桃園區域網路中心

初探 Python AI 深度學習的第一堂課

林澤佑

焉　然

程式碼

http://bit.ly/yenlung

今日使用到的程式放在 **Deep-Learning-Basics**

講師：林澤佑

- 政治大學應用數學博士候選人
- 台灣人工智慧學校講師
- 東吳大學財務工程與精算學系兼任講師
- 錄製 Deep Learning 政大磨課師課程

助教：焉　然

- 政治大學應用數學系畢
- 政治大學應用數學系碩士班
- 政治大學「數學軟體應用」、「設計思考與人工智慧」助教

01.

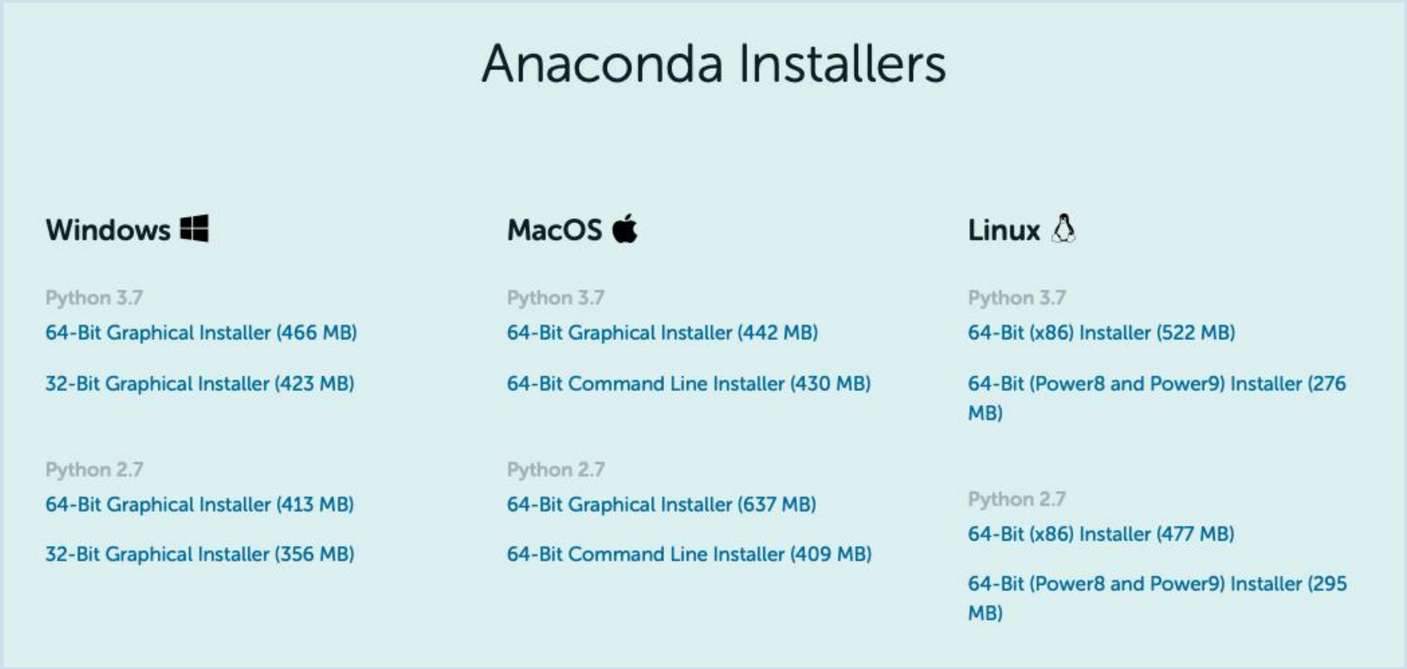
安裝 Anaconda

找到 Anaconda Download 點

也可以 Google 搜尋 Anaconda

https://www.anaconda.com/products/individual

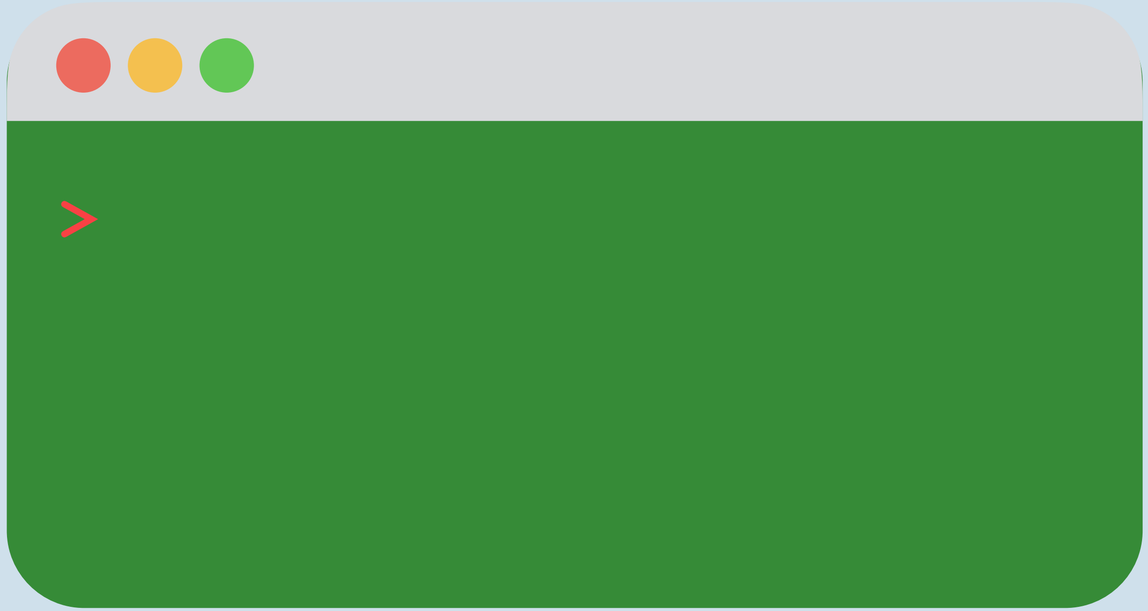
選擇適合你的 OS



執行安裝程式, 請就按下一步、下一步, 完全依預設安裝!

找到你適合的終端機

我們來試試剛開始可能有點可怕的終端機。



Windows 使用者請用
Anaconda Prompt

建議使用虛擬環境

安裝 TensorFlow

```
> conda install tensorflow
```

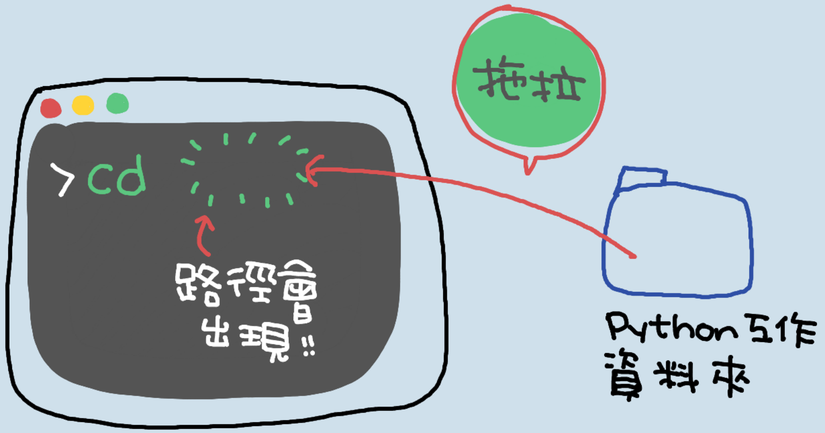
安裝 TensorFlow GPU 版

```
> conda install tensorflow-gpu
```

要先更新 NVIDIA GPU 顯卡 Driver

cd 進入你的工作資料夾

拖拉神技出現工作資料夾

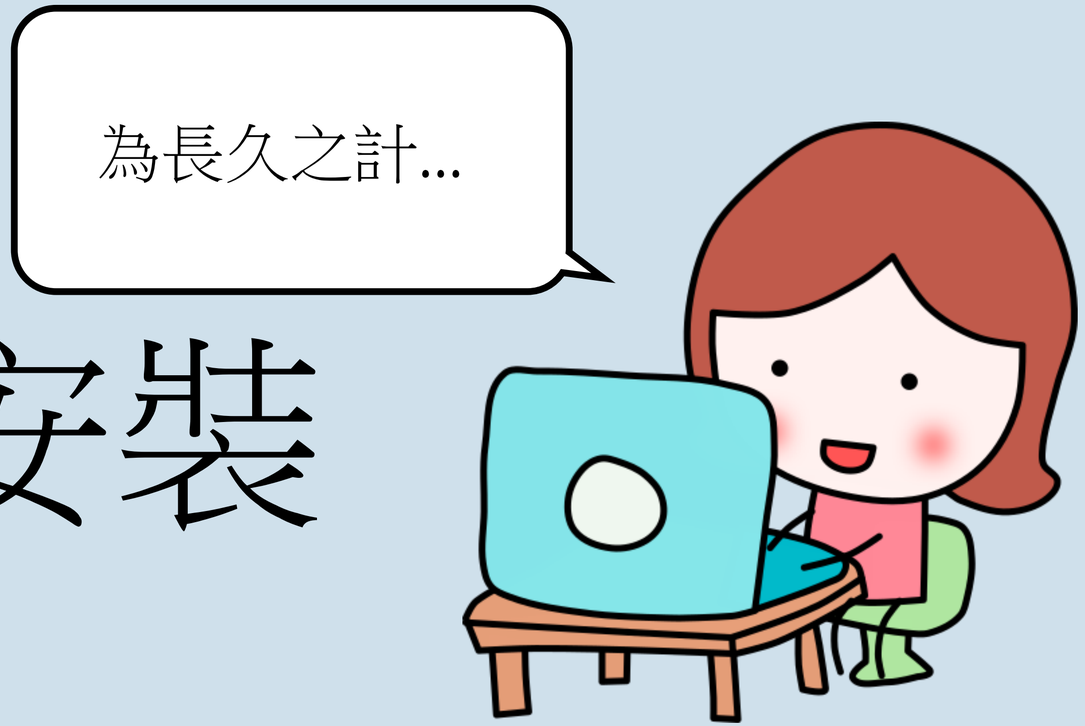


用拖拉的方式，可以找到完整路徑！

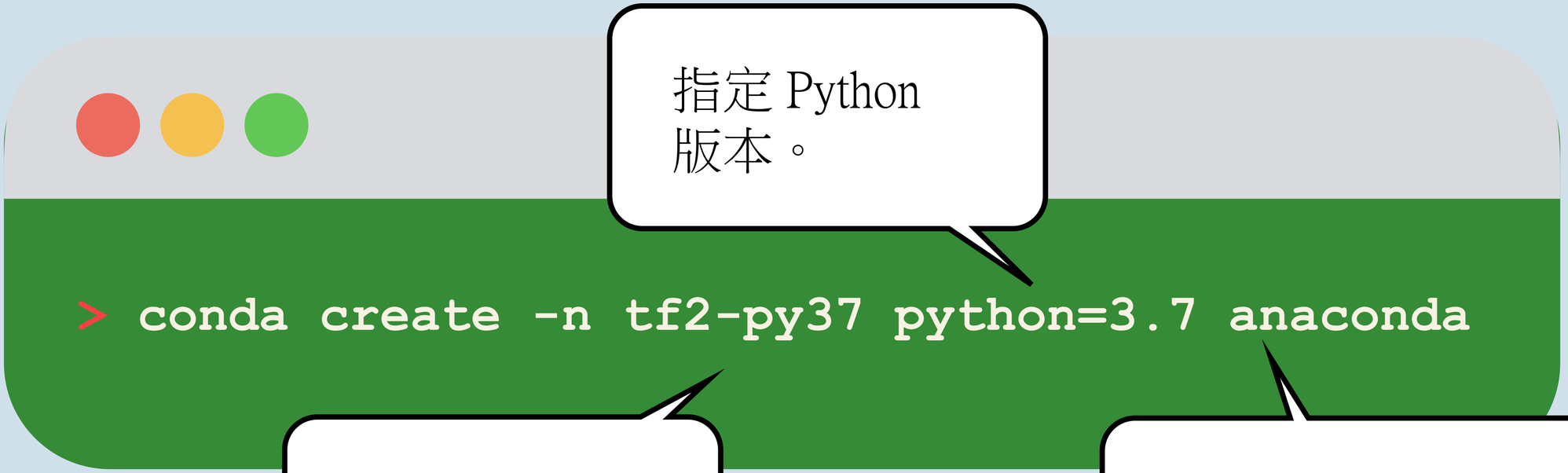
執行 Jupyter Notebook

```
> jupyter notebook
```

虛擬環境安裝



建一個乾淨的 Anaconda 虛擬環境



讓 Jupyter Notebook 找到所有虛擬環境的 Python

```
> conda install nb_conda
```

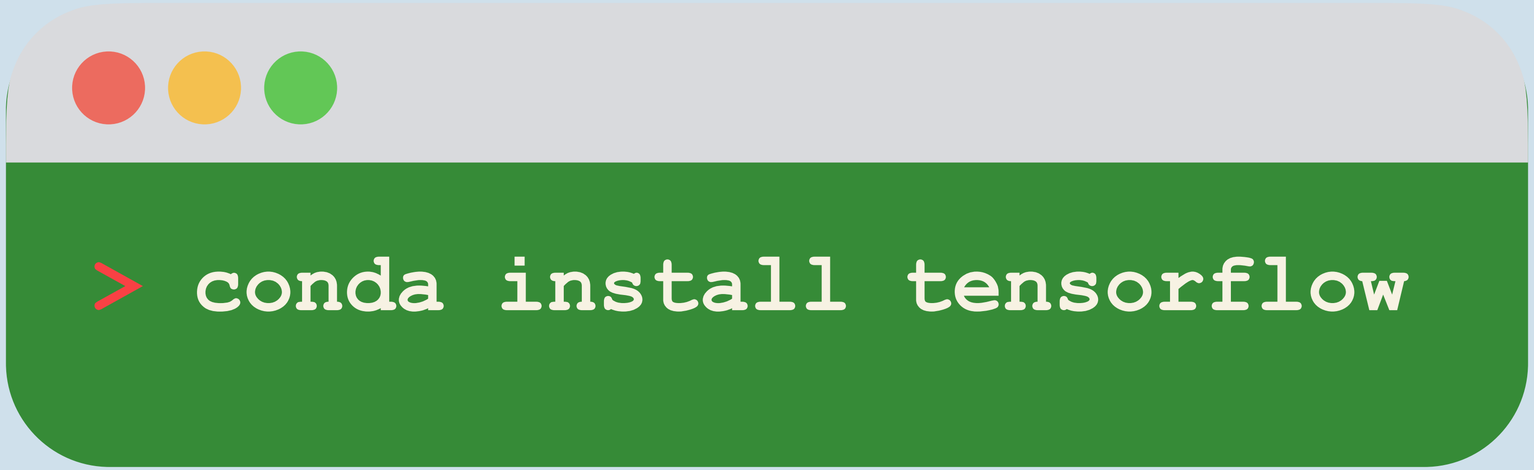
* `conda install` 是未來我們在 Anaconda 安裝套件的指令。

進入一個虛擬環境

```
> conda activate tf2-py37
```

接著和之前一樣了...

安裝 TensorFlow



安裝 TensorFlow GPU 版

```
> conda install tensorflow-gpu
```

要先更新 NVIDIA GPU 顯卡 Driver

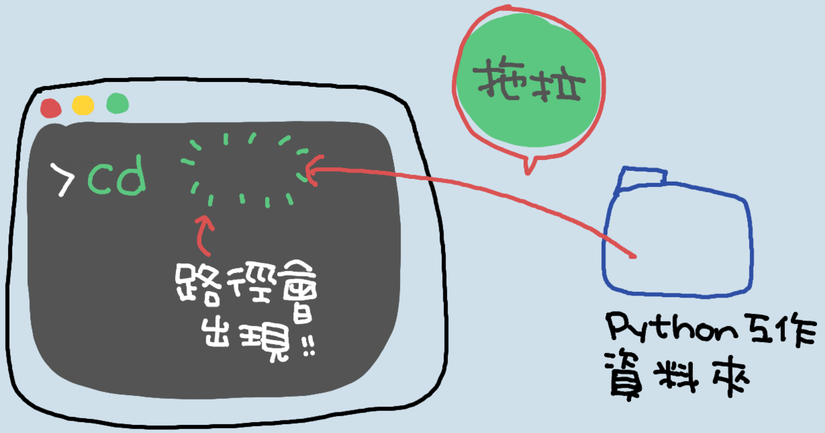
離開虛擬環境

```
> conda deactivate
```

以後直接打開終端機 (Anaconda Prompt Powershell), 直接做以後一直會做的標準動作...

cd 進入你的工作資料夾

拖拉神技出現工作資料夾



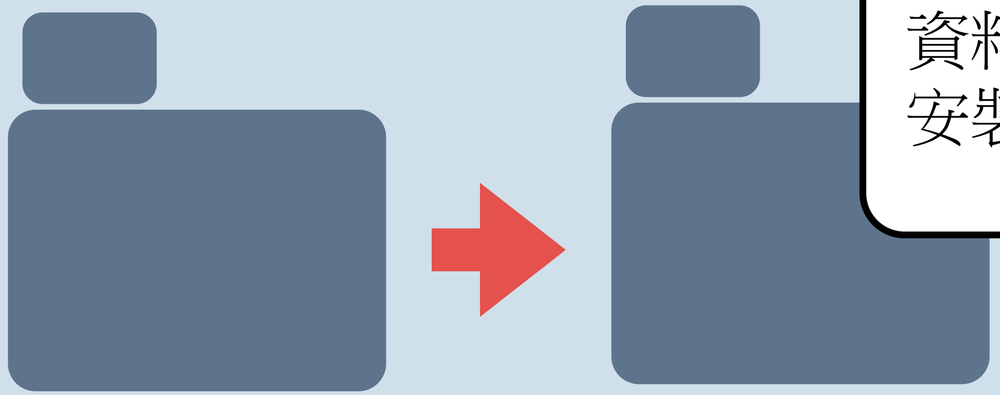
用拖拉的方式，可以找到完整路徑！

執行 Jupyter Notebook

```
> jupyter notebook
```

Anaconda 重裝

Anaconda 重裝



把家目錄中叫 **anaconda3** (之類的)資料夾更名。然後重新安裝 Anaconda 就好!

02.

用 Colab 免安裝

其實也可以直接用 Colab

Colab

不用安裝, 用 Google 帳號登入, 免費使用 GPU 或 TPU。

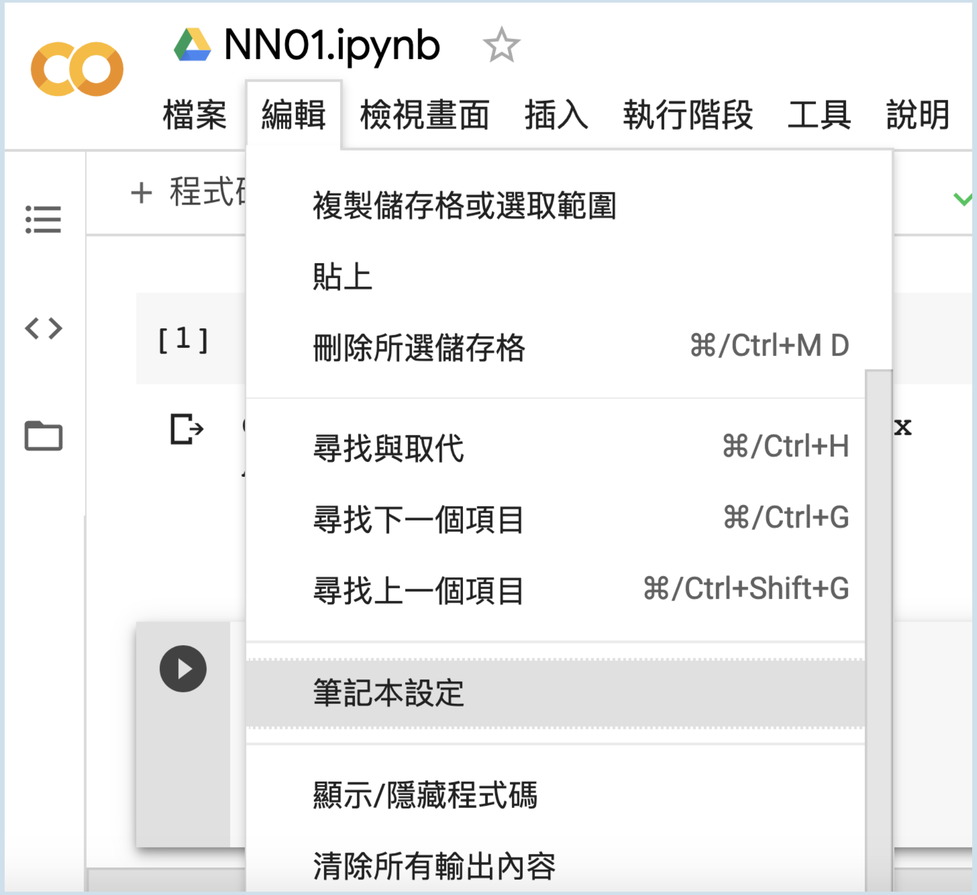
檢查 TensorFlow 版本

```
[1]  %tensorflow_version
```

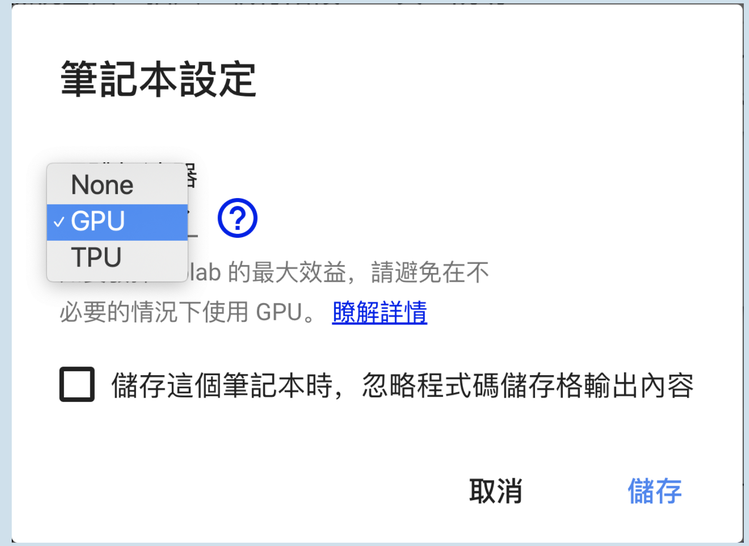
```
Currently selected TF version: 2.x
Available versions:
* 1.x
* 2.x
```

檢查 TensorFlow 版本 (現在應該預設就是 2.x)

打開 GPU



在編輯 > 筆記本設定中，
打開 GPU 或 TPU。



Mount 你的 Google Drive, 存取資料

```
from google.colab import drive

drive.mount('/content/drive')
```

要點網址 copy 金鑰

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au

Enter your authorization code:
```

你會需要點網址, 選擇你的 Google 帳號授權,
然後 copy 出現的金鑰, 回來貼在這裡。

進入你的資料夾

```
[3]  %cd '/content/drive/My Drive/Colab Notebooks'
```

```
/content/drive/My Drive/Colab Notebooks
```

你可以 `cd` 進入任何你 Colab 的資料夾。前面的部份我們沒有要可愛，
Google 真的把你的 Google Drive 家目錄叫 **My Drive**。

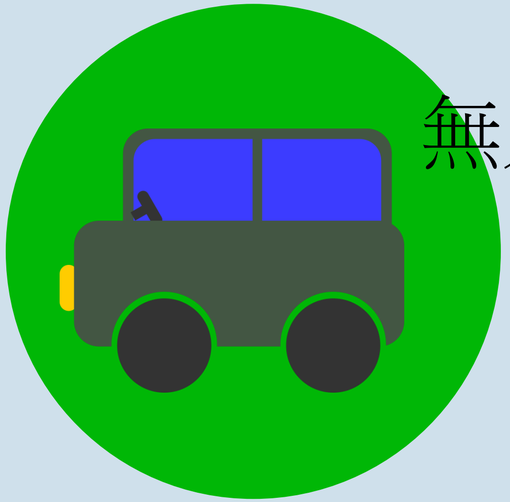
03.

關於 AI 的種種傳聞

傳聞說...

很多人都說 AI 會取代我們

AI 事實上也沒那麼可怕...



無人駕駛

無人商店

自動交易

近 100% 機會會實現

協助醫生診斷

協助音樂創作
Alex da Kid “Not Easy”

何況近期內還做不到...

哆啦 A 夢

不過我們還有 92 年的時間

2112 年 9 月 3 日

雖然 Yann LeCun 對通用型 AI 這個詞有意見...

No, No. 應該叫

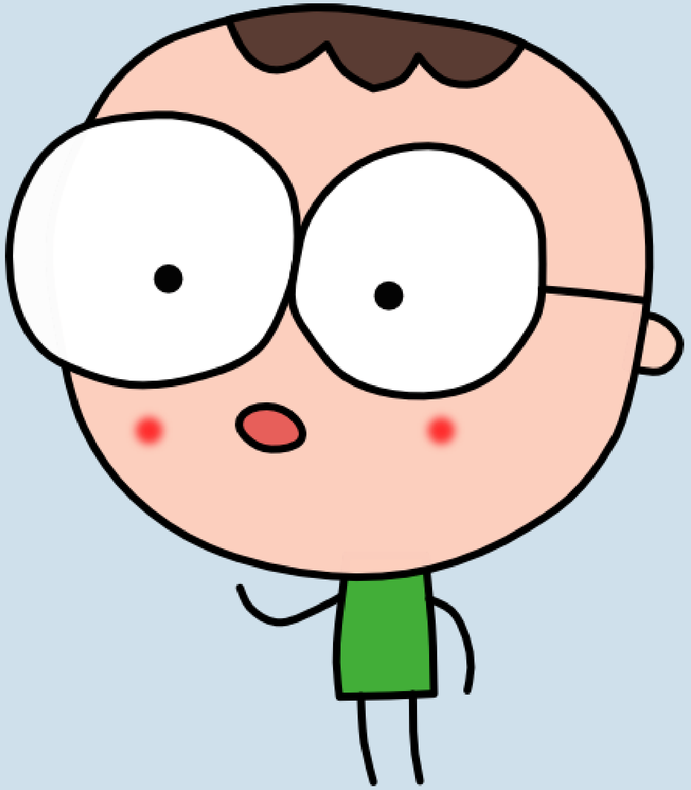
Human Level Intelligence

Yann LeCun

被稱為 deep learning 三巨頭之一、CNN 之父。

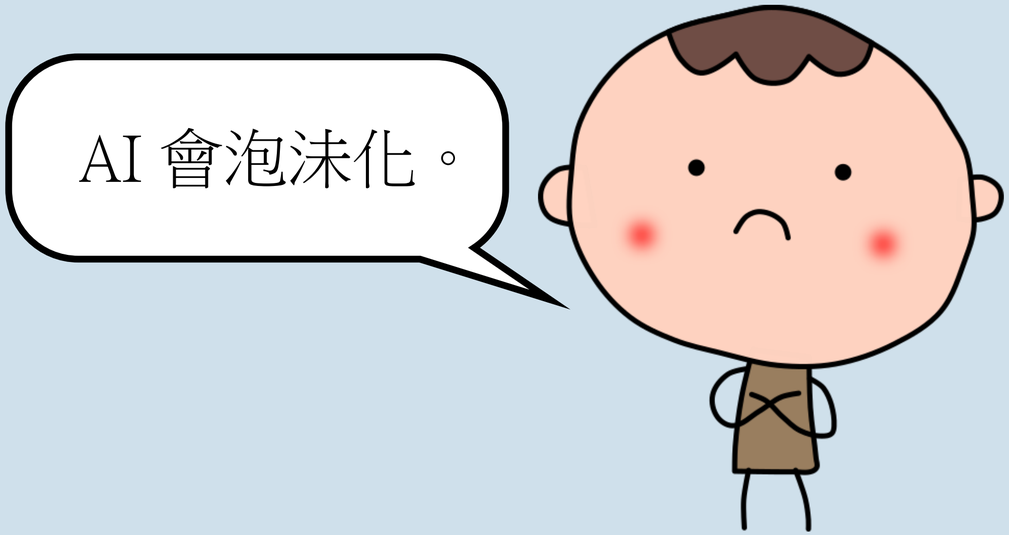
不論如何, AI 不像許多人說得那麼誇張...

現代的 AI 核心是神經網路、
其實沒什麼神奇的地方...



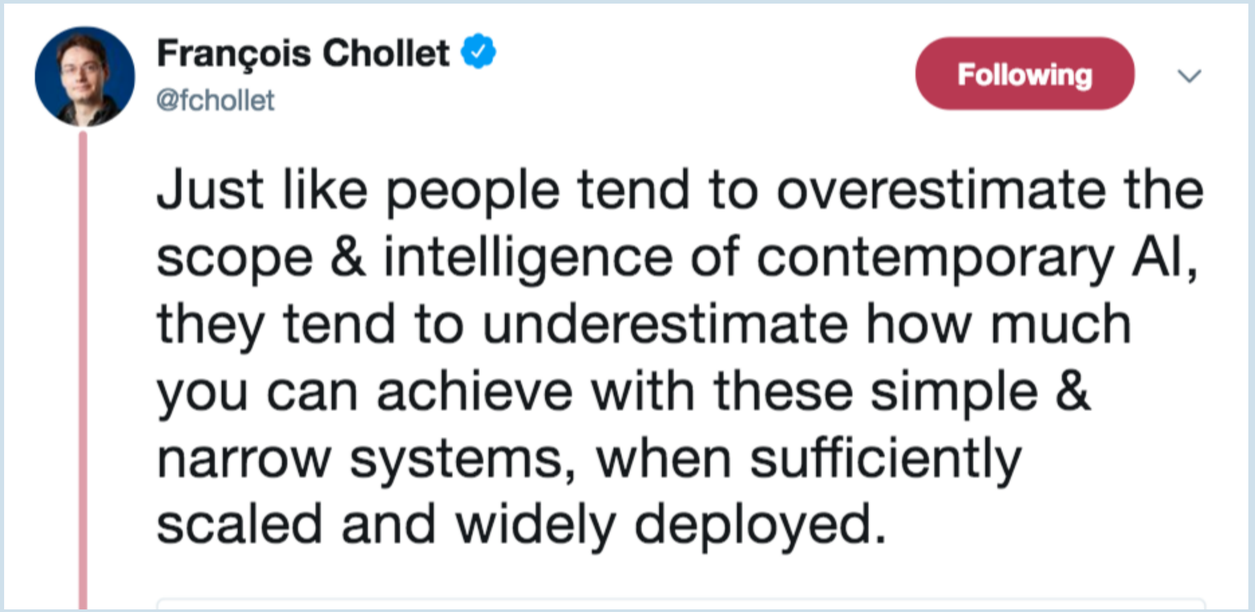
甚至可能會有點令人失望...

很多人太過低估 AI 的影響。



* 因 AI 大規模失業的事件已不斷在發生中...
* 當然有人整天用吹牛做 AI 的是很可能泡沫化。

其實現在的 AI 技術已可以做到很多事...

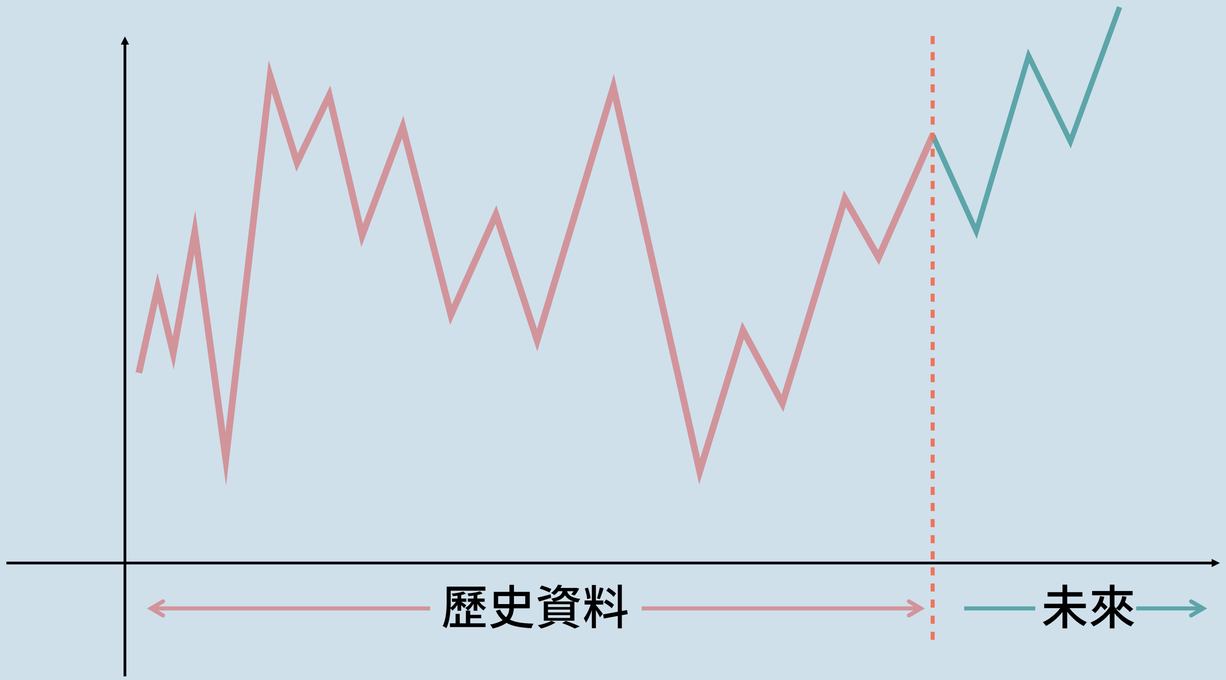


AI 的重點在「預測」

> The current wave of advances in artificial intelligence doesn’ t actually bring us intelligence but instead a critical component of intelligence: prediction.

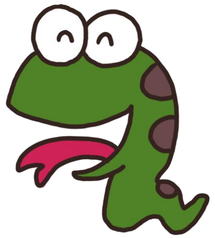
—Agrawal-Gans-Goldfarb, “Prediction Machines”

這裡的預測並不只是對未來的預測

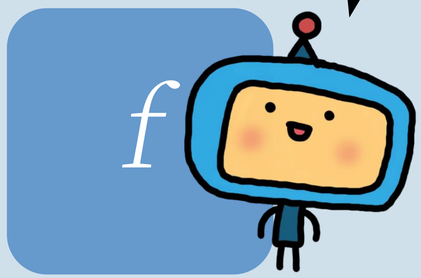


用時間序列歷史資料預測未來的當然叫預測。

還沒碰過的狀況也能正確判斷也叫「預測」

問題	答案
	台灣黑熊
	蟒蛇
⋮	

歷史資料



以前沒看過這張照片，但我知道是台灣黑熊!

深度學習其實人人可以做, 也該人人都來做!

吳恩達 (Andrew Ng)

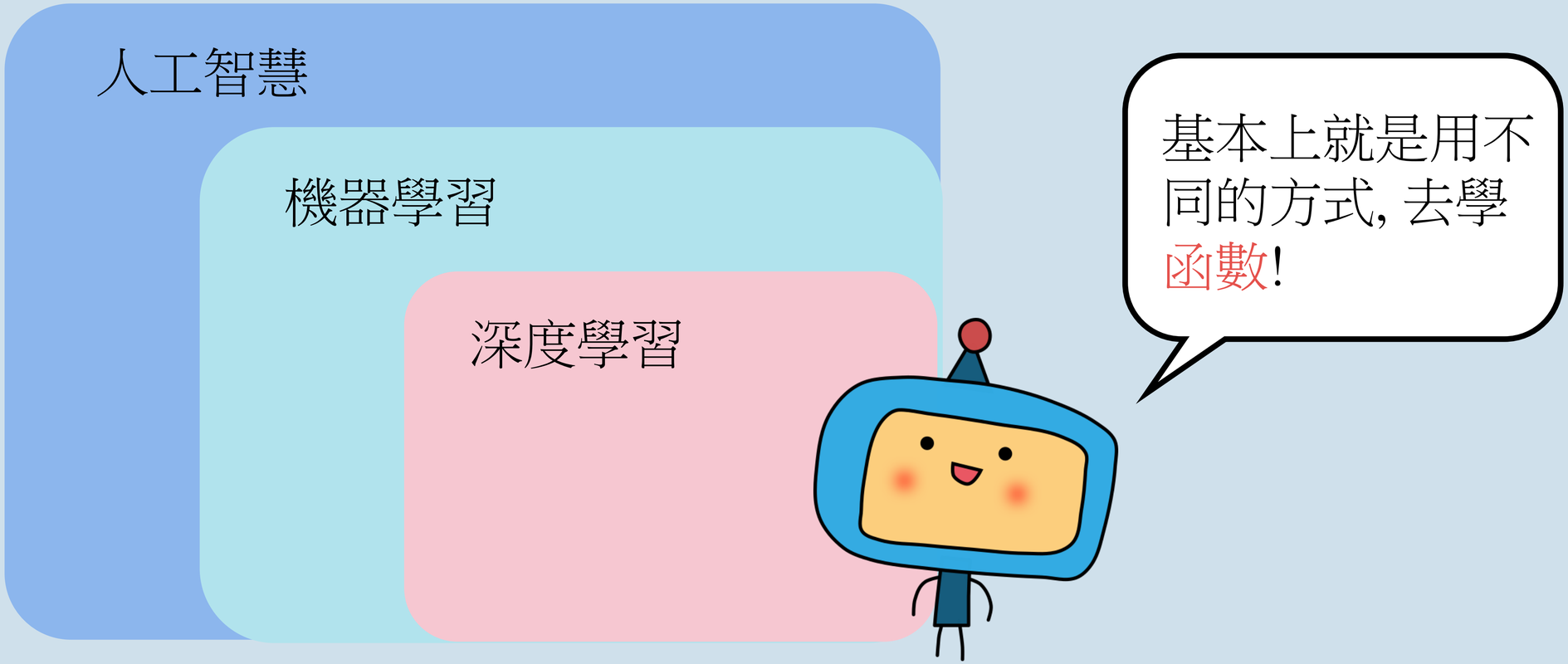
AI for Everyone

Coursera 課程

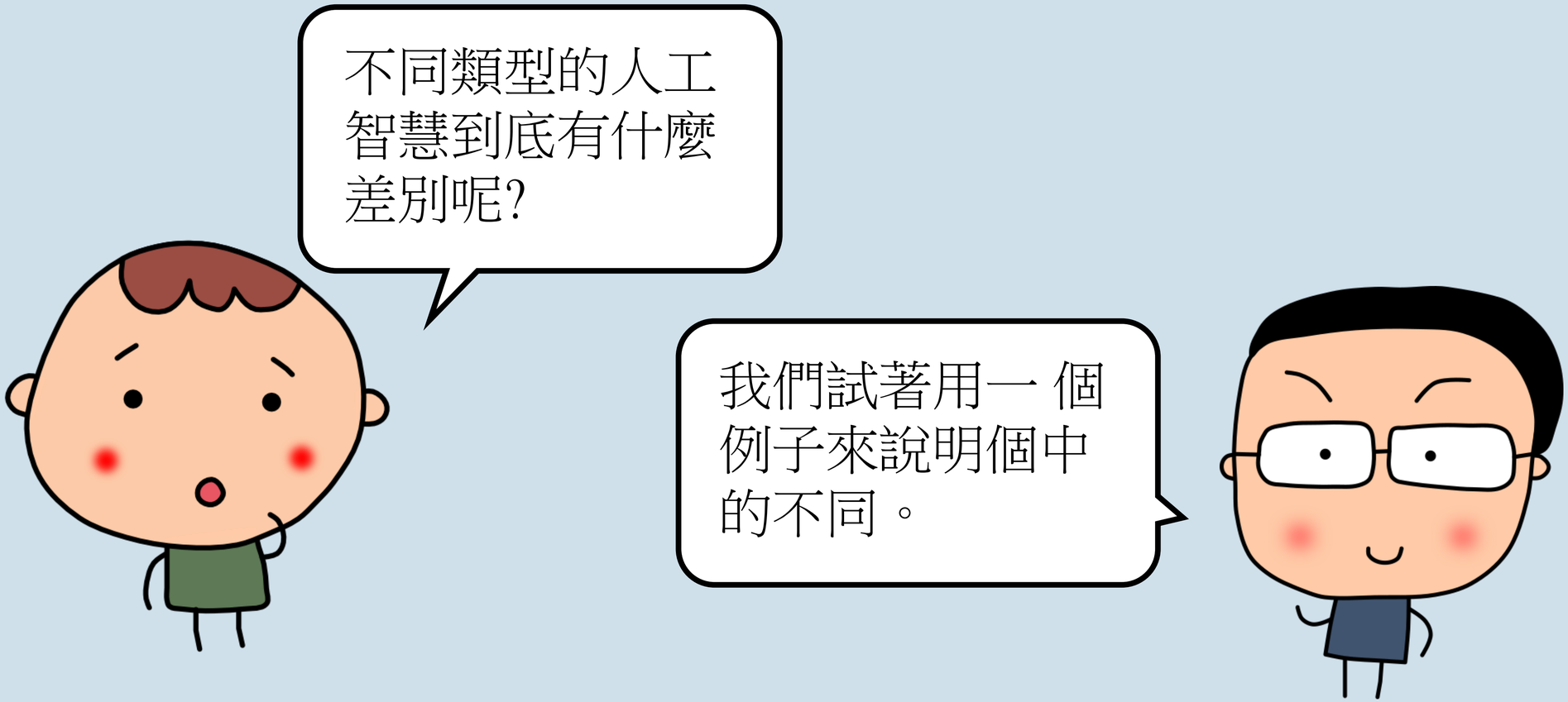
04.

不同的 AI 差在哪?

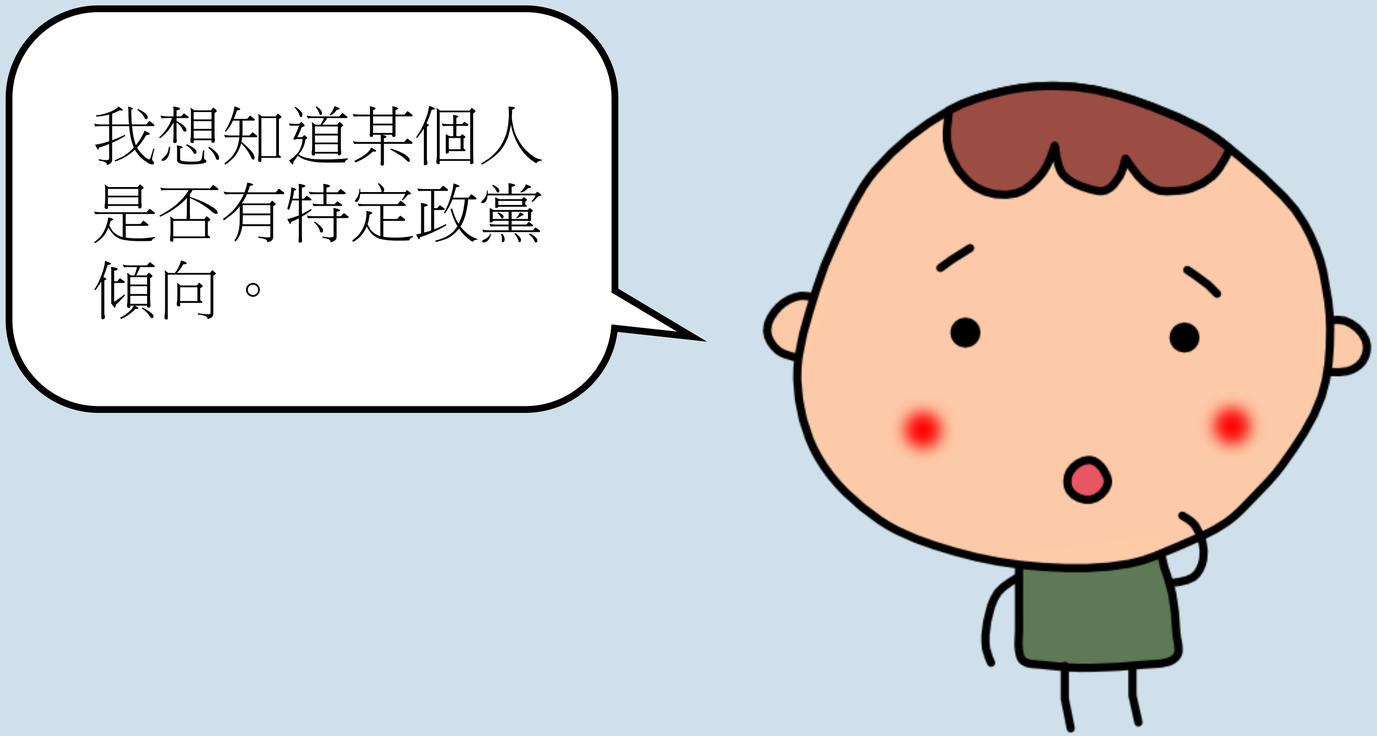
AI 的類型



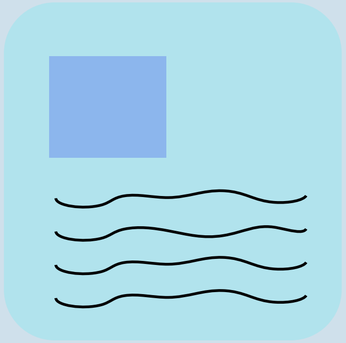
不同類型的差別



先來問一個問題



把問題化為函數的型式



社群媒體、
部落格文章

f

有政黨傾向

or

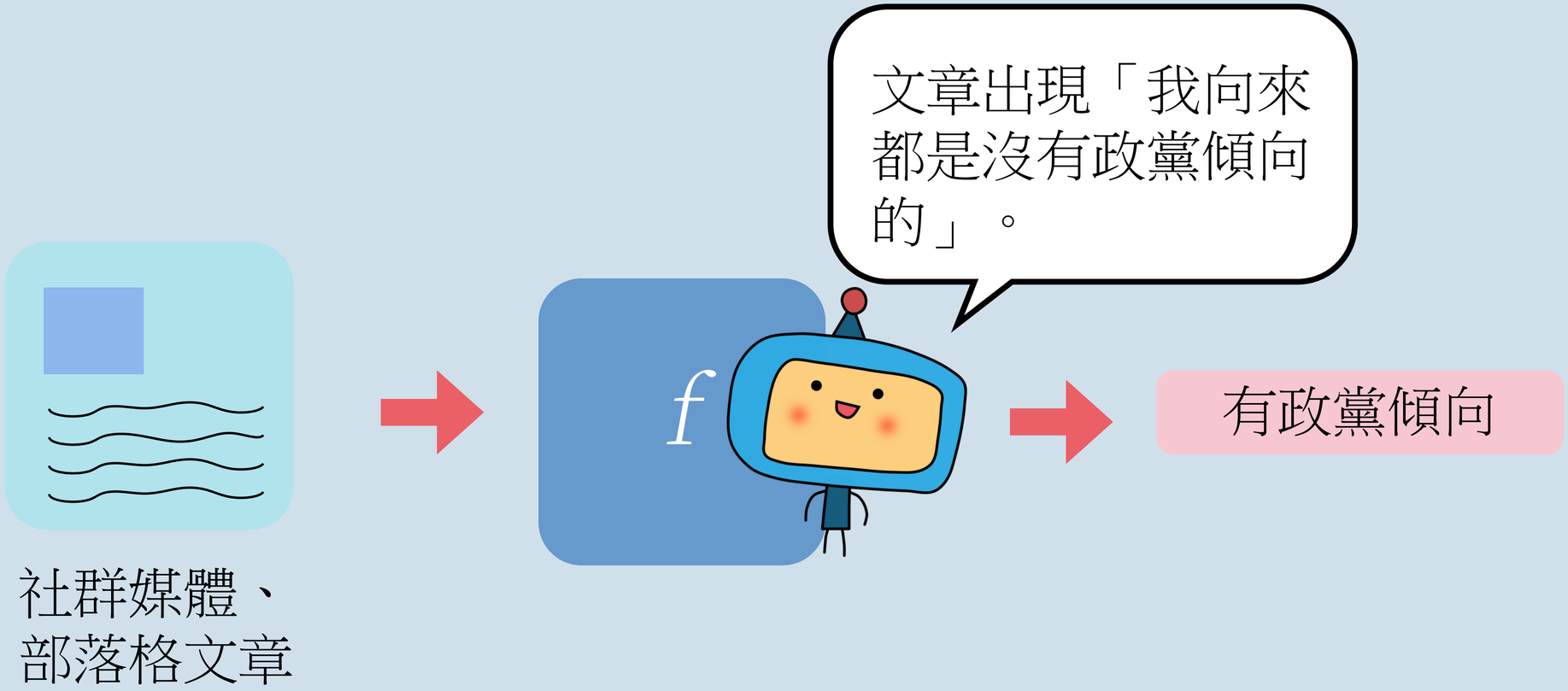
沒有政黨傾向

不管什麼 AI, 就是要去學這個函數, 但風格不同。

Type 1: 古典 AI

可能有個專家說, 如果文章出現自稱「中立客觀」,「沒有特定政黨傾向」...

Type 1: 古典 AI

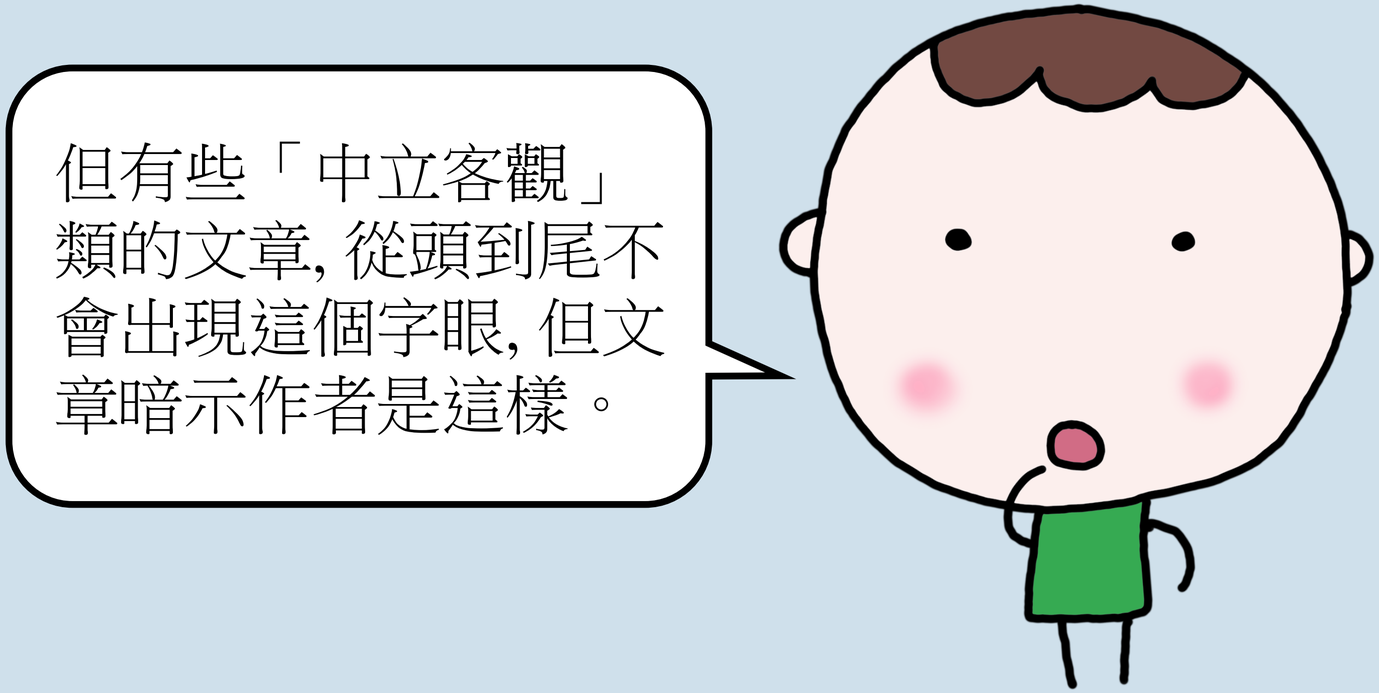


Type 1: 古典 AI

簡單的說, 就是規則基本上是我們告訴電腦的。也許比我們例子更複雜, 也許比我們例子更簡單 (比如說一個公式就可以算出來的問題)。

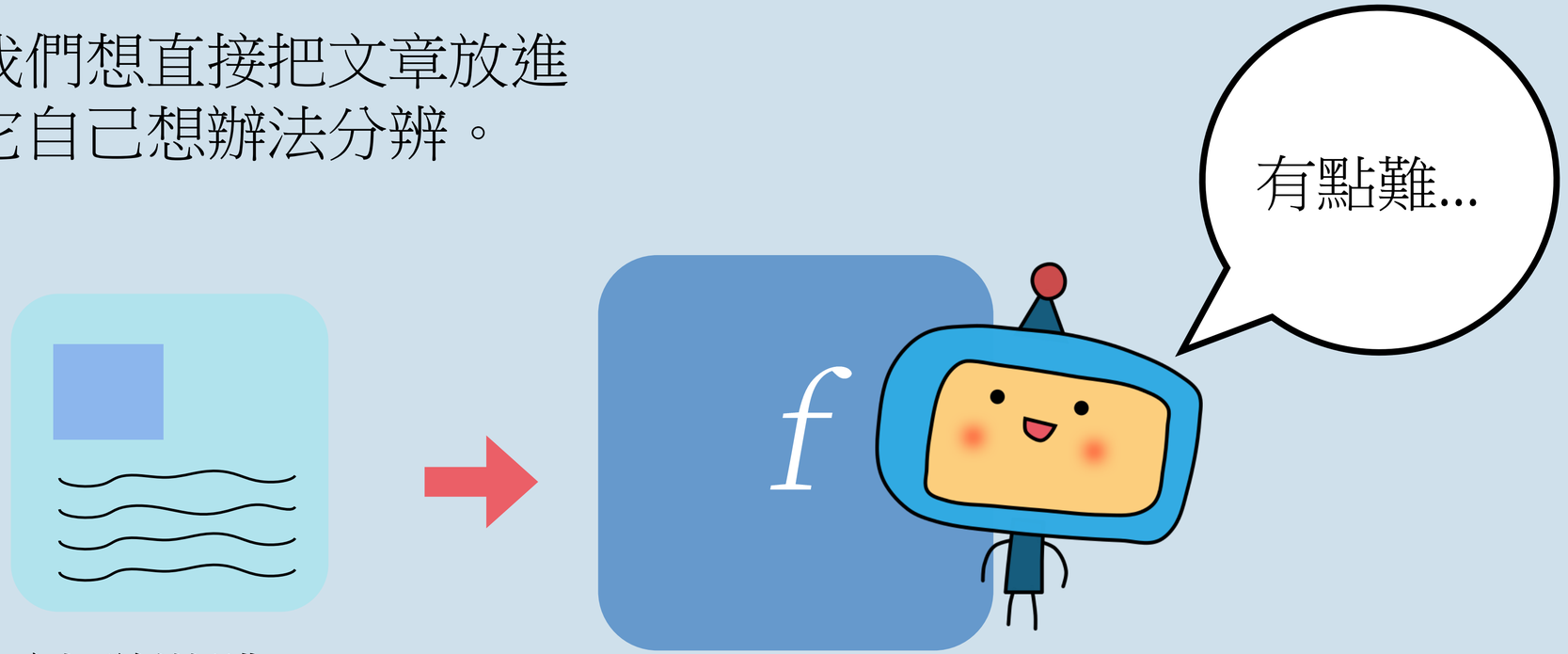
Type 2: 傳統機器學習

剛剛的方式, 我們可能會碰到一個問題...



Type 2: 傳統機器學習

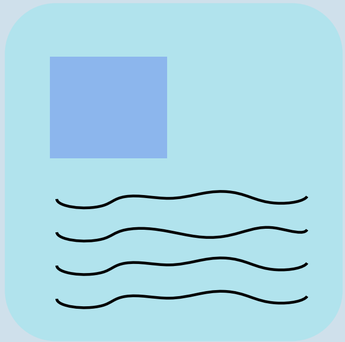
基本上, 我們想直接把文章放進電腦, 讓它自己想辦法分辨。



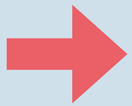
社群媒體、
部落格文章

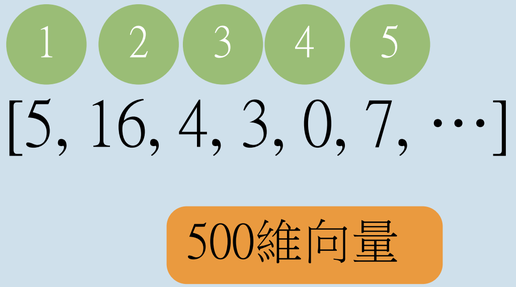
Type 2: 傳統機器學習

於字是我們想辦法把文章的特徵找出來,
比如說最重要 500 個詞出現次數。



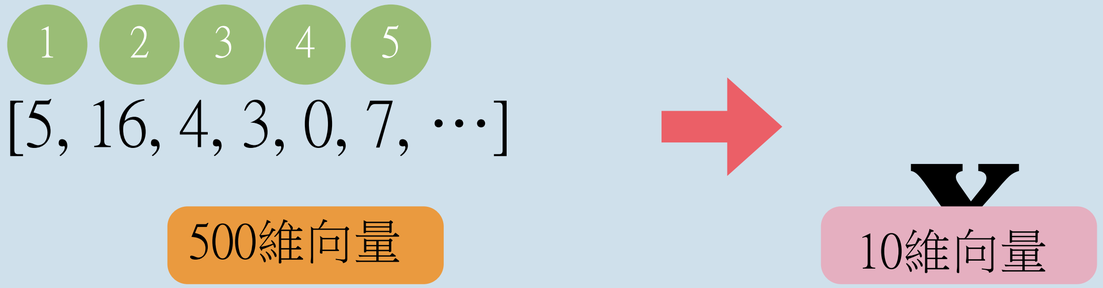
社群媒體、
部落格文章





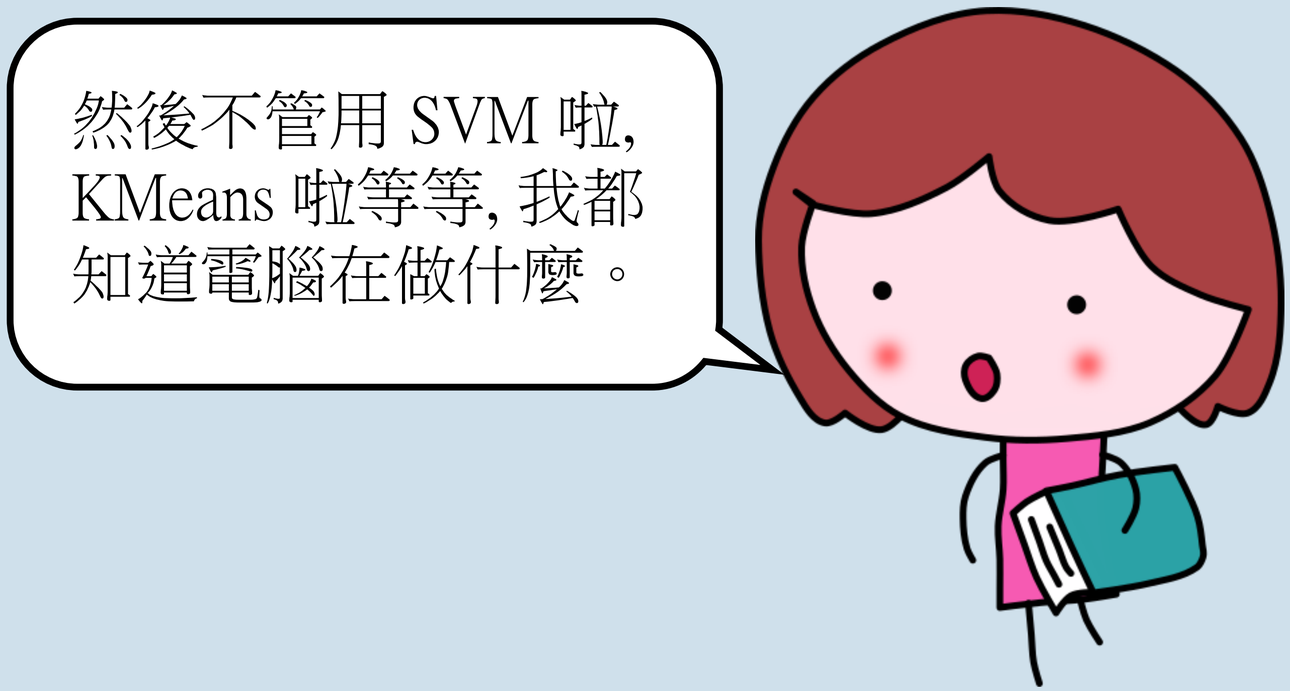
Type 2: 傳統機器學習

500 維可能還是太多了! 於是我們會用各種降維 (dimension reduction) 的方式, 把輸入縮小到 (例如) 10 維向量。

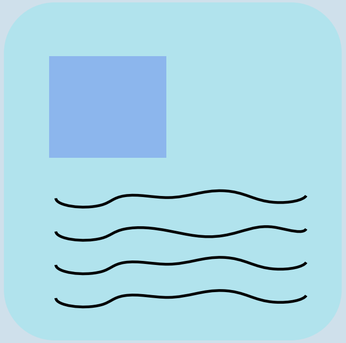


Type 2: 傳統機器學習

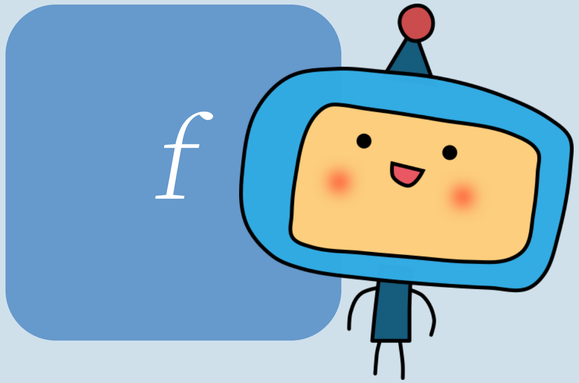
一般我們會花不少時間去做所謂的 feature engineering, 找到輸入資料較好的表現方式。



Type 3: 深度學習



社群媒體、
部落格文章



有政黨傾向

or

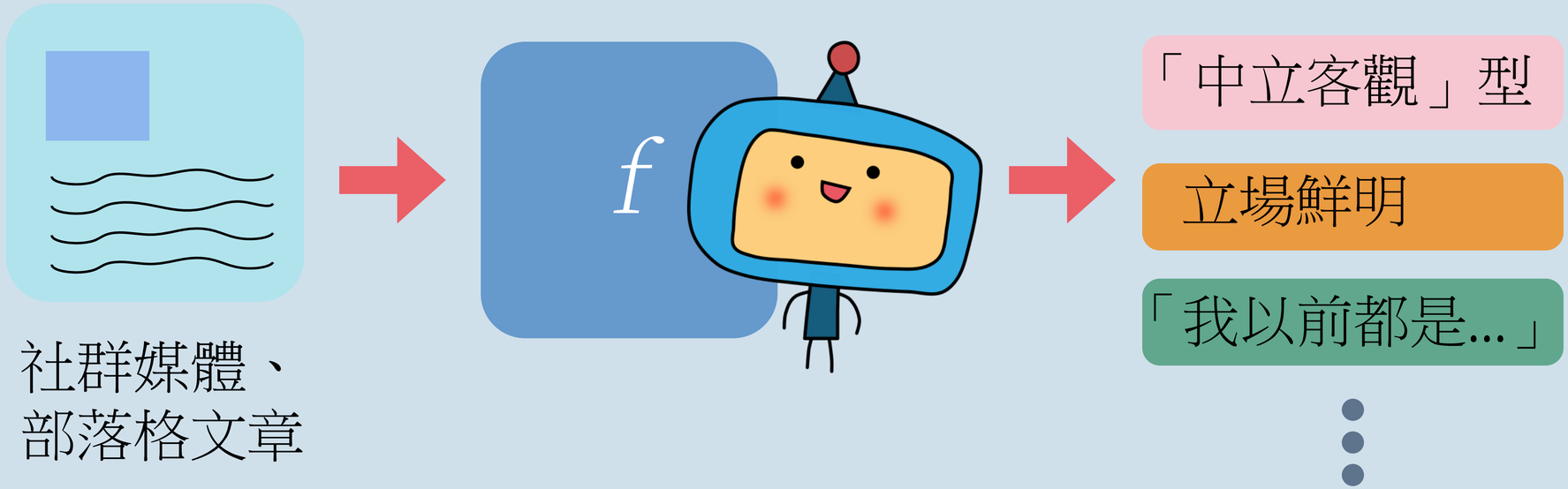
沒有政黨傾向

就把一篇篇我們知道分類的文章放進去訓練...

The End?

Type 3: 深度學習

我們可能會發現, 這樣訓練效果不好, 或是提供給我們的資訊太少, 說不定應該訓練這種分類, 甚至多類別型的分類!

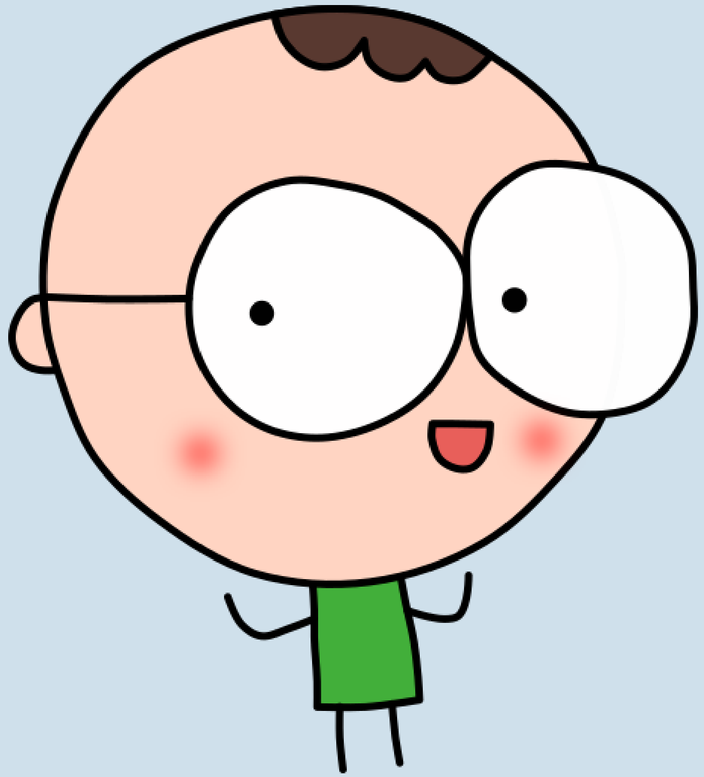


Type 3: 深度學習

深度學習會專注在
「我們怎麼問這個問題?」

整個判斷由電腦做, 所以
一般要大量資料!

小結論



- 例子不是真實專案, 領域專家可能會覺得這樣分析很外行。
- 我們「中立客觀」的分析了三類人工智慧方法的異同。
- 有時界線沒有這麼明顯, 甚至我們也常會混用三種方式。
- 不需要特別獨尊哪種方式, 每種方式都有適合的應用場景。

05.

AI 實作就是
打造函數學習機

函數是什麼?

函數其實是一個解答本

所有的問題, 都有一個答案

所以我們的問題都要化為函數的形式

然後用深度學習的方式打造一個函數學習機,
找出這個函數!

想個問題

在野外看到一隻動物,
我想知道是什麼?

想成函數就是...

輸入一張動物的照片，輸出就是這是什麼動物

f

“台灣黑熊”

我們有部份解答...

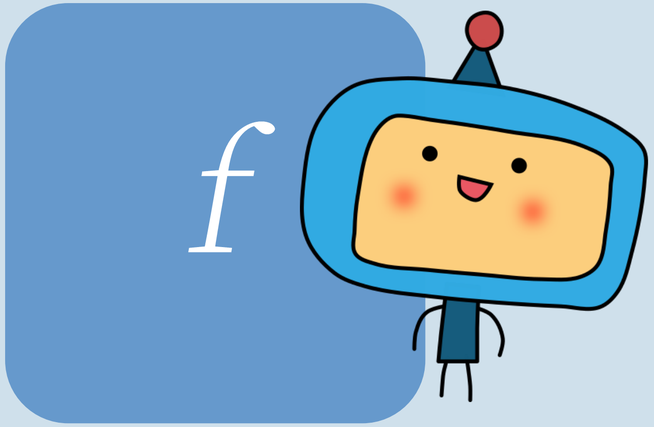
問題	答案
	台灣黑熊
	蟒蛇
⋮	

注意這個例子, 我們不可能收集到所有的情況!

也就是我們還沒碰過的情況可能有無限多題！

問題	答案
	台灣黑熊
	?
	蟒蛇

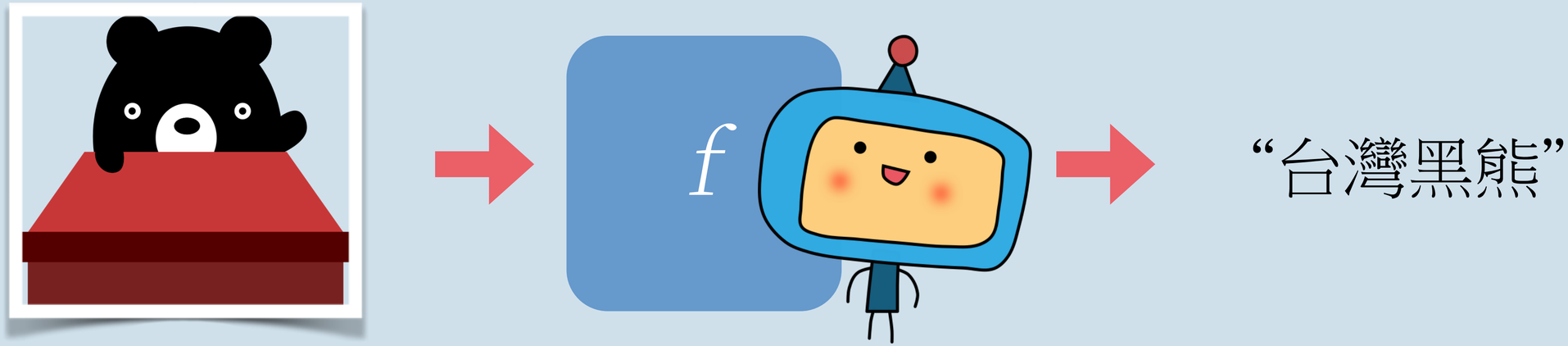
打造「函數學習機」，把函數完全學會!



我們打造一個「函數學習機」，把這個函數學起來!

(方法是不斷做「考古題」)

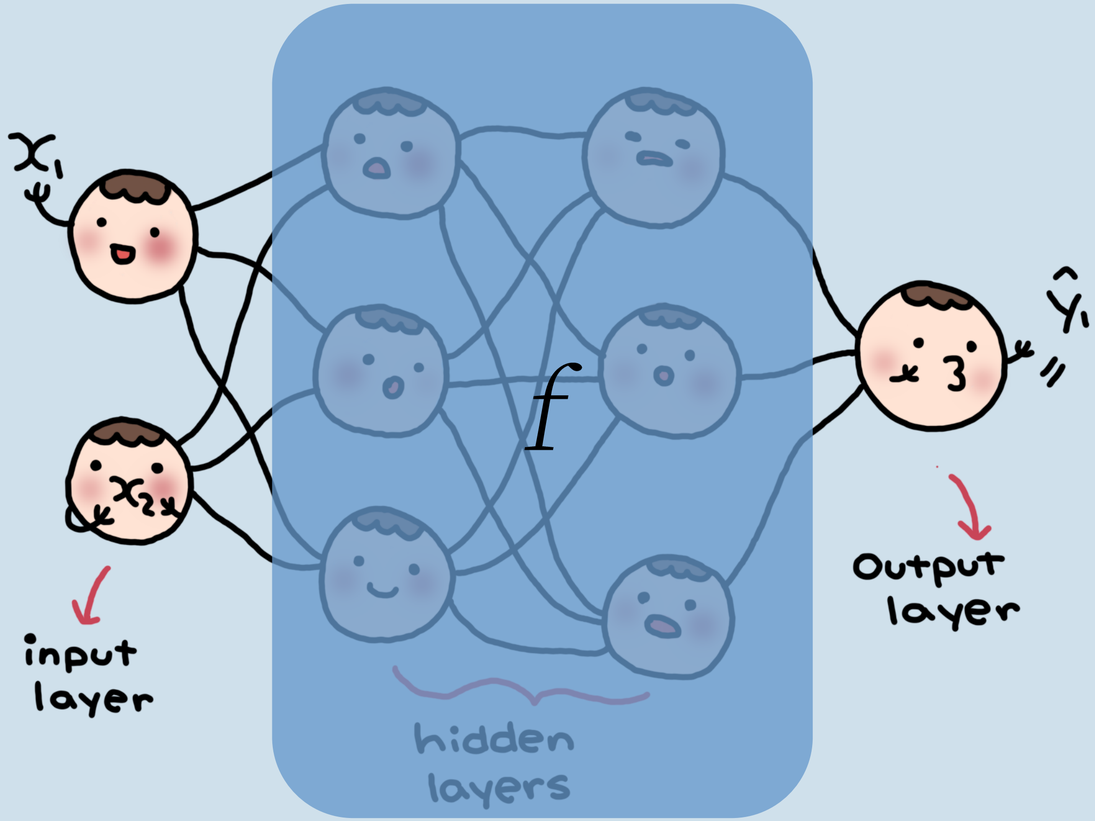
人工智慧



成功的話, 沒看過的照片也可以合理推論出來!

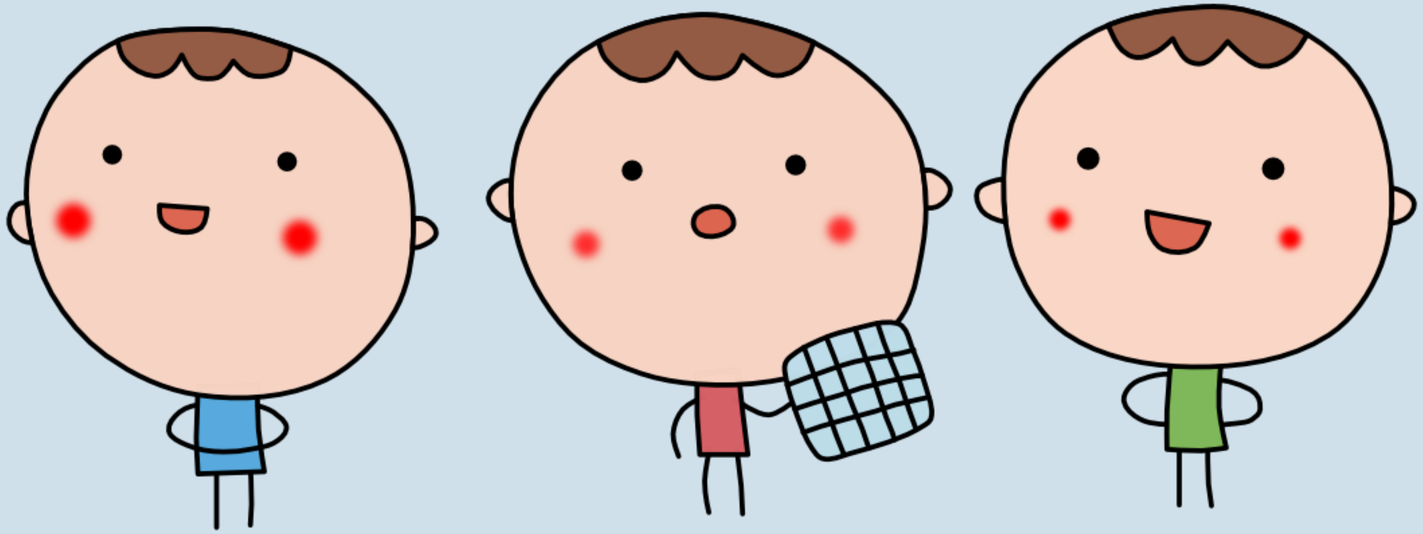
(所以叫「人工智慧」)

深度學習



深度學習就是用深度學習的方式去打造「函數學習機」，把這個函數完整的學起來!

深度學習 (神經網路) 的三大天王



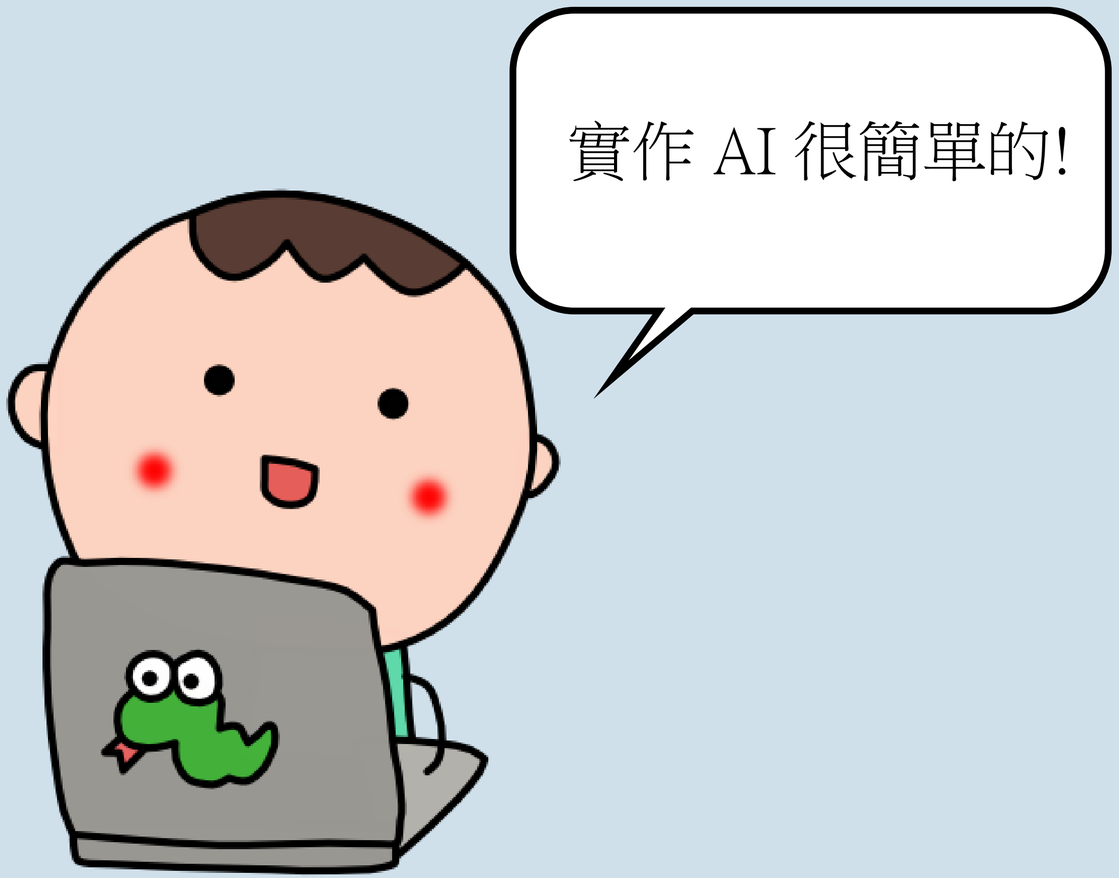
DNN　CNN　RNN

神經網路其實很簡單,
主要只有三種模式。

06.

AI 實作六部曲

AI 實作六部曲



1. 先問一個問題
2. 把問題化為函數的形式
3. 準備訓練資料
4. 打造「函數學習機」
5. 訓練學習
6. 預測

01 先問一個問題

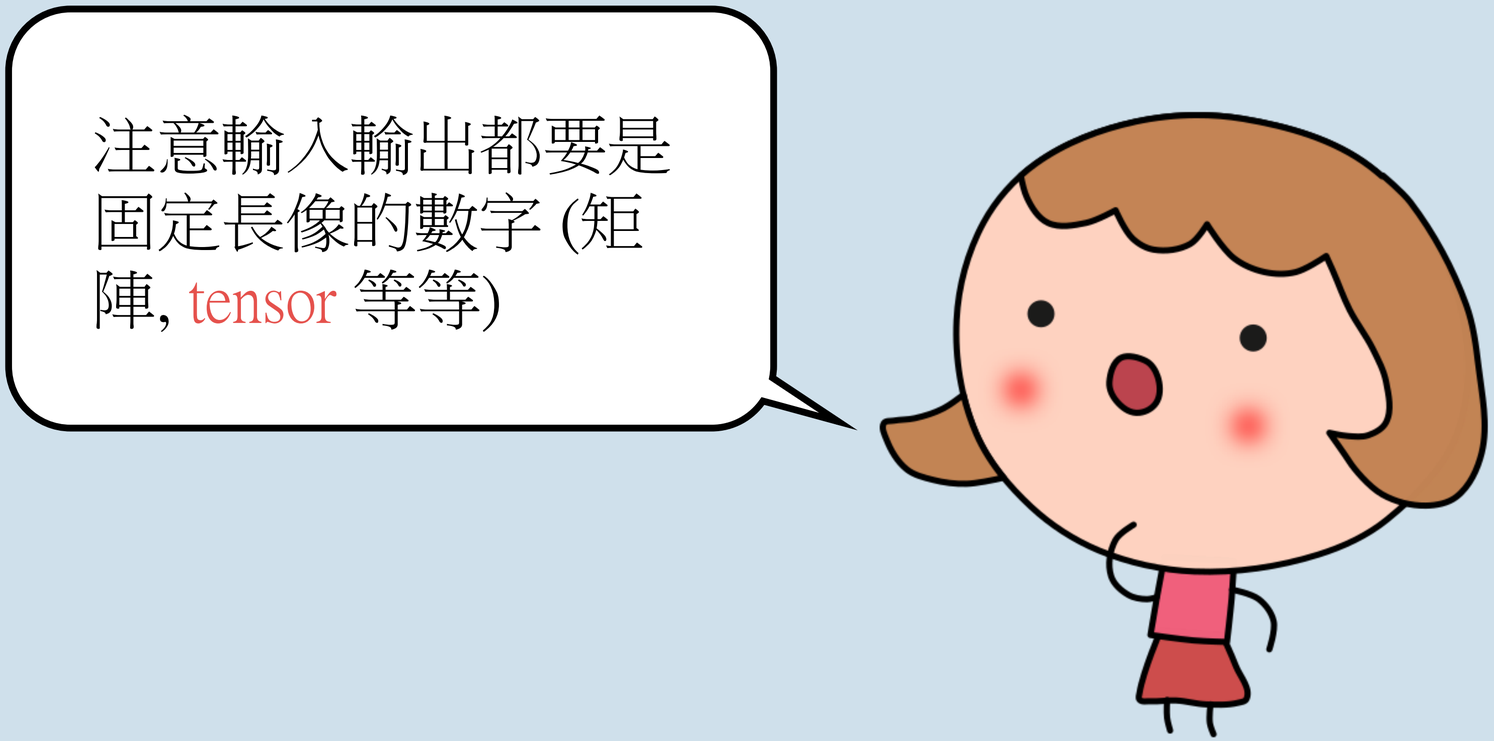
02 把問題化成函數的形式

f

“台灣黑熊”

02 把問題化成函數的形式

為了要讓電腦算, 我們會要求...



02 把問題化成函數的形式

還有輸入、輸出的「大小」基本上是固定的。

$X \in \mathbb{R}^n$

輸入

$Y \in \mathbb{R}^m$

輸出

02 把問題化成函數的形式

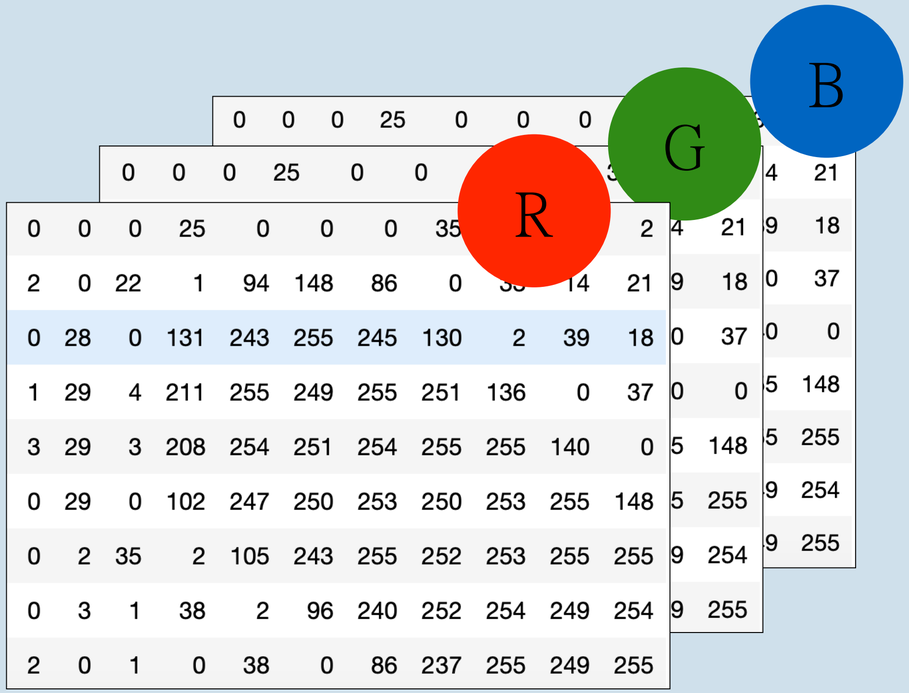
小事一件, 都叫 tensor (張量)。

純量 $x = 9487$ 0階 tensor

向量 $x = [9,4,8,7]$ 1階 tensor

矩陣 $x = \begin{bmatrix} 9 & 4 \\ 8 & 7 \end{bmatrix}$ 2階 tensor

02 把問題化成函數的形式



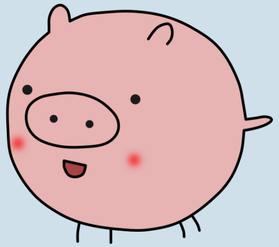
數位相片本來就是一堆數字

02 把問題化成函數的形式

1

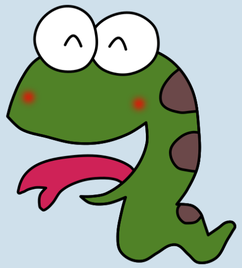
台灣黑熊

2



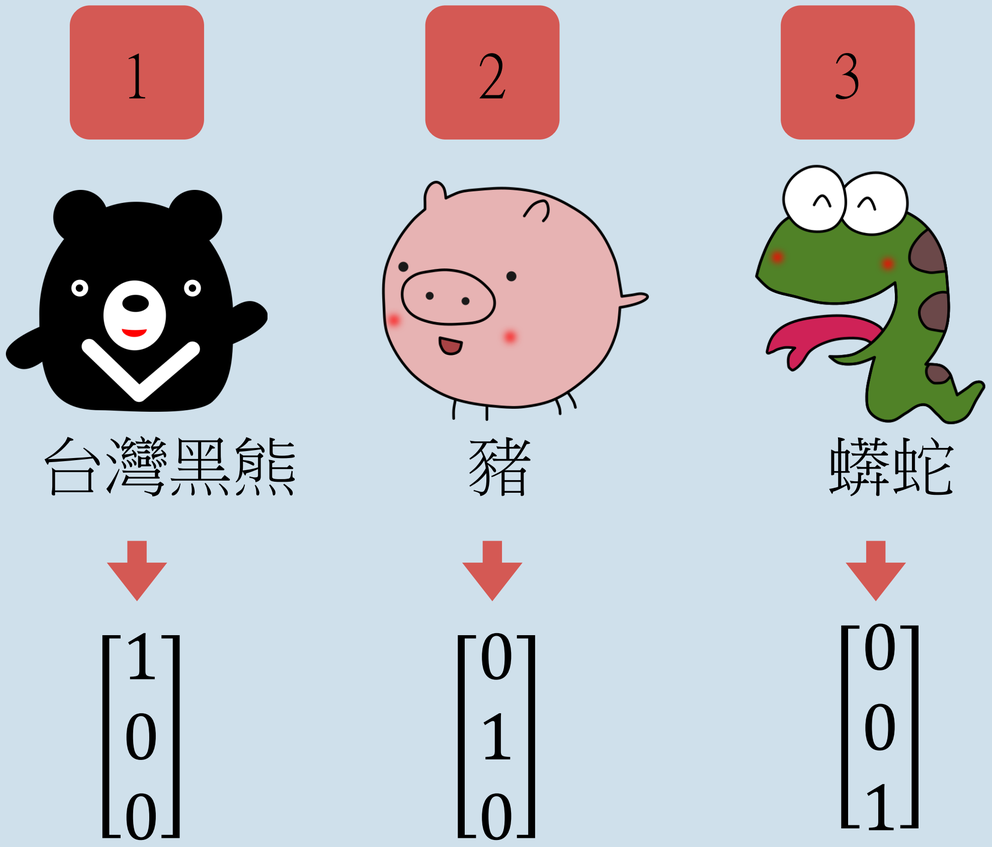
豬

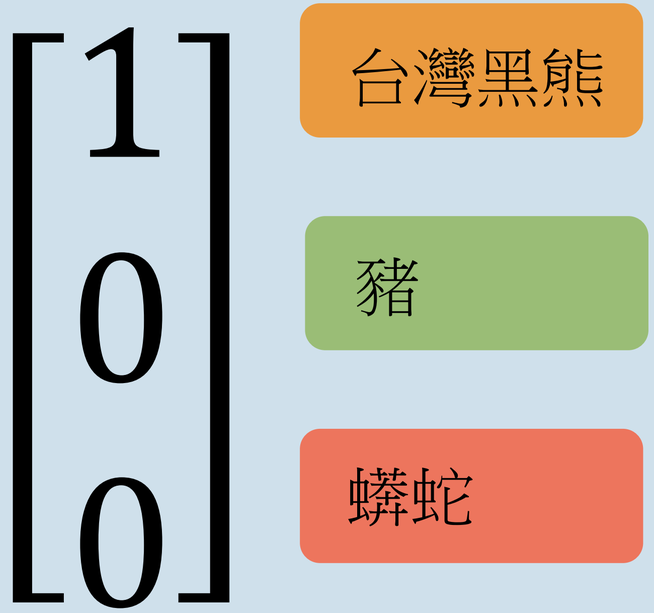
3



蟒蛇

02 把問題化成函數的形式

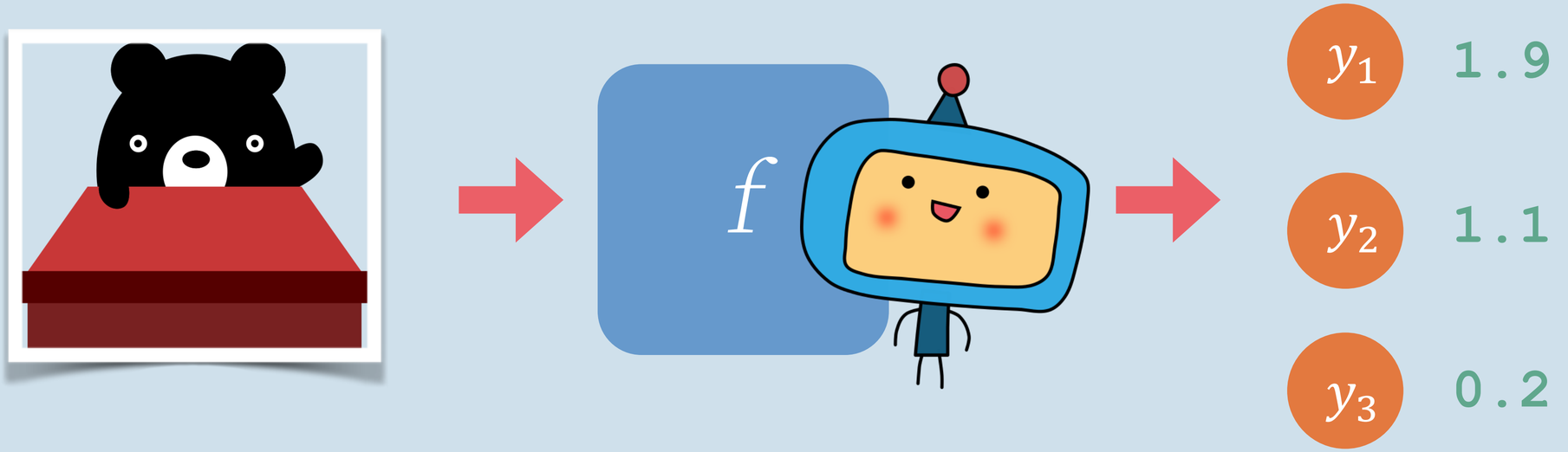




我們常把它做成 one-hot encoding

02 把問題化成函數的形式

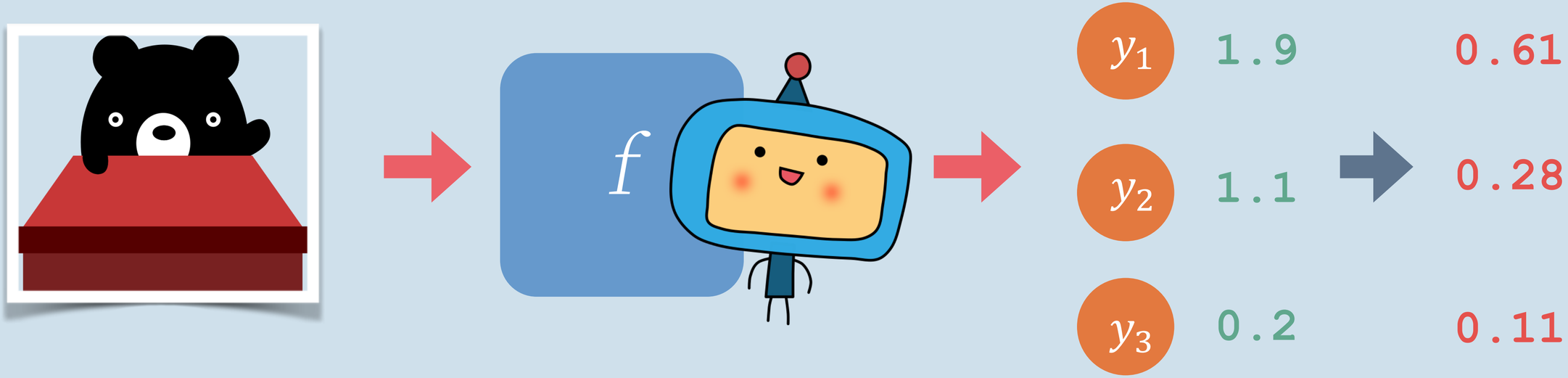
我們的例子, 化成 one-hot encoding 時, 函數學習機會要學輸入一張照片, 輸出三個數字的函數。比如說像這樣的輸出。



於是我們會知道, 電腦判斷最有可能是台灣黑熊。

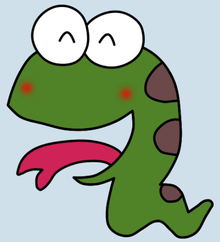
02 把問題化成函數的形式

我們甚至可以經過一個運算, 比方說 softmax, 讓三個輸出的數字加起來合為 1。

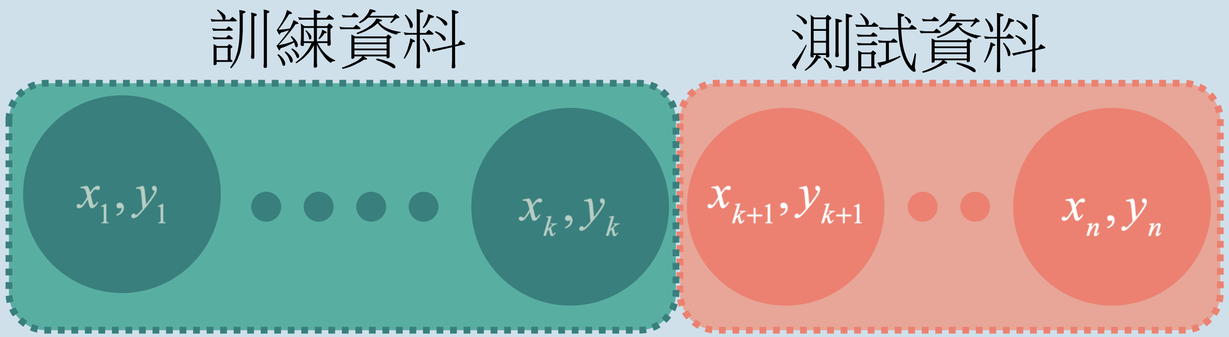


於是我們知道, 我們的神經網路判斷 61% 的可能是台灣黑熊, 28% 的可能是豬, 只有 11% 的可能是蟒蛇。

03 準備訓練資料


, "台灣黑熊"),

, "蟒蛇"), …

$\mathbf{x}_1$ $\mathbf{y}_1$ $\mathbf{x}_2$ $\mathbf{y}_2$

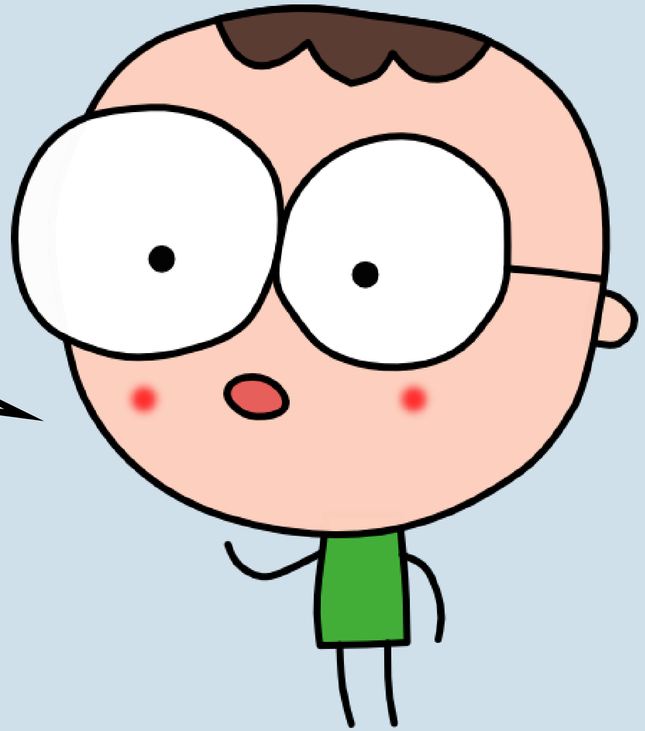


我們需要準備測試資料, 不參數訓練。目的是確認電腦沒有在「背答案」也就是沒有過度擬合 (over fitting) 的情況!

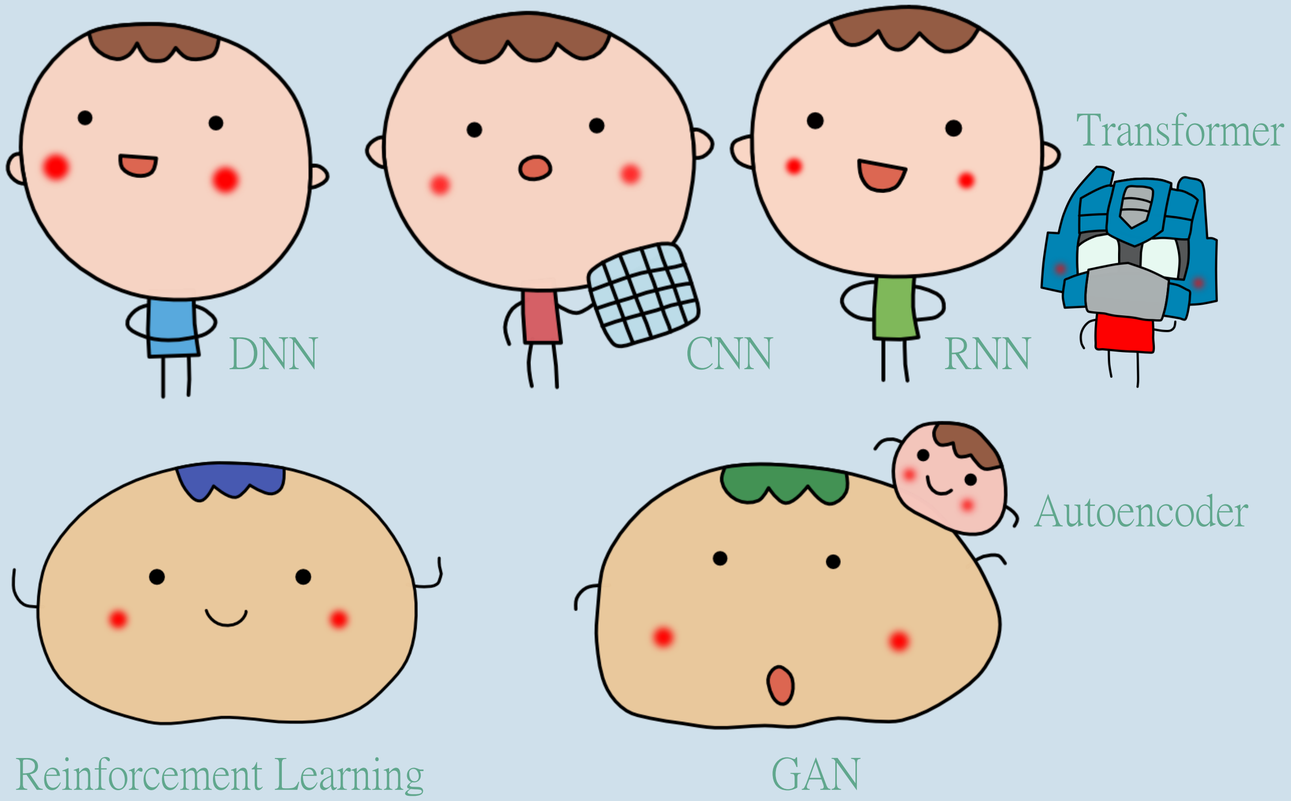
03 準備訓練資料

資料蒐集要注意...

1. 是否有夠多 (常常是上萬筆) 的歷史資料。
2. 是否在合理努力下可以取得這些資料。



04 打造「函數學習機」



深度學習就是我們用深度學習的方式, 打造一台函數學習機。

04 打造「函數學習機」

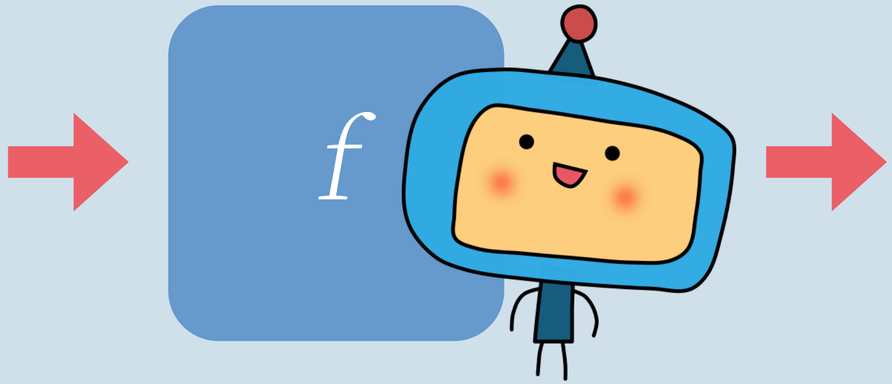
不管我們用了什麼阿貓阿狗法, 打造了一台函數學習機, 就會有一堆參數要調。

$$\theta$$

$$\{w_i, b_j\}$$

04 打造「函數學習機」

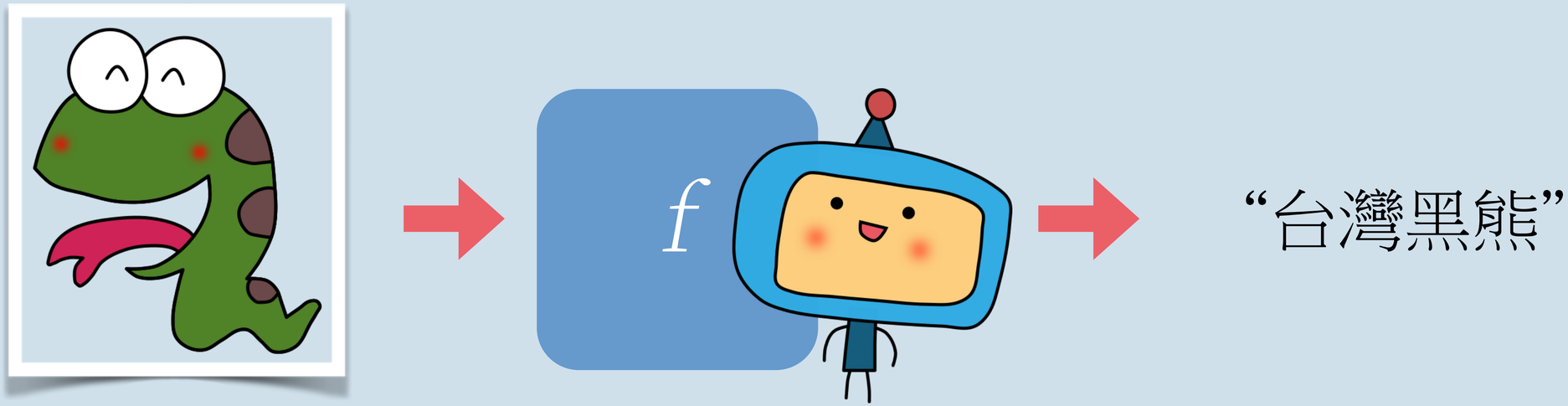
決定好這些參數的值, 就會出現一個函數。



白話文就是,「這個神經網路可以動了」...

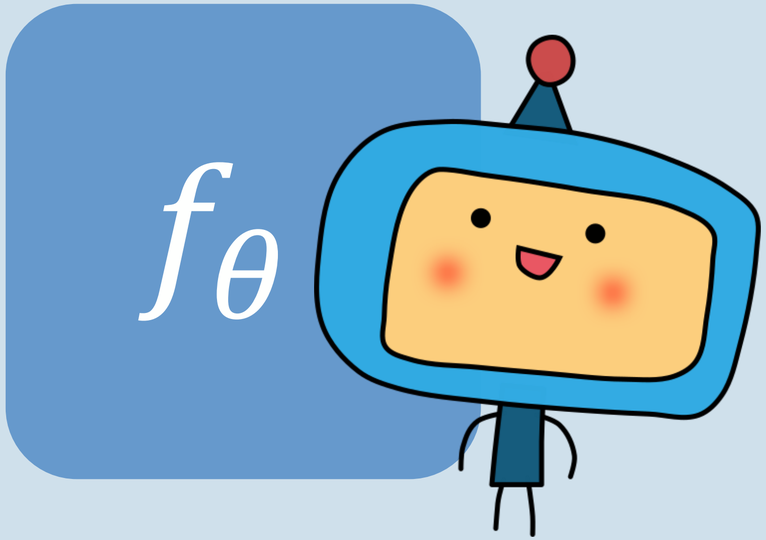
04 打造「函數學習機」

只是, 很有可能會發生這種情況...



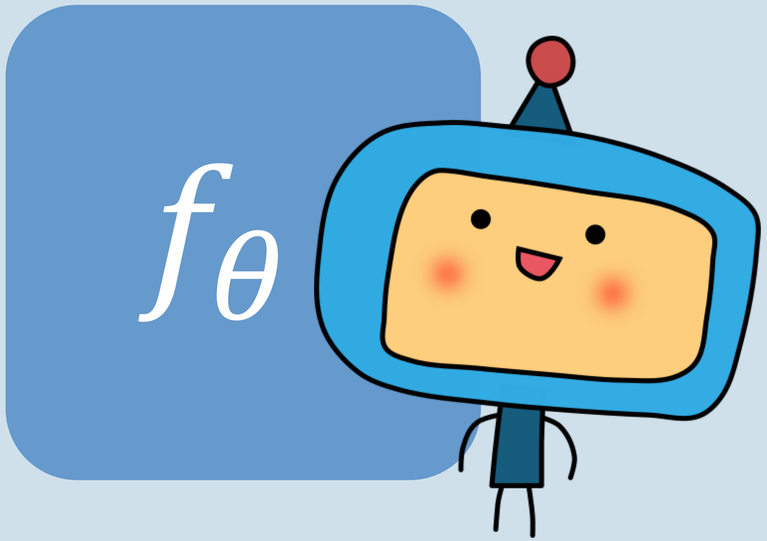
唬爛也要有個限度, 這該怎麼辦呢?

05 訓練 (學習)



當我們的參數 θ 決定了, 我們的函數學習機所學出的函數，就跟著固定了 (也就是每張照片會回答什麼就決定了)。

05 訓練 (學習)



我們會定義個 loss function, 看看我們的神經網路考考古題的時候, 和正確答案差多少?

$$L(\theta)$$

05 訓練 (學習)

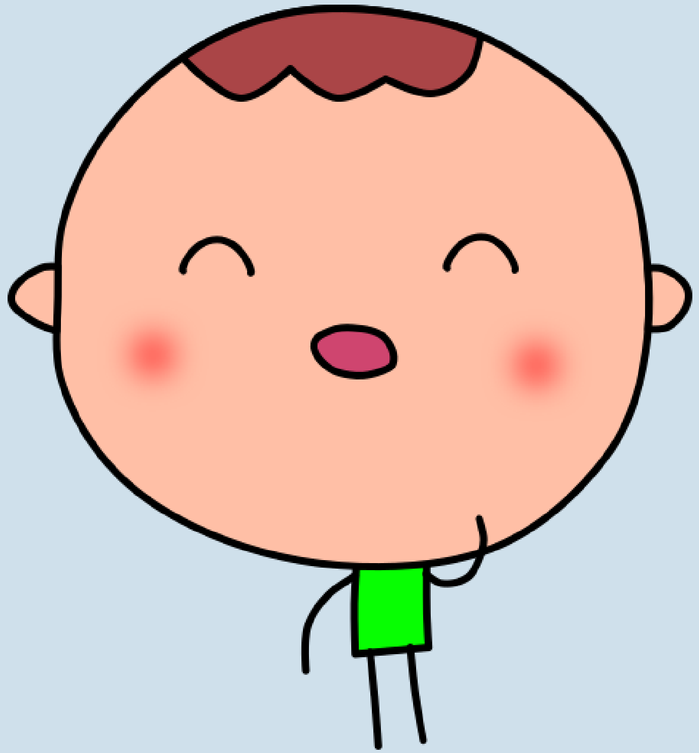
目標是找到一組

$$\theta^*$$

這組參數代入 loss function

$$L(\theta^*)$$

值是最小的 (也就是誤差最小)。



05 訓練 (學習)

基本上就是用

gradient descent

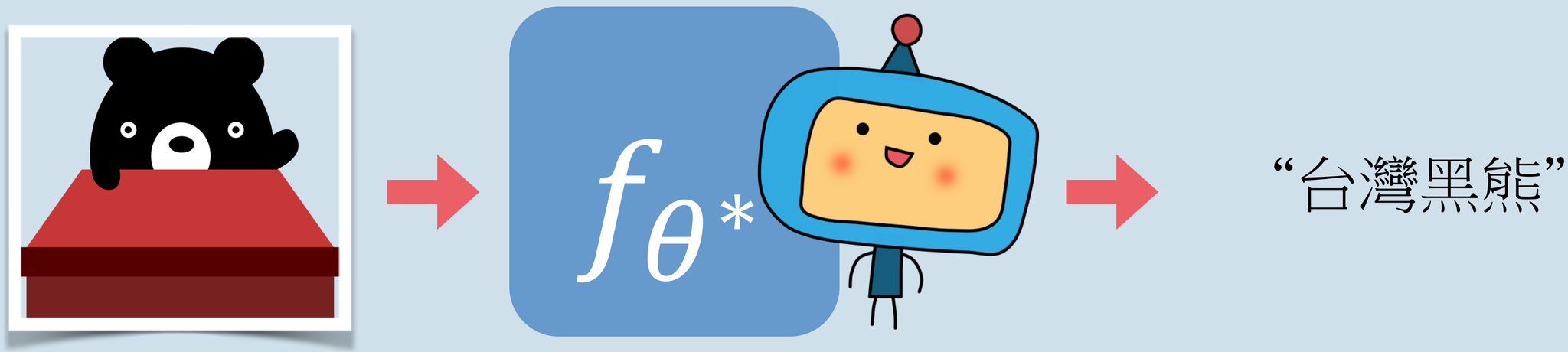
梯度下降法

因神經網路的特性, 也叫

backpropagation

反向傳播法

05 訓練 (學習)



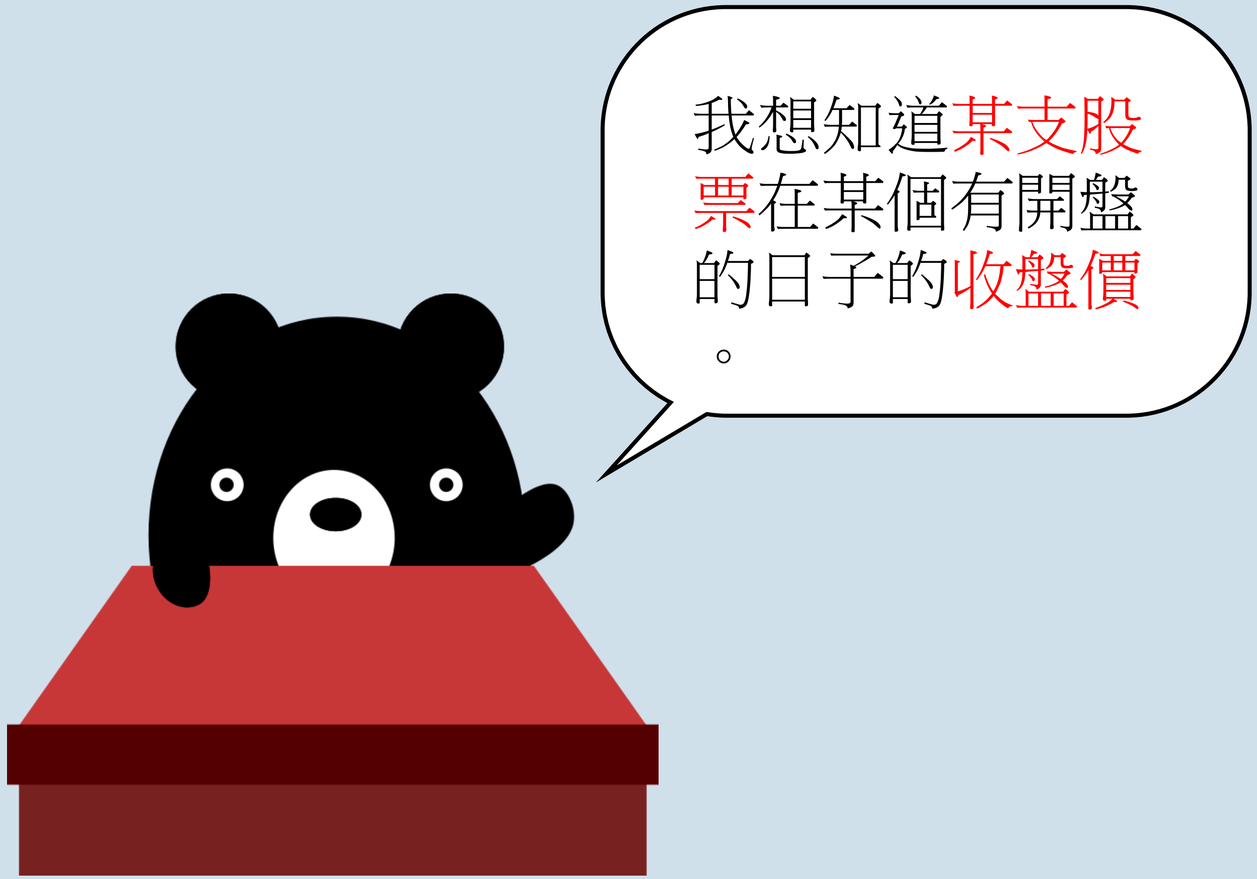
成功的話, 沒看過的也可以合理推論出來!

(所以叫「人工智慧」)

07.

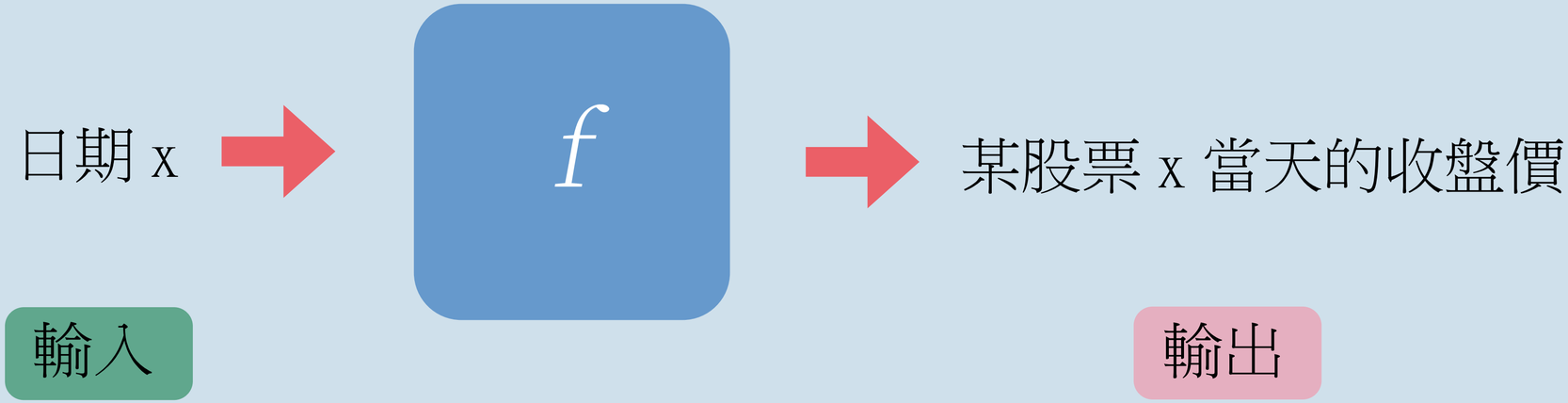
問問題的各種可能

01 股價預測



01 股價預測

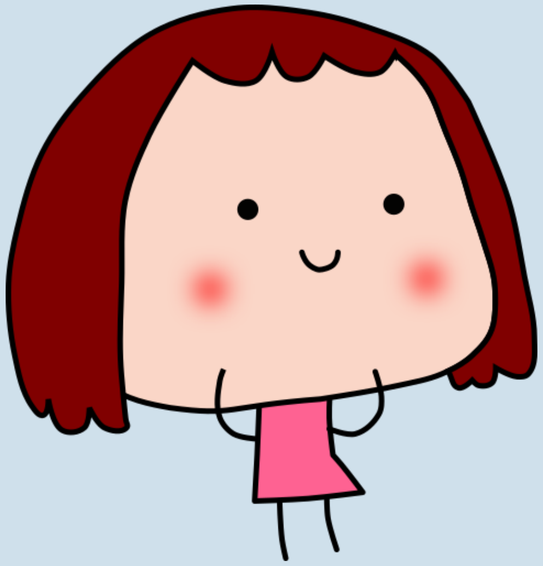
化成函數的形式, 我們當然可以很天真的這麼做...



不合理的是, 輸入日期這個資訊太少, 不太可能這樣就推出收盤價!

01 股價預測

用前 1 週的收盤價預測下一次的收盤價就合理得多。



$x_{t-1}, x_{t-2}, x_{t-3},$
x_{t-4}, x_{t-5}

輸入

f

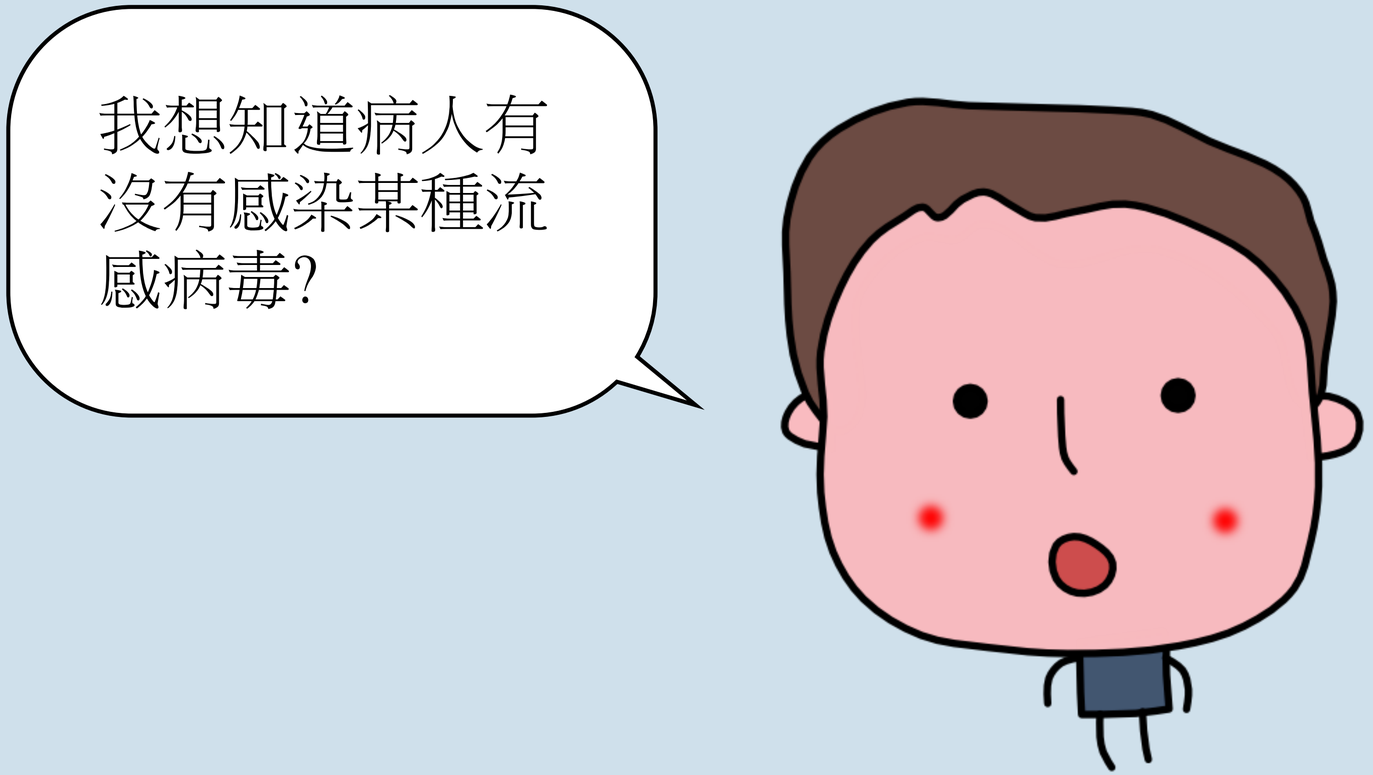
DNN, CNN, RNN

x_t

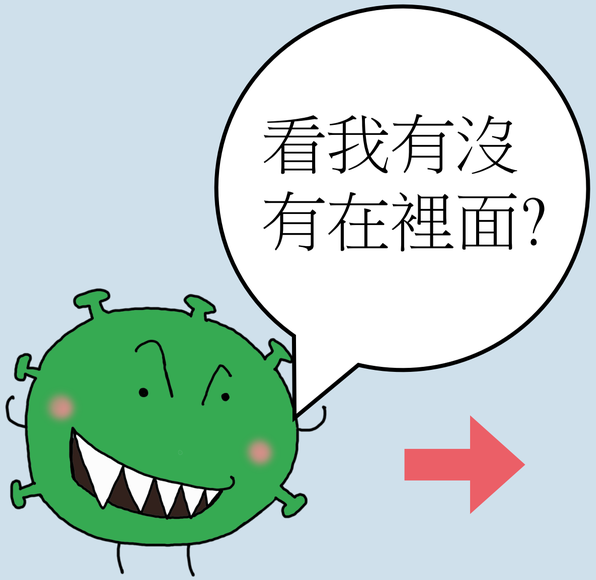
輸出

當然還有很多其他可能的方式!

02 流感病毒篩檢



02 流感病毒篩檢



輸入

f

CNN

有 or 沒有

輸出

03 全壘打預測

03 全壘打預測

某選手第 t-1 年資料 → f → 第 t 年全壘打數

輸入

RNN

輸出

每位選手都有 15 個 features!

```
[Age, G, PA, AB, R, H, 2B, 3B, HR,
 RBI, SB, BB, SO, OPS+, TB]
```

03 全壘打預測

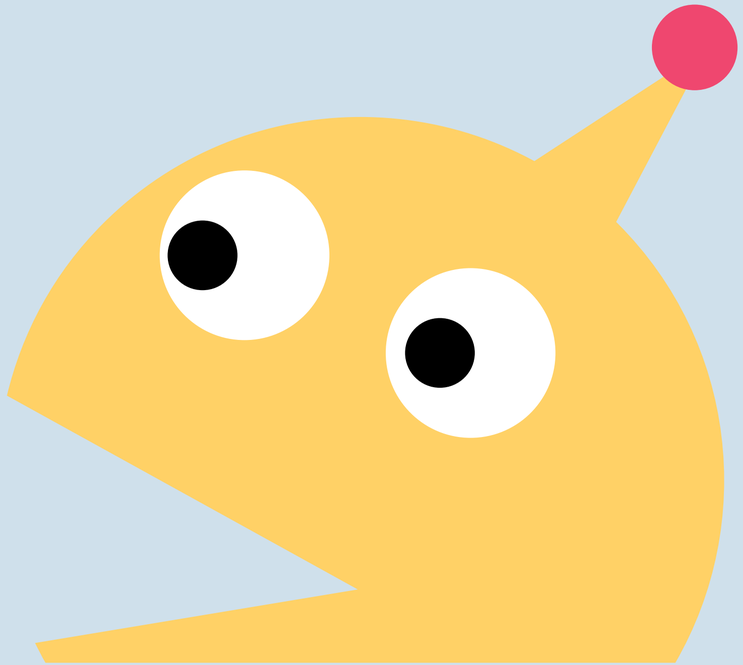
可惜結果不準!

分五段: 0-9, 10-19, 20-29, 30-39, 40+

04 對話機器人

對話機器人

或是自動翻譯



04 對話機器人

客戶說的話 → f → 客服機器人回應

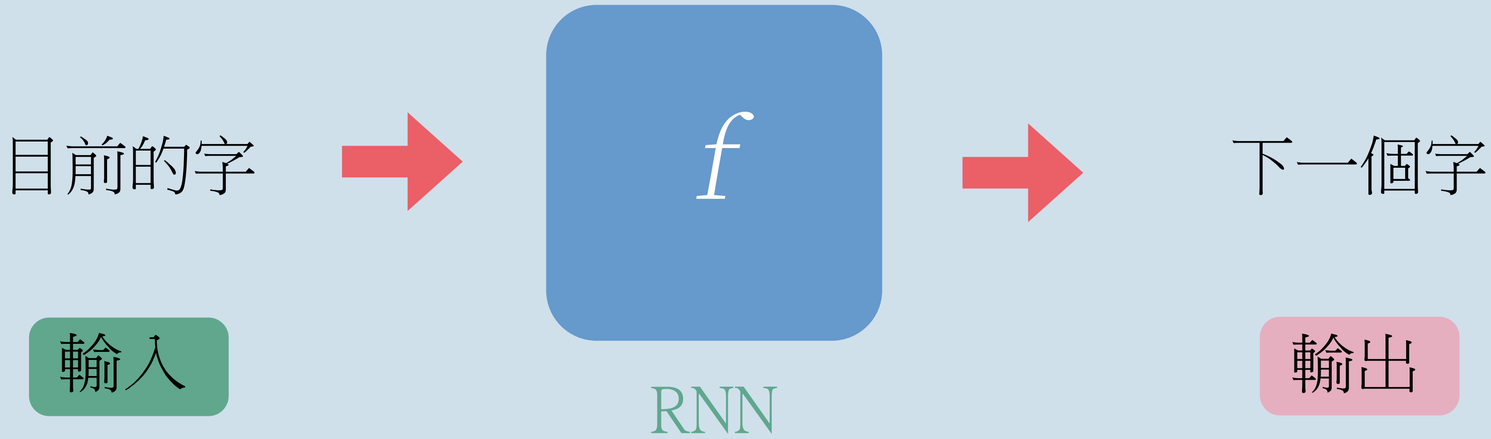
輸入

輸出

注意函數輸入和輸出需要一樣的長度!

04 對話機器人

令人驚呆的是, 最常見的居然是化這這樣的函數...



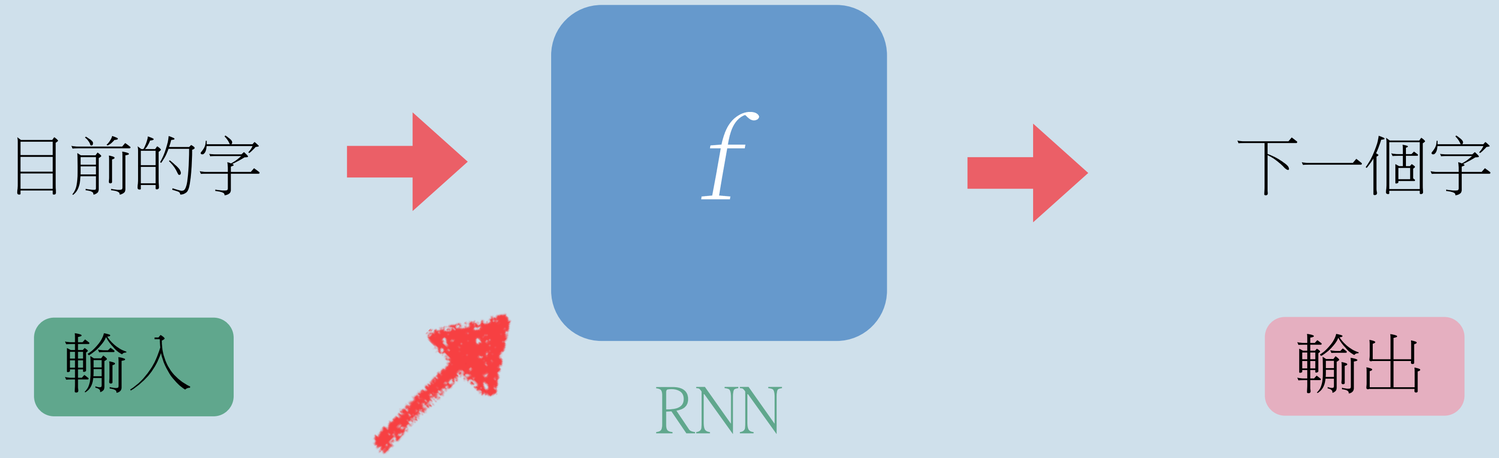
04 對話機器人

比如說, “今天天氣很好。” 這句話, 輸入我們的函數就應該是...

$$
\begin{aligned}
f(\text{“今”}) &= \text{“天”} \\
f(\text{“天”}) &= \text{“天”} \\
f(\text{“天”}) &= \text{“氣”} \\
f(\text{“氣”}) &= \text{“很”} \\
f(\text{“很”}) &= \text{“好”}
\end{aligned}
$$

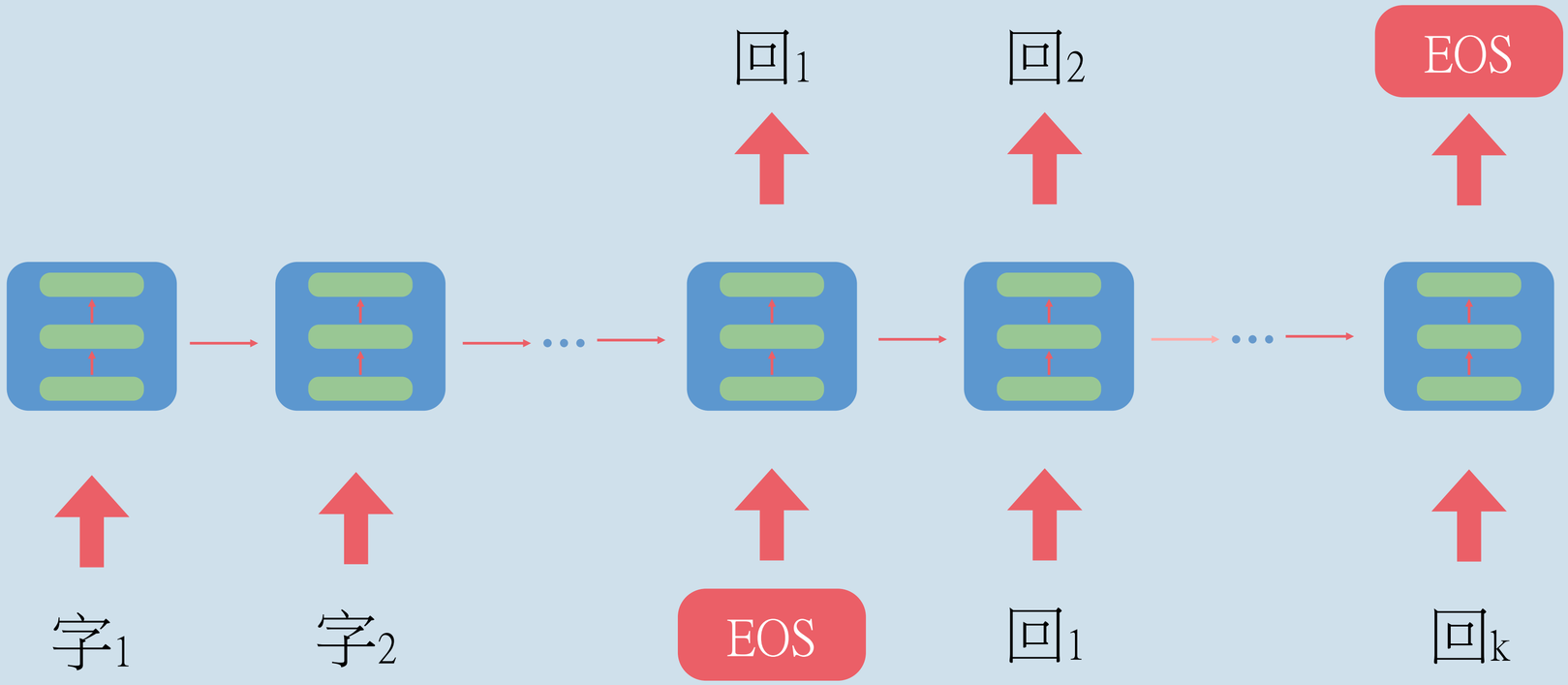
04 對話機器人

結果用有記憶能力的 RNN 做, 它可以是函數的原因是不只輸入一個字, 還有之前的綜合資訊!

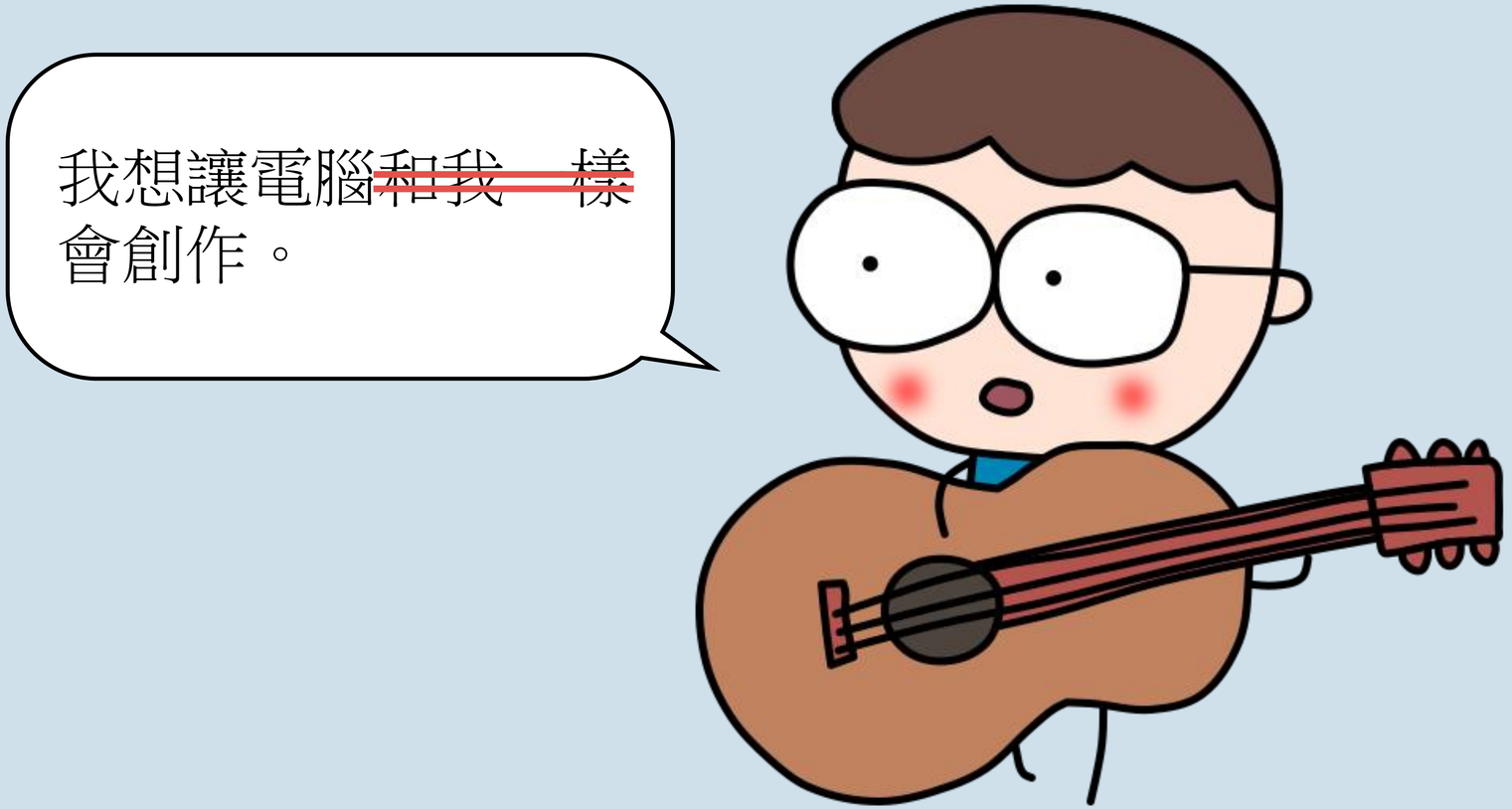


04 對話機器人

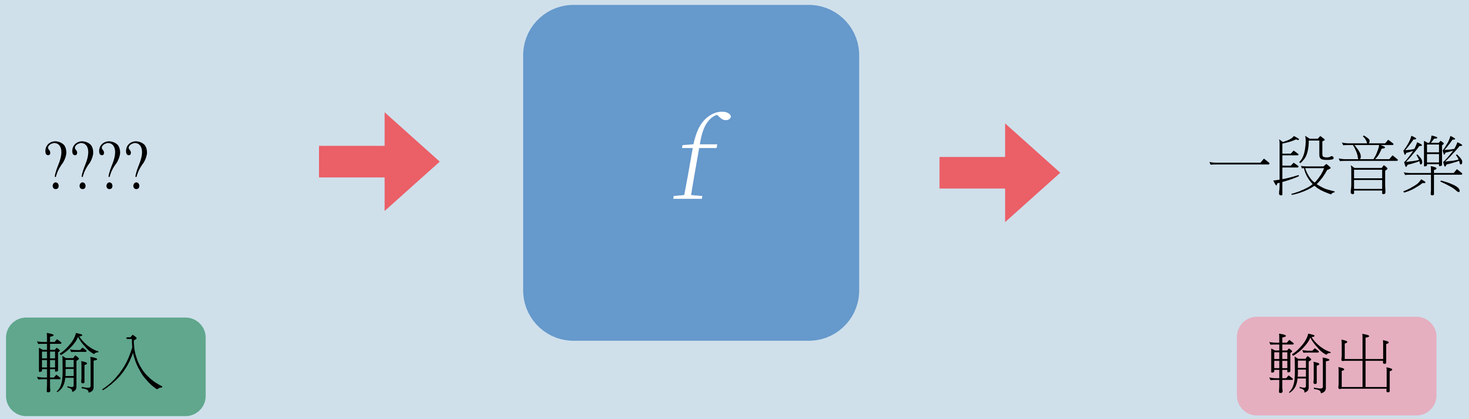
然後就可以做出對話 (翻譯) 機器人!



05 創作機器人



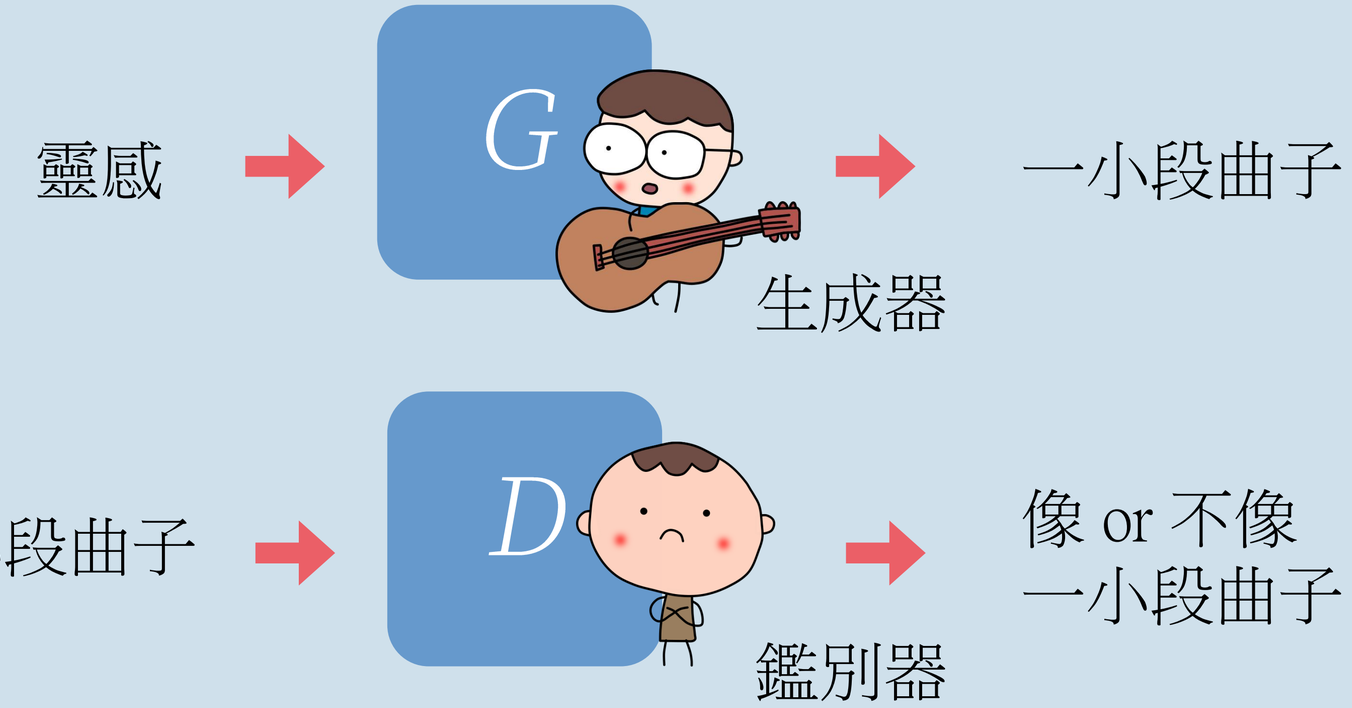
05 創作機器人



到底是要輸入什麼呢?

05 創作機器人

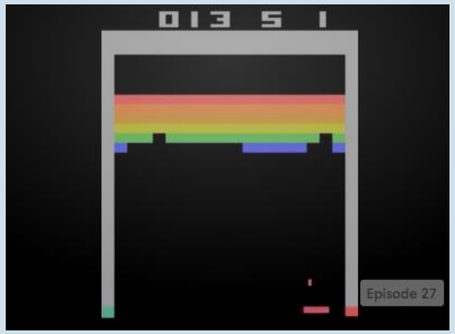
結果是訓練兩個神經網路!



這就是所謂的生成對抗網路 (GAN)!

06「自學型」的 AI

06「自學型」的 AI



輸入

π

最好的動作

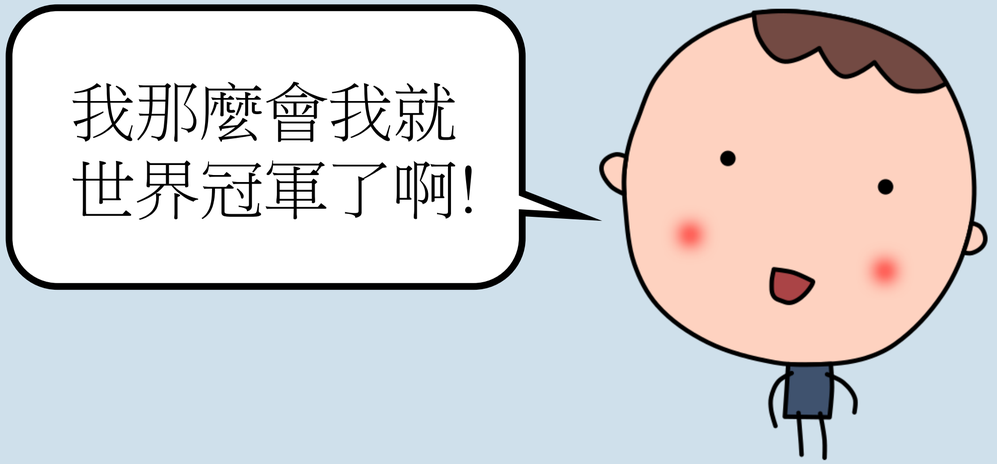
輸出

06「自學型」的 AI

原因是訓練資料很難準備!

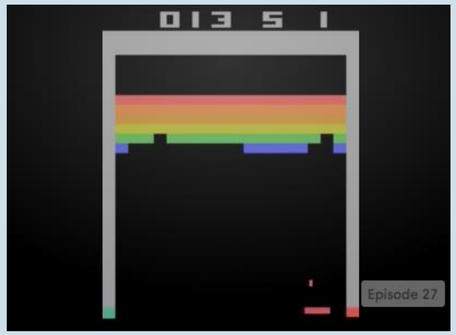
1 變化太多很難各種情境都有訓練資料!

2 以下圍棋為例, 我那麼會的話...



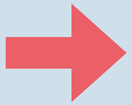
06「自學型」的 AI

直接學哪個動作最好不容易。於是我們學習給每個動作「評分」，通常是計算做了這個動作後「得分的期望值」。



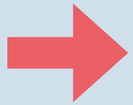
+

輸入



Q

DNN, CNN, RNN



預期的 reward

輸出

這就是強化學習 (reinforcement learning)

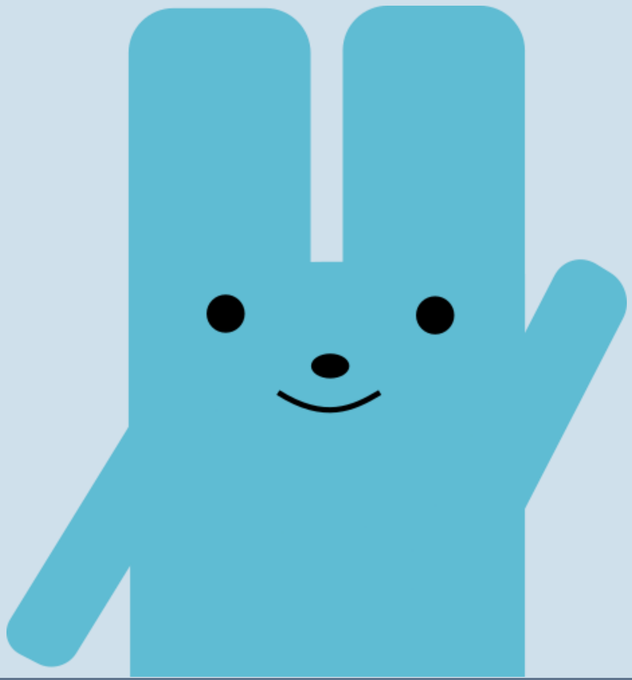
小結論

小結論

(再一次) 重點是一次要非常明確化成函數的形式, 也就是要知確實說清楚輸入、輸出是什麼。

常見問題

身為兔子, 當然很關心...



我想知道怎麼樣紅蘿蔔會長得更快?

滿。心。期。待

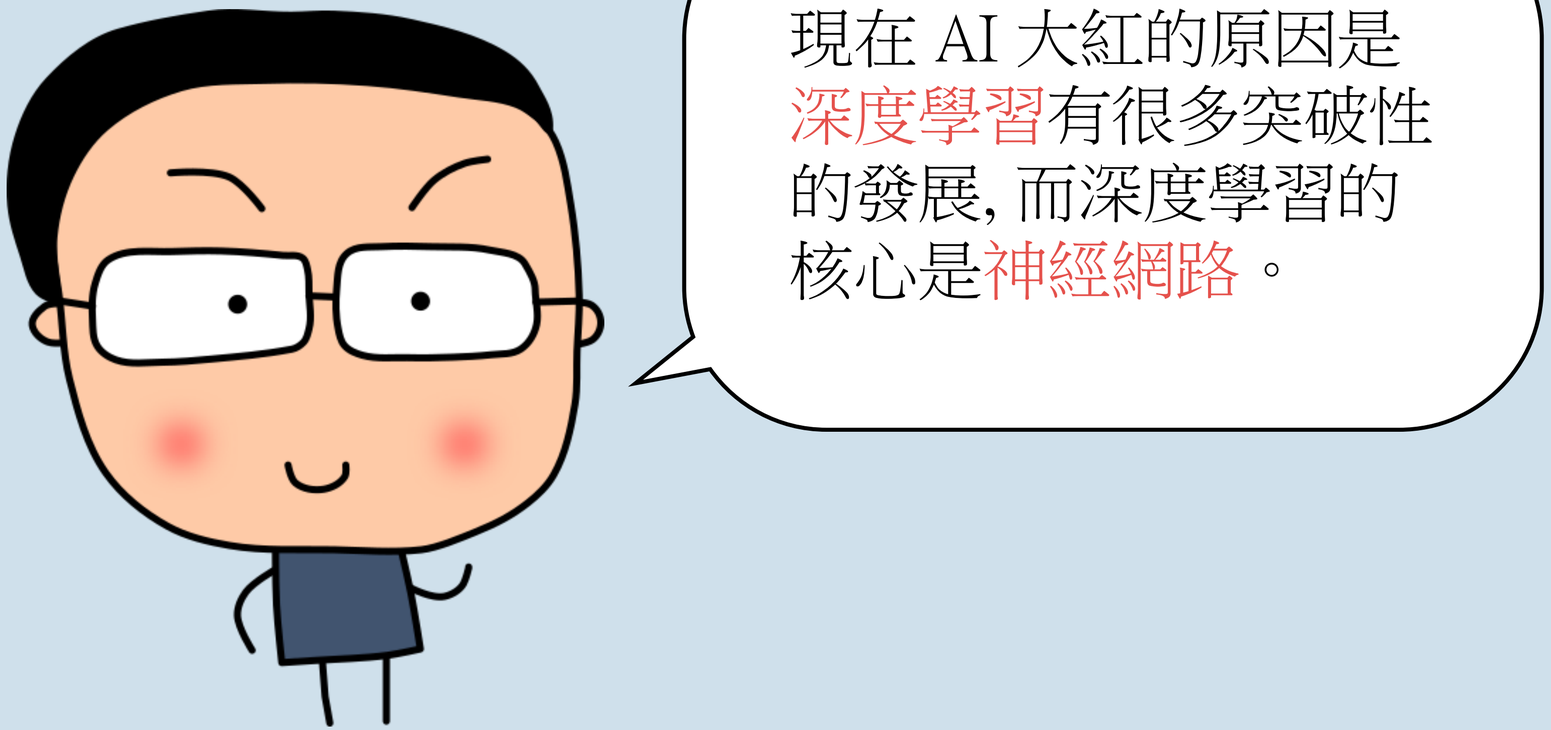
常見問題

不行的原因是沒有把問題化為函數的形勢。大家做 AI 經驗多了，遲早會出現這樣對話的場景...

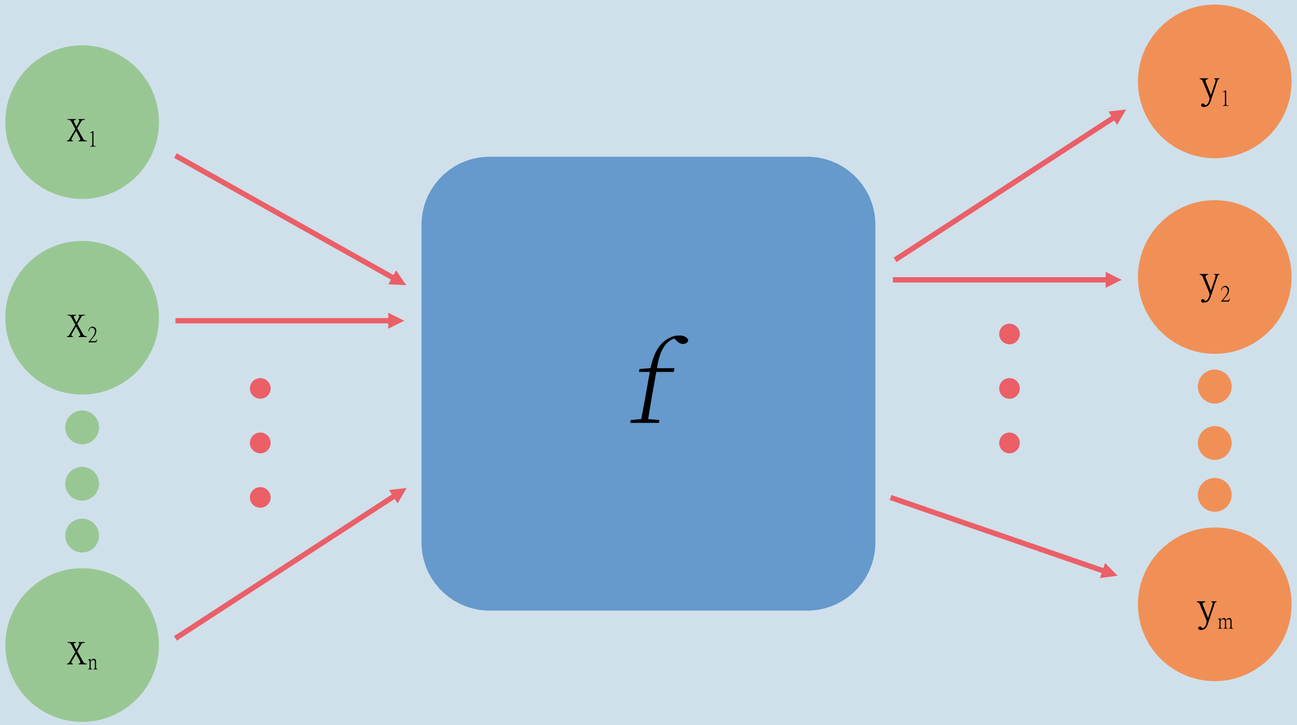
08.

神經網路

深度學習和神經網路

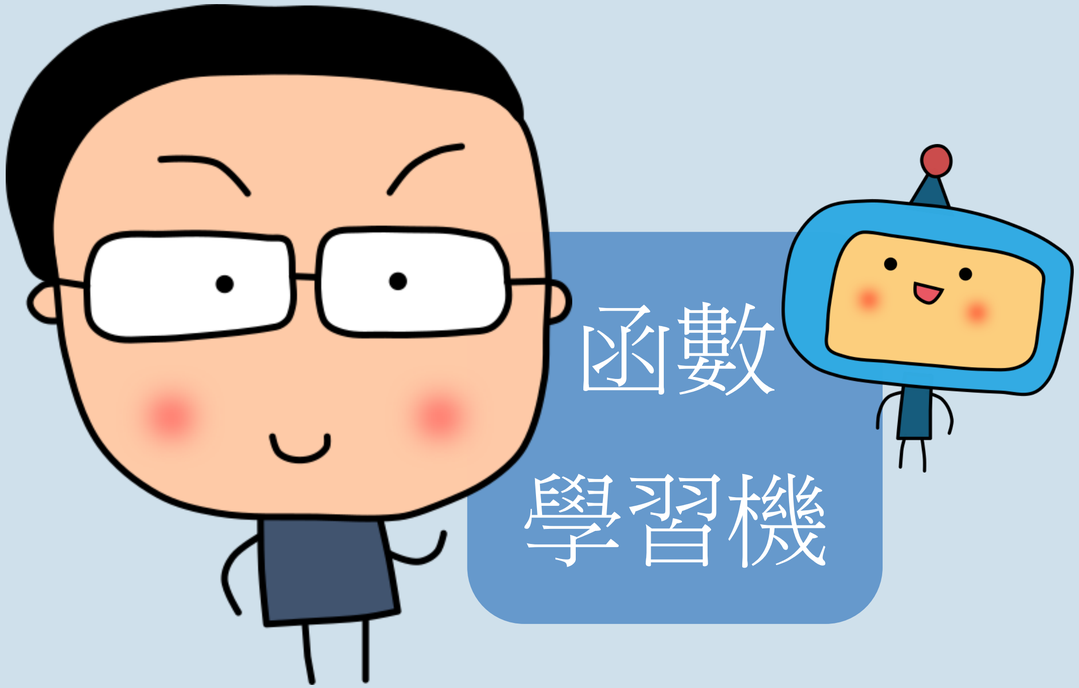


假設我們確定要學哪個函數了



也就是我們輸入和輸出都很明確, 明確到輸入輸出的大小當然也都確定了!

接下來就是要打造函數學習機



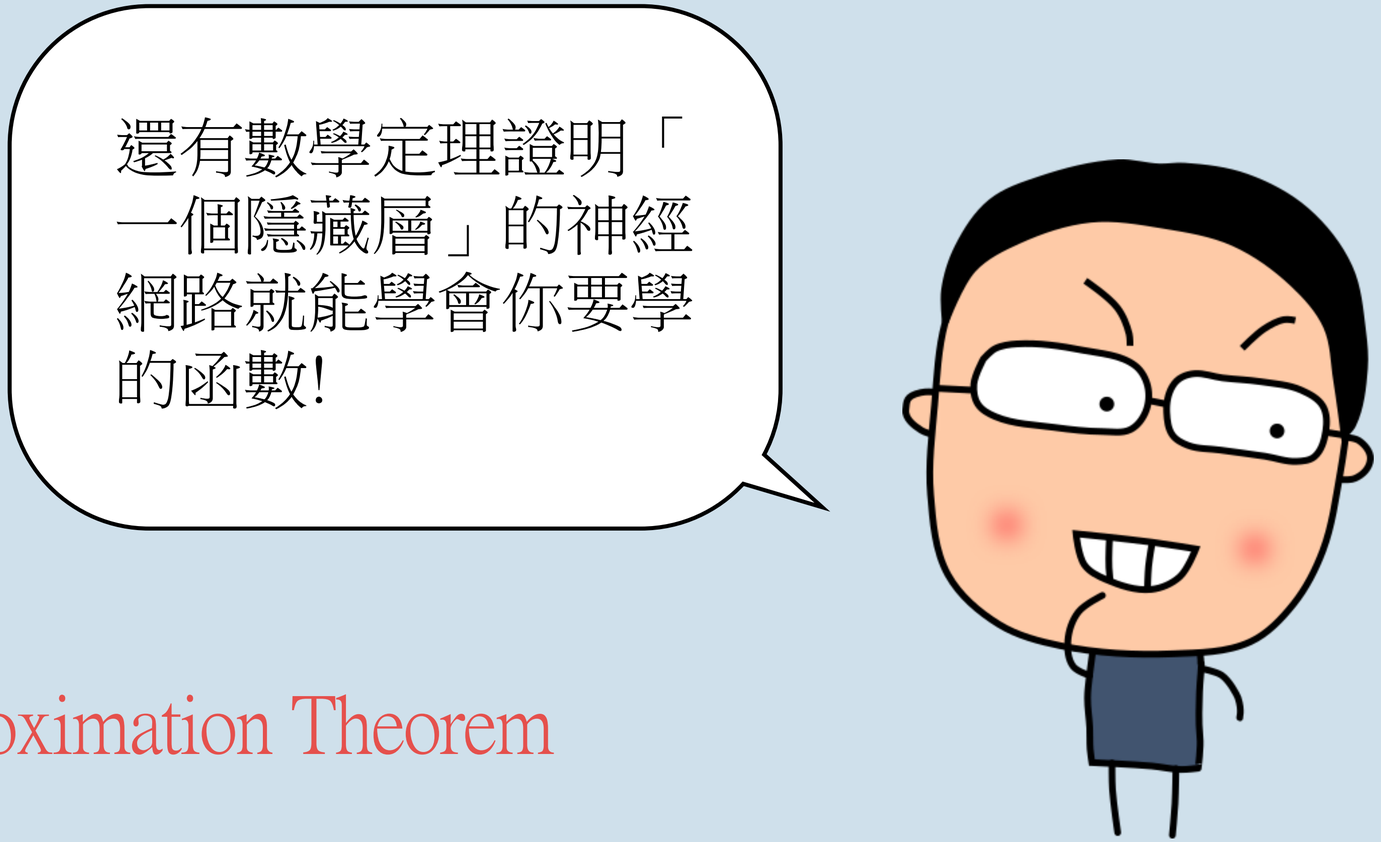
我們要用深度學習 (神經網路) 的方法, 打造一台函數學習機!

神經網路暗黑學習法!

這個世界有個近乎全能
的暗黑函數學習法

神經網路

神經網路暗黑學習法!

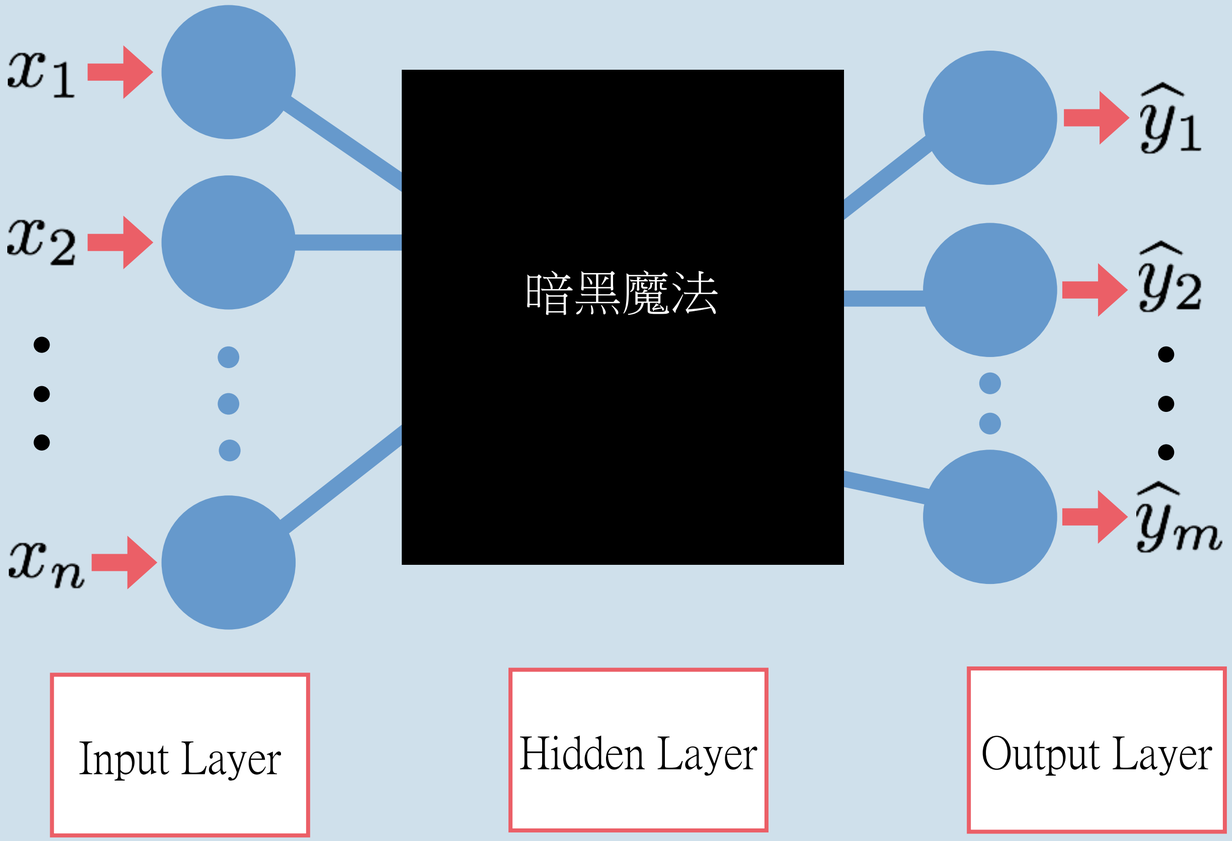


Universal Approximation Theorem

神經網路暗黑學習法!

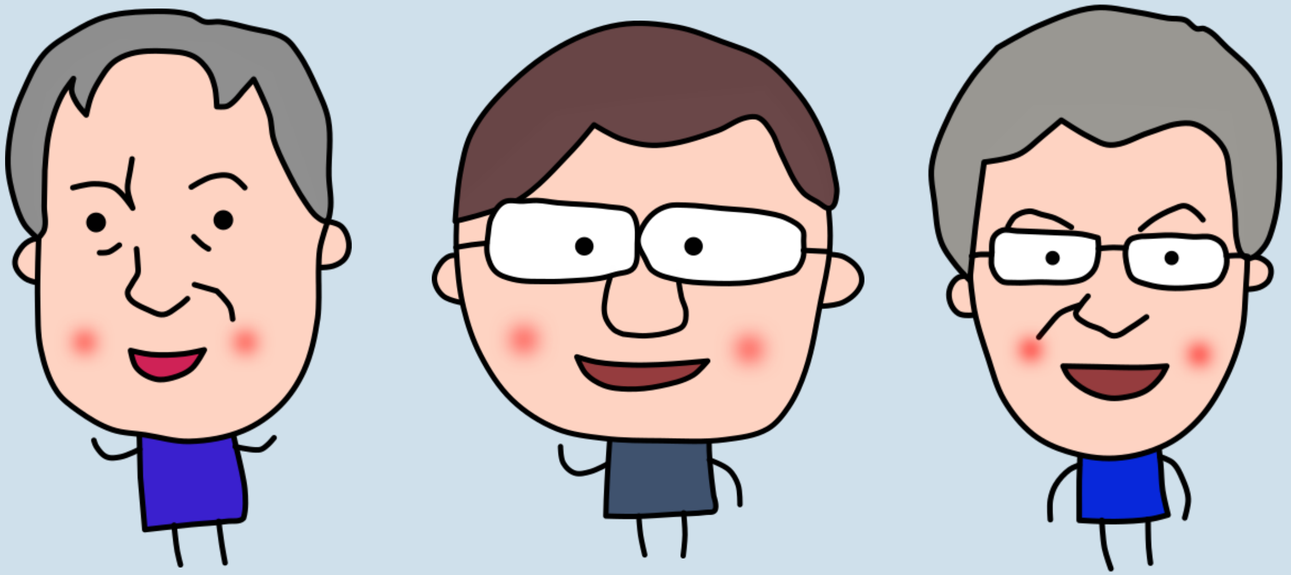
而且我們不用知道函數長什麼樣子 (線性啦、多項式樣啦等等)!

神經網路暗黑學習法!



如同魔法般的, 你問好問題 、準備好資料, 神經網路就告訴你函數應該長什麼樣子!

神經網路暗黑學習法!



在 1980-1990 年代可以說紅極一時!

魔法光芒不再!?

然後它就死掉了...

有一陣子如果研究計畫說要研究「神經網路」, 那計畫還沒送出就可以確定不會通過!

為什麼上個世紀神經網路沒有預期成功?

複雜的軟體

電腦計算能力

大量的數據

Yann LeCun

到了現在, 情況完全不一樣了!

神經網路新時代!

2015-2-26

DeepMind

“Human-level control through deep reinforcement learning”

letter

Deep Q-Learning

原來電腦可以自已學會做一些複雜的任務!

神經網路新時代!

2015-5-28

LeCun-Bengio-Hinton

“Deep Learning”

延申閱讀

介紹 Deep Learning「三巨頭」的故事。

http://bit.ly/ai_history

度過神經網路寒冬的三巨頭, 一起宣告 deep learning 時代的來臨!

神經網路新時代!

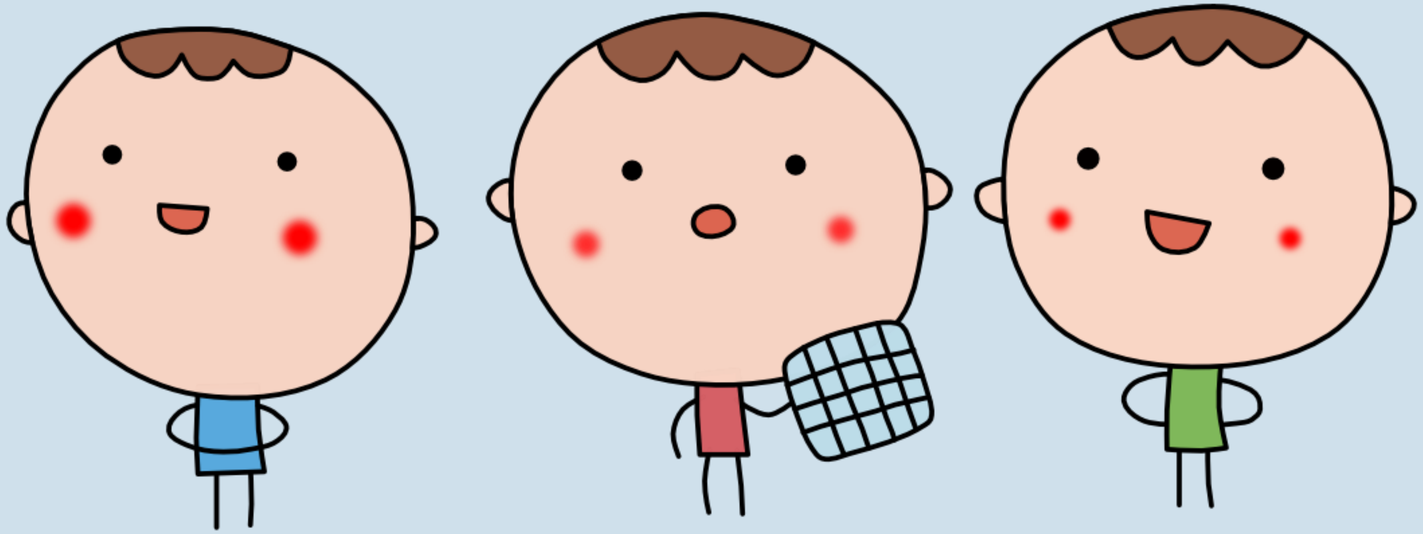
2016-2-26

DeepMind

“Mastering the game of Go with deep neural networks and tree search”

Deep Learning 打敗世界棋王, 讓大家大大期待 deep learning 的能力!

再一次, 就這三大天王改變了全世界!

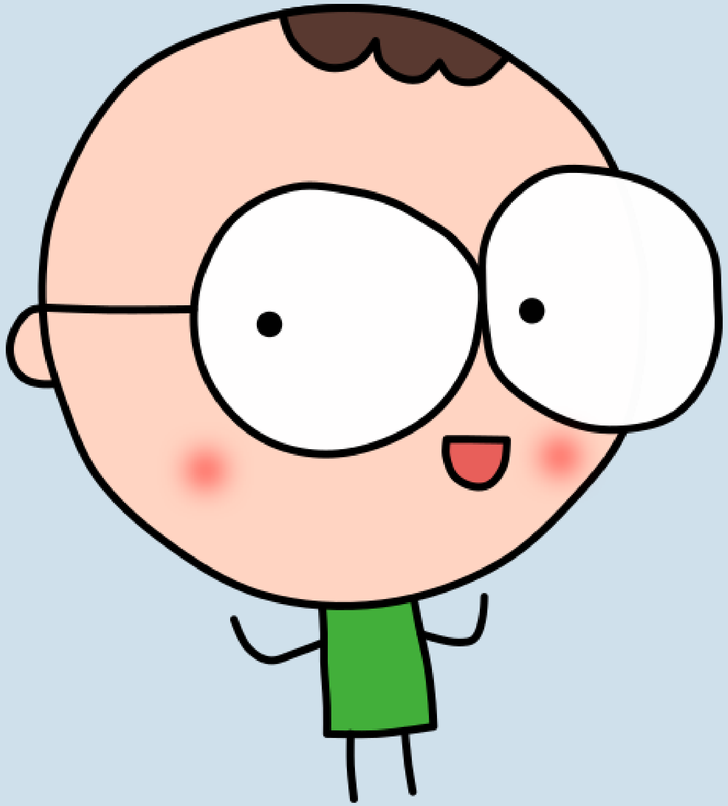


我們會來一一介紹這些神經網路架構有什麼特點。

09.

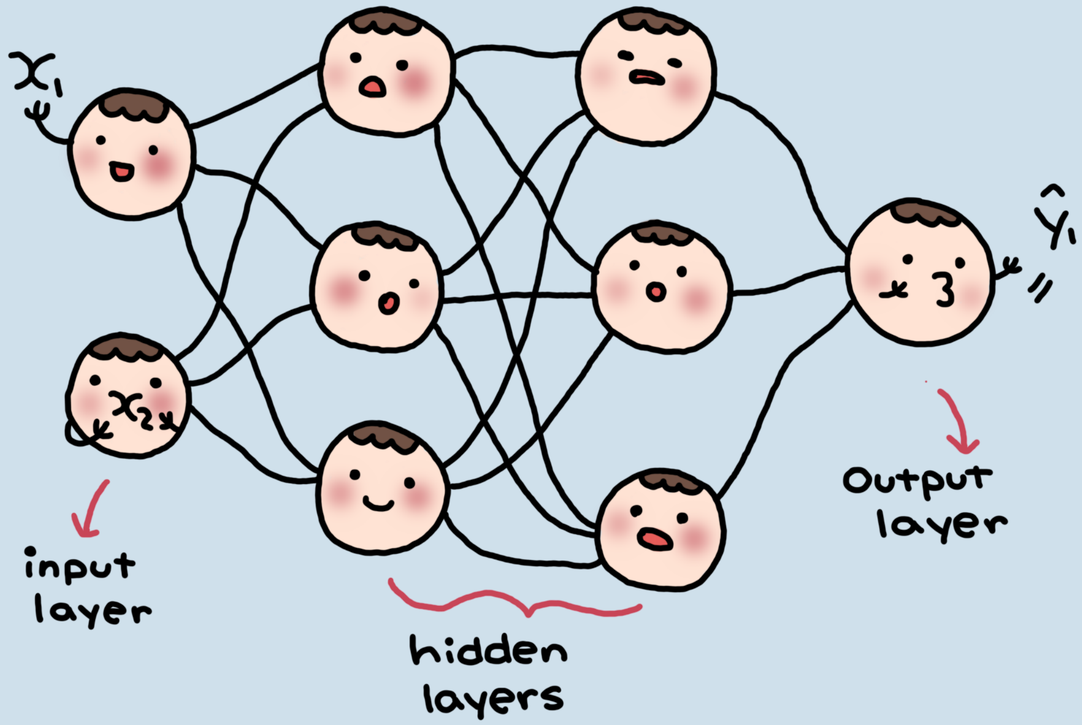
神經網路標準版 DNN

全連結神經網路 (DNN)



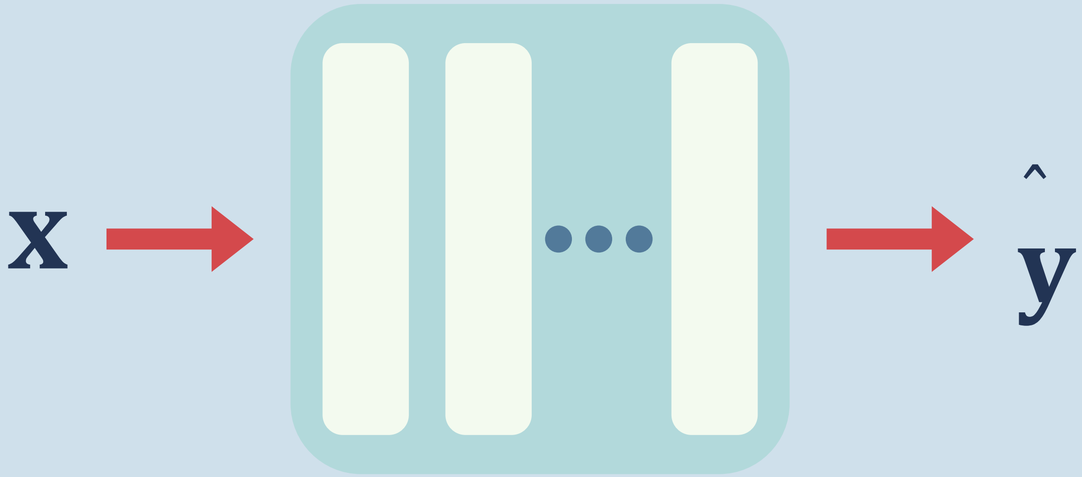
- Fully Connected Neural Networks
- 1980 年代就火紅的 model

全連結神經網路 (DNN)



- 只需要決定兩組數字, 神經網路就決定了!
  1. 幾層隱藏層
  2. 每層幾個神經元
- 層層之間的神經元是完全連結的。

深度學習就是建一層層「隱藏層」



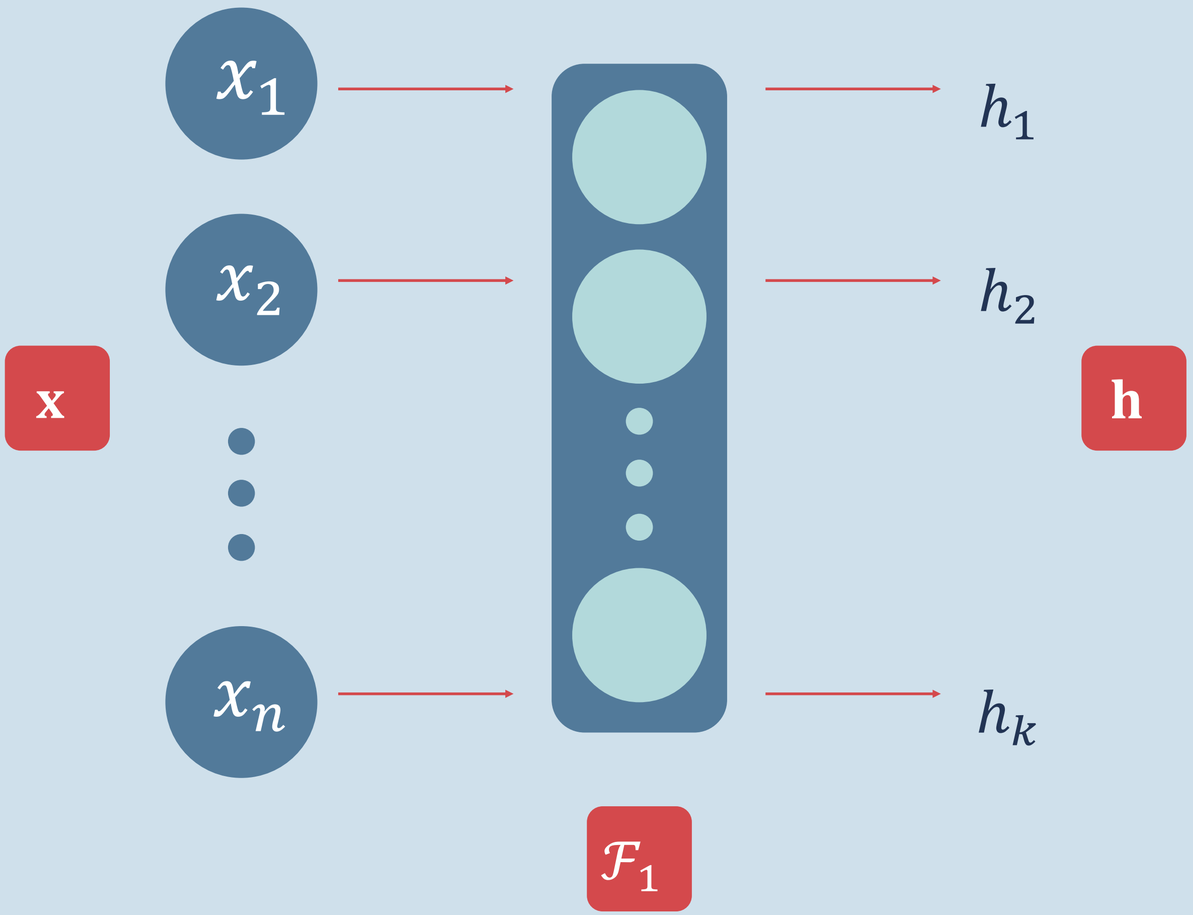
input layer

hidden layers

output layer

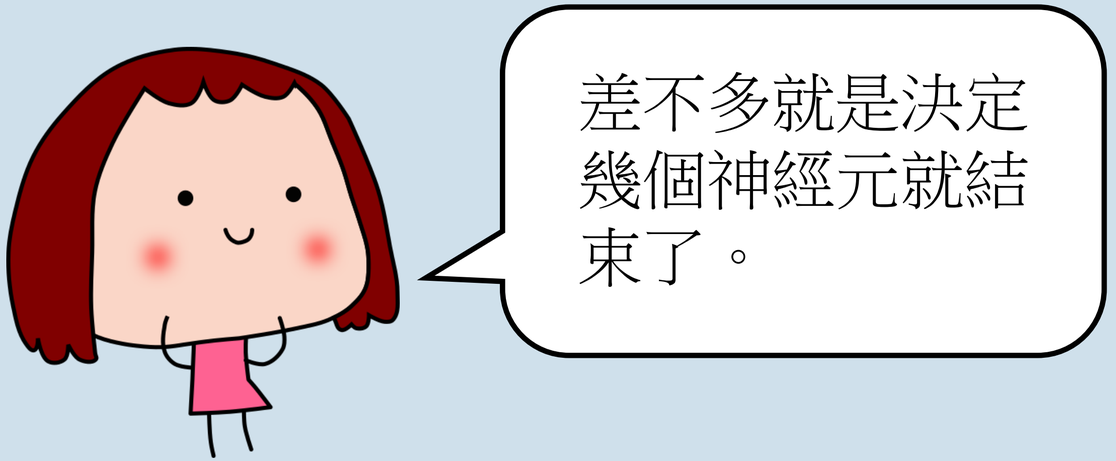
DNN, CNN, RNN 嚴格說是每一層都可以決定是要用三種之中，哪一種方式做隱藏層的設計。

深度學習就是建一層層「隱藏層」



隱藏層基本上就三種選擇:

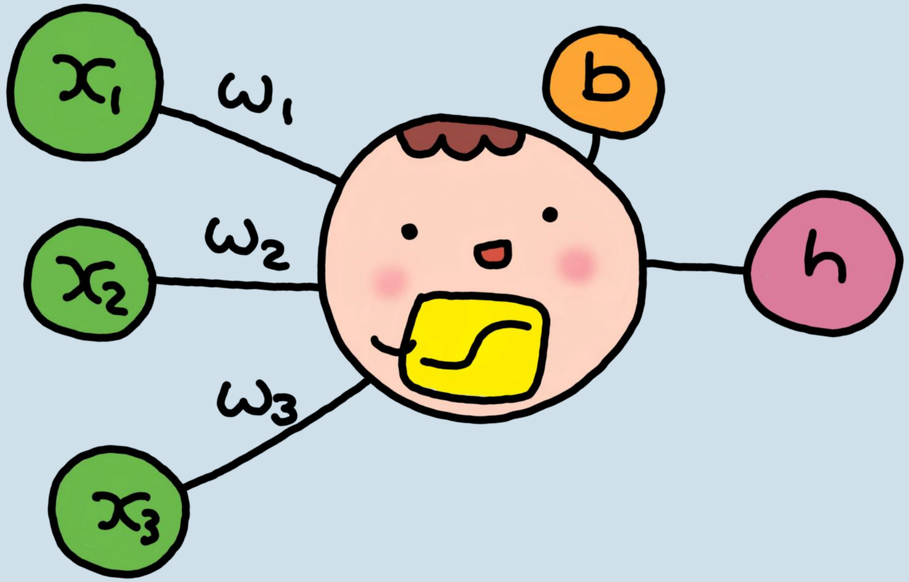
- 全連結層 (Dense) DNN
- 卷積層 (Conv) CNN
- 遞歸層 (LSTM, GRU) RNN



神經元怎麼運作

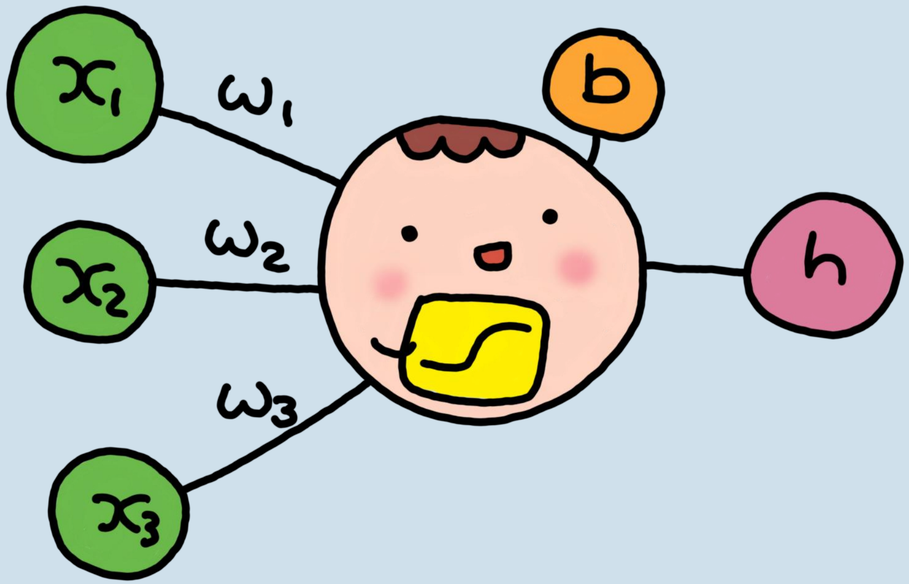
不管哪種神經網路, 神經元就是基本的運算單元, 而每個神經元的運算方式是一樣的!

神經元怎麼運作



每個神經元就是接受若干個刺激輸入, 然後送出一個刺激輸出。

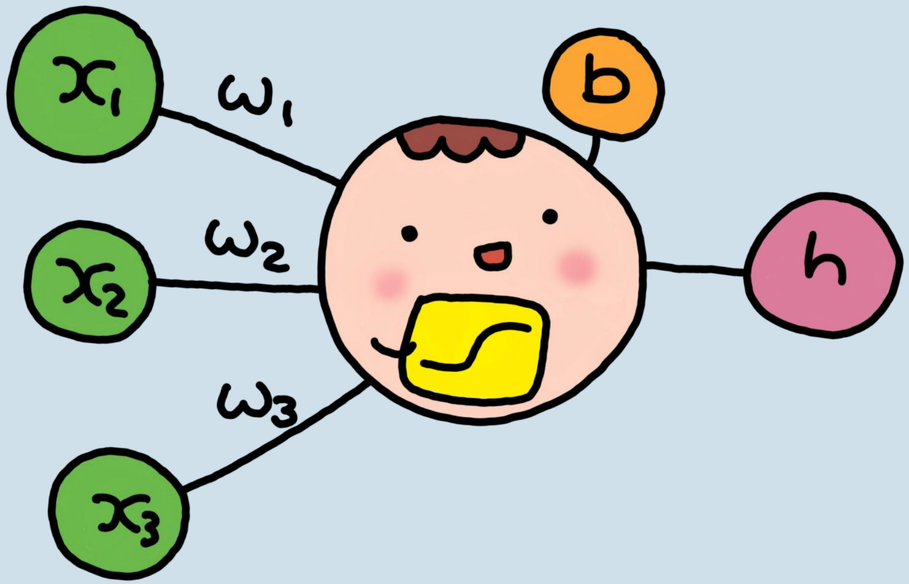
神經元怎麼運作



我們先要這個神經元接受到的總刺激。

$$\sum_{i=1}^{3} w_i x_i$$

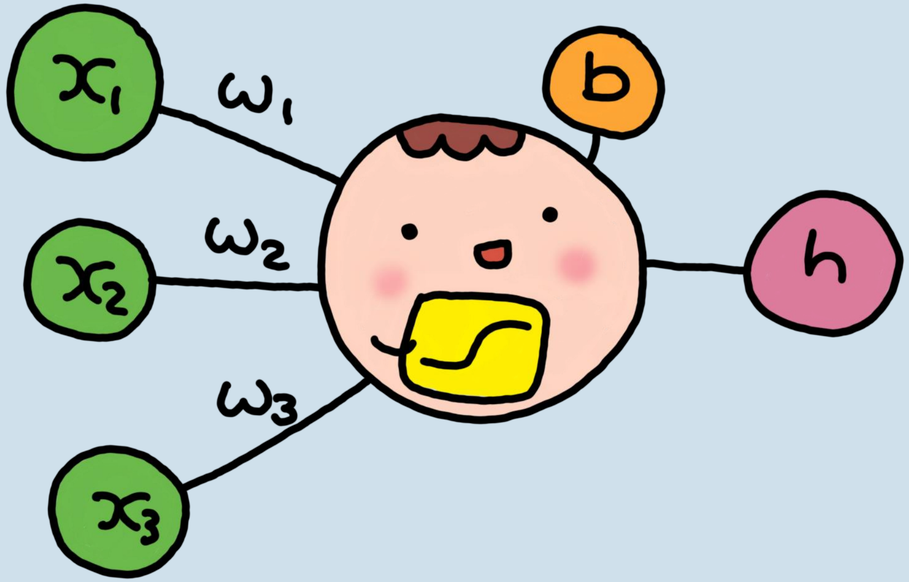
神經元怎麼運作



接著加上偏值 (bias), 做一個基準的調整。不管是權重、偏值都是要經過學習得到的! 這裡我們估且稱調整後的總刺激。

$$\sum_{i=1}^{3} w_i x_i + b$$

神經元怎麼運作

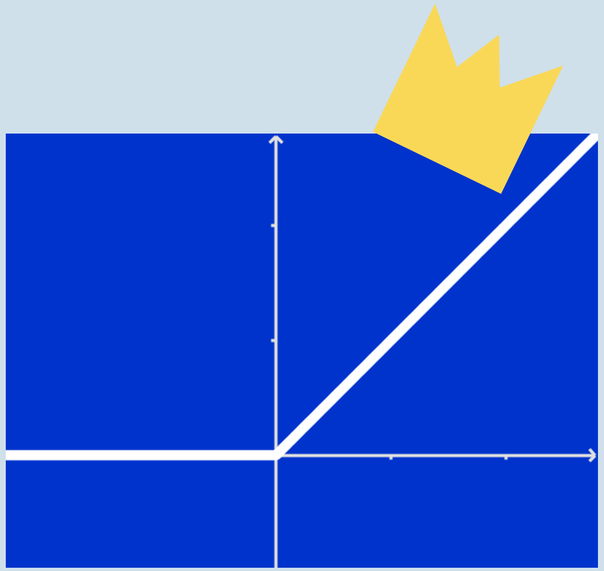


目前的計算都是線性的, 即使經所有神經元算還是線性的, 因此我們需經一個激發函數 (activation function) 轉換再送出。

$$\varphi\left(\sum_{i=1}^{3} w_i x_i + b\right) = h$$

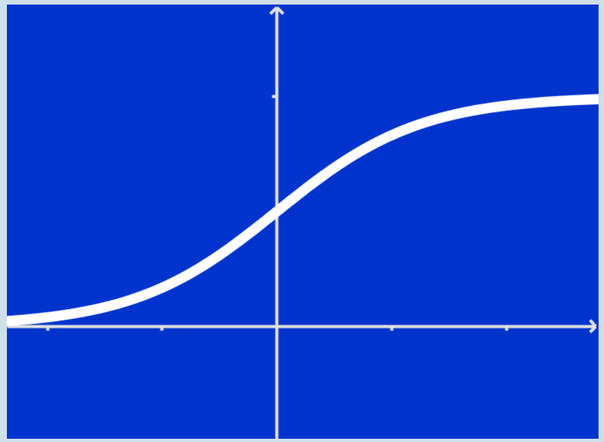
激發函數

以下我們介紹幾個有名的激發函數。



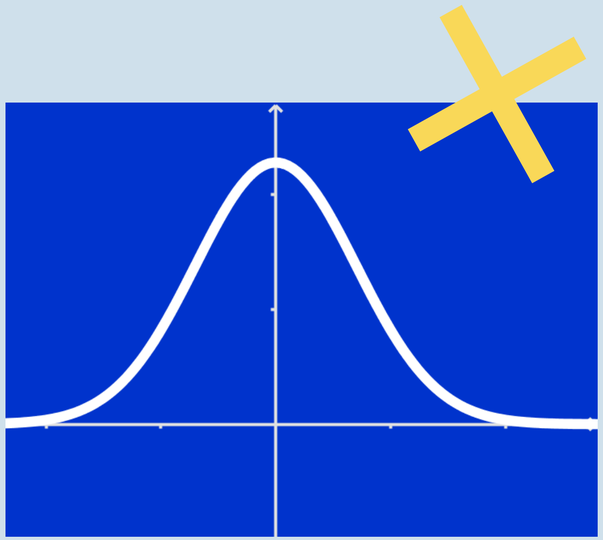
ReLU

本世紀新寵!



Sigmoid

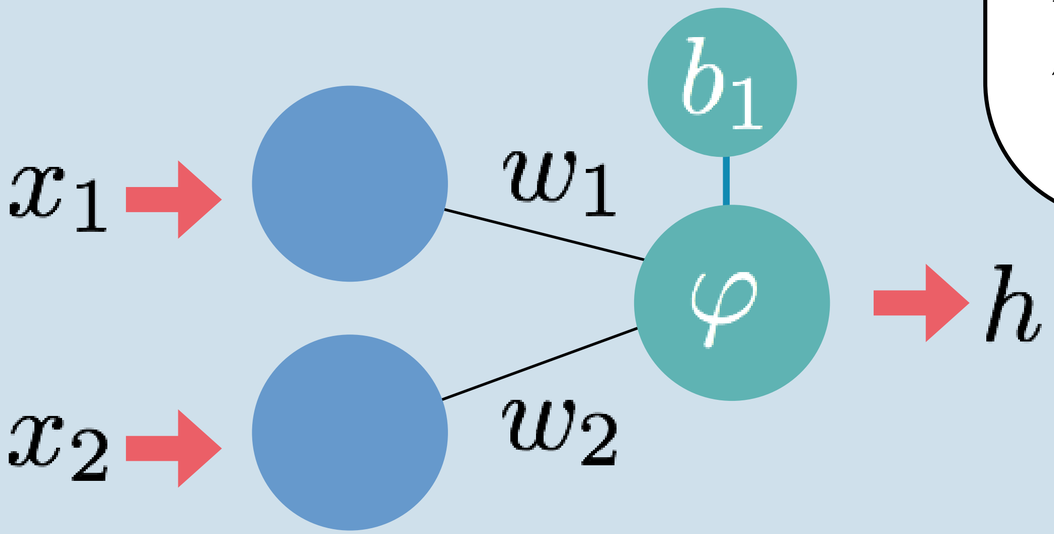
感覺應該很接近人類神經元的動作。



Gaussian

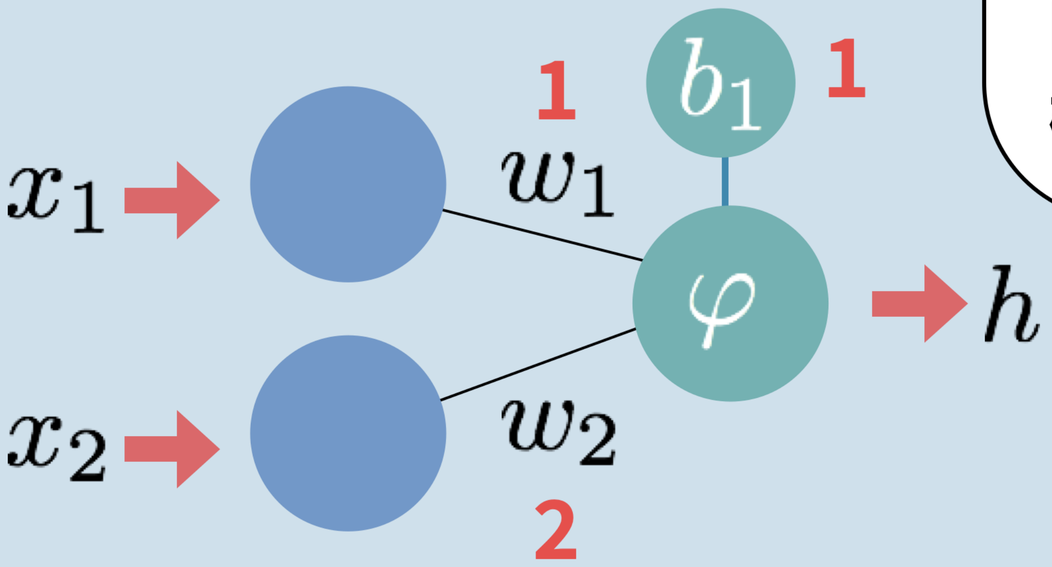
舊時代的懷念, 已很少使用。

把自己當一個神經元!



現在把自己當成一個神經元, 用 ReLU 做我們的激發函數。

把自己當一個神經元!



權重和偏值都是學來的, 但一開始我們會隨機給一個值。

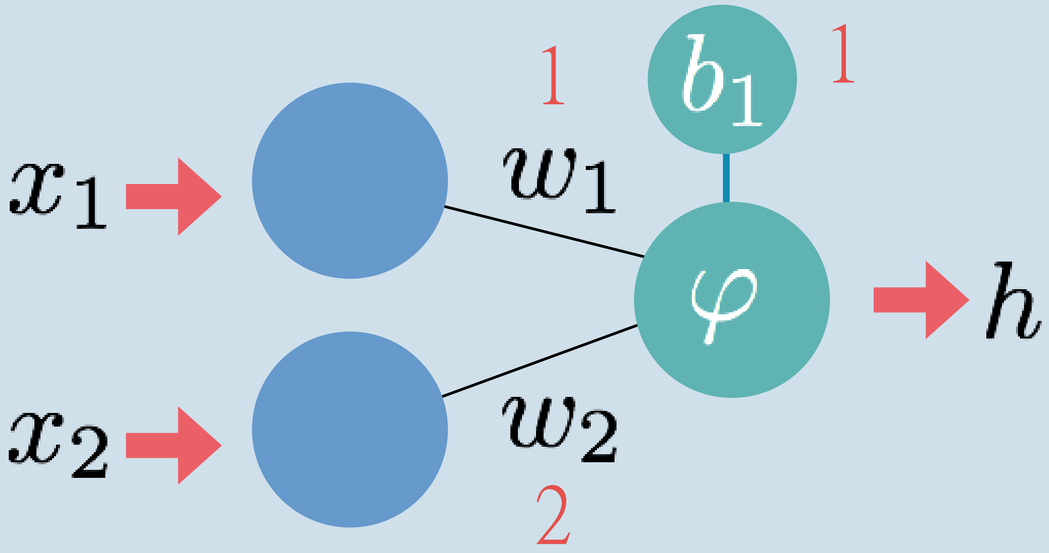
假設初始化得到:

$$w_1 = 1$$

$$w_2 = 2$$

$$b_1 = 1$$

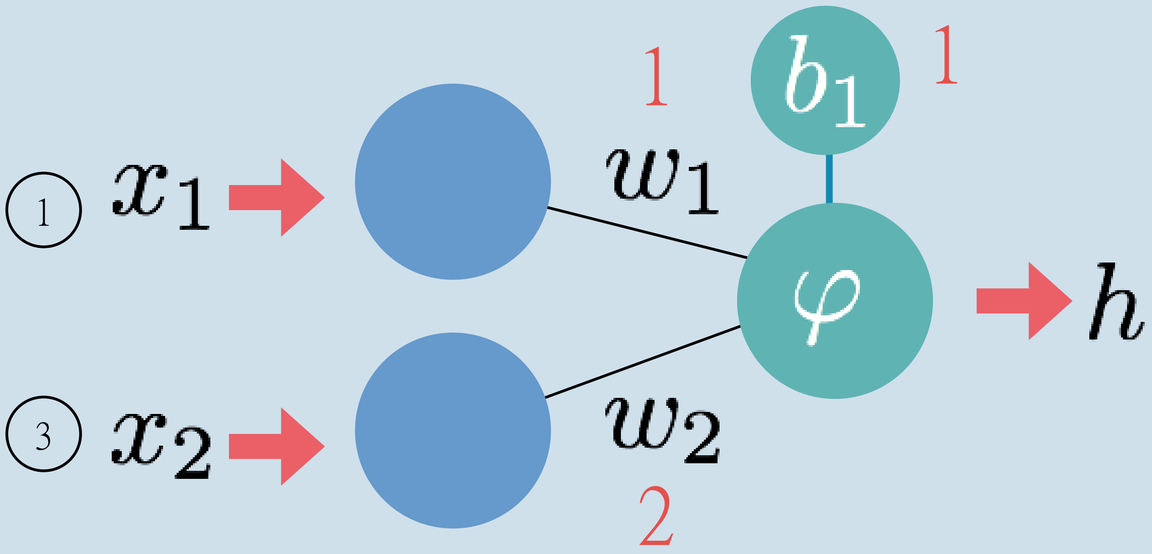
把自己當一個神經元!



記得我們神經元的計算方式。現在有任何輸入我們都應該知道該怎麼輸出了。

$$\varphi(\underset{1}{w_1}x_1 + \underset{2}{w_2}x_2 + \underset{1}{b_1}) = h$$

把自己當一個神經元!



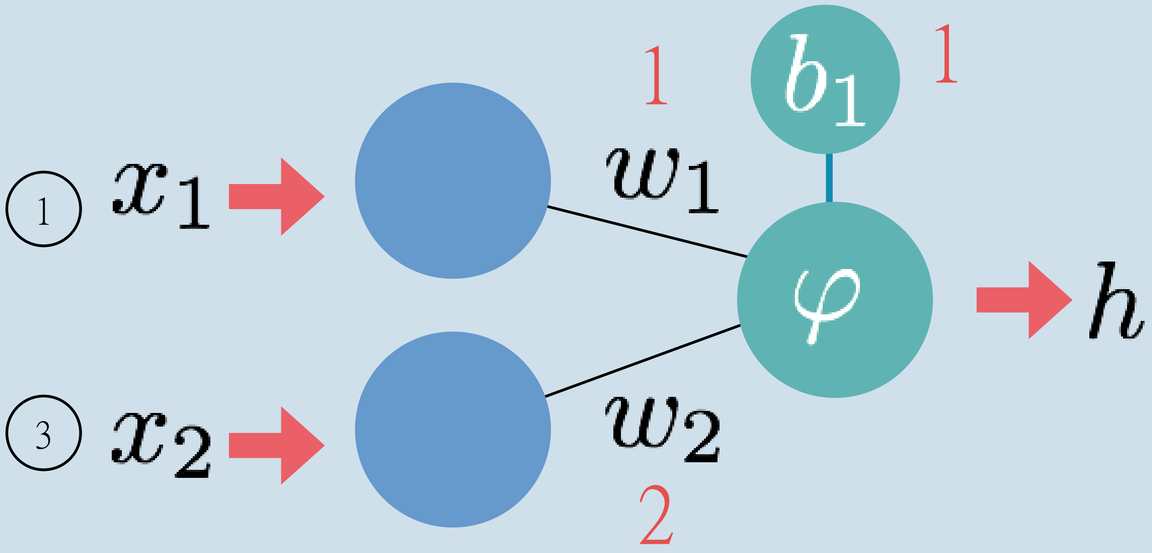
比如說我們接到

$$\mathbf{x} = (x_1, x_2) = (1, 3)$$

的輸入, 輸出應該是多少呢?

$$\varphi(\underset{1}{w_1}x_1 + \underset{2}{w_2}x_2 + \underset{1}{b_1}) = h$$

把自己當一個神經元!

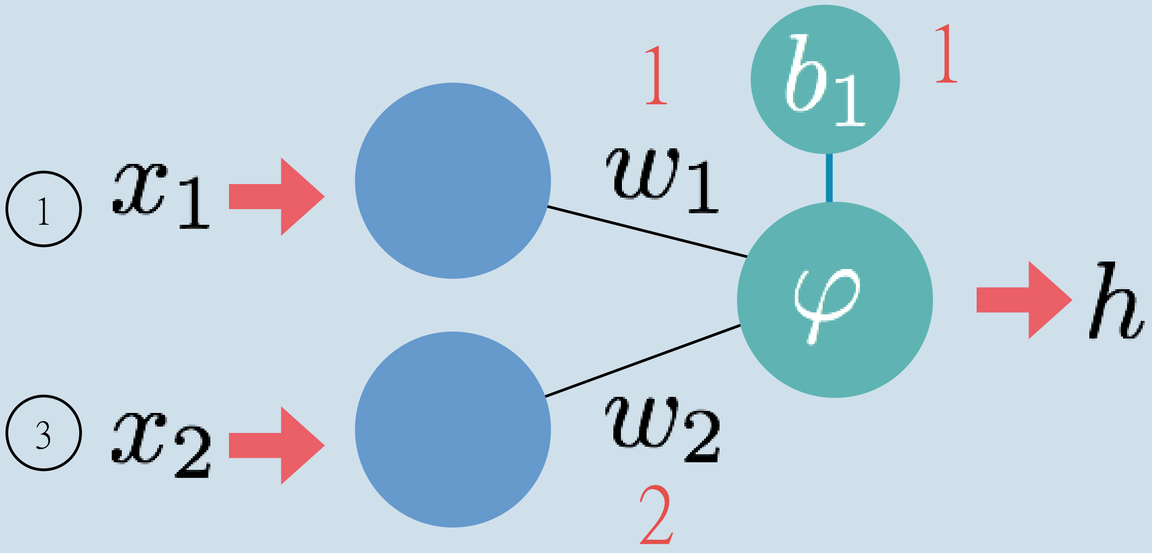


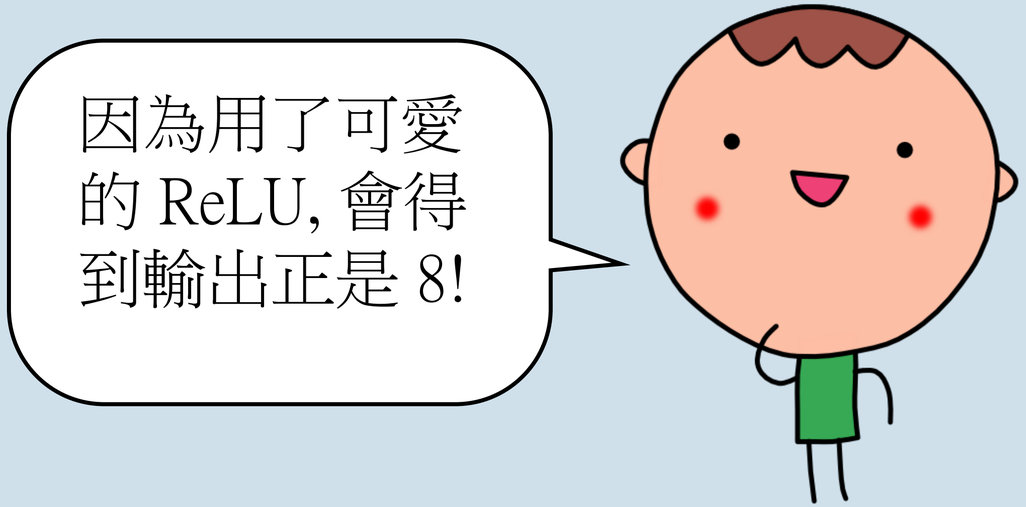
$$\varphi(w_1 x_1 + w_2 x_2 + b_1) = h$$

1 × ① 2 × ③ 1

8

把自己當一個神經元!



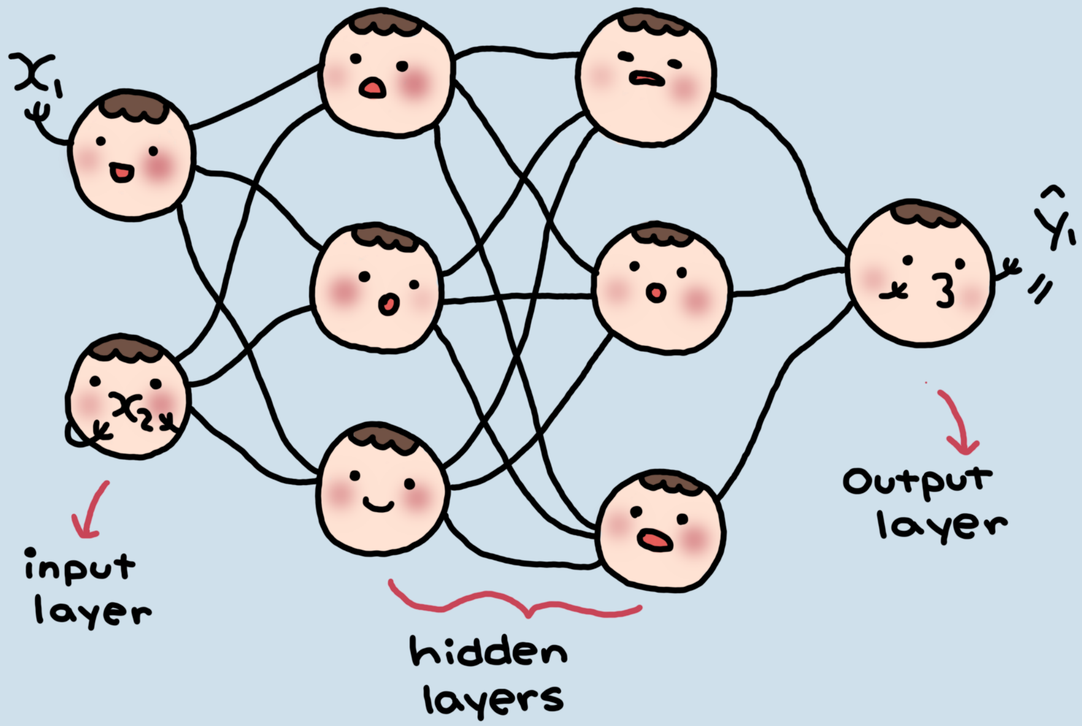


$$\varphi(w_1 x_1 + w_2 x_2 + b_1) = h$$

8

$$\varphi(8) = 8$$

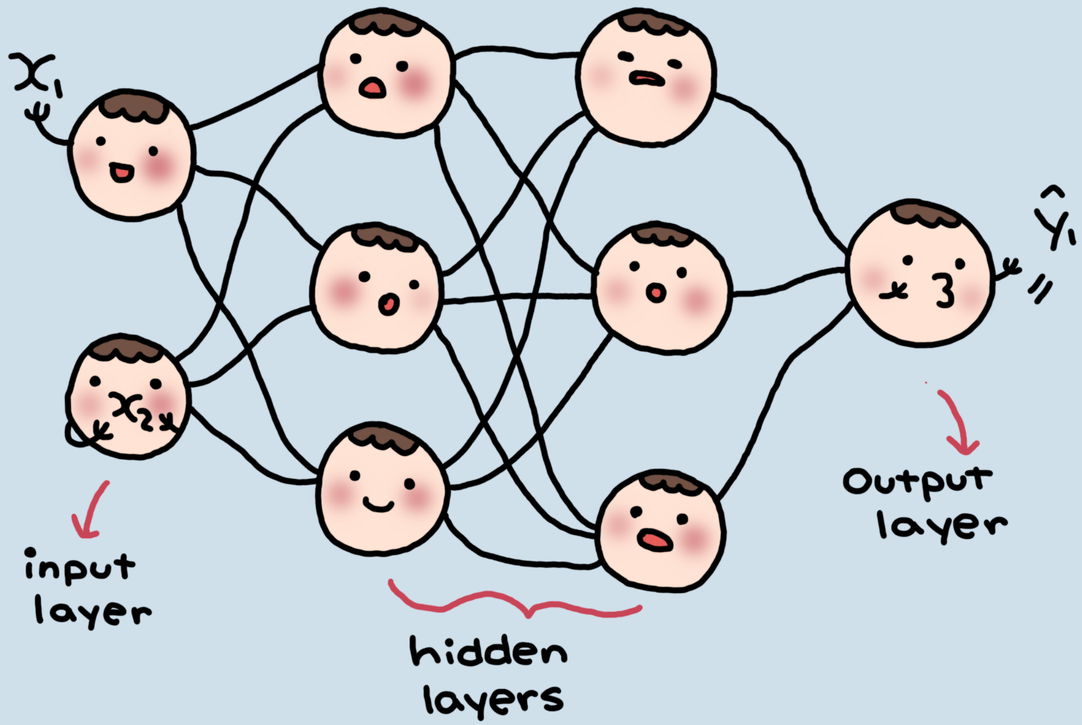
函數學習機建造完成!



我們會把所有要學習的參數, 包括所有的權重、所有的偏值, 合併起來叫做:

$$\theta$$

函數學習機建造完成!



當決定了權重、參數, 也就是一組 θ 的值, 我們輸入任意的數值, 神經網路就會吐出一個輸出給我們了!

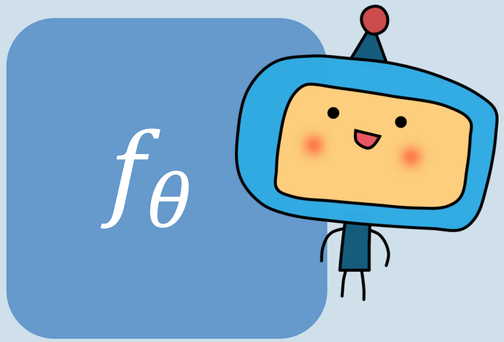
10.

神經網路怎麼學的

訓練我們的神經網路

神經網路需要經過訓練! 方式是把我們的「考古題」(訓練資料)一次次拿給神經網路學。學習法叫 backpropagation。

函數空間

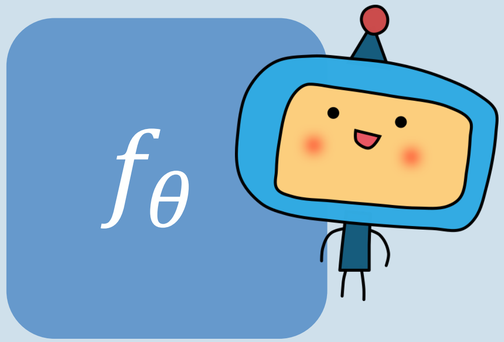


當我們用神經網路的方式, 打造了一台函數學習機之後, 決定一組參數 θ (包括權重、偏值) 就會決定一個函數。

因為我們的函數學習機可以生出無限多個函數, 我們把所有可能生出的函數收集起來, 成為一個函數空間。

$$\{f_\theta\}$$

函數空間



$$\theta^*$$

我們的目標是要挑出一組最好的 θ^*，意思就是這樣做出來的 f_{θ^*} 和目標函數最接近。

Loss Function

我們會用一個叫 loss function 的, 計算訓練資料中, 我們的神經網路輸出和正確答案有多大的差距。

自然我們希望 loss function 的值是越小越好。

Loss Function

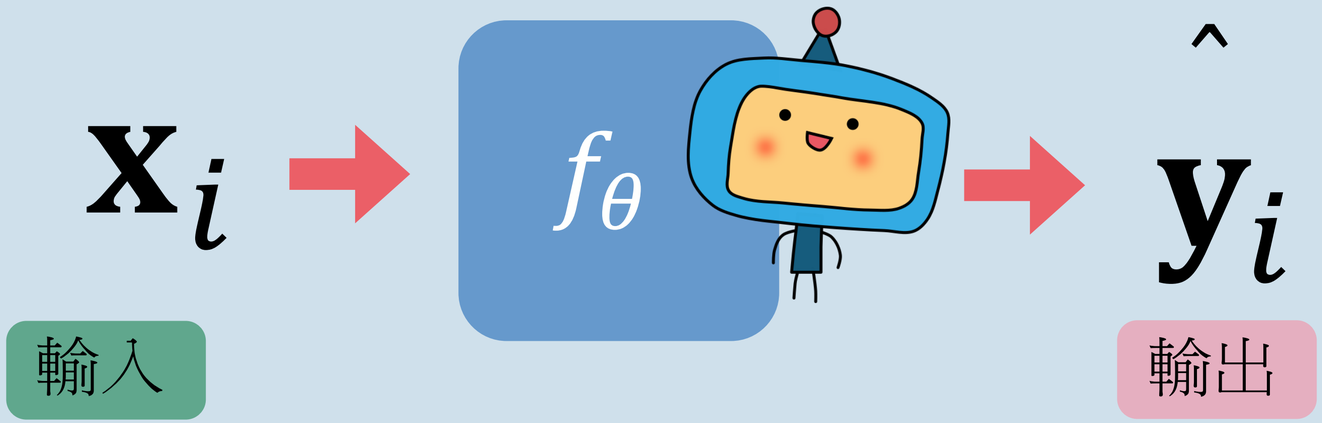
假設我們有訓練資料

$$\{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), \cdots, (\mathbf{x_k}, \mathbf{y_k})\}$$

意思是輸入 $\mathbf{x_i}$, 正確答案應該是 $\mathbf{y}_i$。

Loss Function

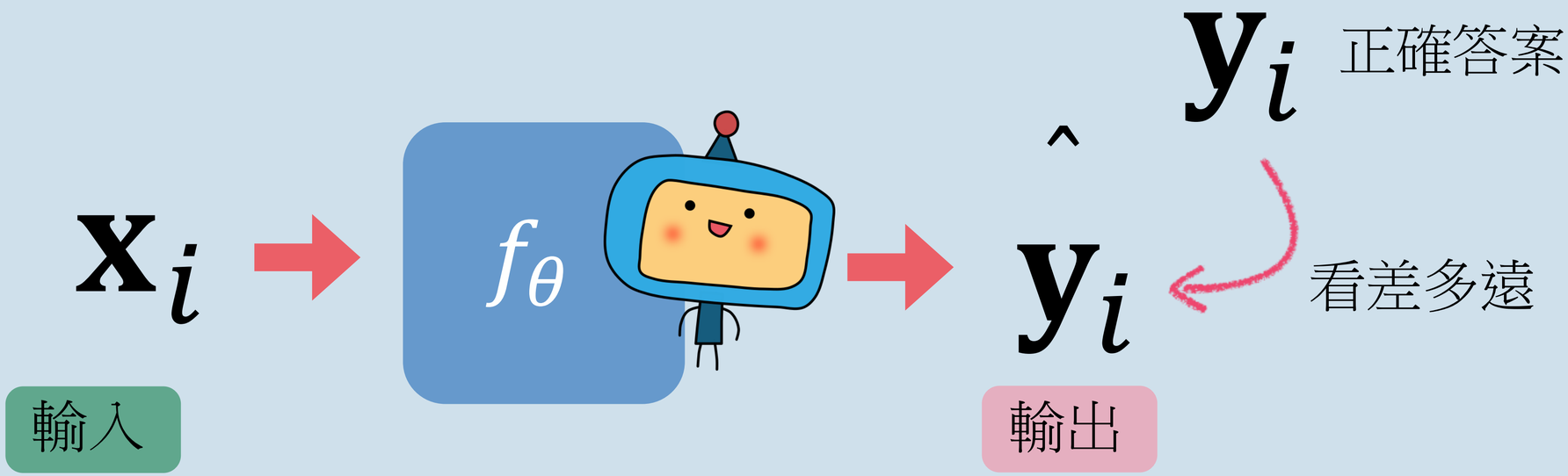
任何一組參數 θ, 就會給定一個函數 f_θ。這個函數 f_θ 對任意的輸入 $\mathbf{x}_i$ 都會給出一個值 $\hat{\mathbf{y}}_i$。



$$f_\theta(\mathbf{x_i}) = \hat{\mathbf{y}}_\mathbf{i}$$

Loss Function

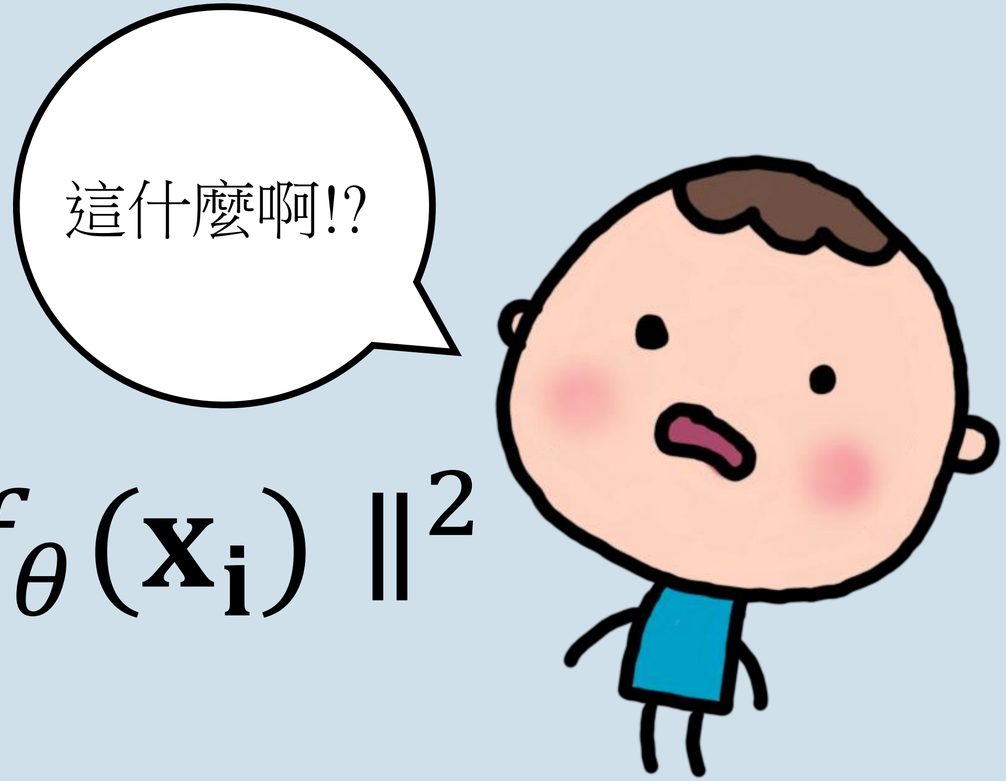
我們當然希望神經網路給出的 $\hat{\mathbf{y}}_i$, 和正確答案 $\mathbf{y}_i$ 差距是越小越好!



$$f_\theta(\mathbf{x_i}) = \hat{\mathbf{y}}_\mathbf{i}$$

Loss Function

Loss function 就是計算神經網路給的答案, 和正確答案差距多少的函數。例如說以下是常見的 loss function:



$$L(\theta) = \frac{1}{2}\sum_{i=1}^{k} \| \mathbf{y_i} - f_\theta(\mathbf{x_i}) \|^2$$

Loss Function

其實就是計算和正確答案的差距!

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{k} \| \mathbf{y_i} - f_\theta(\mathbf{x_i}) \|^2$$

函數學習機給的答案

正確答案

參數調整

那是怎麼調整的呢? 對於某個參數 w 來說, 其實就是用這「簡單的」公式:

$$-\eta \frac{\partial L}{\partial w}$$

參數調整

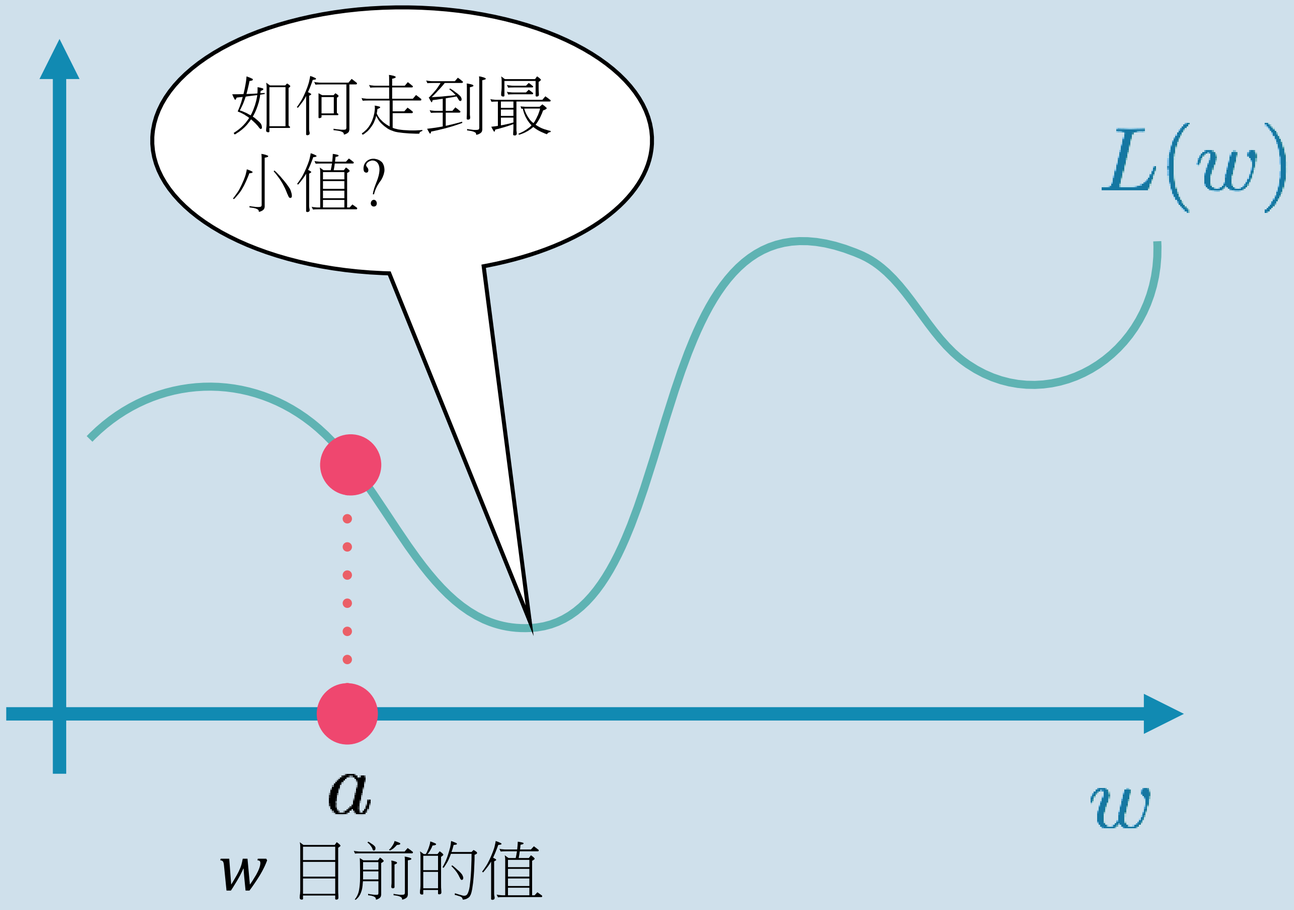
我們來破解函數學習機學習的秘密!

記得 loss function L 是所有權重和偏值 $w_1, w_2, b_1, \ldots$ 等等參數的函數。

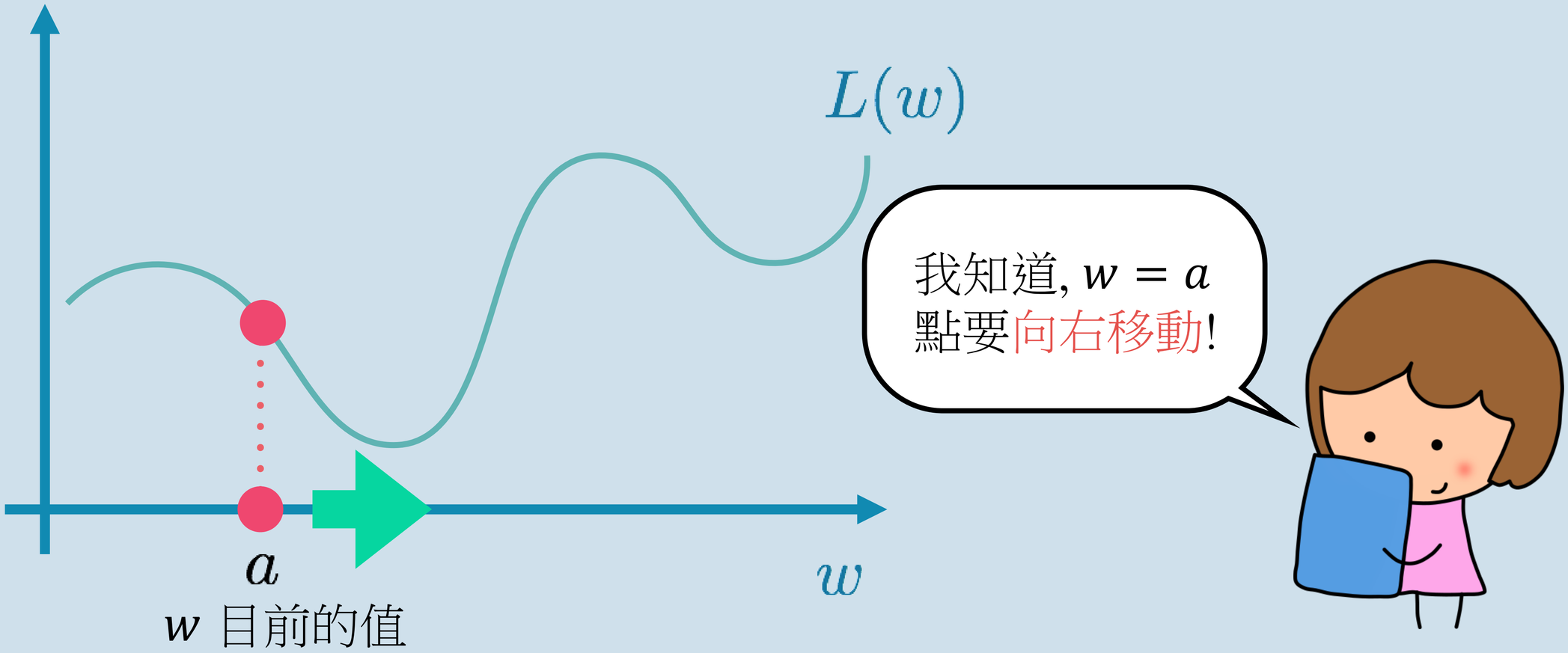
假裝只有一個參數!

數學家都會先簡化問題。

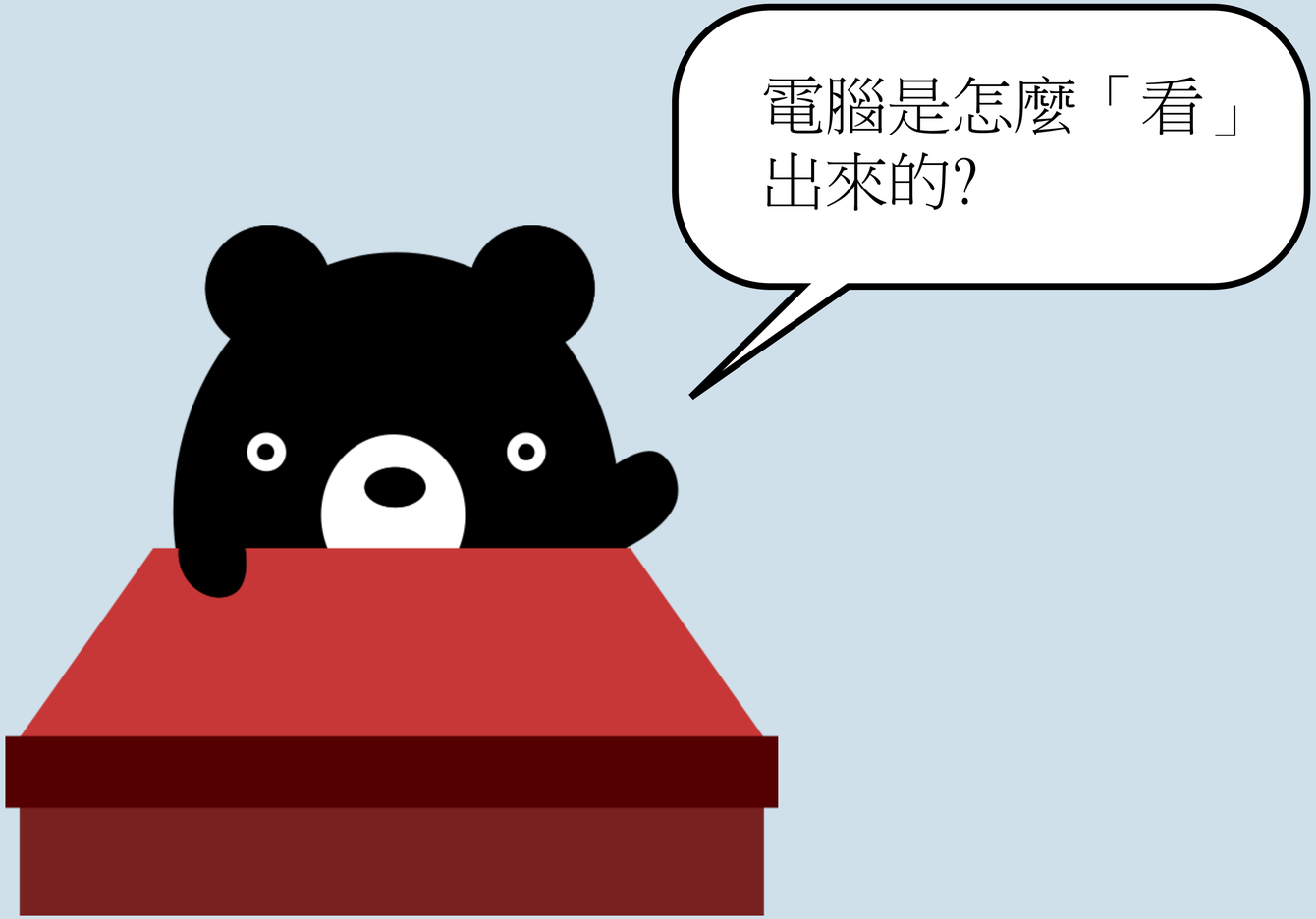
假裝只有一個參數!



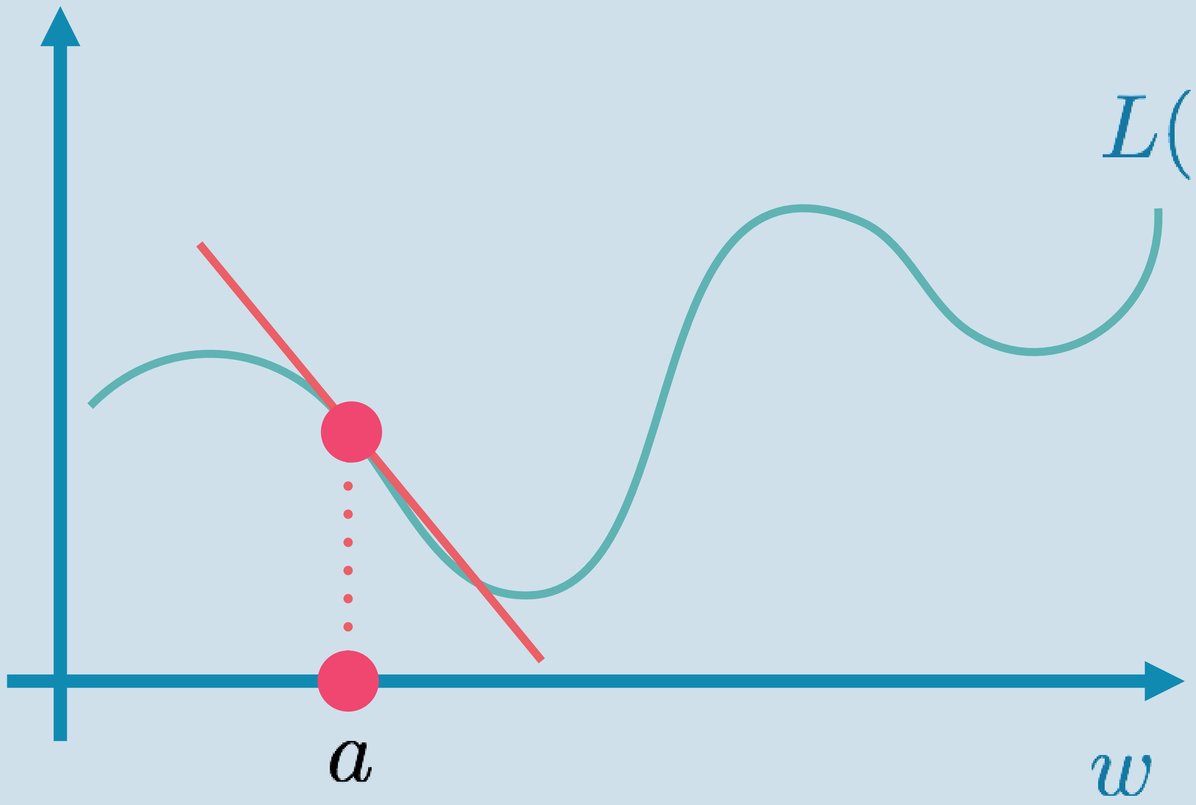
假裝只有一個參數!



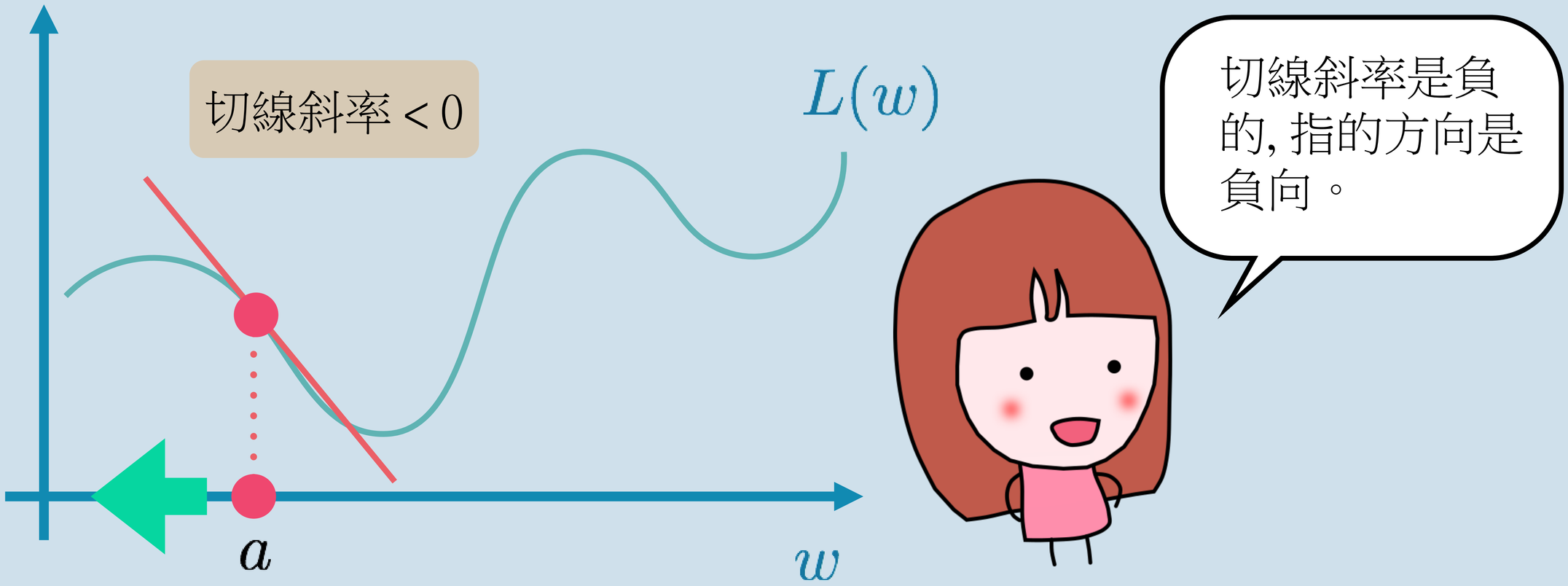
電腦怎麼知道要往哪走?



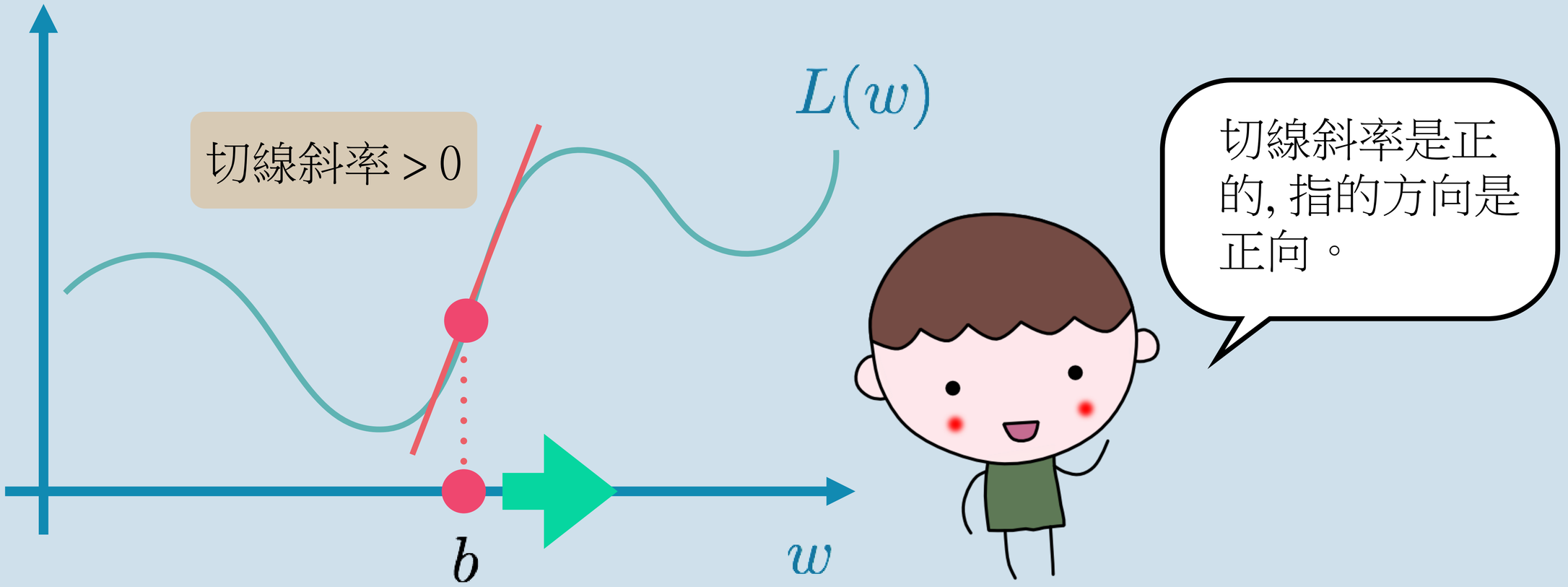
電腦怎麼知道要往哪走?



電腦怎麼知道要往哪走?



電腦怎麼知道要往哪走?



電腦怎麼知道要往哪走?

切線斜率指的方向和我們應該要走的極小值方向是相反的!

事實上切線斜率指的方向正是 (局部) 極大值的方向!

切線斜率炫炫的符號

在 $w = a$ 點的切線斜率符號是這樣:

$$L'(a)$$

對任意的 w 點來說, 我們寫成函數形式是這樣:

$$L'(w) = \frac{dL}{dw}$$

往 (局部) 極小移動!

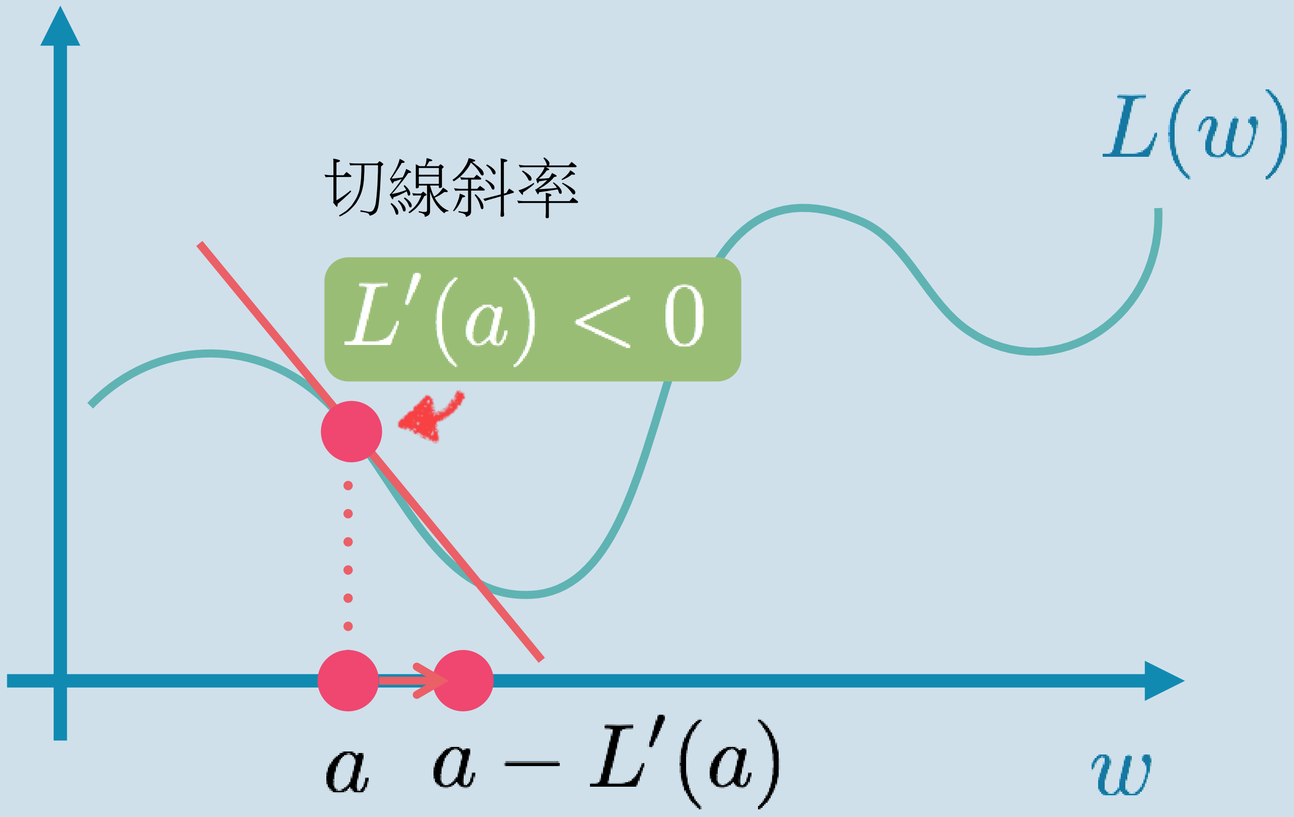
在 $w = a$ 我們可以調整新的 w 值為:

$$a - L'(a)$$

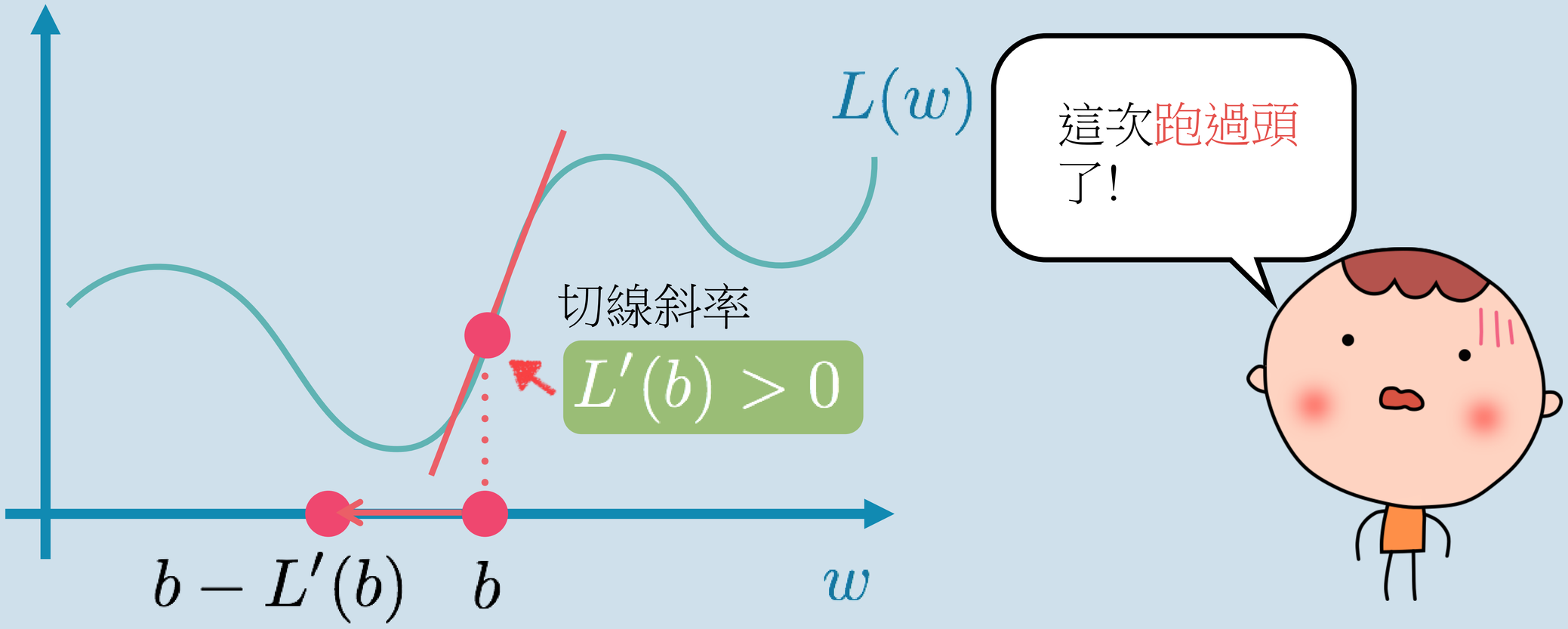
對任意的 w 點來說, 我們會有這樣子的公式去調整 w 的值。

$$w - \frac{dL}{dw}$$

往 (局部) 極小移動!

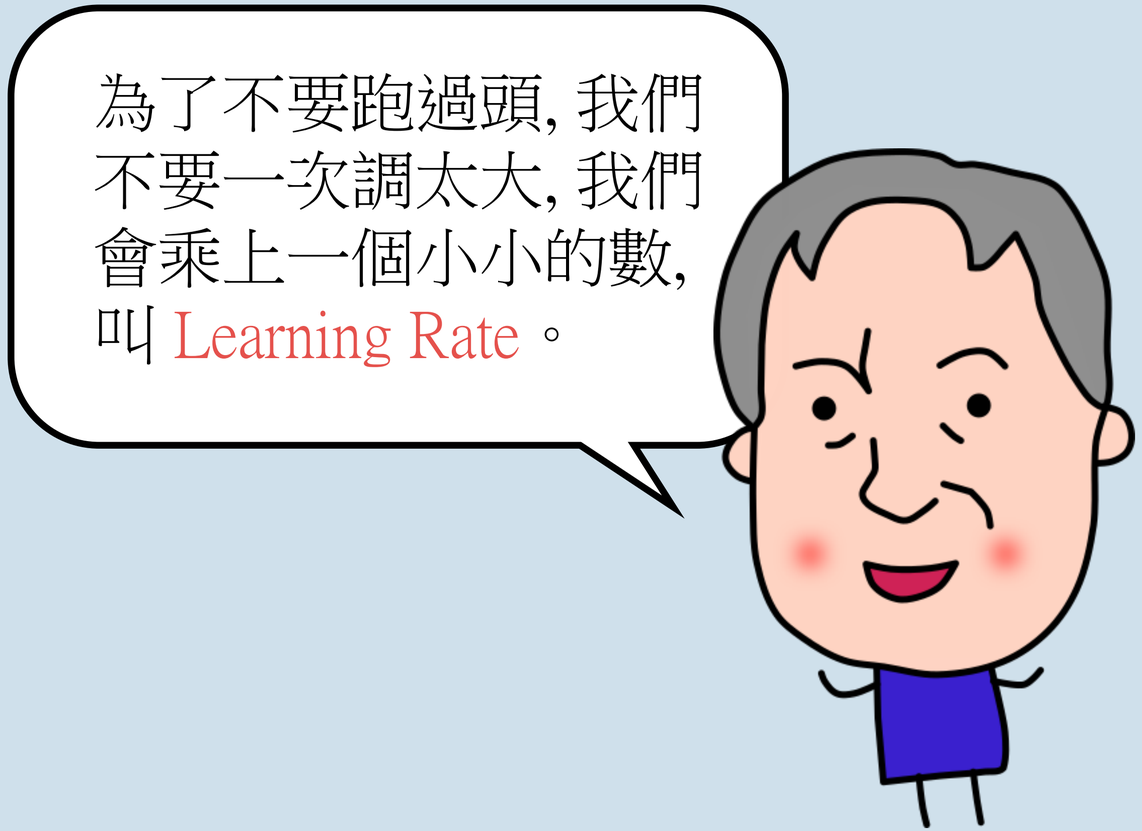


往 (局部) 極小移動!

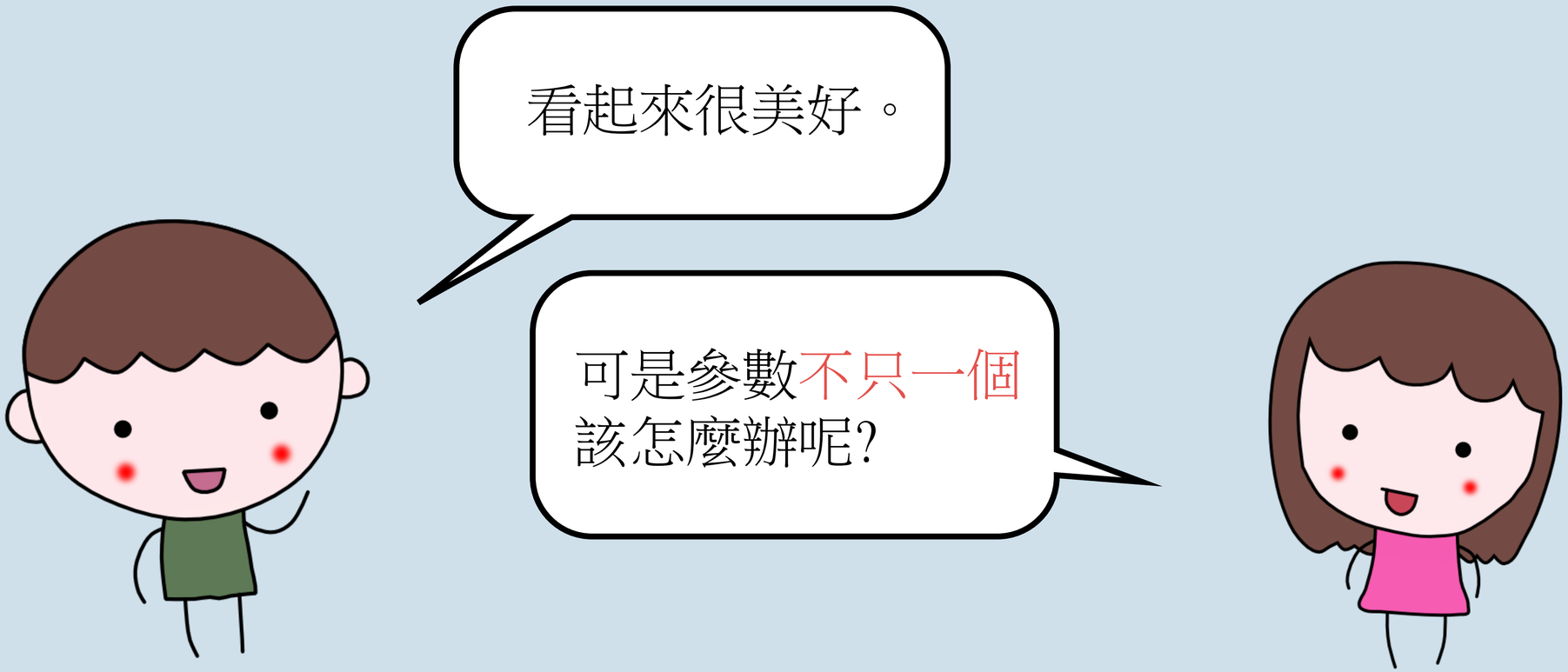


Learning Rate

$$w - \eta \frac{dL}{dw}$$



不只一個參數怎麼辦呢?



數學家比我們想像中邪惡

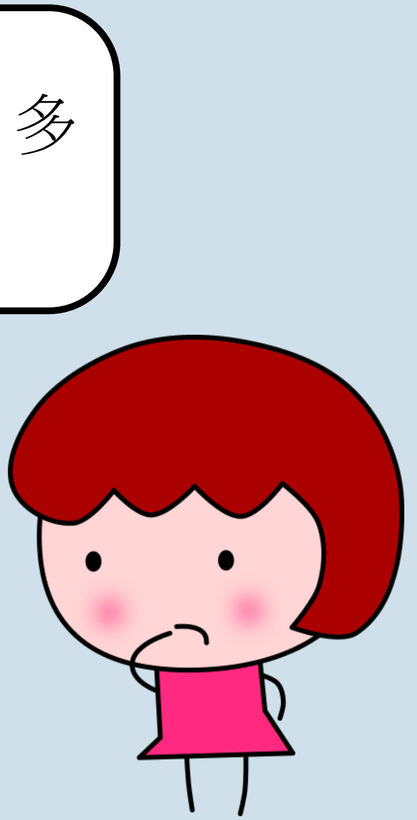
假裝只有一個參數

假設我們的神經網路有三個參數 w_1, w_2, b_1, 而 loss function 長這樣:

$$L(w_1, w_2, b_1) = (b_1 + 2w_1 - w_2 - 3)$$

設我們把這個神經網路初始化, 各參數的值如下:

$$w_1 = 1, w_2 = -1, b_1 = 2$$



假裝只有一個參數

我們現在假裝只有 w_1 一個參數!

不要問我為什麼...

假裝只有一個參數

$$L(w_1, w_2, b_1) = (b_1 + 2w_1 - w_2 - 3)^2$$

$$w_1 = 1, w_2 = -1, b_1 = 2$$

$$L_{w_1}(w_1) = L(w_1, -1, 2) = 4w_1^2$$

L 中只要保留 w_1 這個變數, 其他值都直接代入! 於是 L 當場只剩一個參數!

假裝只有一個參數

$$w_1 \leftarrow w_1 - \eta \frac{dL_{w_1}}{dw_1}$$

然後我們就可以用一個變數調整的方式, 去調整我們的權重!

偏微分

$$\frac{\partial L}{\partial w_1} = \frac{dL_{w_1}}{dw_1}$$

多變數函數, 假裝只有一個變數的微分方式就叫偏微分。

意思是我們會把 w_1 調整為

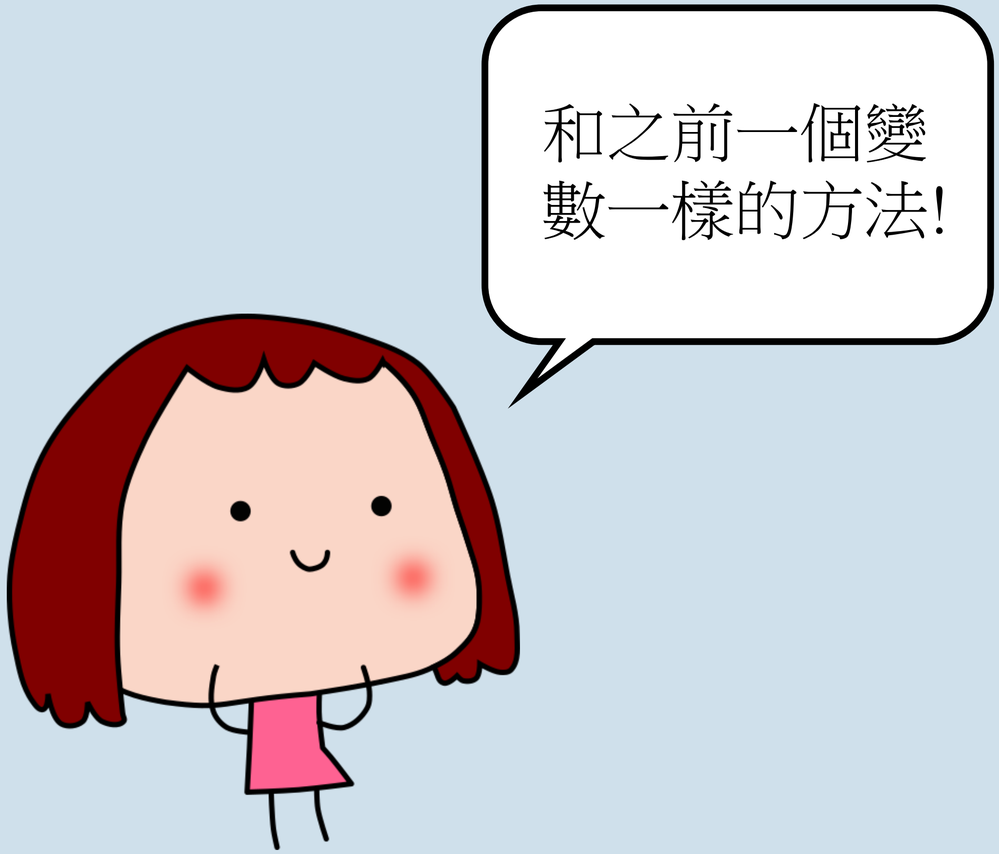
$$w_1 - \eta \frac{\partial L}{\partial w_1}$$

其實就是一一調整!

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_2 \leftarrow w_2 - \eta \frac{\partial L}{\partial w_2}$$

$$b_1 \leftarrow b_1 - \eta \frac{\partial L}{\partial b_1}$$



三個式子寫在一起

$$\begin{bmatrix} w_1 \\ w_2 \\ b_1 \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ w_2 \\ b_1 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b_1} \end{bmatrix}$$

Gradient 梯度

$$\begin{bmatrix} w_1 \\ w_2 \\ b_1 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b_1} \end{bmatrix}$$

這關鍵的向量我們稱為 L 的梯度 (gradient)，記為:

$$\nabla L$$

Gradient Descent

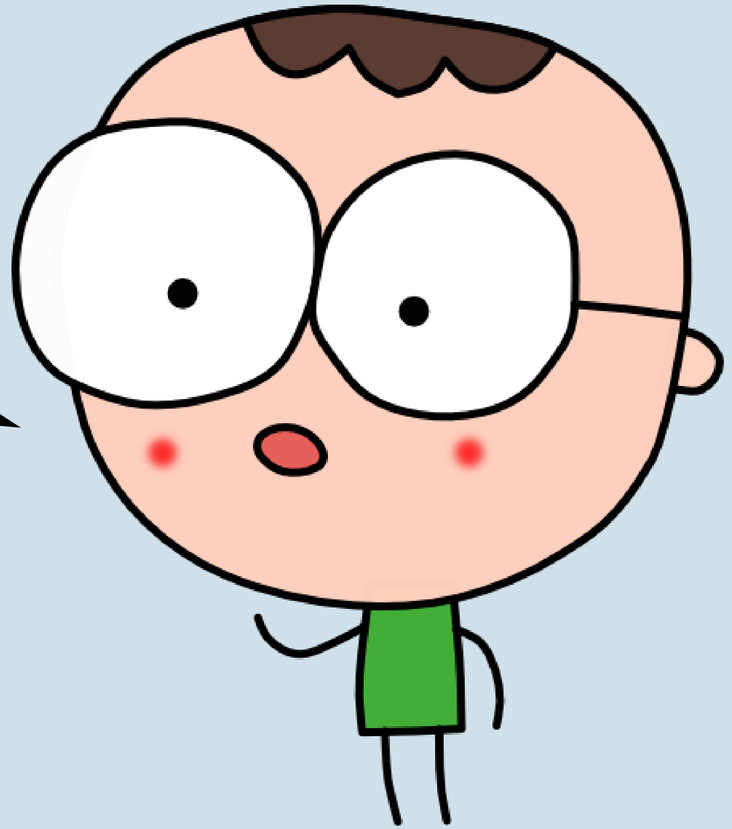
$$\begin{bmatrix} w_1 \\ w_2 \\ b_1 \end{bmatrix} - \eta \nabla L$$

調整參數的公式就可以寫成這樣, 我們稱這個方法為:

Gradient Descent

Gradient Descent

不管你的深度學習函數學習機是怎麼架的, 不管你選什麼 loss function, 你都可以使用 gradient descent 去訓練你的神經網路!



11.

打造第一個神經網路

TensorFlow

現在我們準備一起打造一個神經網路, 我們準備用有名的 TensorFlow 這個 Python 套件。

MNIST 數據集

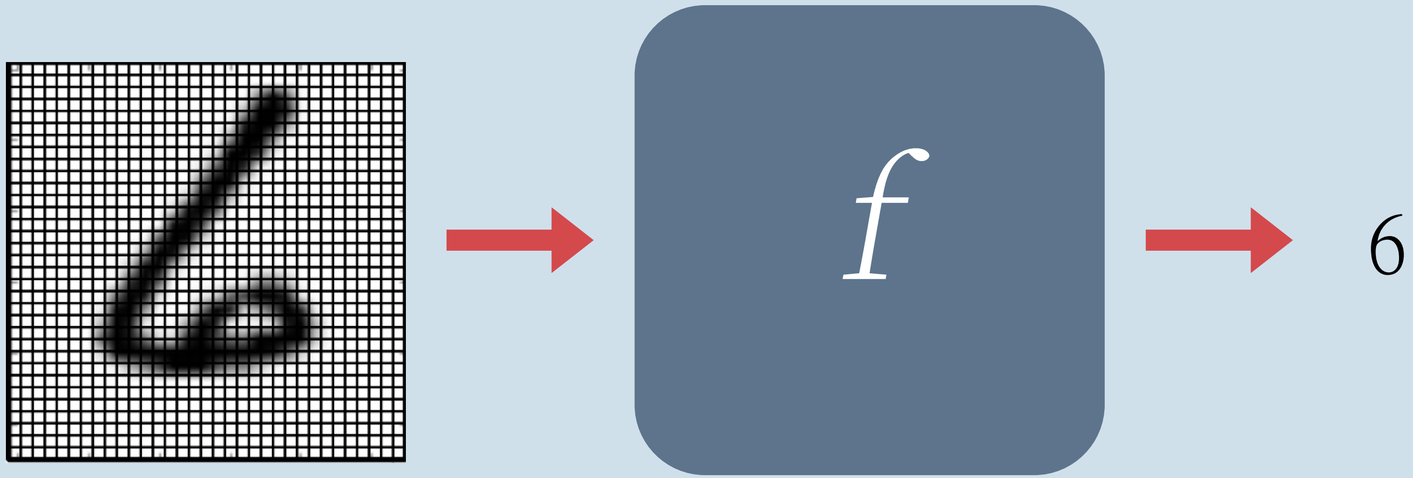
我們介紹一個非常有名的數據集。

Modified

MNIST

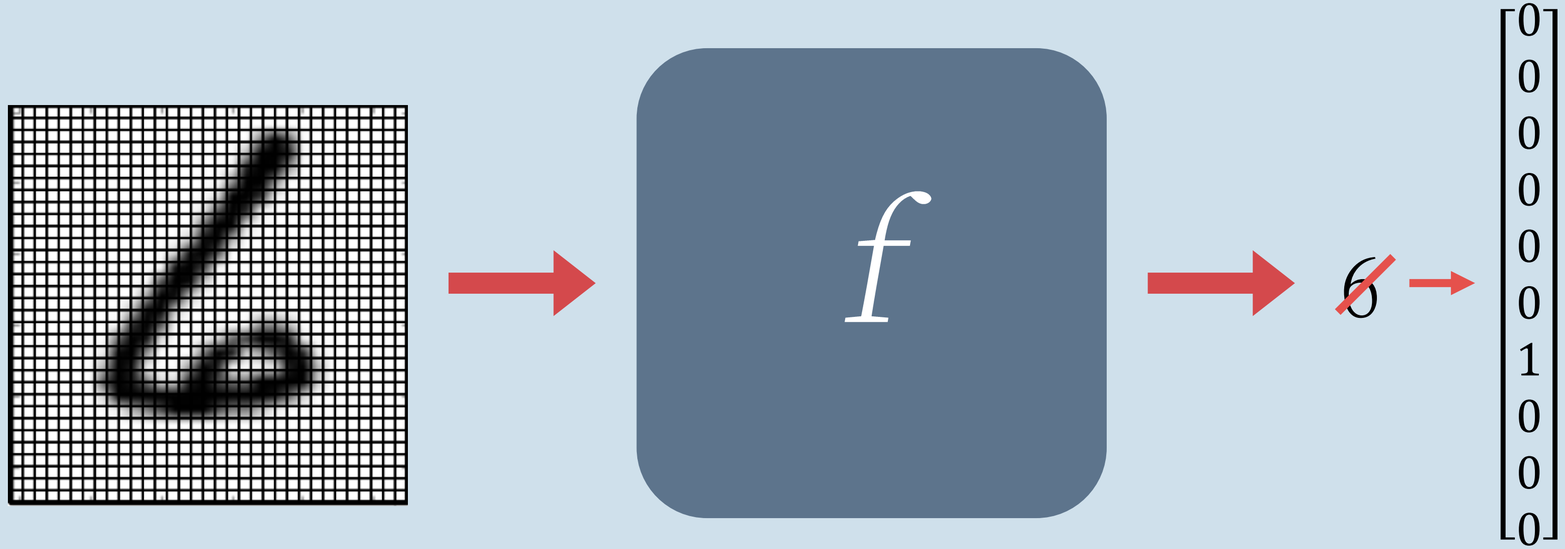
美國國家標準暨技術研究院

把我們的問題, 化為函數



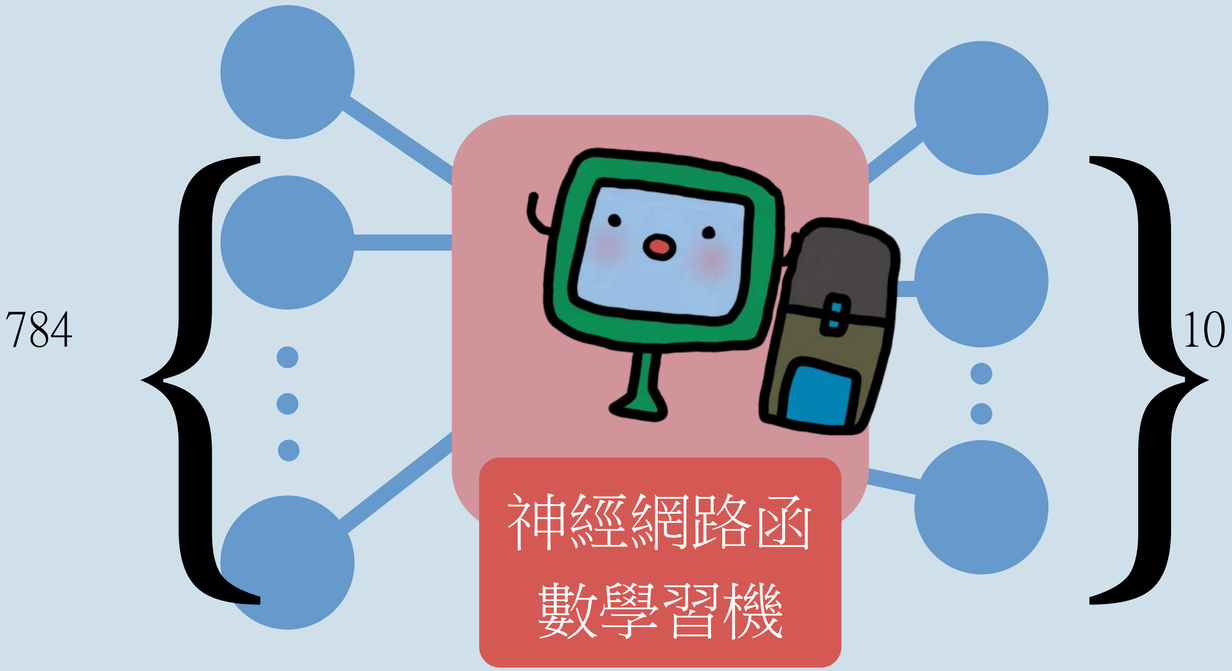
把我們的問題, 化為函數。比如說我們想做手寫辨識, 就是輸入一個掃描的手寫數字, 輸入電腦, 希望電腦輸出這個數字是什麼。

One-hot encoding



我們在做分類問題時, 常會用 one-hot encoding。

確立函數的樣子

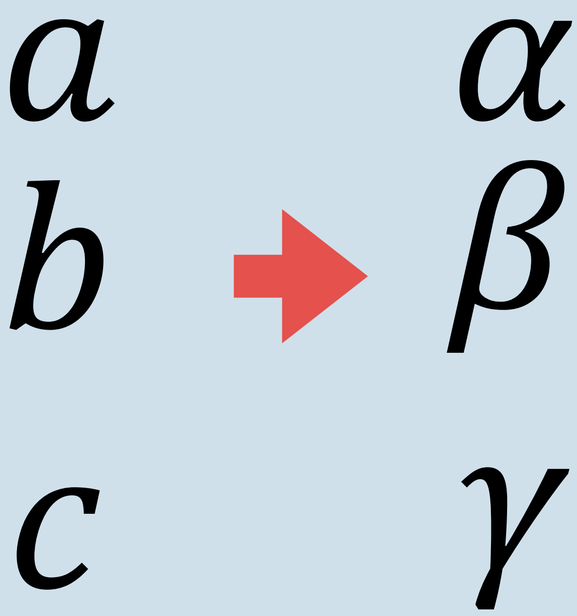


輸入是一張 28×28 的圖,「拉平」是 784 維的向量。

輸出 10 個類別, 也就是 10 維的向量。

Softmax

若干個數字, 我們希望經過一個轉換, 加起來等於 1。但大的還是大的, 小的還是小的, 該怎麼做呢?

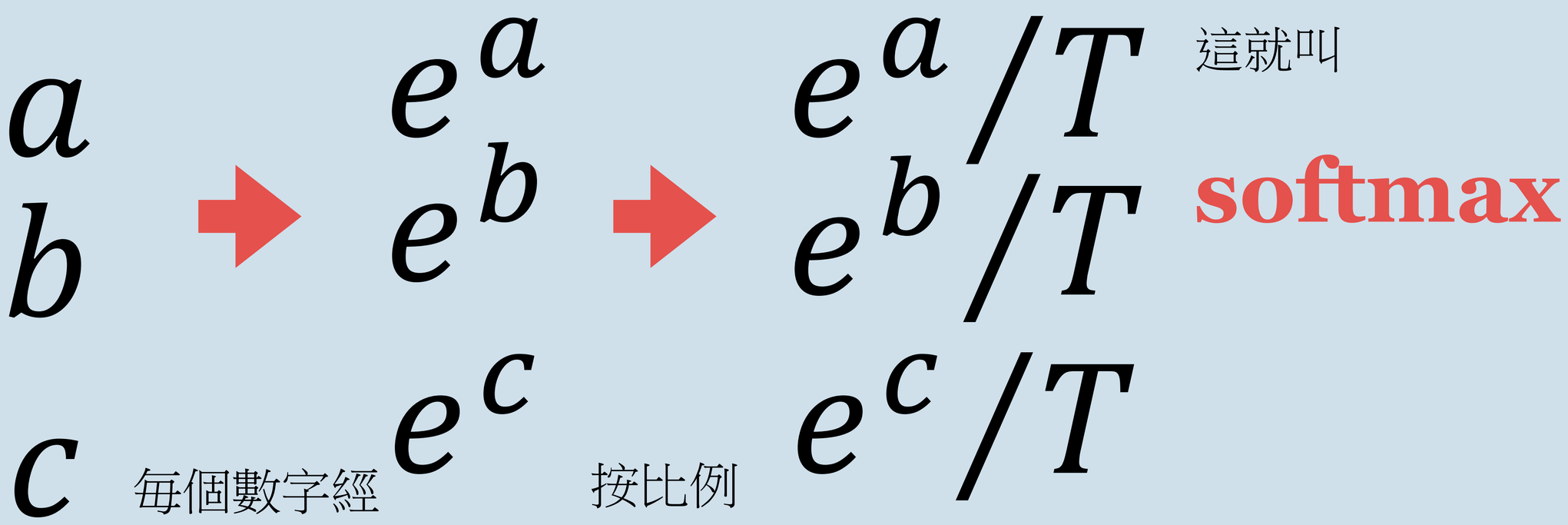


$$\alpha + \beta + \gamma = 1$$

Softmax

很自然我們會這麼做:

$$T = e^a + e^b + e^c$$



01 讀入基本套件

In [1]:

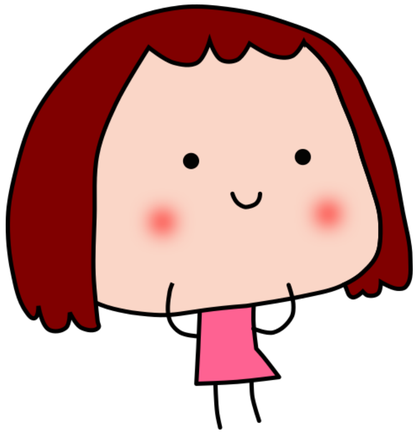
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

02 打造深度學習函數學習機的函式

做 one-hot encoding 的工具

In [2]:

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
```



- **Sequential**: 標準打造一台深度學習函數學習機的方式。
- **Dense**: 做全連結的隱藏層。
- **SGD**: 標準的 gradient descent。

03 讀入 MNIST 數據集

In [3]:

```
from tensorflow.keras.datasets import mnist
```

In [4]:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

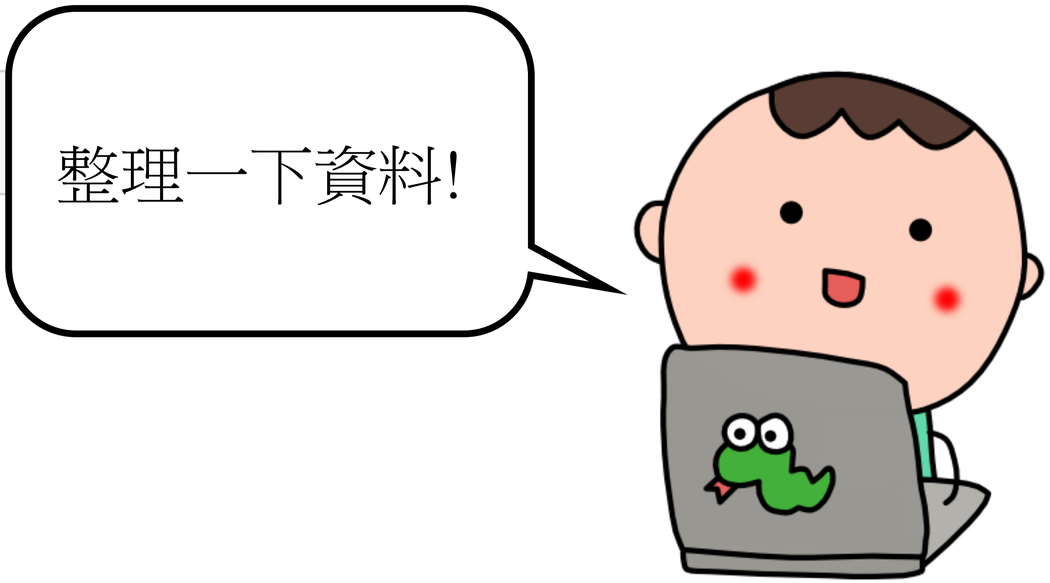
04 整理訓練、測試資料

In [12]:

```
x_train = x_train.reshape(60000, 784)/255
x_test = x_test.reshape(10000, 784)/255
```

In [13]:

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```



- 訓練資料: 拉平、nomalization。
- 測試資料: 做 one-hot encoding。

Step 1: 打造我們的函數學習機

```
In [14]:
model = Sequential()

In [15]:
model.add(Dense(100, input_dim=784, activation='relu'))

In [16]:
model.add(Dense(100, activation='relu'))

In [17]:
model.add(Dense(10, activation='softmax'))
```

假設我們做兩個隱藏層, 都是 100 個神經元、都用 ReLU 做 activation function。

輸出 10 維, 做 softmax。

就差不多只是告訴電腦, 我們怎麼設計神經網路的就結束了!

Step 1: 打造我們的函數學習機

In [18]:

```
model.compile(loss='mse', optimizer=SGD(lr=0.087),
              metrics=['accuracy'])
```

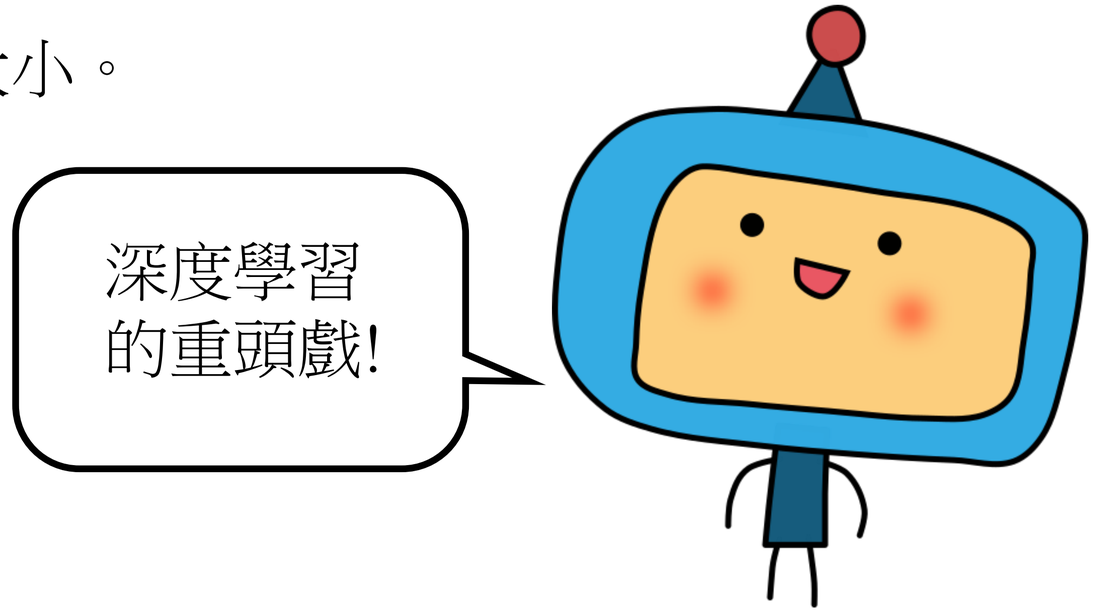
- **loss**: 用哪個 loss function。
- **optimizer**: 我們用 SGD, 設 learning rate。
- **metrics**: 這裡是顯示目前正確率。

Step 2: fit 訓練

In [20]:

```
model.fit(x_train, y_train, batch_size=100, epochs=20)
```

- **batch_size**: mini batch 的大小。
- **epochs**: 訓練次數。



Step 3: Predict

In [21]:

```
predict = model.predict_classes(x_test)
```

這樣 predict 就會是對所有測試資料的預測結果。

12.

圖形辨識天王 CNN

卷積神經網路 CNN

- Convolutional Neural Networks
- 圖形辨識的超級天王

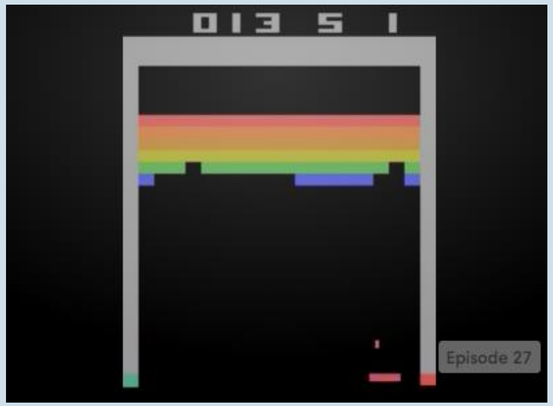
圖形辨識有太多的應用

f

“台灣黑熊”

動物識別。

圖形辨識有太多的應用

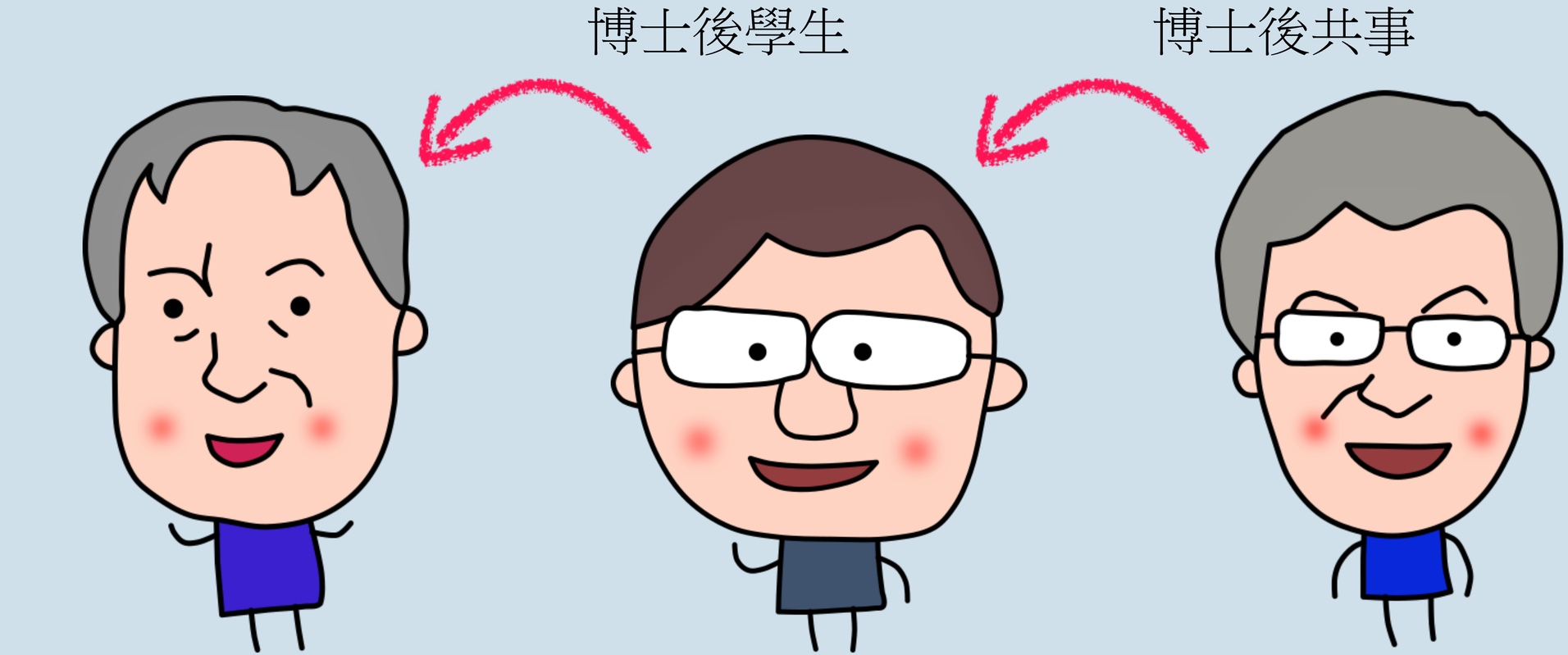


π

往左

用電腦玩遊戲。

深度學習三巨頭



Geoffrey Hinton

Yann LeCun

Yoshua Bengio

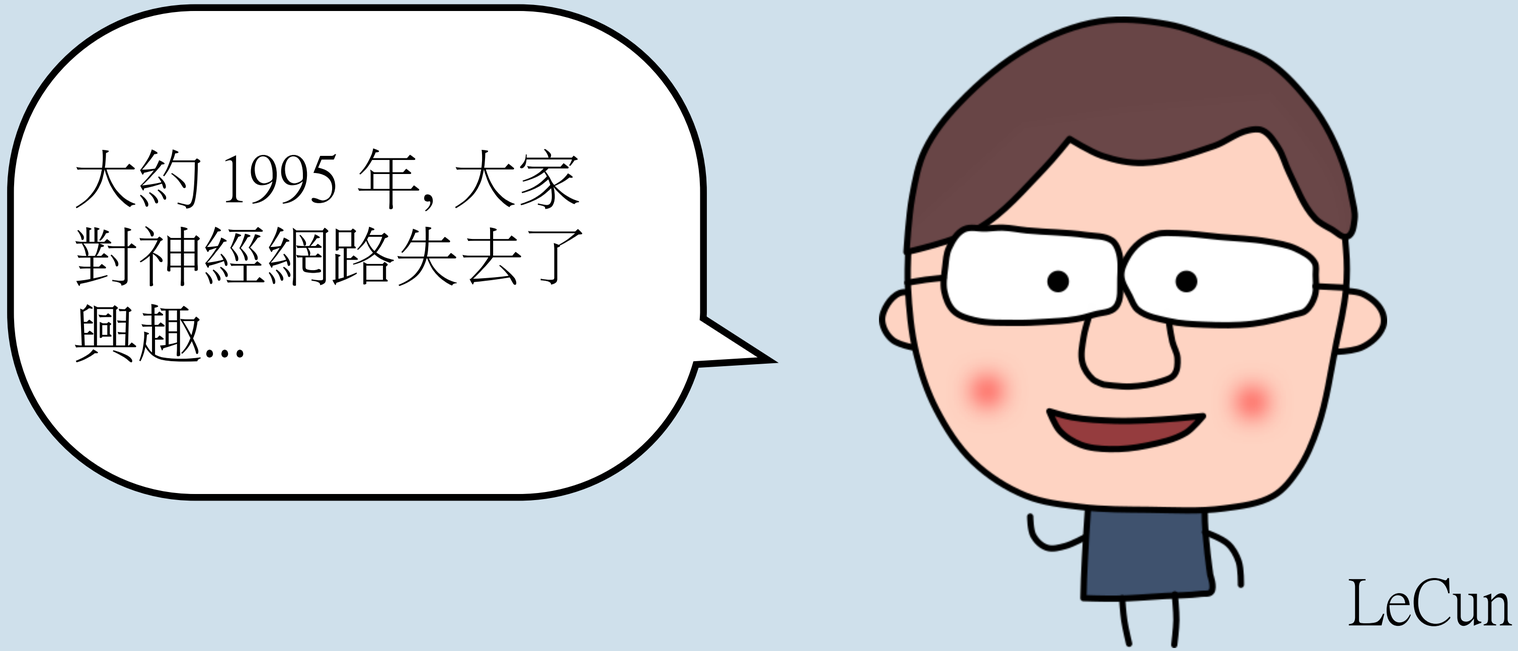
Yann LeCun

Yann LeCun

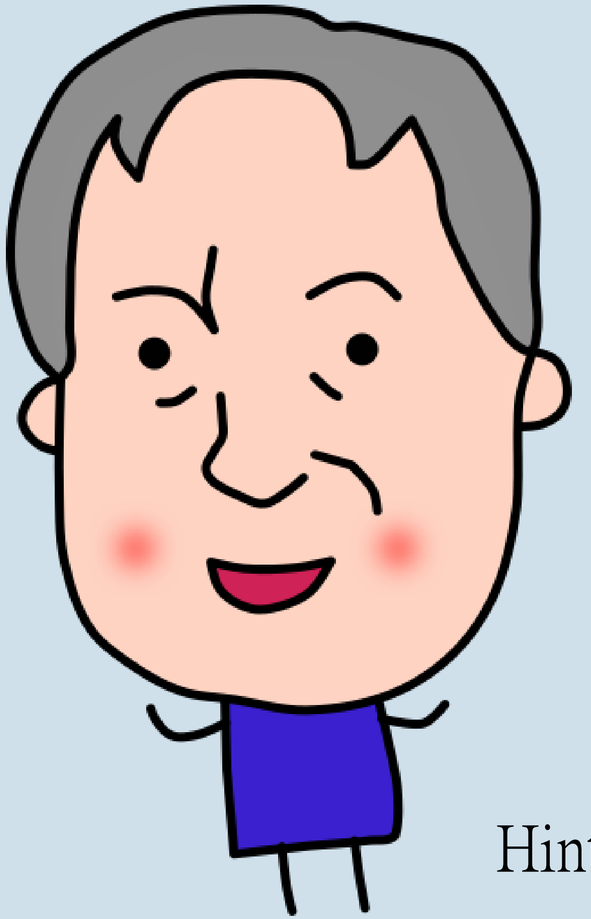
- 1987 Universite Pierre et Marie Curie (巴黎第六大學) 資訊科學博士
- University of Toronto 博士後研究
- 1988- Bell 實驗室
- 2003 NYU 教授
- 2013 Facebook AI Director

博士就研究 ConvNet (不全連結的神經網路)

救了神經網路功臣之一 CNN



救了神經網路功臣之一 CNN

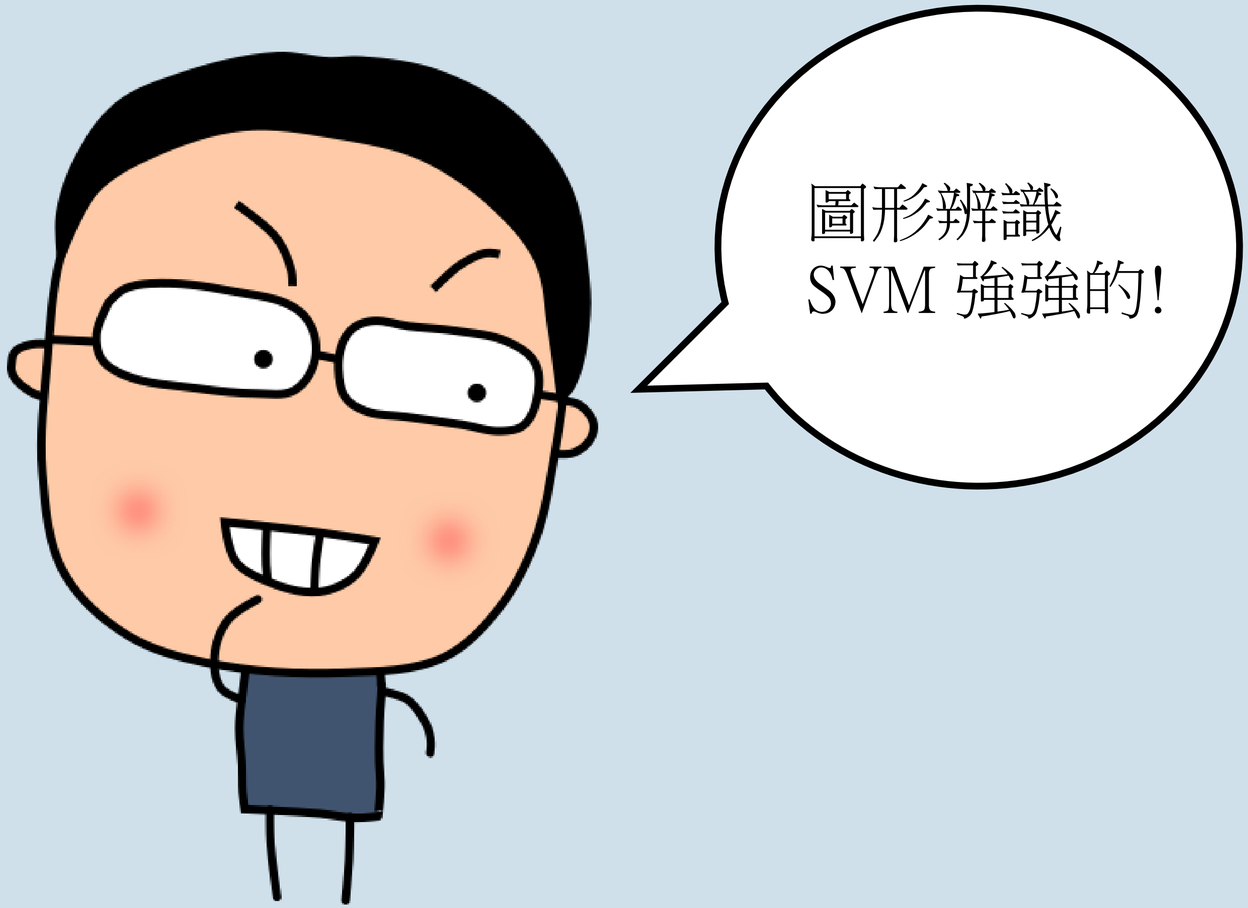


Hinton

神經網路吃虧的地方...

只要別的方法更好、甚至一樣好, 大家就不想用神經網路!

救了神經網路功臣之一 CNN



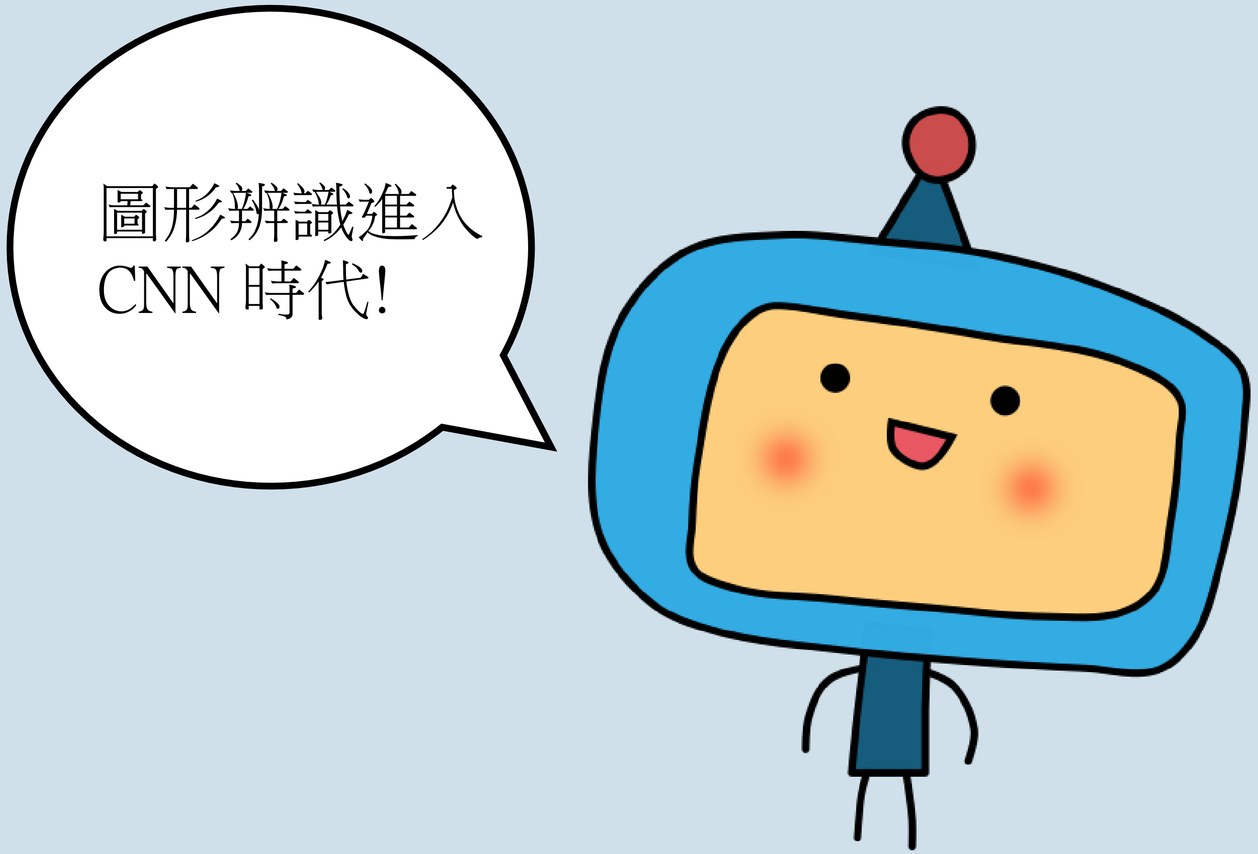
比如 2011 ImageNet 大型圖形辨識比賽冠軍:

XRCE

救了神經網路功臣之一 CNN

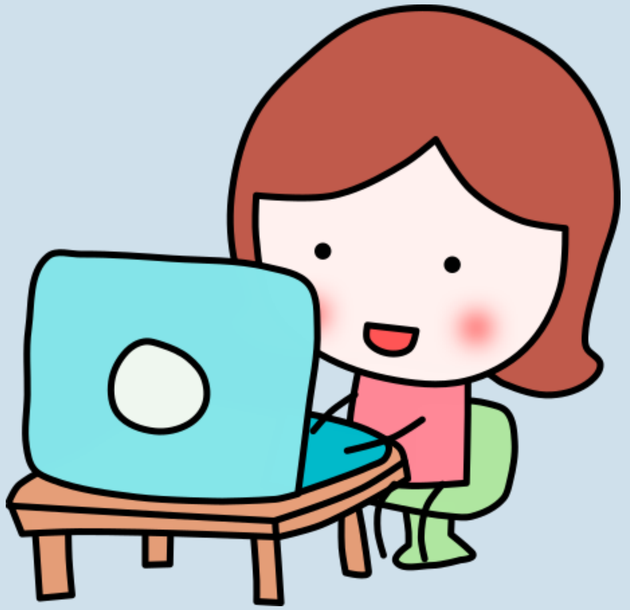
AlexNet

2012 ImageNet 大型圖形辨識比賽冠軍, 狂勝 XRCE!



CNN 經典作品

InceptionV3	GoogLeNet, V1 是 2014 冠軍
VGG16/VGG19	2014 亞軍
ResNet50	2015 冠軍



CNN 的兩種隱藏層

卷積層 (convolutional layer)

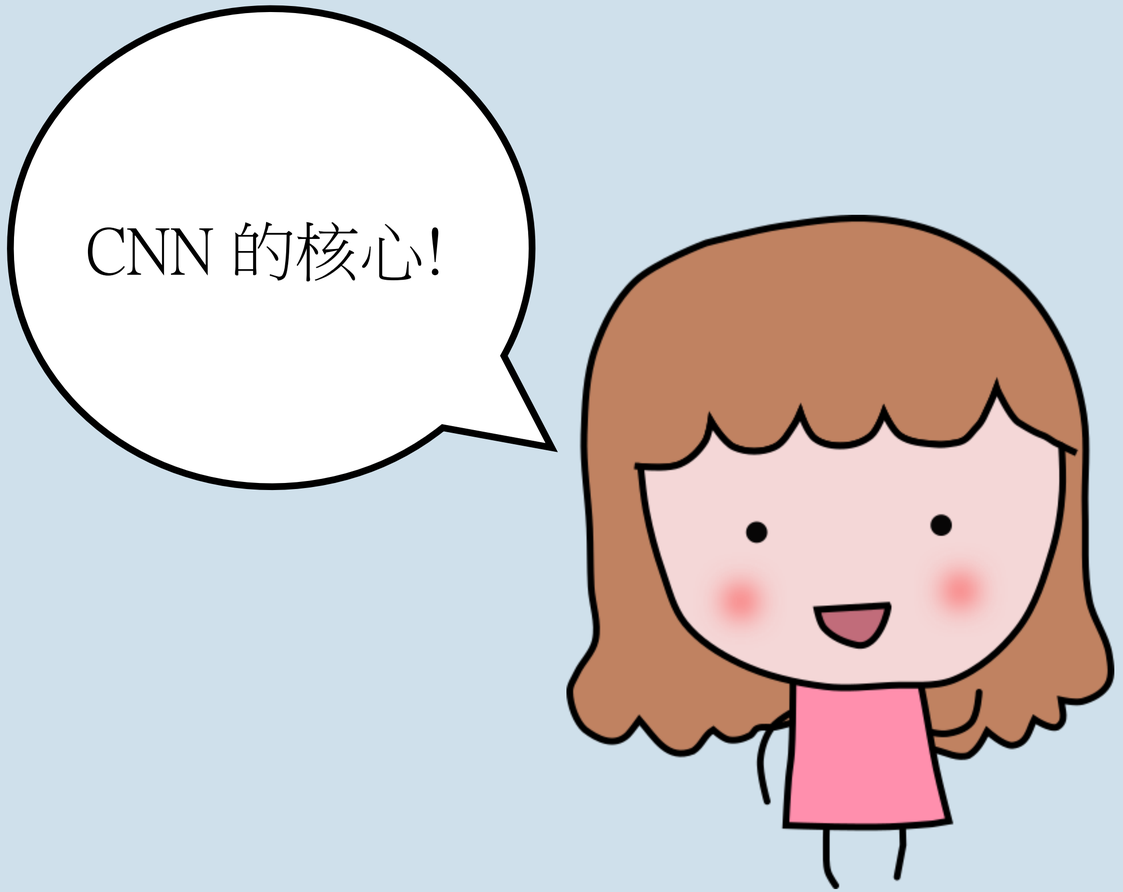
池化層 (pooling layer)

卷積層 Convolutional Layer

1

Convolutional Layer

卷積層



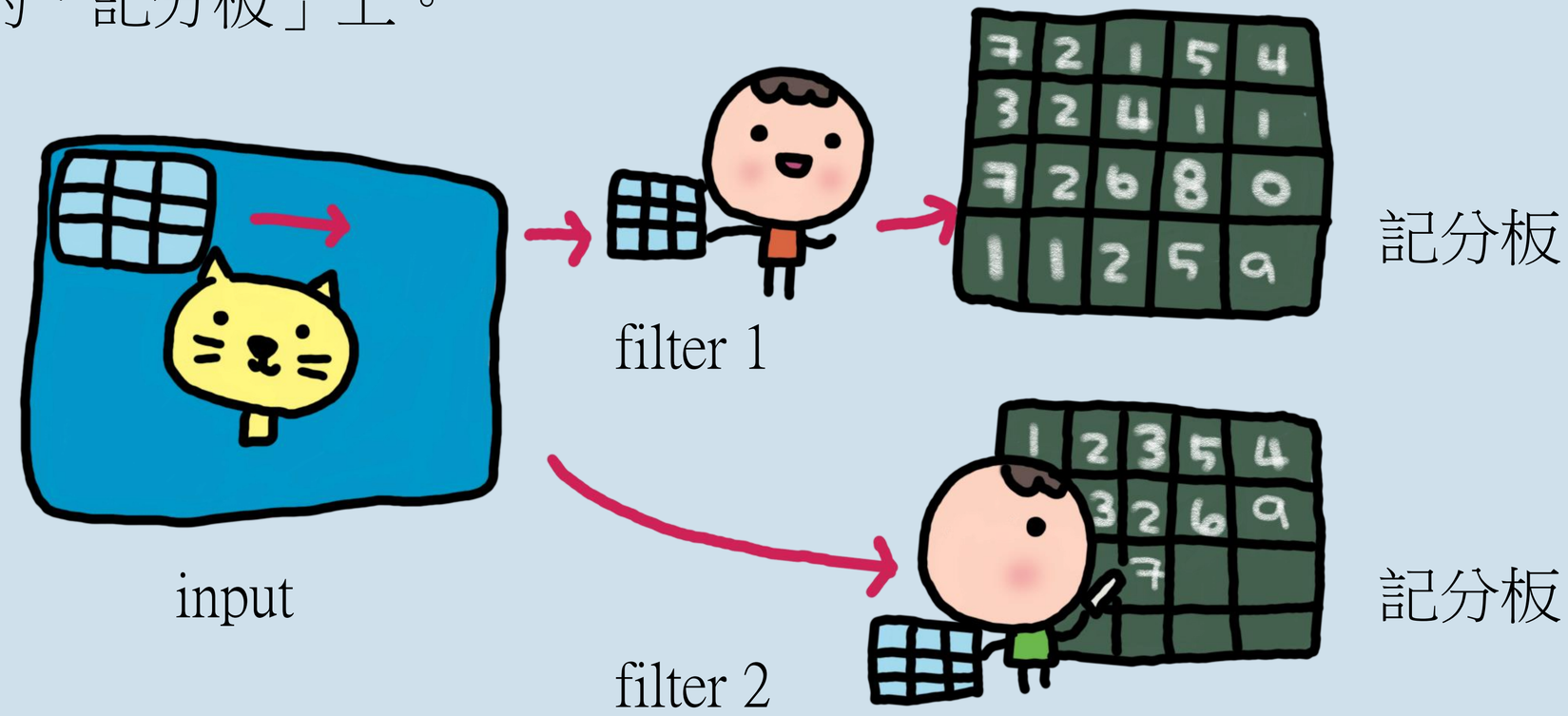
設計卷積層

設計一層卷積層, 我們只需要...

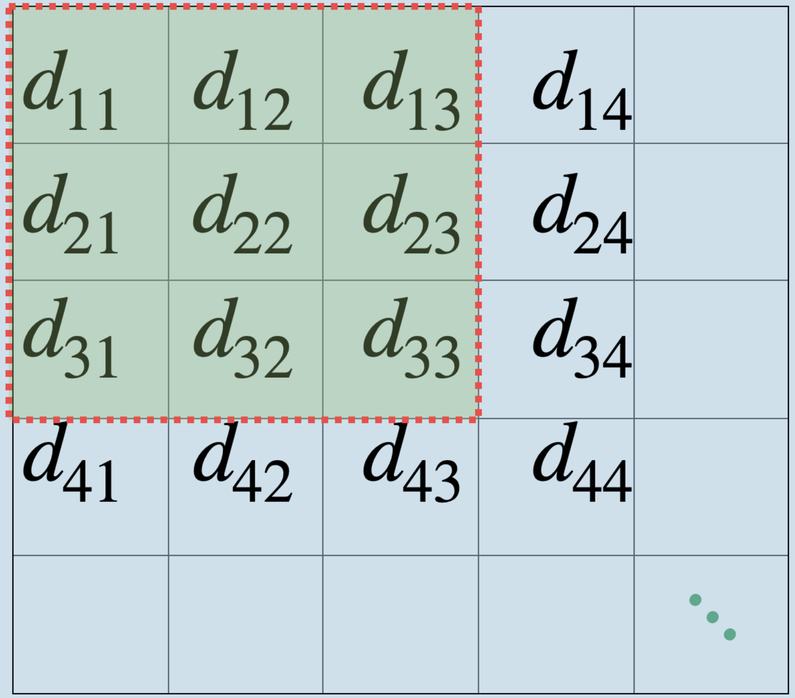
1. 幾個 filter
2. 每個 filter 的大小 (比如 3×3)

Filter 的作用

每個 filter 看一個特徵, 掃過每一個點,
紀錄該點附近的「特徵強度」。於是
紀錄在自己的「記分板」上。



Filter 卷積的運算



input

$\star$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

filter

一個 filter 就像個權重做加權和, 也就是內積 (dot product)。

$=$

$$\begin{aligned} & d_{11} \times 0 + d_{12} \times 1 + d_{13} \times 0 \\ & + d_{21} \times 0 + d_{22} \times 1 + d_{23} \times 0 \\ & + d_{31} \times 0 + d_{32} \times 1 + d_{33} \times 0 \end{aligned}$$

Filter 卷積的運算

內積

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

這學來的

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

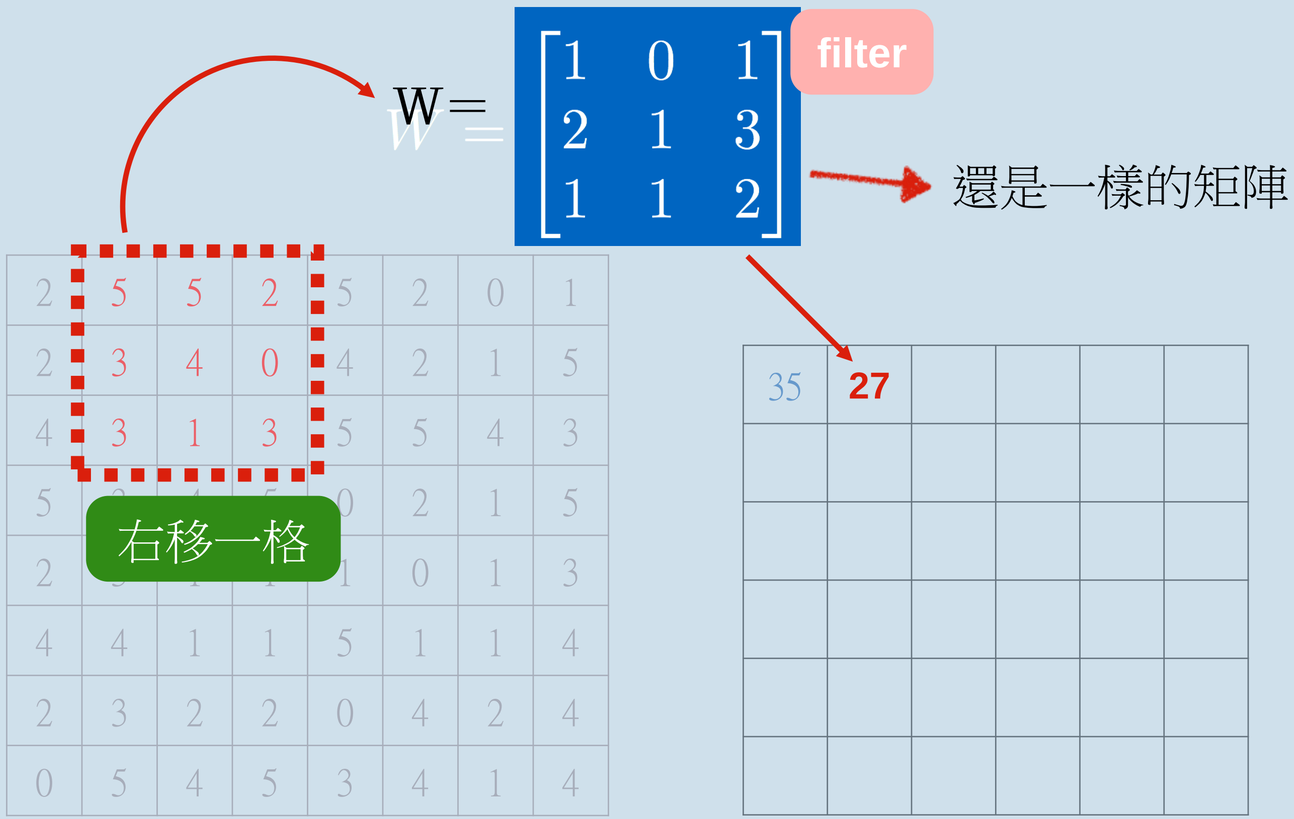
想成這是一張圖所成的矩陣

35					

記分板

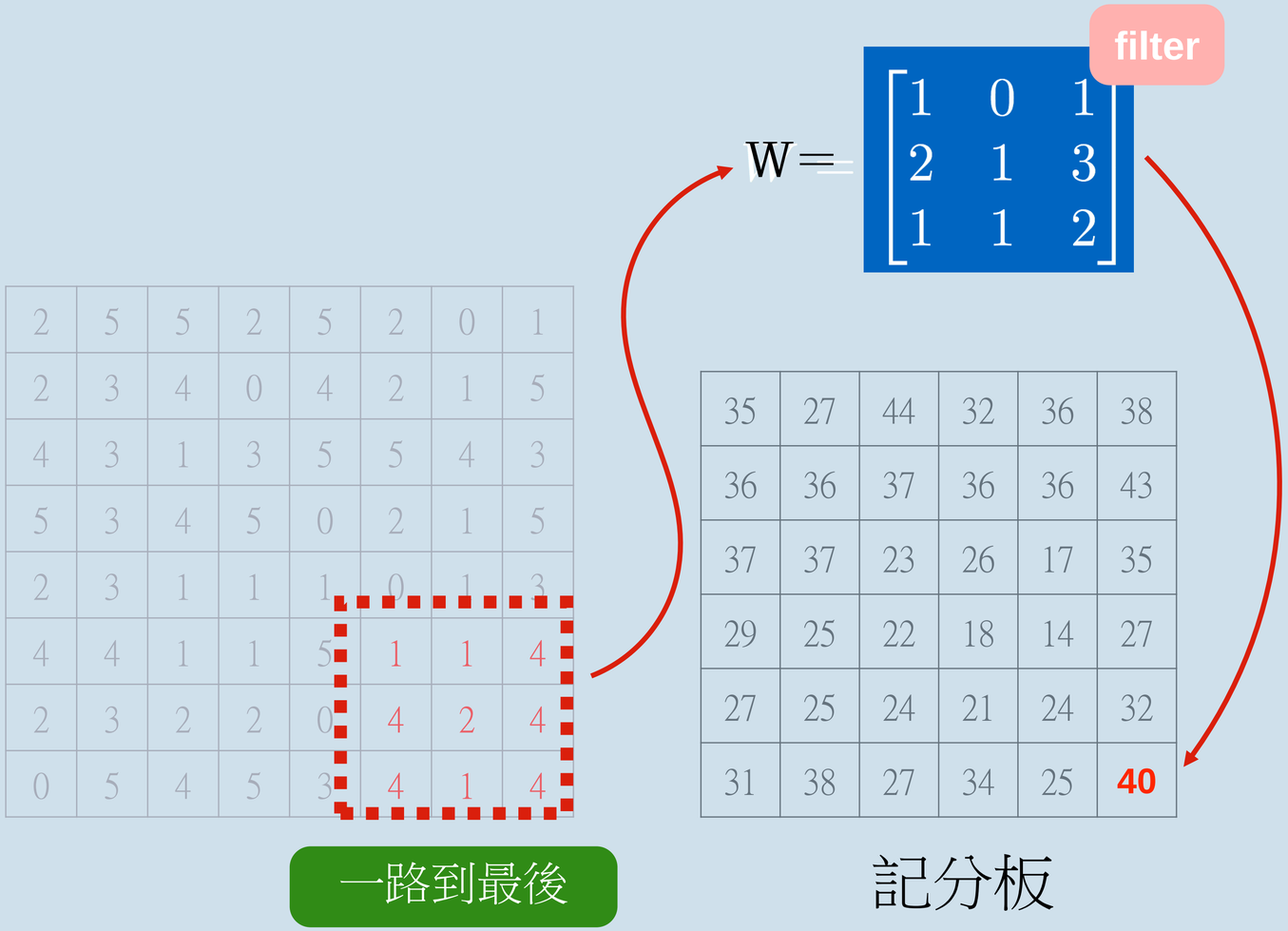
一個 filter 就看一個特徵, 紀錄在記分板上。

Filter 卷積的運算



記分板

Filter 卷積的運算



掃到最後, 完成這個 filter 的計分板。

要注意的是, 這內積的部份只有我們原本的加權和, 事實上還是要加上偏值、經激發函數轉換送出!

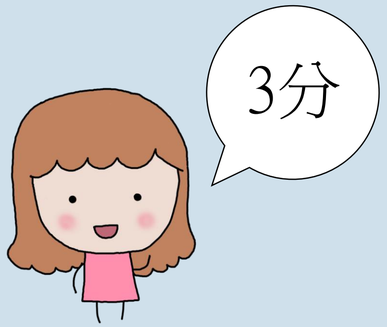
為什麼卷積會抽取特徵?

0	1	0
0	1	0
0	1	0

圖

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter 1



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Filter 2

我們來看同一張圖,對兩個不同的 filter 運算的結果。

可以看出一樣的會得高分!

為什麼卷積會抽取特徵?

1	0	0
0	1	0
0	0	1

圖

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter 1

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Filter 2

換另一張圖發現, 真的像的會得到高分!

卷積層其實神經元動作是一樣的!

輸入的圖每個像素可當成一個神經元

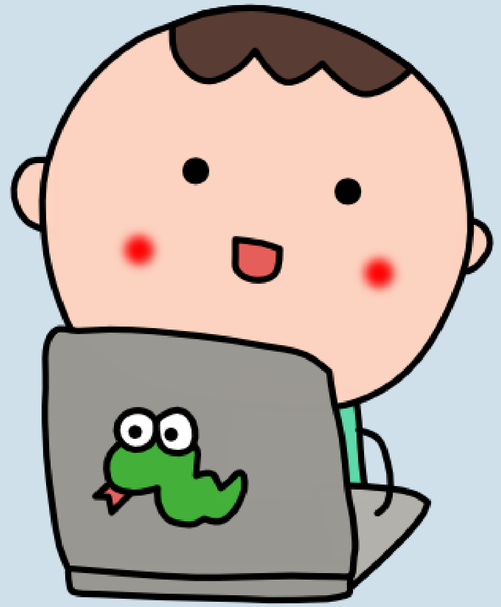
圖片上的點是一個個輸入層神經元

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40



記分板也是一個個神經元組成

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

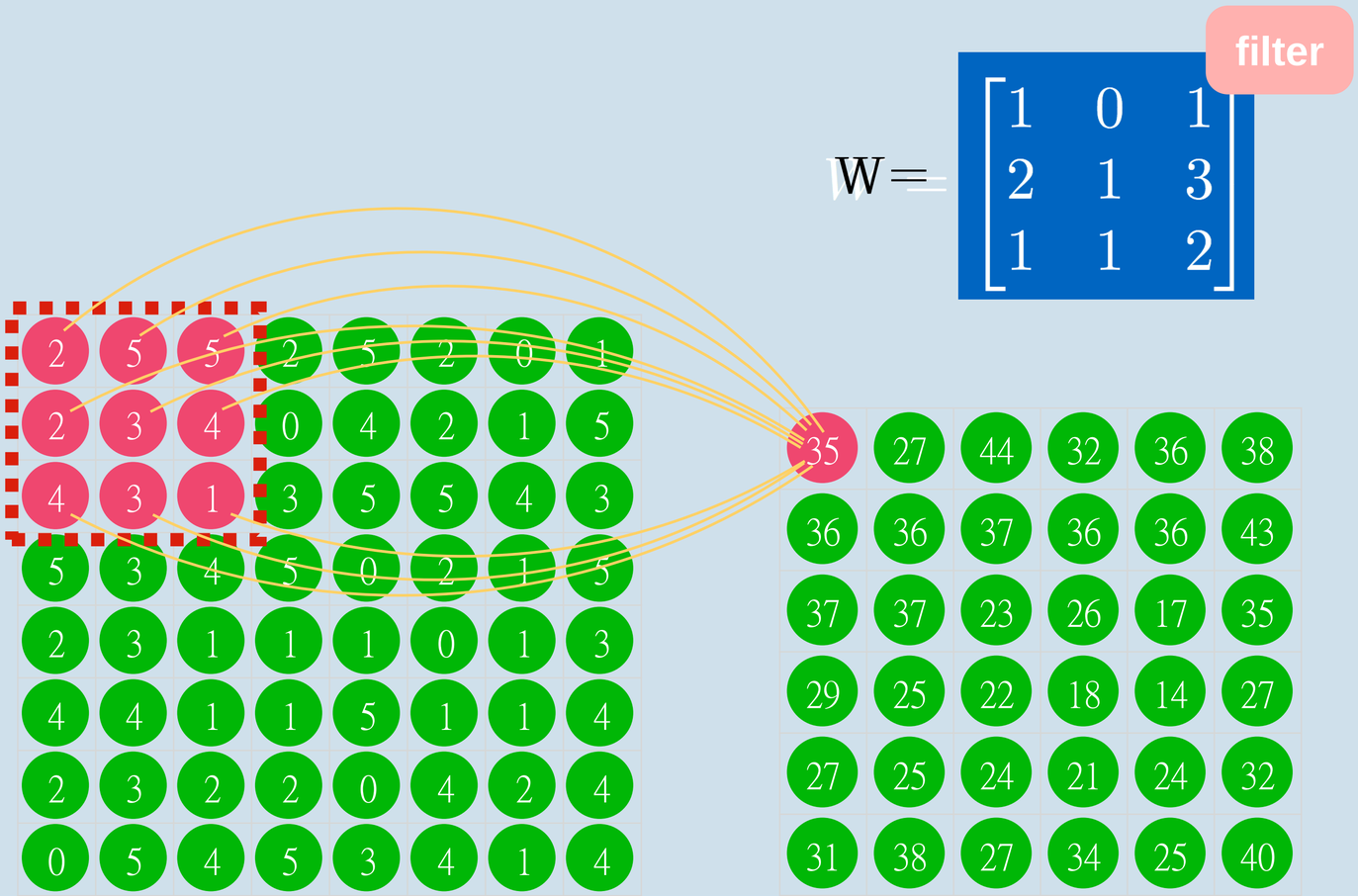
2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

記分板每一個分數的位子也是一個神經元。

Conv 層也是一個個神經元

不是完全連結的神經網路



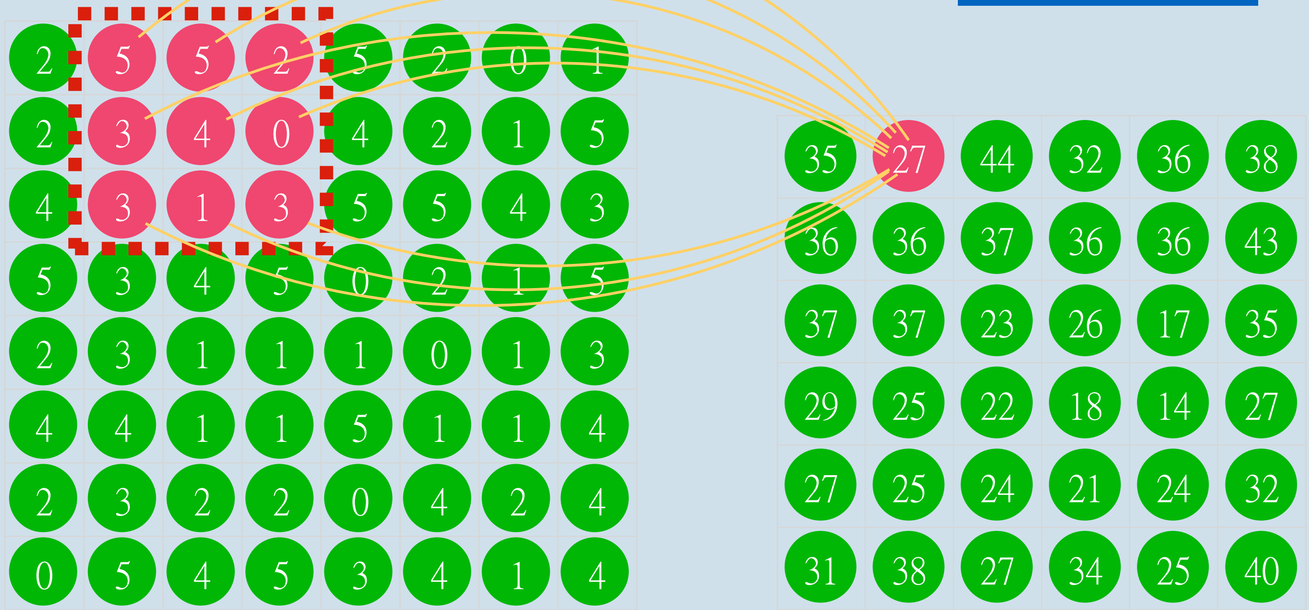
記分板一個數字 (一個神經元) 只和輸入的九個神經元相連。

兩層中沒有完全相連

權重是相同的 (對同一個 filter)

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$



再來 share 同樣的 weights

記分板的大小

注意一張原本 8 × 8 的圖, 經一個 3 × 3 大小的 filter 卷積, 會得到差不多大小 6 × 6 記分板!

而且再來我們會討論到, 我們甚至更喜歡把記分板做成和原本圖的大小一樣。

那要是我們有 10 個記分板, 不就十倍的數據量!?

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

記分板的大小

1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0

圖

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter 1

0	0	0
0	0	0
0	0	0

記分板

如前述的掃描方式, 記分板比原來圖略小, 但邊緣常會掃不到 (像本例中的直線)。

```
padding="valid"
```

記分板的大小

0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

圖

外面加圈 0!

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter 1

padding="same"

2	0	0	0	0
3	0	0	0	0
3	0	0	0	0
3	0	0	0	0
2	0	0	0	0

記分板

注意和原圖一樣大!

我們喜歡把原圖外面加圈 0, 讓記分板和原圖大小一樣!

注意這次有「看到」直線了。

池化層 Pooling Layer

2

Max-Pooling Layer

最大池化層

設計池化層

設計一層池化層, 我們只需要...

1. 決定多大的池化區域 (比如 2×2)
2. 用哪種池化方式 (最常用取極大值)

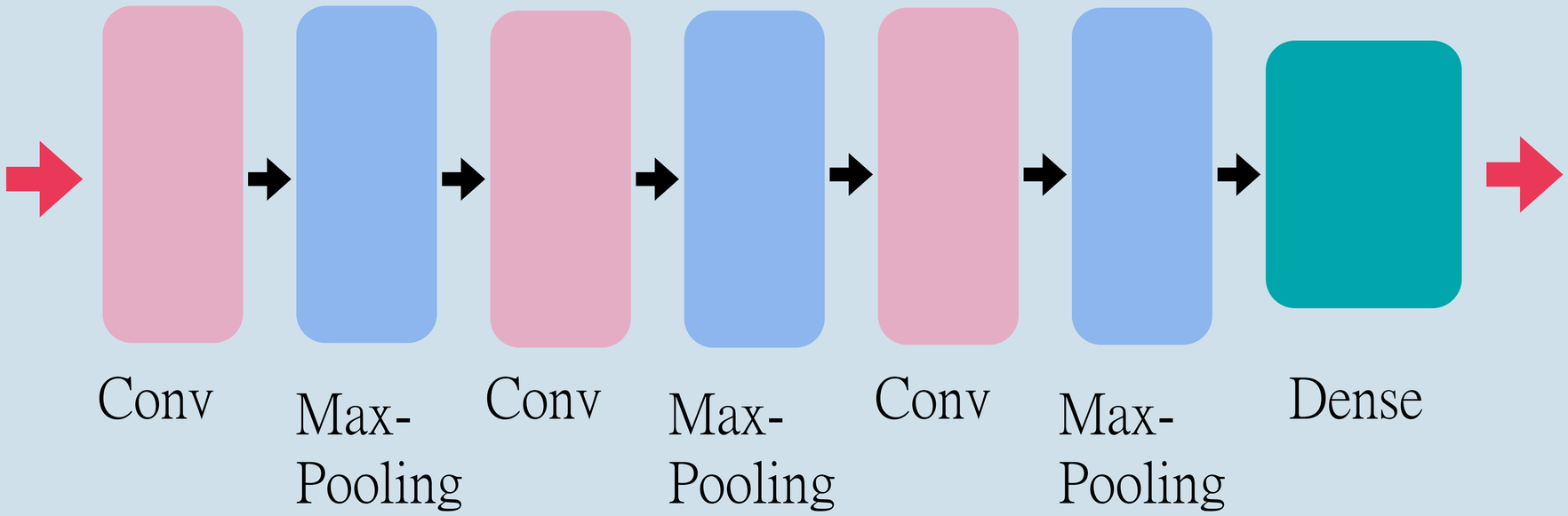
Max-Pooling

35	27	**44**	32	36	38
36	**36**	37	36	36	**43**
37	37	23	**26**	17	**35**
29	25	22	18	14	27
27	25	24	21	24	32
31	**38**	27	**34**	25	**40**

36	**44**	**43**
37	**26**	**35**
38	**34**	**40**

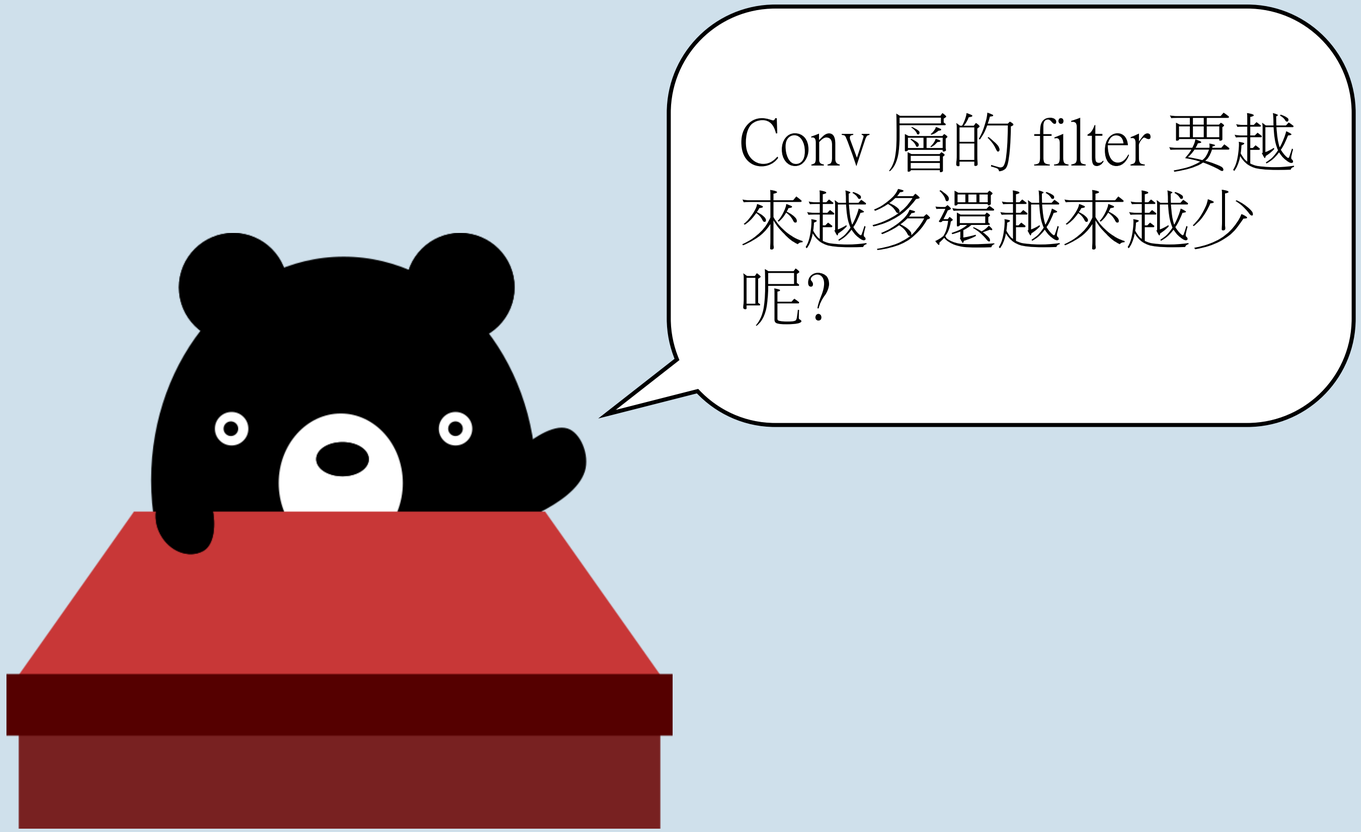
每區選出最大的!!

常見 CNN 設計架構

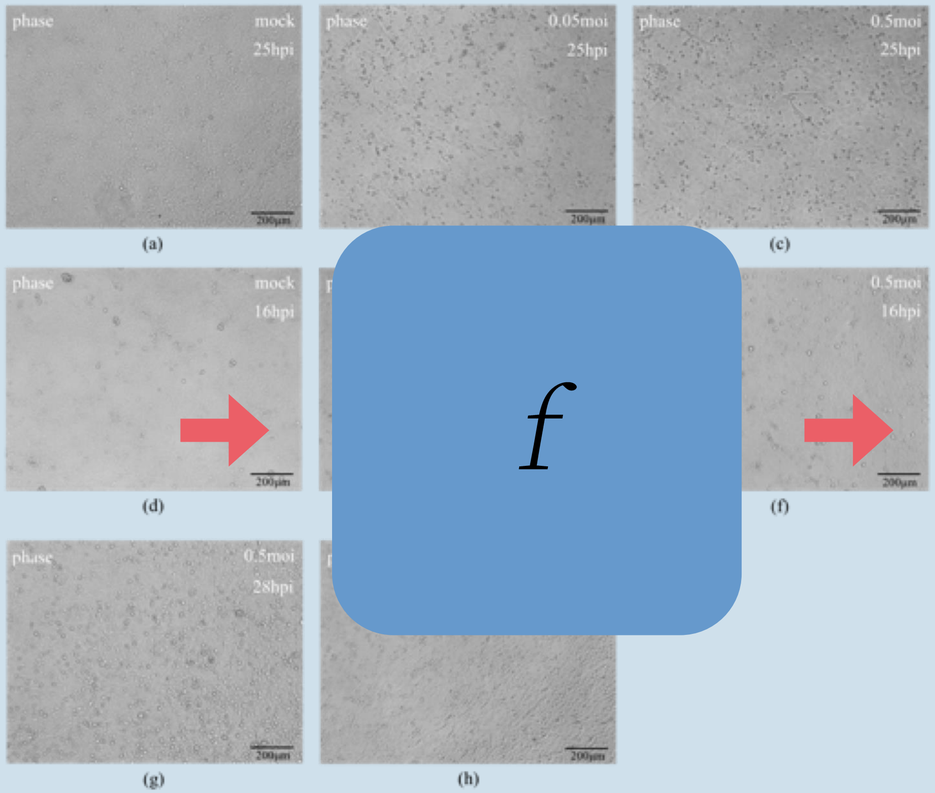


可以不斷重覆卷積、池化、卷積、池化... 最後再接全連結神經網路總結。

重要問題



標準 CNN 應用



f

感染/未感染

Differentiation of Cytopathic Effects Induced by Influenza Virus Infection Using Deep Convolutional Neural Networks

Ting-En Wang, Tai-Ling Chao, Hsin-Tsuen Tsai, Pi-Han Lin, Yen-Lung Tsai, Sui-Yuan Chang

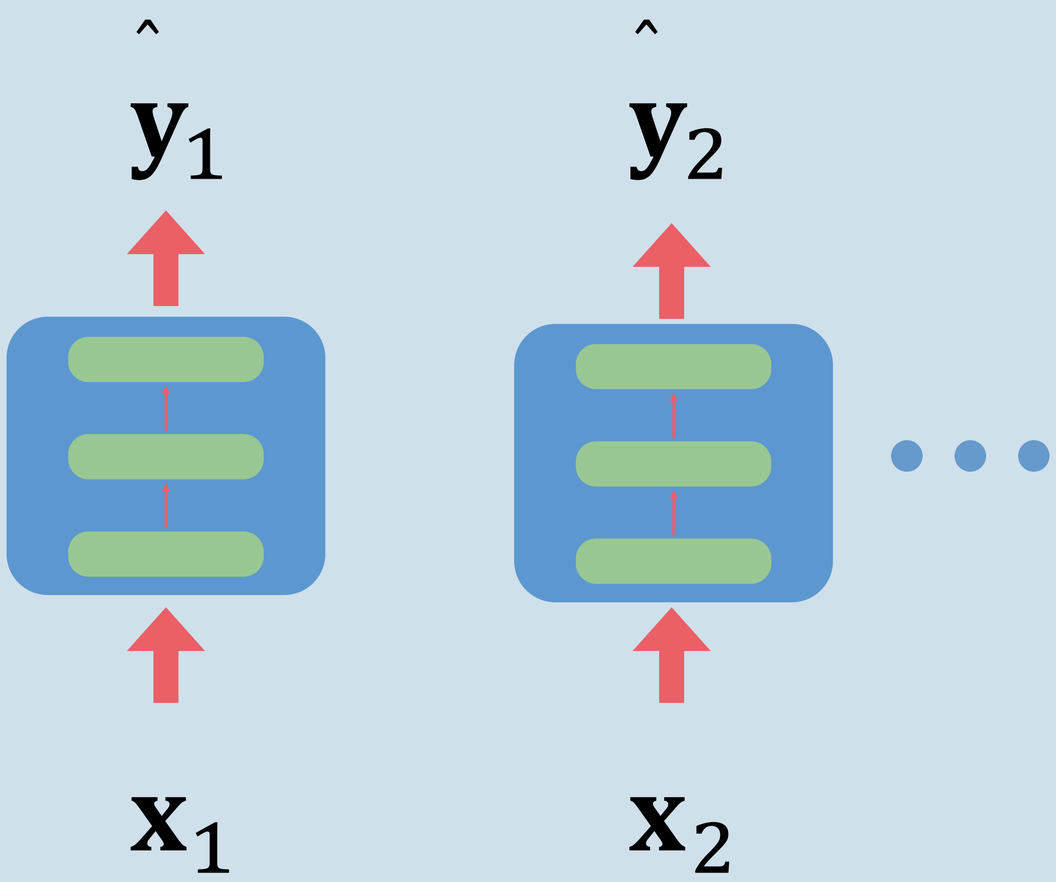
13.

有記憶的 RNN

遞歸神經網路 RNN

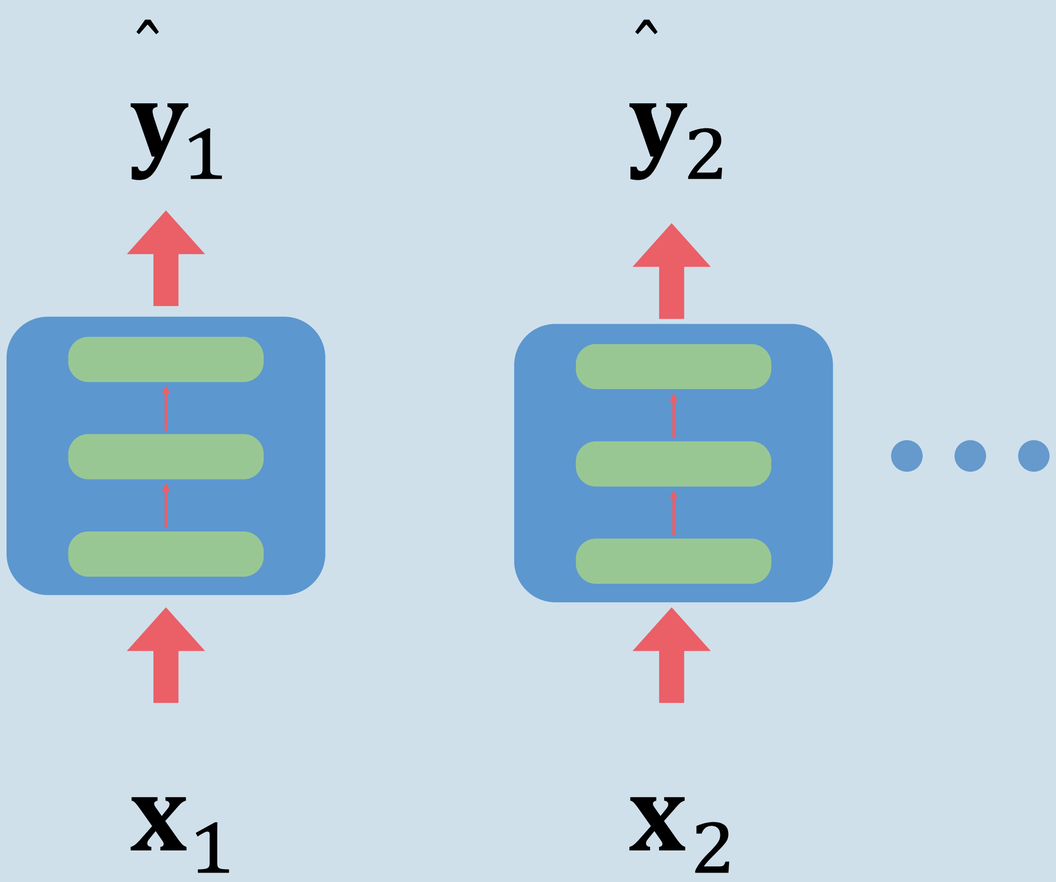
- Recurrent Neural Networks
- 有「記憶」的神經網路

RNN 的特色



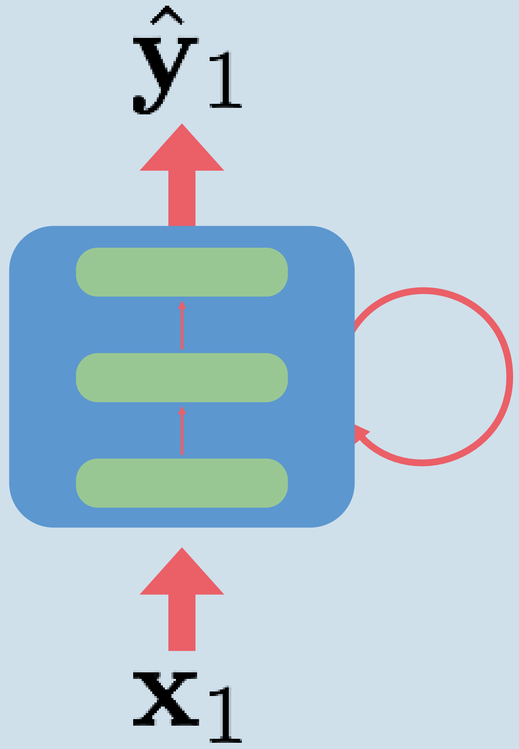
一般的神經網路一筆輸入
和下一筆是沒有關係的...

RNN 的特色



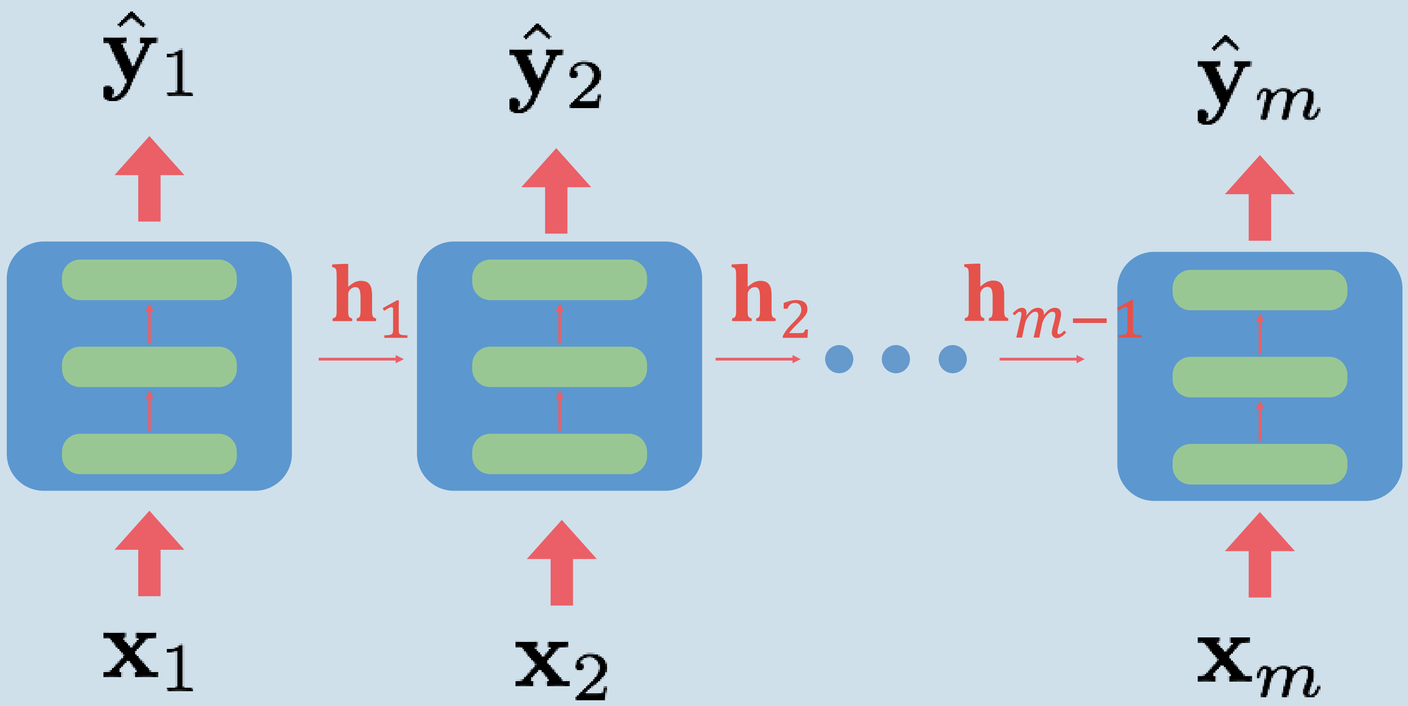
也就是說輸入的順序變了，輸出還是一樣的!

RNN 的特色



RNN 會偷偷把上一次的輸出也當這一次的輸入。

RNN 的特色



很多人畫成這樣, 也就是前面的資訊會傳遞下去。

傳遞的神秘資訊是放在 hidden states

$\mathbf{h}_i$

裡面!

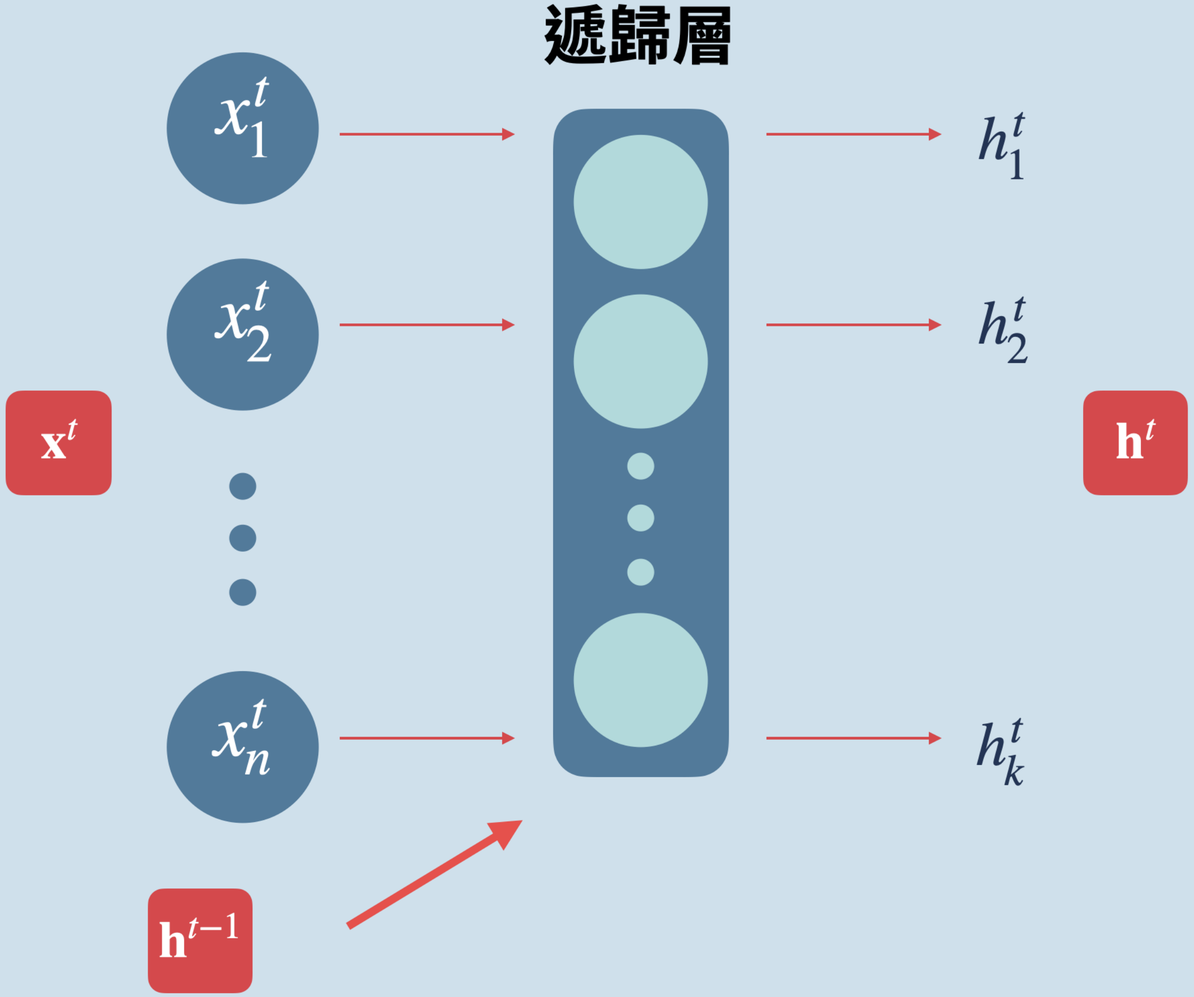
設計遞歸層

RNN 核心遞歸層, 我們基本上只要決定...

要用幾個 RNN 神經元!

這和 DNN 的狀況一樣, 實在太簡單了啊!

遞歸層原理

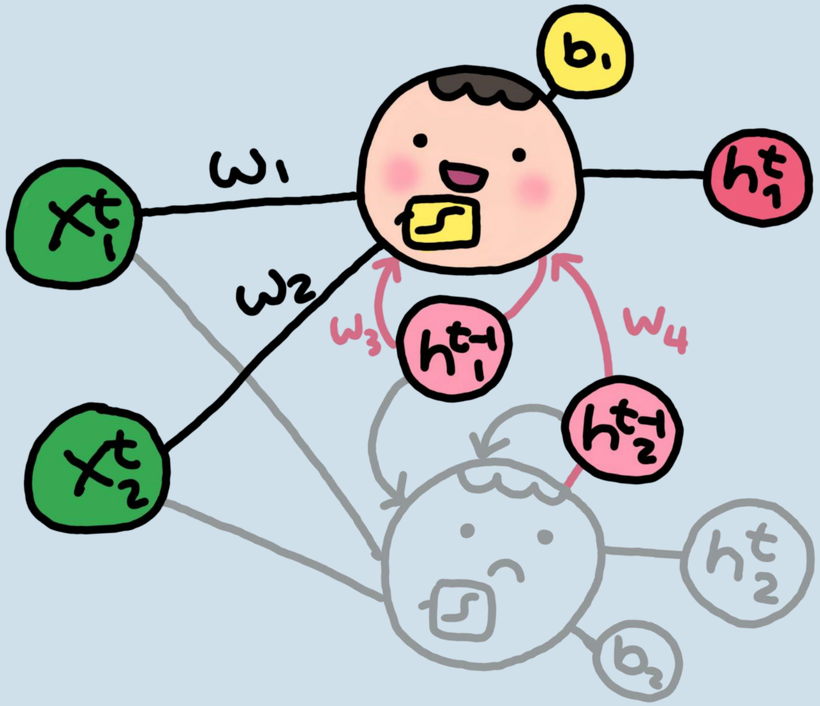


遞歸神經網路的特點是, 遞歸層每個神經元的輸出我們會收集起, 來成一個向量 $\mathbf{h}^t$, 我們叫 hidden state:

$$\mathbf{h}^t = \begin{bmatrix} h_1^t \\ h_2^t \\ \vdots \\ h_t^k \end{bmatrix}$$

這個 hidden state 向量下次會當成輸入的一部份。

遞歸層原理



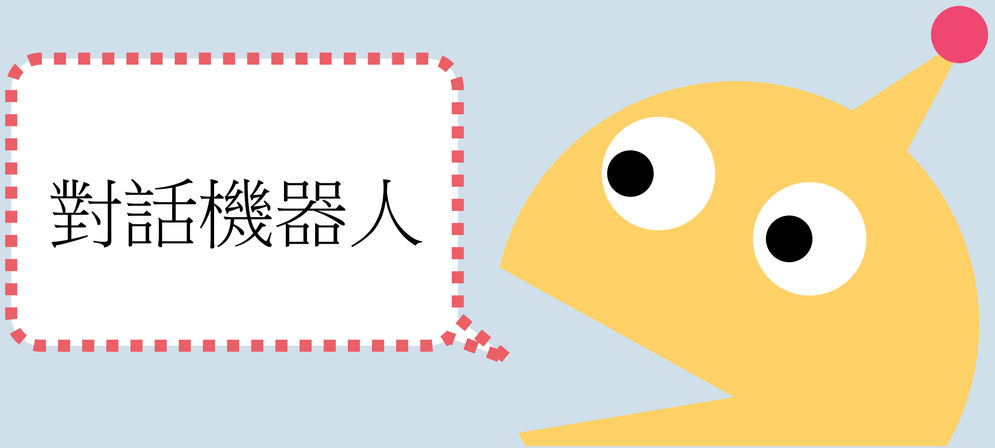
我們用有兩個輸入, 兩個 RNN 神經元的遞歸層來說明。

每一次遞歸層輸出的 2 維 hidden state, 下次會再回傳。

計算方式可以看出, 基本上就是 4 個輸入的標準全連結神經網路!

$$h_1^t = \sigma(w_1^X x_1^t + w_2^X x_2^t + w_1^H h_1^{t-1} + w_2^H h_2^{t-1} + b_1)$$

重要應用: 對話機器人

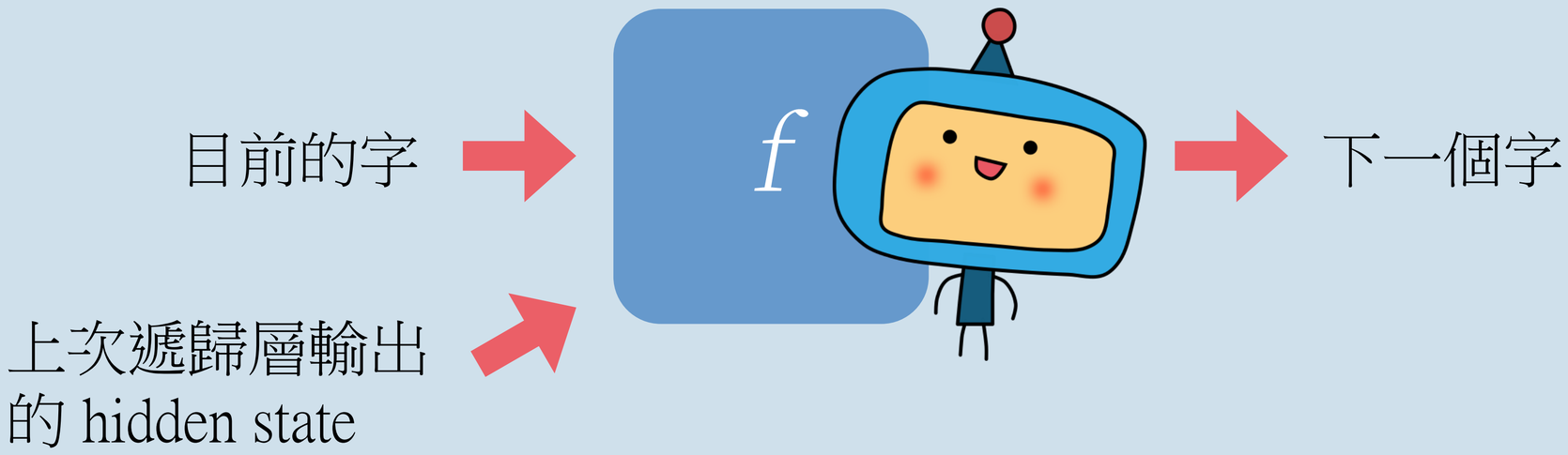


$$f(\text{目前的字}) = \text{下一個字}$$

RNN 最重要的應用之一, 大概就是可以用自然語言交談的對話機器人。

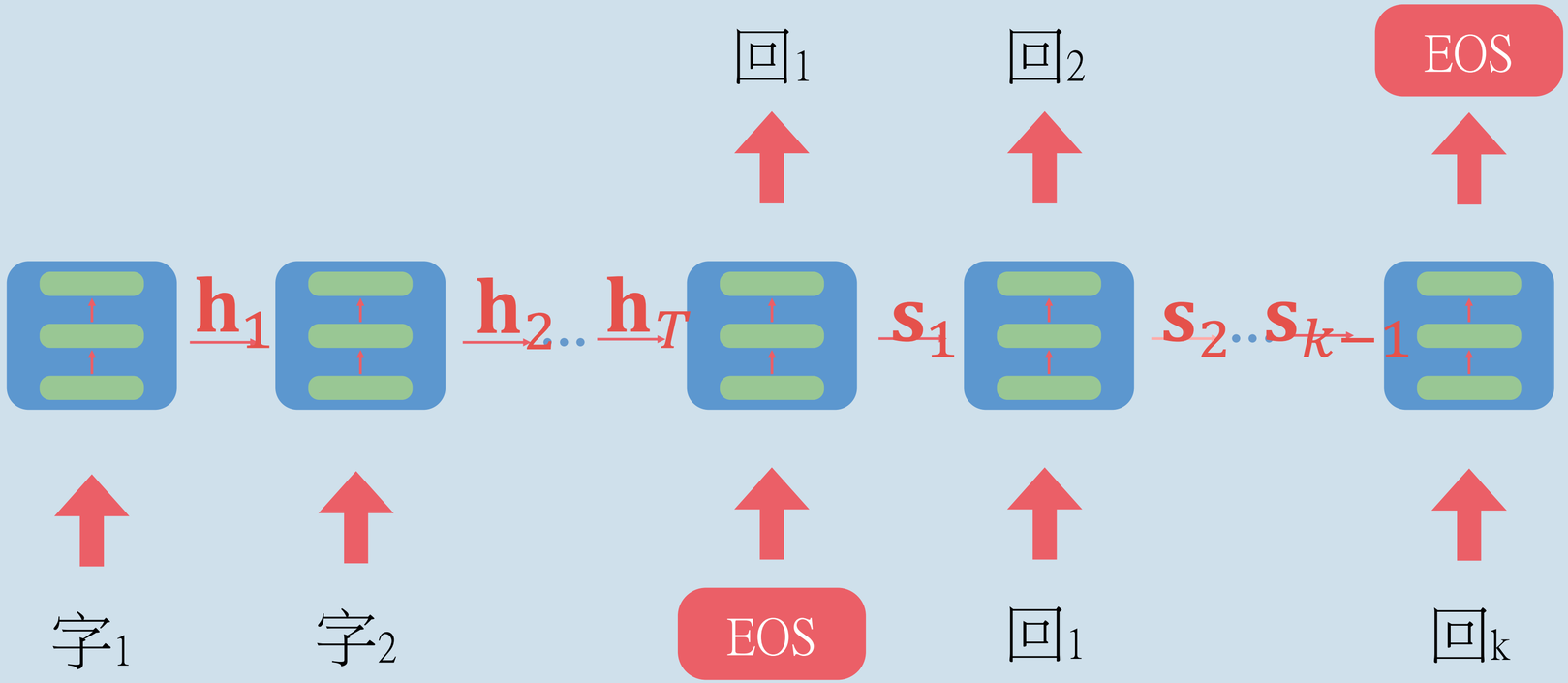
我們前面有說過, 函數學習機就是要用前一個字 (詞), 去預測下一個字 (詞)。

重要應用: 對話機器人

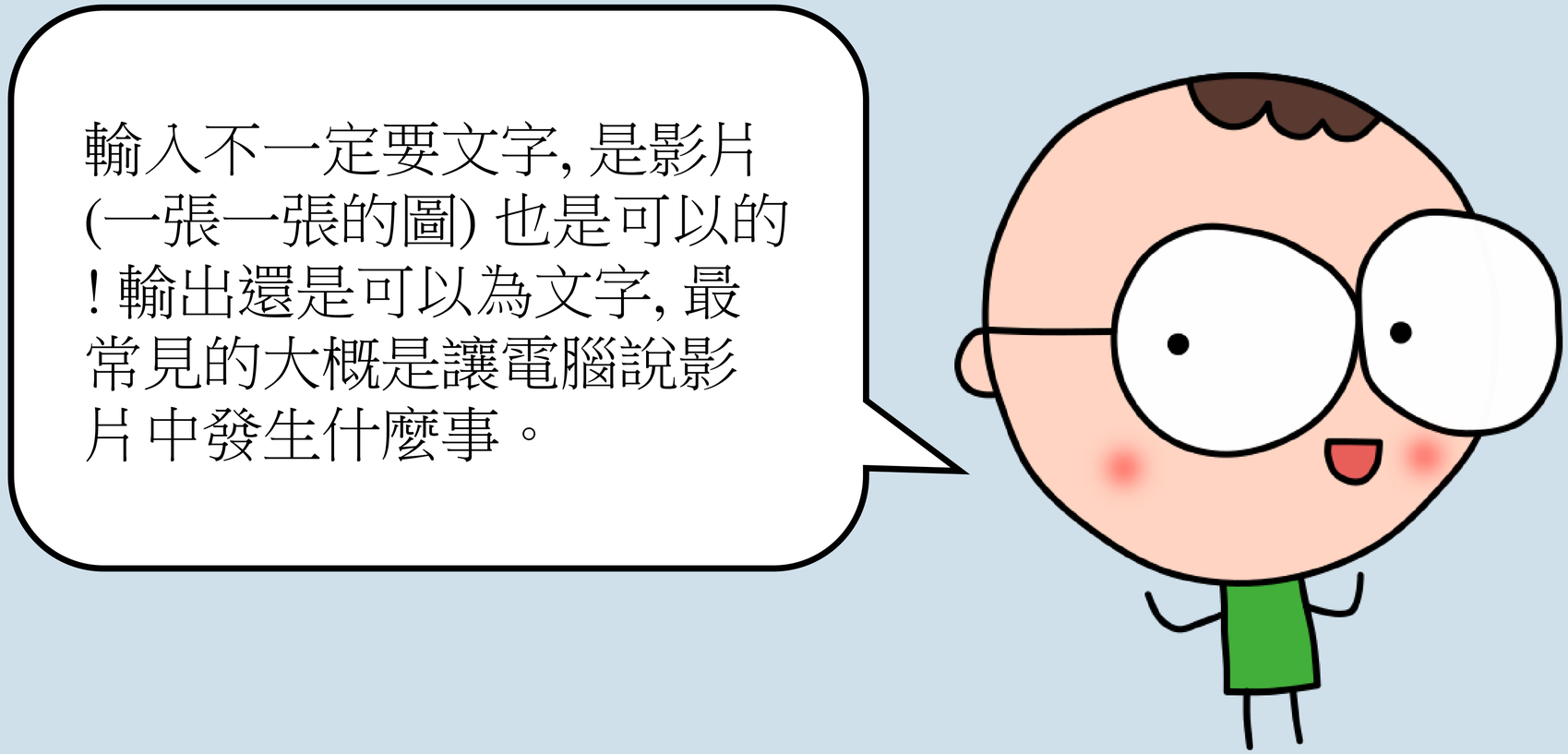


重要應用: 對話機器人

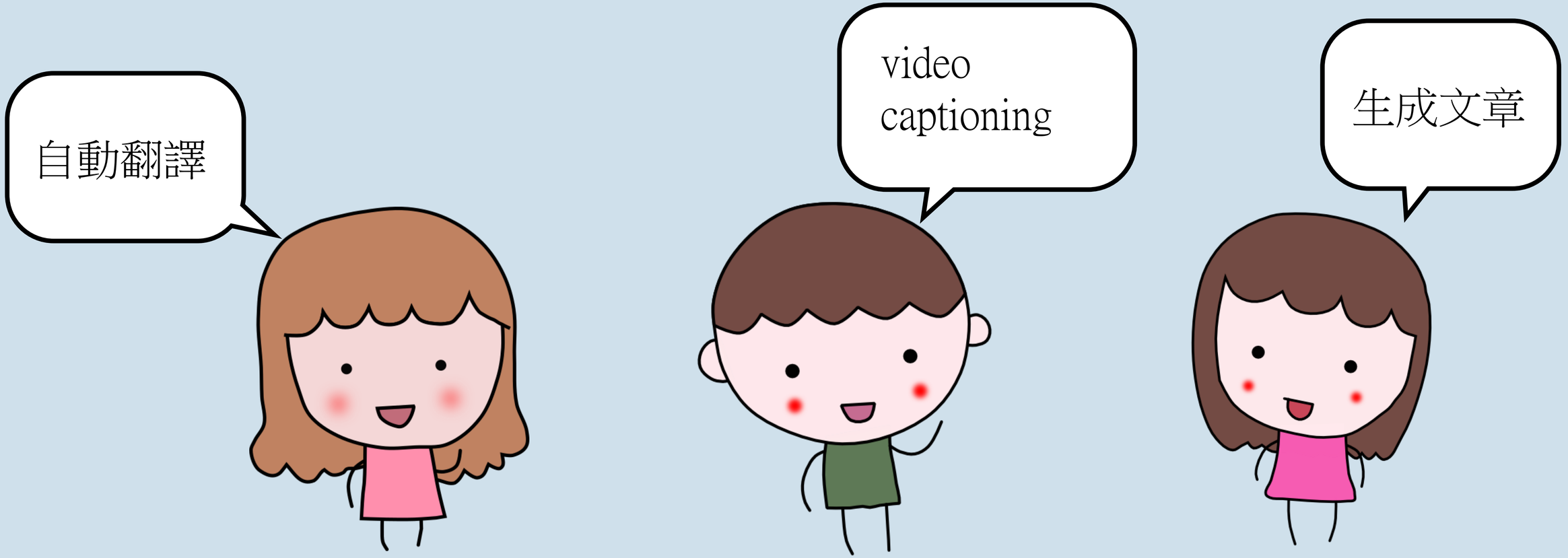
相信大家還記得, 對話機器人的原理。



其實對話機器人模式可以做更多!



其實對話機器人模式可以做更多!



AI 數學家

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ fo
$i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-modu
In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

Andrej Karpathy 生出代數幾何介紹 "Stacks" 的文字!

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

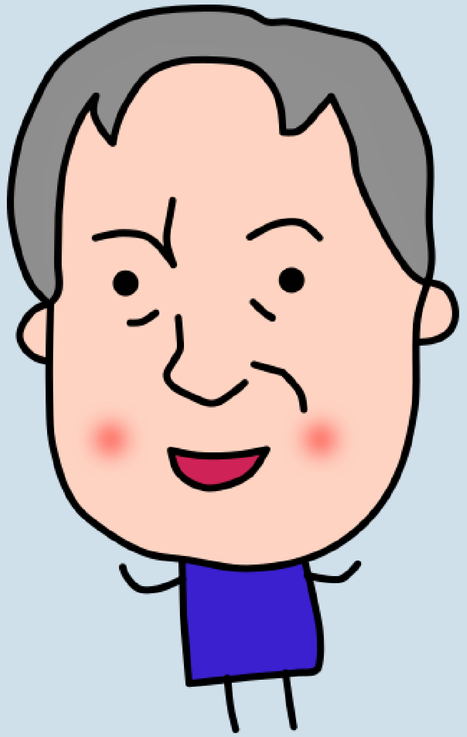
AI 莎士比亞

潘達洛斯：
唉，我想他應該過來接近一天
當小的小麥變成從不吃的時候，
誰是他的死亡鏈條和臣民，
我不應該睡覺。

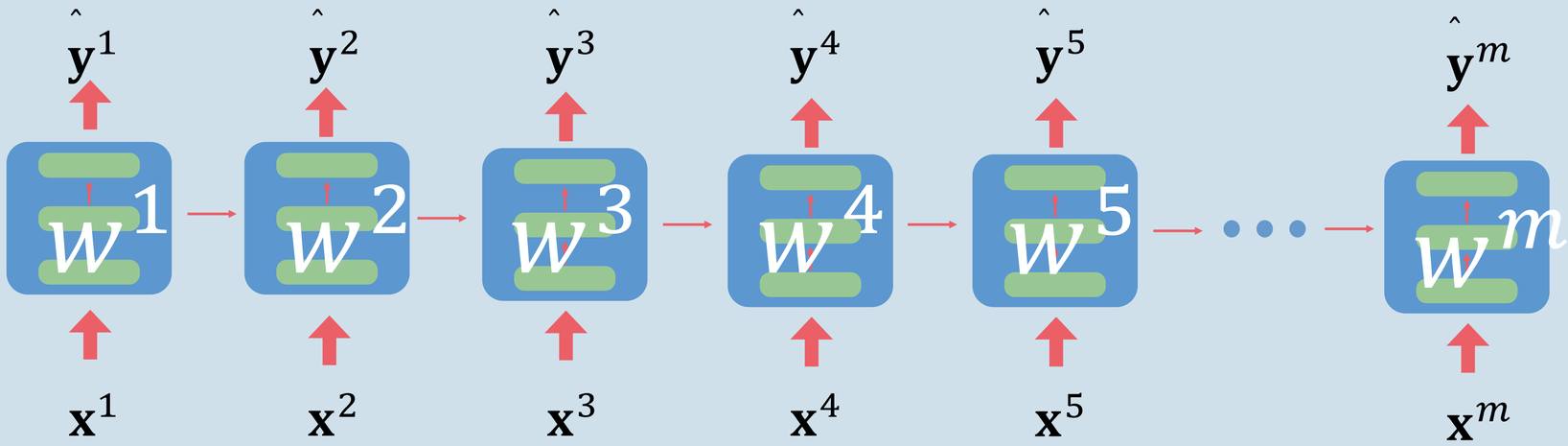
第二位參議員：
他們遠離了我心中產生的這些苦難，
當我滅亡的時候，我應該埋葬和堅強
許多國家的地球和思想。

RNN 的訓練 BPTT

RNN 的訓練有個很炫的名字, 叫 backpropagation through time (BBPT), 其實和一般神經網路的訓練沒有什麼不同!

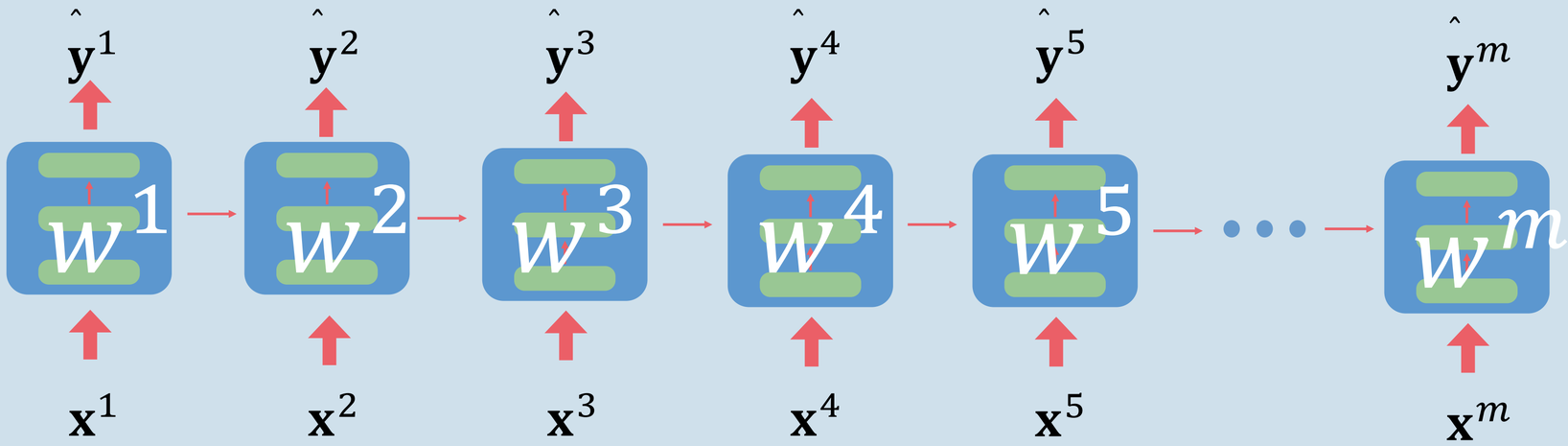


RNN 的訓練 BPTT



我們假設我們的 RNN 函數學習機有個參數叫 w, 記得每個時間點都有這個參數在。雖然是同一個參數, 但依時間不同叫 w^t。

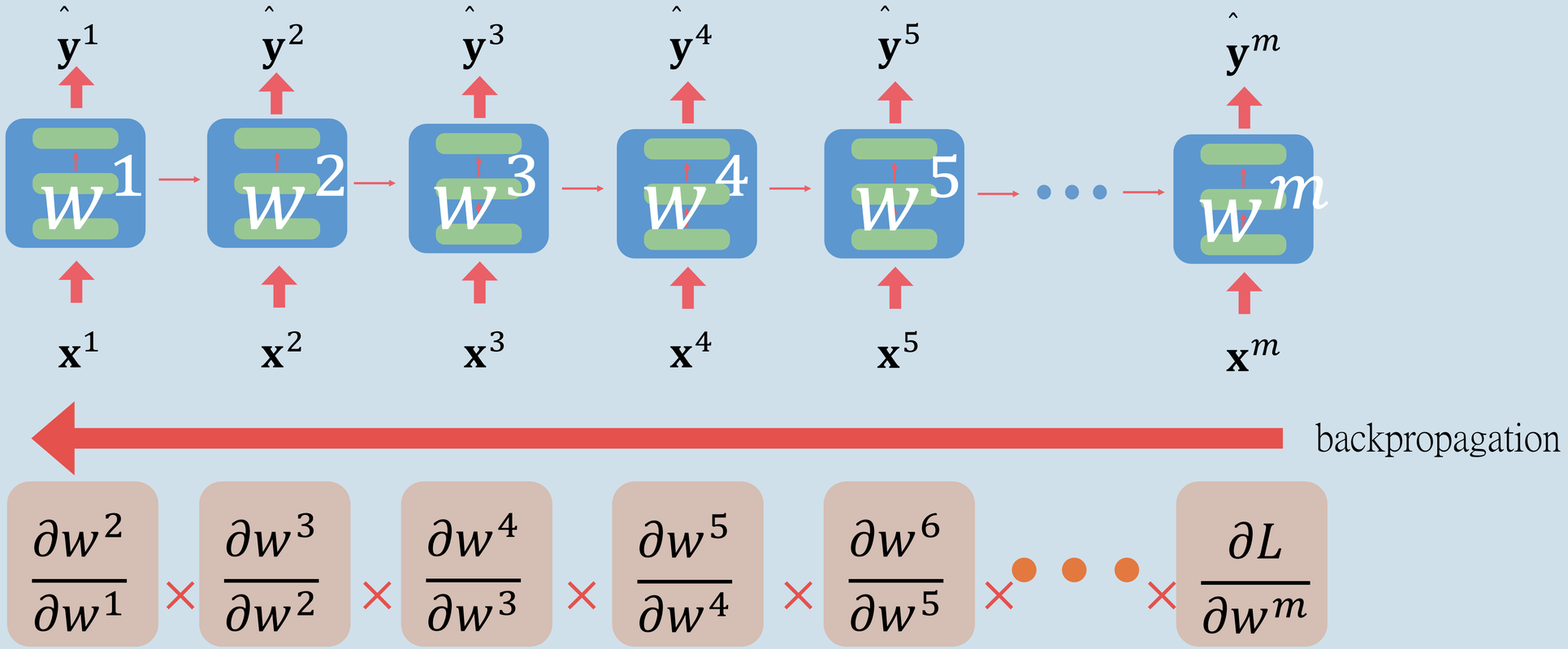
RNN 的訓練 BPTT



訓練的時間我們先把整個過程看成一個很深的神經網路, 把每個時間點的 w^t 看成不同的參數去調整。每一個參數要調多少再平均, 就得到 w 要調整的大小。

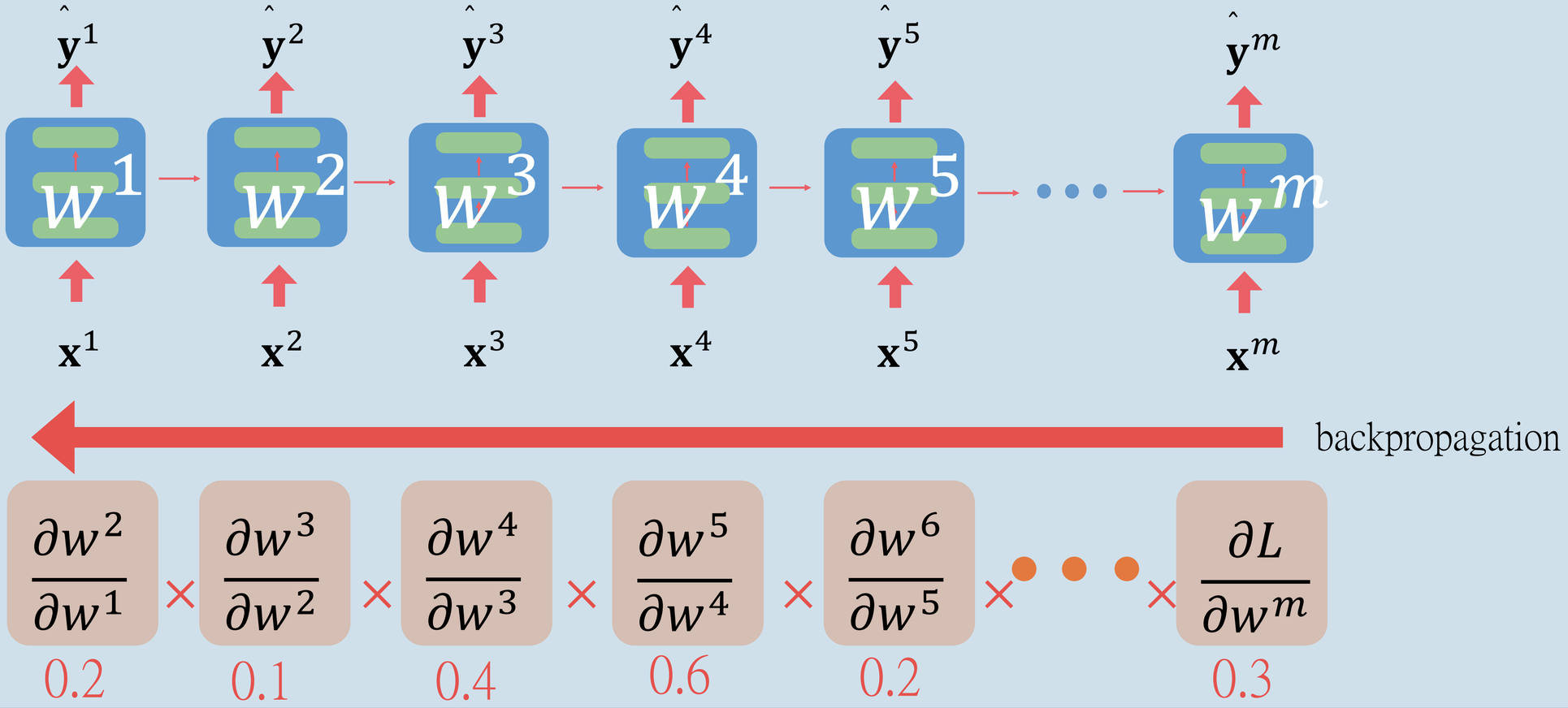
RNN 訓練上的問題

RNN 自然變成「很深的神經網路」成為一個訓練上的問題!



RNN 訓練上的問題

在做 backpropagation 時, 還沒乘到最前面, 往往就變成 0 了! 這個問題叫梯度消失 (vanishing gradient)!



RNN 兩大救星

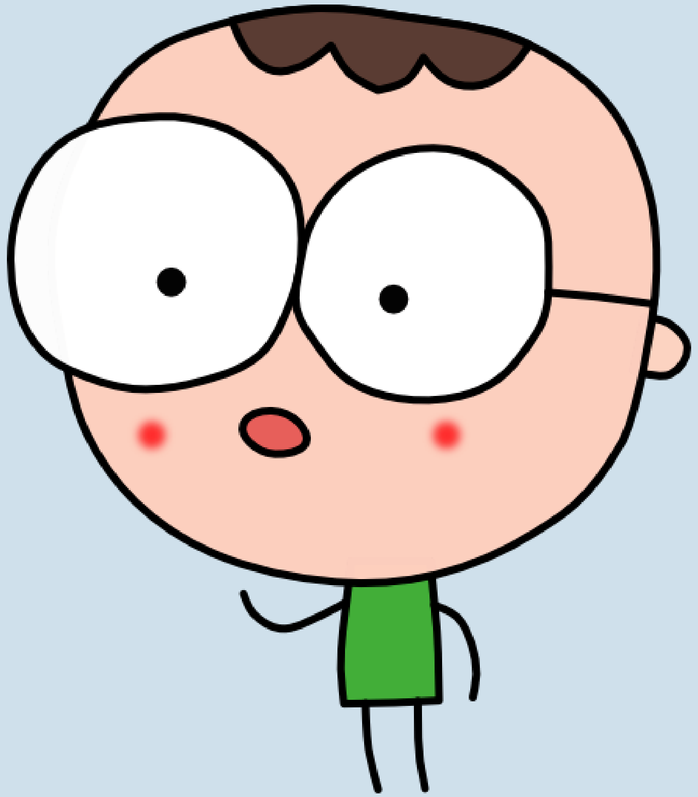
LSTM Long Short Term Memory

Gated Recurrent Unit **GRU**

說用 RNN, 其實是用 LSTM/GRU

現在說到 RNN, 其實包括原始 RNN, LSTM, GRU 等各種變形。

特別要叫原始的 RNN, 我們習慣叫它 Vanilla RNN, 在 TensorFlow 中是 **`SimpleRNN`**。

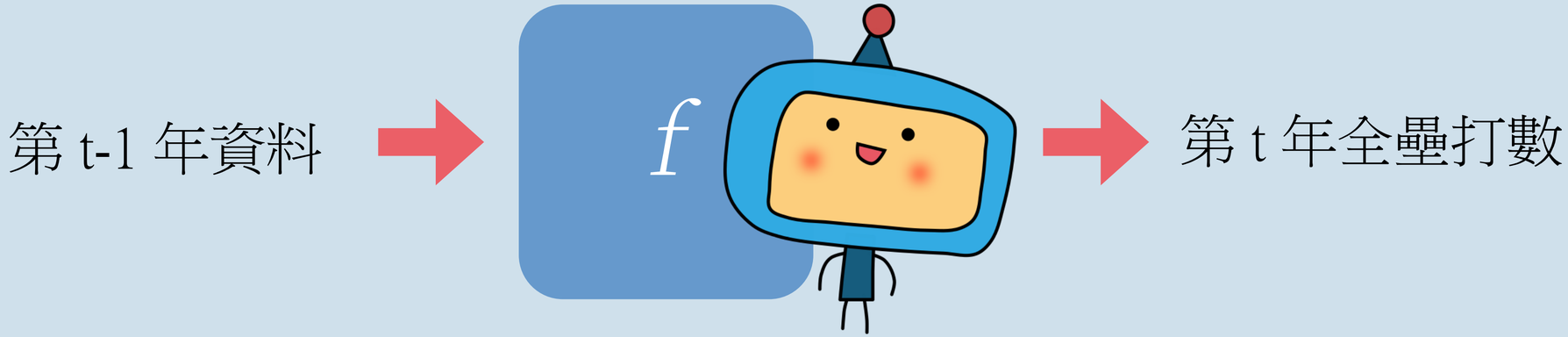


RNN 預測全壘打數

RNN 預測全壘打數

輸入有 15 個 features:

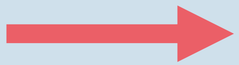
```
[Age, G, PA, AB, R, H, 2B, 3B, HR,
 RBI, SB, BB, SO, OPS+, TB]
```

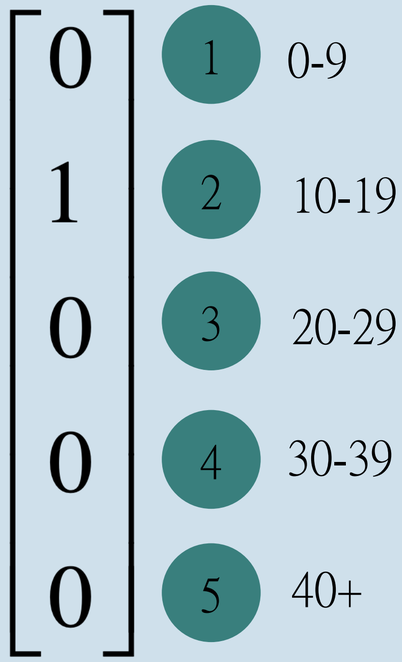


輸出是分五個區間: 0-9, 10-19, 20-29, 30-39, 40+

RNN 預測全壘打數

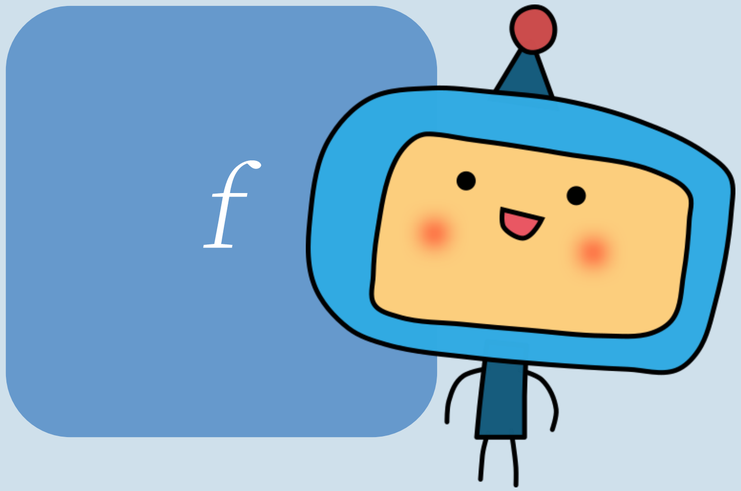
10-19





輸出一樣做 one-hot encoding。

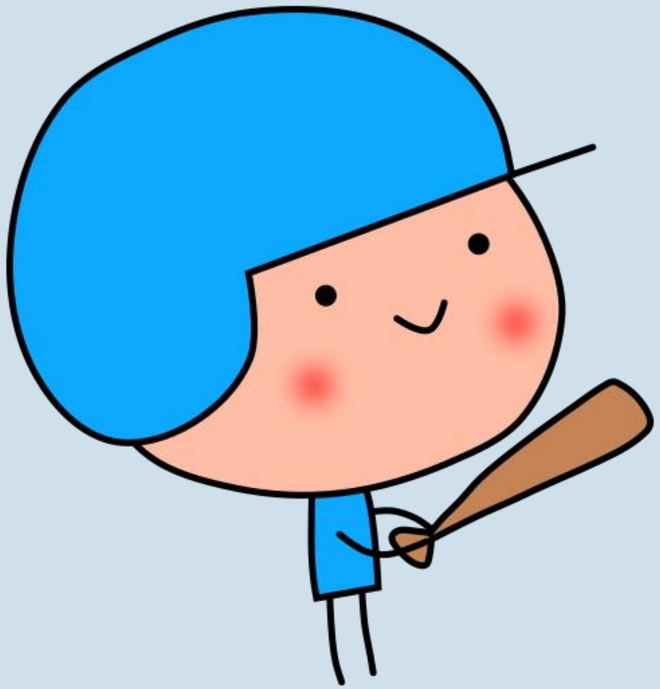
RNN 預測全壘打數



神經網路函數學習機的設計:

- 輸入 15 維向量, 輸出 5 維向量 (區間)。
- 一層 LSTM 隱藏層。
- 輸出層做 softmax。
- 訓練時每次用 10 年資料。

RNN 預測全壘打數



2017 預測結果

(2017 年 6 月預測)

Mike Trout (LAA)
預測 30-39
實際 33

Mookie Betts (BOS)
預測 20-29
實際 24

Jose Altuve (HOU)
預測 20-29
實際 24

Kris Bryant (CHC)
預測 30-39 (第二高 20-29)
實際 29

Daniel Murphy (WSH)
預測 20-29
實際 23

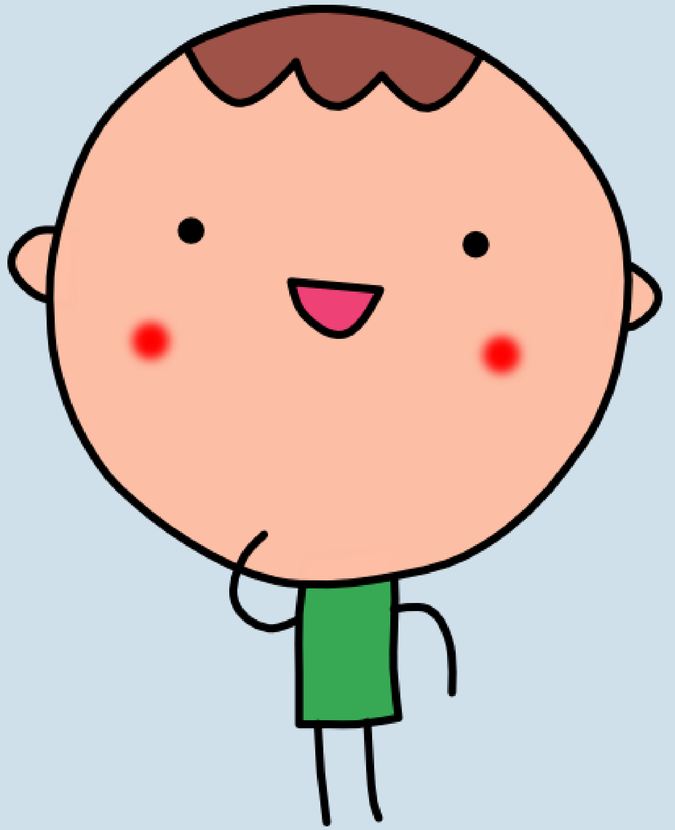
Corey Seager (LAD)
預測 20-29
實際 22

14.

生成模式和特徵截取

Generative Models

為何要討論這個呢?

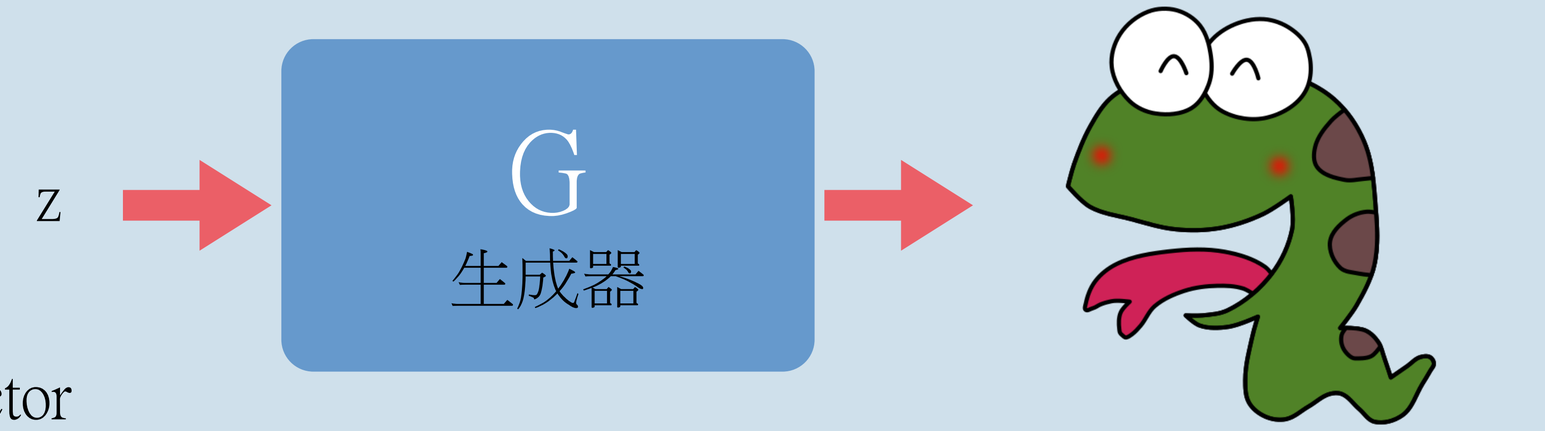


討論生成模式一定要引費曼的話

“What I cannot create,
I do not understand.”

—Richard Feynman

生成模式



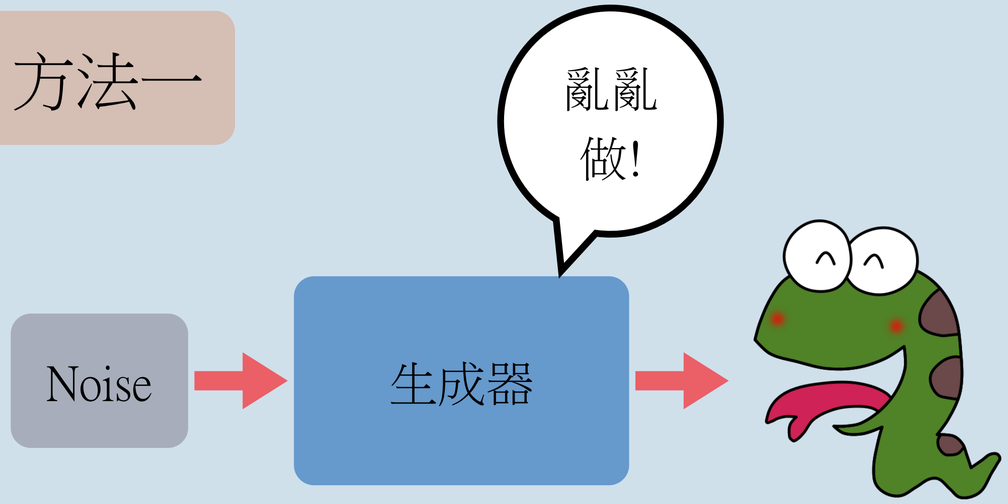
latent vector

特徵向量/本徵向量

生出圖、照片、文字

輸入的特徵向量基本上有兩種作法

方法一



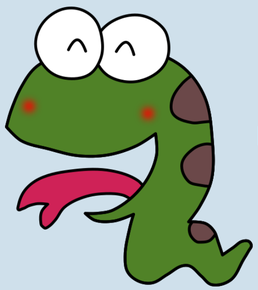
生出圖、照片、文字

(保證生出「正確格式」的東西就好)

在 GAN 裡, 這是最常用的方法!

輸入的特徵向量基本上有兩種作法

方法二



Encoder

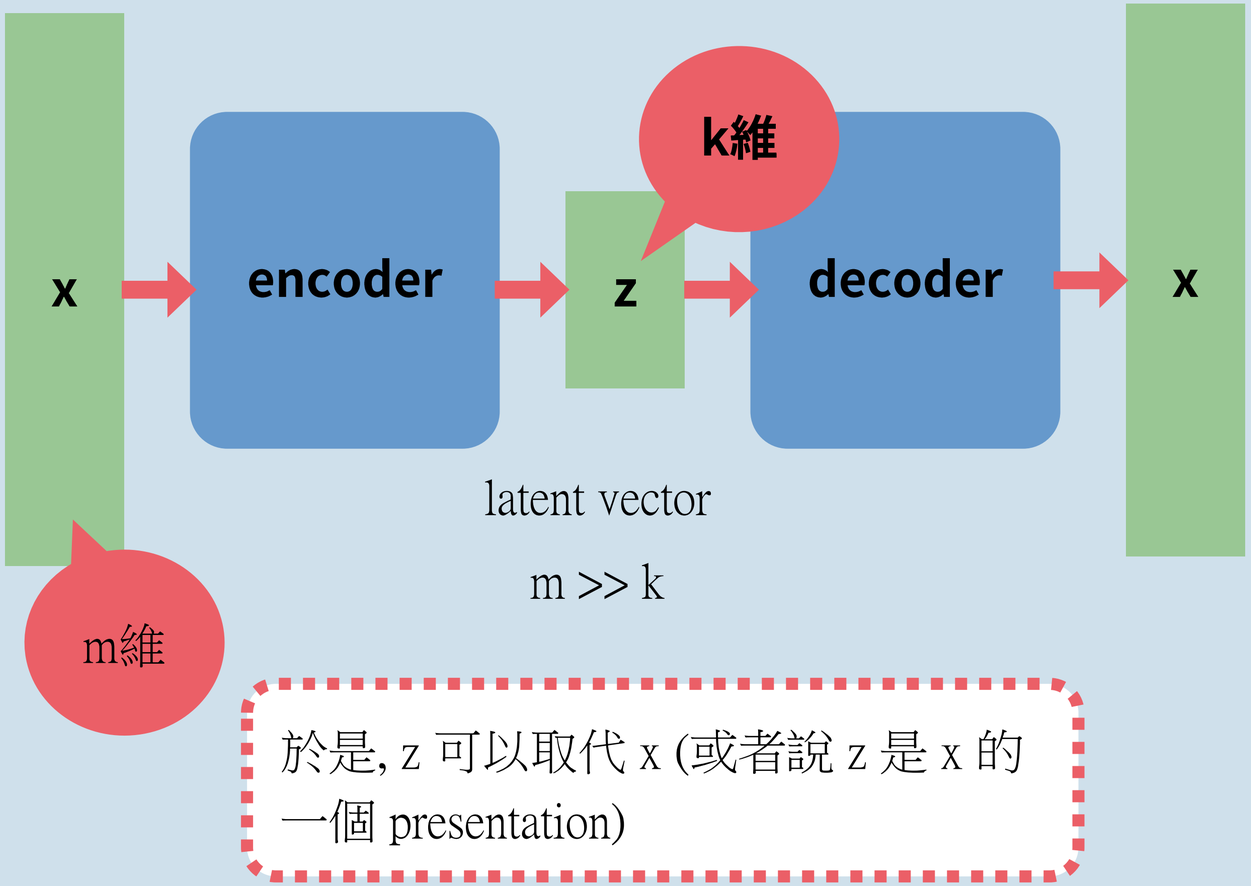
特徵向量

Autoencoder

方法二看來有點神奇, 怎麼能訓練一個截取特徵向量的函數的? 這裡介紹一個常用手法叫自編碼器 (Autoencoder)。

Autoencoder

Autoencoder

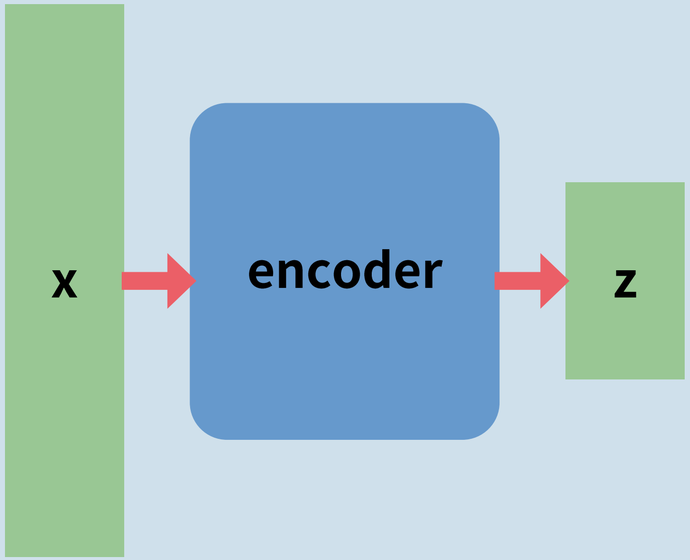


Autoencoder 是輸入什麼, 就輸出什麼的函數。

感覺很奇怪, 原來是中間我們會用一層比較小的 k 維神經元。

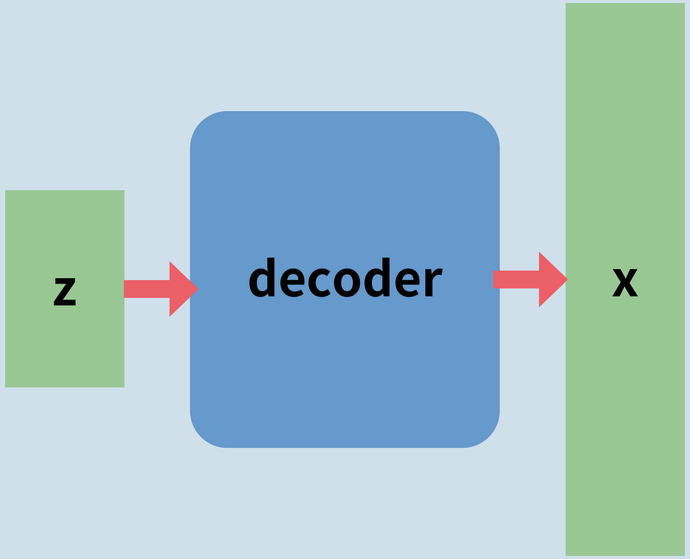
這一層的輸出就是我們準備當特徵向量的。

Autoencoder



以後我們基本上可以用比較小的 z 來取代比較大的 x。

Autoencoder



另一邊就可以當生成器, 輸入一個特徵向量, 就生出一個我們要的圖或任何東西。

答案是不太行

隨便生兩個 laten vectors, 數學上距離很近, 但生出來的東西不一定有什麼關係。

白話文是 z 差不多就是亂數, 我們無以掌控。

所以有了 VAE

Variational AutoEncoder

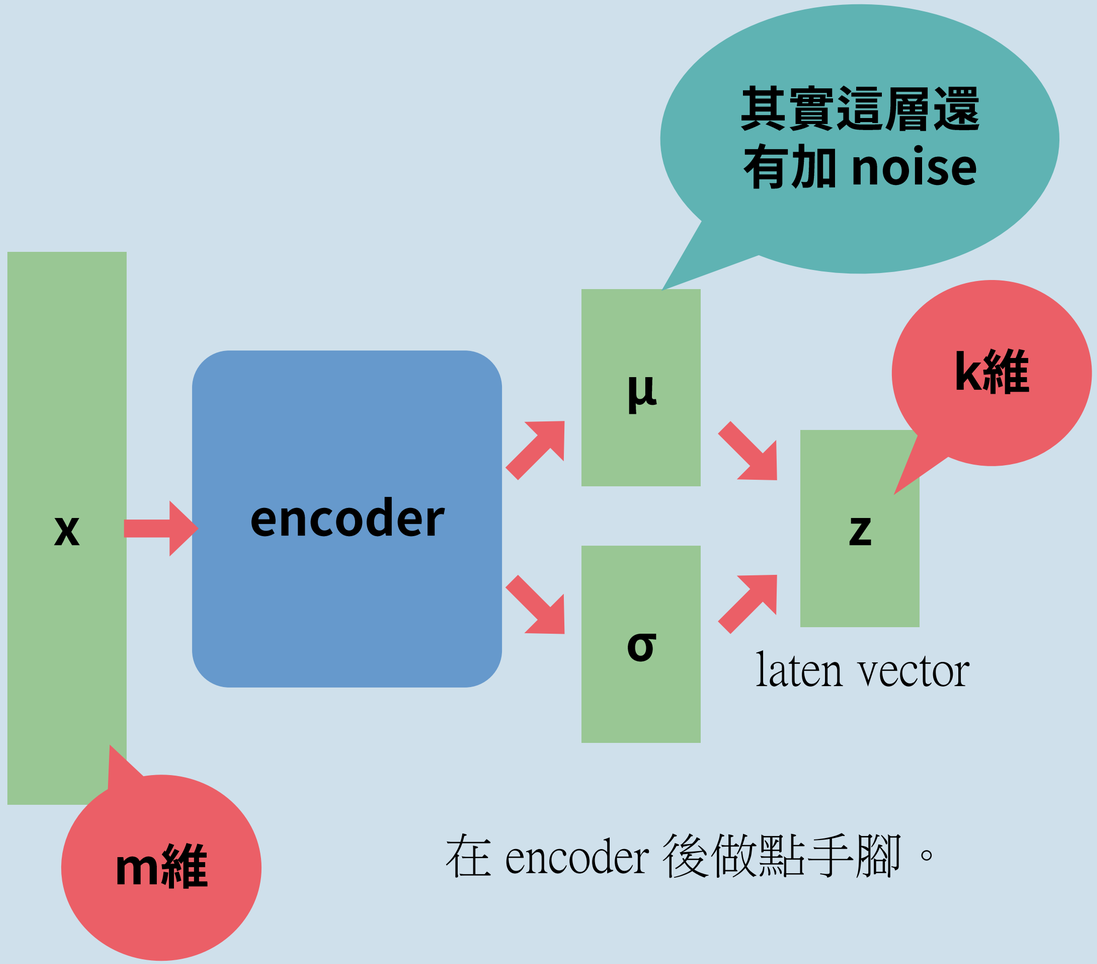
VAE (Variational AutoEncoder)

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix} \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

我們想辦法找每個數的平均值和變異數 (or 標準差)

神秘編碼 laten vector 每個數字是符合某常態分布的, 這樣我們容易掌控!

VAE (Variational AutoEncoder)



在 encoder 後做點手腳。

15.

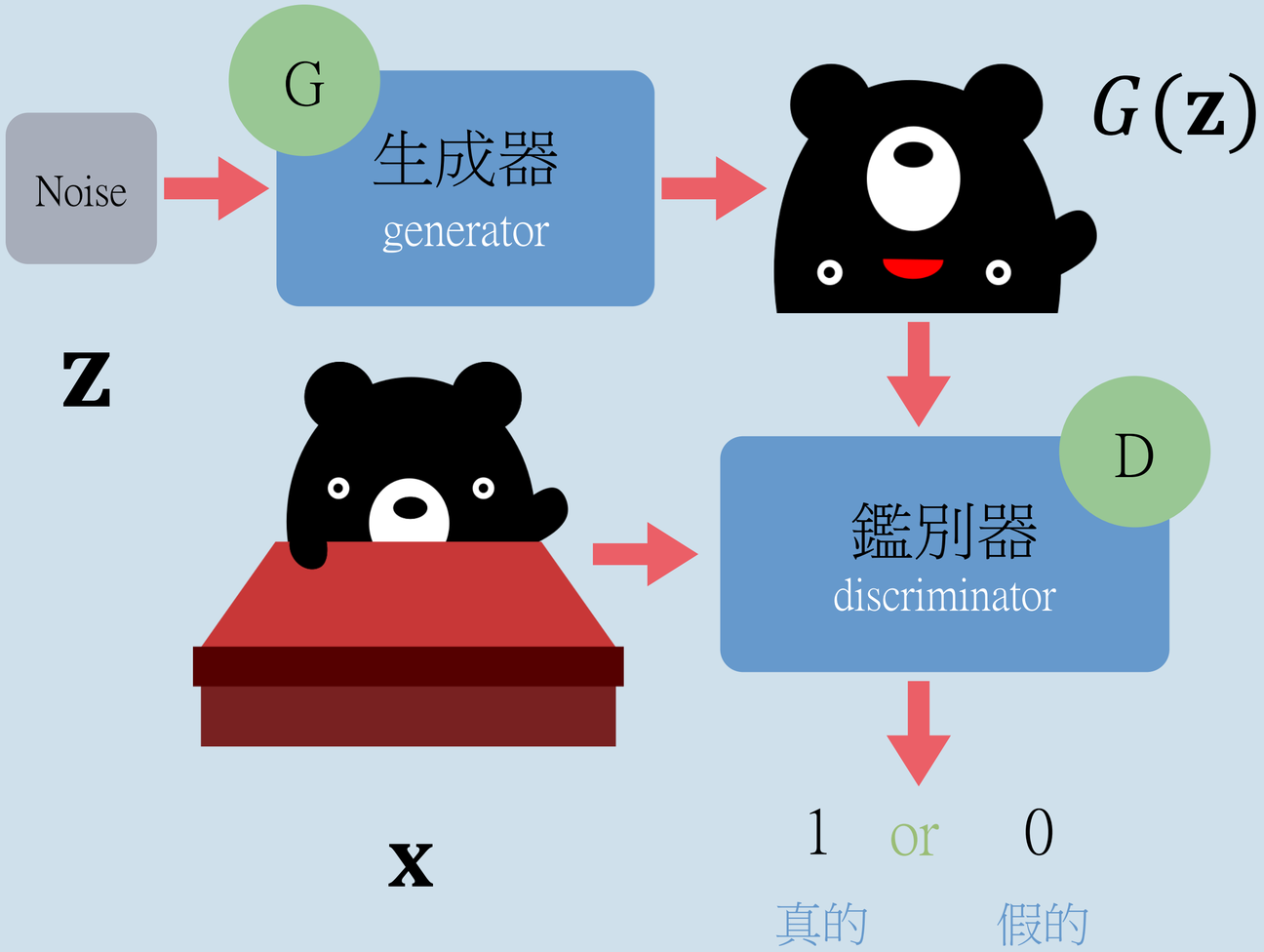
生成對抗網路 GAN

LeCun 認為 GAN 是深度學習最有潛力的 model

> There are many interesting recent development in deep learning...
>
> The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks).

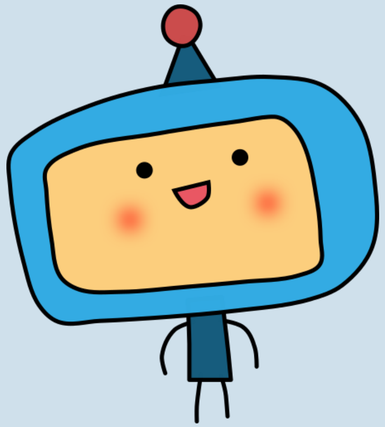
—Yan LeCun (楊立昆), 2016

GAN 的架構



GAN 是兩個神經網路, 一個叫生成器、一個叫鑑別器, 相互對抗!

GAN 就是生成器、鑑別器大對抗!



生成器 G

希望

$D(G(\mathbf{z}))$

接近 1

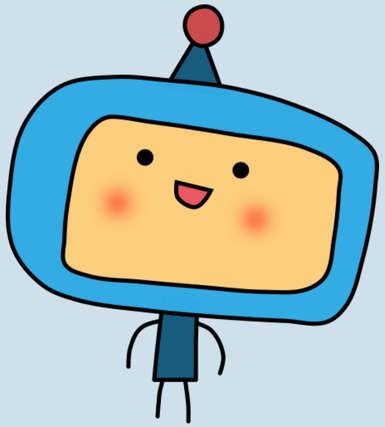
希望

$D(G(\mathbf{z}))$

接近 0

$D(\mathbf{x})$

接近 1



鑑別器 D

GAN 的著名應用

iGAN 人人都是大畫家!

人人都是大畫家!
https://youtu.be/9c4z6YsBGQ0

iGAN

朱俊彦等人 (ECCV 2016)
https://arxiv.org/abs/1609.03552

“Generative Visual Manipulation on the Natural Image Manifold”

Progressive GAN 生成明星照片

Karras-Aila-Laine-Lehtinen

Progressive Growing of GANs for Improved Quality, Stability, and Variation

- NVIDIA 團隊很有名的文章
- 當初用現在基本上沒有在用的 Theano, Python 2, 而且只有單 GPU

Progressive GAN

Karras 等 NVIDIA 團隊 (ICLR 2018)

https://arxiv.org/abs/1710.10196

“Progressive Growing of GANs for Improved Quality, Stability, and Variation”

Progressive GAN 生成明星照片

這攏係假ㄟ啦 (1024x1024 明星照)

Pix2pix: 簡單畫畫就產生很真實的相片

Pix2pix 把衛星圖變地圖。

* 來自 Isola, 朱俊彦等人的原始論文 (2017)

Pix2Pix

Isola, 朱俊彦等人 (CVPR 2017)

https://arxiv.org/abs/1611.07004

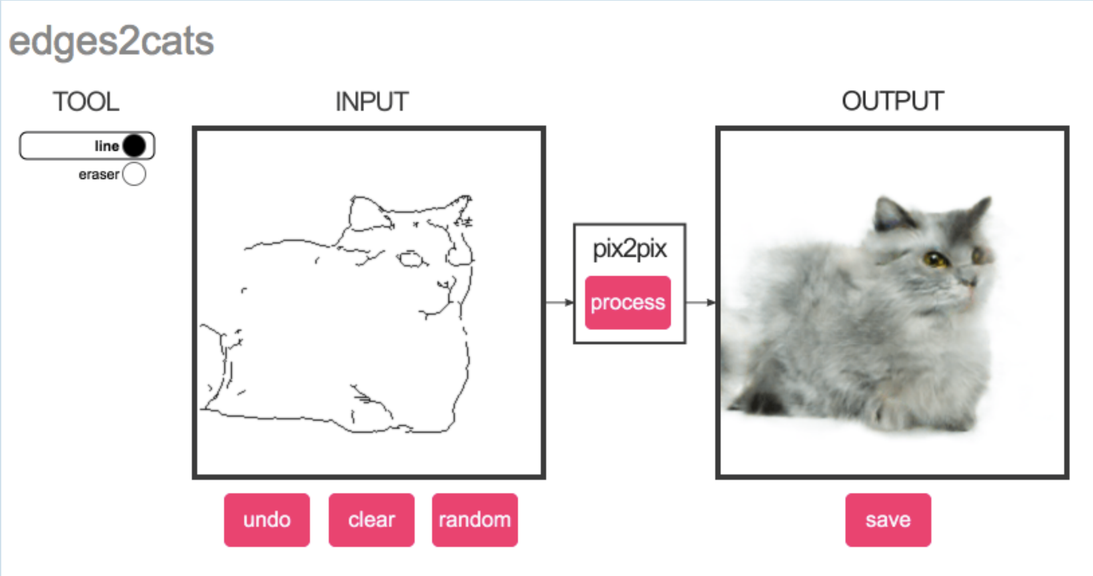
“Image-to-Image Translation with Conditional Adversarial Networks”

Pix2pix: 簡單畫畫就產生很真實的相片

Pix2pix 隨手畫畫變街景。

* 來自 Isola, 朱俊彥等人的原始論文 (2017)

Pix2pix: 簡單畫畫就產生很真實的相片



Pix2pix 線上版。

https://affinelayer.com/pixsrv/

* Christopher Hesse 依原論文做出

CycleGAN

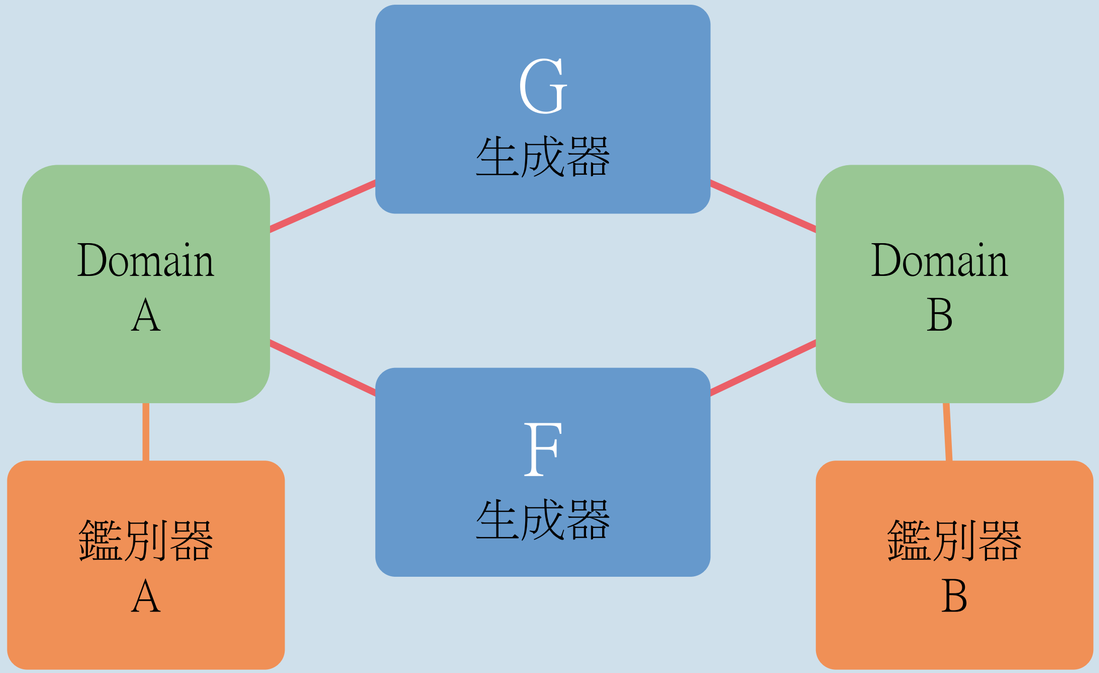
朱俊彥等人 (ICCV 2017)

https://arxiv.org/abs/1703.10593

“Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”

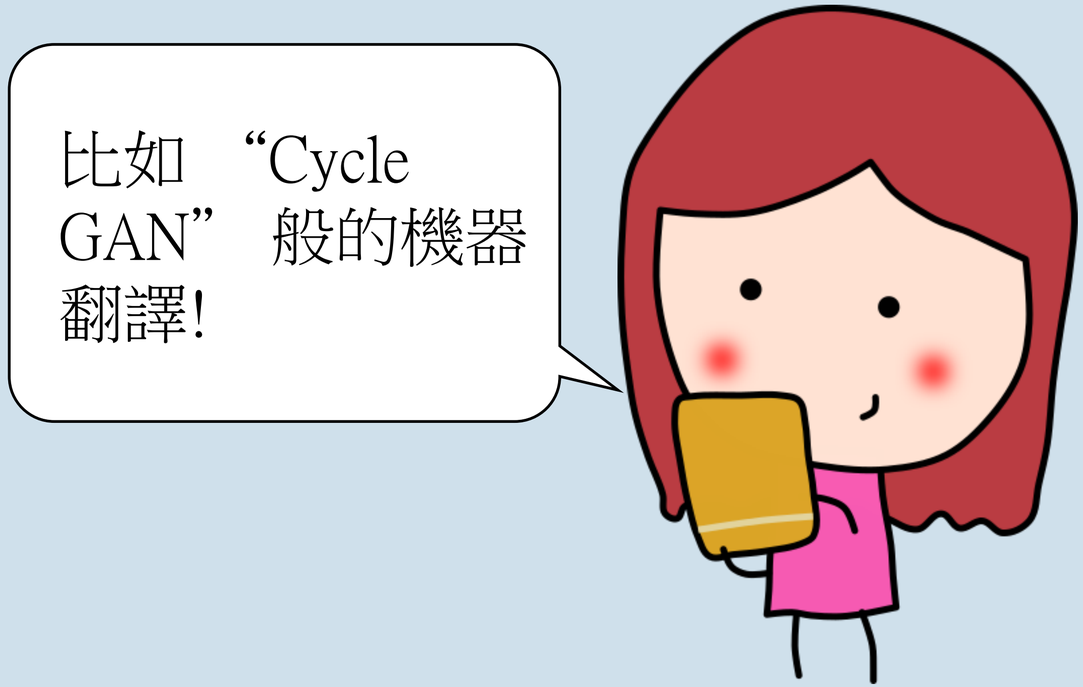
CycleGAN: 讓人驚呆的魔法

CycleGAN



資料「不需要」配對!

於是有無限可能...



CycleGAN: 讓人驚呆的魔法

全世界的人都驚呆了的馬變斑馬。

https://youtu.be/9reHvktowLY

CycleGAN: 讓人驚呆的魔法

CycleGAN 作者群幽自己一默的失敗例子。

CycleGAN: 讓人驚呆的魔法

Goodfellow 也關注的館長變陳沂 (魏澤人老師)。

https://youtu.be/Fea4kZq0oFQ

16.

RL 強化學習

強化學習 Reinforcement Learning

打造 AlphaGo 的神奇魔法

重要實例: AI 玩電動

Deep Q-Learning

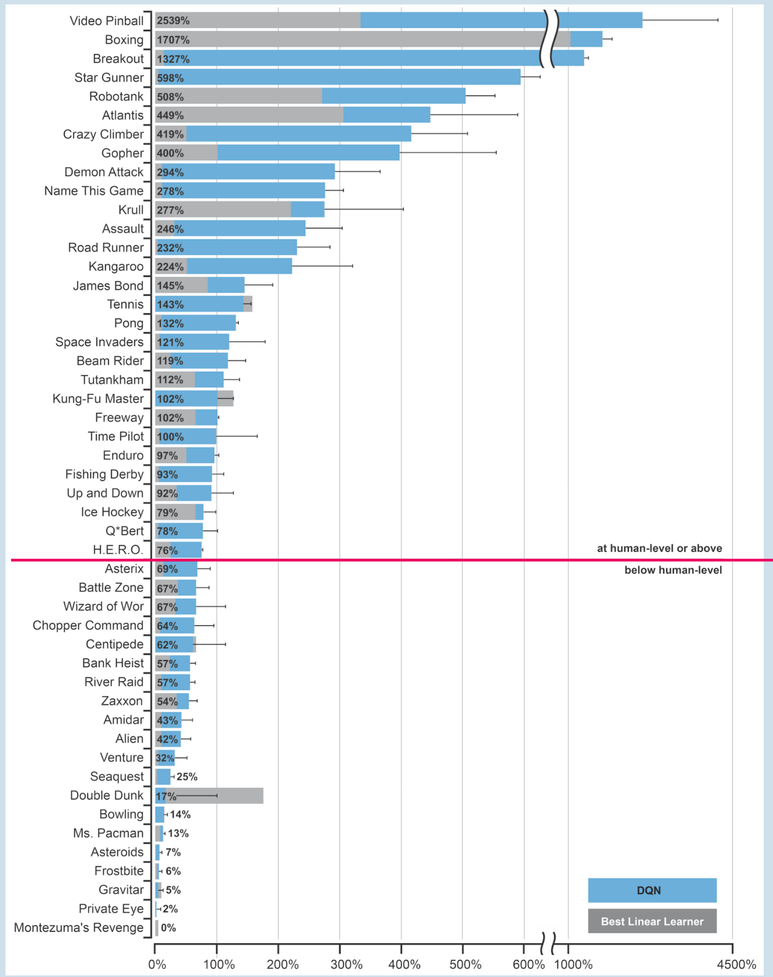
DeepMind

2015 年 Nature 出現一篇

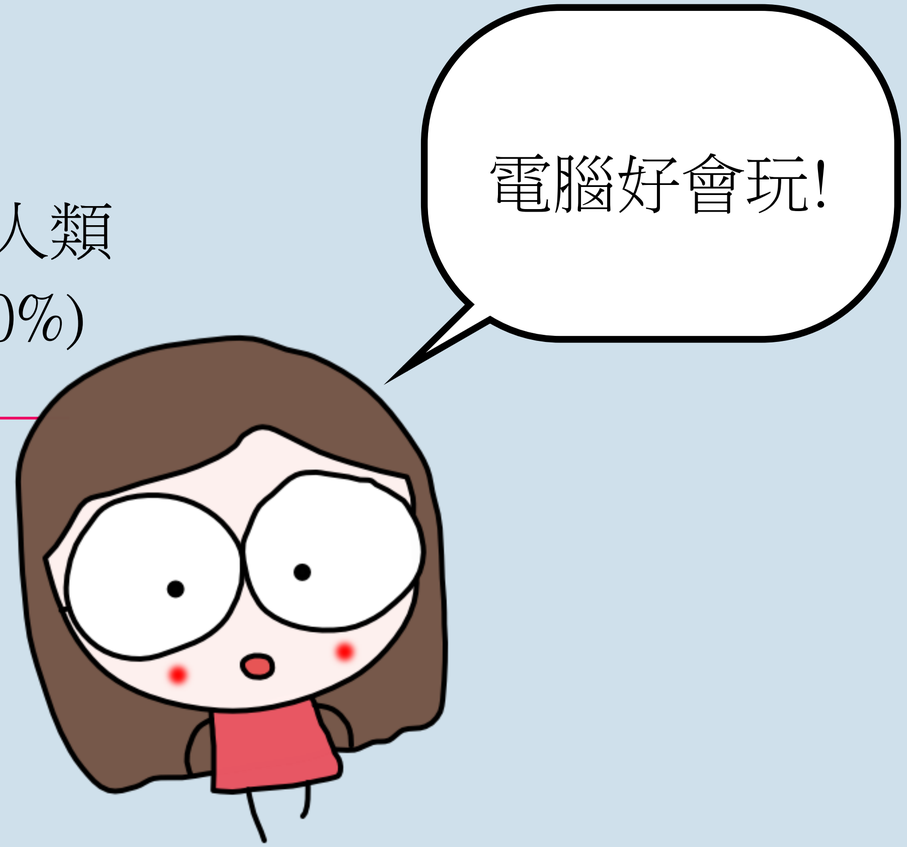
> **“Human-level Control Through Deep Reinforcement learning”**

為題的論文, 基本上就是教電腦玩 Atari 的遊戲。

重要實例: AI 玩電動



AI 勝過人類
(超過 50%)



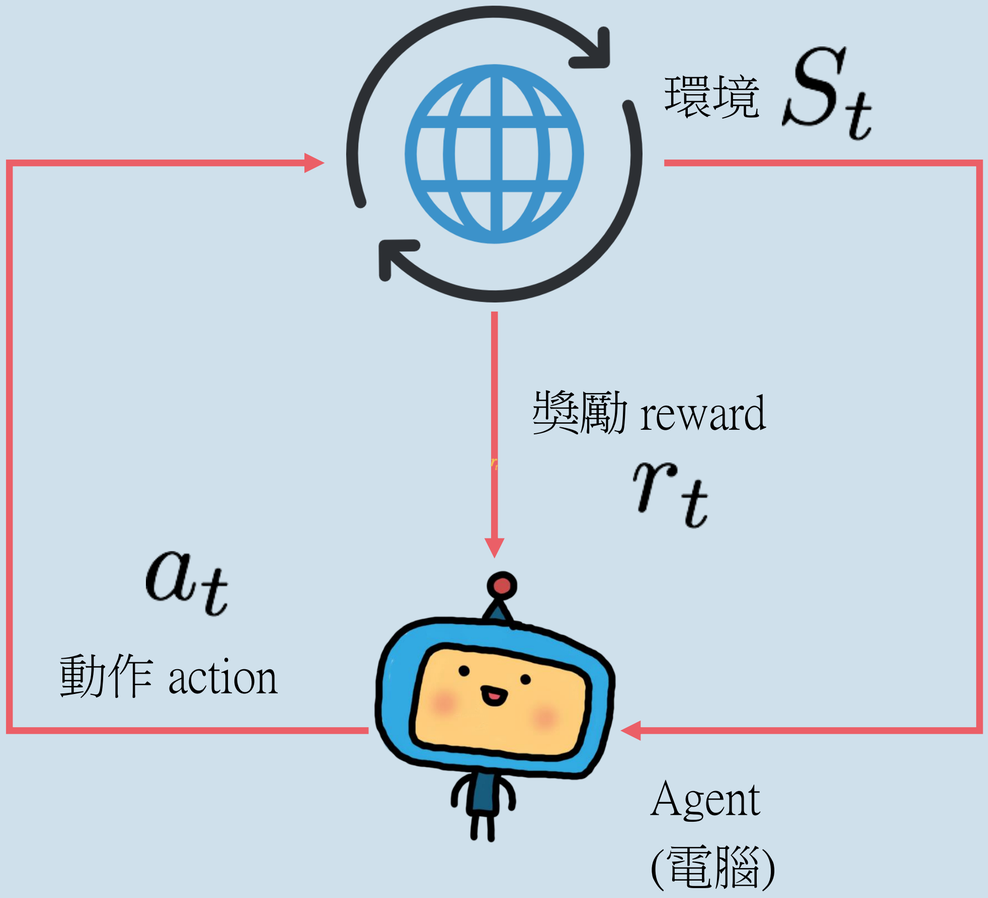
重要實例: AlphaGo

2017 年台灣人工智慧年會
AlphaGo 創始人之一黃士傑博士演講

博士班時期開發 Erica,
拿到電腦圍棋世界冠軍!

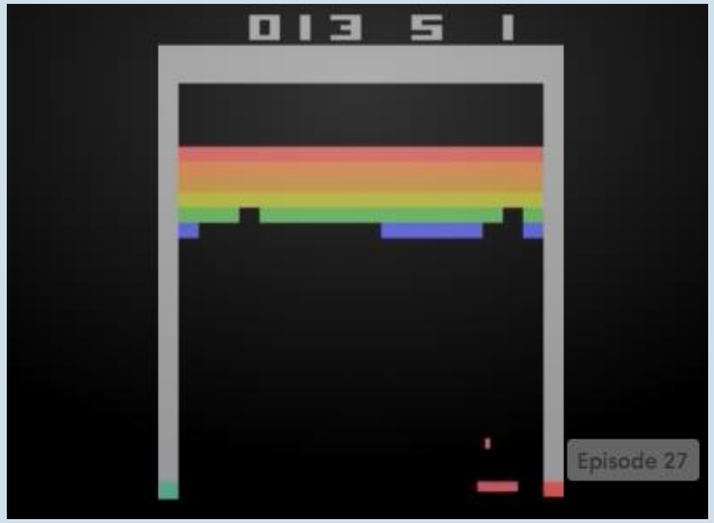
重要實例: AlphaGo

強化學習基本架構



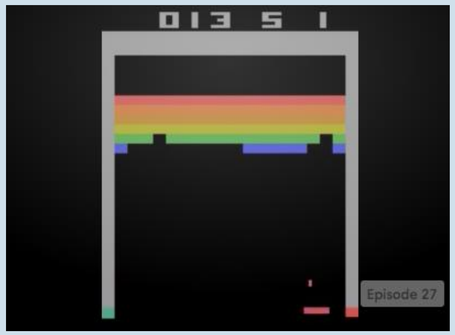
要學哪個函數呢?

我們以玩打磚塊為例



1 Policy Based

policy function



狀態 S_t

π

左

or

右

動作 a_t

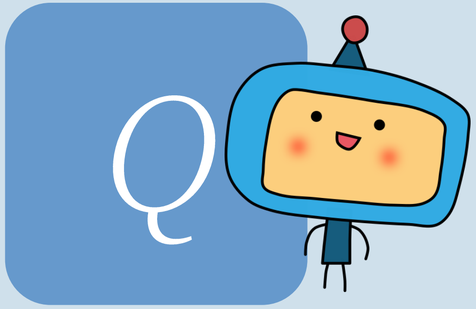
前面說過, 這函數可能很難準備訓練資料!

2 Value Based

\+

動作

Value function

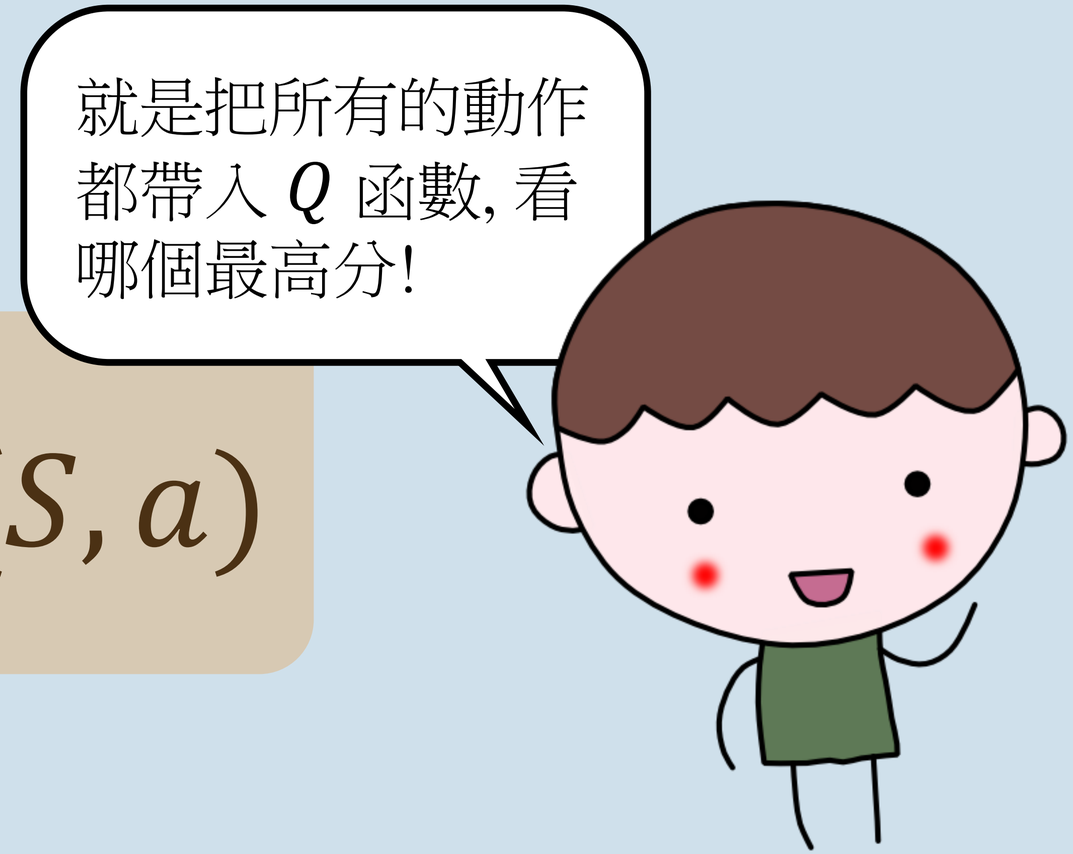


評分

(通常估計 reward)

Value Based

$$\pi(S) = \arg\max_{a} Q(S, a)$$

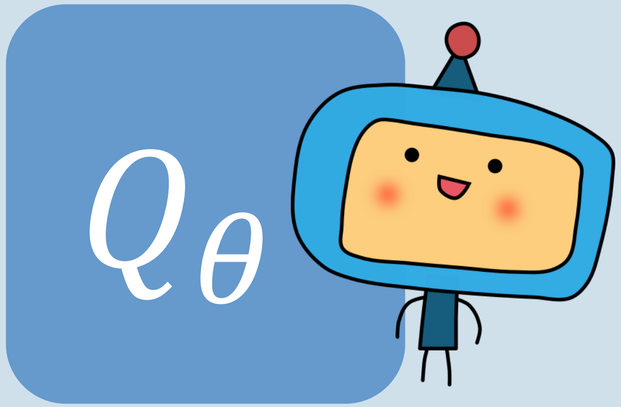


Greedy Policy

$$\pi(S) = \underset{a}{\mathrm{arg}max} Q(S, a)$$

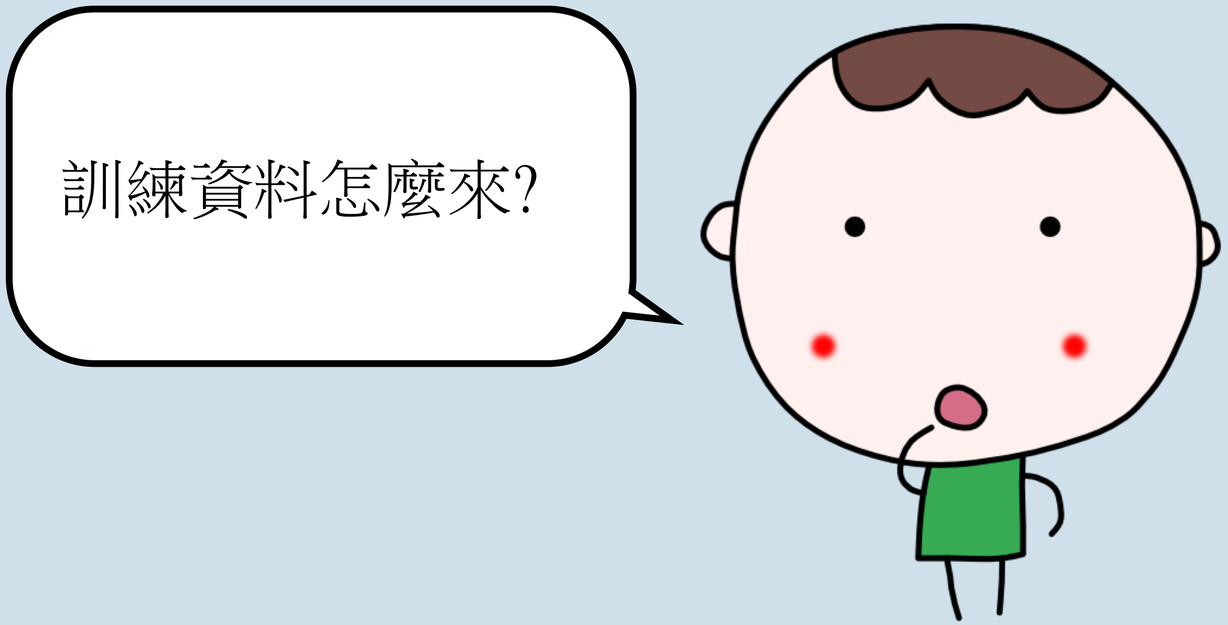
順道一提, 當 Q 函數學成時, 我們完完全全讓電腦依 Q 函數決定最好的動伯, 這叫 greedy policy!

Deep Q-Learning



深度強化學習最常見的 Deep Q-Learing, 就是用神經網路把 Q 函數學起來!

大問題



Deep Q-Learning

簡單的說, 就是我們會讓電腦自己去玩, 然後部份的情況我們知道 Q 值是多少。(雖然開始玩得很差, 這 Q 值可能很沒用)

然後用這些知道的 Q 值當訓練資料, 完整的 Q 函數用深度學習的方式學起來!

ε-Greedy Policy

這裡還有個問題...

開始的時候, 電腦要依什麼規則玩?

不要忘了我們的 Q 函數還爛得不得了啊...

ε-Greedy Policy

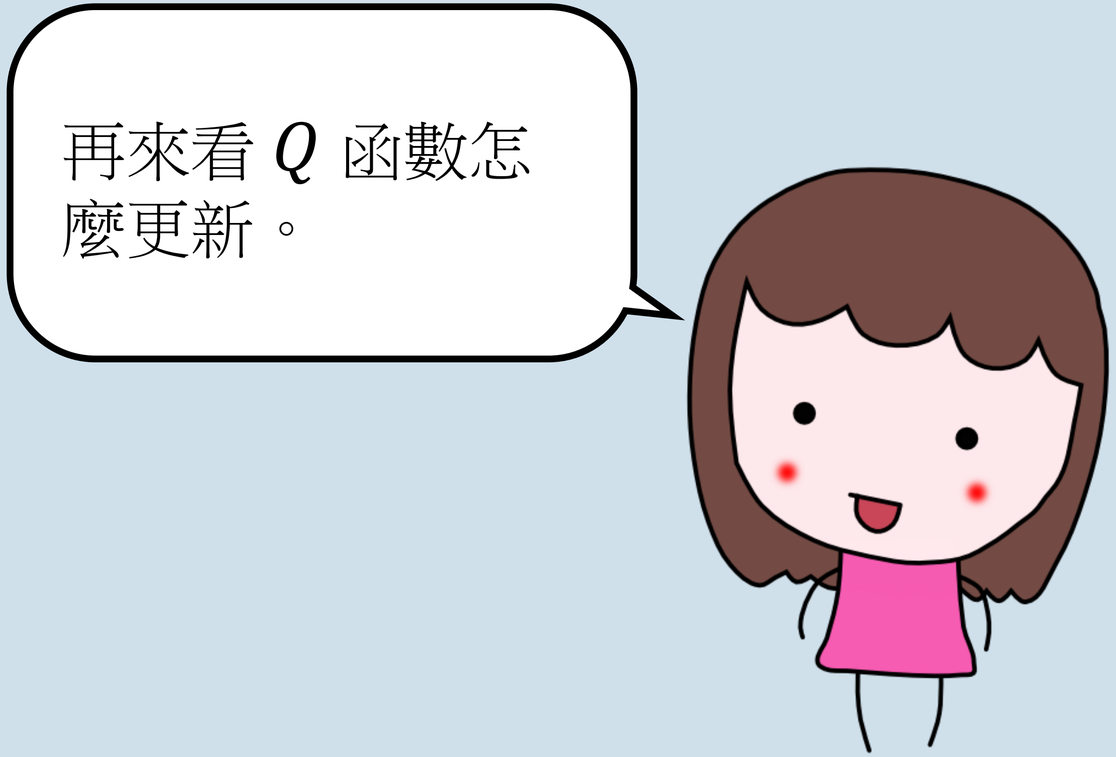
取一個 $\varepsilon \in [0,1]$, 我們每次要做個動作時, 取一個 0 到 1 間的亂數 r。

$$\begin{cases} r > \varepsilon & \text{Greedy Policy (用 } Q \text{ 函數決定)} \\ r \leq \varepsilon & \text{亂亂玩!} \end{cases}$$

Experience Replay

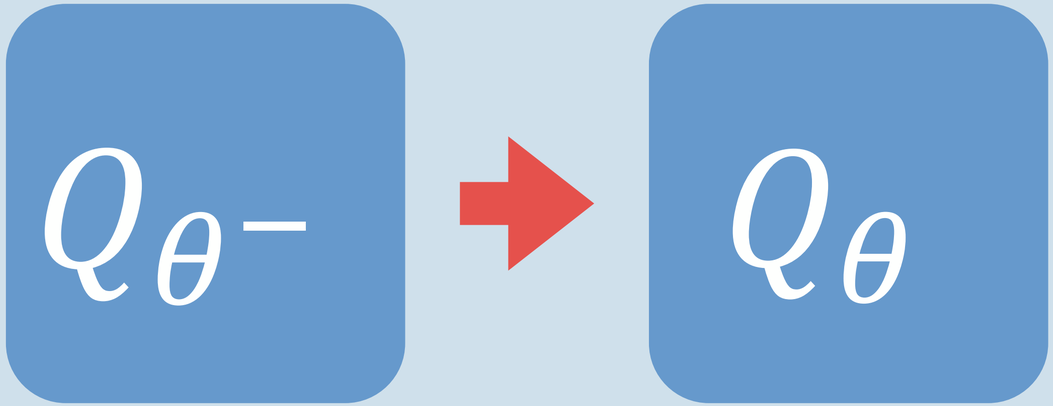
我們怎麼收集過去的經驗呢? 我們會發現一筆記錄應該要記得在某個狀態 S, 此時我們做了某個動作 a, 得到 r 的 reward, 然後進入到狀態 S'。

$$(S, a, r, S')$$



學習和被學的

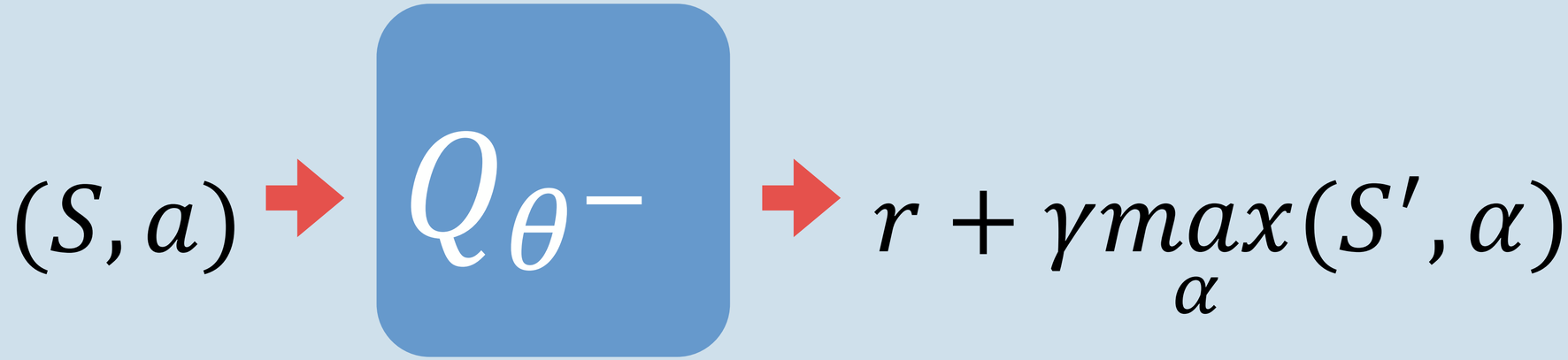
為了方便, 我們把上次參數的狀態叫 θ^-, 現在正要更新的叫 θ。



舊版　　更新版

學習和被學的

有了一筆經驗 (S, a, r, S'), 我們會
有一筆訓練資料是這樣...



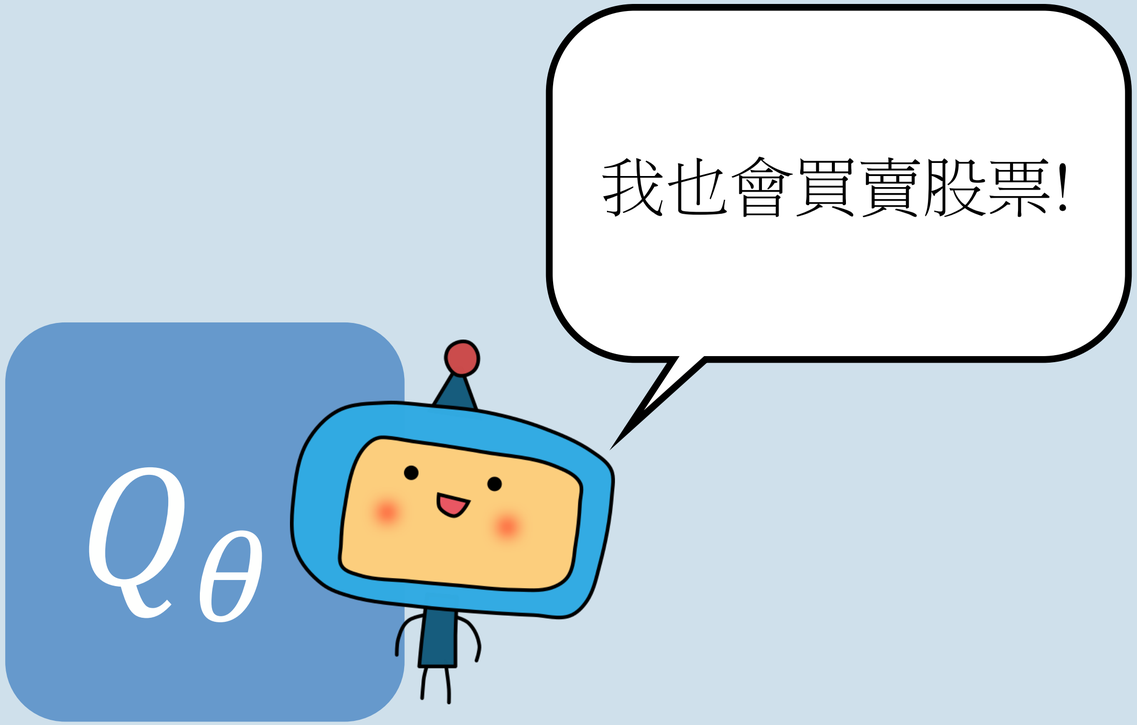
其中 γ 值是我們自己定的 discount, 通常是 0 到 1 間的一個數。

學習和被學的

範例: ETF 自動交易系統

卷積深度 Q-學習之 ETF 自動交易系統

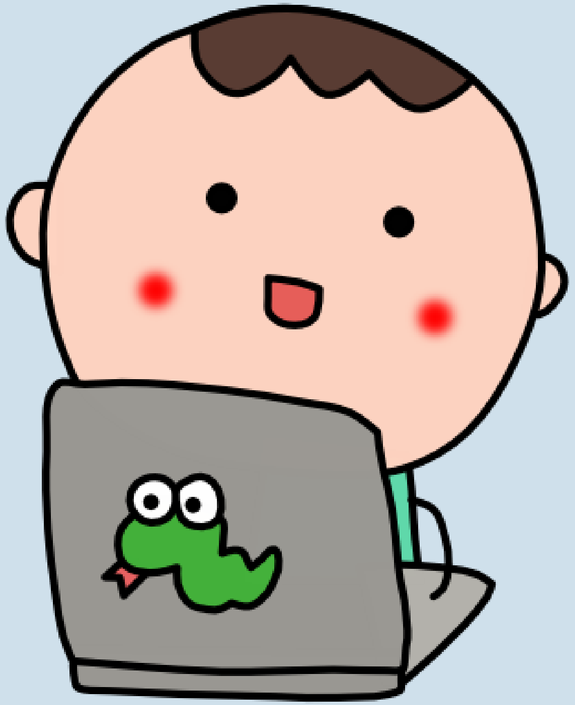
陳非霆



範例: ETF 自動交易系統

基本設定

- 選定一支 ETF
- 開始 20,000 美金
- 經過一年 (最後手上還有 ETF 就全賣)
- 使用 Deep Q-Learning



範例: ETF 自動交易系統

狀態 S_t

過去 20 天的資料
(20x6 的矩陣)

動作 a_t

1. 買 20 單位
2. 買 10 單位
3. 不做交易
4. 賣 10 單位
5. 賣 20 單位

五種 actions

範例: ETF 自動交易系統

交易結果

	CDQN	無腦法
ETF1	17.71%	10.89%
ETF2	16.53%	12.6%
ETF3	16.3%	0.35%
ETF4	14.4%	13.25%
ETF5	14.3%	12.7%
ETF6	13.91%	13.37%
ETF7	13.17%	10.52%
ETF8	12.35%	17.07%
ETF9	11.68%	10.81%
ETF10	11.09%	8.14%

	CDQN	無腦法
ETF11	10.76%	5.26%
ETF12	10.19%	13.17%
ETF13	7.8%	1.42%
ETF14	6.23%	3.56%
ETF15	5.73%	4.61%
ETF16	3.78%	-12.76%
ETF17	2.85%	5.83%
ETF18	1.59%	-4.45%
ETF19	1.07%	-18.09%
ETF20	-0.59%	-0.75%

* 「無腦法」正確的名稱是「買入持有策略」

Q & A