【Chapter 01】人工智慧、機器學習、深度學習介紹

* 關係整理：
  + Artificial Intelligence(人工智慧)：
    - 起源於1950年代，人工智慧之父圖靈(Alan Mathison Turing)提出「圖靈測試」
    - 1980年代約翰瑟爾(John Searle)提出對「人工智慧」分類方式：
      * Strong AI(強人工智慧)：機器能具有與人類相同完整的認知能
      * Week AI(弱人工智慧)：機器不需要具有與人類相同完整的認知能力，只要設計得看起來像具有智慧就即可
    - 加速發展的原因：
      * 大數據Big Data分散式儲存(如Hadoop、NoSQL等)與平行運算(如Spark等)
      * GPU、TPU平行運算
        + GPU (Graphics Processing Unit)：圖形處理器，具有數千個小型且高效率的核心，GPU透過大量核心進行平行運算，可讓深度學習訓練比CPU快10~75倍
        + CPU：具有數顆核心，為循序處理進行最佳化
        + TPU (Tensor Processing Unit)：專為深度學習特定用途設計的特殊規格邏輯IC，用於深度學習訓練速度更快(2016年Google發布)
  + Machine Learning(機器學習)：
    - 透過演算法，使用大量的歷史資料進行訓練，訓練完成後會產生模型，未來當有新的資料，我們可以使用訓練產生的模型進行預測
    - 應用如：推薦引擎、定向廣告、需求預測、垃圾郵件過濾、醫學診斷、自然語言處理、搜索引擎、詐騙偵測、證券分析、視覺辨識、語音辨別、手寫辨別等
    - 類型
      * Supervised Learning(監督式學習)：
        + 具備Feature與Label，透過演算法訓練並建立模型，當有新的資料，我們就可以使用模型進行預測，

類型：

分類

二元分類(Binary Classification)：Label只有2種選項，好像是非題

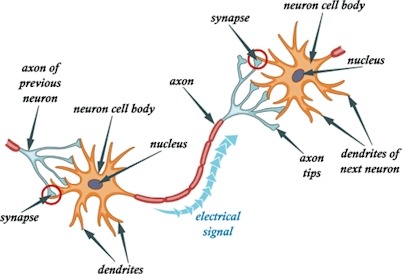
多元分類(Multi Class Classification)：Label有多個選項，好像選擇題

迴歸分析(Regression)：Label是連續的值，好像計算題

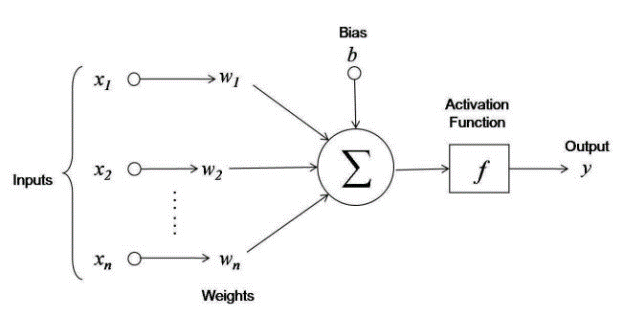
* + - * Unsupervised Learning(非監督式學習)：
        + 從現有資料我們不知道要預測的答案，所以沒有Label，如集群(Clustering)演算法，將資料分成幾個相異性最大的群組，而群組內的相似程度最高
        + 類型：群集(Clustering)
      * Reinforcement Learning(增強式學習)：
        + 藉由定義：動作(Action)、狀態(States)、獎勵(Rewards)的方式，不斷訓練機器循序漸進，學會執行某項任務的演算法
        + 常見演算法：Q-learning、TD (Temporal Difference)、Sarsa
  + Deep Learning(深度學習)：模擬人類神經網路的運作方式
    - 應用於視覺辨識、語音識別、自然語言處理、生物醫學等領域
    - 常見架構：
      * 多層感知器(Multilayer Perceptron)
      * 深度神經網路DNN(Deep Neural Network)
      * 卷積神經網路CNN(Convolutional Neural Network)
      * 遞迴神經網路RNN(Recurrent Neural Network)
    - 以大量矩陣運算模擬神經元運作，矩陣運算的特性是：單一運算都很簡單，但需要大量運算，適合平行運算
    - 通常有1個輸入層、1個輸出層、隱藏層可以非常多層，稱為深度學習
* 機器學習
  + 訓練的資料組成：
    - Features：資料的特徵，如濕度、風向、風速、季節、氣壓
    - Label：資料的標籤，希望預測的目標，如是否降雨、天候、氣溫
  + 階段：
    - 訓練(Training)：訓練資料是過去累積的歷史資料，經過Feature Extraction(特徵萃取)，產生Feature與Label，然後經過機器學習演算法訓練後產生模型
    - 預測(Predict)：新輸入資料，經過Feature Extraction(特徵萃取)產生Feature，使用訓練完成的模型進行預測，最後產生預測結果

【Chapter 02】深度學習的原理

* 神經傳導



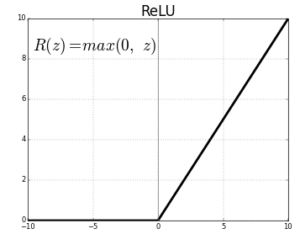
* + 神經系統包含許多神經元(neurons)
  + 軸突(axon)：
    - 神經元長出一細長條的軸突，最長可達一公尺，最短只有幾十分之一毫米【長且粗】
    - 負責以電流方式將訊息傳遞給另一個神經元【傳送】
  + 樹突(dendrites)：
    - 較短且綿密
    - 用於接受其他神經元傳來的電化學訊息，再傳遞給本身的細胞【接收】
  + 突觸(synapse)：
    - 負責輸入與輸出的神經元傳遞的機制
    - 神經元透過釋放化學物質來傳遞訊息，當電壓達臨界值，就會透過軸突傳送電脈衝動作電位，再由突觸決定是否回收或傳遞下去至接收神經元
* 以數學公式模擬神經元的訊息傳導
  + ，其中
    - 為模擬輸入神經元，為模擬軸突
    - 圖中紅色觸為模擬突觸機制
    - y為模擬接收神經元，圖為1個接收神經元
    - 代表接收神經元被活化的程度，偏差值月高，越容易被活化並傳遞訊息
    - 常見activation function：sigmoid、relu



* 激活函數(activation function)：有接收神經刺激使神經元活化之意
  + 模擬神經傳導的運作方式，將上一層神經元訊號傳遞至下一層
  + 通常為非線性函數，讓神經網路可以處理比較複雜的非線性問題
  + 常見激活函數
    - Sigmoid
      * 公式：
      * 圖：



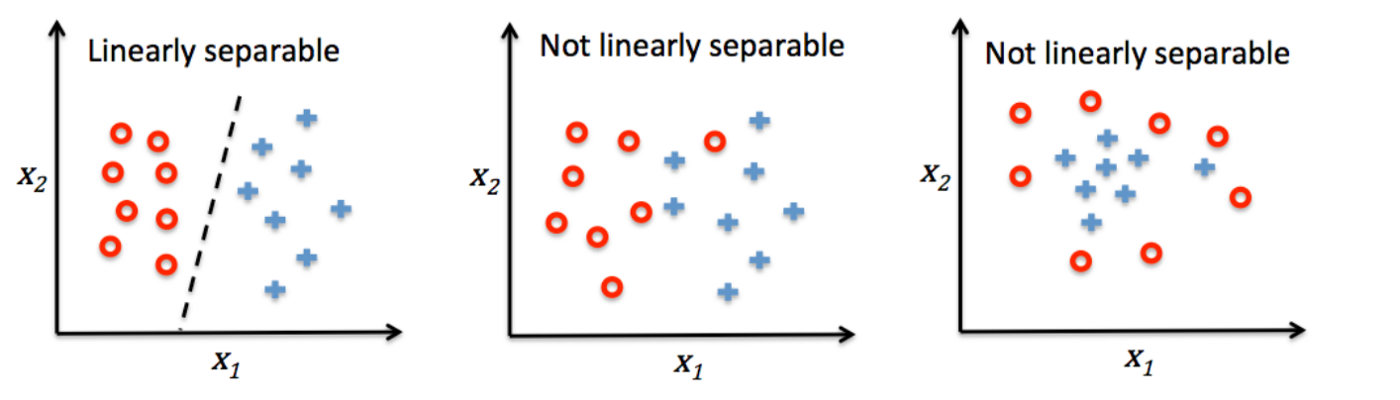
* + - * 特性：與人類感覺神經對訊號的接受類似，當接收神經元接收刺激的總和
        + 小於臨界值，會忽略此刺激：當x<-5，輸出y接近0
        + 大於臨界值，開始接收神經刺激：當x在-5與5之間，隨著x增加，y亦增加
        + 當刺激達到一定程度，感覺會開始鈍化：即使更大的刺激，感覺仍維持不變，當x>5，y趨近於1
    - relu
      * 圖



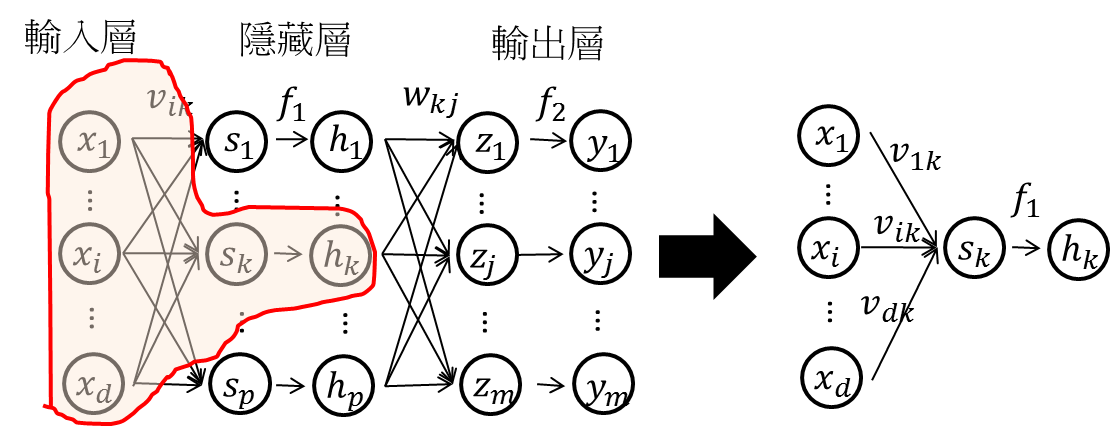
* + - * 特性：與人類感覺神經對訊號的接受類似，當接收神經元接收刺激的總和
        + 小於臨界值，會忽略此刺激：當x<0，y為0
        + 大於臨界值，開始接收神經刺激：當x>0，y等於x
* 以矩陣運算模擬神經網路

→

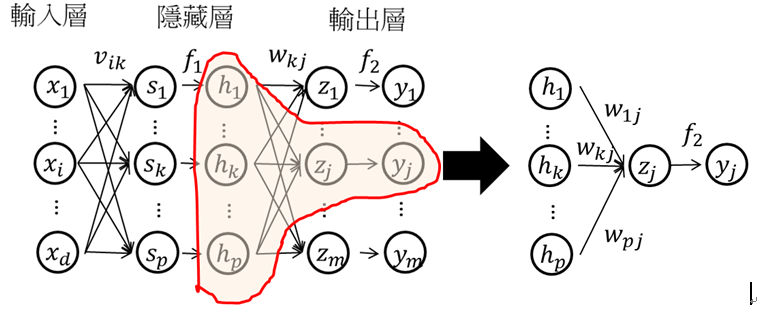
* + 中文表示：輸出=激活函數(輸入x權重+偏差)
* 多層感知器模型(Multilayer perceptron)
  + 感知器Perceptron（稱為Perceptron Learning Algorithm簡稱PLA）
    - 在資料是線性可分的形況下才能正確分類（演算法才會停止）



* + - 優點：最簡單的線性分類演算法
    - 缺點：
      * 一定要線性可分Perception演算法才會停下來
      * Perception演算法的錯誤率不會逐步收斂
      * Perception演算法只知道結果是A類還B類，但沒辦法知道是A, B類的機率是多少
    - 更新預測錯誤
    - 參考資料：https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC3-2%E8%AC%9B-%E7%B7%9A%E6%80%A7%E5%88%86%E9%A1%9E-%E6%84%9F%E7%9F%A5%E5%99%A8-perceptron-%E4%BB%8B%E7%B4%B9-84d8b809f866
  + 多層感知器模型(Multilayer perceptron)
    - 多層感知機是一種前向傳遞類神經網路，至少包含三層結構(1個輸入層、1個輸出層、1個隱藏層)，並且利用到「反向傳遞」的技術達到學習(model learning)的監督式學習
    - MLP是深度神經網路(deep neural network, DNN)的一種special case
    - 說明：
      * 前向傳遞(Forward propagation)



* + - * + 輸入層到隱藏層的值為，為輸入訊號的加權線性和(為第個輸入到第個hidden node的權重)，
        + 經過非線性轉換/激活函數(activation function，)後，得到hidden node的輸出，



* + - * + 隱藏層到輸出層的值為，為hidden node輸出的加權線性和(為第個hidden node輸出到第個輸出值的權重)，
        + 經過非線性轉換/激活函數(activation function，)後，得到hidden node的輸出，
      * 反向傳遞(Backward propagation)：從錯誤中學習
        + 利用最後的目標函數(loss/cost function)來進行參數的更新，一般來說都是用誤差均方和(mean square error)當作目標函數
        + 為第筆資料的輸入值，其輸出值為，其目標的誤差為
        + 所有樣本的誤差和當作目標函數，最佳化的目的就是讓「所有樣本的誤差均方和」越小越好，所以目標是
        + 要找到最佳參數解(參數只有和)，最簡單的方式就是微分方程式等於0找解
        + 但參數量多無法直接找到唯一解(後面公式有偏微分後的結果，很難直接找到唯一解)，所以還是需要依賴gradient descent找最佳解。

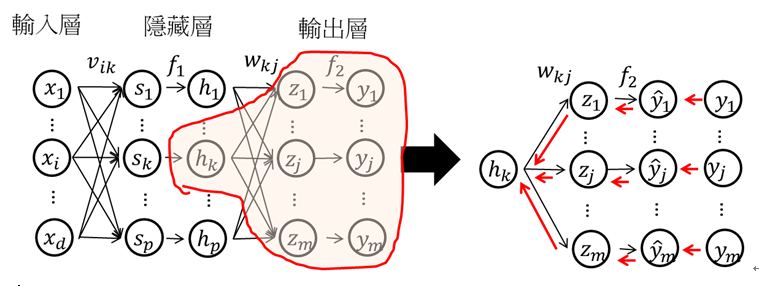
其中 為學習率(learning rate)

(輸出到隱藏層)

(隱藏層到輸入層)

基本上微分解無法直接算出，因此用chain rule方式，可以更有效得到解，以下針對不同層別的連結算倒傳遞(只針對一個樣本去計算)

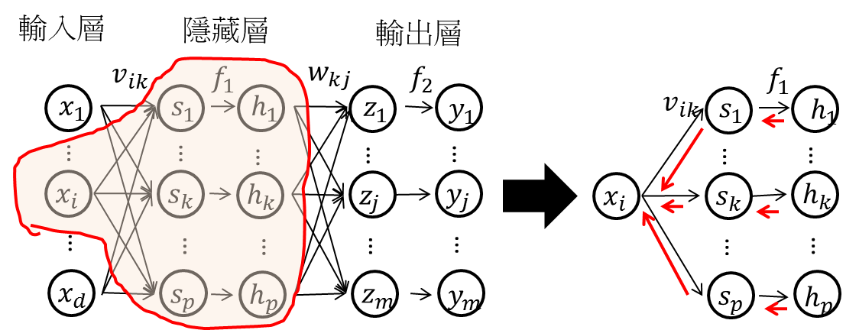
輸出到隱藏層()



chain rule：

可得

隱藏層到輸入層()



chain rule：

可得

MLP神經網路只是在利用gradient descent找最佳參數解，最後帶入MLP內的前向傳遞 (Forward propagation)即可得到最後的預測值。

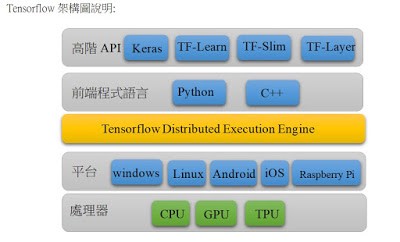
* + - 權重更新(weight update)
      * 損失函數(loss function)：使用損失函數計算模型輸出(預測的結果)與label(數字影響真實的值)之間的誤差
        + Cross Entropy：，其中 為Label， 為預測的結果，為常用的損失函數
      * 最優化方法(Optimizer)：依照誤差值更新神經元連結的權重與偏差，盡量使損失函數的誤差值最小化
        + 隨機梯度下降法(Stochastic gradient descent, SGD)

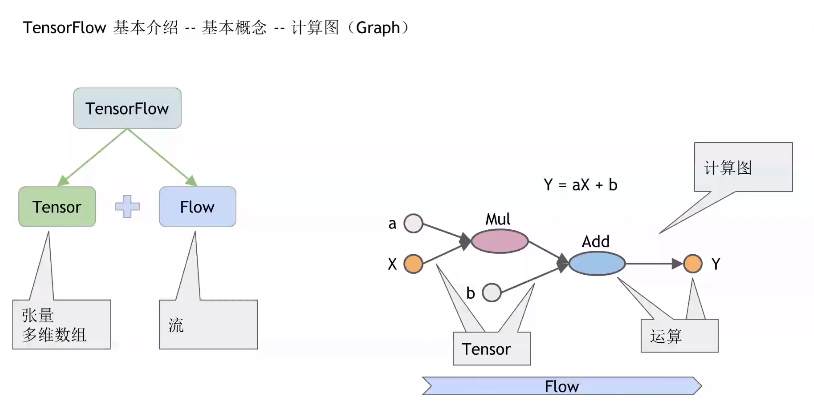
在所有「權重與偏差的組合」所組成的高維度空間中，每個訓練批次沿著每個維度Loss下降的方向走一小步，經過許多次步驟，就可以找到最佳化的「權重與偏差的組合」

* + - * + RMSprop、Adagrad、Adadelta、adam
    - 參考資料：
      * https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-%E5%A4%9A%E5%B1%A4%E6%84%9F%E7%9F%A5%E6%A9%9F-multilayer-perceptron-mlp-%E5%90%AB%E8%A9%B3%E7%B4%B0%E6%8E%A8%E5%B0%8E-ee4f3d5d1b41
      * Optimizer：http://www.denizyuret.com/2015/03/alec-radford-animations-for.html

【Chapter 03】TensorFlow與Keras

* 機器學習或深度學習的應用，如Google的過濾垃圾信、語音辨識、圖片辨識、翻譯等
* TensorFlow架構圖



* + 處理器：TensorFlow可以在CPU(中央處理器)、GPU(圖形處理器)、TPU(google研發專屬晶片)上執行
  + 平台：TensorFlow具有跨平台能力，像是Windows、Linux、Raspberry pi…等，可以在目前主流的平台執行
  + TensorFlow Distributed Execution Engine(分散式執行引擎)
    - 具備分散式運算能力，可同時在數百台的機器上執行訓練模型，大幅縮短模型訓練的時間
  + 前端程式語言：TensorFlow可支援多種前端程式語言，如python、C++等
  + 高階API：TensorFlow是較低階的深度學習API，設計模型時須自行設計，如張量乘積、卷積等底層操作，亦可開發許多種高階的API，如Keras、TF-Learn、TF-Slim、TF-Layer等，建構出各種複雜的深度學習模型
* TensorFlow簡介
  + 由Tensor與Flow所組成
  + Tensor張量
    - 幾何實體或廣義的「數量」，包含0維為純量(數值)，1維為向量，2維以上為矩陣
  + Flow資料流程
    - 讓TensorFlow可支援不同的程式語言介面，且在各種平台執行，所有的TensorFlow程式先建立「計算圖」(computational graph)
* TensorFlow程式設計模式
  + 建立「計算圖」：使用TensorFlow提供的模組，可設計張量運算流程，並建構各種深度學習或機器學習模型

Import tensorflow as tf

Import numpy as np

W = tf.Variable(tf.random\_normal([3, 2]), name=’W’)

b = tf.Variable(tf.random\_normal([1,2]), name=’b’)

y = tf.nn.sigmoid(tf.matmul(X, W)+b, ‘y’)

* + 執行「計算圖」：建立「Session」執行，其作用在用戶端和執行裝置之間建立連結，可將計算圖在各種不同裝置中執行，後續任何與裝置間的資料傳遞，都必須透過Session，並最後取得執行後的結果

with tf.Session() as sess:

init = tf.global\_variables\_initializer()

sess.run(init)

X\_array = np.array([[0.4, 0.2, 0.4],

[0.3, 0.4, 0.5],

[0.3, -0.4, 0.5]])

(\_b, \_W, \_X, \_y) = sess.run((b, W, X, y), feed\_dict={X:X\_array})

* Keras介紹
  + 是一個開放原始碼，基於Python高階深度學習的程式庫，能運作在TensorFlow或Theano之上
  + 主要作者維護者是Google工程師Francois Chollet
  + Keras可快速又方便運算的主要原因
    - 已將訓練模型的輸入層、隱藏層、輸出層做好架構，使用者只需要加入且填寫正確的參數，如神經元個數、activation function的函式…等，花費最少時間建立深度學習模型(建立、訓練、預測等功能)
    - 運作方式為深度學習底層的運作，如：張量(矩陣)運算，Keras須配合使用「後端引擎(backend engine)」進行運算
    - 特色
      * 簡單快速的建立原型prototyping：具備友善的使用者介面、模組化設計、可擴充性
      * 內建各式類神經網路層級，如CNN、RNN，可幫助快速建立神經網路模型
      * 透過後端引擎：Theano與TensorFlow，可在CPU與GPU運行
* Keras程式設計模式
  + 建立多層感知器(MLP)
    - 建立Sequential模型

from keras.models import Sequential

model = Sequential()

* + - 建立輸入層(784)、隱藏層(256)至模型

model.add(Dense(units=256, input\_dim=784, kernel\_initializer='normal', activation='relu'))

* + - 建立輸出層()至模型

model.add(Dense(units=10, kernel\_initializer='normal', activation='softmax'))

* Keras與TensorFlow比較

|  |  |  |
| --- | --- | --- |
|  | Keras | TensorFlow |
| 學習難易度 | 簡單 | 比較困難 |
| 使用彈性 | 中等 | 高 |
| 開發生產力 | 高 | 中等 |
| 執行效能 | 高 | 高 |
| 適合使用者 | 初學者 | 進階使用者 |
| 張量(矩陣)運算 | 不須自行設計 | 需自行設計 |

【Chapter 04】在Windows安裝TensorFlow與Keras

* TensorFlow官網：https://www.tensorflow.org/
* 安裝Anaconda
  + https://medium.com/python4u/anaconda%E4%BB%8B%E7%B4%B9%E5%8F%8A%E5%AE%89%E8%A3%9D%E6%95%99%E5%AD%B8-f7dae6454ab6
* 啟動命令提示字元
  + 啟動命令提示字元：按下搜尋圖示→輸入「cmd」→點選「命令提示字元」
  + 按下Ctrl+Space切換為英文，再輸入命令
  + 設定：點選圖示→點選「內容」→點選「色彩」

→點選「螢幕背景」→點選「黑色」

→點選「螢幕文字」→點選「白色」→按下「確定」

* 建立TensorFlow的Anaconda虛擬環境
  + 建立且切換工作目錄
    - 建立：md \pythonwork
    - 切換：cd \pythonwork
  + 建立anaconda虛擬環境
    - conda create --name tensorflow python=3.5 anaconda

建立虛擬環境 名稱為tensorflow python版本為3.5 加入，會同時安裝基本套件

* + 啟動anaconda虛擬環境：activate tensorflow
  + 關閉tensorflow的anaconda虛擬環境：deactivate tensorflow
* 在Anaconda虛擬環境安裝TensorFlow與Keras
  + 在啟動虛擬環境下
    - 安裝TensorFlow CPU版：pip install tensorflow
    - 安裝Keras：pip install keras
* 啟動Jupyter Notebook
  + 在啟動虛擬環境下，啟動Jupyter Notebook：jupyter notebook
  + 建立新的notebook：點選「New」→點選「Python 3」
  + 新增的Jupyter Notebook：可更改名稱，輸入新名稱後按下「OK」
  + Jupyter Notebook輸入命令方式
    - Shift + Enter：執行程式碼，且游標會移至下一Cell
    - Ctrl + Enter：執行程式碼，且游標仍在目前的Cell
  + 匯入模組
    - 匯入TensorFlow模組，後續以tf引用此模組：import tensorflow as tf
    - 匯入Keras模組：import keras
  + 查看版本
    - 查看TensorFlow版本：tf.\_\_version\_\_
    - 查看Keras版本：keras.\_\_version\_\_
  + 儲存notebook：按下「儲存」，並於畫面上顯示儲存的時間
  + 關閉notebook：點選「File」→點選「Close and Halt」
  + 開啟之前儲存的NoteBook：點選「.ipynb」
  + 關閉Jupyter Notebook瀏覽器：關閉瀏覽器
  + 關閉Jupyter Notebook：在命令提示字元視窗，按下Ctrl+C

【Chapter 05】在Linux Ubuntu安裝TensorFlow與Keras

* 安裝Anaconda：
  + 複製安裝Anaconda下載網址：
    - 輸入網址 (https://www.anaconda.com/products/distribution) →點選Download for Linux→在Linux 64-bit按下滑鼠右鍵→選取Copy Link Location
  + 下載Anaconda3-4.2.0-Linus-x86\_64.sh
    - 【終端機】wget 下載網址(https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86\_64.sh)
  + 安裝Anaconda：
    - 【終端機】bash Anaconda3-4.2.0-Linux-x86\_64.sh -b
    - 其中「-b」是batch批次安裝，會自動省略：閱讀License條款，並自動安裝至/home/user/anaconda3路徑
  + 編輯~/.bashrc加入模組路徑
    - 【終端機】sudo gedit ~/.bashrc
    - 編輯~/.bashrc→將anaconda執行檔路徑加入PATH，在不同路徑皆可執行anaconda
  + 使~/.bashrc修改生效
    - 【終端機】source ~/.bashrc 或重新登入登出使使用者環境變數設定生效
  + 查看python版本
    - 【終端機】python --version
  + 參考資料：
    - 【Ubuntu Linux 安裝 Anaconda】https://officeguide.cc/ubuntu-linux-install-anaconda-data-science-platform-tutorial/
* 安裝TensorFlow與Keras
  + 【終端機】安裝TensorFlow：pip install tensorflow
  + 【終端機】安裝Keras：pip install keras
* 啟動Jupyter Notebook
  + 建立且切換工作目錄
    - 建立：【終端機】mkdir -p ~/pywork
    - 切換：【終端機】cd ~/pywork
  + 進入Jupyter Notebook
    - 【終端機】jupyter notebook
  + 啟動Jupyter Notebook：同Windows說明

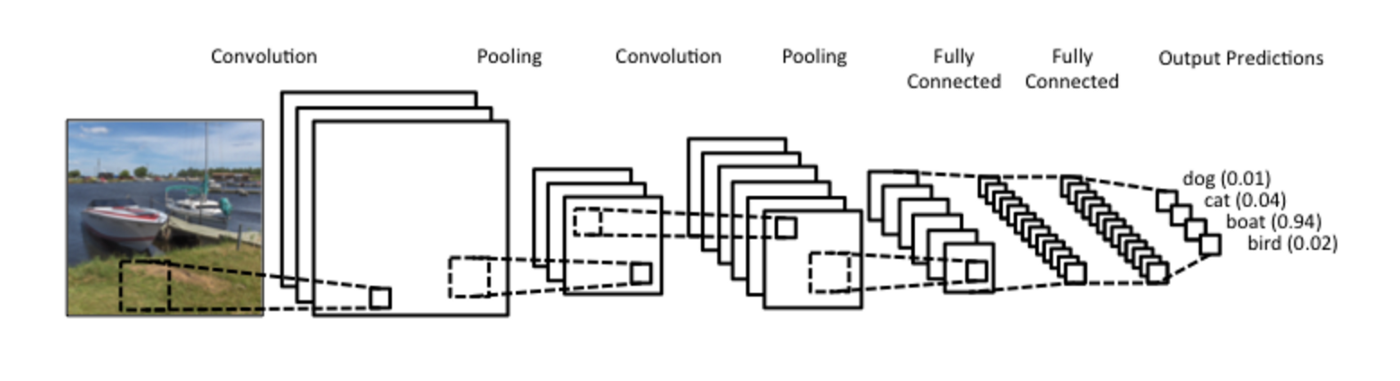
【Chapter 06】Keras MNIST手寫數字辨識資料集介紹

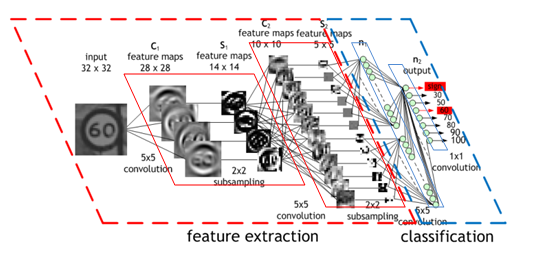
【Chapter 07】Keras 多元感知器(MLP)辨識手寫數字

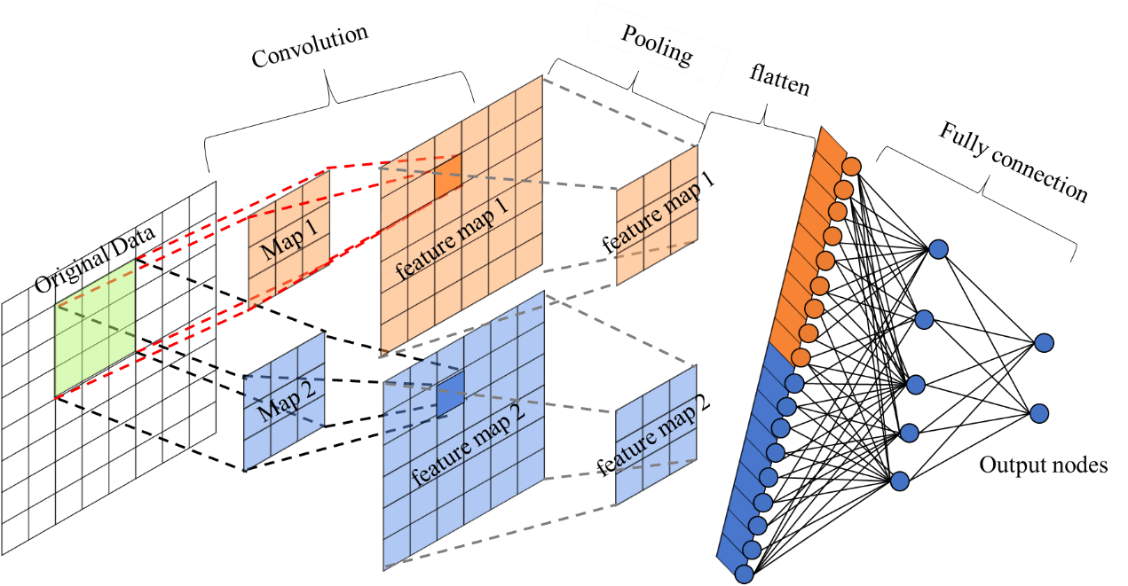
* 以github來做筆記：Sample\_MLP\_MNIST手寫數字.ipynb
* 準確率接近0.9797
* Overfitting過度擬合的現象

【Chapter 08】Keras 卷積神經網路(CNN)辨識手寫數字

* 準確率接近0.99
* 卷積神經網路(CNN)
  + 由一位計算機科學家揚．勒丘恩(Yann LeCun)所提出
  + MLP與CNN的差異：CNN增加特徵提取層(卷積層1[Convolution]、池化層1[Pooling]、卷積層2、池化層2)的處理提取特徵
  + CNN概念



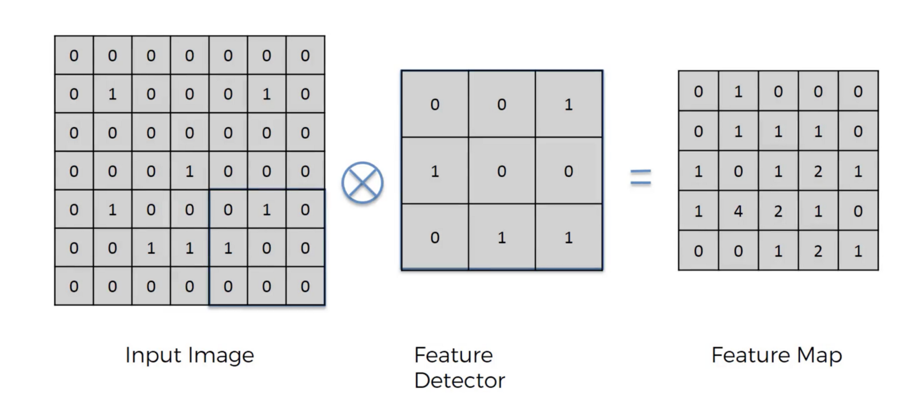




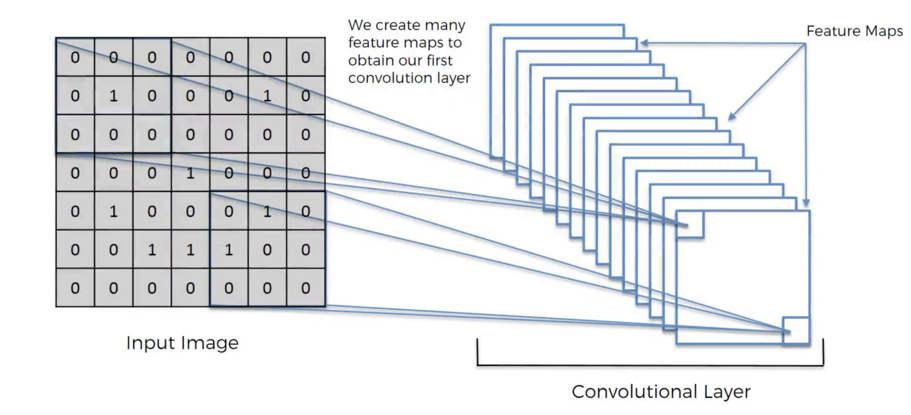
* + - 範例說明：卷積運算的效果類似濾鏡，產生多個影像並擷取不同的特徵，好像將相片卷積起來

|  |  |  |  |
| --- | --- | --- | --- |
|  | 說明 | 影像數 | 影像大小 |
| 輸入層 | 數字影像28X28大小，如數字7的影像 | 1個 | 28X28 |
| 第1次卷積運算 | 產生16個影像，卷積運算並不會改變影像大小 | 16個 | 28X28 |
| 第1次縮減取樣 | 將16個28X28影像，縮小為16個14X14的影像 | 16個 | 14X14 |
| 第2次卷積運算 | 將原本16個的影像，轉換為36個影像 | 36個 | 14X14 |
| 第2次縮減取樣 | 將36個14X14影像，縮小為36個7x7的影像 | 36個 | 7x7 |
| 平坦層 | 將36個7x7的影像以reshape轉換為1維的向量 |  | 36x7x7=1764 個float數字，作為神經元的輸入 |
| 隱藏層 | 共有128個神經元 |  | 128個神經元 |
| 輸出層 | 共有10個神經元，對應到0~9十個數字 |  | 10個神經元 |

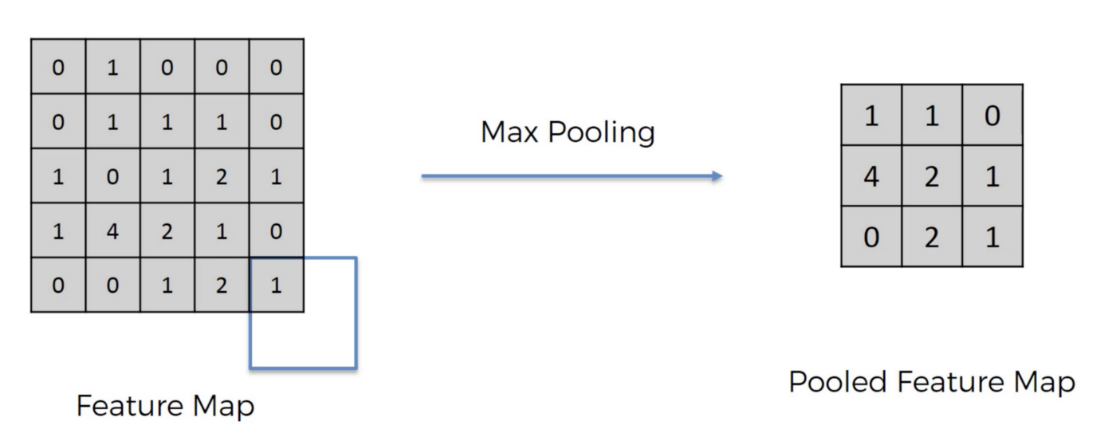
* + 卷積運算
    - 以隨機方式產生，filter weight大小是3x3
    - 要轉換的影像由左而右、由上而下，依序選取3x3的矩陣
    - 影像選取的矩陣(3x3)與filter weight(3x3)乘積，計算產生第1列、第1行的數字(padding=’SAME’會在邊界知外補0)
    - 再以相同的方式，計算第1列、第2列的數字，依照上述相同方式，依序完成所有運算，就完成影像的處理



* + - 使用多個filter weight卷積運算產生多個影像，以數字影像為例，使用隨機產生n個filter weight(W)濾鏡，此作法可幫助提取輸入不同特徵，如邊緣、線條和角等



* + - Max-Pool運算說明：可將影像縮減取樣(downsampling)，影像大小減半，取減半影像中最大值，但影像數不變



* + - * 減少須處理的資料點：減少後續運算所需的時間
      * 讓影像位置差異變小：如手寫數字7，位置上下左右可能不同，但是位置的不同可能會影響辨識，減少影像大小，讓數字的位置差異變小
      * 參數的數量和計算量下降：在一定程度上控制了過度擬合(overfitting)
    - 建立卷積神經網路辨識Mnist資料集：以github來做筆記：Sample\_CNN\_MNIST手寫數字.ipynb
      * 資料預處理
      * 建立模型
      * 訓練模型
      * 評估模型準確率
      * 進行預測
* 參考資料：https://chih-sheng-huang821.medium.com/%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-convolutional-neural-network-cnn-cnn%E9%81%8B%E7%AE%97%E6%B5%81%E7%A8%8B-ecaec240a631

【Chapter 09】Keras Cifar-10影像辨識資料集介紹

【Chapter 09】Keras 卷積神經網路(CNN)辨識Cifar-10影像

* 以github來做筆記：Sample\_CNN\_Cifar-10影像辨識.ipynb
* CNN說明：卷積運算的效果類似濾鏡，產生多個影像並擷取不同的特徵，好像將相片卷積起來

|  |  |  |  |
| --- | --- | --- | --- |
|  | 說明 | 影像數 | 影像大小 |
| 輸入層 | 輸入的照片32X32大小，如「船」的影像 | 1個 | 32X32 |
| 第1次卷積運算 | 產生32個影像，卷積運算並不會改變影像大小 | 32個 | 32X32 |
| 第1次縮減取樣 | 將32個32X32影像，縮小為32個16X16的影像 | 32個 | 16X16 |
| 第2次卷積運算 | 將原本32個的影像，轉換為64個影像 | 64個 | 16X16 |
| 第2次縮減取樣 | 將64個16X16影像，縮小為64個8x8的影像 | 64個 | 8x8 |
| 平坦層 | 將64個8x8的影像以reshape轉換為1維的向量 |  | 64x8x8=4096 個float數字，作為神經元的輸入 |
| 隱藏層 | 共有1024個神經元 |  | 1024個神經元 |
| 輸出層 | 共有10個神經元，對應到0~9個影像 |  | 10個神經元 |

* CNN說明：建立3次的卷積運算的神經網路架構

|  |  |  |  |
| --- | --- | --- | --- |
|  | 說明 | 影像數 | 影像大小 |
| 輸入層 | 輸入的照片32X32大小，如「船」的影像 | 1個 | 32X32 |
| 第1次卷積運算 | 產生32個影像，卷積運算並不會改變影像大小 | 32個 | 32X32 |
| 第1次縮減取樣 | 將32個32X32影像，縮小為32個16X16的影像 | 32個 | 16X16 |
| 第2次卷積運算 | 將原本32個的影像，轉換為64個影像 | 64個 | 16X16 |
| 第2次縮減取樣 | 將64個16X16影像，縮小為64個8x8的影像 | 64個 | 8x8 |
| 第3次卷積運算 | 將原本64個的影像，轉換為128個影像 | 128個 | 8X8 |
| 第3次縮減取樣 | 將128個8X8影像，縮小為128個4x4的影像 | 128個 | 4x4 |
| 平坦層 | 將128個4x4的影像以reshape轉換為1維的向量 |  | 128x4x4=2048 個float數字，作為神經元的輸入 |
| 隱藏層1 | 共有2500個神經元 |  | 2500個神經元 |
| 隱藏層2 | 共有1500個神經元 |  | 1500個神經元 |
| 輸出層 | 共有10個神經元，對應到0~9個影像 |  | 10個神經元 |

【補充】

* 資料類型：
  + 結構資料：數字型資料
  + 非結構資料：圖像、影像、聲音、文字…
* 轉換方式
  + One Heat：
  + Emabing：

【Chapter】