CHAPTER

02

機器學習起點：多層感知器 (MLP)

吳智鴻

編修自 碁峰機器學習教科書

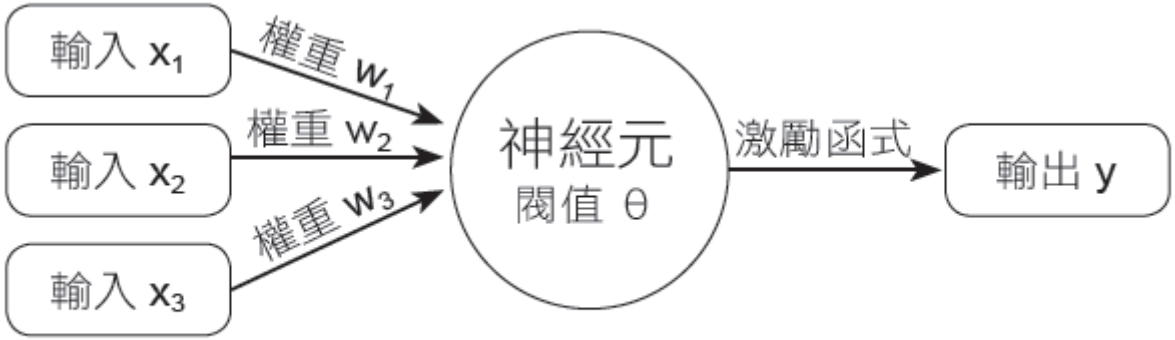
碁峰資訊

2.1 認識多層感知器 (MLP)

2.1.1 認識神經網路

神經元的運作

神經元是彼此相連的，下圖是單獨取出單一神經元的運作模型，每個神經元中都有一個 **閥值**，它的功能是設下一個門檻，如果所接收的訊號值運算後大於這個門檻，神經元就會被觸發，將接收的值經由 **激勵函式** 轉換，輸出到下一個神經元。



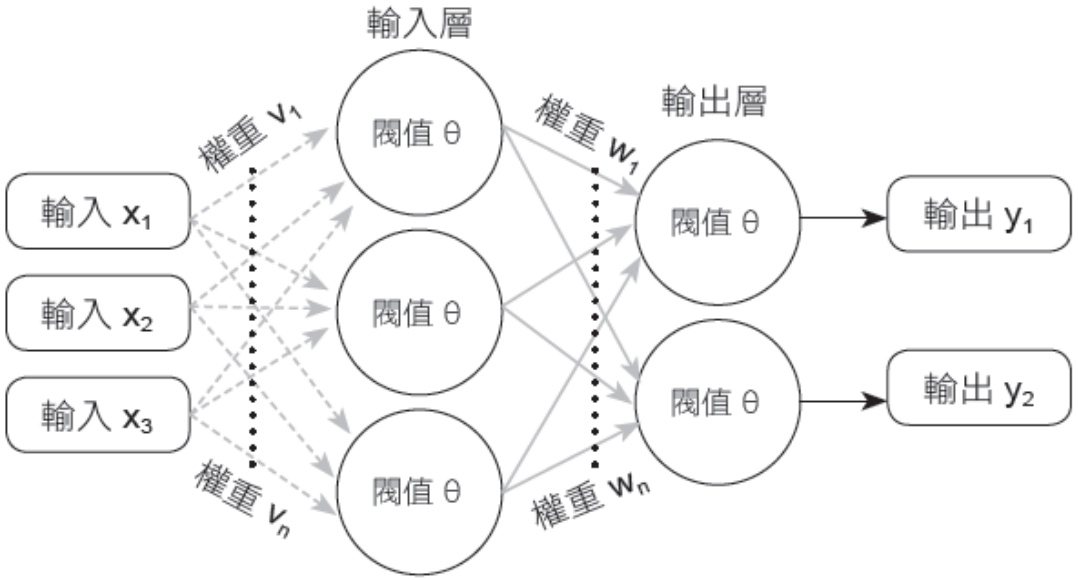
▲ 單一神經元模型：單層感知器

回首頁

其中接收的訊號值就是由其他神經元傳遞過來的多個 **輸入值(x)** 乘上相關的 **權重(w)** 再與 **閥值(θ)** 比較的動作，是很重要的關鍵，**機器學習就是在調整每個輸入值與所配置的權重**。訊號值越大越容易觸發神經元，對於神經網路運作的影響也越大。反之，訊號值越小影響就越小，而太小的訊號甚至可以忽略以節省運算的資源，讓輸出值的誤差降到最小，這個調整轉換輸出值的方式就是 **激勵函式**。

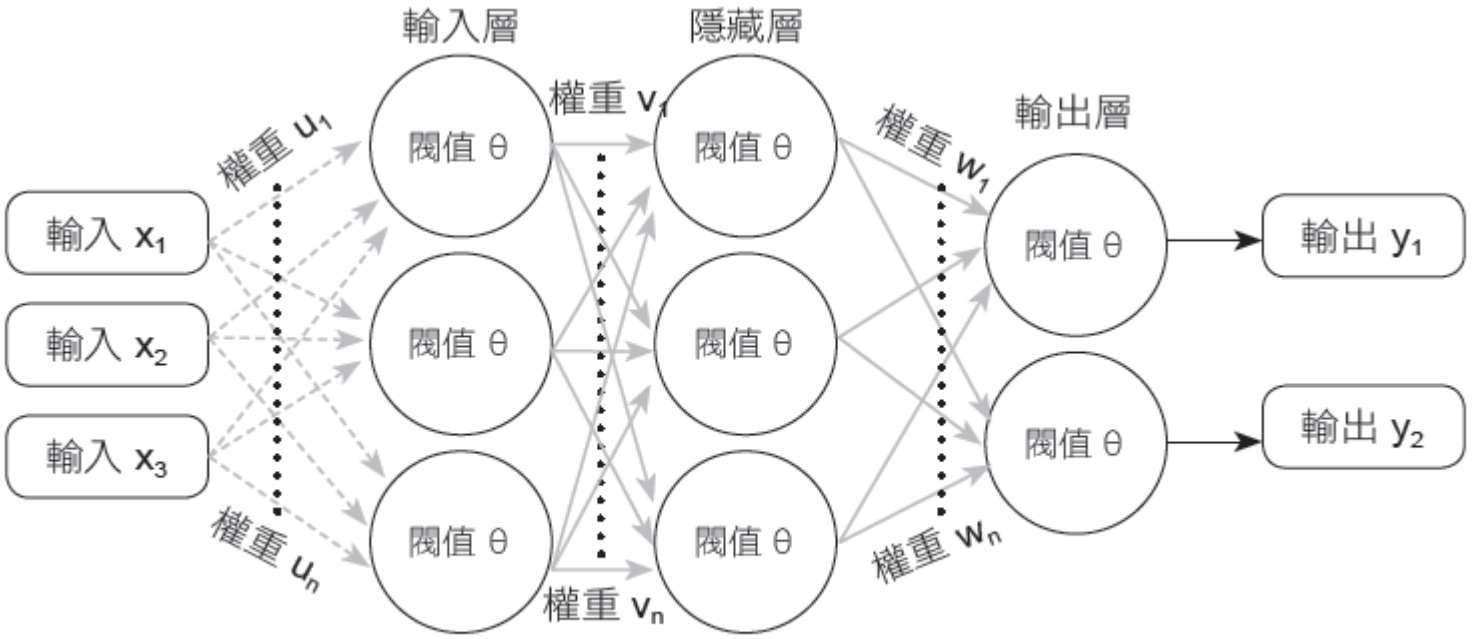
感知器的模型

感知器 (Perceptron) 就是模仿人類大腦皮層中神經網路模型進行學習的機制，所以傳遞訊號的神經元都是按層排列。單一神經元模型就是最單純的 **單層感知器**。為了解決更複雜的問題，於是發展出由接收輸入訊號的 **輸入層** 與產生輸出信號的**輸出層** 所建構的 **2 層感知器**。



▲ 2 層感知器

為了提高學習的準確率，神經網路更發展到有一個 **輸入層**、一個或多個 **隱藏層** 及一個 **輸出層** 的 **多層感知器**(MLP，Multilayer Perceptron)。

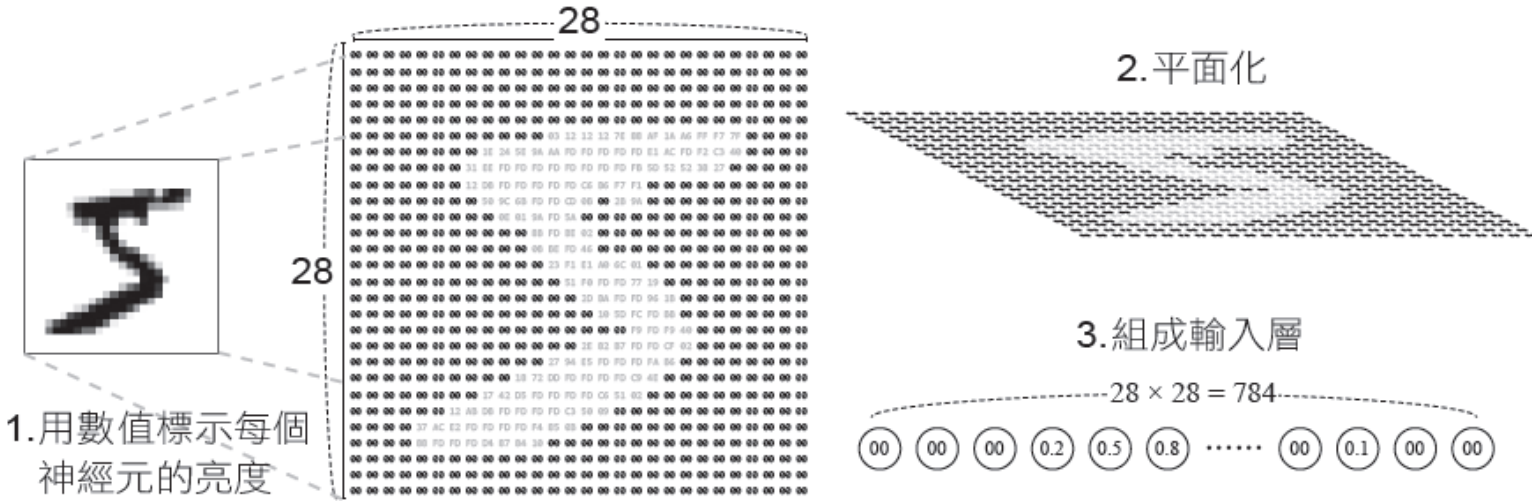


▲ 多層感知器

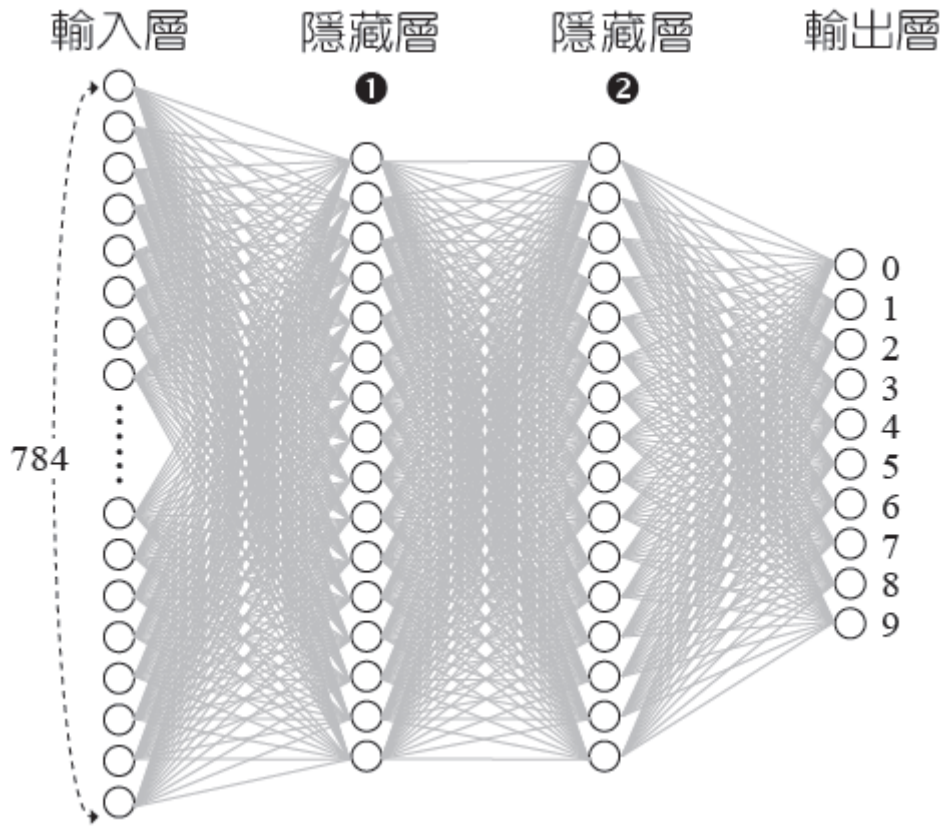
2.1.2 多層感知器的運作

多層感知器的模型

神經元在接收輸入訊號後可以想像它是儲存了一個數字的容器，其值介於0 到1之間。以28 * 28 像素的手寫辨識圖片來說，每個像素就是一個神經元，也就是一張圖片在 **輸入層** 總共有784 個神經元，每個神經元都儲存了一個數字來代表對應像素的灰階值，數值的範圍介於0 跟1 之間。而灰階值0 代表黑色，1 代表白色，這些數字我們稱為 **激勵值**，數值越大則該神經元就越亮。在輸入時要將矩陣平面化(將28 列前後相接成一列)，也就是這784 個神經元組成了神經網路的第一層。

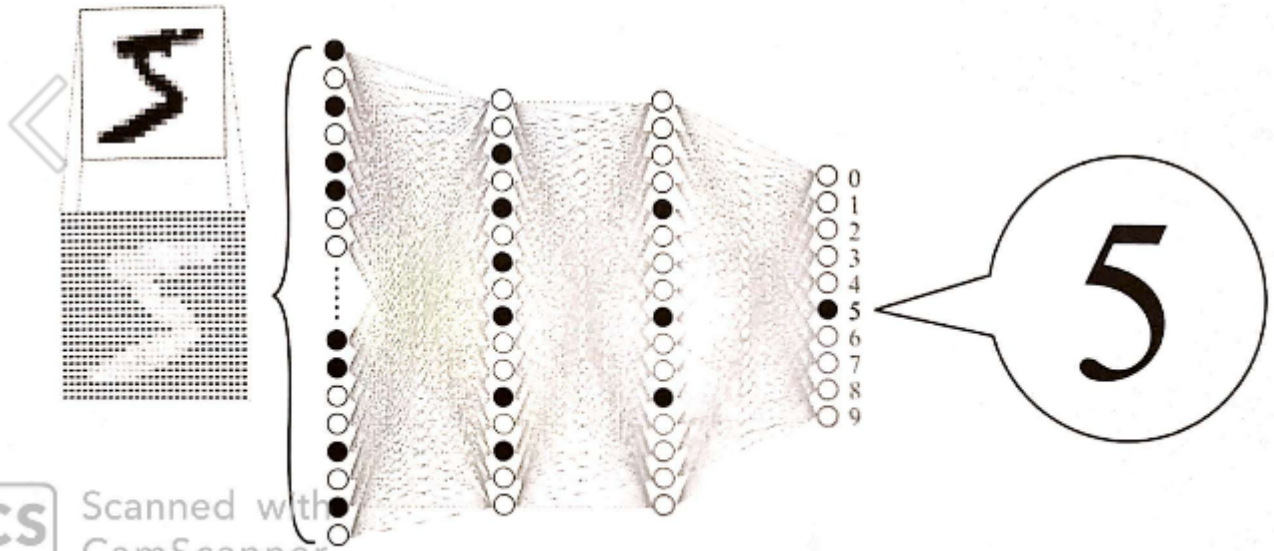


完成了輸入層，先不管其他層的內容，我們來看看它最右方的 **輸出層**，也就是最後判斷的結果，其中有10 個神經元，各代表了數字0 到9，其中也有代表的激勵值。

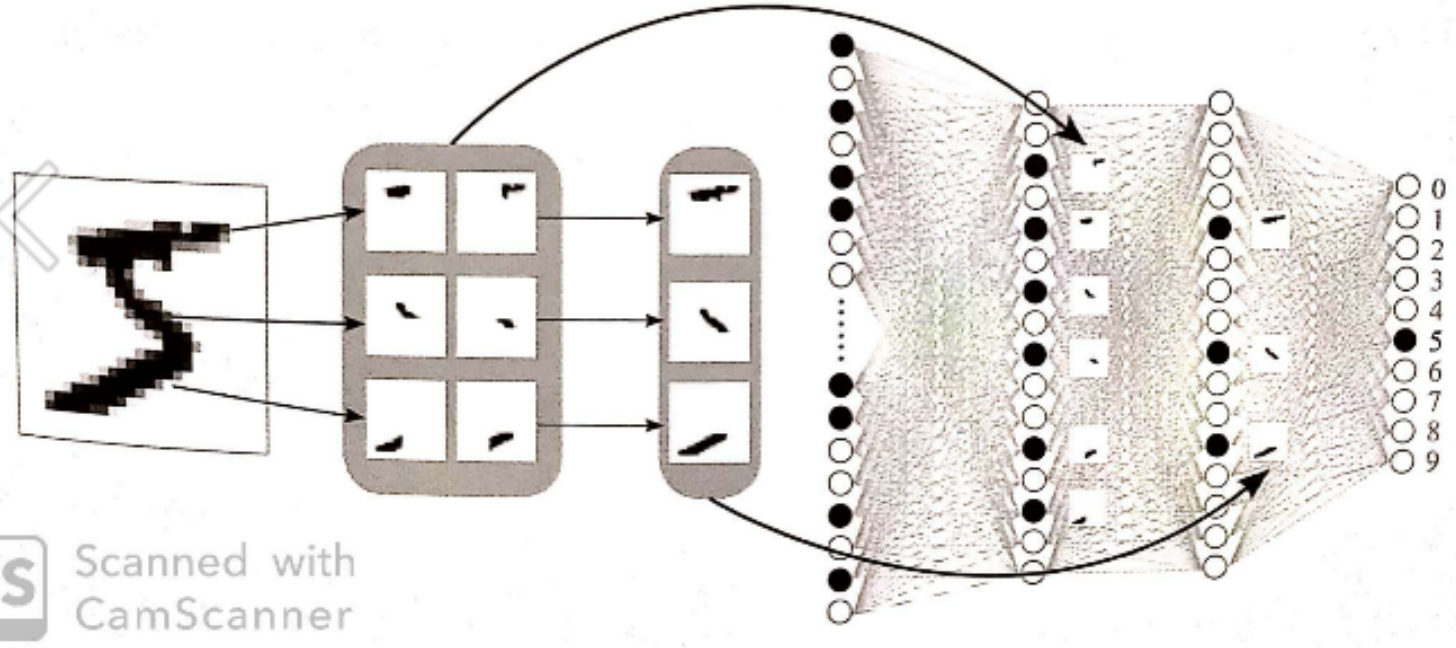


多層感知器的流程

不同以往的資料處理技術，在神經網路中每一層神經元中激勵值操作的結果會影響下一層的激勵值，一層一層之間激勵值的傳遞最後輸出判斷的結果，它的本質是在模仿人類大腦細胞被激發，引發其他神經細胞的連鎖反應。

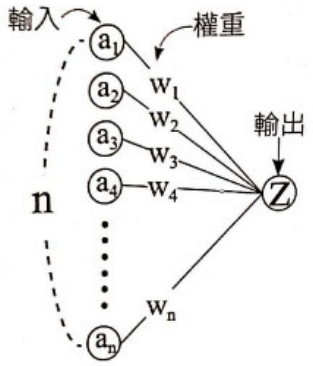


而激勵值是如何在各層之間傳遞的呢？而隱藏層又是如何運作的呢？再回到剛才的問題，在辨識手寫數字圖片時，可以將文字拆解成各個筆劃較好處理。



各層傳遞的數學模型

其中每一層神經元中激勵值的傳遞方式，第一步是把該層每個神經元的值 (a) 乘上藉由訓練所得到的 **權重** (w) 再全部加總起來。接著要設置一個觸發神經元啟動的閥值門檻，這裡稱為 **偏置** (Bias)，請將剛才的權重值總合減去偏置值。因為計算的結果可能為任何的數，但我們必須將這個結果壓縮限制在 0 與 1 之間，這裡就要透過一些函式進行處理，也就是所謂的 **激勵函式** (Activation Functions)。

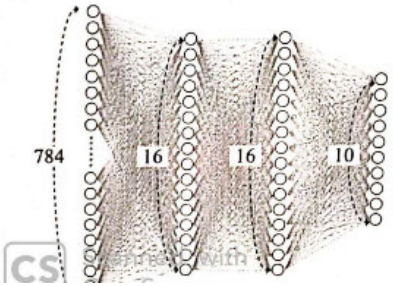


$$f(w_1a_1+w_2a_2+w_3a_3+w_4a_4\ldots+w_na_n-b)$$

$$\text{激勵函式} \rightarrow f\left(\sum_{i=1}^{n} w_i a_i - b\right) \leftarrow \text{偏置}$$

機器學習的目的

以上的動作只是第一層的所有神經元傳遞到下一個神經元的動作，試想以剛才手寫數字圖片辨識來說，輸入層有 784 個神經元，那到第一個隱藏層有 16 個神經元，就必須有 784 x 16 個權重與 16 個偏置，整個過程有一個輸入層、二個隱藏層、一個輸出層，至少就會超過 13,000 個必須要調整的參數。



權重數

$784 \times 16 + 16 \times 16 + 16 \times 10 = 12{,}960$

偏置數

$16 + 16 + 10 = 42$

多層感知器的流程

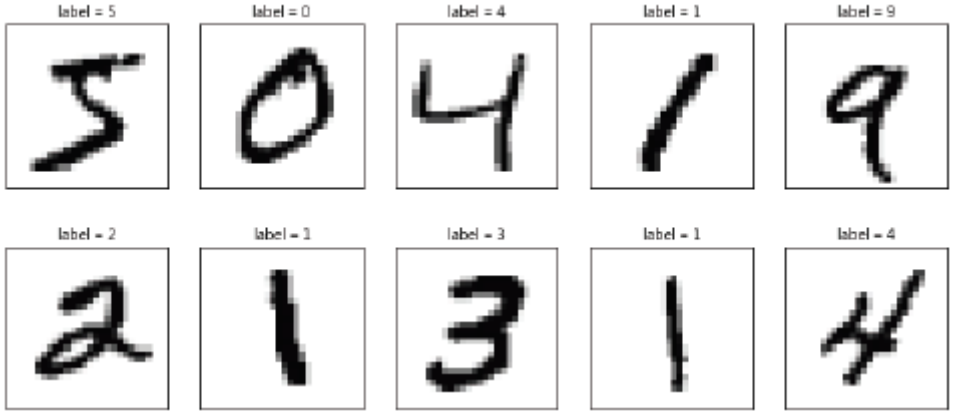
不同以往的資料處理技術，在神經網路中每一層神經元中激勵值操作的結果會影響下一層的激勵值，一層一層之間激勵值的傳遞最後輸出判斷的結果。

各層傳遞的數學模型

其中每一層神經元中激勵值的傳遞方式，第一步是把該層每個神經元的值(a) 乘上藉由訓練所得到的 **權重(w)** 再全部加總起來。接著要設置一個觸發神經元啟動的閥值門檻，這裡稱為 **偏置(Bias)**，請將剛才的權重值總合減去偏置值。因為計算的結果可能為任何的數，但我們必須將這個結果壓縮限制在0 與1 之間，這裡就要透過一些函式進行處理，也就是所謂的 **激勵函式 (Activation Functions)**。

2.2 認識 Mnist 資料集

Mnist 資料集 (Modified National Institute of Standards and Technology database)，是由紐約大學 Yann LeCun 教授蒐集整理許多人0 到9 的手寫數字圖片所形成的資料集，其中包含了60000 筆的訓練資料，10000 筆的測試資料。在Mnist 資料集中，每一筆資料都是由 images (數字圖片) 和 labels (真實數字) 組成的單色圖片資料，很適合機器學習的初學者，練習建立模型、訓練和預測。



回首頁

2.2.1 下載與讀取 Mnist 資料集

下載 Mnist 資料集

在Python 中透過 Keras 就可以下載 Mnist 資料集，請先匯入 mnist 模組，再利用mnist 模組的 load_data 方法，即可載入資料，語法如下：

```
from keras.datasets import mnist
(train_feature, train_label), \
(test_feature, test_label) = mnist.load_data()
```

mnist.load_data() 第一次執行會將資料下載到使用者目錄下的<.keras\datasets>目錄中，檔名為<mnist.npz>。

讀取Mnist 資料集

載入資料後分別放在 (train_feature, train_label) 和(test_feature, test_label) 變數中，其中 (train_feature, train_label) 是訓練資料，(test_feature, test_label) 是測試資料，可以使用load_data() 函式讀入，語法如下：

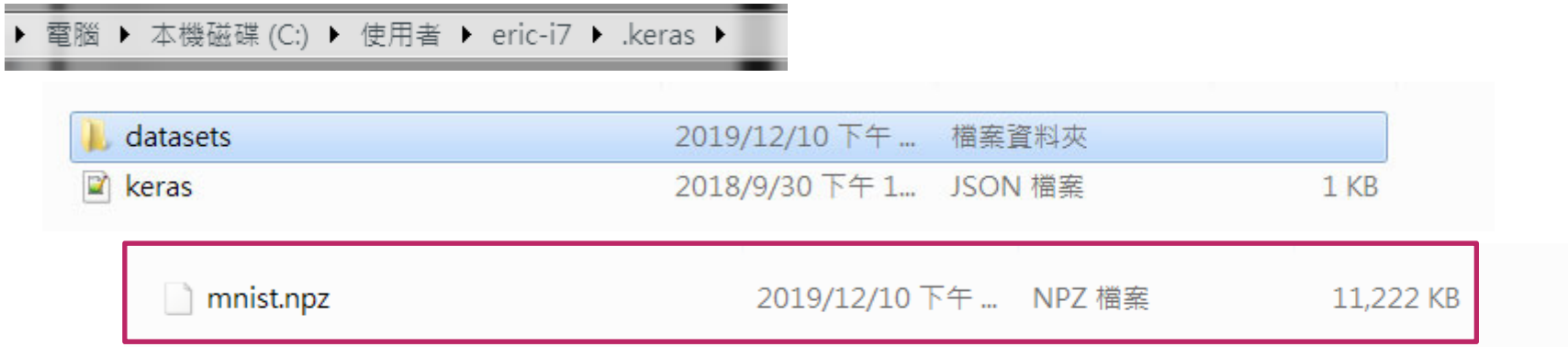
```
(train_feature, train_label),(test_feature, test_label) = mnist.load_data()
```

Prg1

```
In [*]: from keras.datasets import mnist
        (train_feature, train_label), (test_feature, test_label) = mnist.load_data()
```

```
Using TensorFlow backend.

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
 8757248/11490434 [=====================>........] - ETA: 2s
```

成功後使用者目錄下.keras目錄會多了datasets



Prg2 查看訓練資料

In [1]:
```
from keras.datasets import mnist

#讀取MNIST資料集
(train_feature, train_label), (test_feature, test_label) = mnist.load_data()
```

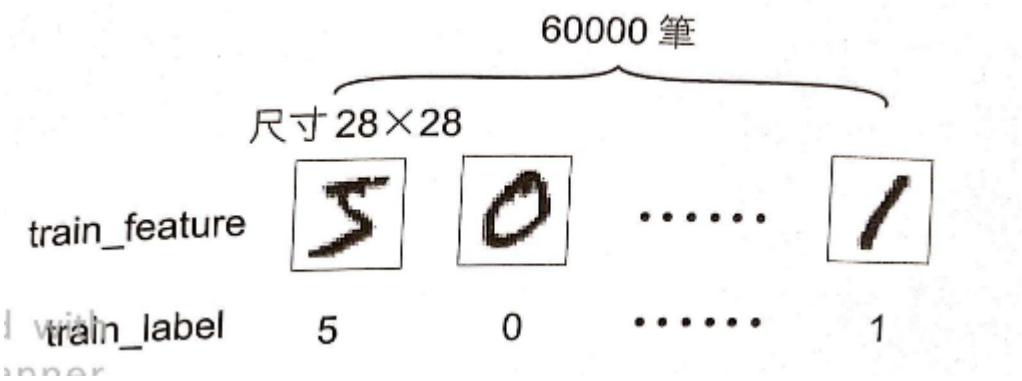
```
Using TensorFlow backend.
```

In [2]:
```
# 查看訓練資料
print(len(train_feature), len(train_label))   #60000 60000
```

```
60000 60000
```

In [4]:
```
# 查看維度
print(train_feature.shape, train_label.shape)   # (60000, 28, 28) (60000,)
```

```
(60000, 28, 28) (60000,)
```



Prg3 顯示訓練資料的圖片與值

In [1]:
```
#prg3 顯示訓練資料的圖片與值
from keras.datasets import mnist

#讀取MNIST資料集
(train_feature, train_label), (test_feature, test_label) = mnist.load_data()
```

```
Using TensorFlow backend.
```

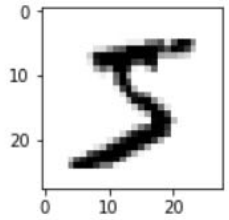
In [2]:
```
# 查看訓練資料
print(len(train_feature), len(train_label))   #60000 60000
```

```
60000 60000
```

In [3]:
```
# 查看維度
print(train_feature.shape, train_label.shape)   # (60000, 28, 28) (60000,)
```

```
(60000, 28, 28) (60000,)
```

In [7]:
```
import matplotlib.pyplot as plt
def show_image(image):
    fig = plt.gcf()
    fig.set_size_inches(2,2)  #數字圖片大小
    plt.imshow(image, cmap = 'binary')  #黑白灰階顯示
    plt.show()

show_image(train_feature[0])  #顯示訓練資料第1個數字
```



In [6]:
```
print(train_label[0])    #顯示第1個訓練資料圖片真實值
```

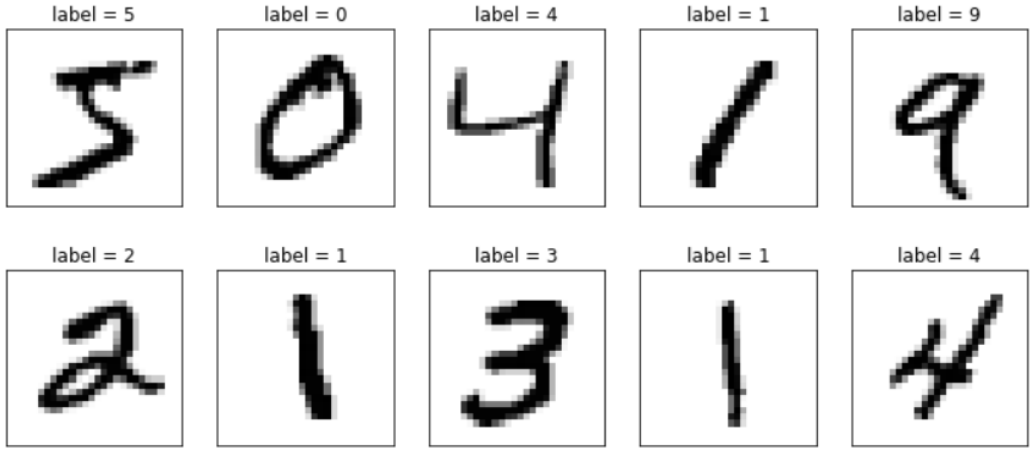
```
5
```

Prg4 顯示多張圖片資料

In [6]:
```
# 顯示多張資料副程式
def show_images_labels_predictions(images,labels,
                                   predictions,start_id,num=10):
    plt.gcf().set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i)
        #顯示黑白圖片
        ax.imshow(images[start_id], cmap='binary')

        # 有 AI 預測結果資料, 才在標題顯示預測結果
        if( len(predictions) > 0 ) :
            title = 'ai = ' + str(predictions[i])
            # 預測正確顯示(o), 錯誤顯示(x)
            title += (' (o)' if predictions[i]==labels[i] else ' (x)')
            title += '\nlabel = ' + str(labels[i])
        # 沒有 AI 預測結果資料, 只在標題顯示真實數值
        else :
            title = 'label = ' + str(labels[i])

        # X, Y 軸不顯示刻度
        ax.set_title(title,fontsize=12)
        ax.set_xticks([]);ax.set_yticks([])
        start_id+=1
    plt.show()
```

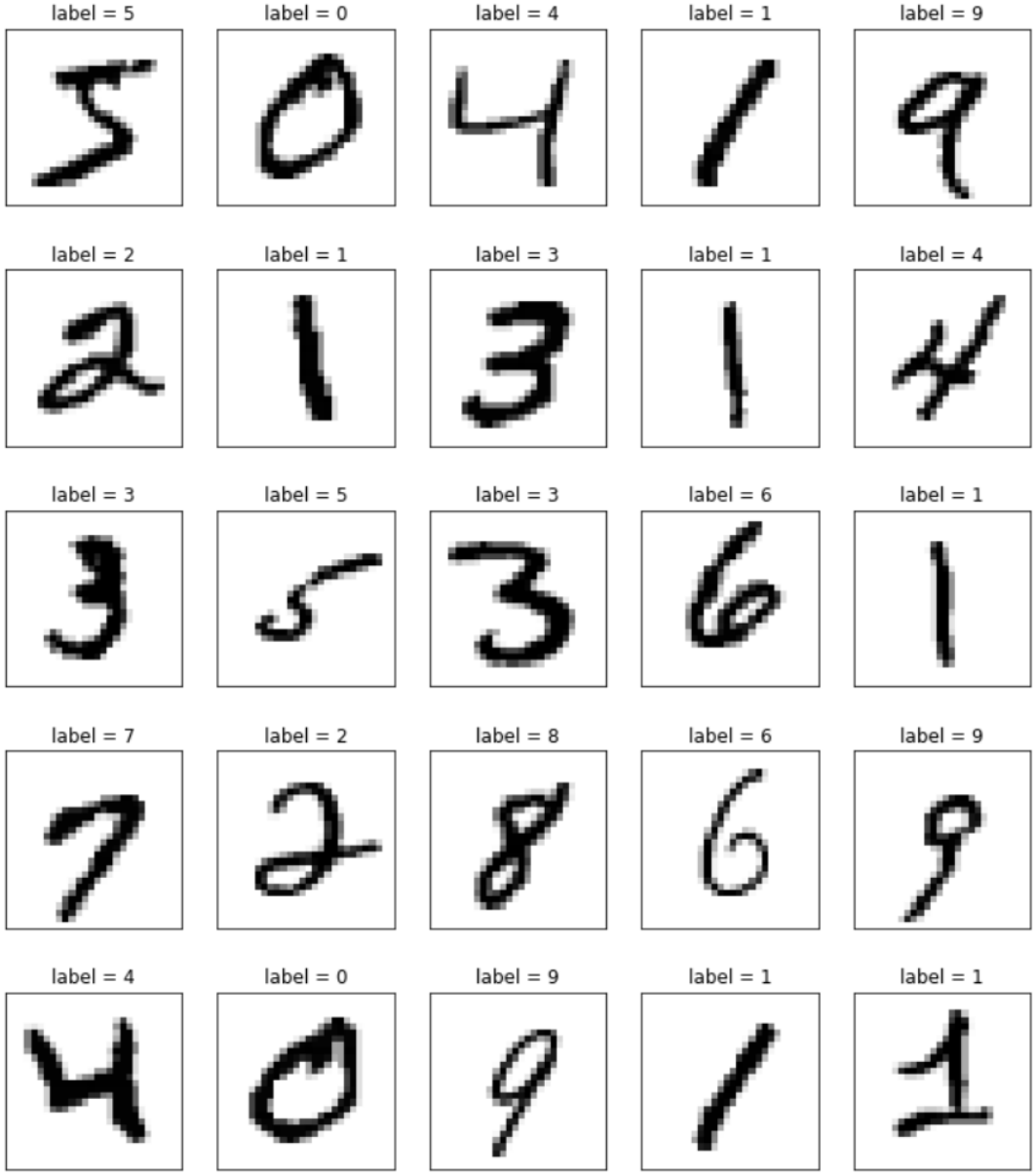
最多25張照片

In [7]:
```
# 顯示訓練資料前10筆資料
show_images_labels_predictions(train_feature, train_label, [], 0, 10)
```



Exercise#1

顯示前25張訓練資料



2.2.2 查看訓練資料

顯示訓練資料內容

訓練資料是由單色的數字圖片(images) 和數字圖片真實值 (labels) 所組成，兩者都是 60000 筆，可以使用 len() 函式查看資料的長度：

```
print(len(train_feature),len(train_label))   # 60000 60000
```

每一筆單色的數字圖片是一個 28*28 的圖片檔，真實值則是一個0~9 的數字。可以使用 shape 屬性查看其維度：

```
print(train_feature.shape,train_label.shape)#(60000, 28, 28)(60000,)
```

顯示訓練資料的圖片與值

以下利用自訂程序 show_image 以黑白灰階來顯示 2*2 吋大小的數字圖片，參數image 是指定要顯示的圖片。

```
import matplotlib.pyplot as plt
def show_image(image):
    fig = plt.gcf()
    fig.set_size_inches(2, 2) # 數字圖片大小
    plt.imshow(image, cmap='binary') # 黑白灰階顯示
    plt.show()
```

2.3 多層感知器模型資料預處理

2.3.1 Feature 資料預處理

Feature (數字圖片特徵值) 就是模型中輸入神經元輸入的資料，每一個 Mnist 數字圖片都是一張 28*28 的二維向量圖片，必須轉換為 784 個 float 數字的一維向量，並將 float 數字標準化，當作輸入神經元的輸入，才能增加模型訓練的效率。因此，總共需要 784 個輸入。

image 轉換

1. 以 reshape() 函式將 28*28 的數字圖片轉換為 784 個數字的一維向量，再以astype 將每個數字都轉換為 float 數字。

```
train_feature_vector =train_feature.reshape(len(train_feature),
  784).astype('float32')
test_feature_vector = test_feature.reshape(len( test_feature),
  784).astype('float32')
```

回首頁

Prg5 # 以reshape()函數將28*28的數字圖片轉換成784個數字的一維向量，再以astype將每個數字都轉換為float數字

In [9]:
```python
# 以reshape()函數將28*28的數字圖片轉換成784個數字的一維向量，再以astype將每個數字都轉換為float數字
train_feature_vector = train_feature.reshape(len(train_feature),784).astype('float32')
test_feature_vector = test_feature.reshape(len(test_feature),784).astype('float32')

#查看資料
print(train_feature_vector.shape, test_feature_vector.shape)
```

```
(60000, 784) (10000, 784)
```

In [10]:
```python
# 顯示第1筆image資料內容，顯示0~255的浮點數，數字代表圖片中美一個點的灰階值
print(train_feature_vector[0])
```

```
[   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    3.   18.
   18.   18.  126.  136.  175.   26.  166.  255.  247.  127.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.   30.   36.   94.  154.  170.  253.
  253.  253.  253.  253.  225.  172.  253.  242.  195.   64.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.   49.  238.  253.  253.  253.  253.  253.
  253.  253.  253.  251.   93.   82.   82.   56.   39.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.   18.  219.  253.  253.  253.  253.  253.
  198.  182.  247.  241.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.   80.  156.  107.  253.  253.  205.
   11.    0.   43.  154.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.   14.    1.  154.  253.   90.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  139.  253.  190.
    2.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   11.  190.  253.
   70.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   35.  241.
  225.  160.  108.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   81.
  240.  253.  253.  119.   25.    0.    0.    0.    0.    0.    0.    0.    0.    0.
```

image 標準化

1. 將 0~255 的數字，除以 255 得到 0~1 之間浮點數，稱為標準化(Normalize)，標準化之後可以提高模型預測的準確度，增加訓練效率。

```
train_feature_normalize = train_feature_vector/255
test_feature_normalize = test_feature_vector/255
```

Prg6 資料標準化 (每一個點變成0~1之間數字)

In [10]:
```
# Image標準化
train_feature_normalize = train_feature_vector/255
test_feature_normalize = test_feature_vector/255

#顯示第1筆的image正規化
print(train_feature_normalize[0])
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215687
0.93333334 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.9843137  0.3647059  0.32156864
0.32156864 0.21960784 0.15294118 0.         0.         0.
```

2.3.2 Label 資料預處理

Label (數字圖片真實值) 原本是 0~9 的數字，為了增加模型效率，神經元輸出比較常採用 One-Hot Encoding 編碼 (一位有效編碼) 的方式，輸出的所有位元中只有 1 個是 1，其餘都是 0。使用 np_utils.to_categorical 方法可以將數字轉換成 One-Hot Encoding 編碼。

0~9 的數字的 One-Hot Encoding 編碼如下：

真實值	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
…										
9	0	0	0	0	0	0	0	0	0	1

1. 首先顯示 Label 真實值，方便前後對照。例如：訓練資料 Label 的前 5 筆。

```
print(train_label[0:5])  # [5 0 4 1 9]
```

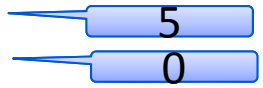
Prg 7 One-Hot Encoding

In [11]:
```
# One-Hot Encoding
print(train_label[0:5])   #顯示訓練資料前五筆資料
```

```
[5 0 4 1 9]
```

In [13]:
```
import numpy as np
from keras.utils import np_utils
np.random.seed(10)

train_label_onehot = np_utils.to_categorical(train_label)
test_label_onehot = np_utils.to_categorical(test_label)

print(train_label_onehot[0:5])
```

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



2.4 多層感知器實戰：Mnist 手寫數字圖片辨識

2.4.1 多層感知器訓練和預測

訓練 (Train)

Mnist 資料集共有 60000 筆訓練資料，將訓練資料的 Feature(數字圖片特徵值)和 Label(數字真值實) 都先經過預處理，作為多層感知器的輸入、輸出，然後進行模型訓練。

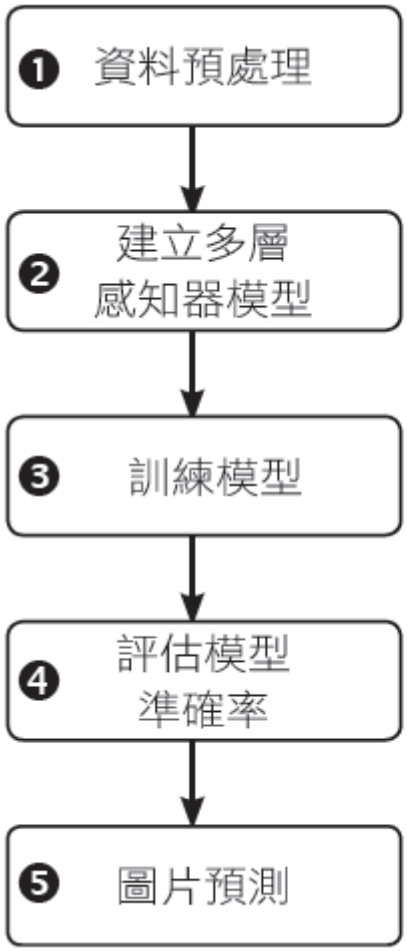
預測 (Predict)

模型訓練完成以後就可以用來作預測，將要預測的數字圖片，先經過預處理變成Feature(數字圖片特徵值)，就可送給模型作預測，得到 0~9 數字的預測結果。

也可以將訓練好的模型儲存起來，以後就可以不再重複訓練，如果要在其他程式中使用，只要載入儲存的模型就可以進行預測。

回首頁

2.4.2 多層感知器手寫數字圖片辨識流程



將 Feature 特徵值換為 784 個 float 數字的 1 維向量，並將 float 數字標準化，將 Label 轉換為 One-Hot Encoding 編碼。

建立含有輸入、隱藏、輸出層的模型。

以訓練資料的 Feature 和 Label，執行指定次數的訓練。

使用測試資料，評估模型準確率。

以訓練完成的模型，對想要測試的數字圖片進行預測。

2.4.3 資料預處理

載入資料

匯入 mnist 模組，以 mnist 模組的 load_data 方法，載入資料。

```
from keras.datasets import mnist
(train_feature, train_label),\
(test_feature, test_label) = mnist.load_data()
```

Feature 特徵值轉換

將 Feature 特徵值轉換為 784 個 float 數字的 1 維向量。

```
train_feature_vector = train_feature.reshape(len(train_feature), 784)
                       .astype('float32')
test_feature_vector = test_feature.reshape(len( test_feature), 784)
                      .astype('float32')
```

Feature 特徵值標準化

將 0~255 的數字，除以 255 得到 0~1 之間浮點數，稱為標準化(Normalize)，以提高模型預測的準確度。

```
train_feature_normalize = train_feature_vector/255
test_feature_normalize = test_feature_vector/255
```

label 轉換為 One-Hot Encoding 編碼

以 to_categorical 方法將訓練和測試的 Label 轉換為 One-Hot Encoding 編碼。

```
train_label_onehot = np_utils.to_categorical(train_label)
test_label_onehot = np_utils.to_categorical(test_label)
```

2.4.4 建立多層感知器模型

多層感知器模型

1. **輸入層**：每一個 Mnist 數字圖片是一張 28*28 的 2 維向量圖片，再以 reshape將 2 維轉換為 784 個 float 數字的 1 維向量，並將 float 數字標準化，當作輸入神經元的輸入，因此，總共需要 784 個輸入神經元。
2. **隱藏層**：輸入層和輸出層中間的內部神經元，稱為隱藏層，隱藏層可以只有 1層，也可以是多層，甚至在隱藏層間再加入Drop Out (拋棄層)。
3. **輸出層**：預測的結果就是輸出層，就是 0~9 共有 10 個數字，代表有 10 個輸出神經元。

建立 Sequential 模型

匯入 Sequential 模組後即可以 Sequential 建立模型。

```
from keras.models import Sequential
model = Sequential()
```

建立輸入層和隱藏層

```
from keras.layers import Dense
model.add(Dense(units=256,
                input_dim=784,
                kernel_initializer='normal',
                activation='relu'))
```

建立輸出層

```
model.add(Dense(units=10,
                kernel_initializer='normal',
                activation='softmax'))
```

2.4.5 訓練模型

設定模型的訓練方式

訓練中必須以 compile 方法定義 Loss 損失函式、Optimizer 最佳化方法和 metrics評估準確率方法，Keras 提供許多內建的方法，可以當作訓練參數。

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

進行訓練

fit 方法可以進行訓練，訓練時必須設定訓練資料和標籤。語法如下：

```
model.fit(x=特徵值, y=標籤, validation_split = 驗證資料百分比,
          epochs=訓練次數, batch_size=每批次有多少筆,verbose = n)
```

verbose:設定是否顯示訓練過程。0:不顯示 1:詳細顯示 2:簡易顯示

Prg8 建立模型

In [13]:
```
#建立模型
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense

#建立模型
model = Sequential()
#輸入層：784, 隱藏層：256，輸出層：10
model.add(Dense(units=256,
                input_dim=784,
                kernel_initializer='normal',
                activation='relu'))
model.add(Dense(units=10,
                kernel_initializer='normal',
                activation='softmax'))
#定義訓練方式
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

#以(train_feature_normalize,train_label_onehot)資料訓練，
#訓練資料保留 20% 作驗證,訓練10次、每批次讀取200筆資料，顯示簡易訓練過程
train_history =model.fit(x=train_feature_normalize,
                         y=train_label_onehot,validation_split=0.2,
                         epochs=10, batch_size=200,verbose=2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
 - 1s - loss: 0.4377 - acc: 0.8830 - val_loss: 0.2181 - val_acc: 0.9410
Epoch 2/10
 - 1s - loss: 0.1909 - acc: 0.9454 - val_loss: 0.1557 - val_acc: 0.9559
Epoch 3/10
 - 1s - loss: 0.1356 - acc: 0.9617 - val_loss: 0.1263 - val_acc: 0.9648
Epoch 4/10
 - 1s - loss: 0.1028 - acc: 0.9701 - val_loss: 0.1121 - val_acc: 0.9681
Epoch 5/10
 - 1s - loss: 0.0811 - acc: 0.9773 - val_loss: 0.0987 - val_acc: 0.9715
Epoch 6/10
 - 1s - loss: 0.0659 - acc: 0.9818 - val_loss: 0.0936 - val_acc: 0.9726
Epoch 7/10
 - 1s - loss: 0.0543 - acc: 0.9854 - val_loss: 0.0912 - val_acc: 0.9737
Epoch 8/10
 - 1s - loss: 0.0458 - acc: 0.9878 - val_loss: 0.0833 - val_acc: 0.9763
Epoch 9/10
 - 1s - loss: 0.0379 - acc: 0.9905 - val_loss: 0.0824 - val_acc: 0.9757
Epoch 10/10
 - 1s - loss: 0.0315 - acc: 0.9919 - val_loss: 0.0808 - val_acc: 0.9763
```

2.4.6 評估準確率

evaluate 方法可以評估模型的損失函式誤差值和準確率，它會傳回串列，第 0 個元素為損失函式誤差值，第 1 個元素為準確率。

例如：使用測試資料評估模型的準確率。

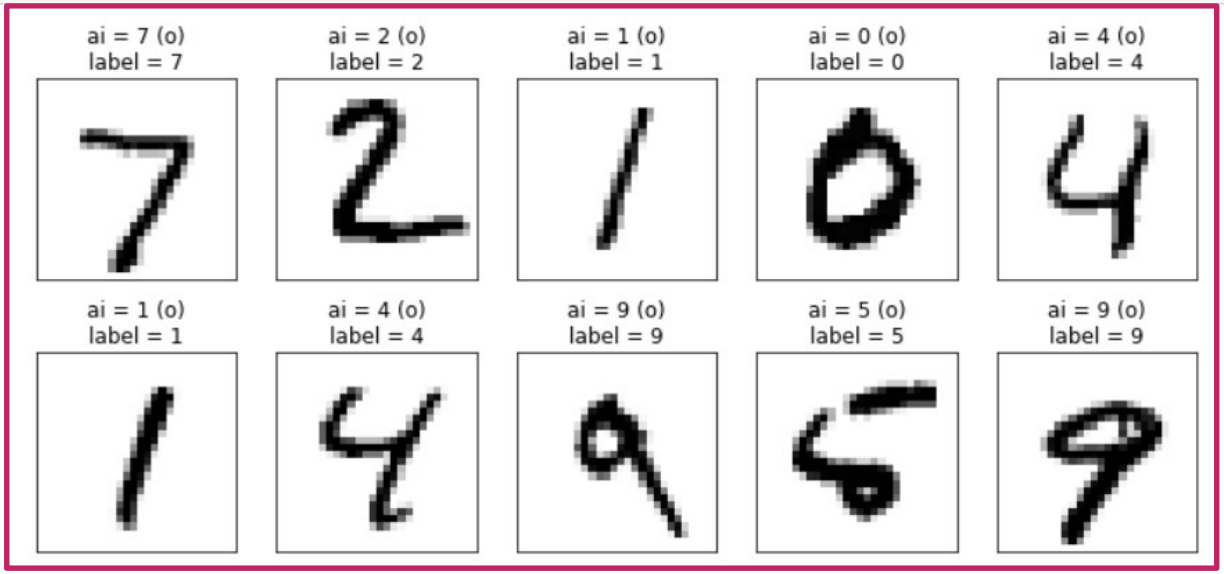
```
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n準確率 =',scores[1])
```

Prg8#2 評估準確度與顯示圖形

In [14]:
```
#評估準確率
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n準確率=',scores[1])
```

```
10000/10000 [==============================] - 0s 30us/step

準確率= 0.9762
```

In [15]:
```
#預測
prediction=model.predict_classes(test_feature_normalize)

#顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature,test_label,prediction,0)
```



2.4.7 進行預測

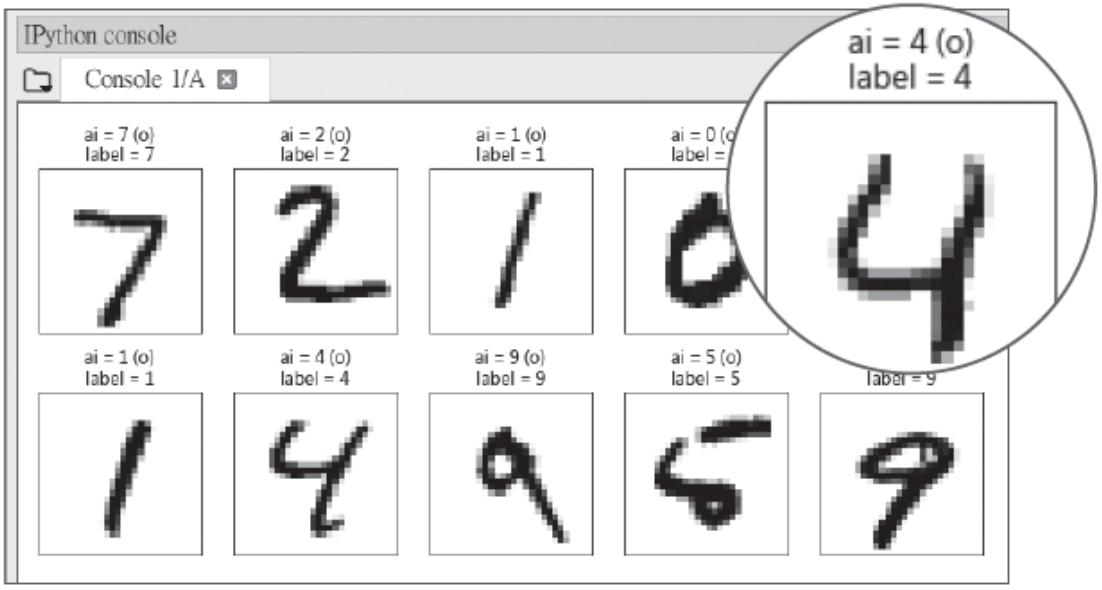
訓練好的模型，就可以用 predict_classes 方法進行預測，本例是以測試資料將其特徵值標準化後的 test_feature_normalize 作預測。

```
prediction=model.predict_classes(test_feature_normalize)
```

以下是顯示訓練好的模型對Mnist 資料集前 10 筆預測的結果。

```
show_images_labels_predictions(test_feature,test_label,prediction,0)
```

結果中ai 是由程式所辨別的數字，下方是Mnist 資料集前10 筆的Label與圖片資料，辨識的結果相當理想呢！



Exercise#2

試著增加訓練次數10->100與batch_size，觀察正確率是否提升？

In [14]:
```
#評估準確率
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n準確率=',scores[1])
```

```
10000/10000 [==============================] - 0s 49us/step

準確率= 0.9793
```

2.5 模型儲存和載入

2.5.1 模型儲存

Keras 使用HDF5 檔案系統來儲存模型，模型儲存一般使用 .h5 為副檔名，語法：

```
model.save(檔名)
```

2.5.2 載入模型

當訓練資料很龐大時，訓練一次可能需要很長的時間，這時就可以直接載入已訓練好的模型作為預測，減少重複訓練的時間。

記得先以 from keras.models import load_model 匯入相關模組，再利用以下的語法載入模型：

```
load_model(檔名)
```

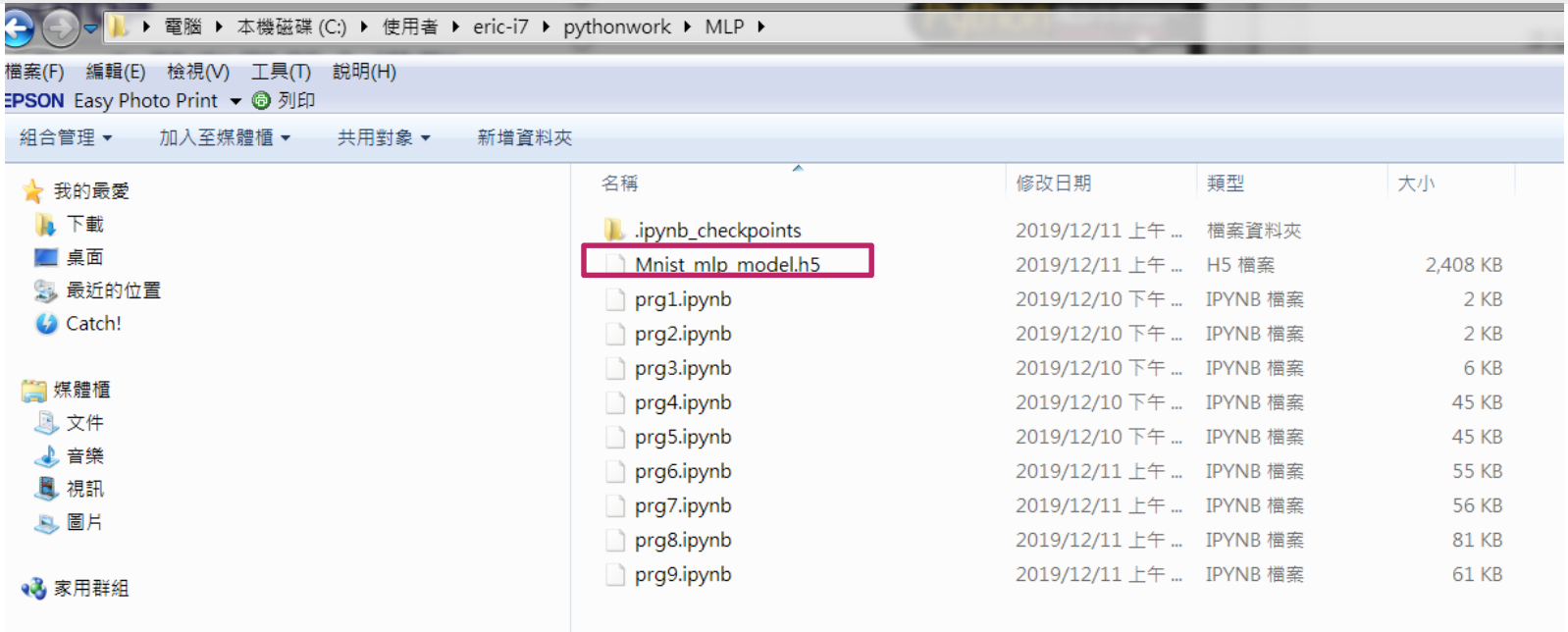
回首頁

Prg9 儲存與載入模型

程式目錄中會多了一個Mnist_mlp_model.h5檔案

```
In [16]: # 將模型儲存在 .HDF5檔案中
         model.save('Mnist_mlp_model.h5')
         print("Mnist_mlp_model.h5 模型儲存完畢")
         del model

         #載入模型
         #load_model('Mnist_ml-Model.h5')
```

```
Mnist_mlp_model.h5 模型儲存完畢
```



回首頁

Prg10 模型載入與預測


```
#Prg10 載入預先訓練好的模型

import numpy as np
np.random.seed(10)
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.models import load_model

def show_images_labels_predictions(images,labels,
                                   predictions,start_id,num=10):
    plt.gcf().set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i)
        #顯示黑白圖片
        ax.imshow(images[start_id], cmap='binary')

        # 有 AI 預測結果資料, 才在標題顯示預測結果
        if( len(predictions) > 0 ) :
            title = 'ai = ' + str(predictions[i])
            # 預測正確顯示(o), 錯誤顯示(x)
            title += (' (o)' if predictions[i]==labels[i] else ' (x)')
            title += '\nlabel = ' + str(labels[i])
        # 沒有 AI 預測結果資料, 只在標題顯示真實數值
        else :
            title = 'label = ' + str(labels[i])

        # X, Y 軸不顯示刻度
        ax.set_title(title,fontsize=12)
        ax.set_xticks([]);ax.set_yticks([])
        start_id+=1
    plt.show()

#建立訓練資料和測試資料，包括訓練特徵集、訓練標籤和測試特徵集、測試標籤
(train_feature, train_label),\
(test_feature, test_label) = mnist.load_data()

#將 Features 特徵值換為 784個 float 數字的 1 維向量
test_feature_vector = test_feature.reshape(len( test_feature), 784).astype('float32')

#Features 特徵值標準化
test_feature_normalize = test_feature_vector/255

#從 HDF5 檔案中載入模型
print("載入模型 Mnist_mlp_model.h5")
model = load_model('Mnist_mlp_model.h5')

#預測
prediction=model.predict_classes(test_feature_normalize)

#顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature,test_label,prediction,0)
```

模型不需訓練，
載入訓練好模型

回首頁

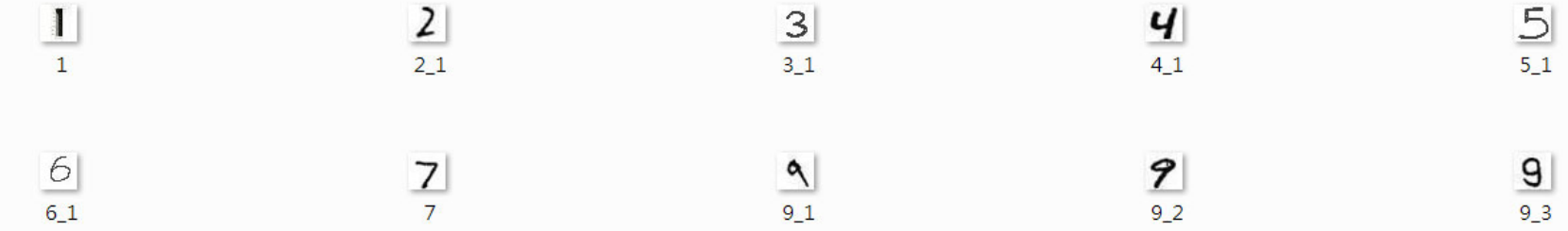
2.5.3 預測自己的數字圖片

安裝 opencv

本範例會使用到 opencv，如果尚未安裝請開啟 Anaconda Prompt 視窗安裝：

```
pip install opencv-python==3.4.3.18
```

自己的圖片需放在程式目錄的 imagedata目錄下



Prg11 預測自己的圖片

Part#1

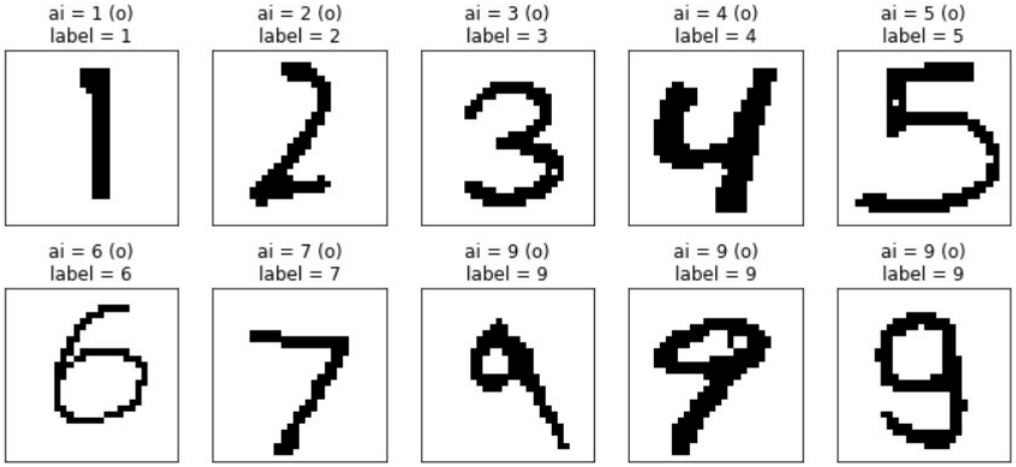
```
#prg11 預測自己的圖片

import numpy as np
np.random.seed(10)
import matplotlib.pyplot as plt
from keras.models import load_model
import glob,cv2

def show_images_labels_predictions(images,labels, predictions,start_id,num=10):
    plt.gcf().set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i)
        #顯示黑白圖片
        ax.imshow(images[start_id], cmap='binary')

        # 有 AI 預測結果資料, 才在標題顯示預測結果
        if( len(predictions) > 0 ) :
            title = 'ai = ' + str(predictions[i])
            # 預測正確顯示(o), 錯誤顯示(x)
            title += (' (o)' if predictions[i]==labels[i] else ' (x)')
            title += '\nlabel = ' + str(labels[i])
        # 沒有 AI 預測結果資料, 只在標題顯示真實數值
        else :
            title = 'label = ' + str(labels[i])

        # X, Y 軸不顯示刻度
        ax.set_title(title,fontsize=12)
        ax.set_xticks([]);ax.set_yticks([])
        start_id+=1
    plt.show()
```

GOTOP

Prg11 預測自己的圖片 Part# 2

```
#建立測試特徵集、測試標籤
files = glob.glob("imagedata\*.jpg" )
test_feature=[]
test_label=[]
for file in files:
    img=cv2.imread(file)
    img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)  #灰階
    _, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV) #轉為反相黑白
    test_feature.append(img)
    label=file[10:11]  # "imagedata\1.jpg" 第10個字元1為label
    test_label.append(int(label))

test_feature=np.array(test_feature) # 串列轉為矩陣
test_label=np.array(test_label)     # 串列轉為矩陣

#將 Features 特徵值換為 784個 float 數字的 1 維向量
test_feature_vector = test_feature.reshape(len( test_feature), 784).astype('float32')

#Features 特徵值標準化
test_feature_normalize = test_feature_vector/255

#從 HDF5 檔案中載入模型
print("載入模型 Mnist_mlp_model.h5")
model = load_model('Mnist_mlp_model.h5')

#預測
prediction=model.predict_classes(test_feature_normalize)

#顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature,test_label,prediction,0,len(test_feature))
```

```
載入模型 Mnist_mlp_model.h5
```

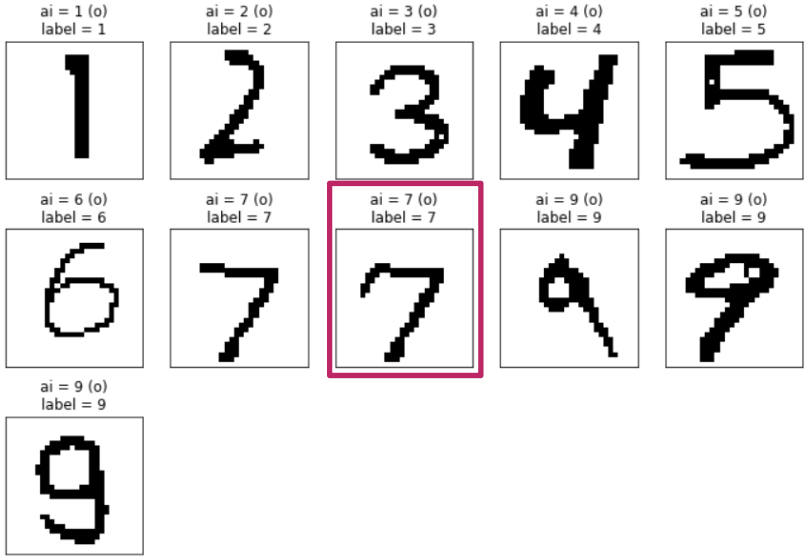


Exercise#2

建立自己的手寫圖片，放在imagedata目錄下讓程式預測，並顯示預測結果

1. 建立一個7_2.jpg手寫圖形 (尺寸28x28)

2. 預測結果



2.6 模型權重的儲存和載入

模型權重儲存

這種模型累積訓練的方式必須使用模型權重來達成，權重是模型的參數(但不包括模型)，可以 save_weights 方法儲存模型權重，語法：

```
model.save_weights(檔名)
```

模型權重載入

只要載入已儲存的模型權重，就會取回上次的模型參數，這樣模型就會繼續上次的訓練，達到累積的效果。使用 load_weights 方法可以載入模型權重，語法：

```
model.load_weights(檔名)
```

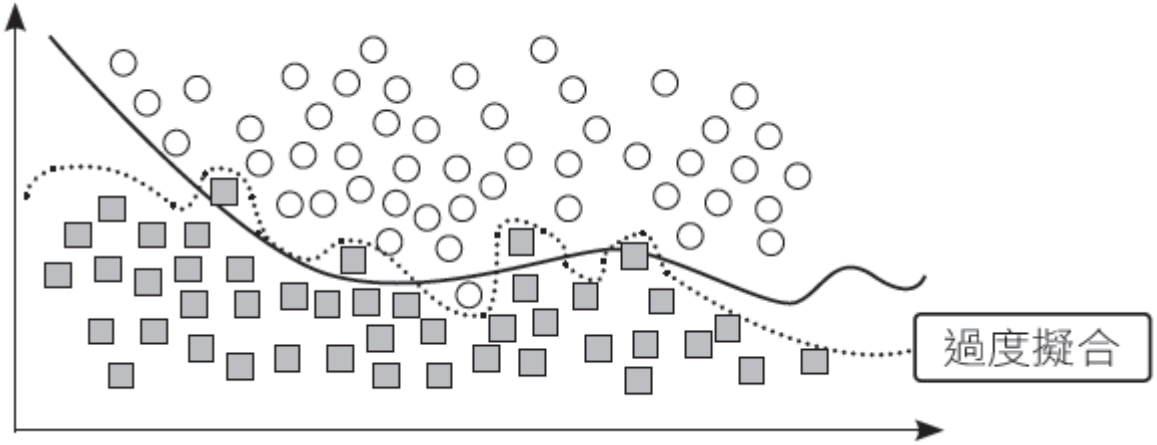
回首頁

2.7 建立多個隱藏層

2.7.1 過渡擬合 (Overfitting)

如果在訓練時 acc 訓練的準確度增加，但是 val_acc 驗證的準確度卻沒有增加，這可能就是過渡擬合 (Overfitting) 的現象。什麼是過渡擬合呢?

下圖中黑色實線曲線是我們訓練後希望找到的較佳曲線，但因為訓練太少或訓練過久，因過渡擬合得到虛線曲線。



2.7.2 加入拋棄層(DropOut) 避免過渡擬合

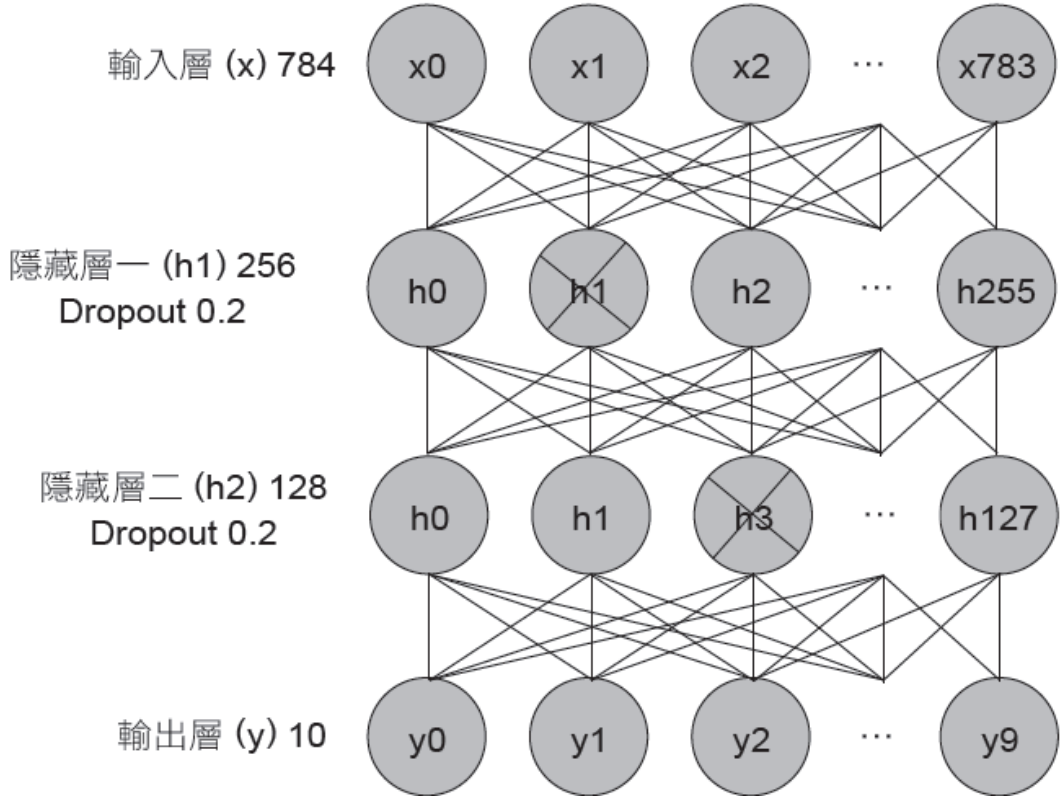
只要加入適當的拋棄層 (DropOut)，就可解決過渡擬合的問題，建立時必須以 from keras.layers import DropOut 匯入模組。語法：

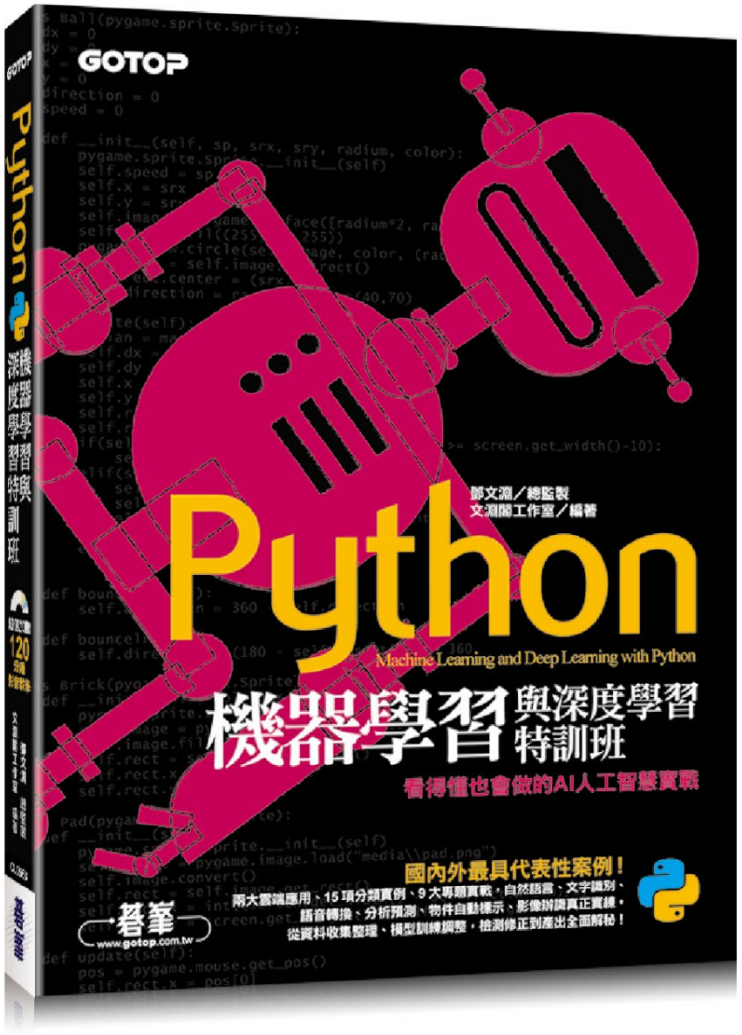
```
model.add(Dropout(放棄百分比))
```

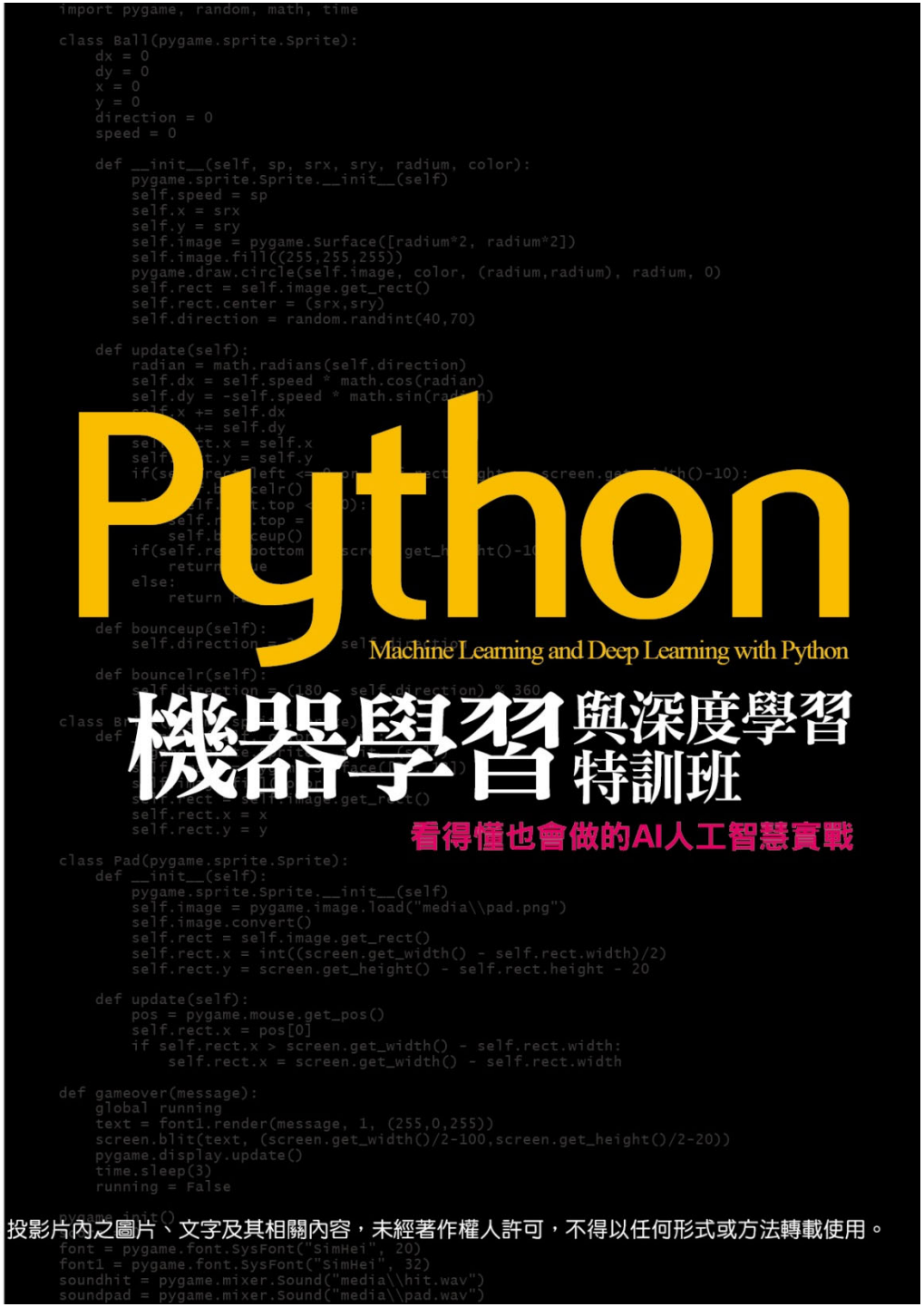
2.7.3 建立含有多個隱藏層的多層感知器

可以加入多層的隱藏層，並在隱藏層中加入適度的拋棄層避免過渡擬合的現象，當然這樣會花費較多的時間，但準確度會明顯提高

我們以加入兩層隱藏層為例，第一層有 256 個神經元，並加入 DropOut(0.2) 的拋棄層，第二層有 128 個神經元，並加入 DropOut(0.2) 的拋棄層。如下圖：







碁峰資訊