```
% SOM routine
% uses the following scripts or functions:
% vec2pat(), pat2vec()
% find_closest_cluster()
% alphafnc(i,j,ictr,jctr,time)
% view_all_pattern_responses()
% eval_test_patterns

% clear;  % delete all memory
% clc;    % clear windows screen
% clf;    % clear figure screen
% shg;    % put figure screen on top of all screens

% load in training patterns
load scrambled_blobs

% shuffle the random number generator
%rng 'shuffle'

% find the size and number of these patterns
[npats,nrows,ncols]=size(scrambled_blobs);

% image represented as a vector has this many elements
vecdim=nrows*ncols;

% the maximum number of epochs
MAX_NUM_ITERS = 100000;

% the maximum learning rate
MAX_ALPHA = 0.5;

% the minimum learning rate
MIN_ALPHA = 0.01;

% specify cluster grid dimensions
nclustrows=8;
nclustcols=8;

% initial neighborhood radius
MAX_RADIUS = (max(nclustrows, nclustcols)/2);

clusters=zeros(nclustrows,nclustcols,vecdim);

% seeds all clusters with random values between 0 and 1
for i=1:nclustrows
   for j=1:nclustcols
      clusters(i, j, :) = rand(vecdim,1);
   end
end

% initialize sample pattern matrix
scrambled_pats = zeros(npats,vecdim);
```

```
% normalize each training pattern
for ipat=1:npats
   temp=pat2vec(squeeze(scrambled_blobs(ipat,:,:)));
   temp=temp/norm(temp);
   scrambled_pats(ipat,:)=temp;
end

% view decoding of 3 test patterns, I, H and X
% I, H and X will be unrecognizable at this point
eval_test_patterns;

% Now, train the clusters.
% draw from input pattern vector pool at random--
% find closest cluster, which is indexed by ibest, jbest
% update both the best-fit cluster as well as its neighbors
% neighborhood operator should shrink over time to influence fewer neighbors
% max influence of a training pattern may also decrease over time

% initialize the current iteration and display counter
curr_iter=0;
display_counter=0;

% initialize the neighborhood function vector
neighborhood = zeros(nclustrows, nclustcols);

% this is our radius decay time constant
r_time_constant = MAX_NUM_ITERS/log(MAX_RADIUS);

% set as infinite loop (and halt w/ control-C); or,  put cap on "time" for max iterations
while curr_iter <= MAX_NUM_ITERS
   % calculate the neighborhood radius at this time
   % exponential decay
   % curr_radius = MAX_RADIUS * exp(-curr_iter/r_time_constant);
   % linear decay
   curr_radius = MAX_RADIUS * (1 - curr_iter/MAX_NUM_ITERS);

   % find the variance of the neighborhood
   variance = curr_radius ^ 2;

   % calculate the learning rate at this time
   if curr_iter > 0
      %power series decay
      %curr_alpha = MAX_ALPHA*((0.005/MAX_ALPHA)^(curr_iter/MAX_NUM_ITERS));
      %linear decay
      %curr_alpha = 1/curr_iter;
      %exponential decay
      curr_alpha = MAX_ALPHA * exp(-curr_iter/MAX_NUM_ITERS);
   else
      curr_alpha = 1;
   end

   % make sure learning rate is at least the minimum
   if curr_alpha < MIN_ALPHA
      curr_alpha = MIN_ALPHA;
```

```
end

% pick a pattern at random
rand_ipat=ceil(rand*npats);

% extract the corresponding vector and transpose it to be a column
training_features=squeeze(scrambled_pats(rand_ipat,:))';

% for the selected pattern, find which cluster is most similar.
% identify this cluster in terms of its grid location, ictr,jctr
[ictr,jctr]=find_best_matching_cluster(training_features,clusters);

% calculate the neighborhood function values for each cluster
for i=1:nclustrows
   for j=1:nclustcols
      % this cluster is the winner and has best neighborhood
      % influence
      if i == ictr && j == jctr
         neighborhood(i,j)=1;
         continue
      else
         % these clusters have a scaled neighborhood influence
         % Gaussian Neighborhood
         % dist_from_bmu=(ictr - i)^2 + (jctr - j)^2;
         % neighborhood(i,j)=exp(-dist_from_bmu/(2*variance));
         % Epanechnikov Neighborhood
         % dist_from_bmu = pdist([ictr,jctr;i,j], 'euclidean');
         % neighborhood(i,j)=max(0,1-(curr_radius-dist_from_bmu)^2);
         % Cut Gaussian Neighborhood
         dist_from_bmu = pdist([ictr,jctr;i,j], 'euclidean');
         if dist_from_bmu < curr_radius
            neighborhood(i,j)=exp(-dist_from_bmu/(2*variance));
         else
            neighborhood(i,j)=0;
         end
      end
   end
end

% update the bmu cluster and its neighbors
for i=1:nclustrows
   for j=1:nclustcols
      % grab the feature vector of the current cluster
      curr_features = squeeze(clusters(i,j,:));

      updated_features=curr_features + curr_alpha*neighborhood(i,j)*...
         (training_features-curr_features);

       % normalize the newly calculated cluster features
      norm_features = norm(updated_features);
      if norm_features > 0
        updated_features = updated_features/norm_features;
      end
```

```
            clusters(i,j,:) = updated_features;
        end
    end

    % review results after each 1000 pattern updates
    if (display_counter>=1000)
        fprintf('updating graphs at iteration: %d\n', curr_iter);

        % pause to let gui update graphs
        pause(3);

        % are clusters performing decoding? Check cluster responses to I, H
        % and X scrambled test patterns
        view_all_pattern_responses(scrambled_pats,clusters);

        eval_test_patterns
        display_counter=0;

        % save cluster results to file clusters.mat
        save('clusters','clusters');
    end

    % increment the iteration and display counter by 1
    curr_iter=curr_iter+1;
    display_counter=display_counter+1;
end
```