CE793: Deep Learning for Engineers
Assignment 8

Dharanidharan Arumugam

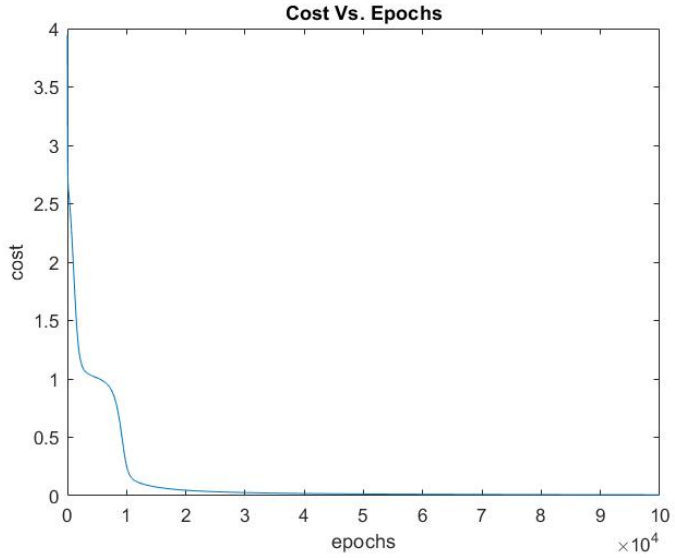
➢ AUTOENCODERS

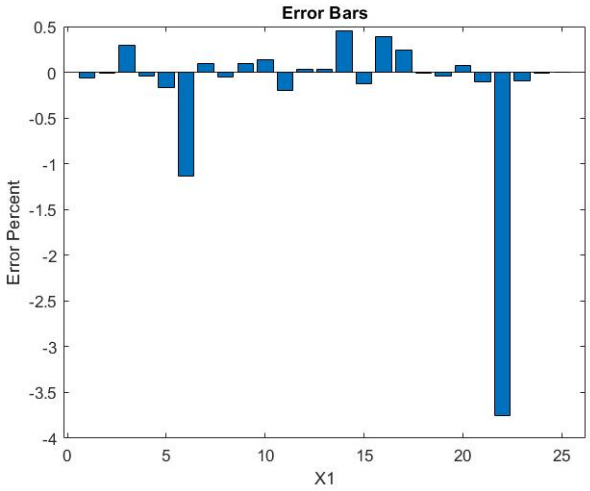
	A. Linear Data Case				B. Non-Linear Data Case			
Instance No.	**h1**		**h2**		**h1**		**h2**	
	h_{11}	h_{12}	h_{21}	h_{22}	h_{11}	h_{12}	h_{21}	h_{22}
1	0.582	0.718	0.737	0.654	0.715	0.461	0.744	0.567
2	0.631	0.737	0.754	0.710	0.759	0.510	0.764	0.640
3	0.288	0.366	0.349	0.280	0.291	0.332	0.355	0.243
4	0.641	0.691	0.712	0.718	0.725	0.545	0.722	0.661
5	0.555	0.363	0.348	0.603	0.377	0.584	0.331	0.542
6	0.244	0.535	0.541	0.244	0.411	0.232	0.535	0.205
7	0.388	0.226	0.218	0.386	0.205	0.497	0.212	0.364
8	0.502	0.399	0.386	0.540	0.384	0.516	0.369	0.468
9	0.717	0.373	0.359	0.778	0.497	0.753	0.380	0.761
10	0.722	0.399	0.388	0.783	0.527	0.750	0.415	0.769
11	0.257	0.615	0.632	0.260	0.481	0.215	0.621	0.215
12	0.665	0.722	0.741	0.744	0.766	0.562	0.752	0.694
13	0.693	0.518	0.524	0.762	0.612	0.678	0.545	0.740
14	0.397	0.746	0.760	0.426	0.655	0.270	0.764	0.330
15	0.657	0.331	0.315	0.719	0.407	0.704	0.310	0.682
16	0.293	0.392	0.377	0.286	0.311	0.325	0.379	0.245
17	0.428	0.436	0.428	0.448	0.385	0.427	0.412	0.372
18	0.628	0.746	0.761	0.708	0.766	0.503	0.772	0.635
19	0.567	0.732	0.749	0.638	0.720	0.438	0.758	0.543
20	0.704	0.450	0.445	0.769	0.559	0.714	0.468	0.751
21	0.529	0.546	0.556	0.582	0.528	0.480	0.546	0.498
22	0.243	0.375	0.358	0.237	0.291	0.290	0.374	0.210
23	0.612	0.678	0.699	0.687	0.696	0.516	0.707	0.617
24	0.644	0.723	0.742	0.723	0.755	0.534	0.752	0.663
25	0.510	0.689	0.709	0.567	0.650	0.399	0.711	0.465

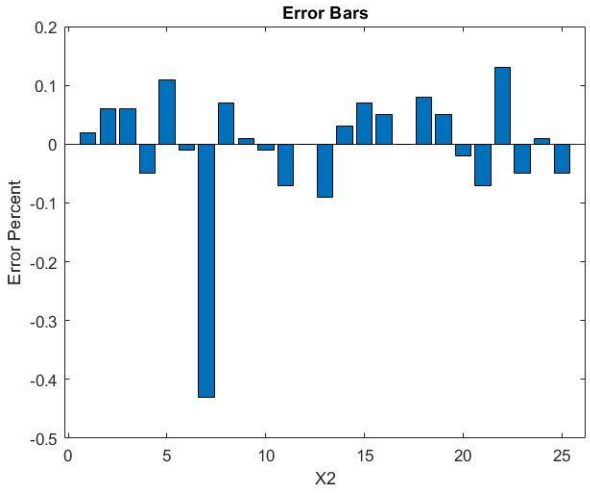
CE793: Deep Learning for Engineers
Assignment 8

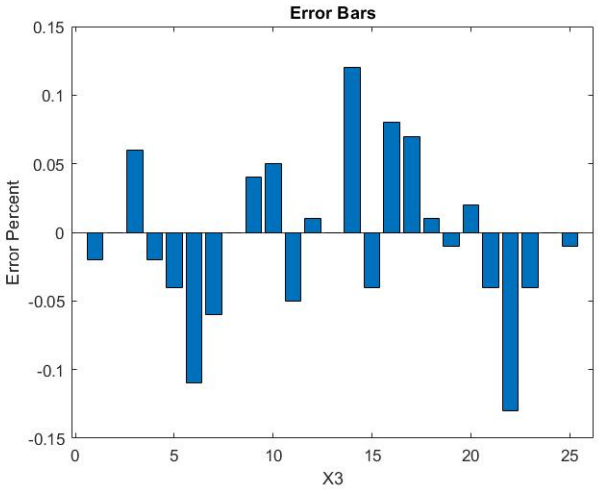
Dharanidharan Arumugam

➢ Plots for Linear Data





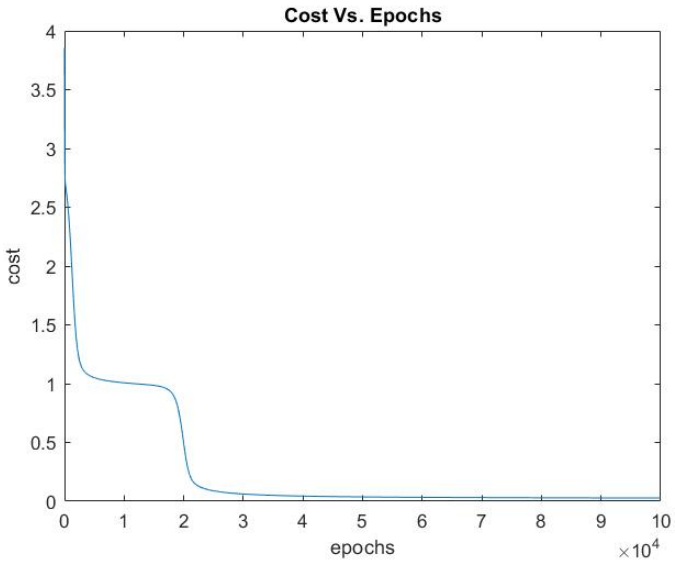


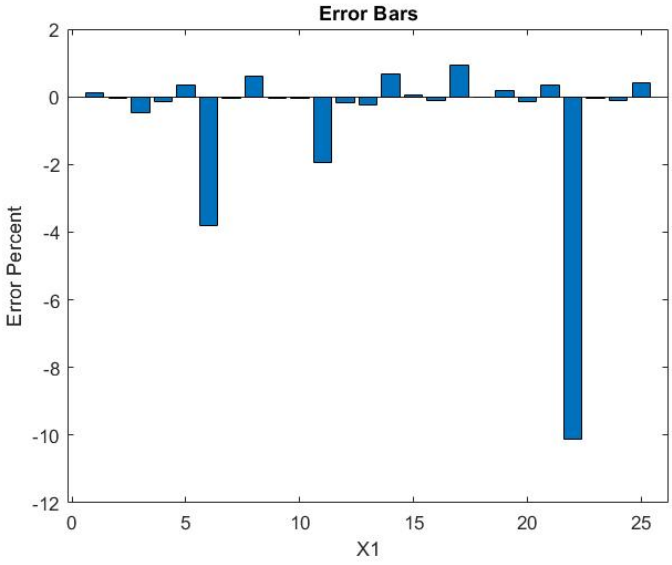


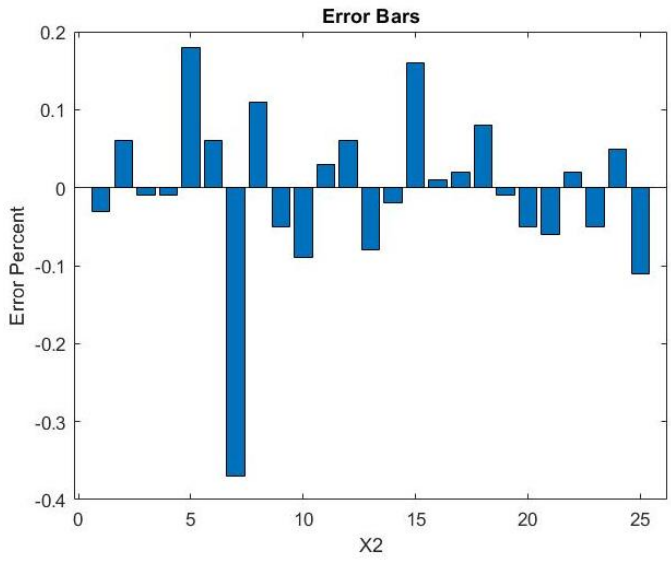
CE793: Deep Learning for Engineers
Assignment 8

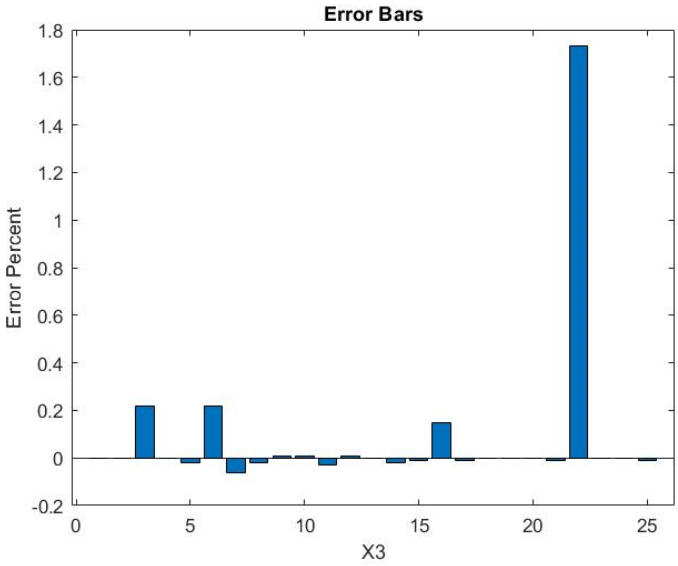
Dharanidharan Arumugam

- **Plots for Non-Linear Data**









CE793: Deep Learning for Engineers
Assignment 8

Dharanidharan Arumugam

❖ MATLAB Code

```
clc;clearvars;close all;
infile         = 'autoencoder_nonlinear.xlsx';

datatable   = readtable(infile);
headers     = datatable.Properties.VariableNames; headers(:,end)=[];
training_data = datatable.Variables; clear datatable;
encode_scheme = [2 2];
decode_scheme = [];

network_architecture.learning_rate  = 0.01;
network_architecture.max_epoch      = 100000;
network_architecture.activation_function = 'sig';

nF      = size(training_data,2);
network_architecture.neurons_scheme = [nF,encode_scheme,decode_scheme,nF];

trainedNeuralNetwork = AutoEncoders(network_architecture,training_data);
cost = trainedNeuralNetwork.cost;
max_epoch = network_architecture.max_epoch;
plot(1:max_epoch,cost);
title('Cost Vs. Epochs');xlabel('epochs');ylabel('cost');

test_data = training_data;
[no_of_instances,no_of_features] = size(test_data);
networkpredictions = predictoutput_autoencoders(trainedNeuralNetwork,test_data);
display(networkpredictions.cost);
error_percent = round((networkpredictions.errors./test_data')*100,2);

k=0;rem=2;
for ftr_num = 1:no_of_features
    figure;
    bar(1:no_of_instances,error_percent(ftr_num,:));
    title_text = strcat('X',string(ftr_num));
    title(title_text);
    title('Error Bars');ylabel('Error Percent');xlabel(title_text);
    k=k+2;rem=no_of_features-2;
end
--------------------------------------------------------------------------------
function trainedNeuralNetwork = AutoEncoders(network_architecture,data)

trainedNeuralNetwork.network_architecture=network_architecture;
learning_rate   = network_architecture.learning_rate;
max_epoch       = network_architecture.max_epoch;
act_fn          = network_architecture.activation_function;
neurons_scheme  = network_architecture.neurons_scheme;

no_of_totalLayers  = length(neurons_scheme);
no_of_synaptics    = no_of_totalLayers-1;

inputs             = (data-mean(data))./std(data);
inputs             = transpose(inputs); clear data;

cost               = zeros(max_epoch,1);
network_synaptics(no_of_synaptics).weights =zeros(5);
```

```
for i = 1:no_of_synaptics
    input_neuron_size  = neurons_scheme(i);
    output_neuron_size = neurons_scheme(i+1);
    network_synaptics(i).weights =rand(output_neuron_size,input_neuron_size);
    network_synaptics(i).biases  =rand(output_neuron_size,1)*0.0001;
end

for epoch = 1:max_epoch

    layers = feedforward(network_synaptics,inputs,act_fn);
    predictions = layers(end).activations;
    errors = inputs-predictions;
    cost(epoch) = cost_function(errors);
    network_synaptics = backpropagate(learning_rate,act_fn,network_synaptics,layers,errors);

end

trainedNeuralNetwork.network_synaptics=network_synaptics;
trainedNeuralNetwork.cost =cost;
end
--------------------------------------------------------------------------
function layers = feedforward(network_synaptics,inputs,act_fn)
    no_of_synaptics = length(network_synaptics);
    layers(1).activations = inputs;
    layers(1).netinputs   = inputs;
    layers(no_of_synaptics+1).activations = 0;
    for synaptic_num = 1:no_of_synaptics
        k=synaptic_num+1;
        weights = network_synaptics(synaptic_num).weights;
        biases  = network_synaptics(synaptic_num).biases;
        layers(k).netinputs = weights*(layers(k-1).activations)+biases;
        if synaptic_num == no_of_synaptics
           layers(k).activations = layers(k).netinputs;
        else
           layers(k).activations = activation_function(layers(k).netinputs,act_fn);
        end
    end
end
--------------------------------------------------------------------------
function
network_synaptics=backpropagate(learning_rate,act_fn,network_synaptics,layers,errors)
    no_of_synaptics = length(network_synaptics);
    m = size(errors,2);
    for s = no_of_synaptics:-1:1
        k = s+1;
        if s==no_of_synaptics
           delta = errors;
        else
           delta =
(transpose(network_synaptics(k).weights)*delta).*derivative_function(layers(k).netinputs,act
_fn);
        end
        network_synaptics(s).weights =
network_synaptics(s).weights+(learning_rate/m)*delta*transpose(layers(s).activations);
        network_synaptics(s).biases  =
network_synaptics(s).biases+(learning_rate/m)*sum(delta,2);
    end
end
--------------------------------------------------------------------------
```

```
function cost = cost_function(errors)
    cost   = sum(errors.*errors,'all')/size(errors,2);
end
```

```
function activations = activation_function(netinputs,ftype)
   switch ftype
       case char('sig')
           activations = 1./(1+exp(-netinputs));
       case char('tan')
           activations = tanh(netinputs);
       case char('lin')
           activations = netinputs;
       otherwise
           disp("No such activation functions are available.")
           disp("Type : for sigmoid fuction - 'sig', tan hyperbolic - 'tan' and  linear
function - lin");
   end
end
```

```
function derivatives = derivative_function(netinputs,ftype)
   switch ftype
       case char('sig')
           activations = activation_function(netinputs,ftype);
           derivatives = activations.*(1-activations);
       case char('tan')
           derivatives = 1-(tanh(netinputs)).^2;
       case char('lin')
           derivatives = ones(size(netinputs));
       otherwise
           disp("No such activation functions are available.")
           disp("Type : for sigmoid fuction - 'sig', tan hyperbolic - 'tan' and  linear
function - lin");
   end
end
```

```
function networkpredictions = predictoutput_autoencoders(trainedNeuralNetwork,test_data)

    act_fn          = trainedNeuralNetwork.network_architecture.activation_function;
    network_synaptics = trainedNeuralNetwork.network_synaptics;

    test_data          = (test_data-mean(test_data))./std(test_data);
    test_data          = transpose(test_data);

    networkpredictions.layers      = feedforward(network_synaptics,test_data,act_fn);
    networkpredictions.targets     = test_data; clear test_data;
    networkpredictions.predicteds  = networkpredictions.layers(end).activations;
    networkpredictions.errors      = networkpredictions.targets-
networkpredictions.predicteds;
    networkpredictions.cost        = cost_function(networkpredictions.errors);

end
```
