Review

Info Vis

Information Visualization
1–23
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1473871620909485
journals.sagepub.com/home/ivi
SAGE

Deep learning multidimensional projections

Mateus Espadoto[1], Nina Sumiko Tomita Hirata[1] and Alexandru C Telea[2]

Abstract
Dimensionality reduction methods, also known as projections, are often used to explore multidimensional data in machine learning, data science, and information visualization. However, several such methods, such as the well-known t-distributed stochastic neighbor embedding and its variants, are computationally expensive for large datasets, suffer from stability problems, and cannot directly handle out-of-sample data. We propose a learning approach to construct any such projections. We train a deep neural network based on sample set drawn from a given data universe, and their corresponding two-dimensional projections, compute with any user-chosen technique. Next, we use the network to infer projections of any dataset from the same universe. Our approach generates projections with similar characteristics as the learned ones, is computationally two to four orders of magnitude faster than existing projection methods, has no complex-to-set user parameters, handles out-of-sample data in a stable manner, and can be used to learn any projection technique. We demonstrate our proposal on several real-world high-dimensional datasets from machine learning.

Keywords
Dimensionality reduction, machine learning, multidimensional projections

Introduction

Exploring high-dimensional datasets is a key task in many application domains such as statistics, data science, machine learning, and information visualization. The main difficulty encountered by this task is a large size of such datasets, seen both in the number of observations (also called samples) and in the number of measurements recorded per observation (also called dimensions, features, or variables). As such, high-dimensional data visualization has become an important separate field in information visualization (infovis).[1–3]

Several techniques have been proposed for high-dimensional data visualization, including glyphs, parallel coordinate plots, table lenses, scatterplot matrices, dimensionality reduction (DR) methods, and multiple views linking the above visualization types. In this family of techniques, DR methods, also called projections, have a particular place: compared to all other techniques, they scale much better in both the number of samples and the number of dimensions they can accommodate (show) on a given screen space area. As such, projections have become the tool of choice for exploring data which has a high number of dimensions (tens up to hundreds) and/or in applications where the individual identity of dimensions is less important, as frequently met in data science and machine learning

[1]Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil
[2]University of Groningen, Groningen, The Netherlands

Corresponding author:
Mateus Espadoto, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matao, 1010, São Paulo 05508-090, Brazil.
Email: mespadot@ime.usp.br

applications. In the last decade, many projection techniques have been proposed[3–5] Among these, t-distributed stochastic neighbor embedding (t-SNE)[6] is arguably one of the best known and most adopted in applications, given that it creates projections with good visual segregation of similar sample clusters. Yet, t-SNE comes with some downsides: it is very slow for large datasets (thousands of observations or more), due to its quadratic nature; its parameters can be tricky to get right, which can lead to unpredictable results;[7] and it lacks the capability of projecting out-of-sample data, which is useful when comparing several (time-dependent) datasets.[3,8,9]

Work has been performed to address the performance issue, such as tree-accelerated stochastic neighbor embedding (SNE),[10]H-SNE,[11] A-SNE,[12] and uniform manifold approximation and projection (UMAP),[13] which is a completely different algorithm but with the stated goal of having t-SNE quality at a higher speed. However, in general, there is no technique in the t-SNE class that jointly addresses scalability, stability, and out-of-sample handling. Moreover, t-SNE refinements are algorithmically not simple to understand and/or implement, which may limit their attractiveness. Such limitations are, to a large extent, shared by many other projection techniques;[3] therefore, our discussion next should be seen in this larger context rather than focusing on t-SNE class methods only. A way to handle these limitations *jointly* and *independently* on the projection technique of choice is of considerable interest.

To address the above goal, we propose a learning-based approach to DR: We take any projection technique (deemed suitable for an application at hand), run it on a small subset of the available data, train a deep neural network to learn the mapping from high- to low-dimensional space produced by the respective projection, and use the trained network to project the entire dataset or similar datasets. Our method has the following contributions:

Quality (C1). We provide similar results to the learned projection, as measured by several well-known metrics in DR literature; briefly put, such metrics quantify how close the two-dimensional (2D) projection reflects the structure (inter-point distances and nearest neighbors) of the high-dimensional dataset.
Scalability (C2). We compute the projection in linear time in the number of dimensions *and* observations. Practically, we project datasets of up to a million observations and hundreds of dimensions in a few seconds using commodity hardware, no matter which projection technique we are emulating.
Ease of use (C3). Our method works without the need to set any complex parameters, except for the number of training epochs, which is minimized by the use of early stopping (see section “Training effort”). Our method is implemented using only open-source infrastructure, so it is easily replicable.
Genericity (C4). We can handle any kind of high-dimensional data that can be represented as high-dimensional vectors and can mimic the behavior of different types of projection techniques.
Stability and out-of-sample support (C5). Our method allows one to project new observations for a learned projection without recomputing it, as is needed with standard t-SNE and any other non-parametric methods.

This article is structured as follows. Section “Related work” discusses related work on multidimensional projections. Section “Method” details our method. Section “Results” presents our results that support our contributions outlined above. Section “Discussion” discusses our proposal. Section “Conclusion” concludes the paper.

Related work

We first introduce a few notations. Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leqslant i \leqslant n$ be a n-dimensional (nD) real-valued observation or sample, and let $D = \{\mathbf{x}_i\}$, $1 \leqslant i \leqslant N$ be a dataset of N samples. Let $\mathbf{x}^j = (x^j_1, \ldots, x^j_N)$, $1 \leqslant j \leqslant n$ be the jth sample of D. Thus, D can be seen as a table with N rows (samples) and n columns (features or dimensions). A projection technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q \tag{1}$$

where $q \ll n$, and typically $q = 2$. The projection $P(\mathbf{x})$ of a sample $\mathbf{x} \in D$ is a 2D point. Projecting a set D yields thus a 2D scatterplot, which we denote next as $P(D)$.

DR

Over the years, tens of DR methods have been developed. These propose quite different trade-offs between the desirable features listed in section “Introduction,” as follows. For more extensive reviews of DR methods, and their quality features, we refer to the literature.[2,4,5,14–17]

Probably, the best known DR method is principal component analysis[18] (PCA), which has been used in several areas for many decades. It is a very simple algorithm with theoretical grounding in linear algebra. PCA is commonly used as pre-processing step for automatic DR on high-dimensional datasets prior to selecting a more specific DR method for visual exploration.[3] PCA scores high on scalability (C2), ease of use (C3), predictability, and out-of-sample capability (C5). However, due to its linear and global nature, PCA lacks quality (C1), especially for data of high intrinsic dimensionality, which is less than ideal for data visualization purposes.

Methods based on manifold learning, such as multidimensional scaling (MDS),[19] Isomap,[20] and locally linear embedding (LLE)[21] with its variations,[22–24] try to reproduce in 2D the high-dimensional manifold on which data are embedded and are designed to capture nonlinear structure in the data. These methods are commonly used in visualization since they generally yield better results than PCA (or similar global/linear methods) in terms of quality (medium-high C1). Unfortunately, those methods are harder to tune (low C3), do not have out-of-sample capability (C5), do not work well for data which is not spread over a manifold (C4), and generally scale poorly for large datasets (low C2).

Methods based on force-directed techniques, also known as spring embedders, such as local affine multidimensional projection (LAMP)[25] and least squares projection (LSP),[26] are popular in the visualization literature and have a long history, with uses other than DR, such as graph drawing. Such methods can have a reasonably good visual quality (medium C1), good scalability (medium-high C2) and are simple to use (high C3). However, most of them lack out-of-sample capability (C5).

In the mid-2000s, the stochastic neighborhood embedding (SNE) family of methods appeared, of which t-SNE[6] is arguably the most popular member. A key praised feature of t-SNE is the ability to visually segregate similar samples in D (C1). Despite its high scoring on the quality criterion (high C1), t-SNE can be very slow (low C2), since it has a complexity of $O(N^2)$ in sample count, is very sensitive to small changes in the data (low C5), can be very hard to tune (low C3)[7] in order to get good visualizations, and does not have out-of-sample capability. There are attempts to improve t-SNE's performance, such as tree-accelerated t-SNE,[10] hierarchical SNE,[11] and approximated t-SNE.[12] However, these methods require quite complex algorithms and still largely suffer from the aforementioned sensitivity, tuning, and out-of-sample problems. As of 2018, uniform manifold approximation and projection (UMAP)[13] appeared, advertised as a method which can generate projections with comparable quality to t-SNE (high C1) but much faster (high C2), and with out-of-sample capability. Despite its advantages, UMAP shares some disadvantages with t-SNE, namely, its sensitivity to small data changes (low C5) and parameter tuning difficulty (low C3).

Deep learning

Neural network approaches have been proposed for DR, such as autoencoders,[27,28] which aim to generate a compressed, low-dimensional representation on their bottleneck layers by training the network to reproduce its inputs on its outputs. Typically, autoencoders produce results comparable to PCA on the quality criterion (low C1). Also, for different types of datasets, one typically needs to design different autoencoder architectures, which is time-consuming (low C3). Yet, autoencoders are easily parallelizable (high C2), predictable, and have out-of-sample capability (C5).

The ReNDA algorithm[29] is a very recent neural-based approach that uses two networks, improving on earlier work from the same authors. One network is used to implement a nonlinear generalization of Fisher's linear discriminant analysis, using a method called GerDA; the other network is an Autoencoder used as a regularizer. According to the results of the original paper, the method scores well on predictability and has out-of-sample capability (C5). However, it requires labeled data, which none of the other algorithms discussed in this study do. Also, the authors do not present results that show how scalable the method is (unknown C2).

Parametric t-SNE (pt-SNE)[9] was proposed to address some of the limitations outlined in section "Introduction." pt-SNE uses a deep learning architecture consisting of restricted Boltzmann machines (RBMs)[30] to pre-train a neural network to reduce dimensionality, followed by a fine-tuning stage that refines this network to minimize a cost function based on t-SNE's Kullback–Leibler (KL) divergence. The key advantage of pt-SNE is its parametric nature (mapping the entire nD input space to the lower-dimensional qD space), which allows out-of-sample behavior by construction. Only few DR methods in existence are parametric and thus have this ability (e.g. PCA,[18] neighborhood component analysis (NCA),[31] autoencoders[27]). From these methods, PCA and NCA do not work well when the intrinsic dimensionality of the input data is higher than the one of the output space, due to their linear nature. More interestingly, a detailed discussion on why pt-SNE is superior to autoencoders in terms of quality (criterion C1) is provided: pt-SNE aims to capture the *local* data structure in nD, using Gaussian distributions, and to transfer this structure to the low-dimensional qD space using the KL divergence cost. In contrast, the cost function of autoencoders aims to maximize data *variance* in qD. This does not create well-separated clusters in qD, even when these exist in nD, as this would decrease variance and thus increase the reconstruction error. Similar to pt-SNE, our method is also parametric

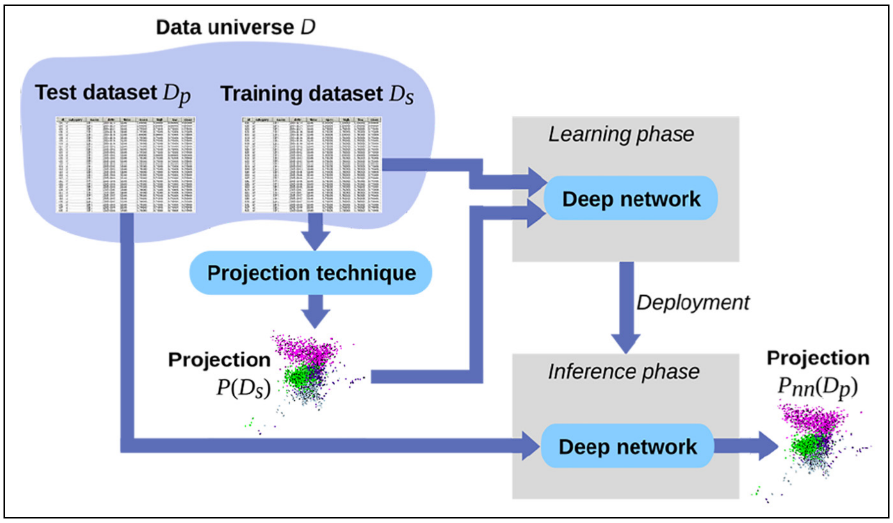

Figure 1. Pipeline for learning projections (see section "Method").

(thus satisfying C5), uses a deep learning approach, and has the same advantages versus autoencoders. In contrast to pt-SNE, however, we have a much simpler neural network architecture, and thus a simpler training process, and a single hyperparameter (number of training epochs or, alternatively, training loss); we use *supervised* learning (with a given 2D scatterplot produced by a user-chosen projection method as ground truth), and thus have a completely different cost function (distance to 2D ground-truth projection rather than KL divergence); and, finally, our method can learn *any* projection technique, not just t-SNE.

Method

Our proposal is very simple: consider a data *universe* $\mathcal{D}$, that is, the union of all datasets created by a given application area, for example, all fashion images, all handwritten digit images, or all astronomical images related to a certain type of measurement. If we admit that there exists some specific structure of the data in such a universe, that is, the data samples are not uniformly distributed along all dimensions, then a *good* projection should capture well this data structure (which is, for example, reflected in terms of segregating different data clusters in the visual space). We hypothesize that the way in which a given projection technique P captures this data structure can be *learned* using a limited number of small training datasets $D \subset \mathcal{D}$ and their respective projections $P(D) \subset \mathbb{R}^2$.

Our proposal follows precisely this: let D_s be a randomly selected subset of one or several datasets $D \subset \mathcal{D}$, and let $P(D_s)$ be the corresponding projection of D_s. Let P_{nn} be a neural network trained on D_s aiming to mimic the behavior of $P(D_s)$. Let $D_p = D \backslash D_s$ be the remaining data in D to be projected by P_{nn}.

Figure 1 presents our idea, which consists of three main steps—creating the training projection, training, and inference. To create the training projection, we project D_s using any user-chosen projection technique P. We next use the projected subset $P(D_s)$ alongside the original high-dimensional D_s to train a feed-forward, fully connected neural network P_{nn} to learn how to project high-dimensional data. Once P_{nn} is trained, we use it to project the remaining points D_p of D. By extension, we use P_{nn} to also project different datasets from the same universe $\mathcal{D}$ as D.

After empirical testing, varying the number of layers and the number of units in each layer, we defined the architecture for P_{nn} as having three fully connected hidden layers, with 256, 512, and 256 units, respectively, using ReLU activation functions, followed by a two-element layer which uses the sigmoid activation function to encode the 2D projection, scaled to the interval $[0, 1]^2$ for implementation simplicity (Figure 2). The number of units in the input layer matches the dimensionality of the input data (n in equation (1)). We stress that this particular network architecture has nothing special about it. Other similar architectures may work as well. The central goal of this article is to propose a novel way to achieve DR by learning projections. Whether other, more specific, network architectures can provide better projections is a question for future research.

We initialize weights with the He et al.[32] uniform variance scaling initializer, and bias elements using a constant value of 0.0001, which showed good results during testing. We use the Adam[33] optimizer, a variant of the well-known stochastic gradient descent, to train P_{nn} for at least 10 epochs and up to 200 epochs on an

Table 1. Quality metrics.

Metric	Definition	Range
Trustworthiness (T)	$1-\frac{2}{NK(2n-3K-1)}\sum_{i=1}^{N}\sum_{j\in U_i^{(K)}}(r(i,j)-K)$	$[0,\mathbf{1}]$
Continuity (C)	$1-\frac{2}{NK(2n-3K-1)}\sum_{i=1}^{N}\sum_{j\in V_i^{(K)}}(\hat{r}(i,j)-K)$	$[0,\mathbf{1}]$
Neighborhood hit (NH)	$\frac{1}{N}\sum_{\mathbf{y}\in P(D)}\frac{\mathbf{y}_k^l}{\mathbf{y}_k}$	$[0,\mathbf{1}]$
Shepard diagram correlation (R)	Spearman rank correlation of Scatterplot $(\parallel \mathbf{x}_i-\mathbf{x}_j \parallel, \parallel P(\mathbf{x}_i)-P(\mathbf{x}_j) \parallel), 1\leqslant i\leqslant N, i\neq j$	$[0,\mathbf{1}]$
Kullback–Leibler divergence	$\sum \mathcal{P}(D) log\left(\frac{\mathcal{P}(D)}{\mathcal{Q}(P(D))}\right)$, where $\mathcal{P}$ and $\mathcal{Q}$ denote probability distributions	Unbounded
Cross-entropy	$\sum \mathcal{P}(D) log\, \mathcal{Q}(P(D))$, where $\mathcal{P}$ and $\mathcal{Q}$ denote probability distributions	Unbounded
Normalized stress	$\frac{\sum_{ij}(\mathbf{D}_{ij}-\mathbf{d}_{ij})}{\sum_{ij}\mathbf{D}_{ij}^2}$	Unbounded
LLE reconstruction error	$\sum_i \parallel P(D)_i-\sum_j W_{ij}P(D)_j\parallel^2$	Unbounded
Isomap cost function	$\frac{\parallel K(\mathbf{D})-K(\mathbf{d})\parallel}{n}$, where $K(D)=-0.5\left(I-\frac{1}{n}\right)D^2\left(I-\frac{1}{n}\right)$ and n is the number of observations	Unbounded

Right column gives the metric ranges; optimal value is marked in bold, and for unbounded metrics, smaller values are always better.

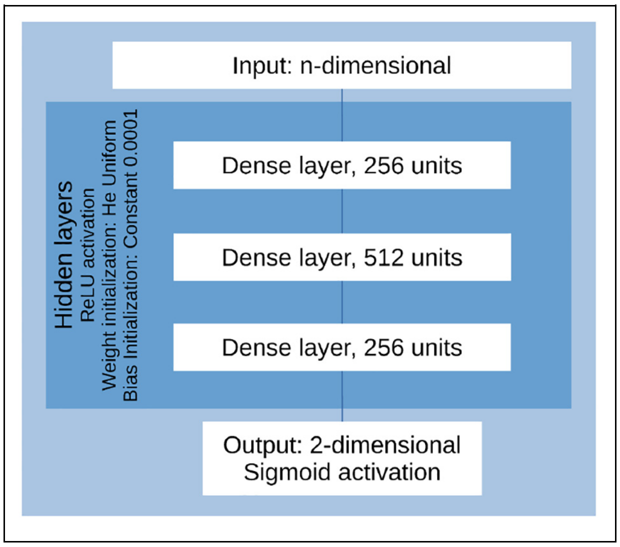

Figure 2. Feed-forward network architecture used.

"early stopping" setup. That is, training automatically stops on convergence, defined as the epoch where the validation loss stops decreasing. In practice, not more than 60 epochs are needed to achieve convergence, the average being 30 epochs (see section "Training effort"). The cost function used is mean squared error (MSE, equation (2))

$$MSE=\frac{1}{N}\sum_{i=1}^{N}\parallel \mathbf{y}_i-\hat{\mathbf{y}}_i\parallel^2 \qquad (2)$$

where $\mathbf{y}_i$ are the ground truth 2D coordinates provided by the training projection and $\hat{\mathbf{y}}_i$ are the 2D coordinates predicted by the network, respectively. MSE showed higher convergence speed during testing than other common cost functions such as mean absolute error and log hyperbolic cosine (logcosh).

We *test* P_{nn} by comparing the projections it delivers on D_p (unseen data during training) with the ground truth $P(D_p)$ obtained by running the projection P we desire to mimic on D_p. For this, we use two classes of metrics, as follows. *General metrics* capture desirable properties of a projection $P(D)$ in a technique-agnostic manner, that is, without considering the specific objective (cost) function that P tries to optimize. Such metrics are often used in projection literature to compare different techniques that do not share implementation similarities.[3,34] *Technique-specific metrics* consider the cost functions used by specific techniques, and thus allow comparing these techniques to our method, that is, show how well our method manages to learn the "style" of a projection by exposing how well it optimizes the underlying cost function. Table 1 shows the definitions of all these metrics, which are described below. For technique-specific metrics, we list next the names of the projection techniques to which a metric applies in brackets after the metric name.

Trustworthiness. T measures the proportion of points in D that are also close in $P(D)$. T tells how much one can trust that local patterns in a projection, for example, clusters, represent actual patterns in the data.[35] In the definition (Table 1), $U_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in the 2D space but not among the K nearest neighbors of point i in $\mathbb{R}^n$, and $r(i,j)$ is the rank of the 2D point j in the ordered set of nearest neighbors of i in 2D. We chose $K = 7$ for this study, in line with the study by van der Maaten and Postma[4] and Martins et al.[36]

Continuity. C measures the proportion of points in $P(D)$ that are also close together in D.[35] In the definition (Table 1), $V_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in $\mathbb{R}^n$ but not among the K nearest neighbors in 2D, and $\hat{r}(i,j)$ is the rank of the $\mathbb{R}^n$ point j in the ordered set of nearest neighbors of i in $\mathbb{R}^n$. As for M_t, we chose $K = 7$.

Neighborhood hit. NH measures how well-separable labeled data are in a projection $P(D)$, in a rotation-invariant fashion, from perfect separation ($NH = 1$) to no separation ($NH = 0$).[26] NH is defined as the number $\mathbf{y}_k^l$ of the k nearest neighbors of a point $\mathbf{y} \in P(D)$, denoted by $\mathbf{y}_k$, that have the same label as $\mathbf{y}$, averaged over $P(D)$. In this article, we used $k = 7$.

Shepard diagram correlation R. The Shepard diagram is a scatterplot of the pairwise (Euclidean) distances between all points in $P(D)$ versus the corresponding distances in D.[25] The closer the plot is to the main diagonal, the better overall distance preservation is. Plot areas below, respectively above, the diagonal indicate distance *ranges* for which false neighbors, respectively missing neighbors, occur. We quantitatively assess a Shepard diagram by computing its Spearman rank correlation R. A value of $R = 1$ indicates a perfect (positive) correlation of distances.

KL divergence (t-SNE). Measures the difference between two probability distributions created from D and $P(D)$, which can be viewed as the relative entropy between two distributions.[6]

Cross-entropy (UMAP, autoencoder). Similar to the KL divergence, cross-entropy also measures the difference between two probability distributions created from D and $P(D)$, but as the total entropy between two distributions.[13]

Normalized stress (MDS, LAMP, LSP). Measures the difference between the distance matrices $\mathbf{D}$ and $\mathbf{d}$ of points in D and $P(D)$, respectively.[25]

MSE (PCA). Measures the MSE between D and the inverse of the transformation computed by PCA, that is, $P^{-1}(P(D))$, as defined in equation (2).

Reconstruction error (LLE). Measures the reconstruction error of $P(D)$ based on LLE's computed weights and cost function.[21]

Isomap cost function (Isomap). Measures the Isomap cost function based on the distance matrices D and d of points in D and $P(D)$, respectively.[20] Although we use class labels for computing NH, note that we do not use class labels *anywhere* else during training or computing the ground-truth projection.

Results

We next show how our proposal covers the requirements listed in section "Introduction." For this, we structure our evaluation into several tasks. We compare our results with those produced by several well-known projection techniques (t-SNE, UMAP, PCA, Isomap, MDS, LAMP, LSP, pt-SNE, and LLE). We use a range of publicly available real-world benchmark datasets that have many observations and dimensions, exhibit a non-trivial data structure, and come from different application domains, as follows (Figure 3 shows samples from these datasets).

MNIST. 70K observations of handwritten digits from 0 to 9, rendered as 28×28-pixel grayscale images, flattened to 784-element vectors.[37]

Fashion MNIST. 70K observations of 10 types of pieces of clothing, rendered as 28×28-pixel grayscale images, flattened to 784-element vectors.[38]

Dogs versus Cats. 25K images of varying sizes divided into two classes (Cats, Dogs).[39] We used the Inception V3[40] convolutional neural network (CNN) pre-trained on the ImageNet dataset[41] to extract features of those images, yielding 2048-element vectors for each image.

IMDB Movie Review. 25K movie reviews[42] from which 500 features were extracted using term-frequency and inverse document frequency (TF-IDF),[43] a standard method in text processing.

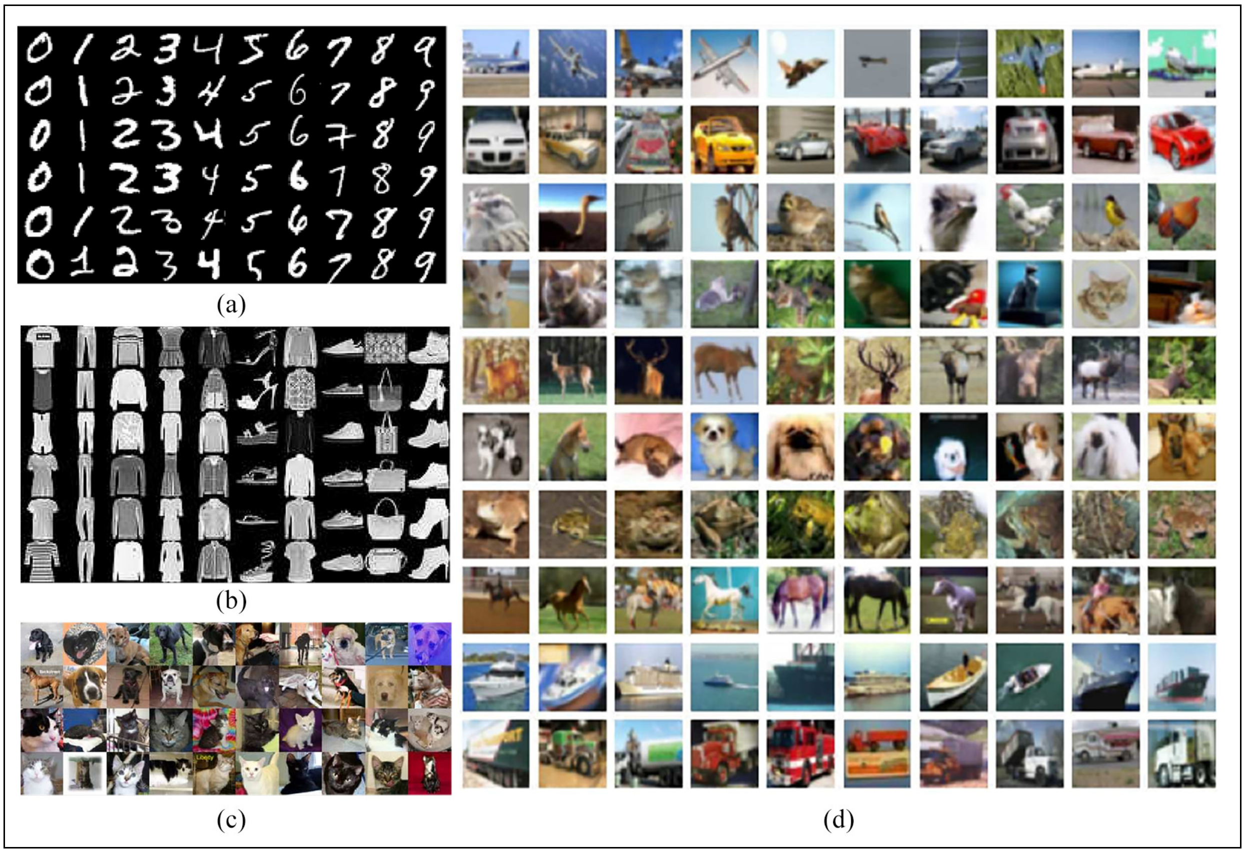

Figure 3. Illustrative examples from the used datasets. (a) MNIST (70K samples, 10 classes), (b) FashionMNIST (70K samples, 10 classes), (c) Cats and Dogs (25 K images, 2 classes), and (d) CIFAR-10 (60K samples, 10 classes).

Wisconsin breast cancer—diagnostic. 569 observations of a fine needle aspirate (FNA) of a breast mass, described with 32 dimensions.[44]

Human activity recognition. 10,299 observations from 30 subjects performing activities of daily living used for human activity recognition (HAR), described with 561 dimensions.[45]

Spambase. 4601 observations of email classified as spam or not spam, with 57 dimensions.[46]

Seismic bumps. 2584 observations and 24 dimensions, used to forecast seismic bumps in a coal mine.[47]

CIFAR-10 and CIFAR-100. 60K 32×32-pixel color images in 10 and 100 classes, respectively.[48] We used the DenseNet[49] CNN pre-trained on the ImageNet dataset to extract features of those images, yielding 1920-element vectors for each image. For each dataset, the split between training and test sets varies for each experiment and is explained in detail next in each task-specific section.

Training effort

It is important to assess what our method needs (training-data-wise and training-effort-wise) to reach the quality of the training projection, or close to that. Figure 4 shows t-SNE, UMAP, MDS, and LAMP projections of subsets of the MNIST dataset with 2 and 10 classes, respectively, alongside our method's results. We used training sets D_s of varying sizes, all randomly and independently sampled from the MNIST dataset. We included the two-class selection (digits 0 and 1) since we know that images for these digits are quite different. Hence, the obtained projections should clearly visually separate samples from these two classes. For the two-class case, we see that our method yields practically the same results as the ground truth methods (t-SNE, UMAP, LAMP, and MDS, respectively), already when using only 1K training samples. For the 10-class case, we obtain very similar results starting from roughly 5K training samples.

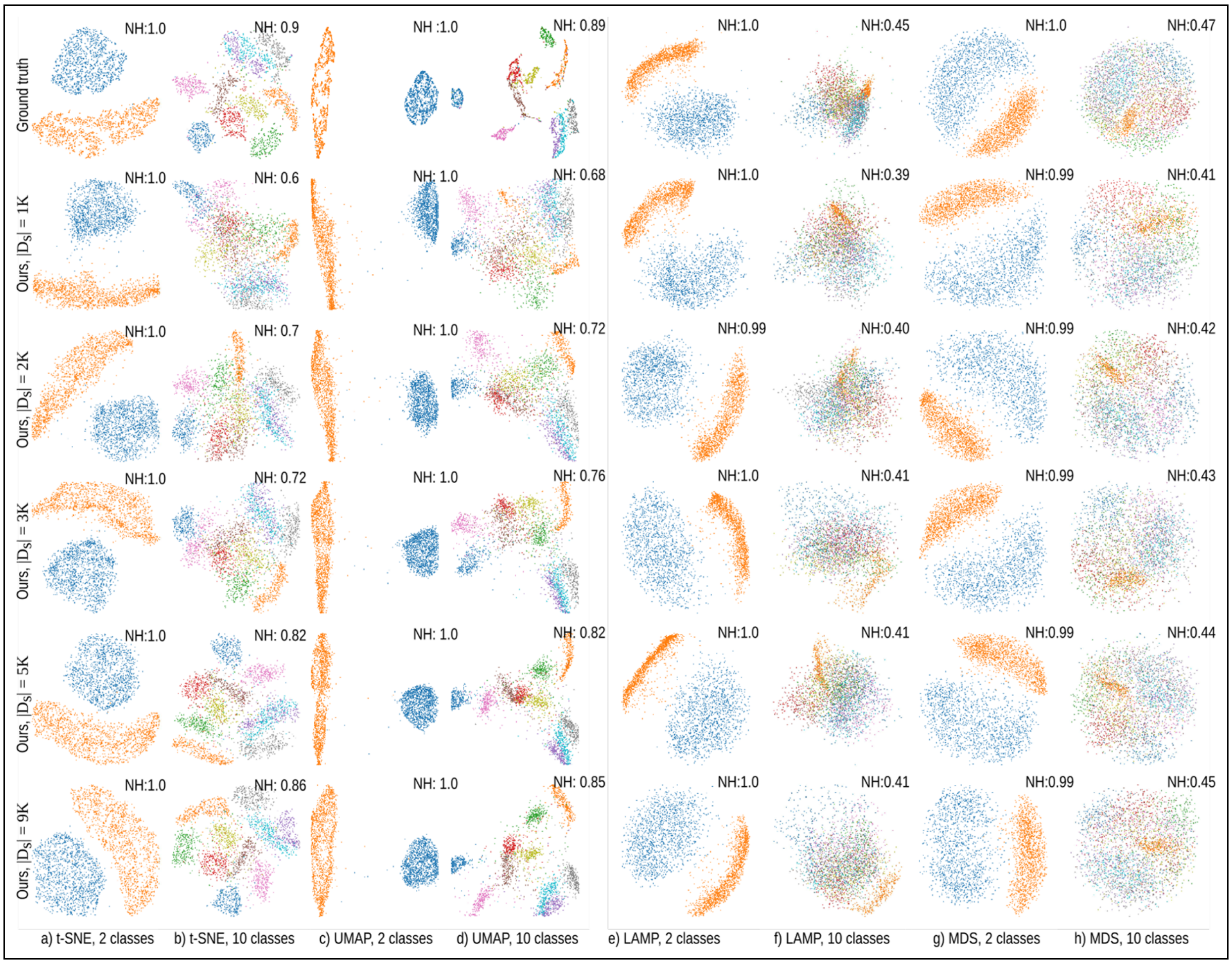

Figure 4. Top row: MNIST dataset, 10K sample projections of 2 and 10 classes created by t-SNE, UMAP, LAMP, and MDS. Next rows: projections done by our method for varying training set sizes $|D_s|$.

Figure 4 also shows how our learned projections get close to the neighborhood hit value (*NH*) of the ground-truth projection as we increase the training-set size. It is important to note that our method *cannot* formally exceed this ground-truth value, as we learn to mimic this ground truth, not surpass it. Hence, when learning from good-quality projections, we will achieve high quality; when learning from less good projections, we will not surpass that quality. Section "Capturing the structure of different datasets" revisits this point with additional examples.

Figure 5 also shows how the quality improves for a fixed training set (3K samples) as we increase the number of training epochs. As visible, we obtain projections already very close to the ground truth from roughly 25.50 epochs for the more complex 10-class MNIST dataset, and from roughly 10 epochs for the simpler 2-class MNIST dataset, respectively.

Figure 6 provides more insight into the training process by showing how the loss (cost) decreases *during training* as we increase the number of epochs (blue and green curves for t-SNE and UMAP, respectively), for both the MNIST and FashionMNIST datasets, when considering only 2 classes (easier problem) or all 10 classes (harder problem). The orange and red curves (for t-SNE and UMAP, respectively) show what the loss is for the *validation* set for the network trained for a given number of epochs. Note that we did not include curves for LAMP and MDS in Figure 6 to avoid overplotting, since these curves are practically identical to the ones already shown for t-SNE and UMAP. As visible, all curves converge quite quickly and similarly for all datasets, all projections. Of course, the validation loss is a bit larger than the training loss. Separately, we see that convergence is rapid for all four considered datasets. We can use these curves in practice to find how many training epochs we need for a desired maximal loss. Conversely, we can fix a preset maximal loss (in practice, 0.005) and compute the number of training iterations required

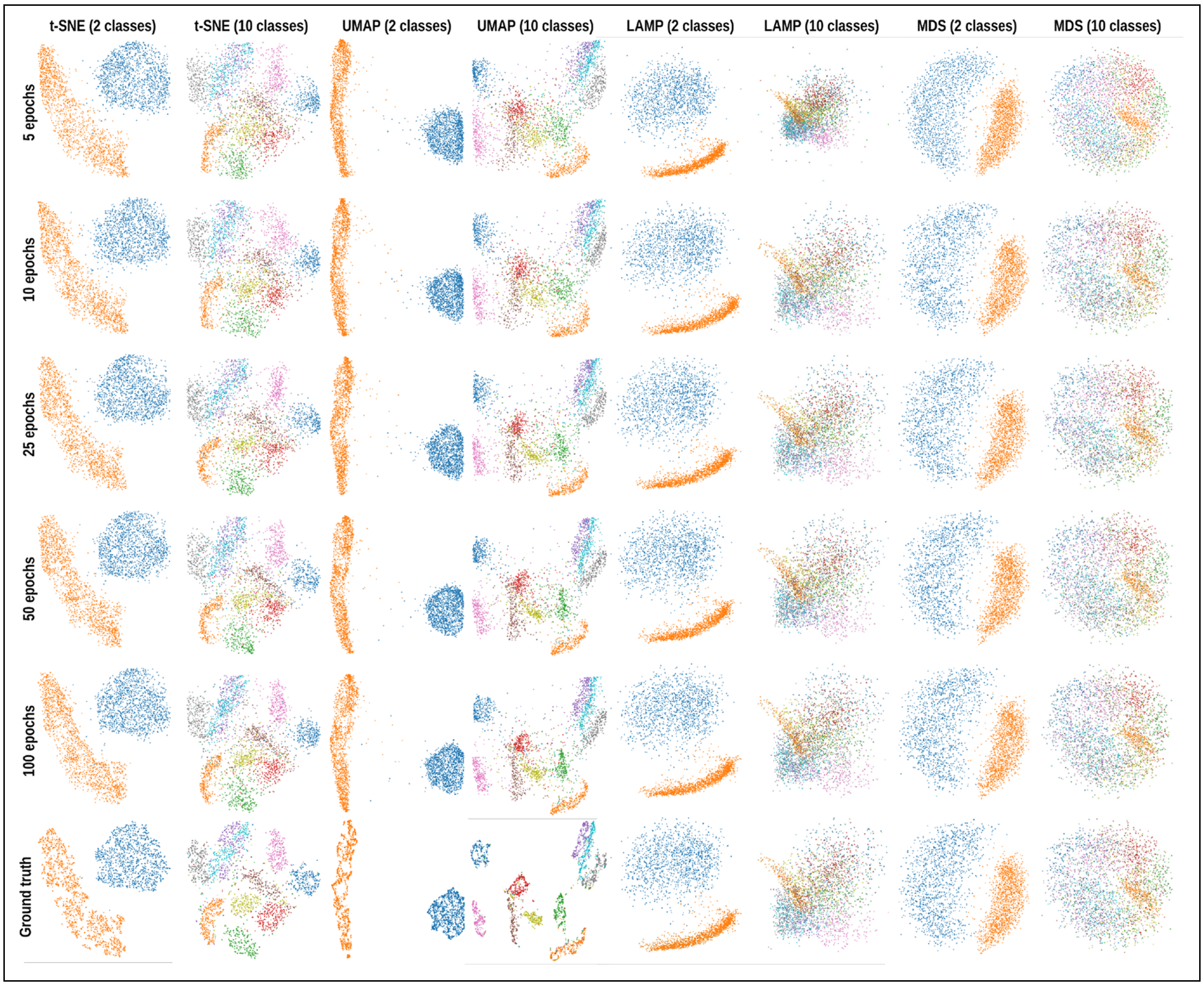

Figure 5. Ground truth: MNIST dataset, 3K sample projections of 2 and 10 classes created by t-SNE, UMAP, LAMP, and MDS. Rows above: projections done by our method using varying numbers of epochs.

for it. Table 2 shows the resulting numbers of training epochs required which we can select this way. This justifies the maximal preset of 200 training epochs (and its average of 30 epochs) mentioned in section "Method."

Capturing the structure of different datasets

In DR, the quality of a projection, measured by any of the metrics discussed in section "Method" (or any other desirable quality metric), highly depends on the kind of input data and kind of projection technique used.[3] We first assess this quality by visually comparing the results of eight different projection techniques with those created by our learning technique, trained on the respective projections. The considered techniques are t-SNE, UMAP, Isomap, PCA, LAMP, LLE, and two autoencoders using one layer and three layers, respectively (AEC1 and AEC3). We included the autoencoder-based techniques as they are related to our approach as they also use deep learning. However, autoencoders work differently, since they do not use an actual 2D ground truth to learn from, as we do. Figure 7 illustrates this for four datasets (MNIST, Fashion MNIST, Dogs vs Cats, and IMDB). Figure 8 adds four extra datasets to the evaluation (Spambase, Seismic, Har, and Wisconsin breast cancer (WBC)). The training set sizes were 5K samples in all cases. Several observations follow.

Learning quality. We see that our method can generate projections which are visually almost identical to the ground-truth ones. However, we also see that the learned projections appear sometimes to be slightly more "fuzzy" than the ground truth ones. This happens more for certain (dataset, technique) combinations,

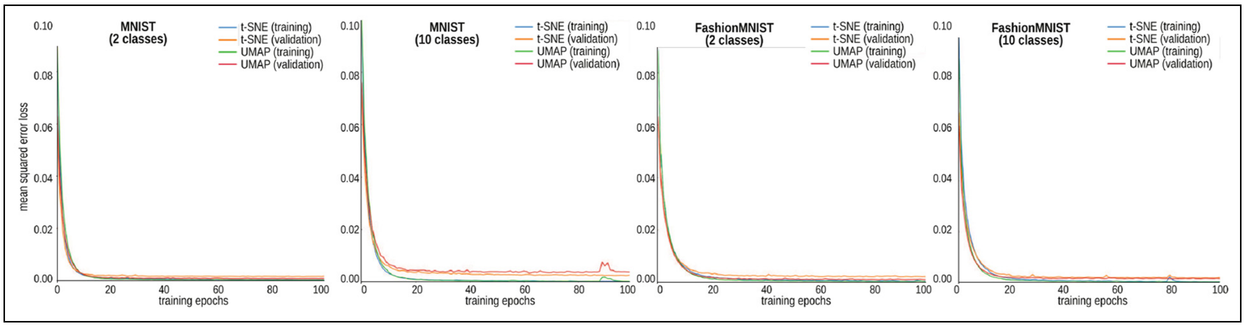

Figure 6. Loss as function of number of training epochs, during both training and validation, for MNIST and FashionMNIST datasets, 2-class and 10-class.

Table 2. Number of training samples versus number of epochs needed to obtain convergence, MNIST dataset.

Projection	Classes	Samples	Epochs
t-SNE	2	1000	57
t-SNE	2	2000	30
t-SNE	2	3000	50
t-SNE	2	5000	32
t-SNE	2	9000	24
t-SNE	10	1000	49
t-SNE	10	2000	33
t-SNE	10	3000	31
t-SNE	10	5000	21
t-SNE	10	9000	13
UMAP	2	1000	44
UMAP	2	2000	21
UMAP	2	3000	31
UMAP	2	5000	28
UMAP	2	9000	42
UMAP	10	1000	31
UMAP	10	2000	30
UMAP	10	3000	33
UMAP	10	5000	23
UMAP	10	9000	21

t-SNE: t-distributed stochastic neighbor embedding; UMAP: uniform manifold approximation and projection.

see for example, MNIST with t-SNE (Figure 7) or Har with t-SNE and UMAP (Figure 8) and far less for other combinations. The similarity of our results with ground-truth is also reflected in the neighborhood hit (*NH*) values: for all datasets, our method yields *NH* values which are very close to those of the ground-truth projection.

Projection quality. Figures 7 and 8 show quite clearly that different ground-truth projection techniques yield very different results in terms of visual structures for the same dataset. For example, for the Seismic dataset, PCA identifies six clearly separated compact clusters; AEC1 and AEC3 only find three clusters here; and LLE finds numerous small-size clusters (Figure 8). We also see that the identified clusters correlate sometimes well with class labels—for instance, all projections achieve a quite good visual separation of the Cats from the Dogs class in the Cats and Dogs dataset (reflected by the high *NH* values of all images in the respective panel in Figure 7). At the other extreme, separating the two classes of IMDB is very hard for all projections. MNIST and Fashion MNIST fall in the middle, with t-SNE and UMAP achieving good class separation, and Isomap and PCA faring worse. Hence, different projection techniques will fare very differently in uncovering structures in the data for different datasets. However, in all cases, our learned projection *mimics very closely* this ground truth. In other words, if one can find a projection technique that works well for a given dataset and task, for example, showing the correlation of data clusters with class labels, our method can learn to do the same. Conversely, if a projection technique scores poorly on a dataset, for example, cannot identify interesting structures in the data, our technique will not be able to do better.

Since we learn from the ground-truth projections by considering them as "black boxes," we claim, although we cannot (of course) formally prove that we can learn any projection in the same way, apart from the eight techniques demonstrated in Figures 7 and 8. This is in contrast to the only other parametric projection technique based on deep-learning that we are aware of,[9] which can only learn t-SNE.

Dataset difficulty. Figure 9 shows additional results that illustrate how our method behaves when the "difficulty" of the dataset to be projected increases. For this, we considered two datasets (MNIST and Fashion MNIST) which all have 10 classes of samples. The dimensions of these samples are known to be predictive of the class labels. That is, a good projection should be able to create 10 well-separated clusters of

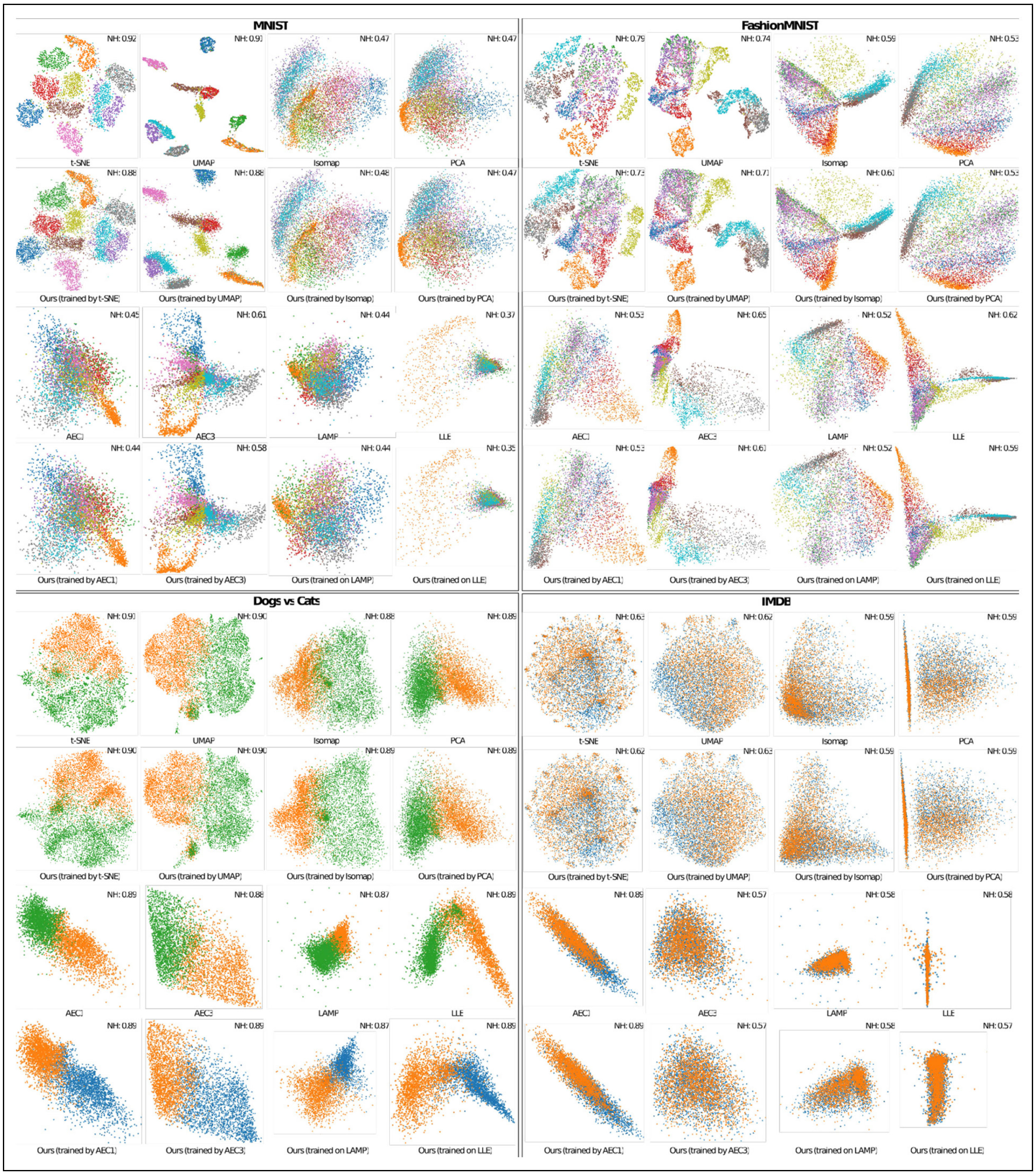

Figure 7. Projections (15K samples) learned from eight techniques (t-SNE, UMAP, Isomap, PCA, LAMP, 1-layer and 3-layer autoencoders (AEC1, AEC3), and LLE) for the MNIST, Fashion MNIST, Cats versus Dogs, and IMDB datasets. Below each projection, the results of our technique are shown. See section "Capturing the structure of different datasets."

same-label samples. The difficulty of separation is also known to be higher for Fashion MNIST than MNIST. We first only consider an easy separation problem, by taking only samples of two classes of each of these datasets. In all cases, we used 5K training samples and projected a different set of 5K samples from each of these four datasets. Figure 9, top two panels, shows ground-truth projections computed by PCA, Isomap, MDS, and LLE for these two-class datasets. We see that all projections can separate the two classes very

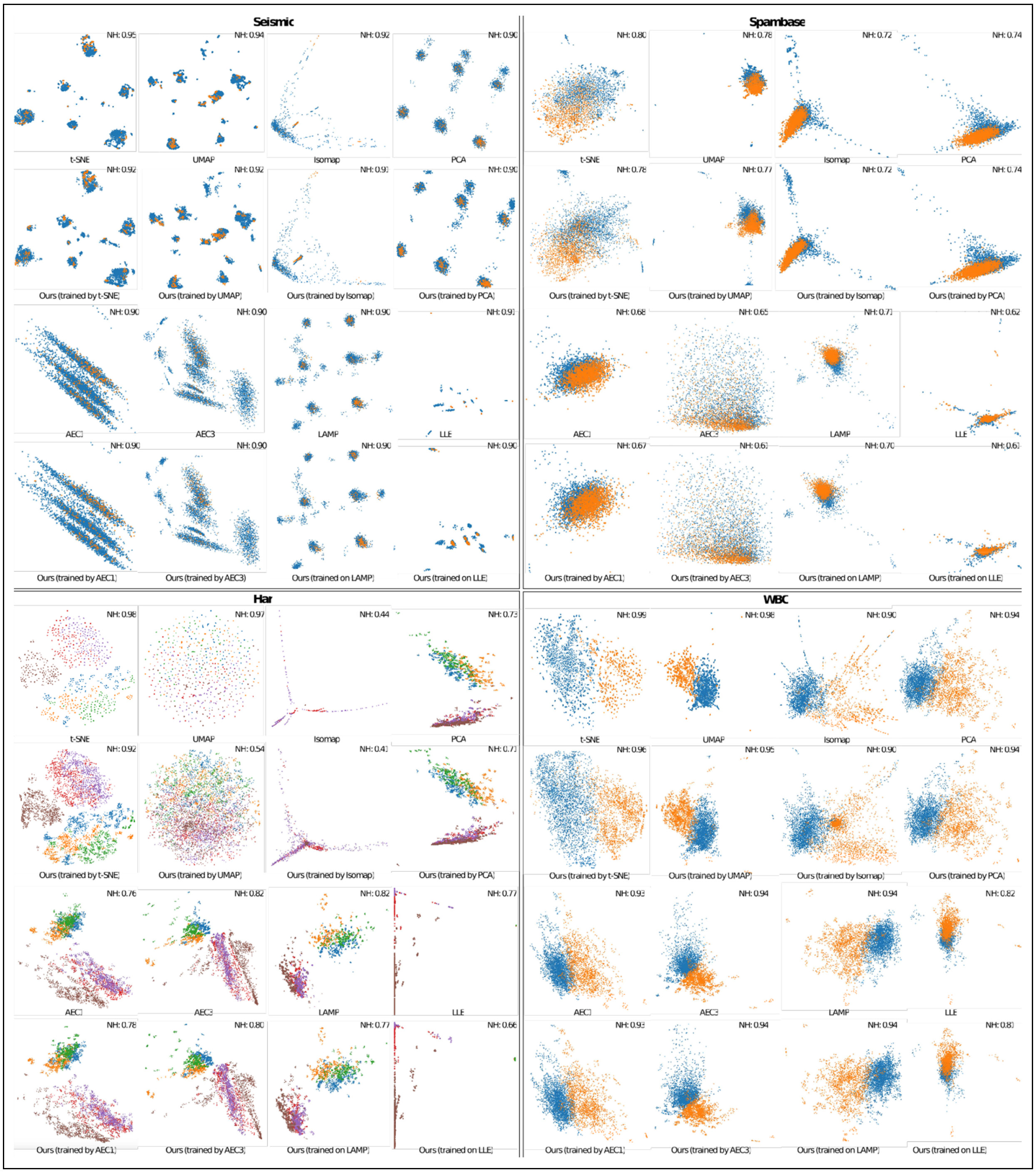

Figure 8. Projections (15K samples) learned from eight techniques (t-SNE, UMAP, Isomap, PCA, LAMP, one-layer and three-layer autoencoders (AEC1, AEC3), and LLE) for the Seismic, Spambase, Har, and WBC datasets. Below each projection, the results of our technique are shown. See section "Capturing the structure of different datasets.".

well (also indicated by the high *NH* values), and so do also our learned projections. The bottom two panels in Figure 9 show the projections of the full 10-class datasets. For these more complicated datasets, we see that all the considered ground-truth techniques have considerable difficulty in separating the 10 classes well, and so do also our learned projections. Hence, we can conclude that our learned projections mimic very closely the behavior of the ground truth ones, whether this means good or poor class separation.

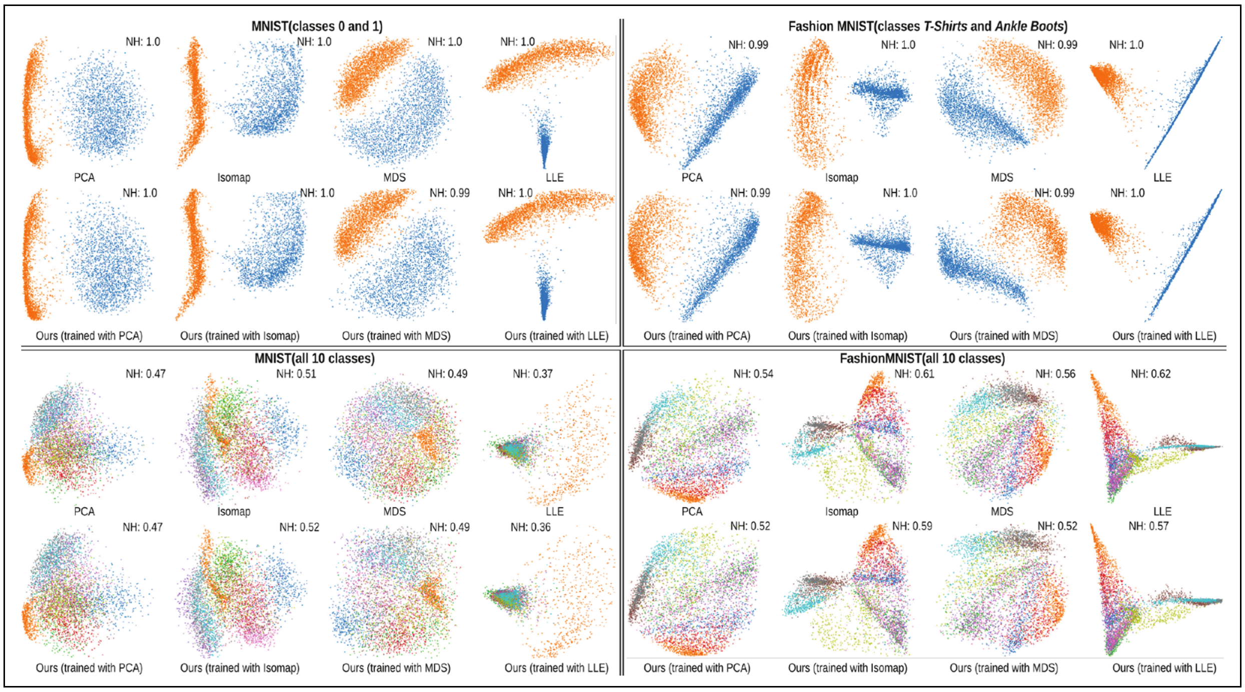

Figure 9. Learning different projections for four datasets (MNIST and FashionMNIST, 2 and 10 classes). In each subfigure, top row: projections of 5K samples made with PCA, Isomap, MDS, and LLE; bottom row: projections of different 5K samples, same datasets, created by our method trained on the data and projections from the top row.

Supplemental Table 5 shows the values of the quality metrics introduced in section "Method" (trustworthiness *T*, continuity *C*, neighborhood hit *NH*, Shepard diagram correlation *R*, and the technique-specific metric *S*) for the 10 projection techniques used to project the 10 datasets in Figures 7–9. For each (dataset, projection) pair, we evaluate the five metrics both on the ground-truth projection and on our learned projection and next study the signed difference between the former and the latter. Figure 10 visualizes these signed differences for all the 100 (dataset, projection) combinations. Care was taken to compute the differences in the correct direction: For the general metrics *T*, *C*, *NH*, and *R*, higher difference values are better, since these indicate that our method yields higher quality than the original learned methods. For the technique-specific metrics *S*, lower is better, as this indicates that our technique minimizes the underlying cost function better than the ground-truth projections. Overall, Figure 10 shows that the metric values of our projection are very close to the corresponding ground-truth values, the differences being maximally 4% and on average under 2%, respectively. This strengthens the insight obtained earlier by visually comparing our learned projections with the ground-truth ones that our method learns very well the characteristics of the projections it was trained to mimic. Apart from that, Figure 10 shows that there is no clear correlation between high (or low) difference values and specific projection techniques or specific datasets, with the exception of the (Har, LSP) combination where our technique yields poorer *T*, *C*, and *NH* values than the ground-truth, but identical *S* values. In other words, our learned projections perform equally well for all techniques over all datasets.

Stability and out-of-sample data

We define *stability* of a projection as the relation between the *visual* changes in *P*(*D*) related to *data* changes in *D*. Ideally, a stable projection technique should not change *P*(*D*) if *D* does not change at all, regardless of changes in parameters of the algorithm *P*; and conversely, when *D* changes, for example, as new samples are added, then the old samples should stay in *P*(*D*) as close as possible to their original locations. This way, the user can relate changes in *P*(*D*) to actual data changes. For a similar reasoning applied to different infovis algorithms, that is, treemaps, see Vernier et al.,[50] or even closer to the context of DR, graph drawing.[51] Hence, stability and out-of-sample capabilities are closely related. Stability in the context of projecting high-dimensional data is argued for, independently, by several authors: Joia et al.[25] argue for stability for preserving the user's mental map for LAMP and use Procrustes analysis to align projections

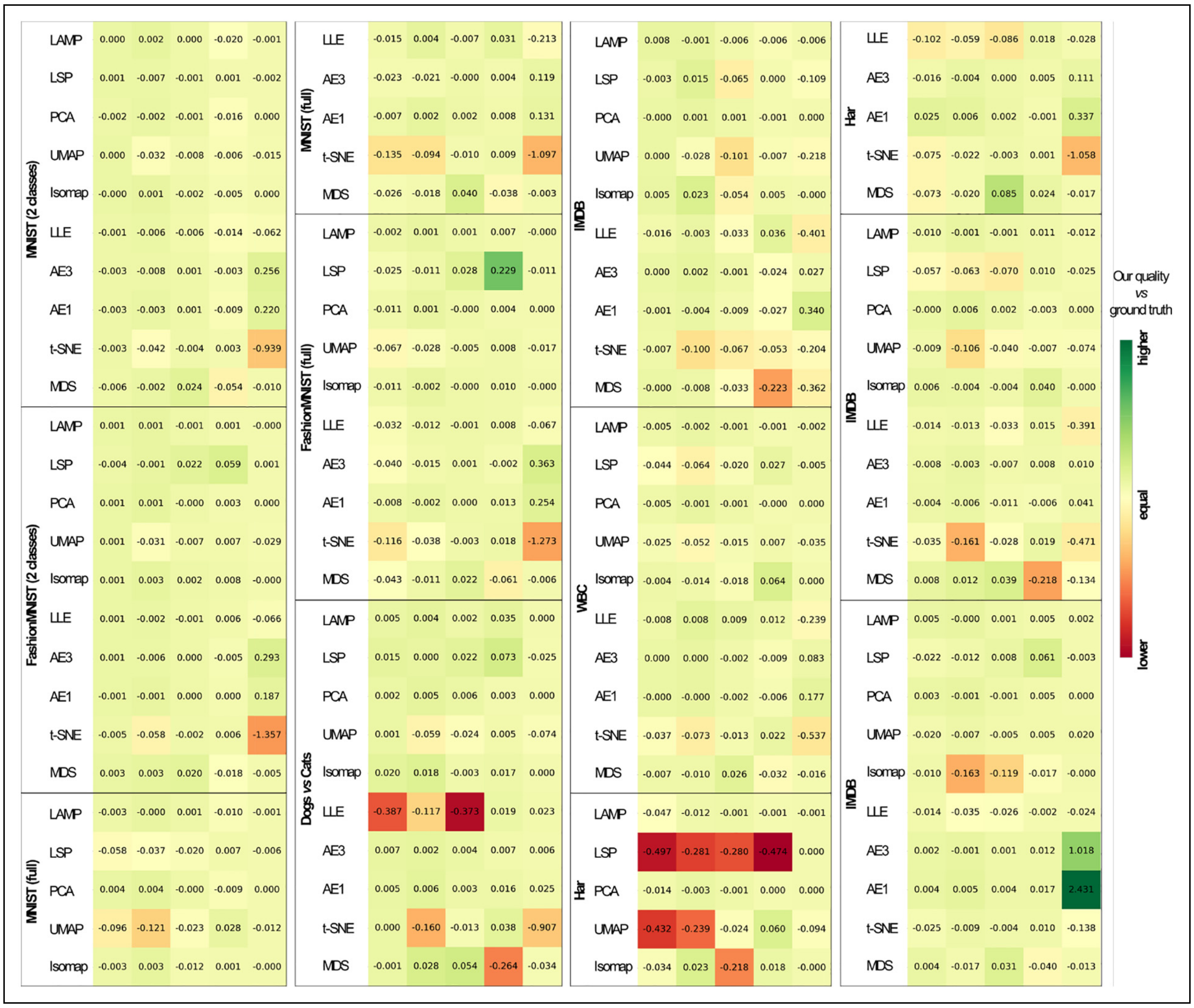

Figure 10. Signed difference between quality metrics computed on ground-truth projections versus our projection (trained with 5K samples), for 10 techniques and 10 datasets. Green indicates cases when our technique exceeded the ground truth quality; red indicates the opposite case.

of different datasets to ensure consistency. Rauber et al.[8] analyze deeper the trade-off stability versus accuracy for t-SNE and adapt the method to this end to handle time-dependent datasets. Even closer to our focus, Van der Maaten[9] argues in detail about why out-of-sample capability for DR is essential for classification and regression, and why this is not optimally achieved by standard techniques such as autoencoders. Nonato and Aupetit[3] argue for stability as being a key feature of DR algorithms in a comprehensive recent survey of such methods. Also, one of the key criteria was that UMAP was developed to satisfy its stability.[13] An additional strong argument for out-of-sample capability, and how add this to techniques such as LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering, is made by Bengio et al.[52]

The joint added-value of stability and out-of-sample capability can actually be explained intuitively, as follows:

Dynamic data. A dataset D may not be a *singular* item, but part of a collection $\{D_i\}$, as in the case of sampling a time-dependent some phenomenon. If projecting every single element D_i of this collection yields fundamentally different projections $P(D_i)$, even though the frames D_i do not change much with respect to each other, then the projection method P may optimally represent *the individual frames D_i taken separately*, but not *the entire collection* $\{D_i\}$. When analysis tasks target the collection, we need the latter, not the former, optimization.[3,8]

Usability. A projection $P(D)$ is the result of a given dataset D, *plus* any parameter settings of the projection algorithm P itself. From a practical viewpoint, users do not want to get massively different results $P(D)$ when only minute details change in either the dataset or the parameters, otherwise the algorithm P is suboptimal: Users see significantly different results for such minute changes, and next wonder whether these reflect indeed significant changes in the data D, or just noise artifacts in D or issues with parameter settings of P. An unstable DR method P does not give an answer to this, leaving the user doubting how to interpret *changes* in the visualization $P(D)$.

As outlined above, obtaining stability and out-of-sample capability is not trivial. Many projection techniques use a random initialization, which means they create quite different results for the same dataset D for different runs. Moreover, small parameter changes, for example, perplexity for t-SNE, or choice of control points for LAMP, to mention just a few, can yield large changes in $P(D)$.[7] Dynamic t-SNE corrects such effects up to a certain level, but comes with additional complexity and significant computational costs.[8]

Figure 11 shows projections of increasingly large, randomly selected, point subsets of the MNIST dataset, using five projection techniques (t-SNE, UMAP, pt-SNE, LAMP, and LSP). We compare these with our method, trained on 5K samples from each of the above projections. Several observations follow. First, by scanning rows in Figure 11, left to right, we see that in all ground-truth projections, except pt-SNE, the same-label clusters *move* in the projection as the sample count changes. This confirms that these methods are not suitable for out-of-sample applications, as users would have difficulties in maintaining the mental map of the data. In contrast, our method (trained with any of the ground-truth projections), and pt-SNE, show stable clusters that stay in the same places in the projection, and only grow denser as more samples are projected—thus, they have the desired out-of-sample capability. The price to pay for this is the increased fuzziness of our out-of-sample projections. This is especially visible when we learn from t-SNE and UMAP: for those rows, our learned projections show clusters where points having different labels (colors) mix more than in the ground-truth projections and have lower *NH* values than these. However, for LAMP and LSP, our learned projections achieve similar visual quality, and sometimes even marginally higher *NH* values as compared to the ground truth. This is explained due to the poorer separation (quality) of the original LAMP and LSP ground-truth, which is thus easier to learn than in the case of t-SNE and UMAP. Separately, compared to pt-SNE, the only ground-truth projection discussed here that has out-of-sample capability, we achieve only marginally lower *NH* values (when trained with t-SNE) and actually *higher NH* values (when trained with UMAP). Summarizing all above, we conclude that our method proposes a good trade-off between stability (and out-of-sample capability) versus projection accuracy.

Computational scalability

One of our main goals is to create a projection technique which scales to large datasets (C2, section "Introduction"). To analyze this, Figure 12 shows a time comparison between t-SNE, UMAP, pt-SNE, MDS, LAMP, LSP, and our method trained to mimic these projections, for increasingly large subsets of the MNIST dataset, up to 1 million samples. We trained our method with (only) 5K samples, in line with training-set sizes found to be sufficient in our earlier experiments (section "Training effort").

We first compare the performance of our *end-to-end* method, that is, computing the training projection (called seeding in Figure 12), training itself, and inference. The thin-dotted lines in Figure 12 show the sum of these three times for our method. As visible, these lines are almost horizontal, which indicates that our timings are dominated by the constant seeding plus training time, and not inference.

As comparison baselines, the continuous (undotted) lines show the timings of the original ground-truth algorithms. We see that our method already runs much faster than the ground-truth algorithms, *even when considering the training cost*: for the maximum sample count (1 million), we are roughly three orders of magnitude faster than t-SNE and two orders of magnitude faster than LAMP. The LSP, MDS, and UMAP implementations we used in the test were not able to handle this large number of samples, shown by their respective curves that stop in Figure 12 at roughly 20K, 30K, respectively, 500K samples. From the slopes of these curves, we see that our method is several orders of magnitude faster than these algorithms for the respective sample counts.

The above comparison of our end-to-end method, including seeding, training, and inference, to the original algorithms, is a worst-case scenario: In practice, one would train only once on a given data universe $\mathcal{D}$ and project many times on the same $\mathcal{D}$. Hence, we next analyze only the inference (projection) time for our method (blue long-dash curve). This time is identical for learning any projection. We see that our method is faster than all other ground-truth methods for all sample counts. For the maximal sample count, we are about two orders of magnitude faster than LAMP, and three-and-a-half orders of magnitude faster than t-SNE. We also see that we are one order of magnitude faster than pt-SNE. This is an important result, since,

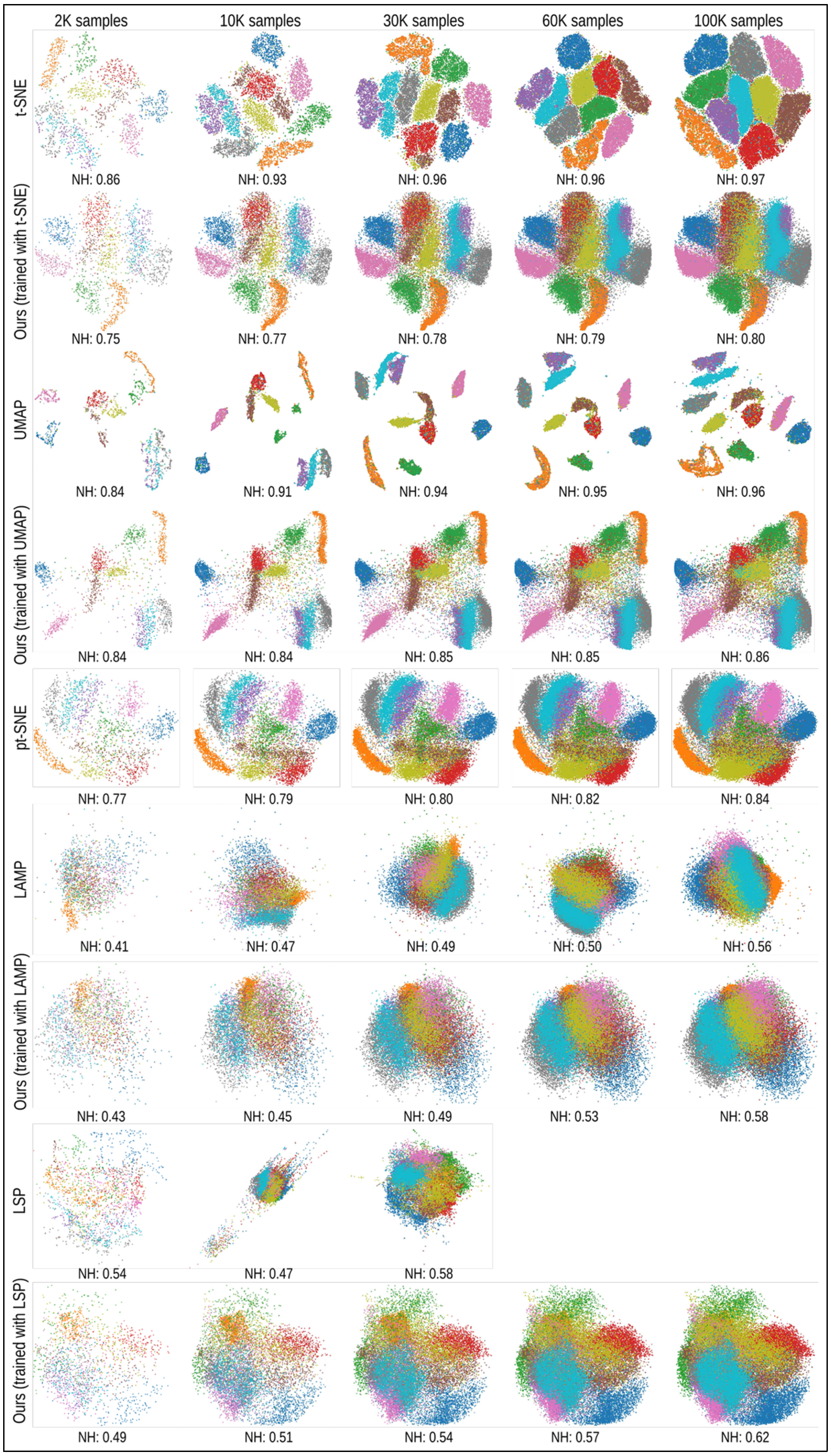

Figure 11. Out-of-sample capability: projecting increasing number of samples from the MNIST dataset with different techniques. Empty spaces indicate tests that did not complete (the ground-truth projection algorithm could not handle too many samples).

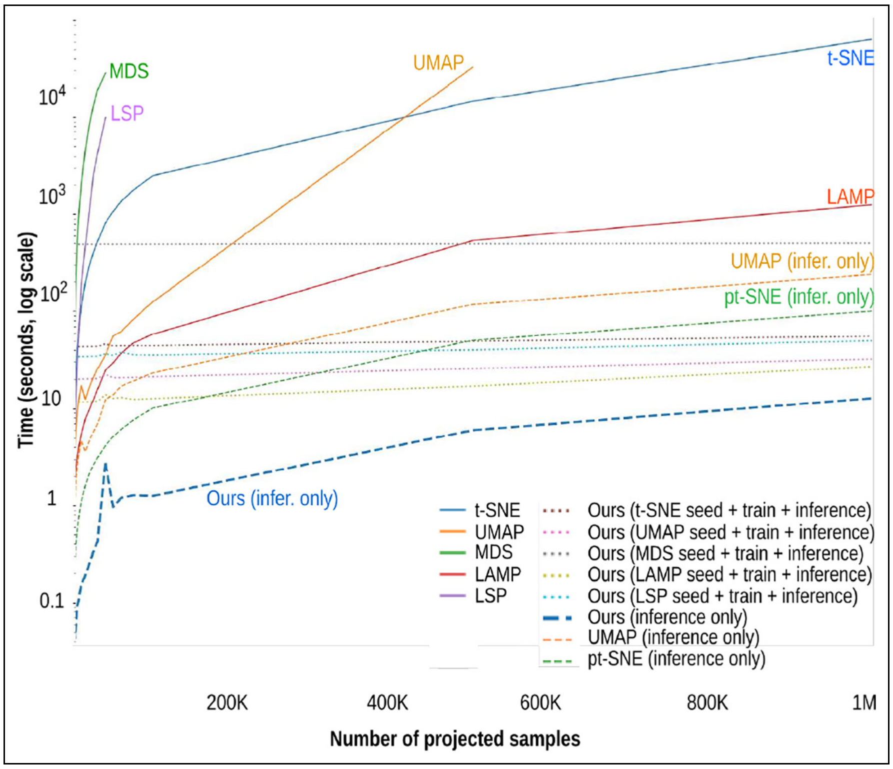

Figure 12. Time to project varying number of samples, MNIST dataset, oversampled to 1 million observations (log time scale).

as already discussed, pt-SNE is the only other parametric projection technique using deep learning that we are aware of. This, together with the quality results discussed in Figure 11, show that our method is a competing alternative, concerning both quality and speed, to pt-SNE.

Finally, we consider UMAP's out-of-sample capability (see section "Related work" for details): we run UMAP on our training set, which makes it learn a function to transform the high-dimensional data to 2D. Note that this is completely different from our deep learning—it is a particular feature of UMAP's implementation, not shared by any other projection techniques we know of. Next, we let UMAP use this learned function to project the test set. In this inference-only scenario, our method (again, the blue long-dash curve) is about one-and-a-half orders of magnitude faster than UMAP (orange dashed curve).

All experiments were run on a 4-core Intel E3-1240 v6 running at 3.7 GHz with 64 GB RAM and an NVidia GeForce GTX 1070 GPU with 8 GB VRAM. Supplemental Table 3 lists all hyperparameters used, and Supplemental Table 4 lists all open-source software libraries used in all our experiments. Our neural network implementation leverages the GPU power using the Keras framework. The t-SNE implementation used is a parallel version of Barnes-Hut t-SNE,[53] run on all four available CPU cores for all tests. The UMAP reference implementation is not parallel, but is quite fast (compared to t-SNE) and well-optimized. Our implementation, plus all code used in this experiment, are publicly available at https://github.com/mespadoto/dlmp.

Projecting unrelated data

So far, we showed that our method can learn from a subset of a given dataset D to project unseen samples from the same D. This serves the concrete purpose of *accelerating* projections of large datasets (see, for example, Figure 12 and related text), by training our method on the projection of a small subset thereof, followed by inference on the entire dataset. The same approach can be used when one wants to project very similar datasets, drawn from the same distribution, that is, sampling the same phenomenon.

A different question arises in this context: can we use our method to project (infer) data which is quite different from the training data? In other words: can we *reuse* the training done on a given type of data from some universe $\mathcal{D}$ (for which we have, for instance, sufficient training samples) to generate a network able to project data from a related, but still different, universe $\mathcal{D}'$?

To answer this question, we conducted the following experiment. We trained our method using UMAP and t-SNE projections of 2K observations from CIFAR-10 (classes *Airplane, Frog* and *Truck*), which can be seen as a sampling of $\mathcal{D}$, the universe of natural images of vehicle-and-animal shapes. Next, we used the trained network to project 4K observations from CIFAR-100 (classes *Trees, Large Carnivores* and *Vehicles 2*), which constitutes a sampling of $\mathcal{D}'$—a universe related, but not identical to, $\mathcal{D}$. We selected these classes because they contain images that are similar perceptually between the two universes $\mathcal{D}$ and $\mathcal{D}'$, with the goal of checking the capability of generalization of our method. Note however that $\mathcal{D}$ and $\mathcal{D}'$ are quite different: while both contain images, these are of different kinds, and acquired by different procedures.

Figure 13 shows the obtained results. First, we show the projections obtained by directly reusing the network trained on $\mathcal{D}$. As visible, the results, shown in Figure 13(b), are quite far from the ground truth (classical t-SNE and UMAP projections, Figure 13(a)). This confirms that $\mathcal{D}$ and $\mathcal{D}'$ are, indeed, quite different, so directly reusing the training from $\mathcal{D}$ to $\mathcal{D}'$ is not possible.

We next consider training the network *from scratch*, using a small number σ of 100–1000 samples from $\mathcal{D}'$, mimicking the situation when the user has only few available data from $\mathcal{D}'$ to train on. The results are shown in Figure 13(c). As visible, when we increase σ, the from-scratch training results get closer to the ground-truth (Figure 13(a)). Next, we consider a network pre-trained on $\mathcal{D}$ (the original universe), which we further train (fine-tune) with σ samples for an increasing number of epochs e (from 100 to 700). The procedure is very similar to *transfer learning*.[54] Figure 13(d) shows the results of this fine tuning for different combinations of σ and e. If we look at the rows of Figure 13(d), we see that the respective images are more similar to Figure 13(a) than the image (c) corresponding to the same row. As we increase σ and e, these images become increasingly more similar to the ground truth. For instance, we see that the fine-tuned network can already capture the green spike detail marked in blue in Figure 13(a) from $\sigma = 500$, $e = 500$, as shown by the red circles in image (d). When training from scratch, this detail requires $\sigma = 1000$ samples to become visible for UMAP and cannot be captured even for this sample count for the t-SNE projection. Hence, we conclude that we can mimic the ground-truth projection of $\mathcal{D}'$ better by fine-tuning a network pre-trained on a different universe $\mathcal{D}$ than training from scratch on $\mathcal{D}'$ with the same number of samples.

We can draw the following conclusions from this experiment:

- Directly extrapolating training from a universe $\mathcal{D}$ to a different universe $\mathcal{D}'$ will not give good projection results.
- Fine-tuning an existing training on $\mathcal{D}$ with a small number of samples drawn from $\mathcal{D}'$ is possible and can lead to results close to the ground-truth projection of data from $\mathcal{D}'$.

We should stress that the above experiment only hints the possibility of transfer-learning-like training of projections. We do not have enough evidence to assess how much additional training data (from $\mathcal{D}'$) and training time is needed, in general, when extrapolating between two different universes. Moreover, the relationship between the additional training data and training effort required to reach a certain similarity to the ground-truth projection and the similarity of the universes $\mathcal{D}$ and $\mathcal{D}'$ is yet unknown. We consider this to be an interesting topic for future work.

Discussion

We next discuss how our proposal meets the requirements introduced in section "Introduction."

Quality (C1)

We showed that our method achieves very similar quality (measured by five established metrics in DR) to projections well known to perform well in this area on six challenging multidimensional datasets having up to thousands of dimensions, which are often used as benchmarks in machine learning. Visual comparison also shows that our projections are very close to those computed by existing methods, which, as discussed already, are slower and harder to configure.

Scalability (C2)

Even when considering training, our method is roughly one order of magnitude faster than t-SNE and roughly five times faster than UMAP for more than roughly 30K samples. As explained in section "Computational scalability," this is a worst case, since one typically trains once and infers many times on a given data universe. For such cases, our method is more than three to four *orders of magnitude* faster than t-SNE and UMAP and allows projecting data of millions of samples in a few seconds. The complexity of our method is linear in the number of observations and dimensions. Besides t-SNE and UMAP, our method is actually also faster than other projection methods such as Isomap, LAMP, and MDS.

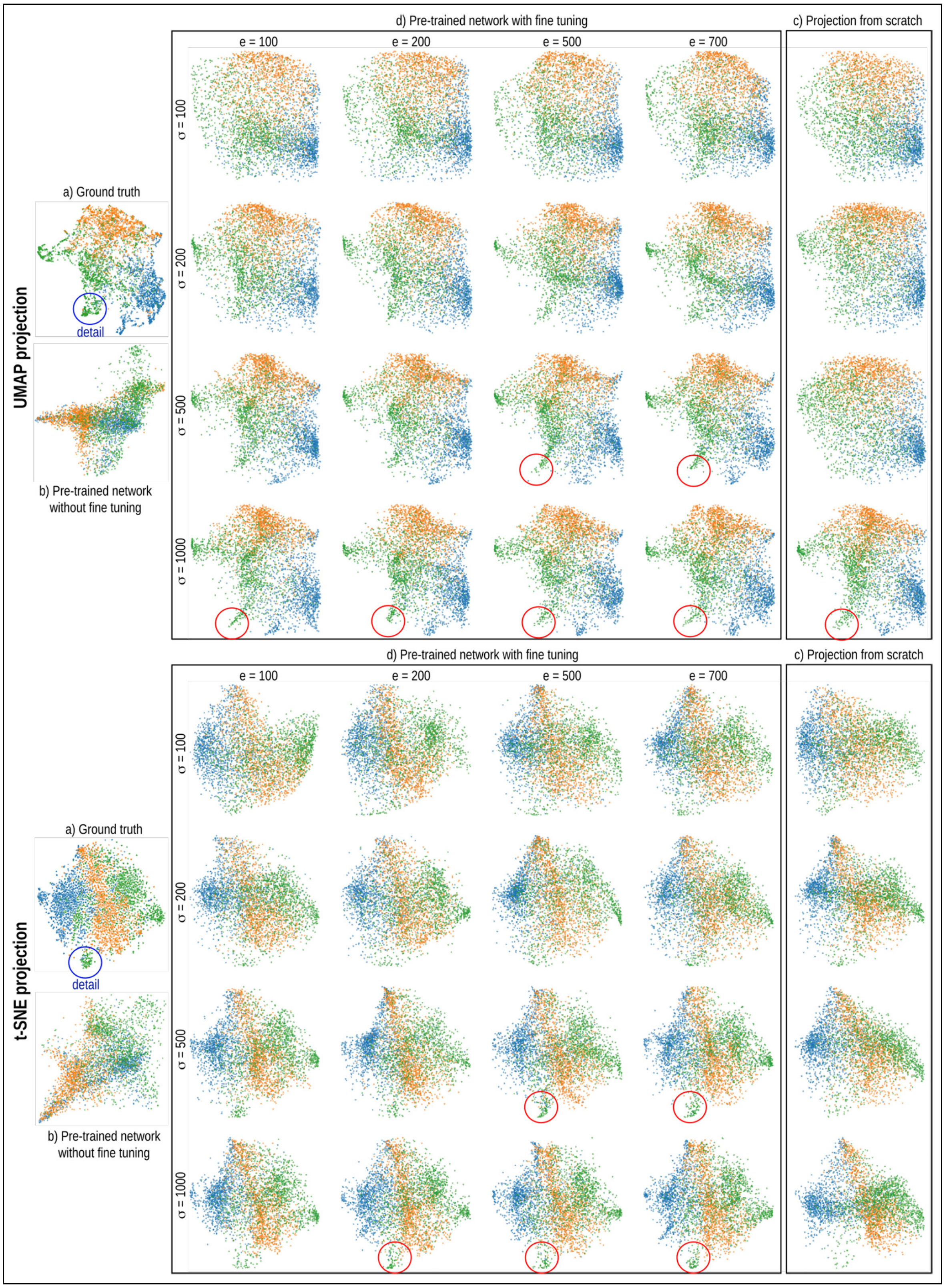

Figure 13. Projecting data (4K samples) by fine tuning pre-trained networks mimicking UMAP and t-SNE. (a) Test projection. (b) Inference by pre-trained network without any fine tuning. Training uses 2K samples from universe $\mathcal{D}$. (c) Projections made by our method trained from scratch from σ samples from the new universe $\mathcal{D}'$. (d) Projections made by with fine-tuning the pre-trained network with varying numbers of training epochs e and using different numbers σ of samples from $\mathcal{D}'$. Red markers show how the fine-tuned network can already capture a ground-truth detail (shown separately in blue) with fewer training samples σ than when training from scratch with the same σ.

Ease of use (C3)

During inference, our method simply executes a trained neural network, which requires no parameter setting. There is no need for guessing the "right" values of parameters such as t-SNE's perplexity.[7] During training, the only free parameter to be set is the maximal loss or, alternatively, number of training epochs. The two are related, see section "Training effort." As also explained there, a preset of 200 training epochs yielded a loss of 0.005, that is, practical convergence, for all examples we considered.

Genericity (C4)

Our method can learn the behavior of any type of projection technique. We provided examples in section "Results" showing this for t-SNE, LAMP, UMAP, MDS, Isomap, LLE, PCA, and pt-SNE. All that is needed to learn is a number of samples from the data universe of interest, represented as n-dimensional feature vectors, and their 2D coordinates computed by the desired projection technique. No other aspects or parameters of the training or inference process are projection-technique specific—that is, projections to be learned can be seen as black boxes. Moreover, no restrictions exist in terms of the dimensionality n of the input feature space in which the data are represented and/or the dimensionality q of the projected data. While we demonstrated our approach only for $q = 2$ (2D projections), which are the most commonly used in infovis, producing higher-dimensional, for example, three-dimensional (3D) projections,[55] is equally easy. Such projections are preferred in certain cases as they can preserve the original data structure better than 2D projections.[55,56] Of course, for our method to be usable, data should come as n-dimensional feature vectors. This is direct, for example, in the case of tabular data or images, as discussed in the examples in the paper; for other data types, such as text or videos, suitable feature extraction methods should be used. This is however not a limitation of our method as opposed to other DR methods. Separately, we note that we have only considered projecting quantitative data so far. However, extending our approach to handle categorical data is straightforward using, for example, one-hot encoding or similar techniques.[57]

Stability and out-of-sample support (C5)

These two issues are strongly interconnected, and actually also connected with the question of how far our networks can generalize what they learn. Let us detail. As outlined in section "Method," we take a training set D_s which is supposed to represent well the overall data distribution in a given so-called data universe $\mathcal{D}$, that is, datasets related to a particular application, such as all handwritten digits, all human face images, all patients in a given population, all street views, and similar. Our approach learns how to project data in $\mathcal{D}$ based on training projections of data in D_s. Hence, the better D_s represents the variability of data in $\mathcal{D}$, the better will our projections mimic actual projections of the same data. Given that neural network inference works deterministically, out-of-sample support is stable in the sense that the same data items (in a dataset $D \subset \mathcal{D}$) are projected to the same locations, which is not the case for many projection methods such as t-SNE, UMAP, and LAMP, to mention just a few. Separately, given the dense structure of the fully connected network we use (which averages activations from multiple units in an earlier layer to determine those of the current layer), our approach is stable in the sense that small changes in an input dataset yield only small changes in the resulting projection (see example in section "Stability and out-of-sample data"). Again, this result is far from evident for many existing projection techniques.

Limitations

Our results show that there is a trade-off between the inherent stability and out-of-sample support of our method (discussed above) and the quality (in terms of cluster separation) of the resulting projections. Compared to t-SNE and UMAP, our projections show fuzzier, or less sharply separated clusters. Compared to the other tested projections (MDS, Isomap, LLE, LAMP, autoencoders, PCA), however, our results are almost identical both visually and in terms of the evaluated quality metrics. This trade-off is needed to provide stability: our method cannot project samples as "freely" as for example, t-SNE, since it needs to behave deterministically, like any parametric DR technique; however, this ensures that the same location in nD space projects to the same place in qD space, which is not the case for t-SNE or any other non-parametric DR technique. A similar trade-off between stability and quality exists actually also for dynamic (t-SNE) projections[8] and also for parametric t-SNE.[9] Note that, if desired, we can reduce fuzziness (or, more formally, achieve a higher fit of the learned representation with the training data) by increasing the number of training epochs, decreasing the training loss, or similar well-known techniques in machine learning. However, this is undesired as it can quickly lead to *overfitting*, that is, it will create suboptimal projections from data which is very different from the training set.

Relation to autoencoders

Both our method and autoencoders use deep learning to perform DR and are parametric techniques. However, the similarities end here: our method learns from a 2D projection (scatterplot) provided by a user-chosen projection technique; in contrast, autoencoders train with the n-dimensional data itself. Autoencoders propose an *own* embedding of the high-dimensional data into 2D. In contrast, we *learn* whichever embedding was provided to us by the training projection.

Generalization

Related to the last point above, the question arises of *how far* can our approach generalize, or, how densely do we need to sample an universe $\mathcal{D}$ by the training set D_s to create good projections. This is an open question in machine (and deep) learning in general. Yet, we can make the following practical points. First, for the types of (non-trivial) data universes, we consider in our evaluations, a few thousands of samples yield already high-accuracy results. Second, the larger $\mathcal{D}$ is, the larger (and better spread) the training set D_s needs to be. Section "Projecting unrelated data" outlines the limits of this extrapolation: the farthest away is $\mathcal{D}$ spread from the training set D_s, the more will our projections differ from the actual ground-truth projections obtained using classical projection methods. Again, this is not a surprise, but a well-known fact in machine learning. We argue that this is not a problem *in practice* when using projections. Indeed, in all cases, we are aware of, researchers typically work for a reasonable amount of time on a *given*, and fixed, data universe $\mathcal{D}$. Hence, they can once train a network from a comprehensive $D_s \subset \mathcal{D}$, after which they can use the network with no changes for data in the same $\mathcal{D}$. Moreover, for cases where one targets a new data universe $\mathcal{D}'$, for which obtaining a comprehensive training set D_s is expensive, the transfer-learning-like approach in section "Projecting unrelated data" can be used. As shown in that section, one can fine-tune a pre-trained network (on widely available data from a related universe $\mathcal{D}$) with as few as hundreds of samples from $\mathcal{D}'$.

Conclusion

We have presented a new method for creating projections of high-dimensional data using a machine learning approach. Based on a small number of 2D projections of a subset of samples from a given data universe, obtained using any user-chosen projection technique, we train a neural network to mimic the 2D projection output and next use the network to infer projections of unseen data from the same universe. Our method can mimic the quality and visual style of a wide range of established projection techniques, including the well-known visual cluster separation provided by SNE-class methods; is orders of magnitude faster than such methods; has a single parameter for training (with documented preset), and no parameters for inference; can handle datasets of any (quantitative) kind and dimensionality; and delivers inherent stability and out-of-sample support. Our method is simple to implement, requiring only generic (and easily available) software for neural networks, and we provide its source code for replication and actual usage. We show how our approach yields good trade-offs between quality (on one side) and speed, ease of use, genericity, generalizability, and stability (on the other side).

Many future work directions are next possible. First, we consider generalizing our approach to compute stable projections of dynamic (time-dependent) high-dimensional data and also mixed quantitative-and-qualitative data. Second, we consider using different network architectures, cost functions, and training procedures for more accurate handling of more complex data universes. Last but not least, we consider more refined approaches to tackle the transfer learning problem for generalizing learning from a given number of jointly considered data universes and projection techniques.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This study was financed in part by FAPESP (201522308-2, 201725835-8 and 2018 25671-9) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

ORCID iD

Mateus Espadoto https://orcid.org/0000-0002-1922-4309

Supplemental material

Supplemental material for this article is available online.

References

1. Kehrer J and Hauser H. Visualization and visual analysis of multifaceted scientific data: a survey. *IEEE TVCG* 2013; 19(3): 495–513.
2. Liu S, Maljovec D, Wang B, et al. Visualizing high-dimensional data: advances in the past decade. *IEEE TVCG* 2015; 23(3): 1249–1268.
3. Nonato L and Aupetit M. Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG* 2019; 25(8): 2650–2673.

4. van der Maaten L and Postma E. *Dimensionality reduction: a comparative review.* Technical report, TiCC TR 2009–005, 26 October 2009. Tilburg: Tilburg University.
5. Sorzano C, Vargas J and Pascual-Montano A. A survey of dimensionality reduction techniques. *arXiv preprint arXiv*:14032877, 2014.
6. van der Maaten L and Hinton GE. Visualizing data using t-SNE. *JMLR* 2008; 9: 2579–2605.
7. Wattenberg M. How to use t-SNE effectively, 2016, https://distill.pub/2016/misread-tsne
8. Rauber P, Falcão AX and Telea A. Visualizing time-dependent data using dynamic t-SNE. In: *Proceedings of Eurographics conference on Visualization (EuroVis): short papers*, Groningen, 6–10 June 2016, pp. 73–77. Geneva: Eurographics Association.
9. van der Maaten L. Learning a parametric embedding by preserving local structure. In: *Proceedings of the twelth international conference on artificial intelligence and statistics*, Clearwater, FL, 16–19 April 2009.
10. van der Maaten L. Accelerating t-SNE using tree-based algorithms. *JMLR* 2014; 15: 3221–3245.
11. Pezzotti N, Höllt T, Lelieveldt B, et al. Hierarchical stochastic neighbor embedding. *Comput Graph Forum* 2016; 35(3): 21–30.
12. Pezzotti N, Lelieveldt B, van der Maaten L, et al. Approximated and user steerable t-SNE for progressive visual analytics. *IEEE TVCG* 2017; 23: 1739–1752.
13. McInnes L and Healy J. UMAP: uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv*:180203426, 2018.
14. Hoffman P and Grinstein G. A survey of visualizations for high-dimensional data mining. In: Fayyad U, Grinstein GG and Wierse A (eds) *Information visualization in data mining and knowledge discovery.* Burlington, MA: Morgan Kaufmann, 2001, pp. 47–82.
15. Engel D, Hüttenberger L and Hamann B. A survey of dimension reduction methods for high-dimensional data analysis and visualization. In: *Proceedings of IRTG workshop*, vol. 27, Kaiserslautern, 10–11 June 2011, pp. 135–149. Wadern: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
16. Cunningham J and Ghahramani Z. Linear dimensionality reduction: survey, insights, and generalizations. *JMLR* 2015; 16: 2859–2900.
17. Xie H, Li J and Xue H. A survey of dimensionality reduction techniques based on random projection. *arXiv preprint arXiv*:170604371, 2017.
18. Jolliffe IT. Principal component analysis and factor analysis. In: Jolliffe IT (ed.) *Principal component analysis.* Berlin: Springer, 1986. pp. 115–128.
19. Torgerson WS. *Theory and methods of scaling*. Hoboken, NJ: Wiley, 1958.
20. Tenenbaum JB, De Silva V and Langford JC. A global geometric framework for nonlinear dimensionality reduction. *Science* 2000; 290(5500): 2319–2323.
21. Roweis ST and Saul LK. Nonlinear dimensionality reduction by locally linear embedding. *Science* 2000; 290(5500): 2323–2326.
22. Donoho DL and Grimes C. Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. *Proc Natl Acad Sci* 2003; 100(10): 5591–5596.
23. Zhang Z and Zha H. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM J Sci Comput* 2004; 26(1): 313–338.
24. Zhang Z and Wang J. MLLE: modified locally linear embedding using multiple weights. In: *proceedings of NIPS*, Vancouver, BC, Canada, 4–9 December 2006, pp.1593–1600. Cambridge, MA: MIT Press.
25. Joia P, Coimbra D, Cuminato J, et al. Local affine multidimensional projection. *IEEE TVCG* 2011; 17(12): 2563–2571.
26. Paulovich F, Nonato L, Minghim R, et al. Least square projection: a fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG* 2008; 14(3): 564–575.
27. Hinton GE and Salakhutdinov RR. Reducing the dimensionality of data with neural networks. *Science* 2006; 313(5786): 504–507.
28. Kingma DP and Welling M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
29. Becker M, Lippel J and Stuhlsatz A. Regularized nonlinear discriminant analysis—an approach to robust dimensionality reduction for data visualization. In: *Proceedings of VISIGRAPP*, Porto, 27 February–1 March 2017, pp. 116–127. SciTePress.
30. Hinton GE. Training products of experts by minimizing contrastive divergence. *Neural Comput* 2002; 14(8): 1771–1800.
31. Goldberger J, Roweis S, Hinton GE, et al. Neighbourhood components analysis. *NIPS* 2005; 17: 513–520.
32. He K, Zhang X, Ren S, et al. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: *Proceedings of IEEE international conference on computer vision (ICCV)*, Santiago, Chile, 7–13 December, pp. 1026–1034. New York: IEEE.
33. Kingma D and Ba J. Adam: a method for stochastic optimization. *arXiv preprint arXiv*:14126980, 2014.
34. Espadoto M, Martins R, Kerren A, et al. Towards a quantitative survey of dimension reduction techniques. IEEE TVCG 2019.
35. Venna J and Kaski S. Visualizing gene interaction graphs with local multidimensional scaling. In: *Proceedings of ESANN*, Bruges, 26–28 April 2006, pp. 557–562. Brussels: D-Side Group.
36. Martins R, Minghim R and Telea A. Explaining neighborhood preservation for multidimensional projections. In: *Proceedings of CGVC*, London, 16–17 September 2015, pp. 121–128. Geneva: Eurographics Association.
37. LeCun Y, Cortes C and Burges C. *MNIST handwritten digit database*. Florham Park, NJ: AT&T Labs, 2010, http://yann.lecun.com/exdb/mnist/
38. Xiao H, Rasul K and Vollgraf R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv*:170807747, 2017.
39. Elson J, Douceur JJ, Howell J, et al. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In: *Proceedings of the 14th ACM conference on*

computer and communications security, Alexandria, VA, 29 October–2 November 2007, pp. 366–374. New York: Association for Computing Machinery.

40. Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision. In: *Proceedings of IEEE conference on computer vision and pattern recognition*, Las Vegas, NV, 27–30 June 2016, pp. 2818–2826. New York: IEEE.
41. Deng J, Dong W, Socher R, et al. ImageNet: a large-scale hierarchical image database. In: *Proceedings of IEEE conference on computer vision and pattern recognition*, Miami, FL, 20–25 June 2009, pp. 248–255. New York: IEEE.
42. Maas AL, Daly RE, Pham PT, et al. Learning word vectors for sentiment analysis. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, Portland, OR, 19–24 June 2011, pp. 142–150. Stroudsburg, PA: Association for Computational Linguistics.
43. Salton G and McGill MJ. *Introduction to modern information retrieval*. New York: McGraw-Hill, 1986.
44. Mangasarian OL, Setiono R and Wolberg WH. Pattern recognition via linear programming: theory and application to medical diagnosis. In: Coleman TF and Li Y (eds) *Large-scale numerical optimization*. Philadelphia, PA: SIAM, 1990, pp. 22–31.
45. Anguita D, Ghio A, Oneto L, et al. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In: *Proceedings of international workshop on ambient assisted living*, Vitoria-Gasteiz, 3–5 December 2012, pp. 216–223. Berlin: Springer.
46. Hopkins M, Reeber E, Forman G, et al. *Spambase data set*. Palo Alto, CA: Hewlett-Packard Labs, 1999.
47. Sikora M and Wróbel L. Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Arch Min Sci* 2010; 55(1): 91–114.
48. Krizhevsky A. Learning multiple layers of features from tiny images. *Technical report, Department of Computer Science, University of Toronto, Toronto, ON, Canada*, 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
49. Huang G, Liu Z, van der Maaten L, et al. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
50. Vernier EF, Comba J and Telea A. Quantitative comparison of dynamic treemaps for software evolution visualization. In: *Proceedings of IEEE working conference on software visualization (VISSOFT)*, Madrid, 24–25 September 2018.
51. Beck F, Burch M, Diehl S, et al. Taxonomy and survey of dynamic graph visualization. *CGF* 2017; 36(1): 133–159.
52. Bengio Y, Paiement J-F, Vincent P, et al. Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and spectral clustering. In: *Proceedings of NIPS*, Vancouver, BC, Canada, 13–18 December 2003, pp. 177–184. NIPS.
53. van der Maaten L. Barnes-Hut-SNE. *arXiv preprint arXiv*:13013342, 2013.
54. Pan SJ and Yang Q. A survey on transfer learning. *IEEE TKDE* 2010; 22(10): 1345–1359.
55. Coimbra D, Martins R, Neves T, et al. Explaining three-dimensional dimensionality reduction plots. *Inf Vis* 2016; 15(2): 154–172.
56. Sanftmann H and Weiskopf D. 3D scatterplot navigation. *IEEE TVCG* 2012; 18(11): 1969–1978.
57. Potdar K, Pardawala P and Pai C. A comparative study of categorical variable encoding techniques for neural network classifiers. *Int J Comput Appl* 2017; 175(4): 7–9.