M.S. THESIS

Improving UMAP for Fast and Accurate Dimensionality Reduction

빠르고 정확한 차원 축소를 위한
UMAP 개선 방법

August 2021

Department of Computer Science and Engineering
College of Engineering
Seoul National University

Hyung-Kwon Ko

Improving UMAP for Fast and Accurate Dimensionality Reduction

Advisor: Jinwook Seo

Submitting a M.S. Thesis of
Computer Science and Engineering

April 2021

College of Engineering
Seoul National University

Hyung-Kwon Ko

Confirming the M.S. Thesis written by
Hyung-Kwon Ko

June 2021

Abstract

Hyung-Kwon Ko

Department of Computer Science and Engineering

College of Engineering | Seoul National University

One effective way of understanding the characteristics of high-dimensional data is to embed it onto a low-dimensional space. Among many existing dimensionality reduction algorithms, Uniform Manifold Approximation and Projection (UMAP) has gained the most attention because of its fast and stable projection result. However, still it is too slow to be adopted for an interactive visual analytics system as it takes for a few minutes to embed even for a toy dataset (e.g., MNIST). Moreover, UMAP is vulnerable to different configurations of hyperparameters, especially to the initialization methods and the number of epochs, which can bring about a serious bias mining insights from the embedding result.

To achieve the responsiveness, we propose a progressive algorithm for UMAP, called Progressive UMAP, for the exploration of datasets by updating the embedding with a batch of points through a progressive computation. Next, to guarantee less biases and the robustness in the embedding, we present a novel dimensionality reduction algorithm called Uniform Manifold Approximation with Two-phase Optimization (UMATO). We discover that the vulnerability comes from the approximation of cross-entropy loss function. UMATO, instead, takes a two-phase optimization approach: global optimization to obtain the overall skeleton of data, and local optimization to identify regional characteristics of a local area. In our experiment with one synthetic and three real-world datasets, UMATO outperformed widely-used baseline algorithms, such as PCA, t-SNE, UMAP, topological autoencoders and Anchor t-SNE, in terms of global quality metrics and 2D projection results. We further examine a case study of UMATO on real-world biological data and the extension to multi-phase optimization.

Our work makes the original contributions to the field of dimensionality reduction, as well as the progressive visual analytics. Lastly, the thesis discusses the future research directions for improving the proposed algorithms.

Keywords: Dimensionality Reduction, Manifold Learning, Topological Data Analysis, Visualization, UMAP

Student Number: 2019-24961

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in a era of big data. As people have more chance to deal with large-scale datasets, new algorithms and tools are required every year to extract useful information from them efficiently. One effective way to understand high-dimensional data is to reduce its dimensionality and project onto a low-dimensional space. If the properties of high-dimensional data are well-captured, we can lose a least amount of information, also attaining useful insights investigating projection results. Many dimensionality reduction algorithms – such as t-Stochastic Neighbor Embedding (t-SNE [36]), LargeVis [56], Uniform Manifold Approximation and Projection (UMAP [39]), topological autoencoders [43], and Anchor t-SNE (At-SNE [14]) – are invented for this aim, and used in diverse fields such as life sciences [1, 4, 37], physics [40], and computer science [21].

1.1 Motivation

Based on the theory of Riemannian geometry and algebraic topology, UMAP is an emerging dimensionality reduction technique. Among many existing algorithms, UMAP has gained the most attention these days by offering better versatility and stability than t-SNE [11].

Although UMAP is also more efficient than t-SNE, it still suffers from an initial delay of a few minutes to produce the first projection, which limits its use in interactive data exploration. This hinders UMAP's further researches on interactive visual analytics using large-scale datasets. Moreover, like most existing dimensionality reduction algorithms, UMAP embedding result is susceptible to different hyperparameter settings (e.g., number of epochs, initialization method); in most cases, it focuses on capturing **local structures** (i.e., the relationships between neighboring points in a high-dimensional space), while ignoring the **global structures** (i.e., the pairwise distance in a high-dimensional space) of datasets. However, such insufficiency can provide a biased and unstable view, leading to misinterpreting important patterns (e.g., a distance between clusters).

In this thesis, we tackle the problems of UMAP: 1) UMAP is not **fast** enough to satisfy the users who want to see the visualization result within a few seconds, as it is not an online nor a progressive algorithm, 2) UMAP is not **accurate** since it provides a biased result that is sensitive to different hyperparameter settings. By providing solutions to these critical problems, we offer a means to better understand the characteristics of dimensionality reduction algorithms, and pave a way on which to focus for future research directions.

1.2 Research Questions and Approaches

To address the aforementioned problems of UMAP, we postulate two research questions:

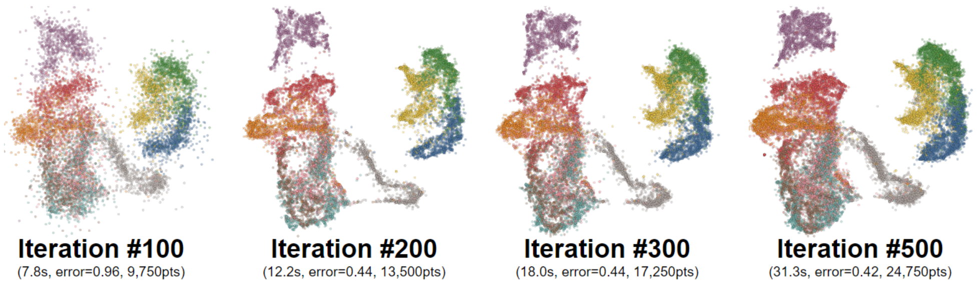

Figure 1.1: Responsive 2D Projections of the Fashion MNIST dataset [65] using Progressive UMAP.

RQ1. How should we explore high-dimensional data through UMAP without long initial computation delays? How do we make UMAP fast and interactive?

RQ2. How should we design a dimensionality reduction algorithm which is less biased and robust over different initialization methods? How do we make UMAP more accurate in terms of the preservation of both global and local structures into the projection?

1.2.1 Progressive Algorithm for UMAP

To answer on RQ1, we present a progressive algorithm for the UMAP, called the Progressive UMAP (Figure 1.1). We improve the sequential computations in UMAP by making them progressive, which allows people to incrementally append a batch of data points into the projection at the desired pace. In our experiment with the Fashion MNIST dataset, we found that Progressive UMAP could generate the first approximate projection within a few seconds while also sufficiently capturing the important structures of the high-dimensional dataset.

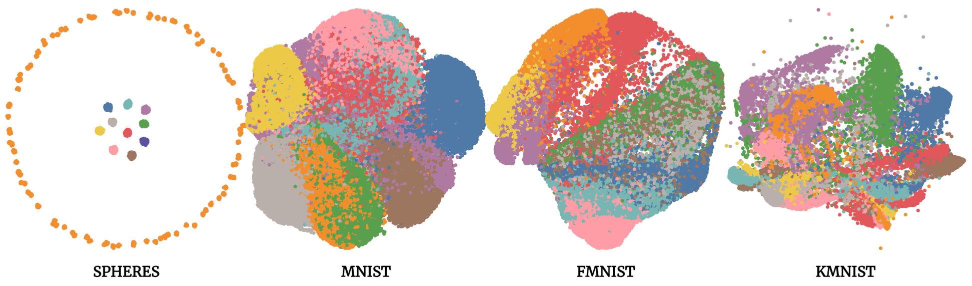

Figure 1.2: 2D projection results of UMATO using one synthetic Spheres [43], and three real-world image datasets [9, 30, 65].

1.2.2 Less Biased and Robust Dimensionality Reduction Algorithm

Addressing RQ2, we present a dimensionality reduction algorithm called Uniform Manifold Approximation with Two-phase Optimization (UMATO, Figure 1.2), which aims to preserve the global as well as the local structures of high-dimensional data. UMATO takes a two-phase optimization approach: global optimization to obtain the overall skeleton of data, and local optimization to identify regional characteristics of a local area. In our experiments with one synthetic and three real-world datasets, UMATO outperformed widely-used baseline algorithms, such as PCA, *t*-SNE, UMAP, topological autoencoders and Anchor *t*-SNE, in terms of global quality metrics and 2D projection results. We further discuss a case study to test UMATO on real-world biological data to show its usefulness and its extension to multi-phase optimization.

1.3 Contributions

The contributions of this thesis are as follows:

1. Development of Progressive UMAP, whxich can embed and visualize high-dimensional data responsively onto a 2D space, with its quantitative and qualitative evaluation.

2. Design, development, and evaluation of UMATO, which is less biased and robust over different initialization method.

Thesis Statement: The expansion of UMAP to progressive visual analytics and two-phase optimization enable an interactive exploration and produce less biased and robust embedding result.

1.4 Thesis Overview

Chapter 2 briefly describes the UMAP algorithm and its computational steps as a background. Next, the main parts of the thesis presents two applications of UMAP. Chapter 3 presents a progressive algorithm for UMAP, called the Progressive UMAP, which enables an interactive exploration of high-dimensional data through embedding a batch of points into the projection without a long initial delays. Chapter 4 delineates the UMATO, a dimensionality reduction algorithm aimed to capture both the global and local structures of high-dimensional data. After finishing the core of thesis, Chapter 5 discusses the lessons learned through the entire studies, and the limitations of each algorithm. Finally, Chapter 6 summarizes the contributions of the thesis.

Chapter 2

Background: UMAP

Although UMAP [39] is grounded in a complex mathematical foundation, its computation can be divided into two parts, graph construction and layout optimization, a configuration similar to t-SNE. In this section, we briefly explain the computation in an abstract manner. For more details about UMAP, please consult the original paper [39].

2.1 Graph Construction

UMAP starts by generating a weighted k-nearest neighbor graph that describes the distances between data points in the high-dimensional space. Given an input dataset $X = \{x_1, \ldots, x_N\}$, the number of neighbors to consider k and a distance metric $d : X \times X \to [0, \infty)$, UMAP first computes $\mathcal{N}_i$, the k-nearest neighbors of x_i with respect to d. Then, UMAP computes two parameters, ρ_i and σ_i, for each data point x_i to identify its local metric space. ρ_i is a nonzero distance from x_i to its nearest neighbor:

$$\rho_i = \min_{j \in \mathcal{N}_i} \{d(x_i, x_j) \mid d(x_i, x_j) > 0\}, \tag{2.1}$$

and the σ_i that satisfies the condition below is found using binary search:

$$\sum_{j \in \mathcal{N}_i} \exp(\frac{-\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}) = \log_2(k). \tag{2.2}$$

Using ρ_i and σ_i, UMAP computes $v_{j|i}$, the weight of the edge from a point x_i to another point x_j:

$$v_{j|i} = \exp(\frac{-\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}). \tag{2.3}$$

To make it symmetric, UMAP computes a single edge with combined weight using $v_{j|i}$ and $v_{i|j}$:

$$v_{ij} = v_{j|i} + v_{i|j} - v_{j|i} \cdot v_{i|j}. \tag{2.4}$$

Note that v_{ij} indicates the similarity between points x_i and x_j in the original space. Let y_i be the projection of x_i in a low-dimensional projection space. The similarity between two projected points y_i and y_j is:

$$w_{ij} = (1 + a||y_i - y_j||_2^{2b})^{-1}, \tag{2.5}$$

where a and b are positive constants defined by the user. Setting both a and b to 1 is identical to using Student's t-distribution to measure the similarity between two points in the projection space as in t-SNE [36].

2.2 Layout Optimization

The goal of layout optimization is to find the y_i that minimizes the difference (or loss) between v_{ij} and w_{ij}. In contrast to t-SNE where the Kullback-Leibler divergence between v_{ij} and w_{ij} is measured as the loss of the projection, UMAP measures the cross entropy between v_{ij} and w_{ij}:

$$C_{UMAP} = \sum_{i \neq j} [v_{ij} \cdot \log(\frac{v_{ij}}{w_{ij}}) - (1 - v_{ij}) \cdot \log(\frac{1 - v_{ij}}{1 - w_{ij}})]. \tag{2.6}$$

The authors of UMAP argued that UMAP returns clearer separation between clusters than t-SNE since C_{UMAP} gives penalties for forming both local and global structures.

y_i is initialized through spectral embedding [5] and iteratively optimized to minimize C_{UMAP}. Given the output weight w_{ij} as $1/(1 + ad_{ij}^{2b})$, the attractive gradient is:

$$\frac{C_{UMAP}}{y_i}^{+} = \frac{-2abd_{ij}^{2(b-1)}}{1 + ad_{ij}^{2b}} v_{ij}(y_i - y_j), \tag{2.7}$$

and repulsive gradient is:

$$\frac{C_{UMAP}}{y_i}^{-} = \frac{2b}{(\epsilon + d_{ij}^2)(1 + ad_{ij}^{2b})}(1 - v_{ij})(y_i - y_j), \tag{2.8}$$

where ϵ is a small value added to prevent division by zero and d_{ij} is a Euclidean distance between y_i and y_j. For efficient optimization, UMAP leverages the negative sampling technique from Word2Vec [41]. After choosing a target point and its negative samples, the position of the target is updated with the attractive gradient, while the positions of the latter do so with the repulsive gradient. Moreover, UMAP utilizes edge sampling [56, 57] to accelerate and simplify the optimization process. In other words, UMAP randomly samples edges with a probability proportional to their weights, and subsequently treats the selected ones as binary edges. Considering the previous sampling techniques, the modified objective function is:

$$O = \sum_{(i,j)\in E} v_{ij}(\log(w_{ij}) + \sum_{k=1}^{M} E_{j_k \sim P_n(j)} \gamma \log(1 - w_{ij_k})). \tag{2.9}$$

Here, v_{ij} and w_{ij} are the similarities in the high and low-dimensional spaces respectively, M is the number of negative samples and $E_{j_k \sim P_n(j)}$ indicates that j_k is sampled according to a noisy distribution, $P_n(j)$, from Word2Vec [41].

Chapter 3

Progressive UMAP: A Progressive Algorithm for UMAP

3.1 Introduction

This chapter presents a novel progressive algorithm for UMAP, called Progressive UMAP, to answer on the first research question; how should we explore high-dimensional data through UMAP without long initial computation delays? how do we make UMAP fast and interactive?.

We bring the UMAP [39], a popular nonlinear dimensionality reduction technique, into Progressive Visual Analytics (PVA). For more than a decade, t-Distributed Stochastic Neighbor Embedding (t-SNE [36]) has been one of the most widely-used dimensionality reduction techniques. However, UMAP, which is based on Riemannian geometry and algebraic topology, has recently emerged as an alternative to t-SNE, offering better efficiency and applicability [11]. UMAP stands out for its fast computation time. It is approximately three times faster than the state-of-the-art t-SNE implementation [33] and has a better stability between runs and support for non-metric distance measures, such as the cosine distance and correlation distance. Since its introduction, UMAP has been quickly adopted by diverse disciplines, such as life sciences [4], physics [40], and computer science [21], attesting to its usefulness. Although performance benchmarks [39] demonstrated that UMAP is much faster than t-SNE, it still is too slow to be used in interactive analysis effectively; when tested on a Macbook Pro with a 3.1 GHz Intel Core i7 and 8GB of RAM, UMAP took about 87 seconds to project the MNIST dataset onto a 2D space, far exceeding the time window of 10 seconds needed to keep the user's attention [42, 52]. The problem compounds as users often have to run the algorithm several times to tune its hyperparameters. To address this issue, we present a progressive algorithm for UMAP (Progressive UMAP). First, we identify a number of sequential computations in UMAP and explain how we improve each one by making it (Section 3.3). Next, we show that Progressive UMAP can yield partial projections of data every few seconds with a quality comparable to the original UMAP (Section 3.4).

3.2 Related Work

3.2.1 Progressive Visual Analytics

A series of studies have introduced and refined the concept of PVA [12, 13, 44, 55] to manage the computational delay caused by a large amount of data or the high complexity of algorithms. Progressive computation is defined as a computation that reports intermediate outputs within a bounded latency that converges towards the true result with an ability to control the execution.

One of the popular applications of PVA is the progressive projection of high-dimensional data. Among the existing dimensionality reduction techniques, t-SNE [36] has received the most attention from PVA researchers. Kim et al. [25] introduced a per-iteration visualization environment in which

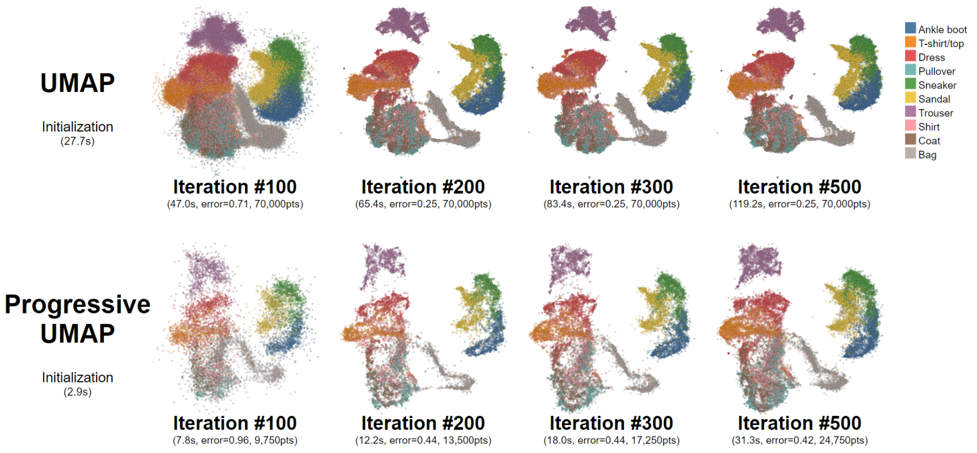

Figure 3.1: Projection of the Fashion MNIST dataset using UMAP and Progressive UMAP. Seventy thousand data points (784 dimensions) of the Fashion MNIST dataset have been projected onto a 2D space. UMAP took 27.7 s for initialization (i.e., neighbor computation and graph construction) that can interrupt interactive data exploration. In contrast, Progressive UMAP allowed users to project a small fraction of the points with shorter initialization time (2.9 s) and progressively append the remaining points to the projection. Although UMAP produced a smaller *average loss* at the end of iterations, Progressive UMAP could project the data points in a reasonable time-bound, sufficiently capturing the characteristics of the data.

users can interact in real time with algorithms that require complex computation, such as multidimensional scaling, t-SNE and latent Dirichlet allocation. Inspired by Ingram and Munzner's Q-SNE [22], Pezzotti et al. [49] presented a controllable t-SNE approximation, which enabled the interactive manipulation of t-SNE results, such as adding, removing and modifying data points. Similarly, Jo et al. [23] proposed a progressive algorithm for indexing and querying the approximated k-nearest neighbors. They also introduced responsive t-SNE, an application of their algorithm, which further reduced the initial delay of t-SNE using progressive neighbor computation. Motivated by these studies, we aim to develop a progressive algorithm for a popular dimensionality reduction technique, UMAP [39], which is more efficient and flexible than t-SNE [11].

Algorithm 1 Progressive Uniform Manifold Approximation and Projection

```
1: procedure ProgressiveUMAP($X$, $num\_iterations$, $ops$)
Input: High-dimensional data $X$, number of iterations $num\_iterations$, the allowed number of tasks per iteration $ops$
Output: Low-dimensional projection $Y$
2:     KNNTable ← new KNNTable(X), iterations ← 0
3:     while $iterations++ < num\_iterations$ do
4:         if size(KNNTable) < size(X) then
5:             $X_{new}$, $X_{updated}$ = KNNTable.run($ops$)
6:             set initial $y_i$ for points in $X_{new}$
7:             update $\rho_i$ (Equation 2.1) and $\sigma_i$ (Equation 2.2)
8:             compute $v_{j|i}$ (Equation 2.3)
9:             compute $v_{ij}$ (Equation 2.4)
10:        end if
11:        compute $\frac{C_{UMAP}}{y_i}$ (Equation 2.7, Equation 2.8)
12:        update $y_i$
13:    end while
14:    return $y_i$
15: end procedure
```

3.3 Progressive UMAP

We found that the current implementation of UMAP [39] could suffer a long initial delay depending on the size of the dataset, because it only works on a fixed set of data points with no support for adding new points progressively. In this section, we elaborate on our novel algorithm, Progressive UMAP (Algorithm 1), which allows users to feed small batches of data points into UMAP incrementally to obtain the desired latency between intermediate projection outputs. To this end, we identify sequential procedures in the original UMAP algorithm and transform them into progressive procedures. The source code for Progressive UMAP is publicly available [1].

[1]https://www.github.com/hyungkwonko/progressive-umap

3.3.1 Computing $\mathcal{N}_i$

To build and maintain the k-nearest neighbor graph, we leverage the KNN lookup table from PANENE [23]. PANENE employs randomized k-d trees [45] to approximate and update the k-nearest neighbors of an increasing number of data points. PANENE accepts a parameter called *ops* that indicates the allowed number of tasks per iteration that can be controlled to find the balance between latency and accuracy. For example, setting *ops* to a larger number will index more points per iteration, which will yield a more accurate projection at the cost of longer latency. Progressive UMAP starts with calling the update procedure of the KNN lookup table which returns two sets of points: X_{new} for newly inserted points and $X_{updated}$ for points whose neighbors are changed due to the insertion.

3.3.2 Computing ρ_i and σ_i

For every data point in $X_{updated}$ and X_{new}, we recompute ρ_i and σ_i according to Equation 2.1 and Equation 2.2. For space efficiency, UMAP used the coordinate list (COO) that only stores row, column, and value information as a list of tuples. Progressive UMAP updates the COO recalculating $v_{j|i}$ (Equation 2.3) for the selected points – $X_{updated}$ and X_{new} – and changing the corresponding values if there is a change from the previous ones. Last, we make the COO symmetric (Equation 2.4).

3.3.3 Layout Initialization

Although spectral embedding produces an effective initial projection, its quadratic time-complexity causes severe delay. Progressive UMAP initializes the positions of newly inserted points in two stages. For the first batch of points, 1) we run the algorithm with a large value of *ops* (e.g., 15,000), using the same spectral embedding technique as UMAP. Since we start with a relatively small number of points, this would take much less time than the original UMAP's spectral embedding. Hereafter, 2) we lower the value of *ops* (e.g., 1,000) not to focus on the appending process but to obtain an optimized projection output fast. Starting with the second batch, we set the initial projected position of each newly inserted point equal to its closest neighbor's position disturbed by a small Gaussian random noise to prevent collisions.

3.3.4 Layout Optimization

Analogously, we go through two stages for layout optimization. As it affects the overall time for convergence and increases the stability of the final output, it is very important to position the first batch of points well so that clusters are unambiguously separated. To this end, 1) we run more iterations (e.g., 40) in the first batch so each cluster can settle its position. Afterwards, 2) we run fewer iterations (e.g., 4) to focus on attaining the projection result fast. However, users can control the number of iterations for each stage; for example, if the size of data is small enough, they can set the algorithm to use the same number of iterations for both stages.

Based on the original UMAP implementation [38], our Progressive UMAP is written in Python. We leveraged PyNENE, a Python binding of PANENE [23], for the KNN lookup table as well as Numba [29] for parallel computation of distances, graph weights and optimization.

3.4 Evaluation and Discussion

To measure the loss of our method and UMAP at run-time we employ the objective function suggested by Tang et al. [57]. In their study, the authors proposed an objective function that subsumes all the edges (both observed

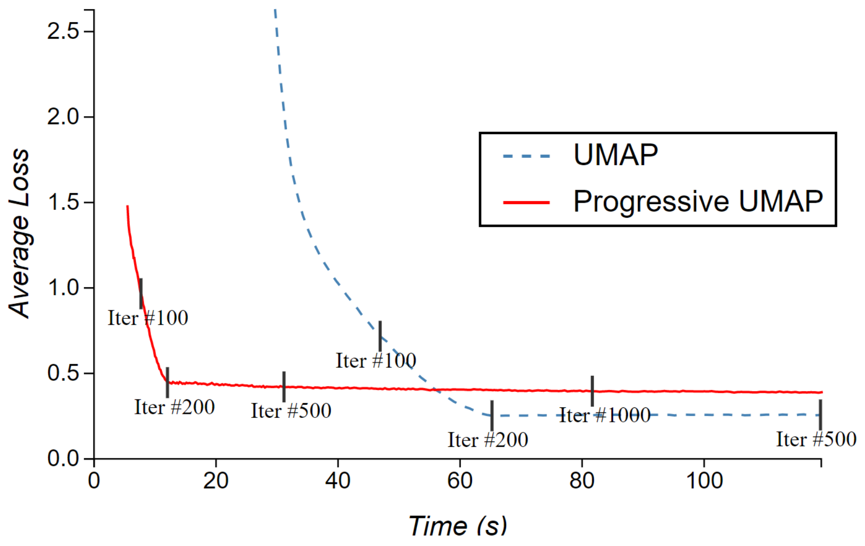

Figure 3.2: Average loss over time UMAP generated its first projection with an initial delay of 27.7 s (dotted line) while Progressive UMAP minimized the delay to 2.9 s. The loss of Progressive UMAP converged to 0.42 which was higher than that of UMAP (0.25). For ease of comparison, the loss at 100, 200 and 500 iterations was marked with grey vertical lines. After all points are added, the final loss of Progressive UMAP was 0.38.

and unobserved) to optimize a graph layout. As summing up all the edges in a complete graph is computationally expensive, they further suggested a fast method based on edge sampling, described as in Equation 2.9. However, applying the objective function directly to both algorithms induces bias since the total numbers of edges in UMAP and Progressive UMAP are different when data points are being inserted progressively in Progressive UMAP. To make the calculations comparable, we divide the loss by the number of sampled edges, a quantity which we will call unbiased loss or *average loss*.

For the evaluation, we ran both UMAP and Progressive UMAP on the Fashion MNIST dataset [65] which has 70,000 rows of 784 dimensions (28×28), each row describing an item from 10 classes (e.g., t-shirt, trouser, etc.). As a baseline, we used the original UMAP implementation [38]. Both algorithms were tested on a machine equipped with an Intel Core i7-4790K CPU (4.0 GHz) and 16 GB of main memory.

UMAP has several important hyperparameters: k (the number of neighbors to consider), min_dist (the minimum distance between points in the low-dimensional space) and *metric* (a metric to compute the distance between points in the high-dimensional space). We set k to 5, *min_dist* to 0.1, and *metric* to the Euclidean distance.

Figure 3.1 shows intermediate 2D projection outputs generated by each algorithm over time. Each point denotes a single row of 784 dimensions in the high-dimensional space, color-coded by class. The initialization process of UMAP took 27.7 seconds as it considered all the data points at the beginning. In contrast, Progressive UMAP took 2.9 seconds to initialize because it could incrementally append data points in later iterations. Similarly, the time required to reach the same *average loss* was faster in Progressive UMAP; Progressive UMAP took 11.1 seconds to obtain the *average loss* of 0.5, while the original UMAP took 52.8 seconds (Figure 4.1). Although Progressive UMAP produced a bigger *average loss* at the end, it located points much faster than UMAP and converged in a reasonable time-bound, sufficiently capturing the important characteristics of the dataset; for example, the intermediate outputs of Progressive UMAP at the 200th iteration (Figure 3.1) manifest a clear separation of data points between clusters.

Next, we tested the effect of *ops* (i.e., the parameter passed to the KNN lookup table) and the number of iterations on optimizing the first projection. We gradually changed *ops* from 300 to 1,000 with an interval of 100. We found it is possible to control the computation time of an iteration by changing ops. However, if ops is too small, it can harm the stability of the projection result since, if we insert too few points at a time, it is impossible to choose the nearest neighbors that capture the local manifold robustly. We also examined the effect of the number of iterations on optimizing the projection (y_i) for the first batch by changing it from 20 to 60 with an interval of 10. As expected, keeping the number of iterations small in the first batch worsened the convergence speed and stability of the projection result. On the other hand, having a large number of iterations in the first batch helped achieve better projection quality but also slowed down convergence. The results of these experiments are also available in our repository.

To sum up, we found that Progressive UMAP is able to not only generate intermediate projection outputs whose quality is comparable to the original UMAP within a reasonable time-bound but also provides an ability to control the trade-off between computation time and the quality of projection.

3.5 Summary

We present a progressive algorithm for the Uniform Manifold Approximation and Projection (Progressive UMAP). Through our quantitative evaluation, we found that Progressive UMAP can generate the approximate projection and update it every few seconds.

Chapter 4

UMATO: A Less Biased and Robust Dimensionality Reduction Algorithm Based on UMAP

4.1 Introduction

This chapter presents a novel dimensionality reduction method, Uniform Manifold Approximation with Two-phase Optimization (UMATO) that obtains less biased and robust embedding over diverse initialization methods. One effective way of understanding high-dimensional data in various domains is to reduce its dimensionality and investigate the projection in a lower-dimensional space [24, 35, 56]. Dimensionality reduction techniques have become the most useful tool for exploring data in applications of many visual analytics systems [15, 23]. Widely-used previous approaches such as t-Stochastic Neighbor Embedding (t-SNE [36]) and Uniform Manifold Approximation and Projection (UMAP [39]) have their innate limitation that they are susceptible to initialization methods, generating considerably different embedding results depending on the initialization method. (Equation 4.6).

t-SNE adopts Kullback-Leibler (KL) divergence as its loss function. The fundamental limitation of the KL divergence is that the penalty for the points that are distant in the original space being close in the projected space is too little (Section 4.3). This results in only the local manifolds being captured, while clusters that are far apart change their relative locations from run to run. Meanwhile, UMAP leverages the cross-entropy loss function, which is known to charge a penalty for points that are distant in the original space being close in the projection space and for points that are close in the original space being distant in the projection space (Section 4.3). Since computing the loss with all points requires too much time, UMAP utilizes diverse sampling techniques (i.e., negative sampling and edge sampling). Although the approximation techniques in UMAP optimization make the computation much faster, this raises another problem that the clusters in the embedding become dispersed as the number of epochs increases (Figure 4.6), which can lead to misinterpretation. UMAP tried to alleviate this by using a fixed number (e.g., 200), which is chosen ad hoc, and by applying a learning rate decay. However, the optimal number of epochs and decay schedule for each initialization method needs to be found in practice.

To solve the aforementioned problems, we avoid using approximation during the optimization process, but it normally would result in greatly increased computational cost. Instead, we first run optimization only with a small number of points that represent the data (i.e., hub points). Finding the optimal projection for a small number of points using a cross-entropy function is relatively easy and robust, making the additional techniques (e.g., negative sampling, edge sampling) employed in UMAP unnecessary. Furthermore, it makes UMAP less sensitive to the initialization method used (Table 4.5). After capturing the overall skeleton of the high-dimensional structure with the hub points, we gradually append the rest of the points in subsequent phases. Although the same approximation techniques as UMAP are used for these points, the projection becomes more robust and unbiased since we have already embedded the hub points and use them as anchors. The gradual addition of points can in fact be done in a single phase; we found additional phases do not meaningfully improve the performance while increasing computation time (Table 4.6). Therefore, we used only two phases in UMATO; the first is global optimization to capture the global structures (i.e., the pairwise similarities in a high-dimensional space) and the second is local optimization to retain the local structures (i.e., the relationships between neighboring points in a high-dimensional space) of the data.

We compared UMATO with popular dimensionality reduction techniques including PCA, Isomap [59], *t*-SNE, UMAP, topological autoencoders [43] and A*t*-SNE [14]. We used one synthetic (101-dimensional spheres) and three real-world (MNIST, Fashion MNIST, and Kuzushiji MNIST) datasets and analyzed the projection results with several quality metrics. In conclusion, UMATO demonstrated better performance than the baseline techniques in all datasets in terms of KL_σ with different σ values, meaning that it reasonably preserved the density of data over diverse length scales. We also presented the 2D projections of each dataset, including the replication of an experiment using the synthetic Spheres dataset introduced by Moor et al. [43] where data points locally constitute multiple small balls globally contained in a larger sphere. Here, we demonstrate that UMATO can better preserve both structures compared to the baseline algorithms (Figure 4.1). Finally, we

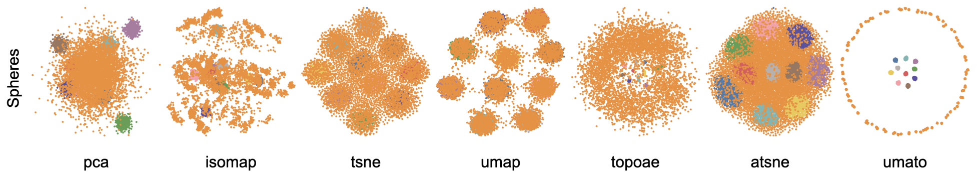

Figure 4.1: 2D projections produced by UMATO and six baseline algorithms. t-SNE, At-SNE, and UMAP showed as if the points from a surrounding sphere were attached to inner spheres, not reflecting the data's global structures. PCA, Isomap and topological autoencoders attempted to preserve the global structures, but failed to manifest the complicated hierarchical structures. UMATO was the only algorithm to capture both the global and local structures among all different sphere classes; this is best viewed in color.

apply UMATO on a real-world biological dataset as a case study to show that it can reveal the hidden structures faithfully (Section 4.5.2).

4.2 Related Work

4.2.1 Dimensionality Reduction

There is a body of research in the field of dimensionality reduction which spans out from machine learning to visualization communities [11, 47, 63]. Most previous dimensionality reduction algorithms focused on preserving the data's local structures. For example, Maaten et al. [36] proposed t-SNE, focusing on the crowding problem with which previous attempts [10, 20] have struggled, to visualize high-dimensional data through projection produced by performing stochastic gradient descent on the KL divergence between two density functions in the original and projection spaces. L. van der Maaten [62] accelerated t-SNE developing a variant of the Barnes-Hut algorithm [3] and reduced the computational complexity from $O(N^2)$ into $O(N \log N)$. After that, grounded in Riemannian geometry and algebraic topology, McInnes et al. [39] introduced UMAP as an alternative to t-SNE. Leveraging the cross-entropy function as its loss function, UMAP reduced the computation time by employing negative sampling from Word2Vec [41] and edge sampling from LargeVis [56, 57]. Moreover, they showed that UMAP can generate stable projection results compared to t-SNE over repetition.

On the other hand, there also exist algorithms that aim to capture the global structures of data. Isomap [59] was proposed to approximate the geodesic distance of high-dimensional data and embed it onto the lower dimension. Global t-SNE [67] converted the joint probability distribution, P, in the high-dimensional space from Gaussian to Student's-t distribution, and proposed a variant of KL divergence. By adding it with the original loss function of t-SNE, Global t-SNE assigns a relatively large penalty for a pair of distant data points in high-dimensional space being close in the projection space. Another example is topological autoencoders [43], a deep-learning approach that uses a generative model to make the latent space resemble the high-dimensional space by appending a topological loss to the original reconstruction loss of autoencoders. However, they required a huge amount of time for hyperparameter exploration and training for a dataset, and only focused on the global aspect of data. Unlike other techniques that presented a variation of loss functions in a single pipeline, UMATO is novel as it preserves both global and local structures by dividing the optimization into two phases; this makes it outperform the baselines with respect to quality metrics in our experiments.

4.2.2 Hubs, landmarks, and anchors

Many dimensionality reduction techniques have tried to draw sample points to better model the original space; these points are usually called hubs, landmarks, or anchors. Silva et al. [53] proposed Landmark Isomap, a landmark version of classical multidimensional scaling (MDS) to alleviate its computation cost. Based on the Landmark Isomap, Yan et al. [66] tried to retain the topological structures (i.e., homology) of high-dimensional data by approximating the geodesic distances of all data points. However, both techniques have the limitation that landmarks were chosen randomly without considering their importance. Hierarchical Stochastic Neighbor Embedding (Hierarchical SNE [48]) chose meaningful landmarks using the k-nearest neighbor graph and transition matrix to increase the interactivity of ongoing process at the cost of accuracy. To take advantage of the computational efficiency in handling big data, Hierarchical SNE aimed at exploring the projection results in diverse scales using filtering and drilling down. Similarly, UMATO extracts significant hubs that can represent the overall skeleton of high-dimensional data, but our goal, however, is to generate a less biased and robust projection using two-phase optimization compared to the one with single optimization. The most similar work to ours is At-SNE [14], which optimized the anchor points and all other points with two different loss functions. Nonetheless, since the anchors wander during the optimization and the KL divergence does not care about distant points (Section 4.3), it hardly captures the global structure. UMATO separates the optimization process into two phases so that the hubs barely moves but guides other points so that the subareas manifest the shape of the high-dimensional manifold in the projection. Applying different cross-entropy functions to each phase also helps preserve both structures.

		Low d	
		Large w_{ij}	Small w_{ij}
High d	Large v_{ij}	a. Small penalty	b. Big penalty (preserves local structures)
	Small v_{ij}	c. Small penalty (ignores global structures)	d. Small penalty

Table 4.1: Analysis of the KL divergence and the first term of cross-entropy loss function for imposing penalties when updating the positions of points in low-dimensional space. Both impose a big penalty when w_{ij} is small but v_{ij} is large.

		Low d	
		Large w_{ij}	Small w_{ij}
High d	Large v_{ij}	e. Small penalty	f. Small penalty (ignores local structures)
	Small v_{ij}	g. Big penalty (preserves global structures)	h. Small penalty

Table 4.2: The same analysis as Table 4.2 for the second term of cross-entropy loss function. The second term of cross-entropy function imposes a big penalty when v_{ij} is small but w_{ij} is large.

4.3 The Meaning of Using Different Loss Functions in Dimensionality Reduction

4.3.1 t-SNE

t-SNE is one of the most popular nonlinear dimensionality reduction algorithms that is widely used for data analysis. Given a set of high-dimensional points $X = \{x_1, x_2, \ldots, x_n\}$, t-SNE maps it into a set of low-dimensional points $Y = \{y_1, y_2, \ldots, y_n\}$. Next, t-SNE calculates the similarity between the points in high-dimensional space (v_{ij}) and low-dimensional space (w_{ij}). Note that v_{ij} indicates the similarity between points x_i and x_j in the original

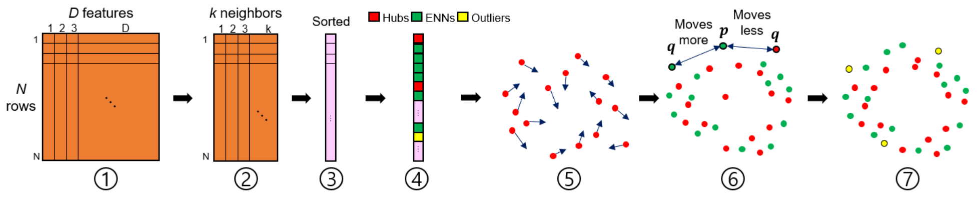

Figure 4.2: The illustration of overall UMATO pipeline. ② We first find the k-nearest neighbors for each point. ③ Then, we count the frequency of each point in the KNN table and sort them in decreasing order. The reason for doing this is to find the most popular points that are largely connected to other ones. ④ Next, we divide the points into three different classes: hub points, expanded nearest neighbors, and outliers. ⑤ We initialize the positions of hub point, and optimize their positions through global optimization. ⑥ Using the hubs' positions, we embed the expanded nearest neighbors, and run local optimization. ⑦ Finally, we embed the outliers.

space and w_{ij} refers to the similarity between points y_i and y_j in the projection space. To compensate the crowding problem [36], it uses a Student t-distribution with one degree of freedom to compute the similarity in low-dimensional space. Lastly, it optimizes the embedded points by minimizing the differences between v_{ij} and w_{ij} using KL divergence:

$$KL = \sum_{i \neq j} v_{ij} \cdot \log(v_{ij}/w_{ij}). \tag{4.1}$$

The KL divergence (Equation 4.1) and the first term of cross-entropy (Equation 2.6) are exactly the same. If v_{ij} and w_{ij} are both large (Table 4.1a) or both small (Table 4.1d), this means that the relationship between points in the original space is well-retained in the projection space. Thus, the positions of points in the projection space do not have to move. As v_{ij} and w_{ij} are similar, $\log(v_{ij}/w_{ij})$ becomes zero, producing a small cost in the end. However, we need to modify the position of w_{ij} if v_{ij} and w_{ij} are different (e.g., v_{ij} is large, but w_{ij} is small; or v_{ij} is small, but w_{ij} is large). The KL divergence imposes a big penalty when v_{ij} is large but w_{ij} is small (Table 4.1b). That is,

Algorithm 2 Uniform Manifold Approximation with Two-phase Optimization (UMATO)

```
1:  procedure UMATO($X, k_X, d, min\_dist, n_h, e_g, e_l$)
Input: High-dimensional data $X$, number of nearest neighbors $k$, projection dimension $d$, minimum distance in projection result $min\_dist$, number of hub points $n_h$, epochs for global and local optimization $e_g$, $e_l$
Output: Low-dimensional projection $Y$
2:      Compute $k$-nearest neighbors of $X$
3:      Obtain sorted list using indices' frequency of $k$-nearest neighbors
4:      Build $k$-nearest neighbor graph structure
5:      Classify points into hubs, expanded nearest neighbors, and outliers
6:      Optimize $CE(f(X_h)||g(Y_h))$ to preserve global configuration (??)
7:      Initialize expanded nearest neighbors using hub locations
8:      Update $k$-nearest neighbors & compute weights (Equation 2.3)
9:      Optimize $CE(f(X)||g(Y))$ to preserve local configuration (Equation 2.9)
10:     Position outliers
11:     return $Y$
12: end procedure
```

if the neighboring points in the high-dimensional space are badly captured in the low-dimensional space, the KL divergence imposes a high penalty to move the point (v_{ij}) into the right position. Thus, we can understand why t-SNE is able to capture the local characteristics of high-dimensional space, but not the global ones. However, the second term of cross-entropy imposes a big penalty when v_{ij} is small but w_{ij} is large (Table 4.2g). Therefore, it moves points that are close together in the high-dimensional space but far apart in the low-dimensional one so that w_{ij} becomes small in the final projection.

4.4 UMATO

In UMATO, as a novel approach, we split the optimization into global and local so that it could generate a low-dimensional projection keeping both structures well-maintained. For the ease of understanding, we provide an

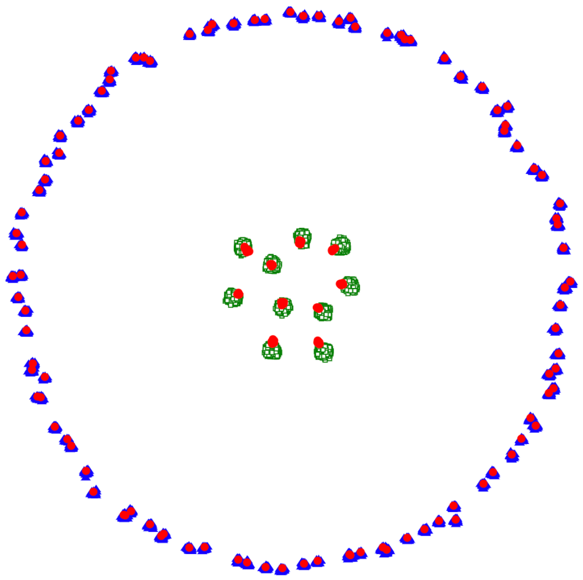

Figure 4.3: Points classification using Spheres dataset. Each point is classified into a hub (red circles), an expanded nearest neighbor (green squares), or an outlier (blue triangles). Best viewed in color.

illustration of UMATO pipeline in Figure 4.2. We present the pseudocode of UMATO in Algorithm 2, and made the source codes of it publicly available[1].

4.4.1 Points Classification

In the big picture, UMATO follows the pipeline of UMAP. We first find the k-nearest neighbors in the same way as UMAP, by assuming the local connectivity constraint, i.e., no single point is isolated and each point is connected to at least a user-defined number of points. After calculating ρ (Equation 2.1) and σ (Equation 2.2) for each point, we obtain the pairwise similarity for every pair of points. Once the k-nearest neighbor indices are established, we unfold it and check the frequency of each point to sort them into descending order so that the index of the popular points come to the front.

[1]https://www.github.com/hyungkwonko/umato

Then, we build a k-nearest neighbor graph by repeating the following steps until no points remain unconnected: 1) choose the most frequent point as a hub among points that are not already connected, 2) retrieve the k-nearest neighbors of the chosen point (i.e., hub), and the points selected from steps 1 and 2 will become a connected component. The gist is that we divide the points into three disjoint sets: hubs, expanded nearest neighbors, and outliers (Figure 4.3). Thanks to the sorted indices, the most popular point in each iteration—but not too densely located—becomes the hub point. Once the hub points are determined, we recursively seek out their nearest neighbors and again look for the nearest neighbors of those neighbors, until there are no points to be newly appended. In other words, we find all connected points that are expanded from the original hub points, which, in turn, is called the expanded nearest neighbors. Any remaining point that is neither a hub point nor a part of any expanded nearest neighbors is classified as an outlier. The main reason to rule out the outliers is, similar to the previous approaches [16, 48], to achieve the robustness of the manifold learning algorithm in practice. As the characteristics of these classes differ significantly, we take a different approach for each class of points to obtain both structures. That is, we run global optimization for the hub points (Section 4.4.2), local optimization for the expanded nearest neighbors (Section 4.4.3), and no optimization for the outliers (Section 4.4.4). In the next section we explain each in detail.

4.4.2 Global Optimization

After identifying hub points, we run the global optimization to retrieve the skeletal layout of the data. First, we initialize the positions of hub points using PCA, which makes the optimization process more stable than using random initial positions [28]. Next, we optimize the positions of hub points by minimizing the cross-entropy function (Equation 2.6). Let $f(X) = \{f(x_i, x_j)| x_i, x_j \in X\}$ and $g(Y) = \{g(y_i, y_j)|y_i, y_j \in Y\}$ be two adjacency matrices in high- and low-dimensional spaces. If X_h represents a set of points selected as hubs in high-dimensional space, and Y_h is a set of corresponding points in the projection, we minimize the cross entropy—$CE(f(X_h)||g(Y_h))$—between $f(X_h)$ and $g(Y_h)$.

UMAP computes the cross-entropy between all existing points using two sampling techniques, edge sampling and negative sampling, for speedup (Table 4.7). However, this often ends up capturing only the local properties of data because of the sampling biases and thus it cannot be used for cases that require a comprehensive understanding of the data. On the other hand, in its first phase, UMATO only optimizes for representatives (i.e., the hub points) of data, which takes much less time but can still approximate the manifold effectively.

4.4.3 Local Optimization

In the second phase, UMATO embeds the expanded nearest neighbors to the projection that is computed using only the hub points from the first phase. For each point in the expanded nearest neighbors, we retrieve its nearest m (e.g., 10) hubs in the original high-dimensional space and set its initial position in the projection to the average positions of the hubs in the projection with small random perturbations. We follow a similar optimization process as UMAP in the local optimization with small differences. As explained in Section 2.1, UMAP first constructs the graph structure; we perform the same task but only with the hubs and expanded nearest neighbors. While doing this, since some points are excluded as outliers, we need to update the k-

		Global quality metrics				Local quality metrics (k = 5)			
Dataset	Algorithm	DTM	KL$_{0.01}$	KL$_{0.1}$	KL$_1$	Cont	Trust	MRRE$_X$	MRRE$_Z$
Spheres	PCA	0.9950	0.7568	0.6525	0.0153	0.7983	0.6088	0.7985	0.6078
	Isomap	0.7784	0.4492	0.4267	0.0095	**0.9041**	0.6266	**0.9039**	0.6268
	t-SNE	0.9144	0.6091	0.5399	0.0130	**0.8916**	**0.7078**	**0.9045**	**0.7241**
	UMAP	0.9209	0.6100	0.5383	0.0134	0.8760	0.6499	0.8805	0.6494
	TopoAE	**0.6890**	**0.2063**	**0.3340**	**0.0076**	0.8317	0.6339	0.8317	0.6326
	At-SNE	0.9448	0.6584	0.5712	0.0138	0.8721	0.6433	0.8768	0.6424
	UMATO (ours)	**0.3888**	**0.1341**	**0.1434**	**0.0014**	0.7884	**0.6558**	0.7887	**0.6557**
Fashion MNIST	PCA	0.2315	0.6929	0.0454	**0.0006**	0.9843	0.9117	0.9853	0.9115
	Isomap	**0.2272**	**0.6668**	**0.0446**	0.0010	0.9865	0.9195	0.9872	0.9196
	t-SNE	0.2768	0.8079	0.0663	0.0017	0.9899	**0.9949**	0.9919	**0.9955**
	UMAP	0.2755	0.8396	0.0641	0.0016	**0.9950**	0.9584	**0.9955**	0.9584
	TopoAE	0.2329	0.7301	0.0446	**0.0008**	0.9908	0.9591	0.9913	0.9590
	At-SNE	0.2973	0.8389	0.0702	0.0017	0.9826	**0.9847**	0.9849	**0.9848**
	UMATO (ours)	**0.2035**	**0.6852**	**0.0342**	0.0008	**0.9911**	0.9500	**0.9919**	0.9502
MNIST	PCA	0.4104	1.4981	0.1349	0.0014	0.9573	0.7340	0.9605	0.7342
	Isomap	**0.3358**	**1.0361**	**0.0857**	**0.0012**	0.9743	0.7527	0.976	0.7528
	t-SNE	0.4263	1.4964	0.1523	0.0024	**0.9833**	**0.9954**	**0.9869**	**0.9963**
	UMAP	0.4172	1.5734	0.1430	0.0026	**0.9891**	0.9547	**0.9907**	0.9547
	TopoAE	0.3686	1.3818	0.1048	**0.0011**	0.9716	0.9429	0.9732	0.9429
	At-SNE	0.4328	1.5623	0.1482	0.0018	0.9768	**0.9765**	0.9830	**0.9777**
	UMATO (ours)	**0.3525**	**1.2785**	**0.1017**	0.0014	0.9792	0.8421	0.9813	0.8422
Kuzushiji MNIST	PCA	0.4215	0.1710	0.1317	0.0014	0.9380	0.7213	0.9420	0.7211
	Isomap	**0.3458**	0.2171	**0.0906**	**0.0012**	0.9573	0.7638	0.9589	0.7635
	t-SNE	0.4254	**0.0483**	0.1369	0.0025	0.9843	**0.9688**	0.9871	**0.9693**
	UMAP	0.3873	**0.0417**	0.1148	0.0026	**0.9893**	0.9563	**0.9908**	0.9564
	TopoAE	0.3730	0.1495	0.1027	**0.0011**	0.9755	0.9442	0.9768	0.9440
	At-SNE	0.3505	0.0807	0.0978	0.0013	0.9786	**0.9671**	0.9824	**0.9676**
	UMATO (ours)	**0.3231**	0.1365	**0.0815**	0.0016	**0.9865**	0.8888	**0.9881**	0.8895

Table 4.3: Quantitative results of UMATO and six baseline algorithms. The hyperparameters of the algorithms are chosen to minimize $\mathrm{KL}_{0.1}$. The best one is in bold and underlined, and the runner-up is in bold. Only first four digits are shown for conciseness.

nearest neighbor indices. This is fast because we recycle the already-built k-nearest neighbor indices by replacing the outliers to the new nearest neighbor.

Once we compute the similarity between points (e.g., v_{ij} and w_{ij}), to optimize the positions of points, similar to UMAP, we use the cross-entropy loss function with edge sampling and negative sampling (Equation 2.9). Here, we try to fix the positions of hubs as much as possible since they have already formed the global structure. Thus, we only sample a point p among the expanded nearest neighbors as one end of an edge, while the point q at the other end of the edge can be chosen from all points except outliers. In UMAP implementation, when q pulls in p, p also drags q to facilitate the optimization (Equation 2.7). When updating the position of q, we only give a penalty to this (e.g., 0.1), if q is a hub point, not letting its position excessively be affected by p. In addition, because the repulsive force can disperse the local attachment, making the point veer off for each epoch and eventually destroying the well-shaped global layout, we multiply a penalty (e.g., 0.1) when calculating the repulsive gradient (Equation 2.8) for the points selected as negative samples.

4.4.4 Outliers Arrangement

Since the isolated points, which we call outliers, mostly have the same distance to all the other data points in high-dimensional space, due to the curse of dimensionality, they both sabotage the global structure we have already made and try to mingle with all other points, thus distorting the overall projection. We do not optimize these points but instead simply append them using the already-projected points (e.g., hubs or expanded nearest neighbors), that belong to each outlier's connected component of the nearest neighbor graph. That is, if $x_i \in C_n$ where x_i is the target outlier and C_n is the connected component to which x_i belongs, we find $x_j \in C_n$ that has already been projected and is closest to x_i. We arrange y_i which corresponds to x_i in low-dimensional space using the position of y_j in the same component offset by a random noise. In this way, we can benefit from the comprehensive composition of the projection that we have already optimized when arranging the outliers. We can ensure that all outliers can find a point as its neighbor since we picked hubs from each connected component of the nearest neighbor graph and thus at least one point is already located and has an optimized position (Section 4.4.2).

4.5 Experiments

4.5.1 Quantitative and Qualitative Evaluation of UMATO Compared to Six Baseline Algorithms

We conducted experiments on one synthetic and three real-world datasets to evaluate UMATO's ability to capture the global and local structures of high-dimensional data [51]. We compared UMATO with six baseline algorithms, PCA, Isomap, t-SNE, UMAP, topological autoencoders, and At-SNE in terms of global (i.e., DTM and KL_σ) and local (i.e., trustworthiness, continuity, and MRREs) quality metrics.

Datasets

Synthetic Spheres Dataset. We leveraged the same Spheres dataset that Moor et al. [43] used in their experiments of topological antoencoders. The Spheres dataset contains eleven high-dimensional spheres which reside in 101 dimensional space having 10,000 rows of 101 dimensions. We first generated ten spheres of radius of 5, and shifted each sphere by adding the same Gaussian noise to a random direction. For this aim, we created d-dimensional Gaussian vectors $X \sim N(0, I(10/\sqrt{d}))$, where d is 101 and the number of points in each sphere is 500. As to embed an interesting geometrical structure to the dataset, the ten spheres of relatively small radii of 5 were enclosed by another larger sphere of radius of 25. The number of points in this outer sphere was set to 5,000.

Real-world Image Datasets. To test UMATO on real-world datasets we used three well-known image datasets for the experiments: MNIST [30], Fashion

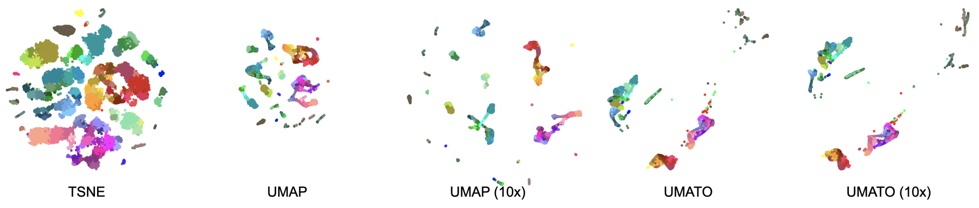

Figure 4.4: 2D visualization of UMATO, UMAP and t-SNE on mouse neocortex dataset [58]. t-SNE separates clusters well but does not explicitly present class information: GABAergic (red/purple), Endothelial (brown), Glutamatergic (blue/green), Non-Neuronal (dark green). UMAP moderately captures both the clusters and classes. In the case of UMATO, it demonstrates the relationship between classes much better than t-SNE and UMAP, retaining some of the local manifolds as well. When the number of epochs for UMAP is changed from the default value (200) to 2,000, the distance between clusters becomes exaggerated. However, UMATO generates more consistent projections regardless of the number of epochs, which does not induce any bias in analyzing inter-cluster distances.

MNIST [65], and Kuzushiji MNIST [9]. The datasets represent images of digits, fashion items, and Japanese characters where each of which consists of 60,000 rows of 784-dimensional (28×28) images from 10 classes.

Experimental setting

Evaluation Metrics. As UMATO presents a dimensionality reduction technique that can capture both the global and local structures of high-dimensional data, we used several quality metrics to evaluate UMATO from diverse aspects. We refer to previous review papers [18, 32] for the best use and implementation of the quality metrics. To assess how well projections preserve the global structures of high-dimensional data, we computed the density estimates, so-called Distance To a Measure (DTM [7, 8]), between the original data and the projections. DTM considers the dispersion of high- and low-dimensional data, where it is defined for a given point as $f_\sigma^X(x) := \sum_{y \in X} \exp\left(-\text{dist}(x, y)^2/\sigma\right)$. By summing up the element-wise absolute values between two distributions, $\sum_{x \in X, z \in Z} f_\sigma^X(x) - f_\sigma^Z(z)$ where x is the point in high-dimensional space X and the z is the corresponding projected point in low-dimensional space Z, we can examine the similarity of two datasets. In our experiments, we used the Euclidean distance and the values were normalized between 0 and 1. The $\sigma \in \mathcal{R}_{>0}$, which represents the length scale parameter, was set to 0.1. Moor et al. [43] adopted the Kullback-Leibler divergence between density estimates with different scales ($\mathrm{KL}_\sigma := \mathrm{KL}(f_\sigma^X || f_\sigma^Z)$) to evaluate the global structure preservation. Following the same notion of the experiments in the paper [43], we used three σ values, 1.0, 0.1, and 0.01, to test whether each algorithm can capture the global aspect with respect to diverse density estimates.

Next, to evaluate the local structure preservation of projections, we used the mean relative rank errors (MRREs [31]), trustworthiness, and continuity [64]. All of these local quality metrics estimate how well the nearest neighbors in one space (e.g., high- or low-dimensional space) are preserved in the other space. In out experiments, for ease of comparing local quality metrics at once, we defined MRREs $:= 1 -$ MRREs, so that a projection with higher MRREs has better retained the k-nearest neighbors like trustworthiness and continuity. The metrics require a hyperparameter k, the number of nearest neighbors. We set it as 5 for our experiment for the fair comparison [43]. Please refer to the supplementary materials for different values (k = 10, 15)
Baselines. We set the most widely used dimensionality reduction techniques as our baselines, including PCA, Isomap [59], t-SNE [36], UMAP [39], and At-SNE [14]. In the case of t-SNE, we leveraged Multicore t-SNE [60] for fast computation. To initialize a projection, we used PCA for t-SNE, following the recommendation in the previous work [34], and spectral embedding for UMAP which was the default. In addition, we compared with topologi-

cal autoencoders (TopoAE [43]) that were developed to capture the global properties of the data using a deep learning-based generative model. Following the convention of visualization in dimensionality reduction, we determined our result projected onto 2D space. We tuned the hyperparameters of each technique to minimize the $\mathrm{KL}_{0.1}$.

Hyperparameter Setting. We generated projections for each dimensionality reduction algorithm that had the lowest $\mathrm{KL}_{0.1}$ measure for the fair comparison to previous work [43]. To tune each algorithm's hyperparameters, we employed the grid search for t-SNE, UMAP, and At-SNE. For t-SNE and At-SNE, we changed the perplexity from 5 to 50 with an interval of 5, and the learning rate from 0.1 to 1.0 with a log-uniform scale. In the case of UMAP, we changed the number of nearest neighbors from 5 to 50 with an interval of 5, and the minimum distance between points in the projection from 0.1 to 1.0 with an interval of 0.1. We used the Python library scikit-optimize [19] to find the best hyperparameters for topological autoencoders. UMATO has several hyperparameters such as the number of hub points, the number of epochs, and the learning rate for global and local optimization. In our experiments, we configured everything except the number of hub points to the same setting for UMATO. We empirically decided to use 200 hub points for the Spheres dataset and 300 hubs for others. We used fewer hub points for the Spheres since it has only 10,000 data points in total, while the other datasets have 60,000 data points. We set the number of epochs to 100 for global optimization and to 50 for local optimization. Lastly, the global learning rate was set to 0.0065, and the local learning rate was set to 0.01.

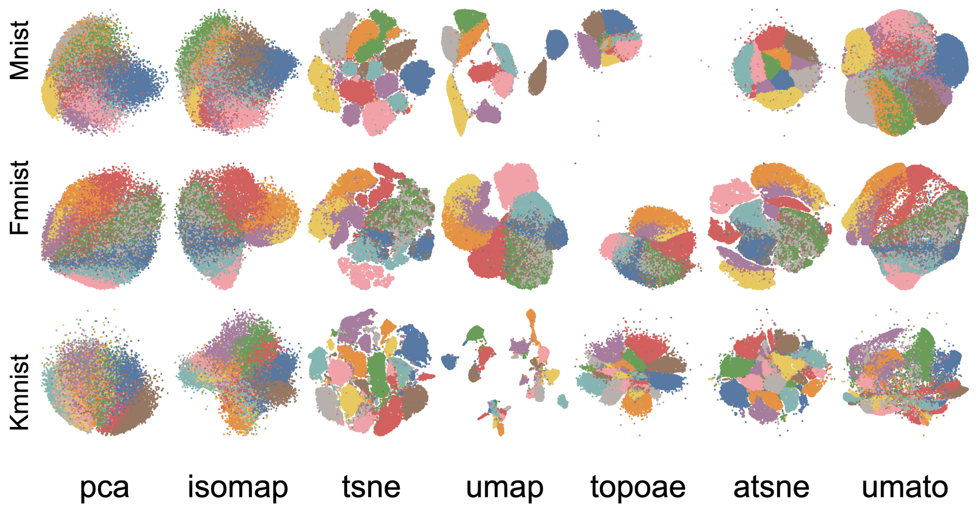

Figure 4.5: 2D projections produced by UMATO and six baseline algorithms on 3 real-world datasets. UMATO generated similar projections to PCA but with the points more locally connected; this is best viewed in color.

Quantitative Results

Table 4.3 displays the quantitative evaluation results. For the Spheres and Fashion MNIST dataset, UMATO was the only method that showed surpassing performance both in the global and local quality metrics. For local metrics, t-SNE, At-SNE, and UMAP generally had the upper-hand, but UMATO showed comparable performance in terms of MRRE$_X$ and continuity for Fashion MNIST and Kuzushiji MNIST datasets. Meanwhile, Isomap and topological autoencoders were good at global quality metrics although UMATO had the lowest (best) KL$_{0.1}$ and DTM except for the MNIST dataset. We noted that UMATO can show the best or comparable performance to Isomap in global quality metrics without computing the geodesic distance.

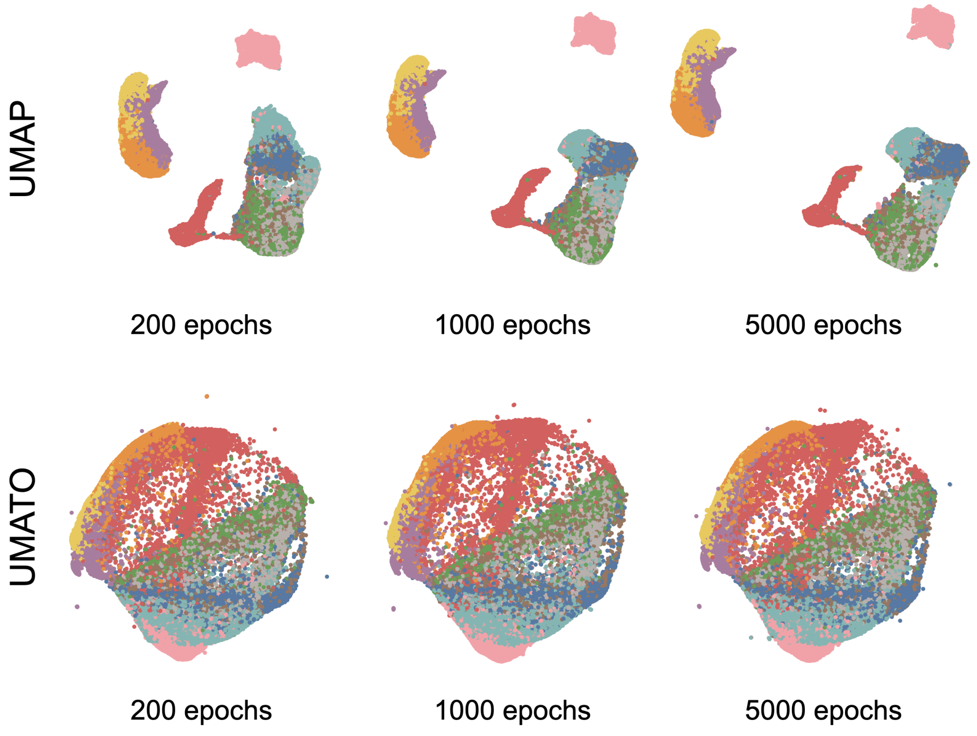

Figure 4.6: Comparing result of UMATO and UMAP with varying number of epochs (Top row) UMAP is susceptible to the number of epochs so that the clusters get dispersed as the epochs increases. (Bottom row) On the other hand, regardless of the number of epochs in the global optimization, UMATO results in almost the same embedding result.

Qualitative Results

Among the five algorithms, only UMATO could preserve both the global and local structures of the Spheres dataset. If we look at the Figure 4.1 made by UMATO, the outer sphere encircles the inner spheres in a circular form, which is the most intuitive to understand the relationship among different classes and the local linkage in detail. In the results from Isomap, t-SNE, UMAP, and At-SNE, the points representing the surrounding giant sphere mix with those representing the other small inner spheres, thus failing to capture the nested relationships among different classes. Meanwhile, topo-

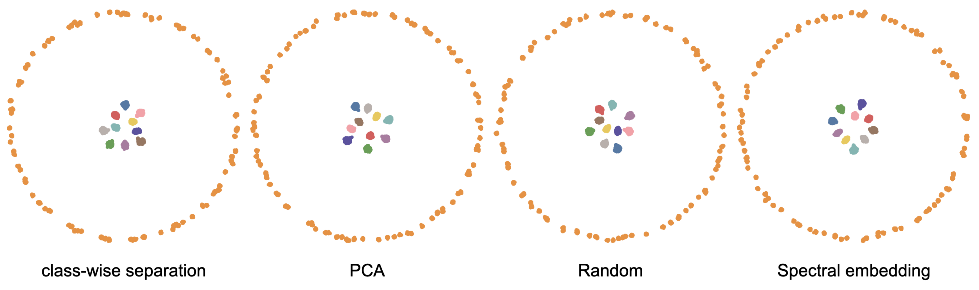

Figure 4.7: UMATO results on the Spheres dataset using different initialization methods. Although the average value of the normalized Procrustes distance of UMATO results is higher than the baselines because of the equidistant clusters of inner spheres, both global and local structures are well-captured with all different initialization methods. Best viewed in color.

logical autoencoders are able to realize the global relationship between classes in an incomplete manner; the points for the outer sphere are too spread out, thus losing the local characteristics of the class. From this result, we can acknowledge how UMATO can work with high-dimensional data effectively to reveal both global and local structures.

For the real-world datasets, UMATO showed a similar projection to PCA but with better captured local characteristics (Figure 4.5). The results from topological autoencoders showed some points detached far apart from their centers, even though the best hyperparameters were used for each. Although At-SNE claimed that it could capture both structures, the results were not significantly different from those of the original t-SNE algorithm when projecting the Spheres and Fashion MNIST datasets.

4.5.2 Case Study: UMATO on Real-world Biological Data

To test UMATO on a real-world biological dataset, we adopted a paired analytics methodology [2, 26] where a visual analytics expert and a subject matter expert work in pair to accomplish an analytical task using a visual-

Sample (%)	1	2	5	10	20
t-SNE	0.9835	0.9944	0.9544	0.9736	0.9824
UMAP	0.4002	0.3319	0.2341	0.1324	0.1327
At-SNE	0.8958	0.9510	0.7593	0.7980	0.9062
UMATO (ours)	**0.3153**	**0.1528**	**0.1206**	**0.0988**	**0.0520**
Sample (%)	30	50	60	80	100
t-SNE	0.9924	0.9959	0.9819	0.9944	0.9765
UMAP	0.1577	0.1109	**0.0713**	0.0951	0.0597
At-SNE	0.9376	0.9999	0.9460	0.9599	0.9999
UMATO (ours)	**0.1411**	**0.0526**	0.1732	**0.0529**	**0.0535**

Table 4.4: The normalized Procrustes distance between two projection results by the percentage of sub-samples. From four dimensionality reduction techniques, we measured the normalized Procrustes distance to check the projection stability using the Flow Cytometry dataset. The winner is in bold.

ization tool. We collaborated with an expert who has a Ph.D. in Bioinformatics with more than 5 years of research experience. We have run UMATO and the baseline algorithms (t-SNE, UMAP) on 23,822 single-cell transcriptomes from two areas at distant poles of the mouse neocortex [58]. Each cell belongs to one of 133 clusters defined by Jaccard–Louvain clustering (for more than 4,000 cells) or a combination of k-means and Ward's hierarchical clustering. Likewise, each cluster belongs to one of 4 classes: GABAergic (red/purple), Endothelial (brown), Glutamatergic (blue/green), Non-Neuronal (dark green).

The embedding result for each method is given in Figure 4.4. In the case of t-SNE, clusters are well-captured, but the classes are much dispersed, while UMAP adequately separates both classes and clusters. Compared to these baseline algorithms, UMATO is able to capture the relationship between classes much better, retaining some of the local manifolds as well. This proves that the hub points worked as the representatives that explain well about the overall dataset, and the algorithm focused more on the manifold at a higher level than the baselines.

Moreover, in biological data analysis, researchers often want to derive semantic meanings from the distance between samples in a low-dimensional embedding [17, 61]. However, UMAP embedding results are susceptible to the number of epochs that changing it could easily lead to exaggerated or reduced distance between clusters (Figure 4.6). As the proper number should be different for every dataset, the user cannot be sure of what value is appropriate. If wrong one is selected, this can induce a misinterpretation that the user considers the distance between clusters as something meaningful. The two-phase optimization of UMATO can solve the problem since the global optimization (first phase) is easy to converge as it runs only with a small portion of points. Therefore, the increasing number of epochs in the global optimization does not harm the final embedding. As the UMAP embedding results are susceptible to the number of epochs, this may cause a negative impact on the accurate interpretation of the results. On the other hand, as UMATO is robust over the number of epochs, we do not have to worry about such biases.

4.6 Discussion

UMATO Compared to t-SNE and UMAP. In case of t-SNE, as denoted in many researches [28, 34], the initialization method matters a lot to the final projection result. Moreover, because of the mathematical defect of leveraging KL divergence as its optimization function, the relative positions of clusters are often mixed up as it mostly focus on capturing the nearest neighbors. On the other hand, by adopting the cross-entropy loss function, UMAP ar-

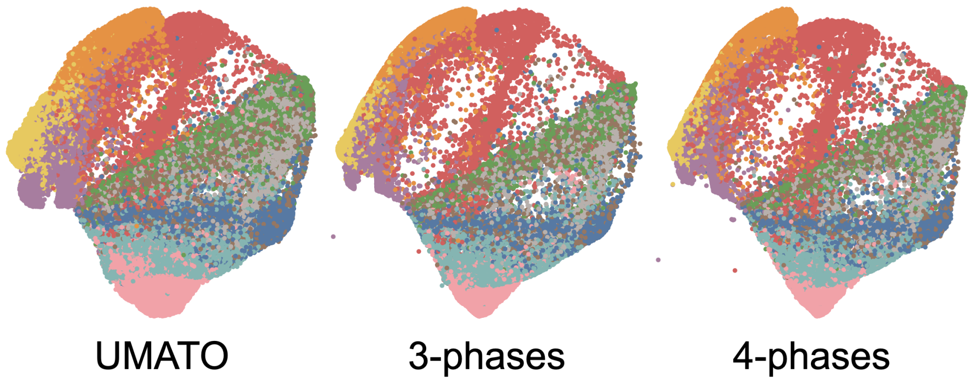

Figure 4.8: 2D projections of the Fashion MNIST dataset using UMATO and UMATO with multi-phase optimizations. Although there was a small difference such as the locations of outliers, we observed that the projection results were quite similar to each other.

gued that it can capture both the global and local properties of the high-dimensional data. However, the approximation techniques harm the original effect of loss function that they affect the algorithm to care more about the nearest neighbors which often results in projections with a biased representation of cluster structures. The projection seems to demonstrate a clear separation between clusters so that the result is good, but their positions are often disarranged that the high-level structures become biased in terms of the overall relationship between clusters.

As we have seen throughout the experiments, UMATO can be used to reveal the most obvious relationship between clusters or individual points of high-dimensional data. Since we only use the hub points to shape the skeleton of projection, we can capture the overall skeleton fast as it runs with small number of points. Compared to the two algorithms, UMATO is more stable, and robust over diverse initialization methods.

Projection Stability. Table 4.4 denotes the results of our experiment on the projection stability of UMATO and other dimensionality reduction techniques.

Algorithm	Spheres	MNIST	FMNIST	KMNIST
t-SNE	0.7878	0.8665	0.8284	0.8668
UMAP	**0.7726**	0.7767	0.7793	0.8213
UMATO (ours)	0.9504	**0.4808**	**0.0120**	**0.2037**

Table 4.5: The average value of normalized Procrustes distance between diverse dimensionality reduction techniques over four datasets. In all real-world datasets, UMATO has shown the most robust embedding results over different initialization methods. Although the UMATO results in the highest normalized Procrustes distance in the Spheres dataset, the embedding results look quite similar (Figure 4.7). The winner is in bold.

When the data size grows, we want to sample a portion of it to speed up the visualization. However, the concern is whether the projection that runs with the sampled indices is consistent with the corresponding part of the projection made with the full dataset. If the algorithm can generate stable and consistent results, the two projections should contain the least bias possible. To compute the projection stability of dimensionality reduction techniques, we used the normalized Procrustes distance to measure the distance between two comparable distributions. Specifically, given two datasets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, we define the Procrustes distance between the two distributions as

$$d_P(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y'_i)^2}. \tag{4.2}$$

We optimized $Y' = \{y'_1, y'_2, \ldots, y'_n\}$ using translation, uniform scaling, and rotation to minimize the squared error $\sum_{i=1}^{n} (x_i - y'_i)^2$. To replicate the experiment by Mcinnes et al. [39], we used the same Flow Cytometry dataset [6, 54], and ran optimal translation, uniform scaling, and rotation to minimize the Procrustes distance between the two distributions. As we can see in Table 4.4, UMATO outperformed t-SNE and At-SNE for all sub-sample sizes. Moreover, although UMAP is known as stable among existing algorithms, UMATO showed even better (lower) Procrustes distance except for one sub-sample size (60%). From this result, we can acknowledge that UMATO can generate more stable and consistent results regardless of sub-sample size than other dimensionality reduction techniques.

Projection Robustness over Diverse Initialization Methods. We tested the robustness of each dimensionality reduction technique with different initialization methods such as PCA, spectral embedding, random position, and class-wise separation. In class-wise separation, we initialized each class with a non-overlapping random position in 2-dimensional space, adding random Gaussian noise. In our results, UMATO embeddings were almost the same on the real-world datasets, while the UMAP and t-SNE results relied highly upon the initialization method. We report this in Table 4.5 with a quantitative comparison using the normalized Procrustes distance (Equation 4.2). In the case of the Spheres dataset, the clusters were equidistant from each other. The embedding results have to be different from run to run due to the limitation of the 2-dimensional space since there is no way to express this relationship (i.e., the crowding problem [36]). However, as we report in Figure 4.7, the global and local structures of the Spheres data can be manifested by UMATO regardless of different initialization methods.

Multi-phase Optimization. The optimization of UMATO can be easily expanded to multiple phases (e.g., three or more phases). Since we have a recursive procedure to expand the nearest neighbors, we can insert the optimization process each time we expand the neighbors to create a multi-phase algorithm. However, our experiment with three- and four-phase optimization using the Fashion MNIST dataset showed that there is no big difference between two-phase optimization and that with more than two phases.

We report the experimental result of multi-phase optimization (e.g., three and four-phase) using the Fashion MNIST dataset both quantitatively (Table 4.6) and qualitatively (Figure 4.8). As in Figure 4.8, there are no significant differences between the 2D projections, although some outliers were located in different places. In the quantitative results (Table 4.6), the original UMATO (with 2 Phases) was the winner in DTM, $KL_{0.1}$, KL_1, continuity, and $MRRE_Z$ but came last in other quality metrics. However, the difference between the original UMATO and the multi-phase optimizations was imperceptible. Therefore, we concluded that developing a multi-phase optimization for UMATO does not bring about any notable improvement in the projection result.

Applying UMATO in Progressive Visual Analytics Systems by Considering the Data Sequence. As the size of data at hand grows enormously these days, researchers have sought to minimize the interaction latency by progressively computing projections. However, most progressive systems achieve progressiveness by randomly splitting data into small batches[23, 27]. This can cause a severe problem in the final projection in case where the batch projected in the early phase are outliers, as they cumulatively affect the other points in the stochastic optimization process. UMATO can provide a remedy to the current PVA systems, as it can consider the sequence of batches following the importance of each point. By leveraging UMATO, the detection and visualization of outliers in a projection result will become much easier and explainable. This is reasonable as UMATO is a PVA-friendly algorithm since it can be easily expanded to multiple phases without losing the aforemen-

Dataset	Algorithm	DTM	$KL_{0.01}$	$KL_{0.1}$	KL_1
Fashion MNIST	2 Phases (UMATO)	**0.2035**	0.6852	**0.0342**	**0.0008**
	3 Phases	0.2058	0.6546	0.0343	**0.0008**
	4 Phases	0.2095	**0.6533**	0.0359	**0.0008**
	Algorithm	Cont	Trust	$MRRE_X$	$MRRE_Z$
	2 Phases (UMATO)	**0.9911**	0.9500	**0.9919**	0.9502
	3 Phases	0.9900	**0.9556**	0.9909	**0.9561**
	4 Phases	0.9895	0.9532	0.9904	0.9536

Table 4.6: Quantitative evaluation of UMATO and UMATO with multi-phase optimizations. Although the optimization process of UMATO can be simply expandable for multiple phases, no apparent distinctions are found in the the results with different numbers of optimization phases. The winner is in bold.

tioned advantages (e.g., stability, robustness over initialization methods) of the algorithm.

4.7 Summary

We present a two-phase dimensionality reduction algorithm called UMATO that can effectively preserve the global and local properties of high-dimensional data. In our experiments with diverse datasets, we have proven that UMATO can outperform previous widely used baselines (e.g., t-SNE and UMAP) both quantitatively and qualitatively.

Algorithm	Runtime (s)
Isomap	3 hours $>$
t-SNE	374.85 ± 11.38
UMAP	26.10 ± 3.97
UMATO (ours)	73.32 ± 8.39

Table 4.7: The runtime for each algorithm using MNIST dataset. UMAP and UMATO take much less time than MulticoreT-SNE [60] when tested on a Linux server with 40-core Intel Xeon Silver 4210 CPUs. The runtimes are averaged over 10 runs. Isomap [59] took more than 3 hours to get the embedding result.

Chapter 5

Discussion

5.1 Lessons Learned

It is very hard to explore a large scale of high-dimensional data with interaction. This is a practical issue in the visualization community, as the users can get distracted easily with such a long and boring waiting. We found that a progressive computation can be a good solution to this problem, by alleviating the heavy computation but to retain the characteristics of final result.

In UMAP, the different configurations often yield inconsistency over embedding results. This is a huge problem as it can mislead the users because the analysis of embedding can be incompatible or confusing from run to run. We identified that using the cross-entropy function without any approximation can mitigate the problem, but the computational cost increases enormously. Leveraging hub points is a detour, as it offers a competitive computation time, but less biases and more robustness.

5.2 Limitations

In the current implementation of Progressive UMAP, not all computation steps are bounded (e.g., the matrix multiplication) in time. For this reason, the algorithm might not be able to handle a dataset that is too large. In future work, we are to identify such problematic computational steps and speed them up using, for example, hardware acceleration.

For UMATO, we plan to provide an application of UMATO for the progressive computation of high-dimensional data to consider the importance or sequence of data points when projecting them. To use UMATO in PVA systems, it should support fast computation for the real-time analysis of the high-dimensional data. The running time of UMATO was faster than Multicore t-SNE [60], but about 3 times slower than UMAP implementation [38] (Table 4.7). To this end, we plan to accelerate UMATO, as in previous attempts with other dimensionality reduction techniques [46, 50], by implementing it on a heterogeneous system (e.g., GPU) for speedup.

Chapter 6

Conclusion

The thesis endeavored to support the thesis statement (The expansion of UMAP to progressive visual analytics and two-phase optimization enable an interactive exploration and produce less biased and robust embedding result.), with the results from two researches where each of them answering one of our research questions. As a summary, the contributions of the thesis are as follows: 1) Development of Progressive UMAP, which can embed and visualize high-dimensional data responsively onto a 2D space, with its quantitative and qualitative evaluation. 2) Design, development, and evaluation of UMATO, which enables to produce less biased and robust embedding over diverse initialization method. Through our researches, we shed a light to the further research of UMAP. We believe this will become more valuable asset in the future where the size of data grows.

Bibliography

[1] El-ad David Amir, Kara L Davis, Michelle D Tadmor, Erin F Simonds, Jacob H Levine, Sean C Bendall, Daniel K Shenfeld, Smita Krishnaswamy, Garry P Nolan, and Dana Pe'er. Visne enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nature biotechnology*, 31(6):545–552, 2013.

[2] Richard Arias-Hernandez, Linda T Kaastra, and Brian Fisher. Joint action theory and pair analytics: in-vivo studies of cognition and social interaction in collaborative visual analytics. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33 of number 33, 2011.

[3] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[4] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel WH Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W Newell. Dimensionality reduction for visualizing single-cell data using umap. *Nature biotechnology*, 37(1):38–44, 2019.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[6] Tess Brodie, Elena Brenna, and Federica Sallusto. Omip-018: chemokine receptor expression on human t helper cells. *Cytometry Part A*, 83(6):530–532, 2013.

[7] Frédéric Chazal, David Cohen-Steiner, and Quentin Mérigot. Geometric inference for probability measures. *Foundations of Computational Mathematics*, 11(6):733–751, 2011.

[8] Frédéric Chazal, Brittany Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, Alessandro Rinaldo, and Larry Wasserman. Robust topological inference: distance to a measure and kernel distance. *The Journal of Machine Learning Research*, 18(1):5845–5884, 2017.

[9] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. December 3, 2018. arXiv: `cs.CV/1812.01718 [cs.CV]`.

[10] James Cook, Ilya Sutskever, Andriy Mnih, and Geoffrey Hinton. Visualizing similarity data with a mixture of maps. In *Artificial Intelligence and Statistics*, pages 67–74, 2007.

[11] Mateus Espadoto, Rafael M Martins, Andreas Kerren, Nina ST Hirata, and Alexandru Cristian Telea. Towards a quantitative survey of dimension reduction techniques. *IEEE transactions on visualization and computer graphics*, 2019.

[12] Jean-Daniel Fekete, Danyel Fisher, Arnab Nandi, and Michael Sedlmair. Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports*, 8(10):1–40, 2019. Jean-Daniel Fekete, Danyel Fisher, Arnab Nandi, and Michael Sedlmair, editors.

[13] Jean-Daniel Fekete and Romain Primet. Progressive analytics: a computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[14] Cong Fu, Yonghui Zhang, Deng Cai, and Xiang Ren. Atsne: efficient and robust visualization on gpu through hierarchical optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 176–186, 2019.

[15] Takanori Fujiwara, Oh-Hyun Kwon, and Kwan-Liu Ma. Supporting analysis of dimensionality reduction results with contrastive learning. *IEEE transactions on visualization and computer graphics*, 26(1):45–55, 2019.

[16] Dian Gong, Xuemei Zhao, and Gerard Medioni. Robust multiple manifolds structure learning. In *Proceedings of the 29th International Conference on Machine Learning*, pages 25–32, 2012.

[17] Carmen Bravo González-Blas, Liesbeth Minnoye, Dafni Papasokrati, Sara Aibar, Gert Hulselmans, Valerie Christiaens, Kristofer Davie, Jasper Wouters, and Stein Aerts. Cistopic: cis-regulatory topic modeling on single-cell atac-seq data. *Nature methods*, 16(5):397–400, 2019.

[18] Antonio Gracia, Santiago González, Victor Robles, and Ernestina Menasalvas. A methodology to compare dimensionality reduction algorithms in terms of loss of quality. *Information Sciences*, 270:1–27, 2014.

[19] Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene-rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karl-

son Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. Scikit-optimize/scikit-optimize: v0.5.2, version v0.5.2, March 2018.

[20] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:857–864, 2002.

[21] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. S ummit: scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE transactions on visualization and computer graphics*, 26(1):1096–1106, 2019.

[22] Stephen Ingram and Tamara Munzner. Dimensionality reduction for documents with nearest neighbor queries. *Neurocomputing*, 150:557–569, 2015.

[23] Jaemin Jo, Jinwook Seo, and Jean-Daniel Fekete. Panene: a progressive algorithm for indexing and querying approximate k-nearest neighbors. *IEEE transactions on visualization and computer graphics*, 26(2):1347–1360, 2018.

[24] Johannes Kehrer and Helwig Hauser. Visualization and visual analysis of multifaceted scientific data: a survey. *IEEE transactions on visualization and computer graphics*, 19(3):495–513, 2012.

[25] Hannah Kim, Jaegul Choo, Changhyun Lee, Hanseung Lee, Chandan K Reddy, and Haesun Park. Pive: per-iteration visualization environment for real-time interactions with dimension reduction and clustering. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[26] Paul A Kirschner, Simon J Buckingham-Shum, and Chad S Carr. *Visualizing argumentation: Software tools for collaborative and educational sense-making*. Springer Science & Business Media, 2012.

[27] Hyung-Kwon Ko, Jaemin Jo, and Jinwook Seo. Progressive uniform manifold approximation and projection. In *22nd Eurographics Conference on Visualization, EuroVis 2020-Short Papers*, pages 133–137. Eurographics Association, 2020.

[28] Dmitry Kobak and George C Linderman. Initialization is critical for preserving global data structure in both t-sne and umap. *Nature Biotechnology*, 39(2):156–157, 2021.

[29] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[30] Yann LeCun and Corinna Cortes. MNIST handwritten digit database, 2010.

[31] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[32] John A Lee and Michel Verleysen. Quality assessment of dimensionality reduction: rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443, 2009.

[33] George C Linderman, Manas Rachh, Jeremy G Hoskins, Stefan Steinerberger, and Yuval Kluger. Efficient algorithms for t-distributed stochastic neighborhood embedding. *arXiv preprint arXiv:1712.09005*, 2017.

[34] George C Linderman, Manas Rachh, Jeremy G Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature methods*, 16(3):243–245, 2019.

[35] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: advances in the past decade. *IEEE transactions on visualization and computer graphics*, 23(3):1249–1268, 2016.

[36] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[37] Ahmed Mahfouz, Martijn van de Giessen, Laurens van der Maaten, Sjoerd Huisman, Marcel Reinders, Michael J Hawrylycz, and Boudewijn PF Lelieveldt. Visualizing the spatial gene expression organization in the brain through non-linear similarity embeddings. *Methods*, 73:79–89, 2015.

[38] Leland McInnes, John Healy, and James Melville. Umap. `https://github.com/lmcinnes/umap`, last accessed: 2020-02-20.

[39] Leland McInnes, John Healy, and James Melville. Umap: uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[40] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 2019.

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[42] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.

[43] Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders. In *Proceedings of the 37th International Conference on Machine*

Learning (ICML), Proceedings of Machine Learning Research. PMLR, 2020. arXiv: `1906.00722 [cs.LG]`. Forthcoming.

[44] Thomas Mühlbacher, Harald Piringer, Samuel Gratzl, Michael Sedlmair, and Marc Streit. Opening the black box: strategies for increased user involvement in existing algorithm implementations. *IEEE transactions on visualization and computer graphics*, 20(12):1643–1652, 2014.

[45] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[46] Corey J Nolet, Victor Lafargue, Edward Raff, Thejaswi Nanditale, Tim Oates, John Zedlewski, and Joshua Patterson. Bringing umap closer to the speed of light with gpu acceleration. *arXiv preprint arXiv:2008.00325*, 2020.

[47] Luis Gustavo Nonato and Michael Aupetit. Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2650–2673, 2018.

[48] Nicola Pezzotti, Thomas Höllt, B Lelieveldt, Elmar Eisemann, and Anna Vilanova. Hierarchical stochastic neighbor embedding. In *Computer Graphics Forum*, volume 35 of number 3, pages 21–30. Wiley Online Library, 2016.

[49] Nicola Pezzotti, Boudewijn PF Lelieveldt, Laurens van der Maaten, Thomas Höllt, Elmar Eisemann, and Anna Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE transactions on visualization and computer graphics*, 23(7):1739–1752, 2016.

[50] Nicola Pezzotti, Julian Thijssen, Alexander Mordvintsev, Thomas Höllt, Baldur Van Lew, Boudewijn PF Lelieveldt, Elmar Eisemann, and Anna Vilanova. Gpgpu linear complexity t-sne optimization. *IEEE transactions on visualization and computer graphics*, 26(1):1172–1181, 2019.

[51] Michael Sedlmair, Tamara Munzner, and Melanie Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE transactions on visualization and computer graphics*, 19(12):2634–2643, 2013.

[52] Ben Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[53] Vin D Silva and Joshua B Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in neural information processing systems*, pages 721–728, 2003.

[54] Josef Spidlen, Karin Breuer, Chad Rosenberg, Nikesh Kotecha, and Ryan R Brinkman. Flowrepository: a resource of annotated flow cytometry datasets

associated with peer-reviewed publications. *Cytometry Part A*, 81(9):727–731, 2012.

[55] Charles D Stolper, Adam Perer, and David Gotz. Progressive visual analytics: user-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.

[56] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web*, pages 287–297, 2016.

[57] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[58] Bosiljka Tasic, Zizhen Yao, Lucas T Graybuck, Kimberly A Smith, Thuc Nghi Nguyen, Darren Bertagnolli, Jeff Goldy, Emma Garren, Michael N Economo, Sarada Viswanathan, et al. Shared and distinct transcriptomic cell types across neocortical areas. *Nature*, 563(7729):72–78, 2018.

[59] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[60] Dmitry Ulyanov. Multicore-tsne. `https://github.com/DmitryUlyanov/Multicore-TSNE`, 2016.

[61] Koen Van den Berge, Hector Roux De Bezieux, Kelly Street, Wouter Saelens, Robrecht Cannoodt, Yvan Saeys, Sandrine Dudoit, and Lieven Clement. Trajectory-based differential expression analysis for single-cell sequencing data. *Nature communications*, 11(1):1–13, 2020.

[62] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[63] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[64] Jarkko Venna and Samuel Kaski. Neighborhood preservation in nonlinear projection methods: an experimental study. In *International Conference on Artificial Neural Networks*, pages 485–491. Springer, 2001.

[65] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. August 28, 2017. arXiv: `cs.LG/1708.07747 [cs.LG]`.

[66] Lin Yan, Yaodong Zhao, Paul Rosen, Carlos Scheidegger, and Bei Wang. Homology-preserving dimensionality reduction via manifold landmarking and tearing. In *Visualization in Data Science (VDS)*, 2018.

[67] Yuansheng Zhou and Tatyana O Sharpee. Using global t-sne to preserve inter-cluster data structure. *bioRxiv*, 2018. eprint: `https://www.biorxiv.org/content/early/2018/07/05/331611.full.pdf`.

국문 초록

고차원 데이터의 특성을 파악하는 효과적인 방법 중 하나는 저차원 공간에 임베딩을 하는 것이다. 많은 차원 축소 알고리즘이 있지만, 균일 매니폴드 근사 및 투영법 (UMAP)은 빠른 속도와 안정적인 투영 결과로 인해 많은 주목을 받았다. 그러나 현재의 UMAP은 실험용 데이터 셋인 MNIST에도 수 분이 걸리는 등, 인터랙티브 시각적 분석 시스템에 도입되기에는 너무 느리다. 또한 UMAP은 하이퍼파라미터 설정이 (특히, 초기화 방법과 epoch 수) 달라지는 것에 취약한데, 이것은 임베딩 결과로 부터 통찰을 얻는 과정에서 큰 오류를 범할 수 있게 한다.

UMAP의 즉각적인 반응성을 얻기 위해서, UMAP의 점진적인 알고리즘인 Progressive UMAP을 제안한다. 이로써 한 배치의 데이터를 추가할 때마다 임베딩 결과를 업데이트 하게되는 점진적인 계산이 가능해진다. 다음으로 적은 편향과 강건한 임베딩을 보장하기 위해 UMATO를 제안한다. 먼저 우리는 이러한 취약함이 최적화를 근사하는 과정에서 일어나는 것을 밝힌다. UMATO는, UMAP과 다르게, 두 단계에 걸친 최적화를 통해서 처음으로 전체적인 구조를 잡고, 그 다음 지역적 특성을 파악한다. 실험을 통해 UMATO가 PCA, *t*-SNE, UMAP, topological autoencoders 그리고 Anchor *t*-SNE와 같은 기존 알고리즘에 비해 전체 구조 평가 지표와 2차원 임베딩 결과에서 더 나음을 보인다. 추가적으로 여러 단계로 최적화 하는 것과 임베딩의 안정성 역시 실험으로 파악한다.

이 연구는 차원 축소뿐만 아니라 점진적 시각화 분야에도 독창적인 공헌을 한다. 마지막으로 연구의 향후 연구 방향을 도모한다.

주요어: 차원 축소, 매니폴드 학습, 위상적 데이터 분석, 시각화, UMAP

학번: 2019-24961