Seth Boren

Q7 Partial Differential Equations

(Dr. Meyer-Baese; Summer 2017)

Part 1

Self organize map (SOM) or Kohonen map

For every input vector x_i , there exist a "Best Matching Unit"(BMU).

The BMU_k is the output node weight that is most similar to the weight of the input node x_k .

For multiple weights per node, the norm of each input vector weight is compared to find the most similar output node weight norm in order to find the BMU.

The variable r represents the distance from the BMU_k node to each of the other nodes within the output layer. The neighborhood function Θ(r) varies with respect to the distance **r**.

A neighborhood radius is established. For problems where the neighborhood radius varies with time, the neighborhood radius shrinks in regards to time. The neighborhood is an area around the BMU in the output layer where the distance from the BMU node to other nodes is less than the neighborhood radius.

For every node in the output layer that lies within the neighborhood radius, the output node weight $W_{i,j}$ is adjusted by the following matrix equation:

Weight Change per Node = $(x_k - W_{i,j}) \quad \Theta(r_{i,j,k})$

Input Vector with multiple weights $= x_k = \begin{bmatrix} x_{k,1} \\ x_{k,2} \\ * \\ * \\ w_{k,m} \end{bmatrix}$ where m = number of weights per vector.

Output Node with multiple weights = $W_{i,j} = \begin{bmatrix} w_{i,j,1} \\ w_{i,j,2} \\ * \\ * \\ w_{i,j,m} \end{bmatrix}$

$\Theta(r_{i,j,k})$ = Scalar value for the Neighborhood Function

where $\Theta(r_{i,j,k})$ represents the neighborhood function value for a specific distance $r_{i,j,k}$,which is the distance from the BMU_k output node and the output node $w_{i,j}$.

<u>The entire process of solving the output node layer matrix is outlined in the attached part 3.</u>

Part 1 continued

The number of operations to calculate the weight change for one output node :

Weight Change per Node = $(x_k - W_{i,j}) \quad \Theta(r_{i,j,k})$

Assuming the calculation of the neighborhood function is 1 operation.

Subtraction operations for $(x_k - W_{i,j})$ is equal to m .

Multiplication operations for $(x_k - W_{i,j}) \quad \Theta(r_{i,j,k})$ is equal to m.

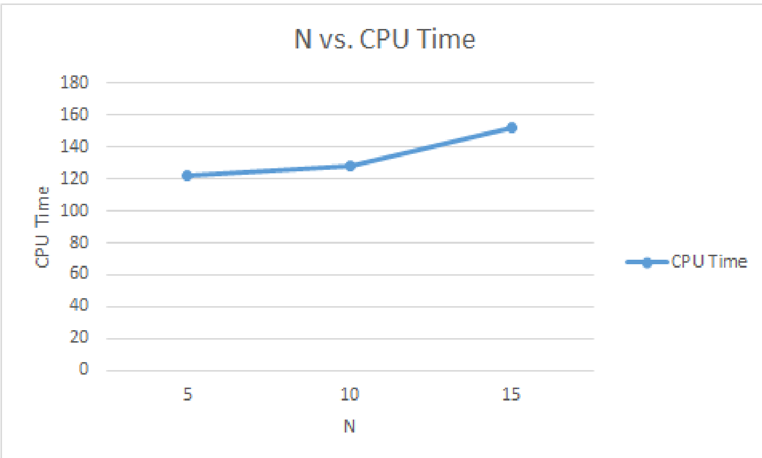
Total Number of Operation for one weight change = 2m

Note: See end of part 3 for full calculation of operations to solve SOM map.

Seth Boren

Part 2

See attached matlab code **Q10_2_Summer_2017.m** for program to calculate the neural net.



Q10 Part 2 Results		
M = 250000		
Iterations	**N**	**Time (seconds)**
1000	5	122.4375
1000	10	127.7656
1000	15	152.5313

Note: See end of part 3 for full calculation of operations to solve SOM map.

Part 3

Self organize map (SOM) or Kohonen map

For every input vector x_i , there exist a "Best Matching Unit"(BMU).
The BMU_k is the output node weight that is most similar to the weight of the input node x_k .

For multiple weights per node, the norm of each input vector weight is compared to find the most similar output node weight norm in order to find the BMU.

The variable r represents the distance from the BMU_k node to each of the other nodes within the output layer.

The neighborhood function Θ(r) varies with respect to the distance **r**.

A neighborhood radius is established. For problems where the neighborhood radius varies with time, the neighborhood radius shrinks in regards to time. The neighborhood is an area around the BMU in the output layer where the distance from the BMU node to other nodes is less than the neighborhood radius.

For every node in the output layer that lies within the neighborhood radius, the output node weight W is adjusted by the following formula:

$$W_{new} = W_{old} + \Theta(r)\,(x_k - W_{old})$$

where
W_{new} = *adjusted output node weight*
W_{old} = *original output node weight*
x_k = *input weight*
$\Theta(r)$ = *neighborhood function*

$$[W]_{old} = \begin{bmatrix} w_{11} & \cdots & w_{1N} \\ \vdots & \ddots & \vdots \\ w_{N1} & \cdots & w_{NN} \end{bmatrix}$$ represents the weights of the N x N 2-dimensional output node layer

The “weight change” for **each** individual output node in the layer is given by the formula:

Weight Change = $\Theta(r_{i,j,k})\,(x_k - W_{i,j})$

where
$w_{i,j}$ = *original output node weight or weight norm*
x_k = *input weight scalar or input weight norm*
$\Theta(r_{i,j,k})$ = *neighborhood function*
$r_{i,j,k}$ = *distance from BMU output node to each individual output node*

Part 3 continued

There does not seem to be a practical way to solve the entire output layer matrix weight adjustment with a single Matrix Equation.

The following is an alternate approach that uses a matrix equations to solve the output layer matrix one column at time.

The **weight change function** related to a specific input scalar weight $\boldsymbol{x_k}$ is given by the following for one column of the N x N output layer :

$$\boldsymbol{Weight\ Change\ for\ Output\ Layer\ Column} = (\ \boldsymbol{x_k}\,[\mathbf{1}] - [\boldsymbol{W}]_{\boldsymbol{Column}}\,)\,\boldsymbol{\Theta}$$

where $[W]_{Column}$ = $[w_{11} \quad w_{21} \quad * \quad * \quad w_{N1}]$ is a transposed column from the 2-D output node layer matrix.

and $\Theta = \Theta(r_{i,j,k})$ = $\begin{bmatrix} \Theta_{i,j} & 0 & 0 & 0 \\ 0 & \Theta_{i,j} & 0 & * \\ 0 & 0 & * & 0 \\ 0 & * & 0 & \Theta_{i,j} \end{bmatrix}$

where $\Theta_{i,j}$ represents the neighborhood function value for a specific distance $r_{i,j,k}$,which is the distance from the BMU$_k$ output node and the output node $w_{i,j}$.

and $[1] = [1 \quad 1 \quad * \quad * \quad 1]$.

Pseudo Code without shrinking the neighborhood radius:

1.) Compare input vector weight norm $\boldsymbol{x_k}$ to all of the output weight norms $w_{i,j}$.
2.) Choose the most similar output weight norm $w_{i,j}$ to be the BMU$_k$ associated with $\boldsymbol{x_k}$.
3.) For each row of the output layer matrix use the following
$$\boldsymbol{Weight\ Change\ for\ Output\ Layer\ Column} = (\ \boldsymbol{x_k}\,[\mathbf{1}] - [\boldsymbol{W}]_{\boldsymbol{Column}}\,)\,\boldsymbol{\Theta}$$
4.) Repeat Step 1 thru 3 for each of the input vectors $\boldsymbol{x_k}$
5.) end

Note: As stated in part 1, for multiple weights the output node weight and input weight are the norm of the respective multiple weights.

Part 3 continued

Defining an initial solution as an adjustment of the BMU node weight to match the input vector weight.
The value of r for the BMU node is always zero.
For both Gaussian Neighborhood functions and Inverse Multiquadrics Neighborhood functions, the neighborhood function is equal to 1 for r values of zero which is the case for the BMU node.
Therefore only one iteration is required to fully adjust the BMU weight to match the input weight, assuming there is not a" learning rate factor" added to the problem.

$$BMU_{new} = BMU_{old} + \Theta(r)\ (\ x_k - BMU_{old})$$

For the BMU weight adjustment this formula becomes:

$$BMU_{new} = BMU_{old} + 1 * (\ x_k - BMU_{old})$$

Which means after this one iteration:

$$BMU_{new} = x_k$$

The number of operations to calculate the weight change for each BMU in the N x N output layer :

$$\boldsymbol{Weight\ Change\ for\ Output\ Layer\ Column} = (\ \boldsymbol{x_k}\,[\mathbf{1}] - [\boldsymbol{W}]_{\boldsymbol{Column}}\ )\ \boldsymbol{\Theta}$$

Assuming the calculation of the neighborhood function is 1 operation.
To calculate the weight change for one column of the output node layer matrix for each input weight:

Multiplication operations for $x_k\,[1]$ is equal to N.
Subtraction operations for $(\,x_k[1] - [W]_{old}\,)$ is equal to N .
Operations to calculate the neighborhood function values for one column of output layer is equal to N.
Multiplication operations for $(\ \boldsymbol{x_k}\,[\mathbf{1}] - [\boldsymbol{W}]_{\boldsymbol{Column}}\ )\ \boldsymbol{\Theta}$ is equal to NxN.

Number of Operations per row of the output layer = N + N + N + N^2 = 3N + N^2

For all BMU weigh adjustments with N columns and k input vectors:

Total Number of Operations = (N)(k)(3N + N^2) = k(3N^2+ N^3)

Part 4

See attached matlab code **Q10_ 4_Summer_2017.m** which has a shrinking neighborhood radius with respect to time.

```
%Seth Boren
%Summer 2017 Q10 Prelim Problem
%Part 2
%Neighborhood Radius/Variance that shrinks with time”
% setting up input and output matrices for a neural net
% n can be whatever number you want, it just says how many vectors are going to be in the
list
% row and row are for the dimensions of the output matrix
t = cputime;

n = 15;
row = 200;
col = 200;
weights = 1;   %weights is like the Z axis
M = row * col; %each neuron has two weights
%-------------------------
vector_list = zeros(weights,n);
sigma_list = zeros (weights,n);
output = zeros(row,col,weights);
norm_output = zeros(row,col);
distance = zeros (row,col);
radius_zone = zeros(row,col);
%creating a basic vector list, each vector in the list has 2 entries, one for each weight
(there are going to be two weights)

for i = 1 : weights              %this is the the rows of the matrix
    for j = 1 : n                %this is the column of the matrix
        vector_list(i,j) = rand; %each entry in the vector list is a number between 0 and
1
        sigma_list (j) = rand;
    end
end

%initialize output weights randomly, each weight between -1 and 1
a = -1;
b = 1;
r = (b-a).*rand + a;
for i = 1 : row                   % X
    for j = 1 : col               % Y
        for k = 1 : weights       % Z
            a = -1;
            b = 1;
            r = (b-a).*rand + a;
            output(i,j,k) = r; % initializing a node in the neural net
        end
    end
end

time = 1;

lambda = 18;
%LOOP STARTS HERE
for time = 1 : 1000
%-------------------------
%choosing the vector_input
    chosen = randperm(n,1);          %randperm chooses an integer between 1 and n

    vector_input = vector_list(:,chosen); %out of the vectors in the list (1 to n) we have
chose an input
    sigma_input  =  sigma_list(chosen); %sigma_input represents the variance for the
neighborhood function for this specific problem
    %compute distance for each node to the vector_input
    for i = 1 : row
        for j = 1: col
            weight_distance_sum = 0;

            if radius_zone(i,j) == 0
                for k = 1 : weights
                    difference_squared = (vector_input(k,1) - output(i,j,k))^2  ;
                    weight_distance_sum = weight_distance_sum + difference_squared;
                end
```

```
            end

            distance(i,j) = sqrt(weight_distance_sum);
        end
    end

    %Initializing BMU
    BMU = zeros(weights,1);
    for i = 1 : weights
        BMU(i,1) = 1;
    end
    BMU = norm(BMU);
    %BMU starts out at 1,1 the maximum number for the distance matrix entries.
    %radius_zone = zeros(row,col);
    %Find the BMU (Best Matching Unit)
    BMU_cand = zeros(weights,1);
    for i = 1 : row
        for j = 1: col

            if radius_zone(i,j) == 0

                BMU_cand = distance(i,j); %candidate for minimum distance

                if BMU_cand <= BMU        %deciding what distance is the smallest
                    BMU = BMU_cand;
                    BMU_row = i;
                    BMU_col = j;
                end

            end

        end
    end
    %at the end of the loop the BMU_row and BMU_col are known

    distance_matrix = zeros(row,col);

    %A BUNCH OF PRACTICE 'sigma_t's and 'learn_t's
    %sigma_t represents the neighborhood radius
    %for prelim question 10 sigma is  constant.
    sigma_t = col/2;
    %learning function has been turned off for this problem
    %learn_t = 0.9*exp(-time/lambda);
    learn_t = 1;
    %now to choose the neighborhood function.  The neighborhood function decides what
nodes are in the vicinity of the BMU
    for i = 1 : row
        for j = 1 : col
            neighbor_cand = [i j];
            centerpoint = [BMU_row BMU_col];
            neighbor_distance = abs(norm(neighbor_cand - centerpoint));
            distance_matrix(i,j) = neighbor_distance;
        end
    end

    distance_matrix;

    for i = 1 : row
        for j = 1:col

            if distance_matrix(i,j) < sigma_t
                radius_zone(i,j) = 0;
            end

            if distance_matrix(i,j) > sigma_t
                radius_zone(i,j) = 1;
            end

        end
    end
```

```
    %Find all the positions for output within the radius and update them
    %theta_t is the neighborhood function, distance_matrix is the
    %difference in weights between the vector_input weights minus the
    %output node weights.  Sigma_t is the radius of the neighborhood.
    for i = 1 : row
        for j = 1:col
            if radius_zone(i,j) == 0
                for k = 1:weights
                    theta_t(k) = exp(  (-(distance_matrix(i,j) )^2) / (2 * (sigma_t^2)) );
                    diff = vector_input(k,1) - output(i,j,k);
                    %adjust weight for output node
                    output(i,j,k) = output(i,j,k) + theta_t(k) * learn_t * diff;
                end
            end
        end
    end


    %compute distance for each node to the vector_input
    %for i = 1 : row
     %   for j = 1: col
       %     for k = 1 : weights
      %          norm_addition = norm(output(i,j,k));
      %          norm_output(i,j) = norm_output(i,j) + norm_addition;
     %       end
      %      norm_output(i,j) = norm_output(i,j)/weights;

      %  end
    %end

    %
    %format short
    %fprintf('------------------------------------------ \n')
    %the_input = norm(vector_input(1,1) + vector_input(2,1));
    %the_output= norm(output(BMU_row,BMU_col,1) + output(BMU_row,BMU_col,2));
    %fprintf('time = %d  \n', time );
    %fprintf('BMU Node = [ %d ] [ %d ] \n', BMU_row, BMU_col);
    %fprintf('Output BMU Node contents = %d \n', output(BMU_row,BMU_col));

end
output
vector_list
clock_time = cputime-t;
%surf(output)
%vector_list
clock_time
```

```
%Seth Boren
%Summer 2017 Q10 Prelim Problem
%Part 4
%Neighborhood Radius/Variance that shrinks with time”
% setting up input and output matrices for a neural net
% n can be whatever number you want, it just says how many vectors are going to be in the
list
% row and row are for the dimensions of the output matrix
t = cputime;

n = 10;
row = 80;
col = 80;
weights = 1;   %weights is like the Z axis
M = row * col; %each neuron has two weights
%------------------------
vector_list = zeros(weights,n);
sigma_list = zeros (weights,n);
output = zeros(row,col,weights);
norm_output = zeros(row,col);
distance = zeros (row,col);
radius_zone = zeros(row,col);
%creating a basic vector list, each vector in the list has 2 entries, one for each weight
(there are going to be two weights)

for i = 1 : weights              %this is the the rows of the matrix
    for j = 1 : n                %this is the column of the matrix
        vector_list(i,j) = rand; %each entry in the vector list is a number between 0 and
1
        sigma_list (j) = rand;
    end
end

%initialize output weights randomly, each weight between -1 and 1
a = -1;
b = 1;
r = (b-a).*rand + a;
for i = 1 : row                   % X
    for j = 1 : col               % Y
        for k = 1 : weights       % Z
            a = -1;
            b = 1;
            r = (b-a).*rand + a;
            output(i,j,k) = r; % initializing a node in the neural net
        end
    end
end

time = 1;

lambda = 18;
%LOOP STARTS HERE
for time = 1 : 80
%------------------------
%choosing the vector_input
    chosen = randperm(n,1);          %randperm chooses an integer between 1 and n

    vector_input = vector_list(:,chosen); %out of the vectors in the list (1 to n) we have
chose an input
    sigma_input  =  sigma_list(chosen); %sigma_input represents the variance for the
neighborhood function for this specific problem
    %compute distance for each node to the vector_input
    for i = 1 : row
        for j = 1: col
            weight_distance_sum = 0;

            if radius_zone(i,j) == 0
                for k = 1 : weights
                    difference_squared = (vector_input(k,1) - output(i,j,k))^2  ;
                    weight_distance_sum = weight_distance_sum + difference_squared;
                end
```

```
            end

            distance(i,j) = sqrt(weight_distance_sum);
        end
    end

    %Initializing BMU
    BMU = zeros(weights,1);
    for i = 1 : weights
        BMU(i,1) = 1;
    end
    BMU = norm(BMU);
    %BMU starts out at 1,1 the maximum number for the distance matrix entries.
    %radius_zone = zeros(row,col);
    %Find the BMU (Best Matching Unit)
    BMU_cand = zeros(weights,1);
    for i = 1 : row
        for j = 1: col

            if radius_zone(i,j) == 0

                BMU_cand = distance(i,j); %candidate for minimum distance

                if BMU_cand <= BMU        %deciding what distance is the smallest
                    BMU = BMU_cand;
                    BMU_row = i;
                    BMU_col = j;
                end

            end

        end
    end
    %at the end of the loop the BMU_row and BMU_col are known

    distance_matrix = zeros(row,col);

    %A BUNCH OF PRACTICE 'sigma_t's and 'learn_t's
    %sigma_t represents the neighborhood radius
    %for prelim question 10 sigma is  constant.
    sigma_0 = row ;
    sigma_t = sigma_0 * exp(-time/lambda);
    %learning function has been turned off for this problem
    %learn_t = 0.9*exp(-time/lambda);
    learn_t = 1;
    %now to choose the neighborhood function.  The neighborhood function decides what
nodes are in the vicinity of the BMU
    for i = 1 : row
        for j = 1 : col
            neighbor_cand = [i j];
            centerpoint = [BMU_row BMU_col];
            neighbor_distance = abs(norm(neighbor_cand - centerpoint));
            distance_matrix(i,j) = neighbor_distance;
        end
    end

    distance_matrix;

    for i = 1 : row
        for j = 1:col

            if distance_matrix(i,j) < sigma_t
                radius_zone(i,j) = 0;
            end

            if distance_matrix(i,j) > sigma_t
                radius_zone(i,j) = 1;
            end

        end
    end
```

```
    %Find all the positions for output within the radius and update them
    %theta_t is the neighborhood function, distance_matrix is the
    %difference in weights between the vector_input weights minus the
    %output node weights.  Sigma_t is the radius of the neighborhood.
    for i = 1 : row
        for j = 1:col
            if radius_zone(i,j) == 0
                for k = 1:weights
                    theta_t(k) = exp(  (-(distance_matrix(i,j) )^2) / (2 * (sigma_t^2)) );
                    diff = vector_input(k,1) - output(i,j,k);
                    %adjust weight for output node
                    output(i,j,k) = output(i,j,k) + theta_t(k) * learn_t * diff;
                end
            end
        end
    end


    %compute distance for each node to the vector_input
    %for i = 1 : row
     %   for j = 1: col
       %     for k = 1 : weights
      %          norm_addition = norm(output(i,j,k));
      %          norm_output(i,j) = norm_output(i,j) + norm_addition;
     %       end
      %      norm_output(i,j) = norm_output(i,j)/weights;

      %  end
    %end

    %
    %format short
    %fprintf('------------------------------------------ \n')
    %the_input = norm(vector_input(1,1) + vector_input(2,1));
    %the_output= norm(output(BMU_row,BMU_col,1) + output(BMU_row,BMU_col,2));
    %fprintf('time = %d  \n', time );
    %fprintf('BMU Node = [ %d ] [ %d ] \n', BMU_row, BMU_col);
    %fprintf('Output BMU Node contents = %d \n', output(BMU_row,BMU_col));

end
output
vector_list
clock_time = cputime-t;
surf(output)
%vector_list
clock_time
```