Generalized Decrease Schedule of Learning Parameters for SOMs

Prashant Kalvapalle, Shoeb Mohammed, Ragib Mostofa
Project Contribution: Equal

Project Report

Abstract

Self-Organizing maps (SOMs) do not optimize an objective function and thus lack an automatic measure which can be used to change learning rate and neighborhood radius for faster convergence. As a solution, we propose an efficient method to dynamically determine learning rate and neighborhood radius which also gives faster learning. We achieve this by quantifying the current learning state as a convex combination of embedding and topological errors with a corresponding criteria to change learning rate and neighborhood radius. We verified that this scheme works sufficiently well for both synthetic and real world datasets. In particular, we achieved as much as 53% speed up in convergence on synthetic datasets and 58% speed up on Iris dataset when compared against a base line scheme for learning rate and neighborhood radius.

1. Problem Statement

In the realm of unsupervised learning, self-organizing maps (or SOMs) are are easily one of the most widely utilized and highly regarded neural network paradigms. First proposed by Finnish professor Teuvo Kohonen [1] in the 1980s, SOMs have routinely succeeded in outperforming non-neural machine learning algorithms, such as the k-means algorithm, when applied to a variety of datasets. As a result, there exists no debate concerning the superiority of SOMs when unsupervised learning is considered

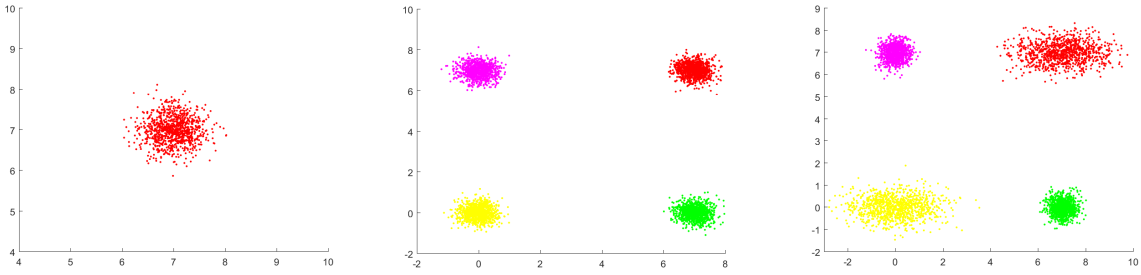
However, there does exist several aspects of the traditional SOM algorithm that could benefit from improvements. Examples include the decrease schedule of learning parameters of the self-organizing map such as the learning speed and neighborhood radii. The decrease schedule of these learning parameters is often used with predefined values that have no relation to the input data. Hence, the SOM learns in a suboptimal number of learning steps. This is mainly due to the fact that SOMs, unlike many supervised neural network algorithms, are not associated with an objective function that allows us to monitor the learning progress. Hence, our project aims to determine a solution to this problem - to introduce an efficient objective function that can be used to optimize the convergence speed of self-organizing maps by generalizing the decrease schedule of learning parameters such as the learning rate and neighborhood radius.

2. Data Description

In order to provide empirical evidence that suggests that our algorithm indeed decreases the number of learn counts required for a self-organizing map to converge, we tested our algorithm on 8 different data sets, as discussed below.

2.1. Gaussian Datasets

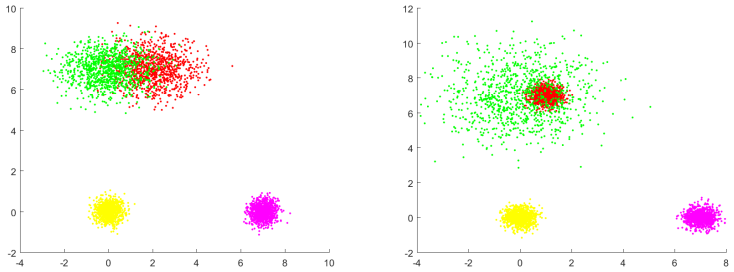
The first 5 datasets used were different variants of synthetic 2-dimensional Gaussian data - single Gaussian, four distinct Gaussians (from homework assignments), four Gaussians with different variance, four Gaussians with two partially overlapping and finally, four Gaussians with one fully contained in the other. The variance of each Gaussian in the aforementioned datasets is set to 0.1, unless mentioned otherwise (check Section 6: Appendices). Provided below, are figures illustrating the different Gaussian datasets.



Single Gaussian
No. of Input Patterns = 1000

Four Gaussians - Distinct
No. of Input Patterns = 4000

Four Gaussians - Different Variance
No. of Input Patterns = 4000



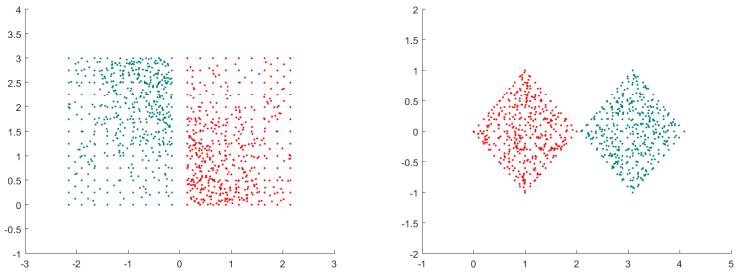
Four Gaussians - Partially Overlapping
No. of Input Patterns = 4000

Four Gaussians - Entirely Overlapping
No. of Input Patterns = 4000

Figure 1: 2D Synthetic Gaussian Datasets

2.2. Non-Gaussian Datasets

We also tested our algorithm on synthetic non-Gaussian datasets such as the wing nut and two diamonds datasets, courtesy of Alfred Ultsch [2], as shown below.



Wing Nut
No. of Input Patterns = 1070

Two Diamonds
No. of Input Patterns = 800

Figure 2: 2D Synthetic Non-Gaussian Datasets

2.3. Iris Dataset

Finally, we checked our algorithm against a practical 4D dataset - Fisher's Iris Data [3]. The properties of this dataset have been summarized below.

Table 1: Data Description of Fisher's Flower Iris Data Set

Origin of Data	
Source	UCI Machne Learnng Repository website
URL	http://archive.ics.uci.edu/ml/datasets/Iris
Data Characteristics	
Background	Multivariate data containing different classes of the Iris flower plant and introduced by Ronald Fisher in 1936
Associated problem	Classification
Attribute characteristics	Real numbers
Number of data points	150
Number of attributes	4
Number of classes	3
Number of labeled data points	150
Number of labeled samples per class	50
List of input/output features	
Input Features	
Sepal Length	[min, max] = [0,1], length of sepals, in cm, of the different classes of iris flowers
Sepal Width	[min, max] = [0,1], width of sepals, in cm, of the different classes of iris flowers
Petal Length	[min, max] = [0,1], length of petals, in cm, of the different classes of iris flowers
Petal Width	[min, max] = [0,1], width of petals, in cm, of the different classes of iris flowers
Output Classes	
Class Iris Setosa	
Class Iris Versicolour	
Class Iris Virginica	
Input/Output Encoding	
Input Encoding	No encoding is required since inputs are already numerical
Output Encoding	The three classes are encoded as three real numbers in $\mathcal{R}$. Class Setosa is encoded as 1, class Versicolor is encoded as 2 and class Virginica is encoded as 3

3. Objective

Demonstrate a computationally efficient generalized decrease schedule of learning rate and neighborhood radius based on the learning state of the network that reduces the number of learning steps required for the SOM to converge on various datasets.

4. Technical Approach

4.1. Embedding Error Function

In their paper about the convergence of SOMs, Yin and Allison [4] have proven that the prototypes of an SOM and the data points of an input space should exhibit the same distribution characteristics, assuming that the SOM lattice size and training period are sufficiently large. Using this idea and further assuming that each of the features of the input space are independent of each other, Lutz Hamel [5] has proposed a statistically efficient and computationally cheap metric to measure the embedding accuracy of the map. This metric uses various statistical tests to evaluate the differences between the mean and variance of the prototypes and the input patterns and check whether each feature has properly been embedded by the SOM prototypes. Inspired by this and the need for a more continuous-valued metric that checks the extent of embedding for each

feature, we propose the following embedding error function,

$$Embedding\ Error = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{\sigma_{p,i}^2(t)}{\sigma_{d,i}^2(t)}\right)$$

where $\sigma_{p,i}^2(t)$ is the variance of the i^{th} dimension of the prototypes at learning step t, $\sigma_{d,i}^2(t)$ is the variance of the i^{th} dimension of the input data at learning step t, N is the total number of dimensions of the input data. It should be noted that for the purposes of this study, we use the following criteria for weight initialization,

$$\mu_{p,i}(0) = \mu_{d,i}(0) \text{ and } \sigma_{p,i}(0) = \frac{1}{10}\sigma_{d,i}(0), \forall i \in \{1, ..., N\}$$

i.e. the initial mean of the i^{th} dimension of the prototypes is equal to the mean of the i^{th} dimension of the input patterns and the initial standard deviation of the i^{th} dimension of the prototypes is set to one-tenth of the standard deviation of the i^{th} dimension of the input patterns. The prototypes are initialized using a random uniform distribution in a small ball in $\mathcal{R}^N$, centered at $\mu_{\mathbf{p}}(\mathbf{0})$ and with a radius proportional to the $\frac{\sigma_{\mathbf{d,i}}(\mathbf{0})}{\mathbf{10}}$.

As has been illustrated in the figure below, as learning progresses, the value of the embedding error function drops from around 1.0 to around 0.20, which means that the prototypes have embedded about 80% of the variance of the input data.

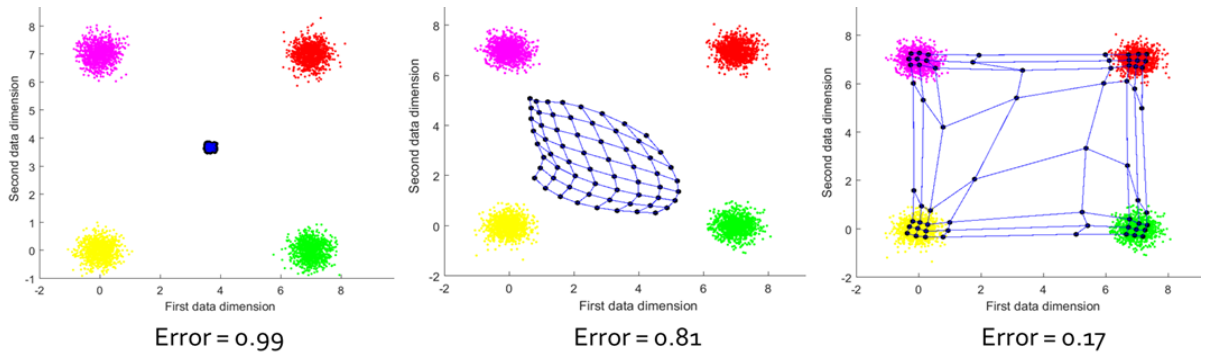


Figure 3: Embedding error at different learn steps during a run of the algorithm

4.2. Topological Error Function

Since the learning progress of an SOM depends not only on how well the prototypes are embedded in the input space but also on how similar the topological shape of the weight vectors are to the topological shape of the input patterns, we are able to better monitor the learning progress by coupling the embedding error function above with another metric that measures topological quality. Addressing this issue, Lutz Hamel describes the topological error in his paper as,

$$Topological\ Error = \frac{1}{n}\sum_{i=1}^{n} err(x_i)$$

with

$$err(x_i) = \begin{cases} 1 & \text{if } bmu(x_i) \text{ and } 2bmu(x_i) \text{ are not neighbors} \\ 0 & \text{otherwise} \end{cases}$$

for training data $x_1, x_2, ..., x_n$ where $bmu(x_i)$ and $2bmu(x_i)$ are the best matching and the 2nd best matching unit for the input pattern x_i on the map, respectively.

Since it is computationally expensive to evaluate the topological error function for all input patterns, for the purposes of our study, we select a random representative sample containing 10% of all input patterns.

The figure below further illustrates how the topological error function is evaluated.

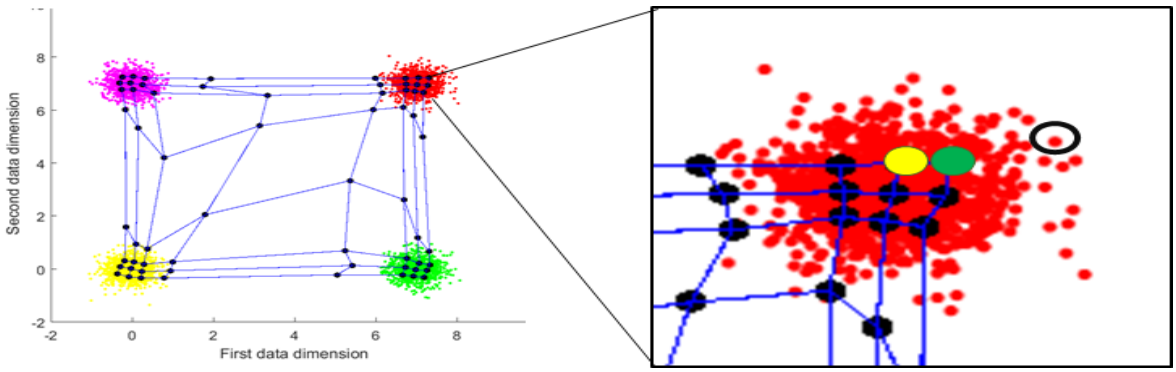


Figure 4: Topological error: For the data point, x_k (black circle), the BMU (green circle) and 2BMU (yellow circle) are neighbors, i.e. $err(x_k) = 0$.

4.3. Objective Function

Finally, using a convex combination of the two error functions defined above, we arrive at an objective function that accurately captures the learning progress of a self-organizing map. This objective function has been provided below,

$$E(t) = \alpha * Embedding\ Error + (1 - \alpha) * Topological\ Error$$

$$E(t) = \alpha * \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{\sigma_{p,i}^2(t)}{\sigma_{d,i}^2(t)} \right) + (1 - \alpha) * \frac{1}{n} \sum_{i=1}^{n} err(x_i)$$

where $0.0 \leq \alpha \leq 1.0$

When plotted, the objective function $E(t)$ has high frequency variations. These high frequency variations are due to many factors such as the random initialization of weights, relative distribution of the weights and data at the start of learning etc. Since we are interested in the overarching trend of the objective function, we smoothen the curve using a sliding window.

Using an α value of 0.5, an example graph of the objective function, $E(t)$ and the corresponding smoothened curve,$E_s(t)$ has been shown below.

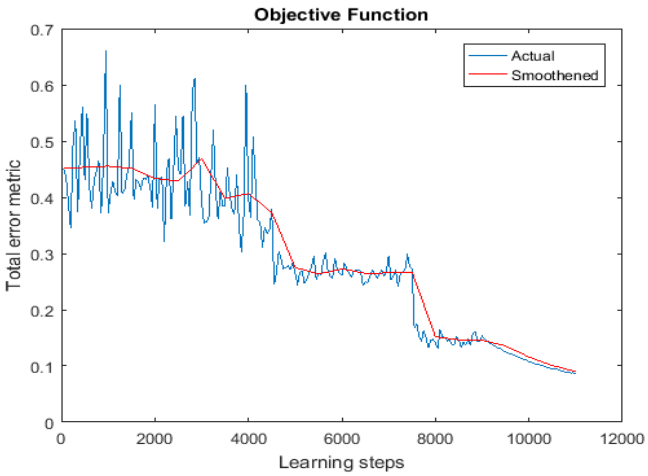


Figure 5: Actual vs Smoothened Curves of the Objective Function

4.4. Generalized Decrease Schedule

For generalised decrease schedule, the learning rate and neighborhood radius are decreased when the smooth objective function is either increasing or constant for last few learn steps. Specifically, if current value of $E_s(t)$ is more than 90% of mean value of $E_s(t)$, where the mean is evaluated over a window of past few learn steps, a decision is made to decrease the learning rate and neighborhood radius (see figure 6). The next decreased value of learning rate and neighborhood radius is dependent on initial and final values for these respective parameters. The initial and final values are required input arguments for our method (of course it is possible to provide default values for these arguments). The initial-to-final range is divided linearly to get a staircase decrease schedule. We also include a sanity check to force neighborhood radius to have a value of at least 1.

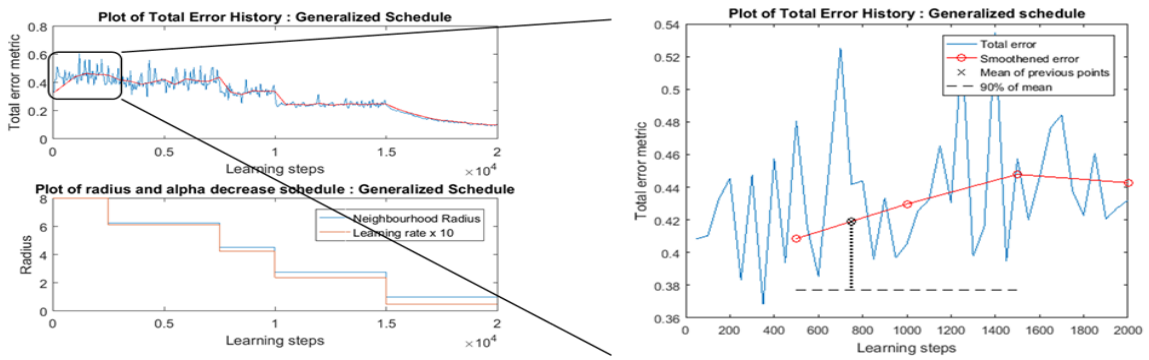


Figure 6: Generalised schedule: If the smooth objective function (shown in red), $E_s(t)$, is flat or increasing, learning rate and neighborhood radius are decreased. On right, the dashed line is 90% of mean value of $E_s(t)$. The mean is evaluated over past 1500 learn steps.

5. Results

5.1. SOM Learning Parameters

The following learning parameters were used for the 8 datasets that our algorithm was tested against.

Table 2: SOM parameters for 2D Gaussian datasets and Iris dataset

Network parameters	
Topology	(2) — (8×8 PE Lattice) for 2D Gaussian datasets (4) — (10×10 PE Lattice) for Iris dataset
Learning parameters	
Initial radius	4 (2D Gaussian), 5 (Iris dataset)
Initial weights	drawn uniformly from [0.0, 1.0]
Learning rate (α)	0.40(initial) and 0.01(final).
Batch size (K)	1 (online learning)
Neighborhood Fn.	Gaussian function: $e^{-\frac{d^2}{\sigma^2}}$ with Manhattan distance
Radius schedule	dynamically determined/hardwired base line
Learning rate schedule	dynamically determined/hardwired base line
Parameters and error measures for performance evaluation	
Monitoring freq. (m)	50 learning steps
Convergence criteria	$E_s(t) \leq 0.1$

5.2. Baseline Decrease Schedule

In order to determine the effectiveness of the generalized schedule, we compare it to a hard wired schedule (baseline) for various datasets. The hard wired schedule (shown below) was empirically optimized using a trial and error method to work for the homework problem (Four Gaussians: Distinct). As can be seen below, the baseline decrease schedule has all the desired properties, i.e. it is stepwise in shape and has an exponentially decaying descent.

5.3. Graphical Results

5.3.1. Decrease Schedules for Select Datasets

In this subsection we compare the hard-wired and generalized decrease schedules of two select datasets, i.e. the distinct four Gaussians dataset and the Iris dataset.

The trajectories of the objective function and the hard-wired and generalized decrease schedules for the distinct four Gaussians dataset has been provided below.

The trajectories of the objective function and the hard-wired and generalized decrease schedules for the Iris dataset has been provided below.

As can be observed from the decrease schedules and objective function graphs above, the generalized schedule reaches the convergence criteria of $E_s(t) \leq 0.1$ faster than the hard-wired schedule for both the distinct four Gaussians and the Iris datasets. It is interesting to note that even though the hard-wired decrease schedules have an exponentially

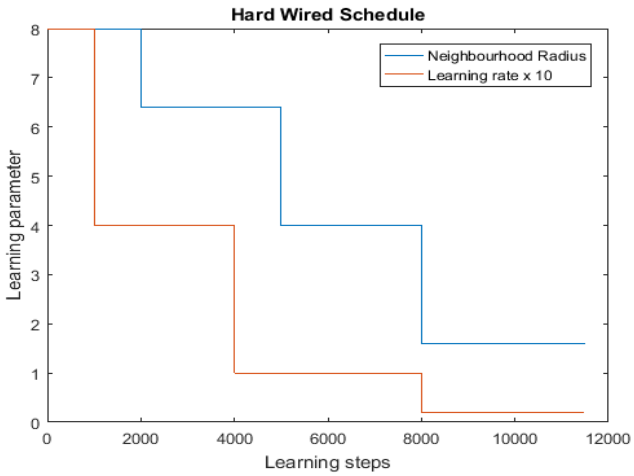
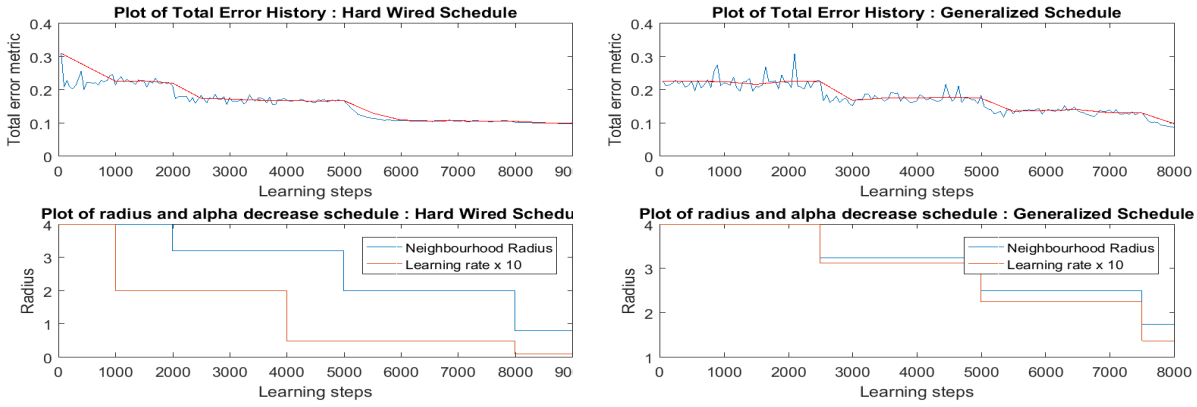


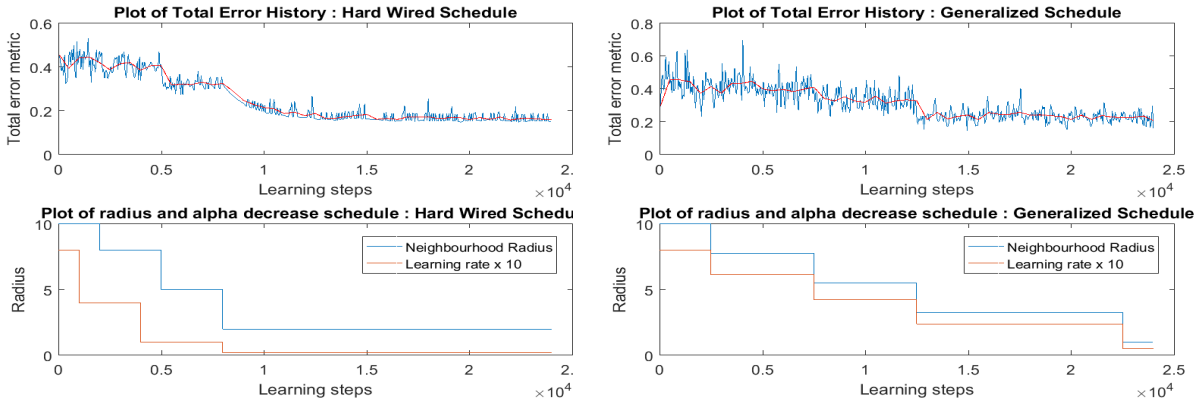
Figure 7: Hard Wired schedule - Learning rate step values approximate trace an exponentially decaying curve. The schedule is hardcoded to run the same parameters independent of the total number of learning steps



Four Gaussians - Hard Wired

Four Gaussians - Generalized

Figure 10: Decrease Schedules for Distinct Four Gaussians Dataset



Iris - Hard Wired

Iris - Generalized

Figure 13: Decrease Schedules for Iris Dataset

decaying shape and the generalized one has a fairly linear decay, the generalized schedule outperforms the hard-wired one. Moreover, for the Iris dataset, the while both schedules are exponentially decaying, the generalized schedule is less steep in its decay. All of this suggests that the shape of the decrease schedule need not be exponentially decaying for best performance, but may be of other shapes too. In fact, the optimal decay shape depends on the particular input data that the SOM is trying to represent.

The number of learning steps taken to converge to the same error ($E_s(t)$) tolerance of 0.1 was compared for 100 runs with both Generalized and Hard Wired decrease schedules. The plots displaying the mean and the standard deviation for the 100 runs on each dataset are shown below.

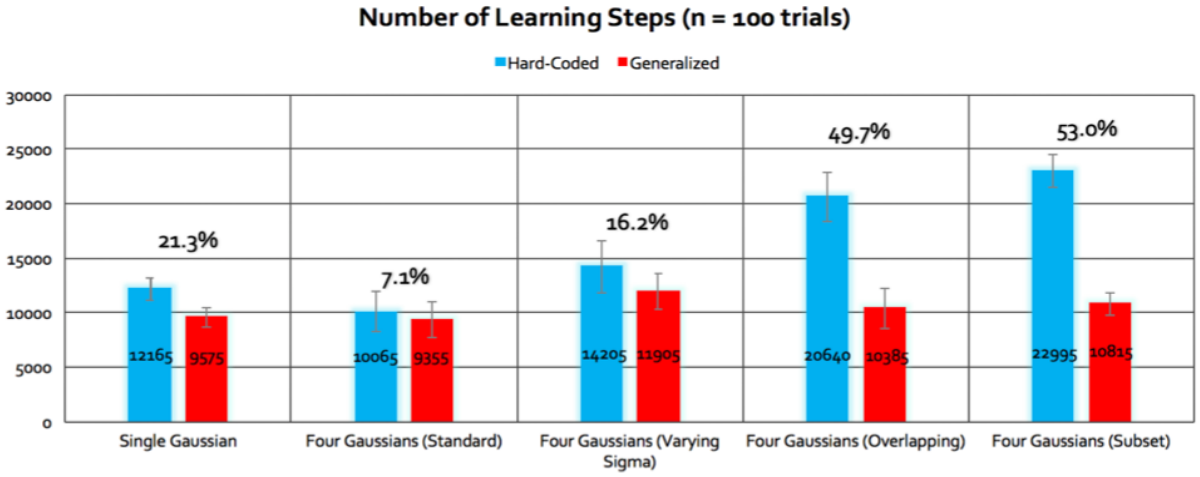


Figure 14: Results of Gaussian Datasets

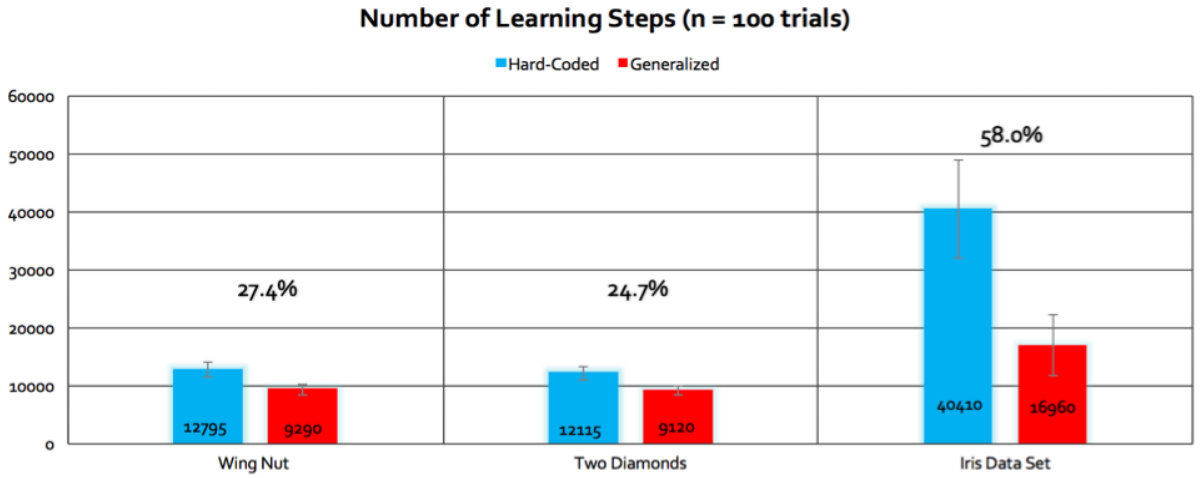


Figure 15: Results of Non-Gaussian Datasets

5.4. Discussion of Results

From the plots, we can conclude that the generalized schedule performs better than the base line schedule. We observed that the base line schedule is not only slower but also

takes twice the number of learning steps for convergence. Moreover, the generalized schedule rarely gives a twisted map and has less variance for learn steps needed for convergence which means it converges more reliably than the hardwired schedule.

Comparison of the decrease schedule plots indicates that the slower learning in case of the hardwired schedule is due to the parameters staying constant at the minimum value for a long time. This is avoided in the generalized schedule.

6. Conclusion

We can draw several useful conclusions as a result of this project. Since we have used our proposed objective function to successfully determine a decrease schedule for the learning rate and neighborhood radius of our self-organizing map, we can conclude that comparison of the mean and variance of input patterns and prototypes is a good indicator of the learning progress of an SOM. Moreover, the objective function can be used to obtain a default decrease schedule for these learning parameters without resorting to a trial and error method to find an optimized schedule. Most importantly, we showed that the generalized decrease schedule causes the SOM to converge significantly faster than the base line decrease schedule, on 8 different datasets, including the real-life Fisher's Iris Dataset.

Further investigation on this method could be directed towards making the learning process robust to twisting, which is indicated by topological error. Traditional method is to restart learning when the lattice twists. We would like to investigate a method that stores past prototype history and backtracks when the lattice twists.

References

[1] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[2] A Ultsch. Clustering with SOM: U*c. In *Proc. Workshop on Self-Organizing Maps*, pages 75–82, Paris, France, 2005. https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/data [Online; accessed 4-April-2017].

[3] R.A Fisher. UCI machine learning repository: Iris data set. https://archive.ics.uci.edu/ml/datasets/Iris, 1936. [Online; accessed 4-April-2017].

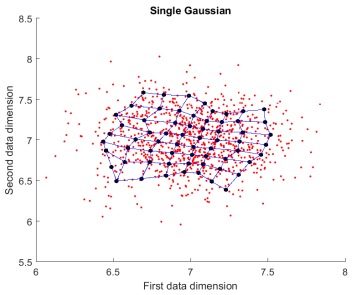
[4] H. Yin and Allison N.M. On the distribution and convergence of feature space in self-organizing maps. *Neural Computation*, 7(6):1178–1187, 1999.

[5] L. Hamel. SOM quality measures: An efficient statistical approach. In *Advances in Self-Organizing Maps and Learning Vector Quantization*, volume 428 of *Advances in Intelligent Systems and Computing*. Springer, 2016.

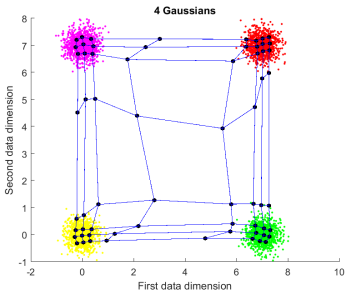
A. Supplementary Figures

A.1. Converged SOM plots

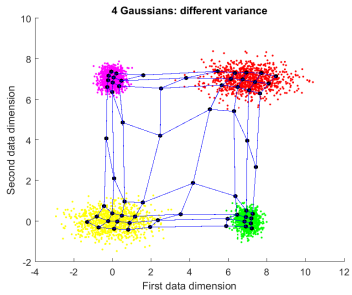
Figures showing representative plots of the final converged prototypes in the data space for each dataset using the Generalized decrease schedule. Notice the topological ordering in every case.



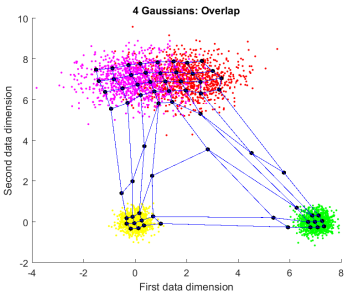
Single Gaussian



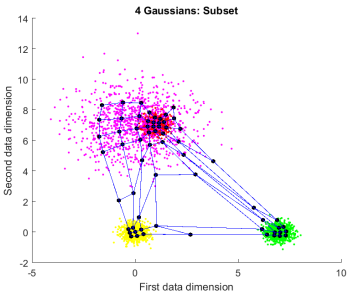
Four Gaussians - Distinct



Four Gaussians - Different Variance

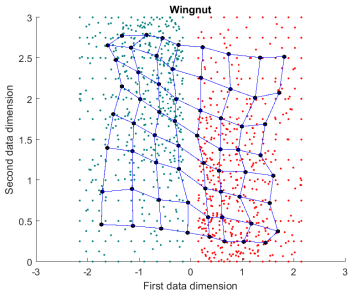


Four Gaussians - Partially Overlapping

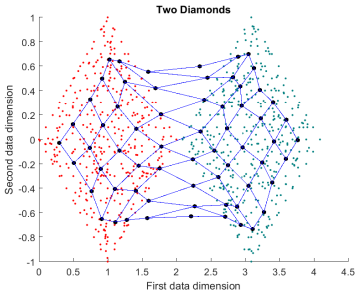


Four Gaussians - Entirely Overlapping

Figure A.1: 2D Synthetic Gaussian Datasets



Wing Nut



Two Diamonds

Figure A.2: 2D Synthetic Non-Gaussian Datasets

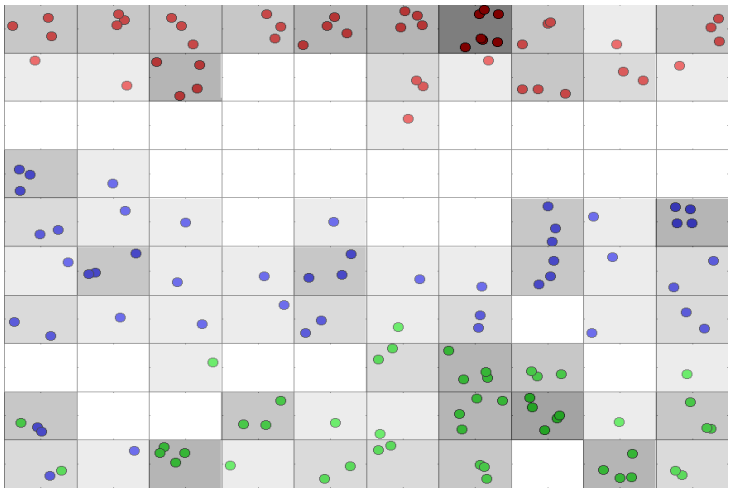


Figure A.3: Density plot for Iris dataset