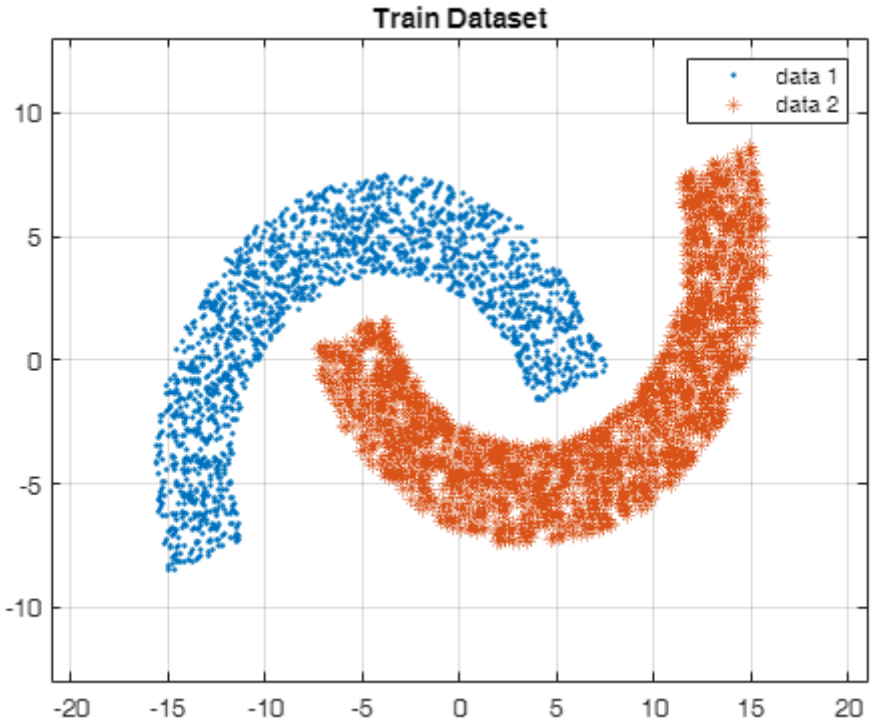
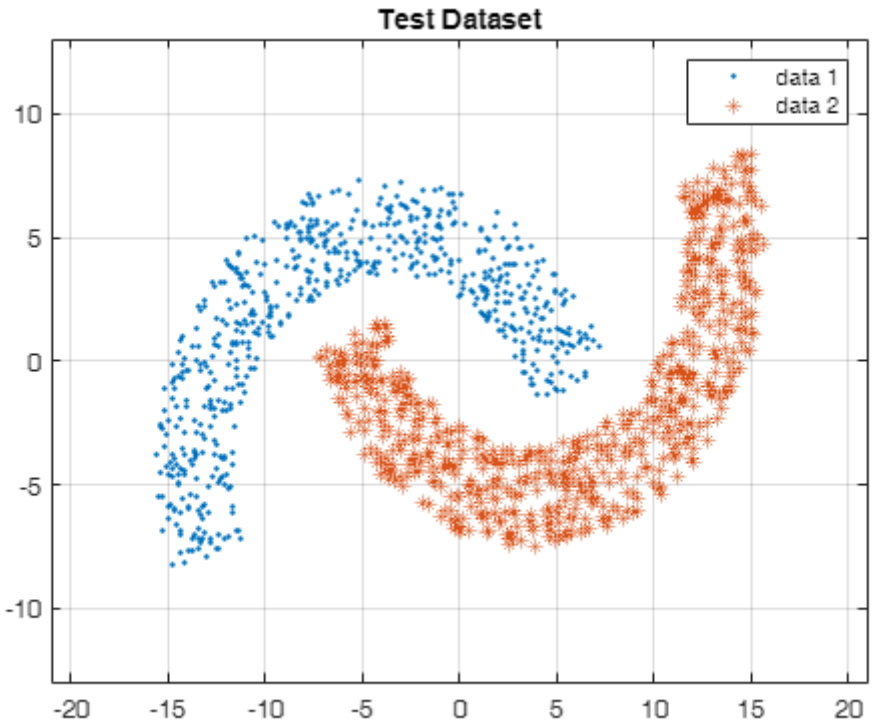
آرین حاجی زاده ۹۹۴۱۱۲۸۱

Moonshape Structure

```
clear;
figure();
[x_train, x_test] = doubleMoonStructure(10, 4, 20, -6 ,5000);
```



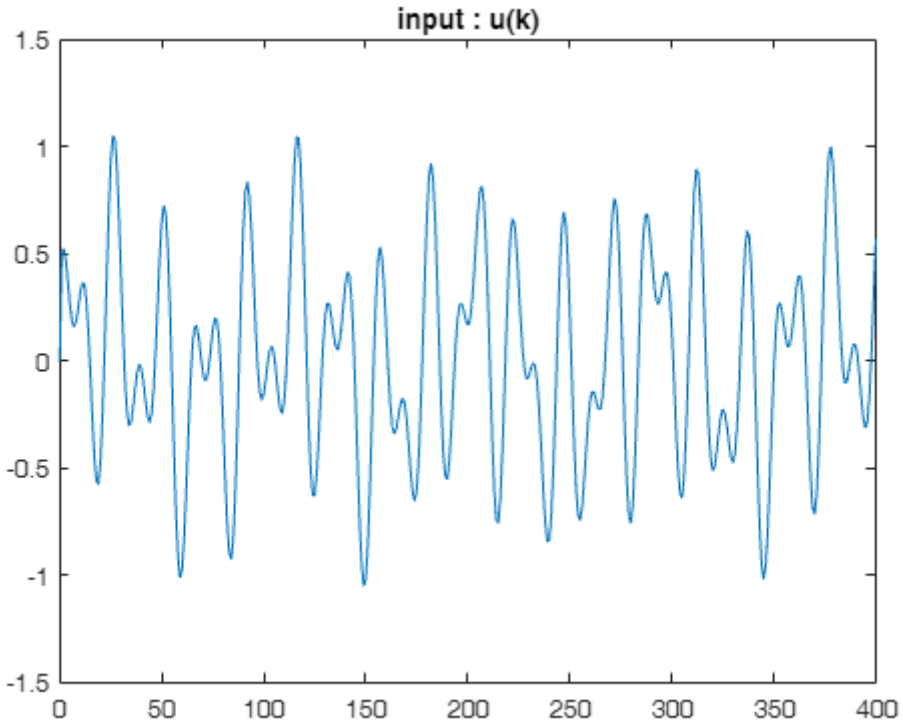



```
X = [x_train(:,1:2)./10;x_train(:,3:4)./10];
x_test = [x_test(:,1:2)./10;x_test(:,3:4)./10];


% 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

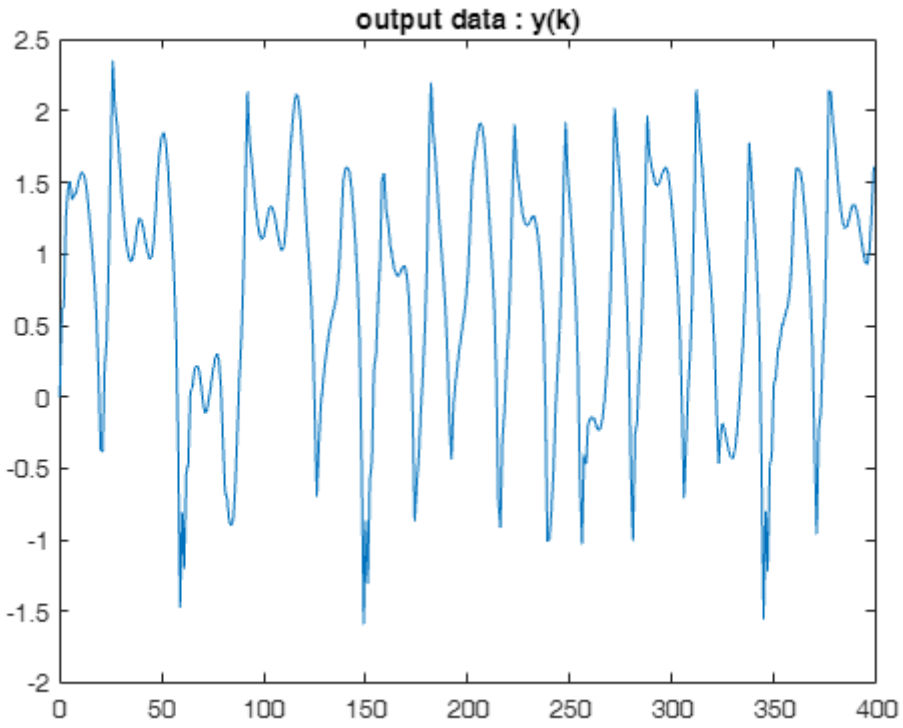
```
% input signal :
clear;
flag = 0;
u(1) = 0;
for k = 2:401
    u(k) = 0.5*sin(pi*(k-1)/11) + 0.4*cos(pi*(k-1)/6.5)+0.2*sin(pi*(k-1)/45);
```

```
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
```

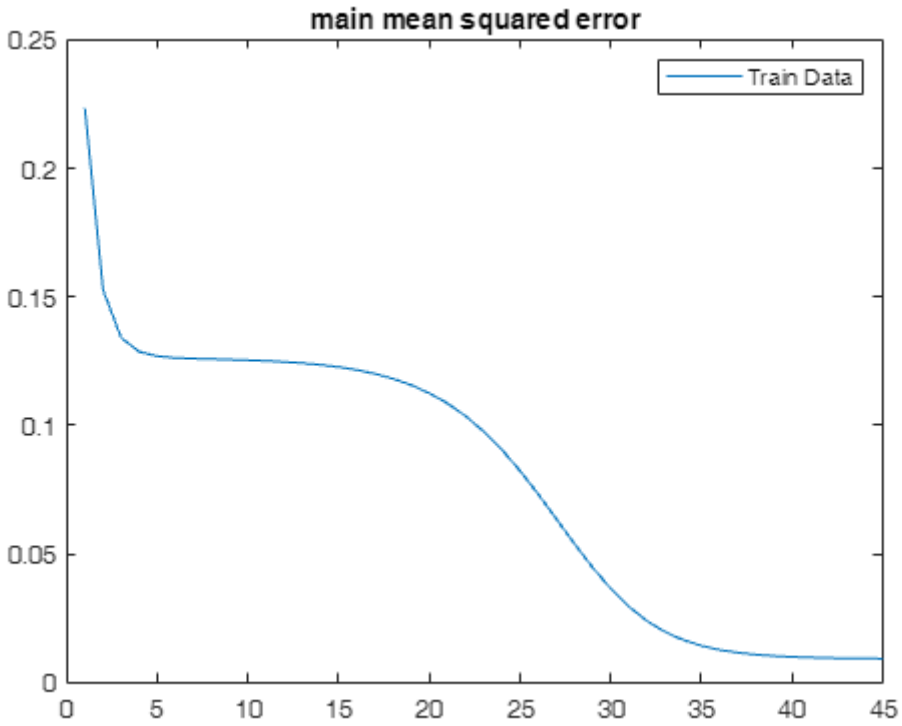
```
% sigmoidGradient = @(z) sigmoid(z) .* (1 - sigmoid(z));


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
```

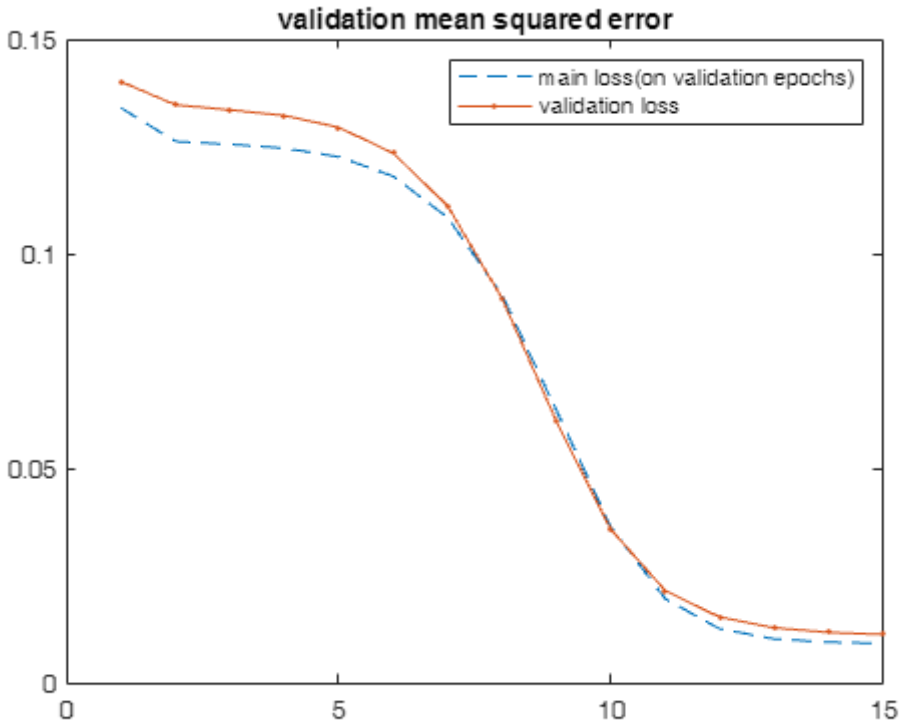
```
Epoch 21, Loss: 0.108588
Epoch 21, Loss validation: 0.111298
Epoch 24, Loss: 0.090540
Epoch 24, Loss validation: 0.089751
Epoch 27, Loss: 0.063680
Epoch 27, Loss validation: 0.061112
Epoch 30, Loss: 0.036557
Epoch 30, Loss validation: 0.035955
Epoch 33, Loss: 0.019765
Epoch 33, Loss validation: 0.021690
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")
```



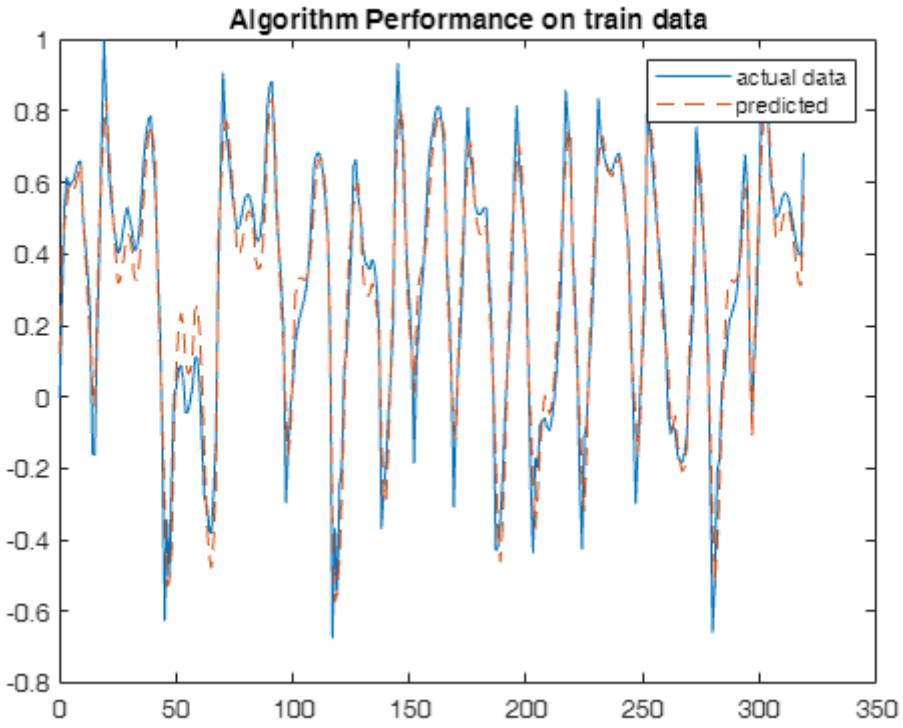

```
figure();
```

```
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"])
```



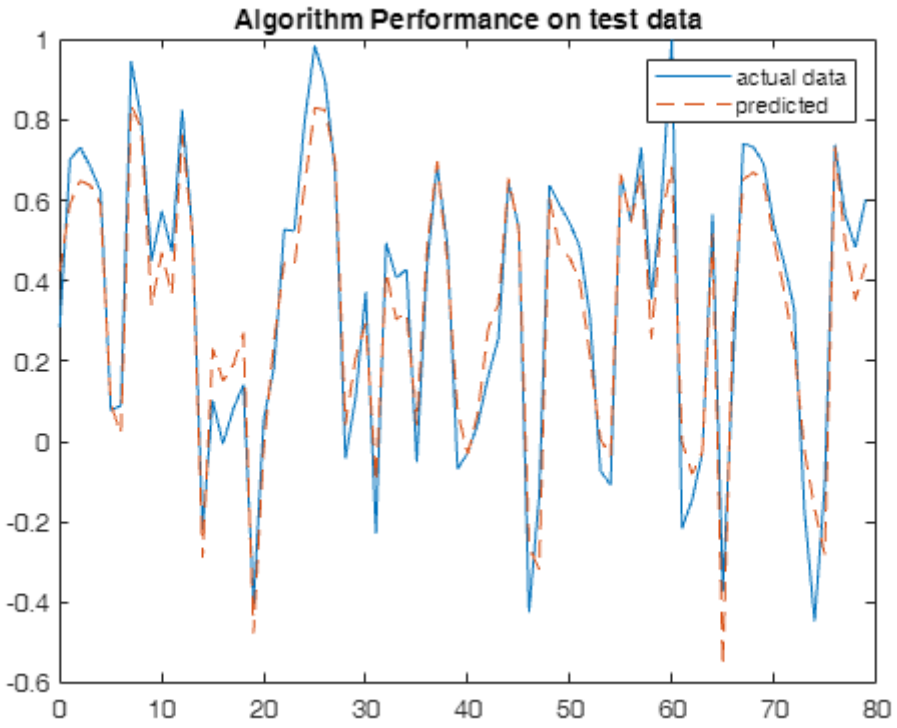

```
% Prediction function
% predict = @(X) sigmoid(X * W_ji.' + b1.') * W_kj.' + b2.';
```

```
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
```