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

深度學習理論與實作
CH2 神經網路的數學概念

陳俊豪 教授

Outline

2-1 初探神經網路：第一隻神經網路

2-2 神經網路的資料表示法：張量Tensor

2-3 神經網路的工具：張量運算

2-4 神經網路的引擎：以梯度為基礎的最佳化

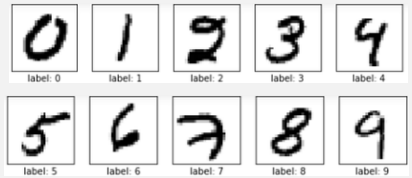
2-5 重新檢視我們的第一個例子

2-1 前言

- ✓ 使用神經網路辨識手寫數字
- ✓ 如何判斷學習成果?

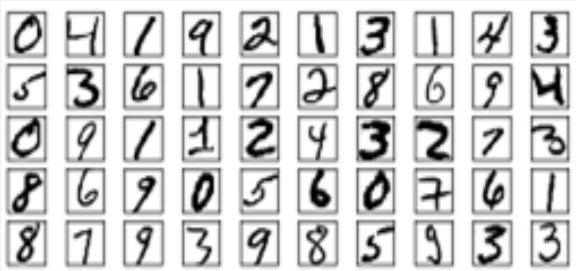
類別 (class)

數字0~9，一共10類



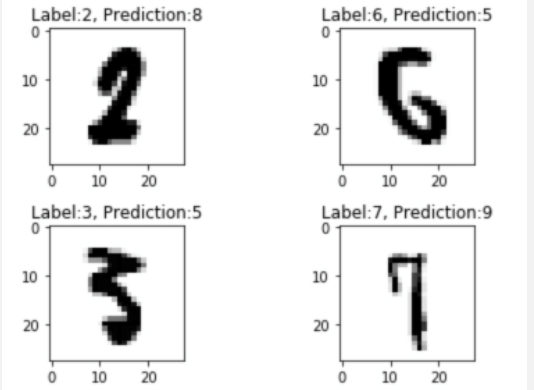
樣本 (samples)

輸入的資料 input data



標籤 (label)

人工標註標籤(答案)

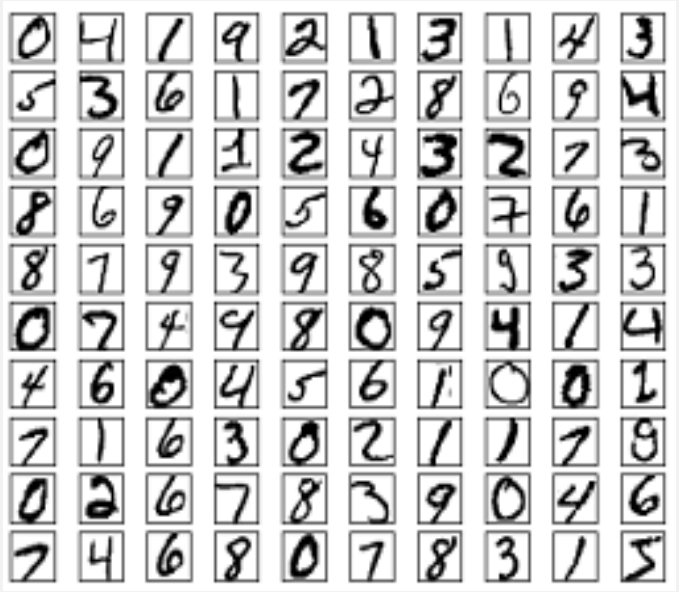


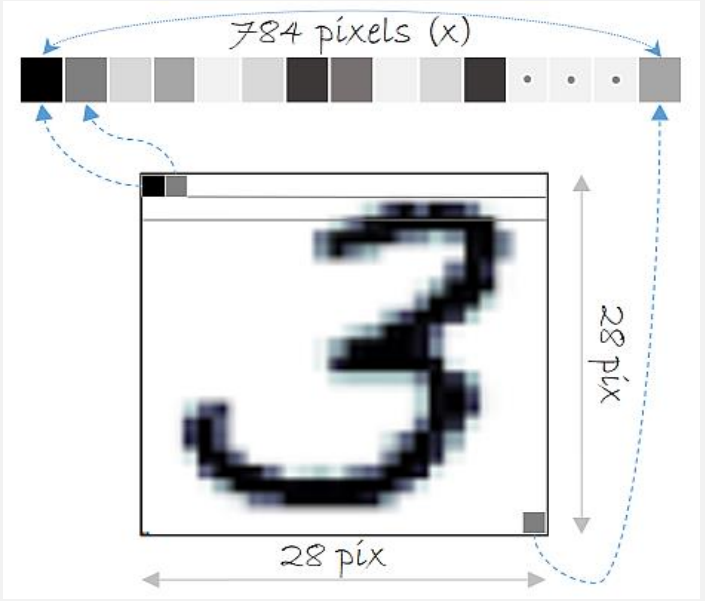
如果機器分類的結果與標籤的一致性很高，代表學習成功；反之則視為學習失敗

2-1-1 MNIST資料集

✓ MNIST

- 美國國家標準技術學院(National Institute of Standards and Technology)建立的**手寫數字的圖像資料集**
- 資料集包含 60,000 張訓練圖片和 10,000 張測試圖片
- 每個圖片大小是 28 * 28 像素





2-1-1 MNIST資料集

✓在 Keras 中載入 MNIST 資料集

```
from keras.datasets import mnist    #從 keras 的 datasets 匯入 mnist 資料集
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
                                    #用 mnist.load_data() 取得 mnist 資料集, 並存 (打包) 成 tuple，此tuple又包含兩個tuple
```

補充：
此處 train_images、train_labels、test_images、test_labels
都是 Numpy 的 ndarray 物件

2-1-1 MNIST資料集

✓了解訓練資料

```
train_images.shape

#train_image 為 NumPy 的 ndarray 物件
#train_image 的 shape 屬性為 3 軸, 60000 維 x28 維 x28 維
```

Out[] : (60000, 28, 28)

補充：shape 是 ndarray 物件的一個屬性，可以顯示該 ndarray 的維度結構

2-1-1 MNIST資料集

✓了解訓練資料

```
len(train_labels)
#標籤也有 60,000 個
```

```
Out[] : 60000
```

```
train_labels
#標籤是 0-9 之間的數字, 資料型別為 uint8
```

```
Out[] : array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

2-1-1 MNIST資料集

✓了解測試資料

```
test_images.shape
#shape 為 3 軸, 10000 維 x28 維 x28 維
```

Out[] : (10000, 28, 28)

```
len(test_labels)
#標籤也有 10,000 個
```

Out[] : 10000

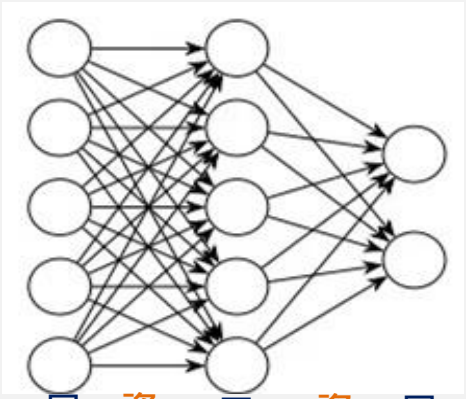
```
test_labels
#標籤是 0-9 之間的數字, 資料型別為 uint8
```

Out[] : array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)

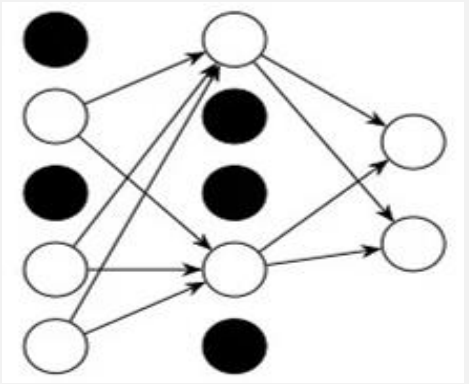
2-1-2 建構神經網路模型

- ✓ NN的基本元件是層 (layer)，每層都會執行資料萃取(data distillation)
- ✓ 層 = 資料處理的模組 = 資料過濾器
  - 密集層 = 全連接(fully connected)：前後層中的神經元全部彼此連接一起
  - 稀疏層 : 前後層中的神經元未全部彼此連接一起，甚至未連結

密集層



稀疏層



2-1-2 建構神經網路模型 – 建構模型

✓ 神經網路架構

```
from keras import models      #匯入套件
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

補充：
1. Dense Layer 為密集層也稱為全連接層(fully connected)，為兩個密集層
2. activation 為激活函數，是一個感知器的開關，決定輸出值的大小與正負
3. Softmax函式，或稱「歸一化指數函式」，是邏輯函式的一種推廣。它能將一個含任意實數的K維向量 「壓縮」到另一個K維實向量 中，使得每一個元素的範圍都在之間，並且所有元素的和為1

2-1-2 建構神經網路模型 – 建構模型

✓ 編譯步驟：建構好一隻神經網路！

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

補充：

optimizer(優化器)：神經網路根據其輸入資料及損失函數值而自行更新的機制

Loss function(損失函數)：用以衡量此神經網路在訓練上的表現，以及引導網路朝向正確的方向修正

Metrics(評量準測)：此範例關注點為數字的辨識度

2-1-2 建構神經網路模型 – 資料集

✓ 準備圖片資料

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
#reshape 和 astype是 Numpy陣列的方法
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

補充：
在處理前，圖片是以 uint8 型別、數值介於[0、255]儲存於(60000、28、28)的陣列中，我們將它轉換為 (float32) 型別的 (60000,28*28) 陣列，數值介於 0跟 1 之間 (除以畫素最大值 255 將圖片正規化)

✓ 準備標籤

```
from keras.utils import to_categorical
#對標籤進行分類編碼
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

2-1-2 建構神經網路模型 – 訓練

✓ 訓練神經網路模型

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Out[] : Epoch 1/5
60000/60000 [==========] - 5s 80us/step - loss: 0.2563 - accuracy: 0.9259
Epoch 2/5
60000/60000 [==========] - 4s 74us/step - loss: 0.1023 - accuracy: 0.9699
Epoch 3/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0674 - accuracy: 0.9798
Epoch 4/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0486 - accuracy: 0.9854
Epoch 5/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0372 - accuracy: 0.9892
```

補充 : loss損失值；accuracy正確率

2-1-2 建構神經網路模型 – 測試

✓ **測試**神經網路模型

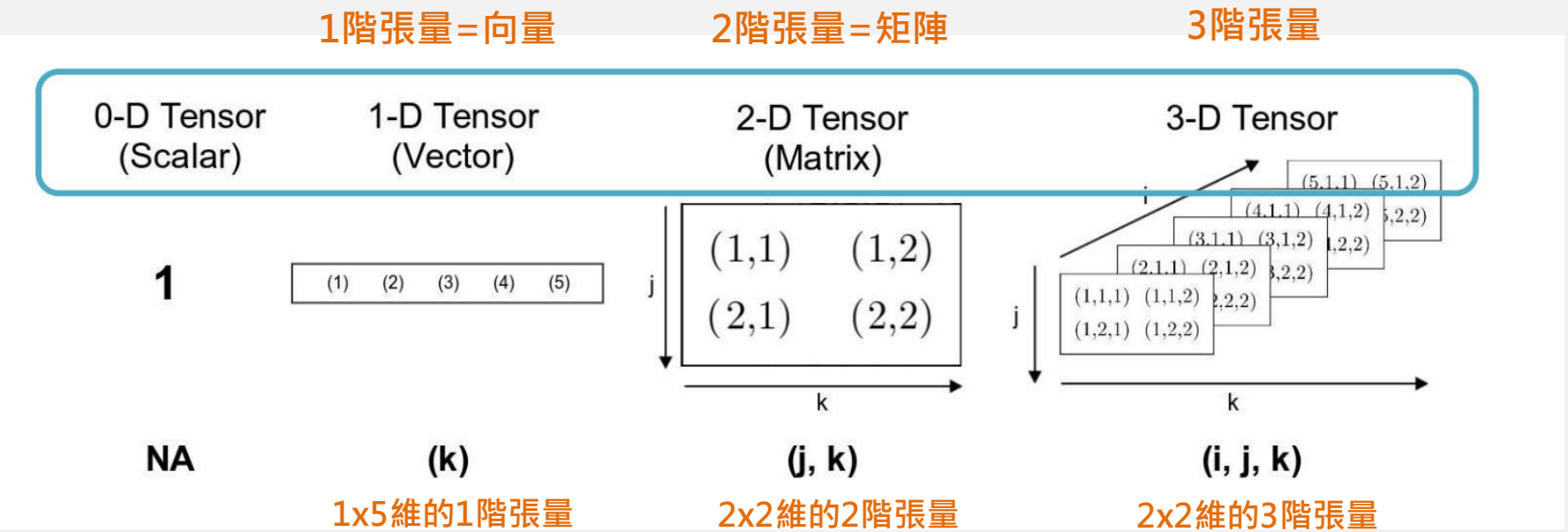
```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

Out[] :test_acc: 0.9785

補充：
測試資料集的正確率97.8%，與訓練集資料的正確率98.9%之間的差距，就是過度配適(overfitting)，意指機器學習模型對新資料的表現比訓練資料集來的差。

2-2 神經網路的資料表示法

✓張量tensor = 多維的Numpy陣列，是一個資料容器，專儲存數值資料



✓每一階所含的元素個數，為該階的維度(dimention)

2-2-1 純量 (0D 張量)

✓ 純量(scalar)：只包含一個數值的張量，稱「純量張量」、「0階張量」、「0軸張量」、「0D張量」

✓在張量中，階=軸，如：純量張量會顯示0個軸 (ndim值為0)

```
import numpy as np
x = np.array(12)      #用 12 這個數值去建一個張量
x                     #看看張量的內容
```

```
Out[] : array(12)     #原來 Numpy 的 array 就是 tensor
```

```
x.ndim                #看看 ndim 屬性 (就是階數)
```

```
Out[] : 0             #ndim 為 0, 是 0 階張量 (純量)
```

2-2-2 向量 (1D 張量)

✓ 向量(vector)是由一組數值排列而成的陣列，為1D張量，只有一軸

```
x = np.array([12, 3, 6, 14, 7])    #用 array() 建一個 5 維的 1 D 張量
x                                  #看看張量的內容
```

```
Out[] : array([12,  3,  6, 14,  7])  #是一個含有 4 個元素 (4 維) 的 1D 張量
```

```
x.ndim        #看看 ndim 屬性 (就是階數)
```

```
Out[] : 1      #ndim 為 1, 是 1 階張量 (向量)
```

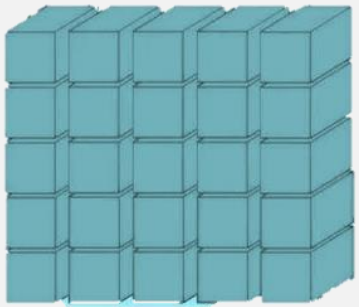
✓ 若一個向量有5個元素，稱為5維向量

✓5D向量只有一個軸，軸上有5個維度

✓5D張量有5個軸，不要將5D向量與5D張量搞混了!

✓每個軸有多少維度，是由array()來建立與決定的

5D向量



5D張量

2-2-3 矩陣 (2D 張量)

✓ 由一組項量組成的陣列就是一個矩陣(matrix)，又稱「2D張量」

✓矩陣上有2個軸，通常稱為列(rows)和行(columns)

```
x = np.array([[5, 78, 2, 34, 0 ],   #用 array() 建一個 3x5 (維) 的 2D 張量，我們可以想成這是由3個5維
              [6, 79, 3, 35, 1],    向量組成的
              [7, 80, 4, 36, 2]])
x.ndim      #看看 ndim 屬性 (就是階數)
```

```
Out[] : 2                          #ndim為2，是2階張量
```

2-2-4 3D 張量、高階張量

✓ 如果將多個矩陣包在一個新陣列中，將得到一個3D張量，可視其為一個數字立方體

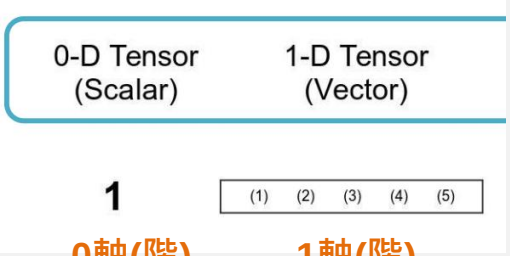
```
x = np.array([[[5, 78, 2, 34, 0],   #用 array() 建一個 3x3x5 (維) 的 3D 張量，可以想成是由3個
                [6, 79, 3, 35, 1],   3X5維的2D張量組成
                [7, 80, 4, 36, 2]],
               [[5, 78, 2, 34, 0],
                [6, 79, 3, 35, 1],
                [7, 80, 4, 36, 2]],
               [[5, 78, 2, 34, 0],
                [6, 79, 3, 35, 1],
                [7, 80, 4, 36, 2]]])
x.ndim   #看看 ndim 屬性 (就是階數)
```

```
Out[] : 3                    #ndim 為 3, 是 3 階張量
```

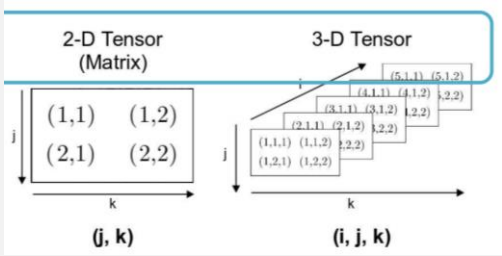
2-2-5 張量的關鍵屬性

✓ 張量是由三個關鍵屬性所決定

軸的數量(階數)



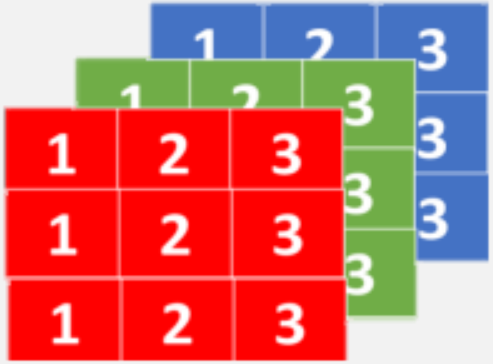
0軸(階) 1軸(階)



2軸(階) 3軸(階)

形狀(shape)

(3x3x3)



Python表達方法(3,3,3)

資料型別(dtype)

常見類型：(可固定長度)

- **Float32 浮點數**
- **Uint8 無號整數**
- **Float64 浮點數**

字串長度可能不一樣，故不適合用張量存放

2-2-5 張量的關鍵屬性

✓ 以MNIST資料集具體說明關鍵屬性

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.ndim)
```

```
Out[] : 3          #訓練集的ndim 為 3, 有 3 個軸
```

```
print(train_images.shape)
```

```
Out[] : (60000, 28, 28)          #shape 為 60000x28x28 維的 3D 張量 (有 3 個元素)
```

```
print(train_images.dtype)
```

```
Out[] : uint8          #元素的資料型別為 0~255 的整數
```

2-2-6 在 Numpy 做張量切片

✓ 張量切片tensor slicing：在張量中，選擇特定的元件的動作

```
my_slice = train_images[10:100]
print(my_slice.shape)
```

#選擇第 10 到第 100 個數字的圖像 (不包含第 100 個)，
將它們放入形狀為 (90, 28, 28) 的張量中

Out[] : (90, 28, 28)

```
my_slice = train_images[ 10 : 100 , : , : ]
my_slice.shape
```

#等同於上面的寫法

Out[] : (90, 28, 28)

```
my_slice = train_images[ 10 : 100 , 0 : 28 , 0 : 28 ]
my_slice.shape
```

#同樣等同於上的寫法

Out[] : (90, 28, 28)

2-2-6 在 Numpy 做張量切片

✓ 可以在每個張量軸上的任意兩個索引之間進行切片

```
my_slice = train_images [ : , 14 : , 14 : ]   #切出所有影像右下角的14x14像素，
my_slice.shape                                這裡的 14: 就等於是 14:28
```

Out[] : (60000, 14, 14)

```
my_slice = train_images[ : , 7 : -7 , 7 : -7 ]   #7 : -7 是從頭 7 個元素到 -7 個元素，但不包含 -7，所以
my_slice.shape                                   0 到 6 和 -7 到 1 前後各 7 個元素被去掉了，只剩下中間的
                                                 14 X 14
```

Out[] : (60000, 14, 14)

2-2-7 資料批次 (batch) 的概念

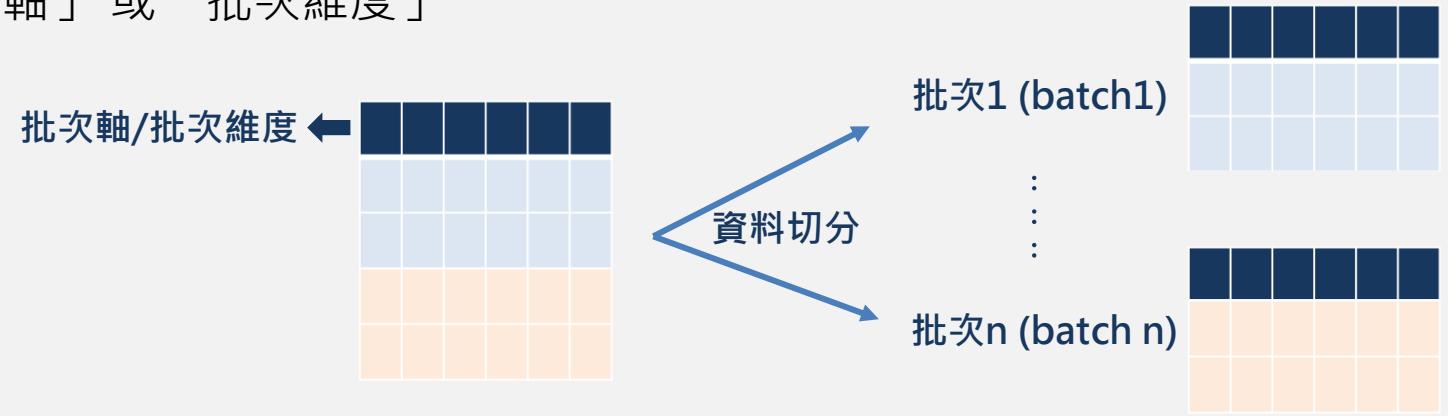
✓ 深度學習模型不會一次學習整個資料集，而是將資料分批執行 (batch)

```
batch = train_images [ : 128 ]    #把 train_images 切片為 128 個圖像為一批 batch
```

```
batch = train_images [ 128 : 256 ]
```

```
batch = train_images [ 128 * n : 128 * ( n + 1 ) ]  # 可以修改程式的 n 值, 找到你想測試的批次量
```

✓ 把整個資料集切成批次的時候，這些批次張量的第0軸(樣本數軸)， 稱為「批次軸」 或「批次維度」



2-2-8 資料張量實例

Dimension(Axis)	0	1	2	3	4
Name	Shape	Vector	Matrix	3D tensor	4D tensor
Another name	0D tensor	1D tensor	2D tensor	3D tensor	4D tensor
Example	12	[12,3,6,14]	([[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]])	[[[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]], [[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]], [[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]]]	...
Shape	() 、empty	(4,)	(3,5) (sample, feature)	(3,3,5) (sample, timesteps, feature)	(5,3,3,4) (sample, channels, height, with) (sample, height, with, channels)
note			向量資料	時間序列資料或序列資料	影像

資料實例

人口精算
純文字文件

股票價格
推特推文

2-2-8 資料張量實例 -0、1階(軸)張量

✓ 0階(軸)張量(樣本軸)

- 所有張量的第 0 軸都是 samples ，也就是每批資料的樣本軸，樣本軸的元素個數就是該批資料的樣本總數

0階張量例子：

公司代號

1階張量例子：

公司代號	公司簡稱	發言日期	發言時間	主旨

2-2-8 資料張量實例 -2階(軸)張量

✓ 2階(軸)張量=向量資料(最常遇到的資料型式)

- **人口精算**資料集 : 紀錄每個人的年齡、郵遞區號、收入每個人可以組成 3 維 (年齡、郵區、收入) 的向量整個資料集共 100000 人

姓名	年齡	郵遞區號	收入
王大明	21	100	3500
陳小華	19	100	6000

shape (10000 , 3) ---- 2D

- **純文字文件**資料集 : 紀錄每個英文單字出現次數，假設有 20000 個單字的辭典，文件裡每個字都可以對應到 20000個單字之一 ，用一個 20000 維的向量來代表辭典，然後每個維度記錄著對應單字出現次數，如何可以將 500 篇文件資料集完整儲存

單字	辭典	出現文章	例句
Apple	牛津詞典	《How to make an apple pie?》	xxx.
Banana	劍橋辭典	《Bananas from Taiwan》	xxxxxxx.

shape (500 , 20000) ---- 2D

2-2-8 資料張量實例 -3階(軸)張量

✓ 3階(軸)張量=時間序列資料或序列資料

- **股票價格**資料集：
  - 每分鐘儲存股票的當前價格、
  - 過去一分鐘的最高價格和最低價格 (共三種價格)
    1) 每一分鐘都被編碼為一個三維向量
    2) 每一個交易日被編碼為 2D (390 , 3) #(美股 6.5小時 = 390 分鐘)
    3) 250天的資料可以被編碼為 3D (250,390,3)
- **推特推文**資料集：
  - 每條推文由 128 個 ASCII 字元組成的 280 個序列字元
  - 每個字元可以被編碼為一個 128 維向量，其中除了與該字元對應的索引位置為 1 ，其餘都是 0
  - 每條推文可以被編碼為 2D (280 , 128)
  - 100 萬條推文則可以被編碼為 3D (1000000 , 280 ,128)

2-3 神經網路的工具: 張量運算

✓ 深度神經網路的所有運算都可以化為張量運算(tensor operations)

- 層(函數) : keras.layers.Dense(512, activation = 'relu')

```
output = relu(dot(W, input) + b)
```

說明 :

- output : 輸出值(2D張量)
- W : 2D張量
- input : 輸入值(2D張量)
- b : 偏執向量

2-3-1 元素間的運算

```
import numpy as np

def naive_relu(x):
    assert len(x.shape) == 2          #若x不是2D張量,將會發生AssertionError

    x = x.copy()    #避免覆寫到輸入張量
    for i in range(x.shape[0]):
        for j in range(x.shape[i]):
            x[i,j] = max(x[i,j], 0) #若元素小於0則為0,大於0則維持原數值
            print(x)
    return x

x = np.array([[-1,-2],[3,-4]])          #example
naive_relu(x)       #呼叫方法
```

1
```
[[ 0 -2]
 [ 3 -4]]
```

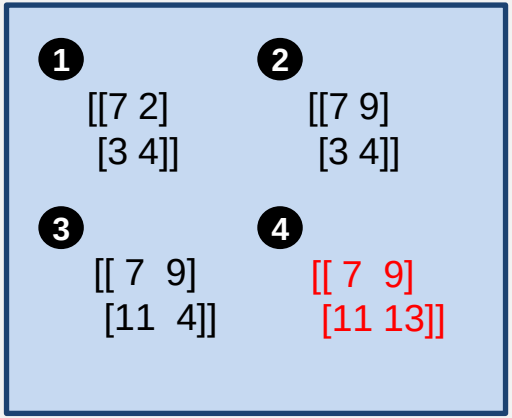
2
```
[[ 0  0]
 [ 3 -4]]
```

3
```
[[ 0  0]
 [ 3 -4]]
```

4
```
[[0 0]
 [3 0]]
```

2-3-1 元素間的運算

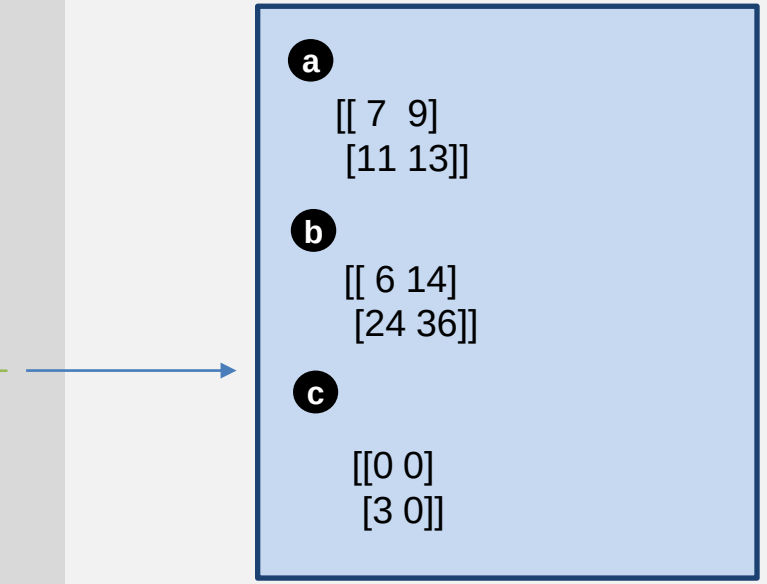
```
import numpy as np

def naive_add(x, y):
    assert len(x.shape) == 2          #若x不是2D張量,將會發生AssertionError
    assert x.shape == y.shape         #x跟y的張量需相同

    x = x.copy()    #避免覆寫到輸入張量
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
        x[i,j] += y[i,j]
        print(x)
    return x

x = np.array([[1,2],[3,4]])    #example
y = np.array([[6,7],[8,9]])    #example
naive_add(x, y)    #呼叫方法
```



2-3-1 元素間的運算

✓ 基本線性代數子程式(Basic Linear Algebra Subprograms, BLAS)

```
import numpy as np

x = np.array([[1,2],[3,4]])    #example
y = np.array([[6,7],[8,9]])    #example
z = np.array([[-1,-2],[3,-4]])          #example

a = x + y      #元素間的相加運算，x和y的元素相加
b = x * y      #元素間的相加運算，x和y的元素相乘
c = np.maximum(z, 0)
```



2-3-2 張量擴張(Broadcasting)

✓ 在Numpy中，不考慮特例情況下，較小的張量將進行**擴張**，以匹配較大的張量

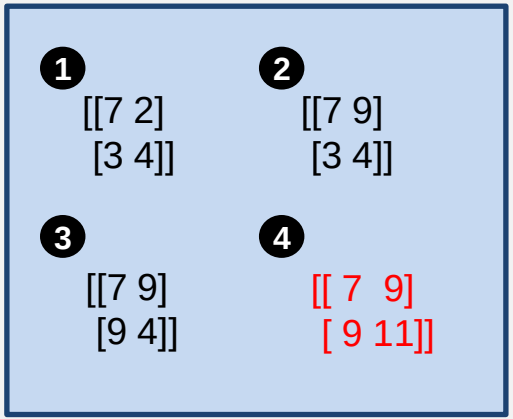
- 較小的張量會加入新的軸(稱為擴張軸)，以匹配較大的張量
- 較小的張量會在這些新的軸上重複寫入元素，以匹配較大的張量的shape

```
x.shape = (32,10)
y.shape = (10,)
```

1. 新增一個空的軸到y張量的第0軸 -> shape = (1,10)
2. 在這新的軸新增32個維度-> shape = (32,10)

2-3-2 張量擴張(Broadcasting)

```
import numpy as np

def naive_add_matrix_and_vector(x, y):
    assert len(x.shape) == 2   #若x不是2D張量,將會發生AssertionError
    assert len(y.shape) == 1   #若y不是1D張量,將會發生AssertionError
    assert x.shape[1] == y.shape[0]

    x = x.copy()       #避免覆寫到輸入張量
    for i in range(x.shape[0]):
        for j in range(x.shape[i]):
            x[i,j] += y[j]
            print(x)
    return x

x = np.array([[1,2],[3,4]])    #example
y = np.array([6,7])        #example
naive_add_matrix_and_vector(x, y)  #呼叫方法
```



2-3-3 張量點積(dot)運算

✓ 點積 (dot)運算，也稱為張量積(tensor product)

※不要與元素間的相乘搞混

```
import numpy as np

z = a * b          #相乘(a x b)
z = np.dot(a, b)   #點積(a・b)
```

2-3-3 張量點積(dot)運算

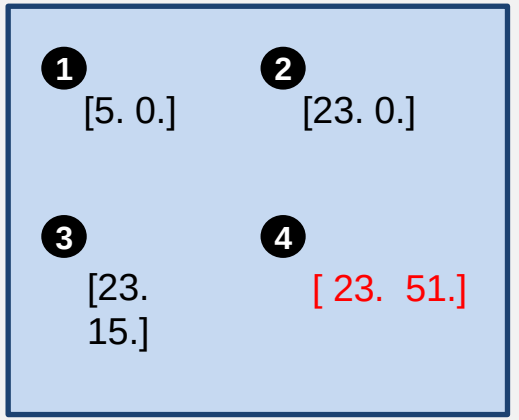
```
import numpy as np

def naive_vector_dot(x, y):
    assert len(x.shape) == 1   #若x不是1D張量,將會發生AssertionError
    assert len(y.shape) == 1   #若y不是1D張量,將會發生AssertionError
    assert x.shape[0] == y.shape[0]

    z=0    #給預設值
    for i in range(x.shape[0]):
        z += x[i] * y[i]              # x[0] * y[0] + x[1] * y[1] ...
        print(z)
    return z

x = np.array([1,2])          #example
y = np.array([5,9])          #example
naive_vector_dot(x, y)   #呼叫方法
```

```
1
5
2
23
```

兩個向量間的點積 = 純量

2-3-3 張量點積(dot)運算

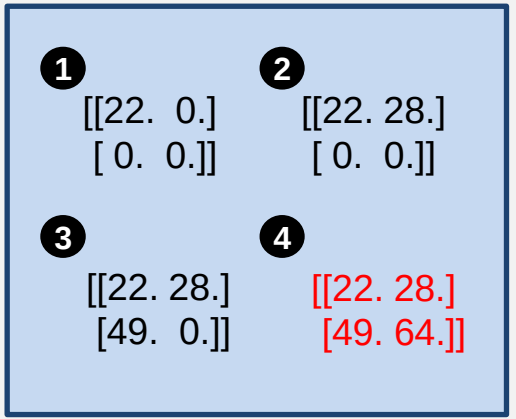
```
import numpy as np

def naive_matrix_vector_dot(x, y):
    assert len(x.shape) == 2    #x是numpy矩陣
    assert len(y.shape) == 1    #x是numpy矩陣向量
    assert x.shape[1] == y.shape[0] #x的第1維必須與y的第0維相等

    z = np.zeros(x.shape[0])   #給預設值
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            z[i] += x[i, j] * y[j]
            print(z)
    return z

x = np.array([[1,2],
              [3,4]])            #example
y = np.array([5,9])              #example
naive_matrix_vector_dot(x, y) #呼叫方法
```



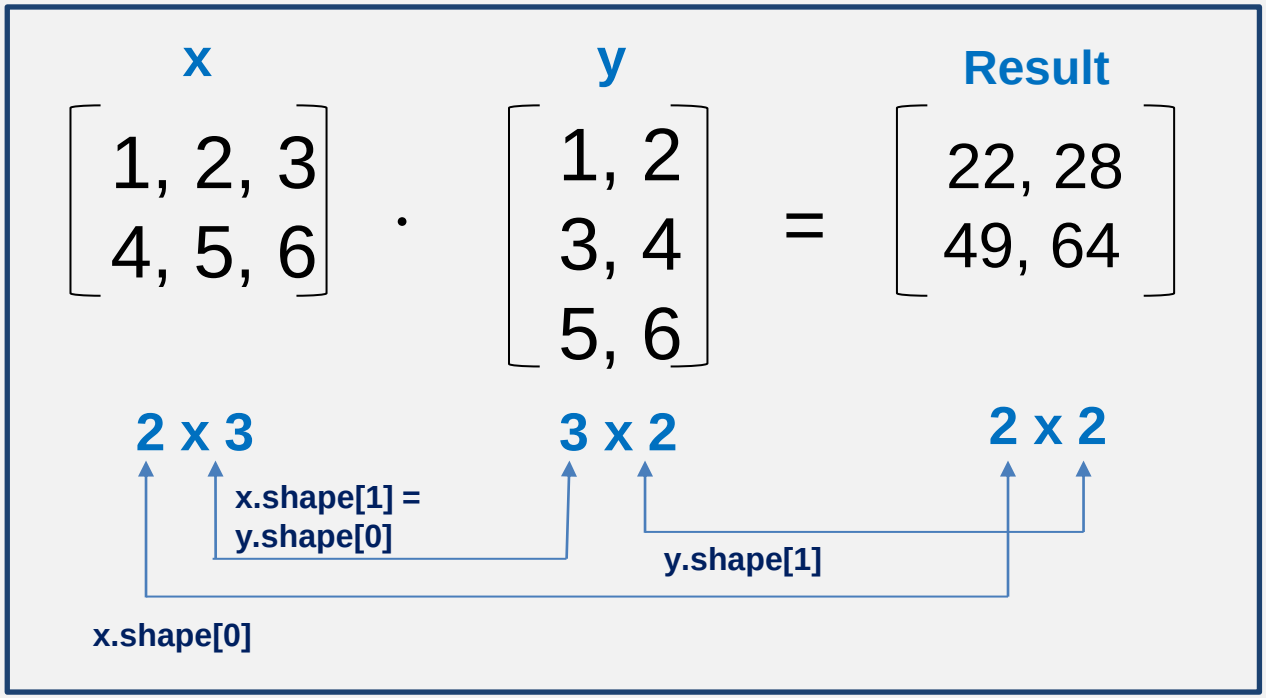
矩陣和向量間的點積 = 向量

2-3-3 張量點積(dot)運算

```
import numpy as np
def naive_matrix_dot(x, y):
    assert len(x.shape) == 2          #x是numpy矩陣
    assert len(y.shape) == 2          #x是numpy矩陣向量
    assert x.shape[1] == y.shape[0]              #x的第1維必須與y的第0維相等

    z = np.zeros((x.shape[0], y.shape[1]))   #給預設值
    for i in range(x.shape[0]):
        for j in range(y.shape[1]):
            row_x = x[i, :]                    #取第1維中的第i個,第2維中的全部
            column_y = y[:, j]     #取第1維中的全部,第2維中的第j個
            z[i,j] = naive_vector_dot(row_x, column_y)  #呼叫naive_vector_dot
            print(z)
    return z
x = np.array([[1,2,3],
              [4,5,6]])
y = np.array([[1,2],
              [3,4],
             [5,6]])
naive_matrix_dot(x, y)
```



2-3-3 張量點積(dot)運算



(a,b,c,d) · (d,) = (a,b,c)

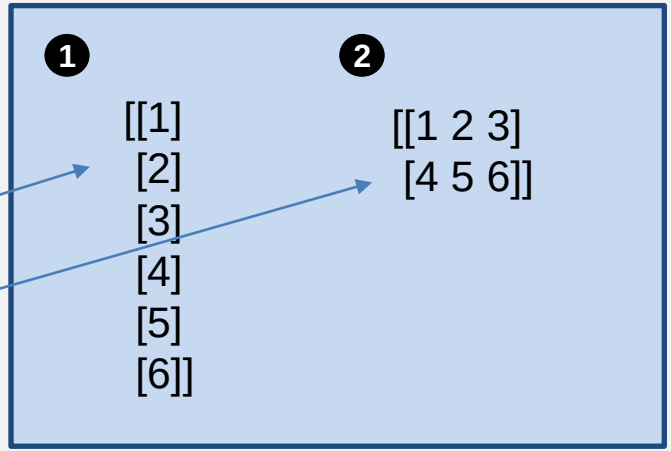
(a,b,c,d) · (d,e) = (a,b,c,e)

.
.
.
.

2-3-4 張量重塑 (reshaping)

✓張量重塑(reshaping)：調整張量各軸的元素數，調整後的張量元素總數不變

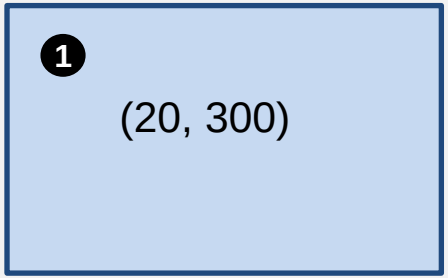
```
import numpy as np

x = np.array([[1,2],
              [3,4],
              [5,6]])

print(x.reshape(6,1))
print(x.reshape(2,3))
```



2-3-4 張量重塑 (reshaping)

- ✓ 矩陣轉置(transposition)時會用到reshaping
  - **矩陣的行轉列(x[i, :] → x[:, i])**

```
import numpy as np

x = np.zeros((300,20))
x = np.transpose(x)
print(x.shape)
```

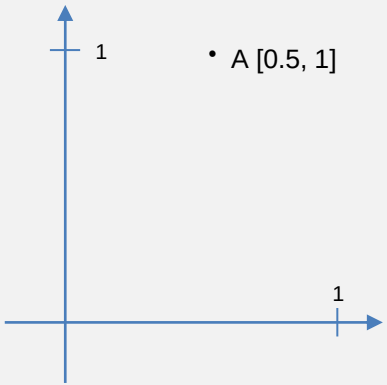


為什麼沒看到reshape?

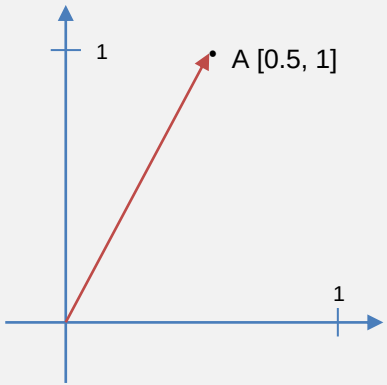
因為transpose暗中呼叫了reshape

2-3-5 張量運算的幾何解釋

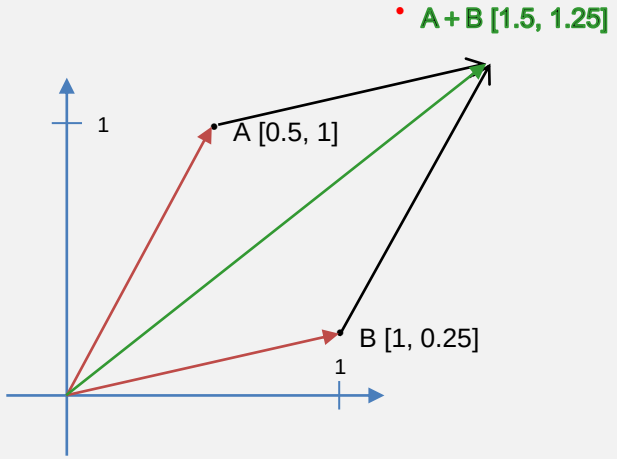
✓ 所有的張量運算都可以透過幾何方式來解釋



點A



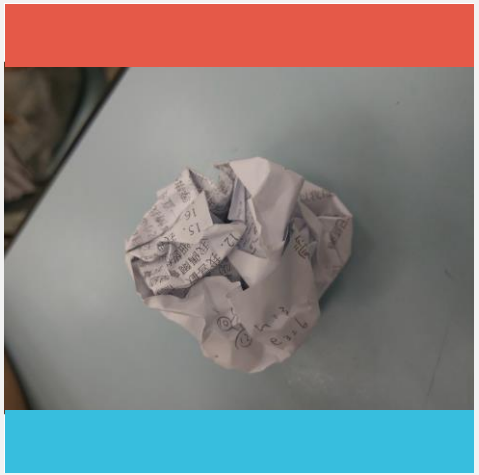
向量A

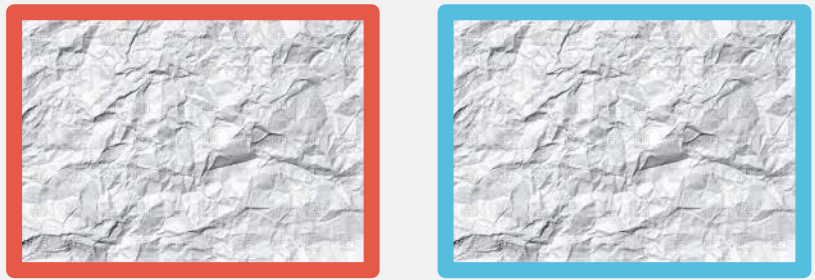


向量A + 向量B

2-3-6 深度學習的幾何解釋

✓運用深度學習將紙團分類

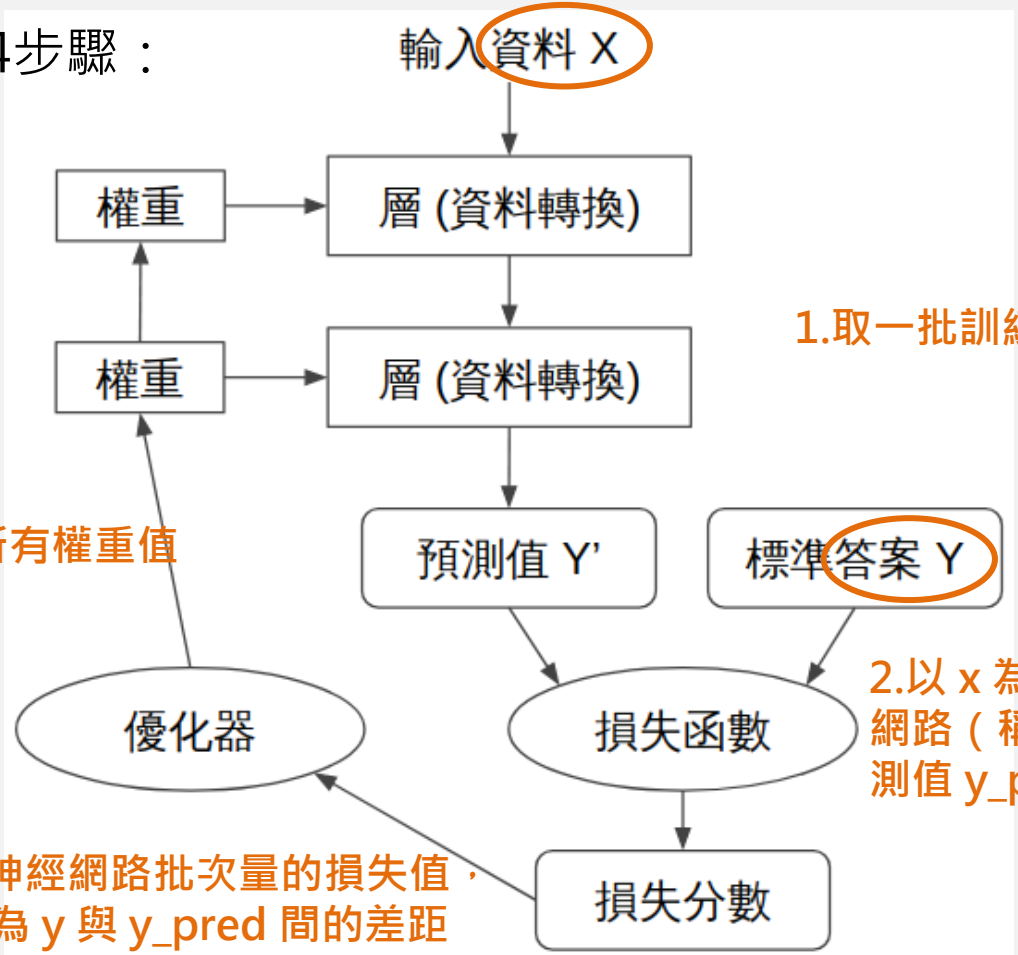




輸入層　　隱藏層　　輸出層

2-4 以梯度為基礎最佳化

✓ 機器學習訓練4步驟：



1.取一批訓練樣本 x 和對應的目標 y

2.以 x 為輸入資料，開始執行神經網路（稱正向傳遞步驟）以獲得預測值 y_pred

3.計算神經網路批次量的損失值，損失值為 y 與 y_pred 間的差距

4.更新神經網路的所有權重值以減少損失值

2-4 以梯度為基礎最佳化

✓ 神經層轉換輸入資料

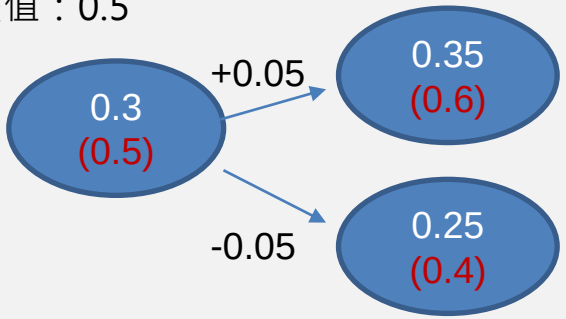
```
output = relu(dot( W , input ) +b )
```

補充 : W , b 是屬性張量，可稱該層**權重**或**可訓練參數**，透過批次調整權重可減少損失值

問題：需計算正反兩向

方法：NN運算具**可微分**的特性，計算**損失值梯度**後，可從梯度的相反方向來更動係數，減少損失值

權重係數初始值：0.3
正向傳遞後損失值：0.5



2-4-1 何謂導函數（微分derivative）

✓ 平滑連續函數

$f(x) = y$

#在x加入很小的數值epsilon_x後，導致y改變的值epsilon_y

$f(x + epsilon\_x) = y + epsilon\_y$

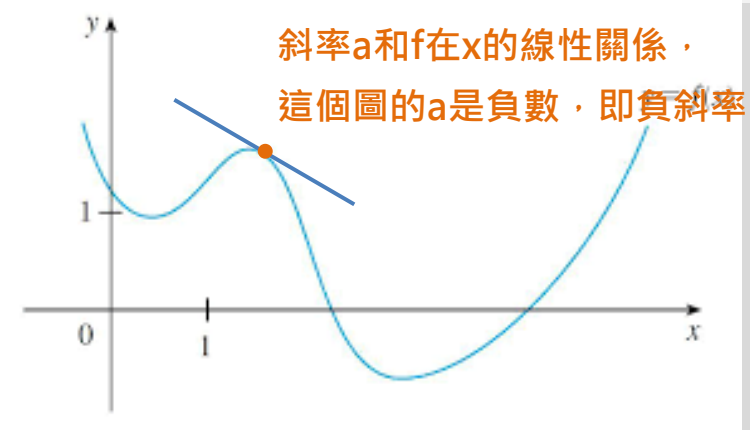
$\#epsilon\_y = a * epsilon\_x$

$f(x + epsilon\_x) = y + a * epsilon\_x$

$f(x + epsilon\_x) - y = a * epsilon\_x$

#因為夠小，$epsilon\_x$,$epsilon\_y$ 兩者成線性關係

$f(x + epsilon\_x) - f(x) = a * epsilon\_x$



斜率a和f在x的線性關係，

這個圖的a是負數，即負斜率

補充：

- 斜率a稱f在x點上的導函數，表示方式：$f'(x) = \frac{dy}{dx}$
- a是正數，則x稍微增大會導致f(x)變大；a是負數，則x稍微增大會導致f(x)變小

  ex: 要使f(x)變小→往斜率反方向移動；斜率為正就$(-epsilon_x)$，斜率為負就$(+epsilon_x)$

2-4-2 梯度（張量運算的導數）

- ✓ 函數 f(x)在x點的斜率就是函數的導函數f′(x)
- ✓ 函數 f(W)在W點的梯度就是函數的張量導函數f′(W)

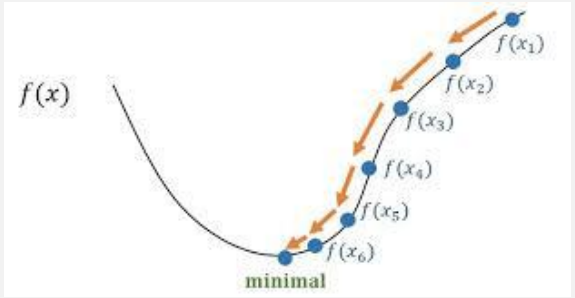
```
y_pred = dot( W , x)
loss_value = loss( y_pred , y) = loss( dot( W , x ), y)

loss_value=f(W) #將x , y固定不變
```

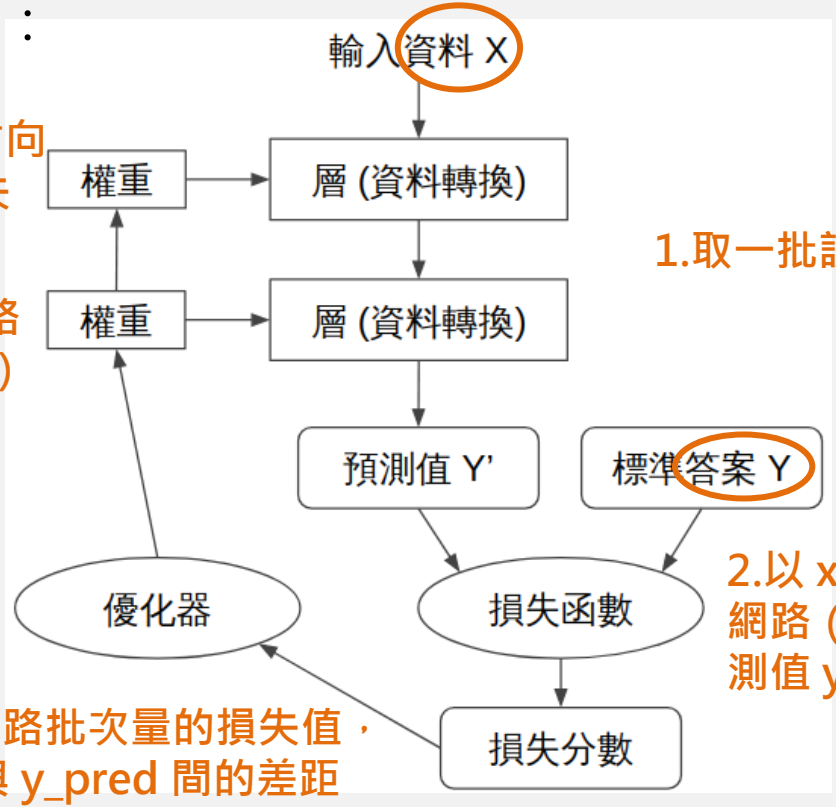
```
W1=W0-step*gradient(f)(W0)
```

補充：
- ✓ gradient f(W0) 可解釋為 f(W)在W0這張量點附近的曲率張量
- ✓ 要降低f(W)的值→往梯度反方向移動

2-4-3 隨機梯度下降 (SGD)

- ✓ 隨機梯度下降法(Stochastic gradient descent, SGD)
- ✓ 隨機(stochastic)：每批資料都是隨機抽取
- ✓ 機器學習訓練5步驟：





1.取一批訓練樣本 x 和對應的目標 y

2.以 x 為輸入資料，開始執行神經網路（稱正向傳遞步驟）以獲得預測值 y_pred

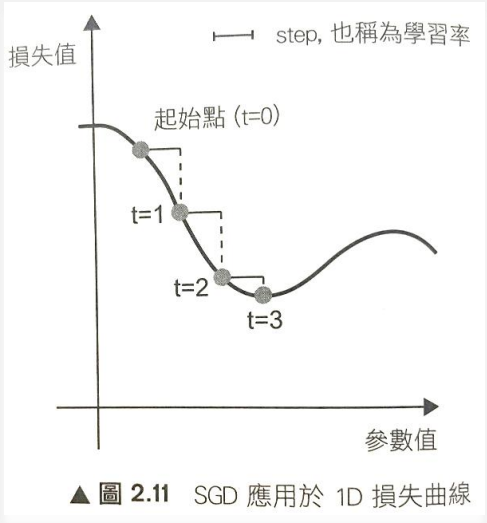
3.計算神經網路批次量的損失值，損失值為 y 與 y_pred 間的差距

4.計算損失值對神經網路權重的梯度（反向傳播）

5.將參數稍微向梯度的反方向移動，從而降低批次的損失

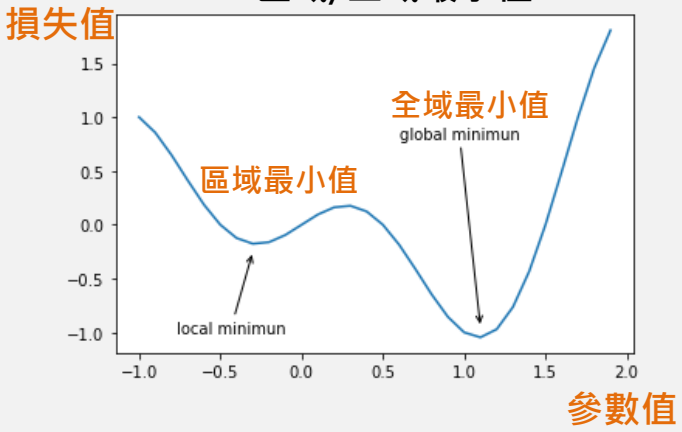
2-4-3 隨機梯度下降 (SGD)

- ✓ 小批次隨機梯度下降 (mini-batch stochastic gradient descent, SGD)
- ✓ 學習率(step)太小，曲線下降需多次迭代，且可能陷入局部最小值
- ✓ 學習率(step)過大，參數的更新可能略過真正的最小值

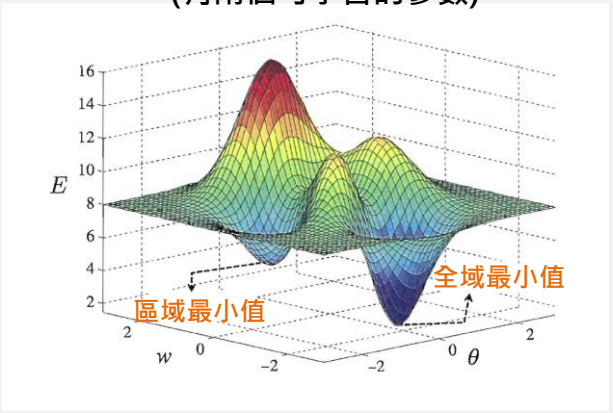


▲ 圖 2.11 SGD 應用於 1D 損失曲線

區域/全域最小值



曲面的損失值梯度下降
(有兩個可學習的參數)



SGD補充：

- 小批次：每次迭代時，只取單一筆樣本和目標，而非一批資料
- 整批：一次把所有可用資料用上，資料更新更準確，但增加執行複雜度

2-4-3 隨機梯度下降(SGD)

✓ 其他SGD(最佳化方法)：動量(momentum)SGD、Adagrad、RMSProp

- 動量：解決SGD收斂速度、區域最小值的問題

```
past_velocity=0
momentum=0.1     #固定動量因數
while loss > 0.01 :    #最佳化迴圈
        w, loss, gradient = get_current_parameters( )
        velocity = past_velocity * momentum – learning_rate * gradient
        w= w + momentum * velocity – learning_rate * gradient
        past_velocity = velocity
        update_parameter(w)
```

補充：
動量會和**當前的斜率**（加速度）有關，同時還要考量**當前的速度**（受過去加速度影響）
更新參數W須根據**當前的梯度值**外，還須根據**以前的參數**（受過去梯度值影響）來更新參數W

2-4-4 連串的導數：反向傳播演算法

✓ 實務上，一個神經網路函數由許多張量運算串連而成
- 如: 神經網路 f 包含三個張量運算 a, b, c(權重矩陣分別W1,W2,W3)

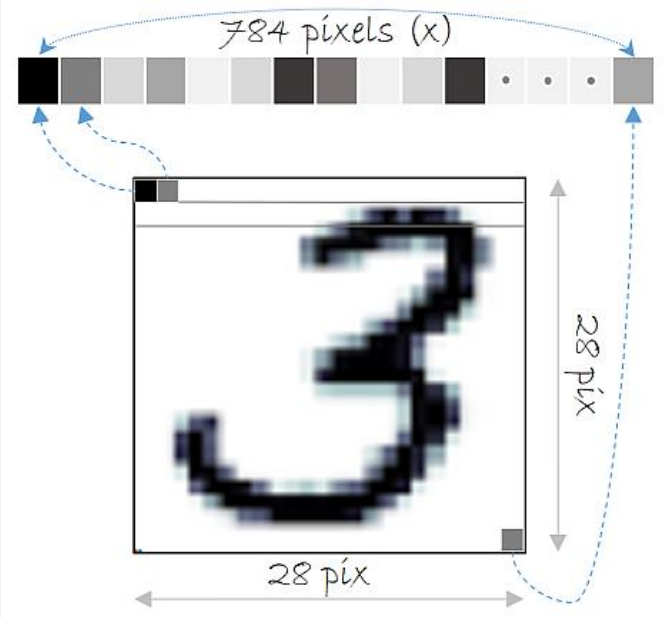
```
# 損失函數 f 表示
f(W1, W2, W3) = a(W1, b(W2, c(W3) ) )
```

✓ **連鎖法則**：$f'\big(g(x)\big) = f'(g) * g'(x)$

✓ **反向傳播演算法**：從最終損失值開始，由後面層向前面層反向運作，套用連鎖法則對神經網路中所有權重計算損失函數的梯度，這個梯度可用來更新權重值以最小化損失函數
- TensorFlow : 具符號微分運算能力的軟體框架來建構神經網路

2-5 回顧範例 -MNIST 資料集

- ✓ MNIST 是一個手寫數字的圖像資料集，我們將用於圖像辨識的範例之中
- ✓ 資料集包含 60000 個訓練圖片和 10000 個測試圖片
- ✓ 每個圖片大小是 28 * 28 像素



2-5 回顧範例 – 輸入資料集

✓ 輸入資料

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

說明 :
灰階影像值分成0~255階，用0~255整數表示。astype為轉換型別，除以255是將灰階值正規化成0~1間的浮點數

2-5 回顧範例 – 建構模型

✓ 神經網路建構

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

補充：
relu：f(x)=max(0, x) ，全名為線性整流函數 (Rectified Linear Unit)，解決深層神經網路梯度消失的問題，正數梯度保持為 1

Softmax：歸一化指數函式，它能將一個含任意實數的K維向量 「壓縮」到另一個K維實向量中，使得每一個元素的範圍都在之間，並且所有元素的和為1

2-5 回顧範例 – 損失函數

✓ 神經網路編譯

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

補充：

- optimizer(優化器)：神經網路根據其輸入資料及損失函數值而自行更新的機制
- categorical_crossentropy：是個損失函數，使用損失值作為回饋訊號來調整權重張量，在訓練過程中嘗試達到損失值最小化
- Metrics(評量準測)：此範例關注點為數字的辨識度

2-5 回顧範例 – 訓練循環

✓訓練循環：跑完所有訓練資料一次稱為一個epoch

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Out[] : Epoch 1/5
60000/60000 [==========] - 5s 80us/step - loss: 0.2563 - accuracy: 0.9259
Epoch 2/5
60000/60000 [==========] - 4s 74us/step - loss: 0.1023 - accuracy: 0.9699
Epoch 3/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0674 - accuracy: 0.9798
Epoch 4/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0486 - accuracy: 0.9854
Epoch 5/5
60000/60000 [==========] - 4s 74us/step - loss: 0.0372 - accuracy: 0.9892
```

共進行2345次梯度更新：60000/128=468.75(每個epoch), 469*5=2345次

總結

✓ 目前為止，你應該知道的內容：

- 用6行程式寫一隻深度學習神經網路
- 張量、張量的運算
- 神經網路如何透過**反向傳播**和**梯度下降**來進行學習?