Suyog Daga
EMI HW6
UFID: 57735749

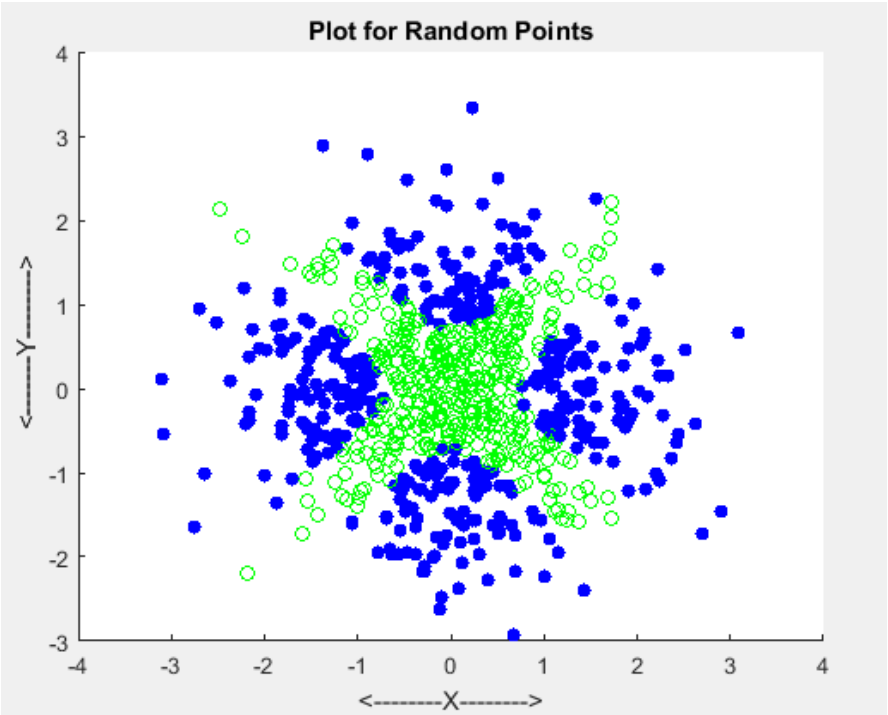
- For training the neural network I have following structure, 3 input neurons: 2 for regular inputs and 1 for bias, 5 hidden layer neurons: 4 neurons for hidden layer and 1 for bias and since the output is binary , I have used 1 neuron in output layer . So overall , the structure for network input-hidden-output , is 3-5-1 where 1 neuron in input and hidden corresponds to bias respectively.
- I started with single neuron in hidden layer and trained the data, but the expected binary could not be achieved. Then I started to increase the hidden layer neuron's , similarly for the case of 2 and 3 neuron's in hidden layer there was change in expected output, but still there was some misclassification. Then for 4 hidden neuron's I got the expected output.
- Next is how many hidden layers and correspondingly how many neuron's in hidden layer should we take ?

1. For hidden layers: Here is some general idea for hidden layers[1]

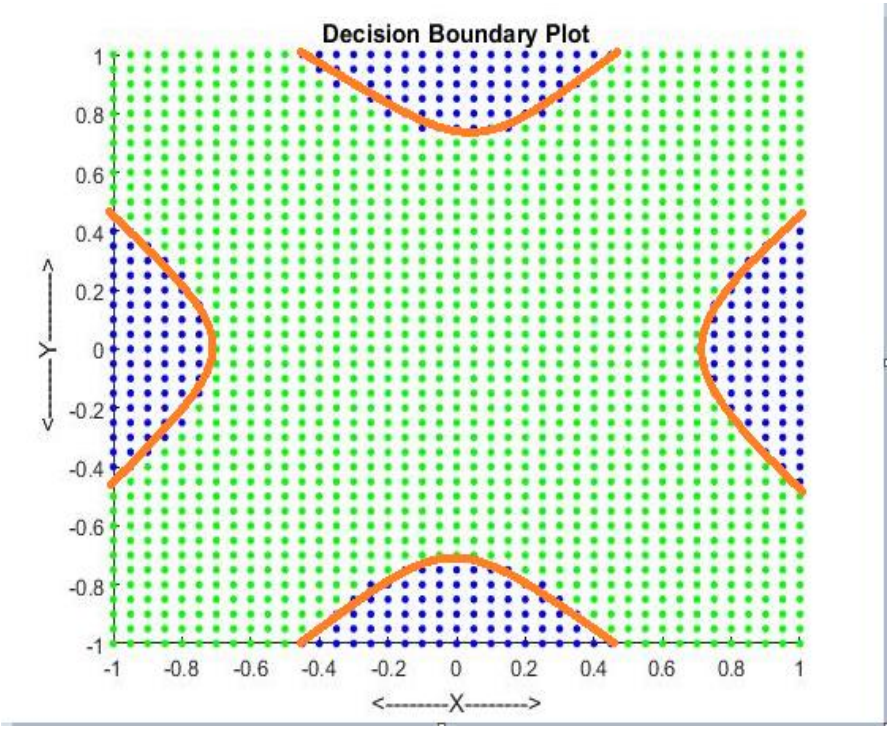
Hidden layers	Result
0	Linear separable function or decision
1	Can be used to approximate a function that contains a continuous mapping from one finite space to another
2	Can be used to represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

2. For number of neurons:
   If we use less neuron's , it can result in underfitting of data. On the other hand, using to many neuron's in hidden layer can cause overfitting of data. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers[1].

- After training the data , and getting the desired binary output for the training data, I have tested the data on random data as well as the interval specified in the question, below are the plots for both the cases.



In this plot, green points are the one classified as ‘0’ and blue ones are the points classified as ‘1’. This plot was generated for random points.



Here points in blue are the classified as ‘1’ and points in green are classified as ‘0’.

- The multilayer neural network doesn’t learn the solution as expected. One method for improving network generalization is to use a network that is just large enough to provide an adequate fit. The larger network you use, the more complex the functions the network can

create .Also, if we take a regularization of parameters, say L2 regularization we can improve generalization accuracy of neural network.

- Matlab Codes:

1) Run Homework6.m and then testing_data.m

% Code for HomeWork6.m

```
%%
clear;
clc;
% inputting the data
input=[ 1,0;0,1;-1,0;0,-1;0.5,0.5;-0.5,0.5;0.5,-0.5;-0.5,-0.5];
output=[1;1;1;1;0;0;0;0];



% weights for network and total number of records
weight_1 = randn(3,4);
weight_2 = randn(1,5);
sample = 8;

% train data
%setting number of iterations and eta
iter = 1000;
eta = 3;
%%

%training data

for k = 1:iter
% calculating delta values for each iterations and adjusting them later

delta_output=zeros(1,5);
delta_hidden=zeros(3,4);

for i = 1:sample
   % adding bias for initial to hidden layer
   a_initial=[1,input(i,:)];

   % calculating Z fo hidden layer
   Z_hidden=a_initial*weight_1;

   % calculating sigmoid function values
   a_hidden=sigmoid_data(Z_hidden);

   % adding bias for hidden to output layer
   a_hidden_output=[1,a_hidden];

   % calculating Z for output layer
   Z_output=weight_2*a_hidden_output';

   % calculating activaion value for output layer
```

```
    a_output=sigmoid_data(Z_output);

    % miscalculation in output
    error_output = a_output - output(i,1);

    %backpropogation
    error_hidden = weight_2.*error_output.*sigmoid_derivative(a_hidden_output);

    % calculating delta

    delta_output = delta_output + a_hidden_output*error_output;
    delta_hidden = delta_hidden + a_initial'*error_hidden(:,2:5);
end
 % updating weights
weight_1=weight_1-eta*delta_hidden/sample;
weight_2=weight_2-eta*delta_output/sample;

end

%%
%testing data for original values
expected_output=zeros(sample,1);
output_values =zeros(sample,1);
for i=1:sample
    % adding bias for initial to hidden layer
    a_initial=[1,input(i,:)];

    % calculating Z fo hidden layer using trained weights
    Z_hidden=a_initial*weight_1;

    % calculating sigmoid function values for hidden layer
    a_hidden=sigmoid_data(Z_hidden);

    % adding bias for hidden to output layer
    a_hidden_output=[1,a_hidden];

    % calculating Z for output layer
    Z_output=weight_2*a_hidden_output';

    % calculating sigmoid function value for output layer
    output_values(i,1)=sigmoid_data(Z_output);

    % setting threshold value to 0.5
    threshold = 0.5;
    if(output_values(i,1)>=threshold)
        % setting expected output to 1 if value > 0.5
        expected_output(i,1)=1;
    else
        % setting expected output to 0 if value < 0.5
        expected_output(i,1)=0;
    end
end
```

% code for testing_data.m

```
%testing data for random values
wt_1 = weight_1;
wt_2 = weight_2;
for t = 1: 100
   input = randn(8,2);

for i=1:sample
   % adding bias for initial to hidden layer
   a_initial=[1,input(i,:)];

   % calculating Z fo hidden layer
   Z_hidden=a_initial*wt_1;

   % calculating sigmoid function values
   a_hidden=sigmoid_data(Z_hidden);

   % adding bias for hidden to output layer
   a_hidden_output=[1,a_hidden];

   % calculating Z for output layer
   Z_output=wt_2*a_hidden_output';

   % calculating activaion value for output layer
   a_output=sigmoid_data(Z_output);

   % setting threshold value to 0.5
   threshold = 0.5;
   figure(1)
   if(a_output>=threshold)
      % setting expected output to 1 if value > 0.5
      % scatter plot of original values
      scatter(input(i,1),input(i,2),'filled','blue');
      hold on;
   else
      % setting expected output to 0 if value < 0.5
      % scatter plot accordingly
      scatter(input(i,1),input(i,2),'green');
      hold on;
   end
end
end
figure(1);
xlabel('<---------X-------->');
ylabel(' <-------Y-------->');
title('Plot for Random Points');
%% plotting decision boundary

% setting xmin and xmax cordinates and change
xmin = -1 ; xmax = 1; change = 0.05;
% setting ymin and ymax cordinates and change
```

```
ymin = -1 ; ymax = 1; change1 = 0.05;

x_limit=[xmin xmax];
y_limit=[ymin ymax];
figure(2);
hold on;
for x=xmin:change:xmax
   for y=ymin:change:ymax
      input = [x,y];
      a_initial=[1,input];

   % calculating Z fo hidden layer
   Z_hidden=a_initial*wt_1;

   % calculating sigmoid function values
   a_hidden=sigmoid_data(Z_hidden);

   % adding bias for hidden to output layer
   a_hidden_output=[1,a_hidden];

   % calculating Z for output layer
   Z_output=wt_2*a_hidden_output';

   % calculating activaion value for output layer
   a_output=sigmoid_data(Z_output);

      if a_output > 0.5
         %scatter(x,y,'filled','red');
         plot(input(1),input(2),'.b','markersize',10);
      else
         %scatter(x,y,'filled','green');
         plot(input(1),input(2),'.g','markersize',10);
      end
      hold on;
   end
end
xlabel('<--------X-------->');
ylabel(' <-------Y-------->');
title('Decision Boundary Plot');
```

- References :
  1. Introduction to Neural Network in Java, Jeff Heaton.
  2. Andrew Ng's tutorial on Neural Networks.

- Note : This code was programmed by me , code has been attached as per request.
  Also for clarifications I have discussed my doubts with Ruturaj, Kunal and Devyash.