* **AUTOENCODERS**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A. Linear Data Case | | | | B. Non-Linear Data Case | | | |
| Instance No. | **h1** | | **h2** | | **h1** | | **h2** | |
| h11 | h12 | h21 | h22 | h11 | h12 | h21 | h22 |
| 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 |

* **Plots for Linear Data**

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

* **Plots for Non-Linear Data**

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

* 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

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