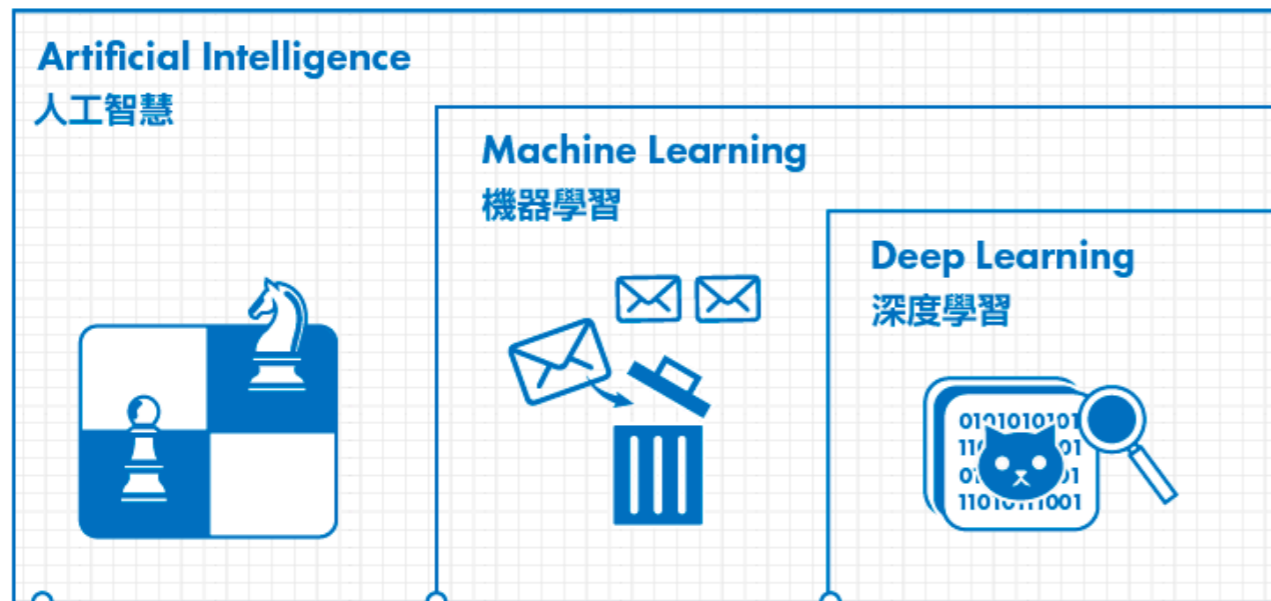

機器學習 簡介

人工智慧/機器學習/深度學習

人工智慧的歷史演進

人工智慧的一項分支是機器學習，機器學習的一項分支是深度學習 (類神經網路)



1950's

電腦出現時便有人工智慧的呼聲出現，讓電腦能夠自行從歷史資料中學會一套技能

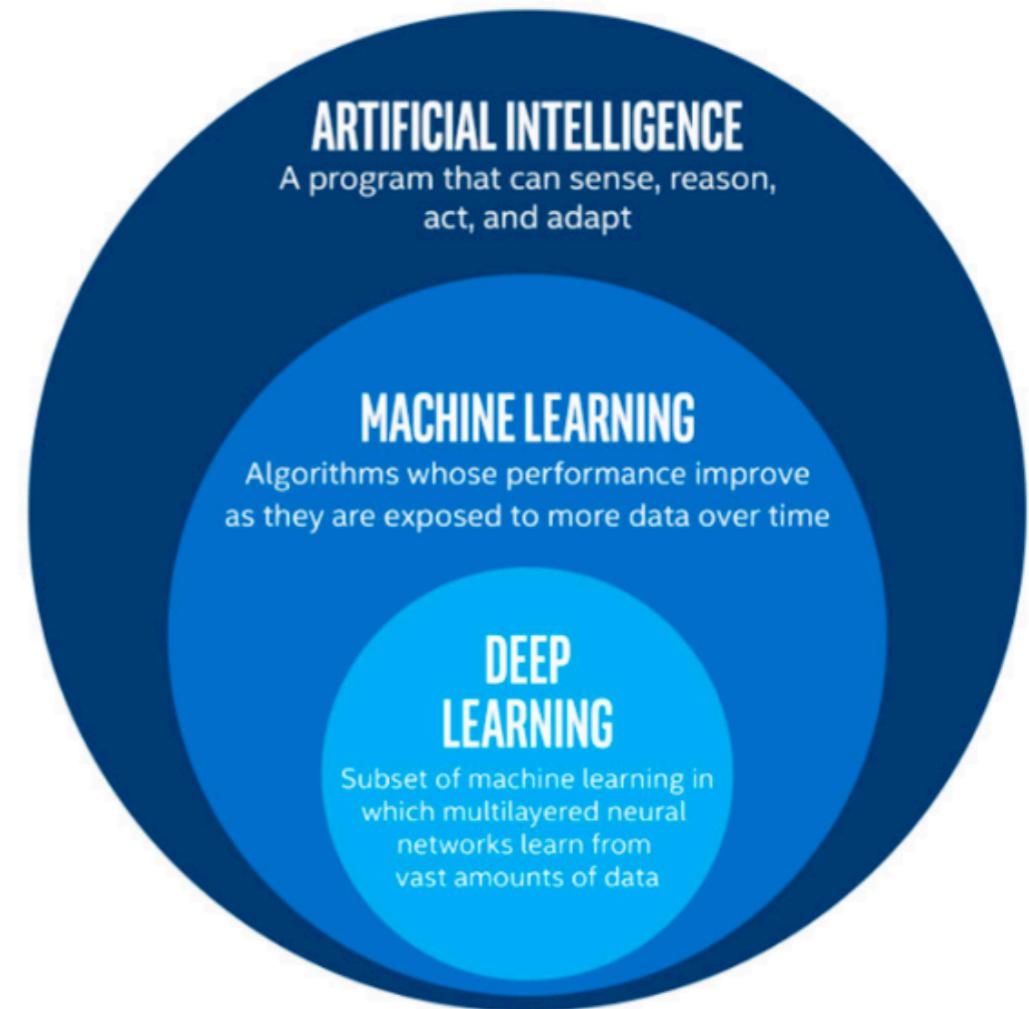
1980's

機器學習崛起，類神經網路紅極一時隨後又消沉以支持向量機 (SVM) 等淺層網路為主流

2010's

2006 年 Geoffrey Hinton 教授提出深度信念網路 (DBN) 打開了神經網路的一絲曙光，並改名為深度學習

2012 年 Hinton 的學生以 GPU+ 深度學習贏得 ImageNet 大賽，開啟深度學習熱潮



<https://www.stockfeel.com.tw/神經網路的復興：重回風口的深度學習/>

何謂人工智慧 (Artificial Intelligence)

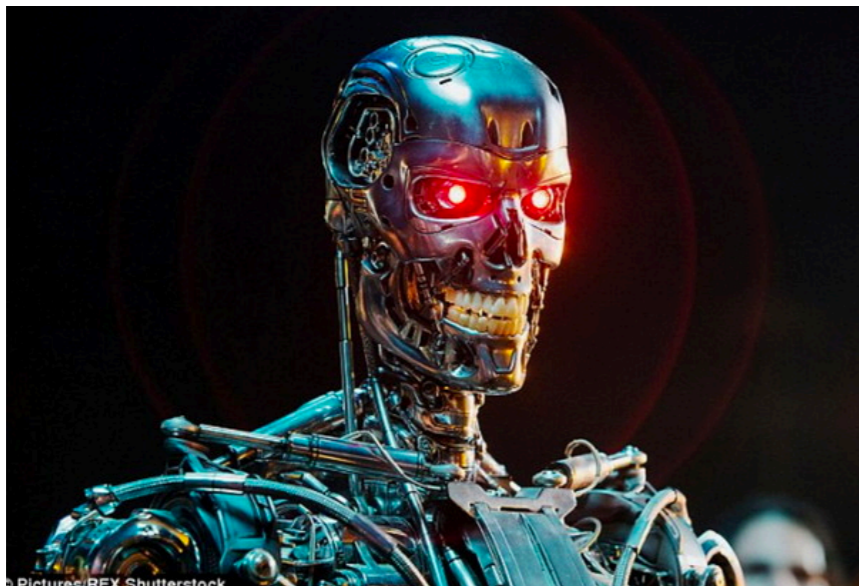
- 人工智慧 (Artificial intelligence, AI)：
約 1950 年開始發展，目的在於讓電腦具備類似人類的知識，應用領域有自然語言處理、電腦視覺、專家系統、機器人、機器學習、問題解決與規劃

「人工」和「智慧」

Strong AI

- Strong AI 可以被理解為是 " 有意識或知覺的AI " ，它既不只是有"人工智慧"而已，它本身也有更高級的類似我們人類意識和知覺的 " 自然智慧 " (Natural Intelligence).

魔鬼終結者的T-900機器人



超級難

如何證實??

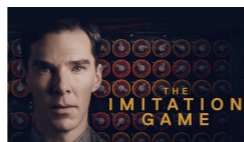
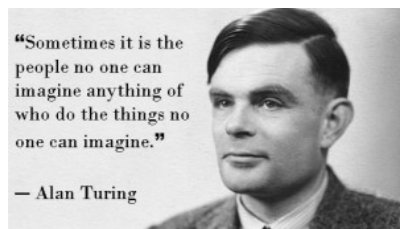
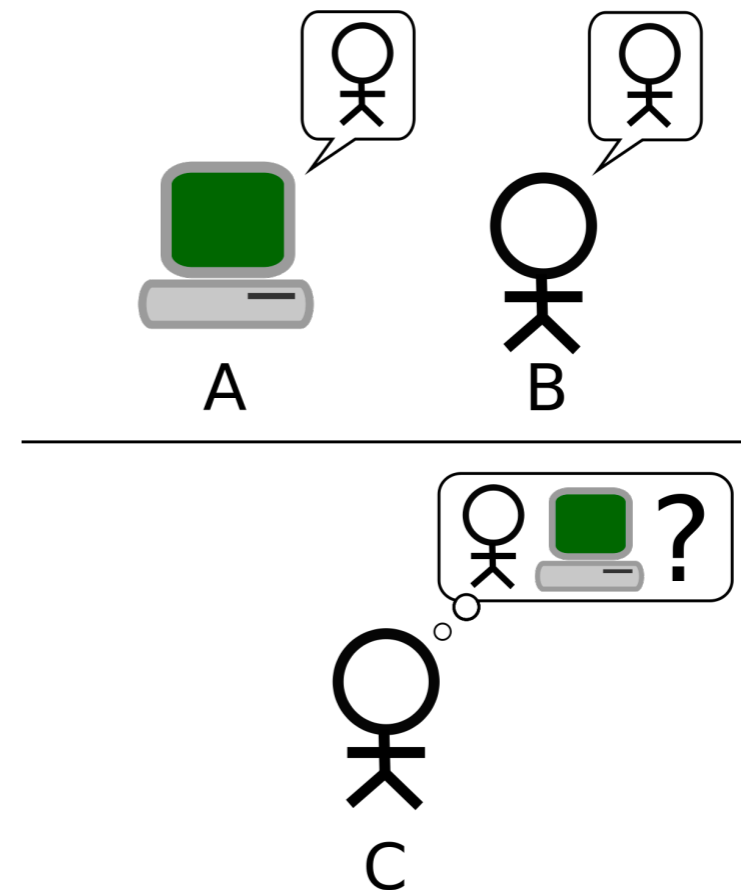
Picture: REX Shutterstock

圖靈測試(Turing test)

- 圖靈於1950年提出的一個關於**判斷機器是否能夠思考**的著名試驗：**模仿遊戲 (Imitation Game)** 來測試機器是否能表現出與人等價或無法區分的智能。

艾倫·麥席森·圖靈(Alan Mathison Turing):

- 現代電腦科學之父，天才數學家，在24歲時奠定了機算機的理论基礎。
- 二戰期間他為盟軍破解了德國的加密通訊機器「恩尼格瑪機Enigma」，協助戰爭時間縮短超過兩年。



Strong AI

超級難

從馬文·閔斯基到 AlphaGo，人工智慧走過了怎樣的 70 年？

<https://technews.tw/2016/02/11/ai-history/>

人工智慧在1970年代受到打擊之後，開始有新的研究方向出現。



STRONG

ARTIFICIAL INTELLIGENCE

WEAK

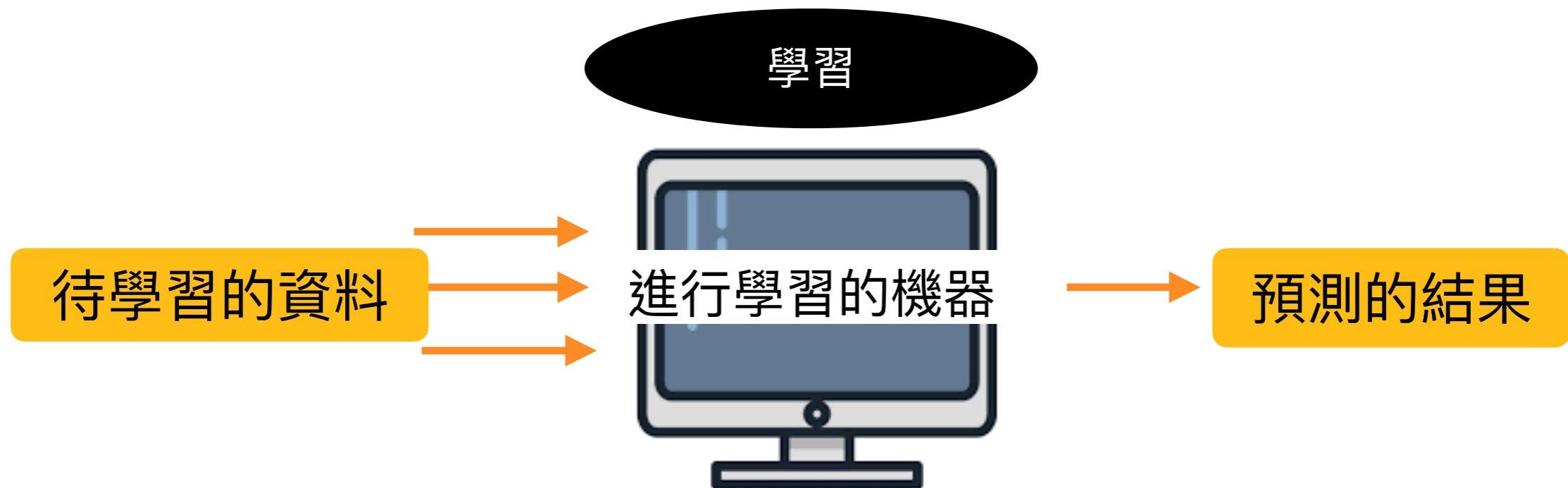
ARTIFICIAL INTELLIGENCE

Weak AI

<https://www.informit.com/articles/article.aspx?p=3100062&seqNum=4>

機器學習 (Machine Learning; ML)

- 約 1980 年開始發展，主要利用大量資料來訓練機器，使其能完成所指定工作
- 透過從過往的資料和經驗中學習並找到其運行規則，最後達到人工智慧的方法。」



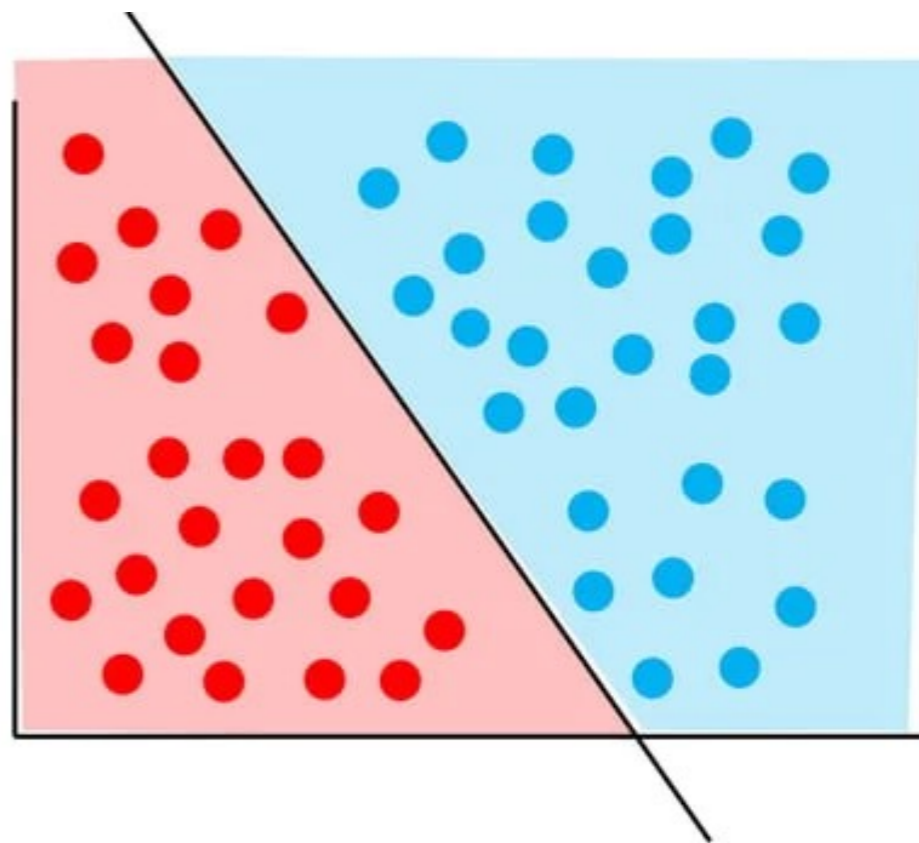
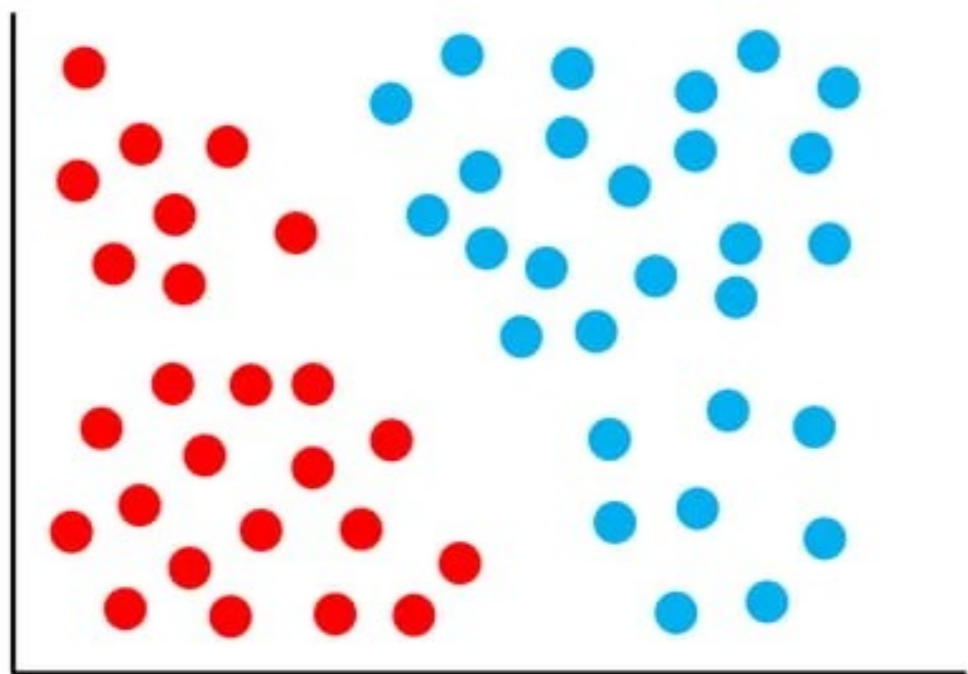
利用經驗改善系統自身的性能

[T. Mitchell, Book 97]

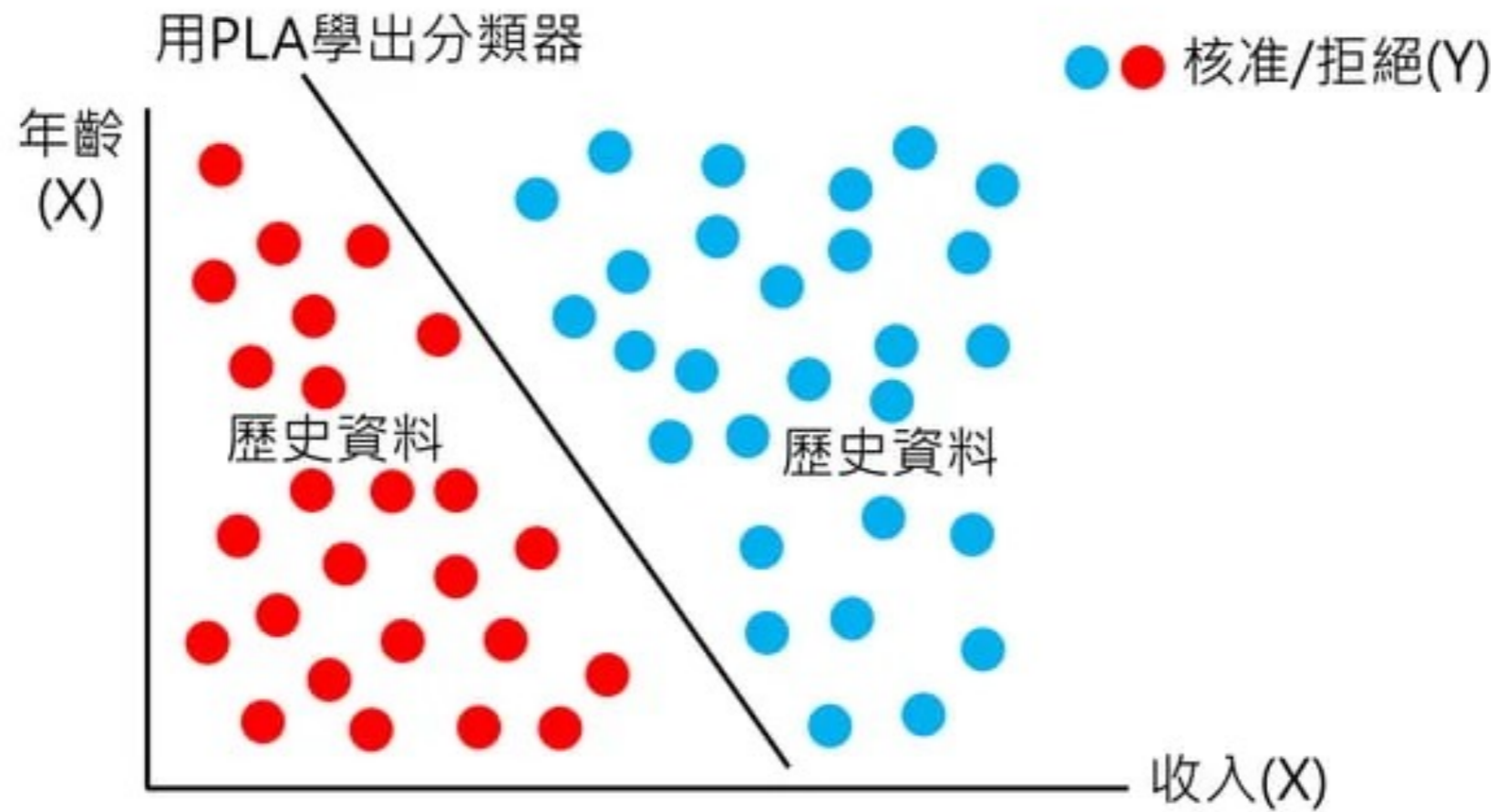
機器學習作什麼？

- 使用適當特徵(X)描述的資料，透過學習演算法，讓機器學習某種能力($X \rightarrow Y$)
- 未知的目標函式(f)定義了 X 和 Y 的實際關係。希望使用資料(D)並透過學習演算法(A)來學習出一個函式(g)，並希望 g 可以和 f 越接近越好

機器怎麼學習？



假設你家開銀行要發信用卡



人工智慧 (Artificial Intelligence)

機器學習 (Machine Learning)

深度學習 (Deep Learning)

Strong AI

Weak AI

監督式學習

非監督式學習

強化學習

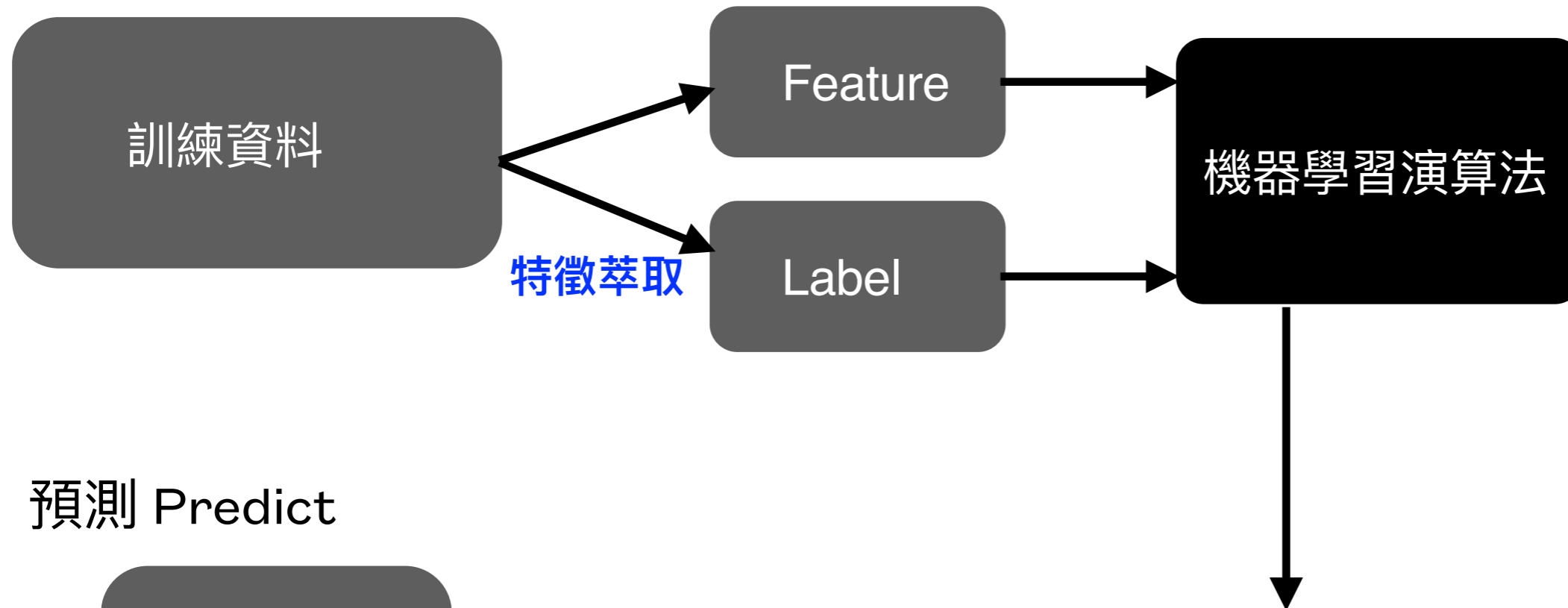
DNN

CNN

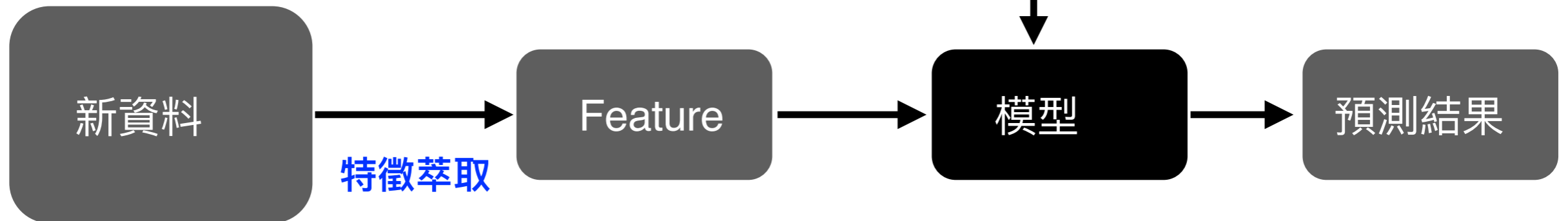
RNN

機器學習

訓練 Training



預測 Predict



機器學習分類

機器學習 (Machine Learning)

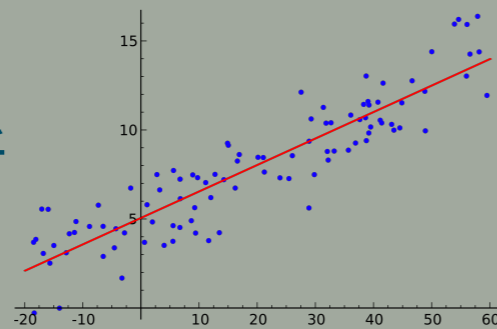
監督學習

分類 classification

二元分類

多元分類

迴歸分析

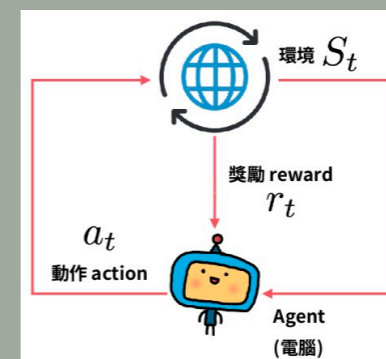


非監督學習

分群 clustering

強化學習

Q-learning



2023 值得注意的 10 大科技趨勢

- AI 日常應用更廣泛
- 元宇宙將局部成真
- Web3 新發展
- 接通數碼與物理世界
- 改變基因來編輯自然界
- 量子電腦開發進度
- 綠色科技進展
- 機械人將會更人性化
- 自動化系統發展
- 可持續科技成潮流



<https://www.cup.com.hk/2023/01/03/2023-tech-trends/>

無人駕駛

在搭自駕中巴前，什麼是自駕技術？

自駕車 a.k.a. 無人車，是能自動感測周圍環境並且無需人干預而自動導航的載具。

眼睛

感知周圍的路面環境
Radar、Lidar、Camera
、RTK GPS、V2X等感測器
，將不同感測資料傳回給系統。

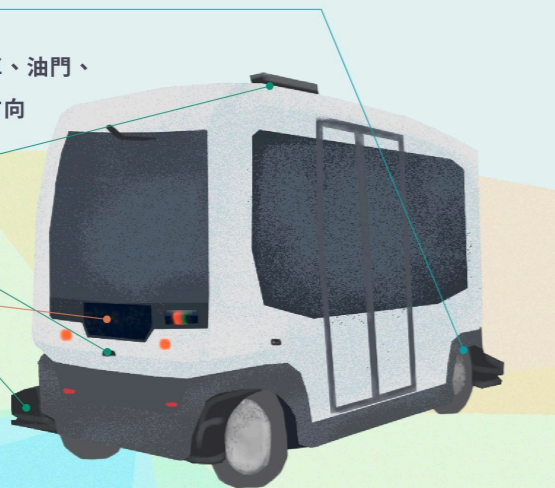
手腳

負責控制剎車、油門、
燈號及行進方向

大腦

通過複雜的計算機設備及配套
軟件，選擇駕駛路線

例如S3環周環境感知系統、V2X技術，
計算感測器傳回的資料，做出行進判斷。



自駕也有分等級！
不是有沒有人操控那麼簡單～

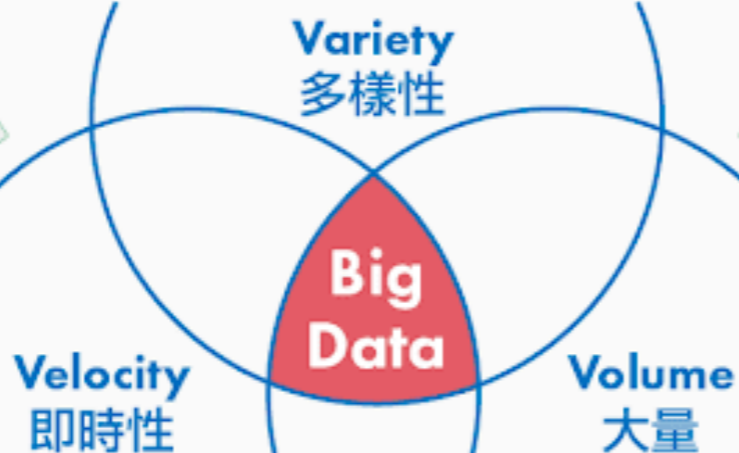
等級					
0	1	2	3	4	5
無自動	駕駛輔助	部分自動	條件自動	高度自動	完全自動
完全由駕駛人 操控	大部份需要駕駛 人操控	系統有多項駕駛輔助 功能，仍需由駕駛人 主控車輛	系統具有辨識環境 的能力，駕駛人可 適當地進行干涉。 進入「真自動駕駛」門檻	在條件許可下可讓 系統完全自駕，駕 駛人不一定介入 操控	任何情況下皆可由 自駕系統完全操控 ，可被駕駛人接管
換句話說就是...					
保持清醒好好開 車，系統只會警 示你倒車要A到 了沒	還是要自己開， 不過系統偶爾會 幫你，例如控制 一下車距	車子可以幫你控制 轉向、加減速等等 ，讓你開得輕鬆一 些	特定情境下就放 手讓車自己開吧	只有少數特殊情 境才需自己來	啥都不用做，坐著 欣賞窗外風景吧
駕駛人監控駕駛環境			自駕系統監控駕駛環境		
手放在方向盤、腳放在煞車上			可以不用握方向盤沒關係		

大數據分析

從儲存、挖掘到溝通，引領產業新面貌

大數據

STOCKFEEL



數據分析專題

BIG DATA



聊天機器人



蘋果 Siri



Google 助理



微軟 ChatGPT

Quick, Draw!



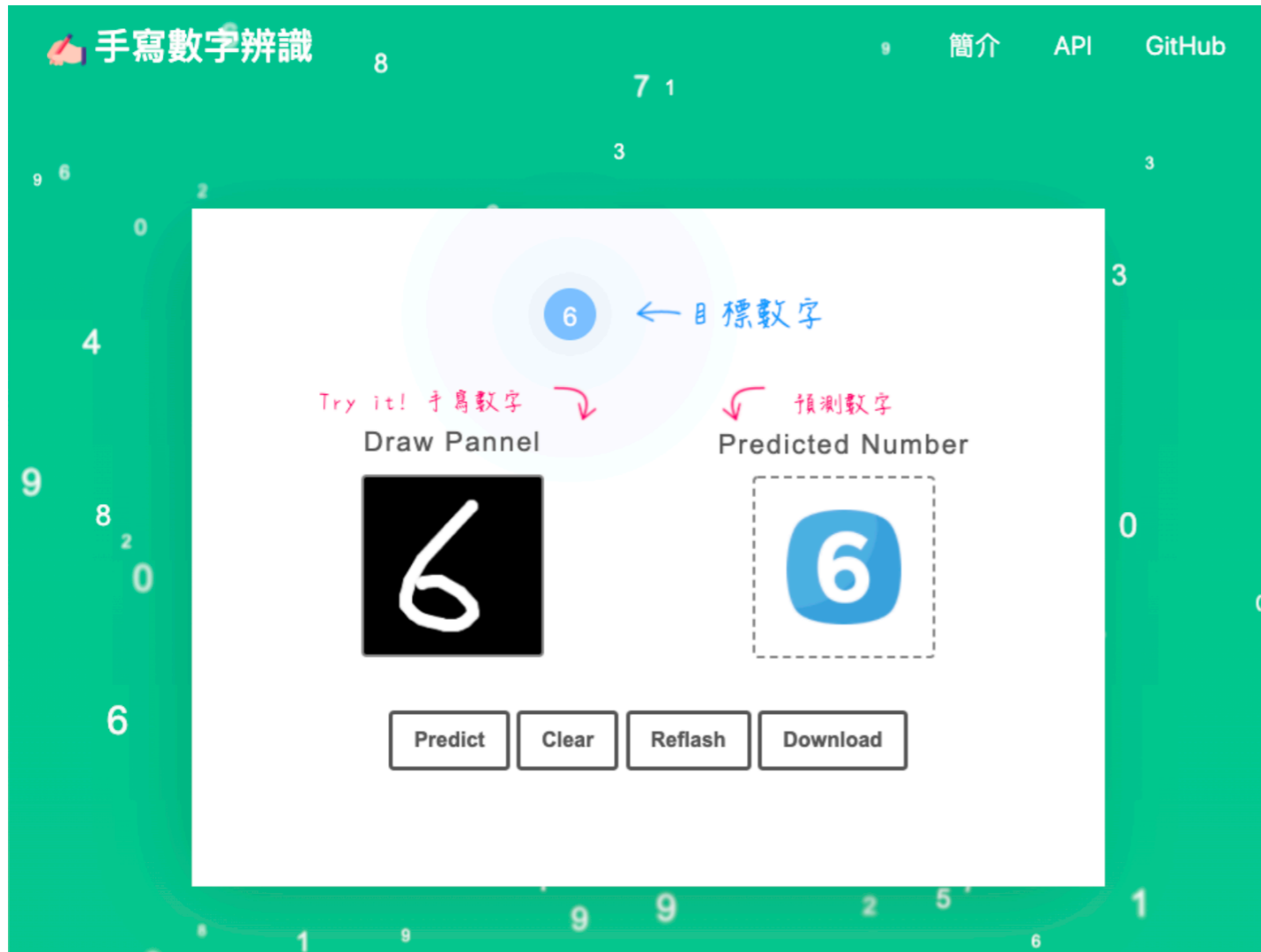
類神經網路能學會辨識塗鴉嗎？

只要將你的繪圖加到[全世界最大的塗鴉資料集](#)，就能協助訓練這個類神經網路。這個資料集的內容會公開分享，為機器學習研究提供參考資料。

開始塗鴉！

https://quickdraw.withgoogle.com/?locale=zh_TW

MNIST 手寫數字辨識



<https://andy6804tw.github.io/digit-recognizer-project/>

MINIST 簡介

MNIST手寫數字辨識資料集

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU

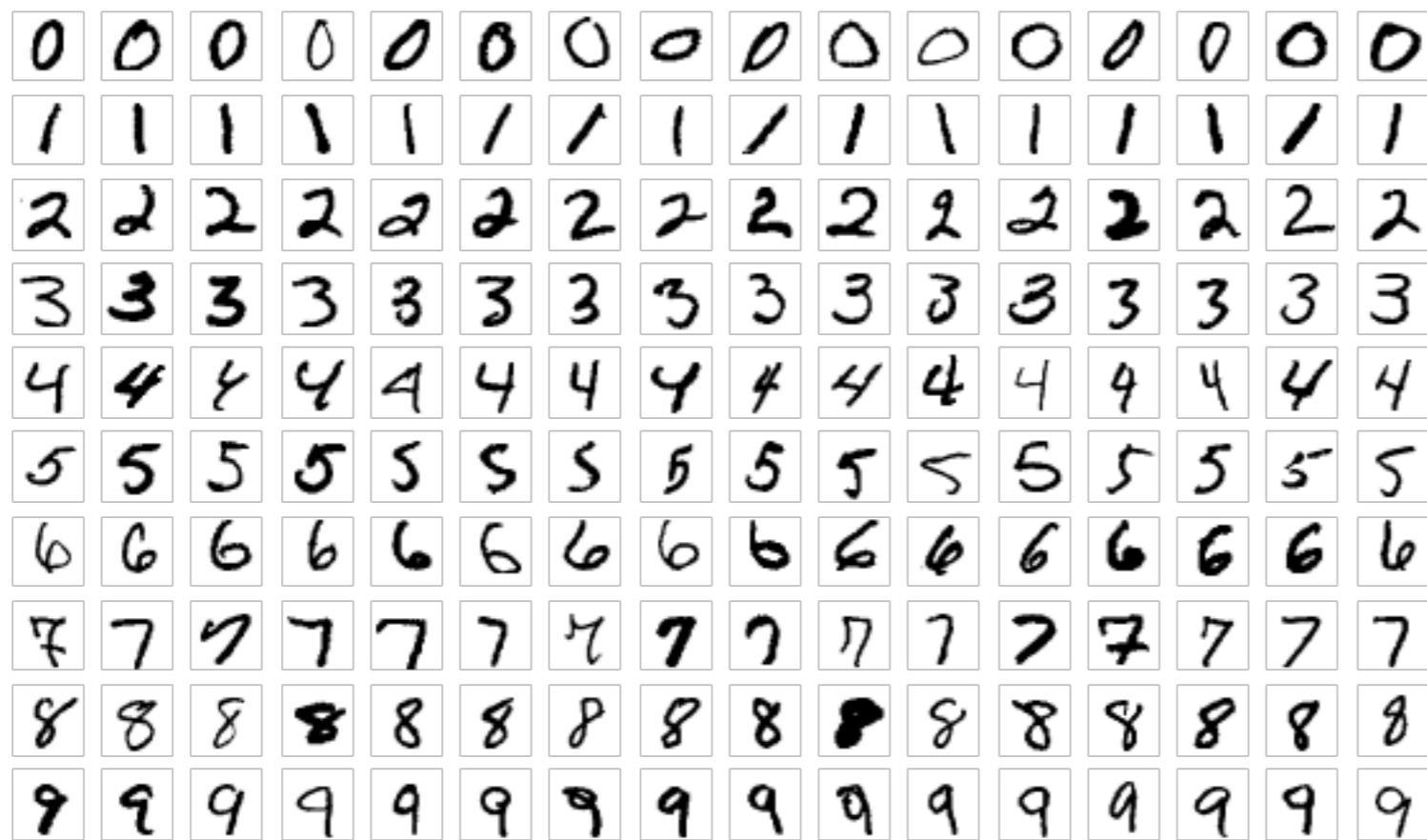
Corinna Cortes, Google Labs, New York

Christopher J.C. Burges, Microsoft Research, Redmond

- 資料量不會太多，單色較簡單
- 適於練習建立模型，訓練，預測



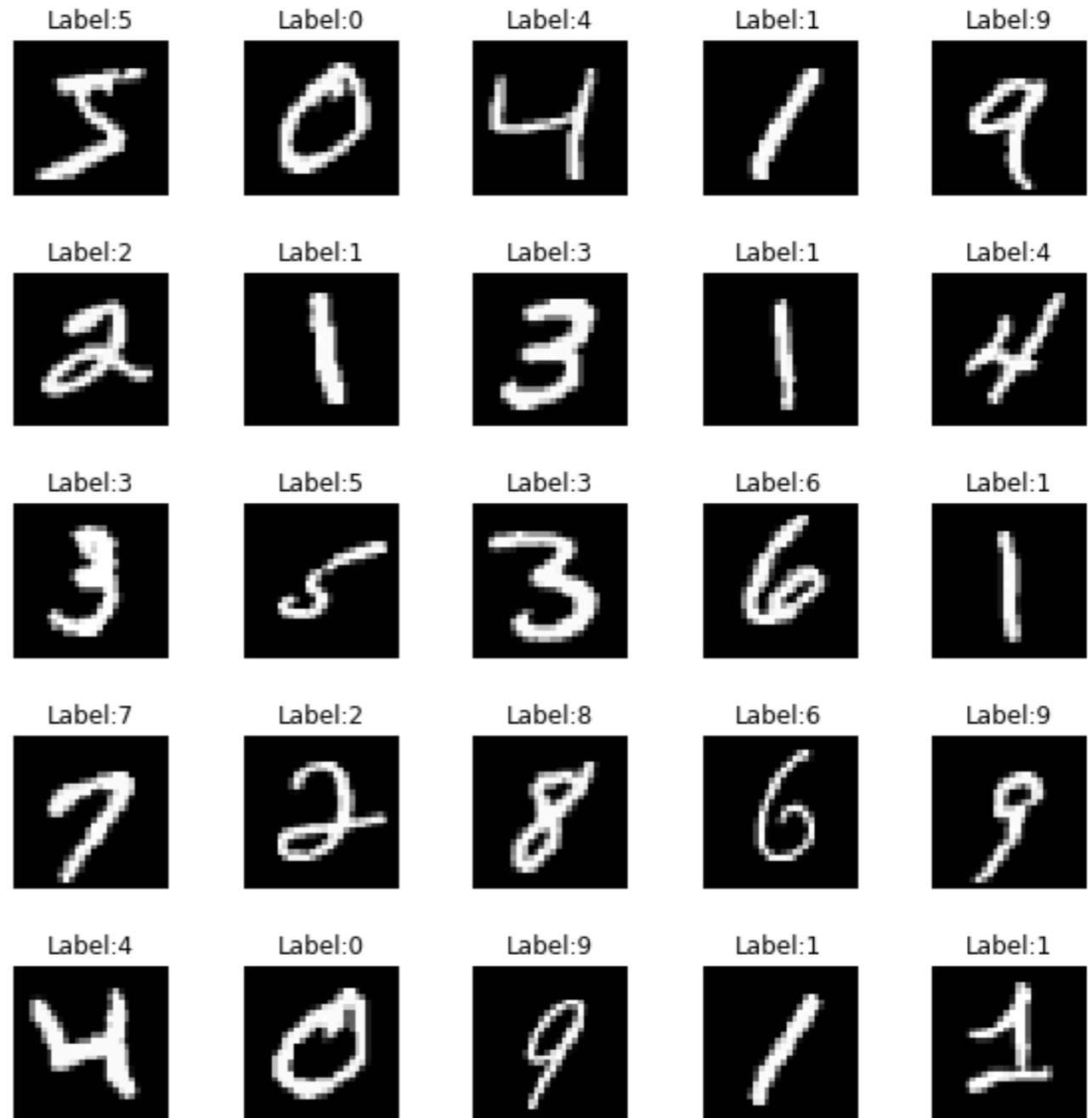
-
- MNIST資料集是由60,000筆訓練資料、10,000筆測試資料所組成。
 - MNIST資料集裡的每一筆資料皆由images(數字的影像)與labels(該圖片的真實數字，其實就是答案)所組成



https://en.wikipedia.org/wiki/MNIST_database

Keras MNIST 手寫數字資料集

- Training data : (60000,28,28)
- Training label: (60000,)
- Test data : (10000, 28,28)
- Test label: (10000,)



資料預處理

- 建立模型必須先將內容進行資料預處理
- 資料預處理分為兩個部分
 - ① Features(數字影像特徵值)資料預處理
28*28數字影像 → 784 , type:float
 - ② Labels(數字影像真實值)資料預處理
數字影像Image的數字影像標準化

```
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,
 18, 18, 18, 126, 136, 175, 26, 166, 255, 247, 127, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 30, 36, 94, 154, 170,
 253, 253, 253, 253, 253, 225, 172, 253, 242, 195, 64, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 49, 238, 253, 253, 253, 253,
 253, 253, 253, 253, 251, 93, 82, 82, 56, 39, 0, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 18, 219, 253, 253, 253, 253,
 253, 198, 182, 247, 241, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 80, 156, 107, 253, 253,
 205, 11, 0, 43, 154, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 1, 154, 253,
 90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0],
```

```
0., 0., 0., 0., 0., 0., 0., 0., 3.,
 18., 18., 18., 126., 136., 175., 26., 166., 255.,
 247., 127., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 30., 36., 94., 154.,
 170., 253., 253., 253., 253., 253., 225., 172., 253.,
 242., 195., 64., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 49., 238., 253., 253.,
 253., 253., 253., 253., 253., 253., 251., 93., 82.,
 82., 56., 39., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 18., 219., 253.,
 253., 253., 253., 253., 198., 182., 247., 241., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 80.,
 156., 107., 253., 253., 205., 11., 0., 43., 154.,
 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 14., 1., 154., 253., 90., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.,
0., 0., 0.01176471, 0.07058824, 0.07058824,
0.07058824, 0.49411765, 0.53333336, 0.68627453, 0.10196079,
0.65098041, 1., 0.96862745, 0.49803922, 0.,
0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.,
0., 0.11764706, 0.14117648, 0.36862746, 0.60392159,
0.66666669, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.88235295, 0.67450982, 0.99215686, 0.94901961,
0.7647059, 0.25098041, 0., 0., 0.,
0., 0., 0., 0., 0.,
0., 0., 0., 0.19215687, 0.93333334,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.98431373, 0.36470589,
0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.,
0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.,
```

- labels資料預處理

label原本是0~9的數字

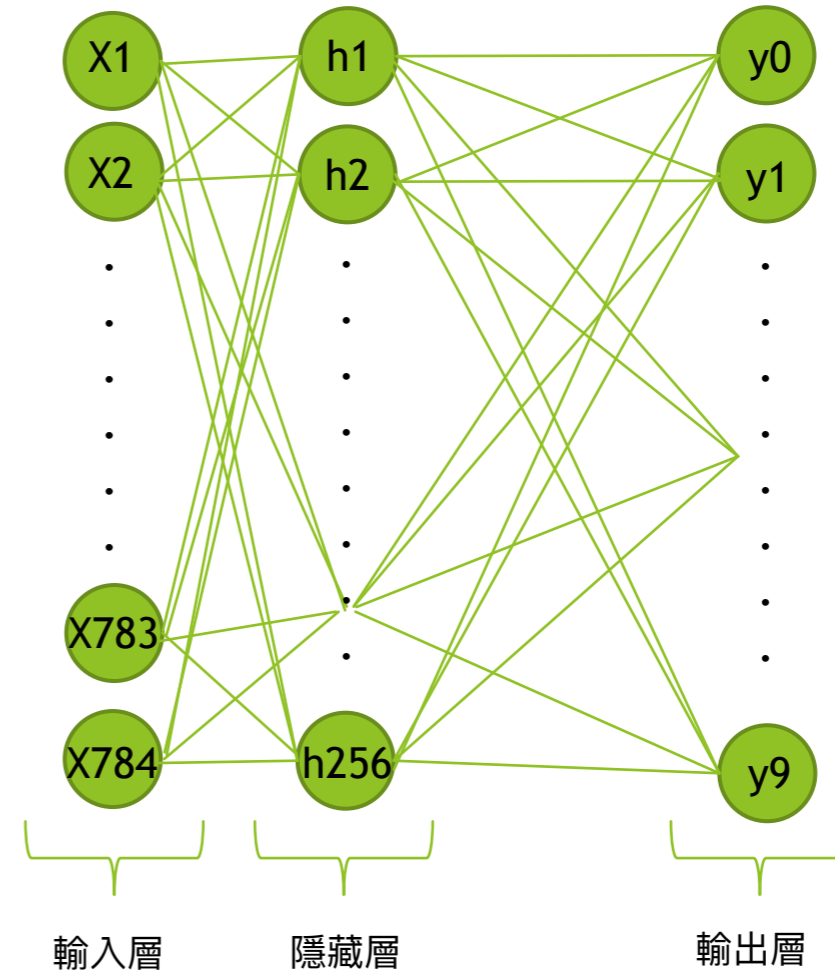
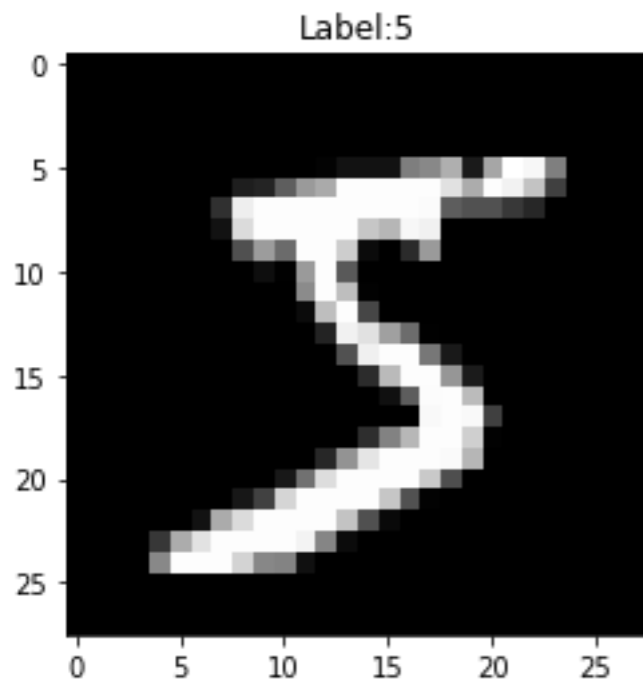


10個0或1的組合

數字3經過 one-hot encoding → 0001000000

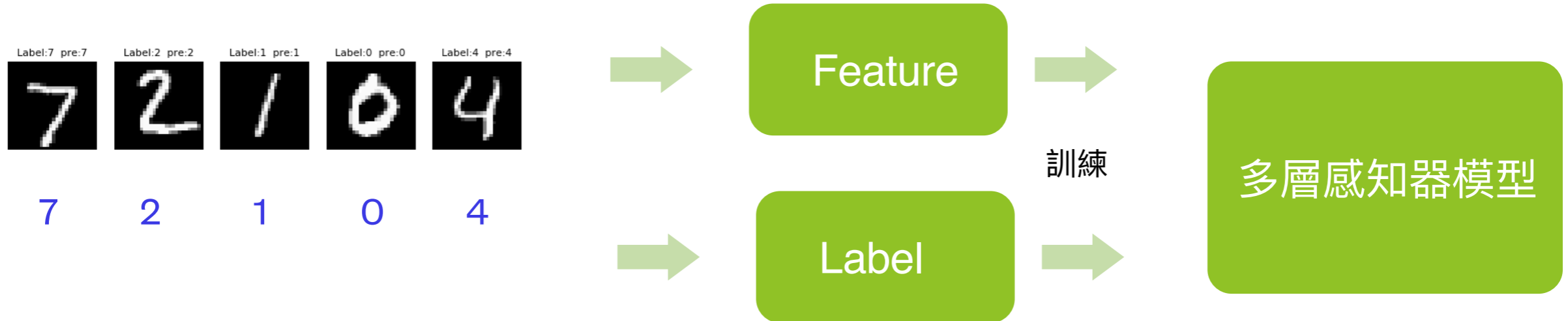
數字7經過 one-hot encoding → 0000000100

多層感知器模型(MLP)介紹



多層感知器(MLP)訓練與預測

- 訓練 Training



- 預測 Predict



MLP 建立步驟

1. 資料預處理

2. 建立模型

3. 訓練模型

4. 評估模型準確率

5. 進行預測

建立模型

```
# 建立 Sequential 順序模組
model = Sequential()

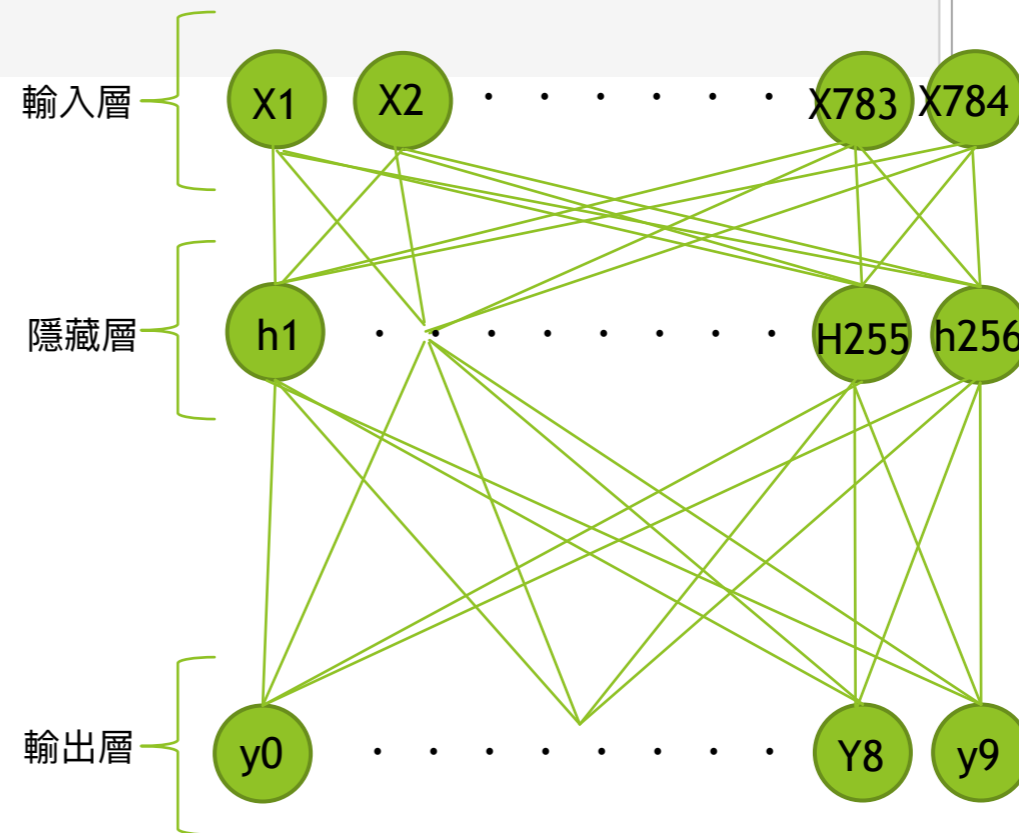
# 模型加入【輸入層】與【隱藏層】
model.add( Dense( input_dim = 28*28
                 , units = 256
                 , kernel_initializer = 'normal'
                 , activation = 'relu'
                 ))
# 輸入層有 28*28=784 個神經元
# 隱藏層有 256 個神經元 (值越大, 訓練越精準, 相對訓練時間也越久)
# 使用 normal 初始化 weight 權重與 bias 偏差值
# 使用 relu 激活函數

# 模型加入【輸出層】
model.add( Dense( units = 10
                 , kernel_initializer = 'normal'
                 , activation = 'softmax'
                 ))
# 輸出層有 10 個神經元 (因為數字只有 0 ~ 9)
# 使用 normal 初始化 weight 權重與 bias 偏差值
# 使用 softmax 激活函數 (softmax 值越高, 代表機率越大)

# 顯示模型摘要資訊
print (model.summary() )
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	200960
dense_1 (Dense)	(None, 10)	2570
Total params: 203,530		
Trainable params: 203,530		
Non-trainable params: 0		
None		



設定模型的訓練方式

```
## 設定模型的訓練方式 ##

model.compile( loss='categorical_crossentropy' # 設定 Loss 損失函數 為 categorical_crossentropy 使用交叉熵
              , optimizer = 'adam'           # 設定 Optimizer 最佳化方法 為 adam
              , metrics = ['accuracy']       # 設定 Model 評估準確率方法 為 accuracy
              )

# 訓練模型
history = model.fit(
    x = X_train_normalize # 訓練的歷史記錄, 會回傳到指定變數 history
    , y = Y_train_OneHot  # 設定 圖片 Features 特徵值 (mnist 提供 60000 筆資料)
    , validation_split = 0.2 # 設定 圖片 Label 真實值 (mnist 提供 60000 筆資料)
    , epochs = 10           # 設定 有多少筆驗證 (60000*0.2=12000 筆驗證, 60000*0.8=48000 筆訓練)
    , batch_size = 128     # 設定 訓練次數 (值 10 以上, 值越大, 訓練時間越久, 但訓練越精準)
    , verbose = 2          # 設定 訓練時每批次有多少筆 (值 100 以上, 值越大, 訓練速度越快, 但需記憶體要夠大)
                          # 是否 顯示訓練過程 (0: 不顯示, 1: 詳細顯示, 2: 簡易顯示)
)
```

- 設定訓練與驗證比例
- 設定epochs次數與每一批次筆數

訓練資料

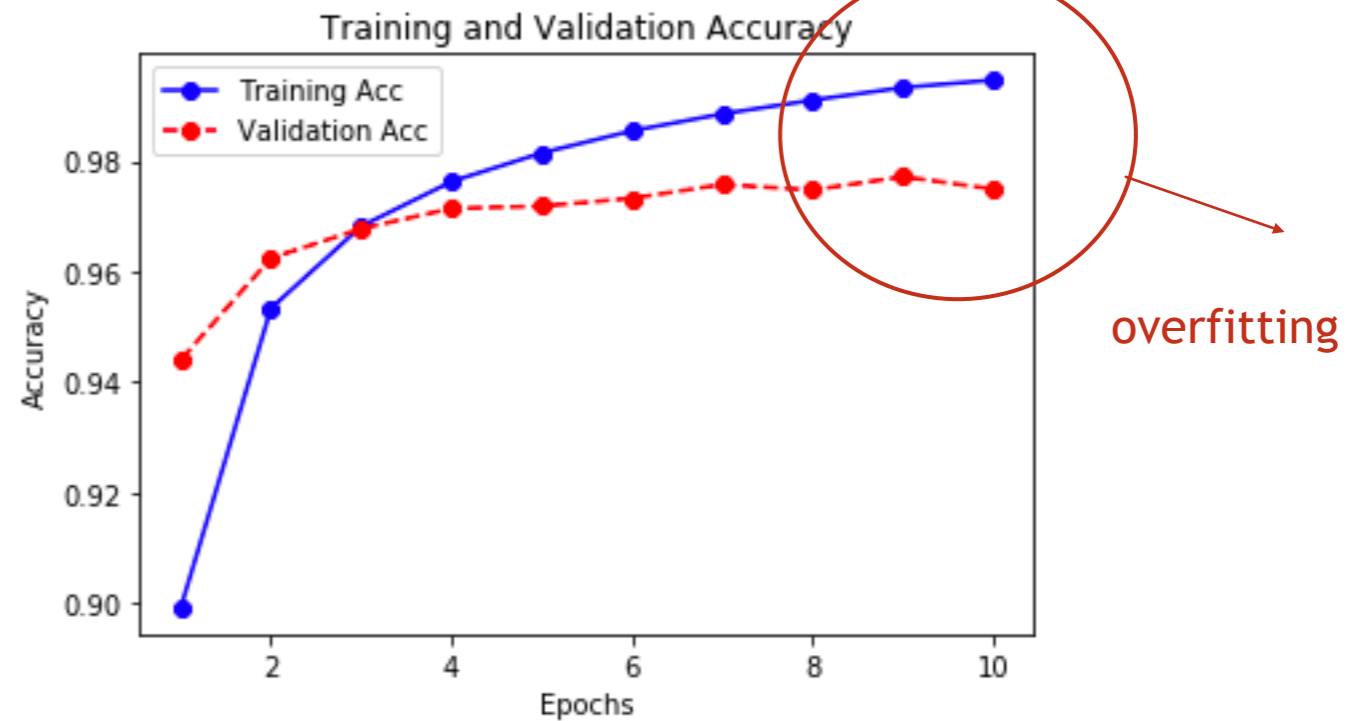
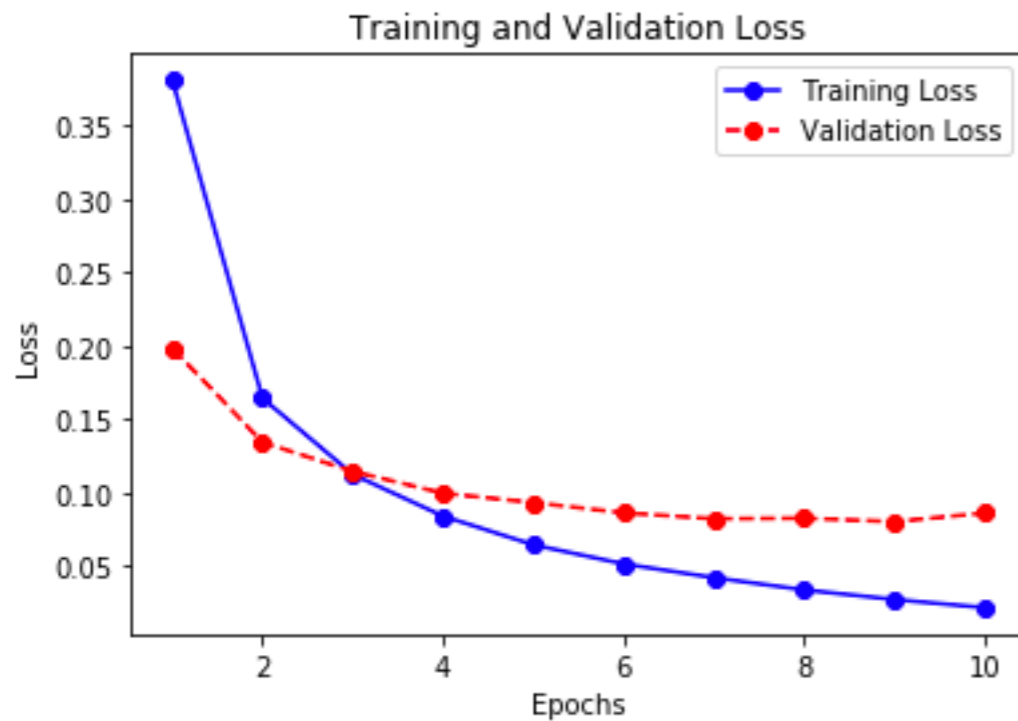
```
# 執行的顯示結果 (這會花一些時間, 然後會逐次顯示訓練結果)
# loss:      使用訓練資料, 得到的損失函數誤差值 (值越小, 代表準確率越高)
# acc:      使用訓練資料, 得到的評估準確率 (值在 0~1, 值越大, 代表準確率越高)
# val_loss: 使用驗證資料, 得到的損失函數誤差值 (值越小, 代表準確率越高)
# val_acc:  使用驗證資料, 得到的評估準確率 (值在 0~1, 值越大, 代表準確率越高)

# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(X_train_normalize , Y_train_OneHot)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))
loss, accuracy = model.evaluate(X_test_normalize , Y_test_OneHot)
print("測試資料集的準確度 = {:.2f}".format(accuracy))

model.save('mnist_mlp.h5')
```

結果

- 多層感知器
- 輸入層:784個神經元,隱藏層:256個神經元,輸出層:10個神經元
- Epoch=10



混淆矩陣 Confusion matrix

- 混淆矩陣(Confusion matrix)是一種對分類模型進行效果評估的方法
- 通過將模型預測的數據與測試數據進行對比，使用準確率，覆蓋率和命中率等指標對模型的分類效果進行度量。

```
import pandas as pd

# 計算分類的預測值
print("\nPredicting ...")
Y_pred = model.predict_classes(X_test_normalize)

# 顯示混淆矩陣
tb = pd.crosstab(Y_test_bk.astype(int), Y_pred.astype(int),
                 rownames=["label"], colnames=["predict"])
print(tb)
```

```
Predicting ...
predict    0     1     2     3     4     5     6     7     8     9
label
0          966     0     1     1     1     3     4     2     1     1
1           0  1124     4     1     0     1     2     0     3     0
2           4     1  1008     1     2     0     2     7     7     0
3           1     1     4   988     0     5     0     5     3     3
4           1     1     2     1   958     0     3     3     0    13
5           2     0     0     7     1   866     6     2     5     3
6           6     2     0     1     4     4   941     0     0     0
7           1     4    10     3     0     0     0  1002     0     8
8           2     0     2     5     6     2     3     3   948     3
9           3     2     0     7     5     1     0     5     0   986
```

Overfitting問題

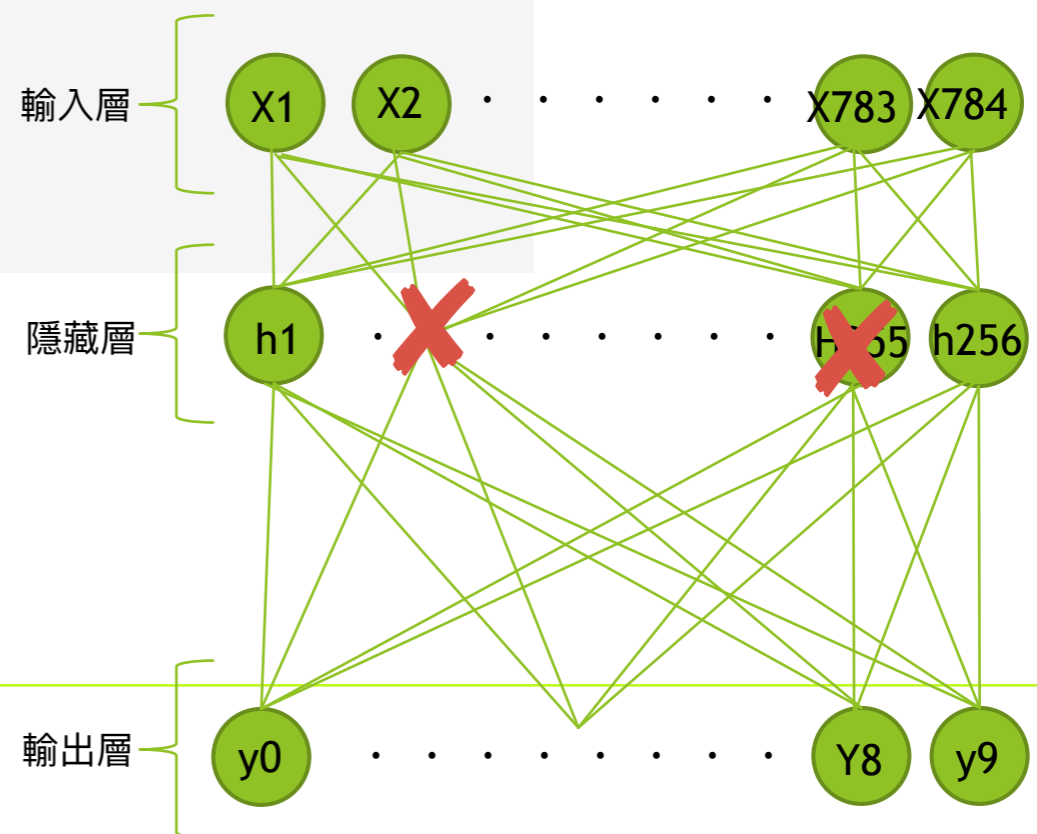
- Overfitting(過度訓練):
當可選擇的參數自由度超過資料所包含的資訊內容時，
這會破壞模型一般化的能力。
 - 解決方法:
 - ① 增加數據量, 大部分過擬合產生的原因是因為數據量太少了。
 - ② 加入DropOut功能，在訓練的時候，我們隨機忽略掉一些神經元和神經元的連結，讓每一次預測結果都不會太過依賴於其中某部分特定神經元。
-

多層感知器加入DropOut功能

- 增加DropOut功能:為了解決Overfitting的問題
- DropOut的功能:每次訓練迭代時,隨機地在隱藏層中放棄神經元,以避免overfitting
- DropOut(0.5)→放棄隱藏層中50%的神經元
DropOut(0.25)→放棄隱藏層中25%的神經元

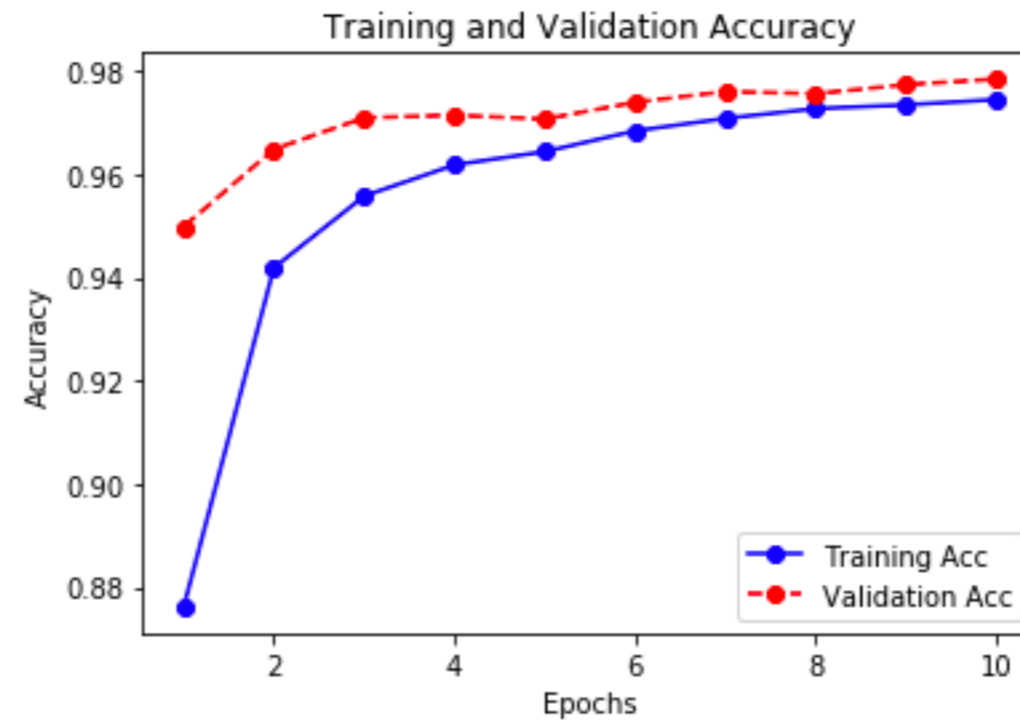
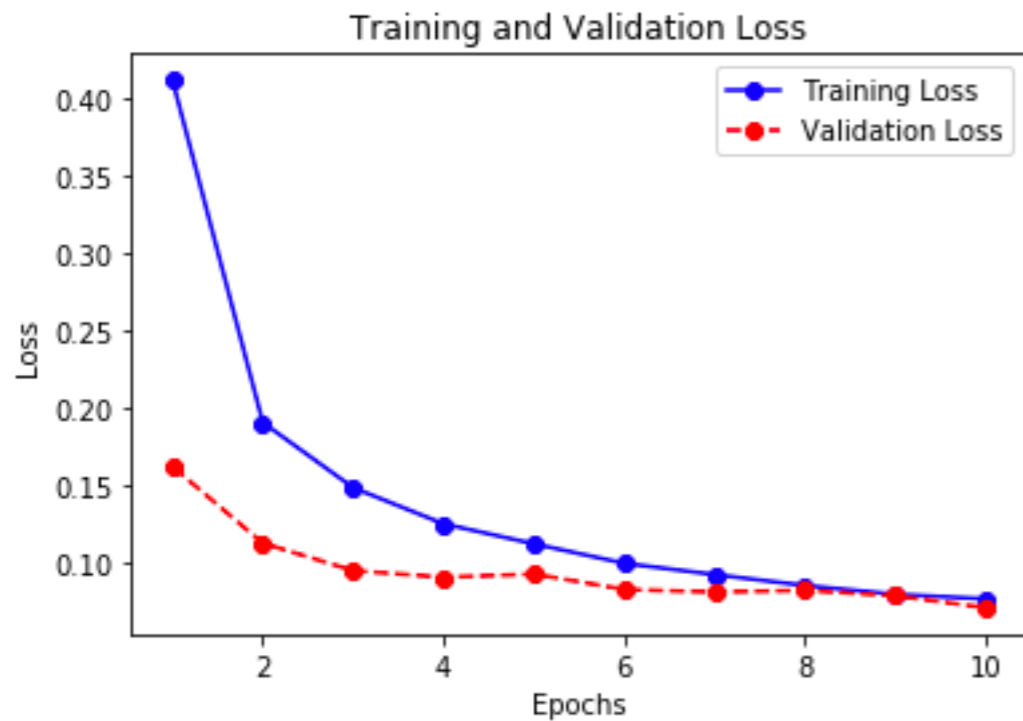
定義模型

```
model = Sequential()  
model.add(Dense(256, input_dim=28*28, activation="relu"))  
model.add(Dropout(0.5))  
model.add(Dense(256, activation="relu"))  
model.add(Dense(10, activation="softmax"))  
model.summary() # 顯示模型摘要資訊
```



結果

- 多層感知器
- 輸入層:784個神經元,隱藏層:800個神經元,輸出層:10個神經元
- Epoch=10
- 加入DropOut(0.5)



應用：使用自己的模型 進行預測

下載 小畫家 app

