Homework 6
Devyash Sanghai - 33831564

1.

Make sure to note the size of the network that you used and explain why you used that number of hidden processing elements?.

Since there is no thumb rule for deciding the determining the hyper parameters.”The selection criterion attempts to "grow" the network beginning with a small initial number of hidden layer nodes (as opposed to pruning a relatively large network).”[1]
The hyperparameters such as the step size/learning rate(eta), Number of iterations for training(k), Number of hidden layer, Number of neurons in the hidden layer, Regularization parameter. It is impractical to optimize the hyper-parameters due to it being high computationally expensive. There are many ways the learning of neural networks can be implement(Batch method,mini batch method,online). Here we have used cross entropy error instead of mean square error as a cost function. Cross entropy gives us ‘convex’ cost function, where as mean square error gives a ‘non convex’ cost function which does not converge to global minima for non linear(sigmoid) neural networks.

The hyperparameters selected by me are:

- Hidden Layer: 1 hidden layer
- Number of neurons in hidden layer: 4(Not including the bias)
- Step size:0.5
- Number of iterations:10000
- Regularization parameter:0

I have taken only 1 hidden layer to reduce computational complexity and as I do not need to add more layers in the neural network as it is a simple classifier. I initially tried to train the classifier using just a single neuron in the hidden layer, however it did not give me the correct output.It classifying linearly. For two, and three the the decision boundary is not appropriate to the output as required.

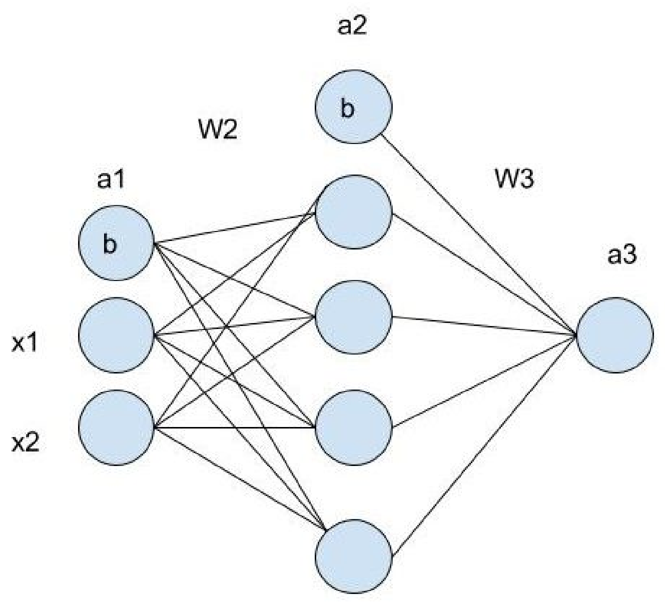

Fig:Neural network overview

In the above neural network diagram, I have taken 5 neurons in the hidden layer(Including bias) which represents the actual structure of neural network in consideration.Here x1 and x2 represent the input b-> bias, a1 the activation function of the input layer, a2 represents the activation function of the hidden layer and a3 is the output activation function.

2. **Plot the corresponding decision boundaries after the networks have finished training.? As well, see how well your solution performs on samples that do not belong to the training set by generating additional samples that follow the same pattern. Keep these generated points within a square that has length 2 on each of its sides.**

The decision boundary for 4 neurons (+1 for bias) is shown below:

- The red color dots represent the class “1” as show in the scatter plot below.
- The green color dots represent the class “0” as shown in the scatter plot below

Empirically, I was getting similar plots even after increasing the number of neurons.

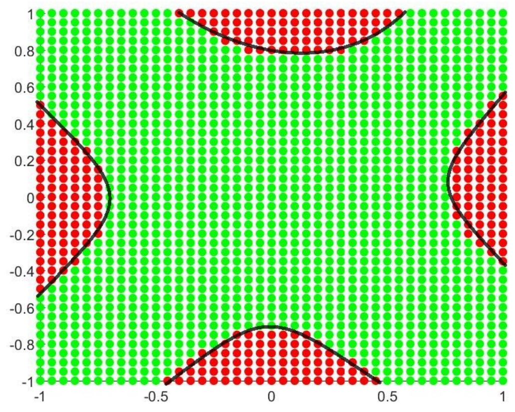

Fig: Plot with decision boundary

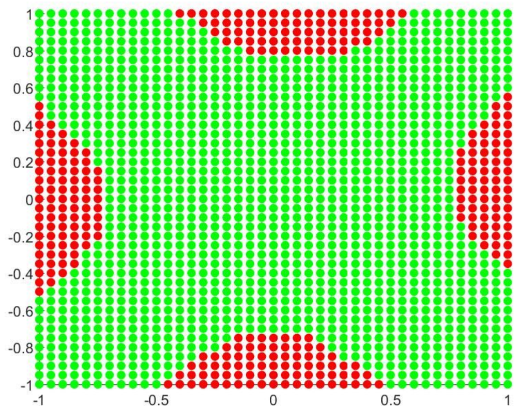

Fig: Plot without decision boundary

Although the question requires only randomly generated points within a square of length 2, Below I have plotted the scatter plot of 800 randomly generated points . I have on purpose plotted more points to show that our trained neural network classifier can also be used to classify points outside the **scope of the training set.**

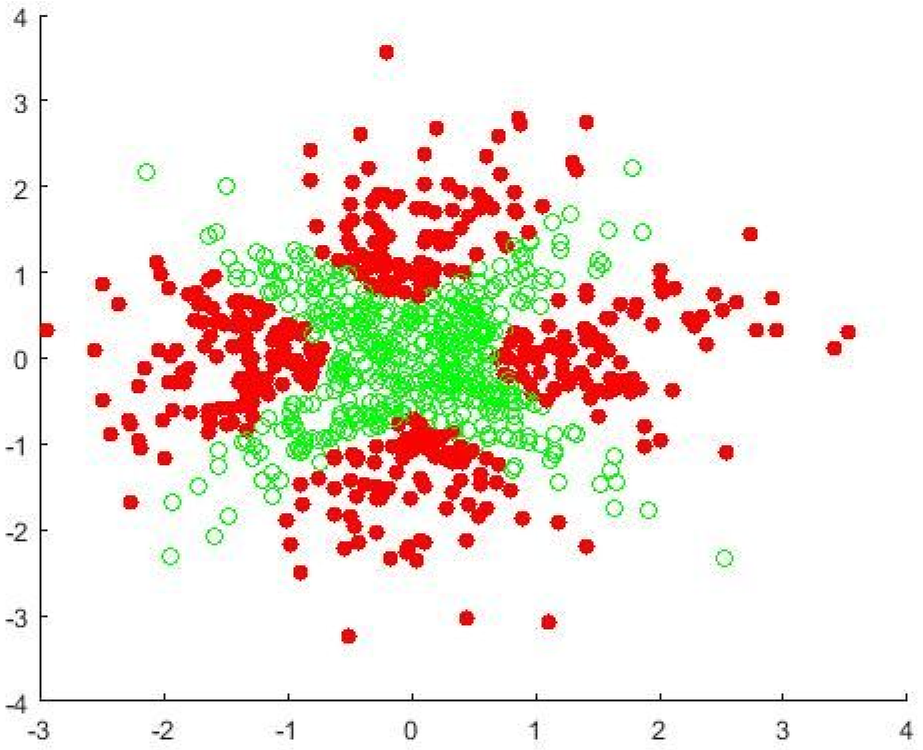

Fig: Randomly generated points

3) Lastly, explain what solution you would choose if you were going to hand-select the parameters of the network. Does the multi-layer neural network learn the solution that you expected? How would you improve the generalization accuracy of the multi-layer neural network?

If I were to hand select the parameters if the network, I would choose initial weights of the neural network as close to the local minima and use **L2 regularization** to avoid overfitting of data.

On the other hand, Selecting the hyper parameter would depend on various factors, firstly I need to make sure I do not overfit the neural network on the training set as well as not underfit the data. The hyperparameters selected by me are:

- Hidden Layer: 1 hidden layer
- Number of neurons in hidden layer: 4(Not including the bias)
- Step size:0.5
- Regularization parameter:0
- Cost function: Cross Entropy

I selected these hyper parameters firstly as I thought the output decision boundary plot would look like a square, but to my surprise the output plot looks more like an X.So **No, the neural network does not get the** output I was initially excepting. But again it depends on the cost function and training of the neural network. To improve the generalization accuracy also known as the out-of-sample error) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.

- Generalization error can be minimized by avoiding overfitting in the learning algorithm
- Increase the number of hidden layers and seeing the learning curve plot, if it reaches a lower value.
- Increase the number of neurons and seeing the learning curve plot, if it reaches a lower value.
- Decrease the step size and in turn increase the number of iterations, but at the same time keeping in mind not to decrease the step size to too low, else the function would never converge.
- Taking different values of regularization parameter. Which in this case I have taken 0. To avoid coding complexity.

Reference:
[1] http://www.dtic.mil/dtic/tr/fulltext/u2/a247725.pdf
[2] Discussion with Suyog Daga, Ruthuraj, Kunal, and Harshit .

Please find the attached code in the rar file. You can also find the code on github : **https://github.com/devyash/MachineLearning_Matlab**

To run the code:

1) **Extract the folder**
2) **Run the main.m**

It will plot both the decision boundary plot and the random generated points plot.

Main.m

```
%clear the previous data
clear;clc;

%X is the training input Y is the training output matrix
X=[ 1,0;0,1;-1,0;0,-1;0.5,0.5;-0.5,0.5;0.5,-0.5;-0.5,-0.5];
Y=[1;1;1;1;0;0;0;0];

%random weights initialized
W2= randn(3,4);
W3= randn(1,5);

%Number of training samples
m=8;

%Train on input the last two parameters are Number of iterations and eta
[W2,W3] = train(W2,W3,X,Y,m,10000,0.5);

%Plotting the decision boundary
decisionBoundary(W2,W3,-1,1,0.1,-1,1,0.1);

%Scatter plot for random samples in multiples of 8 also used for testing fr
%input data, if it is trained efficiently
for i = 1:100
testmatrix=randn(8,2);
[OUTPUT] = test(testmatrix,m,W2,W3);
end
[OUTPUT] = test(X,m,W2,W3);
```

Decision Boundary.m

```
function decisionBoundary(W2,W3,xmin,xmax,dx,ymin,ymax,dy)
%funtion gives decision boundary plot
xlim([xmin xmax])
ylim([ymin ymax])
hold on;
for x=xmin:dx:xmax
        for y=ymin:dy:ymax
                activation = forwardPropagate([x y],W2,W3 );
                if activation > 0.5
        scatter(x,y,'filled','red');
      else
        scatter(x,y,'filled','green');
                end
                hold on;
        end
end

end
```

forwardPropagate.m

```
function [a3] = forwardPropagate( X,W2,W3 )
%function for forward propagation of neural network
   a1=[1,X(1,:)];
   Z2=a1*W2;
   a2withoutbias=(1+exp(-Z2)).^-1;
   a2=[1,a2withoutbias];
   Z3=W3*a2';
   a3=(1+exp(-Z3)).^-1;

end
```

test.m

```
function [OUTPUT] = test( X,m,W2,W3 )
%funtion for testing of the trained neural network returns the plot of the
%neural network
OUTPUT=zeros(m,1);
for i=1:m
   a1=[1,X(i,:)];
```

```
    Z2=a1*W2;
    a2withoutbias=(1+exp(-Z2)).^-1;
    a2=[1,a2withoutbias];
    Z3=W3*a2';
    a3=(1+exp(-Z3)).^-1;
    figure(1)
    if(a3>=0.5)
        OUTPUT(i,1)=1;
        scatter(X(i,1),X(i,2),'filled','red');
        hold on;
    else
        OUTPUT(i,1)=0;
        scatter(X(i,1),X(i,2),'green');
        hold on;
    end

end
end
```

train.m

```
function [W2,W3] = train(W2,W3,X,Y,m,K,eta)
%for loop for iterations
for k=1:K
    delta3=zeros(1,5);
    delta2=zeros(3,4);
%for loop for number of samples of training set
for i=1:m
    %activation matrix of input layer with bias
    a1=[1,X(i,:)];
    %Output after the weight matrix and activation function are multiplied
    Z2=a1*W2;
    %Signmoid perceptron
    a2withoutbias=(1+exp(-Z2)).^-1;
     %activation matrix of hidden layer with bias
    a2=[1,a2withoutbias];

    %Output after the weight matrix and activation function are multiplied
    Z3=W3*a2';

     %activation matrix of output layer with bias
    a3=(1+exp(-Z3)).^-1;
    %calculation of partial derivative
    del3=a3-Y(i,1);
    del2=W3.*del3.*(a2.*(1-a2));
    %total cost
    delta3 = delta3 + a2*del3;
    delta2 = delta2 + a1'*del2(:,2:5);
end
%updating the weights
W2=W2-eta*delta2/m;
W3=W3-eta*delta3/m;
end
end
```