Multi-layer perceptron network trained by Backpropagation algorithm and its variants

Rakshith Subramanyam[#1], Sai Prajwal Kotamraju[#1], Sai Pratyusha Gutti[#1]

EEE, Arizona State University

[1]rsubra17@asu.edu, [2]skotamra@asu.edu, [3]sgutti@asu.edu

***Abstract*— In this project, the multilayer perceptron network has been studied and it has been trained by variants of the backpropagation algorithm in order to understand the effect of different hyperparameters including learning rate, momentum, and the number of active neurons. The multilayer network is trained on a double moon cluster of 2000 data points with a given radius and width. The complexity of the network increases with the vertical distance between the two moons clusters. For testing purpose, a test data of 1000 data points has been created with the same parameters to generate the test results. The three required methods have been simulated. The optimum learning rate was found to be 0.1 and 0.8 was the optimum momentum constant. 5 active neurons gave an optimum fitting for the data.**

***Keywords*— multi-layer perceptron, backpropagation, backpropagation with momentum, Levenberg-Marquardt, hyperparameters**

I. Introduction

From the assignment on perceptron it was observed that perceptron algorithm could perform perfect classification only for those classes which are linearly separable. In this project, the multilayer perceptron is implemented to overcome the drawback of perceptron. The basic structure of the single layer perceptron is maintained in a multilayer perceptron except that there are one or more hidden layers with active neurons in each layer [1].

The organisation of this report is as follows. Section II describes the backpropagation, backpropagation with momentum and Levenberg - Marquardt to solve the pattern classification. Section III covers the simulation procedure and the results obtained for each method. Section IV concludes the project report.

II. Approach

The backpropagation, backpropagation with momentum and the Levenberg - Marquardt are briefly explained below.

A. Backpropagation

Backpropagation (BP) is a family of neural networks which consists of multiple hidden layers connected to each other. It is a fast approach based on the steepest descent algorithm where with the required number of hidden neurons it can classify complex networks with nonlinear decision boundaries. It is characterized by two phases - the forward and backward phases [2]. In the forward phase, the activation is computed from the input to the output layer whereas in the backward phase, the error between the output and the desired classification is propagated backwards to update the weight and biases.

The weights are updated according to the below formula:

$w_{k+1} = w_k - \alpha g_k$ where α is the learning rate. The negative sign in the equation indicates that the weights w are updated in the direction opposite to that of the gradient g.

B. Backpropagation with Momentum

For the updation of weights (w), backpropagation with momentum (BPM) exploits the changes in the error surface along with the trends in the gradient. The error surface with respect to the parameters might be a non-convex surface where the network would get stuck in a local minimum or a plateau. To avoid this, momentum can be included in the back propagation algorithm which is calculated as the sum of the fraction of previous weight update and the present update due to the gradient descent [3].

The momentum constant, μ, is a number between 0 and 1 where 0 implies an update solely based on gradient g and 1 for ignoring the effect of gradient. The weight update is according to the following equation,

$$w_{k-1} = w_k - \alpha_{k+1} g_k + \mu w_{k-1}$$

where α is the learning rate.

C. Levenberg - Marquardt

The LM training algorithm is used to approach second order training speed without having to compute the Hessian matrix as in the Newton's method which is computationally costly. When the parameter μ is zero the algorithm is Newton's method when μ is large it operates as gradient descent with small step size. The μ is decreased with each step with the decrement rate given.

III. Simulation and Results

Using Matlab as the platform for analysis, the multilayer perceptron has been studied and analyzed by varying different parameters in the BP, BPM and LM methods.

A. Effect of Learning rate, Momentum and number of neurons on Back Propagation algorithm with Gradient Descent

[#1] All the authors did equal amount of work

The below results have been analyzed for both zero momentum which corresponds to the back propagation and also non-zero momentum which corresponds to BPM.
Increase in learning rate decreases the time taken for the algorithm to converge. But if the value of learning rate is too high, the algorithm might jump across the valley, failing to get a good solution. So it is always better to have an optimal learning rate which is neither too small nor too high. To analyze the effect of learning rate in detail, the learning curves for all the d values are examined with zero momentum and 3 hidden layer neurons. The network was trained for 20,000 epochs and the results are tabulated as shown below.

TABLE I
EFFECT OF LEARNING RATE ON TEST ACCURACY - CONFUSION MATRIX

Learning Rate Value	Test Set Accuracy (%)	d
0.01	82.5	-8
0.1	96.1	-8
0.5	95.6	-8
0.99	95.1	-8
0.01	99.7	-4
0.1	99.7	-4
0.9	99.9	-4
0.01	100	2
0.1	100	2
0.9	100	2

From Table I, it can be noted that for a small learning rate, the test set accuracy is relatively low. With the increase in learning rate, the test set accuracy increases until a certain optimal learning rate. Increase of learning rate beyond this point decreases the test set accuracy. From this, it can be concluded that the learning rate can neither be too high nor too low.

From our observation, the algorithm reached the convergence point on 12,530th epoch with a learning rate of 0.5 while the algorithm never reached the convergence point (with 20,000 as the maximum epoch limit) for a learning rate of 0.01. Thus,

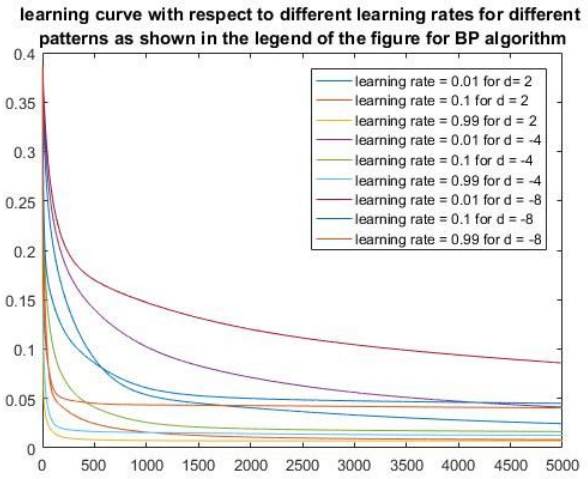


Fig. 1. Learning curves for different cases of learning rate for every case of d using Backpropagation algorithm

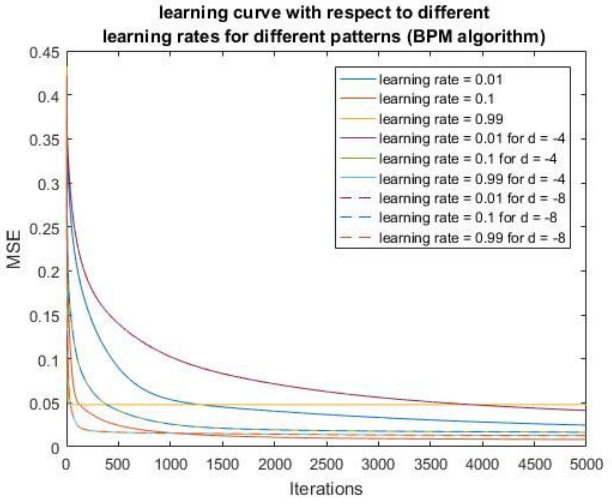


Fig. 2. Learning curves for different cases of learning rate for every case of d using Backpropagation with momentum algorithm

higher learning rates are preferable if there is a time constraint on training the network. Also, learning rate addresses the trade-off between the time taken for the algorithm to converge and the test set accuracy. Fig. 1 and Fig. 2 explains the trends for various cases of d. It can be observed that the MSE falls faster with momentum.

Another important hyperparameter other than learning rate is the number of neurons in the hidden layer. Altering the number of neurons alters the decision boundary generated by the network. To study the effect of this hyper parameter, a multi-layer neural network with one hidden layer of variable number of neurons with a learning rate of 0.05 and zero momentum is considered for case d = -8. The network is left to train for 20,000 epochs and the results are tabulated as shown in Table II.

TABLE II
EFFECT OF NUMBER OF HIDDEN LAYER NEURONS ON TEST ACCURACY

Number of Hidden Layer Neurons	Test Set Accuracy (%)
2	79.5
5	96
25	95.8
100	95.6
175	50.8
250	49.7

From Table II, we see that with the increase in the number of neurons in the hidden layer, the accuracy starts increasing initially because the model switches gradually from underfitting the data to correctly generalizing the data. After a point, with the increase of neurons, the model starts to learn the noise in the data and starts overfitting the data thereby decreasing the test set accuracy.

Another important hyperparameter that influences the training algorithm is momentum. Momentum tends to help the learning when the algorithm gets stuck on a plateau. Theoretically, with the increase in the momentum value, the algorithm gets past the plateau in a fewer iterations. But, with a very high momentum, the value of MSE increases with respect to iterations thereby decreases the test set accuracy.

TABLE III
EFFECT OF HIGHER MOMENTUM VALUES ON TESTSET ACCURACY

Momentum Value	Test Set Accuracy (%)
Less than or equal to 0.9	95.4
0.97	32.3
0.99	47.4
1	47.5

Having discussed the effect of all the three key hyperparameters, the results for d= +2, d=-4 and d=-8 cases using a learning rate of 0.05 and momentum of 0.8 and number of hidden layers to be 3 have been presented. Note that the plots in Fig. 3, Fig. 4, and Fig. 5 also correspond to the decision boundaries for BP algorithm without momentum. i.e. BP algorithm results in the same decision boundary as of BPM.

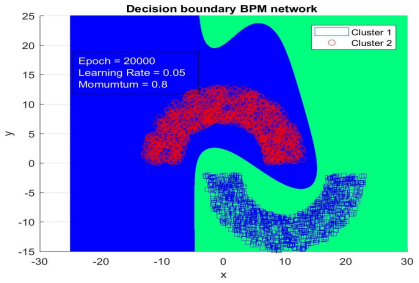


Fig. 3. Decision boundary for d=+2 case with lr= 0.05 and 5 hidden neurons

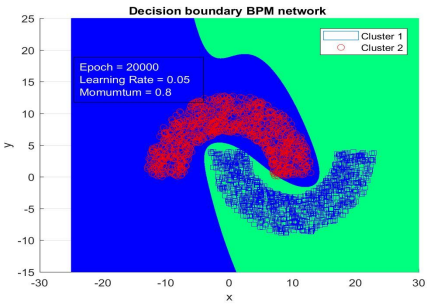


Fig. 4. Decision boundary for d=-4 case with lr= 0.05 and 5 hidden neurons

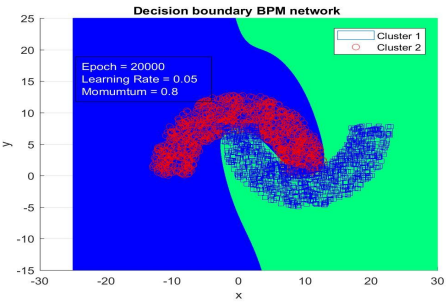


Fig. 5. Decision boundary for d=-8 case with lr= 0.05 and 5 hidden neurons

TABLE IV
TEST SET ACCURACIES FOR 'D' VALUES USING BACKPROPAGATION WITH MOMENTUM - CONFUSION MATRIX

d value	Test Set Accuracy (%)
+2	100
-4	99.8
-8	96

B. Effects of Neurons in Levenberg-Marquardt (LM) training algorithm

The learning curves for different cases of d is as shown in Fig. 6. It can be noted that as the complexity of the training data increases according to d, the convergence time increases. The final MSE convergence value decreases as the distance 'd' between the clusters increases.

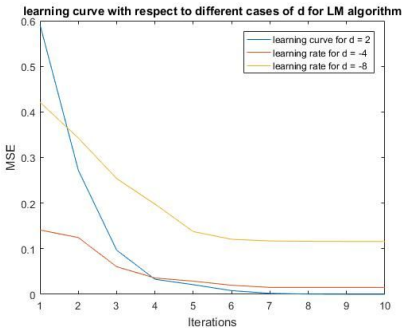


Fig. 6. Learning curves for different cases of d

It was observed in Fig. 7, 8, 9 that the increase in the number of neurons has decreased the bias but the variance was increased. It is concluded that as the number of neurons are increased the model is overfitting to the training data. This can also be seen in the learning curves in Fig. 10. The MSE is higher initially as the number of neurons increase but different cases of d have the same trends of learning curves

The μ parameter when decreased or increased with a bigger steps, algorithm converges more efficiently with a higher classification accuracy. The LM algorithm converges efficiently compared to Backpropagation with momentum

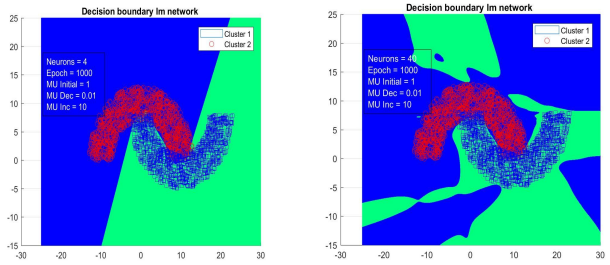


Fig. 7. Nonlinear decision boundary d = -8; neurons = 4 and 40 respectively.

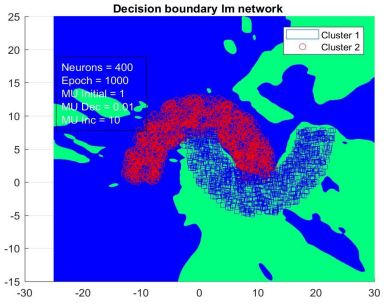
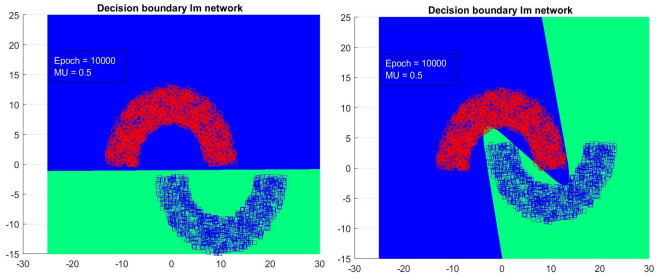


Fig. 8. Nonlinear decision boundary for d = -8; neurons 400.



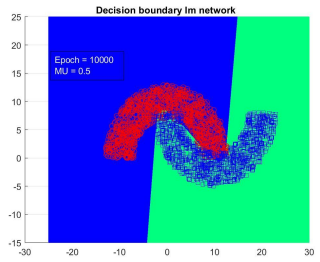


Fig. 9. Decision boundaries using LM. (Top Left) d = 2 case, (Top Right) d = -4 case and (Bottom) d = -8 case

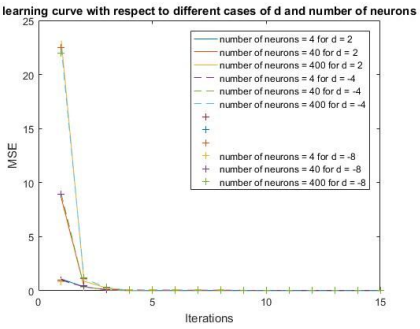


Fig. 10. Learning curve for d =2,-4, -8; neurons = 4, 40, 400.

IV. Conclusion

Thus, the multilayer perceptron has been studied and the data has been trained on a double moon cluster. From the test results we can infer that the optimum learning rate of 0.1 is suitable for the fast convergence time and good accuracy. It was also observed that 5 hidden neurons gave optimum fit and overfitting occurs as we increase the hidden neurons. The corresponding confusions matrix have been tabulated and the graphs have been generated.

TABLE VI
Effect Of Mu Increment and Decrement on Test Accuracy

MU Initial	MU Decrement	MU Increment	Test Set Accuracy
1	0.1	10	99.6%
1	0.01	10	99.5%
1	0.001	100	99.9%

References

[1] Michael Nielsen. How the backpropagation algorithm works [Online]. Available: http://neuralnetworksanddeeplearning.com/chap2.html

[2] Saed Sayad. Artificial Neural Network Perceptron [Online]. Available: http://www.saedsayad.com/artificial_neural_network_bkp.htm

[3] Vu N.P. Dao, Rao Vemuri. A Performance Comparison of Different Back Propagation Neural Networks Methods in Computer Network Intrusion Detection [Online]. Available: http://web.cs.ucdavis.edu/~vemuri/papers/bp-intrusion%20detection.pdf

```
% EEE 511: Project 1
% Multilayer Feedforward network.
%  Authors:  Rakshith  Subramanyam,  Sai  Pratyusha
Gutti and Sai Prajwal Kotamraju
% Date: 9/17/2017
%% Basic Commands
clc;
clear;
%% Data preparation

display_data = 0;
d = [-2 4 8];
T_train = [zeros(1,1000) ones(1,1000)];
T_test = [zeros(1,500) ones(1,500)];

for i = 1:length(d)

    C1 = [0 0];   % center of the circle
    C2 = [10 d(i)];
    R1 = [7 13];  % range of radii
    R2 = [7 13];
      A1  =  [0  2]*pi/2;  %  [rad]  range  of  allowed
angles
    A2 = [2 4]*pi/2;

    nPoints_train = 1000;
    nPoints_test = 500;

    rng(42);
                                  urand_train        =
@(nPoints_train,limits)(limits(1)                    +
rand(nPoints_train,1)*diff(limits));
                            randomCircle_train       =
@(n,r,a)(pol2cart(urand_train(n,a),urand_train(n,r
)));

    rng(43);
     urand_test  =  @(nPoints_test,limits)(limits(1)
+ rand(nPoints_test,1)*diff(limits));
                             randomCircle_test       =
@(n,r,a)(pol2cart(urand_test(n,a),urand_test(n,r))
);

                           [P1x(:,i),P1y(:,i)]       =
randomCircle_train(nPoints_train,R1,A1);
    P1x(:,i) = P1x(:,i) + C1(1);
    P1y(:,i) = P1y(:,i) + C1(2);
                           [P2x(:,i),P2y(:,i)]       =
randomCircle_train(nPoints_train,R2,A2);
    P2x(:,i) = P2x(:,i) + C2(1);
    P2y(:,i) = P2y(:,i) + C2(2);

                         [P1xx(:,i),P1yy(:,i)]       =
randomCircle_test(nPoints_test,R1,A1);
    P1xx(:,i) = P1xx(:,i) + C1(1);
    P1yy(:,i) = P1yy(:,i) + C1(2);
                         [P2xx(:,i),P2yy(:,i)]       =
randomCircle_test(nPoints_test,R2,A2);
    P2xx(:,i) = P2xx(:,i) + C2(1);
    P2yy(:,i) = P2yy(:,i) + C2(2);

     X_train(:,:,i) = [P1x(:,i) P1y(:,i);P2x(:,i)
P2y(:,i)]';
    X_test(:,:,i) = [P1xx(:,i) P1yy(:,i);P2xx(:,i)
P2yy(:,i)]';


    if display_data == 1
        figure
        plot(P1x(:,i),P1y(:,i),'or'); hold on;
        ylim([-20,20])
        plot(P2x(:,i) ,P2y(:,i),'sb'); hold on;
        axis square;
        figure;
        plotpv(X_train(:,:,i),T_train);

    end
end

%% Back propagation with momentum network

display_bpm = 0;
for i = 3:length(d)
    neurons_bpm = 5; % No of neurons
                                          bpm_net        =
feedforwardnet(neurons_bpm,'traingdm');
     rng(48); % For fixed initialization of weights
every time
    %bpm_net.trainParam.max_fail = 150;
    bpm_net = init(bpm_net);

    bpm_net.trainParam.epochs = 20000;
    bpm_net.trainParam.lr = 0.05; % Range 0 - 1
    bpm_net.trainParam.mc = 0.8; % Range 0 - 1
                         [bpm_model,bpm_error]      =
train(bpm_net,X_train(:,:,i),T_train);
    y_train_bpm = bpm_model(X_train(:,:,i));

    y_test_bpm = bpm_model(X_test(:,:,i));

    if display_bpm == 1

        figure
        plotconfusion(T_train,y_train_bpm);
          title(['Confusion matrix BPM train data
for d = ',num2str(d(i))])

        span = -25:0.05:30;
        [p1,p2]=meshgrid(span,span);
        grid_points = [p1(:) p2(:)]';
                               y_grid_points_bpm    =
bpm_model(grid_points);

y_grid_points_bpm(y_grid_points_bpm<=0.5)=0;
y_grid_points_bpm(y_grid_points_bpm>0.5)=1;
        figure

mesh(p1,p2,reshape(y_grid_points_bpm,length(span),
length(span))-5);
        colormap winter
        view(2)
        hold on
```

```
        plot(P1x(:,i),P1y(:,i),'or'); hold on;
        ylim([-15,25])
        plot(P2x(:,i) ,P2y(:,i),'sb'); hold on;
        title('Decision boundary BPM network ');
         xlabel('x'); ylabel('y'); legend('Cluster
1', 'Cluster 2');
        dim = [.2 .5 .3 .3];
                        annotate   =   ['Epoch   =
',num2str(bpm_net.trainParam.epochs),newline
,'Learning                  Rate                  =
',num2str(bpm_net.trainParam.lr),newline
,'Momumtum = ',num2str(bpm_net.trainParam.mc)];
                                            an     =
annotation('textbox',dim,'String',annotate,'FitBox
ToText','on');
        an.Color = 'white';

        figure
        plotperf(bpm_error);
        dim = [.2 .5 .3 .3];
                        annotate   =   ['Neurons=
',num2str(neurons_bpm),    newline,    'Epoch    =
',num2str(bpm_net.trainParam.epochs),newline
,'Learning                  Rate                  =
',num2str(bpm_net.trainParam.lr),newline
,'Momumtum = ',num2str(bpm_net.trainParam.mc)];
                                            an     =
annotation('textbox',dim,'String',annotate,'FitBox
ToText','on');
        an.Color = 'black';
    end

    if display_bpm == 0

        figure
        plotconfusion(T_test,y_test_bpm);
         title(['Confusion matrix BPM test data for
d = ',num2str(d(i))])

        span = -25:0.05:30;
        [p1,p2]=meshgrid(span,span);
        grid_points = [p1(:) p2(:)];
                          y_grid_points_bpm    =
bpm_model(grid_points);

y_grid_points_bpm(y_grid_points_bpm<=0.5)=0;
y_grid_points_bpm(y_grid_points_bpm>0.5)=1;
        figure;

mesh(p1,p2,reshape(y_grid_points_bpm,length(span),
length(span))-5);
        colormap winter
        view(2)
        hold on
        plot(P1xx(:,i),P1yy(:,i),'or'); hold on;
        ylim([-15,25])
        plot(P2xx(:,i) ,P2yy(:,i),'sb'); hold on;
        title('Decision boundary BPm network ')
        dim = [.2 .5 .3 .3];
                          annotate   =   ['d   =
',num2str(d(i)),newline,'Epoch                    =
',num2str(bpm_net.trainParam.epochs),newline
,'Learning                  Rate                  =
',num2str(bpm_net.trainParam.lr),newline
,'Momumtum = ',num2str(bpm_net.trainParam.mc)];
                                            an     =
annotation('textbox',dim,'String',annotate,'FitBox
ToText','on');
        an.Color = 'white';
    end
end


%% Levenberg-Marquardt backpropagation network

display_lm = 0;
for i = 1:length(d)
    neurons_lm = 3;
                                        lm_net         =
feedforwardnet(neurons_lm,'trainlm');
    lm_net.trainParam.epochs = 10000;
    lm_net.trainParam.mu = 0.5; % Range 0 - 1
                          [lm_model,lm_error]      =
train(lm_net,X_train(:,:,i),T_train);
    y_train_lm = lm_model(X_train(:,:,i));
    y_test_lm = lm_model(X_test(:,:,i));

    if display_lm == 0

        figure
        plotconfusion(T_train,y_train_lm);
         title(['Confusion matrix LM train data for
d = ',num2str(d(i))])

        span = -25:0.05:30;
        [p1,p2]=meshgrid(span,span);
        grid_points = [p1(:) p2(:)];
        y_grid_points_lm = lm_model(grid_points);
         y_grid_points_lm(y_grid_points_lm<=0.5)=0;
y_grid_points_lm(y_grid_points_lm>0.5)=1;
        figure;

mesh(p1,p2,reshape(y_grid_points_lm,length(span),l
ength(span))-5);
        colormap winter
        view(2)
        hold on
        plot(P1x(:,i),P1y(:,i),'or'); hold on;
        ylim([-15,25])
        plot(P2x(:,i) ,P2y(:,i),'sb'); hold on;
        title('Decision boundary lm network ')
        dim = [.2 .5 .3 .3];
                          annotate   =   ['Epoch   =
',num2str(lm_net.trainParam.epochs),newline  ,'MU =
',num2str(lm_net.trainParam.mu)];
                                            an     =
annotation('textbox',dim,'String',annotate,'FitBox
ToText','on');
        an.Color = 'white';
        plotperf(lm_error);
    end

    if display_lm == 1

        figure
        plotconfusion(T_test,y_test_lm);
```

```
         title(['Confusion matrix LM test data for
d = ',num2str(d(i))])

        span = -25:0.05:30;
        [p1,p2]=meshgrid(span,span);
        grid_points = [p1(:) p2(:)]';
        y_grid_points_lm = lm_model(grid_points);

%y_grid_points_lm(y_grid_points_lm<=0.5)=0;
y_grid_points_lm(y_grid_points_lm>0.5)=1;
        figure;

mesh(p1,p2,reshape(y_grid_points_lm,length(span),l
ength(span))-5);
        colormap winter
        view(2)
        hold on
        plot(P1xx(:,i),P1yy(:,i),'or'); hold on;
        ylim([-15,25])
        plot(P2xx(:,i) ,P2yy(:,i),'sb'); hold on;
        title('Decision boundary LM network ')
        dim = [.2 .5 .3 .3];
                            annotate    =    ['d    =
',num2str(d(i)),newline,'Epoch                    =
',num2str(lm_net.trainParam.epochs),newline  ,'MU =
',num2str(lm_net.trainParam.mu)];
                                          an     =
annotation('textbox',dim,'String',annotate,'FitBox
ToText','on');
        an.Color = 'white';
    end

end
```