Use of multilayer perceptron network trained by backpropagation algorithm and its variants to solve a nonlinear classification problem

Nanda Kishore Mallapragada, Sai Keerthy Kakarla, Venkata Yuktal Bobba
Arizona State University

***Abstract*— Classification of data into various categories is an essential step in Data Processing and Data Visualization. Multilayer Perceptron (MLP) Networks are very much used for Data classification. Unlike single layer networks, MLPs are nonlinear classifiers having hidden layers which will help in classification to be nonlinear [1]. In this project, we train an MLP using Back-propagation (BP) algorithm and its variants to perform a nonlinear classification on doublemoon data.**

I. INTRODUCTION

Classification is a process of segregating information into various categories. Multilayer networks are more powerful than single-layer networks. For instance, a two-layer network having a sigmoid first layer and a linear second layer can be trained to approximate most functions arbitrarily well. Due to the presence of multilayers in a single network they become efficient nonlinear classifiers of the data. To train these multilayer neural networks, we use the most efficient algorithm called the backpropagation.

A. Backpropagation

The backpropagation algorithm is a generalization of Least Mean Square (LMS) algorithm. Backpropagation computes the gradients from output to input to update the weights and biases for the hidden layers [2]. The BP algorithm uses Stochastic Gradient Descent (SGD) to update the parameters.

$$\theta=\theta-\eta\cdot\nabla J(\theta; x(i); y(i))$$

η=learning rate and ∇J=Gradient of the loss function w.r.t. θ.

B. Backpropagation with momentum

Similar to backpropagation but has a momentum parameter included. This technique accelerates Gradient Descent (where high variance oscillations make it hard to reach convergence) by navigating along the relevant direction and softens the oscillations in irrelevant directions.

C. Levenberg Marquardt Algorithm

The Levenberg Marquardt algorithm locates the minimum of a loss function by an iterative technique where the function is expressed as the sum of squares of non-linear real valued functions [4]. It can be thought as a combination of the Gauss-Newton method and steepest descent.

In this project, we are using Backpropagation and its variants to perform nonlinear classification on doublemoon data. The doublemoons with different distance between them is varied to analyze the effect of classification using different algorithms.

II. RESULTS and ANALYSIS

This section is divided into two cases where in first case we discuss about the results for different classification data with three different variants of Backpropagation. In second case, we discuss the effect of increasing the number of neurons in a neural network.

Case 1:

A. Backpropagation (BP)

As the given problem is linearly separable when d = 2, we can use a single neuron in the hidden layer for the multilayer perceptron to converge. For backpropagation algorithm, when number of epochs is kept constant (=500) and learning rate taken as 0.01, as it is very small it is required to have more number of epochs for it to converge. When the learning rate is in the range of 0.2 to 0.3, it converges and we can see this from performance curves in Fig 6. But as it increases further, the accuracy on the test data decreases. For d=2, the problem is linearly separable and we get an accuracy of 100% and is shown in Fig 1. The decision boundary in this case is a straight line as shown in Fig 2. as the number of units in hidden layer is just 1.

Fig 1. Confusion matrix for d = 2 and d = 4 when learning rate is 0.1 and 0.3 respectively, number of epochs is 500 and number of neurons in hidden layer is 1 and 8 respectively.

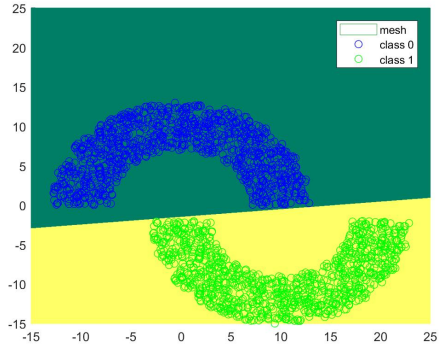

Fig 2. decision boundary for d = 2 when learning rate is 0.1, number of epochs is 1000 and number of neurons in hidden layer is 1.

For d=-4, as the given problem is not linearly separable anymore, it will not converge until we increase the number of neurons in hidden layer. When we consider number of neurons in the hidden layer are 8, then for the learning rate values in the range of 0.7 to 0.1, the backpropagation algorithm converges, which can be seen in Fig 6. We get an accuracy of 100% as shown in Fig 2. The decision boundary in this case is shown in Fig 4. The algorithm diverges for the values greater than 0.7 and for values less than 0.1. However, if we consider 6 neurons with the same learning rate of 0.7, we don't get 100% accuracy.

So, the capability of the network is limited by the number of neurons and if the learning rate increases or decreases more than a certain range, it may never converge [3].

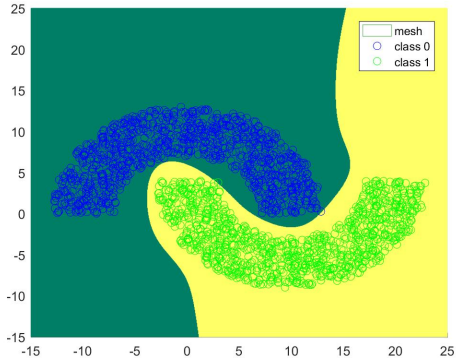

Fig 3. decision boundary for d = -4, when learning rate is 0.7, number of epochs is 1000 and number of neurons in hidden layer is 8.

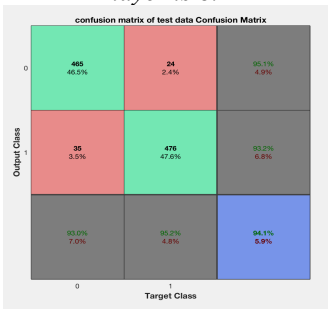

Fig 4. Confusion matrix for d=- 8 when learning rate is 0.4, number of epochs is 500 and number of neurons in hidden layer is 8.

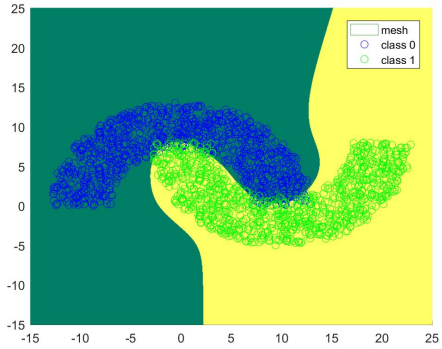

Fig 5. decision boundary for d = -8 when learning rate is 0.7, number of epochs is 1000 and number of neurons in hidden layer is 8.

For d =-8, as we increase the number of neurons the accuracy increases. It is a non-linear classification, with d=-8, the double moon data gets overlapped and it is hard to classify every data point correctly to attain 100% accuracy. We could achieve a maximum accuracy of 94% as shown in Fig 4. And the decision boundary in this case is shown in Fig 5. The performance curve is in Fig 6.

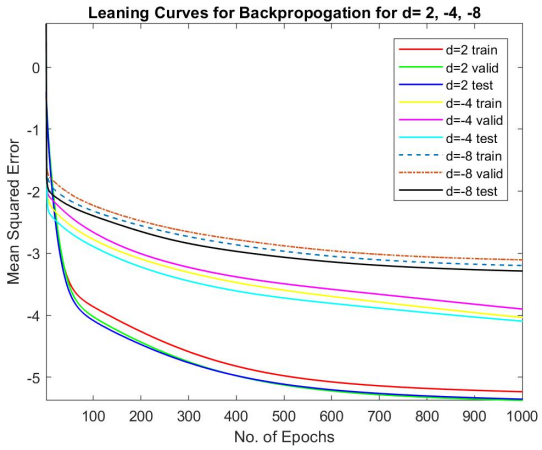

Fig 6. Performance curves of BP for d=2, -4, -8

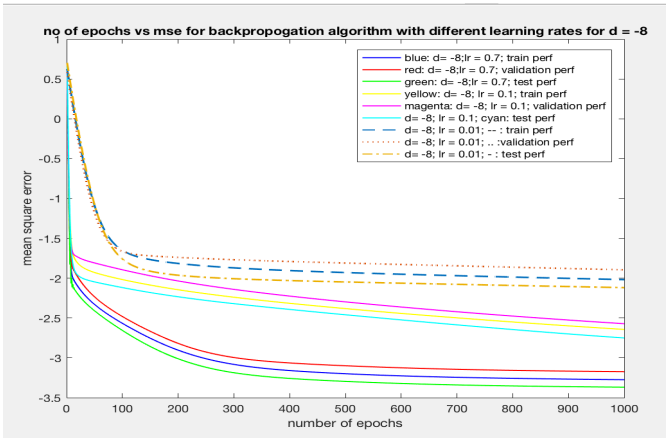

Fig 7. Effect of different learning rates on Performance of network

From Fig 6., the learning curves for d=2 reaches the most minimum MSE (Mean Square Error) when compared to d=-4 and d=-8 cases. This is because the classification gets complicated as the distance value decreases. The effect of learning rate on classification is shown in Fig 7. The learning rate for the algorithms hast to be in a certain range and if it exceeds the range, the performance of network becomes poor.

B. Backpropagation with Momentum

When backpropagation with momentum algorithm is applied for d=2, when (momentum constant) mc = 0.7, with learning rate values in the range of 0.1 to 0.7, it converges. The range of momentum constant (mc) is 0.3 to 0.7 where the algorithm converges and is shown in Fig 8. We can see that the accuracy of the network in both cases is around 100%. If mc is increased more than 0.7, the network performance becomes poor.

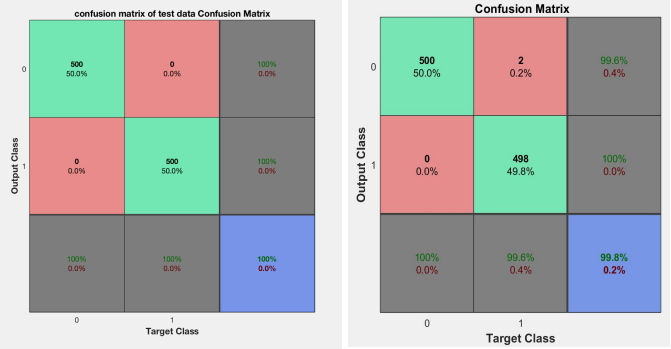

Fig 8. confusion matrix for d = 2 and d=-4 when learning rate is 0.1 and 0.3 respectively and number of epochs is 500 and number of neurons in hidden layer is 8, mc = 0.7.

With effective initial weights and certain set of learning rate and momentum constant, we can achieve the same decision boundary that we got for Backpropagation. For d=2, we get the decision boundary as shown in Fig 2. with 1 neuron in hidden layer. Similarly, for d = -4, with 8 neurons in hidden layer, we achieve non-linear classification as shown in Fig 3.

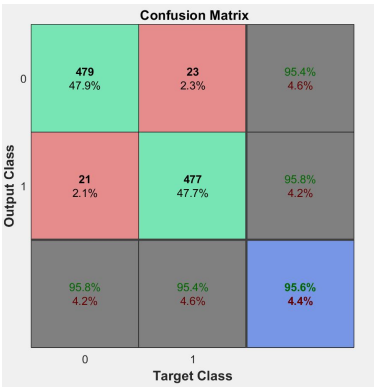

Fig 9. confusion matrix for d = -8 when learning rate is 0.5, number of epochs is 500 and number of neurons in hidden layer is 8, mc = 0.7

For d = -8, it is a nonlinear classification and the double moon data gets overlapped, it is hard to classify every data point correctly to attain 100% accuracy. We could only achieve accuracy of 96% as shown in Fig 9.

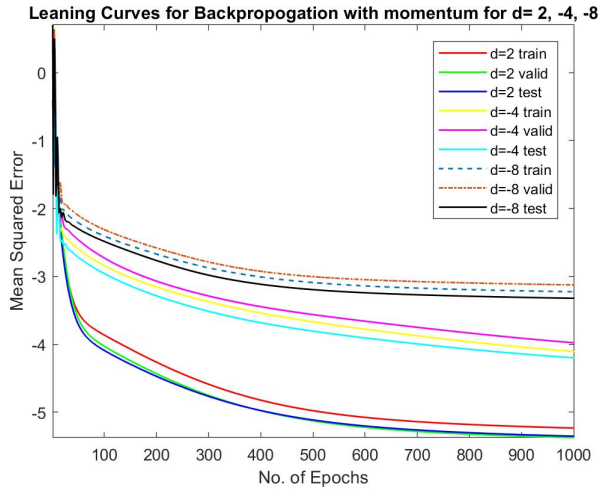

Fig 10. Learning curves for BP with momentum for d=2,-4,-8.

The learning curves for d=2, -4, -8 for BP with momentum is shown in Fig 10. So, for different values of d, if we keep learning rate constant and increase 'mc' value or by keeping 'mc' constant and increasing learning rate, the accuracy decreases. The effect of learning rate by keeping mc constant and the effect of mc by keeping learning rate constant is discussed.

From Table1., keeping mc constant and choosing learning rates beyond the range of 0.5 to 1.5 had poor performance for all the three cases.

Backpropagation with momentum				
learning rate	momentum constant	d = 2	d = -4	d = -8
0.5	0.2	100	99.7	95.4
1.5	0.2	100	99.6	94.7
2	0.2	32.9	50	50
0.001	0.2	63.7	50	49.7
0.01	0.2	99.9	85.6	78.6

Table 1. Effect of learning rate on accuracy for BP with Momentum with 'mc'=0.2.

Keeping learning rate constant around 0.5 and changing mc resulted in Fig 11. Choosing mc beyond 0.8 resulted in poor network performance for all the three cases.

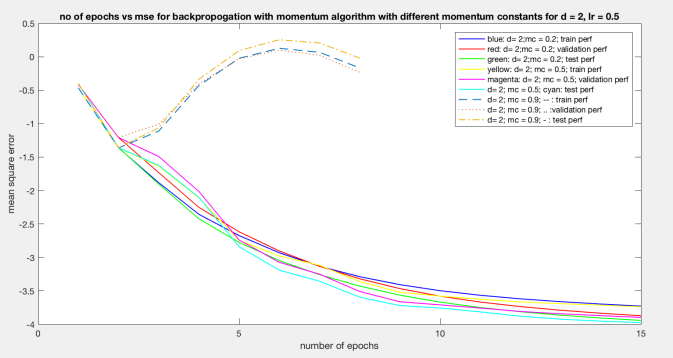

Fig 11. The effect of different mc on performance of the network

C. Levenberg Marquardt Algorithm

Being the most optimized algorithm, even by changing the value of 'mu' or 'mc', the accuracy of Levenberg Algorithm always remained high and converged for every instance. For d=2 and d=-4, the algorithm converged and the accuracy of the network is 100%. For d=-8, the accuracy of the network almost remained same as that of BP with momentum and is around 96% as shown in Fig 13. The decision boundary of LM for d=2 and d=-4 is shown in Fig 12. This is almost same as that of its previous algorithms. The decision boundary of d=-8 is a bit changed as we chose same initial weights for all the three cases and choosing mu=0.1 and is shown in Fig 14.

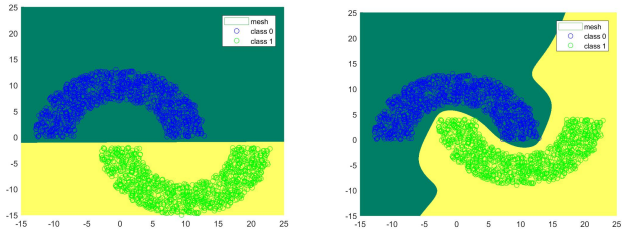

Fig 12. Decision boundary for d=2 and d=-4 using LM

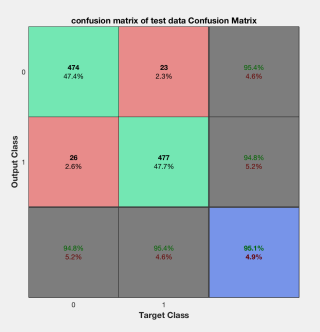

Fig 13. confusion matrix and performance curve for d = -8 when learning rate is 0.7, number of epochs is 500 and number of neurons in hidden layer is 8, mu = 0.1.

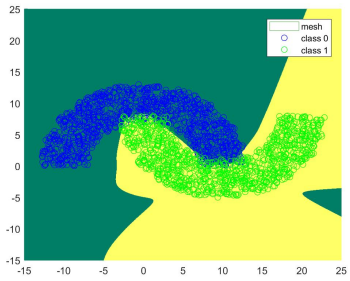

Fig 14. Decision Boundary for d=-8 using LM with mu=0.1

The learning curves for d=2, -4, -8 using LM algorithm are shown in Fig 15.

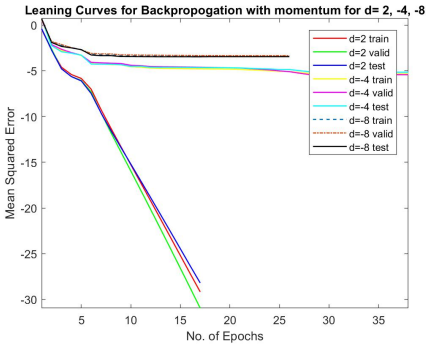

Fig 15. Learning curves for d=2 ,-4, -8 using LM

The effect of mu value on the LM algorithm is shown in Fig 16. Even if the mu value is out of range, being the most optimized algorithm, the network converges but only after taking more number of iterations.

In the second case where we increase the number of neurons in the hidden layer and we observe that the data gets overfitted and the performance of the network completely fails.

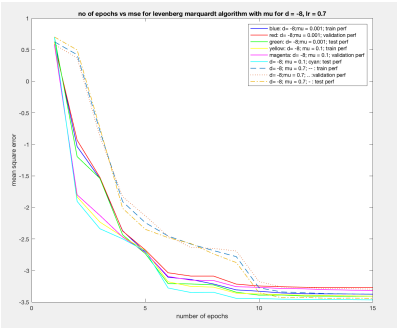

Fig 16. Effect of mu on performance of the network

Case 2: Changing the number of hidden layer neurons for d=-8

For d = -8, as we increase the number of neurons the accuracy increases. But if the number of neurons are very high, then overfitting occurs and testing accuracy is decreased to a large extent. For backpropagation algorithm and BP with momentum, we can see that for number of neurons = 8, the accuracy obtained is 95.6%. The learning curves of BP are shown in Fig 17. As number of neurons increases to 30 the accuracy is decreased to 67% and the decision boundary in this case is all misclassified as shown in Fig 18.

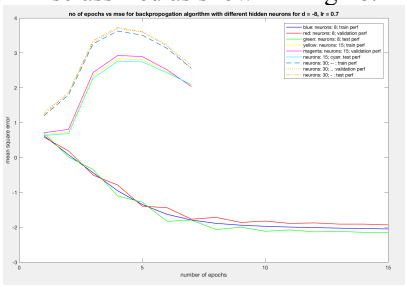

Fig 17. Effect of neurons on the performance of network in BP

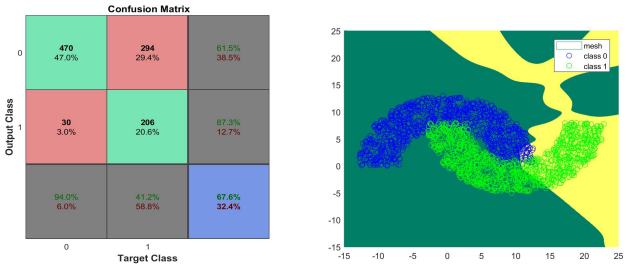

Fig 18. Confusion Matrix and Decision Boundary for BP with 30 neurons in hidden layer.

For LM algortithm, for number of neurons to be 8,15 or 30 the accuracy obtained is better and the performance of the network is great as shown in Fig 19 for 30 hidden neurons.

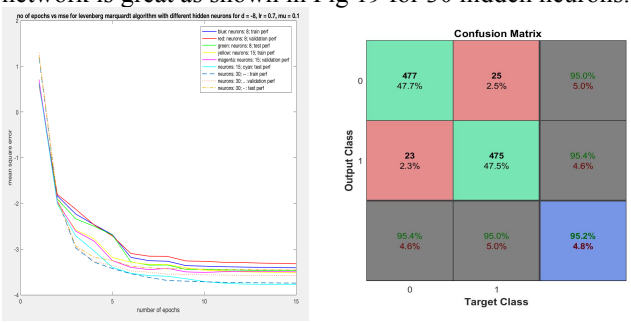

Fig 19. Learning Curves and Conf Matrix for LM

IV REFERENCES

[1] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

[2] J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press, 1989.

[3] A. K. Rigler, J. M. Irvine and T. P. Vogl, "Rescaling of variables in back propagation learning," *Neural Networks*, vol. 3, no. 5, pp 561–573, 1990.

[4] D. F. Shanno, "Recent advances in numerical techniques for large-scale optimization," in Neural Networks for Control, Miller, Sutton and Werbos, eds., Cambridge, MA: MIT Press, 1990. (Chapter 12)

Matlab Function Code:

%% Note: This is a function file which is used to generate the data. So please save this with filename 'dbmoon.m' in the same folder along with the source code

```
function data=dbmoon(N,d,r,w)
% Usage: data=dbmoon(N,d,r,w)
% doublemoon.m - genereate the double moon data set in Haykin's book titled
% "neural networks and learning machine" third edition 2009 Pearson
% Figure 1.8 pp. 61
% The data set contains two regions A and B representing 2 classes
% each region is a half ring with radius r = 10, width = 6, one is upper
% half and the other is lower half
% d: distance between the two regions
% will generate region A centered at (0, 0) and region B is a mirror image
% of region A (w.r.t. x axis) with a (r, d) shift of origin
% N: # of samples each class, default = 1000
% d: seperation of two class, negative value means overlapping (default=1)
% r: radius (default=10), w: width of ring (default=6)
%

% clear all; close all;
if nargin<4, w=6; end
if nargin<3, r=10; end
if nargin<2, d=1; end
if nargin < 1, N=1000; end

% generate region A:
% first generate a uniformly random distributed data points from (-r-w/2, 0)
% to (r+w/2, r+w/2)
N1=10*N;  % generate more points and select those meet criteria
w2=w/2;
done=0; data=[]; tmp1=[];
while ~done
    if (N==1000)
        rng(420)
    elseif (N==500)
        rng(200)
    end
    tmp=[2*(r+w2)*(rand(N1,1)-0.5) (r+w2)*rand(N1,1)];
    % 3rd column of tmp is the magnitude of each data point
    tmp(:,3)=sqrt(tmp(:,1).*tmp(:,1)+tmp(:,2).*tmp(:,2));
    idx=find([tmp(:,3)>r-w2] & [tmp(:,3)<r+w2]);
    tmp1=[tmp1;tmp(idx,1:2)];
    if length(idx)>= N
        done=1;
    end
    % if not enough data point, generate more and test
end
% region A data and class label 0
% region B data is region A data flip y coordinate - d, and x coordinate +r
data=[tmp1(1:N,:) zeros(N,1);
[tmp1(1:N,1)+r -tmp1(1:N,2)-d ones(N,1)]];
```

```
figure;
plot(data(1:N,1),data(1:N,2),'.r',data(N+1:end,1),data(N+1:end,2),'.b');
legend('Class 0','Class 1');
title(['Double moon data set, d = ' num2str(d)])
axis([-r-w2 2*r+w2 -r-w2-d r+w2])

save dbmoon N r w d data;
```

Matlab Source Code:

```
%% Training Data
clc;
clear;
close all;
N=500;
d=[2 -4 -8];
disp('Choose the training algorithm by typing the respective numnber: ');
choice= input('1.BP; 2. BP with Momentum; 3. Levenberg-Marquardt: ');
for i=1:length(d)
train_data1=dbmoon(1000,d(i),10,6); %The function 'dbmoon' produces the data in
double moon based on inputs
train_data(:,1)=train_data1(:,1);
train_data(:,2)=train_data1(:,2);
%% Testing Data
test_data1=dbmoon(500,d(i),10,6); %The function 'dbmoon' produces the data in
double moon based on inputs
test_data(:,1)=test_data1(:,1);
test_data(:,2)=test_data1(:,2);
target_test_data=test_data1(:,3);
%% Training Multilayer Neural Network
target_data=train_data1(:,3);
if (choice ==1)
    net=feedforwardnet(30,'traingd');
elseif (choice ==2)
    net=feedforwardnet(30,'traingdm');
elseif (choice ==3)
    net= feedforwardnet(30,'trainlm');
end
if(choice ==2)||(choice==3)
    net.trainParam.mc=0.7;
end
if (choice==3)
    net.trainParam.mu=0.1;
end
if (i==1)
    net.trainParam.lr=0.1;
    rng(10)
elseif (i==2)
    net.trainParam.lr=0.3;
    rng(21)
elseif (i==3)
    net.trainParam.lr=0.4;
    rng(21)
```

```
end
net=configure(net,train_data',target_data');
[net,tr(i)]=train(net,train_data',target_data');
y=net(test_data');

figure;
plotconfusion(target_test_data',y);


%% Decision Boundary
range= -15:0.05:25;
[p1,p2]=meshgrid(range,range);
pp=[p1(:) p2(:)]';
outp = net(pp);
outp(outp<0.5)=0;
outp(outp>=0.5)=1;
figure;
mesh(p1,p2,reshape(outp,length(range),length(range))-5);
x=train_data(:,1);
y=train_data(:,2);
colormap summer;
view(2);
hold on
plot(x(1:1000),y(1:1000),'ob',x(1001:2000),y(1001:2000),'og');
legend('mesh','class 0','class 1');
hold off
end
figure;
plot(log(tr(1).perf),'r','LineWidth',1)
%xlim([0 1000]);
axis tight;
hold on;
plot(log(tr(1).vperf),'g', 'LineWidth',1)
hold on;
plot(log(tr(1).tperf),'b', 'LineWidth',1)
hold on;
plot(log(tr(2).perf),'y','LineWidth',1)
hold on;
plot(log(tr(2).vperf),'m', 'LineWidth',1)
hold on;
plot(log(tr(2).tperf),'c', 'LineWidth',1)
hold on;
plot(log(tr(3).perf),'--','LineWidth',1)
hold on;
plot(log(tr(3).vperf),'-.', 'LineWidth',1)
hold on;
plot(log(tr(3).tperf),'k', 'LineWidth',1)
xlabel('No. of Epochs');
ylabel('Mean Squared Error');
title('Leaning Curves for Backpropogation with momentum for d= 2, -4, -8');
legend('d=2 train','d=2 valid','d=2 test','d=-4 train','d=-4 valid','d=-4
test','d=-8 train','d=-8 valid','d=-8 test');
hold off;
```