DEEP MANIFOLD COMPUTING AND VISUALIZATION USING ELASTIC LOCALLY ISOMETRIC SMOOTHNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

The ability to preserve local geometry of highly nonlinear manifolds in high dimensional spaces and properly unfold them into lower dimensional hyperplanes is the key to the success of manifold computing, nonlinear dimensionality reduction (NLDR) and visualization. This paper proposes a novel method, called *elastic locally isometric smoothness* (ELIS), to empower deep neural networks with such an ability. ELIS requires that a desired metric between points should be preserved across layers in order to preserve local geometry; such a smoothness constraint effectively regularizes vector-based transformations to become well-behaved local metric-preserving homeomorphisms. Moreover, ELIS requires that the smoothness should be imposed in a way to render sufficient flexibility for tackling complicated nonlinearity and non-Euclideanity; this is achieved layer-wisely via nonlinearity in both the similarity and activation functions. The ELIS method incorporates a class of suitable nonlinear similarity functions into a two-way divergence loss and uses hyperparameter continuation in finding optimal solutions. Extensive experiments, comparisons, and ablation study demonstrate that ELIS can deliver results not only superior to UMAP and t-SNE for and visualization but also better than other leading counterparts of manifold and autoencoder learning for NLDR and manifold data generation.

1 INTRODUCTION

Manifold learning aims to find from a set of higher dimensional data its embedding or representation in a low dimensional latent space. **Nonlinear dimensionality reduction** (NLDR) aims to construct a transformation that is generalizable to unseen data. It is hoped that the lower dimensional representation can be used in conjunction with a simple metric such as the Euclidean distance for downstream tasks such as classification and **visualization**. **Manifold data generation** performs the inverse transformation to generate data from samples in the latent space. We call this collection of manifold related problems **manifold computing**. The basis for manifold computing is the manifold assumption (Mikhail Belkin, 2002; Fefferman et al., 2016).

Great advances have been made in the past two decades in manifold computing and visualization. ISOMAP (Tenenbaum et al., 2000) and LLE (locally linear embedding) (Roweis & Saul, 2000) are classic methods for manifold learning. More recent developments include local geometry-based method (Gashler et al., 2008; Zhang & Wang, 2007; Chen & Buja, 2009; McQueen et al., 2016), graph spectral analysis (Donoho & Grimes, 2003) and latent variable models (Saul, 2020). The most popular high dimensional data visualization methods to date are t-SNE (Maaten, 2014) and UMAP (McInnes et al., 2018), with wide applications such as bio-science and technology (Becht et al., 2019; Dorrity et al., 2020). While the aforementioned are traditional machine learning, deep learning-based methods include autoencoders (Hinton & Salakhutdinov, 2006; Moor et al., 2020). The problem can be considered from the viewpoints of geometry deep learning (Bronstein et al., 2017)) and topology data analysis (Wasserman, 2018; Moor et al., 2020).

The ability to preserve geometric structure of nonlinear manifolds and properly unfold them into lower dimensional hyperplanes is the key to the success of manifold-based computing and visualization. Recently, **Markov-Lipschitz deep learning (MLDL)** (Li et al., 2020) is proposed as a general framework for manifold learning, NLDR, visualization and manifold data generation. The idea is to impose the constraint of geometric isometry across neural network layers to preserve the local geometric structure of manifold data. This effectively transforms a vector-based transformation of conventional neural networks into a local distance-preserving homeomorphism. Such local homeomorphisms avoid the transformation from collapse, twisting, or crossing, so as to improve generalization, stability, and robustness. **Locally isometric smoothness (LIS)** (Li et al., 2020), which imposes straight distance-preserving, is proposed as a method in the MLDL framework. LIS has demonstrated significant advantages in manifold learning and NLDR.

This paper proposes a more advanced method in the MLDL framework, called *elastic locally isometric smoothness* (ELIS), aimed to empower deep neural networks with ability to tackle the high nonlinearity and non-Euclideanity challenges arising from **complicated manifolds in high dimension spaces** that LIS is unable to cope with. Whereas LIS preserves the straight distances between neighboring points, ELIS is based on a similarity metric that is nonlinear in distance and a two-way divergence loss (of nearby neighbors and far-away pairs, respectively); this renders more flexibility and capacity in tackling the challenges yet under the control of the ELIS regularization. As the result, ELIS bridges gaps between non-Euclidean manifolds in the input space and resulting Euclidean hyperplanes in the learned lower dimensional latent space, with geometric structure of the manifolds preserved. Both ELIS and LIS can be considered as a form of graph neural networks (GNN) (Scarselli et al., 2009) but without the aggregation generally present in GNNs. They are more like what is called "manifold learning 2.0" (Bronstein, 2020).

Table 1: Functional Capability of Different Methods

	ELIS,LIS	AE,VAE,TopoAE	ISOMAP,LLE,UMAP,t-SNE
Manifold learning without decoder	Yes	No	Yes
Learned NLDR applicable to test data	Yes	Yes	No
Generate data of learned manifolds	Yes	No	No
Compatible with other NN architectures	Yes	No	No
Scalable to large datasets	Yes	Yes	No

The distinctive features of ELIS (and LIS) in comparison with related methods are summarized in Table 1. ELIS-based neural networks can accomplish all the functionalities in the general MLDL framework, for which none of the methods can achieve. Extensive experiments, comparisons, and ablation study demonstrate that ELIS-based neural networks produce results not only superior to the SOTA t-SNE and UMAP for NLDR and visualization but also better than other algorithms of manifold and autoencoder learning, including LIS, for NLDR and manifold data generation. The main contributions of this paper are summarized below:

(1) Proposing the ELIS constraint in the MLDL framework, based on a similarity metric which is nonlinear in distance. It inherits the metric-preserving property of LIS so that the resulting layer-wise transformation is geometrically smooth, hence topologically homeomorphic, yet possesses more flexibility than LIS in handling highly nonlinear manifolds in high dimensional spaces.

(2) Proposing conditions for a class of nonlinear similarity functions for converting from distance to similarity, in conjunction with a two-way divergence loss. This ensures the metric-preserving and neighbor-confining properties.

(3) Proposing two instances of ELIS-based neural networks: an ELIS encoder for manifold learning and visualization and an ELIS autoencoder for manifold reconstruction and data generation.

(4) Providing several SOTA results that surpass UMAP and other leading algorithms.

In the following, Section 2 introduces LIS and presents ELIS formulations, and the Section 3 presents extensive experiments. The code is provided in the Supplementary Material.

2 Elastic Locally Isometric Smoothness

Both ELIS and LIS are formulated in the MLDL framework (illustrated in Fig.A1 in Appendix) which is aimed to regularize neural transformations through imposing the ELIS constraint between layers to achieve certain well-behaving properties. However, the ELIS formulation tackles challenges of highly nonlinear manifold data in high dimensional spaces using a more flexible and effective way, much inspired by t-SNE (Maaten, 2014) and UMAP (McInnes et al., 2018). Let $X = \{x_1, \dots, x_M\}$ be a set of M samples in the input space $\mathbb{R}^N$ with the index set $\mathbb{S} = \{1, \dots, M\}$. These samples may come from one or several lower dimensional manifolds $\mathcal{M}_X \subset \mathbb{R}^N$. When $\mathcal{M}_X$ is Riemannian, its tangent subspace $T_x(\mathcal{M}_X)$ at any $x \in \mathcal{M}_X$ is locally isomorphic to an Euclidean space of dimensionality $dim(\mathcal{M}_X) < N$. Therefore, we can use a cascade of nonlinear neural transformations to "unfold" nonlinear manifolds in a high dimensional input space into hyper-planar regions in a lower dimensional latent space.

Both ELIS and LIS aim to accomplish the following 4 tasks, of which few neural networks can do all: (1) *Manifold learning*: to learn an embedding in a latent space $\mathcal{M}_Z \subset \mathbb{R}^n$, where $n < N$, based on the local structure of X. (2) *Representation Learning*: to learn the underlying mapping $\Phi : \mathcal{M}_X \Longrightarrow \mathcal{M}_Z$ for the embedding that is generalizable to unseen data $x \notin X, x \in \mathcal{M}_X$. (3) *Visualization*: to visualize the embedding in 2D or 3D space. (4) *Manifold generation*: to find the inverse mapping $\Phi^{-1} : \mathcal{M}_Z \Longrightarrow \mathcal{M}_X$ and generate new data on $\mathcal{M}_X$ from samples in $\mathcal{M}_Z$. ELIS is aimed to surpass LIS.

2.1 THE LIS CONSTRAINT AND NEURAL NETWORKS

The LIS constraint is aimed to best preserve the local distances of the data between two metric spaces, encouraging a vector-based neural transformation $\Phi(X \mid W)$, where W is the transformation matrix of the neural network, to become a well-behaved local distance-preserving homeomorphism. This can be achieved by adding the following LIS loss (Li et al., 2020), imposed between two layers (metric spaces) l and l'

$$\mathcal{L}_{LIS}^{(l,l')}(W) = \sum_{i \in \mathbb{S}} \sum_{j \in \mathcal{N}_i^{(l)}} \left| d(x_i^{(l)}, x_j^{(l)}) - d(x_i^{(l')}, x_j^{(l')})) \right| \tag{1}$$

where $d : X \times X \to \mathbb{R}_{\geq 0}$ is a *dissimilarity metric*, $x_i^{(l')} = \Phi(x_i^{(l)} \mid W)$ is the result of the effective transformation Φ from layer l to l', and $\mathcal{N}_i$ is the set of neighbors of i. Without prior knowledge, d_{ij} is usually computed as the Euclidean distance, albeit it may not well reflect the reality. It is hoped that after a series of proper nonlinear transformations, the input data is transformed into an embedding in the latent space such that the Euclidean distance make more sense in describing mutual relationships between points. In this work, we aim to find such transformations.

The LIS loss effectively minimizes the **bi-Lipschitz constant** of Φ. It is through the neighborhood system, $\mathcal{N} = \{\mathcal{N}_i \mid i \in \mathbb{S}\}$, that the influence of a point on the others is propagated to afar. For this reason, the collection of random variable $x^{(l)}$ constitutes a **Markov random field**. Equ. (1) is defined w.r.t. $\mathcal{N}_i$ (Markovianity) and aimed to minimizing the bi-Lipschitz constant, hence the name Markov-Lipschitz (Li et al., 2020).

The basic LIS loss is augmented by an auxiliary "push-way" term (Li et al., 2020)

$$\mathcal{L}_{push}^{(l,l')}(W) = -\sum_{i \in \mathbb{S}} \sum_{j \notin j \in \mathcal{N}_i^{(l)}} \pi[d_{l'}(x_i^{(l')}, x_j^{(l')}) < B] \; d_{l'}(x_i^{(l')}, x_j^{(l')}) \tag{2}$$

in which $\pi[\cdot] \in \{0, 1\}$ is the indicator function and B is a bound. This term is aimed to help "unfold" nonlinear manifolds, by exerting a spring force to push away from each other those pairs (i, j) which are non-neighbors at layer l but nearby (distance smaller than B) at layer l'.

These two losses are combined to form a LIS-based encoder loss for manifold learning and dimension reduction

$$\mathcal{L}_{Enc} = \sum_{(l,l')} \mathcal{L}_{LIS}^{(l,l')}(W) + \mu \mathcal{L}_{push}^{(l,l')}(W) \tag{3}$$

where μ is a weight and (l, l') is summed over a set of designated layer pairs (currently designed manually). A LIS-based autoencoder can be formulated by applying the LIS constraint between layers within the decoder and between the encoder and decoder layers. LIS-based neural networks have significant advantages (Li et al., 2020).

2.2 THE ELIS CONSTRAINT

The proposed ELIS constraint is aimed to tackle difficulties in “flattening" highly nonlinear manifolds in a high dimensional space into hyperplanes in a lower dimensional space. It imposes a more flexible nonlinear similarity-preserving constraint as opposed to the distance-preserving (isometry) constraint of Vanila LIS. More specifically, ELIS transforms a distance into a similarity metric using a nonlinear function and defines a KL loss based on similarities between nearby pairs and far-away pairs. This makes the metric-preserving constraint of ELIS more flexible than the straight distance-preserving of LIS to accomplish the challenging task.

Moreover, ELIS requires that the smoothness should be imposed in a way to render sufficient flexibility for tackling complicated nonlinearity and non-Euclideanity; this is achieved layer-wisely via nonlinearity in both the similarity and activation functions.

Converting distance to similarity. Following UMAP, we assume that $X^{(l)}$ is fixed (e.g., the input layer) and $X^{(l')}$ at subsequent layers l' are computed as a result of manifold learning. The nonlinear similarities between x_i and x_j at each layer is computed as follows. First, define an nearest neighbor (NN)-*normalized distance*

$$d_{i|j} \overset{\mathsf{def}}{=} d(x_i, x_j) - \rho_i \geq 0 \tag{4}$$

where $\rho_i = d(x_i, x_{nn(i)})$ in which $x_{nn(i)}$ denotes the nearest neighbor of x_i. Then, $d_{i|j}$ is converted to a *similarity metric* $u_{i|j} = g(d_{i|j}) \in [0, 1]$ where g is a *nonlinear* function.

We require that $g(\eta)$ satisfy the following *necessary conditions* $\forall \eta = d_{i|j} \geq 0$:

Condition (1) – it is monotonically decreasing, $g'(\eta) < 0$ for $\eta > 0$;

Condition (2) – its first derivative diminishes in the limit, $\lim_{\eta \to \infty} |g'(\eta)| = 0$.

The first condition ensues a monotonic and inverse relationship between the distance and the similarity. The second condition effectively leads to a neighborhood system bounded softly as opposed to the "hard" bounded neighborhoods in the LIS and provides proper control on contributions of neighboring points to the back-propagation of neural network learning.

We further require that the $g(\eta)$ to be a function of η^2 – for convenience not necessity, such that its first derivative take the form $g'(\eta) = 2\eta h(\eta)$ where $h(\eta)$ is also a function of η^2. $h(\eta)$ can be called *influence function* because it controls how the other neighboring point x_j can influence x_i. **Condition (2)** above restricts the influence from "far-away" point x_j on x_i (between which the distance $\eta_{ij} = \|x_i - x_j\|$ is relatively large) to diminish in the back-propagation process. This provides a properly weighted neighborhood system w.r.t. which the influence between points is limited with certain scope adaptively.

Specifically for ELIS, we define the following σ_i-data-adaptive, ν-parameterized nonlinear similarity

$$u_{i|j}(\sigma_i, \nu) = g(d_{i|j} \mid \sigma_i, \nu) = C_\nu \left(1 + \frac{d_{i|j}^2}{\sigma_i\, \nu}\right)^{-(\nu+1)}, \tag{5}$$

where $\nu \in \mathbb{R}_+$ is similar to the degree of freedom (DoF) parameter in the t-distribution,

$$C_\nu = 2\pi \left(\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)}\right)^2 \tag{6}$$

is a function of ν which sets the limit $\lim_{\nu \to +\infty} g(0 \mid \sigma_i, \nu) = 1$ ($\forall \sigma_i > 0$)), and the data-adaptive parameter $\sigma_i > 0$, playing a calibration role, is estimated from the data by best fitting the equation

$$\sum_{j \neq i} u_{i|j}(\sigma_i, \nu) = \log_2 Q \tag{7}$$

for the perplexity-like hyperparameter Q given. While other choices satisfying the aforementioned necessary conditions, including the normalized Gaussian and Cauchy functions used in t-SNE (Maaten, 2014) and the fitted polynomial function used in UMAP (McInnes et al., 2018), can also work for ELIS, we find Equ. (5) a better choice not only because it produces better results but also because we can use the ν parameter as a continuation tool for preventing the training from converging to bad local minima and for controlling separation margin between different manifolds, as will be shown in the ablation study.

Computing similarities u_{ij} and u'_{ij}. Because the symmetry $u_{i|j} = u_{j|i}$ does not hold due to differences in σ_i for the input layer (l), the following symmetrization is performed

$$u_{ij} = u_{j|i} + u_{i|j} - u_{j|i}u_{i|j}. \tag{8}$$

On the other hand, for the subsequent latent layers, the computation of σ_i and ρ_i for each i would bring about huge computational costs. To overcome this problem, we directly set $\sigma'_i = 1$ and $\rho'_i = 0$ (this also ensures the symmetry $u'_{i|j} = u'_{j|i}$). While σ_i and ρ_i are needed to deal with unevenness and outliers of the data for the input layer, the necessity becomes not so demanding as the layer goes deeper after layers of nonlinear manifold unfolding. From $u_{ij}^{(l)}$ of layer l can be constructed a weighted graph $\mathcal{G}(\mathbb{S}, X^{(l)}, U^{(l)})$ consisting of a set $\mathbb{S}$ of nodes with node attributes $X^{(l)}$ and edge attributes (weights) $U^{(l)} = \{u_{ij}^{(l)} \geq \epsilon > 0 \mid \forall i, j \in \mathbb{S}\}$. The global structure of a manifold is discovered from local geometry of data through the graph $\mathcal{G}$.

Formulating the ELIS losse. ELIS transforms the distance metric d_{ij} into a similarity metric using a nonlinear function $u_{ij} = g(d_{ij})$ and defines the ELIS loss between layers l and l', in terms of similarities $u_{ij}^{(l)} \mid i, j \in \mathbb{S}, i \neq j\}$ at layers l and its counterpart $U^{(l')} = \{u_{ij}^{(l')} \mid i, j \in \mathbb{S}, i \neq j\}$ at layers l'. The ELIS loss is defined by what we call the *two-way divergence* (a.k.a. the fuzzy information for discrimination (Bhandari & Pal, 1993) and the fuzzy set cross entropy in UMAP (McInnes et al., 2018))

$$\mathcal{L}_{ELIS}^{(l,l')}(W \mid X^{(l)}, X^{(l')}) = \sum_{i,j \in \mathbb{S}, i \neq j} u_{ij}^{(l)} \log \frac{u_{ij}^{(l)}}{u_{ij}^{(l')}} + (1 - u_{ij}^{(l)}) \log \frac{1 - u_{ij}^{(l)}}{1 - u_{ij}^{(l')}} \tag{9}$$

The first term is the directed divergence of the two fuzzy sets of similarities, in lieu of the LIS' distance-preserving term of Equ. (1); the second term can be considered as the directed divergence of the two corresponding complement fuzzy sets, replacing the push-way term of Equ. (2).

Equ.(9) is called the "two-way divergence" because the first term on the right side of the equation imposes similarity-based attraction forces between nearby (intra-manifold) pairs whereas the second term exerts dissimilarity-based repulsion forces between far-away (inter-manifold) pairs. In other words, intra-manifold points are transformed to a cluster in the latent space, mainly as the result of the first term whereas inter-manifold point pairs push away from each other to different clusters, mainly due to the second term.

Note also that ELIS applies the two terms in a soft, adaptive way via its weighted neighborhood graph where the edges are effectively restricted by pairs of corresponding nodes (data points) between which the absolute gradients $\left|\nabla_W \mathcal{L}_{ELIS}^{(l,l')}(W \mid X^{(l)}, X^{(l')})\right| \geq \epsilon > 0$ are nonzero, in contrast to the "hard" neighborhood system in LIS.

The ELIS loss can be rearranged as follows

$$\begin{aligned}\mathcal{L}_{ELIS}^{(l,l')}(W \mid X^{(l)}, X^{(l')}) = \sum_{i,j \in \mathbb{S}, i \neq j} & u_{ij}^{(l)} \log u_{ij}^{(l)} + (1 - u_{ij}^{(l)}) \log(1 - u_{ij}^{(l)}) \\ & - u_{ij}^{(l)} \log u_{ij}^{(l')} - (1 - u_{ij}^{(l)}) \log(1 - u_{ij}^{(l')})\end{aligned} \tag{10}$$

When $X^{(l)}$ (hence $u_{ij}^{(l)}$) fixed, the optimization only needs to minimize second part involving $u_{ij}^{(l')}$.

2.3 ELIS ENCODER AND AUTOENCODER

The ELIS encoder consists of a cascade of nonlinear forward neural transformations constrained by the ELIS loss, aimed for manifold learning and NLDR. An ELIS (and LIS) encoder can learn an NLDR transformation without the need for a decoder (as required by autoencoders), and this encoder can generalize to unseen data (that ISOMAP, LLE, t-SNE and UMAP cannot). The total loss for the ELIS encoder is the sum of all the ELIS losses over a prescribed set of layer pairs (l, l')

$$\mathcal{L}_{Enc}(W) = \sum_{(l,l')} \alpha^{(l,l')} L_{ELIS}^{(l,l')}(W) \tag{11}$$

where $\alpha^{(l,l')}$ weight the relative importance of $L_{ELIS}^{(l,l')}$.

The ELIS autoencoder has two purposes: (1) to further regularize or optimize the ELIS encoder-based manifold learning by using an **ELIS decoder**, and (2) to enable generation of new data of the learned manifolds by the trained ELIS decoder. The ELIS autoencoder structure consists of the ELIS encoder and decoder in cascade. The ELIS decoder is aimed to approximate the inverse transformations of the ELIS encoder and is made to be entirely symmetric to the ELIS encoder in its network structure. The overall weight matrices becomes $W = [W_{Enc}, W_{Dec}]$. The loss function is composed of three terms:

$$\mathcal{L}_{AE}(W) = \mathcal{L}_{Enc}(W_{Enc}) + \mathcal{L}_{Dec}(W_{Dec}) + \mathcal{L}_{Rec}(W) \tag{12}$$

where $\mathcal{L}_{Enc}(W_{Enc})$ is the same as Equ. (11), $\mathcal{L}_{Dec}(W_{Dec})$ is defined in the same way following the symmetry, and the **reconstruction loss** $\mathcal{L}_{Rec}(W)$ is the summed over all the corresponding layers

$$\mathcal{L}_{Rec}(W) = \sum_{l=0}^{L-1} \gamma_l \sum_{i=1}^{M} \parallel {x_i}^{(l)} - \hat{x}_i^{(l)} \parallel^2 \tag{13}$$

where $\hat{x}_i^{(l)}$ are the data points at the corresponding layer of the decoder and γ_l are the weights. The constraints due to $\mathcal{L}_{Rec}(W)$ and $\mathcal{L}_{Tie}(W)$ are illustrated by the dashed lines in Fig. A1 in Appendix.

3 Experiments

The following experiments are aimed to evaluate ELIS in comparison with other **five algorithms**: UMAP (McInnes et al., 2018), t-SNE (Maaten, 2014) (for visualization), MLLE (Zhang & Wang, 2007), TopoAE (Moor et al., 2020) and LIS (Li et al., 2020) in terms of visual inspection and numerical metrics for manifold computing, visualization and data generation. **Nine datasets** are used, including **five toy datasets**: (1) SwissRoll (3-D), (2) S-Curve (3-D), (3) Servered Sphere (3-D), (4) SpheresA (101-D) (see (Moor et al., 2020) for the description) and (5) SpheresB (101-D, a modified composition from SpheresA); and **four real-world datasets**: (6) Coil20 (16384-D) and (7) Coil100 (49152-D) (Nene et al., 1996), (8) MNIST (784-D) (LeCun, 2013), and (9) Fashion-MNIST (784-D) (Xiao et al., 2017). The toy datasets are used because their geometric and topological structures are clear for the evaluation. The SpheresA dataset (Moor et al., 2020)is composed of 1 large sphere enclosing 10 small ones in 101-D space. SpheresB differs from SpheresA in that its large sphere consists of only 500 samples (whereas that in SpheresA has 5000) – the data is so sparse that the smallest within-sphere distance on the larger sphere can be greater than that between the larger sphere and some small ones. 5 performance metrics are used for the evaluation, whose exact definitions are given in Appendix A.2.

The **pseudo-codes** of the ELIS encoder and autoencoder and **hyperparameter settings** are described in Appendix A.1. The implementation uses the PyTorch 1.6.1 library running on Ubuntu 18.04 on NVIDIA v100 GPU. The time is spent mainly in the computation of neighbors. At present, the ELIS algorithms computes the neighborhood for every point pair, hence have the complexity of $O(M^2)$ for each cross-layer pair (l, l'). The complexity can be reduced to $O(M^{1.14})$ if using the nearest-neighbor-descent algorithm of UMAP (McInnes et al., 2018).

3.1 Manifold Learning and Generation

Manifold Learning. Table 2 compares performances of the five NLDR methods where **bold** numbers are the best results, and underline the second best. The ELIS encoder has overall the best performance. Fig. 1 visualizes some representative results, and more results are given in Table. A2, Fig. A2 - Fig. A4 in Appendix A.3.

Next, we delve into embedding details of of the Coil20 objects resulting from the ELIS encoder (ELIS-Enc) and UMAP in Fig. 2. First, all the object embeddings in the ELIS-Enc result form closed loops for the 360 degree rotations (refer to Fig. A5 for the embedding-object correspondence). Second, the quality of the ELIS-derived embeddings enables us to infer some symmetries of the objects in the 3D space. Four types of such symmetries are explored and discussed in Appendix A.3. The UMAP result, in contrast, does not possess such quality.

Table 2: Comparison in performance metrics for four datasets

		ELIS-Enc	LIS-Enc	UMAP	t-SNE	TopoAE	MLLE
Swiss Roll	Cont	**1.0000**	**1.0000**	0.9962	0.9969	0.9716	0.9956
	Trust	**1.0000**	**1.0000**	0.9983	0.9993	0.9809	0.9948
SpheresB	Cont	**0.9276**	0.9255	0.9109	0.9155	0.9245	0.8943
	ACC(SVM)	0.9273	0.9100	0.9100	0.8478	**0.9581**	0.0965
	ACC(NN)	**0.9991**	0.9969	0.8469	0.9365	0.9949	0.8265
	AUC	0.9711	0.9318	0.9570	0.9570	**0.9870**	0.9459
Coil20	Cont	0.9956	**0.9973**	0.9962	0.9927	0.9901	0.9395
	ACC(SVM)	**0.8941**	0.8301	0.8472	0.8014	0.7078	0.1556
	NNACC	0.9965	0.9354	0.8917	**0.9965**	0.8160	0.6410
	AUC	0.9770	0.9537	**0.9842**	0.9582	0.8916	0.8824
MNIST	Cont	0.9639	**0.9749**	0.9646	0.9630	0.9618	0.9183
	ACC(SVM)	**0.9699**	0.7468	0.9690	0.9525	0.7450	0.1100
	ACC(NN)	**0.9568**	0.7035	0.9528	0.9567	0.7773	0.7423
	AUC	**0.9725**	0.8779	0.9691	0.9314	0.8000	0.8575

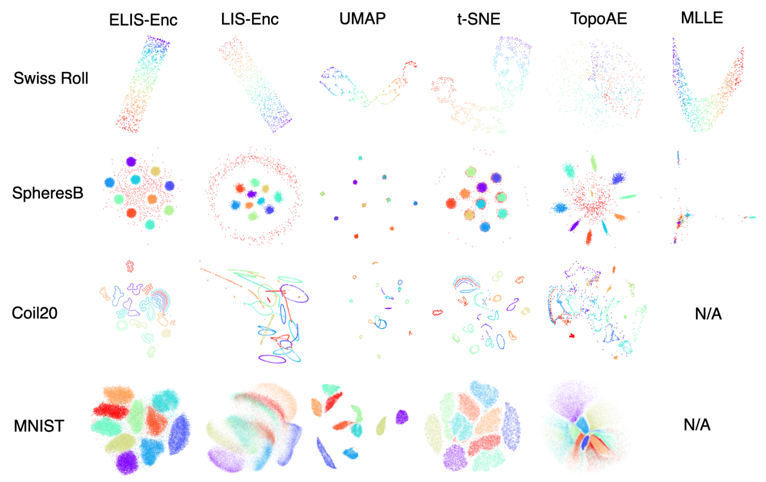

Figure 1: Comparison of visualization results for four datasets

Manifold Data Generation. Fig. 3 compares images generated from interpolated points between two nearest neighbors on the embedding using three autoencoders in comparison. The images generated by the ELIS-AE have clear boundaries and look sharper than those produced by the other two autoencoders. Results for several other objects are shown in Fig. A6 in Appendix A.4.

3.2 Ablation Study

Cross-layer ELIS constraint. The cross-layer ELIS constraint is weighted by $\alpha(l, l')$ as in Equ. (11). Four weight schemes (1) Head-Tail, (2) Head-Mids, (3) Mids-Tail and (4) Head-Mids + Mids-Tail are designed for an L-layer encoder, as described in details in Appendix A.5. The results are compared in Table. A3 and Fig. A7. Overall, the "Head-Mids + Mids-Tail" scheme, which imposes the most extensive cross-layer ELIS constraints and also needs more computation, achieves the best results. This justifies the use of the proposed ELIS method for performance improvements.

Effect of final ν value. The final ν value has significant influence on the within-manifold and between-manifold scatters. Fig. A8 and Fig. A9 in Appendix A.5 demonstrate the effect of varying ν in the input space and varying ν in the latent space on the results, respectively.

Continuation in ν. Continuation in hyperparameter ν in latent space is used during the ELIS encoder learning process. The algorithm starts with a small ν in latent space to include more global

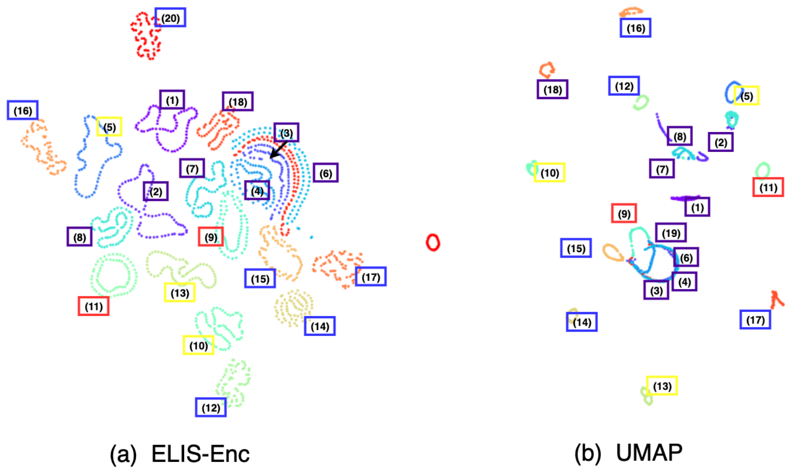

Figure 2: 2D embeddings of Coil20: ELIS encoder vs UMAP, and four manifold groups in terms of symmetry.

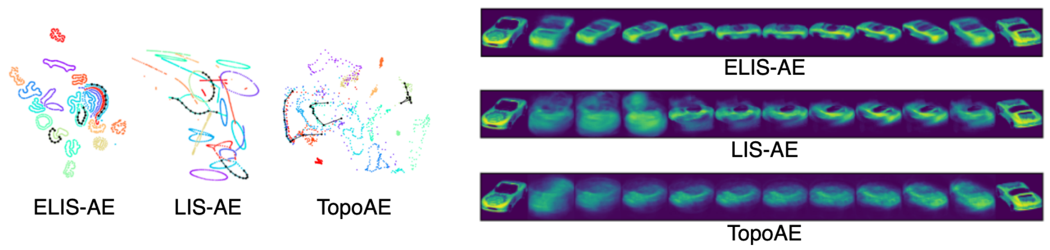

Figure 3: Comparison in manifold data generation.

information. Then it gradually increases the value to focus more locally. The continuation strategy results in significant better solutions as shown in Fig. A10 in Appendix A.5.

The major merits of ELIS. Finally, we summarize the comparative results of the experiments in the following table.

	ELIS	LIS	UMAP	t-SNE	MLLE	TopoAE
Succeed in unfolding toy data	Yes	Yes	No	No	Yes	No
Perfect manifold structures on Coil	Yes	No	Maybe	No	No	No
High Accuracy	Most	No	Some	Some	No	No
Good Reconstruction Quality	Yes	Maybe	N/A	N/A	No	No

4 Conclusion

The proposed ELIS method preserves the nonlinear similarity metric locally across layers of deep neural networks by optimizing two-way divergence loss. It effectively tackles difficulties in deep manifold computing and visualization with the local geometry-preserving property. Empirical results, comparisons, and ablation study demonstrate that ELIS is not only superior to UMAP and t-SNE for NLDR and visualization but also better than other leading manifold and autoencoder learning algorithms for NLDR and manifold data reconstruction and generation. Future work includes the following: (1) extending the unsupervised version of MLDL to self-supervise, semi-supervised and supervised tasks; (2) further formulating MLDL so that cross-layer link hyperparameters α become part of learnable hyperparameters.

REFERENCES

Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel W H Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W Newell. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, 37:38–44, 2019.

D. Bhandari and N. R. Pal. Some new information measures for fuzzy sets. *Information Sciences*, 67: 209–228, 1993.

Michael Bronstein. Latent graph neural networks: Manifold learning 2.0? *Towards Data Science*, Sep 2020. URL `https://towardsdatascience.com`.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

Lisha Chen and Andreas Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104(485): 209–219, 2009.

David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

Michael W. Dorrity, Lauren M. Saunders, Christine Queitsch, Stanley Fields, and Cole Trapnell. Dimensionality reduction by UMAP to visualize physical and genetic interactions. *Nature Communications*, 11(1), December 2020.

Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of American Mathematical Society*, 29(4):983–1049, 2016.

Michael Gashler, Dan Ventura, and Tony Martinez. Iterative non-linear dimensionality reduction with manifold sculpting. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (eds.), *Advances in Neural Information Processing Systems 20*, pp. 513–520, 2008.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Yann LeCun. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges, 2013.

Stan Z Li, Zelin Zang, and Lirong Wu. Markov-lipschitz deep learning. *arXiv preprint arXiv:2006.08256*, 2020.

Laurens van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014.

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

James McQueen, Marina Meila, and Dominique Joncas. Nearly isometric embedding by relaxation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2631–2639, 2016.

Partha Niyogi Mikhail Belkin. Laplacian eigenmaps for dimensionality reduction and data representation. *Technical Report, University of Chicago*, January 2002.

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.

Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-100), 1996.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

Lawrence K. Saul. A tractable latent variable model for nonlinear dimensionality reduction. *Proceedings of the National Academy of Sciences*, 117:425–432, June 2020.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

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

Larry Wasserman. Topological Data Analysis. *Annual Review of Statistics and Its Application*, 5(1): (arXiv:1609.08227), 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zhenyue Zhang and Jing Wang. MLLE: Modified locally linear embedding using multiple weights. In *Advances in Neural Information Processing Systems*, pp. 1593–1600, 2007.

APPENDIX

A.1 THE MLDL FRAMEWORK AND ELIS

Markov-Lipschitz deep learning (MLDL) framework. The MLDL framework is illustrated in Fig.A1 (from Li et al. (2020)). The ML-AutoEncoder (of LIS or ELIS type) transforms the input X to an embedding $X^{(L)}$ at layer L (the latent layer) using the ML-Encoder, and then reconstruct $\hat{X}$ using the ML-Decoder. Whereas a standard neural network consists of a cascade of transformations $\phi^{(l)}$ (blue arrows), an MLDL network imposes the constraint between **any** two layers as appropriate (shown in orange arcs and dashed lines) in the form of cross-layer loss functions weighted by $\alpha^{(l,l')}$. This encourages $\phi^{(l)}$ to become well-behaved local homeomorphisms. The latent features $X^{(L)}$ extracted by the learned ML-Encoder can be used for downstream tasks such as visualization and classification as well as manifold data generation using the learned ML-Decoder.

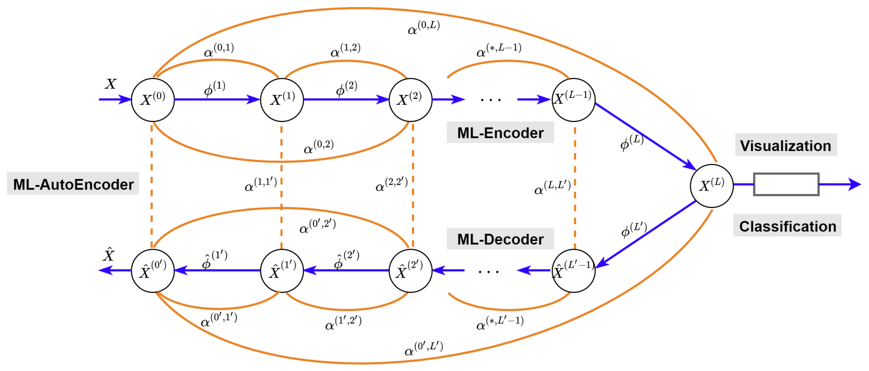

Figure A1: Illustration of Markov-Lipschitz deep learning (MLDL) framework using an ML-AutoEncoder (best viewed in color).

The pseudo-codes for the ELIS encoder and the ELIS autoencoder, related hyperparameter, and a parameter continuation method are described below.

Algorithm 1: ELIS Encoder

```
Input : Data:$X^{(0)}$, learning rate $lr$, epochs $E$, number of encoder layers $L$, Weight
 hyperparameter $\alpha$, $\nu_{List}$, $Q$,
Calculate $d_{i|j}^{(0)}$ with (4)
Calculate $\sigma_i^{(0)}$ with (7)
Calculate $u_{ij}^{(0)}$ with (8)
Initialize the neural network $\{\Phi_{Enc}^{(1)}(\cdot|W_{Enc}^{(1)}), \Phi_{Enc}^{(2)}(\cdot|W_{Enc}^{(2)}), \cdots, \Phi_{Enc}^{(L)}(\cdot|W_{Enc}^{(L)})\}$
while $i = 0$; $i < E$; $i$++ do
    $\nu \longleftarrow \nu_{List}[i]$
    while $l = 1$; $l <= L$; $l$++ do
        Calculate $l$ layer's embedding $X^{(l)} \longleftarrow \Phi_{Enc}^{(l)}(X^{(l-1)}|W_{Enc}^{(l)})$
        Calculate $u_{ij}^{(l)}$ with (5) and (8)
    end
    while $l' = l$; $l' <= L$; $l'$++ do
        Calculate the ELIS losses between layer $l$ and lyaer $l'$, $\mathcal{L}_{Enc}^{(l,l')}$ with (10)
    end
    Update parameters: $W \longleftarrow W - lr \cdot \sum_{l=1}^{L} \sum_{l'=l}^{L} \alpha^{(l,l')} \frac{\partial \mathcal{L}_{Enc}^{(l,l')}}{\partial W}$
end
```

Algorithm 2: ELIS AutoEncoder

```
Input : Data:$X^{(0)}$, learning rate $lr$, epochs $E$, number of encoder layers $L$, Weight
  hyperparameter $\alpha$ $\gamma$, $\nu_{List}$, $Q$
Calculate $d^{(0)}_{i|j}$ with (4)
Calculate $\sigma^{(0)}_{i}$ with (7)
Calculate $u^{(0)}_{ij}$ with (8)
Initialize the neural network $\{\Phi^{(1)}_{Enc}(\ \cdot\ |W^{(1)}_{Enc}), \Phi^{(2)}_{Enc}(\ \cdot\ |W^{(2)}_{Enc}), \cdots, \Phi^{(L)}_{Enc}(\ \cdot\ |W^{(L)}_{Enc}),$
  $\Phi^{(1)}_{Dec}(\ \cdot\ |W^{(1)}_{Dec}), \Phi^{(2)}_{Dec}(\ \cdot\ |W^{(2)}_{Dec}), \cdots, \Phi^{(L)}_{Dec}(\ \cdot\ |W^{(L)}_{Dec})\}$
while $i = 0$; $i < E$; $i$++ do
    $\nu \longleftarrow \nu_{List}[i]$
    while $l = 1$; $l \leq 2L$; $l$++ do
        if $l \leq L$ then
            Calculate $l$ layer's encoder embedding $X^{(l)} \longleftarrow \Phi^{(l)}_{Enc}(X^{(l-1)}|W^{(l)}_{Enc})$;
        else
            Calculate $l$ layer's decoder embedding $X^{(l)} \longleftarrow \Phi^{(l)}_{Dec}(X^{(l-1)}|W^{(l)}_{Dec})$;
        end
        Calculate $u^{(l)}_{ij}$ with (5) and (8)
    end
    while $l' = l$; $l' <= L$; $l'$++ do
        Calculate the ELIS losses between layer $l$ and lyaer $l'$, $\mathcal{L}^{(l,l')}_{ELIS}$ with (10)
    end
    Calculate the reconstruction loss between layer $l$ and lyaer $2L - l$, $\mathcal{L}^{(l,2L-l)}_{Rec}$ with (13)
    Update parameters: $W \longleftarrow W - lr(\sum_{l=1}^{L}\sum_{l'=l}^{L}\alpha^{(l,l')}\frac{\partial\mathcal{L}^{(l,l')}_{Enc}}{\partial W} + \sum_{l=1}^{L}\gamma^{(l,2L-l)}\frac{\partial\mathcal{L}^{(l,2L-l)}_{Rec}}{\partial W})$
end
```

Hyperparameters. Table. A1 summarizes the ELIS hyperparameter setting for different datasets. Other hyperparameters are set the same for all datasets: learning rate $lr = 0.01$ and number of epochs $E = 5000$. The LeakyReLU is used as the activation function.

Table A1: Hyperparameters of ELIS for different datasets

Dataset	Point	Network Structure (Number of parameters)	Q in Equ. (7)	Batchsize
Swiss Roll	800	3, 500, 500, 2 (0.252M)	10	800
S-Curve	800	3, 500, 500, 2 (0.252M)	10	800
Servered Sphere	800	3, 500, 500, 2 (0.252M)	10	800
SpheresA	10000	101, 500, 500, 2 (0.301M)	10	10000
SpheresB	5500	101, 500, 500, 2 (0.301M)	10	5500
Coli20	1440	16384, 500, 500, 2 (8.443M)	10	1440
Coli100	7200	49152, 1000, 500, 250,2 (24.82M)	10	2400
MNIST	60000	784, 1000, 500, 300, 2 (1.434M)	15	4000
Fashion-MNIST	60000	784, 1000, 500, 2 (1.285M)	10	4000

Continuation in $\nu^{(l')}$. In the training process, the parameter $\nu^{(l')}$) in computing sample similarities for the latent layer is graduated from a small number to a large number, e.g. $\nu^{(l')} : 0.01 \rightarrow 100$ (see Equ. (5)), though fixed at a large value, e.g. $\nu^{(l')} = 100$ for the input layer. Empirically, the continuation helps training converge to a good solution; the reasons behind are to be explained in a future work.

A.2 Definitions of performance metrics

1. Cont (Continuity) is asymmetric to **Trust** (from space $X^{(l')}$ to space $X^{(l)}$):

$$Cont = \frac{1}{k_2 - k_1 + 1} \sum_{k=k_1}^{k_2} \left\{ 1 - \frac{2}{Mk(2M - 3k - 1)} \sum_{i=1}^{M} \sum_{j \in \mathcal{N}_{i,k}^{(l)}, j \notin \mathcal{N}_{i,k}^{(l')}} (r_{i,j}^{(l')} - k) \right\}$$

where $r_{i,j}^{(l')}$ is the rank of $x_j^{(l')}$ in the k-NN of $x_i^{(l')}$. M is the size of dataset. $\mathcal{N}_{i,k}^{(l')}$ is the set of indices to the k-NN of $x_i^{(l')}$. k_1 and k_2 are the lower and upper bounds of the k-NN. For sphereA and sphereB, we focus more on global performance, so set $k_1 = [M/14], k_2 = [M/7]$. For other datasets, we set $k_1 = 5$, $k_2 = 10$.

2. Trust (Trustworthiness) measures how well the k nearest neighbors of a point are preserved when going from space $X^{(l)}$ to space $X^{(l')}$:

$$Trust = \frac{1}{k_2 - k_1 + 1} \sum_{k=k_1}^{k_2} \left\{ 1 - \frac{2}{Mk(2M - 3k - 1)} \sum_{i=1}^{M} \sum_{j \in \mathcal{N}_{i,k}^{(l')}, j \notin \mathcal{N}_{i,k}^{(l)}} (r_{i,j}^{(l)} - k) \right\}$$

where $r_{i,j}^{(l)}$ is the rank of $x_j^{(l)}$ in the k-NN of $x_i^{(l)}$.

3. ACC (svm)

The ACC (svm) is calculated as follows.
(1) Compute nonlinear dimensionality reduction methods to obtain 2-dimensional embeddings.
(2) Partition the data by 5-fold cross-validation.
(3) For each fold, train the linear kernel SVM classifier using the training set and test it in the test set.
(4) Calculate the mean value of the classification accuracy.

4. ACC (NN) is defined as follows:

$$ACC(NN) = \frac{\sum_i^M \pi \left[Y_i = Y_{\mathcal{N}_{i,1}^{(L)}} \right]}{M}$$

$\pi[\cdot]$ is the exponential function. Y_i is the label of sample i, $Y_{\mathcal{N}_{i,1}^{(L)}}$ is the label of sample $\mathcal{N}_{i,1}^{(L)}$, where $N_{i,1}^{(L)}$ is the nearest neighbor point of node i in layer L.

5. AUC is defined as follows:

$$AUC(f) = \frac{\sum_{p_0 \in \mathcal{P}^0} \sum_{p_1 \in \mathcal{P}^1} \pi \left[p_0 > p_1 \right]}{|\mathcal{P}^0| \cdot |\mathcal{P}^1|}$$

$$\mathcal{P}^0 = \left\{ \frac{d_{ij}^{(L)} - \min d_{ij}^{(L)}}{\max d_{ij}^{(L)} - \min d_{ij}^{(L)}} | i, j \in \{1, 2, 3 \cdots, M\}, Y_i = Y_j \right\}$$

$$\mathcal{P}^1 = \left\{ \frac{d_{ij}^{(L)} - \min d_{ij}^{(L)}}{\max d_{ij}^{(L)} - \min d_{ij}^{(L)}} | i, j \in \{1, 2, 3 \cdots, M\}, Y_i \neq Y_j \right\}$$

Where $d_{ij}^{(L)}$ is the distance in layer L. $\mathcal{P}^0$ is the set of positive sample pair, $\mathcal{P}^1$ is the set of negative sample pair.

A.3 Manifold Learning and NLDR

Manifold Learning Results. This subsection shows more results of manifold learning and NLDR obtained by using the ELIS-Enc and the ELIS-AE, in comparison with the other methods, on training and testing datasets. Some typical embedding results of manifold learning using the three autoencoder methods are visualized in Fig. A2, where t-SNE and UMAP are not included because these non-transformational methods are unable to generalize to test datasets. LIS-Enc and TopoAE learned poor

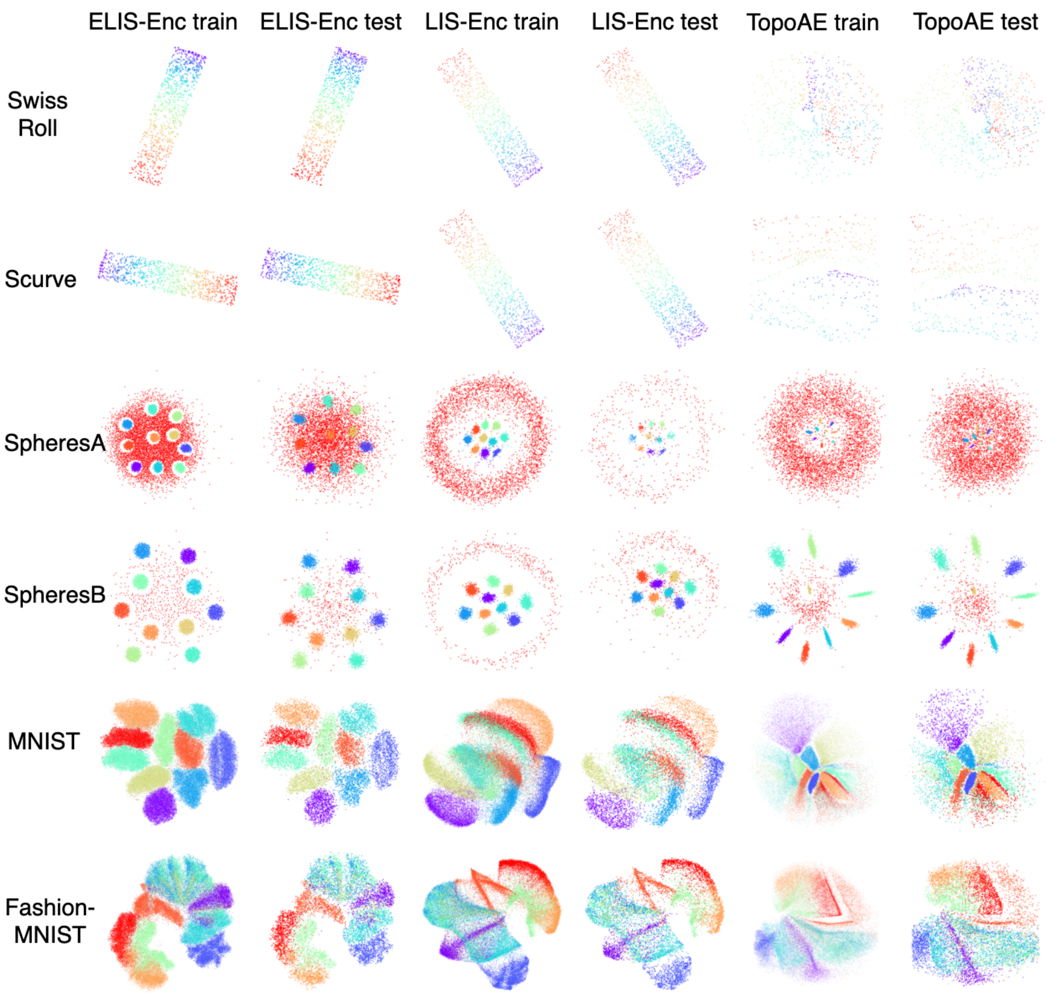

Figure A2: Comparison of visualization results of autoencoders on training and testing sets

results on the training set, so it did not work well on the test set either. ELIS-AE, as a autoencoder-based mehtod, have a huge advantage in terms of generalization performance, because it can handle the test data. And it is very easy to apply to specific tasks such as classification, regression and clustering.

Table. A2 compares performance metrics on 8 datasets, where the ACC(SVM), ACC(NN) and AUC are abscent for the SwissRoll and SeveredSphere because these datasets have no class label.

Fig. A3 and Fig. A4 shows the visualization resules of the toy and real-world datasets on the training datasets. For Swiss roll, Servered Sphere, and S-Curve, ELIS-Enc, LIS-Enc and MLLE all maintained the topology of the original data, however, MLLE method did not hold the relative Euclidean distance (The resulting embedding is square instead of rectangular). For SpheresA, ELIS-Enc, LIS-Enc, and TopoAE show the "big sphere enclosing 10 small spheres" in 2D embedding, but for SpheresB only

Table A2: Comparison in performance metrics with 5 difference methods in eight datasets

		ELIS-Enc	LIS-Enc	UMAP	t-SNE	TopoAE	MLLE
Swiss Roll	Cont	**1.0000**	**1.0000**	0.9962	0.9969	0.9716	0.9956
	Trust	**1.0000**	**1.0000**	0.9983	0.9993	0.9809	0.9948
SeveredSphere	Cont	**0.9997**	0.9932	0.9967	0.9985	0.9854	0.9958
	Trust	**0.9997**	0.9755	0.9989	0.9995	0.9891	0.9836
SpheresA	Cont	0.7850	0.7892	0.7147	0.7548	**0.8064**	0.7272
	ACC(SVM)	0.5213	0.5000	**0.5550**	0.4992	0.4982	0.5000
	ACC(NN)	**0.9985**	0.9912	0.5406	0.7837	0.9944	0.5205
	AUC	0.5698	0.3362	0.5816	0.5603	0.3328	**0.5961**
SpheresB	Cont	**0.9242**	0.9255	0.9109	0.9155	0.9245	0.8943
	ACC(SVM)	0.9558	0.9100	0.9100	0.8478	**0.9581**	0.0965
	ACC(NN)	**0.9987**	0.9969	0.8469	0.9365	0.9949	0.8265
	AUC	0.9780	0.9318	0.9570	0.9570	**0.9870**	0.9459
Coil20	Cont	0.9956	**0.9973**	0.9962	0.9927	0.9901	0.9395
	ACC(SVM)	**0.8941**	0.8301	0.8472	0.8014	0.7078	0.1556
	NNACC	0.9965	0.9354	0.8917	**0.9965**	0.8160	0.6410
	AUC	0.9780	0.9537	**0.9842**	0.9582	0.8916	0.8824
Coil100	Cont	0.9936	**0.9967**	0.9955	0.9950	0.9903	0.7898
	ACC(SVM)	**0.9372**	0.7319	0.8299	0.8278	0.5540	0.0363
	ACC(NN)	**0.9976**	0.8163	0.9232	0.9951	0.4797	0.3350
	AUC	**0.9770**	0.9667	0.9819	0.9759	0.8735	0.7322
MNIST	Cont	0.9639	**0.9749**	0.9646	0.9630	0.9618	0.9183
	ACC(SVM)	**0.9699**	0.7468	0.9690	0.9525	0.7450	0.1100
	ACC(NN)	**0.9568**	0.7035	0.9528	0.9567	0.7773	0.7423
	AUC	**0.9725**	0.8779	0.9691	0.9314	0.8000	0.8575
Fashion -MNIST	Cont	0.9848	**0.9901**	0.9836	0.9777	0.9864	0.9298
	ACC(SVM)	**0.7125**	0.6908	0.7030	0.5518	0.6067	0.1058
	ACC(NN)	0.7092	0.6427	0.7253	**0.7787**	0.5718	0.6145
	AUC	**0.9121**	0.8843	0.9165	0.8256	0.8310	0.7908

ELIS-Enc and LIS-Enc shows the "enclosing" phenomenon. For Coil20 and Coil100, ELIS-Enc, UMAP and TSNE can produce non-intersecting embeddings. However, the ELIS-Enc results are distinguishable and do not cut any of the manifolds. For MNIST and Fashion-MNIST, Both the UMAP and ELIS-Enc methods output the good embeding, But in terms of performance metrics, ELIS-Enc has sufficient advantages.

Symmetry of the objects and ELIS-Enc's embedding in Coil20. For Coil20, information about the symmetry of the objects in the picture can be obtained by analyzing the embedding generated by ELIS-Enc. Details of the ELIS-Enc embedding of the Coil20 are shown in Fig. A5.

We divided the Coil20's manifolds into four patterns based on the symmetry of the objects in the image and the shape of the manifold.

(1) Objects that are single plane mirror symmetric have elongated ellipse embedding shapes; For objects with single plane mirror symmetry, an angle can be found from which an image taken by rotating to the left is approximately equal to an image taken by rotating to the right. The corresponding two-dimensional manifolds are therefore elongated ellipse (The endpoints of the two long axes of the ellipse correspond to the two images obtained by taking pictures along the plane of symmetry.).

(2) Objects that are rotational symmetric have round embedding shapes; For rotational symmetric objects, the resulting pictures are always very similar no matter what angle they are taken from, so that the resulting two-dimensional manifold is squeezed inward into a circle.

(3) Objects that are double vertical mirror symmetric and have nested double ring embeddings; For objects with double vertical mirror symmetry, every 180 degrees of rotation, the resulting image reappears (the reappeared image is very similar to the one from 180 degrees ago, and

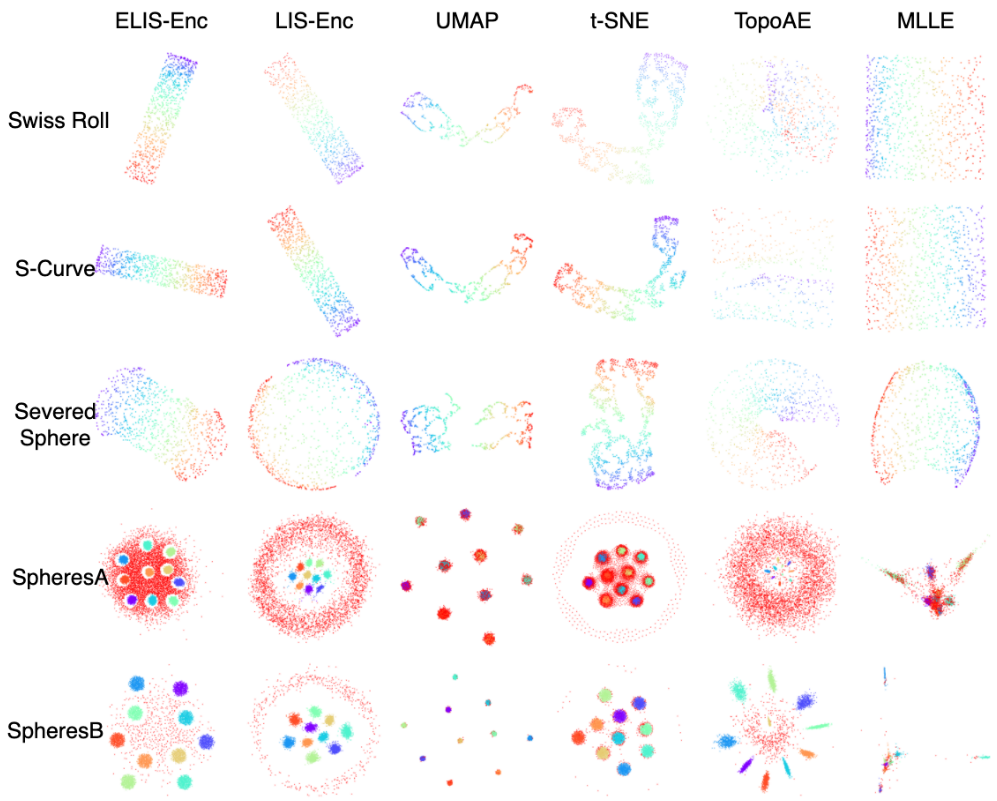

Figure A3: Comparison of visualization results for toy dataset on training set

is very close in two-dimensional space), thus the resulting manifold consists of two nested rings.

(4) Object's symmetry is not evident.

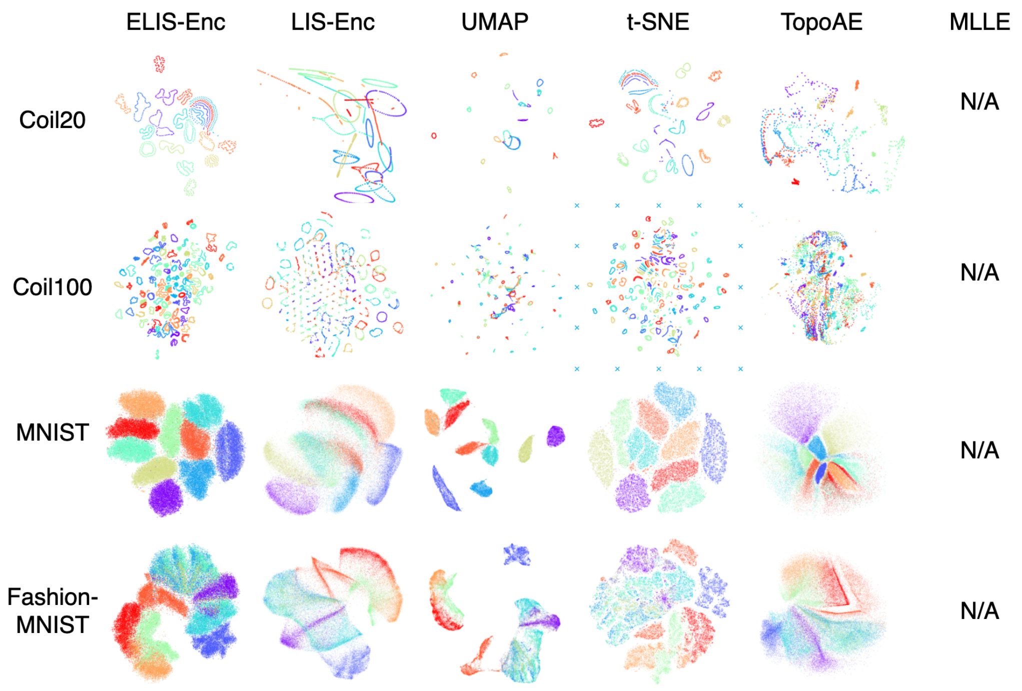

Figure A4: Comparison of visualization results for real-world dataset on training set

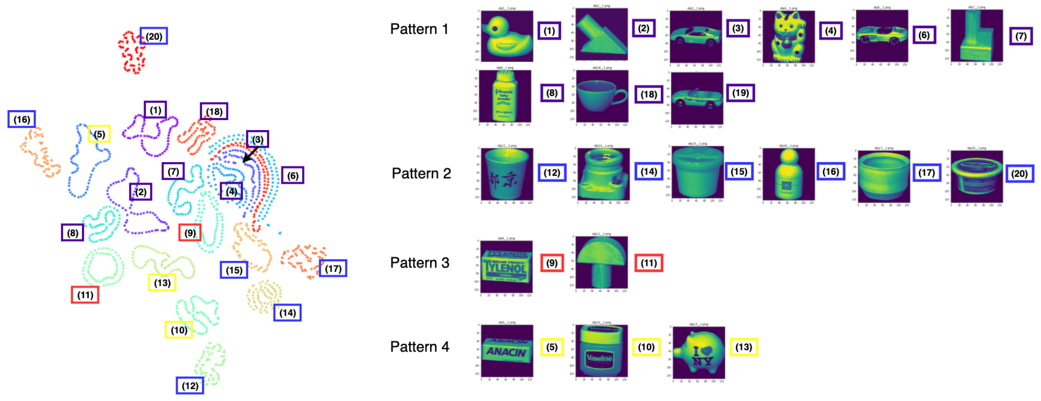

Figure A5: Details of the ELIS-Enc's embedding of the Coil20 and four manifold patterns

A.4 Manifold data generation

The manifold generation task generates a complete manifold structure from finite manifold samples. In this experiment,the test steps are as follows:

(1) Training a network (includes encoder and decoder) that generating 2-dimensional embedding;

(2) Performing linear interpolation in the embeddings;

(3) Mapping the interpolation result back to the data space via the decoder.

Generation results for comparison with the TopoAE and LIS-AE are shown in Fig. A6. The same network structure was used in the experiments.

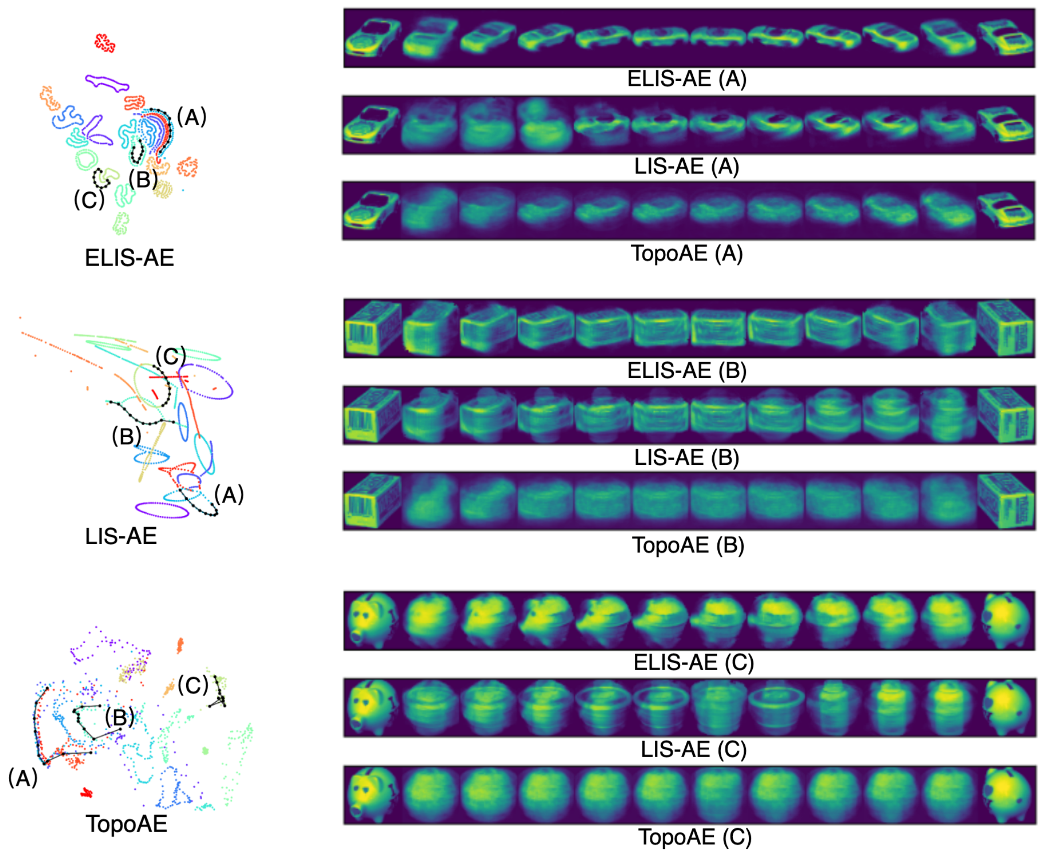

Figure A6: Comparison in visualization with LIS-AE and TopoAE in manifold generation. The left side is three embedding result, and black point in manifolds is the location of the interpolation. The right side is the interpolation results. there are 12 images in the right of the figure, the leftmost and the rightmost images are the original images, and ten images in the middle are the generation results with geodesic distance.

ELIS-AE has an advantage over the other two methods. Both LIS-AE and TopoAE methods do not learn a satisfactory embedding, so the interpolation results are poor. The embedding in LIS-AE has overlapping manifolds, so it generates images belonging to other popular methods (e.g. manifold A). The TopoAE's embedding is messy, so the decoder reconstructs fuzzy images.

A.5 Ablation study

Cross-layer ELIS constraint. The effect of the ELIS-Enc constraint is determined by the weights $\alpha(l, l')$ as in Equ. (11). We set the weights $\alpha^{(l,l')}$ in either of four schemes (where Head, Tail and Mids are used to denote input layer, latent layer and intermediate layers) for an L-layer encoder:

(1) Head-Tail: weight $\alpha^{(0,L)} = 1$ (the constraint is imposed between the input layer and the latent layer);

(2) Head-Mids: weights $\alpha^{(0,l)} = 1/L$ where $l \in \{1, 2, \cdots, L\}$ (the constraints are imposed between the input layer and each of intermediate layers);

(3) Mids-Tail: weights $\alpha^{(l,L)} = 1/L$ where $l \in \{1, 2, \cdots, L\}$ (the constraints are imposed between the latent layer and each of intermediate layers);

(4) Head-Mids + Mids-Tail: weights $\alpha^{(0,l)} = \alpha^{(l,L)} = 1/2L$ where $l \in \{1, 2, \cdots, L\}$ (combination of Head-Mids and Mids-Tail).

In this ablation study, a 10-layer neural network is used and the width of the network is determined depending on the dataset. (Swiss Roll:[3, 50, 45, 35, 30, 25, 20, 15, 10, 5, 2], SpheresA:[101, 50, 45, 35, 30, 25, 20, 15, 10, 5, 2], SpheresB:[101, 500, 400, 300, 300, 200, 200, 100, 100, 2], COIL20:[16384, 50, 45, 35, 30, 25, 20, 15, 10, 5, 2])

The evaluation metrics for four different cross-layer schemes are presented in Table. A3. The results of different cross-layer schemes are shown in Fig. A7.

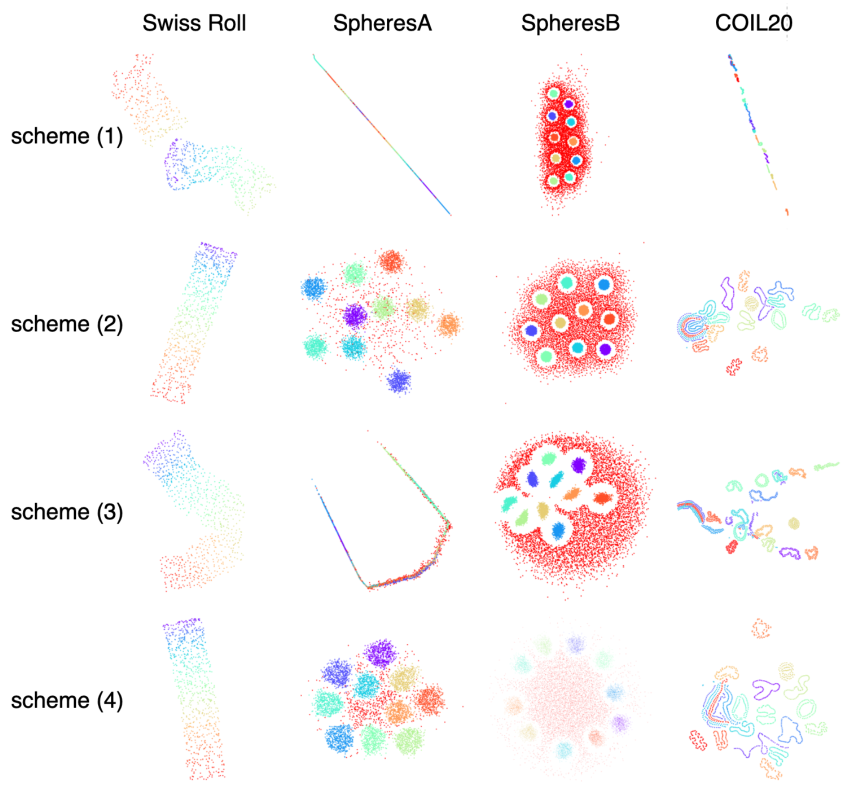

Figure A7: Comparison in visualisation of four different cross-layer schemes.

The visualization results and metrics show that the cross-layer scheme (4) has better results in a 10-layer network. The network is very difficult to train if the ELIS loss acts only on the first and last layers (cross-layer scheme (1)). The network will be easier to train if ELIS losses acts from the first layer and all intermediate layers (cross-layer scheme (2)). The ELIS losses in the middle and last layers (cross-layer scheme (3)) does not improve the performance of the embedding if used alone.

Table A3: Comparison in performance metrics of four different cross-layer schemes.

		Cont	Trust	ACC(SVM)	ACC(NN)	AUC
Swiss Roll	(1)	-	-	-	-	-
	(2)	**0.9999**	**0.9999**	-	-	-
	(3)	-	-	-	-	-
	(4)	**0.9999**	**0.9999**	-	-	-
SpheresA	(1)	-	-	-	-	-
	(2)	**0.9402**	0.8832	0.9149	0.9478	0.9696
	(3)	-	-	-	-	-
	(4)	0.9376	**0.8858**	**0.9529**	**0.9784**	**0.9721**
SpheresB	(1)	**0.9111**	0.6373	0.5225	**1.0000**	0.5486
	(2)	0.9087	0.6341	0.5145	**1.0000**	0.5489
	(3)	0.8520	0.6299	0.5388	**1.0000**	0.4474
	(4)	0.8167	**0.6432**	**0.8740**	0.9936	**0.7461**
Coil20	(1)	-	-	-	-	-
	(2)	**0.9955**	0.9852	0.8454	0.9792	0.9721
	(3)	0.9904	0.9876	0.8459	0.9847	0.9524
	(4)	0.9947	**0.9901**	**0.8867**	**0.9986**	**0.9735**

However, if used in conjunction with the cross-layer scheme (4), it will improve the metric of the resulting latent space.

Effect of ν value. Fig. A8 and Fig. A9 show the effect of data space hyperparameter and latent space hyperparameter on embedding results.

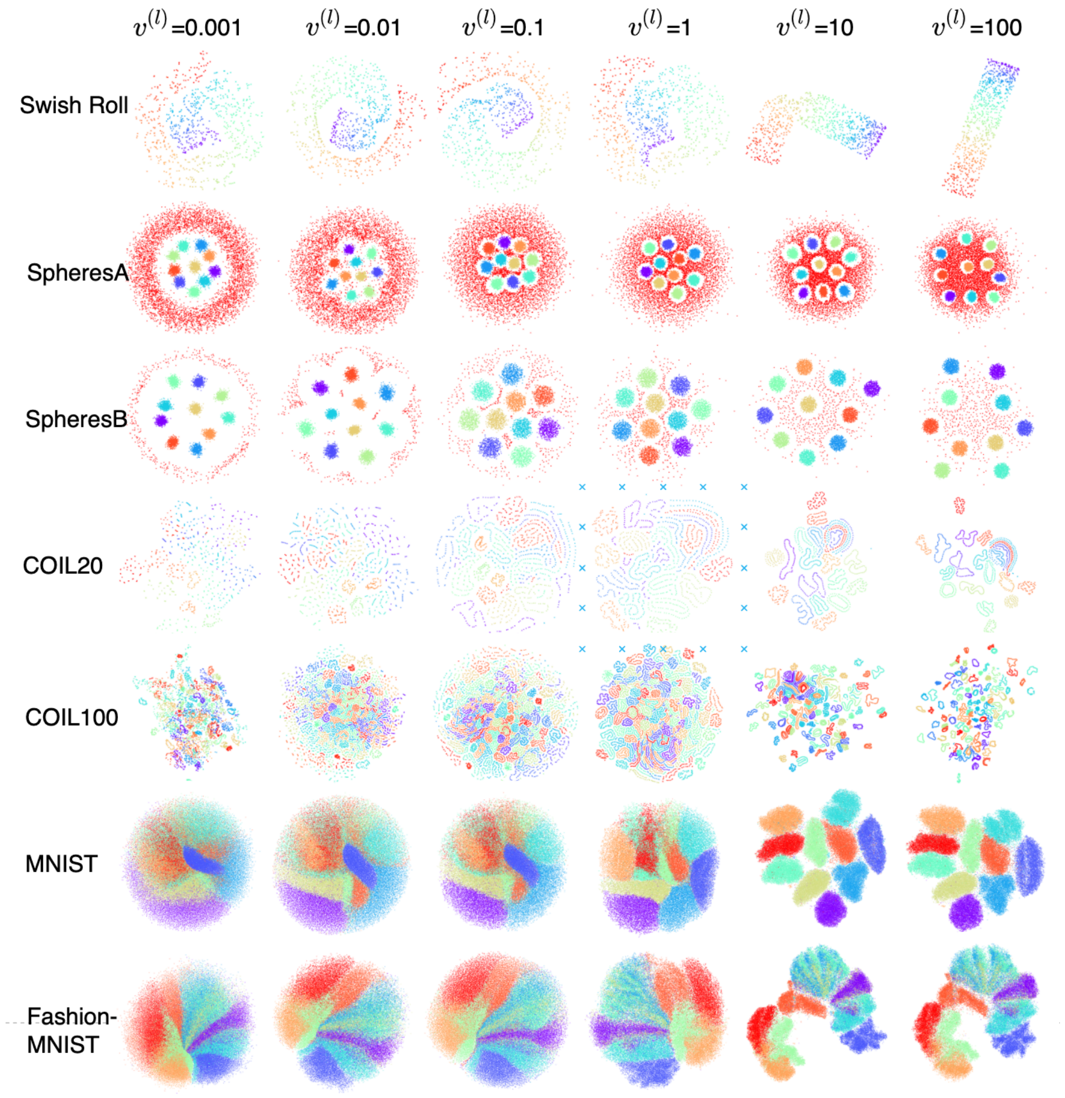

Figure A8: Embedding results with varying ν in input space.

ν in input space controls the range of sensations in data space. if input space's ν is small, the derivative of the probabilistic mapping function of the input layer will be small. The probability will be insensitive to distance in data space. In other words, ELIS-Enc will degradation into LIS-Enc. If more attention is paid to the global information of the input data, raise ν. If more attention is paid to local information about the input data, lower input space's ν. By default, ELIS-Enc does not change this hyperparameter, but defaults to input space's $\nu = 100$.

ν in latent space controls degree of display in latent space (show detailed information or show global information). if latent space's ν is small, ELIS will tend to display global information. If latent space's ν is large, ELIS-Enc will tend to display the local information. Fig. A9 shows, from left to right, a process that goes from showing global information to showing excessive local information.

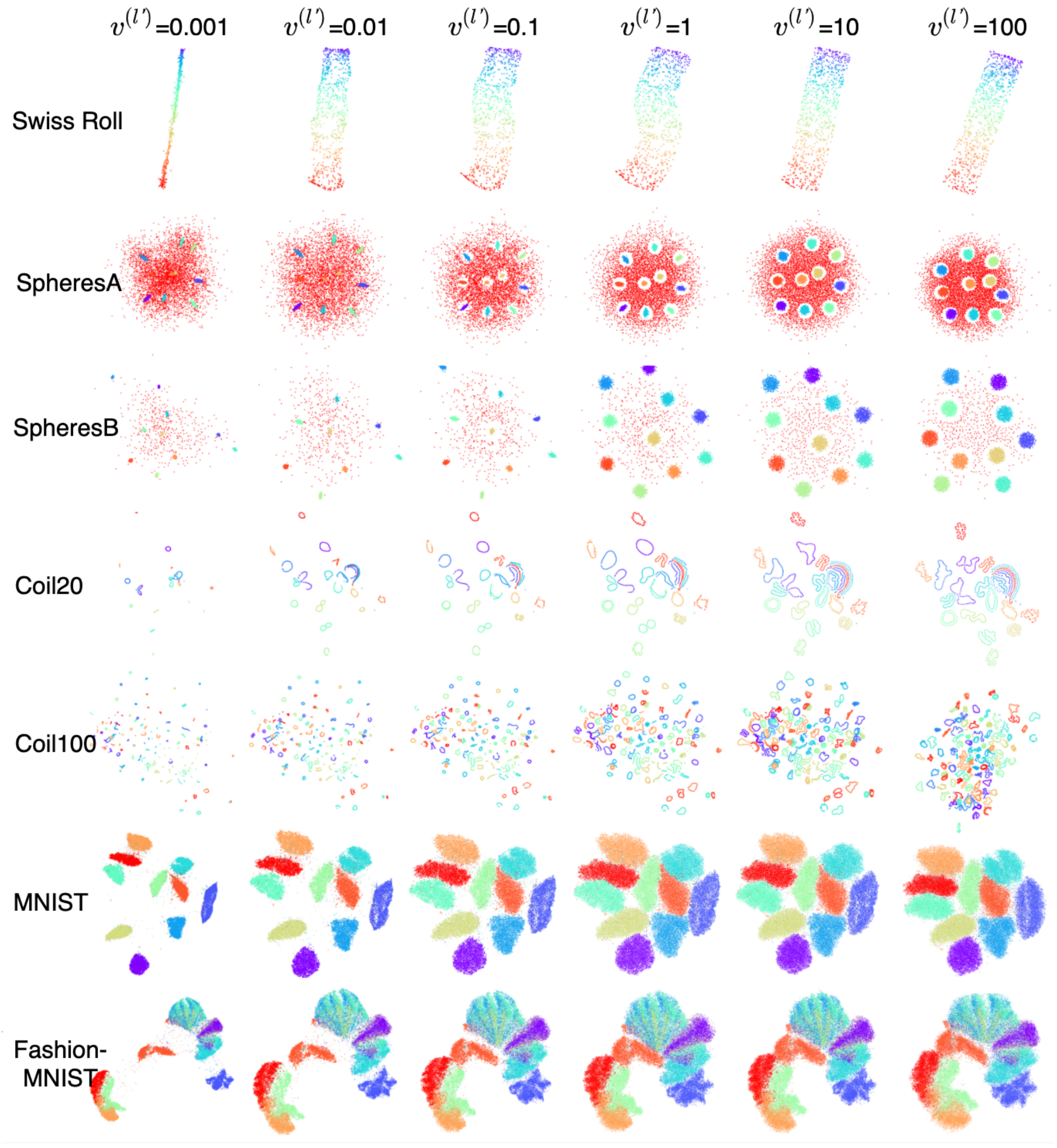

Figure A9: Embedding results with varying ν in latent space.

Continuation strategy. Continuation in hyperparameter latent space's ν is used during the ELIS encoder learning process. The algorithm starts with a small latent space's ν value to include more global information. Then it gradually increases the value to focus more locally. The necessity of parameter continuation is shown in Fig. A10.

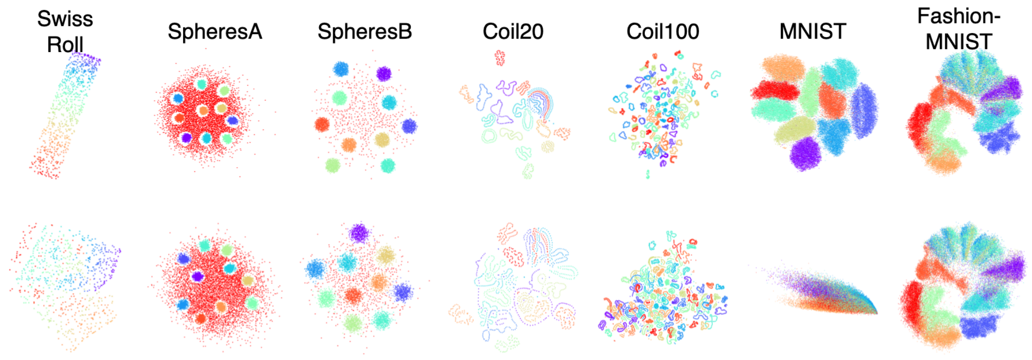

Figure A10: Ablation study of with and without parameter continuation in latent space ν. (The upper row shows results obtained via parameter continuation $\nu = 0.001 \rightarrow \nu = 100$ in latent space, the lower row shows results with a fixed $\nu = 100$)

Experiments prove that the effect of parameter continuation ($\nu = 0.001 \rightarrow \nu = 100$ in latent space) is very obvious. There is a big improvement in Swiss Roll, Coil20 and MNIST.