1 Computer Problem Radial Basis Function Learning

a) Treat this assignment as a function approximation problem, NOT as a classification problem. Using your own code and data structures, implement a simulation of an RBF neural network and learning algorithms in a program. Set up a two input, one output network with one hidden layer network topology. Implement your choice of two of the three learning algorithms described in the text. Use the same training and testing data from HW 4 (Back prop).

For my assignment I chose to implement the fixed centers with instant weight learning, and self organizing centers with LMS weight learning. Both were done in Octave for simplicity. See Appendix for code listings.

b) Train and test a set of RBF networks with the data. For each network, produce a generalization plot (see Haykin Sec 4.14).

Run numerical experiments with the RBF architecture, varying the network by varying the number of hidden nodes in the hidden layer three times, ranging anywhere from 2 to 50. This implies you will be producing three (3) generalization plots for this part of the assignment. Explain your stopping criterion, that is, the epoch number where you decide when to stop training. Discuss the results, and compare to the MLP/BP assignment.

The stopping criteria for my first approach was obvious as it only requires a single pass of the data to instantly learn the weights for the linear output neuron. The stopping criterion for my second approach was obtained by examining the data from a 100 epoch run. I chose 100 epochs for my testing because the network took too long to run per-epoch once I reached 20 nodes. After observing the graphs that are attached I saw that the system was converging fairly quickly (typically within 10 epochs) and as such knew that I could use a simple test of error change to stop the system. In my final code (see Appendix) I have implemented a criterion that the network stops training when the absolute change in error from epoch to epoch is less that 0.001.
In general I find the results satisfactory. I was very interested in how the fixed centers approach would perform, and was very happy to see that it validated my desires by achieving 0 error in training when I raised the number of centers to 800 (one for every datum). I was impressed with the speed that the fixed centers approach achieved in comparison to the self-organizing centers approach. Even once I limited the self-organizing centers to less than 10 epochs it ran very slow in comparison. This could be due to slow coding on my part however.

Comparing these approaches to the back-propagation approach, they perform better in general at a perceived loss of wall clock time. This may be a slightly unfair comparison since my BP code was written in C and this was written in Octave. The code for the RBF was painfully simple to implement in comparison to BP in my opinion.

<table>
<thead>
<tr>
<th>Hidden Layer Size</th>
<th>Training RMS error</th>
<th>Testing RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>0.0000</td>
<td>64.3405</td>
</tr>
<tr>
<td>400</td>
<td>9.4348</td>
<td>12.5318</td>
</tr>
<tr>
<td>4</td>
<td>0.83066</td>
<td>0.83064</td>
</tr>
</tbody>
</table>

Table 1: Network generalization by hidden layer size

Figure 1: Generalization plot for a 4 hidden node network

c) Repeat part (b) with two (2) other training data orderings. Discuss the effects on convergence rates and the number of epochs required to reach a common terminal error. Do not include plots of this.

Because of the order sensitivity of the two RBF approaches I took, the data was always re-randomized before/between epochs. As such, presenting different orderings of data to the system has no result on the performance. In the first approach the data was randomized to allow for random selection of centers from the data, in the second it was randomized to prevent malicious center migration.
Figure 2: Generalization plot for a 20 hidden node network

Figure 3: Generalization plot for a 50 hidden node network
A  Fixed Centers

function out = ass5()
    load centers2;
    load train1;
    load test1;

    n = 4;
    data = centers2;
    t = zeros(n,2);
    x = zeros(n,2);
    d = zeros(n,1);
    Phi = zeros(n);
    dmax = zeros(n);

    for p=1:n
        t(p,:) = [data(p,3),data(p,4)];
        x(p,:) = [data(p,3),data(p,4)];
        d(p) = data(p);
    end

    sigma = 0.001;

    for j=1:n
        for i=1:n
            tr = x(j,:) - t(i,:);
            dmax(j,i) = sqrt((tr(1)*tr(1))+(tr(2)*tr(2)));
            r = dmax(j,i);
        end
    end

    dm1 = max(max(dmax));

    sigma = dm1/sqrt(2*n);

    for j=1:n
        for i=1:n
            r = dmax(j,i);
            Phi(j,i) = exp(- ( (r*r)/(2*sigma*sigma)));
        end
    end
w = pinv(Phi)*d;

x = zeros(800,2); d = zeros(800,1);

data = train1;

for p=1:800
    x(p,:) = [data(p,3),data(p,4)];
    d(p) = data(p);
end

RMS = 0;

for j=1:800
    F = 0;
    for i=1:n
        tr = x(j,:) - t(i,:);
        r = sqrt((tr(1)*tr(1))+(tr(2)*tr(2)));
        P_temp = exp(- ( (r*r)/(2*sigma*sigma)));
        F += w(i) * P_temp;
    end
    er1 = (d(j) - F);
    err = er1 * er1;
    RMS += err;
end

data = test1;
for p=1:n
    x(p,:) = [data(p,3),data(p,4)];
    d(p) = data(p);
end

TRMS = 0;

for j=1:800
    F = 0;
    for i=1:n
        tr = x(j,:) - t(i,:);
        r = sqrt((tr(1)*tr(1))+(tr(2)*tr(2)));
        P_temp = exp(- ( (r*r)/(2*sigma*sigma)));
        F += w(i) * P_temp;
    end
er1 = (d(j) - F);
err = er1 * er1;
TRMS += err;
end

out = [sqrt(RMS/800);sqrt(TRMS/800)];
endfunction

B Self Organizing Centers

function out = ass5()
load centers2;
load train1;
load test1;

% number of centers
n = 4;
% learning parameter for center movement
eta = 0.1;
% learning parameter for LMS
etaL = 0.001;

% epoch count
eCount = 100;
% epoch RMS&TRMS measures [training testing]
epochRMS = zeros(eCount,2);

% the weight vector
w = rand(n+1,1);
% sigma for gaussian
sigma = 0.001;
% Linear Neuron output value
F = 0;

data = train1;
dataT = test1;
t = zeros(n,2);
x = zeros(n,2);
d = zeros(n,1);
rbf_out = zeros(n+1,1);
Phi = zeros(n);
dmax = zeros(n);

% Number of samples to draw
TrainN = 800;

% Generate scrambled index of 'data'
TrainRows = randperm(size(data,1))';

% Only keep so many of them
TrainRows = TrainRows(1:TrainN);

% Extract rows of interest
TrainData = data(TrainRows,:);

for p=1:n
    t(p,:) = [TrainData(p,3),TrainData(p,4)];
end

for p=1:800
    x(p,:) = [TrainData(p,3),TrainData(p,4)];
    d(p) = TrainData(p);
end

for p=1:800
    xT(p,:) = [dataT(p,3),dataT(p,4)];
    dT(p) = dataT(p);
end

closest_center = zeros(n,1);

deltaerror = 1.0;
olderror = 10.0;
epoch = 0;
% now we test then move the centers and weights until converged
% for epoch=1:eCount
while (deltaerror > 0.001)
    epoch++;
    % error value
    RMS = 0;
TRMS = 0;

for j=1:800
  F = 0;
  for i=1:n
    tr = xT(j,:) - t(i,:);
    r = sqrt((tr(1)*tr(1))+(tr(2)*tr(2)));
    P_temp = exp(- ( (r*r)/(2*sigma*sigma)));
    F += w(i) * P_temp;
  end
  F += w(n+1);
  er1 = (dT(j) - F);
  err = er1 * er1;
  TRMS += err;
end

for i=1:800
  F = 0;

  % Find the closest center
  for j=1:n
    tr = x(i,:) - t(j,:);
    closest_center(j) = sqrt((tr(1)*tr(1))+(tr(2)*tr(2)));
    r = closest_center(j);
    P_temp = exp(- ( (r*r)/(2*sigma*sigma)));
    rbf_out(j) = w(j) * P_temp;
    F += rbf_out(j);
  end
  % take care of the bias term
  rbf_out(n+1)=1;
  F += w(n+1);

  % now nudge the center
  [min_val min_ind] = min(closest_center);
  t1 = t(min_ind,:);
  t(min_ind,:)= t1 + eta * (x(i,:) - t(j,:));

  % and train the output neuron via LMS
  e = d(i) - F;
w += $\eta L \cdot \text{rbf\_out} \cdot e$;
err = e * e;
RMS += err;
end
RMS = sqrt(RMS/800);
TRMS = sqrt(TRMS/800);
ePOCHRMS(epoch,:) = [RMS TRMS];
deltaerror = abs(RMS - olderror);
olderror = RMS;
endwhile
out = epochRMS;
endfunction

C Plotting Function
function plotit(tmp)
hold on;
gplot tmp using 1 title "RMS Training Error"
gplot tmp using 2 title "RMS Testing Error"
endfunction

D EPS Capture Plotting Function
function plt2eps (filename)
    gset terminal postscript eps enhanced color;
eval(sprintf("gset output "%s\n",filename,".eps"));
replot;
gset terminal windows;
printf("Wrote %s/%s\n",pwd,filname,".eps");
endfunction