Homework 5: SOM

Andy Reagan

I code and discuss an implementation of Kohonen's original SOM.

I. INTRODUCTION

Originally proposed by Kohonen, the self organizing map describes a general class of unsupervised artificial nueral networks [1].

[1] Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE 78*(9), 1464–1480.

Full code

```
% clear all
% close all

% load the data
training = csvread('CounterProp_Data.csv');

num_output_categories = max(training(:,end));
output_data = zeros(length(training(:,1)),num_output_categories);
for i=1:length(training(:,1))
    output_data(i,end+1-training(i,end)) = 1;
end

% make the interpolation data set
% looks like the x and y in the training data
% go close to 200 and 250, respectively
% so interpolation at every point up to those
[X,Y] = meshgrid(1:200,1:250);
interpolation = [X(:),Y(:)];

% normalize the data to the unit sphere
[training_norm,interpolation_norm] = nomalize_input_andy(training(:,1:end-1),interpolation);

% tell me what happened
fprintf('size␣of␣training␣is:␣\n');
disp(size(training));
fprintf('size␣of␣training_norm␣is:␣\n');
disp(size(training_norm));
fprintf('first␣row␣of␣training␣is:␣\n');
disp(training(1,:));
fprintf('first␣row␣of␣training_norm␣is:␣\n');
disp(training_norm(1,:));
fprintf('length␣of␣first␣row␣of␣training␣is␣(should␣be␣1):␣\n');
disp(sum(training_norm(1,:).^2));
fprintf('length␣of␣first␣row␣of␣interpolation␣is␣(should␣be␣1):␣\n');
disp(sum(interpolation_norm(1,:).^2));

numiter = 50;
% my algorithm breaks down for too many iterations....
% if I put 50, it sometimes works and sometimes doesn't
% fixed! had to use find(...,1)

% train the weight matrices
[~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,0,@scaling_none,true,false);
% again, tell me what happened
% disp(size(V));
% disp(size(W));

figure(111)
plot(1:numiter,mean(errors_all,1))
title('convergence␣of␣counterprop')
xlabel('iteration')
ylabel('avg␣RMSE␣over␣training␣data')
saveas(111,'111.png')

figure(112)
plot(1:numiter,errors_all)
title('convergence␣of␣counterprop')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
saveas(112,'112.png')

[W,V,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,1,@scaling_none,true,false);

figure(113)
plot(1:numiter,mean(errors_all,1))
title('convergence␣of␣counterprop,␣random␣training␣order')
xlabel('iteration')
ylabel('avg␣RMSE␣over␣training␣data')
saveas(113,'113.png')

figure(114)
plot(1:numiter,errors_all)
title('convergence␣of␣counterprop,␣random␣training␣order')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
saveas(114,'114.png')

% looking at a few of these...looks like it flattens out
% with the random order
% and the avg RMSE is about the same

% let's do a little bit better, and average over many runs for the
% RMSE plot
numtrials = 10;
errors_randomVstraight = zeros(2,numiter);
for i=1:numtrials
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,0,@scaling_none,true,false);
    errors_randomVstraight(1,:) = mean(errors_all,1);
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,0,@scaling_none,true,false);
    errors_randomVstraight(2,:) = mean(errors_all,1);
end

errors_randomVstraight = errors_randomVstraight./numtrials;

figure(115)
plot(1:numiter,errors_randomVstraight)
title('counterprop␣convergence,␣100␣trials')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
legend('straight','random')
saveas(115,'115.png')

numiter = 10;

% test it now
% train one with random order
randorder = true;
[kohonen_weights,grossberg_weights,~] = train_counterprop_andy(training_norm,output_data,numiter,randorder,
    @scaling_none,true,false);

test_counterprop_andy(kohonen_weights,grossberg_weights,interpolation_norm,training_norm,training,'116-standard.
    png',false);
test_counterprop_andy(kohonen_weights,grossberg_weights,interpolation_norm,training_norm,training,'116-wpoints.png
    ',true);

% now lets try to look at convergence with exponential scaling
numiter = 50;
errors_exp = zeros(2,numiter);
for i=1:numtrials
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_none,true,false);
    errors_exp(1,:) = mean(errors_all,1);
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_exponential,true,
        false);
    errors_exp(2,:) = mean(errors_all,1);
end

errors_exp = errors_exp./numtrials;

figure(117)
plot(1:numiter,errors_exp)
title('counterprop␣convergence,␣100␣trials')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
legend('no␣scaling','exponential␣scaling')
saveas(117,'117.png')

% now lets try to look at convergence separately vs together
errors_exp = zeros(2,numiter);
for i=1:numtrials
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_none,true,false);
    errors_exp(1,:) = mean(errors_all,1);
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_exponential,false,
        false);
    errors_exp(2,:) = mean(errors_all,1);
end

errors_exp = errors_exp./numtrials;

figure(118)
plot(1:numiter,errors_exp)
title('counterprop␣convergence,␣100␣trials')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
legend('together','separate')
saveas(118,'118.png')

% now lets try to look at convergence w nearest neighbor
numiter = 50;
errors_exp = zeros(2,numiter);
for i=1:numtrials
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_none,false,true);
    errors_exp(1,:) = mean(errors_all,1);
    [~,~,errors_all] = train_counterprop_andy(training_norm,output_data,numiter,false,@scaling_exponential,false,
        true);
    errors_exp(2,:) = mean(errors_all,1);
end

errors_exp = errors_exp./numtrials;

figure(119)
plot(1:numiter,errors_exp)
title('counterprop␣convergence,␣100␣trials')
xlabel('iteration')
ylabel('RMSE␣over␣each␣training␣data')
legend('normal','nearest␣neighbor')
saveas(119,'119.png')

% test randomorder on the 2D space
numiter = 50;
% i'm not sure averaging over weights makes any real sense
% averaging over convergence does, but not weights
numtrials = 1;
% for i=1:numtrials
randorder = true;
[kohonen_weightsr,grossberg_weightsr,~] = train_counterprop_andy(training_norm,output_data,numiter,randorder,
    @scaling_none,true,false);
randorder = false;
[kohonen_weights,grossberg_weights,~] = train_counterprop_andy(training_norm,output_data,numiter,randorder,
    @scaling_none,true,false);
% end
test_counterprop_andy(kohonen_weightsr,grossberg_weightsr,interpolation_norm,training_norm,training,'interpolation
    -randorder.png',false);
test_counterprop_andy(kohonen_weights,grossberg_weights,interpolation_norm,training_norm,training,'interpolation-
    straigtorder.png',true);

% test nearest neighbor in 2d
numiter = 50;
randorder = false;
[kohonen_weightsr,grossberg_weightsr,~] = train_counterprop_andy(training_norm,output_data,numiter,randorder,
    @scaling_none,false,false);
[kohonen_weights,grossberg_weights,~] = train_counterprop_andy(training_norm,output_data,numiter,randorder,
    @scaling_none,false,true);
test_counterprop_andy(kohonen_weightsr,grossberg_weightsr,interpolation_norm,training_norm,training,'interpolation
    -random.png',false);
test_counterprop_andy(kohonen_weights,grossberg_weights,interpolation_norm,training_norm,training,'interpolation-
    nearestn.png',true);
```

```
function [W,V,rmse_all] = train_counterprop_andy(A,B,numiter,randorder,scaling_fun,traintogether,nearest_neighbor)
% train the two counterprop matrices
%
% INPUTS
%
% A(num train patterns,size input)
% B(num train patterns,size output)
%
% OUTPUT
%
% V: Kohonen layer weights
% W: Grossberg layer weights

% how long to train
% numiter = 50;
% taking this from input

% hardcode learning coeff
alpha = 0.7; % kohonen
beta = 0.2; % grossberg


num_training_patterns = length(A(:,1));

% rmse_all = ones(num_training_patterns+(1-traintogether)* ...
%                   num_training_patterns,numiter);
rmse_all = ones(num_training_patterns,numiter);
% rmse_avg = ones(1,numiter);

input_size = length(A(1,:));
output_size = length(B(1,:));

% kohonen later
if nearest_neighbor
    hidden_layer_size = num_training_patterns;
    W = A';
else
```

```
    hidden_layer_size = num_training_patterns+3;
    W = rand(input_size,hidden_layer_size);
end
% disp(size(W));
% grossberg layer
V = rand(hidden_layer_size,output_size);

if ~nearest_neighbor
    fprintf('training␣kohonen␣layer\n');
    for i=1:numiter
        % fprintf('on training iteration no:\n');
        % disp(i);
        if randorder
            order = 1:num_training_patterns;
        else
            order = randperm(num_training_patterns);
        end
        for j=order
            % apply kohonen weights to compute middle layer
            % in your paper, figure 4 shows that this should compute
            % the minumim distance between input and kohonen layer
            % weights...
            % this just applies the weights forward:
            I = A(j,:)*W;
            % and this is more akin the distance
            % (1x3)*(3x78) = (1x78)
            % I = 1-A(j,:)*W;
            % NOTE: these are the "z_j" in the psuedocode

            % find the index of min
            % if we didn't take 1-... in the above, could take the max:
            winning_node = find(I==max(I),1);
            % this variable is akin to "c" in the code
            % winning_node = find(I==min(I),1);

            % update this just to look at in debugging
            % I = I>=max(I);

            % disp(I);
            % disp(winning_node);

            % update the winning node's links in kohonen layer
            % this should move the weights toward the input

            % fprintf('input');
            % disp(A(j,:)');
            % fprintf('winning row');
            % disp(W(:,winning_node));
            W(:,winning_node) = W(:,winning_node) + scaling_fun(alpha,i)*(A(j,:)'-W(:,winning_node));
            % fprintf('winning row after');
            % disp(W(:,winning_node));

            % update links in grossberg layer
            % note that y' is just V(winning_node,:)
            % since the a_j is set to 1
            if traintogether
                % save the error in the output
                output_error = B(j,:)-V(winning_node,:);

                rmse_all(j,i) = rmse(B(j,:),V(winning_node,:));
                V(winning_node,:) = V(winning_node,:) + scaling_fun(beta,i)*(output_error);
            end
        end
        % rmse_avg(1,i) = mean(rmse_all(:,i));
    end
end

% train the grossberg layer
if ~traintogether || nearest_neighbor
    fprintf('training␣just␣grossberg␣layer␣now\n');
    for i=1:numiter
        if randorder
            order = 1:num_training_patterns;
        else
            order = randperm(num_training_patterns);
        end
        for j=order
            I = 1-A(j,:)*W;
            winning_node = find(I==min(I),1);
            output_error = B(j,:)-V(winning_node,:);
            rmse_all(j,i) = rmse(B(j,:),V(winning_node,:));
            V(winning_node,:) = V(winning_node,:) + scaling_fun(beta,i)*(output_error);
        end
```

```
      end
end

end
```

Extra figures