



遞迴神經網路與序列模型

蔡炎龍 & 教研處

「版權聲明頁」

本投影片已經獲得作者授權台灣人工智慧學校得以使用於教學用途，如需取得重製權以及公開傳輸權需要透過台灣人工智慧學校取得著作人同意；如果需要修改本投影片著作，則需要取得改作權；另外，如果有需要以光碟或紙本等實體的方式傳播，則需要取得人工智慧學校散佈權。

課程內容

[講師投影片](#)
[投影片](#)
[影片播放列表](#)

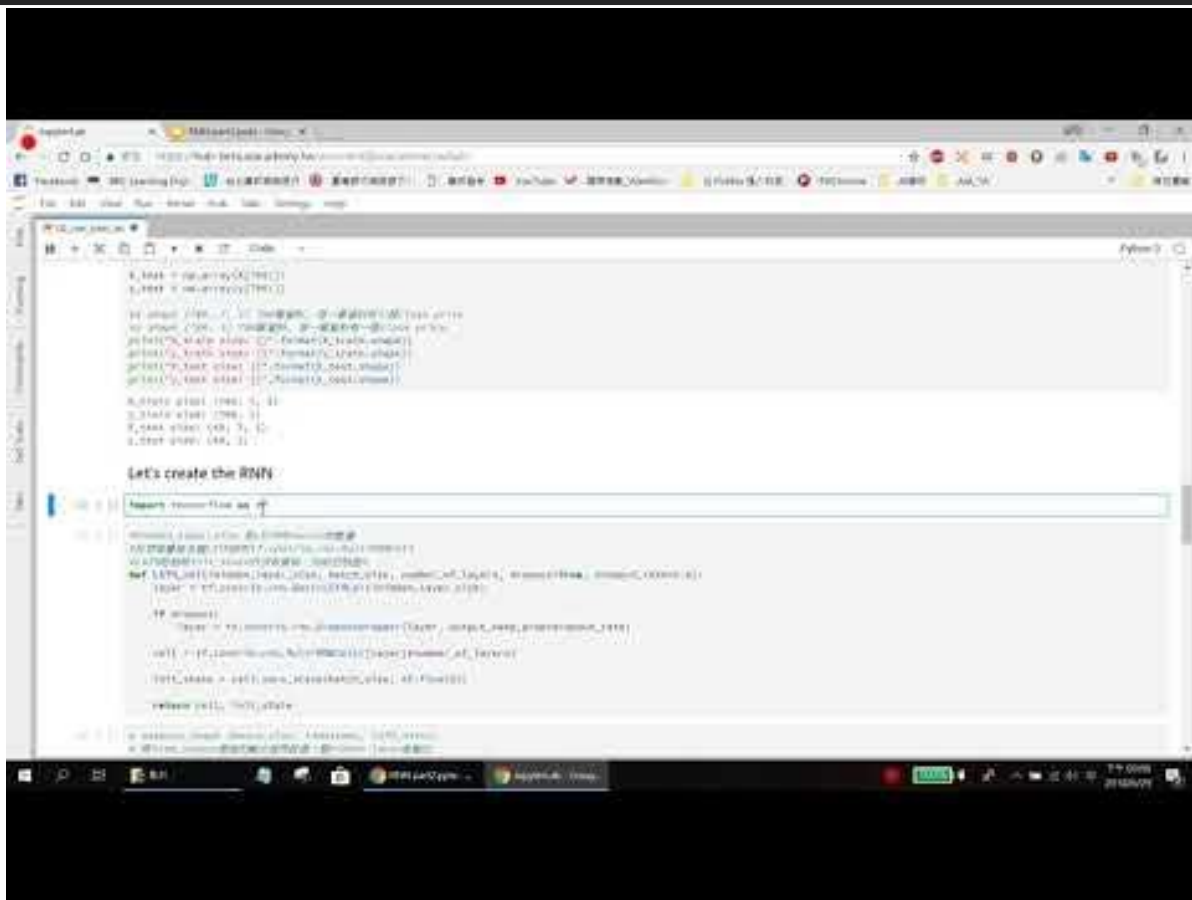
[程式碼: ~/courses-tpe/RNN/part2](#)

1. Stock prediction
(Tensorflow實作)
2. IMDB情意分析
(Keras實作)
3. Stock prediction base on
1D-CNN(Tensorflow 實作)
4. Gradient Descent概念

Stock prediction (Tensorflow 實作)

rnn 04 Tesla_stock_predict

部分程式碼有做修改, 程式請以 hub 上的為主



```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf

# Load data
data = pd.read_csv('tesla_stock_data.csv')

# Preprocess data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

# Split data into training and testing sets
train_data = data_scaled[:int(len(data_scaled) * 0.8)]
test_data = data_scaled[int(len(data_scaled) * 0.8):]

# Create the RNN model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(train_data.shape[1], train_data.shape[2])),
    LSTM(50, return_sequences=False),
    Dense(1)
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
model_checkpoint = ModelCheckpoint('tesla_stock_model.h5', save_best_only=True)
model.fit(train_data, train_data[:, -1], validation_data=(test_data, test_data[:, -1]),
          callbacks=[early_stopping, model_checkpoint], epochs=100)

# Predict on test data
predictions = model.predict(test_data[:, :-1])

# Evaluate the model
mse = model.evaluate(test_data, test_data[:, -1])
print('Mean Squared Error: ', mse)
```



處理時間序列問題

- RNN可以看成是擁有記憶的神經網路，因此拿來做時間序列的問題也是一種選項
- 以預測股價為例，我們希望可以依序輸入過去的股價紀錄，讓神經網路有所學習，找到過去與未來股價的關聯



切取資料(1)

把資料處理成適合放進RNN模型的型態

- 假設window_size = 7, 把資料切成:
X = [[第一天股價, 第二天股價, ..., 第七天股價],
[第二天股價, 第三天股價, ..., 第八天股價]...]
Y = [[第八天股價], [第九天股價]...]

```
#抓取window_size的資料作為觀察資料(x), 下一天作為預測資料(y)
def window_data(data, window_size):
    X = []
    y = []

    i = 0
    while (i + window_size) <= len(data) - 1:
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])

        i += 1
    assert len(X) == len(y)
    return X, y
```



切取資料(2)

```
X_train size: (700, 7, 1)
y_train size: (700, 1)
X_test size: (49, 7, 1)
y_test size: (49, 1)
```

- 丟進RNN的input shape應該是(batch_size, time_step, feature), 整理資料時要注意維度
- 這裡的範例只有收盤價一個 feature, 因此X_train.shape = (700, 7, 1), 第三維的1不可以省略



Gradient Clipping

```
# RNN及LSTM會有梯度爆炸的問題，因此若斜率超過+-5則clip到+-5之內
def opt_loss(logits, targets, learning_rate):

    loss = tf.reduce_mean(tf.pow(logits - targets, 2))

    #Clipping the gradient loss
    optimizer = tf.train.AdamOptimizer(learning_rate)
    gradients = optimizer.compute_gradients(loss)

    capped_gradients = [(tf.clip_by_value(grad, -5, 5), var) for grad, var in gradients if grad is not None]

    train_optimizer = optimizer.apply_gradients(capped_gradients)

    return loss, train_optimizer
```

- RNN是個有記憶的神經網路，可以看做很深的神經網路。為了避免在很深的神經網路遇到梯度爆炸，因此需要gradient clipping



預測結果



程式練習 Tesla stock price predict

- 02_rnn_lstm_stock_prediction.ipynb
 - 練習如何把 data 轉換成 LSTM 可運算的格式
 - 練習建構 LSTM network



參考連結

: <https://github.com/lucko515/tesla-stocks-prediction>

IMDB情意分析(Keras 實作)

0601讀入IMDB數據庫

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。





13

組裝與訓練



概要

前面炎龍老師的所講解關於IMDB的概念，我想沒聽過的同學們，應該有稍稍了解到了，這次我們的實作練習，嘗試以Tensorflow.keras的API來進行實作，先將每筆不定數的詞庫資料固定大小後，並且進行One Hot Encode後，為了避免輸入資料維度過為龐大，我們必須先進行Embedding的動作後，再送入LSTM進行訓練。



數據取得與前處理

```
from keras.datasets import imdb  
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = 10000)
```

透過Keras取得IMDB的資料集。

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

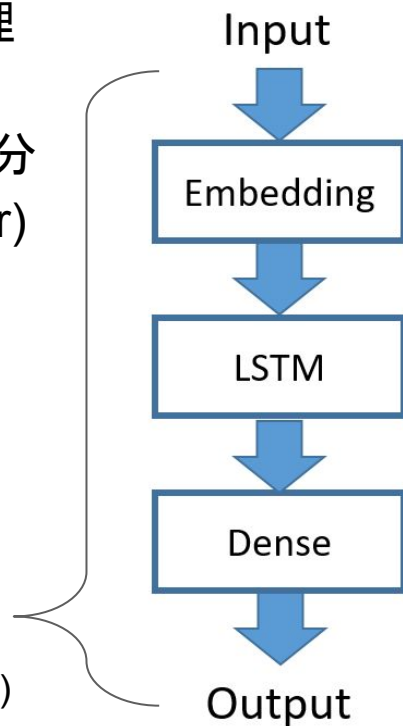
由於RNN的輸入限制，也因為評論的資料數不固定，所以我們要固定輸入資料長度後，才能進行訓練以及辨識的使用。



一圖了解架構程式架構

首先我們需要將輸入的資料，由於文字進行One hot encode處理後維度太大，所以我們先進行Embedding的動作，將輸入的維度壓縮，再來進入我們的LSTM的Model中，由於在IMDB中只需要分辨好與壞的評論結果，所以我們的Dense(Fully connected Layer)只需要輸出一個Neuron。

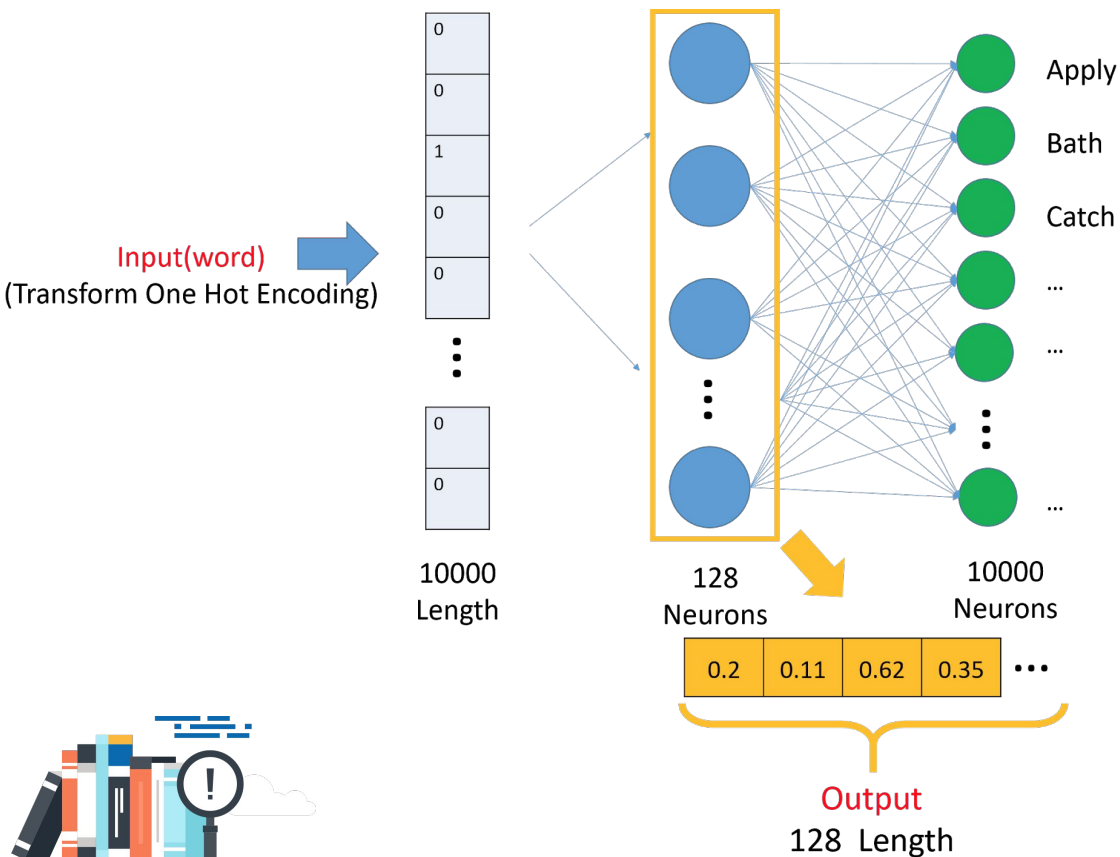
```
model = Sequential()  
model.add(Embedding(10000, 128))  
model.add(LSTM(150))  
model.add(Dense(1, activation = 'sigmoid'))
```



(小提示: Embedding的概念會在下一頁有簡單的說明)



圖解 Embedding



以文字輸入來舉例，利用類神經網路訓練，使其所對應文字 Target 達至最大值，最後每個文字的 One Hot Encode 輸入與權重矩陣運算後後得出向量則就是我們要的。

<https://keras.io/layers/embeddings/>

Stock prediction with 1D-CNN (Tensorflow 實作)

0701用CNN處理時間序列問題

時間序列的資料, 不
一定要用 RNN。

(可以考慮 CNN)



Recap. 圖像的性質 (2)

- 同一個 patterns 會出現在多個地方



Figure reference
<https://www.youtube.com/watch?v=NNgLaU2NlLM>



操盤手的日常



學習如何看圖：CNN !!



Recap. 圖像的性質 (1)

- 不需要看完所有 pixel 來定義 pattern



Recap. 圖像的性質 (2)

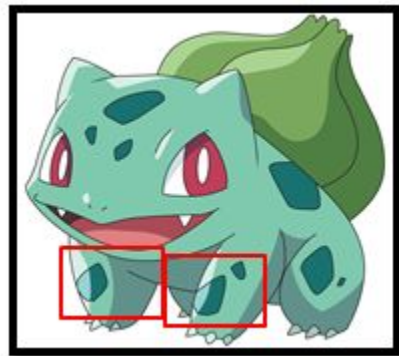
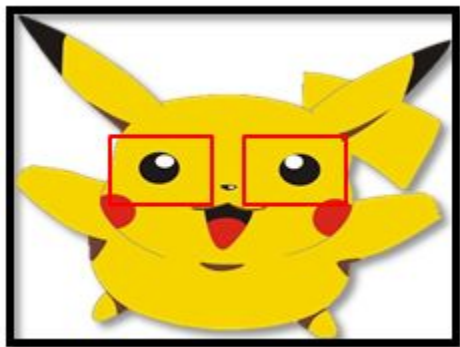


Figure reference

<https://www.youtube.com/watch?v=NN9LaU2NILM>



Recap. 圖像的性質 (3)

- 圖片大小不改變 patterns



Subsampling



時間序列資料也具備圖像的性質！



CNN之於時間序列

- CNN解決問題的特色
 - 特徵pattern出現在局部
 - 同樣pattern 重複出現
 - pattern在壓縮(subsampling)下也不太會失真
- 時間序列的特性
 - 通常與預測值最有相關性的資料是一兩天前的資料, 而並非五十天前的資料 (局部性)
 - 不少時間序列會有週期性的傾向 (重複性)
 - 把天資料化成週資料或月資料來看, 大趨勢的線條不會失真 (可壓縮性)



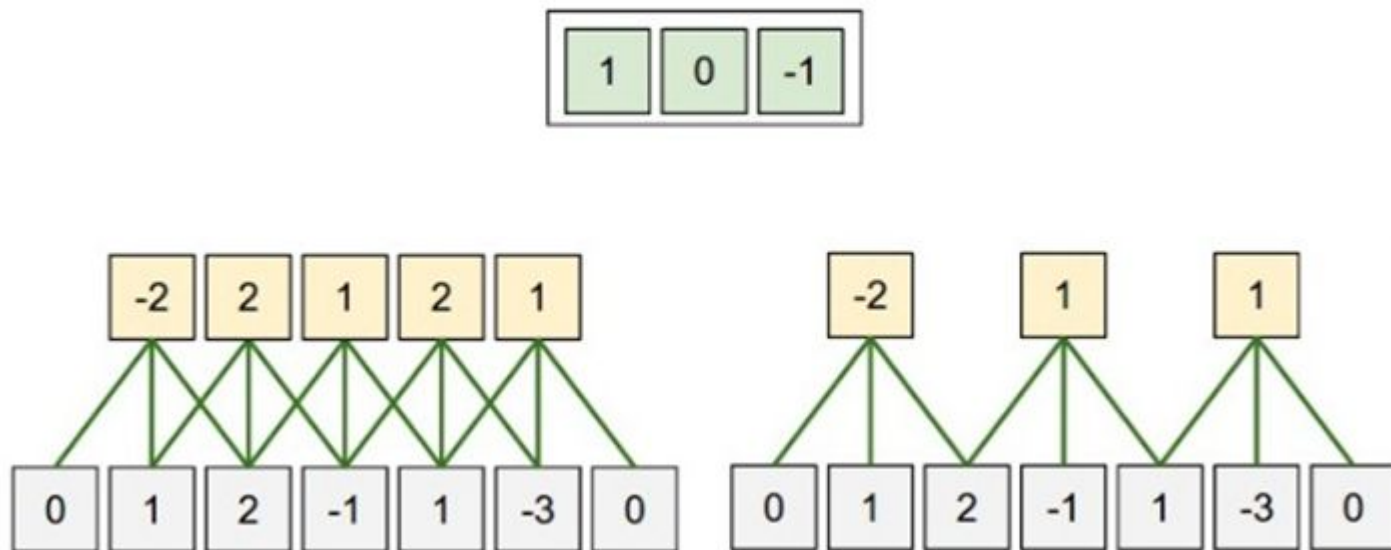
也就是說！
我們可以把時間序列當作一維的圖像看待

可以利用CNN來解決時間序列問題



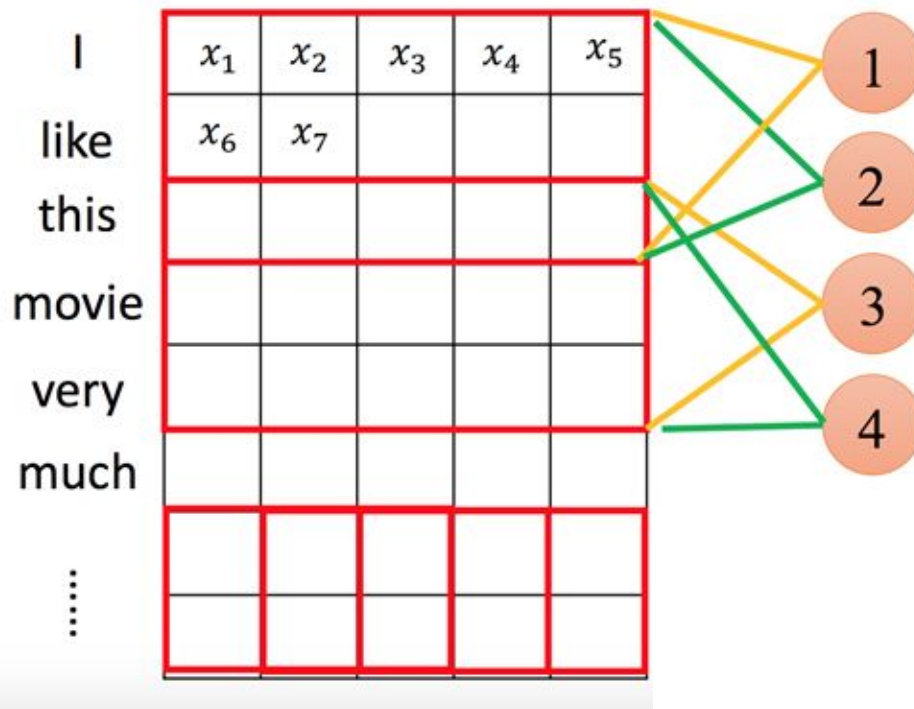
Convolution1D 示意圖

Convolution 1-D



1D signal + multi channel

A document: each word is a vector



Reference

- [Convolutional Neural Network for Sentence Classification](#)
- [Best Practice for Document Classification with Deep learning](#)
- [Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline](#)



Conv1D in Tensorflow

```
##defining placeholders##
with tf.name_scope('input'):
    inputs = tf.placeholder(tf.float32, [None, window_size, 1], name='input_data')
    targets = tf.placeholder(tf.float32, [None, 1], name='targets')

##convolution layer##
with tf.variable_scope("conv_layer1"):
    conv_layer1 = tf.layers.conv1d(inputs, 32, kernel_size=3, strides=1, activation=tf.nn.relu)
with tf.variable_scope("conv_layer2"):
    conv_layer2 = tf.layers.conv1d(conv_layer1, 64, kernel_size=3, strides=1, activation=tf.nn.relu)

##Output layer##
with tf.variable_scope('output_layer'):
    logits = output_layer(conv_layer2, 64, number_of_classes)

##loss and optimization##
with tf.name_scope('loss_and_opt'):
    loss, opt = opt_loss(logits, targets, learning_rate, gradient_clip_margin, batch_size)
```



Conv1D

```
with tf.variable_scope("conv_layer1"):
    conv_layer1 = tf.layers.conv1d(inputs, 32, kernel_size=3, strides=1, activation=tf.nn.relu)
with tf.variable_scope("conv_layer2"):
    conv_layer2 = tf.layers.conv1d(conv_layer1, 64, kernel_size=3, strides=1, activation=tf.nn.relu)
```

- conv1D與過去的conv2D大同小異，filter掃過的方向只會沿著一個軸

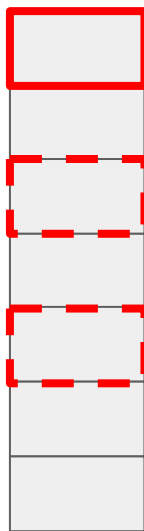
```
X_train size: (700, 7, 1)
y_train size: (700, 1)
X_test size: (49, 7, 1)
y_test size: (49, 1)
```

- 儘管conv1D中filter移動的軸只有一個方向，input_shape仍必須要是三維以上
(若X_train.shape = (700, 7), tensorflow會報錯)



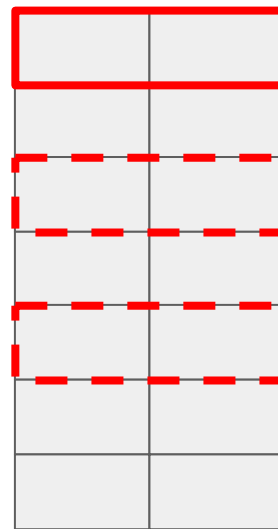
Conv1D(圖解)

shape = (700, 7, 1)
(只有一個feature)
的情況下



filter的方向

shape = (700, 7, 2)
(兩個feature)的情
況下



Difference between CNN & RNN

- RNN屬於深層神經網路，較不容易訓練
- 雖然時間序列問題中，兩種神經網路輸進去資料的shape是一樣的，但是RNN是會把一個一個time step的東西輸進記憶體，而CNN則是一次把所有的時序資料都放進記憶體。在時序資料很長的情況下，RNN會比CNN省記憶體
- CNN就是DNN的變體，因此時間序列也是可以用DNN架模型



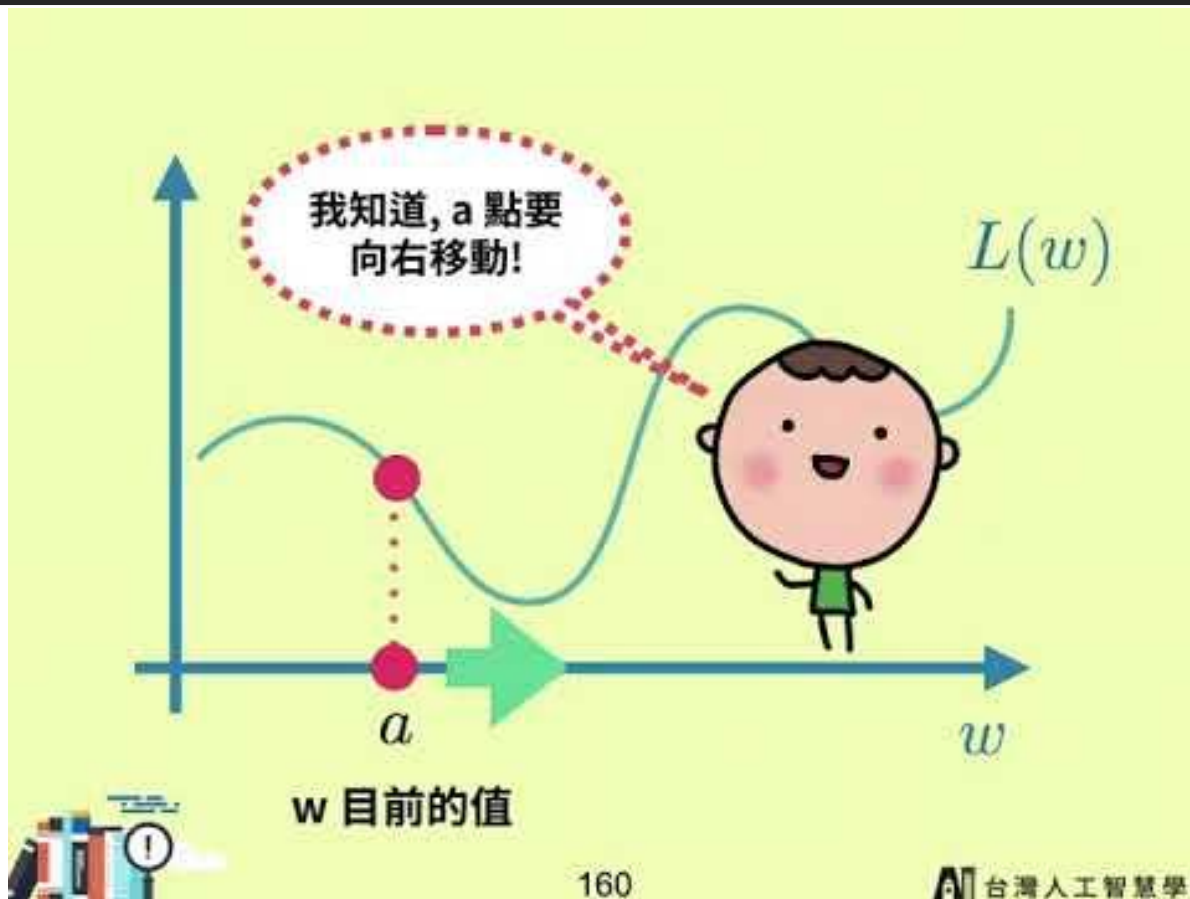
程式練習 Tesla stock price predict

- 03_rnn_conv1d_stock_prediction.ipynb
 - 重複前一份的練習，把 LSTM 改成 Conv1D



Gradient Descent的概念

0801 Gradient Descent 學習法原理



0802多變數其實是一樣的

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。

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

