# Project 1

Geoffrey Clark
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: gmclark1@asu.edu

Venkatavaradhan Lakshminarayanan
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: vvlakshm@asu.edu

Shubham Sonawani
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: sdsonawa@asu.edu

***Abstract*—In this project, we have shown the implementation of different back propagation algorithms to train neural networks. The training data consists of 2000 examples divided into two separate clusters (fig. 1) as stated in problem. Three training sets were used with the separation distance d varying between each set, The d values specified are 2, -4, and -8. Where as the separation between two clusters in each set decreases classification problem becomes much more difficult due to the nonlinearity of the problem. Using the Neural Net tool box of MATLAB, we have successfully trained multilayer perceptron networks as non linear classifiers for the three separation distances d equals 2, -4, and -8.**

## I. Introduction

In data analysis, a fundamental Property of many data sets is the presence of Nonlinear distributions and separations in the data. One way to solve these types of nonlinear classification problems[1], is to utilize neural networks with multiple perceptrons in a single layer. By analyzing the problem statement, we can see through visualization of training data that nonlinear data separation exists and therefore requites us to use a neural network trained for non linear binary classification. In this case we can not use single perceptron for training as it provide only a linear separations boundary. Thus, we have shown the implementation of state of the art back propagation and back propagation with momentum and Levenberg-Marquardt algorithms to solve this problem.

## II. Approach

### A. Generating Data

In order to generate clusters of 1000 data points, we have used Mersenne Twister [2] generator. Here the initial part of code generates cluster of 2000 training data sets and 1000 test data sets for each of the separation distances d equals 2,-4, and -8 units. Each data set consists of: an x coordinate, a y coordinate, and a binary value denoting the cluster the point belongs to. The data is first produced in the form of polar coordinates which are then converted into rectangular coordinates. The generated data clusters have width of 6 units and radius of 10 units. Here, we have taken care of data seeds for all training data sets. Furthermore, to maintain randomness in data generation, each training and test data sets have different seeds[3]. Generated clusters for different Separation distances can be visualized in fig. 1, fig. 2, fig. 3.

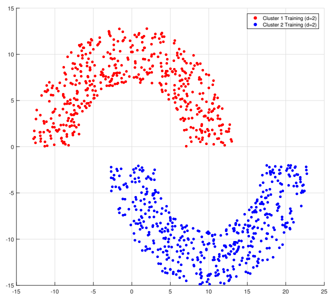

Fig. 1. Visualization of Clusters with separation of 2 units

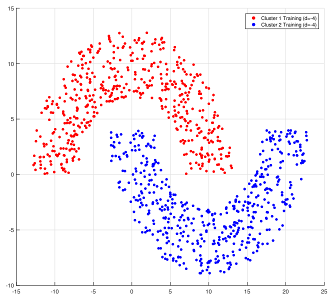

Fig. 2. Visualization of Clusters with separation of -4 units

### B. Training Data

We are using a three layer neural networks with first layer as input layer , second layer as a hidden layer and third as output layer. With the goal of producing a non linear classifier on the data shown in fig. 1:3, we trained the network using three different back propagation functions available in MATLAB

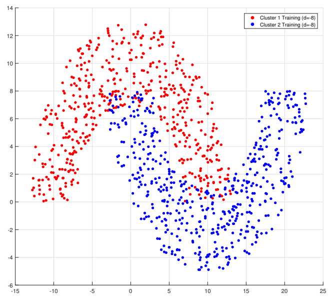

Fig. 3. Visualization of Clusters with separation of -8 units

neural net toolbox. For basic back propagation we have used 'traingd' (Gradient Descend Back Propagation)[4] [5], for back propagation with momentum we have used 'trainbpm' ( Back Propagation with Momentum Algorithm )[6] [7]and for Levenberg Marquardt we have used 'trainlm' (Levenberg Marquardt )[8] [9]. In this case, we have set the 75% of training data is for training and 25% is used for validation. Furthermore, We have trained the network with the different learning rate values of: 1.6, 0.9, and 0.2 for all three separation distances of 2, -4, -8. Lastly early stopping at 2000 iterations was used for all networks.

### C. Testing Data

Here, we have generated 1000 testing data points for clusters d=2, d=-4 and d=-8. fig. 4, fig. 5, and fig. 6 show the trained neural networks for the three different algorithms and three different learning rates. The performance curves for each neural network can be seen in fig. 7, fig. 8, and fig. 9. Below we discuss the algorithms' performance on the data.

### D. Experiment I

*1) Basic Back Propagation:* Now consider simple back propagation algorithm, we are using 'traingd'( Gradient Descend Algorithm)[5] available in MATLAB Neural Net toolbox [10]. We have set the number of units in hidden layer to 3. Here, we can visualize the response of single hidden layer neural network trained with different learning rates in fig. 4:9.

*2) Back Propagation with Momentum:* Here, We are using resilient back propagation algorithm available in MATLAB Neural Net tool box. Important parameter that we have to focused on is momentum rate of back propagation algorithm[6] [7]. Due to addition of momentum rate, there is attenuation in oscillation of gradient descent [6] . Thus, we have tested the output of neural network trained with back propagation algorithm having momentum rate of "0.9" and response can be visualized in terms of decision boundary in fig. 4:9

*3) Levenberg-Marquardt Back Propagation:* Levenberg Marquardt Back Propagation is optimal algorithm for least square estimation when we deal with non linear decision boundaries[8]. as per the given problem, we have used 'trainlm' (Levenberg Marquardt ) [9] parameter available in MATLAB neural net tool box. Response of neural net with Levenberg-Marquardt algorithm can be visualized in fig. 4:9.

### E. Experiment II

Initially we trained neural network with single hidden layer with three different back propagation algorithm and learning rates. however, in this part of experiment, we are restricting data set to cluster with separation distance of -8. Until now, we have not changed the number of units (neurons) in hidden layers which indirectly was restriction on performance of neural network to complex data set (d=-8). But by changing number of units in hidden layer to 2, 5 and 11 we can visualize the change and improvement in performance using learning curves and decision boundaries shown in fig. 10 and fig. 11

## III. RESULTS

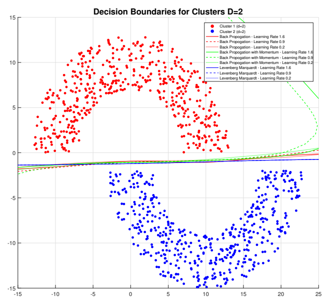

Fig. 4. Response of Neural Network in terms of decision boundary for cluster with separation distance of 2 units

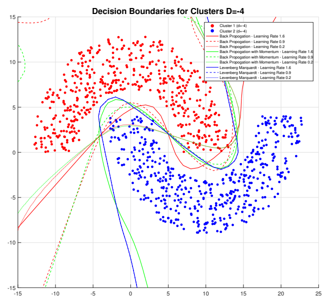

Fig. 5. Response of Neural Network in terms of decision boundary for cluster with separation distance of -4 units

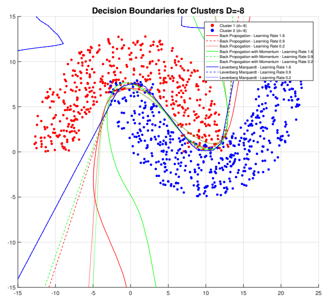

Fig. 6. Response of Neural Network in terms of decision boundary for cluster with separation distance of -8 units

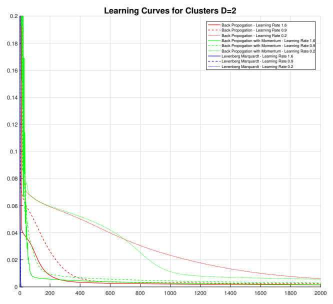

Fig. 7. Performance of three different algorithms for three different learning rates in terms of Learning Curve for data set with separation distance of 2 units

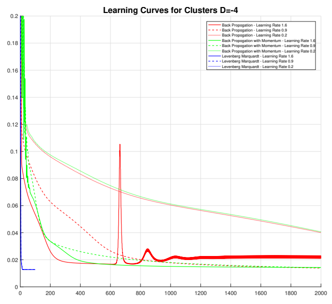

Fig. 8. Performance of three different algorithms for three different learning rates in terms of Learning Curve for data set with separation distance of 2 units

| | | D= 2 Units | | |
|---|---|---|---|---|
| | | Percentage of Correct Classification | | |
| | Learning Rate | Cluster 1 | Cluster 2 | Total |
| GD | 1.6 | 50.00 | 50.00 | 100.00 |
| | 0.9 | 50.00 | 50.00 | 100.00 |
| | 0.2 | 50.00 | 50.00 | 100.00 |
| GDM | 1.6 | 50.00 | 50.00 | 100.00 |
| | 0.9 | 50.00 | 50.00 | 100.00 |
| | 0.2 | 50.00 | 50.00 | 100.00 |
| LM | 1.6 | 50.00 | 50.00 | 100.00 |
| | 0.9 | 50.00 | 50.00 | 100.00 |
| | 0.2 | 50.00 | 50.00 | 100.00 |

Fig. 12. Confusion Matrix Summary of Clusters with Separation distance d= 2 for different conditions

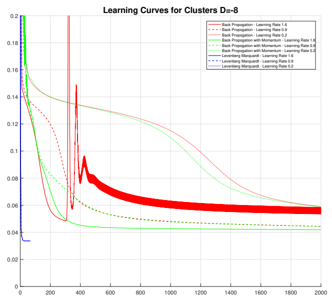

Fig. 9. Performance of three different algorithms for three different learning rates in terms of Learning Curve for data set with separation distance of 2 units

| | | D= -4 Units | | |
|---|---|---|---|---|
| | | Percentage of Correct Classification | | |
| | Learning Rate | Cluster 1 | Cluster 2 | Total |
| GD | 1.6 | 49.7 | 49.9 | 99.6 |
| | 0.9 | 49.9 | 49.9 | 99.8 |
| | 0.2 | 48.2 | 48.8 | 97 |
| GDM | 1.6 | 50.00 | 49.9 | 99.9 |
| | 0.9 | 50.00 | 49.9 | 99.9 |
| | 0.2 | 47.8 | 48.8 | 96.6 |
| LM | 1.6 | 49.9 | 49.9 | 99.8 |
| | 0.9 | 49.9 | 49.9 | 99.8 |
| | 0.2 | 49.9 | 49.9 | 99.8 |

Fig. 13. Confusion Matrix Summary of Clusters with Separation distance d= -4 for different conditions

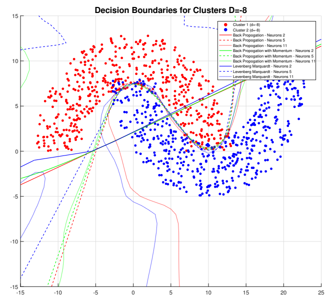

Fig. 10. Response of Neural Network in terms of decision boundary and different numbers of neurons for cluster with separation distance of -8 units

| | | D= -8 Units | | |
|---|---|---|---|---|
| | | Percentage of Correct Classification | | |
| | Learning Rate | Cluster 1 | Cluster 2 | Total |
| GD | 1.6 | 46.5 | 48.7 | 95.2 |
| | 0.9 | 46.9 | 47.6 | 94.5 |
| | 0.2 | 47.5 | 47.1 | 94.6 |
| GDM | 1.6 | 46.9 | 47.9 | 94.8 |
| | 0.9 | 46.2 | 48.0 | 94.2 |
| | 0.2 | 47.5 | 47.1 | 94.6 |
| LM | 1.6 | 46.7 | 47.8 | 94.5 |
| | 0.9 | 46.7 | 47.8 | 94.5 |
| | 0.2 | 46.7 | 47.8 | 94.5 |

Fig. 14. Confusion Matrix Summary of Clusters with Separation distance d= -8 for different conditions

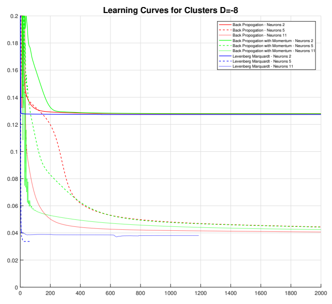

Fig. 11. Performance of neural network with three different algorithms and three different number of hidden neurons in terms of Learning Curve for data set with separation distance of -8 units

| | | d = -8 | | |
|---|---|---|---|---|
| | | Percentage of Correct Classification | | |
| | # of Neurons | Cluster 1 | Cluster 2 | Total |
| GD | 2 | 42.1 | 40.8 | 82.9 |
| | 5 | 46.9 | 47.6 | 94.5 |
| | 11 | 47.3 | 47.8 | 95.1 |
| GDM | 2 | 42.1 | 40.8 | 82.9 |
| | 5 | 46.2 | 48 | 94.2 |
| | 11 | 46.5 | 48 | 94.5 |
| LM | 2 | 42.1 | 40.5 | 82.6 |
| | 5 | 46.7 | 47.8 | 94.5 |
| | 11 | 47.0 | 48.1 | 95.1 |

Fig. 15. Confusion Matrix Summary of Clusters with Separation distance d= -8 for different number of Neurons

## IV. Conclusion

From this project, we understood how to implement back propagation algorithm with the merits and demerits of different algorithms such as back propagation with momentum and Levenberg-Marquardt algorithm. A three layer neural network will solve the problem as long as the data is linearly separable, which means that the neural network model acts as a linear classifier. But as the data starts to overlap deep, it might not have 100% classification as depicted by the confusion matrix. Increasing the number of hidden layers might increase the performance but comes with a cost that the model might overfit the data and lose generality.

From this project, we also learned how different learning rates affects the performance of the neural network. Another important factor witnessed during the training of the network with different learning rates was the behavior of the cost function minimization. For certain learning rates, the training algorithms had high error while approaching global minima and then dropped again for some learning rates. This shows that early stopping proved that it is an efficient way to yield better results. Even more sophisticated techniques like recurrent neural networks might improve the results by a significant standard.

## References

[1] H. Larochelle, "Back propagation," date last accessed 15-oct-2017. [Online]. Available: https://www.youtube.com/watch?v=_KoWTD8T45Q&list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH&index=13

[2] A. Jagannatam, "Mersenne twister  a pseudo random number generator and its variants," 2017.

[3] MATLAB, "Control random number generation," date last accessed 15-oct-2017. [Online]. Available: https://www.mathworks.com/help/matlab/ref/rng.html?searchHighlight=rng&s_tid=doc_srchtitle

[4] P. Baldi, "Gradient descent learning algorithm overview: a general dynamical systems perspective," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 182–195, Jan 1995.

[5] MATLAB, "Gradient descend back propagation algorithm," date last accessed 15-oct-2017. [Online]. Available: https://www.mathworks.com/help/nnet/ref/traingd.html

[6] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the rprop algorithm," in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591 vol.1.

[7] MATLAB, "Resilient back propagation algorithm," date last accessed 15-oct-2017. [Online]. Available: https://www.mathworks.com/help/nnet/ref/trainrp.html

[8] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963. [Online]. Available: https://doi.org/10.1137/0111030

[9] MATLAB, "Levenberg-marquardt algorithm," date last accessed 15-oct-2017. [Online]. Available: https://www.mathworks.com/help/nnet/ref/trainlm.html

[10] Matlab, "Multilayer neural networks and back propagation," (Date last accessed 15-oct-2017). [Online]. Available: https://www.mathworks.com/help/nnet/ug/multilayer-neural-networks-and-backpropagation-training.html

```matlab
% Project 1 Clark Lakshminarayanan Sonawani
clear all
close all

% Generate Clusters

% % Training and Testing clusters for d = 2
% [Train1D2,Train2D2,Test1D2,Test2D2]=GenerateClusters(2);
% % Training and Testing clusters for d = -4
% [Train1Dn4,Train2Dn4,Test1Dn4,Test2Dn4]=GenerateClusters(-4);
% % Training and Testing clusters for d = -8
% [Train1Dn8,Train2Dn8,Test1Dn8,Test2Dn8]=GenerateClusters(-8);
r=10;
w=6;

% Seeds
seed1=1;    seed5=5;
seed2=2;    seed6=6;
seed3=3;    seed7=7;
seed4=4;    seed8=8;

% Cluster 1 Training (Rho-Magnitude Theta-angle[rad])
rng(seed1,'twister');
Cluster1RhoTrain = (r-w/2)+w*rand(1,1000);
rng(seed2,'twister');
Cluster1ThetaTrain = pi*rand(1,1000);

% Cluster 2 Training (Rho-Magnitude Theta-angle[rad])
rng(seed3,'twister');
Cluster2RhoTrain = (r-w/2)+w*rand(1,1000);
rng(seed4,'twister');
Cluster2ThetaTrain = -pi*rand(1,1000);

% Cluster 1 Testing (Rho-Magnitude Theta-angle[rad])
rng(seed5,'twister');
Cluster1RhoTest = (r-w/2)+w*rand(1,500);
rng(seed6,'twister');
Cluster1ThetaTest = pi*rand(1,500);

% Cluster 2 Testing (Rho-Magnitude Theta-angle[rad])
rng(seed7,'twister');
Cluster2RhoTest = (r-w/2)+w*rand(1,500);
rng(seed8,'twister');
Cluster2ThetaTest = -pi*rand(1,500);

% Convert to Carteasian Coordinate system
[Cluster1XTrain, Cluster1YTrain] =
 pol2cart(Cluster1ThetaTrain,Cluster1RhoTrain);
[Cluster2XTrain, Cluster2YTrain] =
 pol2cart(Cluster2ThetaTrain,Cluster2RhoTrain);
[Cluster1XTest, Cluster1YTest] =
 pol2cart(Cluster1ThetaTest,Cluster1RhoTest);
```

```
[Cluster2XTest, Cluster2YTest] =
 pol2cart(Cluster2ThetaTest,Cluster2RhoTest);

Cluster2XTrain=Cluster2XTrain+r;
Cluster2XTest=Cluster2XTest+r;
% Shift Cluster2 d=2
d=2;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1D2=[Cluster1XTrain;Cluster1YTrain];
Train2D2=[Cluster2XTrain;Cluster2YTrain];
Test1D2=[Cluster1XTest;Cluster1YTest];
Test2D2=[Cluster2XTest;Cluster2YTest];
% [C1Train,C2Train,C1Test,C2Test]
% Shift Cluster2 d=-4
d=-4;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1Dn4=[Cluster1XTrain;Cluster1YTrain];
Train2Dn4=[Cluster2XTrain;Cluster2YTrain];

Test1Dn4=[Cluster1XTest;Cluster1YTest];
Test2Dn4=[Cluster2XTest;Cluster2YTest];


% Shift Cluster2  d=-8
d=-8;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1Dn8=[Cluster1XTrain;Cluster1YTrain];
Train2Dn8=[Cluster2XTrain;Cluster2YTrain];

Test1Dn8=[Cluster1XTest;Cluster1YTest];
Test2Dn8=[Cluster2XTest;Cluster2YTest];


% % Organize Data in Clusters
TrainData={Train1D2,Train2D2;
           Train1Dn4,Train2Dn4;
           Train1Dn8,Train2Dn8};

TestData={Test1D2,Test2D2;
          Test1Dn4,Test2Dn4;
          Test1Dn8,Test2Dn8};



% Plot Clusters
```

```
fig1=figure(1);
fig1.Renderer='Painters';
set(fig1,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1D2(1,:),Test1D2(2,:),20,'r','filled')
scatter(Test2D2(1,:),Test2D2(2,:),20,'b','filled')

fig2=figure(2);
fig2.Renderer='Painters';
set(fig2,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1Dn4(1,:),Test1Dn4(2,:),20,'r','filled')
scatter(Test2Dn4(1,:),Test2Dn4(2,:),20,'b','filled')

fig3=figure(3);
fig3.Renderer='Painters';
set(fig3,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1Dn8(1,:),Test1Dn8(2,:),20,'r','filled')
scatter(Test2Dn8(1,:),Test2Dn8(2,:),20,'b','filled')

fig4=figure(4);
fig4.Renderer='Painters';
set(fig4,'units','points','position',[860,200,700,600])

fig5=figure(5);
fig5.Renderer='Painters';
set(fig5,'units','points','position',[860,200,700,600])

fig6=figure(6);
fig6.Renderer='Painters';
set(fig6,'units','points','position',[860,200,700,600])

fig7=figure(7);
fig7.Renderer='Painters';
set(fig7,'units','points','position',[860,200,700,600])
hold on;grid on;
scatter(Test1Dn8(1,:),Test1Dn8(2,:),20,'r','filled')
scatter(Test2Dn8(1,:),Test2Dn8(2,:),20,'b','filled')

fig8=figure(8);
fig8.Renderer='Painters';
set(fig8,'units','points','position',[860,200,700,600])

% Setup Experiment 1
Test1.Algorithms={'traingd','traingdm','trainlm'};
Test1.C={'-r','--r',':r';'-g','--g',':g';'-b','--b',':b'};
Test1.d=[2,-4,-8];
Test1.Lrate=[ 1.6,0.9,0.2];
Test1.Nneurons=5;
Test1.xvec=(-15:1:25);
Test1.yvec=(15:-1:-15);
Test1.grid=zeros(length(Test1.yvec),length(Test1.xvec));
[Test1.X,Test1.Y]=meshgrid(Test1.xvec,Test1.yvec);
```

```
% Train Experiment1
for n=1:length(Test1.Algorithms)
    for o=1:length(Test1.Lrate)
        for m=1:length(Test1.d)
            rng(2);
            % prep training input
            y1 = ones(1,1000);
            y2 = zeros(1,1000);
            train_set = [TrainData{m,1},TrainData{m,2}];
            order = randperm(2000);
            train_set = train_set(:,order);
            target = [y1,y2];
            target = target(order);

            % setup net
            net1{m,n,o} =
 feedforwardnet(Test1.Nneurons,Test1.Algorithms{n});
            net1{m,n,o} = configure(net1{m,n,o},train_set,target);
            net1{m,n,o}.trainParam.lr = Test1.Lrate(o);
            net1{m,n,o}.trainParam.epochs=2000;
            net1{m,n,o}.divideParam.trainRatio = 0.75;
            net1{m,n,o}.divideParam.valRatio = 0.25;
            net1{m,n,o}.divideParam.testRatio = 0.0;
            net1{m,n,o}.trainParam.max_fail=2000;
            if m==2
                net1{m,n,o}.trainParam.mc = 0.9;
            end

            % training algorithm
            [net1{m,n,o},TR] = train(net1{m,n,o},train_set,target);
            train_op = net1{m,n,o}(train_set);


            % testing algorithm
            order = randperm(1000);
            target=[ones(1,500) zeros(1,500)];
            target=target(:,order);
            testing_set = [TestData{m,1},TestData{m,2}];
            testing_set = testing_set(:,order);
            test_op = net1{m,n,o}(testing_set);

            % Boundary Function
            for r=1:length(Test1.xvec)
                for t=1:length(Test1.yvec)
                   classifierGrid(t,r)=net1{m,n,o}
([Test1.xvec(r);Test1.yvec(t)]);
                end
            end

            % Plot Boundaries
            figure(m);
            hold on;grid on;
            contour(Test1.X,Test1.Y,classifierGrid-.5,[0
 0],Test1.C{n,o},'lineWidth',1)
```

```
            xlim([-15 25]);
            ylim([-15 15]);

            %Plot Performance
            q=m+3;
            figure(q);
            hold on;grid on;
            plot(TR.epoch,TR.vperf,Test1.C{n,o},'lineWidth',1)
            xlim([0 2000]);
            ylim([0 .2]);

            for i=1:1000
                 if(test_op(i)>=0.5)
                    test_op(i)=1;
                 else
                     test_op(i)=0;
                 end
            end


            plotconfusion(target,test_op);


            pause;
        end
    end
end


% Setup Experiment 2
Test2.Algorithms={'traingd','traingdm','trainlm'};
Test2.C={'-r','--r',':r';'-g','--g',':g';'-b','--b',':b'};
Test2.d=[2,-4,-8];
Test2.Lrate=0.9;
Test2.Nneurons=[2 5 11];
Test2.xvec=(-15:1:25);
Test2.yvec=(15:-1:-15);
Test2.grid=zeros(length(Test2.yvec),length(Test2.xvec));
[Test2.X,Test2.Y]=meshgrid(Test2.xvec,Test2.yvec);

% Train Experiment2
for n=1:length(Test2.Algorithms)
    for o=1:length(Test2.Nneurons)
        for m=3:length(Test2.d)
            rng(2);
            % prep training input
            y1 = ones(1,1000);
            y2 = zeros(1,1000);
            train_set = [TrainData{m,1},TrainData{m,2}];
            order = randperm(2000);
            train_set = train_set(:,order);
            target = [y1,y2];
            target = target(order);
            % setup net
            net2{m,n,o} =
 feedforwardnet(Test2.Nneurons(o),Test2.Algorithms{n});
            net2{m,n,o} = configure(net2{m,n,o},train_set,target);
            net2{m,n,o}.trainParam.lr = Test2.Lrate;
            net2{m,n,o}.trainParam.epochs=2000;
            net2{m,n,o}.divideParam.trainRatio = 0.75;
            net2{m,n,o}.divideParam.valRatio = 0.25;
            net2{m,n,o}.divideParam.testRatio = 0.0;
            net2{m,n,o}.trainParam.max_fail=2000;
            if m==2
                net2{m,n,o}.trainParam.mc = 0.9;
            end

            % training algorithm
            [net2{m,n,o},TR] = train(net2{m,n,o},train_set,target);
            train_op = net2{m,n,o}(train_set);


            % testing algorithm
            order = randperm(1000);
            target=[ones(1,500) zeros(1,500)];
            target=target(:,order);
            testing_set = [TestData{m,1},TestData{m,2}];
            testing_set = testing_set(:,order);
            test_op = net2{m,n,o}(testing_set);

            % Boundary Function
            for r=1:length(Test2.xvec)
                for t=1:length(Test2.yvec)
                   classifierGrid(t,r)=net2{m,n,o}
([Test2.xvec(r);Test2.yvec(t)]);
                end
            end

            % Plot Boundaries
            figure(7);
            hold on;grid on;
            contour(Test2.X,Test2.Y,classifierGrid-.5,[0
 0],Test2.C{n,o},'lineWidth',1)
            xlim([-15 25]);
            ylim([-15 15]);

            %Plot Performance
            figure(8)
            hold on;grid on;
            plot(TR.epoch,TR.vperf,Test2.C{n,o},'lineWidth',1)
            xlim([0 2000]);
            ylim([0 .2]);
            for i=1:1000
                 if(test_op(i)>=0.5)
                    test_op(i)=1;
                 else
                     test_op(i)=0;
                 end
            end


            plotconfusion(target,test_op);
            pause;
        end
    end
end



% Save Graphs
% Experiment 1
figure(1)
title('Decision Boundaries for Clusters D=2','FontSize',15)
ldg1=legend( 'Cluster 1 (d=2)','Cluster 2 (d=2)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg1.FontSize=7;
print('-painters','-depsc','Exp1_DB2')

figure(2)
title('Decision Boundaries for Clusters D=-4','FontSize',15)
ldg2=legend( 'Cluster 1 (d=-4)','Cluster 2 (d=-4)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg2.FontSize=7;
print('-painters','-depsc','Exp1_DBn4')

figure(3)
title('Decision Boundaries for Clusters D=-8','FontSize',15)
ldg3=legend( 'Cluster 1 (d=-8)','Cluster 2 (d=-8)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg3.FontSize=7;
print('-painters','-depsc','Exp1_DBn8')

figure(4)
```

```
title('Learning Curves for Clusters D=2','FontSize',15)
ldg4=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg4.FontSize=7;
print('-painters','-depsc','Exp1_LC2')

figure(5)
title('Learning Curves for Clusters D=-4','FontSize',15)
ldg5=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg5.FontSize=7;
print('-painters','-depsc','Exp1_LCn4')

figure(6)
title('Learning Curves for Clusters D=-8','FontSize',15)
ldg6=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg6.FontSize=7;
print('-painters','-depsc','Exp1_LCn8')

% Experiment 2
figure(7)
title('Decision Boundaries for Clusters D=-8','FontSize',15)
ldg7=legend( 'Cluster 1 (d=-8)','Cluster 2 (d=-8)',....
        'Back Propogation - Neurons 2','Back Propogation - Neurons
 5','Back Propogation - Neurons 11',....
        'Back Propogation with Momentum - Neurons 2','Back Propogation
 with Momentum - Neurons 5','Back Propogation with Momentum - Neurons
 11',....
        'Levenberg Marquardt - Neurons 2','Levenberg Marquardt -
 Neurons 5','Levenberg Marquardt - Neurons 11')
ldg7.FontSize=7;
print('-painters','-depsc','Exp2_DBn8')

figure(8)
title('Learning Curves for Clusters D=-8','FontSize',15)
```

```
ldg8=legend(....
        'Back Propogation - Neurons 2','Back Propogation - Neurons
 5','Back Propogation - Neurons 11',....
        'Back Propogation with Momentum - Neurons 2','Back Propogation
 with Momentum - Neurons 5','Back Propogation with Momentum - Neurons
 11',....
        'Levenberg Marquardt - Neurons 2','Levenberg Marquardt -
 Neurons 5','Levenberg Marquardt - Neurons 11')
ldg8.FontSize=7;
print('-painters','-depsc','Exp2_LCn8')
```

*Published with MATLAB® R2017a*