

# 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single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

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

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

(b) ReLu

$$\text{Activation function: } f(s) = \max(s, 0)$$

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

(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weight 為normal distribution，其中平均為0，

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

(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。

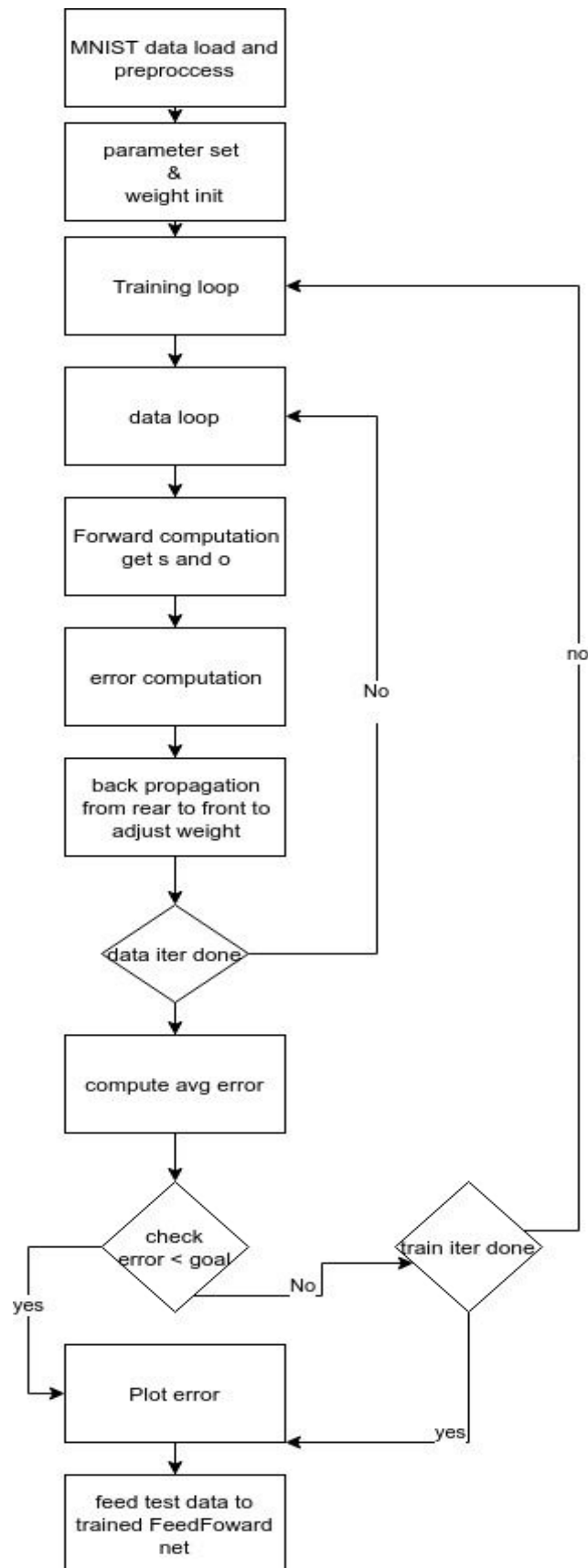
matlab表示如下：

$$\text{error} = \text{sum}(\text{abs}(dk(k) - ok(k)))$$

error 平均為：

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

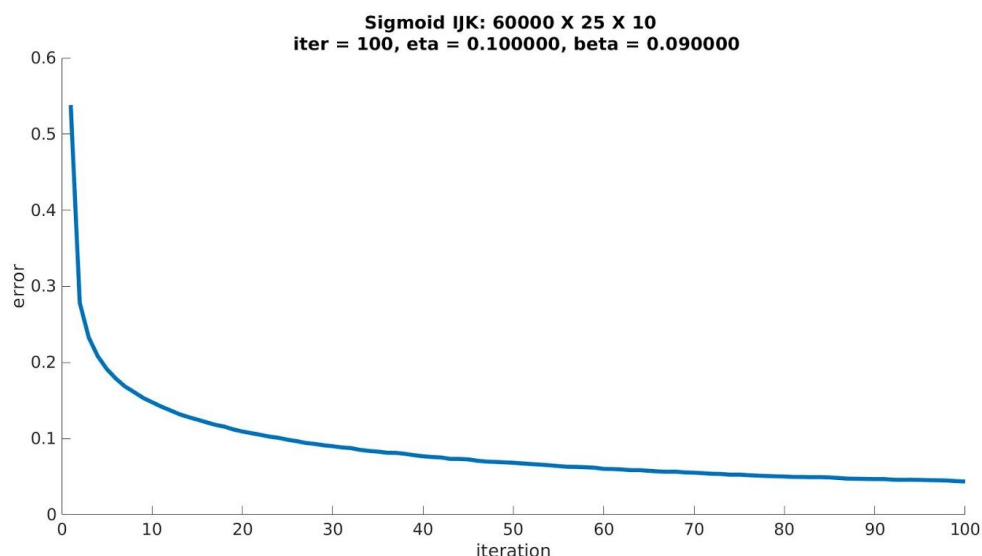
以下為流程圖：



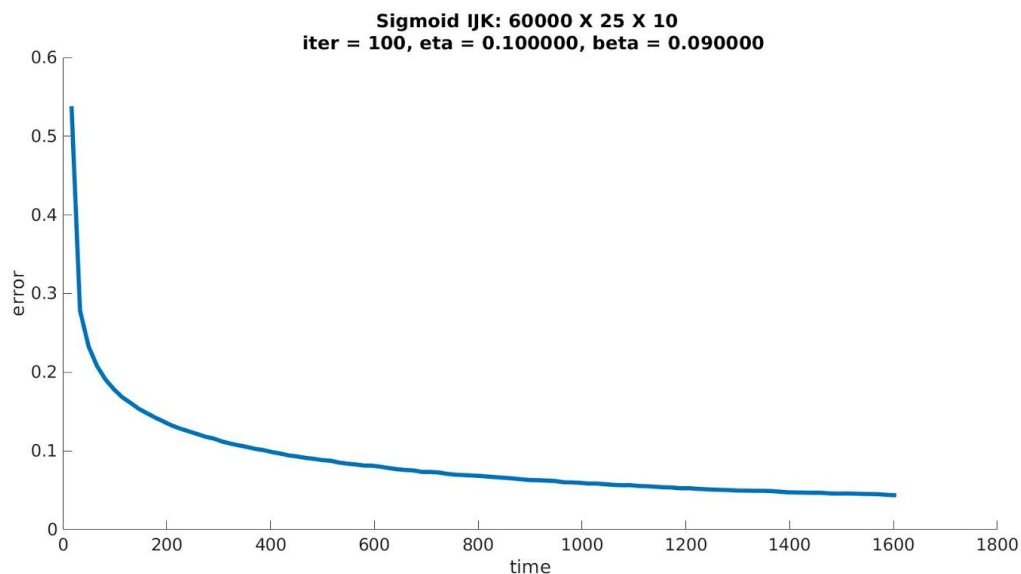
## 2. MNIST datasets

### 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。



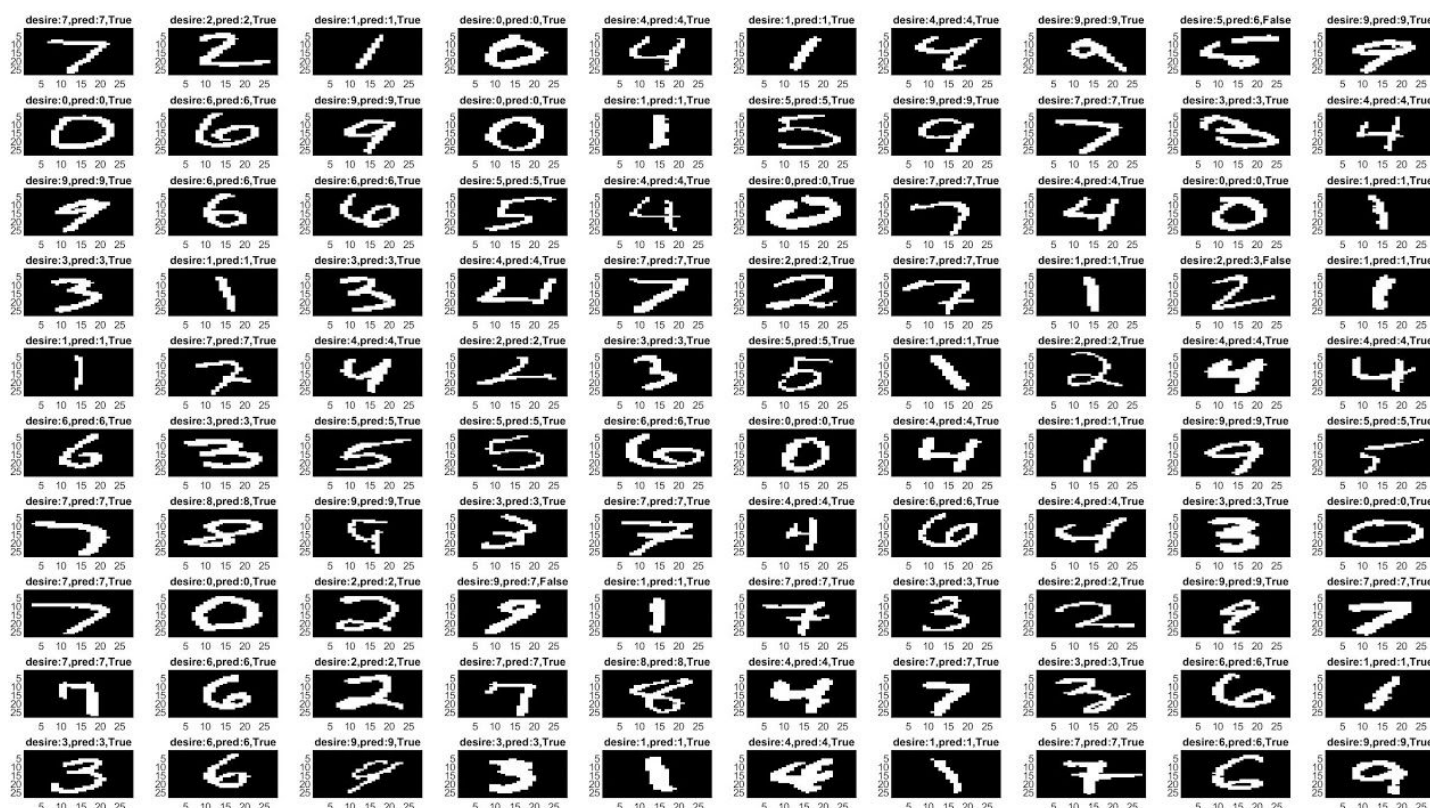
上圖為iteration與error 平均之關係。



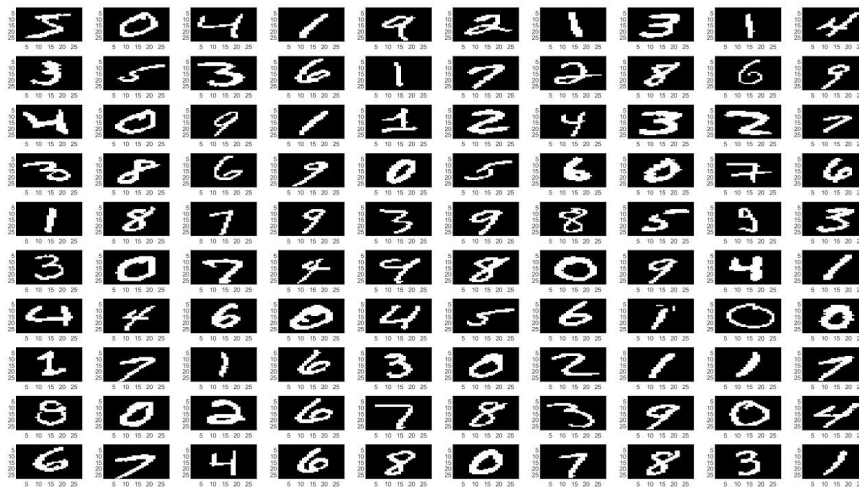
上圖為運行時間與error平均之關係。

Sigmoid Confusion Matrix													
True class	0	958		2	1	1	4	6	5	2	1	97.8%	2.2%
	1		1120	5	1		4	2	1	2		98.7%	1.3%
	2	7	2	974	4	7	5	6	8	18	1	94.4%	5.6%
	3		4	19	943	2	13		10	16	3	93.4%	6.6%
	4	2	4	8		932	1	5	2	5	23	94.9%	5.1%
	5	9	3	1	25	1	818	13	3	13	6	91.7%	8.3%
	6	11	3	5		7	9	918		5		95.8%	4.2%
	7		6	15	6	9	1	2	973	6	10	94.6%	5.4%
	8	6	1	18	11	7	12	9	8	894	8	91.8%	8.2%
	9	3	4	1	9	25	6	4	19	9	929	92.1%	7.9%
		96.2%	97.6%	92.9%	94.3%	94.0%	93.7%	95.1%	94.6%	92.2%	94.7%		
		3.8%	2.4%	7.1%	5.7%	6.0%	6.3%	4.9%	5.4%	7.8%	5.3%		
		0	1	2	3	4	5	6	7	8	9		
		Predicted class											

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。

True class	0		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%		
		0	1	2	3	4	5	6	7	8	9		
		Predicted class											

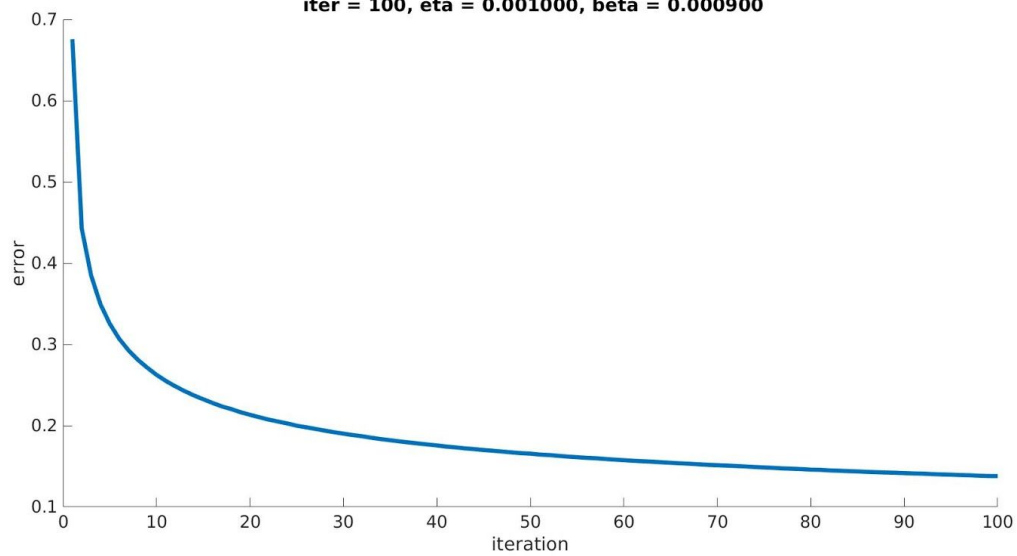
## B. ReLu

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%。

ReLU Confusion Matrix													
True class	0	965		2	1		2	5	1	3	1	98.5%	1.5%
	1		1118	4	3	1	1	4	1	3		98.5%	1.5%
	2	10	1	994	4	6		4	5	8		96.3%	3.7%
	3	5		7	972		14		9	1	2	96.2%	3.8%
	4	1		4		956		6	1	2	12	97.4%	2.6%
	5	3	2		13	1	864	4		3	2	96.9%	3.1%
	6	7	2	1		4	6	933		5		97.4%	2.6%
	7	2	7	8	4	3			996	1	7	96.9%	3.1%
	8	9	1	4	7	7	9	3	5	925	4	95.0%	5.0%
	9	12	5		3	16	6	1	7	5	954	94.5%	5.5%
95.2% 98.4% 97.1% 96.5% 96.2% 95.8% 97.2% 97.2% 96.8% 97.1%													
4.8% 1.6% 2.9% 3.5% 3.8% 4.2% 2.8% 2.8% 3.2% 2.9%													
0 1 2 3 4 5 6 7 8 9													
Predicted class													

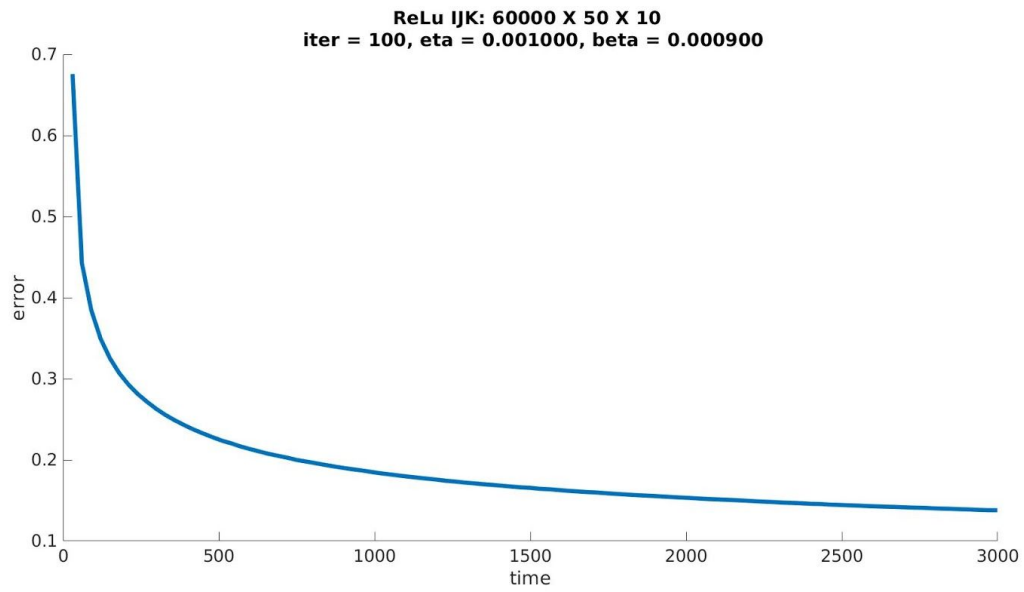
上圖為confusion matrix。

ReLU IJK: 60000 X 50 X 10  
iter = 100, eta = 0.001000, beta = 0.000900

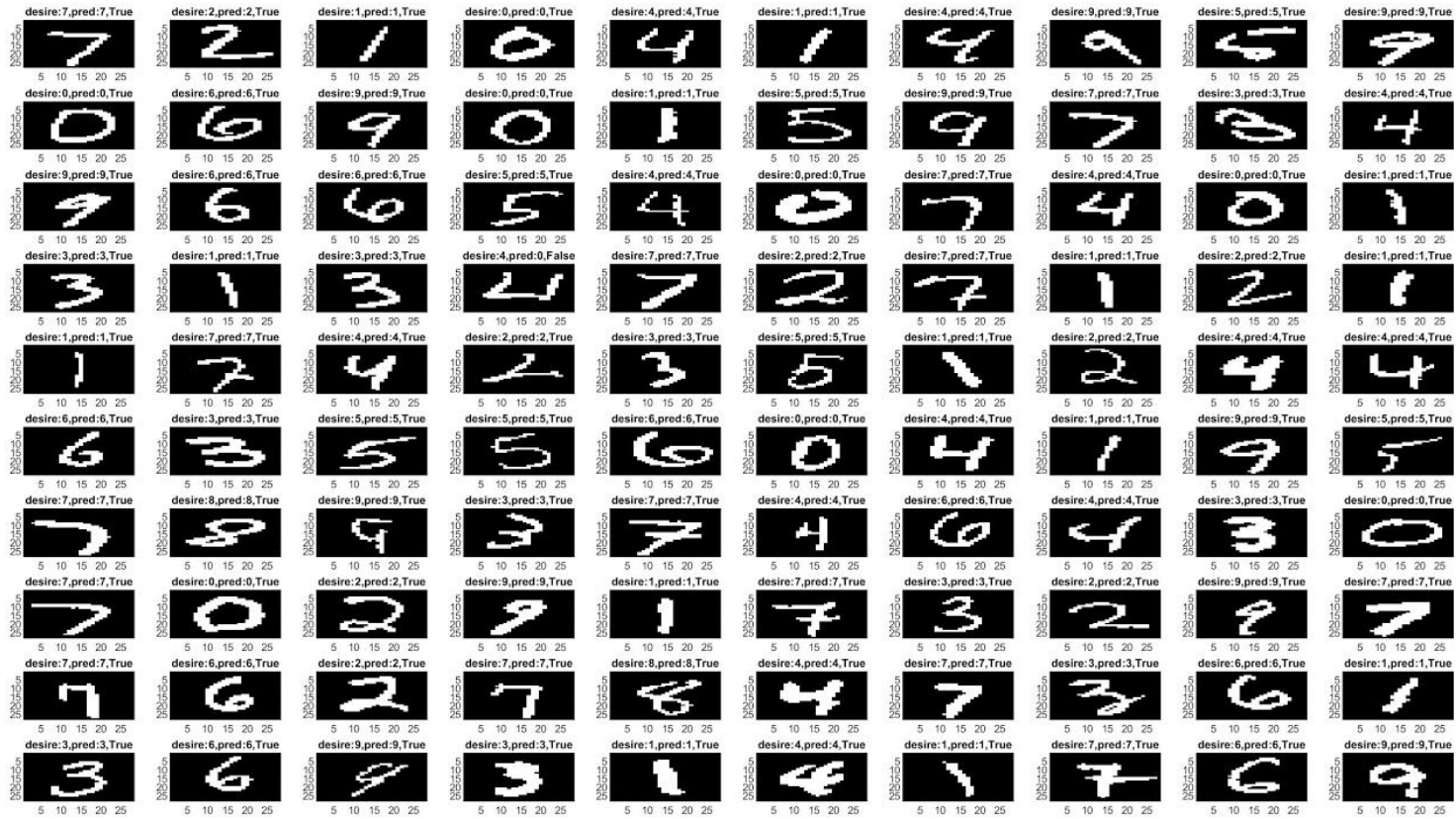




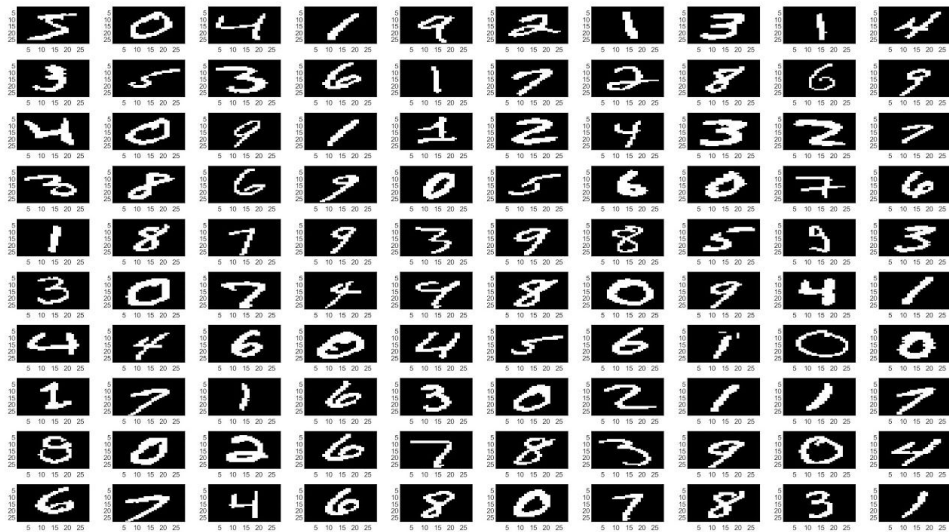
上圖為iteration與error平均作圖。



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。

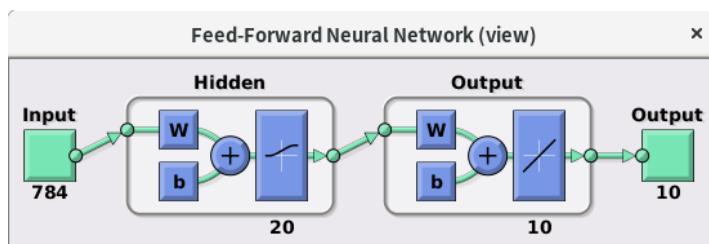


上圖為前100筆訓練資料。

## C. Nerual Network Tool box

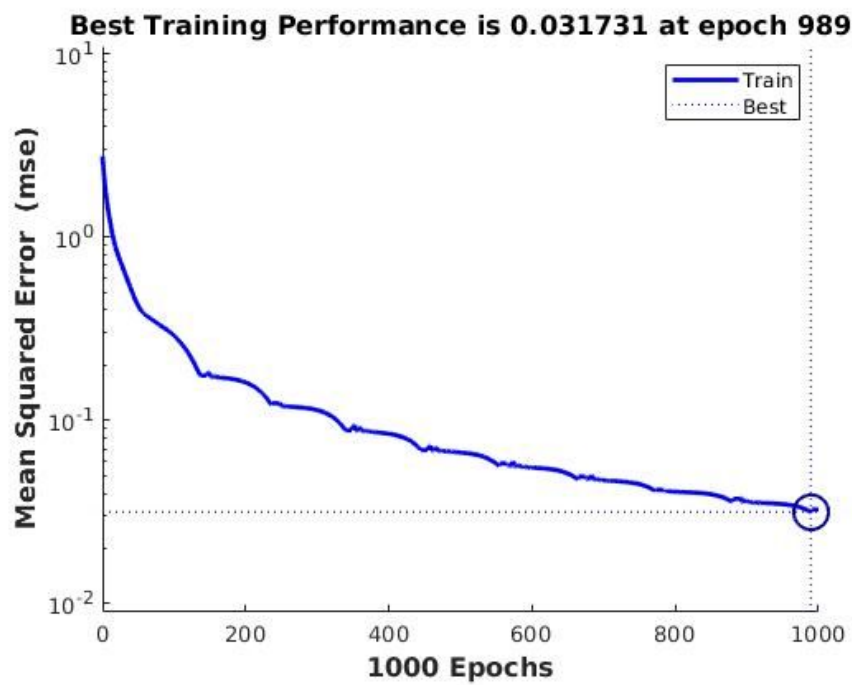
(1)選取 1000 個 examples 做 training, 1000 個做 testing。

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。

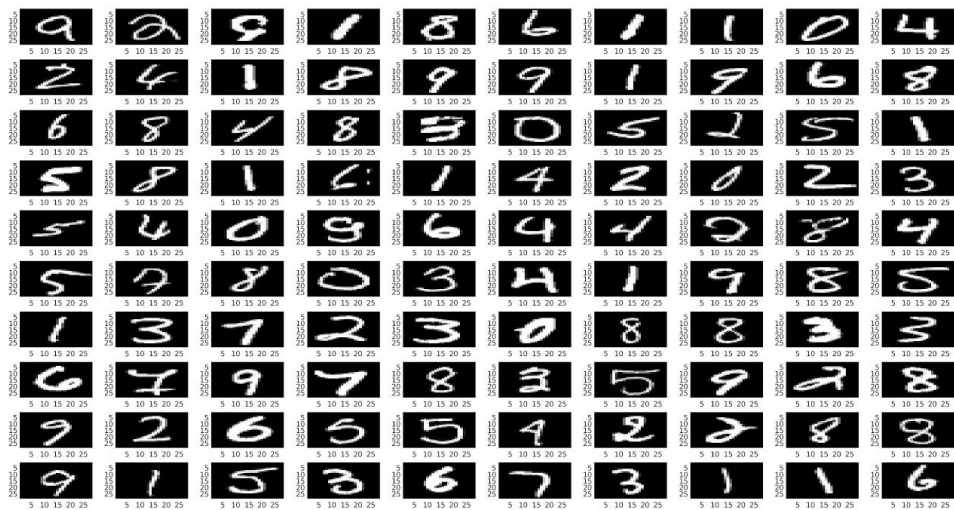
**Package Confusion Matrix**

0	87					1	1		1		96.7%	3.3%
1		109		1					5		94.8%	5.2%
2	4	6	85	3	1	2	7	1	3		75.9%	24.1%
3			6	75	2	12	1	1	4		74.3%	25.7%
4		2	1		76	1	1		2	7	84.4%	15.6%
5	4	3	1	4	5	58	1	5	3		69.0%	31.0%
6	2	4				7	83		1		85.6%	14.4%
7	2	1	3	1		2		96	1	4	87.3%	12.7%
8	3	1	3	2	1	7		2	74	3	77.1%	22.9%
9	1	2	1	2	10	2		3	5	79	75.2%	24.8%
	84.5%	85.2%	85.0%	85.2%	80.0%	63.0%	88.3%	88.9%	74.7%	84.9%		
	15.5%	14.8%	15.0%	14.8%	20.0%	37.0%	11.7%	11.1%	25.3%	15.1%		
	0	1	2	3	4	5	6	7	8	9		

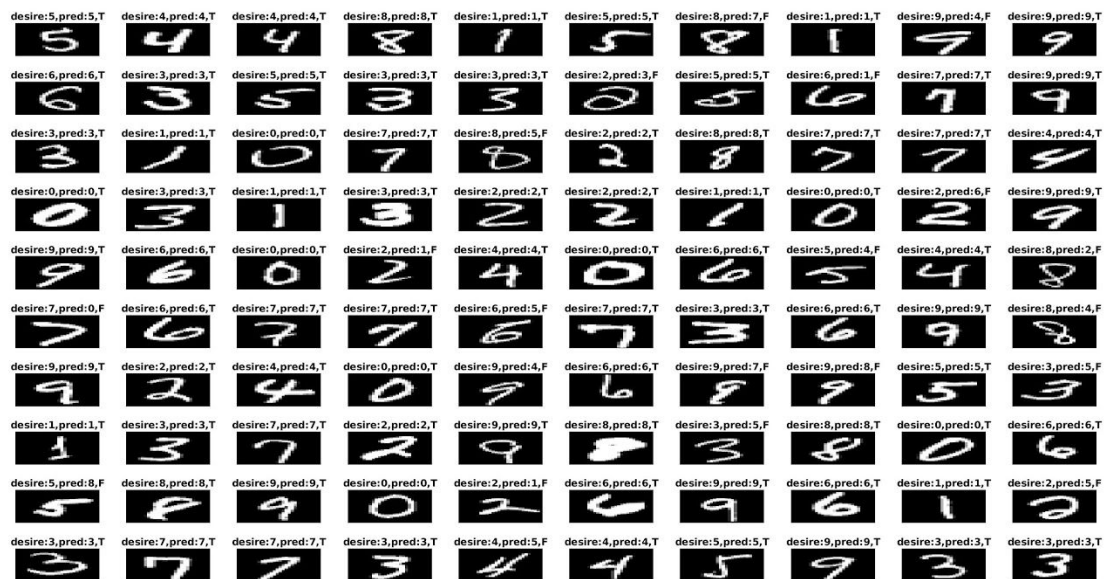
True class

Predicted class

上圖為confusion matrix。

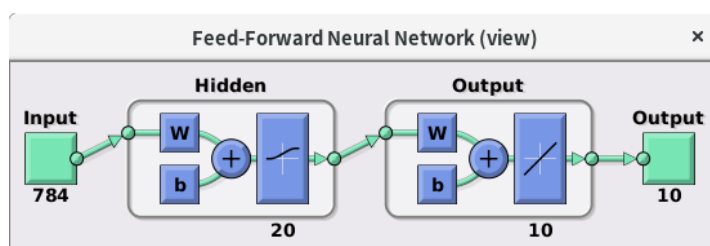


上圖為train data前100項。

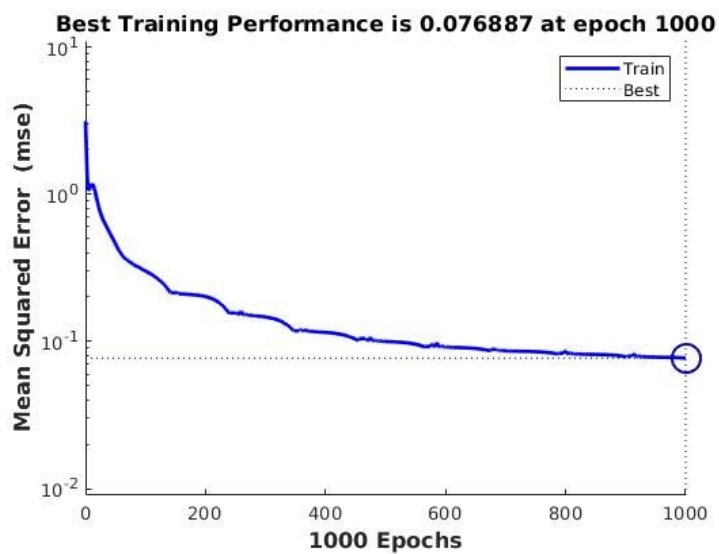


上圖為test data前100筆預測於實際結果。

(2)全部的 60000 個 examples 做 training, 10000 個做 testing。  
準確度為91.70%。

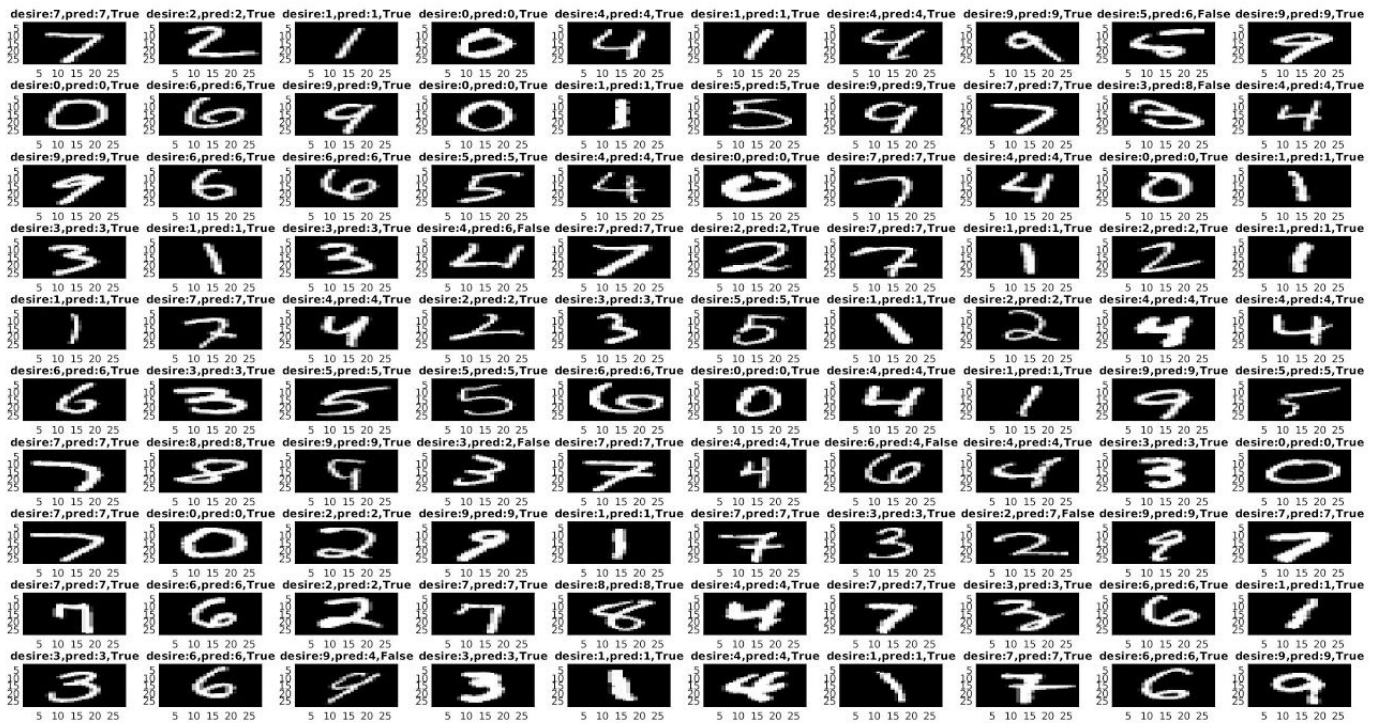


網路架構如上圖。

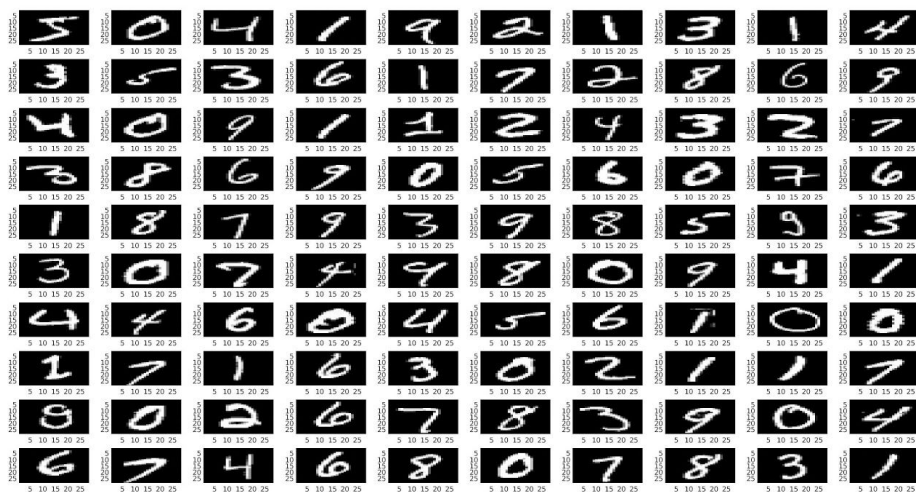


上圖為Mean square error與Epochs的關係。下圖為confusion matrix。

Package Confusion Matrix													
True class	0	960		1	1		5	7	2	4		98.0%	2.0%
	1	1	1104	3	4	1		5	2	15		97.3%	2.7%
	2	24	2	910	9	15	2	10	17	40	3	88.2%	11.8%
	3	20		15	901	2	18	4	14	30	6	89.2%	10.8%
	4	2	1	4		918		13	1	9	34	93.5%	6.5%
	5	25	2	3	36	10	755	16	9	27	9	84.6%	15.4%
	6	16	3	3	1	12	13	901		9		94.1%	5.9%
	7	6	10	21	1	12			944	9	25	91.8%	8.2%
	8	15	6	3	12	9	4	9	11	904	1	92.8%	7.2%
	9	14	5	1	16	58	3	1	22	16	873	86.5%	13.5%
	88.6%	97.4%	94.4%	91.8%	88.5%	94.4%	93.3%	92.4%	85.0%	91.8%			
	11.4%	2.6%	5.6%	8.2%	11.5%	5.6%	6.7%	7.6%	15.0%	8.2%			
	0	1	2	3	4	5	6	7	8	9			
Predicted class													



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traindx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Normalization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。