Submission Assignment 1 Fuzzy and Neural Systems from Jens Settelmeier

We shuffel the data to make sure no sequence of the data information influence the training. Further I wrote a train, validation and test set split function. The data for each subset is picked randomly.

Actually I only worked with a validation set to check if the rbf network performs equal good on the validation set, which was not involved in training (-> adjusting the weights). If it also performs well on the validation set we can assume it is a good generalisation for data which comes from the same distributed data pool. If the test data which is provided is from the same "data pool", the network should perform on this test set as well as on the validation set.

For the h_{i,j} function I used the same as in the lecture provided, so we only adjust the weight to the corresponding winning neuron.

The subfunctions can be find at the end of this live script.

1) Data Preparation:

```
%%
load('data_train.mat')
load('label_train.mat')
data_and_label = [data_train,label_train];

data_and_label = shuffling(data_and_label);

[train_set,val_set,test_set] = data_divider(data_and_label,9,0);

train_labels = train_set(:,end);
val_labels = val_set(:,end);
test_label = test_set(:,end);

train_data = train_set(:,1:end-1);
val_data =  val_set(:,1:end-1);
test_data = test_set(:,1:end-1);
```

2) Find the centers for the RBF neural network

```
epsilon = 10^-5;
learning_rate = 0.1;
iter_limit = 100000;
numNeurons = 20;
C = som(train_data,numNeurons, learning_rate, epsilon, iter_limit) %row vectors of C ar
```

```
C = 20×33
   -0.1112   -0.0953    0.0086    0.0340   -0.0324   -0.1201    0.2275   -0.0969 ···
    0.0956   -0.1125   -0.1640   -0.1110   -0.1563   -0.0950   -0.1358   -0.0652
   -0.0975   -0.1073    0.0858   -0.0633    0.0268   -0.0599    0.0995   -0.0351
   -0.0030    0.0563    0.0120   -0.0555    0.0229   -0.0550   -0.0411    0.2348
   -0.1724   -0.0734   -0.0642    0.2217    0.2141    0.0833   -0.0984   -0.0791
```

```
   -0.2099   -0.1809    0.0607    0.1591   -0.1488    0.0074    0.0266    0.0879
   -0.0271   -0.2072    0.3311   -0.0694   -0.0112   -0.0515    0.0914   -0.1353
    0.0167    0.1006   -0.0966    0.0046   -0.0220    0.0469    0.0111   -0.0449
    0.0621   -0.1014    0.1736   -0.0311   -0.0618   -0.0803   -0.0564   -0.1686
    0.0672    0.0783    0.1911    0.1172   -0.0197    0.0940   -0.0168   -0.0301
     ⋮
```

The learning rate was picked from the lecture slides. C is the matrix with the row center vectors. Epsilon is used as the minimum changing bound which needs to be fullfiled to keep iterating.

3) calculate Phi of f = Phi * W where f is the network's output

```
sigma = 8; % should actually be = d_max * sqrt(2*m)
Phi = big_Phi(train_data,C,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
```

I determine sigma with a quick grid search and have seen later in the script there exist a experimental estimated good value for it.

4) Estimate the Weights of the RBF neural network

```
%% train the weights
W = weights_regression(Phi, train_labels);
W'
```

```
ans = 1×21
  -47.9381  192.0616  221.2551  184.6809   12.7672  336.5089  -11.1079  149.5383 ···
```

5) Cassification accuracy of the training data.

```
%% test the Network on train set:
output_train = Phi * W;

indexes_minus = find(output_train<0);
indexes_plus = find(output_train>=0);

output_train(indexes_minus) = -1;
output_train(indexes_plus) = 1;

abweichung = output_train - train_labels;

accurcy_train = 1 - length(nonzeros(abweichung))/length(output_train)
```

```
accurcy_train = 0.9300
```

I used the sign of the coordinates to determine if it belongs to class -1 or class 1. So 0 can be seen as decission boundary.

We can see that the neural network is way better than uniform random gussing which would provide an accuracy of 50%.

6) Test on a Validation set which was not involved in step 4)

```
%% test the Network on val set:
Phi = big_Phi(val_data,C,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
output_val = Phi * W;

indexes_minus = find(output_val<0);
indexes_plus = find(output_val>=0);

output_val(indexes_minus) = -1;
output_val(indexes_plus) = 1;

abweichung = output_val - val_labels;

accurcy_val = 1 - length(nonzeros(abweichung))/length(output_val)
```

```
accurcy_val = 0.9333
```

So we can see the perfomance on the validation set is almost equal to the one of the training. This means it has a good generalization achieved and especially no overfitting occured. So it should also perform equal good on the provided test data.

7) Classification result on provided test data

```
%% Classification on provided test data
load('data_test.mat')
Phi = big_Phi(data_test,C,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
output_test = Phi * W;
output_test'
```

```
ans = 1×21
    0.8525   -0.1058    1.4954    0.1095    0.9894   -0.3680    0.3104   -0.6488 ···
```

Okay this would be the output without making 0 to an hard decision boundary. See step 8 for a nicer output.

8) Modified classification result on provided test data.

```
output_test_modified = output_test;
```

```
indexes_minus = find(output_test_modified<0);
indexes_plus = find(output_test_modified>=0);

output_test_modified(indexes_minus) = -1;
output_test_modified(indexes_plus) = 1;

output_test_modified'
```

```
ans = 1×21
     1    -1     1     1     1    -1     1    -1     1    -1     1    -1     1 ···
```

This is the ouput for the classification for the provided test data. We can see, it looks like the sequence of labels was equal alocated as for the train data 1,-1,1,-1,...

9) Functions

```
%% function which shuffels the data
%%
```

```
function data= shuffling(data)
[n,~] = size(data);
data = data(randperm(n),:);
end



%% Description
% in:
% data: is a nxm matrix where n is the number of samples and m the
% number of coordinates of the sample
% validation_ratio: the ratio in percentage that should be used from data
% as validation data
% test_ratio: the ratio in percentage that should be used from data
% as test data
% out:
% train_data: random selected data which is used for training
% val_data: random selected data which is used for Validation
% test_data: random selected data which is used for testing
% all three data sets are disjoint
%%
function [train_data, val_data, test_data] = data_divider(data, validation_ratio,test_

[n,~] = size(data);

validation_amount = round(n * validation_ratio/100);
test_amount = round(n * test_ratio/100);

data_indizies = 1:n;
val_indizies = randi([1,n],1,validation_amount);
val_data = data(val_indizies,:);
```

```
data_indizies = setdiff(data_indizies, val_indizies);
data = data(data_indizies,:);
[n,~] = size(data);

data_indizies = 1:n;
test_indizies = randi([1,n],1,test_amount);
test_data = data(test_indizies,:);
data_indizies = setdiff(data_indizies,test_indizies);
train_data = data(data_indizies,:);
end



% Self Organizing Map Function to estimate the centers which are needed for
% the RBF neural Network
% note we update only the winning neuron
function W = som(data,numNeurons, learning_rate, epsilon, iter_limit)
    [~,m] = size(data);
    n_0 = learning_rate;
    W = randn(numNeurons,m); % the probability to get two equal rows is 0.
    W = W/norm(W);
    delta = epsilon +1;
    iter = 0;
    data_copy = data;
    while delta > epsilon && iter<iter_limit
        % random selection of sample from data set and making sure in the
        % next iteration it can not be drawn again
        iter2 = mod(iter,m);
        if iter2 == 0
            data_copy = data;
        end
        i = randi([1,m-iter2],1,1);
        x = data_copy(i,:);
        [n_2,~] = size(data_copy);
        filter_i =setdiff([1:n_2],i);
        data_copy = data_copy(filter_i,:);

        % calculating the dists from sample vector x to every weight neuron
        % vector w, determine the m_ind = argmin(||x-w_j||)
        X_mat = repmat(x,numNeurons,1)';
        x_dists_to_W = sqrt(sum((X_mat-W').^2,1));
        % determine the winning neuron
        [min_val,min_ind] = min(x_dists_to_W);

        % update the neuron w_winning
        eta_n = n_0 * exp(-iter/1000);
        w_winning = W(min_ind,:);
        % note only the winning neuron is updated -> h_{j,i}(iter) = {1 if
        % neuron j is the winning neuron i, 0 otherwise.
        w_update = w_winning + eta_n *1 * min_val;
        delta = norm(w_winning-w_update);

        iter = iter +1;
```

```
    end
end



% To be able to work in matrix vector notation of the RBF Network
function phi = big_Phi(data,centrum,sigma)
[number_of_sample, ~] = size(data);
[number_of_centren,~] = size(centrum);
% check if m == input_dimension
phi = zeros(number_of_sample,number_of_centren);

    for i=1:number_of_centren
        c_mat = repmat(centrum(i,:),number_of_sample,1);
        column_i_of_phi = exp((1/(2*sigma^2)) * sqrt(sum((data-c_mat).^2,2)));
        phi(:,i) = column_i_of_phi;
    end

end



% Function to train/adjust the weights.
function W = weights_regression(phi, label)
 W = (phi'*phi)\(phi'* label);
end
```