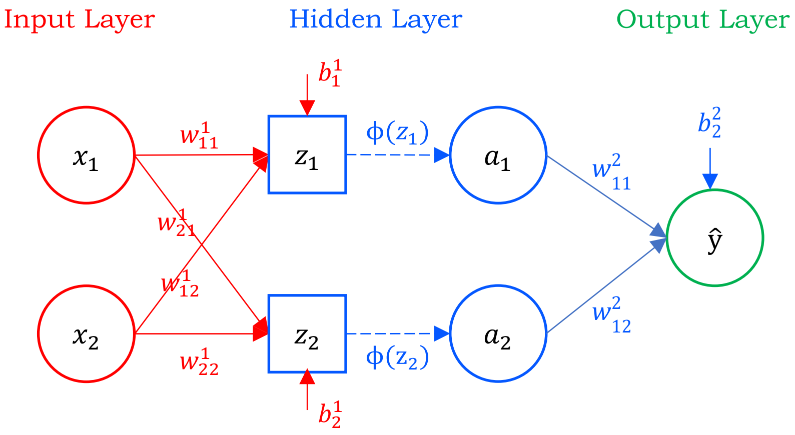
CE793: Deep Learning for Engineers
Assignment 6

Dharanidharan Arumugam

➢ Multi-Layer Perceptron for regression:

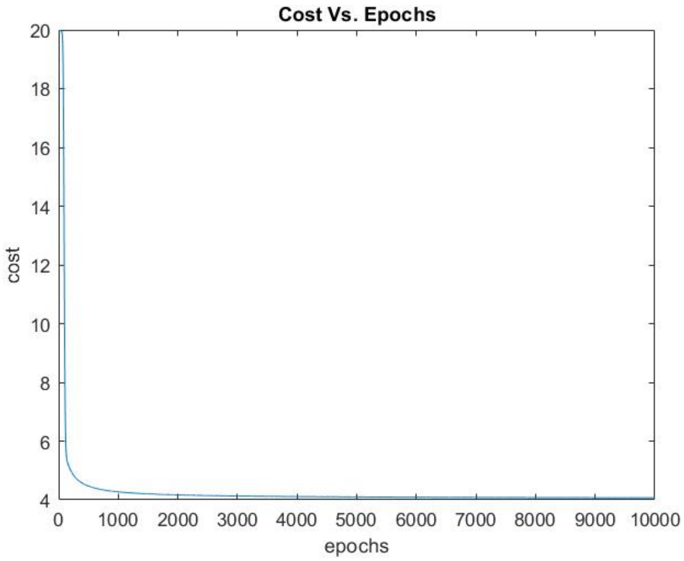
Network Architecture

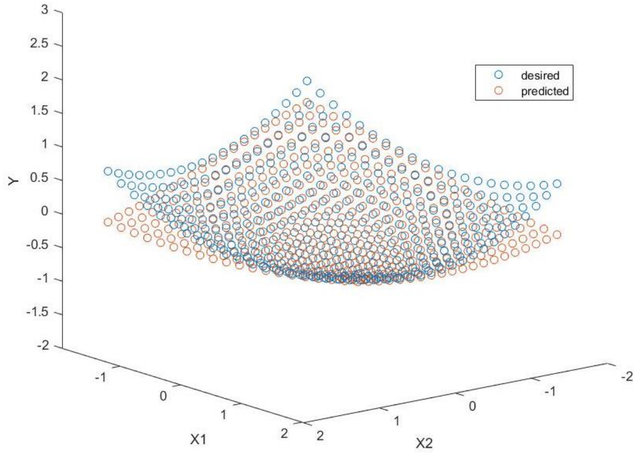


Sigmoide Activation function, $\phi(z) = (1 + e^{-z})^{-1}$

- Total layers: 3 (Input Layer, Hidden Layer, and Output Layer)
- Number of Neurons: Input Layer -2, Hidden Layer-2, and Output Layer-2
- Activation function: Sigmoid

➢ Plots:





CE793: Deep Learning for Engineers
Assignment 6

Dharanidharan Arumugam

❖ MATLAB Code

```
clc;clearvars;close all;
infile         = 'data_data.xlsx';
datatable   = readtable(infile);
headers     = datatable.Properties.VariableNames; headers(:,end)=[];
training_data = datatable.Variables; clear datatable;
network_architecture.learning_rate  = 10^-3;
network_architecture.max_epoch      = 10000;
network_architecture.neurons_scheme = [2,2,1]; %no. of neurons in[input hidden output]layer
network_architecture.activation_function = 'sig';
trainedNeuralNetwork = MultiLayerPerceptron(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'); figure;
test_data = training_data;
no_of_instances = size(test_data,1);
networkpredictions = predictoutput_mlp(trainedNeuralNetwork,test_data);
display(networkpredictions.cost);
scatter(networkpredictions.predicteds,networkpredictions.targets);
title('predictions Vs. targets');xlabel('predicteds');ylabel('targets'); figure;
bar(1:no_of_instances,networkpredictions.errors);
title('Error Bars');xlabel('instances');ylabel('errors'); figure;
test_data = (test_data-mean(test_data))./std(test_data);
hold on
view(3);
scatter3(test_data(:,1),test_data(:,2),networkpredictions.targets');
scatter3(test_data(:,1),test_data(:,2),networkpredictions.predicteds');
hold off
```

```
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);
    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*delta...
                                                            *transpose(layers(s).activations);
        network_synaptics(s).biases  = network_synaptics(s).biases+learning_rate*sum(delta,2);
    end
end
```

CE793: Deep Learning for Engineers
Assignment 6

Dharanidharan Arumugam

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

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

    no_of_features     = neurons_scheme(1);
    no_of_instances    = size(test_data,1);

    test_data          = (test_data-mean(test_data))./std(test_data);
    test_data          = transpose(test_data);
    inputs             = test_data(1:no_of_features,:);
    targets            = test_data(no_of_features+1:end,:)';
    %mean_targets       = mean(targets);std_targets        = std(targets);
    networkpredictions.targets     = test_data(no_of_features+1:end,:); clear test_data

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

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 cost = cost_function(errors)
    cost   = sqrt(errors*transpose(errors));
end
--------------------------------------------------------------------------------------------------
```