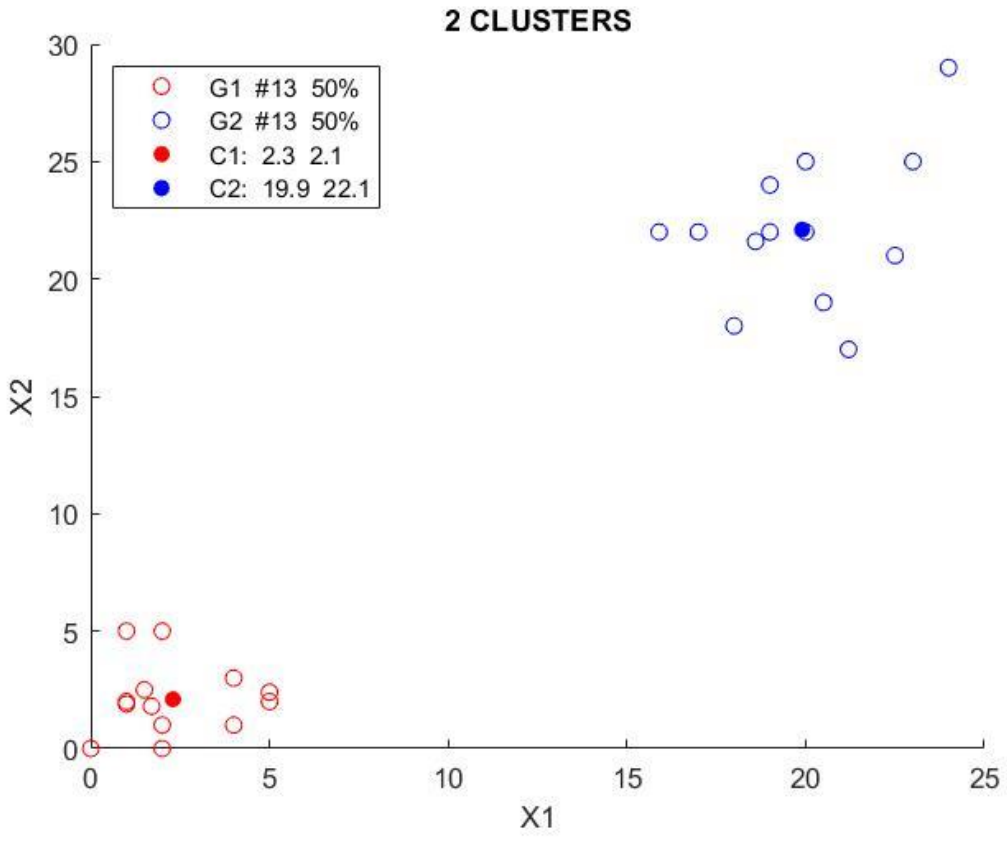
CE793: Deep Learning for Engineers
Assignment 7

Dharanidharan Arumugam

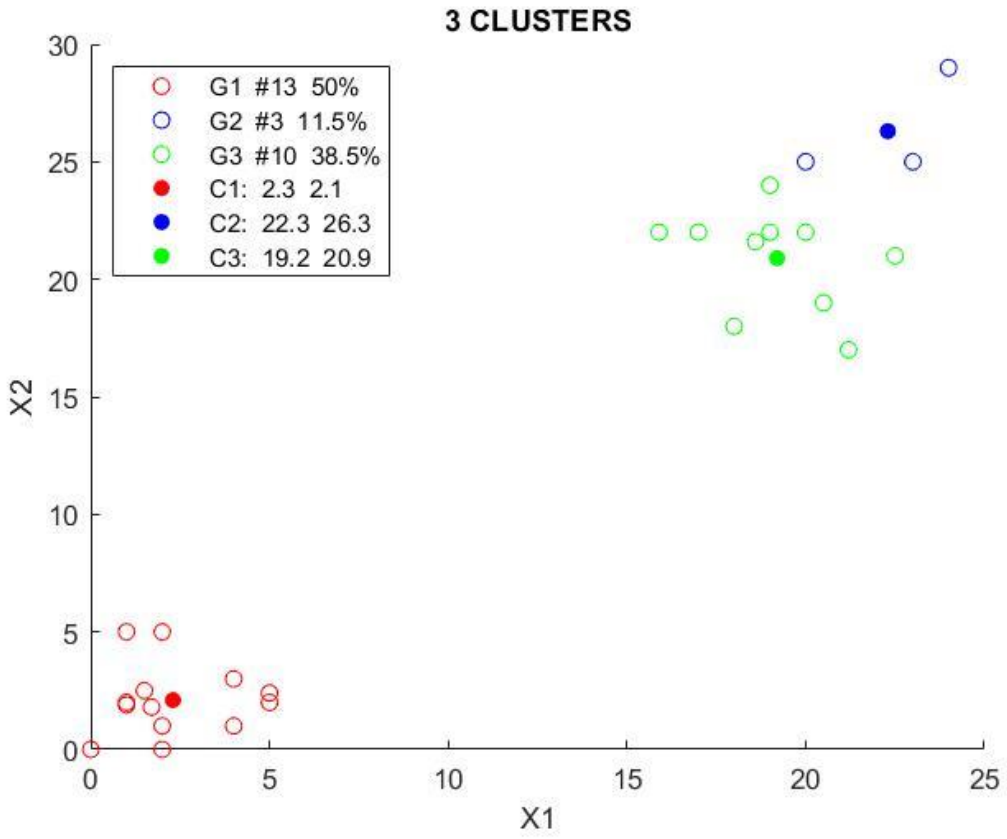
- **Problem 1: KNN Clustering Analysis - Part 1**

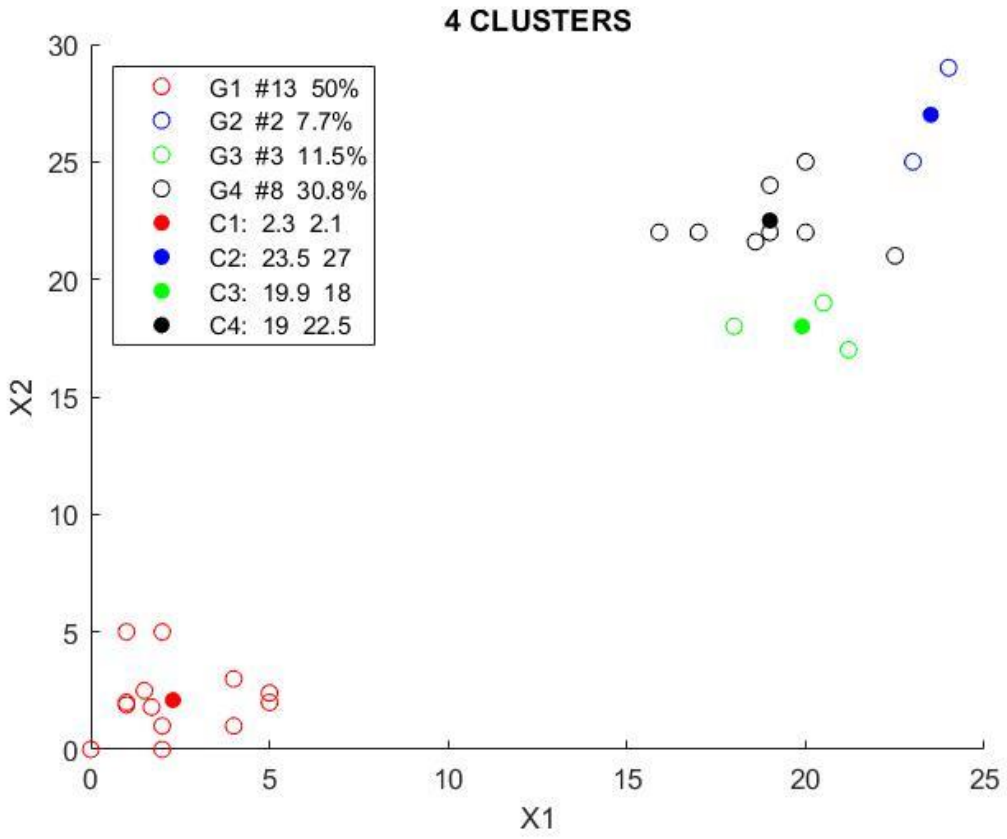
CLUSTERS	CENTROIDS OF CLUSTER GROUP:									
	G1		**G2**		**G3**		**G4**		**G5**	
	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
2	2.3	2.1	19.9	22.1	*	*	*	*	*	*
3	2.3	2.1	22.3	26.3	19.2	20.9	*	*	*	*
4	2.3	2.1	23.5	27.0	19.9	18.0	19.0	22.5	*	*
5	19	22.5	1.3	1.3	23.5	27.0	3.5	3.1	19.9	18.0

- **Plots:**



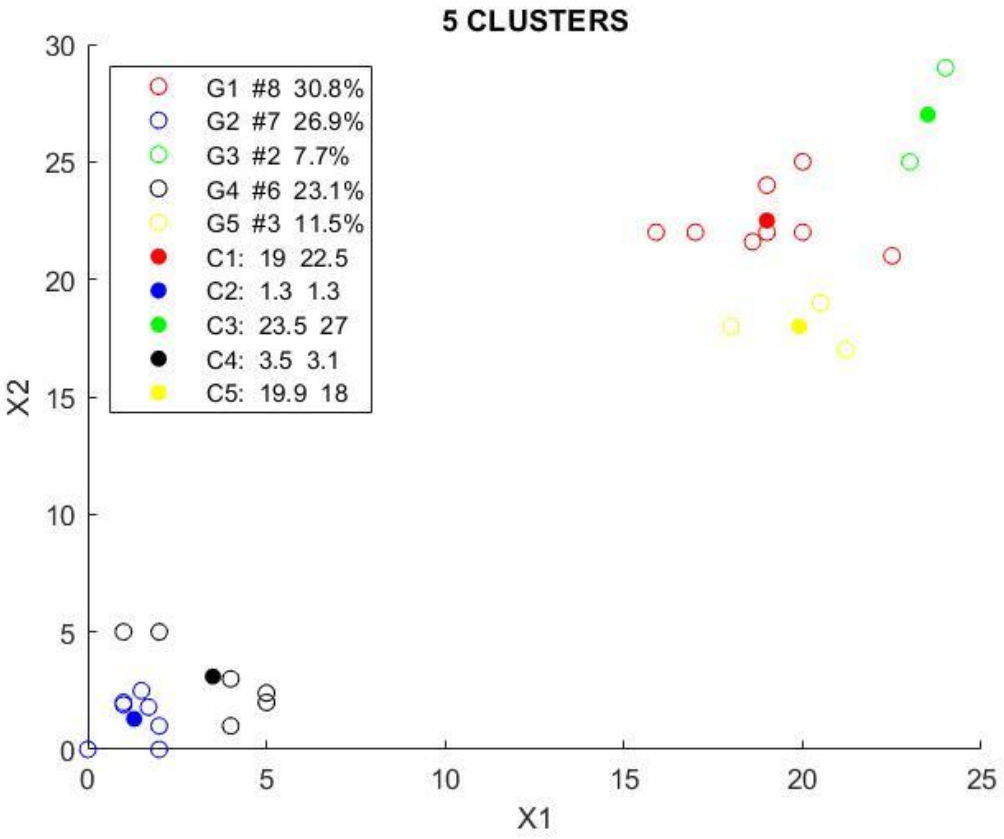
Dharanidharan Arumugam





CE793: Deep Learning for Engineers
Assignment 7

Dharanidharan Arumugam

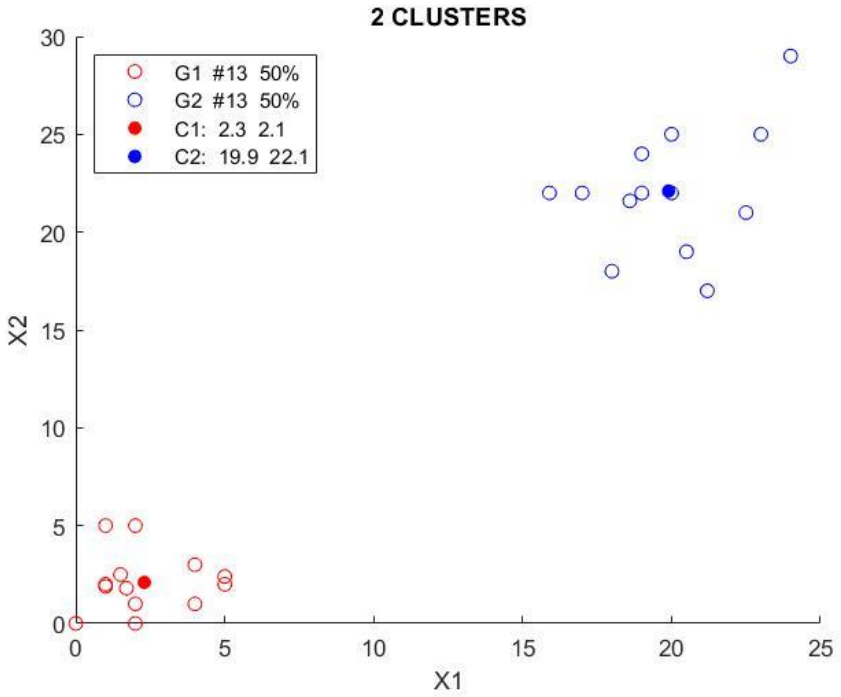


- **Problem 1: KNN Clustering Analysis - Part 2**

Clusters with Initial Centroids	Cluster Centroids			
	G1		**G2**	
	C_1	C_2	C_1	C_2
Using KNN++	2.3	2.1	19.9	22.1
With (5,5) & (8,8)	2.3	2.1	19.9	22.1

Dharanidharan Arumugam

- **Plots:**



❖ MATLAB Code

```
clc;clearvars;close all;clear figure;
infile          = 'clust_data.xlsx';

%% Reading data
datatable    = readtable(infile);
headers      = datatable.Properties.VariableNames; headers(:,end)=[];
data         = datatable.Variables; clear datatable;

%% Calling Clustering function
no_of_clusters = 2;
intial_centroids=[5,5;8,8];
%clusters = KNNplus_clustering(data,no_of_clusters);
clusters  = KNNplus_clustering(data,no_of_clusters,intial_centroids);

%% Plot the final clusters
color = ['r','b','g','k','y'];
lgd_txt=strings(1,no_of_clusters*2);
plots=gobjects(no_of_clusters*2,1);
centroids=round(clusters.centroids,1);
K=no_of_clusters;
for i = 1:K
    Xc= centroids(i,1);
    Yc= centroids(i,2);
    X = data(clusters.cluster_nos==i,1);
    Y = data(clusters.cluster_nos==i,2);
    no= strcat('#',string(clusters.numbers(i)));
    pe= strcat(string(clusters.percentage(i)),'%');
    lgd_txt(i)=strcat('G',string(i),{'  '},no,{'  '},pe);
```

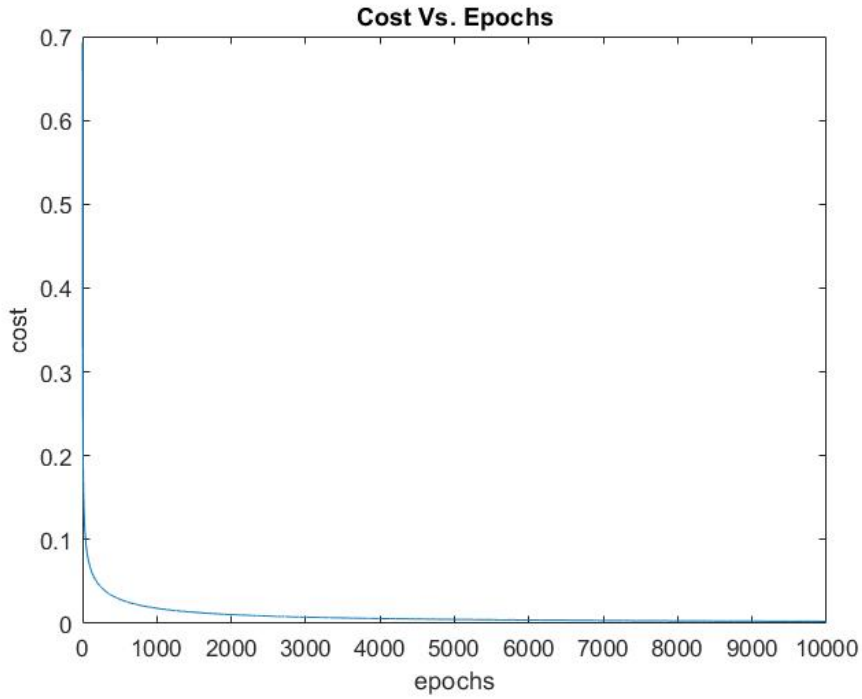
```
        tx= string(centroids(i,1));
        ty= string(centroids(i,2));
        lgd_txt(i+K)=strcat('C',string(i),':',{'  '},tx,{'  '},ty);
        hold on
        plots(i)=scatter(X,Y,color(i));
        plots(i+K)=scatter(Xc,Yc,color(i),'filled');
    end
    title_text = strcat(string(no_of_clusters),{' '},'CLUSTERS');
    xlabel('X1');ylabel('X2');title(title_text);
    legend(plots,lgd_txt,'Location','northwest');
    hold off
```

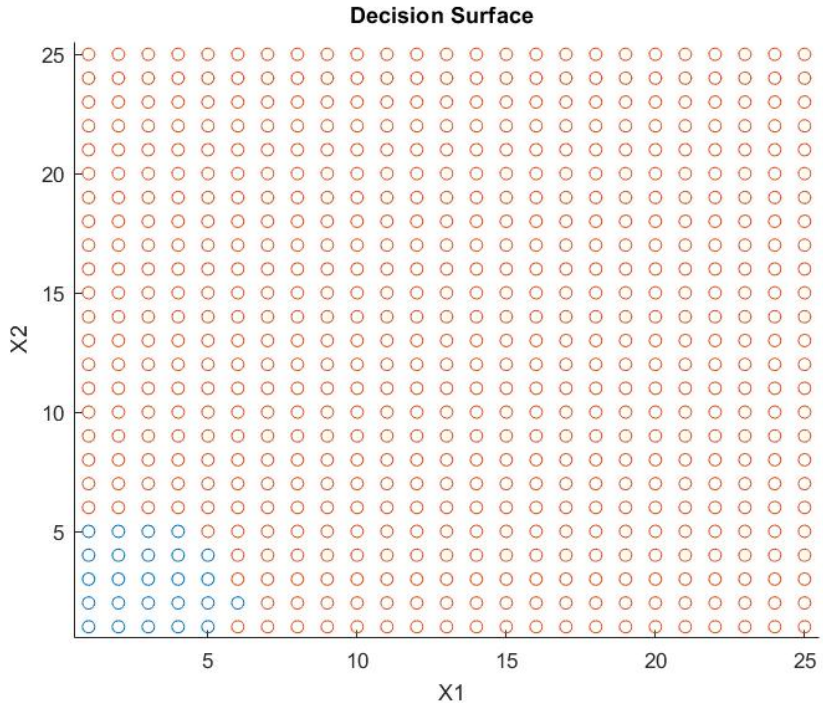
```
    function clusters = KNNplus_clustering(data,no_of_clusters,varargin)

        %data       = (data-mean(data))./std(data);
        [nI,nF]     = size(data);

        %% Parameters defination
        K           = no_of_clusters;
        if K==1
               clusters.cluster_nos = ones(nI,1);
               clusters.numbers     = nI;
               clusters.percentage  = 100;
               clusters.centroids   = mean(data);
               diff                 = data-clusters.centroids;
               clusters.variance    = 0.5*sum(diff.*diff,'all')/nI;
               return;
        end
        dist_old    = 10^6;
        termination_threshold = 10^-8;

        %% Parameters Initiazation
        cluster_nos = zeros(nI,1);
        numbers     = zeros(K,1);
        percentage  = zeros(K,1);
        threshold   = 100;
        iterations  = 0;

        %% Initial centroids
        if isempty(varargin) %for default, use KNN++ approach
            centroids       = zeros(K,nF);
            centroids(1,:)       = data(randi(nI),:);
            dist=zeros(nI,1);
            for i=2:K
                for m = 1:nI
                   instance = data(m,:);
                   delta    = centroids(1:i-1,:)-instance;
                   euclid   = sqrt(sum(delta.*delta,2));
                   dist(m)  = min(euclid);
                end
                [~,n]=max(dist);
                centroids(i,:)=data(n,:);
            end
        elseif length(varargin)==1
            [nC,nDc]  = size(varargin{1});
            if or(nC~=K,nDc~=nF)
                disp('clustering unsuccessful:centroid dimensions do not match');
                return;
            else
```

```
            centroids = varargin{1};
        end
    else
        disp('clustering unsuccessful:too many inputs');
        return;
    end
    %% Clustering
    while threshold > termination_threshold
        % Finding Euclidean distance of data points
        dist_new=0;
        if iterations~=0
           for j=1:K
            instances     = data(cluster_nos==j,:);
            centroids(j,:)= mean(instances,1);
           end
        end
        for i=1:nI
           instance = data(i,:);
           delta    = (centroids-instance);
           euclid   = sqrt(sum(delta.*delta,2));
           [min_dist,cluster_nos(i)]=min(euclid);
           dist_new = dist_new+min_dist;
        end
        % Finding new centroids
        threshold = (abs(dist_old-dist_new)/dist_old);
        dist_old  = dist_new;
        iterations = iterations+1;
    end
    disp(iterations);
    clusters.cluster_nos = cluster_nos;
    clusters.numbers     = zeros(K,1);
    clusters.percentage  = zeros(K,1);
    clusters.centroids   = centroids;
    clusters.variance    = zeros(K,1);
    for j=1:K
        instances     = data(cluster_nos==j,:);
        delta         = instances-centroids(j,:);
        clusters.numbers(j)  = size(instances,1);
        clusters.percentage(j)= round(clusters.numbers(j)/nI*100,1);
        clusters.variance(j) = 0.5*sum(delta.*delta,'all')/clusters.numbers(j);
    end
end
```

Dharanidharan Arumugam

- **Problem 2: Radial Basis Neural Network**





Dharanidharan Arumugam

❖ MATLAB Code

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

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

network_architecture.learning_rate  = 5;
network_architecture.max_epoch      = 10000;
network_architecture.receptors      = [1,1]; %no. of receptors for each class

trainedNeuralNetwork = RadialBasisClassifier(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');

no_of_class = length(network_architecture.receptors);
data=zeros(25*25,2);k=1;
for i=1:25
    for j=1:25
        data(k,:)= [i j];
        k=k+1;
    end
end
predicteds=predictoutput_rbfnn(trainedNeuralNetwork,data);
figure;
for k=1:no_of_class
    class_instances=data(predicteds==k,:);
    scatter(class_instances(:,1),class_instances(:,2));
    hold on
end
```

```
function trainedNeuralNetwork = RadialBasisClassifier(network_architecture,data)

trainedNeuralNetwork.network_architecture=network_architecture;
learning_rate   = network_architecture.learning_rate;
max_epoch       = network_architecture.max_epoch;
receptors       = network_architecture.receptors;

no_of_receptors    = sum(receptors);
no_of_class        = length(receptors);

%%data               = transpose(data);
inputs             = data(:,1:end-1);
targets            = data(:,end); clear data;
[no_of_instances,no_of_features] = size(inputs);

cost               = zeros(max_epoch,1);
weights            = zeros(no_of_class,no_of_receptors+1);

n=1;
centroids=zeros(no_of_receptors,no_of_features);
variance=zeros(no_of_receptors,1);
for class_num = 1:no_of_class
    class_data      = inputs(targets==class_num,:);
    no_of_clusters  = receptors(class_num);
    m               = no_of_clusters+n-1;
```

```
        clusters        = KNNplus_clustering(class_data,no_of_clusters);
        centroids(n:m,:)= clusters.centroids;
        variance(n:m,:) = clusters.variance;
        n               = m+1;
    end

    phi=zeros(no_of_instances,no_of_receptors);
    for r=1:no_of_receptors
        diff=inputs-centroids(r,:);
        phi(:,r)=exp(-0.5*sum(diff.*diff,2)/variance(r));
    end
    phi         = transpose([ones(no_of_instances,1),phi]);
    indices     = sub2ind([no_of_class,no_of_instances],targets,(1:no_of_instances)');

    for epoch = 1:max_epoch
        netinputs   = weights*phi;
        predicteds  = softmax_activation(netinputs);
        cost(epoch,1)= sum(-log(predicteds(indices)))/no_of_instances; %cross entropy

        delta       = predicteds; clear predicted;
        delta(indices)=delta(indices)-1;
        gradient    = delta*transpose(phi)/no_of_instances;
        weights     = weights - learning_rate*gradient;
    end

    trainedNeuralNetwork.weights=weights;
    trainedNeuralNetwork.cost =cost;
    trainedNeuralNetwork.receptors.centroids=centroids;
    trainedNeuralNetwork.receptors.variance=variance;

    end
-------------------------------------------------------------------------
    function activations = softmax_activation(netinputs)
        activations = exp(netinputs);
        activations = activations./sum(activations);
    end
-------------------------------------------------------------------------
    function predicteds = predictoutput_rbfnn(trainedNeuralNetwork,inputs)

        receptors       = trainedNeuralNetwork.network_architecture.receptors;
        weights         = trainedNeuralNetwork.weights;
        centroids       = trainedNeuralNetwork.receptors.centroids;
        variance        = trainedNeuralNetwork.receptors.variance;

        no_of_receptors    = sum(receptors);
        no_of_instances    = size(inputs,1);

        phi=zeros(no_of_instances,no_of_receptors);
        for r=1:no_of_receptors
            diff=inputs-centroids(r,:);
            phi(:,r)=exp(-0.5*sum(diff.*diff,2)/variance(r));
        end
        phi         = transpose([ones(no_of_instances,1),phi]);
        netinputs   = weights*phi;
        activations = transpose(softmax_activation(netinputs));
        [~,predicteds]  = max(activations,[],2);

    end
-------------------------------------------------------------------------
```