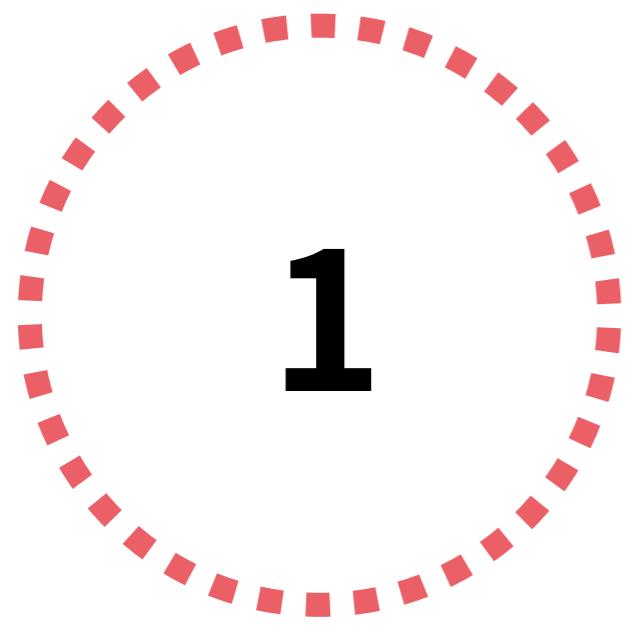




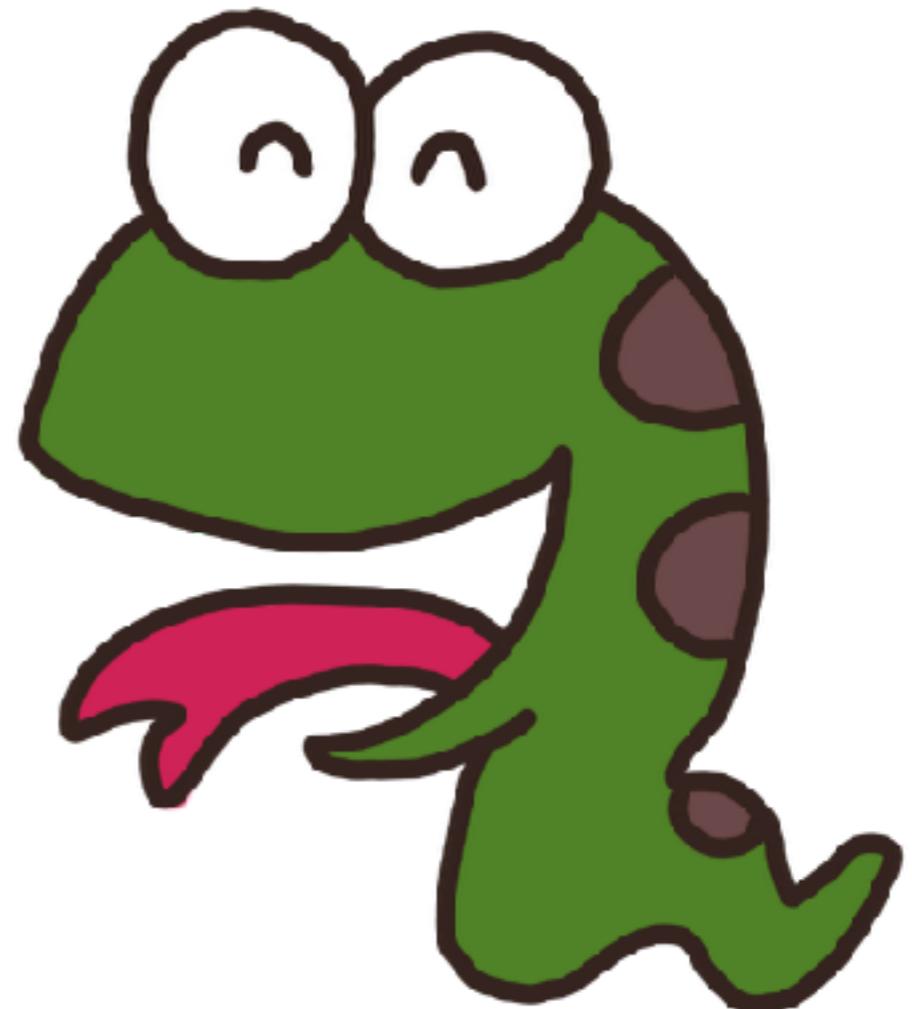
人工智能學校

RNN

蔡炎龍 政治大學應用數學系

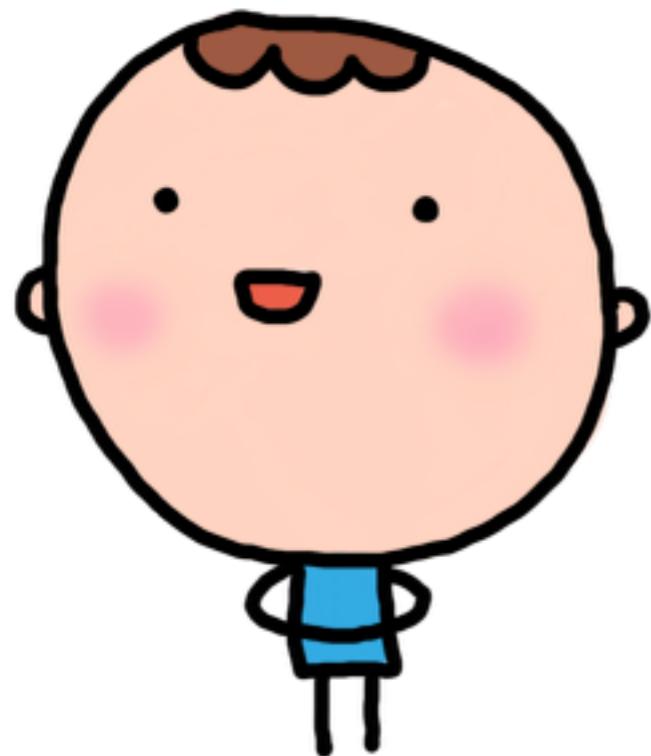


# 三大 神經網路

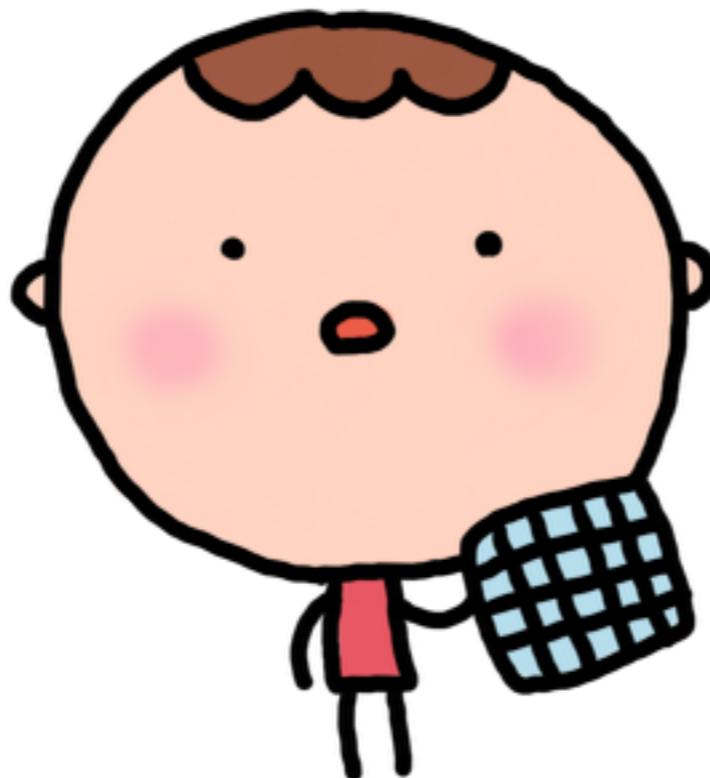


# Deep Learning

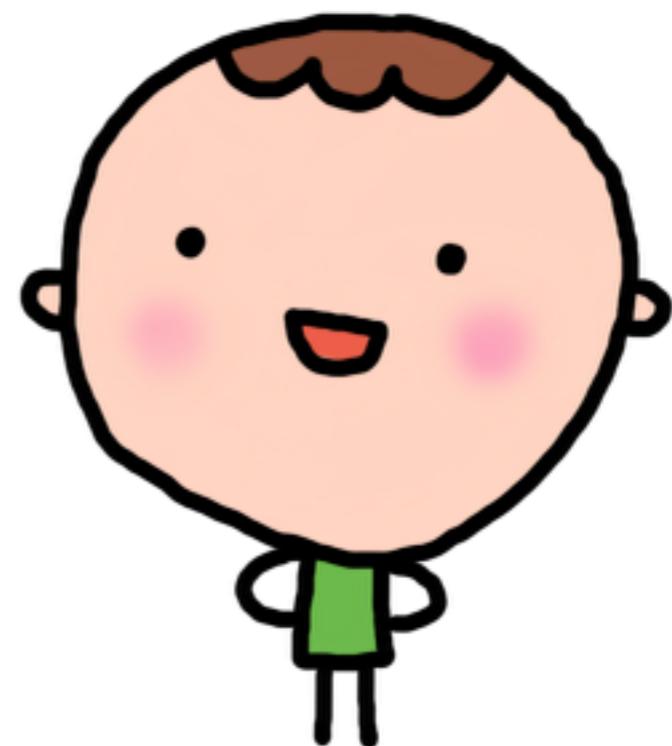
## 三大天王



標準 NN



CNN



RNN

# 真心不騙



eder @edersantana · 1天

When you gotta hustle selling neural nets  
in China. Already profitable AI startup!



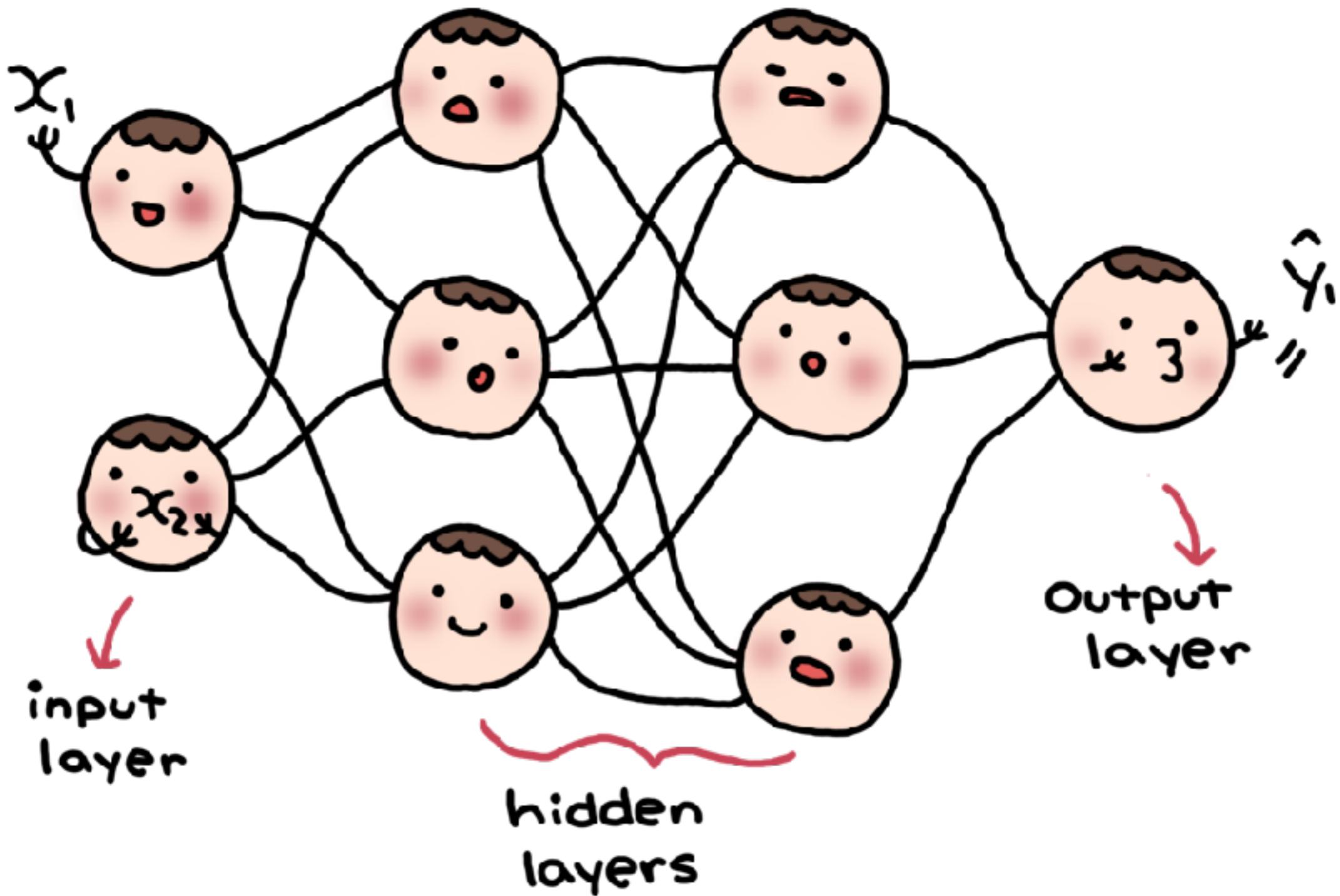
◀ 6

↑↓ 111

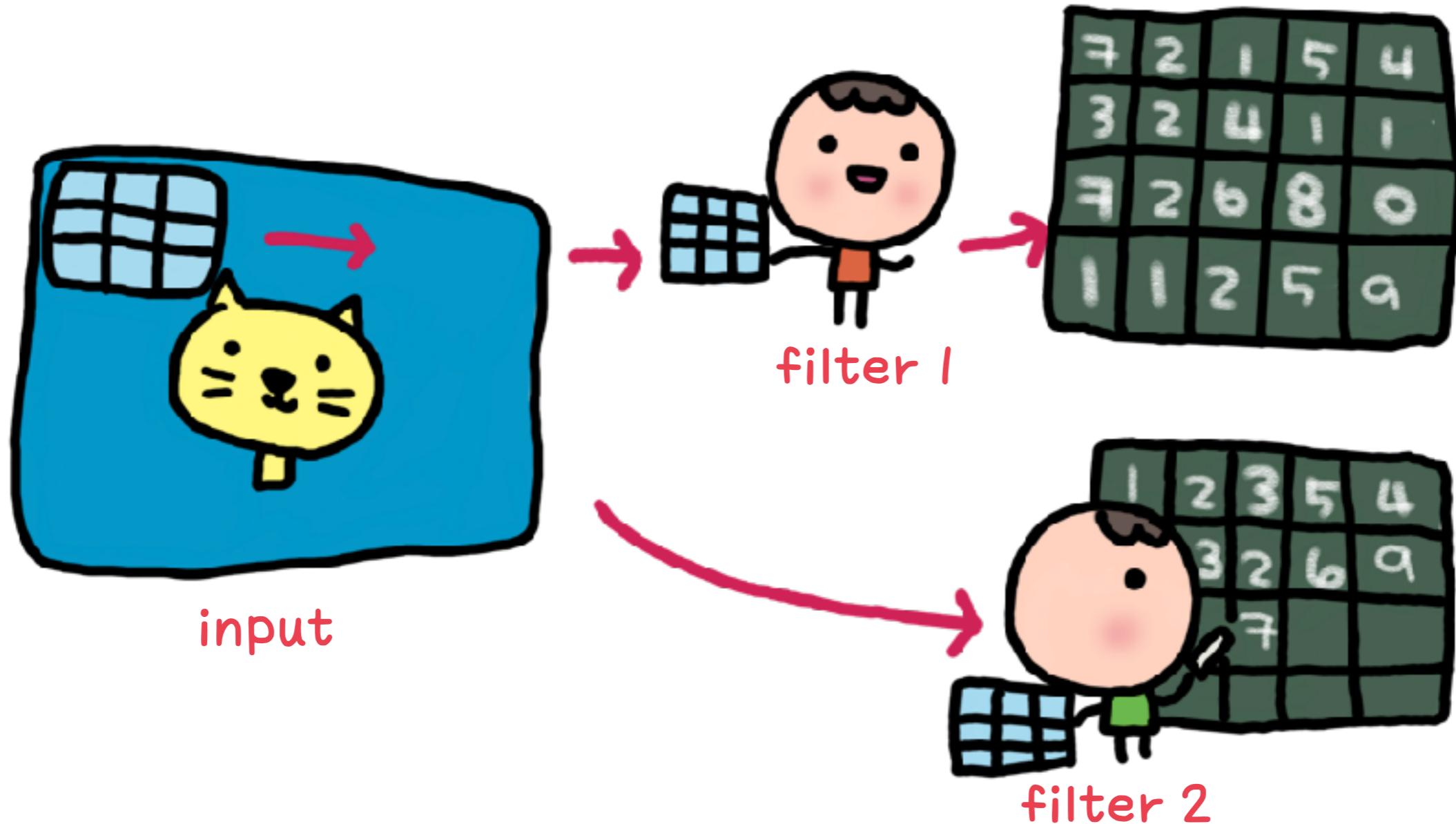
❤ 200



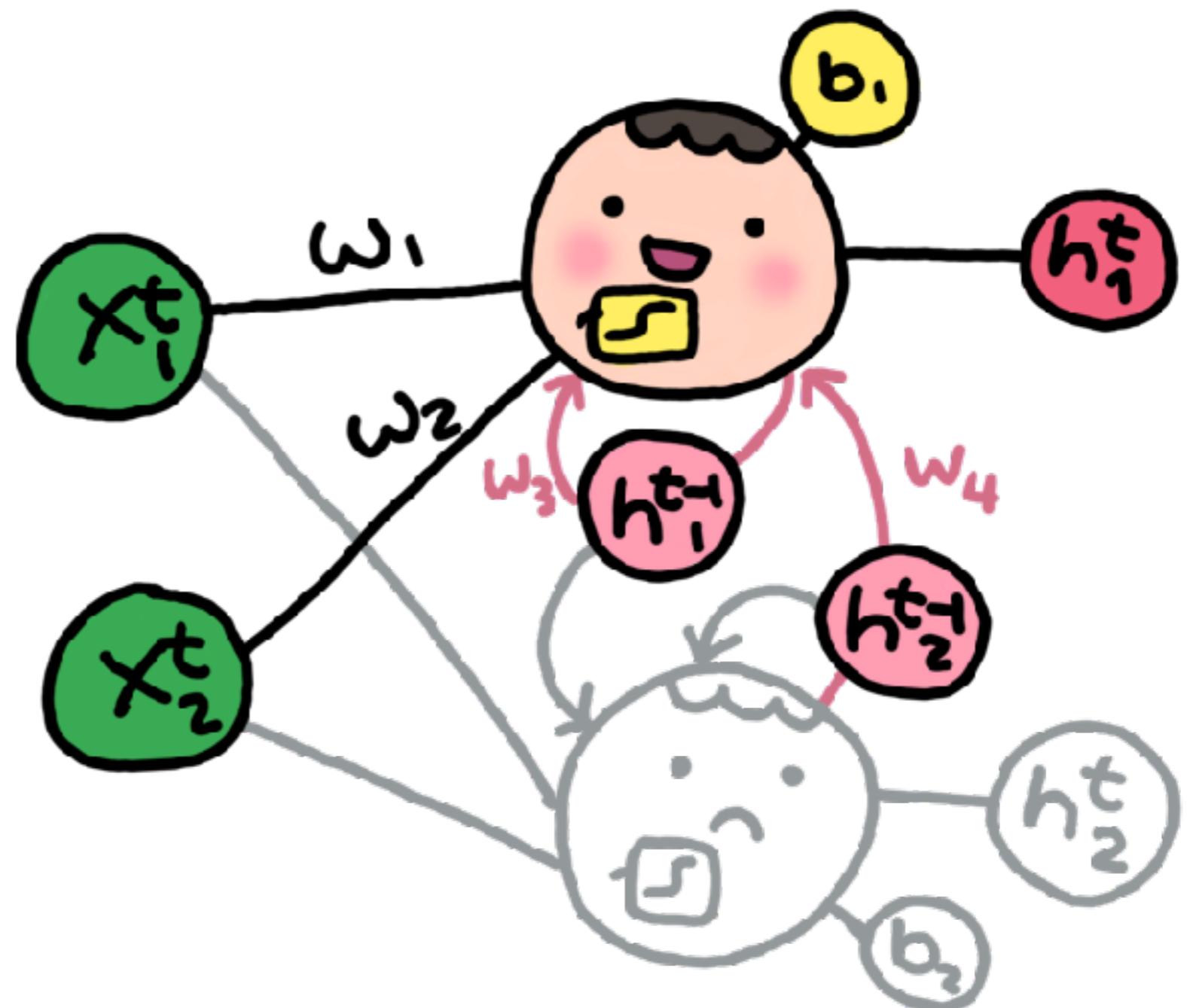
# 標準的神經網路



# Convolutional Neural Network (CNN)



# Recurrent Neural Network (RNN)



$$h_1^t = \sigma(w_1 x_1^t + w_2 x_2^t + w_3 h_3^{t-1} + w_4 h_4^{t-1} + b_1)$$

# 反正就是要學個函數



1

# 我們先問一個問題

在野外看到一隻動物，我想知道是什麼？



2

# 化成函數的形式

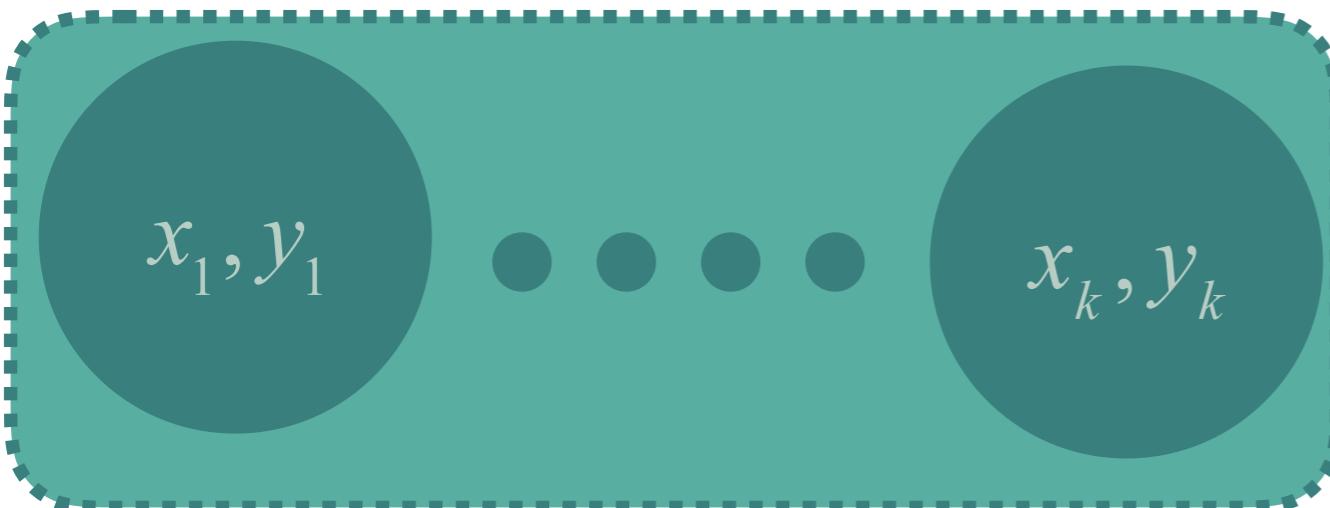


# 3

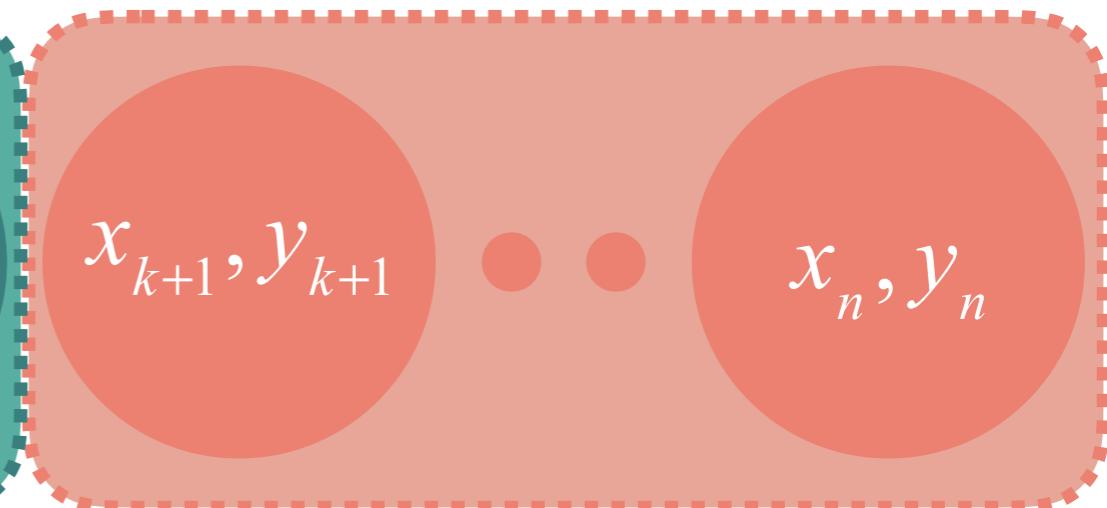
## 準備訓練資料

(, "台灣黑熊"), (, "鱗蛇"), ...

訓練資料



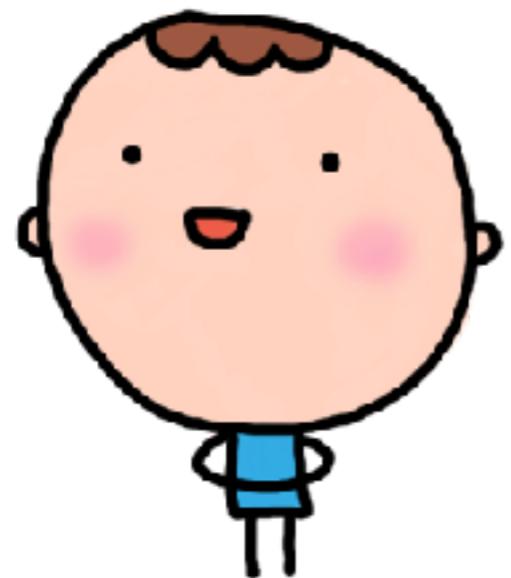
測試資料



4

# 架構我們的神經網路

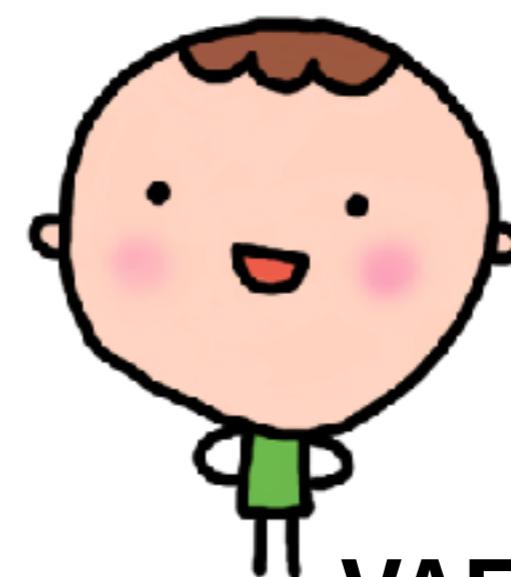
標準 NN



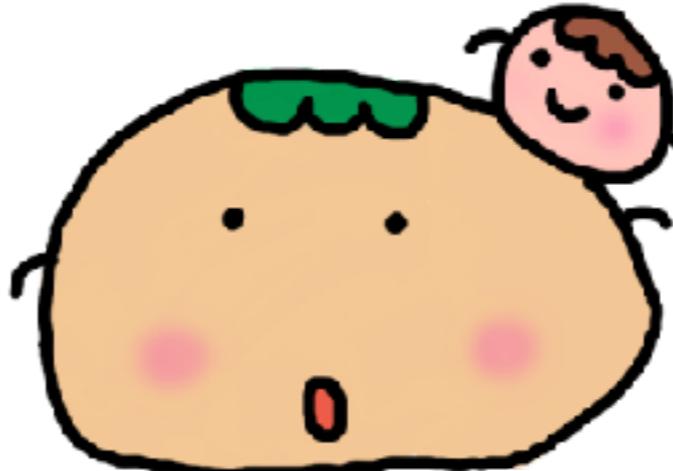
CNN



RNN



VAE



強化學習

生成對抗模式 (GAN)

膠囊

我們架好了神經網路，就剩一些參數要調。

$$\theta = \{w_i, b_j\}$$

決定好這些參數的值，就會出現一個函數。

$$f_\theta$$

# 5

## 學習

學習都是用我們訓練資料送進我們的神經網路，調整我們的參數，然後用個 **loss function** 看我們和觀察值差多少。

$$L(\theta)$$

基本上就是用

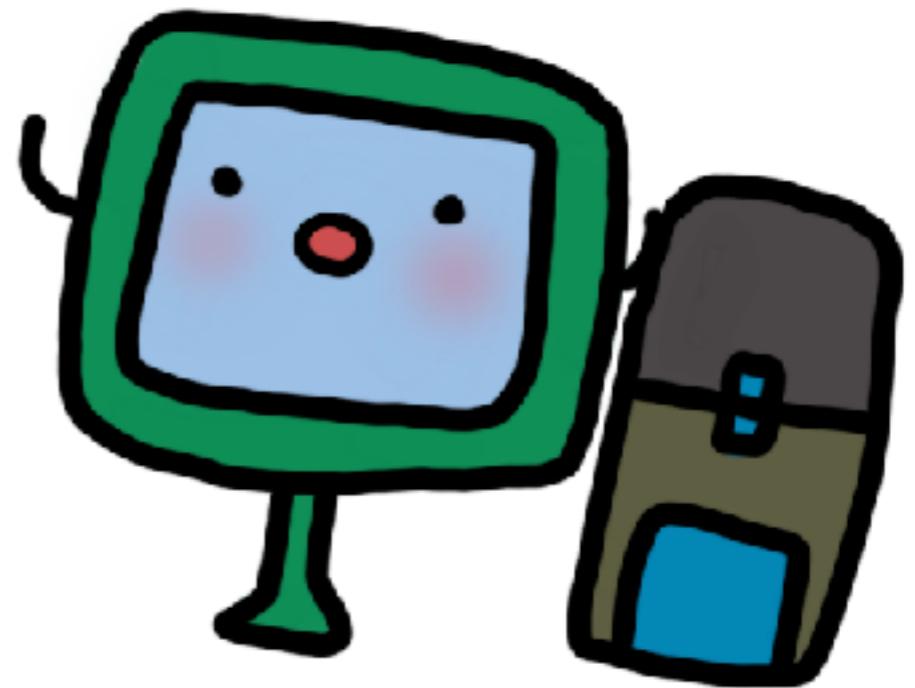
gradient descent

因神經網路的特性，叫

backpropagation

2

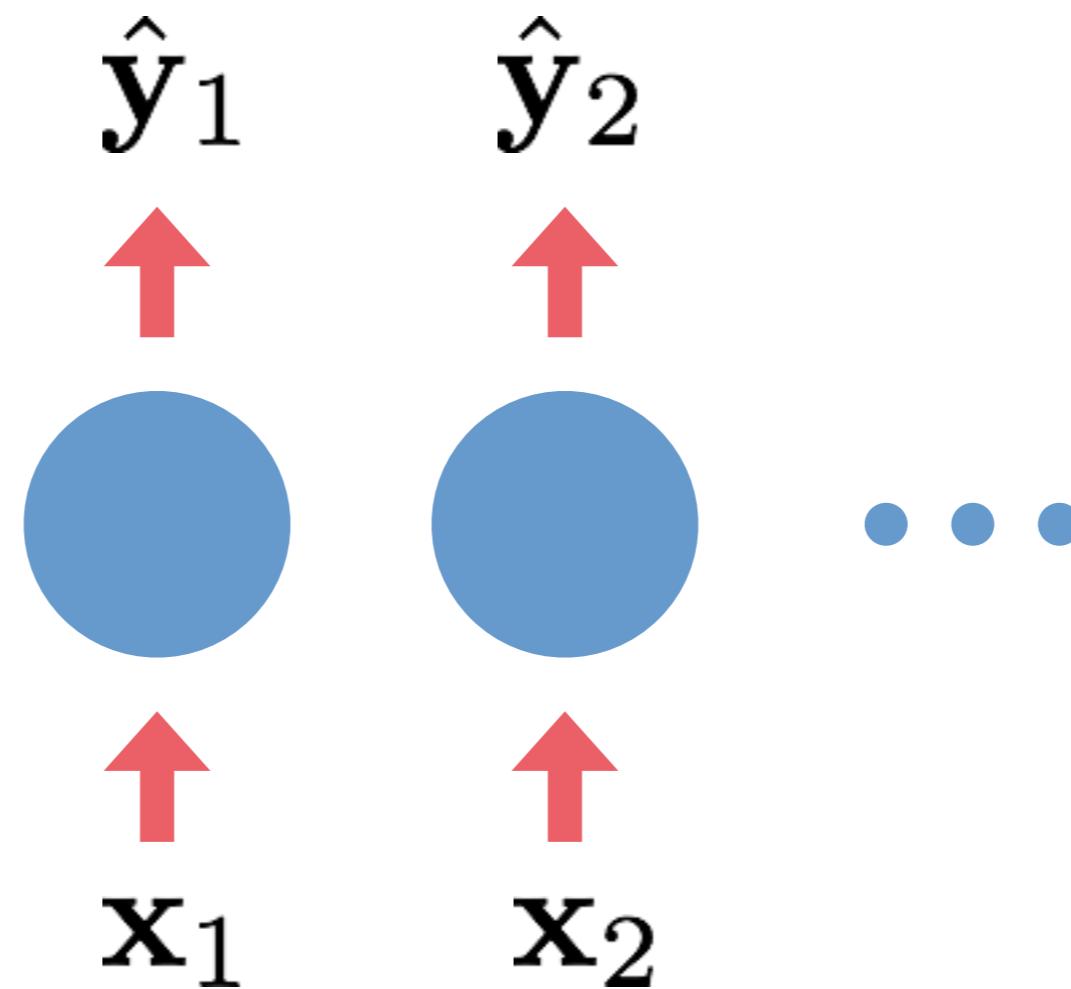
# RNN 簡介



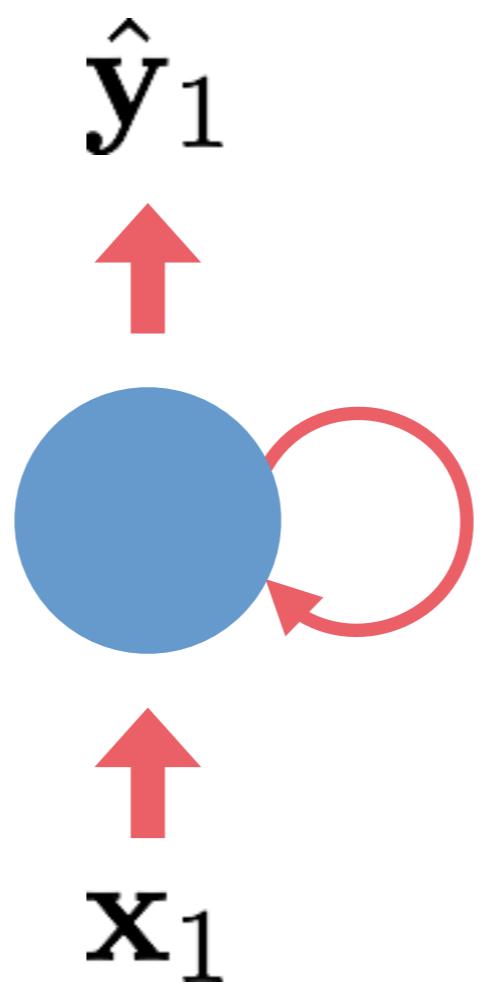
# Recurrent Neural Network

RNN 是**有記憶**的神經網路

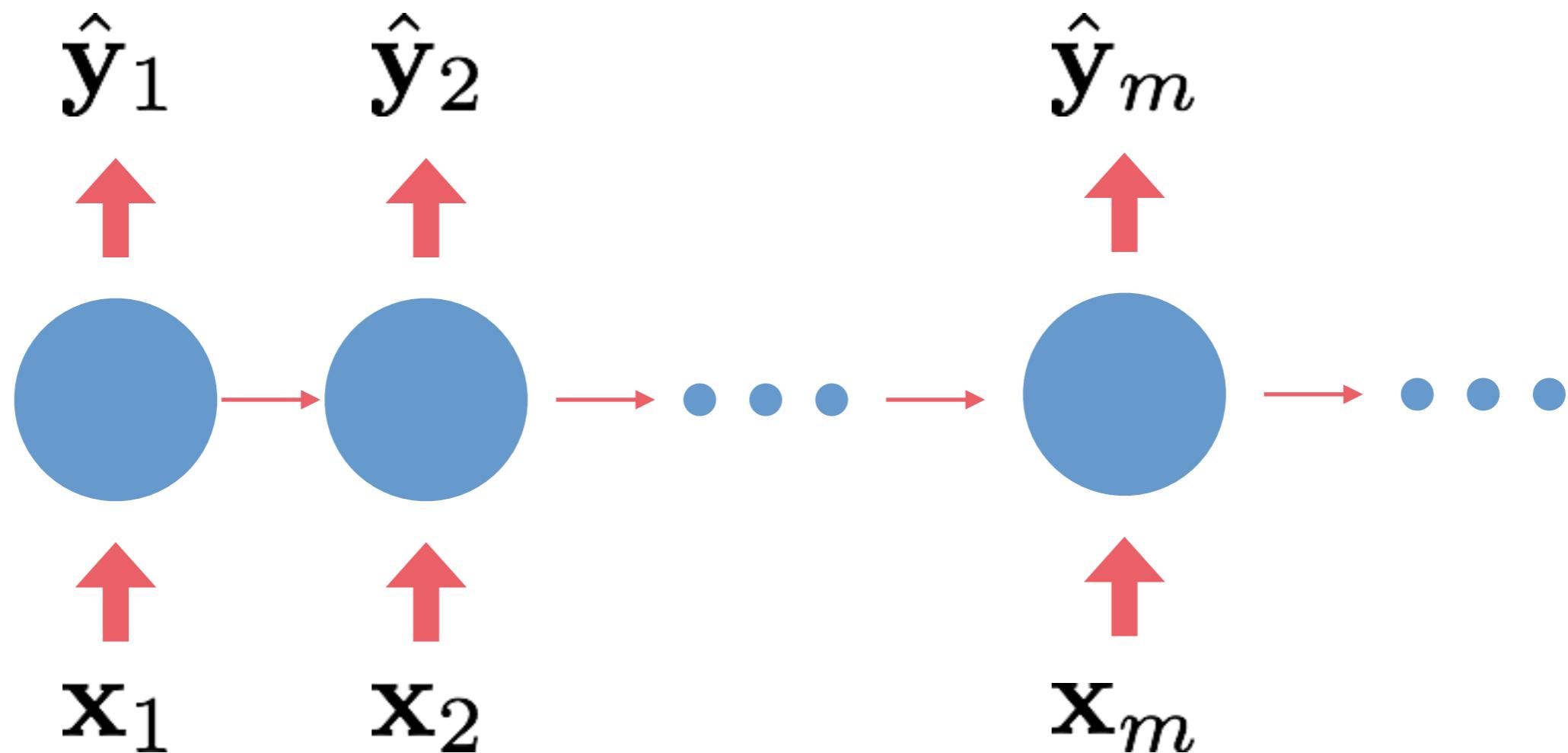
一般的神經網路一筆輸入和下一筆是  
沒有關係的...



RNN 會偷偷把上一次的輸出也當這一次的輸入。

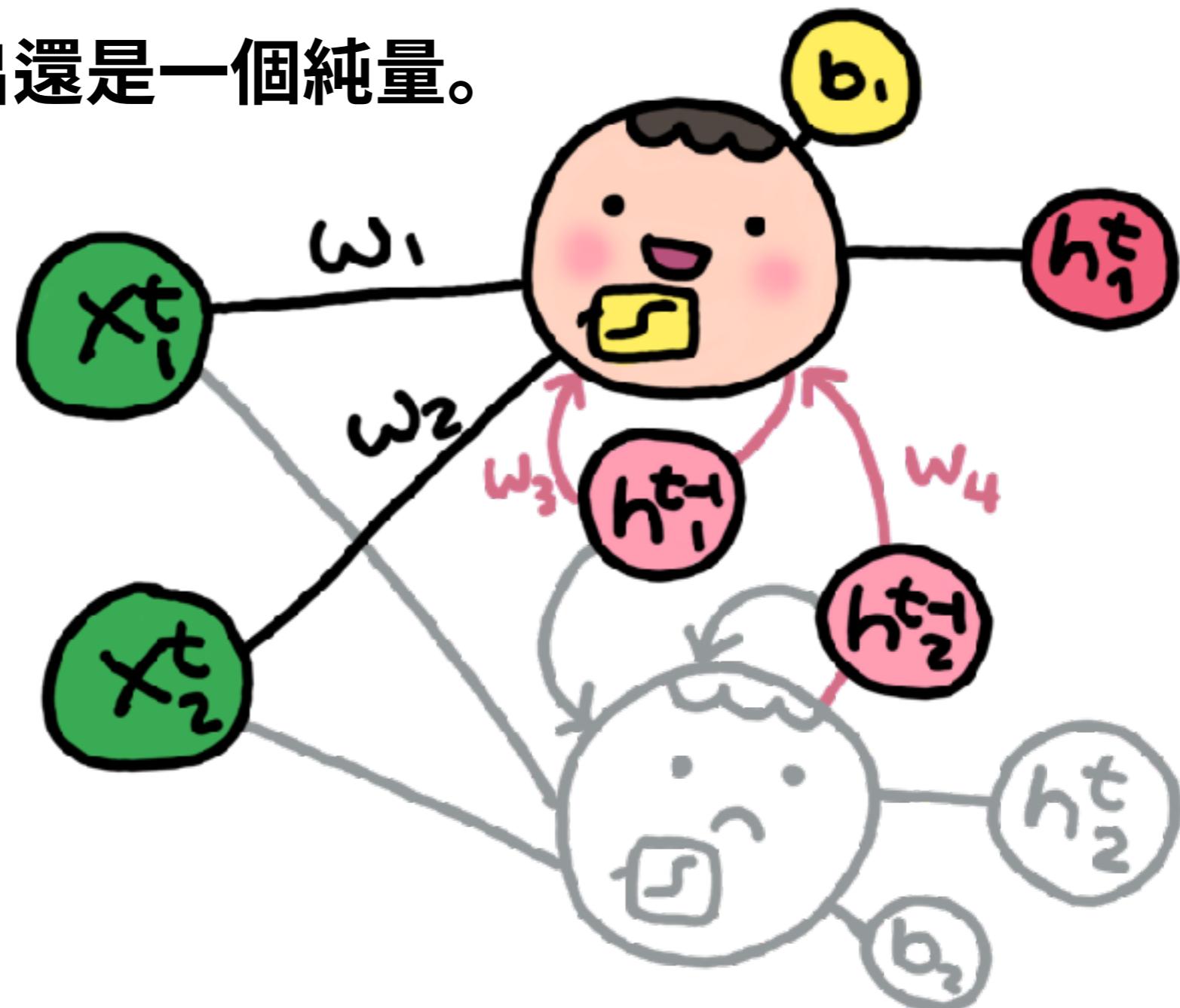


很多人畫成這樣。



# 實際上某一個 RNN Cell 的作用是這樣

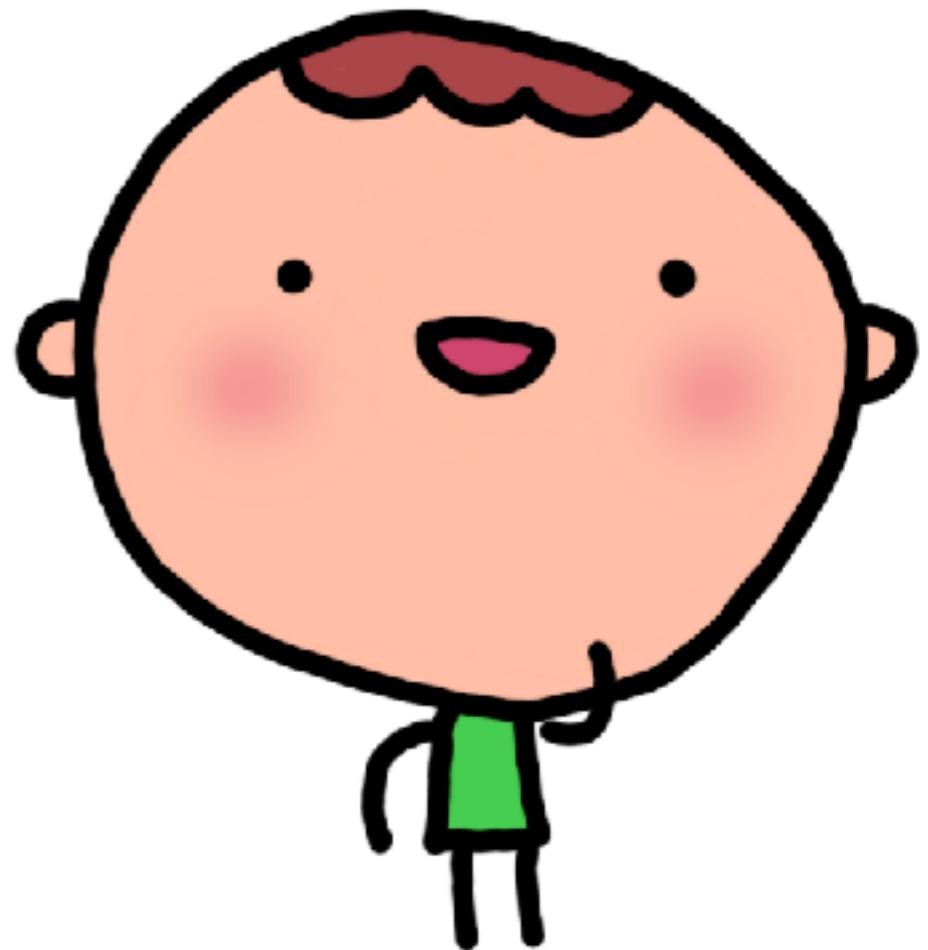
注意每個 cell 輸出還是一個純量。

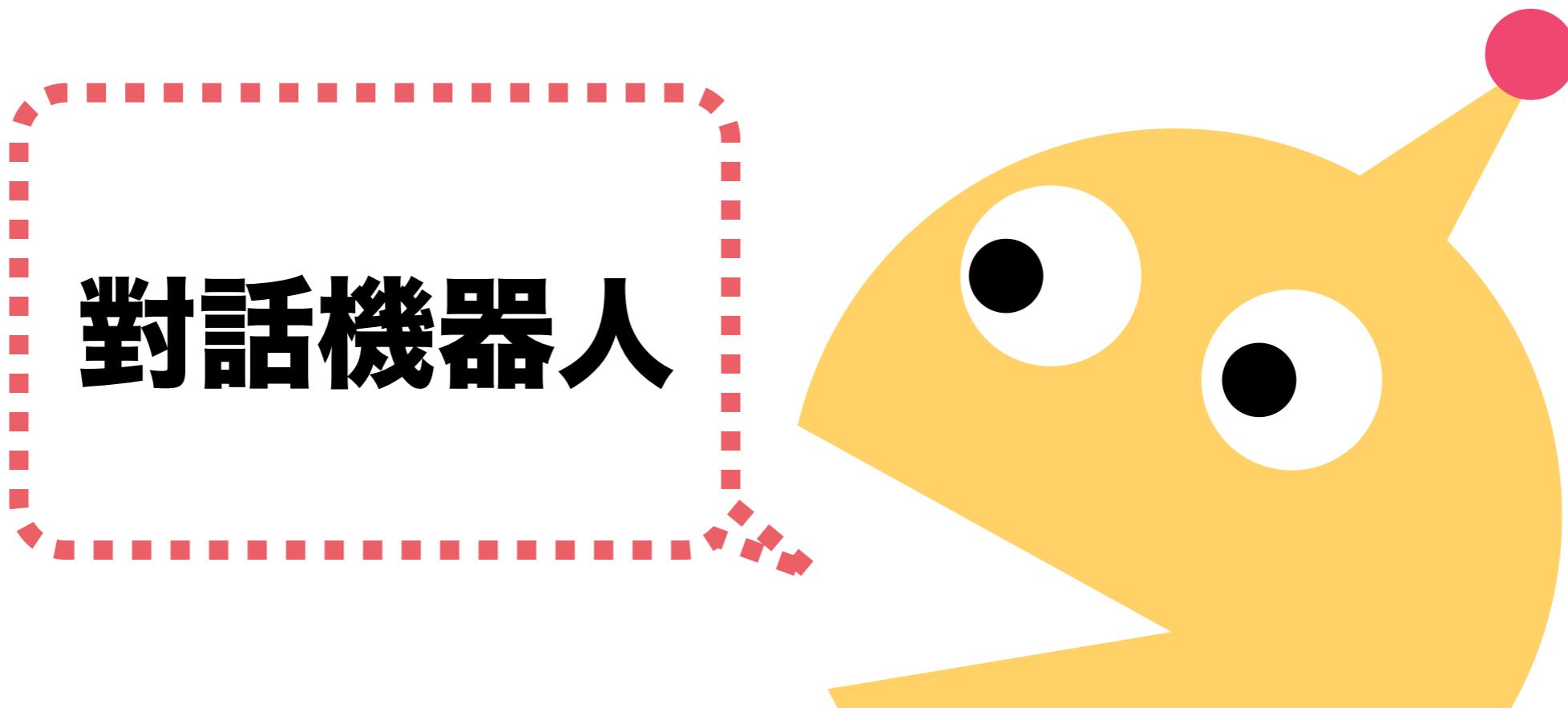


$$h_1^t = \sigma(w_1 x_1^t + w_2 x_2^t + w_3 h_3^{t-1} + w_4 h_4^{t-1} + b_1)$$

2

# RNN 的應用

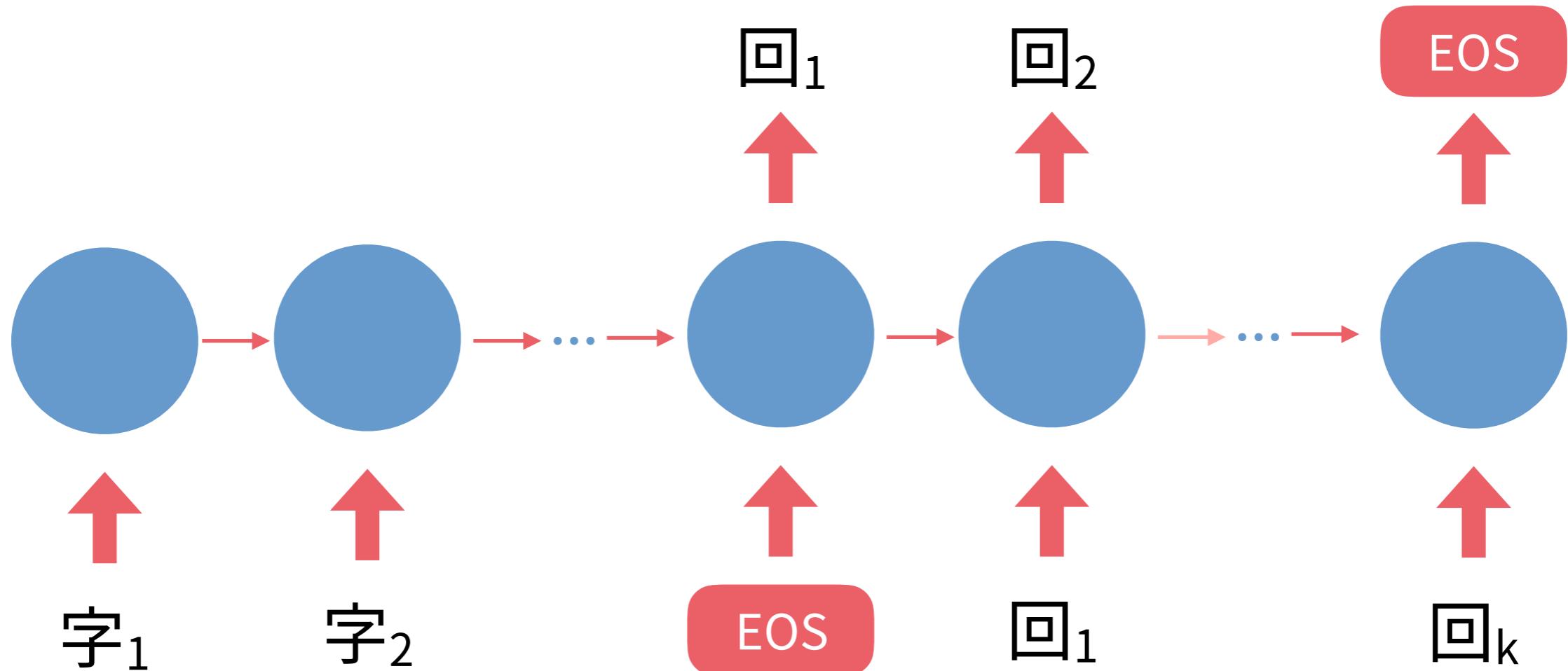




$f(\text{目前的字}) = \text{下一個字}$

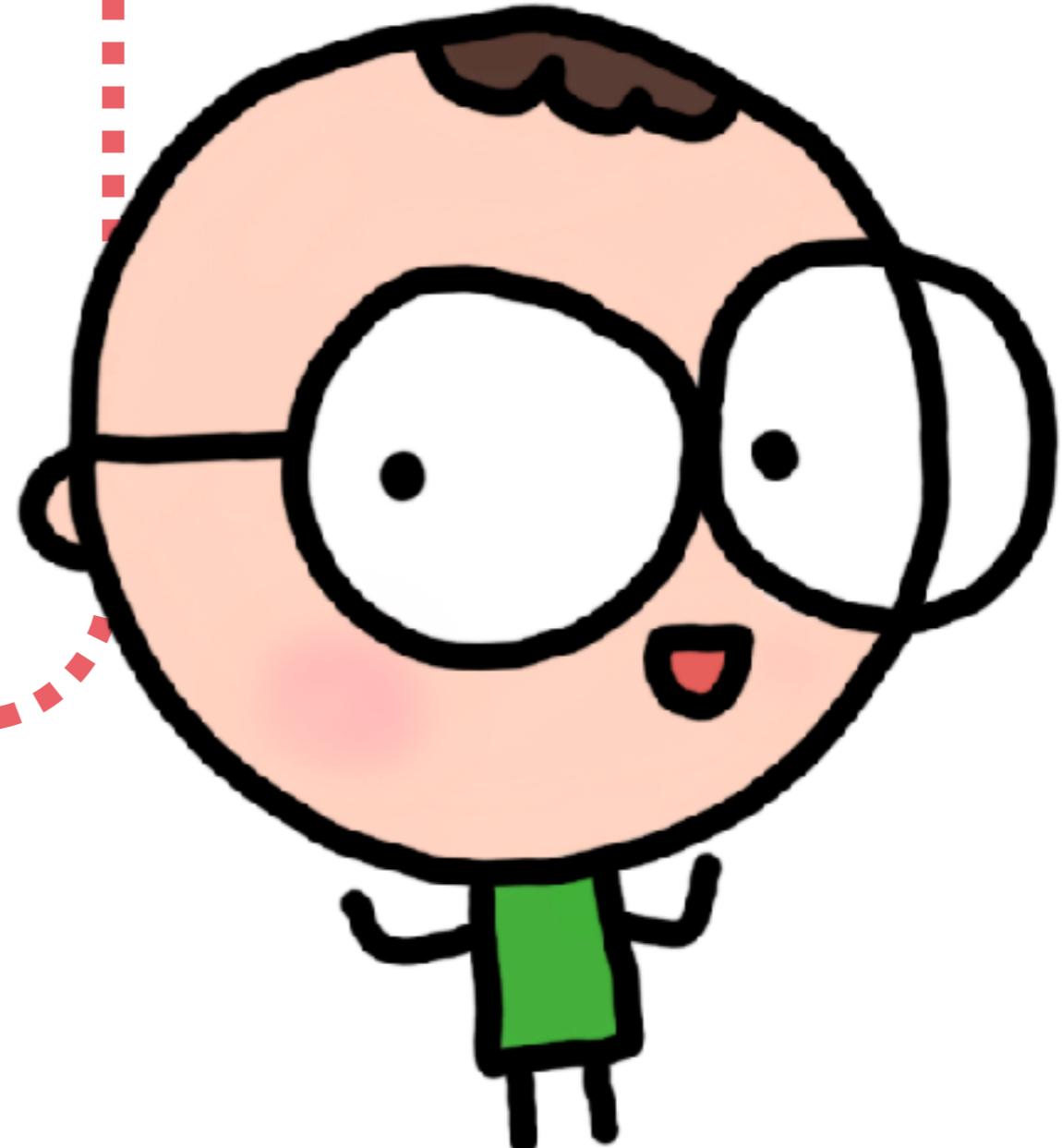
應用

# 對話機器人



注意這樣的模式，每次輸入和輸出都不是固定的長度！

其實輸入不一定要文字，是影  
片（一張一張的圖）也是可以  
的！輸出還是可以為文字，最常  
見的大概是讓電腦說影片中發  
生什麼事。



# 同樣型式的應用

- 翻譯。
- Video Captioning 生成影片敘述。
- 生成一段文字。
- 畫一半的圖完成它。



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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

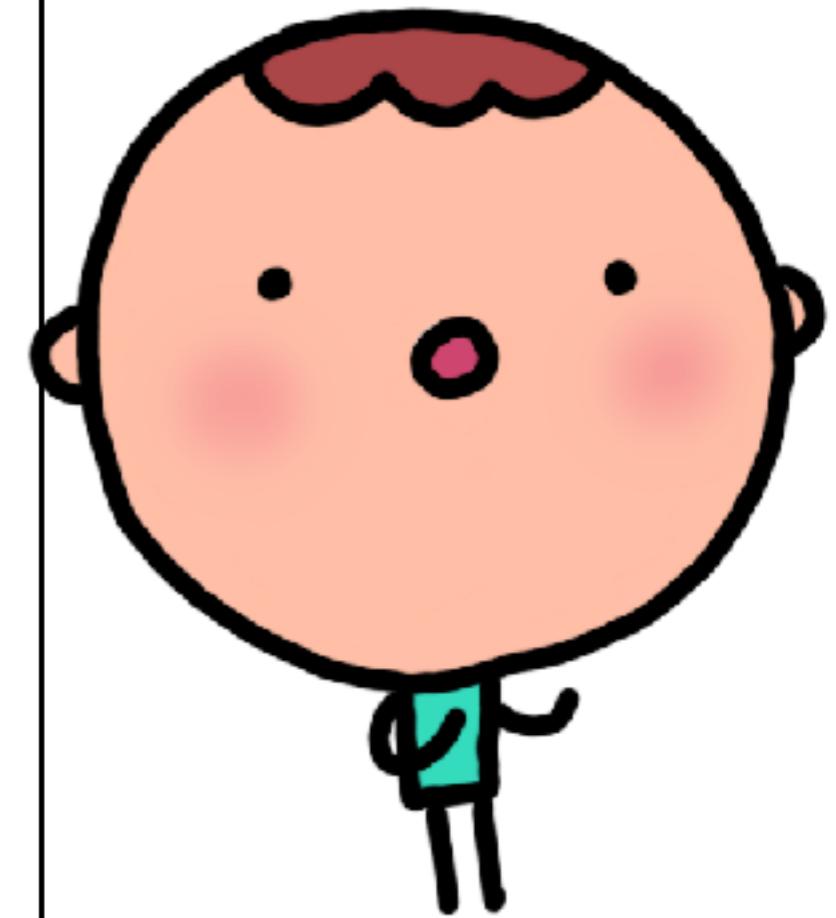
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

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。



電腦仿的莎士比  
亞作品。

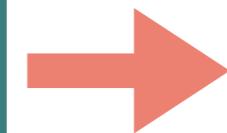
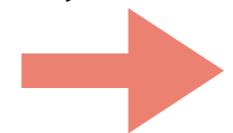
# **MLB 球員全壘打數預測**

**孫瑄正、邢恒毅、劉亮緯、曾煜祐、李錦藤**

**第  $t-1$  年**

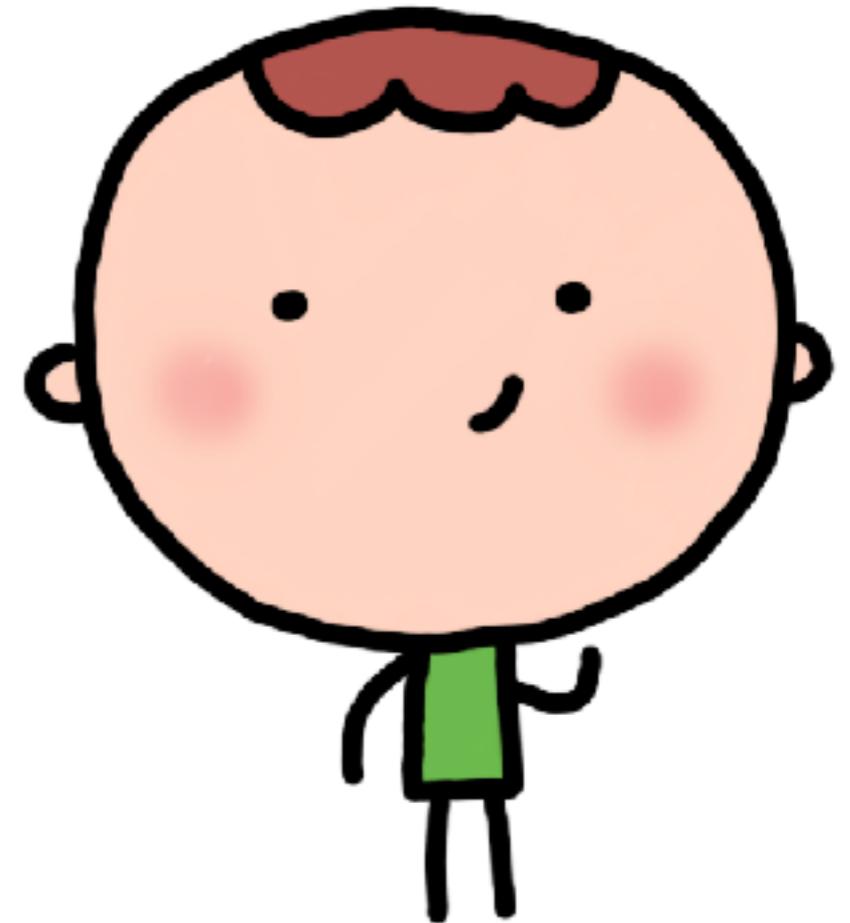
[Age, G, PA, AB, R, H, 2B, 3B, HR,  
RBI, SB, BB, SO, OPS+, TB]

**15 個 features**

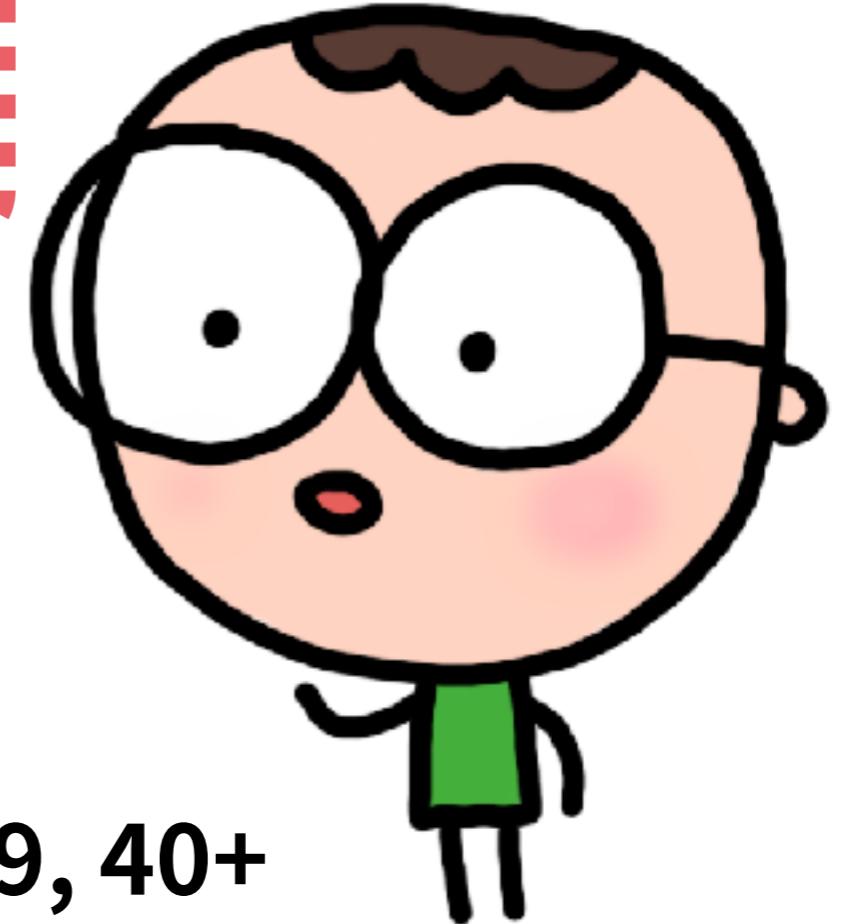


**第  $t$  年  
全壘打數**

- 運用 LSTM, 輸入 10 年的資料猜下一年
- 只有一層 LSTM 層!



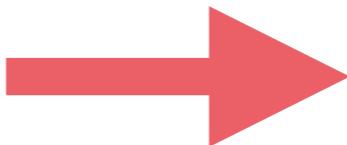
**不要猜精確數目，猜區  
間即可！**



**分五段: 0-9, 10-19, 20-29, 30-39, 40+**

# One-Hot encoding

10-19



0	1	0-9
1	2	10-19
0	3	20-29
0	4	30-39
0	5	40+

# 2017 預測結果

## (2016 年 6 月預測)

**Mike Trout** (LAA)

預測 30-39

實際 33

**Kris Bryant** (CHC)

預測 30-39 (第二高 20-29)

實際 29

**Mookie Betts** (BOS)

預測 20-29

實際 24

**Daniel Murphy** (WSH)

預測 20-29

實際 23

**Jose Altuve** (HOU)

預測 20-29

實際 24

**Corey Seager** (LAD)

預測 20-29

實際 22

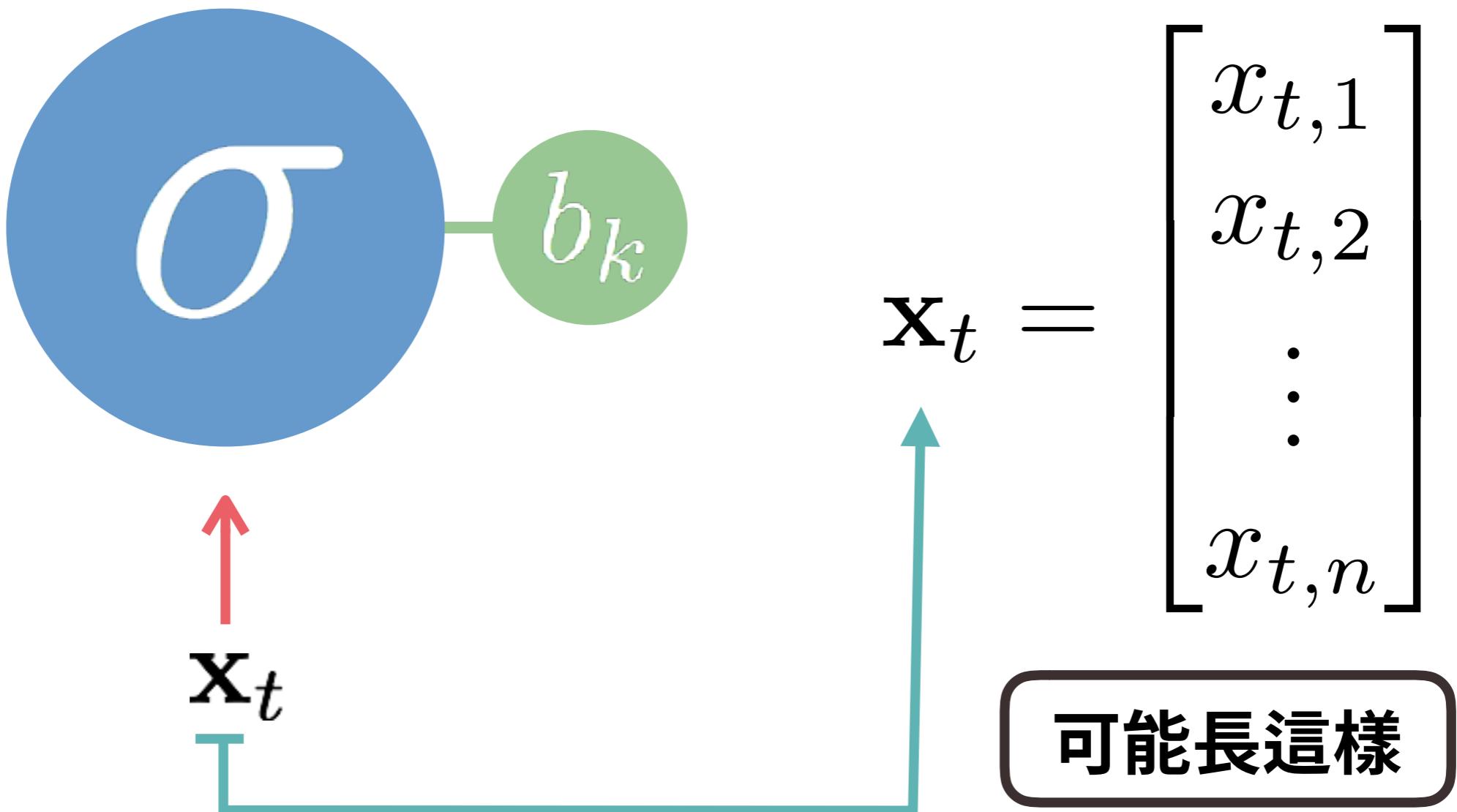
3

基本 RNN  
長這樣



# 注意

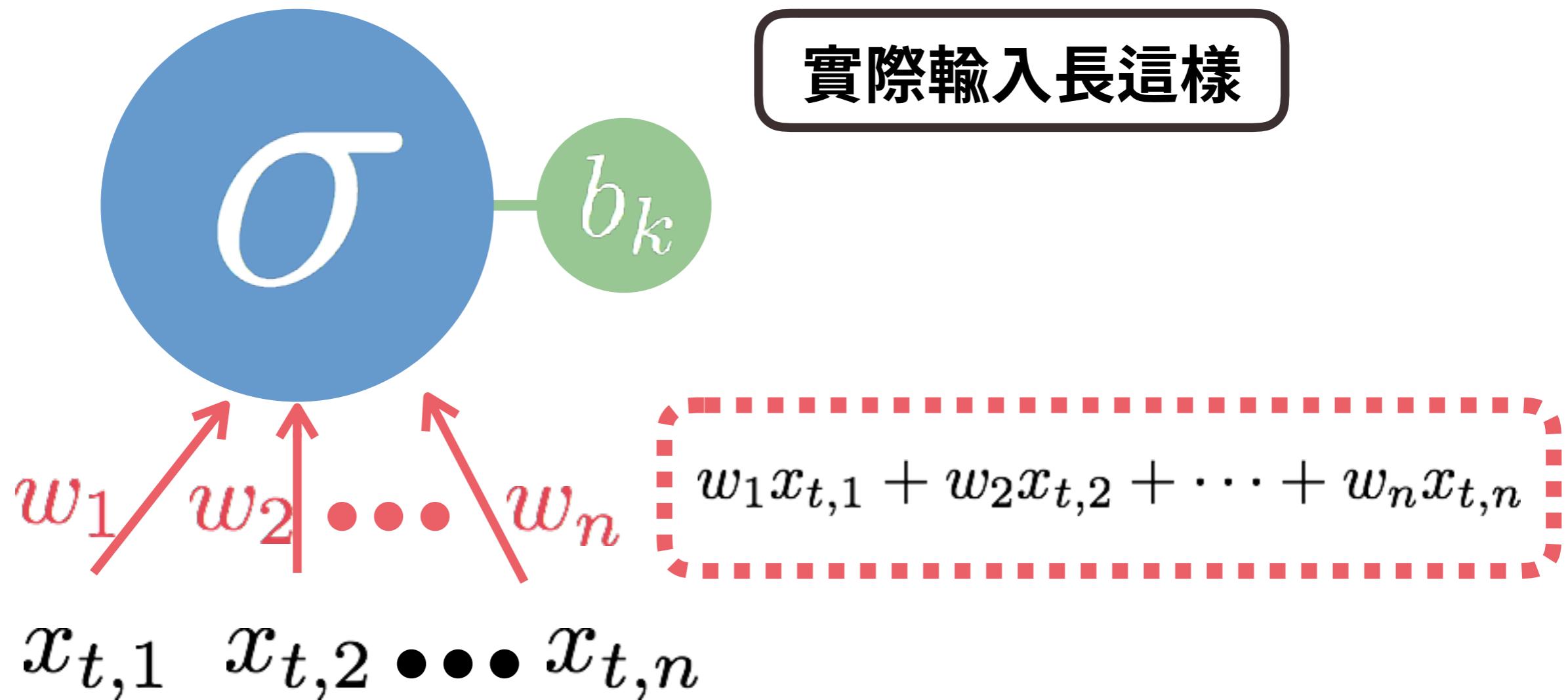
為了讓大家更容易瞭解，我們會用較簡單的圖示。請注意輸入都是個向量、**會有權重**；輸出都是純量。



可能長這樣

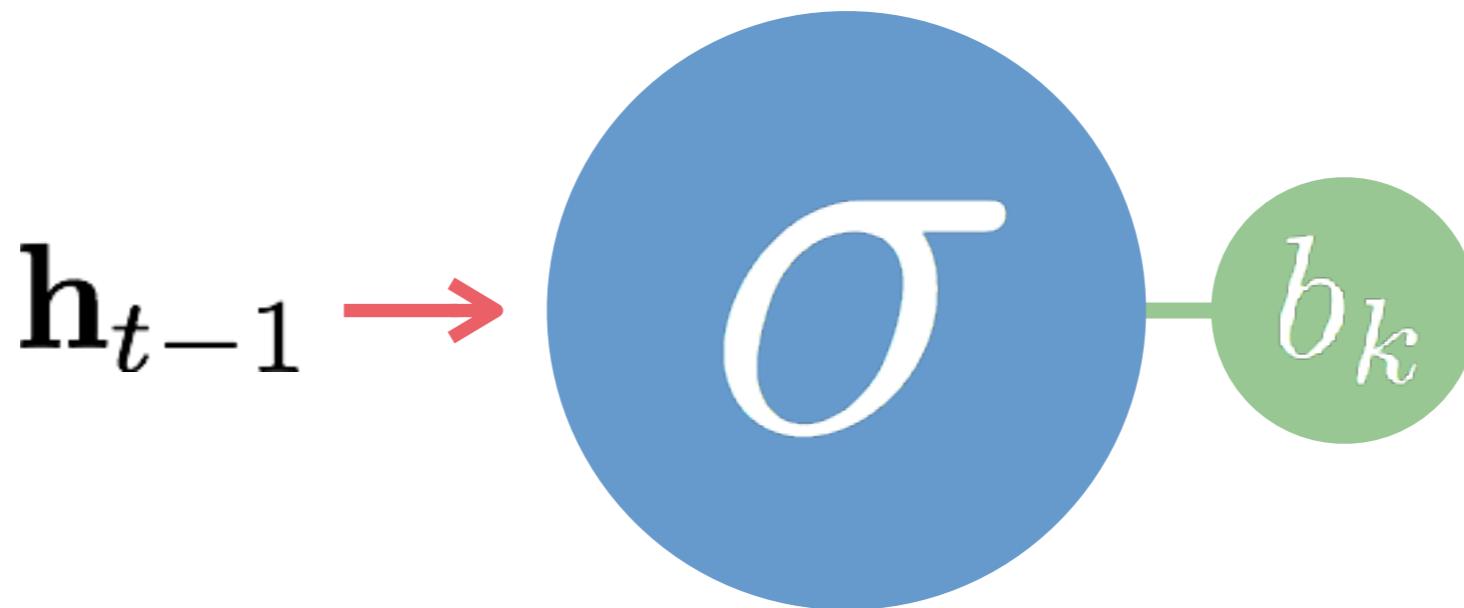
# 注意

為了讓大家更容易瞭解，我們會用較簡單的圖示。請注意輸入都是個向量、  
會有權重；輸出都是純量。

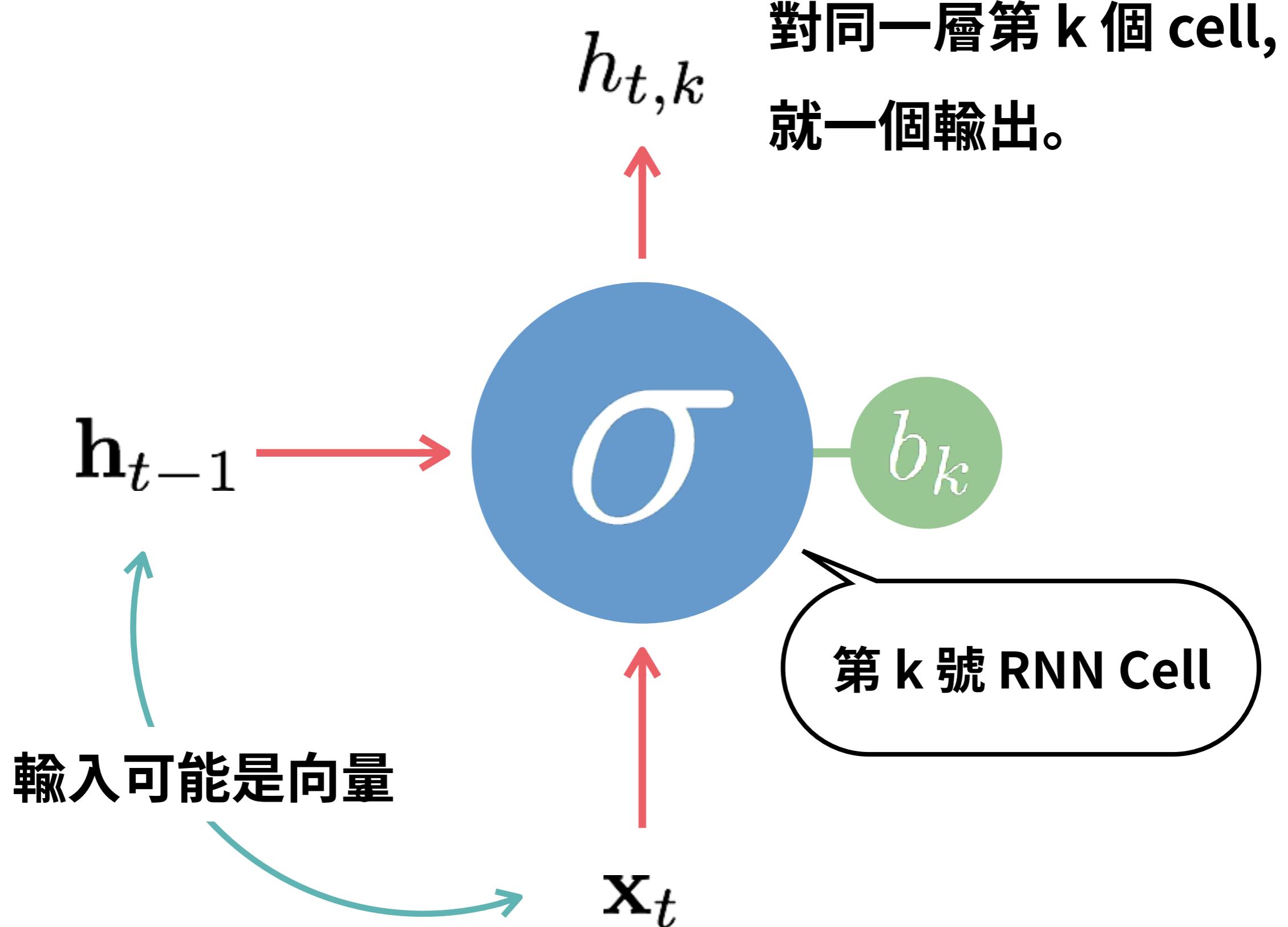


# 注意

為了讓大家更容易瞭解，我們會用較簡單的圖示。請注意輸入都是個向量、**會有權重**；輸出都是純量。

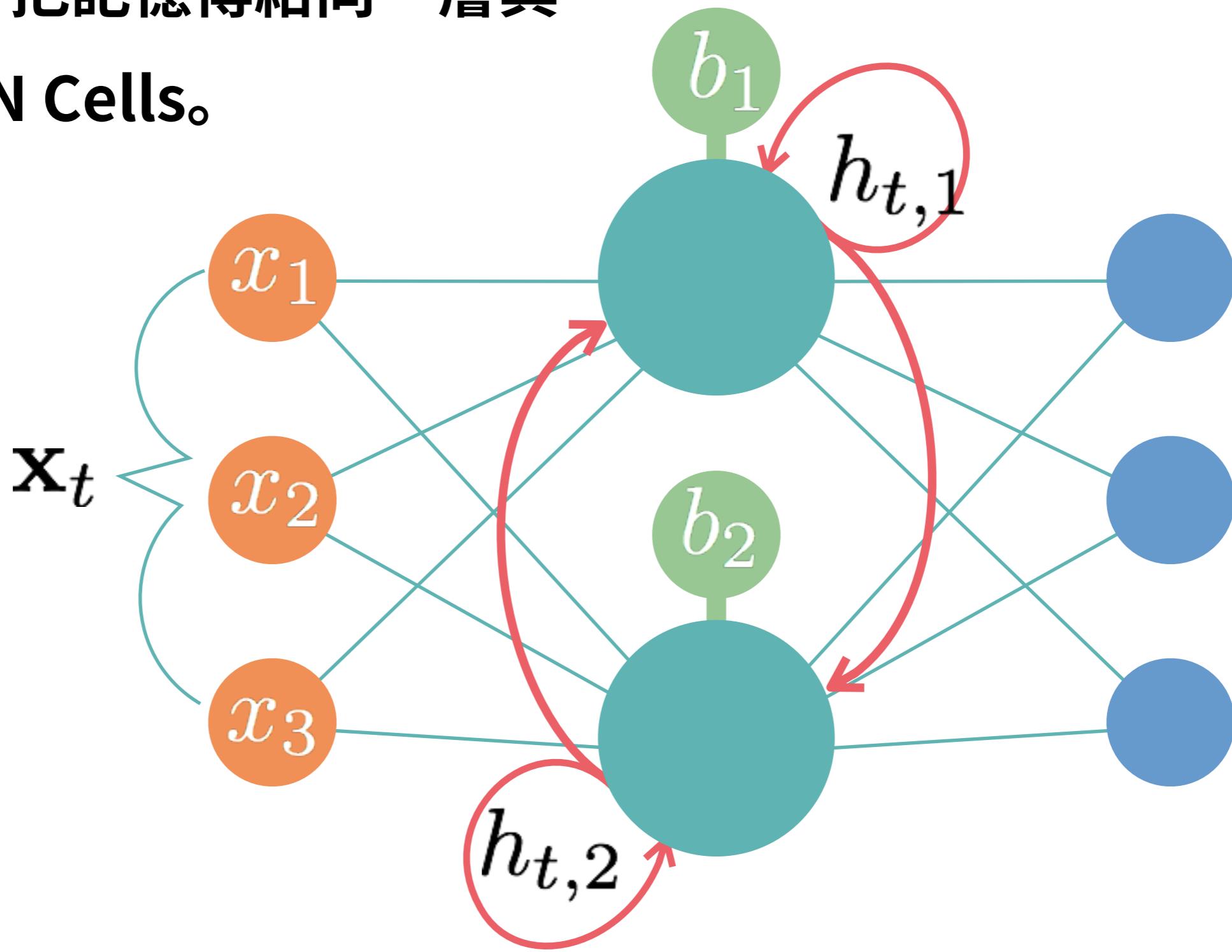


同理  $h$  也是這樣

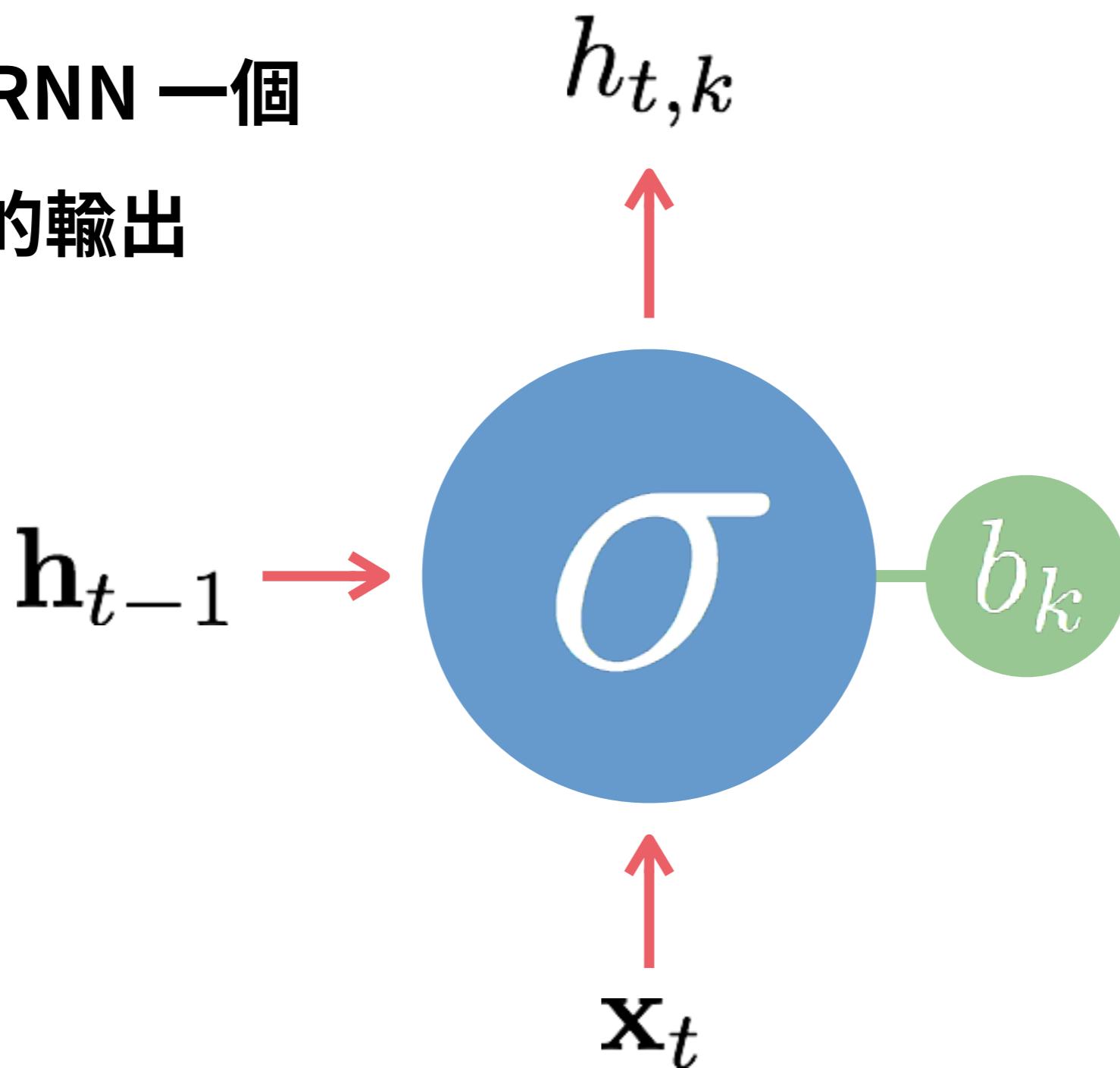


真正連結的樣子，注意 RNN

Cell 會把記憶傳給同一層其他 RNN Cells。



標準 RNN 一個  
Cell 的輸出



$$h_{t,k} = \sigma(\mathbf{w}_h^T \mathbf{h} + \mathbf{w}_x^T \mathbf{x} + b_k)$$

4

比玩具還玩  
具的例子



## 問題

# RNN 預測下一個字

我們來做標準 RNN 應用，預測一句話下一個字。

字 →

用到 RNN 的  
Neural  
Networks

→ 字  
(下一個)

# 字集

我們考慮的字集只有三個字。

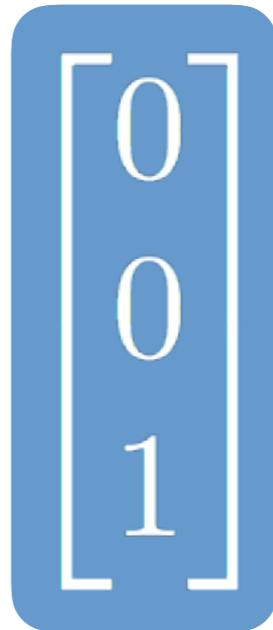
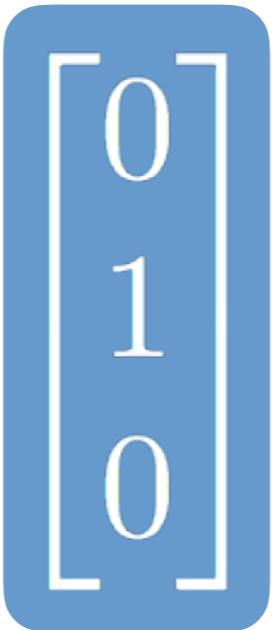
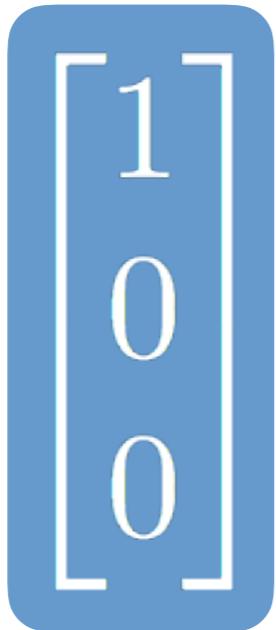
“**好**”， “—”， “**點**”

可以產生如“好一點”，“好好”，“一點一點點”，等等的「句子」。

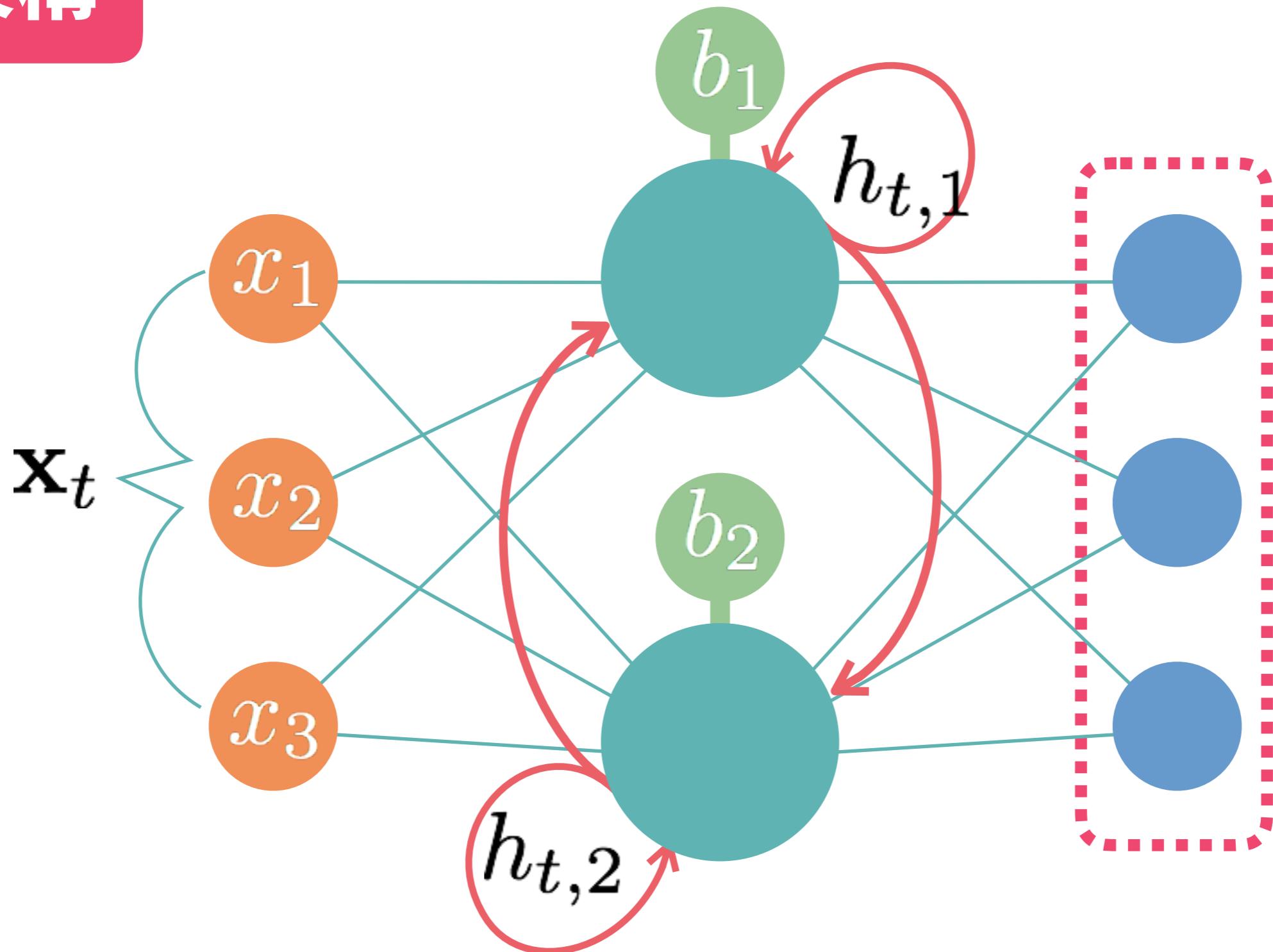
# 字集

用高級的 one hot 編碼。

“**好**” — “**點**”



# 架構



輸入

“好 — 點”

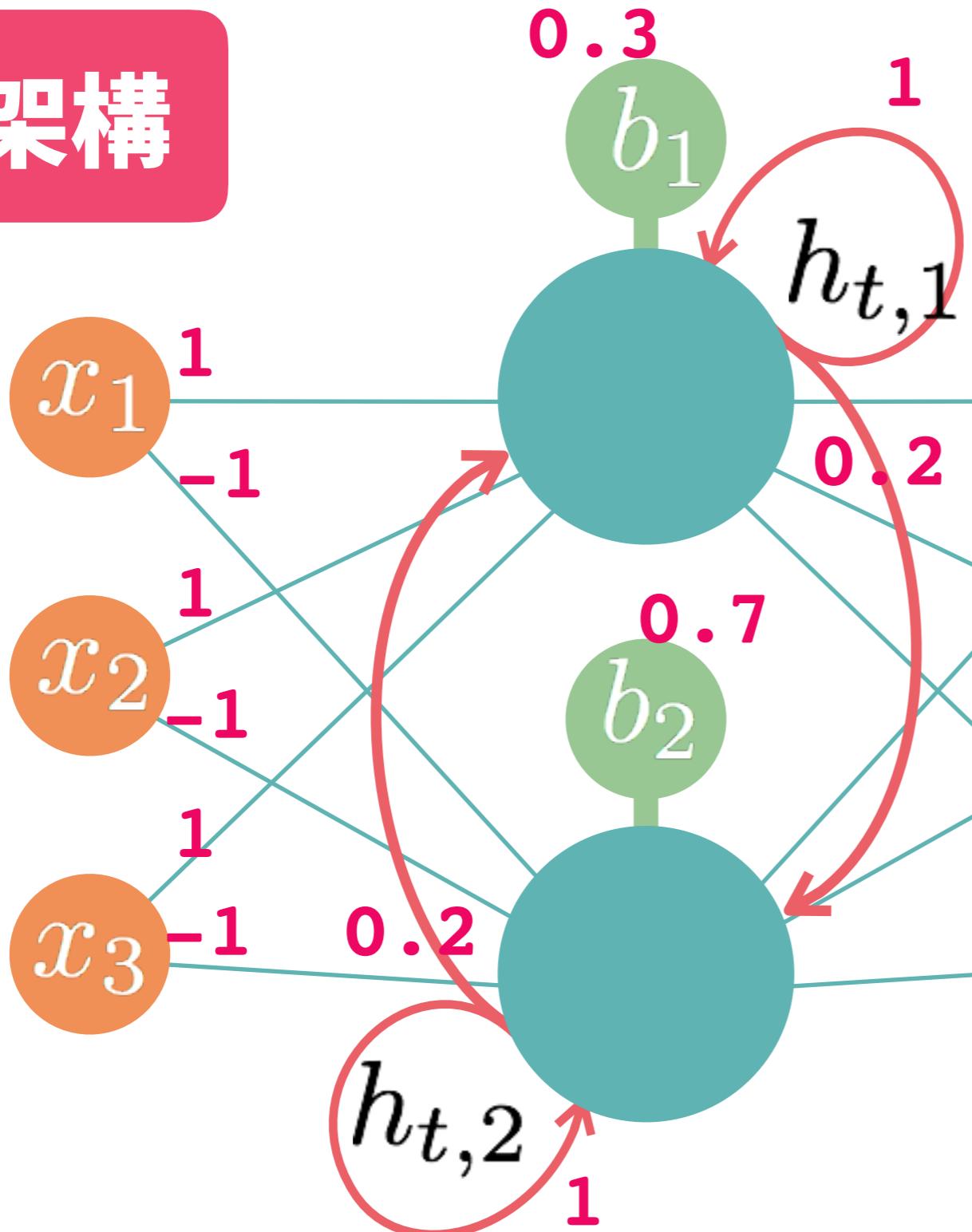
$$\rightarrow \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

$x_1$

$x_2$

$x_3$

# 架構



$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t + \mathbf{b})$$

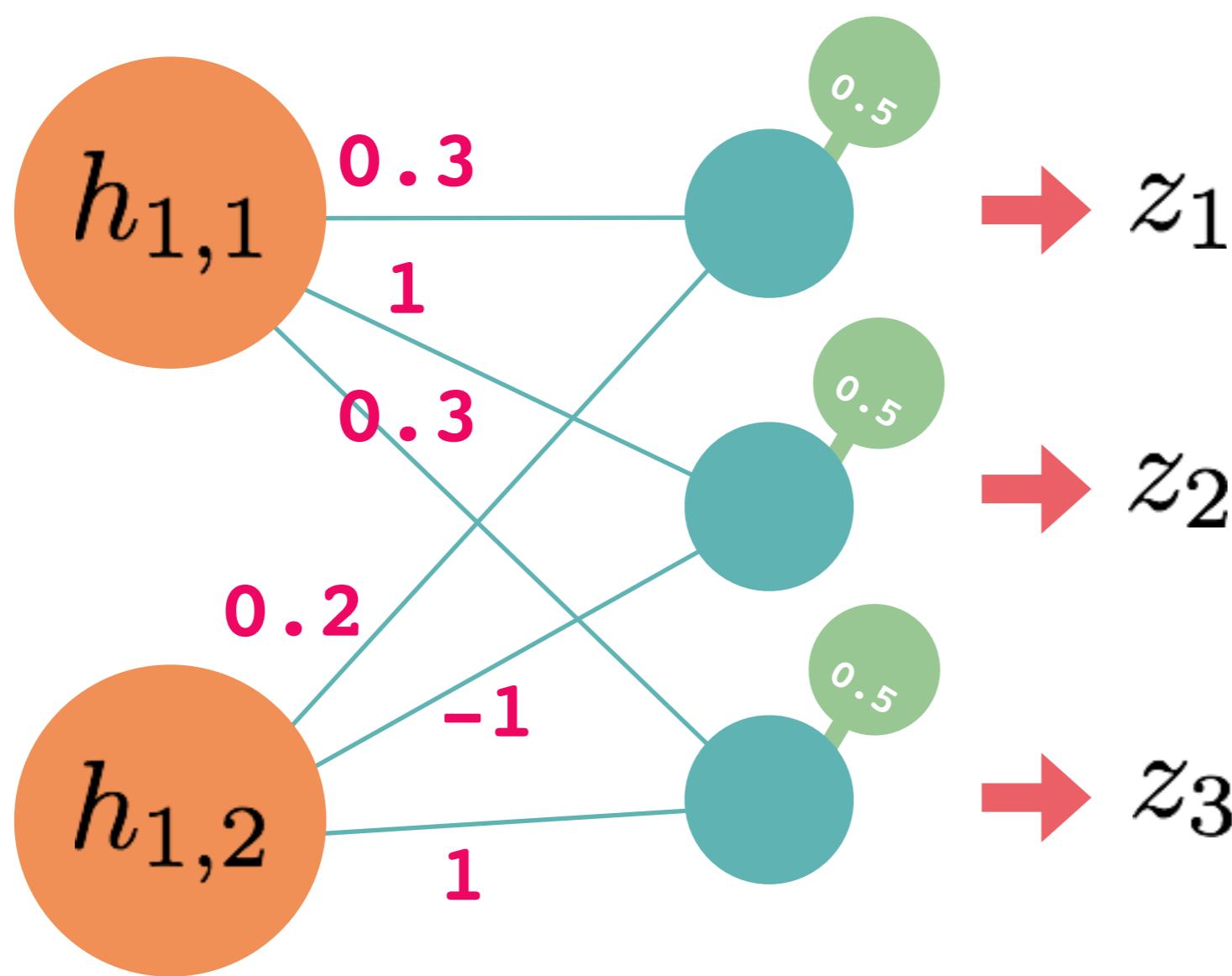
$$W_h = \begin{bmatrix} 1 & 0.2 \\ 1 & 0.2 \end{bmatrix}$$

$$W_x = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$

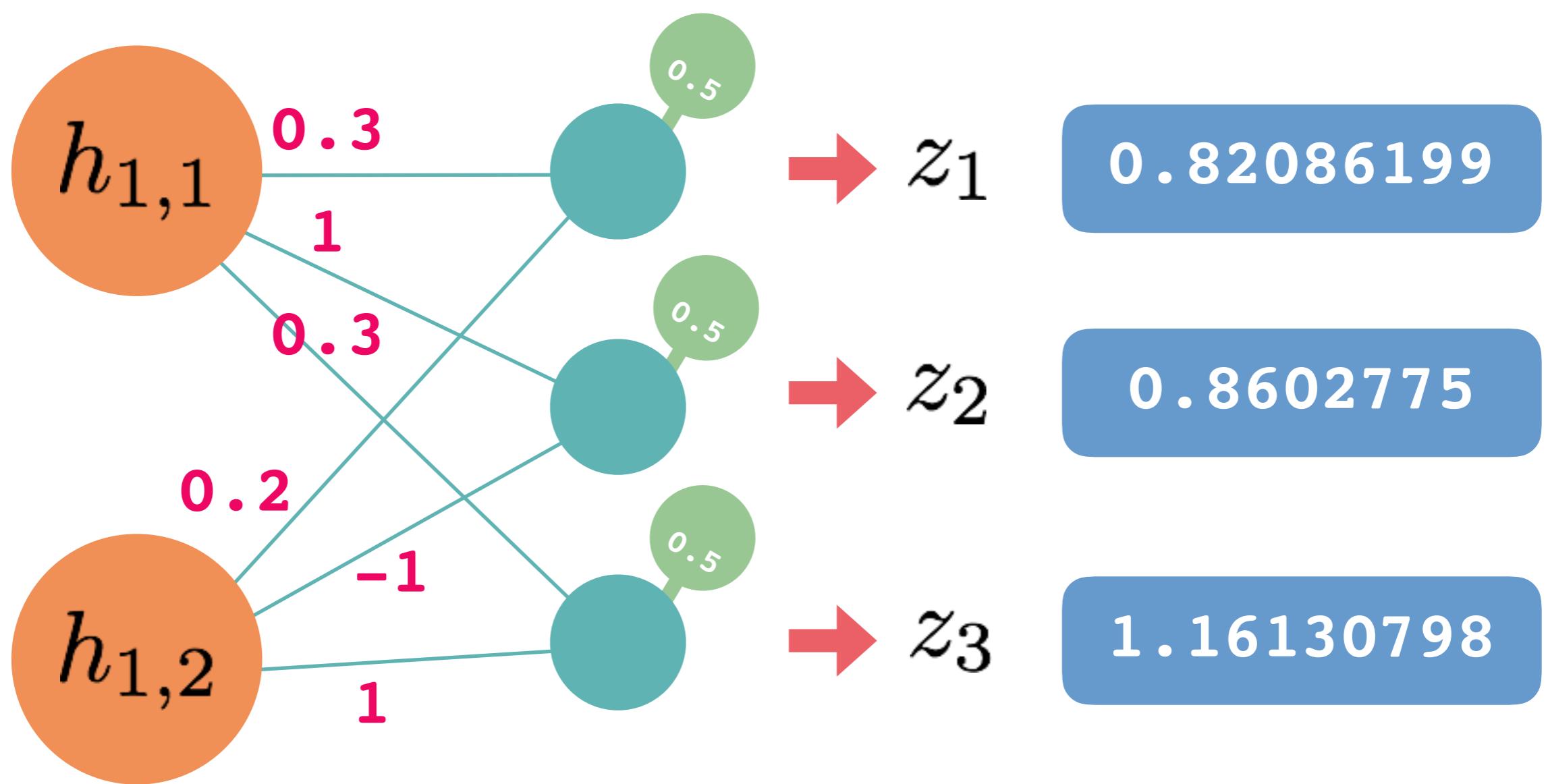
$$\mathbf{b} = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

$$\mathbf{h}_1 = \begin{bmatrix} h_{1,1} \\ h_{1,2} \end{bmatrix} = \begin{bmatrix} 0.78583498 \\ 0.42555748 \end{bmatrix}$$

$$\mathbf{h}_1 = \begin{bmatrix} h_{1,1} \\ h_{1,2} \end{bmatrix} = \begin{bmatrix} 0.78583498 \\ 0.42555748 \end{bmatrix}$$



$$\mathbf{h}_1 = \begin{bmatrix} h_{1,1} \\ h_{1,2} \end{bmatrix} = \begin{bmatrix} 0.78583498 \\ 0.42555748 \end{bmatrix}$$



# 概念

## softmax

$z_1$  0.82086199

$z_2$  0.8602775

$z_3$  1.16130798

希望加起來是 1

# 概念

## softmax

$z_1$

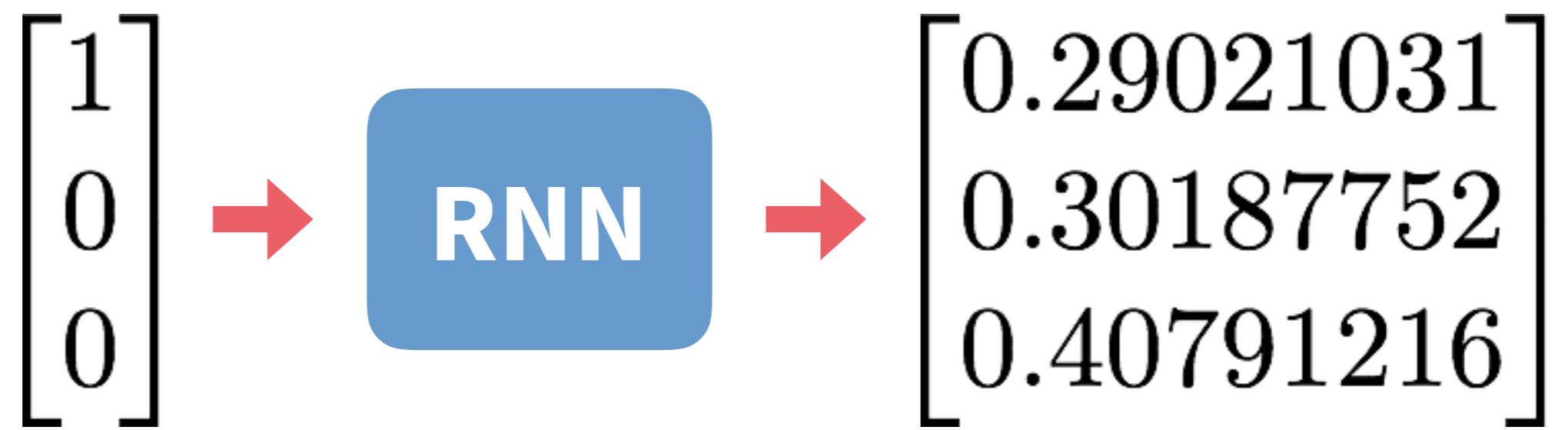
$z_2$

$z_3$

$$\frac{e^{z_1}}{\sum_{i=1}^3 e^{z_i}}$$

$$\frac{e^{z_2}}{\sum_{i=1}^3 e^{z_i}}$$

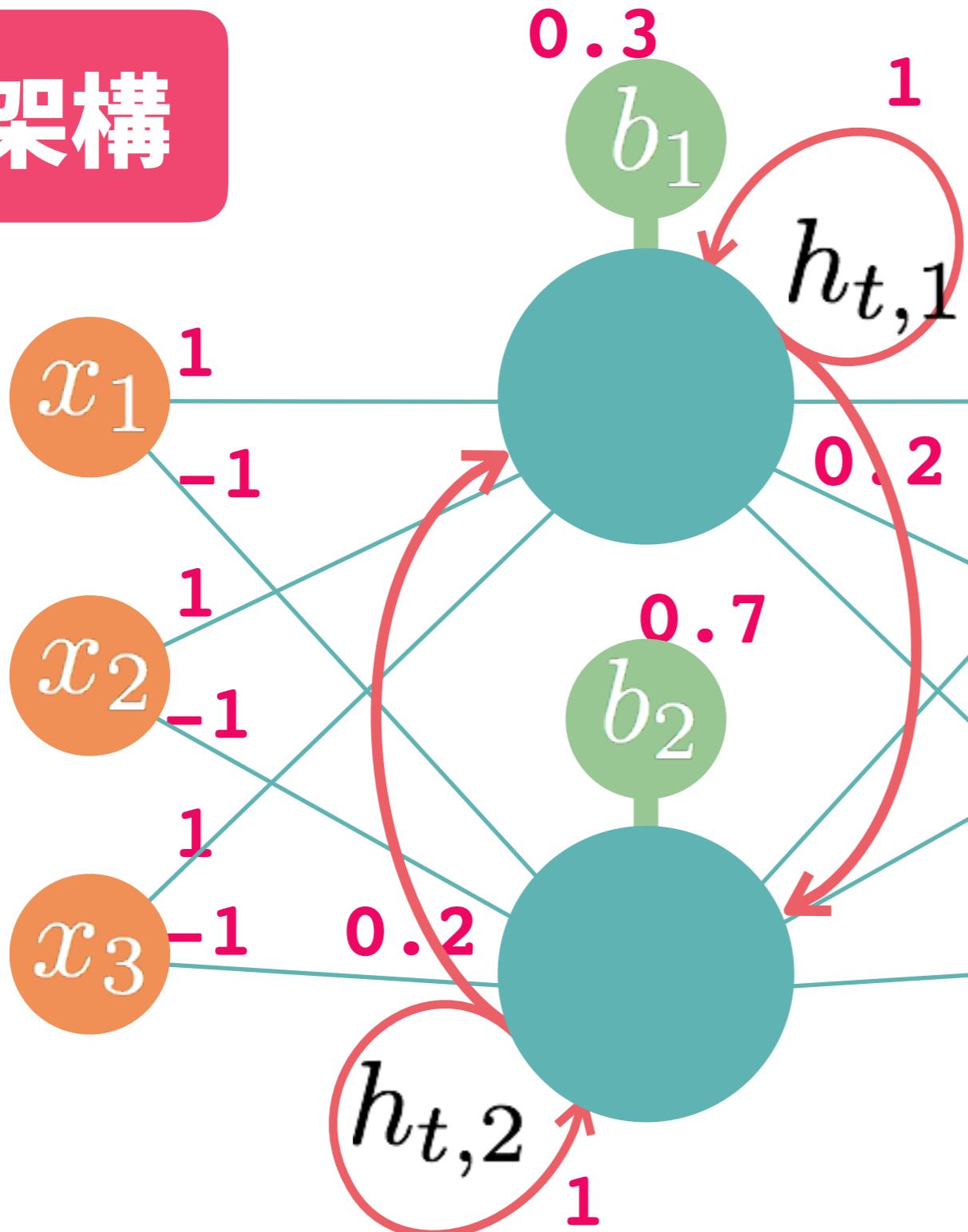
$$\frac{e^{z_3}}{\sum_{i=1}^3 e^{z_i}}$$



再來輸入

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

# 架構



$$W_h = \begin{bmatrix} 1 & 0.2 \\ 1 & 0.2 \end{bmatrix}$$

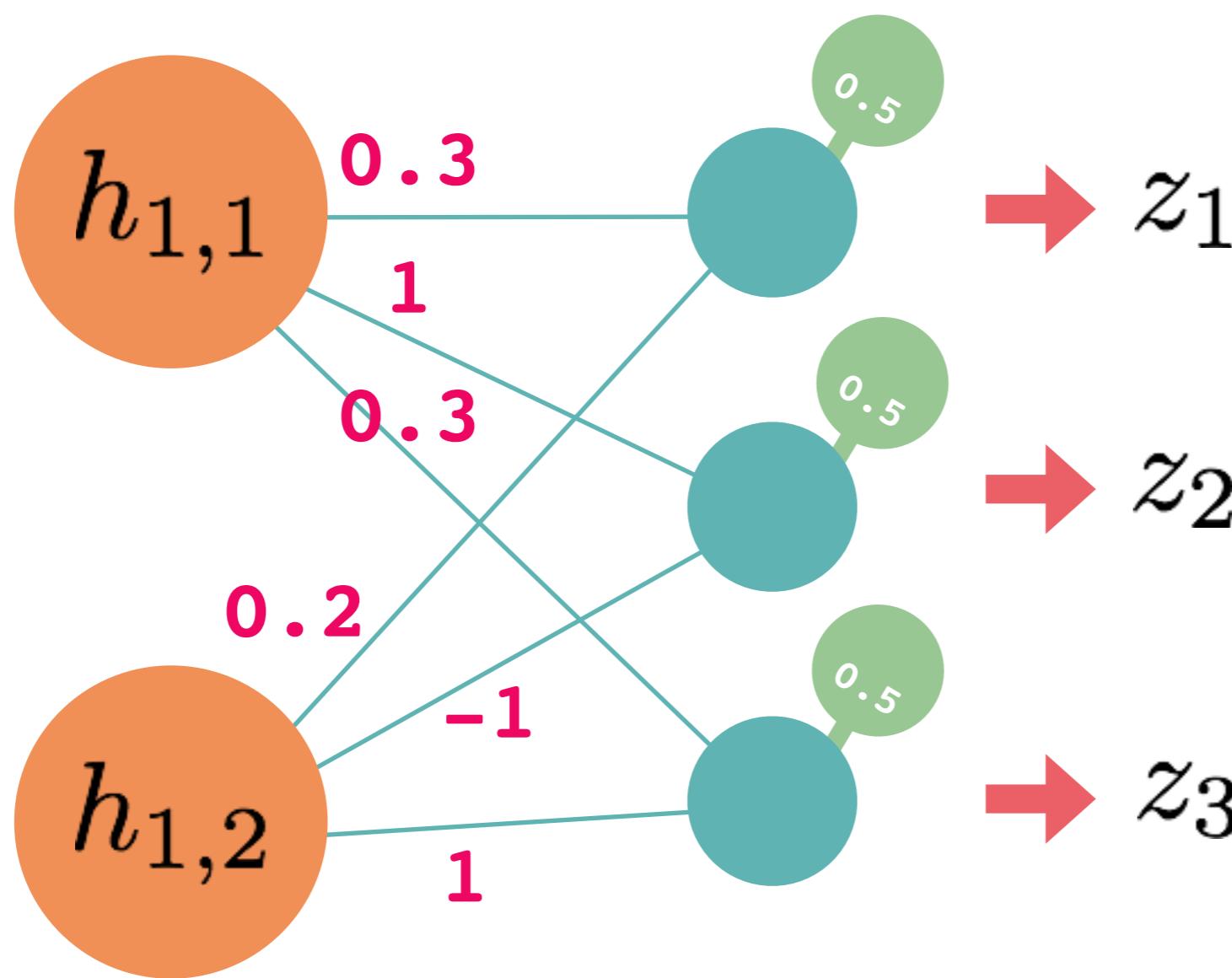
$$W_x = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

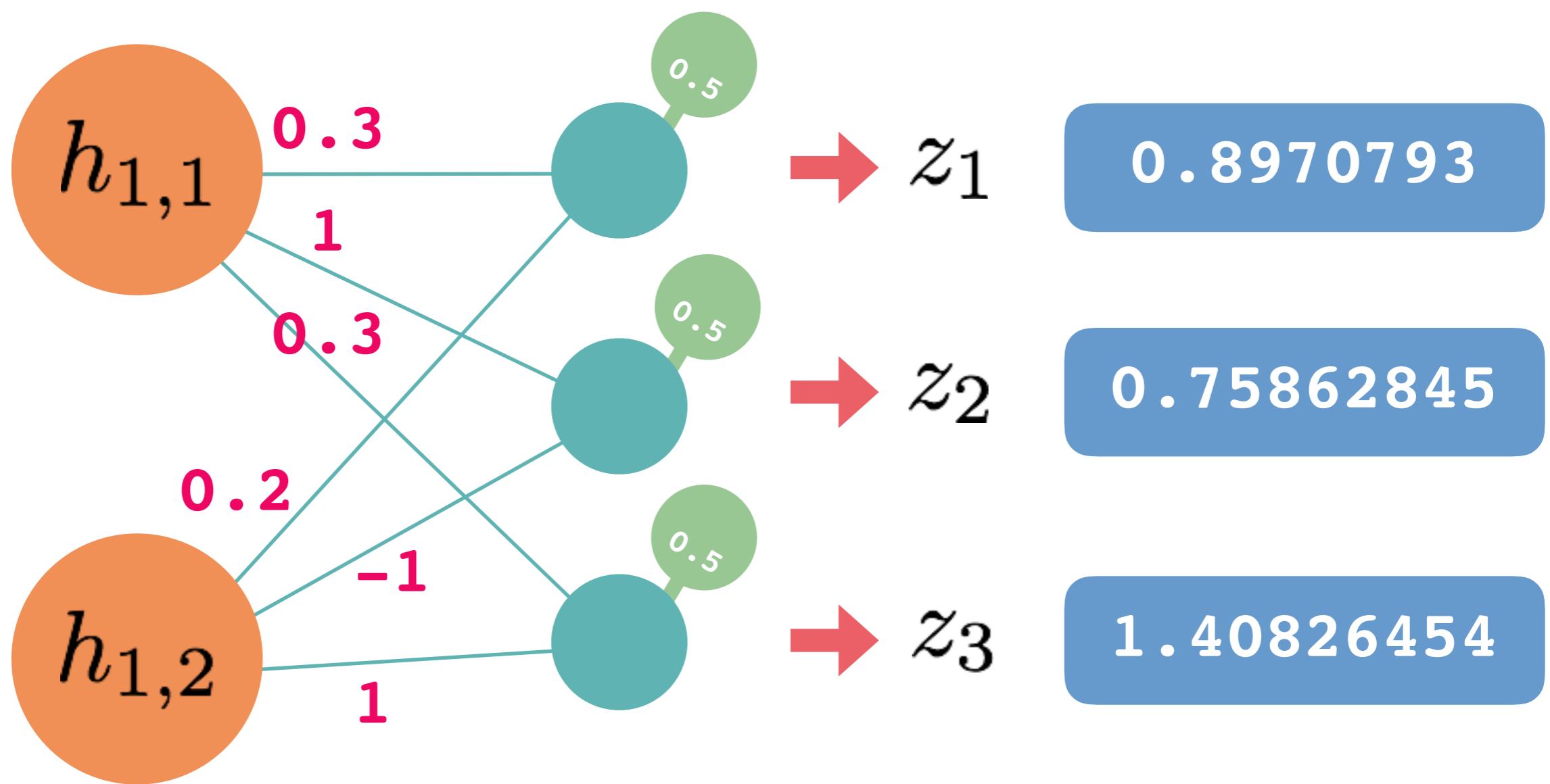
$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_2 = \begin{bmatrix} h_{2,1} \\ h_{2,2} \end{bmatrix} = \begin{bmatrix} 0.89760999 \\ 0.63898154 \end{bmatrix}$$

$$\mathbf{h}_2 = \begin{bmatrix} h_{2,1} \\ h_{2,2} \end{bmatrix} = \begin{bmatrix} 0.89760999 \\ 0.63898154 \end{bmatrix}$$



$$\mathbf{h}_2 = \begin{bmatrix} h_{2,1} \\ h_{2,2} \end{bmatrix} = \begin{bmatrix} 0.89760999 \\ 0.63898154 \end{bmatrix}$$



# 概念

## softmax

$z_1$

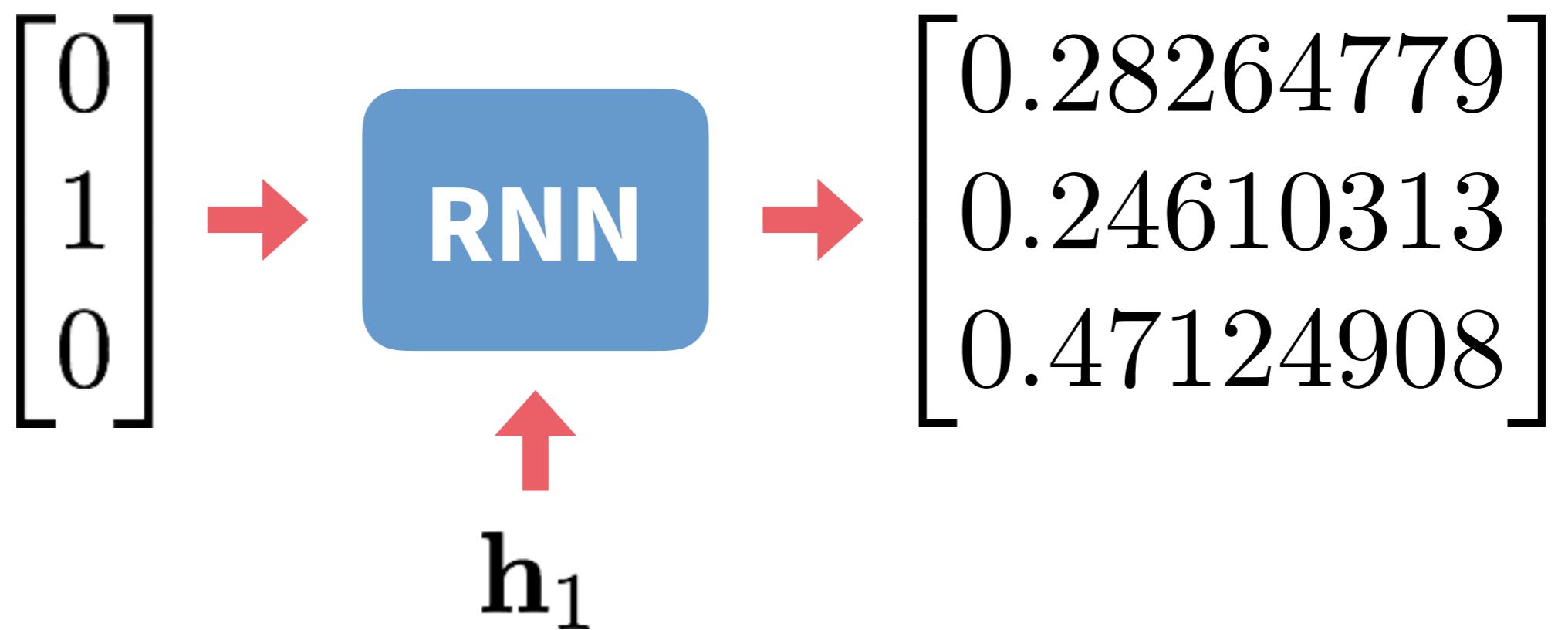
$z_2$

$z_3$

$$\frac{e^{z_1}}{\sum_{i=1}^3 e^{z_i}}$$

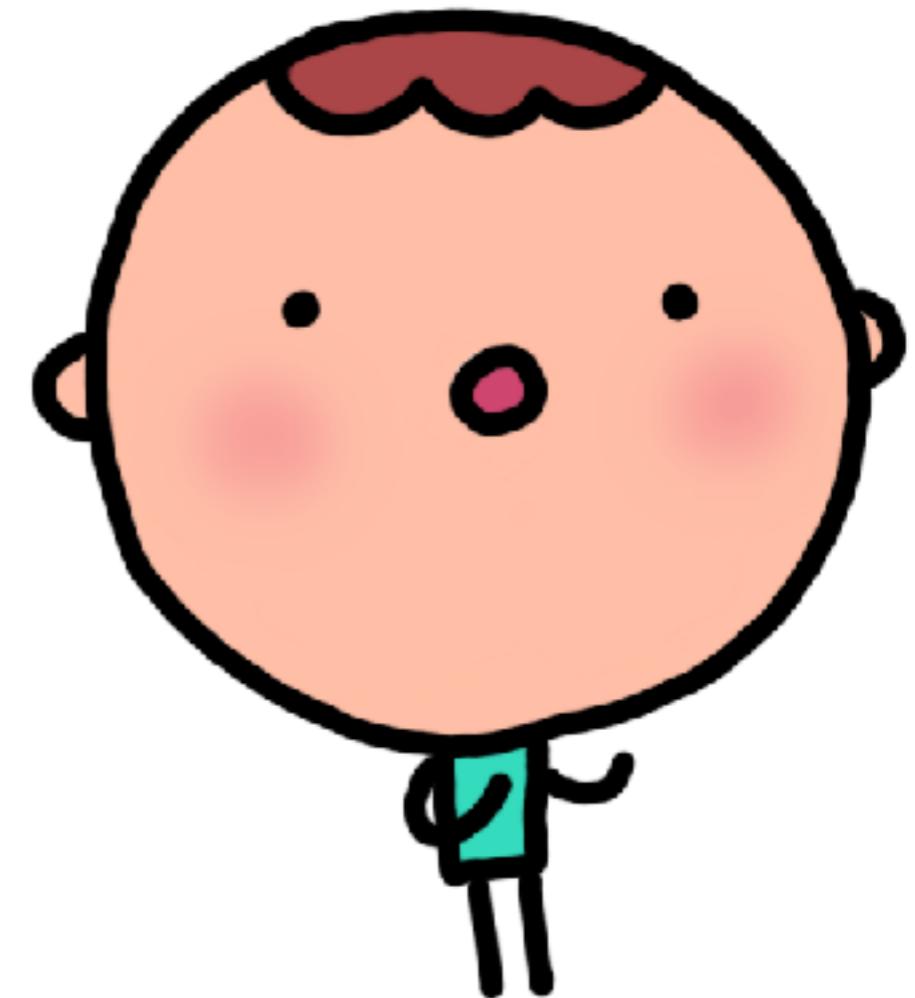
$$\frac{e^{z_2}}{\sum_{i=1}^3 e^{z_i}}$$

$$\frac{e^{z_3}}{\sum_{i=1}^3 e^{z_i}}$$



5

# RNN 的問題



# **Vanishing Gradient**

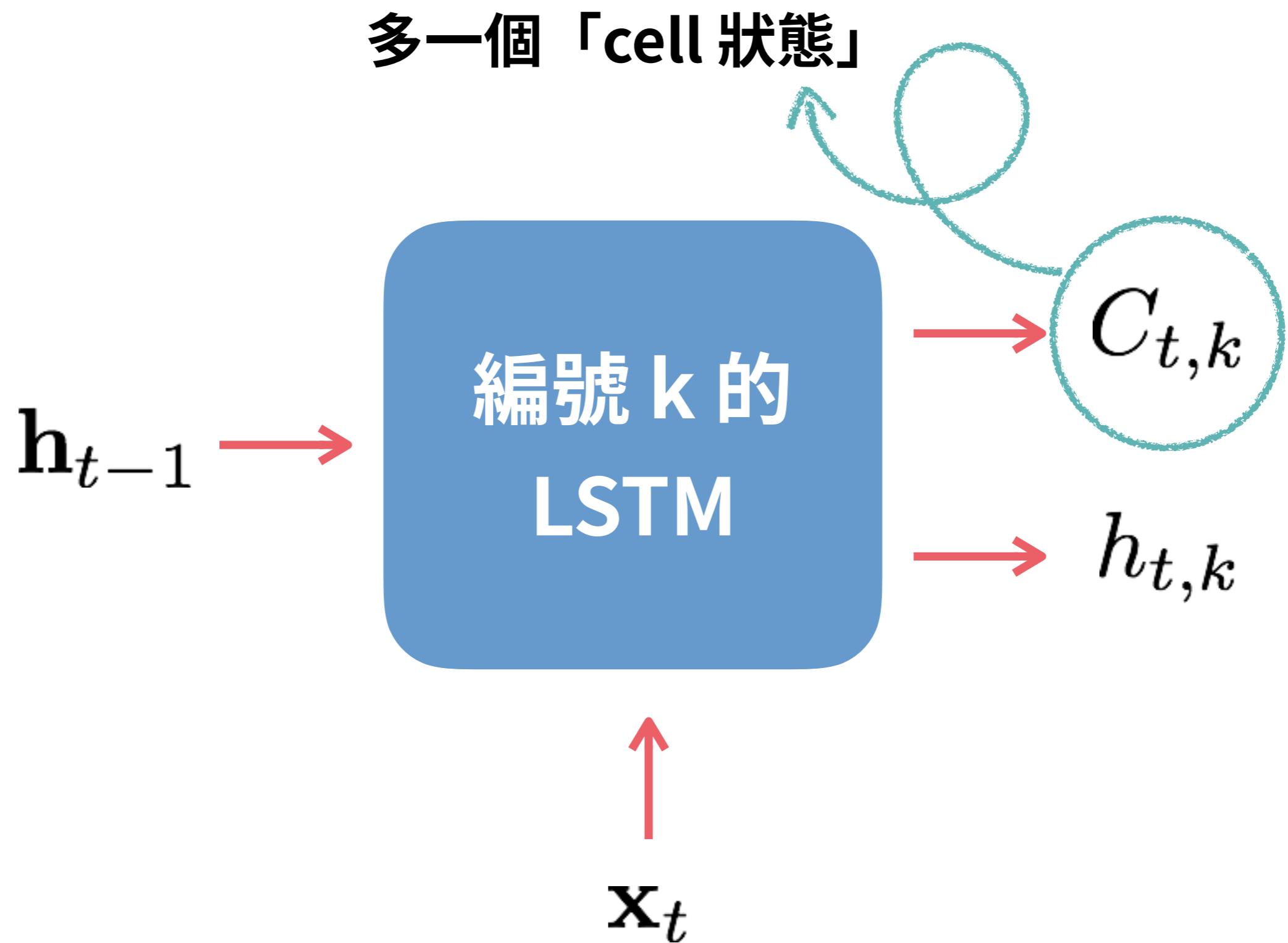
6

LSTM





**Long Short Term Memory**  
**RNN 系的王牌救援**



重要概念

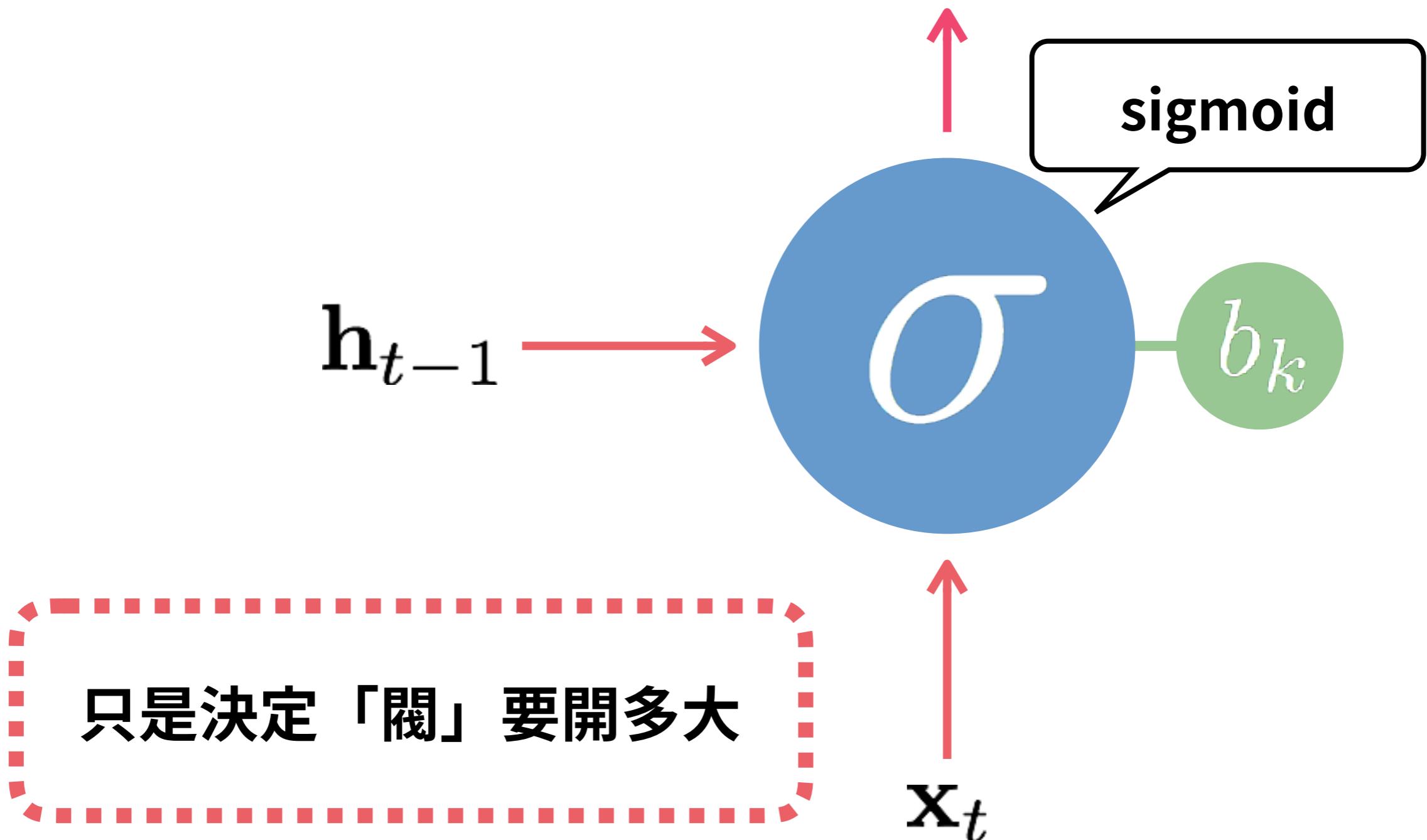
# Gate

控制閥

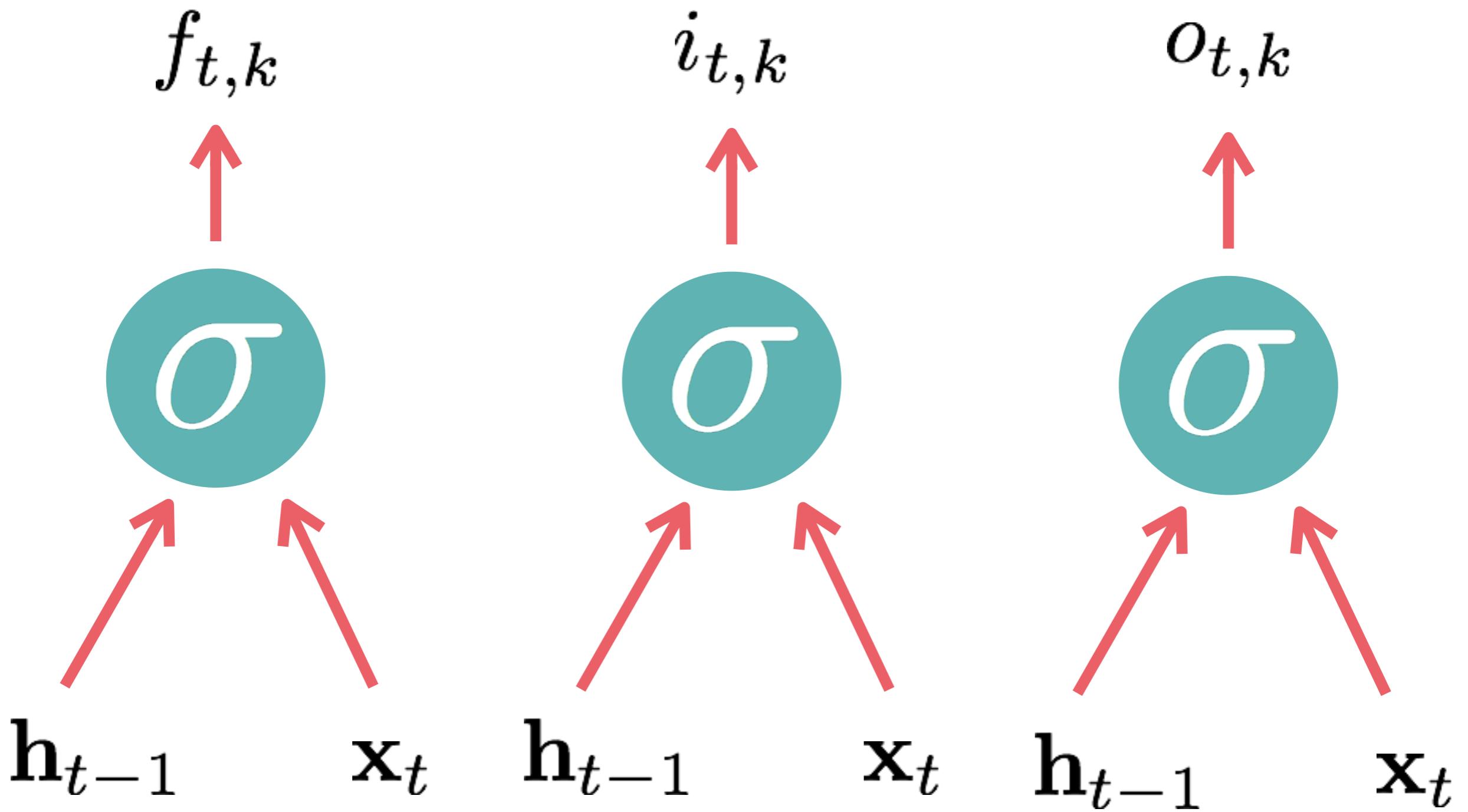
重點

## Gate

輸出 0 到 1 間的一個數



**LSTM 有三個 Gates**

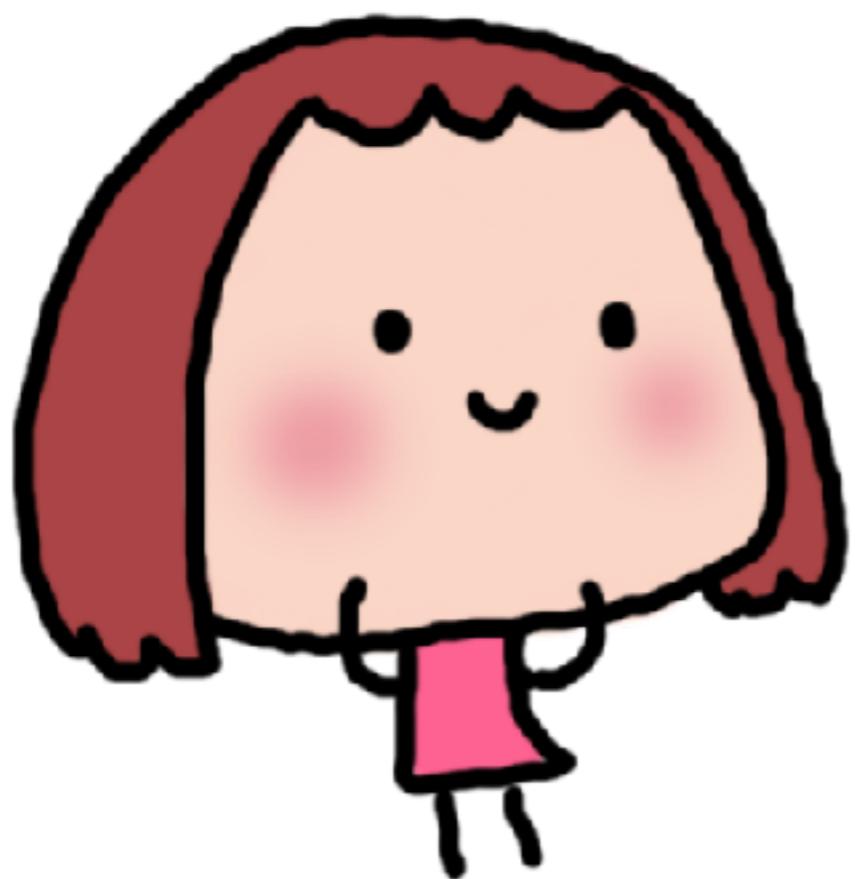


忘記門

輸入門

輸出門

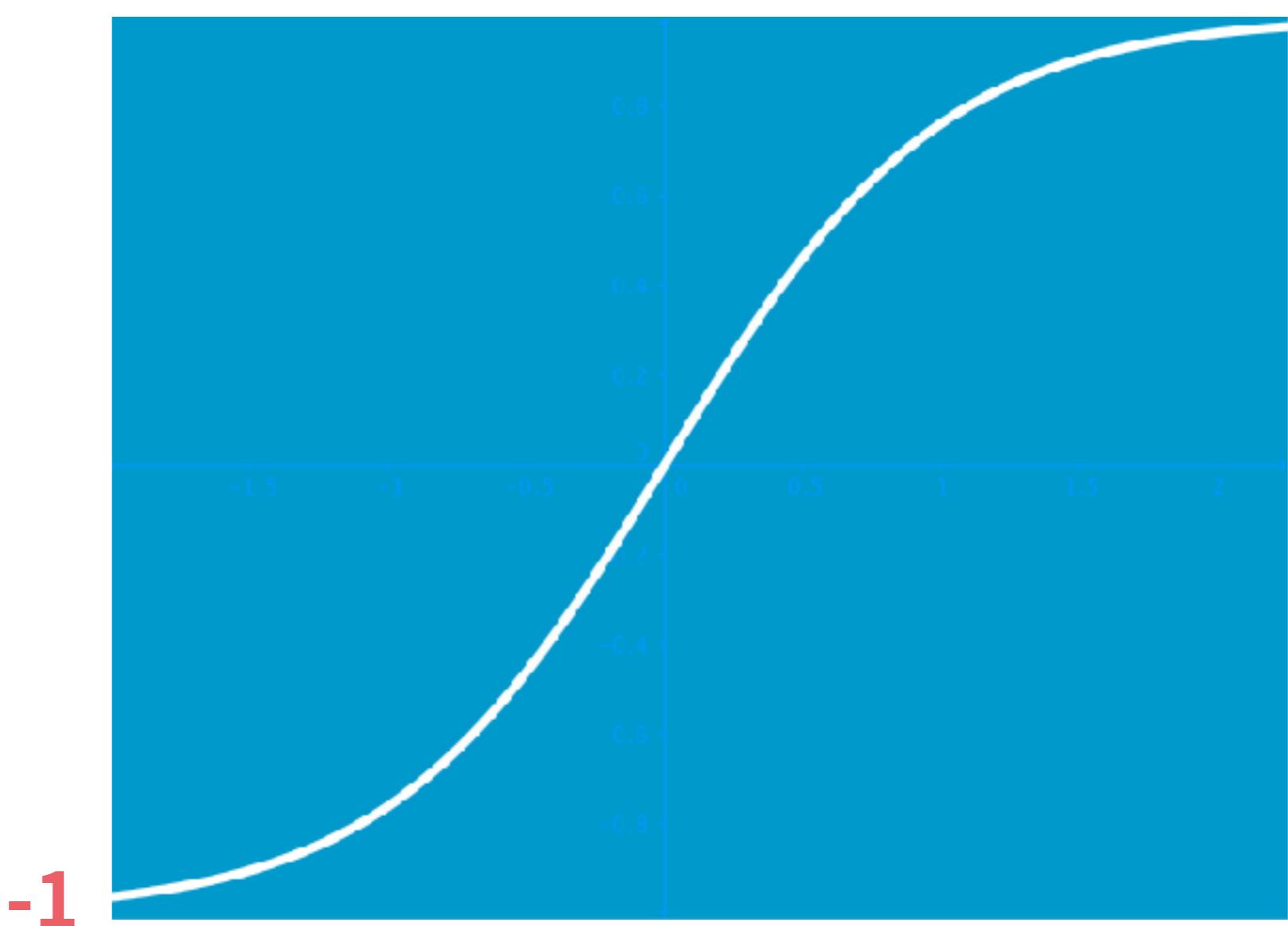
**插播**



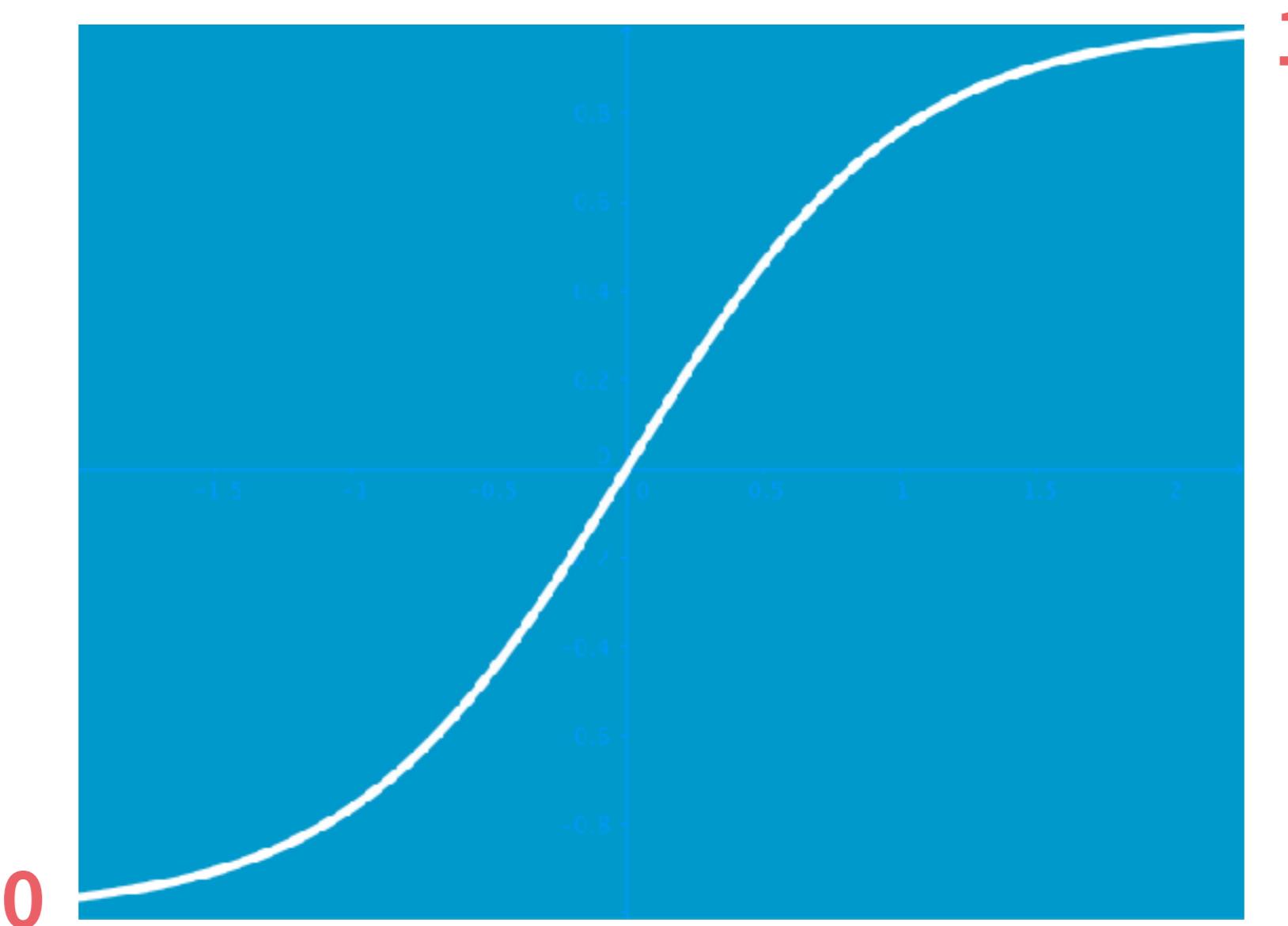
**tanh**

**sigmoid**

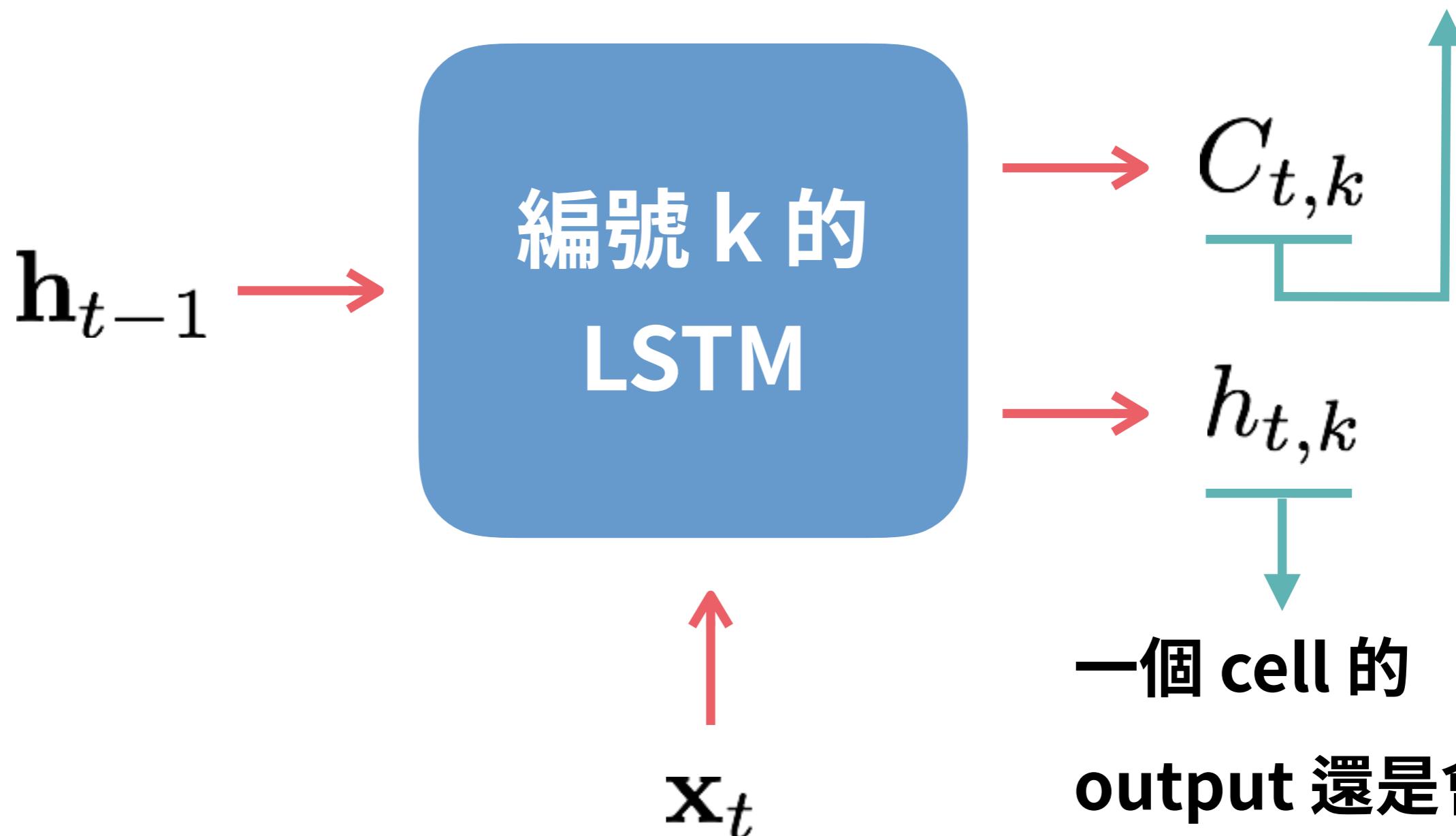
$$\tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$



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



# LSTM 再一次

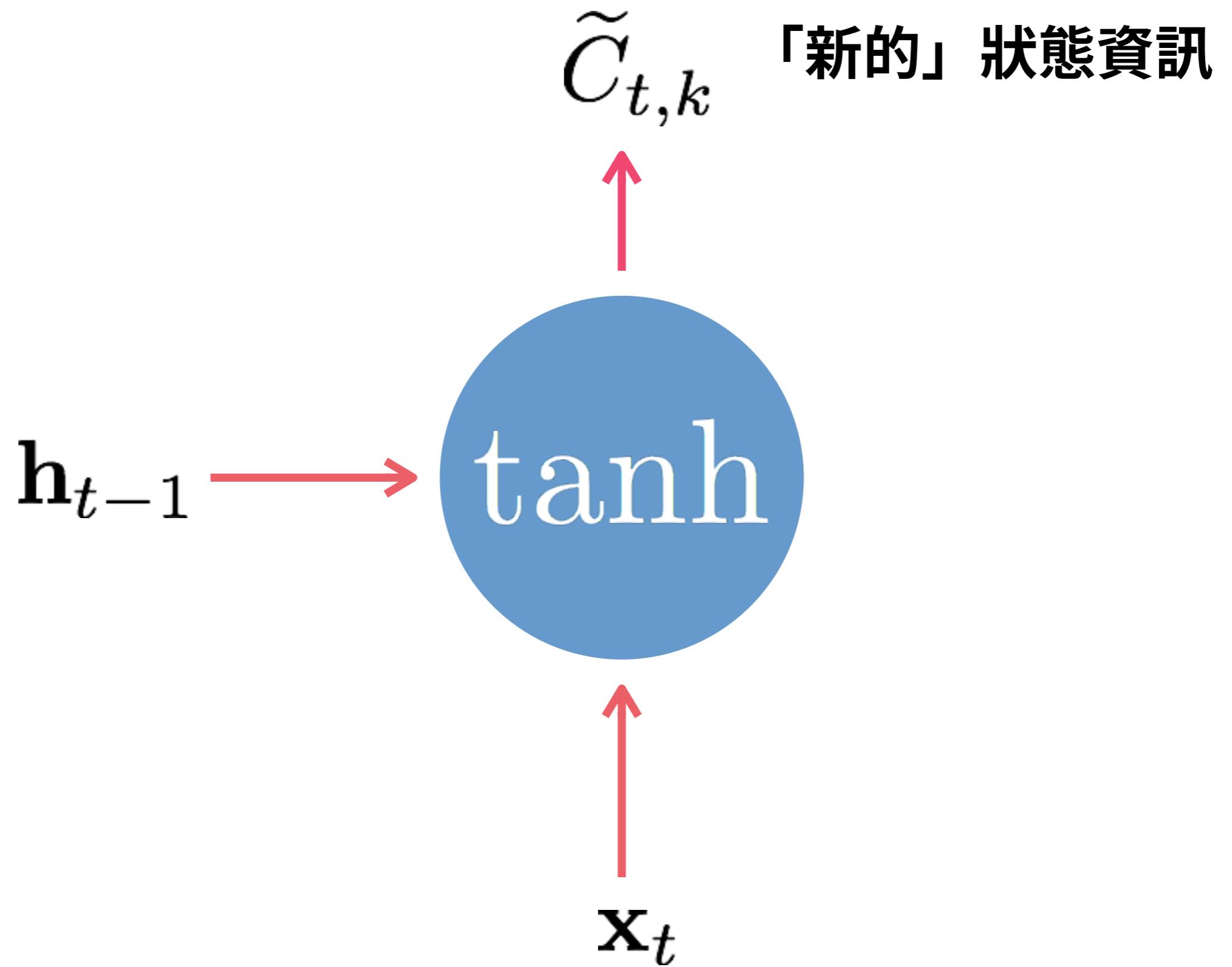


多出來的 cell state  
是屬於這個 cell 的

$$\xrightarrow{} C_{t,k}$$

$$\xrightarrow{} h_{t,k}$$

一個 cell 的  
output 還是會  
和同一層分享



狀態

要忘掉多少？如果是0就會全忘掉

要更新多少？

$$C_{t,k} = f_{t,k} C_{t,k-1} + i_{k,t} \tilde{C}_{t,k}$$

這邊錯了，是  $c(t-1,k)$

輸出

$$h_{t,k} = o_{t,k} \tanh(C_{t,k})$$

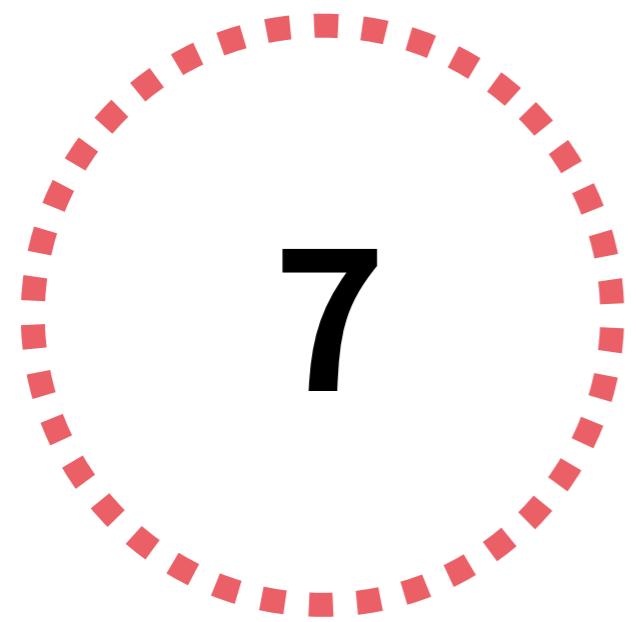
這邊已經考慮到上一個  $h$  的輸入了，可以看上一頁投影片

簡單來說：有個  $C$  在保存狀態， $H$  是我的輸出，然後我的  $C$  會影響到我的  $H$

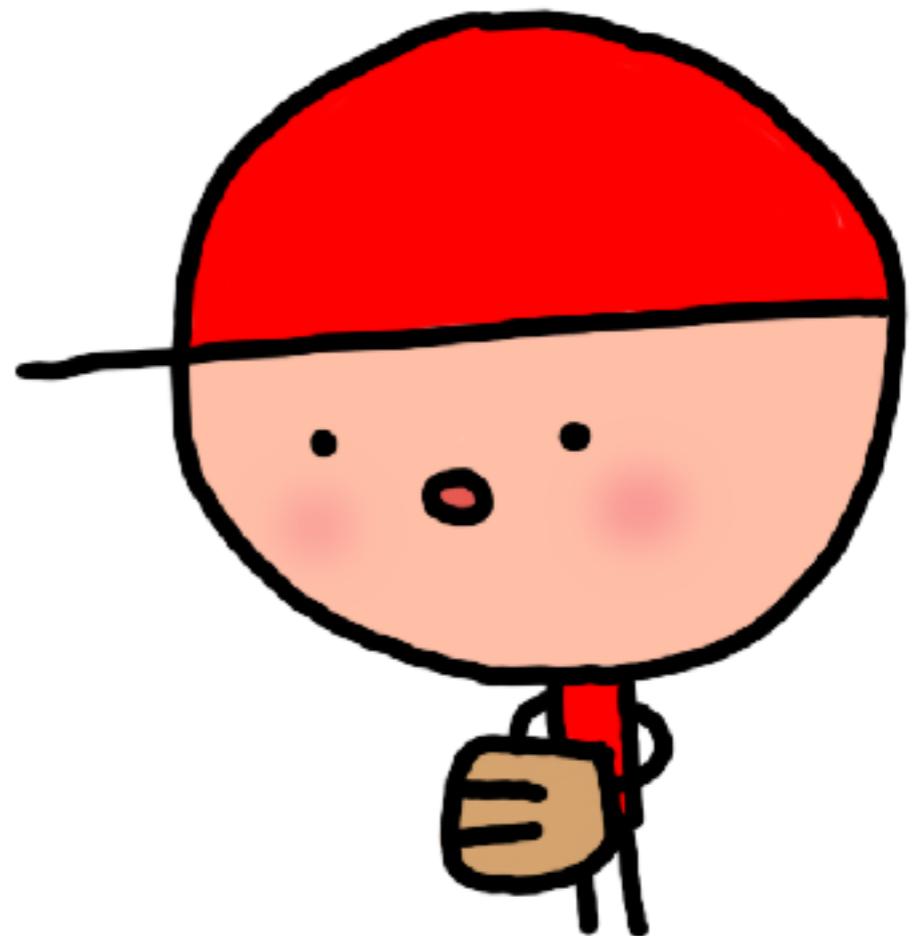
$f, i, o$  都是一個  $0 \sim 1$  的數，但是是獨立分開訓練的

**真要弄得那麼複雜？**

要忘多少和要記多少難道不能一起...

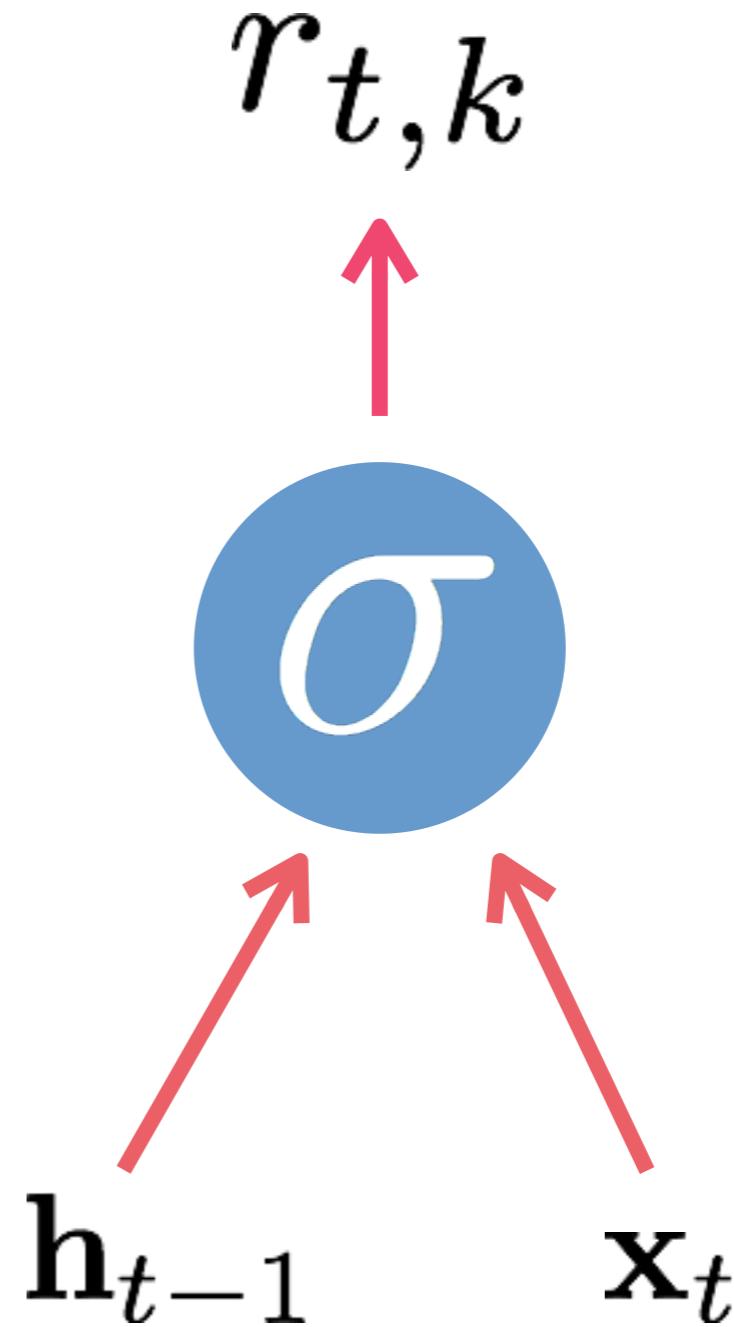
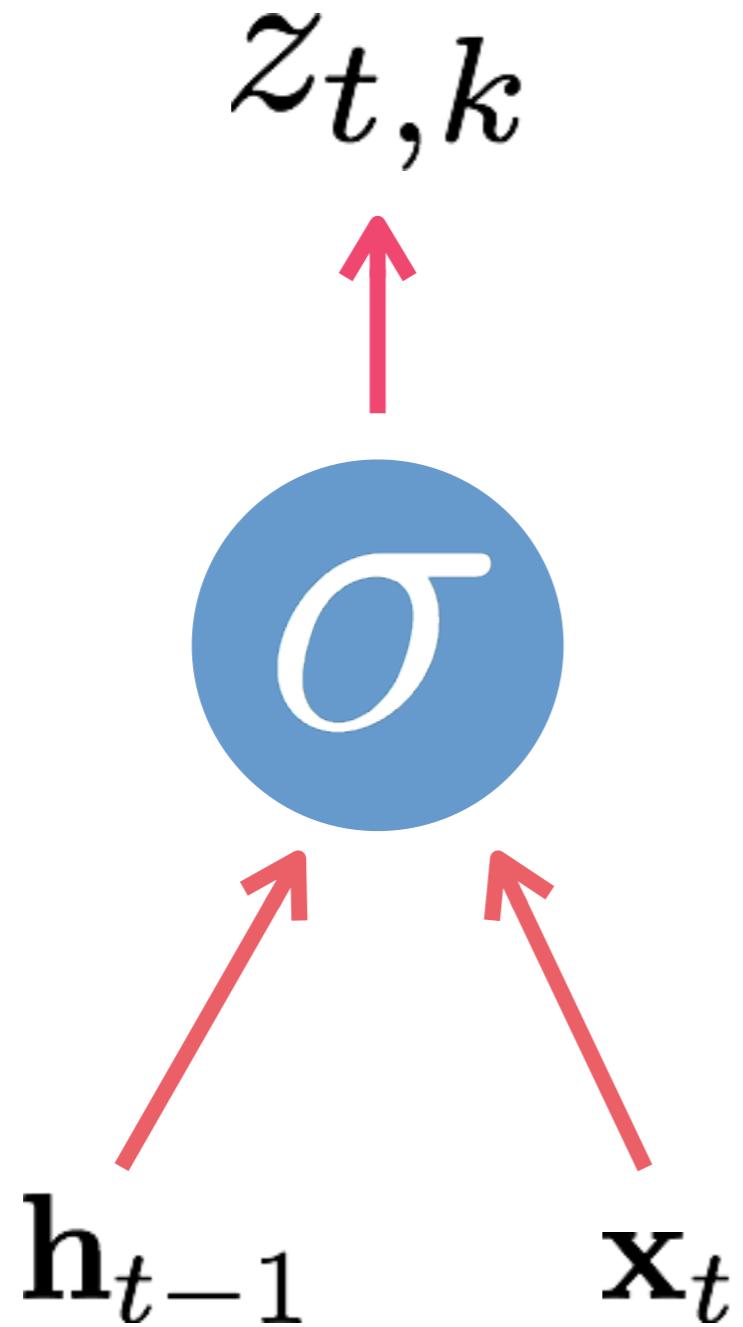


**GRU**



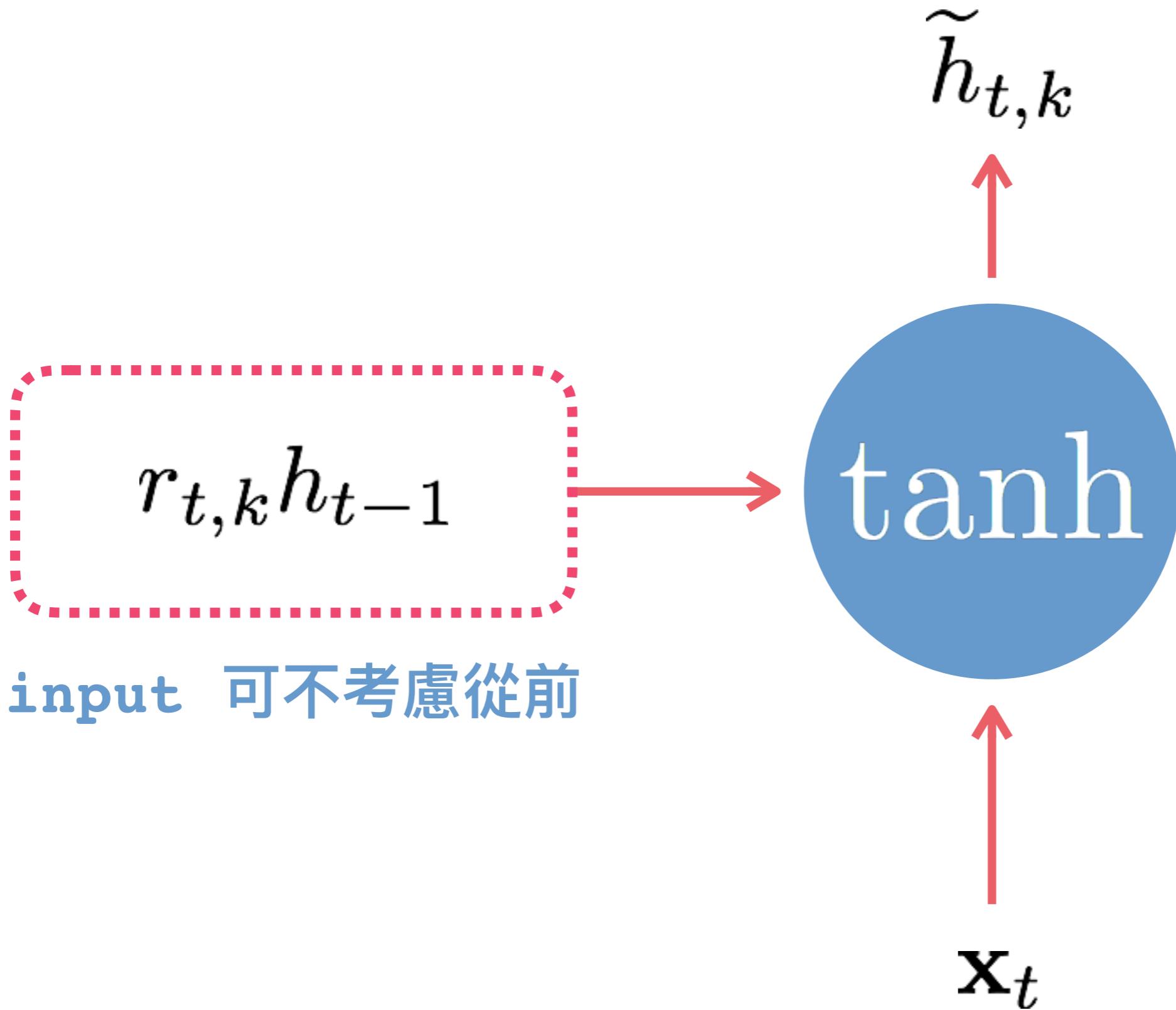
# LSTM 的簡化版

只留兩個 Gates  
雖然名稱有 gated



記憶門

重設門



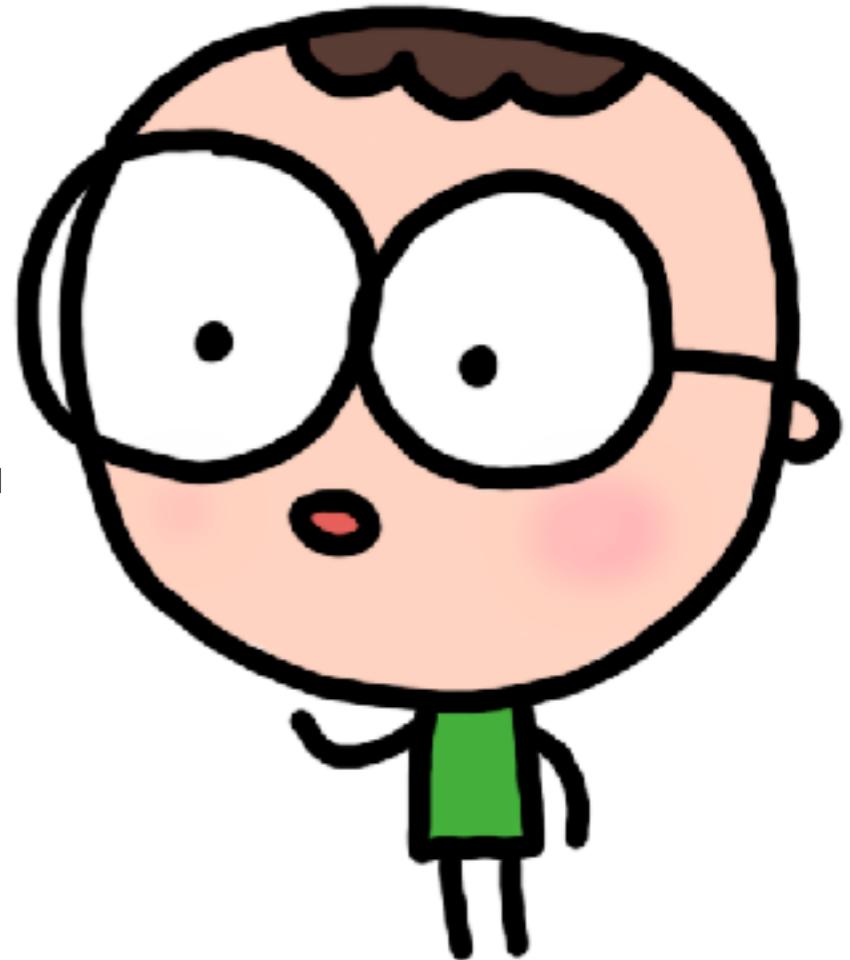
$$h_{t,k} = z_{t,k} h_{t-1,k} + (1 - z_{t,k}) \tilde{h}_{t,k}$$

重點

## RNN 的名字

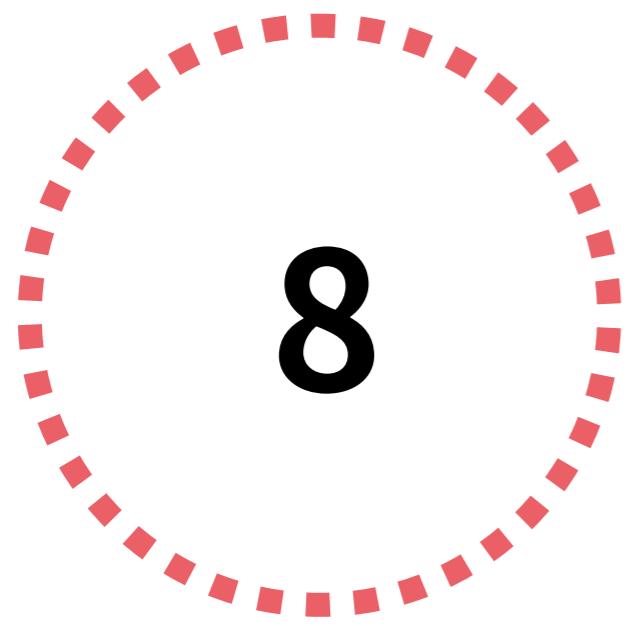
現在說到 RNN，其實包括原始 RNN, LSTM, GRU 等各種變形。

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



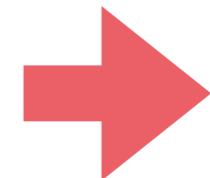
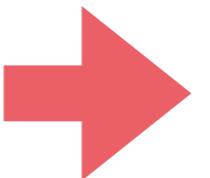
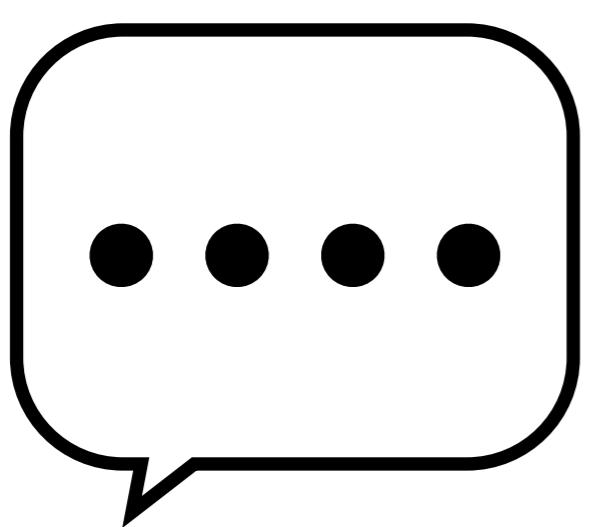
# 實戰篇。





# IMDB 評論





正評  
or  
負評

技巧

## 指定 Backend

在 Jupyter Notebook 中，我們可以指定 Keras 使用的 Backend。

```
%env KERAS_BACKEND=tensorflow
```

技巧

## 讀入 IMDB 資料庫

在 Jupyter Notebook 中，我們可以指定 Keras 使用的 Backend。

```
from keras.datasets import imdb
```

## 重點

# 讀入資料

Keras 轉備好的資料都用 `load_data` 讀入。

```
(x_train, y_train), (x_test, y_test)  
= imdb.load_data(num_words=10000)
```

這裡限制只選最常用 10,000 字。訓練資料和測試資料都是很豪氣的 25,000 筆。

說明

## 資料內容

我們來看一筆輸入資料長像：

```
[1, 1230, 3765, 566, 97,  
189, 102, 86, ...]
```

每個數字代表一個字，這就是一則評論。

注意

## 資料長度

每一則評論長度顯然是不一樣的!

```
for i in range(10):
    print(len(x_train[i]), end=', ')
```

輸出: 218, 189, 141, 550, 147, 43, 123, 562, 233,  
130

9

# 輸入處理



重點

## 輸入處理

```
from keras.preprocessing import sequence
```

重點

## 輸入處理

```
x_train = sequence.pad_sequences(x_train,  
maxlen=100)
```

```
x_test = sequence.pad_sequences(x_test,  
maxlen=100)
```

重點

長度變成一樣！

`x_train.shape`

輸出: (25000, 100)

10

打造 RNN



# 決定架構

- 先將 10000 維的文字壓到 128 維
- 然後用 128 個 LSTM
- 最後一個 output, 直接用 sigmoid 送出

重點

## 讀入函數

```
from keras.models import Sequential  
from keras.layers import Dense, Embedding  
from keras.layers import LSTM
```

重點

# 打開新的神經網路

```
model = Sequential()
```

重點

## Embedding 層

```
model.add(Embedding(10000, 128))
```

重點

## LSTM 層

```
model.add(LSTM(150))
```

重點

## Dense 層 (output)

```
model.add(Dense(1, activation='sigmoid'))
```

11

組裝



重點

## 組裝

---

```
model.compile(loss='binary_crossentropy',  
              optimizer='adam',  
              metrics=[ 'accuracy' ])
```

12

# 訓練



重點

## 訓練

```
model.fit(x_train, y_train,  
          batch_size=32,  
          epochs=15)
```

13

結果



重點

## 訓練

---

```
score = model.evaluate(x_test, y_test)
```

重點

## 訓練

---

```
model_json = model.to_json()
open('imdb_model_architecture.json',
      'w').write(model_json)
model.save_weights('imdb_model_weights.h5')
```

14

結果



重點

## 訓練成績

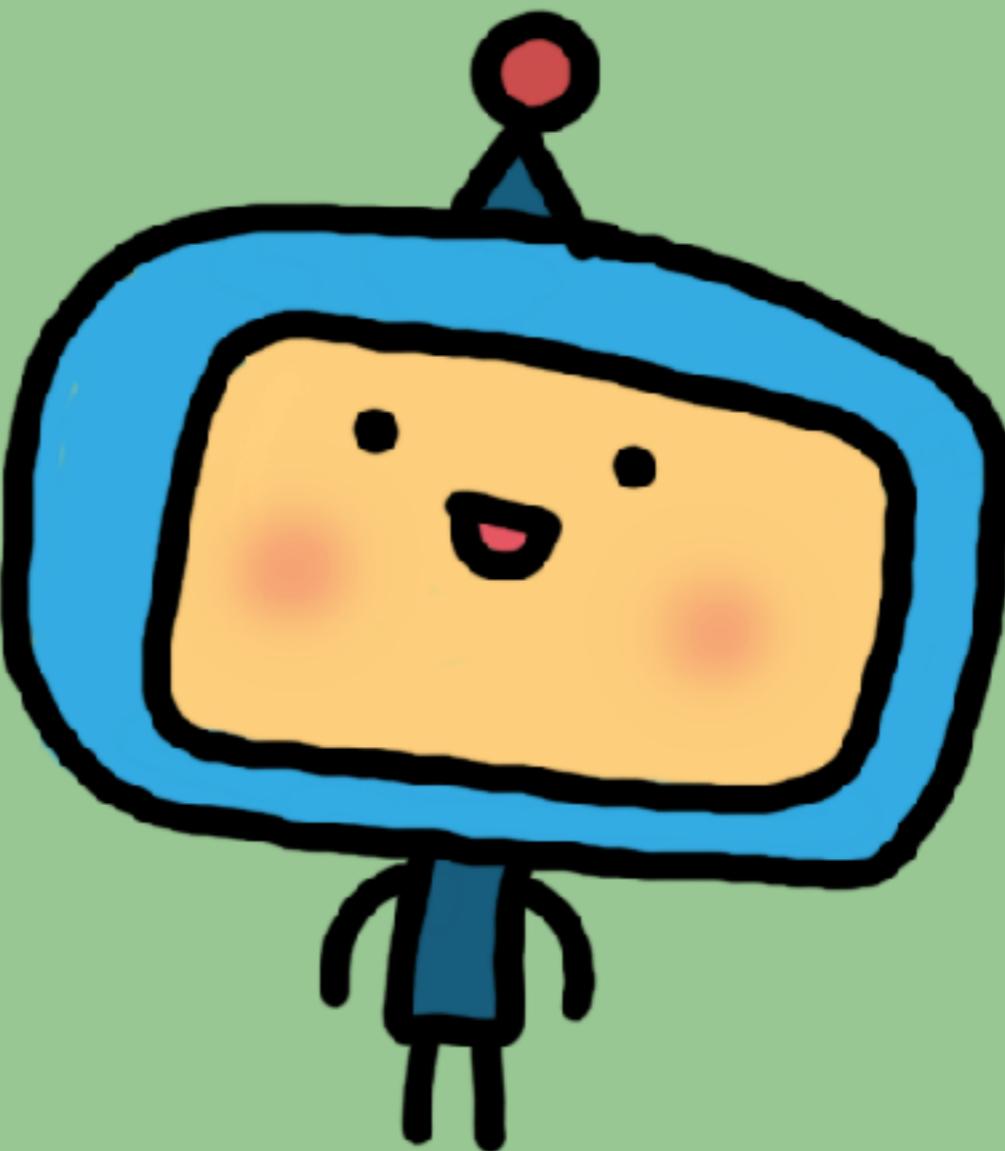
```
print('測試資料的 loss:', score[0])  
print('測試資料正確率:', score[1])
```

重點

## 儲存結果

```
model_json = model.to_json()  
open('imdb_model_architecture.json',  
     'w').write(model_json)  
model.save_weights('imdb_model_weights.h5')
```

# 調校篇。



15

# 基本概念 複習

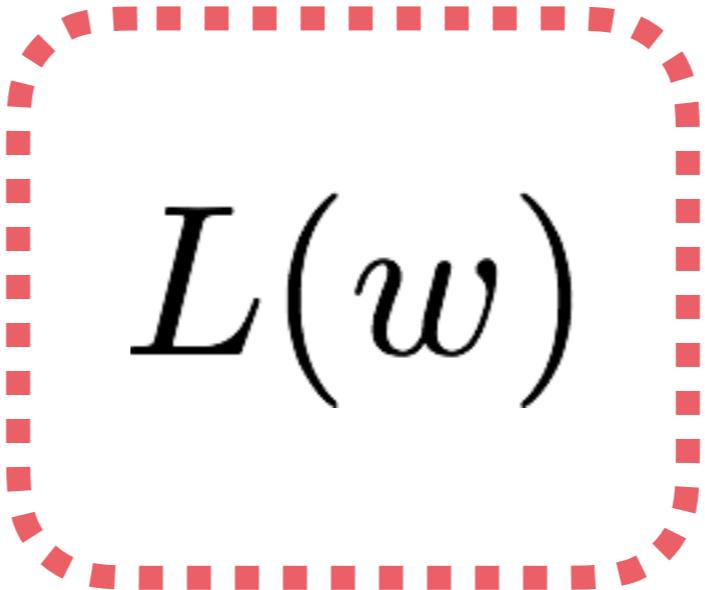


學習都是用我們訓練資料送進我們的神經網路，調整我們的參數。

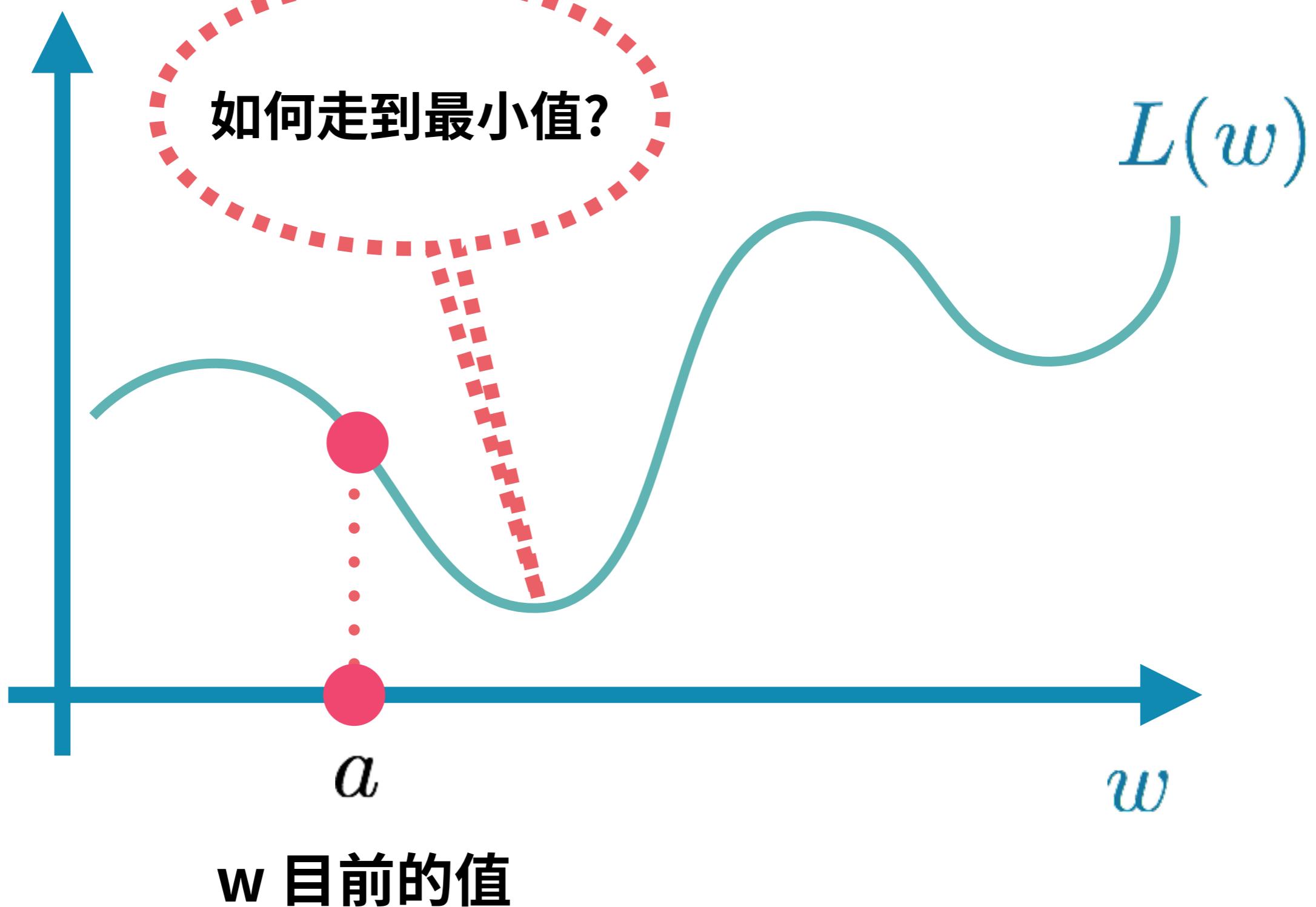
$$\theta = \{w_i, b_j\}$$

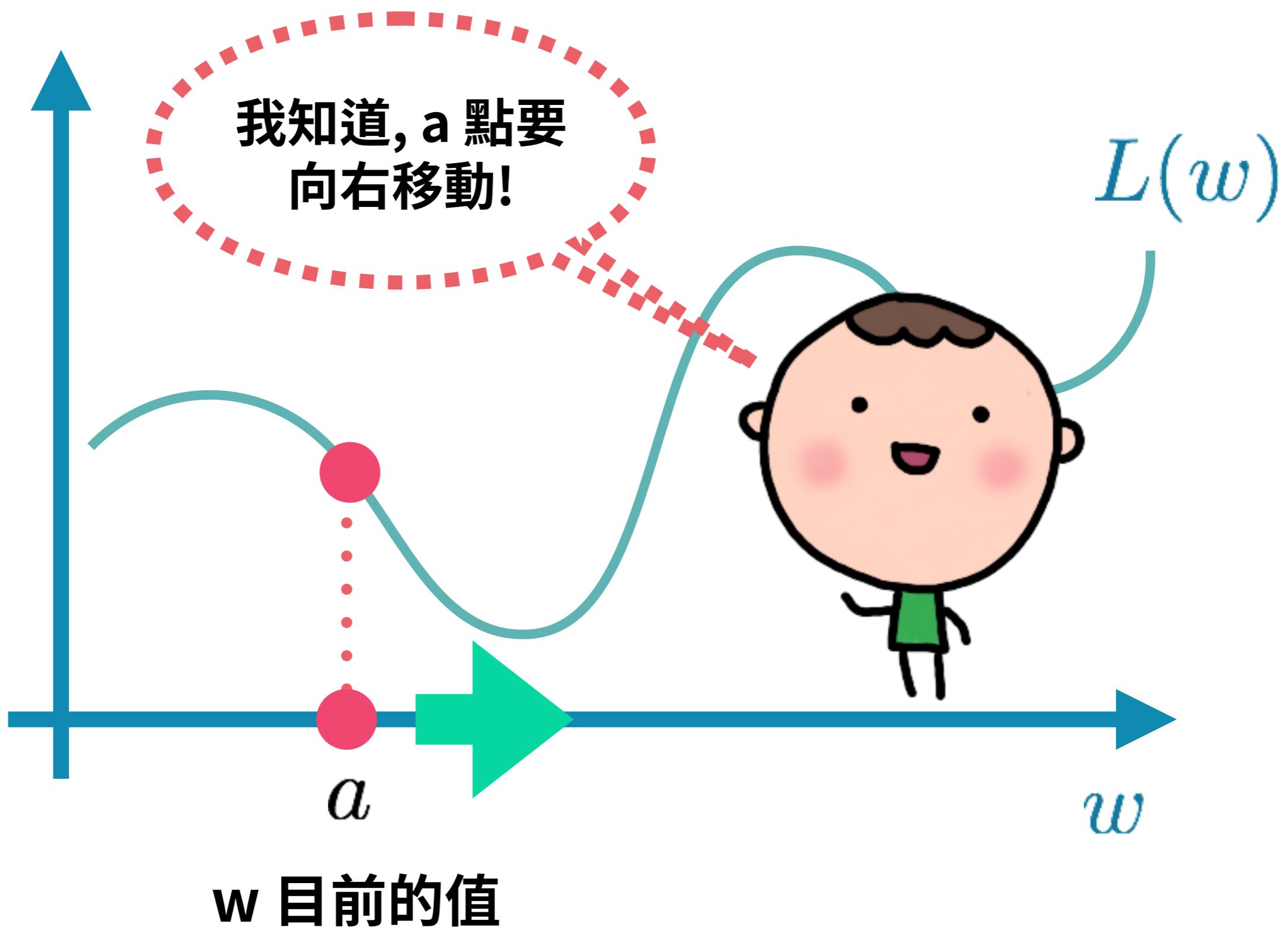
然後用個 **loss function** 看我們和觀察值差多少。

$$L(\theta)$$



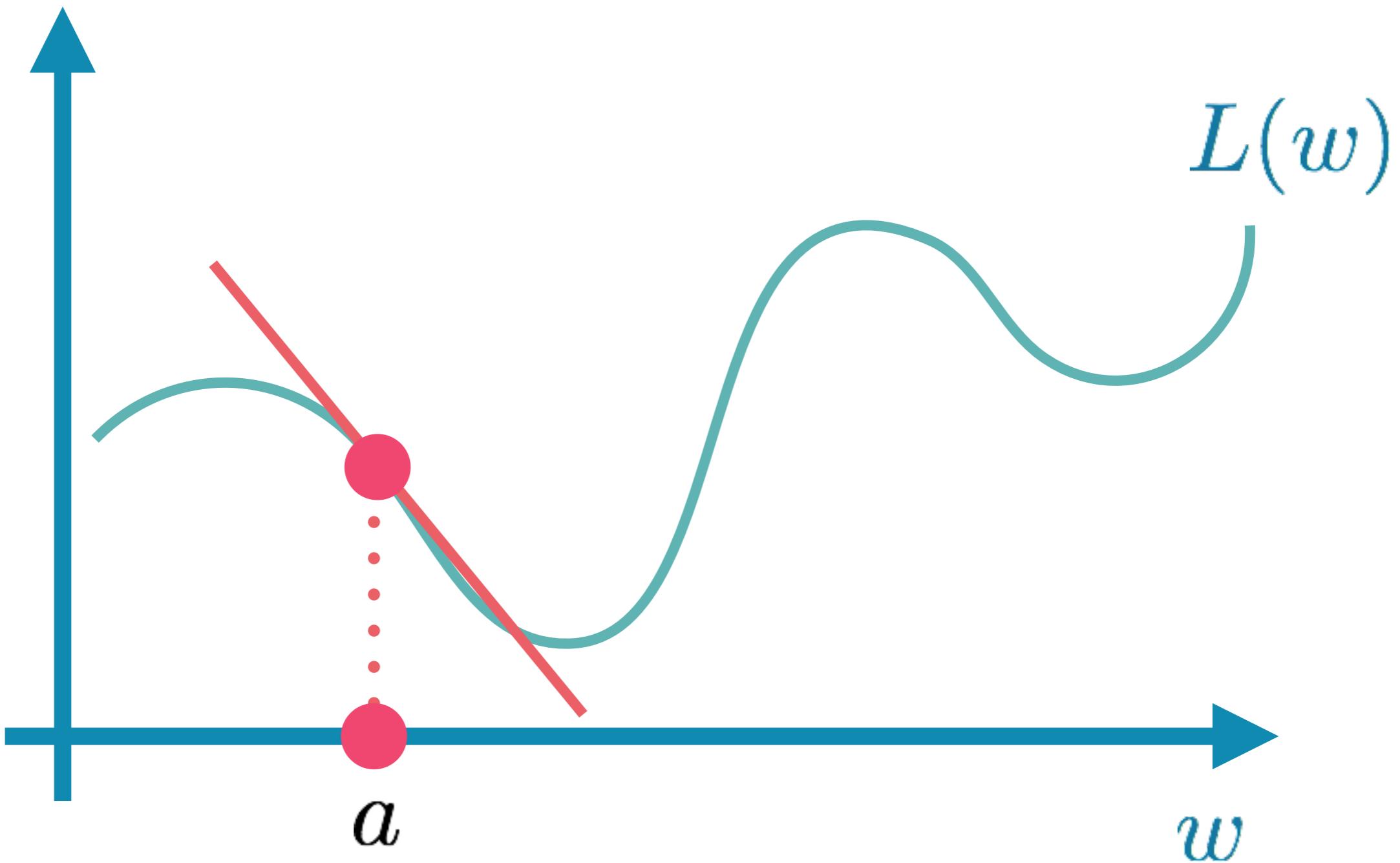
為了簡化, 我們先把  $L$  想成  
只有一個變數  $w$ 。



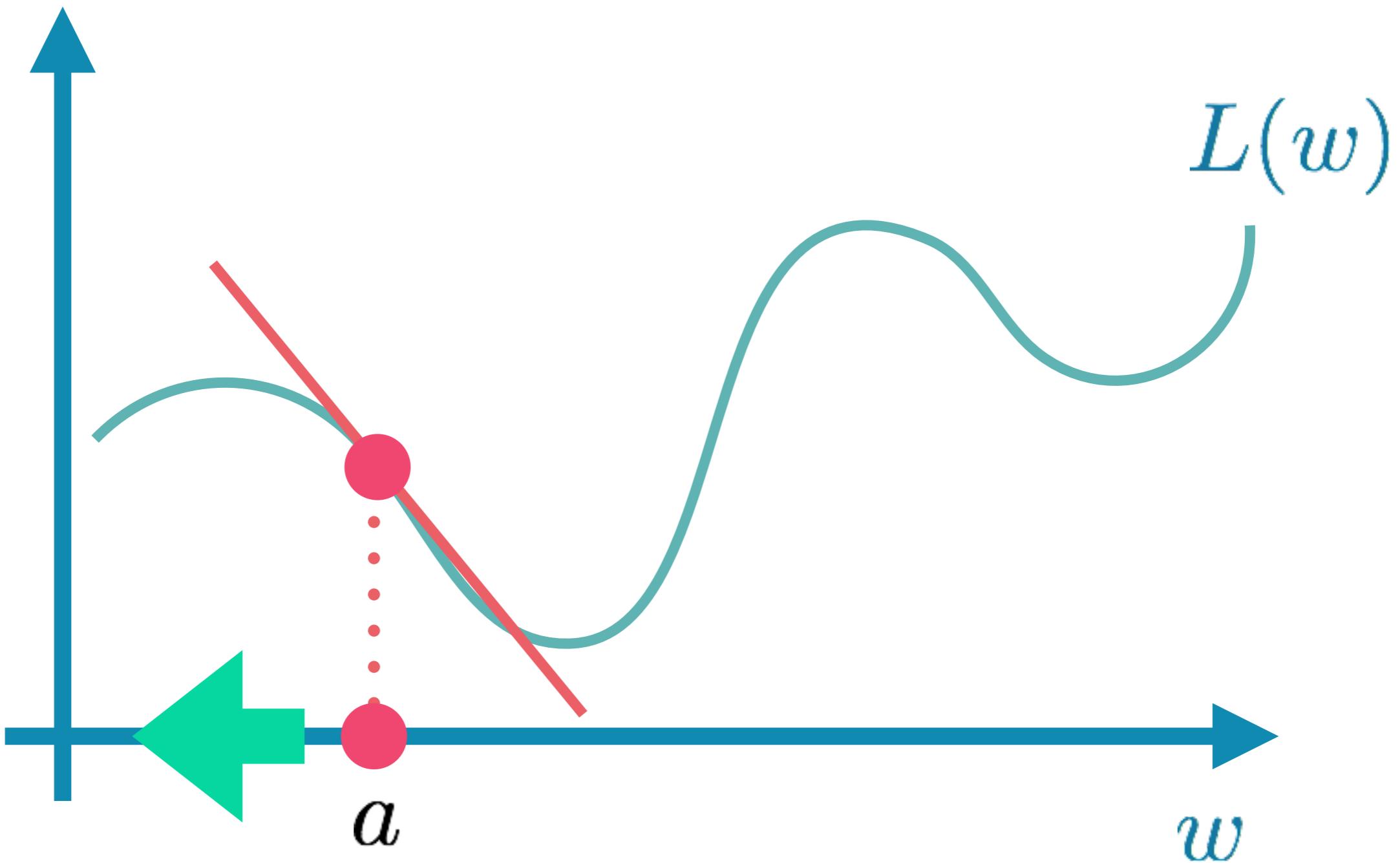




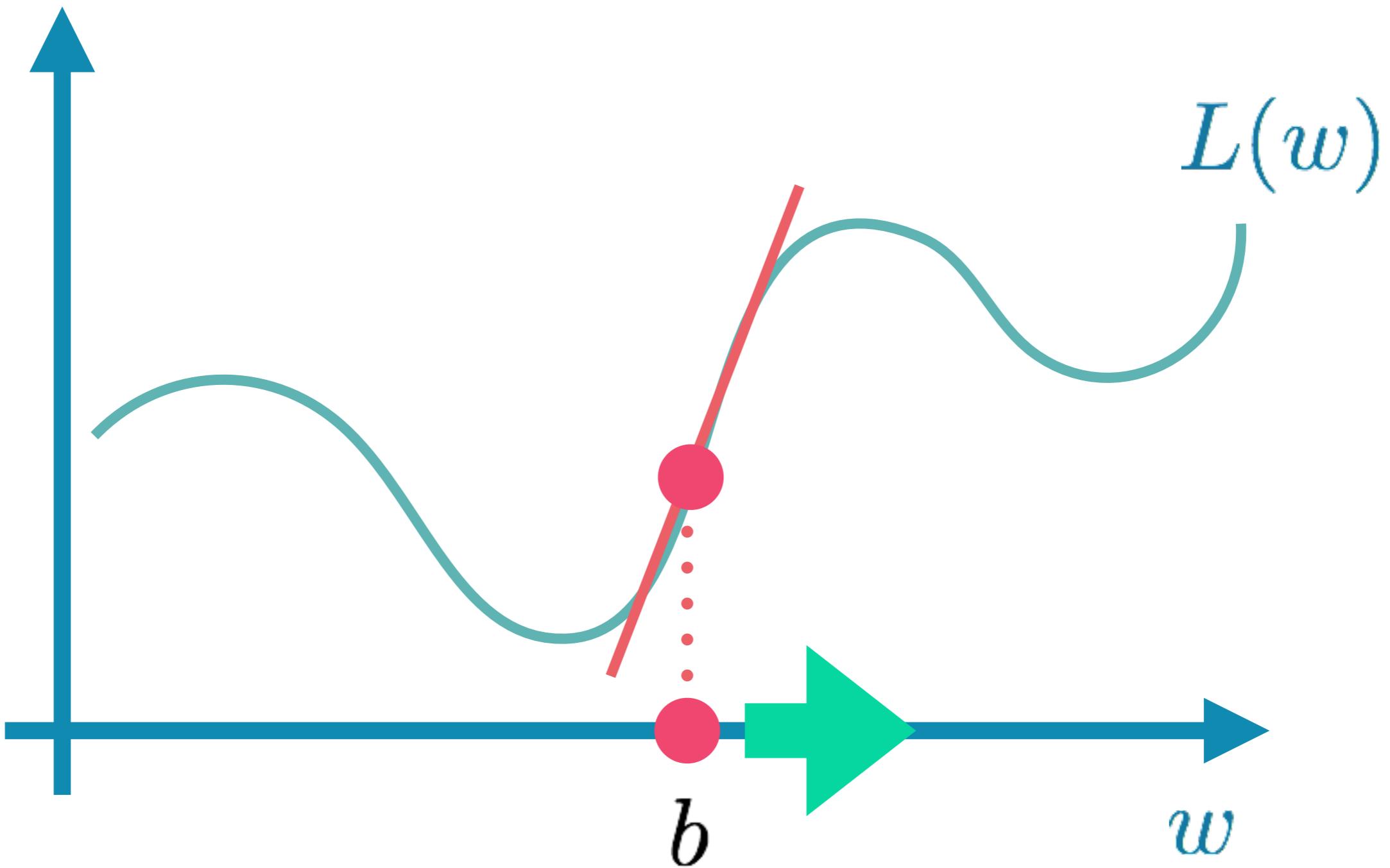
# 切線是關鍵！



切線斜率  $< 0$

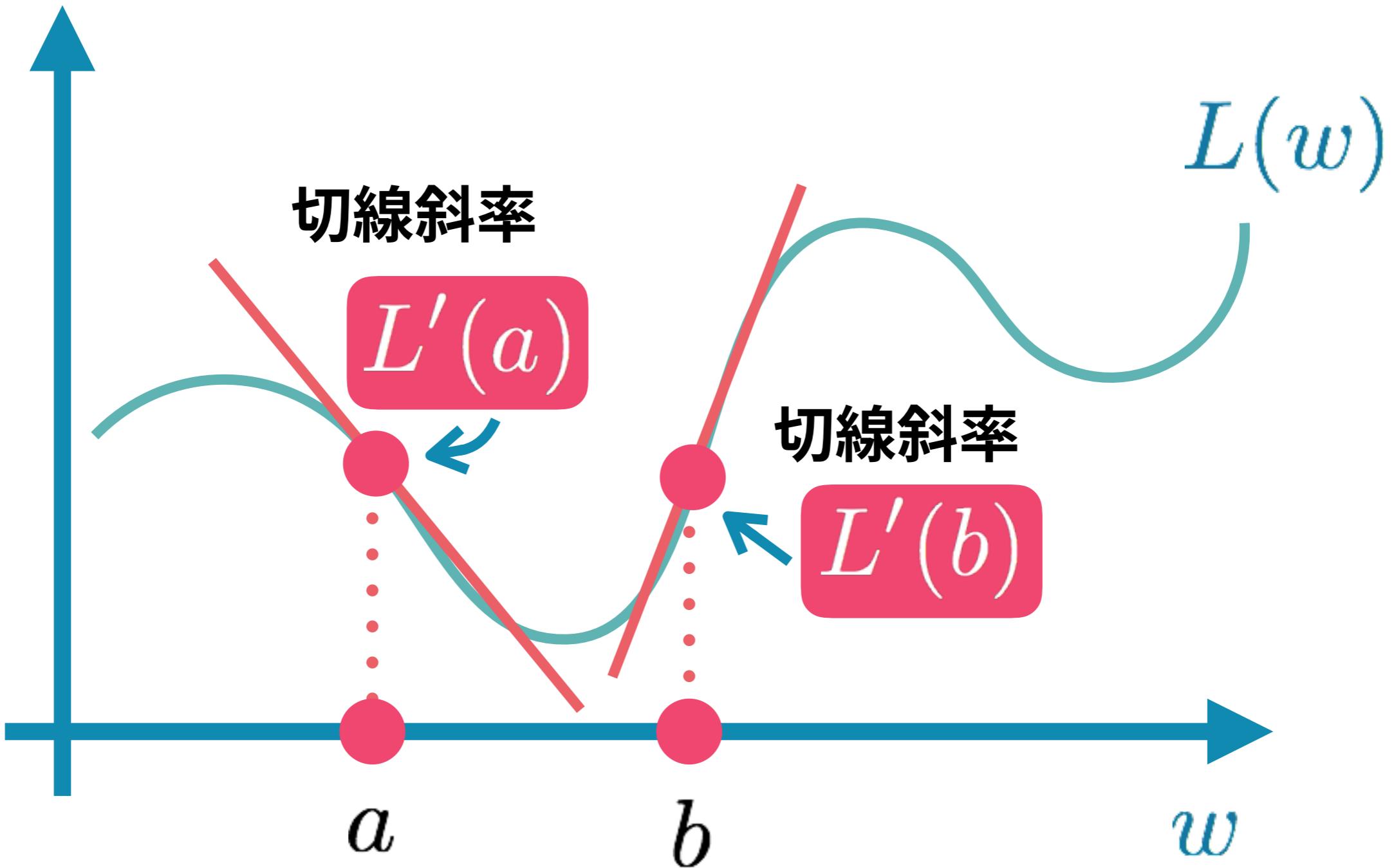


切線斜率  $> 0$

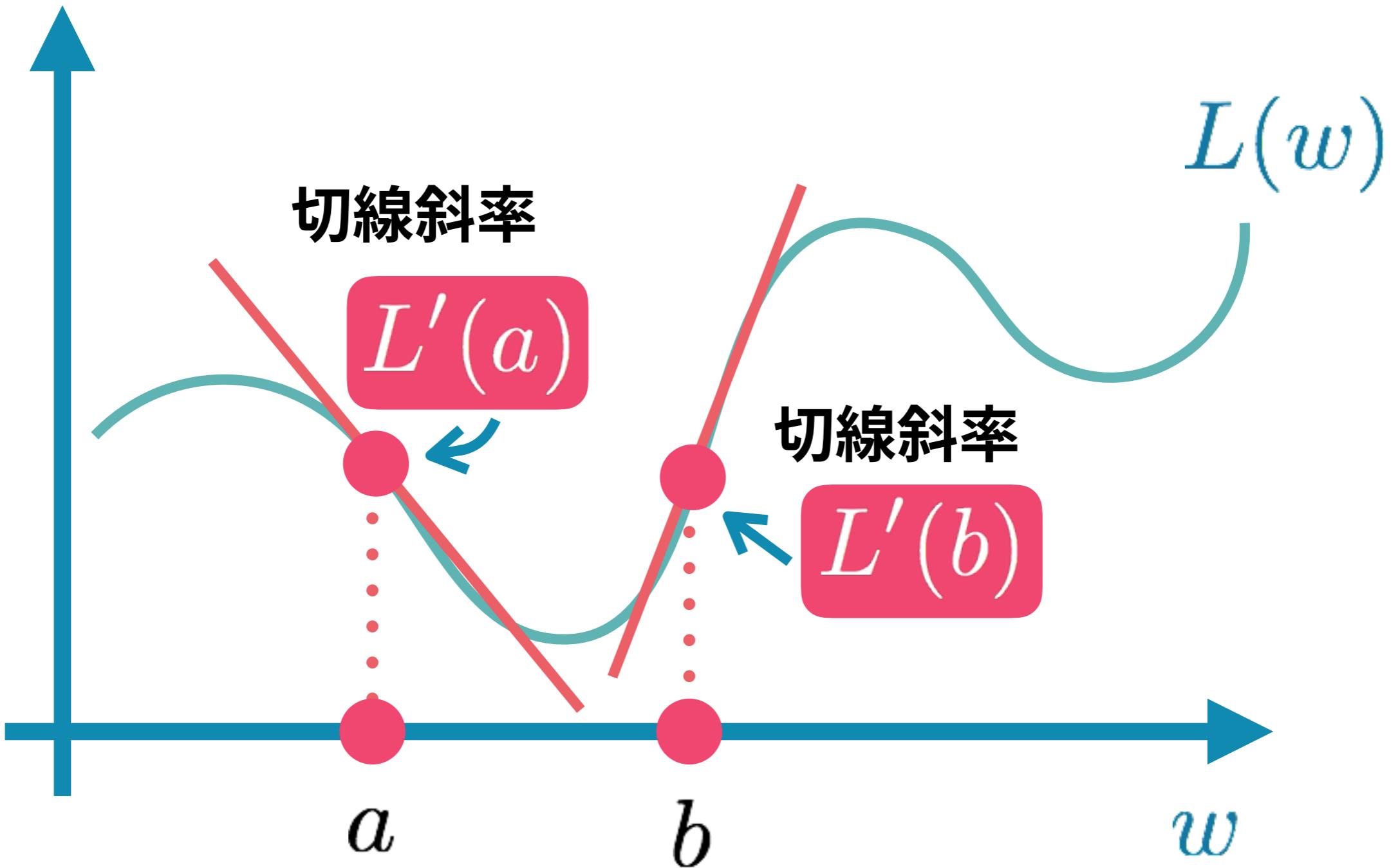


**切線斜率指向 (局部) 最  
大值的方向!!**

# 符號



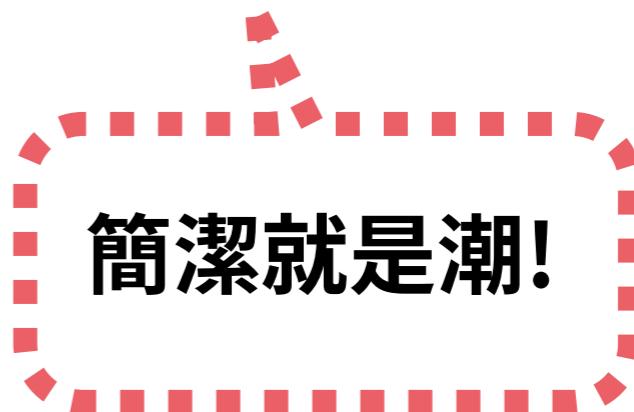
# 切線斜率是變化率



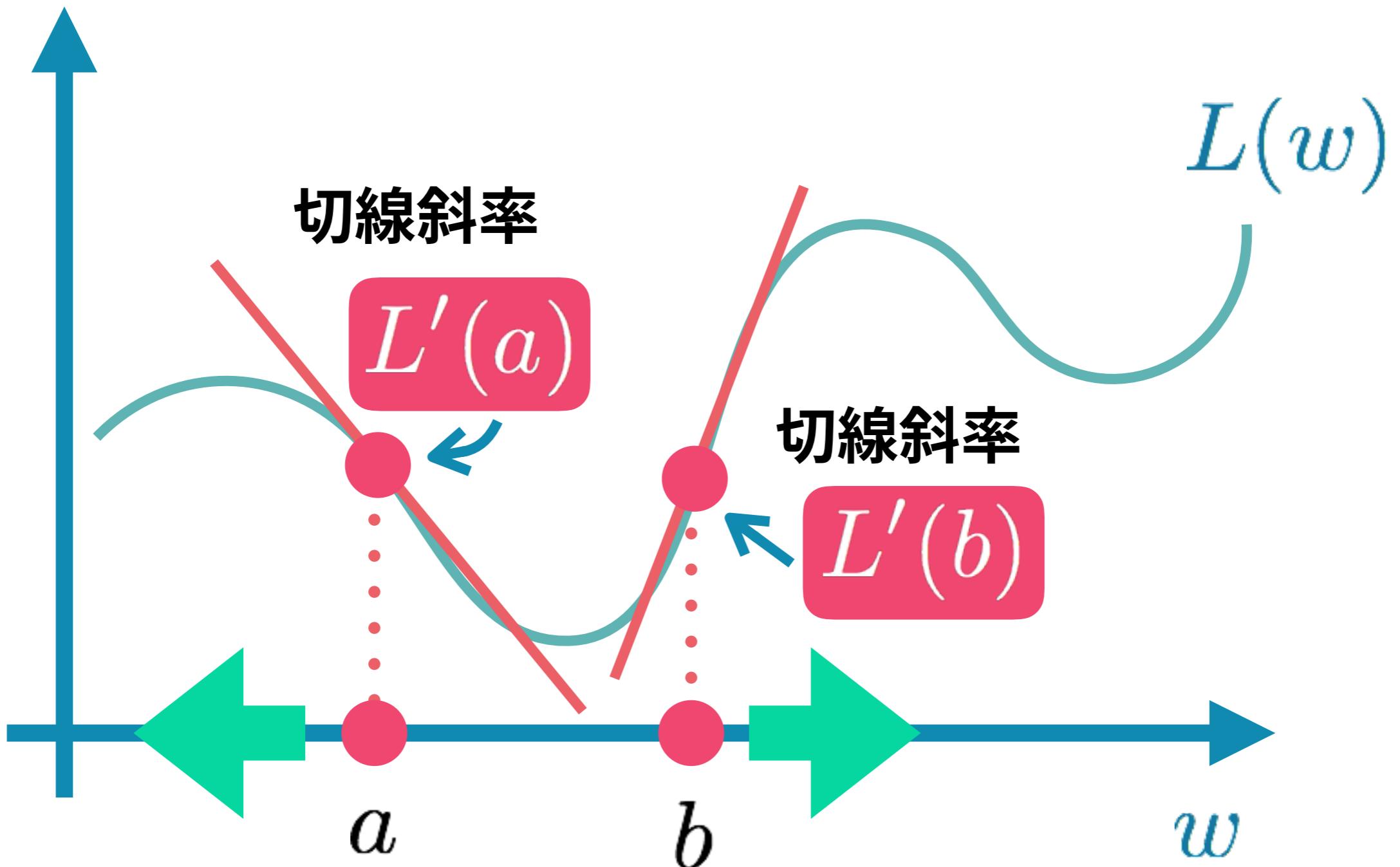
對任意的  $w$ , 我們會寫成:

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

切線斜率函數



# 正負指向（局部）極大



重點

## 往 (局部) 極小值移動

我們想調整  $w$  的值, 往極小值移動, 應該讓新的  $w$  變成:

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

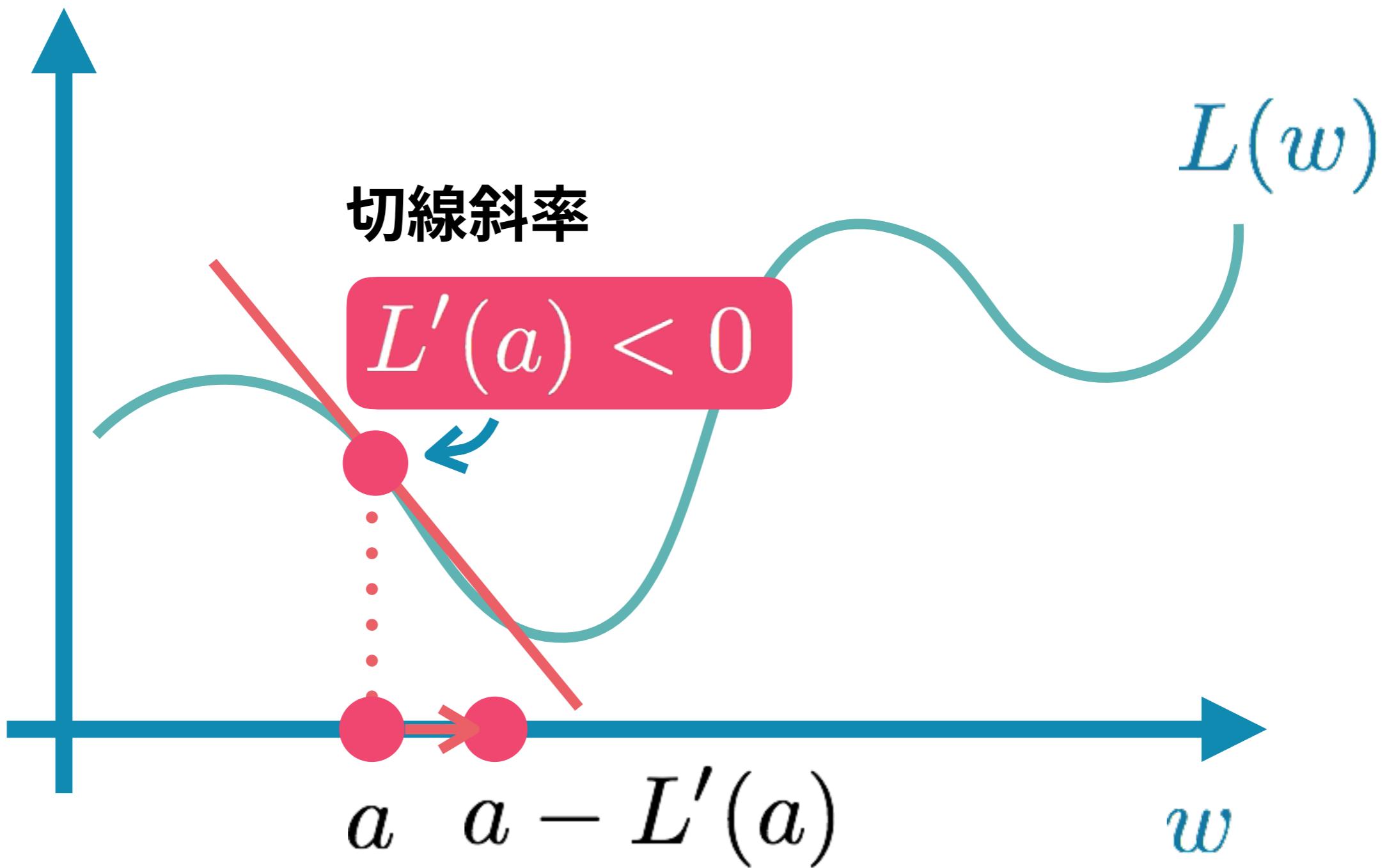
重點

## 往 (局部) 極小值移動

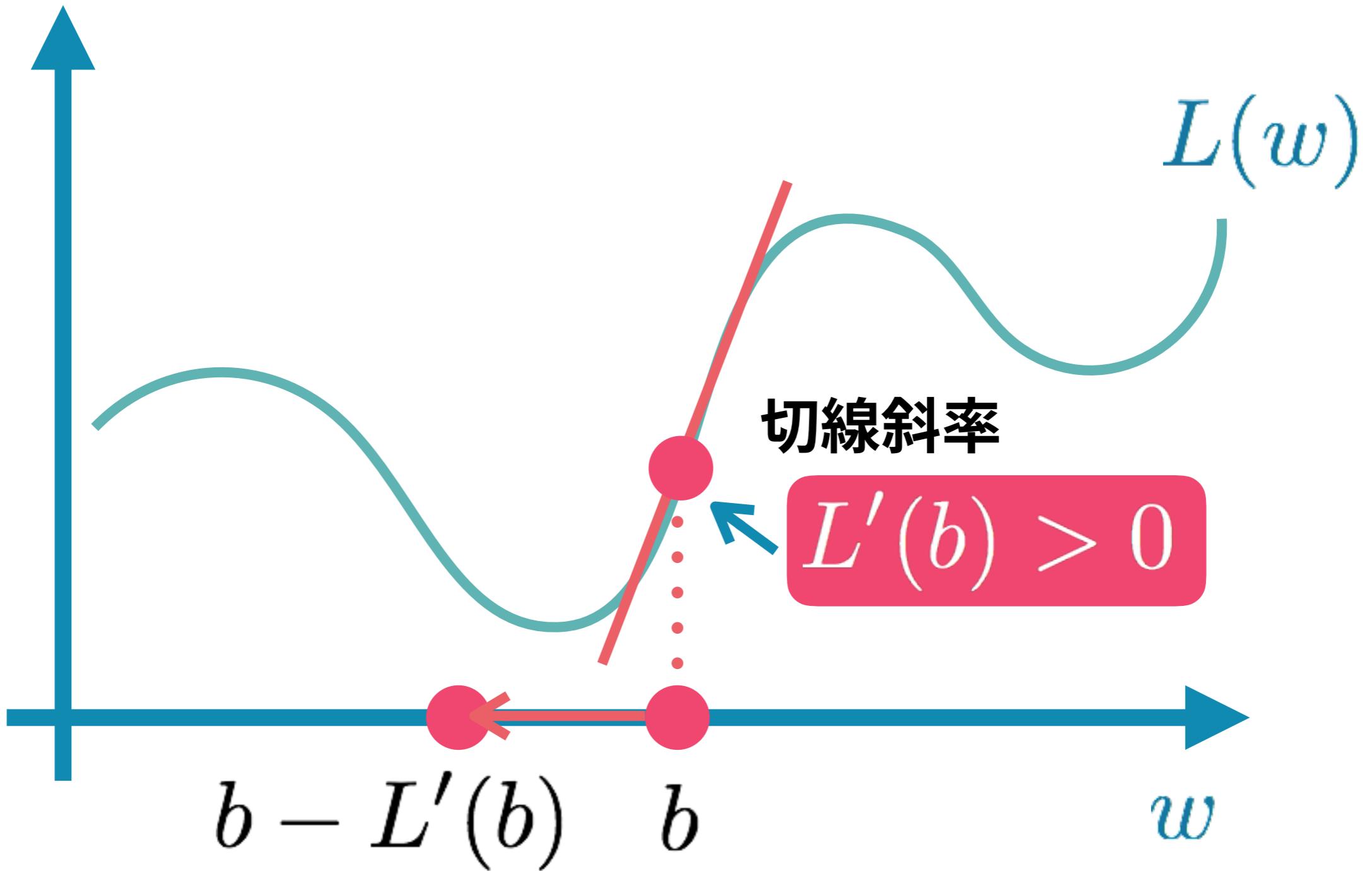
比如現在在  $w=a$ , 要調整為

$$a - L'(a)$$

# 調整示意圖



# 有時會跑過頭!



重點

## 往 (局部) 極小值移動

為了不要一次調太大，我們會乘上一個小小的數，  
叫 Learning Rate:

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

定義

## 偏微分

我們有  $L(w_1, w_2, b_1)$  這三個變數的函數，當我們  
只把  $w_1$  當變數，其他  $w_2, b_1$  當常數的微分。

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

定義

## 偏微分

同理，

$$\frac{\partial L}{\partial w_2} = \frac{dL_{w_2}}{dw_2}$$

$$\frac{\partial L}{\partial b_1} = \frac{dL_{b_1}}{db_1}$$

函數  $L$  的 gradient 就變成：

$$\begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b_1} \end{bmatrix} = \begin{bmatrix} \frac{dL_{w_1}}{dw_1} \\ \frac{dL_{w_2}}{dw_2} \\ \frac{dL_{b_1}}{db_1} \end{bmatrix}$$

符號

## 梯度 (gradient)

Gradient 當然要有很酷的符號：

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b_1} \end{bmatrix}$$

符號

## 梯度 (gradient)

我們調整  $w_1, w_2, b_1$  的方法就變成：

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

大家都知道這叫

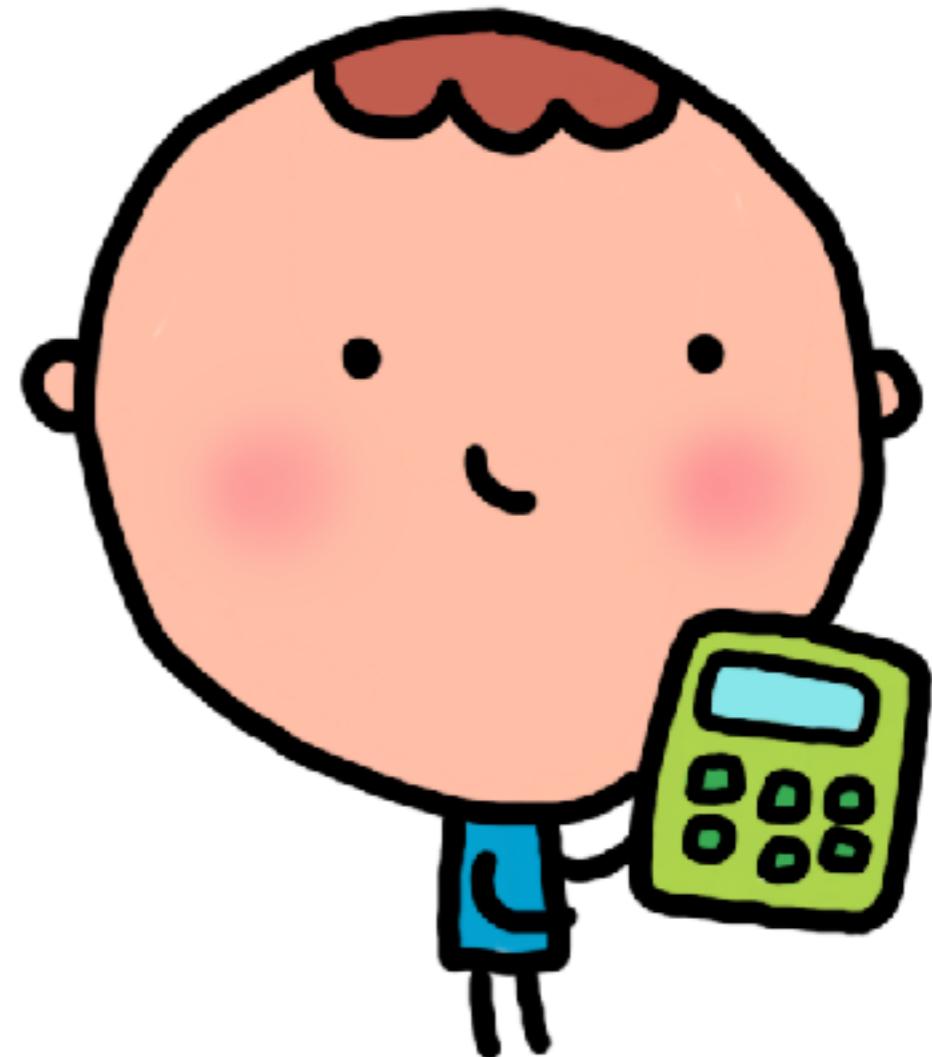
# Gradient Descent

梯度下降

基本上, 把 learning rate 調小, 你  
是可以預期你的學習是會收斂的 --  
在你時間多的狀況下...

16

# 基本統計



假設  $X = \{x_1, x_2, \dots, x_n\}$

相信大家都知道

平均值

$$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

例子

## 平均值的計算

$$X = \{4, 4, 8, 5, 9\}$$

$$\mu = 4 \cdot \frac{1}{5} + 4 \cdot \frac{1}{5} + 8 \cdot \frac{1}{5} + 5 \cdot \frac{1}{5} + 9 \cdot \frac{1}{5}$$

例子

## 平均值的計算

$$X = \{4, 4, 8, 5, 9\}$$

$$\mu = 4 \cdot \frac{2}{5} + 8 \cdot \frac{1}{5} + 5 \cdot \frac{1}{5} + 9 \cdot \frac{1}{5}$$

數字出現的機率

例子

## 平均值的計算

我們可以把  $X$  想成一個隨機變數，所有可能的值為

$$X = \{4, 8, 5, 9\}$$

$$\begin{cases} P(X = 4) = \frac{2}{5} \\ P(X = 8) = P(X = 5) = P(X = 9) = \frac{1}{5} \end{cases}$$

例子

## 平均值的計算

其實我們就是在算這個隨機變數的**期望值**。

$$\mu = \sum_{i=1}^4 x_i P(X = x_i) = E(X)$$

重點

## 變異數的計算

相信大家也還記得變異數是和平均值差多遠 (平方差) 的平均值。

$$\frac{\sum_{i=1}^n (x_i - \mu)^2}{n}$$

重點

## 變異數的計算

再度想成隨機變數，就變成這看來很高級的樣子。

$$\text{Var}(X) = E(X - \mu)^2$$

$$= \sum_i (x_i - \mu)^2 P(X = X_i)$$

重點

## 變異數的重要性質

如果  $X$  和  $Y$  是兩個**獨立**的隨機變數。

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$$

重點

## 變異數的重要性質

如果  $X, Y$  不但獨立, 再加上平均值為 0, 我們有更炫的:

$$\text{Var}(XY) = \text{Var}(X)\text{Var}(Y)$$

## 標準差

$$\sigma = \sqrt{\text{Var}(X)}$$

所以  $X$  的變異數我們常常寫為  $\sigma^2$

# 標準常態分布

平均值是 0, 變異數為 1 的常態分布。如果隨機變數  $X$  是標準常態分布, 我們記為:

$$X \sim N(0, 1)$$

重點

## 用 NumPy

在 NumPy 有個指令會從標準常態分布隨機抽個數字出來。

`randn()`

這在 `np.random` 之下。

重點

# 用 NumPy

取 n 個 samples。

`randn (n)`

重點

## 用 NumPy

平均值改為  $\mu$ , 標準差改為  $\sigma$

**randn( n ) \*  $\sigma$  +  $\mu$**

重點

## 用 NumPy

相反的,  $X$  是一個平均值為  $\mu$ , 標準差為  $\sigma$  的常態分布, 我們要改成標準常態分布。

$$(\text{randn}(n) - \mu) / \sigma$$

17

# 調校之前 比調校還要重要的事



# 問個好問題

要解決的問題，當有不同的問法。通常第一次的問法都要調整，還有一定要考慮是不能有足夠的資料。



**建構你的  
神經網路**

18

初始化



其實基本上，weights 亂亂選就可以。

雖然我們可以說得好像有點學問。

重點

## 權重等參數選取原則

- 不要全部設成 0 (原因用腳想就知道)
- 取小一點的值



重點

# 權重等參數選取原則

activation

initializer

**sigmoid, tanh**

**Xavier**

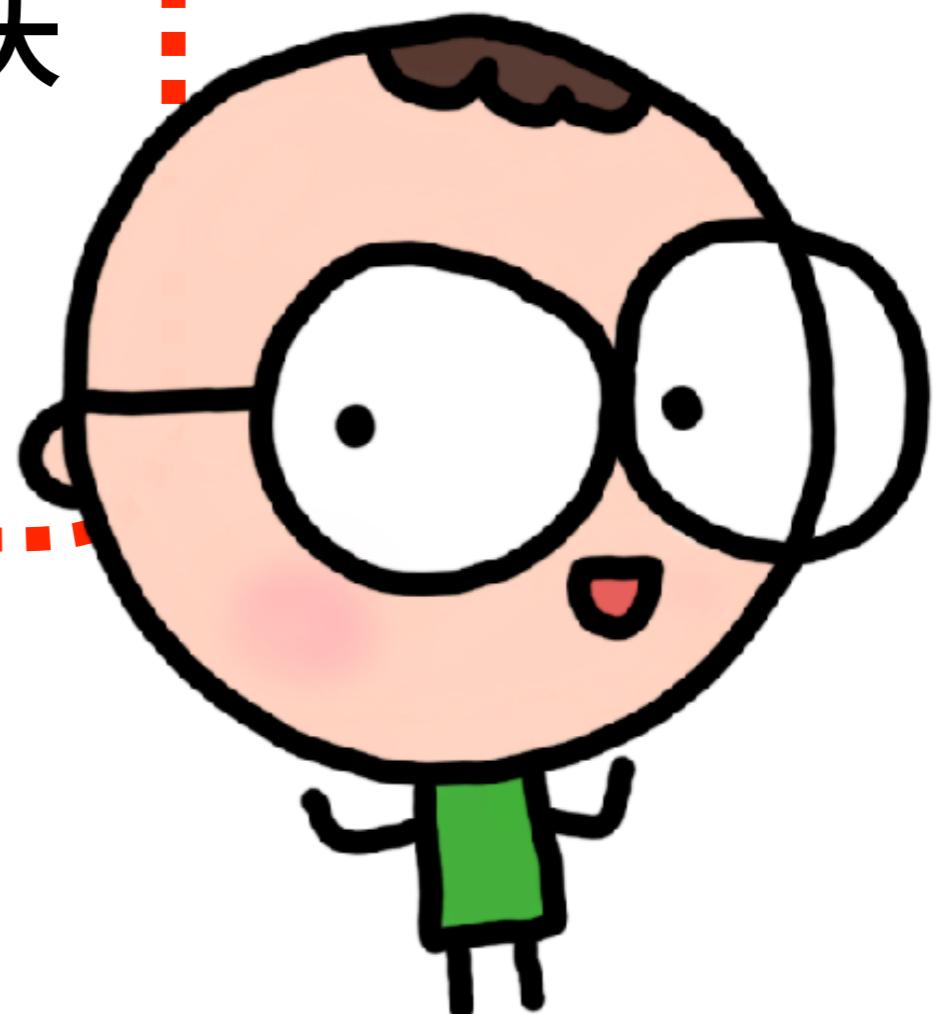
**ReLU**

**He**

**SELU**

**LeCun**

這看來好有學問的，其實只是說我們每層權重的變異數不要一直加大（不致於產生某個權重一開始就太大）。



假設我們神經網路的某一層有  $n=n_{\text{in}}$  個輸入, 權重我

們記為:

$$W = \{W_1, W_2, \dots, W_n\}$$

輸入我們記為

$$X = \{X_1, X_2, \dots, X_n\}$$

這層接收到的輸入就是

$$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

**如果該獨立的都獨立, 平均值又都是 0, 我們有:**

$$\text{Var}(Y) = \sum_{i=1}^n \text{Var}(W_i) \text{Var}(X_i)$$

**如果每個  $W_i, X_i$  基本上一樣的話, 例如**

$$\text{Var } \overline{W} = \text{Var } W_i$$

**如果該獨立的都獨立, 平均值又都是 0, 我們有:**

$$\text{Var}(Y) = \sum_{i=1}^n \text{Var}(W_i) \text{Var}(X_i)$$

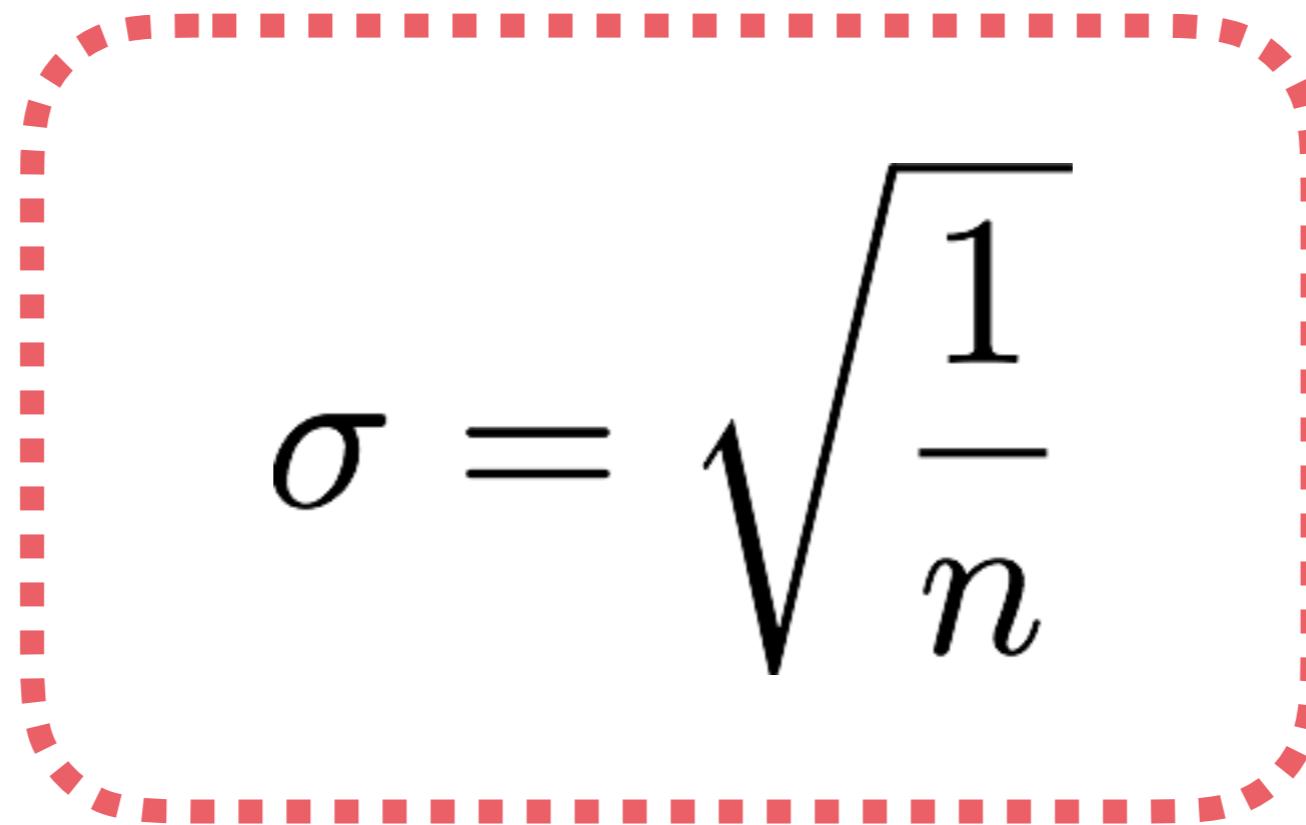
**如果每個  $W_i, X_i$  基本上一樣的話, 例如**

$$\text{Var } \overline{W} = \text{Var } W_i$$

**也就是  $n$  越大, 變異數越大...**

$$\text{Var}(Y) = n \text{Var}(\bar{W}) \text{Var}(\bar{X})$$

所以我們可以由平均值 0, 變異數為  $\frac{1}{n}$  的分布中取值


$$\sigma = \sqrt{\frac{1}{n}}$$

## Xavier Initialization

<http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>

所以我們可以由平均值 0, 變異數為  $\frac{1}{n}$  的分布中取值

$$\sigma = \sqrt{\frac{1}{n}}$$

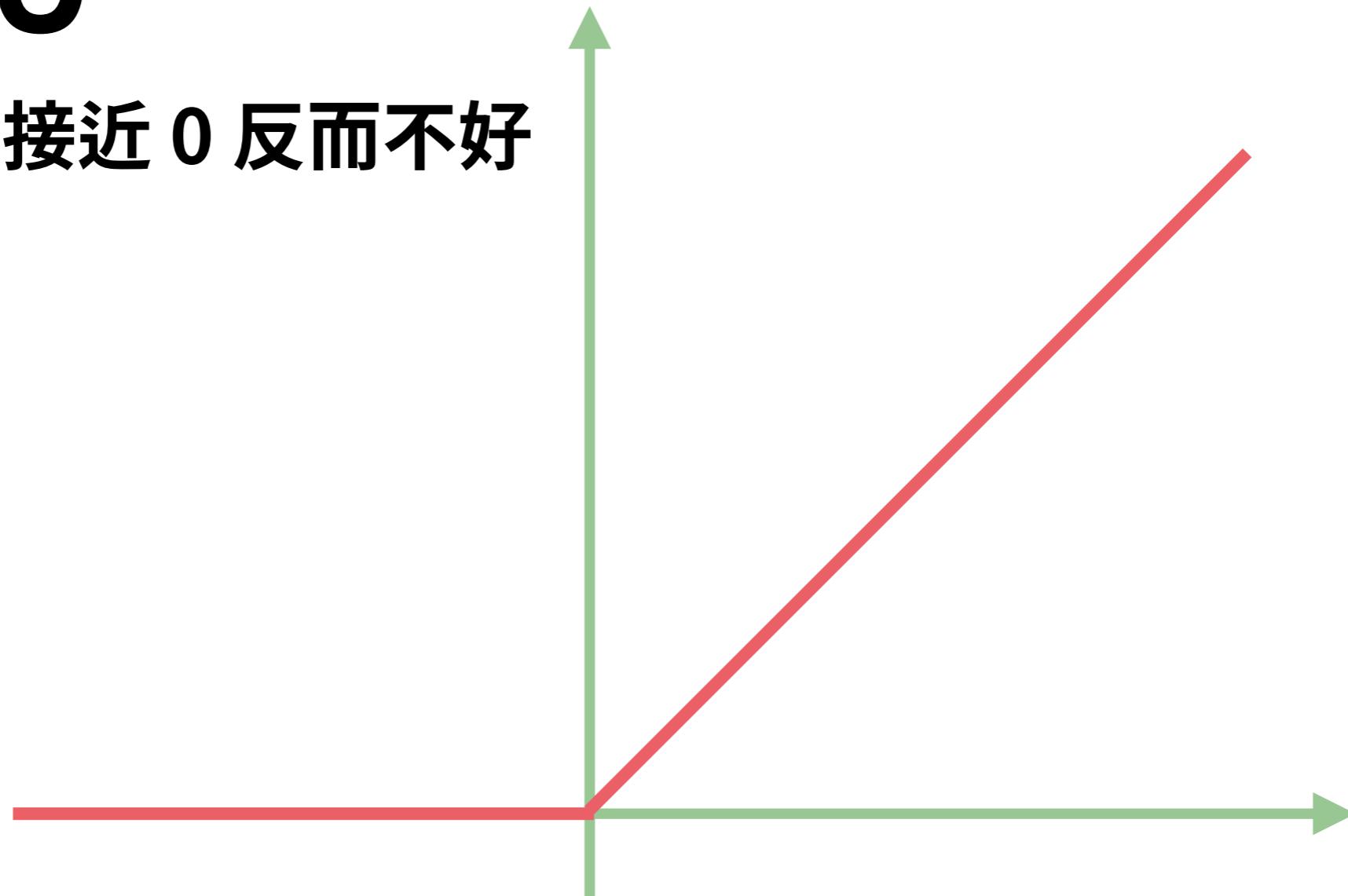


Xavier Initialization

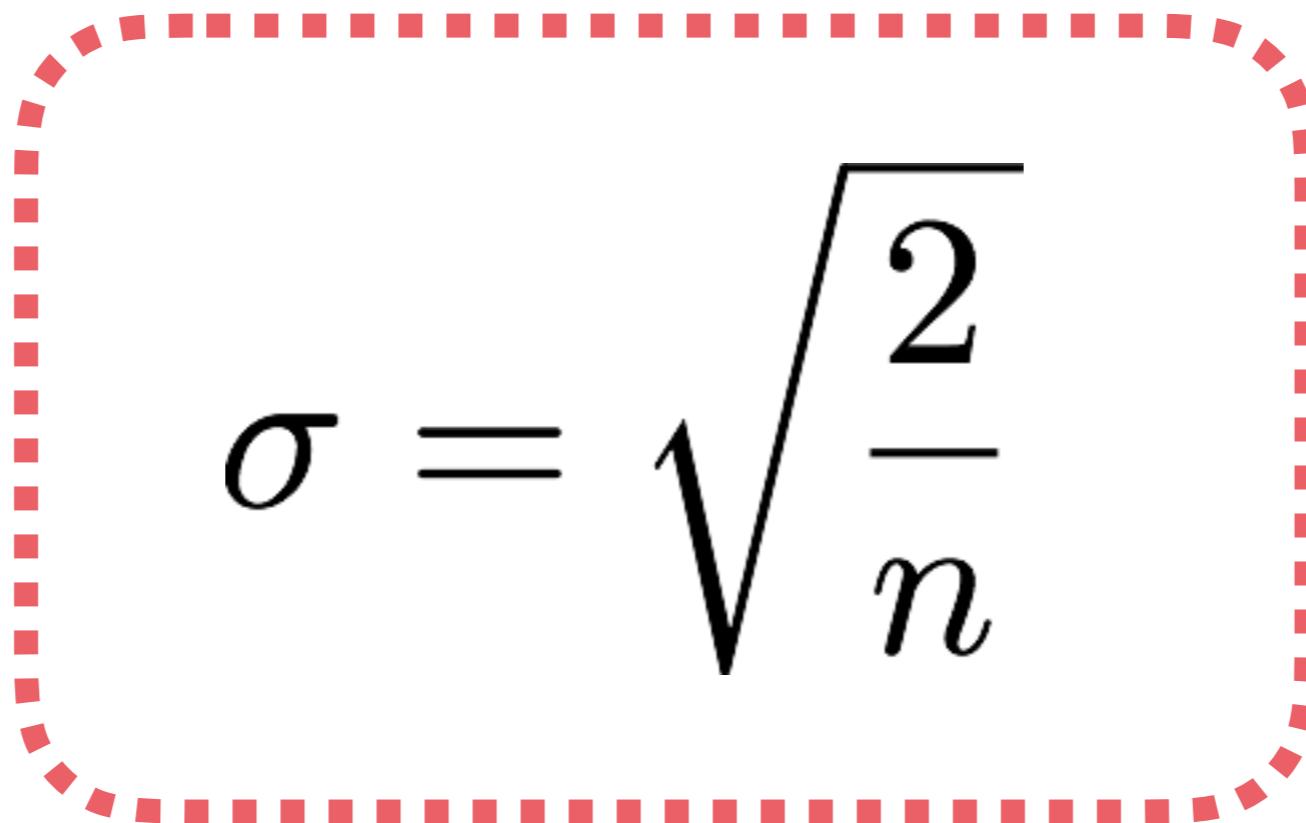
$$n = n_{\text{in}} + n_{\text{out}}$$

# ReLU

ReLU 太接近 0 反而不好



結果大一點就好...

A red dashed circle surrounds a mathematical equation. The equation is  $\sigma = \sqrt{\frac{2}{n}}$ .
$$\sigma = \sqrt{\frac{2}{n}}$$

He Initialization

<https://arxiv.org/abs/1502.01852>

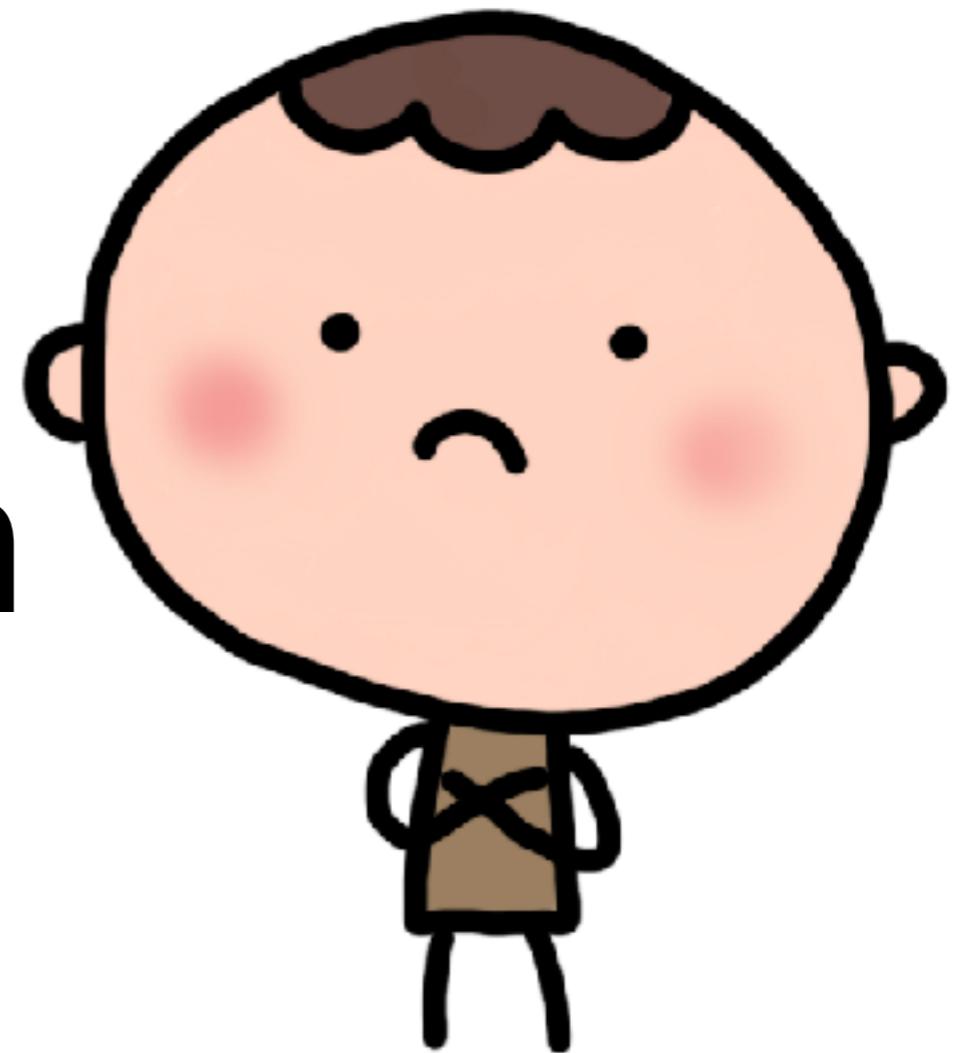
# **SELU**

<https://arxiv.org/abs/1706.02515>

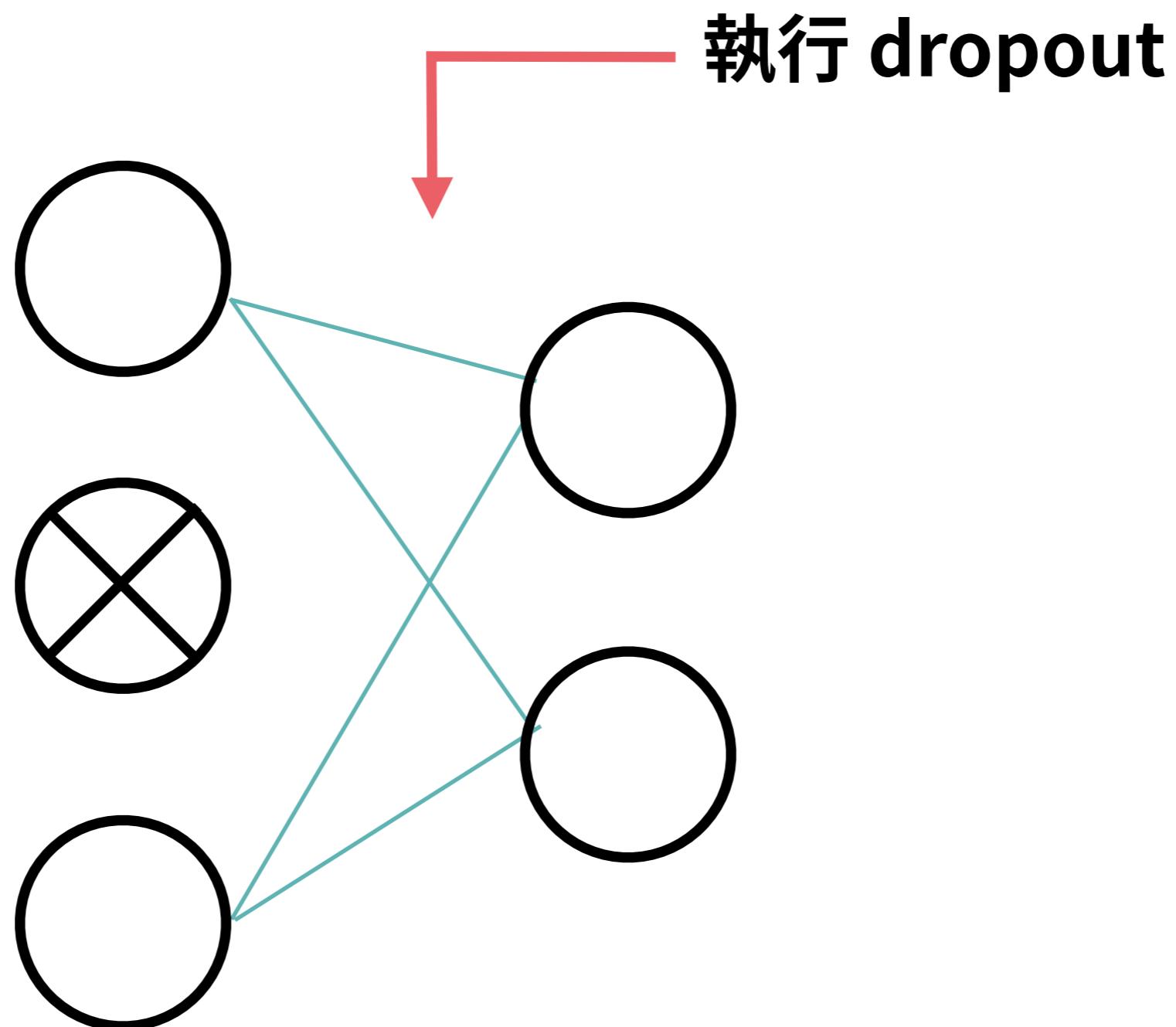
**LeCun Initialization  
加設上下限在一定範圍。**

19

# Regularization



# Dropout



# L2 Regularization

loss function +  $\frac{\lambda}{2n} \|w\|_2^2$

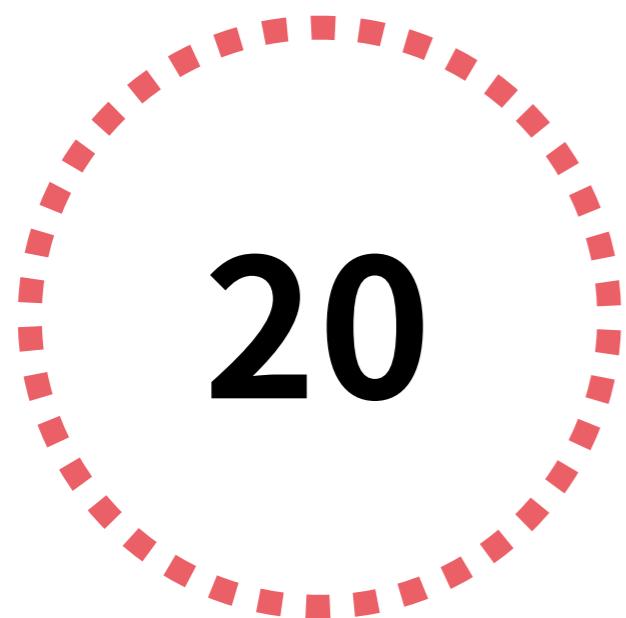
重點

## 設 regularizer

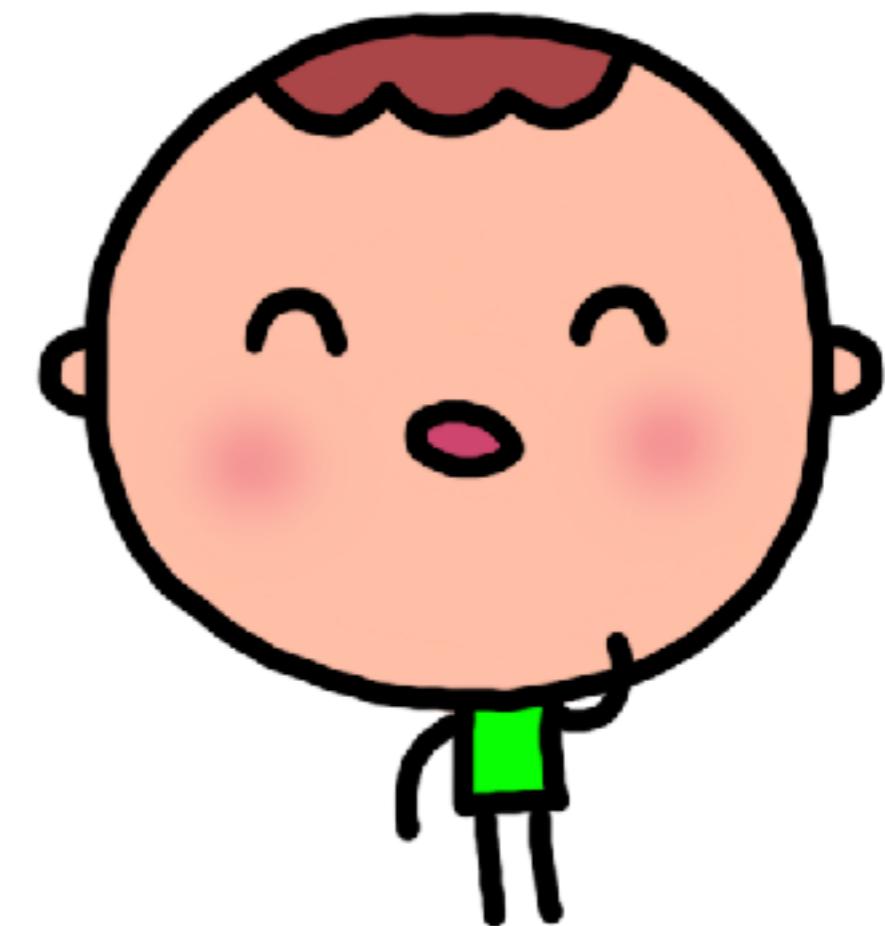
在 Keras 可在每層中設 regularizer, 例如我們先 import 了 keras.regularizer 中的 12, 那在某層可加入以下參數。

w\_regularizer=12(0.01)

**通常 dropout, L2 regularization 選一個就好。**



# Optimizer



為了簡化我們的符號，我們設定：

$\theta_t$  目前參數值所成向量

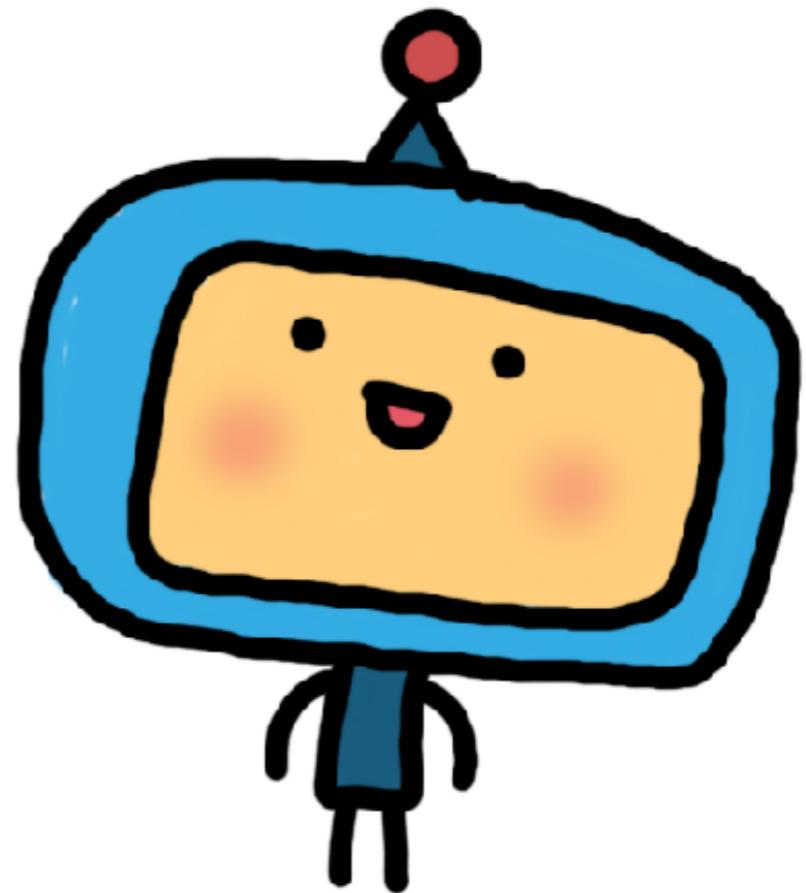
$$g_t = \nabla_{\theta} L(\theta_t)$$

# 標準 Gradient Descent

$$\theta_{t+1} = \theta_t - \eta g_t$$

# 兩個改善方向

- momentum: 方向的穩定和加速器
- 可變速的 learning rate



概念

# Momentum

想像你在 loss function 高山上某一點，負的梯度指  
的是那一點你看到最陡的下坡方向。

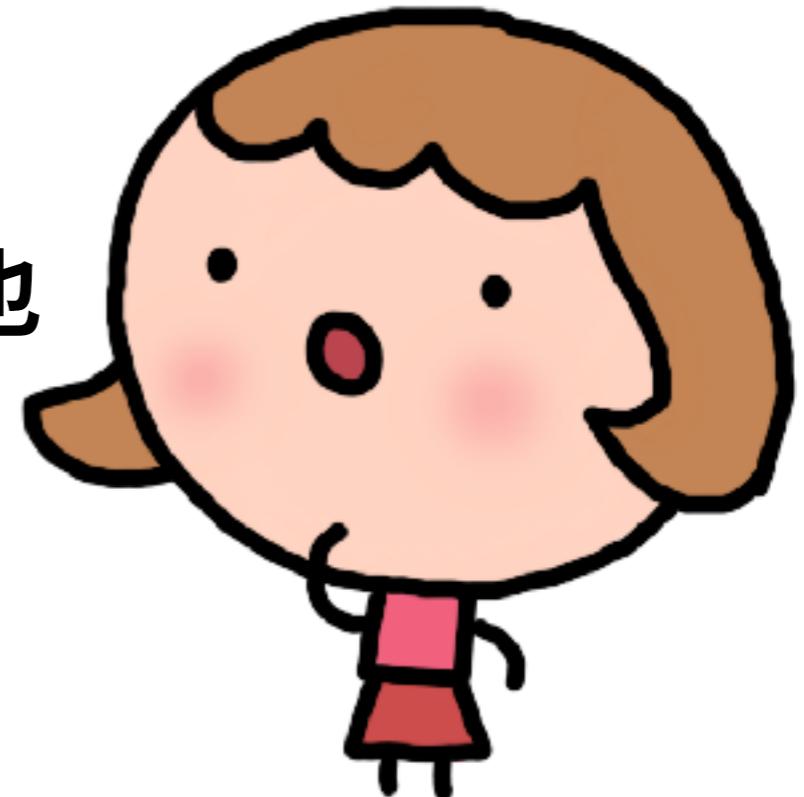


概念

# Momentum

問題是你順著那個方向到了另一點，你可能會發現那個方向又變了！於是你就向另一個方向...

但其實你好好走原來方向，也許一路也就下山了！



概念

## Momentum

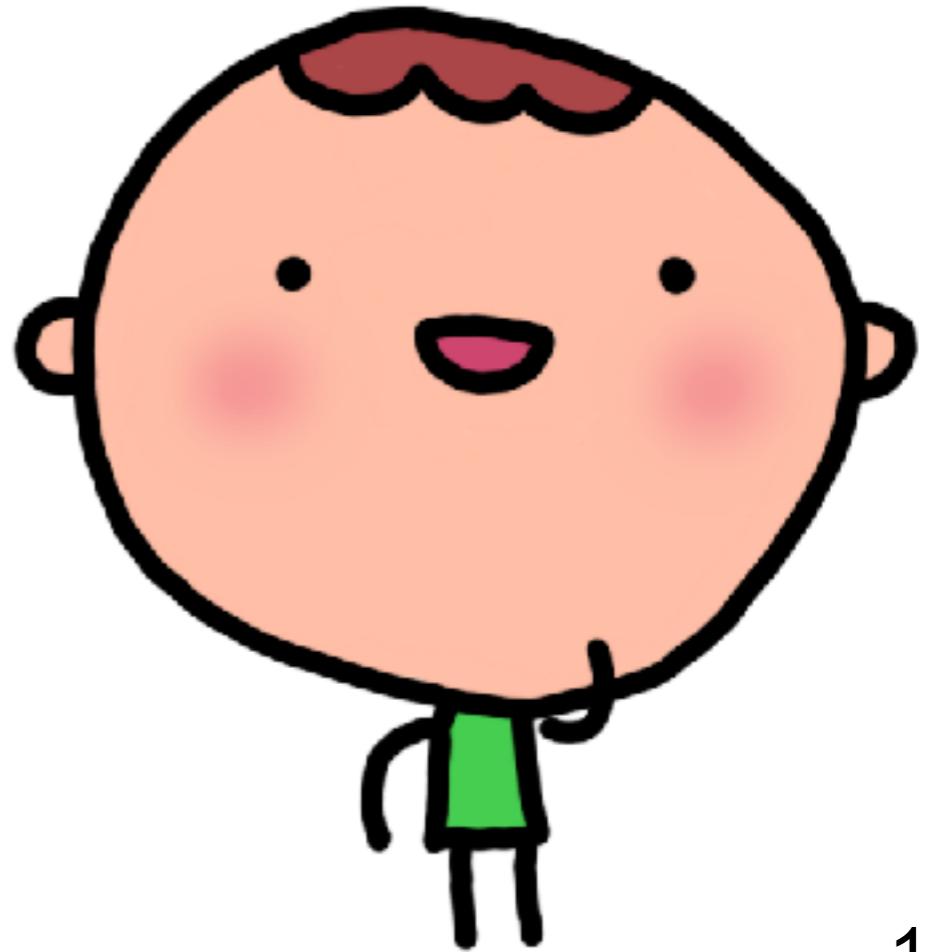
$$v_t = \gamma v_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - v_t$$

如果  $\gamma=0$ , 我們用的就是原本的 gradient descent。

# Learning Rate 變速器

一個很自然的想法: learning rate 在接近目標時可能要小一點。於是有了各種「減速法」。



概念

## Momentum

---

$$v_t = \gamma v_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - v_t$$

重點

## Adam

不想尋找哪個方法最好，通常可以交給 Adam。這是兼具 momentum 和減速的方法。

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

重點

## Adam

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

重點

## Adam

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

原始論文設定：

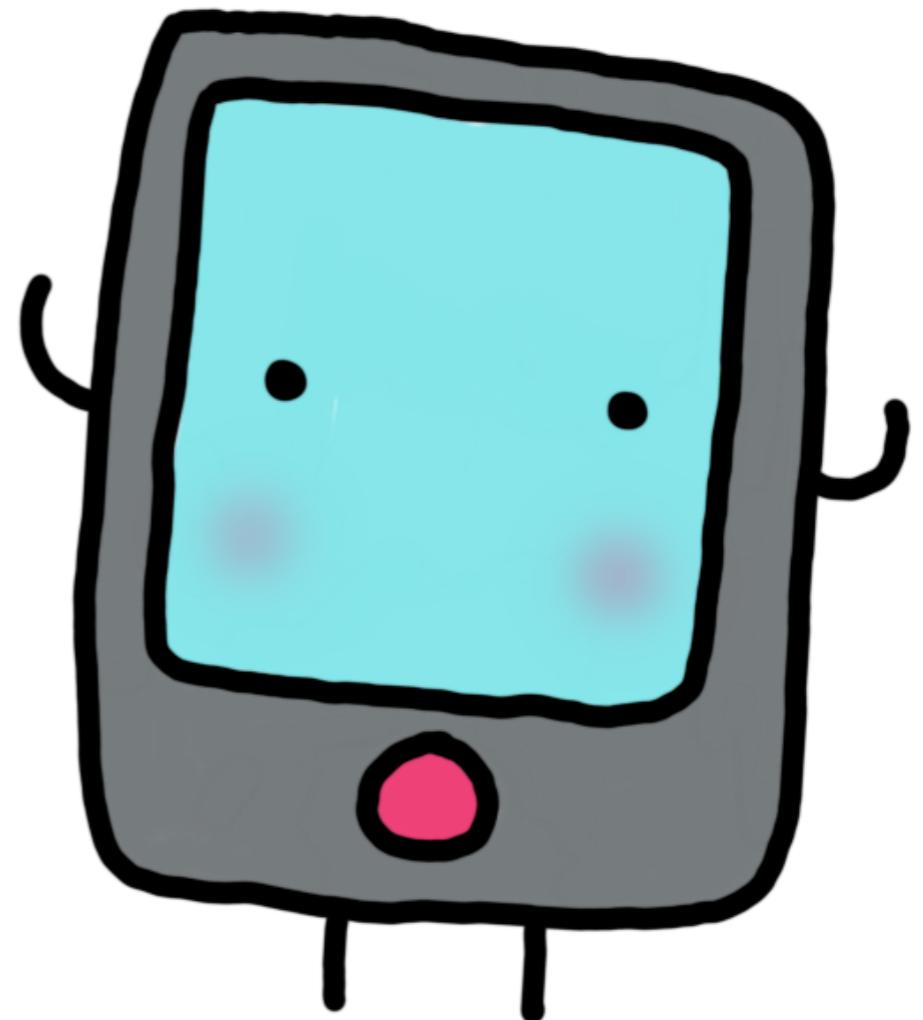
$$\beta_1 = 0.9$$

$$\beta_2 = 0.999$$

$$\epsilon = 10^{-8}$$

21

# Batch Normalization



**對輸入做某種 normalization 是重要的。**

**原因1**

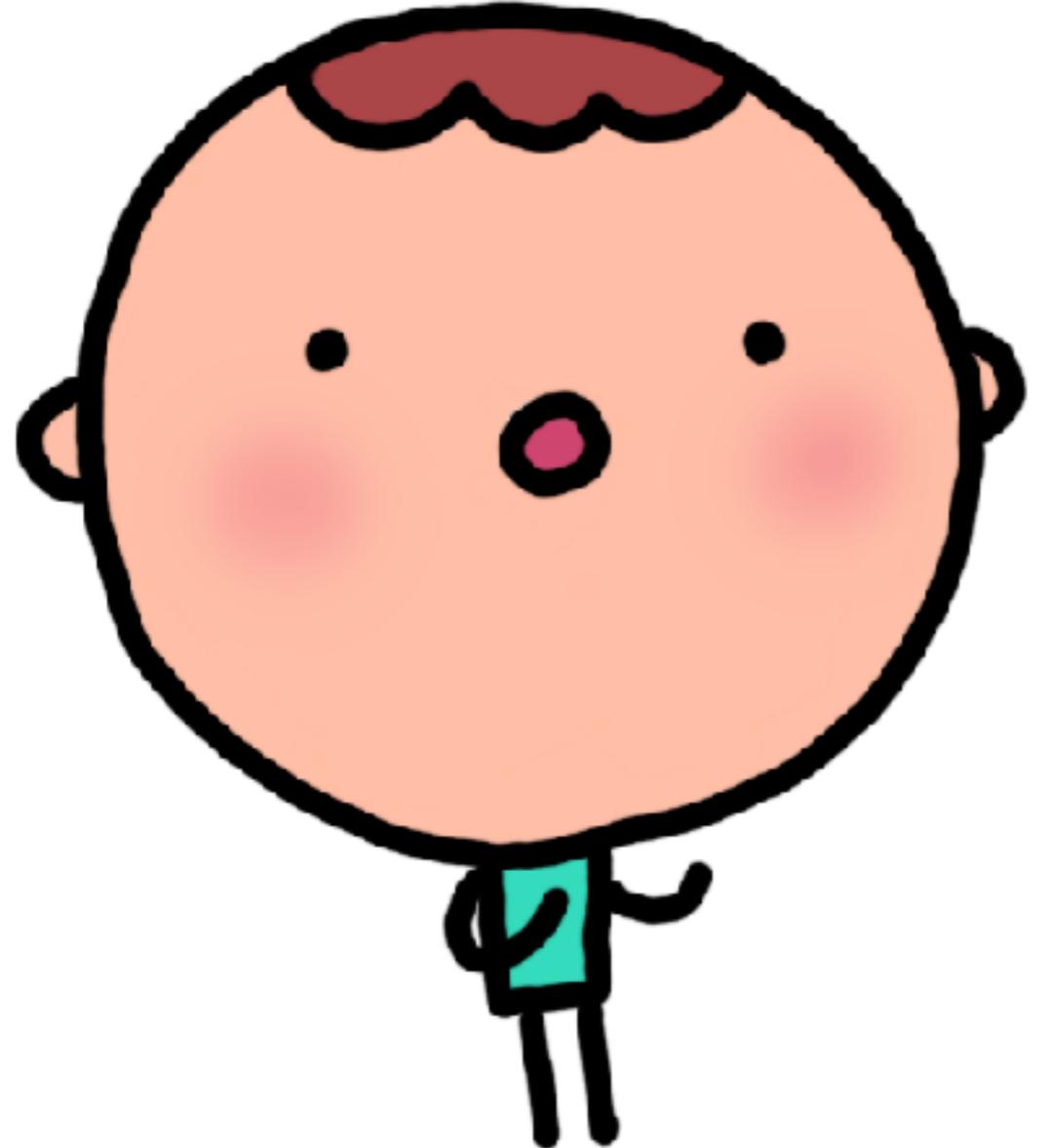
**不會有某個 feature 太過主導。**

原因2

至少在理論上分析是相對容易的。

$\text{Var}(X_i)$

最好是 1



那可以在中間的某層輸入也做  
某種 normalization 嗎？

答案是肯定的，就是：

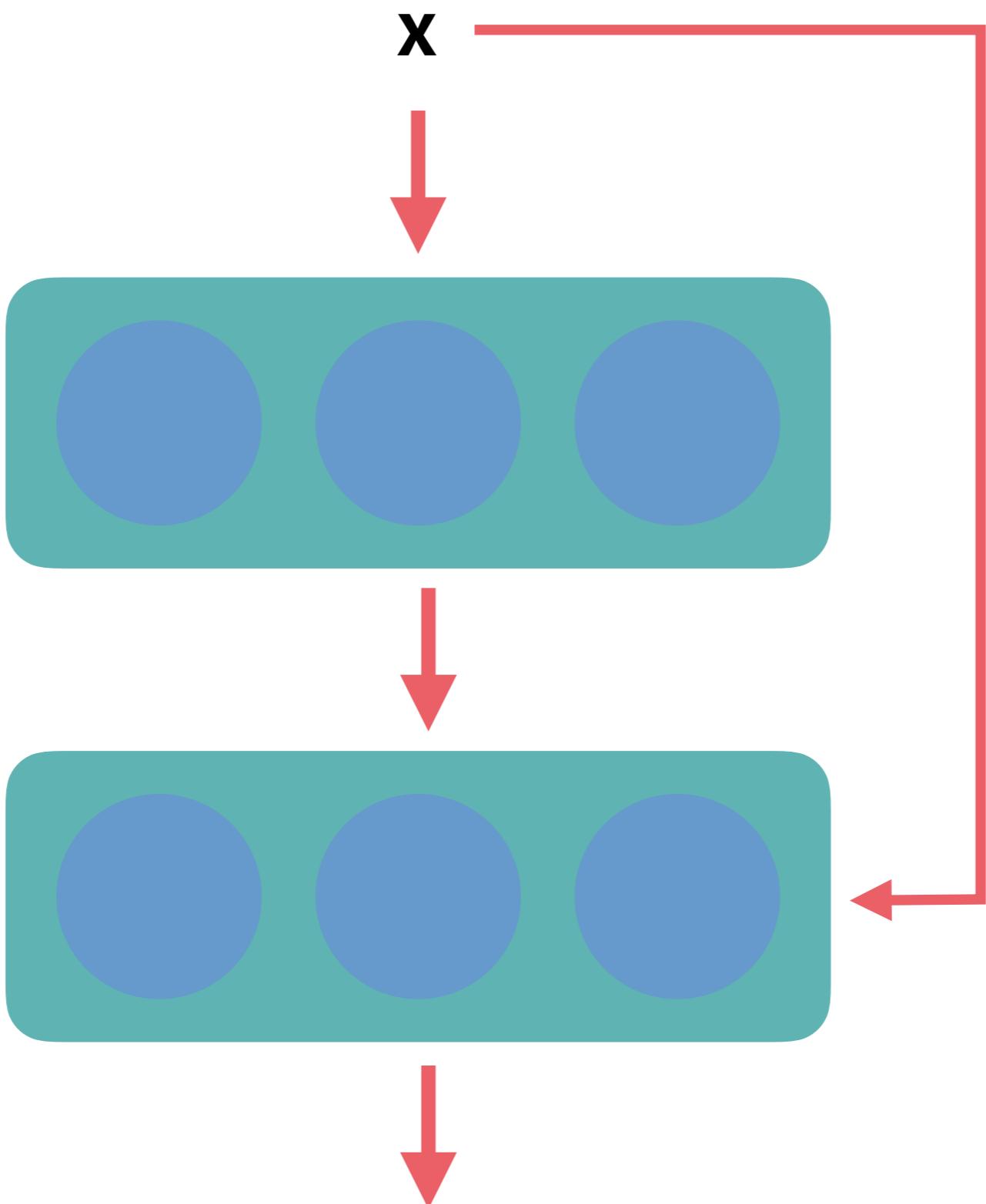
Batch Normalization

22

# ResNet 深度結構



**我們真的可以做深度學習。**



23

SELU  
什麼都來



$$\text{selu}(x) = \lambda \cdot \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

最炫的是參數其實是常數：

```
α = 1.6732632423543772848170429916717  
λ = 1.0507009873554804934193349852946
```

重點

## 使用注意事項

---

- 起始用 LeCun Normal (權重平均 0, 變異數 1)
- 輸入資料也要平均 0, 變異數 1

**可以做很深的深度學習!**