|  |
| --- |
| 作品名稱：mnist手寫數字辨識 |
| 一、說明 |
| 深度學習就是透過各種類神經網路，本專題會使用多層感知器(MLP)、卷積神經網路(CNN)、循環神經網路(RNN)透過mnist資料集產出訓練出來的值，將一大堆的數據輸入神經網路中，讓電腦透過大量數據的訓練找出規律自動學習，最後讓電腦能依據自動學習累積的經驗作出預測。 |
| 二、論文 |
| 多層感知器(MLP)：多層感知器是由多層人工神經元組成的類神經網路, 在 MINST 資料集的手寫數字辨識中要用到的 MLP 為如下具有輸入層, 一個隱藏層, 以及輸出層的類神經網路。    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)。   * 輸入層：數字圖片是一張28\*28的圖片、共有784個神經元所組成了神經網路第一層，數值的範圍介於0~1之間。灰階0代表灰色、1代表白色，又稱為激勵值，數值越大則該神經元就越亮。 * 輸出層：完成輸入層後先不管其他層的內容，我們來看看他最右方的輸入層，也就是最後判斷的結果，其中有10個神經元，各代表數字0~9，其中也有代表的激勵值。 * 隱藏層：為了方便說明，在這裡我們設計了兩個隱藏層，每層有16個神經元。在真實的案例中可依據需求設置調整隱藏層與神經元的數量。   卷積神經網路(CNN)：它是目前深度神經網路(Deep Neural Network)領域發展的主力，在圖片辨別上甚至可以做到比人類還精準之程度。   * 結構圖，和多層感知器相比較，卷積神經網路增加卷積層1、池化層1、卷積層2、池化層2，提取特徵後再以平坦層將特徵輸入神經網路中。以下使用MNIST資料及進行說明：     圖片中最上方有卷積層1、池化層1、卷積層2、池化層2，將原始的圖片以卷積、池化處理後產生更多的特徵小圖片，作為輸入的神經元。   * 卷積層：是將原始圖片與特定的濾鏡(Feature Detector)進行卷積運算，你也可以將卷積運算看成是原始圖片濾鏡特效的處理，filters可以設定濾鏡數目，kernel\_size可以設定濾鏡(filter)大小，每一個濾鏡都會以亂數處理的方式產生不同的卷積運算，因此可以得到不同的濾鏡特效效果，增加圖片數量。 * 池化層：是採用Max Pooling，指挑出矩陣當中的最大值，相當於只挑出圖片局部最明顯的特徵，這樣就可以縮減卷積層產生的卷積運算圖片數量。   循環神經網路(RNN)：它是「自然語言處理」領域最常使用的神經網路模型，LSTM因為RNN前面的輸入和後面的輸入具有關連性，即可以建立回饋迴路。也可以用於語言翻譯、情緒分析、氣象預測及股票交易等。   * 結構圖：循環神經網路中主要有三種模型，分別是SimpleRNN、LSTM和GRU。因為SimpleRNN超簡單，效果不夠好，記不住長期的事情，所以又發展出長短記憶網路(LSTM)，然後LSTM又被簡化為閘式循環網路GRU。     如圖共有三個時間點依序是t-1、t、t+1，在t的時間點：   * X1是神經網路t時間點的輸入，Ot是神經網路t時間點的輸出。 * (U,V,W)都是神經網路共用的參數，W參數是神經網路t-1時間點輸出，並且也作為神經網路t時間點的輸入。 * S1是隱藏狀態，代表神經網路上的記憶，是神經網路目前時間點的輸入X1加上上個時間點的狀態St-1，再加上U與W的參數，共同評估之結果：   St = f(U \* Xt + W \* St-1)  簡單來說就是前面的狀態會影響現在的狀態，現在的狀態也會影響以後的狀態。 |
| 三、實作 |
| MNIST資料集是由紐約大學 Yann Le Cun 教授蒐集整理很多0~9的手寫數字圖片所形成的資料集，這是一個大型手寫數字資料庫，對於機器學習學者來說是初學者，圖片每張大小為28\*28、皆為灰階影像，每個像素為0~255之數值、資料庫當中包含了60000筆的訓練資料、10000筆的測試資料。在MNIST資料集中，每一筆資料都是由下載好的mnist的資料實作出成果。     1. 使用多層感知(MLP)進行辨識訓練：   (1) 建立模型與資料結構：   |  | | --- | | #導入相關套件  import os  os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  from keras.datasets import mnist # keras.datasets：載入MNIST資料集  from keras.models import Sequential # Keras：建立訓練模型  from keras.layers import Dense, Activation, Dropout  from keras.utils import np\_utils  import numpy as np # Numpy：矩陣運算  %matplotlib inline # matplotlib.pyplot 將資料視覺化，可以圖表呈現結果  import matplotlib.pyplot as plt  (x\_train\_image, y\_train\_label), (x\_test\_image, y\_test\_label) = mnist.load\_data() # 呼叫 load\_data() 載入 MNIST 資料集  nb\_classes = 10 # 類別的數目  x\_train\_image = x\_train\_image.reshape(60000, 784).astype('float32')  x\_test\_image = x\_test\_image.reshape(10000, 784).astype('float32')  # 壓縮圖片顏色至0 ~ 1  x\_train\_image /= 255  x\_test\_image /= 255  #依分類數量將圖片標籤轉換格式的陣列  y\_train\_cat = np\_utils.to\_categorical(y\_train\_label, nb\_classes)  y\_test\_cat = np\_utils.to\_categorical(y\_test\_label, nb\_classes)  model = Sequential()  model.add(Dense(50, input\_shape=(784,)))  model.add(Dense(units=nb\_classes))  model.add(Activation('softmax'))  # 定義定義損失函數、優化函數及成效衡量指標  model.compile(loss='categorical\_crossentropy', optimizer='SGD', metrics=['accuracy'])  model.summary() |   執行結果：    (2)開始訓練：   |  | | --- | | epochs = 10  history = model.fit(x\_train\_image, y\_train\_cat, epochs=epochs, batch\_size=128,verbose=1) |   執行結果：    (3)執行後結果：   |  | | --- | | #訓練完成後設定周期數與相關設定  plt.figure(figsize=(8,6))  plt.plot(history.epoch,history.history['loss'])  plt.title('loss')  plt.figure(figsize=(8,6))  plt.plot(history.epoch,history.history['acc'])  plt.title('acc')  #測試資料後的評估模型準確率  scores = model.evaluate(x\_test\_image, y\_test\_cat, verbose=2)  print("accuracy = {:2.2f}%".format(scores[1]\*100.0)) |   執行結果：     1. 使用卷積神經網路(CNN)進行辨識訓練：   (1)建立模型與資料結構：   |  | | --- | | # 導入相關套件  import os  os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  import numpy as np # Numpy：矩陣運算  import pandas as pd  from keras.utils import np\_utils  from keras.datasets import mnist # keras.datasets：載入MNIST資料集  from keras.models import Sequential # Keras：建立訓練模型  from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D  (x\_Train, y\_Train), (x\_Test, y\_Test) = mnist.load\_data()  # 影像特徵值轉換為4維矩陣  x\_Train4D = x\_Train.reshape(x\_Train.shape[0], 28, 28, 1).astype('float32')  x\_Test4D = x\_Test.reshape(x\_Test.shape[0], 28, 28, 1).astype('float32')  # 壓縮圖片顏色至0 ~ 1  x\_Train4D\_normalize = x\_Train4D / 255  x\_Test4D\_normalize = x\_Test4D / 255  #依分類數量將圖片標籤轉換格式的陣列  y\_TrainOne = np\_utils.to\_categorical(y\_Train)  y\_TestOne = np\_utils.to\_categorical(y\_Test)  model = Sequential()  # 新增卷積層  model.add(Conv2D(filters=16, kernel\_size=(5,5), padding='same', input\_shape=(28, 28,1), activation='relu'))  # 新增池化層  model.add(MaxPooling2D(pool\_size=(2,2)))  # 新增卷積層  model.add(Conv2D(filters=16, kernel\_size=(5,5), padding='same', activation='relu'))  # 新增池化層  model.add(MaxPooling2D(pool\_size=(2,2)))  # 防止過擬合  model.add(Dropout(0.25))  # 全連接層局部  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(10, activation='softmax'))  model.summary() |   執行結果：    (2)開始訓練：   |  | | --- | | # 定義定義損失函數、優化函數及成效衡量指標  model.compile(loss='categorical\_crossentropy', optimizer='adam',metrics=['acc'])  train\_history = model.fit(x=x\_Train4D\_normalize,y=y\_TrainOne, validation\_split=0.2,epochs=10, batch\_size=300, verbose=1) |   執行結果：    (3)訓練後結果：   |  | | --- | | def show\_train\_history(train\_history, train, validation):  plt.plot(train\_history.history[train])  plt.plot(train\_history.history[validation])  plt.title('Train History')  plt.ylabel(train)  plt.xlabel('Epoch')  plt.legend(['train', 'validation'], loc='upper left')  plt.show()  import matplotlib.pyplot as plt  show\_train\_history(train\_history,'acc','val\_acc')  show\_train\_history(train\_history, 'loss', 'val\_loss') |   執行結果：    (4)預測值及混淆矩陣：   |  | | --- | | loss, acc = model.evaluate(x\_Test4D\_normalize, y\_TestOne)  print("\nLoss: %.2f, Accuracy: %.2f%%" %(loss, acc\* 100))  import pandas as pd  prediction=model.predict\_classes(x\_Test4D\_normalize)  pd.crosstab(y\_Test,prediction,rownames=['label'], colnames=['predict']) |   執行結果：    (5)建立模型與資料結構(調整模式)：   |  | | --- | | (x\_Train , y\_Train),(x\_Test , y\_Test) = mnist.load\_data()# 呼叫 load\_data() 載入 MNIST 資料集  # 影像特徵值轉換為4維矩陣  x\_Train4D=x\_Train.reshape(x\_Train.shape[0],28,28,1).astype('float32')  x\_Test4D=x\_Test.reshape(x\_Test.shape[0],28,28,1).astype('float32')  # 壓縮圖片顏色至0 ~ 1  x\_Train4D\_normalize = x\_Train4D / 255  x\_Test4D\_normalize = x\_Test4D / 255  # 依分類數量將圖片標籤轉換格式的陣列  y\_TrainOneHot = np\_utils.to\_categorical(y\_Train)  y\_TestOneHot = np\_utils.to\_categorical(y\_Test)  model = Sequential()  # 新增卷積層  model.add(Conv2D(filters=16, kernel\_size=(5,5), padding='same', input\_shape=(28,28,1), activation='relu'))  # 新增池化層  model.add(MaxPooling2D(pool\_size=(2,2)))  # 新增卷積層  model.add(Conv2D( filters=36, kernel\_size=(5,5), padding='same', activation='relu'))  # 新增池化層  model.add(MaxPooling2D(pool\_size=(2,2)))  # 防止過擬合  model.add(Dropout(0.25))  # 全連接層局部  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(10, activation='softmax'))  model.summary() |   執行結果：    (6)再次開始訓練：   |  | | --- | | # 定義定義損失函數、優化函數及成效衡量指標  model.compile(loss='categorical\_crossentropy', optimizer='adam',metrics=['accuracy'])  # 開始訓練  train\_history=model.fit(x=x\_Train4D\_normalize, y=y\_TrainOneHot,validation\_split=0.2, epochs=10, batch\_size=300,verbose=2) |   執行結果：    (7).訓練後結果(準確率變化)：   |  | | --- | | def show\_train\_history(train\_history, train, validation):  plt.plot(train\_history.history[train])  plt.plot(train\_history.history[validation])  plt.title('Train History')  plt.ylabel('train')  plt.xlabel('Epoch')  plt.legend(['train', 'validation'], loc='center right')  plt.show()  import matplotlib.pyplot as plt  show\_train\_history(train\_history, 'acc','val\_acc')  show\_train\_history(train\_history, 'loss','val\_loss') |   執行結果：    (8)混淆矩陣(改變過程)：   |  | | --- | | #訓練後結果  loss, accuracy = model.evaluate(x\_Test4D\_normalize , y\_TestOneHot)  print( "\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy\* 100 ))  #混淆矩陣  import pandas as pd  prediction=model.predict\_classes(x\_Test4D\_normalize)  pd.crosstab(y\_Test,prediction,rownames=['label'],  colnames=['predict']) |   執行結果：     1. 使用循環神經網路(RNN)進行辨識訓練：   (1)建立模型與資料結構：   |  | | --- | | #導入相關套件  import os  os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  from keras.datasets import mnist # keras.datasets：載入MNIST資料集  from keras.models import Sequential # Keras：建立訓練模型  from keras.layers import Dense, Activation, Dropout  from keras.layers.recurrent import SimpleRNN, LSTM, GRU  from keras.utils import np\_utils  %matplotlib inline  # matplotlib.pyplot 將資料視覺化，可以圖表呈現結果  import numpy as np # Numpy：矩陣運算  import matplotlib.pyplot as plt  (x\_train , y\_train),(x\_test , y\_test) = mnist.load\_data() # 呼叫 load\_data() 載入 MNIST 資料集  nb\_classes = 10 # 類別的數目  img\_rows, img\_cols = 28, 28 # 圖片的長與寬  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  # 壓縮圖片顏色至0 ~ 1  x\_train /= 255  x\_test /= 255  # 依分類數量將圖片標籤轉換格式的陣列  y\_train = np\_utils.to\_categorical(y\_train, nb\_classes)  y\_test = np\_utils.to\_categorical(y\_test, nb\_classes)  nb\_units = 50 # 隱藏層節點數值  model = Sequential() # 建立簡單的線性執行模型  model.add(LSTM(nb\_units, input\_shape=(img\_rows, img\_cols))) # 二個維度  model.add(Dense(units=nb\_classes))  model.add(Activation('softmax'))  # 定義定義損失函數、優化函數及成效衡量指標  model.compile(loss='categorical\_crossentropy', optimizer='SGD', metrics=['accuracy'])  model.summary() |   執行結果：    (2)開始訓練：   |  | | --- | | epochs = 10  history = model.fit(x\_train, y\_train, epochs=epochs, batch\_size=128, verbose=1) |   執行結果：    (3)訓練後結果：   |  | | --- | | #訓練完成後設定周期數設定  plt.figure(figsize=(5,3))  plt.plot(history.epoch,history.history['loss'])  plt.title('loss')  plt.figure(figsize=(5,3))  plt.plot(history.epoch,history.history['acc'])  plt.title('acc');  #測試資料後的評估模型準確率  scores = model.evaluate(x\_test, y\_test, verbose=2)  print("accuracy = {:2.2f}%".format(scores[1]\*100.0)) |   執行結果：結果偏低需要再次提高數值後訓練    (4)建立模型與資料結構(調整模式)：   |  | | --- | | (x\_train , y\_train),(x\_test , y\_test) = mnist.load\_data() # 呼叫 load\_data() 載入 MNIST 資料集  nb\_classes = 10 # 類別的數目  img\_rows, img\_cols = 28, 28 # 圖片的長與寬  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  #壓縮圖片顏色至0 ~ 1  x\_train /= 255  x\_test /= 255  # 依分類數量將圖片標籤轉換格式的陣列  y\_train = np\_utils.to\_categorical(y\_train, nb\_classes)  y\_test = np\_utils.to\_categorical(y\_test, nb\_classes)  nb\_units = 128 # 調整隱藏層節點數值  model = Sequential() # 建立簡單的線性執行模型  model.add(LSTM(nb\_units, input\_shape=(img\_rows, img\_cols))) # 新增LSTM維度  model.add(Dense(units=nb\_classes))  model.add(Activation('softmax'))  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) #優化函數改成adam  model.summary() |     (5)再次開始訓練：   |  | | --- | | epochs = 10  history = model.fit(x\_train, y\_train, epochs=epochs, batch\_size=128, verbose=1) |   執行結果：    (6)訓練後結果(準確率提高)：   |  | | --- | | #訓練完成後設定周期數設定  plt.figure(figsize=(5,3))  plt.plot(history.epoch,history.history['loss'])  plt.title('loss')  plt.figure(figsize=(5,3))  plt.plot(history.epoch,history.history['acc'])  plt.title('acc');  #測試資料後的評估模型準確率  scores = model.evaluate(x\_test, y\_test, verbose=2)  print("accuracy = {:2.2f}%".format(scores[1]\*100.0)) |   執行結果： |
| 四、結論 |
| 每一種的類神經網路都會有不同的執行結果，MLP辨識出來的結果為91.22%，使用了三大層面就能夠很快地辨識出結果出來，運算結構較為初始化。CNN執行後會很就，必須要透過龐大的運算，對結構圖是分析的非常清楚，再由卷積層、池化層做出精緻的運算，就像是大張圖片濃縮成小張圖片，最後依據混淆矩陣來判斷出辨識出了的結果且數值為99.18%。RNN執行速度跟MLP差不多，需要調整隱藏層節點數值及LSTM維度數值後重新執行，結果就會提高至98.38%。所以最好的辨識為CNN。 |
| 五、參考 |
| 巨匠電腦python機器學習開發  巨匠電腦python深度學習開發  碁峰Python機器學習與深度學習特訓班 |