Self Organizing Feature Maps

Evan Verworn (4582938) <ev09qz@brocku.ca>

***Abstract*—This report is to show the uses of a Self Organizing Feature Map in cases of clustering. The Self Organizing Maps are then analyzed using density based clustering modelling and cluster validation using the Dunn Index.**

I. Introduction

THE purpose of this assignment is for use to show that we understand Self Organizing Maps and methods of performing analysis on the results. Self Organizing Maps also are useful for reducing the curse of dimensionality problem that arises within other ANNs by eliminating data points that are too similar or outliers.

II. Problem

This report will look at two toy problems in image analysis. First a colour organization problem where a subset of colours are given and the goal is to cluster the colours into similar and dis-similar groups. The second problem will be a type of Optical Character Recognition (or OCR), wherein the goal of this problem is to find the number of different characters presented to it.

A. Problem I: Colour Map

As described above the purpose of this problem is to group similar colours. For the purposes of this problem I am limiting my colours to a subset of 13. These 13 colours are the built in colours of the Java AWT library, they are as follows. Black, Blue, Cyan, Dark Gray, Gray, Green, Light Gray, Magenta, Orange, Pink, Red, White and Yellow.

These colours were chosen because of their similar and dis-similar natures. The solution application should put the colours White and Black into different clusters but possibily cluster White and Light Gray into a similar grouping.

All of the above colours were transformed into their Red, Green and Blue components which then became the input data for this problem. All values are in the $\mathbb{R}$ domain ranging from 0 to 1.

B. Problem II: OCR

Optical Character Recognition is a problem traditionally solved with an adaline Neural Network. However in this case instead of attempting to classify an image as a letter, the goal will be to determine the number of different letters shown to it and group similar characters together.

For the purposes of this problem I will be giving it the first 6 letters of the alphabet (A through F). All letters will be in their capital letter forms, centred in the middle of the image with a plain white background. These images will then be preprocessed before analysis begins.

The first 8 attributes of the input vector will be as follows.

Average Colour
: The average Red, Green and Blue (RGB) colour of the entire image.

Standard Deviation
: The standard deviation of the RGB colours of the entire image.

Average Singular Colour
: This attribute is the average of the Red/Green/Blue channel for the entire image. Each of the colours will have their own value in the final vector.

Standard Deviation of a Singular Colour
: The standard deviation of the Red/Green/Blue channel for the entire image. Each of the colours have their own value in the final vector.

In addition to these values the image will be split into 9 equal sections. With each subsection having an *Average Colour* attribute along with a *Standard Deviation* of the RGB value.

In total this means that each image will have a 26 long input vector, with all values in the $\mathbb{R}$ domain and in the range of 0 to 1. The problem has a much higher dimensionality and also provides the challenge of determining how to best visualize the problem. As the number of features is so high, this means that the no matter what colour mapping technique is used no one plot can properly convey the state of the solution.

C. Using Neural Networks

Neural Networks are a method of artificial intelligence that are easy to implement and excels in processing noisy data, or in this case *clustering* noisy data. Using Self Organizing Feature Maps (SOFM or SOM), we can show the neural net our data and see what patterns emerge. Through statistical analysis we can then see what how accurate our representation is and use this to determine how many different groupings of input data we have and how hard it will be for other AI systems (like adaline networks) to determine the difference between them.

III. Neural Networks

In this section the type of Neural Network that will be described will be that of Self Organizing Feature Maps or SOFMs.

SOFMs are a type of unsupervised computational methods for visualizing high dimensional data [1]. These SOMs are also sometimes called Kohonen Maps after their creator T. Kohonen.

A Kohonen Map is a lattice of nodes that each hold a collection of weights. Each of these nodes are aware of it's location and it's neighbours in the n-dimensional map it exists in. These weights that each node holds are used to organize and cluster the input data on this lattice. Each node is aware, or directly connected to the input vectors that are being grouped onto the map.

Each node is updated using the following high level algorithm. Given a input vector I.

1) find the node with the weights that are the closest in euclidean space to that of the input node I.
2) pull the weights of this found node closer to that of I.
3) pull the neighbours closer to the given I dependant on the amount of distance from the above node.
4) decay the neighbourhood size and learning rate.

And repeat the above for each input vector that you'd like to cluster/group.

Now that the high level algorithm is stated the following is a more detailed step for step version of the process of the SOM.

A. Finding the closest node

When the 'closest' node is said, it is meant using the function.

$$distance = \sqrt{(w_0 - p_0)^2 + (w_1 - p_1)^2 + \ldots + (w_n - p_n)^2}$$

Where w_i is the weight of the input vector and p_i is the weight of the node we're testing.

The node with the lowest distance is the node we will pick for the weight update.

B. Weight Update

The weight update function is simple, the purpose is to just pull the weight of the node closer to that of the input vector. I do this but using the following equation.

$$W_i = W_i + (I_i - W_i) * L * D$$

Where W_i is the weight for the node we are currently updating, I_i is the weight of the input node we're trying to move closer to. L is the learning rate constant, and D is the distance we are from our closest node. In this case the value of $D = 1.0$.

C. Neighbourhood Update

If we only changed the single node we wouldn't be able to cluster similar vectors together. We encourage alike vectors to be placed together by updating a neighbourhood of vectors toward a given input vector. In the above weight update equation the variable D was mentioned. The following is the function that is used to calculate D.

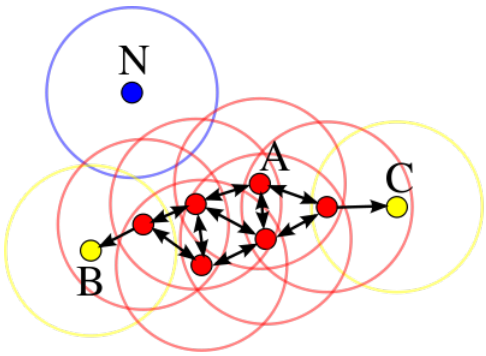

Fig. 1. A is a cluster center, B and C are reachable from A but not dense enough to be considered part of the cluster. Point N is a noise point and is not correlated to any other node.

$$d = \sqrt{min(|\Delta x|, w - |\Delta x|)^2 + min(|\Delta y|, h - |\Delta y|)^2}$$

$$D = N/(d + N)$$

Where (x, y) is the location of the node in the lattice we are measuring the distance for. w is the width of the map and h being the height of the layer. N is the neighbourhood size constant and affect how large of an area is pulled.

D. Constant Decay

As sections of the map are 'pulled' toward the group of input vectors we'd like for the map to slowly converge to one state. This is accomplished by modifying the neighbourhood size constant (N) by slowly decreasing it's value. Eventually it would reach 1 thus meaning the weight update function would only update itself, no one else. This would lead to the map converging on a single state, exactly what we want.

An example of an equation of decay that could be used.

$$N = N_s * e^{(-e/G)} + 1$$

Where N is the neighbourhood size constant, N_s is the starting neighbourhood constant, e is the current epoch the SOM is on and G being the maximum number of epochs.

IV. Methods of Analysis

A. Density Based Clustering

Density based clustering is a method of determining how many clusters the final solution has. Clusters are defined by areas with a high density of data points. A common algorithm that is used in these cases is called DBSCAN.

DBSCAN works by starting to traverse the data points, it then looks in the immediate area (ϵ) for other data points. If it is able to find at least μ data points then the current point should be considered part of a cluster.

See Figure 1 for a visual example of the algorithm.

TABLE I
PART I PARAMETERS

Map Dimensions	200
Learning Rate	0.6
Neighbourhood Size	40
Maximum Epochs	500
# Colours	13

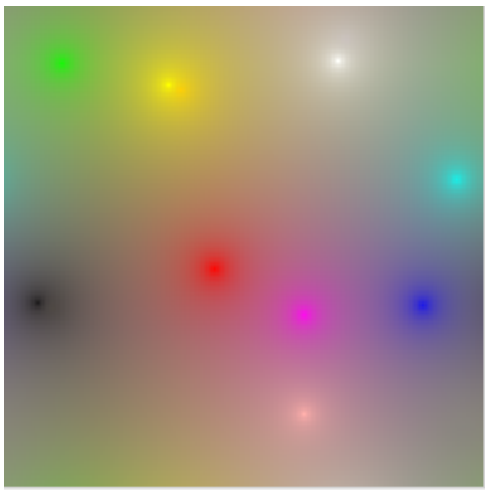

Fig. 2. Part 1 mapping results

B. Cluster Validation Using Dunn Index

Similar to the DBSCAN the Dunn Index tries to identify tightly packed clusters, giving a better score to clusters that are not spread out as far.

The score is based off of the ratio from the spread of a single cluster to the distance of the centoid of the closest different cluster centoid. The equation of the Dunn Index is as follows.

$$D = \min_{i=1..n_c} (\min_{i=i+1..n_c} (d(i,j) / \max_{k_1..n_c} diam(c_k)))$$

A very dense confusing line.

Where D is the Dunn Index, $d(i,j)$ is the inter-distance between the two clusters, $diam(c_k)$ is the intra-cluster size (size of the area of cluster).

The lower the D value the better the representation of the clusters are.

V. Results and Discussion

See Figure 2 and Table I

VI. Image Clustering Example

This part is incomplete.

VII. Conclusion

Unconclusive.

References

[1] Kohonen, T. (2001). Self-Organizing Maps. Third, extended edition. Springer, Berlin.
[2] Dunn, J. (1974). "Well separated clusters and optimal fuzzy partitions". Journal of Cybernetics 4