Self Organizing Map (SOM)
Stochastic Neighborhood Embedding (SNE)

Lecture # 16

Introduction

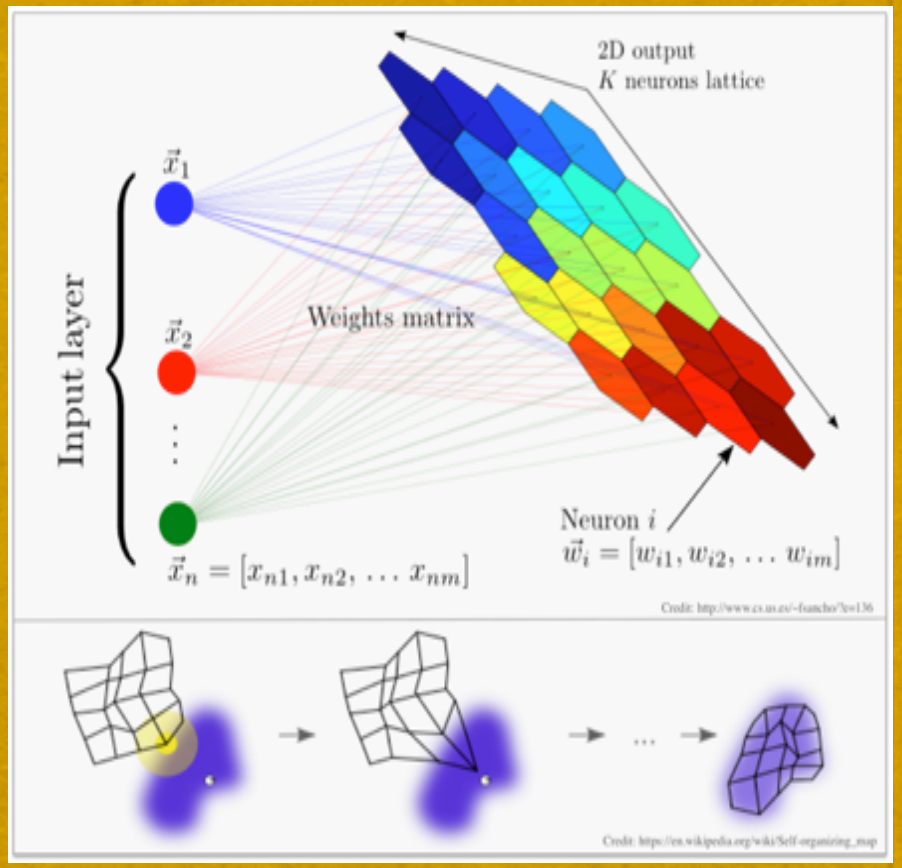
A self-organizing map (SOM) is a visualization technique. It is a form of neural network where the output is an organized visual matrix- a two-dimensional grid with rows and columns. The aim here is to transfer all the input data objects with n attributes (n dimensions) to the output lattice in such a way that objects next to each other are closely related. SOM is an unsupervised learning algorithm. It arranges the datapoints in a lower dimensional space, helping to visualize high dimensional data through a low-dimensional space.

The most common SOM output is a 2-dimensional grid with data objects placed next to each other based on their similarity to one another. SOMs differs from other clustering techniques because there is no specific clustering labels assigned to data objects. Data objects are grouped based on their attribute proximity, with the task of clustering left to visual analysis by the user.

Self Organizing Maps

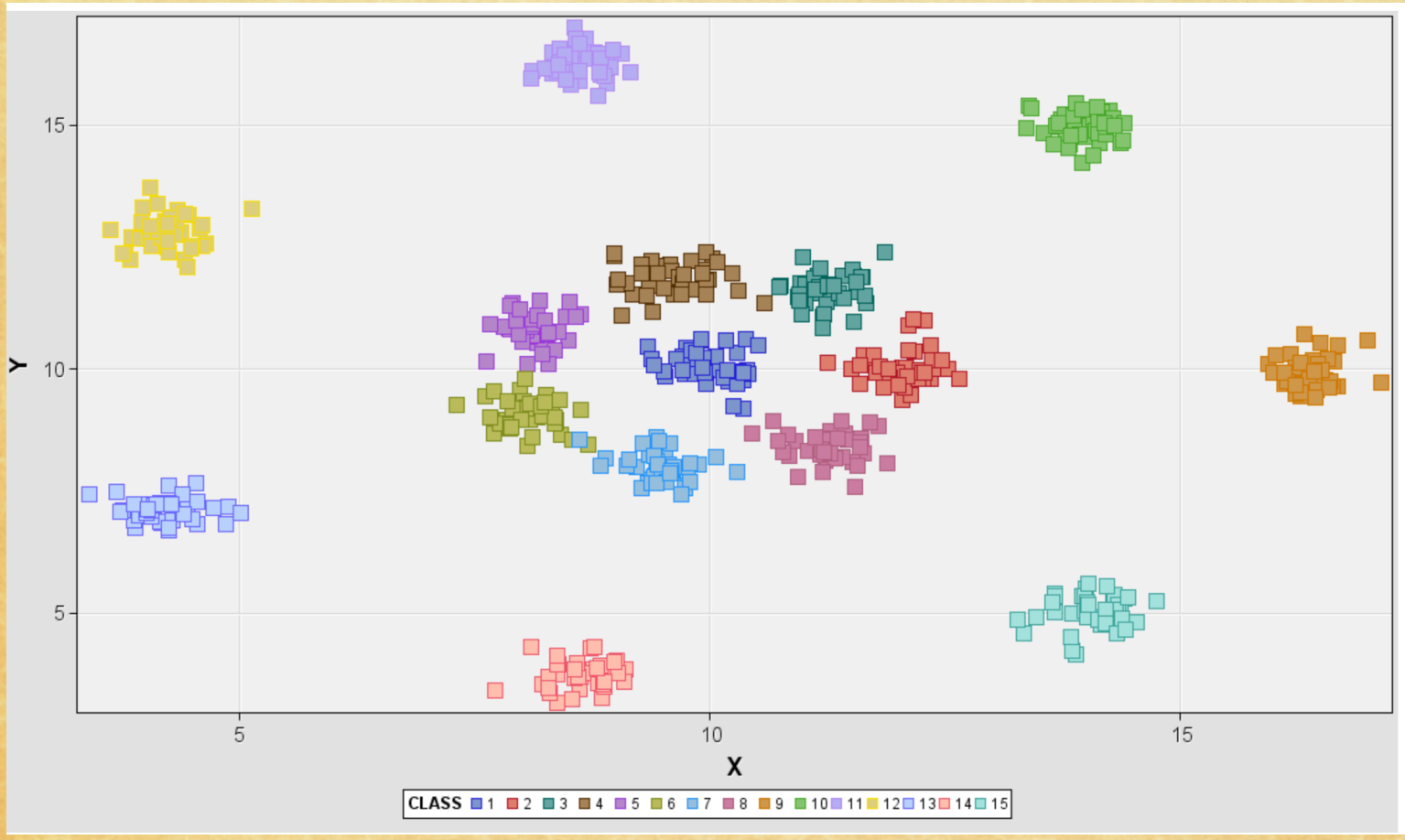
Introduced by Kohonen in 1980s, also known as Kohonen Network.

A class of unsupervised neural networks that reduce dimensions while preserving topology.



SOM is used to project data objects from data space, mostly in n dimensions, to grid space, usually in two-dimensions. Each datapoint occupies a cell or a node in the output lattice arranged by constrained depending on the similarity of the data. Each cell in the SOM grid (i.e. neuron) corresponds to one or a group of datapoints. The aim of the SOM is to compare relative features of the data objects in a simple 2-dimensional setting where the placement of the objects are related to one another.

Example of a self-organizing Map



How it Works

SOM is a neural network and therefore, the model only accepts numerical attributes. Also, since it is unsupervised, there is no target variable. The aim of an SOM algorithm is to find a set of centroids (neurons) to represent the cluster but with topological constraints. The topology refers to an arrangement of centroids in the output grid. All the data are assigned to centroids. The centroids closer to each other in the grid are more closely related. A SOM converts numbers from the data space to a grid space.

Step-by-Step Measurement

Step 1: Topology Specification

The first step is to specify the topology of the output. Often a two dimensional rows and columns with a hexagonal or rectangular lattice is used. Hexagonal architecture is preferable since each node can have six neighbors as compared to rectangular lattice that could only have four neighbors. As a result, in hexagonal lattice the association of a data point with another data point is more precise.

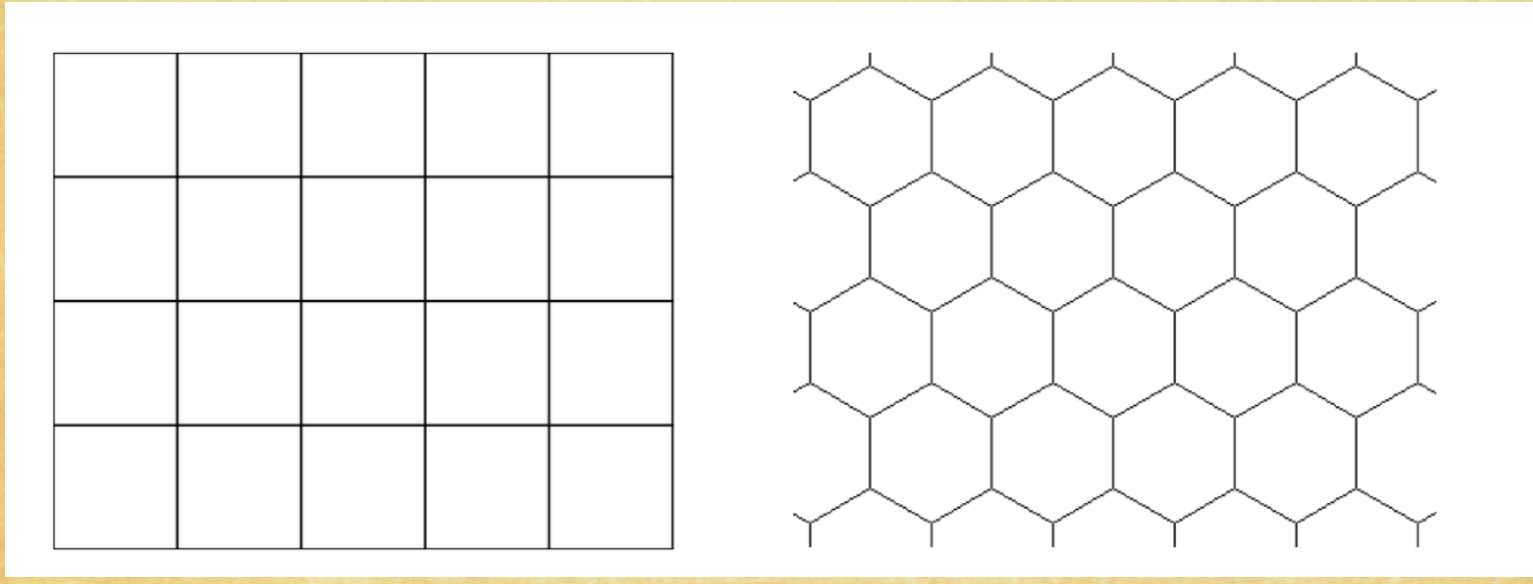
Step 2: Initialize Centroids

A SOM starts the process by initializing the centroids. The initial centroids are values of random data objects from the dataset.

Step 3: Assignment of Data Objects

After centroids are selected and placed on the grid in the intersection of rows and columns, data objects are selected one-by-one and assigned to the nearest centroid. The nearest centroid can be calculated using Euclidean distance for numeric data.

Topology: Hexagonal vs. Rectangular



Step 4: Centroid update

The first step here is to update the closest centroid. The aim is to update the data values of the nearest centroid of the data object, proportional to the difference between the centroid and the data object. The value of the centroids are updated based on the error difference between the predicted and actual values. Through this update, the closest centroid moves closer to the data object in the data space.

The centroid update step is repeated for a number of iterations. Consider the t'th iteration of the update where the data point d(t) is picked up. Let $w_1, w_2, \ldots, w_k$ represent the centroids in the grid space. Let r and c be the number of rows and columns respectively. Then k will be equal to $r * c$. Let w_i be the nearest centroid for a data object d(t). During iteration t, the nearest centroid w_i is updated

$$w_i(t+1) = w_i(t) + f_i(t) \times [d(t) - w_i(t)]$$

The effect of the update is the difference between the centroid and the data point in the data space and the neighborhood function $f_i(t)$. This function decreases after each iteration.

We then update all the centroids in the grid space neighborhood. The neighborhood update step is proportional to the distance from the closest centroid to the centroid that is being updated. The update function is stronger when the distance is closer. A Gaussian function is often used for this:

$$f_i(t) = \lambda_i \, e^{((g_i - g_j)^2 / 2\sigma^2)}$$

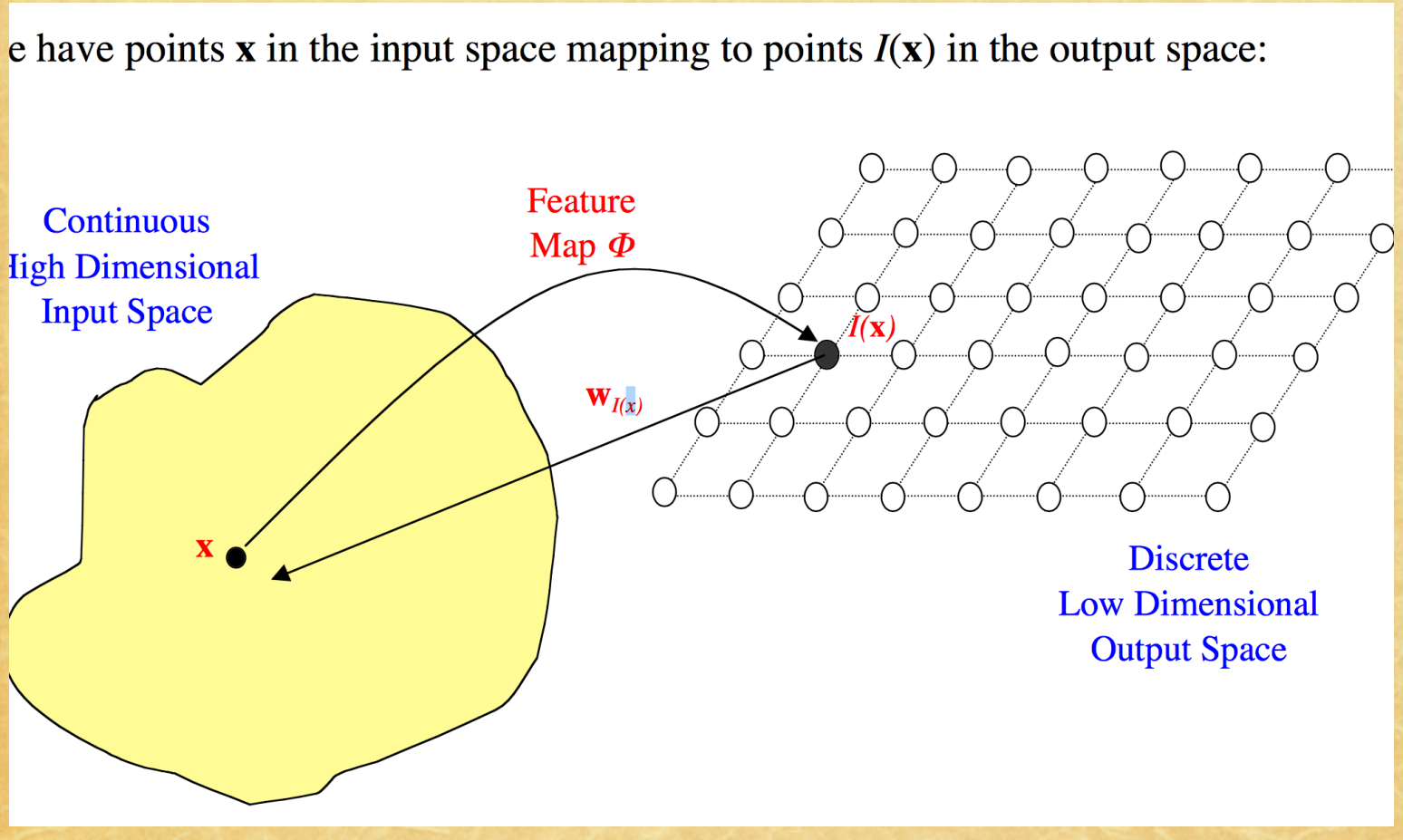
Where λ_i is the learning rate function that takes a value between 0 and 1 and decays for every iteration. This is often a linear rate function or an inverse of the time function. The variable $g_i - g_j$ is the distance between the centroid being updated and the nearest centroid of the data point in the grid space. σ_i is the radius of the centroid updated. By updating the entire neighborhood of centroids in the grid, the SOM self-organizes the centroid lattice.

Step 5: Termination

The algorithm is continued until no significant centroid updates take place or the specific number of run counts is reached. SOM tends to converge towards a solution in most cases

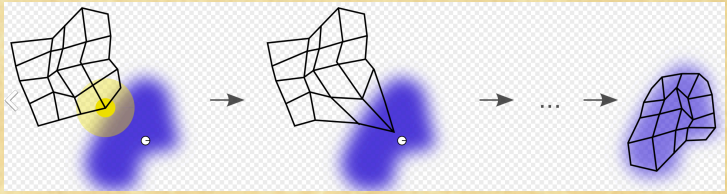
Point x in the input space maps to points I(x) in the output space. Each point I in the output space will map to point w(I) in the input space.

From *J. A. Bullinaria http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf*



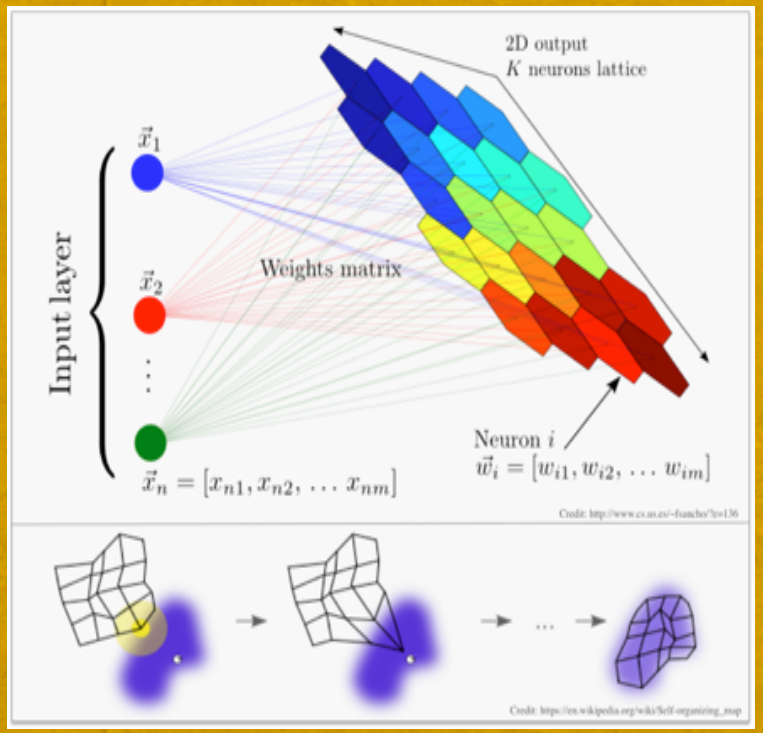
SOM Training

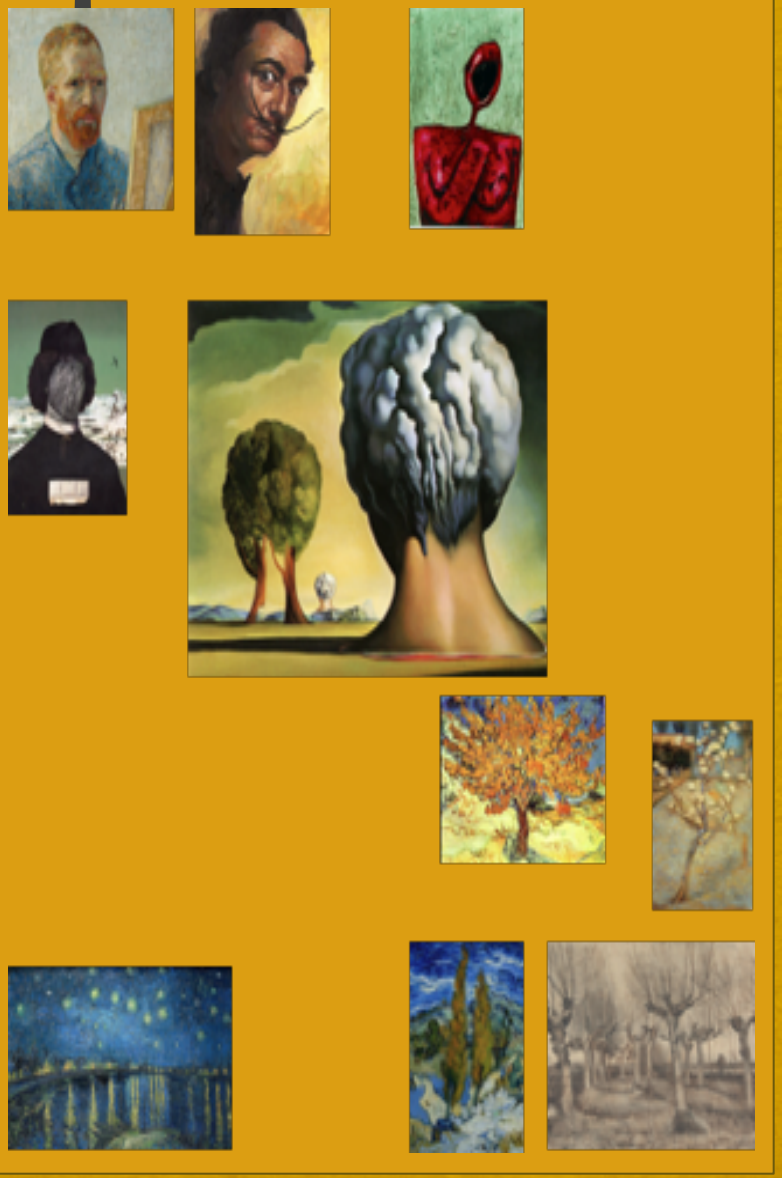
An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training datum drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node (highlighted in yellow) which is nearest to the training datum is selected. It is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).



Self Organizing Maps

A class of unsupervised neural networks that reduce dimensions while preserving topology.





MNIST with SOM

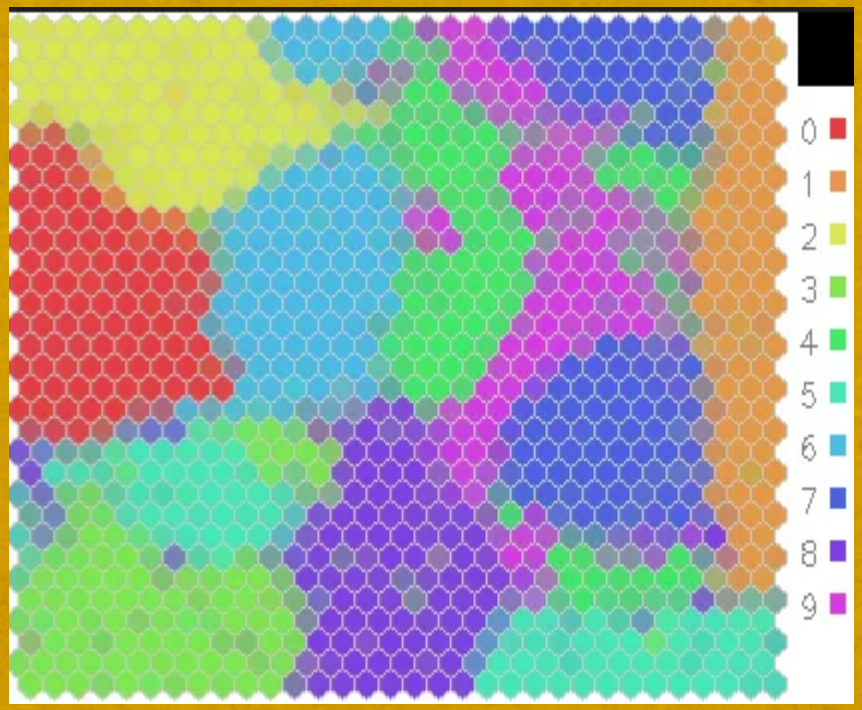
Advantage:

Map any new data to the trained SOM

Differentiates different ways of writing same number ...

Disadvantage:

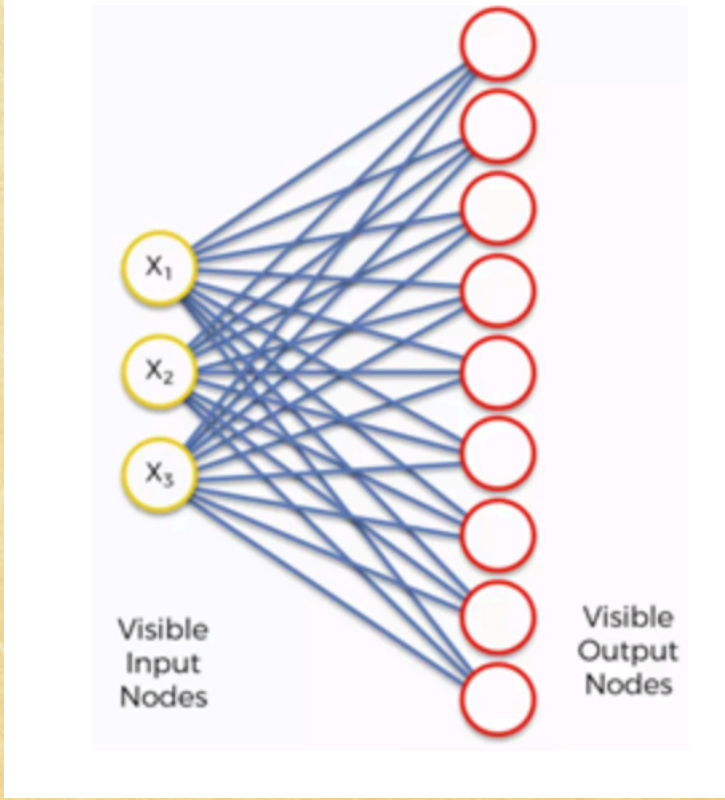
Does not preserve distance



Example of SOM

In the example we have 3 input nodes, if we had 20 the nodes would have 20 characteristics/weights. The SOM determines which nodes are closest to each row of data within the dataset using Euclidean Distance. It uses Best Matching Unit (BMU) to determine which node is the closest to the rows data.

When the SOM has determine what row the node belongs to, the node updates the nodes in a radius around it to move them closer to that node. With each epoch the radius around the nodes shrink, meaning less nodes are pulled towards it making your data more accurate.



Key points about SOM

- SOMs retain topology of the input set.
- SOMs reveal correlations that are not easily identified.
- SOMs classify data without supervision.
- They have no target vector which means no backpropagation.
- They have no lateral connections between output nodes.

Summary of different steps

- Step 1 - We start with a dataset composed of *n_features* independent variables.
- Step 2 - We create a grid composed of nodes, each one having a weight vector of *n features* elements.
- Step 3 - Randomly initialize the values of the weight vectors to small numbers close to 0 (but not 0).
- Step 4 - Select one random observation point from the dataset.
- Step 5 - Compute the Euclidean Distances from this point to the different neurons in the network.
- Step 6 - Select the neuron that has the minimum distance to the point. This neuron is called the winning node.
- Step 7 - Update the weights of the winning node to move it closer to the point.
- Step 8 - Using a Gaussian Neighbourhood function on the mean of the winning node and update the weights of the winning node neighbours to move them closer to the point. The neighbourhood radius is the sigma in the Gaussian function.
- Step 9 - Repeat steps 1 to 5 and update the weights after each observation (Reinforcement Learning) or after a batch of observations (Batch Learning), until the network converges to a point where the neighbourhood stops decreasing.

Sources used for this lecture

Data Science
Concepts and Practice
By Vijay Kotu and Bala Deshpande

Stochastic Neighbor Embedding (SNE)

Lecture # 17

“t-SNE” is a way of converting a high-dimensional data set into a matrix of pair- wise similarities and visualizing the resulting similarity data. t-SNE is capable of capturing much of the local structure of the high-dimensional data while also revealing global structure such as the presence of clusters at several scales.

Dimensionality reduction methods convert the high-dimensional data set $X = \{x_1,x_2,...,x_n\}$ into two or three-dimensional data $Y = \{y_1,y_2,...,y_n\}$ that can be displayed in a scatter plot.Here, we refer to the low-dimensional data representation Y as a map, and to the low-dimensional representations y_i of individual datapoints as map points. The aim of dimensionality reduction is to preserve as much of the significant structure of the high-dimensional data as possible in the low-dimensional map.

Stochastic Neighbor Embedding (SNE) converts the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities. The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. For nearby datapoints, $p_{j|i}$ is relatively high, whereas for widely separated datapoints, $p_{j|i}$ will be very small (for reasonable values of the variance of the Gaussian, σ_i). Mathematically, the conditional probability $p_{j|i}$ is given by

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

Where σ_i is the variance of the Gaussian centered on datapoint x_i. Because we are only interested in modeling pairwise similarities, we set the value of $p_{i|i}$ to zero. For the low-dimensional counterparts y_i and y_j of the high-dimensional datapoints x_i and x_j, it is possible to compute a similar conditional probability, which we denote by $q_{j|i}$. Let we set the variance of the Gaussian that was used in the conditional probability $q_{j|i}$ to $(1/2)^{-1/2}$. The similarity of the map point y_j to map point y_i is:

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}$$

Again, since we are only modelling pairwise similarities, $q_{i|i} = 0$.

If the map points y_i and y_j correctly model the similarity between the high-dimensional data- points x_i and x_j, the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal.

SNE aims to find a low-dimensional data representation that minimizes the mismatch between $p_{j|i}$ and $q_{j|I}$.

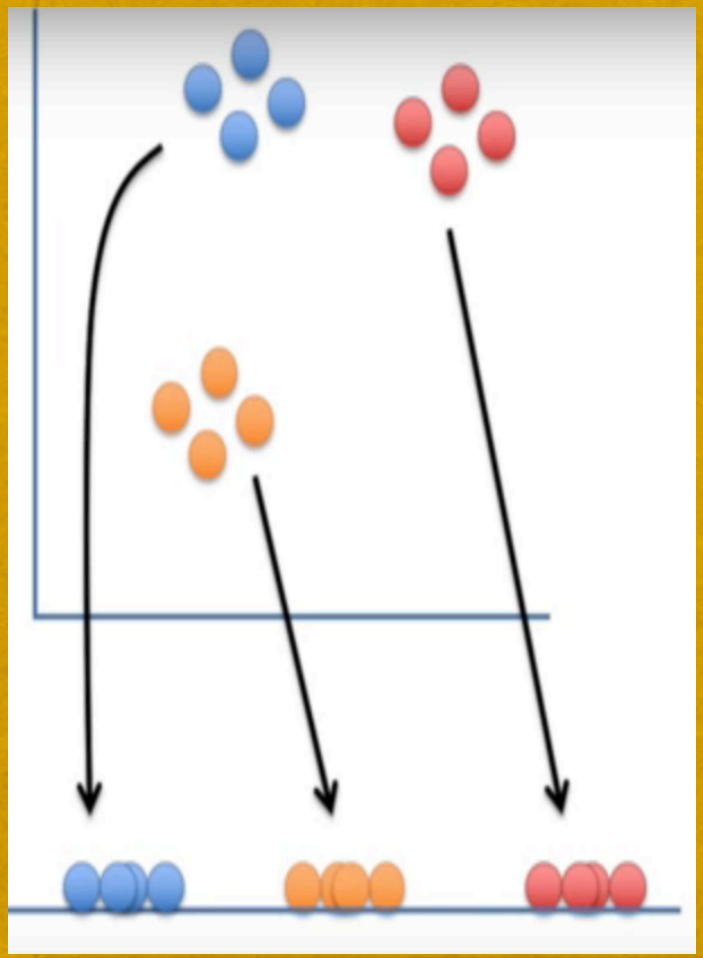
We need to minimize the "cost function" defined as the sum of Kullback-Leibler divergences over all datapoints

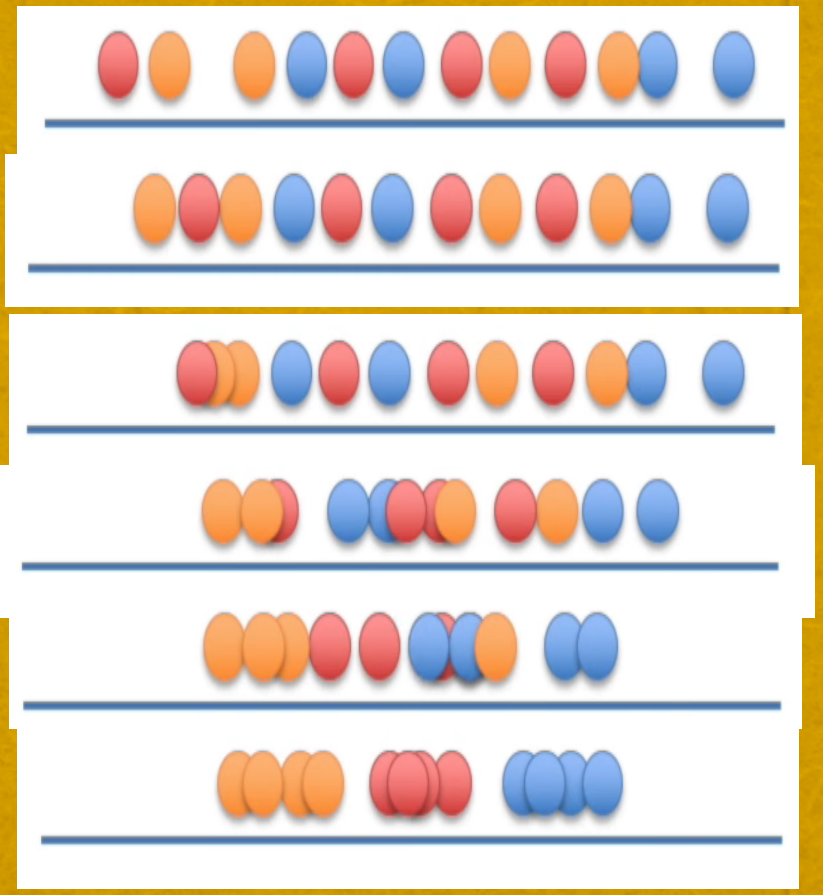
$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Where P_i is the conditional probability distribution over all other datapoints given datapoint x_i and Q_i is the conditional probability distribution over all other map points given map point y_i.

t-SNE

T-distributed Stochastic Neighbor Embedding





credit: statQuest youtube channel

Because the Kullback-Leibler divergence is not symmetric, different types of error in the pairwise distances in the low-dimensional map are not weighted equally. In particular, there is a large cost for using widely separated map points to represent nearby datapoints (i.e., for using a small $q_{i|j}$ to model large $p_{j|i}$) but is a small cost for using nearby map points to represent widely separated datapoints.

The remaining parameter to be selected is the variance σ_i of the Gaussian that is centered over each high-dimensional datapoint, x_i. It is not likely that there is a single value of σ_i that is optimal for all datapoints in the data set because the density of the data is likely to vary. In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions. Any particular value of σ_i induces a probability distribution, P_i, over all of the other datapoints.

SNE performs a search for the value of σ_i that produces a σ_i with a fixed perplexity that is specified by the user. The perplexity is defined by

$$Perp(P_i) = 2^{H(P_i)}$$

Where $H(P_i)$ is the Shannon entropy of P_i

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

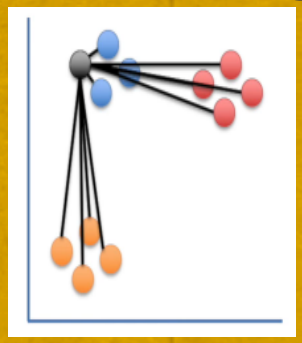
Perplexity is defined as a measure of the effective number of neighbors. The minimization of the cost function is performed using gradient descent method.

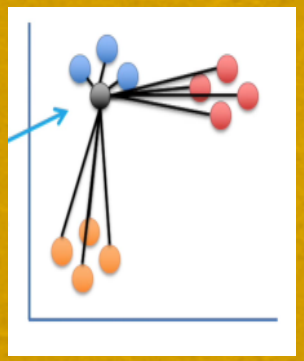
$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

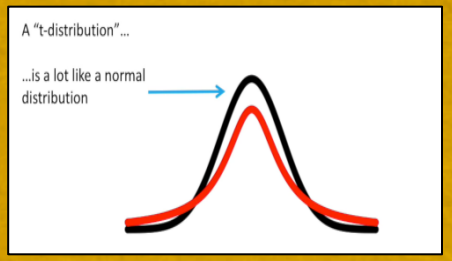
Minimization of this cost function allows measurement of these probabilities.

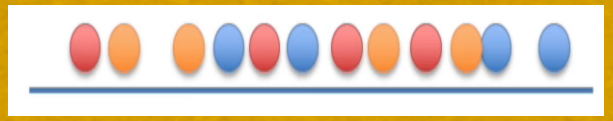
t-SNE

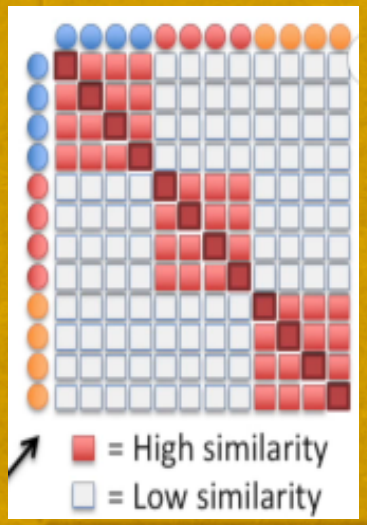
T-distributed Stochastic Neighbor Embedding

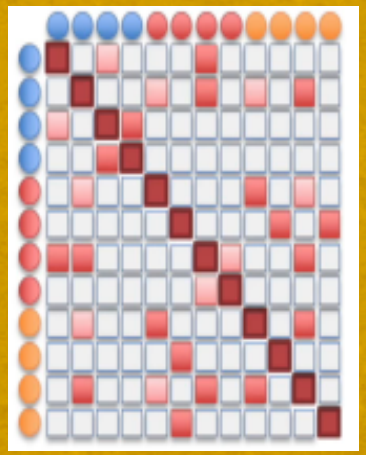












Perplexity

In machine learning perplexity is a measurement as how well a probability distribution predicts a sample. It is used to compare different probability models. A low perplexity indicates the probability distribution is good at predicting the sample. The perplexity of a discrete probability distribution p is defined as

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

Where H(p) is the entropy (in bits) of the distribution and x ranges over events. The base need not be 2. Perplexing is independent of the base provided that the entropy and the exponentiation use the same base. In the case where p models a fair k-sided die, its perplexity is k. A random variable with perplexity k has the same uncertainty as a fair k-sided die- k-ways perplexed about the value of the random variable.

t-distribution Stochastic Neighbor Embedding (t-SNE)

There are two serious problems with SNE (1). Its cost function that is difficult to optimize; (2). Crowding. To avoid these, t-SNE is introduced. The cost function used in t-SNE differs from the one used in SNE in two ways: (a). It uses a symmetric version of SNE cost function with simpler gradient; (b). It uses a t-student distribution rather than a Gaussian to compute the similarities between the two points in low dimensional space. It uses a heavy-tailed distribution in the low-dimensional space to alleviate both the crowding problem and the optimization problems of SNE.

In the above relation for cost function, the symmetric means pij = pji and qij = qji. The gradient of symmetric SNE is expressed as:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

Crowding problem

Definition:

Consider a set of datapoints that lie on a two-dimensional curved manifold which is approximately linear on a small scale, and which is embedded within a higher-dimensional space. It is possible to model the small pairwise distances between datapoints fairly well in a two-dimensional map. Now suppose that the manifold has ten intrinsic dimensions and is embedded within a space of much higher dimensionality. There are several reasons why the pairwise distances in a two-dimensional map cannot faithfully model distances between points on the ten-dimensional manifold. For instance, in ten dimensions, it is possible to have 11 datapoints that are mutually equidistant and there is no way to model this faithfully in a two-dimensional map.

Source used for this lecture

Laurens van der Maaten and Geoffrey Hinton

"Visualizing Data using t-SNE"

Journal of Machine Learning Research 9 (2008) 2579-2605