Programming Assignment 1: Recognition of Handwritten Digits by Multilayer Perceptrons

## network architectures & correct rate



## confusion matrix

1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 5 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 11 | 0 | 14 |
| 2 | 0 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 11 |
| 4 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 1 |
| 5 | 0 | 27 | 0 | 1 | 11 | 0 | 1 | 0 | 7 | 2 |
| 6 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 |
| 7 | 0 | 10 | 4 | 2 | 2 | 0 | 0 | 0 | 4 | 3 |
| 8 | 25 | 0 | 0 | 0 | 0 | 6 | 6 | 3 | 0 | 4 |
| 9 | 1 | 0 | 0 | 1 | 0 | 2 | 0 | 9 | 0 | 0 |

2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 2 | 1 | 1 | 3 | 0 | 7 |
| 2 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 4 |
| 4 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 17 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 8 | 0 |
| 6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 8 | 3 | 0 | 2 | 0 | 0 | 0 | 0 | 4 |
| 8 | 20 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 2 | 1 | 4 | 0 | 2 | 1 | 0 |

3.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 9 | 0 | 0 |
| 2 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 7 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 5 | 0 | 57 | 0 | 0 | 11 | 0 | 2 | 0 | 8 | 0 |
| 6 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 15 | 1 | 2 | 1 | 0 | 0 | 0 | 2 | 4 |
| 8 | 17 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 |
| 9 | 1 | 0 | 0 | 2 | 1 | 3 | 0 | 11 | 0 | 0 |

4.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 1 | 0 | 0 | 3 | 1 | 0 | 1 | 1 | 6 | 0 | 0 |
| 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 2 |
| 4 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 5 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 |
| 6 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 7 | 3 | 1 | 0 | 1 | 0 | 0 | 0 | 4 |
| 8 | 17 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 2 |
| 9 | 0 | 0 | 0 | 2 | 0 | 9 | 0 | 0 | 0 | 0 |

5

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 2 | 2 | 0 | 2 | 47 | 0 | 17 |
| 2 | 0 | 36 | 0 | 2 | 0 | 0 | 0 | 38 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 8 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 2 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 7 | 1 |
| 8 | 17 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 |
| 9 | 1 | 0 | 0 | 1 | 3 | 4 | 0 | 1 | 0 | 0 |

這個專案創建了擁有簡易功能的Multilayer Perceptrons，當變數nn被指派為類別NeuralNetMLP時，此時nn即可以簡易的方式建構出MLP網路架構。



nn有兩個參數可調整，epochs次數、learning rate數alpha。

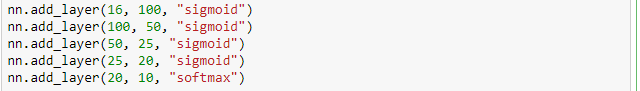
隨即我們呼叫nn裡的add\_layer方法，以建構神經網路。



上圖中是以一層50X25的隱藏層配合sigmoid激活函數，前一層為輸入層，後一層為輸出層(輸出層通常使用softmax居多)。

這邊我們可使用add\_layer這個方法，建構不同架構的神經網路。





設計此種方法能很便捷的變換不同層數或神經元數，以測驗最適合我們的網路架構。

建完網路架構後，我們呼叫fit方法，進入訓練階段。



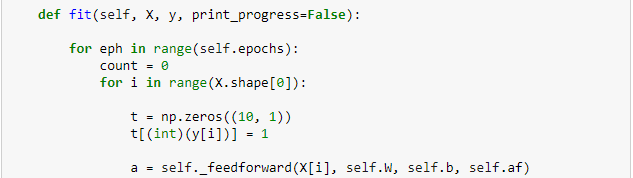
X\_train,y\_train是我們的訓練資料，因input資料為8組手寫數字(x,y)座標數值組成，我們先做正規畫以避免sigmoidy94在過大或過小的區間效果不彰。



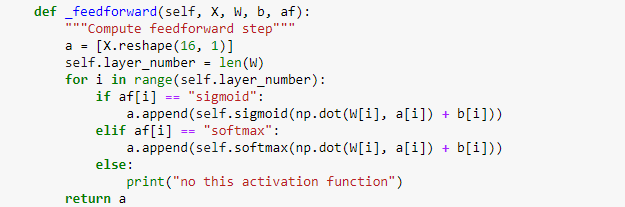
接著我們進入NeuralNetMLP類別詳細解說fit方法是如何訓練的。

fit方法裡對所有樣本作了epochs變數值的次數訓練，每一樣本將會一一進入神經網路裡，所以將會是epoch\*train\_data\_len次數的訓練。

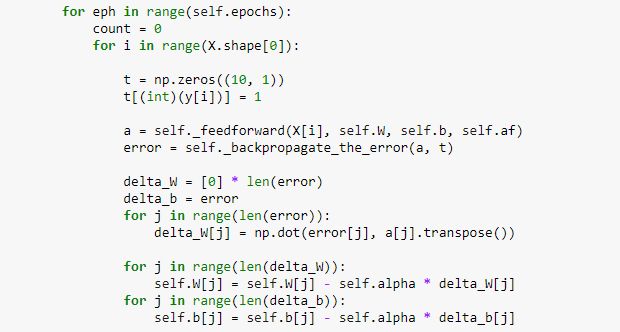
一個樣本的第一步是進入feedforward裡，經過每一次的和緊接著的激活函數計算，目的是算出每一層神經層的a，比如說輸入層+3個隱藏層+輸出層有5層，就將會有5個a被計算出來。



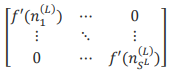
Feedforward裡每一層的W是乘與前一層算出來的a，而第一層的W則是乘與input的特徵值，到了最後一層時，a輸出會是10個數字，也就是和每一個數字互相對應著，最大的就是神經網路計算出最有可能的數字。



計算完每一層的a之後，進入backpropagation的步驟，如同其名，是由最後一層往回算，其目的是算出每一層的delta\_W, delta\_b，再以減掉原本的W, b，作為更新每一層的權重質，每一次的樣本輸入都會做一次的feedforward和backpropagation，全部樣本跑完就是一個epoch。



我們這邊先再詳細講解backpropagation裡的步驟，再回到fit。

Backpropagation裡，輸出層的error是最先算的，也就是算完最後一層，才往前一層算該層的error，回過來再說輸出層的error算法也是與其他層不同的，這邊我們先算也就是程式碼中的F也就是;

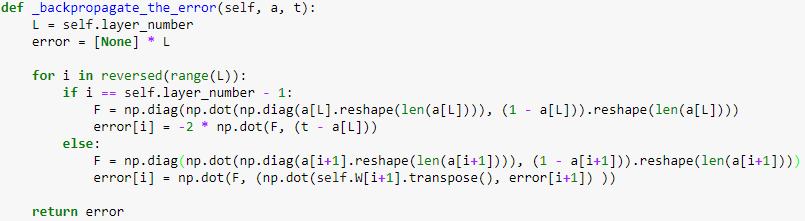
F是一個對角矩陣，這邊我們可以利用到numpy中的diag函數，它塞1維矩陣進去就會變成對角矩陣，也就是我們要的。

而裡面f’的意思是f’=f(1-f)，而f=a(n)，也就是我們之前feedforward計算的a，因此簡化成f’ = a(1-a)。

而它必須對a裡面的每一個質做計算，也就是會有10個f’。

np.dot , =

再丟進diag: ，就成了我們要的F。



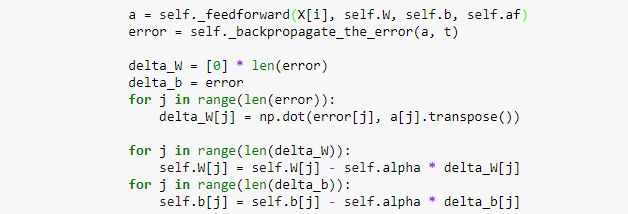
接著，經過one hot後的t減掉輸出層的a再乘上F，就成了我們要的輸出層的error。

隨即的其他層的error算法是將該層的下一層的error還有W相乘(記得轉置)後，再乘上該層算出的F即可得出，F如同前述。



一次的樣本輸入經過feedforward算出a後，再經backpropagation算出error，接下來的步驟回到nn裡的fit，將算出的所有error經過公式再算出每一層delta\_W和delta\_b，作以更新神經網路裡的W和b。





將所有的樣本跑過epochs次數後，得出的W和b即是最適合該訓練資料的權重值和偏差值。