Problem 1

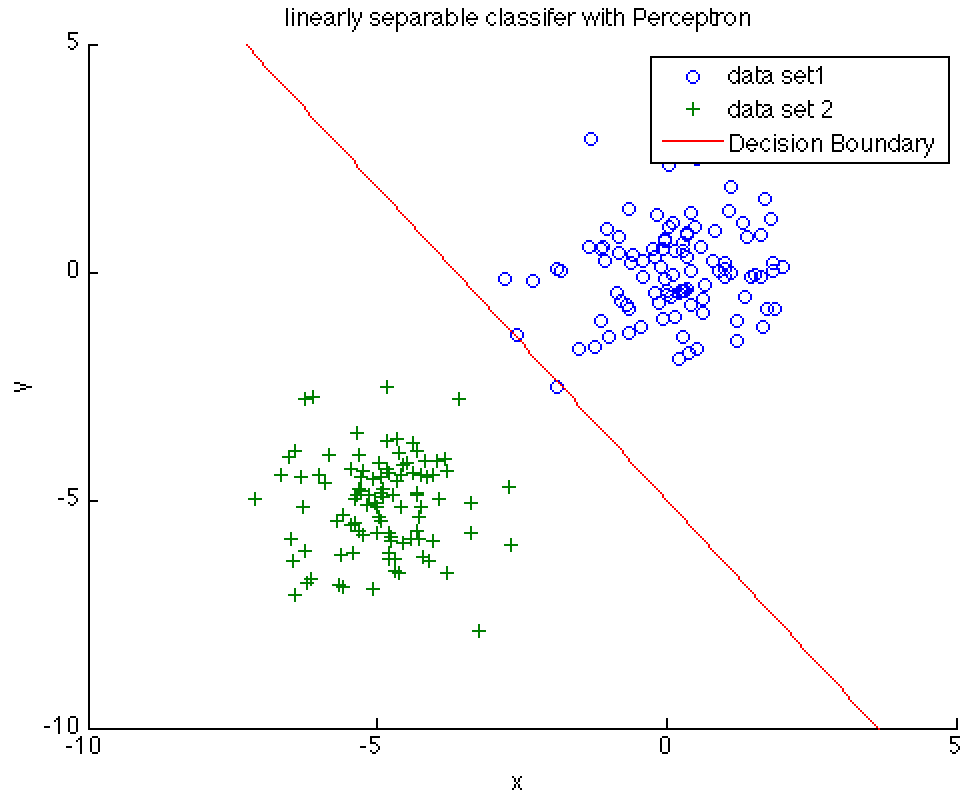
Table of Contents

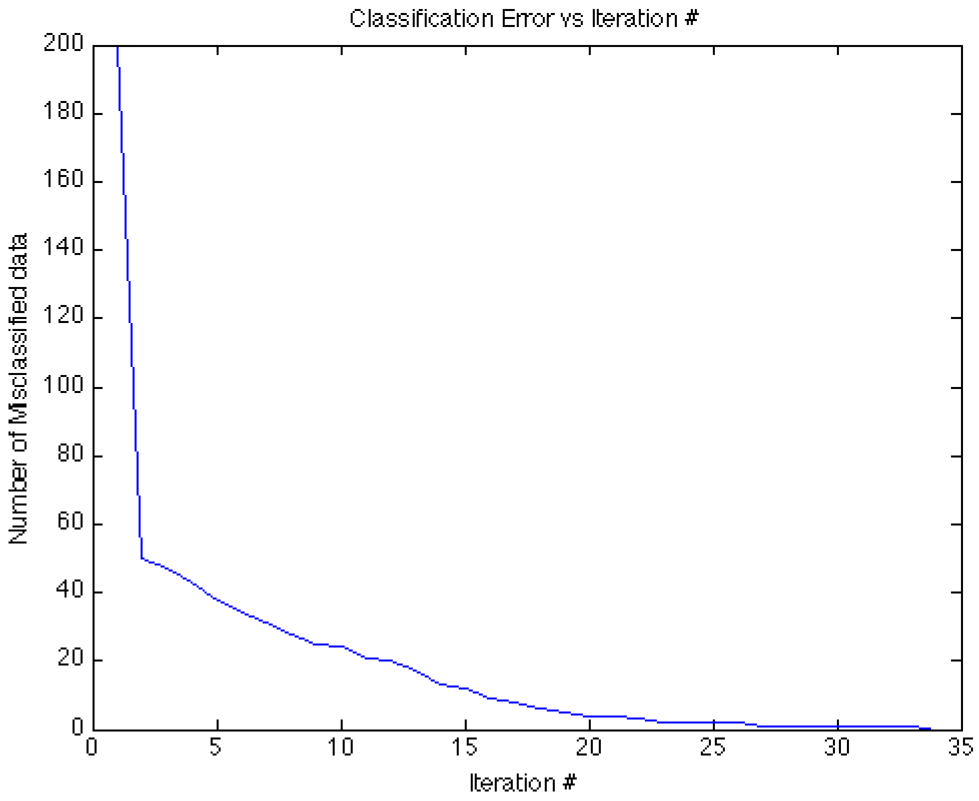
A. Linearly Separable Dataset

The threshold value of the classification (\theta) is set as 5. Fig. 2 shows that the number of misclassified data points decrease exponentially with iteration number. As shown by Fig.1, the decision boundary succesfully separates two linearly-separable dataset with 100% accuracy whtin ~20-50 trials

```
set(0,'DefaultAxesFontSize', 12)
clc; clear
figNum = 1;

N = 100; % number of samples
offset = 5;
x = [randn(2,N)'; (randn(2,N)-offset)'];
A_x = x(1:N,1);
A_y = x(1:N,2);
B_x = x(N+1:2*N,1);
B_y = x(N+1:2*N,2);

% Batch Perceptron Algorithm
theta  = 5;
etta = 0.1; % learning rate

B_x_flipped = -1.*B_x; % replace one of the classes by their negatives
B_y_flipped = -1.*B_y;

w(1,1:3) = [0.1,-0.1,0.1];
k = 1;
data = [ones(1,N)';-ones(1,N)'];
x_modified = [A_x A_y;B_x_flipped B_y_flipped];
data_LS=[data x_modified]; % Linearly Separable dataset
counter = 2*N;
sum = [0,0,0];
while counter >0
    counter = 0;
    sum = [0,0,0];
    for i=1:2*N
        pt = data_LS(i,1:3);
        pt_product = dot(pt,w(k,1:3));
        if pt_product<=theta
            counter = counter+1;
```

```
                sum = sum+(etta.*pt);
            end
        end
        errorCount(k) = counter;
        k = k+1;
        w(k,1:3) = w(k-1,1:3)+sum;
    end

    % Plot data set
    figure(figNum);
    figNum = figNum+1;
    scatter(A_x, A_y, 'o');
    hold on
    scatter(B_x, B_y, '+');
    hold on

    % Calculate & plot decision boundary
    x_intercept = -theta/w(k,2);
    y_intercept = -theta/w(k,3);
    m = y_intercept/x_intercept;
    x = -10:0.1:10;
    y = -x./m-theta;
    hold on
    plot(x,y,'r')
    legend('data set1', 'data set 2', 'Decision Boundary')
    xlabel('x')
    ylabel('y')
    xlim([-10,5])
    ylim([-10,5])
    title('linearly separable classifer with Perceptron')

    % Plot classification % vs Iteration #
    figure(figNum);
    figNum = figNum+1;
    plot(errorCount)
    xlabel('Iteration #')
    ylabel('Number of Misclassified data')
    title('Classification Error vs Iteration #')
```


linearly separable classifer with Perceptron
data set1
data set 2
Decision Boundary
5
0
-5
-10
-10
-5
0
5
x
y

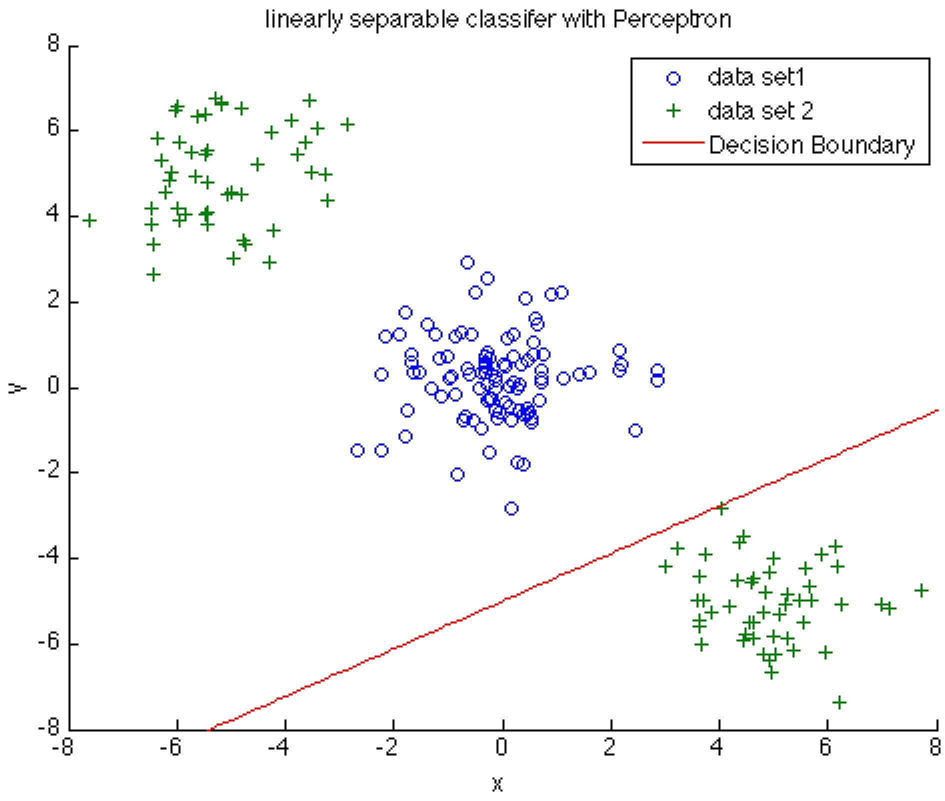

Classification Error vs Iteration #
Number of Misclassified data
Iteration #
200
180
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35

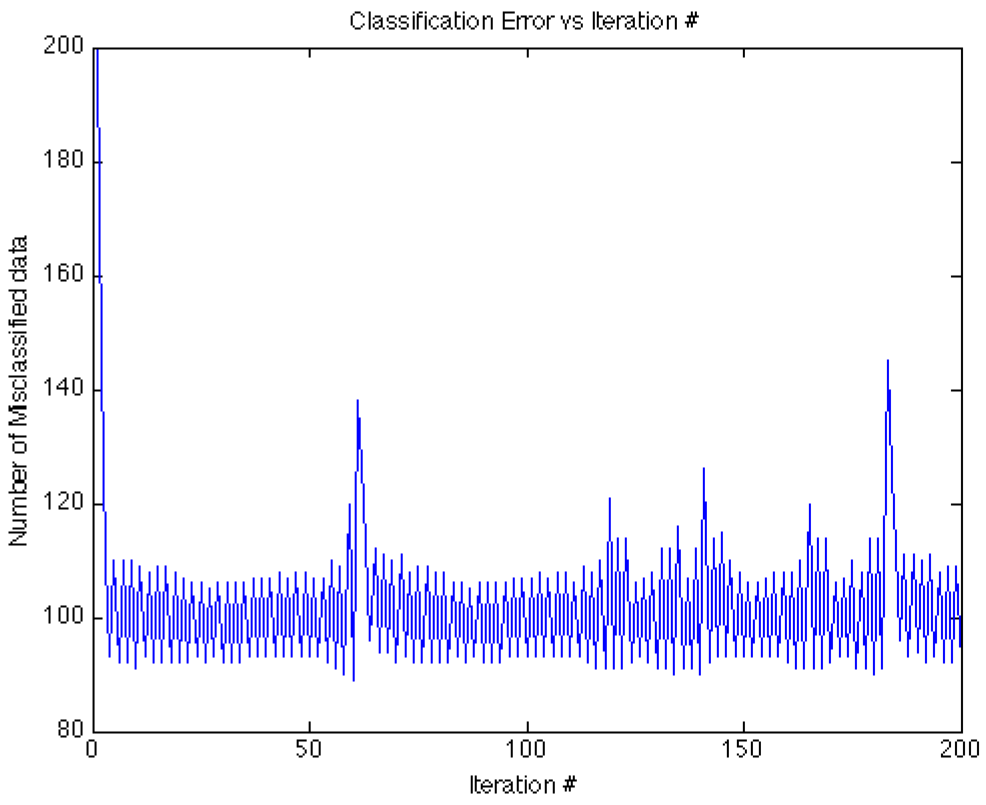
B. Linearly Non-separable dataset

The iteration number cap for the linearly non-separable dataset is set as 500. Also, the threshold value is 5. As shown in Fig. 3, the decision boundary does not successfully separate two datasets. Also Fig. 4 illustrates that the number of misclassified data fluctuates around 100 (50% accuracy). The classification accuracy does not converge.

```
clearvars -except figNum
M = 100;
offset = 5;
X = randn(2, M);
Y_1 = [randn(1, M/2)-offset; randn(1,M/2)+offset];
Y_2 = [randn(1, M/2)+offset; randn(1,M/2)-offset];
Y = [Y_1 Y_2];

% Batch Perceptron Algorithm
theta  = 5; % threshold value
etta = 0.1; % learning rate

Y_flipped= -1.*Y; % replace one of the classes by their negatives

w(1,1:3) = [0.1,-0.1,0.1];
k = 1;
data = [ones(1,M)';-ones(1,M)'];
data_modified = [X';Y_flipped'];
data=[data data_modified]; %Linearly Inseparable Dataset
counter = 2*M;
iteration = 0;
iterationCap = 500;
sum = [0,0,0];
while counter >0 && iteration < 200
    counter = 0;
    sum = [0,0,0];
    for i=1:2*M
        pt = data(i,1:3);
        pt_product = dot(pt,w(k,1:3));
        if pt_product<=theta
            counter = counter+1;
            sum = sum+(etta.*pt);
        end
    end
    errorCount(k) = counter;
    k = k+1;
    w(k,1:3) = w(k-1,1:3)+sum;
    iteration = iteration+1;
end
figure(figNum);
figNum = figNum+1;
scatter(X(1,:), X(2,:), 'o')
hold on
scatter(Y(1,:), Y(2,:), '+')
hold on
```

```
x_intercept = -theta/w(k,2);
y_intercept = -theta/w(k,3);
m = y_intercept/x_intercept;
x = -8:0.1:8;
y = -x./m-theta;


hold on
plot(x,y,'r')
legend('data set1', 'data set 2', 'Decision Boundary')
xlabel('x')
ylabel('y')
xlim([-8,8])
ylim([-8,8])
title('linearly separable classifer with Perceptron')

figure(figNum);
figNum = figNum+1;
plot(errorCount)
xlabel('Iteration #')
ylabel('Number of Misclassified data')
title('Classification Error vs Iteration #')
```



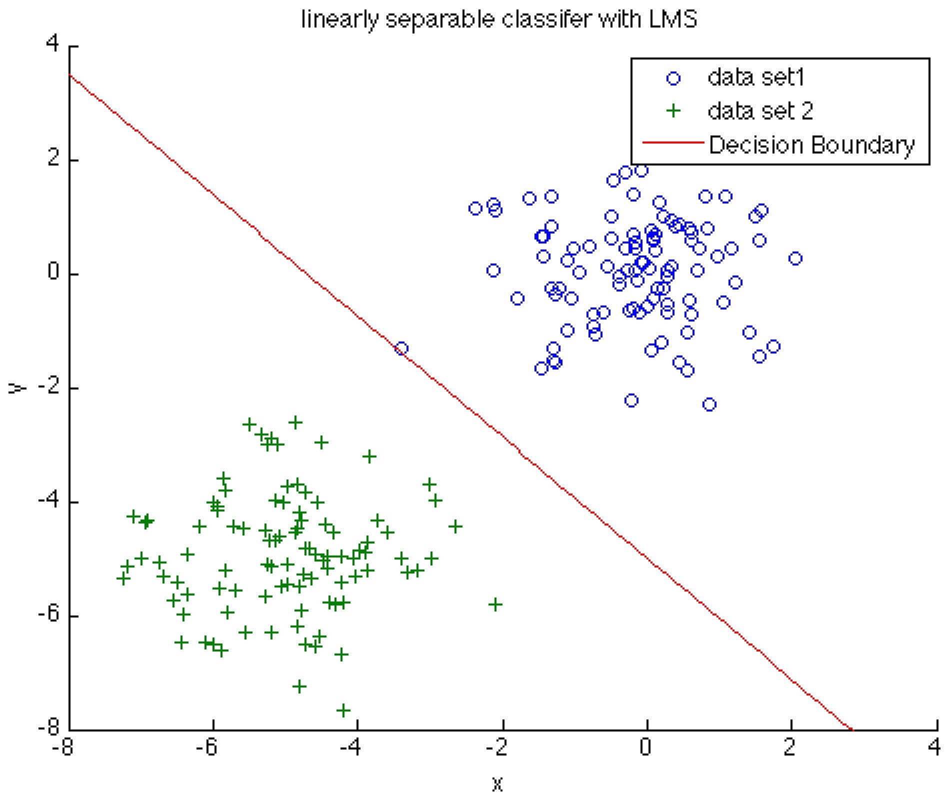


Problem 2

Using Least Mean Square shows faster running time, but it does not garuantees 100% accuracy. Sometimes, the decision boundary generated by this method misclassifies one data point (LMS_miss.fig in the same directory) The misclassification becomes worse when the offset between two data cohorts is smaller. However, the ofset is not adjusted (stays at 5) for better comparison with the first classification method.

```
clearvars -except figNum

N = 100; % number of samples
offset = 5;
X = [randn(2,N)'; (randn(2,N)-offset)'];
A_x = X(1:N,1);
A_y = X(1:N,2);
B_x = X(N+1:2*N,1);
B_y = X(N+1:2*N,2);
B_x_flipped = -1.*B_x; % replace one of the classes by their negatives
B_y_flipped = -1.*B_y;


data = [ones(1,N)';-ones(1,N)'];
x_modified = [A_x A_y;B_x_flipped B_y_flipped];
data_LS=[data x_modified]; % Linearly Separable dataset
b = ones(1,2*N)';
a = inv(data_LS'*data_LS)*data_LS'*b;
```

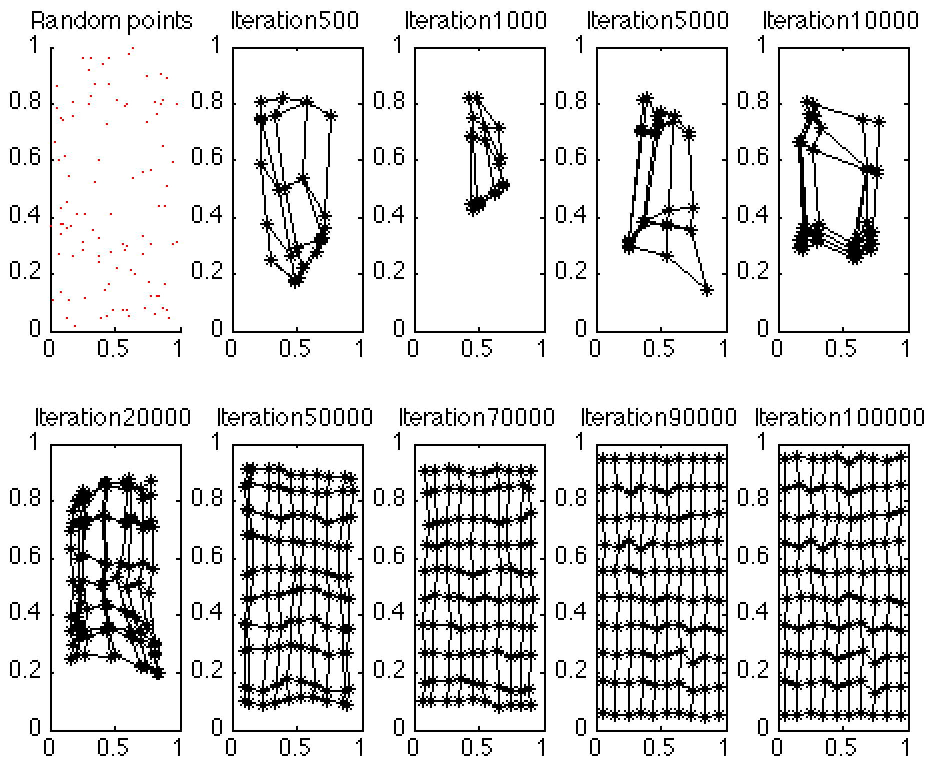
```
figure(figNum);
figNum = figNum+1;
scatter(A_x, A_y, 'o');
hold on
scatter(B_x, B_y, '+');
hold on

x = -10:0.1:10;
x_int_LMS = a(1)/a(2);
y_int_LMS = a(1)/a(3);

m_LMS = y_int_LMS/x_int_LMS;
y_LMS = -x./m_LMS-5;

hold on
plot(x,y_LMS,'r')
legend('data set1', 'data set 2', 'Decision Boundary')
xlabel('x')
ylabel('y')
xlim([-8,4])
ylim([-8,4])
title('linearly separable classifer with LMS')
```



Problem 3

The first part of the Self-Organizing Map (SOM) tests the random distribution. The weights (X) are generated randomly for each time t. Figure 6 shows that the map disperses with the increasing iteration number.

```
clearvars -except figNum
M = rand(100,2);

T1 = 30000;
T2 = 30000;
N0 = 0.3; % initial learning rate
sigma0 = 5;
Q = zeros(100,1);

figure(figNum)
figNum = figNum+1;
subplot(2,5,1)
scatter(M(:,1), M(:,2),'r.')
title('Random points')
subplotCount = 2;
for t=1:100000
    X = rand(1,2); % new input pattern
    for i = 1:100
        Q(i,1) = norm(X(1,:)-M(i,:));
    end
    [C,c] = min(Q); % find the unit w_i that best matches the input pattern xn
    etta = N0*exp(-t/T1); %learning rate decay rule
    r = round(sigma0*exp(-t/T2));
    ch = mod(c-1,10) + 1; % c starts at top left of the
    cv = floor((c-1)/10) + 1; % 2D SOM array and runs downwards!
    M = reshape(M,[10 10 2]);
    X = reshape(X,[1 1 2]);
    for h = max(ch-r,1):min(ch+r,10)
        for v = max(cv-r,1):min(cv+r,10)
            M(h,v,:) = M(h,v,:) + ...
                etta*(X(1,1,:) - M(h,v,:));
        end
    end

    if t == 500 || t == 1000 || t ==5000 || t==10000 || t == 20000 || t == 50000||
        subplot(2,5,subplotCount)
        plot(M(:,:,1), M(:,:,2),'k*',M(:,1,1),M(:,1,2),'k-',M(:,2,1), ...
            M(:,2,2),'k-',M(:,3,1),M(:,3,2),'k-',M(:,4,1),M(:,4,2),'k-', ...
            M(:,5,1),M(:,5,2),'k-',M(:,6,1),M(:,6,2),'k-',M(:,7,1), ...
            M(:,7,2),'k-',M(:,8,1),M(:,8,2),'k-',M(:,9,1),M(:,9,2),'k-',...
            M(:,10,1),M(:,10,2),'k-',...
            M(1,:,1),M(1,:,2),'k-', ...
            M(2,:,1),M(2,:,2),'k-',M(3,:,1),M(3,:,2),'k-',M(4,:,1), ...
            M(4,:,2),'k-',M(5,:,1),M(5,:,2),'k-',M(6,:,1),M(6,:,2),'k-', ...
            M(7,:,1),M(7,:,2),'k-',M(8,:,1),M(8,:,2),'k-', M(9,:,1),M(9,:,2),'k-',
            M(10,:,1),M(10,:,2),'k-',...
            M(:,3,1),...
            M(:,3,2),'k-',M(:,4,1),M(:,4,2),'k-',0,0,'.',1,1,'.');
        subplotCount = subplotCount+1;
        title(['Iteration' num2str(t)])
    end
    M = reshape(M,[100 2]);

end
```

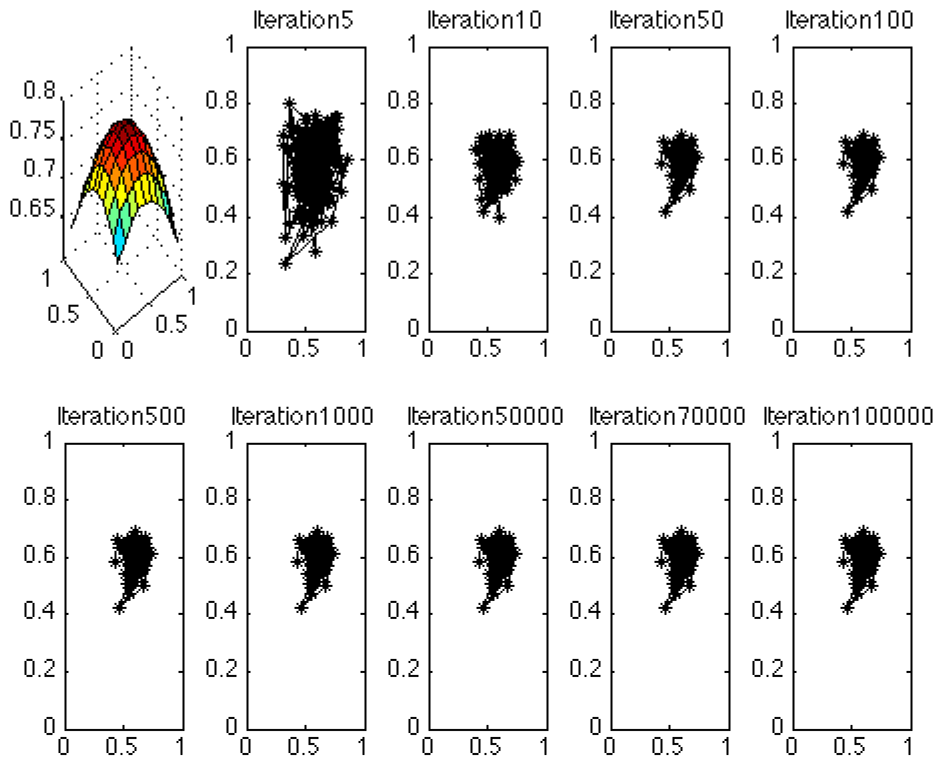


Gaussian Distribution

The second SOM tests the weights that follows Gaussian Distrubution. In Figure 7, the 2D lattice map converges to the center point very quickly. The learning rate is adjusted to 0.2 instead of 0.4, but the kernel size (sigma) was not changed. As we can see, the map plateaus around iteration 100.

```
clearvars -except figNum

% create a random sigma, mu:
mu = [0.5 0.5];
Sigma = cov(randn(100,2));
[X,Y] = meshgrid(0.1:.1:1);
XY = cat(3,X,Y);
XYmmu = bsxfun(@minus,XY,shiftdim(mu(:),-2));
isigXY = squeeze(sum(bsxfun(@times,shiftdim(inv(Sigma),-2),XYmmu),3));
XYisXY = sum(isigXY .* XYmmu,3);
normcdf = 5.*(1/(2*pi*sqrt(det(Sigma)))) * exp(-0.5 * XYisXY);
normcdf_lattice = reshape(normcdf,[100,1]);
figure(figNum)
```

```
figNum = figNum+1;
subplot(2,5,1)
surf(X,Y,normcdf);
M = rand(100,2);

T1 = 30000;
T2 = 30000;
N0 = 0.2; % initial learning rate
sigma0 = 5;
Q = zeros(100,1);

subplotCount = 2;
for t=1:100000
    for i = 1:100
        X = [normcdf_lattice(i) normcdf_lattice(i)];
        Q(i,1) = norm(X(1,:)-M(i,:));
    end
    [C,c] = min(Q); % find the unit w_i that best matches the input pattern xn
    etta = N0*exp(-t/T1); %learning rate decay rule
    r = round(sigma0*exp(-t/T2));
    ch = mod(c-1,10) + 1; % c starts at top left of the
    cv = floor((c-1)/10) + 1; % 2D SOM array and runs downwards!
    M = reshape(M,[10 10 2]);
    X = reshape(X,[1 1 2]);
    for h = max(ch-r,1):min(ch+r,10)
        for v = max(cv-r,1):min(cv+r,10)
            M(h,v,:) = M(h,v,:) + ...
                etta*(X(1,1,:)- M(h,v,:));
        end
    end

    if t == 5 || t == 10 || t ==50 || t==100 || t == 500 || t == 1000|| t ==50000
        subplot(2,5,subplotCount)
        plot(M(:,:,1), M(:,:,2),'k*',M(:,1,1),M(:,1,2),'k-',M(:,2,1), ...
            M(:,2,2),'k-',M(:,3,1),M(:,3,2),'k-',M(:,4,1),M(:,4,2),'k-', ...
            M(:,5,1),M(:,5,2),'k-',M(:,6,1),M(:,6,2),'k-',M(:,7,1), ...
            M(:,7,2),'k-',M(:,8,1),M(:,8,2),'k-',M(:,9,1),M(:,9,2),'k-',...
            M(:,10,1),M(:,10,2),'k-',...
            M(1,:,1),M(1,:,2),'k-', ...
            M(2,:,1),M(2,:,2),'k-',M(3,:,1),M(3,:,2),'k-',M(4,:,1), ...
            M(4,:,2),'k-',M(5,:,1),M(5,:,2),'k-',M(6,:,1),M(6,:,2),'k-', ...
            M(7,:,1),M(7,:,2),'k-',M(8,:,1),M(8,:,2),'k-', M(9,:,1),M(9,:,2),'k-',
            M(10,:,1),M(10,:,2),'k-',...
            M(:,3,1),...
            M(:,3,2),'k-',M(:,4,1),M(:,4,2),'k-',0,0,'.',1,1,'.');
        subplotCount = subplotCount+1;
        title(['Iteration' num2str(t)])
    end
    M = reshape(M,[100 2]);

end
```



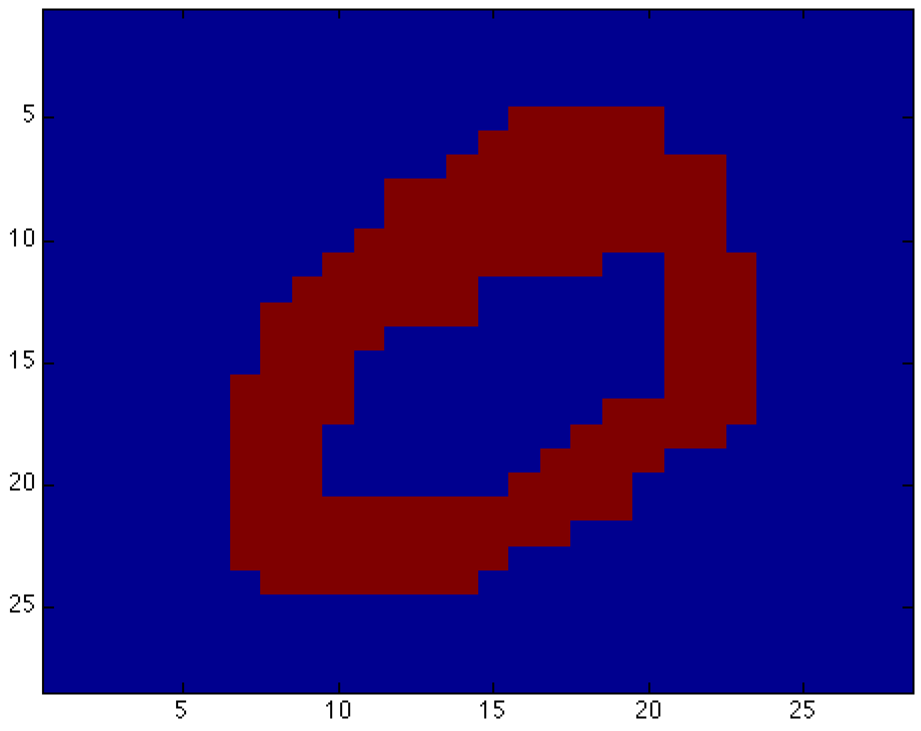
Problem 4

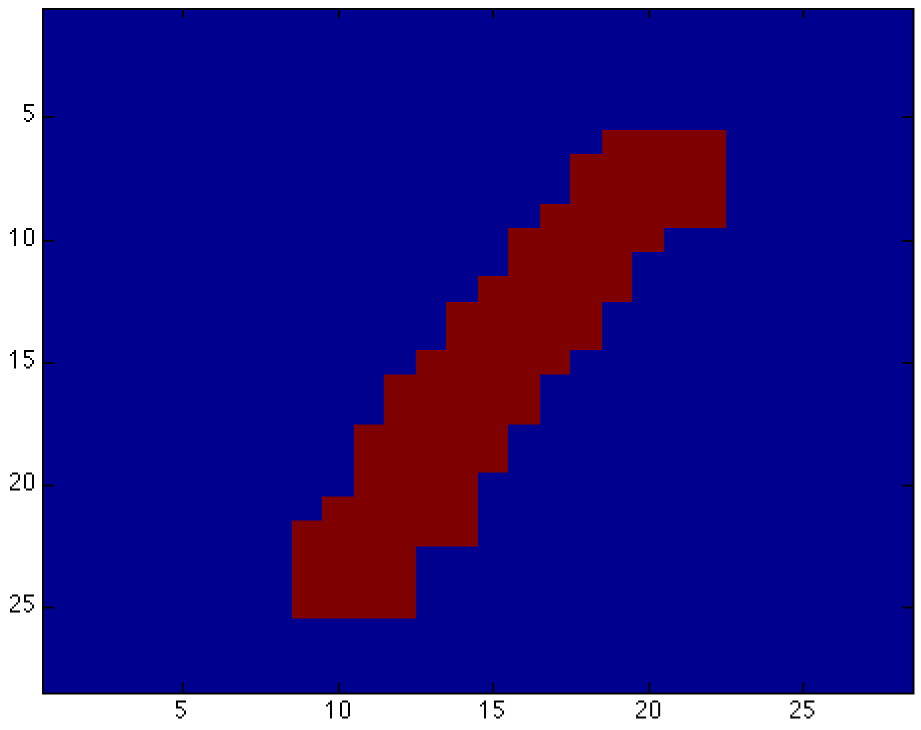
```
clearvars -except figNum
load('mnist_all.mat')


% Reshape arrays in to a 28 by 28 matrix
train_image{1,1} = reshape(train0(1,:),28,28)';
train_image{1,2} = reshape(train1(1,:),28,28)';
train_image{1,3} = reshape(train2(1,:),28,28)';
train_image{1,4} = reshape(train3(1,:),28,28)';
train_image{1,5} = reshape(train4(1,:),28,28)';
train_image{1,6} = reshape(train5(1,:),28,28)';
train_image{1,7} = reshape(train6(1,:),28,28)';
train_image{1,8} = reshape(train7(1,:),28,28)';
train_image{1,9} = reshape(train8(1,:),28,28)';
train_image{1,10} = reshape(train9(1,:),28,28)';

test_image{1,1} = reshape(test0(1,:),28,28)';
test_image{1,2} = reshape(test1(1,:),28,28)';
test_image{1,3} = reshape(test2(1,:),28,28)';
test_image{1,4} = reshape(test3(1,:),28,28)';
test_image{1,5} = reshape(test4(1,:),28,28)';
test_image{1,6} = reshape(test5(1,:),28,28)';
test_image{1,7} = reshape(test6(1,:),28,28)';
```

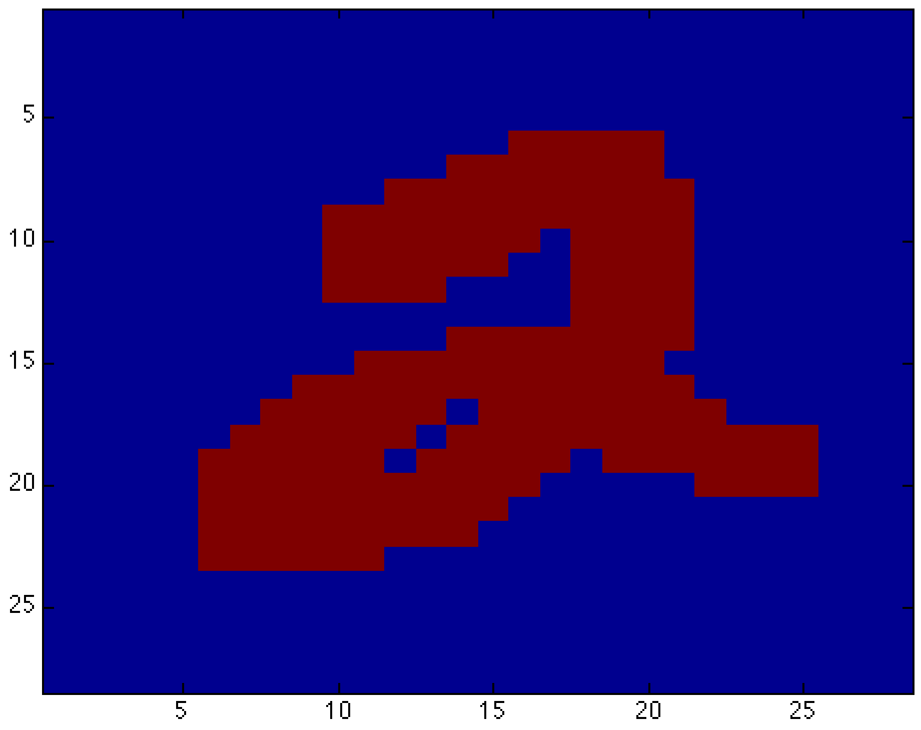
```
test_image{1,8} = reshape(test7(1,:),28,28)';
test_image{1,9} = reshape(test8(1,:),28,28)';
test_image{1,10} = reshape(test9(1,:),28,28)';
% Plotting settings
linespec = {'linestyle' '-';'linestyle' ':';'linestyle' '-.';'linestyle' '--'}; %
markerspec = {'o';'*';'+';'.'}; % Marker specs for gray-scale
color = {'color' 'blue'; 'color' 'red'; 'color' 'green'; ...
    'color' 'cyan'; 'color' 'black'; 'color' 'yellow'};
% fontsize{1,:} for title, fontsize{2,:} for axis, fontsize{3,:} for legend
fontsize = {'Fontsize' 13 'Fontweight' 'bold'; ...
    'Fontsize', 10, 'Fontweight', 'bold'; ...
    'Fontsize', 11, 'Fontweight', 'bold'};
% linespec2{1,:} for line, linespec2{2,:} for marker
linespec2 = {'linewidth' 1.3; 'MarkerSize' 6};
%  Make 2-D coordinates for each data points
% First, mask the image into a binary image
% Note that the second row of the cell is the binary image
for i = 1:size(train_image,2)
    train_image{2,i} = mask_image(train_image{1,i});
    test_image{2,i} = mask_image(test_image{1,i});
end

% Make a 784 by 3 matrix that has off-set, x-coordinate and y-coordinate
% Note that the third row of the cell is the 2-D coordinates
for i = 1:size(train_image,2)
    train_image{3,i} = find_coord_2D(train_image{2,i});
    test_image{3,i} = find_coord_2D(test_image{2,i});
end

% Use five layer perceptron to recognize the digit in 2-D space
% Make a batch matrix
train_coord = []; d = [];
for i = 1:size(train_image,2)
    train_coord = [train_coord; train_image{3,i}];
    d = [d; (i-1)*ones(size(train_coord,1),1)];
end
% Run 5-layers perceptrons for 2-D space
% Note that the groups are either 0 or 1, so the d matrix is actually a
% vector of ones: more details in the homework page
% The wegith cell is organized such that:
% w{1,i} = u1, w{2,i} = u2, w{3,i} = w
params.eta = 0.1; params.a = 0.2; params.b = 0.2;
params.logistics = 1; params.tangentf = 0; params.saveweights = 0;
[w_l{1},w_l{2},w_l{3},e_l] = perceptron_2layers_2D(train_coord,d,params);

% Use acquired weights to predict the test image set

params.logistics = 1; params.tangentf = 0;
for i = 1:size(train_image,2)
    class(i) = test_perceptron_2layers_2D(w_l,test_image{3,i},params);

end
```

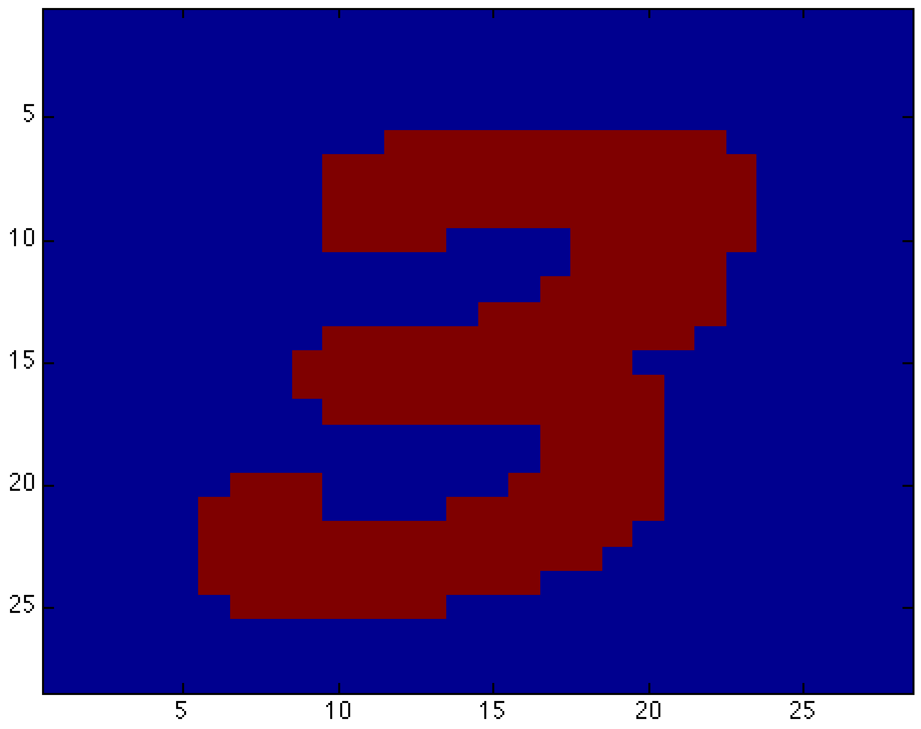
```
%Make 3-D coordinatges for each data points



for i = 1:10
    figure(figNum)
    figNum = figNum+1;
    imagesc(train_image{2,i})
end
```

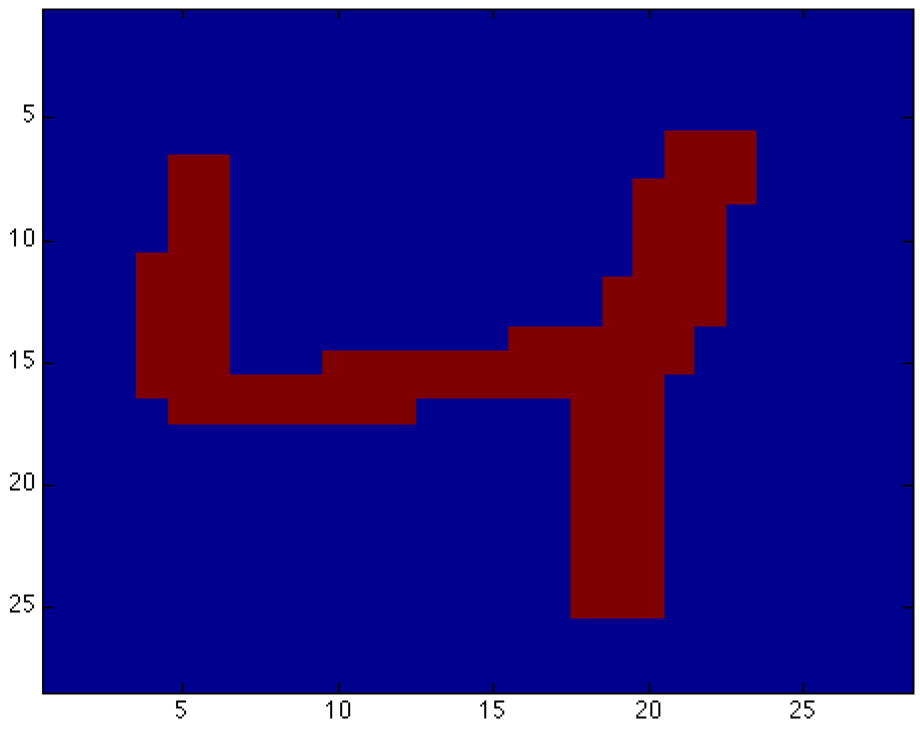
```
j =

        5000


ans =

   -0.1634


j =

       10000


ans =

   -0.0201
```

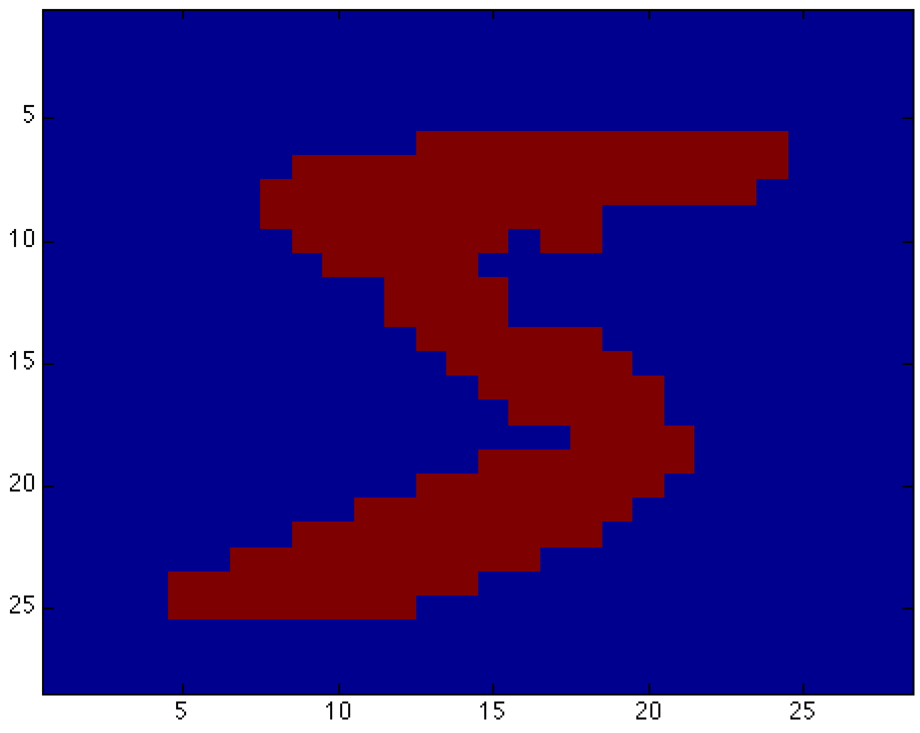

5
10
15
20
25
5
10
15
20
25

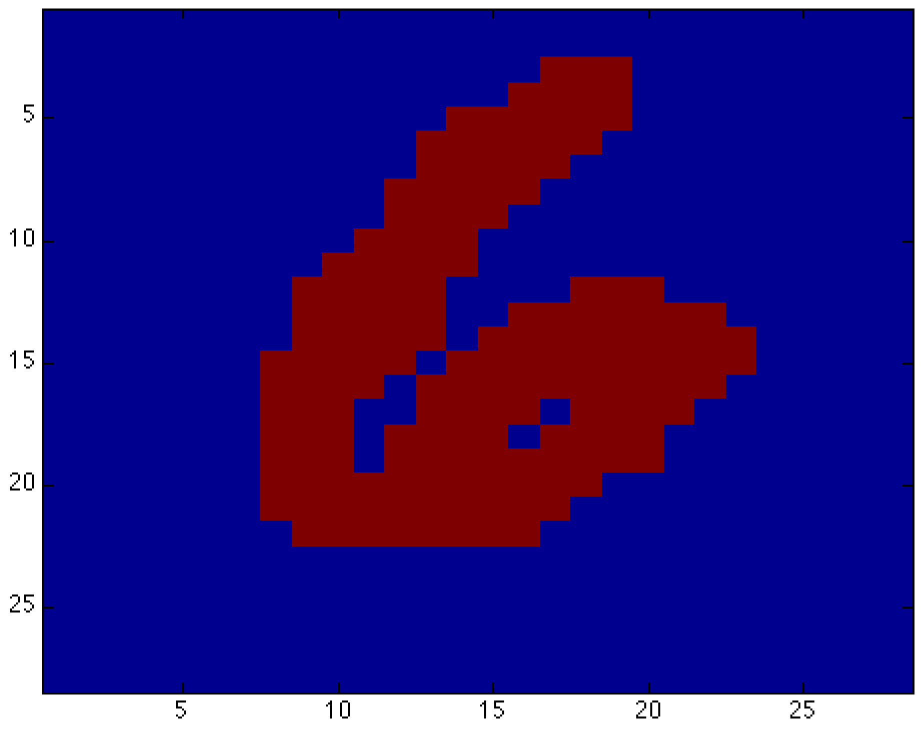

5
10
15
20
25
5
10
15
20
25

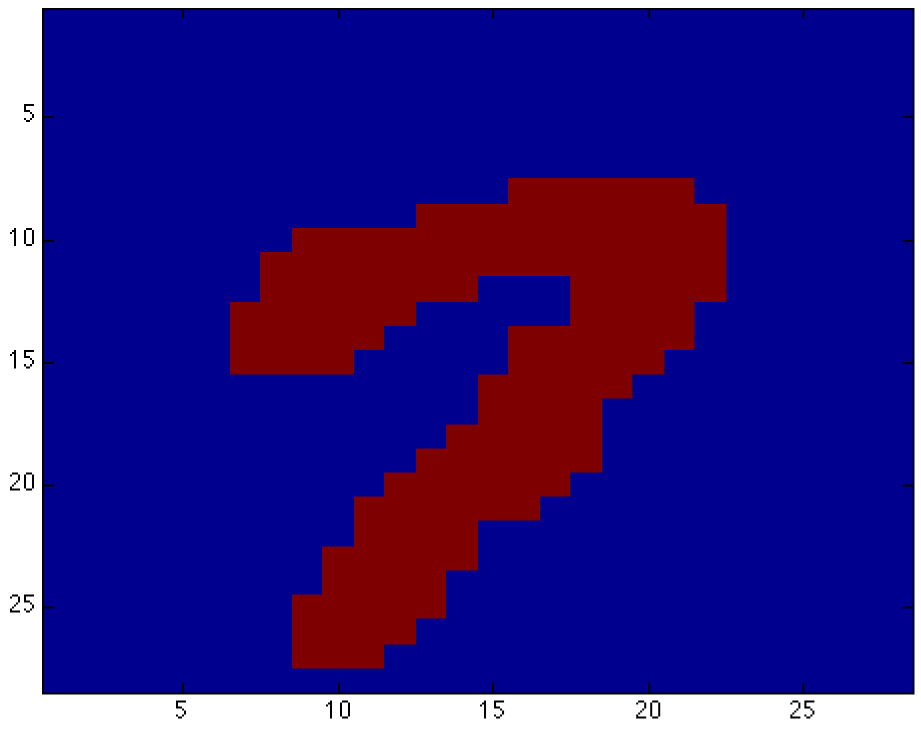

5
10
15
20
25
5
10
15
20
25

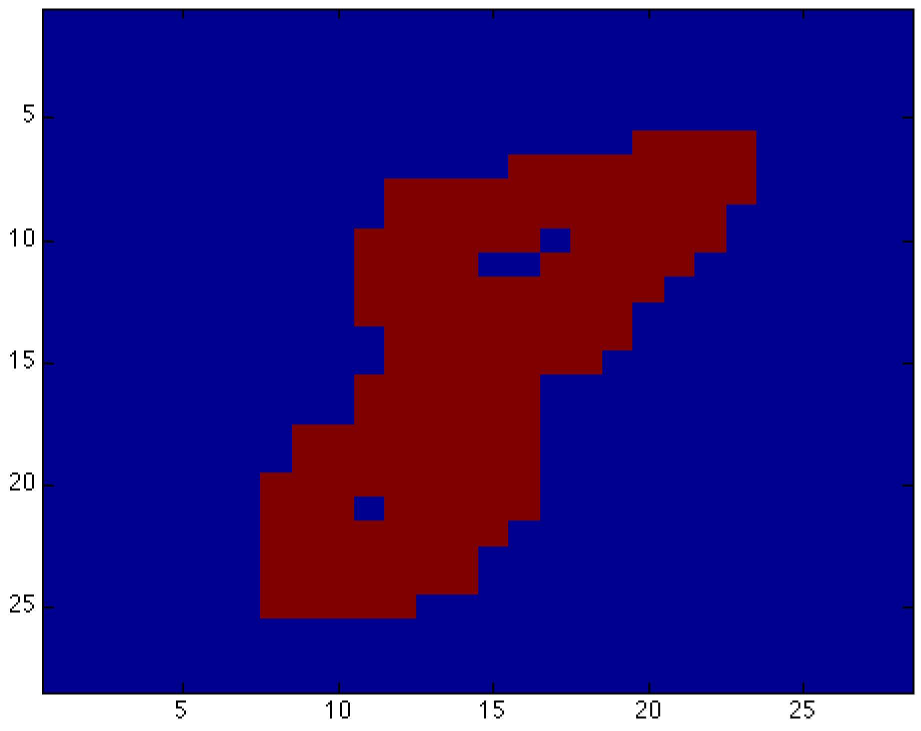

5
10
15
20
25
5
10
15
20
25

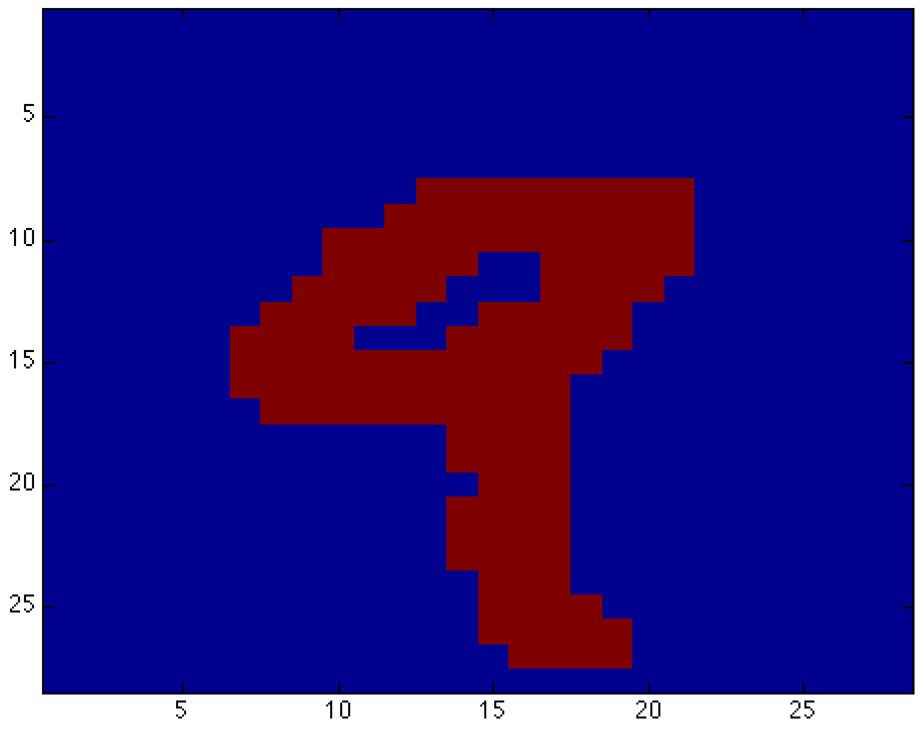

5
10
15
20
25
5
10
15
20
25


5
10
15
20
25
5
10
15
20
25


5
10
15
20
25
5
10
15
20
25


5
10
15
20
25
5
10
15
20
25


5
10
15
20
25
5
10
15
20
25


5
10
15
20
25
5
10
15
20
25

Published with MATLAB® R2013a