國立高雄科技大學
National Kaohsiung University of Science and Technology

深度學習建立預測模型
(二元分類、多類別分類、純量回歸)

資工系 陳俊豪 教授

Outline

3-1 神經網路的核心元件

3-2 Keras簡介

3-3 建立一個深度學習的作業環境

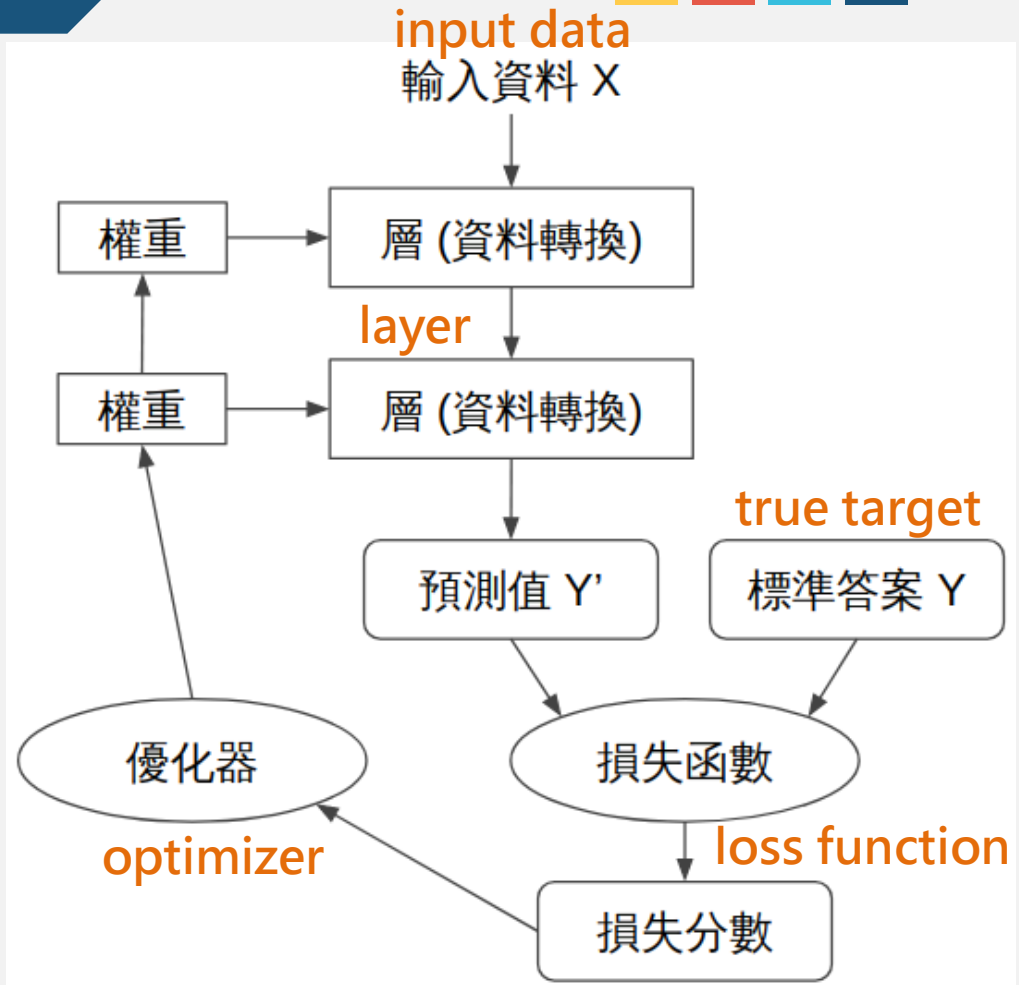
3-4 二元分類範例：將電影評論分類為正評或負評

3-5 多類別分類範例：分類數位新聞專欄

3-6 迴歸範例：預測房價

3-1 神經網路的核心元件

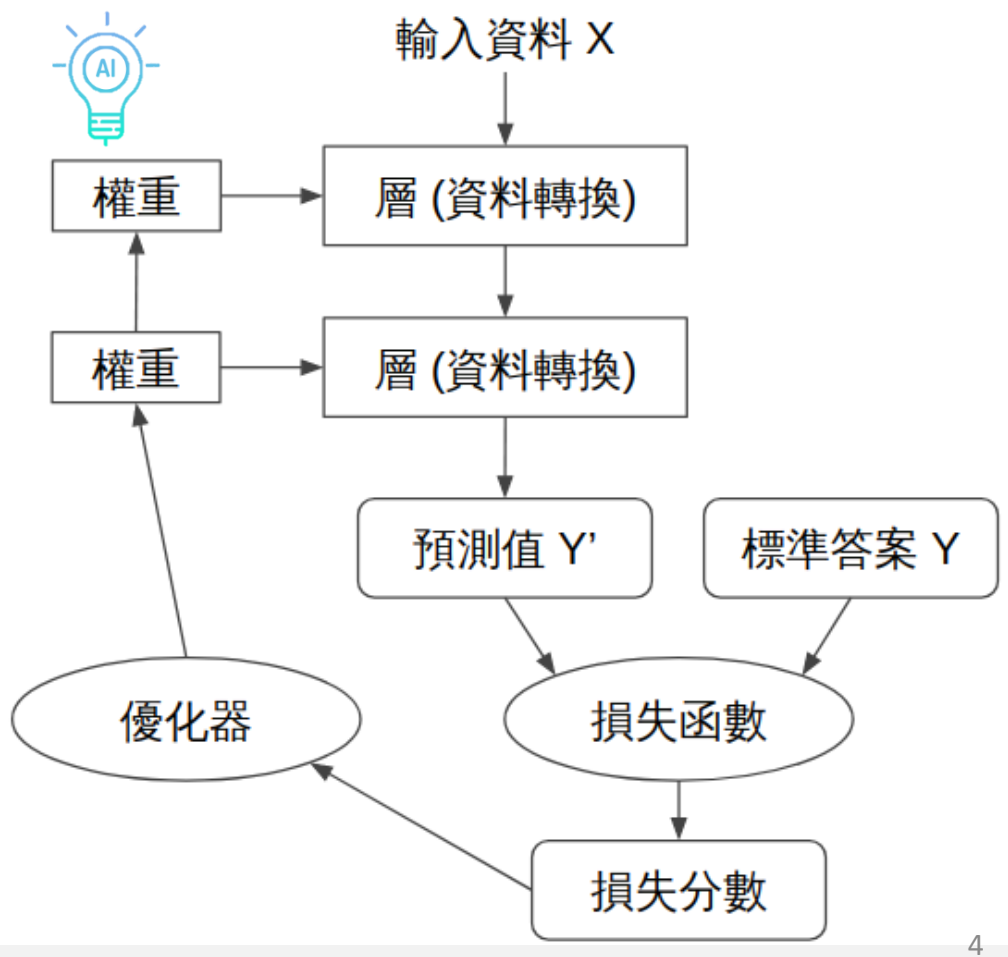
- ✓ **層(layers)**
  - 用來組成一個神經網路模型
- ✓ **輸入資料(input data)和目標(target)**
  - 用來訓練及檢視一個神經網路
- ✓ **損失函數(loss function)**
  - 用來取得學習的回饋訊號
- ✓ **優化器(optimizer)**
  - 用來決定學習進行的方式



3-1-1 層(layers)

✓層(layers)

- 是神經網路的基本處理模組，可輸入一個或多個張量，權重運算後，在輸出一個或多個張量
- 層的權重經過隨機梯度下降法(SGD)不斷更新權重，**權重即為神經網路的智慧所在**



3-1-1 層(layers)

✓不同資料型態需要不同的layer處理：

- 1D向量資料儲存在2D張量中
  - ✓ 其shape為(樣本，特徵)，用密集連接層(densely connected layer)連接
- 序列(sequence)資料儲存於3D張量中
  - 其shape為(樣本samples ，時戳timesteps ，特徵features)，用循環層(recurrent layer，如LSTM)來處理
- 影像資料(e.g., rgb)儲存於4D張量中，通常由2D卷積層(Conv2D layer)處理

```
from keras import layers    #從 keras 套件中匯入layers模組
Layer = layers.Dense (32, input_shape=(784, ))
```

補充：
Dense: 使用layers模組中的Dense類別建立一個layer
32：指定輸出單位的數量
此layer會把輸入端第1軸的784維資料轉換(用張量運算)成輸出端第1軸的32維資料

3-1-1 層(layers)

✓ 下游層承接上層的輸出張量作為其輸入 ➔ 考量層與層之間有相容性問題

- 若使用Keras則不用擔心此問題
- 因為除輸入層外，後續增加到模型的每一層輸入shape都由Keras自動建立

```
from keras import models
From keras import layers
model=models.Sequential()
model.add(layers.Dense(32, input_shape=(784, )))
model.add(layers.Dense(32)) #第2層就不用指定input_shape了
```

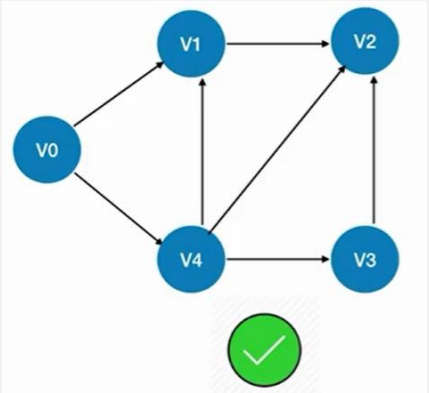
補充：
Keras會自動將其輸入形狀推斷為前一層的輸出形狀，當然units還是要指定，才可以控制神經網路的寬度

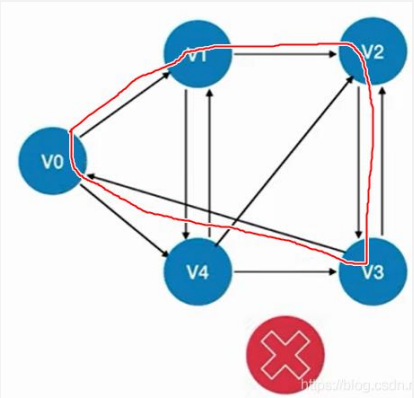
3-1-2 模型：由層組成的神經網路

✓ 深度學習模型：

- 有向無環(Directed Acyclic Graph, DAG) 的層的連結

有向無環DAG，若一個有向圖中，不存在循環，則為「有向無環」





- 最常見的是線性堆疊的層，單一輸入對應到單一輸出
- 神經網路的架構很重要，不同架構會對應到不同的假設空間
- 假設空間涵蓋最佳張量組合點，運算出的效果才會好

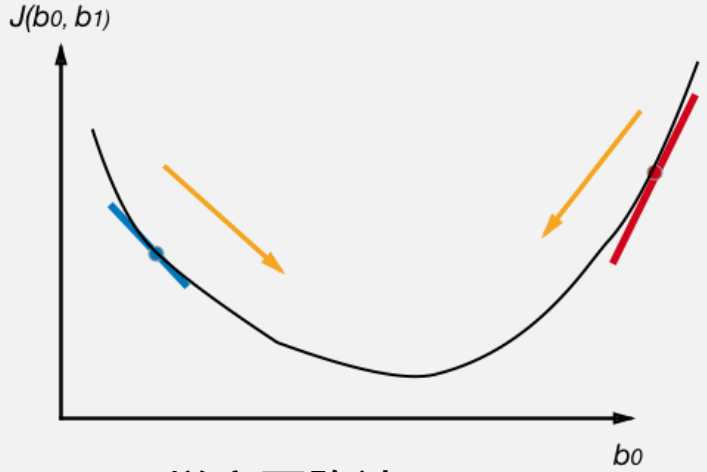
3-1-3 損失函數和優化器

✓ 損失函數(目標函數)

- 訓練期間要逐漸將此函數最小化，衡量任務成功與否的指標

✓ 優化器

- 決定損失函數來更新神經網路，使損失值變小，通常是以**梯度下降法(SGD)**或衍生的方法來執行



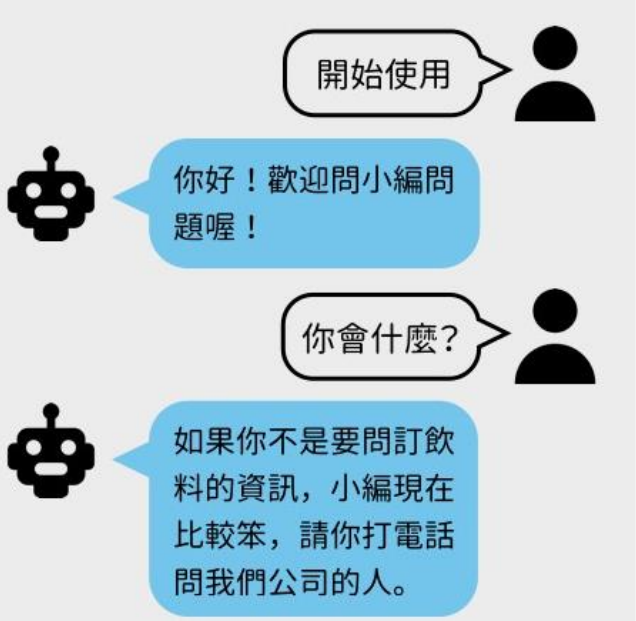
梯度下降法(SGD)

每一個輸出有一個損失函數，但梯度下降過程基於單一的純量損失分數，所以有多個損失函數的神經網路，損失分數最後透過平均值計算合併為單一純量

3-1-3 損失函數和優化器

✓ 神經網路的損失函數與當前任務的成功與否有很大的關連，若損失函數不符合問題，會得不到想要的結果

不當的損失函數，會得到答非所問的結果



選擇正確損失函數的參考準則：

問題	可參考損失函數
二元分類	二元交叉熵(binary crossentropy)
多類別分類	分類交叉熵(categorical crossentropy)
迴歸	均方差(meansquared error)
序列學習	連結時序分類(connectionist temporal classification)

3-2 Keras簡介

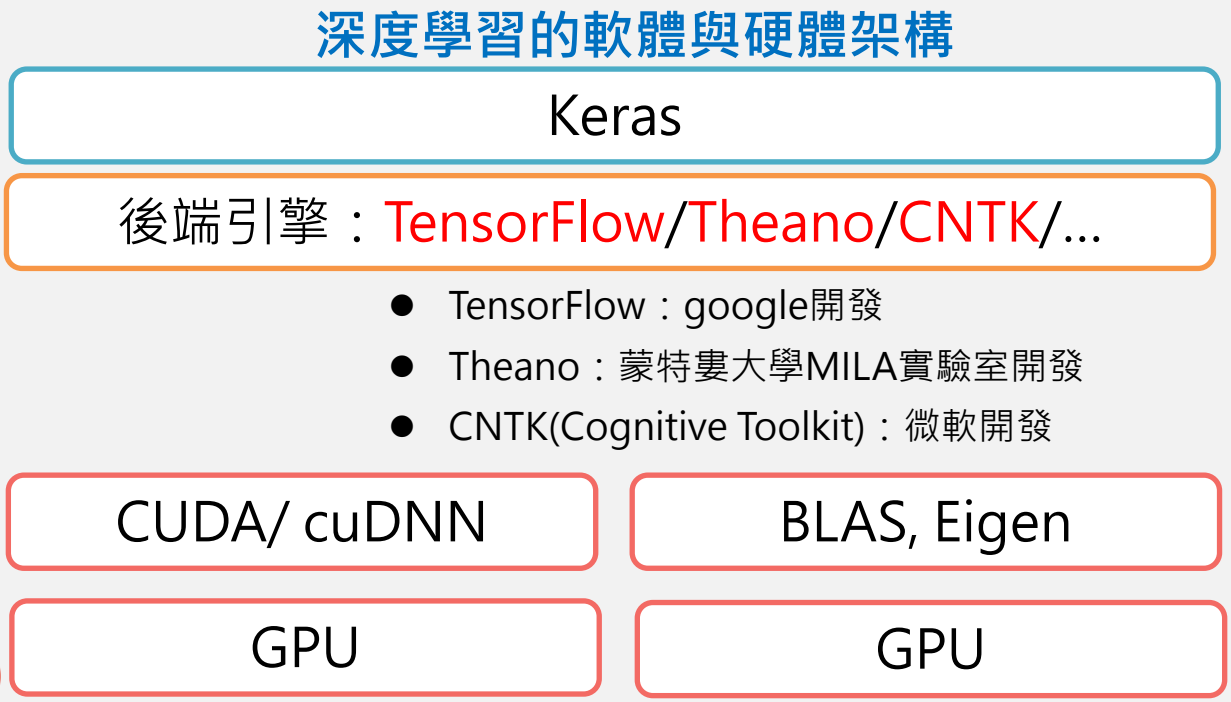
✓ Keras (http://keras.io) 是python的深度學習框架

✓特點包含

- 允許相同的程式碼在CPU或GPU上執行
- 具備友善的應用程式介面API
- 內建程式庫支援卷積神經網路(電腦視覺)、循環神經網路(序列資料處理)，以及兩者的任何組合
- 支援任意神經網路架構，包括多元輸入與多元輸出模型、神經層共享、模型共享等。(引數共享，權重共享)
- 相容性廣泛，適用於python2.7到3.7
- 可使用在商業專案上，也在學界被廣泛使用

3-2-1 Keras、TensorFlow、Theano與CNTK

✓ Keras提供開發深度學習模型所需要的高階功能模組，因此無法處理張量運算與微分等作業，故需依賴後端引擎



GPU上，TensorFlow會運行優化的NVIDIA CUDA深度神經網路程式庫(cuDNN)

CPU上，TensorFlow會運行內建的Eigen

3-3 建立一個深度學習的作業環境

✓ 使用雲端運算環境作為入門途徑(低成本)，若想成為高手，可以額外準備GPU！

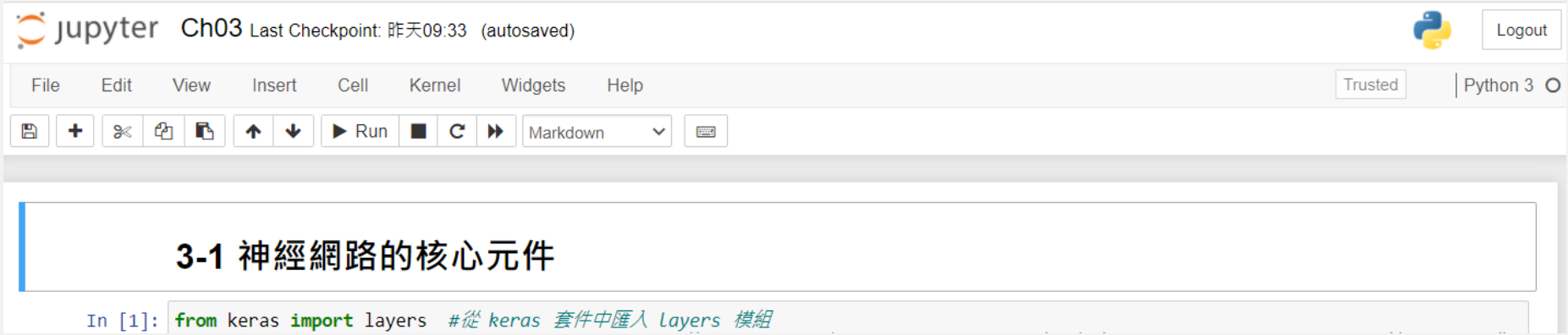
Step1 : 開啟Terminal

Step2 : 輸入jupyter notebook會開啟瀏覽器

Step3 : 開新檔案 (右上方new，選python3)

Step4 : 即可開啟jupter環境編寫程式

```
from keras import models    #從 keras 套件中匯入 models 模組
from keras import layers    #從 keras 套件中匯入 layers 模組
```



3-4 二元分類

✓ 二元分類

- 這一類演算法都常被稱作**二元分類（two-class classification）**，被用來解決只有兩種結果的問題：**是或否**、**開或關**、**抽煙或不抽煙**、**買或不買**等

- 舉例來說：
  - ✓ 這位顧客會不會續約？
  - ✓ 這是一張貓還是狗的圖片？
  - ✓ 這位顧客會不會點最上面的連結？
  - ✓ 如果繼續開一千英里，這個輪胎會不會爆胎？
  - ✓ 抵五元或打七五折，哪一個促銷手段能吸引更多顧客？

3-4-1 二元分類 - IMDB 資料集

- ✓ 二元分類範例 ➔ 將電影評論分類為正評或負評
- ✓ IMDB 資料集(Internet Movie Database)：
  - ✓ 與電影與電視相關的評論資料庫
  - ✓ 擷取 50,000 個高度兩極化的正負評論
  - ✓ 將擷取的評論，25,000條用來訓練，25,000條用來測試，正負評論各佔50%
  - ✓ 注意事項：**永遠不應該把訓練機器學習的資料，拿來測試模型的優劣**

補充：事先建立一個單字對應數字的字典，再把評論中的單字依序換成該單字在字典中的編號。字典會依照單字的常用程度來為單字編號，編號越前面代表越常用。編號指的是字典單字鍵的值，不是 index 位置，例如：

單字	from	the	and	a	…	in	…	wonderful
編號	0	1	2	3	…	8	…	386

評論內容是： “in a wonderful morning …” -> [8 、3 、386 、…]

3-4-1 二元分類 - IMDB 資料集

✓ 載入 IMDB 資料集

```
from keras.datasets import imdb  #從 keras.datasets 套件中匯入 imdb 資料

(train_data, train_labels),(test_data, test_labels)=
imdb.load_data(num_words=10000)

        #從 imdb 中讀取, 只有在訓練集當中最常用的前 10,000 個單字才會被載入,
        #分別存入(訓練資料, 訓練標籤) 和 (測試資料, 測試標籤)
```

補充 : num_words=10000，這個參數表示讀取資料時，只允許單字對應數字的字典中編號 0~9999 的單字載入

3-4-1 二元分類 - IMDB 資料集

✓ 檢視資料集：

```
train_data[0]  #取第 0 篇評論
```

Out[] : [1, 14, 22, 16, 43, 530, 973, 1622, 1385 …]

```
train_labels[0]  #第 0 篇評論的評價
```

Out[] : 1

補充 : 1 代表此評論屬於正面，若是 0 則表示負面

3-4-1 二元分類 - IMDB 資料集

✓ 確認num_words參數是否有效：

```
max([max(sequence) for sequence in train_data])
```

Out[] : 9999 **#訓練資料的單字索引最大值**

補充 :
1. for 迴圈取出 train_data 內的第 1 筆資料 [評論 1] ，並取出[評論 1]後指定給 sequence

2. 取出評論 1 最大的 sequence

3. 迴圈跑完，取出所有評論中，最大的 sequence

此步驟用來確認 num_words = 10000 是否有包含

3-4-1 二元分類 - IMDB 資料集

✓ 如何將評論的索引值解碼成英文單字

```
word_index = imdb.get_word_index()                    #單字對數字的字典
print(word_index)
```

Out[] : {‘fawn’: 34701, ‘tsukino’: 52006, ‘nunnery’: 52007, ‘sonja’: 16816, ‘vani’: 63951,
‘woods’: 1408, ‘spiders’: 16115, ‘hanging’: 2345 }

✓ 反轉成索引值對英文單字

```
reverse_word_index = dict( [(value, key) for (key, value) in word_index.items()])
print(reverse_word_index)                             #數字對單字的字典
```

Out[] : {34701: 'fawn', 52006: 'tsukino', 52007: 'nunnery', 16816: 'sonja', 63951: 'vani’,
1408: 'woods', 16115: 'spiders', 2345: 'hanging’}

3-4-1 二元分類 - IMDB 資料集

✓ 將第一筆評論的所有數字轉換成英文單字

```
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
          #此處假設 train_data[0] 內容為 [ 5343, 3234, 9098 ]

print(decoded_review)
```

Out[] : “marshall shoots honeymoon”

補充 : reverse_word_index.get(i - 3, '?') ，使用 i – 3原因為 load_data()
會自動將所有數字 + 3 ，因為 0~2 有特殊用途，故在使用時須 + 3
才是真正單字

3-4-2 IMDB 資料集 – 預處理

- ✓ 由 IMDB 匯入的 train_data 及 test_data 均為二層的整數 list，要先轉換成張量才能輸入到神經網路，轉換方法有二
  - 方法一
    - ✓ 填補資料中每個 list 的內容，使他們具有相同的長度 (元素數量)，再將整筆資料轉換成 shape 為 **(樣本數 , 填補後的樣本長度)** 的整張張量
  - 方法二
    - ✓ 對資料中的每個子 list 做 One-hot 編碼，將其轉換成 0 和 1 組成的向量。例: [3,5] -> [0,0,1,0,1,0,0,0,0…] shape : (樣本數,10000)

3-4-2 IMDB 資料集 – 預處理

✓將2層的整數 list 編碼成二元矩陣

```
import numpy as np          #匯入 numpy 模組並命名為np
def vectorize_sequences(sequences, dimension=10000):
                            #sequences 參數將傳入2層的list

    results = np.zeros((len(sequences), dimension))
                #建立全0的矩陣，其形狀為(len(sequences), dimension) ，
                其中len(sequences)為樣本數
    for i, sequence in enumerate(sequences):
                #用 enumerate()為每個字串列編號，編號會存到i，子串列存到sequence
        results[i, sequence] = 1.
                #將 result[i] 中的多個元素(以sequence串列的每個元素值為索引)設為 1.0
    return results
x_train = vectorize_sequences(train_data)  #向量化訓練資料
x_test = vectorize_sequences(test_data)  #向量化測試資料
```

3-4-2 IMDB 資料集 – 預處理

✓轉換後，樣本資料變成：

```
x_train[0]
```

```
Out[] : array([0., 1., 1., ..., 0., 0., 0.])   #由0、1組成的向量
```

✓將標籤資料向量化: (用numpy的asarray()來轉換即可)

```
y_train = np.asarray(train_labels).astype('float32')  #向量化訓練標籤
y_test = np.asarray(test_labels).astype('float32')  #向量化測試標籤
```

補充 : 將 list 轉成 1D 陣列，型別設為 32 bits 的浮點數

3-4-3 IMDB 資料集 – 建立神經網路

✓模型定義

```
from keras import models
from keras import layers

model = models.Sequential()
#使用 models 模組的 Sequential 類別, 建立一個物件讓新增的神經網路層可以進行堆疊
model.add(layers.Dense(16, activation='relu', input_shape=(10000, )))
#輸入層也是隱藏層
model.add(layers.Dense(16, activation='relu'))  #隱藏層
model.add(layers.Dense(1, activation='sigmoid'))  #輸出層
```

補充：
為了獲得更豐富多樣的假設空間，以利於深度轉換的表現，因此需要一個非線性函數或啟動函數。relu是深度學習中最常用的啟動函數，也稱「非線性函數」，其他的可用函數，如：prelu、elu。

3-4-3 IMDB 資料集 – 建立神經網路

✓編譯模型

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',  #調整之前的損失函數, 並進行 compile
              metrics=['accuracy'])
```

✓調整優化器

```
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                                                    #調整優化器參數lr的值
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3-4-3 IMDB 資料集 – 建立神經網路

✓使用自行定義的損失函數與 metrics 函數

```
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                        loss=losses.binary_crossentropy,
                        #自行指定其他的損失函數與 metrics 函數
                        metrics=[metrics.binary_accuracy])
```

3-4-4 驗證神經網路的模型

✓設定驗證資料集

```
x_val = x_train[:10000]              #取輸入資料的前 10000 個做驗證 (val)
partial_x_train = x_train[10000:]    #輸入資料的第 10000 個開始才是訓練資料

y_val = y_train[:10000]              #對應的, 要取標籤的前 10000 個做為驗證標籤
partial_y_train = y_train[10000:]    #從標籤的第 10000 個開始才是訓練資料的標籤
```

```
Out[] : array([0., 1., 1., ..., 0., 0., 0.])
```

3-4-4 驗證神經網路的模型

✓訓練模型

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])  #建立訓練模型

history = model.fit(partial_x_train,
#呼叫 fit() 開始訓練 (使用 partial_x_train 輸入資料、 partial_y_train 標籤、20 個訓練週期、次訓練週期使用 512 筆資料)
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))  #同時傳入驗證集的資料與標籤
```

補充：
驗證動作是在fit()中的每一訓練週期(epoch)，不是全部訓練完再做！

3-4-4 驗證神經網路的模型

✓訓練模型結果

```
Out[] : Epoch 1/20 15000/15000 [==============================] - 2s 159us/step - loss: 0.5051 - acc: 0.7871 - val_loss: 0.3777 - val_acc: 0.8703
        Epoch 2/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.2994 - acc: 0.9045 - val_loss: 0.3004 - val_acc: 0.8895
        Epoch 3/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.2175 - acc: 0.9281 - val_loss: 0.3086 - val_acc: 0.8717
        Epoch 4/20 15000/15000 [==============================] - 2s 139us/step - loss: 0.1748 - acc: 0.9438 - val_loss: 0.2826 - val_acc: 0.8842
        Epoch 5/20 15000/15000 [==============================] - 2s 136us/step - loss: 0.1422 - acc: 0.9539 - val_loss: 0.2855 - val_acc: 0.8857
        Epoch 6/20 15000/15000 [==============================] - 2s 135us/step - loss: 0.1148 - acc: 0.9651 - val_loss: 0.3154 - val_acc: 0.8774
        Epoch 7/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.0978 - acc: 0.9711 - val_loss: 0.3130 - val_acc: 0.8842
        Epoch 8/20 15000/15000 [==============================] - 2s 136us/step - loss: 0.0806 - acc: 0.9764 - val_loss: 0.3864 - val_acc: 0.8653
        Epoch 9/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.0660 - acc: 0.9821 - val_loss: 0.3638 - val_acc: 0.8779
        Epoch 10/20 15000/15000 [==============================] - 2s 136us/step - loss: 0.0558 - acc: 0.9849 - val_loss: 0.3862 - val_acc: 0.8790
        Epoch 11/20 15000/15000 [==============================] - 2s 135us/step - loss: 0.0427 - acc: 0.9901 - val_loss: 0.4153 - val_acc: 0.8786
        Epoch 12/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.0376 - acc: 0.9921 - val_loss: 0.4631 - val_acc: 0.8678
        Epoch 13/20 15000/15000 [==============================] - 2s 140us/step - loss: 0.0300 - acc: 0.9929 - val_loss: 0.4740 - val_acc: 0.8732
        Epoch 14/20 15000/15000 [==============================] - 2s 138us/step - loss: 0.0238 - acc: 0.9949 - val_loss: 0.5065 - val_acc: 0.8722
        Epoch 15/20 15000/15000 [==============================] - 2s 138us/step - loss: 0.0176 - acc: 0.9980 - val_loss: 0.5366 - val_acc: 0.8705
        Epoch 16/20 15000/15000 [==============================] - 2s 136us/step - loss: 0.0164 - acc: 0.9972 - val_loss: 0.5744 - val_acc: 0.8704
        Epoch 17/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.0102 - acc: 0.9993 - val_loss: 0.6291 - val_acc: 0.8636
        Epoch 18/20 15000/15000 [==============================] - 2s 138us/step - loss: 0.0119 - acc: 0.9971 - val_loss: 0.6446 - val_acc: 0.8674
        Epoch 19/20 15000/15000 [==============================] - 2s 139us/step - loss: 0.0054 - acc: 0.9997 - val_loss: 0.7283 - val_acc: 0.8569
        Epoch 20/20 15000/15000 [==============================] - 2s 137us/step - loss: 0.0099 - acc: 0.9975 - val_loss: 0.7054 - val_acc: 0.8651
```

3-4-4 驗證神經網路的模型

- ✓ model.fit() 回傳 的history 物件
  - 是一個包含有關訓練過程中所有發生資料的字典

```
history_dict = history.history
history_dict.keys()
```

```
Out[] : dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

補充 : 這個字典包含 4 個項目，是訓練和驗證時監控的指標。

3-4-4 驗證神經網路的模型

✓繪製訓練與驗證的損失分數

```
import matplotlib.pyplot as plt    #匯入 matplotlib.pyplot 模組, 後續程式用 plt 使用
history_dict = history.history
loss_values = history_dict['loss']   #取得每次訓練的 loss 訓練損失分數並存成 loss_values 變數
val_loss_values = history_dict['val_loss']    #取得每次驗證的 val_loss 驗證損失分數並指定給
                                              val_loss_values 變數

epochs = range(1, len(loss_values)+ 1) #len(loss_values) 項目個數為 20,範圍從 1 到 21 (不含 21) 的期間

plt.plot(epochs, loss_values, 'bo', label='Training loss')
                              #以 'bo' 指定用藍色點點畫出 x 軸為訓練週期、y 軸為訓練損失分數的圖表, 標籤設為訓練損失分數
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
                              #以 'b' 指定用藍色線條畫出 x 軸為訓練週期、y 軸為驗證損失分數的圖表, 標籤設為驗證損失分數
plt.title('Training and validation loss')
plt.xlabel('Epochs')  #將此圖表標題的 x 軸設為訓練週期分數
plt.ylabel('Loss')    #將此圖表標題的 y 軸設為損失
plt.legend()          #可以追加每個輸出圖表的圖像名稱
plt.show()            #顯示圖表
```

3-4-4 驗證神經網路的模型

✓繪製訓練與驗證的損失分數結果

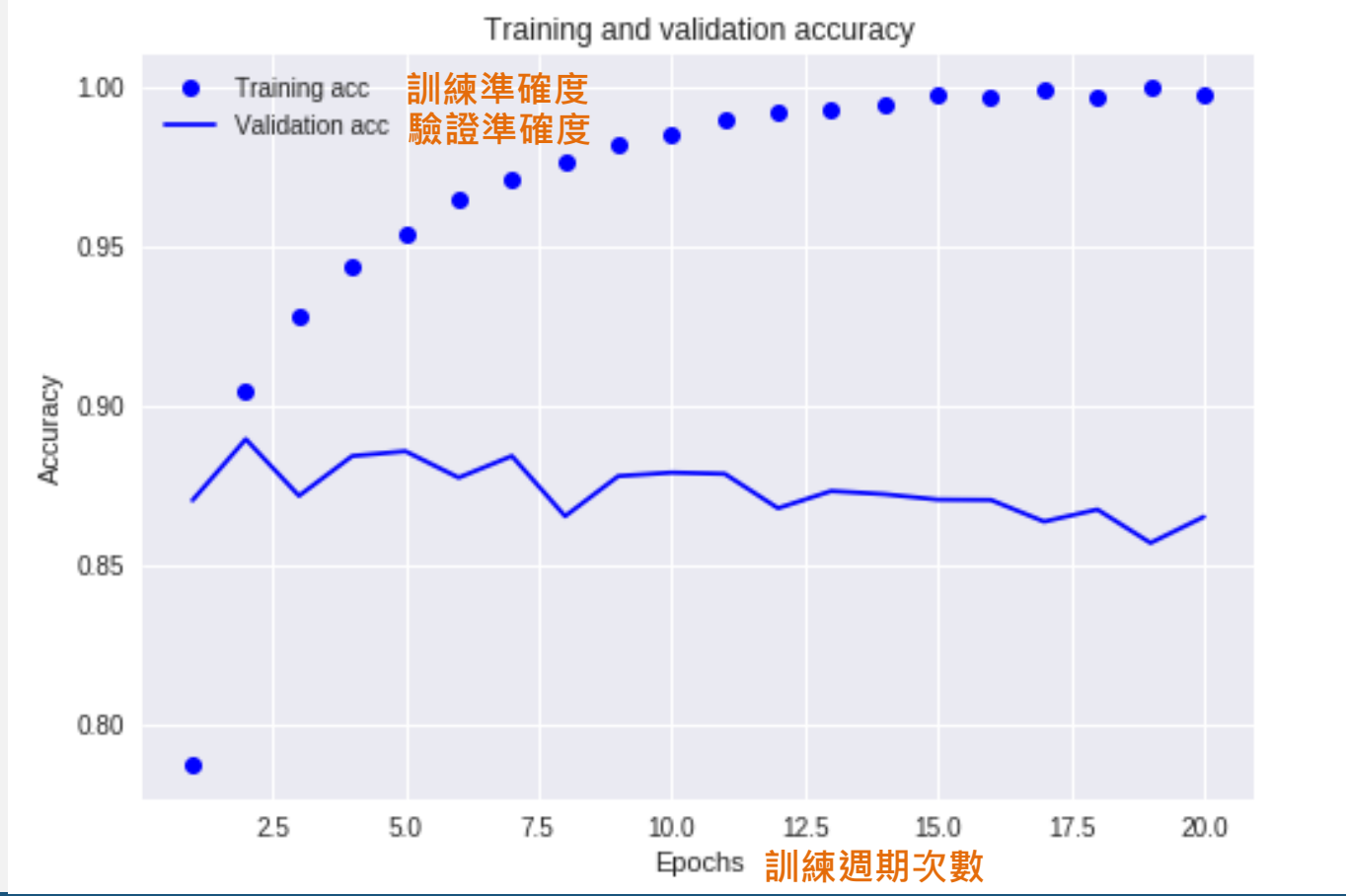
3-4-4 驗證神經網路的模型

✓繪製訓練和驗證的準確度

```
plt.clf()  #清除圖表
acc = history_dict['acc']
val_acc = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

3-4-4 驗證神經網路的模型

✓繪製訓練和驗證的準確度結果



3-4-4 驗證神經網路的模型

✓用Epochs = 4，重新開始訓練模型

```
model = models.Sequential()  #建立模型
model.add(layers.Dense(16, activation='relu', input_shape=(10000, )))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

model.fit(x_train, y_train, epochs=4, batch_size=512)  #訓練 4 個週期的神經網路
results = model.evaluate(x_test, y_test)               #輸入測試資料與測試標籤進行評估
```

3-4-4 驗證神經網路的模型

✓重新開始訓練模型結果

```
Out[] : Epoch 1/4
25000/25000 [==============================] - 3s 120us/step - loss: 0.4739 - acc: 0.8215
Epoch 2/4
25000/25000 [==============================] - 3s 100us/step - loss: 0.2675 - acc: 0.9092
Epoch 3/4
25000/25000 [==============================] - 2s 99us/step - loss: 0.2033 - acc: 0.9281
Epoch 4/4
25000/25000 [==============================] - 2s 99us/step - loss: 0.1713 - acc: 0.9386
25000/25000 [==============================] - 2s 83us/step
```

results

Out[] : [0.31117002926826476, 0.87608]

補充 : 此方法可達到88%的準確度。

3-4-5 使用訓練完成的神經網路對新資料進行預測

✓使用predict預測方法對評論文章做出評價**判讀**(到底這篇文章是正評還是負評)

```
model.predict(x_test)        #使用模型對x_test測試集的評論文章做判讀
```

```
Out[] : array([[0.16204852],
       [0.9998516 ],
       [0.40186763],
       ...,
       [0.08137858],
       [0.04903173],
       [0.45961794]], dtype=float32
```

補充：
神經網路對某些樣本很有信心(0.99或更高(正面)，或0.02或更低(負面)
但對其他某些樣本則是不太有信心(0.65或0.4)。

3-5 多元分類

✓多元分類

- 這種演算法被稱為**多元分類（multi-class classification）**，顧名思義，它可以用來解決有多種（或很多種）回答的問題，例如：哪種口味、哪個人、哪個部分、哪間公司、哪位參選人

✓例如:

- 這是哪種動物的圖片？(貓、狗、倉鼠)
- 這種雷達訊號是來自哪種飛機？(機型1、2、...、N)
- 這個電影屬於哪一種類型？(愛情、恐怖、動作、驚悚、動作...)
- 這則推特（twitter）所包含的情緒為何？(開心、憤怒、期待、難過...)
- 哪種口味的冰淇淋賣的比上一季好？(香草、巧克力、薄荷)

3-5-1 多元分類 - 路透社資料集

- ✓ 多元分類範例 ➔ 分類數位新聞專欄
  - 路透社資料集(Reuters dataset)：
    - ✓ 1986年路透社發布的一組簡短新聞和主題的資料集，被廣泛用於文章分類的實驗中
    - ✓ 將新聞分類成 46 個不同的主題(如：財經、政治、旅遊、運動...)
- ✓ 每個資料點只能歸入一類，屬於「單標籤多類別分類」
- ✓ 載入資料集：

```
from keras.datasets import reuters  #從 keras.datasets 套件中匯入 reuters 資料集
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
#從 reuters 資料集中讀取訓練資料、訓練標籤、測試資料、測試標籤
#與IMDB資料集一樣，將資料量限制在10,000個最常出現的單字
```

3-5-2 路透社資料集 - 預處理

✓ 檢視資料集：

```
len(train_data) #訓練資料個數
```

Out[] : 8982

```
len(test_data) #測試資料個數
```

Out[] : 2246

```
train_data[10] # 如同IMDB評論，每個樣本都是單字索引的整數list，第10筆訓練資料(已將單字轉成整數索引值)
```

Out[] : [1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979, 3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]

✓ 將其解碼轉回單字

```
word_index = reuters.get_word_index()
reverse_word_index = dict([ (value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?’ ) for i in
train_data[0]]) #這些索引值有位移 3 個位置, 因為 0, 1 與 2 分別是保留索引值, 代表「填補」、「開始位置」與「未知」
```

3-5-2 路透社資料集 - 預處理

✓ 檢視資料集

```
train_labels[10]   #第10筆訓練資料的標籤
Out[] : 3          #為第3類, 樣本資料標籤介於0~45之間
```

✓ 將資料加以編碼(向量化處理)

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):  #使用 one-hot 編碼
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)  #向量化訓練資料
x_test = vectorize_sequences(test_data)    #向量化測試資料
```

3-5-2 路透社資料集 - 預處理

✓ 對標籤進行向量化的方式有2種：

主題	財經	政治	…	旅遊
One-hot 編碼	1000	0100	…	0001

使用one-hot編碼(轉換成0&1)

3-5-2介紹如何編碼

主題	財經	政治	…	旅遊
張量	0	1	…	45

將標籤list轉換為整數標籤

3-5-6介紹如何編碼

3-5-2 路透社資料集 - 預處理

✓ one-hot編碼運作

```
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))  #把所有元素設為 0
    for i, label in enumerate(labels):
        results[i, label] = 1.   #第 i 個元素設為 1
    return results

one_hot_train_labels = to_one_hot(train_labels)  #向量化訓練標籤
one_hot_test_labels = to_one_hot(test_labels)    #向量化測試標籤
```

```
#使用Keras 內建函式 : categorical() 進行one-hot 編碼
from keras.utils.np_utils import to_categorical # 新版使用此模組from keras.utils

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

3-5-3 建立神經網路

✓模型定義

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000, )))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))   #輸出46維的向量
```

補充：
最後一層使用softmax啟動函數，代表神經網路將在46個不同的輸出類別上輸出機率分佈，即對每個輸入樣本，神經網路將產生46維輸出向量，其中輸出[i]是該樣本對應到類別i的機率，46個機率值的總和為1

3-5-3 建立神經網路

✓ 編譯模型

```
model.compile(optimizer='rmsprop ',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

補充 :

- Optimizer 優化器: 適合處理複雜的 error surface，但仍然需要先設置一個全局學習率 η
- Loss損失函數:測量兩個機率之間分佈的差異，透過最小化這兩個分佈之間的距離
- metrics成效衡量指標:對於test set中的N條資料，統計系統能夠判斷準確的資料條數M，最後進行簡單相除得到M/N作為評判標準

3-5-4 驗證

✓ 在訓練資料中，另外切出1000個樣本作為驗證資料集

```
#切片驗證資料集
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

✓ 訓練神經網路20個週期(epochs)

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,              #訓練模型(20個週期)
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

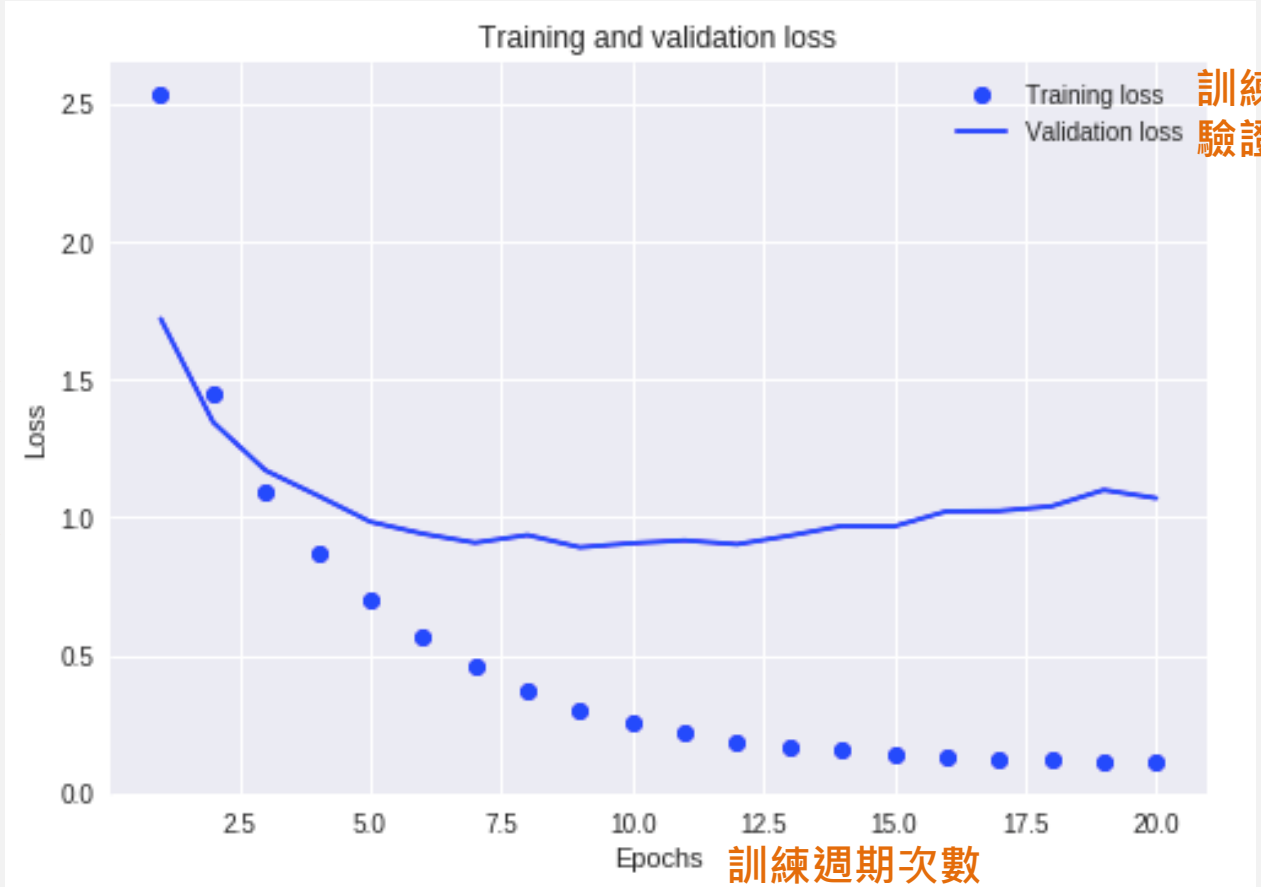
3-5-4 驗證

✓ 繪製訓練和驗證的損失

```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss)+ 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

3-5-4 驗證

✓ 繪製訓練和驗證的損失結果



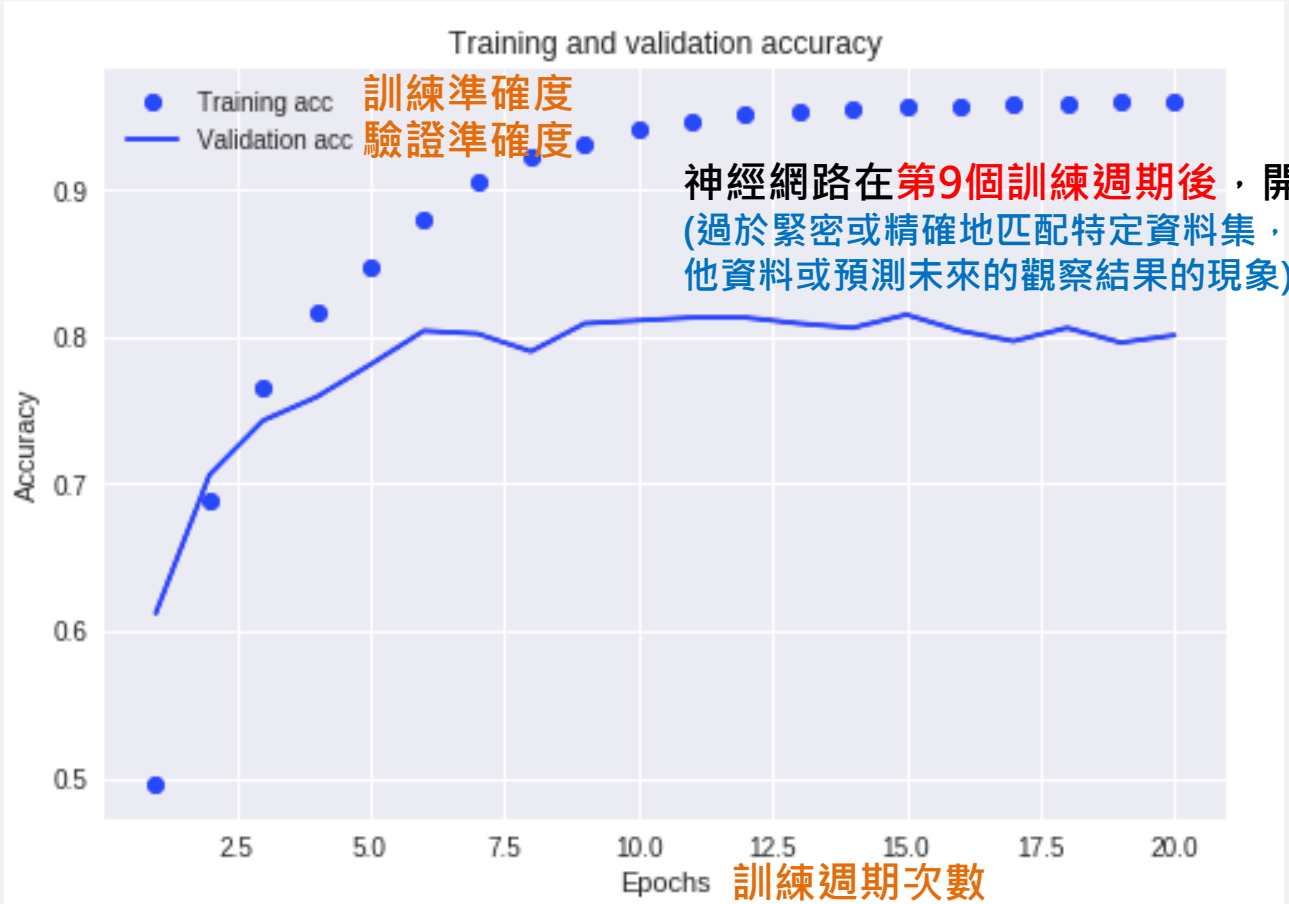
3-5-4 驗證

✓ 訓練和驗證的準確度

```
plt.clf()       #先清除畫面

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

3-5-4 驗證

✓ 訓練和驗證的準確度結果



神經網路在第9個訓練週期後，開始發生過度配適
(過於緊密或精確地匹配特定資料集，以致於無法良好地調適其他資料或預測未來的觀察結果的現象)

3-5-4 驗證

✓ 設定9 Epochs，從頭開始重新訓練模型

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000, )))
model.add(layers.Dense(64, activation='relu'  ))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop  ',
                    loss='categorical_crossentropy  ',
                    metrics=['accuracy'])

model.fit(partial_x_train,
                  partial_y_train,
                  epochs=9,  #9 個週期就好!!!
                  batch_size=512,
                  validation_data=(x_val, y_val))

results = model.evaluate(x_test, one_hot_test_labels)
```

比隨機猜測(如：丟銅板)50%的準確度高

Out[] : [0.9565213431445807, 0.7969723953695458 9] #準確度達將近80%

#損失函數

3-5-4 驗證

✓ 使用測試集test_label做隨機猜測 random.shuffle()

```
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels)== np.array(test_labels_copy)
float(np.sum(hits_array))/ len(test_labels)
```

Out[] : [0.18655387355298308]　**#準確度達將近19%**

補充：
在測試集的資料上，我們訓練出來的模型(準確度將近80%)，比起隨機猜測(準確度將近19%)而言，結果是相當不錯的！

3-5-5 對新資料進行預測

- ✓ 可以使用模型的predict方法，來取得所有**46個主題**的機率分佈
- ✓ 對新資料產生預測值

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

```
predictions[0].shape
```

Out[] : (46,) **#每個項目的預測長度都是46的向量**

```
np.sum(predictions[0])
```

Out[] : 1.0 **#該向量中機率總和為1**

```
np.argmax(predictions[0])
```

Out[] : 4 **#項目中得到最大數值就是預測類別**

3-5-6 用另一種方式處理標籤與損失

✓ 將類別轉換為整數張量

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

✓ 此方法唯一會改變的是**損失函數**的選擇

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

在數學上仍為相同，只是介面不同

損失函數	categorical_crossentropy		sparse_categorical_crossentropy	
輸出	[1, 0, 0] [1, 0, 0] [1, 0, 0]	分類標籤	0 1 2	整數標籤

3-5-7 擁有足夠大型中間層的重要性

✓在之前的範例中，由於最終輸出有46維，因此應**避免中間層小於最終輸出維度**

✓實驗看看若使用具有小於46units的中間層，具有**資訊瓶頸的模型**會如何?

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000, )))
model.add(layers.Dense(4, activation='relu')) #中間層改為 4 維
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
model.fit(partial_x_train,
                partial_y_train,
                epochs=20,
                batch_size=128,
                validation_data=(x_val, y_val))
```

補充：結果驗證準確度近71%，下降8~9%，因為試圖壓縮大量資訊到一個維度的中間層表示空間

3-6 預測房價：迴歸範例

✓ 常見機器學習問題：

- 分類：根據輸入的資料預測該資料所應歸屬的類別
  - ✓ 例如：將路透社的數位新聞專欄分為46個不同的主題

- 迴歸(regression)：用來預測連續值，而非離散標籤
  - ✓ 例如：根據氣象資料預測明天溫度
  - ✓ 例如：根據開發規格預測完成軟體專案所需時間

注意！迴歸與邏輯斯迴歸不一樣！

- 邏輯斯迴歸(logistic regression)：常用於分類（二元或多類別），處理目標值為**類別型**
  - ✓ 例如：預測某位學生會考試**及格或失敗**?(利用唸書小時數等特徵)

- 迴歸(regression)：常用於產出結果數字，處理目標值為**數值型**
  - ✓ 例如：由父母身高預測子女**身高**
  - ✓ 例如：用人口成長（自變項）預測（解釋）電話**用戶數**的成長

3-6-1 迴歸範例 -波士頓住房價格資料集

- ✓ 迴歸(預測連續值)範例 ➔ 預測波士頓住房價格
  - ● 波士頓住房價格資料集：
    - ✓ 1970年代中期波士頓的郊區資料，包含犯罪率、當地財產稅等
    - ✓ 當時房屋價格介於10,000美元～50,000美元
    - ✓ 包含506筆資料（404筆訓練樣本，102筆測試樣本）
    - ✓ 輸入資料的每個特徵都有不同的單位刻度，某些為比例介於0, 1之間，某些為1~12 或 0~100的數值
- ✓ 迴歸目標 ➔ 預測某郊區房屋一人自住房屋的價格中位數，以1千美元為單位
- ✓ 載入資料集：

```
from keras.datasets import boston_housing

(train_data, train_targets),(test_data, test_targets)=boston_housing.load_data()
#前幾個例子中叫train_labels, 現在叫train_targets, 二者在資料集上意義是相同的
```

3-6-1 迴歸範例 -波士頓住房價格資料集

✓ 檢視資料集：

```
train_data.shape #查看訓練資料的shape
```

Out[] : (404, 13)

```
test_data.shape #查看測試資料的shape
```

Out[] : (102, 13)

補充：
13個數值特徵，如：犯罪率、每個住宅的平均房間數、高速公路的可達性.....

3-6-1 迴歸範例 -波士頓住房價格資料集

✓ 檢視房價的中位數：

```
train_targets  #訓練資料的標籤，就是實際成交房價的中位數
```

Out[] : array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1, 17.9, 23.1, 19.9, 15.7, 8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8, 32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9, 23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7, 12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7, 9.6, 31.5, 24.8, 19.1, 22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3, 15.6, 10.5, 6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5, 8.3, 14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5, 14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1, 28.7, …, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])

補充：
以上數值以$1,000美元為單位

3-6-2 波士頓住房價格資料集 – 預處理

✓ 正規化資料：
- 為了統一13個特徵值的刻度
- 將特徵減去特徵的平均值並除以標準差
- 特徵值會以0為中心，且以標準差為刻度

```
mean = train_data.mean(axis=0) #沿著第 0 軸 (batch_size 軸) 做平均
train_data -= mean
std = train_data.std(axis=0)       #沿著第 0 軸 (batch_size 軸) 算標準差
train_data /= std

test_data -= mean
test_data /= std
```

補充：
要對測試資料進行正規化，需使用訓練資料集來計算平均數、標準差，不可以使用測試集的資料數值，否則資訊洩漏，會使模型不準確

3-6-3 建立神經網路

✓ 因為可用訓練樣本較少，避免過度配適(overfitting)問題，使用較小的神經網路，包含兩個隱藏層，每層64個單元

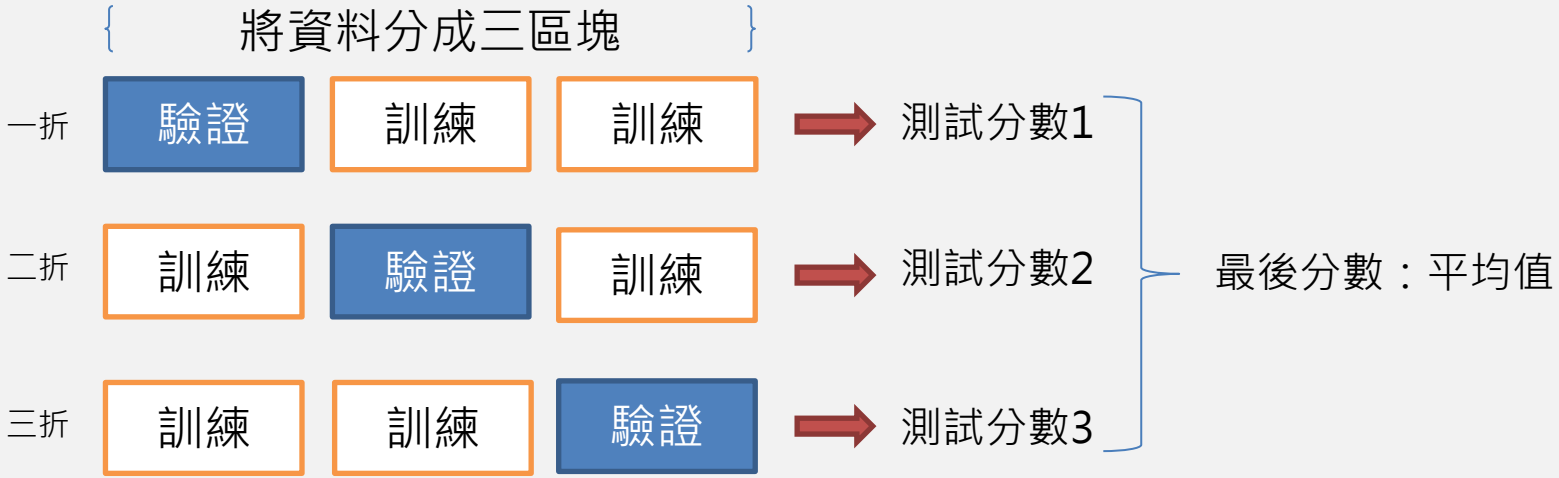
```
#模型定義
from keras import models
from keras import layers

def build_model():
    model = models.Sequential()   #建構一個 sequential 模型
    model.add(layers.Dense(64, activation='relu', input_shape=(train_data.shape[1], )))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    #mse損失函數：計算預測值與目標值間差異的平方(常用於迴歸問題)
    #mae平均絕對誤差：預測值與目標值間差異的絕對值
    #例如：mae=0.5,表示預測差距為平均500美元(基本單位為1,000美元)
    return model
```

補充：
純量迴歸會輸出一個浮點數型別的數值(迴歸值)，若使用啟動函數將限制輸出值的範圍
例如: sigmoid啟動函數用於最後一層則神經網路只能預測0~1之間的數值

3-6-4 K-fold驗證來驗證模型的成效

- ✓ 驗證分數會因驗證資料集切割的不同產生很大變異性，會阻礙評估模型優劣的可靠性，解決方法是透過**K折交叉驗證**，將資料集太小導致變異性增大的缺點平均掉

- ✓ K折交叉驗證(K-fold cross validation)
  - 將可用資料拆分為K個區塊(e.g., K=3)，選一個驗證區塊，其餘區塊當訓練集跑一遍；然後選下一個區塊當驗證集，其餘區塊當作訓練集跑第二遍，以此類推



3-6-4 K-fold驗證來驗證模型的成效

✓ K折驗證

```
import numpy as np
k = 4   #進行 4 折交叉驗證
num_val_samples = len(train_data) // k  #驗證區塊的樣本數
num_epochs = 100
all_scores = []

for i in range(k):
    print('processing fold #', i)   #準備驗證資料：資料來自 #k 區塊
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(                    #準備訓練資料：資料來自 #k 以外的所有區塊
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
         axis=0)
     partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples] ,
         train_targets[(i + 1) *num_val_samples:]],
         axis=0)
```

3-6-4 K-fold驗證來驗證模型的成效

✓ K折驗證(承上頁)

```
model = build_model()  #建構 Keras 模型(已編譯)
model.fit(partial_train_data, partial_train_targets,
          epochs=num_epochs, batch_size=1, verbose=0) #訓練該模型
val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)  #以驗證資料來評估模型
all_scores.append(val_mae)
```

Out[] :

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

補充：verbose輸出顯示模式：0 = 靜音模式, 1 = 進度條, 2 = 每輪一行

3-6-4 K-fold驗證來驗證模型的成效

✓ K折驗證結果

```
all_scores
```

Out[] : [2.588258957792037, 3.1289568449719116, 3.1856116051248984,3.0763342615401386]

```
np.mean(all_scores)      #驗證分數的平均值
```

Out[] : 2.9947904173572462

驗證分數從2.588~3.186差距非常大，K折驗證的平均值3.0可知價格誤差平均約為3,000美元（1,000美元為單位），這只能表示K-fold能驗證模型的優劣，不能讓預測變得更準確

3-6-4 K-fold驗證來驗證模型的成效

✓ 儲存每折的驗證紀錄

```
num_epochs = 500 #更改成500週期
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)

  val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]  #準備驗證資料：資料來自 #k 區塊
  val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

  partial_train_data = np.concatenate(
      [train_data[:i * num_val_samples],   #準備訓練資料：資料來自 #k 以外的所有區塊
       train_data[(i + 1) * num_val_samples:]],
       axis=0)

  partial_train_targets = np.concatenate(
      [train_targets[:i * num_val_samples],
       train_targets[(i + 1) * num_val_samples:]],
       axis=0)
```

3-6-4 K-fold驗證來驗證模型的成效

✓ 儲存每折的驗證紀錄(承上頁)

```
model = build_model()    #建構 Keras 模型(已編譯)
history = model.fit(partial_train_data, partial_train_targets,
                    validation_data=(val_data, val_targets),
                    epochs=num_epochs, batch_size=1, verbose=0) #訓練該模型(在silent靜音模式下，verbose=0)
mae_history = history.history['val_mean_absolute_error'] #紀錄每個時期的預測差距
all_mae_histories.append(mae_history)
```

Out[] :

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

✓ 建立連續平均 K 折驗證分數的歷史

```
average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

3-6-4 K-fold驗證來驗證模型的成效

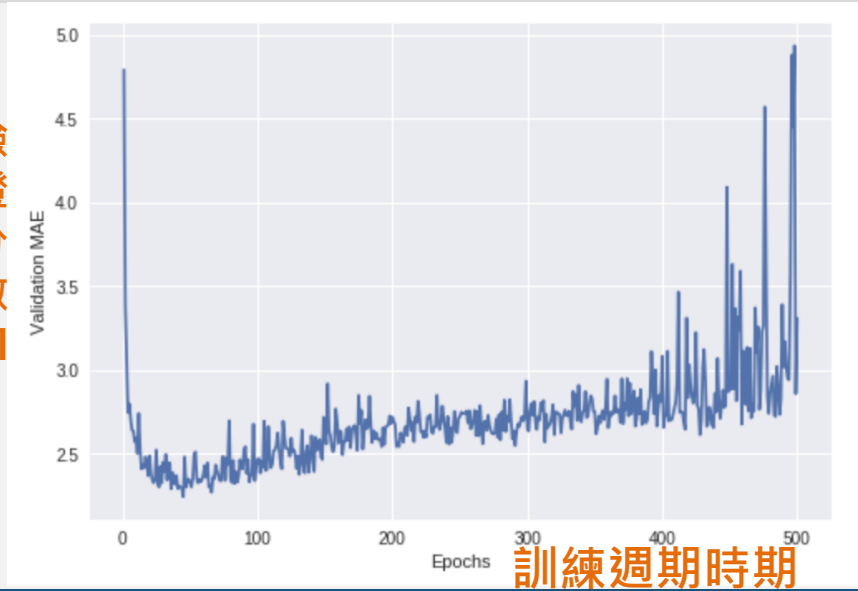
✓ 繪製驗證分數

```
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history)+ 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

#所有紀錄的數量

#所有紀錄的平均 K 折驗證分數

Out[] :



3-6-4 K-fold驗證來驗證模型的成效

- ✓ 由於單位刻度縮放、相對較高變異度，上頁的圖會看不出細節
- ✓ 省略前10個資料點（因與其他部分比例不同）
  - 將資料點取代成前一點的**指數移動平均值(EMA)**，使曲線變平滑
- ✓ **指數移動平均值(EMA)**
  - 核心：現在的資料會被過去資料所影響；時間點越近的資料影響越大，越遠則越小
  - 公式：$E_t = a * V_t + (1-a) * E_{t-1}$

    E_t：時間點t的指數移動平均值

    a：平滑係數，通常介於0~1

    V_t：時間點t的原始數值

    E_{t-1}：時間點t-1的指數移動平均值
- ✓ 指數關係：以10天指數移動平均值當範例

$$E_{10} = aV_{10} + (1-a)E_9$$

$$E_9 = aV_9 + (1-a)E_8$$

$$E_{10} = aV_{10} + (1-a)[aV_9 + (1-a)E_8] \rightarrow E_{10} = a(V_{10} + (1-a)V_9) + (1-a)^2E_8$$

每多一天，原始數值會多乘(1-a)倍，成指數關係

- ✓ 將所有天數EMA展開
  - $E_{10} = a(V_{10} + (1-a)V_9 + (1-a)^2V_8 + \cdots (1-a)^9V_1) + (1-a)^9\ E_1$

3-6-4 K-fold驗證來驗證模型的成效

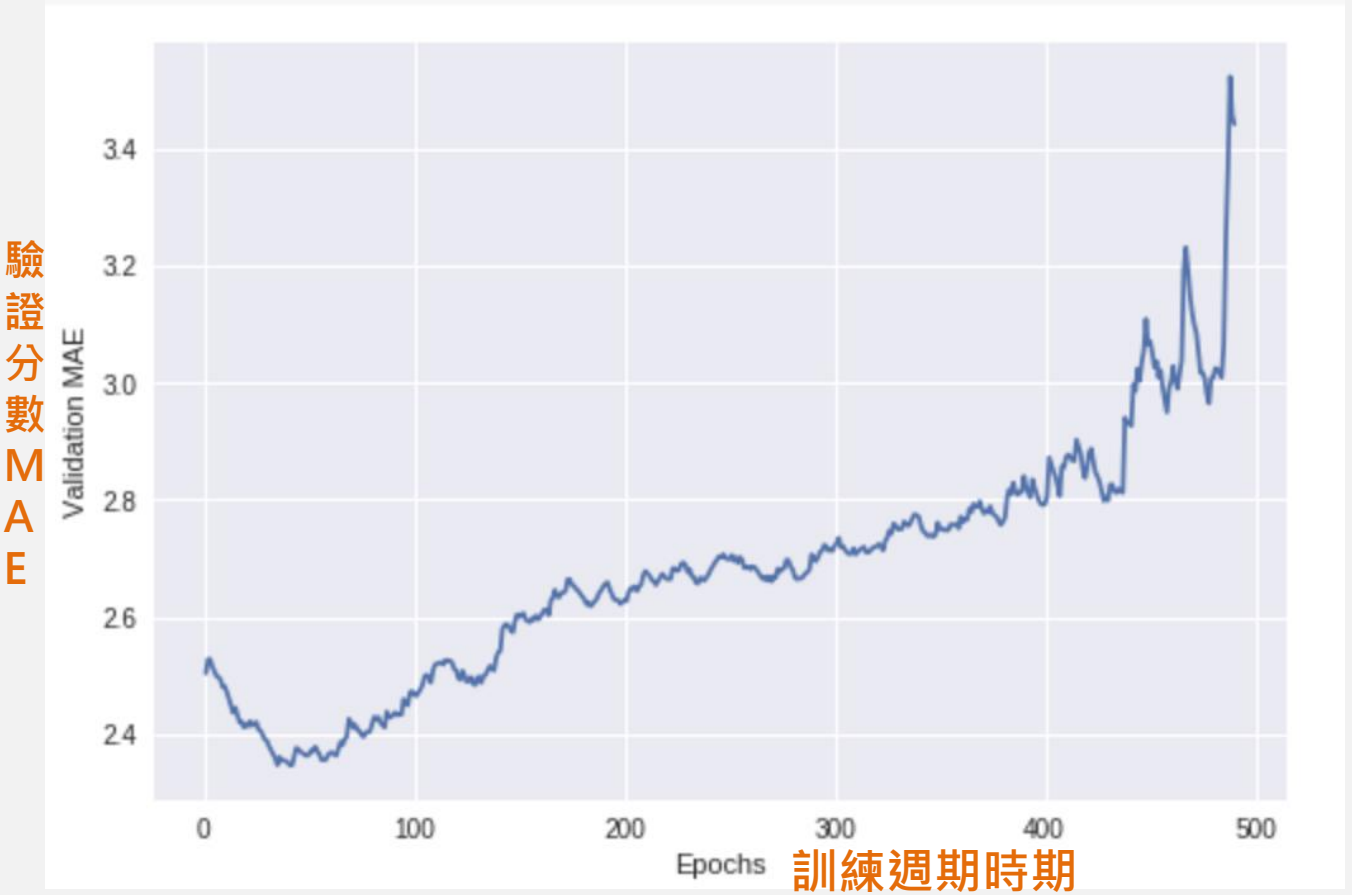
✓ 排除前十筆資料點，使用指數移動平均值(EMA) ，繪製驗證分數

```
def smooth_curve(points, factor=0.9): #指數值為 0.9
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point *(1 - factor)) #運用指數平均數
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:]) #先去除前十筆資料再平均

plt.plot(range(1, len(smooth_mae_history)+ 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

3-6-4 K-fold驗證來驗證模型的成效

✓ 排除前十筆資料點，使用指數移動平均值(EMA) ，繪製驗證分數結果

Out[] :



3-6-4 K-fold驗證來驗證模型的成效

✓ 訓練最終模型

```
model = build_model()  #建立一個用最佳參數 compile 過的新模型
model.fit(train_data, train_targets, #以整個資料進行訓練
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
model = test_mae_score
```

Out[] : 2.5532484335057877

最終驗證結果，仍有2,550美元的誤差

總結

✓ 目前為止，應該知道的內容：

- 處理向量資料中最常見的機器學習任務：二元分類、多類別分類、純量迴歸
- 資料預 (前)處理preprocessing
- 正規化：當資料具有不同量刻度或數值範圍的特徵，對每個特徵進行單獨的轉換
- 訓練過程過度配適，導致在沒見過的新資料集上得到更差的結果
- 當訓練資料過少，使用一個或兩個隱藏層的小型神經網路，避免過度配適
- 如果資料分為多個類別時，使用的中間層太小，可能會導致資料瓶頸
- 迴歸與分類使用不同的損失函數和不同的評量指標
- 處理少量資料時，K折驗證可以可靠地評估模型