Neural Networks LAB #5

Report Submitted by Ahmed,Muhammad Farhan (EMARO)

Introduction

TASK-1:

K-means clustering

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori.

The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other.

The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step.

After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$D = \sum_{l=1}^{n} \sum_{j=1}^{k} u_{lj} \, \|\mathbf{x}_l - \mathbf{y}_j\|^2$$

Method adopted

Task 1: K-means clustering

We Write a function implementing the *k* means clustering algorithm.
The function receives two input arguments: a training set and a number of clusters, *k*.

function [y u] = kmeans(x,k)

where *y* is a matrix of prototype vectors and *u* a membership matrix that represent the assignment of points to clusters (see below). If the input vectors are *np* and of size *d*, then *y* will be *k* x *d* and *u* will be *np* x *k*.

function [y u] = kmeans(x,k)

structure of function: The algorithm works by epoch, and its structure is the familiar one: initialize - loop - finalize.

The init phase is an initial value for *y* ,randomly (but to stay inside the "cloud" of points of the training set).

The loop consists of two alternate actions:

1. Starting from the current value of *y*, assign each point in the training set (row of *x*) to its nearest prototype (row of *y*).
2. Starting from the current assignment, recomputed each row of *y* as the mean of the points assigned to it.

The stopping criterion is: no change happens. If either of the two above actions does not produce any change, then the other will not either, so it is useless to keep on looping, and we stop.

Finalization Graphs

1. No of iterations Vs Distortion.
2. K means clustring.

How to structure of variables: We use a **membership** or **indicator vector** for each point. For a given point, its membership vector *u* has size equal to *k*. The point is assigned to prototype *j* (and therefore to cluster *j*) if u(j) == 1, and u(k) == 0 for all other k, k ~= j.

So the *j*-th column of *u* acts as a "filter", indicating all inputs that are assigned to prototype *j*. We can take the mean of the whole training set where each point is **multiplied by its membership**. In this way, points with zero membership will not contribute to the mean:

$$\mathbf{y}_j = \frac{1}{\sum_{l=1}^{n} u_{lj}} \sum_{l=1}^{n} u_{lj}\, \mathbf{x}_l$$

In Matlab this mean can be computed very efficiently:

```
y = u' * x; % this is a matrix with k rows; each row is a sum of points
s = sum(u); % this is a vector with k entries; each entry is the number of points in a cluster
for j = 1:size(y,1)
  y(j,:)=y(j,:)./s(j);  % this turns each sum into a mean
end
```

Test:

To objectively check the progress of the algorithm, we compute the (squared) distortion:

$$D = \sum_{l=1}^{n} \sum_{j=1}^{k} u_{lj} \, \|\mathbf{x}_l - \mathbf{y}_j\|^2$$

Pecedure adopted(Algorithm)

a. Accept the two input arguments x and k were x is two dimensional dataset without target and k is no of clusters.
b. initialize our variables

- np=no. of input patterns.
- d=depth of input =2
- Select ramdom points in training set.(prototypes).
- Initialize u as zeros(np,k)
- Nepoches = 100, diff =1,n=0;

c. main loop(While)

- define stopping criteria as n<nepochs and diff ~=0
- for L=1 to np
- calculate distance d and membership matrix u
- calculate distortion D
- end for loop.
- Save y.
- Calculate new protopype matrix y as conteriods.
- Calculate diff.
- Increment n=n+1;
- Display results graphically.

End main loop

TASK-2:

Kohonen's self Organizing Map

Kohonen Self Organising Feature Maps or SOMs as were invented by a man named Teuvo Kohonen, a professor of the Academy of Finland, and they provide a way of representing multidimensional data in much lower dimensional spaces - usually one or two dimensions.In addition, the Kohonen technique creates a network that stores information in such a way that any topological relationships within the training set are maintained.

A SOM does not need a target output to be specified unlike many other types of network. Instead, where the node weights match the input vector, that area of the lattice is selectively optimized to more closely resemble the data for the class the input vector is a member of. From an initial distribution of random weights, and over many iterations, the SOM eventually settles into a map of stable zones. Each zone is effectively a feature classifier, so you can think of the graphical output as a type of feature map of the input space. If we take another look at the trained network shown in figure 1, the blocks of similar color represent the individual zones. Any new, previously unseen input vectors presented to the network will stimulate nodes in the zone with similar weight vectors

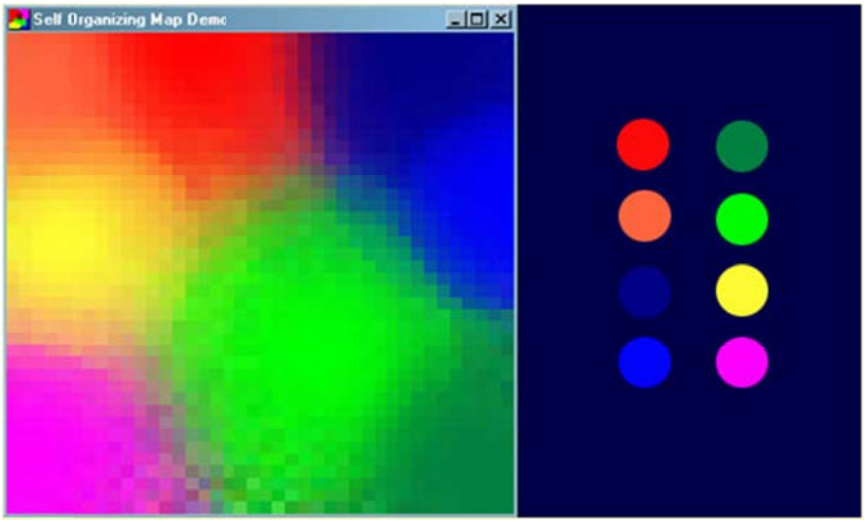


Figure 1
Screenshot of the demo program (left) and the colors it has classified (right).

Method adopted

Task 2: kohonen's self organizing Map

- Learning by pattern. Updates occur at each pattern, and the learning rate eta is decreased at each iteration.
- The second is involving not only the nearest prototype (as in *k* means), but, to a lesser extent, also the other prototypes.

Idea of the algorithm: There is a vector *y* of *k* prototypes. For them, a fixed grid (a so-called topology) is defined: each prototype is considered connected to other prototypes as (for instance) in the following diagram:

At each pattern, prototype number *j* is updated with the following updating rule:

$$\mathbf{y}_j(t+1) = \mathbf{y}_j(t) + h\,\eta\,(\mathbf{x} - \mathbf{y}_j(t))$$

Neighbohood: *h* is a *neighborhood* function.

- if $\mathbf{y}_i$ is the winner, h=1
- if $\mathbf{y}_i$ is one step away from the winner, h = g
- otherwise, h = 0

Or we can write *h* as an explicit function of the number of hops on the grid:

h = (1 - g*hops) if positive, or else 0 (piecewise linear)
or
h = exp(-hops^2/g) (Gaussian)

In all cases, 0 < g < 1. h decreases with distance (the number of hops on the grid) and g **decreases with the iteration** t. In other words, the neighborhood shrinks with time.

Computing the number of hops: on the square grid, the number of hops between two prototypes is the difference in column indexes plus the difference in row indexes. If the units are layed out by rows as in the figure (first row 1...4, second row 5...8, and so on), a function to convert prototype number (*j*) into row and col indexes (*r*, *c*) is:

```
function [row, col] = togrid(j,numcol)
```

and a function to compute the distance (on the grid) between prototype j and prototype k is:

```
function d = griddist(j,k,numcol)
```

Pecedure adopted(Algorithm)

d. Accept the two input arguments x and k were x is two dimentional dataset with out target and k is no. of clusters.
e. Initilize our variables

- np=no. of input patterns.
- d=depth of input =2
- Select ramdom points in training set.(prototypes).
- Initialize u as zeros(np,k)
- Nepoches = 100, diff =1,n=0;
- eta0=1;% max value of learning rate
- etaf=0;% final eta
- numcol=sqrt(k); % no of colums = 2 for k=4.

f. main loop(While)

- initilze D=0;

- Shuffle the dataset randomly
- Compute eta for this iteration.
- g=eta; % g varies from eta0 to etaf form (1 to 0);
- for i=1 to np
- for j=1 to k
- compute distance of x from all prototypes y
- end this for loop
- calculate difference
- find index of minimum distance prototype.
- for i=1 to k
- find no. of hops(for this min distance index)
- if k ~ index then update h
- else if k=index then h==1 (winner)
- end this loop
- upply apdate for this pattern.Because training is by pattern.
- End main for loop (npatters.)
- Print distortion.
- Update n = n+1;
- Plot results for visualization.
- End main while loop.(nepochs).

Problems:

What should be the stopping criteria ?

TASK-3:

Neural Gas:

Method

Repeated Task 2 changing *h* as follows:

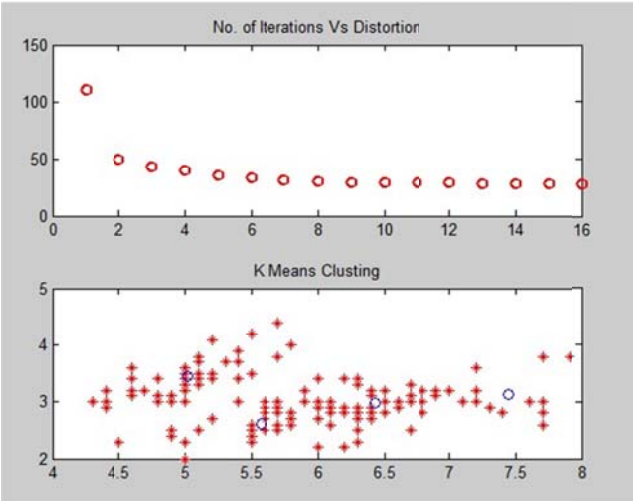
Instead of distance on a topology (grid), *h* is a function of the position (rank) of the current prototype when ordering all prototypes by increasing distance from the winner.

That is, the nearest prototype (winner) has distance 0; the second nearest has distance 1; the third nearest has distance 2...

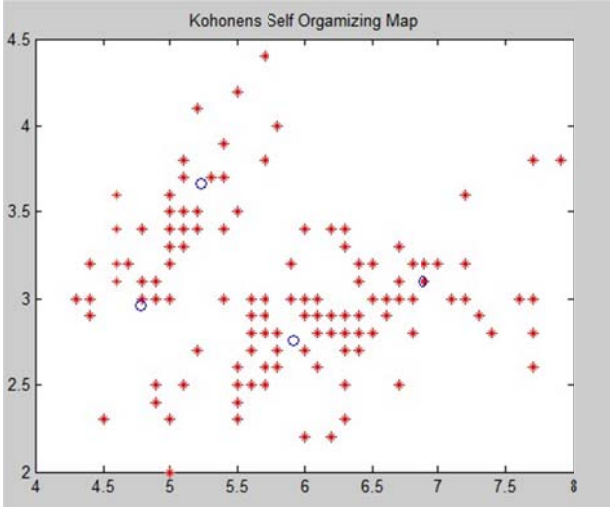
Make *h* decrease exponentially with the rank: $h = \exp(-\text{rank}/g)$, and g decrease with iterations

Results:

K Means with k=4



Kononen's self organizing map with k=4 , eta exponentially decaying form 1 to zero



Neural Gas with K=4 eta exponentially decaying form 1 to zero