Multidimensional Scaling Using Artificial Neural Networks

A Thesis submitted by
Partha Sil

In the partial fulfillment of the requirements for the degree of
MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE

Under the guidance of
Dr. RAJAT KUMAR DE
Professor
Machine Intelligence Unit
Dr. SMARAJIT BOSE
Professor
Interdisciplinary Statistical Research Unit

Indian Statistical Institute, Kolkata

July, 2021

CERTIFICATE

This is to certify that the dissertation entitled “**Multidimensional Scaling Using Artificial Neural Networks**" submitted by **Partha Sil** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of the institute and, in my opinion, has reached the standard needed for submission.

Rajat Kumar De   [illegible]

Rajat Kumar De
Professor,
Machine Intelligence Unit,

Smarajit Bose
Professor,
Interdisciplinary Statistical Research Unit,

Indian Statistical Institute,
Kolkata-700108, INDIA

Acknowledgement

I would like to express my heartiest gratitude to my dissertation supervisor Dr. Rajat Kumar De for stretching the hand of constant and graceful assistance towards me, throughout the dissertation time-span. Without his mentor-ship and guidance it would be rather impossible for me to make this whole accomplishment done. Would also like thank my co-guide Dr. Samarjit Bose for helping me to build a sound understanding of the underlying mathematics of the model.

Partha Sil.

Partha Sil
M.Tech in Computer Science
Indian Statistical Institute
Kolkata - 700108 , India

Abstract

Manifold Learning has been widely exploited in the arena of data analysis, machine learning and pattern recognition. The main assumption behind manifold learning is that the input high-dimensional data lies intrinsically on a low-dimensional manifold. This technique is to be used for non-linear dimensionality reduction. Although there are very well known dimensionality reduction techniques are already designed such as Principal Component Analysis (PCA), Independent Component Analysis, Linear Discriminant Analysis, and others but they are unable to capture the non linear structure of the data so that researchers are interested in this area. After that many manifold learning algorithms are developed such as Multidimensional Scaling (MDS), Locally linear embedding (LLE), Hessian Eigenmapping, t-distributed Stochastic Neighbor Embedding (t-SNE) etc. Multidimensional Scaling is one of them that seeks vectorial representation of the data points given the pairwise distance between the data points.There are two variant of Multidimensional Scaling one is metric Multidimensional Scaling and other is non-metric Multidimensional Scaling. Our interest on metric Multidimensional Scaling. The methodologies that are available to implement classical metric-MDS boil down to finding eigen values and eigen vectors of some matrix and which is computationally difficult for large dimensional matrix that motivate us to implement it using neural network setup. We are implementing it using Artificial Neural Networks and experiment it on famous Iris and Wine datasets and compare our results with some existing methods on few other datasets also.

Keywords: *Artificial Neural Networks, MDS, LLE, Sammon mapping, MLP, back-propagation.*

Contents

List of Figures

1 Introduction

Human participation plays a vital role in most decisions making cases once we are analyzing the information. Although there are huge storage capacity and computational power of computers is accessible in modern time but they are unable to totally replace the flexibleness, perceptual abilities, creativity, and general knowledge of human being. We must have a correct interaction between human and computers to perform the tasks. The real world data we are using in sciences and technologies are often high dimensional which makes it very difficult to understand the data and extract patterns. One can achieve such an understanding by making a visual insight into the data set. Visual data mining is one of the field of data science where the human is integrated in the data analysis process. It mainly covers data visualization and graphical presentation of information. The idea behind the visualization is to provide data in some visual form so that we can understand them, gain insight into the data, draw some conclusions and able to take some decisions. Visualization also helps us to find clusters, outliers, or various regularities in the data. In many cases, the dataset used in the experiment is high-dimensional but it may happen that the data lies near a lower-dimensional manifold i.e. the high-dimensional data may be some indirect measurements of an underlying source, which may not be measured directly. To learn low-dimensional manifold from high-dimensional data is same as to learn the underlying source. This learning is called manifold learning and it is one of dimensionality reduction technique used in information processing fields including pattern recognition, data compression, machine learning, and database navigation. One of such manifold learning techniques is multidimensional scaling. The aim of multidimensional scaling (MDS) is to project high dimensional data points in a low dimension, often two or three-dimensional space such that the pairwise distance between points preserved. Although any distance measure can be used but we use Euclidean distance as it is the most popular one given its ease of interpretation and calculation and this distance is used in classical MDS.
Amongst the earliest approaches to MDS is so-called classical multidimensional scaling, also named as Torgerson-Gower scaling after Torgerson (1958) and Gower (1966). They had showed that given high dimensional points a low dimensional representation will be found through an eigendecomposition. Classical MDS is closely associated to principal component analysis. Kruskal provide a new way to calculate the parameters for the MDS model by minimizing a stress function. His approach has become the foremost widely used version of MDS. But the mathematical optimization problem underlying least-squares MDS isn't trivial. Afterthat, Jan de Leeuw had made many alternative theoretical contributions to the numerical algorithm to implement MDS. He had introduced the SMACOF algorithm to implement MDS. In this thesis we introduce a new way to implement MDS by training artificial neural networks.

2 Activation functions

Activation functions are used at each node in a neural network. After appropriately weighing the inputs, the result is passed through the activation function which determines the output. The activation functions are mainly needed to introduce non linear nature in the network so that the network can capture complex patterns. In addition, the use of the activation function limits output of the neuron to a certain range for some cases. Here I will introduce some commonly used activation functions used in neural networks.

2.0.1 Various types of activation functions

- **Threshold function:** The threshold function is a binary step function sometimes used to quantify the output of a neuron in the output layer. In this, we consider a threshold value and if the value of net input say x is greater than the threshold then the neuron is activated. An example

$$a(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{1}$$

- **Logistic/Sigmoid:** Logistic or Sigmoid function is defined as

$$a_k(x) = \frac{1}{1 + e^{-kx}} \tag{2}$$

where k is a parameter that determines the steepness of the function.

- **Tanh:** Hyperbolic tangent function is defined as

$$a(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

- **ReLU:** The Rectified Linear Unit or ReLU can be defined as

$$a(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{4}$$

- **Leaky ReLU:** Leaky ReLU can be defined as

$$a_\alpha(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases} \tag{5}$$

where α is a parameter.

- **Softmax:** Softmax function is defined as

$$a : \mathbf{R^k} \to [0,1]^k \text{ given by } a(\vec{x})_i = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{6}$$

where $\vec{x} = (x_1, x_2, \ldots, x_k) \in \mathbf{R^k}$.

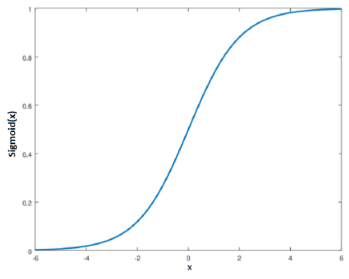
- **GELU:** GELU is defined as

$$GELU(x) = x\phi(x) = 0.5x(1 + erf(\frac{x}{\sqrt{2}}) \tag{7}$$

Here ϕ is the cumulative distribution of $N(0,1)$ and $erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$.

3 Artificial Neural Networks

Neural networks are extensively used in machine learning that try to mimic human nervous system. There are many sorts of artificial neural networks used as a computational tool like Kohonen Self Organizing Neural Network, Radial basis function Neural Network, Modular Neural Networks, Feedforward Neural Network, Convolutional Neural Network(CNN), Recurrent Neural Network(RNN), LSTM etc. In this section I will describe two variants of feedforward neural network architectures single-layer perceptron networks and multilayer perceptron networks. First I will discuss on single layer perceptron as it is the simplest of the architectures and this helps to understand multilayer perceptron networks better.


(a) Sigmoid

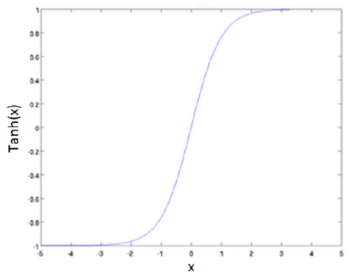
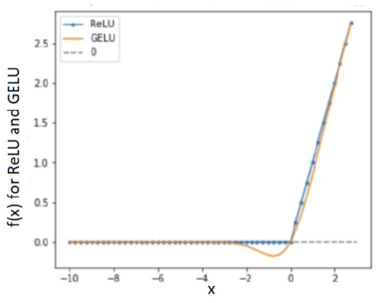
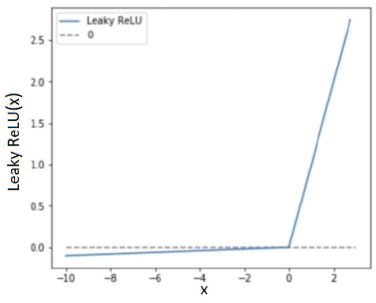

(b) Tanh

Figure 1: The left side represents the Sigmoid function and the right represents the Tanh function



(a) ReLU and GELU



(b) Leaky ReLU

Figure 2: The left side represents the ReLU and GELU functions and the right represents the Lekey ReLU function with $\alpha = 0.01$

3.1 Single layer perceptron network

Single layer perceptron is the first proposed neural network model. The model has two layers input layer(layer 0) and output layer(layer 1). Here input vector is multiplied with weight vector first, then passes through an activation function to get an output. The architecture of a single layer perceptron is given in Figure 3.

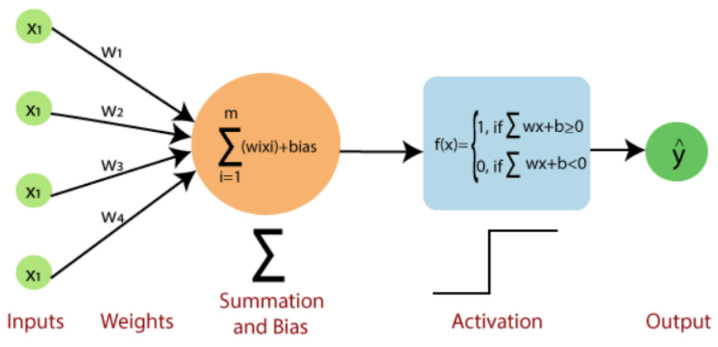


Figure 3: Single layer perceptron with threshold function as activation

3.2 Multilayer perceptron network

A multilayer perceptron network(MLP) is an extension of single layer perceptron having more layers. The hidden layers are positioned between the input layer and the output layer. The prediction and classification are performed by the output layer. The weights in the MLP are trained with the back propagation learning algorithm.

Now I will introduce mathematical representation for an MLP-network. Suppose we have a

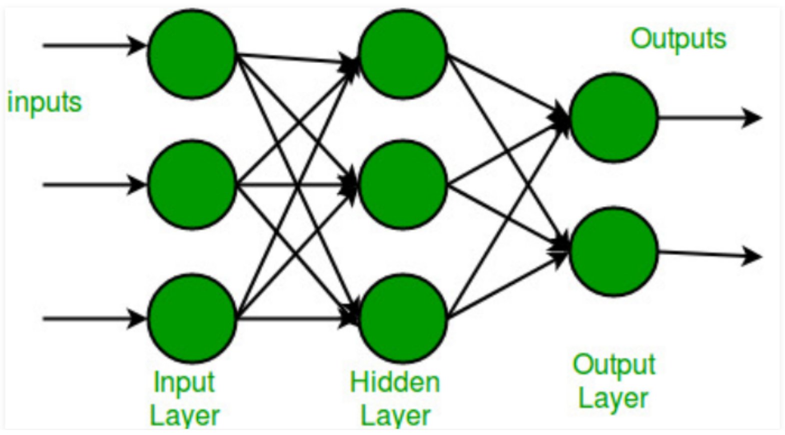


Figure 4: Multilayer perceptron layer with single hidden layer

multilayer neural network with L layers, marked by $l = 0, 1, \dots, L$, where $l = 0$ denotes inputs, $l = 1, \dots, (L-1)$ denotes hidden layers, and $l = L$ denotes the Lth layer (outputs). Each layer l has n_l neurons. The inputs to neurons in the lth layer correspond to the outputs of neurons in the layer $(l-1)$. Therefore, the output value $y_j^{(l)}$ of the jth neuron in the lth layer is computed as follows:

$$y_j^{(l)} = f(a_j^{(l)}) = f(\sum_{k=0}^{n_l} w_{jk}^{(l)} y_k^{(l-1)}), \quad j = 1, \dots, n_l, \quad l = 1, \dots, L. \tag{8}$$

where $f(.)$ is the activation function of neurons, $w_{jk}^{(l)}$ are the weights of connections between the kth neuron in the layer $(l-1)$ and the jth neuron in the lth layer, and$y_0^{(l)} = 1$ and $a_j^{(l)}$ represents the input to the jth neuron in the lth layer. Note that $y_k^{(0)} = x_k, k = 0, \dots, n_0$. The perceptron with one hidden layer of neurons is shown in Fig−4.

3.2.1 Training of an MLP

To train a MLP we update weights by optimizing a loss function. One of the most well-known training algorithms is named as backpropagation algorithm and we discuss it bellow.
Backpropagation: The network learning problem is to determine optimal weights

$$W = \{w_{jk}^{(l)}, j = 1, \dots, n_l, k = 0, \dots, n_l - 1, l = 1, \dots, L\}.$$

The back-propagation algorithm starts from initializing the weights to small random values, random choice of an input vector X_i from $\{X_1, X_2, \dots, X_m\}$, and propagation of the signal forward through the network. The output vector $Y_i = (y_{i1}, y_{i2}, \dots, y_{id})$ for the input vector X_i is computed and the error function becomes $E_i(W) = \frac{1}{2}\sum_{j=1}^{d}(y_{ij} - t_{ij})^2$ where $T_i = (t_{i1}, \dots, t_{id})$ is the target vector associated to X_i on the output layer L. If the value of the error function $E_i(W)$ is non zero, the weights W need to be updated. Compute the error $\delta_j^{(L)}$ of the jth neuron in the output layer

$$\delta_j^{(L)} = f'(a_j^{(L)})(y_{ij} - t_{ij})$$

where f' is the derivative of the activation function f . Compute the error $\delta_k^{(l)}$ of the kth neuron for the preceding layers by propagating the errors backward:

$$\delta_k^{(l)} = f'(a_k^{(l)}) = \sum_{s=1}^{n_{l+1}} w_{sk}^{(l+1)} \delta_s^{(l+1)}$$

For the hidden layer update the weights using

$$\Delta w_{jk}^{(l)} = -\eta \delta_j^{(l)} y_{jk}^{(l-1)}$$

where η is the learning rate.

4 Dimensionality reduction

The dimensionality reduction techniques are taking high dimensional data as input and map those high dimensional data to a lower dimensional space. Visualizing data that has more than three dimension is problematic therefore the data we want to visualize must have dimension less or equal to three. Here we will discuss some of the dimensionaity reduction techniques such as Principal Component analysis(PCA), Sammon mapping, Locally linear Embedding, and Multidimensional scaling.

4.1 Principal Component analysis(PCA)

Principal component analysis or PCA is the most widely used technique for dimensionality reduction. It is concerned with explaining the variance-covariance structures of a set of variables through a linear combinations of those variables which are called principal components. Given a set of n variables $X_1, X_2, \ldots, X_n$ the corresponding principal components $PC_1, PC_2, \ldots, PC_n$ are orthogonal linear combination of $X_1, X_2, \ldots, X_n$ i.e.

$$\begin{aligned} PC_1 &= a_{11}X_1 + a_{12}X_2 + \cdots + a_{1n}X_n \\ \mathrm{PC}_2 &= a_{21}X_1 + a_{22}X_2 + \cdots + a_{2n}X_n \\ &\cdots\cdots \\ \mathrm{PC}_n &= a_{n1}X_1 + a_{n2}X_2 + \cdots + a_{nn}X_n \end{aligned}$$

such that PC_1 has the highest variance, called first principal component, PC_2 has the second highest variance, called second principal component and so on. Although there are n principal components corresponding to n variables $X_1, X_2, \ldots, X_n$ but in practice much of the variablity can be captured by a small number k of principal components.

4.2 Sammon mapping

Sammon mapping or Sammon projection is an algorithm that maps a high-dimensional space to a space of lower dimensionality such that pairwise distances in high-dimensional space is as close as possible to pairwise distances lower-dimension projected space. It is a non linear reduction technique. Suppose the distance between ith and jth objects in the original space by d^*_{ij} and the distance between their projections by d_{ij}. Sammon's mapping aims to minimize the following error function, which is calleds Sammon's stress:

$$E = \frac{1}{\sum\limits_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}.$$

The minimization can be done using steepest descent or using some iterative methods. Many implementations prefer to use the first Principal Components as a starting configuration.

4.3 Locally Linear Embedding

Locally Linear Embedding (LLE) another kind of dimensionality reduction technique. The idea behind LLE is to find a set of weights that perform local linear interpolations that closely approximate the data. The steps in LLE are

- Define neighbors for each data point X_i

- Define weights that allow neighbors to interpolate original data accurately (represented as a matrix W)
- Given those weights, find new data points Y_i's that minimize interpolation error in lower dimensional space

Here the weight matrix W in second step is obtained by minimizes the cost function

$$\epsilon(W) = \sum_i \|X_i - \sum_j W_{ij} X_j\|^2$$

subject to the constraints that $W_{ij} = 0$ if X_j is not a neighbor of X_i and that the rows of the weight matrix sum to one: $\sum_j W_{ij} = 1$.

4.4 Multidimensional Scaling

Multidimensional scaling (MDS) is a non linear dimensionality reduction techniques for generating a spatial representation out of the proximities of objects. The main principle of multidimensional scaling is that distances of datapoints in the representation that the method creates should be as close the real distances as possible.
Multidimensional scaling can be divided into metric and non-metric multidimensional scaling. A particular case of metric multidimensional scaling is classical multidimensional scaling where the distance is Euclidean distance. Here I will briefly discuss about classical multidimensional scaling as our work is motivated from here and also introduce about non-metric multidimensional scaling.

4.4.1 Classical Multidimensional Scaling

Classical MDS can be considered the first algebraic approach to MDS. It has been independently proposed by several authors: Torgerson (1958), Gower (1966), and Kloek and Theil (1965). Given a distance matrix D^X, classical MDS attempts to find t data points $y_1, \ldots, y_k$ in d dimensions, such that if d^Y_{ij} denotes the Euclidean distance between y_i and y_j, then the matrix D^Y is similar to the matrix D^X. Here we minimize the following stress

$$\min_Y \sum_{i=1}^{k} \sum_{i=1}^{k} (d^X_{ij} - d^Y_{ij})^2 \tag{9}$$

where $d^X_{ij} = ||x_i - x_j||$ and $d^Y_{ij} = ||y_i - y_j||$.
The distance matrix D^X can be converted to a kernel matrix K in the following way

$$K = -\tfrac{1}{2} H D^X H$$

where $H = I - \frac{1}{k}\mathbf{1}\mathbf{1}^T$ where $\mathbf{1}$ is the column vector of all 1's.
Now we have the following theorem
Theorem: Let D be a distance matrix. If K is define as above then D is Euclidean if and only if K is positive semi-definite.
Now our distance matrices D^X and D^Y are Euclidean therefore the corresponding kernel matrices are positive semi-definite. We know that if K is positive definite then K can be written as $K = T^T T$. Then the stress in equation (10) can be reduce to

$$\min_Y \sum_{i=1}^{k} \sum_{i=1}^{k} (x_i^T x_i - y_i^T y_i)^2 \tag{10}$$

This expression can be converted to the following trace

$$\min_Y Tr(X^TX - Y^TY)^2$$

Now consider the singular value decomposition of X^TX and Y^TY which are given by $X^TX = V\Lambda V^T$ and $Y^TY = Q\hat{\Lambda}Q^T$. Then the cost function reduce to

$$\begin{aligned}
&\min_{Q,\hat{\Lambda}} Tr(V\Lambda V^T - Q\hat{\Lambda}Q^T)^2 \\
&= \min_{Q,\hat{\Lambda}} Tr(\Lambda - V^TQ\hat{\Lambda}Q^TV)^2 \\
&= \min_{G,\hat{\Lambda}} Tr(\Lambda - G\hat{\Lambda}G^T)^2 \\
&= \min_{G,\hat{\Lambda}} Tr(\Lambda^2 + G\hat{\Lambda}G^TG\hat{\Lambda}G^T - 2\Lambda G\hat{\Lambda}G^T)
\end{aligned}$$

where $G = V^TQ$.
Now for fixed $\hat{\Lambda}$ one can minimize for G and the result is $G = I$.
So, the cost function becomes $\min_{\hat{\Lambda}} Tr(\Lambda^2 + \hat{\Lambda}^2 - 2\Lambda\hat{\Lambda}) = \min_{\hat{\Lambda}} Tr(\Lambda - \hat{\Lambda})^2$
To make the two matrices Λ and $\hat{\Lambda}$ similar as possible one can choose top t diagonal elements of Λ as $\hat{\Lambda}$.
Now, from $G = V^TQ$ we have $Q = V$ as $G = I$. Then $Y = \hat{\Lambda}^{\frac{1}{2}}Q = \hat{\Lambda}^{\frac{1}{2}}V$.
Thus classical multidimensional scaling boils down to finding the eigen values and eigen vectors of the kernel matrix corresponding to D^X.

4.4.2 Non Metric Multidimensional Scaling

In contrast to metric MDS, non metric multidimensional scaling dealing with non-metric data which means that the dissimilarities cannot be interpreted as distances i.e. the dissimilarities given in some ordinal scale. The standard non-metric multidimensional scaling problem is as follows: Given a symmetric matrix with zero diagonals $\Delta = [\delta_{ij}]_{nxn}$ of dissimilarities find $Y = [y_i] \in \mathbf{R^{dxn}}$ where columns are points in projected space such that

$$\forall i,j,k,l, \;\; \delta_{ij} < \delta_{kl} \Leftrightarrow ||x_i - x_j||_2^2 < ||x_k - x_l||_2^2 \tag{11}$$

5 Our Methodology

I have already discussed the multilayer perceptron networks and the multidimensional scaling in details in previous sections. Here I will combine those two concepts to implement MDS. I have chosen the MLP model for the purposes of construction for the reason that it can be expressed in a very compact algebraic form and it has universal approximation properties. We know that the aim of MDS is find an embedding of a given set of points such that the pairwise distance between points are preserved. We have already seen that classical MDS boils down eigendecomposition problem which is computationally difficult for large dimensional matrix. Here we will approximate this linearity by introducing non linearity with help of neural networks. Now I will present the idea of our works.
Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n d-dimensional points we try to find points $Y = \{y_1, y_2, \ldots, y_n\}$ in k-dimensional space where $k << d$ such that

$$||x_i - x_j||_2 \approx ||y_i - y_j||_2 \;\forall i \neq j$$

which is same as to minimize the following objective function

$$\textstyle\sum_{i<j}(||x_i - x_j||_2^2 - ||y_i - y_j||_2^2)^2$$

In our work, the embedded points y_i's are the outputs of a multilayer perceptron network N corresponding to the point x_i's i.e. $y_i = N(x_i)$. To train the network, each pair (x_i, x_j), where $i < j$ of multidimensional points feed to the network and the error $E_{ij} = (||x_i - x_j||_2^2 - ||y_i - y_j||_2^2)^2$ is calculated. After feeding all the pairs the total error becomes

$$E_{total} = \sum_{i<j} E_{ij} = \sum_{i<j}(||x_i - x_j||_2^2 - ||y_i - y_j||_2^2)^2$$

We have considered a scaled version of the above loss function for our work which is given bellow

$$E = \frac{1}{\sum_{i<j}||x_i - x_j||_2^2} E_{total} = \frac{1}{\sum_{i<j}||x_i - x_j||_2^2} \sum_{i<j}(||x_i - x_j||_2^2 - ||y_i - y_j||_2^2)^2$$

we have used stochastic gradient decent with momentum to update the weights in the network. We also use hyperbolic tangent activation function in our network. Now, I will discuss the weight update rules to train the network.
Suppose the network has L layers and the l-th layer contains n_l number of nodes for $l = 1, 2, \ldots, L$. The inputs are the given data points x_i's. Let $w_{ji}^{(l)}$ is the weight between the jth neuron in the lth layer and the ith neuron in the layer $(l-1)$. The weight $w_{j0}^{(l)}$ is the bias term. Let $y_j^{(l)}$ be the output of the j-th neuron of the l-th layer. Then

$$y_j^{(l)} = f(a_j^{(l)}) = f(\sum_{i=0}^{n_l - 1} w_{ji}^{(l)} y_i^{(l-1)}) \tag{12}$$

where f is the activation function. The error E_{pq} corresponding to the points x_p and x_q is given by

$$E_{pq} = c(||x_p - x_q||_2^2 - ||y_p - y_q||_2^2)^2$$

where $c = \frac{1}{\sum_{i<j}||x_i - x_j||_2^2}$.
For the last layer i.e. $l = L$,

$$\frac{\partial E_{pq}}{\partial w_{kj}^{(L)}} = c.2(||x_p - x_q||_2^2 - ||y_p - y_q||_2^2).(-2)(y_{pk}^{(L)} - y_{qk}^{(L)}).(f'(a_{pk}^{(L)})y_{pj}^{(L-1)} - f'(a_{pk}^{(L)})y_{qj}^{(L-1)})$$
$$= \text{-4c}(||\text{x}_p - x_q||_2^2 - ||y_p - y_q||_2^2)(y_{pk}^{(L)} - y_{qk}^{(L)})(f'(a_{pk}^{(L)})y_{pj}^{(L-1)} - f'(a_{qk}^{(L)})y_{qj}^{(L-1)})$$

Now, $f(z) = tanh(z)$. So, $f'(z) = 1 - tanh^2(z) = 1 - f(z)^2$.
Then, $f'(a_{pk}^{(L)}) = 1 - y_{pk}^{(L)2}$, and $f'(a_{qk}^{(L)}) = 1 - y_{qk}^{(L)2}$.

Therefore,

$$\frac{\partial E_{pq}}{\partial w_{kj}^{(L)}} = -4c(||x_p - x_q||_2^2 - ||y_p - y_q||_2^2)(y_{pk}^{(L)} - y_{qk}^{(L)})((1 - y_{qk}^{(L)2})y_{pj}^{(L-1)} - (1 - y_{qk}^{(L)2})y_{qj}^{(L-1)})$$

Let,

$$\delta_k^{(L)}(p,q) = -4c(||x_p - x_q||_2^2 - ||y_p - y_q||_2^2)(y_{pk}^{(L)} - y_{qk}^{(L)})$$

$$\Delta_{kj}^{(L)}(p) = \delta_k^{(L)}(p,q)(1 - y_{pk}^{(L)2})$$

$$\Delta_{kj}^{(L)}(q) = \delta_k^{(L)}(p,q)(1 - y_{qk}^{(L)2})$$

$$\text{So, } \frac{\partial E_{pq}}{\partial w_{kj}^{(L)}} = \Delta_{kj}^{(L)}(p)y_{pj}^{(L-1)} - \Delta_{kj}^{(L)}(q)y_{qj}^{(L-1)}$$

Thus, $\Delta w_{kj}^{(L)} = -\eta\frac{\partial E_{pq}}{\partial w_{kj}^{(L)}} = -\eta(\Delta_{kj}^{(L)}(p)y_{pj}^{(L-1)} - \Delta_{kj}^{(L)}(q)y_{qj}^{(L-1)})$.
For the hidden layer $l = L-1, L-2, \ldots, 1$,

$$\Delta w_{ji}^{(l)} = -\eta\frac{\partial E_{pq}}{\partial w_{ji}^{(l)}} = -\eta(\Delta_{ji}^{(l)}(p)y_{pi}^{(l-1)} - \Delta_{ji}^{(l)}(q)y_{qi}^{(l-1)})$$

where $\Delta_{ji}^{(l)}(p) = \delta_j^{(l)}(p)(1 - y_{pj}^{(l)2})$ and $\Delta_{ji}^{((l)}(q) = \delta_j^{(l)}(q)(1 - y_{qj}^{(l)2})$.
Also,

$$\delta_j^{(l)}(p) = \sum_k \Delta_{kj}^{((l+1)}(p) w_{kj}^{((l+1)} \text{ and } \delta_j^{(l)}(q) = \sum_k \Delta_{kj}^{((l+1)}(q) w_{kj}^{((l+1)}.$$

Now, we are using stochastic gradient decent with momentum for updating the weights so the weight update rules are becomes

$$\Delta w_{ji}^{(l)}(t) = -\eta \frac{\partial E_{pq}}{\partial w_{ji}^{(l)}} + \alpha \Delta w_{ji}^{(l)}(t-1)$$

$$\Delta w_{ji}^{(l)}(t) = -\eta(\Delta_{ji}^{(l)}(p) y_{pi}^{(l-1)} - \Delta_{ji}^{(l)}(q) y_{qi}^{(l-1)}) + \alpha \Delta w_{ji}^{(l)}(t-1)$$

In our experiment η takes values from $\{0.01, 0.001\}$ and α takes values from $\{0.7, 0.8, 0.9\}$.

6 Experiments and results

The first step is training the Multilayer perceptron network. For the traning we must have sufficient amount of data points for input and the corresponding distances of the data points in the original space. But we can calculate the distances by using our desire metric from the data points in the original space. We can calculate the distances for the outputs in the same way. After we trained the network, the MLP can be given data points from the original space as an input and the perceptron transforms these points into the projected space space. For the case of successful training the transformation should preserve the distances as well as possible. In each layer of the network the weights are randomly initialized that may results in diffrent stress values. For lower stress value we get better performance of the network.

In this section we will discuss about three experiments and the corresponding results. The first one is a simple linear transformation from three dimensions into two dimensions. The second experiment is a classification problem on the basis of the well-known Iris flower data set introduced by R.A. Fisher. The last experiment is also a classification problem but using the wine data set having larger number of attributes. We also give some comparative results on various datasets at the end of this section.

6.1 Results on diagonal data

In this experiment we simulated data from the diagonal of a unit cube and project the data into two dimension. We train the MLP by taking 25 simulated samples. The training set is defined as

$$X_{train} = \begin{bmatrix} 0.04 & 0.04 & 0.04 \\ 0.08 & 0.08 & 0.08 \\ \dots & \dots & \dots \\ 1.00 & 1.00 & 1.00 \end{bmatrix}$$

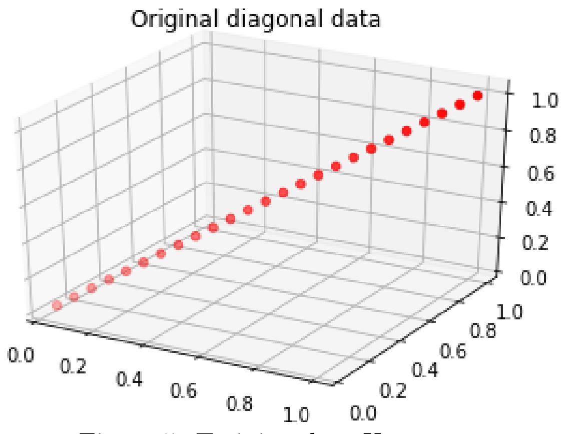


Figure 5: Training data X$_{train}$

$$D = \begin{bmatrix} 0 & 0.0693 & 0.1386 & \cdots & 1.5935 & 1.6628 \\ 0.0693 & 0 & 0.0693 & \cdots & 1.5242 & 1.5935 \\ 0.1386 & 0.0693 & 0 & \cdots & 1.4549 & 1.5242 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1.5935 & 1.5242 & 1.4549 & \cdots & 0 & 0.0693 \\ 1.6628 & 1.5935 & 1.5242 & \cdots & 0.0693 & 0 \end{bmatrix}$$

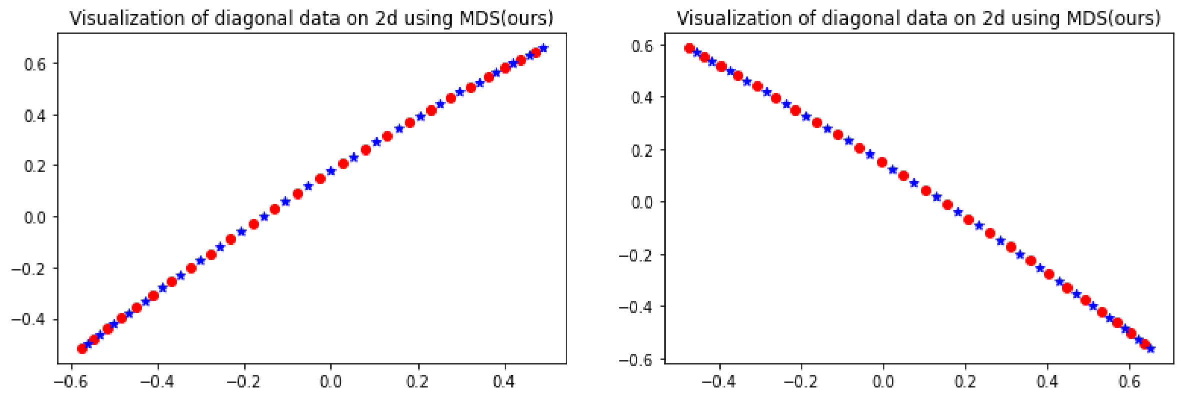


Figure 6: Visualization of diagonal data for number of nodes in hidden layer 1(left), 2(right) respectively

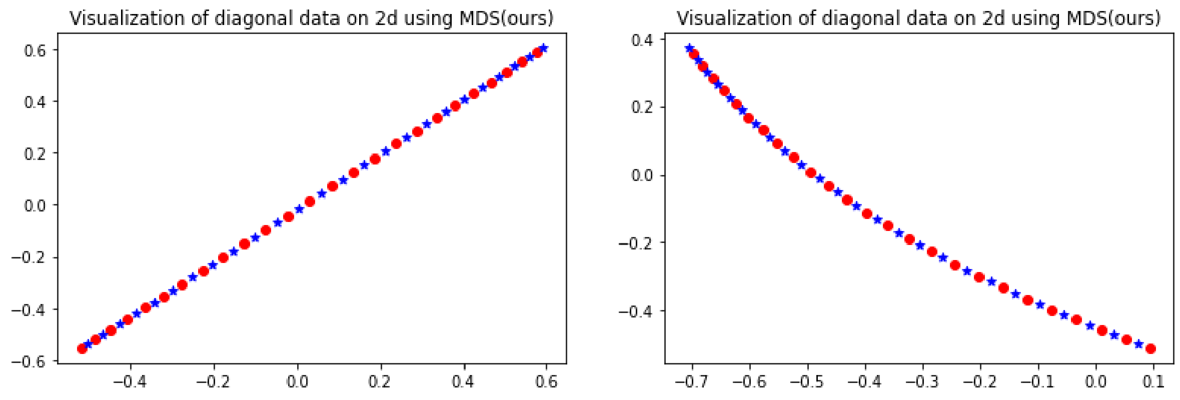


Figure 7: Visualization diagonal data for number of nodes in hidden layer 3(left), 4(right) respectively

The corresponding 25x25 distance matrix is calculated from X_{train} and represented by the matrix D. For error calculation we only need to consider upper triangular part of the distance matrix.
After training the weights the original inputs X_{train} were entered to the trained network. We also generated a test set X_{test} to check our model performance. X_{test} nothing but $X_{train} - 0.02$.

$$X_{test} = \begin{bmatrix} 0.02 & 0.02 & 0.02 \\ 0.06 & 0.06 & 0.06 \\ \dots & \dots & \dots \\ 0.98 & 0.98 & 0.98 \end{bmatrix}$$

We also feed the test set to the trained network. We mark the training point by red dots and test points by blue stars.

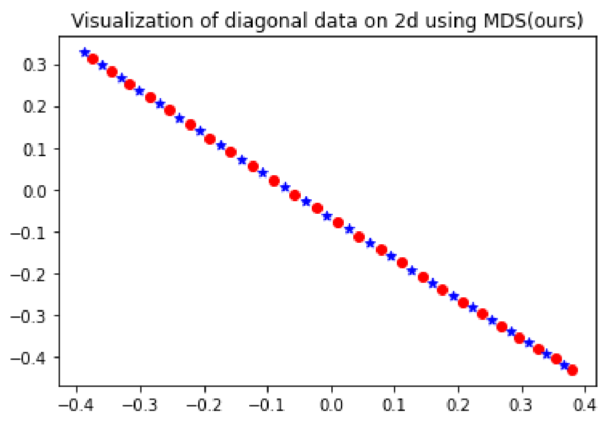


Figure 8: Visualization diagonal data for number of nodes in hidden layer is 5

The visualized results show the rotation and location of points varies notably. Artificial neural network generates the initial weights randomly so if we train the network twice with the same training data we will get different weights both times. Therefore, when looking the visualization of the results one must not be confused by the values of location or by the different rotation. Instead one should concentrate on on the fact that pairwise distances between points are preserved.
From the visualization it is clear that the projected points form a straight line most of the cases and the test points are located in between the output of the training data which is desirable. Even when the size of the hidden layer is only one, the results seem correct. As the results seem promising so it is reasonable to carry on into the more meaningful experiments.

6.2 Results on iris dataset

The Iris data set (Fisher 1936) is a classification dataset consisting of 150 samples and three classes: Iris Setosa, Iris Versicolour, and Iris Virginica with 50 samples from each class. Each sample has four numerical attributes, which are sepal length, sepal width, petal length and petal width. For our experiment we consider the standardized Iris data and have done $80-20\%$ split of the dataset into training set and test set. The dimensionality reduction was performed separately to both 2D and 3D.

Here are the visualized results of the experiment conducted on the Iris data set for single hidden layer with different size of nodes. 'o'-marks indicate the network output with the data used for training and '*'-marks indicate the network output for the testing set. The color indicates the class of the flower (purple = Iris Setosa, blue = Iris Versicolour, yellow = Iris Virginica).

From the visualization result in 2D, we found that Iris Setosa can be seen clearly apart from the other two. Versicolour and Virginica are overlapping a little bit but they can be easily distinguished from each other. The results of 3D case are very similar to ones in 2D. For only one node in hidden layer and we are projecting data into two dimension, the network output forms a completely straight line. Similar effect can be observed in the case of projecting the data into 3D.

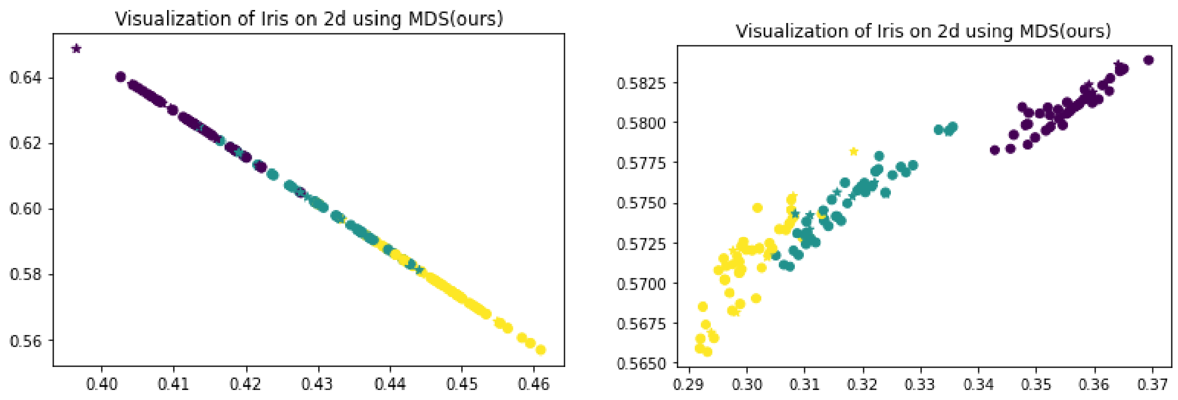


Figure 9: Visualization of Iris data for number of nodes in hidden layer 1(left), 2(right) respectively

6.3 Results on wine dataset

The wine data set has thirteen numerical variables: alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines, and proline. The data set contains 178 instances from three different classes. The classification is made based on the type of wine. All wines were produced in the same region but by using different varieties. or our experiment we consider the

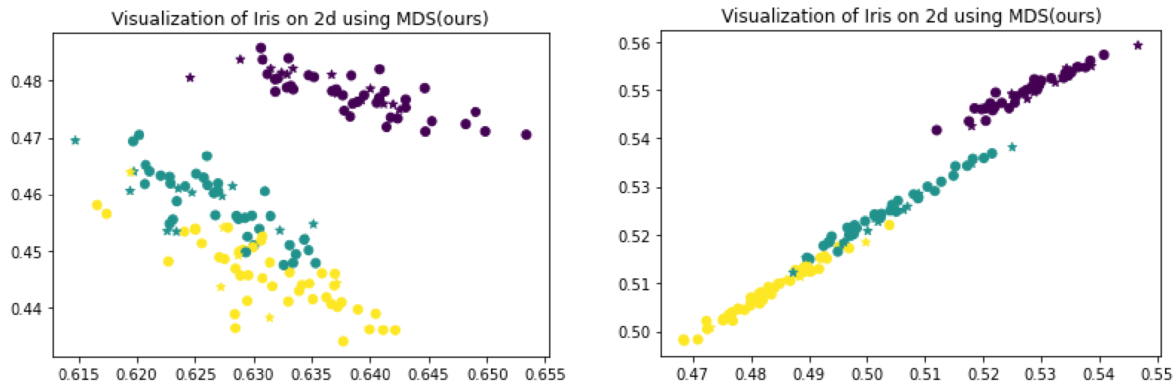


Figure 10: Visualization of Iris data for number of nodes in hidden layer 3(left), 4(right) respectively

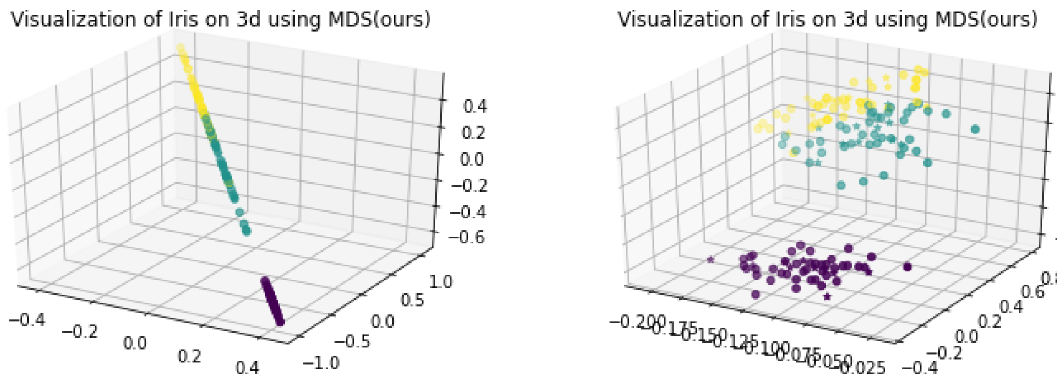


Figure 11: Visualization of Iris data for number of nodes in hidden layer 1(left), 2(right) respectively

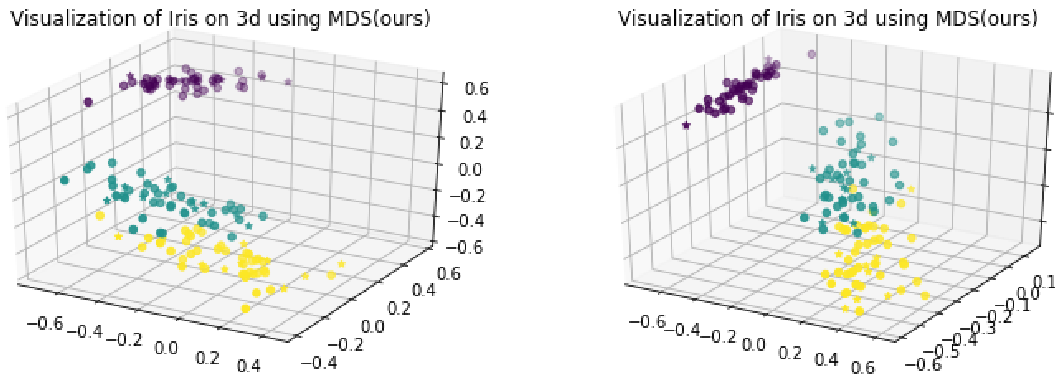


Figure 12: Visualization of Iris data for number of nodes in hidden layer 3(left), 4(right) respectively

standardized Iris data and have done 80% – 20% split of the dataset into training set and test set. The dimensionality reduction was performed separately to both 2D and 3D.

The wine data set is usually used by choosing only few variables at a time for visualization so that the differences between the classes might be clear to see. However, in this case where we try to visualize thirteen variables in 2D it is only anticipated that results do not distinguish the classes clearly. Fortunately the results show some kind of concentrations instead of total chaos. Here are the visualized results of the experiment conducted on the Iris data set for single hidden layer with different size of nodes. 'o'-marks indicate the network output with the data used for training and '*'-marks indicate the network output for the testing set. The color indicates the class of the flower (yellow = class0, blue = class1, yellow = class2).

From the visualization result in 2D, we found that class0 can be seen clearly apart from the other two but class2 and class3 are overlapping a little bit. Similar effect can be observed in the case of projecting the data into 3D.

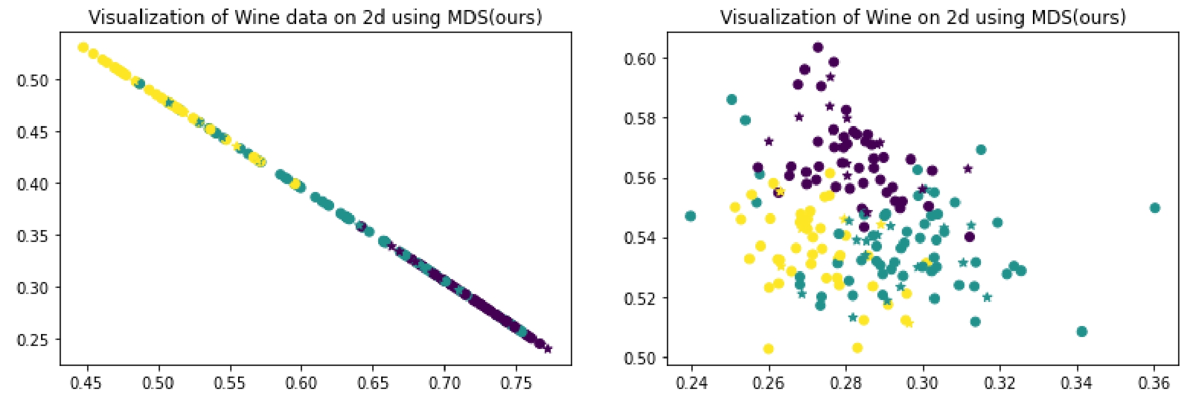


Figure 13: Visualization of wine data for number of nodes in hidden layer 1(left), 2(right) respectively.

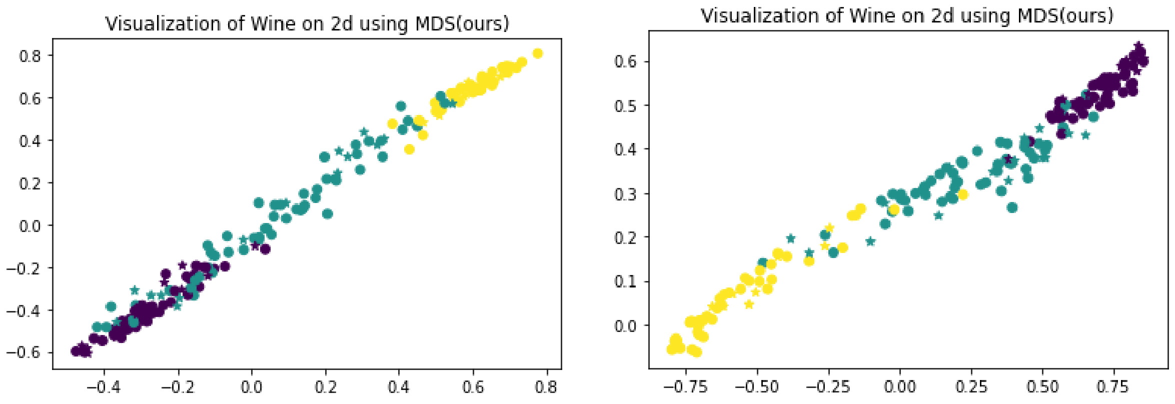


Figure 14: Visualization of wine data for number of nodes in hidden layer 3(left), 4(right) respectively.

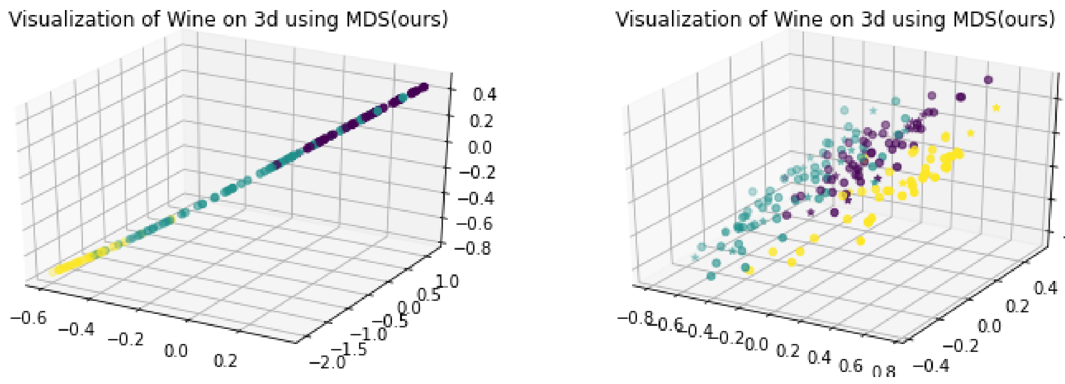


Figure 15: Visualization wine data for number of nodes in hidden layer 1(left), 2(right) respectively.

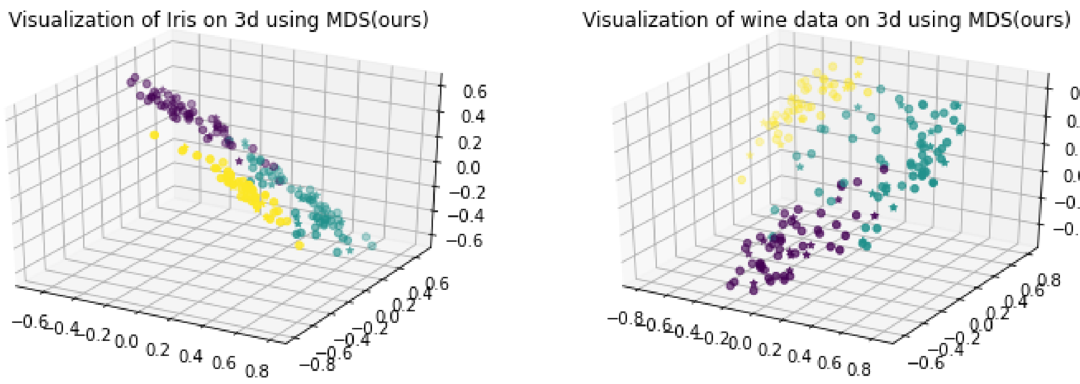


Figure 16: Visualization of wine data for number of nodes in hidden layer 3(left), 4(right) respectively.

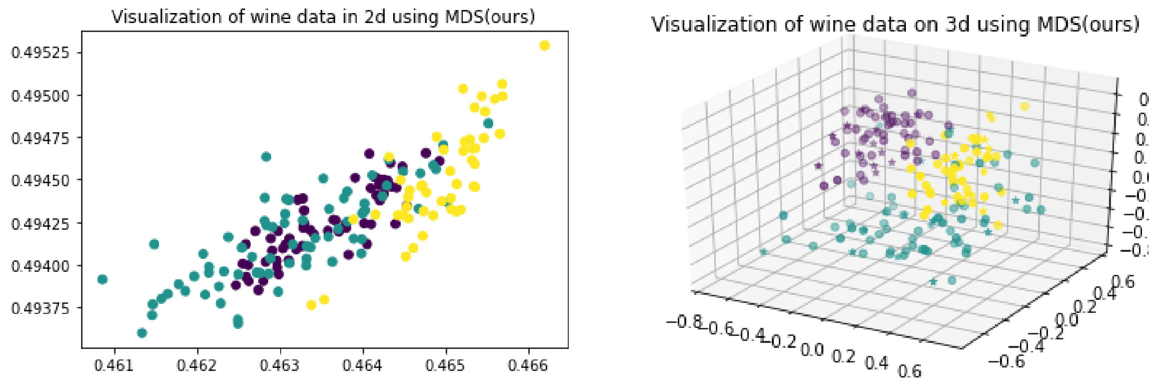


Figure 17: Visualization of wine data in 2d and 3d with 5 nodes in hidden layer

6.4 Some comparative results

As we are interested in dimensionality reduction then one best way to check model performance is by plotting our original data into lower dimension. We have visualized the data into two dimension by using various visualization techniques such as Sammon mapping, classical MDS, PCA, LLE and MDS(ours) methods and the comparison results are given bellow.

6.4.1 Comparative result on Iris dataset

Here, we have visualized Iris dataset by using Sammon mapping, classical MDS, PCA, LLE together with ours MDS methods. In each method, Iris Setosa(purple) forms a purely different cluster whereas there is a slightly overlap between the other two clusters of points of classes Iris Versicolour(blue) and Iris Virginica(yellow). From the figure it is clear that our method provide the best visualization compared to the other methods.

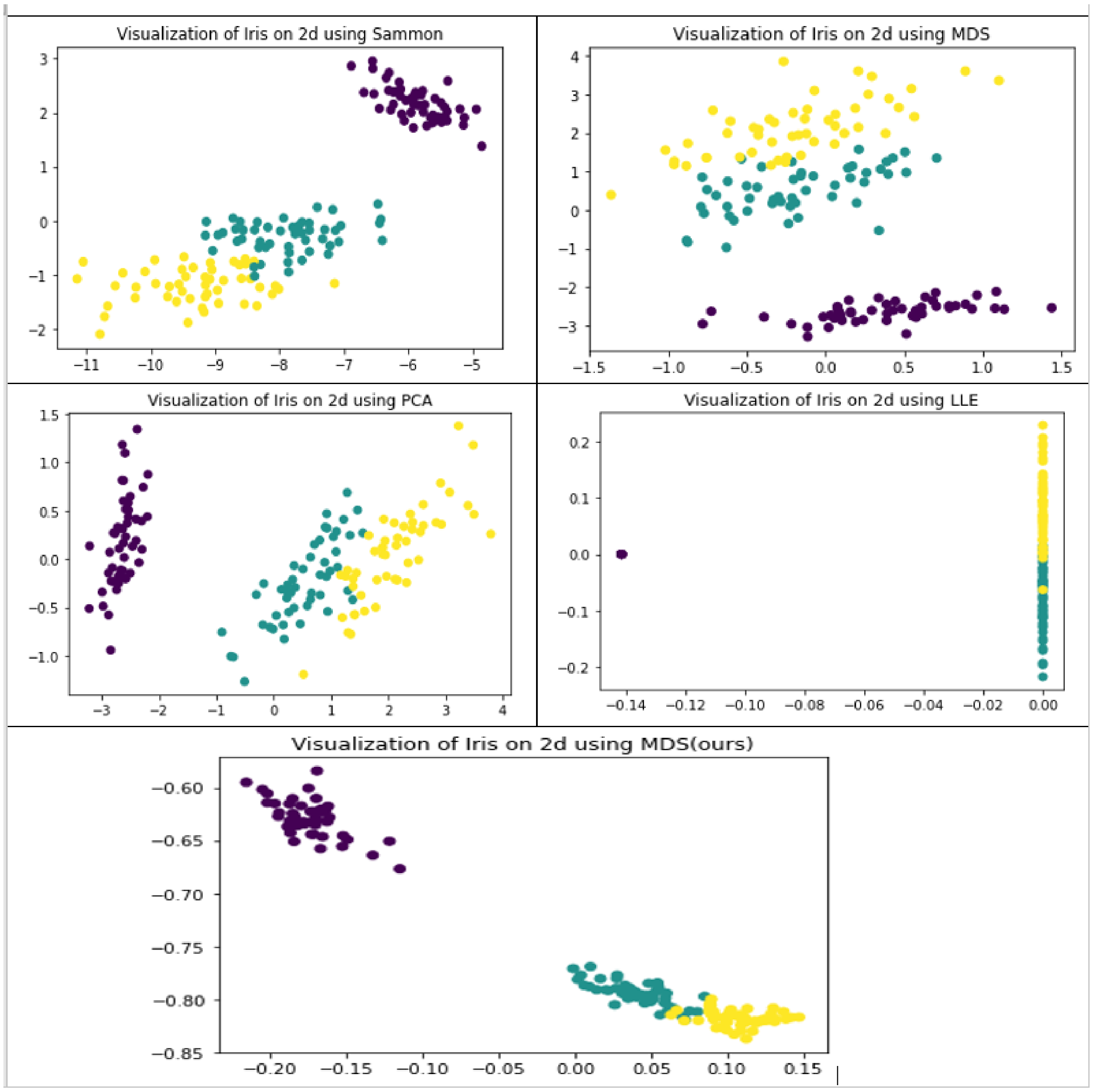


Figure 18: Visualization of Iris data using Sammon, MDS, PCA, LLE and MDS(ours)

6.4.2 Comparative result on Wine dataset

The visualization of the wine dataset by using Sammon mapping, classical MDS, PCA, LLE and MDS(ours) is given in the bellow figure. From the figure it is clear that LLE performs very poorly on wine data. PCA performs better than LLE but not giving better result like classic MDS and Sammon mapping. There is a slight overlap between the classes in each visualization result but our method outperform over others.

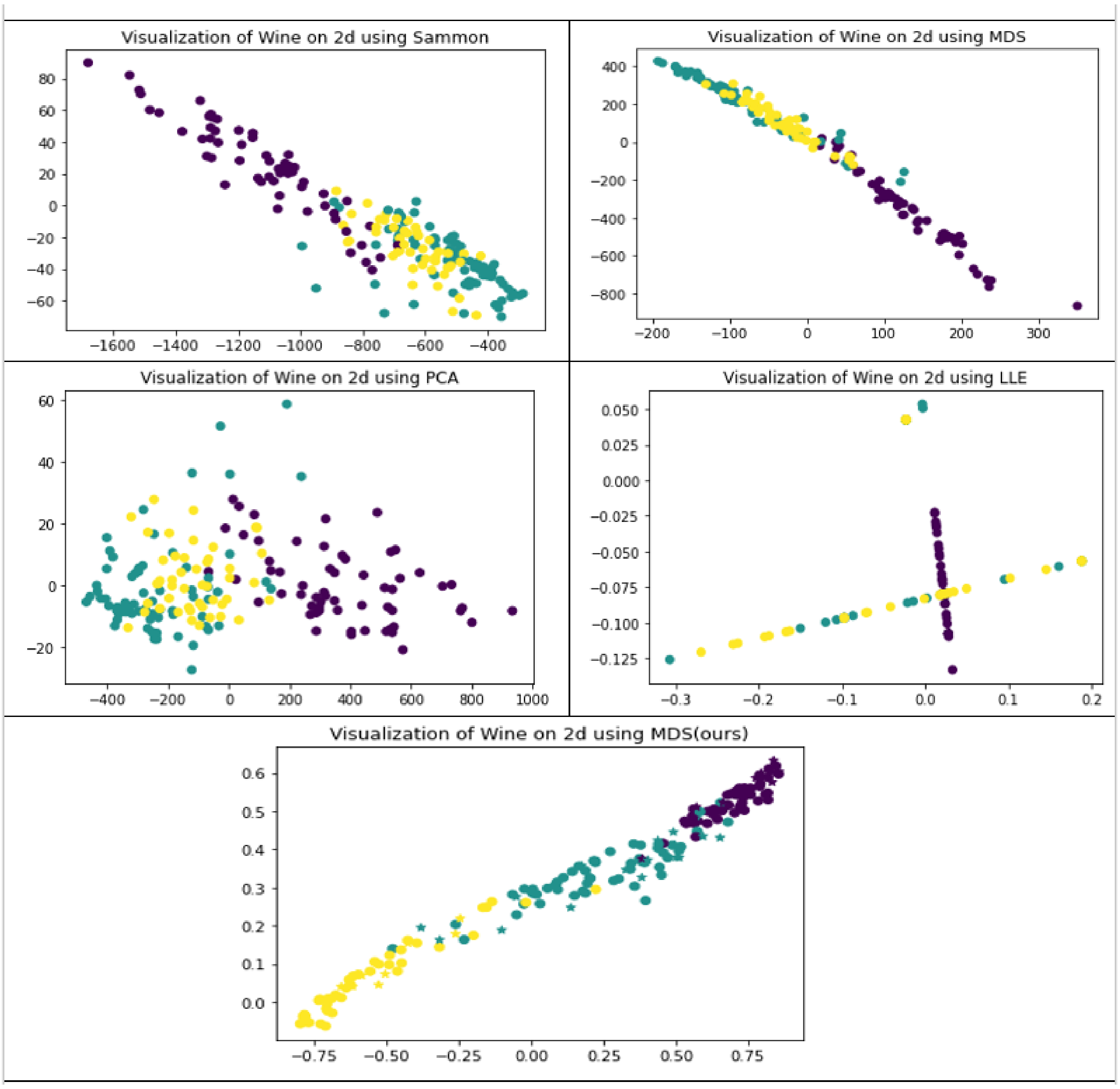


Figure 19: Visualization of Wine data using Sammon, MDS, PCA, LLE and MDS(ours)

6.4.3 Comparative result on digits dataset

This dataset is made up of 1797 8x8 images. Each image is a hand-written digit. In order to utilize an 8x8 figure like this, we transform each image into a feature vector with length 64. Although the dataset contains ten classes but in our experiment we only consider six classes, from digits 0 to 5, to get a clear representation of the classes. A small section of the data and the visualization results by different techniques are given bellow

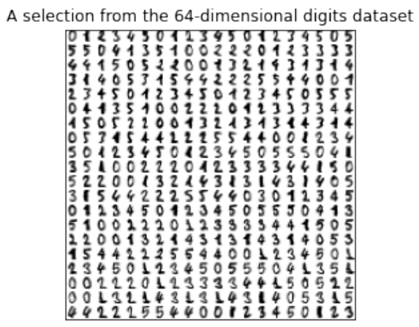


Figure 20: Some points in digits dataset

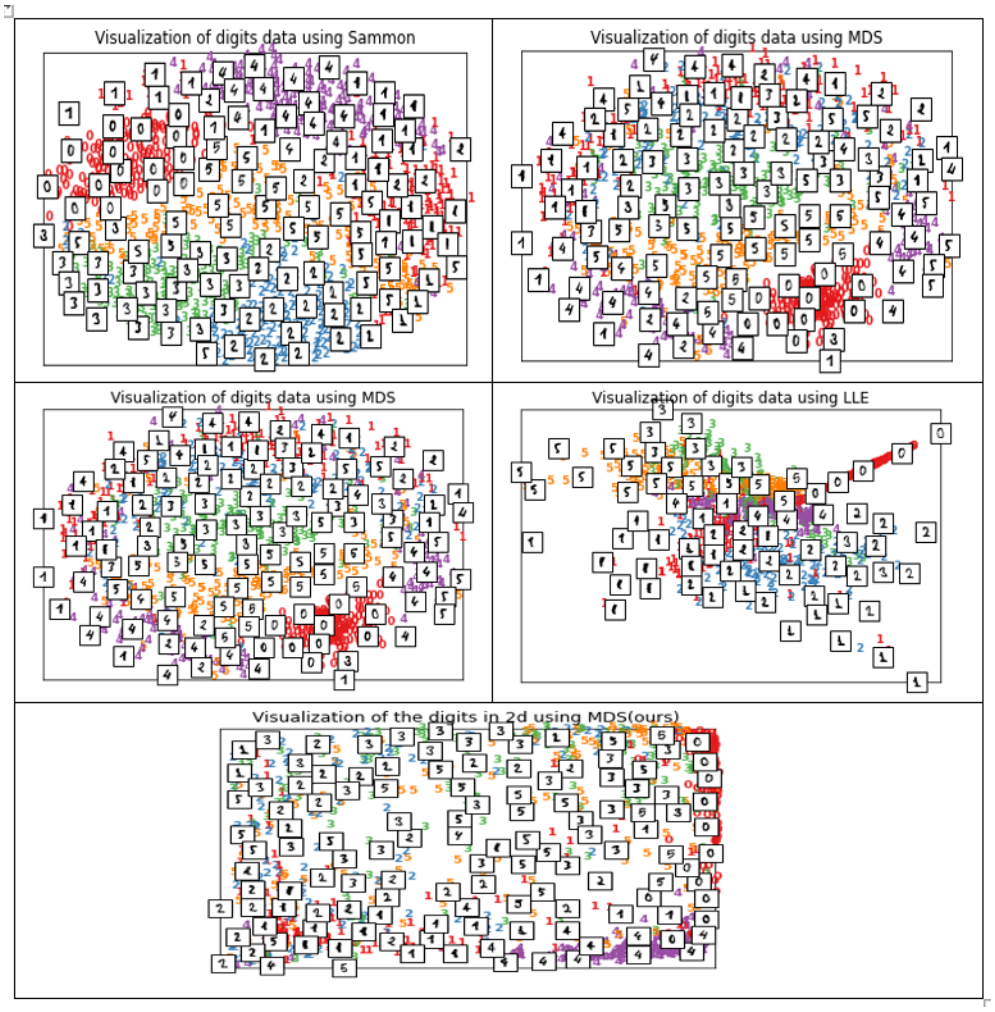


Figure 21: Visualization of digits data using Sammon, MDS, PCA, LLE and MDS(ours)

6.4.4 Comparative result on Sphere data

This dataset is a synthesized dataset of 1000 sample points generated from the surface of a unit sphere. This is not a labelled dataset unlike other datasets of our experiment. A pictorial view of the dataset is given bellow.

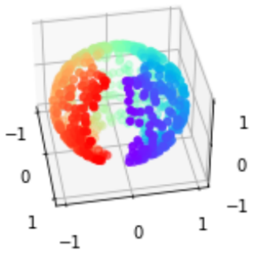


Figure 22: Sphere data

Like other datasets, we visualize this dataset into two dimension by various techniques same as before. From the visualization we can conclude that our method gives similar result like sammon mapping and classical MDS but perform much better than PCA and LLE.

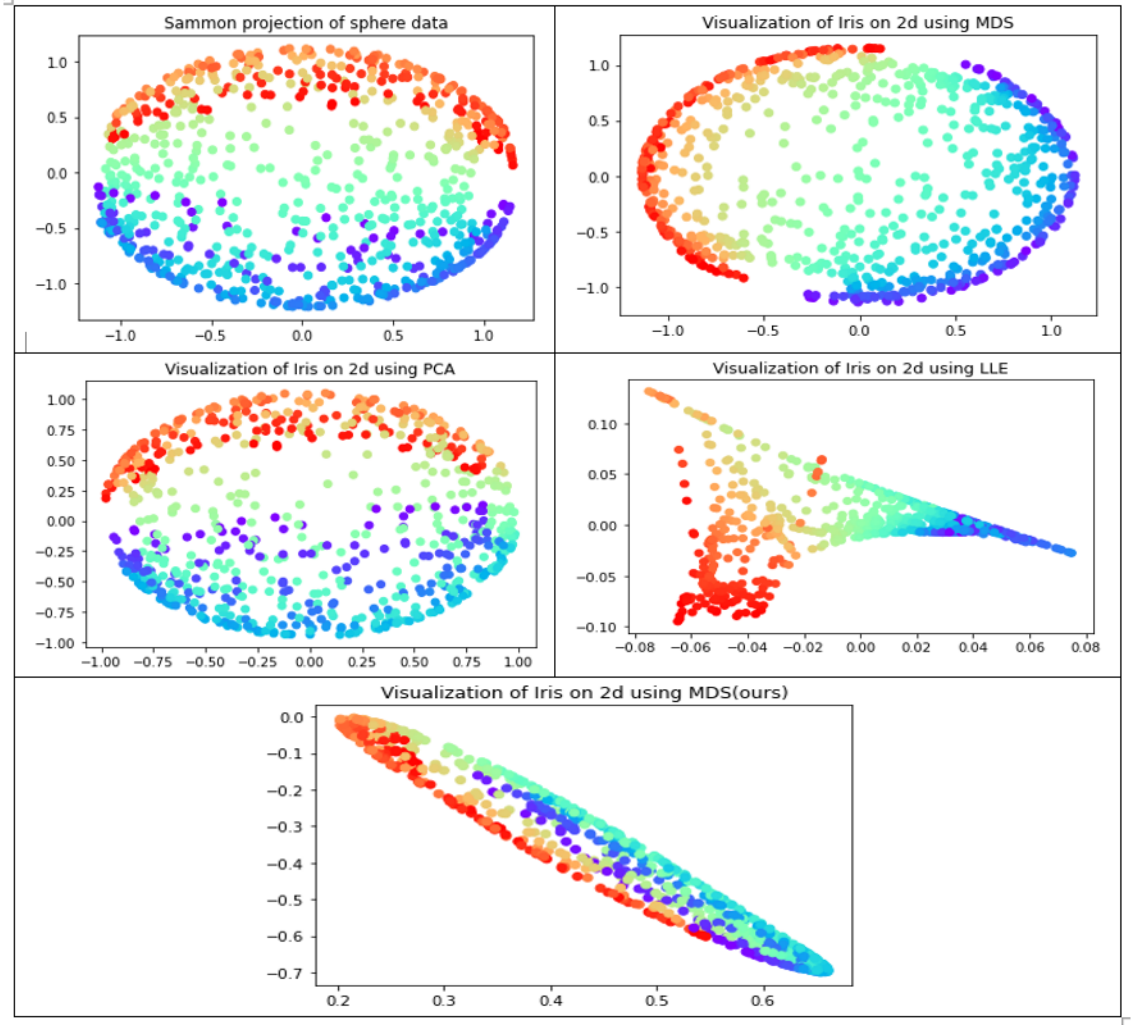


Figure 23: Visualization of sphere data using Sammon, MDS, PCA, LLE and MDS(ours)

7 Conclusion

The experiments we have done above give some proof that the our method does indeed achieve its goals. From the comparison results we can conclude that our method perform well in many casses. Nevertheless, the algorithm needs to be further examined and verified. The four experiments in the thesis do not give accurate results on the trustworthiness of the algorithm. As we are using multilayer perceptron network in our work then an important part is choosing the appropriate number of neurons in hidden layers which is not automatically determined by our method. One of the disadvantage of MLP is the number of parameters can grow very high if we increase the number of nodes in hidden layers or the number of hidden layers that makes computations difficult and time consuming. Another drawback of MLP is that a trained network cannot be trained with new data without starting the whole process from the beginning. Also layer weights generate randomly that take longer time for convergence. For future work, it will be interesting to see if we can use some pretrained networks to get faster convergence. Also, it will be interesting to see how our model performs on large dimensional datasets like image datasets.

References

[1] Michael A. A. Cox and Trevor F. Cox. *Multidimensional Scaling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[2] Ali Ghodsi. Dimensionality reduction a short tutorial. 01 2006.

[3] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Locally linear embedding and its variants: Tutorial and survey, 2020.

[4] J.B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–29, 1964.

[5] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[6] J.W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, 1969.