Comparing Methods of Nonlinear Dimensionality Reduction for Artificial Intelligence

Zachry Engel, PhD and Mitchell Wadas
Lone Star Analysis

1. Introduction

Dimensionality reduction is the process of transforming data from a high-dimensional space to a low-dimensional space such that the low-dimensional data is representative of the original data. Dimensionality reduction is prevalent in fields which deal in large amounts of data, and it has broad applications ranging from data visualization to image compression. Today we will focus on the application of nonlinear dimensionality reduction as a preprocessing step for artificial intelligence. To understand the 'curse of dimensionality' and the importance of dimensionality reduction to artificial intelligence, consider a dataset in a Euclidean space. Each additional dimension exponentially increases the spatial volume of the dataset. This exponential increase in spatial volume caused by each additional features makes observations more sparsely distributed in the feature space and this increased sparsity makes it more difficult for artificial intelligence algorithms to draw optimal decision boundaries in the feature space. This increased sparsity can be corrected by introducing many more samples, but in real applications, additional data cannot always be readily generated, and rarely in the volume required. Another element of the so called 'curse of dimensionality' is the introduction of noise; high-dimensional data will often introduce collinearity, extraneous features, and other sources of noise which (especially when combined with sparsity) can be misleading to artificial intelligence algorithms. Finally, each feature in an input vector requires its own parameters in an AI model. Each parameter introduces an exponential increase to the complexity of optimization of weights, decreases transparency of a model, exponentially increases the time to train a model and introduces the possibility of overfitting. By removing sources of noise and reducing the feature set into a more digestible dimension, proper dimensionality reduction allows for elegant, generalizable, and transparent artificial intelligence. These are the characteristics we will look for when comparing methods of dimensionality reduction.

2. NeRV

Neighbor Retrieval Visualizer (NeRV) (Venna et al. 2010) frames dimensionality reduction as an information retrieval problem; a good projection allows the neighbors of a data point to be retrieved in a lower dimension. NeRV focuses on the application of information retrieval for *similarity visualization*, the visualization of high dimensional neighbors in two dimensions. NeRV provides a rigorous measure of goodness for this task and introduces a method to optimize the measure. We evaluate the efficacy of NeRV, and the measure of goodness it provides, for preprocessing of the MNIST dataset.

The goal of similarity visualization is to produce a low-dimensional output such that, for any point *i* in the input data, the nearest neighbors of *i* in the output are maximally similar to the nearest neighbors of *i* in the input. In this way, similarity visualization can be evaluated using the traditional information retrieval metrics of precision and recall. NeRV defines the quality of a visualization as a weighted combination of precision and recall where the parameter λ controls the tradeoff between the two.

$$0 \leq \lambda \leq 1$$

$$Quality = \lambda Precision + (1 - \lambda) Recall$$

To define quality in terms of precision and recall in an unsupervised setting, NeRV extends the metrics of precision and recall to probabilistic neighborhoods. NeRV defines the probability of neighbor identity in both the input p_{ij} and the output q_{ij} as a function of pairwise distance $d(i, j)$. In Venna et al. (2010) the authors of NeRV show that, for any point i, the Kullback-Leibler divergence D between the distribution of neighbor probabilities around i in the input p_i and output can be interpreted as a generalization of precision or recall depending on the direction.

$$D(p_i, q_i) \approx 1 - recall$$

$$D(q_i, p_i) \approx 1 - precision$$

The NeRV cost function then becomes the weighted combination of average KL divergence in both directions and by minimizing this cost function, NeRV maximizes a weighted combination of precision and recall.

$$Cost = \lambda E_i[D(p_i, q_i)] + (1 - \lambda) E_i[D(q_i, p_i)]$$

Venna et al. (2010) suggest that the projection be initialized by principal component analysis, and that the conjugate gradient method be used to minimize the cost function, however any combination of initialization and optimization can be used.

In practice, NeRV suffers from several limitations. For unsupervised applications, the definition of neighbor probabilities requires a measure of distance in the input space that reliably reflects the similarity, or lack thereof, between points; this requirement creates practical difficulties on the MNIST dataset where distance metrics are not readily available. Consider the example of a hand-drawn zero, this image can be represented as a vector of pixels. If the image was shifted a pixel to the right or to the left, the resulting pixel vector would be quite far, by most metrics, from the original image. In this example, using Euclidean distance to define the input neighbors will result in optimal projection of the data in the information retrieval sense, but the projection will not be particularly useful in creating separable clusters for a classification task. This limitation may be overcome by a more suitable distance metric, one that is application specific, but in many cases, determining a suitable distance metric requires considerable thought and research. This is a limitation that makes the NeRV algorithm less generalizable.

NeRV suffers from similar limitations in the supervised application as well. Venna et al. (2010) describe a supervised application of NeRV where input distances are specified with class characteristics in mind. A simple example of such a distance metric is given below.

$$d(q, p) = \begin{cases} 0 \; if \; class_q = class_p \\ 1 \; if \; class_q \neq class_p \end{cases}$$

While this measure of distance may create separable clusters in the projection, it does not allow us to retrieve information about how data points are different from one another (only that they are) or how members of the same class relate to each other. These characteristics make this measure unsuitable for many supervised learning tasks. Another limitation we encountered is that the NeRV algorithm cannot be fit to one dataset and applied to another, consequently, the supervised application of NeRV requires prior knowledge of the new dataset and could not be used in preprocessing for machine learning.

Finally, the NeRV implementation provided by the Aalto University School of Science suffers poor performance on the MNIST dataset, both computationally and in the resulting projection. Execution times

for NeRV reach upwards of 1 hour when projecting a sample of 5000 hand-drawn digits. For our application of NeRV, as a preprocessing step to an artificial intelligence (AI) pipeline, this performance is limiting; an AI pipeline may receive many more than 5000 observations and often, the value provided is in real-time decision making. The time required to execute NeRV as a preprocessing step may hinder the ability of an AI pipeline to assist end users in making timely and data driven decisions.

The resulting projection from NeRV does not satisfy the goals of preprocessing either. NeRV does not create separable clusters and leaves the low-dimensional feature space rather noisy. As shown below, even a projection which considers an error in precision 9 times as expensive as an error in recall (lambda = .1) does not create distinct or separable clusters. The significant overlap between clusters in both visualizations makes classification of the two-dimensional output quite challenging. Compare the three projections in Figures 1-3 below as K effective Neighbors approaches 500 (the true number of observations from each class.)

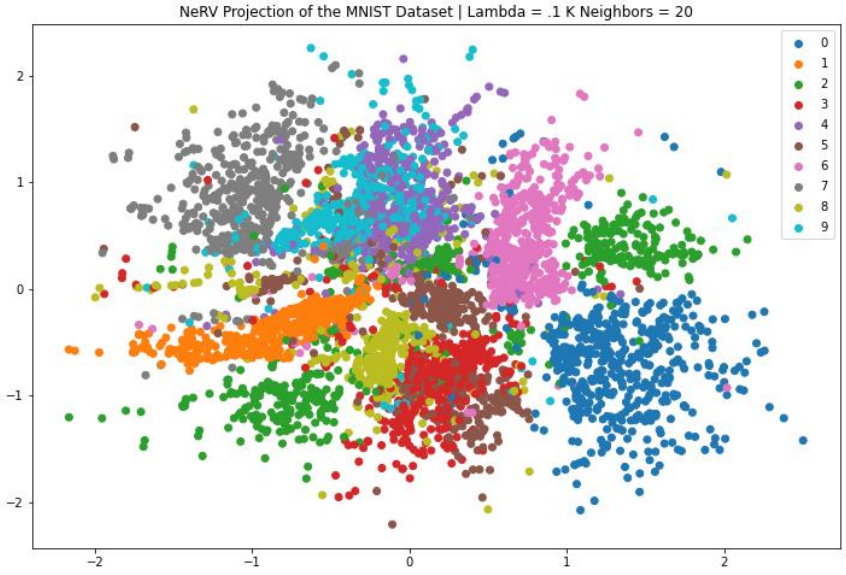

Figure 1: NeRV Projection of the MNIST Dataset | Lambda = .1, K Neighbors=20

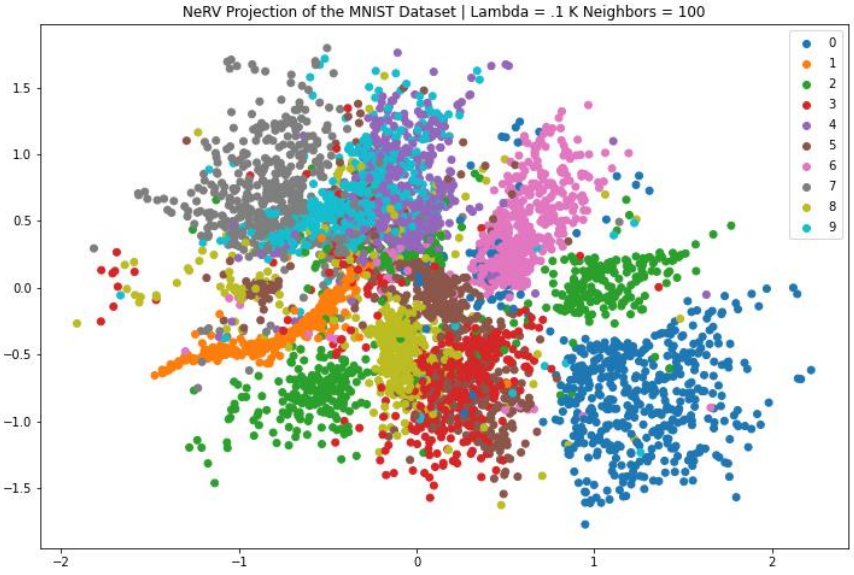

Figure 2: NeRV Projection of the MNIST Dataset | Lambda = .1, K Neighbors=100

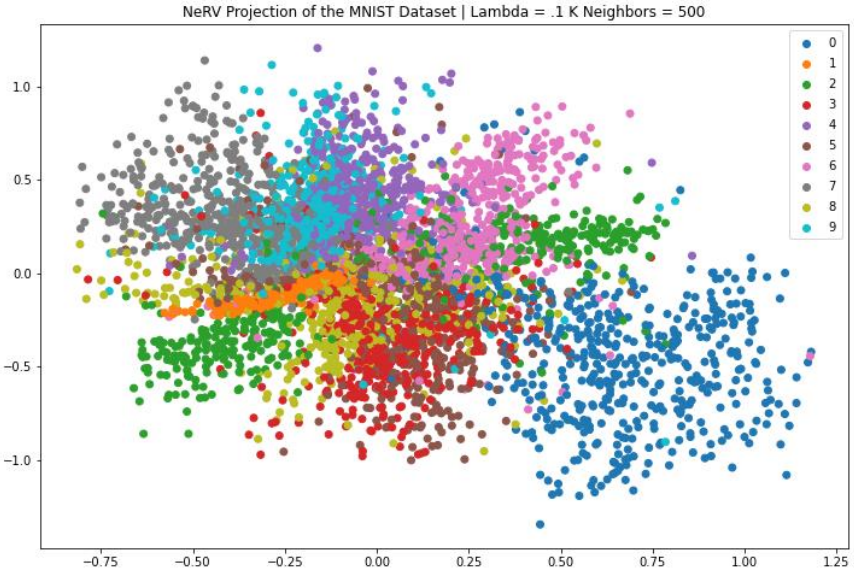

Figure 3: NeRV Projection of the MNIST Dataset | Lambda = .1, K Neighbors=500

3. UMAP

UMAP stands for Uniform Manifold Approximation and Projection (McInnes et al. 2020) and is a dimensionality reduction technique developed by Leland McInnes, John Healy, and James Melville. UMAP utilizes a framework developed around the fields of geometry and algebraic topology. More specifically, UMAP relies on a branch of topology known as Topological Data Analysis (TDA), which relies on the use of simplicial complexes to create topological spaces which allows the user to create a topological manifold that represents the data. Originally, it was proposed as an alternative to the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm, which was the state of the art at the time of the development of UMAP. However, since its development, UMAP has been considered by many as superior to t-SNE. A key difference between t-SNE and UMAP is that UMAP utilizes pure mathematical principles, as opposed to t-SNE which is a pure Machine Learning semi-empirical algorithm. The essence of the UMAP algorithm is the utilization of high-dimensional graph representation and then it reduces the data to a lower-dimensional graph using manifolds and projections.

UMAP begins by approximating the manifold that the data lies on. However, this initial approximation assumes the existence of the manifold. This is one key difference between UMAP and NeRV, because NeRV does not rely on the existence of a manifold for the data. This manifold is created using an algebraic concept known as a "fuzzy simplicial complex". Once the manifold has been created, UMAP measures the distance between points in the manifold, but it is not necessarily through Euclidean distance metric, UMAP allows us to utilize any distance metric we desire. It then uses the exponential probability distribution as:

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

Which represents the weighted distance from each point i to its nearest neighbor. Instead of perplexity, UMAP utilizes the number of nearest neighbors k defined as:

$$k = 2^{\sum_i p_{ij}}$$

The nearest neighbors can be approximated by the UMAP algorithm, or it can be defined by the user before the computation has begun. This obviously will affect the lower dimension graph created. Furthermore, the base form of UMAP uses a family of functions for modeling the distance probabilities in low dimension without normalizing the data. To calculate this probability, we need to define a minimum distance (min_dist) metric, which defines the minimum distance required between two points in the given distance metric. The probability used is:

$$q_{ij} \approx \begin{cases} 1 & \text{if } y_i - y_j \leq \text{min\_dist} \\ e^{-(y_i - y_j) - \text{min\_dist}} & \text{if } y_i - y_j > \text{min\_dist} \end{cases}$$

Once these values have been defined, the cost function used by UMAP is the binary cross entropy function to calculate the loss. This function is minimized to find the best low-dimensional representation of the high-dimensional data. The binary cross entropy function is defined as:

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

This minimization is achieved through a process known as stochastic gradient decent, which allows for fast and efficient computation of the minimal cross entropy value.

A visual representation of this process with a simplified explanation will be shown in the Figures 4, 5, and 6 that follow. First, let us consider a sample dataset such as the noisy sine wave below.

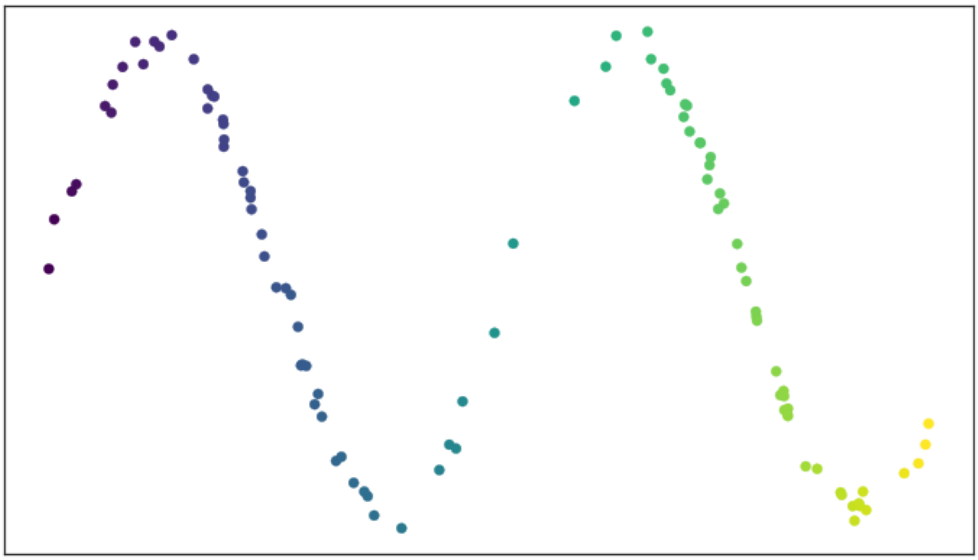

Figure 4: Random Sine Data

To create a weighted graph, we will create an open cover over each of the data points using circles of the same radius. This cover will be used to create our simplicial complexes for UMAP. The resulting covers are shown below

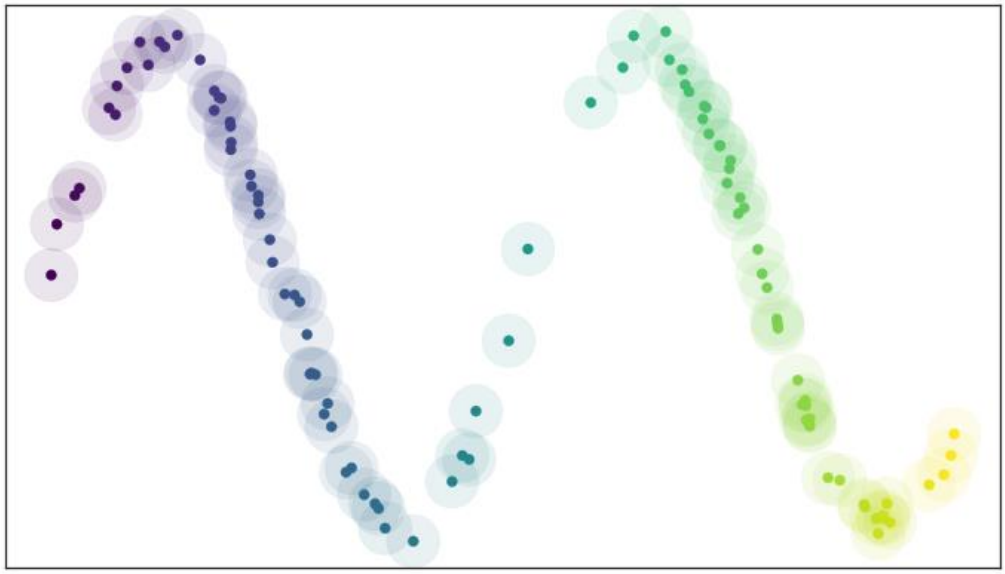

Figure 5: Random Sine Data with Circular Covers

We can now use these covers to create 0-, 1-, and 2- simplices which are represented by points, lines, and triangles respectively. We create these by drawing lines between points whose covers overlap, up to three points per simplex. The resulting graph of data points and simplices is shown below.

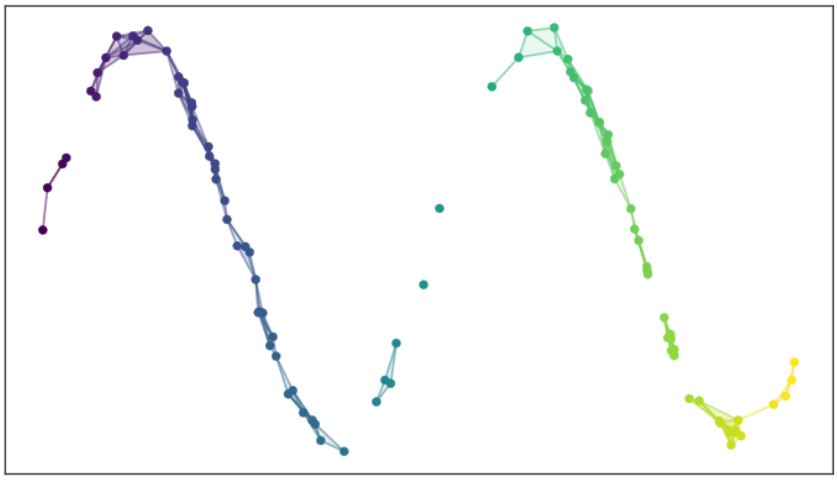

Figure 6: Random Sine Data with Simplices

Using the distances between these points, we can calculate weights in this graph that will be used to reduce the dimensionality of the graph. We show how these graphs are created in 2-dimensions since it is easy to visualize. The process remains the same for higher dimensional spaces. For data in *n*-dimensions, we start by creating *n*-dimensional spheres around each point. We then connect the points in the space using *n*-, *n-1*-, *…, 1*-, *0*- simplices and use the distance of the branches of the simplices as weights to reduce the dimensionality of the data.

A criticism of UMAP is its reliance on a manifold existing for the high-dimensional data being reduced. If no such manifold exists for the data, one has to be created and approximated for the algorithm to work. Though in practice, the likelihood of this occurring is slim.

A key advantage of UMAP is its speed, efficiency, and ability to retain the global data structure of the original dataset. This allows the practitioner of UMAP to observe with greater ease, the similarities in data. Furthermore, the parameters used in the UMAP algorithm such as number of nearest neighbors, minimum distance, and choice of distance metric are easy to interpret. Furthermore, UMAP can easily and efficiently reduce the dimension of any dataset to any dimension of the user's choice, which provides a clear advantage for data visualization.

A major advantage of UMAP as it exists now is the fully built UMAP library in Python. It takes full advantage of common libraries used in Python such as Pandas, Sci-kit Learn, and NumPy. These libraries have been fully optimized for performance, speed, and accuracy. Many authors, Leland McInnes being the chief among them, created a comprehensive library for UMAP that has been optimized for speed, efficiency, and accuracy. All the UMAP parameters, from the basic parameters such as nearest neighbors, minimum distance, and distance metric, are exposed by the library, which allows the user to easily test how different parameters affect the outcome of the dimensionality reduction. As an example, we will test UMAP on the MNIST handwritten digits dataset. This dataset contains 70,000 pixelated handwritten numbers ranging from 0-9. These digits were pixelated to a 28x28 grid. This results in a 784-dimensional point, that we reduce to a 2-dimensional point for easier visualization and classification. As an example, we show how UMAP reduces the data using 500 training samples per digit in Figures 7 and 8. Another advantage of the UMAP library in Python is we can choose supervised or unsupervised learning techniques to create the

2-dimensional points. Both techniques show reasonably separable data clusters, which allows for efficient classification. To further demonstrate UMAP's ability to create superior clusters, Figure 9 shows clustering on the MNIST dataset using k-means (left) against UMAP (right). Together, these figures clearly show superior clustering using UMAP.

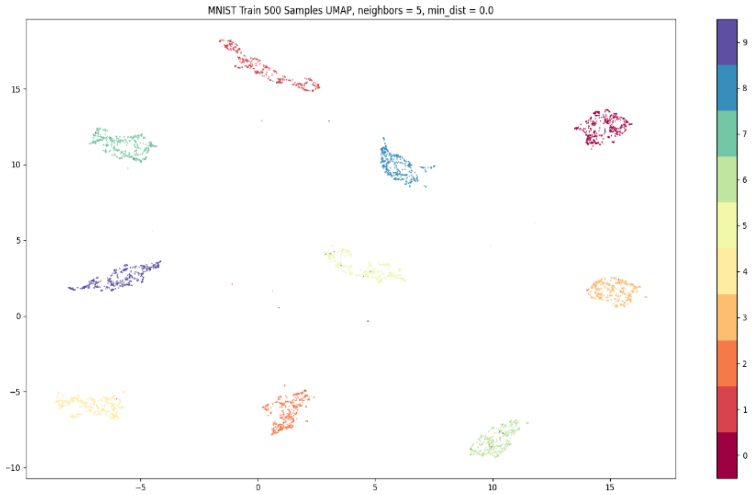

Figure 7: Training Data for MNIST using UMAP with hyperparameters| neighbors=5, min_dist=0.0

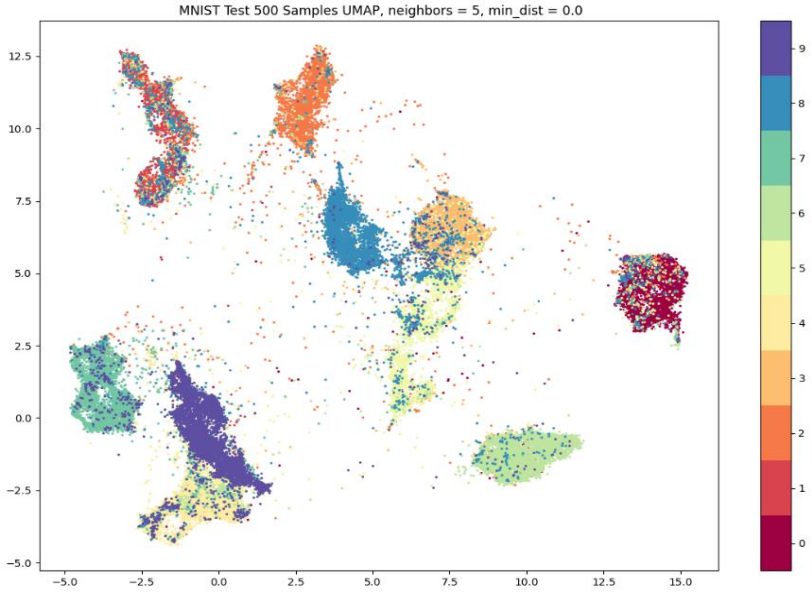

Figure 8: Testing Data for MNIST using UMAP with hyperparameters| neighbors=5, min_dist=0.0

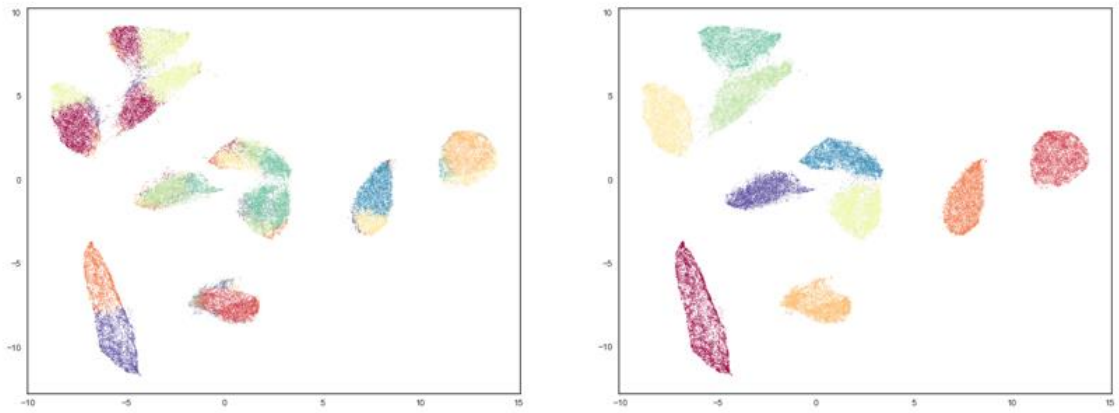

Figure 9: Clustering on the MNIST dataset using k-means (left) against UMAP (right)

4. Future Work

While UMAP does have a full-featured library, it may be useful to change stochastic gradient decent, which is the optimization algorithm used in UMAP, for TruSolve. The Binary Cross Entropy function is used as the cost function because it satisfies the smoothness criteria needed for stochastic gradient decent. While this does work well, the added flexibility that a non-linear stochastic optimizer such as TruSolve would add to this process is not to be underestimated. By introducing TruSolve as the optimization algorithm used in UMAP, we would be able to consider any cost function we like. The choice of cost function would be a new hyper-parameter we can use in the UMAP algorithm that may lead to new mappings that may in fact separate clusters more efficiently. Furthermore, as TruSolve is being further investigated, it is proving to be an extremely efficient and flexible optimization algorithm. Use of stochastic gradient descent for optimization has an inherent constraint in that it can only use differentiable cost functions. It also restricts the distance metrics that can be used. By only allowing for smooth and differentiable functions, it limits our ability to consider discrete distance metrics as a hyper-parameter.

5. Conclusion

The ability to reduce the dimensionality of a high-dimensional dataset to a more compact and easily managed dataset, while still maintain the global structure of the data is an incredibly powerful tool that we can utilize for artificial intelligence. NeRV and UMAP are two interesting methods for dimensionality reduction that have been extensively studied. Both methods have positive and negative aspects that have been examined in this white paper. Through our analysis of these two algorithms, we have determined that UMAP is the superior technique for dimensionality reduction for most of the cases tested. While UMAP does rely on the existence of a manifold, it preserves the global structure of the data more efficiently than NeRV. Furthermore, it separates the clusters better than NeRV, as demonstrated by the MNIST examples given in Figures 7, 8, and 9 above. As it stands now, UMAP has a fully flushed out Python library that has been extensively studied and upgraded for fully optimized performance. NeRV does not have a corresponding Python library, which leads to slower results and an inability to handle large datasets such as MNIST. UMAP can handle these large datasets with varying training set sizes and hyperparameters. Table 1 below provides a brief summary of UMAP and NeRV. It shows why UMAP is a better algorithm for clustering than NeRV.

Table 1: Review of UMAP and NeRV

	UMAP	NeRV
Speed	Best	Bad
Cluster Separability	Good	Bad
Programming Library	Best	Bad
Functionality	Good	Bad
Ease of Understanding	Good	Good
Use in Industry	Best	Bad

References

Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. **Information Retrieval Perspective to Nonlinear Dimensionality Reduction for Data Visualization.** *Journal of Machine Learning Research*, 11:451-490, 2010.

Jarkko Venna and Samuel Kaski. **Nonlinear Dimensionality Reduction as Information Retrieval.** In Marina Meila and Xiaotong Shen, editors, *Proceedings of AISTATS 2007*, the 11th International Conference on Artificial Intelligence and Statistics. Omnipress, 2007. JMLR Workshop and Conference Proceedings, Volume 2: AISTATS 2007.

Leland McInnes, Tim Sainburg, Graham Clendenning, Ayush Dalmia, Sebastian Pujalte. **UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.** https://umap-learn.readthedocs.io/en/latest/basic_usage.html

Kristian Nybo. **Dredviz: Dimensionality Reduction for Information Visualization**. *Dredviz,* Aalto University School of Science, https://research.cs.aalto.fi/pml/software/dredviz/.

Leland McInnes, John Healy, James Melville. **UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.** Journal of Open Source Software, 2018

Soham Joshi. **Modeling Contagion Networks using Topological Data Analysis and Reimannian Manifolds.** https://sjoshi5.github.io/projects/TDA/#/.