The University of Hong Kong

ELEC6604 Neural Networks, Fuzzy Systems and Genetic Algorithms

Assignment 1

Lecturer: Dr. Pang

Student Name: Zhao Yunqing

UID: 3035541156

04/02/2019 Hong Kong

Contents

Part 1 Perceptron

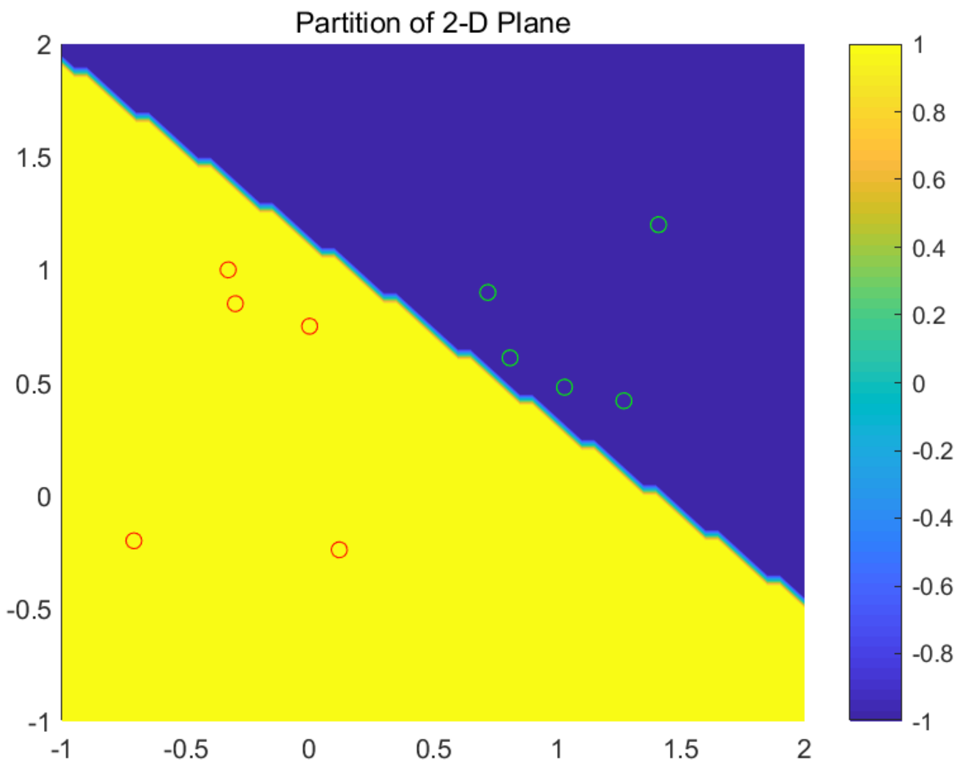
Introduction

The perceptron is an algorithm for supervised learning of binary classifiers, represented by a vector of numbers, being weights of the linear classifier.

Binary Dataset

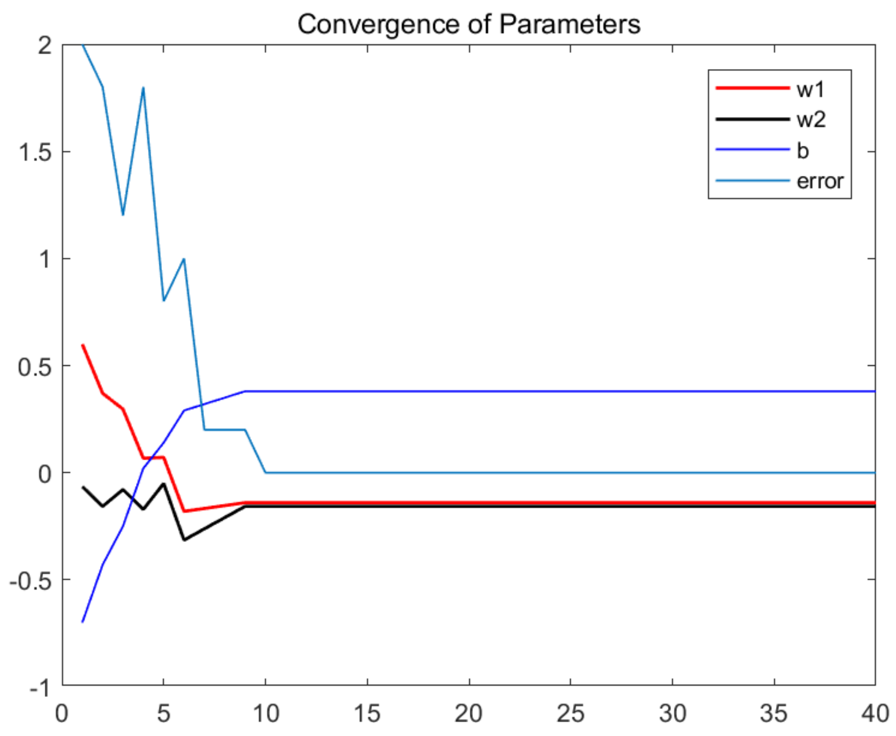
The used dataset is of binary format, which will be divided by the well-trained Perceptron.

Because the data set is linear separate, so we can see that the result is successful: In this case, the hyper plain shows the well partition of the binary dataset (Graph 1).



Graph 1

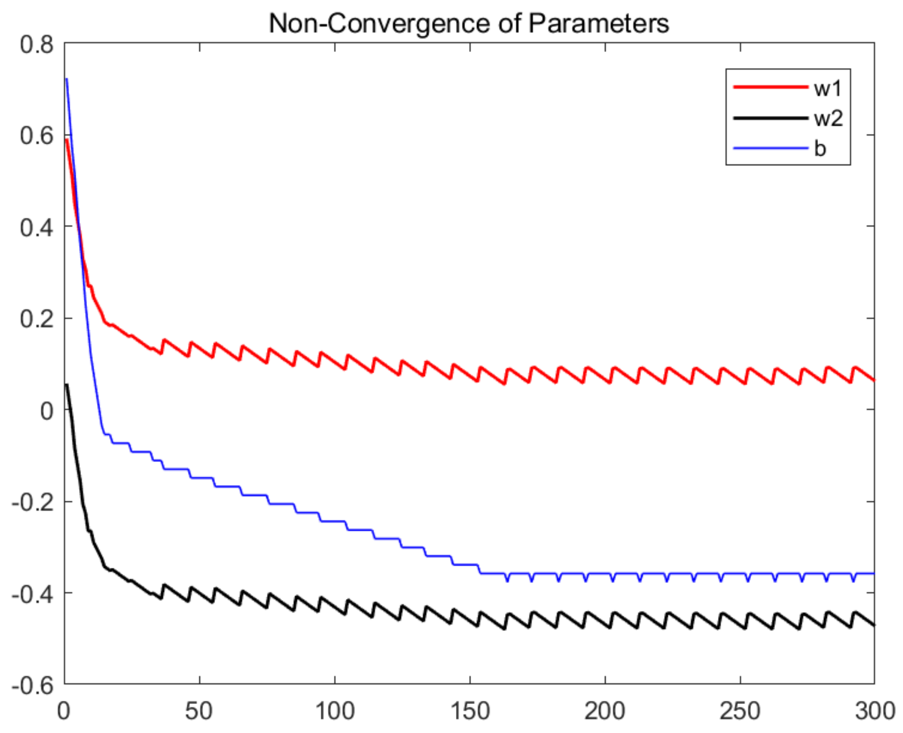
The Graph 2 shows the convergence of the network parameters. Since the dataset is linear separate, so the parameters will be adjusted to appropriate horizon, and no change happens after that all the data can be correctly divided, or loss function is optimized to the best value.



Graph 2

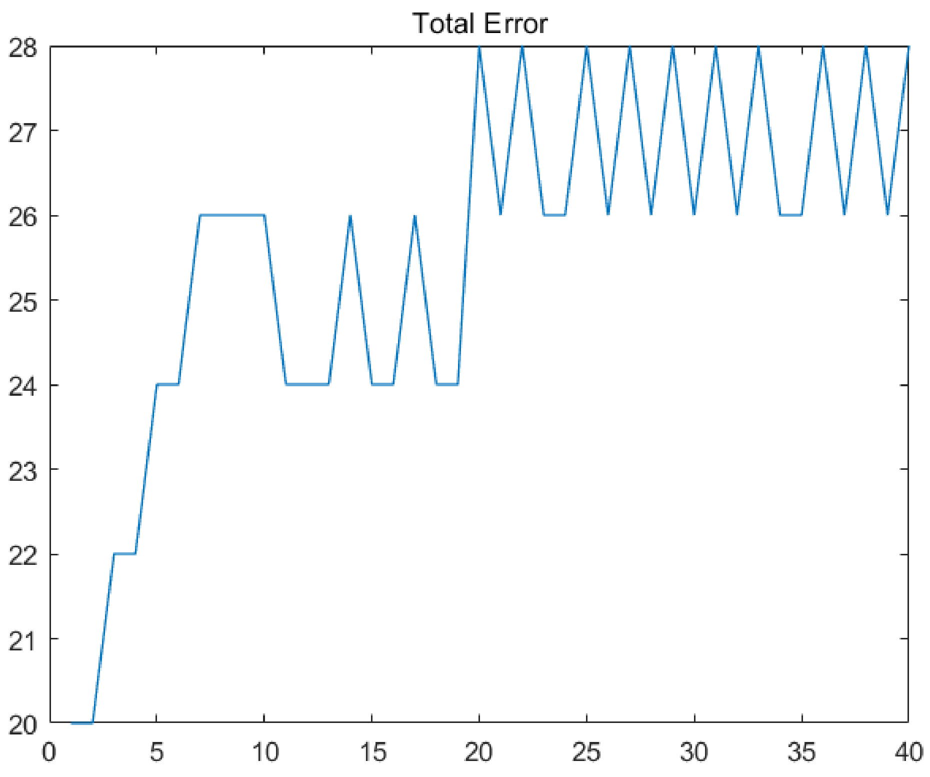
NOT linearly separable Dataset

As the dataset is non-separate by using hyper plain, so from Graph 3.1 we can see that the Weights and Bias parameters of the perceptron are non-convergence, and the dataset cannot be correctly divided as well.



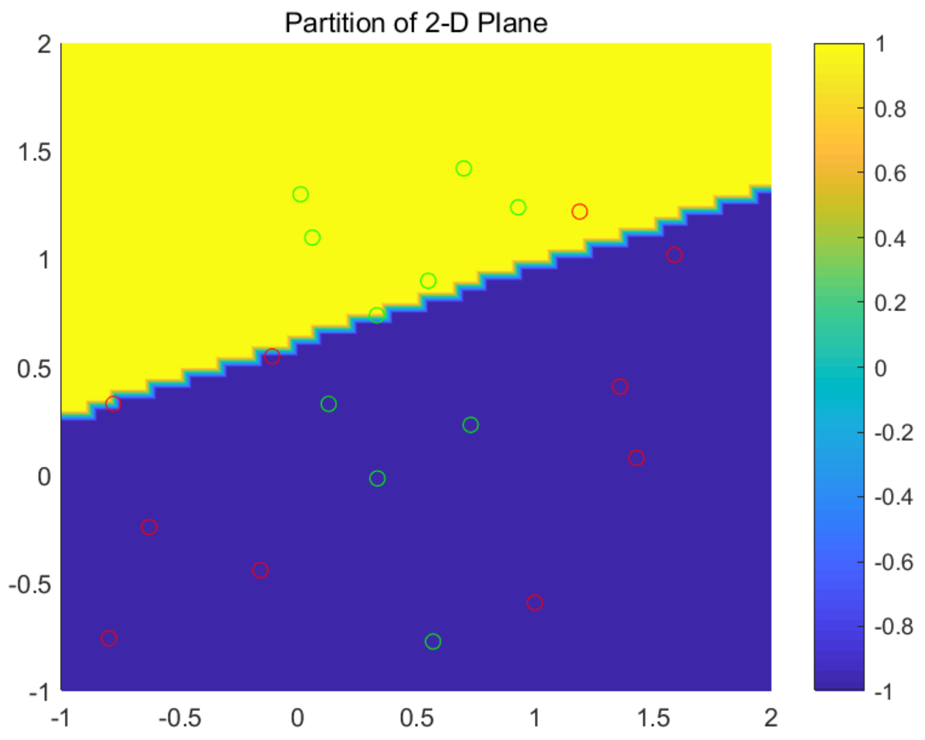
Graph 3.1

The Graph 3.2 demonstrates the non-convergence of the total error.



Graph 3.2

Hence, by these proofs we can conclude that the nonlinear dataset cannot be correctly divided, as the Graph 3.3 in the following:



Graph 3.3

Part 2 Artificial Neural Network

Introduction

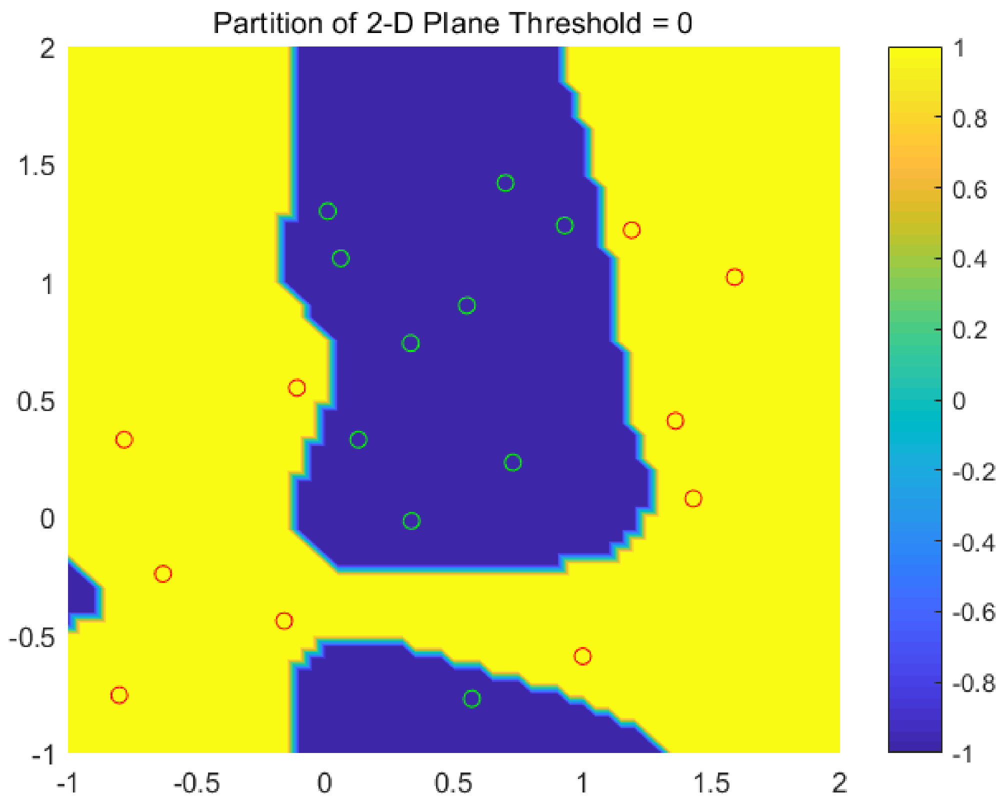
A multilayer neural network is a feedforward neural network with one or more hidden layers. The network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons.

Partition of 2D Plane

Next, I use the MATLAB nntoolbox to generate the neural network and use the same dataset as been used in Part 1, the nonlinear dataset.

As the Graph 4 shows, the data points are correctly divided by the margins.

This means the neural network can fit the any function and regulation (One hidden layer for continuous function), which cannot be done only by naïve perceptron.

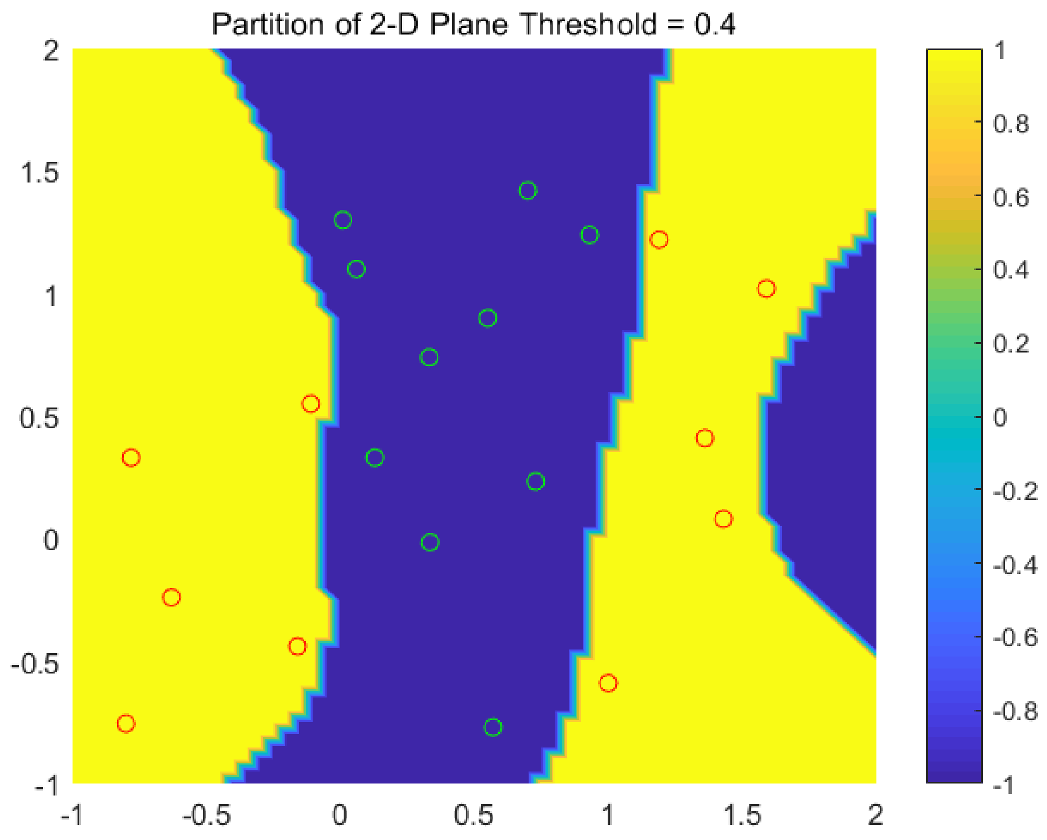


Graph 4

Next, we will find that if we change the threshold value, different result will be given.

The threshold means the sensitive degree when we use classification, the higher the threshold, it means the higher confidence we regard the object as another class.

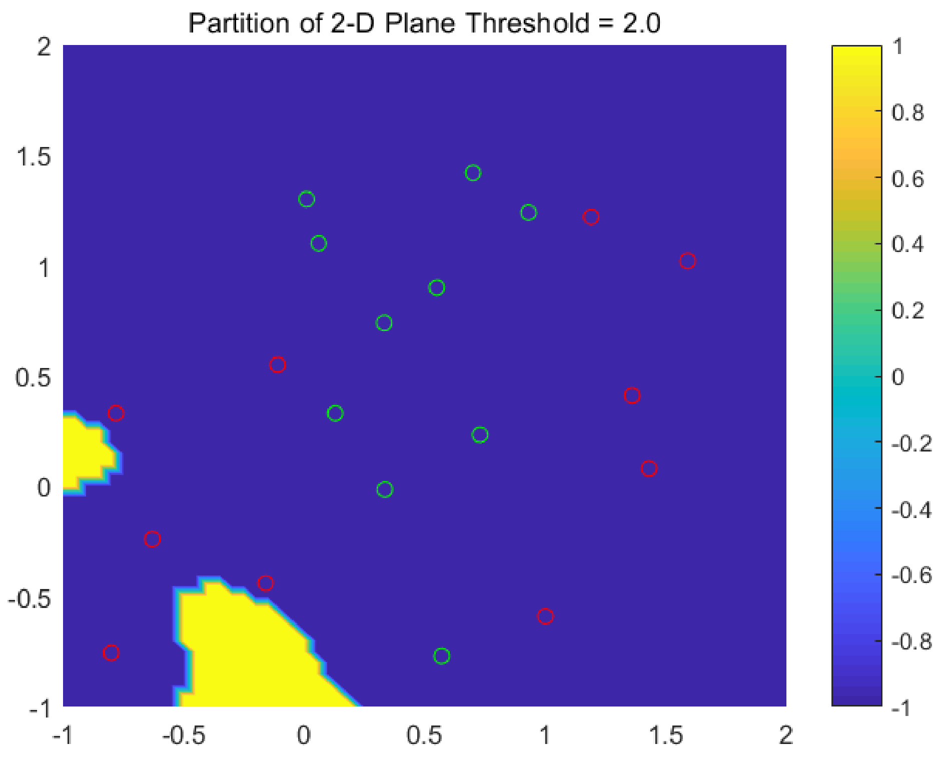
However, if we set the abnormal value of the threshold, we may get the wrong partition of the dataset. The different pictures of threshold as follow:



Graph 5

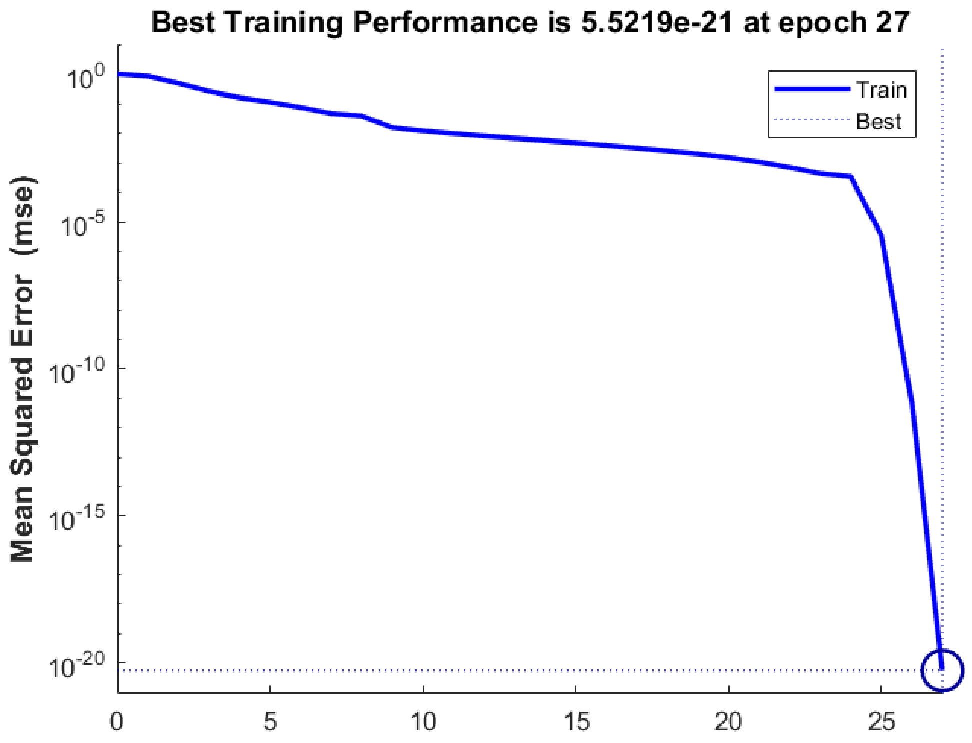
From Graph 5, we can see the partition margin as above, some points are at the margin of the picture, which means the possible wrong classification of the data.

If we change the value into threshold = 2, the totally incorrect answer will be given. This means the wrong value of threshold chose. The Graph 6 shows the result as follows.



Graph 6

Hence, we **cannot** set the threshold casually, which may result in the bad result. Finally, The Graph 7 shows the correct error decreasing process, meaning the convergence of the learning algorithm.

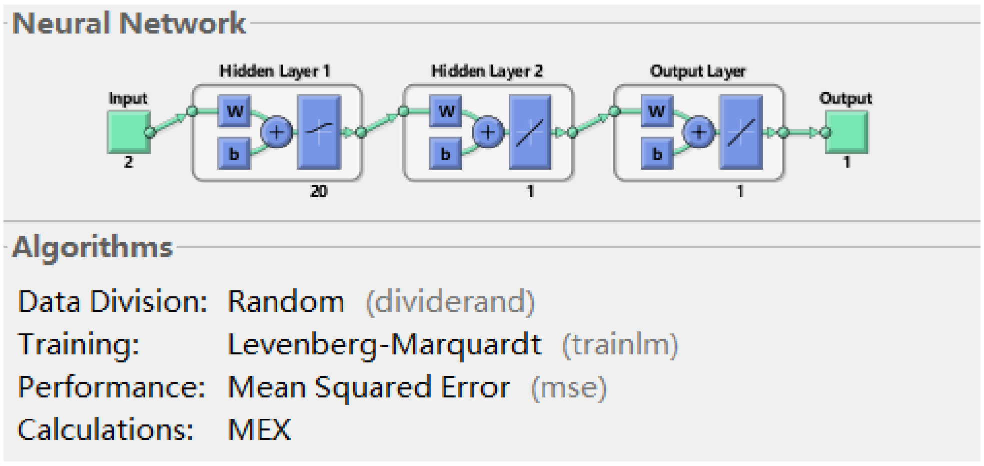


Graph 7

Part 3 Discussion

Structure and Parameters of ANN

Firstly, the structure of ANN as follows:



Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Graph 8

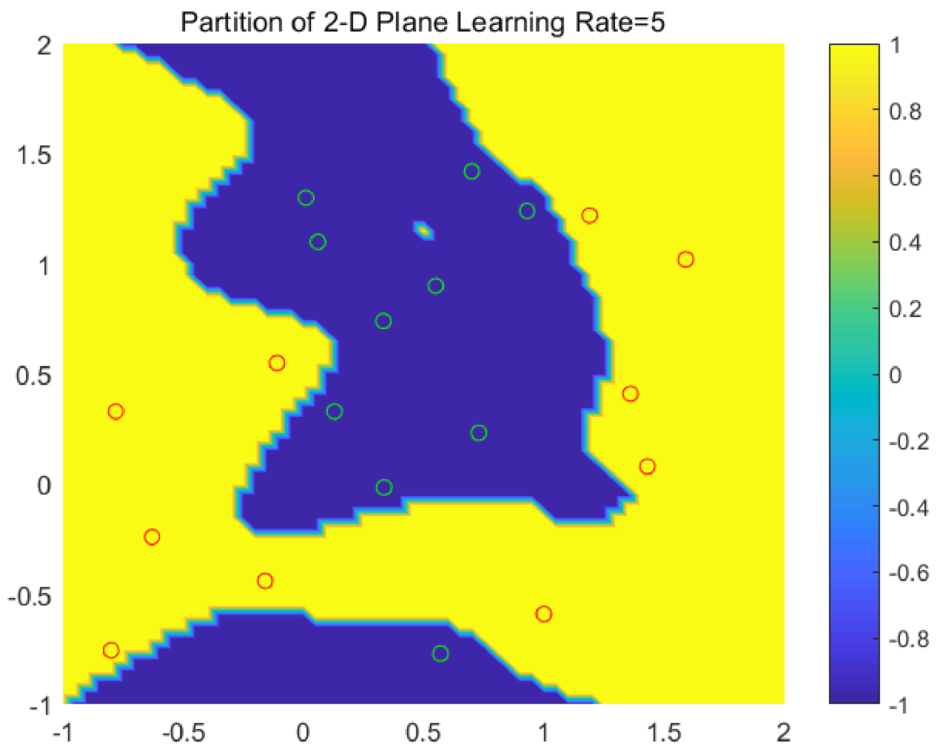
Secondly, the Numerical Array of Weights and bias are as follows:

net.IW{1}	net.IW{2}	net.b{1}	net.LW{1}	net.b{2}	net.lW{2}	net.b{3}
-6.5715	-6.7150	10.7763	1.2579	3.7835	5.4346	4.1648
8.9244	-4.8706	-4.9714	-2.0495			
-7.1477	-6.8632	8.8739	0.2934			
-5.1219	7.8525	2.9170	-4.1297			
-2.5572	-11.0574	-1.2274	2.9522			
-7.3927	-8.5725	-2.0874	-1.8892			
-5.8742	-5.7900	-2.6948	1.8706			
9.8724	4.5504	5.4467	-0.0987			
7.6795	4.4574	7.4790	0.0274			
2.9547	8.2793	8.8027	0.7262			

Table 1

Learning Rate

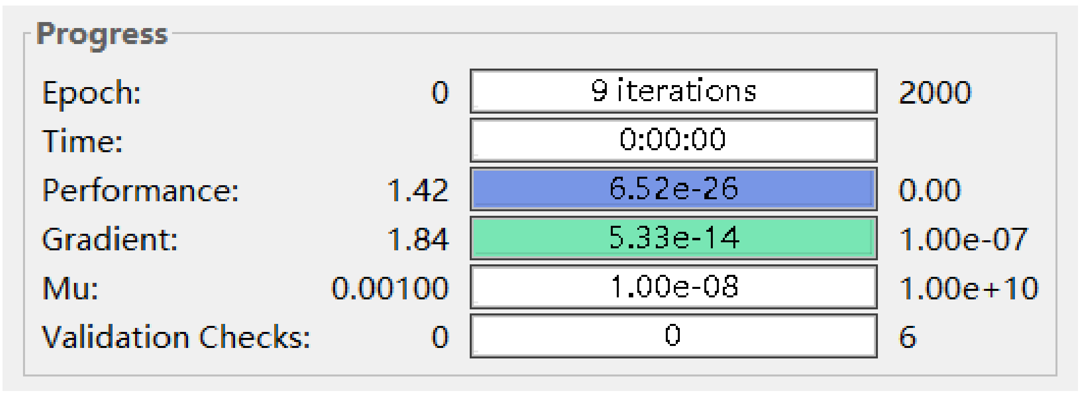
In the following part, we will discuss the corresponding results by changing the hyper parameters like learning rate, hidden nodes, etc.



Graph 9

From the Graph 8 above, we can see that if we set the Learning Rate bigger than the normal value, we can sill gain the correct answer.

Besides, like the Graph 9 as follows, the Iteration to be needed to arrive the best performance is smaller than that of the previous, which means the faster learning process of the model.



Progress

Epoch:	0	9 iterations	2000
Time:		0:00:00	
Performance:	1.42	6.52e-26	0.00
Gradient:	1.84	5.33e-14	1.00e-07
Mu:	0.00100	1.00e-08	1.00e+10
Validation Checks:	0	0	6

Graph 10

However, in the case of big dataset, if we set the high learning rate, it may result in the explosion of loss value, and the oscillation of loss as well.

Hence, we should set the learning rate in a lower degree. Nevertheless, if we use the very small learning rate, we can also get the correct partition result, but it will be in a slower process.

Abstractly, the much smaller learning rate may also get the correct answer, but it may lead to the overfitting in the training dataset, but poor performance will be gained on the test dataset.

In a conclusion, the learning rate must be balanced in a appropriate value to get the good model.

Hidden Layer Nodes

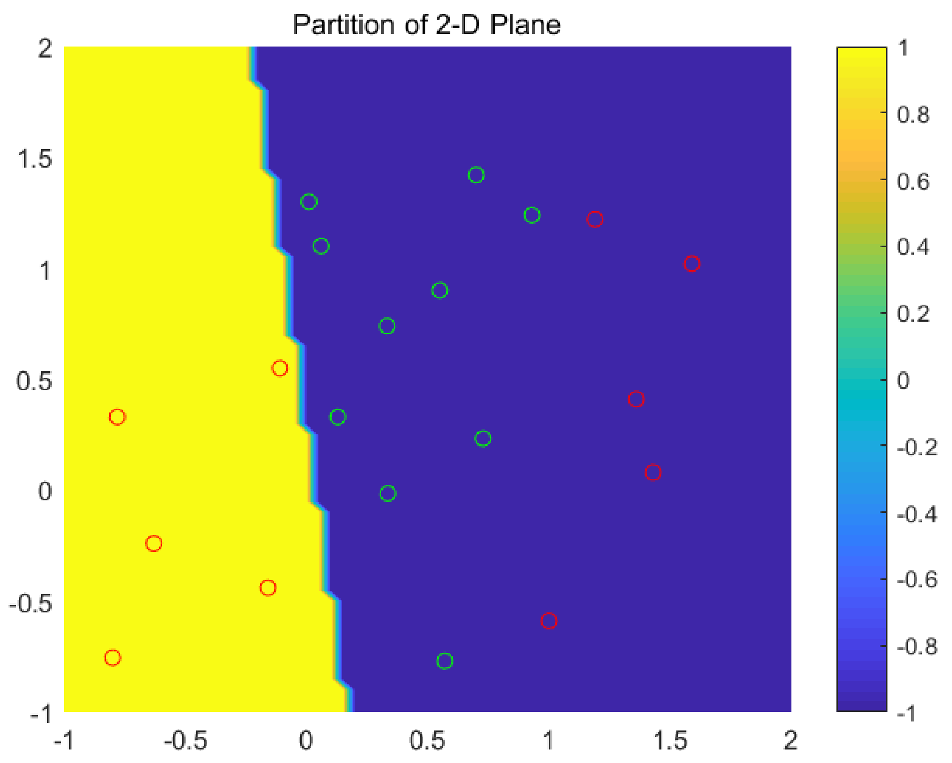
We must note that the Nonlinear Separable Dataset cannot be divided by the linear function, either single layer or multiple layer. It is because, mathematically, multiple layer linear connection can be processed to the form of single layer linear connection, which equals to the form of Perceptron (like Graph 3.3). Hence, we must set the nonlinear element in the hidden layer nodes, usually we use sigmoid function.

Generally speaking, there are so many ways to decide the number of hidden layer nodes. For example, one solution is that the hidden layer nodes approximately equal to the number of lines which needed to divide the dataset, just for reference, and case by case. In this example, we need two lines or nodes to divide the data.

However, if we use more nodes, it will increase the number of parameters, which means the more computational resources to be used, and even overfitting, which means great performance at the train set, but poor at test set.

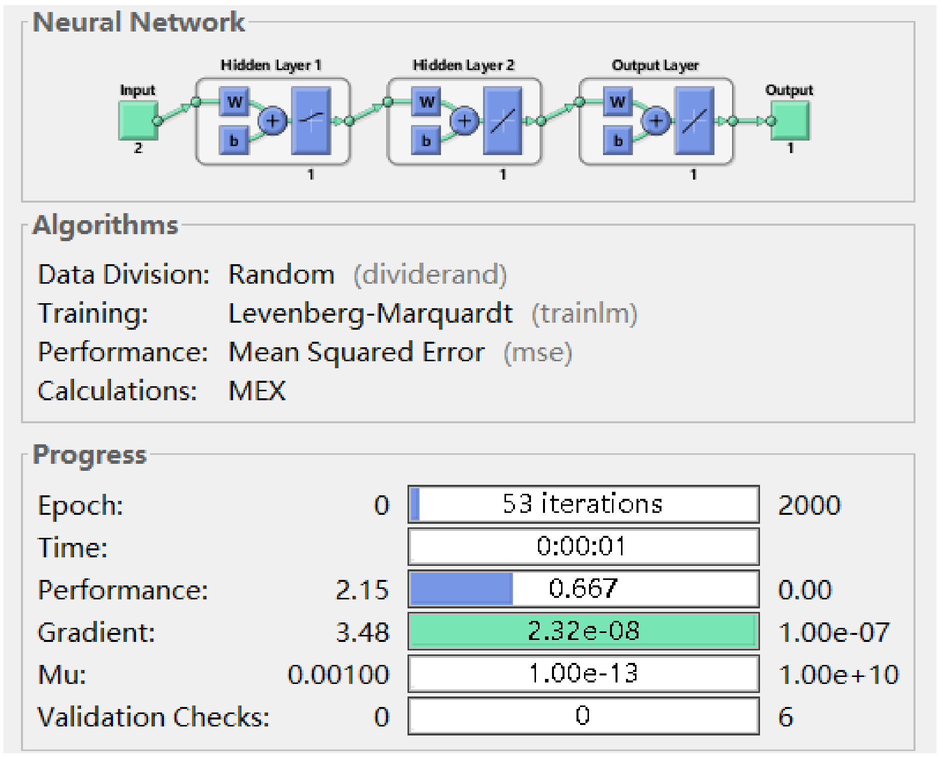
So, the appropriate number of parameters (nodes) should be used. In this 20-points-composed dataset, the test set equal to the train set, so we acquire the good result.

If the scale of dataset is bigger, the performance will be poor at test set when hidden layers nodes are far more than the right value range.



Graph 11

Conversely, if we do not use the enough number of hidden nodes compared to the appropriate value, we cannot get the right partition of the dataset (Graph 11), the result shows that the learning ability of our artificial neural network is weaker (Graph 12, as follows).



Algorithms

Data Division:	Random (dividerand)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Calculations:	MEX

Progress

Epoch:	0	53 iterations	2000
Time:		0:00:01	
Performance:	2.15	0.667	0.00
Gradient:	3.48	2.32e-08	1.00e-07
Mu:	0.00100	1.00e-13	1.00e+10
Validation Checks:	0	0	6

Graph 12

Part 4 Summary and Conclusion

As the summary, firstly I used the perceptron to successfully divide the linear separable data points, but performed poor on the nonlinear distribution dataset, this introduced the artificial neural network (ANN).

The artificial neural network performed good at the partition of 2D plane of the dataset, however we can get the various property of the neural network by changing the hyperparameters of the network structure.

From the experience above, the importance should be attached to the tuning of the structure and activation function used by us. Only when appropriate hyper parameters used by the structure can it achieve the better performance. Meanwhile, the learning rate and complexity of the structure will affect the speed of learning as well.

Reference

[1] https://www.ijcsmc.com/docs/papers/November2014/V3I11201499a19.pdf

[2] https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

[3] https://www.jeremyjordan.me/nn-learning-rate/

[4] https://en.wikipedia.org/wiki/Perceptron

[5] http://www.saedsayad.com/artificial_neural_network_bkp.htm

[6] https://en.wikipedia.org/wiki/Artificial_neural_network

Appendix: MATLAB Code

Perceptron

```
classdef Perceptron
    % define the class Perceptron
    properties
    % define the variables involved in the class
        Inputs;
        Labels;
        LearningRate;
        Iteration;
        w;
        b;
        w1_revise; % record the trend of change of w1
        w2_revise; % record the trend of change of w2
        b_revise; % record the trend of change of bias
        loss;
        error_cal; % record dynamic Total Error
    end

    methods  % define the functions used
        function obj = Perceptron() % Constructor
        end

        function obj = setInfo(obj,Inputs, Labels,
LearningRate, Iteration)
% core setInfo
            obj.Inputs = Inputs;
            obj.Labels = Labels;
            obj.LearningRate = LearningRate;
            obj.Iteration = Iteration;
        end

        function obj = Learning(obj)
% Learning Algorithm
            [m, n] = size(obj.Inputs);
            obj.w = rand(1, n)-0.5;
            obj.b = 0.8;
            obj.w1_revise = zeros(1, obj.Iteration);
% Zero array to be inserted in
            obj.w2_revise = zeros(1, obj.Iteration);
            obj.b_revise  = zeros(1,  obj.Iteration);
            obj.error_cal = zeros(1,  obj.Iteration);
            for N=1:obj.Iteration
                totalerror = 0;
% Clear totalerror by every iteration
                for set= 1:m
                    output =
sign(obj.Inputs(set,:)*(obj.w)' + obj.b );
                    error = obj.Labels(set) - output;
                    obj.w = obj.w + obj.LearningRate *
error * obj.Inputs(set);
                    obj.b = obj.b + obj.LearningRate *
error;
                    totalerror = totalerror +
abs(error);
                end
                obj.w1_revise(1, N) = obj.w(1);
                obj.w2_revise(1, N) = obj.w(2);
                obj.b_revise(1, N) = obj.b;
                obj.error_cal(1, N) = totalerror;
            end
        end
    end
end
```

Plot 1 Linear Division

```
cell = Perceptron; % acquire class
Iteration = 2000;
LearningRate = 0.55;
Inputs = [-0.329,1; -0.71,-0.2; 0,0.75; 0.12,-0.24; -
0.3,0.85;
          0.81,0.61; 0.72,0.9; 1.03,0.48; 1.27,0.42;
1.41,1.2]; % Datasets
Labels = [1,1,1,1,1,-1,-1,-1,-1,-1];

cell = cell.setInfo(Inputs, Labels, LearningRate,
Iteration);
cell = cell.Learning();
figure(1);
    [XXX,YYY]=meshgrid(-1:0.05:2);
    RRR=zeros(size(XXX));

    for x = 1:61
```

```
        for y = 1:61
            % Obtain an input data vector / binary
            feature
            I = [ XXX(x,y) YYY(x,y) ];
            % Calculate the output axis z
            tmpout = I(1)*cell.w(1) + I(2)*cell.w(2) +
cell.b;
            if (tmpout > 0) % default threshold
                out = 1; RRR(x,y) = 1;
            else
                out = -1; RRR(x,y) = -1;
            end
        end
    end

    Z1 = [1,1,1,1,1];
    Z2 = [-1,-1,-1,-1,-1];
    % figure(1);
    plot3(Inputs(1:5, 1),Inputs(1:5, 2),Z1, 'ro');
    hold on
    plot3(Inputs(6:10, 1),Inputs(6:10, 2),Z1, 'go');
    % for convinence, set the points upside 3-d and
    view(2)
    % legend('Class 1','Class 2');
    title('Partition of 2-D Plane');
    hold on
    % mesh(XXX,YYY,RRR),view(2),colorbar;
    % hold on
    surf(XXX,YYY,RRR),view(2),colorbar ;
    shading interp
    hold off
    print(gcf,'-
dpng','C:\Users\Yunqing\Desktop\SEM2\fuzzy\Assignment
1\Plot1.png');

figure(2); % show the convergence of the partameters
of Perceptron
    plot(cell.w1_revise(1:40), 'r-
','LineWidth',1.3); % w1
    hold on
    plot(cell.w2_revise(1:40), 'k-
','LineWidth',1.3);% w2
    hold on
    plot(cell.b_revise(1:40), 'b-
','LineWidth',0.8); % b
    hold on
    plot(cell.error_cal(1:40), '-
','LineWidth',0.8); % total error
    legend('w1', 'w2', 'b', 'error'); % cutline
    title('Convergence of Parameters');
    print(gcf,'-
    dpng','C:\Users\Yunqing\Desktop\SEM2\fuzzy\Assign
    ment1\Linear_Convergence.png'); % to specified
    address
    hold on
    ylim([-5 5]); % limit axis_y horizon
    hold off
```

Plot 2 Non-Linear Division

```
cell = Perceptron; % acquire class

Iteration = 2000;
LearningRate = 0.00549;
Inputs = [-0.8,-0.755; -0.781,0.33; -0.63,-0.24; -
0.16,-0.44; -0.11,0.55;
          1.36,0.41; 1.59,1.02; 1.43,0.08; 1.19,1.22;
1,-0.59;
          0.729,0.233; 0.129,0.33; 0.335,-0.015;
0.332,0.74; 0.55,0.9;
          0.06,1.1; 0.7,1.42; 0.93,1.24; 0.57,-0.77;
0.01,1.3]; % augmented Datasets
Labels = [1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1];

cell = cell.setInfo(Inputs, Labels, LearningRate,
Iteration);
cell = cell.Learning();
% disp(cell.w1_revise);
figure(1);
    [XXX,YYY]=meshgrid(-1:0.05:2);
    RRR=zeros(size(XXX));

    for x = 1:61
        for y = 1:61
            % Obtain an input data vector
            I = [ XXX(x,y) YYY(x,y) ];
            % Calculate the output
            tmpout=I(1)*cell.w(1)+I(2)*cell.w(2)+
            cell.b;
            if (tmpout > 0)
                out = 1; RRR(x,y) = 1;
            else
                out = -1; RRR(x,y) = -1;
            end
        end
    end

    Z1 = [1,1,1,1,1,1,1,1,1,1];
    Z2 = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1];
    % figure(1);
    plot3(Inputs(1:10, 1),Inputs(1:10, 2),Z1, 'ro');
    hold on
    plot3(Inputs(11:20, 1),Inputs(11:20, 2),Z1, 'o');
    % for convinence, set the points upside 3-d and
    view(2)
    % legend('Class 1','Class 2');
    title('Partition of 2-D Plane');
    hold on
    % mesh(XXX,YYY,RRR),view(2),colorbar;
    % hold on
    surf(XXX,YYY,RRR),view(2),colorbar ;
    shading interp
    hold off
    print(gcf,'-
    dpng','C:\Users\Yunqing\Desktop\SEM2\fuzzy\Assign
    ment1\Plot2.png'); % to specified address

figure(2);
    plot(cell.w1_revise(1:80),'r-','LineWidth',1.3);
    hold on
    plot(cell.w2_revise(1:80), 'k-','LineWidth',1.3);
    hold on
    plot(cell.b_revise(1:80), 'b-','LineWidth',0.8);
    hold on
    legend('w1', 'w2', 'b');
    title('Convergence of Parameters');
    print(gcf,'-
    dpng','C:\Users\Yunqing\Desktop\SEM2\fuzzy\Assign
    ment1\NonConvergence1.png'); % to specified
    address
    hold on
    % ylim([-5 10]);
    hold off
figure(3);
    plot(cell.error_cal(1:40), '-','LineWidth',0.8);
```

Artificial Neural Network

```
classdef NN
    properties % define the variables involved
        Inputs;
        Labels;
        LearningRate;
        Iteration;
        w;
        b;
        w1_revise;
        w2_revise;
        b_revise;
        loss;
        error_cal;
        net
    end

    methods  % define the functions used
        function obj = NN()   % constructor
        end

        function obj = setInfo(obj, Inputs, Labels,
        LearningRate, Iteration)
        obj.Inputs = Inputs;
        obj.Labels = Labels;
        obj.LearningRate = LearningRate;
        obj.Iteration = Iteration;
        end

        function obj = Learning(obj)
            net=newff(minmax(obj.Inputs),obj.Labels,
[10 2],{'logsig', 'purelin'}, 'trainlm');
            net.trainParam.epochs=obj.Iteration;
            % maximum learning step=2000
            net.trainParam.goal=0; % training goal = 0
            net.trainParam.show=1;
            % every single step to plot result
            net.trainParam.lr=0.005; %learning rate
            net.divideParam.valRatio = 0/100;
            net.divideParam.testRatio = 0/100;
            net=train(net,obj.Inputs,obj.Labels);
            obj.net = net;
        end
    end
end
```

Plot for ANN

```
classdef NN
    properties % define the variables involved
        Inputs;
        Labels;
        LearningRate;
        Iteration;
        w;
        b;
        w1_revise;
        w2_revise;
        b_revise;
        loss;
        error_cal;
        net
    end

    methods  % define the functions used
        function obj = NN()   % constructor
        end

        function obj = setInfo(obj, Inputs, Labels,
        LearningRate, Iteration)
        obj.Inputs = Inputs;
        obj.Labels = Labels;
        obj.LearningRate = LearningRate;
        obj.Iteration = Iteration;
        end

        function obj = Learning(obj)
            net=newff(minmax(obj.Inputs),obj.Labels,
[10 2],{'logsig', 'purelin'}, 'trainlm');
            net.trainParam.epochs=obj.Iteration;
             % maximum learning step=2000
            net.trainParam.goal=0; % training goal = 0
            net.trainParam.show=1;
             % every single step to plot result
            net.trainParam.lr=0.005; %learning rate
            net.divideParam.valRatio = 0/100;
            net.divideParam.testRatio = 0/100;
            net=train(net,obj.Inputs,obj.Labels);
            obj.net = net;
        end
    end
end
```