圖形識別作業二(a)

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ABIJK、BIJK、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。針對各個問題，先產生資料，接下來呼叫自定義網路，最後建立error plot與decision region plot部份。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。送入網路資料先進行一次正規化。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。當訓練結束後印出error與iteration關係，並且在函數範圍內以一定間距灑點，並依照class上色，繪製出decision region。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trainlm。

以下為所需公式：

(a) Sigmoid

Activation function : f(s) =

微分形式：(s) =

(b) ReLu

Activation function:

微分形式：  
(c) Adjust weight

對於IJK網路來說：

-+= for all j,i

-+= 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，

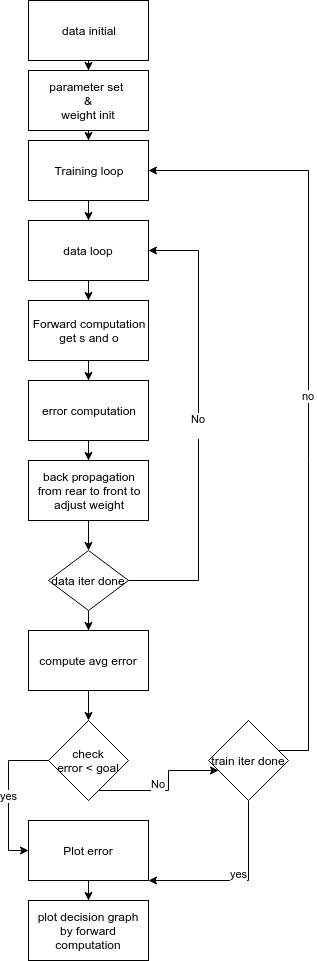
標準差為

(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 平均為:

以下為流程圖：



# 2.Two spiral problem

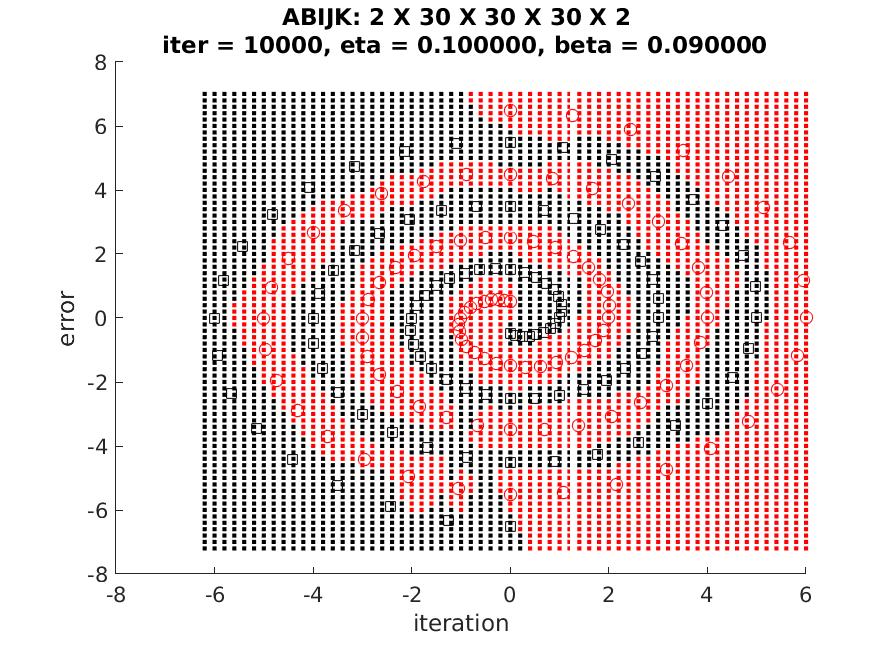
## A. 敘述

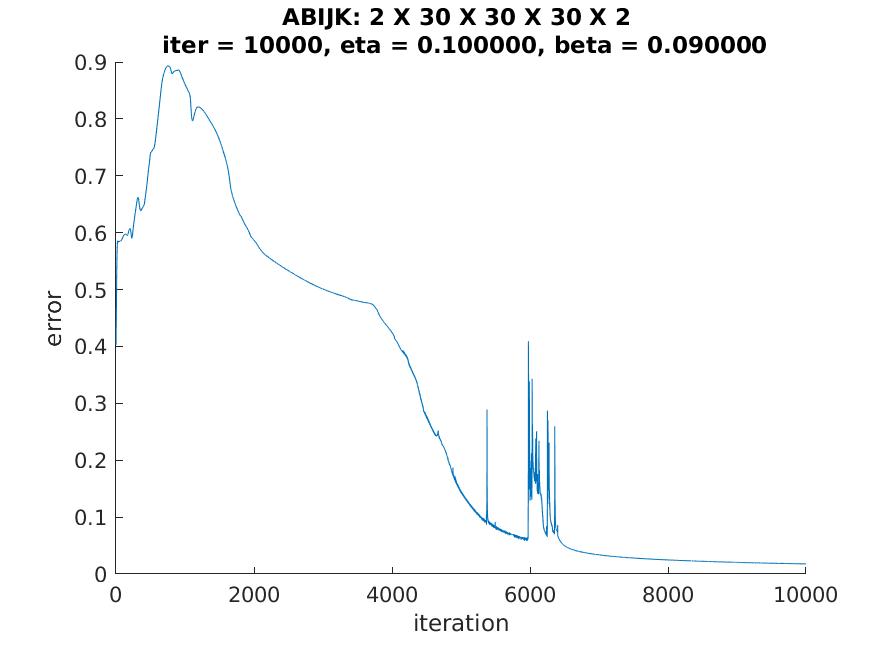
此dataset為三個問題中最困難的因此前兩部份均使用ABIJK網路進行trainning。

## B.結果

### (a) Sigmoidal function

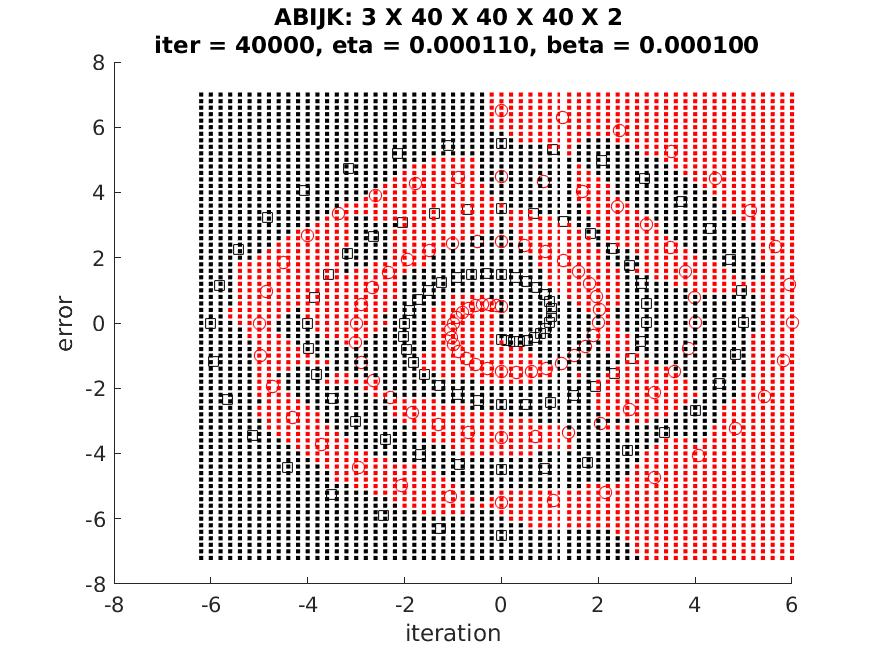
若使用ABIJK三層hidden layer進行training，其中hidden layer均為30，learning rate(eta)為0.1，beta為0.09，iteration數為10000，停止條件為error avg小於0.001。可以幾乎完全分離。

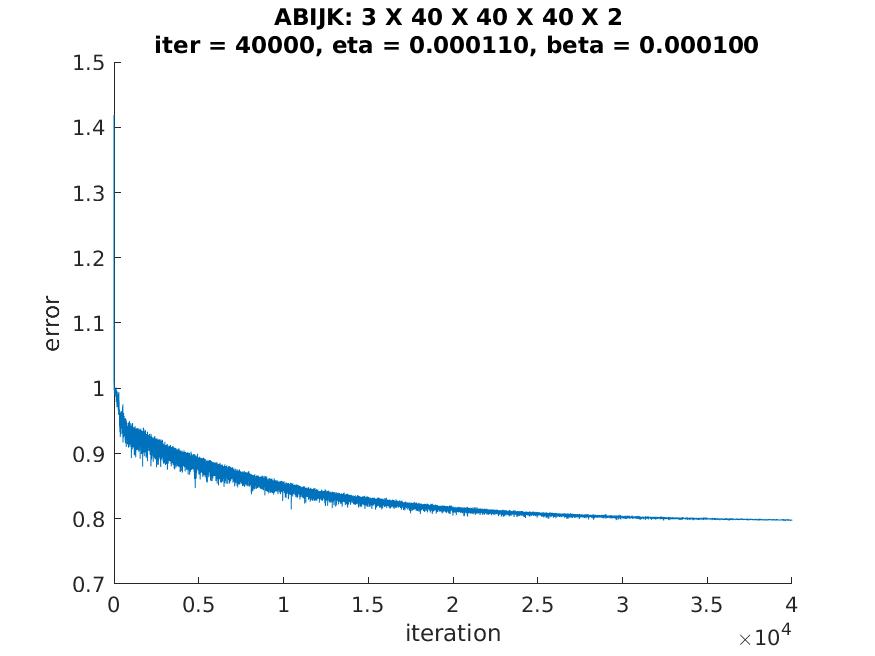




### (b) ReLu

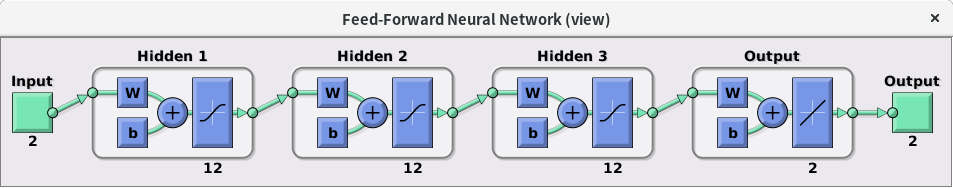
由於一直train不出來，因此ReLu部份新增了兩個輸入，分別是x\*y與x\*x+y\*y(以上x與y均經過正規化)。使用三層hidden layer為[40 40 40]，learning rate(eta)為0.00011，beta為0.0001(momentum term未乘上learning rate，因此數字很小)。iteration共40000次。此版本幾乎分離。

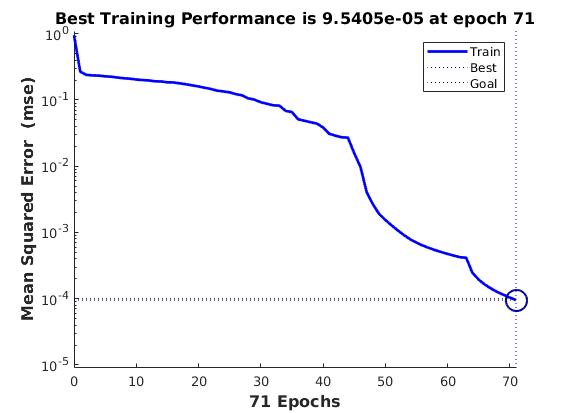


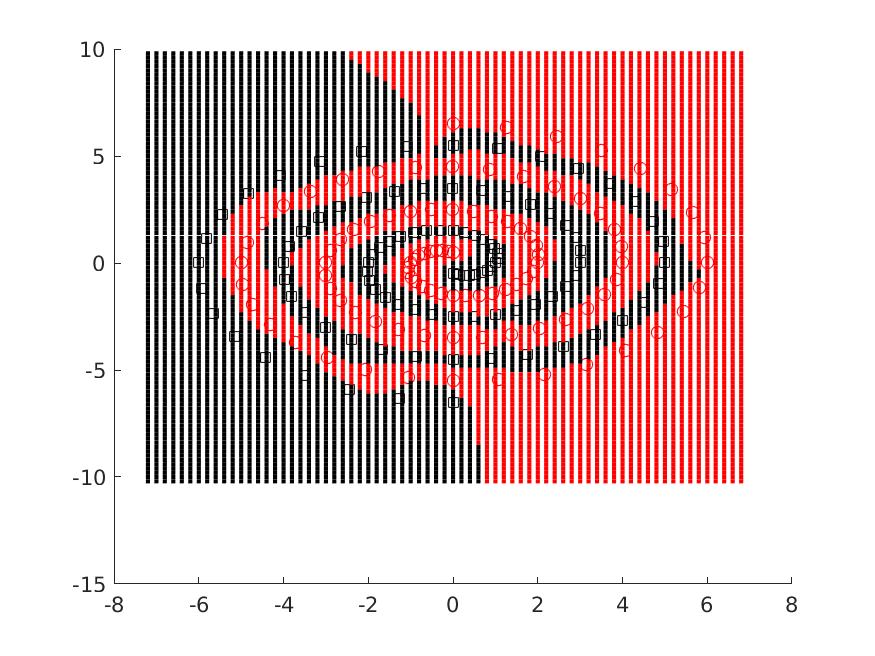


### (c) Neural networks toolbox

使用FeedFowradNet，網路為[12 12 12]，learning rate為0.01，epoch為10000，無使用momnetum，training rule使用trainlm(Levenberg-Marquardt backpropagation)。完全分離兩點集合。







## C.討論

Sigmoid的training速度較慢，而且容易卡在local minimum，所以momentum term很重要。hidden layer變多與每層neuron數目上升均可以使分辨精準度上升，但是learning rate需要往下降。至於ReLU，在此題中如直接使用均會直接發散，原因是ReLU反應過於激烈，因此需要把learning rate調很低(與sigmoid相比)，但是training 速度較快。因此這裡特別試了將最後一層改用Sigmoid的方式。可以分辨精準度上升(還是很糟)。

神經數目從toolbox可知大約[12 12 12]即可達成，但是可能是優化與learning rate關係，所以ㄉ要以更多的神經元才可以達成目標。

# 3. Double-moon problem

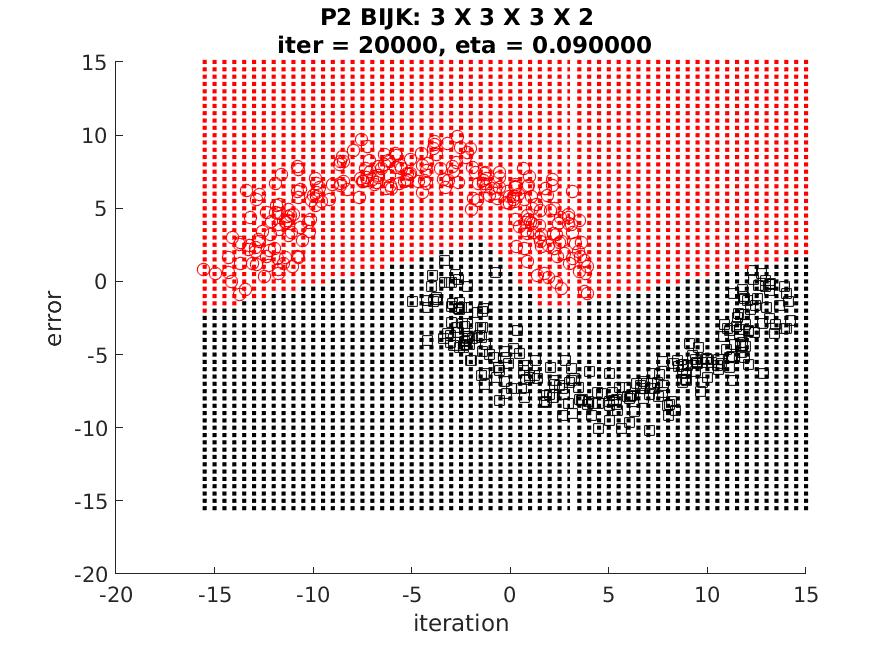
## A. 敘述

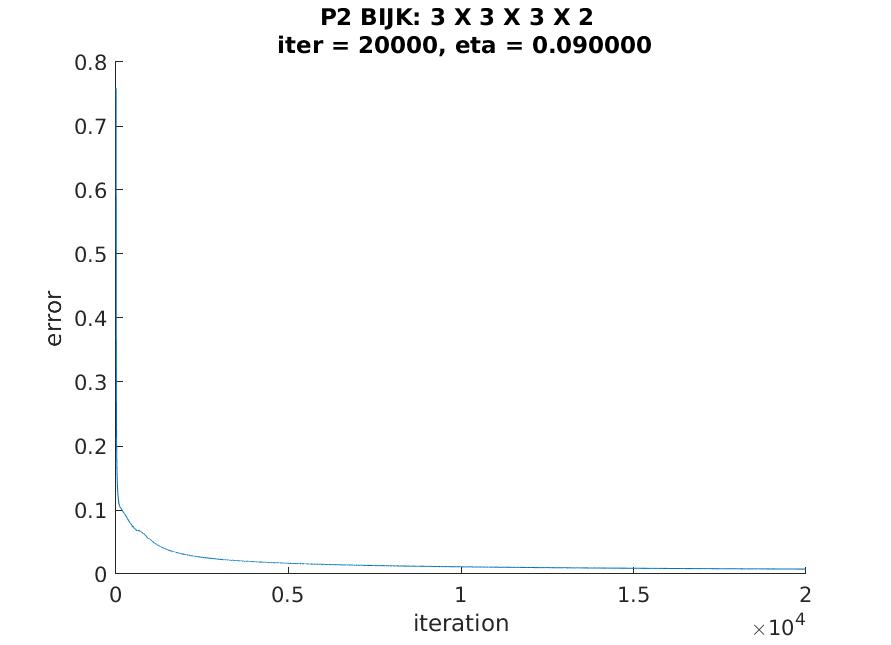
此dataset為三個問題中最簡單的因此前兩部份均使用BIJK網路進行trainning。由於半月形的特性，此問題有機會以閃電狀函數解決。

## B.結果

### (a) Sigmoidal function

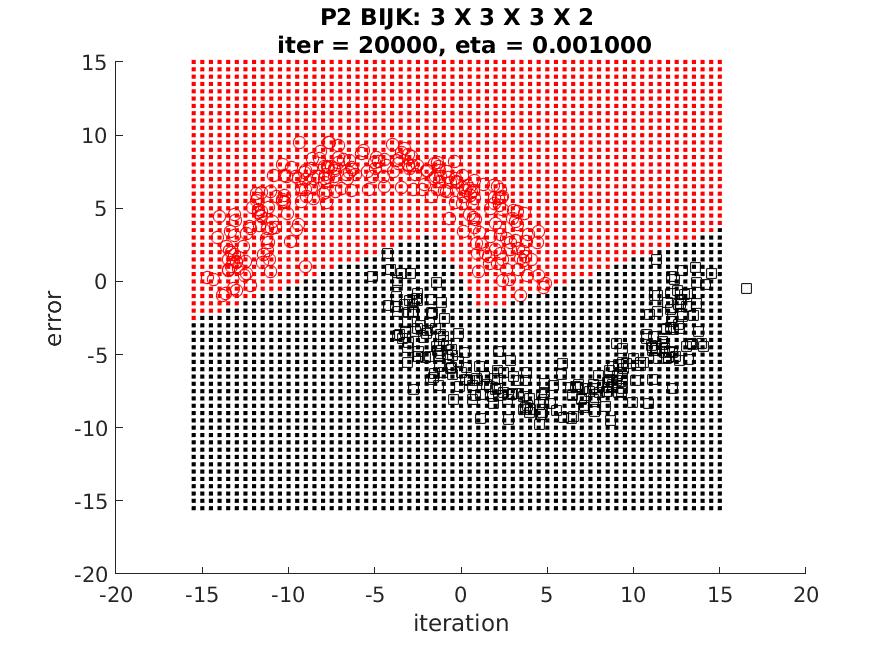
若使用BIJK兩層hidden layer進行training，其中兩層hidden layer分別為3，learning rate(eta)為0.09，beta為0.09，iteration數為20000，停止條件為error avg小於0.001。可以完全分離。

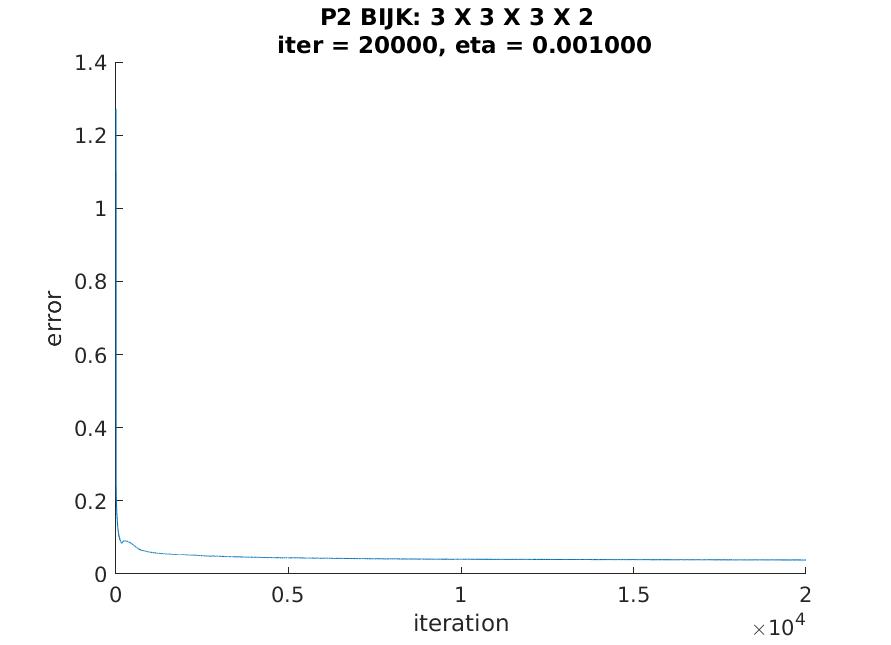




### (b) ReLu

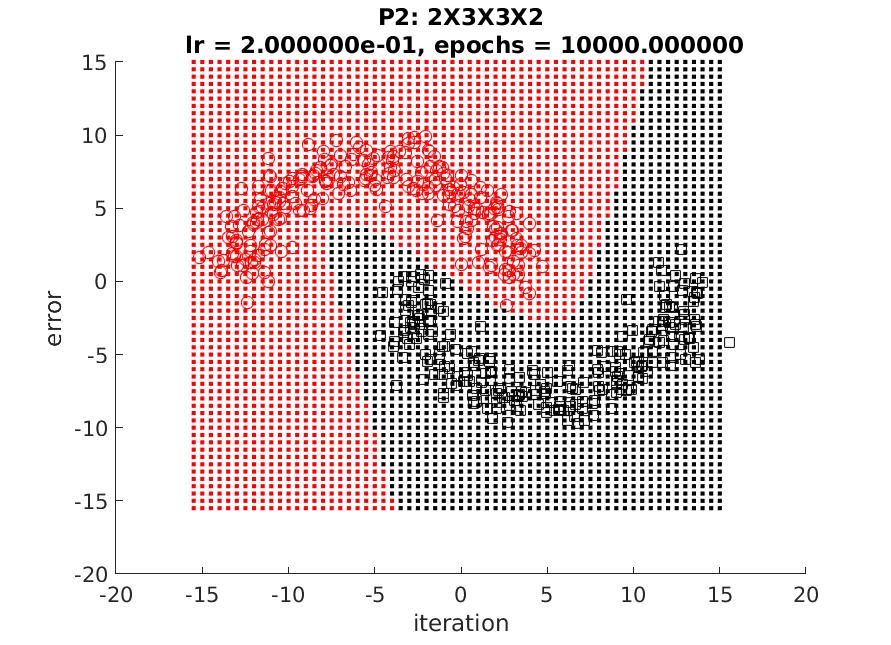
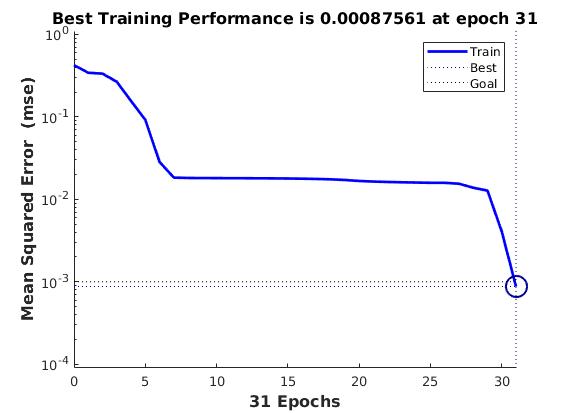
使用BIJK兩層hidden layer進行training，其中兩層hidden layer分別為3，learning rate(eta)為0.001，beta為0.001，iteration數為20000，停止條件為error avg小於0.001。可以完全分離。





### (c) Neural networks toolbox

使用FeedFowradNet，網路為[3 3]，learning rate為0.2，epoch為10000，無使用momnetum，training rule使用trainlm(Levenberg-Marquardt backpropagation)。完全分離兩點集合。



## C.討論

此題中因為層數不是很多，所以ReLu造成的發散情況不明顯，所以可以很快速的train出結果。此外3種方式都用相同的神經元數目，因此此種選擇方式應該是不錯的解。

# 4. Given 4 classes with Gaussian distribution

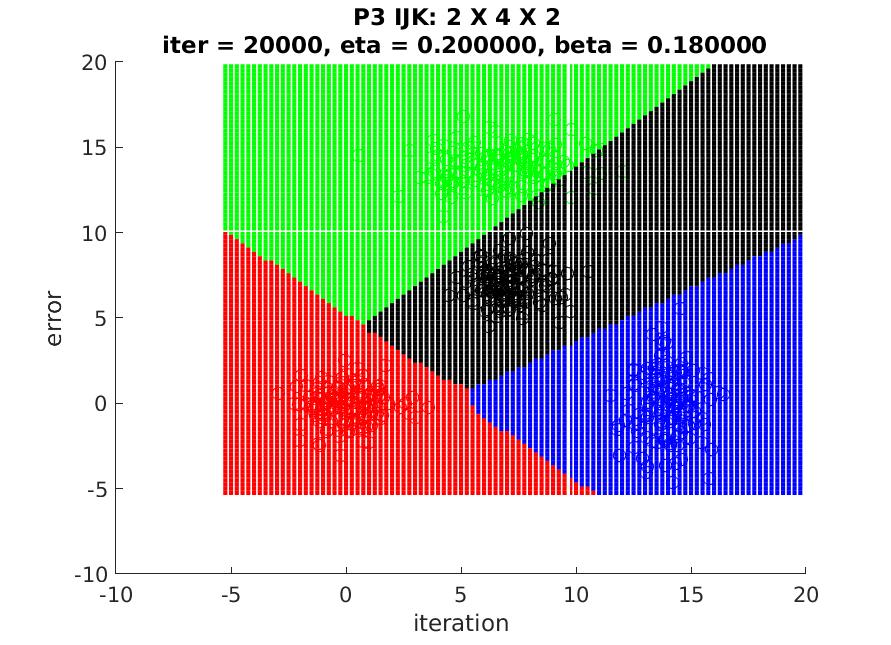
## A. 敘述

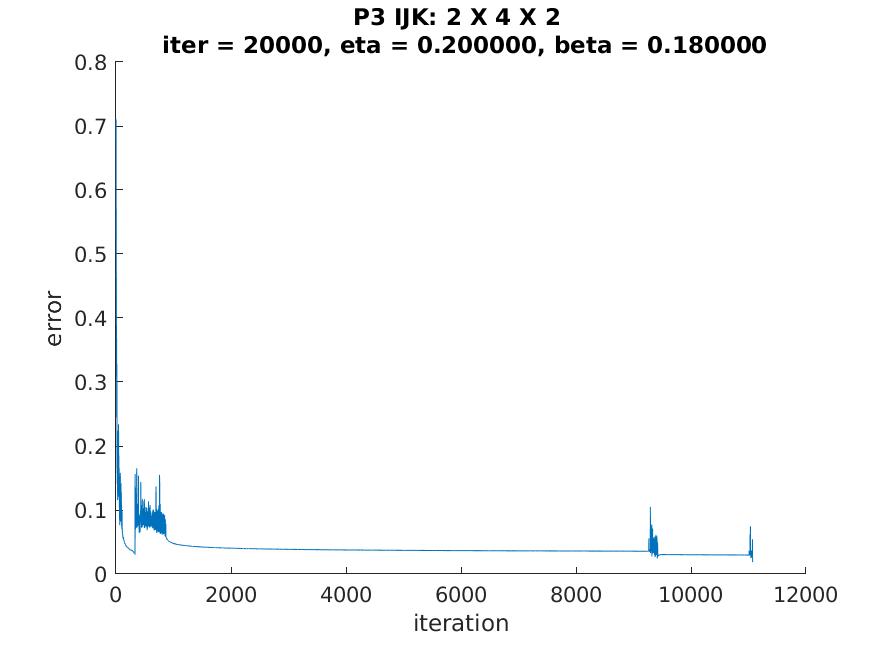
此dataset為三個問題中可以肉眼觀察透過直線分離的因此前兩部份均使用IJK網路進行trainning。由於三個class，彼此有所落差，因此將資料進行正規化，以去除不同大小的影響。

## B.結果

### (a) Sigmoidal function

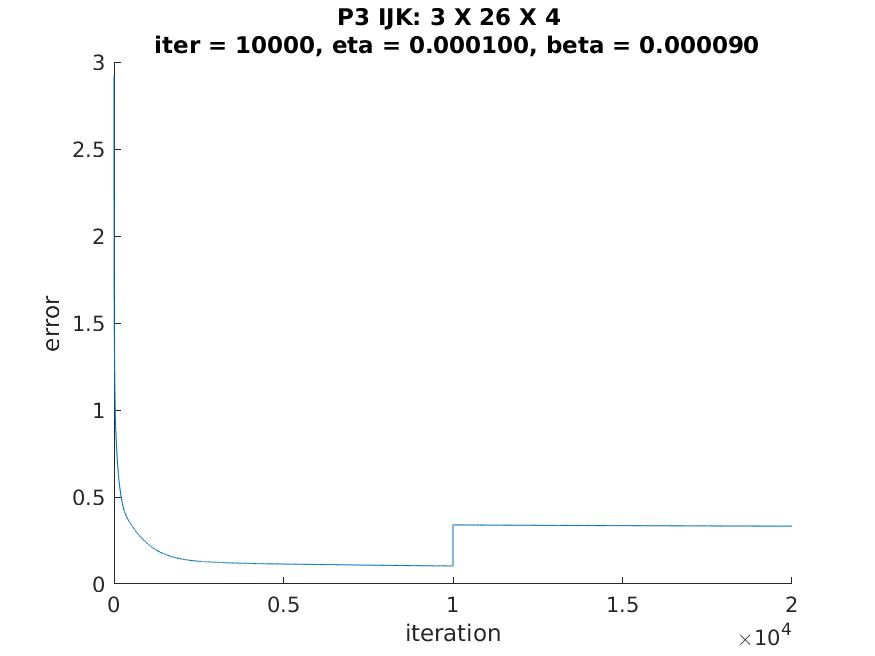
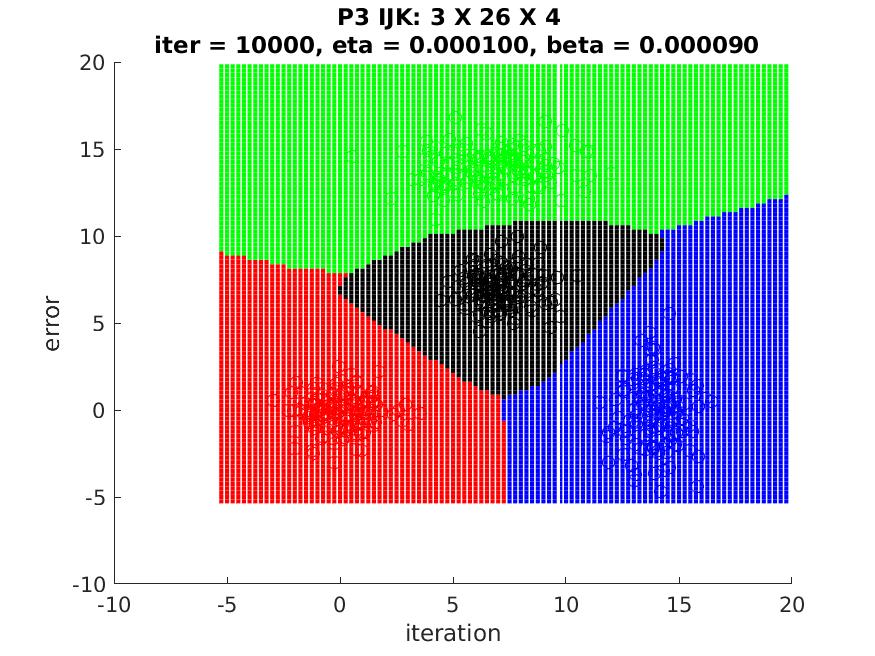
若使用IJK一層hidden layer進行training，其中hidden layer為4，learning rate(eta)為0.2，beta為0.18cd，iteration數為20000，停止條件為error avg小於0.001。可以幾乎完全分離。





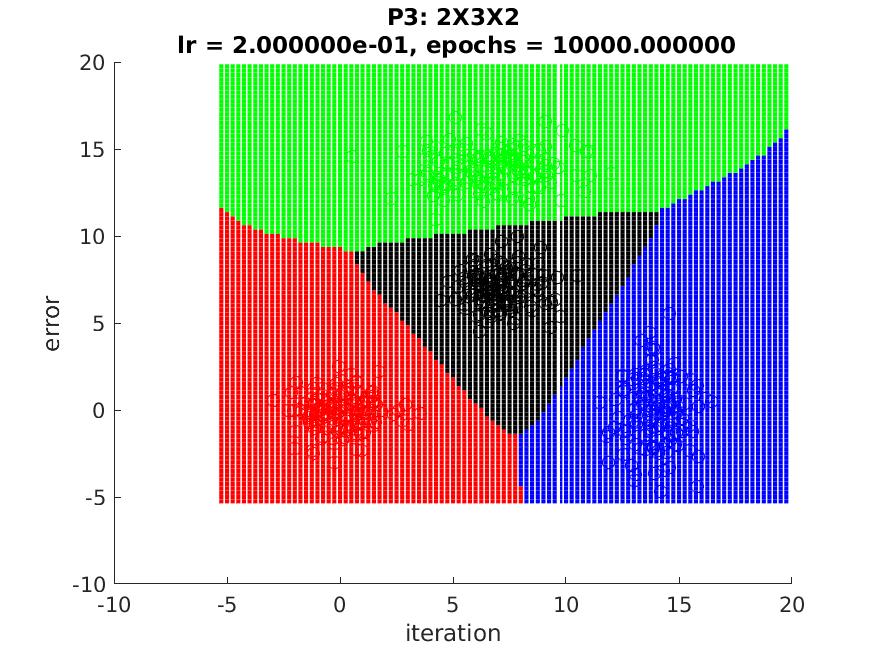
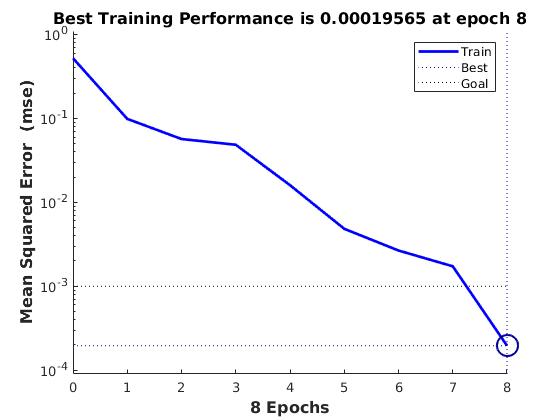
### (b) ReLu

使用IJK一層hidden layer進行training，其中hidden layer為25，learning rate(eta)為0.0001，beta為0.00009，iteration數為20000，停止條件為error avg小於0.001。可以完全分離。



### (c) Neural networks toolbox

使用FeedFowradNet，網路為hidden layer 一層[3]，learning rate為0.2，epoch為10000，無使用momnetum，training rule使用trainlm(Levenberg-Marquardt backpropagation)。完全分離兩點集合。



## C.討論

此問題由於有四個class，但是彼此間十分分離，因此使用一層hidden layer進行運算(IJK網路)，肉眼來看大約需要三條直線即可分離，不過由於離群值的存在，因此我自己的Activation function需要多一點神經元，而ReLu由於一直train不出來，所以中間hidden layer加到了26個才成功分離。

# 5.結論

Sigmoid特性是輸出值限於[0 ~ 1]，所以比較不容易讓網路崩毀，但是也因為這個特性，它比較不敏感，需要較高的learning rate，而ReLu十分敏感，很容易讓網路崩毀(值不再更新)，因此需要讓每次train的資料進行一次shuffle，同時如果有多種class，資料需做正規化，才可以減少網路不更新的風險，而learning rate也要調很低。因此有許多的修正方式，如更換部份層的Activation function，或是用一些learning rate的動態調整方式來優化training過程。Neural Network toolbox，其本身有做大量優化，而且也有多種train方式選擇，用起來十分方便。

hidden node數量理論上根據所需分割的區塊數與切割線段數決定，但是實際應用上不是這麼容易找到好的解，因此可以多加一些node用時間和空間換取準確度。層數愈多可以分的區塊愈複雜，但是對於ReLU來說網路果沒經過處理很容易失效，可以從上面ReLu部份的error average得知，網路很容易卡在一個階段，error average完全不更新。另外如果沒有將資料進行shuffle，網路也蠻容易卡住。

# 6. 原始碼

**1. Sigmoid.m**

function o = Sigmoid(s)  
 o = 1/(1+exp(-s));  
end

**2. deSigmoid.m**

function o = deSigmoid(s)

o = s\*(1.0-s);

end

**3. ReLu.m**

function o = ReLu(s)

o = max(s,0);

end

**4. deReLu.m**

function o = deReLu(s)

o = (s > 0);

end

**5. Activation function**

function o = Activation(s)

o = ReLu(s); %替換此函式更換不同Activation function

end

**6. deActivation function**

function o = deActivation(s)

o = deReLu(s); %替換此函式更換不同Activation function微分

end

**7. MLP\_P1.m (第一題)**

lear all;

N=97;

i = 0:1:96;

theta = i.\*pi/16;

r = 6.5\*(104-i)/104;

data(1:97,1) = r.\*sin(theta);

data(1:97,2) = r.\*cos(theta);

data(1:97,3) = 0;

data(1:97,4) = 0;

data(1:97,5) = 1;

data(1:97,6) = 0;

data(98:98+96,1) = -1\*r.\*sin(theta);

data(98:98+96,2) = -1\*r.\*cos(theta);

data(98:98+96,3) = 0;

data(98:98+96,4) = 0;

data(98:98+96,5) = 0;

data(98:98+96,6) = 1;

x\_max = max(data(:,1));

x\_min = min(data(:,1));

y\_max = max(data(:,2));

y\_min = min(data(:,2));

n\_data(:,1) = data(:,1);

n\_data(:,2) = data(:,2);

data(:,1) = (data(:,1) - x\_min)/(x\_max - x\_min);

data(:,2) = (data(:,2) - y\_min)/(y\_max - y\_min);

data(:,3) = data(:,1).\*data(:,1)+data(:,2).\*data(:,2);

data(:,4) = data(:,1).\*data(:,2);

layer = [40 40 40];

itermax = 40000;

eta = 0.00011;

beta = 0.00010;

Lowerlimit = 0.001;

title\_text = sprintf('ABIJK: 3 X %d X %d X %d X 2 \n iter = %d, eta = %f, beta = %f', layer(1),layer(2),layer(3),itermax,eta,beta);

file\_text = sprintf('P1\_ABIJK\_3X%dX%dX%dX2\_iter\_%d\_eta\_%f\_beta\_%f', layer(1),layer(2),layer(3),itermax,eta,beta);

[wkj,wji,wib,wba,error\_r,ite] = train\_ABIJK\_net(data,eta,beta,layer,4,2,itermax,Lowerlimit);

fig\_error = figure(1);

hold on;

plot(ite, error\_r);

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_error,strcat(file\_text,'\_error.jpg'));

saveas(fig\_error,strcat(file\_text,'\_error.fig'));

fig\_decision = figure(2);

hold on;

for n=1:1:N

plot(n\_data(n,1), n\_data(n,2),'r o');

end

for n=N+1:1:N\*2

plot(n\_data(n,1), n\_data(n,2),'k s');

end

for ix=-30:1:31

for iy=-35:1:36

dx=0.2\*(ix-1);

dy=0.2\*(iy-1);

ndx = (dx-x\_min)/(x\_max - x\_min);

ndy = (dy-y\_min)/(y\_max - y\_min);

oa=[ndx ndy ndx\*ndx+ndy\*ndy ndx\*ndy 1]';

ok = FeedFoward\_ABIJK(wba,wib,wji,wkj,oa,4,2,layer);

% Real output

if ok(1,1)<0.5

plot(dx,dy, 'k .');

elseif ok(1,1)>0.5

plot(dx,dy, 'r .');

end

end

end

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));

**8. MLP\_P2.m (第二題)**

clear all;

N=250;

theta1 = linspace(-180,180, N)\*pi/360;

r = 8

data(1:N,1) = -5 + r\*sin(theta1)+randn(1,N);

data(1:N,2) = r\*cos(theta1)+randn(1,N);

data(1:N,3) = 1;

data(1:97,4) = 0;

data(N+1:2\*N,1) = 5 + r\*sin(theta1)+randn(1,N);

data(N+1:2\*N,2) = -r\*cos(theta1)+randn(1,N);

data(N+1:2\*N,3) = 0;

data(N+1:2\*N,4) = 1;

itermax=20000;

eta=0.09; % (n -> eta -> learning rate)

beta=0.09; % momentum term

Lowerlimit=0.001;

layer = [3 3];

title\_text = sprintf('P2 BIJK: 2 X %d X %d X 2 \n iter = %d, eta = %f,beta = %f',layer(1),layer(2),itermax,eta,beta);

file\_text = sprintf('P2\_BIJK\_2X%dX%dX2\_iter\_%d\_eta\_%f\_beta\_%f',layer(1),layer(2),itermax,eta,beta);

[wkj,wji,wib,error\_r,ite] = train\_BIJK\_net(data,eta,beta,layer,2,2,itermax,Lowerlimit);

fig\_error = figure(1);

hold on;

plot(ite, error\_r);

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_error,strcat(file\_text,'\_error.jpg'));

saveas(fig\_error,strcat(file\_text,'\_error.fig'));

fig\_decision = figure(2);

hold on;

for n=1:1:N

plot(data(n,1), data(n,2),'r o');

end

for n=N+1:1:N\*2

plot(data(n,1), data(n,2),'k s');

end

for ix=-30:1:31

for iy=-30:1:31

dx=0.5\*(ix-1);

dy=0.5\*(iy-1);

ob=[dx dy 1]';

ok = FeedFoward\_BIJK(wib,wji,wkj,ob,2,2,layer);

% Real output

if ok(1)<0.5

plot(dx,dy, 'k .');

elseif ok(1)>0.5

plot(dx,dy, 'r .');

end

end

end

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));

**9. MLP\_P3.m (第三題)**

clear all;

mu1 = [0 0];

sigma1 = [1 0; 0 1];

rng default; % For reproducibility

R = mvnrnd(mu1,sigma1,150);

N = 150;

data(1:N,1) = R(:,1);

data(1:N,2) = R(:,2);

data(1:N,3) = 1;

data(1:N,4) = 0;

data(1:N,5) = 0;

data(1:N,6) = 0;

mu2 = [14 0];

sigma2 = [1 0; 0 4];

R = mvnrnd(mu2,sigma2,150);

data(N+1:2\*N,1) = R(:,1);

data(N+1:2\*N,2) = R(:,2);

data(N+1:2\*N,3) = 0;

data(N+1:2\*N,4) = 1;

data(N+1:2\*N,5) = 0;

data(N+1:2\*N,6) = 0;

mu3 = [7 14];

sigma3 = [4 0; 0 1];

R = mvnrnd(mu3,sigma3,150);

data(2\*N+1:3\*N,1) = R(:,1);

data(2\*N+1:3\*N,2) = R(:,2);

data(2\*N+1:3\*N,3) = 0;

data(2\*N+1:3\*N,4) = 0;

data(2\*N+1:3\*N,5) = 1;

data(2\*N+1:3\*N,6) = 0;

mu4 = [7 7];

sigma4 = [1 0; 0 1];

R = mvnrnd(mu4,sigma4,150);

data(3\*N+1:4\*N,1) = R(:,1);

data(3\*N+1:4\*N,2) = R(:,2);

data(3\*N+1:4\*N,3) = 0;

data(3\*N+1:4\*N,4) = 0;

data(3\*N+1:4\*N,5) = 0;

data(3\*N+1:4\*N,6) = 1;

% MLP

Lowerlimit=0.02;

itermax=20000;

eta=0.2; % (n -> eta -> learning rate)

beta=0.18; % momentum term

layer=[4];

title\_text = sprintf('P3 IJK: 2 X %d X 2 \n iter = %d, eta = %f, beta = %f',layer(1),itermax,eta,beta);

file\_text = sprintf('P3\_IJK\_2X%dX2\_iter\_%d\_eta\_%f\_beta\_%f',layer(1),itermax,eta,beta);

[wkj,wji,error\_r,ite] = train\_IJK\_net(data,eta,beta,layer,2,4,itermax,Lowerlimit);

fig\_error = figure(1);

hold on;

plot(ite, error\_r);

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_error,strcat(file\_text,'\_error.jpg'));

saveas(fig\_error,strcat(file\_text,'\_error.fig'));

fig\_decision = figure(2);

hold on;

plot(data(1:N,1),data(1:N,2),'ro');

plot(data(N+1:2\*N,1),data(N+1:2\*N,2),'bo');

plot(data(2\*N+1:3\*N,1),data(2\*N+1:3\*N,2),'go');

plot(data(3\*N+1:4\*N,1),data(3\*N+1:4\*N,2),'ko');

for ix=-5\*4:1:20\*4

for iy=-5\*4:1:20\*4

dx=0.25\*(ix-1);

dy=0.25\*(iy-1);

oi=[dx dy 1]';

ok = FeedFoward\_IJK(wji,wkj,oi,2,4,layer);

[M,I] = max(ok);

if (I == 1)

plot(dx,dy, 'r .');

elseif (I == 2)

plot(dx,dy, 'b .');

elseif (I == 3)

plot(dx,dy, 'g .');

elseif (I == 4)

plot(dx,dy, 'k .');

end

end

end

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));

**10. train\_ABIJK\_net.m**

function [wkj,wji,wib,wba,error\_r,ite] = train\_ABIJK\_net(data,eta,beta,layer,input,output,itermax,Lowerlimit)

nvectors=length(data);

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

neuron\_hid\_layerI=layer(2);

neuron\_hid\_layerI\_with\_bias=neuron\_hid\_layerI+1;

neuron\_hid\_layerB=layer(3);

neuron\_hid\_layerB\_with\_bias=neuron\_hid\_layerB+1;

noutdim=output;

%initialize

wkj = normrnd(0,sqrt(2/(input+output)),noutdim,neuron\_hid\_layerI\_with\_bias);

wji = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerJ\_with\_bias,neuron\_hid\_layerI\_with\_bias);

wib = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerI\_with\_bias,neuron\_hid\_layerB\_with\_bias);

wba = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerB\_with\_bias,ninpdim\_with\_bias);

wkj\_tmp = zeros(size(wkj));

wji\_tmp = zeros(size(wji));

wib\_tmp = zeros(size(wib));

wba\_tmp = zeros(size(wba));

olddelwkj=zeros(noutdim , neuron\_hid\_layerJ\_with\_bias); % weight of Wkj (J -> K)

olddelwji=zeros(neuron\_hid\_layerJ\_with\_bias , neuron\_hid\_layerI\_with\_bias); % weight of Wji (I -> J)

olddelwib=zeros(neuron\_hid\_layerB\_with\_bias , neuron\_hid\_layerI\_with\_bias); % weight of Wji (B -> I)

olddelwba=zeros(neuron\_hid\_layerB\_with\_bias , ninpdim\_with\_bias); % weight of Wji (A -> B)

oa = zeros(ninpdim\_with\_bias,1);

oa(ninpdim\_with\_bias) = 1;

sb = zeros(neuron\_hid\_layerB\_with\_bias,1);

ob = zeros(neuron\_hid\_layerB\_with\_bias,1);

ob(neuron\_hid\_layerB\_with\_bias) = 1; % output of data

si = zeros(ninpdim\_with\_bias,1); % input of hidden layer i

oi = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oi(neuron\_hid\_layerI\_with\_bias) = 1; % output of hidden layer i

sj = zeros(neuron\_hid\_layerI\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

dk = zeros(noutdim,1); % desired output

iter=0;

error\_avg=10;

% internal variables

deltak = zeros(1,noutdim);

deltaj = zeros(1,neuron\_hid\_layerJ\_with\_bias);

deltai = zeros(1,neuron\_hid\_layerI\_with\_bias);

deltab = zeros(1,neuron\_hid\_layerB\_with\_bias);

sumback = zeros(1,max(neuron\_hid\_layerJ\_with\_bias, max(neuron\_hid\_layerI\_with\_bias,neuron\_hid\_layerB\_with\_bias)));

counter = 0;

while (error\_avg > Lowerlimit) && (iter<itermax)

iter=iter+1;

error=0;

data\_index = randperm(length(data));

if counter > 1000

counter = 0;

eta = eta\*0.9;

beta = beta\*0.9;

end

counter = counter + 1;

% Forward Computation:

for ivector=1:nvectors

rvector = data\_index(ivector);

oa=[data(rvector,1:input) 1]';

dk=[data(rvector,input+1:output+input)]';

for j=1:neuron\_hid\_layerB

sb(j)=wba(j,:)\*oa;

ob(j)=Activation(sb(j)); % sigmoid

end

ob(neuron\_hid\_layerB\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerI

si(j)=wib(j,:)\*ob;

oi(j)=Activation(si(j)); % sigmoid

end

oi(neuron\_hid\_layerI\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); % sigmoid

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); % signmoid

end

error=error+sum(abs(dk-ok)); % abs is absolute each element

% Backward learning:

for k=1:noutdim

deltak(k)=(dk(k)-ok(k))\*deActivation(ok(k)); % gradient term

end

for j=1:neuron\_hid\_layerJ\_with\_bias

for k=1:noutdim

wkj\_tmp(k,j)=wkj(k,j)+eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

olddelwkj(k,j)=eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

end

end

for j=1:neuron\_hid\_layerJ

sumback(j)=0.0;

for k=1:noutdim

sumback(j)=sumback(j)+deltak(k)\*wkj(k,j);

end

deltaj(j)=deActivation(oj(j))\*sumback(j);

end

for j=1:neuron\_hid\_layerI\_with\_bias

for k=1:neuron\_hid\_layerJ

wji(k,j)=wji(k,j)+eta\*deltaj(k)\*oi(j)+beta\*olddelwji(k,j);

olddelwji(k,j)=eta\*deltaj(k)\*oi(j)+beta\*olddelwji(k,j);

end

end

for j=1:neuron\_hid\_layerI

sumback(j)=0.0;

for k=1:neuron\_hid\_layerJ\_with\_bias

sumback(j)=sumback(j)+deltaj(k)\*wji(k,j);

end

deltai(j)=deActivation(oi(j))\*sumback(j);

end

for j=1:neuron\_hid\_layerB\_with\_bias

for k=1:neuron\_hid\_layerI

wbi(k,j)=wib(k,j)+eta\*deltai(k)\*ob(j)+beta\*olddelwib(k,j);

olddelwib(k,j)=eta\*deltai(k)\*ob(j)+beta\*olddelwib(k,j);

end

end

for j=1:neuron\_hid\_layerB

sumback(j)=0.0;

for k=1:neuron\_hid\_layerI\_with\_bias

sumback(j)=sumback(j)+deltai(k)\*wib(k,j);

end

deltab(j)=deActivation(ob(j))\*sumback(j);

end

for i=1:ninpdim\_with\_bias

for j=1:neuron\_hid\_layerB

wba(j,i)=wba(j,i)+eta\*deltab(j)\*oa(i)+beta\*olddelwba(j,i);

olddelwba(j,i)=eta\*deltab(j)\*oa(i)+beta\*olddelwba(j,i);

end

end

wkj = wkj\_tmp;

end

ite(iter)=iter;

error\_avg=error/nvectors;

error\_r(iter)=error\_avg;

end

end

**11. train\_BIJK\_net.m**

function [wkj,wji,wib,error\_r,ite] = train\_BIJK\_net(data,eta,beta,layer,input,output,itermax,Lowerlimit)

nvectors=length(data);

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

neuron\_hid\_layerI=layer(2);

neuron\_hid\_layerI\_with\_bias=neuron\_hid\_layerI+1;

noutdim=output;

%initialize

wkj = normrnd(0,sqrt(2/(input+output)),noutdim,neuron\_hid\_layerJ\_with\_bias);

wji = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerI\_with\_bias,neuron\_hid\_layerJ\_with\_bias);

wib = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerI\_with\_bias,ninpdim\_with\_bias);

wkj\_tmp = zeros(size(wkj));

wji\_tmp = zeros(size(wji));

wib\_tmp = zeros(size(wib));

olddelwkj=zeros(noutdim , neuron\_hid\_layerJ\_with\_bias); % weight of Wkj (J -> K)

olddelwji=zeros(neuron\_hid\_layerI\_with\_bias , neuron\_hid\_layerJ\_with\_bias); % weight of Wji (I -> J)

olddelwib=zeros(neuron\_hid\_layerI\_with\_bias,ninpdim\_with\_bias); % weight of Wji (B -> I)

ob = zeros(ninpdim\_with\_bias,1);

ob(ninpdim\_with\_bias) = 1;

si = zeros(ninpdim\_with\_bias,1); % input of hidden layer i

oi = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oi(neuron\_hid\_layerI\_with\_bias) = 1; % output of hidden layer i

sj = zeros(neuron\_hid\_layerI\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

dk = zeros(noutdim,1); % desired output

iter=0;

error\_avg=10;

% internal variables

deltak = zeros(1,noutdim);

deltaj = zeros(1,neuron\_hid\_layerJ\_with\_bias);

deltai = zeros(1,neuron\_hid\_layerI\_with\_bias);

sumback = zeros(1,max(neuron\_hid\_layerJ\_with\_bias, neuron\_hid\_layerI\_with\_bias));

while (error\_avg > Lowerlimit) && (iter<itermax)

iter=iter+1;

error=0;

% Forward Computation:

r\_index = randperm(length(data));

for ivector=1:nvectors

rvector = r\_index(ivector);

ob=[data(rvector,1:input) 1]';

dk=[data(rvector,input+1:input+output)]';

for j=1:neuron\_hid\_layerI

si(j)=wib(j,:)\*ob;

oi(j)=Activation(si(j)); % sigmoid

end

oi(neuron\_hid\_layerI\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); % sigmoid

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); % signmoid

end

error=error+sum(abs(dk-ok)); % abs is absolute each element

% Backward learning:

for k=1:noutdim

deltak(k)=(dk(k)-ok(k))\*deActivation(ok(k)); % gradient term

end

for j=1:neuron\_hid\_layerJ\_with\_bias

for k=1:noutdim

wkj(k,j)=wkj(k,j)+eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

olddelwkj(k,j)=eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

end

end

for j=1:neuron\_hid\_layerJ

sumback(j)=0.0;

for k=1:noutdim

sumback(j)=sumback(j)+deltak(k)\*wkj(k,j);

end

deltaj(j)=deActivation(oj(j))\*sumback(j);

end

for j=1:neuron\_hid\_layerI\_with\_bias

for k=1:neuron\_hid\_layerJ

wji(k,j)=wji(k,j)+eta\*deltaj(k)\*oi(j)+beta\*olddelwji(k,j);

olddelwji(k,j)=eta\*deltaj(k)\*oi(j)+beta\*olddelwji(k,j);

end

end

for j=1:neuron\_hid\_layerI

sumback(j)=0.0;

for k=1:neuron\_hid\_layerJ\_with\_bias

sumback(j)=sumback(j)+deltaj(k)\*wji(k,j);

end

deltai(j)=deActivation(oi(j))\*sumback(j);

end

for i=1:ninpdim\_with\_bias

for j=1:neuron\_hid\_layerI

wib(j,i)=wib(j,i)+eta\*deltai(j)\*ob(i)+beta\*olddelwib(j,i);

olddelwib(j,i)=eta\*deltai(j)\*ob(i)+beta\*olddelwib(j,i);

end

end

wkj = wkj\_tmp;

end

ite(iter)=iter;

error\_avg=error/nvectors;

error\_r(iter)=error\_avg;

end

end

**12. train\_IJK\_net.m**

function [wkj,wji,error\_r,ite] = train\_IJK\_net(data,eta,beta,layer,input,output,itermax,Lowerlimit)

nvectors=length(data);

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

noutdim=output;

%initialize

wkj = normrnd(0,sqrt(2/(input+output)),noutdim,neuron\_hid\_layerJ\_with\_bias);

wji = normrnd(0,sqrt(2/(input+output)),neuron\_hid\_layerJ\_with\_bias,ninpdim\_with\_bias);

wkj\_tmp = zeros(size(wkj));

wji\_tmp = zeros(size(wji));

olddelwkj=zeros(noutdim , neuron\_hid\_layerJ\_with\_bias); % weight of Wkj (J -> K)

olddelwji=zeros(neuron\_hid\_layerJ\_with\_bias , ninpdim\_with\_bias); % weight of Wji (I -> J)

oi = zeros(ninpdim\_with\_bias,1);

oi(ninpdim\_with\_bias) = 1;

sj = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

dk = zeros(noutdim,1); % desired output

iter=0;

error\_avg=10;

% internal variables

deltak = zeros(1,noutdim);

deltaj = zeros(1,neuron\_hid\_layerJ\_with\_bias);

sumback = zeros(1,neuron\_hid\_layerJ\_with\_bias);

while (error\_avg > Lowerlimit) && (iter<itermax)

iter=iter+1;

error=0;

% Forward Computation:

for ivector=1:nvectors

oi=[data(ivector,1:input) 1]';

dk=[data(ivector,input+1:input+output)]';

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); % sigmoid

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); % signmoid

end

error=error+sum(abs(dk-ok)); % abs is absolute each element

% Backward learning:

for k=1:noutdim

deltak(k)=(dk(k)-ok(k))\*deActivation(ok(k)); % gradient term

end

for j=1:neuron\_hid\_layerJ\_with\_bias

for k=1:noutdim

wkj\_tmp(k,j)=wkj(k,j)+eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

olddelwkj(k,j)=eta\*deltak(k)\*oj(j)+beta\*olddelwkj(k,j);

end

end

for j=1:neuron\_hid\_layerJ

sumback(j)=0.0;

for k=1:noutdim

sumback(j)=sumback(j)+deltak(k)\*wkj(k,j);

end

deltaj(j)=deActivation(oj(j))\*sumback(j);

end

for i=1:ninpdim\_with\_bias

for j=1:neuron\_hid\_layerJ

wji(j,i)=wji(j,i)+eta\*deltaj(j)\*oi(i)+beta\*olddelwji(j,i);

olddelwji(j,i)=eta\*deltaj(j)\*oi(i)+beta\*olddelwji(j,i);

end

end

wkj = wkj\_tmp;

end

ite(iter)=iter;

error\_avg=error/nvectors;

error\_r(iter)=error\_avg;

end

end

**13. FeedForward\_ABIJK.m**

function ok = FeedFoward(wba,wib,wji,wkj,oa,input,output,layer)

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

neuron\_hid\_layerI=layer(2);

neuron\_hid\_layerI\_with\_bias=neuron\_hid\_layerI+1;

neuron\_hid\_layerB=layer(3);

neuron\_hid\_layerB\_with\_bias=neuron\_hid\_layerB+1;

noutdim=output;

sb = zeros(neuron\_hid\_layerB\_with\_bias,1);

ob = zeros(neuron\_hid\_layerB\_with\_bias,1);

ob(neuron\_hid\_layerB\_with\_bias) = 1;

si = zeros(ninpdim\_with\_bias,1); % input of hidden layer i

oi = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oi(neuron\_hid\_layerI\_with\_bias) = 1; % output of hidden layer i

sj = zeros(neuron\_hid\_layerI\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

for j=1:neuron\_hid\_layerB

sb(j)=wba(j,:)\*oa;

ob(j)=Activation(sb(j)); %

end

ob(neuron\_hid\_layerB\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerI

si(j)=wib(j,:)\*ob;

oi(j)=Activation(si(j)); %

end

oi(neuron\_hid\_layerI\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); %

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); %

end

end

**14. FeedFoward\_BIJK.m**

function ok = FeedFoward\_BIJK(wib,wji,wkj,oa,input,output,layer)

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

neuron\_hid\_layerI=layer(2);

neuron\_hid\_layerI\_with\_bias=neuron\_hid\_layerI+1;

noutdim=output;

ob = zeros(ninpdim\_with\_bias,1);

ob(ninpdim\_with\_bias) = 1;

si = zeros(ninpdim\_with\_bias,1); % input of hidden layer i

oi = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oi(neuron\_hid\_layerI\_with\_bias) = 1; % output of hidden layer i

sj = zeros(neuron\_hid\_layerI\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

dk = zeros(noutdim,1); % desired output

for j=1:neuron\_hid\_layerI

si(j)=wib(j,:)\*ob;

oi(j)=Activation(si(j)); % sigmoid

end

oi(neuron\_hid\_layerI\_with\_bias)=1.0;

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); % sigmoid

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); % signmoid

end

end

15. FeedFoward\_IJK.m

function ok = FeedFoward\_IJK(wji,wkj,oi,input,output,layer)

ninpdim\_with\_bias=input+1;

neuron\_hid\_layerJ=layer(1);

neuron\_hid\_layerJ\_with\_bias=neuron\_hid\_layerJ+1;

noutdim=output;

sj = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of hidden layer j

oj = zeros(neuron\_hid\_layerJ\_with\_bias,1);

oj(neuron\_hid\_layerJ\_with\_bias) = 1; % output of hidden layer j

sk = zeros(neuron\_hid\_layerJ\_with\_bias,1); % input of output layer k

ok = zeros(noutdim,1); % net output

dk = zeros(noutdim,1); % desired output

for j=1:neuron\_hid\_layerJ

sj(j)=wji(j,:)\*oi;

oj(j)=Activation(sj(j)); % sigmoid

end

oj(neuron\_hid\_layerJ\_with\_bias)=1.0;

for k=1:noutdim

sk(k)=wkj(k,:)\*oj;

ok(k)=Activation(sk(k)); % signmoid

end

end

**16. Nerual Network Tool box 第一題 MLP\_P1.m**

clear all;

N=97;

i = 0:1:96;

theta = i.\*pi/16;

r = 6.5\*(104-i)/104;

data(1:97,1) = r.\*sin(theta);

data(1:97,2) = r.\*cos(theta);

data(1:97,3) = 1;

data(1:97,4) = 0;

data(98:98+96,1) = -1\*r.\*sin(theta);

data(98:98+96,2) = -1\*r.\*cos(theta);

data(98:98+96,3) = 0;

data(98:98+96,4) = 1;

x\_input = [data(:,1) data(:,2)];

y\_output = [data(:,3) data(:,4)];

x\_input = x\_input.';

y\_output = y\_output.';

net = feedforwardnet([12 12 12]);

net.trainParam.lr = 0.01;

net.trainParam.epochs = 10000;

net.trainParam.goal = 0.0001;

net.divideFcn= 'dividerand';

net.divideParam.trainRatio= 1;

net.divideParam.valRatio= 0;

net.divideParam.testRatio=0;

net = train(net,x\_input,y\_output);

view(net);

title\_text = sprintf('P1: 2X12X12X12X2\n lr = %d, epochs = %f', net.trainParam.lr, net.trainParam.epochs);

file\_text = sprintf('P1\_2X12X12X12X2\_lr\_%d\_epochs\_%f', net.trainParam.lr, net.trainParam.epochs);

fig\_decision = figure(1);

hold on;

for n=1:1:97

plot(data(n,1), data(n,2),'r o');

end

for n=98:1:194

plot(data(n,1), data(n,2),'k s');

end

for ix=-7\*5:1:7\*5

for iy=-10\*5:1:10\*5

dx=0.2\*(ix-1);

dy=0.2\*(iy-1);

final\_out = net([dx dy].');

% Real output

if final\_out(1)<0.5

plot(dx,dy, 'k .');

elseif final\_out(1)>0.5

plot(dx,dy, 'r .');

end

end

end

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));

**17. Nerual Network Tool box 第二題 MLP\_P2.m**

clear all;

N=250;

theta1 = linspace(-180,180, N)\*pi/360;

r = 8

data(1:N,1) = -5 + r\*sin(theta1)+randn(1,N);

data(1:N,2) = r\*cos(theta1)+randn(1,N);

data(1:N,3) = 1;

data(1:97,4) = 0;

data(N+1:2\*N,1) = 5 + r\*sin(theta1)+randn(1,N);

data(N+1:2\*N,2) = -r\*cos(theta1)+randn(1,N);

data(N+1:2\*N,3) = 0;

data(N+1:2\*N,4) = 1;

x\_input = [data(:,1) data(:,2)];

y\_output = [data(:,3) data(:,4)];

x\_input = x\_input.';

y\_output = y\_output.';

net = feedforwardnet([3 3]);

net.trainParam.lr = 0.2;

net.trainParam.epochs = 10000;

net.trainParam.goal = 0.001;

net.divideFcn= 'dividerand';

net.divideParam.trainRatio= 1;

net.divideParam.valRatio= 0;

net.divideParam.testRatio=0;

net = train(net,x\_input,y\_output);

view(net);

title\_text = sprintf('P2: 2X3X3X2\n lr = %d, epochs = %f', net.trainParam.lr, net.trainParam.epochs);

file\_text = sprintf('P2\_2X3X3X2\_lr\_%d\_epochs\_%f', net.trainParam.lr, net.trainParam.epochs);

fig\_decision = figure(1);

hold on;

for n=1:1:N

plot(data(n,1), data(n,2),'r o');

end

for n=N+1:1:N\*2

plot(data(n,1), data(n,2),'k s');

end

for ix=-30:1:31

for iy=-30:1:31

dx=0.5\*(ix-1);

dy=0.5\*(iy-1);

final\_out = net([dx dy].');

% Real output

if final\_out(1)<0.5

plot(dx,dy, 'k .');

elseif final\_out(1)>0.5

plot(dx,dy, 'r .');

end

end

end

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));

**18. Nerual Network Tool box 第三題 MLP\_P3.m**

clear all;

mu1 = [0 0];

sigma1 = [1 0; 0 1];

rng default; % For reproducibility

R = mvnrnd(mu1,sigma1,150);

N = 150;

data(1:N,1) = R(:,1);

data(1:N,2) = R(:,2);

data(1:N,3) = 1;

data(1:N,4) = 0;

data(1:N,5) = 0;

data(1:N,6) = 0;

mu2 = [14 0];

sigma2 = [1 0; 0 4];

R = mvnrnd(mu2,sigma2,150);

data(N+1:2\*N,1) = R(:,1);

data(N+1:2\*N,2) = R(:,2);

data(N+1:2\*N,3) = 0;

data(N+1:2\*N,4) = 1;

data(N+1:2\*N,5) = 0;

data(N+1:2\*N,6) = 0;

mu3 = [7 14];

sigma3 = [4 0; 0 1];

R = mvnrnd(mu3,sigma3,150);

data(2\*N+1:3\*N,1) = R(:,1);

data(2\*N+1:3\*N,2) = R(:,2);

data(2\*N+1:3\*N,3) = 0;

data(2\*N+1:3\*N,4) = 0;

data(2\*N+1:3\*N,5) = 1;

data(2\*N+1:3\*N,6) = 0;

mu4 = [7 7];

sigma4 = [1 0; 0 1];

R = mvnrnd(mu4,sigma4,150);

data(3\*N+1:4\*N,1) = R(:,1);

data(3\*N+1:4\*N,2) = R(:,2);

data(3\*N+1:4\*N,3) = 0;

data(3\*N+1:4\*N,4) = 0;

data(3\*N+1:4\*N,5) = 0;

data(3\*N+1:4\*N,6) = 1;

x\_input = [data(:,1) data(:,2)];

y\_output = [data(:,3) data(:,4) data(:,5) data(:,6)];

x\_input = x\_input.';

y\_output = y\_output.';

net = feedforwardnet([3]);

net.trainParam.lr = 0.2;

net.trainParam.epochs = 10000;

net.trainParam.goal = 0.001;

net.divideFcn= 'dividerand';

net.divideParam.trainRatio= 1;

net.divideParam.valRatio= 0;

net.divideParam.testRatio=0;

net = train(net,x\_input,y\_output);

view(net);

title\_text = sprintf('P3: 2X3X2\n lr = %d, epochs = %f', net.trainParam.lr, net.trainParam.epochs);

file\_text = sprintf('P3\_2X3X2\_lr\_%d\_epochs\_%f', net.trainParam.lr, net.trainParam.epochs);

fig\_decision = figure(1);

hold on;

plot(data(1:N,1),data(1:N,2),'ro');

plot(data(N+1:2\*N,1),data(N+1:2\*N,2),'bo');

plot(data(2\*N+1:3\*N,1),data(2\*N+1:3\*N,2),'go');

plot(data(3\*N+1:4\*N,1),data(3\*N+1:4\*N,2),'ko');

for ix=-5\*4:1:20\*4

for iy=-5\*4:1:20\*4

dx=0.25\*(ix-1);

dy=0.25\*(iy-1);

final\_out = net([dx dy].');

% Real output

[M,I] = max(final\_out);

if (I == 1)

plot(dx,dy, 'r .');

elseif (I == 2)

plot(dx,dy, 'b .');

elseif (I == 3)

plot(dx,dy, 'g .');

elseif (I == 4)

plot(dx,dy, 'k .');

end

end

end

title(title\_text);

xlabel('iteration');

ylabel('error');

saveas(fig\_decision,strcat(file\_text,'\_decision.jpg'));

saveas(fig\_decision,strcat(file\_text,'\_decision.fig'));