Class 3　CNN、RNN 與 LSTM

課程地圖

從圖片到時間序列

CNN 看一小塊、RNN 帶著記憶、
LSTM 會選擇要記什麼

實作：台積電大波動時，LSTM 能贏 SVR 嗎？

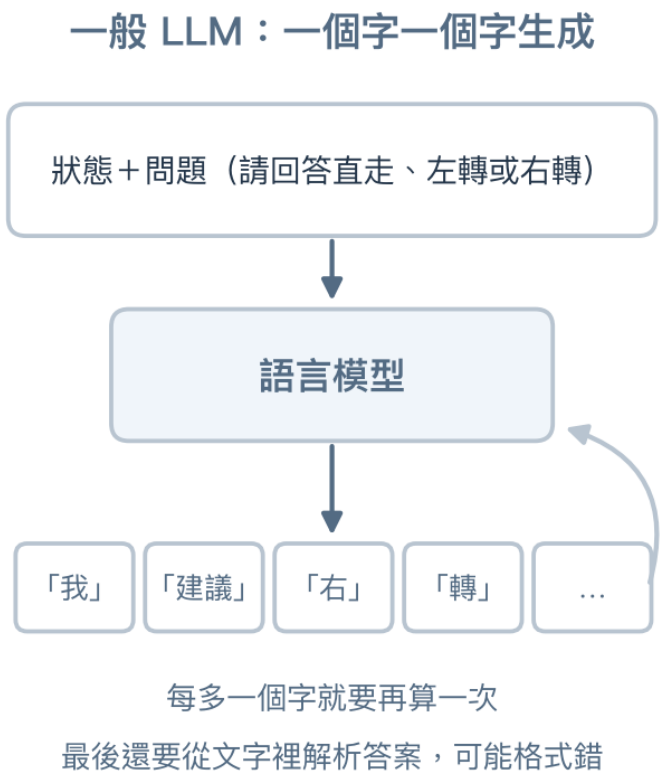


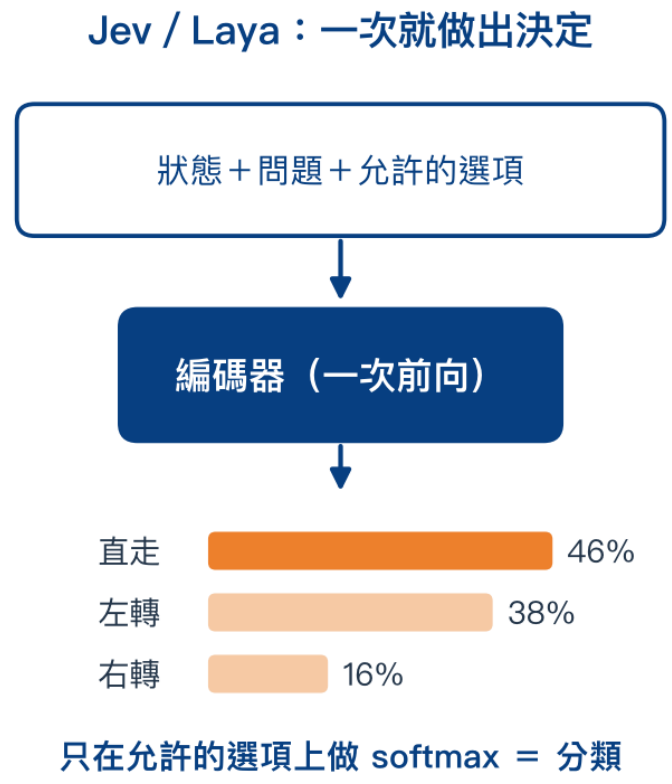
今天三個網路：看圖片的 CNN、記得前文的 RNN、會選擇記住什麼的 LSTM。

開場時事：Jev 為什麼這麼紅

時事（2026 年 9 月）

- 2026/9/15 TypeSafe AI 發表，創辦人是前 OpenAI 研究員 Diogo Almeida
- 不寫文字，一次前向就回答，官方稱 70–500 ms
- 只回答事先列出的選項，不會格式錯
- 機率經過校準訓練（RLCD）



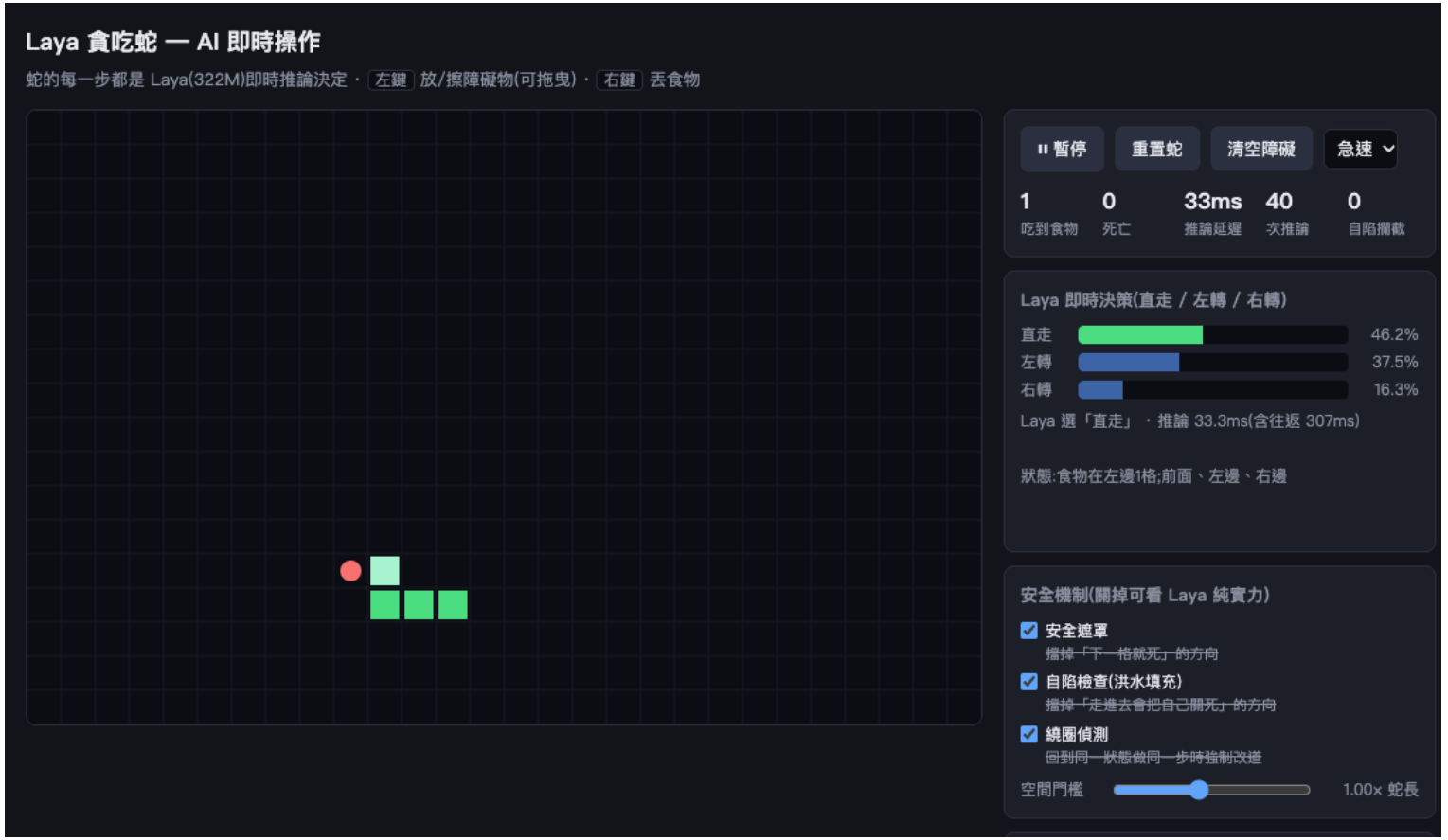


原理就是 Class 2 的分類：輸入一次，只在允許的選項上做 softmax。

示範：讓 Laya 開貪吃蛇

現場示範

- 每一步問一題：直走、左轉、還是右轉？
- 輸入：食物方位、前方左右有沒有空
- 約 30 ms 回答三個機率，選最高的走



和坦克的「移動頭」是同一件事：一個輸入、三個選項、一組機率。

今天的路線

課程地圖

① CNN

0–55 分

圖片太大，全連接塞不下，怎麼辦？

② RNN

55–90 分

同一個詞，意思不同，怎麼分？

③ LSTM

100–128 分

記憶要留多久、什麼時候讀？

④ 實作

128–158 分

LSTM 在大波動時比 SVR 好嗎？

⑤ 坦克

158–180 分

期中：訓練一台會自己打仗的坦克

90–100 休息 ▶ Colab A 手算 ▶ Colab B 台積電控制台 ▶ Colab C 坦克記憶

先看兩種「有結構」的資料，再回到股票做公平比較。

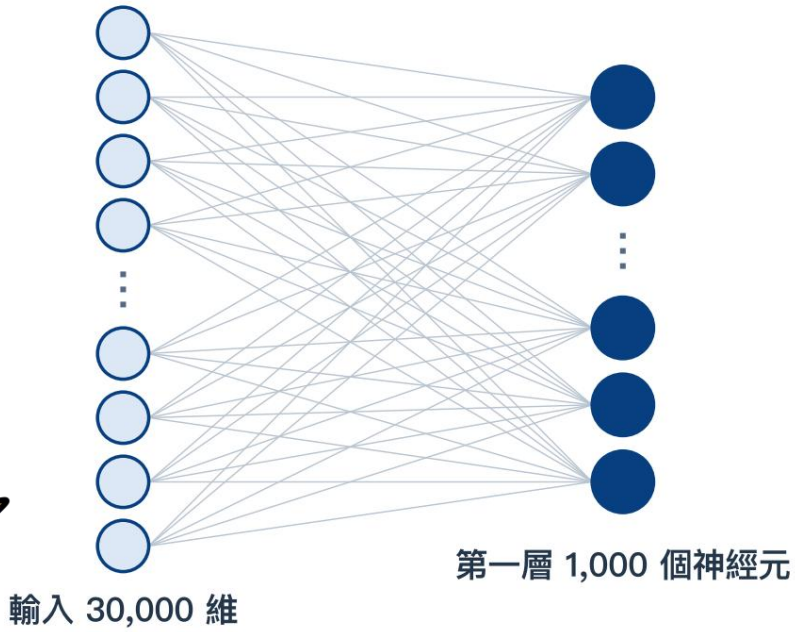
回顧：前兩堂的網路長什麼樣

複習

- Class 1：數字預測，損失 MSE
- Class 2：機率預測，損失交叉熵
- 共同點：每個神經元連到「每一個」輸入

輸入是一張圖片時，權重會爆炸





第一層權重數

$$30{,}000 \times 1{,}000 = 3 \times 10^7$$

參數越多 → 彈性越大
→ 越容易過擬合（Class 2）

Fully connected：每個神經元看全部輸入。輸入一大，參數就爆炸。

第一段　CNN：讓網路看懂圖片

章節導覽

上一段

全連接網路：每個神經元看全部輸入。

這一段要回答

一張圖片三萬個像素，神經網路要怎麼看？



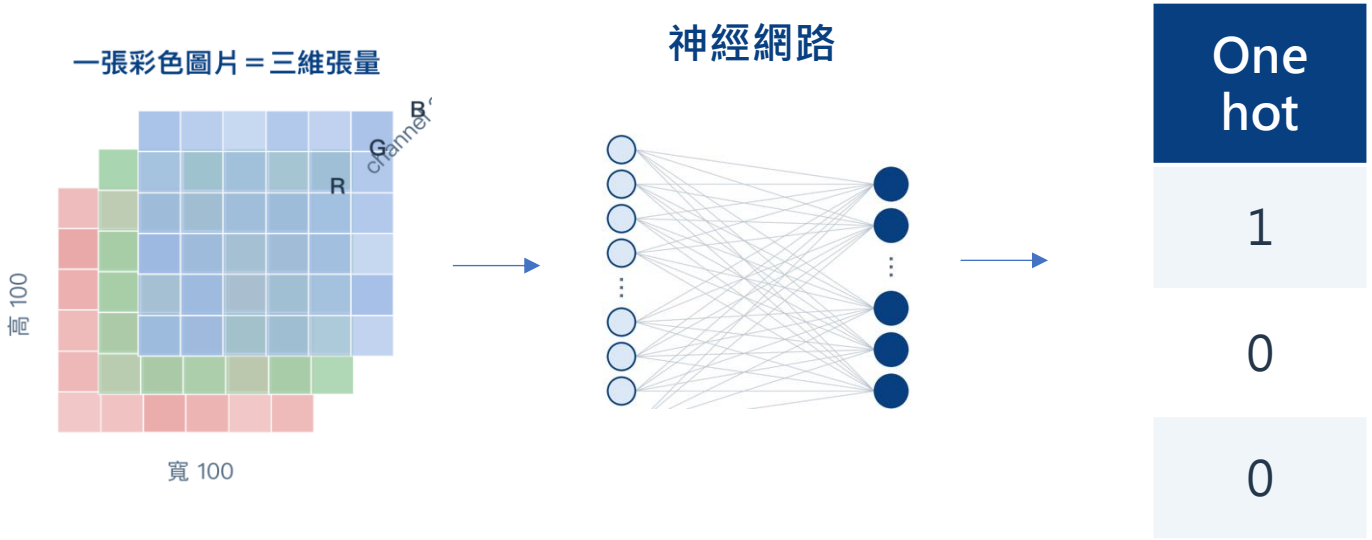
1. 圖片在電腦裡是什麼
2. 三個觀察 → 三個簡化
3. CNN 能用在哪、不能用在哪

跟著三個觀察走：只看一小塊、同圖案到處出現、縮小不影響。

任務：影像分類

先定義題目是什麼? 答案是什麼?

- 輸入：像素*RGB 如 100×100 x3
- 輸出：判斷類別的 one-hot，例如「貓」那一維是 1
- 損失：交叉熵 (Class 2 學過)
- 圖片 = 寬 × 高 × channel 的三維張量

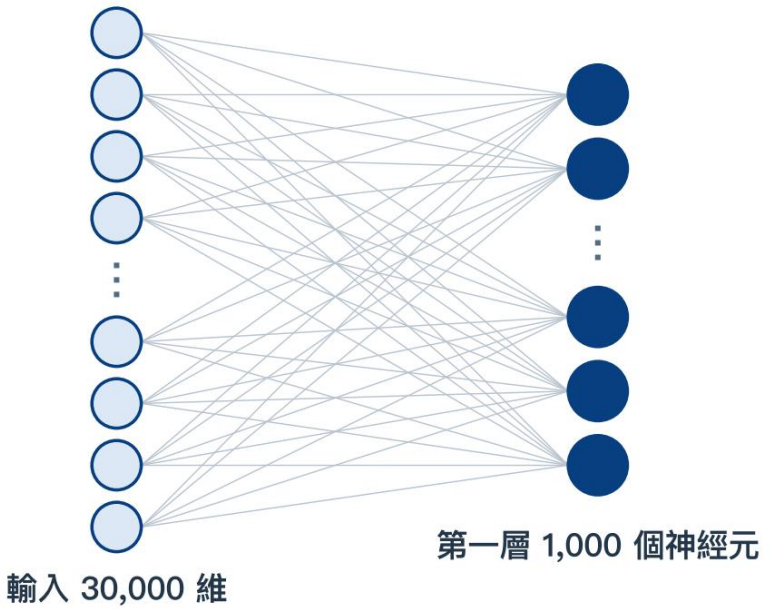


彩色圖片 = 100 × 100 × 3 個數字；每個數字是某位置、某顏色的強度。

直接拉直丟進全連接：參數太多

問題

- 30,000 維輸入、1,000 個神經元
- 第一層就有 3×10^7 個權重
- 彈性越大，越容易過擬合
- 能不能利用「圖片」本身的特性來少用參數？



第一層權重數

$$30{,}000 \times 1{,}000 = 3 \times 10^7$$

參數越多 → 彈性越大
→ 越容易過擬合（Class 2）

$30{,}000 \times 1{,}000 = 3\times10^7$：只是第一層。

觀察 1：辨識靠的是小範圍的關鍵 pattern

觀察

- 神經元的工作：偵測某個 pattern 有沒有出現
- 鳥嘴、眼睛、鳥爪 → 綜合起來是鳥
- 這些 pattern 都很小

→ 神經元不用看整張圖



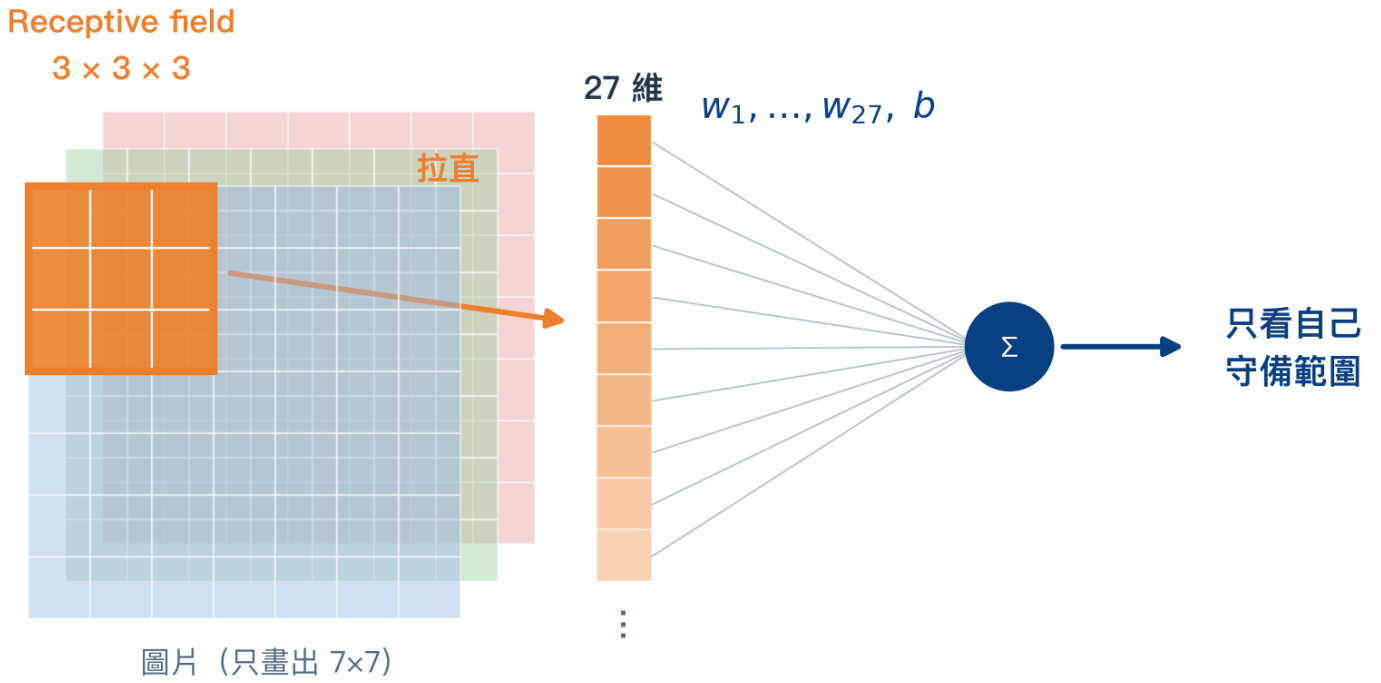
每個神經元只要看一小塊，就足以偵測一個關鍵 pattern。

簡化 1：Receptive field（感知域）

核心觀念

$$a = \sigma(w_1x_1 + \ldots + w_{27}x_{27} + b)$$

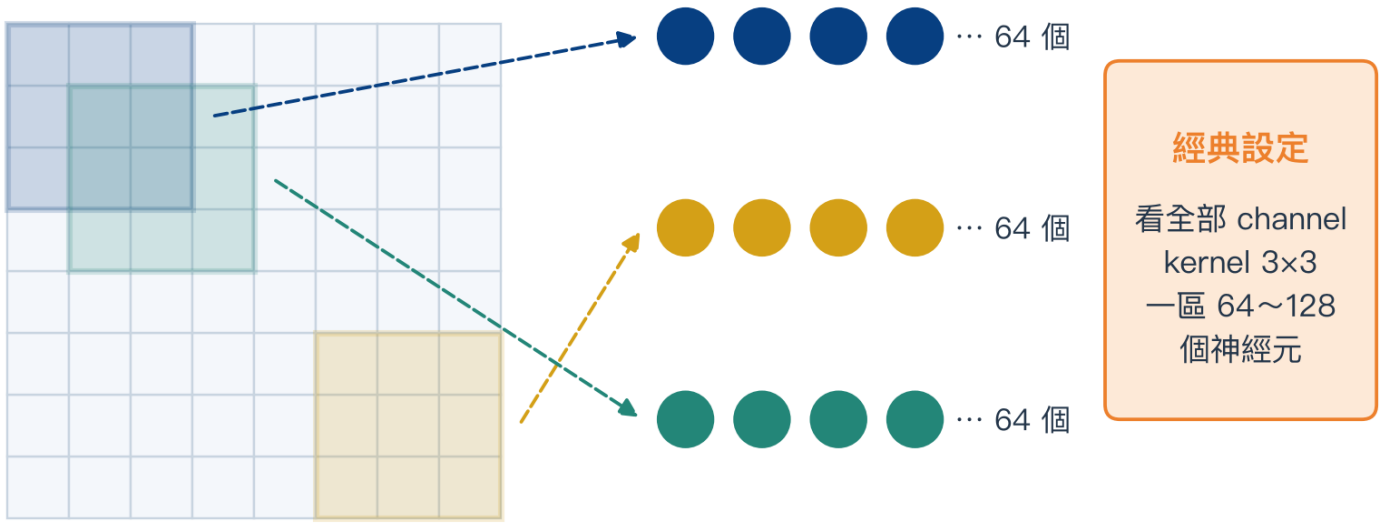
- 每個神經元只看 3×3×3 的小立方體
- 拉直成 27 維 → 27 個權重 + 1 個 bias
- 和 Class 1 的神經元算法一樣
- 只是輸入變少了



27 個權重取代 30,000 個：神經元只關心自己的守備範圍。

守備範圍可以重疊，一區有一組神經元

- 範圍彼此可以重疊
- 同一個範圍放 64～128 個神經元，各找不同 pattern
- 經典設定：看全部 channel
- kernel size 3×3 最常見



守備範圍可以重疊

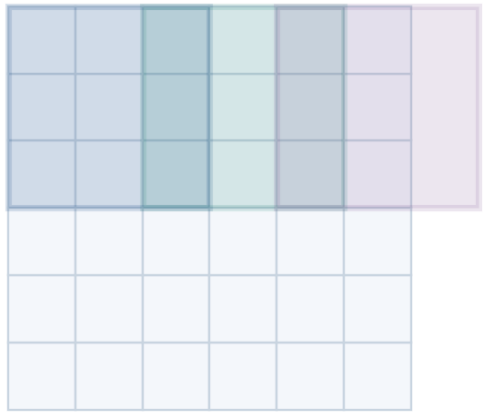
經典設定：kernel 3×3、看全部 channel、一區一組神經元。

Stride 與 padding：範圍怎麼排滿整張圖

核心觀念

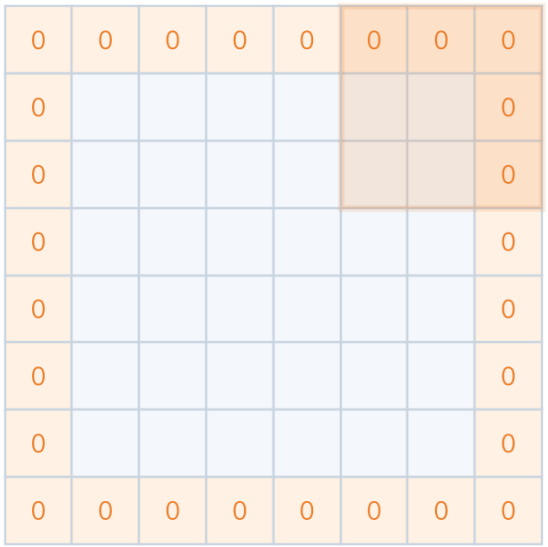
- Stride：**每次移動幾格**，通常 1 或 2
- Padding：**超出邊界補 0**

stride = 2：每次往右移 2 格



三個範圍互相重疊一格，交界的圖案也看得到

padding：超出邊界的地方補 0



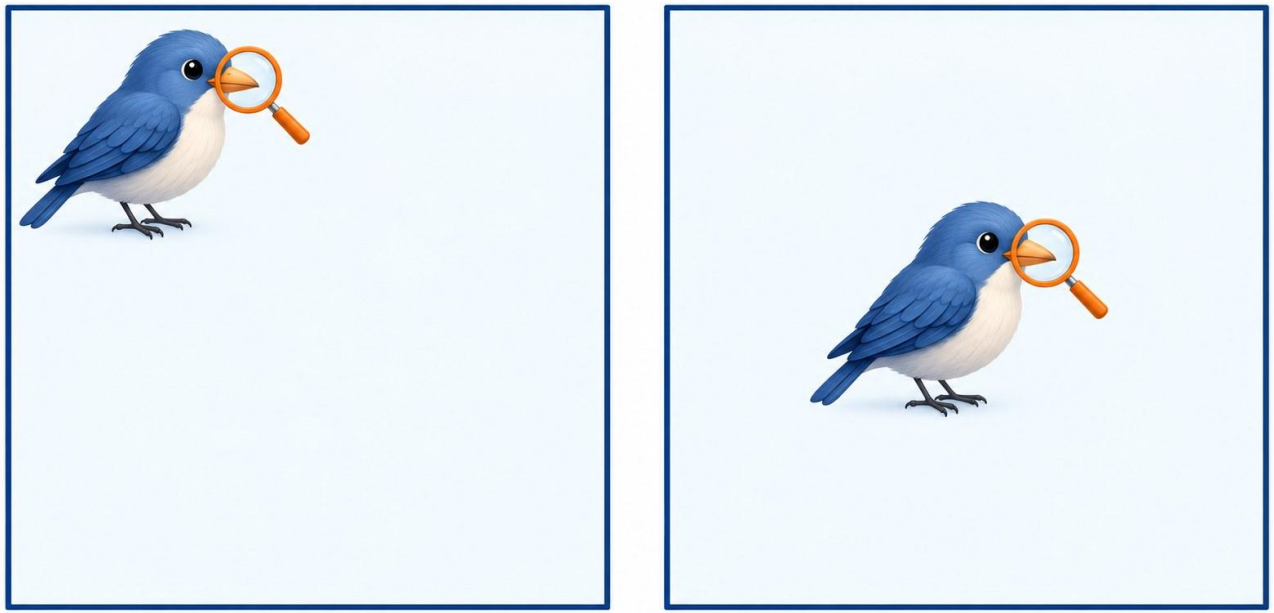
stride 決定走多快；padding 讓邊邊也被看到。

觀察 2：同樣的 pattern 出現在不同位置

觀察

- 鳥嘴可能在左上角，也可能在中間
- 每個位置都放一個「鳥嘴偵測器」？
- 它們做的事完全一樣

→ 重複了，參數太多

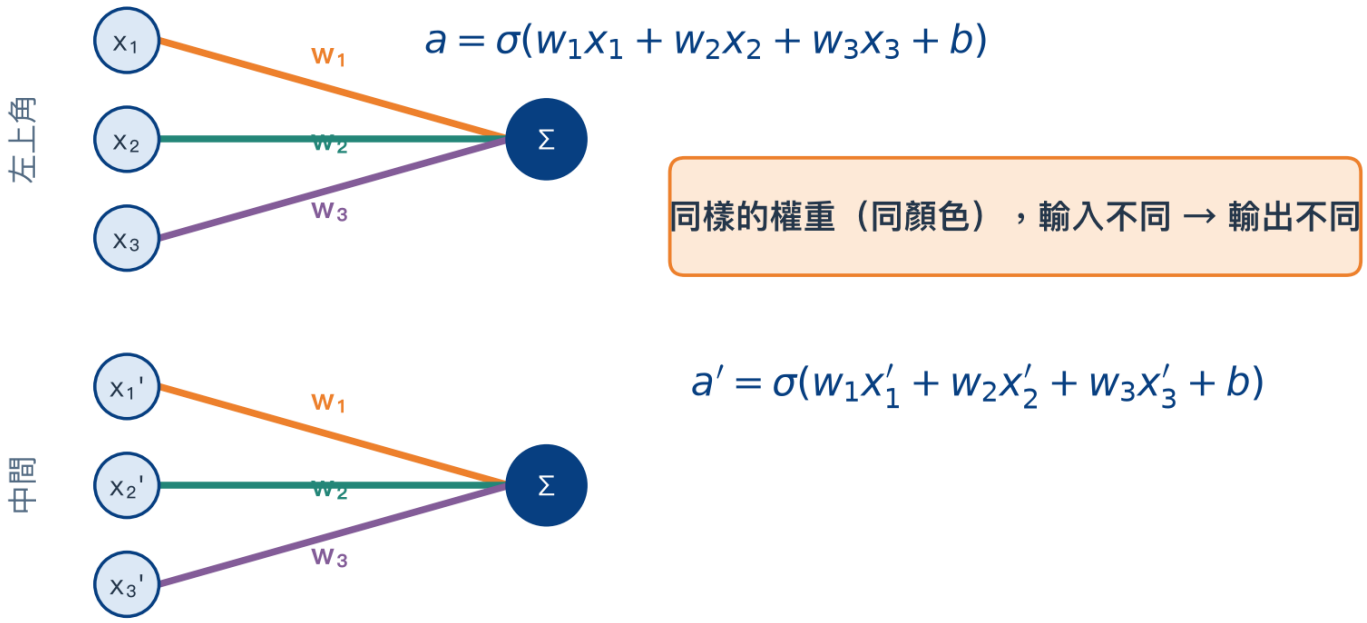


偵測鳥嘴的神經元，在每個位置做的事都一樣。

簡化 2：參數共享

核心觀念

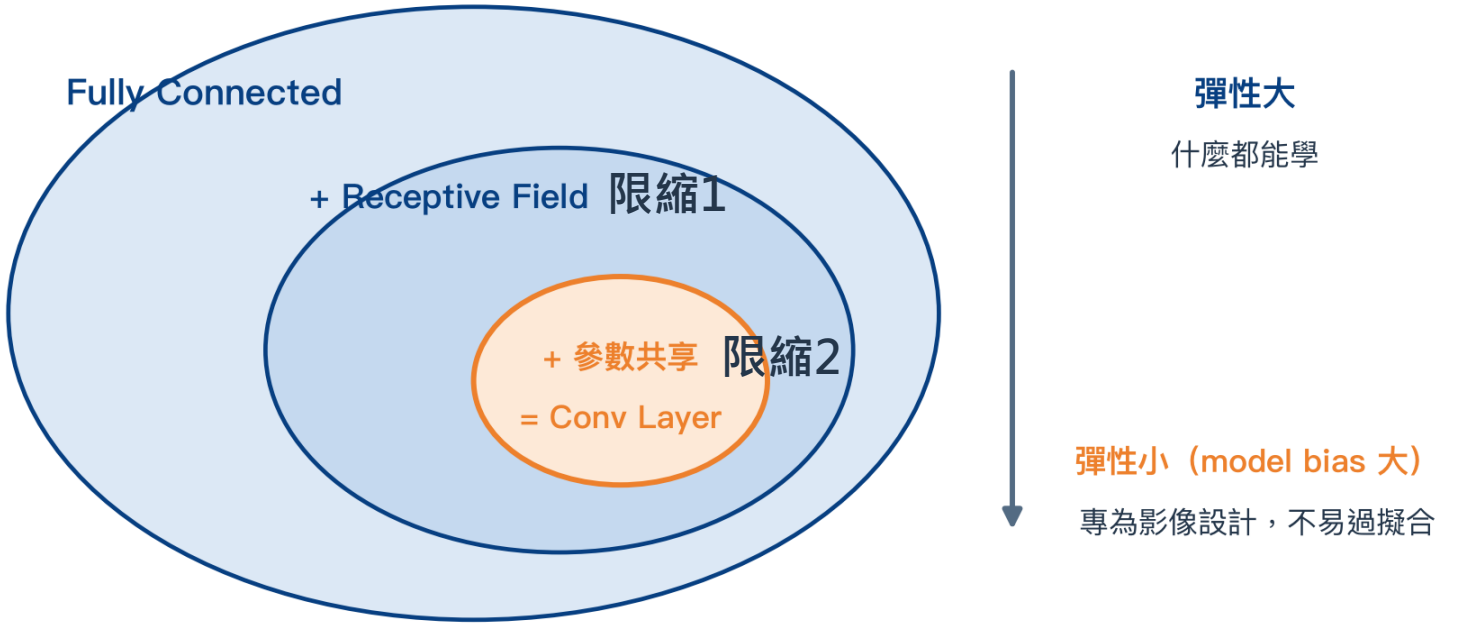
- 兩個神經元的權重一模一樣（同顏色）
- 守備範圍不同 → 輸入不同
- 所以輸出還是不同
- 同一個範圍內的神經元不會共享



同樣的權重，乘上不同位置的輸入，輸出就不同。

整理：CNN 是「被限制」的全連接

- 全連接：彈性最大，什麼都能學
- 加 receptive field：只能看一小塊
- 再加參數共享：不同位置用同一組
- ＝ Convolutional layer



CNN 的 model bias 大，但這些限制剛好符合影像的特性。

簡單理解：Filter 掃過圖片

圖解

$$feature[i, j] = \Sigma\, image[i+u, j+v] \cdot filter[u, v]$$

6 × 6 圖片

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

stride = 1

Filter 1

1	−1	−1
−1	1	−1
−1	−1	1

找「右下斜線」

左上 9 格 · Filter 1 = 3

往右一格 · Filter 1 = −1

（對應位置相乘，再全部加起來）

3	−1	−3	−1
−3	1	0	−3
−3	−3	0	1
3	−2	−2	−1

4 × 4
Feature map

最大值 3：
左上、左下出現斜線

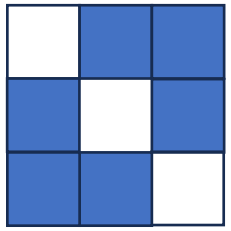
把 filter 放在左上角，對應位置相乘相加；往右移一格再算一次。

圖示理解：遮罩

圖解

$$feature[i, j] = \Sigma\, image[i+u, j+v] \cdot filter[u, v]$$

1	0	0
0	1	0
0	0	1



6 × 6 圖片

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

stride = 1

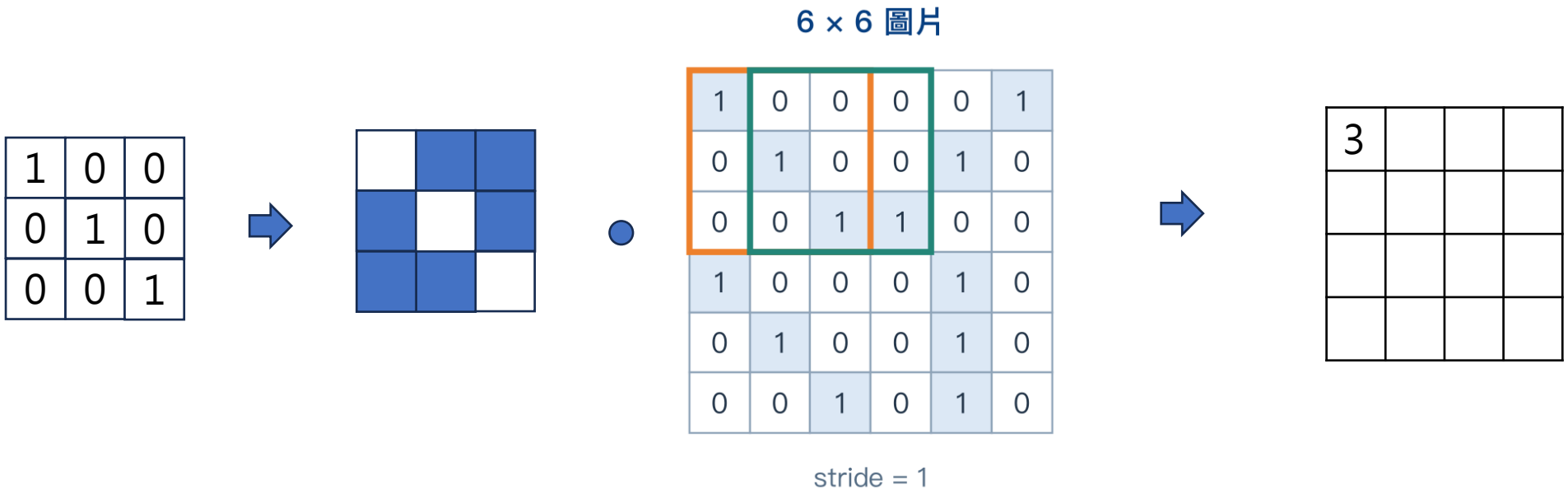
3	1	0	1
0	2	2	0
0	0	2	2
3	0	1	1

把 filter 放在左上角，對應位置相乘相加；往右移一格再算一次。

圖示理解：遮罩 (動手玩玩看)

圖解

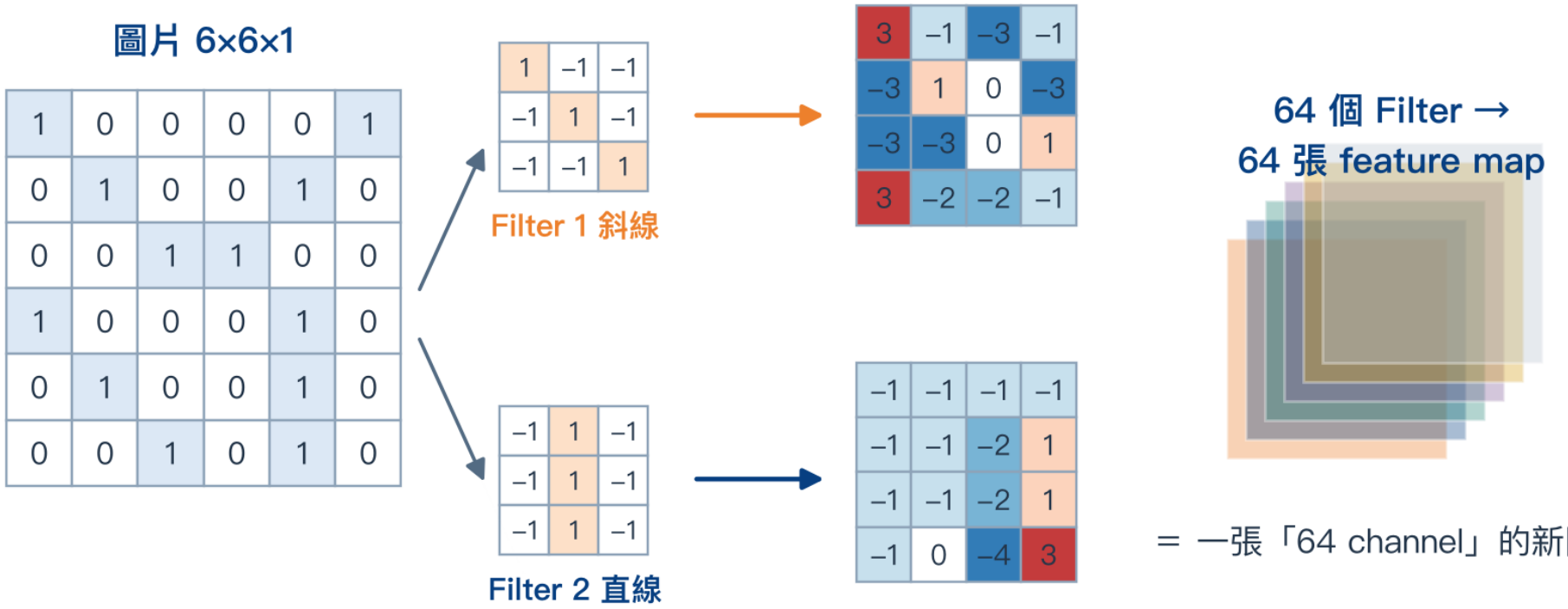
$$feature[i, j] = \Sigma\, image[i+u, j+v] \cdot filter[u, v]$$



把 filter 放在左上角，對應位置相乘相加；往右移一格再算一次。

多個 Filter → 一疊 Feature map

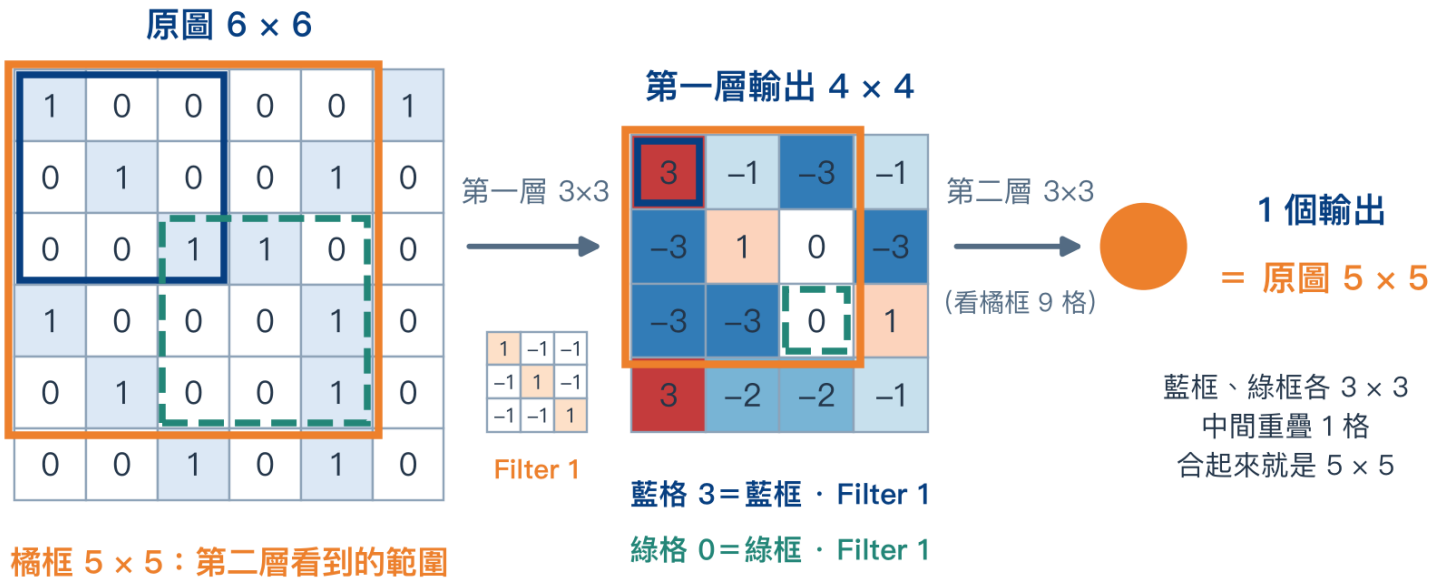
圖解



64 個 filter 產生 64 張 feature map，可以看成一張 64 channel 的新圖片。

只看3×3會不會漏掉圖片其他部分

- 第二層的 3×3，對應第一層的 3×3 個輸出
- 每個輸出又看原圖 3×3
- 合起來在原圖上是 5×5
- 疊越深，看的範圍越大

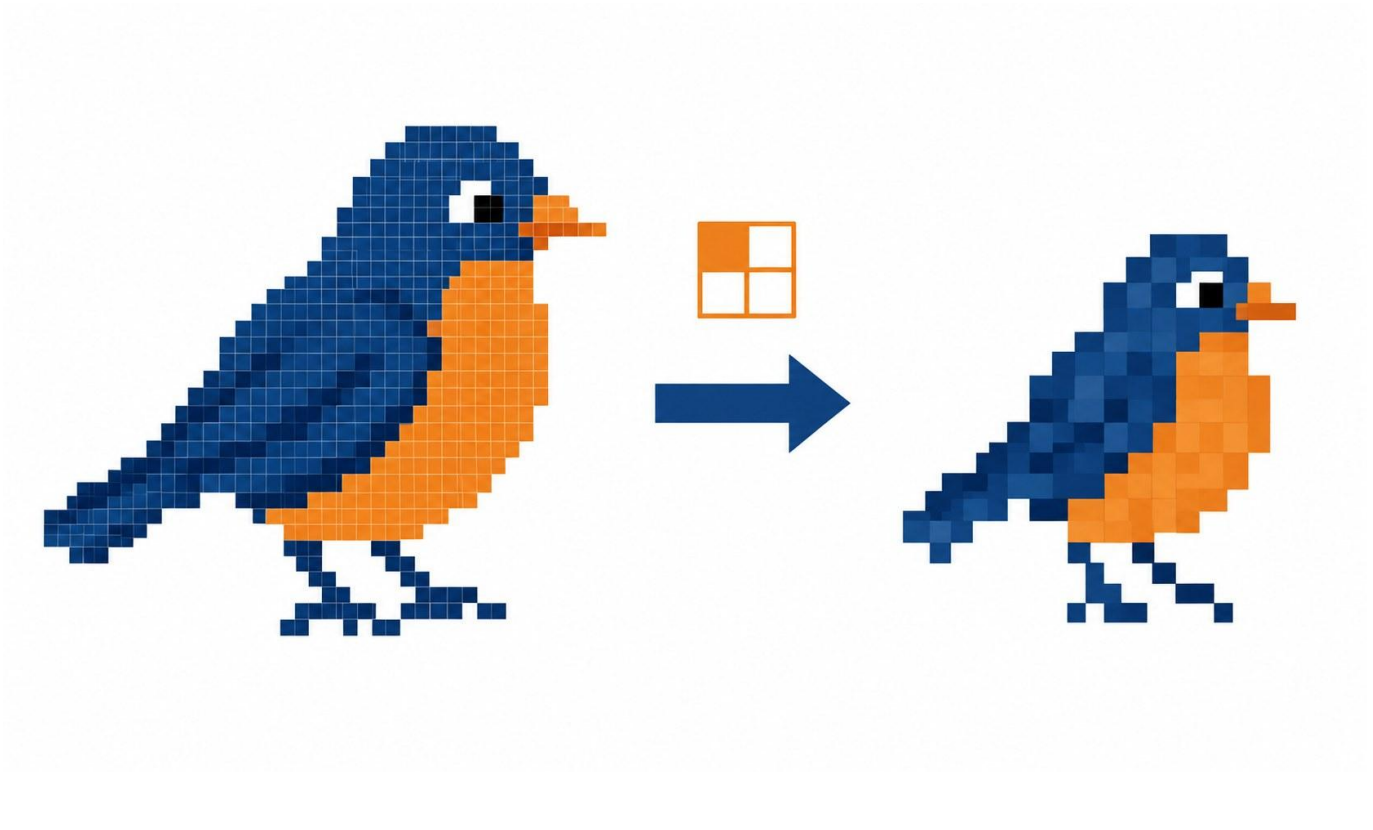


回答前面的疑問：kernel 很小，但疊深了就看得到大 pattern。

觀察 3：縮小圖片，還是同一隻鳥

觀察

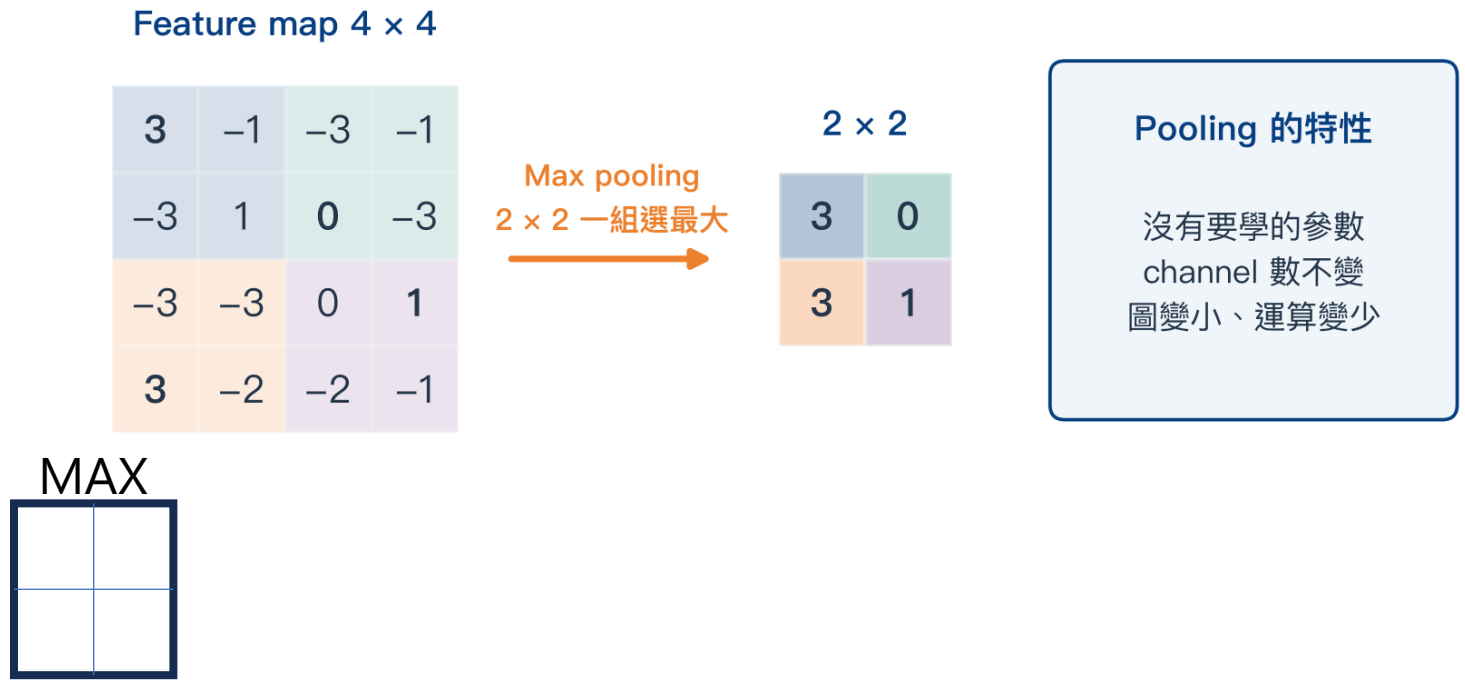
- 把偶數行、奇數列拿掉
- 圖片變成 1/4
- 內容看起來沒變
- → 可以 subsampling，減少運算



圖片縮小不影響判讀：這帶出 pooling。

Max pooling：每組選一個代表

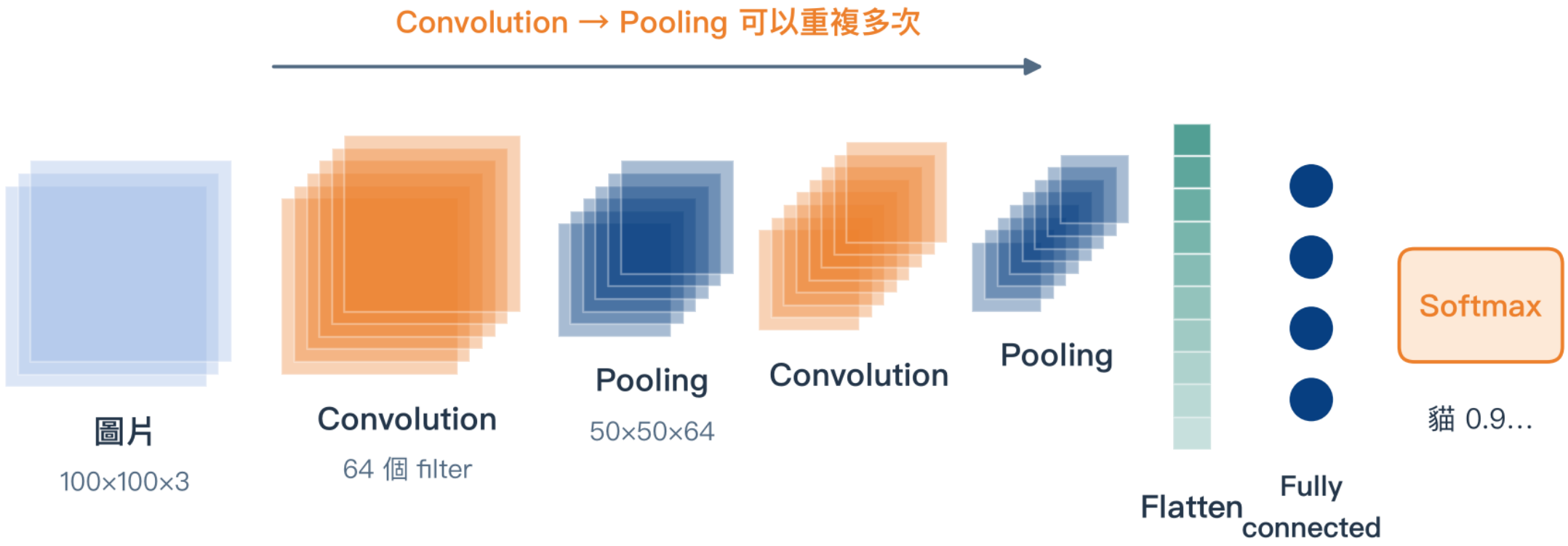
- 2×2 一組，選最大值
- 沒有要學的參數，像 ReLU 一樣固定
- channel 數不變，圖變小
- 也有 mean pooling 等其他版本



Pooling 讓圖變小、運算變少；運算夠時，很多網路乾脆不用。

完整的 CNN

架構

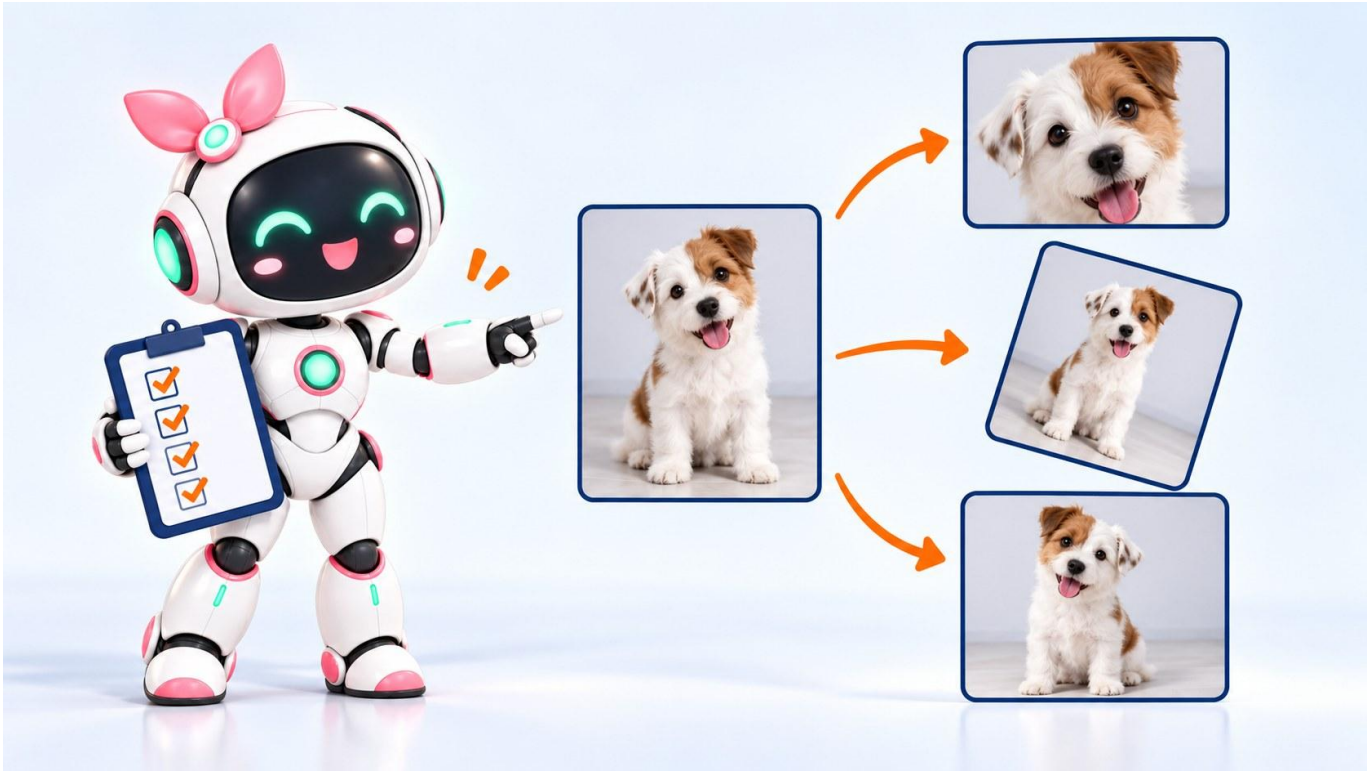


Convolution、pooling 重複幾次 → flatten → 全連接 → softmax。

CNN 的限制：放大、旋轉就不認得

限制

- 放大的狗，拉直後數字完全不同
- CNN 本身處理不了縮放、旋轉
- Data augmentation：裁切放大、旋轉、翻轉
- 讓訓練資料先看過各種樣子



CNN 不會自動處理縮放與旋轉：靠資料擴增補上。

期中預告：坦克的 32 條射線，也是一張「圖片」

應用

①CNN ②RNN ③LSTM ④實作 ⑤坦克



32 條射線排成一列（顏色越深＝牆越近；第 0 條＝車頭正前方）



同一個 3 格 filter 滑過整圈：兩處都找到「近、更近、近」＝牆角

方向以車頭為準 → 同一個 pattern 在任何方向都一樣 → 參數共享

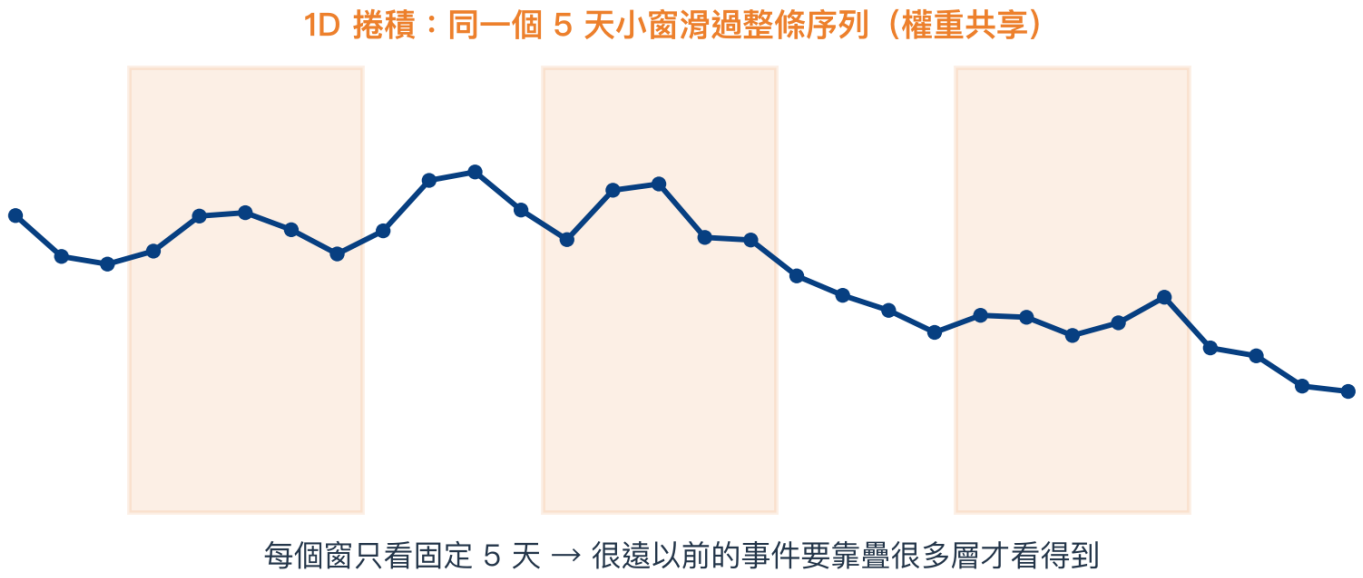
最後一條和第一條相鄰 → padding 要「繞回去」，不是補 0

期中範例沒有用 CNN：這是可以嘗試的改進方向

觀察 1、2 在這裡也成立：牆角是小範圍的 pattern，出現在哪個方向都一樣。

從空間到時間：1D 捲積

- 股價也有「局部 pattern」：連漲三天、跳空
- 同一個 5 天小窗滑過整條序列
- 但每個窗只看固定 5 天
- 很久以前的事件，要疊很多層才看得到



CNN 看「附近」；序列還需要「記得前面發生過什麼」→ RNN。

例子：智慧訂票系統（slot filling）

題目

- 「I would like to arrive Taipei on November 2nd」
- 系統要填：

  目的地 = Taipei

  抵達時間 = November 2nd

- 每個詞要分到某個 slot（或都不屬於）



Slot filling：句子裡每個詞，屬於哪一個欄位？

詞彙怎麼變成向量

前處理｜課後閱讀（放映略過）

- 1-of-N encoding：就是 one-hot
- 字典外的詞 → 多一維 other
- 也可以用字母 n-gram
- 有了向量，就能丟進網路



1-of-N（就是 Class 2 的 one-hot）

字典外的詞？

多一維 other：Gandalf → other

字母 n-gram

apple → app、ppl、ple 三維為 1

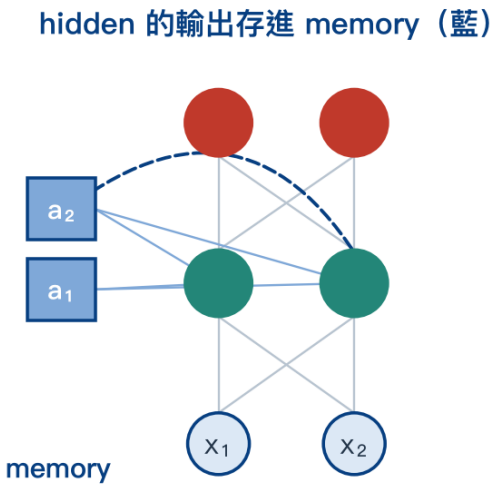
和 Class 2 的 one-hot 一樣：類別變成 0 / 1 向量。

RNN：hidden 的輸出存進 memory

核心觀念

$$a^t = \sigma(W \cdot x^t + U \cdot a^{t-1} + b)$$

- hidden layer 的輸出存進 memory（藍）
- 下一個輸入進來時
- hidden 同時考慮 x 與 memory
- → 同輸入可能有不同輸出



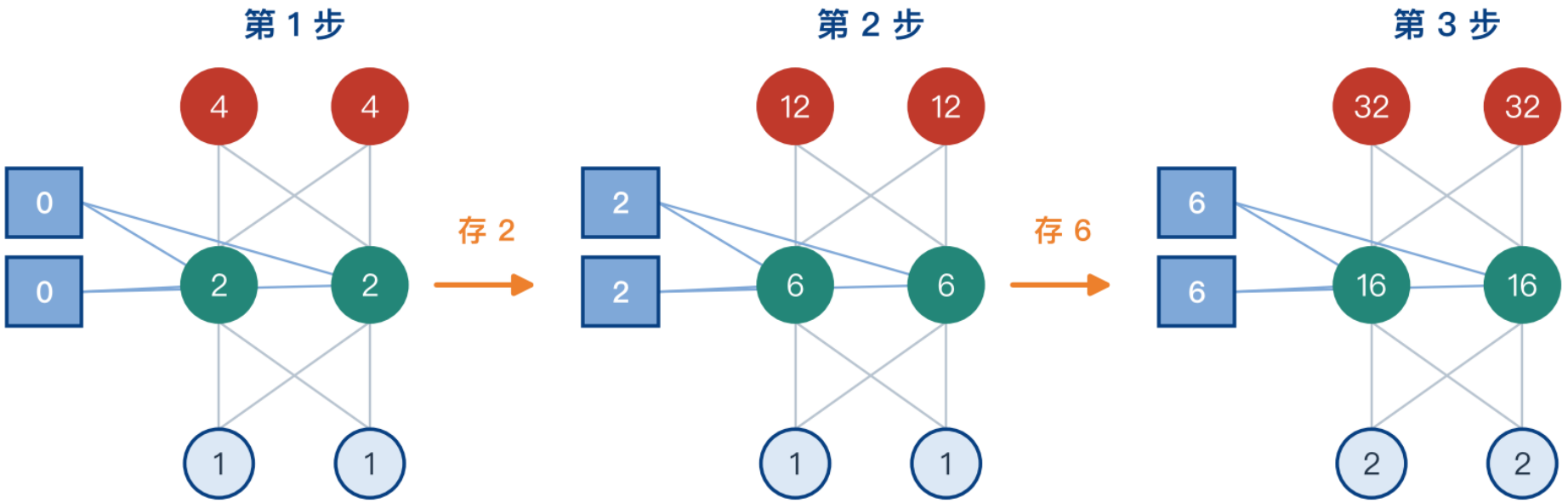
下一次輸入時

hidden 同時考慮
新的輸入 x 和 memory 裡的 a

→ 同樣輸入，可能得到不同輸出

memory 存上一步的 hidden；這一步把它當作額外的輸入。

手算：所有權重都是 1



所有權重 = 1、沒有 bias、線性激活；memory 一開始是 0

輸入 [1,1]、[1,1]、[2,2] → 輸出 [4,4]、[12,12]、[32,32]。

想一想④

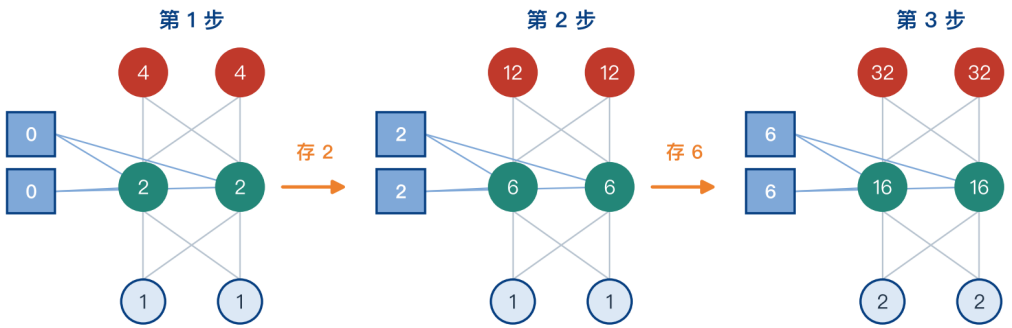
手算

把順序換成 [2,2]、[1,1]、[1,1]，最後一步的輸出是多少？

A [32, 32]

B [44, 44]

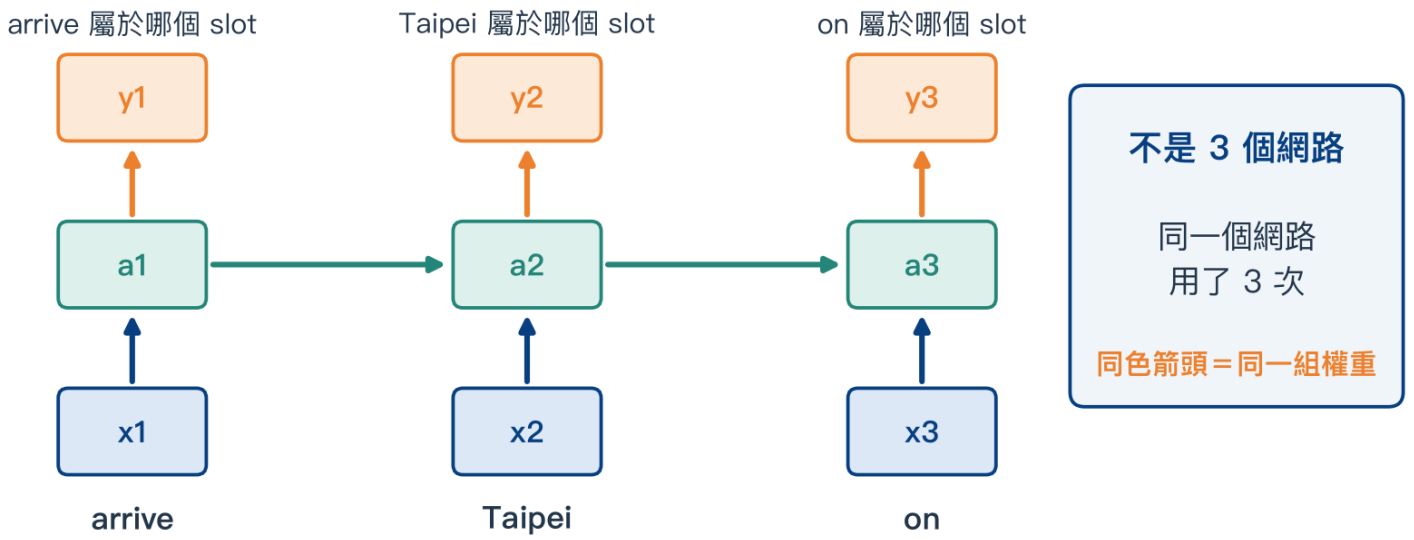
C [16, 16]



同樣三個輸入，換了順序，答案就不同。

展開來看：同一個網路用了很多次

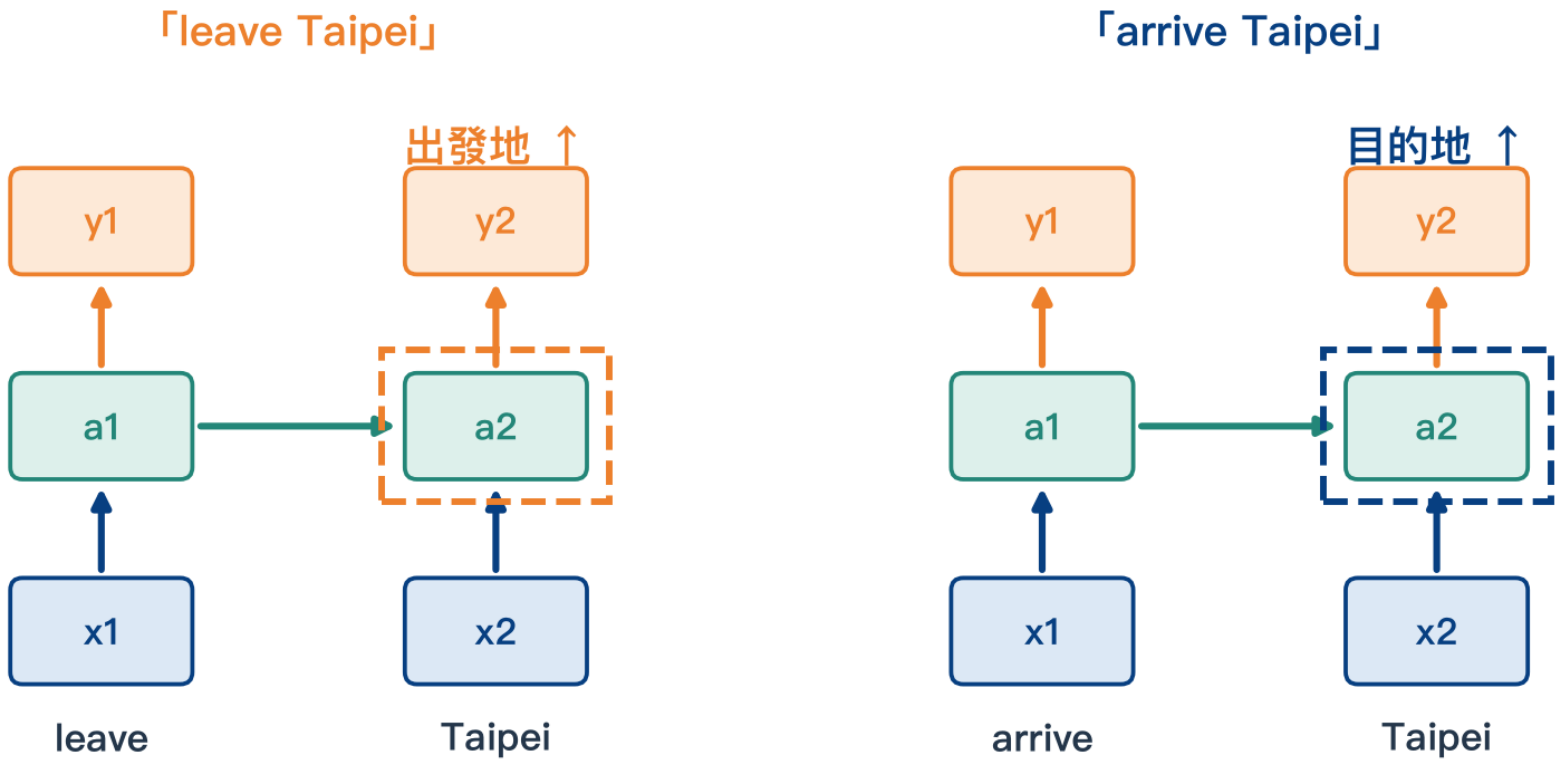
- arrive → a^1 → y^1
- Taipei 和 a^1 → a^2 → y^2
- on 和 a^2 → a^3 → y^3
- 同色箭頭 = 同一組權重



不是三個網路，是同一個網路在三個時間點被使用三次。

同一個 Taipei，因為記憶不同而答案不同

圖解



Taipei 的輸入一樣，但存在 memory 裡的 a1 不同 → 輸出不同

leave 和 arrive 的向量不同 → a^1 不同 → Taipei 那一步的輸出不同。

人咬狗、狗咬人：一般神經網路分不出來

手算



① 熱編碼

人	1	0	0
咬	0	1	0
狗	0	0	1

② 每個字各自算 h = W·x

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

h(人) = [1, 0]
h(咬) = [0, 1]
h(狗) = [2, 1]

③ 一句話＝三個字的 h 加起來

人咬狗：人 [1, 0] + 咬 [0, 1] + 狗 [2, 1] = **[3, 2]**

狗咬人：狗 [2, 1] + 咬 [0, 1] + 人 [1, 0] = **[3, 2]**

一模一樣

「狗」在句首或句尾，算出來都是 [2, 1]。

一般神經網路沒有「時間」，分不出誰咬誰。

同樣三個字只是換順序，一般神經網路算出來一模一樣：它沒有「時間」，不知道誰先誰後。

RNN：記憶讓順序有意義

手算

RNN： $a^t = Wx^t + 0.5\,a^{t-1}$　　（W 和上一頁相同；一開始 a = [0, 0]；記憶每一步打五折）

人咬狗

第 1 步：人	×0.5	第 2 步：咬	×0.5	第 3 步：狗	
W·x = [1, 0]	→	W·x = [0, 1]	→	W·x = [2, 1]	
+ 0.5·a = [0, 0]		+ 0.5·a = [0.5, 0]		+ 0.5·a = [0.25, 0.5]	
a = [1, 0]		a = [0.5, 1]		a = [2.25, 1.5]	[2.25, 1.5]

不同！

狗咬人

第 1 步：狗	×0.5	第 2 步：咬	×0.5	第 3 步：人	
W·x = [2, 1]	→	W·x = [0, 1]	→	W·x = [1, 0]	
+ 0.5·a = [0, 0]		+ 0.5·a = [1, 0.5]		+ 0.5·a = [0.5, 0.75]	
a = [2, 1]		a = [1, 1.5]		a = [1.5, 0.75]	[1.5, 0.75]

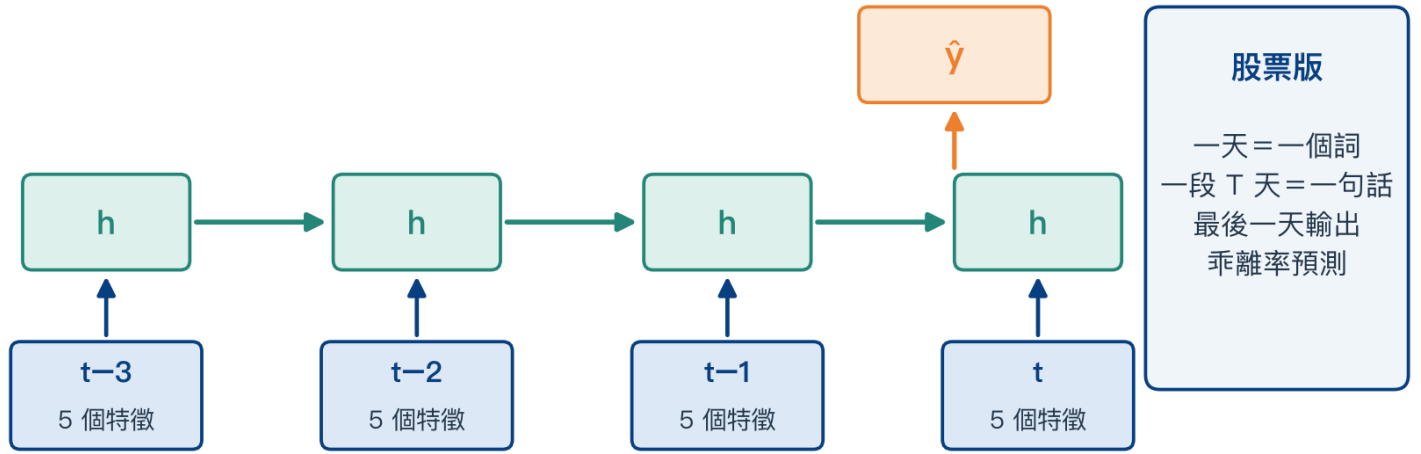
若記憶不打折（×1），兩句最後都是 [3, 2]，又分不出來：記憶怎麼混合，就是 RNN 要學的權重。

同樣的 W，加上打五折的記憶：人咬狗 [2.25, 1.5]、狗咬人 [1.5, 0.75]，順序不同，結果就不同。

換成股票：一天就是一個詞

銜接

- 每天 5 個特徵（Class 2 的乖離率、報酬）
- 取最近 T 天，一天一天讀進 RNN
- 最後一天的 hidden → 預測乖離率
- 每段視窗都從空白記憶開始



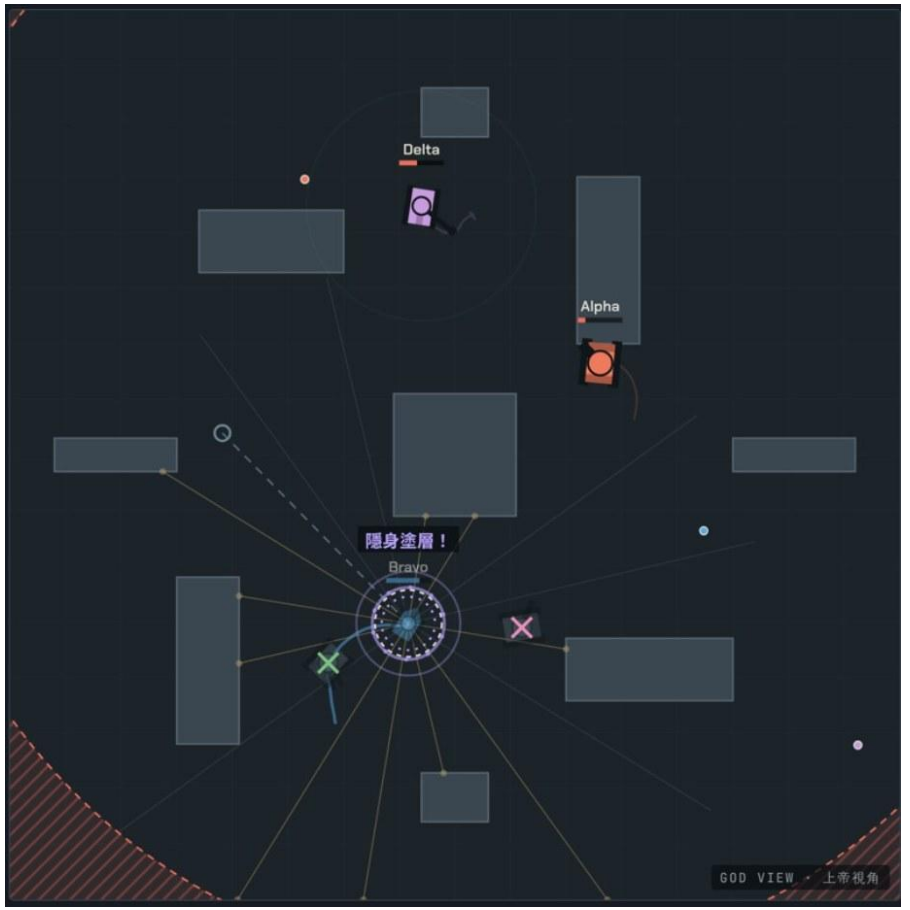
一段 T 天的歷史 = 一句話；最後一天輸出預測。

換成坦克：敵人會從觀測裡消失

銜接

①CNN ②RNN ③LSTM ④實作 ⑤坦克

- 坦克每一步看到 91 個數字 = 一個詞
- 一局最多 1,200 步 = 一個很長的句子
- 敵人躲到牆後或開隱身：觀測裡沒有他
- 只看當下的網路，就不知道他去哪了



期中的坦克只看得到「這一刻」：看不到的敵人，要靠記憶。

休息 10 分鐘

10 分鐘後回來

1. 打開 Class 3 Colab
2. 檔案 → 在雲端硬碟中儲存副本
3. 執行階段 → 全部執行



回來前，先打開 Colab，另存副本。

第三段　LSTM：會選擇的記憶

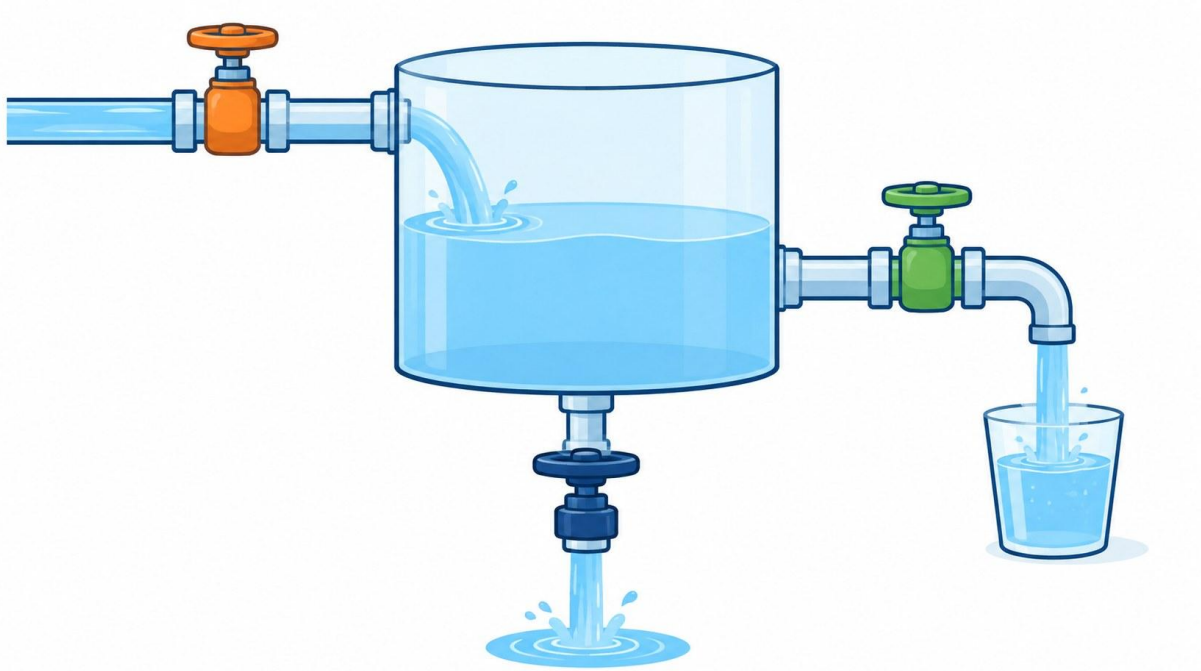
章節導覽

上一段

RNN：memory 每一步都被新的 hidden 洗掉

這一段要學習

讓神經網路也可以控制什麼時候該寫入、該忘記、該讀出？



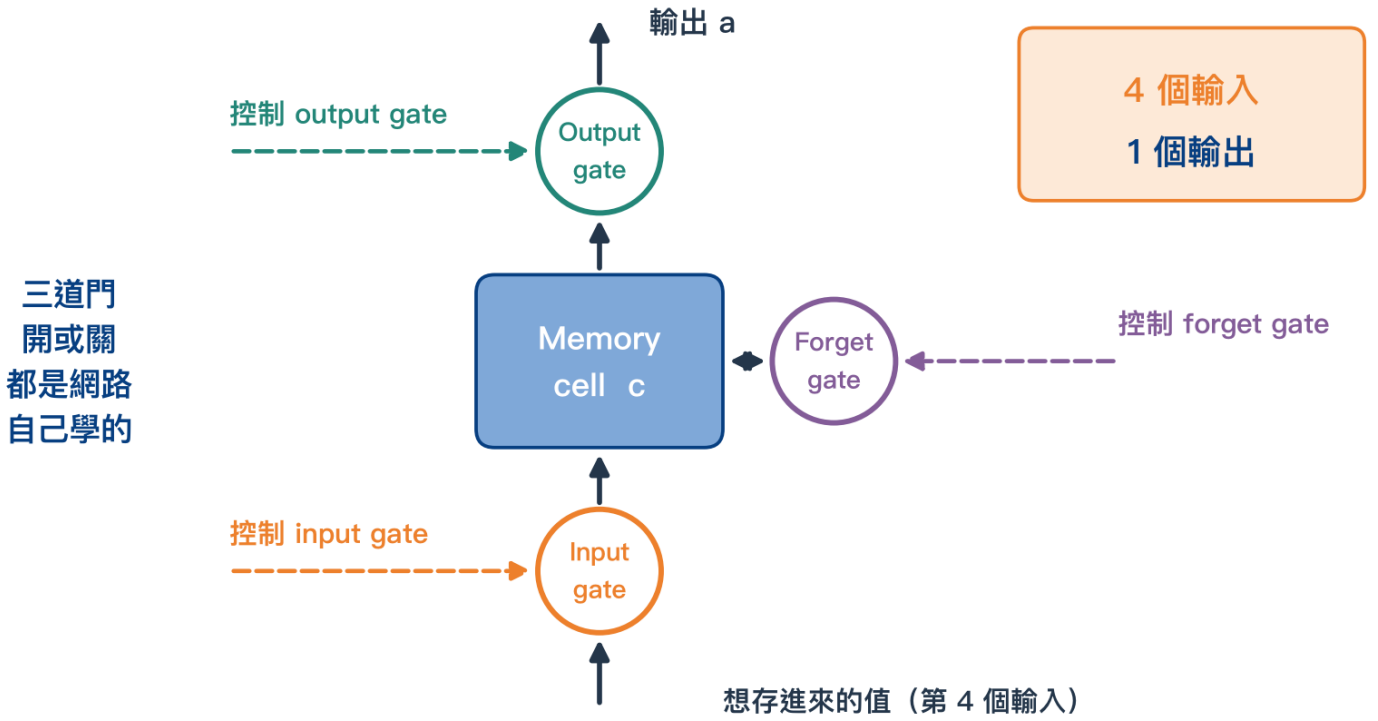
1. 三道門：input、forget、output
2. 公式：門就是 0～1 的乘數
3. 手算一個 LSTM cell

Long short-term memory：比較長的短期記憶。

LSTM cell：三道門、四個輸入

核心觀念

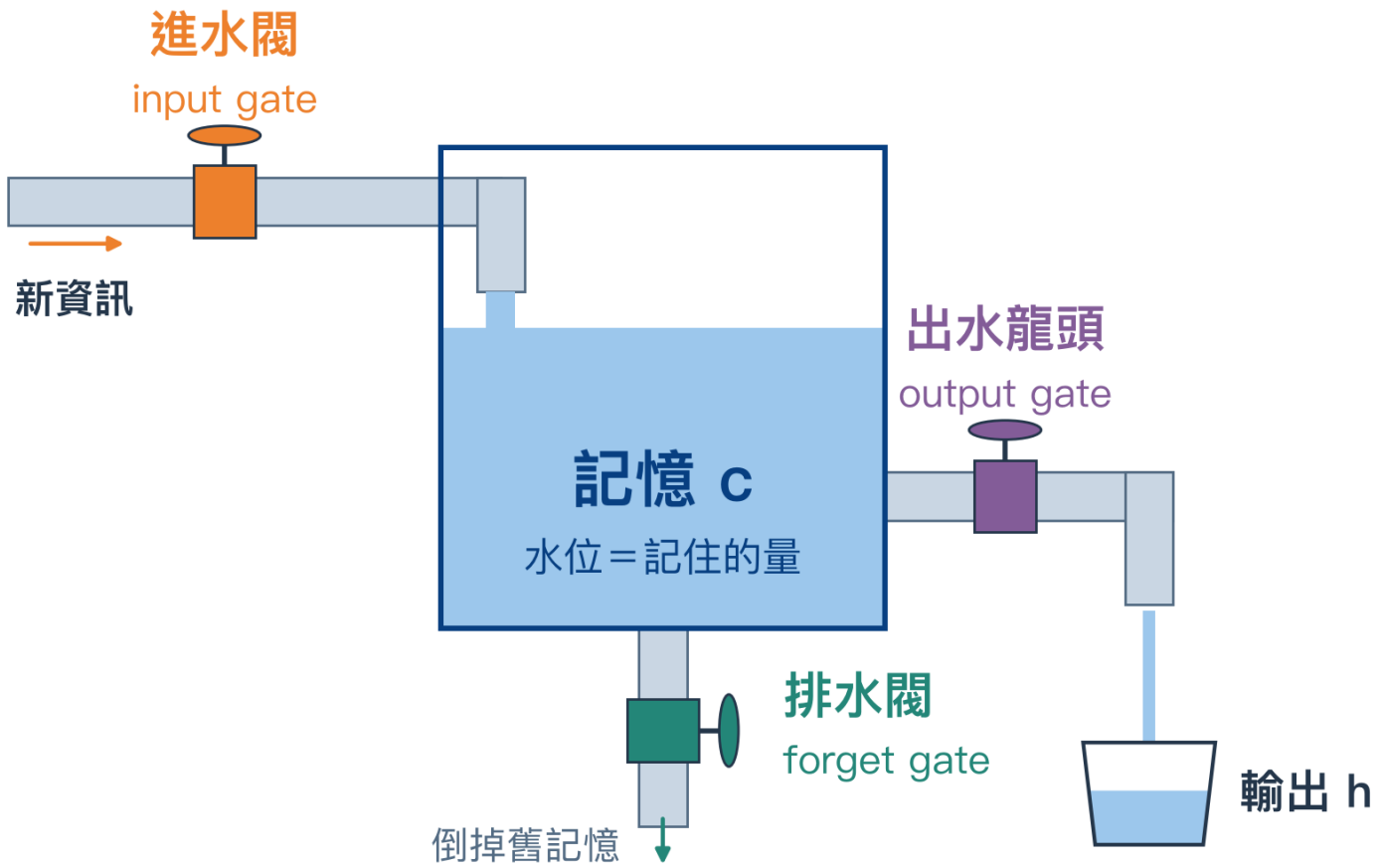
- Input gate：新的值能不能寫進來
- Forget gate：舊的記憶要不要清掉
- Output gate：記憶能不能被讀出去
- 門開或關，都是網路自己學的



4 個輸入（想存的值 + 3 個門的控制訊號），1 個輸出。

類比：一個有三個閥門的水箱

- 水箱 = 記憶 c
- 進水閥 = input gate
- 排水閥 = forget gate
- 出水龍頭 = output gate

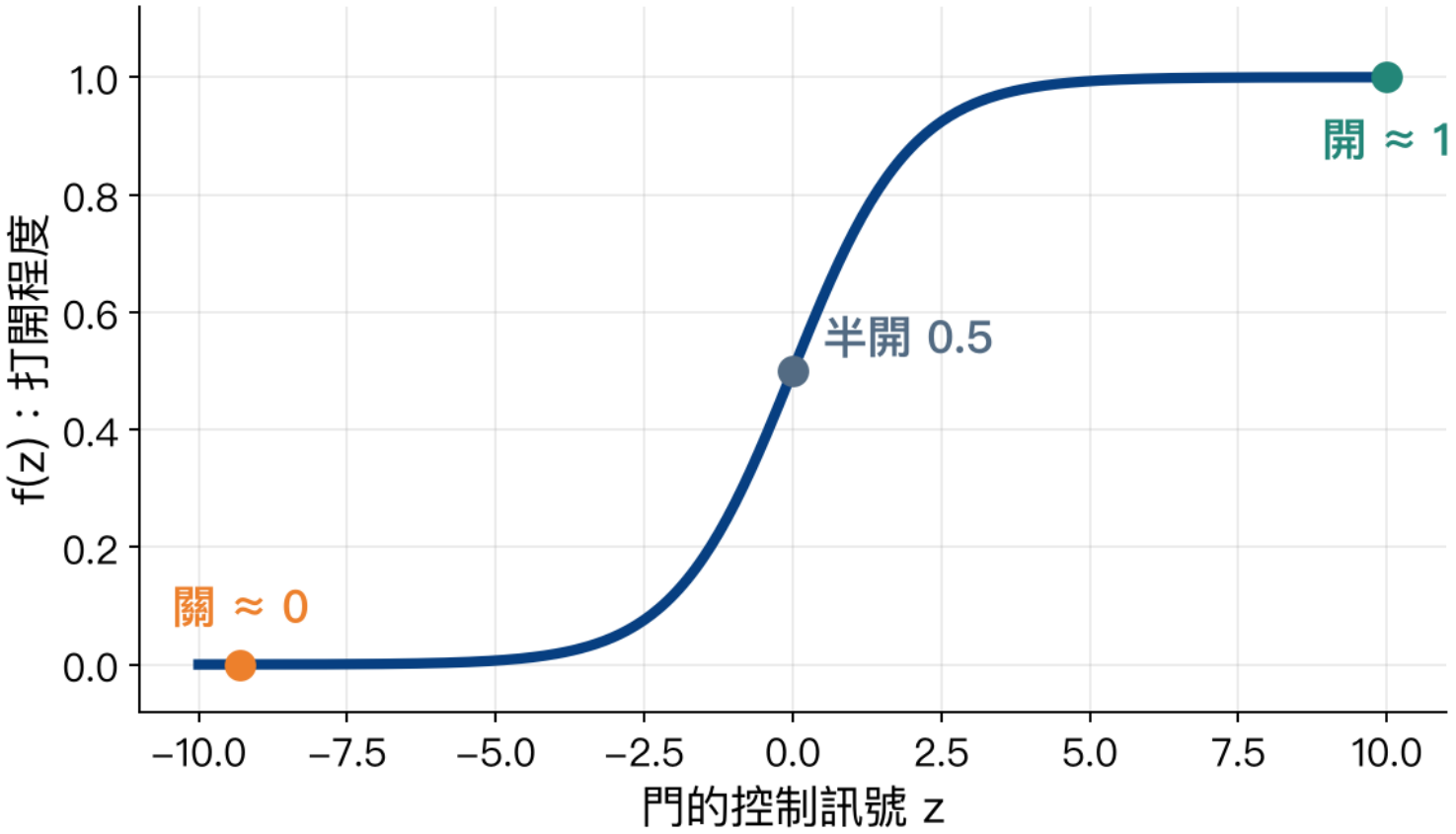


三個閥門各自控制：進水、排水、出水。

門就是 sigmoid：0 到 1 的開關

核心觀念

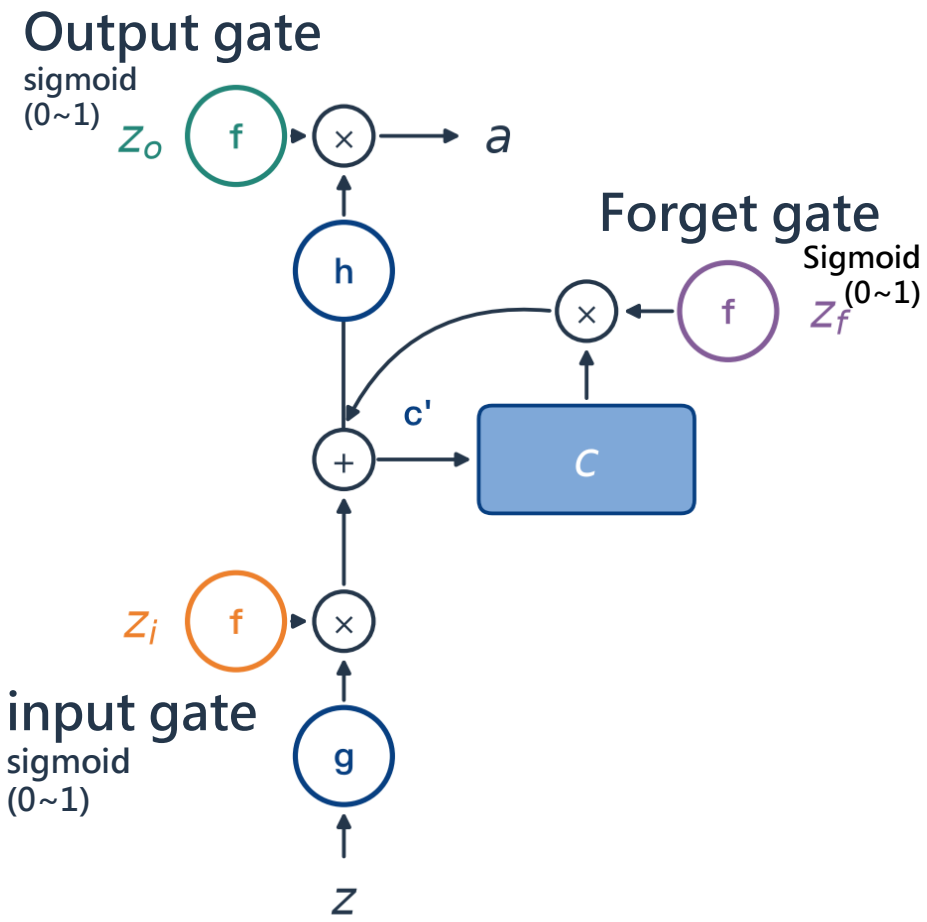
- 門的輸出用 sigmoid（Class 2 學過）
- 接近 1：門打開
- 接近 0：門關上
- 中間值：開一半



門 = 一個 0～1 的數字，拿去乘別的訊號。

LSTM 的公式

公式



寫入與遺忘

$$c' = g(z)\, f(z_i) + c\, f(z_f)$$

讀出

$$a = h(c')\, f(z_o)$$

f 通常是 sigmoid：輸出 0～1＝門打開的程度

f(z_f)=1 是「記得」、=0 是「洗掉」：名字和直覺相反

$c' = g(z)\cdot f(z_i) + c\cdot f(z_f)\ ;\ a = h(c')\cdot f(z_o)$

手算 LSTM：先看規則

手算

	x_1（值）	x_2（寫入 / 清除）	x_3（讀出）	memory	輸出 y
t = 1	3	1	0	3	0
t = 2	4	1	0	7	0
t = 3	2	0	0	7	0
t = 4	1	0	1	7	7
t = 5	3	−1	0	0	0

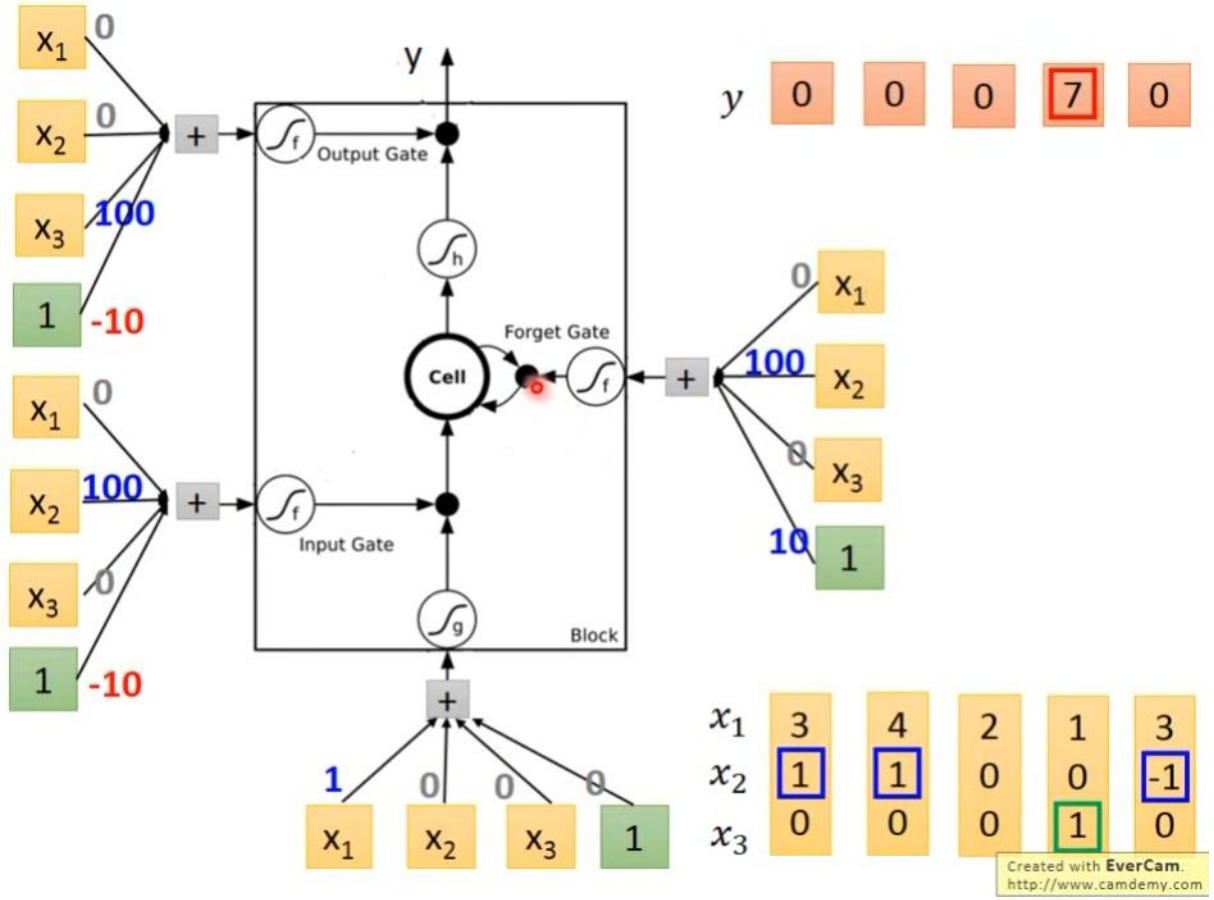
規則：$x_2 = 1$ 把 x_1 加進 memory；$x_2 = -1$ 清空；$x_3 = 1$ 才把 memory 讀出來

$x_2 = 1$ 寫入、$x_2 = -1$ 清空、$x_3 = 1$ 讀出。

手算 LSTM：權重這樣設

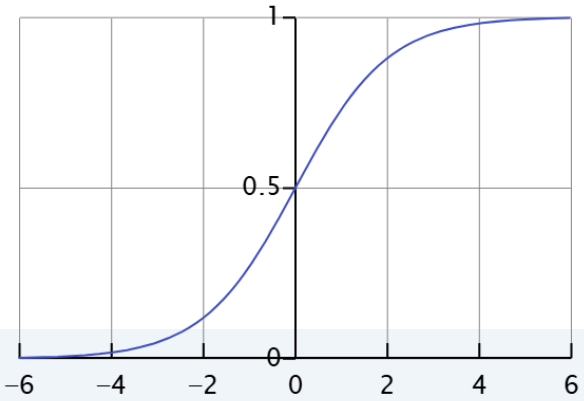
手算

①CNN ②RNN ③LSTM ④實作 ⑤坦克



想存的值 z	$x_1 \times 1$	直接把 x_1 當作要存的值
input gate z_i	$100 \cdot x_2 - 10$	平常關（−10）；$x_2 = 1$ 才打開
forget gate z_f	$100 \cdot x_2 + 10$	平常開（記得）；$x_2 = -1$ 才關（洗掉）
output gate z_o	$100 \cdot x_3 - 10$	平常關；$x_3 = 1$ 才打開

這些權重是梯度下降從資料學來的；這裡先假設已經學好，看它怎麼運作

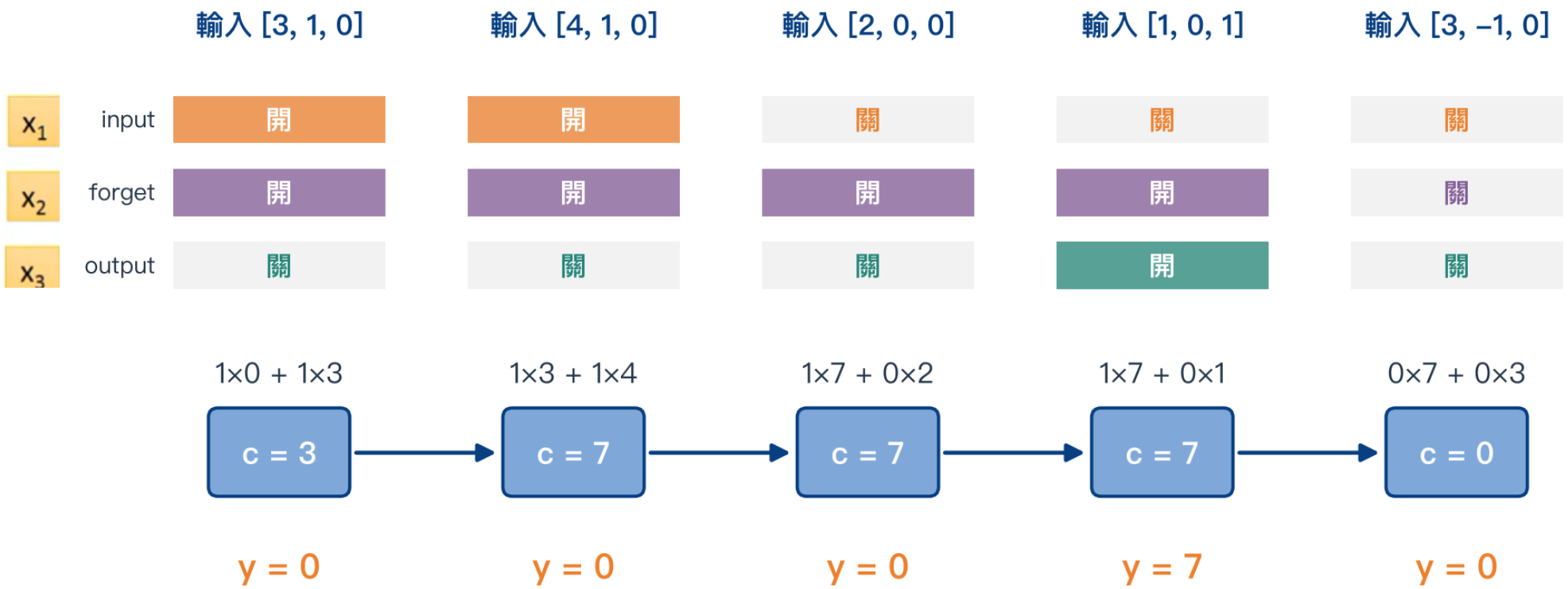


sigmoid

bias −10 讓門平常關；bias +10 讓 forget gate 平常開。

手算 LSTM：一步一步算

手算



g、h 取線性；sigmoid(90) ≈ 1、sigmoid(−10) ≈ 0（圖中四捨五入）

3 → 7 → 7 → 讀出 7 → 清成 0：和規則完全一樣。

想一想⑥

手算

在第 6 步再輸入 [5, 1, 1]，輸出是多少？

A　0

B　5

C　12

	x_1（值）	x_2（寫入 / 清除）	x_3（讀出）	memory	輸出 y
t = 1	3	1	0	3	0
t = 2	4	1	0	7	0
t = 3	2	0	0	7	0
t = 4	1	0	1	7	7
t = 5	3	−1	0	0	0

規則：$x_2 = 1$ 把 x_1 加進 memory；$x_2 = -1$ 清空；$x_3 = 1$ 才把 memory 讀出來

先看 memory 現在是多少，再看三個門。

想一想⑥：解答

手算

在第 6 步再輸入 [5, 1, 1]，輸出是多少？

A 0

B 5 ✓

C 12

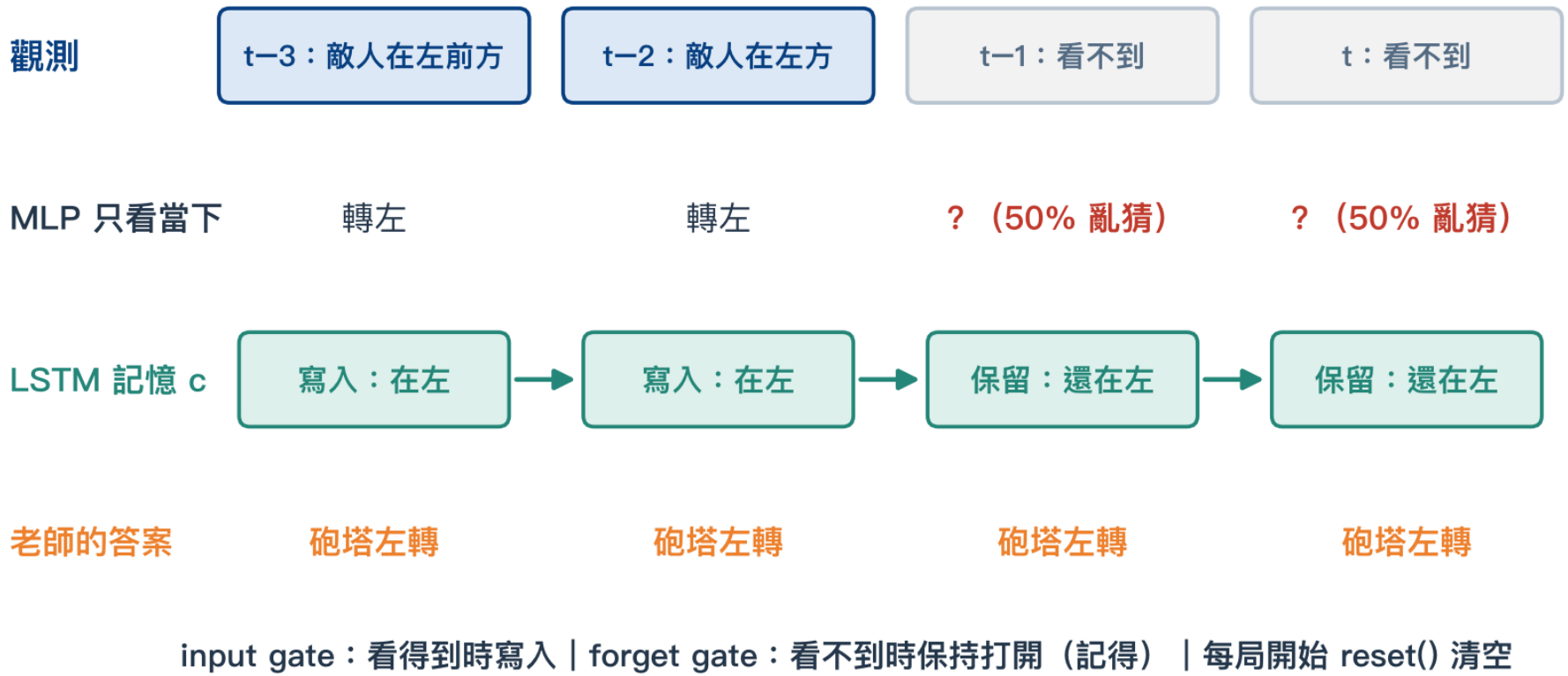
memory 已在第 5 步清成 0；$x_2 = 1$ 把 5 寫入 → c = 5；$x_3 = 1$ 打開 output gate → 輸出 5。

	x_1（值）	x_2（寫入 / 清除）	x_3（讀出）	memory	輸出 y
t = 1	3	1	0	3	0
t = 2	4	1	0	7	0
t = 3	2	0	0	7	0
t = 4	1	0	1	7	7
t = 5	3	−1	0	0	0

規則：$x_2 = 1$ 把 x_1 加進 memory；$x_2 = -1$ 清空；$x_3 = 1$ 才把 memory 讀出來

memory 已在第 5 步清成 0；$x_2 = 1$ 把 5 寫入 → c = 5；$x_3 = 1$ 打開 output gate → 輸出 5。

LSTM 在坦克裡：記住最後看到的敵人



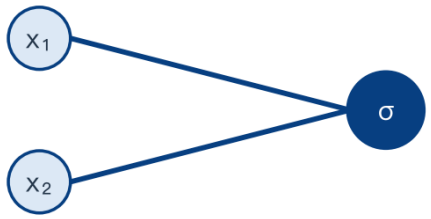
input gate：看得到時寫入｜forget gate：看不到時保持打開（記得）｜每局開始 reset() 清空

看得到時寫入、看不到時保留、每局開始清空：三道門剛好對應坦克的需要。

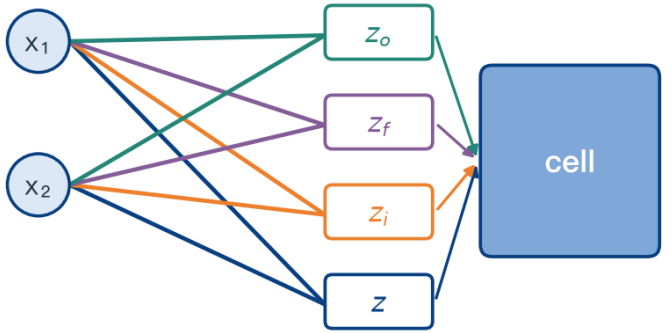
LSTM 和一般神經元的關係

- 把一個神經元換成一個 LSTM cell
- 輸入 x 乘 4 組不同的權重
- 分別控制 z、z_i、zf、z_o
- 參數量約是一般網路的 4 倍

一般神經元：1 組權重



LSTM cell：4 組權重



同樣神經元數，LSTM 參數約是一般網路的 4 倍（就像要插 4 條電源線才能跑）

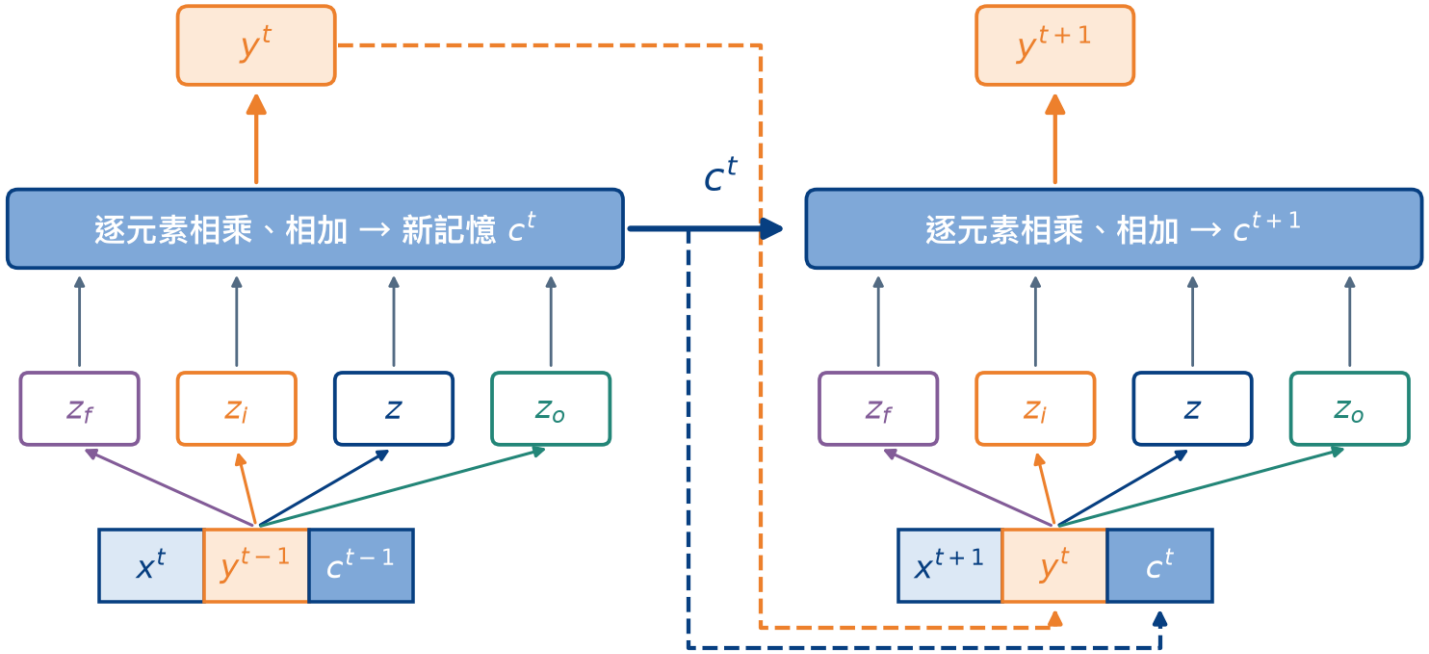
一般神經元插 1 條電源線；LSTM 要插 4 條。

整排 LSTM：用向量一起算

圖解｜課後閱讀（放映略過）

下一步的四個門看的是 $[x^{t+1}, y^t, c^t]$ 接成的一個向量

y^t 接回來（橘虛線）；peephole 把 c^t 也接回來（藍虛線）



$[x^{t+1}, y^t, c^t]$ 接成一個向量 → 乘 4 個矩陣 → 4 個門 → 新記憶 c^{t+1}。

實務：一行程式就有 LSTM

Python

- PyTorch：nn.LSTM(輸入維度, hidden)
- Keras：layers.LSTM(hidden)
- GRU：兩個門，參數少 1/3，表現相近

名稱	門	記憶	參數（相同 hidden）
Simple RNN	0	每步洗掉	1 倍
LSTM	3	可長期保留	約 4 倍
GRU	2	合併記憶	約 3 倍

```
lstm = nn.LSTM(input_size=5, hidden_size=8, batch_first=True)
out, (h, c) = lstm(X)        # X.shape = (樣本數, T, 5)
pred = head(out[:, -1])       # 只用最後一天的輸出
```

前面複雜的運算，程式裡就是一行。

三個網路的對照

整理

CNN 空間

- 看一小塊（receptive field）
- 不同位置共用 filter
- pooling 縮小

適合：圖片、棋盤

RNN 時間

- memory 存上一步的 hidden
- 不同時間共用權重
- 記憶每步被洗掉

適合：短句、短序列

LSTM 時間 + 選擇

- 三道門控制寫入、遺忘、讀出
- 記憶可以保留比較久
- 參數約 4 倍

適合：長一點的序列

共同點：權重共享。差別：在空間共享，還是在時間共享。

第四段　實作：LSTM 能贏 Class 2 的 SVR 嗎？

章節導覽

上一段

Class 2：台積電乖離率，SVR、Ridge、神經網路都很難贏「價格不動」。

這一段要回答

加上時間記憶，會不會更好？記憶真的有用到嗎？



1. 回顧 Class 2 的題目與基準
2. 公平比較 SVR 與 LSTM
3. 檢查：LSTM 真的用到歷史了嗎？

同一份資料、同一個答案、同一批日期：只換模型。

回顧 Class 2：題目與基準

複習

①CNN ②RNN ③LSTM ④實作 ⑤坦克

error ratio = MAE(model) / MAE(baseline)

- 答案：今天收盤價 vs 7 天後的 15 日均線（%）
- 基準②價格不動：未來 7 天都填今天收盤價
- 誤差比 = 模型 MAE ÷ 基準 MAE，< 1 才算贏
- 這次訓練資料從 1 年加到 3 年

切分	期間	天數	基準 MAE
訓練	2021–2023	722	—
驗證	2024	235	1.28
測試	2025	236	1.10

比的不是分數高低，是有沒有贏過「什麼都不做」。

波動區間怎麼選

資料

①CNN ②RNN ③LSTM ④實作 ⑤坦克



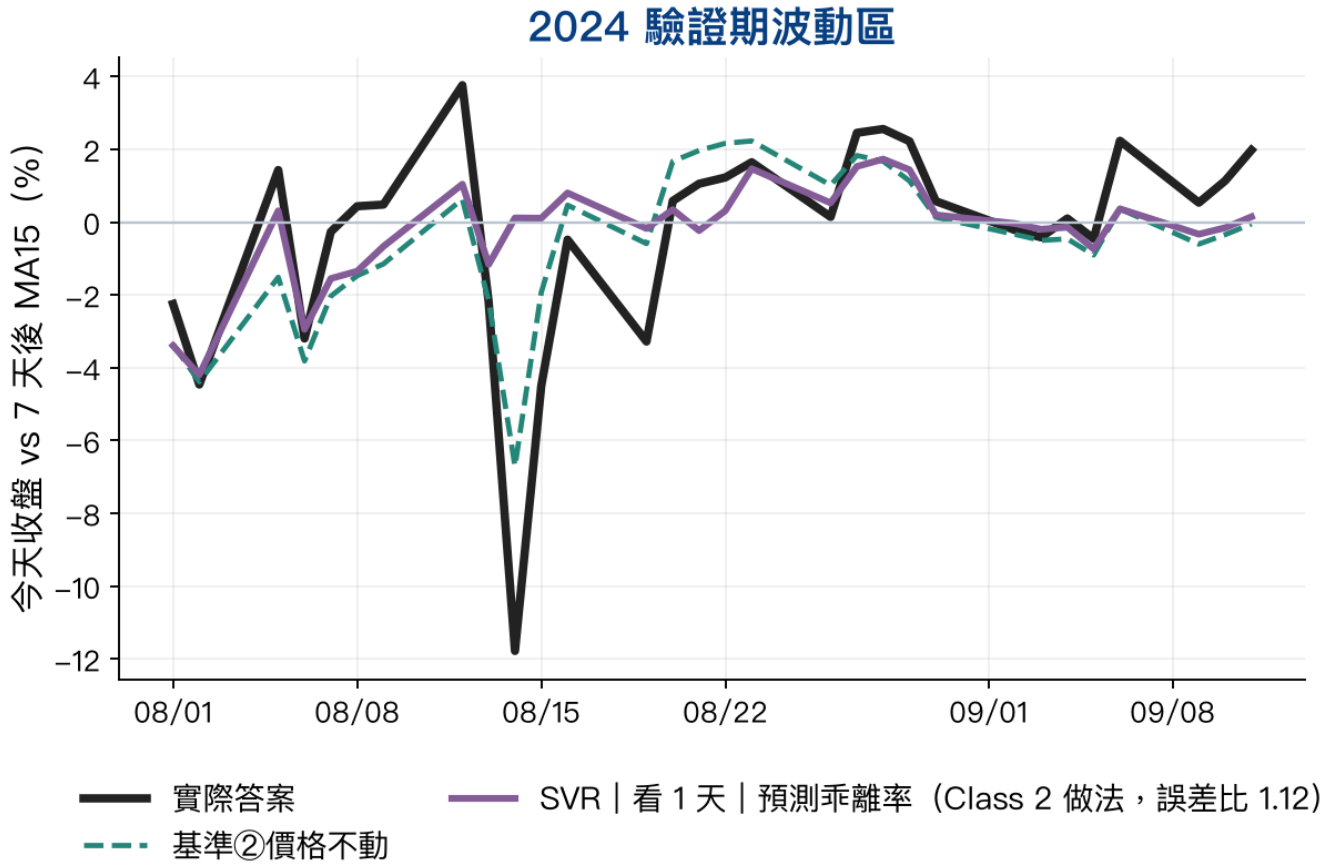
每年挑「過去 20 天波動」最高的連續 30 天：07-19～09-02（2024）、04-07～05-19（2025）

先看 Class 2 的做法：遇到大波動就失準

結果

①CNN ②RNN ③LSTM **④實作** ⑤坦克

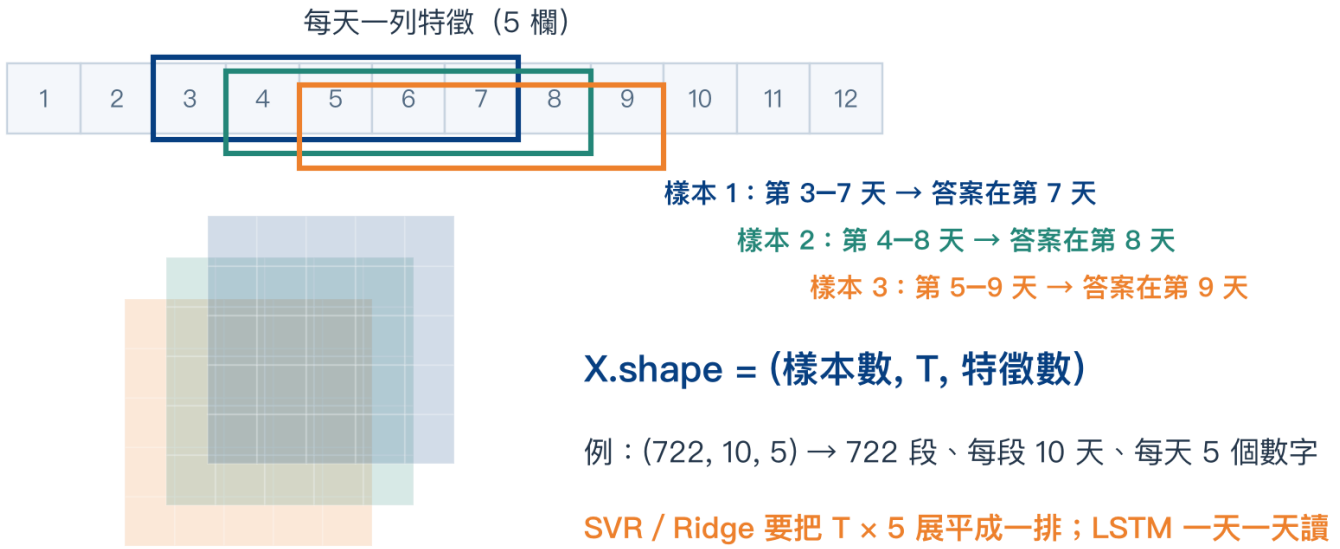
- SVR | 看 1 天 | 預測乖離率
- 2024 全年：誤差比 1.10
- 2024 波動區：1.12
- 2025 波動區：1.35



Class 2 的做法四段都輸基準，2025 波動區差 35%。

把歷史排成 LSTM 的輸入

- 看 10 天 = 最近 10 個交易日，每天 5 個特徵
- X.shape = (樣本數, 10, 5)
- SVR：把 10 × 5 攤平成 50 個欄位
- LSTM：一天一天讀，記憶往下傳

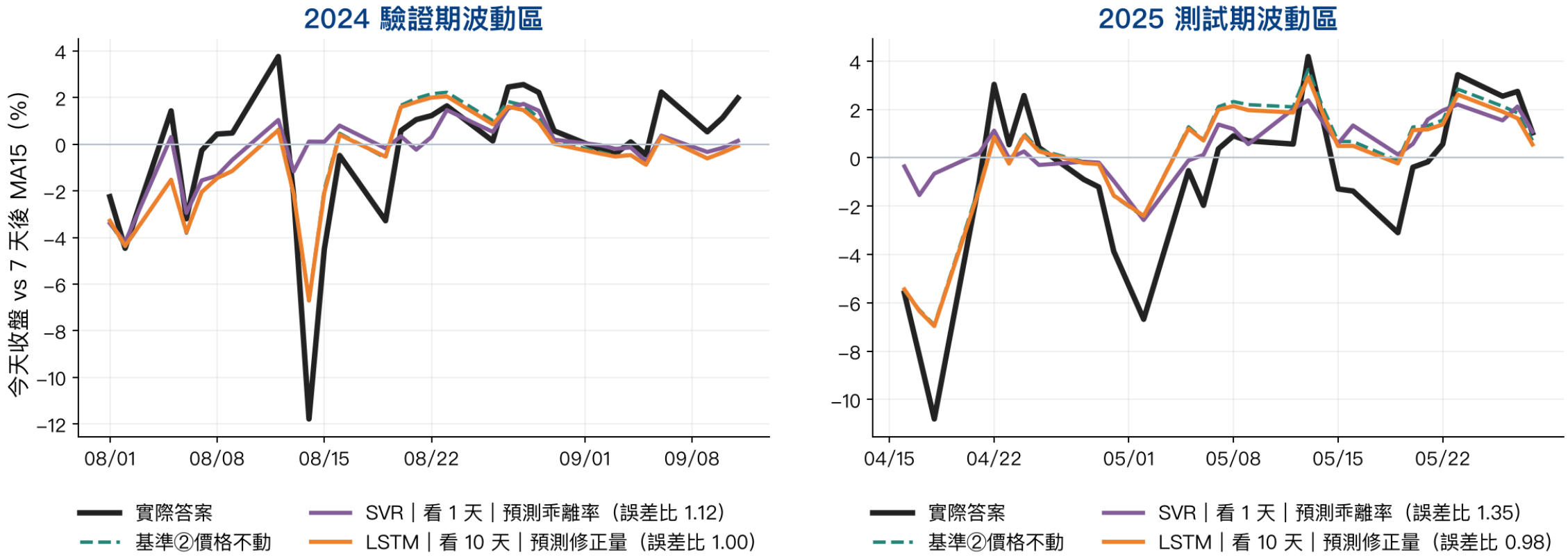


公平比較：兩邊都看同樣 10 天，只差「怎麼讀」。

波動區的預測曲線

結果

①CNN ②RNN ③LSTM ④實作 ⑤坦克



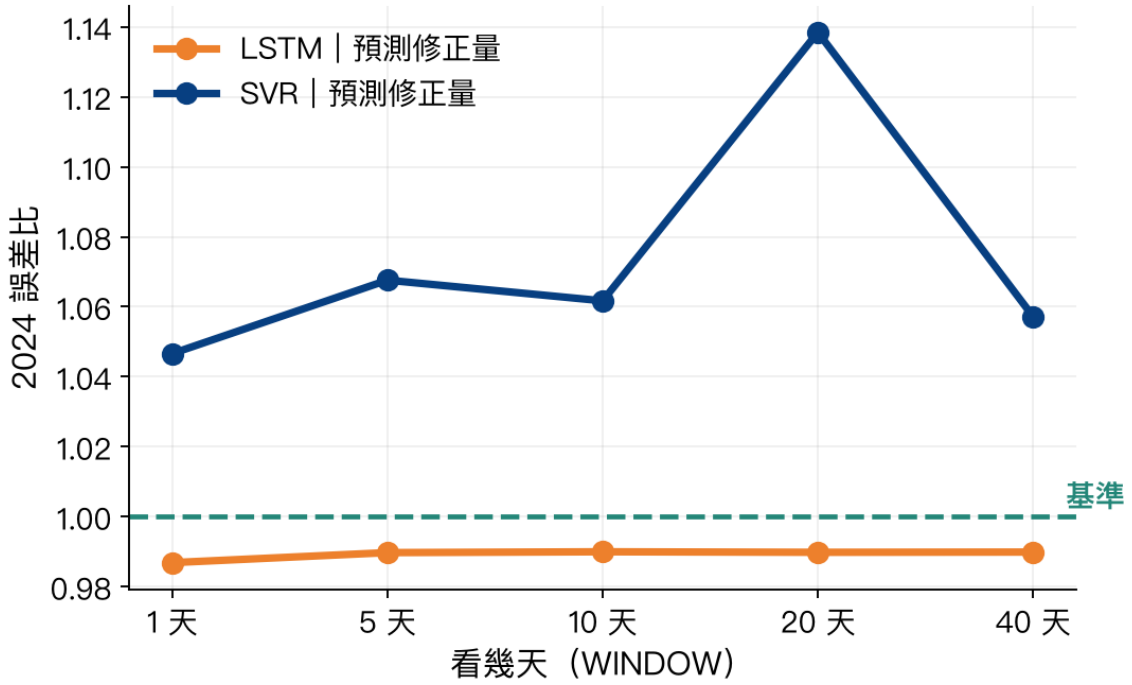
橘線（LSTM）幾乎貼著綠虛線（基準）：它幾乎沒有修正。

檢查：LSTM 真的用到記憶了嗎？

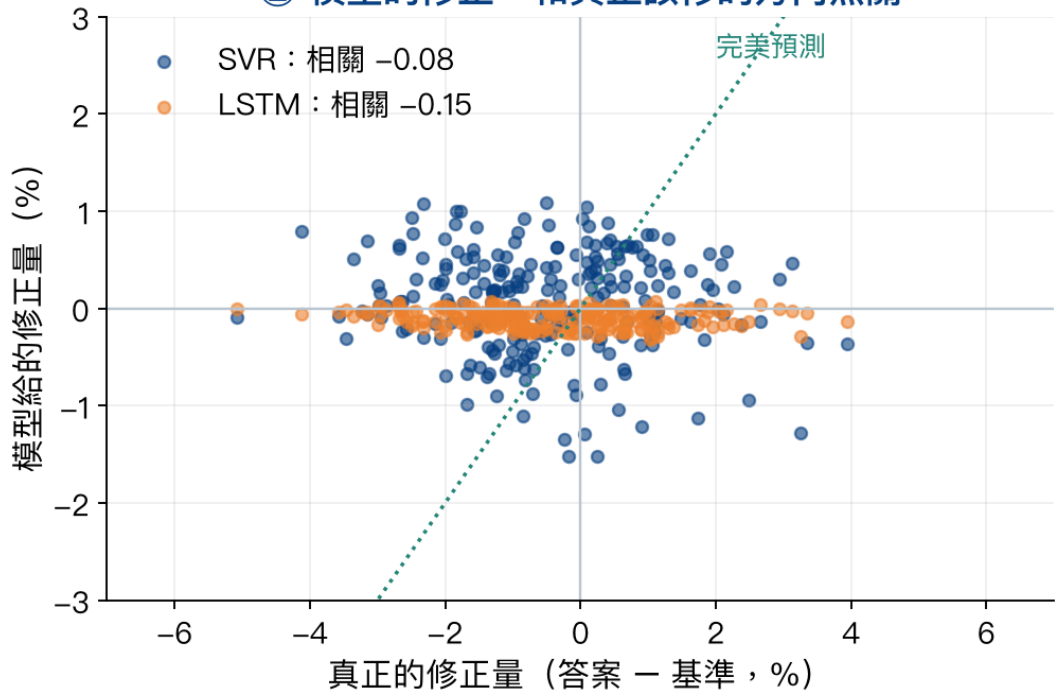
診斷

①CNN ②RNN ③LSTM ④實作 ⑤坦克

① 看 1 天和看 40 天，LSTM 幾乎一樣



② 模型的修正，和真正該修的方向無關

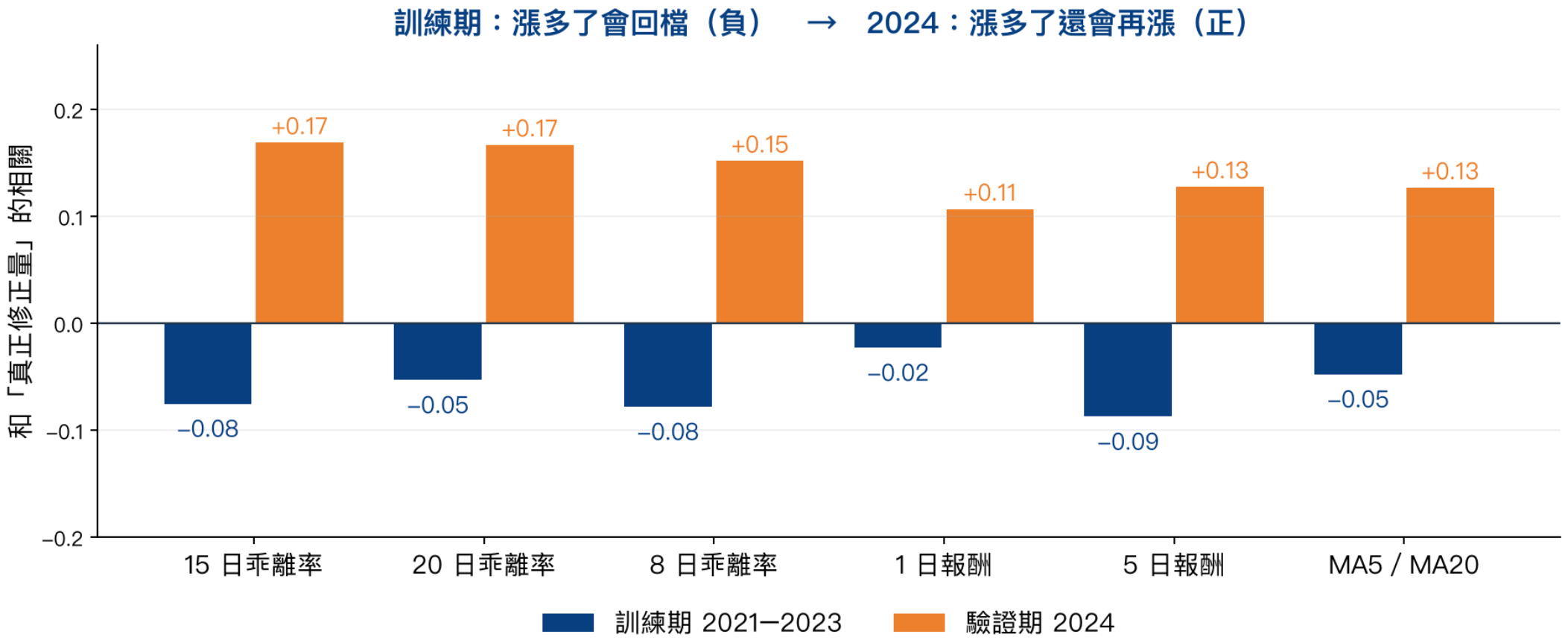


看 1 天、40 天結果一樣；給的修正量和真正該修的方向無關（相關 -0.15）。

為什麼學不到？2024 的規律反過來了

診斷

①CNN ②RNN ③LSTM ④實作 ⑤坦克



訓練期學到「漲多會回檔」，2024 卻是「漲多還會再漲」：從過去學到的規律，用不到未來。

實作 B：台積電控制台

▶ 動手做・打開 Colab

動手做

①CNN ②RNN ③LSTM ④實作 ⑤坦克

建議 25 分鐘　Colab 步驟

- [ ] B1 看波動區間（可自己挑日期）
- [ ] B2 跑 Class 2 SVR，當作要打敗的對象
- [ ] B3 控制台：調好預設 LSTM；可加 KD、RSI、MACD
- [ ] B4 疊圖比較，選一個回合做最後測試



完成標準

驗證誤差比 < 1，並用 🔍 診斷說明：是真的學到，還是幾乎沒修正

一次只改一個旋鈕；每個回合寫下你改了什麼。

控制台的旋鈕：每一個都對應課程

動手做

①CNN ②RNN ③LSTM **④實作** ⑤坦克

- TARGET：預測乖離率 → 預測修正量
- WINDOW 1 vs 40：結果變了嗎？（記憶檢查）
- HORIZON：預測 1、3、7、14 天後
- TRAIN_FROM：只用 2023 vs 2021 起
- 早停、L2、DROPOUT；換 MODEL 比一比

旋鈕	對應	想一想
TARGET	預測乖離率 / 修正量	為什麼差這麼多？
WINDOW	看幾天	1 vs 40 天？
HORIZON、TASK	幾天後 / 股價 / 漲跌	換題目會變好嗎？
TRAIN_FROM	訓練資料	1 年夠嗎？
特徵	KD、RSI、MACD、自己造	兩期同號嗎？
MODEL	LSTM / RNN / SVR	記憶有幫助嗎？
早停、L2	Class 2 泛化	何時分岔？

每個旋鈕都是學過的觀念：今天只是把它們用在序列上。

所以，LSTM 有比較好嗎？

解讀

①CNN ②RNN ③LSTM ④實作 ⑤坦克

- 預測乖離率：SVR、LSTM 都輸基準
- 預測修正量：LSTM 好 1%，但不是記憶的功勞
- 股價歷史裡可學的線索太少，換模型救不了
- 記憶要有用，歷史裡要真的有答案（坦克）

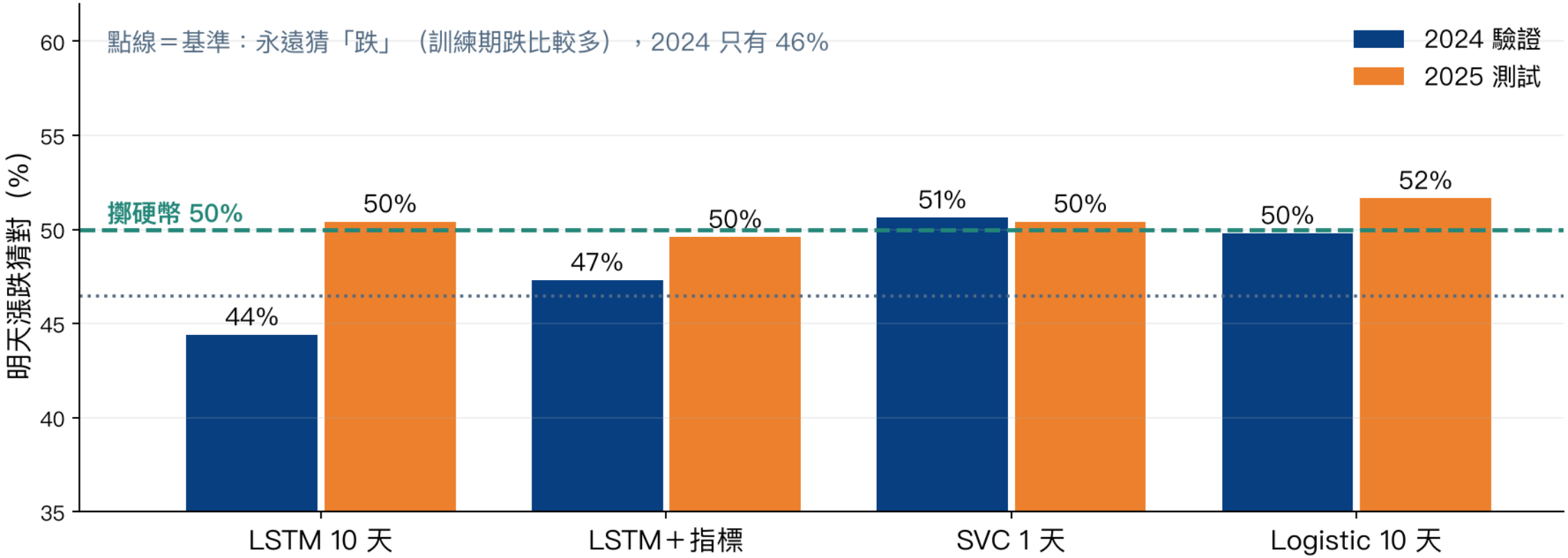
誤差比	2024 全年	2024 波動	2025 全年	2025 波動
SVR | 1 天 | 乖離率	1.10	1.12	1.13	1.35
LSTM | 20 天 | 乖離率	1.04	1.12	1.04	1.16
SVR | 10 天 | 修正量	1.06	0.96	1.07	0.92
LSTM | 10 天 | 修正量	0.99	1.00	0.99	0.98

模型越複雜，越要檢查它是不是真的學到東西：分數好 1% 不等於有用。

預測明天漲或跌：和擲硬幣差不多

換個問法

①CNN ②RNN ③LSTM ④實作 ⑤坦克

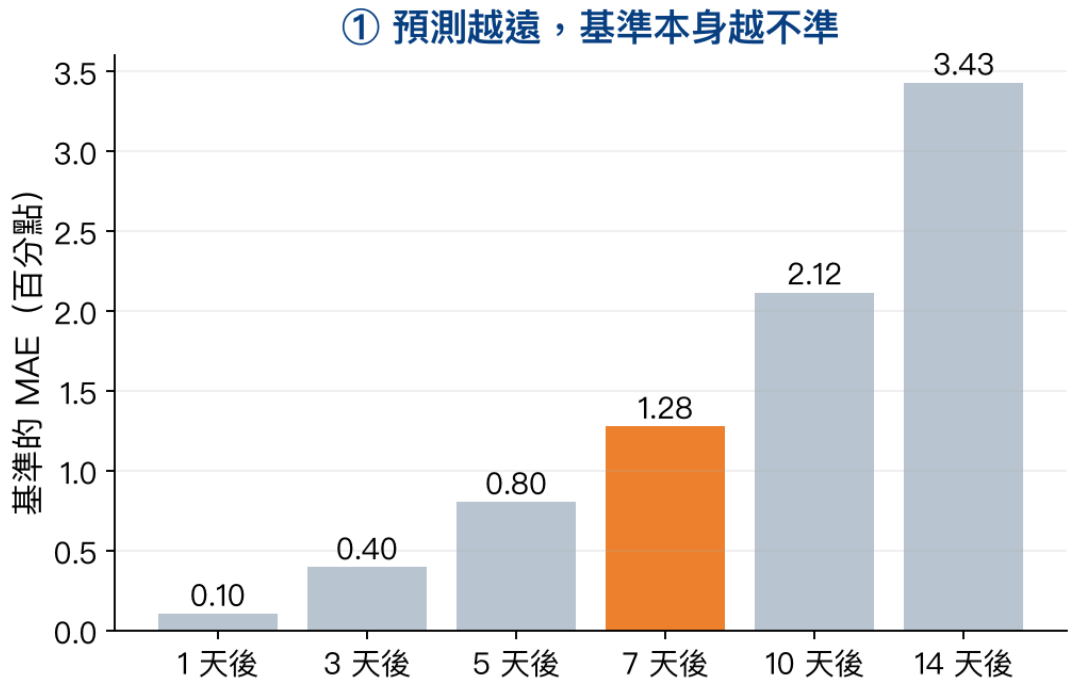


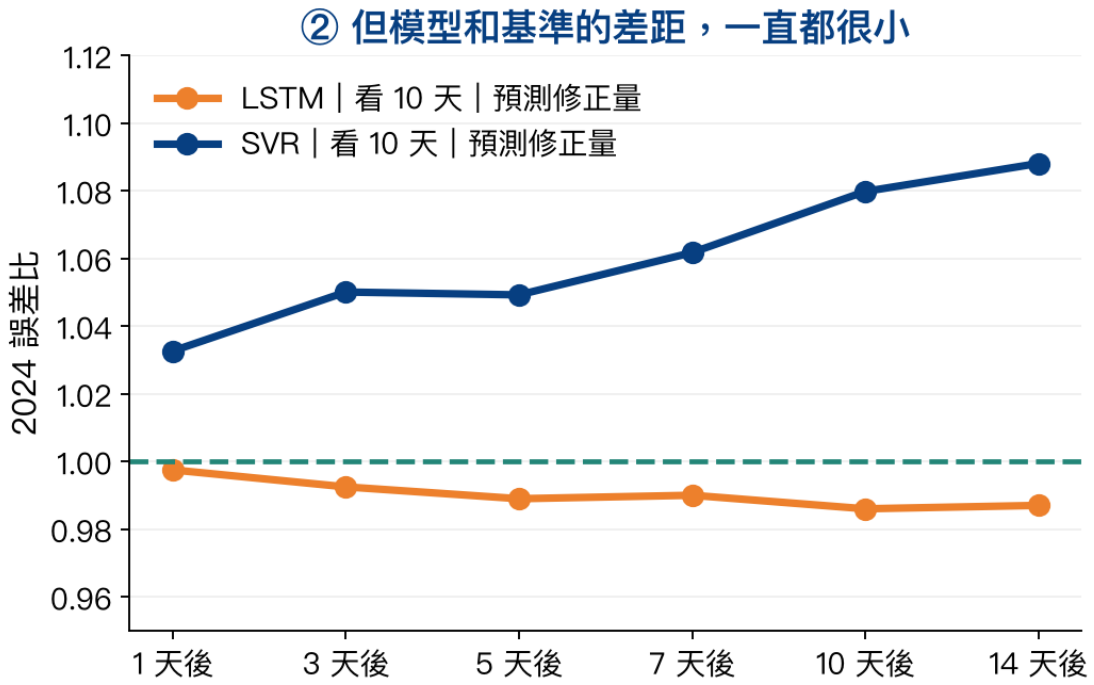
改成分類（Class 2 的 sigmoid + 交叉熵）也一樣：正確率都在 50% 附近。

練習：預測幾天後？1 天到 14 天

動手做

①CNN ②RNN ③LSTM ④實作 ⑤坦克





預測越遠，基準越不準；但模型和基準的差距一直很小：Colab 用 HORIZON 自己試。

第五段　期中專題：坦克競技場

章節導覽

①CNN　②RNN　③LSTM　④實作　⑤坦克

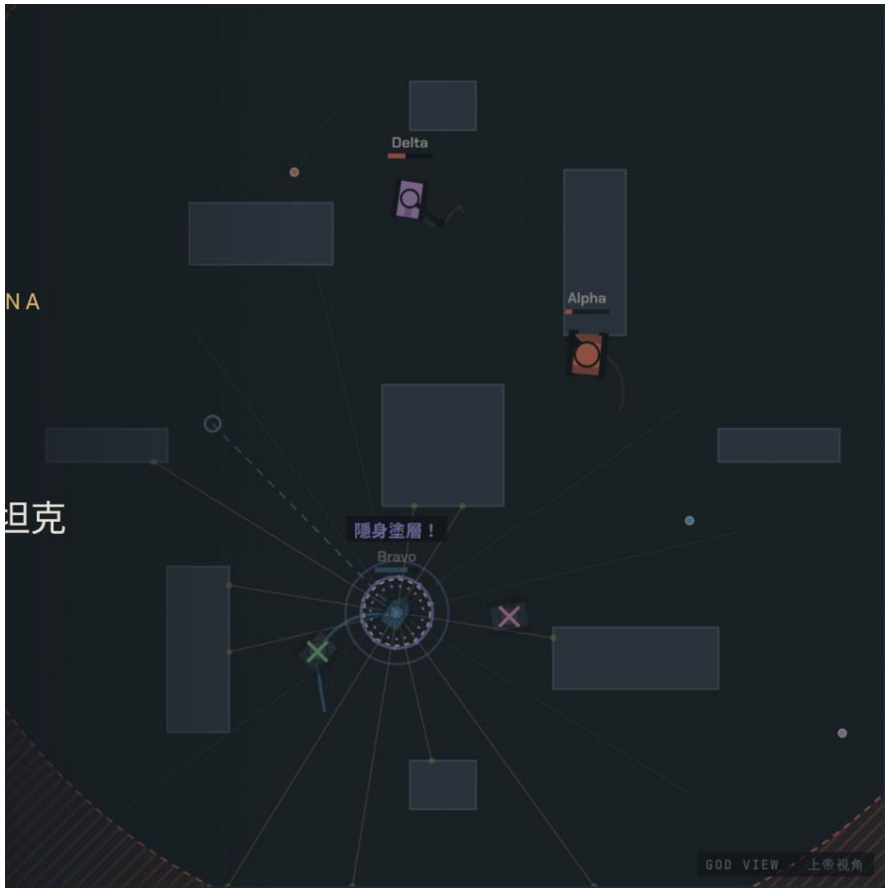
上一段

股票：用過去 T 天預測未來；LSTM 記得看不到的事。

這一段要回答

用神經網路，訓練一台會自己打仗的坦克。

1. 四步流程、四週時程
2. 看到什麼、能做什麼
3. 兩個範例：疊幀 MLP 與 LSTM



比賽時上場的不是你寫的規則，而是學會規則的神經網路。

期中要做什麼：四步、四週

期中

①CNN ②RNN ③LSTM ④實作 ⑤坦克

1

寫規則

用 if-else 寫戰術，當「老師」

2

錄資料

老師打上千局，記下「看到什麼、做了什麼」

3

訓練網路

用分類模型學會老師的決定

4

上場比賽

只靠神經網路，和其他組對戰

四週時程

W1 提案：選機體與模組、一頁策略卡　W2 跑通 baseline　W3 期中檢查：至少一項改進 + 前後對照　W4 繳交與決賽（加入沒看過的地圖）

寫規則 → 錄資料 → 訓練 → 上場：這就是模仿學習（behavior cloning）。

坦克看到什麼、能做什麼

期中

①CNN ②RNN ③LSTM ④實作 ⑤坦克

輸入 x：91 個數字（只有同一時刻、以自己車頭為準）



地形射線 ×32

每 11.25° 一條
到牆或場邊的距離

敵人 ×4

看得到的最近 4 台
距離、方向、速度

子彈 ×4

最近 4 顆飛來的子彈

自身 ×15

血量、冷卻、圈心方向、技能

輸出 y：5 個分類頭（Class 2 的分類，一次做 5 個）

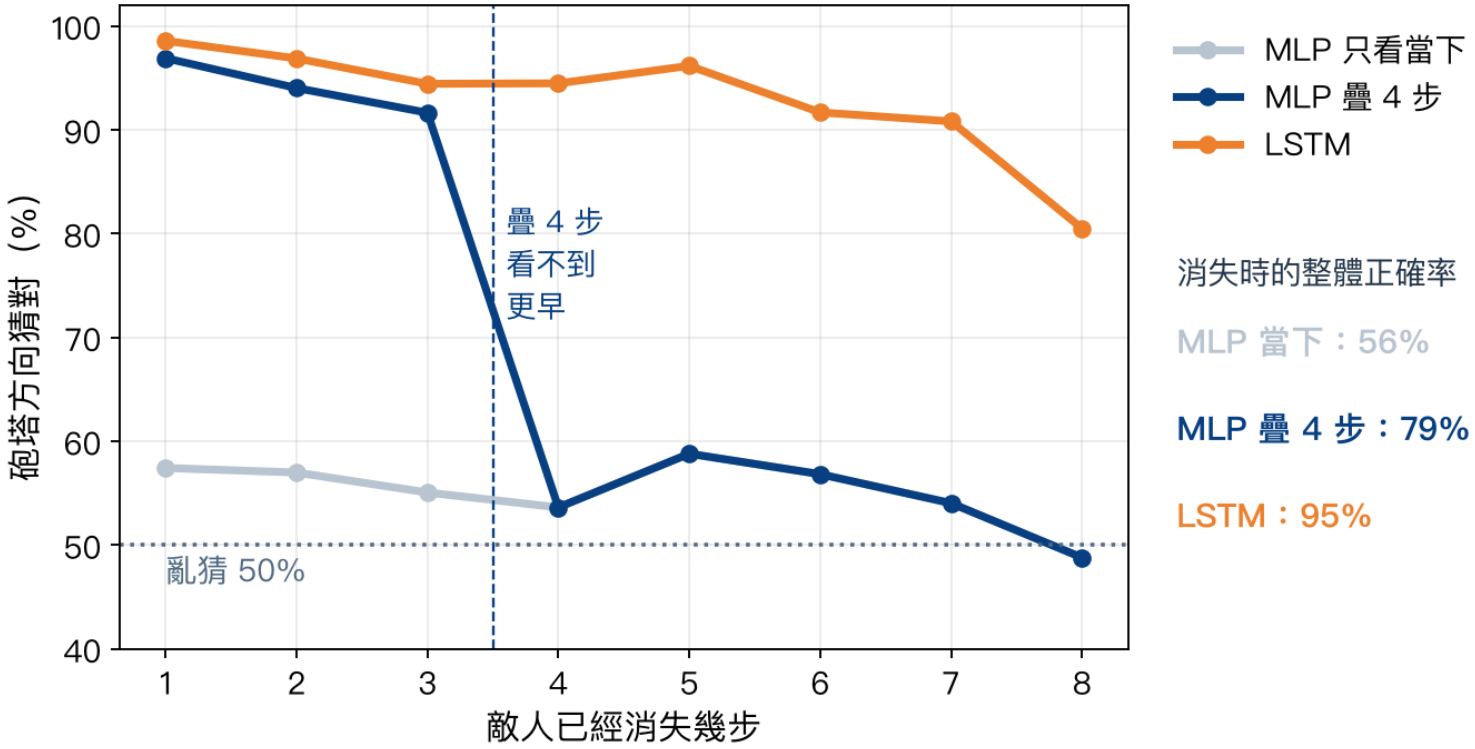
移動	車身	砲塔	開火	技能
前進 / 後退 / 停止	左轉 / 不動 / 右轉	左轉 / 不動 / 右轉	否 / 是	不用 / 使用

輸入 91 個數字、輸出 5 個分類頭：就是 Class 2 的分類，一次做 5 個。

動手做：敵人躲起來時，記憶有用嗎？

動手做（Colab C）

- 合成資料：敵人有時躲到牆後
- 只看當下：消失後只能亂猜
- 疊 4 步：消失 3 步內還記得
- LSTM：消失 8 步仍有 8 成



疊幀只記得固定幾步；LSTM 的記憶可以留得比較久。

繳交與這週就開始

期中

①CNN ②RNN ③LSTM ④實作 ⑤坦克

- 確認規格書，跑 example_random.py 確認連線
- 照兩個範例的 README 跑一次完整流程
- 決定機體與模組
- 寫1-2頁心得報告 (圖文佳)：戰術、配裝理由、怎麼驗證、怎麼贏

繳交	注意
agent.py	只吃 91 維觀測；每步 10 ms 內、只用 CPU
model.pt	正規化參數跟模型一起存
train/	老師規則、訓練程式（比賽不執行）
report.pdf	心得報告

每局開始呼叫 reset()：有記憶的網路要在這裡清空。

本堂重點

總結

CNN	只看一小塊、不同位置共用 filter、pooling 縮小；限制換來少參數。
RNN	memory 讓同樣輸入有不同輸出；不同時間共用權重。
LSTM	三道 0~1 的門控制寫入、遺忘、讀出；參數約 4 倍。
實作	股價歷史裡可學的線索很少：LSTM 好 1% 不是記憶的功勞；分數好時先檢查模型學到什麼。
期中	老師看全局、坦克只看觀測：看不到的資訊靠疊幀或 LSTM 補。

模型越複雜，越要用公平的比較來證明它真的有幫助。