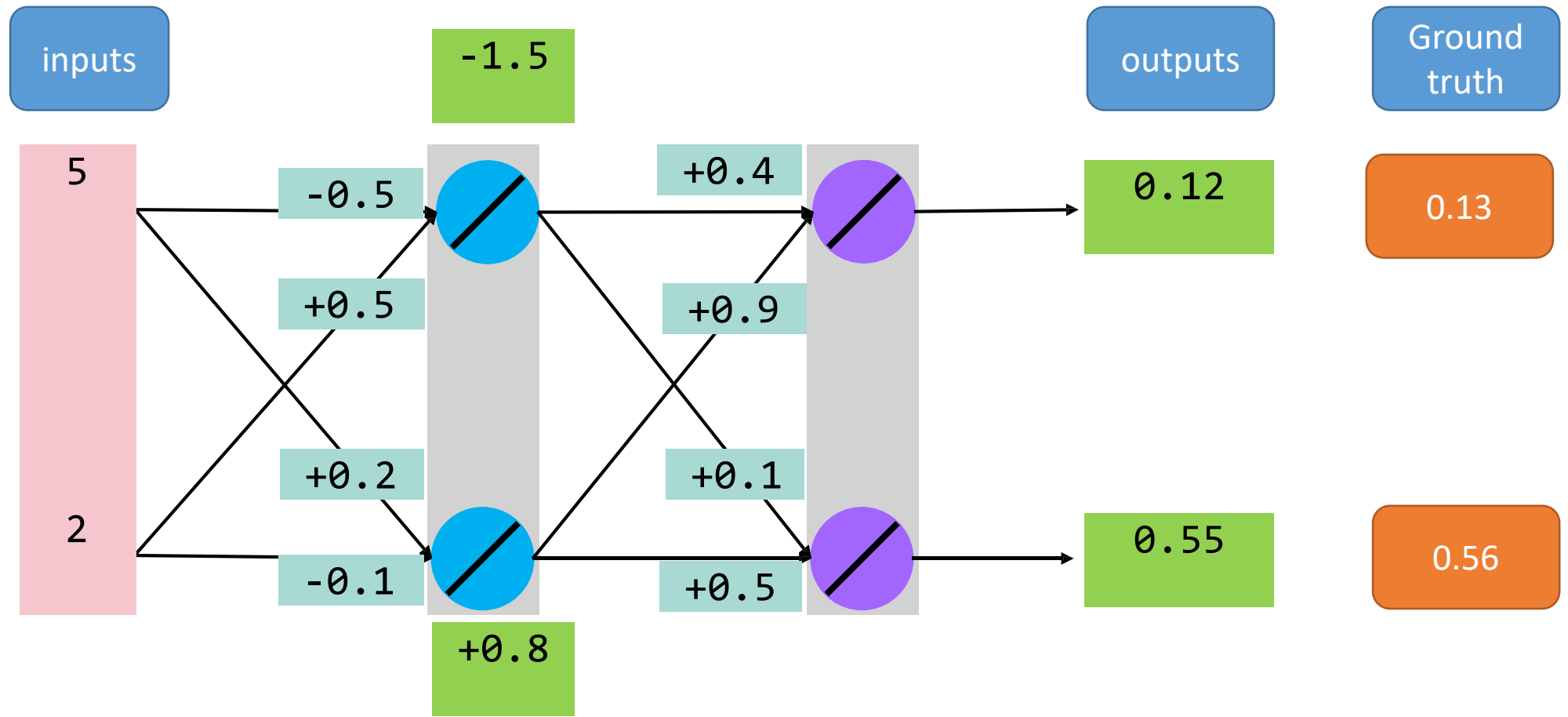
□ **Universal approximation theorem:**  
一層隱藏層的網路，可近似任意函數



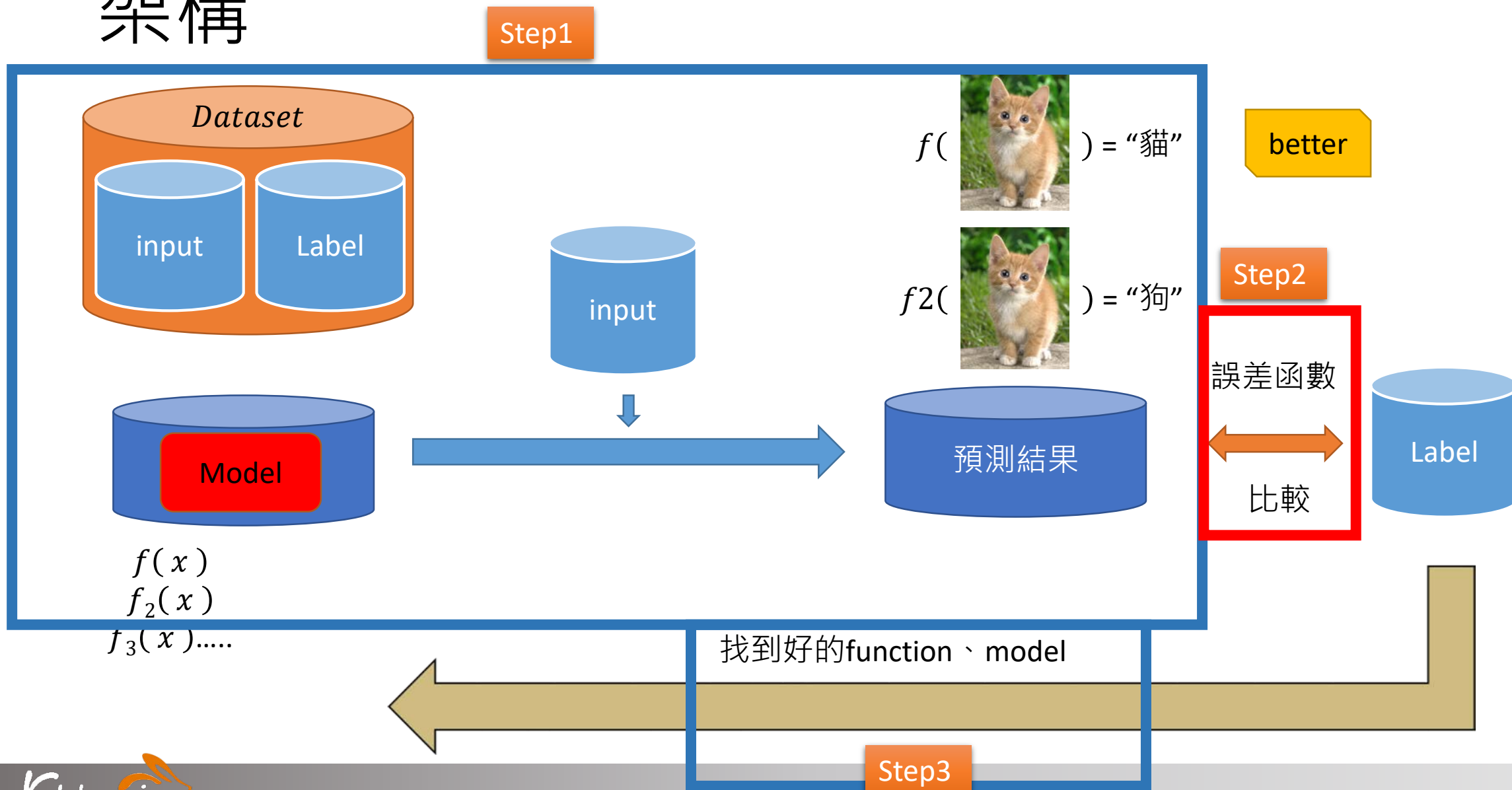
# Fully Connected Neural Network



# Fully Connected Neural Network



# 架構

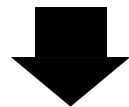


# 流程架構

定義方程式  
(模型)



評估  
&  
尋找



找到最佳  
方程式

特定的網絡架構

A set of functions,  $f(\cdot)$   
 $\{f(\theta_1), \dots, f(\theta^*), \dots, f(\theta_n)\}$

不斷修正  $f$  的參數

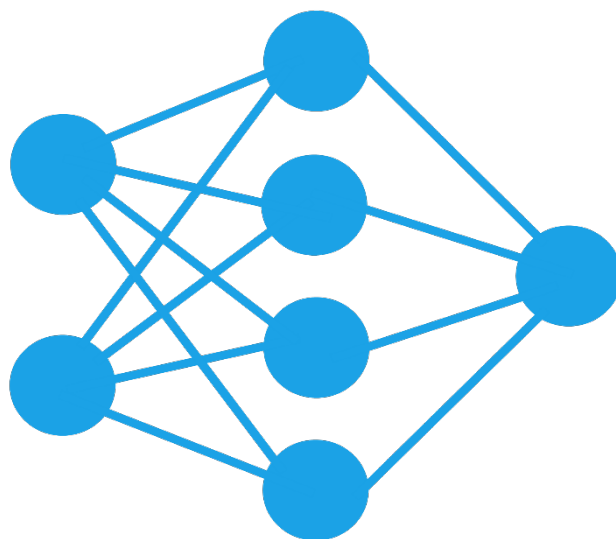
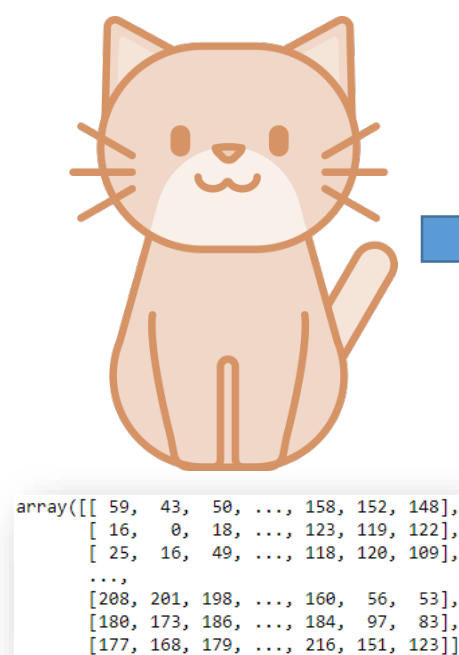
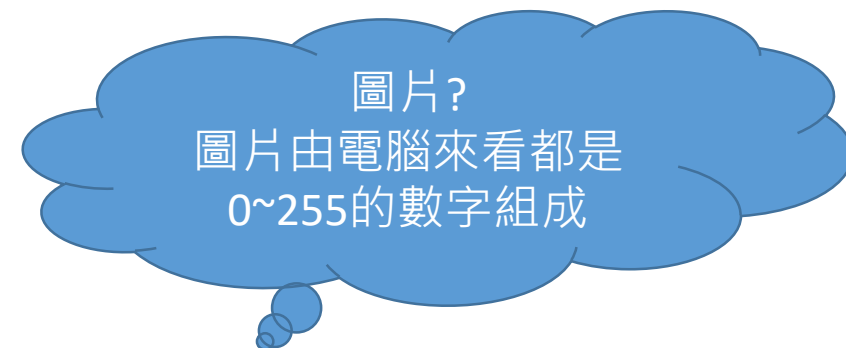
$f(\theta_1) \rightarrow$   
 $f(\theta_2) \rightarrow$   
 $f(\theta_9) \dots$

找到最適合的參數

$f(\theta^*)$

# 如何評估模型(loss function)

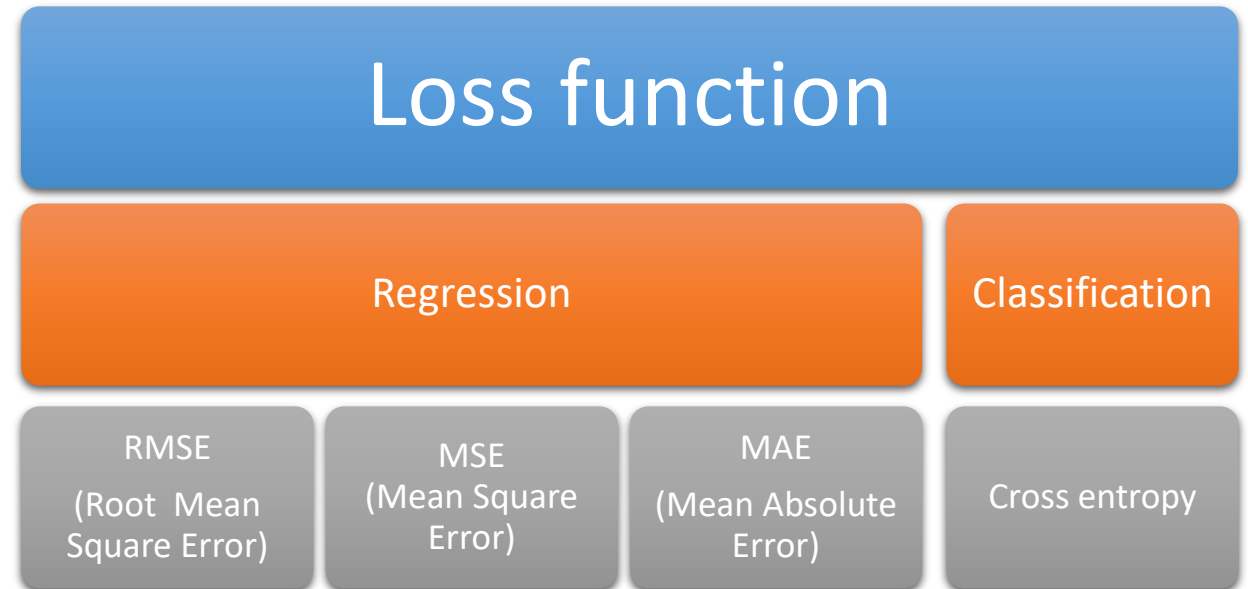
- 神經網路輸出值與ground truth越接近越好
- 計算誤差值:loss function 誤差函數



	輸出	Ground truth
貓的機率	0.9	1
狗的機率	0.1	0

# 誤差函數 (Loss function)

- 計算輸出值與ground truth 差多少
- 用處?
  - 找到最佳網路的參數 $\theta$ ，使loss最小
- 如何找 $\theta$ ?
  - 梯度下降 Gradient descent
- 常見loss function
  - 回歸:RMSE、MSE、MAE
  - 分類:cross entropy



# 誤差函數 (Loss function)

## Loss function

### Regression

### Classification

RMSE

(Root Mean Square Error)

MSE

(Mean Square Error)

MAE

(Mean Absolute Error)

Cross entropy

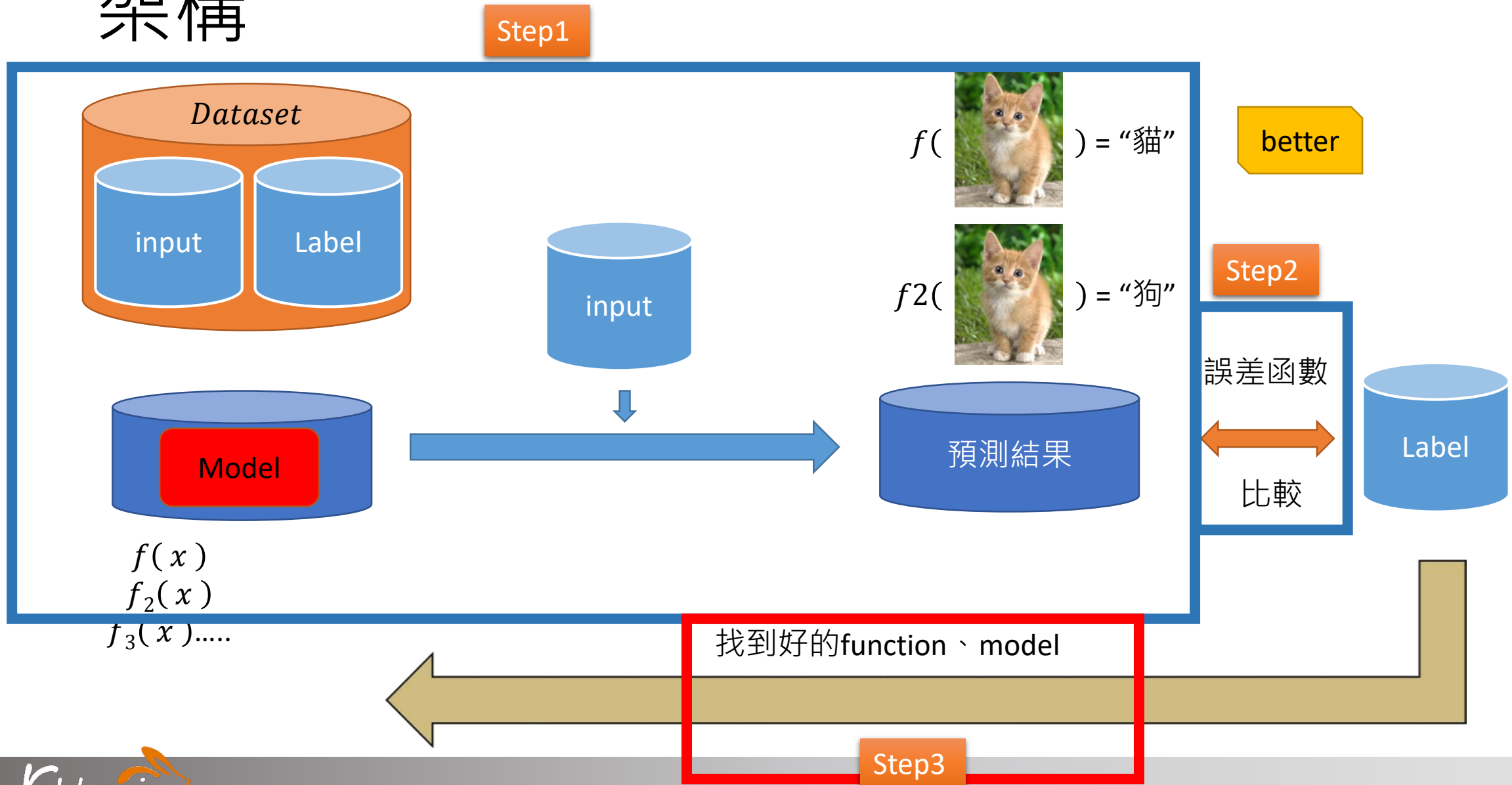
$$\text{RMSE} = \sqrt{\frac{1}{m} \sum_{i=0}^n (y_i - g_i)^2}$$

$$\text{MSE} = \frac{1}{m} \sum_{i=0}^n (y_i - g_i)^2$$

$$\text{MAE} = \frac{1}{m} \sum_{i=0}^n |y_i - g_i|$$

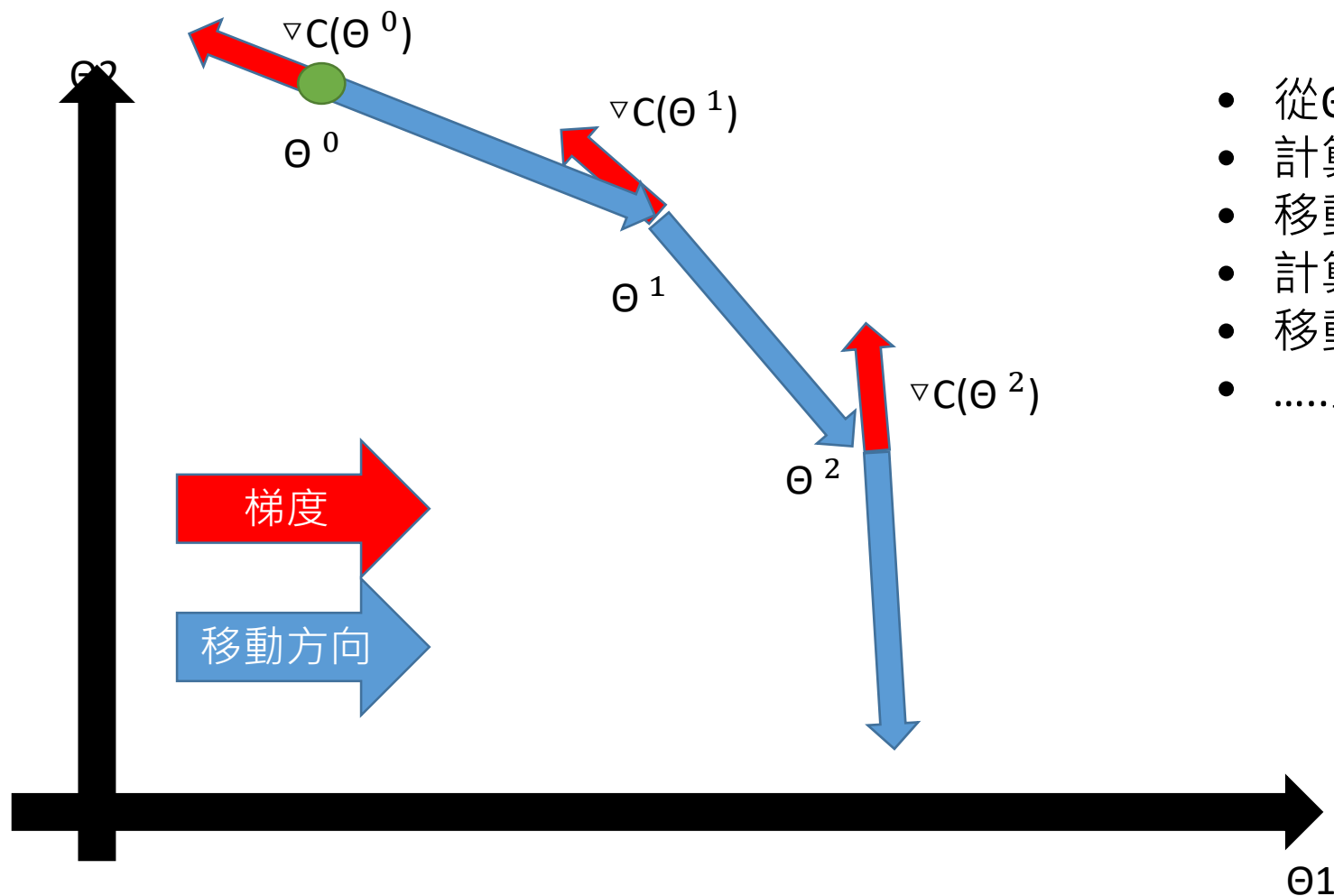
$$\text{C.E} = -[g * \log(y) + (1 - g) *$$

# 架構





# 梯度下降(Gradient Descent)



- 從 $\theta^0$ 開始
- 計算 $\theta^0$ 的梯度
- 移動到 $\theta^1 = \theta^0 - \eta \nabla C(\theta^0)$
- 計算 $\theta^1$ 的梯度
- 移動到 $\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$
- .....重複直到Loss最小

# 梯度下降(Gradient Descent)

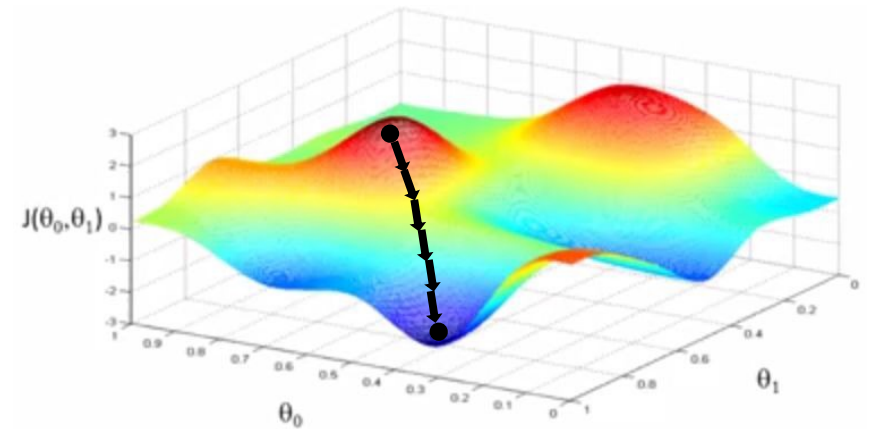
- $\Theta$ (模型內參數)一開始是隨機選擇的
- 訓練N epochs(次)
  - 計算 損失函數(Loss function)的梯度(gradient)
  - 以學習率去更新 $\Theta$

$$\frac{\partial L}{\partial \theta}$$

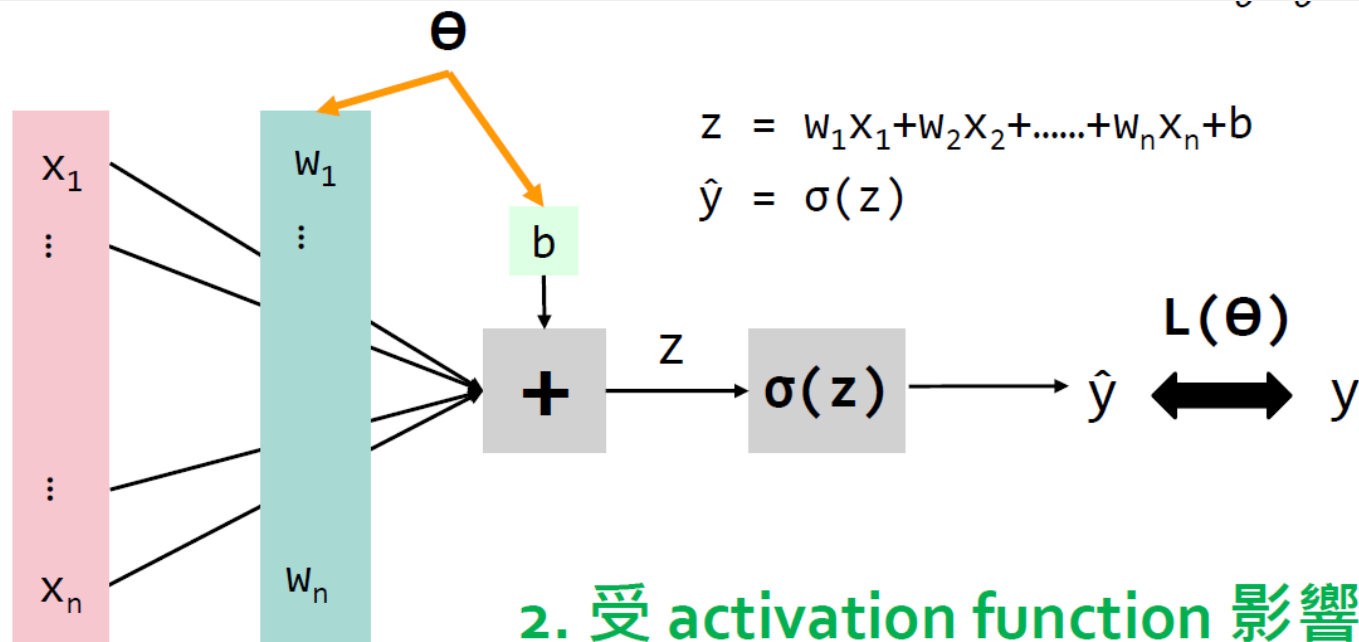
$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$

沿著 gradient 的反方向走

Learning rate



# 影響Gradient 的因素



$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} = \boxed{\frac{\partial L}{\partial \hat{y}}} \boxed{\frac{\partial \hat{y}}{\partial z}} \frac{\partial z}{\partial \theta}$$

1. 受 loss function 影響

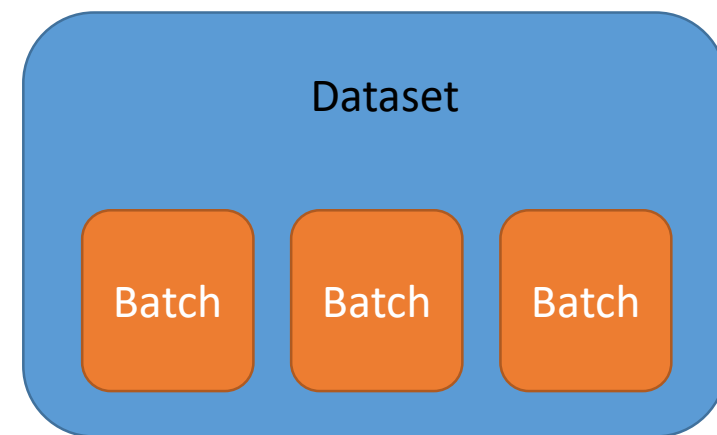
- 受 Loss function 影響
- 受 activation function 影響

# 梯度下降(Gradient Descent)的缺點

- 一個 epoch 更新一次，收斂速度很慢
  - 一個 epoch 等於看過所有 training data 一次

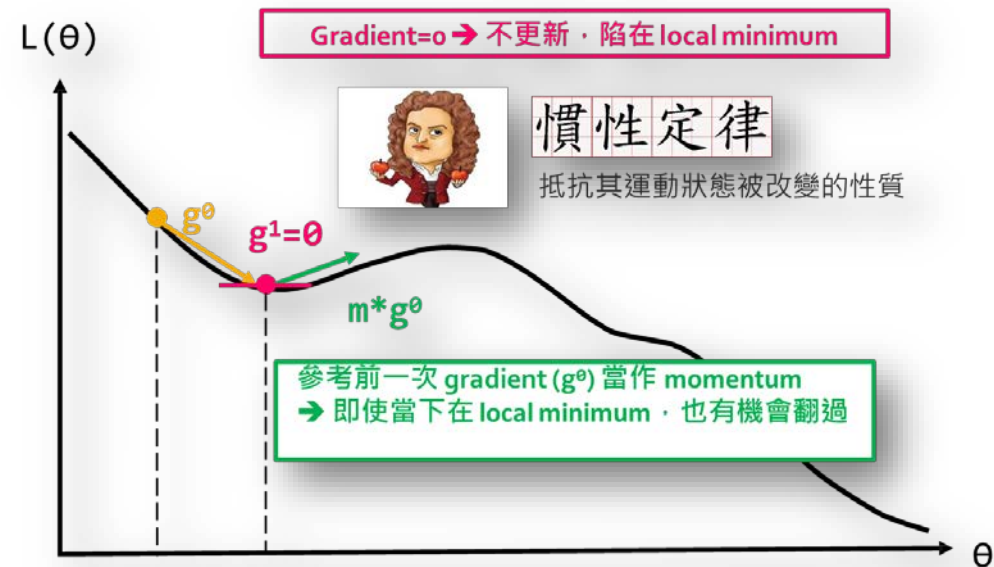
## 加速方法

- 隨機抽一筆資料 stochastic gradient descent (SGD)
  - 一筆一筆更新也很慢
- 一次看多筆資料 **mini-batch SGD (主流方法)**
  - 依照 mini-batch 把所有 training data 拆成多份
  - 假設全部有 1000 筆資料
    - Batch size = 100 可拆成 10 份，一個 epoch 內會更新 10 次
    - Batch size = 10 可拆成 100 份，一個 epoch 內會更新 100 次
  - 如何設定 batch size?
    - 常用 32, 128, 256, ...

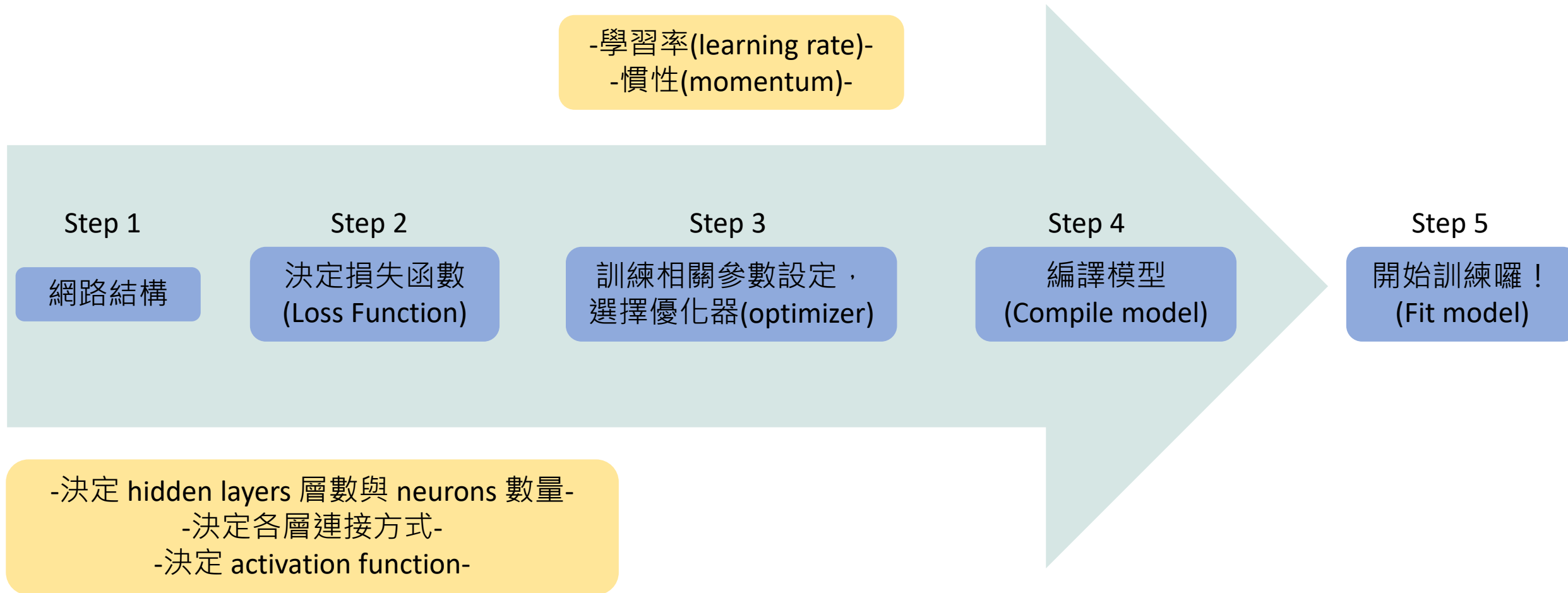


# 梯度下降(Gradient Descent)的缺點

- Gradient based method 不能保證找到全域最佳解
  - 可以利用 momentum 降低困在 local minimum(or saddle point) 的機率
  - Momentum
- 所有參數共用learning rate
  - 個別參數都有learning rate
    - Adagrad/Adadelata/ RMSprop
- 結合動量+adaptive learning rate
  - Adam



# 五步驟建立架構



# 實戰教學

神經網路 with keras

High level API

**Keras**

Fast.ai

Framework

Tensorflow

Pytorch

Caffe

chainer

MXNet

CNTK

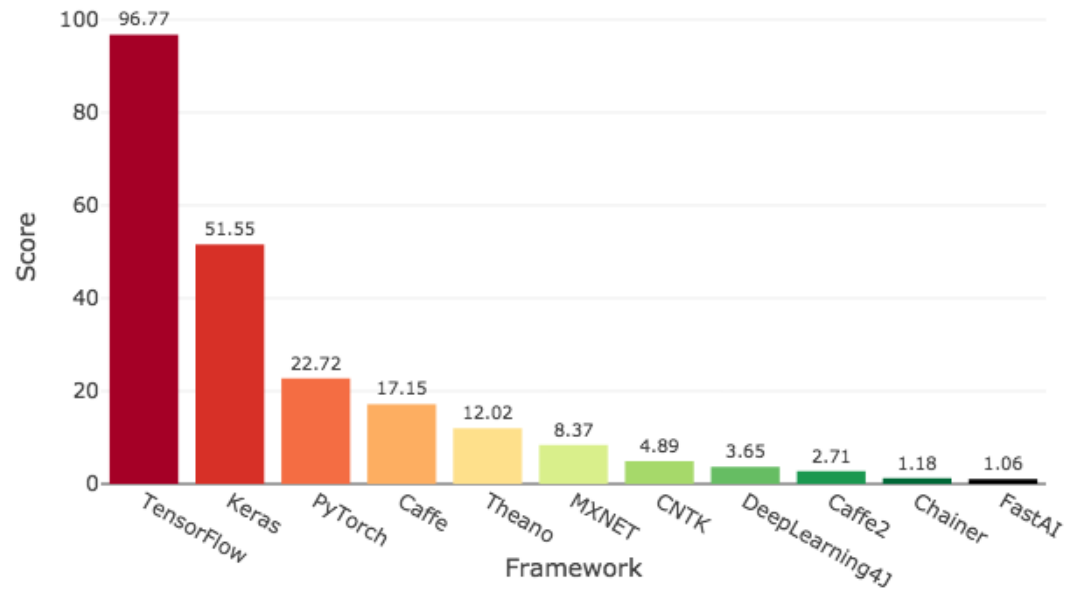
Language specific

Matconvnet

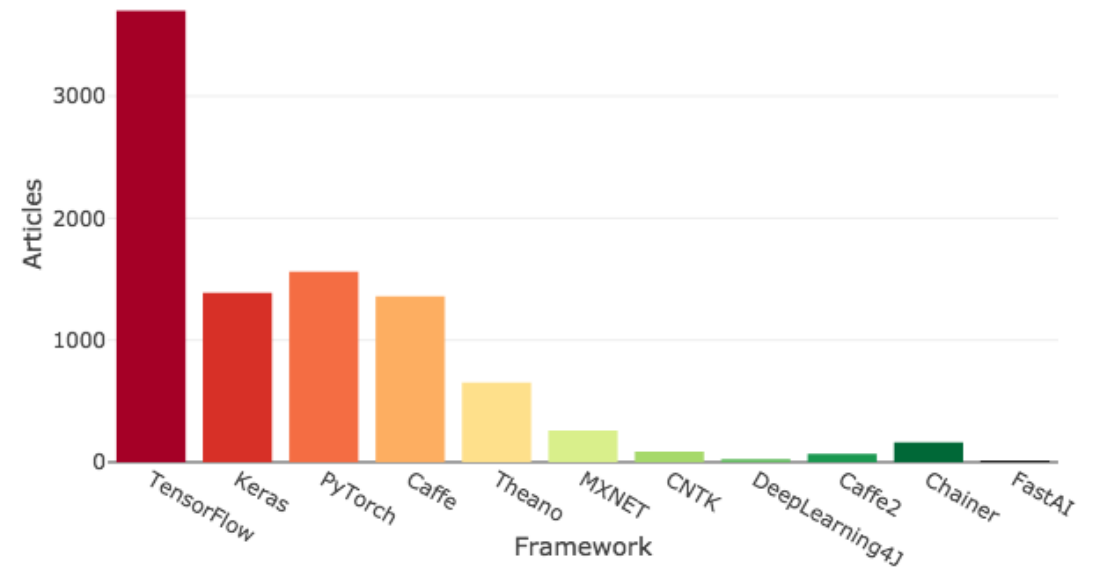
Deeplearning4j



Deep Learning Framework Power Scores 2018



ArXiv Articles



# AI相關工具

- 架模型
  - 牽扯到許多微積分等公式
  - 自己架設....耗時麻煩
- 現有的工具
  - Caffe、Caffe2
  - theano
  - TensorFlow
  - Pytorch
  - **Keras** : 建立在TensorFlow或theano上，適合初學者

Caffe

theano



PYTORCH

# keras?

- 站在巨人肩膀上
  - 包裝tensorflow與theano，較容易讓初學者理解
  - 容易架設
  - 程式簡單，數行可以達到Tensorflow數十行的功能
- 缺點
  - 速度較tensorflow慢
  - 較難實現特殊架構



# 輸入前處理

- 電腦只看得懂數字
  - 把類別換成label : One-hot encoding
    - 3種類別，依序為[貓,狗,猴子]
    - 貓的label(ground truth) 為 [1,0,0]
- 若數字範圍差太多
  - re-scale或normalization 至0~1 or -1~1
    - 距離:0~10000m
    - 全部除10000 將數值化為0~1

```
array([[ 59,  43,  50, ..., 158, 152, 148],  
       [ 16,   0,  18, ..., 123, 119, 122],  
       [ 25,  16,  49, ..., 118, 120, 109],  
       ...,  
       [208, 201, 198, ..., 160,  56,  53],  
       [180, 173, 186, ..., 184,  97,  83],  
       [177, 168, 179, ..., 216, 151, 123]])
```



[0.9,0.05,0.05]

# 1. 網路結構

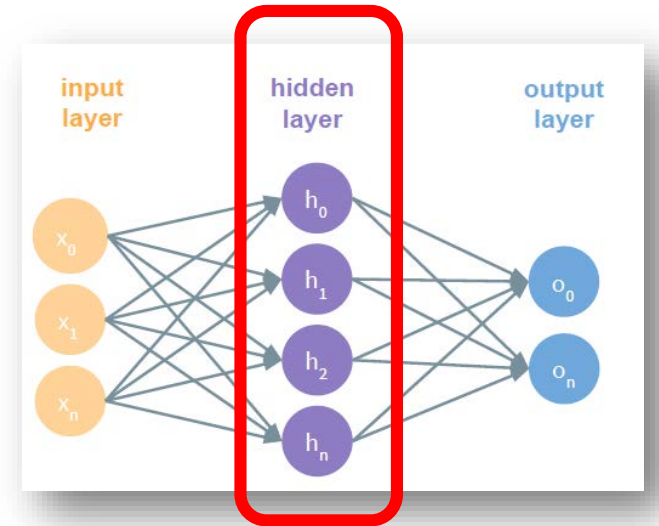
```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

# 宣告這是一個 Sequential 次序性的深度學習模型
model = Sequential()

# 加入第一層 hidden layer (128 neurons)
# [重要] 因為第一層 hidden layer 需連接 input vector
# 故需要在此指定 input_dim
model.add(Dense(128, input_dim=200))
```

Model 建構時，是以次序性的疊加(add) 上去

Dense(kernel 數)  
就是fully connect層

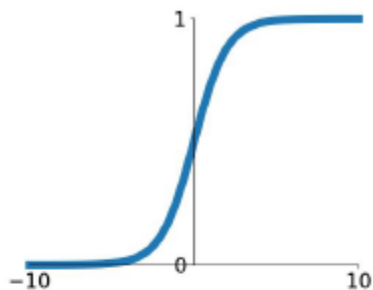


# 1-2 Activation Functions

基本款，深層網路不用

## Sigmoid

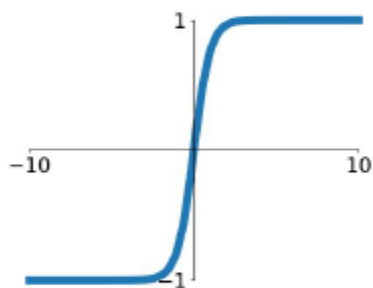
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



基本款，深層網路不用

## tanh

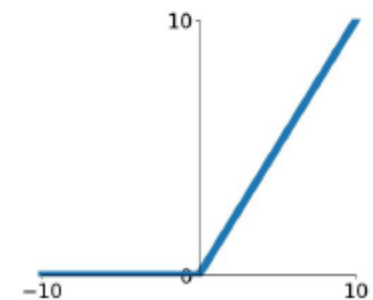
$$\tanh(x)$$



## ReLU

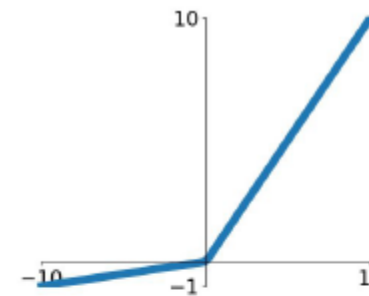
$$\max(0, x)$$

最常用



## Leaky ReLU

$$\max(0.1x, x)$$

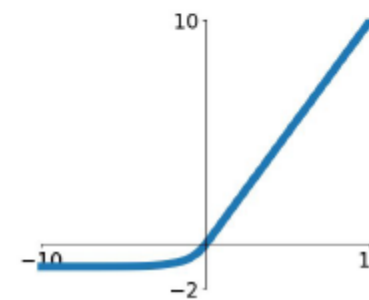


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# 網路結構 (Conti)

```
# 宣告這是一個 Sequential 次序性的深度學習模型
model = Sequential()

# 加入第一層 hidden layer (128 neurons) 與指定 input 的維度
model.add(Dense(128, input_dim=200))
# 指定 activation function
model.add(Activation('sigmoid'))

# 加入第二層 hidden layer (256 neurons)
model.add(Dense(256))
model.add(Activation('sigmoid'))

# 加入 output layer (5 neurons)
# 分五類
model.add(Dense(5))
model.add(Activation('softmax'))

# 觀察 model summary
model.summary()
```

## Softmax?

可以將一系列數值壓縮至 0~1 並且合為1，類似於機率表示

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

# Model Summary

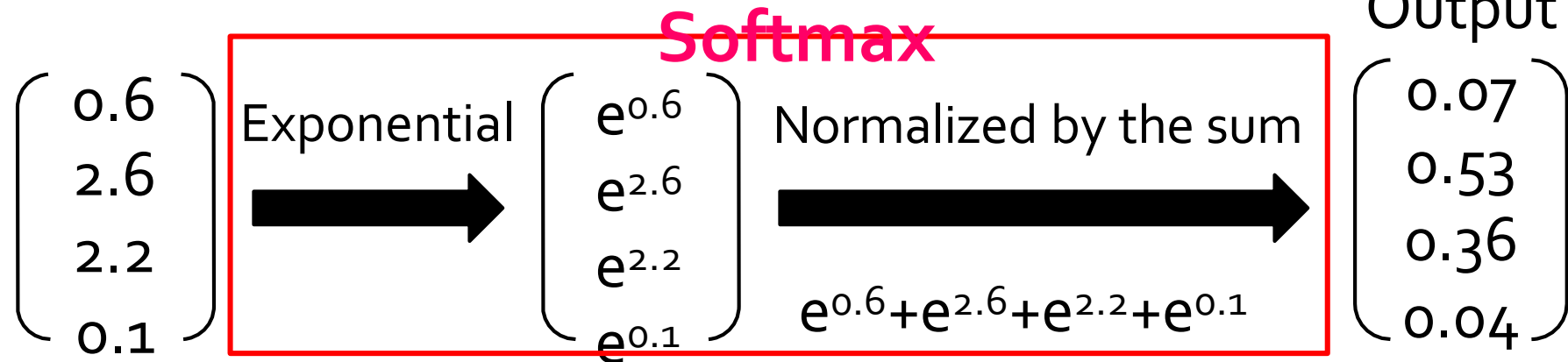
- 可以看到架設好的網路架構與參數量

Layer (type)	Output Shape	Param #	Connected to
dense_10 (Dense)	(None, 128)	25728	dense_input_4[0][0]
activation_10 (Activation)	(None, 128)	0	dense_10[0][0]
dense_11 (Dense)	(None, 256)	33024	activation_10[0][0]
activation_11 (Activation)	(None, 256)	0	dense_11[0][0]
dense_12 (Dense)	(None, 5)	1285	activation_11[0][0]
activation_12 (Activation)	(None, 5)	0	dense_12[0][0]
Total params: 60037			



## 1-2 輸出層Activation

- Classification 常用 softmax 當 output 的 activation function



- Normalization: network output 轉換到[0,1] 之間且 softmax output 相加為 1 → 像 “機率”
- 保留對其他 classes 的 prediction error

## 2. 決定模型的 loss function: 分類問題

- binary\_crossentropy (logloss): 兩類

- $-\frac{1}{N} \sum_{n=1}^N [y_n \log(\hat{y}_n) + (1 - y_n) \log(1 - \hat{y}_n)]$
- $-\frac{1}{2} [0 * \log(0.9) + (1 - 0) * \log(1 - 0.9) + 1 * \log(0.1) + 0 * \log(1 - 0.1)]$
- $= -\frac{1}{2} [\log(0.1) + \log(0.1)] = -\log(0.1) = 2.302585$

Prediction	Answer
0.9	0
0.1	1

## 2. 決定模型的 loss function: regression 問題

- mean\_squared\_error
  - $[(0.9 - 0.8)^2 + (0.1 - 0.2)^2] / 2 = 0.01$
- mean\_absolute\_error
  - $[|0.9 - 0.8| + |0.1 - 0.2|] / 2 = 0.1$
- mean\_absolute\_percentage\_error
  - $[|0.9 - 0.8| / |0.9| + |0.1 - 0.2| / |0.1|] * 100 / 2 = 55$
- mean\_squared\_logarithmic\_error
  - $[\log(0.9) - \log(0.8)]^2 + [\log(0.1) - \log(0.2)]^2 * 100 / 2 = 0.247$

Prediction	Answer
0.8	0.9
0.2	0.1

### 3. 選擇 optimizer 與參數

- 基本款 optimizer SGD(Stochastic gradient descent optimizer)

```
# 指定 optimizer  
from keras.optimizers import SGD, Adam, RMSprop, Adagrad  
sgd = SGD(lr=0.01, momentum=0.0, decay=0.0)
```

- lr: 學習率
- momentum: 慣性
- decay: 學習率是否會慢慢衰退
- 動量型: Momentum/Nesterov Momentum
- 個別參數調整型: RMSprop
- 混合動量型與個別參數調整型: Adam

## 4. 編譯模型 (Compile model)

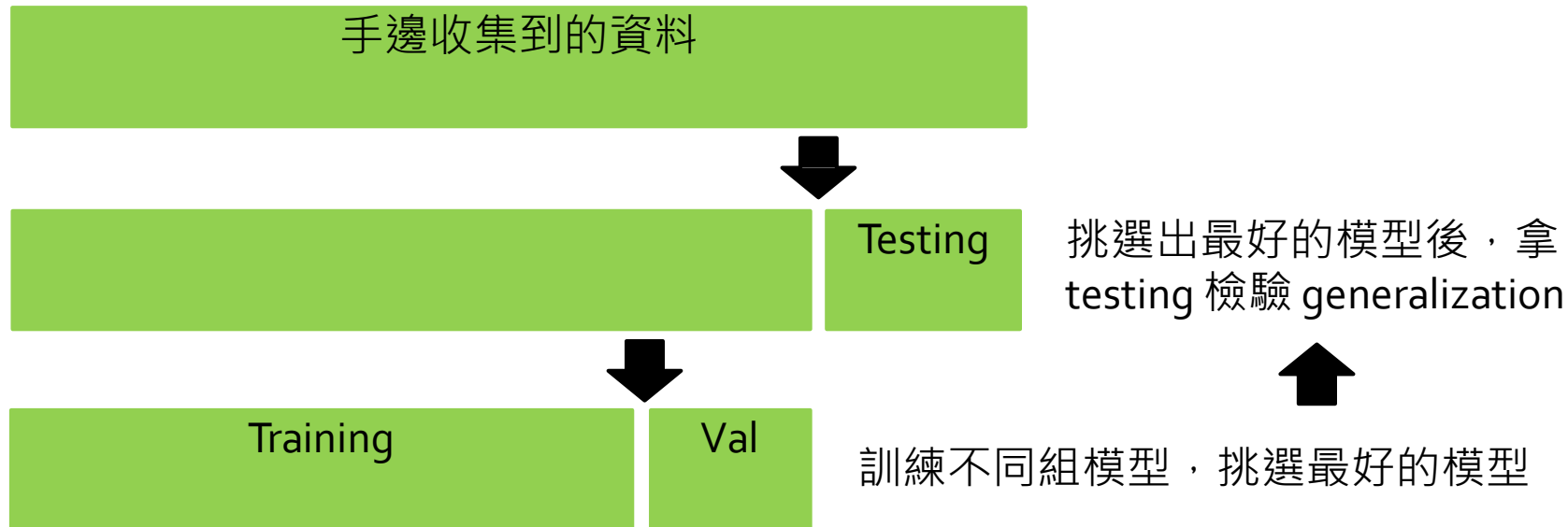
```
# 指定 loss function 和 optimizer  
model.compile(loss='categorical_crossentropy',  
              optimizer=sgd)
```

- Compilation before execution

# 資料怎麼分

- 收集的資料分三分

- Testing dataset: 真正跑分，一開始便分出來，當作檢測未來應用的基準
- Training dataset: 訓練時依據此dataset更新網路和其他參數使用
- Validation dataset: 每次epoch訓練完，評斷看看模型用，不能加入訓練



# Validation Dataset

- 兩種作法
- 利用 `model.fit` 的參數 `validation_split`
  - 從輸入(`X_train`, `Y_train`) 取固定比例的資料作為 validation
  - 不會先 shuffle 再取 validation dataset
  - 固定從資料尾端開始取
  - 每個 epoch 所使用的 validation dataset 都相同
- 手動加入 validation dataset
  - `validation_data=(X_valid, Y_valid)`

## 5. 開始訓練(Fit model)

```
# 指定 batch_size, nb_epoch, validation 後，開始訓練模型!!!  
history = model.fit( X_train,  
                    Y_train,  
                    batch_size=16,  
                    verbose=0,  
                    epochs=30,  
                    shuffle=True,  
                    validation_split=0.1)
```

- 給訓練資料與驗證資料後，開始訓練
- batch\_size: min-batch 的大小
- nb\_epoch: epoch 數量
  - 1 epoch 表示看過全部的 training dataset 一次
- shuffle: 每次 epoch 結束後是否要打亂 training dataset
- verbose: 是否要顯示目前的訓練進度，0 為不顯示