深度學習

第 8 章 循環神經網路
(Recurrent Neural Networks)

講師：紀俊男

本章大綱

- 簡單 RNN 簡介與實例
- 長短期記憶（LSTM）簡介與實例
- 閘道循環單元（GRU）簡介與實例

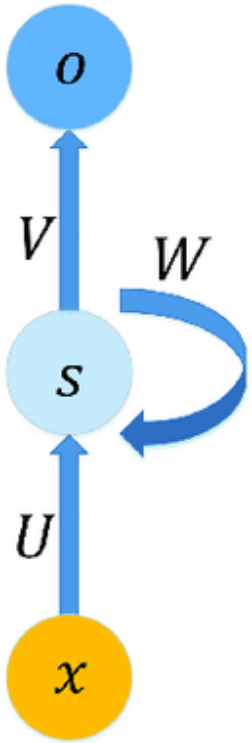
簡單 RNN 簡介

Introduction to Vanilla RNN

RNN 全名

Recurrent Neural Networks

（循環神經網路、遞迴神經網路）



猜猜看，大街上的人們穿什麼？

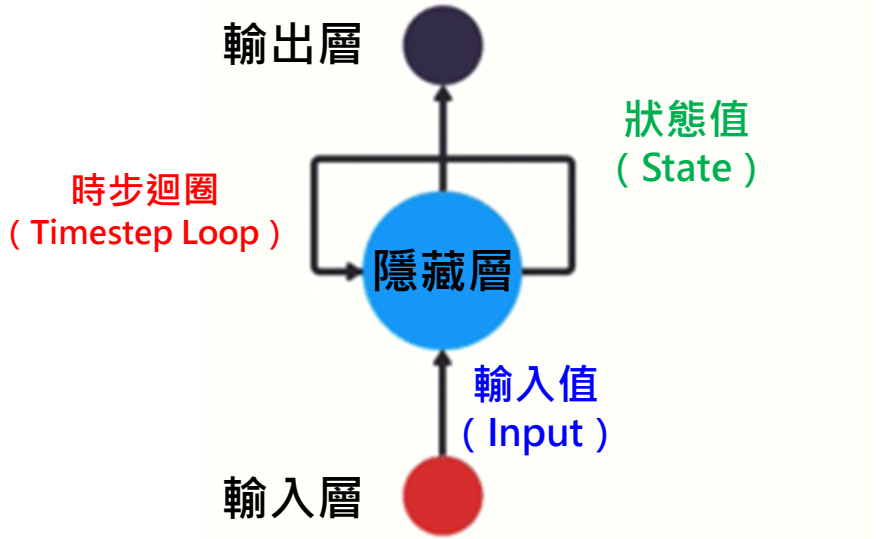
得對前一個輸入「有記憶」，
才能正確地推論！

今天氣溫 37 度

- 穿短袖 95%
- 穿外套 5%

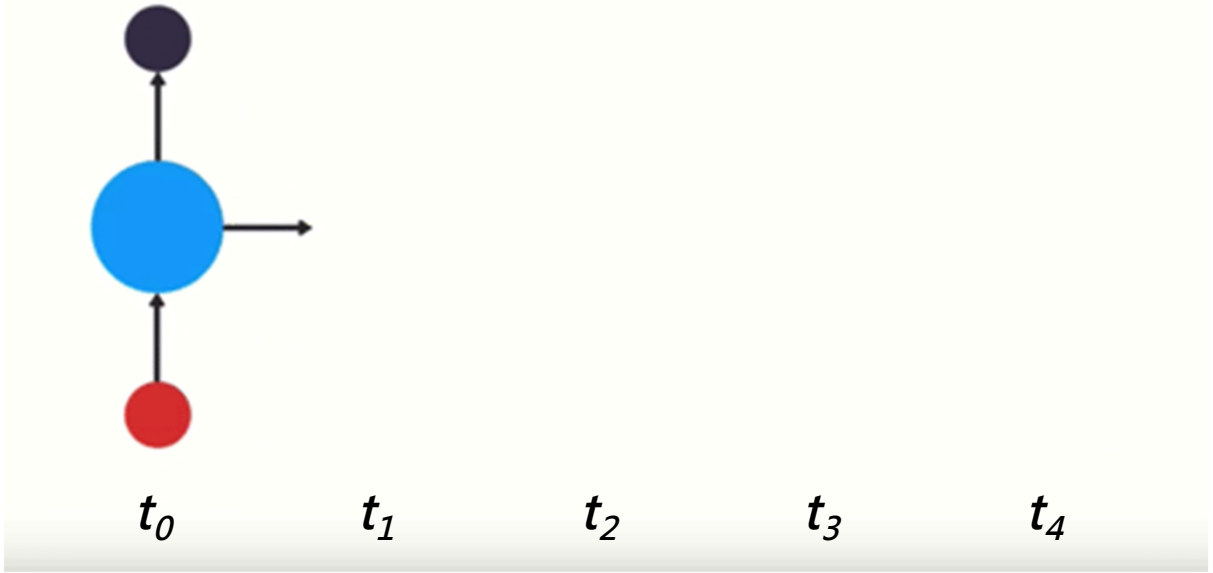
如何讓神經網路「記住」先前的輸入值？

將「上一個時間點」的隱藏層輸出
接回來，成為新的輸入值之一



依照時間軸
展開

比較常見的表示法

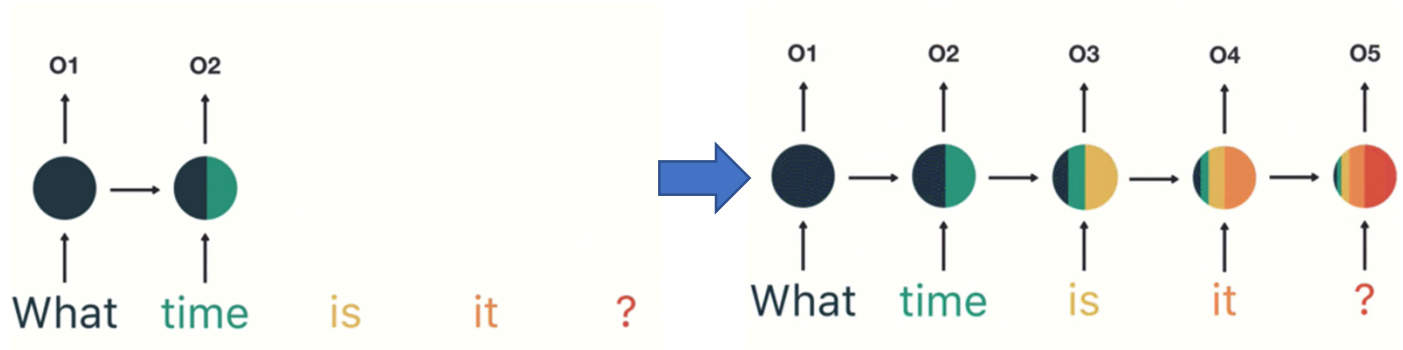


數學表示式： $S_t = f(S_{t-1}, \ x_t)$

(S_{t-1} 稱為 S_t 的「短期記憶」)

一個 RNN 運作範例

- 用「輸入文字」，預測「使用者意圖」



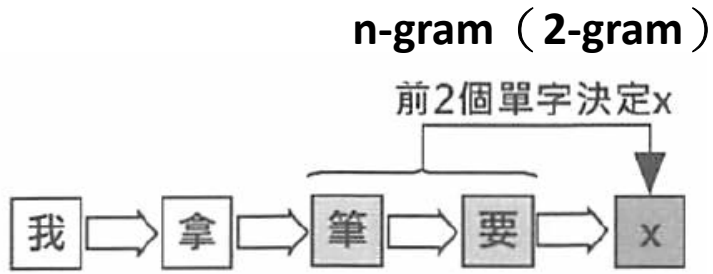
RNN 擅長處理的問題

- 序列問題（Series Problems）：資料排列順序是「有意義」的問題
  - 空間序列：如「照片」（是的！RNN 也可以處理照片！雖然沒 CNN 那麼厲害！）
  - 時間序列：如「語言」、「音樂」、「股價」...

非序列資料

序列資料

空間序列

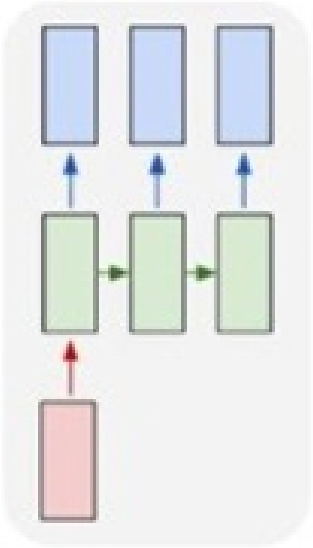


“我拿筆要” → 「寫字」

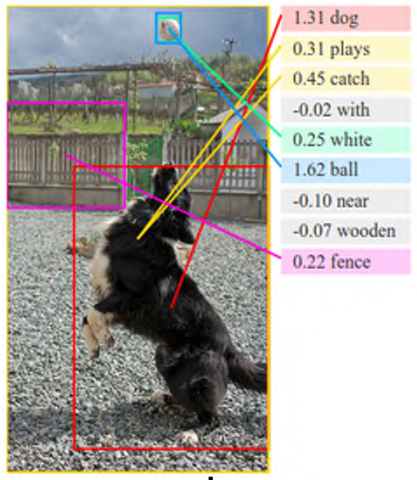
時間序列

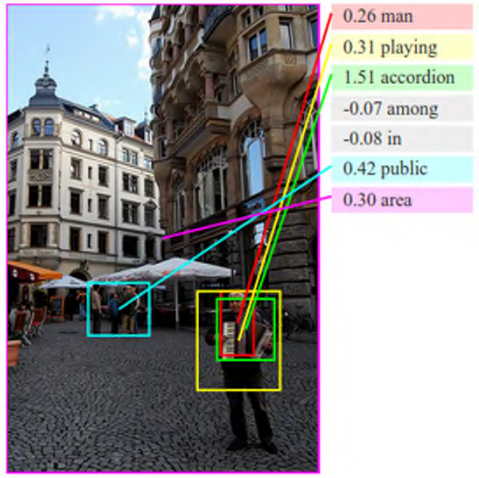
序列問題有幾種？

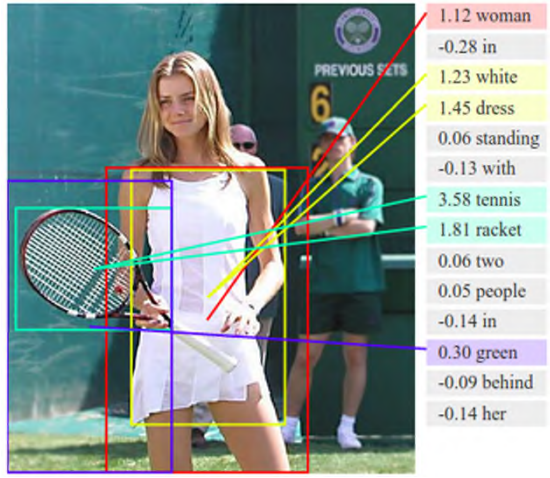
- 一對多：一個輸入，多個輸出



圖片標題生成（Caption Generation）
（Chen & Zitnick, 2014）





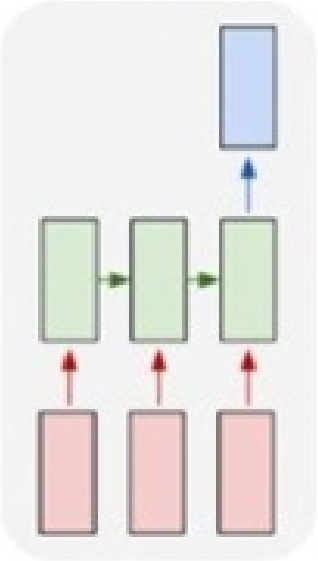


→ dog, play/catch, white, ball, fence

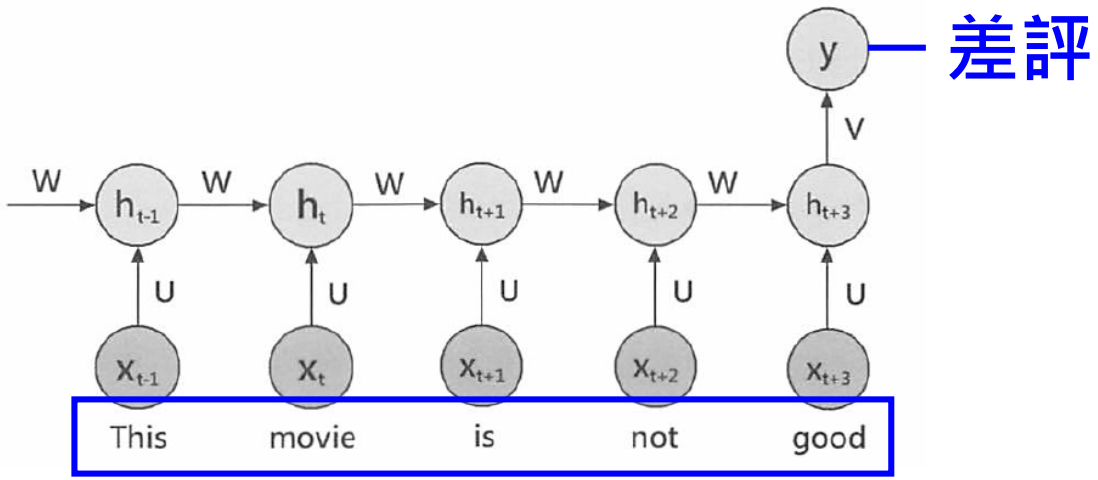
→ A dog tried to catch a white ball inside the fence

序列問題有幾種？

- 多對一：多個輸入，一個輸出



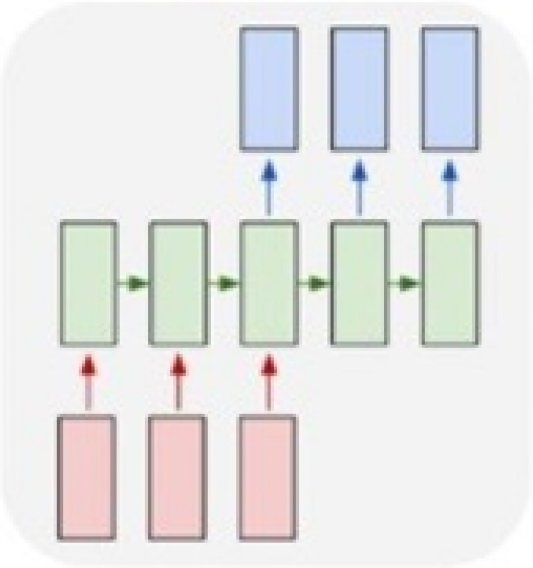
情感分析（Sentiment Analysis）
（Tang et al. 2015）



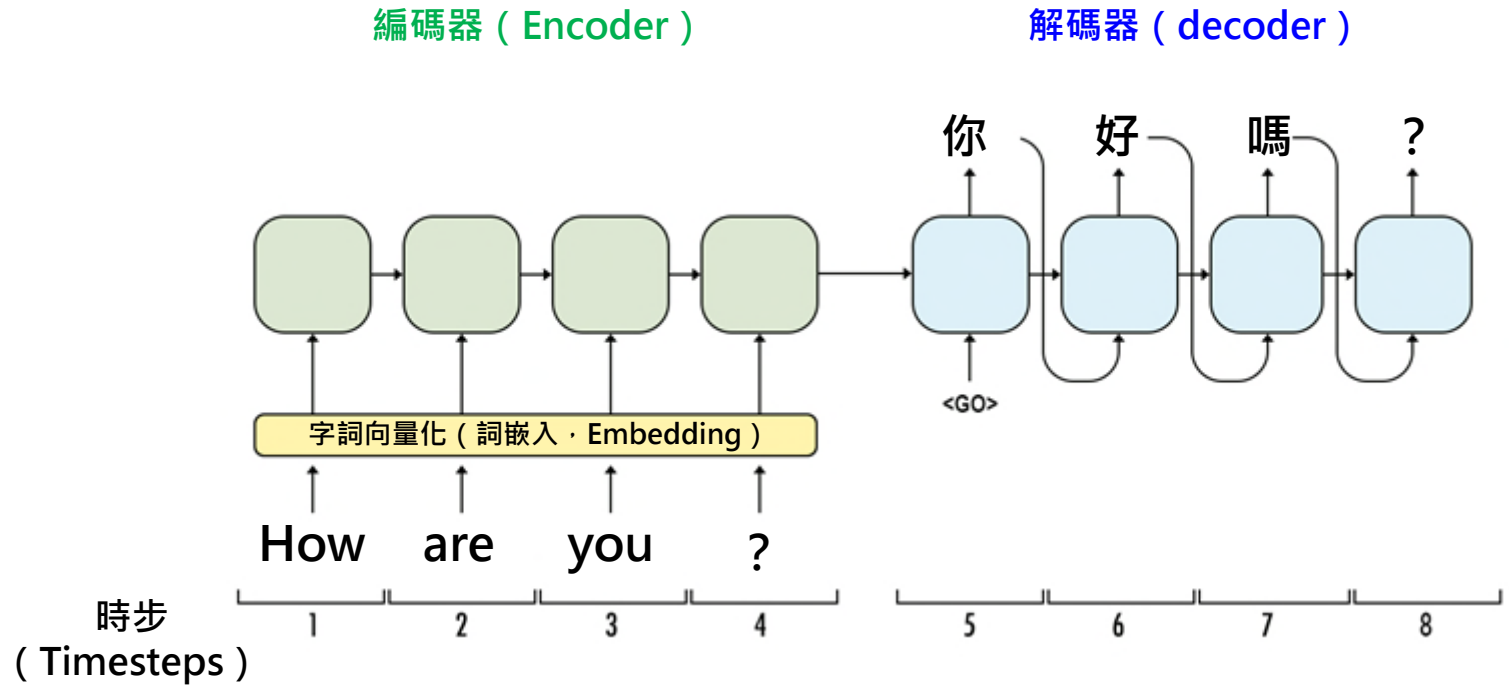
其它應用：股市分析 – 輸入前 30 天資料 → 輸出今日漲跌之預測

序列問題有幾種？

- 多對多（非即時）：多個輸入完畢後，產生多個輸出

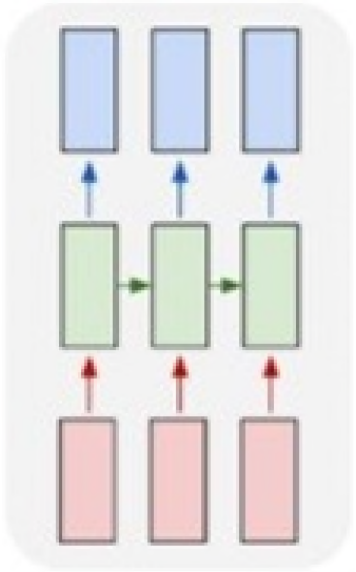


機器翻譯

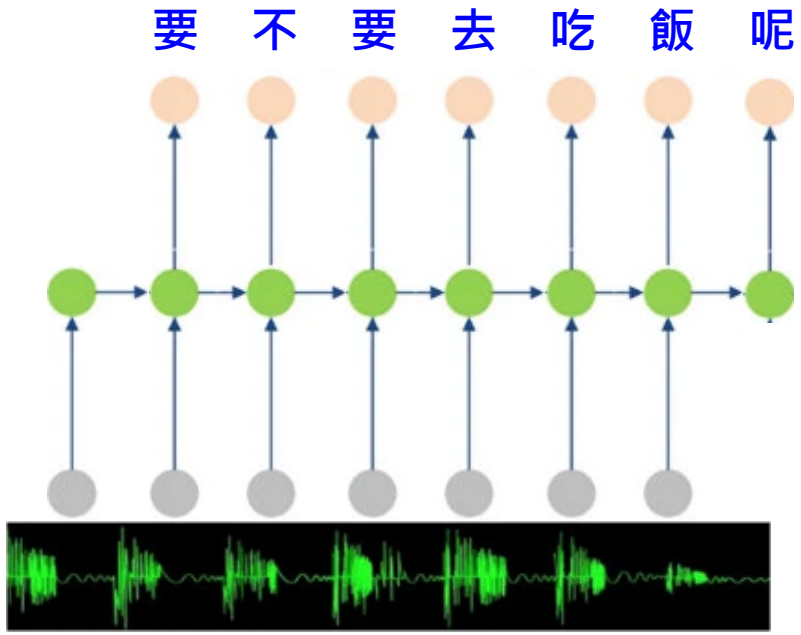


序列問題有幾種？

- 多對多（即時）：一邊產生多個輸入，同時產生多個輸出



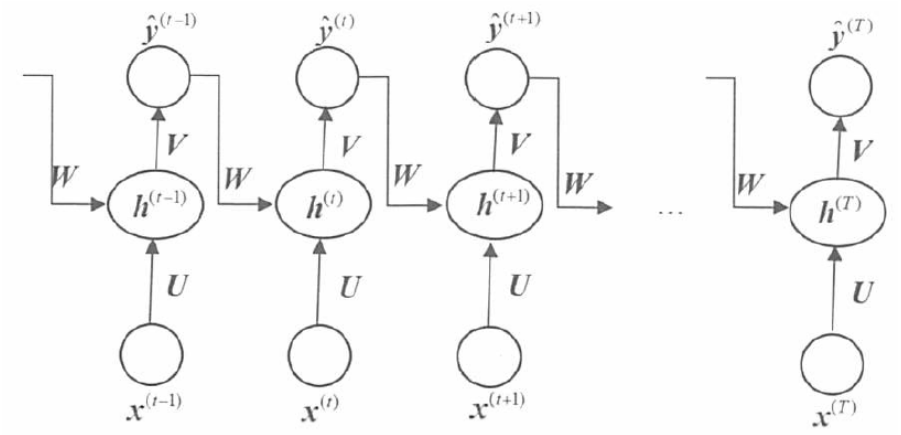
語音辨識



為何不是把「輸出」接入下一個時步呢？

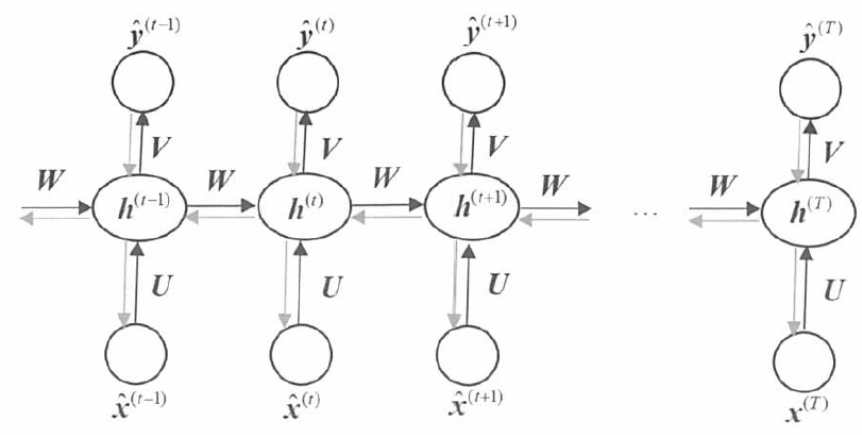
Jordan 神經網路（1986）

將本期輸出層資訊，連接到下一層



Elman 神經網路（1990）

將本期隱藏層資訊，連接到下一層



- Elman 神經網路的優點：比較容易做出「一對多」、「多對一」、「多對多」的循環神經網路。
- Jordan / Elman 神經網路：又稱「簡單循環神經網路（Vanilla RNN）」

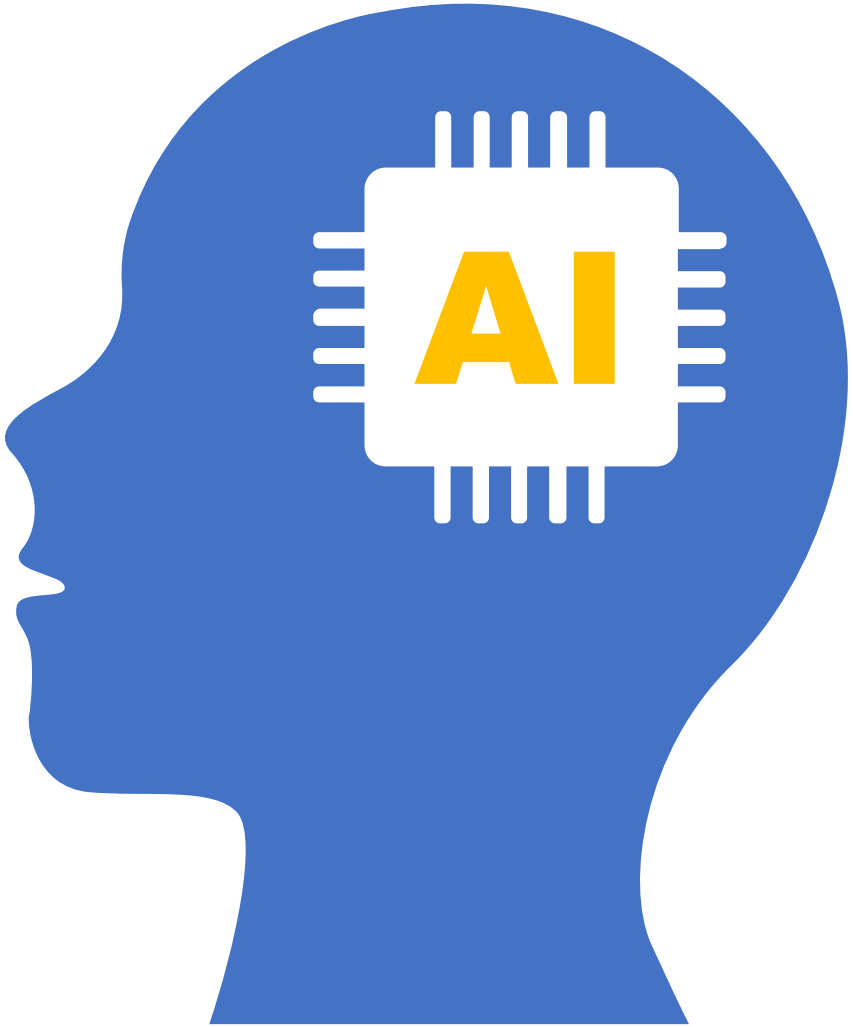
簡單 RNN 範例

The Example of Vanilla RNN

範例完整原始碼：
https://lurl.cc/ccAvB0

範例：預測今天吃什麼

- 小明記錄了一整個月每天中午吃什麼
- 你有辦法用過往記錄，預測他今天吃什麼嗎？



下載與瀏覽資料集

- 如果想看一下資料集內容，可依講師指示，下載並瀏覽資料集

Meals.csv

Date	Meal
2020/1/1	煎餃
2020/1/2	自助餐
2020/1/3	大魯麵
2020/1/4	自助餐
2020/1/5	黃金炒飯
2020/1/6	大魯麵
2020/1/7	煎餃
2020/1/8	自助餐
2020/1/9	大魯麵
2020/1/10	自助餐
2020/1/11	黃金炒飯
2020/1/12	大魯麵
2020/1/13	煎餃
⋮	
2020/1/29	黃金炒飯
2020/1/30	大魯麵
2020/1/31	煎餃

規律：

- 煎餃 → 自助餐 → 大魯麵 → 自助餐 → 黃金炒飯 → 大魯麵

下載與瀏覽資料集

- 要記住多久之前的資料，才能找出規律？

煎餃 → 自助餐 → 大魯麵 → 自助餐 → 黃金炒飯 → 大魯麵

一天前：　煎餃 → 自助餐　自助餐 → 大魯麵？ / 黃金炒飯？

兩天前：　煎餃、自助餐 → 大魯麵

大魯麵、自助餐 → 黃金炒飯

下載相關函式庫＆資料集

```
# Install HappyML
import os

if not os.path.isdir("HappyML"):
    os.system("git clone https://github.com/cnchi/HappyML.git")

# Get Dataset
Dataset_File = 'Meals.csv'

if not os.path.isfile(Dataset_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dataset_File)
```

下載 HappyML

下載 Meals.csv 資料集

設定「可客製化」常數

```
1  # In[] Customizable Constants
2
3  train_size = 0.75
4
5  win_size = 2
6  sample_step = 1
7  win_moving = 1
8
9  data_batch = 10
```

Training Set 佔 75%

win_size　win_moving

煎餃 → 自助餐 → 大魯麵 → 自助餐 → ...

sample_step → [煎餃, 自助餐]

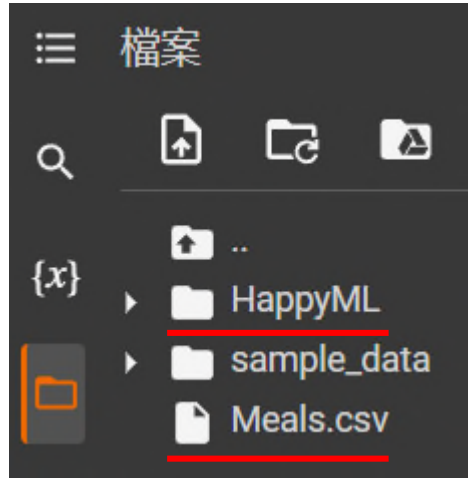
每次產生 10 筆過往資料

[煎餃, 自助餐], [自助餐, 大魯麵], ... (共 10 筆)

隨堂練習：環境設定

- 請將下列程式碼寫好，執行看看。
- 請看一下左側的「檔案」面板，是否已經有 HappyML 與 Meals.csv 這些相關函式庫與檔案。

```
1  # Install HappyML
2  import os
3
4  if not os.path.isdir("HappyML"):
5      os.system("git clone https://github.com/cnchi/HappyML.git")
6
7  # Get Dataset
8  Dataset_File = 'Meals.csv'
9
10 if not os.path.isfile(Dataset_File):
11     os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dataset_File)
12
13 # Customizable Constants
14 train_size = 0.75
15
16 win_size = 2
17 sample_step = 1
18 win_moving = 1
19
20 data_batch = 10
```



資料集前處理

```
1  # Load Data
2  import HappyML.preprocessor as pp
3  import numpy as np
4
5  dataset = pp.dataset(file="Meals.csv")
6
7  # Remove the date, keep meal only
8  dataset = pp.onehot_encoder(dataset, columns=[1]).iloc[:, 1:]
```

載入 Meals.csv 資料集
（DataFrame 格式）

Index	Date	Meal
0	2020/1/1	煎餃
1	2020/1/2	自助餐
2	2020/1/3	大魯麵
3	2020/1/4	自助餐
4	2020/1/5	黃金炒飯

onehot_encoder()

Date	Meal_大魯麵	Meal_煎餃	Meal_自助餐	Meal_黃金炒飯
2020/1/1	0	1	0	0
2020/1/2	0	0	1	0

.iloc[: , 1:]

	Meal_大魯麵	Meal_煎餃	Meal_自助餐	Meal_黃金炒飯
0	0	1	0	0
1	0	0	1	0

資料集前處理

```
1  # Keep the One Hot Encoding Mapping
1  mapper = [s[5:] for s in list(dataset.columns)]
3
4  # Convert dataset from DataFrame as NDArray
2  dataset = dataset.values
6  train_data, test_data = np.split(dataset, [int(train_size * len(dataset))])
```

dataset.columns

list(Meal_大魯麵 | Meal_煎餃 | Meal_自助餐 | Meal_黃金炒飯)

['Meal_大魯麵', 'Meal_煎餃', 'Meal_自助餐', 'Meal_黃金炒飯']

mapper = ['大魯麵', '煎餃', '自助餐', '黃金炒飯']

int(0.75 x 31 / 23.25)

np.split(dataset, [23])

31 列

	0	1	2	3
0	0	1	0	0
1	0	0	1	0
...				
30	0	1	0	0

dataset[:23, :] → train_data

dataset[23: , :] → test_data

隨堂練習：資料集前處理

- 請將前幾頁的程式碼寫好，執行看看。

- 請用程式碼儲存格看一下 mapper 變數的內容，是否長得像這樣：

mapper	list	4	['大魯麵', '煎餃', '自助餐', '黃金炒飯']

- 請用程式碼儲存格看一下 train_data 與 test_data 的內容，是否長得像這樣：

train_data

	0	1	2	3
0	0	1	0	0
1	0	0	1	0
...				
22	0	0	0	1

test_data

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
...				
7	0	1	0	0

RNN 資料集的準備

```
1  # In[] Data Preprocessing as X & Y
2  from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
```

- 何謂 TimeseriesGenerator？

可以幫你把這種資料

Meal
煎餃
自助餐
大魯麵
自助餐
黃金炒飯
大魯麵

準備成這種資料

前兩天吃什麼（X）	今天吃什麼（Y）
[煎餃, 自助餐]	[大魯麵]
[自助餐, 大魯麵]	[自助餐]
[大魯麵, 自助餐]	[黃金炒飯]
[自助餐, 黃金炒飯]	[大魯麵]

RNN 資料集的準備

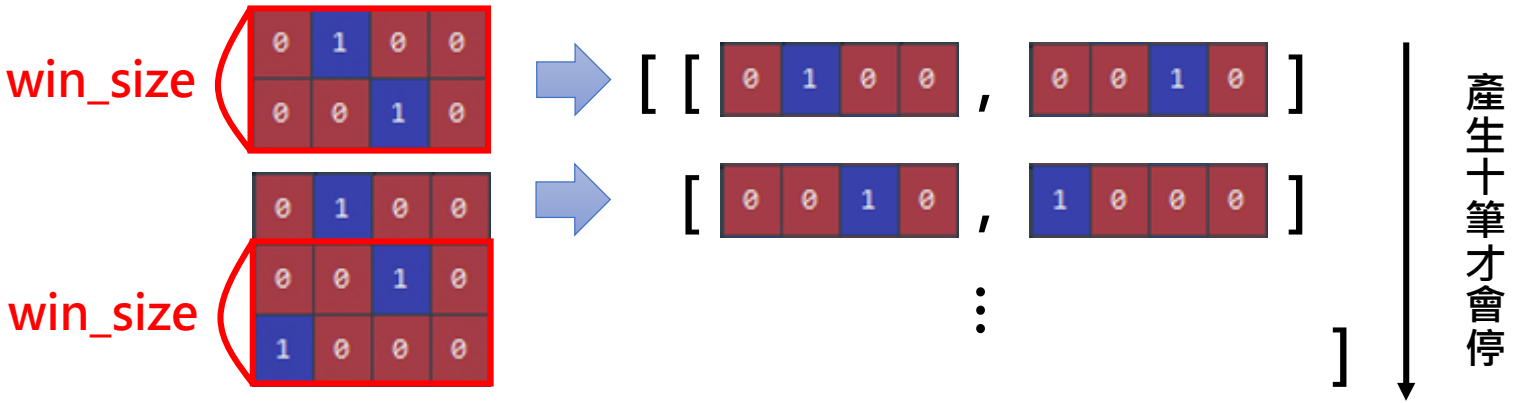
- 訓練集部分（測試集部分雷同，略）

```
4  train_set = TimeseriesGenerator(
5          data=train_data,  ← 自變數 X 所在地
6          targets=train_data,  ← 應變數 Y 所在地
7          length=win_size,  ← 過往視窗大小（=2）
8          sampling_rate=sample_step, ← 視窗內取樣頻率（=1）
9          stride=win_moving,  ← 視窗滑動距離（=1）
10         batch_size=data_batch)  ← 一批產生多少資料集（=10）
```

train_data

	大魯麵	煎餃	自助餐	黃金炒飯
	0	1	2	3
0	0	1	0	0
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0

X　Y　win_size



RNN 資料集的準備

- **TimeseriesGenerator 回傳值型態**

  是一個 tensorflow.keras.preprocessing 套件之下的 sequence 物件

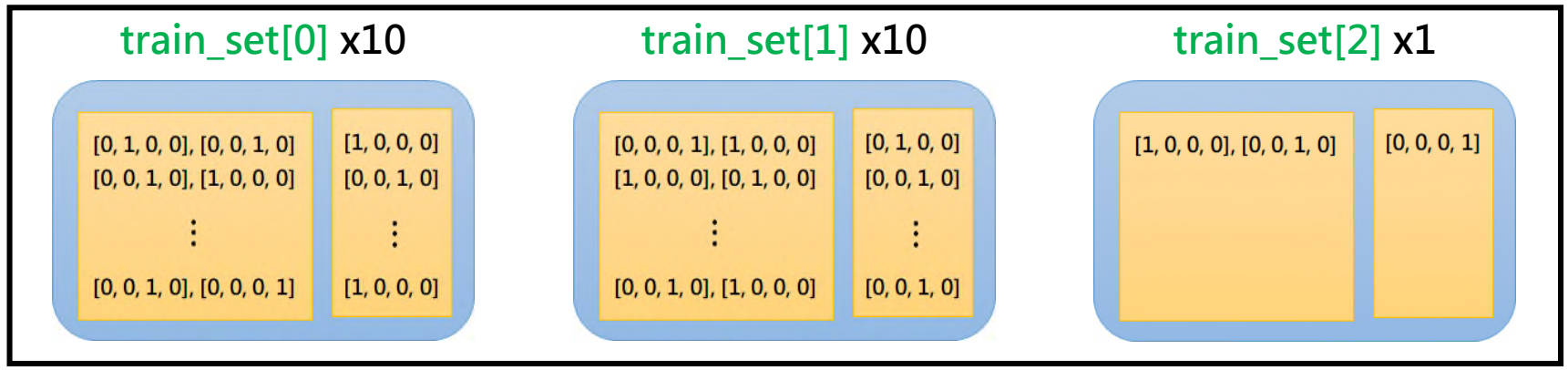
```
>>> print(len(train_set))
3
```

train_set 是一個「長度為 3」的 sequence 陣列

可以做出 21 筆輸入資料

（扣掉最前兩筆、沒有「前天」與「昨天」吃飯記錄）

data_batch = 10



RNN 資料集的準備

- RNN 資料集的標準格式

```
1  import numpy as np
2
3  X_train = np.array(
4          [
5              [[0, 1, 0, 0], [0, 0, 1, 0]],
6              [[0, 0, 1, 0], [1, 0, 0, 0]],
7              [[1, 0, 0, 0], [0, 0, 1, 0]],
8                      ......
9              [[0, 0, 0, 1], [1, 0, 0, 0]]
10         ])
11
12 Y_train = np.array(
13         [
14             [1, 0, 0, 0],
15             [0, 0, 1, 0],
16             [0, 0, 0, 1],
17                 ...
18             [0, 1, 0, 0]
19         ])
```

第 1 維：過往時步數

第 0 維：樣本數

第 2 維：特徵數

第 0 維：樣本數

第 1 維：標準答案

自變數 X：必為「三維」

（樣本數, 時步數, 特徵數）

應變數 Y：「一維」或「二維」

「一維」（樣本數）：[0, 2, 3, ...1]

「二維」（樣本數, 標準答案）：

[[1, 0, 0], [0, 0, 1], ...]

RNN 資料集的準備

- 完整程式碼

```
# In[] Data Preprocessing as X & Y
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

train_set = TimeseriesGenerator(
        data=train_data,
        targets=train_data,
        length=win_size,
        sampling_rate=sample_step,
        stride=win_moving,
        batch_size=data_batch)

test_set = TimeseriesGenerator(
        data=test_data,
        targets=test_data,
        length=win_size,
        sampling_rate=sample_step,
        stride=win_moving,
        batch_size=data_batch)
```

隨堂練習：資料集準備

- 請將前幾頁的程式碼寫好，執行看看。

- 執行完畢之後，請於「草稿儲存格」輸入下列指令。瞭解在十筆一批的設定下，一共有幾批資料：
  - len(train_set)：瞭解訓練集一共有幾批
  - len(test_set)：瞭解測試集一共有幾批

- 請再輸入下列指令，瞭解每一批資料大概的長相
  - train_set[0]
  - train_set[1]
  - train_set[2]
  - test_set[0]

模型建造 & 訓練

```
# In[] Create Model
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
    6 你可以自己多堆疊幾層 SimpleRNN 上去
# TO-DO: Add more RNN layers & do the validation set
model = Sequential() 1 簡單 RNN 所使用的物件  2 輸入層用（None, 時步數, 特徵數）做為維度
model.add(layers.SimpleRNN(10, input_shape=(win_size, dataset.shape[1])))
model.add(layers.Dense(dataset.shape[1], activation="softmax"))
model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["acc"])
          4 RNN 用 RMSProp 較有利    3 輸出層為「四選一」（四種餐點）的問題
# In[] Train Model
    7 你可以用 TensorBoard 找最佳訓練週期
# TO-DO: Find the best epochs
model.fit(train_set, epochs=50)
```

只要輸入這兩個維度就夠

5 開始訓練

隨堂練習：建造&訓練模型

- 請將前幾頁的程式碼寫好，執行看看。
- 您最後顯示的「正確率」有多少呢？

用「測試集」評估你的模型

```
1  # In[] Model Evaluation
2  test_loss, test_acc = model.evaluate(test_set)
3  print("Loss of Test Set:", test_loss)
4  print("Accuracy of Test Set:", test_acc)
```

```
1/1 [==============================] - 0s 235ms/step - loss: 0.8041 - acc: 0.8333
Loss of Test Set: 0.8041180968284607
Accuracy of Test Set: 0.83333331346511184
```

正確率約 83.33%

用「測試集」預測中午吃什麼？

```
# In[] Model Prediction
Y_pred = np.argmax(model.predict(test_set), axis=-1)
Y_real = np.argmax(test_set[0][1], axis=-1)

# Convert from number to label
print("Prediction:", [mapper[i] for i in Y_pred])
print("Real Value:", [mapper[i] for i in Y_real])
```

4 第 0 批測試集 test_set[0]

自變數 X_test
test_set[0][0]

應變數 Y_test
test_set[0][1]

```
(array([[[1, 0, 0, 0], [0, 1, 0, 0]],
        [[0, 1, 0, 0], [0, 0, 1, 0]],
        [[0, 0, 1, 0], [1, 0, 0, 0]],
        [[1, 0, 0, 0], [0, 0, 1, 0]],
        [[0, 0, 1, 0], [0, 0, 0, 1]],
        [[0, 0, 0, 1], [1, 0, 0, 0]]]),
 array([[0, 0, 1, 0],
        [1, 0, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1],
        [1, 0, 0, 0],
        [0, 1, 0, 0]]))
```

1 model.predict(test_set) 的結果

```
[[0.07518168, 0.08385396, 0.4345999 , 0.40636444],
 [0.6038025 , 0.1214216 , 0.15407781, 0.12069804],
 [0.29100767, 0.11095769, 0.5400307 , 0.05800389],
 [0.03912609, 0.05065365, 0.08229412, 0.82792616],
 [0.5974031 , 0.05617252, 0.31091046, 0.03551393],
 [0.11970779, 0.6071506 , 0.15893021, 0.11421138]]
```

大魯麵　煎餃　自助餐　黃金炒飯

維度：（6 筆, 4 特徵）

2 以最後一個（axis=-1）維度（4 特徵）為準

```
      [0]         [1]         [2]         [3]
[[0.07518168, 0.08385396, 0.4345999 , 0.40636444],
 [0.6038025 , 0.1214216 , 0.15407781, 0.12069804],
 [0.29100767, 0.11095769, 0.5400307 , 0.05800389],
 [0.03912609, 0.05065365, 0.08229412, 0.82792616],
 [0.5974031 , 0.05617252, 0.31091046, 0.03551393],
 [0.11970779, 0.6071506 , 0.15893021, 0.11421138]]
```

3 傳回每列最大數字的索引值（如：[2]）

Y_pred = [2, 0, 2, 3, 0, 1]

mapper

5 Y_pred = [2, 0, 2, 3, 0, 1]

0	1	2	3
大魯麵	煎餃	自助餐	黃金炒飯

[“自助餐”, “大魯麵”, ...]

用「測試集」預測中午吃什麼？

- 程式碼執行結果

```
1  # In[] Model Prediction
2  Y_pred = np.argmax(model.predict(test_set), axis=-1)
3  Y_real = np.argmax(test_set[0][1], axis=-1)
4
5  # Convert from number to label
6  print("Prediction:", [mapper[i] for i in Y_pred])
7  print("Real Value:", [mapper[i] for i in Y_real])
```

```
1/1 [==============================] - 0s 215ms/step
Prediction: ['黃金炒飯', '大魯麵', '自助餐', '黃金炒飯', '大魯麵', '煎餃']
Real Value: ['自助餐', '大魯麵', '自助餐', '黃金炒飯', '大魯麵', '煎餃']
```

還不錯！

預測錯誤之處

隨堂練習：模型的評估＆預測

- 請將前幾頁的程式碼寫好，執行看看。
- 您用 .evaluate() 評估出來的「正確率」有多少呢？
- 您的測試集預測結果，與真實結果相差多少呢？

預測使用者的輸入資料

午餐潛規則： 煎餃 → 自助餐 → 大魯麵 → 自助餐 → 黃金炒飯 → 大魯麵

```
1  # In[] User input
2  from tensorflow.keras.utils import to_categorical
3
4  meals = []
5  meals += [mapper.index(input("請問你前天吃什麼："))]
6  meals += [mapper.index(input("請問你昨天吃什麼："))]
7
8  meals = np.reshape(to_categorical(meals, dataset.shape[1]), (1, win_size, dataset.shape[1]))
9  meal_today = np.argmax(model.predict(meals), axis=-1)
10 print("我猜你今天會吃：", mapper[meal_today[0]])
```

1 “大魯麵” “自助餐”

2 mapper

0	1	2	3
大魯麵	煎餃	自助餐	黃金炒飯

3 meals: [] += [0] +=[2] → meals: [0, 2]

4 to_categorical([0, 2], 4)

(筆數, 時步, 特徵)

5 np.reshape([[1, 0, 0, 0], [0, 0, 1, 0]], (1, 2, 4))

x2 時步

6 meals = [[[1., 0., 0., 0.], [0., 0., 1., 0.]]] x1 筆

x4 特徵

7 model.predict(meals) 的結果

argmax() [0] [1] [2] [3]

[[0.03912609, 0.05065365, 0.08229412, 0.82792616]]

meal_today = [3] (meal_today[0] == 3)

8 mapper[3] → “黃金炒飯”

預測使用者的輸入資料

- 程式碼執行結果

```
1  # In[] User input
2  from tensorflow.keras.utils import to_categorical
3
4  meals = []
5  meals += [mapper.index(input("請問你前天吃什麼："))]
6  meals += [mapper.index(input("請問你昨天吃什麼："))]
7
8  meals = np.reshape(to_categorical(meals, dataset.shape[1]), (1, win_size, dataset.shape[1]))
9  meal_today = np.argmax(model.predict(meals), axis=-1)
10 print("我猜你今天會吃：", mapper[meal_today[0]])
```

```
請問你前天吃什麼：大滷麵

請問你昨天吃什麼：自助餐
我猜你今天會吃：黃金炒飯
```

隨堂練習：模型的評估＆預測

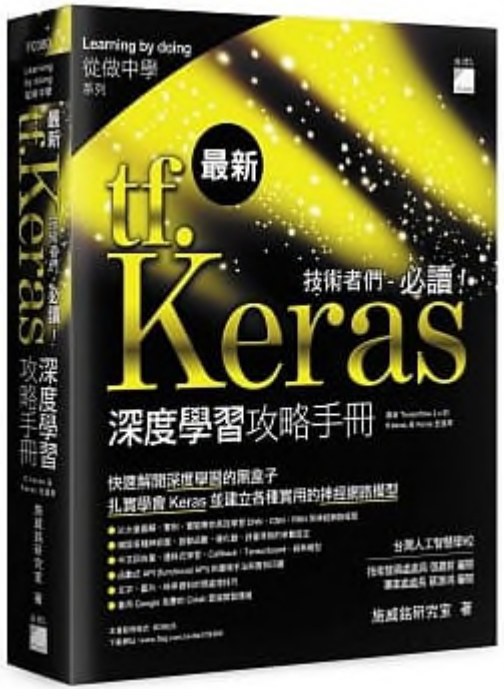
- 請將前幾頁的程式碼寫好，執行看看。
- 依照下列的「午餐潛規則」，輸入兩個值，讓模型預測看看：

  煎餃 → 自助餐 → 大魯麵 → 自助餐 → 黃金炒飯 → 大魯麵

- 您的預測結果，是否正確呢？

其實，還有另一種 RNN...

- **Stateful RNN**：可以**自動**決定「**最適時步**」的 RNN 模型



哪裡可以學得到：

- **tf.Keras** 技術者們必讀！（第 4-2 小節）
- 旗標出版社（2020-02）
- 已絕版，建議到圖書館借閱。
- 博客來連結：https://bit.ly/2GyB9Cl

為何老師不教：

- 下一小節的 **LSTM** 更好用！
- **LSTM** 可以完全取代 **Stateful RNN**！

長短期記憶簡介

Introduction to Long Short-term Memory (LSTM)

何謂 LSTM？

“Long” Short-term Memory

(X Long / Short Term Memory)

O 正確翻譯： 長效 短期記憶

X 錯誤翻譯： 長或短 期記憶

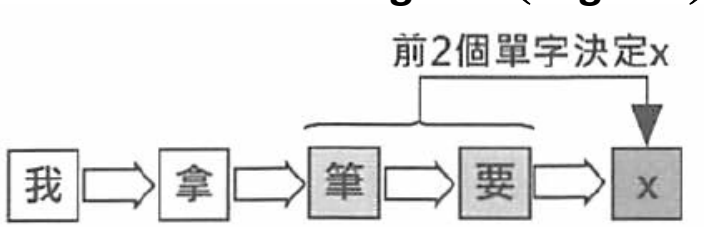
發明人： 德國 Hochreiter & Schmidhuber, 1997

為何會有 LSTM？

- 簡單 RNN 的問題：（1）時步長度固定

簡單 RNN
(Vanilla / Naïve RNN)

n-gram（2-gram）

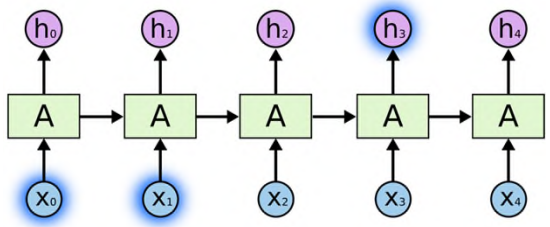


我拿“筆要”→「寫字」

固定使用 2-grams
(” 拿筆” 比 “筆要” 更有參考價值)

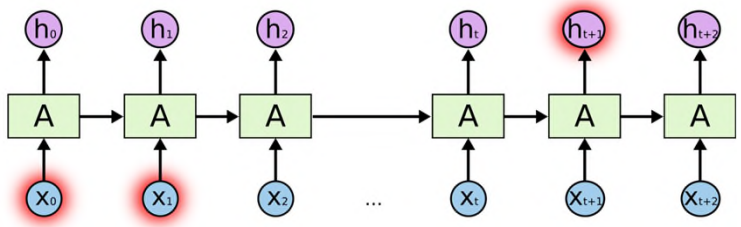
實務上

The clouds are in the sky.



有時所需時步長度較短

I am in France... I speak *French*.



有時所需時步長度較長

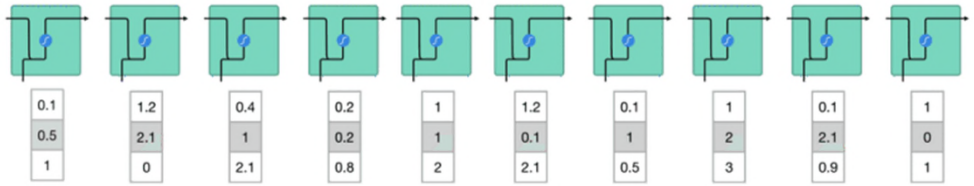
時步長度 n，應該也得是「可訓練、可變動」的參數才對！

為何會有 LSTM？

- 簡單 RNN 的問題：（2）長時步 RNN 梯度爆炸 or 消失問題

時步 n = 120（股票半年線）

權重向前傳遞



權重向後修正

權重 W = 1.1 （時步 n = 120）

$$1.1^{120} = 92709.0688\ldots$$

梯度爆炸！！

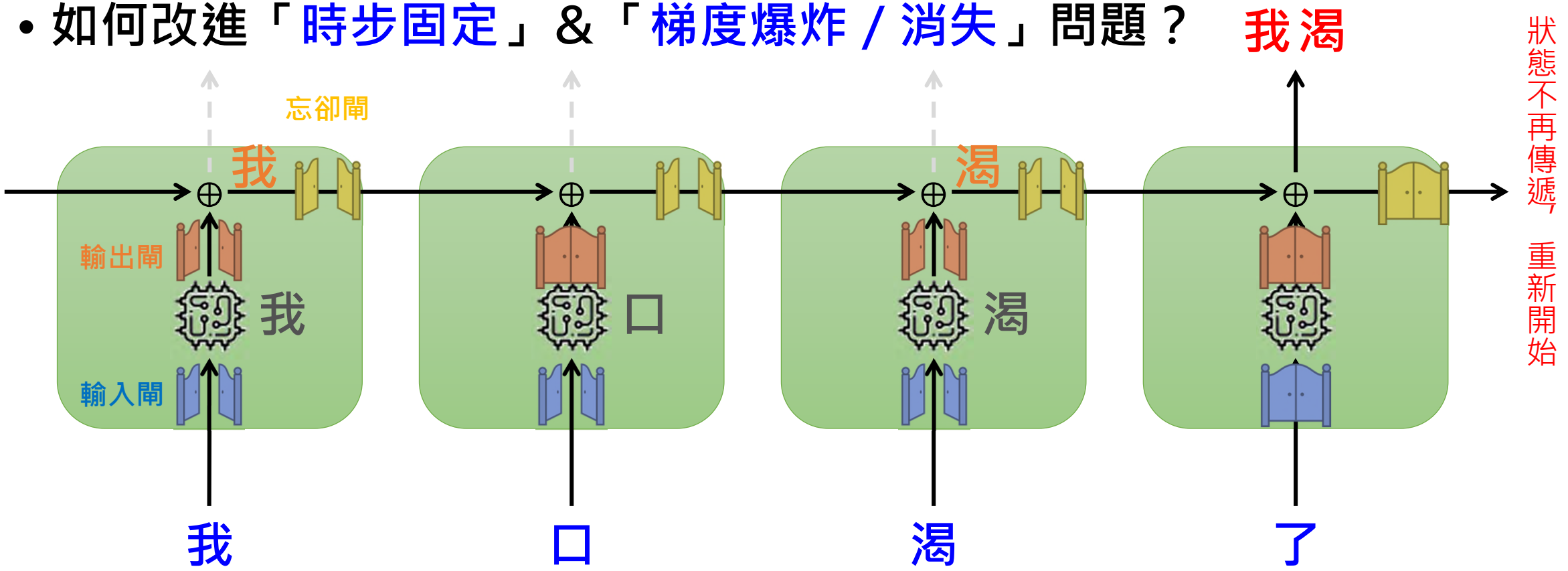
權重 W = 0.9 （時步 n = 120）

$$0.9^{120} = 0.00000323\ldots$$

梯度消失～～

為何會有 LSTM？

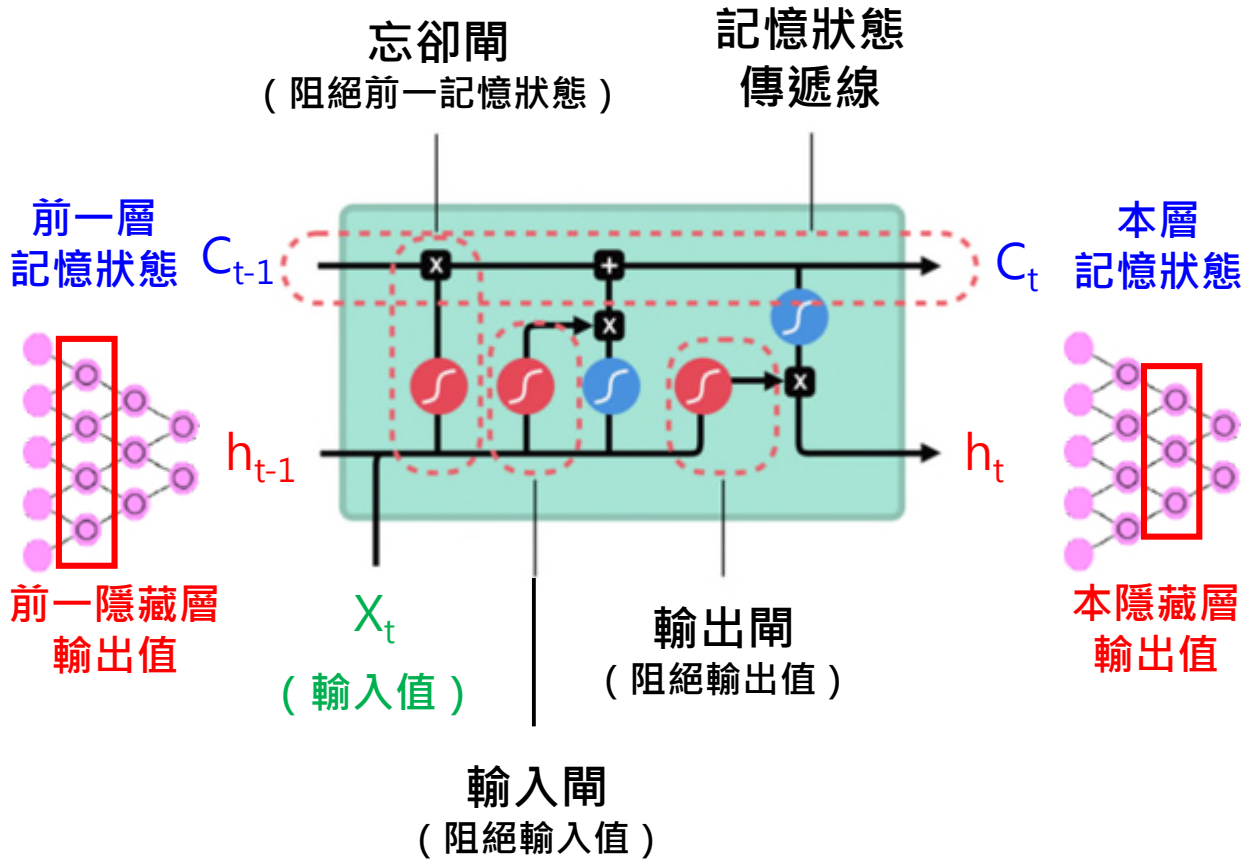
- 如何改進「時步固定」&「梯度爆炸 / 消失」問題？

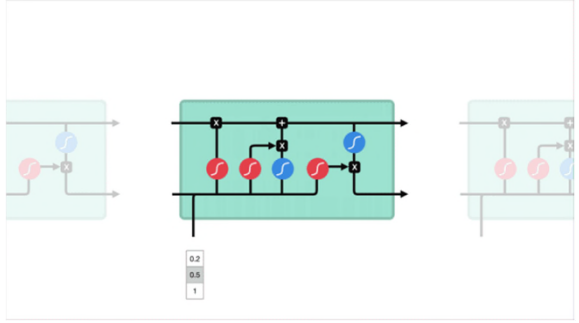


輸入閘、輸出閘：可學習，自由開關 → 時步不再固定！　　忘卻閘：關閉後，狀態不再傳遞 → 梯度解決！

LSTM 結構

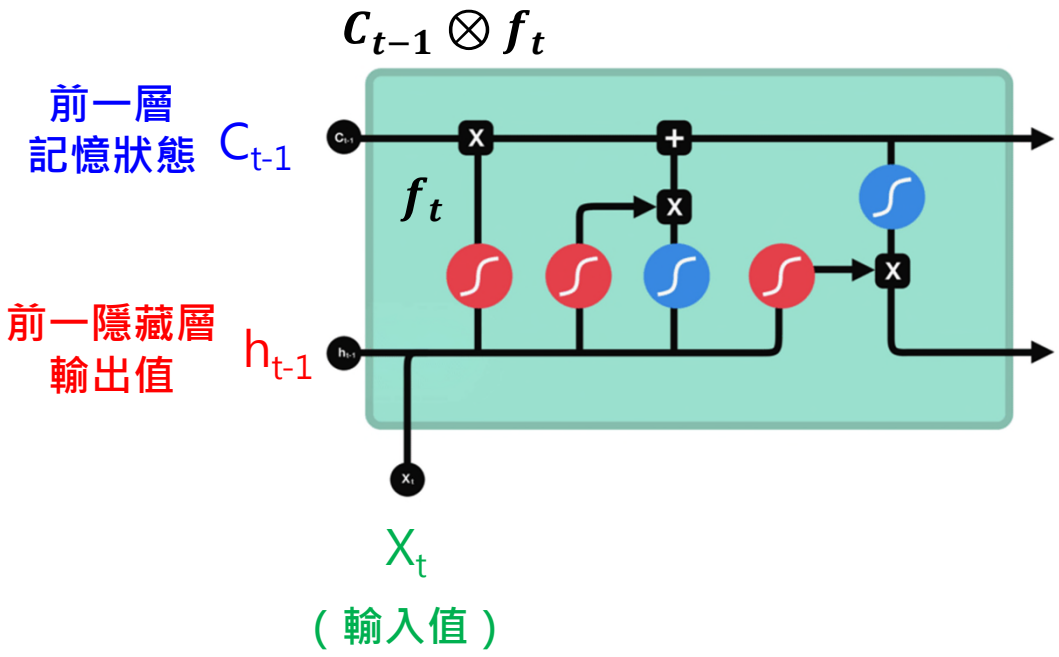
- 隆重介紹：LSTM 單元！





LSTM 結構

- 忘卻閘（Forget Gate）

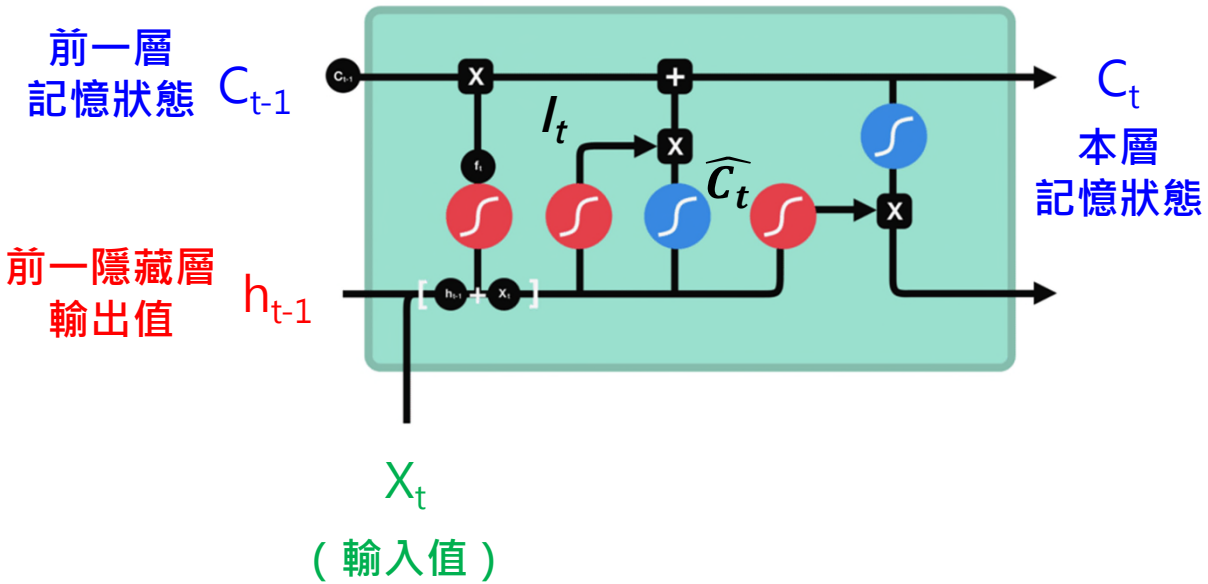


- 作用：
  - 切斷來自迴授線路前一層狀態 C_{t-1}。
- 運作原理：
  - 合併本層輸入值與前隱藏層輸出值 $[X_t, h_{t-1}]$ 做為最終輸入值。
  - 乘上可訓練的忘卻閘權重 W_f
    $$W_f \odot [X_t \quad h_{t-1}]$$
  - 有時會加上忘卻閘偏移量 b_f
    $$W_f \odot [X_t \quad h_{t-1}] + b_f$$
  - 最後使用 Sigmoid 激活函數 σ，來決定這次是否要「忘卻」前一層記憶狀態。若 $f_t < 0.5$→忘卻之：
    $$f_t = \sigma(W_f \odot [X_t \quad h_{t-1}] + b_f)$$

LSTM 結構

- **輸入閘（Input Gate）**

$$C_t = C_{t-1} \oplus (I_t \otimes \widehat{C_t})$$



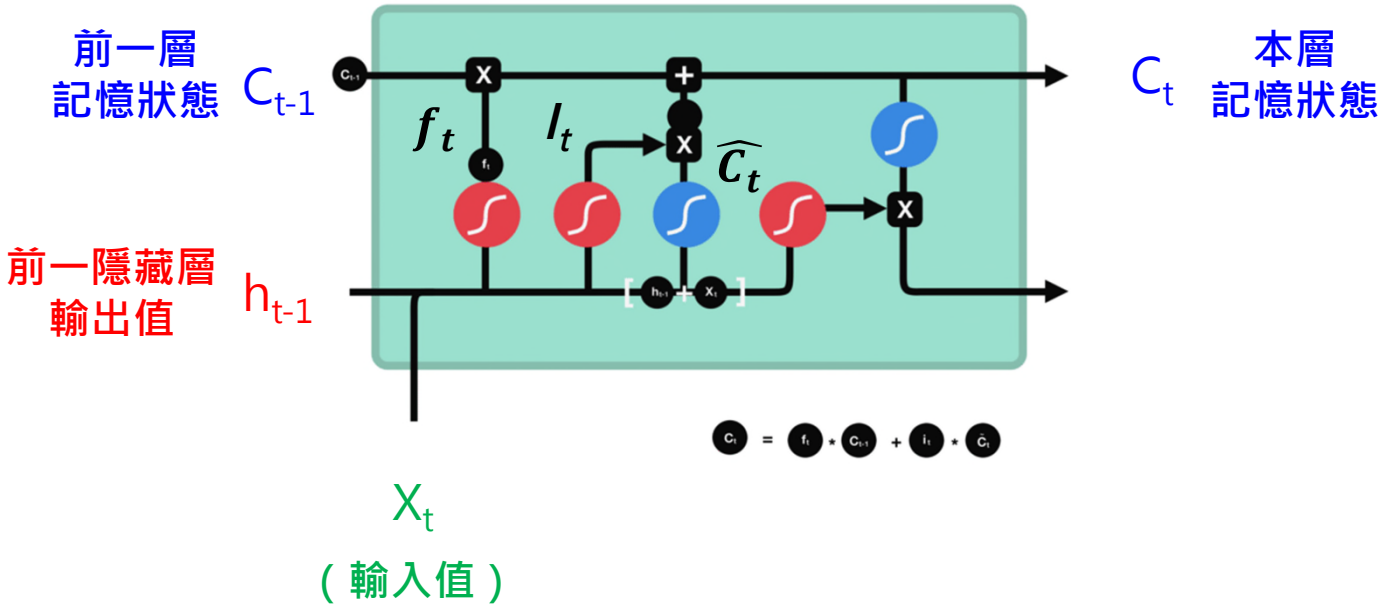
- 作用：
  - 決定是否放行**輸入值 X_t** 至 Memory Cell。以便將來可以將它「解凍」使用。

- **運作原理**：
  - 合併**本層輸入值**與**前隱藏層輸出值** [X_t, h_{t-1}] 做為**最終輸入值**。
  - **最終輸入值**搭配**輸入閘權重 W_i**，以及 **Sigmoid 函數**，決定輸入閘是否開啟：$I_t = \sigma(W_i \odot [X_t \quad h_{t-1}] + b_i)$
  - 另一方面，**最終輸入值**搭配**資料權重 Wc**，以及 **tanh() 函數**，預測下一個狀態值：$\widehat{C_t} = tanh(W_c \odot [X_t \quad h_{t-1}] + b_c)$
  - $\widehat{C_t}$ 會因為 **tanh() 函數**的關係，而被映射至範圍（-1, 1）
  - 用 $(I_t \otimes \widehat{C_t})$ 來決定 $\widehat{C_t}$ 是否可以成為狀態的一部份。

LSTM 結構

- 記憶狀態傳遞線（Cell State Line）

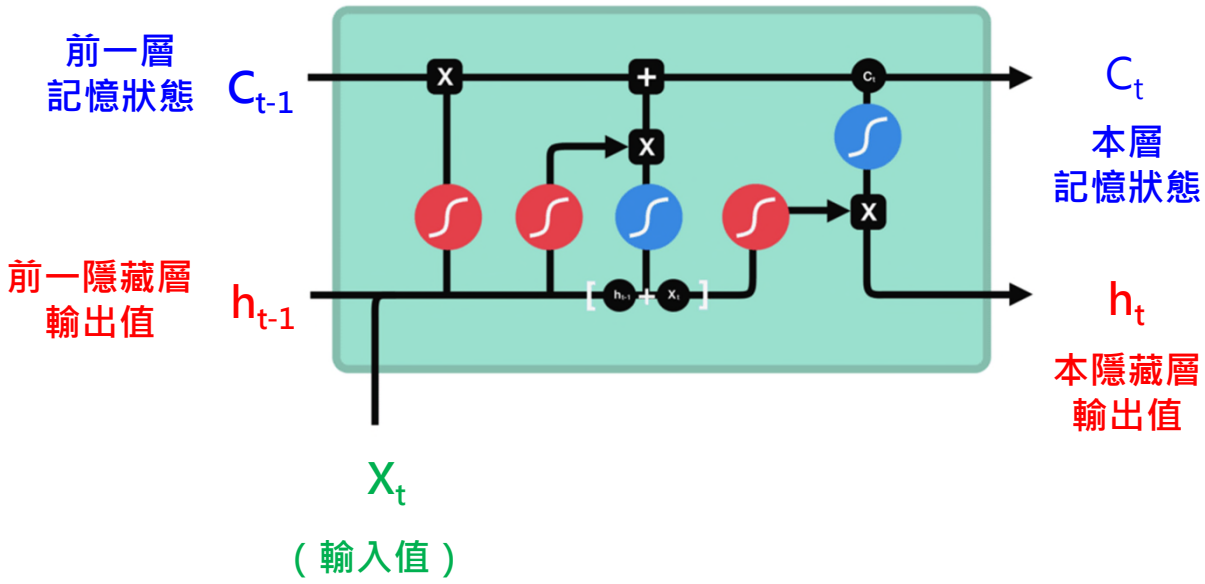
$$C_t = (C_{t-1} \otimes f_t) \oplus (I_t \otimes \widehat{C_t})$$



LSTM 結構

- **輸出閘（Output Gate）**

$$C_t = (C_{t-1} \otimes f_t) \oplus (I_t \otimes \widehat{C_t})$$



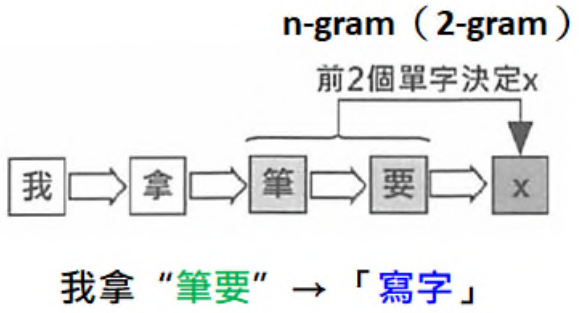
- **作用：**
  - 決定是否將**記憶狀態**，**流放**到下一個隱藏層。

- **運作原理：**
  - 合併**本層輸入值**與**前隱藏層輸出值** [X_t, h_{t-1}] 做為**最終輸入值**。
  - **最終輸入值**搭配**輸出閘權重 W_o**，以及 **Sigmoid 函數**，決定輸入閘是否開啟：$O_t = \sigma(W_o \odot [X_t \quad h_{t-1}] + b_o)$
  - 另一方面，**本次記憶狀態 C_t**，配合 **tanh() 函數**，將輸出重映射至範圍 (-1, 1)：
    $$h_t = tanh(C_t)$$
  - 用 $(O_t \otimes C_t)$ 來決定 C_t 是否可以成為**本隱藏層輸出值**。

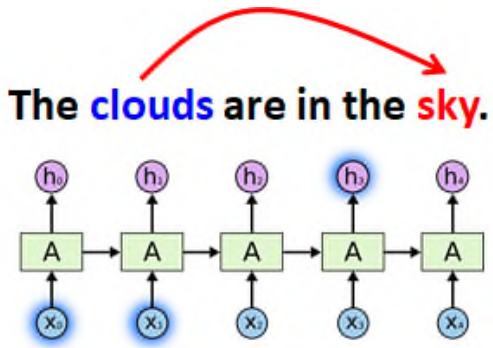
何時該用 LSTM

- 需要彈性的「非連續」時步
- 「時步」超級長、擔心梯度爆炸或消失

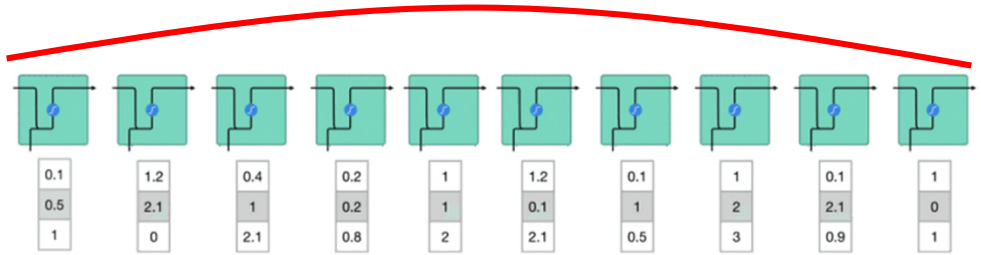
X



O



時步 n = 120



小節整理：LSTM

- 什麼是 LSTM：有「輸入、輸出、忘卻」三閘，可以自由地
  - 選擇要記住什麼（輸入閘）
  - 選擇要傳遞什麼（輸出閘）
  - 選擇之前記住什麼、但現在不要了（忘卻閘）

- LSTM 的好處
  - 透過「輸入+輸出閘」，可以把模型訓練成時步長度次次不同。
  - 透過「忘卻閘」，可以在梯度爆炸、或消失之前，丟棄不正確的成果。
  - 簡單 RNN 的短期記憶，可以改用 LSTM 「長一點」的短期記憶，不用擔心梯度爆炸、或消失問題。

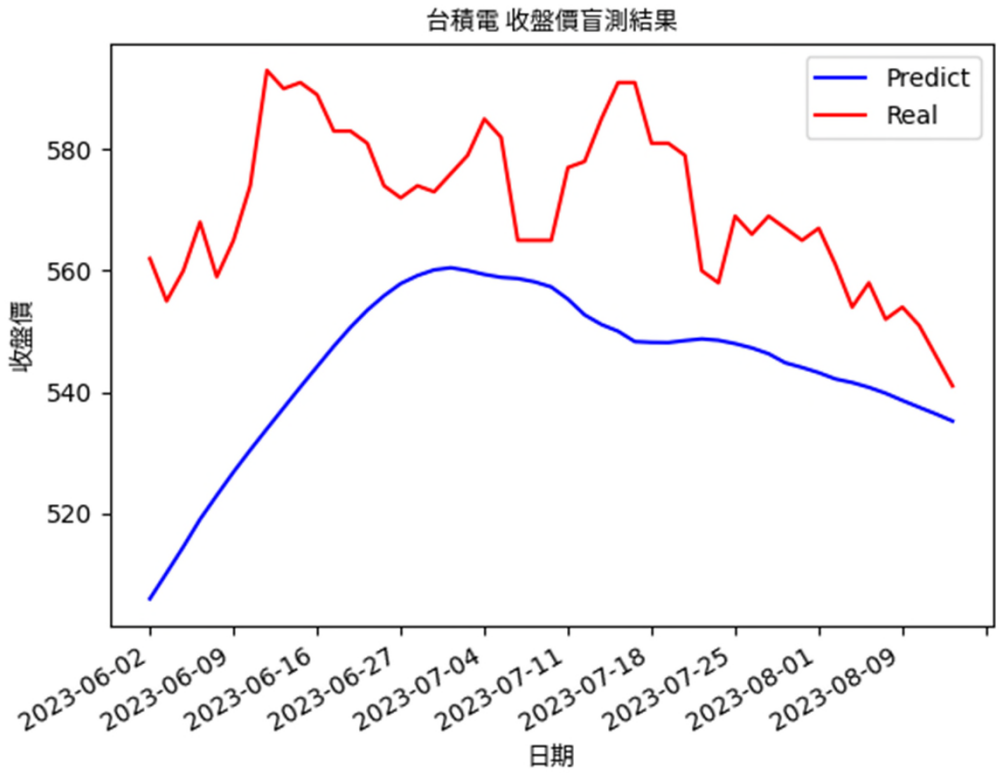
- 何時該用 LSTM
  - 所需「時步」是跳躍、不固定的。
  - 時步很長，擔心有梯度爆炸或消失時

長短期記憶範例

The Example of
Long Short-term Memory (LSTM)

範例完整原始碼：
https://lurl.cc/8CZzQx

範例介紹：台灣股價預測



- **資料說明**
  - **來源**：Yahoo Finance
  - **欄位**：
    - 開盤、收盤、最高、最低、成交量
- **方法解說**
  - **自變數**：過去 30 天的
    - 開盤、收盤、最高、最低、成交量
  - **應變數**：今日收盤價

取得必要函式庫＆資料集

```
# Install HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")

# Download Dataset
Dataset_File = 'TaiwanStockID.csv'

if not os.path.isfile(Dataset_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dataset_File)

# Install Yahoo Finance package
!pip install yfinance
```

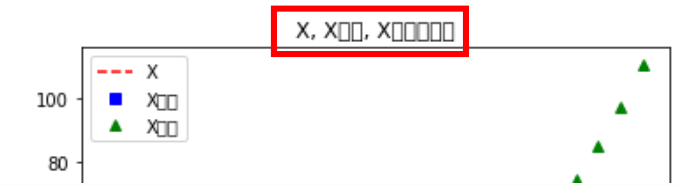
下載 HappyML

下載 TaiwanStockID.csv 資料集
（股票名稱 vs. 股票代號對照表）

安裝 Yahoo Finance
（抓取股票漲跌資料）

安裝中文字形

解決顯示中文時的亂碼問題




```
# Chinese Font Settings for Chart Plotting (Colab)
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

# 用這個方法可以檢查系統有沒有中文字體（空空如也=沒有）
!fc-list :lang=zh family
# 下載台北思源黑體，並命名taipei_sans_tc_beta.ttf
!wget -O taipei_sans_tc_beta.ttf https://drive.google.com/uc?id=1eGAsTN1HBpJAkeVM57_C7ccp7hbgSz3_&export=download
# 移至指定路徑
!mv taipei_sans_tc_beta.ttf /usr/local/lib/python3.10/dist-packages/matplotlib/mpl-data/fonts/ttf

# 自定義字體變數
myfont = FontProperties(fname=r'/usr/local/lib/python3.10/dist-packages/matplotlib/mpl-data/fonts/ttf/taipei_sans_tc_beta.ttf')

# 後續在相關函式中增加fontproperties=myfont屬性即可。如：plt.xlabel("時間", fontproperties=myfont)
```

← 非必要，示範用指令，需手工輸入至「草稿儲存格」觀看結果

需依照 Colab 使用的 Python 版本做更改

```
# Install Yahoo Finance package
!pip install yfinance
```

安裝 yfinance 時可以得知

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.28)
```

重要常數設定

```
1  # Customizable Constants
2  train_size = 0.6
3  val_size = 0.2
4
5  win_size = 30
6  sample_step = 1
7  win_moving = 1
8
9  data_batch = 5
```

← 訓練集 60%、驗證集 20%、測試集 20%

← 時步長度：30 天、取樣頻率：每天、滑動距離：1 天

← 指定「時步資料」一次要產出幾筆

引入必要套件

```
1  # Import necessary libraries
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5
6  import dateutil.parser as psr
7
8  import yfinance as yf
```

←可將任意「日期」輸入格式，
轉換成 datetime 物件的套件

← 抓取股價資料的套件

隨堂練習：執行環境設定

- 將前幾頁程式碼撰寫好、執行看看。確認沒有錯誤訊息：

```
# Install HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")

# Download Dataset
Dataset_File = 'TaiwanStockID.csv'

if not os.path.isfile(Dataset_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dataset_File)

# Install Yahoo Finance package
!pip install yfinance

# Chinese Font Settings for Chart Plotting (Colab)
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

# Check for Chinese Fonts (Optional)
!fc-list :lang=zh family
```

隨堂練習：執行環境設定（續）

```
# Download Chinese Fonts
!wget -O taipei_sans_tc_beta.ttf https://drive.google.com/uc?id=1eGAsTN1HBpJAkeVM57_C7ccp7hbgSz3_&export=download
# Move Chinese Fonts to /matplotlib path
!mv taipei_sans_tc_beta.ttf /usr/local/lib/python3.10/dist-packages/matplotlib/mpl-data/fonts/ttf

# Define variable for Chinese Font
myfont = FontProperties(fname=r'/usr/local/lib/python3.10/dist-packages/matplotlib/mpl-data/fonts/ttf/taipei_sans_tc_beta.ttf')

# Customizable Constants
train_size = 0.6
val_size = 0.2

win_size = 30
sample_step = 1
win_moving = 1

data_batch = 5

# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import dateutil.parser as psr

import yfinance as yf
```

取得股票名稱、起迄日期

```
1  # Read Stock ID
2  stockIDMap = pd.read_csv("TaiwanStockID.csv")
3
4  stockInput = input("請輸入台灣股票名稱、或代號：")
5  if stockInput.isdigit():
6      stockID = int(stockInput)
7      condition = stockIDMap["StockID"] == stockID
8      stockName = stockIDMap[condition].iloc[0]["StockName"]
9  else:
10     stockName = stockInput
11     condition = stockIDMap["StockName"] == stockName
12     stockID = stockIDMap[condition].iloc[0]["StockID"]
13
14 startDate = psr.parse(input("請輸入查詢起始日期："))
15 endDate = psr.parse(input("請輸入查詢截止日期："))
```

← 讀入「股票名稱 vs. 股票代號」對照表

← 使用者輸入股票名稱 or 股票代號

← 如果輸入的是「股票代號」，不用轉換。

← 如果是「名稱」，進行轉換

← 可將任何日期格式，轉成 datetime

"Aug. 31 2020"
"2020-08-31"
"31/8/2020"

→ datetime(2020, 8, 31, 0, 0, 0)

讀入股價資訊

```
1  # Download the Stock Data
2  stockQuery = "{}.TW".format(stockID)  ← 標準代號。如：" 2330.TW" 、" GOOL" ...
3  dataset = yf.download(stockQuery, start=startDate.strftime("%Y-%m-%d"), end=endDate.strftime("%Y-%m-%d"))
```

"2022-01-01"　　"2023-08-15"

Yahoo Finance 要求之日期標準格式

隨堂練習：讀入資料集

- 請將前幾頁的程式碼寫好，並執行看看：

```
請輸入台灣股票名稱、或代號: 台積電
請輸入查詢起始日期: Jan 1 2022
請輸入查詢截止日期: Aug 15 2023
```

請提供至少一年半以上的區間，
否則最後繪圖的時候會不好看。

- 使用「草稿儲存格」，檢查 dataset 變數，確認股價資料已經成功讀入：

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-03	619.0	632.0	618.0	631.0	611.182373	69089158
2022-01-04	645.0	656.0	644.0	656.0	635.397217	79434666
2022-01-05	669.0	669.0	646.0	650.0	629.585571	69593809
2022-01-06	638.0	646.0	636.0	644.0	623.774048	53210211
2022-01-07	643.0	646.0	632.0	634.0	614.088196	38249908

前處理：切分「自變數」與「應變數」

```
1  # In[] Preprocessing: Decomposition
2  X = dataset.loc[:, ["High", "Low", "Open", "Close", "Volume"]]
3  Y = dataset.loc[:, ["Close"]]
```

自變數 X

Date	High	Low	Open	Close	Volume
2018-01-02 00:00:00	232.5	231	231.5	232.5	1.80553e+07
2018-01-03 00:00:00	238	235.5	236	237	2.93081e+07
2018-01-04 00:00:00	240	236.5	240	239.5	2.90966e+07
2018-01-05 00:00:00	240	238	240	240	2.24383e+07
2018-01-08 00:00:00	242.5	240.5	242	242	2.02337e+07

應變數 Y

Date	Close
2018-01-02 00:00:00	232.5
2018-01-03 00:00:00	237
2018-01-04 00:00:00	239.5
2018-01-05 00:00:00	240
2018-01-08 00:00:00	242

小提示：將來可以放不同**自變數**，（如：20 日平均線值）增加**準度**。

也可以預測「**收盤價**」以外的數值。

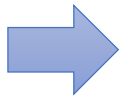
前處理：特徵縮放（MinMaxScaler）

```python
# In[] Preprocessing: Feature Scaling (Normalization) with MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
X_scale = scaler.fit_transform(X)
Y_scale = scaler.fit_transform(Y)
```

↑ 將所有數值，等比例縮放至 0～1 之間

Date	High	Low	Open	Close	Volume
2018-01-02 00:00:00	232.5	231	231.5	232.5	1.80553e+07
2018-01-03 00:00:00	238	235.5	236	237	2.93081e+07
2018-01-04 00:00:00	240	236.5	240	239.5	2.90966e+07
2018-01-05 00:00:00	240	238	240	240	2.24383e+07
2018-01-08 00:00:00	242.5	240.5	242	242	2.02337e+07



	0	1	2	3	4
0	0.0823529	0.103158	0.0810277	0.103814	0.108017
1	0.103922	0.122105	0.0988142	0.122881	0.175722
2	0.111765	0.126316	0.114625	0.133475	0.174449
3	0.111765	0.132632	0.114625	0.135593	0.134388
4	0.121569	0.143158	0.12253	0.144068	0.121124

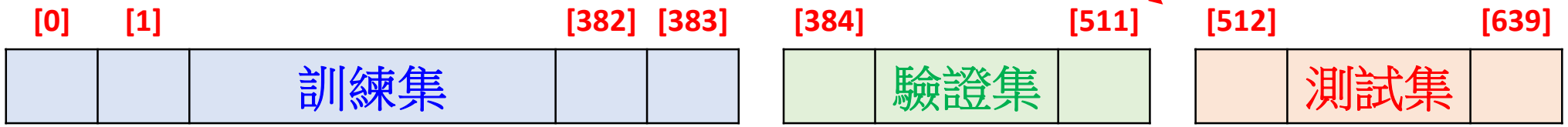
前處理：切分訓練、驗證、測試集

```
1  # In[] Preprocessing: Split Training & Testing Data
2  X_train, X_val, X_test = np.split(X_scale,
3                   [int(train_size * len(X_scale)), int((train_size + val_size) * len(X_scale))])
4  Y_train, Y_val, Y_test = np.split(Y_scale,
5                   [int(train_size * len(Y_scale)), int((train_size + val_size) * len(Y_scale))])
```

60% 640 60% + 20% 640

640 * 0.6 = 384 640 * (0.6+0.2) = 512

np.split(Y_scale, [384, 512])



前處理：製作時步資料

```python
# In[] Preprocessing: Generate Recurrent Data
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

train_set = TimeseriesGenerator(
        data=X_train,
        targets=Y_train,
        length=win_size,
        sampling_rate=sample_step,
        stride=win_moving,
        batch_size=data_batch)

val_set = TimeseriesGenerator(
        data=X_val,
        targets=Y_val,
        length=win_size,
        sampling_rate=sample_step,
        stride=win_moving,
        batch_size=data_batch)

test_set = TimeseriesGenerator(
        data=X_test,
        targets=Y_test,
        length=win_size,
        sampling_rate=sample_step,
        stride=win_moving,
        batch_size=data_batch)
```

X_train

	0	1	2	3	4
0	0.0823529	0.103158	0.0810277	0.103814	0.108017
1	0.103922	0.122105	0.0988142	0.122881	0.175722
2	0.111765	0.126316	0.114625	0.133475	0.174449
...					
59	0.143137	0.155789	0.146245	0.15678	0.208132
60	0.147059	0.155789	0.132411	0.158898	0.150746
61	0.145098	0.168421	0.140316	0.169492	0.129392
62	0.145098	0.157895	0.148221	0.15678	0.122183

Y_train

	0
0	0.103814
1	0.122881
2	0.133475
...	
59	0.15678
60	0.158898
61	0.169492
62	0.15678

train_set

5 特徵

```
[[[0.08235294, ..., 0.10801733], ... [0.14313725, ..., 0.20813164]],
]
```

30 時步

```
[[0.15889831],
]
```

收盤價

隨堂練習：資料前處理

- 請將前幾頁的程式碼寫好，並執行看看。
- 請用「草稿儲存格」，看看切分「自變數 X」與「應變數 Y」的結果是否如您所預期？
- 一樣，請觀察 X_scale 與 Y_scale，看看是否數值已經縮放到 0～1 之間？
- 觀察一下 X_train、X_val、X_test，與 Y_train、Y_val、Y_test，看看切分出來的訓練集、驗證集、測試集，是否如您所預期？
- 用 len(train_set) 看看，以每十筆資料為一批，共有幾批資料？
- 在草稿儲存格，輸入 train_set[0]，看看第一批的十筆資料，是否一切正常？

模型建造

```
1  # Create Model
2  from tensorflow.keras.models import Sequential
3  from tensorflow.keras import layers
4
5  model = Sequential()
6  model.add(layers.LSTM(units=50, return_sequences=True, input_shape=(win_size, X.shape[1])))
7  model.add(layers.Dropout(0.2))
8  model.add(layers.LSTM(units=50, return_sequences=True))
9  model.add(layers.Dropout(0.2))
10 model.add(layers.LSTM(units=50))
11 model.add(layers.Dropout(0.2))
12 model.add(layers.Dense(Y.shape[1], activation="linear"))
13
14 from tensorflow.keras.optimizers import RMSprop
15 model.compile(optimizer=RMSprop(learning_rate=1e-5), loss="mse", metrics=["mse"])
```

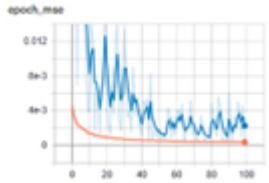
將所有「輸出序列」
（隱藏層輸出 + 狀態）
全數輸出至下一層

樣本數 時步數 特徵數

輸入層維度 = (None, 30, 5)

← 最後一層 LSTM，不必輸出「狀態」，只需要丟出「隱藏層輸出」即可

學習速率如果太大，
驗證集會在收斂終點附近震盪 →



隨堂練習：模型建造

- 請將前幾頁的程式碼寫好，並執行看看。

- 在草稿儲存格，輸入 model.summary()，看看模型是否已經成功地建造好了？

```
In [12]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 50)            11200
_________________________________________________________________
dropout (Dropout)            (None, 60, 50)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 60, 50)            20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 51,651
Trainable params: 51,651
Non-trainable params: 0
_________________________________________________________________
```

模型校正

- 原始碼

TensorBoard 設定

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

# Create TensorBoard log directory
import os
from datetime import datetime
from tensorflow.keras.callbacks import TensorBoard

logdir = os.path.join("logs", datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)

# Start the TensorBoard
%tensorboard --logdir logs
```

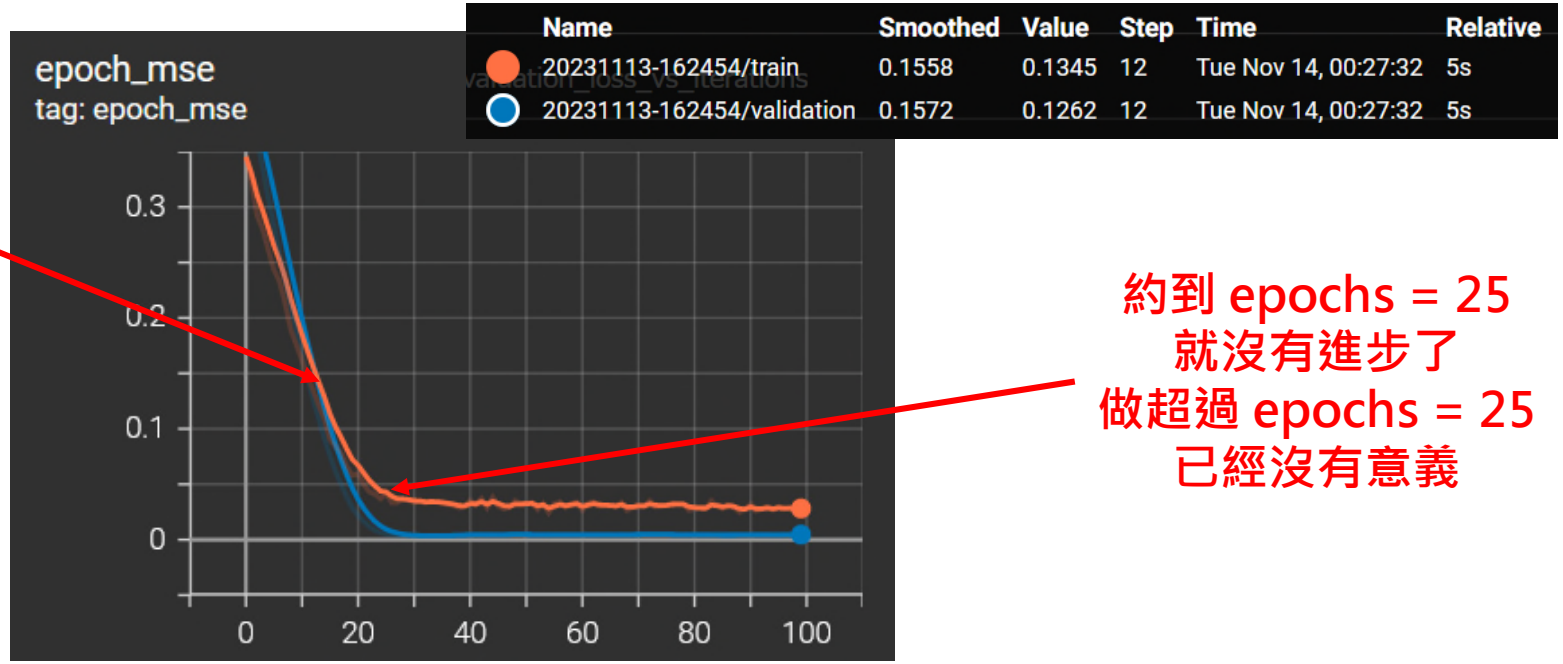
模型訓練（含 TensorBoard 數據寫入）

```
# Train Model
model.fit(train_set, validation_data=val_set, epochs=100, callbacks=[tensorboard_callback])
```

模型校正

- 使用 TensorBoard 觀看結果

在 epochs = 12
左右就開始交叉



Name	Smoothed	Value	Step	Time	Relative
20231113-162454/train	0.1558	0.1345	12	Tue Nov 14, 00:27:32	5s
20231113-162454/validation	0.1572	0.1262	12	Tue Nov 14, 00:27:32	5s

約到 epochs = 25
就沒有進步了
做超過 epochs = 25
已經沒有意義

模型訓練

- 使用 epochs = 12 重新訓練模型

```
1  # Train Model (After Correction)
2  model.fit(train_set, validation_data=val_set, epochs=12)
```

```
Epoch 1/12
18/18 [==============================] - 0s 12ms/step - loss: 0.0284 - mse: 0.0284 - val_loss: 0.0045 - val_mse: 0.0045
Epoch 2/12
18/18 [==============================] - 0s 11ms/step - loss: 0.0248 - mse: 0.0248 - val_loss: 0.0044 - val_mse: 0.0044
Epoch 3/12
18/18 [==============================] - 0s 12ms/step - loss: 0.0252 - mse: 0.0252 - val_loss: 0.0044 - val_mse: 0.0044
....
Epoch 11/12
18/18 [==============================] - 0s 12ms/step - loss: 0.0241 - mse: 0.0241 - val_loss: 0.0043 - val_mse: 0.0043
Epoch 12/12
18/18 [==============================] - 0s 15ms/step - loss: 0.0245 - mse: 0.0245 - val_loss: 0.0045 - val_mse: 0.0045
```

隨堂練習：模型校正＆訓練

- 請將前幾頁的程式碼寫好，並執行看看。
- 請啟動 TensorBoard，觀看模型 Loss 與 MSE 的值。
- 模型調校（１）：看看 TensorBoard 的結果，調整您 fit() 內的 epochs 數
- 模型調校（２）：如果驗證集出現上下震盪情況，代表學習速率太快，抵達終點時老是衝過頭，請試著把 compile() 內的 RMSprop 學習速率調小。
- 調整完畢後，拿掉 TensorBoard 相關程式碼，重新跑一次模型訓練流程。

模型效能評估

```
# In[] Model Evaluation
test_loss, test_mse = model.evaluate(test_set)
print("Loss of Test Set:", test_loss)
print("MSE of Test Set:", test_mse)
```

```
2/2 [==============================] - 0s 10ms/step - loss: 0.0039 - mse: 0.0039
Loss of Test Set: 0.00387108745053410 53
MSE of Test Set: 0.00387108745053410 53
```

測試結果視覺化

- 程式碼（1）：預測值 vs. 真實值之準備

dataset - DataFrame

Date	High	Low
2020-08-21 00:00:00	427	420
2020-08-24 00:00:00	434.5	425
2020-08-25 00:00:00	436	430.5
2020-08-26 00:00:00	442	435
2020-08-27 00:00:00	453.5	444
2020-08-28 00:00:00	440.5	431
2020-08-31 00:00:00	439.5	426.5

取出等量的列標籤

```
1  # In[] Model Prediction
2
3  # Get the Predict Price (with date)
4  Y_pred = model.predict(test_set)  ← 用測試集預測出收盤價
5  Y_pred_price = scaler.inverse_transform(Y_pred) ← 解除特徵縮放，變回原來的值
6  Y_pred_price = pd.DataFrame(data=Y_pred_price)
7  Y_pred_price.columns = ["收盤價"]
8  Y_pred_price.index = dataset.index[-len(Y_pred_price):].strftime("%Y-%m-%d").tolist()
9
10 # Get the Real Price (with date)
11 Y_real_price = dataset.iloc[-len(Y_pred_price):]["Close"].to_frame()
12 Y_real_price.columns = ["收盤價"]
13 Y_real_price.index = dataset.index[-len(Y_pred_price):].strftime("%Y-%m-%d").tolist()
```

變成 DataFrame（方便製圖時有 X/Y 軸標籤）

由後往前取出列名做為日期　只留下日期，去掉時間　轉成串列

取出等量的收盤價真實值　Series 轉 DataFrame

測試結果視覺化

- 程式碼（2）：圖形繪製

```
15  # Plot the predict vs. real price
16  import matplotlib.ticker as ticker
17
18  fig, ax = plt.subplots(1, 1)
19
20  ax.plot(Y_pred_price, color="blue", label="預測值")
21  ax.plot(Y_real_price, color="red", label="真實值")
22
23  tick_spacing = 5
24  ax.xaxis.set_major_locator(ticker.MultipleLocator(tick_spacing))
25  fig.autofmt_xdate()
26
27  plt.title("{} 收盤價盲測結果".format(stockName))
28  plt.xlabel("日期")
29  plt.ylabel("收盤價")
30  plt.legend(loc="best")
31  plt.show()
```

設定一張含有 1x1 子圖的圖片
（傳回 fig 圖片、ax 軸線資料物件）

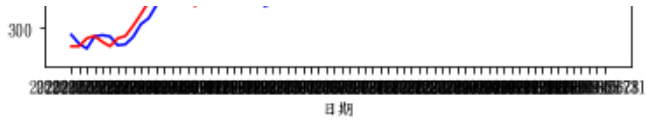
餵給 ax 軸線資料

← 設定 X 軸標籤取樣頻率為 5

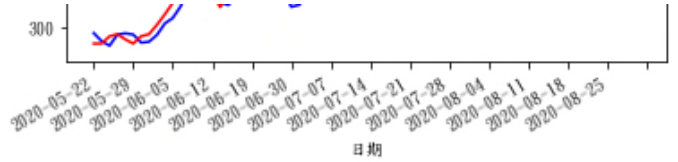
← 設定 X 軸日期標籤「自動旋轉」美化

圖片標題、X/Y 軸名稱、圖例設定

X 軸標籤未設定前



X 軸標籤設定後



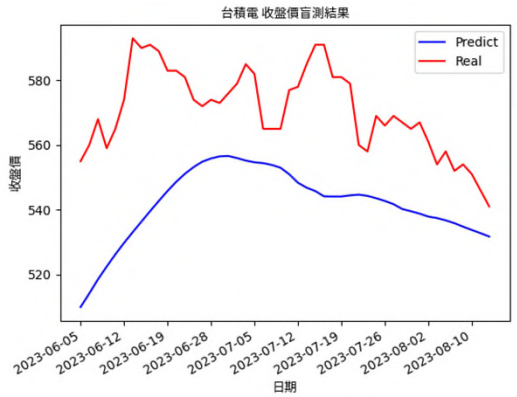
隨堂練習：結果視覺化

- 請將前幾頁的程式碼寫好，並執行看看。
- 請用草稿儲存格，看看 Y_pred_price 與 Y_real_price 的樣貌。

Index	收盤價
2020-05-22	297.04602
2020-05-25	293.16827
2020-05-26	290.98642

Index	收盤價
2020-05-22	292
2020-05-25	292
2020-05-26	295.5

- 您能跑出測試集「預測值 vs. 真實值」的對照折線圖嗎？



預測「明日漲跌」

- 做出「明日指數」的預測

dataset - DataFrame

Date	High	Low
2020-08-21 00:00:00	427	420
2020-08-24 00:00:00	434.5	425
2020-08-25 00:00:00	436	430.5
2020-08-26 00:00:00	442	435
2020-08-27 00:00:00	453.5	444
2020-08-28 00:00:00	440.5	431
2020-08-31 00:00:00	439.5	426.5

↓ 抓資料中的最後一個交易日
（並且轉成 Python 的 datetime 物件）

```
1  # In[] Predict Tomorrow's Adjusted Closed Price
2  # Get Trade Day
3  import datetime
4  this_trade_day = dataset.index[-1].to_pydatetime()
5  next_trade_day = this_trade_day + datetime.timedelta(days=1)
6  if next_trade_day.isoweekday() in set((6, 7)):
7      next_trade_day += datetime.timedelta(days=8-next_trade_day.isoweekday())
8
9  # Show Predict Price
10 lookback_data = [[]]
11 for i in range(-win_size, 0):
12     lookback_data[0].append(X_scale[i].tolist())
13
14 lookback_data = np.reshape(lookback_data, (1, win_size, X_scale.shape[1]))
15 tomorrow_pred = scaler.inverse_transform(model.predict(lookback_data))
```

← 加一天

如果是週六日
加到週一為止

ISO Weekday：週一=1、週二=2...週日=7

如果是週六：8－6＝2（再加兩天）
如果是週日：8－7＝1（再加一天）

-30

從最後一筆，
往回抓 30 筆

調整維度

（1 樣本, 30 時步, 5 特徵）

反向「特徵縮放」

預測「明日指數」

預測「明日漲跌」

- 印出「明日漲跌」結果，並與「真實值」對照

```
16  print()
17  print("{}預測收盤價 ----------".format(stockName))
18  print("最後一日 ({}) :{:.2f}".format(this_trade_day.strftime("%Y/%m/%d"), Y_pred_price.iloc[-1][0]))
19  print("次交易日 ({}) :{:.2f}".format(next_trade_day.strftime("%Y/%m/%d"), tomorrow_pred[0][0]))
20  print("預測漲跌:{:+.2f}".format(tomorrow_pred[0][0] - Y_pred_price.iloc[-1][0]))
21
22  # Show Real Price
23  print()
24  print("{}真實收盤價 ----------".format(stockName))
25  print("最後一日 ({}) :{:.2f}".format(this_trade_day.strftime("%Y/%m/%d"), dataset.iloc[-1]["Close"]))
26
27  if next_trade_day.date() < datetime.date.today():
28      next_trade_day_end = next_trade_day + datetime.timedelta(days=1)
29      tomorrow_real = yf.download(stockQuery,
30                                  start=next_trade_day.date().strftime("%Y-%m-%d"),
31                                  end=next_trade_day_end.date().strftime("%Y-%m-%d"))
32      print("次交易日 ({}) :{:.2f}".format(next_trade_day.strftime("%Y/%m/%d"), tomorrow_real.iloc[0]["Close"]))
33      print("真實漲跌:{:+.2f}".format(tomorrow_real.iloc[0]["Close"] - dataset.iloc[-1]["Close"]))
```

↓ 印出個股名稱

↓ 印出最後交易日日期

↓ 印出最後交易日預測價格

↑ 印出次一交易日預測價格

↑ 印出「次一交易日預測價格」vs.「最後交易日預測價格」漲跌幅

↓ 印出最後交易日真實價格

↓ 如果「次一交易日」比「今天」要小 (i.e. 證交所已經有資料)

抓「次一交易日」的真實價格

↑ 印出「次一交易日真實價格」vs.「最後交易日真實價格」漲跌幅

隨堂練習：預測「明日漲跌」

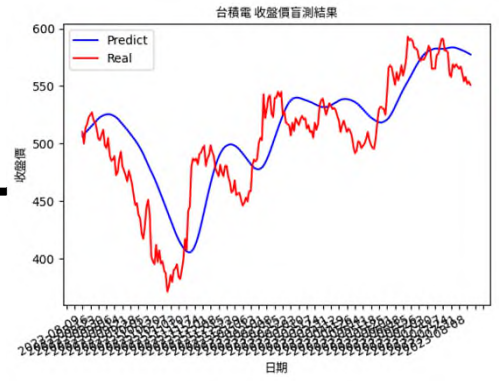
- 請將前幾頁的程式碼寫好，並執行看看。

```
台積電預測收盤價 ----------
最後一日（2023/08/14）：531.69
次交易日（2023/08/15）：530.31
預測漲跌：-1.39

台積電真實收盤價 ----------
最後一日（2023/08/14）：541.00
[**********100%%**********]  1 of 1 completed
次交易日（2023/08/15）：542.00
真實漲跌：+1.00
```

- 假設今天是 2023/08/11（五），你想知道台積電次一交易日（2023/08/14（一））的「預測股價」與「預測漲跌」。以 30 天為時步，回測至 2018/1/1。請重新執行程式，並輸入下列數值：

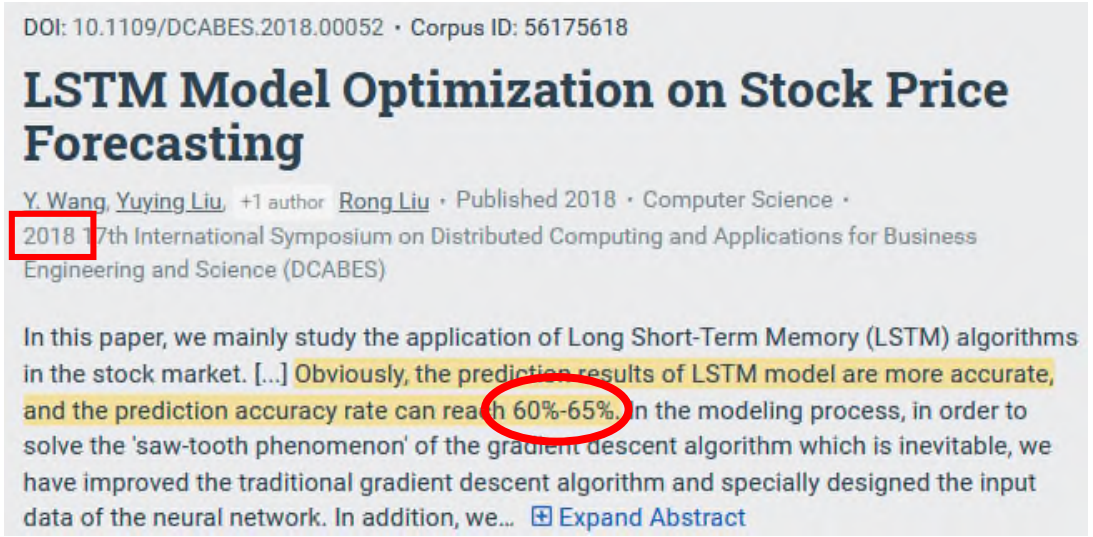
```
請輸入台灣股票名稱、或代號：台積電
請輸入查詢起始日期：Jan 1 2018
請輸入查詢截止日期：Aug 11 2023
```




```
台積電預測收盤價 ----------
最後一日（2023/08/10）：577.43
次交易日（2023/08/11）：576.46
預測漲跌：-0.97

台積電真實收盤價 ----------
最後一日（2023/08/10）：551.00
[**********100%%*****
次交易日（2023/08/11）：546.00
真實漲跌：-5.00
```

小節整理：以 LSTM 預測股價



DOI: 10.1109/DCABES.2018.00052 · Corpus ID: 56175618

LSTM Model Optimization on Stock Price Forecasting

Y. Wang, Yuying Liu, +1 author Rong Liu · Published 2018 · Computer Science ·
2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)

In this paper, we mainly study the application of Long Short-Term Memory (LSTM) algorithms in the stock market. [...] Obviously, the prediction results of LSTM model are more accurate, and the prediction accuracy rate can reach 60%-65%. In the modeling process, in order to solve the 'saw-tooth phenomenon' of the gradient descent algorithm which is inevitable, we have improved the traditional gradient descent algorithm and specially designed the input data of the neural network. In addition, we... ⊞ Expand Abstract

- **LSTM 預測股價準度**
  - 60% ~ 65%
- **以下列方法支撐的個股，預測準度較高：**
  - **技術面**：受歡迎的新製程、產品、服務 → 上漲
- **以下列兩種方法支撐的個股，預測準度較低：**
  - **基本面**：萬年小漲小跌，股利不錯。
  - **消息面**：靠消息大漲大跌，炒股最愛。
- **警語**
  - **請勿**靠著本程式**押身家**，後果自負！

閘道循環單元簡介

Introduction to Gated Recurrent Unit (GRU)

何謂 GRU

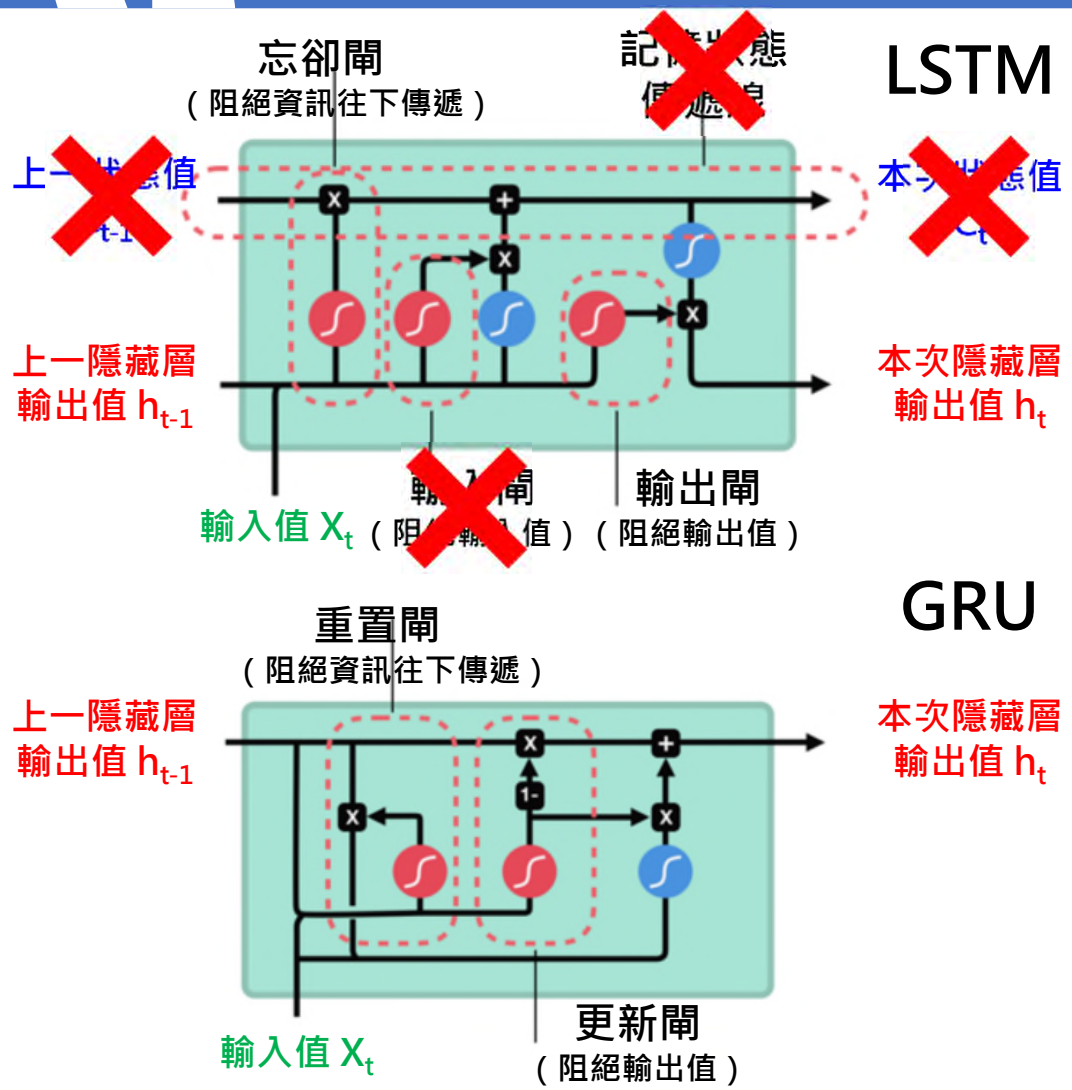
Gated Recurrent Unit
（閘道循環單元）

韓裔科學家　KyungHyun Cho

2014, 博士後研究員
加拿大蒙特婁大學 約書亞·班吉歐實驗室
（現任 紐約大學（NYU）助理教授）

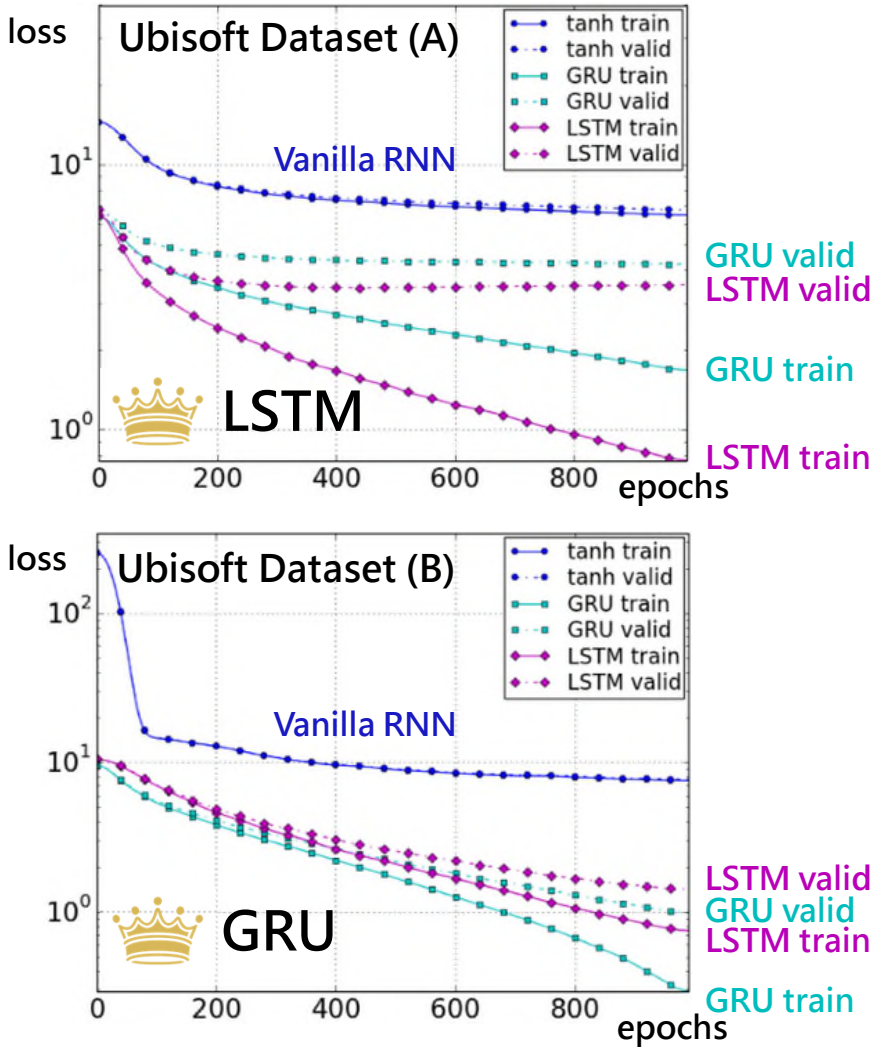
目的：
- 減少 LSTM 計算參數，加快執行。
- 減少 LSTM 佔用的記憶體。

GRU 與 LSTM 之比較



- **取消「記憶狀態傳遞線」**
  - 僅用「上一隱藏層輸出」迴授代替。
  - 記憶體**佔用變少**。
- **保留「忘卻閘」（Forget Gate）**
  - 改名叫「**重置閘**」（Reset Gate）。
- **取消「輸入閘」（Input Gate）**
  - 資料一律讓它進來。
  - 不需要的話，不要放出去就好了。
  - 參數變少，**計算變快**。
- **保留「輸出閘」（Output Gate）**
  - 改名叫「**更新閘**」（Update Gate）

GRU 表現如何？



- **訓練速度**
  - GRU 優於 LSTM
- **記憶體用量**
  - GRU 優於 LSTM
- **正確率 / 損失率**
  - 互有高下
  - 一般而言，LSTM 會稍優

小節整理：何時該用 GRU

- **時步長度不長（<10）**
  - **在意**訓練速度：Vanilla RNN > GRU > LSTM
  - **不在意**訓練速度：LSTM > GRU > Vanilla RNN

- **時步長度很長（≧10）**
  - **在意**訓練速度，願意犧牲一點正確率：GRU > LSTM
  - **不在意**訓練速度，以高正確率為目標：LSTM > GRU

閘道循環單元範例
（含自然語言處理講解）

The Example of
Gated Recurrent Unit (GRU)

範例完整原始碼：
https://lurl.cc/Lkq7gR

自然語言處理略說

一系列的對話

需對先前對話「有記憶」！

今天天氣如何？　不錯啊！晴天喔！　那明天呢？

需要記憶 ⇒ 特別適合 RNN

如何做「自然語言處理」？

自然語言「前處理」

斷詞 → 數位化 → 序列對齊 → 編碼 → 模型訓練

['I', 'love', 'jogging']
['and', 'you']

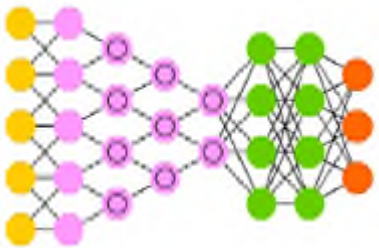
['我', '愛', '跑步']
['你', '呢']

[1, 2, 3]
[4, 5]

[1, 2, 3]
[0, 4, 5]

[[0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0],
[0, 0, 0, 1, 0, 0]]

[[1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 1]]



「斷詞」（Tokenization）

- 英文斷詞

```
1  # 英文斷詞                                          ↓ 載入英文斷詞套件
2  from tensorflow.keras.preprocessing.text import text_to_word_sequence
3  print("英文斷詞：", text_to_word_sequence("I love jogging, and you?"))
```

英文斷詞： ['i', 'love', 'jogging', 'and', 'you']

（預設全轉「小寫」）

「斷詞」（Tokenization）

- 中文斷詞（環境設定）

```
1  # Install jieba (結巴)
2  !pip install jieba
3
4  # Get the Tokenization Dictionary for Traditional Chinese
5  import os
6  Dictionary_File = 'dict.txt.big'
7
8  if not os.path.isfile(Dictionary_File):
9      os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dictionary_File)
10
11 # Get the Stop Words File for Traditional Chinese
12 StopWords_File = "stopWords_big5.txt"
13
14 if not os.path.isfile(StopWords_File):
15     os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + StopWords_File)
```

← 安裝中文斷詞套件

← 下載「簡繁對照」與「台灣地區用語」詞彙表

官網更新檔： https://bit.ly/3P5Ka6S

龍鳳餅 3 nr
龙凤饼 3 nr
龐巴迪 6 nr
庞巴迪 6 nr
龐統之 3 nr
庞统之 3 nr

← 下載「停止詞」詞彙表（標點、虛字、語助詞...）

#
%
不然
乃至於

「斷詞」（Tokenization）

- 中文斷詞

```
1  import jieba  ← 載入中文斷詞套件
2
3  # Set Dictionary for Traditional Chinese  ← 載入「自定義詞彙表」
4  # jieba.set_dictionary(Dictionary_File)
5                                              （可有可無，可增加正確率）
6  # Tokenization
7  result = list(jieba.cut("我喜歡跑步，你呢？")) ← .cut() 預設傳回迭代器
8  print("中文斷詞（有標點）：", result)
9
10 # Remove Stop Words from Set
11 stopWords = set("$!&#%\()+-*/_,.   ?:;'\"<=>^`|~[]{}’0123456789?_“”、。《》！，：；？「」（） ")  ← 這一串只能拷貝貼上
12 print("中文斷詞（無標點）：", [word for word in result if word not in stopWords])
13
14 # Remove Stop Words from Files
15 stopWords = set()
16 with open(StopWords_File, "rt", encoding="utf-8") as f:
17   for line in f:
18     line = line.strip() # Remove trailing \n
19     stopWords.add(line)
20 print("中文斷詞（更精簡）：", [word for word in result if word not in stopWords])
```

中文斷詞（有標點）：['我', '喜歡', '跑步', '，', '你', '呢', '？']

中文斷詞（無標點）：['我', '喜歡', '跑步', '你', '呢']

利用「停止詞」檔案去除冗餘文字

中文斷詞（更精簡）：['喜歡', '跑步']

隨堂練習：斷詞

- 請用下列程式碼，試著針對英文「斷詞」：

```
from tensorflow.keras.preprocessing.text import text_to_word_sequence
print("英文斷詞：", text_to_word_sequence("I love jogging, and you?"))
```

- 請參考下列程式碼，試著對中文「斷詞」：

```
# Install jieba (結巴)
!pip install jieba

# Get the Tokenization Dictionary for Traditional Chinese
import os
Dictionary_File = 'dict.txt.big'

if not os.path.isfile(Dictionary_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dictionary_File)

# Get the Stop Words File for Traditional Chinese
StopWords_File = "stopWords_big5.txt"

if not os.path.isfile(StopWords_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + StopWords_File)
```

隨堂練習：斷詞

- 請參考下列程式碼，試著對中文「斷詞」（續）：

```
1  import jieba
2
3  # Set Dictionary for Traditional Chinese
4  # jieba.set_dictionary(Dictionary_File)
5
6  # Tokenization
7  result = list(jieba.cut("我喜歡跑步, 你呢？"))
8  print("中文斷詞（有標點）：", result)
9
10 # Remove Stop Words from Set
11 stopWords = set("$!&#%\()+-*/_,.   ?:;'\"<=>^`|~[]{}’0123456789?_“”、。《》！，　：；？「」（） ")
12 print("中文斷詞（無標點）：", [word for word in result if word not in stopWords])
13
14 # Remove Stop Words from Files
15 stopWords = set()
16 with open(StopWords_File, "rt", encoding="utf-8") as f:
17   for line in f:
18     line = line.strip() # Remove trailing \n
19     stopWords.add(line)
20 print("中文斷詞（更精簡）：", [word for word in result if word not in stopWords])
```

文字「數位化」

```
# Create a Tokenizer object
from tensorflow.keras.preprocessing.text import Tokenizer

tk = Tokenizer(
        num_words=None,
        filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
        lower=True,
        split=' ',
        char_level=False,
        oov_token='NiD'
    )
```

← 建立對照表時，是否只取詞頻最高的前幾個字？（None=不限制）

← 需要過濾掉的文字（停止詞）

← 建立出來的對照表，文字是否全轉小寫？

← 斷詞用文字

← 是否斷詞至「字母」層級（True='i', 'l', 'o' ... False='i', 'love', ...）

← Out Of Vocabulary Token。對照表內找不到時，用哪個符號標示。
（NiD = Not In Dictionary，可自由使用 NiD 以外任意文字）

文字「數位化」

```
1  # Create Mapping by Corpus
2  corpus = ["I love jogging, and you?",
3            "I love reading!"]
4  tk.fit_on_texts(corpus)
5
6  # Show the Mapping Table
7  print(tk.word_index)     # WORD vs. NUMBER
8  print(tk.index_word)     # NUMBER vs. WORD
9
10 # Test for Mapping Text into Sequence
11 input_text = ["I love jogging!",
12              "and I love reading, too!"]
13
14 seq = tk.texts_to_sequences(input_text)
15 print(seq)
16
17 # Test for Mapping Sequence into Text
18 text = tk.sequences_to_texts(seq)
19 print(text)
```

← 利用「語料庫（Corpus）」建立對照表

← 秀出對照表

{'NiD': 1, 'i': 2, 'love': 3, 'jogging': 4, 'and': 5, 'you': 6 , 'reading': 7}

{1: 'NiD', 2: 'i', 3: 'love', 4: 'jogging', 5: 'and', 6: 'you', 7: 'reading'}

← 丟入一句話做「數位化」

[[2, 3, 4] , [5, 2, 3, 7, 1]]　（2=i, 3=love, 4=jogging, ...,1=NiD）

← 將數字轉回文字

['i love jogging', 'and i love reading NiD']　（不在對照表中的字，無法轉回來）

隨堂練習：文字數位化

- 請執行前兩頁的程式碼，了解如何做「文字數位化」：

```python
# Create a Tokenizer object
from tensorflow.keras.preprocessing.text import Tokenizer

tk = Tokenizer(
        num_words=None,
        filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
        lower=True,
        split=' ',
        char_level=False,
        oov_token='NiD'
    )
```

```python
# Create Mapping by Corpus
corpus = ["I love jogging, and you?",
          "I love reading!"]
tk.fit_on_texts(corpus)

# Show the Mapping Table
print(tk.word_index)    # WORD vs. NUMBER
print(tk.index_word)    # NUMBER vs. WORD

# Test for Mapping Text into Sequence
input_text = ["I love jogging!",
              "and I love reading, too!"]

seq = tk.texts_to_sequences(input_text)
print(seq)

# Test for Mapping Sequence into Text
text = tk.sequences_to_texts(seq)
print(text)
```

序列對齊（Sequence Alignment）

- **什麼是「序列對齊」**
  - 將所有「**數位化**」過的句子，**截長補短**成**相同長度**。
- **為何要「序列對齊」**
  - 方便**輸入**神經網路的「**輸入層**」。
- **如何「序列對齊」**

```
[[2, 3, 4],
 [5, 2, 3, 7, 1]]
```

```
[[0, 0, 2, 3, 4]
 [5, 2, 3, 7, 1]]
```

```
1  # Create a Sequence Padding Object
2  from tensorflow.keras.preprocessing.sequence import pad_sequences
3
4  padded_seq = pad_sequences(
5          sequences=seq, ← 要被對齊的序列
6          maxlen=5,      ← 一律調整成五個字
7          dtype="int32", ← 使用整數做為輸出型態
8          padding="pre", ← 要補字的話，補在前面
9          truncating="post",  ← 句子太長，要砍字的話，砍後面
10         value=0 ← 不足一律補 0
11     )
12
13 print(padded_seq)
```

1	“NiD”
2	“i”
3	“love”
4	“jogging”
5	“and”
6	“you”
7	“reading”

隨堂練習：序列對齊

- 請執行前一頁的程式碼，了解如何做「序列對齊」：

```
1  # Create a Sequence Padding Object
2  from tensorflow.keras.preprocessing.sequence import pad_sequences
3
4  padded_seq = pad_sequences(
5          sequences=seq,
6          maxlen=5,
7          dtype="int32",
8          padding="pre",
9          truncating="post",
10         value=0
11     )
12
13 print(padded_seq)
```

```
[[0, 0, 2, 3, 4]
 [5, 2, 3, 7, 1]]
```

編碼（Encoding）

太耗
記憶體

- 法一：獨熱編碼法（One-Hot Encoding）

```
[[0, 0, 2, 3, 4]
 [5, 2, 3, 7, 1]]
```

0	[1, 0, 0, 0, 0, 0, 0, 0]
1	[0, 1, 0, 0, 0, 0, 0, 0]
2	[0, 0, 1, 0, 0, 0, 0, 0]
3	[0, 0, 0, 1, 0, 0, 0, 0]
4	[0, 0, 0, 0, 1, 0, 0, 0]
5	[0, 0, 0, 0, 0, 1, 0, 0]
6	[0, 0, 0, 0, 0, 0, 1, 0]
7	[0, 0, 0, 0, 0, 0, 0, 1]

```
[[[1., 0., 0., 0., 0., 0., 0., 0.,]
  [1., 0., 0., 0., 0., 0., 0., 0.,]
  [0., 0., 1., 0., 0., 0., 0., 0.,]
  [0., 0., 0., 1., 0., 0., 0., 0.,]
  [0., 0., 0., 0., 1., 0., 0., 0.,]]

 [[0., 0., 0., 0., 0., 1., 0., 0.,]
  [0., 0., 1., 0., 0., 0., 0., 0.,]
  [0., 0., 0., 1., 0., 0., 0., 0.,]
  [0., 0., 0., 0., 0., 0., 0., 1.,]
  [0., 1., 0., 0., 0., 0., 0., 0.,]]]
```

作法：

```
1  from tensorflow.keras.utils import to_categorical
2  to_categorical(padded_seq)
```

編碼（Encoding）

仍然很耗記憶體

- 法二：多熱編碼法（Multi-Hot Encoding）

['i love jogging!'， 'and i love reading, too!']

1	"NiD"
2	"i"
3	"love"
4	"jogging"
5	"and"
6	"you"
7	"reading"

0	1	2	3	4	5	6	7
		"I"	"love"	"jogging"			
0	0	1	1	1	0	0	0

→ [0. 0. 1. 1. 1. 0. 0. 0.]

"NiD"

0	1	2	3	4	5	6	7
	"too"	"I"	"love"		"and"		"reading"
0	1	1	1	0	1	0	1

→ [0. 1. 1. 1. 0. 1. 0. 1.]

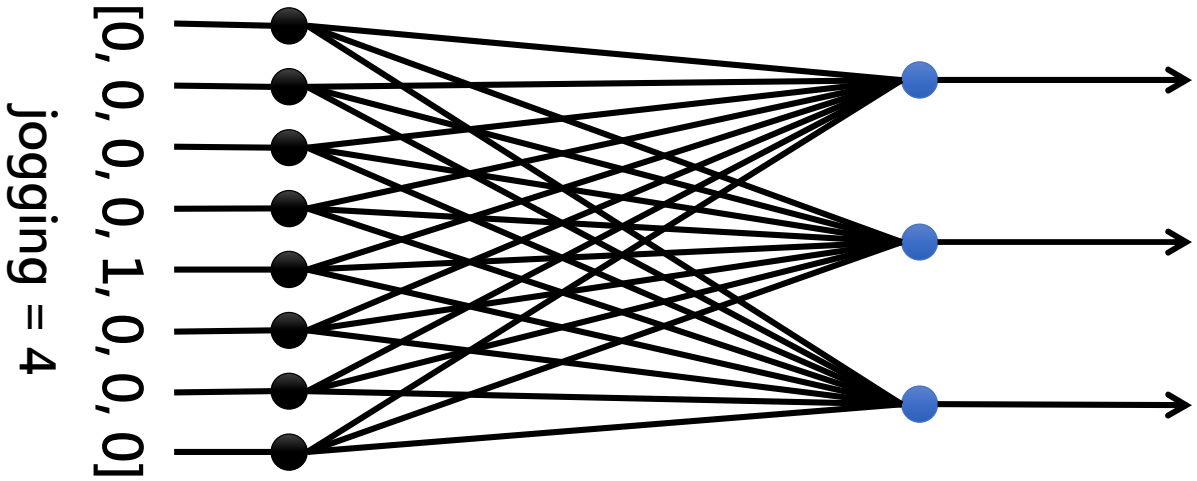
作法：

```
1  from tensorflow.keras.preprocessing.text import Tokenizer
2
3  input_text = ["I love jogging!",
4                "and I love reading, too!"]
5  print(tk.texts_to_matrix(input_text))
```

編碼（Encoding）

- 法三：詞向量嵌入法（Word Embedding）

原理：利用神經網路，將詞編碼降維後，得到一個壓縮過後的「詞向量」



以後，" jogging"

=

[-0.00180965 0.00570974 -0.01133073]

詞編碼

[0, 0, 0, 0, 1, 0, 0, 0] ⊙

權重 W
（可自行習得）

```
[[ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [-0.00180965  0.00570974 -0.01133073]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]]
```

= [-0.00180965 0.00570974 -0.01133073]

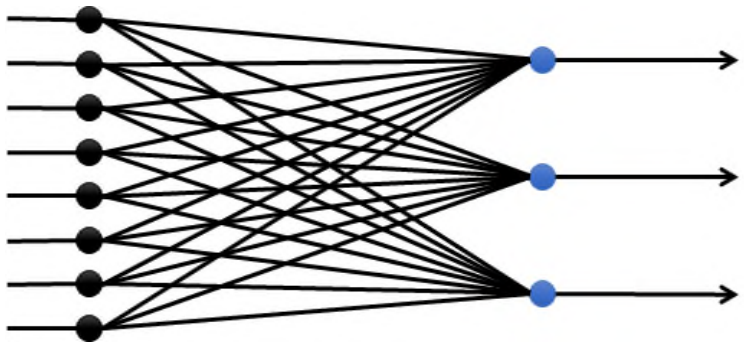
詞向量

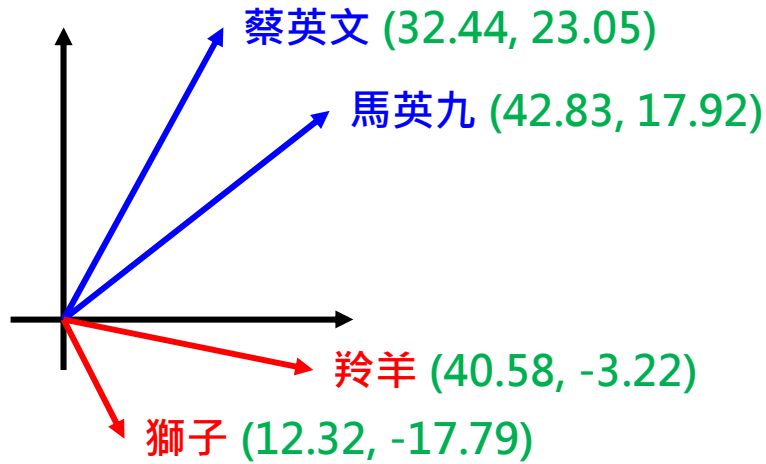
1x8　　8x3　　1x3

編碼（Encoding）

- **詞向量嵌入法**更厲害的地方

詞向量的**歐幾里得距離**接近 = 詞的**語意**接近





作法：

```python
import tensorflow as tf
from tensorflow.keras import layers

padded_seq = [[0, 0, 2, 3, 4], [5, 2, 3, 7, 1]]

emb = layers.Embedding(8, 3)
result = emb(tf.constant(padded_seq))
print(result.numpy())
```

```
[[[ 0.03134513 -0.02481314  0.04111188]
  [ 0.03134513 -0.02481314  0.04111188]
  [ 0.01575652 -0.04276028  0.01101475]
  [ 0.03992618  0.0463698  -0.01132149]
  [ 0.00481454 -0.01783461 -0.02836373]]

 [[ 0.04684695 -0.04968093 -0.04310885]
  [ 0.01575652 -0.04276028  0.01101475]
  [ 0.03992618  0.0463698  -0.01132149]
  [-0.0261304  -0.00135983  0.02886284]
  [ 0.02545213 -0.01250492  0.04925909]]]
```

隨堂練習：編碼

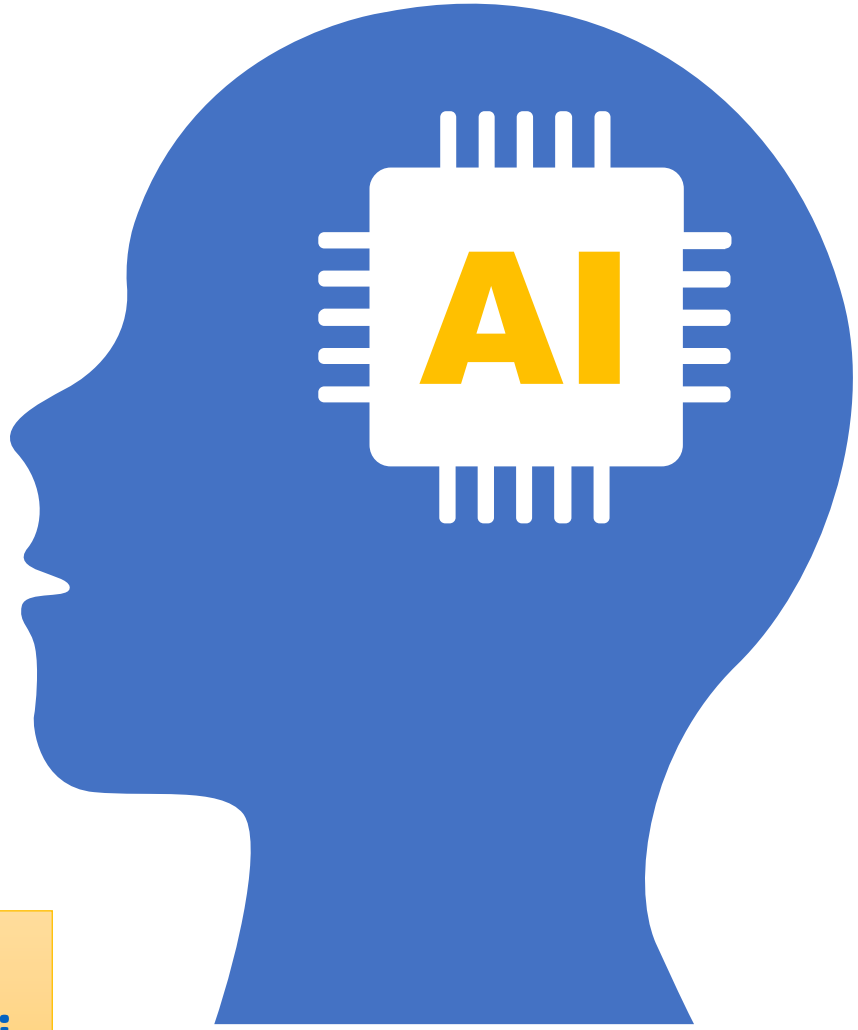
- 請執行前一頁的程式碼，了解如何做「編碼」：

```
1  # One-Hot Encoding
2  from tensorflow.keras.utils import to_categorical
3
4  print("獨熱編碼 -------------")
5  print(to_categorical(padded_seq))
6
7  # Multi-Hot Encoding
8  print("多熱編碼 -------------")
9  print(tk.texts_to_matrix(input_text))
10
11 # Word Embedding
12 import tensorflow as tf
13 from tensorflow.keras import layers
14
15 emb = layers.Embedding(8, 3)
16
17 # tf.constant(): Convert immediate values into tensor
18 result = emb(tf.constant(padded_seq))
19 print("詞向量嵌入 -------------")
20 print(result.numpy())
```

範例：IMDb 留言正負評預測

- IMDb = Internet Movie Database
- Amazon 線上串流影音平台
- 訓練集 x25000 + 測試集 x25000
- 每則留言已經用人工標註為正負評（二選一）
- 目標
  - 用 GRU 製作一個二元分類器
  - 輸入評論文字
  - 猜測該評論是正評、抑或負評

範例完整原始碼：
https://lurl.cc/gSZb4j



載入 IMDb 資料庫

```
# In[] Load IMDb Database
from tensorflow.keras.datasets import imdb

most_freq_words = 10000  ← 指定只要載入詞頻最高的前 10000 個字（不限詞頻全載入會爆！）
(X_train, Y_train), (X_test, Y_test) = imdb.load_data(num_words=most_freq_words)
```

測試 IMDb 是否載入成功

每個資料集都有 25000 筆

Shape of X_train:(25000,) Y_train:(25000,) X_test:(25000,) Y_test:(25000,)

```
1  # In[] Test if IMDb Loading Successfully
2
3  print("Shape of X_train:{} Y_train:{} X_test:{} Y_test:{}".format(
4            X_train.shape, Y_train.shape, X_test.shape, Y_test.shape))
5  print("X_train[0]:{}".format(X_train[0]))
6  print("Y_train[0]:{}".format("正評" if Y_train[0]==1 else "負評"))
```

X_train[0]:[1, 14, 22, 16, … , 19, 178, 32]
Y_train[0]:正評

← IMDb 幫我們做到「文字數位化」了

← =0：負評　=1：正評

解碼 X_train 第一條評論

```
1  # In[] Decode the first comment of X_train
2
3  # Get the Word vs. Index Dictionary
4  words_dict = imdb.get_word_index()
5
6  # Reverse the key-value (word-index) as value-key (index-word)
7  index_dict = {v:k for k, v in words_dict.items()}
8
9  # The first 3 indices were reserved for "padding", "start of sequence", "unknown"
10 # The real index of words must be (i-3)
11 msg_list = [index_dict.get(i-3, "?") for i in X_train[0]]
12 print(" ".join(msg_list))
```

← 取得「字詞 vs. 編號」的對照表

← 將 key-value 反轉，變「編號 vs. 字詞」

將字詞從「串列」黏回「字串」。

前三個編號 0, 1, 2 保留給系統用。必須 i-3 才是真正的對照表內索引編號。

如果對照表內查不到，就顯示？符號

? this film was just brilliant casting location scenery story direction everyone's really suited the part...

隨堂練習：載入 IMDb 資料庫

- 請將前幾頁的原始碼輸入，並且執行看看。
- 觀察一下，載入 IMDb 資料庫後，訓練集與測試集是否為宣稱的 25000 筆？
- 您是否能成功的解碼第一條評論？看完後，您覺得人工標註為「正評」是正確的嗎？

序列對齊

```
# In[] Sequence Alignment
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 500  ← 統一取每個評論的前 500 字（超過截斷，不足補 0）
                 避免過長的評論編碼後，對照表太大，塞爆記憶體
X_train = pad_sequences(
                    sequences=X_train,
                    maxlen=max_len,
                    padding="pre",
                    truncating="post",
                    value=0)

X_test = pad_sequences(
                    sequences=X_test,
                    maxlen=max_len,
                    padding="pre",
                    truncating="post",
                    value=0)

print("Shape of X_train:{} X_test:{}".format(X_train.shape, X_test.shape))
```

每個評論 500 字 OK

Shape of X_train:(25000, 500) X_test:(25000, 500)

隨堂練習：序列對齊

- 請將前一頁的原始碼輸入，並且執行看看。
- 觀察一下，訓練集與測試集內的每個評論，是否統一成長度 500 字的大小？

```
# In[] Sequence Alignment
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 500

X_train = pad_sequences(
                    sequences=X_train,
                    maxlen=max_len,
                    padding="pre",
                    truncating="post",
                    value=0)

X_test = pad_sequences(
                    sequences=X_test,
                    maxlen=max_len,
                    padding="pre",
                    truncating="post",
                    value=0)

print("Shape of X_train:{} X_test:{}".format(X_train.shape, X_test.shape))
```

```
Shape of X_train:(25000, 500) X_test:(25000, 500)
```

模型建造

```
# In[] Model Creation
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

model = Sequential()
model.add(layers.Embedding(input_dim=most_freq_words, output_dim=24, input_length=max_len))
model.add(layers.Dropout(0.2))
model.add(layers.GRU(20, activation="relu"))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation="sigmoid"))

model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["acc"])
model.summary()
```

詞向量從 10000 壓縮至 24 維

10000

每個評論固定 500 字長
（方便輸入神經網路訓練）

隨堂練習

- 請將前一頁的原始碼輸入，並且執行看看。

- 觀察一下，model.summary() 指令，是否成功地列出已經建造好的模型？

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 24)           240000
_________________________________________________________________
dropout_4 (Dropout)          (None, 500, 24)           0
_________________________________________________________________
gru_2 (GRU)                  (None, 20)                2760
_________________________________________________________________
dropout_5 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 21
=================================================================
Total params: 242,781
Trainable params: 242,781
Non-trainable params: 0
```

模型校正

- TensorBoard 設定與啟動

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

# Create TensorBoard log directory
import os
from datetime import datetime
from tensorflow.keras.callbacks import TensorBoard

logdir = os.path.join("logs", datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)

# Start the TensorBoard
%tensorboard --logdir logs
```

模型校正

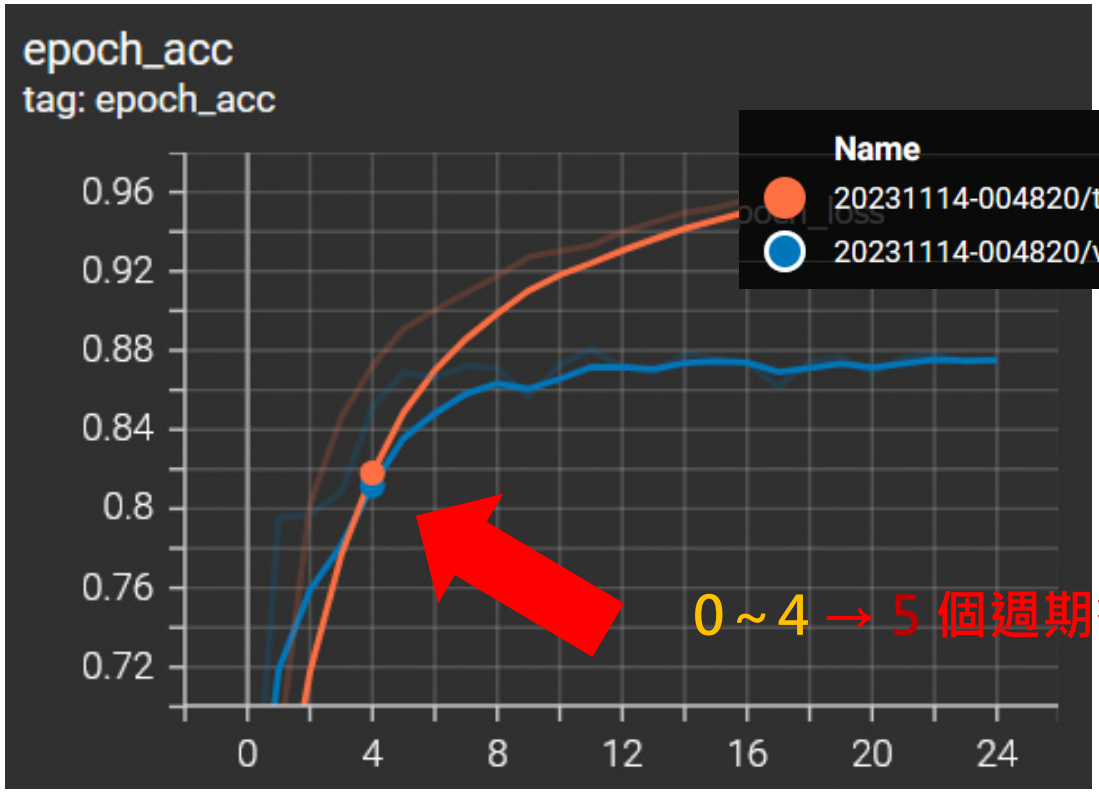
- 模型試訓練

```
1  # Train Model
2  model.fit(X_train, Y_train, validation_split=0.2, batch_size=128, epochs=25, callbacks=[tensorboard_callback])
```

```
Epoch 19/25
600/600 [==============================] - 20s 33ms/step - loss: 0.4169 - accuracy: 0.8080 - val_loss: 0.4903 - val_accuracy: 0.7655
Epoch 20/25
600/600 [==============================] - 20s 33ms/step - loss: 0.4049 - accuracy: 0.8185 - val_loss: 0.4959 - val_accuracy: 0.7615
Epoch 21/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3910 - accuracy: 0.8270 - val_loss: 0.4576 - val_accuracy: 0.7825
Epoch 22/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3753 - accuracy: 0.8297 - val_loss: 0.4716 - val_accuracy: 0.7700
Epoch 23/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3591 - accuracy: 0.8423 - val_loss: 0.4759 - val_accuracy: 0.7670
Epoch 24/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3566 - accuracy: 0.8412 - val_loss: 0.4467 - val_accuracy: 0.7870
Epoch 25/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3365 - accuracy: 0.8530 - val_loss: 0.4650 - val_accuracy: 0.7740
```

模型校正

- 找到合適的訓練週期（Epochs）



Name	Smoothed	Value	Step	Time	Relative
20231114-004820/train	0.8181	0.8723	4	Tue Nov 14, 09:06:43	13m 5s
20231114-004820/validation	0.8117	0.851	4	Tue Nov 14, 09:06:43	13m 5s

0～4 → 5 個週期很適合！

隨堂練習：模型校正＆訓練

- **請將前幾頁的程式碼寫好，並執行看看。**
  - 註：此模型之訓練十分耗時（epochs = 25 約需 1 小時）。若上課時間緊迫，請同學於下課之後自行試試看。

- **請啟動 TensorBoard，觀看模型 Loss 與 Accuracy 的值。**

- **調整完畢後，拿掉 TensorBoard 相關程式碼，重新跑一次模型訓練流程。**

模型評估

```
1  # In[] Model Evaluation
2  test_loss, test_acc = model.evaluate(X_test, Y_test)
3  print("Loss of Test Set:", test_loss)
4  print("Accuracy of Test Set:", test_acc)
```

```
25000/25000 [==============================] - 16s 624us/sample - loss: 0.2999 - acc: 0.8828
Loss of Test Set: 0.29988131525516 51
Accuracy of Test Set: 0.8828
```

正確率：88.28%

隨堂練習：模型評估

- 請將前一頁的原始碼輸入，並且執行看看。
- 您模型的正確率有多高呢？

課後作業

- **路透社新聞分類器**
  - **資料集介紹**
    - 11228 則新聞 = 8982 訓練集 + 2246 測試集
    - 共分 46 類（0～45）
  - **載入方法**

```
from tensorflow.keras.datasets import reuters

most_freq_words = 10000
(X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=most_freq_words)
```

- **要求**
  - 利用 LSTM 或是 GRU，建造一個路透社新聞分類器。
  - 請自行尋找最佳的 epochs。
  - 請用 .evaluate()，印出您模型的正確率。

本章總結

- **何時使用 RNN？**
  - 當您的模型，需要「**記憶**」**稍早輸入**，才有辦法提高正確率時。

- **選用哪種 RNN 模型？**
  - 若**時步不長**（<10）：可以用 **SimpleRNN**（Vanilla RNN）
  - 若**時步很長**（>=10）：考慮用 **LSTM**
  - 時步長，LSTM **參數太多跑不動**時：改用 **GRU**