University of Surrey

EEEM005 Assignment (NNSIMULATION)

AI & AI Programming

Faiz Ahmad Khan

August 2022

Abstract

This assignment uses the University of Wisconsin Breast Cancer Diagnosis Dataset to experiment with different parameters of Multi-Layer Shallow Neural Networks to optimise performance. The objective is to understand how to solve pattern recognition problems using backpropagation. In addition, experiments were conducted to distinguish between two equiprobable classes of overlapping two-dimensional Gaussian distributions.

Contents

Abstract 2
List of Figures 4
Executive Summary 1
Introduction 2
  Model 2
  Optimisers 2
    Resilient Backpropagation 2
    Levenberg-Marquardt Backpropagation 3
    Scaled Conjugate Gradient Descent 3
  Dataset 3
Experiments 4
  Experiment 1 4
  Experiment 2 5
  Experiment 3 6
  Experiment 4 7
Conclusion 8
References 9
Appendix 10
  Experiment 1 10
  Experiment 2-3 11
  Experiment 4 14

List of Figures

Figure 1: Neural Network Architecture 2
Figure 2: Error rate vs Epochs 4
Figure 3: Number of Classifiers vs Error 5
Figure 4: Ensemble performance for multiple nodes and epochs 5
Figure 5: Comparison of Optimizer Performance 6
Figure 6: Predicting decision boundary with MLP 7

Executive Summary

The objective of this assignment is to understand how to solve pattern recognition problems using backpropagation. To achieve this, the assignment uses the University of Wisconsin's Breast Cancer Dataset to experiment on a Multi-Layer Perceptron. Parameters such as the number of hidden nodes and epochs are optimised to achieve the best results. The number of hidden nodes varied from 2 to 32 and the number of epochs from 1 to 64. Each combination of node and epoch was averaged over 30 runs with equal random data splits among the training and test sets.

The breast cancer data is used to solve binary classification problems to predict malignant and benign tumours.

Next, an ensemble of individual classifiers is used to check if performance improvements can be achieved. The majority vote ensemble is repeated 30 times with equal random data splits and the average error is evaluated. The ensemble model uses between 3 and 25 individual classifiers with random starting weights. An odd number is selected to prevent ties in the voting. This experiment allows us to observe the performance change as the number of classifiers changes. The aim is to find the appropriate number of classifiers beyond which the increase in computational complexity does not return meaningful benefits.

Multiple optimisers were also tried to find the best choice for the classification task at hand. The optimizers studied are LM Backpropagation, Resilient Backpropagation and SCG Descent.

Additionally, experiments were conducted to distinguish between two equiprobable classes of Gaussian distribution with an overlapping circular decision boundary with the Multi-Layer Perceptron.

Introduction

The use of Neural Networks has become synonymous with Artificial Intelligence. A majority of industrial and research work is now dedicated to the development and optimisation of neural networks. These networks are modelled on the neurons and synapses of the brain. This report examines the use of multi-layer perceptron networks on a breast cancer dataset.

Model

A multilayer perceptron is a type of feed forward Neural Network. It consists of input and output layers as well as a single or multiple hidden layers. In this assignment we use the MATLAB Neural Network toolbox to create a model with a single hidden layer with multiple number of Neurons. These are the simplest model of neural networks. It has been proved that an MLP with a single layer can approximate any mathematical function[1]. The probabilities are evaluated through the use of a sigmoid function. The neural network generated is modelled in Figure 1.

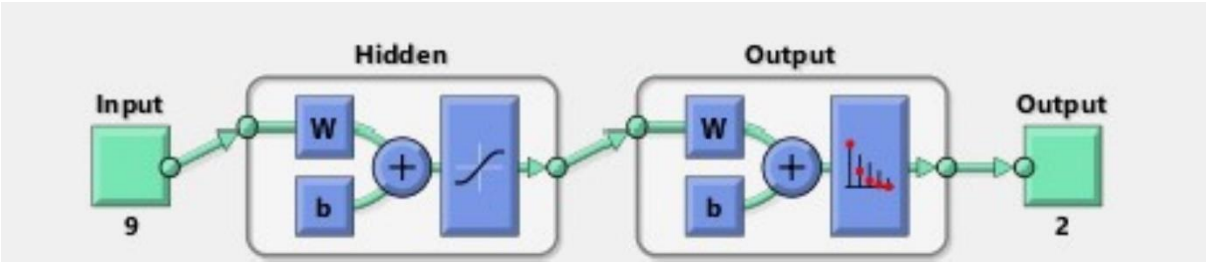

Figure 1: Neural Network Architecture

Steepest Gradient descent is used for training an MLP as it is like a numerical first order optimization problem. This algorithm is used to find the minima of a function. The algorithm takes gradual steps in the direction of the steepest descent.

$$\Delta W = -\eta \Delta L(w) | w = w_k \quad (1)$$

The learning rate is denoted by η and gradient by $\Delta L(w)$. During training, the loss is accumulated across all the epochs and the weights are updated once every epoch. A disadvantage is that this process is slow for large datasets, thus stochastic gradient descent (SGD) is used. Here, the weights are updated after each data point.

Optimisers

Resilient Backpropagation

This is a first order optimisation algorithm. In this method, the weights are updated in a similar manner to the steepest gradient descent. It only takes into account the sign of the partial derivative and is not dependant on each weight. For each weight if there is a change in the sign of the partial derivative compared to the last epoch, the weight is then multiplied by a

factor η^- which is less than one and if the sign remains the same then it is multiplied by η^+ which is greater than 1. Thus, if two gradients are in the same direction, the update is accelerated, if the directions are opposite then decelerated.[2]

Levenberg-Marquardt Backpropagation

This is also known as the damped least squares method and is often used in least squares curve fitting. It one of the fastest weight update mechanisms. The sum of square errors is used as a loss function. This method interposes between the Gauss-Newton numeric algorithm and the gradient descent method. It is more robust than the Gauss-Newton algorithm but is slower. When the weights are far from a local minimum it uses gradient descent. Once the weights are closer to a minima the much faster Newton's method is used. LM method takes advantage of both the Hessian and the gradient.[3]

Scaled Conjugate Gradient Descent

It is an iterative algorithm often used to solve optimisation problems. Here, a set of conjugate bases are used to approximate the steps in Newtons Second Order. This is done in place of calculating the inverse Hessian. The basic idea is to combine the model-trust region approach used in the Levenberg-Marquardt algorithm, with the conjugate gradient approach.[4]

Dataset

The dataset used in this report is the University of Wisconsin Breast Cancer Dataset. It has 669 records of different measures of tumours. 65.5% of the data is classified as benign and 34.5% as Malignant.

Another dataset is generated using two Gaussian Distributions. One with mean μ1 = [0, 0] and variance σ_1^2 = 1. The other one has mean μ2 = [2, 0] and variance $\sigma_2^2 = 4$ The decision boundary of the two gaussians is a circle centred at [-2/3, 0] and radius 2.34.

Experiments

Experiment 1

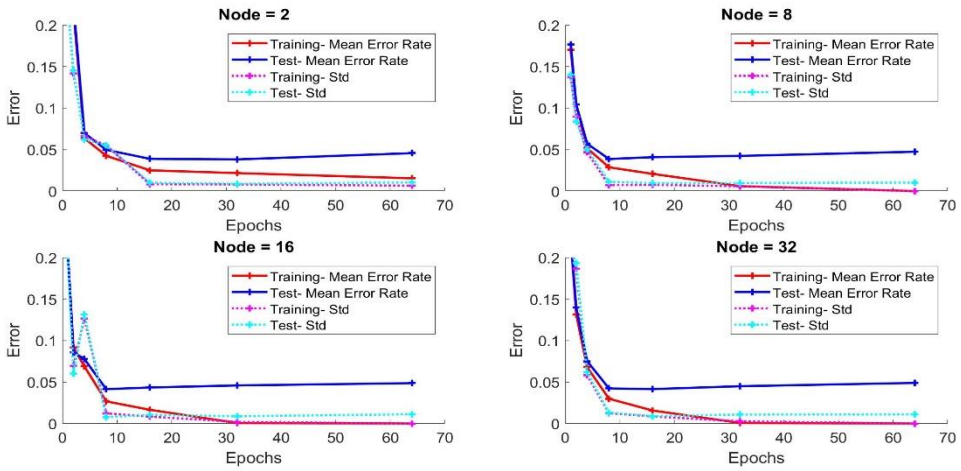

Figure 2: Error rate vs Epochs

In this experiment, the MLP is trained on the breast cancer dataset. The number of neurons is selected from the set [2, 8, 16, 32] and the number of epochs from the set [1, 2, 4, 8, 16, 32, 64]. Each combination of nodes and epochs is selected 30 times with random 50-50 data splits. The mean error rate and standard deviation (Std) are plotted in Figure 2. It can be observed that for the first few epochs the error rate is high irrespective of the number of neurons. This is called underfitting as there is not enough training data for this stage. As the training process proceeds and the number of epochs increases, so does the performance. In most cases around the 8^{th} epoch the error is at its lowest. After this point, the test and train mean errors start to deviate and this indicated that overfitting is starting to occur. At higher epochs the model is highly overfitted.

It can also be observed that as the number of neurons used is increased, the error tends to drop and converge more decisively. A disadvantage of using many nodes is that the training time increases as well with no meaningful performance advantage. From the experiments it was observed that using 8 nodes with 8 epochs provides good performance while maintaining a balance between over fitting and under fitting.

Experiment 2

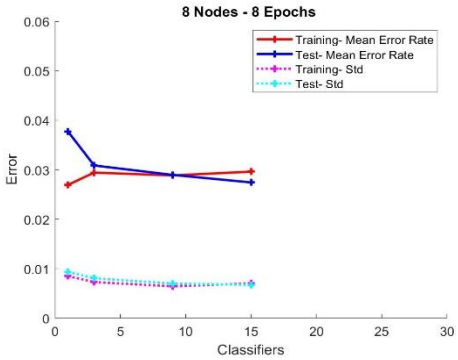

Figure 3: Number of Classifiers vs Error

The ideal parameters selected from experiment 1 were, number of nodes = 8 and epochs = 8. A Hard voting ensemble model was created using these parameters and a 30-iteration average was used to estimate the performance. The mean error rate was about 0.028. From Figure 3 it can be observed that as the number of classifiers is increased the error rates start to drop. At about 9 classifiers, the training and testing errors converge.

From Figure 4 the effect of the number of epochs and nodes in relation to the ensemble performance can be observed. The network was tested with combinations of 2,8 and 32 nodes and 16, 8 and 4 epochs with 15 classifiers. As the number of hidden nodes are increased, so does the ensemble classifier performance. The error rate is the lowest when there are 32 nodes, but the difference between error rate for 8 and 32 nodes is minute. Thus, to reduce complexity, 8 nodes are the ideal. An interesting observation can be made with the test mean error rate at 8 nodes. Here it can be observed that as the number of epochs is increased from 8 to 16, the error rate increases slightly. This may be due to overfitting in the scenario.

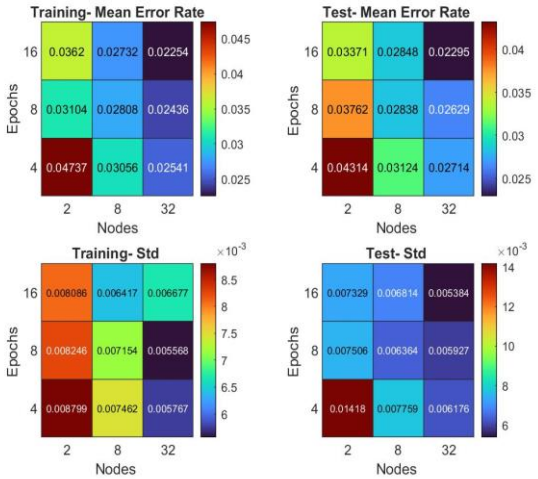

Figure 4: Ensemble performance for multiple nodes and epochs

Through these experiments it is observed that the ensemble models produce better overall performance than individual classifiers. Even at lower training epochs the ensemble classifier outperforms an individual classifier trained at higher epochs.

Experiment 3

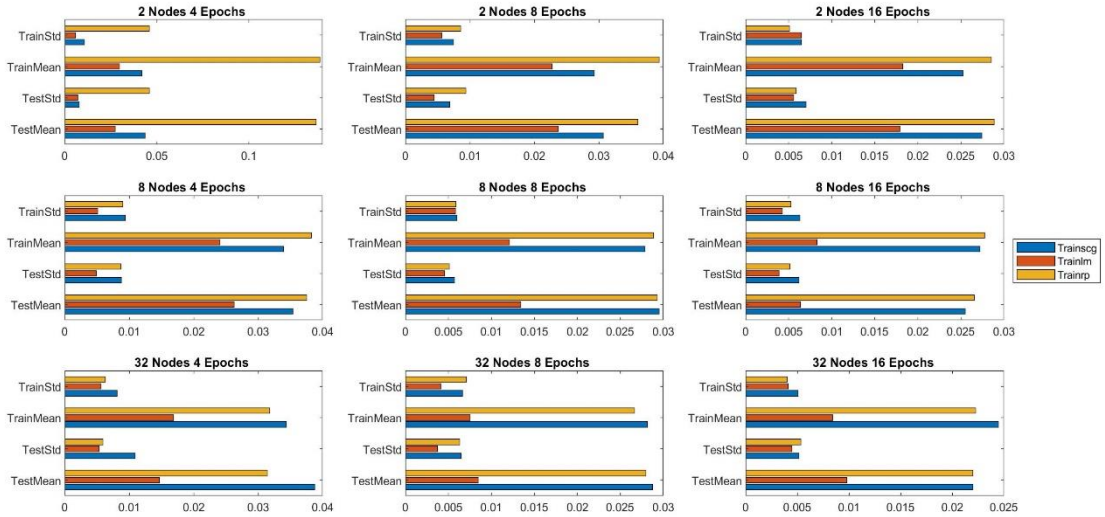

Figure 5: Comparison of Optimizer Performance

In this experiment, multiple optimizers were evaluated in a similar fashion to the previous experiments. The optimizers used were resilient backpropagation (trainrp), Levenberg-Marquardt (trainlm) and scaled conjugate gradient descent (trainscg). The error rates are recorded and plotted in Figure 5.

It can be observed from the plots that trainlm requires the least number of epochs to converge. Trainrp has the highest error along with taking the greatest number of epochs to converge. Trainlm exhibits the best performance followed by trainscg and finally by trainrp. Trainscg and trainlm both have a smooth learning curve unlike trainrp which is more oscillatory which might be attributed to the oscillatory nature of Newton's first order approximation. Trainlm is clearly the most appropriate optimizer for the task at hand. Trainscg can be selected as well especially considering the fact that trainlm is known to be slow and not suitable for complex networks.

Experiment 4

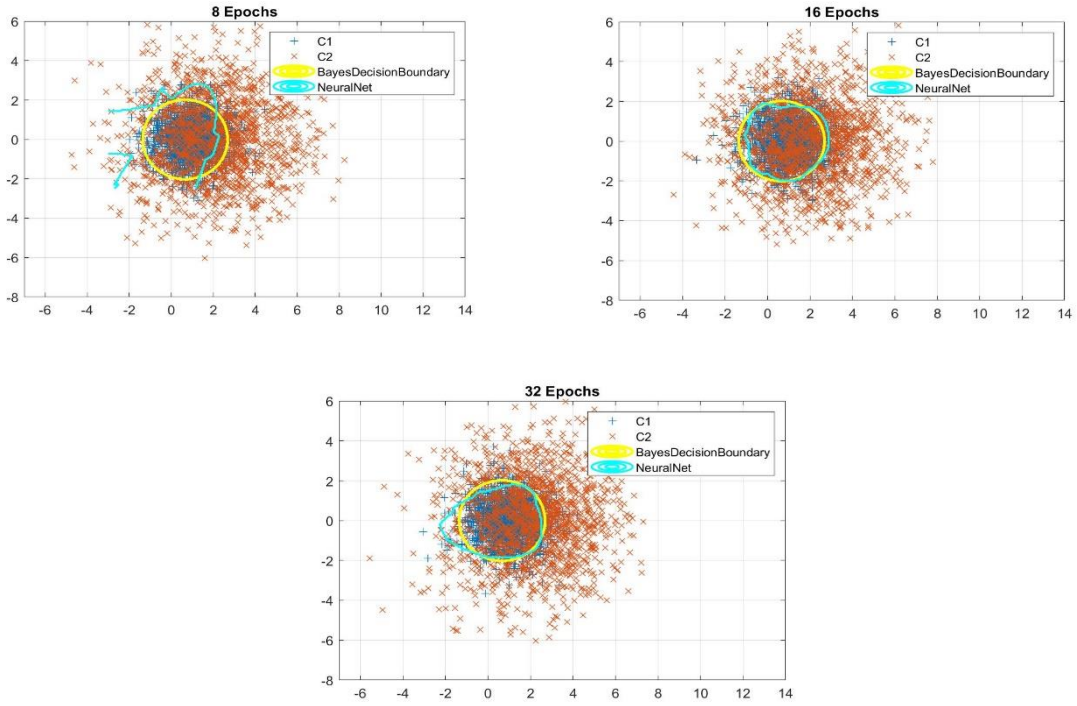

Figure 6: Predicting decision boundary with MLP

The goal of this experiment is to prove that given two gaussian datasets of with mean μ1 = [0, 0] and variance σ_1^2 = 1 and the other has mean μ2 = [2, 0] and variance $\sigma_2^2 = 4$, the decision boundary of the two gaussians is a circle centred at [-2/3, 0] and radius 2.34. Mathematically, the decision boundary can be plotted by utilizing the point where the probability is equal.

A soft voting ensemble of 15 classifiers is used to make a prediction at each point in a grid around the distribution in the 2D space. Training was carried out for 8, 16 and 32 epochs and is represented in Figure 6. From the figure it can be observed that as the number of epochs is increased the curve better fits the decision boundary. At 8 epochs there is observable underfitting as the boundary is very rough. Due to back propagation, some information loss always occurs and thus a completely accurate circular boundary cannot be achieved.

Conclusion

The results of the first experiment conclude that the best performance is achieved with 8 hidden nodes when trained for 8 epochs. Hyper parameters such as number of hidden nodes and epochs are a major influence on the performance of MLPs especially in regard to under and over fitting. A larger number of nodes or epochs also increases the computational overhead with no comparable benefit.

From the second experiment, it was observed that ensemble classifiers show significant performance improvement over individual classifiers, and that voting is a reliable way to improve performance and to increase generalization of a trained network. There exists an ideal number of classifiers for a given problem and using larger number of classifiers results in greater computational costs with no significant benefit.

From the third experiment it was observed that the Levenberg-Marquardt Backpropagation optimizer is the most suitable one for the task at hand. The simplicity of the dataset and network play a large factor in this.

In the final experiment, we observed decision boundaries of two equally probable Gaussian distributions were approximated successfully using MLP. We can see the effect of number of epochs on the accuracy of the predicted boundary.

References

[1] J. G. Attali and G. Pagès, “Approximations of Functions by a Multilayer Perceptron: a New Approach,” *Neural Networks*, vol. 10, no. 6, pp. 1069–1081, Aug. 1997, doi: 10.1016/S0893-6080(97)00010-5.

[2] M. Riedmiller, M. Riedmiller, and H. Braun, “RPROP - A Fast Adaptive Learning Algorithm,” *PROC. OF ISCIS VII), UNIVERSITAT*, 1992, Accessed: Aug. 16, 2022. [Online]. Available: http://130.203.136.95/viewdoc/summary?doi=10.1.1.52.4576

[3] K. Levenberg, “A method for the solution of certain non-linear problems in least squares,” *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944, doi: 10.1090/qam/10666.

[4] C. C. Pz, C. C. Pz, T. T. Jervis, T. T. Jervis, W. J. Fitzgerald, and W. J. Fitzgerald, “Optimization Schemes for Neural Networks,” *CAMBRIDGE UNIV. ENG. DEPT., U.K., TECH. REP. CUED/FINFENG/TR*, vol. 144, 1993, Accessed: Aug. 16, 2022. [Online]. Available: http://130.203.136.95/viewdoc/summary?doi=10.1.1.17.6623

Appendix

Experiment 1

```
clear
clc

load cancer_dataset.mat
inp = cancerInputs ;
targ = cancerTargets ;

nodes = [2 8 16 32];
trainFcn = 'trainscg';

epochs = [1 2 4 8 16 32 64];
iter = 30;

for node_num = 1: length(nodes)
    std_train = zeros(1,length(epochs));
    std_test = zeros(1,length(epochs));
    mean_train = zeros(1,length(epochs));
    mean_test = zeros(1,length(epochs));

    for epoch_num = 1: length(epochs)
        net.trainParam.showWindow = false ;
        net = patternnet(nodes(node_num), trainFcn);
        net.trainParam.epochs = epochs(epoch_num);

        net.divideParam.trainRatio = 50/100;
        net.divideParam.valRatio = 0;
        net.divideParam.testRatio = 50/100;

        for l = 1: iter
            net = init(net);
            [net,ttrain] = train(net,inp,targ);

            tgt = targ (1 ,:);
            testTarg = [tgt(ttrain.testInd); 1-tgt(ttrain.testInd)];
            trainTarg = [tgt(ttrain.trainInd); 1-tgt(ttrain.trainInd)];

            alloutput = net(inp);
            output = alloutput (1 ,:);
            trainOut = [output(ttrain.trainInd); 1-output(ttrain.trainInd)];
            testOut = [output(ttrain.testInd); 1-output(ttrain.testInd)];

            [er_train(l),a,b,c] = confusion(trainTarg,trainOut);
            [er_test(l),a,b,c] = confusion(testTarg,testOut);
            performance = perform(net,targ,alloutput);
        end

        mean_train(epoch_num) = mean(er_train);
        std_train(epoch_num) = std(er_train);
        mean_test(epoch_num) = mean(er_test);
        std_test(epoch_num) = std(er_test);
    end

figure(1)
subplot(2,2 ,node_num)
hold on

plot(epochs,mean_train,'r-+','linewidth',2)
plot(epochs,mean_test,'b-+','linewidth',2)
plot(epochs,std_train,'m:+','linewidth',2)
plot(epochs,std_test,'c:+','linewidth',2)
legend ('Training- Mean Error Rate ','Test- Mean Error Rate ','Training- Std','Test- Std')
xlabel ('Epochs')
ylabel ('Error')
t = 'Node = '+ string(nodes(node_num));
title (t)
ax = gca ;
ax.FontSize = 13;
axis([0 70 0 0.2])
end
```

Experiment 2-3

```
clear
clc

load cancer_dataset.mat
inputs = cancerInputs ;
targets = cancerTargets ;

nodes = [2 8 32];
epochs = [4 8 16];
num_classifiers = [1 3 9 15];
iter = 30;
optimisers = ["trainscg", "trainlm", "trainrp"];

std_train = zeros(1, length(num_classifiers));
std_test = zeros(1, length(num_classifiers));
mean_train = zeros(1, length(num_classifiers));
mean_test = zeros(1, length(num_classifiers));

for i = 1: length(num_classifiers)
    test_error = zeros(1, iter );
    train_error = zeros(1, iter );

    net = patternnet(8,"trainscg");
    net.trainParam.epochs = 8;

    for l=1: iter
        nets = {};
        net.divideParam.trainRatio = 50/100;
        net.divideParam.valRatio = 0;
        net.divideParam.testRatio = 50/100;
        net.trainParam.showWindow = false ;
        for j = 1: num_classifiers(i)
            net = init(net);
            [net , ttrain ] = train(net,inputs,targets);
            nets{j} = net ;
        end

        tgt = targets(1 ,:);
        trTarg = [tgt(ttrain.trainInd);1- tgt(ttrain.trainInd)];
        tsTarg = [tgt(ttrain.testInd);1- tgt(ttrain.testInd)];

        out = zeros(1, length(tgt));
        for j = 1: num_classifiers(i)
            net = nets{j};
            outs = net(inputs);
            out = out + round(outs(1 ,:));
        end

        out = (sign(out/num_classifiers(i) - 0.5) /2) + 0.5;

        trOut = [out(ttrain.trainInd); 1- out(ttrain.trainInd)];
        tsOut = [out(ttrain.testInd); 1-out(ttrain.testInd)];
        [train_error(l),a,b,c] = confusion( trTarg , trOut );
        [test_error(l),a,b,c] = confusion( tsTarg , tsOut );
    end

    mean_train(i) = mean( train_error );
    std_train(i) = std( train_error );
    mean_test(i) = mean( test_error );
    std_test(i) = std( test_error );
end

figure (1)
hold on
plot(num_classifiers , mean_train ,'r-+','linewidth' ,2)
plot(num_classifiers , mean_test ,'b-+','linewidth' ,2)
plot(num_classifiers , std_train ,'m:+','linewidth' ,2)
plot(num_classifiers , std_test ,'c:+','linewidth' ,2)
legend('Training- Mean Error Rate ','Test- Mean Error Rate ','Training- Std','Test- Std')
xlabel(' Classifiers ')
ylabel('Error')

title ('8 Nodes - 8 Epochs ')
axis ([0 30 0 0.06])
axis = gca ;
axis . FontSize = 11;

% Changing nodes
mean_train = zeros(length(nodes), length(epochs));
std_train = zeros(length(nodes), length(epochs));
mean_test = zeros(length(nodes), length(epochs));
std_test = zeros(length(nodes), length(epochs));

for i = 1: length(nodes)
for j = 1: length(epochs)
train_error = zeros(1, iter);
test_error = zeros(1, iter);

net = patternnet(nodes(i),"trainscg");
net.trainParam.epochs =epochs(j);

```

```
for l=1: iter
nets = {};
net.trainParam.showWindow = false ;
net.divideParam.trainRatio = 50/100;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 50/100;

for k = 1:15
net = init(net);
[net ,a] = train(net ,inputs,targets);
nets{k} = net ;
end

[a, ttrain ] = train(net ,inputs,targets);

tgt = targets (1 ,:);
trTarg = [tgt(ttrain.trainInd);1- tgt(ttrain.trainInd)];
tsTarg = [tgt(ttrain.testInd);1- tgt(ttrain.testInd)];

out = zeros(1, length(tgt));

for k = 1:15
net = nets{k};
outs = net(inputs);
out = out + round(outs(1 ,:));
end

out = (sign( out /15 - 0.5) /2) + 0.5;

trOut = [out(ttrain.trainInd ); 1- out(ttrain.trainInd)];
tsOut = [out(ttrain.testInd ); 1-out(ttrain.testInd)];

[train_error(l),a,b,c] = confusion(trTarg , trOut);
[test_error(l),a,b,c] = confusion(tsTarg , tsOut);
end
mean_train(i,j) = mean( train_error );
std_train(i,j) = std( train_error );
mean_test(i,j) = mean( test_error );
std_test(i,j) = std( test_error );
end
end
xv = {'2', '8', '32'};
yv = {'16','8','4'};

figure (2)
axis = gca ;
axis.FontSize = 11;
set(findall(gcf ,'type','line'),'linewidth' ,2);

subplot (2 ,2 ,1)
h = heatmap(xv ,yv , flipud(mean_train),'Colormap',turbo);
h.XLabel = 'Nodes';
h.YLabel = 'Epochs';
h.Title = ' Training- Mean Error Rate ';

subplot (2 ,2 ,2)
h = heatmap (xv ,yv , flipud(mean_test),'Colormap',turbo);
h.XLabel = 'Nodes ';
h.YLabel = 'Epochs ';
h.Title = 'Test- Mean Error Rate ';

subplot (2 ,2 ,3)
h = heatmap (xv ,yv , flipud(std_train),'Colormap',turbo);
h.XLabel = 'Nodes ';
h.YLabel = 'Epochs ';
h.Title = 'Training- Std ';

subplot (2 ,2 ,4)
h = heatmap (xv ,yv , flipud(std_test),'Colormap',turbo);
h.XLabel = 'Nodes ';
h.YLabel = 'Epochs ';
h.Title = 'Test- Std ';

mean_train = zeros( length( optimisers ), length(nodes )* length(epochs));
std_train = zeros( length( optimisers ), length(nodes )* length(epochs));
mean_test = zeros( length( optimisers ), length( nodes )* length(epochs));
std_test = zeros( length( optimisers ), length( nodes )* length(epochs));

% Changing classifiers .
for s = 1: length( optimisers )
for i = 1: length( nodes )
for j = 1: length(epochs)
train_error = zeros(1, iter );
test_error = zeros(1, iter );


net = patternnet(nodes(i), optimisers(s));
net.trainParam.epochs =epochs(j);


for l=1: iter
nets = {};
net.divideParam.trainRatio = 0.5;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0.5;
```

```
net.trainParam.showWindow = false ;
for k = 1:15
net = init( net );
[net ,a] = train(net ,inputs,targets);
nets {k} = net ;
end

[~, ttrain ] = train(net ,inputs,targets);

tgt = targets(1 ,:);
trTarg = [tgt(ttrain.trainInd);1- tgt(ttrain.trainInd)];
tsTarg = [tgt(ttrain.testInd);1- tgt(ttrain.testInd)];

out = zeros(1, length(tgt));
for k = 1:15
net = nets{k};
outs = net(inputs);
out = out + round( outs(1 ,:));
end

out = sign( out /15 - 0.5) /2 + 0.5;

trOut = [out(ttrain.trainInd); 1- out(ttrain.trainInd)];
tsOut = [out(ttrain.testInd); 1-out(ttrain.testInd)];

[ test_error(l),a,b,c] = confusion( tsTarg , tsOut );
[ train_error(l),a,b,c] = confusion( trTarg , trOut );

end
mean_train(s, 3*i -3+ j) = mean( train_error );
std_train(s, 3*i -3+ j) = std( train_error );
mean_test(s, 3*i -3+j) = mean( test_error );
std_test(s, 3*i -3+j) = std( test_error );

end
end
end

figure (3)
for i = 1:9
subplot (3,3 ,i)
X = categorical({ 'TrainMean ','TestMean ','TrainStd ','TestStd '});
Y = [ mean_train(:,i),mean_test(:,i),std_train(:,i), std_test(:,i)]';
barh(X,Y)
%ylim ([0 , 0.09])
ll = string ( nodes( ceil (i /3) )) + ' Nodes '+ string(epochs(i- ceil (i /3) *3+3) ) + ' Epochs ';
title (ll)
end
```

Experiment 4

```
clear

C1 = mvnrnd([1 0] ,[1 0; 0 1] , 3300/ 2) ;
C2 = mvnrnd([2 0] ,[4 0; 0 4] , 3300/ 2) ;

inputs = [ C1 ; C2 ]';

targets = [ones(size( C1 ,1) ,1) ,zeros(size( C1 ,1) ,1);zeros(size( C2 ,1) ,1) ,ones(size( C2 ,1),1)]';
trainFcn = 'trainscg';
figure(1)
plot( C1(: ,1) , C1(: ,2) ,'+')
hold on
plot( C2(: ,1) , C2(: ,2) ,'x')
axis equal
grid on

x1 =linspace(-4 , 6 , 300) ;
y1 =linspace(-4 , 4 , 300) ;


opt = zeros(length( x1 ),length( y1 ));
for iterate = 1:length( y1 )
    for j = 1:length( x1 )
        val1 = -1/2*log(det([1 0; 0 1]) ) - 1/2* transpose ([ y1( iterate ); x1( j) ] -[1;0]) * inv([1 0; 0 1]) *([ y1( iterate ); x1( j) ] -[1;0]) ;
        val2 = -1/2*log(det([4 0; 0 4]) ) - 1/2* transpose ([ y1( iterate ); x1( j) ] -[2;0]) * inv([4 0; 0 4]) *([ y1( iterate ); x1( j) ] -[2;0]) ;
        if val1 > val2
            opt( j, iterate ) = 1;
        end
    end
end

figure(1)
hold  on
contour( y1 , x1 , opt ,[0 ,1] ,'y','linewidth', 2)

y1 =linspace(-3 , 3 ,100) ;
x1 =linspace( -2.5 , 4 , 100) ;
z =zeros(length( x1 ),length( y1 ));

net = patternnet(8 ,"trainscg");
net.trainParam.epochs = 16;
nets = {};
net.divideParam.trainRatio = 300 / 3300 ;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 3000 / 3300 ;
net.trainParam.showWindow = false ;
for j = 1:8
net = init( net );
[ net , a] = train( net , inputs , targets );
nets{j} = net ;
end

for iterate = 1:length( y1 )
    for j = 1:length( x1 )
        outp = 0;
        for k = 1:5
            net = nets{ k};
            outps = net([ y1( iterate ); x1( j)]);
            outp = outp +round( outps(1 ,:) );
        end
        z( j, iterate ) =sign( outp /5 - 0.5) /2 + 0.5;
    end
end

figure(1)
hold on
contour( y1 , x1 , z ,[0 ,1] ,'c','linewidth', 1.5)
legend("C1","C2","BayesDecisionBoundary","NeuralNet")
axis([ -7 ,14 , -8 ,6])
```