LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-791165

Self-Organizing Maps and Their Applications to Data Analysis

R. Ponmalai, C. Kamath

September 24, 2019

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Self-Organizing Maps and Their Applications to Data Analysis

Ravi Ponmalai and Chandrika Kamath
Lawerence Livermore National Laboratory

September 20, 2019

Abstract

Self-Organizing Maps(SOMs) are a form of unsupervised neural network that are used for visualization and exploratory data analysis of high dimensional datasets. Our goal was to understand how we can use a SOM to gain insights about datasets. We do this by first understanding the initialization, training, error metrics, and convergence properties of the SOM. Next we discuss the ways to interpret and visualize a Self-Organizing Map. Finally we used real datasets to understand what the Self-Organizing Map can tell us about labeled and unlabeled data. Based on experiments with our datasets we found that the Self-Organizing Map can tell us about the spacing and position of high dimensional clusters, help us find non-linear patterns, and give us insight into the shape of our data.

Contents

1 What Is a Self-Organizing Map and When Should It Be Used?

Analyzing high dimensional data can be very difficult because we can only visualize data in 2 or 3 dimensions. Before and after training machine learning models on large high-dimensional datasets, it is desirable to have some intuitive understanding of the relationships and patterns in a dataset to guide the learning process and to help interpret results. The Self-Organizing Map(SOM)[11] is a dimensionality reduction technique that can give us insights about high dimensional data with minimal required computing. Self-Organizing Maps can be used for exploratory data analysis, clustering problems, and visualization of high dimensional datasets. SOMs were invented by Teuvo Kohonen in 1980 and have been the subject of many research papers. SOMs are built using a pre-defined 2-D lattice of nodes. This lattice of nodes has a structure that is defined by giving each node a location in $\mathbb{R}^2$, represented as a vector l_i where i is the index of that node. A visual of the lattice space can be seen in Figure 1 on the left. Each node in the lattice also has a position in the input data space, represented as a vector $w_i \in \mathbb{R}^d$ where d is the dimension of the input data. Each lattice node is a connection between $\mathbb{R}^d$ and $\mathbb{R}^2$. Figure 1 shows how the structure of the lattice is preserved in a higher dimensional space.

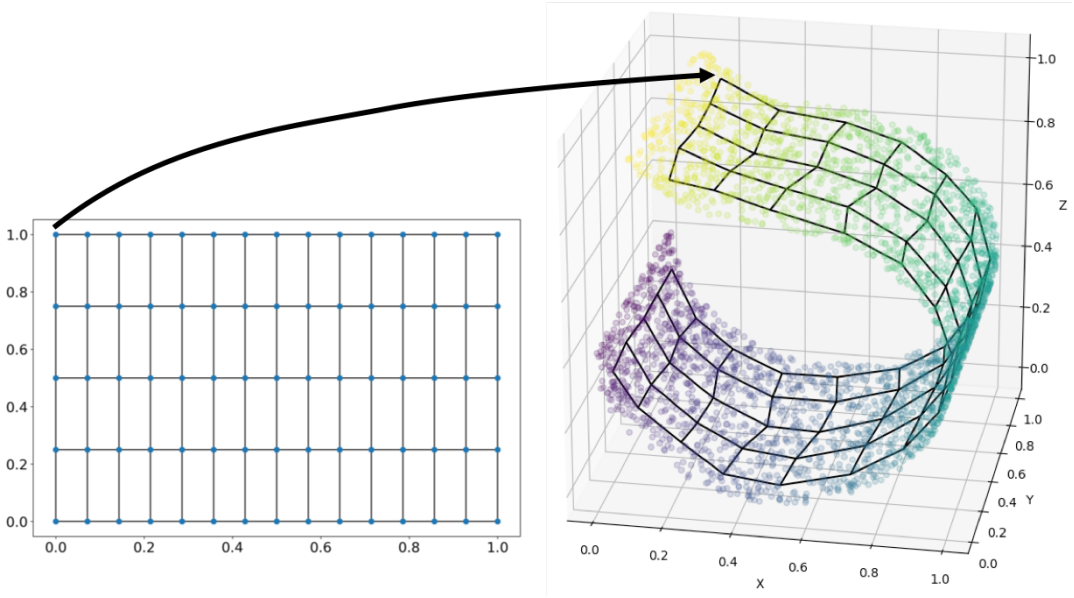


Figure 1: On the left: A plot of a 15×5 lattice in the lattice space, $\mathbb{R}^2$, the black lines indicate neighbors and the blue points indicate the location of the node in lattice space. On the right: a plot in the input data space, $\mathbb{R}^3$, of a SOM trained with the lattice on the left, fit to 3-D data that has a non-linear relationship. The circles are data points, the color of each data point is its position along a non-linear manifold. The arrow shows the connection between a node's lattice vector at $(1, 0)$ in the lattice space and weight vector at $(0.17, 0.88, 0.94)$ in the data space.

The driving concept behind the self-organization property of a SOM is that if two nodes are near each other in the lattice space, then they should be near each other in the input data space.

A SOM is trained by updating the locations of the weight vectors. The goal of this update step is to move the weight vectors to be representatives of the centers of clusters in the data and to mirror the order of the lattice. In Figure 2 we demonstrate the training of a SOM on uniform random 2-D data. In practice this is not useful because 2-D data has no need for dimensionality reduction, however it is useful to illustrate the self-organizing property of the SOM. Figure 2 shows how a SOM unfolds and then stretches to fit the shape of the data.

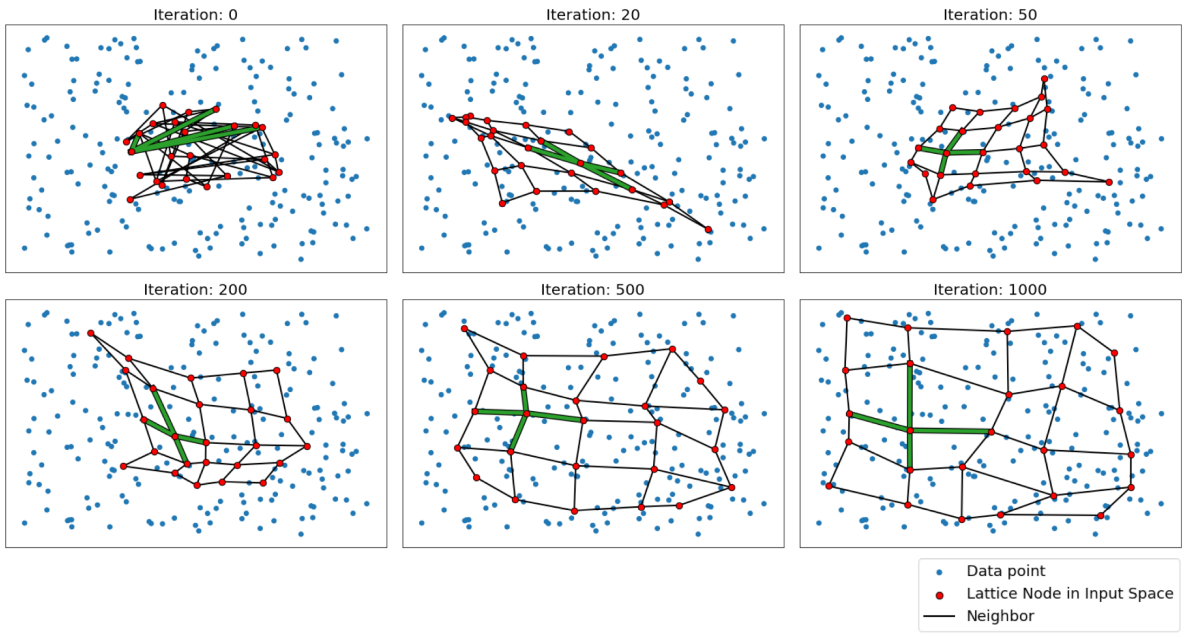


Figure 2: The training of a 5×5 SOM on uniform random points for 2000 iterations. A more in-depth look at SOM training can be found in Section 3. The green segments highlight the neighbors of one lattice node.

As seen in Iteration 0 of Figure 2, the weight vectors of a SOM can start out with random values. As the weight vector of each lattice node moves, its neighbors move with it. This is especially evident in Iteration 20 of Figure 2, where we can see that the lattice node on the bottom right was pulled towards points on the bottom right and that the neighbors of that node also moved towards that point. The amount that a neighbor of a node moves is controlled by a function called the neighborhood function and it is proportional to the distance of the two nodes in the lattice space as seen in Figure 4. A general idea of how the SOM is trained can be seen in Algorithm 1.

Algorithm 1: Main Idea Behind the SOM Training Algorithm

initialize lattice nodes;
initialize weight vectors;
$N \leftarrow$ Iteration count;
for $i \leftarrow 1$ **to** N **do**
    $x \leftarrow$ pick a random point in the dataset;
    $c \leftarrow$ the lattice node closest to x;
    move the weight vector of c closer to x;
    move the weight vectors of the neighbors of c slightly closer to x;
end

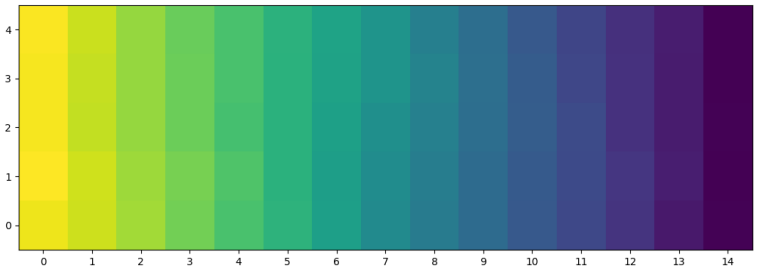


Figure 3: A visualization of the average color of each node in the lattice space of the 15×5 SOM shown in Figure 1, based on the nearest data points to each node.

We can use a trained SOM to understand how different clusters of data are distributed through a data space or how a response variable changes across a data set. For example if we take the SOM in Figure 1 and for each lattice node in the SOM, we plot the average output value(represented by color) of all the data points nearest to that node in the input space we get a plot like Figure 3. More examples of applications of a SOM can be found in Section 6 and the appendix.

Throughout this report we will use a few datasets of interest to demonstrate the practical use of a SOM. We will summarize those data sets here.

- Fischer's Iris Dataset[3] - This dataset is a classic dataset in statistics and machine learning. The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. This is a 4 dimensional dataset.
- Wine Dataset[3] - This dataset describes certain chemical properties of wines. There are 3 classes, each corresponding to a different type of wine. Each of the 178 data points has 13 continuous features. This dataset is very easily separable which makes for an easy demonstration of the SOMs exploratory data analysis capabilities.
- ET462/ET5000 Dataset- The ET462 dataset [8] is a 4 dimensional dataset of additive manufacturing simulation data generated from the Eagar-Tsai Model. The dataset consists of 462 samples each with a continuous response variable corresponding to depth of the melt pool. Using Gaussian Process Regression trained on ET462 and applied to best

candidate sampled points, a new 5000 sample dataset was created[5]. This dataset is of particular interest because we are interested in the parameter space that leads to an ideal melt pool depth.

- BPA Wind Energy Dataset [7] - Each data point in this dataset represents wind data from one day at a wind farm. Wind was sampled once an hour per day so each data point is 24 dimensional. The data was collected over five years(2007-2011), which is 1826 days with 6 days removed because of missing data. It is an unlabeled dataset that we are using for unsupervised learning to try and find daily patterns in wind generation.
- Uniform2D - This is a dataset of 1000 2-D points sampled from a uniform distribution.
- Normal2D - This is a dataset of 1000 2-D points sampled from a multivariate normal distribution with zero mean in each dimension and an identity covariance matrix.
- MAGIC Gamma Telescope Dataset[3]- This is a two class dataset with 19020 data points, each with 10 dimensions. This dataset is from a simulation of high energy gamma particles hitting an atmospheric Cherenkov telescope.

This report is organized as follows. Section 2 contains information about everything that is done before training a SOM. This includes weight vector initialization methods, lattice structure and size. Section 3 contains information about different methods to train a SOM. This section also discusses the time it takes to train a SOM and neighborhood function choices. Once a SOM is trained, we need to evaluate it, Section 4 describes different error metrics used to determine how well a SOM fits a dataset. Section 5 explains the different methods used to visualize the output of a SOM and also how to interpret those visualizations. Section 6 demonstrates the use of a SOM on real datasets and gives some advice on training a SOM. Finally Appendix A contains images of trained SOMs and output visualizations to give the reader an idea of where SOMs can be used and Appendix B contains some examples of using SOMs to analyze toy datasets.

2 Building a Self-Organizing Map

Lattice Structure Both rectangular and hexagonal grids have been used to define the structure of the lattice. Previous literature[11, 13] has stated that the hexagonal grid is better for visualization and can better fit clusters of data. We believe that hexagonal better fits data clusters because the hexagonal lattice has 6 nearest neighbors whereas the rectangular one only has 4, to see this compare the rectangular and hexagonal connections of (1,1) in Figure 4. This would result in a higher density of weight vectors in areas with high data point density.

Lattice Size The number of nodes in a SOM lattice can have a huge effect on the resulting map. Kohonen's [11] advice is to use $5\sqrt{n}$ nodes where n is the number of data points. If a rectangular map is used, Kohonen also suggests that the optimal ratio of height to width of the lattice is equal to the ratio of the two largest eigenvalues of the autocorrelation matrix.

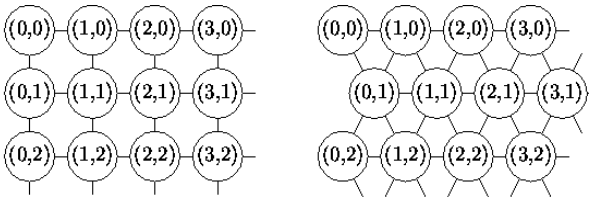


Figure 4: Rectangular(left) and Hexagonal(right) Lattice structures. The numbers in each node indicate that node's position in lattice space and the edges connect lattice nodes that are neighbors.

2.1 Weight Vector Initialization Methods

The initialization of the weight vectors of a Self-Organizing Map have little affect over a long training period. However in the interest of time, it is desirable to choose an initialization strategy that will find an equilibrium quickly.

Random Initialization The weight vectors of Self-Organizing Maps can be initialized with random values. Doing so demonstrates the organizing power of the SOM when starting from an arbitrary initial state. Random initialization can be slow to converge.

Random Sampling Initialization Another simple option for initialization is using random data points from the input data as values for the initial weight vectors[6]. The advantage of this method is that the distribution of the initial values of the SOM will match the distribution of the data. This method is thought to provide a faster convergence than true random initialization.

Best Candidate Sampling Initialization The best candidate sampling algorithm places samples randomly while trying to maximize the distance between samples. It has been used to generate subsets of data with low bias and even spacing[8]. The method works by first picking one point randomly from the input dataset, Then, for each of the remaining weight vectors a pre-specified number of random points are sampled from the input dataset. The distance of the nearest neighbor for each point in the sample is calculated. The point with the largest nearest neighbor distance is chosen as the value of the current weight vector. This is repeated until all the weight vectors are assigned a value.

Principal Component Initialization Principal Component Initialization, or Linear Initialization can be a very useful initialization method for the SOM. In many cases this method will require fewer training iterations compared to other methods[1]. Before initializing the weight vectors we calculate the eigenvectors of the covariance matrix of the data. Then for the two largest eigenvalues, the corresponding eigenvectors are taken. The values of the weight vectors are then computed by creating a rectangular array along this subspace of the data. Using this method, the initial weight vectors start already ordered. It is recommended to use a small learning rate and neighborhood function for training a SOM initialized in this way. Even if the data we are modeling does not have a linear relationship, PCI can still be effective because it starts the weight vectors in the center of the data in an organized fashion.

Initialization Experiments Empirical tests were done to assess the effect each initialization method has on the speed of convergence. For each initialization method hyperparameters such as the neighborhood radius over time were chosen to be optimal for that method. More information about the selection of hyperparameters will be discussed in section 3.3. Each test computed the average number of iterations until the SOM reached equilibrium over 10 runs. Equilibrium and SOM training are discussed in Section 3. We can see that on most datasets PCI finds equilibrium in the least number of iterations. It should also be noted that the time per iteration is independent of the initialization method so total training time is proportional to iteration count.

Dataset	Random Initialization	Random Sampling	Best Candidate Sampling	Principal Component Initialization
ET5000	19.8	**14.8**	17.8	17
Normal2D	13.2	14.6	14.8	**10**
MAGIC	19	14.6	15.4	**10**
Iris	11.2	9.2	10.4	**7**
Uniform2D	14.6	16.4	17.8	**14**

Table 1: The average number of batch iterations needed for convergence of a given dataset and weight initialization method. Fastest initializations are in bold.

3 Training a Self-Organizing Map

3.1 Data Preprocessing

Input data should be rescaled to ensure each feature has the same scale. In practice we have found it simplest to rescale each feature to be between 0 and 1. Outliers that are not representative of the dataset should also be removed.

3.2 Training Step

A Self-Organizing Map is initialized with a lattice of nodes. Each node has a position in the input space and a position in the lattice space. These positions are represented by two vectors which we will call the weight vector and lattice vector respectively. We will denote the weight vector as w_i and the lattice vector as $l_i,\ i \in 1, 2, 3\ ...\ n$. n is the number of nodes in the lattice and i is the node index. We also denote D as the set of input data vectors.

3.2.1 Online Update Step

The two most common ways to train a SOM are using an online update step or batch update step. The main difference is that for each iteration the online update step uses one random data point and the batch update step considers the entire dataset. In the online update step each iteration considers one randomly sampled point from the input dataset. Let that point be denoted as $x \in \mathbb{R}^d$. Note that d is the dimension of the input data space. First the best matching unit(BMU) of that point is computed. The best matching unit of a data point x is defined as the node that is closest to the data point in the input space. Let $c(x)$ be the index of the best matching unit of the point x.

$$c(x) = \arg\min_i \|x - w_i\|$$

Once the BMU is computed, we can calculate the updated value of the weight vectors for the $k+1$-th iteration.

$$w_i(k+1) = w_i(k) + \alpha(k) h_{i,c(x)} * (x - w_i(k))$$

Here we have the neighborhood function represented as $h_{i,c(x)}$. This is a function of distance in the lattice space from the best matching unit of x, $c(x)$, to the i-th lattice node. Suitable choices and conditions for h will be discussed in the next section. We also have the learning rate, $\alpha(k)$, included in the update function. The conditions for $\alpha(k)$ are defined in [10] as follows:

$$\alpha(k+1) \leq \alpha(k)$$
$$\sum_{k=0}^{\infty} \alpha(k) = +\infty$$
$$\sum_{k=0}^{\infty} \alpha(k)^2 < +\infty$$

In each iteration, this update function is applied to each of the nodes in the lattice.

3.2.2 Batch Update Step

The batch update step is somewhat similar to the online update step except instead of using a single data point for each iteration, it uses the entire dataset. To compute it, first two things must be calculated. Given the current $w_i(k)$ for $i \in 1, 2, 3 \ ... \ n$ we must first compute the BMU for all data points. Also for a given node j we also want to compute the number of data points that have node j as their BMU. We denote this value to be n_j.

$$n_j = \sum_{x \in D} \left(\begin{cases} 1 & c(x) = j \\ 0 & \text{otherwise} \end{cases} \right)$$

The batch update step in its final form is:

$$w_i = \frac{\sum_{j=1}^{n} n_j h_{j,i} \bar{x}_j}{\sum_{j=1}^{n} n_j h_{j,i}}$$

In this instance $\bar{x}_j$ is the average of all data points closest to the j-th lattice node, in the input space. Also note that $h_{i,j}$ is the neighborhood distance from node i to node j. Note that the batch update step does not require a learning rate. In our experiments we have noticed that even without a learning rate the Batch SOM will find an equilibrium. We determine the SOM reaches equilibrium by computing the average displacement of the nodes from the previous iteration. Once the average movement of the nodes goes below some ϵ we determine it to be finished training. In our experiments we used $\epsilon = 10^{-4}$. It should be noted that the average movement will always go to zero eventually.

3.2.3 Batch versus Online

As noted in the previous section, the batch update does not require a learning rate function. This is advantageous because it eliminates the number of needed parameters. The other advantage of batch computation is that it is completely deterministic. Some authors[2] have proposed that the stochastic property of the online update step can allow for improved stability. They also found that the samples on which the SOM is trained is far more influential to the final result, compared to the initial conditions. Kohonen [12] states that only the batch version of the SOM is recommendable for practical applications because of the aforementioned lack of a learning rate parameter and its convergence is an order of magnitude safer and faster. In this report all of our experiments are done with batch computation.

3.3 Neighborhood Function Choices

The neighborhood function, $h_{i,j}$, must be chosen well because this function is what creates the connection between the input space and the lattice space. This connection is what creates the self-organizing property of the map. Our notation denotes the neighborhood distance of a winning node j to node i as $h_{i,j}$. As stated in [6, 11] the requirements for a neighborhood function are:

- $h_{i,j}$ should attain its maximum when $\|l_i - l_j\| = 0$ and be symmetric about the winning node j.
- As $\|l_i - l_j\|$ increases, $h_{i,j}$ should decrease.

$$h_{i,j} \to 0 \text{ as } \|l_i - l_j\| \to \infty$$

Any function that meets these requirements is a valid neighborhood function, below are some examples. These functions are visualized in Figure 5.

- Gaussian - $h_{i,c(x)} = \exp\left(-\frac{\|l_i - l_{c(x)}\|}{2\sigma(k)^2}\right)$
- Step/Bubble - $h_{i,c(x)} = \begin{cases} 1 & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$
- Triangle - $h_{i,c(x)} = \begin{cases} 1 - \frac{\|l_i - l_{c(x)}\|}{\sigma(k)} & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$
- Gaussian with cutoff - $h_{i,c(x)} = \begin{cases} \exp\left(-\frac{\|l_i - l_{c(x)}\|}{2\sigma(k)^2}\right) & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$

Effect of Varying Neighborhood Functions Because a majority of the training iterations have a neighborhood with 1 or no neighbors, the choice of neighborhood function is almost inconsequential. In the organizing phase, as long as the radius is of appropriate size, it is likely that the map will self organize. Once the convergence phase has started, it does not matter what the neighborhood function is, as long as the neighborhood size only captures one or zero neighbors of a given node.

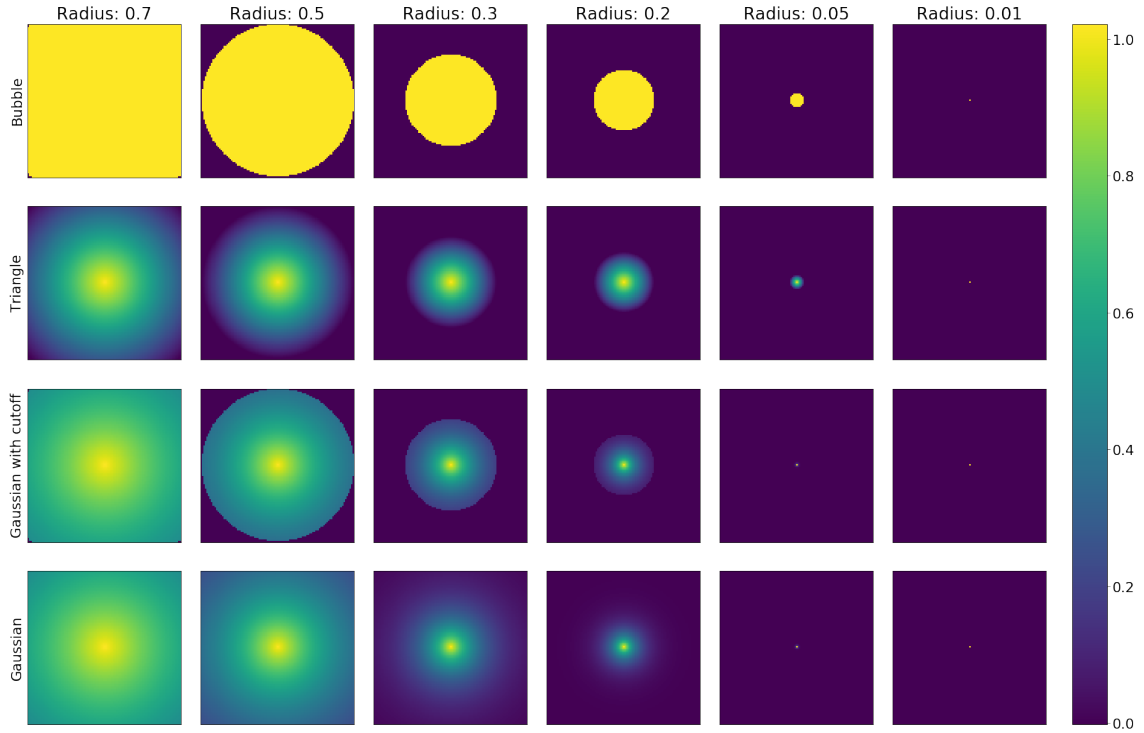


Figure 5: Here each of the previously discussed neighborhood functions are shown when measuring lattice space distance from a node in the center of the lattice

3.4 Training Time of a Self-Organizing Map

It is clear to see that both the batch and online implementations of the SOM training step take linear time relative to the number of iterations. Experiments were done to see how the number of lattice nodes, dimensionality of the dataset, and number of data points affected the computation time of the batch update step. In each experiment the variable of interest was changed and all other parameters of the network were kept constant. Since the execution time of the update step is unaffected by what the data is, all of these experiments were done with uniform random data.

Number of Lattice Nodes A SOM of varying lattice size was trained on a 2-D dataset of 100 uniform random samples for 100 iterations. The number of lattice nodes in each trial ranged from 4 to 3600. Figure 6 shows us that there is a quadratic relationship between an increase in the number of nodes and training time.

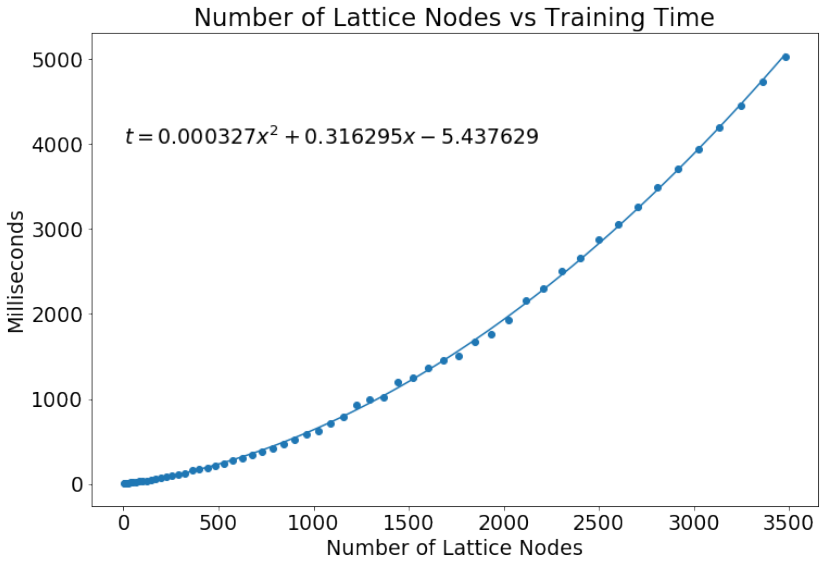


Figure 6: Training time for 100 iterations of SOMs with varying numbers of lattice nodes. A quadratic was fit to the data and the equation of the quadratic is shown.

Input Dataset Dimension A SOM of lattice size 10×10 was trained on 1000 uniform random samples for 100 iterations. The dimension of those uniform random samples ranged from 2 to 50. Figure 7 shows us that there is a linear relationship between an increase in the dimension of the input dataset and the training time.

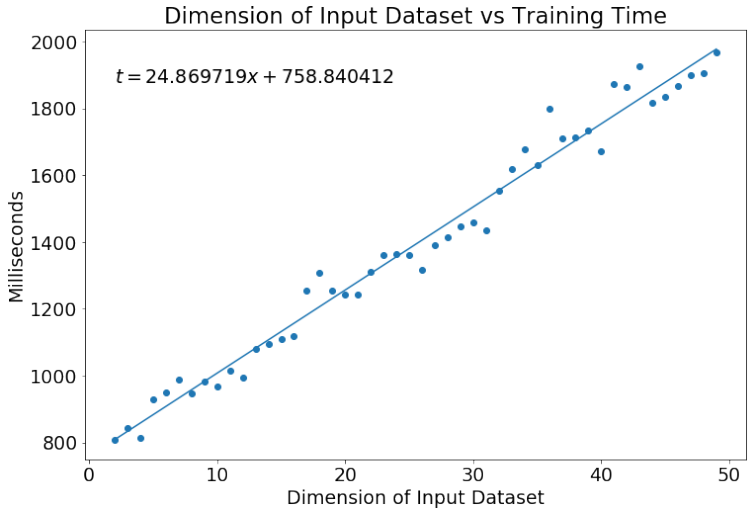


Figure 7: Training time for 100 iterations of 4×4 SOMs on datasets with different input dataset dimensions

Number of Data Points A SOM of lattice size 10×10 was trained on 2-D uniform random samples for 100 iterations. The number of sample ranged from 10000 to 20000.

Figure 8: Empirical results show us that an increase in the number of data points results in a linear increase in training time.

These timing studies lead us to believe that a batch update is of order $O(ndl^2)$ where n is the size of the dataset, d is the dimension of the input dataset, and l is the number of lattice nodes.

We also provide total time for SOMs of various parameters to give a general idea of the length of time it takes for a SOM to train. All of the following timings were collected from 100 batch iterations.

Number of Lattice Nodes	10000 Data Points		20000 Data Points	
	2-D Data	20-D Data	2-D Data	20-D Data
4	2.8337	3.1760	5.9169	6.1413
100	4.0901	6.7627	7.9115	16.5943

Table 2: Time in seconds for 100 batch training iterations of a SOM and dataset with the given parameters.

4 Evaluating a Self-Organizing Map

In higher dimensions it is very difficult to know that a map that has found equilibrium is accurately representing the shape of the dataset. Because of this it can be very difficult to evaluate the quality of a mapping. In order to evaluate a trained SOM we need to know what constitutes a good mapping. A well fitted SOM should preserve the topology of the data, fit

the data well, and avoid modeling noise.

4.1 Error Metrics

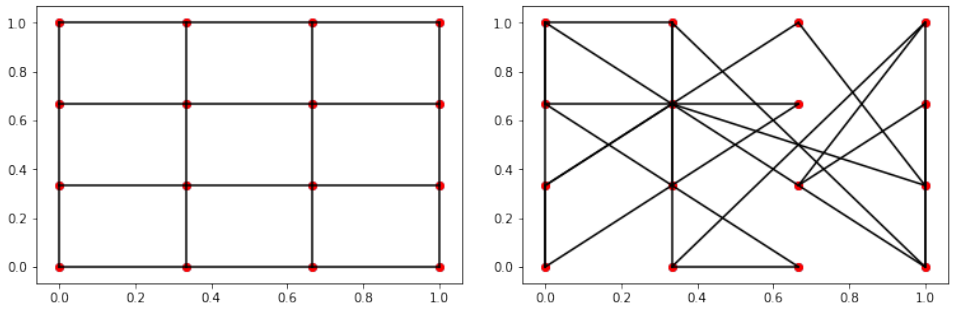
Error metrics for Self-Organizing Maps are essential for evaluating a SOM. There are a wide variety of error metrics designed for SOMs and they can all measure something different about the SOM. Despite the usefulness of these metrics it has been proved [4] that SOM learning dynamics cannot be described by the gradient of a single energy function. In practice this means that no single metric will tell us everything we need to know about how well a SOM models data.

4.1.1 Quantization Error

Quantization Error(E_Q)[11] is the average distance of data point to the nearest lattice node. Let $c = \arg\min_i \|x - w_i\|$.

$$E_Q = \sum_i \|x_i - w_c\|^2$$

E_Q can measure how well the mapping fits the distribution of the data. It does not measure if the lattice fits the topology or shape of the data. This can be easily proved. Consider a SOM that has an equal number of data points and nodes where the weight vector of each node is equal to the value of a unique data point. This is shown in Figure 9 where the red points are data points and the line segments represent neighborhood connections. This theoretical example has $E_Q = 0$, if we took the same mapping and permuted the values of the weight vectors we would still have $E_Q = 0$. This is because E_Q does not measure how well the map preserves the topology of the data.



(a) A perfect mapping (b) Same mapping, shuffled nodes

Figure 9: Because every point in the dataset has a lattice node directly on it, both maps have $E_Q = 0$. The issue with (b) is that it is not an organized map and the lack of organization is not represented by E_Q.

Although E_Q does not measure topology preservation, it does give a good idea of how well the map fits the data. This is a good metric if we are trying to determine if the map has nodes close to dense clusters. It should be noted that just like in K-Means clustering, adding more nodes to the lattice would result in a lower final E_Q. Adding more nodes to get a lower E_Q

could start to overfit the data, especially if the ratio of nodes to data points to lattice nodes is close to or below 1.

4.1.2 Topographic Error

Topographic Error[9] aims to capture information that Quantization Error cannot. It measures how well the shape of the data is preserved in output space. Topographic error in its most simplest form is computed by calculating the proportion of data points where the best matching unit and the second best matching unit are neighbors in the lattice space. Let c_i be equal to the index of the i-th best matching unit and let D be the input dataset.

$$E_T = \frac{1}{|D|} \sum_{x \in D} te(x)$$

$$te(x) = \begin{cases} 1 & c_1(x) \text{ and } c_2(x) \text{ are neighbors.} \\ 0 & \text{otherwise.} \end{cases}$$

Topographic error can give a general idea of how well the map preserves the topology of the data. It can be improved by having a more fine grain measurement of how well topology is preserved. This could be implemented by changing the te function to return the shortest path distance from the first and second best matching units as opposed to just 0 if the first and second BMUs are not neighbors.

4.1.3 Distribution Matching Error

It has been proposed that a SOM is completely embedded in a dataset if its weight vectors appear to be drawn from the same distribution as the input data[15]. The Distribution Matching Error(DME) is a metric that attempts to measure this. For each dimension of the input data, a two sample Kolmogrov-Smirnov(KS) test is performed. The two sample KS test can be used to determine if two samples are drawn from the same distribution. One advantage of the KS test for this application is that the KS test does not require the two compared samples to be the same size. It also does not require the distribution that the data is drawn from to be known. The DME is computed by calculating the proportion of dimensions where the two sample KS test fails. The bottom row of Figure 10 shows us what the distributions of a SOM with a low DME would look like.

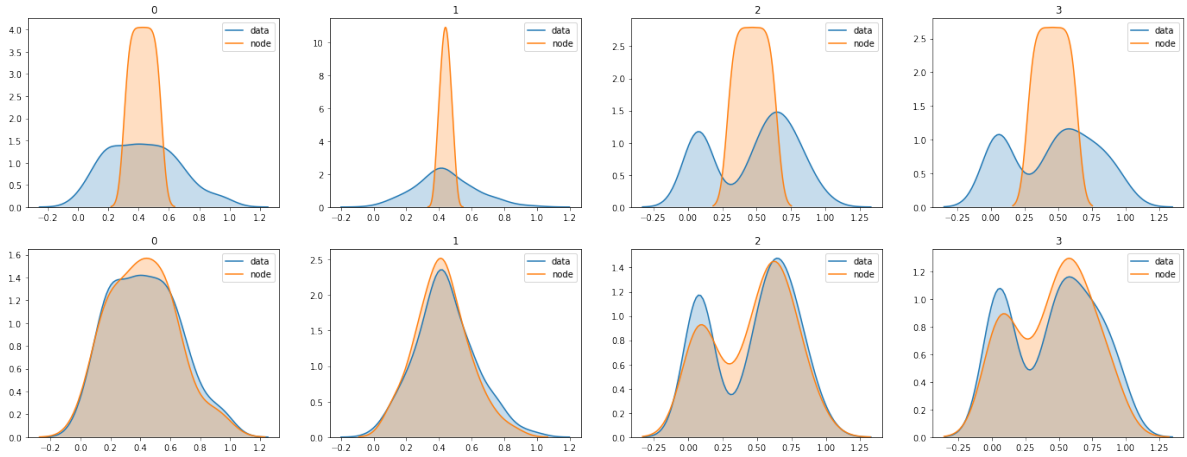


Figure 10: Distributions of the four components of the 4-D Iris Dataset and the components of the SOM before and after training. PCI initialization was used. It is clear that at the end of training, the SOM components take on the distribution of the data.

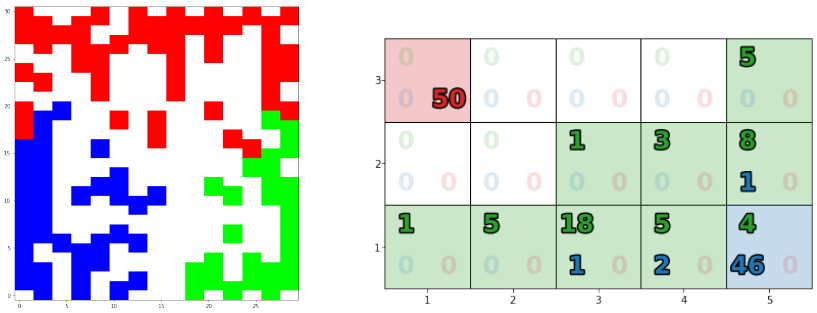
5 Interpreting and Visualizing a Self-Organizing Map

The Self-Organizing Map is intended to be a visualization tool. Inherently once a map is trained, it consists of organized nodes in the input space. All SOM visualizations are rooted in either analysis on the locations of those nodes or analysis of input data points mapped to the nodes.

5.1 Class Representation Maps

Even though the Self-Organizing map is an unsupervised process, it can be used for supervised learning. It is common to want to know where different classes lie in the 2-D lattice. We can plot this in a few ways, first by just looking at the most common type of class in each node. In Figure 11(a) we have a class map of the Wine Dataset[3] from the UCI Machine Learning Repository. This is an example of plotting the most frequently occurring class in each node. This map tells us that the data is easily separable along class boundaries. For a more informative visualization, we can display the counts of each class, pictured for the Iris dataset in Figure 11(b).

We also can look at continuous variables to try and find clusters as seen in Figure 12. The analysis of continuous variables as an output to the SOM is not as useful without the component plane plots. Component Plane Plots are discussed in Section 5.3. The way we compute continuous representation maps can be found in Algorithm 2.



(a) (b)

Figure 11: Class Representation Maps for a 15×15 SOM trained on the Wine Dataset(a) and a 3×5 SOM trained on the Iris Dataset(b). For each node of the lattice, the class representation map shows the color of the majority class that is near it. White is used when a node has no data points closest to it. The plot in (b) has a count of the number of data points per class in each node.

Algorithm 2: Creating the Continuous Class Map

Result: Computes the average value of the response variable of the nearest neighbor data points for each node

$N \leftarrow$ number of nodes in the lattice;
result $\leftarrow$ a N length vector of zeros;
counts $\leftarrow$ a N length vector of zeros;
for $i \leftarrow 1$ **to** N **do**
    $Y_i \leftarrow$set of response variables for data points that are closer to node i than any other node;
    for $j \leftarrow 1$ **to** $|Y|$ **do**
        result$[i] \leftarrow$ result$[i] + Y_i[j]$;
        counts$[i] \leftarrow$counts$[i] + 1$
    end
end
result $\leftarrow$ result/counts

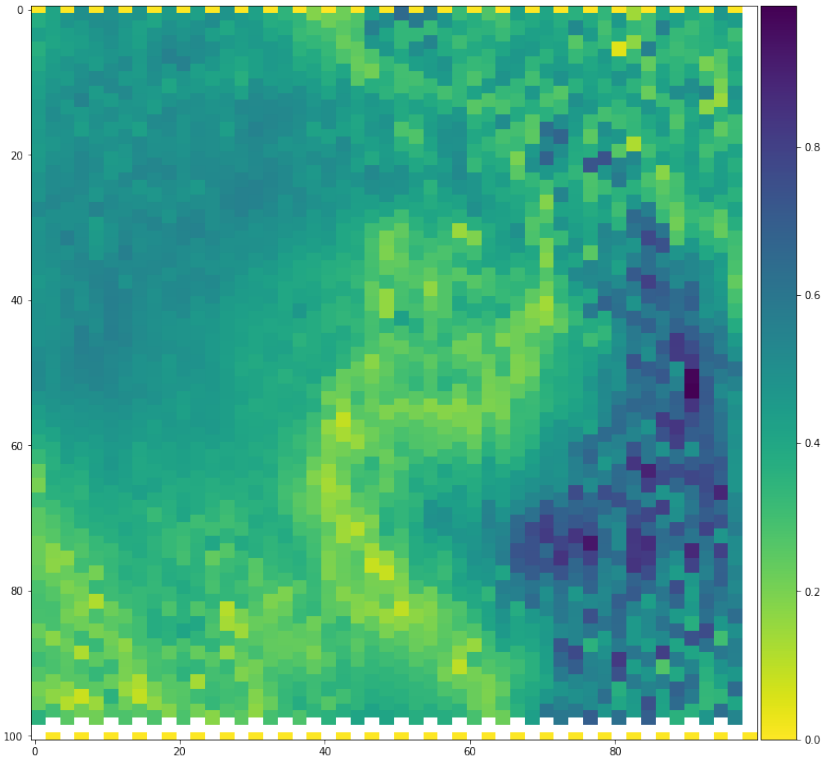
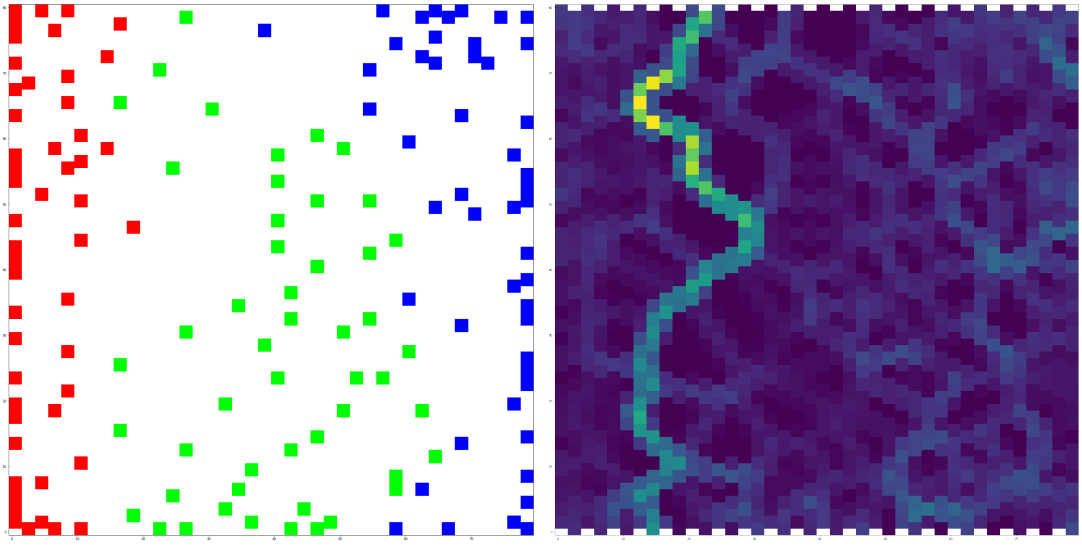


Figure 12: Continuous Response Map for the ET 5000 dataset on a 50×50 SOM. In this Continuous Response Map, we are showing the average value per node of our response variable. This graphic was created using Algorithm 2.

5.2 U-Matrix

The U-Matrix[14] is a tool that tells us about the spacing of a nodes in the input space. The U-Matrix is visualized as a cell for each node in the lattice space. The color of each node is proportional to the average distance in the input space to neighbors of that node. The U-Matrix can be used to find clusters and outliers. In Figure 13 we trained a 40×40 SOM on the Iris dataset. Note that lighter colors on the U-Matrix indicate higher distance so a darker color would mean nodes in that area are dense.

Notice how in the U-Matrix shown in Figure 13(b), we can see that there is a large area spacing the red and green points. This indicates to us that the red points are spaced very far from the green points. The U-Matrix is not always effective. If your map is twisted or does not fit the data well, it could give inaccurate information. When interpreting any output of a Self-Organizing Map, one should make sure that the map is an accurate representation of the data.



(a) Class Representation Map after training a 40×40 SOM on the Iris Dataset.

(b) The corresponding U-Matrix. Note the gap between the red and green classes.

Figure 13: The Class Representation Map and U-Matrix for a 40×40 SOM trained on the Iris dataset. In (b) brighter colors indicate a larger gap between nodes.

5.3 Component Planes

Component planes are plots that show the value of the weight vectors of the SOM in each dimension. This plot can help in analysis of the effect of a particular variable on the output

From this plot we can learn about what features contribute to what class. If we look at dimension 13 in 14 and compare it to the class map in the same figure, we can see that the blue class tends to have higher values in dimension 13. This indicates that this dimension is an important indicator of the blue class. Although trends can be found in these plots with careful analysis, one should rely on more data driven methods to make generalizations.

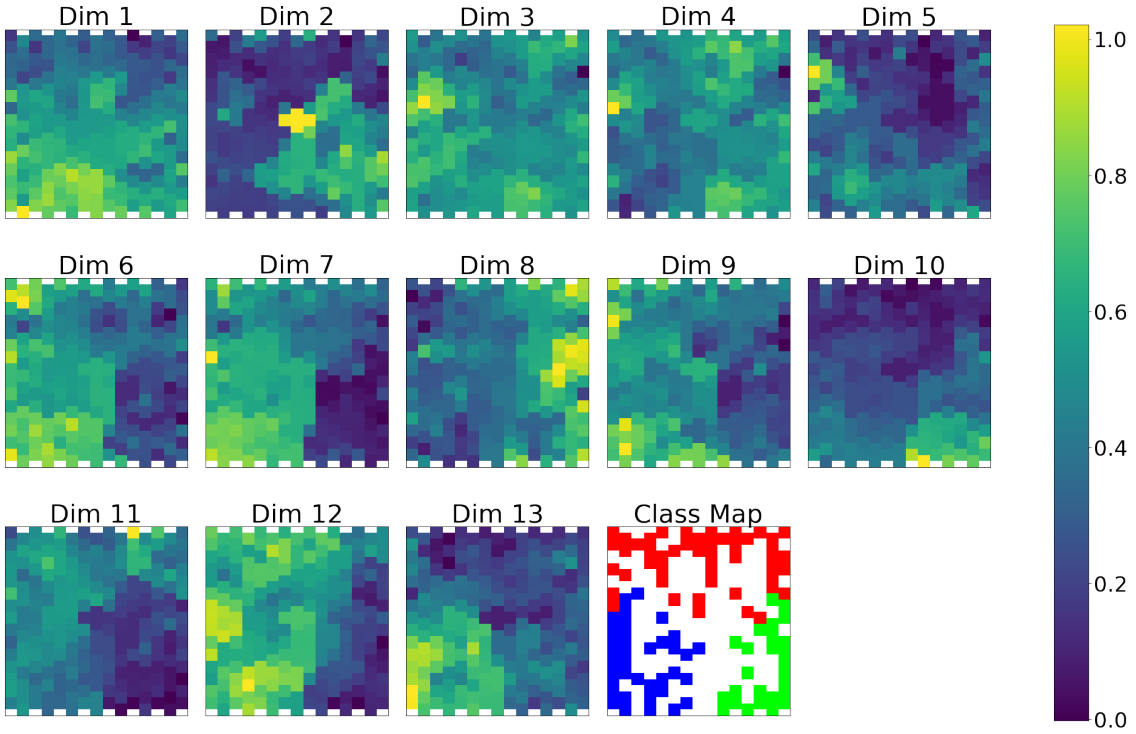


Figure 14: Component Planes and Class Map for the 13-D Wine Dataset of a 20×20 SOM

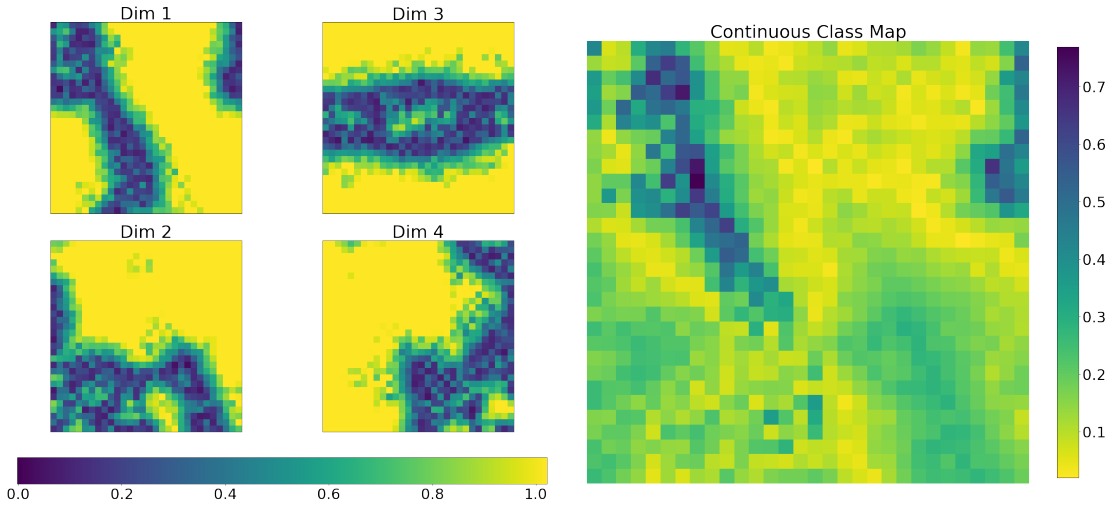


Figure 15: Component Planes and Continuous Class Map for ET5000 dataset of 30×30 SOM. We are interested in when the response variable takes on low values, which is bright yellow in the class map. Although they are not our points of interest we can see that when the response value is very high, Dimension 1 takes on low values and Dimension 2 takes on high values.

5.3.1 2-D Component Planes

In the previous section, we see it is clear to see the effort required to visually inspect and make assumptions about the data by looking at a single component plane. It is even harder

to try and correlate a trend from two features to a response variable. One approach to this problems is using different color channels to represent different variables. Figure 16 shows the a 2-D component plane for the first two dimensions of the ET-5000 dataset. The areas that are distinctly red imply that the first dimension is very high while the second dimension is low. The same is true for distinctly green areas, it implies that the value of the first dimension is very high while the second dimension is very low. A general idea of how different component values affect the color of these maps can be seen in Figure 17.

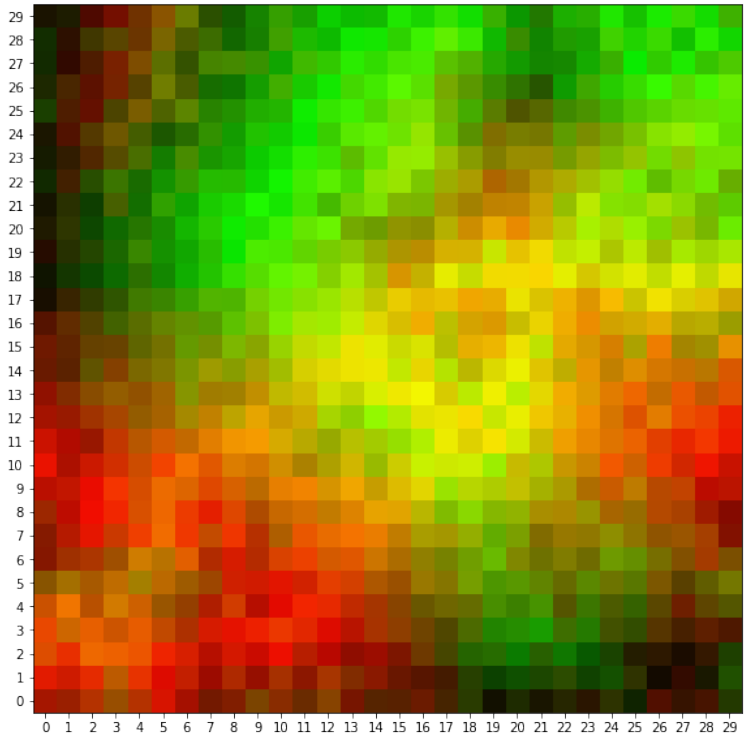


Figure 16: 2-D Feature Map of the ET-5000 dataset for the first two input dimensions.

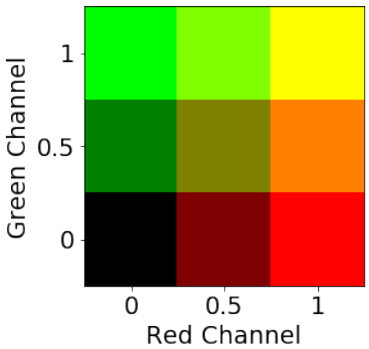
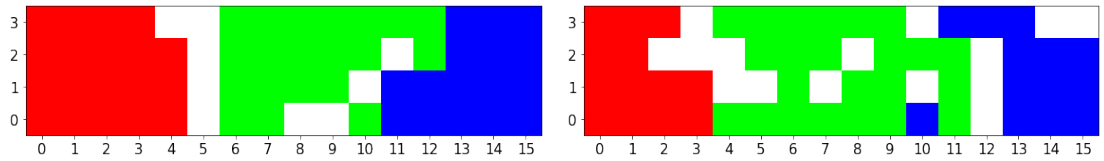


Figure 17: Examples of the color of a cell given different component values.

6 Practical Advice and Applications of a Self-Organizing Map

6.1 The Effect of Noise

It is possible that some dataset will have noisy variables that do not relate to the trend we want to observe and study. Self-Organizing Maps are inherently resistant to noise because they 'look' for structure and random noise lacks structure. The noise filtering power of a SOM can be illustrated by adding random noise features to every data point. We trained one SOM on Iris and one SOM on Iris with random data added as an extra dimension and then plotted the corresponding class representation maps in Figure 18. In Figure 18(b) we have added 8 dimensions of random noise to the Iris dataset, making the original features of the dataset only a third of the total features. You can see that the differences in the class mapping between the data with no noise and the data with noise are minimal. The computation time also had a negligible difference, .5598 seconds for Iris without noise and .6140 seconds for Iris with noise. This is just one example, random noise is unpredictable and it is possible that it could have a large effect on the output of the SOM.



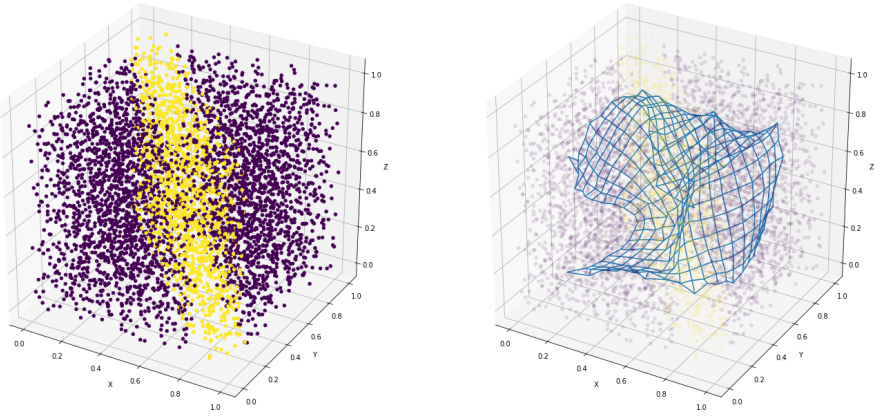
(a) Class mapping on Iris (b) Class mapping on Iris + 8-D random noise

Figure 18: The class mappings are nearly identical and they both tell us that this dataset is mostly separable.

6.2 When Do SOMs Give Us Useful Information About a Dataset?

The Self-Organizing Map is a useful tool when we are working with data that can be defined as a 2-D topological surface. Figure 1 shows data in $\mathbb{R}^3$ but it is clear that the data points lie along a 2-D surface that could easily be defined in $\mathbb{R}^2$. A dataset that cannot be defined by a 2-D surface cannot be well described by a SOM. In Figure 19 we have a dataset with two classes in $\mathbb{R}^3$. The class labels of this dataset have a clear 2-D relationship but since describing the shape of the data requires a third dimension, the SOM lattice, as seen in Figure 19(b) is twisted and does not preserve any information about the dataset. In higher dimensions it is very difficult to tell if our data can be described by a 2-D surface. Because the training times of SOMs are relatively small, one can always train a SOM on a dataset and see if it yields interpretable results.

The Self-Organizing Map is also useful if we want to determine if clusters of data are separated or intermixed. If we are only looking for information about mixed clusters, the SOM does not require that the data be drawn from a surface.



(a) A two class dataset in $\mathbb{R}^3$ (b) A SOM fit to that dataset.

Figure 19: A two class dataset that cannot be modeled by a SOM. Notice how the lattice is twisted and does not represent the shape of the data. However this SOM might tell us if there are any areas where the classes mix.

6.3 How to Approach a New Dataset

When applying a SOM to any new dataset the first step is to rescale the data so each feature spans the same range. Common practice is to scale these features from 0 to 1. Note that will let the SOM treat each feature as equally important. This can be a drawback when certain features are more important to the trend we are trying to measure. This can be addressed by rescaling features to different ranges, more important features should have a larger range. If possible, feature selection should be done to remove features that do not help describe the desired relationship in the data.

Once the dataset has the appropriate scale, we can train a SOM on it. Since training a SOM takes little time, it is advised to run SOMs with many different parameter choices to find the ideal configuration. For every different configuration of parameters one should plot the U-Matrix, Frequency Map, Component Planes, and Class Representation Map(if applicable). First one should use a lattice with the size and shape recommended in Section 2. This will most likely result in a SOM that has at least one point in every node. From that point, it is recommended to increase the number of nodes and iterations until either no noticeable change in the output is visible or running time becomes a limiting factor. If a trend does not become visible at any point in this process it is possible that the SOM is unable to find a pattern in this data. A SOM failing to provide information about a dataset could happen for a number of reasons, some of which were discussed in the subsection preceding this one.

6.4 Applying a Self-Organizing Map to the Iris Dataset

As previously discussed, the Iris Dataset is a 4 dimensional dataset of 150 points. There are 3 classes with 50 samples per class. We can use a Self-Organizing Map to give us information about the separability of the classes. First we must train a SOM on the dataset. Using the rule of thumb for number of nodes, $5\sqrt{150} = 61.23$, we should use close to 61 nodes to represent our dataset. The ratio of the eigenvectors tell us that we should use a lattice size of 16×4. We initialize our weights with PCI.

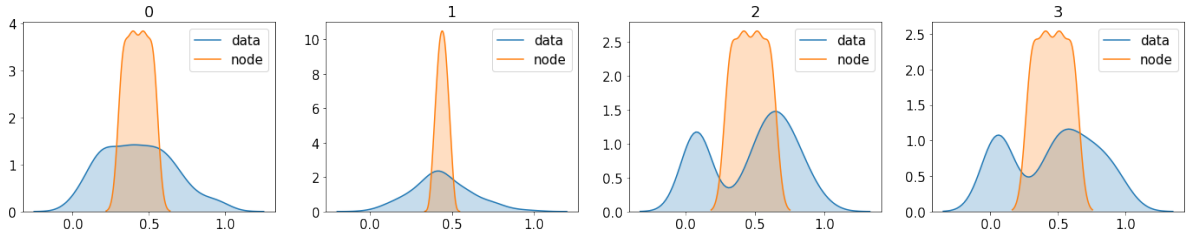


Figure 20: Distributions of each component of the Iris dataset before training. Note that this is the same figure as shown in Section 4.1.3

We then train the SOM for 250 batch iterations with a starting sigma of .25 and final sigma equal to the distance to the closest neighbor. A gaussian neighborhood function is used.

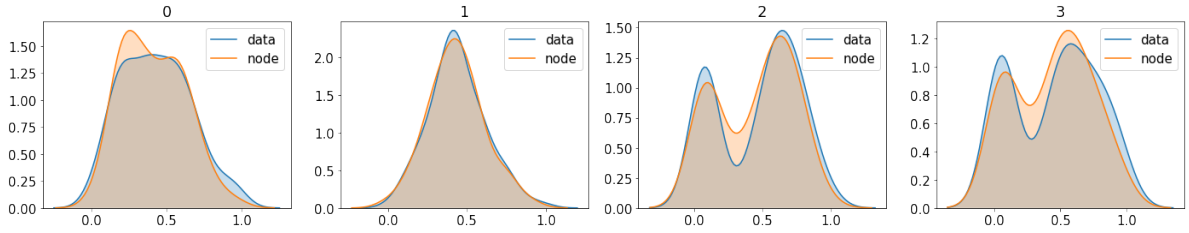
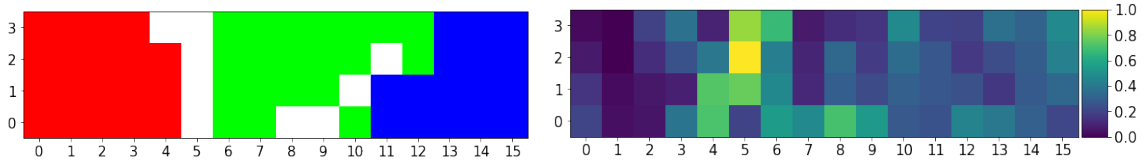


Figure 21: Distributions of each component of the Iris dataset after training. Note that this is the same figure as shown in Section 4.1.3

The SOM seems to fit the distribution of the data, this means we can trust that the SOM represents the data well. We can now plot our class mapping and U-Matrix



(a) Class mapping on Iris

(b) U-Matrix of Iris

Figure 22: From the class representation map we see that the classes are mostly separable. The U-Matrix tells us that the red and green classes are spaced apart, whereas the green and blue classes are touching.

We can see that the data is separable. We can also see that the distance between the red and green classes is high, but the distance between the blue and green classes is low. This leads us to believe that each of the classes are in their own cluster and the green and blue cluster are probably somewhat intermixed.

6.5 Applying a Self-Organizing Map to Wind Energy Data

Since our wind energy dataset is an unlabeled dataset, all of our analysis of the dataset will be done on the final weight vectors of the SOM. Since each 24 dimensional point has an implicit ordering to it (dimension 2 happened after dimension 1 etc.), we can plot each data point as a single plot.

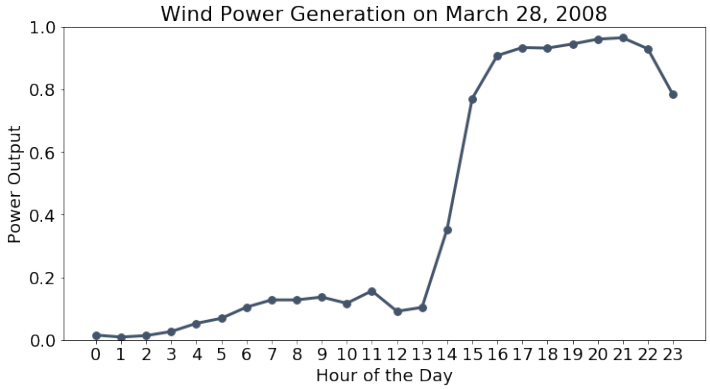


Figure 23: Example of plotting a point from 24 dimension space on a 2-D plot

We trained a 21x10 SOM for 1000 iterations on the data. We can see in Figure 24 that there is a smooth transition from one weight vector to its neighbors. This implies that the data is an ideal candidate for clustering. We then clustered the values of the weight vectors using K-Means clustering with K=5. The output of our SOM is below with the colors corresponding to cluster membership. Each box represents a node of the SOM and in each node we have inserted a plot similar to Figure 23. Figure 24 helps illustrate that in our 2-D mapping, windy days that are near each other in the lattice space, have similar properties. For example the days that have high wind all day is on the opposite side of the days that have low wind all day.

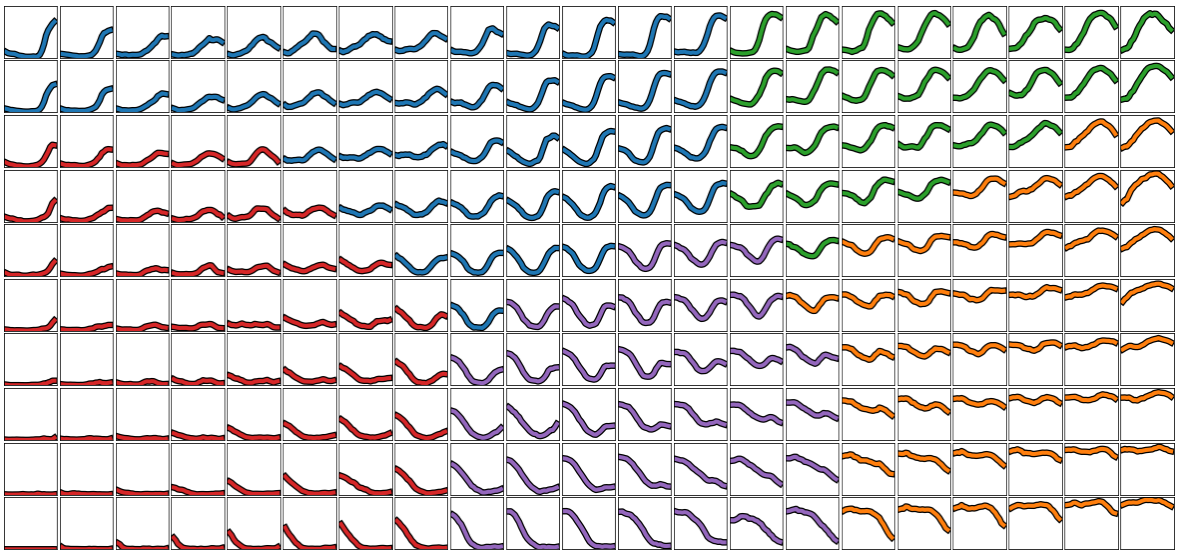


Figure 24: Output of 21x10 SOM trained on Wind Energy Dataset, each color represents a cluster. Each cluster is indicative of a commonly occurring windy day.

6.6 Applying a Self-Organizing Map to Additive Manufacturing Data

The ET-462 dataset is a 4 dimensional dataset with a 1 dimensional response variable. This dataset was gathered from a simulation of a laser melting a layer of metal powder that then merges with the melted metal below. This simulation is based on the Eagar-Tsai model and is a very simple simulation that is not very accurate. For each simulation run the laser speed, laser power, beam size, and absorptivity of the metal were specified and the depth of the melt pool was recorded at the conclusion of each simulation. A more comprehensive explanation of the dataset can be found in Kamath and Fan[8]. When using a laser for additive manufacturing of metals, it is optimal to have the depth of the melt pool lie within a certain range. The goal of our project was to use the ET-462 dataset to determine the parameter space in which the melt pool depth stays close to 60 microns. This dataset does not have many samples so a new 5000 sample dataset, the ET-5000 dataset[5], was created using best-candidate sampling. Using the ET-462 dataset as a training set, Gaussian Process Regression was used to predict the melt pool depth at each of these 5000 points.

Our goal was to find out what parameters lead to melt pool depths close to 60 microns. To try and understand the dataset we trained a 30×30 SOM on ET5000. The continuous class map as seen in Figure 25 shows the normalized average distance to 60 microns for each node. The points we are looking for are more yellow in this plot. This class map is telling us that the samples we want are not in a cluster, but instead in small veins throughout the lattice. This SOM analysis did not give us a clear picture of how the data was shaped or if the points we wanted are actually in a cluster.

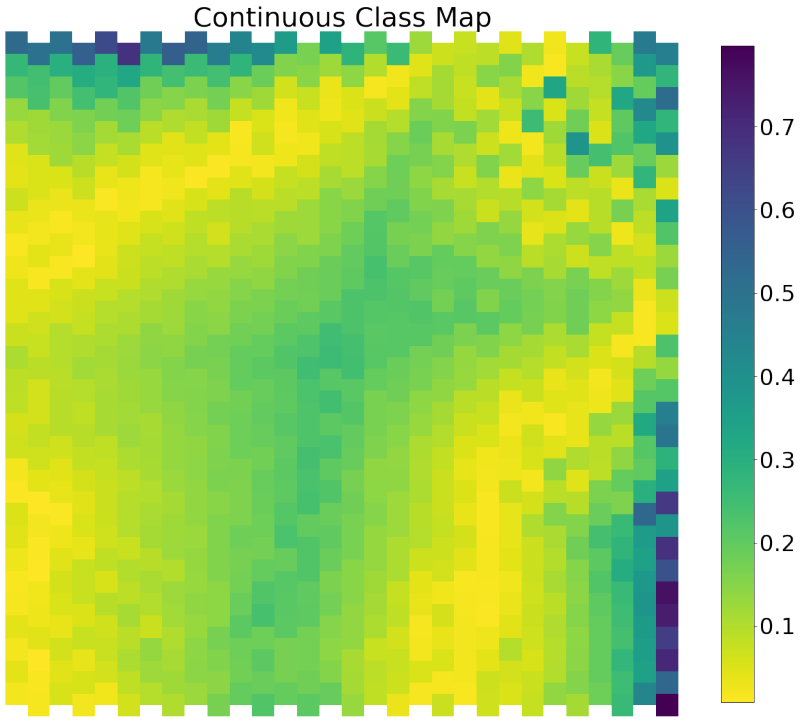


Figure 25: Continuous Class Map for ET5000 dataset of 30×30 SOM

Since our dataset was generated using a method that has a distribution similar to uniform sampling, the entire dataset was not an ideal candidate for analysis with a SOM because it spanned the entire unit hypercube in $\mathbb{R}^4$. We saw in Figure 19 why this does not work well. Since we are only interested in what parameters lead us to a depth of close to 60 microns, we chose to train a SOM on just what we consider the good points(within 5 microns of 60μm). When this was done, we looked at the SOM lattice in 3-D by only plotting three of the features. We knew that the first two features, speed of the laser (v) and laser power (q) were essential to the depth value so they were not removed. We also knew that there was a linear relationship in v and q that dictated where the good points lie. In Figure 26 we see that all the points within 5 microns of 60 have a linear relationship between v and q. This indicates that a certain v has a small range of allowable q values.

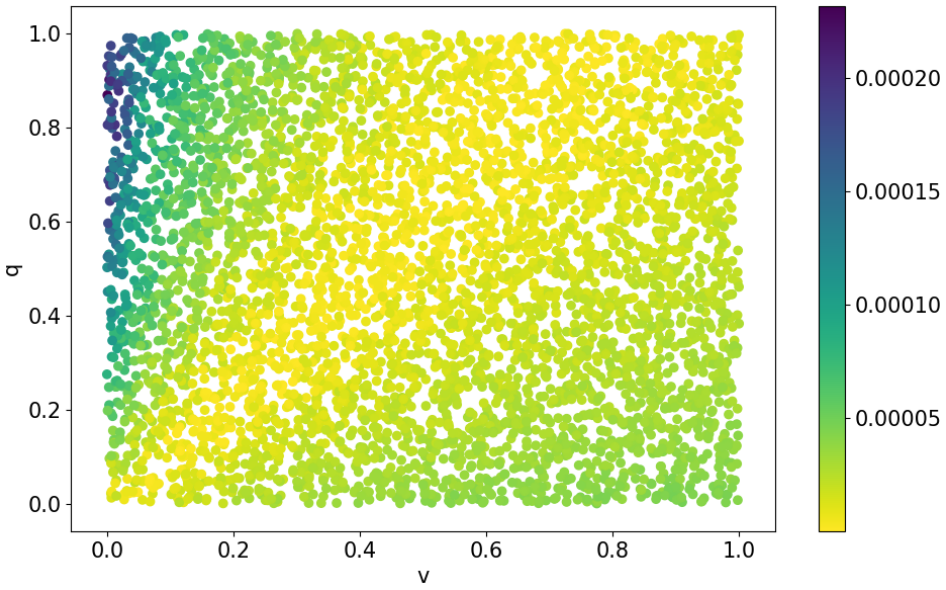
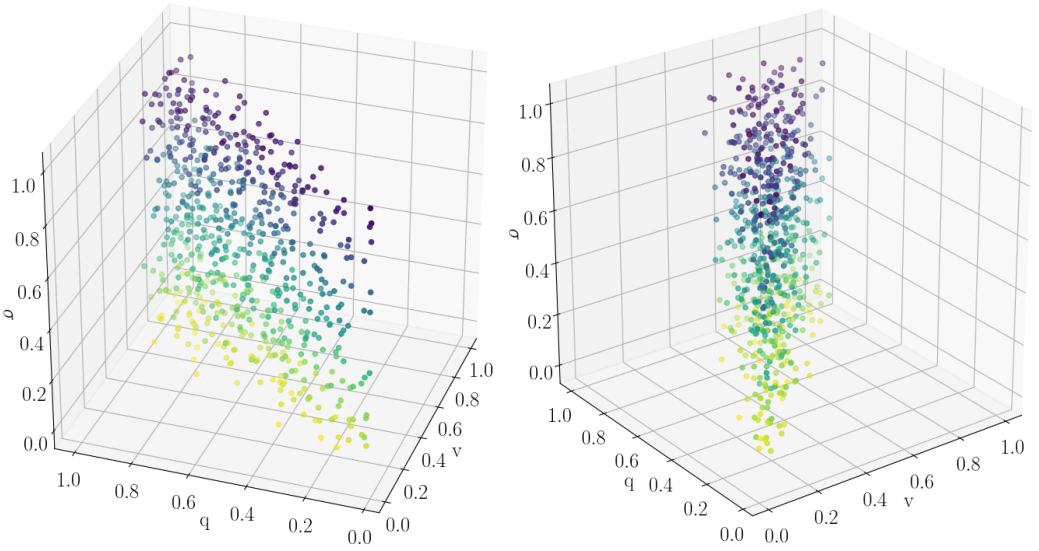


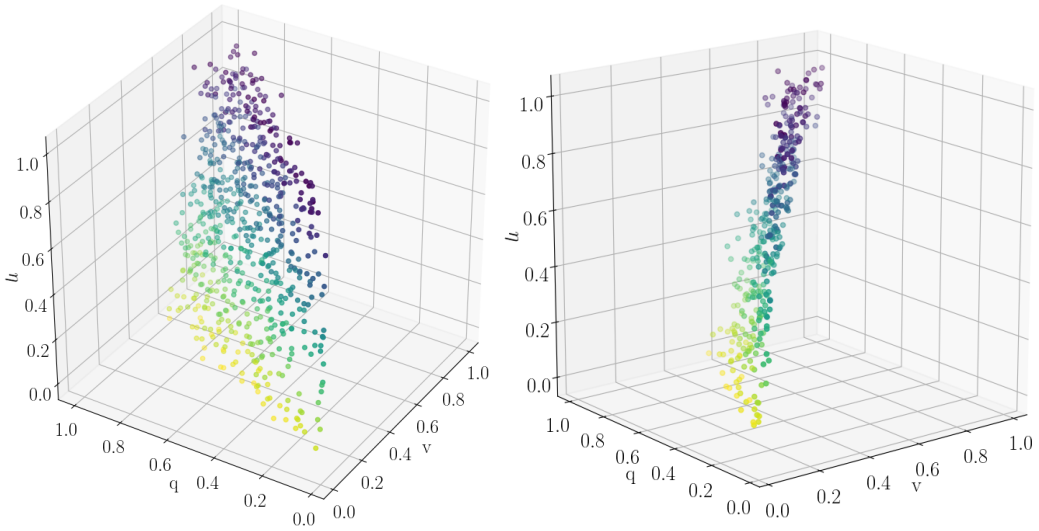
Figure 26: The color of each point indicates distance from 60 μm. A brighter yellow indicates that point is closer to 60μm. We can see that all the points within 5 microns of 60 are concentrated along the diagonal.

Using this knowledge, we chose to plot the points within 5 microns of 60 using both beam size(σ) and absorptivity(η) as our third variable.



(a) (b)

Figure 27: A 3-D scatter plot of the points within 5 microns of 60 using σ as our Z-axis. To aid in visualization, color of each point is proportional to the σ value. Note that as σ increases, there is little to no noticeable change in where the good points lie, this would imply σ has little to no effect on the depth value.



(a) (b)

Figure 28: A 3-D scatter plot of the points within 5 microns of 60 using η as our Z-axis. To aid in visualization, color of each point is proportional to the η value. It should be noted that the η value clearly has a larger effect on the depth than the σ.

We can see that there is a surface that the good points tend to lie along. As we can see in Figure 28(b), the η value affects the depth not just v and q. If this surface is a linear surface then it is very easy to generate a parameter space and quantify variability of the depth using normal distance to the plane. We tested the linearity of this by training a SOM on just the good points shown in Figure 28, using only the v,q, and η dimensions. We then took each lattice node and computed the normal distance to the principal component plane of the dataset. When looking at the normal distance to the PCA plane in Figure 29 we see that there is clearly a non-linear relationship, almost like a saddle. Note that the principal component plane was chosen because it is the best approximation of what the defining plane would be if the data was on a linear surface, any deviation from it can be thought of as deviation from linearity. SOMs helped us get a general idea of the shape of the data we are interested in, more work will need to be done to see if SOMs can also tell us more about what parameter values would result in an ideal depth.

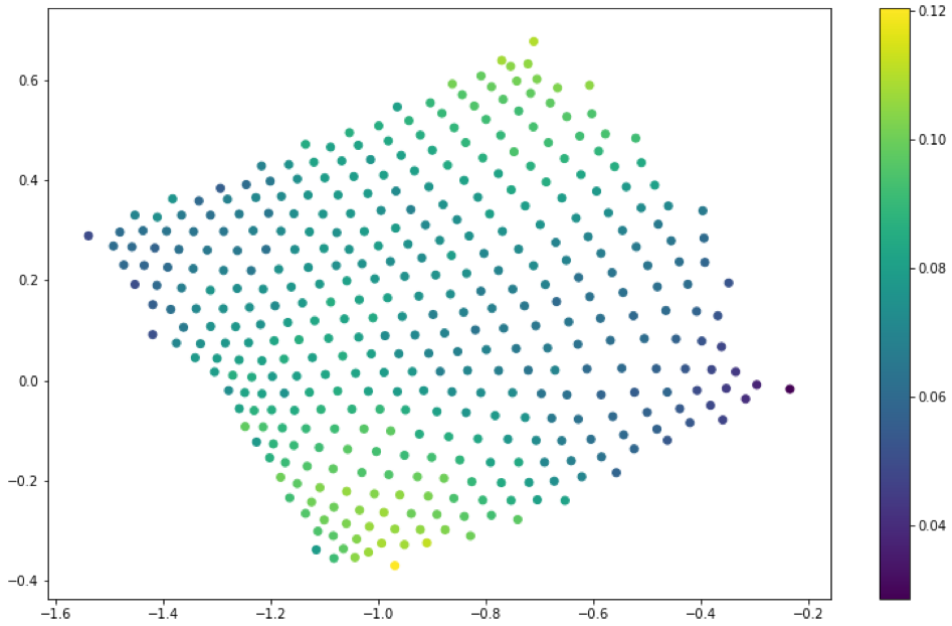


Figure 29: SOM nodes projected onto the principal component plane. The color of each point indicates distance to the principal component plane.

More examples of SOM analysis on real datasets can be found in Appendix A and B.

7 Conclusion

The Self-Organizing Map is a powerful tool for data analysis. We have shown how they can be used to understand the spacing of clusters and the shape of high dimensional data. We have also demonstrated that the Self-Organizing Map is a time efficient method for data sets with both high numbers of samples and also high dimensionality. Finally using real datasets we showed how to interpret and evaluate a Self-Organizing Map to make assumptions about unknown datasets. Because of the low resource requirement of a SOM, we believe that applying a SOM to any high dimensional dataset has the potential to result in valuable and actionable information about that dataset.

8 Acknowledgment

This work is part of the IDEALS project funded by the ASCR Program (Drs. Lucille Nowell and Laura Biven, Program Managers) at the Office of Science, US Department of Energy. It was performed while Ravi Ponmalai, a fourth-year undergraduate at UC Irvine, was a summer intern at LLNL in 2019.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] Akinduko, A. A., Mirkes, E. M., and Gorban, A. N. Som: Stochastic initialization versus principal components. *Information Sciences 364-365* (2016), 213 – 221.

[2] Cottrell, M., Olteanu, M., Rossi, F., and Villa-Vialaneix, N. N. Self-OrganizingMaps, theory and applications. *Revista de Investigacion Operacional 39*, 1 (Jan. 2018), 1–22.

[3] Dua, D., and Graff, C. UCI machine learning repository, 2017.

[4] Erwin, E., Obermayer, K., and Schulten, K. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics 67*, 1 (May 1992), 47–55.

[5] Franzman, J., and Kamath, C. Understanding the effects of tapering on gaussian process regression.

[6] Haykin, S. *Neural networks and learning machines*. Prentice Hall/Pearson, New York, 2009.

[7] Kamath, C., and Fan, Y. J. Finding motifs in wind generation time series data. In *2012 11th International Conference on Machine Learning and Applications* (Dec 2012), vol. 2, pp. 481–486.

[8] Kamath, C., and Fan, Y. J. Regression with small data sets: a case study using code surrogates in additive manufacturing. *Knowledge and Information Systems 57*, 2 (Nov 2018), 475–493.

[9] Kaski, S. Data exploration using self-organizing maps, 1997.

[10] Kohonen, T. *Self-organization and Associative Memory: 3rd Edition*. Springer-Verlag, Berlin, Heidelberg, 1989.

[11] Kohonen, T. *Self-Organizing Maps*. Springer Berlin Heidelberg, 2001.

[12] Kohonen, T. Essentials of the self-organizing map. *Neural Networks 37* (2013), 52 – 65. Twenty-fifth Anniversay Commemorative Issue.

[13] Kohonen, T. *MATLAB Implementations and Applications of the Self-Organizing Map*. Unigrafia Oy, Helsinki, Finland, 2014.

[14] Ultsch, A. U*-matrix: a tool to visualize clusters in high dimensional data.

[15] Yin, H., and Allinson, N. M. On the distribution and convergence of feature space in self-organizing maps. *Neural Computation 7*, 6 (1995), 1178–1187.

Appendix A Examples of Applications of Self-Organizing Maps

This section will contain images of SOMs trained on example datasets and the resulting visualizations.

Rainbow Dataset This is a 2-D four class dataset.

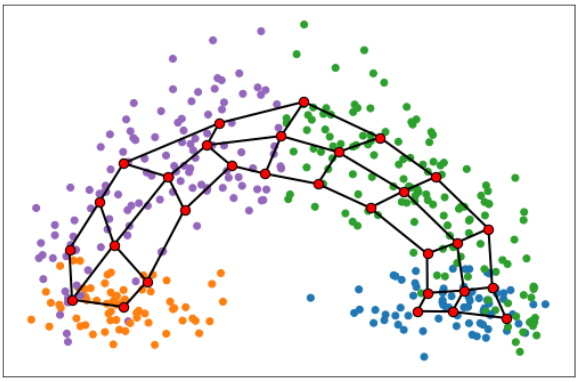


Figure 30: 2-D Rainbow dataset. Color of each point corresponds to the point's label.

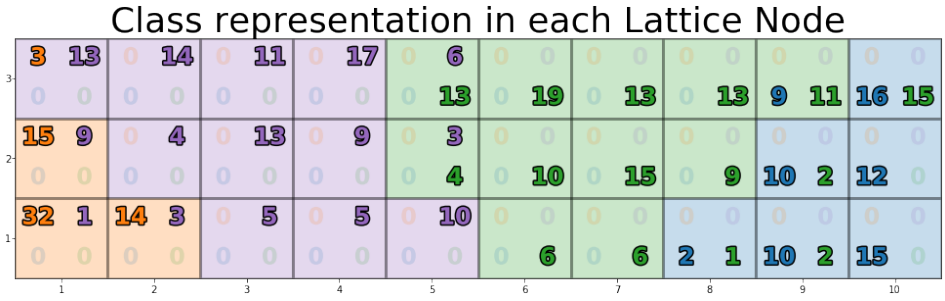


Figure 31: 2-D Rainbow dataset Class Representation Map

Four Rings Dataset This is a 3-D dataset with 4 interlocking rings.

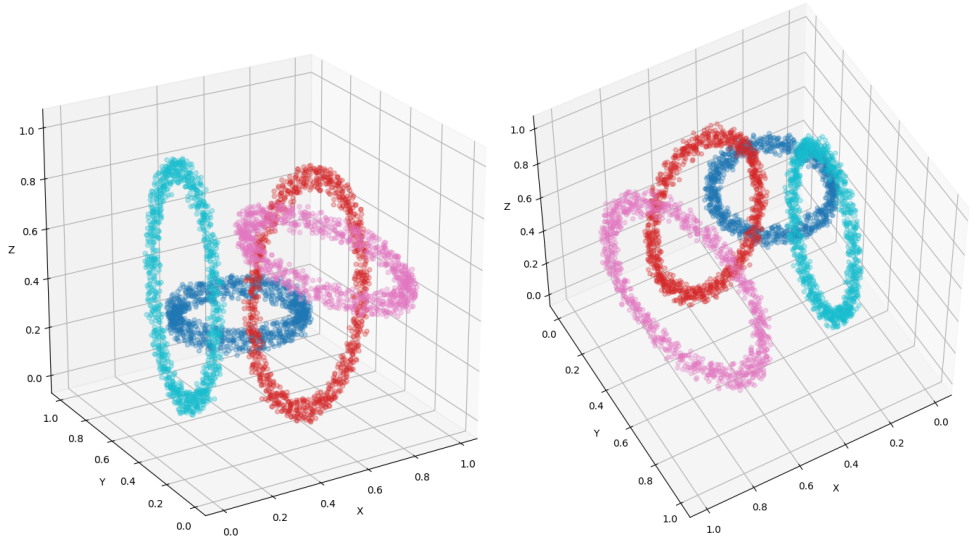
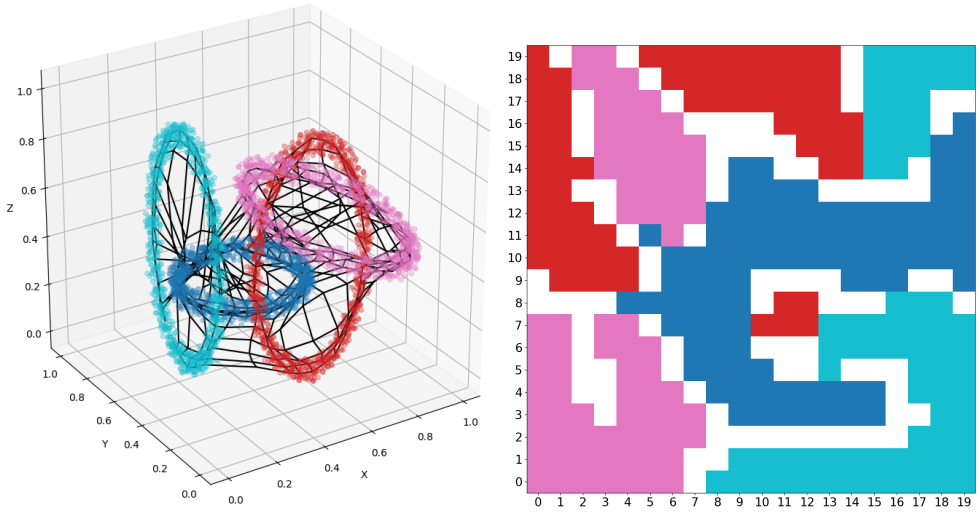


Figure 32: Two views of the Four Rings dataset.



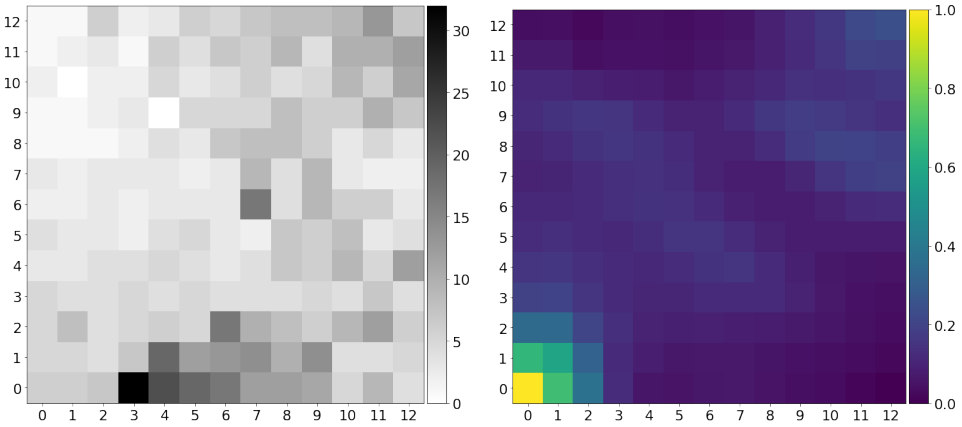
(a) (b)

Figure 33: A 20×20 SOM trained on the Four Rings Dataset and the corresponding class map.

Appendix B Applying a Self-Organizing Map to Unknown Datasets

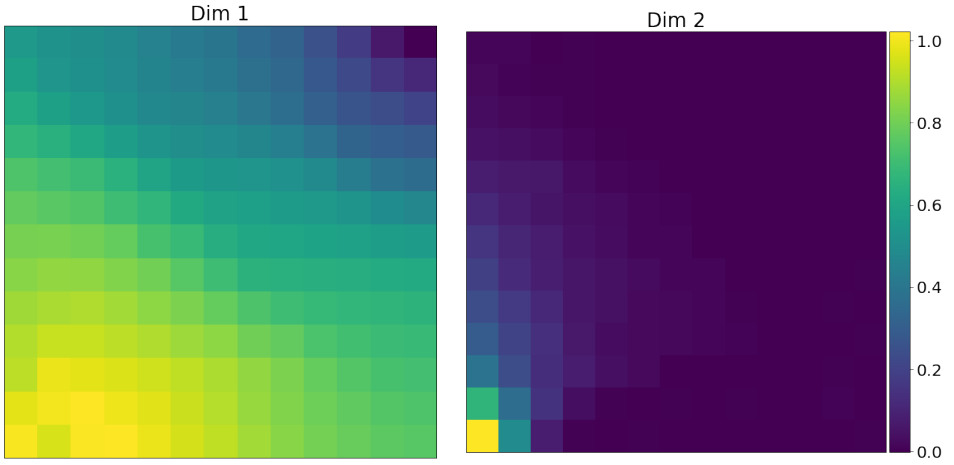
To test what information can be learned using a SOM, six datasets were created for analysis. These datasets are in both 2 and 3 dimensions but for the purposes of this exercise were never plotted by the author until after all analysis had been done. The sizes of the SOM lattice for each dataset was created in accordance with the paragraph on lattice size in section 2. PCI initialization and batch training were used for all the datasets.

Dataset 1 Dataset 1 was a 2-D unlabeled dataset. A 12×12 SOM was trained for 250 iterations on this 993 sample dataset. This is a 2-D dataset so we will try and use the SOM analysis to make a guess as to the exact shape of the data. Firstly, looking at the component planes (Figure 34) we notice that in dimension 2(we will refer to this as D2 from now on), most of the nodes are very close to 0. When we consider those same nodes in D1, we see that their values change smoothly from 1 to 0. This implies that all those points close to zero in D2 span the entire D1 axis. Now we consider the bottom left nodes with a D2 value close to 1. For those nodes, when we look at the value of the corresponding node in D1, we see it is also very high. This implies that specific node is close to the point (1,1) in the input space. We can confirm this by looking at the U-Matrix, where we see that it is very far from its other nodes. Our final guess is that this plot looks like many random points along the D1 axis with a spike on the D2 axis at D1=1.



(a) Frequency Plot

(b) U-Matrix



(c) Component Planes

Figure 34: The Frequency Plot, U-Matrix, and Component Planes of dataset 1

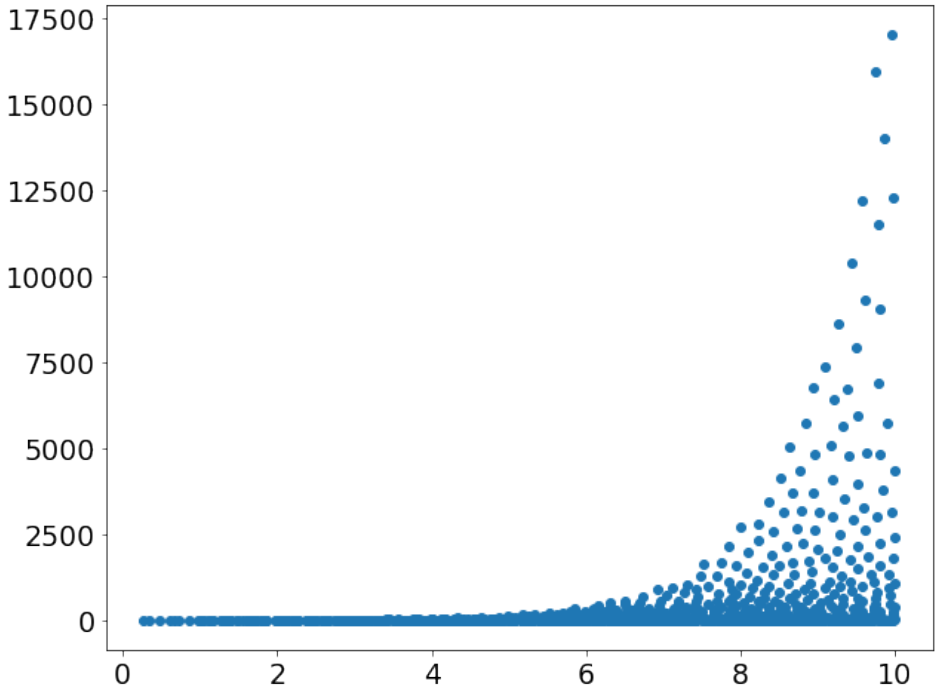
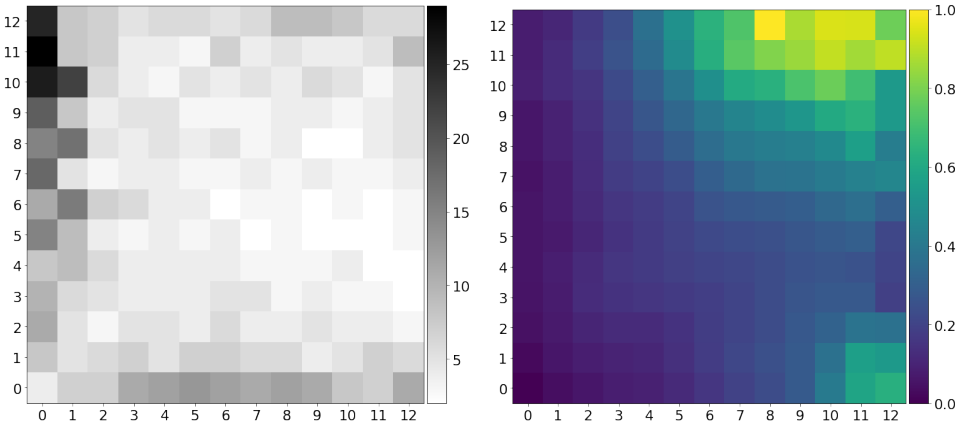


Figure 35: Dataset 1 Plot

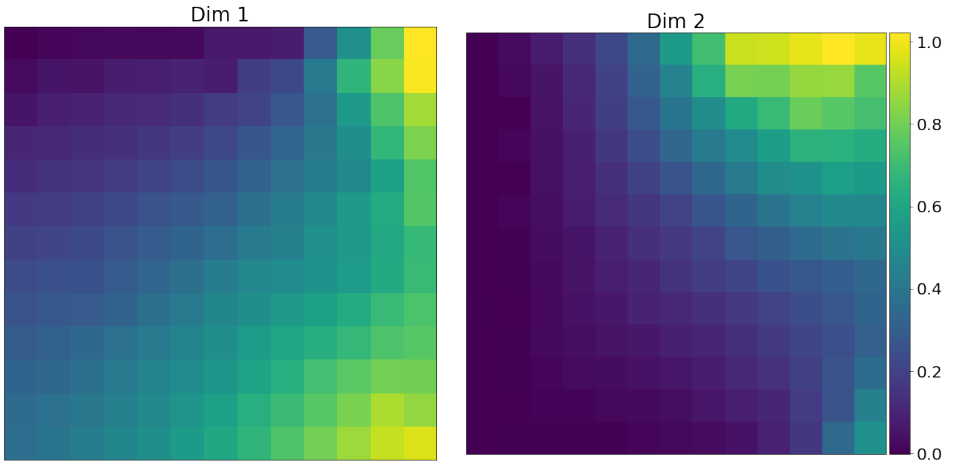
This dataset was created by taking 2000 best candidate sampled points in the range $0 \leq x, y \leq 10$. A subset of those points was created by only taking the points with $y < x$, and the new y value was equal to the exponential of y. Our prediction was very similar to this dataset but could have been improved by mentioning that the density of the points decreased in the spike. We could have made this observation by noting both the higher spacing in the U-Matrix and the decreased values in the frequency plot.

Dataset 2 Dataset 2 was a 2-D unlabeled dataset. A 12×12 SOM was trained for 250 iterations on this 1007 sample dataset. Unlike dataset 1, we will begin our analysis of this dataset with the frequency plot. We notice that a huge majority of the points lie along the left and bottom edges of the lattice. When looking at D1 for those edges we see that starting in the top left corner and following it down to the bottom right corner, there is a smooth change from 0 to 1. In D2 we see that the majority of the points on these edges are close to zero with the exception being the bottom right corner where the value is closer to .5. This indicates that this dataset is similar to dataset 1 except that the spike at D1=1 is shorter than the one in dataset 1. The other difference from dataset 1 is that there are many points not on the left and bottom edges. Our guess is that this is similar to dataset 1 except with random points that span the entire square added to the dataset.



(a) Frequency Plot

(b) U-Matrix



(c) Component Planes

Figure 36: The Frequency Plot, U-Matrix, and Component Planes of dataset 2

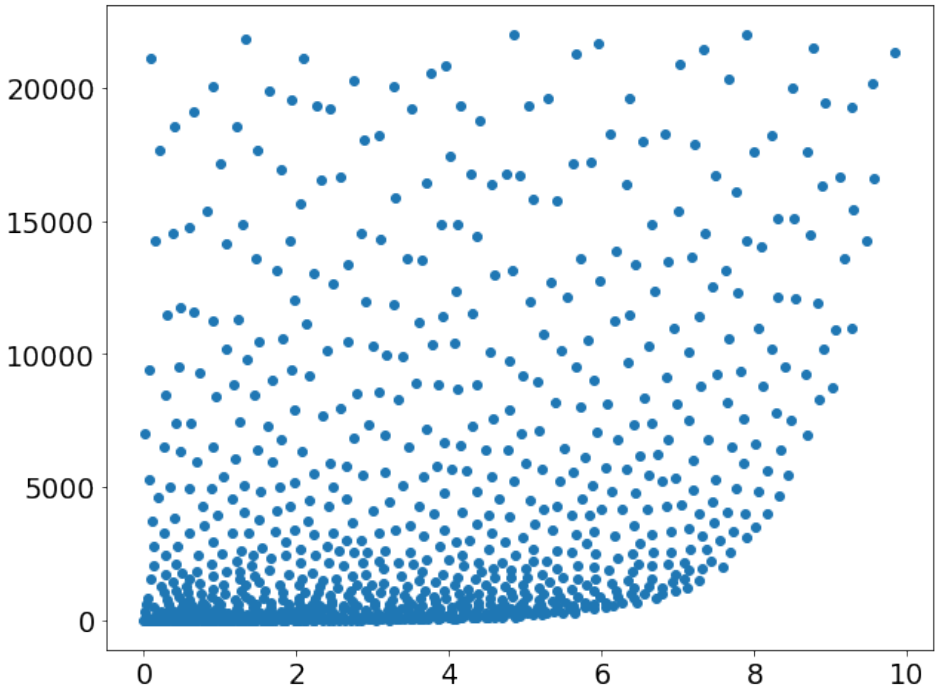
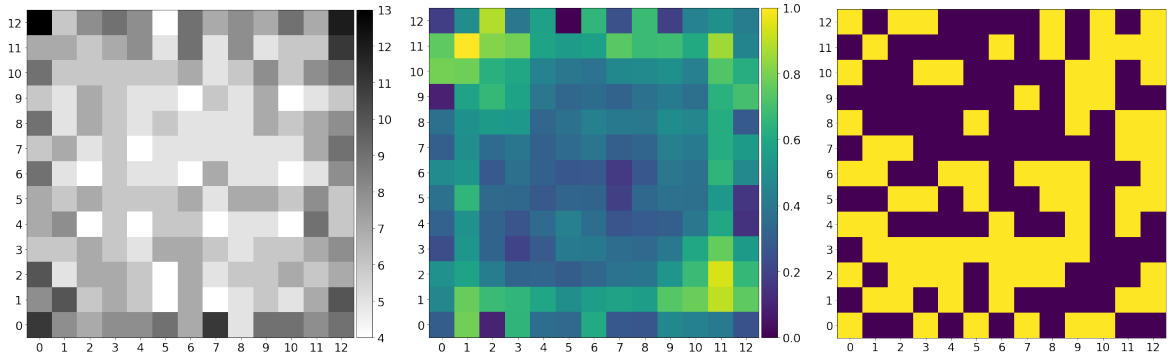


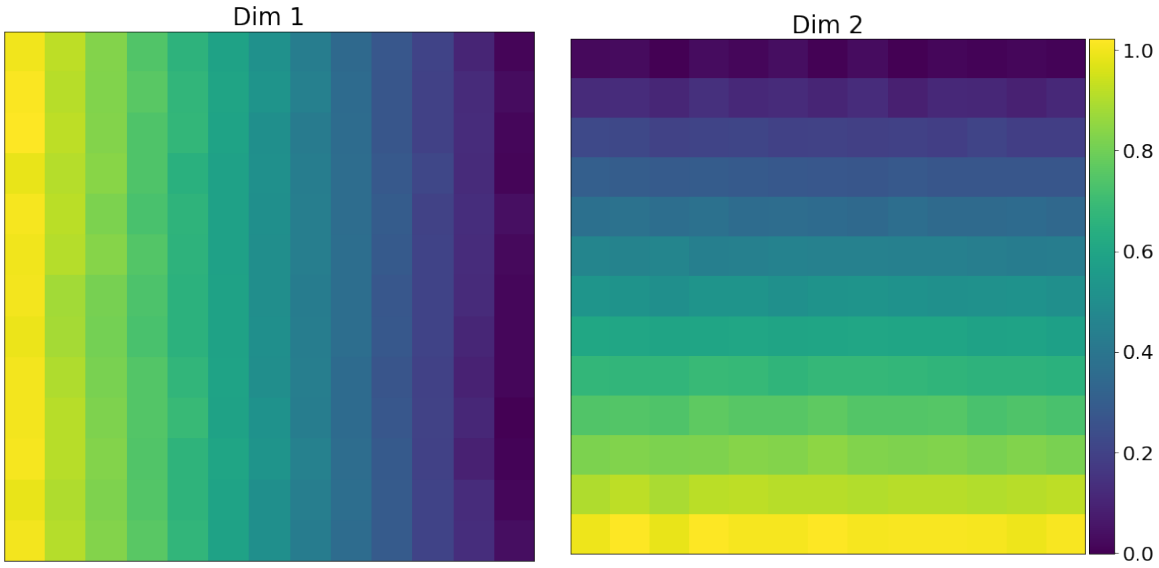
Figure 37: Dataset 2 Plot

This dataset was created by taking 2000 best candidate sampled points in the range $0 \leq x, y \leq 10$. A subset of those points was created by only taking the points with $y > x$, and the new y value was equal to the exponential of y. This is the other subset from dataset 1. We were not as accurate with this dataset. We guessed that this was similar to dataset 1 with random points added everywhere. Although this dataset is similar in shape to dataset 1, the extra points are not purely random and they are not everywhere.

Dataset 3 Dataset 3 was a 2-D labeled dataset. A 12×12 SOM was trained for 250 iterations on this 1100 sample dataset. The smoothness of both component plots tells us this SOM spans the entire unit square in an organized fashion. This is consistent with uniform random data like Figure 2. The Class Representation Map also supports this because there is no discernible pattern. Note that in this case, instead of plotting the most occurring class per cell, we plotted yellow if there are any yellow points in the node and purple if there are no yellow points in that node. It is worth noting that the U-Matrix has a slight gap between the outer ring of nodes. This could be an artifact of the randomness or it could be points that are not uniformly distributed. Despite that our guess is that these are random points with random labels spanning the entire unit square.



(a) Frequency Plot (b) U-Matrix (c) Class Representation Map



(d) Component Planes

Figure 38: The Frequency Plot, U-Matrix, and Component Planes of dataset 3

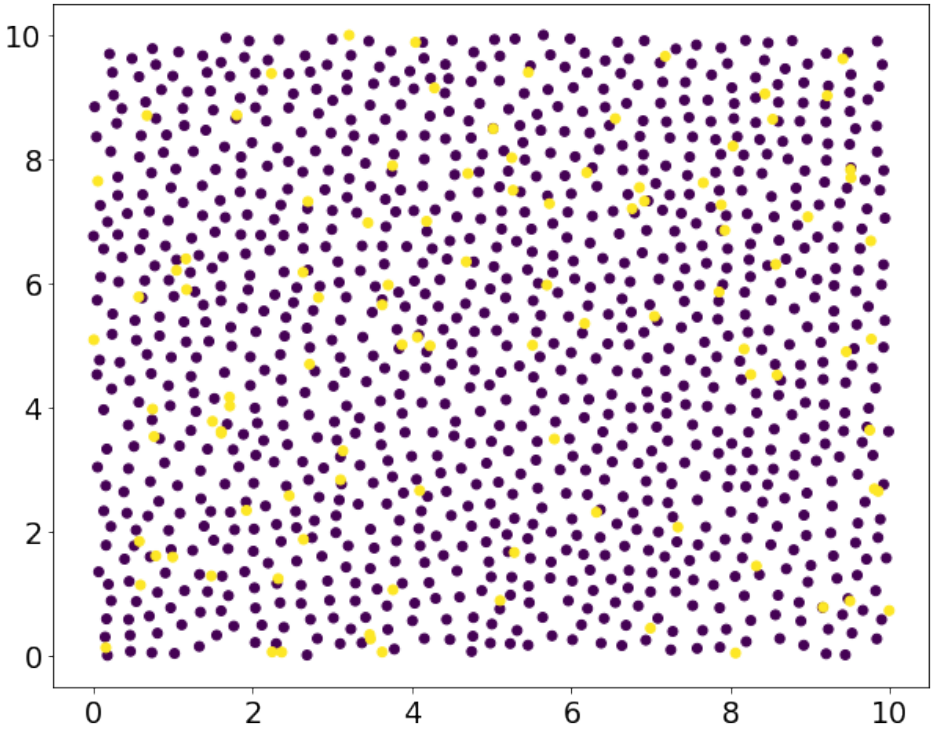
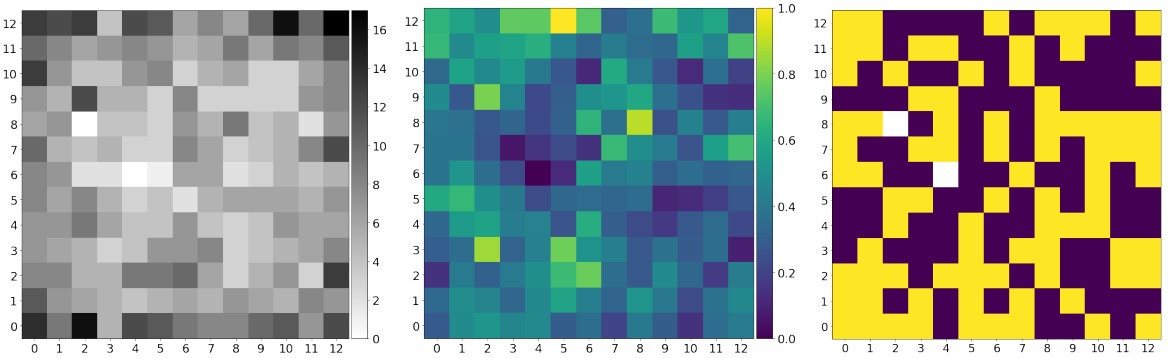


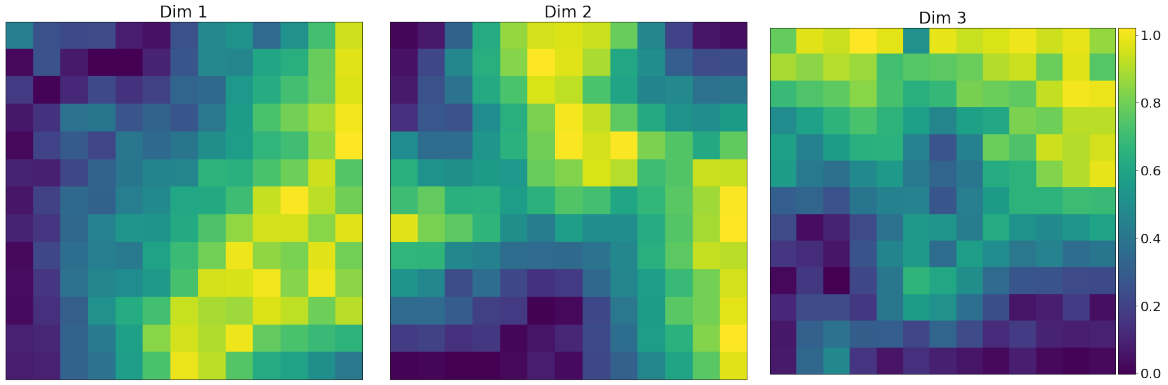
Figure 39: Dataset 3 Plot

Dataset 3 was a collection of two types of randomly sampled points. The purple points were created with best candidate sampling while the yellow points were created with uniform random sampling. Our prediction was that the labels and points were random. While this is not exactly true it is close. If we were to do this analysis again, we would have more closely considered how many of each class were in each node. This would have shown us that the density of data differed among the classes. This could have been done by making multiple class representation maps. Firstly we should have plotted the most occurring class per node, then for each class a plot that simply shows if any points with that class is in that node. These new plots would have shown us that the density of the purple points was constant and the density of the yellow points was slightly more variable.

Dataset 4 Dataset 4 was a 3-D labeled dataset. A 12×12 SOM was trained for 250 iterations on this 1100 sample dataset. The Frequency Plot, U-Matrix, and Class Representation Map indicate that there is no pattern to the data. The component planes do not tell us much about where different classes are. Our guess is that this is a 3-D version of dataset 2, random points with random labels spanning the entire unit cube.



(a) Frequency Plot (b) U-Matrix (c) Class Representation Map



(d) Component Planes

Figure 40: The Frequency Plot, U-Matrix, Class Representation Map, and Component Planes of dataset 4

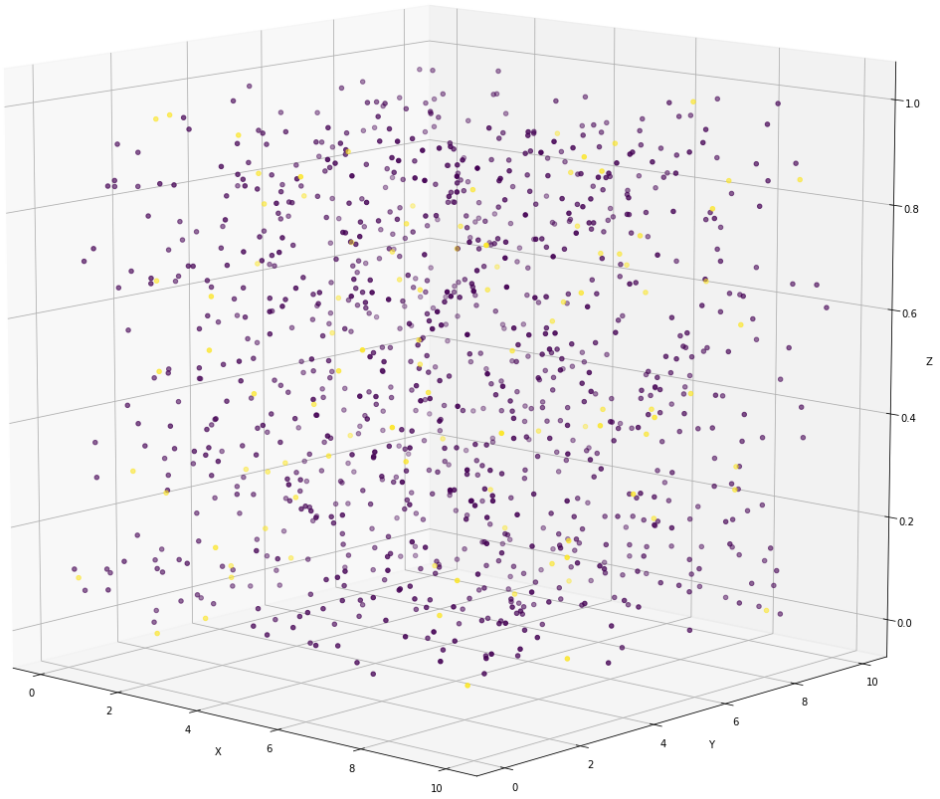
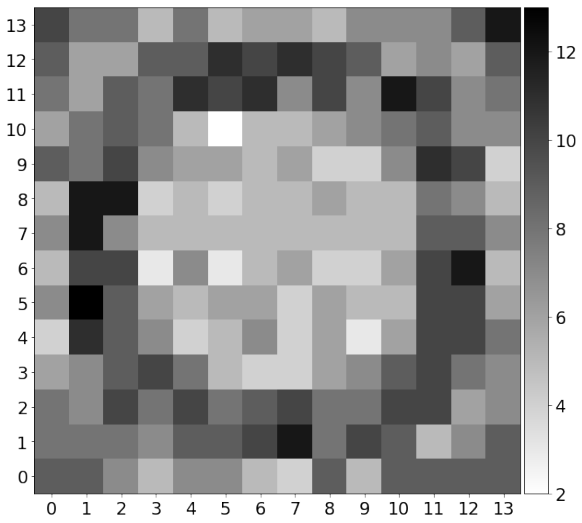


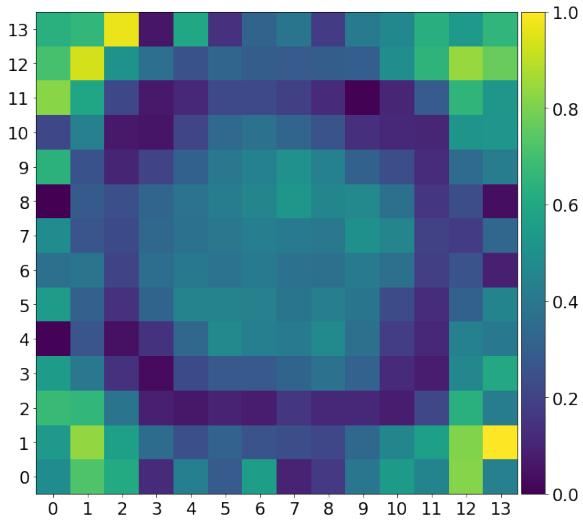
Figure 41: Dataset 4 Plot

Dataset 4 was the same as dataset 3 with an additional noise column. Because of nature of this dataset we have only included one view as it gives an accurate depiction of the data. Our guess was that this was a 3-D version of dataset 3, while this was correct, we made the same mistake as dataset 3 which was not considering the different densities of classes.

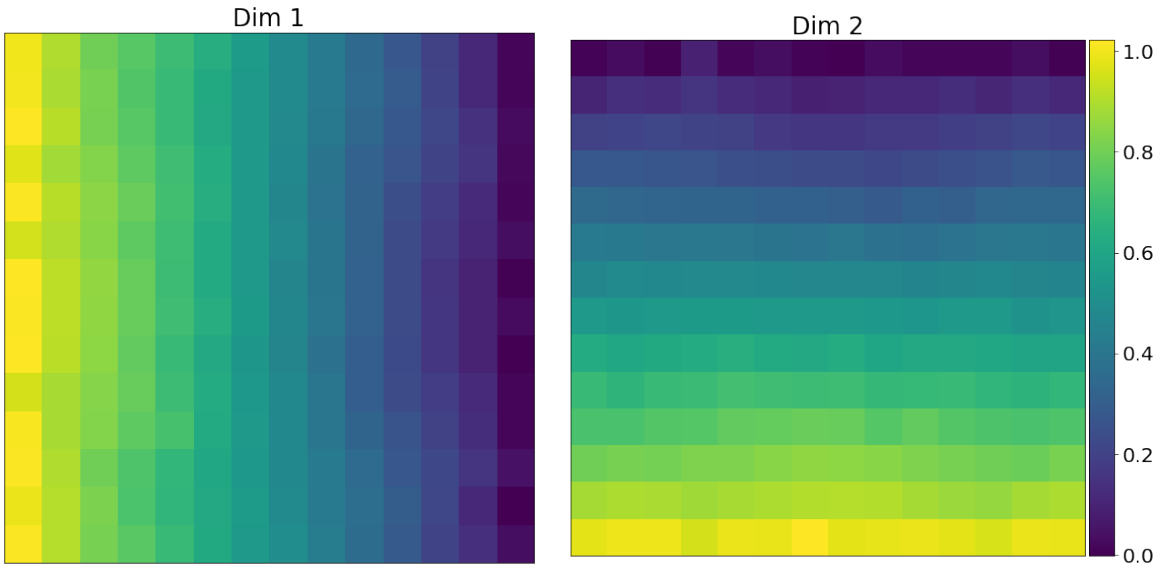
Dataset 5 Dataset 5 was a 2-D unlabeled dataset. A 13×13 SOM was trained for 250 iterations on this 1436 sample dataset. Based on the component planes, the SOM weight vectors most likely span the space uniformly. This implies that the data spans the entire unit square. However the frequency map and U-Matrix clearly show that there is a cluster of points in a ring in the center of the data. Our guess is that these are random points in the unit square with a ring of points added to the dataset.



(a) Frequency Plot



(b) U-Matrix



(c) Component Planes

Figure 42: The Frequency Plot, U-Matrix, and Component Planes of dataset 5

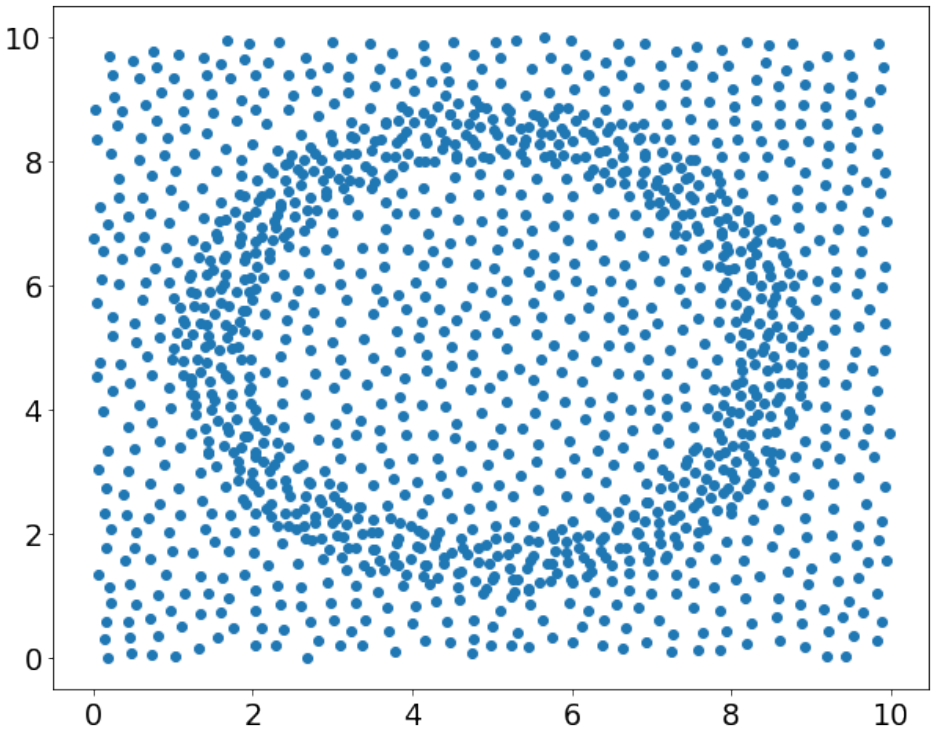
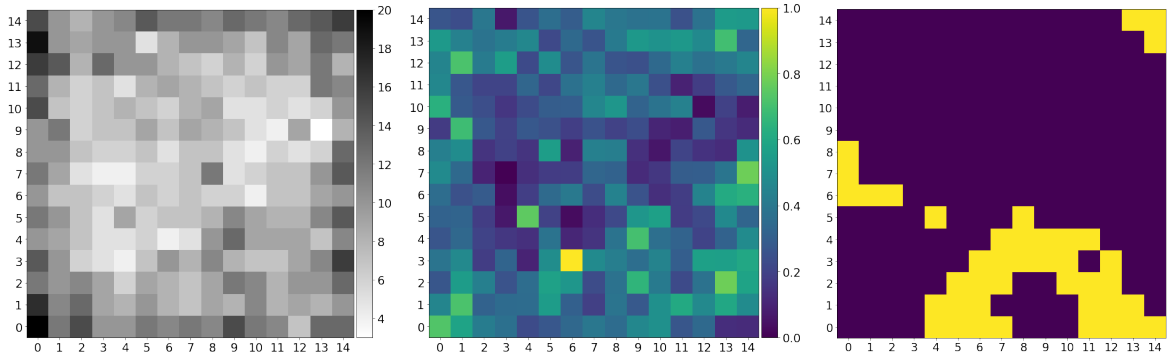


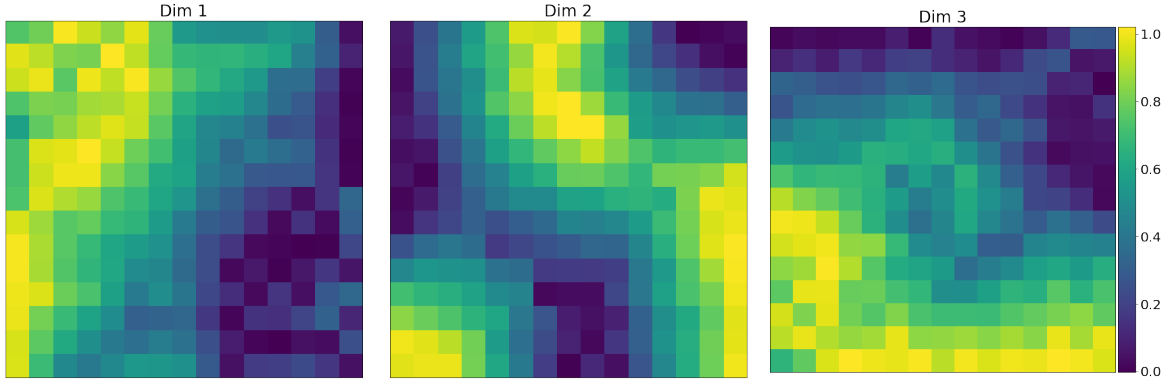
Figure 43: Dataset 5 Plot

Dataset 5 was exactly what we predicted it would be, a ring cluster with other random points. In this case the points were picked with best candidate sampling.

Dataset 6 Dataset 6 was a 3-D labeled dataset. A 14×14 SOM was trained for 250 iterations on this 2000 sample dataset. Surprisingly this component plane has an identical structure to the one in dataset 4. This implies that they are the same datasets with different labels. This would mean that the dataset also spans the unit cube. This dataset differs from dataset 4 because it seems to have some structure to the labels. If we look at the class representation map we see one large cluster of points labeled as yellow. This cluster is connected but has purple nodes on either side of it. Because our data spans a cube, we would guess that the cluster is a plane cut through the cube like in Figure 19. The other clusters could be small separate clusters or connected to the main cluster, it is difficult to tell.



(a) Frequency Plot (b) U-Matrix (c) Class Representation Map



(d) Component Planes

Figure 44: The Frequency Plot, U-Matrix, Class Representation Map, and Component Planes of dataset 6

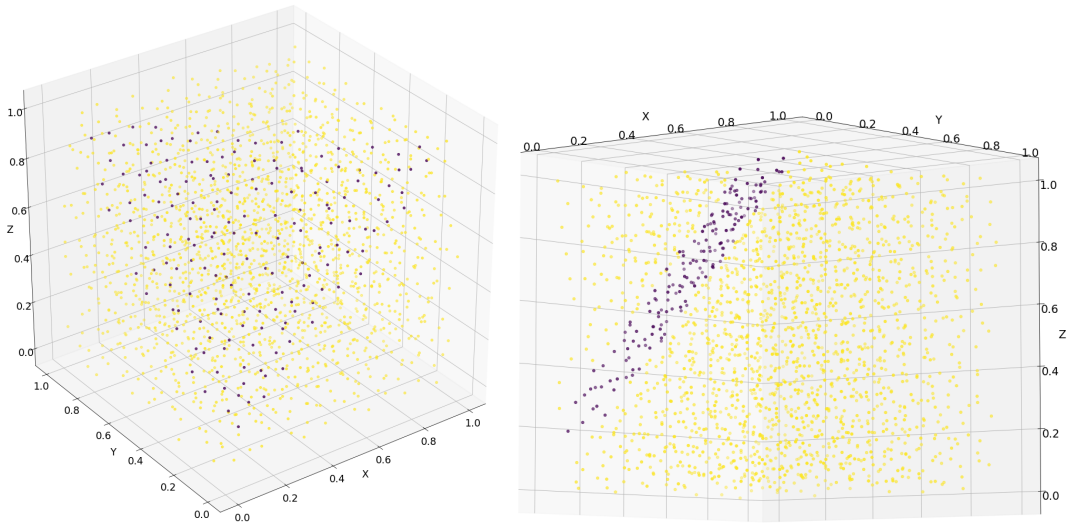


Figure 45: Two views of dataset 6. Note that the colors on the class representation map are reversed, they were switched for a cleaner 3-D viewing.

Dataset 6 was a 3-D dataset of 2000 best candidate sampled points in the unit cube. The labels are what made this dataset interesting, the purple points are slice of the cube for values that meet the criterion $0 < z - x - y < .2$. This ended up looking like a plane that cut off a corner of the cube. Our analysis was not entirely incorrect. We predicted that there was a plane that divided the cube but we missed that there was only one cluster of purple points. As previously discussed in Section 6, since this data spans the entire cube, it would be very difficult to get an accurate picture of the class distribution using a SOM with a 2-D lattice.