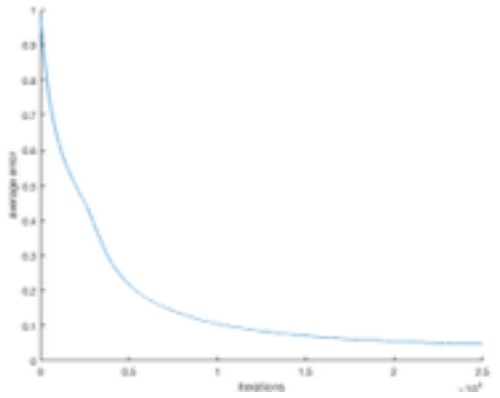
圖形識別概論Project #2(a)

0516218軒轅照雯

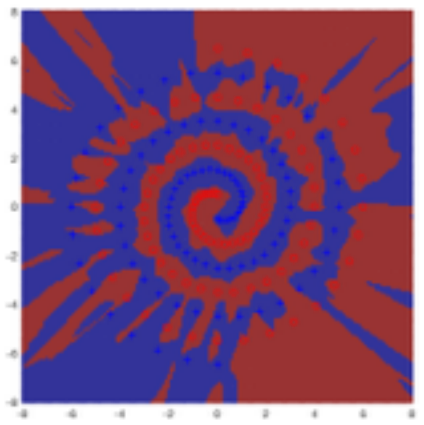
1. Two spiral problem

(1)parameters: number of hidden layers: 1
number of hidden nodes: 100
learning rate parameter: 0.01
stop criterion: average error < 0.02 or iterations > 25000

(2)average error vs. iteration



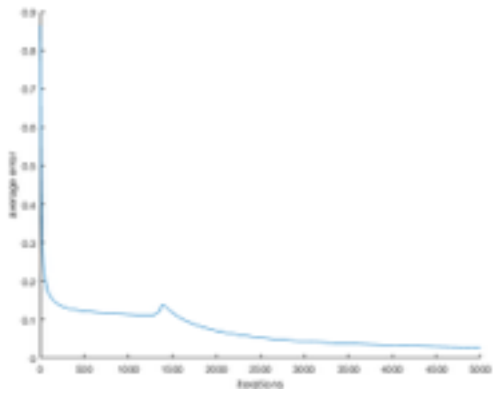
(3)decision region



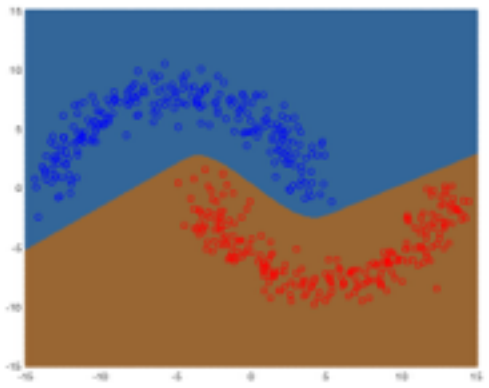
2. Double-moon problem

(1)parameters: number of hidden layers: 1
number of hidden nodes: 3
learning rate parameter: 0.01
stop criterion: average error< 0.002 or iterations > 5000

(2)average error vs. iteration



(3)decision region

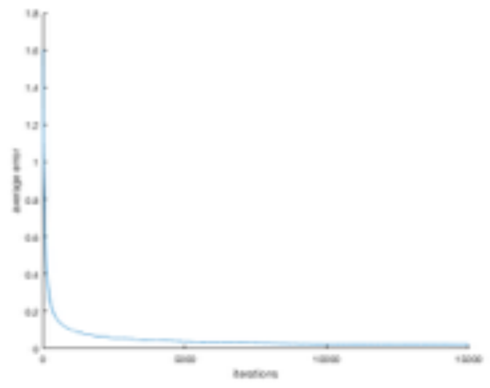


3. Gaussian distribution

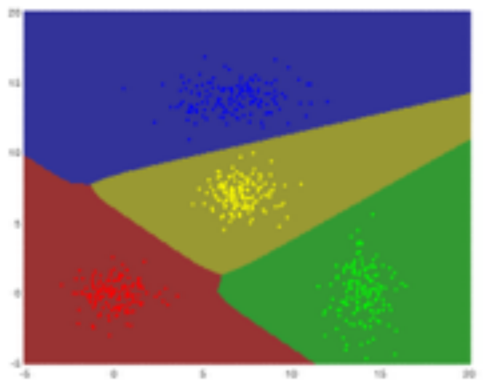
(a)activation function: sigmoidal function

(1)parameters: number of hidden layers: 1
number of hidden nodes: 3
learning rate parameter: 0.01
stop criterion: average error < 0.002 or iterations > 15000

(2)average error vs. iteration



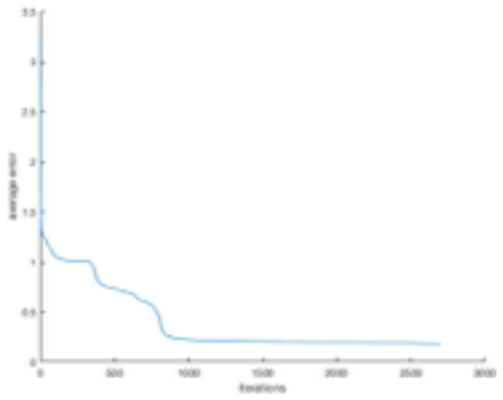
(3)decision region



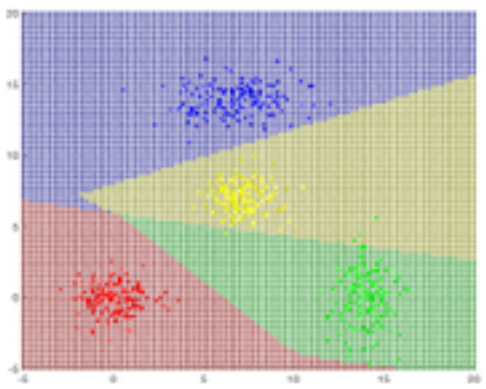
(b)activation function: ReLu function

(1)parameters: number of hidden layers: 1
number of hidden nodes: 3
learning rate parameter: 0.001
stop criterion: average error < 0.002 or iterations > 2700

(2)average error vs. iteration



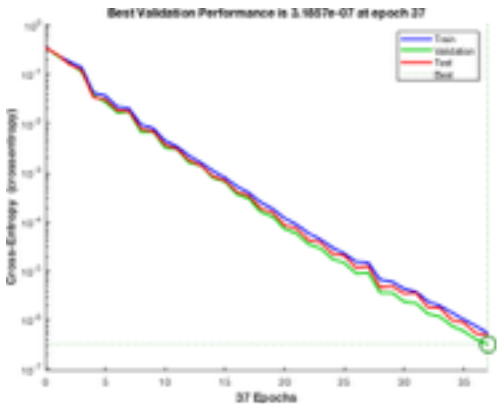
(3)decision region



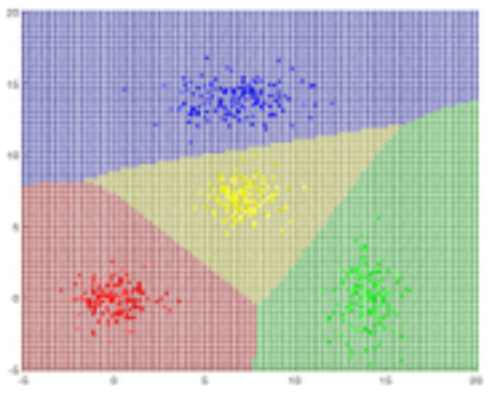
(c)MATLAB neural networks toolbox

(1)parameters: number of hidden layers: 1
number of hidden nodes: 3
learning rate parameter: 0.01
training data: 70%
validation data: 10%
testing data: 20%
stop criterion: average error = 0 or iterations > 1000 or
gradient< 1.00e-6 or validation checks > 6

(2)cross-entropy vs. iteration



(3)decision region



(d)comparison

-stop criterion: since MATLAB neural networks toolbox has lots of stop criterion, it terminate because gradient < 1.00e-6, while the others terminate due to iterations > threshold.

-convergence speed: (c)>(b)>(a)

Using ReLu as activation function will result in faster convergence than using sigmoidal function, but not always converge to 0.

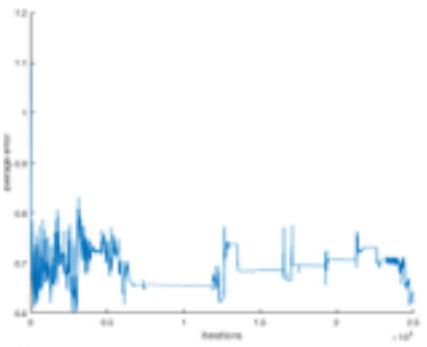
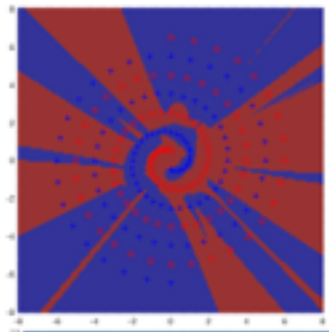
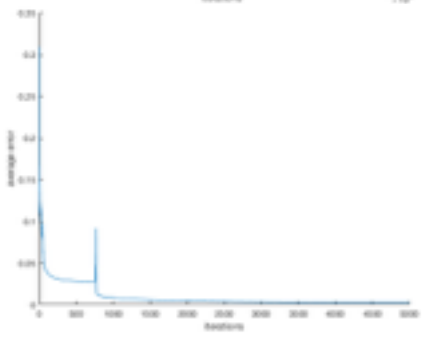
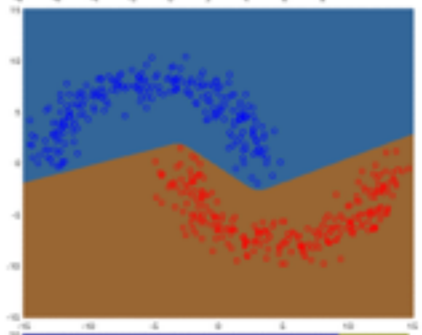
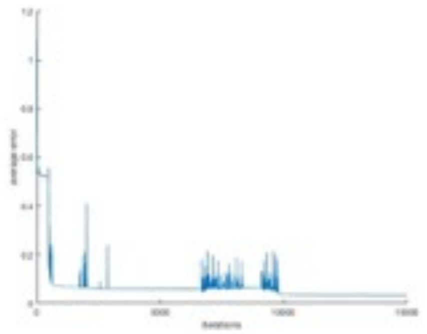
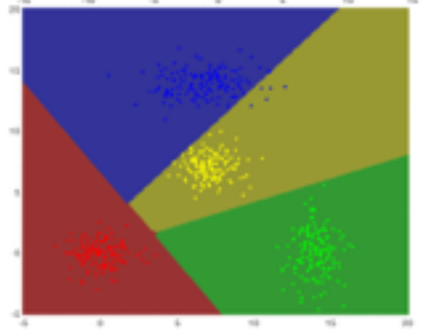
4. Discussion

(1)How to determine the hidden node number in each problem?

I read lots of references, but I couldn't find a specific method to determine the number of hidden nodes, since there are so many factors should be taken into consideration. However, in this project, I simply decided it **based on the number of input and output nodes**.

There are some general advice such as the number of hidden nodes in each layer should be somewhere between the size of the input and output layer, potentially the mean, which can be applied to problem 3; another one is that the number of hidden nodes shouldn't need to exceed twice the number of input nodes, which can be applied to problem 2. Both of the problems can be well classified by 3 hidden nodes, that is, they can be well classified by 3 lines, which is obvious from the distribution of the data. For problem 1, on account of the complexity, I decided by others' experiences and tried several potential candidates(hidden nodes=50, 60, 77, 100), then chose the one with the best performance(100).

(2)adding momentum terms with coefficient=0.3 to each problem

From the results of the three problems after adding momentum terms, we can see that only the double moon problem performs better(converges faster).The momentum term is used to prevent the system from converging to a local minimum or saddle point and increase the speed of convergence of the system. However, setting the momentum coefficient too high can create a risk of overshooting the minimum, which can cause the system to become unstable; if too low, can't reliably avoid local minima, and also can slow the training of the system.So the reason why the other problems doesn't perform better may due to the setting of momentum coefficient.

(3)use high order inputs

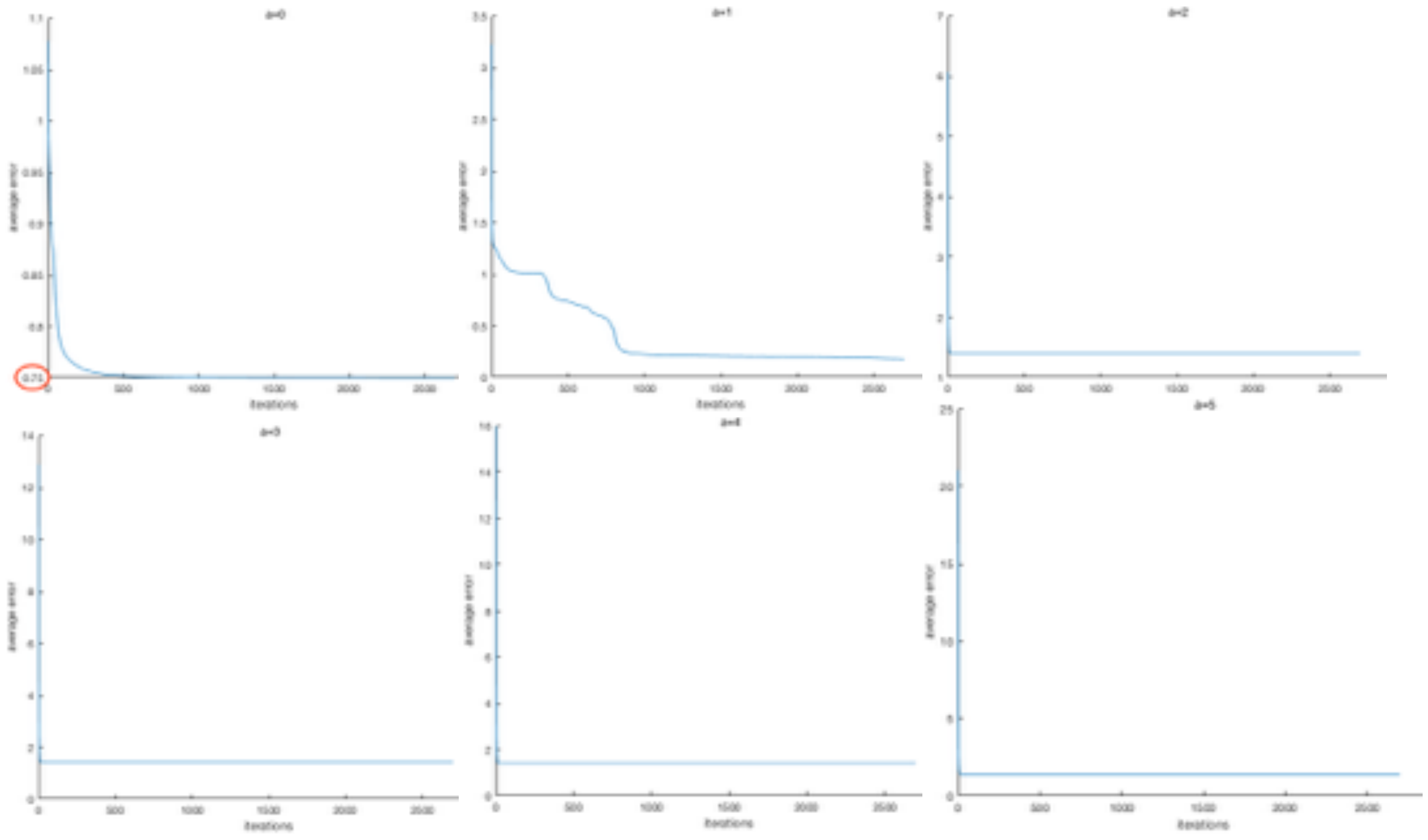
I used second order inputs for the two spiral problem in order to get a higher accuracy, since it's obviously that the problem may not be well classified by only using the first order inputs.

(4)changing learning rate parameter

I chose a lower learning rate parameter only for problem 3(b), since the dying Relu problem is likely to occur when learning rate is too high.

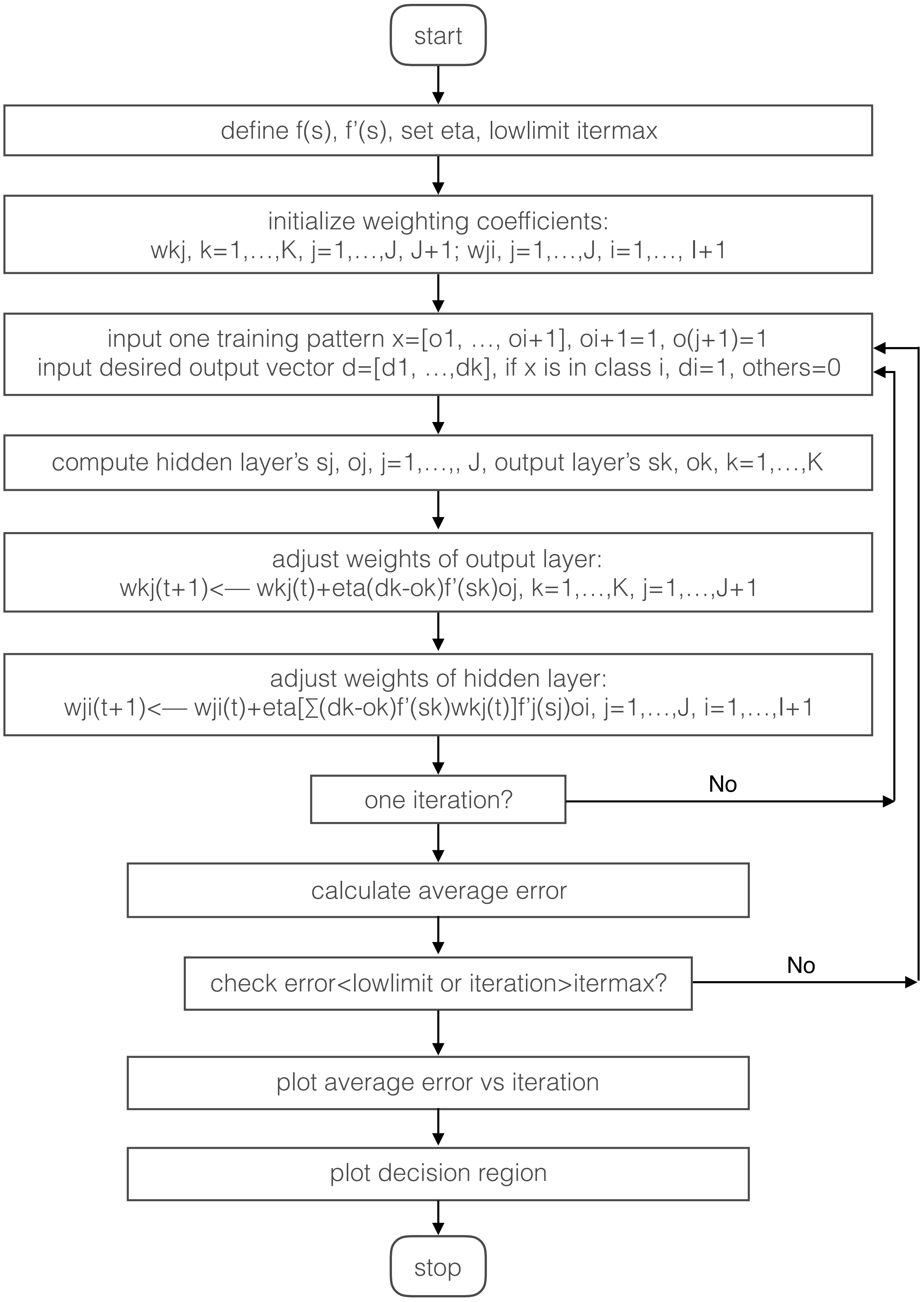
(5)changing initial weights

Although using ReLu as activation may converge faster, it may not always converge to 0(global minimum), due to the initialization of weights. So, I tried 6 kinds of initial weights: wkj=randn(K, J)+a, wji=randn(J-1, I)+a, a=0~5, chose a=1, which result in the lowest average error.



5. Flowchart of programming

My programs are mostly based on the appendix k of the reference book with some modification on input nodes, output nodes, hidden nodes, activation function and momentum term for different problems.


start
define f(s), f'(s), set eta, lowlimit itermax
initialize weighting coefficients:
wkj, k=1,…,K, j=1,…,J, J+1; wji, j=1,…,J, i=1,…, I+1
input one training pattern x=[o1, …, oi+1], oi+1=1, o(j+1)=1
input desired output vector d=[d1, …,dk], if x is in class i, di=1, others=0
compute hidden layer's sj, oj, j=1,…,, J, output layer's sk, ok, k=1,…,K
adjust weights of output layer:
wkj(t+1)<— wkj(t)+eta(dk-ok)f'(sk)oj, k=1,…,K, j=1,…,J+1
adjust weights of hidden layer:
wji(t+1)<— wji(t)+eta[∑(dk-ok)f'(sk)wkj(t)]f'j(sj)oi, j=1,…,J, i=1,…,I+1
one iteration?
No
calculate average error
check error<lowlimit or iteration>itermax?
No
plot average error vs iteration
plot decision region
stop

6. Reference

http://playground.tensorflow.org
http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html
https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network-I-will-be-using-a-three-layer-model
https://www.hindawi.com/journals/acisc/2009/721370/
https://www.tinymind.com/learn/terms/relu
https://www.willamette.edu/~gorr/classes/cs449/momrate.html

7. Matlab programs

(1)

```
qi=0:96;
r=6.5.*(104-qi)./104;
q=pi/16.*qi;
x1 = r.*sin(q);
y1 = r.*cos(q);
x2 = -r.*sin(q);
y2 = -r.*cos(q);
data=[x1' y1' (x1.*x1)' (y1.*y1)' (x1.*y1)' ones(97,1) zeros(97,1);
    x2' y2' (x2.*x2)' (y2.*y2)' (x2.*y2)' zeros(97,1) ones(97,1);];
r=randperm(97*2);
data=data(r, :);
I=5+1;  %input+const
J=100+1;  %hidden+const
K=2;    %class
n=97*2;
wkj=randn(K, J);
wkj_temp=zeros(size(wkj));
wji=randn(J-1, I);
old_dwkj=zeros(size(wkj));
old_dwji=zeros(size(wji));
oi=[zeros(I-1, 1);1];
sj=zeros(J-1,1);
oj=[sj;1];
sk=zeros(K,1);
ok=zeros(K,1);
dk=zeros(K,1);
lowlimit=0.02;
itermax=25000;
iter=0;    %iteration
%eta=0.7;beta=0.3;  %add momentum term
eta=0.01;beta=0.0;
erroravg=10;
deltak=zeros(1, K);
sumback=zeros(1, J-1);
deltaj=zeros(1, J-1);
while (erroravg>lowlimit) && (iter<itermax)
   iter=iter+1;
   error=0;
   %forward computation
   for i=1:n
      oi=[data(i,1:5) 1]';
      dk=[data(i,6:7)]';
      for j=1:J-1
         sj(j)=wji(j, :)*oi;
         oj(j)=1/(1+exp(-sj(j)));   %sigmoidal
      end
      oj(J)=1.0;   %const
      for k=1:K
         sk(k)=wkj(k, :)*oj;
```

```
            ok(k)=1/(1+exp(-sk(k)));   %sigmoidal
        end
        error=error+sum(abs(dk-ok));
    %backward learning
        for k=1:K
        deltak(k)=(dk(k)-ok(k))*ok(k)*(1.0-ok(k));   %ok'(k)=ok(k)(1-ok(k))
        end
        for j=1:J
            for k=1:K
                wkj_temp(k,j)=wkj(k,j)+eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
                old_dwkj(k, j)=eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
            end
        end
        for j=1:J-1
            sumback(j)=0.0;
            for k=1:K
                sumback(j)=sumback(j)+deltak(k)*wkj(k,j);
            end
            deltaj(j)=oj(j)*(1.0-oj(j))*sumback(j);

        end
        for i=1:I
            for j=1:J-1
                wji(j,i)=wji(j,i)+eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
                old_dwji(j,i)=eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
            end
        end
        wkj=wkj_temp;
    end
    ite(iter)=iter;
    erroravg=error/n;
    error_r(iter)=erroravg;
end
figure;
hold on;
plot(ite, error_r);
xlabel('iterations');
ylabel('average error');
figure;
hold on;
axis square;
%plot the decision regions
for ix=1:1:161
    for iy=1:1:161
        dx=-8+0.1*(ix-1);
        dy=-8+0.1*(iy-1);
        oi=[dx dy dx*dx dy*dy dx*dy 1]';
        for j=1:J-1
            sj(j)=wji(j,:)*oi;
            oj(j)=1/(1+exp(-sj(j)));
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k,:)*oj;
            ok(k)=1/(1+exp(-sk(k)));
        end
        if ok(1, 1)> 0.5                %sigmoidal
            plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
        else
            plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
        end
    end
end
plot(x1, y1, 'ro');
plot(x2, y2, 'b+');
```

(2)

```
N=250;
theta1 = linspace(-180,180, N)*pi/360;
r=8;
x1 = -5 + r*sin(theta1)+randn(1,N);
y1 = r*cos(theta1)+randn(1,N);
x2 = 5 + r*sin(theta1)+randn(1,N);
y2 = -r*cos(theta1)+randn(1,N);
data=[x1.', y1.', ones(250, 1), zeros(250, 1);
      x2.', y2.', zeros(250, 1), ones(250, 1)];
r=randperm(500);
data=data(r, :);
I=2+1;  %input+const
J=3+1;  %hidden+const
K=2;    %class
n=500;
wkj=randn(K, J);
wkj_temp=zeros(size(wkj));
wji=randn(J-1, I);
old_dwkj=zeros(size(wkj));
old_dwji=zeros(size(wji));
oi=[0 0 1]';
sj=[0 0 0]';
oj=[0 0 0 1]';
sk=[0 0]';
ok=[0 0]';
dk=[0 0]';
lowlimit=0.002;
itermax=5000;
iter=0;    %iteration
%eta=0.7;beta=0.3;  %add momentum term
eta=0.01;beta=0.0;
erroravg=10;
%internal variables
deltak=zeros(1, K);
sumback=zeros(1, J-1);
deltaj=zeros(1, J-1);
while (erroravg>lowlimit) && (iter<itermax)
    iter=iter+1;
    error=0;
    %forward computation
    for i=1:n
        oi=[data(i,1:2) 1]';
        dk=[data(i,3:4)]';
        for j=1:J-1
            sj(j)=wji(j, :)*oi;
            oj(j)=1/(1+exp(-sj(j)));
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k, :)*oj;
            ok(k)=1/(1+exp(-sk(k)));
        end
        error=error+sum(abs(dk-ok));
    %backward learning
        for k=1:K
        deltak(k)=(dk(k)-ok(k))*ok(k)*(1.0-ok(k));
        end
        for j=1:J
            for k=1:K
                wkj_temp(k,j)=wkj(k,j)+eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
                old_dwkj(k, j)=eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
            end
        end
        for j=1:J-1
            sumback(j)=0.0;
            for k=1:K
                sumback(j)=sumback(j)+deltak(k)*wkj(k,j);
            end
            deltaj(j)=oj(j)*(1.0-oj(j))*sumback(j);
        end
        for i=1:I
            for j=1:J-1
                wji(j,i)=wji(j,i)+eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
                old_dwji(j,i)=eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
            end
        end
        wkj=wkj_temp;
    end
    ite(iter)=iter;
    erroravg=error/n;
    error_r(iter)=erroravg;
end
figure;
hold on;
plot(ite, error_r);
xlabel('iterations');
ylabel('average error');
figure;
hold on;
axis([-15 15 -15 15]);
%plot the decision regions
for ix=1:1:301
    for iy=1:1:301
        dx=-15+0.1*(ix-1);
        dy=-15+0.1*(iy-1);
        oi=[dx dy 1]';
        for j=1:J-1
            sj(j)=wji(j,:)*oi;
            oj(j)=1/(1+exp(-sj(j)));
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k,:)*oj;
            ok(k)=1/(1+exp(-sk(k)));
        end
        if ok(1, 1)> 0.5
            plot(dx, dy, '.', 'color', [0.2, 0.4, 0.6]);
        else
            plot(dx, dy, '.', 'color', [0.6, 0.4, 0.2]);
            end
        end
end
plot(x1,y1,'bo');
plot(x2,y2,'rs');
```

(3)a

```
m1=[0 0];
a1=[1 0;0 1];
m2=[14 0];
a2=[1 0;0 4];
m3=[7 14];
a3=[4 0;0 1];
m4=[7 7];
a4=[1 0;0 1];
rng default
c1 = mvnrnd(m1,a1,150);
c2 = mvnrnd(m2,a2,150);
c3 = mvnrnd(m3,a3,150);
c4 = mvnrnd(m4,a4,150);
data=[c1 ones(150, 1) zeros(150, 3);
     c2 zeros(150, 1) ones(150, 1) zeros(150, 2);
     c3 zeros(150, 2) ones(150, 1) zeros(150, 1);
     c4 zeros(150, 3) ones(150, 1)];
r=randperm(150*4);
data=data(r, :);
I=2+1;  %input+const
J=3+1;  %hidden+const
K=4;    %class
n=600;
wkj=randn(K, J);
wkj_temp=zeros(size(wkj));
wji=randn(J-1, I);
old_dwkj=zeros(size(wkj));
old_dwji=zeros(size(wji));
oi=[zeros(I-1, 1);1];
sj=zeros(J-1,1);
oj=[sj;1];
sk=zeros(K,1);
ok=zeros(K,1);
dk=zeros(K,1);
lowlimit=0.002;
itermax=15000;
iter=0;
%eta=0.7;beta=0.3;   %add momentum term
eta=0.01;beta=0.0;
erroravg=10;
deltak=zeros(1, K);
sumback=zeros(1, J-1);
deltaj=zeros(1, J-1);
while (erroravg>lowlimit) && (iter<itermax)
    iter=iter+1;
    error=0;
    %forward computation
    for i=1:n
        oi=[data(i,1:2) 1]';
        dk=[data(i,3:6)]';
        for j=1:J-1
            sj(j)=wji(j, :)*oi;
            oj(j)=1/(1+exp(-sj(j)));    %sigmoidal
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k, :)*oj;
            ok(k)=1/(1+exp(-sk(k)));
        end
        error=error+sum(abs(dk-ok));
    %backward learning
        for k=1:K
        deltak(k)=(dk(k)-ok(k))*ok(k)*(1.0-ok(k));
        end
        for j=1:J
            for k=1:K
                wkj_temp(k,j)=wkj(k,j)+eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
                old_dwkj(k, j)=eta*deltak(k)*oj(j)+beta*old_dwkj(k,j);
            end
        end
        for j=1:J-1
            sumback(j)=0.0;
            for k=1:K
                sumback(j)=sumback(j)+deltak(k)*wkj(k,j);
            end
            deltaj(j)=oj(j)*(1.0-oj(j))*sumback(j);
        end
        for i=1:I
            for j=1:J-1
                wji(j,i)=wji(j,i)+eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
                old_dwji(j,i)=eta*deltaj(j)*oi(i)+beta*old_dwji(j,i);
            end
        end
        wkj=wkj_temp;
    end
    ite(iter)=iter;
    erroravg=error/n;
    error_r(iter)=erroravg;
end
figure;
hold on;
plot(ite, error_r);
xlabel('iterations');
ylabel('average error');
figure;
hold on;
%plot the decision regions
for ix=1:1:251
    for iy=1:1:251
        dx=-5+0.1*(ix-1);
        dy=-5+0.1*(iy-1);
        oi=[dx dy 1]';
        for j=1:J-1
            sj(j)=wji(j,:)*oi;
            oj(j)=1/(1+exp(-sj(j)));
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k,:)*oj;
            ok(k)=1/(1+exp(-sk(k)));
        end
        if ok(1, 1)> 0.5
            plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
        elseif ok(2,1) > 0.5
            plot(dx, dy, '.', 'color', [0.2, 0.6, 0.2]);
        elseif ok(3,1) > 0.5
            plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
        elseif ok(4,1) > 0.5
            plot(dx, dy, '.', 'color', [0.6, 0.6, 0.2]);
        else
            [m,index]=max(ok);
            switch index
              case 1
                 plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
              case 2
                 plot(dx, dy, '.', 'color', [0.2, 0.6, 0.2]);
              case 3
                 plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
              case 4
                 plot(dx, dy, '.', 'color', [0.6, 0.6, 0.2]);
            end
        end
    end
end
plot(c1(:,1),c1(:,2),'r.');
plot(c2(:,1),c2(:,2),'g.');
plot(c3(:,1),c3(:,2),'b.');
plot(c4(:,1),c4(:,2),'y.');
```

(3)b

```
m1=[0 0];
a1=[1 0;0 1];
m2=[14 0];
a2=[1 0;0 4];
m3=[7 14];
a3=[4 0;0 1];
m4=[7 7];
a4=[1 0;0 1];
rng default  % For reproducibility
c1 = mvnrnd(m1,a1,150);
c2 = mvnrnd(m2,a2,150);
c3 = mvnrnd(m3,a3,150);
c4 = mvnrnd(m4,a4,150);
data=[c1 ones(150, 1) zeros(150, 3);
      c2 zeros(150, 1) ones(150, 1) zeros(150, 2);
      c3 zeros(150, 2) ones(150, 1) zeros(150, 1);
      c4 zeros(150, 3) ones(150, 1)];
r=randperm(150*4);
data=data(r, :);
I=2+1;  %input+const
J=3+1;  %hidden+const
K=4;    %class
n=600;
wkj_temp=zeros(K, J);
bestwkj=zeros(K, J);
bestwji=zeros(J-1, I);
oi=[zeros(I-1, 1);1];
sj=zeros(J-1,1);
oj=[sj;1];
sk=zeros(K,1);
ok=zeros(K,1);
dk=zeros(K,1);
lowlimit=0.002;
itermax=2700;
iter=0;    %iteration
eta=0.001;  %no momentum
besterror=10;
```

```
erroravg=10;
deltak=zeros(1, K);
sumback=zeros(1, J-1);
deltaj=zeros(1, J-1);
for a=0:5
    wkj=randn(K, J)+a;
    wji=randn(J-1, I)+a;
    iter=0;
while (erroravg>lowlimit) && (iter<itermax)
    iter=iter+1;
    error=0;
    %forward computation
    for i=1:n
        oi=[data(i,1:2) 1]';
        dk=[data(i,3:6)]';
        for j=1:J-1
            sj(j)=wji(j, :)*oi;
            oj(j)=max(0, sj(j));      %ReLu
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k, :)*oj;
            ok(k)=max(0, sk(k));
        end
        error=error+sum(abs(dk-ok));
    %backward learning
        for k=1:K
           deltak(k)=(dk(k)-ok(k))*(sk(k)>0);   %ok'(k)=0, 1;
        end
        for j=1:J
            for k=1:K
                wkj_temp(k,j)=wkj(k,j)+eta*deltak(k)*oj(j);
            end
        end
        for j=1:J-1
            sumback(j)=0.0;
            for k=1:K
                sumback(j)=sumback(j)+deltak(k)*wkj(k,j);
            end
            deltaj(j)=sumback(j)*(sj(j)>0);   %oj'(j)=0,1
        end
        for i=1:I
            for j=1:J-1
                wji(j,i)=wji(j,i)+eta*deltaj(j)*oi(i);
            end
        end
        wkj=wkj_temp;
    end
    ite(iter)=iter;
    erroravg=error/n;
    error_r(iter)=erroravg;
end
 figure;
 hold on;
 plot(ite, error_r);
 xlabel('iterations');
 ylabel('average error');
 if erroravg<besterror
    bestwkj=wkj;
    bestwji=wji;
    besterror=erroravg;
    besta=a;
 end
end
figure;
hold on;
wkj=bestwkj;
wji=bestwji;
for ix=1:1:101
    for iy=1:1:101
        dx=-5+0.25*(ix-1);
        dy=-5+0.25*(iy-1);
        oi=[dx dy 1]';
        for j=1:J-1
            sj(j)=wji(j,:)*oi;
            oj(j)=max(0, sj(j));
        end
        oj(J)=1.0;
        for k=1:K
            sk(k)=wkj(k,:)*oj;
            ok(k)=max(0, sk(k));
        end
        [m,index]=max(ok);
         switch index
           case 1
              plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
           case 2
              plot(dx, dy, '.', 'color', [0.2, 0.6, 0.2]);
           case 3
              plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
           case 4
              plot(dx, dy, '.', 'color', [0.6, 0.6, 0.2]);
         end
    end
end
plot(c1(:,1),c1(:,2),'r.');
plot(c2(:,1),c2(:,2),'g.');
plot(c3(:,1),c3(:,2),'b.');
plot(c4(:,1),c4(:,2),'y.');
```

(3)c

```
m1=[0 0];
a1=[1 0;0 1];
m2=[14 0];
a2=[1 0;0 4];
m3=[7 14];
a3=[4 0;0 1];
m4=[7 7];
a4=[1 0;0 1];
rng default
c1 = mvnrnd(m1,a1,150);
c2 = mvnrnd(m2,a2,150);
c3 = mvnrnd(m3,a3,150);
c4 = mvnrnd(m4,a4,150);
input=[c1;c2;c3;c4];
target=[ones(150, 1) zeros(150, 3);
        zeros(150, 1) ones(150, 1) zeros(150, 2);
        zeros(150, 2) ones(150, 1) zeros(150, 1);
        zeros(150, 3) ones(150, 1)];
x = input';
```

```
t = target';
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.
hiddenLayerSize = 3;
net = patternnet(hiddenLayerSize, trainFcn);
net.trainParam.lr=0.01;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 20/100;
[net,tr] = train(net,x,t);
y = net(x);
e = gsubtract(t,y);
view(net);
figure, plotperform(tr);
figure, plottrainstate(tr);
figure, ploterrhist(e);
figure, plotconfusion(t,y);
figure, plotroc(t,y);
figure;
hold on;
%plot the decision regions
for ix=1:1:101
   for iy=1:1:101
      dx=-5+0.25*(ix-1);
      dy=-5+0.25*(iy-1);
      pred=sim(net,[dx;dy]);
      if pred(1, 1)==1
         plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
      elseif pred(2, 1)==1
         plot(dx, dy, '.', 'color', [0.2, 0.6, 0.2]);
      elseif pred(3, 1)==1
         plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
      elseif pred(4, 1)==1
         plot(dx, dy, '.', 'color', [0.6, 0.6, 0.2]);
      else
         [m,index]=max(pred);
         switch index
           case 1
              plot(dx, dy, '.', 'color', [0.6, 0.2, 0.2]);
           case 2
              plot(dx, dy, '.', 'color', [0.2, 0.6, 0.2]);
           case 3
              plot(dx, dy, '.', 'color', [0.2, 0.2, 0.6]);
           case 4
              plot(dx, dy, '.', 'color', [0.6, 0.6, 0.2]);
         end
      end
   end
end
plot(c1(:,1),c1(:,2),'r.');
plot(c2(:,1),c2(:,2),'g.');
plot(c3(:,1),c3(:,2),'b.');
plot(c4(:,1),c4(:,2),'y.');
```