UNIVERSITÉ DE BOURGOGNE

MSC COMPUTER VISION

VISUAL PERCEPTION

Self Organizing Networks
Kohonen Networks

Author:
Mohamed EISSA

April 30, 2015

1 Introduction

Self organizing networks are type of artificial neural networks with the following characteristics

1. An unsupervised approach (unlike back propagation)
2. Learning is done changes in synaptic weights

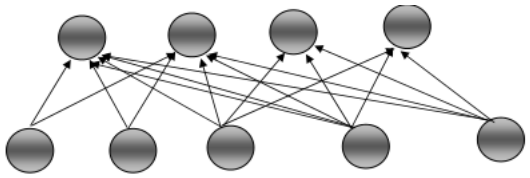

Figure 1: Self Organizing Network

In this report, we are going to demonstrate an implementation for self organizing network. the procedure is done in two steps

1. Training Phase: synaptic weights are changed to match input, euclidean distance is calculated between every input vector all current weight vectors,then the weight vector correspond to the minimum distance is going to be updated
   - Euclidean Distance is
   $$d_k = \sum (x_i - w_{ik})^2$$
   - Winner class is equal to
   $$d_{min} = min(d_0, d_1, ...d_n)$$
   where n is number of clusters or weights vectors
   - The weight updating function is
   $$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)[x_i(t) - w_{ij}(t)]$$
2. Testing Phase: synaptic weights stay fixed. We test the performance of the network. After the weights got fixed from the training phase, euclidean distance is going to be calculate again between the input vector and fixed weight vector, and the winner class is the one corresponding to the minimum distance

The implementation is done using **MATLAB**, files provided are

1. **kohonen_train.m**: Train function for Kohonen network
2. **kohonen_test.m**: Test function for Kohonen network
3. **kohonen_simple_data.m**: Trying previous defined functions on simple data of 4 input vectors each of size 4
4. **kohonen_real_data.m**: Trying previous defined functions on real data of 20 input vectors each of size 650, the first 10 input vectors are control subjects, the second 10 input vectors are patients subjects, this is the **main file to run** against provided real test data

2 Training Phase

Training function is as follows:

```
%% Kohonen Training Phase Function
function weights = kohonen_train(traindata,alpha,clusters_number,
   iterations)

% Get the number of input vectors and length of every one
[input_vectors_number, input_vector_length] = size(traindata);

% Create random weights matrix of size(clusters_number,
   input_vector_length)
weights = rand(clusters_number,input_vector_length);

% Create a temp vector that is going to use save euclidean distances
clusters_vector = zeros(clusters_number,1);

% 3 nested loop to build the weights matrix
% 1st loop is in range of number of iterations, it decrease the alpha
   every
% iteration
% 2nd loop is in range of number of input vectors, it calculates weight
% according to the minimum euclidean distance
% 3rd loop is in range of number of weight vectors, it calculates the
% euclidean distance
for i=1:iterations
    for j=1:input_vectors_number
        input_vector = traindata(j,:)';
        for k=1:clusters_number
            weight_vector = weights(k,:)';
            clusters_vector(k) = norm(input_vector-weight_vector);
        end
        [~, min_index] = min(clusters_vector);
        delta = input_vector - weights(min_index,:)';
        weights(min_index,:) = weights(min_index,:) + alpha*delta';
    end
    alpha = alpha / 2;
end

end
```

3 Testing Phase

Testing function is as follows:

```
%% Kohonen Test Phase Function
function test_results = kohonen_test(testdata,weights)

% Get number of test vectors (number of subjects)
[test_vectors_number, ~] = size(testdata);

% Get number of clusters
[clusters_number, ~] = size(weights);

% Create a temp vector that is going to use save euclidean distances
clusters_vector = zeros(clusters_number,1);
% Create a temp vector to save test results
test_results = zeros(test_vectors_number,1);

% 2 nested loop to build the assign every input vector for a cluster
% 1st loop is in range of number of input vectors, it assign the input
% vector to a cluster according to the minimum euclidean distance
% 2nd loop is in range of number of weight vectors, it calculates the
% euclidean distance between the input vector and weight vector
for j=1:test_vectors_number
    test_vector = testdata(j,:)';
    for k=1:clusters_number
        weight_vector = weights(k,:)';
        clusters_vector(k) = norm(test_vector-weight_vector);
    end
    [~, min_index] = min(clusters_vector);
    test_results(j) = min_index;
end

end
```

4 Results

The result for the provided data are as follows

```
====================
start training phase with real test data
end training
====================
start labeling phase
control class is
     1

patient class is
     2

end labeling
====================
start testing phase with real test data
Testing results
    'Control'    'Patient'    'Control'    'Patient'

end testing
====================
```