* **Multi-Layer Perceptron for regression:**

Input Layer

Hidden Layer

Output Layer

**Network Architecture**

* 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:**

|  |  |
| --- | --- |
|  |  |

* 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

-------------------------------------------------------------------------

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

-------------------------------------------------------------------------