深度學習 - 第二章

Status	Book
Created time	@November 28, 2024 8:22 AM

第二章

Google Colab 檔案

Ch2.ipynb

2-1 初探神經網路：第一隻神經網路（辨識手寫數字）

- 解釋神經網路，最簡單的方式，就是使用 Python 程式庫 Keras 來辨識並分類手寫的數字。
- MNIST 資料集
  - 含有 60,000 張訓練圖片外加 10,000 張測試圖片的資料集。
  - 可以拿來驗證你的演算法是否按預期的邏輯在運作。
  - 包含在 Keras 套件裡，以四個 Numpy 陣列的形式，預先載入在 Keras 當中。
- 關於類別（class）、樣本（samples）與標籤（label）
  - 分類問題（classification）：讓機器把輸入資料加以分類到不同的類別。
  - 樣本：學習階段，輸入的資料。
  - 標籤：人工標註，跟隨著每個樣本。
  - 機器要學習把樣本資料歸類到和標籤所標示一樣的類別上。

    - 結果與標籤一致性很高 → 學習成功

程式2.1：MNIST 資料集

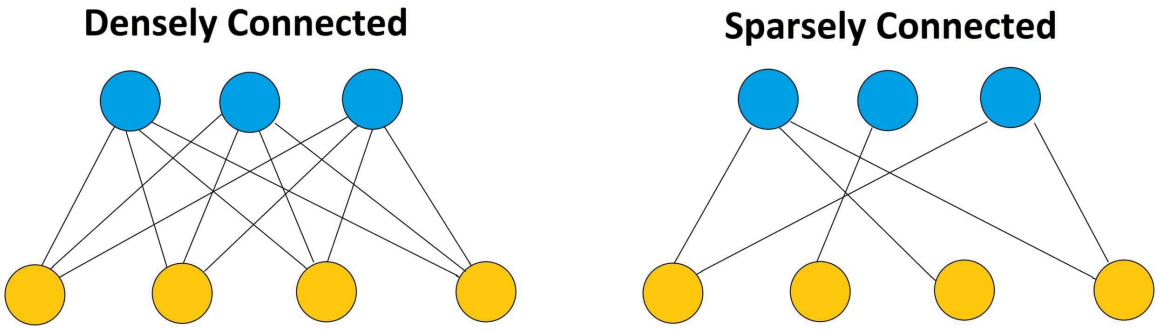
```
from keras.datasets import mnist
from keras.src.backend import shape

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# (train_images, train_labels) 訓練集（training set）
# (test_images, test_labels) 測試集（testing set）
# 此處圖片都被編碼成 Numpy 陣列, 0 ~ 9 的數字陣列, 每張圖片對應一個標籤

print(train_images.shape) # 3軸, 60000 維 x28 維 x28 維
print(len(train_labels)) # 標籤 60000 個
print(train_labels) # 標籤是 0 ~ 9 之間的數, 資料型態 unit8

print(test_images.shape) # 3軸, 10000 維 x28 維 x28 維
print(len(test_labels)) # 標籤 10000 個
print(test_labels) # 標籤是 0 ~ 9 之間的數, 資料型態 unit8

# 操作流程：
# 提供訓練集 train_images 和 train_labels 給神經網路
# 神經網路學習歸類, 並與標籤對比, 歸類錯就加以修正
# 對 test_images 中的圖片進行預測, 並驗證是否與 test_labels 中記錄的標籤符合
```

程式2.2：神經網路架構

- 神經網路的基本元件就是層（layer）
    - 一個層就是一個資料處理的模組，資料過濾器。
    - 資料進去，輸出比較有用的結果，萃取特定的轉換或表示法（representation）。
    - 深度學習將許多層連接，每一層漸次執行資料萃取（data distillation）。
    - 深度學習模型就像資料過濾器，由一連串越來越精細的資料過濾層所組成。

```
from keras import models
from keras import layers

network = models.Sequential()

# 兩個密集層（Dense layer）, 密集層也稱為全連接（fully connected）
# 第一密集層, 層寬度由 Dense() 第 0 參數指定
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
# 第二密集層（也是最後一層）, 有10個輸出的 softmax 層
# 輸出一個含有 10 個機率評分（probability scores）的陣列（機率總和為 1）
network.add(layers.Dense(10, activation='softmax'))
```

- 全連接層 fully connected
  - 前後層中神經元全部都彼此連接。
  - 不是密集連接，稱為稀疏層。
  - Wij 是訊號傳遞的權重參數，代表第 i 個神經元傳遞到第 j 個神經元。



- 讓網路準備接受訓練，需要三項元件才能進行編譯（compilation）
  - 損失函數（loss function）：衡量神經網路在訓練資料上的表現，引導正確的修正方向。
  - 優化器（optimizer）：根據輸入資料及損失函數值而自行更新的機制。
  - 訓練和測試的評量準則（metrics）：在本範例中為辨識數字的正確性（accuracy）。

```
from keras import models
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'))

network.compile(optimizer='rmsprop', # 指定優化器
                loss='categorical_crossentropy', # 指定損失函數
                metrics=['accuracy']) #指定評量準則
```

- 資料預先處理
  - 讓所有數值都能介於 [0, 1] 的區間。
  - 訓練圖片 unit8 型別、數值介於 [0, 255] 儲存於（60000, 28, 28）的陣列中。
  - 轉換用 reshape 和 astype
    - 轉換後為float32型別、數值介於 [0, 1] （因為除以255）的（60000, 28 * 28）陣列。

```
from keras import models
from keras import layers
from keras.utils import  to_categorical

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

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

# 資料預先處理
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
```

- 標籤需要進行分類編碼。

```
from keras import models
from keras import layers
from keras.utils import  to_categorical
from MNIST import train_images, test_images, train_labels, test_labels

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

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

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

# 對標籤進行分類編碼
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# 訓練網路
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

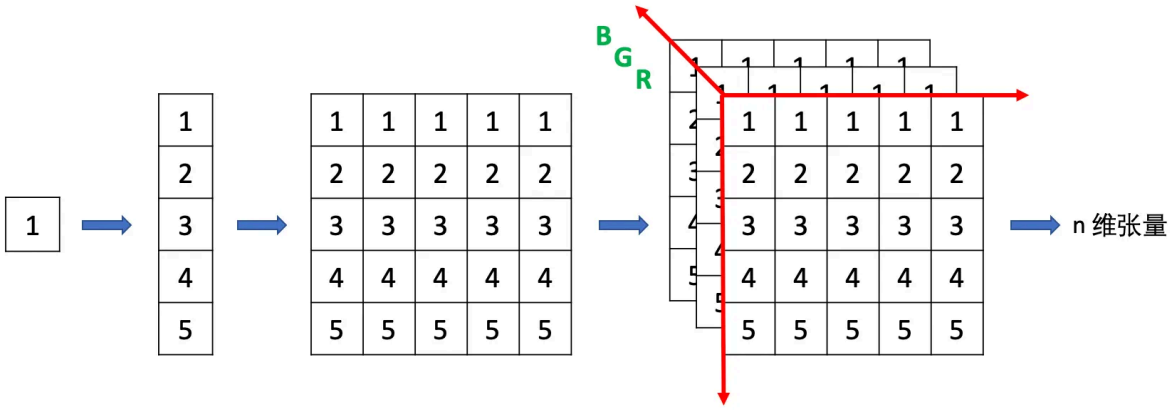
- 訓練期間會顯示損失值（loss, 即損失函數值），以及網路對目前訓練資料的正確率（acc）。
- 檢查模型表現

```
from keras import models
from keras import layers
from keras.utils import  to_categorical
from MNIST import train_images, test_images, train_labels, test_labels

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

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

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

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

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

# 檢查模型在測試資料集上的表現
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

- 過度**配適**（overfitting）：模型學到了訓練數據中的噪音和細節，失去了對未見數據的泛化能力。
- 訓練集的正確率和測試集的正確率之間的差距，是過度**配適**（overfitting）的結果。

2-2 神經網路的資料表示法：張量 Tensor

- 多維 Numpy 陣列，也稱為張量（tensor），目前機器學習系統都使用張量作為基礎的資料結構，比起一般的list，Numpy 元素類型必須相同。
- 張量（tensor）：和 list、tuple 一樣都是資料容器，不過它儲存的幾乎都是數值資料，矩陣便是一種 2D 張量。
- 張量 tensor 的維、階、軸
  - 是一種多階（rank）或稱多軸（axis）的數學結構。
    - 數值純量（scalar）→ 0 階的張量。
    - 向量（vector）→ 1 階張量。
    - 矩陣（matrix）→ 2階張量。
  - 張量每一階所含的元素個數，稱為該階的維度（dimention）。
    - 每一列列和行都有兩個元素，該張量是一個 2x2 維的 2 階張量。
    - 用 Numpy 的 shape 屬性所顯示出來的是張量各階的維度。



- 向量 vs. 張量
  - nD 向量與 nD 張量是完全不一樣的。
  - nD 向量 → n 維向量。
  - nD 張量 → n 階（軸）張量。

2-2-1 純量（0D 張量）

- 包含一個數值的張量，純張量、0 階張量、0 軸張量、0D 張量。
- Numpy 中，float32、float64 型別就是純量張量。

2-2-2 向量（1D 張量）

- 由一組數值排列而成的陣列。
- 一個向量有 5 個元素，稱之 5 維向量。
- 5D 向量 → 1 個軸，軸上有 5 個維度。
- 5D 張量 → 5 個軸，軸上有的維度量由 array() 建立、決定，技術上更準確的名稱為 5 階張量。
- 維度（dimensionality）：軸上元素數量。

2-2-3 矩陣（2D 張量）

- 由一組向量組成的陣列就是一個矩陣。
- 矩陣有兩個軸，列（rows）、行（columns）。

2-2-4 3D 張量和高階張量

- 多個矩陣包裝在一個新的陣列中。
- 將多個 3D 張量放到一個陣列裡，可組成一個 4D 張量，依此類推可持續向上發展。
- 深度學習通常處理 0D 到 4D 張量，處理視訊資料則可能提高到 5D 張量。

2-2-5 張量的關鍵屬性

- 軸的數量（階數）
  - Numpy 中，軸的數量
  - N為張量的 ndim。
- 形狀（shape）
  - 描述張量上的每個軸有多少個維度。
- 資料型別（Python 程式庫中通常稱為 dtype）
  - 常見型別為 float32、uint8、float64，極少數為字元（char）。
  - Numpy 中不存在字串張量，張量會存放固定長度的資料，字串長度不一，不適合存放。

```
import matplotlib.pyplot as plt
from MNIST import train_images

print(train_images.ndim) # 3 個軸
print(train_images.shape) # 60000×28×28 維 3D 張量
print(train_images.dtype) # uint8, 資量型別 0 - 255 整數

# train_images
# 由 8 位元（bit）整數所組成的 3D 張量
# 由 60,000 個 28 X 28 的矩陣組成
# 每個矩陣是一個灰階圖像, 像素質 0 - 255

# 顯示這個 3D 張量中的一個矩陣（每個矩陣都是一個手寫數字）
digit = train_images[4]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```

2-2-6 在 Numpy 做張量切片 Tensor Slicing

```
from MNIST import train_images

# 選擇第 10 到第 100 個數字的圖像（不包括第 100 個）
# 放入形狀為（90, 28, 28）的張量中
my_slice = train_images[10:100]
# my_slice = train_images[10:100, :, :]
# my_slice = train_images[10:100, 0:28, 0:28]
print(my_slice.shape)

# 張量軸上任意兩個索引間切片
# 切出影像右下角的 14 X 14 像素
my_slice = train_images[:, 14:, 14:]
print(my_slice.shape)

# 切出影像居中的 14 X 14 像素
```

```
my_slice = train_images[:, 7:-7, 7:-7]
print(my_slice.shape)
```

2-2-7 資料批次（batch）的概念

```
from MNIST import train_images

# 把 train_images 切片為 128 個圖像為一批 batch
batch = train_images[:128]
print(batch.shape)

# 下一批 batch
batch = train_images[128:256]
print(batch.shape)

# 第 n 批
# batch = train_images[128*n : 128*(n+1)]
```

- 資料張量第 0 軸就是樣本數軸，MNIST中，樣本數軸上的每個元素就是一張數字圖像。
- 資料集切成批次的時候，批次張量的第 0 軸稱為批次軸或批次維度，深度學習經常遇到的術語。

2-2-8 資料張量實例

- 向量資料 → 2D 張量，shape 為（samples, feature）。
- 時間序列資料或序列資料 → 3D 張量，shape 為（samples, time steps, feature）。
- 影像 → 4D 張量，shape 為（samples, height, width, channels）或（samples, channels, height, width）。
- 視訊 → 5D 張量，shape 為（samples, frames, height, width, channels）或（samples, frames, channels, height, width）。
- 以上所有張量的第 0 軸都是 samples，樣本軸的元素個數就是該批次資料的樣本總數，切成批次，就是批次量的個數。
- 因為多了樣本軸，所以每個實例的張量都提升。

2-3 神經網路工具：張量運算

- 深度神經網路所有運算都可以化為張量運算（tensor operations）
- 一個 Keras 的層可以看成一個函數，該函數將輸入矩陣加以運算，然後傳回一個新的陣列。
- output = relu(dot(W, input) + b)
  - input 與 W 張量的點積運算後得到的 2D 張量再與向量 b 相加。
  - relu(x) 是正向的線性輸出函數，代表 max(x, 0)，當 x<0 時，max(x, 0) 就是 0。

2-3-1 逐元素的計算

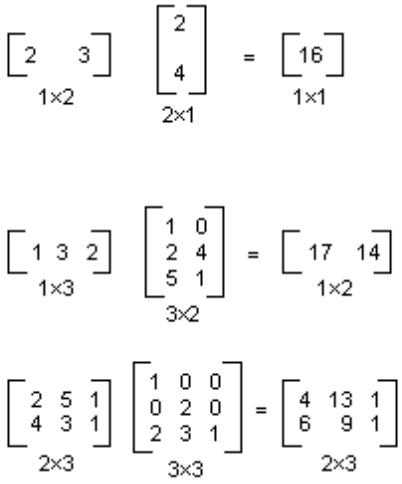
- 逐元素（element-wise）運算：對張量中的每個數值進行各自獨立的運算。
- 處理 Numpy 陣列時可以直接用經過最佳化的函式代替
  - 委托給基本線性代數子程式（Basic Linear Algebra Subprograms, BLAS）來執行。
    - BLAS 屬於最底層、平行化的高效率張量運算，通常是 Fortran 或 C 撰寫的。
- 在 GPU 上運行 Tensorflow 會透過全面向量化的 CUDA 來執行逐元素運算。

2-3-2 張量擴張（Broadcasting）

- 若張量不同，張量小的進行張量擴張
  1. 較小的張量會加入新的軸（稱為擴張軸），以匹配較大的張量。
  2. 較小的張量在這些新的軸上重複寫入元素，以匹配較大的張量形狀。

2-3-3 張量點積運算

- 點積（dot）運算，也稱張量積（tensor product），別混淆「 * 」符號。
- 張量積運算不是對稱的。
- 就是矩陣乘法

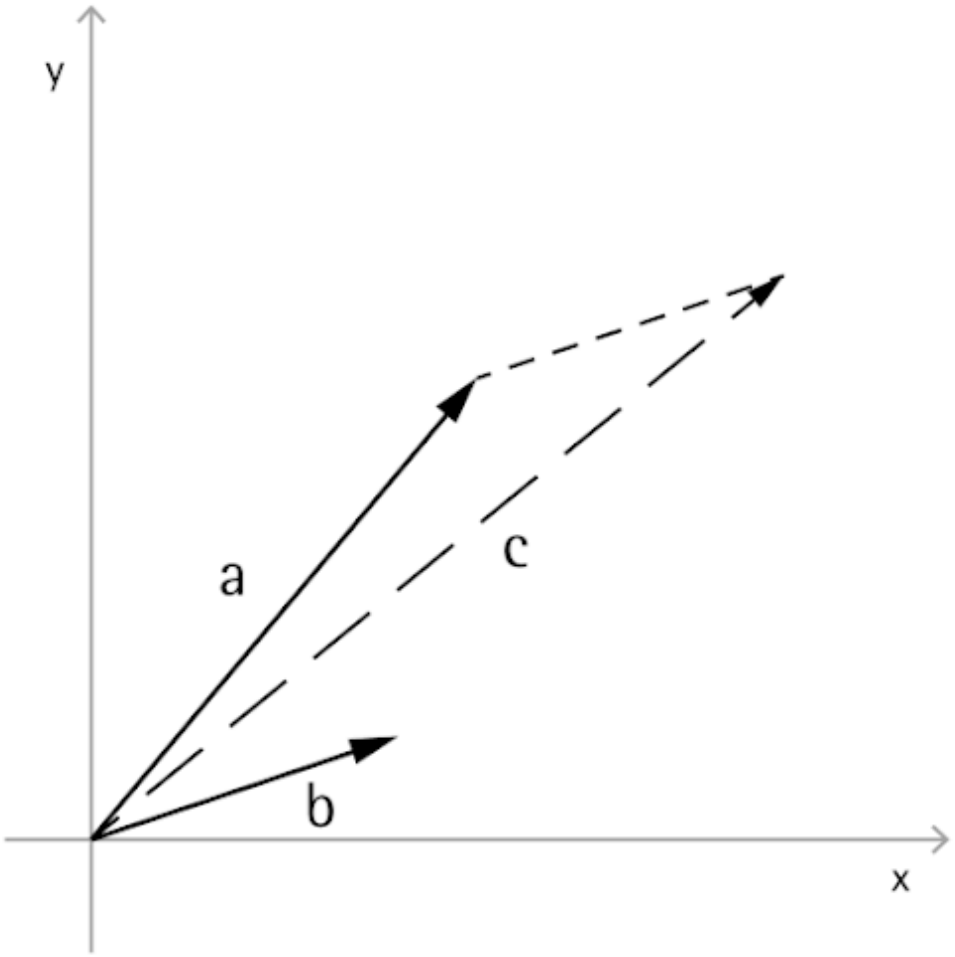


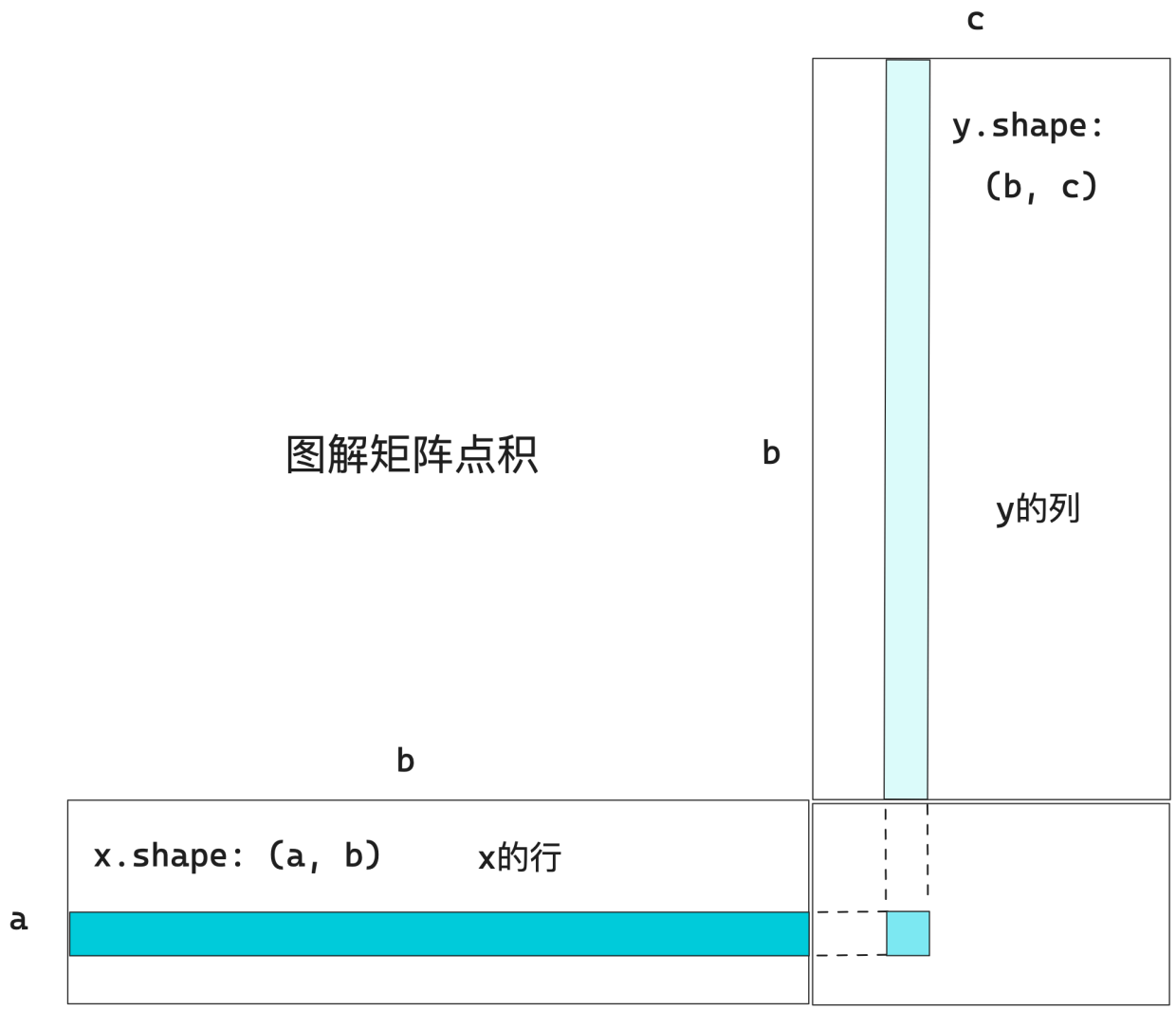
2-3-4 張量重塑

- 張量重塑（reshaping）：調整張量各軸內的元素。
- 做矩陣轉置（transposition）就會用到重塑
    - 矩陣列和行交換，x[i, :] 變成 x[:, i]。

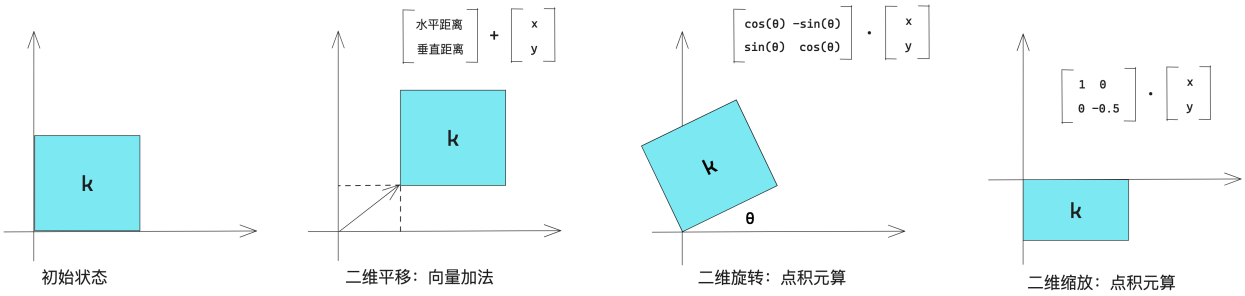
2-3-5 張量運算的幾何解釋

- 所有張量運算都可以透過幾何來解釋。


y
a
c
b
x

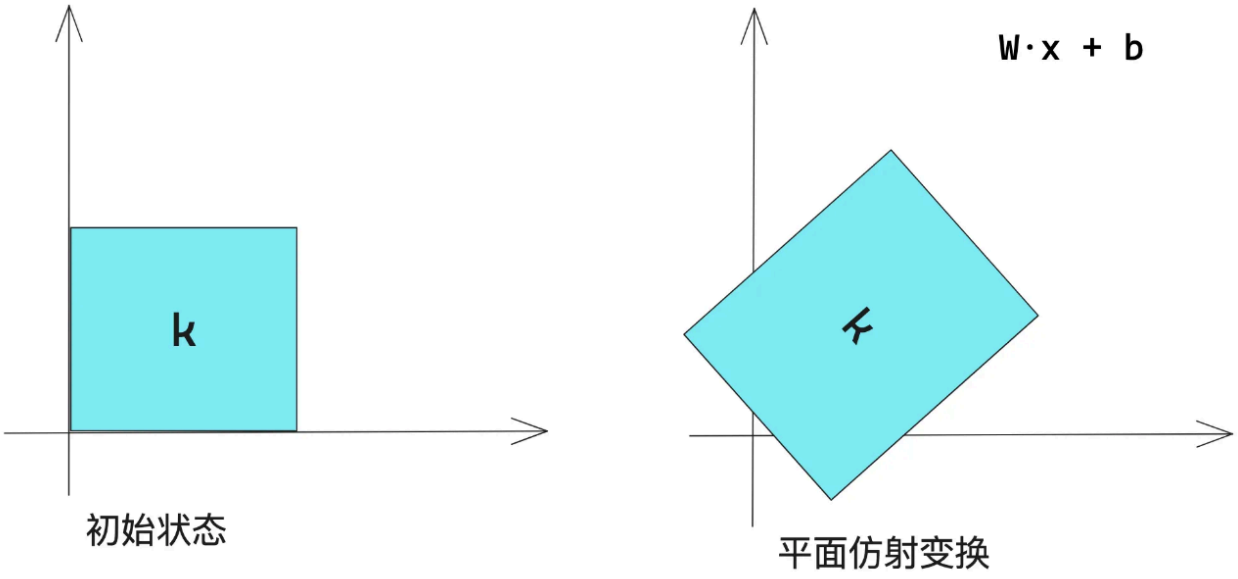


- 張量加法的意義就是朝特定方向以特定距離平移（translating）物體。



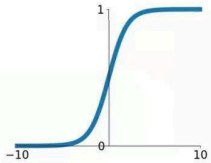
- 平移、旋轉（rotation）、縮放（scaling）都可以用張量運算來表示
  - 平移

        - 對一個點加上向量 → 朝特定方向移動特定距離。
        - 對一群點（某平面物件的各點）加上相同向量 → 平移。
    - 旋轉
        - 將平面物件的各點與 2x2 矩陣 R 進行點積運算。
        - R = [[cos(theta), -sin(theta), sin(theta), cos(theta)]]，theta 為選轉的角度大小。
    - 縮放
        - 將平面物件的各點與 2x2 矩陣 S 進行點積運算。
        - S = [[垂直比例, 0], [0, 水平比例]]，為對角矩陣，只有在左上到右下，才有非 0 元素。
    - 線性變換（Linear transform）
        - 與任意陣列的點積操作可以實現線性變換，縮放與旋轉定義上也屬於線性變換的一種。
    - 仿射變換（Affine transform）
        - 線性變換 + 平移，無激活函數，一個沒有激活函數的密集層就是仿射層。
    - 搭配 relu 激活函數的密集層
        - 重複進行多次仿射變換，得到的結果依舊可以用單次的仿射變換來重現。
        - affine2(affine1(x)) = W2 · (W1 · x + b1) + b2 = (W2 · W1) · x + (W2 · b1 + b2)
            - W2 · W1 · x 對輸入 x 的線代變換。
            - 與 W2 · b1 + b2 進行加法操作（平移運算）。
        - 建構多個密集層卻沒有搭配任何激活函數，其效果等同於單一的密集層。
        - 有激活函數，多個密集層組合可以用來實現非常複雜、非常線性的幾何轉換，讓深度神經網路建立非常廣大的假設空間。



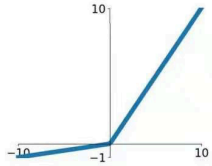
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



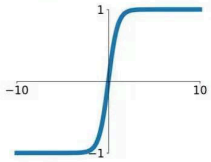
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

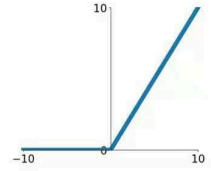


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

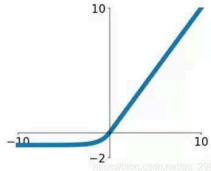
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



2-3-6 深度學習的幾何解釋

- 想像紅色紙與藍色紙重疊並揉在一起變為一顆紙球，皺巴巴的紙球就是輸入資料，而每張色紙是分類問題中的一類資料。
- 神經網路就是要弄清楚紙球的變換過程，讓紙球盡可能恢復平整，使這兩個類別（紅色紙和藍色紙）能再次清楚地被分開。
- 為複雜的、高度繁複的資料找到簡潔的轉換表示法。
- 將資料一小部分一小部分拆解，層層堆疊的結果，使極為複雜的資料在拆解過程中，變得更好處理。

2-4 神經網路的引擎：以梯度為基礎的最佳化

- output = relu(dot(W, input) + b)
  - W 和 b 是該層的屬性張量，被統稱為層的權重（weights）或可訓練參數（trainable parameters），分別為內核（kernel）屬性和偏值（bias）屬性。
- 權重張量會隨機初始化
  - 一開始的轉換表示法是毫無意義的，根據後來回饋訊號逐漸調整權重。
  - 漸進的調整，也稱為訓練（training），就是機器學習中所謂的學習。
- 學習方式可以表示成訓練迴圈（training loop）
  1. 取出一批次的訓練樣本 x 和對應的目標 y_true（標籤）。（基本輸入動作）
  2. 以 x 為輸入資料，開始執行神經網路（正向傳播）以獲得預測值 y_pred。（張量運算）
  3. 計算神經網路的批次損失值，損失值就是 y_true 與 y_pred 間的差距。（張量運算）
  4. 更新神經網路的所有權重值，以減少損失值。（梯度下降法）
  - 多次循環後，損失值非常低，這時神經網路已經學會將輸入對應到正確目標。
- 梯度下降法（gradient descent）
  - 假設有個函數 z = x + y，y 值的微小變動會造成 z 值的微小變動。
  - 知道 y 的變動方向，就可以推論出 z 的變動方向，從數學觀點來看，嚴格來說這個函數在這一個點上可微分（differentiable）。
  - 將許多這樣的函數連接起來，最終得到的複雜函數仍然會保有可微分性質。這一個特性可套用至將「模型參數」映射到「模型在一批次資料上的損失值」的函數上，對模型參數做細微的變動，就會造成損失值上細微且可預測的變動。
  - 用梯度來描述：當參數模型往不同方向變動時，損失值會如何變化。
  - 計算出梯度，便可調整模型參數。

2-4-1 何謂導函數（或稱微分derivative）→ 就是講基礎微分，沒啥w

- 導函數能找出最小化 f(x) 函數值的 x。

- 要使 f(x) 變小，只要知道 f(x) 的斜率（導函數），然後將 x 往斜率的反方向移動一點點就可以了。

2-4-2 張量運算的導數：梯度 → 就是講在張量上的微分、偏微分

- 梯度用來泛指輸入為張量的函數之導函數概念。
- 在張量上求梯度，就是在函數所描繪的多維曲線上求曲率（curvature）。
- grad(loss_value, W0)，在 W0 附近，loss_value=f(W) 梯度最陡方向（還有斜率大小）之張量。
- 函數 f(W)，經由移動張量 W 往梯度反方向移動來降低 f(W) 的值
  - 例如：W1 = W0 -step*grad(f(W0), W0)。
    - step 是一個很小的數，曲率不一定會很小，要藉由乘以 step 確保 W1 與 W0 不會離太遠。

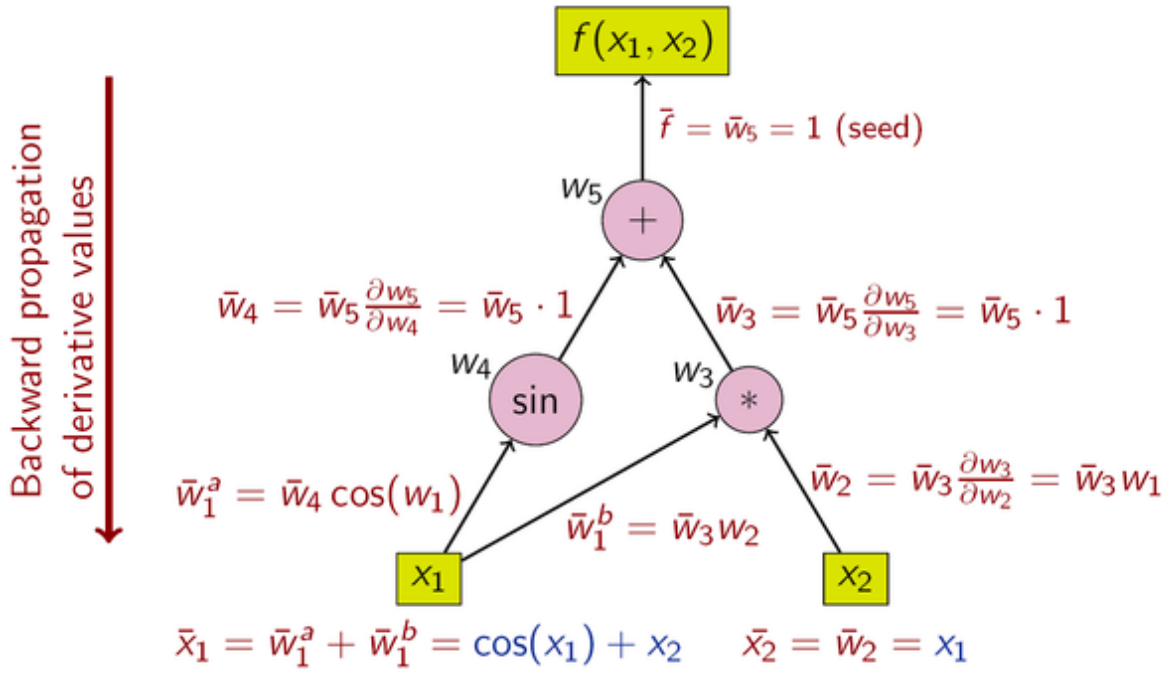
2-4-3 隨機梯度下降

- 學習率（learning rate），是一個純量因子，調整梯度下降的速度
  - 太小 → 陷入區域最小值。
  - 太大 → 跳到不相干的位置，掠過真正的最小值。
- 小批次隨機梯度下降（mini-batch stochastic gradient descent, mini-batch SGD）
  - 每批次資料是隨機抽取的。
  - 批次可能小到只取單一筆樣本和標籤。
- SGD 變體有：動量（momentum）SGD、Adagrad、RMSProp 等等。
- 所有的 SGD 稱為最佳化方法（optimization methods）或優化器（optimization）。
- 動量解決了 SGD 的兩個問題，收斂速度和區域最小值
  - 學習率低 SGD 找出最佳參數，過程會停留在 local minimum 而不是達到 global minimum。
  - 如果有足夠動量就不會陷入一般低谷中，而會停止於 global minimum。

2-4-4 連鎖導數：反向傳播 Backpropagation 演算法

- 反向傳播演算法（Backpropagation algorithm）

    - 藉助簡單運算的導數，得出簡單運算的複雜組合之梯度。
    - 透過連鎖率（chain rule）求得連鎖函數的導數。
        - 如果想要知道某節點對其他節點的導數，只需要將兩節點間的導數相乘即可。
- 計算圖（computation graph）進行自動微分（automatic differentiation）。



- TensorFlow 中的梯度磁帶（Gradient Tape）
    - 利用 Python 的 with 區塊，自動將區塊中的張量運算過程記錄為計算圖。
    - 自動計算出任意輸出相對於任意變數/變數組的梯度。

2-5 重新檢視我們的第一個例子

- 神經層組合而成的模型。
- 將輸入資料映射至預測值。
- 損失函數將預測值和目標進行對比，產生損失值，以衡量模型預測能力的好壞。
- 優化器用該損失值更新模型權重。

```
from keras import models
from keras import layers
from keras.utils import to_categorical
from keras.datasets import mnist

# 輸入資料
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28)) # 重塑成 (60000, 784) 的
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28)) # 重塑成 (10000, 784) 的資
test_images = test_images.astype('float32') / 255

# 對標籤進行 OneHot 編碼
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

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

# 神經網路編譯
network.compile(optimizer='rmsprop', # 梯度下降規則 RSMProp
                loss='categorical_crossentropy', # 損失函數, 過程中嘗試最小化的數值
                metrics=['accuracy'])

# 訓練迴圈
# 128 個樣本的小批次訓練資料來作 5 次 epochs, 共進行 2345 次梯度更新
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

2-5-1 從頭開始重新建構模型 → 請至 GitHub 上觀看（選擇性觀看）