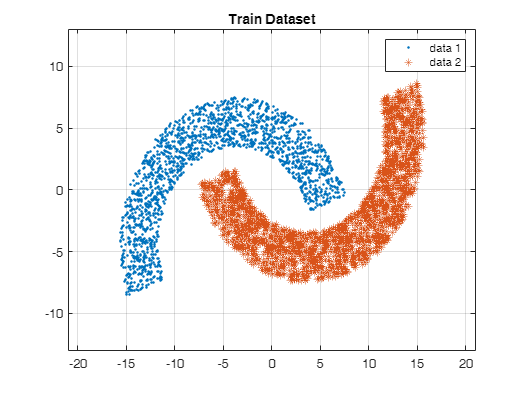
آرین حاجی زاده 99411281

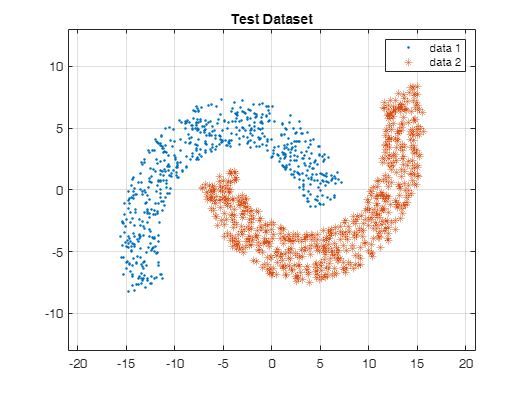
Moonshape Structure

clear;

figure();

[x\_train, x\_test] = doubleMoonStructure(10, 4, 20, -6 ,5000);





X = [x\_train(:,1:2)./10;x\_train(:,3:4)./10];

x\_test = [x\_test(:,1:2)./10;x\_test(:,3:4)./10];

% desired output classes

y\_test = [ones(size(x\_test,1)/2,1); -1\*ones(size(x\_test,1)/2,1)];

d\_k = [ones(size(X,1)/2,1); -1\*ones(size(X,1)/2,1)];

flag = 1;

x\_train = X;

Given non-linear function

% input signal :

clear;

flag = 0;

u(1) = 0;

for k = 2:401

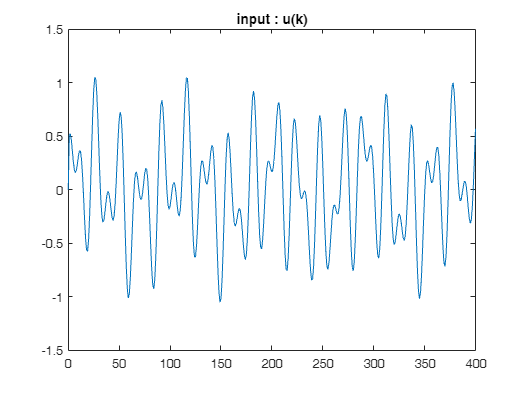
u(k) = 0.5\*sin(pi\*(k-1)/11) + 0.4\*cos(pi\*(k-1)/6.5)+0.2\*sin(pi\*(k-1)/45);

end

figure();

plot(0:k-1,u);

title("input : u(k)")



alpha = 1.2;

beta = [1.1 1.5];

% y\_nlf -> y non-linear function

y\_nlf(1,:) = [0,0];

y\_nlf(2,:) = alpha\*u(2)\*ones(1,2);

X(1,:) = [u(1) y\_nlf(1,1) 0];

X(2,:) = [u(2) y\_nlf(2,1) y\_nlf(1,1)];

for k = 3:400

y\_nlf(k,:) = alpha\*((y\_nlf(k-1)\*y\_nlf(k-2)\*(y\_nlf(k-2) + beta))/(1+(y\_nlf(k-2)^2).\*(y\_nlf(k-1)^2)) + u(k));

X(k,:) = [u(k), y\_nlf(k-1,1),y\_nlf(k-2)];

end

for i = 1:size(X,2)

X(:,i) = X(:,i)./max(X(:,i));

end

tmp = 1;

tmp2 = 1;

% Cross varidation (train: 70%, test: 30%)

cv = cvpartition(size(X,1),'HoldOut',0.2);

idx = cv.test;

% Separate to training and test data

x\_train = X(~idx,:);

y\_train = y\_nlf(~idx,:);

x\_test = X(idx,:);

y\_test = y\_nlf(idx,:)./max(y\_nlf(idx,:));

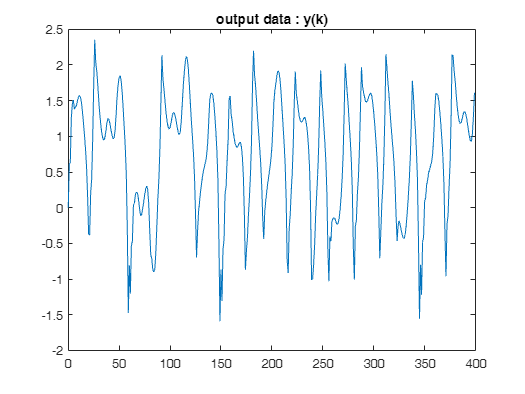
%normalizing

d\_k = y\_train(:,1)/max(y\_train(:,1));

figure;

plot(0:399,y\_nlf(:,1));

title("output data : y(k)")



%MLP

input\_layer\_size = size(x\_train, 2); % Number of features

hidden\_layer\_size = 30; % You can choose the size of the hidden layer

output\_layer\_size = 1; % For binary classification

learning\_rate = 0.005;

num\_epochs = 100; % You can choose the number of epochs

% Randomly initialize weights

W\_ji = rand(hidden\_layer\_size, input\_layer\_size) \* 0.01;

%bias

b1 = zeros(hidden\_layer\_size, 1);

% Randomly initialize weights

W\_kj = rand(output\_layer\_size, hidden\_layer\_size) \* 0.01;

%bias

b2 = zeros(output\_layer\_size, 1);

% Activation function

% sigmoid = @(z) 1 ./ (1 + exp(-z));

% Derivative of sigmoid for backpropagation

% sigmoidGradient = @(z) sigmoid(z) .\* (1 - sigmoid(z));

sigmoid = @(vs) (1-exp(-0.6\*vs)) ./ ( 1 + exp(-0.6\*vs) );

sigmoidGradient = @(vs) (1-sigmoid(vs).^2);

% momentum rate

momentum = 0;

momentum\_idx = 0.0005;

prev\_deltaW\_kj = 0;

prev\_dletaW\_ji = 0;

prev\_b1 = 0;

prev\_b2 = 0;

% this is for showing the validation data

tst\_show = 3;

tmp\_validation = 1;

loss\_validation = 1;% initialization

% Training the MLP

for epoch = 1:num\_epochs

% Forward propagation

v\_j = x\_train \* W\_ji.' + b1.';

y\_j = sigmoid(v\_j);

v\_k = y\_j \* W\_kj.' + b2.';

y\_k = sigmoid(v\_k);

% Compute the loss (mean squared error in this case)

loss = mean((y\_k - d\_k).^2);

% Backward propagation

dalta\_k = (d\_k - y\_k) .\* sigmoidGradient(v\_k);

dalta\_j = (dalta\_k \* W\_kj) .\* sigmoidGradient(v\_j);

% Update weights and biases

delta\_W\_kj = (learning\_rate \* (dalta\_k.' \* y\_j)) + (momentum\_idx \* prev\_deltaW\_kj);

W\_kj = W\_kj + delta\_W\_kj;

prev\_deltaW\_kj = delta\_W\_kj;

% the same update process here but just for biases

b2 = b2 + learning\_rate \* sum(dalta\_k, 1).' + b2\*momentum\_idx;

delta\_W\_ji = (learning\_rate \* (dalta\_j.' \* x\_train)) + (momentum\_idx \* prev\_dletaW\_ji);

W\_ji = W\_ji + delta\_W\_ji;

prev\_dletaW\_ji = delta\_W\_ji;

b1 = b1 + learning\_rate \* sum(dalta\_j, 1).' + (b1\*momentum\_idx);

e(epoch) = loss;

predict = @(test1) (sigmoid(test1 \* W\_ji.' + b1.') \* W\_kj.' + b2.');

% Display loss every 100 epochs

if mod(epoch, tst\_show) == 0

fprintf('Epoch %d, Loss: %f\n', epoch, loss);

pred = sigmoid(predict(x\_test));

loss\_validation(tmp\_validation) = mean((y\_test(:,1) - pred).^2);

loss\_compare(tmp\_validation) = loss;

fprintf('Epoch %d, Loss validation: %f\n\n', epoch, loss\_validation(tmp\_validation));

tmp\_validation = tmp\_validation + 1;

end

% cross validation stop condition

if(tmp\_validation > 2)

if(abs(loss\_validation(tmp\_validation -1) - loss\_validation(tmp\_validation -2)) < 0.001 && ...

abs(loss\_validation(tmp\_validation -1) - loss\_compare(tmp\_validation -1)) < 0.01)

tmp\_epoch = epoch;

break;

end

end

tmp\_epoch = epoch;

end

Epoch 3, Loss: 0.134093

Epoch 3, Loss validation: 0.140215

Epoch 6, Loss: 0.126291

Epoch 6, Loss validation: 0.134815

Epoch 9, Loss: 0.125622

Epoch 9, Loss validation: 0.133639

Epoch 12, Loss: 0.124734

Epoch 12, Loss validation: 0.132321

Epoch 15, Loss: 0.122718

Epoch 15, Loss validation: 0.129576

Epoch 18, Loss: 0.118190

Epoch 18, Loss validation: 0.123574

Epoch 21, Loss: 0.108588

Epoch 21, Loss validation: 0.111298

Epoch 24, Loss: 0.090540

Epoch 24, Loss validation: 0.089751

Epoch 27, Loss: 0.063680

Epoch 27, Loss validation: 0.061112

Epoch 30, Loss: 0.036557

Epoch 30, Loss validation: 0.035955

Epoch 33, Loss: 0.019765

Epoch 33, Loss validation: 0.021690

Epoch 36, Loss: 0.012800

Epoch 36, Loss validation: 0.015521

Epoch 39, Loss: 0.010424

Epoch 39, Loss validation: 0.013018

Epoch 42, Loss: 0.009655

Epoch 42, Loss validation: 0.011953

Epoch 45, Loss: 0.009399

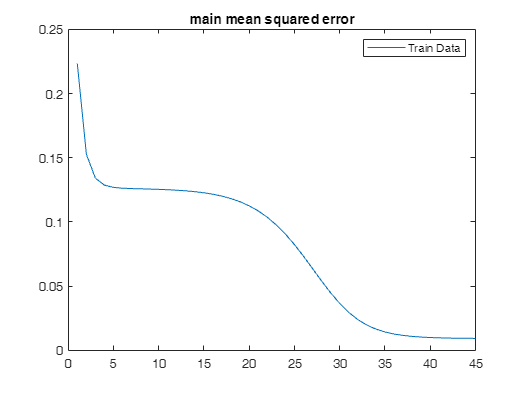
Epoch 45, Loss validation: 0.011458

figure();

plot([1:tmp\_epoch],e);

title("main mean squared error");

legend("Train Data")



figure();

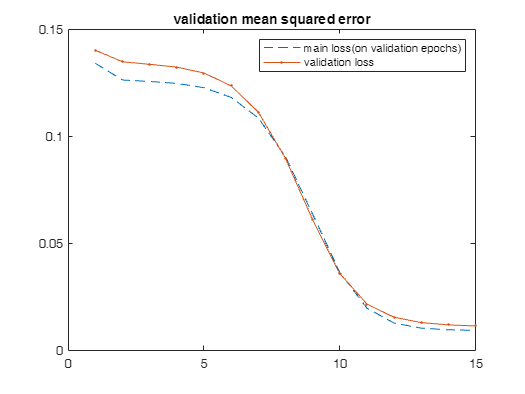
plot([1:tmp\_validation-1],loss\_compare,'--');

hold on

plot([1:tmp\_validation-1],loss\_validation,'.-');

title("validation mean squared error")

legend(["main loss(on validation epochs)","validation loss"])



% Prediction function

% predict = @(X) sigmoid(X \* W\_ji.' + b1.') \* W\_kj.' + b2.';

figure;

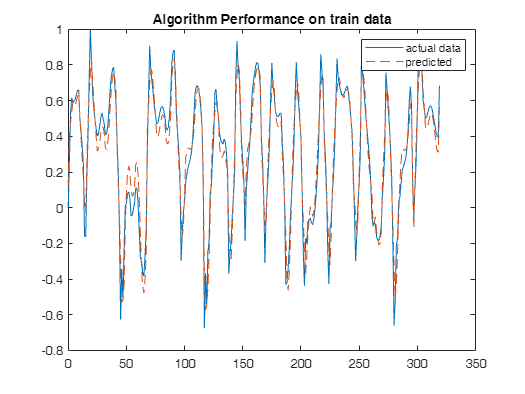
plot(0:size(d\_k,1)-1,d\_k);

hold on

plot(0:size(d\_k,1)-1,y\_k,"--");

title("Algorithm Performance on train data");

legend(["actual data","predicted"])



% Now you can use predict function to get the predictions on new data

% predictions = predict(x\_test);

% final\_err = y\_test - predictions;

% disp(mean(final\_err));

figure;

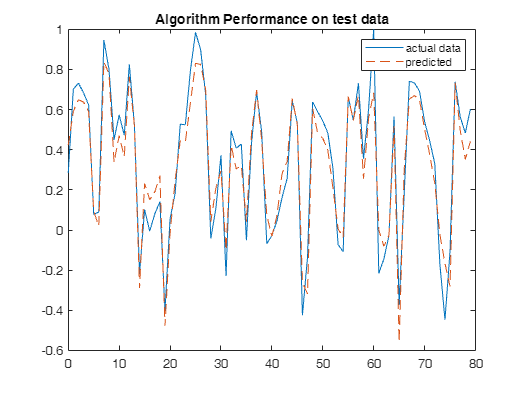
plot(0:size(x\_test,1)-1,y\_test(:,1));

hold on

plot(0:size(x\_test,1)-1,pred,"--");

title("Algorithm Performance on test data");

legend(["actual data","predicted"]);



if(flag == 1)

x\_range = [min(x\_train(:,1)), max(x\_train(:,1))];

y\_range = [min(x\_train(:,2)), max(x\_train(:,2))];

% Create a meshgrid of x and y values

[x\_grid, y\_grid] = meshgrid(linspace(x\_range(1), x\_range(2), 100), linspace(y\_range(1), y\_range(2), 100));

% Flatten the meshgrid into a matrix of input points

input\_points = [x\_grid(:), y\_grid(:)];

% Make predictions for each input point using the learned weights and biases

v\_j = input\_points \* W\_ji.' + b1.';

y\_j = sigmoid(v\_j);

v\_k = y\_j \* W\_kj.' + b2.';

y\_k = sigmoid(v\_k);

% Reshape the predictions back into a meshgrid

predictions = reshape(y\_k, size(x\_grid));

% Plot the decision boundary

figure;

contour(x\_grid, y\_grid, predictions, [0.5, 0.5], 'LineWidth', 2, 'Color', 'k');

hold on;

plot(x\_train(1:size(x\_train)/2, 1), x\_train(1:size(x\_train)/2, 2), 'bo', 'MarkerSize', 5);

plot(x\_train(size(x\_train)/2 + 1 : end, 1), x\_train(size(x\_train)/2 + 1 : end, 2), 'ro', 'MarkerSize', 5);

legend('Decision Boundary', 'Class 1', 'Class 2');

xlabel('x');

ylabel('y');

title('Decision Boundary');

end

function [train, test] = doubleMoonStructure( radius, width, rotation, separationDistance, ...

datasetSize, trainTestRatio, drawPatterns )

switch nargin

case 4

datasetSize = 2000;

trainTestRatio = 0.75;

drawPatterns = true;

case 5

trainTestRatio = 0.75;

drawPatterns = true;

case 6

drawPatterns = true;

end

nTrain = cast(datasetSize \* trainTestRatio, "uint16");

iTrain = cast(nTrain / 2, "uint16");

N = datasetSize / 2;

theta = rotation \* pi / 180;

R = [ cos(theta), -sin(theta) ;

sin(theta), cos(theta) ];

r = radius + width/2;

xBias = r - (width/2);

yBias = -separationDistance;

magnitude = (r-width)\*ones(N,1) + rand(N,1)\*width;

phase = rand(N,1)\*pi;

class = [magnitude.\*cos(phase) - xBias/2, magnitude.\*sin(phase) - yBias/2];

class = (R \* class')';

train = class(1:iTrain,:);

test = class(iTrain+1:end,:);

magnitude = (r-width)\*ones(N,1) + rand(N,1)\*width;

phase = pi + rand(N,1)\*pi;

class = [magnitude.\*cos(phase) + xBias/2, magnitude.\*sin(phase) + yBias/2];

class = (R \* class')';

train = [ train, class(1:iTrain,:) ];

test = [ test, class(iTrain+1:end,:) ];

plotOffset = 4;

if drawPatterns

figure('Name','Train Dataset');

plot(train(:, 1), train(:, 2), '.');

xlim([-r-4-xBias/2,2\*r-width/2+4-xBias/2]);

ylim([-separationDistance/2-r-4, r+4+separationDistance/2]);

hold on; grid on;

plot(train(:, 3), train(:, 4), '\*');

title('Train Dataset');

legend(["data 1", "data 2"]);

figure('Name','Test Dataset');

plot(test(:, 1), test(:, 2), '.');

xlim([- r - xBias/2 - plotOffset, 2\*r - width/2 - xBias/2 + plotOffset]);

ylim([-separationDistance/2 - r - plotOffset, r + separationDistance/2 + plotOffset]);

hold on; grid on;

plot(test(:, 3), test(:, 4), '\*');

title('Test Dataset');

legend(["data 1", "data 2"]);

end

end