圖形識別 作業2(b)

姓名：戴宏周
學號：0416302

1.流程圖與程式架構

程式架構是先建立Sigmoid和ReLu的Activation function和微分函式，接下來建立IJK網路，其中Activation function與微分函數帶入上述函式，以讓整個程式更加模組化，同時新增train rule選項，讓IJK網路模組有更高彈性。先從MNIST網頁下載所需檔案，透過讀取函式先將檔案讀入並以sinlge方式儲存，並針對網路格式進行transpose和組合。接下來呼叫自定義網路，進行訓練，最後建立error plot與decision region plot部份，並且將測試dataset送入網路取得預測結果，最後透過confusion matrix視覺化，印出結果，並且印出需求提到的train image和test image結果。最後將結果自動存檔。

網路部份，先初始化所需參數，如:weight,網路output，並設定必要參數如:eta,beta,itermax。訓練過程主要透過一個迴圈，當迭代次數滿足，或是error小於目標值則中止。迴圈中先進行一次forward computation，計算網路結果，並計算error，再進行一次back propagation，調整weight。針對不同的Activation function給入不同的gradient term。此外sigmoid與Relu均有帶入momentum term。

Nerual Networks toolbox則是使用feedforwardNet產生網路，並設定所有資料都用於training。此外training方式使用traingmm。

以下為所需公式：

(a) Sigmoid

Activation function : f(s) = $\frac{1}{1+e^{-\lambda s}}$

微分形式：f' (s) = $\lambda * f(s) * (1 - f(s))$

(b) ReLu

Activation function: $f(s) = max(s, 0)$

微分形式：$f'(s) = 1 \; if \; s > 0, \; else = 0$

(c) Adjust weight

對於IJK網路來說：

$$\Delta W_{ji}(t) = -\eta \frac{\partial E}{\partial W_{ji}} + \beta \Delta W_{ji}(t-1) = \eta \left(\sum_k (d_k - o_k) f'_k(S_k) W_{kj} \right) f'_j(S_j) O_i + \beta \Delta W_{ji}(t-1) \text{ for all j,i}$$

$$\Delta W_{kj}(t) = -\eta \frac{\partial E}{\partial W_{kj}} + \beta \Delta W_{kj}(t-1) = \eta (d_k - O_k) f'_k(S_k) O_j + \beta \Delta W_{kj}(t-1) \text{ for all k, j}$$

若有多層則按照相同邏輯擴增。

(d) weight initialization (Xavier initialization)

初始wieght 為normal distribution，其中平均為0，

標準差為 $\sqrt{\frac{2}{Number\ of\ input + Number\ of\ Output}}$

(e)Error計算

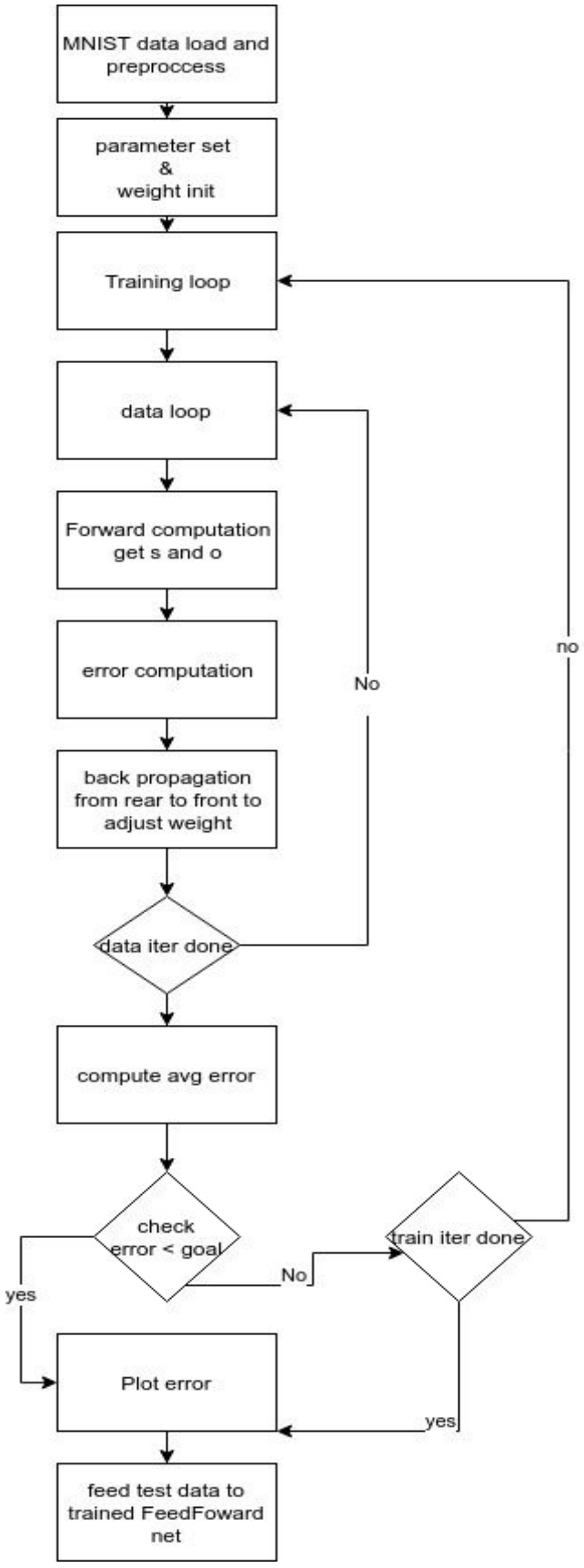
error為desire output(dk)向量與network ouput(ok)向量相減值，並將所有誤差加總。matlab表示如下：

$$error = sum(abs(dk(k) - ok(k))$$

error 平均為:

$$error\_average = \frac{sum\ of\ all\ error}{Number\ of\ input\ data}$$

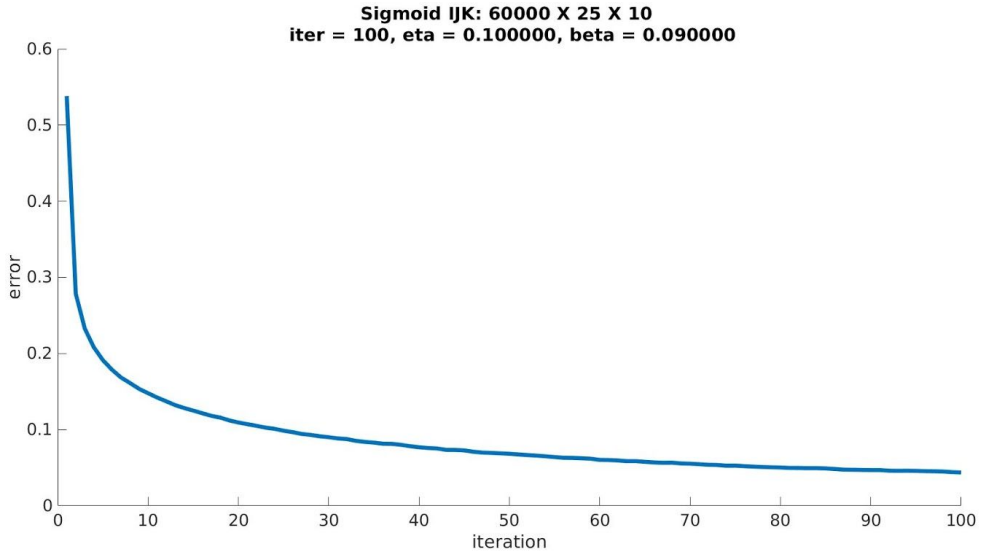
以下為流程圖：



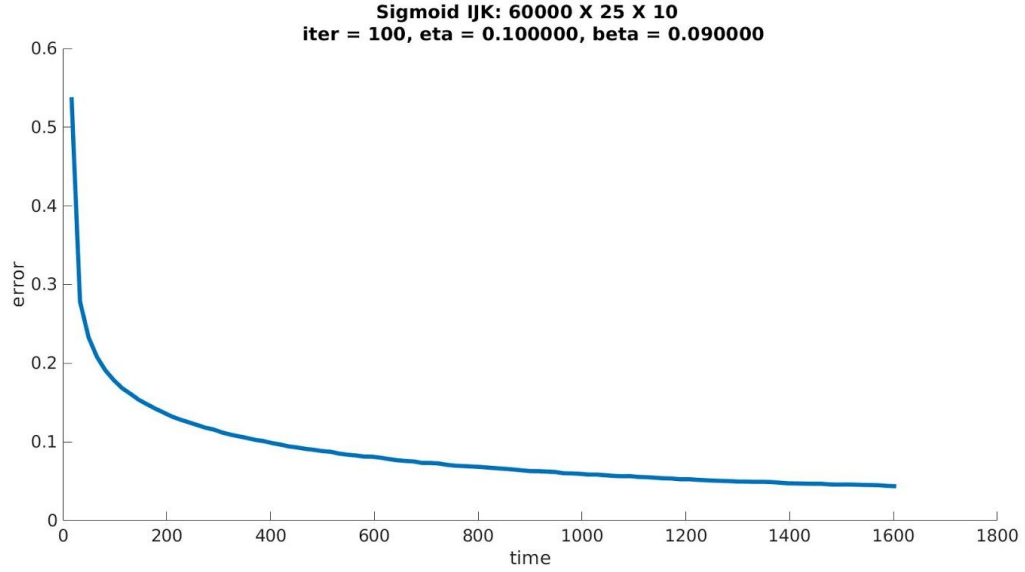
2. MNIST datasets

A. Sigmoid

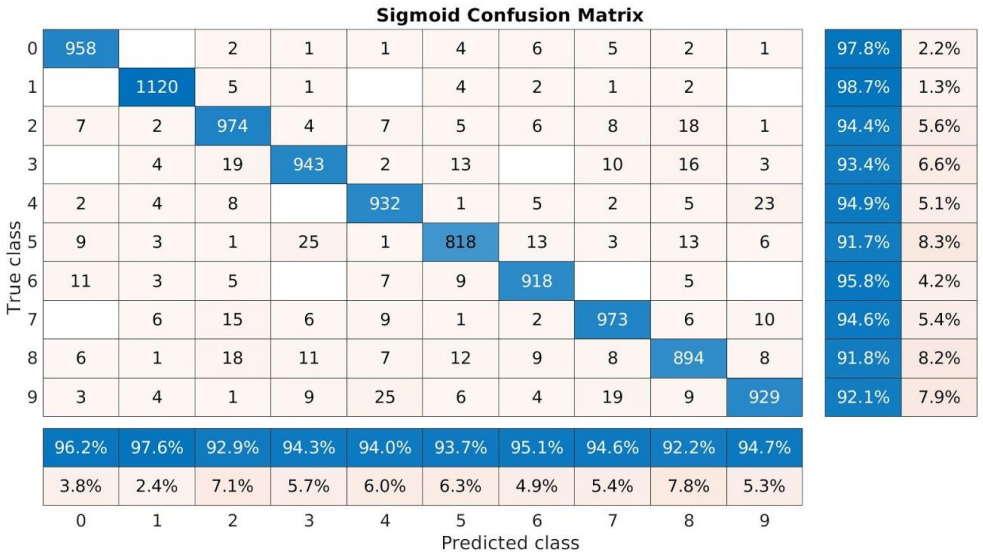
Sigmoid使用上一個作業的IJK網路進行訓練，所有資料都經過random shuffle。僅一層hidden layer，neuron為25，輸入為784(28*28)個輸入，輸出為10(將數字展開成長度為10的陣列，當對應0~9，例如:0則第一元素為1，為5則第六元素為1其餘元素為0)。iteration 次數為100，運算1600秒。learning rate(eta)為0.1，beta(momentum term)為0.09。準確度可94.59%。以下為error與confusion matrix。60000代表有60000筆訓練資料。



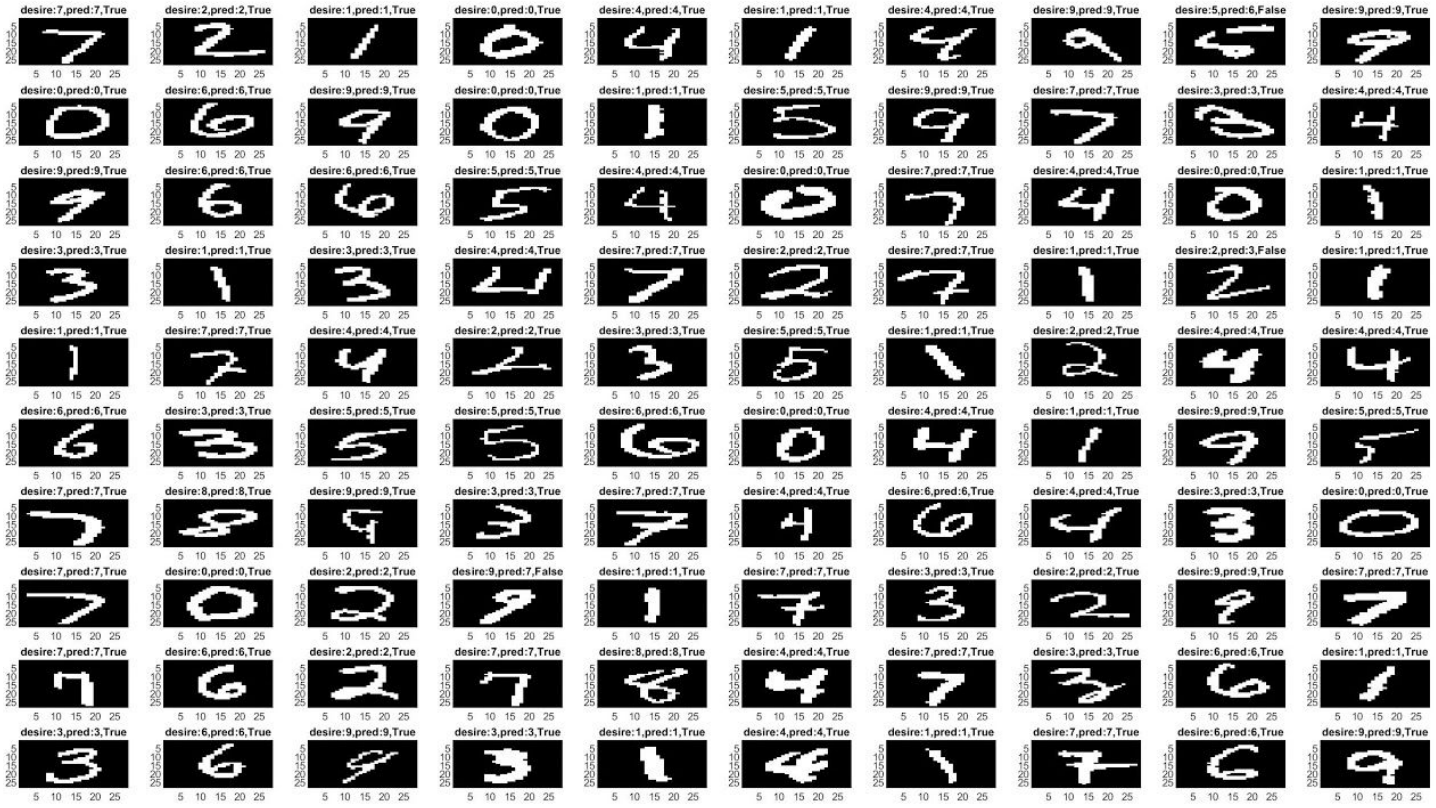
上圖為iteration與error 平均之關係。



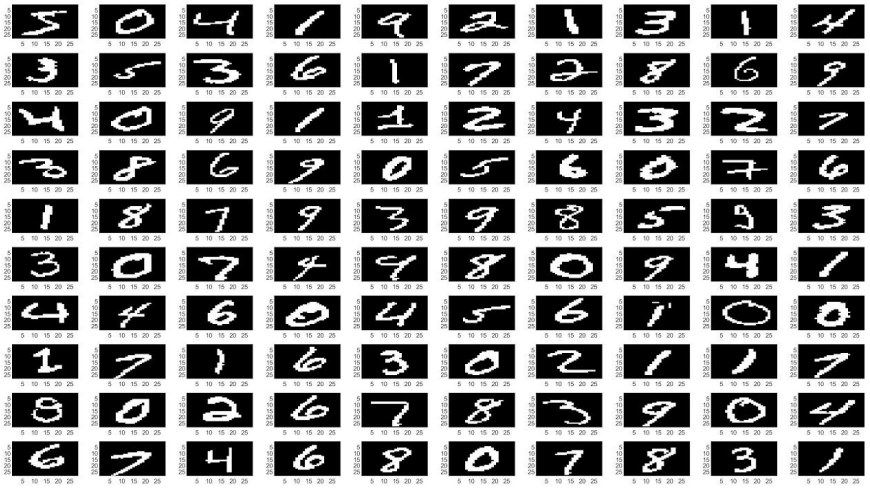
上圖為運行時間與error平均之關係。



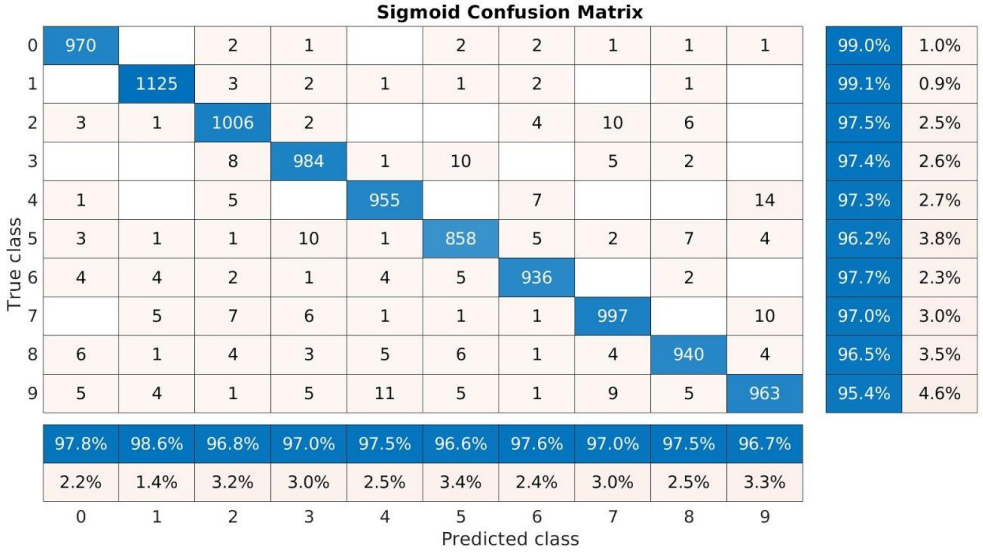
上圖為test資料的confusion matrix，以及各個class準確度。



上圖為測試資料前100張圖的label與預測結果，以及是否正確。



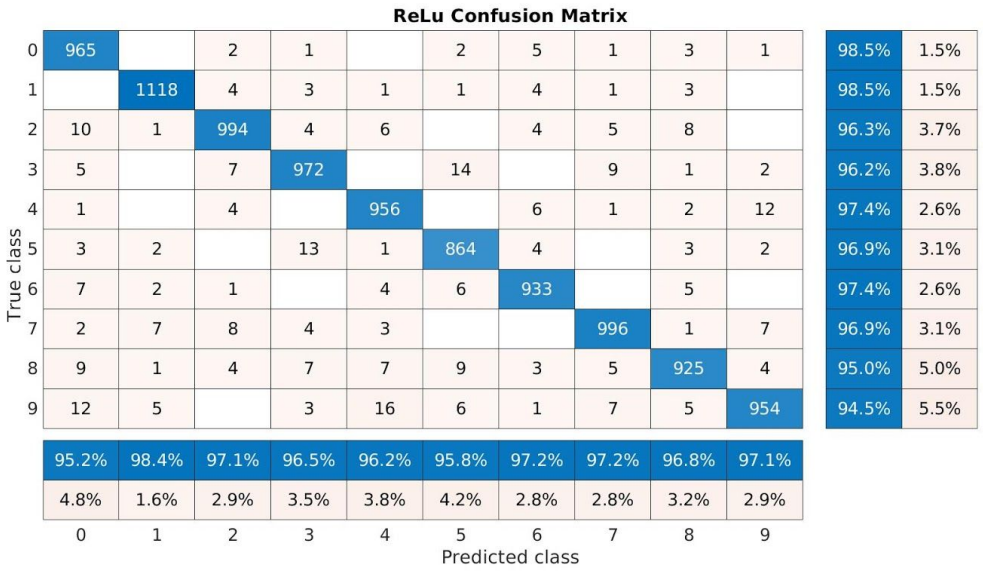
上圖為訓練資料前100筆。
若是將hidden layer neuron改為100，eta和beta保持0.1與0.09，且同樣訓練100個iteration，則準確度會上升為97.34%，confusion matrix如下圖。



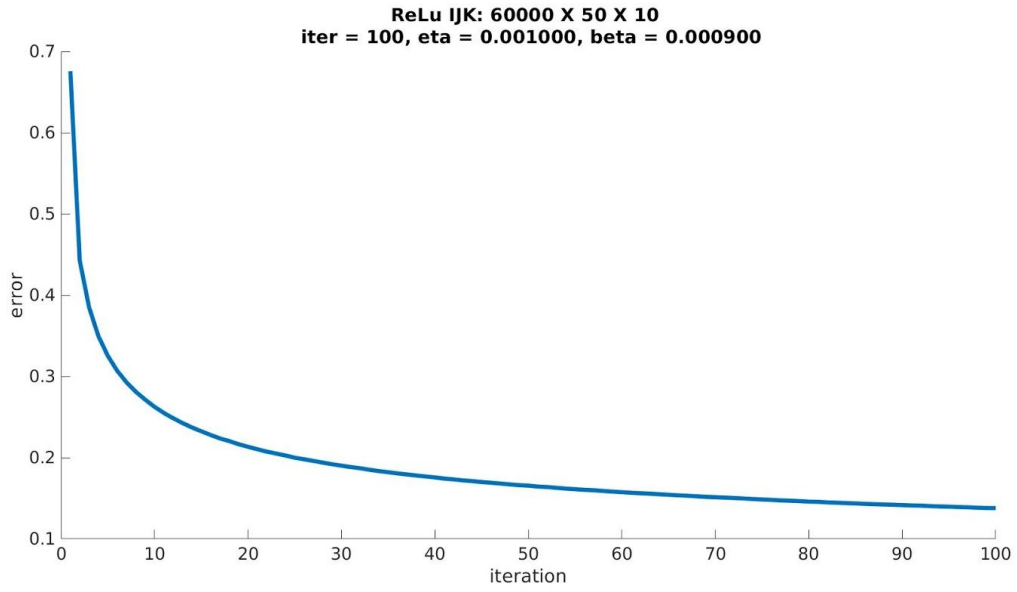
	0	1	2	3	4	5	6	7	8	9	
0	970		2	1		2	2	1	1	1	99.0% 1.0%
1		1125	3	2	1	1	2		1		99.1% 0.9%
2	3	1	1006	2			4	10	6		97.5% 2.5%
3			8	984	1	10		5	2		97.4% 2.6%
4	1		5		955		7			14	97.3% 2.7%
5	3	1	1	10	1	858	5	2	7	4	96.2% 3.8%
6	4	4	2	1	4	5	936		2		97.7% 2.3%
7		5	7	6	1	1	1	997		10	97.0% 3.0%
8	6	1	4	3	5	6	1	4	940	4	96.5% 3.5%
9	5	4	1	5	11	5	1	9	5	963	95.4% 4.6%
	97.8%	98.6%	96.8%	97.0%	97.5%	96.6%	97.6%	97.0%	97.5%	96.7%	
	2.2%	1.4%	3.2%	3.0%	2.5%	3.4%	2.4%	3.0%	2.5%	3.3%	

B. ReLu

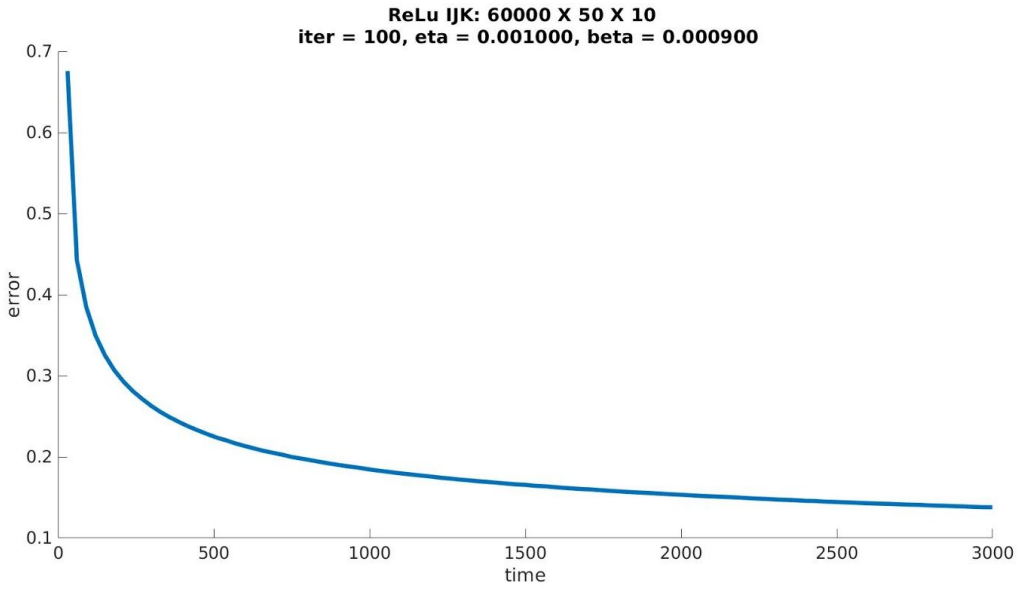
一樣使用IJK網路，不過hidden layer使用100個neuron。iteration為100次，eta為0.001，beta為0.0009。以下是結果與圖片。準確度為95.71%。



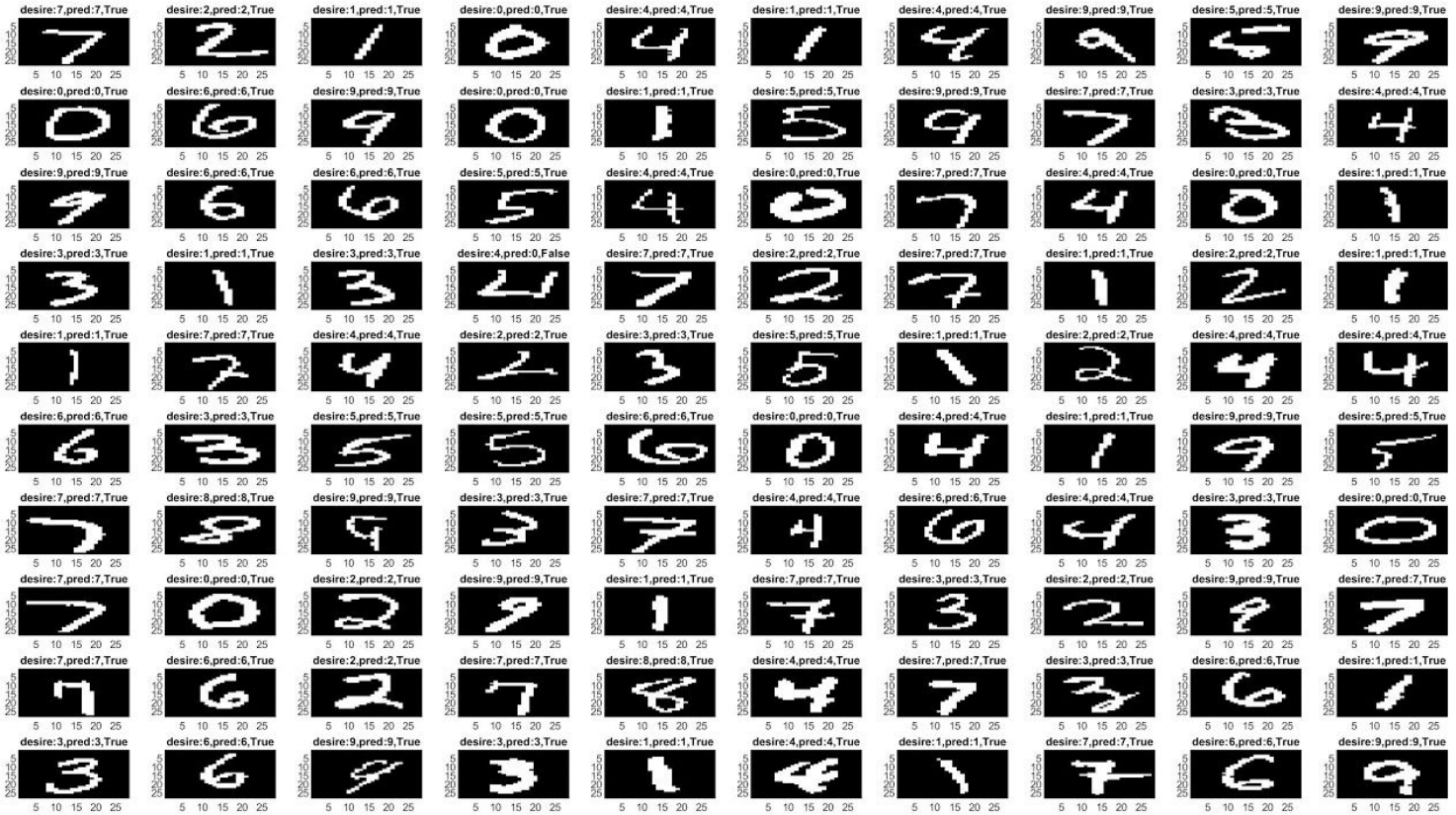
上圖為confusion matrix。



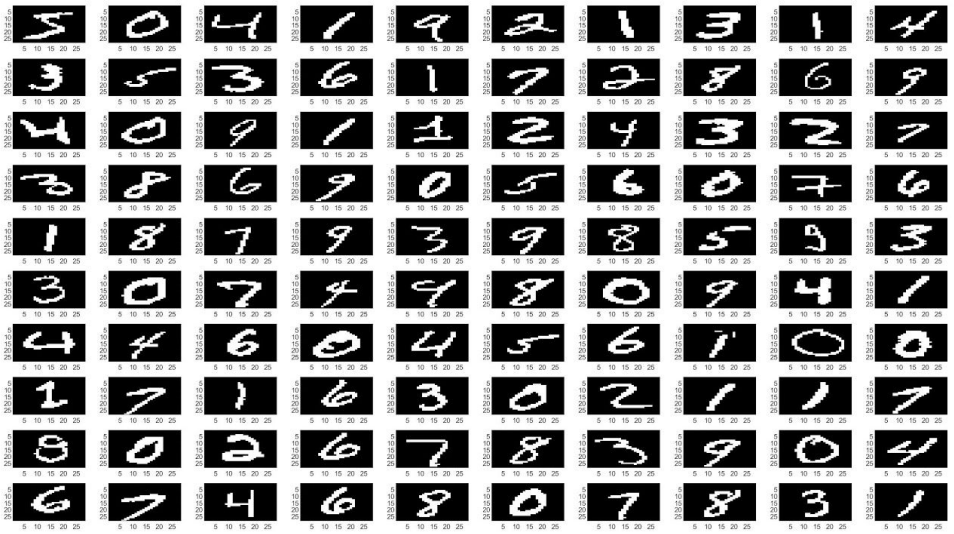
上圖為iteration與error平均作圖。



上圖為運行時間與error平均作圖。



上圖為test case前100筆之label與預測結果，以及是否正確。

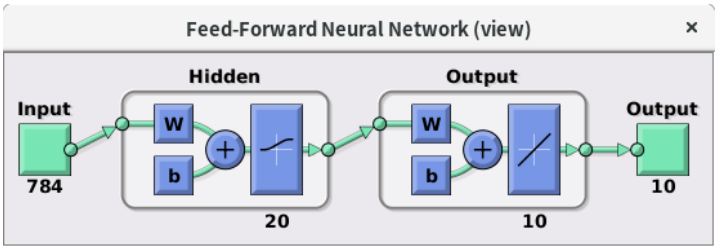


上圖為前100筆訓練資料。

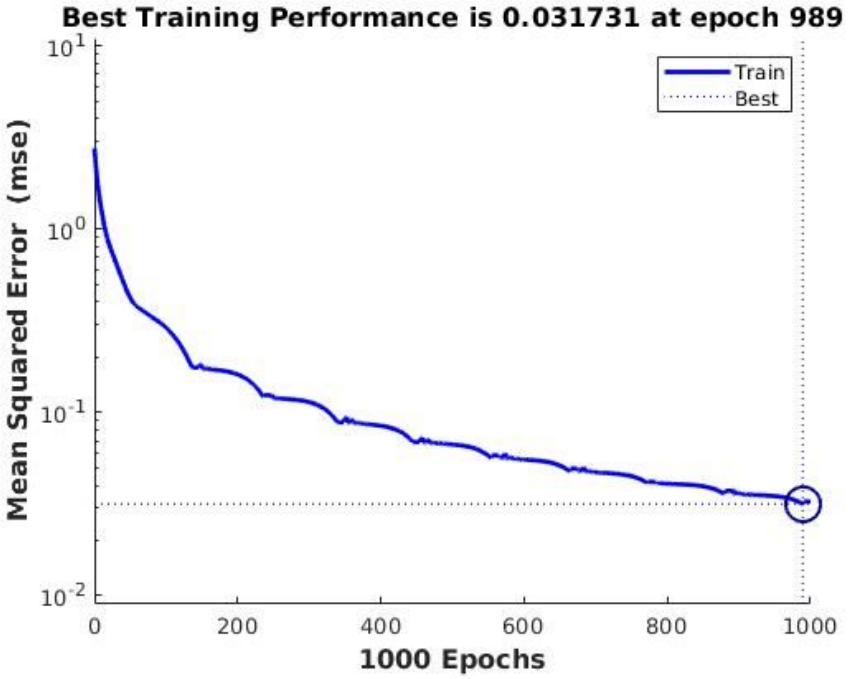
C. Nerual Network Tool box

(1)選取 1000 個 examples 做 training，1000 個做 testing。

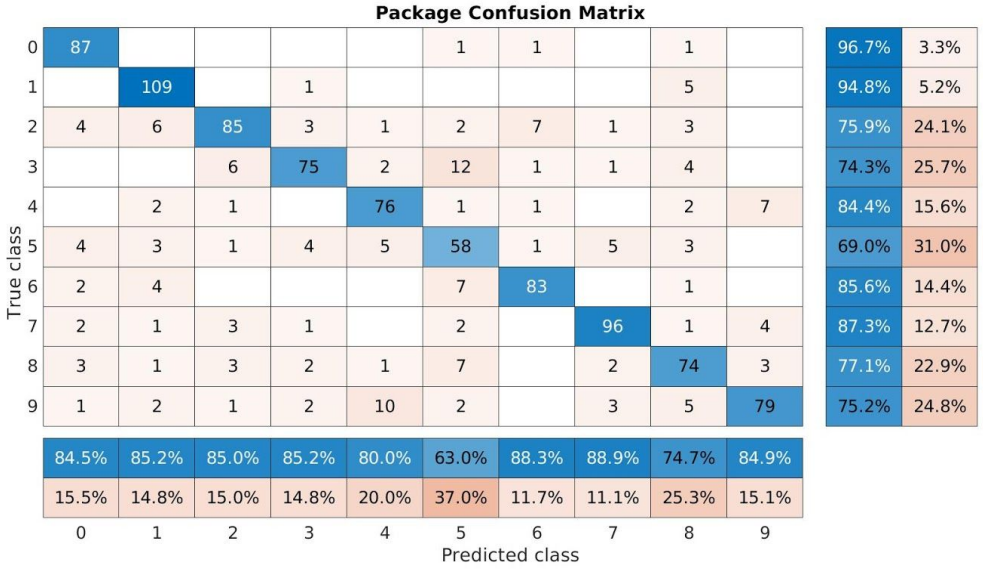
此部份採用FeedFowardNet。僅使用一層hidden layer，neuron數為20，learning rate為0.0001，mc(momentum)設為0.9，training 方式為 'traingdx'，epoch為1000，將net.performParam.normalization設為'standard'，hidden layer的learning rule為'logsig'，準確度為84.8%。



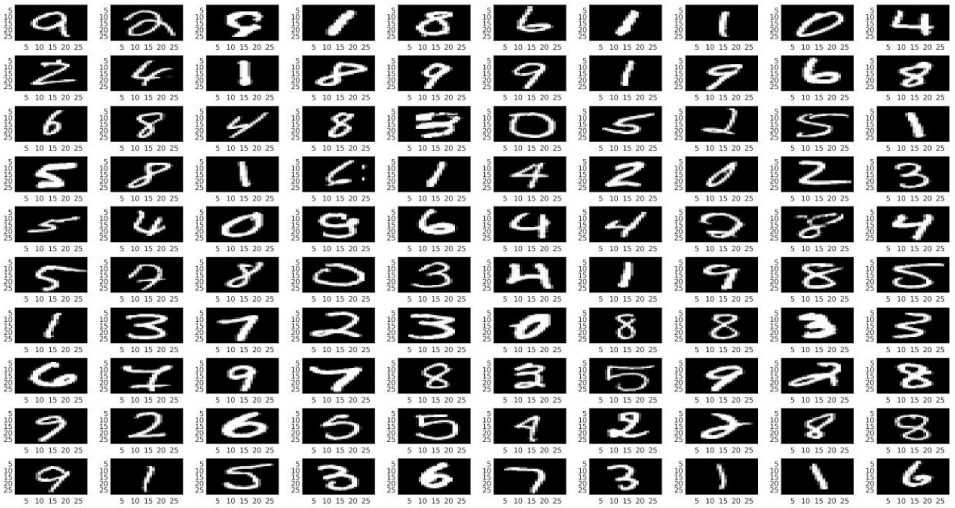
網路架構如上圖。



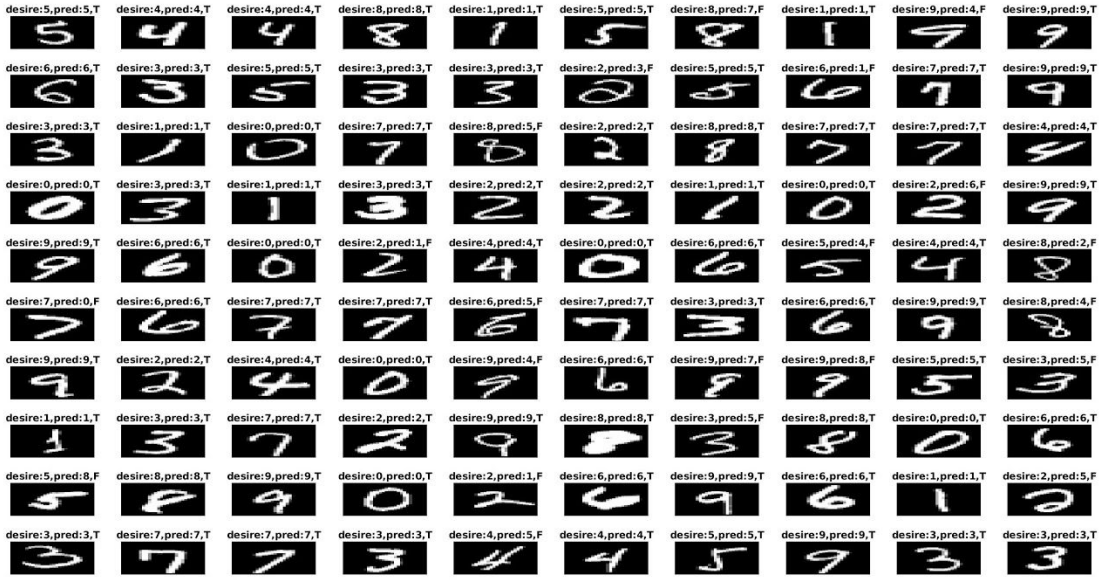
上圖為mean squared error與epochs的關係圖。



上圖為confusion matrix。



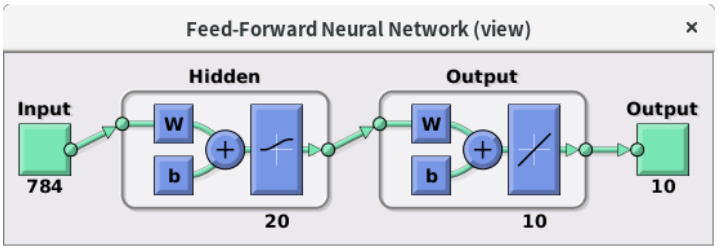
上圖為train data前100項。



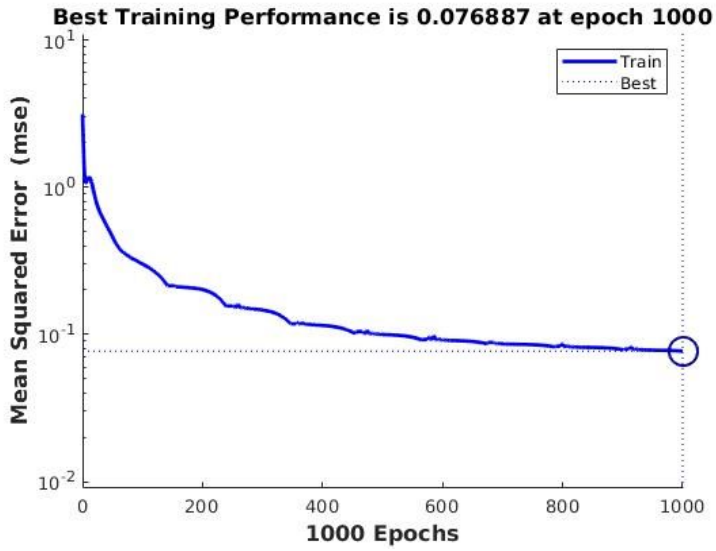
上圖為test data前100筆預測於實際結果。

(2)全部的 60000 個 examples 做 training，10000 個做 testing。

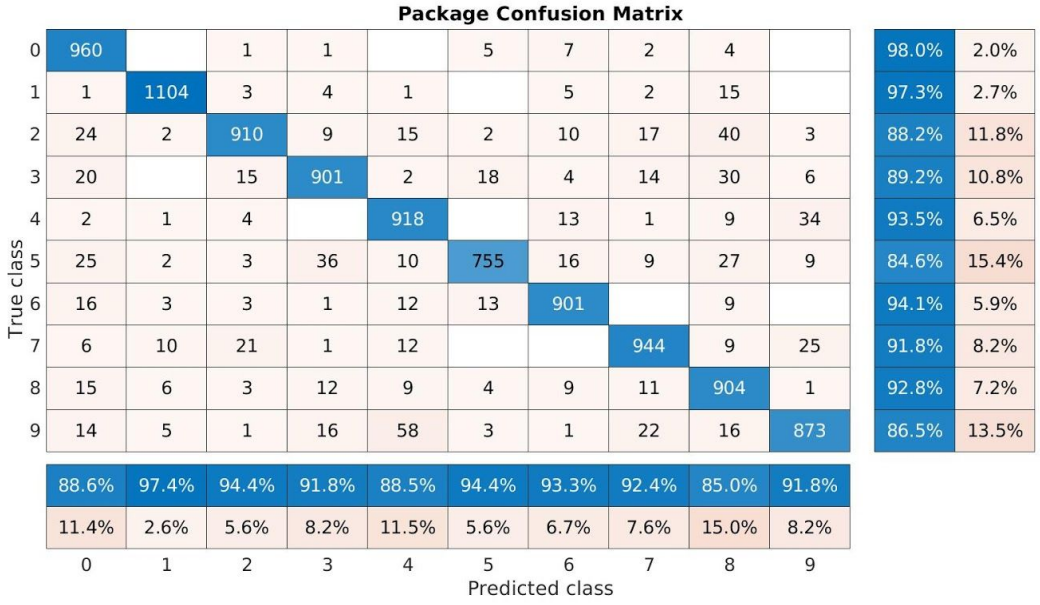
準確度為91.70%。

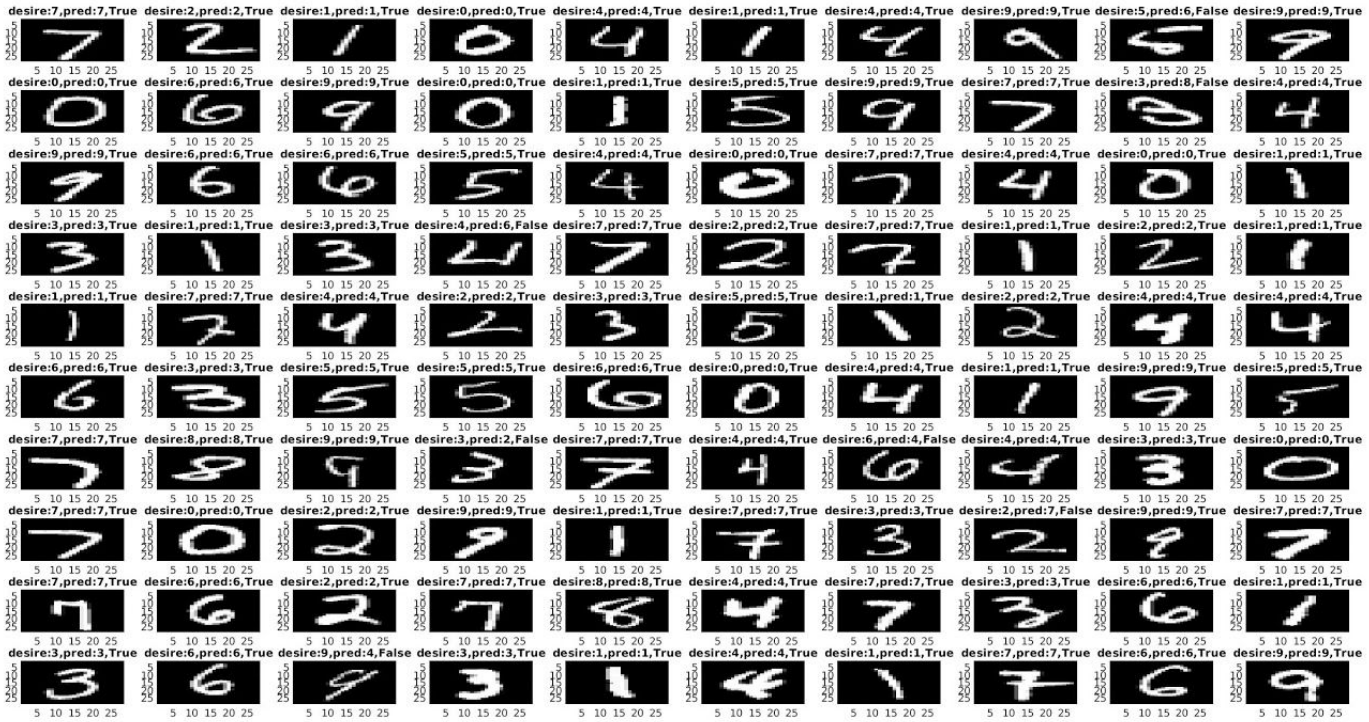


網路架構如上圖。

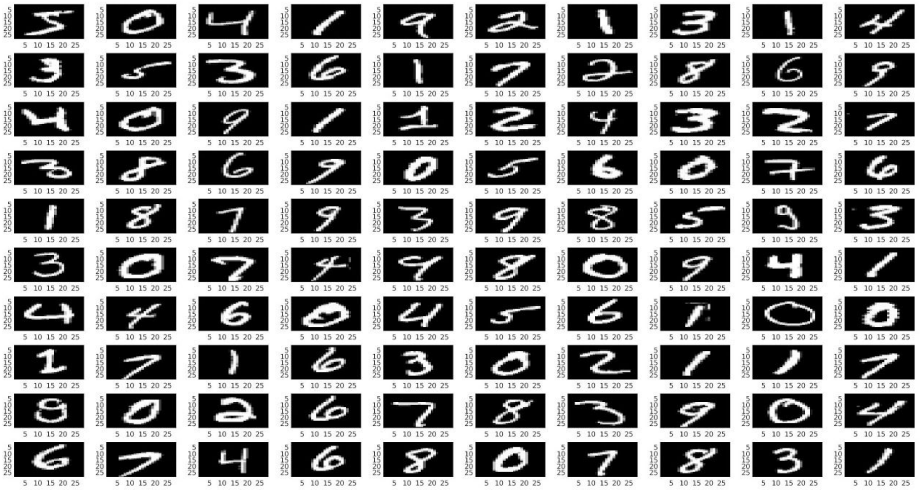


上圖為Mean square error與Epochs的關係。下圖為confusion matrix。





上圖為test case前100項之label與預測結果。



上圖為train data前100張圖片。

3.結果討論

此次作業應用常見的MNIST手寫資料集，採用上次作業的IJK網路。hidden layer的neuron決定主要是先估計一個較大的數目如100，最後看情況慢慢向下減少neuron數目。由於此次是分類圖片，因此比較難推算如何分割資料，主要是先推估後調整。

有趣的是，自製的sigmoid以及ReLu表現均比Neural Network ToolBox的FeedFowardNet好，我其實有嘗試過新增Neuron數，在上面的Sigmoid中，將Neuron數從25提升到100個，準確度會從94.59%提昇到97.34%，ReLu部份也有一樣特性，即使learning rate(eta)和momentum term(beta)均不變。將Sigmoid改為ReLU一樣需要調整learning rate至很低，因為ReLu比較敏感，若是用很大的learning rate很容易失敗。但是Neural Network Toolbox的表現卻不是這樣，當Neuron數提昇結果反而變得不好。我覺得這可能是因為使用traingdx訓練方式，此方式會導致learning rate在內部動態變化，此外，我在調整FeedFowardNet時遇到一些阻礙，原因是她內部的Normalizaiton和regularization會變動output(Y label)結果，因此我將其改用standard（會將值正規化至(-1,1)），才讓整個網路成功運作。而Neural Network Tool box部份調整過很多設定，包括訓練方式等，最後才成功讓結果接近我自己寫的MLP，我想這部份還有很多功能和設定有待我探索與學習。

而train data數目對網路訓練影響的結果也是很顯著，同樣的Neural Network toolbox程式，減少了train data，準確度明顯下降至84.8%。可見數據對train的重要性。不過此次將訓練資料大幅縮小至1000張，跟原先相比訓練速度快很多，所以若是缺乏時間可以先減少資料集。

4.程式碼

程式碼部份見e3電子檔。檔案說明如下：

1. MLP_Sigmoid.m：為使用Sigmoid Activation的程式。
2. MLP_ReLu.m：為使用ReLu Activation的程式。
3. MLP_Package.m：為使用Neural Network ToolBox feedfowardnet的程式。
4. MLP_Package_sample1000.m:為使用Neural Network ToolBox feedfowardnet的程式。但是資料量僅1000筆。
5. train_IJK_net.m：為MLP_Sigmoid.m與MLP_ReLu.m所呼叫之IJK網路function。
6. FeedFoward_IJK.m：為MLP_Sigmoid.m與MLP_ReLu.m所呼叫之IJK網路用於FeedFoward之function，用於獲取網路驗證結果。
7. Sigmoid.m：Sigmoid Activation function。
8. deSigmoid.m：Sigmoid function的微分。
9. ReLu.m：ReLu Activation。
10. deReLu.m：ReLu的微分。
11. Get_MNIST.m：獲取MNIST圖片之讀取function。
12. Get_MNISTLABEL.m：獲取MNIST label之讀取function。