Πανεπιστήμιο Κύπρου
University of Cyprus

ECE 421

Department of Electrical and Computer Engineering

Charalambos Rotsides

15/11/2022

Assignment 2

Exercise 1:

A. This is a problem without an objective function

$$st. \sum_i -x_i * w_i > \epsilon$$

$$\sum_i y_i * w_i > \epsilon$$

```
clear all; close all; clc;
clear('model');
X = load('.\datasets\dataLinearSeparability.mat');

%model.obj = [1 2 3];

X0=X.X0;
X1=X.X1;
N0=length(X0);
N1=length(X1);
x=zeros(length(X0),length(cell2mat(X0(1))));
y=zeros(length(X1),length(cell2mat(X1(1))));
for i=1:length(X0)
    x(i,:)=cell2mat(X0(i))';
end
for i=1:length(X0)
    y(i,:)=cell2mat(X1(i))';
end
model.A = sparse([-x;y]);
%model.modelsense = 'Max';
model.sense(1:N0) ='>';
model.sense(N0+1:N0+N1)='>';
model.rhs(1:N0) =10^-3;
model.rhs(N0+1:N0+N1)=10^-3;
model.lb(1:7)=-inf;
model.ub(1:7)=inf;
result = gurobi(model);
res=result.x;
   for i=1:N0
           if(~(x(i,:)*res <-10^-3))
                disp("X:"+i);
           end
   end
```

B.

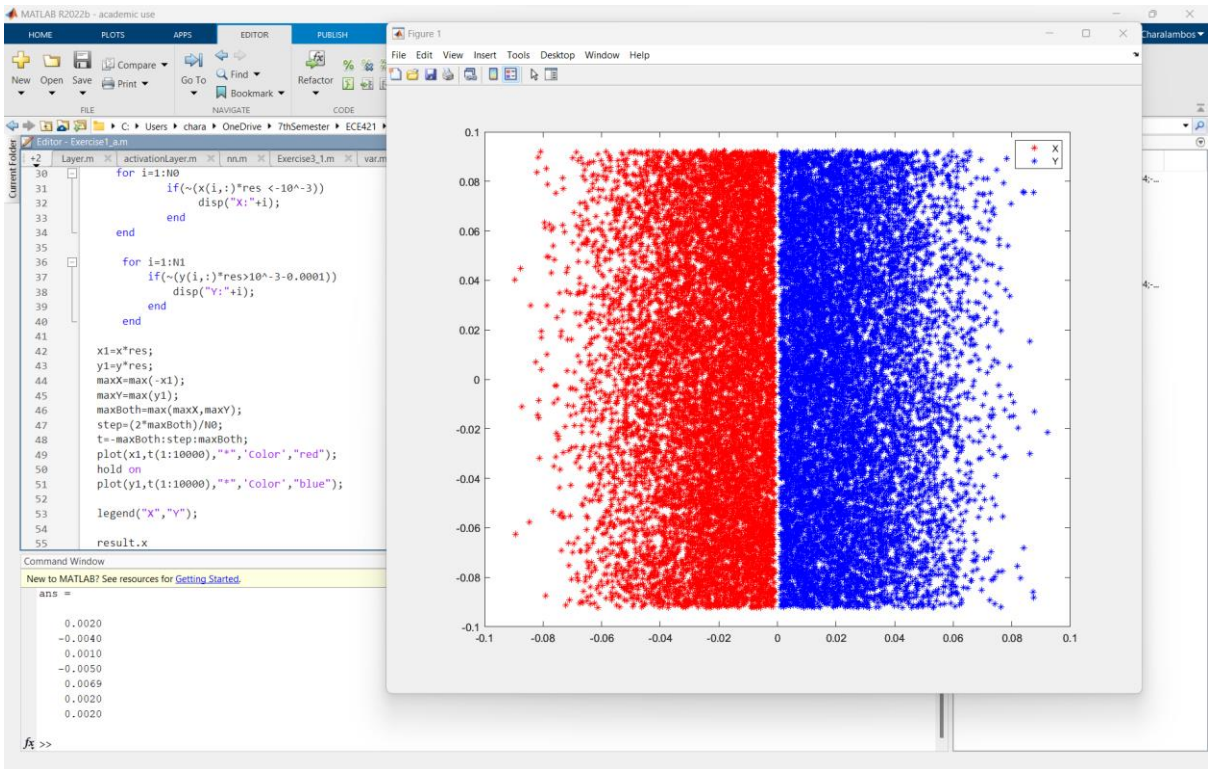
```
res=result.x;
   for i=1:N0
           if(~(x(i,:)*res <-10^-3))
                disp("X:"+i);
           end
   end

    for i=1:N1
        if(~(y(i,:)*res>10^-3-0.0001))
            disp("Y:"+i);
        end
    end

x1=x*res;
y1=y*res;
maxX=max(-x1);
maxY=max(y1);
maxBoth=max(maxX,maxY);
step=(2*maxBoth)/N0;
t=-maxBoth:step:maxBoth;
plot(x1,t(1:10000),"*",'Color',"red");
hold on
plot(y1,t(1:10000),"*",'Color',"blue");

legend("X","Y");

result.x
```



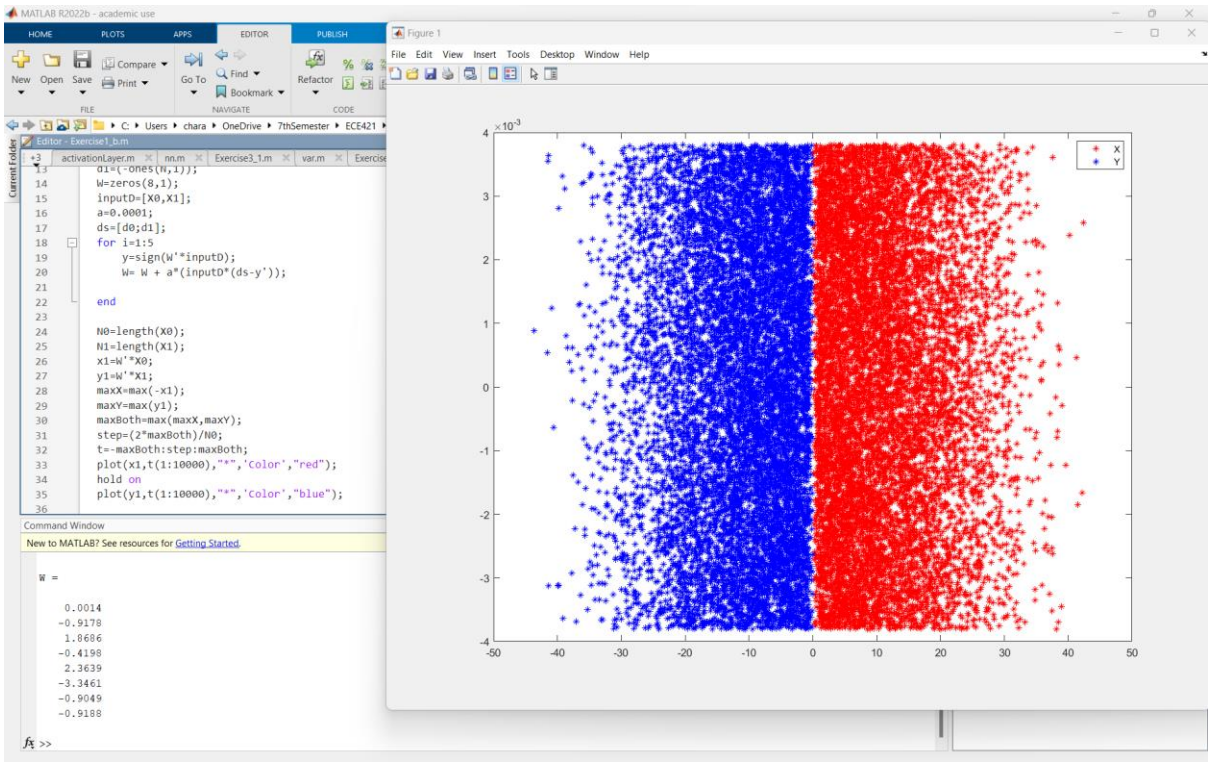
$$W_i = ans_i$$

The ans are the weights that classify our data.

B.

```
close all; clear all; clc;
data=load('.\datasets\dataLinearSeparability.mat');
N=length(data.X0);
X0=zeros(N,length(cell2mat(data.X0(1))));
X1=zeros(N,length(cell2mat(data.X1(1))));
for i=1:N
    X0(i,:)=cell2mat(data.X0(i))';
    X1(i,:)=cell2mat(data.X1(i))';
end
X0=[ones(N,1),X0]';
X1=[ones(N,1),X1]';
d0=(ones(N,1));
d1=(-ones(N,1));
W=zeros(8,1);
inputD=[X0,X1];
a=0.0001;
ds=[d0;d1];
for i=1:5
    y=sign(W'*inputD);
    W= W + a*(inputD*(ds-y'));

end
```



Exercise 2:

The Complete code is split into 3 classes there is the main class which is the NeuralNetwork , the layer class which is a layer that contains the weights of the network, and the activationLayer class which contains the activation functions.

The NeuralNetwork class contain a list of the layers and when we forward propagate then each layer gets the output of the previous layer as an input, except the first one which gets our input.

It also records the error while training and then can plot it.

The samples are handled one by one.

```
classdef NeuralNetwork
    properties
        layers
        layerN
        lr
        guess
        err
        t
        mu
        s
    end

    methods
        function obj=init(obj)
            obj.layers={};
            obj.layerN=1;

        end
        function obj=addLayer(obj,layer)
            obj.layers{obj.layerN}=layer;
            obj.layerN=obj.layerN+1;
        end

        function obj=feedForward(obj,X)
            obj.layers{1}=obj.layers{1}.forward(X);
            for i=2:obj.layerN-1
                obj.layers{i}= obj.layers{i}.forward(obj.layers{i-1}.output);
            end
            obj.guess=obj.layers{obj.layerN-1}.output;
        end

        function obj=train(obj,X,d,lr,training_times)

            for t=1:training_times
                avgErr=0;
                for sample=1:size(X,2)
                    input=X(:,sample)';
                    obj=obj.feedForward(input);
                    desired=d(sample,:);
                    error=obj.guess-desired;
                    error=error';
                    avgErr=avgErr+sum(error.^2)/length(error);
                    for l=obj.layerN-1:-1:1
                        obj.layers{l}=obj.layers{l}.backwords(error,lr);
                        error=obj.layers{l}.input_error;
                    end
                    error=0;
                end
                obj.err{t}=avgErr/size(X,2);
            end
        end
        function plotErr(obj)
            plot(1:length(obj.err),cell2mat(obj.err))
        end
        function obj=getMuNs(obj,data)
                 obj.mu=mean(data);
                 obj.s=std(data);
        end



    end
```

```
classdef Layer
    properties
        W
        input
        output
        input_error
        output_error
        bias
    end

   methods
       function obj=init(obj,input_size,output_size)
            obj.W=(rand(input_size,output_size)*2)-1;
            obj.bias=(rand(1,output_size)*2)-1;
       end
       function obj=forward(obj,x)
                obj.input=x;
               obj.output=x*obj.W ;
               obj.output=obj.output+ obj.bias;
       end

       function obj=backwords(obj,output_error,lr)
            obj.input_error= output_error*obj.W';
            weights_error=obj.input'*output_error;

           obj.W = obj.W - lr.*weights_error;

           obj.bias=obj.bias-lr*output_error;
       end
   end
```

```
classdef activationLayer < Layer

    properties
        activation
        dactivation

    end

    methods
        function obj=init(obj,activation,dactivation)
            obj.activation=activation;
            obj.dactivation=dactivation;
        end
        function obj=forward(obj,X)
            obj.input=X;
            obj.output=obj.activation(X);

        end

        function obj=backwords(obj,output_error,lr)

            obj.input_error=obj.dactivation(obj.input).*output_error;
        end
    end

      methods(Static)

      function ds=dsigmoid(x)
            ds=(1./(exp(-x)+1)).*(1-(1./(exp(-x)+1)));
        end
        function s=sig(x)
            s=(1./(exp(-x)+1));
        end

        function r=tanhP(x)
            r=1-tanh(x).^2;
        end
```

The activation layer inherits the traits of layer but the backwords and forwards functions work differently.

```
nn=nn.addLayer(Layer().init(2,4));
nn=nn.addLayer(activationLayer().init(@activationLayer.sig,@activationLayer.dsigmoid));
nn=nn.addLayer(Layer().init(4,7));
nn=nn.addLayer(activationLayer().init(@tanh,@activationLayer.tanhP));
nn=nn.addLayer(Layer().init(7,5));
nn=nn.addLayer(Layer().init(5,1));
```

The code above build a fully connected Neural Network

That has 4 normal layers and the nodes of each layer are as follows:

Layer1: 4 nodes

Layer2: 7 nodes

Layer3: 5 nodes

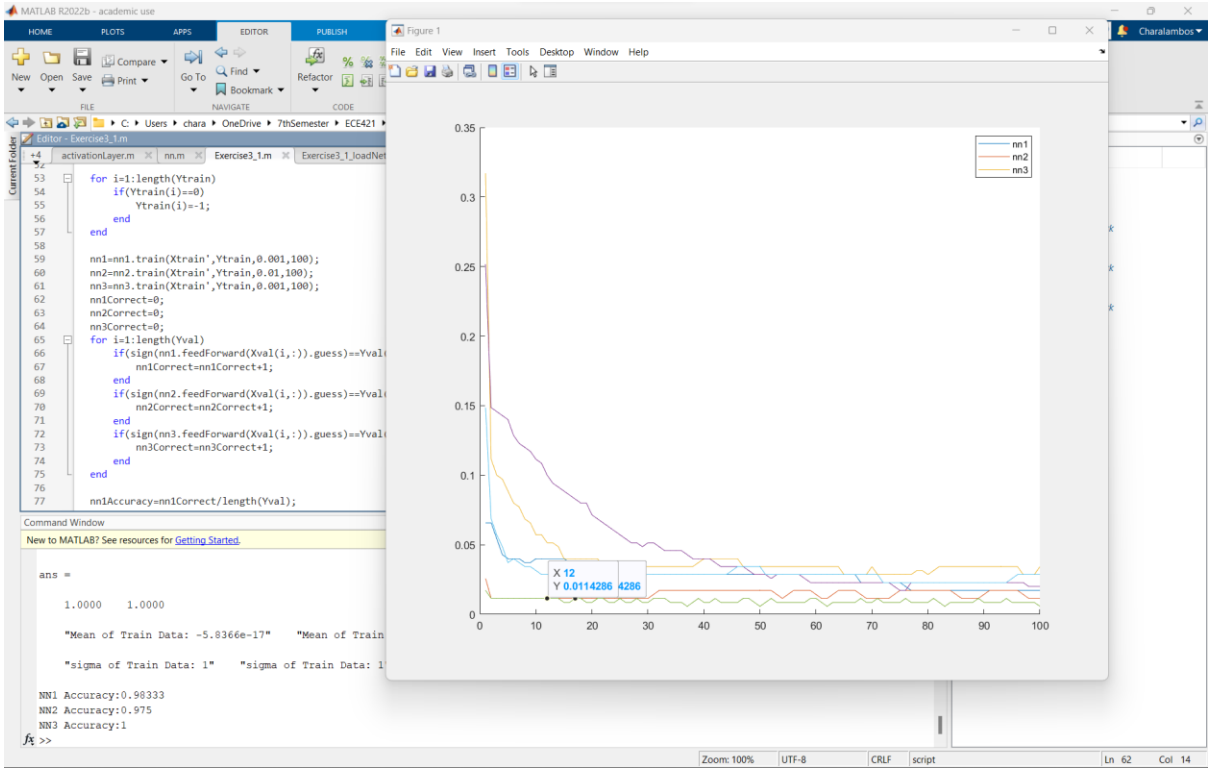
Layer4: 1 node

Exercise 3:

```
nn1=nn1.addLayer(Layer().init(2,10));
nn1=nn1.addLayer(activationLayer().init(@activationLayer.sig,@activationLayer.dsigmoid));
nn1=nn1.addLayer(Layer().init(10,20));
nn1=nn1.addLayer(activationLayer().init(@tanh,@activationLayer.tanhP));
nn1=nn1.addLayer(Layer().init(20,1));
nn1=nn1.addLayer(activationLayer().init(@sign,@activationLayer.signP));

nn2=nn2.addLayer(Layer().init(2,10));
nn2=nn2.addLayer(activationLayer().init(@activationLayer.sig,@activationLayer.dsigmoid));
nn2=nn2.addLayer(Layer().init(10,20));
nn2=nn2.addLayer(activationLayer().init(@tanh,@activationLayer.tanhP));
nn2=nn2.addLayer(Layer().init(20,1));
nn2=nn2.addLayer(activationLayer().init(@sign,@activationLayer.signP));

nn3=nn3.addLayer(Layer().init(2,5));
nn3=nn3.addLayer(activationLayer().init(@activationLayer.sig,@activationLayer.dsigmoid));
nn3=nn3.addLayer(Layer().init(5,10));
nn3=nn3.addLayer(activationLayer().init(@tanh,@activationLayer.tanhP));
nn3=nn3.addLayer(Layer().init(10,1));
nn3=nn3.addLayer(activationLayer().init(@sign,@activationLayer.signP));
```

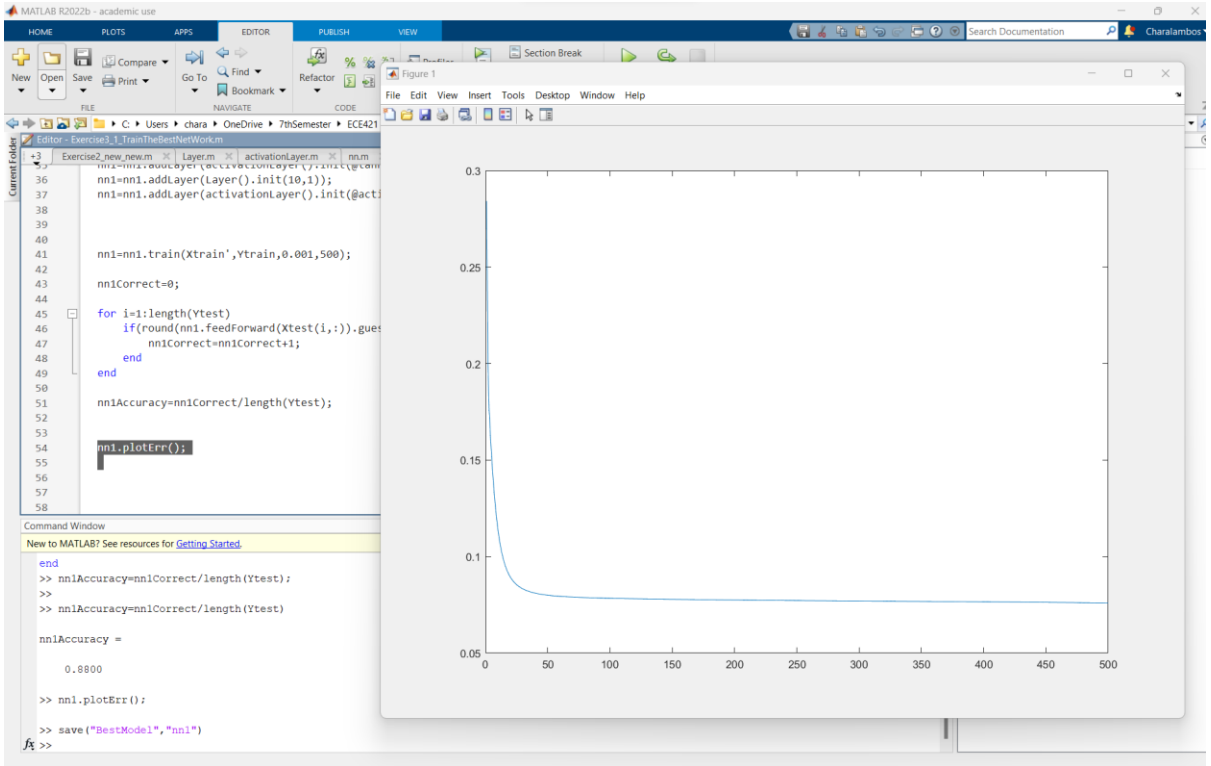


The graph of the loss is like because the samples are inputted one by one if the samples where put together the the loss function would be much smoother.

Best model is nn3 with accuracy=1

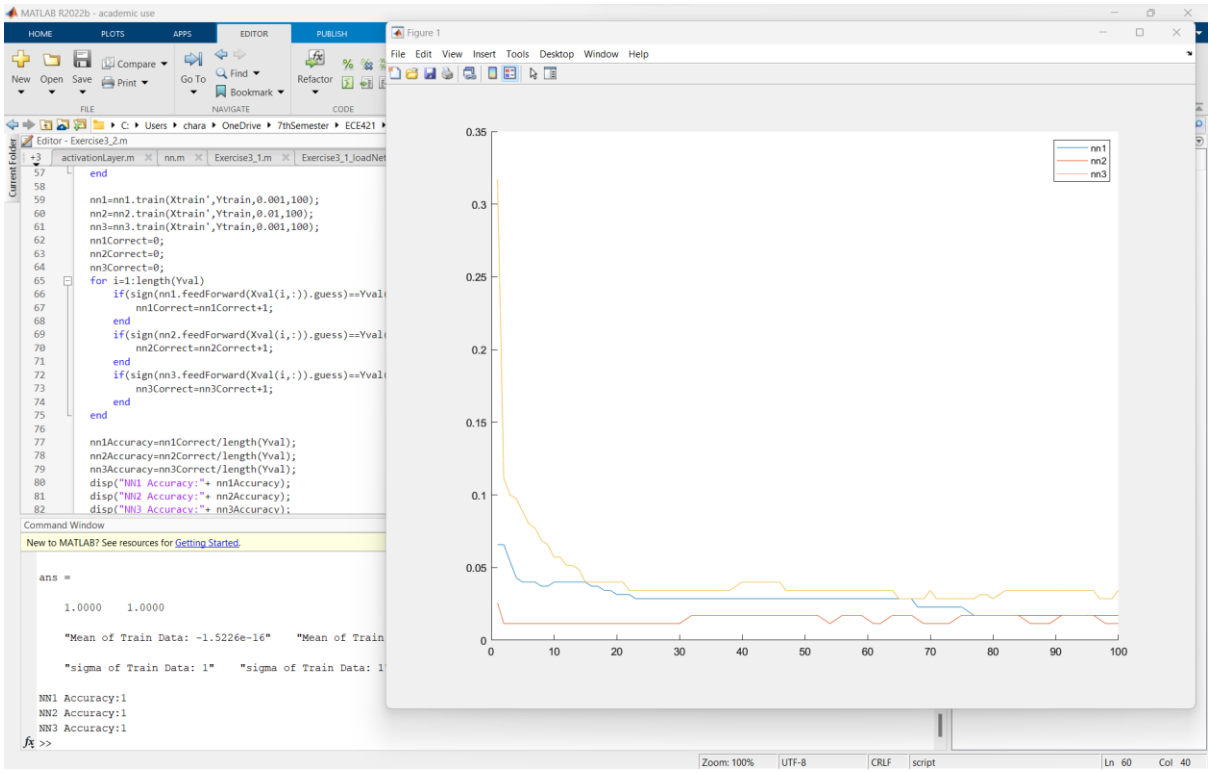
Precision is 1 for all of them since it either class 1 or class 2

I choose nn3 for best accuracy but it need a smaller learning rate and more repetitions.

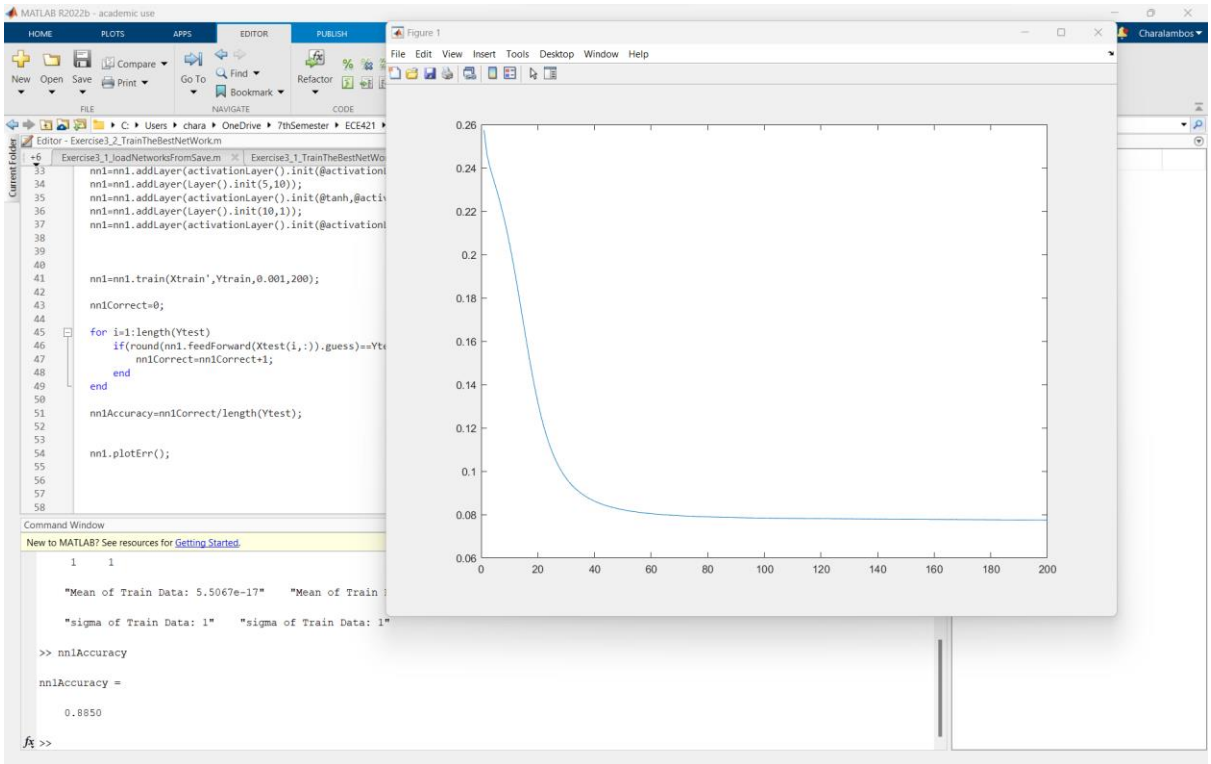


NN3 trained on all the training data and tested on the testing data with accuracy of 0.88. With smaller learning rate and more repetitions it can get better.

2.



All Have accuracy of 100% thus I choose NN2 since it converges faster.



```
>> nn1.layers{1}.W

ans =

   -0.6450   -0.5440    0.6995   -0.3948   -0.6893
    1.2143   -1.6126   -0.1325    0.9323   -0.2135
```

To see the weight of the model