MATHEMATICAL INSIGHTS OF T-SNE AND UMAP

Explanation of Mathematics involved in two non-linear dimensionality reduction techniques for unsupervised learning

Team Members:
Aditya Krishna Das
Yasaswani Rongali
Yash Kumar

WHAT WILL THIS PRESENTATION COVER?

01 **DIMENSIONALITY REDUCTION: WHAT AND WHY?**

02 **PRINCIPAL COMPONENT ANALYSIS**

Understanding in brief about one of the most widely used DR techniques

03 **t-SNE**

Understanding steps involved in t-SNE

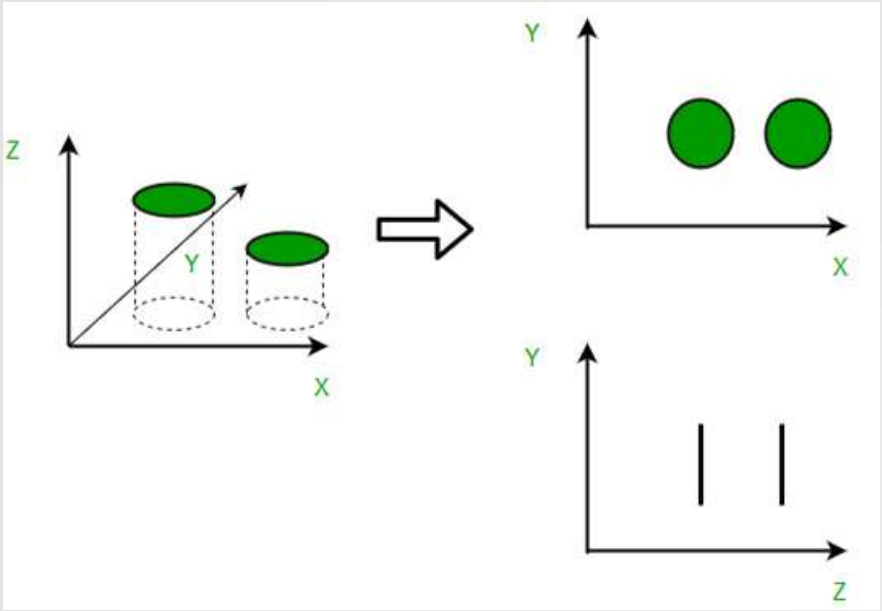
04 **UNDERSTANDING UMAP**

A dig into UMAP theory

01

DIMENSIONALITY REDUCTION: WHAT AND WHY?

WHAT IS DIMENSIONALITY REDUCTION?



Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables.

In simpler words, Dimensionality reduction is nothing but the reduction of n dimension data to n’ dimension data, where n > n’.

WHY USE DIMENSIONALITY REDUCTION?

- **Saves computational resources:**

  As dimensionality reduction reduces the training time of models by simplifying calculations, the need for computational resources to train those models will be very low.

- **Visualization of high-dimensional data:**

  When we reduce the dimensionality of higher dimensional data into two or three components, then the data can easily be plotted on a 2D or 3D plot.

- **Mitigate the problem of overfitting:**

  Dimensionality reduction finds a lower number of variables or removes the least important variables from the model.

02

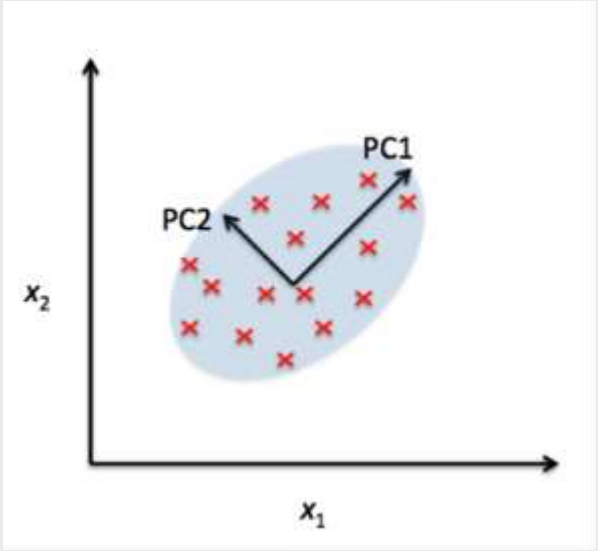
PCA

Understanding in brief about one of the most widely used DR techniques

PRINCIPAL COMPONENT ANALYSIS

The idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

1. Standardize the Data
2. Computation of covariance matrix.
3. Computation of the eigen values and eigen vectors of the covariance matrix to identify the principle components



PCA

Let X be a m cross n matrix

Where m is the number of data points and n is the number of dimensions

$XP=Y$

Here P is the transformation matrix and Y is the transformed dataset which has lesser number of dimensions.

STEP 1:

The data is standardized.

$$x_{new} = \frac{x - \mu}{\sigma}$$

STEP 2:

Covariance matrix is found using the formula:

$X^T X/m$

STEP 3:

Eigen values and Eigen vectors of the covariance matrix are calculated.

03

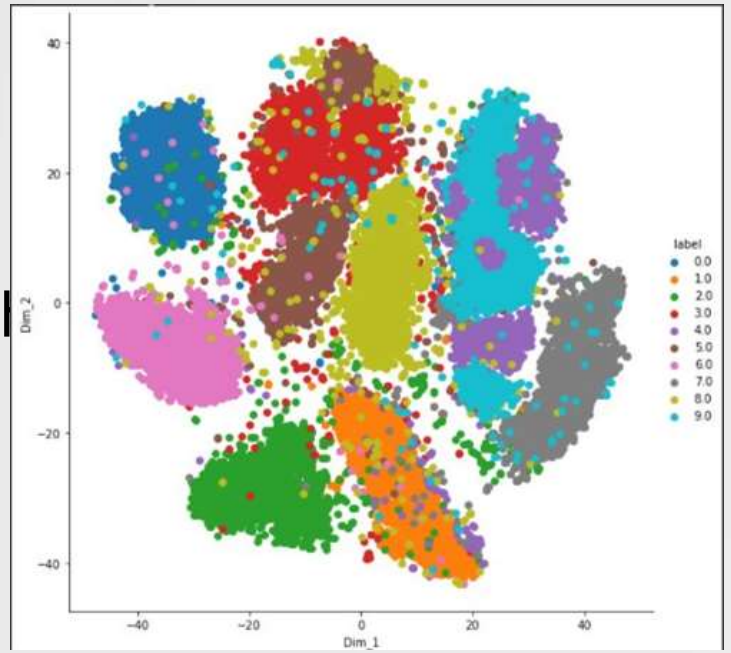
t-SNE

Understanding steps involved in t-SNE

t-DISTRIBUTED **S**TOCHASTIC **N**EIGHBOUR **E**MBEDDING

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.

In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high dimensional space.

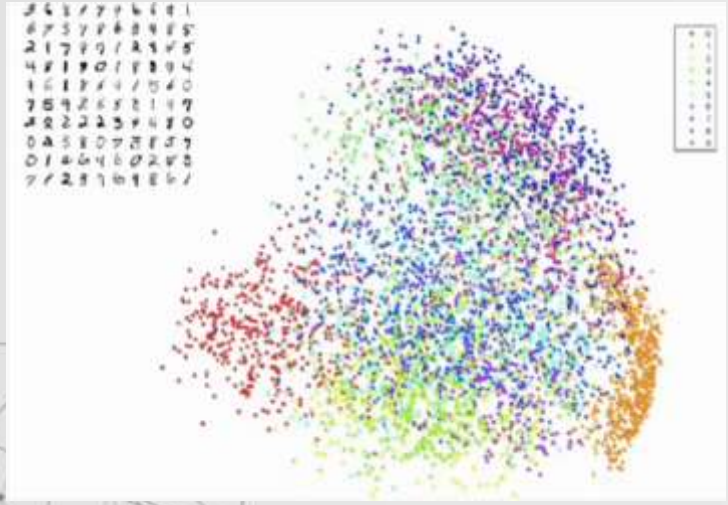


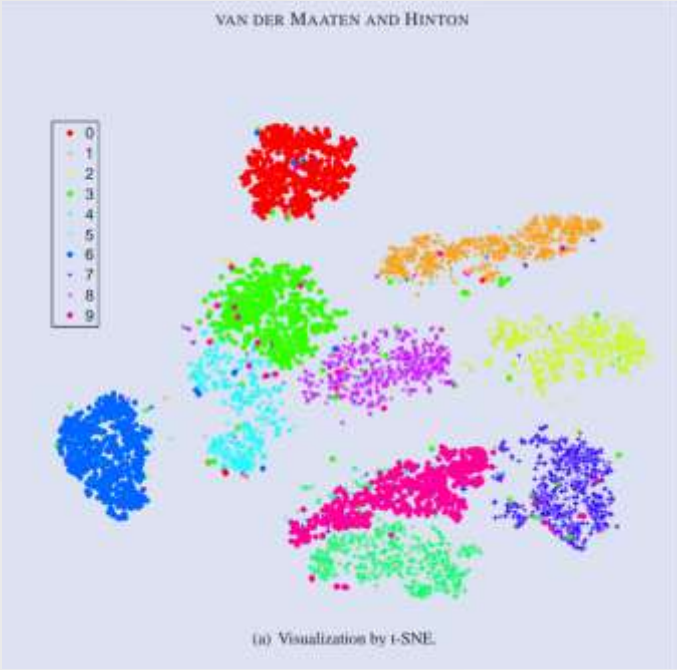
Why is t-SNE better than PCA?

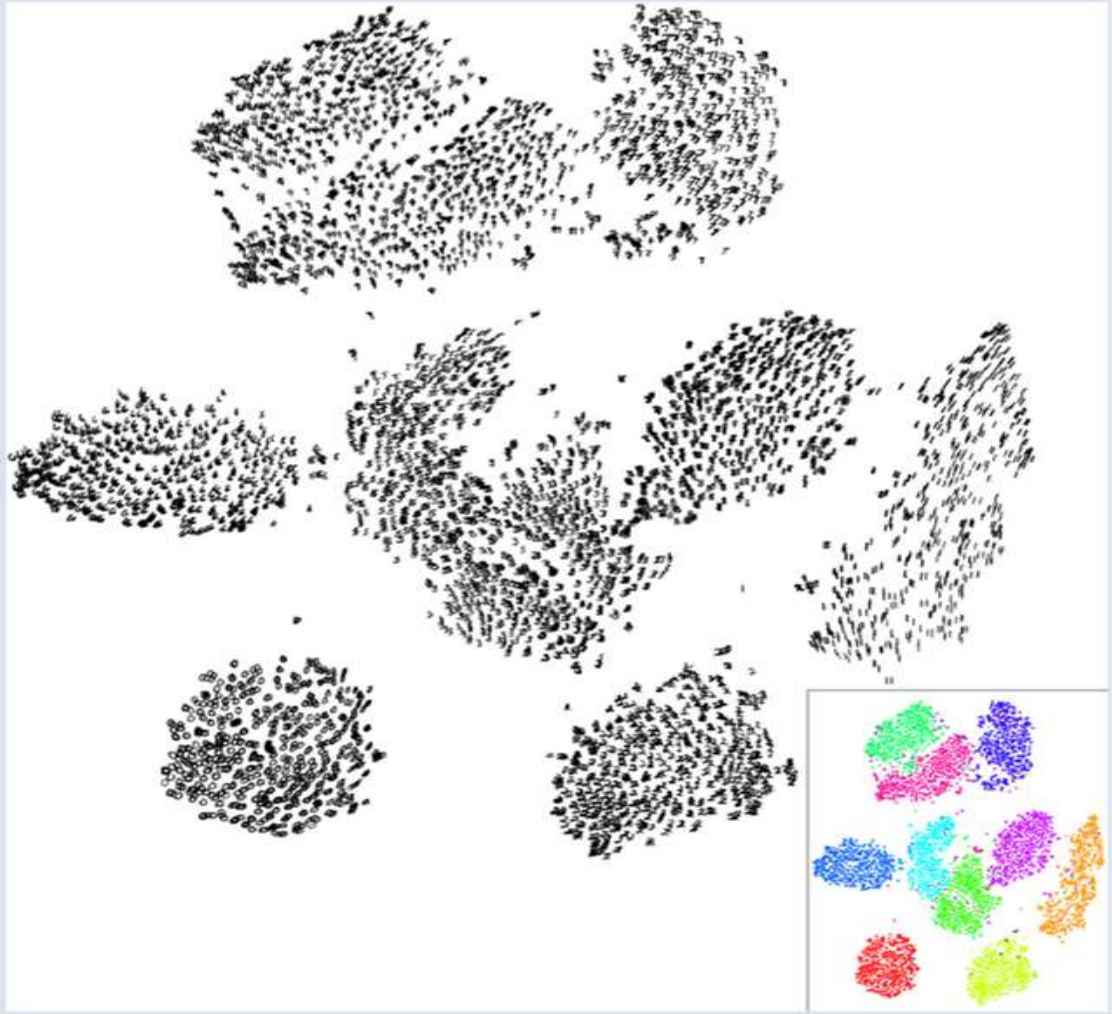
t-SNE can handle outliers whereas PCA is highly influenced by outliers present in the data.

t-SNE does a better job (it tries to preserve topology neighbourhood structure) as compared to PCA when it comes to visualising the different patterns of the clusters.

Visualization by PCA







Algorithm of t-SNE

Calculating **conditional probability distribution** of pairs of points from high dimensional data

Computing **joint probability distribution** by taking the average of conditional probability distribution for a pair of points

Creating a dataset of points in the target dimension and then calculating the **low dimensional joint probability distribution** for them.

Computing **Cost Function**

Using Gradient Descent and momentum to **move the points in the low dimensional graph** after each iteration.

CALCULATION OF CONDITIONAL PROBABILITY AND TAKING THEIR AVERAGE

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

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 .

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

n=number of higher dimensions

If we sum p_{ij} for all points x_j other x_i, then for that point x_i, the sum would be equal to 1. By this, we get σ_i.

https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

COMPUTING LOW DIMENSIONAL JOINT PROBABILITY DISTRIBUTION

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

The low dimensional graph is initialised randomly and the low dimensional joint probability distribution is calculated using the given formula.
The expression on the RHS is a Student t-distribution expression.

COMPUTING COST FUNCTION

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$
$$= \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij}.$$

Cost Function basically calculates the sum of difference of high dimensional and low dimensional joint probability distribution scores of all pairs of points.

https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

USING GD AND MOMENTUM TO UPDATE LOCATION OF POINTS IN THE LOW DIMENSIONAL GRAPH

Calculation of low dimensional joint probability distribution, gradient descent and the low dimensional data representation set is done in each iteration as after each iteration, position of all points change as they move towards their cluster and away from others.

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}.$$

→ Gradient Descent

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right),$$

→ Set containing low dimensional representation of data

https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

04

UNDERSTANDING UMAP

A dig into UMAP theory

UNIFORM **M**ANIFOLD **A**PPROXIMATION AND **P**ROJECTION

UMAP is a new dimensionality reduction technique that offers a number of advantages over t-SNE, most notably **increased speed** and **better preservation of the data's global structure**.
UMAP, at its core, works very similarly to t-SNE - both use graph layout algorithms to arrange data in low-dimensional space.
In the simplest sense, UMAP

1. Constructs a high dimensional graph representation of the data
2. Optimizes a low-dimensional graph to be as structurally similar as possible.

1. CALCULATION OF SIMILARITY SCORES

The first thing that UMAP does is calculate the distance between each pair of high dimension points. After this, the similarity scores are calculated by the formula:

$$p_{i|j} = e^{-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}}$$

$d(x_i, x_j)$=Distance (need not be Euclidean distance) between points i and j

ρ_i=Distance of point i from its Nearest Neighbour

The value of $\boldsymbol{\sigma}_i$ is chosen such that

$\sum p_{i|j}$ (for j=1 to n)=$\log_2(n)$, where n=no. of nearest neighbours.

Formula used in t-SNE:

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

UMAP scales the curve so that regardless of how close or far the neighbouring points are, the sum of similarity scores will be equal to $\log_2$(number of nearest neighbours).

https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668

2. SYMMETRIZATION OF SIMILARITY SCORES

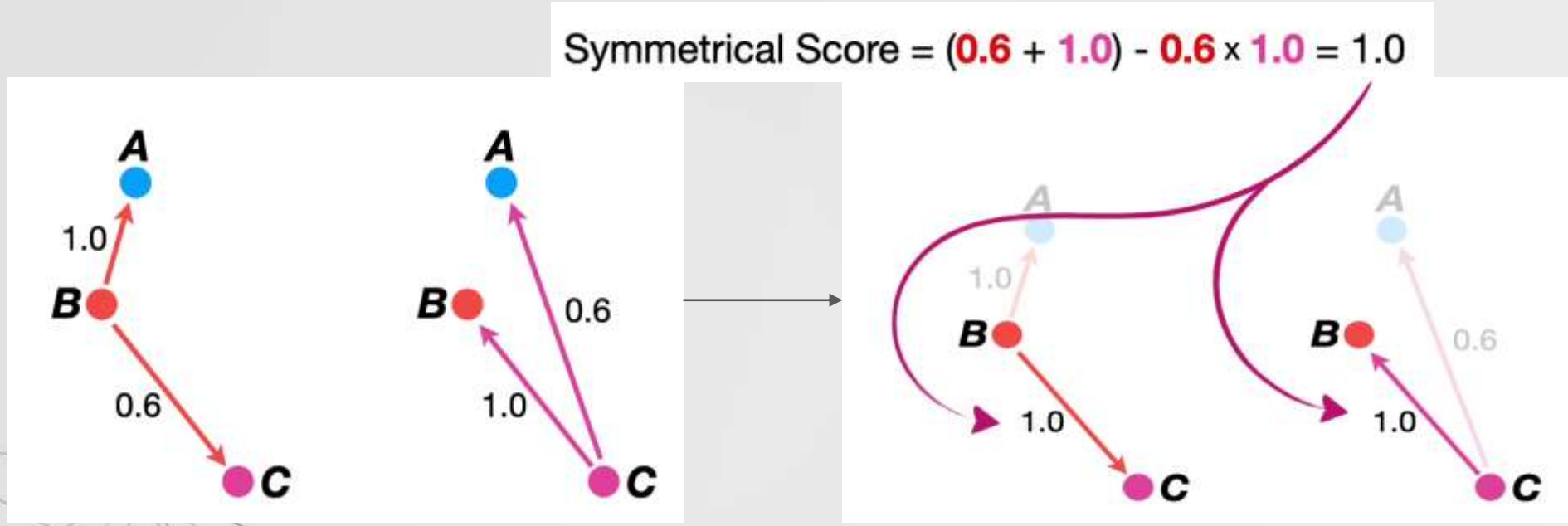
Since, there should be a unique similarity score for a pair of points, we apply symmetrization to $p_{i|j}$ and $p_{j|i}$ by using the formula given below.

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$

These symmetric similarity scores are used for deciding the order in which points are chosen to be moved to form a cluster. Pair of points having higher symmetric similarity score are given preference.

2. SYMMETRIZATION OF SIMILARITY SCORES

Example of symmetrizing HD similarity scores of points B and C.



3. LOW DIMENSIONAL(LD) SIMILARITY SCORES

NOTE: **p and x refer to high dimension and q and y refer to low dimension**

For calculating the low dimensional similarity scores of a pair of points, we use the formula:

$$q_{ij}=1/(1+a|y_i-y_j|^{2b})$$

The UMAP defaults use min_dist = 0.1, spread = 1, which results in a=1.577 and b=0.8951. If you use min_dist = 0.001, spread = 1 then you get the result for a=1.929 and b=0.7915.

WHY ARE LOW-DIMENSIONAL (LD) SIMILARITY SCORES USED?

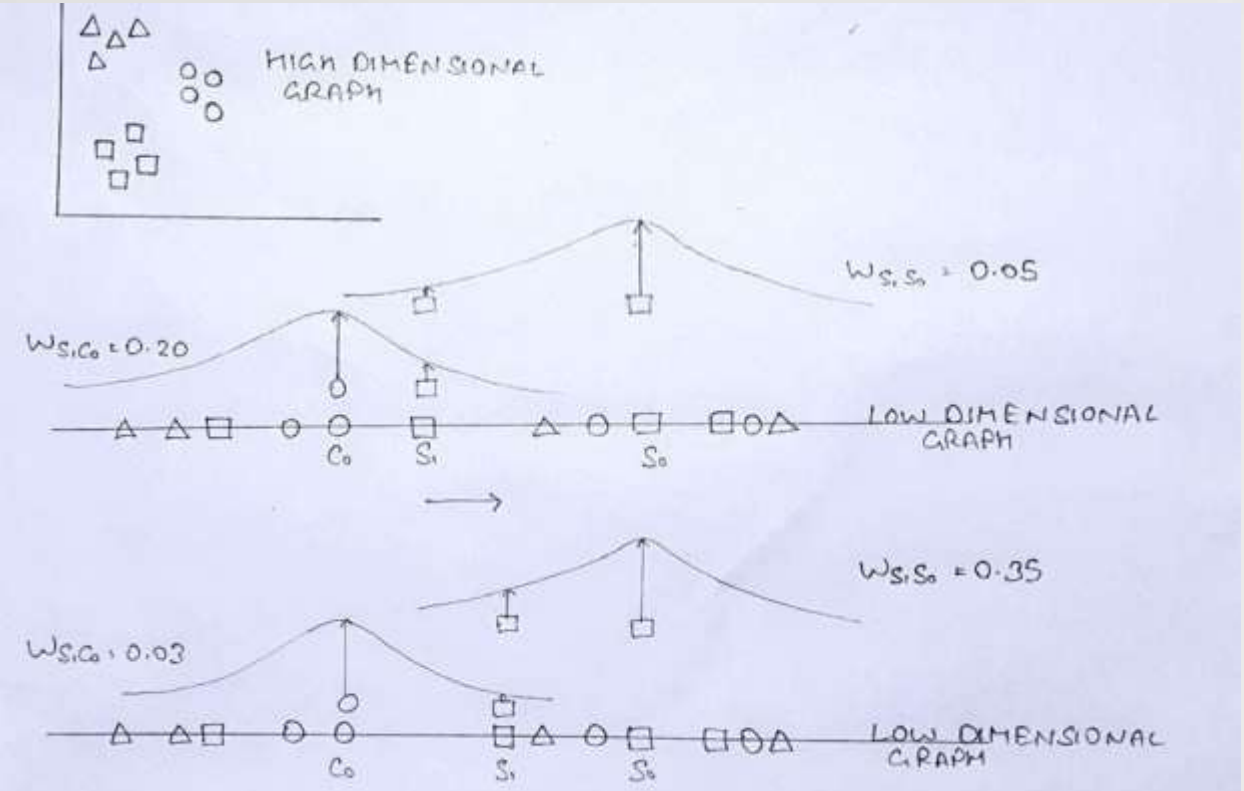
To form the clusters in the low dimensional graph.

AND HOW?

LD similarity scores are used to know whether or not a point is in a proper position in the LD graph.
If a pair of points are **neighbours in the HD graph**, then they **should have high LD similarity scores**. So, after the LD graph is initialized UMAP brings neighbour points close to each other and this is judged by their similarity scores, that is, if their LD similarity score is increasing the points are getting closer. Similarly for pair of **points that are not neighbours**, UMAP tries to **reduce their LD similarity scores** and this is how clusters are formed.

AND HOW?

In this example, a 2D graph is converted to 1D by being randomly initialised (just for the explanation here) and point S_1 is moved towards its neighbour S_0 and away from its not neighbour C_0.
(The LD joint probability values are random values used only for explanation.)



SPECTRAL EMBEDDING

In t-SNE the low dimensional graph is randomly initialised but in UMAP, the low dimensional graph is initialised using **Spectral Embedding**.

The Spectral Embedding (Laplacian Eigenmaps) algorithm consists of three stages:

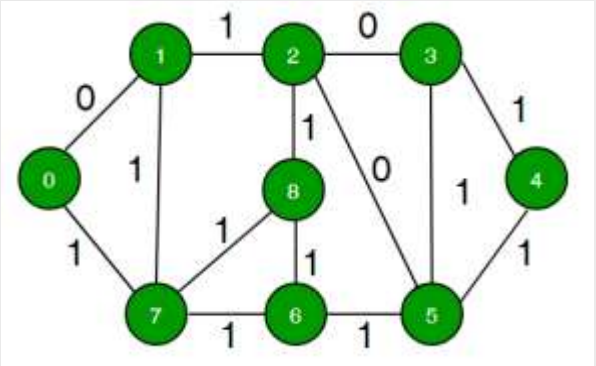
1. Constructing the Adjacency Graph
2. Choosing the Weights
3. Obtaining the Eigenmaps

STEP 1: The first step is to construct an adjacency graph based on the given data. We put an edge between nodes i and j if the corresponding data-points are “close”. In UMAP two points are said to be close if one of them is among the N-nearest neighbours of the other.

https://medium.com/mlearning-ai/demystifying-spectral-embedding-b2368bba580

SPECTRAL EMBEDDING

STEP 2: While assigning weights we make **W_{ij} = 1 if vertices i and j are connected by an edge** (that is, if they are among their nearest neighbours), otherwise put $W_{ij} = 0$.



STEP 3: After the second step, we will have the weight matrix (W) with us. Using W, we will obtain the diagonal weight matrix (D), whose entries are column (or row, since W is symmetric) sums of W, i.e., $D_{ii} = \sum_j W_{ji}$. Once we have obtained D, we will obtain the Laplacian Matrix (L), where L = D-W.

$L f_0 = \lambda_0 D f_0$
$L f_1 = \lambda_1 D f_1$
…
$L f_{k-1} = \lambda_{k-1} D f_{k-1}$
where, $f_0, f_1 \ldots f_{k-1}$, represent the eigenvectors to this problem, ordered according to their eigenvalues, i.e., $0 = \lambda_0 \leq \lambda_1 \ldots \leq \lambda_{k-1}$.

From these k eigenvectors, we leave out the eigenvector f_0 corresponding to eigenvalue 0, and use the next m eigenvectors for obtaining the **lower m-dimensional representations, i.e., $x_i = [f_1(i), \ldots f_m(i)]$**

TWO IMPORTANT PARAMETERS OF UMAPs

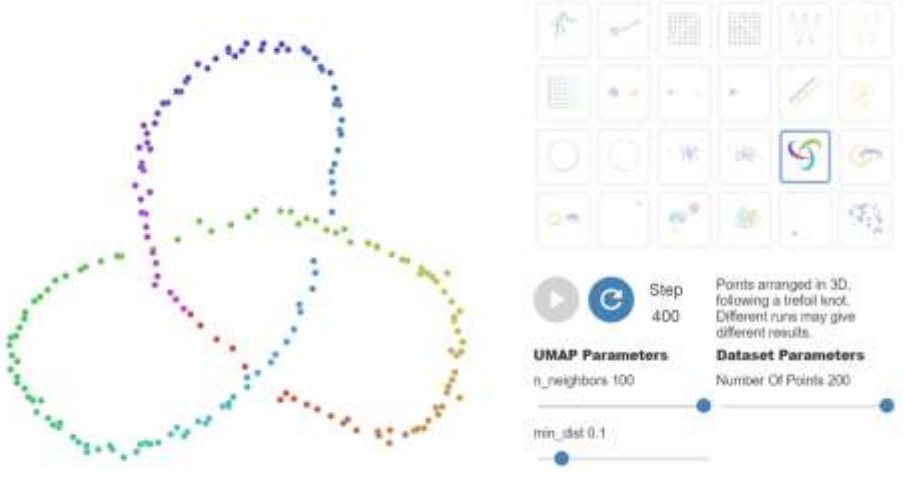
We'll consider the two most commonly used parameters: `n_neighbors` and `min_dist`, which are effectively used to control the balance between local and global structure in the final projection.

NUMBER OF NEAREST NEIGHBOURS

The most important parameter is `n_neighbors` - the number of approximate nearest neighbors used to construct the initial high-dimensional graph. It effectively controls how UMAP balances local versus global structure - **low values will push UMAP to focus more on local structure** by constraining the number of neighboring points considered when analyzing the data in high dimensions, while **high values will push UMAP towards representing the big-picture structure** while losing fine detail.

MINIMUM DISTANCE

The second parameter we'll investigate is `min_dist`, or the minimum distance between points in low-dimensional space. This parameter controls how tightly UMAP clumps points together, with **low values leading to more tightly packed embeddings**. **Larger values of `min_dist` will make UMAP pack points together more loosely**, focusing instead on the preservation of the broad topological structure.



CODE FOR t-SNE

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
import scipy as sp
from keras.datasets import mnist
(X_train, y_train), _ = mnist.load_data()
np.shape(X_train)
data = np.reshape(X_train,(len(X_train),784))
labels=y_train
data,labels
from sklearn.preprocessing import StandardScaler
std_data = StandardScaler().fit_transform(data)
print(std_data.shape)
```

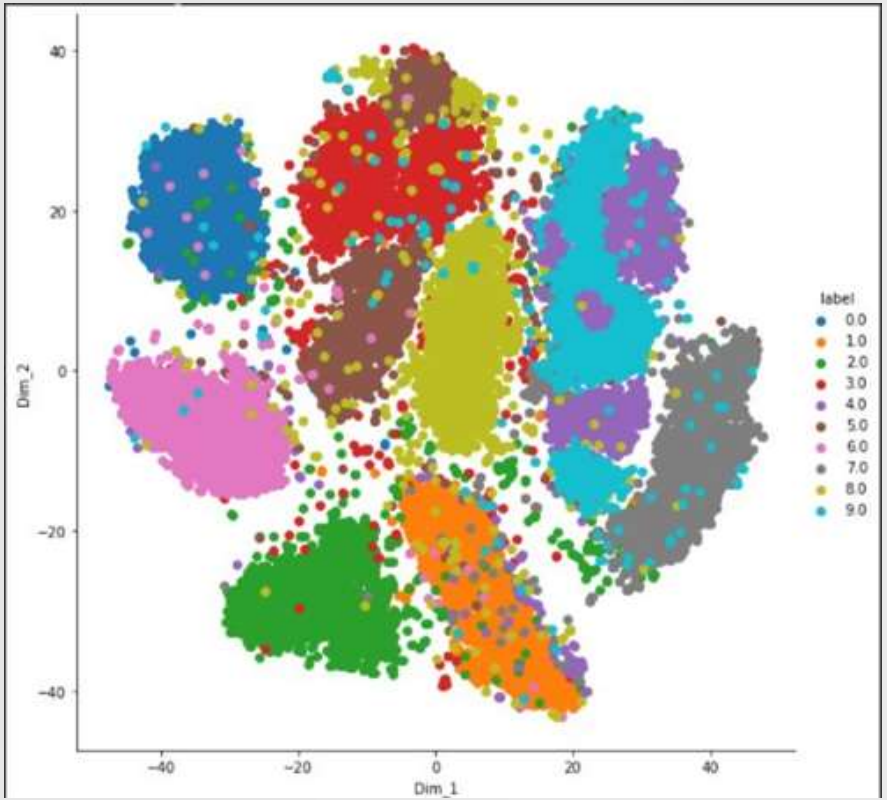
```
from sklearn.manifold import TSNE

model =
TSNE(n_components=2,perplexity=100,
learning_rate=1000,n_iter=500,
random_state=0)

tsne_data =
model.fit_transform(data[:25000])

tsne_data = np.vstack((tsne_data.T,
labels[:25000])).T
tsne_df = pd.DataFrame(data=tsne_data,
columns=("Dim_1", "Dim_2", "label"))
sn.FacetGrid(tsne_df, hue="label",
height=8).map(plt.scatter, 'Dim_1',
'Dim_2').add_legend()
plt.show()
```

OUTPUT OF t-SNE ON MNIST DATA



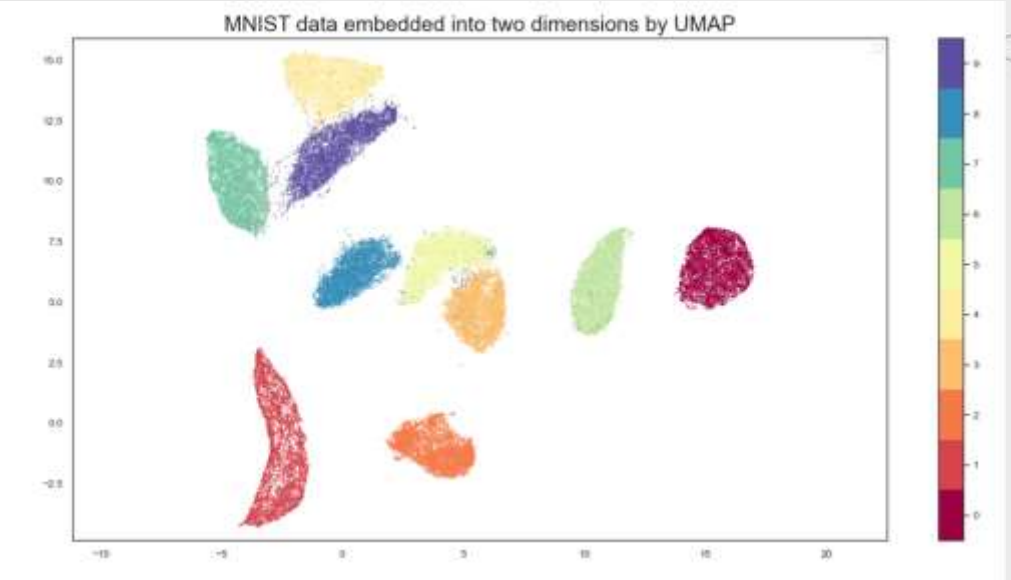
CODE FOR UMAP

```
import umap.umap_ as umap
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set(context="paper", style="white")

mnist = fetch_openml("mnist_784", version=1)

reducer = umap.UMAP(random_state=42)
embedding = reducer.fit_transform(mnist.data)

fig, ax = plt.subplots(figsize=(12, 10))
color = mnist.target.astype(int)
plt.scatter(embedding[:, 0], embedding[:, 1], c=color, cmap="Spectral", s=0.1)
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title("MNIST data embedded into two dimensions by UMAP", fontsize=18)
plt.legend()
plt.show()
```



COMPARISON OF TWO COST FUNCTIONS AND THEIR EFFECT ON LOCAL AND GLOBAL STRUCTURE

COST FUNCTION OF t-SNE

NOTE: p and x refer to high dimension and q and y refer to low dimension

1) $$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij}.$$

2) $$P(X) \approx e^{-X^2} \qquad Q(Y) \approx \frac{1}{1+Y^2}$$

3) $$KL(X,Y) \approx -P(X) \log Q(Y) = e^{-X^2} \log(1+Y^2)$$

COST FUNCTION OF UMAP:

4) $$CE(X,Y) = \sum_i \sum_j \left[p_{ij}(X) \log \left(\frac{p_{ij}(X)}{q_{ij}(Y)} \right) + (1 - p_{ij}(X)) \log \left(\frac{1 - p_{ij}(X)}{1 - q_{ij}(Y)} \right) \right]$$

5) $$X \to 0 : CE(X,Y) \approx \log(1+Y^2)$$

6) $$X \to \infty : CE(X,Y) \approx \log \left(\frac{1+Y^2}{Y^2} \right)$$

https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668

THANKS