類神經網路 (ANN) 的分類問題

郭翊萱 711133115

March 30, 2023

類神經網路，又稱 ANN (Artificial Neuralwork)，其核心概念是人類希望能利用計算機模擬人類神經元收發信號的學習系統，透過設置變數來模擬神經元分布，對函式進行估計和模擬，讓資訊 (細胞) 經過變數的時候搭配權重 (Weight) 與偏差 (Bias)，利用閥值模擬細胞決定是否將訊息傳遞給下一個神經元。類神經網路 (ANN) 透過大量的人工神經元連結進行計算，並能自行透過學習更多資訊進而對模型進行改進，因此是一種具備學習功能的方法，它通過統計學的學習方法 (Learning Method) 進行最佳化用來解決各種各樣的實際問題，包含影像辨識與語音處理，概念也使類神經網路帶動了全球流行 AI 人工智能的風潮，並成為深度學習 (Deep Learning) 領域中常見的圖形辨識方法之一。在本節中將嘗試將類神經網路模型應用於經典的機器手臂方程式 (Robot Arm Equation) 探討連續型資料問題以及圖形識別 (Pattern Recognition) 探討類別型資料問題，並試驗如何調整參數與樣本數來使類神經網路模型 (ANN) 最佳化。

1 類神經網路 (ANN) 理論介紹

首先先了解類神經網路 (ANN) 的基礎理論。如上所說，類神經網路是一種模仿大腦運作的機器學習方式，透過大量的人工神經元連結進行計算，並能自行透過學習更多資訊進而對模型進行改進，其結構一般包含輸入層 (Input)、隱藏層 (Hidden Layer) 以及輸出層 (Ouput)，其中隱藏層的數量是可以進行改變的，其模型結構如圖 **??**[1] 。

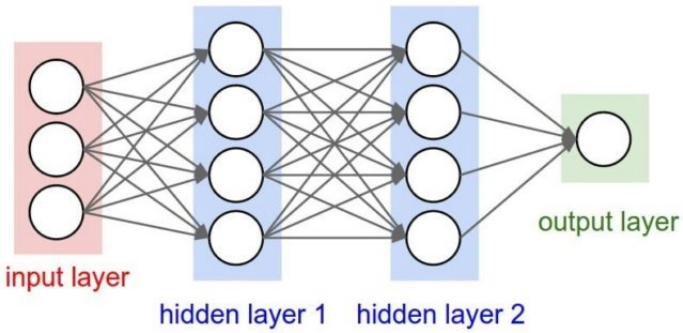

圖 1: 類神經網路 (ANN) 示意圖

[1] 圖片來源: https://case.ntu.edu.tw/blog/?p=26340

欲了解類神經網路的細部原理，首先須先了解甚麼是機器學習? 在機器學習中一般有三種常見的領域，包含迴歸 (Regression)、分類 (Classification) 以及結構學習 (Structured Learning)。迴歸是透過建立函數來求得數值，分類是利用函數來進行選擇題，結構學習則是利用函數來進行創造，產生具有結構的物件 (例如: 文章)。為了了解類神經網路理論，假設欲利用 youtube 後台所有數據來預測 2 月 26 日的觀看量[2]，則一般會遵循以下步驟:

- 建立包含未知參數的函數

  假設函數為 $y = b + wx_1$，其中 x_1: views on $2/25$、y : views on $2/26$，w (weight) 與 b (bias) 為模型之未知參數，其從已有資料中進行學習。

- 定義損失函數 (Loss Function)

  定義函數的輸入 (Input) 為未知參數 b、w，而損失函數 (Loss Function) 則用來表示在將 input 丟入函數後，此函數的預測效果有多好，其函數可寫作如式 (**??**)。

$$Loss = \frac{1}{N} \sum_{n} e_n \tag{1}$$

  其中 $e = (y - \hat{y})^2$。

- 利用不同的未知參數計算不同的損失函數並畫圖

- 最佳化模型 (Optimization)

  最佳化模型有很多方法，其中有一個常見的方法是梯度下降 (Gradient Descent)，嘗試求得合適的未知參數 w、b 使得損失函數的值最小，在此文不進行過多討論。

在此範例中，所有的模型調整全部建立在線性模型 (Linear Model) 之上進行調整，但線性模型往往太過簡單，就算選出了具有最小損失函數的模型，其依舊被初始設置的線性模型所限制 (model bias)。而如何設置更複雜的模型？Sigmoid Function 協助達成了此目的，其被表示如式 (**??**)。

$$sigmoid = \frac{1}{1 + e^{-x}} \tag{2}$$

故將新的預測模型寫作式 (**??**)。

[2]資料參考來源: 李宏毅機器學習 https://speech.ee.ntu.edu.tw/ hylee/ml/2022-spring.php

$$y = c\frac{1}{1+e^{-(b+w_1x_1)}} = c \times sigmoid(b + wx_1) \tag{3}$$

式 (**??**) 中的 w 控制函數坡度、b 控制左右移動，c 則可以改變函數的高度，若是具有兩個隱藏層，則利用此原理可以將類神經網路模型寫作式 (**??**)。

$$\hat{y} = b_k^2 + \sum_{i=1}^{q} w_{ki}^2 g(\sum_{j=1}^{p} w_{ij}^1 x_j + b_i^1) + b_k^2, 1 \leq k \leq r \tag{4}$$

其中函式 $g(\bullet)$ 為映射函數 (Sigmoid Function)，其數學式如式 (**??**)。

$$g(z) = c_1\frac{1-e^{-c_2z}}{1+e^{-c_2z}} \tag{5}$$

在式 (**??**) 中，q 為映射函數 (Sigmoid Function) 的數量、p 為輸入的特徵 (feature) 的數量、w_{ij}^1 與 b_i^1 代表第一個隱藏層的第 i 個神經元與第 j 個 input 的權重 (Weight) 跟偏誤 (Biases)，而 w_{ki}^2 與 b_k^2 則是第二層隱藏層的第 k 個神經元與前一隱藏層的第 i 個輸出的權重與偏誤，為了使模型最佳化，期望能得到使此模型的損失函數 (Loss Function) 最小的未知參數。

以上即是類神經網路的基本原理，但據此亦可知雖然其原理簡單有效，如果資料為高維資料時，如何求得使損失函數最小的未知參數也變得困難，如何定義需要多少隱藏層也成為挑戰，故下節將展示機器手臂 (Robot Arm Equation) 與圖形識別 (Pattern Equation) 在不同訓練過程下產生的不同測試結果。

2 機械手臂方程式 Robot Arm Equation

為了更了解類神經網路 (ANN) 的應用，利用機械手臂來作為連續型資料的範例[3]，如圖 **??**[4] 中展示了簡易的兩截式機械手臂。圖 **??** 中的機械手臂有兩截手臂，長度分別是 l_1 跟 l_2，其位置會隨著角度 θ_1 與 θ_2 改變而改變，使手臂頂端可以移動到目的地 (x, y) (Desired Location)，而為了了解手臂頂端可以抵達的範圍，必須計算出 θ_1 與 θ_2 的角度，即嘗試解逆運動方程式，如式 (**??**) 的 FKE 與式 (**??**) 的 LKE。

[3] 資料參考: https://ntpuccw.blog/python-in-learning/sml-lesson-4-artificial-neural-network-ann-deterministic-approach/

[4] 圖片來源: https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html

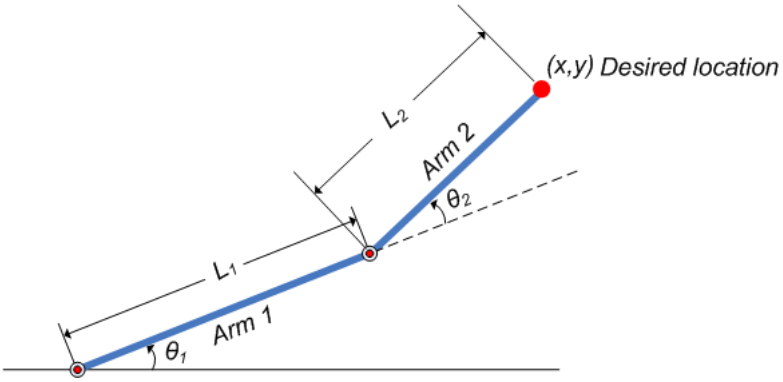

圖 2: 兩截式機械手臂示意圖

$$\begin{aligned} x &= l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) \\ y &= l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) \end{aligned} \tag{6}$$

$$\theta_2 = cos^{-1}(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2})\theta_1 = tan^{-1}(\frac{y}{x}) - tan^{-1}(\frac{l_2 sin(\theta_2)}{l_1 + l_2 cos(\theta_2)}) \tag{7}$$

直接解逆運動方程式看似是可行的，但是在機械手臂從兩截變成三截甚至以上時此方式是很困難的，因此使用類神經網路 (ANN) 來達成此效果。類神經網路會將手臂頂端能到達的位置 (x, y) 作為輸入值，而角度 θ_1 與 θ_2 作為輸出值來建立模型，最後透過訓練來找出輸入與輸出的關係，並計算當面對新的位置資料時，預測出的輸出值 (角度) 與真實值的均方根誤差 (RMSE)。

在使用類神經網路 (ANN) 建立模型時，一般會使用兩種套件，一種是 sklearn 套件中的 $neural_network.MLPRegressor$，另一種則是 $Neurolab.newff$，首先將展示套件 $neural_network.MLPRegressor$ 建立類神經網路的過程，並接著分別利用兩種套件嘗試改變樣本數大小、神經元個數與隱藏層個數，找出表現最好，即 RMSE 與 SSE 最小之類神經網路模型。

2.1 最佳化模型 Optimization

在此節中將一一詳述建立類神經網路模型 (ANN) 的過程，建立模型的過程一般分為四個步驟，首先先繪製機械手臂頂端經過路線的扇形圖，並從此扇形區域中抽取訓練資料，接著分別準備類神經網路的輸入 (位置 (x, y)) 與輸出值 (角度 (θ_1, θ_2))，並分別利用兩個套件 $neural_network.MLPRegressor$ 與 $Neurolab.newff$ 建立類神經網路模型，最後繪製真實值與估計值的散布圖並計算其均方根誤差 (RMSE)、

Rsquared、損失函數 (Loss function) 以及迭代次數 (Number of iteration)，此處利用套件 $neural_network.MLPRegressor$ 來進行過程之展示。

- 繪製扇形圖

  首先利用以下程式碼繪製扇形圖，展示機械手臂頂端會經過的區域，其圖形如下圖 **??** 之藍色區域。

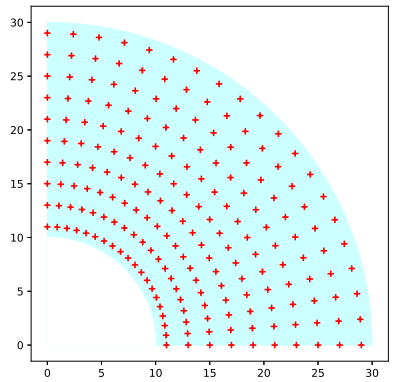

圖 3: 扇形圖

```
l1, l2 = 20, 10
f1 = lambda x: np.sqrt((l1 + l2)**2 - x **2)
f2 = lambda x: np.sqrt(l2 **2 - x **2)

plt.figure(figsize = (6, 6))
x = np.linspace(0, l1+l2, 200)
plt.fill_between(x, f1(x), 0, color = "#CCFFFF")
x = np.linspace(0, l2, 200)
plt.fill_between(x, f2(x), 0, color = "white")
```

- 準備輸入值與輸出值

  接著根據式 (**??**) 的 FKE 與式 (**??**) 的 LKE，分別設置類神經網路的輸入值 (位置) 與輸出值 (角度)，輸入值繪製如圖 **??** 中之紅點，其程式碼亦呈現如下，其中，我們會利用函數 $ravel()$ 將輸入值與輸出值的向量合併成矩陣。

```
theta2 = np.arccos((X.ravel()**2 + Y.ravel()**2 - l1**2 -
   l2**2)/(2*l1*l2))
theta1 = np.arctan(Y.ravel()/X.ravel()) - np.arctan(l2*np.
   sin(theta2)/(l1+l2*np.cos(theta2)))

InputX = np.c_[X.ravel(), Y.ravel()]
OutputY = np.c_[theta1, theta2]
```

- 設置參數並建立類神經網路模型

為了更了解類神經網路模型，利用上述建立的輸入值與輸出值觀察改變隱藏層數量，觀察不同隱藏層數量產生的不同均方根誤差 (RMSE)、Rsquared、損失函數 (Loss function) 以及迭代次數 (Number of iteration) 的值，其圖展示如圖 **??**，程式碼亦呈現如下。

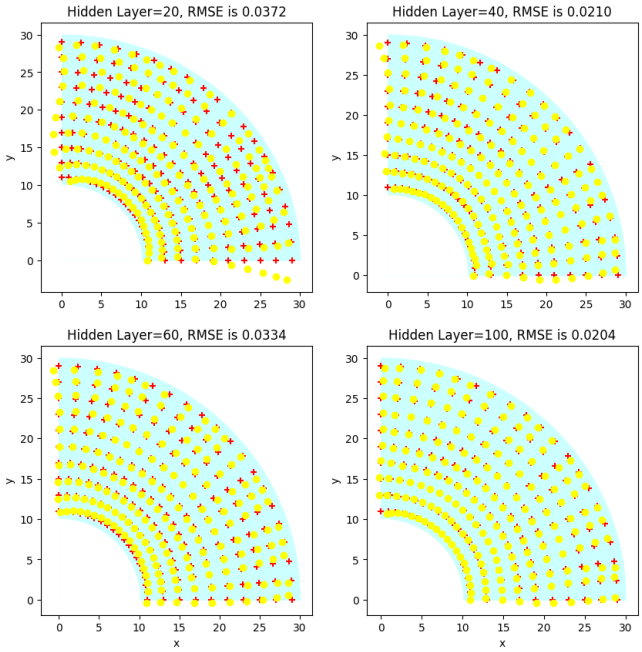

圖 4: 改變隱藏層之類神經網路模型

表 1: 不同隱藏層之類神經網路模型比較

	number=20	number=40	number=60	number=100
RMSE	0.0372	0.021	0.0334	0.0204
Loss Function	0.0007	0.0003	0.0006	0.0002
Number of Iteration	1502	4712	1662	3187
R-squared	0.996	0.998	0.997	0.999

圖 **??** 中紅色的點是真實值，黃色的點是類神經網路預測出來的值，而表 **??** 則展示了不同隱藏層的模型好壞，根據此表可知，隱藏層數設為 100 時類神經網路模型的表現會最好，RMSE、R-squared 與 Loss Function 都是最小的。在建立類神經網路模型時，設置參數 $max_iter = 8000$，即限制迴圈更新估計值的最大迴圈數為 8000，演算法則設置為 $solver = lbfgs$，即 LBFGS 演算法，另外也有 adam 與 sgd 兩種演算法可供選擇，最後，此模型之非線性函數 $activation$ 則選用了 logistic 演算法。

```
v = 100
hidden_layers = (v, )
solver = "lbfgs"
mlp_reg = MLPRegressor(max_iter = 8000, solver = solver,
hidden_layer_sizes = hidden_layers, verbose = False,
activation = "logistic", tol=1e-6, random_state = 0)
mlp_reg.fit(InputX, OutputY)  # Training

OutputY_hat = mlp_reg.predict(InputX) # Calculate fitted
   values
theta1_hat, theta2_hat = OutputY_hat[:,0], OutputY_hat
   [:,1]

x_hat = l1 * np.cos(theta1_hat) + l2 * np.cos(theta1_hat+
   theta2_hat)
y_hat = l1 * np.sin(theta1_hat) + l2 * np.sin(theta1_hat+
   theta2_hat)
```

以上即是在類神經網路在機械手臂問題上如何建立模型的過程，但以上升成訓練資料的方式其實並非最好的選擇，因為這些訓練資料並非亂數生成的，而該如何更好的在特定區域內公平抽取訓練資料呢？此處將藉助均勻分配來協助達到此目的，並展示利用均勻分配抽出訓練資料與測試資料後，如何分別利用套件 $neural_network.MLPRegressor$ 與 $Neurolab.newff$ 建立類神經網路模型、觀察訓練誤差的變化、並比較兩種套件改變隱藏層與神經元的情況下各自的 RMSE 與 SSE 之變化。

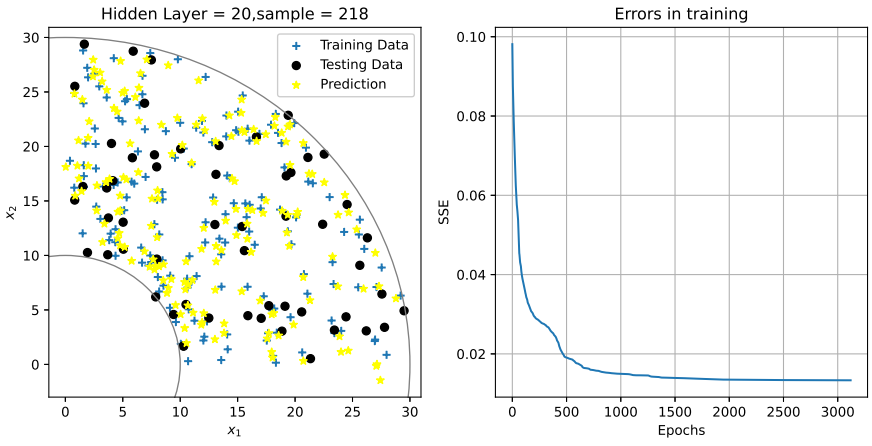

圖 5: 套件 $MLPRegressor$ 之類神經網路模型

```
def randsphere(center, radius, n_per_sphere):
  r = radius
  ndim = center.size
  x = np.random.normal(size=(n_per_sphere, ndim))
```

```
  ssq = np.sum(x ** 2, axis=1)
  fr = r * gammainc(ndim / 2, ssq / 2) ** (1 / ndim) / np.sqrt
     (ssq)
  frtiled = np.tile(fr.reshape(n_per_sphere, 1), (1, ndim))
  p = center + np.multiply(x, frtiled)
  return p
```

首先分別繪製利用兩套件所訓練的類神經網路模型的訓練結果圖，如圖 ?? 與圖 ??，分別展示了在隱藏層數為 20 的情況下各自的預測位置的成果，以及隨著 epoch 增加訓練誤差的變化。首先利用均勻分配生成一個圓內的隨機變數，其樣本數設 1000，再擷取位在第一象限的隨機變數作為資料，如上程式碼所示，其中函數 $randsphere$ 是用來產生高度空間球體內均勻散佈的點，其第一個參數 $center$ 代表中心點位置。$radius$ 代表半徑、n_per_sphere 則是樣本數；所以 randsphere(np.array([0, 0], 30, 1000) 代表在半徑 30 的圓內均勻產生 1000 個亂數。

而在圖 ?? 與圖 ?? 的範例中，最後得到樣本數分別為 209 個與 218 個的資料，並將其切成訓練資料與測試資料，其程式碼呈現如下，再計算 RMSE 以及 SSE 的值。此處將分別就不同的隱藏層數與樣本數所得到的 RMSE 與 SSE 進行比較，而圖 ?? 與圖 ?? 僅僅展示了隱藏層為 20 個的情況下繪製的圖形，其他參數所繪製出的圖則不予贅述，其結果分別展示於下表 ??、表 ?? 與表 ??。

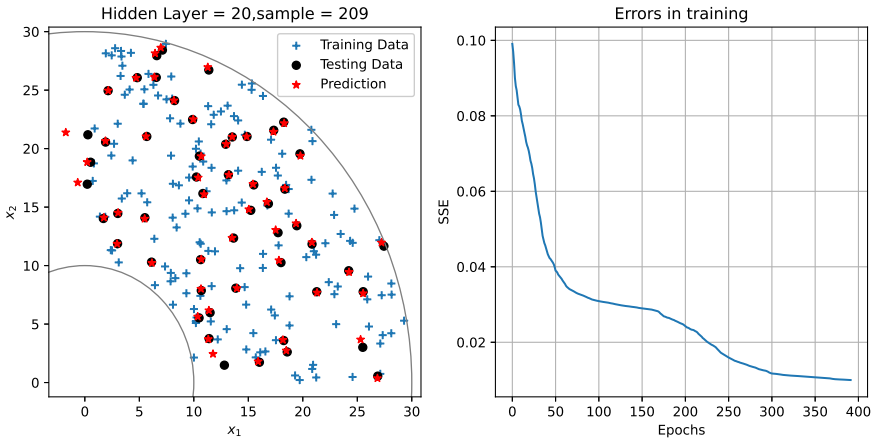

圖 6: 套件 $Neurolab.newff$ 之類神經網路模型

```
l1, l2 = 20, 10
n = 1000
# data distributed randomly in a circle
center = np.array([0, 0])
radius_in, radius_out =10, 30
p = randsphere(center, radius_out, n)
p = p[(p[:,0] > 0) & (p[:,1] > 0), :] # sample in the first
   quadrant
```

```
d = np.sum(p**2, axis=1)

p = p[d >= radius_in**2, :] # sample in the fan area
x1, x2 = p[:,0], p[:,1]
x_train, x_test, y_train, y_test = \
    train_test_split(x1, x2, test_size = 0.25)

theta2 = np.arccos((x_train**2 + y_train**2 - l1**2 - l2**2)
   /(2*l1*l2))
theta1 = np.arctan(y_train/x_train) - np.arctan(l2 * np.sin(
   theta2)/(l1 + l2 * np.cos(theta2)))
```

表 2: 機械手臂兩種套件改變隱藏層的類神經網路模型好壞比較

	MLPRegressor		*Neurolab.newff*	
	RMSE	SSE	RMSE	SSE
Hidden Layer = 20	0.061	0.01	0.008	0.031
Hidden Layer = 40	0.0296	0.01	0.007	0.044
Hidden Layer = 60	0.023	0.01	0.0077	0.032
Hidden Layer = 100	0.031	0.01	0.0078	0.279

根據表 **??** 觀察兩個套件在樣本數相同的情況下改變隱藏層數的 RMSE 與 SSE，假設一開始從均勻分配抽出的樣本數為 1000，則套件 $MLPRegressor$ 在不同隱藏層數下的 RMSE 都比套件 $Neurolab.newff$ 高，但 SSE 卻是 $neoralab.newff$ 都較低，另外，設置隱藏層數為 60 時兩個套件的 RMSE 與 SSE 都最小，因此判定在樣本數為 1000 時隱藏層數設 60 的模型表現最佳。接著根據表 **??** 觀察在隱藏層數都是 60 時改變樣本數兩個套件的模型表現，此時同樣也是一個套件在 RMSE 的表現較好，另一個套件在 SSE 的表現最好，因此兩種套件互有優劣，另外，在隱藏層設為 40 時樣本數為 3000 的模型表現最好。

表 3: 機械手臂兩種套件改變樣本數的類神經網路模型好壞比較

	MLPRegressor		*Neurolab.newff*	
	RMSE	SSE	RMSE	SSE
n = 1000	0.0342	0.01	0.0075	0.031
n = 2000	0.0259	0.01	0.005	0.006
n = 3000	0.0374	0.01	0.004	0.0038
n = 5000	0.04	0.01	0.003	0.01

表 4: 機械手臂兩種套件同時改變樣本數與隱藏層的類神經網路模型好壞比較

	MLPRegressor		*Neurolab.newff*	
	RMSE	SSE	RMSE	SSE
Hidden Layer = 20, n = 2000	0.0513	0.03	0.006	0.11
Hidden Layer = 20, n = 5000	0.0463	0.03	0.007	0.03
Hidden Layer = 60, n = 2000	0.0283	0.03	0.005	0.013
Hidden Layer = 60, n = 3000	0.03	0.03	0.004	0.035
Hidden Layer = 60, n = 5000	0.034	0.03	0.0035	0.016
Hidden Layer = 40, n = 2000	0.05	0.03	0.0054	0.015

最後，根據表 **??** 觀察同時改變樣本數與隱藏層的模型表現，套件 $Neoralab.newff$ 在 RMSE 與 SSE 的表現都比套件 $MLPRegressor$ 還要好，另外，設置隱藏層為 60 樣本數 2000 的模型是表現最佳的類神經網路模型。

3 圖形識別 Pattern Recognition

除了在連續型資料上的應用，類神經網路模型 (ANN) 也常常應用在圖形識別 (Pattern Recognition) 上，如圖 **??**，該筆資料集含有手寫的 0-9 的數字，由於手寫習慣都不同，也會有潦草跟工整的區別，因此如何成功判別該手寫數字是哪個數字也成為一大挑戰，而類神經網路即可解決此問題。

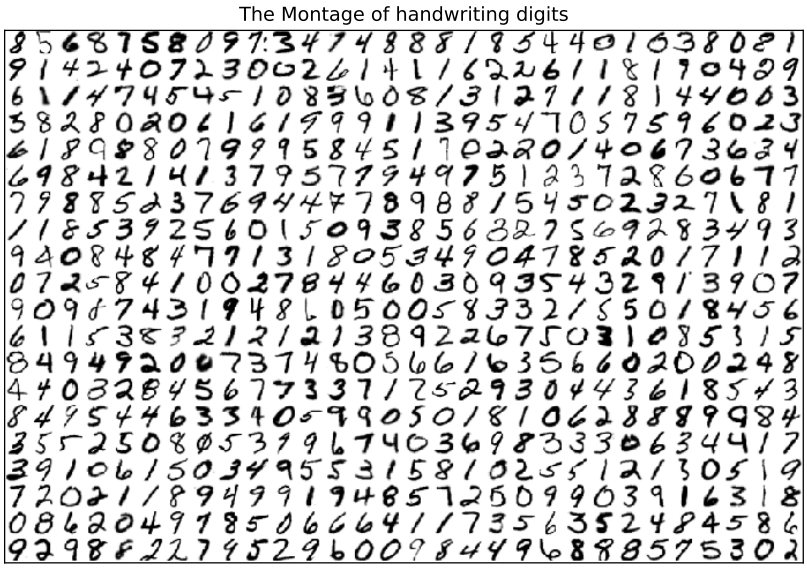

圖 7: 手寫數字範例圖

以下之程式碼呈現了如何利用 Python 讀入資料進行展示。此筆資料的原始資料為 $1000 * 784$ 的圖片影像，即共 1000 筆資料，每一張圖片的像素為 $28 * 28$，首先將每張圖片的像素從矩陣展開成 $1 * 784$ 的向量，接著取出 600 張圖片，並設置 $n = 20$ 且 $m = 30$ 來展示解析度為 $20 * 30$ 的圖片矩陣。如下之迴圈設置矩陣 M，將前述展開之 $1 * 784$ 向量重新 reshape 回 $28 * 28$ 的矩陣並一一放入矩陣 M 中。

```
n, m = 20, 30
sz = np.sqrt(X.shape[1]).astype(”int”)
M = np.zeros((m*sz, n*sz))
A = X[:m*n,:]

for i in range(m) :
    for j in range(n) :
        M[i*sz: (i+1)*sz, j*sz:(j+1)*sz] = A[i*n+j,:].reshape(
           sz, sz)
```

觀察過手寫數字資料後，接著利用類神經網路 (ANN) 來建立分類模型，此處使用 sklearn 套件中的 $MLPClassifier$ 來進行訓練，以下程式碼展現模型建立的方式，首先建立一個 $MLPClassifier()$ 的學習器，與 $MLPRegression$ 相同，其中 $solver$ 採用 adam 演算法，另外，有時候訓練模型時可能會因為在某個區域極小值卡住而導致結果不理想，因此會設置 $random_state = 0$ 給予類神經網路不同的初始值，並固定隨機亂數產生的位置，若希望產生的位置不同時，則不要給固定的數字。在完成訓練後，可以嘗試繪製混淆矩陣來觀察訓練模型的測試資料準確度，其範例如圖 **??**，其中採用了隱藏層數量為 20 來進行訓練。

```
v = 20
hidden_layers = (v,)
solver = ”adam” # default solver

clf = MLPClassifier(max_iter = 10000, solver = solver,
hidden_layer_sizes = hidden_layers, verbose = True,
activation = ”logistic”, tol = 1e-6, random_state = 0)
```

根據圖 **??** 可知，此類神經網路模型 (ANN) 的測試資料準確度為 86 %，其中手寫數字 4 最常被誤判成手寫數字 6、手寫數字 2 也最常被誤判成 6、數字 1 跟 5 較常被誤判成 8，數字 7 則較常被誤判成 9。

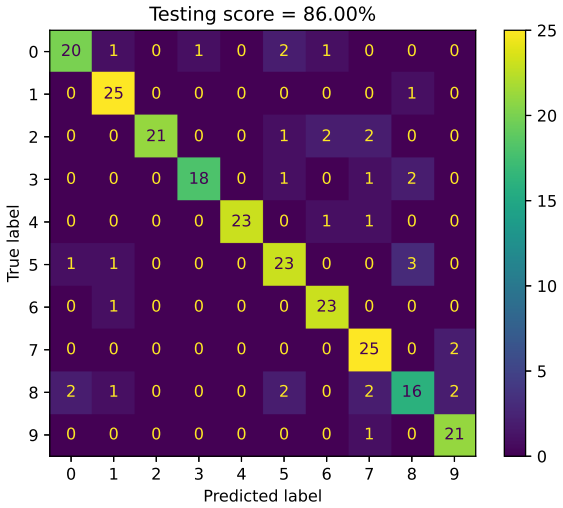

圖 8: 1000 筆資料集之類神經網路分類模型的測試資料混淆矩陣

接著，與機械手臂相同，同樣嘗試觀察不同隱藏層與不同樣本數對類神經網路模型的分類準確度的影響，在上述範例中使用的是資料集 Digits train 中的 1000 筆手寫資料集再進行分割，而在下面的試驗中，將同時利用 Digits train 10000 資料集中的 10000 筆手寫資料來建立類神經網路模型，並比較改變隱藏層數時準確度的變化，其結果如表 **??**。另外，此處亦展示 10000 筆手寫資料、20 個隱藏層的分類模型混淆矩陣，如圖 **??**。

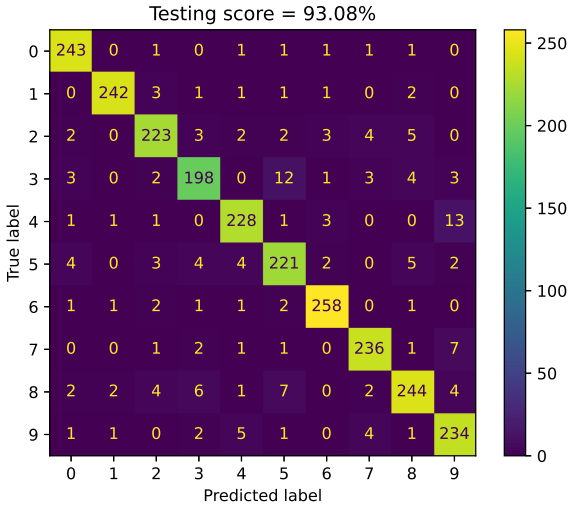

圖 9: 10000 筆類神經網路分類模型的測試資料混淆矩陣

根據圖 **??** 可以明顯發現，手寫數字 3 最常被分類到數字 5，而手寫數字 4 最常被分類到數字 9，數字 7 也很常被分類成數字 9，此結果與樣本數量為 1000 時的誤判不太相同，在 1000 筆資料時，數字 4 最常被分類到數字 6。

表 5: 兩筆資料集改變隱藏層之測試資料分類準確度

	n=1000	n=10000
Hidden Layer = 20	86%	93.08%
Hidden Layer = 40	91.2%	93.76%
Hidden Layer = 60	90.4%	95%
Hidden Layer = 100	88.8%	95.96%
Hidden Layer = 100	92.8%	96.2%

根據表 **??** 可以觀察到，在樣本數為 1000 筆跟 10000 筆時隱藏層為 100 的準確度都最高，另外，樣本數為 10000 筆時準確度似乎隨著隱藏層數增加而增加。

4 結論 Conclusion

在此節中探討了類神經網路 (ANN) 的理論以及其在連續型資料與類別型資料的應用，期望能幫助了解深度學習的基礎理論以及如何成功利用 Python 來對機械手臂以及圖像識別進行分析，並了解不同的參數設定對於模型效果的影響。